GE/PAC 4000 SOFTWARE NOTEBOOK

TABLE OF CONTENTS

COMPUTER SOFTWARE

The successful operation of any process system depends on two functions - hardware and software. Hardware is the group of electronic components that acts physically upon information and signals from the process. That portion of software called the system program is the intelligence that instructs the computer hardware to step through the various cycles of the process. Other software packages aid in preparing the system program. General Electric process computer software:

- Facilitates a systematic approach in writing computer programs
- Sufficiently flexible to allow for future changes in user process

Complete software packages for GE/PAC 4000 have been developed and are classified into 3 categories: Language Processors, Standard On-Line Functions, and Program Debugging Aids:

Language Processors

- Translates functions into computer instructions
- Increases speed and accuracy in preparing programs
- Aids in documentation and debugging of a program
- Operate on GE/PAC 4000, GE 412 or GE 225 computer systems

Process Assembler Language

- Enables programmer to code program in systematic and well documented fashion.
- Generates single and double precision constants to a specified scale factor.
- Provides built-in check features which detect and notify programmer of coding errors.

Fortran II Compiler

- Enables powerful algebraic and verbal statements to be written with minimum time and effort.
- Permits experienced Fortran programmers to make easy transition to GE/PAC programming.
- Allows previously written Fortran programs to be adapted to the GE/PAC 4000.

Computer Software (Continued)

Standard On-Line Functions

- Common routines suitable for numerous applications
- Require little or no re-work by user programmer
- Use of pre-checked routines eliminates programming and debugging time

Monitor

- Schedules and gives priority assignment to system functions
- Enables user to have system running on-line in minimum of time
- Provides communication between functions and peripheral equipment

Math Routines

- Includes fixed, floating, single and double precision math routines
- Available from General Electric library to all GE/PAC 4000 users
- GE/PAC 4000 users receive up-to-date abstracts of new routines as they are developed.

Debugging Aids

- Debugging aids proven by field experience provide powerful means of keeping program documentation up to date.
- Debugging aids provide user with loader, dump, memory change, and trace routines.
- On-line debugging aids are plug-in package to Monitor.

STANDARD ON-LINE FUNCTIONS

Standard on-line functions are portions of a total system program that are common from one computer application to another. Because they are used so frequently, optimum execution time and utilization of memory have been stressed in their development. For most applications the use of these functions require little or no re-work by user programmers. Use of these pre-checked routines eliminates programming and debugging time.

MONITOR

MONITOR provides the skeleton of a real-time program by scheduling and giving priority assignment to system functions. By using MONITOR the user is capable of having an on-line program running with minimum of time spent on the program. Additional functions can be added to MONITOR requiring no re-working of the running system.

The routines that make up MONITOR are clear and well defined which makes for easy understanding of the system. The layout of the MONITOR package facilitates the addition of system functions by user programmers. The routines that comprise MONITOR are:

- Time and diagnostic count
- Executive control program
- Save registers routine
- Restore register routine
- Turn off program routine
- Set program delay
- Input-output drive
- Output program consisting of:
    - Output subroutine
    - Decimal floating point routine
    - Decimal fixed point routine
    - Octal conversion routine
    - BCD conversion routine
    - Build driver table routine
    - Binary to decimal conversion subroutine

Standard On-Line Functions

In addition to the above, MONITOR also includes the following two routines for core/drum GE/PAC 4000 computers:

Drum transfer subroutine
Drum transfer drive

The Time and Diagnostic Count routine essentially performs two functions. The routine maintains the value of current time (in seconds) which is used by the Executive Program in determining execution time of the functional programs. The second function of this routine is to detect if certain peripheral devices such as typewriters, punches, etc. fail. This routine will usually turn on a corrective action function which initiates remedial action.

The Executive Control Program (E.C.P.) is the real heart of the MONITOR system as it initiates functional programs according to their scheduled interval and priority. The ECP provides for easy writing and addition of function programs as it performs real-time housekeeping chores such as saving and reloading arithmetic and index registers. The number of functional programs that the ECP can manage is limited only by the computer core or drum size and the length of the individual functions. Most MONITOR systems allow 25 to 50 functional programs.

To accomplish on-line requirements as they occur, the GE/PAC 4000 uses automatic priority interrupt which may interrupt a program at any time. Since the various registers (arithmetic, location counter, index) must be saved for the interrupted program, MONITOR provices the Save Register and Restore Register routines that user interrupt programs can use.

To further reduce housekeeping chores of functional programs using MONITOR, Set Program Delay and Turn Program Off routines are supplied. These routines enable a user function to be initiated at a specified interval or be completely turned off until some exterior influence (such as a demand) re-initiates the function.

The Input/Output program permits user functional programs to communicate with peripheral devices such as typewriters, readers, punches, etc. in an orderly, step-by-step fashion. The Input/Output program determines if the peripheral is available for use by the functional program, reserves the peripheral for the functional program, reads in or puts out data, and makes the peripheral free for other functions upon conclusion of the present function. The Input/Output program eliminates a large

Standard On-Line Functions

amount of editing and bookkeeping that would be required if it were not available. The output portion of the program converts binary floating point to decimal fixed point output, binary fixed point to decimal fixed point output, binary to octal output, binary data to numeric and alphabetic character output. The Input/Output program operates the peripherals through automatic priority interrupt which insures the peripherals to operate at their maximum speed.

For GE/PAC 4000 computers equipped with a magnetic drum memory, MONITOR also provides the transfer linkage for information flowing between core and drum. The user functional program simply uses the Drum Transfer Subroutine to initiate a transfer of information between core and drum. While the transfer is in progress, MONITOR initiates other functions that require action thereby permitting full utilization of the central processor. When the transfer is complete, the ECP will re-initiate the fundamental program that requested the transfer. Completion of a drum/core transfer is detected by automatic priority interrupt which insures that the date involved in the transfer is acted upon expeditiously by the function.

MONITOR Compatible Functions

MONITOR Compatible Functions are those functions that are developed for specific applications which may be used by other computer users. These routines are written in a general form so that they require a minimum of re-working to be suitable for similar other applications. Examples are scanning, alarming, conversion equations, demand print, etc.

Math Routines

Math Routines which are usually used in subroutine form include fixed point, single and double precision math functions. These routines are available from the General Electric Company library to all GE/PAC 4000 computer users. They may be obtained in punched card form, symbolic listing or paper tape where the General Electric Company will assemble the routines to fit the user parameters. Examples of those routines are square root, trigometric, exponential functions, etc.

LANGUAGE PROCESSORS

Language processors are those preparation aids that enable a programmer to translate a function into actual computer instructions. Use of a symbolic language greatly increases the speed and accuracy of preparing a program and also aids in the documentation and debugging of the program. To provide the programmer with the most effectual means of coding a program, the General Electric Company has developed two powerful language processors - Process Assembler Language and GE/PAC Fortran. These language processors are extremely flexible - providing for magnetic tape, paper tape, or punched card input. Additional service to the user has been provided as these processors operate on GE/PAC 4000, GE 412 or GE 225 computer systems. The GE 225 language processor version enables routines to be assembled or compiled at the various General Electric Information Processing Centers throughout the country and overseas.

PROCESS ASSEMBLER LANGUAGE

The Process Assembler Language accepts coded symbolic instructions and translates them into computer instructions. These symbolic instructions are coded by the programmer on a coding sheet from which cards are punched when the coding is completed. The cards then may be read into the computer on which the Assembler Program is operating.

The output from the Assembly Program is a listing from a printer or typewriter of the object program and a paper tape which is used to load the new program into the computer. In addition to this translation, the Assembler Program has built-in check features which detect and notify the programmer of coding errors. The listing provides documentation of the program and is invaluable as a debugging aid.

An On-Line Process Assembler Language program accepts the same symbolic language as above and has the ability to assemble programs while the GE/PAC 4000 is handling process data. Its advantage is that it can be used to assemble or re-assemble programs at the user site, and integrates them into the system with no interruption to process computing.

The listing provides the symbolic instructions as coded and also the octal core locations, instructions and constants of the assembled program. For

Language Processors (Continued)

further aid in debugging, the listing reproduces all comments from the coding sheet. Three columns of numbers represent:

Core location
Relative address
Absolute address

The Assembly Program makes maximum use of the relative addressing feature of the GE/PAC 4000. The Assembly Program assembles the operand value relative to the location of the instruction itself rather than the absolute value. With this feature, it is possible to move the program within memory, thereby adding a new dimension of flexibility to system program organization.

In addition to the computer hardware instructions, the Assembly Program makes maximum use of pseudo instructions for storage assignments, symbol definition, and generation of constants to provide better programmer efficiency. These instructions include block storage reservation, single- and double-word floating constants, etc.

GE/PAC FORTRAN Compiler

To make the writing of new programs easy and efficient as possible, G.E. has created a Fortran Compiler for GE/PAC 4000. This compiler goes a step beyond the Process Assembler Language program in that it enables the programmer to write his program in terms of "statements" which employ familiar language and symbols rather than the symbolic code required by the Process Assembler Language. An example of such a statement might be:

Y = A/B + C - SIN(D + E)

where A, B, C, D and E are symbols which have been defined by the programmer in previous statements. A statement such as this, when presented to the Fortran Compiler, will cause the compiler to automatically generate all the step-by-step machine instructions necessary to perform the calculations called for in the statement. Thus, the programmer is freed from the time-consuming details of step-by-step programming and allowed to concentrate more fully on the problem at hand.

In preparing the Fortran Compiler for the GE/PAC 4000, G.E. has incorporated several special features which facilitate the writing and running of programs in a real-time process control environment.

Language Processors (Continued)

Compatibility With Monitor

The Compiler has been designed so that the programs it produces will have numerous special provisions for operation within the G.E. Monitor system. Thus, new programs may be easily incorporated into existing Monitor systems.

Compatibility With Process Assembler Language

The programmer is free to intermix GE/PAC Fortran statements with Process Assembler Language statements within a single program. This allows the programmer to switch back and forth between the two languages arbitrarily, always free to choose the language in which he can proceed more efficiently. Output from the compiler is in the form of Process Assembler Language Symbolic coding.

Bit Manipulation Capability

Special Fortran statements are available to the programmer through which he may exploit the ability of the GE/PAC 4000 to manipulate individual bits within a word. In this manner, individual bits may be treated as separate variables and may be set, reset, tested, and operated upon with Boolean algebraic expressions.

Drum-Core Transfers

Transfers of information between drum storage and core storage may be implemented through the Fortran Compiler by means of special statements provided for this purpose.

Memory Economy

The Compiler has been designed so that the programs which it produces will require a minimum of memory space. This conservation of memory can mean increased flexibility during future additions of modifications of programs in cases where it is not desired to increase the memory size of the machine.

Floating Point Operation

The Fortran Compiler will accept data in either integer or floating point form, and programs produced by the Compiler may be made to output data in either form.

Statement Repertory

A large repertory of allowed statements, plus a full complement of "library" subroutines, makes for ease and flexibility in programming with the Fortran Compiler.

DEBUGGING AIDS

Field experience has shown that debugging aids provide a powerful means of updating program documentation which leads to a successful program. To this end, General Electric has developed extremely useful debugging aids, both on- and off-line.

Debugging aids are used by the programmer to check the operation of progress for their correctness. Program loaders provide the means of load programs or data into computer storage through card or paper-tape readers. Conversely, the dump program is used to record the contents of computer memory, either through the typewriter or paper-tape punch.

The Memory Change program provides the means for changing the contents of a core or drum location through the computer console switches. Documentation showing the location, contents before the change, and contents after the change are typed out on the console typewriter. The memory change program has the added facility of displaying the location and its contents before the change is executed.

The on-line debugging routines enable unchecked routines to be debugged on the computer without interrupting normal process monitoring and control. Safeguards are built into these on-line debugging aids to guard against unchecked programs destroying operating programs.

SECTION IV

FUNCTIONAL DESCRIPTION - GE/PAC SOFTWARE

SECTION IV

FUNCTIONAL DESCRIPTION - GE/PAC SOFTWARE

A. GENERAL

The successful operation of any process system depends on two functions - hardware and software. Hardware is the group of electronic components that acts physically upon information and signals from the processes. That portion of software called the system program is the intelligence that instructs the computer hardware to step through the various cycles of the processes. Other software packages aid in preparing the system program. General Electric process computer software:

- o Facilitates a systematic approach in writing computer programs.
- o Sufficiently flexible to allow for future changes in user processes.

Complete software packages for GE/PAC 4000 have been developed and are classified into 3 categories: Language Processors, Standard On-Line Functions, and Program Debugging Aids:

1. Language Processors

- o Translates functions into computer instructions.
- o Increases speed and accuracy in preparing program.
- o Aids in documentation and debugging of a program.
- o Operate on all GE/PAC 4000 Central Processors.

(a) Process Assemble Language

- o Enables programmer to code program in systematic and well documented fashion.
- o Generates single and double precision constants to a specified scale factor.
- o Provides built-in check features which detect and notify programmer of coding errors.

(b) Fortran Compiler

- o Enables powerful algebraic and verbal statements to be written with minimum time and effort.
- o Permits experienced Fortran programmers to make easy transition to GE/PAC programming.
- o Allows previously written Fortran programs to be adapted to the GE/PAC 4000 series process computers.

2. Standard On-Line Functions

- o Common routines suitable for numerous applications.
- o Require little or no re-work by user programmer.
- o Use of pre-checked routines eliminates programming and debugging time.

(a) Monitor

- o Schedules and gives priority assignment to system functions.
- o Enables user to have system running on-line in minimum time.
- o Provides communication between functions and peripheral equipment.

(b) Math Routines

- o Includes fixed, floating, single and double precision math routines.
- o Available from General Electric library to all GE/PAC 4000 users.
- o Assembled by General Electric to fit user routine parameters.
- o GE/PAC 4000 users receive up-to-date abstracts of new routines as they are developed.

(c) Free Time System

- o Real time language processing.
- o Real time debugging.
- o Use of functional programs in a real time environment.
- o Allows 100 percent usage of computer arithmetic and logic capabilities.

3. Debugging Aids

- o Debugging aids proven by field experience provide powerful means of keeping program documentation up to date.
- o Debugging aids provide user with loader, dump, memory change, and trace routines.
- o On-line debugging aids are a plug-in package to Monitor.

B. LANGUAGE PROCESSORS

Language processors are those preparation aids that enable a programmer to translate a function into actual computer instructions. Use of a symbolic language greatly increases the speed and accuracy of preparing a program and also aids in the documentation and debugging of the program. To provide the programmer with the most effectual means of coding a program, the General Electric Company has developed two powerful language processors - Process Assembler Language and Fortran II. These language processors are extremely flexible - providing for magnetic tape, paper tape, or punched card input. Additional service to the user has been provided as these processors operate on GE/PAC 4000, GE 412 or GE 225 computer systems. The GE 225 Language Processor version enables routines to be assembled or compiled at the various General Electric Information Processing Centers throughout the country and overseas.

1. Process Assembler Language

The Process Assembler Language accepts coded symbolic instructions and translates them into computer instructions. These symbolic instructions are coded by the programmer on a coding sheet from which cards are punched when the coding is completed. The cards then may be read into the computer on which the Assembler Program is operating.

The output from the Assembly Program is a listing from a printer or typewriter of the object program and a paper tape which is used to load the new program into the computer. In addition to this translation, the Assembler Program has built-in check features which detect and notify the programmer of coding errors. The listing provides documentation of the program and is invaluable as a debugging aid.

An On-Line Process Assembler Language program accepts the same symbolic language as above and has the ability to assemble programs while the GE/PAC 4000 is handling process data. Its advantage is that it can be used to assemble or re-assemble programs at the user site, and integrates them into the system with no interruption to process computing.

The listing provides the symbolic instructions as coded and also the octal core locations, instructions and constants of the assembler program. For further aid in debugging, the listing reproduces all comments from the coding sheet. Three columns of numbers represent:

Core location
Relative address
Absolute address

The Assembly Program makes maximum use of the relative addressing feature of the GE/PAC 4000. The Assembly Program assembles the operand value relative to the location of the instruction itself rather than the absolute value. With this feature, it is possible to move the program within memory, thereby adding a new dimension of flexibility to system program organization. In addition to the computer hardware instructions, the Assembly Program makes maximum use of pseudo instructions for storage assignments, symbol definition, and generation of constants to provide better programmer efficiency. These instructions include block storage reservation, single and double word constants, single and double word floating constants, etc.

2. FORTRAN Compiler

To make the writing of new programs as easy and efficient as possible, G.E. has created a Fortran Compiler for GE/PAC 4000. This compiler goes a step beyond the Process Assembler Language program in that it enables the programmer to write his program in terms of "statements" which employ familiar language and symbols rather than the symbolic code required by the Process Assembler Language. An example of such is:

$$Y = A/B + C - SIN (D + E)$$

where A, B, C, D, and E are symbols which have been defined by the programmer in previous statements. A statement such as this, when presented to the Fortran Compiler, will cause the compiler to automatically generate all the step-by-step machine instructions necessary to perform the calculations called for in the statement. Thus, the programmer is freed from the time-consuming details of step-by-step programming and allowed to concentrate more fully on the problem at hand.

In preparing the Fortran Compiler for the GE/PAC 4000, G.E. has incorporated several special features which facilitate the writing and running of programs in a real-time process control environment.

(a) Compatibility with Monitor

The compiler has been designed so that the programs it produces will have numerous special provisions for operation within the G.E. Monitor system. Thus, new programs may be easily incorporated into existing Monitor systems.

(b) Compatibility with Process Assembler Language

The programmer is free to intermix Fortran statements with Process Assembler Language statements within a single program. This allows the programmer to switch back and forth between the two languages arbitrarily, always free to choose the language in which he can proceed more efficiently. Output from the compiler is in the form of Process Assembler Language Symbolic coding.

(c) Bit Manipulation Capability

Special Fortran statements are available to the programmer through which he may exploit the ability of the GE/PAC 4000 to manipulate individual bits within a word. In this manner, individual bits may be treated as separate variables and may be set, reset, tested and operated upon with Boolean algebraic expressions.

(d) Drum-Core Transfers

Transfers of information between drum storage and core storage may be implemented through the Fortran Compiler by means of special statements provided for this purpose.

(e) Memory Economy

The Compiler has been designed so that the programs which it produces will require a minimum of memory space. This conservation of memory can mean increased flexibility during future additions of modifications of programs in cases where it is not desired to increase the memory size of the machine.

(f) Floating Point Operation

The Fortran Compiler will accept data in either integer or floating point form, and programs produced by the Compiler may be made to output data in either form.

(g) Statement Repertoire

A large repertoire of allowed statements, plus a full complement of "library" subroutines, makes for ease and flexibility in programming with the Fortran Compiler.

C. STANDARD ON-LINE FUNCTIONS

Standard on-line functions are portions of a total system program that are common from one computer application to another. Because they are used so frequently, optimum execution time and utilization of memory have been stressed in their development. For most applications, the use of these functions require little or no re-work by user programmers. Use of these pre-checked routines eliminates programming and debugging time.

1. MONITOR

MONITOR provides the skeleton of a real time program by scheduling and giving priority assignment to system functions. By using MONITOR the user is capable of having an on-line program running with minimum of time spent on the program. Additional functions can be added to MONITOR requiring no re-working of the running system.

The routines that make up MONITOR are clear and well defined, which makes for easy understanding of the system. The layout of the MONITOR package facilitates the addition of system functions by user programmers. The routines that comprise MONITOR are:

(a) Time and diagnostic count.
(b) Executive control program.
(c) Save register routine.
(d) Restore register routine.
(e) Turn off program routine.
(f) Set program delay
(g) Input/Output drive.

(h) Output program consisting of:
- Output subroutine
- Decimal floating point routine
- Decimal fixed point routine
- Octal conversion routine
- BCD conversion routine
- Build driver table routine
- Binary to decimal conversion subroutine

In addition to the above, MONITOR also includes the following two routines for core/drum GE/PAC 4000 computers:
- Drum transfer subroutine
- Drum transfer drive

The Time and Diagnostic Count routine essentially performs two functions. The routine maintains the value of current time (in seconds) which is used by the Executive Program in determining execution time of the functional programs. The second function of this routine is to detect if certain peripheral devices such as typewriters, punches, etc. fail. This routine will usually turn on a corrective action function which initiates remedial action.

The Executive Control Program (E.C.P.) is the real heart of the MONITOR system as it initiates functional programs according to their scheduled interval and priority. The ECP provides for easy writing and addition of functional programs as it performs real-time housekeeping chores such as saving and reloading arithmetic and index registers. The number of functional programs that the ECP can manage is limited only by the Computer core or drum size and the length of the individual functions. Most MONITOR systems allow 25 to 50 functional programs.

To accomplish on-line requirements as they occur, the GE/PAC 4000 uses automatic priority interrupt which may interrupt a program at any time. Since the various registers (arithmetic, location counter, index) must be saved for the interrupted program, MONITOR provides the Save Register and Restore Register routines that user interrupt programs can use.

To further reduce housekeeping chores of functional programs using MONITOR, Set Program Delay and Turn Program Off routines are supplied. These routines enable a user function to be initiated at a specified interval or be completely turned off until some exterior influence (such as a demand) re-initiates the function.

The Input/Output program permits user functional programs to communicate with peripheral devices such as typewriters, readers, punches, etc. in an orderly, step-by-step fashion. The Input/Output program determines if the peripheral is available for use by the functional program, reserves the peripheral for the functional program, turns power on the peripheral, reads in or puts out data, and makes the peripheral free for other functions upon conclusion of the present function. The Input/Output program eliminates a large amount of editing and bookkeeping that would be required if it were not available. The output portion of the program converts binary floating point to decimal fixed point output, binary fixed point to decimal fixed point output, binary to octal output, binary data to numeric and alphabetic character output. The Input/Output program operates the peripherals through automatic priority interrupt which insures the peripherals to operate at their maximum speed.

For GE/PAC 4000 computers equipped with a magnetic drum memory, MONITOR also provides the transfer linkage for information flowing between core and drum. The user functional program simply uses the Drum Transfer Subroutine to initiate a transfer of information between core and drum. While the transfer is complete, the ECP will re-initiate the functional program that requested the transfer. Completion of a drum/core transfer is detected by automatic priority interrupt which insures that the data involved in the transfer is acted upon expeditiously by the function.

MONITOR Compatible Functions

MONITOR Compatible Functions are those functions that are developed for specific applications which may be used by other computer users. These routines are written in a general form so that they require a minimum of re-working to be suitable for similar other applications. Examples are scanning, alarming, conversion equations, demand print, etc.

2. MATH ROUTINES

Math Routines which are usually used in subroutine form include fixed point, single and double precision math functions. These routines are available from the General Electric Company library to all GE/PAC 4000 computer users. They may be obtained in punched card form, symbolic listing or paper tape where the General Electric Company will assemble the routines to fit the user parameters. Examples of those routines are square root, trigometric, exponential functions, etc. GE/PAC 4000 users receive up-to-date abstracts of each routine as it is developed.

3. FREE TIME SYSTEM

The combined purpose of the Free-Time and Language Processing and Debugging Systems is to provide the user with the ability to compile, test, and execute functional programs in a real-time environment.

Existing service programs may be initiated by a control card. New programs may be tested and entered into the overall system in easy stages starting with an untested program and arriving at an operating real-time program.

The system provides "load and go" or "compile and go" with debugging at the symbolic level using the names of FORTRAN variables. The dynamic relocation of programs and allocation of storage frees the programmer from any concern other than the successful compilation and testing of his program.

GE/PAC 4000 LIBRARY PROGRAMS JANUARY 1966

LIB.
CTL.
NO.

PROGRAM TITLES
* * * * * * *

* *

PROGRAM CLASSIFICATION -(A)- SERVICE ROUTINES, LOADERS, DUMPS, OPERATOR PROGRAMS, MAGNETIC TAPE HANDLING SYSTEMS, ETC.

* *

YPA01	LDR41-GE/PAC LOADER PACKAGE.
YPA03	BTS41-GE/PAC PROG. LOAD RTN. PERIPH BUFF NO. 4200/4201
YPA04	BTS42-GE/PAC PROG. LOAD RTN. PERIPH BUFF NO. 4201
YPA05	DUP41-PAPER TAPE DUPLICATOR - OFF LINE
YPA06	DUP42-PAPER TAPE DUPLICATOR - ON LINE, MONITOR I
YPA08	CLK41-ON LINE CLOCK, MONITOR I
YPA09	POS41-PERIPHERAL IN/OUT OF SERVICE, MONITOR I
YPA10	PST41-PROGRAM STATUS, MONITOR I
YPA13	MCG42-MEMORY CHANGE W/PCH OPTION, MONITOR I
YPA17	POS42-PERIPHERAL IN/OUT OF SERVICE, MONITOR II
YPA18	PST42-PROGRAM STATUS, MONITOR II
YPA19	CLK42-ON CLOCK, MONITOR II
YPA20	OPR41-ON LINE OPERATOR PROGRAM, MONITOR I - REF YPG28
YPA21	OPR42-ON LINE OPERATOR PROGRAM, MONITOR II - REF YPG29
YPA22	MCG43-MEMORY CHANGE - ON LINE - I/O TYPER, MONITOR II
YPA24	MCG44-MEMORY CHANGE WITH PCH OPTION, MONITOR II
YPA25	DUP44-PAPER TAPE DUPLICATOR - ON LINE - MONITOR II
YPA26	OPR43-ON LINE OPERATOR PROGRAM - I/O TYPER
YPA27	DUP43-PAPER TAPE DUPLICATOR - I/O TYPER
YPA28	OLD41-ON LINE DUMP, MONITOR I
YPA29	OLD42-ON LINE DUMP, MONITOR II
YPA30	DMP41-OFF LINE DUMP, ALL CORE
YPA31	DMP42-OFF LINE DUMP, DRUM-CORE
YPA32	LDR43-ON LINE PAPER TAPE LOADER - I/O TYPER
YPA33	OLD43-ON LINE PAPER TAPE DUMP - I/O TYPER
YPA34	CMC41-CONSOLE SWITCH DRUM/CORE MEMORY CHANGE - OFF LINE
YPA42	PRT41-PRINTER HEADING SUBROUTINE MONITOR IV
YPA46	CLK43-ON-LINE CLOCK - I/O TYPER
YPA47	CDP41-OFF-LINE COMPARE DUMP
YPA50	DSN41-DEMAND SCAN (PRELIMINARY)
YPA51	CCG41-CONTROLLER CHANGE (PRELIMINARY)

* *

PROGRAM CLASSIFICATION -(C)- MATH ROUTINES FLOATING POINT TRANSCENDENTALS,
CURVE FITTING, SPECIAL FORMULAS, ETC.

* *

YPC01 GMR41-GE/PAC MATH ROUTINES - FLOATING POINT

YPC02 GMR42-GE/PAC MATH ROUTINES-FIXED POINT (ONLY SQRT AVAIL)

YPC03 GCC41-GE/PAC CODE CONVERSION ROUTINES

YPC04 FLR41-FORTRAN LIBRARY ROUTINES (FLOW CHARTS YPC01)

YPC05 FLR42-DBL.WD.FL.PT. FORTRAN LIBRARY (FLOW CHARTS YPC07)

YPC07 GMR43-GE/PAC MATH ROUTINES - DBL WD - FL.PT.

YPC08 TPS41-THERMODYNAMIC PROPERTIES OF STM(SUPERHEATED)-PKG 1

YPC09 TPS42-THERMODYNAMIC PROPERTIES OF STM(SATURATED)-PKG 2

YPC10 TPS43-THERMODYNAMIC PROPERTIES OF STM(COMPRESSED)-PKG 3

* *

PROGRAM CLASSIFICATION -(E)- GENERALIZED SYSTEMS MONITORS, PERIPHERAL I/O
+ OTHER HARDWARE COMMUNICATION REGISTER DRIVER
PROGRAMS, SIMULATORS, ETC.

* *

YPE02 SIM02-GE/PAC SIMULATOR ON GE412

YPE03 SBM41-FORTRAN SUB-MONITOR, PERMANENT CORE

YPE04 SIM03-GE/PAC SIMULATOR ON THE GE225

YPE05 SBM42-FORTRAN SUB-MONITOR, DRUM-CORE (FLOW CHART YPE03)

YPE07 CAL41-CALENDAR PROGRAM

YPE08 -MONITOR I COMPATIBLE PROGRAMS - REF YPE10,YPG13

YPE10 MTR41-GE/PAC 4000 MONITOR I - ALL CORE - REF YPG13

YPE11 ECP41-EXECUTIVE CONTROL PROGRAM, MONITOR I

YPE12 ITC41-TIME AND DIAGNOSTIC COUNT, MONITOR I

YPE13 IOD41-INPUT/OUTPUT DRIVER, MONITOR I AND II

YPE14 IOP41-OUTPUT PROGRAM, MONITOR I

YPE15 INP41-INPUT SUBROUTINE AND PROGRAM, MONITOR I

YPE16 SRG41-SAVE REGISTERS SUBROUTINE, MONITOR I

YPE17 RRG41-RESTORE REGISTERS SUBROUTINE, MONITOR I

YPE18 OFF41-TURN PROGRAM OFF SUBROUTINE, MONITOR II

YPE19 DEL41-SET PROGRAM DELAY SUBROUTINE, MONITOR I

YPE20 OUT41-OUTPUT SUBROUTINES, MONITOR I

YPE21 TPN41-TURN PROGRAM ON SUBROUTINE, MONITORS I,II,III,IV

YPE22 PR141-PRIORITY CHANGE STORAGE SUBRTN., MONITOR I,II

YPE23 CAD41-CORRECTIVE ACTION DIAGNOSTIC, MONITOR I

YPE24 MOR41-MULTIPLE OUTPUT REQUEST SUBRTN., MTRS I,II,III,IV

YPE25 MDR41-MULTIPLE OUTPUT DIST. DRIVER, MTRS I,II,III,IV

YPE27 SND41-SCAN DRIVER, MONITOR I

YPE29	PAV41-PERIPHERAL AVAILABILITY SUBRTN., MTR. I,II,III,IV
YPE30	IOP43-OUTPUT PROGRAM - I/O TYPER
YPE32	INZ41-INITIALIZE ROUTINE, MONITOR I
YPE33	IOD 43-INPUT/OUTPUT DRIVER - I/O TYPER, MONITOR II
YPE34	OPR43-ON-LINE OPERATOR PROGRAM I/O TYPER DRUM/CORE
YPE35	TCO41-TIMED CONTACT OUTPUT REQUEST SR. MTR. I,II,III,IV
YPE36	TCD41-TIMED CONTACT OUTPUT DRIVER MONITOR I,II,III,IV
YPE37	INS42-INPUT REQUEST SUBRTN., BUFFERED INPUT, MTR II
YPE38	INP43-INPUT PROGRAM - I/O TYPER, MONITOR II
YPE39	ITC42-TIME AND DIAGNOSTIC COUNT, MONITOR II
YPE40	MAP41-CORE MAP MAINTENANCE SUBROUTINE, MONITOR II
YPE41	MTR42-GE/PAC 4000 MONITOR II, DRUM-CORE, REF YPG15
YPE42	ECP42-EXECUTIVE CONTROL PROGRAM, MONITOR II
YPE43	DTR41-DRUM TRANSFER REQUEST SUBRTN, MONITOR II,IV
YPE44	DTD41-DRUM TRANSFER DRIVER, MONITOR II
YPE45	IOP42-OUTPUT PROGRAM, MONITOR II
YPE46	CAD42-CORRECTIVE ACTION, MONITOR II
YPE47	INP42-INPUT SUBROUTINE, MONITOR II
YPE48	OUT42-OUTPUT REQUEST SUBRTN., MONITOR II,IV
YPE49	SND42-SCAN DRIVER SUBRTN , MONITOR II
YPE50	DEL42-SET PROGRAM DELAY SUBROUTINE, MONITOR II,IV
YPE53	OFF42-TURN PROGRAM OFF, MONITOR II,IV
YPE54	SRG42-SAVE REGISTERS SUBRTN., MONITOR II
YPE55	RRG42-RESTORE REGISTERS SUBRTN., MONITOR II
YPE56	INZ42-INITIALIZE ROUTINE, MONITOR II,IV
YPE57	-MONITOR II COMPATIBLE PROGRAMS - REF YPE41,YPG15
YPE58	CAD43-CORRECTIVE ACTION DIAGNOSTIC - I/O TYPER
YPE59	OUD43-OUTPUT DRIVER - I/O TYPER
YPE63	INS41-INPUT SUBROUTINE, MONITOR II
YPE64	FMR41-FIND/RESTORE WORKING CORE AREA SUBRTN. MTR II,IV
YPE75	MTR44-GE/PAC AU2 MONITOR IV
YPE76	ECP44-EXECUTIVE CONTROL PROGRAM, MONITOR IV
YPE77	ITC44-TIME AND DIAGNOSTIC COUNT, MONITOR IV
YPE78	SRG44-SAVE REGISTERS SUBRTN., MONITOR IV
YPE79	RRG44-RESTORE REGISTERS SUBRTN., MONITOR IV
YPE80	MAP44-CORE MAP MAINTENANCE PROGRAM, MONITOR IV
YPE81	DTD44-DRUM TRANSFER DRIVER, MONITOR IV
YPE82	FRP44-FIND REGISTER POINTER, MONITOR IV
YPE83	IOP44-OUTPUT PROGRAM, MONITOR IV
YPE84	IND44-INPUT DRIVER, MONITOR IV
YPE85	OUD44-OUTPUT DRIVER, MONITOR IV
YPE86	SND44-SCAN DRIVER, MONITOR IV
YPE87	CAD44-CORRECTIVE ACTION DIAGNOSTIC, MONITOR IV
YPE88	OFF44-TURN PROGRAM OFF SUBROUTINE, MONITOR IV
YPE89	-MONITOR IV COMPATIBLE PROGRAMS - REF YPE75,YPG18

* *

PROGRAM CLASSIFICATION -(F)- LANGUAGE PROCESSORS TRANSLATORS, ASSEMBLERS,
COMPILERS, INTERPRETERS, SPECIAL SERVICE
ROUTINES FOR LANGUAGE PROCESSORS - ETC.

* *

YPF01	PAL41-PAL/412 ASSEMBLER
YPF02	PAL42-PAL 2K GE/PAC ASSEMBLER - OFF LINE
YPF03	PAL43-PAL 4K GE/PAC ASSEMBLER - OFF LINE

YPF04 PAL44-PAL GE/PAC ASSEMBLER DRUM/CORE ON LINE
YPF05 PAL45-PAL/225 ASSEMBLER
YPF07 CPT42-BINARY CARD TO PAPER TAPE TRANSLATOR-225
YPF08 COR41-PAL CORRECTION PROGRAM - OFF LINE
YPF09 COR42-PAL CORRECTION PROGRAM - ON LINE
YPF10 XLT41-EXTRACT LOAD TAPE - OFF LINE
YPF11 XLT42-EXTRACT LOAD TAPE - ON-LINE
YPF12 FTN41-GE/PAC FORTRAN COMPILER-412
YPF13 FTN42-GE/PAC FORTRAN-PAL COMPILER/ASSEMBLER-225
YPF14 FTN43-GE/PAC FORTRAN COMPILER - OFF-LINE
YPF15 FTN44-GE/PAC FORTRAN COMPILER - ON LINE
YPF16 FTN45-DBL WD. FL. PT. GE/PAC FORTRAN COMPILER-412
YPF17 PTP01-PAP TO PAL TRANSLATOR-412
YPF18 TAS03-GE/PAC TASC (OPERATES ON 412 ONLY)
YPF19 CCP41-CARD CONV.-12 BIT BINARY TO GE/PAC INTERNAL CODE

* *

PROGRAM CLASSIFICATION -(G)- SPECIAL PROJECTS REPORTING
SYSTEMS, MANUALS

* *

YPG10 GE/PAC 4000 INSTRUCTION REFERENCE MANUAL
YPG11 GE/PAC 4000 PROGRAMMING TECHNIQUES MANUEL
YPG12 GE/PAC 4000 PROCESS ASSEMBLER LANGUAGE (PAL)
YPG13 GE/PAC 4000 MONITOR I USERS MANUAL - ALL CORE -AU1
YPG14 GE/PAC 4000 FORTRAN REFERENCE MANUAL
YPG15 GE/PAC 4000 MONITOR II USERS MANUAL-DRUM/CORE AU1
YPG16 GE/PAC 4000 MTR I ALL CORE AU1 COLLECTION W-U, FC
YPG17 GE/PAC 4000 MONITOR II D/C AU1 COLLECTION W-U, FC
YPG18 GE/PAC 4000 MONITOR IV USERS MANUAL (SUPERSEDED BY YPG31)
YPG19 GE/PAC 4000 FREE TIME SYSTEM USERS MANUAL
YPG28 GE/PAC 4000 OPERATOR CONSOLE MANUAL OPR41,YPA20
YPG29 GE/PAC 4000 OPERATOR CONSOLE MANUAL OPR42,YPA21
YPG30 GE/PAC 4000 DOUBLE-WORD FORTRAN REF MANUAL, YPF16
YPG31 GE/PAC 4000 MONITOR TRAINING MANUAL
YPG35 GE/PAC OCT.FL.PT.SIN.REG.-DEC.FT.PT.CONV.TABLES

* *

PROGRAM CLASSIFICATION -(Q)- QUASI PACKAGES EACH DIFFERENT COMBINATION OF
QUASI S WILL BE SEPARATELY + UNIQUELY NUMBERED

* *

YPQ01 QUA41-GE/PAC 4040 QUASI INST. S.R. PKG / W/ MPY STEP
YPQ02 QUA42-GE/PAC 4040 QUASI INST. S.R. PKG 2 NON-MPY STEP
YPQ03 QUA43-GE/PAC 4040 QUASI PKG. DBL. WD. FL. PT.-MPX STEP
YPQ04 PLA41-PARTIAL WORD FIXED POINT ARITH. PKG. 1
YPQ05 QUA44-SINGLE WORD FL. PT. QUASI FOR AU2 - REF YPG10
YPQ06 QUA45-DOUBLE WORD FL. PT. QUASI FOR AU2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

PROGRAM CLASSIFICATION  -R-  MANUAL INPUTS - OPERATOR PANEL AND OTHER DECADE
                             SWITCH PROGRAMS, DFS INPUTS, CARD INPUTS (CUSTOM
                             ORIENTED, NOT STANDARD FORMATTED SERVICE USF).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

YPR33 ADS41-ANALYZE DECADE SWITCH SETTING SUBROUTINE


* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

PROGRAM CLASSIFICATION  -T-  TRACKING  ANALOG  SCANNING  PLANT  MONITORING,
             (T)             PROCESS OR UNIT TRACKING, ETC.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


YPT06 SCR41-SCAN REQUEST SUBROUTINE, MONITOR I
YPT11 SCR42-SCAN REQUEST SUBROUTINE, MONITOR II
YPT15 SCR44-SCAN REQUEST SUBRTN., MONITOR IV
YPT37 SCF41-SCAN OFFSET PROGRAM, MONITOR I
YPT59 SCF42-SCAN OFFSET PROGRAM, MONITOR II,IV
```

STANDARD ON-LINE FUNCTIONS

Standard on-line functions are portions of a total system program that are common from one computer application to another. Because they are used so frequently, optimum execution time and utilization of memory have been stressed in their development. For most applications the use of these functions require little or no re-work by user programmers. Use of these pre-checked routines eliminates programming and debugging time.

MONITOR

MONITOR provides the skeleton of a real-time program by scheduling and giving priority assignment to system functions. By using MONITOR the user is capable of having an on-line program running with minimum of time spent on the program. Additional functions can be added to MONITOR requiring no re-working of the running system.

The routines that make up MONITOR are clear and well defined which makes for easy understanding of the system. The layout of the MONITOR package facilitates the addition of system functions by user programmers. The routines that comprise MONITOR are:

- Time and diagnostic count
- Executive control program
- Save registers routine
- Restore register routine
- Turn off program routine
- Set program delay
- Input-output drive
- Output program consisting of:
  - Output subroutine
  - Decimal floating point routine
  - Decimal fixed point routine
  - Octal conversion routine
  - BCD conversion routine
  - Build driver table routine
  - Binary to decimal conversion subroutine

SOFTWARE SPECIFICATIONS

Object Computer: GE/PAC 4000

PROCESS ASSEMBLER LANGUAGE

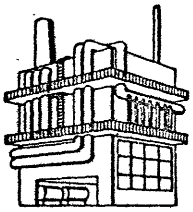
Compiling Computer:	GE 215, GE 225, GE 235, GE 412
Input:	Punched Card or Magnetic Tape
Memory Requirement:	8K Core
Peripheral Equipment:	Console Typewriter High-Speed Printer Card Reader or Magnetic Tape Units Paper Tape Punch!
Compiling Computer:	GE/PAC 4000
Input:	Punched Cards or Paper Tape
Memory Requirement:	2 - 4K Core (Off-line) 2K Core plus 6K Drum (On-line) (in addition to process memory requirements)
Peripheral Equipment:	Card and Paper Tape Reader Console Typewriter or Printer Paper Tape Punch
Output:	Paper Tape and Program Listing

FORTRAN II COMPILER

Compiling Computer:	GE 215, GE 225, GE 235, GE 412
Input:	Punched Cards or Magnetic Tape
Memory Requirement:	8K Core
Peripheral Equipment:	Card Reader Console Typewriter Card Punch High-Speed Printer Magnetic Tape
Output:	Punched Cards and Program Listing (Output is in Process Assembler Language format)
Compiling Machine:	GE/PAC 4000
Input:	Paper Tape
Memory Requirement:	8K Core (Off-line) 4K Core plus 8K Drum (On-line)
Peripheral Equipment:	Paper Tape or Card Reader Paper Tape or Card Punch Console Typewriter High-Speed Printer (Option)
Output:	Paper Tape or Cards and Program Listing (Output is in Process Assembler Language format)

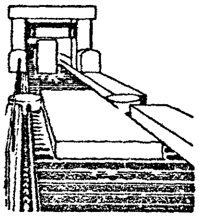
Mnemonics	Description
MAQ	Move A Into Q
OOM	Operate On Memory
STA	Store Contents of A
STQ	Store Contents of Q
GROUP V Fixed Point Arithmetic	
ADD	Add
ADK	Add K to A
ADM	Add A to Memory
ADO	Add One to Bit K
DAD	Double Add
DLA	Double Left Arithmetic
DRA	Double Right Arithmetic
DSU	Double Subtract
DVD	Divide
MPY	Multiply
NEG	Negate
SLA	Shift Left Arithmetic
SRA	Shift Right Arithmetic
SUB	Subtract
GROUP VI Floating Point Arithmetic	
FAD	Flaoting Point Add
FSU	Floating Point Subtract
FMP	Floating Point Multiply
FDV	Floating Point Divide
FIX	Fix Floating Number
FLO	Float Fixed Number
GROUP VII Word Logic	
ANA	Add to A
ANM	Add to Memory
CPL	Complement A
DLL	Double Left Logical
DRC	Double Right Circular
DRL	Double Right Logical
ERA	Exclusive Or to A
ERM	Exclusive Or to Memory
ORA	Or to A
ORM	Or to Memory
SLL	Shift Left Logical
SRC	Shift Right Circular
SRL	Shift Right Logical
GROUP VIII Bit Logic	
CBK	Change Bit K
IBK	Isolate Bit K
LBM	Load Bit Mask
RBK	Reset Bit K
SBK	Set Bit K

Mnemonics	Description
GROUP IX Word Tests	
RNZ	Reset TSTF If A is Non Zero
SNZ	Set TSTF If A is Non Zero
TNZ	Test A Non Zero
TZE	Test A Zero
GROUP X Partial Word Tests	
CLO	Count Least Significant Ones
CLZ	Count Least Significant Zeros
CMO	Count Most Significant Ones
CMZ	Count Most Significant Zeros
COM	Compare On Mask
LXC	Load X With Count
TSC	Test and Shift Circular
GROUP XI Bit Test	
REV	Reset TSTF If Bit K is Even
ROD	Reset TSTF If Bit K is Odd
SEV	Set TSTF If Bit K is Even
SOD	Set TSTF If Bit K is Odd
TEV	Test Bit K Even
TOD	Test Bit K Odd
GROUP XII Validity Tests	
JNO	Jump If No Overflow
JNP	Jump If No Parity Error
GROUP XIII List Instructions	
ABL	Append To Beginning of List
AEL	Append To End of List
RBL	Remove From Beginning of List
REL	Remove From End of List
GROUP XIV Variable Field Arithmetic	
ADF	Add Field
LDF	Load Field
SBF	Subtract Field
STF	Store Field
TFE	Test Field Equal
TFL	Test Field Less
GROUP XV Program Linkage	
LRM	Load Registers From Memory
SRM	Store Registers Into Memory

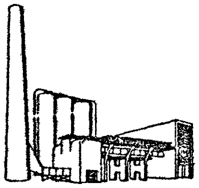


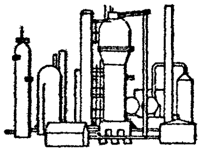
GE-PAC® 4000

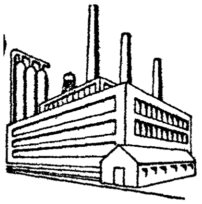
GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER

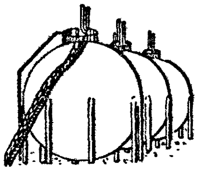


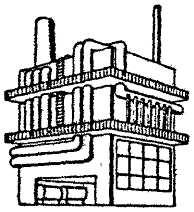
PROGRAMMING TECHNIQUES MANUAL

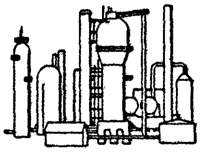


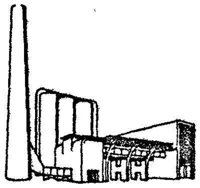


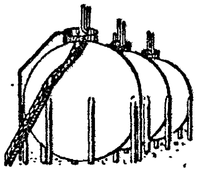


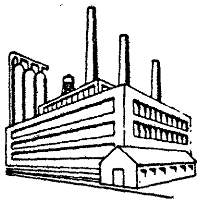


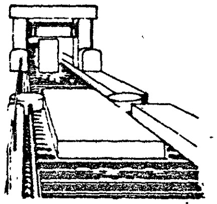












PROCESS COMPUTER
BUSINESS SECTION
PHOENIX, ARIZONA

GENERAL ELECTRIC

Library Control No. YPG11M

Revised December, 1966

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. BOX 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

GENERAL ELECTRIC
PROCESS COMPUTERS

GE/PAC 4000 PROGRAMMING TECHNIQUES MANUAL

Library Control No. YPG11M

REVISION CONTROL SHEET

APPROVED BY: R. G. Erickson DATE: December 30, 1966

V.	RECORD OF CHANGE	DATE	REV.	RECORD OF CHANGE	DATE
	Revised in entirity and republished				
	to include all model number changes.				

CONTENTS

CONTENTS

1.1 ARITHMETIC AND CONTROL UNIT

The computation center of the computer, the arithmetic and control unit performs a wide variety of arithmetical and logical operations at high speed.

It is subdivided into two parts: elements comprising the arithmetic unit, and instruction format and sequencing.

1.1.1 ELEMENTS COMPRISING THE ARITHMETIC UNIT

The arithmetic and control unit (AU) is comprised of data registers, full address, and control Flip-Flops as shown in Figure 1. The registers used are as follows: A, B, I, P, and J. Control Flip-Flops, along with the full address, work in conjunction with the registers to perform arithmetic and bit manipulation operations, and for checking and remembering conditions occuring in the AU. Names of the Flip-Flops employed are: Test (TSTF), Overflow (OVRF), Permit Automatic Interrupt (PAIF), Priority Interrupt First (PI1F), Execution (XECF), Demand (DEMF). Adders are: Carry - Full Adders; A, B, and P. The function and description of each of these elements is discussed in the succeeding paragraphs.

A-Register -- the A-Register is the primary working register for the arithmetic unit. It is comprised of 24 high speed Flip-Flops in a bit configuration numbered 0-23; bit 23 is the most significant. Functionally, it acts as temporary storage for data coming from or going to the input/output equipment of the computer. It is the accumulator register during arithmetic and bit manipulation operations. Transfer of data from A to internal registers of the AU is accomplished serially. Data transferred from A to registers and devices external to the AU is accomplished in parallel.

B-Register -- the B-Register is a 24-bit parallel entry buffer register used between core memory and the AU. It is comprised of 24 high speed Flip-Flops arranged in the same bit configuration as the A-Register. B is the communication link for information transfer between memory and the AU registers A, I, and P.

I-Register -- the I-Register is a 24-bit register comprised of 24 high speed Flip-Flops arranged in the same bit configuration as the A- and B-Registers. It is the holding register for the bits that control the operation of the AU.

Full Adder A -- the FA$\underline{A}$ is used in any AU operation involving the A-Register.

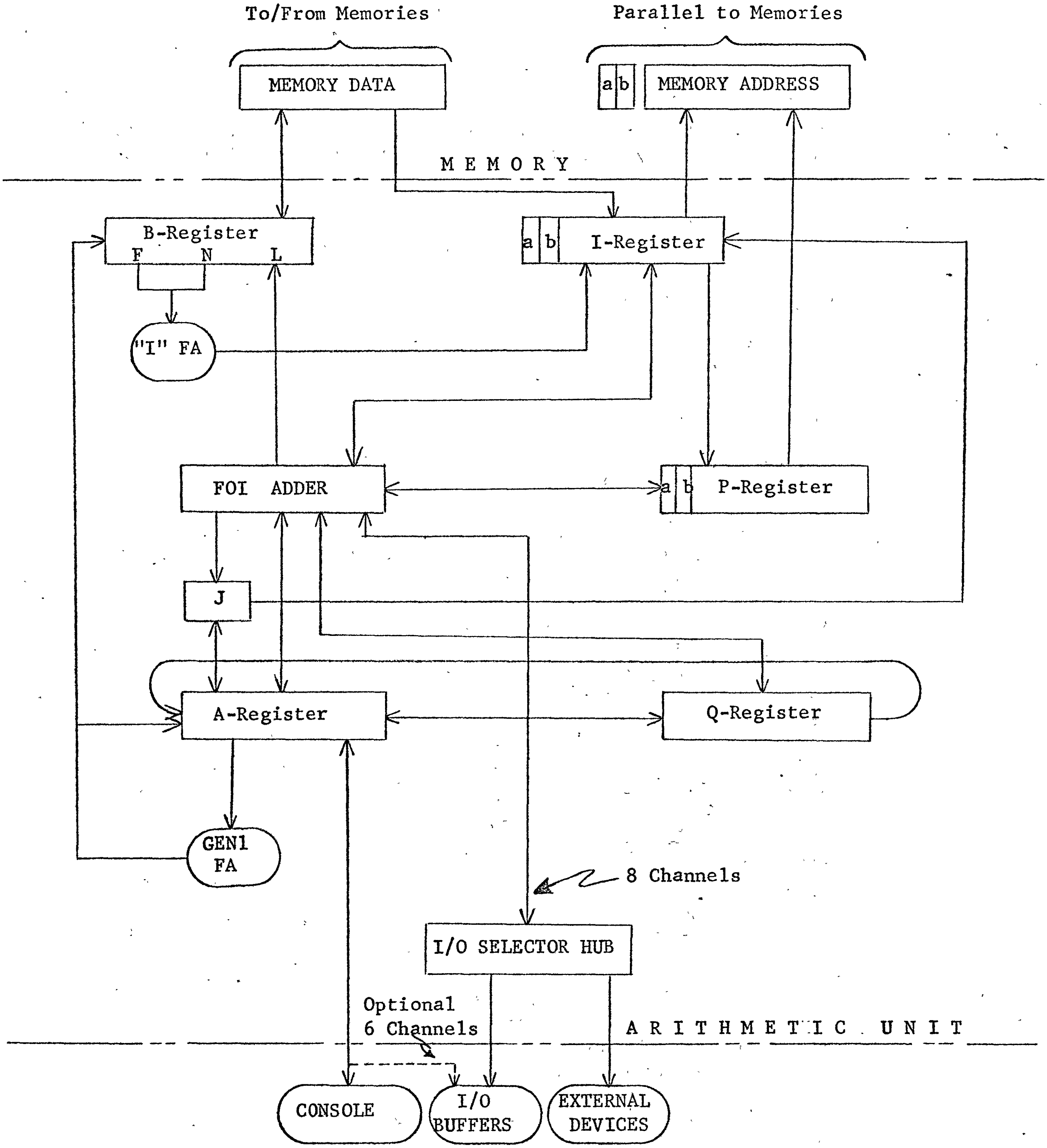


Figure 1 - CENTRAL PROCESSOR SIMPLIFIED BLOCK DIAGRAM

Full Adder B -- the FAB is used in any AU operation involving the I and B-Registers.

Full Adder P -- the FAP is used in any AU operation involving the modification of the contents of the P-Register.

1.1.2 INSTRUCTION FORMAT AND SEQUENCING

There are five types of instructions employed by the arithmetic and control unit: Full Operand, Quasi, GEN 1, GEN 2, and GEN 3. Each distinct instruction involves either memory addressing, Input/Output device selection, bit manipulation of the A-Register, or extended function commands. The bit configuration of each type of instruction determines the operation to be performed and the control necessary to perform it. Figure 2 summarizes these types.

The Full Operand type of instruction is used to perform arithmetic operations, logical operations, index control operations, and data transfers to and from memory. Data transferred to memory may be internal with the AU or may originate from input/output equipment. Data transferred from memory may go to the AU internal or be transferred through the AU to the input/output equipment.

Quasi instruction types are extended function commands that provide operations not wired into the AU hardware. They are implemented by packaged subroutines. The defined operation is analogous to a Save I Branch instruction.

Save I may be understood by analyzing what occurs when a quasi instruction is implemented. The quasi instruction is transferred from memory to the B-Register, then to the I-Register to be decoded. The operand, Bits 12-0, is stored in core cell 002 or, if desired, are modified, then stored (Save I). The next instruction is addressed from Bits 23-18 which selects one core cell location 040-077. The instruction contained in one of these cells "branches" the sequential program into a subroutine. Quasi instructions have the same instruction format as the Full Operand instructions but are distinguished by a "1" placed into Bit 23. Since Bits 23-18 are used to address the next instruction, the 1 bit in position 23 insures that the core cell addressed is located between 040 and 077. Examples of quasi instructions employed by this system are: Mulitply, Divide, Floating Point Arithmetic, and those instructions involving the use of double-length registers.

GEN 1 instructions are used for bit manipulations of the A-Register. By controlling the operation of Full Adder A, individual bits of the A-Register may be shifted in position, masked by ones or zeros, tested for polarity, or counted for numbers of ones or zeros contained therein. Microcoding of the instruction may be manipulated to perform any desired function on the A-Register.

GEN 2 instructions are employed by the GE/PAC system to select modules and devices in the input/output equipment. The format is microcoded, using unique address bits to select devices and modules. GEN 2 instructions are only partially decoded in the AU and the selection portion of its bit configuration (Bits 14-0) are never decoded in the AU. They are transferred to the input/output equipment for device and module selection.

GEN 3 instructions are used to manipulate the contents of the A- and Q-Registers, affect the J-Counter,and optionally affect the Overflow Flip-Flop.

All five instruction types may be modified. Modification is accomplished by indexing and relative addressing. Indexing may be performed on all the instruction types; relative addressing is limited to Full Operand and Quasi instructions.

One of the core cells 001 through 007 is specified by Bits 17 through 15 of the instruction. The contents of the selected core cell is added to Bits 13-0 of the instruction.

Relative addressing, as well as indexing, augments Bits 13-0 of the instruction (the operand in the case of the Full Operand and Quasi instructions). Modification takes place by adding the core cell address of the instruction to the instruction operand. The core cell address for most instructions is the bit information contained in the P-Register.

Instruction sequencing for the AU is as follows:

1. The P-Register transfers information to the memory address register (MAR) and a core cell is addressed.
2. The instruction contained therein is transferred, in parallel, via the memory data register (MDR) to AU buffer register B.
3. B (23-14) is transferred, in parallel, to the I-Register for instruction decodings.
4. B (13-0) is transferred, in serial, through Full Adder B to I (13-0). If desired, the contents of I are modified at this point.

Depending on the instruction decoded, the contents of the I-Register, modified or unmodified, will cause:

1. Address memory and retrieve data for arithmetic and logical operations.
2. Transfer data to or from memory.
3. Transfer data to or from the input/output equipment.

<table>
<tr><th></th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>FULL OPERAND
Hardware</td><td>0</td><td colspan="5">ØP</td><td colspan="3">X</td><td>*</td><td colspan="14">Y</td></tr>
<tr><td>FULL OPERAND
Quasi</td><td>0</td><td colspan="5">ØP</td><td colspan="3">X</td><td>*</td><td colspan="14">Y</td></tr>
<tr><td>GEN 1</td><td colspan="6">0 0 0 1 0 1</td><td colspan="3">X</td><td colspan="10">G</td><td colspan="5">K</td></tr>
<tr><td>GEN 2</td><td colspan="6">0 1 0 1 0 1</td><td colspan="3">X</td><td colspan="4">S</td><td colspan="11">D</td></tr>
<tr><td>GEN 3</td><td colspan="6">1 0 0 1 0 1</td><td colspan="3">X</td><td colspan="10">G</td><td colspan="5">K</td></tr>
</table>

Figure 2 - INSTRUCTION FORMATS

ØP	Instruction Octal
X	Index Word Indicator
*	Relative Addressing Indicator
G	Micro-coded Subcommand
K	Bit Position of A-Register or Length of Shift
S	GEN2 Subcommand
D	I/O Device Address

Bit position 23 of the Full Operand formats above is used to designate either a hardware of quasi implemented instruction.

These formats will assist in clarifying the following discussions on programming features.

1.2 REAL-TIME PROGRAM CONTROL

A real-time process is characterized by the occurrence of many events; some continuous, others random in nature. Events may occur simultaneously. Consequently, real-time programs are distinctively different from their non-real-time counterparts.

The most distinctive characteristic feature of a real-time program is the seemingly disorganized method of execution.

1. Programs need not be completed without the occurrence of several interruptions to execute entirely unrelated programs.
2. Internal data transfer is rarely accomplished in one operation.
3. Intercommunication between programs becomes significant and time consuming. Information is constantly stored and retrieved from small tables. Table storage and retrieval rates are asynchronous.
4. If a program is likely to be interrupted prior to completion, care must be taken to insure that its working storage is not destroyed.
5. Subroutines common to several programs must receive special attention. Temporary constants used by a subroutine with respect to a given program must be preserved until execution of the subroutine is completed.

The second distinctive feature is that the program is an active element in the process system. It must maintain real-time, and initiate actions at specified real-times, and specified time delays.

The third distinctive feature is the requirement that it should never wait upon I/O equipment; there is always some time-critical function to be implemented. In an off-line computer system, lost time increases costs. Lost time in an on-line operation will cause a failure; endangering system security.

The preceding characteristic features indicate that the "housekeeping functions", necessary in every computer program, become extremely important in the real-time program. The program capabilities of a process control computer must first provide efficient houskeeping (program control) capabilities, then provide arithmetic capabilities.

1.2.1 PROGRAM CONTROL

A digital computer is a serial device; it performs its program operations one-by-one.

A computer program consists of an ordered sequence of instructions to the computer. These instructions are placed in memory cells which have sequential addresses; ordering addresses will order the instructions.

The term "program control" describes the instruction sequencing process and is associated with a location in memory. Program control is specified by the contents of the program counter. Program control is normally transferred from one location to the next sequential location as instructions are executed. Program control can be transferred to an arbitrary location by "branch" instructions. It can conditionally skip a location by "jump" instructions.

Nominally, the program counter contains the address within the program of the instruction which is to be executed next. However, there are many exceptions. Most modern process computers have "execute" and "program interrupt" functions which complicate the instruction sequencing process. It is paradoxical that this complication is one of programming description and not one of hardware.

The effect upon program control by the following GE/PAC functions is similar:

1. Execute (XEC) instruction
2. Quasi subroutine linkage instruction
3. Program interrupt

All three are "execute" actions which command the computer to obtain its next instruction from some location other than that specified by the program counter. <u>One</u> instruction, out of normal sequence, is executed. If this (inserted) instruction is not a branch (jump, XEC, Quasi), program control is transferred to the next sequential location (with respect to the program counter) in the normal sequence.

The location of the object instruction is determined by the specific action; XEC uses its operand address, Quasi's use their OP codes. Program interrupt requires an address source external to the computer.

An XEC or Quasi action inhibits normal incrementing of the program counter during the execution of the initiating instruction. The execution of the inserted instruction is normal. A program interrupt action inhibits normal incrementing of the program counter during the execution of the inserted instruction; a branch in the interrupt location may transfer program control.

1.2.2 MEMORY ADDRESSABILITY

GE/PAC memory addressability satisifes two prime addressing requirements for process computers. The first of these is the ability to transfer a program from bulk storage (drum memory) to core memory and instantly relocate all of its memory addressing instructions. The term "dynamic relocatability" describes this requirement. The solution is GE/PAC relative addressing, which modifies the Y operand address of the instruction by is location in memory as the instruction is executed. It allows the on-line Executive Control Program to instantly relocate a program from drum to any available place in core. More efficient use of both core and drum is possible. Difficulties of incorrect problem definition, system analysis, or programming analysis are minimized.

The second addressing requirement is efficient program addressability to at least 64,000 words of random access core memory. This requirement is medium range, but has extremely long range implications. GE/PAC relative addressing and drum/core transfer provide a maximum of one million core words.

1.2.3 ADDRESS MODIFICATION

1.2.3.1 Relative Address Modification

The GE/PAC instruction format allocates a one-bit field, I* to specify relative address modification. This bit is called the relative addressing indicator and is designated by an asterisk. Relative address modification may occur in any Full Operand and Quasi instruction. It may not occur in any GEN1 or GEN2 instruction; this bit position, I_{14}, is used for other purposes in these instructions.

When * is zero in a Full Operand or Quasi instruction, no relative modification occurs. When * is 1, a modified operand address[1] is computed by adding an addend address to the signed 13-bit operand address. The normal addend address is that of the location that the instruction occupied in memory. An exception exists when the instruction being executed is the "object instruction" of a Quasi instruction or of a program interrupt; in these cases, the addend address is obtained from the program counter. Since this address is meaningless, branch vectors for Quasi's and program interrupts must not use relative addressing.

[1] The address is 14 bits long in the GE/PAC 4040; it is 16 bits long in the 4050/4060.

1.2.3.2 Indexing Address Modification

The GE/PAC instruction format allocates a three-bit field, I_x to indicate indexing address modification. The field specifies seven consecutive core memory locations for use as index words. The content of the field is call the "X-word address."

When indexing address modification is specified (the field is non-zero), the 14-bit address[2] in the specified index word is added to the 14-bit operand address[3] of the instruction in the I-Register to form the effective operand address.

X-Word Address	Effect Upon Address Modification
000	No Modification
001	Indexing Modification Using X-Word 1
010	Indexing Modification Using X-Word 2
011	Indexing Modification Using X-Word 3
100	Indexing Modification Using X-Word 4
101	Indexing Modification Using X-Word 5
110	Indexing Modification Using X-Word 6
111	Indexing Modification Using X-Word 7

Indexing address modification occurs as defined above when the X-word indicator is non-zero for all instructions except the six indexing control instructions. The indexing control instructions provide the means to load, store, increment, and test individual index words without using the A-Register. Details are presented in the GE/PAC 4000 Instruction Reference Manual. The usage of X-word 1 and X-word 2 is restricted. X-word 1 is used automatically by the arithmetic unit for subroutine linkage, Quasi instruction linkage, and program interrupt linkage. X-word 2 is automatically used for Quasi instruction linkage. Generally, these two X-words should not be used within a program.

When the X-word address in an indexing control instruction is zero, the instruction changes its meaning:

STX	($06X_8$)	becomes DMT 060
INX	($24X_8$)	LDX ($16X_8$), LXK ($07X_8$) and store zeros
LXC	($17X_8$)	into location zero.

STX, LXC, LDX, LXK, INX and TXH are illegal for X=0, by programming convention in the assembly program.

[2] The address is 14 bits long in the GE/PAC 4040; it is 16 bits long in the 4050/60; 15 bits in the 4020.

[3] Relative address modification, if required, occurs prior to indexing address modification.

1.2.3.3 The Purpose and Nature of Indexing

The fundamental purpose of indexing is to accomplish address modification of memory addressing instructions. A memory address is always positive. Indexing address modification is implemented within the arithmetic unit by addition of the index and positive operand address Y[4] of the instruction.

Memory addressing instructions must have positive operand addresses. For example, the instruction LDA -1, X which is correctly executed on many computers, is an illegal GE/PAC statement.[5] The GE/PAC assembly program will tag all negative absolute addresses as possible errors.

An index is, in essence, an address. Therefore it is considered as positive. The TXH instruction presumes both the index X and the test value K to be positive, it is dangerous to interpret an index as negative.

Indexing also permits the modification of GEN1 and GEN2 instructions. A portion of the operation code extends into the operand address field; indexing permits modification of the instruction within its lower 14 bits (I_{13-0}). Modification is MODULO 16K.

1.2.4 SUBROUTINE LINKAGE

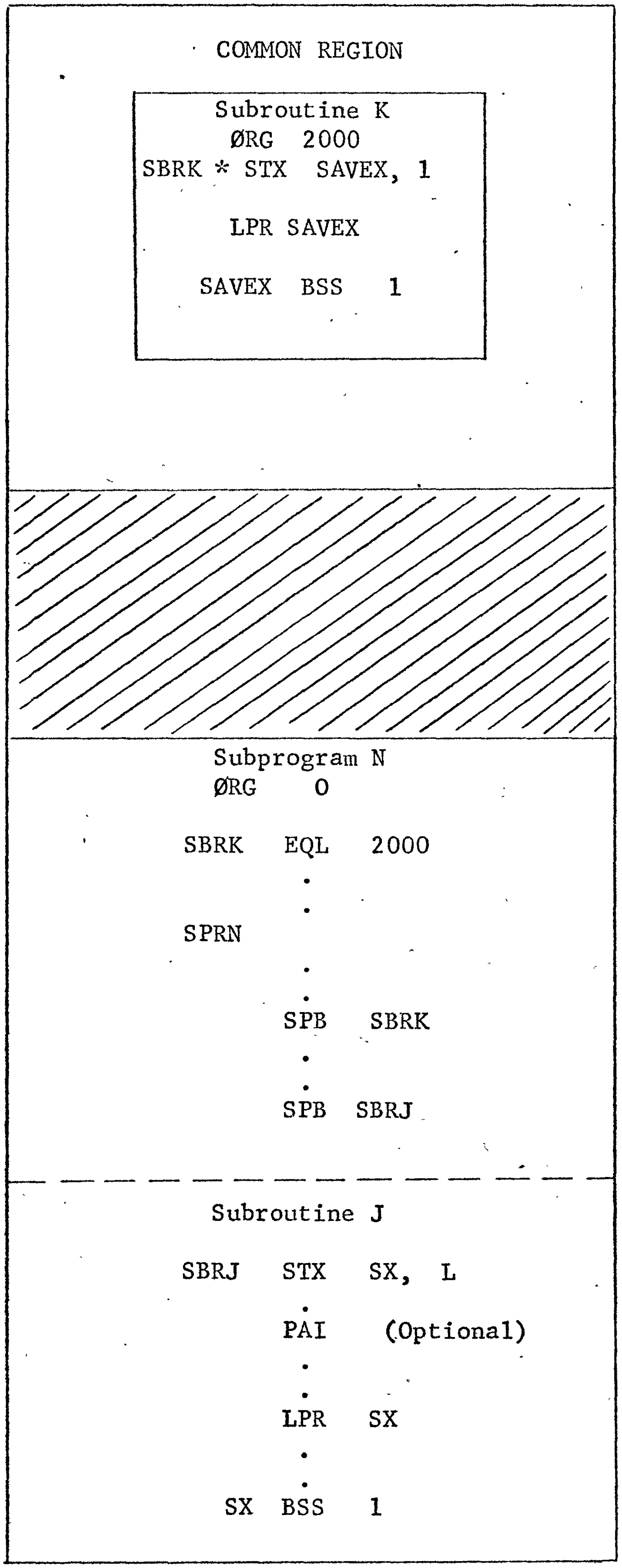
SPB saves the status[6] (interrupt, test, overflow, and program counter) of a main program, inhibits interrupt, and transfers program control as directed by the operand address. Return from a subroutine to the main program is implemented by the LDP instruction which restores part and by LPR which restores all of the saved status.

Subroutine linkage to a subroutine located absolutely within the COMMON are or within (assembled with) the same subprogram is accomplished directly by the SPB instruction. Subroutine linkage to a subroutine located without (separately assembled disjoint subprogram) is accomplished indirectly via a request to the system ECP. Figure 3 illustrates these techniques.

[4] If relative addressing is specified, this is the position address (*±Y) MODULO memory size.

[5] The following example is cited:
Assume the indicated index word contains 3101_{10}. The arithmetic unit will interpret the address "-1" as being 16383_{10}. The effective operand address is then (19484_{10}) MODULO memory size. If the memory size is less than or equal to 16K the effective operand address is 3100_{10}. If the memory size exceeds 16K, the effective operand address will be 19484_{10}.

[6] SPB also saves the Trapping Mode Status if the memory protection option is present. This option available only in GE/PAC 4060.



```
     ØRG  2000
SBRK * STX  SAVEX, 1

    LPR SAVEX

  SAVEX  BSS   1
```

```
ØRG    0

SBRK   EQL   2000
        .
        .
SPRN
        .
        .
       SPB   SBRK
        .
        .
       SPB  SBRJ
```

```
SBRJ   STX   SX,  L
        .
       PAI    (Optional)
        .
        .
       LPR   SX
        .
        .
  SX  BSS    1
```

Figure 3 - SUBROUTINE LINKAGE (MONITOR SPECIFICATIONS)

In a real-time process system program X-word 1 must always be saved at the beginning of a subroutine used, that is, the first instruction executed after an SPB must be an STX. If program interrupts are not used, and the subroutine does not contain a Quasi instruction, this rule is not applicable.

Subroutines in the COMMON region must operate in the program interrupt inhibited mode. Subroutines within a subprogram may optionally operate in the program interrupt permitted mode.

If a subroutine calling sequence contains parameters, X-word 2 may be used by the subroutine to access these parameters providing the access is made prior to the execution of any Quasi or PAI instruction.

```
Calling Sequence          Subroutine

SPB   SBR                 STX   02, 1
LDX   Y1,3                STA   SA
LDA   T1,3                STX   SX3,3
   (Error Return)         XEC   0,2         1st Parameter
   (Normal Return)        XEC   1,2         2nd Parameter
                          INX   3,2         Set Normal Return
                          STX   SX,2
```

1.2.5 PROGRAM SWITCHES

1. THE COMPUTED GO-TO SWITCH

This section describes the GE/PAC coding necessary to implement the FORTRAN Computed GO-TO Switch.

GO TO $(N_1, N_2, N_3 \ldots\ldots\ldots N_n)$,I

The N_i are program labels (locations). The value I is a fixed point integral variable and may be assigned or computed by the object program.

Typical coding:

```
LDA     I1          I = I1 / I2
DVD     I2
STQ     I
 .
 .
 .
LDX     I,3         Go To (N1, N2, N3,...NN),I
BRU     *,3
BRU     N1
BRU     N2
BRU     N3
 .
 .
BRU     NN
```

2. THE ASSIGNED GO-TO SWITCH

The ASSIGNED GO-TO requires careful study; it is a ready source of program error in a relative addressing computer.

GO TO I, $(N_1, N_2, N_3 \ldots\ldots N_n)$

In the previous section, I was an index value; here it is one of the n labels N_k.

The natural way of coding this switch on a non-relative machine is as follows:

```
A   LDA     N+K
    STA     I
     .
     .
     .
N   BRU     I   (or XEC I)
    BRU     N1
    BRU     N2
     .
     .
     .
    BRU     NN
```

The preceding coding will not work on a relative addressing computer. The instruction BRU NK is assembled relative to N+K; its operand address is NK - (N+K). If executed in location I, it transfers program control to location NK - (N+K) + I instead of location NK.

The ASSIGNED GO TO Program switch should be implemented by the computed switch technique. Thus:

```
A   LXK   K, 2 \  or  / LDK  K
    STX   I, 2 /      \ STA  I

N   LDX   I, 2
    BRU   *, 2    or    XEC*,2
    BRU   N1
    BRU   N2
     .
     .
     .
    BRU   NN
```

1.2.6 TESTS AND THE CONDITIONAL BRANCH BTS

GE/PAC has two conditional branch instructions, BTR and BTS, which conditionally branch on the status (set/reset) of a Test Flip-Flop (TSTF). With the exception of the test for arithmetic overflow, BTR and BTS are used for all conditional branches based upon internal effect tests. The TSTF serves as a memory element to remember the result of a previous test and will retain this status until changed by program. Since its status is not destroyed by BTR or BTS, many branches may be made upon the result of a single test. The status of the TSTF is saved by SPB and can be restored by LPR.

Two types of testing instructions are available to the programmer:

1. The first type uses the letter T as the first character of its mnemonic. It places the result of the test (true/false) into the TSTF.

2. The second type uses the letter S or R as the first character of its mnemonic. It affects (sets/resets) the TSTF only when the test is true.

Type T tests are ordinarily used for conventional decision making. Type S or R tests are most useful in the evaluation of logic equation (refer to paragraph 1.3.3 for an example).

Three program control instructions (STF, RTF, LPR) are available to set the TSTF to a predetermined status.

1.2.7 INDEX AND LOOP CONTROL

Two GE/PAC instructions, INX and DMT are used for index control TXH and DMT are used for end of loop testing. Figure 4 illustrates three common loop control examples. The first two examples describe backwards loop control; the third example describes forward loop control. Note that the Loop Counter Word I in example 1 may be either an index word or an arbitrary memory location.

1.2.8 QUASI INSTRUCTIONS

A Quasi instruction is (from the hardware viewpoint) a special two-address instruction. Its octal specifies a fixed address in a branch (SPB) vector leading to a subroutine. Its Y address is a normal operand address. The computer computes the effective operand address from the Y, *, and X fields of the instruction and saves this address in index word two. It then executes the SPB instruction in the location specified by the instruction octal.

FORTRAN

Example 1

```
  00 I = 49
  01    .
        .
        .
        .
        .
     I = I - 1
     IF  (I) 02, 01, 01
  02
  03
```

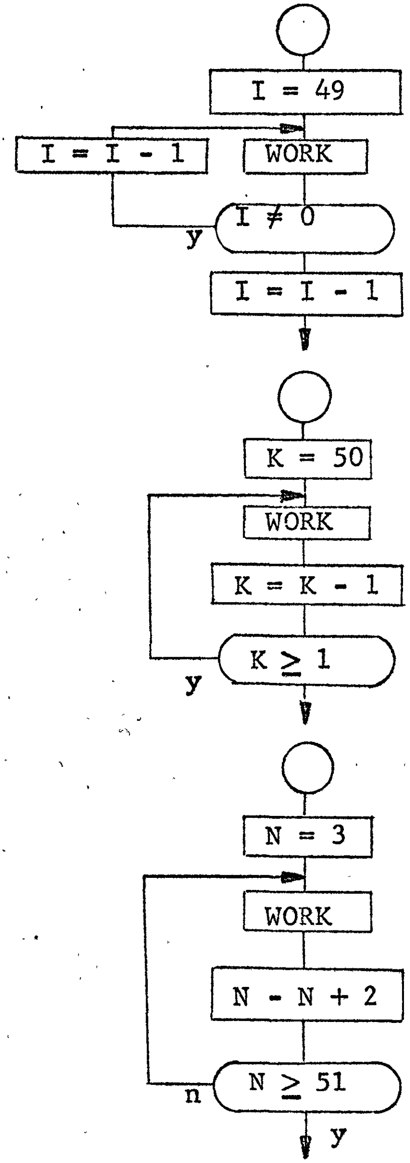
Example 2

```
  10 K = 50
  11    .
        .
        .
        .
        .
     K = K - 1
  12 IF (K) 12, 12, 11

  13
```

Example 3

```
  20 DO  22 N = 3, 51, 2
          .
          .
          .
          .
  22 CONTINUE
```

FLOW CHART



SYMBOLIC PROGRAM

```
          LDA  D49
          STA  I
P01         .
            .
            .
          DMT  I
          BTS  P01


          LXK  50,K
P11         .
            .
            .
            .
          INX  -1,K
          TXH  1,K
          BTS  P11


          LXK  3,N
P21



          TXH  51,N
          BTS  *+3
          INX  +2,N
          BRU  P21
```

Figure 4 - LOOP CONTROL EXAMPLES

A Quasi instruction is one which resembles, or is used as an instruction executed by a group of commands. For ëxample, the Floating Add (FAD), which is implemented by a subroutine, is used by the programmer the same as a hardware implemented FAD instruction. Therefore, a Quasi executes programs in memory allowing the programmer to demand an instruction not in the hardware. Several benefits are as follows:

1. A more effective instruction repertoire permits easier coding.
2. Upward program compatibility is possible.
3. This technique reduces memory requirements of programs when compared to the use of conventional subroutine techniques.

Not all functions are best implemented as Quasi's. Square root, which normally uses the A-Register as its operand, does not require a two-address subroutine link. Therefore, to implement it by a Quasi wastes the Y operand address.

1.2.9 XEC AND ITS USAGE

The address portion of an Execute instruction (XEC) specifies an object instruction to be executed. It does not set the location counter to the location of the object instruction as would a branch instruction. Thus, XEC calls a one-instruction subroutine and specifies immediate return to the main routine.

The use of XEC operation arises directly from the fact that the object instruction does not imply its own successor unless it is a branch. The XEC simplifies modification of non-indexable instructions (such as index control instructions) as well as providing the ability to effectively modify programs (remotely) which may not be directly modified. Effective use may also be made of the XEC in the case of a subroutine calling sequences where the calling sequence to the subroutine may include several parameters specified in actual machine instructions which the subroutine treats as second-order one-word subroutines.

XEC completes a set of four program control operations:

1. Program control is retained by the main program (normal instruction sequencing).
2. Program control is given to another program (branching).
3. Program control is usurped by another program (program interrupt).
4. Program control is lent to another (one instruction) program (execute).

Several examples of XEC usage are:

1. Indexing non-indexable instructions

```
    .                .             .
    .                .             .
    .                .             .
    XEC  KL, J  XEC  KI, J        XEC
    .                .             .
    .                .             .
    .                .             .
KL  LXK  3,K    KI   INX  2,K   KT   TXH  25,K
    LXK  7,K         INX  3,K        TXH  28,K
```

2. Program switches -- (Described in paragraph 1.3.5)
3. Double indexing arrays

```
A = TABLE (I,J)              XEC  LT,  I
                             STA  A
                              .
                              .
                              .
                         LT  LDA  TABLE, J
                             LDA  TABLE+JM, J
                             LDA  TABLE +2JM, J
                             LDA  TABLE +3JM, J
```

4. Executing "manufactured" instructions. When it becomes necessary to modify an instruction (other than by indexing), it must be placed in COMMON and be executed via XEC. Extreme care must be exercised when manufacturing memory addressing instructions.

5. When several programs are identical, except for one instruction, XEC can reduce overall storage requirements by combining the programs.

```
    LXK   I,J              PROG  STX  SX,   1
                                  .
    SPB   PROG                    .
          .                       .
          .                       .
          .                       .
          .                       .
          .                      XEC, INS,  J
                                  .
                                  .
                                 LPR  SX
                            INS  ADD  NUM
                                 SUB  NUM
                                 DVD  NUM
                                  .
                                  .
                                  .
```

1.2.10 PROGRAM INTERRUPT

Program interrupt is the only satisfactory hardware method to synchronize a computer program with external conditions.

An external event may be sensed and a signal (indicating event true/event false) is connected to the GE/PAC Automatic Program Interrupt module. Two types of event detection are provided by the GE/PAC API module:

1. The event has occurred (was and/or is true) and a program interrupt, to acknowledge the occurrence, has not yet occurred.
2. The event is presently occurring (was and is true) and a program interrupt, to acknowledge the occurrence, has not yet occurred.

In either case, the GE/PAC API module remembers the event until a program interrupt acknowledges its occurrence. The GE/PAC API will then "forget" the event.

Type 1 event detection is normally used to provide event recording, plus accumulation, and process-to-program synchronization. Type 2 event detection controls interrupt driven I/O equipment.

Two classes of program interrupt (relative to program control) are provided in GE/PAC systems:

1. Inhibitable Interrupts

   The program controls the occurrence of such interrupts. Execution of an IAI or SPB instruction places the computer in the program interrupt inhibited mode. Inhibitable interrupts are delayed until a PAI instruction is executed to place the computer in the program interrupt permitted mode. The instructions LDP and LPR will either inhibit or permit interrupt depending upon Bit 21 of the referenced memory location.

2. Noninhibitable Interrupts

   The program does not normally control the occurrence of such interrupts. A program interrupt which acknowledges the occurrence of the event is made within 1 to 500 microseconds of the occurrence of the event.

To assure program integrity during program-to-subroutine communication, a hardware restriction must be placed upon the SPB instruction to guarantee program execution of one additional instruction (an STX) prior to a possible noninhibitable program interrupt. A similar restriction upon the BRU instruction is necessary to simplify FORTRAN program-to-subprogram communication. These restrictions may "hang up" the program by locking out all interrupts if an SPB * or BRU * is executed. BTS * and XEC * are other non-interruptable program stops. None of these stops, however, are normally acceptable in a real-time program.

Program interrupt may not occur more frequently than every other instruction; the instruction succeeding the interrupt-inserted instruction is determined by the program counter.

If an LDP or LPR (to permit interrupt) is executed, one additional instruction will be executed before an inhibitable interrupt occurs. A non-inhibitable interrupt may occur immediately following the LDP or LPR.

The program sequence PAI, IAI will not permit any requested program interrupts to occur. The sequence PAI, NØP, IAI allows the highest priority requested interrupt to occur, etc.

The program sequence PAI, NØP, BRU * -1 will not necessarily guarantee the servicing of all requested program interrupts. This sequence should be written as PAI, NØP, NØP, BRU * -2.

A program interrupt is accomplished by executing one instruction out-of-normal program sequence. Sequential memory locations in an Interrupt Control Table are reserved for this purpose, corresponding to Interrupts #0, #1, ... The Interrupt Control Table starts at location 100_8 in 4040 and 4050/4060 systems; starting location 200_8 in 4020 systems. Although the only theoretical limitation to the number of interrupts is memory size, available "off the shelf" hardware is limited to a maximum of 64 interrupts.

Only six GE/PAC instructions should be used within the Interrupt Control Table. Other instructions will cause "program bugs" which are extremely elusive to find. Relative addressing <u>must not</u> be used in these interrupt control instructions. Permissable GE/PAC instructions are DMT, SPB, NØP, BRU, ØLD, and IDL.

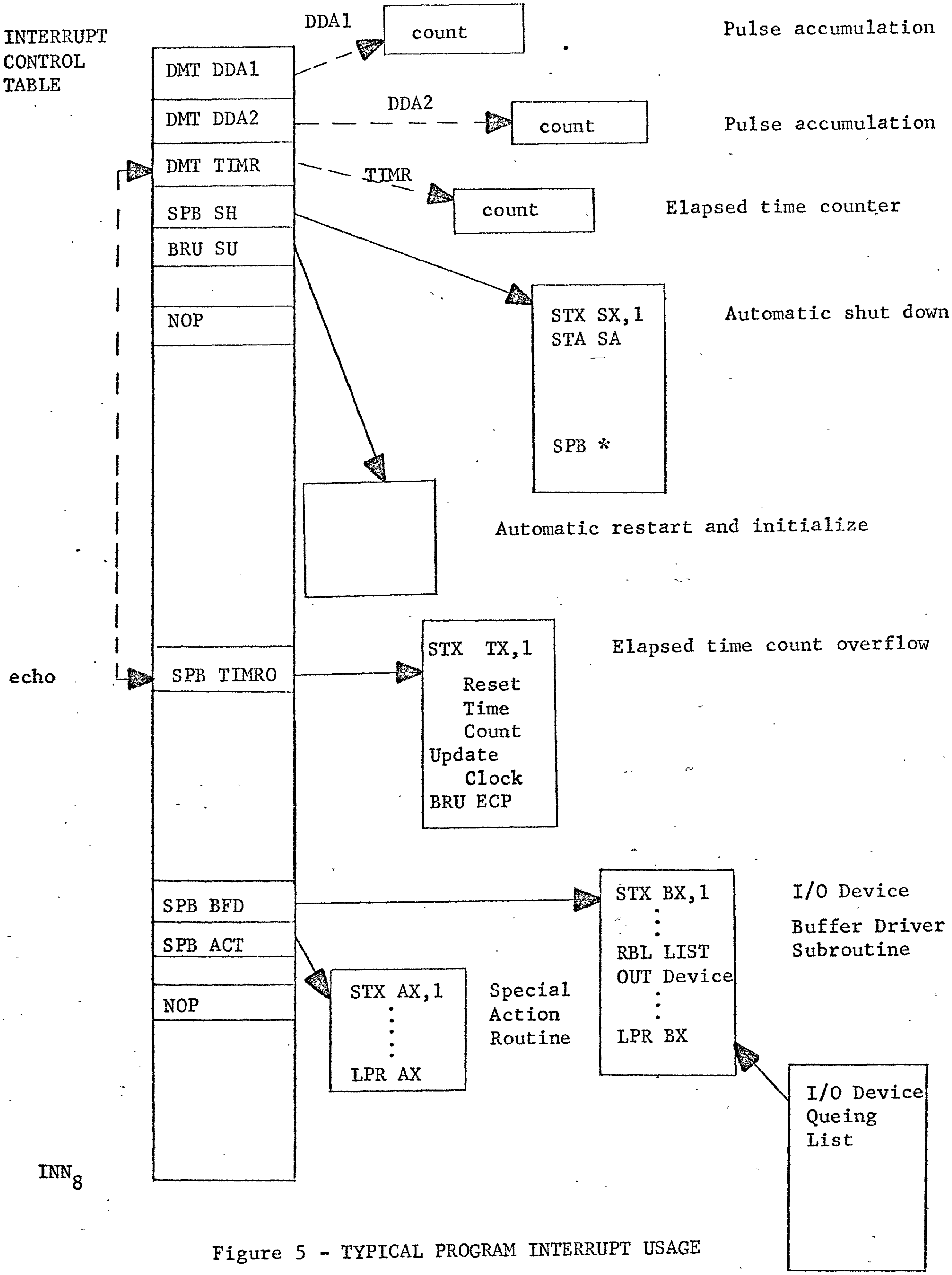


Figure 5 - TYPICAL PROGRAM INTERRUPT USAGE

1.3 DATA MANIPULATION

The "bit" is the fundamental unit of data. A larger and more convenient unit is the "word" which, in GE/PAC, consists of an ordered set of 24 "bits". GE/PAC is word-organized with respect to addressing and arithmetic; it is both word-organized and bit-organized with respect to logic.

Data Manipulation is accomplished in one or more of the following registers:

1. A-Register
2. Q-Register
3. Addressed memory location Z

The group of General instructions provide the means to load or store each register. Arithmetic, Logic, and Test instructions implement data manipulation operations.

The A-Register, being the primary working register, is affected by a majority of the data manipulation operations. The Q-Register is used as an extension of the A-Register or as an auxiliary register. Few operations affect it. The Operate on Memory instruction (ØØM) provides a means of applying all of the A-Register operations to an addressed memory location.

1.3.1 DATA REPRESENTATION

Data representation is provided for two types of data, numeric variables and logic variables.

1.3.1.1 Numeric variables

Numeric variables are represented by sequences of digits. The GE/PAC word, with bits interpreted as digits, provides binary representation of any real number with a precision of 23 binary digits.

Three common binary representations for negative numbers are:

1. Sign plus absolute value
2. One's complement
3. Two's complement

In each case the left-most bit is interpreted as the sign of the number, 0 meaning plus and 1 meaning minus.

Two's complement representation is used in GE/PAC for fixed-point arithmetic.

Sign plus absolute value representation is used in GE/PAC for floating-point arithmetic.

The integral and fractional portions of the number are separated by a binary point. The binary point has no analogy in hardware. In fixed-point arithmetic, the binary point is supplied mentally by the programmer and is referred to as the scale of the number. In floating-point arithmetic, the binary point is defined by an exponent within the floating-point number representation; the mantissa is always a normalized fraction. Arithmetic data formats are illustrated in Figure 6. Figure 7 presents the number range of fixed- and floating-point variables.

Note on Numeric Data Representation

Any real number may be represented in binary notation as:

$$\pm \ldots + X_n 2^n + X_{n-1} 2^{n-1} + . + X_1 2 + X_0 + X_{-1} 2^{-1} + .+$$

$$X_{-n} 2^{-n} + \ldots$$ where the X_i are the coefficients of the

powers of 2 and may assume the values 0 and 1. If a one-to-one correspondence between these X_i and bits in computer words is made, numbers may be represented by sequences of bits. The 24-bit word provides sequence lengths of 24 and 48 bits and can approximate any real number with a precision of 23 (46) bits.

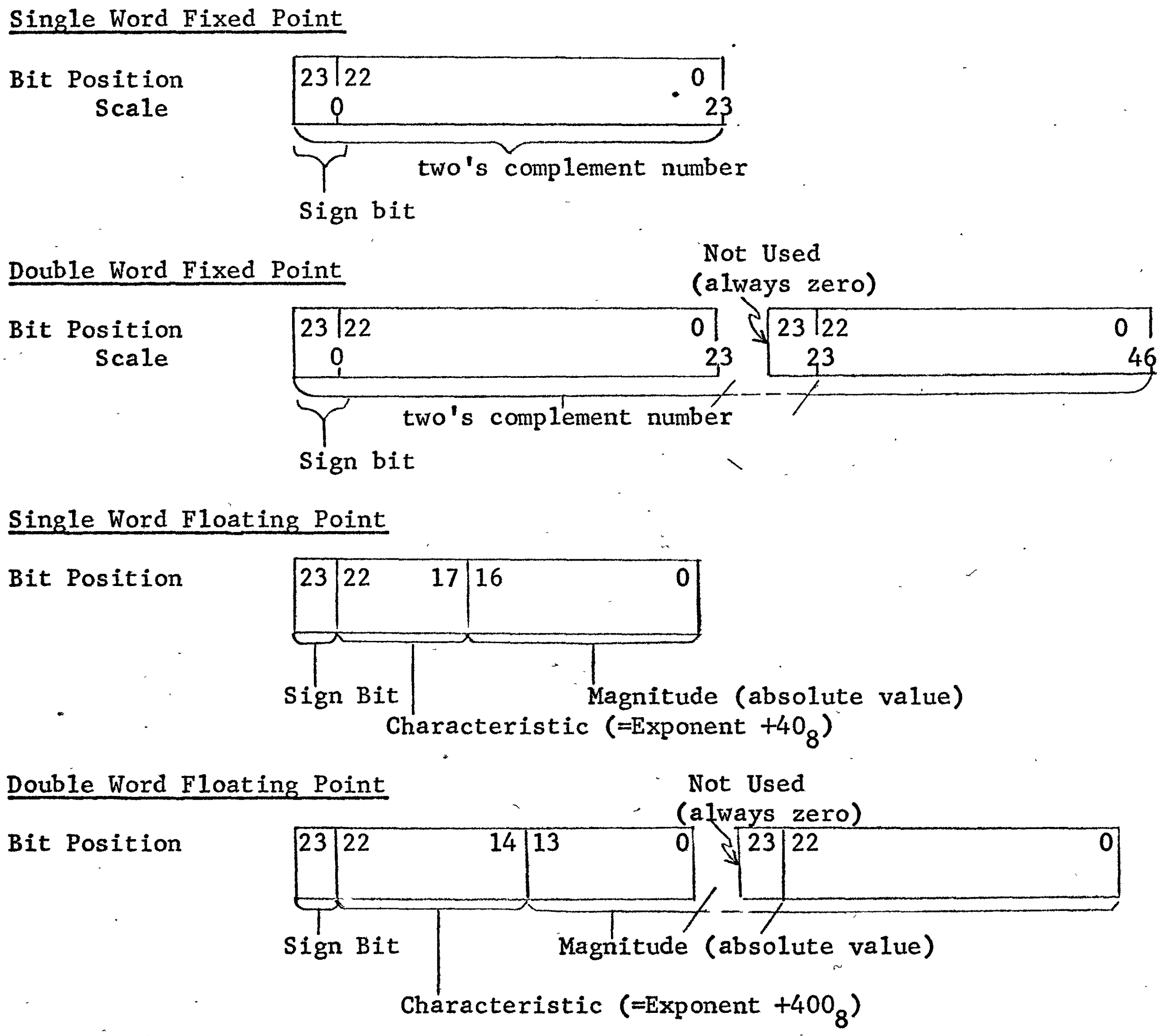
The two's complement of a number $S_{n+23} \ldots X_{n+1} X_n$ is numerically equal to the number

$$[(1_{n+24} 0_{n+23} \ldots 0) - (0_{n+24} S_{n+23} X_{n+22} \ldots X_n)]$$

Thus $-7_{10} = 10000_2 - 00111_2 = 1001_2$,

$-(-1)_{10} = 10000_2 - 01111_2 = 0001_2$,

and $-0 = 10000_2 - 00000_2 = 0000_2$



Single Word Fixed Point

Double Word Fixed Point

Single Word Floating Point

Bit Position
23
22
17
16
0
Sign Bit
Magnitude (absolute value)
Characteristic (=Exponent +40$_8$)

Double Word Floating Point

Not Used
(always zero)
Bit Position
23
22
14
13
0
23
22
0
Sign Bit
Magnitude (absolute value)
Characteristic (=Exponent +400$_8$)

(NOTE: Sign bit 0 = +; 1 = -)

Figure 6 - ARITHMETIC DATA FORMATS

FIXED POINT NUMBER RANGE

Format	Minimum Number	Maximum Number	Increment
Single Word			
As found in Memory	40000000_8	37777777_8	00000001_8
Decimal equivalent	-2^s	2^s-1	2^{s-23}
Double Word			
As found in Memory	40000000 00000000_8	37777777 37777777_8	00000000 00000001_8
Decimal equivalent	-2^s	2^s-1	2^{s-47}

Notes a. s = scale of fixed point number.

b. $2^s = 10^{0.30s}$, approximately.

c. Although the negative endpoint, -2^s, can be represented, its negative 2^s, can not. Therefore -2^s is not normally considered to be part of the range of a numeric variable.

FLOATING POINT NUMBER RANGE

Format	Minimum Number	Maximum Number	Increment
Single Word			
As found in Memory	77777777_8	37777777_8	$0C000001_8$
Approximate decimal equivalent.	$-0.00000*10^9$	$+0.99999*10^9$	2^{C-49}
Double Word			
As found in Memory	77777777 37777777_8	37777777 37777777_8	0CC00000 00000001_8
Approximate decimal equivalent.	-0.999,999 $999,99*10^{77}$	+0.999,999 $999,99*10^{77}$	2^{CC-69}

Notes a. The number "zero" is arbitrarily defined to be the number 00000000_8 ($00000000,00000000_8$ for double word format) as it would occur in memory, and has the approximate decimal equivalent $+0.0*10^{-9}$ ($+0.0*10^{-77}$, double word).

b. The number "minus zero", which is representable in "sign plus absolute value" notation can never result from a GE/PAC floating point operation and should be considered illegal.

c. C = characteristic of floating number. Single-word range of C is $0 \leq C \leq 63$; double-word range of CC is $0 \leq CC \leq 511$.

d. The increment is not a representable floating number. The minimum positive representable number is 00200000 and has the approximate decimal equivalent $+0.5 * 10^{-9}$. The minimum double-word representable number is $02000000,00000000_8$ having the decimal equivalent $+0.5*10^{-77}$.

Figure 7-FIXED AND FLOATING POINT NUMBER RANGE

An alternate method to negate a number is to form the one's complement and add one.

$$\text{Thus } -7_{10} = \overline{0111}_2 + 0001_2 = 1000_2 + 0001_2 = 1001_2,$$

$$-(-1) = \overline{1111}_2 + 0001_2 = 000_2 + 0001_2 = 0001_2,$$

$$\text{and } -0_{10} = \overline{0000}_2 = 0001_2 = 1111_2 + 0001_2 = 0000_2.$$

This latter method is used in GE/PAC since:

1. The former method requires an additonal bit in the A-Register.
2. The latter method treats subtraction as a special case of addition.

Another way to interpret a two's complement number is to think of the sign bit as being the coefficient of a negative multiplier. All other multipliers are positive.

$$\text{Thus } -7 = 1001_2 = -8_{10} + 0_{10} + 0_{10} + 0_{10} + 1_{10},$$

$$1_{10} = 0001_2 = + 0_{10} + 0_{10} + 0_{10} + 1_{10},$$

$$\text{and } -2_{10} = 1110_2 = -8_{10} + 4_{10} + 2_{10} + 0_{10}.$$

1.3.1.2 Logic Variables

Yes/no, true/false, on/off, and set/reset conditions are represented by logic variables and obey the laws of Boolean algebra. If a one-to-one correspondence between logic variables and bits are made, a set of twenty-four such variables may be represented by a GE/PAC computer word. In such a representation, 1 means true and 0 means false.

GE/PAC provides word logical instructions to operate on sets of bits, and bit logical instructions to operate on individual bits.

1.3.2 ARITHMETIC OPERATIONS

1.3.2.1 Arithmetic Overflow

The result of an arithmetic operation (addition, subtraction, multiplication, or division) upon two 24-bit numbers may exceed 24 bits. Any further computation involving the result is meaningless. Arithmetic overflow occurs when the magnitude of the resulting number is too large. GE/PAC provides arithmetic overflow detection for all of its arithmetic instructions except ADØ and NEG.

Arithmetic underflow occurs when a result is too small in magnitude to affect the least significant bit of the register in which the operation is performed. When underflow occurs, the result is replaced by a zero but no underflow detection is provided.

If overflow occurred during the execution of an ADD, SUB, AKA, SKA, MPY, DVD, SLA, DLA, FIX, FLØ, FAD, FSU, FMP, or FDV, the Overflow Flip-Flop (OVRF) is set. Otherwise, the OVRF remains unchanged. Testing overflow with the JNØ instruction after each arithmetic instruction is inefficient. This test need be given only once (at the end of the computation) to provide valid arithmetic overflow detection.

A carry out of the A-Register and arithmetic overflow are unrelated in two's complement fixed point arithmetic. Such a carry is neither a necessary nor sufficient condition for arithmetic overflow. Each arithmetic operation has a different criterium for determining whether or not overflow occurred. These criteria are:

ADD

If the contents of A and Z have like signs, and the sign of the sum differs from that of its arguments, the OVRF is set; otherwise the OVRF is unaffected.

SUB

If the contents of A and the one's complement of the contents of Z have like signs and the sign of the difference differs from that of its arguments, the OVRF is set; otherwise the OVRF is unaffected.

MPY

	REGISTER			ARITHMETIC OVERFLOW
	Z	Q	A	
VALUE	M	M	0	yes
	M	M	0	no
	(Z)	(Q)	(A)	no

M in the above table is the octal number 40000000. It is an illegal number in that this maximum negative number is its own negative. That is, its negative is not positive in the accepted sense because the so-called sign bit is a zero. Overflow occurs during MPY only if the multiplier and multiplicand are 40000000_8 and the contents of the A-Register are not negative.

If arithmetic overflow occurs as a result of MPY, the OVRF is set; otherwise the OVRF is unchanged.

DVD

If the magnitude of the contents of Z exceeds that of the contents of A, the OVRF is unaffected. If the magnitude of A exceeds that of Z, the OVRF is set. If the magnitude of A equals that of Z and the sign of the resulting remainder is the same as the sign of the divisor, the OVRF is unaffected; if these signs differ, the OVRF is set.

SLA and DLA

If the contents of A are positive: If any "1" bit is shifted into A_{23}, the OVRF is set; otherwise the OVRF is unchanged. If the contents of A are negative: If any "0" bit is shifted into A_{23}, the OVRF is set; otherwise the OVRF is unchanged.

Floating-Point Instructions

Arithmetic overflow occurs when the magnitude of the exponent of the result is too large for the exponent field of the floating-point format.

Single Word Mode

If the magnitude of the result of a floating operation is less than 1 X 2^{-33}, it is replaced by zero; if the magnitude is greater than 1 X 2^{31}, it is replaced by the signed maximum floating number and the OVRF is set.

Double Word Mode

If the magnitude of the result of a floating operation is less than 1 X 2^{-513}, it is replaced by zero; if the magnitude is greater than 1 X 2^{511}, it is replaced by the signed maximum floating number and the OVRF is set.

1.3.2.2 Fixed Point Arithmetic Scaling Operations

The term scaling describes the analysis process associated with the programming of numerical problems using fixed-point arithmetic operations are subject to the following rules:

Addition and Subtraction

Variables must be scaled alike when added or subtracted. If scales differ, each variable is scaled by multiplying it by an appropriate power of two so the resulting scales are the same. The scale of the result is that of its addends.

Multiplication

Variables may have different scales when multiplied. The scale of the result is equal to the sum of the scales of its factors.

Division

Variables may have different scales when divided. The scale of the quotient is equal that of the dividend, diminished by the scale of the divisor. The scale of the remainder is equal to the scale of the dividend, diminished by 23.

Scaling may be accomplished in one or both of two ways:

1. Mental multiplication by a power of 2.
2. Actual multiplication by a power of 2 as implemented by a shift (SRA, SLA).

Most precise results are obtained when scaling is planned so the number of significant bits in the working registers A and Q is maximized at all times.

1.3.2.3 Floating-Point Arithmetic Scaling Operations

Floating-point arithmetic minimizes or eliminates the need for scaling. Floating-point Scaling consists of analyzing all equations comprising a problem, and reformulating if necessary, to insure all variables (including results and partial results) are representable by the floating-point format.

1.3.3 LOGIC AND BIT LOGIC OPERATIONS

GE/PAC has conventional logical instructions (CPL, ØPRA, ANA, ERA) which address memory and operate on the addressed 24-bit words. In addition, a GEN1 class of instruction allows the programmer to address individual bits of a word in the A-Register and perform the CPL, ØRA, ANA and ERA operations on the addressed bits. Thus:

SBK	K	Sets the Kth bit of A to a one
CBK	K	Changes the Kth bit of A
TØD	K	Tests the Kth bit of A

(Also, others are available)

Much of a control program consists of decision making based upon the status of true/false bit-logic variables. These conditions are normally read into the computer via a Digital Input Scanner.

Assume that one step in a start-up process consists of determining that one of two pumps is on with its valve open and that a main valve is open.

A = Motor on, pump #1
B = Valve open, pump #1
C = Motor on, pump #2
D = Valve open, pump #2
E = Main valve open

The programming technique does not depend upon the arrangement of variables; all <u>may</u> be assumed in the same word without loss of generality. The step may be expressed in the form of a logic equation:

If $(A * B * \overline{C} * \overline{D} + \overline{A} * \overline{B} * C * D) * E = 1$, GØ TØ ØK

GE/PAC coding for this equation is given in Figure 9.

A shorter program can be produced by using GE/PAC conventional logic instructions (CPL, ØRA, ANA, ERA) if rigid assumptions, with respect to the location and arrangement of bit variables, are made. When using the logic instructions, all five variables must be assumed to be in sequence in the same digital word. GE/PAC coding for this equation is presented in Figure 10.

A comparison of the two methods emphasizes the value of bit logic.

```
          LDA GRØUP           00
          TØD BIT A                 T = A
          BTR *+5                   IF (T-1) 02,01
          REV BIT B           01    T = A*B*-C*-D
          RØD BIT C
          RØD BIT D
          BRU *+5                   GØ TØ 03
          SET                 02    T=1
          RØD BIT B                 T=A*B*-C*-1
          REV BIT C
          REV BIT D
          REV BIT E           03    T=T*E
          BTS ØK                    IF (T-1) 04,ØK
          BRU ALARM           04    GØ TØ ALARM


*             USE OF GEPAC BIT LOGIC INSTRUCTIONS
                   TO EVALUATE LOGIC EQUATION
              IF (A*B*-C*-D + -A*-B*C*D) * E = 1, GØ TØ OK
```

Figure 8 - BIT LOGIC USAGE

```
          LDA GRØUP                 EABCD---------
          SRL 1                     ØEABCD--------
          ERA GRØUP                  1-010--------- IF ØK
          ERA MASK1
          ANA MASK2
          TZE
          BTS ØK
          BRU ALARM
*                   BITA MUST BE BIT 22
*                   BITB MUST BE BIT 21
*                   BITC MUST BE BIT 20
*                   BITD MUST BE BIT 19
*                   BITE MUST BE BIT 23
MASK1     CØN /44000000             100100000....
MASK2     CØN /56000000             101110000....


*             USE OF GEPAC CONVENTIONAL LOGIC INSTRUCTIONS
*                  TO EVALUATE LOGIC EQUATION
*             IF (A*B*-C*-D + -A*-B*C*D) * E = 1, GØ TØ OK


*
```

Figure 9 - CONVENTIONAL LOGIC USAGE

Bit logic method advantages:

a. Permits a natural and straight-forward algorithmic evaluation of logic equations:

   1. Suited for use by junior programmers without loss in quality of coding
   2. Suited for use in a compiler
   3. To reduce programming time

b. Permits symbolic representation of bit variables. Minimizes the need for programming dictation of the specific arrangement of inputs to the Digital Input Scanner.

Disadvantages:

a. Requires more memory and execution time

Conventional logic method advantages:

a. Requires less memory and execution time if optimumally coded

Disadvantages:

a. Requires ingenuity in choice of specific technique and bit arrangement, therefore:

   1. Quality of coding is dependent upon the caliber of the programmer. Non-optimum coding can require more memory and execution time than produced by the bit logic method.
   2. Not suited for use in a compiler
   3. Increased programming time required

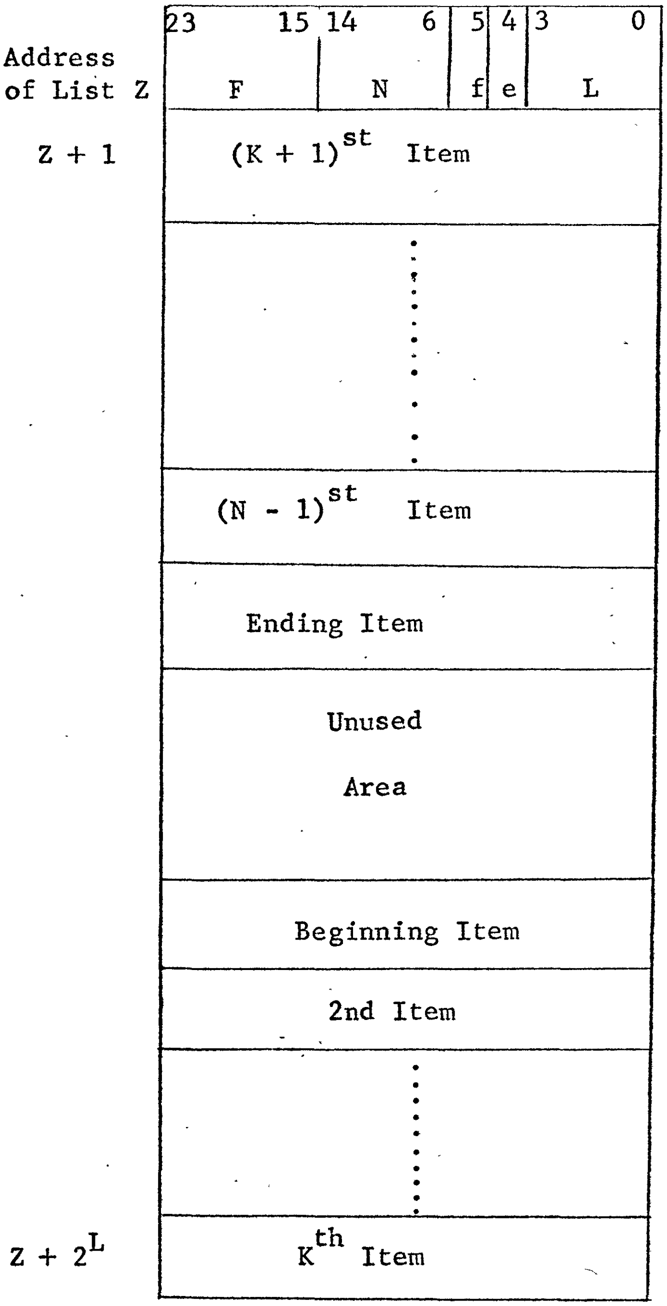
b. Symbolic representation of contact closures is very awkward and requires restrictions on hardware grouping of variables.

c. A bit arrangement that is optimum for one test is probably not optimum for another test in the same program involving the same set of variables.

1.3.4 LISTS AND THEIR USAGE

1.3.4.1 Definitions

A List consists of an ordered set of items contained within a fixed length block of memory ($2^L + 1$ words where L is an integer, $1 \leq L \leq 9$).



LIST CONTROL WORD

L Specifies maximum size of list (2^L items where $1 \leq L \leq 9$).

F Specifies location of beginning item ($0 \leq F \leq 2^L - 1$).

N Specifies number of items currently in list ($0 \leq N \leq 2^L$)

Prior to Execution:

Address of beginning item
$Z + 1 + (F+1) \text{ MOD} 2^L$

Address of ending item
$Z + 1 + (F + N) \text{ MOD} 2^L$

The list is empty if N=0 and e=1

The list is full if N=0 and f=1

Figure 10 - GE/PAC LIST ADDRESSING DETAILS

The first word of the block is a list control word which is the address or label of the list. The maximum size of any given list is 2^L items. The physical ordering of a list is circular in nature; the address 2^L Relative to the list control word is followed sequentially by the relative address 1. The terms beginning item and ending item are arbitrarily attached to the first and last members of the set of items currently forming the list. Figure 10 illustrates list addressing details. Instructions are provided for appending additional items to the beginning or end of the list and for removing the beginning or ending items.

These instructions are:

ABL	Z	Append item to beginning of list Z
AEL	Z	Append item to end of list Z
RBL	Z	Remove beginning item from list Z
REL	Z	Remove ending item from list Z

An item, when appended, becomes the new beginning/ending item. The removal of an item forces the adjacent item to become the new beginning/ending item. The current beginning and ending items are the <u>only</u> items addressable by program; normal table indexing techniques allow addressing of any entry in a table.

1.3.4.2 The Queuing List and Its Usage

A queuing list channels information to and through Input/Output devices. Assume a request for the on demand print out of the current value of an analog input is made. The scan command word and appropriate control information would be appended to a queue controlling the scanner operation. When available, the raw count value of this input will be appended to a queue awaiting conversion from binary to engineering units.

Another queue is awaiting line to the binary to BCD conversion routine. The resulting ordered set of BCD characters is appended to the output driver queue which controls a typewriter.

ABL places high priority items at the beginning of the list, RBL removes them ahead of all other items. Consequently, high priority items are appended at the beginning of the list to become the next item removed. Thus AEL and RBL provide normal queuing.

1.3.4.3 The Stacking List and Its Usage

The stack or push down list is so named because the most recently appended item is always removed prior to the removal of less recently appended items. The phrase last in - first out is sometimes used to describe the operation of a stack.

Typical stack usage is a common temporary storage block for programs subject to program interrupt. Each program, by appending all of its intermediate results in the stack, could be destroyed by a higher priority program. When all higher priority programs are completed, the destroyed program is restored and continues from its point of interruption.

The stack requires that the relative priorities of all programs having data within the stack do not change. Use of the stack on GE/PAC 4040 is relatively slow. When stacking of data is not satisfactory for a given program, one of the following alternates must be used:

1. The program is assigned unique locations in COMMON for its intermediate results.
2. The program is executed in the interrupt inhibited mode.
3. The program is restarted from its initial conditions.

AEL and REL are stacking instructions.

1.3.5 GEN 1 INSTRUCTIONS

The GEN 1 instruction is executed as follows. Instruction retrieval and indexing occur during timing states 1 and 2. Since indexing address modification affects bits 13 through 0 of the instructions, indexing can change the meaning of the GEN 1 instruction. Implementation of timing state 3 is as follows:

1. The complemented value of k, (I_{4-0}), is placed in the J-Counter.
2. The control fields determine inputs to the serial adder, to the Test Flip-Flop, and to the shifting control.
3. The contents of the A-Register are ringshifted bit by bit through the adder, the output of the adder being shifted into A_{23} of the A-Register.
4. The J-Counter is incremented one for each one-bit shift. The final contents of this counter is:

   a. 37_8 if $\Delta_c = 1$

   b. $27_8 - k$ if $\Delta_c = 0$, $k \leq 23_{10}$

   c. 37_8 if $\Delta_c = 0$, $k = 24_{10}$

1024 unique GEN 1 instruction octals are possible. The most useful octals are utilized and are listed in the GE/PAC 4000 Instruction Reference Manual

1.3.6 GEN 3 INSTRUCTIONS

GEN 3 commands are differentiated from other commands by the op code 45_8 They are also subdivided into commands by the micro-coding of the operand portion of the command word These commands are used to manipulate the contents of the A- and Q-Registers, affect the J-Counter, and optionally affect the Overflow Flip-Flop GEN 3 commands are also used within the Quasi subroutines for floating-point arithmetic operations

1.3.7 COUNT INSTRUCTIONS

CMØ and CMZ instructions count the number of most significant ones/zeros of the data word in the A-Register. CLØ and CLZ count the number of least significant ones/zeros in the A-Register.

These instructions leave the determined "count" value in the J-Counter; an LXC instruction must be executed to transfer it to an index word. Quasi instructions must not occur between a count instruction and the LXC Therefore, each count instruction must be immediately followed by its associated LXC No other conventions apply when using count instructions with GE/PAC 4050 or 4060 computers

Two conventions apply on the GE/PAC 4040 computer:

1 Count instructions are not protected from program interrupt. Certain Quasi instructions use the J-Counter without restoring it. Each count instruction must be immediately followed by its associated LXC; these instructions must be between an IAI and PAI in a permitted program.

```
IAI
CLZ
LXC   X
PAI
```

2. Non-inhibitable program interrupt action routines must save and restore the J-Counter if the action routine contains any GEN 1 instruction.

   The J-Counter is saved with the coding:

```
LXC   X           COUNT → X.
```

   The J-Counter is restored with this coding:

```
LDØ   O,X
CLZ
```

   The A-Register is destroyed by this operation.

Figures 11 and 12 present the flow chart and coding for a typical application which uses a count instruction A simple executive routine examines bits in a computer word, transferring program control to a subroutine as a control bit is set

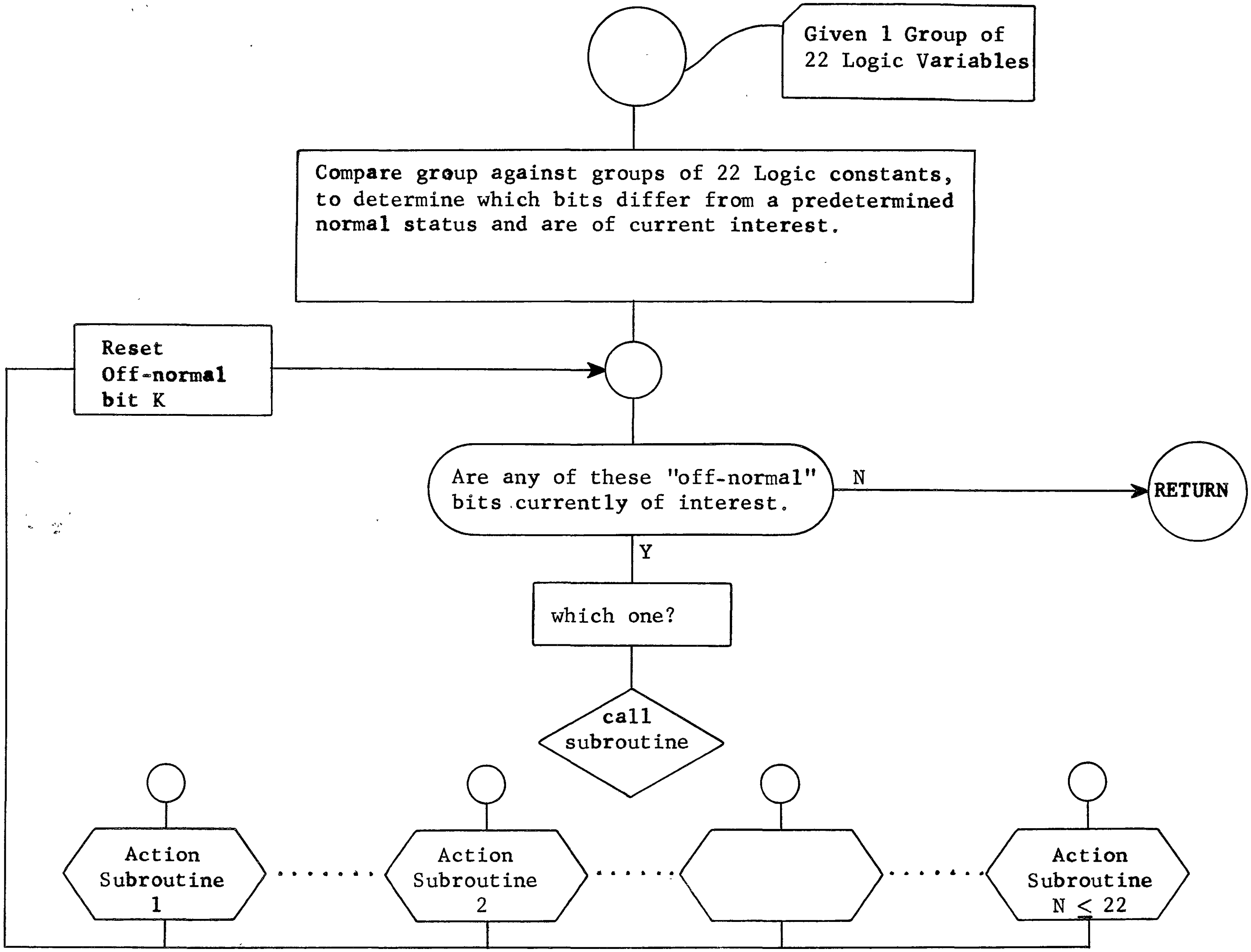

Given 1 Group of
22 Logic Variables
Compare group against groups of 22 Logic constants, to determine which bits differ from a predetermined normal status and are of current interest.
Reset
Off-normal
bit K
Are any of these "off-normal" bits currently of interest.
N
RETURN
Y
which one?
call
subroutine
Action
Subroutine
1
Action
Subroutine
2
Action
Subroutine
N ≤ 22

Figure 11 - COUNT INSTRUCTION APPLICATION

This technique can be used for any or all the following purposes:

1. As the ECP in Monitor
2. As a demand routine ECP
3. As a change of state ECP
   Associated with program interrupt and/or contact status groups read in via the Digital Input Scanner (DIS).

```
ACTA    LDA GRØUP           000    GRØUP ØF 22 LØGIC VARIABLES
        ERA NØRMAL                 DETERMINE ØFF-NØRMAL VARIABLES
        ANA INTRST                 DETERMINE ØFF-NØRM ØF INTERST
ACTB    TZE                 001 B  IF (ANY ØFFNØRMAL OF
        BTS ACTC                X     INTEREST) --, 003, 002
        IAI                 002    DETERMINE WHICH ONE K
        CLZ
        LXC 3
        PAI
        SPB ACTION,3               CALL ACTIONF(K)
        RBK 0,3                    RESET OFFNORMAL BIT K
        BRU ACTB                   GØ TØ 002
ACTC    LPR SX              003    RETURN
NORMAL  BSS 1
INTRST  BSS 1
ACTION  BRU ACT1
        BRU ACT2
         .  .  .
        BRU ACTN
SX      BSS 1
```

Figure 12 - COUNT INSTRUCTION CODING

1.4 INPUT AND OUTPUT CONTROL

1.4.1 GEN 2 INSTRUCTIONS

GEN 2 instructions are microcoded instructions used for control of input/output equipment and for certain computer actions. Instruction format follows:

23——18	17 – 15	14 – 12	11 ———— 0
25	X	S	D

25 - Instruction octal
X - X word indicator
S - Instruction secondary-octal
D - Device address

Device address D = 0000_8 is a ficticious address referring to the computer itself.

The following instructions are provided:

Program Control

25010000	SSA	Set Stall Alarm
25020000	PAI	Permit Automatic Program Interrupt
25030000	IAI	Inhibit Automatic Program Interrupt
25040000	JND	Jump if no Demand
25050000	RCS	Read Console Switches

Validity Tests

25060000	JNØ	Jump if No Overflow
25070000	JNP	Jump if No Core Parity

Device addresses D = $11DD_8$ refer to peripheral devices. Device address D = 2400_8 refers to the GE/PAC drum. Other device addresses may be defined as required.

The following generic instructions are recognized by the assembly program:

2500	SEL	Select device D
2501	ACT	Activate device D
2502	ØPR	Operate device D

2504	ØUT	Output from A-Register and Initiate operation of device D
2505	IN	Input to A-Register and initiate operation of device D
2506	JNR	Jump if not ready
2507	JNE	Jump if no error

Specific meanings for each action are determined by the specific requirements of each given device D. Refer to specific descriptions of the various devices for detailed information.

Since indexing address modification affects Bits 13 through 0 of the instruction, indexing can change the meaning of the GEN2 instruction.

The device address is a function, not of the device, of the physical location of the device within the GE/PAC cabinetry. Therefore, it is extremely important for each system program to permit device addresses to be changed without costly reprogramming or major reassemblies. A recommended GE/PAC technique is to list all device addresses in a table and assign a common symbol to each table entry. This device address table must be located absolutely. Each program using input/output devices will refer symbolically to the appropriate device address as illustrated in Figure 13. A recommended alternate technique is to use the EQL pseudo-op as illustrated in Figure 14.

1.4.2 INPUT/OUTPUT COMMUNICATION

GE/PAC 4040 provides up to twenty-four independent communication paths between the A-Register and Input/Output devices external to the computer. Each path is specified by the most significant 6 bits of the device address and links the arithmetic unit with a module. Module refers to the circuitry which controls a device directed by the S portion of the GEN2 instructions. Associated cabinetry is also described as a module. Examples of modules are peripheral buffers, analog input scanners, and digital input scanners.

The term device designates an electronic or mechanical unit controlled indirectly by the computer via the GEN2 instruction. The device is the interface between the computer and an operator or between the computer and the process. Examples of devices are paper tape readers and punches, analog-input sensors, digital-input sensors, and operator console controls.

```
        ORG
RDR    *CON 0,1100                 CONSOLE PAPER TAPE READER
        . .                         .
TYP1   *CON 0,1103                 TYPEWRITER NO. 1
RDR2   *CON 0,1105                 PAPER TAPE READER NO. 2
        . .                         .
DIS    *CON 0,4400                 DIGITAL INPUT SCANNER
AIS    *CON 0,2100                 ANALOG INPUT  SCANNER
MCO    *CON 0,4300                 MULTIPLE CONTACT OUTPUT CONTROL
TCO    *CON 0,4600                 TIMED CONTACT OUTPUT CONTROL
DRUM   *CON 0,4500                 DRUM
******
TYPE   LDX TYP1,X
       LDA CHARACTER
       OUT 0,X
        .
        .
        RECOMMENDED DEVICE ADDRESS ADDRESSING TECHNIQUE
```

FIGURE 13

```
RDR    EQL /1100                   CONSOLE PAPER TAPE READER
        .                           .
TYP1   EQL /1107                   TYPEWRITER NO. 1
RDR2   EQL /1101                   PAPER TAPE READER NO. 2
        .                           .
DIS    EQL /2100                   DIGITAL INPUT SCANNER
AIS    EQL /4400                   ANALOG INPUT  SCANNER
MCO    EQL /4600                   MULTIPLE CONTACT OUTPUT CONTROL
TCO    EQL /4500                   TIMED CONTACT OUTPUT CONTROL
DRUM   EQL /4600                   DRUM
******
TYPE   LDA CHARACTER
       OUT TYP1
        .
        .
        RECOMMENDED DEVICE ADDRESS ADDRESSING TECHNIQUE
```

FIGURE 14

The select instruction, SEL, is never required in a normal system program. If the necessary switching and data tramsmission between the arithmetic unit and device cannot be accomplished within 24 microseconds, a SEL must be given prior to the operation of the device.

1.4.3 I/O CONTROLLER INSTRUCTIONS

The GE/PAC 4050/60 I/O Controller instruction ØDL combines the functions performed by RBL and ØUT to transfer data directly from a list in memory to an output device without using the A-Register. If the B-Register is substituted for the A-Register in the description of RBL and ØUT, the composite function is identical to that performed by ØDL. Similiarly, IDL combines the IN and AEL functions without affecting the A-Register. These instructions, when used in a device's non-inhibitable location(s), perform hardware block buffering of data between the list and the device. Figure 15 illustrates format details.

Output block buffering is accomplished in the following manner. The output program assembles output data into a list by AEL'ing each output item into a separate word of the list. The data format is the same as the ØUT instruction. The device's non-inhibitable interrupt is then ACT'ivated. Whenever the output device is ready to receive data, it requests a non-inhibitable data interrupt.

The ØDL instruction in the device's interrupt location is executed when the interrupt request is serviced. The program counter is unaffected. If the list is not empty, data is removed from the list, transferred to the device's buffer register, and a device operation initiated. If the list is empty, a list-empty echo signal is transmitted to the program interrupt module. This signal requests an inhibitable program interrupt. An SPB instruction in this echo interrupt location transfers program control back to the output program; additional data can be assembled into the output list, if required. When a hardware malfunction occurs, the data interrupt does not occur. This condition can be sensed by periodically executing the appropriate JNE instruction.

Input block buffering is accomplished in an analogous manner using the IDL instruction. Input parity errors are interpreted as hardware malfunctions.

The I/O list must be preceded by an I/O control word as shown in Figure 15.

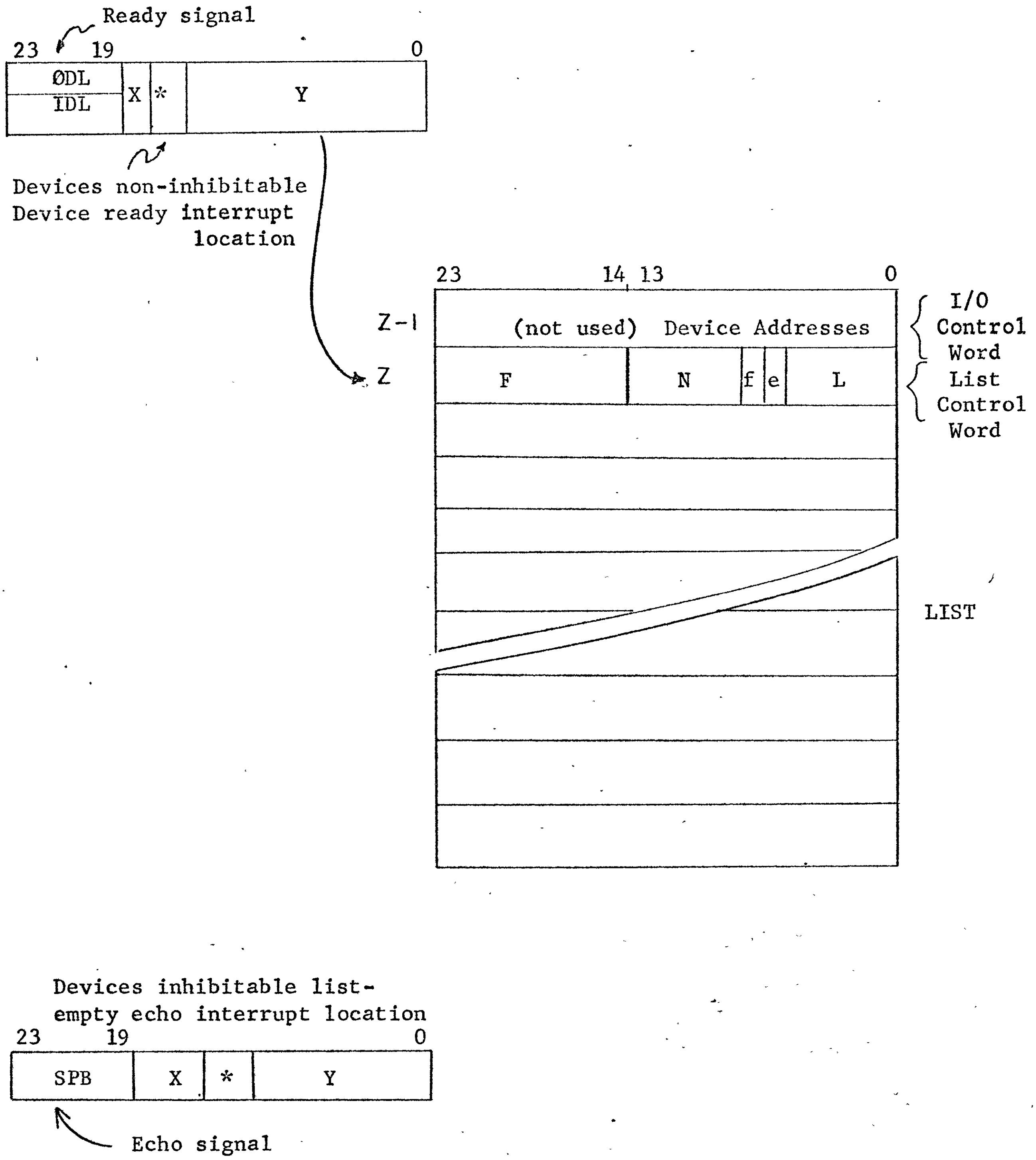


Figure 15 - I/O CONTROLLER INSTRUCTIONS

1.5 DATA COMMUNICATION

1.5.1 DRUM MEMORY

The GE/PAC drum is available for bulk storage in various sizes from 16K to 256K words. Transfer is accomplished by full words, between any core address and any drum address. Transfer may be specified in blocks of N words $(0 \leq N < 16K)$.

Write protect pinboard, located on the drum cabinet, provides selective drum memory protection in blocks of 4K words Figure 16 illustrates details.

A program load option is available to provide pushbutton transfer of 18 drum words (drum addresses 00_8 through 21_8) to core locations 00_8 through 21_8.

1.5.1.1 Instructions

Four instructions provide program control. (DRUM EQL 2400_8 is the device address of the drum).

ØUT DRUM

ØUT sets the drum's JNR signal not-ready and causes the drum controller to fetch the contents Y of location zero. Y is the address of a block of three consecutive drum control words. The drum controller performs a drum operation as specified by these control words.

Location 00	23——16	15——0 Y

Must be zero

23 0=Read Drum 1=Write	22—18 not used	17——0 Starting Drum Address
23——14 not used		13——0 number of words to transfer
23——16 not used		15——0 Starting Core Address

A parity check is made on each word as the transfer is accomplished; the Drum Error Flip-Flop is set when a parity error is detected. The Drum Error Flip-Flop is reset by each new ØUT instruction. If a parity error occurs during a drum operation the block transfer will halt uncompleted. The faulty word is written on drum for a core to drum transfer; it is now written on core for a drum to core transfer. The drum address of the faulty word may be computed by performing a field subtraction (see SFA instructions) of 1 from the least significant 9-bit field of the address displayed in the drum address register loaded on the front of the drum cabinet.

If a drum transfer operation is in progress when ØUT is executed, the new ØUT is ignored.

JNR DRUM

JNR transfers program control to the second sequential location if a drum transfer is in prograss; program is transferred to the first sequential location if the drum is not in operation.

The JNR signal (test line 1) may optionally be connected to the Automatic Program Interrupt module to initiate a program interrupt when the signal changes from not ready to ready.

JNE DRUM

JNE transfers program control to the second sequential location if the Drum Error Flip-Flop is reset; it transfers program control to the first sequential location if the Flip-Flop is set (a parity error causing a drum operation abort has occurred). The Drum Error Flip-Flop is reset by:

1. Pushing the DC POWER switch on the computer console to the initialize position.

2. Pushing the CLEAR ERROR switch on the system console located on the front of the drum cabinet.

3. Each new ØUT instruction

ABT DRUM

ABT aborts any current drum operation following the completion of the current word transfer. The JNR signal will indicate "ready" (within 100 microseconds) when the drum operation is terminated. ABT is ignored if the drum is not in operation.

ACT DRUM

ACT resets the drum ready signal for 8 microseconds. If the drum is not in operation and if the drum ready signal is connected to a program interrupt input, a program interrupt is requested. If the drum is in operation the interrupt is not requested.

INTERRUPT OPERATION

One interrupt operation is recommended. Program control is accomplished by an SPB instruction in an inhibitable interrupt location. The JNR (test line 1) signal is used as both the change and level inputs to a two-input program interrupt. ACT is used to conditionally request this program interrupt.

1.5.1.2 Timing

The time required to effect the transfer includes drum access time plus actual transfer time:

access time:	variable - 0 to 16.66 MS
transfer time:	0.0650MS per word

When a drum transfer operation is in progress, the arithmetic units are slowed down by small percentages as shown in the following tables.

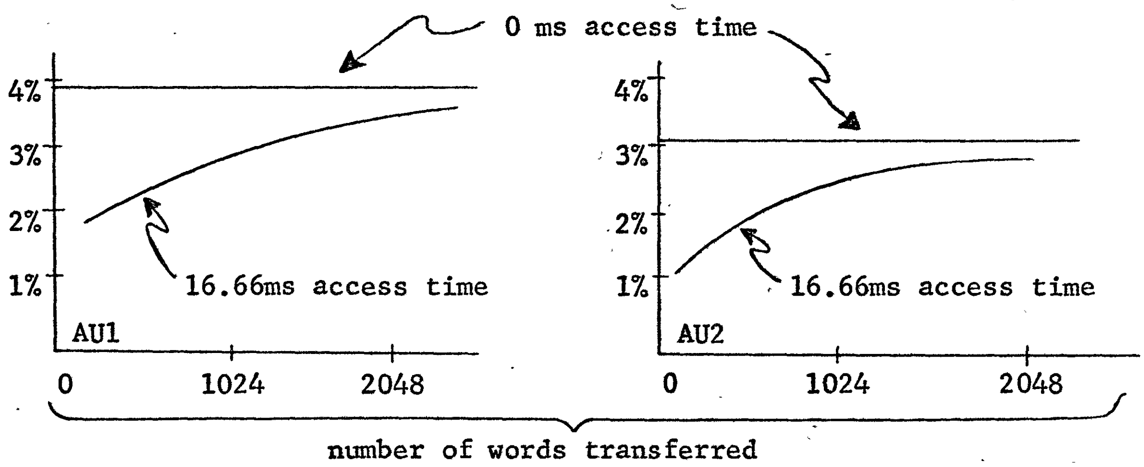
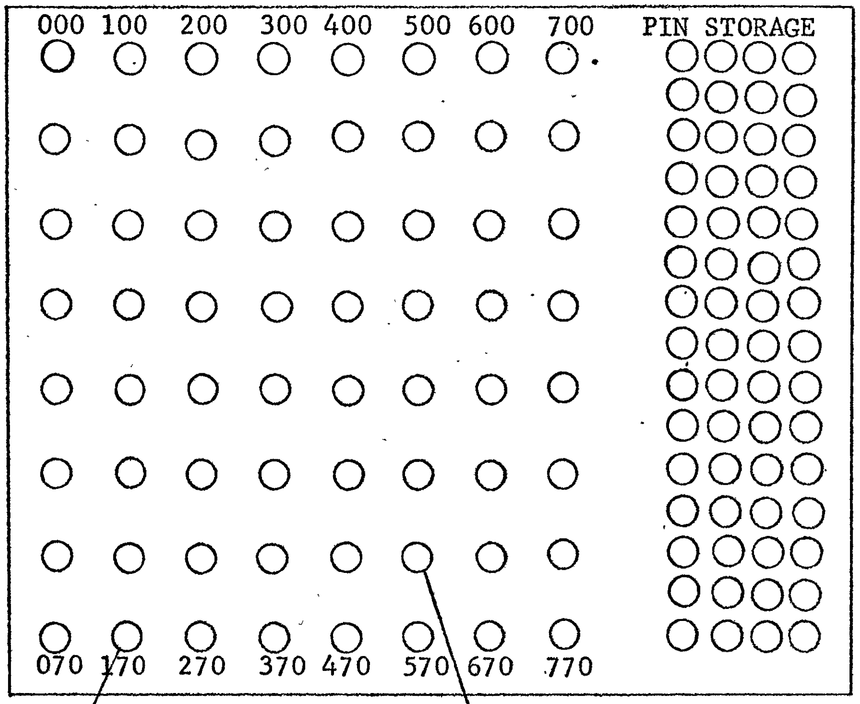


Figure 16 - ARITHMETIC UNIT SLOWDOWN
(dur.ing drum operation)



Pin inserted protects;
Pin removed unprotects.

Octal address of first word of protected block of 4096 words is 560000.

Figure 17 - DRUM WRITE PROTECT PINBOARD

1.6 PROCESS COMMUNICATION

1.6.1 DIGITAL INPUT SCANNER

The Digital Input Scanner (DIS) is a solid state device which reads groups of 21 logic variables into the A-Register. The logic variables normally indicate the status (true/false, yes/no, set/reset, open/closed, on/off) of various process system control devices such as relays, valves, and motors. Indication of the status of various components within the GE/PAC computer system is another typical application of logic variables.

23	22	21	20	19		2	1	0
0	0	0	L_{20}	L_{19}	- - - - - - - - - - -	L_2	L_1	L_0

Digital Input (A-Register) Format

Certain inputs may be subject to short circuit conditions. In this event, the project engineer may provide fused protection of the input circuits. One fuse protects 20 logic inputs in a group; the 21st input (L_{20}) indicates the status (good/blown) of the fuse. Indication of a good fuse ($L_{20} = 1$) does not necessarily imply that the corresponding data is valid. $L_{20} = 0$ implies bad data.

The DIS is packaged in standard configurations of 168, 504, and 1344 inputs:

Number of Inputs	Group Number K
1 to 8 groups, 21 inputs each	00_8 thru 07_8
1 to 24 groups, 21 inputs each	00_8 thru 27_8
1 to 64 groups, 21 inputs each	00_8 thru 77_8

An option provides change detection on a group and provides a true signal when any of the 21 variables of the group changes state. The isgnal may be connected to a program interrupt input or may be connected as a DIS input to be treated as a logic variable itself.

Figure 19 presents two coding examples of DIS usage. Figure 19a asks a digital question. Further information on the evaluation of a Logic equations is presented in paragraph 1.3.3. Figure 19b is a scan for digital alarm routine. Seven consecutive groups are scanned. Selected points are alarmed (printed in red) when they change from normal to off-normal and alarmed again (printed in black) when the return to normal.

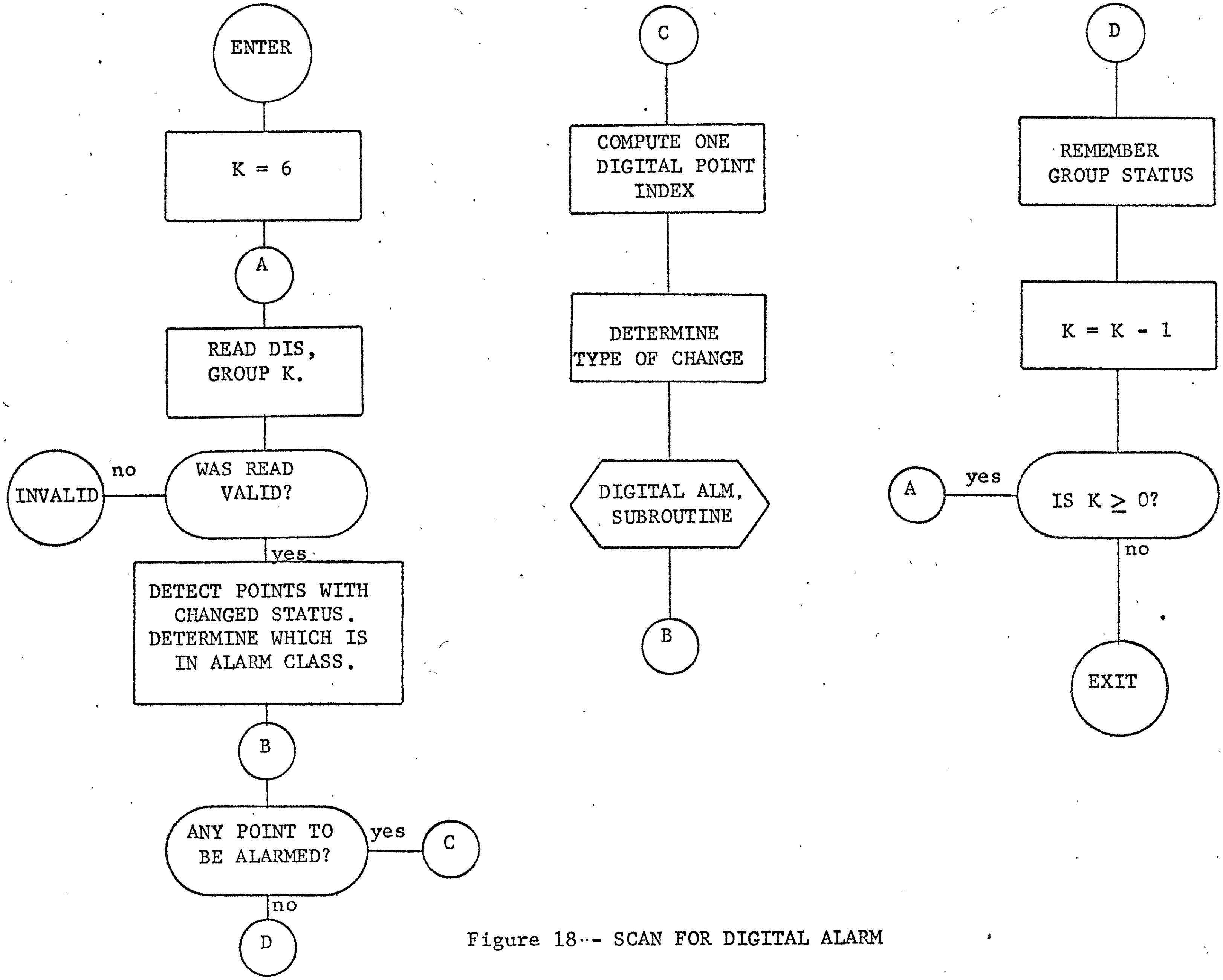


Figure 18 -- SCAN FOR DIGITAL ALARM

Operation of the DIS requires one instruction:

IN DIS +K

IN replaces (A_{20-0}) by the K^{th} group of 21 logic variables. Bits A_{23}, A_{22}, and A_{21} are set to zero. DIS is the device address of the Digital Input Scanner and must be defined by an EQL Pseudo-Op. The range of the group number K is $0 \leq K \leq 77_8$.

Values of K greater than 77_8 modify the device address, becoming an indeterminate program bug.

1.6.2 ANALOG INPUT SCANNER

Quantities (temperatures, rates of flow, weights, pressures, etc.) are transduced into voltages by analog input devices. The Analog Input Scanner (AIS) is a module which controls the selection and conversion of these voltages into numeric count values.

The functional relation between the numeric count value and its corresponding physical quantity is one of several forms depending upon the specific Input Sensor Device utilized. One AIS operation nominally requires 25 milliseconds. The AIS is not directly used by a functional program. Analog values are read by queueing request to a scan control program.

The Analog to Digital Converter (A/D Converter), which measures the input voltage and generates an equivalent numeric count value, is the heart of the AIS. Two different types of A/D Converters are available in GE/PAC scanners:

Successive Approximation A/D Converter
Integrating A/D Converter

Each AIS module must have one of the above types; it may not have both. If both types are required in a given application, two GE/PAC scanner modules must be supplied. Refer to paragraph 1.6.3.1 for description of each type.

The scanner is operated by outputting a scan command word (Figure 20) from the Arithmetic Unit (ØUT, ØDL instructions). Detailed operating instructions are presented in paragraph 1.5.3.3.

There are two modes of operation as seen from the program control viewpoint:

1. Addressing of inputs individually.
2. Addressing of inputs in sets of N (N=2,4,8)

```
DIS    CON /DDDD                 DEVICE ADDRESS
       LDX DIS,4
       IN  3,4                   READ DIS, GROUP 3
       TEV 20                  B IF (VALID READING) --,INVALID,010
       BTS INVALD
       TOD 5             010   B IF (BIT5 * BIT 20) --,020
       REV 20
       BTS(TRUE)
          (FALSE)        020
```

Figure 19a - EXAMPLE OF DIGITAL QUESTION

```
K      EQL
L      EQL
I      EQL
DIS    EQL(DEVICE ADDRESS)
DSA    LXK 6,K           000     K = 6
       LXK 120,L                 L = 6 * 20
DSAA   IN  DIS,K         005     READ DIS, GROUP K
       TEV 20                  B IF (VALID READING) --,INVALID,010
       BTS INVALD        010
       STA TEMP                  SAVE GROUP.
       ERA DISVAL,K              DETECT POINTS WITH CHANGED STATUS.
       ANA ALARM,K               DETERMINE WHICH ARE IN ALARM CLASS.
DSAB   TZE               015   B IF (ANY POINTS TO BE ALARMED) -,30,2
       BTS DSAD
       IAI               020     DETERMINE WHICH ONE
       CLZ
       LXC I                      BIT I
       PAI
       RBK 0,I                   RESET BIT I AND SAVE
       STA TEMP1
       LDA L                     COMPUTE DIGITAL POINT INDEX J = L+I
       ADD I
       STA J
DISC   LDA TEMP                  DETERMINE TYPE OF CHANGE
       ERA NORMAL,K                1 = NORMAL TO OFF-NORMAL (RED)
       IBK I                       0 = RETURN TO NORMAL     (BLACK)
       STA COLOR
       SPB                       CALL DIGITAL ALARM ROUTINE
       LDX J,3                     CALLING SEQUENCE (POINT NUMBER)
       LDA COLOR                   CALLING SEQUENCE (COLOR CODE)
       LDA TEMP1                 GO TO 015
       BRU DSAB
DSAD   LDA TEMP          030     REMEMBER GROUP STATUS
       STA DISVAL,K
       INX -20,L                 L = L - 20
       DMT K                     K = K - 1
       BTS DSAA                  IF (K) 040, 005, 005
          (EXIT)         040
TEMP   BSS 1
TEMP1  BSS 1
DISVAL BSS 7                     CURRENT STATUS OF DIGITAL INPUTS,
NORMAL BSS 7                     NORMAL  STATUS OF DIGITAL INPUTS.
ALARM  BSS 7                     ALARM CLASS OF DIGITAL INPUTS.
                               X   1 = ALARM, 0 = DO NOT ALARM
```

Figure 19b - SCAN FOR DIGITAL ALARM ROUTINE

Any AIS can be used in the first mode. The second mode is limited to use with scanners having the automatic group advance option.

Each Scan Command Word (SCW) uniquely defines a single analog input when the AIS is used in the single input mode. A zero in Bit 18 of the Scan Command Word indicates this mode. Each group may consist of a maximum of 256 inputs.

Each Scan Command Word specifies a set of N-W analog inputs when the Scan Command Word indicates this mode. The initial ØUT instruction results in the conversion of the addressed input (group W). Succeeding IN instructions increment the group address and produces conversion of the inputs in groups W+1, W+2,.. and N-1.

The group input mode of operation allows a higher scanning rate than the single input mode. Scanning rates are summarized in the following table:

NOMINAL SCANNING RATES (points per second) for All-Core Monitor

<u>Successive Approximation A/D Converter (Hg Relays)</u>

	Single Input Mode	Group Input Mode		
		2 pts/group	4 pts group	8/g
Free-Running Scanner	50	90	140	200
4040 Programmed Control	45	82	127	
4060 Programmed Control	49	87	140	

<u>Integrating A/D Converter (Hg Relays)</u>

	Single Input Mode	Group Input Mode		
		2 pts/group	4 pts/group	8/g
Free-Running Scanner	28	N/A	N/A	N/A
4040 Programmed Control	-	N/A	N/A	N/A
4060 Programmed Control	24	N/A	N/A	N/A

Figure 20 - SCANNING RATES

To permit successful operation of the AIS in the group input mode, all points to be so used must be grouped (arranged so inputs with like matrix and point addresses (SCW_{14-7}) have the same voltage scale). Since this process involves the participation of customers, Engineering, Marketing and Programming personnel, the group input mode should not be needlessly used; this avoids an unnecessary expense to the customer. Points to be used in the single input mode can be arbitrarily grouped.

1.6.2.1 Analog to Digital Converters

The successive approximation A/D converter successively generates the sign and each of 12 bits of the count value by comparing the input voltage to known signed voltages. The conversion operation is accomplished in approximately 700 microseconds. The resulting numeric count value represents a voltage value which occurred sometime during the 700 microsecond period. The count value is placed right justified into C_{23-6}.

The integrating converter generates a sequence of pulses whose instantaneous signed pulse rate is directly proportional to the instantaneous signed input voltage. A counter circuit accumulates the number of pulses occurring during a known period of time (integrating time) and places it right justified into C_{23-6}. It is an integrated average over the specified time period.

The successive approximation converter is considerably faster than the integrating converter. It measures instantaneous voltage and is sensitive to noise errors. Therefore, every input signal is usually filtered through an individual resistance-capacitance filter circuit. Unfortunately these circuits frequently introduce another type of error called common mode. The advantage of the integrating converter is that it tends to filter the input signal; filter circuits are not always necessary.

1.6.2.2 Specification of the Scan Command Word and C-Register Formats

The various fields which compose these formats are described in this subsection.

Sensor Address (SCW 16-7)

Bits 16 thru 7 uniquely define a single analog input. Bits 16 thru 15 specify the inputs group address.

Operation Mode(SCW 18-19)

Bit 18 specifies the mode of operation.

Voltage Scale Control (SCW2-0)

Bits 2-0 specify the voltage range (-FSV volts to + FSV volts) within which the input voltage is presumed to lie. The scanner generates (in the C-Register) a signed numeric count value proportional to the input voltage when the voltage lies within this range. If the voltage is not within the specified range, a signed maximum count value is generated and the converter overflow bit(s) is set. To obtain maximum precision, the voltage scale should be chosen so that the magnitude of the input voltage falls between ½ FSV and FSV.

The measured numeric voltage is equal to:

Successive Approximation Converter:

$$\text{numeric voltage} = \frac{\text{FSV}}{4000} * \text{numeric count value}$$

Integrating Converter:

$$\text{numeric voltage} = \frac{\text{FSV}}{\text{Full scale counts}} * \text{numeric count value}$$

Operation Control (SCW 6-5)

Scanners with the successive approximation converter only measure DC voltages. Scanners with the integrating converter measure either AC or DC voltages and can be used to measure the pulse rate of non-continuous voltages. AC measurements are restricted to multiples and submultiples of 60 cycles (50 cycles when using a 50 cycle converter).

Integrating Time Control (SCW 4-3)

This field is applicable only to scanners with integrating converters. Refer to paragraph 1.6.2.1 for a discussion of its usage.

Unused Fields (SCW 23-20, SCW 17)

These fields are reserved for future use with automatic scanning functions. In specific applications, program information may be placed in the fields.

Converted Count Value (C23-6)

The two's complement representation of the converted numeric count value is placed in C_{23-6}. The AIS is fully compatible with GE/PAC two's complement fixed-point arithmetic.

Error Indicator Bits (C2-0)

Bits C_{2-0} are error indicators associated with the preceding scan operation.

C_{2-0} Last Scan Operation

000 No error. Count value is valid.

001 Scanner overload error. Count value is meaningless.

This is a rare hardware failure within the point selection circuitry of the Common scanner control. A scanner overload alarm message should be printed with the point address. The AIS may or may not be usable for reading other analog inputs.

010 Undefined (hardware error)

011 Integrating A/D Converter:

An open or high resistance thermocouple was detected during the last scan operation. Count value is meaningless. An alarm message should be printed.

100 Undefined (hardware error)

101 Converter Overflow: count value is maximum, but meaningless.

110 Undefined (hardware error)

111 Successive approximation A/D Converter: converter overflow error. If the input device was a thermocouple, this may be indicative of an open or high resistance thermocouple. Count value is maximum value, but meaningless.

Integrating A/D Converter:
Both converter overflow and open thermocouple errors. Count value is meaningless.

SCAN COMMAND WORD (Successive Approximation Converter)

23 — 20	19-18	17 — 15	14 13 12 11	10 9 8 7	6 — 3	2 — 0
reserved for upward compatibility		group address W	matrix address M/N	point address P/Q	must be zero	voltage Scale Control

00 = single Input mode
01 = group Input mode (2 or 4 points per group)
10 = single Input mode
11 = group Input mode (8 points per group)

SCAN COMMAND WORD (Integrating Converter)

23 — 20	19-18	17 — 15	14 13 12 11	10 9 8 7	6 — 5	4 — 3	2 — 0
reserved for upward compatibility		group address W	matrix address M/N	point address P/Q	operation control	integrating time control	voltage scale control

C-REGISTER

23	22 — 6	5 4 3	2	1	0
S	Converted integral count value (2's complement representation, scaled B17)	zero			

Scaled offset corrected count

If integrating converter:
converter overflow indicated

If successive approximation:
converter overflow or open thermocouple indicated

If integrating converter:
open thermocouple indicated

If successive approximation:
xeros indicated

Scanner overload or converter overflow or open thermocouple indicated

Voltage Scale Control

SCW 02-00	FULL SCALE VOLTAGE (FSV)			
	Successive Approximation Converter, Model 4130			
	Model 4121 (PRESTON) Low-Level Amplifier	Model 4122 (PRESTON) Low-Level high-common-mode Amplifier	Model 4127 (PRESTON) High-Level Amplifier	VIDAR Model 4135 Integrating Converter
0	10 mv	10 mv	80 mv	10 mv
1	20 mv	40 mv	160 mv	20 mv
2	40 mv	160 mv	320 mv	40 mv
3	80 mv	10 v	640 mv	80 mv
4	160 mv	10 mv	2.5 v	160 mv
5	10 v	40 mv	10 v	320 mv
6	--	160 mv	--	640 mv
7	--	10 v	--	1 v

Operation Control for Integrating Converter

SCW 06-05	Mode of Operation
0	Voltage, DC
1	Voltage, AC
2	Voltage, DC with open thermocouple detection.
3	Count pulses per unit integrating time.

Integrating Time Control for Integrating Converter

SCW 04-03	Integration Time (MS)		Full scale counts at 1000 KC count rate		Operation time (MS) Single input mode	
	60 cycle	50 cycle	60 cycle	50 cycle	60 cycle	50 cycle
0	13.67	20	1:16,667	1:20,000	35.2	38.5
1	33.33	40	1:33,333	1:40,000	51.9	58.5
2	100.0	100	1:65,535	1:65,535	118.5	118.5
3	1000.0	1000	1;65,535	1:65,535	1018.5	1018.5

Figure 21 - AIS CONTROL AND DATA FORMATS

1.6.2.3 Instructions

Program control over the AIS is accomplished with the following instructions:

ØUT AIS

ØUT transfers a scan command word from the A-Register to the AIS scanner command register T, resets the scanner ready and data ready signals, and initiates one complete scan operation (connection of a specified input point to the scanners voltage measuring circuitry, signal amplification if required, and conversion to an integral count value). The time required to effect the operation depends upon the A/D converter. The successive approximation converter requires approximately 20 milliseconds; the time for the integrating converter depends on the integrating time specified. When the operation is completed, the data ready signal is set. This signal may be used to request a program interrupt. ØUT may be executed at any time; if the AIS is in operation, this previous operation will be aborted. The contents of A are preserved by this instruction.

ØPR AIS

ØPR causes the AIS to go through one partial scan operation. In the single input mode, a reconversion is made on the previously specified input point. The data ready signal will be set in approximately 750 μs with the successive approximation converter, and the integrating time with the integrating converter.

In the group input mode, the group address is reset to zero and reconversion occurs. The successive approximation converter sets the data ready signal for the input of group 0 in 3 MS. The integrating converter sets the signal in the specified integrating time plus 2 MS. ØPR may be executed at any time; any previous operation is aborted.

IN AIS

IN transfers the count value and error indicator bits from the scanner converter C-Register into the A-Register. The contents of the C-Register are not destroyed. In the group input mode, the data ready signal is reset, and conversion of the input in the next group is initiated. The data ready signal for this next input occurs in 1.7 MS with the successive approximation converter.

IN may be executed at any time. However, if the data ready signal were reset, the transferred data is meaningless.

The execution of IN, to transfer the count from the last group, (the addressed group if the single input mode) sets the scanner ready signal. This signal may be used to effect a program interrupt. Any additional IN will reread the count from the last group and initiate a reconversion of the input in the last group.

JNR AIS

JNR transfers program control to the second sequential location when the data ready signal is reset. If the signal is set, program control is transferred to the first sequential location.

JNE AIS

JNE transfers program control to the second sequential location if an error has not occurred during some preceding scan operation; otherwise program control is transferred to the first sequential location. The error Flip-Flop is reset by JNE.

ACT AIS

ACT forces the scanner ready and JNR signals to be reset for 16 µs. If the scanner is ready and if the scanner ready (JNR) signal is a program interrupt input, this program interrupt is requested; otherwise the interrupt is not requested.

1.6.2.4 Interrupt Operation

a. Single interrupt operation
Hardware required:
One 2-input program interrupt
    Change input: JNR signal
    Level input: JNR signal

Program required:
SPB instruction in interrupt location
    1 I/O buffer driver program
    1 Scan command buffer list
    1 Data buffer list

b. Three interrupt operation (GE/PAC 4050/4060 only)
Hardware required:
GE/PAC 4050/4060 Arithmetic Unit
One 2-input program interrupt
    Change input: Scanner ready signal (RDY2)
    Level input: Scanner ready signal (RDY2)

One 2-input program interrupt
Change input: Data ready signal (RDY1)
Level input: Data ready signal (RDY1)

One 1-input program interrupt
Change input: ØDL list-empty echo signal

Program required:

ØDL	instruction in scanner ready interrupt location.
IDL	instruction in data ready interrupt location.
1	Scan command buffer list
1	Data buffer list
SPB	instruction in echo interrupt location.

1.6.2.5 Analog Input Point Addressing Schemes

Each input is subject to three different identification schemes:

1. System address (e.g. BF101)
   This address is assigned by the customer and/or the system analyst.
2. Termination (scan command) address (WMNPQ)
   This address is assigned by the system engineer and refers to the location of the termination point in the AIS termination rack.
3. Point index
   This address is assigned by the programming analyst. It should render a functional relationship to the System Address.

1.6.2.6 Accuracy in Measurement of Analog Quantities

Measurement of an analog quantity involves four separate steps; each step contributed to measurement inaccuracies.

1. Conversion of the analog quantity to an analog voltage by a transducer. Equations defining this relationship for standard transducers are usually accurate to no more than ½%. Additional accuracy can be obtained by individual calibration of each sensor.
2. Transmission of the analog voltage from the sensor to the AIS. Errors are introduced by voltage drops and electromagnetic inductive effects (noise).

3. Conversion of the analog voltage to a numeric voltage by the AIS and read-in of this numeric value into the computer memory. The conversion process is subject to several errors which can be corrected by program. Details are given below.
4. Conversion of the corrected numeric voltage to a numeric quantity corresponding to the original analog quantity. The computer program uses the inverse to the equation referred to in 1 above.

The GE/PAC scanner can be programmed to convert an input voltage into a numeric count value that differs from the true voltage value by no more than 0.001*(full scale voltage).

Inaccuracies in measuring input voltages by a GE/PAC scanner are determined by the algebraic sum of three types of errors. These errors are gain errors (errors in the slope of the input-output relationship), offset errors (the degree by which the intercept does not go through zero) and repeatability errors (noise). This total error, which is approximately 0.005*(full scale voltage), is subject to reduction by programming techniques.

Offset errors can be reduced by making weighted average measurement[7] on a short circuit (zero input) and reducing all other readings by this measured offset value. The equation for this is:

True reading = actual reading - offset reading (signed offset and actual readings are made on the same voltage scale).

Gain errors can be reduced by making weighted average measurements[7] on a known reference voltage and all readings corrected by multiplying by a gain correction factor calculated from this reading.

$$\text{Gain correction factor} = \frac{\text{Reference voltage actual count - offset count}}{\text{Theoretical count for reference voltage}}$$

Where reference voltage actual count and offset count are made on the same scale and theoretical count for reference voltage is calculated by:

$$\text{Theoretical count for reference voltage} = (\text{Reference voltage}) * \frac{\text{Full scale count}}{\text{Full scale volts}}$$

[7] - A suggested weighted average is 7/32 * (sum of last four readings)+ 1/8 * (new reading) when a reading is made once every 20 seconds.

Once the gain correction factor is determined, all other readings should be corrected by multiplying by this factor:

True reading = (actual reading -- offset) * (gain correction factor) (all measurements are made on the same voltage scale)

Repeatability errors can be reduced by digital filtering. This is the process of averaging a number of[8] readings to minimize the effects of noise in the system.

The gain correction factor and the offset correction term both vary as a function of time and temperature. Therefore, measurements must be made often to determine these quantities to maintain small changes between measurements. Experience indicates an interval of approximately five minutes is reasonable under normal environmental conditions. If the machine is located in an environment where temperature changes are very rapid (0.5°F/min.) shorter intervals are desirable.

The degree of improvement in each of these error terms is dependent upon several external effects but generally speaking improvements in the gain errors can be 5 to 1; improvements in repeatability errors can be 2 or 3 to 1.

The offset error improvement is more dependent upon the scale used. On more sensitive scales the improvement may be as high as 5 to 1 also.

A special type of scanner measurement error occurs whenever the scan operation results in a converter overflow (the input voltage magnitude exceeds the specified full scale voltage). The resulting numeric count value is meaningless. Furthermore, the voltage value measured in the next scan operation may be in error because of residual effects from the preceding voltage overload. The magnitude of this error is a function of this voltage overload, but should never exceed 0.01 * (full scale voltage).

If this error is unacceptable, then a dummy scan (preferably of an offset value) may be inserted following the converter overflow to absorb the residual voltage effects.

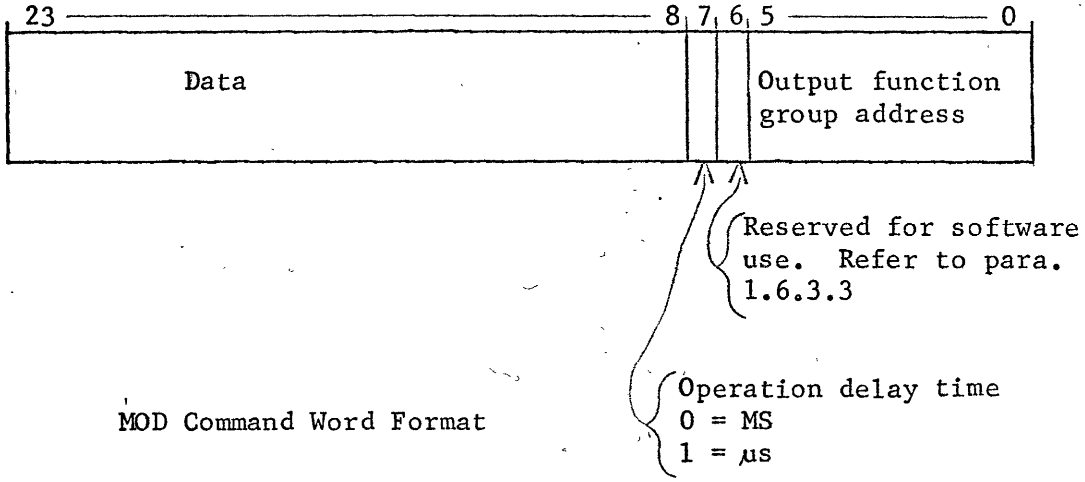
[8] -A suggested weighted average is 7/32* (sum of last 4 readings) + 1/8* (new reading) where a reading is made once every 20 seconds.

1.6.3 MULTIPLE OUTPUT DISTRIBUTOR

The Multiple Output Distributor (MOD) consists of a Multiple Output Controller and output functions. The Multiple Output Controller (MOC) module outputs data from the arithmetic unit to output functions within the MOD. The output functions remember the data, and control the following output devices:

1. Display output devices
2. Binary output devices
3. Analog output devices

The MOD operates through a command word from the computer to the MOC command register.


23
8
7
6
5
0
Data
Output function group address
Reserved for software use. Refer to para. 1.6.3.3
Operation delay time
0 = MS
1 = µs

MOD Command Word Format

The MOC transfers the data to the memory of the output function specified by the command word group address. The MOD is then ready to accept another output. If the output function's memory is electromechanical, the transfer operation requires 4 milliseconds; if solid state, the transfer requires 40 microseconds.

Overload detection circuitry will abort the output operation and set a MOD overload indicator if the MOC attempts to select more than one output function address.

The MOD is usable with 0 or 1 program interrupts. One interrupt is required for efficient on-line operation.

The MOD is available in the following sizes:

Number of Outputs	Output Function Group Addresses
1 to 8 groups, 16 outputs each	00_8 thru 07_8
1 to 16 groups, 16 outputs each	00_8 thru 17_8
1 to 32 groups, 16 outputs each	00_8 thru 37_8
1 to 64 groups, 16 outputs each	00_8 thru 77_8

1.6.3.1 Instructions

MOD program control is accomplished through the following instructions:

ØUT/MOD

ØUT transfers a command word from the A-Register to the MOC command register. It then initiates transfer of the data portion of the command word to the output function specified by the address portion of the command word. The contents of the A-Register are unchanged. Bit 7 of the command word specifies one of MOC delay times. The delay time must be long enough to allow the transfers to occur. If a MOD overload occurs, the output operation is aborted and the ready signal is set after the specified delay time.

ØUT may be executed at any time; if the MOC is in operation, the new operation is ignored.

JNR MOD

JNR transfers program control to the second sequential location if the normal ready signal is reset. If it is set, program control is transferred to the first sequential location.

ACT MOD

ACT resets the ready signal for 8 μs. If this signal is a program interrupt input and the MOC is not in operation, a program interrupt is requested.

JNE MOD

JNE transfers program control to the second sequential location if a MOD overload did not occur during a preceding output operation. Otherwise, program control is transferred to the first sequential location and the MOD overload indicator is reset.

INTERRUPT OPERATION

Single interrupt operation is recommended. The ready signal (test line 1) is used as the input to an inhibitable one-input program interrupt. Interrupt is conditionally requested by the ACT instruction.

1.6.3.2 Output Functions

Three standard output functions are described below. The memory elements in each are bistable latching relays. The relay requires less than 4 MS to switch positions.

The function group address does not uniquely specify an output function. Consequently, several output functions could have the same output function group address.

1. Display Output Function

   This function provides the control for one BCD display digit (including 4 bit BCD to decimal conversion if required). It requires 4 data bits of an MOD command word.

2. Binary Output Function

   This function provides control for 8 logic (binary) outputs corresponding to 8 data bits of the MOD command word.

3. Analog Output Function

   This function provides control for conversion from a 10-bit positive binary integer in a MOD command word to an analog voltage output.

Normally, one or more levels of electromechanical or pneumatic control exist between the output function and the actuating device. Consequently, the output function is not the final output actuating device. The MOD ready signal cannot be used to indicate the actual time of output action, nor as proof that the output will actually occur. If reliability considerations require, a set of contacts may be attached to the actuating device. Its status may then be read by the digital input scanner.

1.6.3.3 On-Line Programming Usage

All outputs (program control viewpoint) are initiated by an MOD driver subroutine, never by the originating subprogram. Each output has one of two relationships to the originating program:

1. Program continuation is independent of the output; therefore, the originating program is not notified when initiation of the output operation is completed.

2. Program continuation is dependent upon the output; therefore, the originating program must be notified when initiation of the output operation is completed.

The originating program uses Bit 6 of the MOD command word to specify which relationship applies to that output. Figure 22 presents the flow chart and coding applicable to MOD usage.

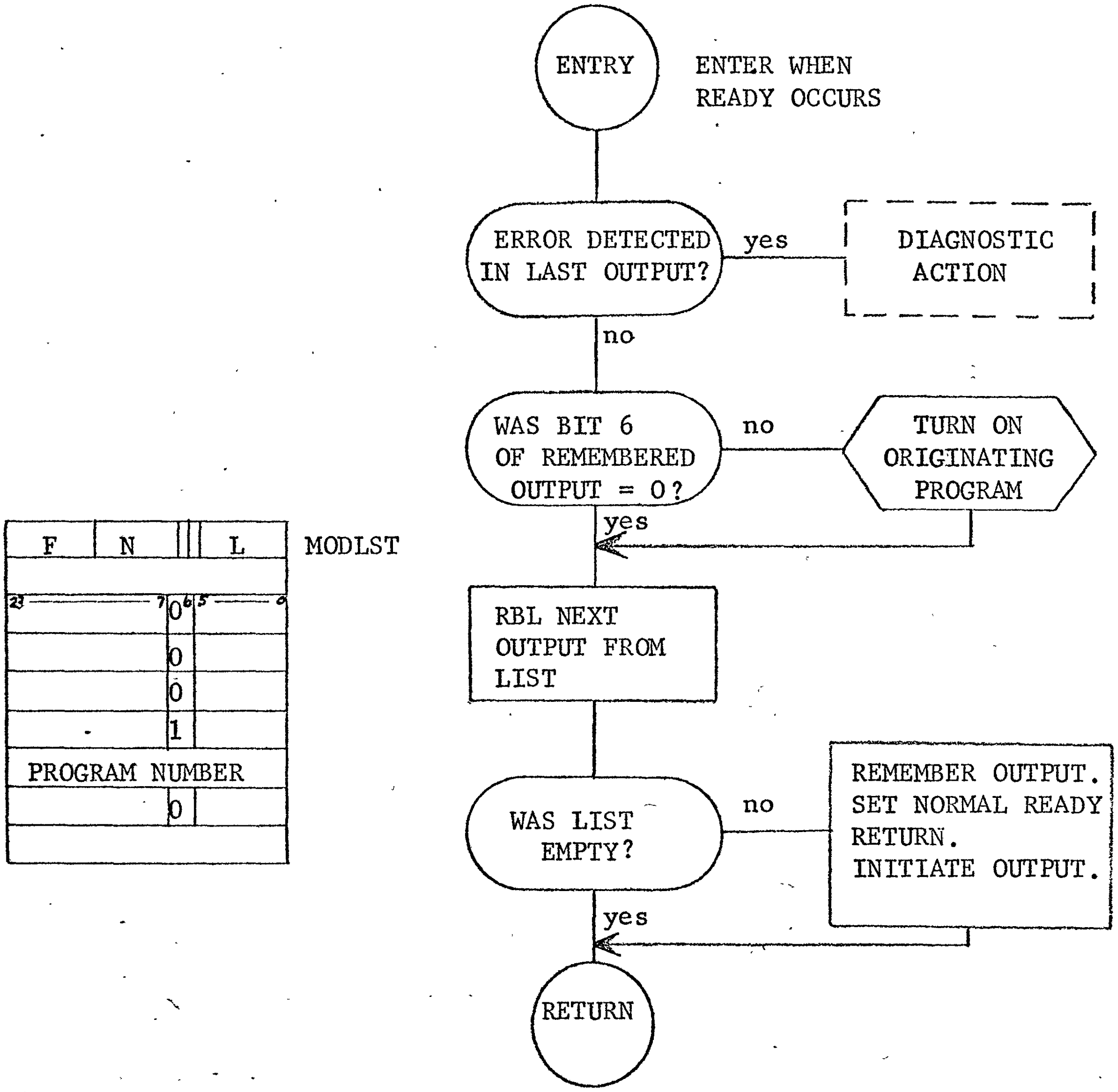

ENTRY
ENTER WHEN READY OCCURS
ERROR DETECTED IN LAST OUTPUT?
yes
DIAGNOSTIC ACTION
no
WAS BIT 6 OF REMEMBERED OUTPUT = 0 ?
no
TURN ON ORIGINATING PROGRAM
yes
RBL NEXT OUTPUT FROM LIST
WAS LIST EMPTY?
no
REMEMBER OUTPUT. SET NORMAL READY RETURN. INITIATE OUTPUT.
yes
RETURN
F
N
L
MODLST
23
7
6
5
0
0
0
0
1
PROGRAM NUMBER
0

Figure 22 - MULTIPLE OUTPUT DISTRIBUTOR DRIVE ROUTINE

```
* INTERRUPT BRANCH VECTOR LOCATION
       SPB MODDRV             INHIBITABLE- IF NORMAL READY SIGNAL
* MOD DRIVER ROUTINE--
MODDRV STX MODX,1         00
MOD    EQL                    DEVICE ADDRESS.
       STA SAVEA
       JNE MOD                 B IF (ERROR DETECTED IN LAST OUTPUT)--,
       BRU DIAGNOSTIC          X     10, DIAGNOSTIC ROUTINE.
       LDA MODCOM         10 B IF (LAST OUTPUT A NORMAL OUTPUT)--,
       TEV 6                   X     20, 30.
       BTS MOD20
       RBL MODLST         20    REQUEST TURNON OF ORIGINATING PROGRAM
       NOP
       STA *+3
       LDZ
       SPB TPNCO1                   CALL MONITOR-TURN-PROGRAM-ON-SPR(
       BSS 1                   X         O, PROGRAM-NUMBER).
MOD20  RBL MODLST         30    FETCH NEXT OUTPUT COMMAND WORD.
       BRU MOD40               B IF (LIST EMPTY) --, 40, 50.
       STA MODCOM         40    REMEMBER OUTPUT COMMAND WORD.
       RBK 6                    FORCE NORMAL READY SIGNAL.
       OUT MOD                  INITIATE OUTPUT OPERATION.
MOD40  LDA SAVEA          50    RETURN TO INTERRUPTED PROGRAM.
       LPR MODX
MODLST CON 0,0000000L
       BSS L
MODX   BSS 1
SAVEA  BSS 1
MODCOM BSS 1
```

1.7 PERIPHERAL COMMUNICATION

The ability for plant operating personnel to communicate with the digital computer is called peripheral communication. This chaper describes the 4201 Peripheral Buffer, the instructions, and interrupt operation.

1.7.1 Peripheral Buffer

The peripheral buffer controls peripheral device operation. Control information flows from the I-Register to the peripheral buffer module. Data information flows by character between the arithmetic unit and the module.

Peripherals available for use in the buffer are:

Paper tape reader, input
7-bit binary
Photoelectric

Paper tape punch, output
7-bit binary

Fixed carriage typer, output
6-bit BCD

Long carriage typer, output
6-bit BCD

Fixed carriage typer, Input/Output
6-bit BCD

Card reader, input
12-bit binary

A maximum of eight peripherals can be attached to the peripheral buffer. One of these must be a paper tape reader.

The peripheral buffer can be connected to any channel; several buffers can share the same channel. However, to use the standard load program, the buffer for console peripherals will use device addresses $11DD_8$.

Each peripheral buffer provides eight device addresses: octals DD00 through DD07. Console peripherals have standard addresses. Other peripherals are assigned addresses by the requisition engineer.

Console Devices	Standard Addresses
Paper tape reader	1100_8
Paper tape punch	1101_8
Fixed carriage typer	1102_8

The normal operation of a peripheral device is described in the following:

1. Operation Initiation
   The operation cycle is initiated by an IN (ØUT, IDL, ØDL) instruction. The following actions occur:

   1. One character of data is transferred (see IN (ØUT) below)
   2. Peripheral buffer becomes unavailable (the PBA Signal is reset)
   3. Device becomes not ready (the device's ready signal is reset)
   4. Device's deadman timer begins timing
   5. Mechanical operation of the device begins

2. Peripheral Buffer Availability
   The device requires the peripheral buffer only during the first portion of its operation cycle (see Figure 24). The device will then release the peripheral buffer and allow it to become available (the PBA Signal is set). At this time the buffer can be used to initiate the operation of any peripheral that may be ready.

3. Operation Completion
   Upon completion of the device's operation cycle, the following actions occur:

   1. The device becomes ready (the device's ready signal is set).
   2. The device's deadman timer is turned off.

Each peripheral has an independent deadman timer which detects and indicates device failure to complete its operating cycle. When a device fails to turn off its deadman timer within 4 to 8 seconds after an operation initiation, the device's deadman error indicator is set. If the failure occurs prior to peripheral buffer availability, the peripheral buffer remains unavailable until released by the execution of a JNE instruction. If the failure occurs after the PBA signal is set, the buffer can be used to operate other peripheral devices. In either case the device's ready signal remains reset until the proper JNE instruction is executed.

More than ninety percent of all peripheral failures occur prior to peripheral buffer availability. These errors may be due to:

1. The attempted operation of a non-operable device. A device is non-operable if it has a mechanical defect, lacks AC power, lacks DC power, is non-existant, or has been switched off-line. A non-operable device normally appears ready (device ready signal is set) until an attempt is made to operate it.
2. The attempted outputting of an illegal character to a typewriter (an illegal character will not necessarily cause a failure on certain types of typewriters.)

1.7.1.1 Instructions

JNR 25X6DD00

JNR transfers program control to the second sequential location when a peripheral device attached to the addressed peripheral buffer is in operation. If all of these devices are ready, program control is transferred to the first sequential location. The test implicity indicates whether or not the last initiated operation (IN, ØUT, IDL, ØDL) is completed.

The JNR signal is the logical AND of the individual device ready signals for all eight devices. A non-operable device will test ready.

The JNR signal my optionally be connected to the Automatic Program Interrupt module to initiate an interrupt whenever the signal changes from not ready to ready. The PERIPHERAL READY lamp, on the computer console illuminates when any device is not ready.

ØUT 2504DD0D

The ØUT instruction may be executed at any time to transfer the right-most seven bits[9] of the A-Register to the N-Register within the peripheral buffer, and initiate one operation of the addressed device. The previous contents of the N-Register is lost. The contents of the A-Register is not changed. Although ØUT may be executed at any time, no action occurs if the peripheral buffer or the addressed device is in operation. If the output device is a paper tape punch, odd parity is generated on seven bits and is punched as a bit in an eighth channel.

[9] If the addressed device is a card punch or card reader, 12 bits of data are transferred.

Figure 26 lists the standard typewriter character set, and typewriter codes.

Figure 23 lists the standard punched paper tape format.

IN 2505DDOD

The IN instruction transfers the current contents of the addressed device's read mechanism to the rightmost seven bits[10] of the A-Register and initiates one operation of the addressed device. The remaining bits of the A-Register are set to zeros by this instruction.

Although IN may be executed at any time, the information transferred is meaningless if the addressed peripheral buffer or device is not ready. The operation of the addressed buffer and device is unaffected, the peripheral buffer error indicator is set and the PB ALARM lamp is turned on.

If the input device is a paper tape reader, these seven data bits are checked for odd parity. Even parity sets the peripheral buffer error indicator and illuminates the PB ALARM lamp. The lamp can be turned off by depressing the CLEAR switch. IN should be immediately followed by a JNE instruction to check for correct parity.

JNE PERIPHERAL 2507DDED

JNE transfers program control to the second sequential location if the error indicator(s) specified by E is reset. If the specified error indicator(s) is set, program control is transferred to the first sequential location and an action specified by E is taken.

[10]- If the addressed device is a card punch or card reader, 12 bits of data are transferred.

E	Error Indicator(s)	Action
0	None	None
1	Addressed Device's Input Demand Flip-Flop	Reset Input Demand Flip-Flop
2	Addressed device's deadman error indicator	None
3	Addressed device's deadman error indicator	Reset Deadman Error Indicator
4	Parity Error Flip-Flop	None
5	Parity Error Flip-Flop	Reset Parity Error
6	Any deadman error indicator in the addressed peripheral buffer or the Parity Error Flip-Flop	None
7	Addressed device's Deadman error indicator or the Parity Error Flip-Flop	Reset deadman error indicator; Reset parity Error Flip-Flop

ACT PERIPHERAL 2501DDOD

ACT forces the addressed device's ready signal to be reset for 8 µs. If the device is ready and if this ready signal is a program interrupt input, the device's program interrupt is requested; otherwise the interrupt is not requested.

If all peripheral devices are ready and the JNR signal is a program interrupt input, the JNR program interrupt is requested; otherwise the interrupt is not requested.

1.7.1.2 Interrupt Operation

Single Interrupt Operation (one peripheral at a time)
The peripheral buffer, with the JNR signal connected to the API module and one buffer driver program IDL or ØDL instructions), operates one device at a time. Interrupt is conditionally requested by the ACT instruction.

GE/PAC 4040
Hardware Required:
One 2-input program interrupt
    Change input: JNR signal (Test Line 1)
    Level input: PBA signal
Program Required:
SPB instruction in interrupt location.
One I/O Buffer driver program
One I/O Buffer list.

GE/PAC 4060

Hardware Required:
One 2-input program interrupt
    Change input: JNR signal (test line 1)
    Level input : PBA PBE signal
Program Required:
IDL (ØDL) instruction in non-inhibitable interrupt location
One I/O buffer list

Multi-interrupt Operation (time shared operation)

The GE/PAC peripheral buffer, with individual device ready signals connected to the API module and individual buffer driver programs (individual IDL or ØDL instructions in the AU2), for each device provides the functional equivalent of an individual hardware controller for each peripheral. Each controller buffers characters between a list in memory and its associated device. Each controller operates it semi-independently of all other devices. Each peripheral requires the use of the peripheral buffer for only the first part of its complete cycle; the buffer can then initiate some other peripheral. This mode allows a greater data transfer rate (e.g., 15 characters can be typed by each of two 15 cps-typewriters in approximately 1 1/30 second). Hardware requirements are on type-2 API input per peripheral device and additional core memory to accommodate individual buffer drivers and character tables. ACT conditionally requests the addressed device's program interrupt.

GE/PAC 4040

Hardware required (per peripheral)
One 2-input program interrupt
    Change input: Device's ready signal ($RDY2_m$)
    Level input: PBA signal
Program required (per peripheral)
SPB instruction in interrupt location.
One I/O Buffer driver program.
One I/O Buffer list.

GE/PAC 4050/60

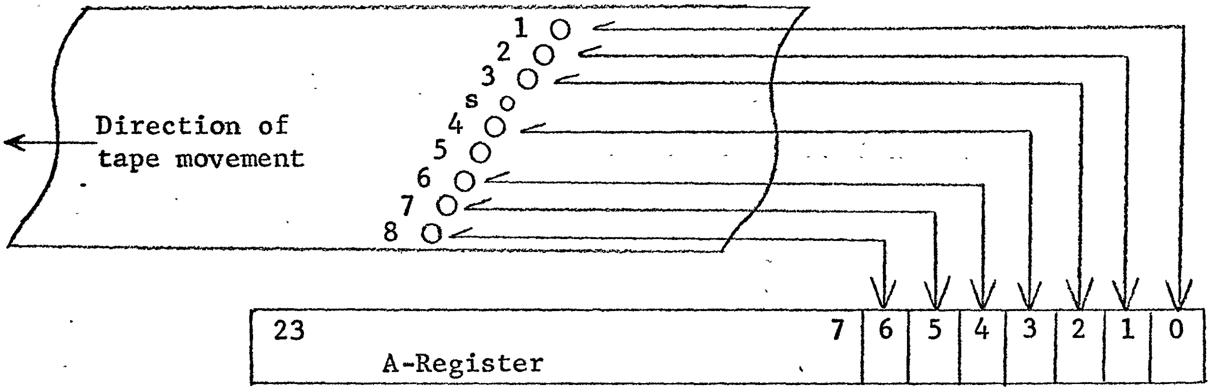
Hardware required (per peripheral)
One 2-input program interrupt
    Change input: device's ready signal ($RDY2_m$)
    Level input: PBA signal for output devices
                 PBA-PBE signal for input devices

GE/PAC 4050/60
Program required (per peripheral)
IDL (ØDL) instruction in device's non-inhibitable interrupt location
One I/O buffer list

1.7.1.3 Error Detection

A device failure or a parity error can be detected by periodically executing a JNE instruction (2507DD60) placed within a frequently entered program. However, a peripheral buffer error (PBE) signal is available for optional use as a program interrupt input (the PBE signal is equivalent to the JNE signal with E=6).

Hardware required:
One 1-input program interrupt:
Change input: PBE signal



An enlarged segment of paper tape is illustrated. Format details are:

1. The paper tape has nine channels, C_1 thru C_8 and C_s running lengthwise along the tape.
2. Channel C_s is the sprocket hole channel. A hole in this channel defines a data frame. Taht is, the hole indicates the presence of data in corresponding punch positions of the other eight channels. Data frames need not be equally spaced. Nominal spacing is 10 frames per inch.
3. A hole in a punch position defines a "1" data bit; no hole defines a zero data bit. Data holes are twice the size of sprocket holes.
4. The IN (ØUT) instruction transmits data between a data frame and bits A_{6-0} of the A-Register as indicated
5. Channel C_5 is the parity channel. A hole is punched as required to quarantee an odd number of holes in each data frame.

Figure 23 - GE/PAC PAPER TAPE FORMAT

Peripheral device	Time lapse between ØUT (IN) instruction and ready signals. Peripheral Buffer Available signal, ms	Device ready signal, ms
Typer, fixed carriage		
normal character, LC	30	65
normal character, UC	95	130
carriage return		up to ___
Typer, long carriage		
normal character	40	100
carriage return		up to ___
Paper tape reader	4ms	10ms
Paper tape punch	4ms	9ms
Card reader		
normal character		
card feed		
Input typer, Selectric	Dependent upon Operator	

NOTE: All times are approximate.

Figure 24- Peripheral Operation Timing

Type size	Number of characters in print line				
Characters per inch	IBM 11" Carriage	Selectric 15½" Carriage	IBM 12" Carriage	MODEL 20" Carriage	B 30" Carriage
10	85	130	88	167	265
12	102	156	106	201	318
14	-	-	124	234	371

NOTE: The vertical line spacing on all typewriters is 6 lines per inch.

Figure 25 - Typewriter Formatting Data

CHARACTER	OCTAL CODE
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	10
9	11
A	21
B	22
C	23
D	24
E	25
F	26
G	27
H	30
I	31
J	41
K	42
L	43
M	44
N	45
Ø	46
P	47
Q	50
R	51
S	62
T	63
U	64
V	65
W	66
X	67
Y	70
Z	71
Space	20
.	33

CHARACTER	OCTAL CODE	NOTE
,	73	
+	60	
-	52	
*	54	
/	61	
=	75	
(	35	
)	55	
$	53	
"	76	
>	16	F
<	36	F
[	12	F
]	34	F
:	15	F
;	56	F
'	57	F
#	13	F
@	14	F
&	32	F
\	37	F
↑	40	F
←	72	F
%	74	F
?	17	F
!	77	F
TAB	140	
CR	100	
TAB	141 to 157	OT
CR	101 to 137	OT
Black	160	OT
Red	161	OT
No action	177	R
Punch on	162	R
Punch off	164	R
Control Mode Shift	1XX	IT
Delete	177	P
Stop	170	P,R

NOTE: Characters are applicable on all standard GE/PAC Input/Output devices, unless otherwise specified.

F - Not available on long-carriage typewriter
OT- Output typewriter only
R - Tape reproduction devices only
IT- Input typewriter only (hold this key down while typing a character generates the octal 1XX; this is interpreted as one of 64 control actions). For example, typing a # in this mode generates 113_8 in the input typer's register.
P - Tape preparation device only.

Figure 26 - GE/PAC Characters & Codes

1.7.2. CARD READER (GE/PAC 4242B)

The continuous feed card reader is used with the GE/PAC 4201B peripheral buffer and can be programmed to read data from punched card bulk storage. Data is read one column at a time from standard 80-column punched cards. Card feed is continuous at a maximum rate of 400 cards per minute. Data transmitted to the computer is the 12-bit binary image of a card column. Card row 12 corresponds to bit 11 of a GE/PAC word; Card row 9 corresponds to bit 0.

1.7.2.1 Physical Description

The 4242B Card Reader is a desk mounted Elliott electro-mechanical device which senses data by photo-cell diodes. It is restricted to use the 4201B peripheral buffer and has the following characteristics:

Card feed rate:	350 to 400 cards/min.
Card feed mode:	Serial by column
Column feed period:	
within card	Approx. 1.3 milliseconds
card feed	Approx. 17. milliseconds
Card specifications:	Standard 80-column card; either square or round corners. Refer to Figure 23.
Card hopper capacity:	600 cards max.
Card stacker capacity:	600 cards max. Limited to 400 cards for convenient card removal.
Code:	12-bit column image.
Code conversion:	None
Data validity checking:	None
Read mechanism checking:	Implicit checking via program check of pseudo columns 81 and 84.

Figure 27 illustrates the operator's panel located at the top front of the reader chassis. The functions of these switches are as follows:

AC/DC ON/OFF (rocker-typer switch.)
Setting this switch to the ON position supplies AC/DC power to the reader. In the OFF position, power is removed from the motor and read photocells.

AUTO/MAN (rocker-type switch)
When this switch is in the MAN position the card reader is in an off-line state and can not be controlled from the computer. Attempted program operation will result in the device's deadman error indicator being set.

When this switch is in the AUTO position the reader is computer controlled.

If the switch is set to MAN during active card reading, data reading will cease immediately.

This switch has precedence over the STOP switch.

AUTO DMND (momentary action pushbutton)
This switch is enabled when the AUTO/MAN switch is in the AUTO position and the reader's clutch Flip-Flop is set.

By depressing and releasing this button, the reader's input demand indicator is set; its Ready signal is cycled from ready to not ready and back to ready. The reader's clutch Flip-Flop is interlocked with AUTO DMD so that the pushbutton is disabled when the clutch is on. However, since the clutch is turned off (by program) at column 1 of the last card to be read, this button should never be used while the reader is in operation.

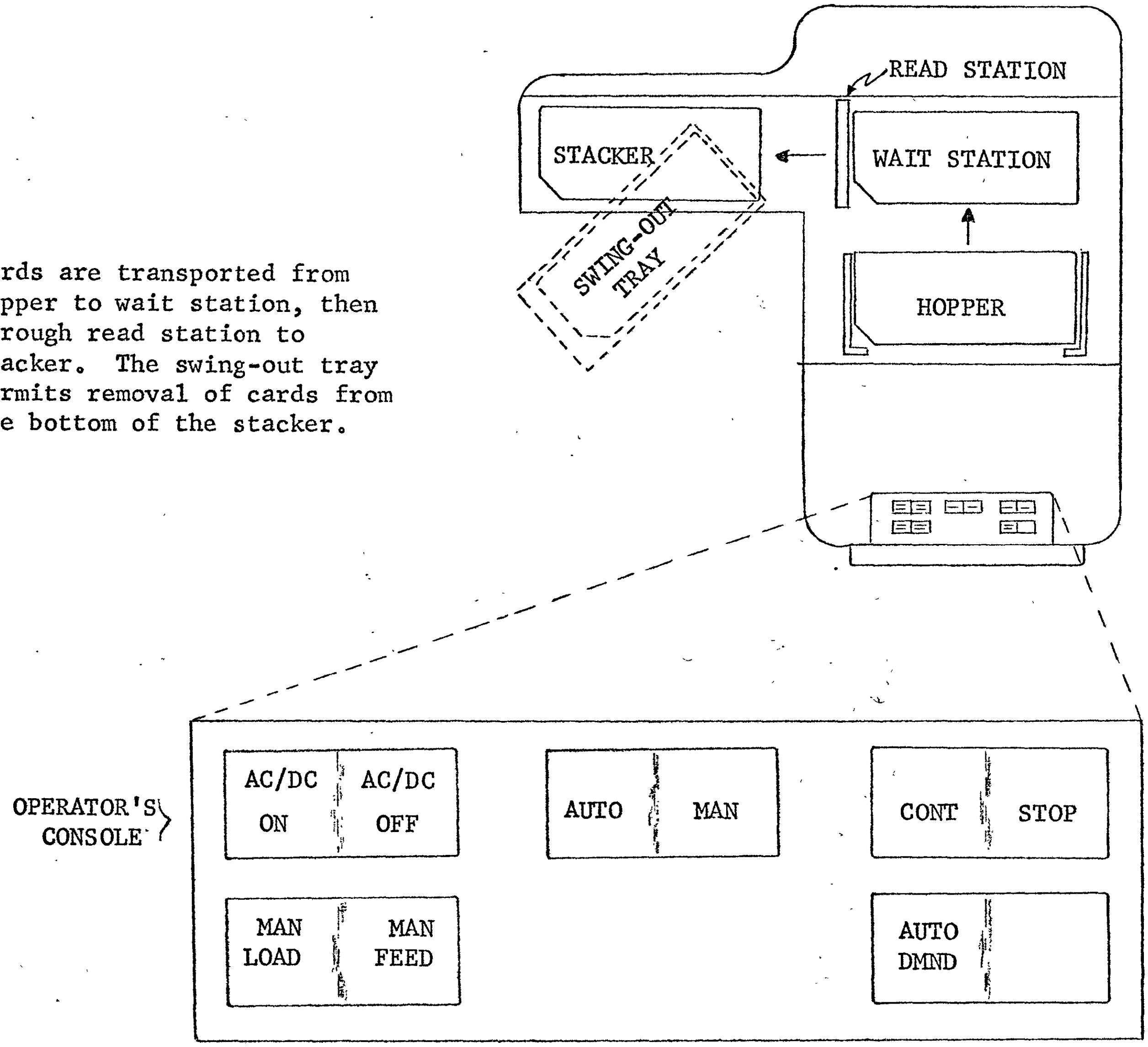
LOAD/FEED (momentary action pushbutton)
This switch is enabled only when the AUTO/MAN switch is in the MAN position.

Depressing LOAD causes one card to be moved from the Card Hopper to the Wait Station (Figure 27).

Depressing FEED causes cards to feed from the Card Hopper through the read mechanism to the card stacker at the rate of 400 cards per minute.

STOP/CONT (rocker-type switch)
When STOP is depressed, feeding is inhibited; data input (from IN, IDL instructions) is not inhibited. If STOP is depressed during card feeding, before column 60 passes the read station, card feed terminates after that card is fed. If STOP is depressed after column 60 passes the read station, one additional card is fed through the read station.

When CONT is depressed, card reading continues. However, STOP must be depressed prior to depressing CONT to release the stop inhibit. Otherwise, no action occurs.



Cards are transported from hopper to wait station, then through read station to stacker. The swing-out tray permits removal of cards from the bottom of the stacker.

Figure 27 - CARD READER (GE/PAC 4242B)

1.7.2.2 Operation

The following procedure is the normal method of initializing the reader for programmed operation:

1. Turn power on.
2. Set AUTO/MAN switch to the MAN position.
3. Insert cards in hopper, 9 edge first, face down.
4. Depress LOAD switch. (First card is transported to wait station.)
5. Set AUTO/MAN switch to AUTO position.
6. Depress AUTO/DMND switch.

The card reader is now on-line, and ready to accept commands from the computer.

Programmed operation of the GE/PAC 4242B card reader is via the 4201B peripheral buffer (see para. 1.7.1) using either the GEN 2 instructions ACT, JNR, JNE, ØUT, and IN or IDL in an interrupt location. (See para. 1.4.3).

Card feeding is initiated by an ØUT command (ØUT READER) which sets the reader's clutch Flip-Flop. Actual card movement lags the ØUT command by approximately 12 milliseconds.

Card feed termination is initiated by an ØUT command (ØUT READER+/10) which resets the reader's clutch Flip-Flop. Actual card movement will continue for approximately 25 to 130 milliseconds following the ØUT command.

When a card column passes under the photocell read heads, that 12-bit data column is ready to be read. The IN (IDL) command to transfer the data must be given within 0.7 milliseconds or the data is lost. The IN and IDL commands transfer data to the AU and affect the device ready signal; these commands <u>do not</u> affect card movement. Card movement timing with reference to the photocells is shown in figure 27.

1.7.2.3 Card Movement Timing (from Wait Station to Stacker)

Card Position	Time (Milliseconds)
ØUT (set clutch Flip-Flop)	0
Card movement begins	12 ms.
Col. 1 ready to read	16 ms.
Cols. 2, 3, 4 and 5	1.3 ms. apart
Col. 5 ready to read	21.2 ms.
Col. 7, 8, ... 85 ready to read	1.5 ms. apart
Col. 80 ready to read	133.7 ms.
Col. 81 ready to read	135.2 ms.
Col. 84 ready to read	139.9 ms.

Col. 1 of second card ready to read	~ 166 ms.
Timing between successive cards (col. 1)	150 ms. apart

The program must synchronize itself to card movement for proper reading of data and must maintain its own column count. The reader's device ready signal provides this capability. This signal normally indicates ready, but is set not ready:

1. 700 µs prior to advent of column 1 under read heads, and
2. by each IN (IDL) instruction.

The signal is reset to ready,by signals originating from the card reader's electro-mechanical transport mechanism which indicate that a card column is in reading position.

If no IN instructions are executed during card feed, the device ready signal cycles once for each card transported through the reader, the pulse occurring at column one of each card. If k IN's are executed per card, the signal will cycle k + 1 times (consequently, if k is less than 84, the remaining columns of the card are not read). If an IN is not given within 0.7 milliseconds of the signal's changing from not-ready to ready, that data column moves past the read station and is lost.

If the program gives 84 properly synchronized IN's, the eighty-first data column is 0000_8 and the eighty-fourth is 7777_8. If these IN's input other than 0000_8 and 7777_8 respectively, the program must presume either hardware failure or improper program-to-hardware synchronization. If more than 84 IN's are given, the ready signal continues to cycle with 1.5 millisecond period as long as IN's are given, until 700 µs prior to the advent of column 1 of the following card. Since the total number of these cycles is variable, there would be no means (other than unique data in column 1) to uniquely signal column 1 of this next card.

Termination of programmed card feeding is initiated by the GEN 2 instruction ØUT READER+/10. This command does not inhibit data input; data may be inputted (IN) as long as card columns continue to pass by the card reader's read station.

If this ØUT command is given between columns 1 and 50 of a card, card feeding will cease at the end (column 85) of the same card. If this command is given following column 60, card feed will cease at the end of the following card. If this command is given between columns 50 and 60, the cessation of card feed will be uncertain.

Cards may be fed one-by-one by the repeated program execution of the instructions ∅UT READER and ∅UT READER+/10. The ∅UT to stop-card-feed must be given before column 50 (but not before the device ready signal is set hot ready for column 1). If the ∅UT to feed the next card can be given prior to column 60 of the current card, the card feed rate is 400 cpm. If a program cannot give this start-card-feed ∅UT prior to column 60, it should delay giving it until after column 85 has passed the read-station to maximize reader life. In this event, the card feed rate is 350 cpm.

The reader may become non-operational for various reasons:

1. Input hopper empty
2. Output stacker full
3. AUTO/MAN switch in MAN position
4. AC/DC power switch in OFF position
5. CONT/STOP switch in STOP position

Error detection and control must be by program. Examples of error control via program are:

1. Reader status tests
2. Sum checks
3. End of card checks (pseudo col. 81-84).

1.7.2.4 Input Hopper Empty

When the input hopper becomes empty as signaled by the hopper empty contact, the last card cycle is completed normally, a deadman error occurs, and card feeding is inhibited. To continue operation of the reader, the operator must set the AUTO/MAN line switch to the MAN position, reload the input hopper and return the AUTO/MAN switch to the AUTO position.

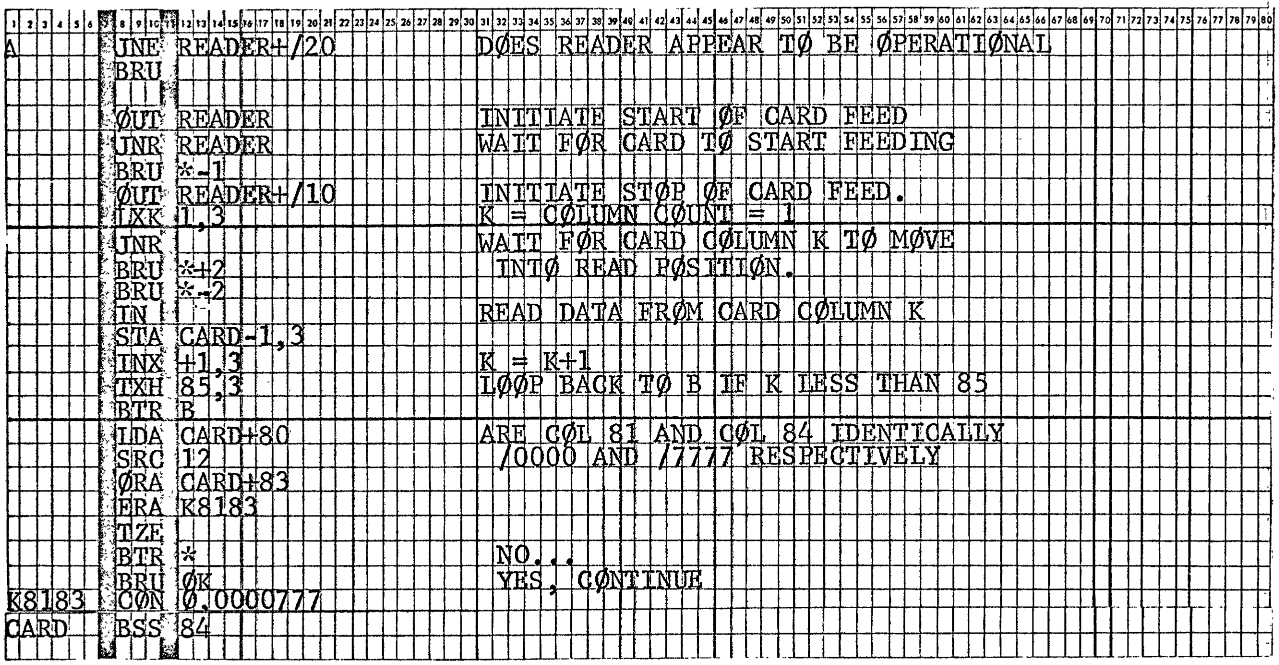
1.7.2.5 Stacker Full

When the output stacker becomes full as signaled by the stacker full contact, a deadman error occurs and further card feeding is inhibited. To continue operation of the reader, the operator must set the AUTO/MAN line switch to the MAN position, remove cards from the stacker and return the AUTO/MAN switch to the AUTO position.

Programming Examples

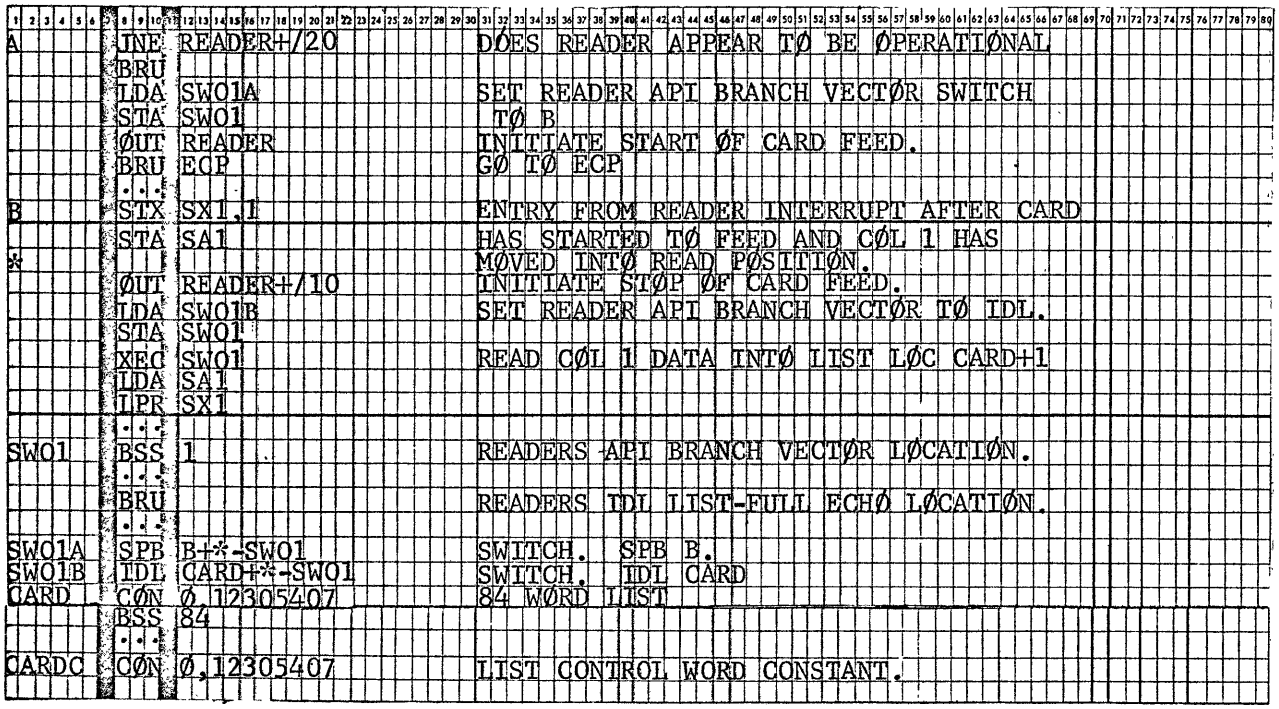
Figure 28 is an example of program for reading one card using the JNR instruction.

Figure 29 is an example program for reading one card using program interrupt and the GE/PAC 4050/4060 instruction IDL.



```
A      JNE READER+/20         DØES READER APPEAR TØ BE ØPERATIØNAL
       BRU

       ØUT READER             INITIATE START ØF CARD FEED
       JNR READER             WAIT FØR CARD TØ START FEEDING
       BRU *-1
       ØUT READER+/10         INITIATE STØP ØF CARD FEED.
       LXK 1,3                K = CØLUMN CØUNT = 1
       JNR                    WAIT FØR CARD CØLUMN K TØ MØVE
       BRU *+2                 INTØ READ PØSITIØN.
       BRU *-2
       IN                     READ DATA FRØM CARD CØLUMN K
       STA CARD-1,3
       INX +1,3               K = K+1
       TXH 85,3               LØØP BACK TØ B IF K LESS THAN 85
       BTR B
       LDA CARD+80            ARE CØL 81 AND CØL 84 IDENTICALLY
       SRC 12                   /0000 AND /7777 RESPECTIVELY
       ØRA CARD+83
       ERA K8183
       TZE
       BTR *                   NO...
       BRU ØK                  YES, CØNTINUE
K8183  CØN Ø,0000777
CARD   BSS 84
```

Figure 28 - CARD READ PROGRAM



```
A      JNE READER+/20         DØES READER APPEAR TØ BE ØPERATIØNAL
       BRU
       LDA SW01A              SET READER API BRANCH VECTØR SWITCH
       STA SW01                TØ B
       ØUT READER             INITTIATE START ØF CARD FEED.
       BRU ECP                GØ TØ ECP
       ...
B      STX SX1,1              ENTRY FROM READER INTERRUPT AFTER CARD
       STA SA1                HAS STARTED TØ FEED AND CØL 1 HAS
*                             MØVED INTØ READ PØSITIØN.
       ØUT READER+/10         INITIATE STØP ØF CARD FEED.
       LDA SW01B              SET READER API BRANCH VECTØR TØ IDL.
       STA SW01
       XEC SW01               READ CØL 1 DATA INTØ LIST LØC CARD+1
       LDA SA1
       LPR SX1
       ...
SW01   BSS 1                  READERS API BRANCH VECTØR LØCATIØN.
       ...
       BRU                    READERS IDL LIST-FULL ECHØ LØCATIØN.
       ...
SW01A  SPB B+*-SW01           SWITCH.  SPB B.
SW01B  IDL CARD+*-SW01        SWITCH.  IDL CARD
CARD   CØN Ø,12305407         84 WØRD LIST
       BSS 84
       ...
CARDC  CØN Ø,12305407         LIST CONTROL WORD CONSTANT.
```

Figure 29 - CARD READ PROGRAM

1.7.3 LINE PRINTER (GE/PAC 4262)

The GE/PAC 4262 line printer produces printout in accordance with specified data from an external source. It is used with the peripheral buffer and can be programmed to print hardcopy documentation in multiple copy form. Each print action can record a maximum of 120 characters per line at a nominal rate of 300 lines per minute.

Each print operation is a three phase action:

1. Transfers, under arithmetic unit control, the desired line image from computer memory to the printer's internal buffer memory. The line image is variable length. It consists of a line space control code, followed by the desired ordered sequence of character codes and terminated by a print command code.

2. Printing action followed by resetting the printer's buffer memory to spaces.

3. Paper feed of a number of lines (or to page position) specified by the line space code.

1.7.3.1 Physical Description

The 4262 line printer is an Anelex, Series 5, asynchrous buffered console printer with the following characteristics:

Printing rate: 300 lines per minute-nominal

Text Specifications:

Columns per line: 120
Horizontal character spacing: 10/inch
Vertical line spacing: 6/inch
Average character height: 0.10 inch

Paper Form Specifications:

Form width: 4 to 20 inches

Form length: 1 line to 22 inches

Multiple Copy Capability:

The number of acceptable copies varies with the type of paper.

Any number of copies up to original plus five carbons can be provided with proper choice of paper.

Paper Feed and Line Spacing:

Paper advances after printing.

Slewing is under control of a vertical format unit tape.

Printer option: paper loading and receiving baskets

Figure 30 illustrates the operators control panel mounted on the printer chassis. The functions of these indicators and backlighted pushbuttons follow:

ON Backlighted green round pushbutton.
Depressing this button provides AC power to the printer. Indicator glows when power is on and the printer buffer is ready to receive data.

OFF Backlighted red round pushbutton

YOKE OPEN Backlighted red square indicator.
This indicator illuminates when the printer yoke is not fully seated in the closed position. The printer yoke is located in front of the print drum.

ALARM STATUS Backlighted red square indicator.
This indicator illuminates when the printer fails to become ready after depressing the ON pushbutton. It provides an indication of some error.

NO PAPER Backlighted red square indicator.
This indicator illuminates when less than 30 lines of paper supply remain. On-line printing, one line at a time for each depression of the START pushbutton, is permitted while the low paper condition exists, until the paper supply is exhausted.

TOP OF FORM Backlighted yellow square pushbutton.
Depress this pushbutton to slew the paper form to the top-form (ie. the first print line position of the next form).

TRACTOR INDEX Backlighted yellow square pushbutton.
Depress this pushbutton to align the paper feed tractors with the vertical format unit. Once this is complete, the vertical format tape loop may be installed.

TEST PRINT Backlighted red square pushbutton.
Depress this pushbutton to initiate repeated printout of a test character. Operation is halted by depressing the STOP pushbutton. This is a maintenance operation. The TEST PRINT button is inoperative in the on-line mode. The START and TOP OF FORM buttons are inoperative in the TEST PRINT mode of operation.

START Backlighted green square pushbutton.
Depress this pushbutton to switch the printer on-line and cause the printer's buffer memory to be cleared to SPACEes.

STOP Backlighted red square pushbutton.
Depress this pushbutton to stop printer operation.

a) If the printer is on-line, it is switched off-line. If the printer is in operation or has received at least one code in its buffer, the switching is automatically delayed until completion of the printing and slewing action; however, if the print command code is not received within 300 MS, a print command is forced.

b) If the printer is off-line and operating in the TEST PRINT mode, depress STOP to terminate the TEST PRINT operation.

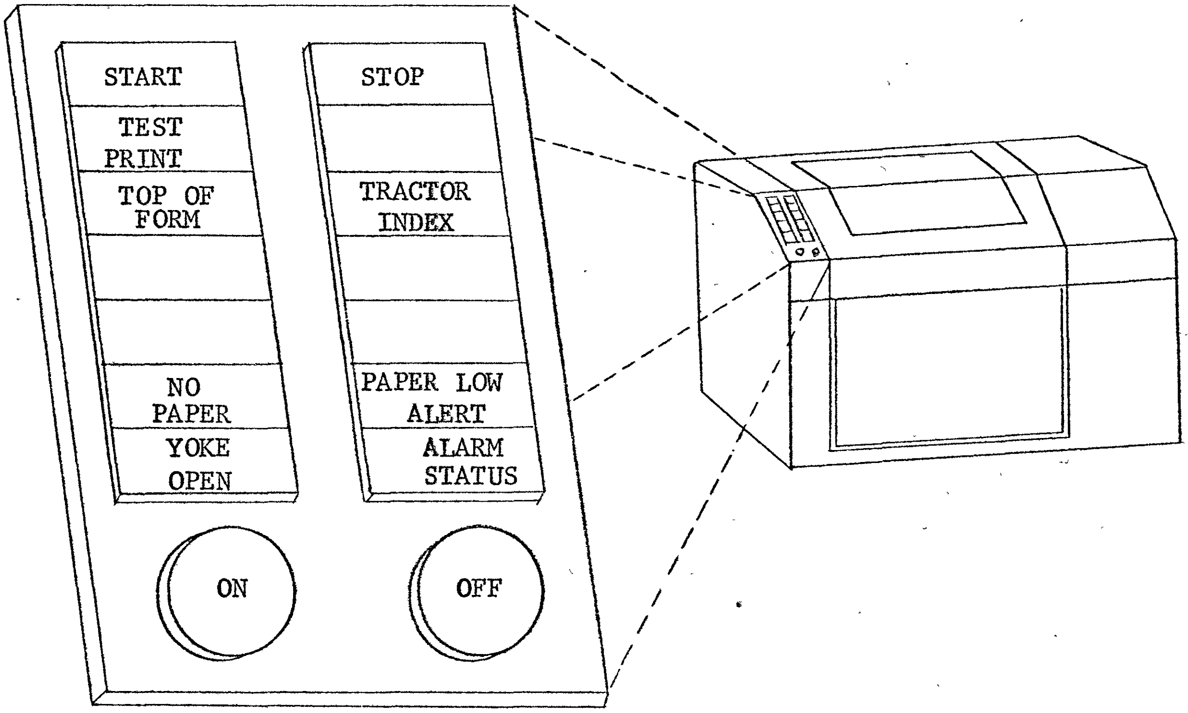


Figure 30 - OPERATORS CONTROL PANEL

1.7.3.2 Output Formatting Information

The printer buffer memory consists of 122 6-bit character code positions. The nominal print line length is 120 character graphics.

Each code position in the buffer initialized to a SPACE (20_8) code following a print action or by depressing the START pushbutton on the printer's operators panel.

The central processor transmits 7-bit GE/PAC codes to the printer controller via the peripheral buffer at a rate synchronized by the printer's device ready signal. The controller assembles the lower 6-bits of these codes (00_8 through 77_8) in order of receipt into the buffer from left to right. The leftmost code position (the line space code) is loaded first. The receipt of any code between 100_8 and 177_8 terminates the transmission of further characters. The lower six bits of this print command code is OR'ed with 20_8 and the result loaded into the next code position of the buffer. The code should normally be a carriage return (100_8) to cause a SPACE code to be loaded into the buffer. The remaining code positions are already SPACEes; the print action follows.

The print line image in programmable memory may vary in length from 2 to 122 code positions. If the desired line of output consists of less than 120 characters, it is not necessary to pad the line image with SPACEes.

The first code position of the line image must contain a non-printing line space code. Paper advances after printing and the number of lines skipped after printing is specified by the line space code and the 8-channel tape loop in the vertical format unit (VFU) on the printer chassis. The line space code specifies which channel of the VFU tape loop will control the paper slewing. Single line spacing requires 12ms to complete. Slewing of paper occurs at the rate of 18 inches per second. The maximum slewing distance is from top-form to top-form.

The line space codes and the corresponding slewing actions follow:

Line Space Code	Slew Action Following Printing
00	Slew to top form
01	Single space
02 03 04 05 06 07	Slew to form position specified by channel k of VFU tape (k = line space code + 1).
77	No paper advance

The vertical format tape is an endless loop of one inch wide tape and is used for format control. The tape may be paper or mylar, etc. Eight channels on the tape provide a Top-of-form position and seven other form-positions.

The length of the VFU tape must be an integral multiple of the length of the form, and must be between 11 and 22 inches long. The VFU tape has punch positions spaced six per inch to match the line spacing on the form. A hole in a punch position of a given channel defines a form position for that channel. For example, channel 2 has holes corresponding to every line of the form.

The line space code is followed in the line image by 0 to 120 character codes. The legal printing characters are the graphics corresponding to the codes between 00_8 and 77_8 listed in Figure 26.

The line image is terminated by the print command code described above.

Character seqeunces are printed left justified unless preceded by SPACE characters in the line image. It is possible to print different fields of the same line with separate print commands if the line space code is a 77_8 for each field except the final field.

1.7.3.3 Operation

Programmed operation of the GE/PAC 4262 line printer is via the peripheral buffer (see para. 1.8.1) using either the GEN 2 instructions ACT, JNR, JNE, and ØUT or the ØDL instruction (see para. 1.4.3).

The printer's device ready signal is set when the printer's controller is not busy and is able to accept another character code. The signal is reset by each ØUT (ØDL) and remains reset during the 30 to 60 µs while the controller is loading a code into the printer's buffer and during the 194 ms of the mechanical print action. Receipt of the print command code will inhibit further loading of the buffer until the print action is complete.

For the printer to operatre at its nominal rated speed, the entire line image must be transferred to the printer's buffer within 12 ms following the completion of the previous mechanical print action. If the buffer loading time exceeds 12 ms, the printer speed is calculated as follows:

$$\frac{60000}{182\ T} = \text{lines per minute}$$

T is the loading time in milliseconds.

If the printer's device ready signal remains reset for more than 4 to 8 seconds, its deadman error indicator is set. The indicator can be tested by a JNE instruction with $E = 2_8$ or $E = 3_8$ (see para. 1.7.1.1).

A deadman error may be due to one or more of the following faults:

1. Printer off line
2. Paper supply is low
3. Broken paper
4. Open yoke
5. Printer or Computer Power failure
6. Blown fuses
7. Printer-to-Computer cable not plugged in

1.7.3.4 Special Programming Considerations

1. During the loading of the printer buffer, the printer's device ready signal cycles at an extremely rapid rate. It is reset by the ØUT (ØDL) instruction and set within the next 50 microseconds. This means that the central processor can give an ØUT (ØDL) to the printer at 76-microsecond intervals.

   Momentary loading on the GE/PAC 4060 central processor, during the 9 ms period that the buffer is being loaded from an ØDL list,is as high as 32%. Average loading over a complete print cycle is 1.5%.

2. If a program erroneously outputs more than 122 codes to the printer, the 123rd and subsequent codes are lost. The printer continues to accept and discard these codes until a print command is received. The 120 codes in the print positions of the buffer are printed.

1.7.4. INPUT/OUTPUT TYPER - GE/PAC 4270B

The I/O typer, used with the peripheral buffer, can be programmed to provide an intercommunication interface between an operator and the computer. The I/O typer has three mutually exclusive modes of usage:

1. Standard Typer

   The typer, when manually switched off-line, is electrically disconnected from the computer. It can be used manually as a standard fixed carriage typer. When the typer is in this MANUAL mode, it is unavailable to the computer.

2. On-Line Output Typer

   The typer, when manually switched on-line to the AUTO mode, can be program operated as a fixed carriage output typer. Operation is identical to the GE/PAC 4221B output typer. A pushbutton allows the operator to signal the program when he wishes to use the typer as an input device. The remainder of the keyboard is inoperative in this mode.

3. On-Line Input Keyboard

   The typer, when manually switched on-line to the AUTO mode, can be program operated as an input keyboard to allow an operator to type data into the computer's arithmetic unit, one character at a time. Suitable hardware and program interlocks are provided to make it impossible to lose input data. A typed record of the data input is generated.

1.7.4.1 Physical Description

The I/O typer consists of a standard fixed-carriage IBM Selectric typer equipped with the following manual controls:

1. A two position power-on/power-off switch marked ON and OFF. This switch controls AC power to the typer's motor; it does not control DC logic power to the typer's controls.

2. A backlighted double acting (push-push) pushbutton marked AUTO. It is illuminated when in the on-line mode. When the switch is not illuminated, the I/O typer is in the manual mode (off-line), the DC power from the computer is turned off, or the lamp within the switch is burned out.

3. A backlighted single acting INPUT pushbutton. It can be illuminated in the on-line mode by the program control to indicate to an operator that the keyboard is unlocked and that the program is waiting for the operator to type on the input keyboard.

4. A single acting pushbutton marked CNTRL. If depressed during the time an input character is typed, a binary "1" is generated in the most significant data position of the typer's 8-bit data register. The character's code is generated in the lower 6 data bits of the data register. Odd parity is normally generated in the 8th bit of the data register.

Figure 31 illustrates the I/O typer keyboard which displays the layout of the foregoing controls with the character keys.

1.7.4.2 Operation

Operation of the I/O typer requires two power sources.

An AC source powers the typer's mechanical operation (i.e., its motor). This source is manually switchable by the ON/OFF toggle switch on the typer. The AC power should remain on at all times. If the AC power is off, an attempt by the computer to output a character (ØUT) locks the typer's keyboard (if unlocked) and starts the typer's deadman timer timing; it performs no other action. The typer is unable to perform its type action and an eventual deadman error results. If the AC power is off and an input action is attempted (IN), the keyboard unlocks and meaningless data transfers into the arithmetic unit; no error indication is given.

The second power source is DC from the computer central processor.

1.7.4.3 Operation as off-line Typer:

Operation of the typer in the off-line mode requires the following switch positions:

1. The OFF/ON toggle switch must be in the ON position.

2. The backlighted AUTO pushbutton should not be illuminated. If it is illuminated, depress the button to set the manual mode, turn the light off and unlock the keyboard. If the pushbutton is not lighted, the operator should press the switch twice to cycle it to Auto and back to Manual; the pushbutton is illuminated and then extinguished.

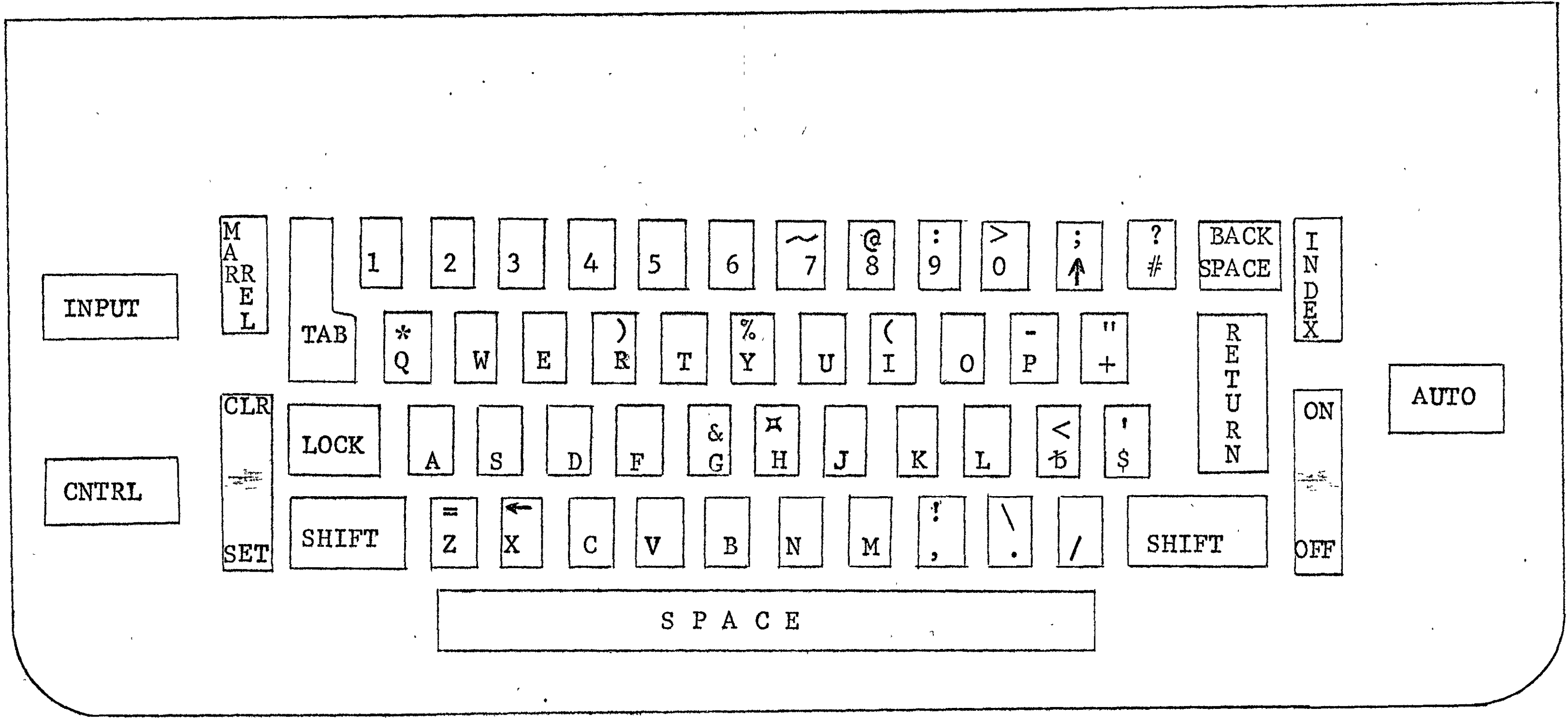


Figure 31 - MODEL 4270 INPUT KEYBOARD

A false Manual indication, resulting from a burned-out lamp, is detected by pressing the switch twice.

In the Manual mode, the typer is a standard Selectric typer; all switches and keys except the INPUT and CNTRL pushbuttons are operative.

Any attempt by the program to output (ØUT) or input (IN) to this typer in the Manual mode is ignored; the typer's Deadman Error Flip-Flop is set.

1.7.4.4 Operation as an on-line Device:

Operation of the typer in the on-line mode requires the following switch positions:

1. The ON/OFF toggle switch must be in the ON position.

2. The backlighted AUTO pushbutton should be illuminated. If the pushbutton is not illuminated depress the button to set the AUTO mode, turn the light on, automatically lock the keyboard, and clear the typer's data register. If pushing the button does not turn the lamp on, the lamp should be replaced.

   In the on-line mode, only the ON/OFF switch, the AUTO pushbutton, and the INPUT pushbutton are normally operative.

1.7.4.5 Operation as an on-line Output Typer:

Program operation is identical to that of the GE/PAC 4221B output typer. Hardware operation differs in two ways:

1. Each ØUT instruction locks the keyboard (if unlocked).

2. The code corresponding to the character graphic actually printed is recorded[11] in the typer's data register. Therefore, if an IN instruction follows an ØUT instruction, the program can perform a program echo check. An echo check determines if the character actually printed is the same as the character that was transmitted to the output buffer for printing. The check is accomplished by reading this code (IN) and comparing it to the code transmitted (ØUT) to the N-Register.

[11] The control characters PRINT-RED and PRINT-BLACK are the only exceptions. The data register is unchanged when these characters are typed.

> The last character typed must be a lower case character to allow normal operation of the typer in the off-line mode.

Since the INPUT pushbutton is operative in the on-line output mode, the operator may signal the program desired to use the I/O typer as an input device. Pushing the INPUT pushbutton simulates the ACT instruction (see paragraph 1.7.1.1), cycling the Typer's Device Ready signal, and sets the I/O typer's Input Demand Flip-Flop. Pushing the INPUT button does not change the I/O typer's mode of operation. It remains in the output-only mode until the program gives an IN instruction.

1.7.4.6 Operation as an on-line Input Keyboard:

The program controlling the on-line operation of the I/O typer is responsible for detecting any cycling of the typer's device ready signal (via JNR instruction or program interrupt), interrogating its Input Demand Flip-flop, and deducing the cause for the cycling (See Table 1). The Input Demand Flip-Flop is interrogated by a JNE DEVICE /10 instruction (see paragraph 1.8.1.1). The instruction reset the Flip-Flop following interrogation.

When the I/O typer is used as an input keyboard, the following operating procedures apply:

1. Depress the INPUT pushbutton.

2. The program senses the resulting cycling of the typer's device ready signal. The program interrogates the typer's Input Demand Flip-Flop at its discretion, and gives an IN command to permit the operator to use the input keyboard. The IN instruction transfers the previous meaningless contents of the typer's data register into the arithmetic unit and unlocks the typer's keyboard; it disables the INPUT pushbutton, enables the CNTRL pushbutton, and turns on the INPUT lamp. The typer's deadman timer does not begin timing (see paragraph 1.7.1).

3. The operator may then use the input keyboard to type a character's code (plus odd parity) into the typer's eight-bit data register. There are 128 input characters, 64 correspond to the GE/PAC character set and 64 correspond to the joint usage of the CNTRL pushbutton and the typer's keyboard (see Figure 31).

Table 1 - ON-LINE STATUS OF I/O TYPER

I/O typer's program status	Control Switch Status					Possible causes for a cycling of I/O Typer's Device Ready Signal
	Before/After Cycling	Keyboard	Input Button	Input Lamp	Input Demand Flip-Flop	
Idle	Before	locked	enabled	off	reset or set	ACT instruction
	After	locked	enabled	off	unchanged	
	Before	locked	enabled	off	reset	1) INPUT pushbutton pressed;
	After	locked	enabled	off	set	2) Also possibly an ACT instr.*
In use as an output device	Before	locked	enabled	off	reset or set	1) Typing of the last character is completed,
	After	locked	enabled		unchanged	2) Also possibly an ACT instr.*
	Before	locked	enabled	off	reset	1) Typing of last character is completed and INPUT pushbutton pressed;
	After	locked	enabled	off	set	2) Also possibly an ACT instr.*
In use as an input device	Before	locked	enabled	off	reset or set	ACT instruction
	After	locked	enabled	off	unchanged	
	Before	locked	enabled	off	reset	1) INPUT pushbutton pressed,
	After	locked	enabled	off	set	2) Also possibly an ACT instr.*
	Before	unlocked	disabled	on	reset or set	ACT instruction
	After	unlocked	disabled	on	unchanged	
	Before	unlocked	disabled,	on	reset	1) Operator typed a character into the typers'data register
	After	locked	enabled	off	set	2) Also possibly an ACT instr.*

* The ACT instruction possibly occurred in action to the first action.

Initiation of the type action (depressing any key on the typer) resets the typer's device ready signal and starts its deadman timer timing. Successful completion of the type action (the printing of the character's graphic) generates the character's code in the typer's data register and performs the following actions:

1. Sets Input Demand Flip-Flop
2. Locks keyboard
3. Enables INPUT Button
4. Disable CNTRL Button
5. Extinguishes INPUT lamp
6. Sets device ready signal
7. Disables deadman timer

4. The program must sense the cycling of the typer's device ready signal. Action is the same as 2 above, except that the data input is meaningful.

5. The program must alternately effect steps 3 and 4 above, to input a sequence of characters. It then interprets the sequence as prescribed by the specific program's console operation specifications and performs the prescribed action.

1.7.4.7 Special Programming Considerations

1. The program has direct access to the typer's device ready signal if each peripheral has its own device ready interrupt. If a system uses one of the other peripheral buffer options (JNR signal, one interrupt, two interrupts, three interrupts) the program must deduce which of several device ready signals actually cycled. For every input demand or an input keyboard action, however, the appropriate device's Input Demand Flip-Flop is set.

2. In paragraph 1.7.5.1, the I/O typer is described as capable of three mutually exclusive modes of operation. Although this is true in the strict sense, a properly designed program can alternately operate the typer in the on-line mode as an input keyboard and an output typer. This gives (in the human frame of reference) the impression of the instantaneous man-to-program intercommunication.

3. The space character typed in the output mode has the unique graphic blank corresponding to the code 20_8. The space character generated from the input keyboard is not unique. Depressing the space bar prints the graphic blank and generates the code 20_8. Depressing the key prints the graphic b̸ and generates the code 20_8.

4. Pushing the b̸ key when the CNTRL pushbutton is depressed generates 120_8 with correct odd parity. Pushing the Space Bar when CNTRL is depressed generates 120_8 with incorrect even parity.

5. The INPUT pushbutton should always be depressed momentarily. Continuing to depress the pushbutton longer than 4 to 8 seconds forces a deadman error; however, the error is automatically cleared when the pushbutton is released.

1.8 GE/PAC CONSOLE

The GE/PAC programmer's console is an integral part of the central processor. This console is one means for the programmer and maintenance man to communicate with the computer in actual machine language. Plant operating personnel are not required to, nor expected to, know how to use the programmer's console in process control applications. Once the programming has been completed, little use will be made of the programmer's console except for the displaying various registers for maintenance personnel.

The underlying philosophy toward the use of consoles in the GE/PAC system is that each module of the system may have its associated console. More specifically, the arithmetic unit has a programming console which contains displays and functions only related to the arithmetic unit. System displays and functions are included on a separate system console.

1.8.1 CONSOLE DESCRIPTION (4040)

Twenty-four lights and console switches are used to display and enter data or commands into the computer. These lights and switches are divided into groups of three to represent eight octal digits for programming convenience. Two modes of operation are possible: automatic and manual. The MAN/AUTO/OFF CONSOLE switch controls the mode. It is located in the lower center portion of the console. During automatic operation, the console switches can be read into the A-Register only by the programmed instruction Read Console Switches (RCS) 25050000_8. If a console switch is down, a "1" is set in that bit position in the A-Register; if up, the contents of A are not changed. The console switches are sometimes referred to as break-point switches during automatic operation. The break-point refers to a decision that can be made as the result of a switch or switches being set (down). On-line routines which use the console switches are normally called by the DEMAND Button. This sets the Demand Flip-Flop (DEMF). The programmed instruction Jump If No Demand (JND) 25040000_8 is interrogated periodically to determine if the switches should be read and deciphered.

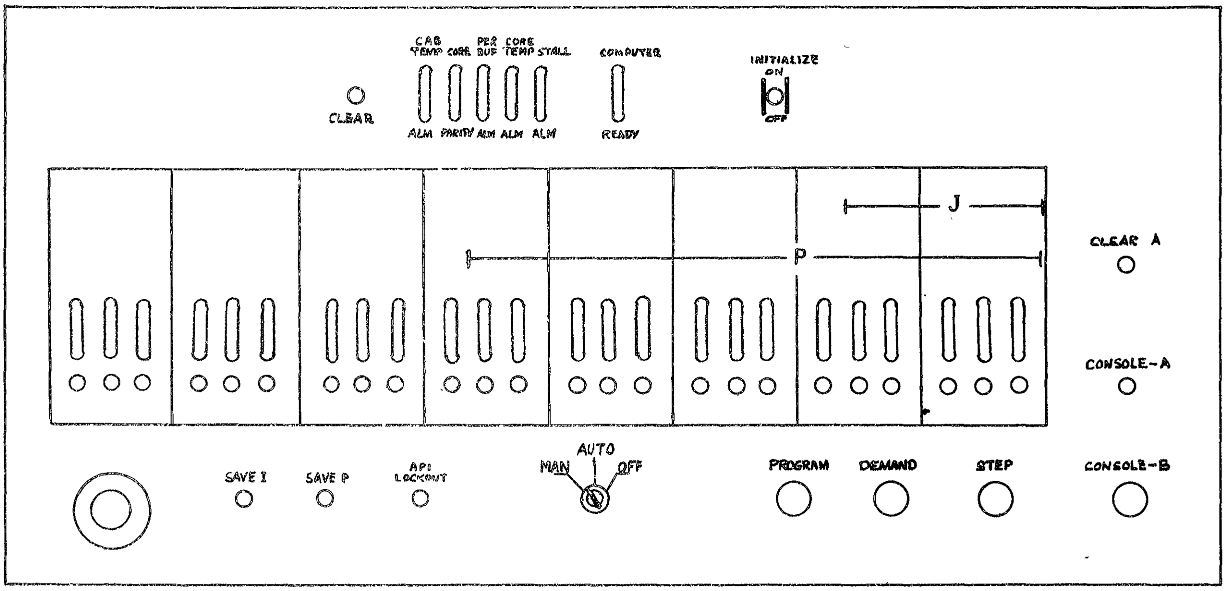


Figure 32 - GE/PAC CONSOLE AU1 (Model 4040)

In the automatic mode, all console switches and buttons except the following are disabled:

a. Clear Alarm
b. Power Off
c. Demand
d. Save P
e. Save I
f. Program Switches
g. Register Select Switch

The CONSOLE OFF position allows the console to be disabled when in the automatic (running) mode of operation. A removable key is provided to lockout the console in the automatic mode.

Manual operation of the console enables all console switches and buttons. Manual operation normally involves changes to the instruction register and memory locations. Paragraph 1.8.2 explains this step-by-step procedure.

1.8.2 REGISTER DISPLAYS

The selector switch in the lower left corner of the console allows the following registers to be displayed:

Selector Switch Position	Bit Position Within Display	Registers (Information Displayed)
A - Accumulator	23 - 0	Accumulator Register (Data)
B - Buffer	23 - 0	B - Register (Next sequential instruction as it appears in memory)
I - Instruction	23 - 0	Instruction Register (Instruction last executed)
	23 - 18	Operation Code
	17 - 15	X Word Indicator
	14	Relative Addressing Indicator
	13 - 0	Effective Operand Address
Flip-Flops	23 - 18	Status Flip-Flops
	23	Demand Flip-Flop
	22	Overflow Flip-Flop
	21	Permit Automatic Interrupt Flip-Flop
	20	Test Flip-Flop
	18	Peripheral Ready Flip-Flop
P - REGISTER	13 - 0	Program Counter (Program Control Address)
J - J Counter	23	S_1 Control Sequence State
	22	S_2 Control Sequence State
	21	S_3 Control Sequence State
	20	C_1 Control Sequence State
	19	C_2 Control Sequence State
	18	C_3 Control Sequence State
	4 - 0	J Counter
AUX - Auxiliary Switch		Optional Displays

ALARM LIGHTS

The following alarm lights are displayed on the console:

a. Core Parity
b. Peripheral Error (PB ALARM)
c. Core Temperature
d. Cabinet Temperature
e. Stall Alarm
f. Computer Ready

1.8.3 CONSOLE DESCRIPTION (4050 and 4060)

Data is displayed and entered into the computer via twenty-four lights and switches located on the front of the computer cabinet. These lights and switches are divided into groups of three to represent eight octal digits for programming convenience. Two modes of operation are possible: automatic and manual. The MAN/AUTO/CONSOLE OFF switch controls the mode. During automatic operation, the console switches can be read into the A-Register only by the programmed instruction Read Console Switches (RCS). On-line routines which use the console switches are normally called by the DEMAND button. This sets the Demand Flip-Flop (DEMF). The programmed instruction Jump If No Demand (JND) is periodically interrogated to determine when the console switches should be read and deciphered.

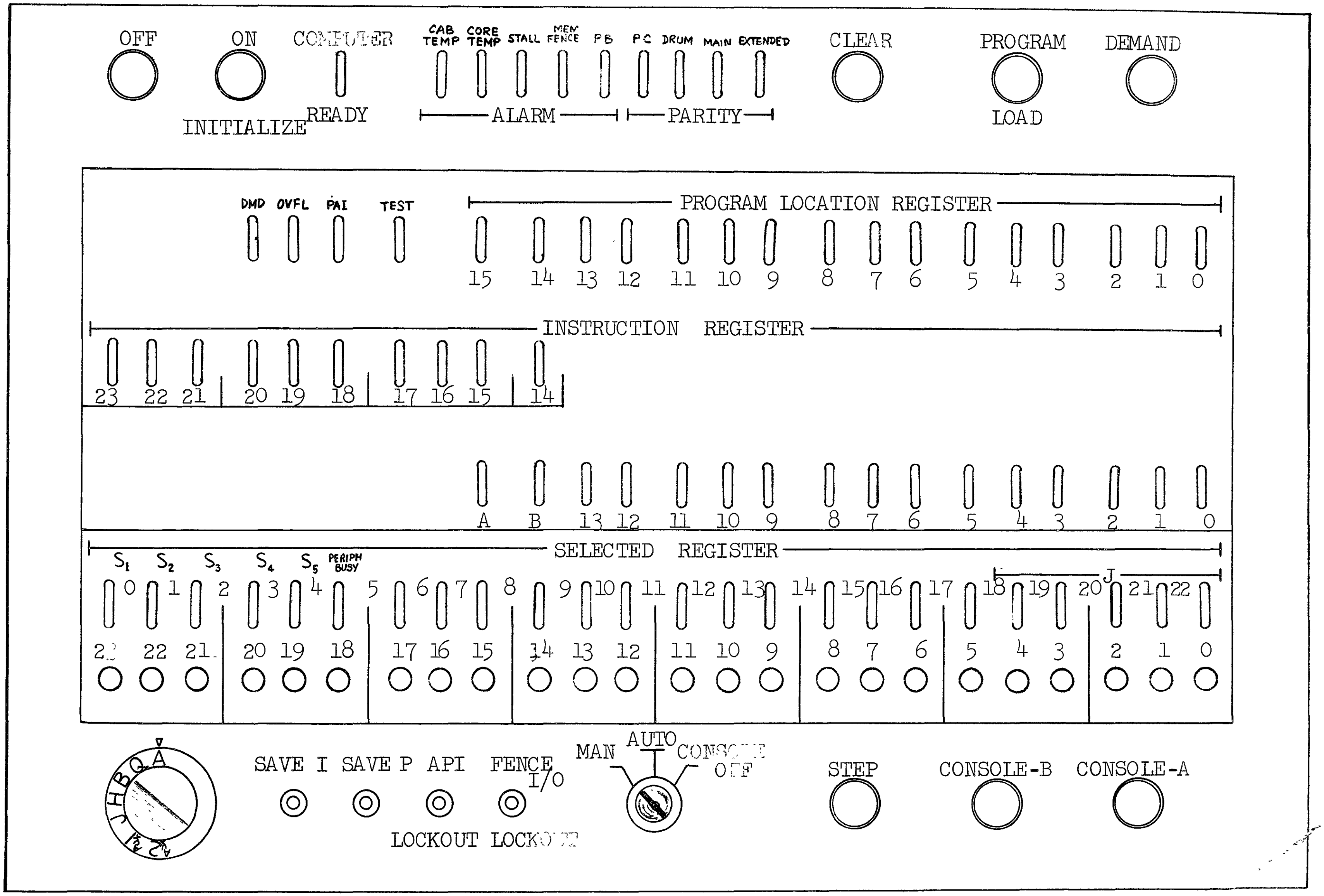


Figure 23 - GE/PAC CONSOLE (AU2 Model 4050/4060)

1.8.4 REGISTER DISPLAYS

The selector switch in the lower left corner of the console allows the following registers to be displayed:

Selector Switch Position	Bit Position Within Display	Register (Information Displayed)
A	23 - 0	Accumulator Register (Data)
Q	23 - 0	Auxiliary Accumulator Registe (Combine data with A-Register to for double-length precisio
B	23 - 0	Buffer Register between the A and memory. (Next sequential instruction as it appears in memory)
H	14 - 0	Special purpose register for maintenance use only.
J	4 - 0	J Counter
	23	S_1 Control Sequence State
	22	S_2 Control Sequence State
	21	S_3 Control Sequence State
	20	S_4 Control Sequence State
	19	S_5 Control Sequence State
	20	Demand Flip-Flop
	19	Overflow Flip-Flop
	18	Permit Automatic Interrupt Flip-Flop
	17	Test Flip-Flop
	15 - 0	P - Register (permanently displayed)
AUX1		Optional Display
AUX2		Optional Display

ALARM LIGHTS

The following alarm lights are displayed on the console:

a. Cabinet temperature
b. Core temperature
c. Stall alarm
d. Memory Fence Alarm
e. Peripheral Buffer Alarm
f. Peripheral Controller Parity
g. Drum Parity
h. Main Core Parity
i. Extended Memory Parity
j. Computer Ready

GE-PAC 4000

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER

PROCESS ASSEMBLER LANGUAGE

November 1965 revised

Library-Control No. YPG12M

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. BOX 2918
PHOENIX, ARIZONA - 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

GENERAL ELECTRIC
PROCESS COMPUTERS

GE/PAC 4000 PROCESS ASSEMBLER LANGUAGE

Library Control No. YPG12M

REVISION CONTROL SHEET

APPROVED BY: David R Frost DATE: June 25, 1965

EV.	RECORD OF CHANGE	DATE	REV.	RECORD OF CHANGE	DATE
	Page 1; para 1.1	6/25/65			
	Page 13; para 2.6				
	Page 16; para 2.8.1, step 3				
	Page 17; para 2.8.1, example				
	Page 26; Flag Ø and I				
	Appendices E and F new				
	David R Frost				
	Page 7; CON D and CON F	11/18/65			
	Page 12; para 2.5.1				
	Page 13; para 2.5.3				
	Page 20; para 2.12.1				
	Page 21; para 2.12.2				
	Page 27; added codes 9, 18, 19				
	David R Frost				
	Page 27; Audit Code 9	3/14/66			
	David R. Frost				

	E A Schelling 6/65	C	E A Schelling 3/66			PAGE 1
	E A Schelling 11/65					REISSUED

TABLE OF CONTENTS

INTRODUCTION

This manual defines the standard Process Assembler Language for the GE/PAC 4000 Process Computer System. This manual is intended for the experienced programmer.

Translator programs are available for a variety of hardware configurations. The language may be restricted for some translators. The Library Write-Ups for each translator state if the complete language can be translated. The language definition includes:

- coding form description and usage
- pseudo operation definitions
- hardware operations
- processing method and output

The hardware operations are listed in Appendix D. Detailed definitions of their characteristics are presented in the GE/PAC 4000 Instruction Reference Manual.

INTRODUCTION

This manual defines the standard Process Assembler Language for the GE/PAC 4000 Process Computer System. This manual is intended for the experienced programmer.

Translator programs are available for a variety of hardware configurations. The language may be restricted for some translators. The Library Write-Ups for each translator state if the complete language can be translated. The language definition includes:

- coding form description and usage
- pseudo operation definitions
- hardware operations
- processing method and output

The hardware operations are listed in Appendix D. Detailed definitions of their characteristics are presented in the GE/PAC 4000 Instruction Reference Manual.

1. *STATEMENT FORMAT*

Assembly program input information is written on the "Process Language Statement Coding Form" (Figure 1). Each line on the coding form represents one instruction to the assembler. The coding form is comprised of four Fields defined in the ensuing paragraphs.

1.1 LOCATION FIELD - Columns 1 thru 6

The Location Field is used to identify the location of an instruction. A name written in this field becomes associated with the instruction written on the same line. Any reference to the instruction may be made by that name. Names used in the Location Field must consist of six or fewer alphanumeric characters; the first of which must be alphabetic and start in column one. A decimal point is considered as an alphabetic character in this context. *

1.2 OP CODE FIELD - Columns 8 thru 10

The Op Code Field contains a two or three character operation code which identifies the operation to be executed. The legal operations include the pseudo operations described in Section Two and the Assembler Instructions outlined in Appendix D.

1.3 OPERAND FIELD - Columns 12 thru 68

The Operand Field may contain a combination of parameters. When merged, the parameters define the operand or operands required by the operation code. Each operand is formed by parameter groups. The following four basic parameter types are permitted in this field.

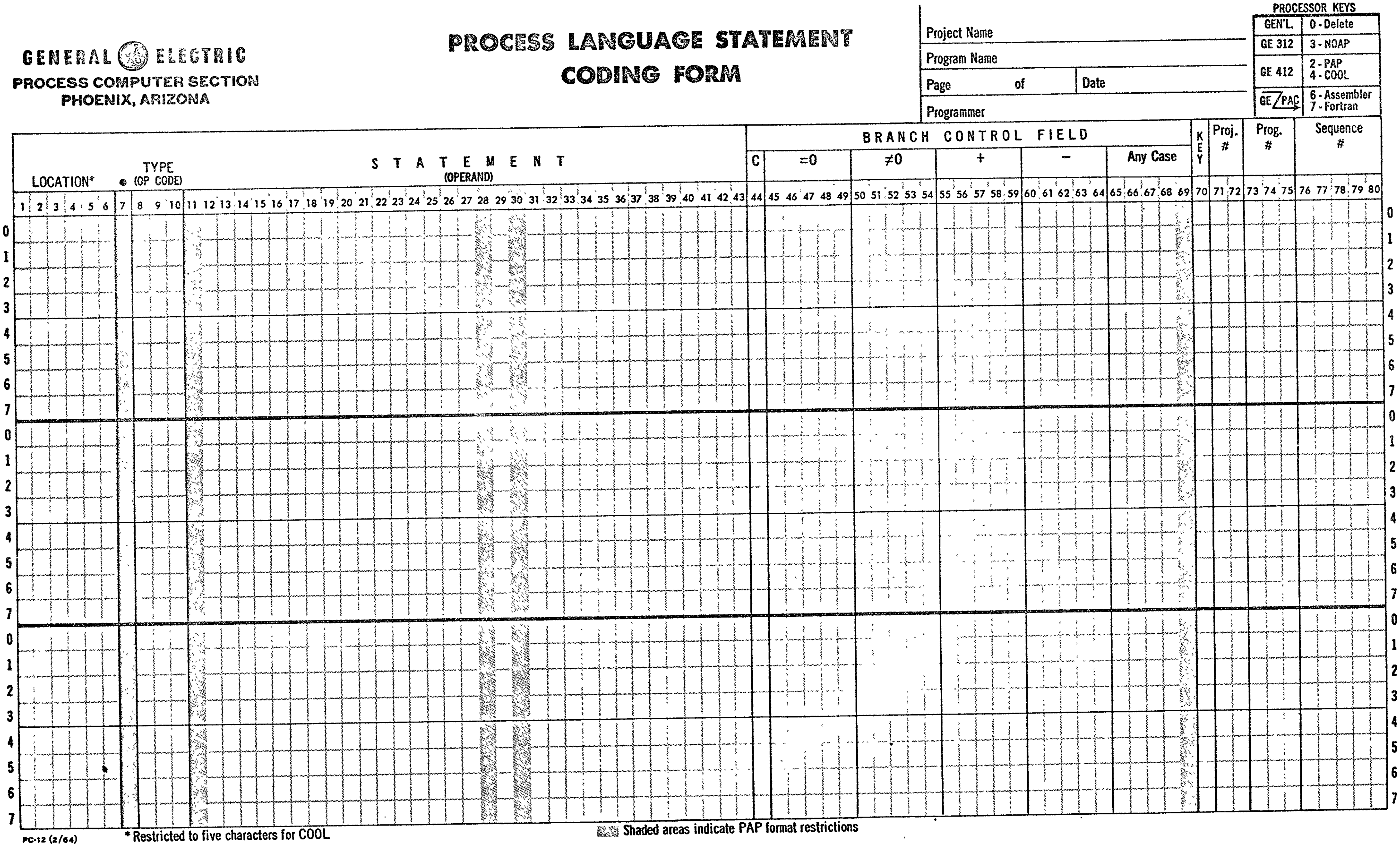
Symbolic - A name or label composed of six or fewer alphanumeric characters; the first of which must be alphabetic. Any such symbolic parameter corresponds to a name written in the Location Field of some instruction.

Decimal - A decimal integer value.

Octal - An octal integer value, preceded by a slash (/).

Relative[1] - An asterisk (*) which acquires the value associated with the memory location of its own instruction.

[1] -Refers to Present Location.



GENERAL ELECTRIC
PROCESS COMPUTER SECTION
PHOENIX, ARIZONA

PROCESS LANGUAGE STATEMENT CODING FORM

Project Name

Program Name

Page of Date

Programmer

PROCESSOR KEYS

GEN'L.	0 - Delete
GE 312	3 - NOAP
GE 412	2 - PAP 4 - COOL
GE/PAC	6 - Assembler 7 - Fortran

	LOCATION*	●	TYPE (OP CODE)	STATEMENT (OPERAND)	BRANCH CONTROL FIELD						KEY	Proj. #	Prog. #	Sequence #	
					C	=0	≠0	+	−	Any Case					
	1 2 3 4 5 6	7	8 9 10	11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43	44	45 46 47 48 49	50 51 52 53 54	55 56 57 58 59	60 61 62 63 64	65 66 67 68 69	70	71 72	73 74 75	76 77 78 79 80	
0															0
1															1
2															2
3															3
4															4
5															5
6															6
7															7
0															0
1															1
2															2
3															3
4															4
5															5
6															6
7															7
0															0
1															1
2															2
3															3
4															4
5															5
6															6
7															7

PC-12 (2/64)

* Restricted to five characters for COOL

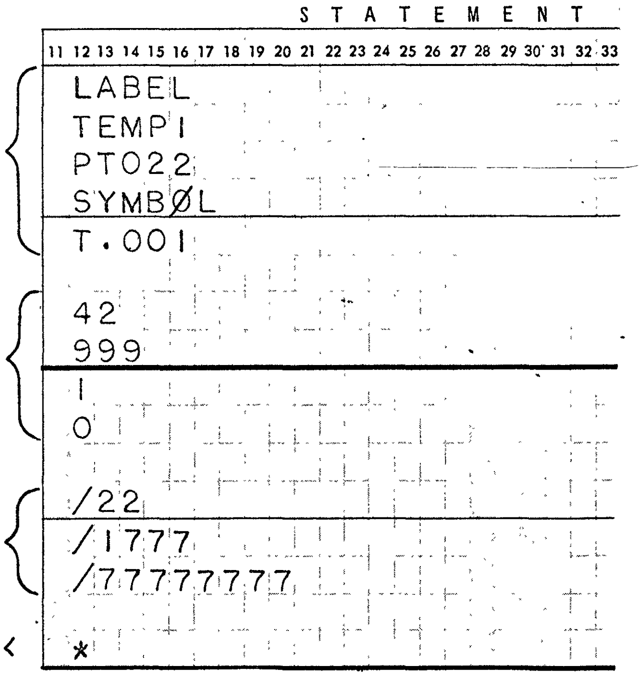
▨ Shaded areas indicate PAP format restrictions

FIGURE 1

A single operand value may be composed of one or a combination of the four parameter types listed on page one. These parameters are combined by using the following operators:

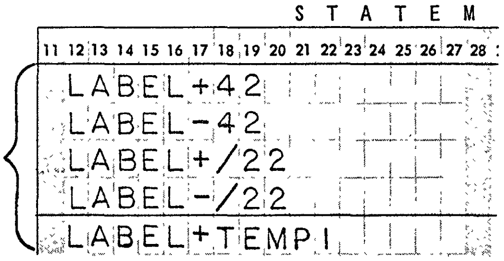
+ add
- subtract
* multiply
/ divide

Examples of the four parameter types follow:



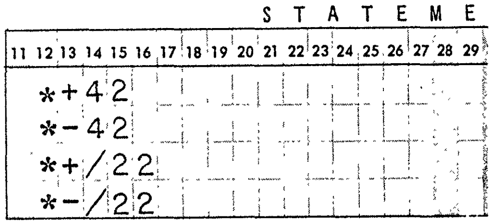
	STATEMENT 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
SYMBOLIC	LABEL
	TEMP1
	PT022
	SYMBØL
	T.001
DECIMAL	42
	999
	1
	0
OCTAL	/22
	/1777
	/77777777
RELATIVE	*

Examples of parameter combinations which form a single operand are illustrated below:



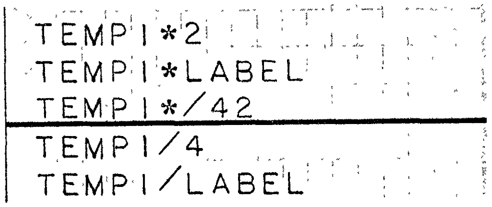
	STATEM 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Single operand as a combination of a symbolic and a decimal or octal.	LABEL+42
	LABEL-42
	LABEL+/22
	LABEL-/22
	LABEL+TEMP1

Combinations with *


S T A T E M E
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

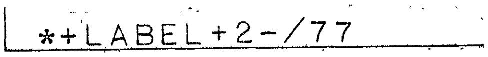
```
*+42
*-42
*+/22
*-/22
```

Multiplication & division



```
TEMP1*2
TEMP1*LABEL
TEMP1*/42
TEMP1/4
TEMP1/LABEL
```

All four types combined



```
*+LABEL+2-/77
```

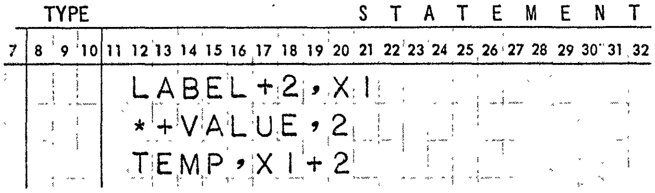
The parameters in the preceding examples are combined strictly from left to right. The meaning of asterisk or slash depends upon its relationship to the remaining parameters.

For example, asterisk represents relative addressing if it is the first character of a parameter. In all other cases, it is a multiplication sign. The slash indicates an octal parameter when it is the first character of a parameter; otherwise it is the division sign.

Most assembly language statements in any program involve only simple operand combinations. Rarely will the multiplication or division capabilities be utilized. It is important to understand that the combinations of parameters in the statement field, simple or complex, involve the values of the symbols; not the contents of the locations referenced.

The rules for combining parameters to form a single operand value are presented on page three. However, many computer instructions require more than one operand value. Most memory addressing instructions, for example, may have two operands; one to specify the memory address and a second to specify an index modification word. Multiple operands are desirable in many other occasions. When several operands are required, the comma (,) is used to separate operand values.

Examples of two-operand instructions are presented in the following:



	TYPE	STATEMENT
7	8 9 10	11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
		LABEL+2,X1
		*+VALUE,2
		TEMP,X1+2

The first blank character in the Statement Field terminates the construction of the operand. Characters appearing after the first blank column are treated as comments.

1.4 IDENTIFICATION FIELD - Columns 70 thru 80

The last eleven columns are reserved for the complete and unique identification of each line of coding. Column seventy is used to indicate the language used. The remainder of this field is allocated for identification of the project and program number, and the sequence within each program.

This field is essential for proper maintenance of the symbolic information. It is used by other programs which process symbolic records to produce additions, deletions, and corrections.

2. *PSEUDO OPERATIONS*

The pseudo operations are written on the coding form in the Op Code Field. They direct the assembler in storage assignments, symbol definitions, and generation of constants. All labels appearing in the Operand Field of pseudo operations must be predefined (i.e., they must be common symbols or they must have appeared in the Location Field of a statement preceding this statement). A summary follows:

	7	8 9 10	11 12 13	
		ØRG		Core Starting Address
		BSS		Block Storage Reservation
		DCW		Drum Loader Control Word
*		CØN	D,	Single Word Fixed Decimal Constant
		CØN	F,	Single Word Floating Decimal Constant
		CØN	A,	Variable Length Alpha-Numeric Constant
		CØN	Ø,	Single Word Octal Constant
		CØN	G,	Single Word General Constant
		DCN	D,	Double Word Decimal Constant
		DCN	F,	Double Word Floating Constant
		DCN	Ø,	Double Word Octal Constant
		GEN		Generate Duplicates
		EQL		Symbolic Equivalence
		DEF		Define A New Operation
		SLW		Slew Printer Page
		END		End Of Program

In addition, the following conventions and special pseudo operations are available:

a. An asterisk in column one identifies the entire line as a comment which will appear on the output assembly listing.

b. An asterisk in column seven of a statement indentifies the Location Field name as a common, absolute symbol. (refer to page 23)

c. A dash in column seven of the statement identifies the Location Field name as absolute (but not Common). (refer to page24)

d. Special pseudo operations for use with MONITOR (refer to Monitor manual).

e. Special pseudo operations for use with FORTRAN (refer to para. 2.12).

* - Revised 11/65

2.1 ORG - ORIGIN/CORE STARTING ADDRESS

ORG specifies the core starting address of the program. The address must be written in the Operand Field with the following restriction. Any symbolic that is used must be defined previously from a common symbol tape or by appearing in the Location Field of an earlier instruction. Note the following:

1. The ORG command produces control information for the loader.

2. The origin value initializes the location counter that appears on the output listing.

3. An asterisk in column seven informs the loader not to relocate instructions following ORG, even when instructed to relocate at load time. This inhibition is maintained until the next ORG or DCW statement is attained.

The Location Field is not used with the ORG pseudo-op.

Because of the relative addressing characteristics of the GE/PAC computer, instructions are assembled in the relative rather than the absolute form. They may be relocated at will during loading. Therefore, the ORG command is often unnecessary and is infrequently used.

2.2 BSS - BLOCK STORAGE RESERVATION

BSS is used to reserve or skip a block of memory. The size of the block is designated in the Operand Field. Any symbolic that is used must be defined previously from a common symbol type or by appearing in the Location Field of an earlier instruction.

A symbol written in the Location Field is entered into the assembler's table of symbolic equivalences. It is entered with the location value corresponding to the first location of the reserved block. This pseudo operation also generates control information for use by the loader.

2.3 DCW - DRUM CONTROL WORD

DCW is primarily used to specify the starting drum location for direct loading onto drum. Any symbolic that is used must be defined previously from a common symbol tape or by appearing in the Location Field of an earlier instruction. Two operand values may be used; the first designates the drum starting address. The second specifies a core starting address, equivalent to that required in ORG. The Location Field is not used with the DCW pseudo-op. Note the following:

1. DCW produces control information to be used by the loader. This information denotes a drum load.

2. The optional second operand is used to initialize the assembler location counter as with ORG. If a drum load is required, a separate ORG cannot be used because it generates core load control information.

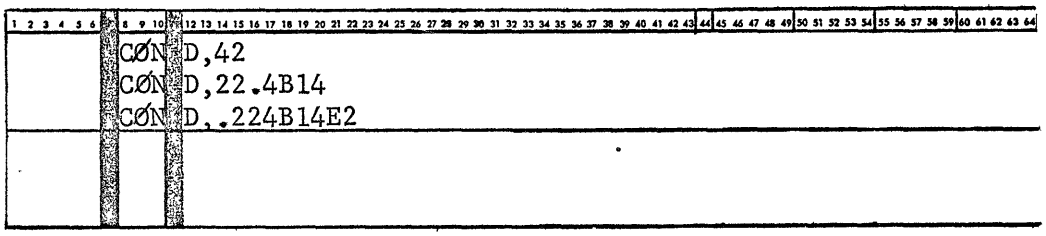
3. An asterisk in column seven informs the loader not to relocate instructions following DCW, even when instructed to relocate at load time. This inhibition is maintained until the next ORG or DCW is attained.

2.4 CON - CONSTANT

CON generates program constants. The five types are specified by an alphabetic character in column 12 as illustrated on page seven.

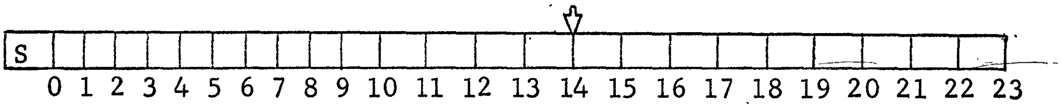
2.4.1 Single Word Fixed Decimal Constant

The first operand, D, identifies the constant as a fixed point decimal. The second operand presents the value of the constant. A binary scale factor and a power of ten exponent may be expressed in the constant. The binary scale factor indicates the bit position of the binary point relative to the sign bit in the word.



```
       CØN D,42
       CØN D,22.4B14
       CØN D,.224B14E2
```

In the preceding examples two and three, the binary point is assumed as indicated by the arrow:



Two bit-numbering methods are presented on the GE/PAC Console. Bits are numbered from left to right for scaling purposes. The binary point, in fixed point arithmetic, is specified relative to the sign (left most) bit.
Bits are numbered from right to left for instruction purposes. There is a class of instructions which permit bit manipulation

in the GE/PAC Computer. For this class of instructions, the bits are numbered from right to left i.e., bit zero is the right-most bit of the A Register and bit twenty-three is the sign bit.

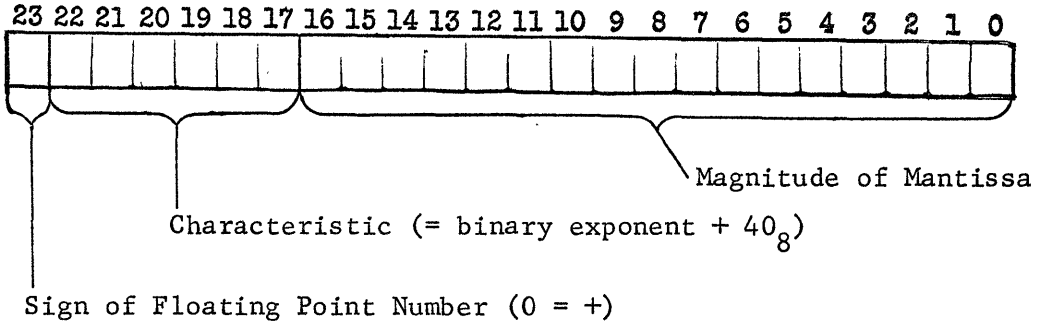
The assembler will generate the binary equivalent for a CON constant. A symbol appearing in the Location Field of any constant will be entered in the assembler table of symbolic equivalences. All references to the constant may be made by that name.

2.4.2 Single Word Floating Point Decimal Constant

The first operand, F, in the following example, identifies the constant as a floating decimal. The second operand presents the value of the constant. The number following the E indicates the power of ten exponent.

```
       CØN F,22.5
       CØN F,225E-1
       CØN F,.225E2
```

A single word floating point constant is formed with the characteristic in the six bits following the sign bit and the normalized mantissa in the rightmost seventeen bits of a word.



2.4.3 Variable Length Alphanumeric Constant

The first operand, A, identifies the constant as an alphanumeric. The second operand presents the number of characters of the constant (between 1 and 52). The third presents the characters of the constant. The constants are translated to the GE/PAC 6 bit code and packed four per word from left to right.
* The last word is completed with blanks if there are not four characters to fill it.

EXAMPLES:

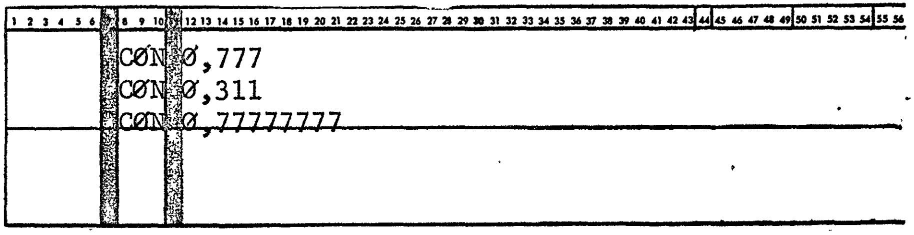
```
       CØN A,3,ABC            GENERATE ØNE WØRD--4TH CHAR. BLANK
       CØN A,4,ABC                     (SAME AS ABØVE)
       CØN A,10,ABCDEFGHIJ    GENERATES THREE WORDS
```

* - Revised 1/65

2.4.4 Single Word Octal Constant

The first operand, Ø, identifies the constant as an octal constant. The second operand presents the value of the constant.

Examples:



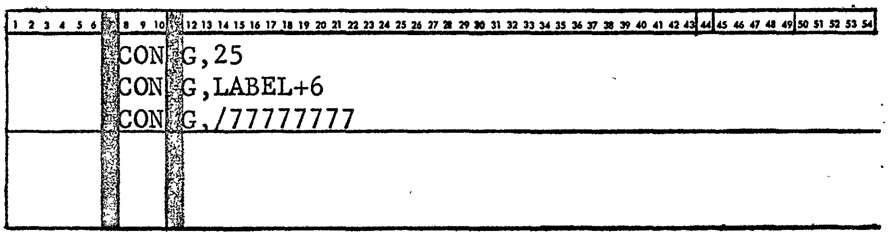
```
CØN Ø,777
CØN Ø,311
CØN Ø,77777777
```

The value of the second operand is converted to binary, right justified, and entered in the program as a single word constant.

2.4.5 Single Word General Constant

The first operand, G, identifies it as a general constant. The second operand specifies the value of the constant. The second operand may be any combination of parameters defined as legal for the Operand Field. A decimal, a symbolic, and an octal constant are illustrated here:

Examples:



```
CON G,25
CON G,LABEL+6
CON G,/77777777
```

Decimal constants of this type must be integers. B and E as used in the CØN,D pseudo op are not allowed.

2.5 DCN - DOUBLE WORD CONSTANT

DCN is used to generate double word program constants. The type of constant and its value is specified in the Operand Field.

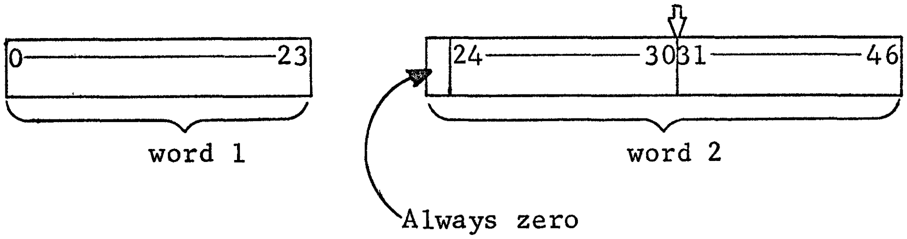
2.5.1 Double Word Fixed Decimal

The first operand, D, identifies the constant as a fixed point decimal. * Bit zero of the Q-Register does not enter into the calculation of the double register. The second operand presents the value of the constant. A binary scale factor and a power of ten exponent may be expressed in the constant. The binary point is relative to the sign bit in the first word.

Examples:

```
       DCN D,42
       DCN D,22.4B30
       DCN D,.224B30E2
```

In the preceding examples two and three, the binary point is assumed as indicated by the arrow:



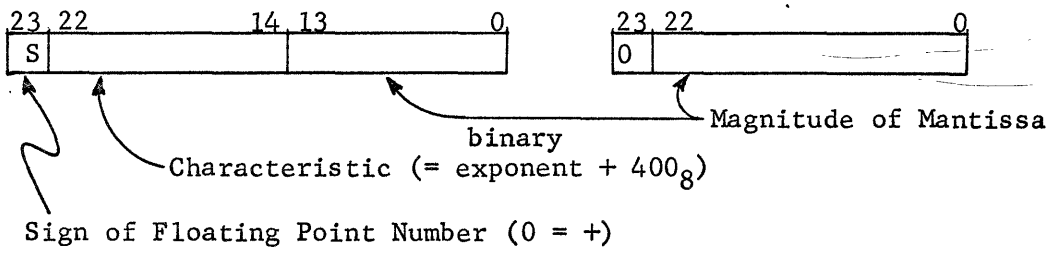
2.5.2 Double Word Floating Point Decimal Constant

The first operand, F, identifies the constant as a floating decimal. The second operand presents the value of the constant.

Examples:

```
       DCN F,232.5
       DCN F,2.325E2
       DCN F,2325.E-1
```

The format of the double word floating constant is illustrated below:


23
22
14
13
0
S
23
22
0
0
Magnitude of Mantissa
binary
Characteristic (= exponent + 400₈)
Sign of Floating Point Number (0 = +)

2.5.3 Double Word Octal Constant

The first operand, Ø, identifies the constant as an octal constant. The value of the constant is presented in the second operand. It is converted to binary, right justified, and
* entered as a double word logical constant (48 bits). Bit zero of the Q-Register is considered as part of the construction of the constant.

Examples:

```
       DCN Ø,77
       DCN Ø,311
       DCN Ø,7777777777777777
```

2.6 GEN - GENERATE DUPLICATES

A GEN instruction operand specifies the total number of times the following instruction will appear. The operand may be a combination of parameters defined as legal for the Operand Field. Symbols used must be predefined.

```
       GEN 3
       CØN Ø,4
N      EQL 2
       GEN N
       CØN Ø,7
*
*                      IS EQUIVALENT TO
*
       CØN Ø,4
       CØN Ø,4
       CØN Ø,4
       CØN Ø,7
       CØN Ø,7
```

* The GEN instruction cannot be used to duplicate the following pseudo-ops:

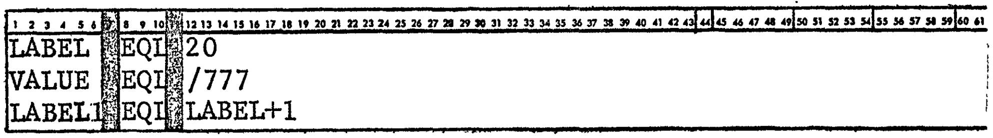
ØRG	EQL
BSS	DEF
DCW	SLW
CØN,A	END
GEN	LIB
	IDN

* - Revised 11/65

2.7 EQL - ASSIGN A SYMBOLIC EQUIVALENCE

EQL is used to assign a value to a symbol during assembly. This function is useful in sharing storage areas, establishing communication between programs, and assigning parameter values in a form adaptable for changes.

Enter the symbol into the Location Field and its equivalent value into the Operand Field. The operand value may be any combination of parameters described in Section One. There is one restriction; symbols written in the Operand Field must be previously defined.



```
LABEL   EQL 20
VALUE   EQL /777
LABEL1  EQL LABEL+1
```

2.8 DEF - DEFINE A NEW OPERATION

DEF permits the programmer to define new operation codes (i.e., codes that are not defined in the PAL language).

DEF is used for the following reasons:

1. An op-code may not exist for all GE/PAC Computers; therefore it is not included in the basic PAL language. The programmer may define the op-code when frequently used in his system[2].

2. Tables of parameters may consist of more than one value packed within a word. In this case, it is more convenient to write the values separtately and allow the assembler to combine them into single words. Otherwise, the values would have to be hand-assembled to one-word constants and entered by using the CON pseudo-op. DEF permits the programmer to define new operation codes that will accept multiple values as separate operands.

A detailed procedure for using the DEF pseudo-op follows.

2.8.1 Op-Code Definition

Machine operations (Op-Codes) are written with optional operands separated by commas. The assembled instruction results from combining the values of the operands with a "base octal" associated with the op-code. The base octal is twenty-four bits long.

[2]-For infrequently used operations, it may be easier to insert the undefined instruction as an octal constant.

Each operand has two format characteristics:

- Operand width (specified as the largest octal value that may be generated)
- Position (Where the value is placed in the finished word)

Example: The machine op-code <u>ADD</u> is comprised of the following:

- Base octal of 11000000
- Two operands

First operand value is placed into bits 14-0 (width 77777_8, position 0)
Second operand value (index register) is placed into bits 17-15 (width 7, position 15)

When defining a new op-code, specify the base octal, widths, and operand positions. Up to four operands are normally allowed[3] (coded in columns 12-68). The assembler then follows the instructions each time the new op-code is encountered. The assembler itself uses this technique to process op-codes which are permanently defined.

Defining a new op requires defining "audit codes." An audit code specifies an operand width and position to the assembler. Each audit code has a number associated with it. Sixty-four audit codes are available for the assembler. Audit code numbers 0-20 are pre-defined (Refer to APPENDIX C). Codes 21-40 should be avoided by the programmer; they are reserved for PAL and Monitor for future op-code additions. Audit code 63 has special use (Refer to para. 2.8.2). Therefore, only audit code numbers 41-62 may be assigned freely by the programmer. It is recommended that the programmer start with 62, numbering in reverse.

Three steps in defining a new operation follow:

1. Determine width and position of all operands
2. Define audit codes for those widths and positions
3. Define the op-code by assigning to it the audit codes from step 2 above.

The steps in detail:

1. For each operand of a new op-code, determine its width and the position of its right-most bit (anchor bit).

[3]-Refer to para. 2.8.2 when more than four operands are required.

Example: Assume a new op-code, called AAA, were needed with the following characteristics:

a. A base octal of 12034560

b. Two operands, whose values are placed in bits 17-15 and bits 3-0. These operands are described as follows: Operand one has a width of three bits and an anchor bit of 15. This width is expressed as the largest number it can contain in octal (in this case, 7 is the largest number 3 bits can contain). The second operand has a width of four bits (/17) and an anchor bit of zero.

2. Define audit codes for each operand as follows:

   a. Write the audit code in columns one and two of the Location Field
   b. Write DEF in the Op Code Field
   c. In the Operand Field, write two operands separated by a comma. The first is the width, expressed in octal; the second is the anchor bit position.

Example:

The coding for AAA above follows:

Location	Op-Code	Operand
62	DEF	/7,15
61	DEF	/17,0

Audit Code Width Anchor Bit

3. When all necessary audit codes are defined, the op-code itself may be defined.

   -Write the new operation mnemonic in columns one
* through three of the Location Field.
   -Write DEF in the Op Code Field.
   -Write the base octal and audit codes, separated by commas, in the Statement Field.

Example:

Location	Op-Code	Operand
AAA	DEF	/12034560,62,61

*

Mnemonic

Base Octal

First - Second operand audit codes, in the order the operands will appear.

The new op-code may now be used (note that a symbolic operand may be used):

1 2 3 4 5 6	8 9 10	12 13 14 15
A	EQL	3
	AAA	4 2
	AAA	A, 0

$$\text{Result}\quad \begin{matrix} 12 & \left[\begin{matrix}4\\3\end{matrix}\right] & 3 & 4 & 5 & 6 & \left[\begin{matrix}2\\0\end{matrix}\right] \\ 12 & & 3 & 4 & 5 & 6 & \end{matrix}$$

1st Operand   2nd Operand

Different op-codes may share audit codes if they have operands which coincide exactly in width and position (i.e., the programmer need define only one audit ~~code for a particular~~ operand width and position, then that audit code may be used freely for as many op-codes as desired).

2.8.2 Extra Operands Option

When generating special tables of data, four operands may be inadequate.

Example: A table of constants is desired where each word is divided as follows; each section is able to take on an independent value.

23	22 21 20 19	18 17 16 15 14 13	12	11 10 9 8	7 6	5 4 3 2 1 0

The normal technique for op-codes permits only four operands, which is insufficient (the example requires 7 operands). The use of DEF in the Extra Operands Option permits up to 12 operands. The steps for defining an op-code when using the Extra Operands Option are nearly identical to those for a normal op-code.

1. For each operand, determine its width and the position of its right most bit.

Example:

width (no. of bits)	largest number it can contain	anchor bit
1	1	23
4	17_8	19
6	77_8	13
1	1	12
4	17_8	8.
2	3	6
6	77_8	0

2. Define audit codes for those positions and bit lengths

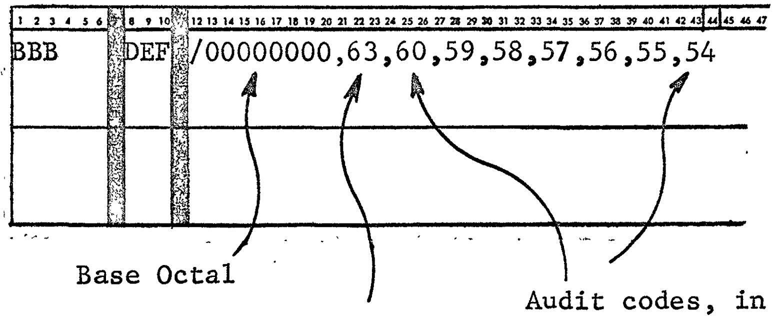
Example:

```
60      DEF  /1,23
59      DEF  /17,19
58      DEF  /77,13
57      DEF  /1,12
56      DEF  /17,8
55      DEF  /3,6
54      DEF  /77,0
```

Note: If an audit code, with identical width and anchor bit for an earlier op-code is defined, it need not be defined again, but may be used for this op-code.

3. When all necessary audit codes are defined, the op-code itself is defined. To signal the Extra Operands Option, place audit code 63 as the first audit code.

Example: (BBB is used as the mnemonic for this op-code).



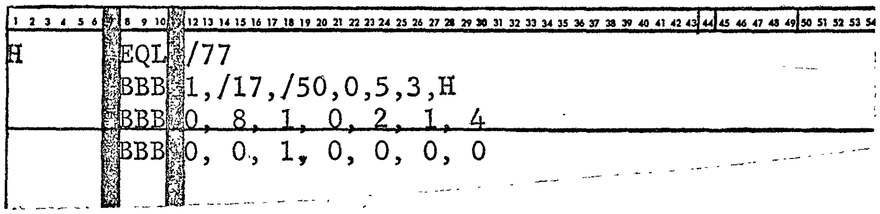
```
BBB     DEF  /00000000,63,60,59,58,57,56,55,54
```

Base Octal

Extra Operands Option

Audit codes, in the same order the operands will appear

4. Using the new mnemonic:



```
H       EQL /77
        BBB 1,/17,/50,0,5,3,H
        BBB 0, 8, 1, 0, 2, 1, 4
        BBB 0, 0, 1, 0, 0, 0, 0
```

are assembled as:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	77202777
0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	20021104
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	00020000

2.8.3 General Rules

1. DEF is available only for additions to the language, not for substituting new definitions for permanently defined PAL op-codes.
2. Operands of the DEF instruction itself may be written in decimal, octal, or symbolic. However, all symbols must be previously defined. The usual rules for determining if operands are relative or absolute do not apply. All operands are considered absolute.
3. Operands for newly defined op-codes follow the usual rules for translation as absolute or relative symbolics (Refer to page 23)
4. Audit codes must be numeric.
5. Audit codes 41-62 only may be used. (it is recommended that the programmer start with 62 and number in reverse).
6. New op-codes and their audit codes and their audit code definitions are a part of the "Common Symbol Tape," which may be preserved for subsequent assemblies.
7. The Extra Operands Option may be used for defining all op-codes. However, normal use of DEF for op-codes re-requiring four or fewer operands will save space that the assembler can use to store more symbols.

2.8.4 Error Conditions

1. When a mnemonic is used to define an operation that has been permanently defined, the first definition applies and the new definition will be flagged as a location error (L). Permanently defined op-codes are those that make up the PAL language codes.
2. When defining an op-code, if audit code 63 (Extra Operands Option) is written anywhere except directly after the base octal, an illegal operand error (I) will occur. Definition of the op-code is terminated at this point.
3. When defining an audit code, the numbers 0 through 20 will be flagged as a location error (L). They are reserved for the assembler.

2.9 SLW - SLEW PRINTER PAGE

Upon encountering the SLW pseudo-op, the assembler positions the paper to the top of the next page. Assemblers operating on computers without output page control ignore the SLW instruction.

2.10 END - END OF PROGRAM

END is used to terminate the assembly and must be the last statement in a program. Upon encountering END, the assembler generates control information for the loader.

An operand value may be specified to indicate the starting point of the program. When this is done, the starting address is also communicated to the loader.

2.11 MONITOR Pseudo-Ops

Pseudo-Ops exist to generate constants for MONITOR. They are used to generate parameters for calling sequences and format words for input/output. These Pseudo-Ops are described in the GE/PAC MONITOR Manual and are available only through the MONITOR Common Symbol Tape. Refer to paragraph 2.8.3, item #6, of page 19.

2.12 LOADER Pseudo-ops

Two pseudo-ops are required for the GE/PAC Loader to call pre-assembled library routines and functions as continuous parts of a program. One identifies the pre-assembled program, routine, or function; another requests that such programs be read from a library tape.

These pseudo-ops are used by FORTRAN. They may also be used in any PAL language program to call standard pre-assembled subroutines without including the subroutine as a symbolic part of the program.

2.12.1 IDENTIFY - IDN

IDN when used must be the first instruction of a program. It identifies the program as a library subroutine which may
* be called by some other program. No ORG or DCW may appear in a library subroutine. The identification is limited to six alphanumerics, the first of which must be alphabetic.

Example: IDN SQRTF

The example states that the following program is SQRTF and may be called by another program.
The assembler generates two lines on the listing. The octals are preceded by an "8" control frame and contain the six-bit BCD representation of the identification requested. No memory is reserved by the assembler.

* Revised 11/65

2.12.2 LIBRARY - LIB

LIB pseudo-ops are interpreted by the loader as a command to call preassembled functions (identified by IDN) from a library tape of standard pre-assembled routines. Any LIB pseudo-op must immediately precede the END card for the program.

Example: SQRTF LIB

This example states that the programmer requests the SQRTF function to be included as an adjoining part of his program. The assembler generates two lines on the listing. The octals are preceded by a "9" control frame and contain the six-bit BCD representation of the program identification requested. * The loader, after encountering an LIB call, searches for the specified library subroutine (SQRTF in above example). The routine will have been identified with the IDN pseudo-op. One memory location is reserved by the assembler for each LIB pseudo-op. The loader fills in the relative branch to the actual locations of the requested subroutines when they are loaded into drum or core.

* - Revised 11/65

3. PROCESSING & OUTPUT

GE/PAC uses a two-pass assembler. During the first pass, the assembler examines the storage allocation and forms a table of symbolic equivalences. Each symbol appearing in the Location Field is entered into the symbol table. During the second pass, the assembler completely forms each instruction code. For each symbolic operation, the assembler searches a table of permanent operation codes to acquire its corresponding base octal and audit codes for operand treatment. The Operand Field is scanned, and equivalent values are computed using the symbol table formed during the first pass. The base octal and the operands are then combined using the audit codes. The final step in the assembly of each statement is the preparation of a line of output for listing and direct loading.

3.1 ABSOLUTE AND RELATIVE VALUES

An operand address of an instruction which references memory may be specified to be assembled absolute or relative. An absolute address is the true location of the referenced memory cell. A relative address is the difference between the address of the referenced memory cell and the address of the instruction. The result is negative when the referenced address issmaller than the instruction's address.

An operand as written by the programmer can consist of one or a combination of the following terms:

An absolute label
An integer } Absolute terms

A relative label
The * (present location) } Relative terms

Combinations are formed using add (+), subtract (-), multiply (*), and divide (/) operators (Refer to para. 1.3).

An operand is assembled relative if any relative term (a relative label or the asterisk) appears in the combination of terms. An operand is assembled absolute when no relative term appears.

A label usually represents an address in memory. To reference an address absolute, the label associated with that address must be specified as absolute. Otherwise, it is assumed to be relative by the assembler. A label specified as Common to more than one program (Refer to para. 3.2) is assembled as an absolute label in all programs in which it is referenced. Two additional methods to specify a label absolute are as follows:

1. Place a dash in column seven of the line of coding where the label appears in the Location Field.

   Example: ALPHA -LDA BETA ALPHA will be referenced absolute

2. Use the EQL pseudo-op to equate the label to an absolute operand.

   Example: GAMMA EQL ALPHA+2 Both ALPHA and 2 are absolute. Therefore, the operand is absolute and GAMMA will be considered absolute.

3.2 COMMON SYMBOLS

A label is specified to be Common by placing an asterisk (*) in column seven of the line of coding where the label appears in the Location Field. Each Common label and its value is available through a common symbol tape to all other programs in a system.

3.3 ASSEMBLER VARIATIONS

Variations in the capabilities of language translators originate from the different operating environment of each. For example, the memory available affects the size of the symbol table. It is important that the characteristics of each translator be examined prior to its use.

3.4 OUTPUT

Each translator of the assembly language possesses its own output characteristics. However, all translators can produce the two basic output requirements:

- an assembly listing, which includes the original symbolic information plus the assembled instructions in the relative and absolute forms. The output listing will also contain error flags. They are defined in Appendix B.

- a punched paper tape for direct loading in the GE/PAC system.

APPENDIX A

CHARACTER CODE TRANSLATION

The list of characters in Table I comprises the total character set associated with the assembler language. The six bit octal associated with each character represents the equivalent code produced by the paper tape preparation device for input. The six bit octal also represents the code produced when using the alphanumeric constant.

The additional characters listed in Table II are legal in the Comment Field. Table III presents functions that have meaning in paper tape versions of the assembler, but are not part of the Assembler Language.

TABLE II

CHARACTER		TYPER
(	=	35
)	=	55
=	=	75
"	=	76
$	=	53

TABLE III

CHARACTER		TYPER
carriage return	=	100
tab	=	140
punch on	=	162
punch off	=	164
print red	=	161
print black	=	160
delete	=	177
stop	=	170

TABLE I

CHARACTER		TYPER
A	=	21
B	=	22
C	=	23
D	=	24
E	=	25
F	=	26
G	=	27
H	=	30
I	=	31
J	=	41
K	=	42
L	=	43
M	=	44
N	=	45
Ø	=	46
P	=	47
Q	=	50
R	=	51
S	=	62
T	=	63
U	=	64
V	=	65
W	=	66
X	=	67
Y	=	70
Ƶ	=	71
0	=	00
1	=	01
2	=	02
3	=	03
4	=	04
5	=	05
6	=	06
7	=	07
8	=	10
9	=	11
+	=	60
-	=	52
/	=	61
*	=	54
.	=	33
,	=	73
space	=	20

APPENDIX B ERROR FLAGS

The assembler performs validity tests on each instruction.
When errors or suspected errors are detected, one of the following indicators will appear on the output listing.

FLAG	DEFINITION	CAUSE
L	Location Field Error	1. First character of the label is not alphabetic, (See Appendix A, Table I). 2. Using the DEF pseudo-op when: - the mnemonic assigned is a GE/PAC machine operation. - requesting "Extra Operands" definition when mnemonic has been previously defined as machine-typed operation. - there is an illegal audit code number. 3. Location Field is blank when a symbol is required. 4. Location Field contains a symbol when not allowed.
* Ø	Operation Field Error	1. The Op-code not part of the language or was not added to the table through the DEF pseudo-op. Thi: often occurs when definition was attempted but was illegal. Consequently, it was not added to the operation table. 2. This op-code cannot be GENerat
I *	Illegal Operand	1. Blank operand when an operand is required. 2. Operand not blank when it should have been. 3. One or more required operands missing. 4. Too many operands. 5. Operand value too large. 6. Negative operand value in an instruction that will not accept one.. 7. Illegal constant
X	Index Word Error	1. Index word 1 or 2 specified. 2. Required index missing. 3. Specified index word is greater than seven.
U	Undefined Symbol	Occurs only when a symbol appears in the Operand Field and: 1. It never appeared in the Location Field or on the Common Symbol Tape. 2. It appeared in the Location Field, but the symbol table was full at that time.
C	Illegal Character	A character, not associated with the assembler languag was found in one of the following fields: 1. Location 2. Op-Code 3. Operand (Refer to Appendix A, Table II)
M	Multiply-defined Symbol	1. Symbol in Location Field was flagged because: - it has appeared in the same field on a previous record - it appeared on the requested EQL tape with a value unequal to the one being assigned. - it was saved from a previous assembly with a value unequal to the one being assigned. 2. Any record which references a multiply-defined symbol in the Operand Field will also be flagged.
2	Second Pass definition of symbol different from First Pass	
R	Relative Operand Error	Operand value was relative and should be absolute.
F	Tables full	

APPENDIX C

AUDIT CODES

Each machine instruction may have operand values. The assembler has a list of operand-associated numbers called audit codes. These numbers uniquely identify individual operand requirements. Two basic characteristics of audit codes are: the operand width expressed as a mask and an anchor position.

A maximum of sixty-four audit codes are available. Audit codes 0 through 40 are reserved for the assembler and Monitor; 41 through 62 are available for programmer definition. The first twenty audit codes are defined and listed below.

AUDIT NUMBER	OPERAND WIDTH	ANCHOR POSITION	SPECIAL REMARKS	OPERAND ERROR-FLAG CONDITIONS					
				*	Zero	Not Blank	Blank	2	1
0	0	0	No operand accepted			I			
1	14	0	Full operand. May be absolute or relative.				I		
2	3	15	For instructions with optional Tag Field. Must be absolute if not blank	R	X			X	X
3	14	0	Full operand. Must be absolute.	R			I		
4	12	0	Must be absolute.	R			I		
5	24	0	Pass without audit (any value accepted).				I		
6	3	15	For instructions requiring a tag. Must be absolute.	R	I		I	X	X
7	5	0	K bits. Must be absolute.	R			I		
8	14	0	Value will be negated (TXH). Must be absolute.	R			I		
* 9	23	0	Operand may be absolute or relative (DEL)				I		
10	3	15	For STX instruction.	R	X		I	X	
11	14	0	Operand may be relative or absolute and may be negative (INX)				I		
12	5	15	Absolute only	R			I		
13	1	22	Absolute only	R			I		
14	1	21	Absolute only	R			I		
15	1	20	Absolute only	R			I		
16	1	23	Absolute only	R			I		
17	23	0	Absolute only	R			I		
18	3	15	For STR-type instruction	R			I		
19	12	0	Absolute only; may be negative	R			I		

* - Revised 3/66

APPENDIX D

MACHINE OPERATIONS

MNEMONIC	NAME
ABL	APPEND ITEM TO BEGINNING OF LIST
ABT	ABORT DEVICE D's OPERATION
ACT	ACTIVATE DEVICE D's INTERRUPT
ADD	ADD
ADO	ADD ONE TO BIT K
AEL	APPEND ITEM TO END OF LIST
AFA *	ADD FIELD TO A
AKA	ADD K TO A
ANA	AND TO A
BRU	BRANCH UNCONDITIONALLY
BTR	BRANCH IF TSTF RESET
BTS	BRANCH IF TSTF SET
CBK	CHANGE BIT K
CLO	COUNT LEAST SIGNIFICANT ONES
CLZ	COUNT LEAST SIGNIFICANT ZEROS
CMO	COUNT MOST SIGNIFICANT ONES
CMZ	COUNT MOST SIGNIFICANT ZEROS
CPL	COMPLEMENT A
DAD	DOUBLE ADD
DLA	(SHIFT) DOUBLE LEFT ARITHMETIC
DLD	DOUBLE LENGTH LOAD
DLL	(SHIFT) DOUBLE LEFT LOGICAL
DMT	DECREMENT MEMORY AND TEST
DRA	(SHIFT) DOUBLE RIGHT ARITHMETIC
DRC	(SHIFT) DOUBLE RIGHT CIRCULAR
DRL	(SHIFT) DOUBLE RIGHT LOGICAL
DST	DOUBLE LENGTH STORE
DSU	DOUBLE SUBTRACT
DVD	DIVIDE
ERA	EXCLUSIVE OR TO A
FAD	FLOATING ADD
FDV	FLOATING DIVIDE
FIX	FIX FLOATING NUMBER
FLO	FLOAT FIXED NUMBER
FMP	FLOATING MULTIPLY
FSU	FLOATING SUBTRACT
IAI	INHIBIT AUTOMATIC INTERRUPT
IBK	ISOLATE BIT K
IDL *	INPUT FROM DEVICE TO LIST
IN	INPUT FROM DEVICE D
INX	INCREMENT X
JND	JUMP IF NO DEMAND

MNEMONIC	NAME
JNE	JUMP IF DEVICE D NOT IN ERROR
JNO	JUMP IF NO OVERFLOW
JNP	JUMP IF NO PARITY ERROR
JNR	JUMP IF DEVICE D NOT READY
LBM	LOAD BIT MASK
LDA	LOAD THE A REGISTER
LDB *	LOAD HIGHSPEED I/O BUFFER
LDF *	LOAD FIELD
LDI	LOAD INDIRECT
LDK	LOAD A WITH K
LDO	LOAD ONE INTO
LDP	LOAD PLACE
LDQ	LOAD THE Q REGISTER
LDR *	LOAD REGISTERS
LDX	LOAD X WORD
LDZ	LOAD ZEROS INTO A
LMO	LOAD MINUS ONE
LPR	LOAD PLACE AND RESTORE
LXC	LOAD X WITH COUNT
LXK	LOAD X WITH K
MAQ	MOVE A TO Q
MPY	MULTIPLY
NEG	NEGATE
NOP	NO OPERATION
ODL *	OUTPUT FROM DEVICE TO LIST
OOM	OPERATE ON MEMORY
OPR	OPERATE DEVICE D
ORA	OR TO A
OUT	OUTPUT TO DEVICE D
PAI	PERMIT AUTOMATIC INTERRUPT
RBK	RESET BIT K
RBL	REMOVE BEGINNING ITEM FROM LIST
RCS	READ CONSOLE SWITCHES
REL	REMOVE ENDING ITEM FROM LIST
REV	RESET TSTF IF BIT K IS EVEN
RNZ	RESET TSTF IF A IS NONZERO
ROD	RESET TSTF IF BIT K IS ODD
RPT *	REPEAT INSTRUCTION IN LOCATION Y
RST	RESET TSTF
SBK	SET BIT K
SEL	SELECT DEVICE D
SET	SET TSTF
SEV	SET TSTF IF BIT K IS EVEN
SFA *	SUBTRACT FIELD FROM A
SKA	SUBTRACT K FROM A
SLA	SHIFT LEFT ARITHMETIC
SLL	SHIFT LEFT LOGICAL

MNEMONIC	NAME
SNZ	SET TSTF IF A IS NONZERO
SOD	SET TSTF IF BIT K IS ODD
SPB	SAVE PLACE AND BRANCH
SRA	SHIFT RIGHT ARITHMETIC
SRC	SHIFT RIGHT CIRCULAR
SRL	SHIFT RIGHT LOGICAL
SSA	SET STALL ALARM
STA	STORE CONTENTS OF A
STB *	STORE HIGHSPEED I/O BUFFER
STF *	STORE FIELD
STI	STORE INDIRECT
STQ	STORE CONTENTS OF Q
STR *	STORE REGISTERS
STX	STORE X
SUB	SUBTRACT
TER	TEST EVEN AND RESET BIT K
TES	TEST EVEN AND SET BIT K
TEV	TEST BIT K EVEN
TFE *	TEST FIELD EQUAL
TFL *	TEST FIELD LESS
TNM	TEST NOT MINUS ONE
TNZ	TEST A NONZERO
TOD	TEST BIT K ODD
TOR	TEST ODD AND RESET BIT K
TOS	TEST ODD AND SET BIT K
TSC	TEST AND SHIFT CIRCULAR
TXH	TEST X HIGH OR EQUAL
TZC	TEST ZERO AND COMPLEMENT
TZE	TEST A ZERO
XEC	EXECUTE

* - GE/PAC 4050 and 4060 only

APPENDIX E

BINARY CARD FORMAT

Data Card

Column 1, row 12	- 0 if Core, 1 if Drum or Disc
Column 1, row 11	- 0 if Absolute, 1 if Relocatable
Column 1, rows 0-5	- Number of words (n) $n \leq 32$
Column 1, rows 6-9	- Checksum, 4 most significant bits
Column 2	- Checksum, 12 least significant bits
Column 3, row 12	- 0
Column 3, rows 11-9	- Starting address, 11 most significant bits
Column 4	- Starting address, 12 least significant bits

Starting at column 5 are 2n columns containing n words of data, 12 bits in each column, the most significant half first.

Additional groups of starting addresses (same format as columns 1-4) and data may follow the first group if there is sufficient room before column 69.

The checksum is formed by separating each data word and the starting address into 2 parts, each 12 bits long. Each half if added to the 16 bit checksum. Overflow out of the 16th bit is ignored.

Transfer (END) Card

Column 1, rows 12-5	- 0
Column 1, rows 6-9	- Checksum, 4 most significant bits
Column 2	- Checksum, 12 most significant bits
Column 3	- Transfer address, first half
Column 4	- Transfer address, second half

A blank card is considered a TRA card, no transfer.

Identification Card (IDN)

Columns 1, 2	- 0
Column 3	- 4031_8 (reading top to bottom - I)
Column 4	- 2445_8 (reading top to bottom DN)
Columns 5, 6	- 24 bits representing the first IDN word
Columns 7, 8	- 24 bits representing the second IDN word

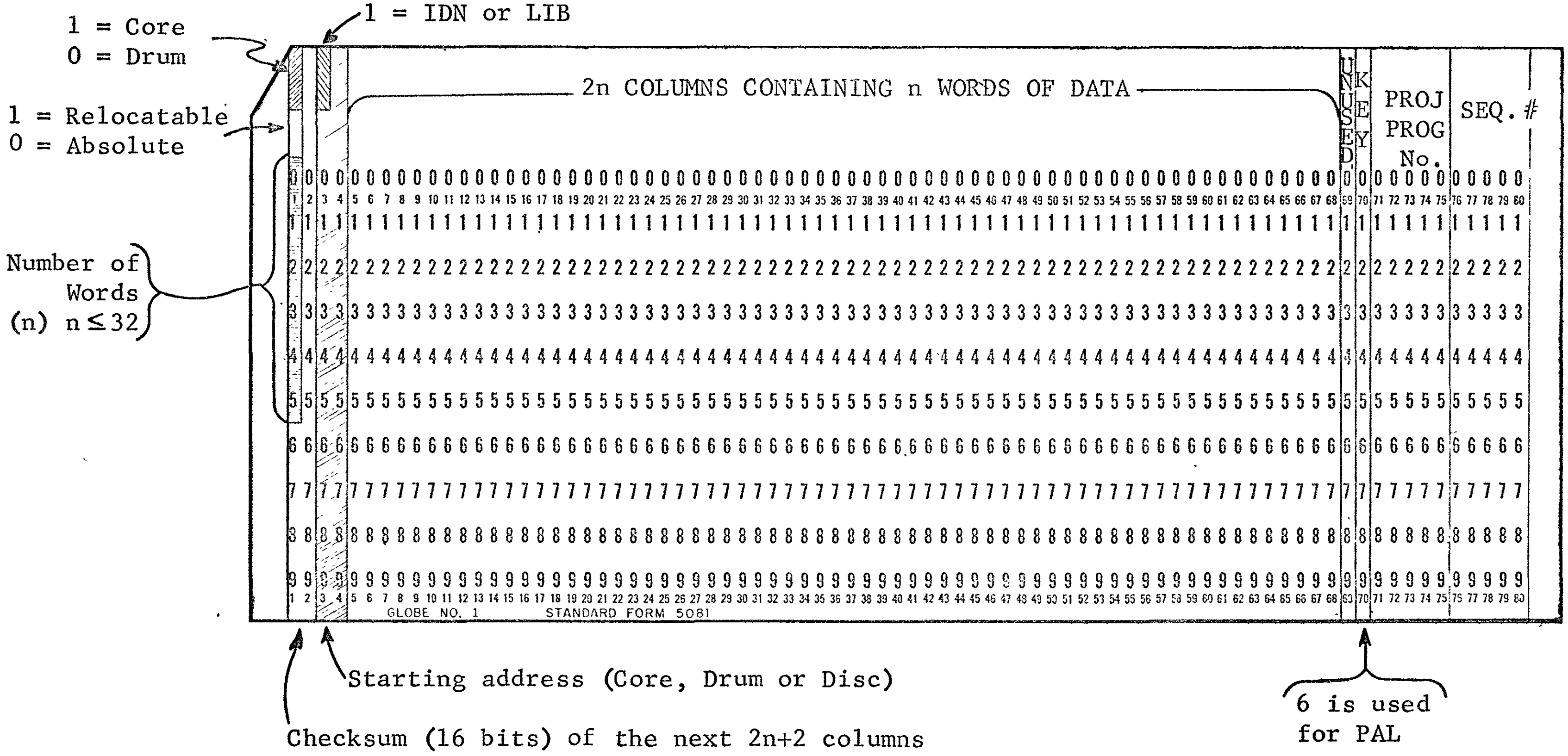
There is no checksum

LIB Card

Columns 1, 2	- 0
Column 3	- 4043_8(reading top to bottom -L)
Column 4	- 3122_8 (reading top to bottom IB)
Columns 5, 6	- 24 bits representing the first LIB word
Columns 7, 8	- 24 bits representing the second LIB word

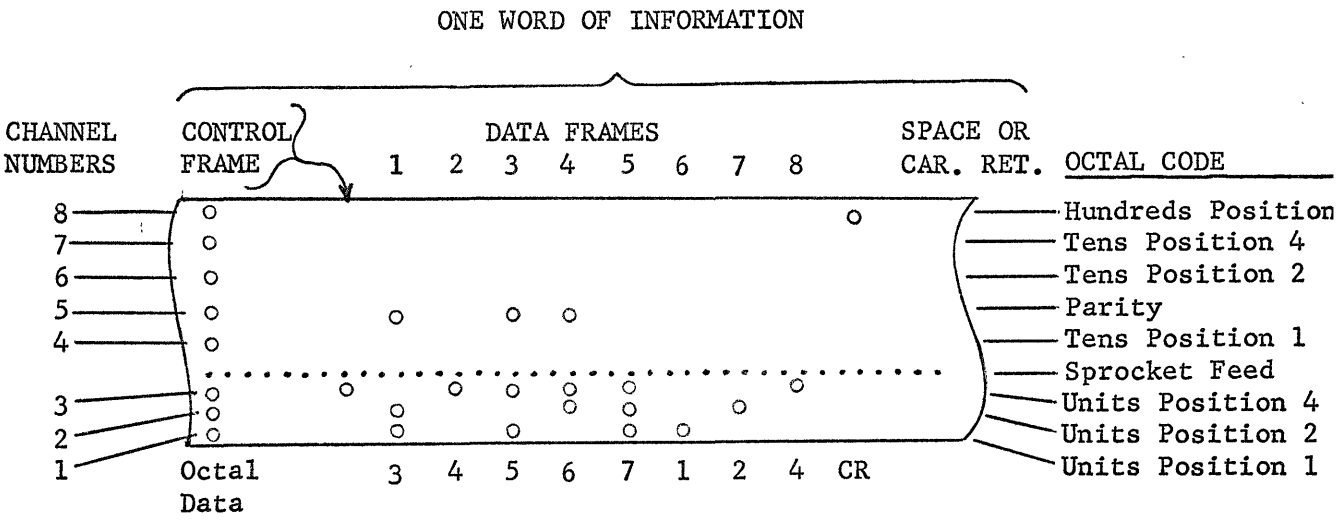
There is no checksum

APPENDIX E FIGURE I

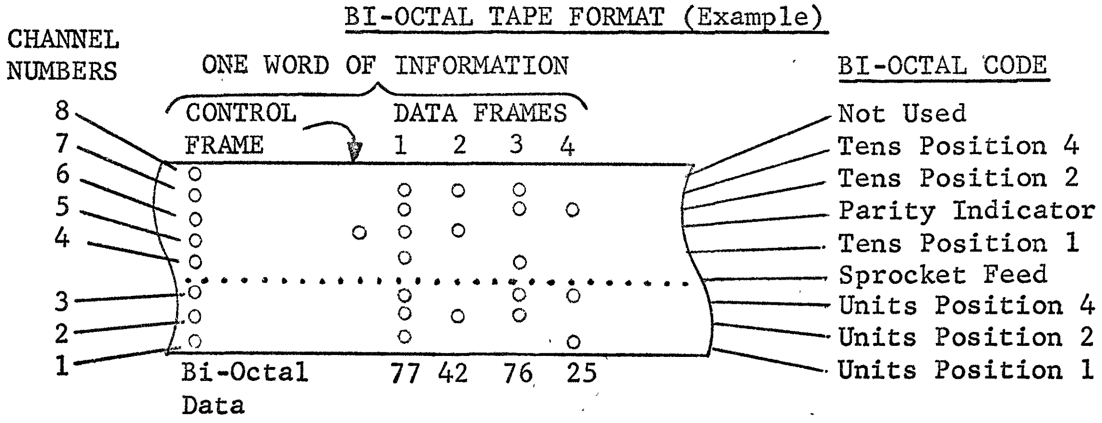


APPENDIX F

OCTAL LOAD TAPE FORMAT (Example)



BI-OCTAL TAPE FORMAT (Example)



Note: All characters have odd parity

INFORMATION TYPE	PAL PRINTOUT	BI-OCTAL CONTROL FRAME
Data	Blank	00
Starting Address (ORG)	1	01
Checksum		02
Skip (BSS)	3	03
IDN	8	10
LIB	9	11
END	*	12
Drum Starting Add. (DCW)	@	(See Note)

Note: All load tape information is in bi-octal format except a drum starting address. The drum starting address appears on the bi-octal tape in the same format as an octal tape: a $(14)_8$ Control Code is followed by eight octal frames, followed by a carriage return $(100)_8$.

GE/PAC 4000

not APPLICABLE to LUL-22

PAL45 - PAL-225 Assembler

Library Control No. YPF05XR2

February 1966

Proprietary Information of the
General Electric Company

In the development of the software described, the General Electric Company reserves the right to modify the design for reasons of improved performance and operational flexibility.

Programming Library
General Electric Company
Process Computer Section
P. O. Box 2918
Phoenix, Arizona - 85002

Library Control No. YPF05XR1

IDENTIFICATION

PAL45 - PAL-225 Assembler

PURPOSE

To provide complete PAL assembly capability on the GE-225.

OBJECTIVES

1. To assemble symbolic program(s) punched on cards in the PAL format.

2. To print an assembly listing.

3. To produce a binary card load deck in a format acceptable to:

   a. The 225 GE/PAC Simulator
   b. Card equipped GE/PAC computers
   c. The 225 Card to Paper Tape Translator for paper tape generation.

4. To save the common symbols for subsequent assemblies when selected as a console switch option.

SYSTEM COMMUNICATION

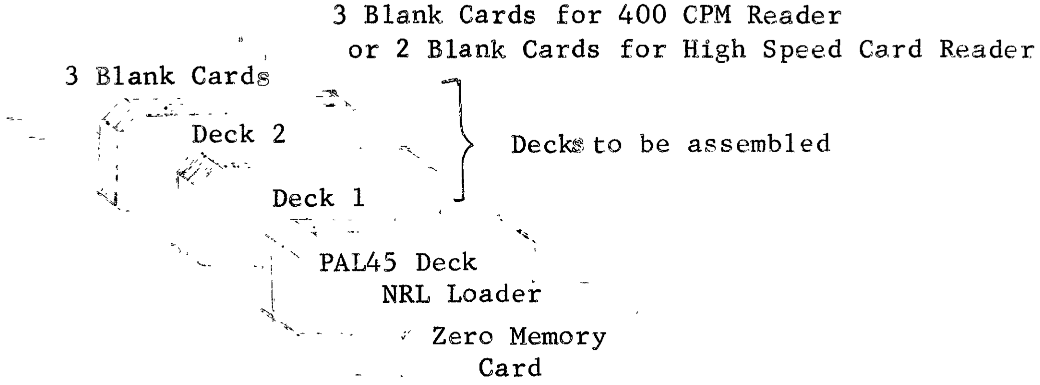
Console Operation - GE 225

1. Place a blank tape on Tape Unit 4.

   If a Common Symbol Tape is required, place it on Tape Unit 5. Place a blank tape on Tape Unit 6 to write a Symbol Tape.

2. Place the PAL45 binary card deck and the symbolic program(s) deck(s) to be assembled in the following order in the card reader.

   Program decks follow the PAL45 deck with the "END" card of one program followed by the first card of the next program. Place three (400 CPM Reader) or two (High Speed Card Reader) blank cards at the end of the last deck.



Console Operation - contd

3. Set the Card Punch ready.

4. Clear Console Switches.

5. Select options with the following switch settings:

Switch 17 - UP = Do not use Common Symbol Tape.
DOWN = Use Common Symbol Tape.

18 - UP = Clear Common Symbol Table between assemblies.
DOWN = Save Common Symbols between assemblies.

19 - UP = Do not write a Common Symbol Tape.
DOWN = Write a Common Symbol Tape.

6. Press the following console buttons in the order listed:

a. Reset Error
b. Reset P
c. Read Card
d. Manual to Automatic
e. Start

7. Contents of the console switches 17-19 are typed for operator verification. (0 = UP, 1 = DOWN).

Example: 100 = Switch 17 only

If incorrect, reset switches to the correct setting and change the status of switch 0. Return to Step 7.

8. Change the status of Switch 0 to start the assembly of the first program.

9. To process another series of programs after "END JOB" is typed:

a. Position the deck to be assembled in the card reader.
b. Set switches 17-19 to select options.
c. Change status of Switch 0.
d. Return to Step 7.

PROGRAM COMMUNICATION

Operating Delays

Under certain conditions PAL45 must communicate with the operator. A message is typed and the program delays for operator action. To contimue, change the status of switch 0.

MESSAGE	CONDITION	OPERATOR ACTION/RECOVERY PROCEDURE
PUT BLANK ON 6	The program is ready to write a Common Symbol Tape	Verify that a blank reel of tape is on tape 6 of controller 1. Change status of switch 0 to continue.
* PUT SYM TAPE ON 5	The program is ready to read a Common Symbol Tape.	Place symbol tape on Tape 5 of controller 1. Change status of switch 0 to continue.
END JOB	All assemblies are completed	See Step 9, Console Operation, to process another series of programs.
HLT PR	Printer not ready	Restore the indicated peripheral to ready status. Change status of switch 0 to contine.
HLT TP	Tape Controller not ready	
HLT CR	Reader not ready	
HLT CP	Punch not ready	
ERR PR	Print Error	Line is printed again.
ERR TP	Tape Error	The tape is repositioned by the program and the tape operation is repeated.
ERR CR	Error detected while reading a card or a card jam occurred.	Position card in error ahead of remaining cards. Restore the reader to ready status. Change switch 0 to continue.
ALM CR	Last Card in hopper has been read	This typeout followed by HLT CR signifies that the feed hopper is empty. Place remainder of deck in the reader. Restore the reader to ready status. Change switch 0 to continue.

Language Restrictions

PAL45 accepts without restrictions the PAL language described in the Process Assembler Language Manual.

* Addition

GENERAL INFORMATION

Any number of programs may be assembled at one time by PAL45. Symbolic program card deck(s), punched in the PAL format, as well as an optional system Common Symbol tape are acceptable input. Each program to be assembled requires two passes to complete assembling, listing, and punching a binary load deck.

Pass 1 forms a table of all symbols appearing in the location field together with their values. A symbolic magnetic tape is produced for the second pass input.

Pass 2 lists the assembled data and the contents of the symbolic cards. Appendix A illustrates the Program Listing Format. A binary card deck is punched to be used for loading into the 225 GE/PAC Sinulator or card-equipped GE/PAC computers. CPT42, a separate program available for GE-225 computers which have paper tape punches, translates cards to paper tape for GE/PAC computers lacking card readers. (YPF07 - CPT42 Binary Card to Paper Tape Translator/225.)

PAL45 saves the common symbols after completion of the second pass for optional use by the next program. Processing continues automatically with the assembly of the next program in the reader. A blank card following an "END" card signals that the last program of the deck has been assembled. The Common Symbol tape is written at this time if that option has been selected.

Any combination of the following options may be specified by console switch settings:

1. Switch 17 - a Common Symbol tape is read in for use by the first program and the common symbols are saved for all programs which follow in the deck.

2. Switch 18 - all Common Symbols are saved for all the programs following in the deck. (Differs from switch 17 only in that no Common Symbol tape is read in.)

3. Switch 19 - a Common Symbol tape is written after the last assembly.

   Note: when Common Symbols from other than the last program in the deck are to be saved, use options 2 and 3 together.

The Binary Card format and examples of punched cards for PAL45 are contained in Appendix B.

The table storage in PAL45 is set to hold up to 800 symbols and defined op-codes.

Input

Symbolic program card deck(s) punched in the PAL format.
System Common Symbol (Magnetic) Tape (optional).

Output

Assembly listing on the high speed printer.
Binary card load deck.

Hardware

8K Core
Card Reader and Punch
Magnetic Tape on Controller 1
High Speed Printer

REFERENCES

YPA01	LDR41	GE/PAC Loader Package
YPE04	SIM03	GE/PAC Simulator on the GE-225
YPF01	PAL41	PAL/412 Assembler
YPF07	CPT42	Binary Card to Paper Tape Translator/225
YPG12	GE/PAC	4000 Process Assembler Language (PAL) Manual

APPENDIX A

PROGRAM LISTING FORMAT

Loca-tion	Derelativized Equivalent	True Contents		Symbolic Card Contents Columns 1 - 68			I.D. Field Cols. 70 - 80
	100001000	100001000			*ØRG /1000		60002900010
001000	00004000	00004000		ALPHA	LDA /4000	COMMENTS MAY	60002000020
001001	32401006	32401006			STA BETA,4	APPEAR AFTER	60002900030
001002	26300001	26300001			INX 1,3	THE FIRST	60002900040
001003	24337773	24337773			TXH 5,3	BLANK	60002900050
001004	34001000	34077774			BTS ALPHA		60002900060
001005	14001005	14040000	U		BRU GAMMA		60002900070
001006	500000005	500000005		BETA	*BSS 5		60002900080
	140004000	140004000			ØRG /4000		60002900090

Note that BETA is an absolute label - reference to it in location 1001 is absolute in both the True and Derelativized Equivalent columns.

ALPHA is relative; therefore its reference in location 1004 is relative in the True Contents column, absolute in the Derelativized Equivalents column.

GAMMA is undefined. The error flag U appears.

Note that an asterisk appears in column seven of the first ØRG record while it is omitted from the second. An ØRG with an asterisk in column 7 produces a loader-control word which, at time of loading, will restrain the loader from accepting a relocation value. If column 7 is left blank the control code produced (sign bit is set) will instruct the loader to accept, as a relocation value, whatever was entered through the console switches

APPENDIX B - BINARY CARD FORMAT

Data Card

Column 1, row 12	- 0 if Core, 1 if Drum or Disc
Column 1, row 11	- 0 if Absolute, 1 if Relocatable
Column 1, rows 0-5	- Number of words (n) n ≤ 32
Column 1, rows 6-9	- Checksum, 4 most significant bits
Column 2	- Checksum, 12 least significant bits
Column 3, row 12	- 0
Column 3, rows 11-9	- Starting address, 11 most significant bits
Column 4	- Starting address, 12 least significant bits

Starting at column 5 are 2n columns containing n words of data, 12 bits in each column, the most significant half first.

Additional groups of starting addresses (same format as columns 1-4) and data may follow the first group if there is sufficient room before column 69.

The checksum is formed by separating each data word and the starting address into 2 parts, each 12 bits long. Each half is added to the 16 bit checksum. Overflow out of the 16th bit is ignored.

Transfer (END) Card

Column 1, rows 12-5	- 0
Column 1, rows 6-9 and Column 2	- Checksum
Column 3	- Transfer address, first half
Column 4	- Transfer address, second half

A blank card is considered a dummy transfer card; no transfer to the routine.

Identification Card (IDN)

Columns 1,2	- 0
Column 3	- 4031_8 reading top to bottom (-I)
Column 4	- 2445_8 reading top to bottom (DN)
Columns 5, 6	- 24 bits representing the first IDN word
Columns 7, 8	- 24 bits representing the second IDN word

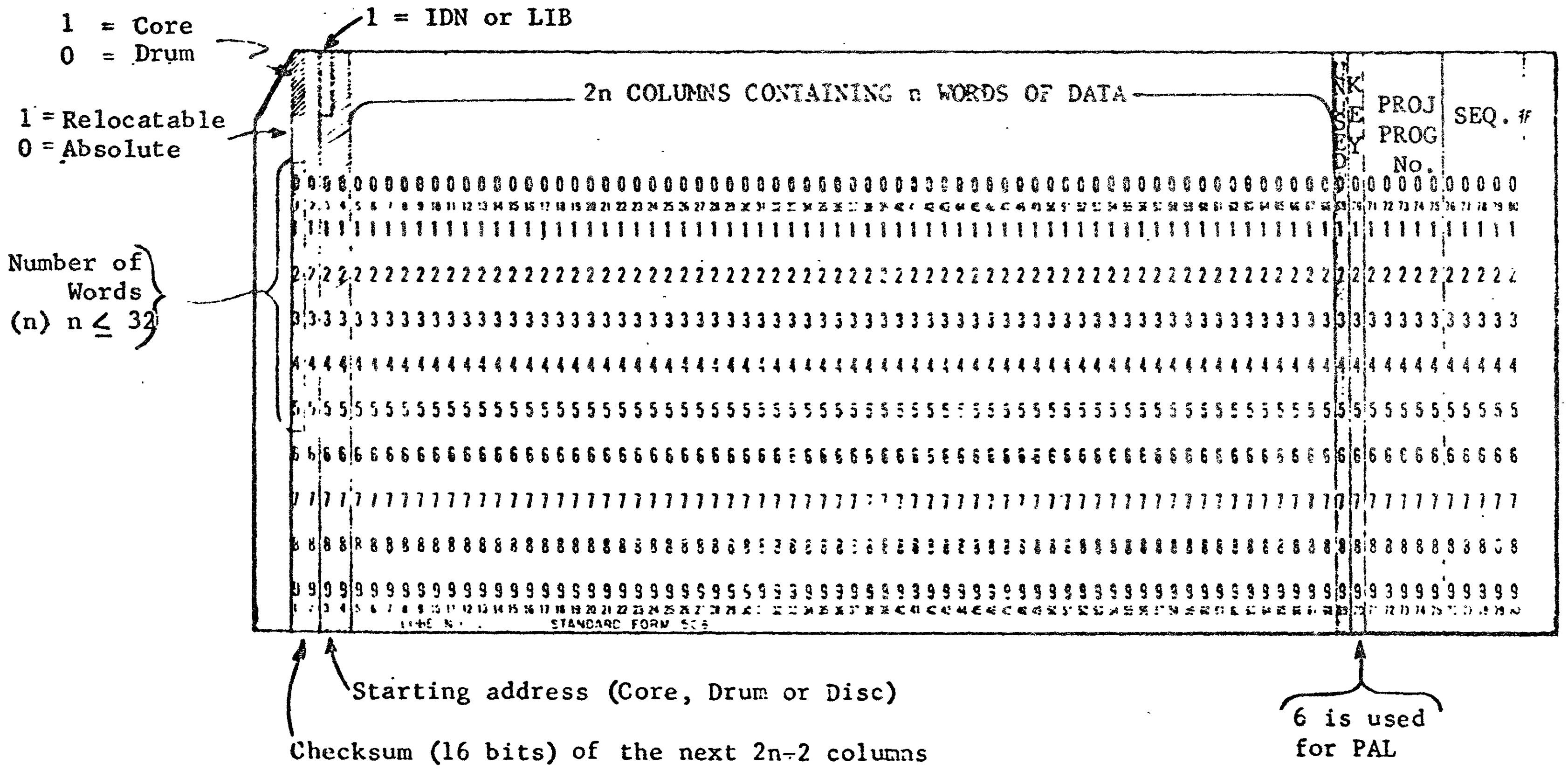
There is no checksum on the card.

LIB Card

Columns 1, 2	- 0
Column 3	- 4043_8 reading top to bottom (-L)
Column 4	- 3122_8 reading top to bottom(IB)
Columns 5, 6	- 24 bits=location of the LIB call
Columns 7, 8	- 24 bits representing the first LIB word
Columns 9, 10	- 24 bits representing the second LIB word

There is no checksum on the card.

APPENDIX B - BINARY CARD FORMAT


1 = Core
0 = Drum
1 = IDN or LIB
1 = Relocatable
0 = Absolute
Number of Words (n) n ≤ 32
2n COLUMNS CONTAINING n WORDS OF DATA
UNUSED
KEY
PROJ PROG No.
SEQ. #
Starting address (Core, Drum or Disc)
Checksum (16 bits) of the next 2n-2 columns
6 is used for PAL

Additional groups of starting addresses (same format as columns 1-4) and data may follow the first group if there is sufficient room before column 69.

The checksum is formed by separating each data word and the starting address into two parts, each 12 bits long. Each half is added to the 16 bit checksum. Overflow out of the 16th bit is ignored.

APPENDIX B - Binary Card Format, contd.

Data Card

Column 1, row 12	- 0 if Core, 1 if Drum or Disc
Column 1, row 11	- 0 if Absolute, 1 if Relocatable
Column 1, rows 0-5	- Number of words (n) $n \le 32$
Column 1, rows 6-9	- Checksum, 4 most significant bits
Column 2	- Checksum, 12 least significant bits
Column 3, row 12	- 0
Column 3, rows 11-9	- Starting address, 11 most significant bits
Column 4	- Starting address, 12 least significant bits

Starting at column 5 are 2n columns containing n words of data, 12 bits in each column, the most significant half first.

Additional groups of starting addresses (same format as columns 1-4) and data may follow the first group if there is sufficient room before column 69.

The checksum is formed by separating each data word and the starting address into two parts, each 12 bits long. Each half is added to the 16 bit checksum. Overflow out of the 16th bit is ignored.

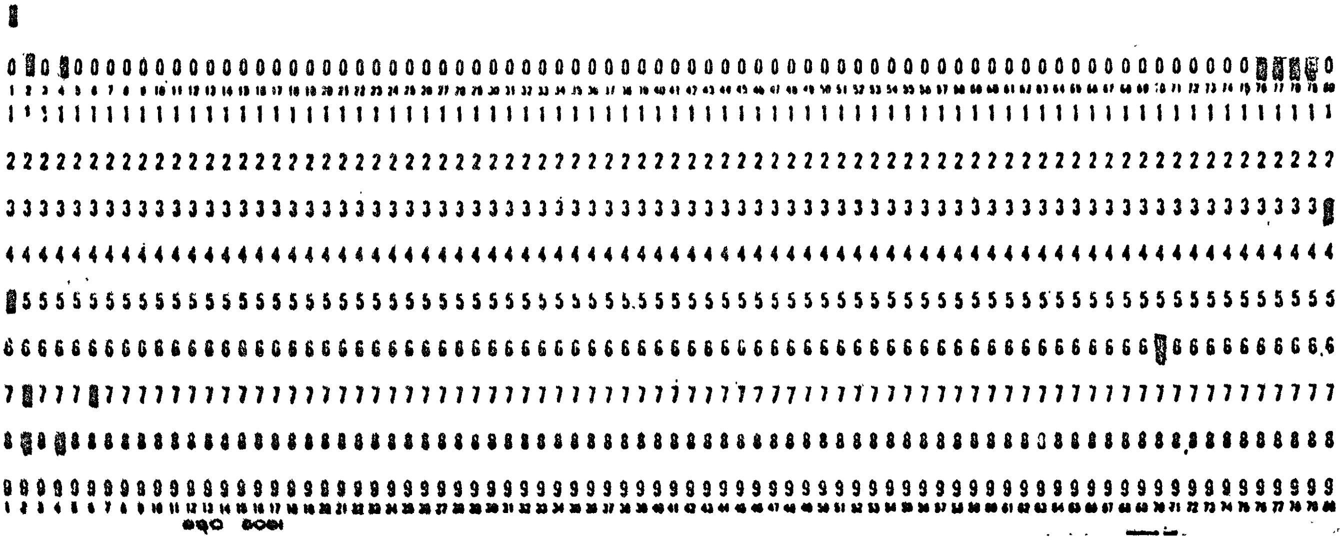


Figure 3 - EXAMPLE OF DATA CARD

Core, relocatable starting address
Card contains 1 data word
Checksum is 001006_8
Starting address is 1002_8
Data word is 00000004

APPENDIX B - Binary Card Format, contd.

Transfer (END) Card

Column 1, rows 12-5	- 0
Column 1, rows 6-9 and Column 2	- Checksum
Column 3	- Transfer address, first half
Column 4	- Transfer address, second half

A blank card is considered a TRA card - no transfer.

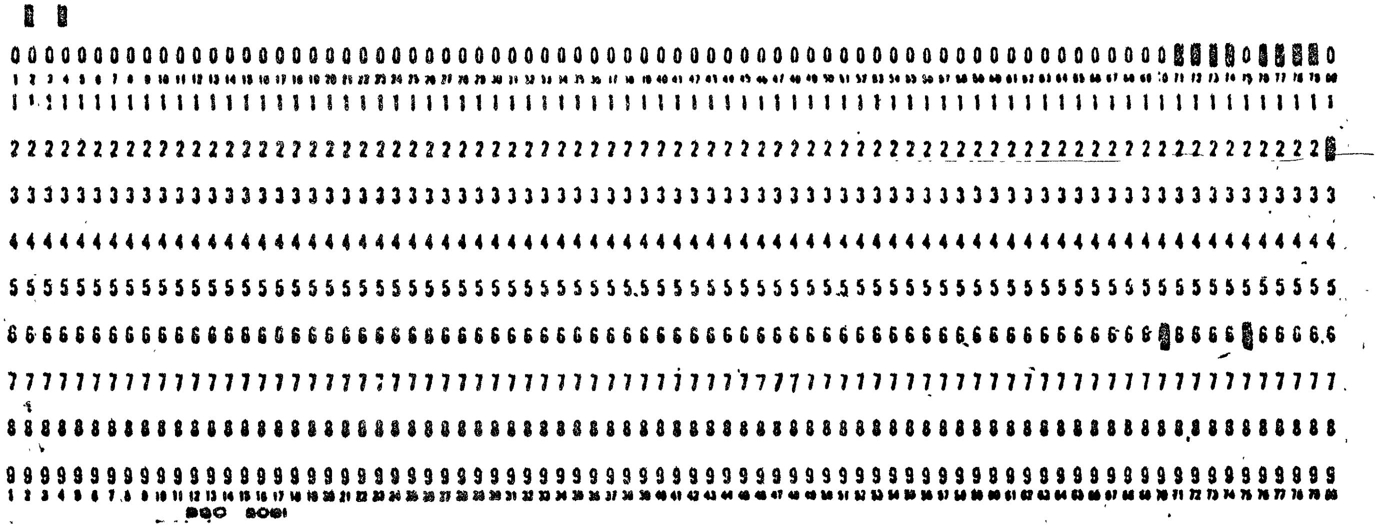


Figure 4 - EXAMPLE OF TRANSFER CARD

Checksum is 002000_8
Transfer starting address is 00002000 (absolute starting address of 2000_8)

APPENDIX B - Binary Card Format, contd.

Identification Card (IDN)

Columns 1, 2	- 0
Column 3	- 4031$_8$ reading top to bottom (-I)
Column 4	- 2445$_8$ reading top to bottom (DN)
Columns 5, 6	- 24 bits representing the first IDN word
Columns 7, 8	- 24 bits representing the second IDN word

There is no checksum on the card.

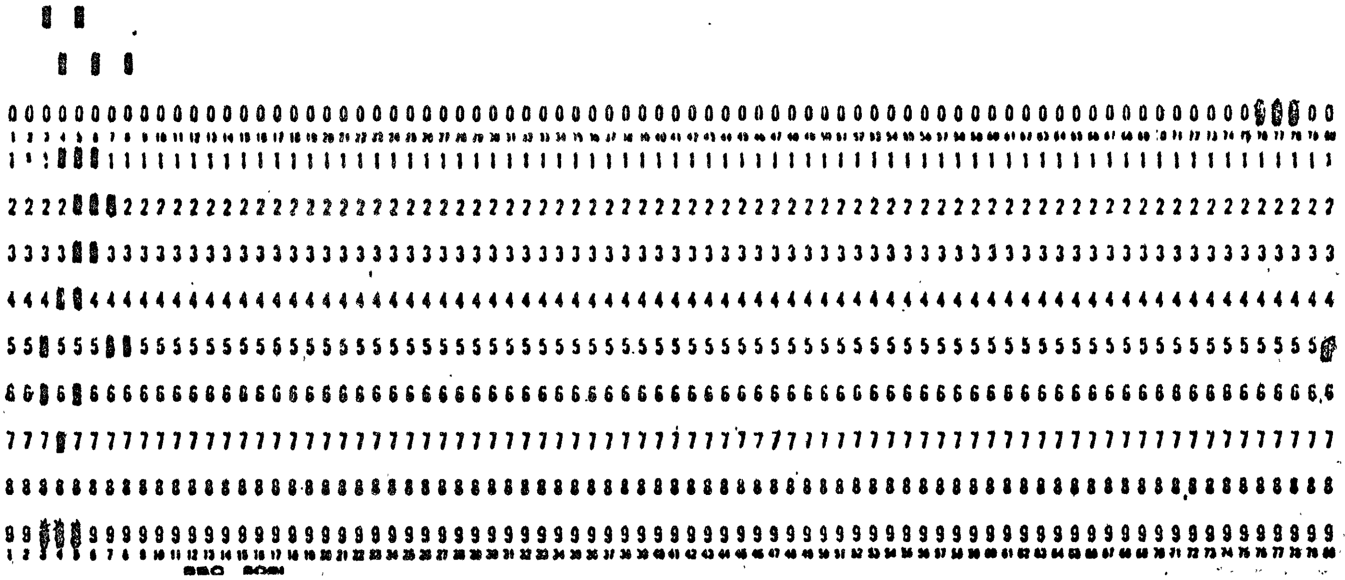


Figure 5 - EXAMPLE OF IDN CARD

Columns 5, 6 (47512700)
Columns 7, 8 (02202020)
} Coded in GE/PAC code; identifies the program as PRG02

APPENDIX B - Binary Card Format, contd.

Library Card (LIB)

Columns 1, 2	- 0
Column 3	- 4043_8 reading top to bottom (-L)
Column 4	- 3122_8 reading top to bottom (IB)
Columns 5, 6	- 24 bits - location of the LIB call
Columns 7, 8	- 24 bits representing the first LIB word
Columns 9, 10	- 24 bits representing the second LIB word

There is no checksum on the card.

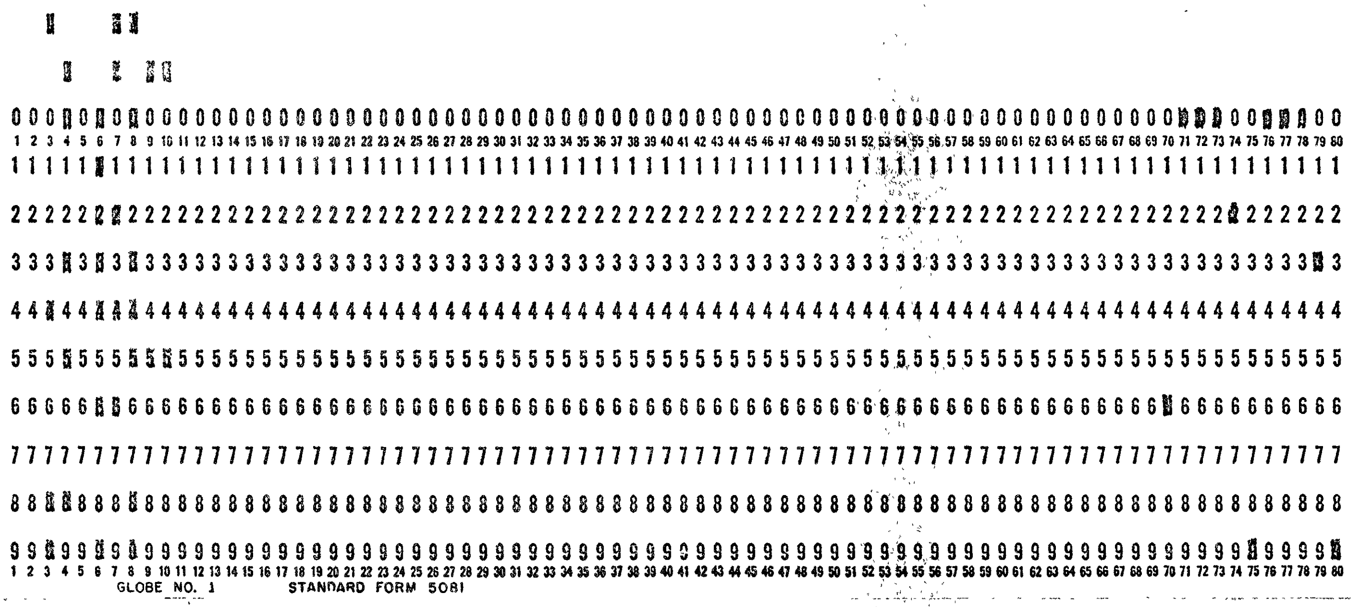


Figure 6 - EXAMPLE OF LIB CARD

Columns 5, 6	00001751	location
Columns 7, 8	62505163	Coded in GE/PAC code: Call Program SQRT
Columns 9, 10	20202020	

GE/PAC 4000

FORTRAN REFERENCE MANUAL

May 1965 revised

Library Control No. YPG14M

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER SECTION
P. O. BOX 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

GENERAL ELECTRIC
PROCESS COMPUTERS

GE/PAC 4000 FORTRAN REFERENCE MANUAL	Library Control No. YPG14M

REVISION CONTROL SHEET

APPROVED BY: FRWeinberger DATE: May 25, 1965

/.	RECORD OF CHANGE	DATE	REV.	RECORD OF CHANGE	DATE
	Page 2; para. 6	3/9/65			
	" 9; additional functions				
	" 22; para. 3				
	" 30; " 2, 4				
	" 33; " 2, 3, 4				
	" 34; " 2				
	" 42; " 3, 5				
	" 49; page numbers				
	FRWeinberger				

5/65

PAGE
1

REISSUED

CONTENTS

FOREWORD

This is a reference manual for the GE/PAC FORTRAN language. A previous familiarity, on the part of the reader, with basic FORTRAN concepts is assumed. No attempt is made to present the essentials of FORTRAN in a form usable for a fundamentals course in FORTRAN language usage. The primary purpose of this manual is to describe the specific statements and capabilities of the GE/PAC FORTRAN language.

Additions have been made to the FORTRAN II language to increase programming flexibility. In addition, they provide the programmer with special statements which enable him to produce a total real-time system.

Existing FORTRAN II programs can be compiled in GE/PAC FORTRAN, provided that the specific restrictions noted herein are not violated.

INTRODUCTION TO GE/PAC FORTRAN

* A GE/PAC FORTRAN program is a sequence of statements, each of whose characteristics is described in this manual. These statements may be classified according to the following general categories.

1. COMPUTATION statements which comprise the working body of the program; e.g., numerical calculation and bit manipulation.

2. CONTROL statements which specify the flow of control during execution.

3. DECLARATION statements which supply information about the program.

4. INPUT-OUTPUT statements which provide communication with the system peripherals.

5. PAL LANGUAGE statements which may be included in the program in order to "tailor" certain critical areas.

Program Preparation

The statements representing a FORTRAN program are first entered on a coding form similar to that of Figure 1. The lines of the coding form are divided into sixty-nine columns, each of which may contain one character. Each statement is written on a separate line; if more than one line is required, as many as four additional lines may be used as a "continuation" of the statement.

Columns 1 through 5 may be used for a statement number. Such numbers may be used as labels to which other statements in the program refer.

A non-zero, non-blank character in column 6 indicates that the line is a continuation line.

The body of the statement itself is entered in columns 7 through 69 of the initial and any necessary continuation lines.

* - Revised 1/65

Column 70 must contain a "7" in all lines of all statements of categories 1 through 4 above. Lines which contains a "6" in column 70 are of category five. All category five statements are transmitted unchanged into the object program.

A "C" in column 1 indicates that the statement is a comment. Like category 5 statements, comment statements are passed directly into the object program at the point of encounter. The text of the comment may be entered in columns 2 through 69 of the initial line and in columns 7 through 69 of continuation lines.

A "B" in column 1 indicates the statement is of type "Boolean" and that certain numerical calculations contained in the statement are to be performed with logical rather than arithmetic operations.

Except for columns 1 and 6 and certain alphanumeric fields, blanks are ignored and may be used freely to increase legibility.

The first line of any program, irrespective of content, is assumed to be the title line and contains the text (columns 1 through 39) which will head pages of the output listing and the program identification number (columns 71 through 75) which will be transmitted to all generated object lines.

* The last line of a program must be an END statement. An END statement may contain no other characters in columns 7 through 69 except "END".

Sample Program

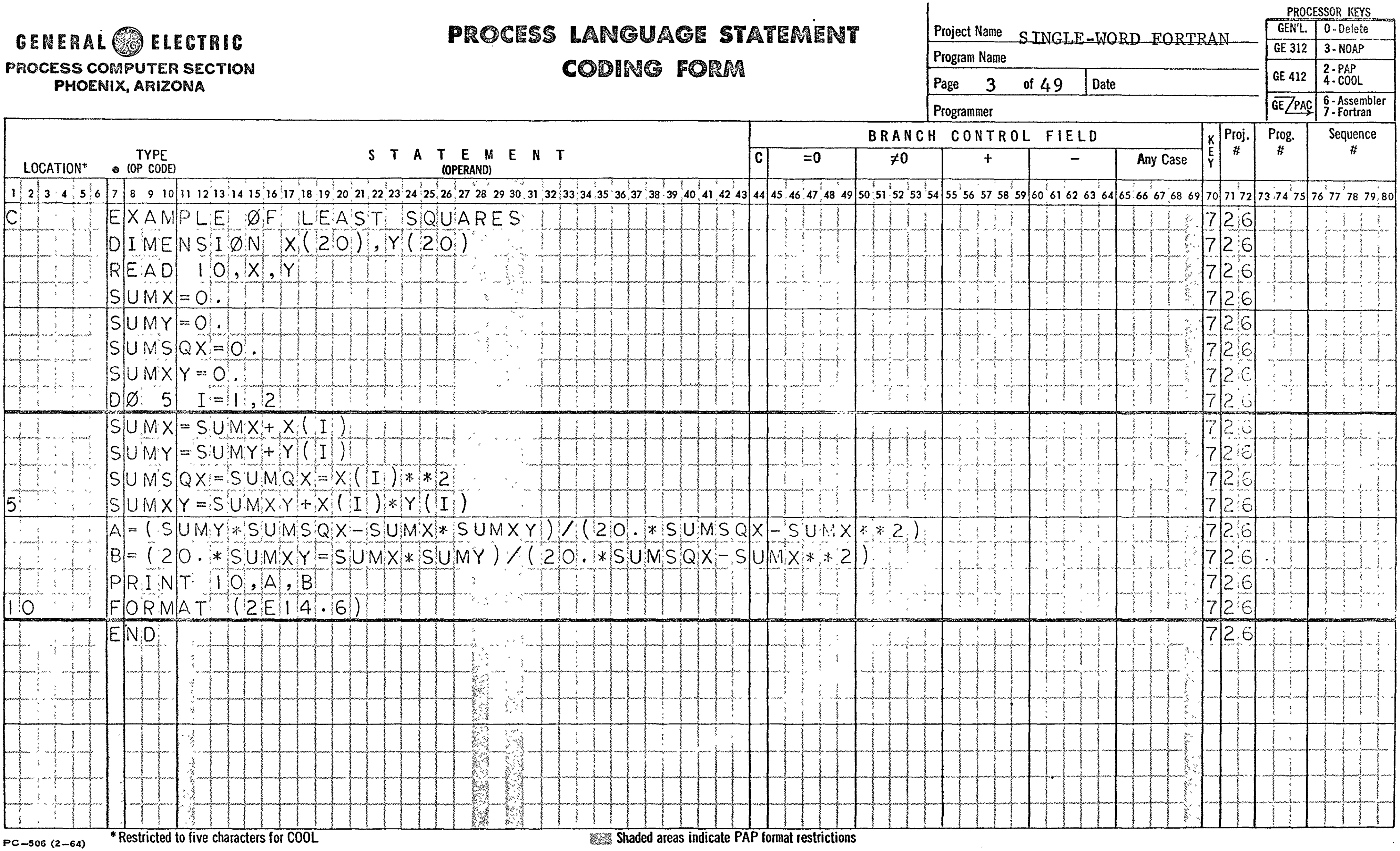
It is desired to fit the best straight line approximation to a population of data by the method of least squares. The following program indicates how this would be done using FORTRAN. The general formula for the approximation is of the form y = a+bx. To solve for a and b by the method of least squares we evaluate the formulas:

$$a = \frac{(\Sigma y) \quad (\Sigma x^2) \quad - \quad (\Sigma x) \quad (\Sigma xy)}{(n) \quad (\Sigma x^2) \quad - \quad (\Sigma x)^2}$$

$$b = \frac{(n) \quad (\Sigma xy) \quad - \quad (\Sigma x) \quad (\Sigma y)}{(n) \quad (\Sigma x^2) \quad - \quad (\Sigma x)^2}$$

Line 1 is the TITLE CARD. It is also a FORTRAN comment card. The Sample Program is illustrated in Figure 1 on the following page.

* - Revised 5/65



GENERAL ELECTRIC
PROCESS COMPUTER SECTION
PHOENIX, ARIZONA

PROCESS LANGUAGE STATEMENT CODING FORM

Project Name SINGLE-WORD FORTRAN
Program Name
Page 3 of 49 Date
Programmer

PROCESSOR KEYS	
GEN'L.	0 - Delete
GE 312	3 - NOAP
GE 412	2 - PAP 4 - COOL
GE/PAC	6 - Assembler 7 - Fortran

```
LOCATION  TYPE/STATEMENT                                         KEY PROJ.
C         EXAMPLE ØF LEAST SQUARES                                7  26
          DIMENSIØN X(20),Y(20)                                   7  26
          READ 10,X,Y                                             7  26
          SUMX=0.                                                 7  26
          SUMY=0.                                                 7  26
          SUMSQX=0.                                               7  26
          SUMXY=0.                                                7  26
          DØ 5 I=1,2                                              7  26
          SUMX=SUMX+X(I)                                          7  26
          SUMY=SUMY+Y(I)                                          7  26
          SUMSQX=SUMQX=X(I)**2                                    7  26
5         SUMXY=SUMXY+X(I)*Y(I)                                   7  26
          A=(SUMY*SUMSQX-SUMX*SUMXY)/(20.*SUMSQX-SUMX**2)         7  26
          B=(20.*SUMXY=SUMX*SUMY)/(20.*SUMSQX-SUMX**2)            7  26
          PRINT 10,A,B                                            7  26
10        FORMAT (2E14.6)                                         7  26
          END                                                     7  26
```

PC-506 (2-64) *Restricted to five characters for COOL ▨ Shaded areas indicate PAP format restrictions

FIGURE 1

The dimension statement of line 2 declares X and Y to be one dimensional arrays each containing 20 numbers. This declaration sets aside two groups of 20 consecutive storage spaces for X and Y and allows them to appear as subscripted variables within the program.

Line 3 is an input statement that reads a record of previously prepared data from tape under control of FORMAT statement 10 and places it in the consecutive locations of the X and Y arrays.

Calculation actually begins with line 4 in which a variable called SUM is set equal to zero. Similar operations are performed on lines 5, 6, and 7.

Line 8 sets up a series of repetitive calculations in what is known as a DO loop. It causes the calculations up to and including statement 5 on line 12 to be performed 20 times. The subscripting variable I is set to 1 for the first execution of the following four instructions and is increased by 1 for each subsequent execution. In this manner it is possible to refer to successive values of X and Y for calculation purposes.

In line 9 a value of X is added to SUM and the result is assigned to SUM. A similar operation is done for Y on line 10.

In line 11 a value of X is squared, added to SUMSQX, and assigned to SUMSQX.

In line 12 a value of X is multiplied by a value of Y, added to SUMXY, and assigned to SUMXY.

After the previous four calculations have performed 20 times, control passes from the DO loop to the statement on line 13 in which A is calculated. Then B is calculated on line 14.

Line 15 is an output statement which causes the values of A and B to be printed on-line according to FORMAT statement number 10 (line 16). This FORMAT statement indicates two answers are to be printed on the same line, each answer occupies 14 spaces (including blanks) and there are 6 digits after the decimal point.

Line 17 is a control card indicating the END of the example.

COMPUTATION

Basic Elements

* The basic elements used as operands in FORTRAN computations are constants, variables, and functional references, all of which represent numerical quantities. In the FORTRAN language, these elements are represented by symbols composed of character strings.

Quantities

Two types of numerical quantities are recognized by FORTRAN, arithmetic and logical. Within the designation arithmetic, two modes are recognized, real and integer. Integer quantities represent integers within the range -8,388,608 (-2^{23}) through 8,388,607 ($2^{23}-1$) inclusive. Real quantities represent real numbers and are represented in the computer by floating-point configurations comprised of a 6 bit exponent and a 17+sign bit mantissa. The approximate range of floating quantities is -10^9 through -10^{-9}, 0 and 10^{-9} through 10^9.

Within the designation logical, only octal digits (0 through 7 inclusive) are recognized. A maximum value of 77777777 is allowed.

Constants

Constants are numbers of arithmetic or logical type which appear in the source program in explicit form.

Arithmetic integer constants are written as a string of decimal digits.

Examples:

```
0
1
1964
```

Logical integer constants are written as a string of octal digits.

Examples:

```
0
252525
77777777
```

In either type, the integer represented must lie within the range specified above for integer quantities.

Real constants are written as a string of decimal digits which includes a decimal point.

Examples:

```
 .0
1.
2.71828
```

Real constants may be given a scale factor by appending an "E" followed by an integer constant, which indicates the power of ten by which the number is to be multiplied. This scale factor may be preceded by a "+" or "-" sign to indicate positive or negative powers of ten. If no sign is given, it is assumed to be positive.

Examples:

1.E-12	means 10^{-12}
.00314159E+3	means 3.14159
19.64E+2	means 1964.

As another alternative, a real constant may be expressed as an integer constant followed by a scale factor.

Examples:

55E-3	means .055
132E45	means 132×10^{45}
69E4	means 690000

In any representation the number must lie within the range described above for floating quantities.

Identifiers

Identifiers are used to name the variables, subprograms, and dummy arguments which appear in a FORTRAN program. An identifier is a string of letters and digits, the first of which must be a letter. The string may be any desired length, but only the first six characters will be used. Identifiers may be declared * as integer or real, either explicitly or implicity. Implicit definition depends upon the first letter of the identifier. If the first letter is from the group (I, J, K, L, M, and N) then the identifier is of the class integer. If the first letter is not from this group then it is of class real. Explicit definition of identifiers is accomplished by the declarative statements REAL and INTEGER.

Examples:

Real	RATE DECREMENT
Integer	L307 MØNTH

* - Revised 1/65

Variables

Variables represent quantities which may assume many different values and are referred to by name. They may be either scalar or array variables, depending on the nature of the quantity they represent.

Scalar Variables

Scalar variables represent a single real or integer quantity and are written as simple identifiers.

Examples:

```
DISTANCE
Z
N3
```

Array Variables

An array variable represents a single element within an array of quantities. The array variable is denoted by the array name followed by a subscript list enclosed in parentheses. The subscript list contains one or more arithmetic expressions separated by commas. Each expression corresponds to a subscript and the values of the expressions determine which array element is to be referenced. The number of subscripts in the list must equal the number of dimensions specified for the array.

Examples:

```
A (4)
BETA (M+3)
JOHN (2*ITEM-13)
```

Subscripts

A subscript may be any expression of the type, arithmetic; however, its significance is of integer mode and limited in range by the size of the array dimension involved. Therefore, when necessary, the value of any expression used as a subscript is truncated to an integer and reduced modulo 2^{14} before being employed in reference.

Single-Bit Arrays

* As the operands of certain GE/PAC 4000 FORTRAN statements (SET BIT, RESET BIT, IF BIT), the individual bits in the binary representations of scalar variables (not array variables) may be referred to by appending a single subscript (in parentheses) to the scalar identifier. The subscript is calculated as any other subscript but refers to the bit position in the memory word and consequently has meaning only in the range 0 through 23. The appearance of a scalar identifier with a subscript is legal only in SET BIT, RESET BIT and IF BIT statements.

Examples:

```
FLAGS (J)
TRIGRS (13)
INHIB (XMINOF(J,K+2))
```

Function References

A function is a subprogram which acts upon one or more quantities called arguments to produce a single quantity called the function value. Functional references are denoted by the identifier which names the function, followed by an argument list enclosed in parentheses.

```
identifier (argument, argument, ..., argument)
```

An argument may be an expression or an array identifier.

The function value may in turn act as an element whose mode is determined by the mode of the identifiers naming it, or by use of the appropriate form of the FUNCTION statements, and may, therefore, be independent of the types and modes of its arguments.

Examples:

```
ATANF (ALPHA)
DATE (MONTH, DAY, YEAR)
GAMMA (N, Z*SQRTF(ZETA))
```

Reserved Functions

Certain commonly and frequently used arithmetic functions are provided as part of the system library and will be incorporated in the compiled program by the compiler through the use of a "LIB" operation. The names of these functions must, therefore, be "reserved" and limited to use as subprogram identifiers for the library subprograms to which they refer.

The following table lists these names together with information about the functions themselves.

			Modes I=integer R=real	
Subprogram Identifier	Function	Number of Args	Function Value	Arg(s)
ATANF	Trigonometric Arctangent	1	R	R
SINF	Trigonometric Sine	1	R	R
COSF	Trigonometric Cosine	1	R	R
SQRTF	Square Root	1	R	R
LOGF	Log base e	1	R	R
EXPF	exponential (e to a power)	1	R	R
ABSF	absolute value	1	R	R
XABSF	absolute value	1	I	I
SINHF	Hyperbolic sine	1	R	R
COSHF	Hyperbolic cosine	1	R	R
TANHF	Hyperbolic tangent	1	R	R
MODF	$arg_1 - [arg_1/arg_2]\ arg_2$ (See Note)	2	R	R
XMODF	arg_1 modulo arg_2	2	I	I
SIGNF	Signum (arg_2) *arg_1	2	R	R
XSIGNF	Signum (arg_2) *arg_1	2	I	I
DIMF	(arg_1-arg_2) if $arg_1 > arg_2$; else 0	2	R	R
XDIMF	(arg_1-arg_2) if $arg_1 > arg_2$; else 0	2	I	I
MAXOF	Maximum value of args.	Var.	R	I
MAX1F	Maximum value of args.	Var.	R	R
XMAXOF	Maximum value of args.	Var.	I	I
XMAX1F	Maximum value of args.	Var.	I	R
MINOF	Minimum value of args.	Var.	R	I
MIN1F	Minimum value of args.	Var.	R	R
XMINOF	Minimum value of args.	Var.	I	I
XMIN1F	Minimum value of args.	Var.	I	R
INTF	Integer part (truncation)	1	R	R
XINTF		1	I	R
XFIXF		1	I	R
FLOATF	Float an integer	1	R	I

NOTE - [ ] indicate "greatest integer in"

EXPRESSIONS

An expression is a sequence of elements separated by operational symbols and/or parentheses in accordance with conventional mathematical notation and certain FORTRAN restrictions. Two types of expressions are recognized, arithmetic and logical.

Formation of Expressions

An expression has a single numeric value equal to the result of the calculation specified by the numeric quantities and arithmetic or logical operations comprising it. The arithmetic operational symbols are "+", "-", "*", "/", and "**", denoting respectively addition, subtraction, multiplication, division, and exponentiation.

A logical or "Boolean" expression is formed in the same fashion as an arithmetic expression except that the operational symbols recognized are "+", "/", "*", and "-", denoting logical sum ("or"), logical difference ("exclusive or"), logical product ("and") and logical complement ("not") respectively. An expression is declared Boolean by the placement of a "B" in column 1 of the first line of the FORTRAN statement in which it occurs. In any statement declared Boolean, all operational symbols in expressions are interpreted in the logical sense except in those expressions used as subscripts to arrays (subscript expressions are universally of type arithmetic-integer). In Boolean expressions, all constants are interpreted as type logical-integer and appear as octal integers. All other representations are illegal.

An expression may be as simple as a single element, i.e., a constant, variable, or function reference.

Examples:

```
3.142
OMEGA (T)
COSF (DELTA)
```

Compound expressions may be formed by using operation symbols to combine basic elements.

Examples:

```
Z+2
SUMX/N
SQRTF(B**2-4*A*C)
```

An expression may be enclosed in parentheses and considered as a basic element.

Examples:

```
(Z-3)/LAMBDA
(ALEPH)
COSF (SINF(2*PI*R*T))
```

Any expression may be preceded by a "+" or "-" sign.

Examples:

```
+TEN
-(A*B)
-SINF (+ALPHA)
```

With the exception of logical complement, no two operational symbols may appear in sequence. The expression:

```
X*-Y
```

is illegal in arithmetic expressions, but is allowed in Boolean expressions. The use of parentheses, yields the correct arithmetic form:

```
X*(-Y)
```

Adherence to the above rules will allow the formation of all permissible expressions.

If the precedence of arithmetic operations is not stated explicitly by parentheses, it is implicitly interpreted to be as follows, in order of decreasing precedence:

Symbol	Operation
**	exponentiation
* and /	multiplication and division
+ and -	addition and subtraction

In Boolean expressions, the implied precedence in decreasing order is:

Symbol	Operation
~	logical complement
*	logical product
/	logical difference
+	logical sum

For example, the expression

```
U*V+W/X**Y+Z
```

is taken to be

```
(U*V)+(W/(X**Y))+Z
```

Since sequences of operations of equal precedence can result in ambiguities, they are resolved by grouping from the left. Thus

```
A**B**C and
X/Y/Z
```

are interpreted as

```
(A**B)**C and
(X/Y)/Z
```

respectively.

Evaluation of Expressions

Except for Boolean expressions, which are modeless, the numerical value of any expression may be of integer or real mode, as determined by the modes of its elements. There are three possible combinations: all elements are integer (integer expression); all elements are real (real expression); or both real and integer elements occur (mixed expression). All combinations are permissible in GE/PAC 4000 FORTRAN.

Integer Expressions

An integer expression is evaluated using integer arithmetic throughout to yield an integer result. Fractional parts arising in division are truncated, not rounded. For example, 5/3 gives 1, 4/7 gives 0.

Examples:

```
L
I*2+m
(J-3)*MAN+INDEX
```

Real Expressions

A real expression is evaluated using floating-point arithmetic throughout to yield a real result.

Examples:

```
(Y(N+1)+Y(N-1))/2.*DX)
COSF(ALPHA+BETA)
```

Mixed Expressions

Mixed expressions are evaluated by converting all integer variables to real variables and then treating the expression as if it were real. The result is given as a real quantity.

Examples:

```
Z*2*(I+L)
A**J+J**C
R(K+2)*ATANF(X1)
```

Boolean Expressions

Boolean expressions are evaluated using full memory word logical operations upon the elements without regard to mode. All constants encountered are interpreted in octal. Array elements may occur, the subscripts of which will be interpreted as arithmetic-integer expressions. All other elements (e.g., function values, expression values) are treated as modeless full-word binary configurations.

Examples:

```
WORD*MASKI+FIELD
I*77/COMUTR
A(2*I-5)*(-B(2*I)/-C(I))
```

COMPUTATION STATEMENTS

Assignment Statement

The assignment statement specifies an expression which is to be evaluated, and a variable called the statement variable to which the expression value is to be assigned.

Form: variable = expression

Note that the "=" sign means replacement, not equality. The first example below is not a mathematical equation but a valid assignment statement meaning "take the value of Z, cube it and assign the result to X."

Examples:

```
X = Z**3
J = K*(L-5)
A(X) = LOGF (1+2*X)
```

The value of the expression in an assignment statement is made to agree with the mode of the statement variable before the replacement is performed. If the statement variable is real, an integer expression value will be converted prior to replacement, and conversely. For example, in the statement

```
A = 3*J+K
```

the integer value of the expression is converted to floating-point before assignment to A.

Bit Assignment Statements

The bit assignment statements, SET BIT and RESET BIT, provide for the assignment of values (1 or 0) to the individual bits of scalar variables.

Form: SET BIT scalar variable (subscript),...,scalar variable (subscript)

RESET BIT scalar variable (subscript),..., scalar variable (subscript)

The individual bits referred to in the list following the SET BIT or RESET BIT preamble are "set" (set = 1) or "reset" (set = 0) as indicated.

Examples:

```
SET BIT JOE (2*I-1), SAM(23),FLAG(PERIF(J))
RESET BIT X(3),FLAG(PERIF(J+1)),LOC(0)
```

CONTROL STATEMENTS

In a FORTRAN program, control normally passes sequentially from one statement to the next in the order in which they are presented to the compiler. Control statements allow the programmer to alter this normal program flow. To implement this, FORTRAN source statements may be labeled with numbers which are referred to by control statements.

Statement Numbers

A statement number consists of an unsigned integer constant of up to five digits. Leading zeros are ignored. The value of the integer must be greater than zero.

Although statement numbers appear in the source program as integers they must not be confused with numerical quantities. They represent a distinct type of quantity in a FORTRAN program known as a statement number, and are used for the identification of addresses within the object program.

Since statement numbers are used for identification they must be uniquely defined; i.e., no two statements may have the same number. No order or sequence is implied by statement number magnitudes. Non-referenced statements need not be numbered, in fact it is wasteful of compiler storage to do so unnecessarily.

Unconditional GO TO Statement

Form: GO TO n

when n is a statement number.

This statement transfers control to the statement numbered n.

Example:

```
GO TO 13
```

Computed GO TO Statement

The computed GO TO statement transfers control to one of a group of statements; the particular statement chosen is determined by the computed value of an expression.

Form: GO TO ($n_1, n_2, n_3, \ldots, n_k$), expression

where $n_1, n_2, n_3, \ldots, n_k$ are statement numbers. Control will be transferred to statement number $n_1, n_2, n_3, \ldots, n_k$ depending on whether the expression has the value 1,2,3,..., or k, respectively.

Example:

```
GO TO (37,24,36), SIZE
```

will transfer control to statement number 24 if SIZE has the value 2. The value of the expression will be truncated if required. Expression values outside the range 1,2,3,...,k will cause a runtime error indication.

Example:

```
GO TO (1,2,3,27), Y+2
```

will transfer control to statement number 27 if Y has the value 2.6183, but a value of 3.142 for Y will cause an error indication.

Assigned GO TO Statement

Form: GO TO variable or
GO TO variable, ($n_1, n_2, n_3, \ldots, n_k$)

This statement transfers control to the statement whose number was last assigned to the variable by an ASSIGN statement. The variable must appear in a previously executed ASSIGN statement, or a runtime error indication will result.

Examples:

```
GO TO JAIL
GO TO ERROR
```

The variable of an assigned GO TO statement is a control variable and has a statement number rather than a numeric quantity as its value. A control variable may be shared between a program and its subprograms as may any other variable.

The ($n_1, n_2, n_3, \ldots, n_k$) is a parenthetical statement number list of footnote value only and may be included or omitted at the option of the user.

ASSIGN Statement

Form: ASSIGN integer TO variable

This statement sets the value of the variable to be used by a subsequent assigned GO TO statement. The integer is the number of the statement to which control will be transferred by the assigned GO TO statement.

Examples:

```
ASSIGN 37 TO JAIL
ASSIGN 3 TO ERROR
```

This FORTRAN capability is very useful in transferring to a sequence of statements that is used as a subroutine by other parts of the program. For example, the statement

```
        ASSIGN 13 TO EXIT
        GO TO 44
13      ...
```

will transfer control to the sequence beginning at 44. If the sequence ends with

```
GO TO EXIT
```

control will be transferred back to statement 13.

IF Statement

Form: IF (expression) n_1,n_2,n_3

where n_1,n_2,n_3 are statement numbers. This statement transfers control to statement n_1,n_2, or n_3 depending on whether the value of the expression is less than, equal to, or greater than zero respectively.

Examples:

```
IF(Y(I)-LIMIT) 6, 12, 18
IF(SUM) 3, 4, 5
```

In the first example above control is transferred to statement number 6 if Y(I)<LIMIT, to statement 12 if Y(I)=LIMIT, and to statement 18 if Y(I)>LIMIT.

IF BIT Statement

Form: IF BIT (scalar variable(subscript))n_1,n_2

where n_1 and n_2 are statement numbers. This statement transfers control to the statement numbered n_1 if the bit of the variable referred to is a "1" and to statement n_2 if it is a "0".

Examples:

```
IF BIT (FLAG(NOFAIL)) 14, 13
IF BIT (X(23)) 6, 7
IF BIT (J(0)) 9, 10
```

The second and third examples accomplish negative-positive and odd-even tests respectively.

IF SENSE SWITCH Statement

Form: IF(SENSE SWITCH expression) n_1,n_2

where n_1 and n_2 are statement numbers.

This statement provides a test for the individual status of each of the 24 switches on the GE/PAC 4000 console. These switches are numbered from 23 through 0, going from left to right. The expression may be of either mode but will be converted to arithmetic-integer in use and has meaning only in the range 0 through 23. If the console switch specified by the value of the expression is in the down (=1) position at the time of execution, control is transferred to statement n_1; if it is in the up (=0) position, control transfers to statement n_2.

Examples:

```
IF (SENSE SWITCH 7) 102, 103
IF (SENSE SWITCH IOREDY) 6, 7
```

IF ACCUMULATOR OVERFLOW Statement

Form: IF ACCUMULATOR OVERFLOW n_1, n_2

where n_1 and n_2 are statement numbers.

This statement provides a test of the GE/PAC 4000 overflow trigger. Control passes to statement n_1 if the overflow trigger is "on" and to n_2 if the overflow trigger is "off" at the time of execution. In either case, execution of the test resets the overflow trigger to the "off" state.

Example:

```
IF ACCUMULATOR OVERFLOW 23, 31
```

IF I/O Statements

A series of IF statements has been provided for testing the status of individual I/O devices. These statements are described fully in the INPUT-OUTPUT section of this manual.

DO Statement

The DO statement allows a series of statements to be executed repeatedly under the control of a variable whose value changes between repetitions.

Forms: DO n integer scalar variable = expression_1, expression_2
DO n integer scalar variable = expression_1, expression_2, expression_3

* where n is a statement number and $\text{expression}_1 \leq \text{expression}_2$ at object time. If, as in the first form, expression $_3$ is not stated, it is assumed to have the value, 1.

The DO statement causes the following statements up to and including statement n to be executed repeatedly. This group of statements is called the range of the DO statement. The scalar variable of the DO statement is called the index or induction variable and must be of integer mode. The values of expression_1, expression_2, and expression_3 are called respectively the initial, limit and increment values of the index. Each may be of either mode but will be converted to arithmetic-integer mode before use.

The initial execution of all statements within the range is always performed with the initial value assigned to the index, regardless of the value of the limit and increment. After each execution of the range the increment is added to the value of the index and the result compared with the limit. If the result has not passed the limit, the statements within the range are again executed using the new value of the index.

After the last execution program control passes to the statement immediately following statement n. Exit from the range may also be accomplished by a transfer from within the range of the DO statement. The value of the index is retained for computation purposes on both normal and abnormal exits from the DO loop.

The range of a DO statement may include other DO statements provided that the range of an "inside" DO loop is completely contained within the range of any "outside" DO loop. In other words, the range of two DO statements may not partially intersect each other. Only total intersection or no intersection is allowed.

The index of a DO statement is treated as any other scalar variable. Its value may be used for calculation outside the range of the DO statement as well as within the ranges. In addition, the values of the limit, increment, and index may be altered within the range of the DO statement.

*

It is also permissible to transfer into the range of a DO statement from outside the range.

Examples:

```
DO 37 I= 3, 12
DO 15 INDEX = FIRST, LAST, INCREMENT
```

As an example of the use of the DO statement consider the following sequence which will sum all the numbers within a suitably specified array.

```
      SUM = 0.
      DO 3 I=1,N
    3 SUM = SUM+X(I)
```

CONTINUE Statement

Form: CONTINUE

The rules of FORTRAN state that the range of a DO statement cannot end with a control transfer statement. In order to gain this capability without violation of the rule, a dummy statement, CONTINUE, is provided that may be used to end the range of a DO, or as the target point for transfer statements within the range of a DO where repetition of all or part of the range is conditional. Consider the following statement sequence:

* - Revised 1/65

```
          DO 3 I=START, STOP
            .
            .
            .
          IF (ALPHA) 3, 17, 51
     3    CONTINUE
```

A negative value of ALPHA will initiate another execution of the range. The CONTINUE statement provides a target address for the IF statement and ends the range of the DO statement. The following sequence is illegal and must not be used since the DO loop ends with a conditional transfer.

```
     2    DO 3 I=START, STOP
            .
            .
            .
     3    IF (ALPHA) 2, 17, 51
```

CALL Statement

Forms: CALL identifier
CALL identifier (argument, argument, ..., argument)

This statement is used to call (transfer control to) a subroutine subprogram. The identifer is the name of the subroutine.

The arguments, as in the case of functions, may be expressions or array identifiers. Unlike a function, however, a subroutine may have more than one result and may use one or more of its arguments to return these results to the calling program. The first form of the CALL statement is used where a subroutine requires no arguments.

Examples:

```
          CALL DUMP
          CALL MATMPY (X(I,J),Y(J,K))
          CALL SEARCH (MTABLE,RALPH)
```

The mode of the subroutine name has bearing on the mode(s) of its results.

RETURN Statement

Form: RETURN

This statement returns control from an external subprogram to the calling program. Therefore, the last statement executed in a subprogram will be a RETURN statement, though it need not be physically the last statement within the program. Any number of RETURN statements may be used and they may occur at any point within the subprogram at which execution is to be terminated. A RETURN statement is necessary to return control whether a subprogram is explicitly referred to in a CALL statement, or implicitly referred to by a functional reference.

STOP statement

Form: STOP

Since, in a process control application, it is not permissible to stop the computer by program, this statement is interpreted as an instruction to MONITOR to stop operating this program. It therefore generates the calling sequence which would be associated with the statement:

* TURN PROGRAM OFF,0,1,0,0,0

This means that the next time the program is turned on, it will be entered at the beginning with all flip-flops reset except the PAIF, which is set.

STOP must appear by itself on a card.

END Statement

Form: END

The END statement is used to communicate to the compiler the logical end of a program or sub-program. It causes the compiler to finish the compilation by reserving memory locations for all variables named, etc.

END must appear by itself on a card.

* - Revised 5/65

INPUT-OUTPUT

Input-Output Statements

Input-output statements call for the transmission of information records between computer memory and various input or output units which are attached to the computer. In general, an input-output statement provides:

1. Specification of the operation required; whether input or output and the particular unit involved.

2. Reference to a data format specifying the conversions required between internal and external data forms. This reference is to the number of a FORMAT statement.

3. A list of the variables whose values are to be transmitted. The list order of the variables corresponds exactly to the order in which the information exits in the input medium or will exist in the output medium. For example, the statement

   ```
   PRINT 20, ALEPH, BETH, GIMEL
   ```

   specifies that the values of ALEPH, BETH and GIMEL are to be printed on line according to the format specified in the FORMAT statement numbered 20.

Input-Output Records

* All information appearing in external media is grouped into records. The amount of information contained in one record and the manner in which records are separated depends upon the medium. For punched cards, each card constitutes one record; for punched paper tape, a record consists of 80 or less frames, followed by a carriage return; in printing, a record is one line of 120 or less characters, etc. The actual amount of information contained in each record is specified by the FORMAT statement.

Each execution of an input or output statement initiates the transmission of a new data record. Therefore, the statement

```
READ 12, UN, DEUX, TROIS
```

is not necessarily equivalent to the statements,

```
READ 12, UN
READ 12, DEUX
READ 12, TROIS
```

* - Revised 1/65

since in the latter example at least three records are required, whereas in the former example one, two, three or even more records may be required depending on the FORMAT statement numbered 12.

If an input-output statement requests less than a full record of information, the unrequested portion of the record is lost and cannot be recovered by another input-output statement.

If an input-output statement requests more than one record of information, successive records are transmitted until the data request is satisfied.

Input-Output Lists

The list of variables in an input-output statement specifies the order of transmission of the variable values. During input, new values of listed variables may be used as subscripts or in control expressions for variables appearing later in the list. For example,

```
READ 4, J, A(J), B(J+1)
```

reads in a new value of J and uses it in the subscripts for A and B.

The transmission of array variables may be controlled by indexing similar to that used in a DO statement. A series of subscripted array variables and/or scalar variables followed by an index control may be enclosed in parentheses and will act as a single element of the input-output list. For example,

```
READ 7, (X(I),I = 1,4)
```

is equivalent to

```
READ 7, X(1), X(2), X(3), X(4);
```

and

```
PRINT 8, (X(I), Y(I), Q, I = 1,2)
```

is equivalent to

```
PRINT 8, X(1), Y(1), Q, X(2), Y(2), Q.
```

As in the DO statement, the initial, limit, and increment values of the index may be given as expressions; e.g.,

```
PRINT 2, K, (ARRAY(M), M = 1,J,K).
```

The indexing may also be compounded as in the following example,

```
PUNCH 3, ((TIME(K,L), K = 1,4), L = 1,8).
```

This statement outputs elements of the array TIME in the order

```
TIME (1, 1), TIME (2, 1)...TIME (4, 1), TIME (1, 2)...
TIME (4, 3), etc.
```

If an entire array is to be transmitted, the indexing values may be omitted and only the array identifier written. The array is transmitted as in the previous example, in order of increasing subscripts with the first subscript varying most rapidly. Therefore, the previous example could be replaced by

```
PUNCH 3, TIME
```

READ Statement

Forms: READ n, list
READ (expression) n, list

where n is a FORMAT statement number.

This statement causes information to be read from the input device identified by the value of the expression, converted to internal binary representation according to FORMAT statement n, and stored as values of the variables specified by the input-output list. If no expression is entered (first form), the device specified will be the one corresponding to the value 0.

PRINT Statement

Forms: PRINT n, list
PRINT (expression) n, list

where n is a FORMAT statement number.

This statement causes information contained in memory as values of the variable specified by the input-output list to be converted to BCD representations according to FORMAT statement n and printed on the output device specified by the value of the expression.

As with READ, absence of the expression and its enclosing parentheses will produce a reference to device number 0.

PUNCH Statement

Forms: PUNCH n, list
PUNCH (expression) n, list

where n is a FORMAT statement number.

This statement accomplishes exactly the same function as the PRINT statement and in an identical manner, except that the information to be output will be punched rather than printed.

Input-Output Tests

Prior to actual execution of I/O operations, it may be desirable to test the status of the "about to be employed" I/O device. For this purpose a series of IF statements of the form:

```
IF I/O type     n1,n2,n3
IF I/O type     (expression) n1,n2,n3
```

where n_1, n_2, n_3 are statement numbers, has been provided to accomplish the desired testing.

The six specific forms recognized are:

```
IF READ      n1,n2,n3
IF READ      (expression) n1,n2,n3
IF PRINT     n1,n2,n3
IF PRINT     (expression) n1,n2,n3
IF PUNCH     n1,n2,n3
IF PUNCH     (expression) n1,n2,n3
```

As with the corresponding I/O statement, the device number is indicated by the value of the expression with absence of the expression signifying device number 0.

n_1 is the statement number of the first statement in a routine to which control is to pass when the tested device is "busy", or whose "table" is full.

n_2 is the statement number of the beginning of a "faulty" or "bad" or "error condition" routine for the tested device.

n_3 is the statement number of the starting statement in an "out-of-service" routine.

If no such impediments are found to exist at the time of execution by the testing routines in the execution system, control passes to the next sequential statement following the IF I/O statement.

FORMAT Statement

All input or output of BCD information requires the use of a FORMAT statement specifying the external format of the data and the types of conversions to be performed. Any FORMAT statement may be used with any input-output medium, and since they are not executed, they may appear anyplace in the program.

Form: FORMAT $(S_1, S_2, \ldots, S_k)$

where S is a data field specification.

Numerical Fields

Four different types of numerical data conversion are available:

1. Type E

   Internal Form - Binary floating-point
   External Form - Decimal floating-point

2. Type F

   Internal Form - Binary floating-point
   External Form - Decimal fixed-point

3. Type I

   Internal Form - Binary integer
   External Form - Decimal integer

4. Type O

   Internal Form - Binary integer
   External Form - Octal integer

The four types of conversions are specified by the following general forms:

1. Ew.d
2. Fw.d
3. Iw
4. Ow

The letter E, F, I, or O designates the conversion type; w is an
* integer specifying the data field width with a maximum of 63. And
d is an integer specifying the number of decimal places to the right
of the decimal point. The value of d must be $\leq$15. For E type output
conversion, w should exceed d by at least seven.
For example, the statement:

```
          FORMAT (I3, F10.4, E15.5, 08)
```

could be used to print the line

```
        37      -14.2639    0.46972E 3    37777
```

on the output listing.

Note that plus signs are not printed.

Note also that in type F conversion there is a decimal point but no exponent, whereas type E conversion has an exponent. On output, the exponent always has the form shown, i.e., an "E" followed by a signed two digit integer. On input, however, the "E" or the sign or the entire exponent may be omitted. The following are all valid E15.6 data fields for input:

```
 .327409+2
 .964718E4
 .003276-4
305976.
```

The field width w includes all of the characters (decimal point, signs, blanks, etc.) which comprise the number. If a number is too long for its specified data field, the excess characters are lost. Since numbers are right justified within their fields the digits lost are those with the most significance. The spacing of output data for legibility must be considered when specifying a data field width.

During input, the appearance of a decimal point "." in an "E" or "F" data field specification overrides the "d" specification of the field. In the absence of an explicit decimal point, the point is positioned d places from the right of the field not counting the exponent, if one is present. For example, a number appearing externally as 314159E-1 with a data field specification of E12.5 will be interpreted as 3.14159E-1.

In addition, a scale factor, indicated by a signed or unsigned decimal integer followed by the letter, "P", may be imployed with E and F fields. If used with an "F" field, it will achieve multiplication of floating quantities by powers of 10 before conversion on output or after conversion on input. If used with a "E" field, it will be ignored on input and will cause P whole-number digits to precede the decimal point and the exponent to be decreased by P. The P-factor must not be negative for use with E-fields. The maximum absolute value must be ≤ 9. A P-factor holds for all succeeding E and F fields in the FORMAT statement. To end its influence, a P-factor of zero must be stated.

Examples:

```
6PE9.3
2P4F7.2
```

Suppose that the above examples are employed as follows:

```
      PRINT 7, X, (PRCNT(I), I = 1,4)
    7 FORMAT  (PE10.3, 2P4F7.2)
```

Assume further that the values of the variables concerned are:

X	6,494,650
PRCNT (1)	.012345
PRCNT (2)	.596938
PRCNT (3)	.856062
PRCNT (4)	1.56032

The resulting printed line would appear as:

```
0.649E 7      1.23      59.69      85.61      156.03
```

Alphanumeric Fields

Alphanumeric data can be transmitted in much the same manner as numeric data through use of the form Aw where A is a control character and w is the number of characters in the field and has a maximum of 63. The alphanumeric characters are transmitted as the value of a variable in an input-output list. The variable may be of either mode. For example,

```
        READ 17, V
   17   FORMAT (A3)
```

will cause three characters to be read and placed in memory as the value of the variable V.

If the format specification had indicated a field width greater than 4, the additional characters will be grouped in fours and placed in V + 1, V + 2, --- etc., until the entire field width is satisfied. If the last such group contains less than 4 characters, they will be left-justified.

Alphanumeric Format Fields

An alphanumeric format field may be specified by preceding the alphanumeric string with the specification nH, where n is the number of characters in the string including blanks and has a maximum value of 120. For instance, the following sequence:

```
       PRINT 3
   3   FORMAT (9H NOT DONE)
```

will print the words NOT DONE on line. The n characters of the data field are not transmitted as the value of a variable, but are stored in the memory space allotted for the FORMAT statement itself. The n characters may be replaced by n other characters by means of an input statement which references the FORMAT statement. The value of n must be≤ 120.

An input-output list is not required for the transmission of this type of field. During input, n characters are extracted from the input record and used to replace the n characters within the specification. During output the n characters specified or the n characters which have replaced them become part of the output record. For example, the sequence –

```
      READ 3
    3 FORMAT (9H NOT DONE)
      PRINT 3
```

will print the words ALL DONE on line provided ALL DONE appears in positions 1-9 of the input record being processed by the READ 3 statement.

Alphanumeric Notes

Alphanumeric information can also be placed in an output line by use of a special feature, called a "note". For instance, the words "NOT DONE" as shown in the previous example could be inserted by:

```
*     PRINT 3
    3 FORMAT ($ NOT DONE$)
```

The number of characters between the two "$" symbols would be counted and the same information would be generated as in the previous example. Note that the "$" symbol cannot be a member of a note, since it signals termination of the note.

Blank or Skip Fields

Blanks may be introduced into an output record, or characters skipped on an input record by use of the specification nX. The control character is X and n is the number of blanks or characters skipped with a maximum value of 31. For example, the statement:

```
*     FORMAT (5H JOB I3, 10X, 4HDONE)
```

may be used to output the line-

```
      JOB 397      DONE
```

with 10 blanks separating the two quantities.

Mixed Fields

An alphanumeric format field may be placed among the other fields of the FORMAT statement to enhance the readability of output listings. For example:

```
      FORMAT (8H FORCE = F9.4, 5H LBS.)
```

may be used to print-

```
      FORCE = 297.6374 LBS
```

Note that the separating comma may be omitted after an alphanumeric format field.

Repetition of Field Specifications

Repetition of a field specification may be specified by preceding the
* control character E, F, I, A, or O by an unsigned integer less than or equal to seven giving the number of repetitions desired. The specified number of repetitions must not be zero. For example:

```
FORMAT (3I2, 2F12.6, 2O4)
```

is equivalent to

```
FORMAT (I2, I2, I2, F12.6, F12.6, O4, O4)
```

Repetition of Groups

A group of field specifications may be repeated by enclosing the group within parentheses and preceding the whole group with the number of desired repetitions. For example,

```
FORMAT (3I2, 2(F12.4, 3E8.2))
```

is equivalent to

```
FORMAT (3I2, F12.4, 3E8.2, F12.4, 3E8.2).
```

Alphanumeric fields may also be repeated in this manner,

```
     PRINT 7                                PRINT 8
*  7 FORMAT (3(5H BANG))     or        8    FORMAT (3($ BANG$))
```

will cause the following on line output,

```
BANG BANG BANG
```

Five levels of repeated groups are allowed.

Multiple Record Formats

In the case where a group of successive input-output records have different field specifications, a slash, "/", is used to separate those field specifications. For example, the statement

```
FORMAT (4O8/I3, 3F12.8/12A3)
```

is equivalent to

```
FORMAT (4O8)
```

for the first record,

```
FORMAT (I3, 3F12.8)
```

for the second record, and

```
FORMAT (12A3)
```

for the third record.

* - Revised 1/65

The comma separating data field specifications may be omitted when a slash is used. Blank records may be written on output or records skipped on input by using consecutive slashes. On printed output a slash always causes a skip to a new line after completing the record. Both the slash and the closing parenthesis at the end of format indicate the termination of a record. If an input-output statement list indicates that data transmission is to continue after the closing parenthesis of a format statement is reached, the format is repeated from the last left parenthesis.

The statement

```
FORMAT (2E12.6, 2(3F7.4, 208, I7))
```

is equivalent to

```
FORMAT (2E12.6, 3F7.4, 208, I7, 3F7.4, 208, I7)
```

for the first record, and

```
FORMAT (3F7.4, 208, I7)
```

for all succeeding records.

The total of all field widths specified for a record is the length of that record. If the record length specified is greater than the maximum allowable on a particular input-output device, the excess characters are lost.

Drum Transfer Statements

Two statements, READ DRUM (drum to core) and WRITE DRUM (core to drum), provide a method for transferring blocks of information (programs and data) between core memory, where execution takes place, and drum memory, where large amounts of bulk storage are available. Each statement calls for the transfer of a single continguous block and, therefore, no input-output list is employed.

Forms: READ DRUM (symbolic$_1$), (symbolic$_2$), (symbolic$_3$)
WRITE DRUM (symbolic$_1$), (symbolic$_2$), (symbolic$_3$)

where

Symbolic$_1$ is the initial drum address in PAL symbolics.
Symbolic$_2$ is the number of words to be transferred in PAL symbolics.
Symbolic$_3$ is the initial core address in PAL symbolics.

COMMUNICATION WITH REAL-TIME MONITOR

In order that programs may undergo dynamic scheduling and relocation as components of a real-time complex, certain statements which communicate initiation, termination, delay and segmentation of the components to the overall control or MONITOR routine become necessary. Description of a series of such statements provided within GE/PAC FORTRAN follows.

TURN PROGRAM ON Statement

* Form: TURN PRØGRAM n ØN, expression

where n is an integer constant indicating the "program number" by which the program to be "turned on" is known to the MONITOR. The value of the expression specifies the time of next execution of the "turn on" program with a 0 value indicating immediate execution.

Example:

* TURN PROGRAM 21 ON, NOW + 1000

TURN PROGRAM OFF Statement

Form: TURN PROGRAM OFF, m_1, m_2, m_3, Return, m_4

This statement turns the current program off, i.e., places it in the inactive state

Example:

* TURN PROGRAM OFF, 0, 1, 0, NXENTY, 0

* m_1, m_2, m_3, m_4, and Return are PAL symbolic fields interpreted as follows:

m_1:	1 = set ØVRF
m_2:	1 = set PAIF
m_3:	1 = set TSTF
m_4:	1 = set TMFF to TRAP
Return:	Symbolic location for ECP entry at time of next "turn on"

(TMFF-Memory Fence Flip-Flop)

DELAY PROGRAM Statement

Form: DELAY PROGRAM, m_1, m_2

This statement places the current program in an inactive state for a specified length of time.

m_1 and m_2 are PAL symbolic fields indicating the following:

m_1:	1 = program's area of occupancy is now available. 0 = area remains unavailable
m_2:	indicates length of delay

SEGMENT Statement

* Form: SEGMENT n, m_1, m_2, m_3, NXENTY, m_4

where n is the program number of the following "segment" to be assigned and recognized by MONITOR (as in TURN PROGRAM ON).

Aside from its declarative properties (see section of this manual titled DECLARATION), SEGMENT n is equivalent to the following statement sequence.

TURN PROGRAM n ON, 0
TURN PROGRAM OFF, m_1, m_2, m_3, NXENTY, m_4
END

m_1, m_2, m_3, m_4, and NXENTY are as defined in TURN PROGRAM OFF.

* - Revised 5/65

DECLARATIONS

A declaration describes certain properties of a FORTRAN program, as opposed to the imperative assignment, control or input-output statements. Several FORTRAN statements are reserved for the purpose of supplying the system with declarative information. These statements are primarily concerned with the interpretation of identifiers occurring in the source program and with memory allocation in the object program.

Classification of Identifiers

Each identifier in a source program is classified in accordance with the FORTRAN element it identifies. Five main classifications are recognized.

1. Scalar identifiers
2. Array identifiers
3. Subprogram identifiers
4. Dummy identifiers
5. Single bit arrays identifiers *

The classification is made according to the context in which the identifier makes its first physical appearance in the source program. This first appearance amounts to a declaration, explicit or implicit, of the proper interpretation of the identifier throughout the program.

Mode Declarations

In addition to being classified, each identifier appearing in a FORTRAN program is of mode integer or real. The statements INTEGER and REAL are used to specify identifer modes explicitly. An identifier may appear in only one of these statements and this appearance must precede the use of the identifier in any non-declarative statement.

Identifiers whose mode is not explicitly declared are assigned modes implicitly according to the following convention.

1. Identifiers beginning with I, J, K, L, M or N are assigned integer mode.

2. Identifiers not included in the above classification are assigned real mode.

* - Revised 1/65

INTEGER Statement

Form: INTEGER identifier, identifier,..., identifier

This statement declares the listed identifiers to be integer.

Example:

```
INTEGER PHI, KAPPA, SIGMA
```

Notice that KAPPA need not appear in this statement since it would be declared integer mode implicitly.

REAL Statement

Form: REAL identifier, identifier,..., identifier

This statement declares the listed identifiers to be real mode.

Example:

```
REAL FORCE, MASS, LOG
```

Notice that FORCE need not appear in this statement since it would be declared real mode implicitly.

DIMENSION Statement

The DIMENSION statement declares an identifier to be an array identifier, and also specifies the number and limits of the array subscripts. Any number of arrays may be declared in a single DIMENSION statement.

Information provided by a DIMENSION statement is required for the allocation of storage for arrays. Each array variable appearing in a program must be previously declared as an element of an array in a DIMENSION statement. Each array variable must have the same number of subscripts as were declared for the array, and the value of each subscript must lie within the limits specified by the DIMENSION statement.

Form: DIMENSION $S_1, S_2, \ldots, S_k$

when S is an array specification.

Each array specification gives the array name and the minimum and maximum values that each of its subscripts may assume. The minimum and maximum values for each subscript must be given as signed or unsigned integer constants with the maximum value greater than the minimum value. An array identifier may have any number of subscripts.

Two forms for specification of the maximum and minimum subscript values are recognized:

identifier$(\text{min}_1/\text{max}_1, \ldots)$
identifier$(\text{max}_1, \text{max}_2, \ldots)$.

In the latter form, a minimum value of one is implied, i.e., the latter specification is equivalent to

identifier $(1/\text{max}_1,\ 1/\text{max}_2, \ldots)$.

The two forms are syntactically independent and may, therefore, both occur as different subscript limit specifications in the array specification.

Example:

```
DIMENSION X(10), Y(-1/5,20), Z(-3/-1,2,0/3,5,2)
```

Subprogram Definition Statements

The two types of subprograms which may be called, or referred to by a FORTRAN program are classified as external or internal subprograms.

Internal subprograms are defined within the program that calls them, and are defined within a single statment known as an arithmetic function definition statement. Internal subprograms may be used only within the program containing their definition. External subprograms are defined separately from, i.e., externally to the program calling them. They are complete autonomous FORTRAN programs within themselves, and as such are compiled independently. There are two types of external FORTRAN subprograms which may be declared: FUNCTION subprograms and SUBROUTINE subprograms, both of which are described below. Any subprogram, whether internal or external may call other subprograms; however, recursion is not allowed. All subprograms constitute closed subroutines; i.e., they appear only once in the object program regardless of the number of times they are called.

Dummy Identifiers

A subprogram definition statement declares those identifiers appearing as arguments of the subprogram to be dummies. They are used as ordinary identifiers within the subprogram definition and indicate the mode and use of the arguments. Dummy identifiers are replaced by actual arguments when the subprogram is executed.

Arithemetic Function Definition Statement

Form: identifier(identifier, identifier,..., identifier) = expression

This statement completely defines an internal subprogram. The first identifier is the name of the subprogram being defined.

Arithmetic function subprograms are functions; i.e , they are single valued and must have at least one argument. The mode of the function is determined by the mode of the function identifier.

The identifiers enclosed in parentheses represent the arguments of the function. These so-called dummy identifiers have meaning and must be unique only within the defining statement. They may in fact be identical to identifiers appearing elsewhere in the program. They must agree in order, number, and mode with the actual arguments given at execution time.

The use of an argument within the definition statement is specified by the use of its dummy identifier. Expressions are the only permissible arguments for internally defined functions, therefore, dummy identifiers may appear only as scalar variables in the defining expression. The use of an array identifier or a subprogram identifier is not allowed.

Identiiiers appearing in the definition statements which do not represent arguments are treated as ordinary variables. In addition, external functions or other previously defined internal functions may appear in the definition statement.

Examples:

```
F(X,Y) = (X+Y)*(X-Y)
SINH(ZETA) = (EXP(ZETA/B)-EXP(-ZETA/B))/2
Q(X,Y,Z) = F(X,Y)/SINH(Z)
```

In the second example above, ZETA is a dummy identifier and B is an ordinary identifier. At execution time the function will be evaluated using the current value of B. The third example is allowable if the first and second definitions precede it in the program.

If the accompanying source program contained the statement

```
Y = A*SINH(ALPHA),
```

the arithmetic function defining SINH would be evaluated using the current value of ALPHA as its argument. The result would be multiplied by A and the product assigned as the value of the variable Y.

All internal subprogram definitions of any FORTRAN II program must precede the first non-declarative statement of the program.

FUNCTION Subprograms

Like the Arithmetic Function Definition Statement, the FUNCTION subprogram is also a function, is single valued, and is referenced as a basic element in an expression. FUNCTION subprograms may be used when more than one FORTRAN statement is needed to define the functional relationship. In order to logically separate a FUNCTION subprogram from the calling programs, the subprogram always begins with a FUNCTION declaration and returns control to the main program via the RETURN statement.

FUNCTION Statement

Forms:

```
FUNCTION identifier (identifier, identifier, ..., identifier)
REAL FUNCTION identifier (identifier, identifier,..., identifier)
INTEGER FUNCTION identifier (identifier, identifier,..., identifier)
```

This statement declares the following program to be a function subprogram. The first identifier is the name of the subprogram being defined. This identifier must appear as a scalar variable and is assigned the value of the function resulting from execution of the subprogram. The function mode is declared implicitly by the initial letter of the function name, or may be declared explicitly by using the second or third forms of the FUNCTION statement.

Those identifiers appearing in the list enclosed by parentheses are dummy identifiers which represent the function arguments, and must agree in number, order, and mode with the actual arguments given at execution time. These arguments may be array names as well as expressions. Therefore, the dummy identifiers may appear as array identifiers or scalar identifiers. A FUNCTION statement must have at least one argument. Dummy identifiers representing array names must appear in a DIMENSION statement in the FUNCTION subprogram as well as the calling program, and must agree with the specification of the arrays in the calling program.

Examples:

```
FUNCTION SEARCH (LIST,ALPHA)
REAL FUNCTION INDEX (A1,B2,C3)
INTEGER FUNCTION DELTA (ARG1,ARG2)
```

SUBROUTINE Subprograms

The SUBROUTINE subprogram is not a function in that it may be multi-valued and is referred to only by the CALL statement. The SUBROUTINE subprogram begins with the SUBROUTINE declaration and returns control to the calling program via the RETURN statement.

SUBROUTINE Statement

Form: SUBROUTINE identifier (identifier,identifier,...,identifier)

This statement declares the following program to be a SUBROUTINE subprogram. The first identifier is the name of the subroutine. Those identifiers appearing in the list enclosed in parentheses are dummy identifiers which represent the subprogram arguments. As in the FUNCTION statement, these arguments may be scalar or array identifiers.

Dummy identifiers representing array names must be declared in a DIMENSION statement within the subprogram and must agree with the specifications of corresponding arrays specified in the program containing the CALL statement.

Contrary to the FUNCTION and Arithmetic Function Definition Statements, the result of a SUBROUTINE subprogram is not necessarily a single value and the value of the name of the subprogram may or may not (depending on the subprogram) be meaningful. Instead specific results are normally returned as values of variables in the argument list.

A SUBROUTINE subprogram does not necessarily require arguments; in the absence of an argument list no results can be returned via the arguments.

Examples:

```
SUBROUTINE BURST
SUBROUTINE FACTOR (COEF1,COEF2,ROOT)
```

Implicit Declaration of Identifiers

Identifiers appearing in declaration statements such as DIMENSION, FUNCTION, SUBROUTINE, etc., are explicitly classified by their appearance in that statement. If the first appearance of an identifier is in an imperative statement rather than a declarative statement, the identifier is classified implicitly according to its context.

For example, in

```
DIMENSION ALPHA (5,10)
```

the identifier ALPHA is explicitly declared to be a two-dimensional array. Similarly

```
FUNCTION ALPHA (X,Y)
```

explicitly declares ALPHA to be a single-valued function with arguments X and Y. Conversely,

```
ALPHA (X,Y) = 2*X+Y
```

implicitly declares the identifier ALPHA to be an internal subprogram because it appears in an arithmetic function definition statement. Note also in the three previous examples that in the absence of declarative statements to the contrary, ALPHA is implicitly declared to be the identifier of a real quantity. As a further example note that in the statement

```
SUBROUTINE ALPHA (X,Y)
```

ALPHA is explicitly declared to be the name of a subprogram with arguments represented by dummy identifiers X and Y.

* However, if the expression

```
ALGFNC + ALPHA (A,B)+Z/3
```

has not been preceded in a program by any of the four previously mentioned declaration statements ALPHA will be implicitly declared to be an external subprogram whose presence will be mandatory at runtime for execution of the object program.

Memory Allocation

Memory allocation statements are used to supply the system with supplemental information regarding the storage of variables and arrays; and when the program is to be subordinate to a real-time system, to supply memory assignment of the program in bulk memory and specify coincident cross-references as may be necessary.

DEFINE Statement

Form: DEFINE identifier$_1$(symbolic), identifier$_2$(symbolic),...

The DEFINE statement provides a method of incorporating uniform reference to variables which are to be part of a "permanent memory space" in an overall system.

* The identifiers occurring in the list may be array or scalar identifiers, or subprogram names. The symbolics enclosed within the parenthesis may be PAL symbolic expressions to which the preceding identifier will be equated by the compiler.

Identifiers appearing in DEFINE statements are assumed to have space allocated by the overall system and are not, therefore, assigned space by the memory allocation section of the compiler.

* The temporary storage area created by the compiler is named $TEMP. It can be assigned to permanent memory by naming it in the DEFINE Statement.

EXAMPLE:

```
     DEFINE X(VO1T25), SQRTF(/3000), $TEMP(/4020), COMMON(/3127)
```

COMMON Statement

Form: COMMON identifier, identifier,...,identifier

Within the space allocated by the overall system as "permanent memory", a section is set aside with the symbolic label of the first locations as "Common". For this purpose, the symbol, COMMON, is reserved in both the FORTRAN and PAL languages and is used solely for reference to this specially named section of memory.

The identifiers in the COMMON statement may be either array or scalar names provided that the array names also appear in a DIMENSION statement in the same program.

During allocation, each variable to be allocated in COMMON is assigned the next block (one word for scalars, array size for arrays) relative to the previous allocation with the first such allocation being a simple equation to the symbol, "COMMON".

EXAMPLE:

```
     COMMON    SCALAR,J, ARRAY, A
```

In the above example, suppose that SCALAR and J are scalars and that ARRAY and A are dimensioned elsewhere as ARRAY (-1/3,5) and A(100). The resultant equations to COMMON would be:

```
*                     SCALAR      EQL     COMMON
                      J           EQL     COMMON + 1
                      ARRAY       EQL     COMMON + 2
                      A           EQL     COMMON + 27
```

In view of the implicit property of the symbol, COMMON, as a member of the same space as that referred to by the DEFINE statement, the above equations to COMMON can be seen as equivalent to those which would be produced by

```
DEFINE SCALAR (COMMON), J(COMMON + 1),
ARRAY (COMMON + 2), A(COMMON + 27)
```

EQUIVALENCE Statement

The EQUIVALENCE statement allows more than one identifier to represent the same quantity.

Form: EQUIVALENCE $(R_1,R_2,...,R_n),(R_k,R_{k+1},...,R_m)$

where R denotes a location reference.

The location references of an EQUIVALENCE statement may be simple scalar or array identifiers, or may be identifiers appended by an integer constant enclosed in parentheses. All location references enclosed within the same parenthetical expression share the same storage location. Such a group is known as an equivalence set. For example,

```
EQUIVALENCE (BLACK,WHITE)
```

states that the identifiers BLACK and WHITE refer to the same storage location.

To refer to a specific location in an array, that location must be appended to an array identifier as an integer constant. For example, if A is a scalar variable and B is an array the statement

```
EQUIVALENCE (A,B(5))
```

specifies that A and the fifth location of the array B share the same storage location.

To refer to a specific quantity in a multiply dimensioned array the location of that quantity must first be calculated. As an example, consider the three dimensional array specified by

DIMENSION CUBE (L_1/U_1, L_2/U_2, L_3/U_3)

where L_i and U_i denote the lower and upper limits of the i th subscript. To calculate the location of

CUBE (K_1,K_2,K_3)

use the formula

$$\text{LOCATION} = (U_2-L_2+1)(U_1-L_1+1)(K_3-L_3)+(U_1-L_1+1)(K_2-L_2)+K_1-L_1$$

Therefore the statements

```
DIMENSION SPACE (12), VOLUME (2, 2, 2)
EQUIVALENCE (SPACE(3), VOLUME (7))
```

specifies that the quantities SPACE (3) and VOLUME (1,2,2) will share the same storage location. Notice that only the relative location of the quantities within the array matters, since the entire array is adjusted to satisfy the EQUIVALENCE statement. In the example above, the statement

```
EQUIVALENCE (SPACE(1), VOLUME (5))
```

would have had the same effect as

```
EQUIVALENCE (SPACE(3), VOLUME (7))
```

Where the location of a variable is known relative to a second variable, this location may be specified by appending an integer constant to the identifier of the second variable. The integer to be used is determined by considering the sequence of quantities as an unidimensional array. For example, if we have in storage at

```
LOCATION

      L1:  ALEPH
      L2:  BETH
      L3:  GIMEL
      L4:  DALETH
```

the statement

```
EQUIVALENCE (GAMMA,ALEPH(3))
```

will specify that GAMMA and GIMEL refer to the same storage location.

Note that the property of equivalence is transitive; the statement

EQUIVALENCE (X,Y),(Y,Z)

has the same effect as

EQUIVALENCE (X,Y,Z)

Joint DEFINE, COMMON and EQUIVALENCE Rules

In the event that identifiers appear mutually in DEFINE, COMMON and/or EQUIVALENCE statements, the DEFINE statement takes precedence followed by the COMMON statement.

All equivalences to DEFINE and COMMON variables are equated off, i.e., removed from the remaining equivalence sets and entered in the object code as equations to symbols occurring in DEFINE statements or to the symbol, COMMON.

BEGIN PROGRAM AT Statement

Form: BEGIN PROGRAM AT n

where n is a PAL symbolic specifying the location in bulk memory into which the ensuing program is to be loaded.

SEGMENT Statement

* Form: SEGMENT n, m_1, m_2, m_3, NXENTY, m_4

where n is the MONITOR recognized program number of the following segment. This statement is used to break a large program into a number of smaller segments.

In addition to segmentation by turning "off" and turning the next part (segment) "on", the SEGMENT statement causes the execution (at compile time) of a FORTRAN "END". This execution allocates storage, published errors and removes all known names and labels from the tables before proceeding to the next job. Therefore, the only information carried forward to the next "segment" will be the relative location in bulk memory. Care must be taken to assure that variables which are common to segments are declared in each segment by "COMMON" or "DEFINE" statements.

* - Revised 1/65

DIAGNOSTICS

Statement Diagnostics

Statements which violate the syntax or semantic rules of the FORTRAN language are detected during compilation and are discarded. An error message is printed on the line printer and compilation proceeds as if the erroneous statement had never been encountered. The error message consists of the statement in the form in which it entered the computer; then one character of the statement is indicated by a dollar sign, "$", printed beneath it. For example, in

```
13 FUNC = 2*X+Y*+A
                $
```

The character "+" is marked as an error. In the case of syntax errors, the marked character itself is unacceptable as in the above example. In the case of a semantic error, an identifier or other construct is being improperly used. and the dollar sign indicates the last character of the construction. For instance, in the statement

```
ASUM = PART**3+4
          $
```

the dollar sign would indicate that the identifier PART is being used incorrectly; e.g., PART may be the name of a subprogram.

The compiler will try all possible legal interpretations of a statement before finally discarding it. The dollar sign position indicates the greatest amount of correct information, starting from the left, that was found under any assumption about the statement.

A comment indicating the reason for the error is then printed at the left margin after the marked line. These comments are as follows:

SYNTAX

This comment usually occurs because of erroneous punctuation or illegally constructed arithmetic expressions.

NUMBER

A constant, label, or input-output symbolic unit number is too large or is incorrectly constructed.

ID DECLARATION

The identifier indicated is being used in a manner contradictory to a previous declaration.

SUBSCRIPTS

There are too many subscripts, an expression describing a subscript is incorrectly constructed, or the number of subscripts used in an array variable does not agree with the number declared in the DIMENSION statement.

ALLOCATION

1. A negative or zero array size was specified in a DIMENSION statement.

2. The rules of COMMON or EQUIVALENCE have been violated. In either statement the identifier causing the violation is marked.

PROGRAM OVERFLOW

The working core storage available to the compiler has been exceeded.

ARGUMENTS

A FORTRAN reserved function has been employed with the wrong form and/or number of arguments.

Program Diagnostics

Comments concerning labeling and allocation errors are listed at the end of compilation. The label errors comment is followed by a list of statement labels, and the allocation errors comment by a list of the offending identifiers.

LABEL ERRORS

1. The following statement numbers were used in control instructions within the program but referred to an unnumbered statement.

2. Two or more statements have the same label.

3. The statement closing a DO loop was never reached.

4. The final statement in the range of a DO loop was a transfer statement.

5. The DO loop was illegally nested.

ALLOCATION ERRORS

The identifiers that follow this error message violated DEFINE, COMMON and/or EQUIVALENCE rules, such as the use of a function or subroutine name as the name of a variable in either a COMMON or EQUIVALENCE statement.

APPEDNIX A

SUMMARY OF GE/PAC FORTRAN STATEMENTS

* - Revised 5/65

GE/PAC 4000

DOUBLE-WORD FORTRAN REFERENCE MANUAL

Library-Control No. YPG30M

This manual published by:
PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER SECTION
P. O. BOX 2918
PHOENIX, ARIZONA - 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

CONTENTS

FOREWORD

This is a reference manual for the GE/PAC Double-Word FORTRAN language. A previous familiarity, on the part of the reader, with basic FORTRAN concepts is assumed. No attempt is made to present the essentials of FORTRAN in a form usable for a fundamentals course in FORTRAN language usage. The primary purpose of this manual is to describe the specific statements and capabilities of the GE/PAC Double-Word FORTRAN language.

Additions have been made to the FORTRAN II language to increase programming flexibility. In addition, they provide the programmer with special statements which enable him to produce a total real-time system.

Existing FORTRAN II programs can be compiled in GE/PAC FORTRAN, provided that the specific restrictions noted herein are not violated. Programs written for GE/PAC Single-Word FORTRAN can be compiled in Double-Word FORTRAN provided that the restrictions noted in BIT statements and BOOLEAN statements are followed.

1 INTRODUCTION TO DOUBLE-WORD FORTRAN

A GE/PAC FORTRAN program is a sequence of statements, each of whose characteristics is described in this manual. These statements may be classified according to the following general categories.

1. COMPUTATION statements which comprise the working body of the program; e.g., numerical calculation and bit manipulation.

2. CONTROL statements which specify the flow of control during execution.

3. DECLARATION statements which supply information about the program.

4. INPUT-OUTPUT statements which provide communication with the system peripherals.

5. PAL LANGUAGE statements which may be included in the program in order to "tailor" certain critical areas.

PROGRAM PREPARATION

The statements representing a Double-Word FORTRAN program are first entered on a coding form similar to that of Figure 1. The lines of the coding form are divided into sixty-nine columns, each of which may contain one character. Each statement is written on a separate line; if more than one line is required, as many as four additional lines may be used as a "continuation" of the statement.

Columns 1 through 5 may be used for a statement number. Such numbers may be used as labels to which other statements in the program refer.

A non-zero, non-blank character in column 6 indicates that the line is a continuation line.

The body of the statement itself is entered in columns 7 through 69 of the initial and any necessary continuation lines.

Column 70 must contain a "7" in all lines of all statements of categories 1 through 4 above. Lines which contain a "6" in column 70 are of category 5. All category 5 statements are transmitted unchanged into the object program.

A "C" in column 1 indicates that the statement is a comment. Like category 5 statements, comment statements are passed directly into the object program at the point of encounter. The text of the comment may be entered in columns 2 through 69 of the initial line and in columns 7 through 69 of continuation lines.

A "B" in column 1 indicates the statement is of type "Boolean" and that certain numerical calculations contained in the statement are to be performed with logical rather than arithmetic operations.

Except for columns 1 and 6 and certain alphanumeric fields, blanks are ignored and may be used freely to increase legibility.

The first line of any program, irrespective of content, is assumed to be the title line and contains the text (columns 1 through 39) which will head pages of the output listing and the program identification number (columns 71 through 75) which will be transmitted to all generated object lines.

The last line of the program must be an END statement. An END statement may contain no other characters in columns 7 through 69 except "END".

SAMPLE PROGRAM

It is desired to fit the best straight line approximation to a population of data by the method of least squares. The following program indicates how this would be done using FORTRAN. The general formula for the approximation is of the form $y = a+bx$. The solve for a and b by the method of least squares we evaluate the formulas:

$$a = \frac{(\Sigma y) \quad (\Sigma x^2) \quad - \quad (\Sigma x) \quad (\Sigma xy)}{(n) \quad (\Sigma x^2) \quad - \quad (\Sigma x)^2}$$

$$b = \frac{(n) \quad (\Sigma xy) \quad - \quad (\Sigma x) \quad (\Sigma y)}{(n) \quad (\Sigma x^2) \quad - \quad (\Sigma x)^2}$$

Line 1 is the TITLE CARD. It is also a FORTRAN comment card. The Sample Program is illustrated in Figure 1 on the following page.

The dimension statement of line 2 declares X and Y to be one dimensional arrays each containing 20 numbers. This declaration sets aside two groups of 20 consecutive storage spaces for X and Y and allows them to appear as subscripted variables within the program.

Line 3 is an input statement that reads a record of previously prepared data from tape under control of FORMAT statement 10 and places it in the consecutive locations of the X and Y arrays

GENERAL ELECTRIC
PROCESS COMPUTER SECTION
PHOENIX, ARIZONA

PROCESS LANGUAGE STATEMENT CODING FORM

Project Name

Program Name

Page of Date

Programmer

Type Code: 0–deletion, 2–PAP, 3–NOAP, 4–COOL

LOCATION* | ○ TYPE | STATEMENT | BRANCH CONTROL FIELD: C | =0 | ≠0 | + | – | Any Case | Type | Proj. # | Prog. # | Sequence #

```
1 2 3 4 5 6 7 8 9 10 11 ... 43 44 45-49 50-54 55-59 60-64 65-69 70 71 72 73-75 76-80
C     EXAMPLE ØF LEAST SQUARES                                       726
      DIMENSIØN X(20),Y(20)                                          726
      READ 10,X,Y                                                    726
      SUMX=0.                                                        726
      SUMY=0.                                                        726
      SUMSQX=0.                                                      726
      SUMXY=0.                                                       726
      DØ 5 I=1,20                                                    726
      SUMX=SUMX+X(I)                                                 726
      SUMY=SUMY+Y(I)                                                 726
      SUMSQX=SUMSQX=X(I)**2                                          726
5     SUMXY=SUMXY+X(I)*Y(I)                                          726
      A=(SUMY*SUMSQX-SUMX*SUMXY)/(20.*SUMSQX-SUMX**2)                726
      B=(20.*SUMXY=SUMX*SUMY)/(20.*SUMSQX-SUMX**2)                   726
      PRINT 10,A,B                                                   726
10    FØRMAT (2E14.6)                                                726
      END                                                            726
```

PC-12 (3/63) *Restricted to five characters for COOL

Shaded areas indicate PAP format restrictions

FIGURE 1

Calculation actually begins with line 4 in which a variable called SUM is set equal to zero. Similar operations are performed on lines 5, 6, and 7.

Line 8 sets up a series of repetitive calculations in what is known as a DO loop. It causes the calculations up to and including statement 5 on line 12 to be performed 20 times. The subscripting variable I is set to 1 for the first execution of the following four instructions and is increased by 1 for each subsequent execution. In this manner it is possible to refer to successive values of X and Y for calculation purposes.

In line 9 a value of X is added to SUM and the result is assigned to SUM. A similar operation is done for Y on line 10.

In line 11 a value of X is squared, added to SUMSQX, and assigned to SUMSQX.

In line 12 a value of X is multiplied by a value of Y, added to SUMXY, and assigned to SUMXY.

After the previous four calculations have performed 20 times, control passes from the DO loop to the statement of line 13 in which A is calculated. Then B is calculated on line 14.

Line 15 is an output statement which causes the values of A and B to be printed on-line according to FORMAT statement number 10 (line 16). This FORMAT statement indicates two answers are to be printed on the same line, each answer occupies 14 spaces (including blanks) and there are 6 digits after the decimal point.

Line 17 is a control card indicating the END of the example.

2 COMPUTATION

BASIC ELEMENTS

The basic elements used as operands in Double-Word FORTRAN computations are constants, variables, and functional references, all of which represent numerical quantities. In the FORTRAN language, these elements are represented by symbols composed of character strings.

QUANTITIES

Two types of numerical quantities are recognized by FORTRAN, arithmetic and logical. Within the designation arithmetic, two modes are recognized, real and integer. Integer quantities represent integers within the range -8,388,608 (-2^{23}) through 8,388,607 ($2^{23}-1$) inclusive, and occupy one machine word. Real quantities represent real numbers and are represented in the computer by floating-point configurations comprised of a 9 bit exponent and a 37+sign bit mantissa and occupy two machine words. The approximate range of floating quantities is -10^{76} through -10^{-77}, 0 and 10^{-77} through 10^{76}.

Within the designation logical, only octal digits (0 through 7 inclusive) are recognized. An octal quantity occupies one machine word and has a maximum value of 77777777.

CONSTANTS

Constants are numbers of arithmetic or logical type which appear in the source program in explicit form.

Arithmetic integer constants are written as a string of decimal digits.

Examples:

```
0
1
1964
```

Logical integer constants are written as a string of octal digits.

Examples:

```
0
252525
77777777
```

In either type, the integer represented must lie within the range specified above for integer quantities.

Real constants are written as a string of decimal digits which includes a decimal point.

Examples:

.0
1.
2.71828

Real constants may be given a scale factor by appending an "E" followed by an integer constant, which indicates the power of ten by which the number is to be multiplied. This scale factor may be preceded by a "+" or "-" sign to indicate positive or negative powers of ten. If no sign is given, it is assumed to be positive.

Examples:

1.E-12	means 10^{-12}
.00314159E+3	means 3.14159
19.64E+2	means 1964.

As another alternative, a real constant may be expressed as an integer constant followed by a scale factor.

Examples:

55E-3	means .055
132E45	means 132 x 10^{45}
69E4	means 690000

In any representation the number must lie within the range described above for floating quantities.

IDENTIFIERS

Identifiers are used to name the variables, subprograms, and dummy arguments which appear in a FORTRAN program. An identifier is a string of letters and digits, the first of which must be a letter. The string may be any desired length, but only the first six characters will be used. Identifiers may be declared as integer or real, either explicitly or implicity. Implicit definition depends upon the first letter of the identifier. If the first letter is from the group (I,J,K,L,M, and N), then the identifier is of the class integer. If the first letter is not from this group then it is of class real. Explicit definition of identifiers is accomplished by the declarative statements REAL and INTEGER.

Examples:

```
        Real    RATE
                DECREMENT

     Integer    L307
                MØNTH
```

VARIABLES

Variables represent quantities which may assume many different values and are referred to by name. They may be either scalar or array variables, depending on the nature of the quantity they represent.

SCALAR VARIABLES

Scalar variables represent a single real or integer quantity and are written as simple identifiers.

Examples:

```
        DISTANCE
        Z
        N3
```

ARRAY VARIABLES

An array variable represents a single element within an array of quantities. The array variable is denoted by the array name followed by a subscript list enclosed in parentheses. The subscript list contains one or more arithmetic expressions separated by commas. Each expression corresponds to a subscript and the values of the expressions determine which array element is to be referenced. The number of subscripts in the list must equal the number of dimensions specified for the array.

Examples:

```
        A (4)
        BETA (M+3)
        JOHN (2*ITEM-13)
```

SUBSCRIPTS

A subscript may be any expression of the type, arithmetic; however, its significance is of integer mode and limited in range by the size of the array dimension involved. Therefore, when necessary, the value of any expression used as a subscript is truncated to an integer and reduced modulo 2^{14} before being employed in reference.

SINGLE-BIT ARRAYS

As the operands of certain GE/PAC 4000 FORTRAN statements (SET BIT, RESET BIT, IF BIT), the individual bits in the binary representations of integer scalar variables (not array variables or real veriables) may be referred to by appending a single subscript (in parentheses) to the scalar identifier. The subscript is calculated as any other subscript but refers to the bit position in the memory word and consequently has meaning only in the range 0 through 23. The appearance of an integer scalar identifier with a subscript is leagal only in SET BIT, RESET BIT and IF BIT statements.

Examples:

```
LIGHT (J)
MØDES (13)
INHIB (XMINOF(J,K+2))
```

FUNCTION REFERENCES

A function is a subprogram which acts upon one or more quantities called arguments to produce a single quantity called the function value. Functional references are dentoed by the identifier which names the function, followed by an argument list enclosed in parentheses.

```
identifier (argument, argument,..., argument)
```

An argument may be an expression or an array identifier.

The function value may in turn act as an element whose mode is determined by the mode of the identifiers naming it, or by use of the appropriate form of the FUNCTION statments, and may, therefore, be independent of the types and modes of its arguments.

Examples:

```
ATANF (ALPHA)
DATE (MONTH, DAY, YEAR)
GAMMA (N, Z*SQRTF(ZETA))
```

RESERVED FUNCTIONS

Certain commonly and frequently used arithmetic functions are provided as part of the system library and will be incorporated in the compiled program by the compiler through the use of a "LIB" operation. The names of these functions must, therefore, be "reserved" and limited to use as subprogram identifiers for the library subprograms to which they refer.

The following table lists these names together with information about the functions themselves.

Subprogram Identifier	Function	Number of Args	Modes I=integer R=real	
			Function Value	Arg(s)
ATANF	Trigonometric Arctangent	1	R	R
SINF	Trigonometric Sine	1	R	R
COSF	Trigonometric Cosine	1	R	R
SQRTF	Square Root	1	R	R
LOGF	Log base e	1	R	R
EXPF	exponential (e to a power)	1	R	R
ABSF	absolute value	1	R	R
XABSF	absolute value	1	I	I
SINHF	Hyperbolic sine	1	R	R
COSHF	Hyperbolic cosine	1	R	R
TANHF	Hyperbolic tangent	1	R	R
MODF	arg_1 - [arg_1/arg_2] arg_2 (See Note)	2	R	R
XMODF	arg_1 modulo arg_2	2	I	I
SIGNF	Signum (arg_2) *arg_1	2	R	R
XSIGNF	Signum (arg_2) *arg_1	2	I	I
DIMF	(arg_1-arg_2) if arg_1 >arg_2; else 0	2	R	R
XDIMF	(arg_1-arg_2) if arg_1 >arg_2; else 0	2	I	I
MAXOF	Maximum value of args.	Var.	R	I
MAX1F	Maximum value of args.	Var.	R	R
XMAXOF	Maximum value of args.	Var.	I	I
XMAX1F	Maximum value of args.	Var.	I	R
MINOF	Minimum value of args.	Var.	R	I
MIN1F	Minimum value of args.	Var.	R	R
XMINOF	Minimum value of args.	Var.	I	I
XMIN1F	Minimum value of args.	Var.	I	R
INTF	Integer part (truncation)	1	R	R
XINTF		1	I	R
XFIXF		1	I	R
FLOATF	Float an integer	1	R	I

NOTE - [ ] indicate "greatest integer in"

3 EXPRESSIONS

An expression is a sequence of elements separated by operational symbols and/or parentheses in accordance with conventional mathematical notation and certain FORTRAN restrictions. Two types of expressions are recognized, arithmetic and logical.

FORMATION OF EXPRESSIONS

An expression has a single numeric value equal to the result of the calculation specified by the numeric quantities and arithmetic or logical operations comprising it. The arithmetic operational symbols are "+", "-", "*", "/", and "**", denoting respectively addition, subtraction, multiplication, division, and exponentiation.

A logical or "Boolean" expression is formed in the same fashion as an arithmetic expression except that the operational symbols recognized are "+", "/", "*", and "-", denoting logical sum ("or"), logical difference ("exclusive or"), logical product ("and") and logical complement ("not") respectively. An expression is declared Boolean by the placement of a "B" in column 1 of the first line of the FORTRAN statement in which it occurs. In any statement declared Boolean, all operational symbols in expressions are interpreted in the logical sense except in those expressions used as subscripts to arrays (subscript expressions are universally of type arithmetic-integer). In Boolean expressions, all constants are interpreted as type logical-integer and appear as octal integers. All other representations are illegal. All variables in Boolean expressions must have integer definitions; all other representations are illegal.

An expression may be as simple as a single element, i.e., a constant, variable, or function reference.

Examples:

```
3,142
OMEGA (T)
COSF (DELTA)
```

Compound expressions may be formed by using operation symbols to combine basic elements.

Examples:

```
Z+2
SUMX/N
SQRTF(B**2-4*A*C)
```

An expression may be enclosed in parentheses and considered as a basic element.

Examples:

```
(Z-3)/LAMBDA
(ALEPH)
COSF (SINF(2*PI*R*T))
```

Any expression may be preceded by a "+" or "-" sign.

Examples:

```
+TEN
-(A*B)
-SINF (+ALPHA)
```

With the exception of logical complement, no two operational symbols may appear in sequence. The expression:

```
I*-J
```

is illegal in arithmetic expressions, but is allowed in Boolean expressions. The use of parentheses, yields the correct arighmetic form:

```
I*(-J)
```

Adherence to the above rules will allow the formation of all permissible expressions.

If the precedence of arithmetic operations is not stated explicitly by parentheses, it is implicitly interpreted to be as follows, in order of decreasing precedence:

Symbol	Operation
**	exponenciation
* and /	multiplication and division
+ and -	addition and subtraction

In Boolean expressions, the implied precedence in decreasing order is:

Symbol	Operation
-	logical complement
*	logical product
/	logical difference
+	logical sum

for example, the arithmetic expression

```
U*V+W/X**Y+Z
```

is taken to be

```
(U*V)+(W/(X**Y))+Z
```

Since sequences of operations of equal precedence can result in ambiguities, they are resolved by grouping from the left. Thus

```
A**B**C and
X/Y/Z
```

are interpreted as

```
(A**B)**C and
(X/Y)/Z
```

respectively.

EVALUATION OF EXPRESSIONS

Except for Boolean expressions, which are modeless, the numerical value of any expression may be of integer or real mode, as determined by the modes of its elements. There are three possible combinations: all elements are integer (integer expression); all elements are real (real expression); or both real and integer elements occur (mixed expression). All combinations are permissible in GE/PAC 4000 FORTRAN.

INTEGER EXPRESSIONS

An integer expression is evaluated using integer arithmetic throughout to yield an integer result. Fractional parts arising in division are truncated, not rounded. For example, 5/3 gives 1, 4/7 gives 0

Examples:

```
L
I*2+m
(J-3)*MAN+INDEX
```

REAL EXPRESSIONS

A real expression is evaluated using floating-point arithmetic throughout to yield a real result.

Examples:

```
((Y(N+1)+Y(N-1))/2.*DX)
COSF (ALPHA+BETA)
```

MIXED EXPRESSIONS

Mixed expressions are evaluated by converting all integer variables to real variables and then treating the expression as if it were real. The result is given as a real quantity.

Examples:

```
Z*2*(I+L)
A**J+J**C
R(K+2)*ATANF(X1)
```

BOOLEAN EXPRESSIONS

Boolean expressions are evaluated using single-word logical operations upon the elements. All constants encountered are interpreted in octal. Array elements may occur, the subscripts of which will be interpreted as arithmetic-integer expressions. All other elements (e.g., function values, expression values) are treated as one-word binary configurations.

Examples:

```
IWORD*MASKI+IFIELD
I*77/JCOMUTR
M(2*I-5)*(-N(2*I)/-L(I))
```

4 COMPUTATION STATEMENTS

ASSIGNMENT STATEMENT

The assignment statement specifies an expression which is to be evaluated, and a variable called the statement variable to which the expression value is to be assigned.

Form: variable = expression

Note that the "=" sign means replacement, not equality. The first example below is not a mathematical equation but a valid assignment statement meaning "take the value of Z, cube it and assign the result to X."

Examples:

```
X = Z**3
J = K*(L-5)
A(X) = LOGF (1+2*X)
```

The value of the expression in an assignment statement is made to agree with the mode of the statement variable before the replacement is performed. If the statement variable is real, an integer expression value will be converted prior to replacement, and conversely. For example, in the statement

```
A = 3*J+K
```

the integer value of the expression is converted to floating-point before assignment to A.

BIT ASSIGNMENT STATEMENTS

The bit assignments statements, SET BIT and RESET BIT, provide for the assignment of values (1 or 0) to the individual bits of integer scalar variables.

Form: SET BIT integer scalar variable (subscript),..., integer scalar variable (subscript)

RESET BIT integer scalar variable (subscript),..., integer scalar variable (subscript)

The individual bits referred to in the list following the SET BIT or RESET BIT preamble are "set" (set = 1) or "reset" (set =0) as indicated.

Examples:

```
SET BIT JOE (2*I-1),ISAM(23),JFLAG(PERIF(J))
RESET BIT K(3),IFLAG(PERIF(J+1)),LOC(0)
```

5 CONTROL STATEMENTS

In a FORTRAN program, control normally passes sequentially from one statement to the next in the order in which they are presented to the compiler. Control statements allow the programmer to alter this normal program flow. To implement this, FORTRAN source statements may be labeled with numbers which are referred to by control statements.

STATEMENT NUMBERS

A statement number consists of an unsigned integer constant of up to five digits. Leading zeros are ignored. The value of the integer must be greater than zero.

Although statement numbers appear in the source program as integers they must not be confused with numerical quantities. They represent a distinct type of quantity in a FORTRAN program known as a statement number, and are used for the identification of addresses within the object program.

Since statement numbers are used for identification they must be uniquely defined; i.e., no two statements may have the same number. No order or sequence is implied by statement number magnitudes, Non-referenced statements need not be numbered, in fact it is wasteful of compiler storage to do so unnecessarily.

UNCONDITIONAL GO TO STATEMENT

Form: GO TO n

when n is a statement number.

This statment transfers control to the statement numbered n.

Example:

```
GO TO 13
```

COMPUTED GO TO STATEMENT

The computed GO TO statement transfers control to one of a group of statements; the particular statement chosen is determined by the computed value of an expression.

Form: GO TO ($n_1,n_2,n_3,\ldots,n_k$), expression

where $n_1,n_2,n_3,\ldots,n_k$ are statement numbers. Control will be transferred to statement number $n_1,n_2,n_3,\ldots,n_k$ depending on whether the expression has the value 1,2,3,...,or k, respectively.

Example:

```
GO TO (37,24,36), SIZE
```

will transfer control to statement number 24 if SIZE has the value 2. The value of the expression will be truncated if required. Expression values outside the range 1,2,3,...,k will cause a runtime error indication.

Example:

```
GO TO (1,2,3,27), Y+2
```

will transfer control to statement number 27 if Y has the value 2.6183, but a value of 3.142 for Y will cause an error indication.

ASSIGNED GO TO STATEMENT

Form: GO TO integer variable or
GO TO integer variable, ($n_1,n_2,n_3,\ldots,n_k$)

This statement transfers control to the statement whose number was last assigned to the variable by an ASSIGN statement. The variable must appear in a previously executed ASSIGN statement, or a runtime error indication will result.

Examples:

```
GO TO JAIL
GO TO ERROR
```

The variable of an assigned GO TO statement is a control variable. and has a statement number rather than a numeric quantity as its value. A control variable may be shared between a program and its subprograms as may any other variable.

The ($n_1,n_2,n_3,\ldots,n_k$) is a parenthetical statement number list of footnote value only and may be included or omitted at the option of the user.

ASSIGN STATEMENT

Form: ASSIGN integer TO integer variable

This statment sets the value of the variable to be used by a subsequent assigned GO TO statement. The integer is the number of the statement to which control will be transferred by the assigned GO TO statement.

Examples:

```
ASSIGN 37 TO JAIL
ASSIGN 3 TO IERROR
```

This FORTRAN capability is very useful in transferring to a sequence of statements that is used as a subroutine by other parts of the program. For example, the statement

```
     ASSIGN 13 to IEXIT
     GO TO 44

13   ...
```

will transfer control to the sequence beginning at 44. If the sequence ends with

```
GO TO IEXIT
```

control will be transferred back to statement 13.

IF STATEMENT

Form: IF (expression) n_1,n_2,n_3

where n_1,n_2,n_3 are statement numbers. This statement transfers control to statement $n_1,n_2,$ or n_3 depending on whether the value of the expression is less than, equal to, or greater than zero respectively.

Examples:

```
IF(Y(I)-LIMIT) 6, 12, 18
IF(SUM) 3, 4, 5
```

In the first example above control is transferred to statement number 6 if Y(I)<LIMIT, to statement 12 if Y(I)=LIMIT, and to statement 18 if Y(I)>LIMIT.

IF BIT STATEMENT

Form: IF BIT (integer scalar variable (subscript))n_1,n_2

where n_1 and n_2 are statement numbers. This statement transfers contol to the statement numbered n_1 if the bit of the variable referred to is a "1" and to statement n_2 if it is a "0".

Examples:

```
IF BIT (IFLAG(NOFAIL)) 14, 13
IF BIT (K(23)) 6, 7
IF BIT (J(0)) 9, 10
```

The second and third examples accomplish negative-positive and odd-even tests respectively.

IF SENSE SWITCH STATEMENT

Form: IF(SENSE SWITCH expression) n_1,n_2

where n_1 and n_2 are statement numbers.

This statement provides a test for the individual status of each of the 24 switches on the GE/PAC 4000 console. These switches are numbered from 23 through 0, going from left to right. The expression may be of either mode but will be converted to arithmetic-integer in use and has meaning only in the range 0 through 23. If the console switch specified by the value of the expression is in the down (=1) position at the time of execution, control is transferred to statement n_1; if it is in the up (=0) position, control transfers to statement n_2.

Examples:

```
IF (SENSE SWITCH 7) 102, 103
IF (SENSE SWITCH IOREDY) 6, 7
```

IF ACCUMULATOR OVERFLOW STATEMENT

Form: IF ACCUMULATOR OVERFLOW n_1, n_2

where n_1 and n_2 are statement numbers.

This statement provides a test of the GE/PAC 4000 overflow trigger. Control passes to statement n_1 if the overflow trigger is "on" and to n_2 if the overflow trigger is "off" at the time of execution. In either case, execution of the test resets the overflow trigger to the "off" state.

Example:

```
IF ACCUMULATOR OVERFLOW 23, 31
```

IF I/O STATEMENTS

A series of IF statements has been provided for testing the status of individual I/O devices. These statements are described fully in the INPUT-OUTPUT section of this manual (Refer to page 23).

DO Statement

The DO statement allows a series of statements to be executed repeatedly under the control of a variable whose value changes between repetitions.

Forms: DO n integer scalar variable = expression$_1$, expression$_2$
DO n integer scalar variable = expression$_1$, expression$_2$, expression$_3$

where n is a statement number and expression $\leq$ expression$_2$ at object time. If, as in the first form, expression$_3$ is not stated, it is assumed to have the value, 1.

The DO statement causes the following statements up to and including statement n to be executed repeatedly. This group of statements is called the range of the DO statement. The scalar variable of the DO statement is called the index or induction variable and must be of integer mode. The values of expression$_1$, expression$_2$, and expression$_3$ are called respectively the initial, limit and increment values of the index. Each may be of either mode but will be converted to arithmetic-integer mode before use.

The initial execution of all statements within the range is always performed with the initial value assigned to the index, regardless of the value of the limit and increment. After each execution of the range the increment is added to the value of the index and the result compared with the limit. If the result has not passed the limit, the statements within the range are again executed using the new value of the index.

After the last execution program control passes to the statement immediately following statement n. Exit from the range may also be accomplished by a transfer from within the range of the DO statement. The value of the index is retained for computation purposes on both normal and abnormal exits from the DO loop.

The range of a DO statement may include other DO statements provided that the range of an "inside" DO loop is completely contained within the range of any "outside" DO loop. In other words, the range of two DO statements may not partially intersect each other. Only total intersection or no intersection is allowed.

The index of a DO statement is treated as any other scalar variable. Its value may be used for calculation outside the range of the DO statement as well as within the ranges. In addition, the values of the limit, increment, and index may be altered within the range of the DO statement.

It is also permissible to transfer into the range of a DO statement from outside the range.

Examples:

```
DO 37 I = 3, 12
DO 15 INDEX = FIRST, LAST, INCREMENT
```

As an example of the use of the DO statement consider the following sequence which will sum all the numbers within a suitably specified array.

```
    SUM = 0
    DO 3 I=1,N
3   SUM = SUM+X(I)
```

CONTINUE STATEMENT

Form: CONTINUE

The rules of FORTRAN state that the range of a DO statement cannot end with a control transfer statement. In order to gain this capability without violation of the rule, a dummy statement, CONTINUE, is provided that may be used to end the range of a DO, or as the target point for transfer statements within the range of a DO where repetition of all or part of the range is conditional. Consider the following statement sequence:

```
      DO 3 I=START, STOP
      .
      .
      .
      IF (ALPHA) 3, 17, 51
    3 CONTINUE
```

A negative value of ALPHA will initiate another execution of the range. The CONTINUE statement provides a target address for the IF statement and ends the range of the DO statement. The following sequence is illegal and must not be used since the DO loop ends with a conditional transfer.

```
    2 DO 3 I=START, STOP
      .
      .
      .
    3 IF (ALPHA) 2, 17, 51
```

CALL STATEMENT

Forms: CALL identifier
CALL identifier (argument, argument, ..., argument)

This statement is used to call (transfer control to) a subroutine subprogram. The identifer is the name of the subroutine.

The arguments, as in the case of functions, may be expressions or array identifers. Unlike a function, however, a subroutine may have more than one result and may use one or more of its arguments to return these results to the calling program. The first form of the CALL statement is used where a subroutine requires no arguments.

Examples:

```
      CALL DUMP
      CALL MATMPY (X(I,J),Y(J,K))
      CALL SEARCH (MTABLE,RALPH)
```

The mode of the subroutine name has bearing on the mode(s) of its results.

RETURN STATEMENT

Form: RETURN

This statement returns control from an external subprogram to the calling program. Therefore, the last statement executed in a subprogram will be a

RETURN statement, though it need not be physically the last statement within the program. Any number of RETURN statements may be used and they may occur at any point within the subprogram at which execution is to be terminated. A RETURN statement is necessary to return control whether a subprogram is explicitly referred to in a CALL statement, or implicitly referred to by a functional reference.

STOP STATEMENT

Form: STOP

Since, in a process control application, it is not permissible to stop the computer by program, this statement is interpreted as an instruction to MONITOR to stop operating this program. It therefore generates the calling sequence which would be associated with the statement:

```
TURN PROGRAM OFF,0,1,0,0,0
```

This means that the next time the program is turned on, it will be entered at the beginning with all flip-flops reset except the PAIF, which is reset.

STOP must appear by itself on a card.

END STATEMENT

Form: END

The END statement is used to communicate to the compiler the logical end of a program or sub-program. It causes the compiler to finish the compilation by reserving memory locations for all variables named, etc.

END must appear by itself on a card.

6 INPUT-OUTPUT

INPUT-OUTPUT STATEMENTS

Input-output statements call for the transmission of information records between computer memory and various input or output units which are attached to the computer. In general, an input-output statement provides:

1. Specification of the operation required; whether input or output and the particular unit involved.

2. Reference to a data format specifying the conversions required between internal and external data forms. This reference is to the number of a FORMAT statement.

3. A list of the variables whose values are to be transmitted. The list order of the variables corresponds exactly to the order in which the information exits in the input medium or will exist in the output medium. For example, the statement

   ```
   PRINT 20, ALEPH, BETH, GIMEL
   ```

   specifies that the values of ALEPH, BETH and GIMEL are to be printed on line according to the format specified in the FORMAT statement numbered 20.

INPUT-OUTPUT RECORDS

All information appearing in external media is grouped into records. The amount of information contained in one record and the manner in which records are separated depends upon the medium. For punched cards, each card constitutes one record; for punched paper tape, a record consists of 80 or less frames, followed by a carriage return; in printing, a record is one line of 120 or less characters, etc. The actual amount of information contained in each record is specified by the FORMAT statement.

Each execution of an input or output statement initiates the trasmission of a new data record. Therefore, the statement

```
READ 12, UN, DEUX, TROIS
```

is not necessarily equivalent to the statements,

```
READ 12, UN
READ 12, DEUX
READ 12, TROIS
```

since in the latter example at least three records are required, whereas in the former example one, two, three or even more records may be required depending on the FORMAT statement numbered 12.

If an input-output statement requests less than a full record of information, the unrequested portion of the record is lost and cannot be recovered by another input-output statement.

If an input-output statement requests more than one record of information, successive records are transmitted until the data request is satisfied.

INPUT-OUTPUT LISTS

The list of variables in an input-output statement specifies the order of transmission of the variable values. During input, new values of listed variables may be used as subscripts or in control expressions for variables appearing later in the list. For example,

```
            READ 4, J, A(J), B(J+1)
```

reads in a new value of J and uses it in the subscripts for A and B.

The transmission of array variables may be controlled by indexing similar to that used in a DO statement. A series of subscripted array variables and/or scalar variables followed by an index control may be enclosed in parentheses and will act as a single element of the input-output list. For example,

```
            READ 7, (X(I), I = 1,4)
```

is equivalent to

```
            READ 7, X(1), X(2), X(3), X(4);
```

and

```
            PRINT 8, (X(1), Y(I), Q, I = 1,2)
```

is equivalent ot

```
            PRINT 8, X(1), Y(1), Q, X(2), Y(2), Q.
```

As in the DO statement, the initial, limit, and increment values of the index may be given as expressions; e.g.,

```
            PRINT 2, K, (ARRAY(M), M =1,J,K).
```

The indexing may also be compounded as in the following example:

```
PUNCH 3, ((TIME(K,L), K = 1,4), L = 1,8).
```

This statement outputs elements of the array TIME in the order

```
TIME (1, 1), TIME (2, 1)...TIME (4, 1), TIME (1, 2)...
TIME (4, 3), etc.
```

If an entire array is to be transmitted, the indexing values may be omitted and only the array identifier written. The array is transmitted as in the previous example, in order of increasing subscripts with the first subscript varying most rapidly. Therefore, the previous example could be replaced by

```
PUNCH 3, TIME
```

READ STATEMENT

```
Forms:  READ n, list
        READ (expression) n, list
```

where n is a FORMAT statement number.

This statement causes information to be read from the input device identified by the value of the expression, converted to internal binary representation according to FORMAT statement n, and stored as values of the variables specified by the input-output list. If no expression is entered (first form), the device specified will be the one corresponding to the value 0.

PRINT STATEMENT

```
Forms:  PRINT n, list
        PRINT (expression) n, list
```

where n is a FORMAT statement number.

This statement causes informationcontained in memory as values of the variable specified by the input-output list to be converted to BCD representations according to FORMAT statement n and printed on the output device specified by the value of the expression.

As with READ, absence of the expression and its enclosing parentheses will produce a reference to device number 0.

PUNCH STATEMENT

```
     Forms:  PUNCH n, list
             PUNCH (expression) n, list
```

where n is a FORMAT statement number.

This statement accomplishes exactly the same function as the PRINT statement and in an identical manner, except that the information to be output will be punched rather than printed.

INPUT-OUTPUT TESTS

Prior to actual execution of I/O operations, it may be desirable to test the status of the "about to be employed" I/O device. For this purpose a series of IF statements of the form:

```
             IF I/O type     n1,n2,n3
             IF I/O type     (expression) n1,n2,n3
```

where n_1, n_2, n_3 are statement numbers, has been provided to accomplish the desired testing.

The six specific forms recognized are:

```
             IF READ         n1,n2,n3
             IF READ         (expression) n1,n2,n3
             IF PRINT        n1,n2,n3
             IF PRINT        (expression) n1,n2,n3
             IF PUNCH        n1,n2,n3
             IF PUNCH        (expression) n1,n2,n3
```

As with the corresponding I/O statement, the device number is indicated by the value of the expression with absence of the expression signifying device number 0.

n_1 is the statement number of the first statement in a routine to which control is to pass when the tested device is "busy", or whose "table" is full.

n_2 is the statement number of the beginning of a "faulty" or "bad" or "error condition" routine for the tested device.

n_3 is the statement number of the starting statement in an "out-of-service" routine.

If no such impediments are found to exist at the time of execution by the testing routines in the execution system, control passes to the next sequential statement following the IF I/O statement.

FORMAT STATEMENT

All input or output of BCD information requires the use of a FORMAT statement specifying the external format of the data and the types of conversions to be performed. Any FORMAT statement may be used with any input-output medium, and since they are not executed, they may appear anyplace in the program.

Form: FORMAT $(S_1, S_2, \ldots, S_k)$

where S is a data field specification.

NUMERICAL FIELDS

Four different types of numerical data conversion are available:

1. Type E

   Internal Form - Binary floating-point
   External Form - Decimal floating-point

2. Type F

   Internal Form - Binary floating-point
   External Form - Decimal fixed-point

3. Type I

   Internal Form - Binary integer
   External Form - Decimal integer

4. Type 0

   Internal Form - Binary integer
   External Form - Octal integer

The four types of conversions are specified by the following general forms:

1. Ew.d
2. Fw.d
3. Iw
4. Ow

The letter E, F, I, or 0 designates the conversion type; w is an integer specifying the data field width and must be ≤63. And d is an integer specifying the number of decimal places to the right of the decimal point. The value of d must be ≤15. For E-type output conversion, w should exceed d by at least 7. For example, the statement:

```
FORMAT (I3, F10.4, E15.5, 08)
```

could be used to print the line

```
        37        -14.2639     0.46972E 3     37777
```

on the output listing.

Note that plus signs are not printed.

Note also that in type F conversion there is a decimal point but no exponent, whereas type E conversion has an exponent. On output the exponent always has the form shown, i.e., an "E" followed by a signed two digit integer. On input, however, the "E" or the sign or the entire exponent may be omitted. The following are all valid E15.6 data fields for input:

```
.327409+2
.964718E4
.003276-4
305976.
```

The field width w includes all of the characters (decimal point, signs, blanks, etc.) which comprise the number. If a number is too long for its specified data field the excess characters are lost. Since numbers are right justified within their fields, the digits lost are those with the most significance. The spacing of output data for legibility must be considered when specifying a data field width.

During input, the appearance of a decimal point "." in an "E" or "F" data field specification overrides the "d" specification of the field. In the absence of an explicit decimal point, the point is positioned d places from the right of the field not counting the exponent, if one is present. For example, a number appearing externally as 314159E-1 with a data field specification of E12.5 will be interpreted as 3.14159E-1.

In addition, a scale factor, indicated by a signed or unsigned decimal integer followed by the letter, "P", may be employed with E and F fields. If used with an "F" field, it will achieve multiplication of floating quantities by powers of 10 before conversion on output or after conversion on input. If used with an "E" field, it will be ignored on input and will cause P whole number digits to precede the decimal point and the exponent to be decreased by P. The P-factor must not be negative for use with E-fields. The maximum absolute value must be ≤9. A P-factor holds for all succeeding E and F fields in the FORMAT statement. To end its influence, a P-factor of zero must be stated.

Examples:

```
 6PE9.3
2P4F7.2
```

Suppose that the above examples are employed as follows:

```
        PRINT 7, X, (PRCNT(I), I = 1,4)
    7   FORMAT (PE10.3, 2P4F7.2)
```

Assume further that the values of the variables concerned are:

X	6,494,650
PRCNT (1)	.012345
PRCNT (2)	.596938
PRCNT (3)	.856062
PRCNT (4)	1.56032

The resulting printed line would appear as:

```
0.649E   7      1.23      59.69      85.61      156.03
```

ALPHANUMERIC FIELDS

Alphanumeric data can be transmitted in much the same manner as numeric data through use of the form Aw where A is a control character and w is the number of characters in the field and is ≤63. The alphanumeric characters are transmitted as the value of a variable in an input-output list. The variable may be of either mode. For example,

```
         READ 17, V
    17   FORMAT (A3)
```

will cause three characters to be read and placed in memory as the value of the variable V.

If the format specification had indicated a field width greater than 4, the additional characters will be grouped in fours and placed in V + 1, V + 2, --- etc., until the entire field width is satisfied. If the last such group contains less than 4 characters, they will be left-justified.

ALPHANUMERIC FORMAT FIELDS

An alphanumeric format field may be specified by preceding the alphanumeric string with the specification nH, where n is the number of characters in the string including blanks and has a maximum value of 120. For instance, the following sequence:

```
      PRINT 3
    3 FORMAT (9H NOT DONE)
```

will print the words NOT DONE on line. The n characters of the data field are not transmitted as the value of a variable, but are stored in the memory space allotted for the FORMAT statement itself. The n characters may be replaced by n other characters by means of an input statement which references the FORMAT statement. The value of n must be ≤120.

An input-output list is not required for the transmission of this type of field. During input, n characters are extracted from the input record and used to replace the n characters within the specification. During output the n characters specified or the n characters which have replaced them become part of the output record. For example, the sequence-

```
     READ 3
   3 FORMAT (9H NOT DONE)
     PRINT 3
```

will print the words ALL DONE on line, provided ALL DONE appears in positions 1-9 of the input record being processed by the READ 3 statement.

ALPHANUMERIC NOTES

Alphanumeric information can also be placed in an output line by use of a special feature, called a "note". For instance, the words "NOT DONE" as shown in the previous example could be inserted by:

```
     PRINT 3
   3 FORMAT ($ NOT DONE$)
```

The number of characters between the two "$" symbols would be counted and the same information would be generated as in the previous example. Note that the "$" symbol cannot be a member of a note, since it signals termination of the note.

BLANK OR SKIP FIELDS

Blanks may be introduced into an output record, or characters skipped on an input record by use of the specification nX. The control character is X and n is the number of blanks or characters skipped with a maximum value of 31. For example, the statement:

```
     FORMAT (5H JOB I3, 10X, 4HDONE)
```

may be used to output the line-

```
     JOB 397       DONE
```

with 10 blanks separating the two quantities.

MIXED FIELDS

An alphanumeric format field may be placed among the other fields of the FORMAT statement to enhance the readability of output listings. For example:

```
FORMAT ( 8H FORCE = F9.4, 5H LBS.)
```

may be used to print-

```
FORCE - 297.6374 LBS
```

Note that the separating comma may be omitted after an alphanumeric format field.

REPETITION OF FIELD SPECIFICATIONS

Repetition of a field specification may be specified by preceding the control character E, F, I, A, or 0 by an unsigned integer less than or equal to seven giving the number of repetitions desired. The specified number of repetitions must not be zero. For example:

```
FORMAT (3I2, 2F12.6, 204)
```

is equivalent to

```
FORMAT (I2, I2, I2, F12.6, F12.6, 04, 04)
```

REPETITION OF GROUPS

A group of field specifications may be repeated by enclosing the group within parentheses and preceding the whole group with the number of desired repetitions. For example,

```
FORMAT (3I2, 2(F12.4, 3E8.2))
```

is equivalent to

```
FORMAT (3I2, F12.4, 3E8.2, F12.4, 3E8.2).
```

Alphanumeric fields may also be repeated in this manner,

```
  PRINT 7                      PRINT 8
7 FORMAT (3(5H BANG))   or   8 FORMAT (3($ BANG$))
```

will cause the following on line output,

```
BANG BANG BANG
```

Five levels of repeated groups are allowed.

MULTIPLE RECORD FORMATS

In the case where a group of successive input-output records have different field specifications, a slash, "/", is used to separate those field specifications. For example, the statement

```
FORMAT (408/I3, 3F12.8/12A3)
```

is equivalent to

```
FORMAT (408)
```

for the first record,

```
FORMAT (I3, 3F12.8)
```

for the second record, and

```
FORMAT (12A3)
```

for the third record.

The comma separating data field specifications may be omitted when a slash is used. Blank records may be written on output or records skipped on input by using consecutive slashes. On printed output, a slash always causes a skip to a new line after completing the record. Both the slash and the closing parenthesis at the end of format indicate the termination of a record. If an input-output statement list indicates that data transmission is to continue after the closing parenthesis of a format statement is reached, the format is repeated from the last left parenthesis.

The statement

```
FORMAT (2E12.6, 2(3F7.4, 208, I7))
```

is equivalent to

```
FORMAT (2E12.6, 3F7.4, 208, I7, 3F7.4, 208, I7)
```

for the first record, and

```
FORMAT (3F7.4, 208, I7)
```

for all succeeding records.

The total of all field widths specified for a record is the length of that record. If the record length specified is greater than the maximum allowable on a particular input-output device, the excess characters are lost.

DRUM TRANSFER STATEMENTS

Two statements, READ DRUM (drum to core) and WRITE DRUM (core to drum), provide a method for transferrring blocks of information (programs and data) between core memory, where execution takes place, and drum memory, where large amounts of bulk storage are available. Each statement calls for the transfer of a single contingucus block and, therefore, no input-output list is employed.

Forms: READ DRUM (symbolic$_1$), (symbolic$_2$), (symbolic$_3$)
WRITE DRUM (symbolic$_1$), (symbolic$_2$), (symbolic$_3$)

where

Symbolic$_1$ is the initial drum address in PAL symbolics.
Symbolic$_2$ is the number of words to be transferred in PAL symbolics.
Symbolic$_3$ is the initial core address in PAL symbolics.

7 COMMUNICATION WITH REAL-TIME MONITOR

In order that programs may undergo dynamic scheduling and relocation as components of a real-time complex, certain statements which communicate initiation, termination, delay and segmentation of the components to the overall control or MONITOR routine become necessary. Description of a series of such statements provided within GE/PAC FORTRAN follows.

TURN PROGRAM ON STATEMENT

Form: TURN PRØGRAM n ØN, expression

where n is an integer constant indicating the "program number" by which the program to be "turned on" is known to the MONITOR. The value of the expression specifies the time of next execution of the "turn on" program with a 0 value indicating immediate execution.

Example:

```
TURN PROGRAM 21 ON, NOW + 1000
```

TURN PROGRAM OFF STATEMENT

Form: TURN PROGRAM OFF, m_1, m_2, m_3, Return, m_4

This statement turns the current program off, i.e., places it in the inactive state

Example:

```
TURN PROGRAM OFF, 0, 1, 0, NXENTY, 0
```

m_1, m_2, m_3, m_4, and Return are PAL symbolic fields interpreted as follows:

m_1:	1 = set ØVRF
m_2:	1 = set PAIF
m_3:	1 = set TSTF
m_4:	1 = set TMFF to TRAP
Return:	Symbolic location for ECP entry at time of next "turn on"

(TMFF-Memory Fence Flip-Flop)

DELAY PROGRAM STATEMENT

Form: DELAY PROGRAM, m_1, m_2

This statement places the current program in an inactive state for a specified length of time.

m_1 and m_2 are PAL symbolic fields indicating the following:

m_1: 1 = program's area of occupancy is now available.
0 = area remains unavailable

m_2: indicates length of delay

SEGMENT STATEMENT

Form: SEGMENT n, m_1, m_2, m_3, NXENTY, m_4 .

where n is the program number of the following "segment" to be assigned and recognized by MONITOR (as in TURN PROGRAM ON).

Aside from its declarative properties (see section of this manual titled DECLARATION), SEGMENT n is equivalent to the following statement sequence.

TURN PROGRAM n ON, 0
TURN PROGRAM OFF, m_1, m_2, m_3, NXENTY, m_4

m_1, m_2, m_3, m_4, and NXENTY are as defined in TURN PROGRAM OFF.

8 DECLARATIONS

A declaration describes certain properties of a FORTRAN program, as opposed to the imperative assignment, control or input-output statements. Several FORTRAN statements are reserved for the purpose of supplying the system with declarative information. These statements are primarily concerned with the interpretation of identifiers occurring in the source program and with memory allocation in the object program.

CLASSIFICATION OF IDENTIFIERS

Each identifier in a source program is classified in accordance with the FORTRAN element it identifies. Four main classifications are recognized:

1. Scalar identifiers
2. Array identifiers
3. Subprogram identifiers
4. Dummy identifiers
5. Single bit arrays identifiers

The classification is made according to the context in which the identifier makes its first physical appearance in the source program. This first appearance amounts to a declaration, explicit or implicit, of the proper interpretation of the identifier throughout the program.

MODE DECLARATIONS

In addition to being classified, each identifier appearing in a FORTRAN program is of mode integer or real. The statements INTEGER and REAL are used to specify identifer modes explicitly. An identifier may appear in only one of these statements and this appearance must precede the use of the identifier in any non-declarative statement.

Identifiers whose mode is not explicitly declared are assigned modes implicitly according to the following convention.

1. Identifiers beginning with I, J, K, L, M or N are assigned integer mode.

2. Identifiers not included in the above classification are assigned real mode.

INTEGER STATEMENT

Form: INTEGER identifier, identifier,..., identifier

This statement declares the listed identifiers to be integer; each of them will be assigned to a single word.

Example:

```
INTEGER PHI, KAPPA, SIGMA
```

Notice that KAPPA need not appear in this statement since it would be declared integer mode implicitly.

REAL STATEMENT

Form: REAL identifier, identifier,..., identifier

This statement declares the listed identifiers to be real mode; each of them will be assigned to two words.

Example:

```
REAL FORCE, MASS, LOG
```

Notice that FORCE need not appear in this statement since it would be declared real mode implicitly.

DIMENSION STATEMENT

The DIMENSION statement declares an identifier to be an array identifier, and also specifies the number and limits of the array subscripts. Any number of arrays may be declared in a single DIMENSION statement.

Information provided by a DIMENSION statement is required for the allocation of storage for arrays. Each array variable appearing in a program must be previously declared as an element of an array in a DIMENSION statement. Each array variable must have the same number of subscripts as were declared for the array, and the value of each subscript must lie within the limits specified by the DIMENSION statement.

Form: DIMENSION S_1, S_2,...,S_k

when S is an array specification.

Each array specification gives the array names and the minimum and maximum values that each of its subscripts may assume. The minimum and maximum values for each subscript must be given as signed or unsigned integer constants with the maximum value greater than the minimum value. An array identifier may have any number of subscripts.

Two forms for specification of the maximum and minimum subscript values are recognized:

identifier ($\mathrm{min}_1/\mathrm{max}_1$,...)
identifier (max_1,max_2,...).

In the latter form, a minimum value of one is implied, i.e., the latter specification is equivalent to

identifier (1/max_1, 1/max_2,...).

The two forms are syntactically independent and may, therefore, both occur as different subscript limit specifications in the array specification.

Example:

```
DIMENSION X(10), Y(-1/5,20), Z(-3/-1,2,0/3,5,2)
```

SUBPROGRAM DEFINITION STATEMENTS

The two types of subprograms which may be called, or referred to by a FORTRAN program are classified as external or internal subprograms.

Internal subprograms are defined within the program that calls them, and are defined within a single statement known as an arithmetic function definition statement. Internal subprograms may be used only within the program containing their definition. External subprograms are defined separately from (externally to) the program calling them. They are complete autonomous FORTRAN programs within themselves, and as such are compiled independently. There are two types of external FORTRAN subprograms which may be declared: FUNCTION subprograms and SUBROUTINE subprograms, both of which are described below. Any subprogram, whether internal or external may call other subprograms; however, recursion is not allowed. All subprograms constitute closed subroutines, i.e., they appear only once in the object program regardless of the number of times they are called.

DUMMY IDENTIFIERS

A subprogram definition statement declares those identifiers appearing as arguments of the subprogram to be dummies. They are used as ordinary identifiers within the subprogram definition and indicate the mode and use of the arguments. Dummy identifiers are replaced by actual arguments when the subprogram is executed.

ARITHMETIC FUNCTION DEFINITION STATEMENT

Form: identifier(identifier, identifier,..., identifier) = expression

This statement completely defines an internal subprogram. The first identifier is the name of the subprogram being defined.

Arithmetic function subprograms are functions; i.e., they are single valued and must have at least one argument. The mode of the function is determined by the mode of the function identifier.

The identifiers enclosed in parentheses represent the arguments of the function. These so-called dummy identifiers have meaning and must be unique only within the defining statement. They may in fact be identical to identifiers appearing elsewhere in the program. They must agree in order, number, and mode with the actual arguments given at execution time.

The use of an argument within the definition statement is specified by the use of its dummy identifier. Expressions are the only permissible arguments for internally defined functions, therefore, dummy identifiers may appear only as scalar variables in the defining expression. The use of an array identifier or a subprogram identifier is not allowed.

Identifiers appearing in the definition statements which do not represent arguments are treated as ordinary variables. In addition, external functions or other previously defined internal functions may appear in the definition statement.

Examples:

```
F(X,Y) = (X+Y)*(X-Y)
SINH(ZETA) = (EXP(ZETA/B)-EXP(-ZETA/B))/2
Q(X,Y,Z) = F(X,Y)/SINH(Z)
```

In the second example above, ZETA is a dummy identifier and B is an ordinary identifier. At execution time the function will be evaluated using the current value of B. The third example is allowable if the first and second definitions precede it in the program.

If the accompanying source program contained the statement

```
Y = A*SINH(ALPHA).
```

The arithmetic function defining SINH would be evaluated using the current value of ALPHA as its argument. The result would be multiplied by A and the product assigned as the value of the variable Y.

All internal subprogram definitions of any FORTRAN II program must precede the first non-declarative statement of the program.

FUNCTION SUBPROGRAMS

Like the Arithmetic Function Definition Statement, the FUNCTION subprogram is also a function, is single valued, and is referenced as a basic element in an expression. FUNCTION subprograms may be used when more than one FORTRAN statement is needed to define the functional relationship. In order to logically separate a FUNCTION subprogram from the calling programs, the subprogram always begins with a FUNCTION declaration, ends with an END statement, and returns control to the main program via the RETURN statement.

FUNCTION STATEMENT

Forms:

```
FUNCTION identifier (identifier, identifier,..., identifier)
REAL FUNCTION identifier (identifier, identifier,..., identifier)
INTEGER FUNCTION identifier (identifier, identifier,..., identifier)
```

This statement declares the following program to be a function subprogram. The first identifier is the name of the subprogram being defined. This identifier must appear as a scalar variable and is assigned the value of the function resulting from execution of the subprogram. The function mode is declared implicitly by the initial letter of the function name, or may be declared explicitly by using the second or third forms of the FUNCTION statement.

Those identifiers appearing in the list enclosed by parentheses are dummy identifiers which represent the function arguements, and must agree in number, order, and mode with the actual arguments given at execution time. These arguments may be array names as well as expressions. Therefore, the dummy identifiers may appear as array identifiers or scalar identifiers. A FUNCTION statement must have at least one argument. Dummy identifiers representing array names must appear in a DIMENSION statement in the FUNCTION subprogram as well as the calling program, and must agree with the specification of the arrays in the calling program.

Examples:

```
FUNCTION SEARCH (LIST,ALPHA)
REAL FUNCTION INDEX (A1,B2,C3)
INTEGER FUNCTION DELTA (ARG1,ARG2)
```

SUBROUTINE SUBPROGRAMS

The SUBROUTINE subprogram is not a function in that it may be multivalued and is referred to only by the CALL statement. The SUBROUTINE subprogram begins with the SUBROUTINE declaration, ends with an END statement, and returns control to the calling program via the RETURN statement.

SUBROUTINE STATEMENT

Form: SUBROUTINE identifier (identifier,identifier,...,identifier)

This statement declares the following program to be a SUBROUTINE subprogram. The first identifier is the name of the subroutine. Those identifiers appearing in the list enclosed in parentheses are dummy identifiers which represent the subprogram arguments. As in the FUNCTION statement, these arguments may be scalar or array identifiers.

Dummy identifiers representing array names must be declared in a DIMENSION statement within the subprogram and must agree with the specifications of corresponding arrays specified in the program containing the CALL statement.

Contrary to the FUNCTION and Arithmetic Function Definition Statements, the result of a SUBROUTINE subprogram is not necessarily a single value and the value of the name of the subprogram may or may not (depending on the subprogram) be meaningful. Instead, specific results are normally returned as values of variables in the argument list.

A SUBROUTINE subprogram does not necessarily require arguments; in the absence of an argument list no results can be returned via the arguments.

Examples:

```
SUBROUTINE BURST
SUBROUTINE FACTOR (COEF1,COEF2,ROOT)
```

IMPLICIT DECLARATION OF IDENTIFIERS

Identifiers appearing in declaration statements such as DIMENSION, FUNCTION, SUBROUTINE, etc., are explicitly classified by their appearance in that statement. If the first appearance of an identifier is in an imperative statement rather than a declarative statement, the identifier is classified implicitly according to its context.

For example, in

```
DIMENSION ALPHA (5,10)
```

the identifier ALPHA is explicitly declared to be a two-dimensional array. Similarly

```
FUNCTION ALPHA (X,Y)
```

explicitly declares ALPHA to be a single-valued function with arguments X and Y. Conversely,

```
ALPHA (X,Y) = 2*X+Y
```

implicitly declares the identifier ALPHA to be an internal subprogram because it appears in an arithmetic function definition statement. Note also in the three previous examples that in the absence of declarative statements to the contrary, ALPHA is implicitly declared to be the identifier of a real quantity. As a further example note that in the statement

```
SUBROUTINE ALPHA (X,Y)
```

ALPHA is explicitly declared to be the name of a subprogram with arguments represented by dummy identifiers X and Y.

However, if the expression

```
ALGFNC + ALPHA (A,B)+Z/3
```

has not been preceded in a program by any of the four previously mentioned declaration statements, ALPHA will be implicitly declared to be an external subprogram whose presence will be mandatory at runtime for execution of the object program.

MEMORY ALLOCATION

Memory allocation statements are used to supply the system with supplemental information regarding the storage of variables and arrays; and when the program is to be subordinate to a real-time system, to supply memory assignment of the program in bulk memory and specify coincident cross-references as may be ne cessary.

DEFINE STATEMENT

Form: DEFINE identifier_1(symbolic), identifier_2(symbolic),...

The DEFINE statement provides a method of incorporating uniform reference to variables which are to be part of a "permanent memory space" in an overall system.

The identifiers occurring in the list may be array or scalar identifiers, or subprogram names. The symbolics enclosed within the parentheses may be PAL symbolic expressions to which the preceding identifier will be equated by the compiler.

Identifiers appearing in DEFINE statements are assumed to have space allocated by the overall system and are not, therefore, assigned space by the memory allocation section of the compiler.

The temporary storage area created by the compiler is named $TEMP. It can be assigned to permanent memory by naming it in the DEFINE Statement.

Example:

```
DEFINE X(V01T25), SQRTF(/3000), $TEMP(/4020), COMMON(/3127)
```

COMMON STATEMENT

Form: COMMON identifier, identifier,...,identifier

Within the space allocated by the overall system as "permanent memory", a section is set aside with the symbolic label of the first locations as "Common". For this purpose, the symbol, COMMON, is reserved in both the FORTRAN and PAL languages and is used solely for reference to this specially named section of memory.

The identifiers in the COMMON statement may be either array or scalar names provided that the array names also appear in a DIMENSION statement in the same program.

During allocation, each variable to be allocated in COMMON is assigned the next block (one word for scalars, array size for arrays) relative to the previous allocation with the first such allocation being a simple equation to the symbol, "COMMON".

Example:

```
COMMON    SCALAR,J, ARRAY, A
```

In the above example, suppose that SCALAR and J are scalars and that ARRAY and A are dimensioned elsewhere as ARRAY (-1/3,5) and A(100). The resultant equations to COMMON would be:

```
SCALAR    EQL   COMMON
J         EQL   COMMON + 2
ARRAY     EQL   COMMON + 3
A         EQL   COMMON + 53
```

In view of the implicit property of the symbol, COMMON, as a member of the same space as that referrred to by the DEFINE statement, the above equations to COMMON can be seen as equivalent to those which would be produced by

```
DEFINE SCALAR (COMMON), J(COMMON +2),
ARRAY (COMMON + 3), A(COMMON +53)
```

EQUIVALENCE STATEMENT

The EQUIVALENCE statement allows more than one identifier to represent the same quantity.

Form: EQUIVALENCE $(R_1,R_2,...,R_n),(R_k,R_{k+1},...,R_m)$

where R denotes a location reference.

The location references of an EQUIVALENCE statement may be simple scalar or array identifiers, or may be identifiers appended by an integer constant enclosed in parentheses. All location references enclosed within the same parenthetical expression share the same storage location. Such a group is known as an equivalence set. For example,

```
EQUIVALENCE (BLACK,WHITE)
```

states that the identifiers BLACK and WHITE refer to the same storage location.

To refer to a specific location in an array, that location must be appended to an array identifier as an integer constant. For example, if A is a scalar variable and B is an array the statement

```
EQUIVALENCE (A,B(5))
```

specifies that A and the fifth location of the array B share the same storage location.

To refer to a specific quantity in a multiply dimensioned array, the location of that quantity must first be calculated. As an example, consider the three dimensional array specified by

$$\text{DIMENSION CUBE } (L_1/U_1,\ L_2/U_2,\ L_3/U_3)$$

where L_i and U_i denote the lower and upper limits of the i th subscript.[1] To calculate the location of

$$\text{CUBE } (K_1,K_2,K_3)$$

use the formula

$$\text{LOCATION} = (U_2-L_2+1)(U_1-L_1+1)(K_3-L_3)+(U_1-L_1+1)(K_2-L_2)+K_1-L_1$$

Therefore the statements

```
DIMENSION SPACE (12), VOLUME (2,2,2)
EQUIVALENCE (SPACE(3), VOLUME (7))
```

specifies that the quantities SPACE (3) and VOLUME (1,2,2) will share the same storage location. Notice that only the relative location of the quantities within the array matters, since the entire array is adjusted to satisy the EQUIVALENCE statement. In the example above, the statement

```
EQUIVALENCE (SPACE(1), VOLUME (5))
```

would have had the same effect as

```
EQUIVALENCE (SPACE(3), VOLUME (7))
```

Where the location of a variable is known relative to a second variable, this location may be specified by appending an integer constant to the identifier of the second variable. The integer to be used is determined by considering the sequence of quantities as an unidimensional array. For example, if we have in storage at

```
LOCATION

          L1:  ALEPH
          L2:  BETH
          L3:  GIMEL
          L4:  DALETH
```

the statement

```
EQUIVALENCE (GAMMA,ALEPH(3))
```

will specify that GAMMA and GIMEL refer to the same storage location.

Note that the property of equivalence is transitive; the statement

EQUIVALENCE (X,Y),(Y,Z)

has the same effect as

EQUIVALENCE (X,Y,Z)

JOINT DEFINE, COMMON AND EQUIVALANCE RULES

In the event that identifiers appear mutually in DEFINE, COMMON and/or EQUIVALENCE statements, the DEFINE statement takes precedence followed by the COMMON statement.

All equivalences to DEFINE and COMMON variables are equated off, i.e., removed from the remaining equivalence sets and entered in the object code as equations to symbols occurring in DEFINE statements or to the symbol, COMMON.

BEGIN PROGRAM AT STATEMENT

Form: BEGIN PROGRAM AT n

where n is a PAL symbolic specifying the location in bulk memory into which the ensuing program is to be loaded.

SEGMENT STATEMENT

Form: SEGMENT n, m_1, m_2, m_3, NXENTY, m_4

where n is the MONITOR recognized program number of the following segment. This statement is used to break a large program into a number of smaller segments.

In addition to segmentation by turning "off" and turning the next part (segment) "on", the SEGMENT statement causes the execution (at compile time) of a FORTRAN "END". This execution allocates storage, published errors and removes all known names and labels from the tables before proceeding to the next job. Therefore, the only information carried forward to the next "segment" will be the relative location in bulk memory. Care must be taken to assure that variables which are common to segments are declared in each segment by "COMMON" or "DEFINE" statements.

9 DIAGNOSTICS

STATEMENT DIAGNOSTICS

Statements which violate the syntax or semantic rules of the FORTRAN language are detected during compilation and are discarded. An error message is printed on the line printer and compilation proceeds as if the erroneous statement had never been encountered. The error message consists of the statement in the form in which it entered the computer; then one character of the statement is indicated by a dollar sign, "$", printed beneath it. For example, in

```
13 FUNC = 2*X+Y*+A
                $
```

The character "+" is marked as an error. In the case of syntax errors, the marked character itself is unacceptable as in the above example. In the case of a semantic error, an identifier or other construct is being improperly used, and the dollar sign indicates the last character of the construction. For instance, in the statement

```
ASUM = PART**3+4
          $
```

the dollar sign would indicate that the identifier PART is being used incorrectly; e.g., PART may be the name of a subprogram.

The compiler will try all possible legal interpretations of a statement before finally discarding it. The dollar sign position indicates the greatest amount of correct information, starting from the left, that was found under any assumption about the statement.

A comment indicating the reason for the error is then printed at the left margin after the marked line. These comments are as follows:

SYNTAX

This comment usually occurs because of erroneous punctuation or illegally constructed arithmetic expressions.

NUMBER

A constant, label, or input-output symbolic unit number is too large or is incorrectly constructed.

ID DECLARATION

The identifier indicated is being used in a manner contradictory to a previous declaration.

SUBSCRIPTS

There are too many subscripts, an expression describing a subscript is incorrectly constructed, or the number of subscripts used in an array variable does not agree with the number declared in the DIMENSION statement.

ALLOCATION

1. A negative or zero array size was specified in a DIMENSION statement.

2. The rules of COMMON or EQUIVALENCE have been violated. In either statement the identifier causing the violation is marked.

PROGRAM OVERFLOW

The working core storage available to the compiler has been exceeded.

ARGUMENTS

A FORTRAN reserved function has been employed with the wrong form and/or number of arguments.

PROGRAM DIAGNOSTICS

Comments concerning labeling and allocation errors are listed at the end of compilation. The label errors comment is followed by a list of statement labels, and the allocation errors comment by a list of the offending identifiers.

LABEL ERRORS

1. The following statement numbers were used in control instructions within the program but referred to an unnumbered statement.

2. Two or more statements have the same label.

3. The statement closing a DO loop was never reached.

4. The final statement in the range of a DO loop was a transfer statement.

5. The DO loop was illegally nested.

ALLOCATION ERRORS

The identifiers that follow this error message violate DEFINE, COMMON and/or EQUIVALENCE rules, such as the use of a function or subroutine name as the name of a variable in either a COMMON or EQUIVALENCE statement.

APPENDIX A

SUMMARY OF GE/PAC FORTRAN STATEMENTS

COMMENT RECORD

GE PAC 4000 Double-Word Fortran Reference Manual

HOW WELL DOES THIS PUBLICATION PERFORM ITS INTENDED FUNCTION? YOUR CRITICISM IS INVITED FOR THE IMPROVEMENT OF THIS DOCUMENT. PLEASE COMMENT ON THE EFFECTIVENESS OF PRESENTATION AND SUGGEST ANY INCLUSIONS WHICH WERE INADVERTENTLY OMITTED.

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

To enable us to send you the missing information, please complete the following and forward to the PROGRAMMING LIBRARY

GENERAL ELECTRIC COMPANY
PROCESS COMPUTER SECTION
P. O. BOX 2918
PHOENIX, ARIZONA - 85002

NAME______________
COMPANY______________
ADDRESS______________
CITY__________STATE______
ZIP CODE______

GE/PAC 4000

MONITOR TRAINING MANUAL

December 1965 Revised
Library Control No. YPG31M

This manual published by:
PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. BOX 2918
PHOENIX, ARIZONA - 85002

CONTENTS

INTRODUCTION

The GE/PAC MONITOR Training Manual presents ways of using the Monitor System on the GE/PAC 4000 Process Automation Computers.

This manual has been designed especially for the Monitor user. It tells him what Monitor does and what he must do in order to use it. The individual subprograms which make up the Monitor System are discussed. The communication links which must be included in the functional programs in order to communicate with Monitor are also given. A detailed analysis of the Monitor logic is reserved for the program write-ups and is not included in this manual.

The reader is expected to be familiar with real-time system requirements. He should also know GE/PAC programming techniques and the PAL Assembly Language.

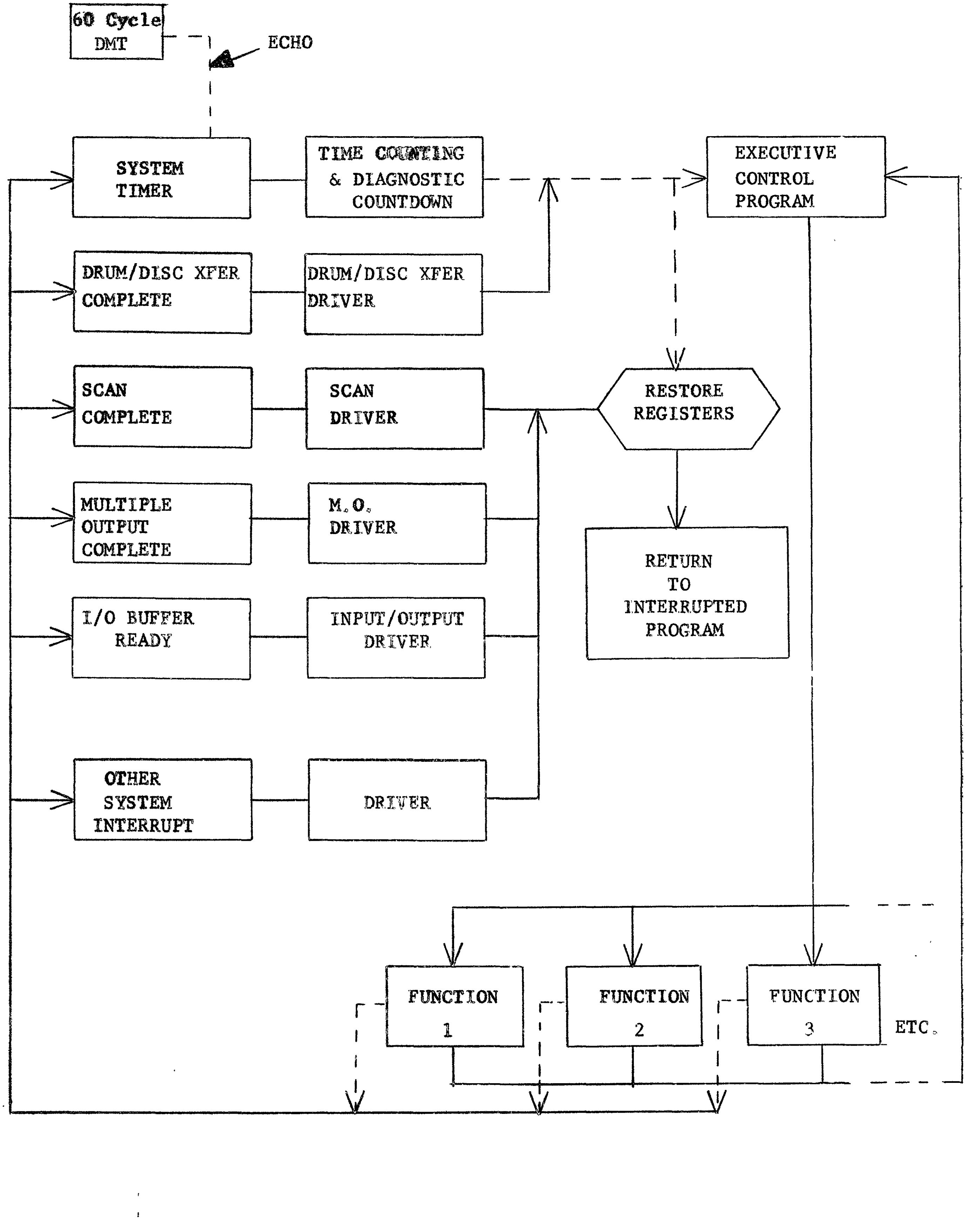
GENERAL

A real-time process is characterized by the occurrence of many events, some continuous, others random in nature. Events may occur simultaneously A digital computer, however, is a serial device; that is, it performs its program operations serially, one by one. Therefore, the matching of the digital computer and a real-time process requires a control system which coordinates the requirements and characteristics of both.

A GE/PAC Monitor is an operating system composed of a library of subprograms which provide the basis for a process computer system. It is the framework to which the specific functional programs are added. The Monitor accomplishes the timing and scheduling operations, input/output, internal data transfer, timed contact, multiple output, corrective action, initialization, etc. Many options are available which may or may not be included in a tailored Monitor System.

The user requests a tailored Monitor by checking the desired options on the Monitor Checklist. The Checklist is obtained from the Programming Librarian.

GE/PAC MONITOR INTERRUPTS



1. SYSTEM DESCRIPTION

Monitor consists of the component programs, as described below.

1.1 Executive Control Program (ECP)

The Executive Control Program schedules the execution of programs based on priority, execution time, and core availability. All time-critical interrupts are permitted before system programs are executed.

System programs are executed in priority order by comparing the programs' next execution time ($PROG_n$) with the current time (TIME). The highest priority program for which the execution time is equal to or less than the current time is executed. When the execution time is current for a program, the ECP requests a transfer from drum or disc to core providing:

1. The program is not in core
2. The program is not presently being transferred.
3. A core area is available.

After the transfer has been completed, the program is initiated. If there is no available core area for that program, ECP tests the execution time for the next lower priority program.

If a program has a "turned off" or "locked out" code in its PRØG location, it is not executed until a time for execution is assigned.

In the AU2, there are three classifications of register storage for functional programs. They are:

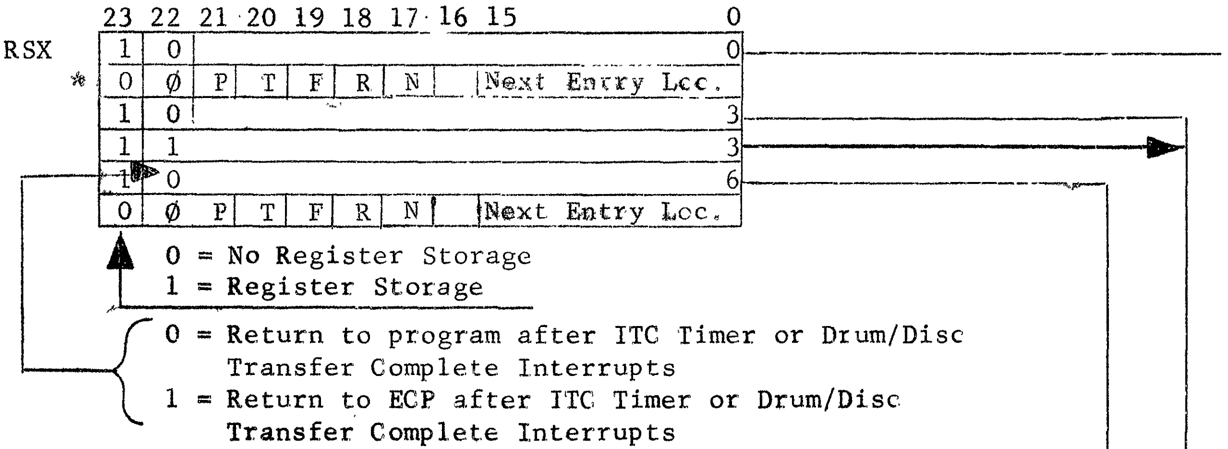
1. Programs which have no register storage of their own.
2. Programs which have their own 8-word block of register storage.
3. Programs which share an 8-word block of register storage with other functional programs.

The Register Pointer Table (RSX) tells what classification is assigned for each program and contains the following information:

__Note__: All programs in the AU1 have their own 8-word block of register storage. The RSX Table is eliminated when all AU2 system programs have their own 8-word register storage block.

Register Pointer Table (AU2 Only)

This table contains an index to the 8-word storage block for programs having full register storage or sharing storage. For programs with no register storage, the table contains the flip-flop status and the program's next entry location.



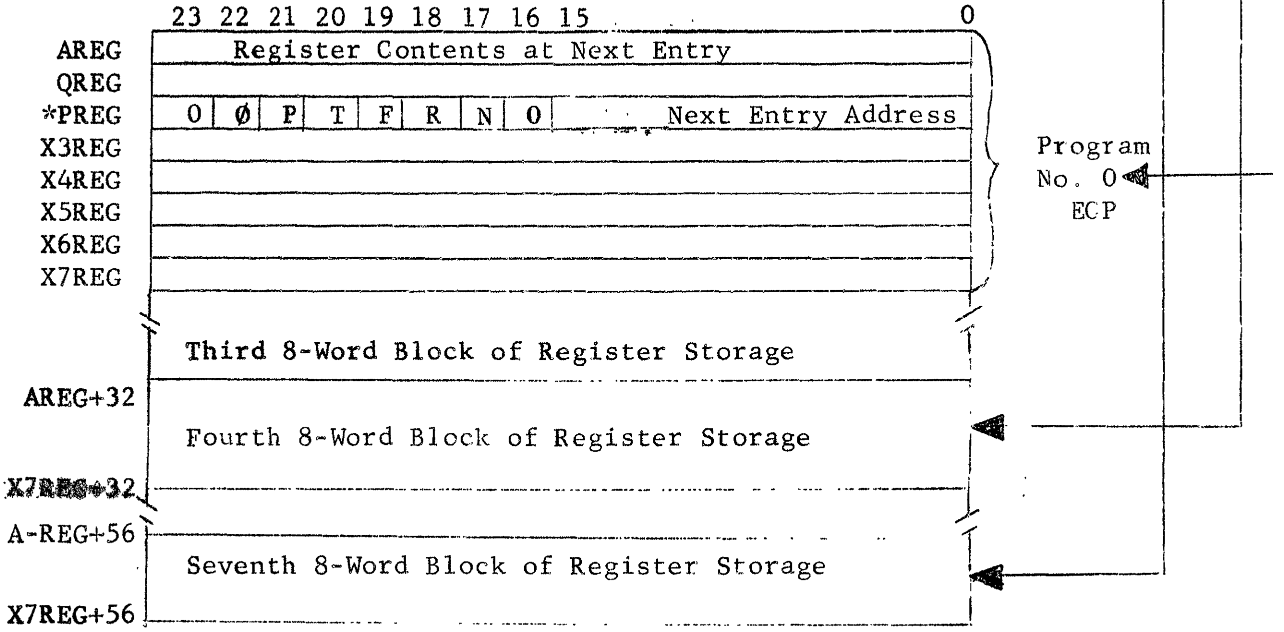
	23	22	21	20	19	18	17	16	15 0
RSX	1	0							0
*	0	Ø	P	T	F	R	N		Next Entry Loc.
	1	0							3
	1	1							3
	1	0							6
	0	Ø	P	T	F	R	N		Next Entry Loc.

0 = No Register Storage
1 = Register Storage

0 = Return to program after ITC Timer or Drum/Disc Transfer Complete Interrupts
1 = Return to ECP after ITC Timer or Drum/Disc Transfer Complete Interrupts

For programs sharing full register storage, the index would be the same.

Register Storage Table (REGSTG) stored in permanent core.



	23	22	21	20	19	18	17	16	15 0
AREG	Register Contents at Next Entry								
QREG									
*PREG	0	Ø	P	T	F	R	N	0	Next Entry Address
X3REG									
X4REG									
X5REG									
X6REG									
X7REG									
	Third 8-Word Block of Register Storage								
AREG+32	Fourth 8-Word Block of Register Storage								
X7REG+32									
A-REG+56	Seventh 8-Word Block of Register Storage								
X7REG+56									

*PREG Flip-Flops
Ø = 1 - Set Overflow; 0 - Reset
P = 1 - Set Permit Interrupt; 0 - Reset
T = 1 - Set Test Flip-Flop; 0 - Reset
F = 1 - Set Memory Fence; 0 - Reset (AU2)
R = 1 - Absolute Permanent Core Location
0 - Relative Address (Drum/Core)
For an all-core system, R is always zero.
N = 1 - Negative Relative Entry Address (Bits 0-15)
0 - Positive Relative Entry Address (Bits 0-15)
For an all-core system, N is always zero.

The other tables assisting the ECP in performing its functions are:

PRØG - Program Execution Time Table

TIME | 00001433 |

Program #			
1	PRØG	40000000	PROGRAM OFF
2		00001342	PROGRAM CURRENTLY RUNNING
3		00001465	PROGRAM DELAYED
4		40000001	LOCKED OUT
5		40000000	OFF
6		40000000	OFF
7		00000000	PROGRAM ON (Based on event)

Drum Transfer Control Table

		23	22	21	20	19	18 17 15	13 0
Program No. 0 Transfer Group	DRMLØC	0	BEGINNING DRUM/DISC ADDRESS					
	SIZE	A	C	T	N	S	INDEX TO 3-WORD GROUP IN SAVE TABLE	NUMBER OF WORDS IN PROGRAM (SAVED BLOCK NOT INCLUDED)
	CØRLØC							BEGINNING CORE ADDRESS
	DRMLØC+3 SIZE+3 CØRLØC+3	Drum Transfer Control Group for Program #1						
	CØRLØC+3n	Drum Transfer Control Group for Program #n						

A - Area Availability on Entry From ECP
 1 = Available, 0 = Unavailable
C - 1 = Program is in core
 0 = Program is not in core
T - 1 = Program is in transfer from drum/core
 0 = Program is not in transfer
N - 1 = Program is running with core area available
 0 = Program is running with core area unavailable
S - 1 = Save temporary storage on drum if overwritten
 0 = Do not save temporary storage

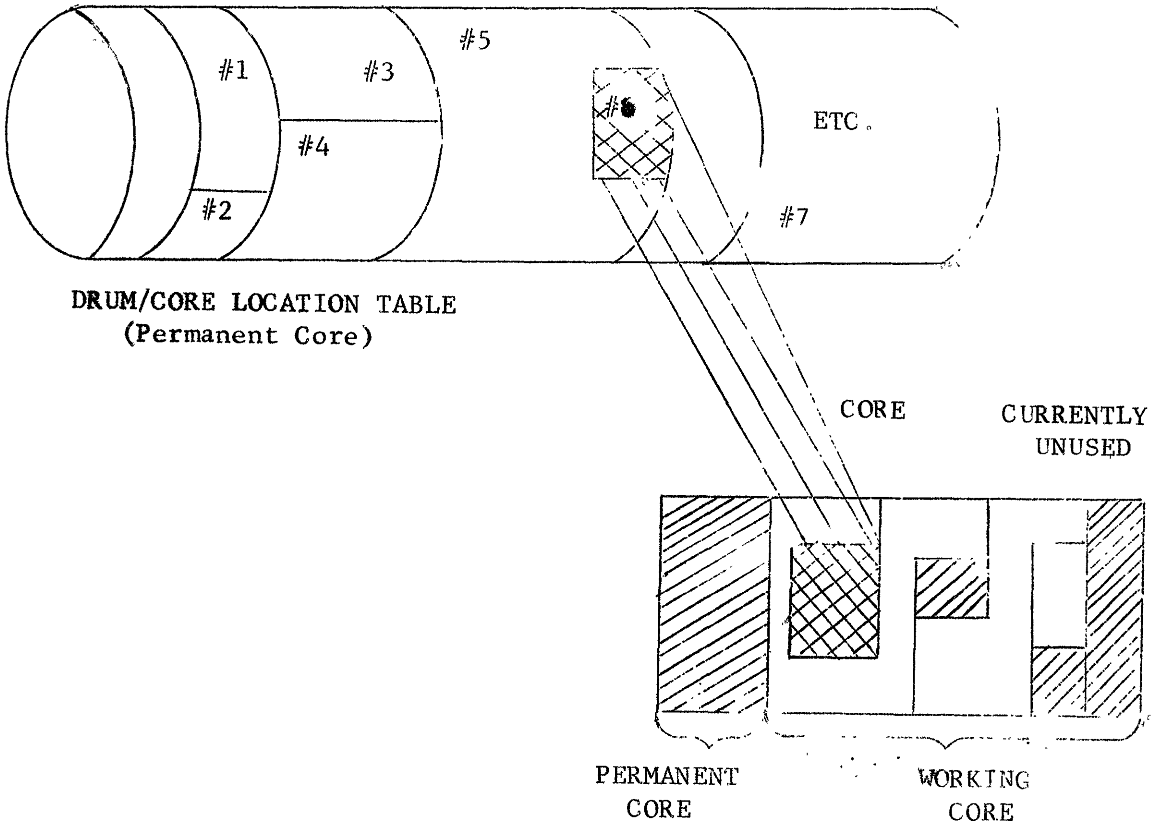
The Save Status Area Control Table is used for programs requiring temporary storage to be saved in an unprotected area on drum before overwriting. This feature is called "Save Status". The ECP transfers the temporary storage of programs having "Save Status" to drum before another program is transferred in its place. However, when a functional program is turned off, its temporary storage is not automatically saved on drum.

Save Status is specified in the Drum Transfer Group Table above in the SIZE Word, Bit 19.

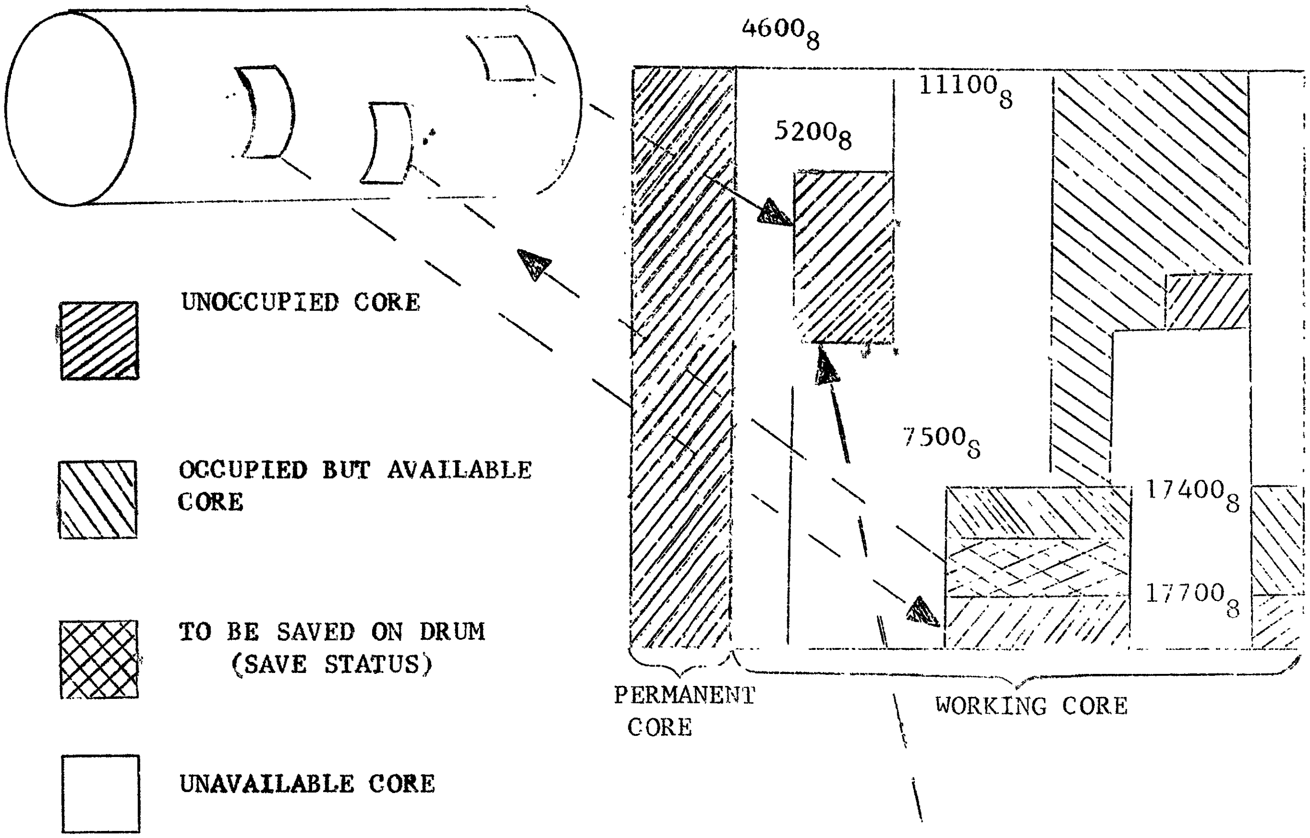
Save Status Area Control Table

	23	22	15	13	0
SAVTBL	0	BEGINNING DRUM OR DISC ADDRESS FOR SAVED AREA			
SVSIZE				NUMBER OF WORDS IN SAVED BLOCK AT END OF PROGRAM	
SVLØC			BEGINNING CORE ADDRESS		

ECP Drum/Core Communications



ECP Drum/Core Communications



Each bit in the following two tables represents 64 core locations (100_8).

OCCUPIED AREA MAP

CØRMAP	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1

UNUSED

1 = Core Area Occupied by a Functional Program

AVAILABLE AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AVLMAP	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

UNUSED

1 = Core Area Unavailable for Overwriting

1.2 Time and Diagnostic Count Driver (ITC)

ITC uses two interrupts to control the system timing. The first (non-inhibitable) interrupt occurs each 16 2/3 MS. On interrupt, a DMT counter is decreased by one. When the counter equals minus one, the second (inhibitable) interrupt is triggered.

The second interrupt causes entry to ITC. The DMT Counter is initialized by storing the number of 16 2/3 MS intervals in a time count. This number, NCYCLE, must be evenly divisible into sixty and is specified by the System Programmer at assembly time. NCYCLE determines the length of time represented by one time count (1 second, 1/2 second, 1/4 second, etc.). ITC increases the time of day (TIME) by one. The time of day is cleared at midnight. ITC also references all program and auxiliary time counters at the beginning of each day. Auxiliary Time Counters are used by functional programs which are turned on by the Monitor Initialize Routine which require initiation at set time intervals (1 minute, 2 minutes, etc.). Typical programs would be the Scan or Performance Calculations.

ITC also tests for interrupt driven device failures on the Peripheral Buffer, Output Distributor, Scanner, etc. Each device is assigned a specified COUNT which is used for counting the time required for activating a particular device. This location (COUNT) represents the maximum number of time count intervals in which action should be completed. If a COUNT becomes negative, the device has failed and the Corrective Action Program is turned on.

When an interrupt occurs during the execution of an interruptable system subroutine, ITC returns immediately to the interrupted subroutine.

ITC returns control to the interrupted program or to the ECP depending on the indicator set in the Register Storage Table (RSX).

1.3 Save Registers Subroutine (SRG)

SRG saves the register contents and/or next entry location for a functional program. For functional programs having full register storage or sharing register storage, the P Counter, A, Q, and X3 through X7 Registers for the interrupted program are transferred to the Register Storage Table. Otherwise, only the next entry point to the functional program is saved.

1.4 Restore Registers Subroutine (RRG)

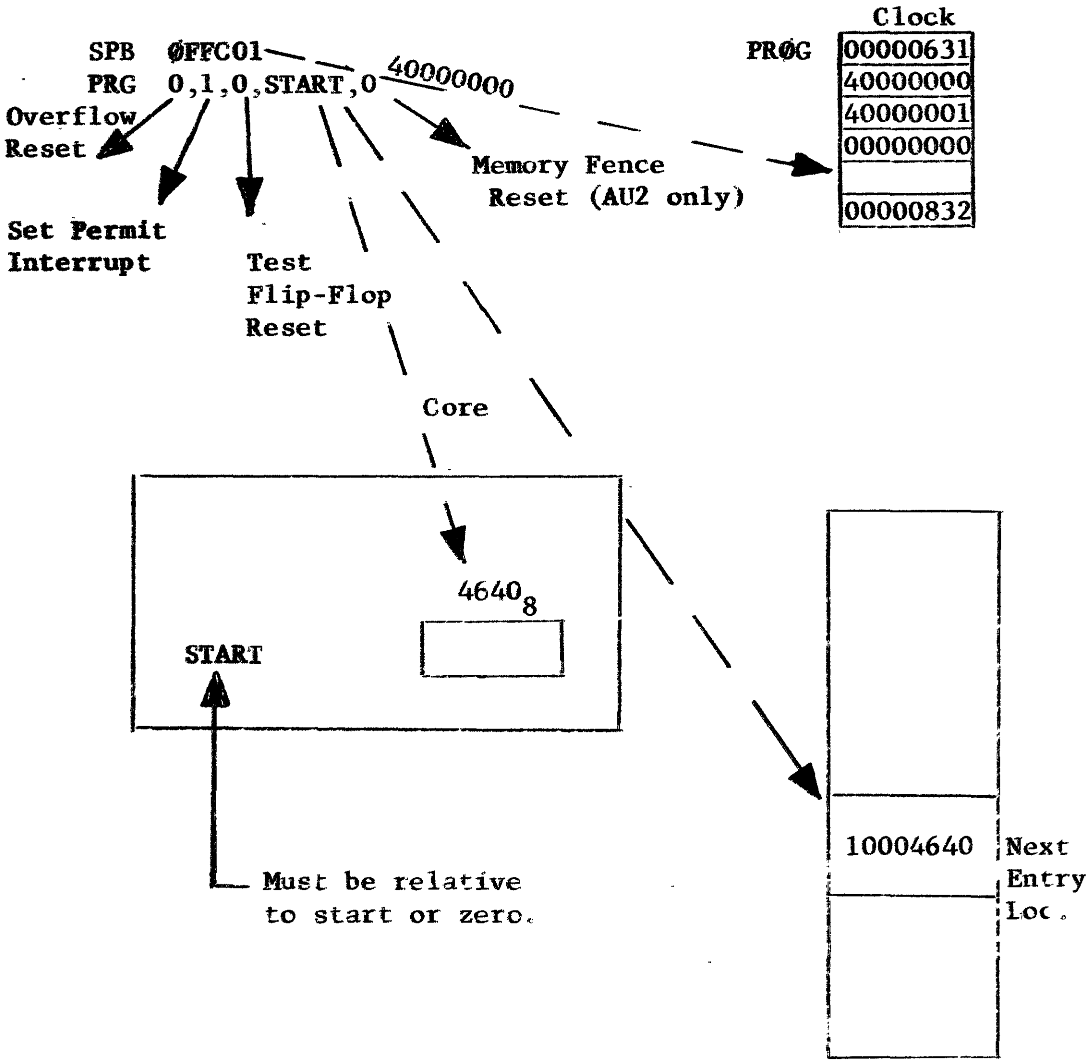
RRG returns to an interrupted program at the designated entry point. The contents of register storage are transferred to the register locations for programs having full register storage. These values represent the contents of the various registers for the running program at the time of its last interrupt.

1.5 Turn Program Off Subroutine (ØFF)

The Turn Program Off Subroutine stops the ECP from initiating the execution of functional programs. Programs are turned off by placing the "off" constant, 40000000_8, as the next execution time. Only a running program may turn itself off.

After the "off" constant is stored, control is transferred to the ECP.

To communicate with ØFF, use the following calling sequence:



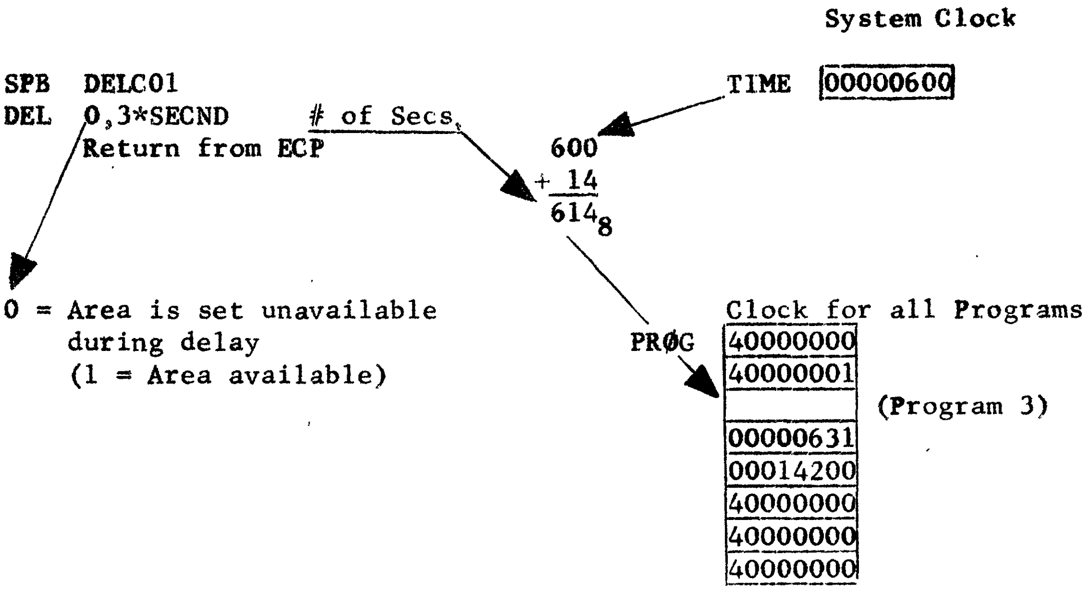
```
SPB   ØFFC01
PRG   0,1,0,START,0
```

1.6 Set Program Delay Subroutine (DEL)

DEL delays the execution of a functional program for a specified time period. The delay, in time counts, is added to the current time and stored in PRØG$_n$ for the calling function.

The A, Q and X3 through X7 Registers are saved for those programs having register storage.

To set a delay, use the following calling sequence:



```
SPB  DELC01
DEL  0,3*SECND      # of Secs.
     Return from ECP
```

0 = Area is set unavailable
during delay
(1 = Area available)

System Clock

TIME 00000600

$$\begin{array}{r} 600 \\ +\ 14 \\ \hline 614_8 \end{array}$$

PRØG

Clock for all Programs

```
40000000
40000001
                (Program 3)
00000631
00014200
40000000
40000000
40000000
```

The above example shows a 1/4 second system.

1.7 Turn Program on Subroutine (TPN)

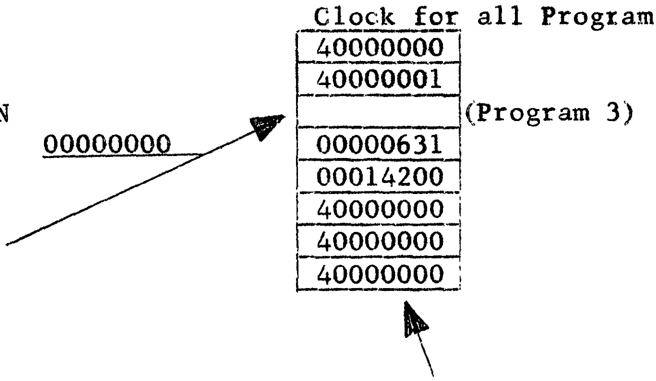
The Turn Program On Subroutine is used to change the execution time of functional programs. The execution time and the program number are given in the calling sequence.

After the new execution time is stored for a program, control is returned to the calling function.

Programs which are "locked out", next time of execution 40000001_8, may not be turned on by TPN.

TURN PROGRAM 3 ON

```
     LDZ
     SPB  TPNC01
     CØN  G,HLØG
          RETURN

HLØG EQL  3
```



Clock for all Programs	
40000000	
40000001	
	(Program 3)
00000631	
00014200	
40000000	
40000000	
40000000	

TO TURN A PROGRAM ON ONLY IF IT IS CURRENTLY OFF

```
     LDA  Execution Time or LDZ
     SPB  TPNC02
     CØN  G,HSCAN
          RETURN

HSCAN EQL  8
```

Program 8 is turned on.

TPN returns with all ones in the A-Register when a request is made to turn a program on which is "locked out".

1.8 Map Maintenance Subroutine (MAP)

MAP is used to update the core map tables (CØRMAP and AVLMAP). Core areas may be set occupied, unoccupied, available, or unavailable:

```
SPB  MAP01 - Set Area Occupied or
SPB  MAP02 - Set Area Unoccupied or
SPB  MAP03 - Set Area Unavailable or
SPB  MAP04 - Set Area Available
     Return to the calling program
```

Occupied Core Area Map Table

CØRMAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CØRMAP	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

In this table, "one" bits indicate that the core area represented is **occupied** by a functional program whether its area is available or not.

Available Core Area Map Table

AVLMAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AVLMAP	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

In this table, "one" bits indicate that the core area is unavailable for overwriting.

Each bit represents 2^K words of working core area. For example, if k = 5, each bit represents 32 words of core. The working core area starts at fixed location such as 12000_8.

In our example, 5 programs are in core occupying areas:

12000_8	- 12337_8	(word 1, bits 0 - 6)
12540_8	- 13277_8	(word 1, bits 11 - 21)
13600_8	- 14737_8	(word 2, bits 4 - 22)
16500_8	- 17337_8	(word 4, bits 2 - 14)
17400_8	- 17777_8	(word 4, bits 16 - 23)

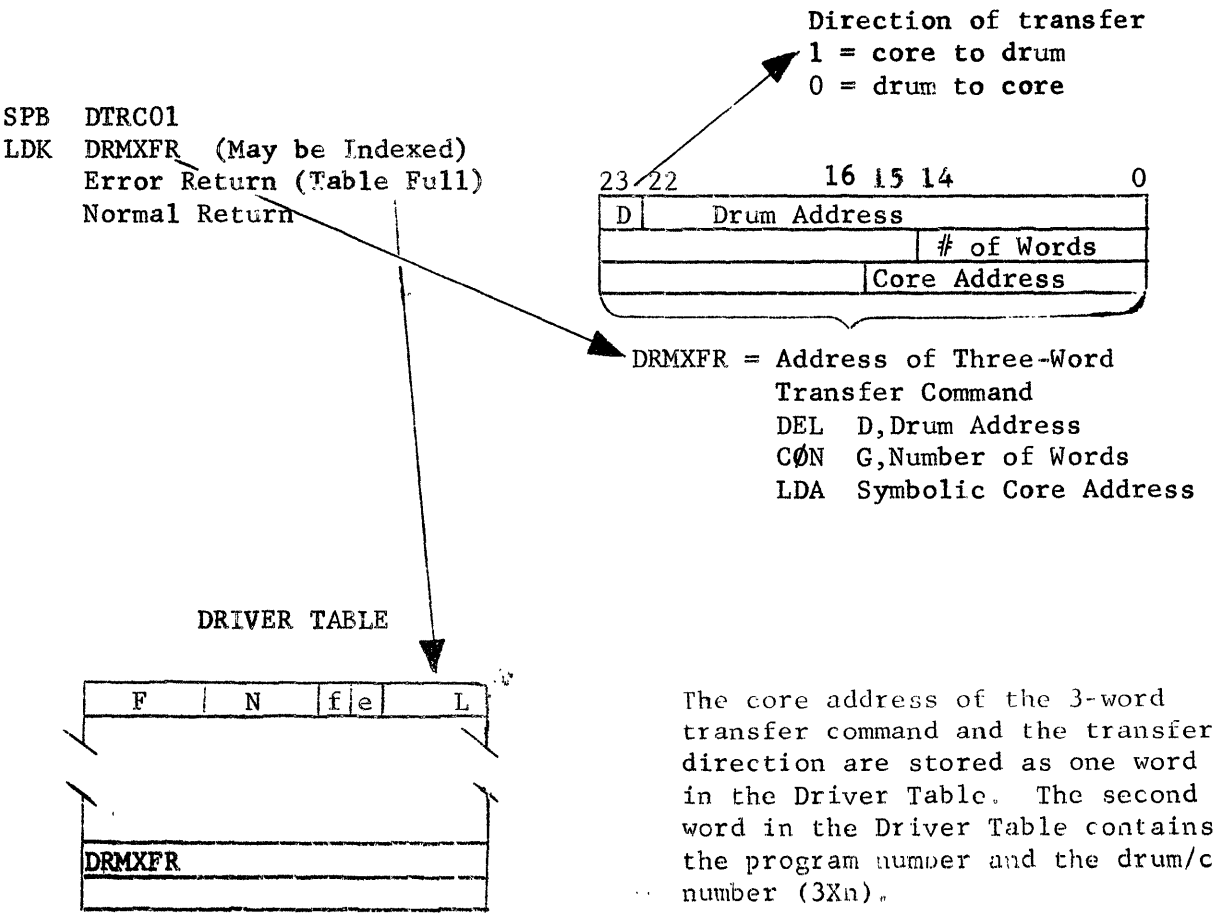
NOTE: The program in core area 12000_8 - 12337_8 is running with its core area available.

The beginning of the map table is the upper right hand bit of the table and the end is the lower left hand bit.

1.9 Drum/Disc Transfer Request Subroutine (DTR)

DTR requests transfers between drum or disc and core memory. The core address of the 3-word transfer command and the program number are stored in the Driver Table DTR may return to the calling program immediately after the request is stored or after the completion of the actual transfer.

To return to the calling program immediately after the request is stored in the Driver Table,



```
SPB  DTRC01
LDK  DRMXFR  (May be Indexed)
     Error Return (Table Full)
     Normal Return
```

```
DRMXFR = Address of Three-Word
         Transfer Command
         DEL  D,Drum Address
         CØN  G,Number of Words
         LDA  Symbolic Core Address
```

The core address of the 3-word transfer command and the transfer direction are stored as one word in the Driver Table. The second word in the Driver Table contains the program number and the drum/core number (3Xn).

To return to the calling program after the requested transfer has been completed,

```
SPB  DTRC02
LDK  DRMXFR (May be Indexed)
     Error Return (Driver Table Full)
     Normal Return
```

1.10 Drum/Disc Transfer Driver (DTD)

DTD initiates transfers between core memory and drum or disc. It is entered from the Drum Transfer Complete Interrupt.

Following each interrupt, a transfer is initiated by an ØUT Drum/Disc Command. Requests from the Scan Driver are given priority. Return from DTD is to the interrupted program or the ECP. (Refer to 1.2).

1.11 Find Register Pointer Subroutine (FRP)

FRP determines a program's type of register storage. It gives the starting address of the 8-word storage block or the next entry address for the specified program.

```
LDA  Program Number
SPB  FRPC01
     Returns with the address in the A-Register
     and Test Flip-Flop Status
```

The test flip-flop is set if the program has full register storage. Otherwise, it is reset.

See Register Pointer Table under 1.1.

1.12 Peripheral Availability Subroutine (PAV)

The Peripheral Availability Subroutine is used as a combination call to check for peripheral availability, make an output request, or perform peripheral substitution. When a peripheral is not free, appropriate diagnostic action is taken or peripheral substitution is made if the peripheral is bad or out-of-service.

PAV also provides an optional feature for checking data areas. A data area is assigned to each output message representing the drum, disc, or permanent core area containing the message. When an output request is made for availability and the area is available, the area is set unavailable and is set free when the last character of the output message is completed.

Data area numbers range from 1-24. Zero is reserved as a dummy area number when availability is not required. Area number zero is always considered available. The data number is used as a tag to identify a message area. The tag may apply to an individual message or a buffer which that message occupies. Two messages which share the same buffer may not have the same tag. Until the driver removes the data tag, no other program can use this tag.

A peripheral availability call must not occur in a subroutine used by more than one functional program. The PAVC01 call selects an alternate output device when the requested device is not free.

```
         SPB   PAVC01     (Fixed Message Call)
        {DEL   0,FØRMAT
        {DEL   0,DATA                 No Test for Data Area Required
         FØR   DEVICE,0 ---------->      0 = Dummy Area Number
               RETURN

Core Format
and Data
Locations
```

REASON DEVICE IS UNAVAILABLE	ACTION
Stacking Table Full	Delay 1/2 Sec. and try again.
Device is Bad or Out-of-Service	Request output on alternate device.
No Alternate is Available	Bypasses output request.

Peripheral Availability (contd.) Message Number

```
        SPB   PAVC02               (Checks stacking table and data area
        FØR   DEVICE,AREANØ         availability)
                  .
                  .
                  .
       Place the Output
       Data in the Table                     Return is made when the
                                             area is available and the
                                             device's stacking table
                                             is not full.
                  .
                  .
        SPB   PAVC03               (Requests output and places request in
        SPB   ØUTC02                Stacking Table)
        DEL   1,FØRMAT
        DEL   0,DATA
        FØR   DEVICE,AREANØ
              RETURN

Drum          Core
Address       Address
```

<u>To Place a Message at the Top of the Stacking Table</u>
(Priority Request)

```
        SPB   PAVC04
        DEL   0,FØRMAT
        DEL   0,DATA
        FØR   LGTYPE, 0      Area Number
              Return

              Device Priority Number

Core Addresses
of Format and
Data Locations
```

```
LGTYPE   EQL 2
```

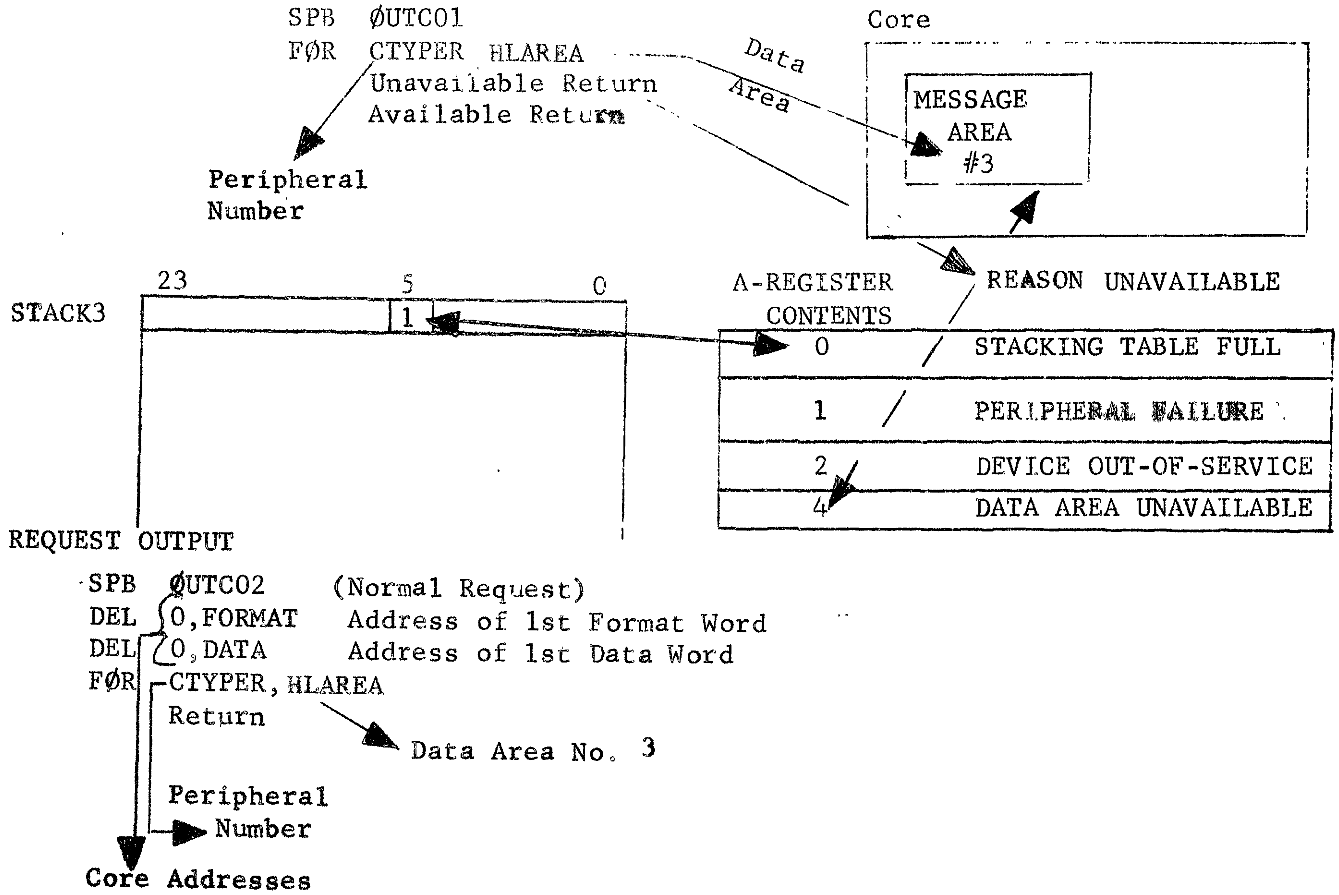
1.13 Output Subroutines (ØUT)

ØUT is composed of three functions. The ØUTC01 call is used to test for peripheral availability and data area status. The ØUTC02 (Normal Request) and ØUTC03 (Priority Request) calls make the output request. The given data and format locations are stored in the proper tables and the peripheral ALERT bit is set for the Output Program.

When the output request returns to the "available return" (data area and peripheral free), the system program may then assemble its data in the data area in preparation for the actual output request. For every "available return", there must be an associated output request.

Output Requests may be made for the printer, typewriter (including I/O), paper tape and card punches, and magnetic tape.

To Test Peripheral Availability and Data Area Status,



```
SPB  ØUTC01
FØR  CTYPER HLAREA
     Unavailable Return
     Available Return
```

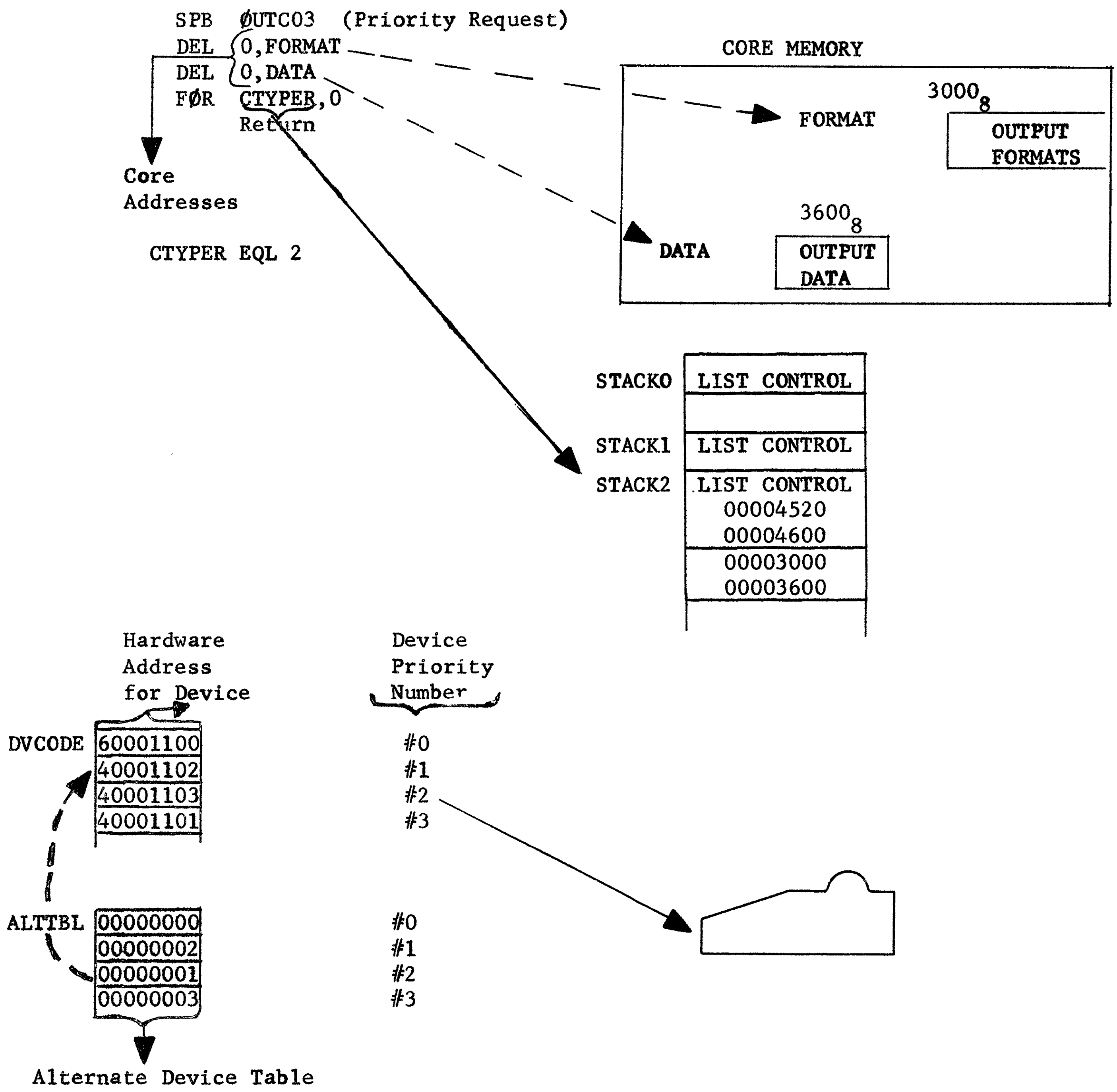
A-REGISTER CONTENTS	REASON UNAVAILABLE
0	STACKING TABLE FULL
1	PERIPHERAL FAILURE
2	DEVICE OUT-OF-SERVICE
4	DATA AREA UNAVAILABLE

REQUEST OUTPUT

```
SPB  ØUTC02      (Normal Request)
DEL  0,FORMAT    Address of 1st Format Word
DEL  0,DATA      Address of 1st Data Word
FØR  CTYPER, HLAREA
     Return
```

HLAREA EQL 3

Output Tables



```
SPB   ØUTC03   (Priority Request)
DEL   0,FORMAT
DEL   0,DATA
FØR   CTYPER,0
      Return

CTYPER EQL 2
```

	Hardware Address for Device	Device Priority Number
DVCODE	60001100	#0
	40001102	#1
	40001103	#2
	40001101	#3
ALTTBL	00000000	#0
	00000002	#1
	00000001	#2
	00000003	#3

STACK0	LIST CONTROL
STACK1	LIST CONTROL
STACK2	LIST CONTROL 00004520 00004600
	00003000 00003600

Format and data words must be stored on drum or in permanent core. Format words and data words are stored in separate tables. Addresses given in the ØUTC02 and ØUTC03 calls must be absolute symbols. Either common (*) or local absolute (-) may be used.

1.14 Output Program (IØP)

Monitor outputs information by using a group of related routines and subroutines within the Output Program. The Output Program is turned on after receiving an output request from the Output Subroutines.

The stored format word is decoded into separate fields. Format words which are stored on drum or disc are transferred to an eight-word core buffer for processing. Data which is stored on drum or disc is placed in a sixteen-word core buffer. Both buffers are within the Output Program's temporary storage (save) area. The proper conversion routine, as indicated in Bits 23-21, is then entered. Each conversion routine is discussed as subcategories of 1.14:

1.14.1 Binary to Fixed-Point Decimal
1.14.2 Binary to Four-Bit BCD
1.14.3 Binary to Six-Bit BCD
1.14.4 Binary to Non-Edited Character
1.14.5 Binary to Octal
1.14.6 Clock Output
1.14.7 Floating Point to E-Type Floating Point Decimal
1.14.8 Floating Point to Fixed Point Decimal

The following subroutines are also contained within the Output Program to assist in placing the converted information in the Driver Table, printing error messages (EEE*,999*, 000*, *) etc. They are:

Store Character in Driver Table Subroutine
Store Error Code in BCD Table Subroutine
Build Driver Table Subroutine
Error Subroutines
Update Index and Load Data Subroutine

Error typeouts indicate the following:

999* - Number too large
000* - Number too small
EEE* - Incorrect format word
* - Field size less than three

The <u>End of Message</u> word is all bits set $(77777777)_8$. Every table of format words must contain an end word which is generated by:

```
CØN Ø,77777777
```

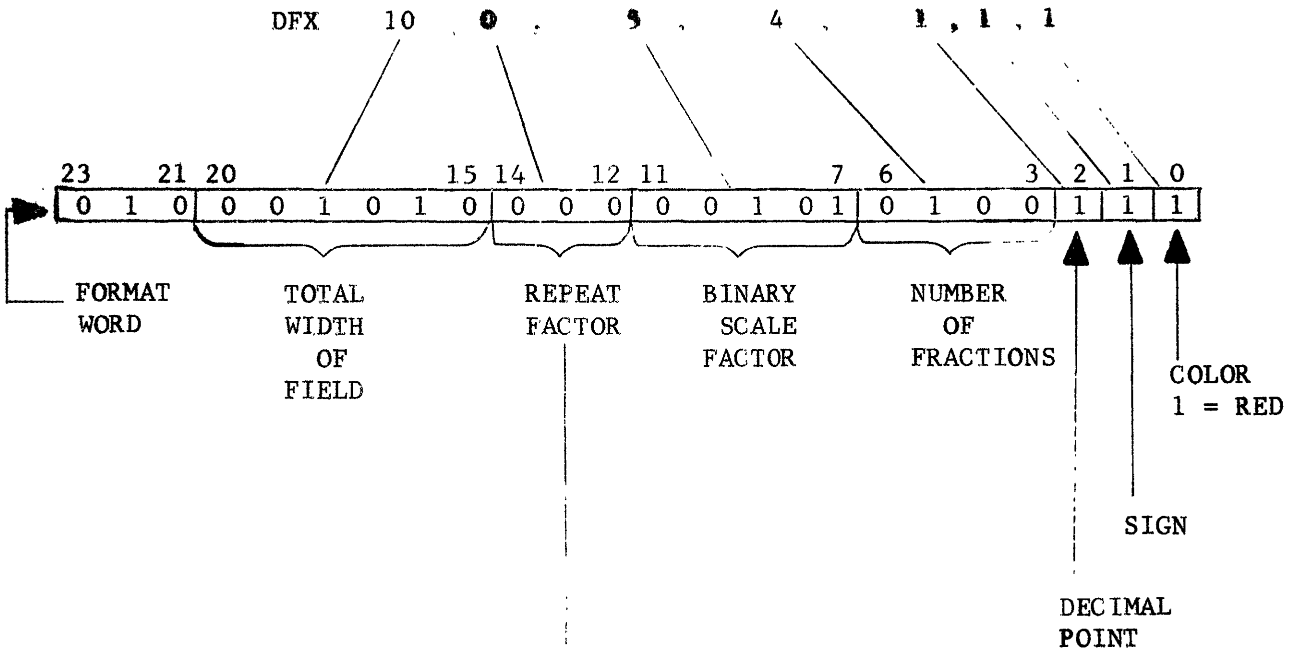
The format words shown in the following conversion routines include an example of the MONITOR Pseudo-operation. These pseudo-op instructions can only be used when assembling with GE/PAC Monitor or the GE/PAC Monitor EQL tape.

1.14.1 Binary to Fixed-Point Decimal (DFX41)

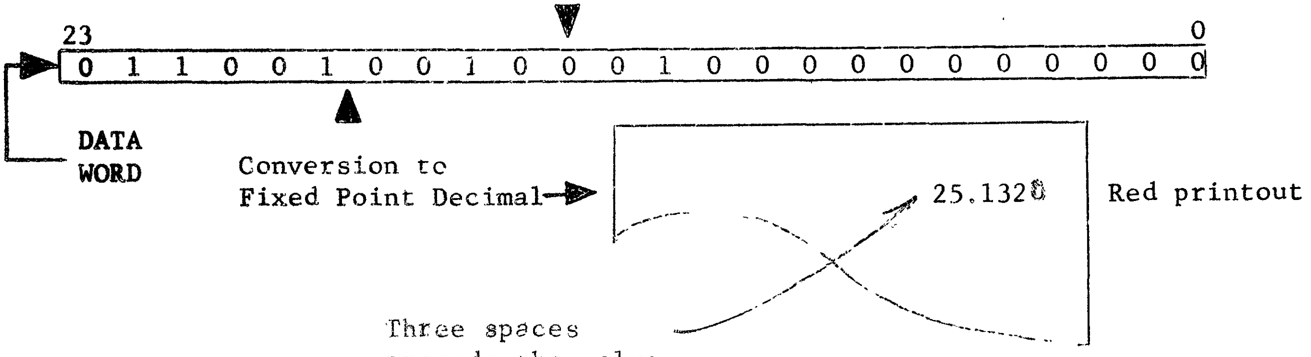
DFX41 converts information from binary fixed-point to decimal fixed-point. Leading zeros are automatically suppressed. Numbers exceeding the field size are typed as all nines followed by an asterisk.

<u>Binary to Fixed-Point Decimal</u>

Pseudo-op Example:



One value, scaled B5, is typed in a ten column field. The values are printed in red with decimal point and four fractional digits. The sign is printed if the value is negative.



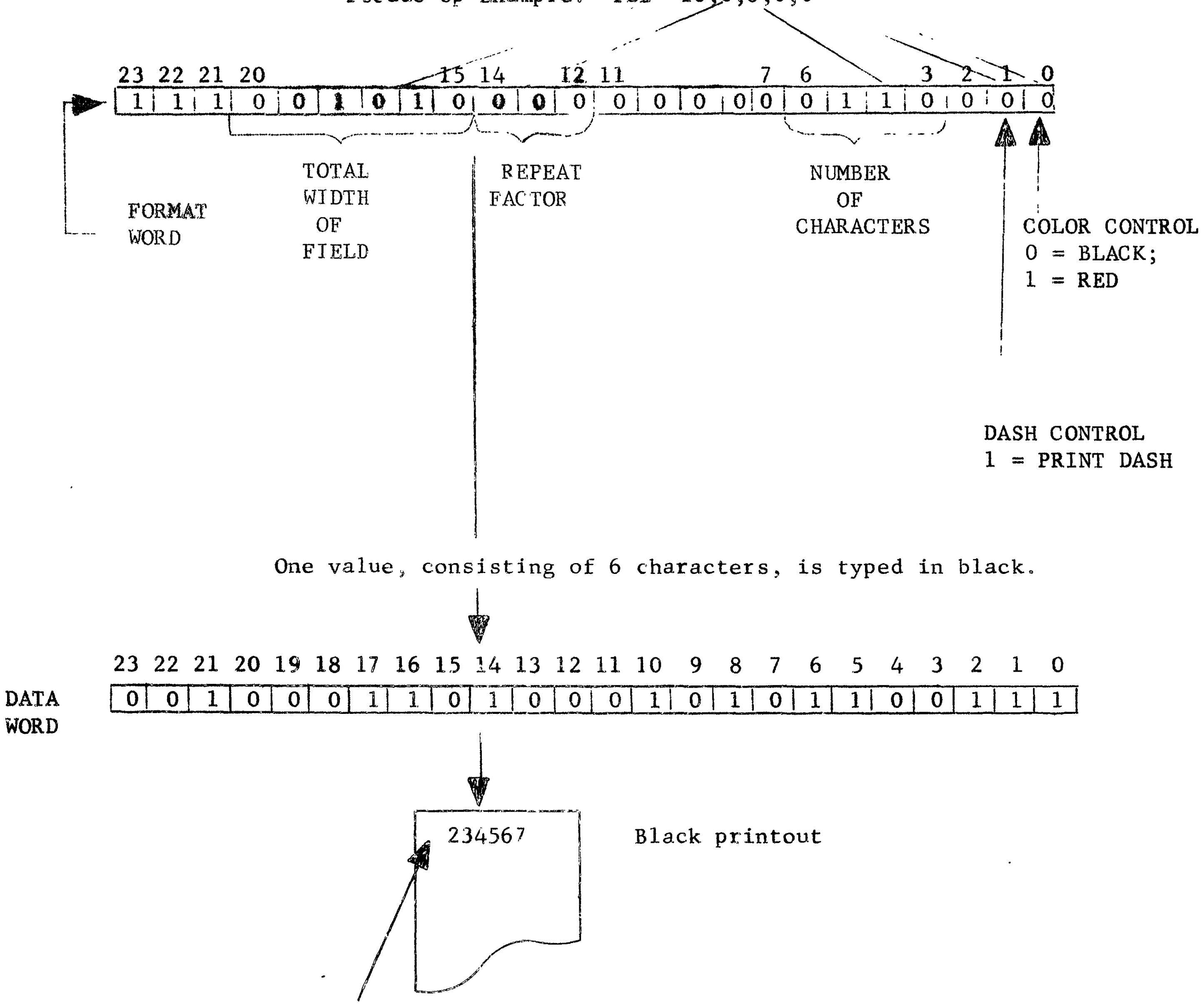
Three spaces precede the value to make a ten-column field.

1.14.2 Binary to Four-Bit BCD (FBB41)

FBB41 converts binary data, left justified, to six 4-bit BCD characters per word. After the conversion, each BCD character is stored as one word in the Output Driver Table.

<u>Binary to Four-Bit BCD</u>

Pseudo-op Example: FBB 10,0,6,0,0



23	22	21	20				15	14		12	11				7	6			3	2	1	0	
1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	

One value, consisting of 6 characters, is typed in black.

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA WORD	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1	0	1	1	0	0	1	1	1

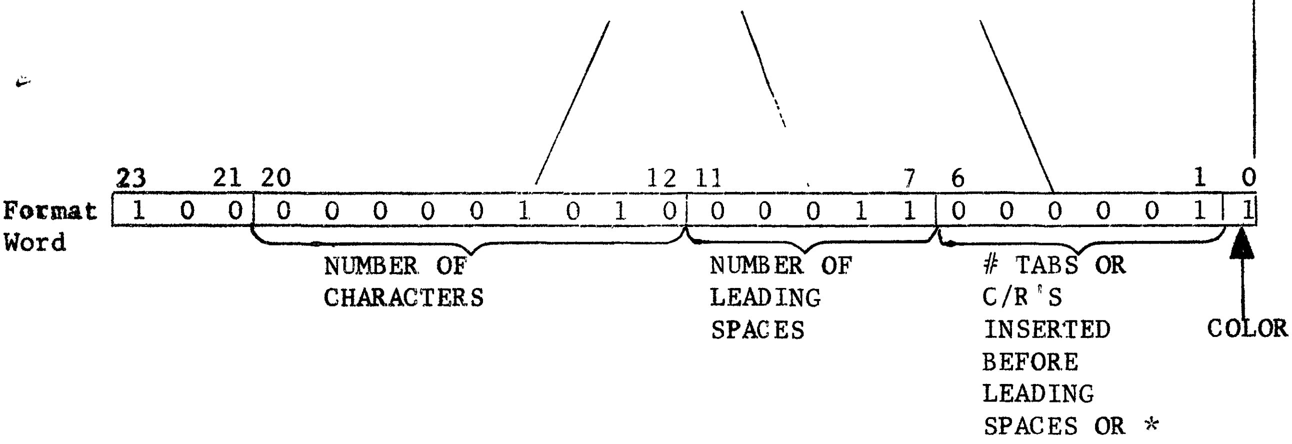
Note: If the repeat factor is used, each data word must have the same number of characters.

1.14.3 Binary to Six-Bit BCD (4 Characters Per Word)

BCD41 converts binary information to six-bit BCD characters. The data word contains six-bit BCD characters, left justified. After conversion, each word contains four BCD characters.

Binary to 6-Bit BCD

Pseudo-op Example: BCD 10 , 3 , 1 , 1

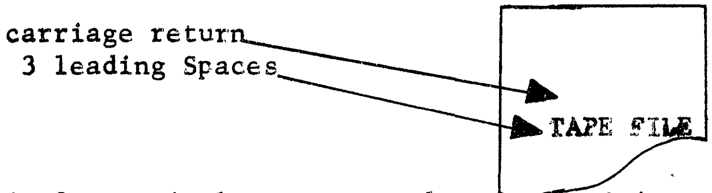


One carriage return and three spaces are made before the BCD characters are typed.

Codes 01-17$_8$ = 1-15 consecutive carriage returns. Codes 41-57$_8$ = 1-15 consecutive tabs.

CØN A,9,TAPE FILE (pseudo-op generating data words)

	23 18	17 12	11 6	5 0
Data Words	1 1 0 0 1 1	0 1 0 0 0 1	1 0 0 1 1 1	0 1 0 1 0 1
	0 1 0 0 0 0	0 1 0 1 1 0	0 1 1 0 0 1	1 0 0 0 1 1
	0 1 0 1 0 1	0 1 0 0 0 0	0 1 0 0 0 0	0 1 0 0 0 0



Alphanumeric typeout in red.

*AU2 Vertical Page Control Code for Printer

00	- No Advance
01	- Slew 1 Line (Single Space)
02	- Slew 2 Lines (Double Space)
03	- Slew 3 Lines (Triple Space)
04-07	- Page Control Unique to each System
77	- Slew to Next Page

1.14.4 Binary to Non-Edited Character (BCN41)

BCN41 stores characters (one character per word) into the Output Driver Table with no transformation. One character is contained in Bits 23-0 of each data word, right justified. The number of filled bits depends on the output device being used. The maximum number of non-edited characters permitted in one BCN format word is 63.

<u>Binary to Non-Edited Character</u>

Pseudo-op Example: BCN 12

Format Word

23		21	20					15	14	0
1	0	1	0	0	1	1	0	0	NOT USED	

No. of Characters

Table of Characters

23 — 7	6						0
0 ——— 0	1	0	1	0	0	1	1
0 ——— 0	0	1	0	1	1	0	0
0 ——— 0	0	0	0	0	0	1	0
0 ——— 0	0	1	1	1	1	0	1
0 ——— 0	0	1	0	0	0	0	1
0 ——— 0	0	0	0	1	0	0	0
0 ——— 0	0	0	0	0	0	0	0
0 ——— 0	0	0	1	1	0	1	1
0 ——— 0	0	1	0	0	1	0	1
0 ——— 0	1	0	0	0	0	1	1
0 ——— 0	0	0	0	0	1	0	0
0 ——— 0	0	1	1	1	1	1	0

Punches 12 characters with no transformation.

Paper Tape

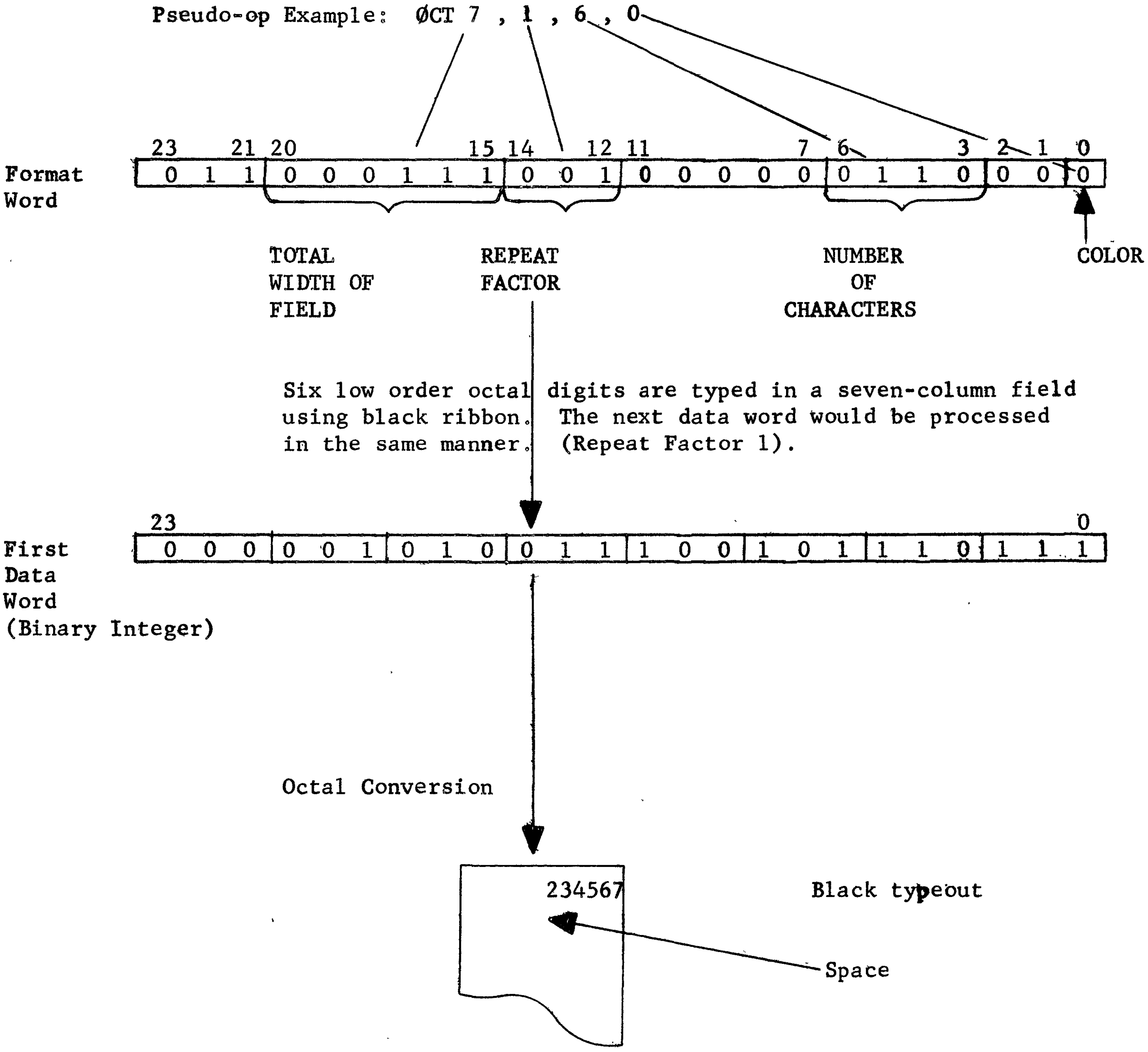
CHANNEL NUMBERS	DATA FRAMES #1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	
8	0									0			Hundreds Pos.
7		0		0	0				0			0	2nd Tens Pos.
6	0			0				0				0	2nd Tens Pos.
5	0				0		0	0					Parity Ind.
4		0		0		0		0				0	2nd Tens Pos.
●●●●●●●	●	●	●	●	●	●	●	●	●	●	●	●	Sprocket Feed
3		0		0					0		0	0	1st Units Pos.
2	0		0					0		0		0	1st Units Pos.
1	0			0	0			0	0	0			1st Units Pos.
	123	54	2	75	41	10	0	33	45	103	4	76	Pos. = Position

*Even frames require a parity punch, otherwise, a parity flip-flop is set.

1.14.5 Binary to Octal (ØCT41)

ØCT41 converts binary integers to eight octal characters.

Binary to Octal



Pseudo-op Example: ØCT 7 , 1 , 6 , 0

Format Word

```
23    21 20             15 14     12 11           7 6        3 2   1  0
 0  1  1| 0  0  0  1  1  1| 0  0  1| 0  0  0  0  0| 0  1  1  0| 0  0 | 0
```

TOTAL WIDTH OF FIELD

REPEAT FACTOR

NUMBER OF CHARACTERS

COLOR

Six low order octal digits are typed in a seven-column field using black ribbon. The next data word would be processed in the same manner. (Repeat Factor 1).

First Data Word (Binary Integer)

```
23                                                                          0
 0  0  0| 0  0  1| 0  1  0| 0  1  1| 1  0  0| 1  0  1| 1  1  0| 1  1  1
```

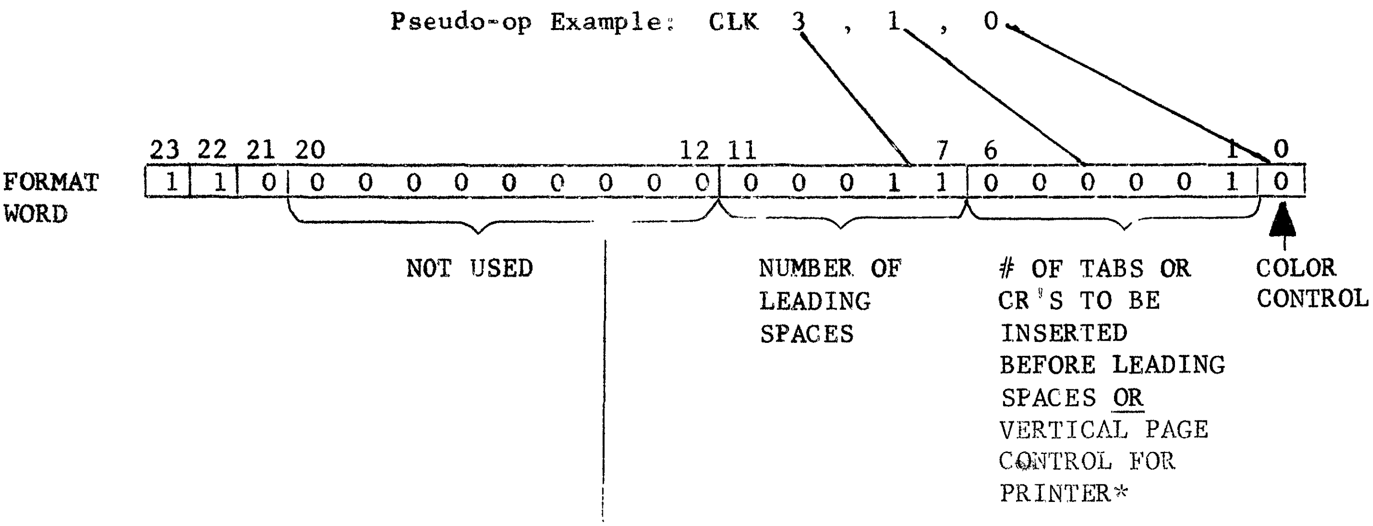
Octal Conversion

Black typeout

1.14.6 Clock Output (CLC41)

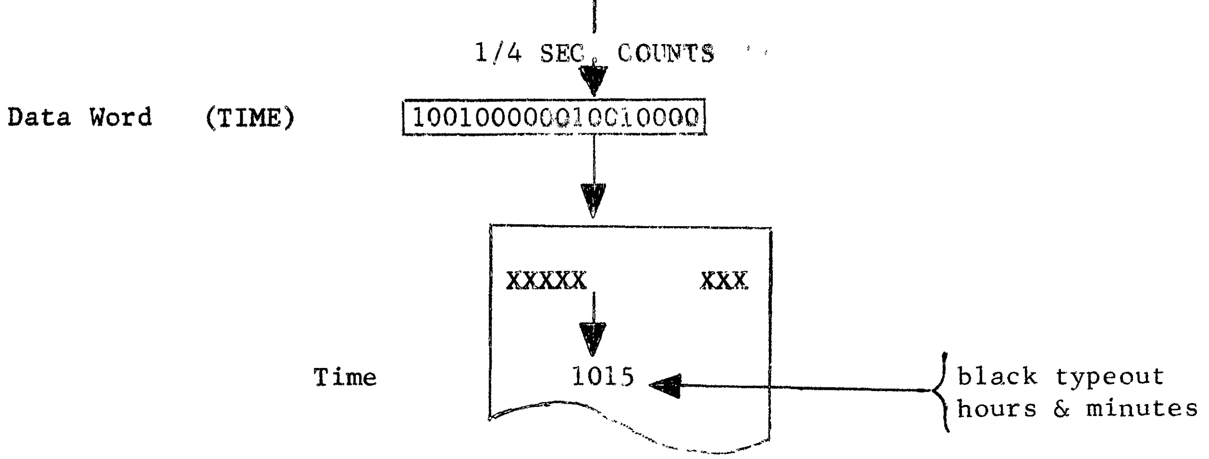
CLC41 converts the time of day from system time counts to decimal hours and minutes. It is then printed as four decimal digits.

Time Count To Hours And Minutes



Use Octal Codes 01-17 for the inserted number of consecutive carriage returns (1-15). The inserted number for 1-15 consecutive tabs are octal codes 41-57.

The above example types or punches time in hours and minutes preceded by a carriage return and three spaces using black ribbon.



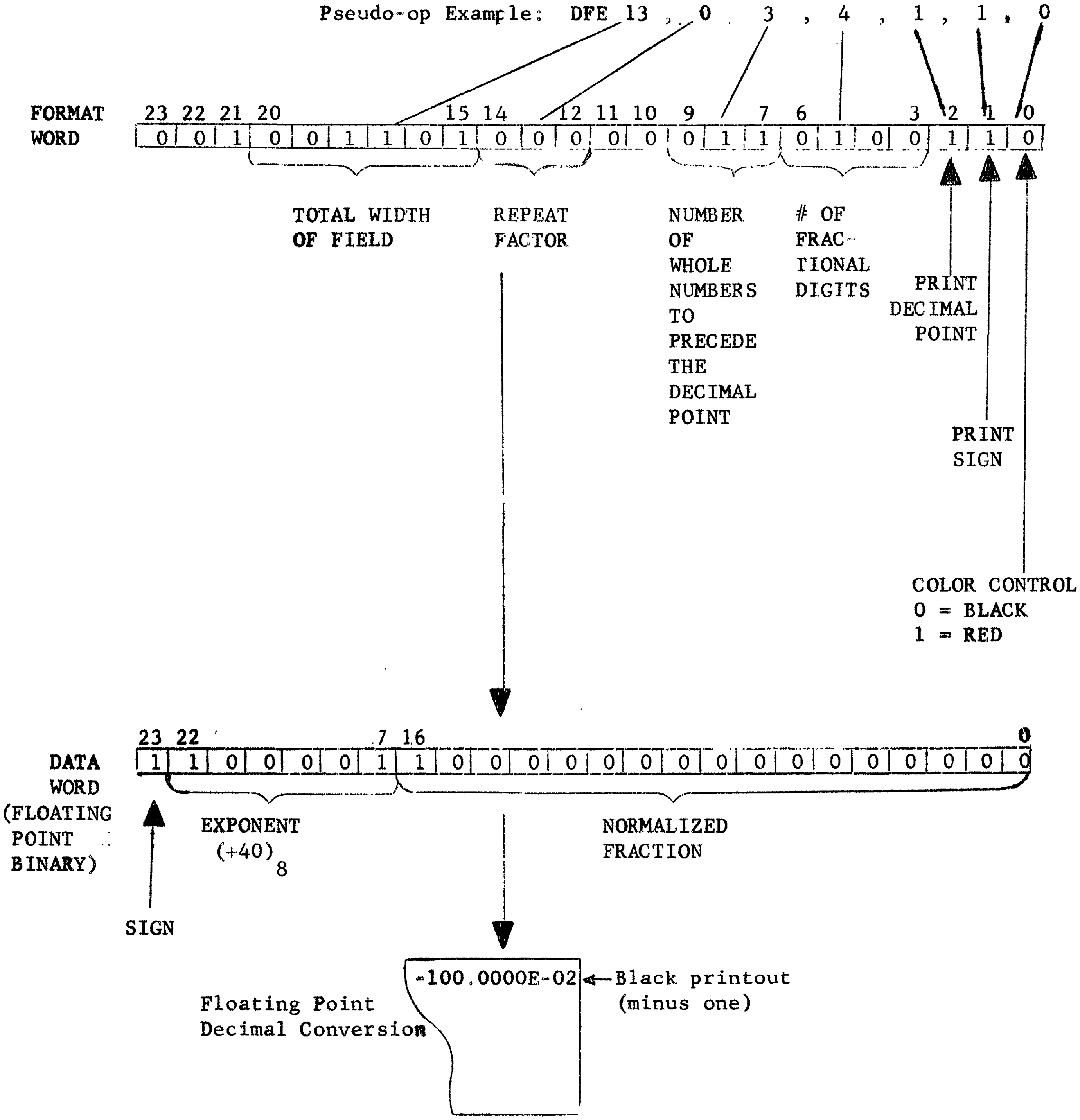
*Vertical Page Control Code for Printer

00	-	No Advance
01	-	Slew 1 Line (Single Space)
02	-	Slew 2 Lines (Double Space)
03	-	Slew 3 Lines (Triple Space)
04-07	-	Page Control Unique to each System
77	-	Slew to Next Page

1.14.7 Floating Point to E-Type Floating Point Decimal (DFE41)

DFE41 converts binary floating point information to decimal floating point.

<u>Floating Point to Floating Point Decimal</u>

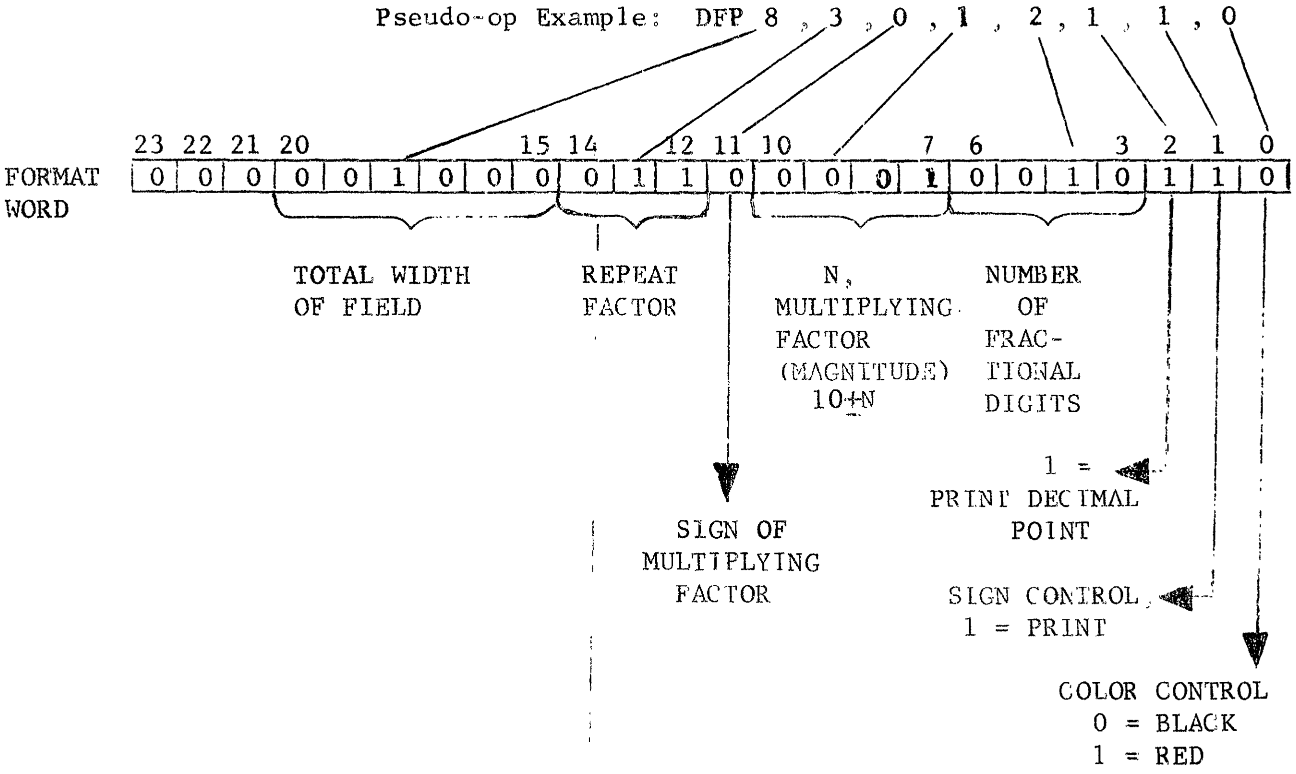


1.14.8 Floating Point to Fixed Point Decimal (DFP41)

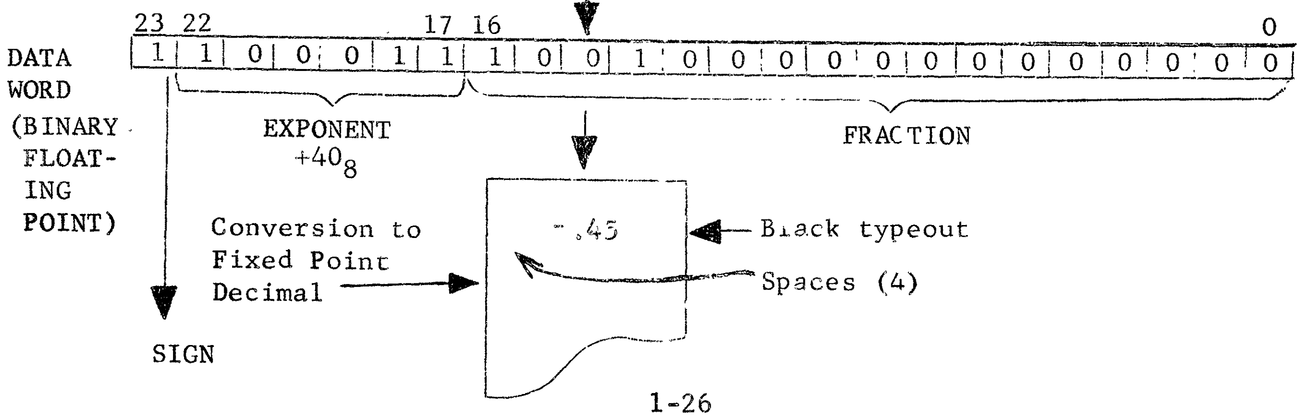
DFP41 converts binary floating-point information to decimal fixed-point. The floating point number is multiplied by 10^N if N is negative, or divided by 10^N if N is positive. (N is the multiplying factor, maximum range of absolute 9 located in the format field.)

The permitted range for any number is $< 2^{24}$ or $> 2^{-23}$

Floating Point to Fixed Point Decimal



This pseudo-operation specifies type or punch four values with multiplying factor of 10^{+2} and two fractional digits in an 8-column field. It will print the decimal point, and sign, using the black ribbon on the typewriter.



1.15 Input Driver (IND - AU2 Only)

The Input Driver is entered from any of the following interrupts:

1. I/O Device Ready
2. I/O Typer ØDL Echo
3. 350 CPM Card Reader IDL Echo
4. 350 Card Reader Ready Interrupt

IND initiates a card reader, paper tape reader, or I/O Typer request. Other functions accomplished by IND are:

1. Resets diagnostic counts for I/O Devices.
2. Tests for an Input Demand from the 350 CPM Card Reader.
3. Tests for a valid card read.
4. Notifies the Corrective Action Diagnostic Program of a photo-cell card reader error.

1.16 Input/Output Driver Program - I/O Typer (IØD)

IØD is turned on by the Input Driver (IND). Selection codes and output characters for the I/O typer are stored one character per word in the Output Driver Table. Input characters from the I/O Typer, Paper Tape Reader, or 70 CPM Card Reader, are stored one per word in the Input Data Table for the requesting program.

When the end-of-tape, stop code, requested number of characters has been read, or if the input list is full, IØD exits to the ECP.

For I/O typer input, an "IN" Command is given to unlock the keyboard. At the end of the record, IØD locks the keyboard and turns the Input Program on.

1.17 Output Driver (ØUD - AU2 only)

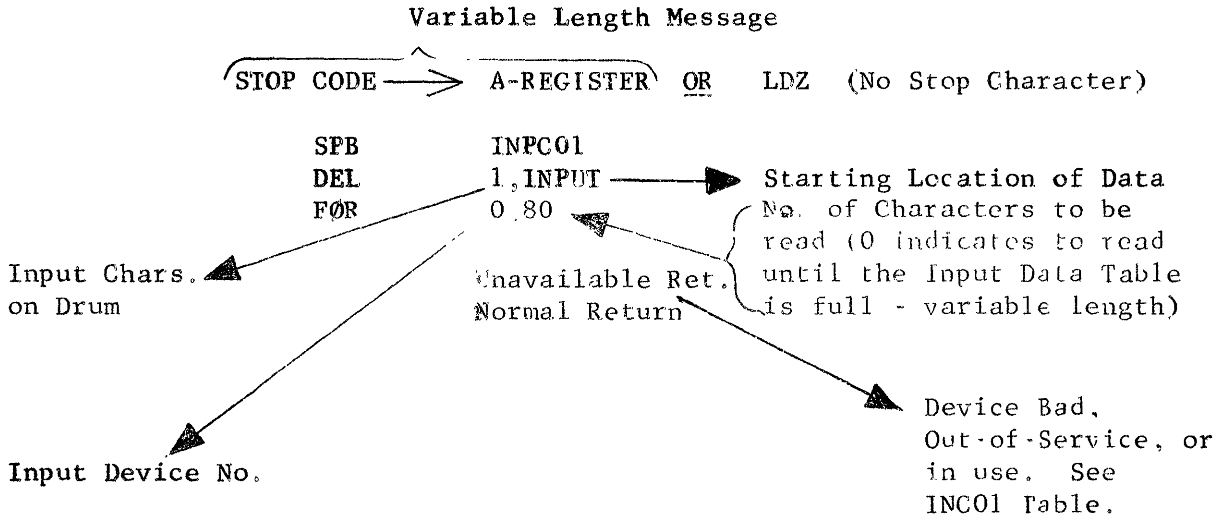
ØUD is entered from the ØDL Echo Interrupt when a peripheral devices' Driver Table is empty. Flags are reset when there is no conversion in progress for output typers, paper tape punches, card punches, or high speed printers.

1.18 Input Request Subroutine (INS)

INS accepts input requests from the paper tape reader, card reader, or I/O typewriters. Input from more than one peripheral may be requested at any time. However, each device is limited to one request at a time. Each request must be tested for completion of input by the calling function.

The following calling sequence is used for Buffered Requests. A buffered request returns immediately to the "normal return" of the calling sequence after the read is initiated. During this time, the calling function may process the characters placed in the alternate input list by the previous read request. In this manner, the calling program may process one list of characters while the Driver is storing characters in another table. (This procedure drives the input device near maximum speed.)

<u>Buffered Input Request</u>



Variable Length Message

STOP CODE → A-REGISTER OR LDZ (No Stop Character)

```
SPB     INPC01
DEL     1,INPUT
FØR     0,80
        Unavailable Ret.
        Normal Return
```

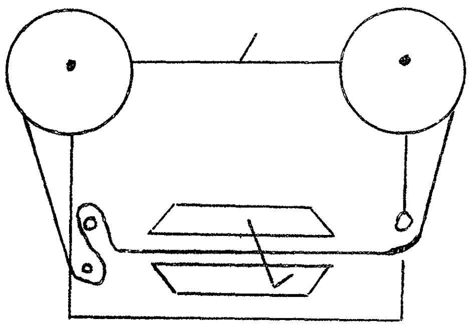
Starting Location of Data

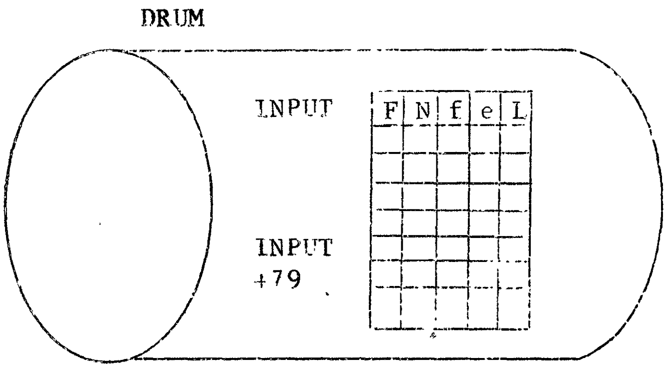
No. of Characters to be read (0 indicates to read until the Input Data Table is full - variable length)

Input Chars. on Drum

Input Device No.

Device Bad, Out-of-Service, or in use. See INC01 Table.





READ COMPLETION REQUEST

```
SPB  INPC02
CON  G, 0 <---------- Device Number
     Bad Read Return
     Read Completed Return
```

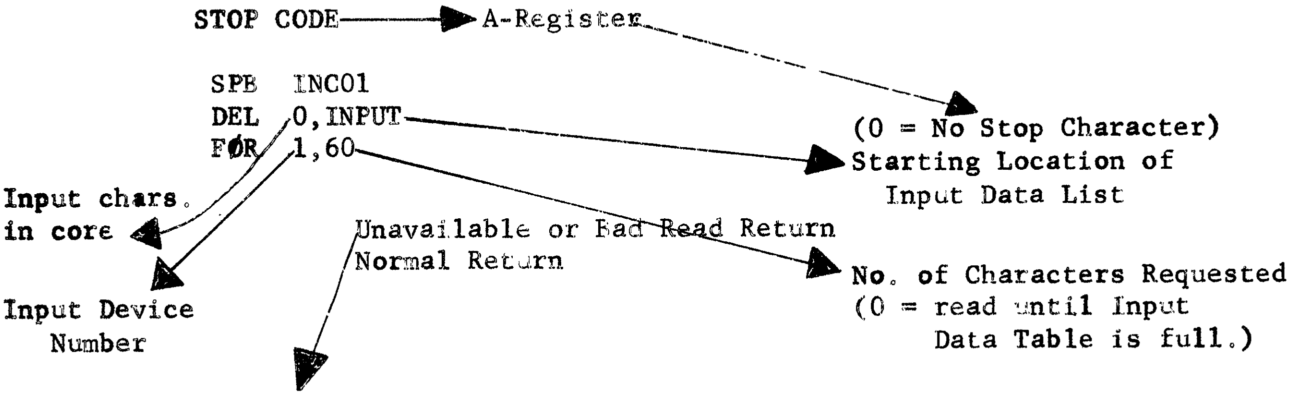
The requested number of characters has been read.
A "Stop Code" was encountered.
The end-of-tape was found.
The calling function's input data table is full.

A-REG CONTENTS	BAD READ REASON	POSSIBLE ACTION
C	Parity Error Aborted by Operator	Quit
bit 3	Device Failed During Read	Select Alternate Input Device
bit 4	Input Data Table Overflow	Remove from list before table is full
bit 5	No I/O Typer Stop Code	Quit
bit 6	I/O Typer Request Aborted	Quit
bit 7	I/O Typer Time Abort	Initiate another input request
bit 8	350 CPM Card Hopper Empty	Wait for operator to place cards in reader

1.18 Input Request Subroutine - Contd.

When a non-buffered input request is made, the calling function is locked out until the read is completed or an error is encountered.

Non-Buffered Input Request



STOP CODE → A-Register

```
 SPB  INC01
 DEL  0,INPUT
 FØR  1,60
```

(0 = No Stop Character)
Starting Location of Input Data List
Input chars. in core
Input Device Number
Unavailable or Bad ~~Read~~ Return
Normal Return
No. of Characters Requested (0 = read until Input Data Table is full.)

A-REG CONTENTS	REASON UNAVAIL.	ACTION
bit 0	Device Bad	Select Alternate
bit 1	Device Out-of-Service	Select Alternate
bit 2	Device in use	Set Delay
bit 3	Device Failed During Read	Select Alternate
bit 4	Input Data Table Overflow	Remove from list before table is full
bit 5	No I/O Typer Stop Code	Quit
bit 6	I/O Typer Request Aborted	Quit
bit 7	I/O Typer Time Abort	Initiate another input request.
bit 8	350 CPM Card Hopper Empty	Wait for operator to place cards in reader
zero	Parity Error Aborted by Operator	Quit

A flex code 177_8 acts as a true delete code and is skipped when reading paper tape. The delete code should not be used as the "stop" character.

After the first character has been read on paper tape, ten consecutive blanks serve as end-of-tape.

1.19 Input Program (INP)

After an input request has been completed, INP notifies the calling function of the completion. It also performs the following actions - resetting the flags for the input device, testing the "end" code of the input record, and notifying the calling function of an error.

The following error messages may be initiated:

1. NO STOP CODE (I/O Typers only - Bit 5 is set in the A-Register, and INP returns to the calling function at the "bad read" return.)

2. DEVICE n PARITY is typed for a paper tape reader parity error followed by the message, ENTER DATA VALUE XXX or] . The operator may correct the error by typing the appropriate three characters on the I/O Typer or abort the request by typing a right bracket.

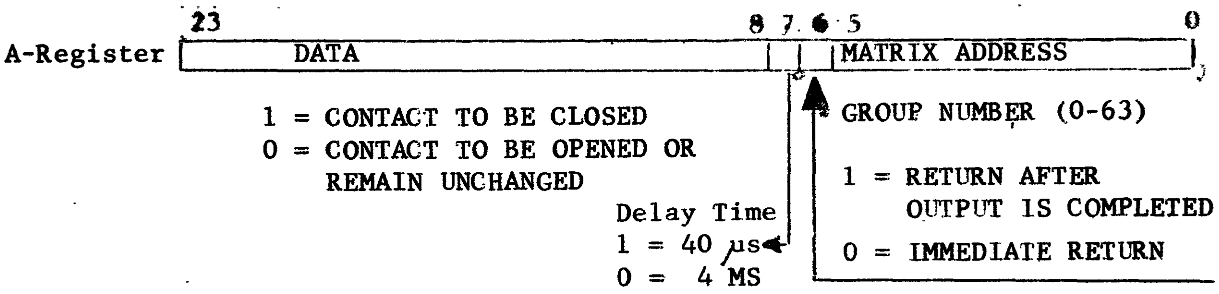
   For an abort request, INP sets the A-Register to zero and enters the calling function at the "bad read" return.

An "end" code (stop, abort, or end job character) is removed from the Input Data Table.

1.20 Multiple Output Request Subroutine (MØR)

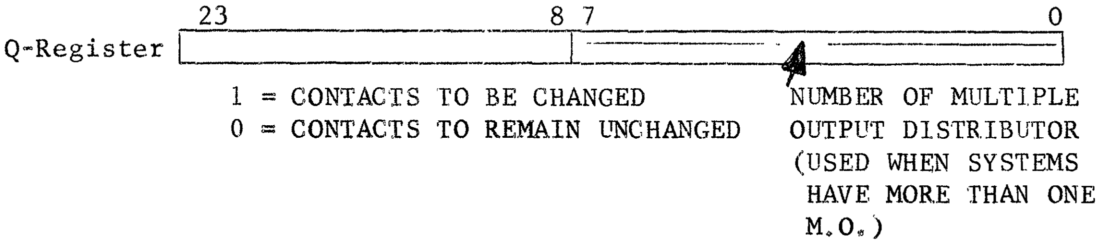
MØR permits requests for analog, decimal, or binary contact status to be made for the Multiple Output Distributor. MØR has the provision for non-timed priority or normal requests or for timed-latched contact outputs.

Non-Timed Requests



1.20 Multiple Output Request Subroutine (MØR) - contd.

Non-Timed Requests - contd.

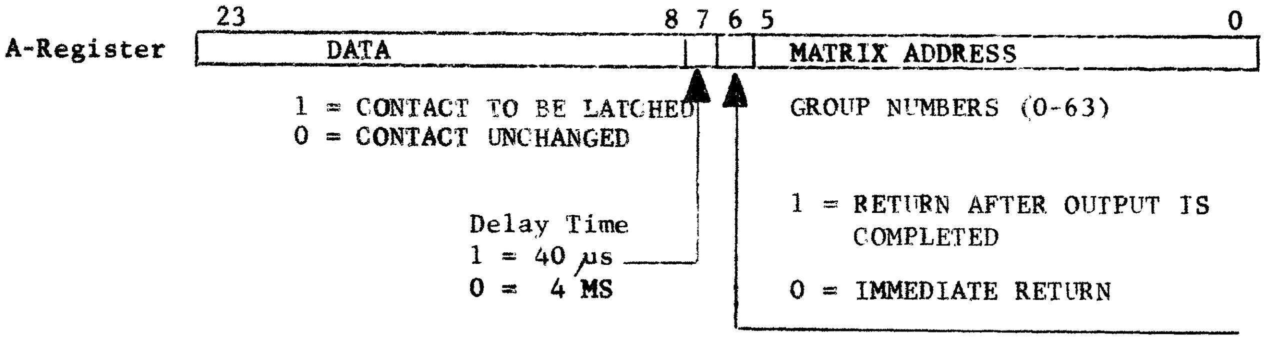

23
8 7
0
Q-Register
1 = CONTACTS TO BE CHANGED
0 = CONTACTS TO REMAIN UNCHANGED
NUMBER OF MULTIPLE OUTPUT DISTRIBUTOR (USED WHEN SYSTEMS HAVE MORE THAN ONE M.O.)

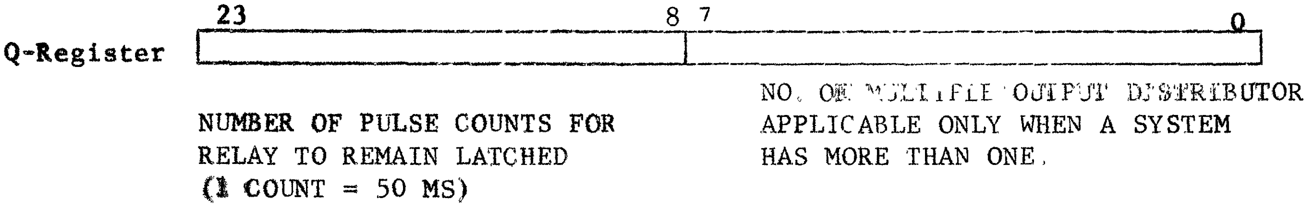
SPB MØRC01 (Normal Request) or SPB MØRC02 (Priority Request)

ERROR RETURN
NORMAL RETURN

A-REGISTER	ERROR	POSSIBLE SYSTEM ACTION
0	DRIVER TABLE FULL	Delay and try again.
1	GROUP FAILED PREVIOUSLY	Quit - try next output.
2	OUTPUT FAILED (OVERLOAD)	Hardware condition (2 groups addressed
4	INVALID GROUP ADDRESS	Programming Error
8	M.O. TIMER FAILED	No output may be done.

Timed Request




Q-Register
23
8
7
0
NUMBER OF PULSE COUNTS FOR RELAY TO REMAIN LATCHED (1 COUNT = 50 MS)
NO. OF MULTIPLE OUTPUT DISTRIBUTOR APPLICABLE ONLY WHEN A SYSTEM HAS MORE THAN ONE.

SPB MØRC03 (Normal Request) or SPB MØRC04 (Priority Request)

ERROR RETURN
NORMAL RETURN

A-REGISTER	ERROR	POSSIBLE SYSTEM ACTION
0	NO TIMER AVAILABLE	Delay and try again.
1	GROUP FAILED PREVIOUSLY	Quit - try next output.
2	OUTPUT FAILED (OVERLOAD)	Hardware condition (2 groups addressed)
4	INVALID GROUP ADDRESS	Programming Error
8	M.O. TIMER FAILED	No output may be done.

1.21 Multiple Output Distributor Driver (MDR)

MDR is initiated by the MØD Interrupt to execute requests made by MØR. To output a request, the command word from the driver table is placed in the A-Register. The data is transferred to the location specified by the group address of the command word. The Multiple Output Distributor is then ready to accept another request.

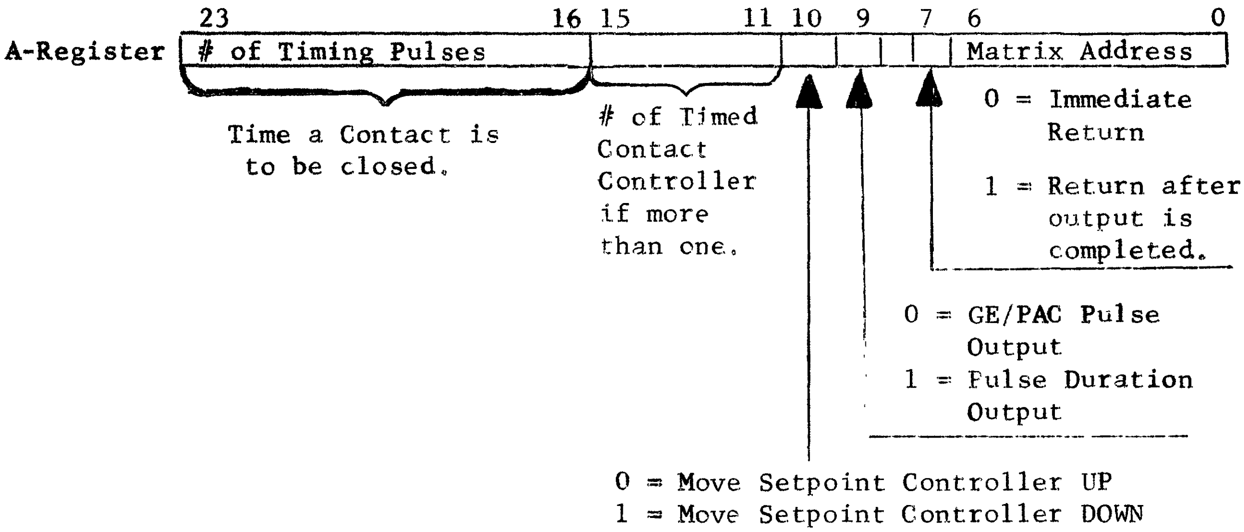
All output commands are initiated from the Priority Driver Table before information is taken from the Normal Driver Table.

1.22 Timed Contact Output Request Subroutine (TCØ)

The Timed Contact Output Request Subroutine supplies the communication link for making requests through the Timed Contact Output Controller. The TCØ Output Command Word is stored in the Normal or Priority Output Driver Table.

Timed Contact Output Request

TCØ Command Word


23
16 15
11 10
9
7 6
0
A-Register
# of Timing Pulses
Matrix Address
Time a Contact is to be closed.
# of Timed Contact Controller if more than one.
0 = Immediate Return
1 = Return after output is completed.
0 = GE/PAC Pulse Output
1 = Pulse Duration Output
0 = Move Setpoint Controller UP
1 = Move Setpoint Controller DOWN

SPB TCØC01 (Normal Request) or SPB TCØC02 (Priority Request)

ERROR RETURN

NORMAL RETURN

A-REGISTER CONTENTS	ERROR
0	Driver Table Full
2	Requested Group has failed on 2 consecutive overloads
4	Invalid Matrix Address
8	Timer Failure

1.23 Timed Contact Output Driver (TCD)

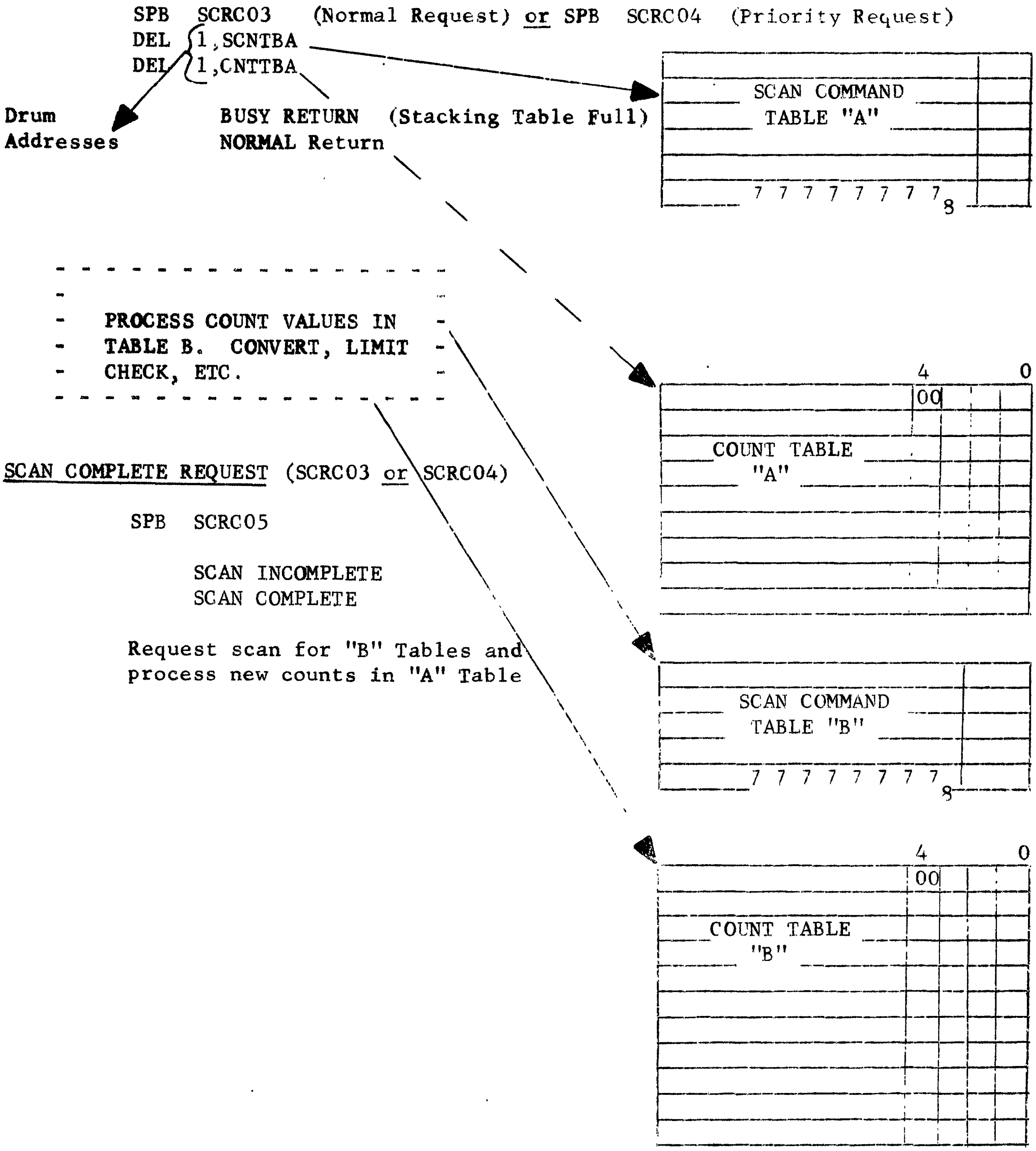
The Timed Contact Output Controller is used for addressing and controlling timed contact output groups from the Arithmetic Unit. To output a request, TCD places the command word in the A-Register. The data is transferred to the location specified by the group address of the command word. The Timed Contact Output Controller is then ready to accept another request.

All output commands are initiated from the Priority Driver Table before information is taken from the Normal Driver Table.

1.24 Scan Request Subroutine (SCR)

The Scan Request Subroutine stores the addresses of the Scanner Command Word and Count Value Tables in a stacking table. The Scanner Commands are executed by the Scan Driver. SCR process normal or priority, buffered and non-buffered scan requests. A normal request is processed on a first in/first out basis. Priority requests are processed on a last in/first out basis.

<u>Buffered requests</u> permit the Scanner to be driven near maximum speed by allowing a functional program to process one set of count values while another group of analog values is being scanned for the same program. It is recommended that scanning be done from core to save on drum/disc transfers.



```
SPB  SCRC03    (Normal Request) or SPB  SCRC04  (Priority Request)
DEL  1,SCNTBA
DEL  1,CNTTBA
```

Drum Addresses

BUSY RETURN (Stacking Table Full)
NORMAL Return

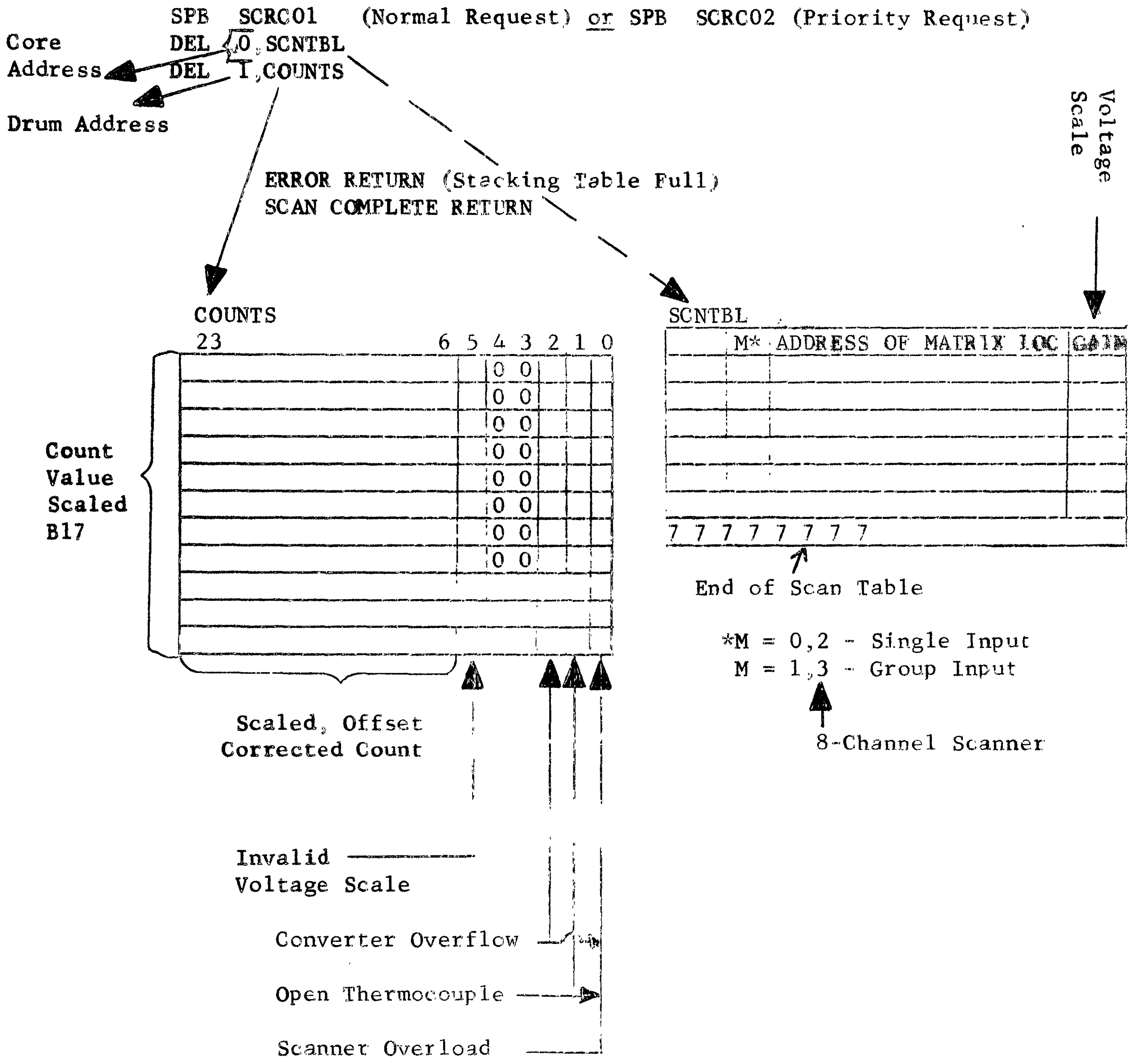
PROCESS COUNT VALUES IN TABLE B. CONVERT, LIMIT CHECK, ETC.

<u>SCAN COMPLETE REQUEST</u> (SCRC03 <u>or</u> SCRC04)

```
SPB  SCRC05

     SCAN INCOMPLETE
     SCAN COMPLETE
```

Request scan for "B" Tables and process new counts in "A" Table

A **non-buffered scan request** locks out the calling program until the analog scan request is processed.



```
SPB   SCRC01   (Normal Request) or SPB   SCRC02 (Priority Request)
DEL   0, SCNTBL
DEL   1, COUNTS
      ERROR RETURN (Stacking Table Full)
      SCAN COMPLETE RETURN
```

Reference: Scan Command Word, Programming Techniques Manual

1.25 Scan Driver (SND)

The Scan Driver outputs Scanner Commands to a Driver Table. They are sent to the Scanner by an ODL or ØUT Command. The count values are returned by an IDL or IN Instruction and stored in the specified table. The count values are converted, offset corrected, and scaled before storing. After all requested points have been scanned, the calling function is turned on.

1.26 Scan Offset Program (SCF)

The analog readings of the shorted pair are obtained for each voltage scale. This new offset is calculated from the weighted average of the current reading and four previous offset values for each voltage scale.

1.27 Corrective Action Diagnostic (CAD)

CAD performs peripheral, drum, disc, multiple output distributor, scanner, and timed-contact output corrective actions for the GE/PAC Monitor. Corrective action is taken for the following peripherals:

100 CPM Card Punch
350 CPM Card Reader
300 LPM Printer

70 CPM Card Reader
Output Typers
I/O Typers
Paper Tape Punch
Paper Tape Reader

Corrective actions are as listed:

1. Substitute an alternate output peripheral when an output peripheral fails.
2. Permit operator recovery for the 350 CPM Card Reader, 100 CPM Card Punch, and 300 LPM Printer.
3. Reset device flags and switches for the failed device.
4. Turn on any programs "locked out" for the failed device.
5. Type alarm messages indicating the exact location of the failure, when possible.

After the necessary actions are completed, CAD turns itself off and exits to the ECP.

1.28 Initialization Routine (INZ)

INZ initializes the start-up conditions for a Monitor System. Initial storage, switches, and variable locations are set for Monitor and Compatible Programs.

Each system should add their own system initialization to this routine.

1.29 Console Switch On-Line Operator Program (ØPR)

This program is used to communicate with the on-line service routines. Each routine is requested by using the Console Switches on the GE/PAC Console. (Reference: GE/PAC 4000 Operator System Console Reference Manual, Library Control No. YPG29.) The following routines operate under ØPR42:

1.29.1 On-Line Memory Change with Punch Option (MCG)

Types and/or changes memory contents.
Records program changes by typewriter and paper tape.

1.29.2 On-Line Loader (LDR)

Transfers information stored on paper tape into memory.
Compares memory contents with tape information.
Relocates tape information when storing into memory.
Loads pre-assembled library subroutines.

1.29.3 On-Line Dump (ØLD)

Punches memory contents on paper tape.
Types memory contents on the Console Typewriter.

1.29.4 On-Line Clock (CLK)

Resets the system time.
Updates the program execution times and auxiliary time counters based on the new system time.

1.29.5 Program Status (PST)

Initiates or stops the execution of functional programs operating under MONITOR control.
Locks out any functional program preventing it from being turned on.
An option, which permits a program to be turned on at a designated time, is also available. Without this option, a program is turned on immediately.

1.29.6 Peripheral In/Out of Service (PØS)

Removes peripheral devices from service without an alarm notification.

Restores failed or out-of-service peripherals to service.

1.29.7 On-Line Paper Tape Duplicator (DUP)

Duplicates paper tapes.

1.29.8 Extract Load Tape Program (XLT)

Produces a bi-octal tape from the PAL output symbolic tape.

1.29.9 PAL GE/PAC Assembler (PAL)

Assembles symbolic programs punched on paper tape or cards. Types an assembly listing.
Produces a bi-octal paper tape.

1.29.10 Demand Scan (DSN)

Assists in calibrating analog points by demand scanning of one analog point at a time.

1.29.11 Controller Change (CCG)

Changes the value of an analog point on the Analog Scan Controller.

1.30 On-Line Operator System (I/O Typer) ØPR43

This program calls on-line service routines requested by the I/O Typewriters. The following routines may be called under ØPR43:

1.30.1 On-Line Memory Change with Punch Option for I/O Typer (MCG43)

Changes memory locations through the I/O Typer.
Types the contents of memory locations.
Punches a tape record of any changed location.

1.30.2 On-Line Paper Tape Loader - I/O Typer (LDR43)

Transfers information stored on paper tape into memory.
Compares memory contents with tape information.
Relocates tape information when storing into memory.
Loads pre-assembled library subroutines.

1.30.3 On-Line I/O Typer Dump (ØLD43)

Punches memory contents in the bi-octal format on the paper tape punch.

Types or prints memory contents in the octal format on the I/O Typer or Printer.

1.31 Find/Restore Working Core Area Subroutines (FMR)

The Find Working Core Area Subroutine is used to find space for reading data from paper tape, scanning analog points, transferring a program segment, building an output data table, transferring an untested program of the Free-Time System, etc. When a space is found, this area is set to unavailable, occupied status.

The Restore Working Core Area Subroutine releases an area by setting it unoccupied and available.

To find a working core area,

```
LDA  Number of Locations
SPB  FMRC01
     Unavailable Return
     Normal Return with the location of the free
     core area in the A-Register
```

The "Unavailable Return" is taken when a working core area is not available. The system programmer, at this time, should set a delay and then initiate another FMRC01 subroutine call.

To release a working core area,

```
DLD  A-Register with the Number of Locations
     Q-Register with the starting core location
SPB  RMRC01
     Normal Return
```

1.32 Run, Stop System Subroutine (RMP)

RMP requests transfers for system subroutines from drum or disc into a working core area. After transfer, RMP branches to the first location of the subroutine. When the subroutine has completed its functions, RMP releases the working core areas and returns to the calling program.

To request a system subroutine,

```
          Set program area unavailable. (See MAP
          DLD  A-Register with the drum location
               Q-Register with the number of words
          SPB  RMPC01
               Busy Return
               Normal Return
```

To release its working core area, the subroutine must know its own size and its present core location.

```
          DLD  A-Register with No. of Words
               Q-Register with Core Location
          LDX  Return,2
          SPB  SMPC01
```

If the requested program is busy with another request, return is made at the busy return. If either FMRC01 or DTRC02 is busy, a quarter second delay is initiated in the calling program and the request is re-initiated after the delay.

APPENDIX A

MONITOR PSEUDO-OPS

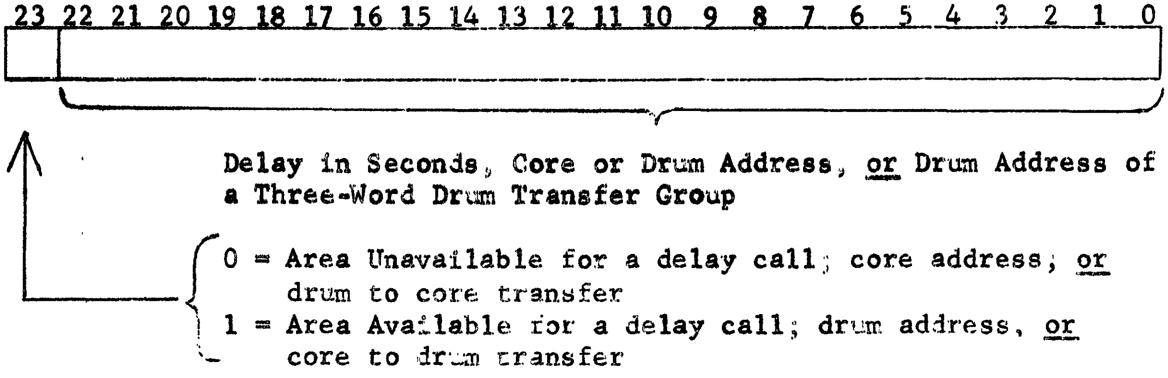
Pseudo-ops defined by MONITOR are:

BCD - Binary to 6-Bit BCD
BCN - Binary to Non-Edited Character
CLK - Clock
DEL - Delay
DFE - Floating Point to E Type Floating Point Decimal
DFP - Floating Point to Fixed-Point Decimal
DFX - Binary to Fixed-Point Decimal
FBB - Binary to 4-Bit BCD
FØR -
ØCT - Binary to Octal (integers)
PRG
SIZ

The BCD, BCN, CLK, DFE, DFP, DFX, FBB, and ØCT pseudo-ops are used with the Output Program as format words. Refer to the Output Program, subcategories of 1.14, for explanation of these pseudo-ops. The DEL, FØR, PRG, and SIZ pseudo-ops are used in subroutine calls and are explained below:

The DEL pseudo-op is used in the Scan Request, Input Request, Output Request, 1st word of the 3-word Drum Transfer Request, and Set Program Delay Subroutine Calls. Coding examples and format are as follows:

```
DEL    0, 15*SECND            DEL    1,FMTB
```

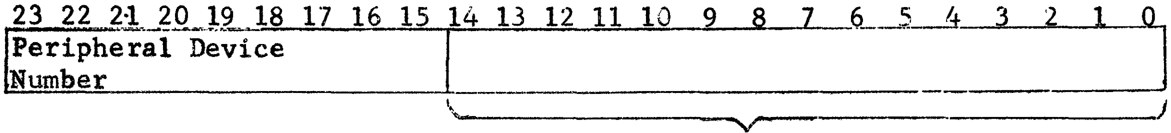


Delay in Seconds, Core or Drum Address, or Drum Address of a Three-Word Drum Transfer Group

0 = Area Unavailable for a delay call; core address; or drum to core transfer
1 = Area Available for a delay call; drum address, or core to drum transfer

The FØR pseudo-op is used as noted below:

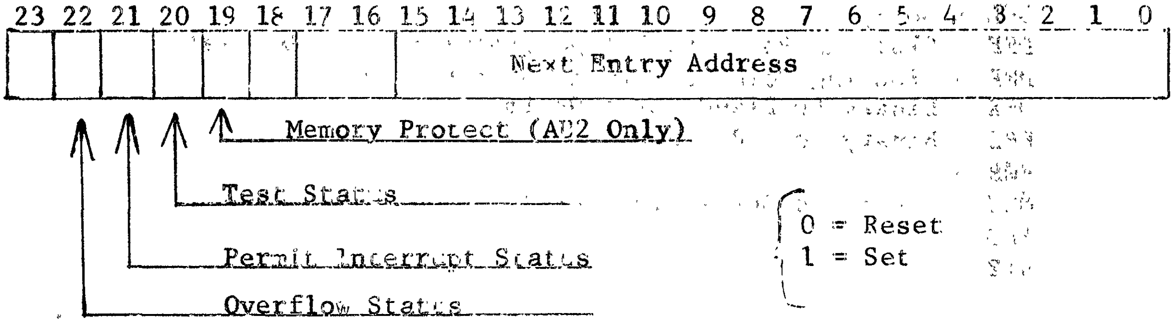
```
FØR LTYPER, 0            FØR CRRDR, 24
```



Message Area Number in ØUTC01, ØUTC02, OUTC03, PAVC01, PAVC02, PAVC03, and PAVC04 subroutine calls, or Number of characters to be read in INC01 or INPC01 subroutine calls.

The PRG pseudo-op is used to maintain the REGSTG and RSX (AU2 only) Tables of ECP and by the Turn Program Off Subroutine. It contains the flip-flop status of overflow, permit interrupt, test, and memory fence (AU2 only), plus the next entry location.

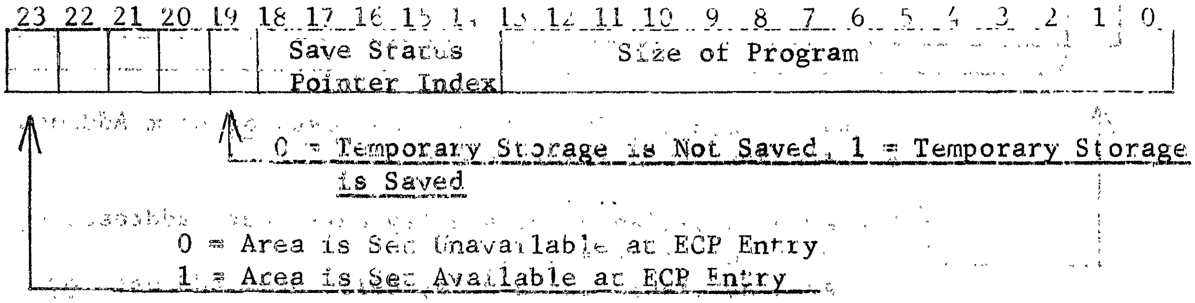
```
PRG  0, 1, 0, START    (Used by AU1 - No Memory Fence Flip-Flop)
PRG  0, 1, 0, START,0  (Used by AU2 memory fence status)
```



For a Drum/Core system, if the next entry address is set to zero, the program begins at its first location.

The SIZ pseudo-op specifies the required parameters for the second word of the three-word ECP Drum Transfer Group.

SIZ 1, 1, 3, /400 (This pseudo-op tells the area availability status, Bit 23; save status, Bit 19; save status pointer index, Bits 18-14; and Size of the Program.)



23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Save Status Pointer Index
Size of Program
0 = Temporary Storage is Not Saved, 1 = Temporary Storage is Saved
0 = Area is Set Unavailable at ECP Entry
1 = Area is Set Available at ECP Entry

APPENDIX B

COMMUNICATION CALLS

		Page
DELC01	Set Program Delay	1-9
DTRC01	Drum Transfer Request (Immediate Return)	1-12
DTRC02	Drum Transfer Request (Drum Transfer Completed Return)	1-12
FMRC01	Find a Working Core Area	1-43
FRPC01	Find Register Pointer	1-13
INC01	Non-Buffered Input Request	1-30
INPC01	Buffered Input Request	1-28
INPC02	Read Completed Request	1-29
MAP01	Core Map Maintenance - Set Area Occupied	1-11
MAP02	Core Map Maintenance - Set Area Unoccupied	1-11
MAP03	Core Map Maintenance - Set Area Unavailable	1-11
MAP04	Core Map Maintenance - Set Area Available	1-11
MØRC01	Normal Non-Timed Multiple Output Request	1-32
MØRC02	Priority Non-Timed Multiple Output Request	1-32
MØRC03	Normal Timed Multiple Output Request	1-33
MØRC04	Priority Timed Multiple Output Request	1-33
ØFFC01	Turn Program Off Request	1-8
ØUTC01	Peripheral Availability & Data Area Output Request	1-16
ØUTC02	Normal Output Request	1-16
ØUTC03	Priority Output Request	1-17
PAVC01	Peripheral Availability Request with Alternate Selection	1-14
PAVC02	Data Area Availability Request	1-15
PAVC03	Output Request Which Places Request in Stacking Table	1-15
PAVC04	Priority Request (Places request at the top of the list)	1-15
RMPC01	Request a System Subroutine	1-44
RMRC01	Release a Working Core Area	1-43
SCRC01	Normal Non-Buffered Scan Request	1-38
SCRC02	Priority Non-Buffered Scan Request	1-38
SCRC03	Normal Buffered Scan Request	1-37
SCRC04	Priority Buffered Scan Request	1-37
SCRC05	Scan Completed Request	1-37
SMPC01	Release Working Core Area of a System Subroutine	1-44
TCØC01	Normal Timed Contact Output Request	1-35
TCØC02	Priority Timed Contact Output Request	1-35
TPNC01	Turn Program On Request	1-10
TPNC02	Turn Program On Request if Program is Currently Off	1-10

APPENDIX C

SYMBOLS

ALERT - Peripheral Activate Flag
The peripheral device priority number corresponds to the associated bit.
Example: Device #3 = Bit 3 of Flag

ALTFLG- No Alternate Device Flag
This flag shows that an output message for the requested output peripheral has no working alternate, therefore, the output message was destroyed.
Example: Device priority number i corresponds to bit i.

ALTTBL- Alternate Device Priority Numbers for Peripheral Devices
The alternate for device priority number i is found in ALTTBL+i.
Example: Priority Device #5 = ALTTBL+5

AUXTM - Table of Auxiliary Time Counters

AVLMAP- Available Area Map - See 1.1 of this manual.

BAD - Peripheral Device Failure Flag
The peripheral device priority number corresponds to the associated bit.
Example: Device #4 = Bit 4 of Flag

CØRMAP- Occupied Area Map - See 1.1 of this manual.

DMCRNØ- Three Times the Running Program Number

DTAREA- Data Area Bit Word
Area_i corresponds to Bit_{i-1}
Example: Area 2 = Bit 1
Area 3 = Bit 2

DVCØDE- Device Codes
A table of hardware addresses for peripheral devices arranged in descending order.
Example: #0 = Highest Priority Device

FAILUR- Peripheral or Hardware Failure Device Flag
Bit 0 - Output Typer or Paper Tape Punch or
High Speed Peripheral Failure
1 - MOD Timer Failure
2 - Drum or Disc Timer Failure
3 - Scanner Timer Failure
4 - TCO Timer Failure
5 - I/O Typer, Paper Tape Reader, or 70 CPM Reader Failure
A one in any of the above bits indicates a failure.

ØØS - Peripheral Device Out-of-Service Flag
The peripheral device number corresponds to the associated bit.

PRIØNØ- One Times the Running Program Number

PRØCFG- Output Peripheral Device-in-Progress Flag
The peripheral device number corresponds to the associated bit.

PRØG - Program Execution Times
The table starts with Program No. 1.

PRØGNØ- Eight Times the Running Program Number

SELECT- Peripheral Action Flag
The peripheral device number corresponds to the associated bit.

TIME - Time in System Counts (Cleared at Midnight)

WAITFG- Output Peripheral Message Completion Flag
The peripheral device number corresponds to the associated bit.

WAITRQ- Number of Words Reserved in the Stacking Table for Output Peripheral Message (Each output request requires two words in the table).
This table is indexed by device priority number.

XFER - Number of Drum Transfer Requests Waiting for a Program
This table is referenced by program number.
Example: Program 7 = XFER+7

GE-PAC 4000

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER

MONITOR USER'S MANUAL

GENERAL ELECTRIC COMPANY
PHOENIX, ARIZONA

Library Control No. YPG51M

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. BOX 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

CONTENTS

INTRODUCTION

The objective of the MONITOR USER'S MANUAL is to present ways of using the Monitor System in conjunction with GE/PAC 4000 Process Automation Computers. Use of this manual is predicated upon, and presupposes the reader's familiarity with, real-time system requirements, GE/PAC programming techniques, and Process Assembler Language.

Designed especially for the user, the manual explains what Monitor does and what the programmer must do to insure its successful operation. The publication also discusses individual subprograms which comprise the Monitor System, and includes the communications links which must be provided in the functional programs to allow necessary exchanges of data between Monitor and the outside world. A detailed analysis of Monitor logic is not included in this manual; rather, it is covered in the various Monitor program write-ups which are available through the Programming Library.

Examples used throughout the text showing various calling sequences are not to be construed as being the only method of using such calls. It would take more space than is practicable to allow in such a text to show all the possibilities of each subroutine call.

The GE/PAC family of programming manuals consists of separate, complete booklets, each of which deals with a specific subject. MONITOR USER'S MANUAL is the eighth in this series of publications, and can be identified as such by the use of the numeral eight prefix in paragraph headings and subheadings.

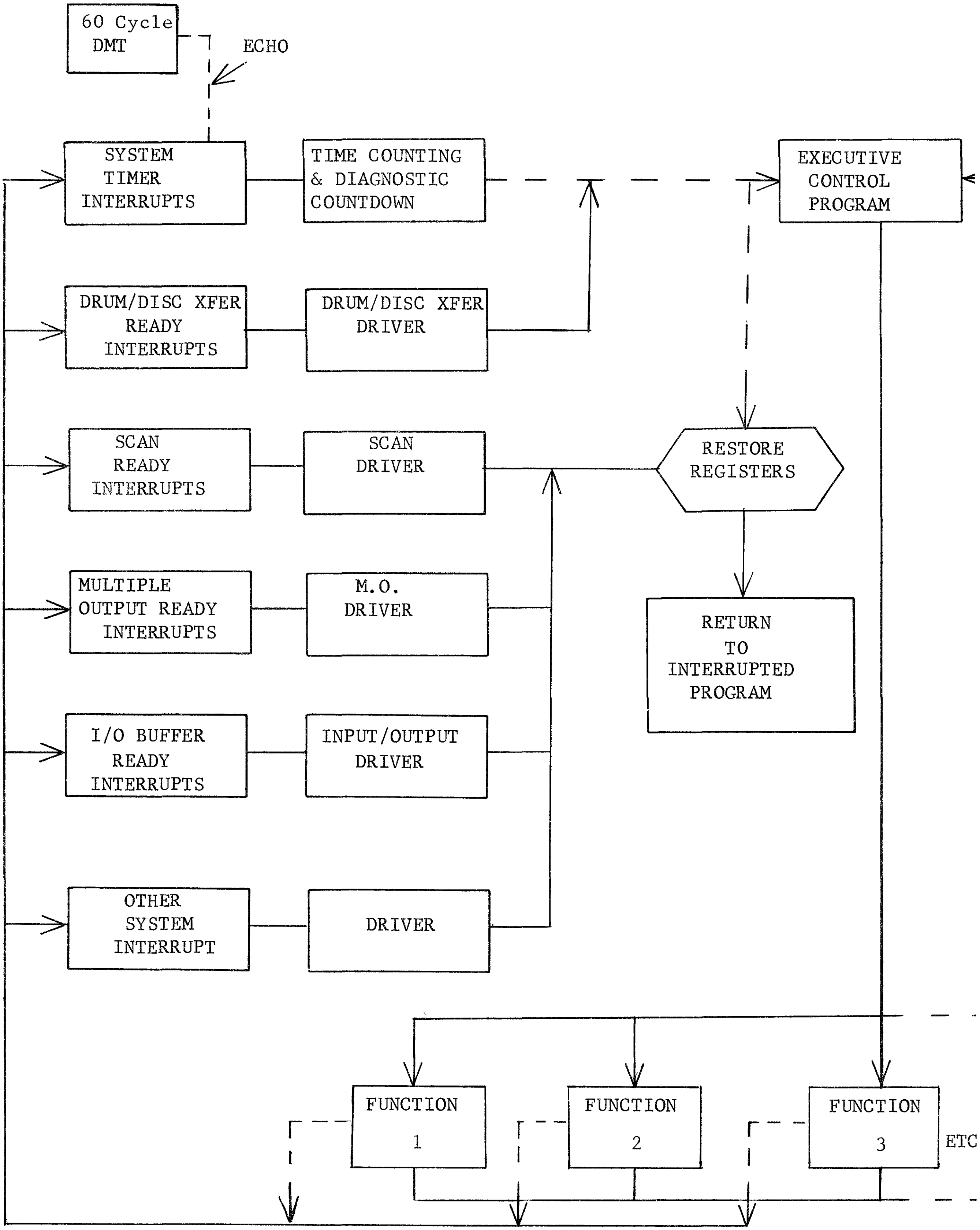
GENERAL

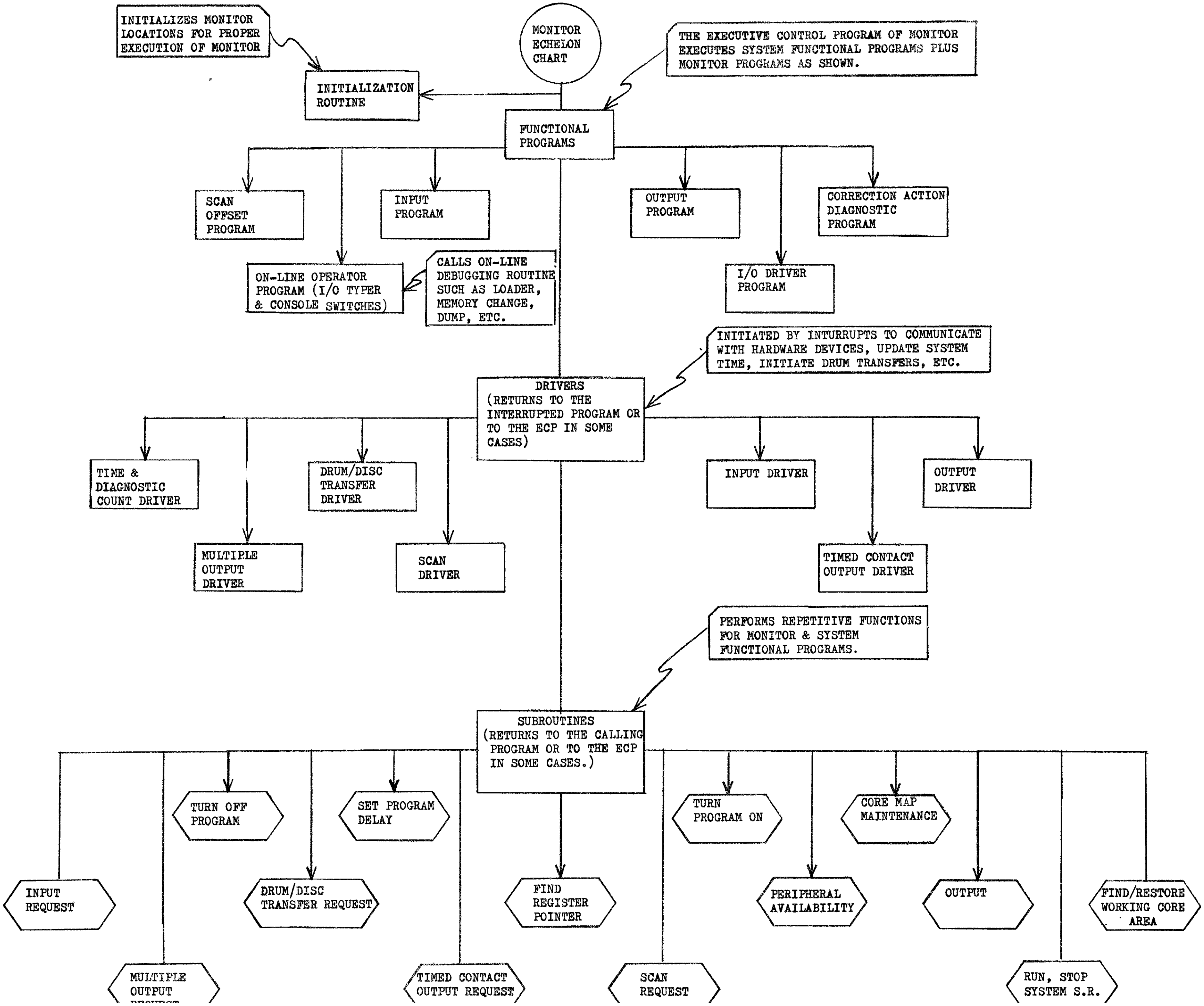
A real-time process is characterized by the occurrence of many events, some continuous, others random in nature. Events may occur simultaneously. A digital computer, however, is a serial device; that is, it performs its program operations serially, one by one. Therefore, the matching of the digital computer and a real-time process requires a control system which coordinates the requirements and characteristics of both.

A GE/PAC Monitor is an operating system composed of a library of subprograms which provide the basis for a process computer system. It is the framework to which the specific functional programs are added. The Monitor accomplishes the timing and scheduling operations, input/output, internal data transfer, timed contact, multiple output, corrective action, initialization, etc. Many options are available which may or may not be included in a tailored Monitor System.

The user requests a tailored Monitor by checking the desired options on the Monitor Checklist, a copy of which is included in this manual (see Appendix D). Additional copies of the Checklist may be obtained upon request from the Programming Library.

GE/PAC MONITOR INTERRUPTS




MONITOR ECHELON CHART
INITIALIZES MONITOR LOCATIONS FOR PROPER EXECUTION OF MONITOR
INITIALIZATION ROUTINE
THE EXECUTIVE CONTROL PROGRAM OF MONITOR EXECUTES SYSTEM FUNCTIONAL PROGRAMS PLUS MONITOR PROGRAMS AS SHOWN.
FUNCTIONAL PROGRAMS
SCAN OFFSET PROGRAM
INPUT PROGRAM
OUTPUT PROGRAM
CORRECTION ACTION DIAGNOSTIC PROGRAM
ON-LINE OPERATOR PROGRAM (I/O TYPER & CONSOLE SWITCHES)
CALLS ON-LINE DEBUGGING ROUTINE SUCH AS LOADER, MEMORY CHANGE, DUMP, ETC.
I/O DRIVER PROGRAM
INITIATED BY INTERRUPTS TO COMMUNICATE WITH HARDWARE DEVICES, UPDATE SYSTEM TIME, INITIATE DRUM TRANSFERS, ETC.
DRIVERS (RETURNS TO THE INTERRUPTED PROGRAM OR TO THE ECP IN SOME CASES)
TIME & DIAGNOSTIC COUNT DRIVER
DRUM/DISC TRANSFER DRIVER
INPUT DRIVER
OUTPUT DRIVER
MULTIPLE OUTPUT DRIVER
SCAN DRIVER
TIMED CONTACT OUTPUT DRIVER
PERFORMS REPETITIVE FUNCTIONS FOR MONITOR & SYSTEM FUNCTIONAL PROGRAMS.
SUBROUTINES (RETURNS TO THE CALLING PROGRAM OR TO THE ECP IN SOME CASES.)
TURN OFF PROGRAM
SET PROGRAM DELAY
TURN PROGRAM ON
CORE MAP MAINTENANCE
INPUT REQUEST
DRUM/DISC TRANSFER REQUEST
FIND REGISTER POINTER
PERIPHERAL AVAILABILITY
OUTPUT
FIND/RESTORE WORKING CORE AREA
MULTIPLE OUTPUT REQUEST
TIMED CONTACT OUTPUT REQUEST
SCAN REQUEST
RUN, STOP SYSTEM S.R.

8.1 SYSTEM DESCRIPTION

Monitor consists of the component programs, as described below.

8.1.1 Executive Control Program (ECP)

The Executive Control Program runs permitted and schedules the execution of programs based on priority, execution time, and core availability. All time-critical interrupts are permitted before system programs are executed.

System programs are executed in priority order by comparing the programs' next execution time ($PR\emptyset G_n$) with the current time (TIME). The highest priority program for which the execution time is equal to or less than the current time is executed. When the execution time is current for a program, the ECP requests a transfer from drum or disc to core providing:

1. The program is not in core.
2. The program is not presently being transferred.
3. A core area is available.

After the transfer has been completed, the program is initiated. If there is no available core area for that program, ECP tests the execution time for the next lower priority program.

When a program priority change request has been made, the ECP will begin its search at the top of the priority list as a result of this change. Examples of such program actions are: MØD (Multiple Output Distributor), TCØ (Timed Contact Output), Output Completed Returns, Scan Complete Returns, ITC (Interrupt Time Counter), or Drum/Disc transfer interrupts.

If a program has a "turned off" or "locked out" code in its PRØG location, it is not executed until a time for execution is assigned.

There are three classifications of register storage for functional programs. They are:

1. Programs which have no register storage of their own.
2. Programs which have their own 8-word block of register storage.
3. Programs which share an 8-word block of register storage with other functional programs.

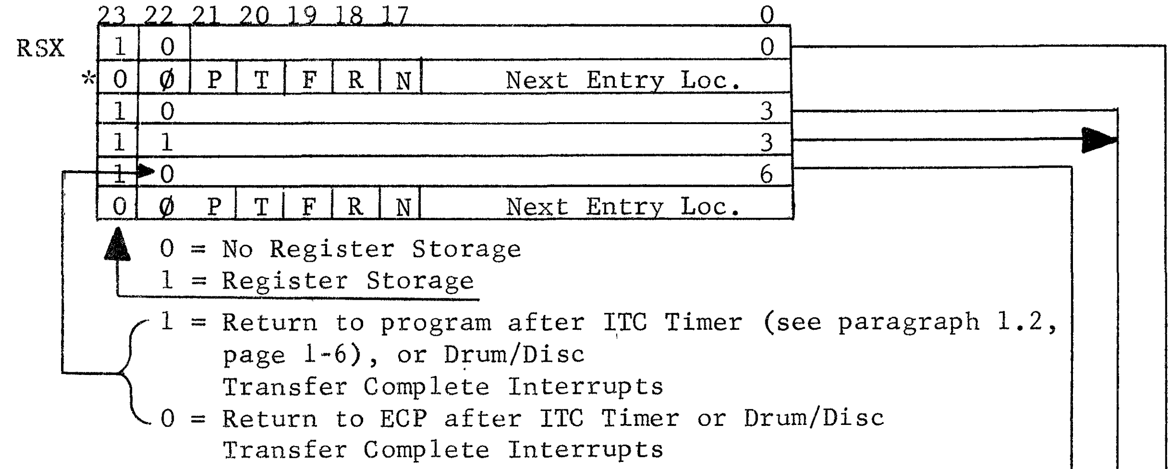
The Register Pointer Table (RSX) tells what classification is assigned for each program and contains the following information.

The current ECP priority order is defined in the PRGTBL Table. The latter is included in the Monitor ØFile tape and allows the programmer to change the priority of a program after the initial Monitor Assembly.

Note: Any direct branches to the ECP must be done in the inhibited state. The RSX Table is eliminated when all system programs have their own 8-word register storage block.

Register Pointer Table

This table contains an index to the 8-word storage block for programs having full register storage or sharing storage. For programs with no register storage, the table contains the flip-flop status and the program's next entry location.

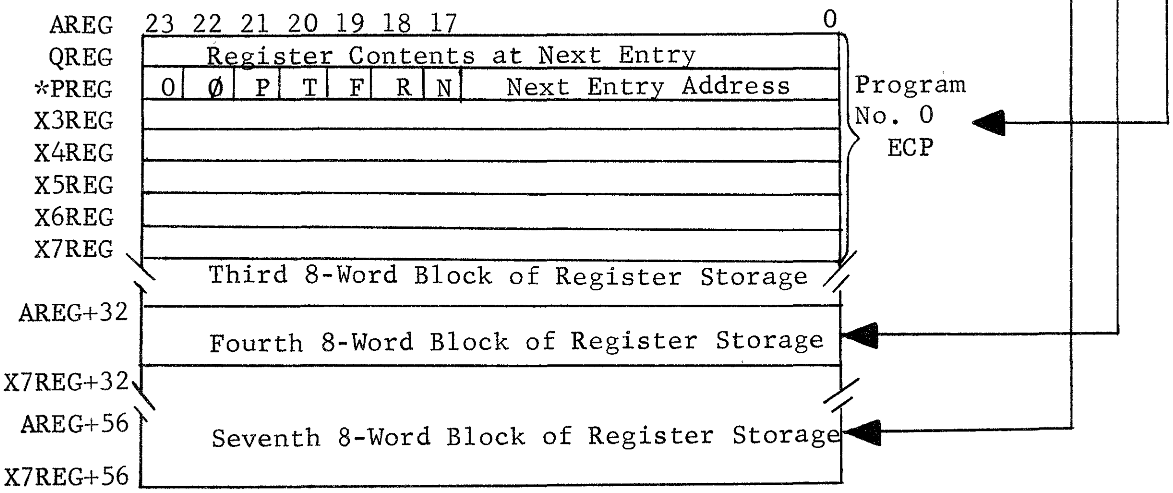

23 22 21 20 19 18 17 0
RSX
1 0 0
* 0 Ø P T F R N Next Entry Loc.
1 0 3
1 1 3
1 0 6
0 Ø P T F R N Next Entry Loc.

0 = No Register Storage
1 = Register Storage

1 = Return to program after ITC Timer (see paragraph 1.2, page 1-6), or Drum/Disc Transfer Complete Interrupts
0 = Return to ECP after ITC Timer or Drum/Disc Transfer Complete Interrupts

For programs sharing full register storage, the index would be the same.

Register Storage Table (REGSTG) stored in permanent core.


AREG
QREG
*PREG
X3REG
X4REG
X5REG
X6REG
X7REG
23 22 21 20 19 18 17 0
Register Contents at Next Entry
0 Ø P T F R N Next Entry Address
Program No. 0 ECP
Third 8-Word Block of Register Storage
AREG+32
Fourth 8-Word Block of Register Storage
X7REG+32
AREG+56
Seventh 8-Word Block of Register Storage
X7REG+56

*PREG Flip-Flops

Ø = 1 - Set Overflow; 0 - Reset
P = 1 - Set Permit Interrupt; 0 - Reset
T = 1 - Set Test Flip-Flop; 0 - Reset
F = 1 - Set Memory Fence; 0 - Reset
R = 1 - Absolute Permanent Core Location
0 - Relative Address (Drum/Core)
For an all-core system, R is always zero.
N = 1 - Negative Relative Entry Address (Bits 0-15)
0 - Positive Relative Entry Address (Bits 0-15)
For an all-core system, N is always zero.

The other tables assisting the ECP in performing its functions are:

PRØG - Program Execution Time Table

TIME 00001433

Program #			
1	PRØG	40000000	PROGRAM OFF
2		00001342	PROGRAM CURRENTLY RUNNING
3		00001465	PROGRAM DELAYED
4		40000001	LOCKED OUT
5		40000000	OFF
6		40000000	OFF
7		00000000	PROGRAM ON (Based on event)

Drum Transfer Control Table

<table>
<tr><th></th><th></th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>15</th><th>13</th><th>0</th></tr>
<tr><td rowspan="3">Program No. 0 Transfer Group</td><td>DRMLØC</td><td>0</td><td colspan="9">BEGINNING DRUM/DISC ADDRESS</td></tr>
<tr><td>SIZE</td><td>A</td><td>C</td><td>T</td><td>N</td><td>S</td><td colspan="3">INDEX TO 3-WORD GROUP IN SAVE TABLE</td><td colspan="2">NUMBER OF WORDS IN PROGRAM (SAVED BLOCK NOT INCLUDED)</td></tr>
<tr><td>CØRLØC</td><td colspan="6"></td><td colspan="4">BEGINNING CORE ADDRESS</td></tr>
<tr><td></td><td>DRMLØC+3
SIZE+3
CØRLØC+3</td><td colspan="10">Drum Transfer Control Group for Program #1</td></tr>
<tr><td></td><td>CØRLØC+3n</td><td colspan="10">Drum Transfer Control Group for Program #n</td></tr>
</table>

A - Area Availability on Entry From ECP
    1 = Available; 0 = Unavailable
C - 1 = Program is in core
    0 = Program is not in core
T - 1 = Program is in transfer from drum/core or has requested DTRC02 XFER
    0 = Program is not in transfer
N - 1 = Program is running with core area available
    0 = Program is running with core area unavailable
S - 1 = Save temporary storage on drum if overwritten
    0 = Do not save temporary storage

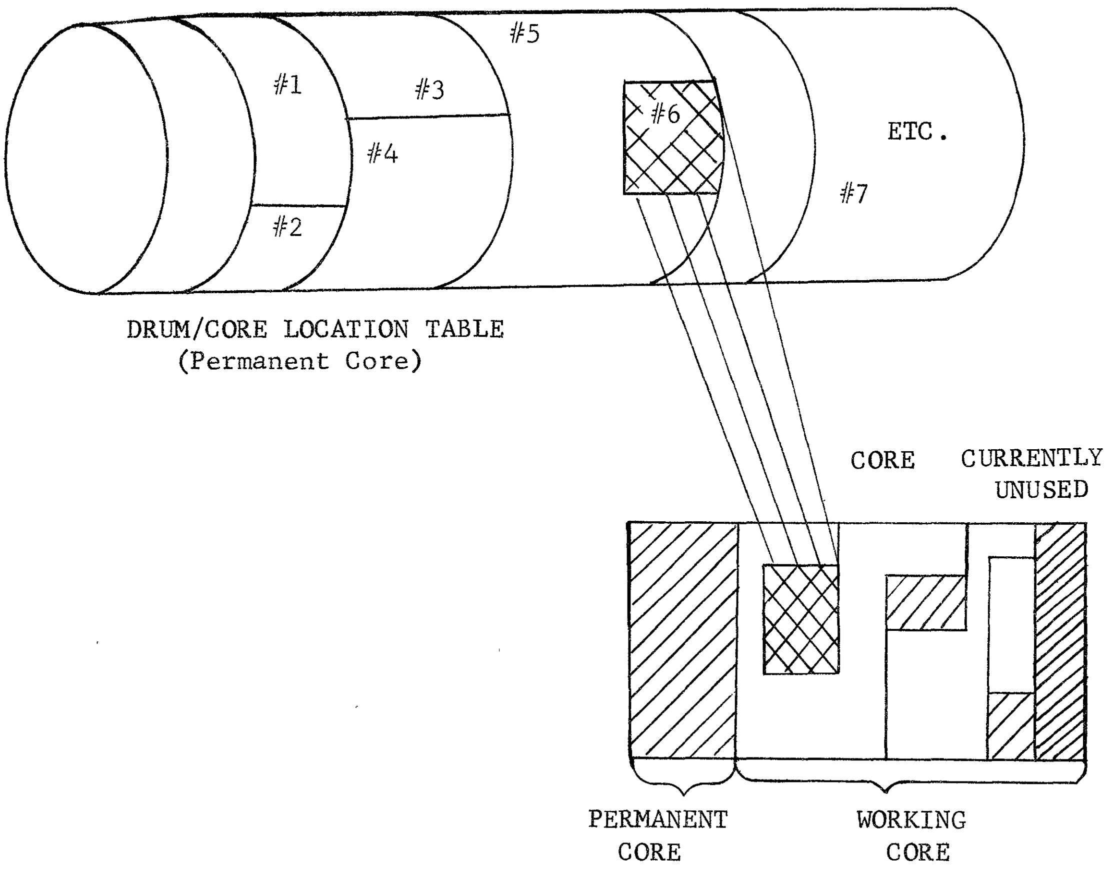
The Save Status Area Control Table is used for programs requiring temporary storage to be saved in an unprotected area on drum before overwriting. This feature is called "Save Status". The ECP transfers the temporary storage of programs having "Save Status" to drum before another program is transferred in its place. However, when a functional program is turned off, its temporary storage is not automatically saved on drum.

Save Status is specified in the Drum Transfer Group Table above in the SIZE Word, Bit 19.

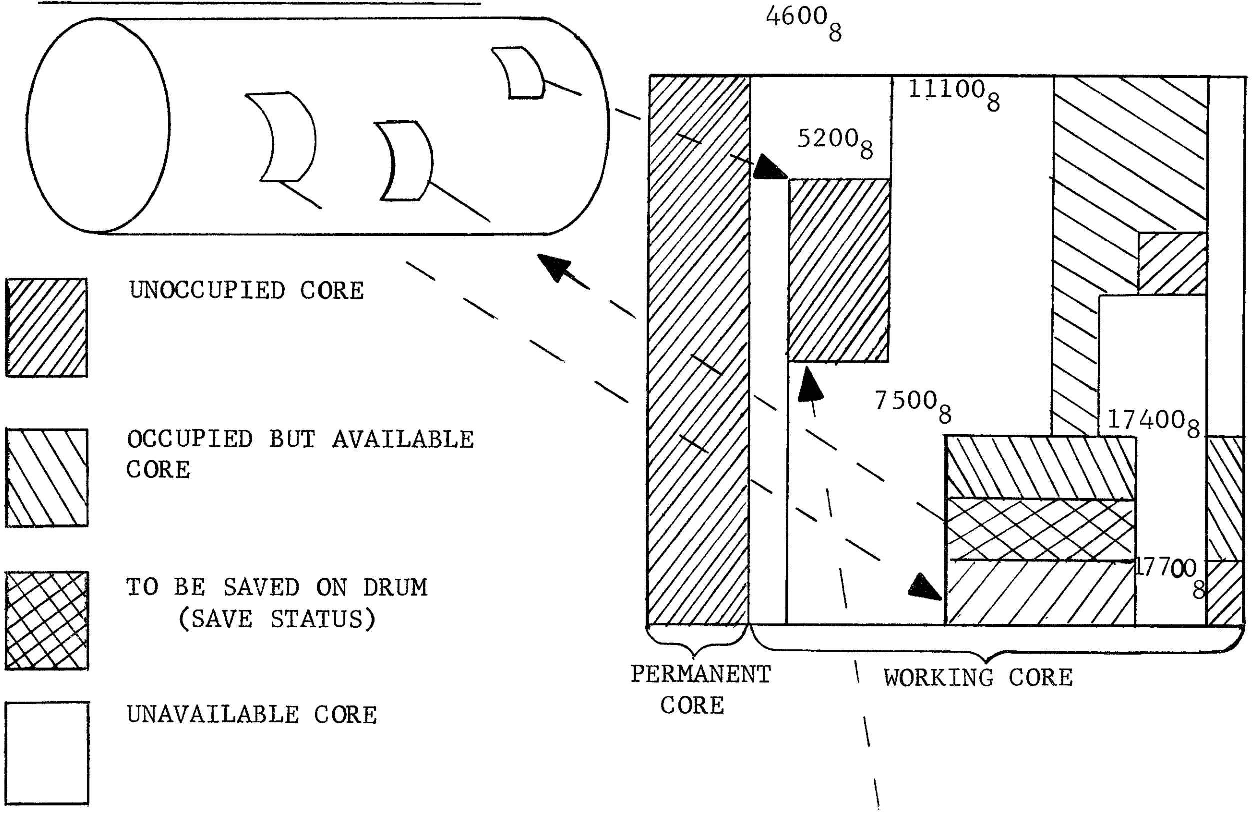
Save Status Area Control Table

	23	22	15	13	0
SAVTBL	0	BEGINNING DRUM OR DISC ADDRESS FOR SAVED AREA			
SVSIZE	0 0 0 0 0 0 0 0 0 0			NUMBER OF WORDS IN SAVED BLOCK AT END OF PROGRAM	
SVLØC			BEGINNING CORE ADDRESS		

ECP Drum/Core Communications



ECP Drum/Core Communications



Each bit in the following tables represents 64 core locations 100$_8$ (standard block size).

OCCUPIED AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CØRMAP	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1

UNUSED

1 = Core Area Occupied by a Functional Program

AVAILABLE AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AVLMAP	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

UNUSED

1 = Core Area Unavailable for Overwriting

OCCUPIED SAVE STATUS AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STSMAP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

UNUSED

1 = Occupied by save status

8.1.2 Time and Diagnostic Count Driver (ITC)

ITC uses two interrupts to control the system timing. The first (non-inhibitable) interrupt occurs each 16 2/3 MS. On interrupt, a DMT counter is decreased by one. When the counter equals minus one, the second (inhibitable) interrupt is triggered.

The second interrupt causes entry to ITC. The DMT Counter is initialized by storing the number of 16 2/3 MS intervals in a time count. This number, NCYCLE, must be evenly divisible into sixty and is specified by the System Programmer at assembly time. NCYCLE determines the length of time represented by one time count (1 second, 1/2 second, 1/4 second, etc.). ITC increases the time of day (TIME) by one. The time of day is cleared at midnight and the calendar is updated. ITC also references all program and auxiliary time counters at the beginning of each day. Auxiliary Time Counters are used by functional programs which are turned on by the Monitor Initialize Routine which require initiation at set time intervals (1 minute, 2 minutes, etc.). Typical programs would be the Scan or Performance Calculations.

ITC also tests for interrupt driven device failures on the Peripheral Buffer, Output Distributor, Scanner, etc. Each device is assigned a specified CØUNT which is used for counting the time required for activating a particular device. This location (CØUNT) represents the maximum number of time count intervals in which action should be completed. If a CØUNT becomes negative, the device has failed and the Corrective Action Program is turned on. The system programmer may also count for special timing functions. When a system count becomes negative, the ITC turns on a system program (APRØGM), which handles diagnostic functions.

When an interrupt occurs during the execution of an interruptable system subroutine. ITC returns immediately to the interrupted subroutine.

ITC returns control to the interrupted program or to the ECP depending on the indicator set in the Register Storage Table.

8.1.3 Save Registers Subroutine (SRG)

SRG saves the register contents and/or next entry location for a functional program. For functional programs having full register storage or sharing register storage, the P Counter, A, Q, and X3 through X7 Registers for the interrupted program are transferred to the Register Storage Table. Otherwise, only the next entry point to the functional program is saved.

8.1.4 Restore Registers Subroutine (RRG)

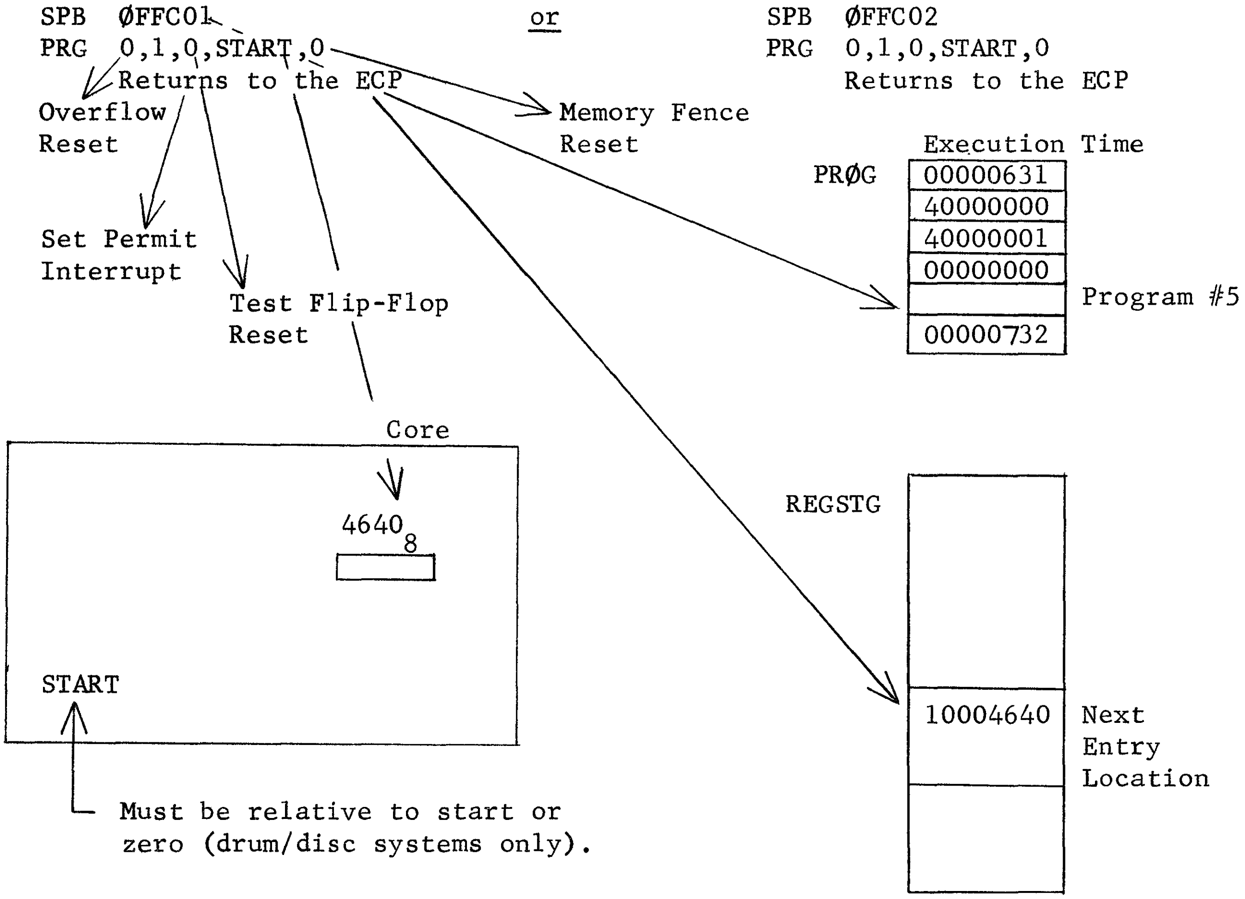
RRG returns to an interrupted program at the designated entry point. The contents of register storage are transferred to the register locations for programs having full register storage. These values represent the contents of the various registers for the running program at the time of its last interrupt.

8.1.5 Turn Program Off Subroutine (ØFF)

The Turn Program Off Subroutine stops the ECP from initiating the execution of functional programs. Programs are turned off by placing the "off" constant, 40000000_8, as the next execution time. Only a running program may turn itself off.

After the "off" constant is stored, control is transferred to the ECP.

To communicate with ØFF, use the following calling sequences:



```
SPB  ØFFC01              or          SPB  ØFFC02
PRG  0,1,0,START,0                   PRG  0,1,0,START,0
     Returns to the ECP                   Returns to the ECP
```

(Reference: Page 1-2)

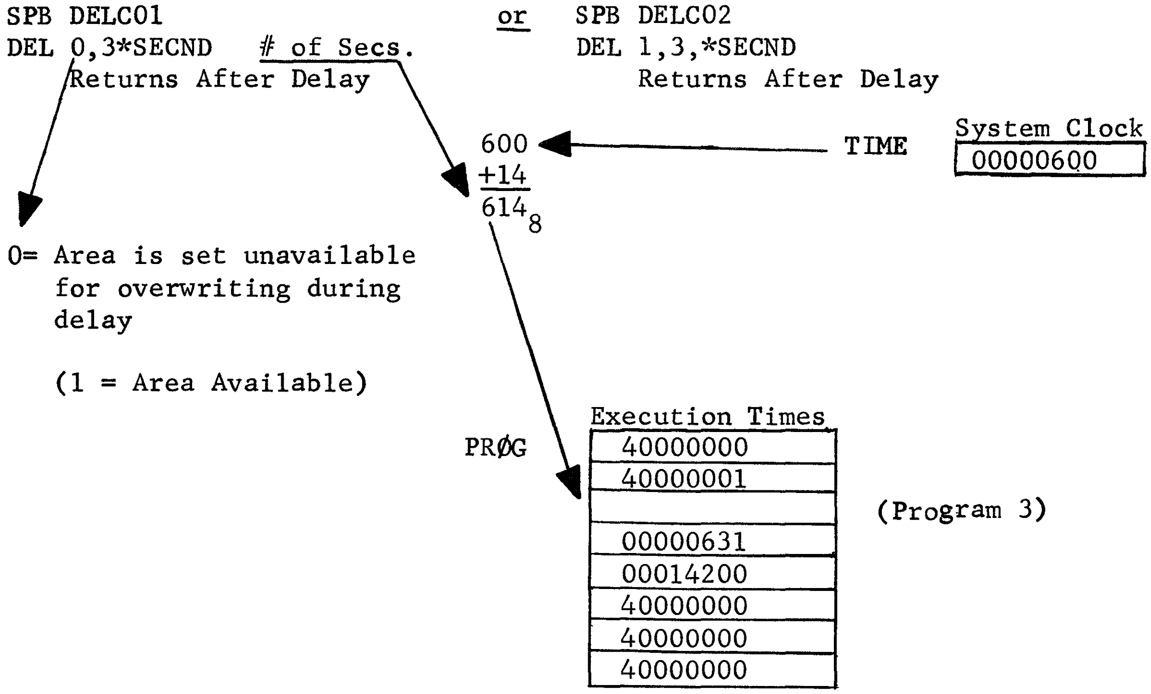
The ØFFC02 call gives the programmer the ability to set the program area unoccupied and available while the program is turned off. Save status is not saved on drum/disc. Programs that are transferred from drum to core each time executed could effectively use this call.

8.1.6 Set Program Delay Subroutine (DEL)

DEL delays the execution of a functional program that is running currently for a specified time period. The delay, in time counts, is added to the current time and stored in PRØG for the calling function. Only a running program can delay itself.

The A, Q and X3 through X7 Registers are saved for those programs having register storage.

To set a delay, use the following calling sequence:



```
SPB DELC01                        or   SPB DELC02
DEL 0,3*SECND   # of Secs.             DEL 1,3,*SECND
    Returns After Delay                    Returns After Delay
```

0= Area is set unavailable for overwriting during delay

(1 = Area Available)

Execution Times
40000000
40000001
00000631
00014200
40000000
40000000
40000000

The above example shows a 1/4 second system.

DELC02 call gives the programmer the ability to set the program area unoccupied and available during long delays. Save status will not be saved on drum/disc. This call should be used by programs which run at long time intervals. Setting the area unoccupied cuts down ECP map search time. Programs that are transferred from drum to core each time executed would also use this call effectively.

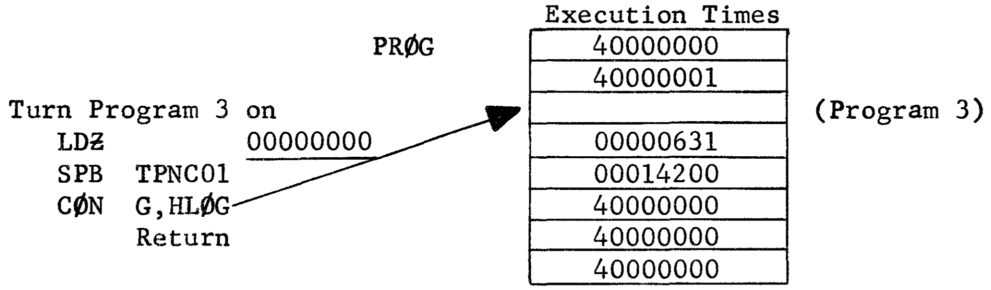
8.1.7 Turn Program on Subroutine (TPN)

The Turn Program On Subroutine is used to change the execution time of functional programs. The execution time and the program number are given in the calling sequence.

After the new execution time is stored for a program, control is returned to the calling function.

Programs which are "locked out", next time of execution 40000001_8, may not be turned on by TPN.

TPN returns with all ones in the A-Register when a request is made to turn a program on which is "Locked out".



```
Turn Program 3 on
   LDƵ       00000000
   SPB  TPNC01
   CØN  G,HLØG
        Return
```

	Execution Times	
PRØG	40000000	
	40000001	
		(Program 3)
	00000631	
	00014200	
	40000000	
	40000000	
	40000000	

```
HLØG  EQL  3
```

The TPNC01 call can turn a program on immediately or set any execution time desired in the PRØG Table.

To enter an execution time only if the program has the off constant in the PRØG Table:

```
   LDA  Execution Time or LDƵ
   SPB  TPNC02
   CØN  G,HSCAN
        Return

HSCAN  EQL  8
```

The following calling sequence allows the system programmer to specify which program will run next. If the program is in core, it will be entered immediately; if it is on drum, its transfer will be initiated.

```
   LDƵ
   SPB  TPNC03
   CØN  G,MSCAN

MSCAN  EQL  7
```

At the next ECP entry, the MSCAN program is turned on. If this program has the "lockout" constant in its PRØG location, the request is ignored.

An example of when to use the TPNC03 subroutine call would be: If program #X decides that another program must run next, program X makes a TPNC03 call and either delays or turns itself off. The use of this calling sequence is only required when a program is dynamic and changing and the computer speed is not fast enough to handle functions in their normal sequence.

8.1.8 Map Maintenance Subroutine (MAP)

MAP is used to update the core map tables (CØRMAP and AVLMAP). Core areas may be set occupied, unoccupied, available, or unavailable:

```
SPB  MAP01 - Set Area Occupied or
SPB  MAP02 - Set Area Unoccupied or
SPB  MAP03 - Set Area Unavailable or
SPB  MAP04 - Set Area Available
     Return to the calling program
```

Unoccupied Core Area Map Table

CØRMAP

CØRMAP	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

In this table, "one" bits indicate that the core area represented is occupied by a functional program whether its area is available or not.

Available Core Area Map Table

AVLMAP

AVLMAP	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

In this table, "one" bits indicate that the core area is unavailable for overwriting.

Each bit represents 2^k words of working core area. For example, if k = 5, each bit represents 32 words of core. The working core area starts at fixed location such as 12000_8.

In our example, 5 programs are in core occupying areas:

12000_8	-	12337_8	(word 1, bits 0 - 6)
12540_8	-	13277_8	(word 1, bits 11 - 21)
13600_8	-	14737_8	(word 2, bits 4 - 22)
16500_8	-	17337_8	(word 4, bits 2 - 14)
17400_8	-	17777_8	(word 4, bits 16 - 23)

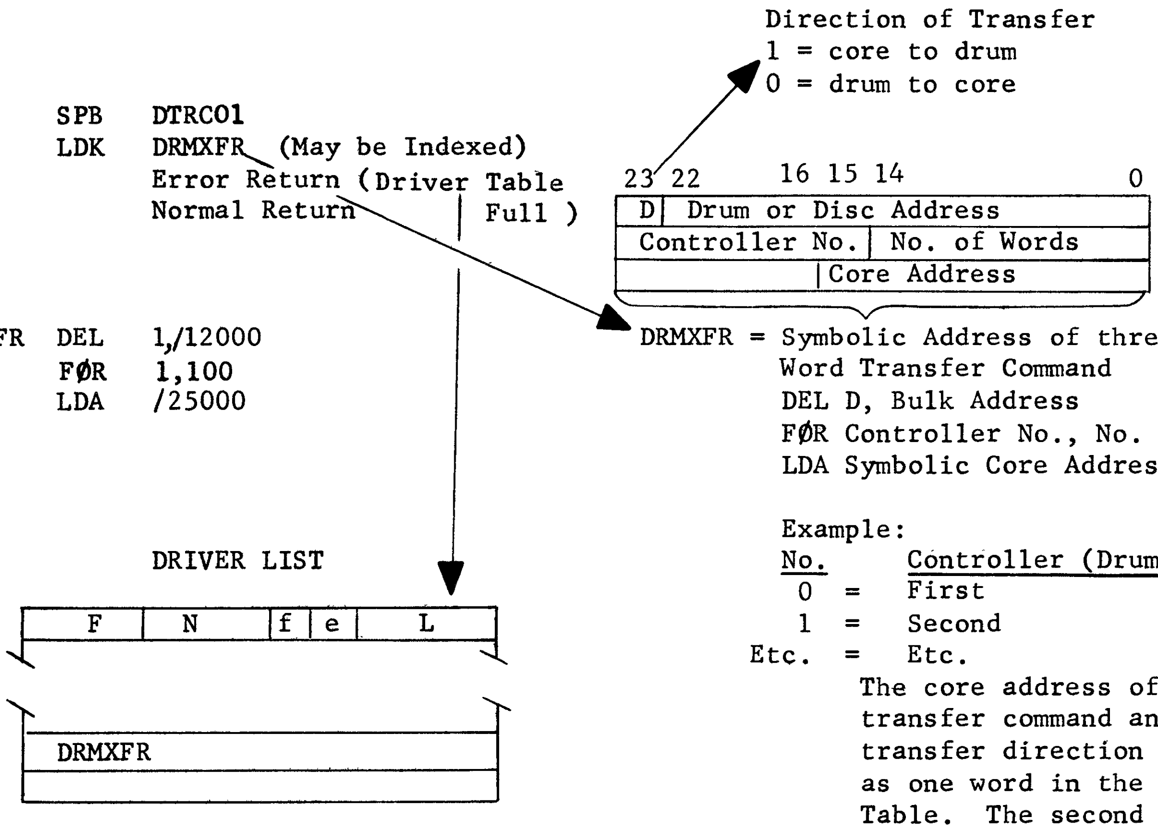
NOTE: The program in core area 12000_8 - 12337_8 is running with its core area available.

The beginning of the map table is the upper right hand bit of the table and the end is the lower left hand bit.

8.1.9 Drum/Disc Transfer Request Subroutine (DTR)

DTR requests transfers between drum or disc and core memory. The core address of the 3-word transfer command and the program number are stored in the Driver Table. DTR may return to the calling program immediately after the request is stored or after the completion of the actual transfer.

To return to the calling program immediately after the request is stored in the Driver Table,



```
        SPB   DTRC01
        LDK   DRMXFR (May be Indexed)
              Error Return (Driver Table
              Normal Return            Full )

DRMXFR  DEL   1,/12000
        FØR   1,100
        LDA   /25000
```

DRMXFR = Symbolic Address of three-Word Transfer Command
DEL D, Bulk Address
FØR Controller No., No. of Words
LDA Symbolic Core Address

Example:

No.		Controller (Drum or Disc)
0	=	First
1	=	Second
Etc.	=	Etc.

The core address of the 3-word transfer command and the transfer direction are stored as one word in the Driver Table. The second word in the Driver Table contains the program number and the drum/core number (3Xn).

To return to the calling program after the requested transfer has been completed,

```
SPB  DTRC02
LDK  DRMXFR (May be Indexed)
     Error Return (Driver Table Full)
     Normal Return
```

The Bulk Transfer Driver Program will process disc transfers as though all disc storage (within a controller) is continuous addressing. The programmer is not concerned with data or programs physically split between disc platters.

8.1.10 Drum/Disc Transfer Driver (DTD)

DTD initiates transfers between core memory and drum or disc. It is entered from the Drum Transfer Complete Interrupt.

Following each interrupt, a transfer is initiated by an ØUT Drum/Disc Command. Return from DTD is to the interrupted program or the ECP. (Refer to 1.2).

8.1.11 Find Register Pointer Subroutine (FRP)

FRP determines a program's type of register storage. It gives the starting address of the 8-word storage block or the next entry address for the specified program.

```
     LDA  Program Number
     SPB  FRPC01
          Returns with the address in the A-Register
          and Test Flip-Flop Status
```

The test flip-flop is set if the program has full register storage. Otherwise, it is reset.

See Register Pointer Table under 1.1.

8.1.12 Peripheral Availability Subroutine (PAV)

The Peripheral Availability Subroutine is used as a combination call to check for peripheral availability, make an output request, or perform peripheral substitution. When a peripheral is not free, appropriate diagnostic action is taken or peripheral substitution is made if the peripheral is bad or out-of-service.

PAV also provides an optional feature for checking data areas. A data area is assigned to each output message representing the drum, disc, or permanent core area containing the message. When an output request is made for availability and the area is available, the area is set unavailable and is set free when the last character of the output message is completed.

Data area numbers range from 1-24 (23 and 24 are reserved for Monitor). Zero is reserved as a dummy area number when availability is not required. Area number zero is always considered available. The data number is used as a tag to identify a message area. The tag may apply to an individual message or a buffer which that message occupies. Two messages which share the same buffer may not have the same tag. Until the driver removes the data tag, no other program can use this tag.

A peripheral availability call must not occur in a subroutine used by more than one functional program. The PAVC01 call selects an alternate output device when the requested device is not free.

Fixed Message Calls

```
           SPB  PAVC01   (Normal Request) or SPB PAVC04 (Priority
                         Request)
           DEL  0,FØRMAT
           DEL  0,DATA          No Test for Data Area Required
           FØR  DEVICE,0           0 = Dummy Area Number
                Return

  DEVICE   EQL  3

Core Format
and Data
Locations
```

Note: A fixed message call may be used if the format and data words are built at assembly time. A priority call places the request at the top of the stacking list.

Format and data words on drum or disc must be in the lower 256K of a drum or disc system.

Peripheral Availability (cont'd.)

REASON DEVICE IS UNAVAILABLE	ACTION
Stacking List Full	Delays and tries again
Device is Bad or Out-of-Service	Request output on alternate device.
No Alternate is Available	Bypasses output request.

```
                 Message Number

      SPB  PAVC02                    (Checks stacking list  and data area
      FØR  DEVICE,AREANØ              availability)

                .
                .
                .
    Place the Output
    Data in the Table                     Return is made when the
                                          area is available and the
                                          device's stacking list
                .                         is not full.
                .

      SPB  PAVC03           (Requests output and places request in
      SPB  ØUTC02            Stacking List)
      DEL  A,FØRMAT
      DEL  A,DATA
      FØR  DEVICE,AREANØ
           Return

  Drum        Core
  Address     Address

  A = 0 = Core Address
      1 = Drum Address
```

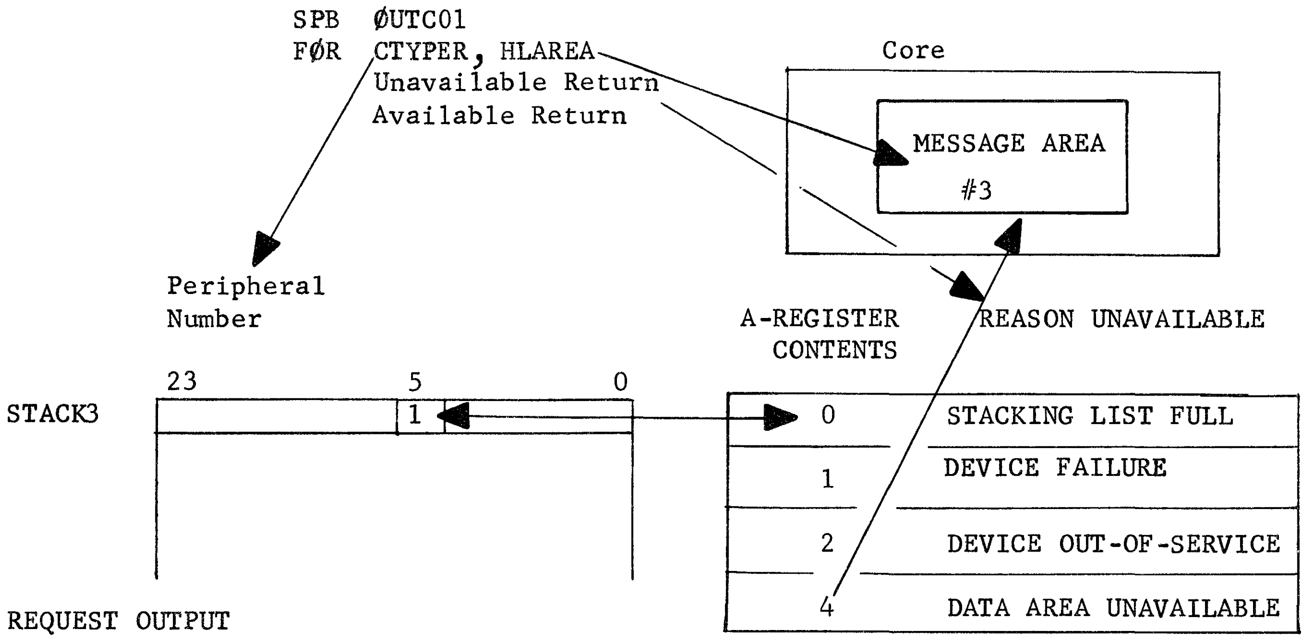
8.1.13 Output Subroutines (ØUT)

ØUT is composed of three functions. The ØUTC01 call is used to test for peripheral availability and data area status. The ØUTC02 (Normal Request) and ØUTC03 (Priority Request) calls make the output request. The given data and format locations are stored in the proper tables and the peripheral ALERT bit is set for the Output Program.

When the output request returns to the "available return" (data area and peripheral free), the system program may then assemble its data in the data area in preparation for the actual output request. For every "available return", there must be an associated output request.

Output Requests may be made for the printer, typewriter (including I/O), paper tape punches, and magnetic tape.

To Test Peripheral Availability and Data Area Status,



```
SPB  ØUTC01
FØR  CTYPER, HLAREA
     Unavailable Return
     Available Return
```

A-REGISTER CONTENTS	REASON UNAVAILABLE
0	STACKING LIST FULL
1	DEVICE FAILURE
2	DEVICE OUT-OF-SERVICE
4	DATA AREA UNAVAILABLE

REQUEST OUTPUT

```
SPB  ØUTC02    (Normal Request)
DEL  A,FORMAT  Address of 1st Format Word in the Format Table
DEL  A,DATA    Address of 1st Data Word in the Data Table
FØR  CTYPER,HLAREA
     Return
                    Data Area No. 3

     Peripheral
        Number
Core Addresses

     HLAREA  EQL  3

A = 1 = Drum Address
    0 = Core Address
```

Note: Format and data words on drum or disc must be in the lower 256K of a drum or disc system.

Output Tables

```
    SPB   ØUTC03  (Priority Request)
    DEL  {A,FORMAT
   |DEL  {A, DATA
   |FØR   CTYPER, 0
   |      \___ ___/
   |          V
   |        Return
   |
Core
Address

CTYPER    EQL 2
```

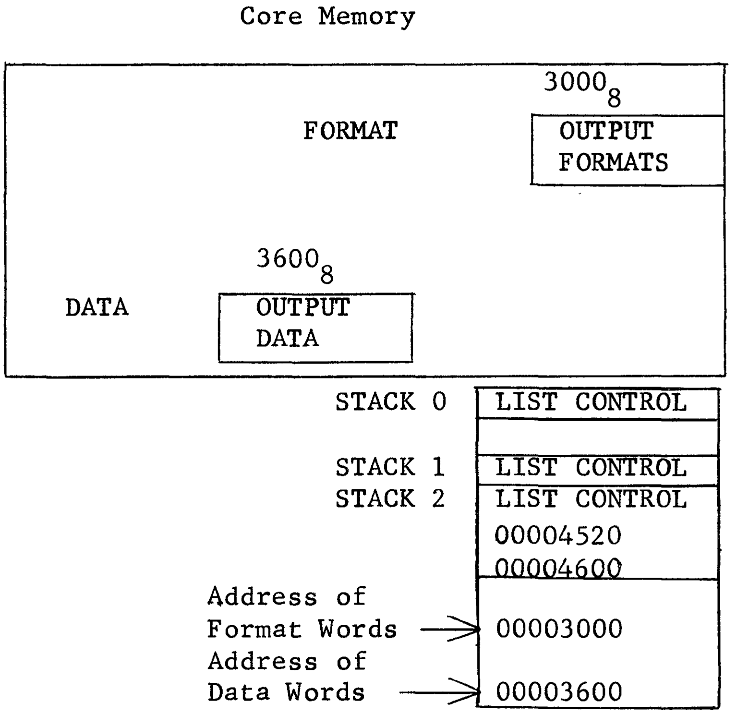
In the device code table (DVCØDE), the octal address is deciphered as follows:

Example: 60101100_8
The first two digits are called the Type Code (60). Type codes are used to specify the following peripherals:

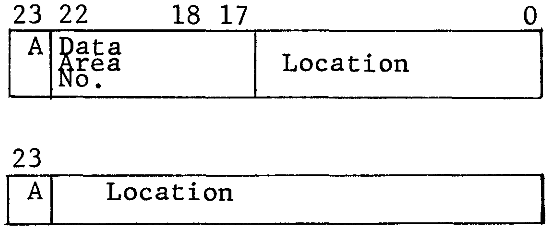
40 - Paper Tape Punch or Output Typewriter
42 - Card Punch
44 - High Speed Printers
50 - I/O Typer
60 - Paper Tape Reader or 70 CPM Reader
76 - 350 CPM Reader

The third digit is the classification code which specifies the peripheral number.

0 = Printer #1
1 = Printer #2
ETC.



Request for Output

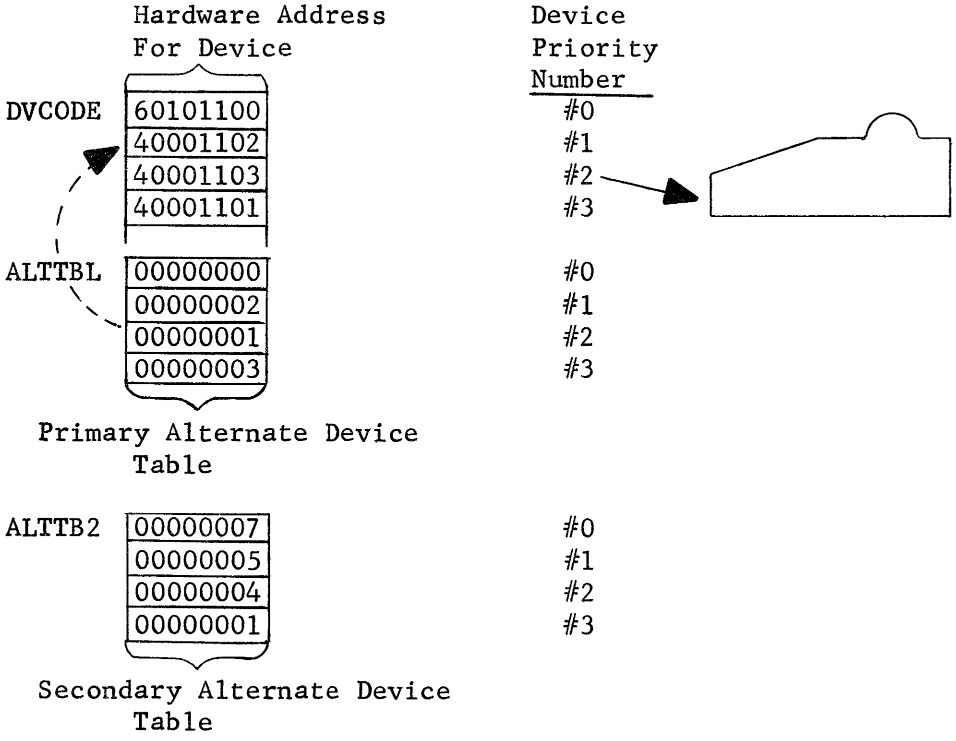


A = 0 = Core Address
1 = Drum Address

The classification code is a running number (0-7) for input peripherals (type Codes 50, 60, and 76). The 350 CPM Readers should have the highest priority classification code.

In the code, the last four digits specify the hardware address for that peripheral.

Format and data words must be stored on drum or in permanent core. Format words and data words are stored in separate tables. Addresses given in the ØUTC02 and ØUTC03 calls must be absolute symbols. Either common (*) or local absolute (-) may be used. Absolute addressing must be used in drum or disc systems only.



	Hardware Address For Device	Device Priority Number
DVCODE	60101100	#0
	40001102	#1
	40001103	#2
	40001101	#3
ALTTBL	00000000	#0
	00000002	#1
	00000001	#2
	00000003	#3

Primary Alternate Device Table

ALTTB2	00000007	#0
	00000005	#1
	00000004	#2
	00000001	#3

Secondary Alternate Device Table

8.1.14 Output Program (IØP)

Monitor outputs information by using a group of related routines and subroutines within the Output Program. The Output Program is turned on after receiving an output request from the Output Subroutines.

The stored format word is decoded into separate fields. Format words which are stored on drum or disc are transferred to an eight-word core buffer for processing. Data which is stored on drum or disc is placed in a sixteen-word core buffer. Both buffers are within the Output Program's temporary storage (save) area. The proper conversion routine, as indicated in Bits 23-21, is then entered. Each conversion routine is discussed as subcategories of 1.14:

8.1.14 Output Program (IØP) (cont'd.)

1.14.1 Binary to Fixed-Point Decimal
1.14.2 Binary to Four-Bit BCD
1.14.3 Binary to Six-Bit BCD
1.14.4 Binary to Non-Edited Character
1.14.5 Binary to Octal
1.14.6 Clock Output
1.14.7 Floating Point to E-Type Floating Point Decimal
1.14.8 Floating Point to Fixed Point Decimal

The following subroutines are also contained within the Output Program to assist in placing the converted information in the Driver Table printing error messages (EEE*,999*,000*, *) etc. They are:

Store Character in Driver Table Subroutine
Store Error Code in BCD Table Subroutine
Build Driver Table Subroutine
Error Subroutines
Update Index and Load Data Subroutine

Error typeouts indicate the following:

999*	-	Number too large
000*	-	Number too small
EEE*	-	Incorrect format word
*	-	Field size less than three

The End of Message word is all bits set $(77777777)_8$.
Every table of format words must contain an end word which is generated by:

```
CØN Ø,77777777
```

The repeat factor is used when more than one data word is associated with a format word. For example, if four data words use the same format, three is placed in this field. If only one data word is used, the repeat factor should be zero.

The multiplying factor positions the decimal point for typing under log headings.

Example:	Pressure of Boiler "A"	Temperature of Steel Furnace
Multiplying Factor	X10^{-3} lbs. (-3)	X10^{+2} degrees (+2)

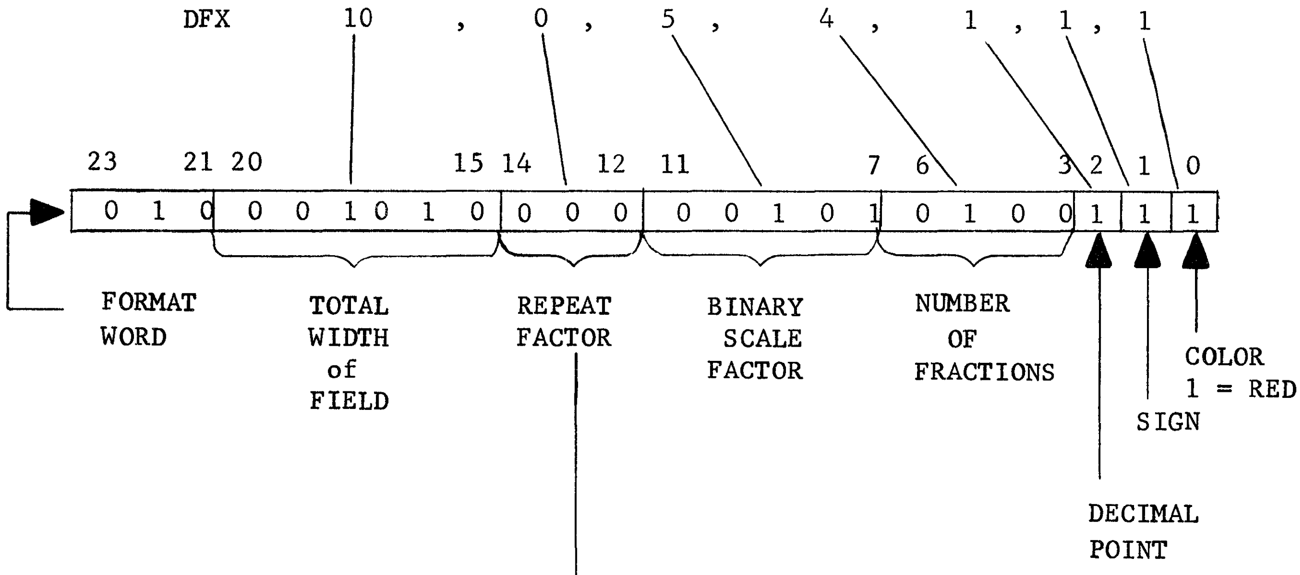
The format words shown in the following conversion routines include an example of the MONITOR Pseudo-operation. These pseudo-op instructions can only be used when assembling with GE/PAC Monitor or the GE/PAC Monitor EQL tape.

8.1.14.1 Binary to Fixed-Point Decimal (DFX41)

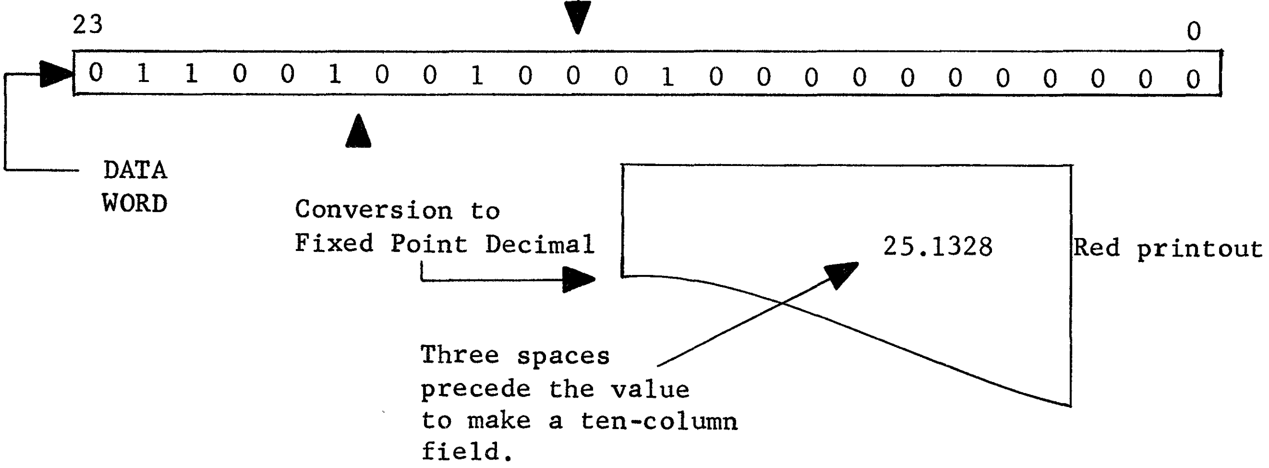
DFX41 converts information from binary fixed-point to decimal fixed-point. Leading zeros are automatically suppressed. Numbers exceeding the field size are typed as all nines followed by an asterisk.

Binary to Fixed-Point Decimal

Pseudo-op Example:



One value, scaled B5, is typed in a ten column field. The values are printed in red with decimal point and four fractional digits. The sign is printed if the value is negative.

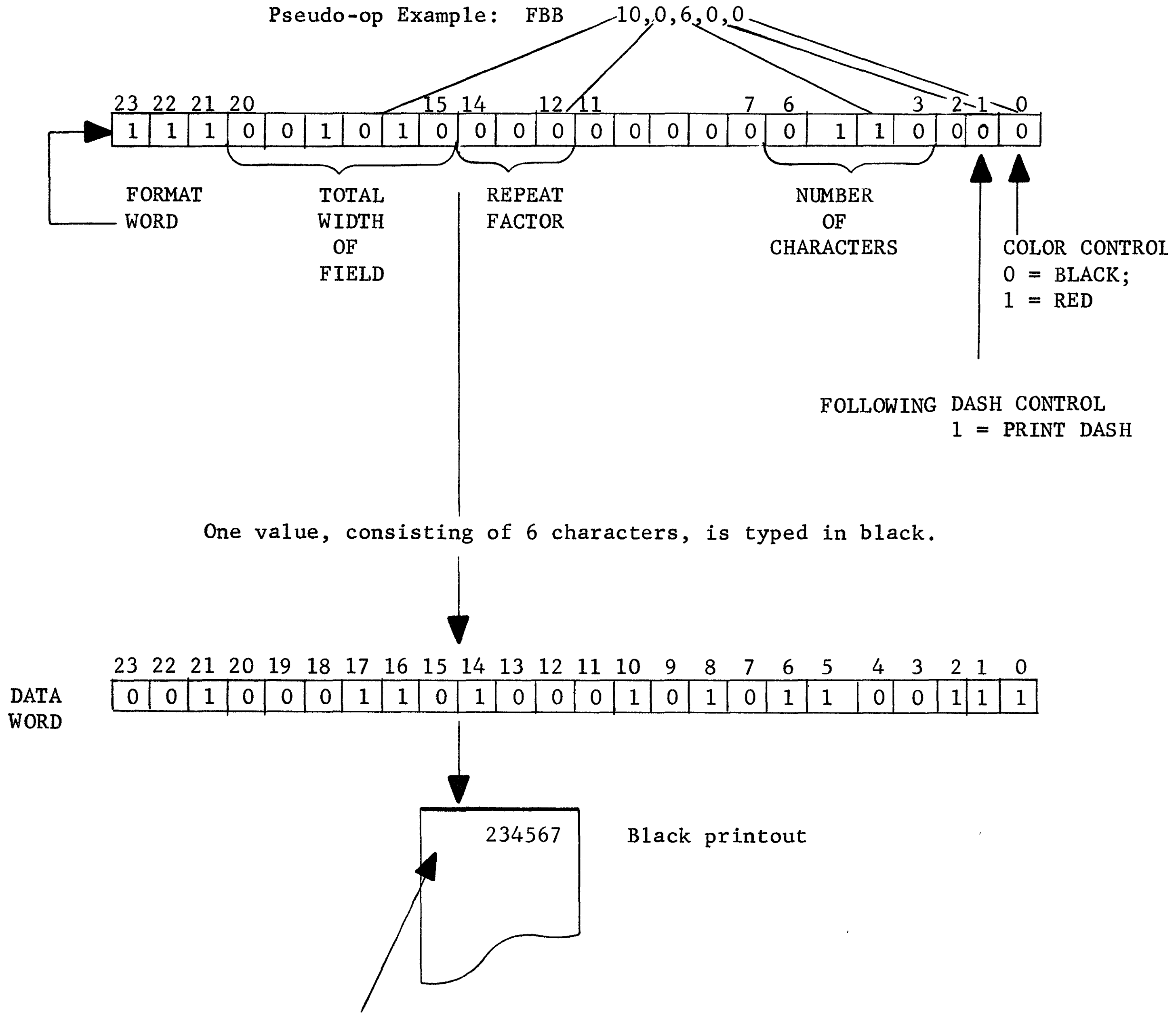


23
0
0 1 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
DATA WORD
Conversion to Fixed Point Decimal
25.1328
Red printout
Three spaces precede the value to make a ten-column field.

8.1.14.2 Binary to Four-Bit BCD (FBB41)

FBB41 converts binary data, left justified, to six 4-bit BCD characters per word. After the conversion, each BCD character is stored as one word in the Output Driver Table.

Binary to Four-Bit BCD



One value, consisting of 6 characters, is typed in black.

4 blank spaces = ten column field

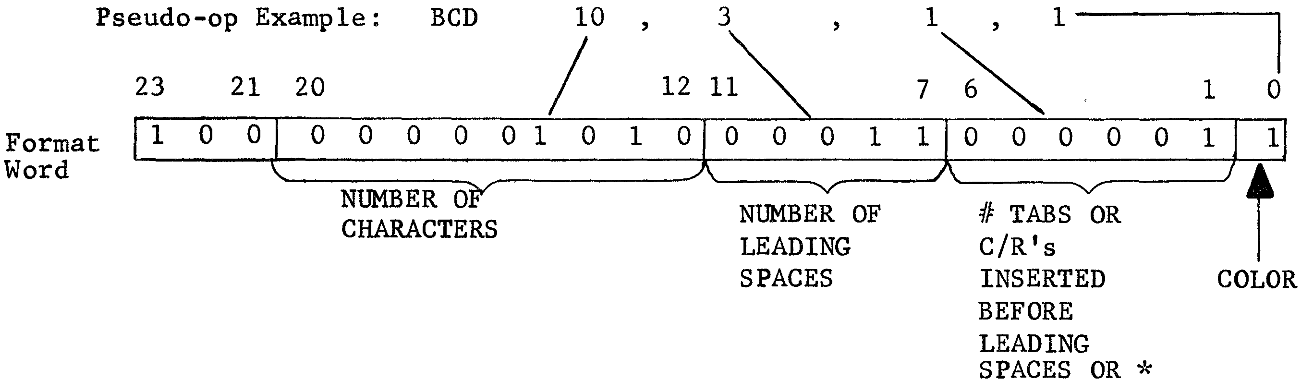
Note: If the repeat factor is used, each data word must have the same number of characters.

8.1.14.3 Binary to Six-Bit BCD (4 Characters Per Word)

BCD41 converts binary information to six-bit BCD characters. The data word contains six-bit BCD characters, left justified. After conversion, each word contains four BCD characters.

<u>Binary to 6-Bit BCD</u>

Pseudo-op Example: BCD 10 , 3 , 1 , 1



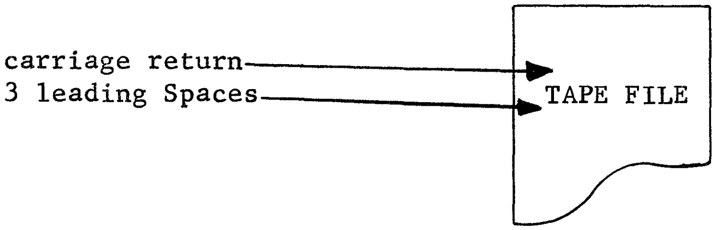
	23 21	20 12	11 7	6 1	0
Format Word	1 0 0	0 0 0 0 01 0 1 0	0 0 0 1 1	0 0 0 0 0 1	1
		NUMBER OF CHARACTERS	NUMBER OF LEADING SPACES	# TABS OR C/R's INSERTED BEFORE LEADING SPACES OR *	COLOR

One carriage return and three spaces are made before the BCD characters are typed.

Codes 01-17$_8$ = 1-15 consecutive carriage returns. Codes 41-57$_8$ = 1-15 consecutive tabs.

CØN A,9,TAPE FILE (pseudo-op generating data words)

	23 18	17 12	11 6	5 0
Data Words	1 1 0 0 1 1	0 1 0 0 0 1	1 0 0 1 1 1	0 1 0 1 0 1
	0 1 0 0 0 0	0 1 0 1 1 0	0 1 1 0 0 1	1 0 0 0 1 1
	0 1 0 1 0 1	0 1 0 0 0 0	0 1 0 0 0 0	0 1 0 0 0 0



Alphanumeric typeout in red.

*Vertical Page Control Code for Printer (Octal)

00	-	No Advance
01	-	Slew 1 line (Single Space)
02	-	Slew 2 lines (Double Space)
11 12 13 14 15 16 17 20		Slew paper to control channel 1 through 8 respectively (1 is considered top of next page)

8.1.14.3 Binary to Six-Bit BCD (4 Characters Per Word) - cont'd.

By inserting 40_8 in the BCD (FORTRAN Option) word, the Output Program uses the first character of the data line as the form control. Control codes are as follows:

Octal Control Code	
20 (Blank)	Single Space
00	Double Space
01 . . . 20	Slew to Channel N
60 (Plus)	Suppress Spacing

8.1.14.4 Binary to Non-Edited Character (BCN41)

BCN41 stores characters (one character per word) into the Output Driver Table with no transformation. One character is contained in Bits 23-0 of each data word, right justified. The number of filled bits depends on the output device being used. The maximum number of non-edited characters permitted in one BCN format word is 63.

Binary to Non-Edited Character

Pseudo-op Example: BCN 12

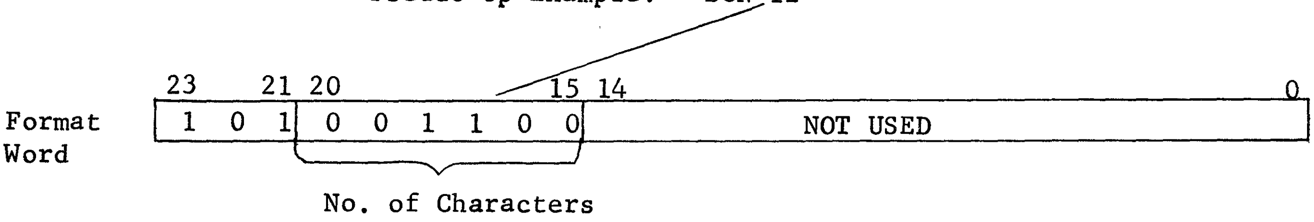


Table of Characters

23	7	6						0
0	0	1	0	1	0	0	1	1
0	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	1	0
0	0	0	1	1	1	1	0	1
0	0	0	1	0	0	0	0	1
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	1	1
0	0	0	1	0	0	1	0	1
0	0	1	0	0	0	0	1	1
0	0	0	0	0	0	1	0	0
0	0	0	1	1	1	1	1	0

Punches 12 characters with no transformation.

8.1.14.4 Binary to Non-Edited Character (BCN41) cont'd.

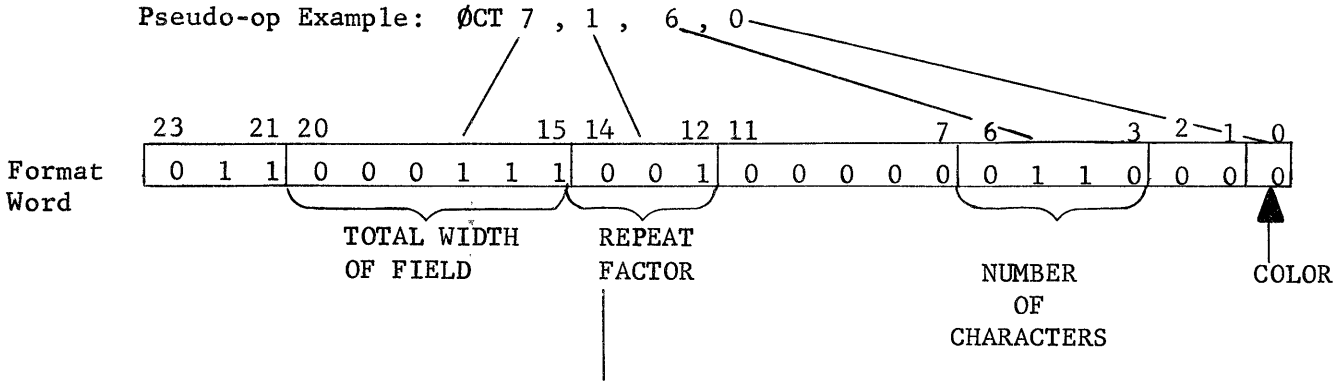
	CHANNEL NUMBERS	DATA FRAMES #1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	
Paper Tape	8	0									0			Hundreds Pos.
	7		0		0	0				0			0	2nd Tens Pos. 4
	6	0			0				0				0	2nd Tens Pos. 2
	5	0				0		0	0					Parity Ind.
	4		0		0		0		0				0	2nd Tens Pos. 1
	●●●●●●●	●	●	●	●	●	●	●	●	●	●	●	●	Sprocket Feed
	3		0		0					0		0	0	1st Units Pos.
	2	0		0					0		0		0	1st Units Pos.
	1	0			0	0			0	0	0			1st Units Pos.
		123	54	2	75	41	10	0	33	45	103	4	76	Pos. = Position

*Even frames require a parity punch which hardware generates on output, otherwise a parity flip-flop is set during input mode.

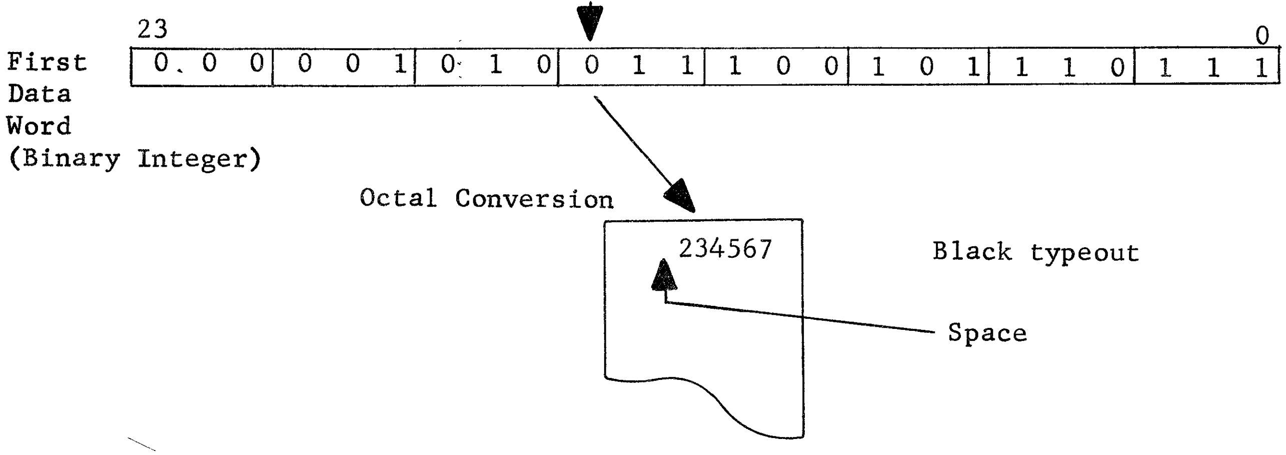
8.1.14.5 Binary to Octal (ØCT41)

ØCT41 converts binary integers to octal characters.

Binary to Octal



Six low order octal digits are typed in a seven-column field using black ribbon. The next data word would be processed in the same manner. (Repeat Factor 1).

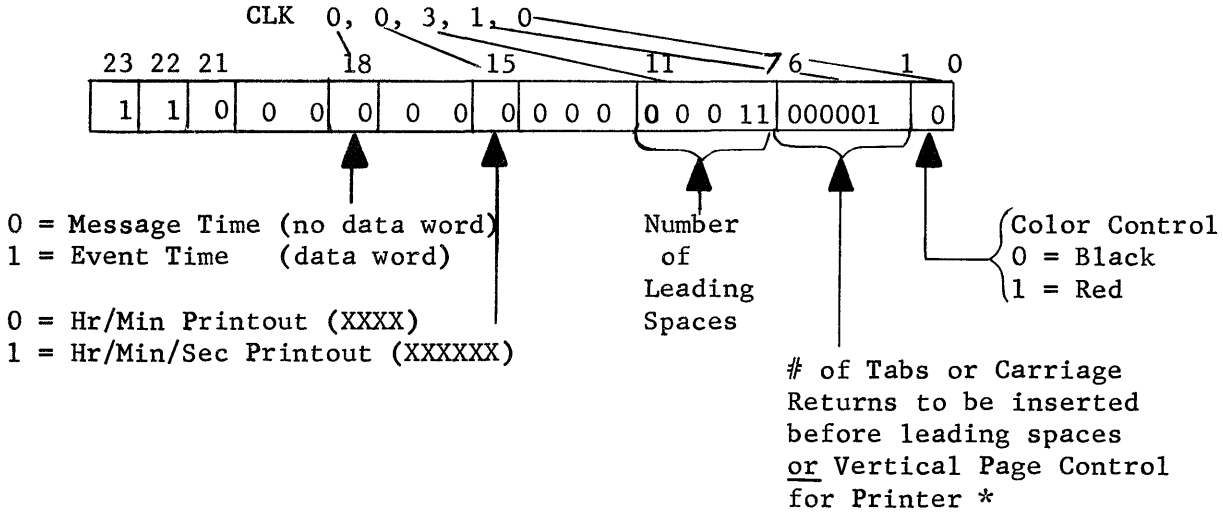


8.1.14.6 Clock Output (CLC41)

CLC41 converts the time of day from system time counts to decimal hours and minutes. It is then printed as four or six decimal digits.

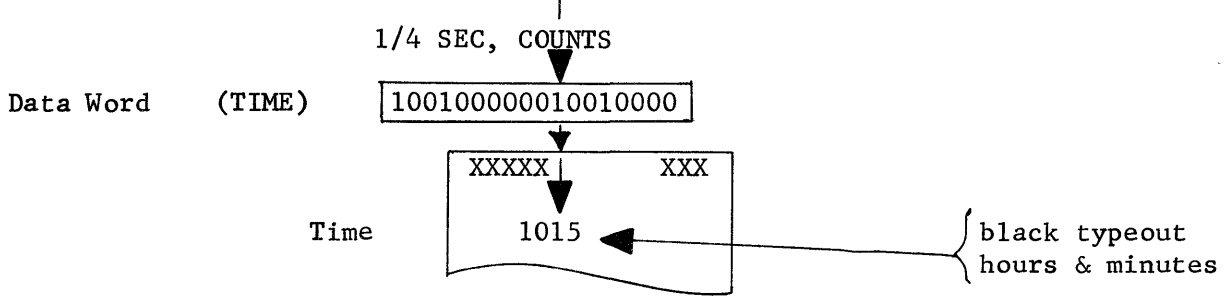
Time Count To Hours, Minutes and Seconds

Psuedo-op Example:


CLK 0, 0, 3, 1, 0
23 22 21 18 15 11 7 6 1 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 11 000001 0
0 = Message Time (no data word)
1 = Event Time (data word)
0 = Hr/Min Printout (XXXX)
1 = Hr/Min/Sec Printout (XXXXXX)
Number of Leading Spaces
# of Tabs or Carriage Returns to be inserted before leading spaces or Vertical Page Control for Printer *
Color Control
0 = Black
1 = Red

Use Octal Codes 01-17 for the inserted number of consecutive carriage returns (1-15). The inserted number for 1-15 consecutive tabs are octal codes 41-57.

The above example types or punches time in hours and minutes preceded by a carriage return and three spaces using black ribbon.



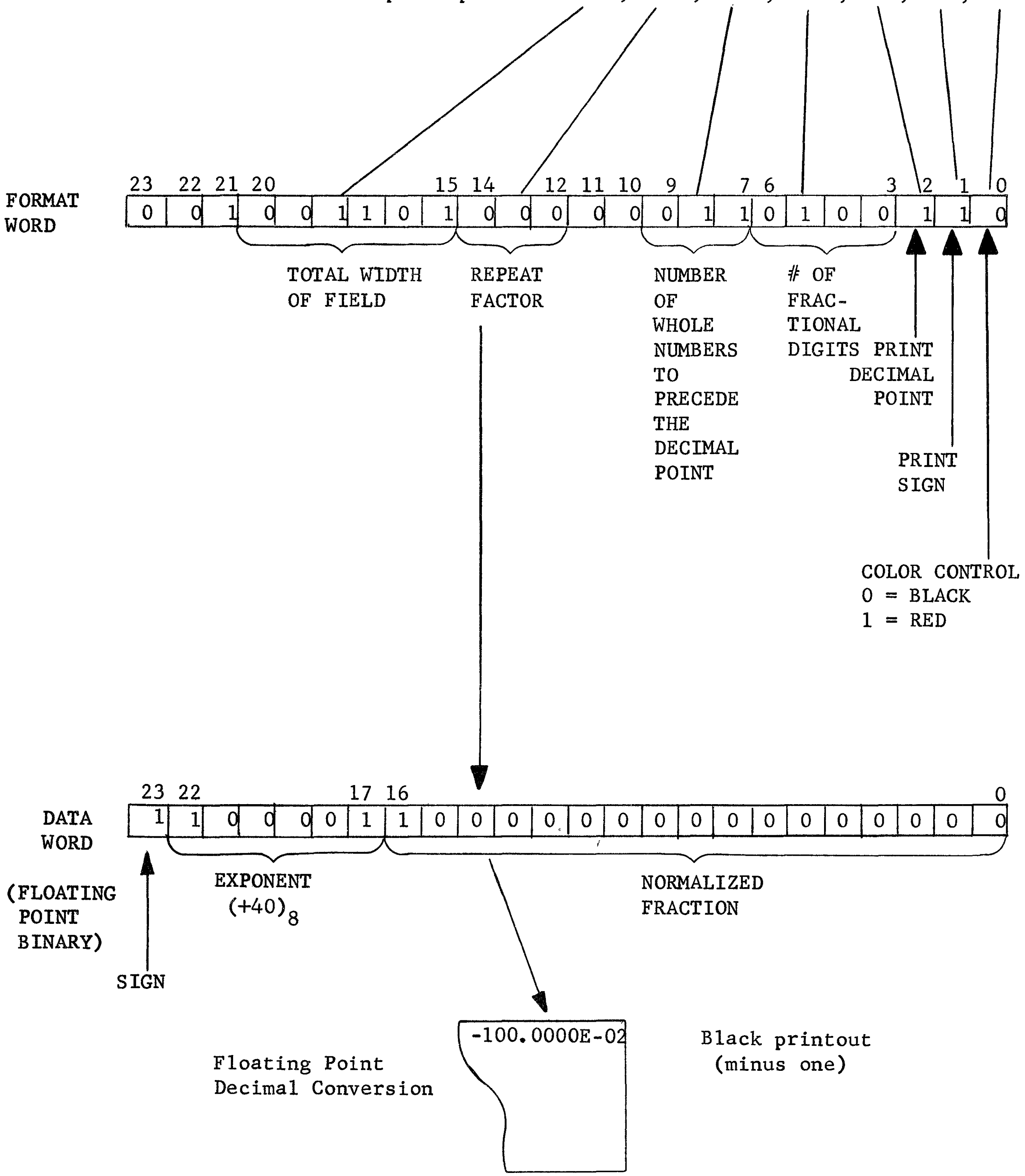
* Please refer to page 1-22 for Vertical Page Control Code for Printer.

8.1.14.7 Floating Point to E-Type Floating Point Decimal (DFE41)

DFE41 converts binary floating point information to decimal floating point.

<u>Floating Point to Floating Point Decimal</u>

Pseudo-op Example: DFE 13 , 0 , 3 , 4 , 1 , 1 , 0

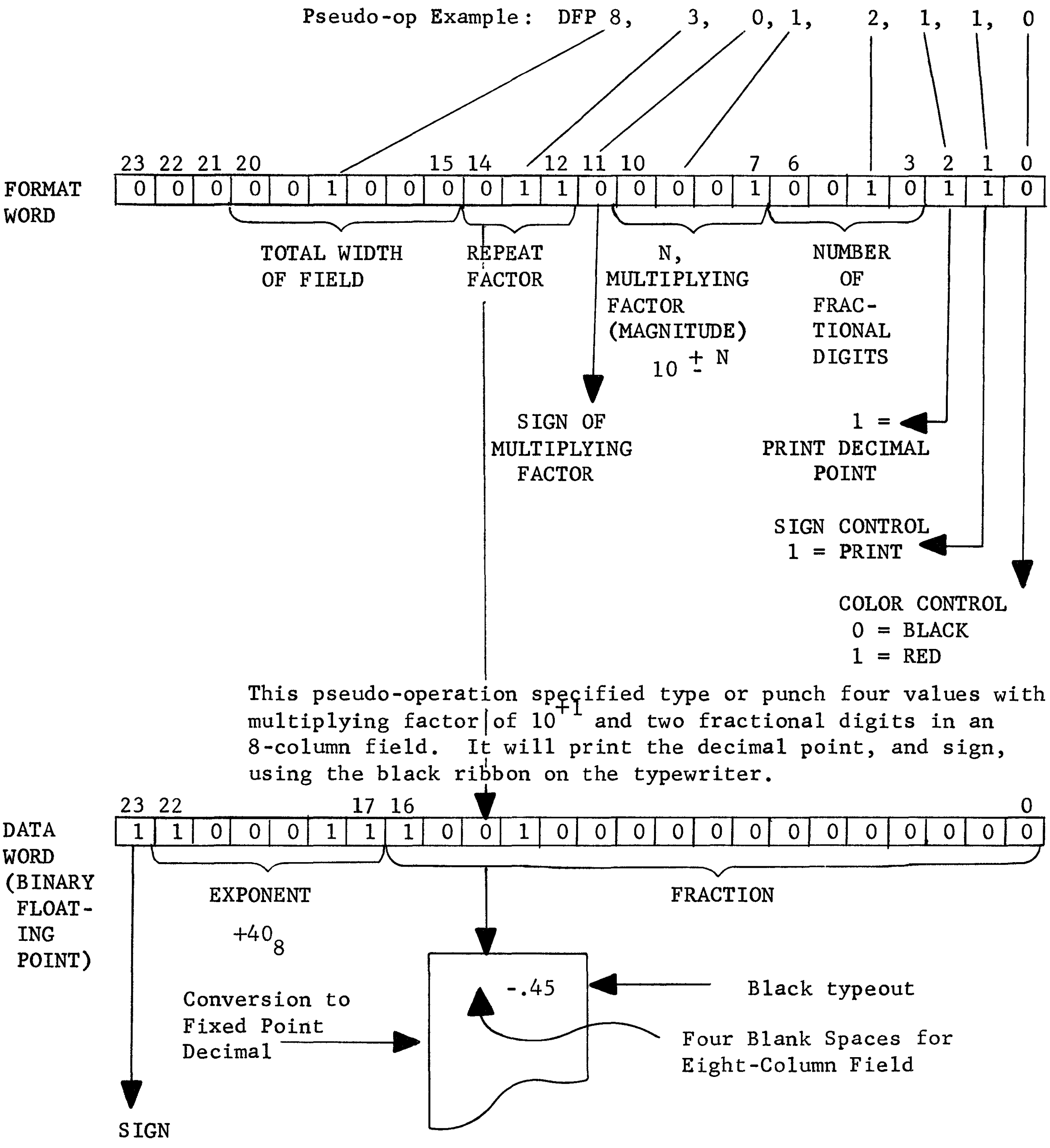


8.1.14.8 Floating Point to Fixed Point Decimal (DFP41)

DFP41 converts binary floating-point information to decimal fixed-point. The floating point number is multiplied by 10^N if N is negative, or divided by 10^N if N is positive. (N is the multiplying factor, maximum range of absolute 9, located in the format field.) Leading zeros are automatically suppressed.

The permitted range for any number is $< 2^{24}$ or $> 2^{-23}$

Floating Point to Fixed Point Decimal



This pseudo-operation specified type or punch four values with multiplying factor of 10^{+1} and two fractional digits in an 8-column field. It will print the decimal point, and sign, using the black ribbon on the typewriter.

8.1.15 Input Driver (IND)

The Input Driver is entered from any of the following interrupts:

1. I/O Device Ready
2. I/O Typer ØDL Echo
3. 350 CPM Card Reader IDL Echo
4. 350 Card Reader Ready Interrupt

IND initiates a card reader, paper tape reader, or I/O Typer request. Other functions accomplished by IND are:

1. Resets diagnostic counts for I/O Devices.
2. Tests for an Input Demand from the 350 CPM Card Reader.
3. Tests for a valid card read.
4. Notifies the Corrective Action Diagnostic Program of a photo-cell card reader error.

8.1.16 Input/Output Driver Program - I/O Typer (IØD)

IØD is turned on by the Input Driver (IND). Selection codes and output characters for the I/O typer are stored one character per word in the Output Driver Table. Input characters from the I/O Typer Paper Tape Reader or 70 CPM Card Reader, are stored one per word in the Input Data Table for the requesting program.

When the end-of-tape, stop code, requested number of characters has been read, or if the input list is full, IØD exits to the ECP.

For I/O typer input, an "IN" Command is given to unlock the keyboard. At the end of the record, IØD locks the keyboard and turns the Input Program on.

8.1.17 Output Driver (ØUD)

On the AU2, ØUD is entered from the ØDL Echo Interrupt when a peripheral device's Driver Table is empty. Flags are reset when there is no conversion in progress for output typers, paper tape punches, card punches, or high speed printers.

For the AU1, ØUD is entered from the Output Device Ready Interrupt when the previous output has been completed. Each device is assigned its own driver table and ØUD outputs the next character unless the driver table is empty. If the table is empty, flags are reset when there is no conversion in progress for the output typers and paper tape punches.

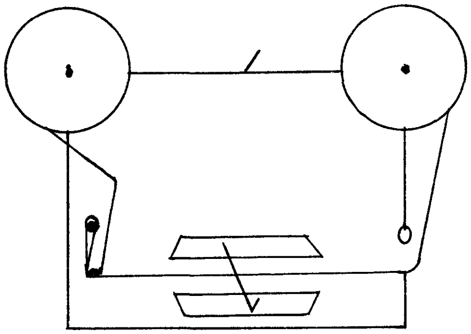
8.1.18 Input Request Subroutine (INS)

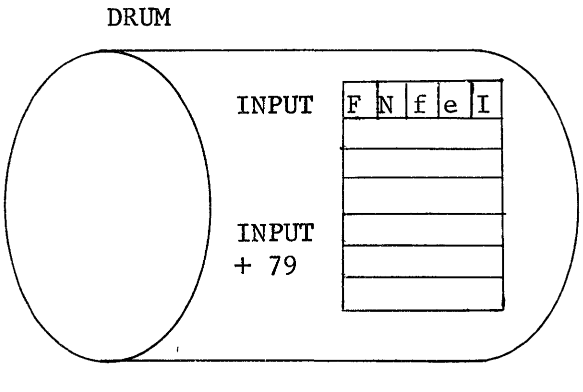
INS accepts input requests from the paper tape reader, card reader, or I/O typewriters. Input from more than one peripheral may be requested at any time. However, each device is limited to one request at a time. Each buffered request must be tested for completion of input by the calling function.

The following calling sequence is used for Buffered Requests. A buffered request returns immediately to the "normal return" of the calling sequence after the read is initiated. During this time, the calling function may process the characters placed in the alternate input list by the previous read request. In this manner, the calling program may process one list of characters while the Driver is storing characters in another table in core. (This procedure drives the input device near maximum speed.)

Buffered Input Request

```
                         Variable Length Message
          STOP CODE ⟶  A-REGISTER  OR  LDZ  (No Stop Character)

                SPB       INPC01
                DEL       1,INPUT ───▶ Starting Location of Data
                FØR       0,80 ◀──     No. of Characters to be
                                       read (0 indicates to read
Input Chars. ◀            Unavailable Ret. until the Input Data Table
on Drum                   Normal Return    is full - variable length)

                                              ▶ Device Bad,
                                                Out-of-Service, or
Input Device No. ◀                              in use.  See
                                                INC01 List.
```





READ COMPLETION REQUEST

```
SPB  INPC02
CØN  G, 0 ◄────── Device Number
     Bad Read Return
     Read Completed Return
```

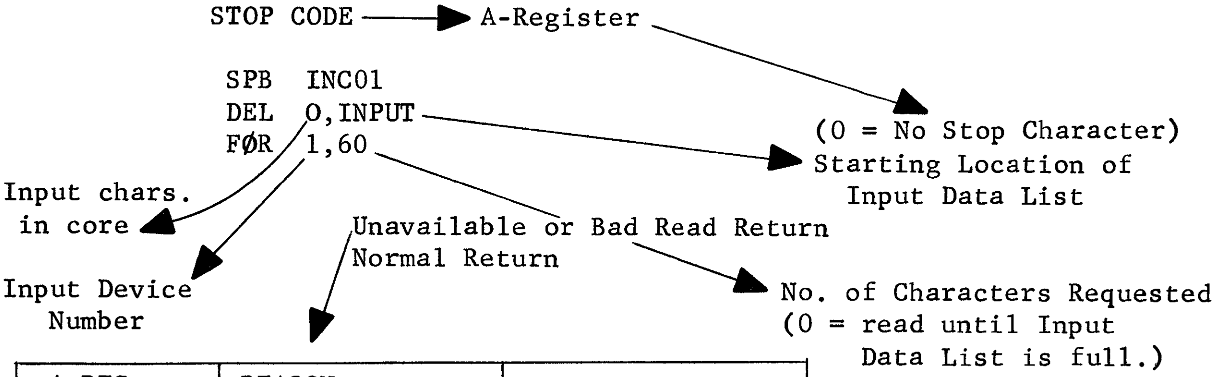
The requested number of characters has been read.
A "Stop Code" was encountered.
The end-of-tape was found.
The calling function's input data list is full.

A-REG CONTENTS	BAD READ REASON	POSSIBLE ACTION
0	Parity Error Aborted by Operator	Quit
bit 3	Device Failed During Read	Select Alternate Input Device
bit 4	Input Data List Overflow	Remove from list before list is full.
bit 5	No I/O Typer Stop Code	Quit
bit 6	I/O Typer Request Aborted	Quit
bit 7	I/O Typer Time Abort	Initiate another input request
bit 8	350 CPM Card Hopper Empty	Wait for operator to place cards in reader

8.1.18 Input Request Subroutine - Cont'd.

When a non-buffered input request is made, the calling function is locked out until the read is completed or an error is encountered.

Non-Buffered Input Request



```
STOP CODE ——▶ A-Register

 SPB  INC01
 DEL  0,INPUT
 FØR  1,60
```

(0 = No Stop Character)

Starting Location of Input Data List

Input chars. in core

Input Device Number

Unavailable or Bad Read Return

Normal Return

No. of Characters Requested (0 = read until Input Data List is full.)

A-REG CONTENTS	REASON UNAVAIL.	ACTION
bit 0	Device Bad	Select Alternate
bit 1	Device Out-of-Service	Select Alternate
bit 2	Device in use	Set Delay
bit 3	Device Failed During Read	Select Alternate
bit 4	Input Data List Overflow	Remove from list before list is full.
bit 5	No I/O Typer Stop Code	Quit
bit 6	I/O Typer Request Aborted	Quit
bit 7	I/O Typer Time Abort	Initiate another input request.
bit 8	350 CPM Card Hopper Empty	Wait for operator to place cards in reader
zero	Parity Error Aborted by Operator	Quit

A flex code 177_8 acts as a true delete code and is skipped when reading paper tape. The delete code should not be used as the "stop" character. The stop code is used as the end-of-record indicator. For the I/O typer, the right bracket is the end-of-record indicator.

After the first character has been read on paper tape, ten consecutive blanks serve as end-of-tape.

8.1.19 Input Program (INP)

After an input request has been completed, INP notifies the calling function of the completion. It also performs the following actions - resetting the flags for the input device, testing the "end" code of the input record, and notifying the calling function of an error.

The following error messages may be initiated:

1. NO STOP CODE (I/O Typers only - Bit 5 is set in the A-Register, and INP returns to the calling function at the "bad read" return.)

2. DEVICE n PARITY is typed for a paper tape reader parity error followed by the message, ENTER DATA VALUE XXX or]. The operator may correct the error by typing the appropriate three characters on the I/O Typer or abort the request by typing a right bracket. For systems without I/O typers, the operator may correct the error by placing the octal character in the console switches and pressing the "demand" button. To abort, lower switch 22 only, and press "demand".

   For an abort request, INP sets the A-Register to zero and enters the calling function at the "bad read" return.

An "end" code (stop, abort, or end job character) is removed from the Input Data Table.

8.1.20 Multiple Output Request Subroutine (MØR) - (Local Terminals Only)

MØR permits requests for analog, decimal, or binary contact status to be made for the Multiple Output Distributor. MØR has the provision for normal, timed, and pulsed requests.

Non-priority or priority calls can be made for each Multiple Output Request. In addition, an unlatch call can be made for a timed request. The required information must be loaded in the A and Q-Registers before the subroutine call is made.

Information for each request is placed in its respective Priority or Non-Priority Driver Table. However, no information is placed in the Driver for a request to unlatch a previous request. In this case, a previous request is aborted before the requested time has elapsed.

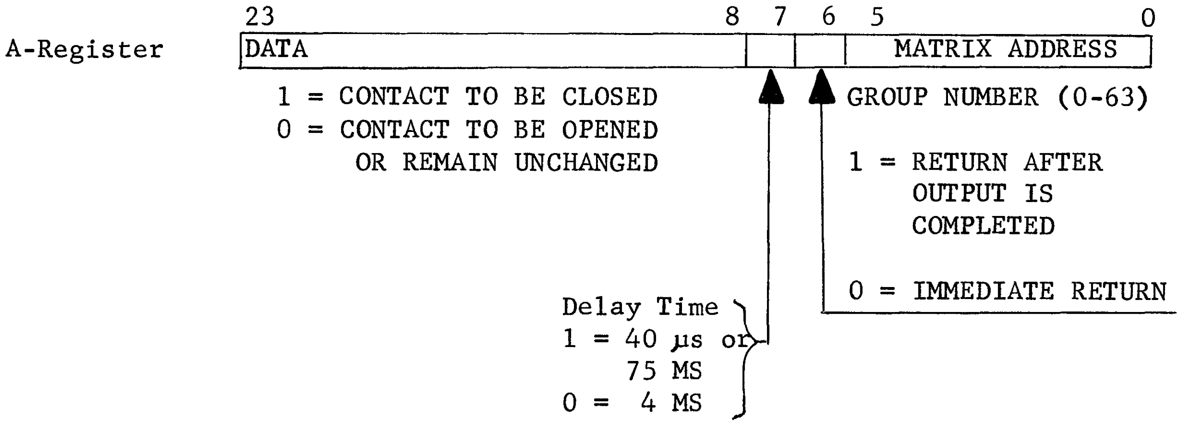
The operation delay time is the time required to transfer a command word from the A-Register to the MØD Command

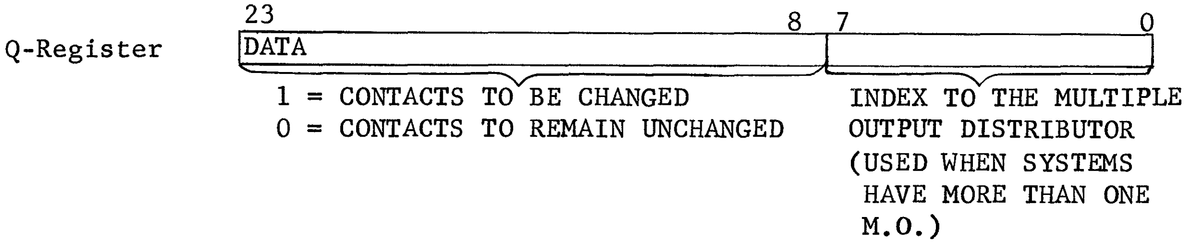
Multiple Output Request Subroutine (MØR) (Cont'd.)

Register and to initiate the transfer of data portion of the command word to the output function specified by the matrix address.

The error return indicators in the A-Register are the same for all subroutine calls.

Normal Requests

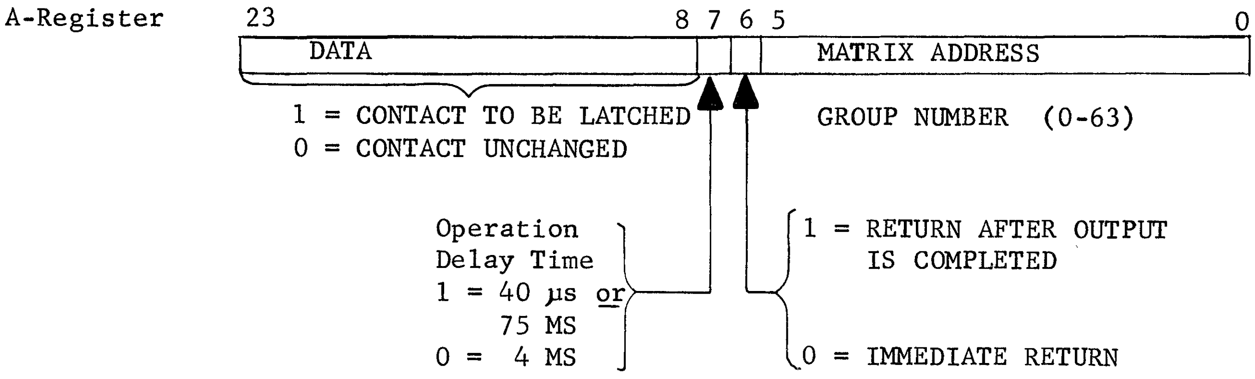


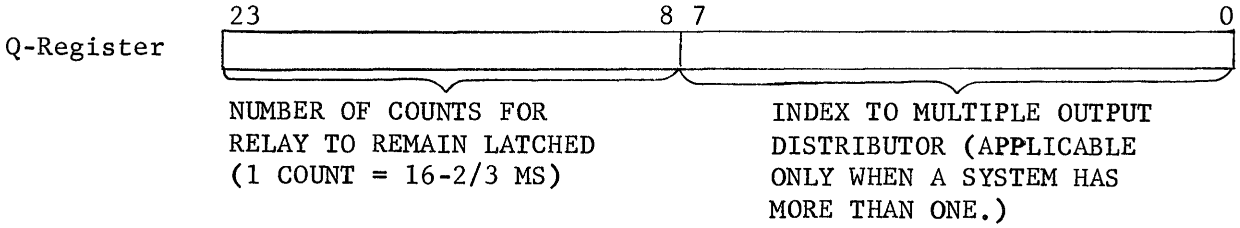


SPB MØRC01 (Non-Priority Request) or SPB MØRC02 (Priority Request)
Error Return
Normal Return

A-REGISTER CONTENTS	ERROR	POSSIBLE SYSTEM ACTION
0	DRIVER LIST FULL	Delay and try again.
1	GROUP FAILED PREVIOUSLY	Quit - try next output.
2	OUTPUT FAILED (OVERLOAD)	Hardware condition (2 groups addr'd.)
4	INVALID GROUP ADDRESS OR MØD INDEX	Programming Error.
8	M.O. TIMER FAILURE	No output may be done.

Timed Request



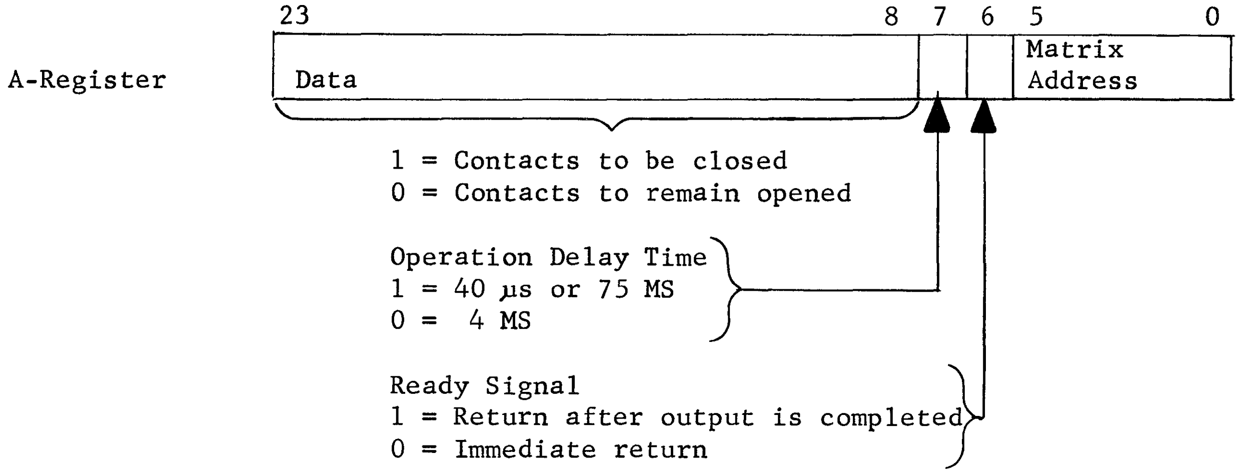


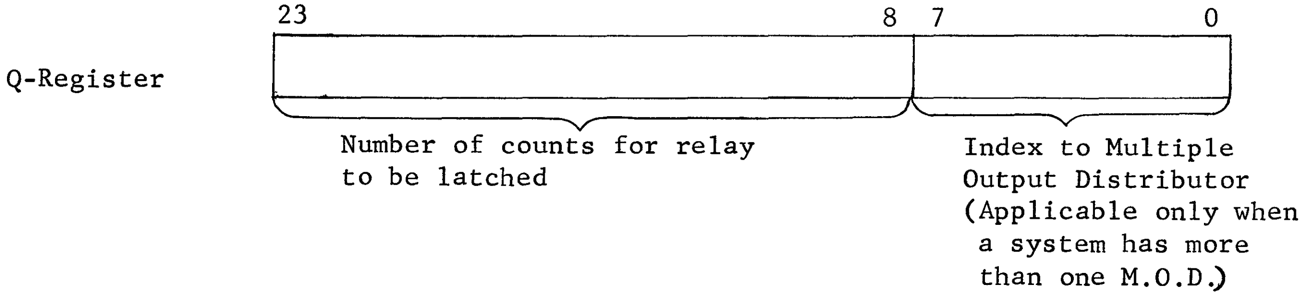
SPB MØRC03 (Non-Priority Request) or SPB MØRC04 (Priority Request) or
SPB MØRC05 (Unlatch a Previous Request)

Error Return
Normal Return

A-REGISTER CONTENTS	ERROR	POSSIBLE SYSTEM ACTION
0	NO TIMER AVAILABLE OR STACKING LIST FULL	Delay and try again.
1	GROUP FAILED PREVIOUSLY	Quit - try next output.
2	OUTPUT FAILED (OVERLOAD)	Hardware condition (2 groups addressed)
4	INVALID GROUP ADDRESS OR INVALID MØD INDEX	Programming Error
8	M.O. TIMER FAILED	No output may be done.

Pulsed Requests




23
8
7
0
Q-Register
Number of counts for relay to be latched
Index to Multiple Output Distributor (Applicable only when a system has more than one M.O.D.)

```
SPB    MØRC06  (Non-Priority Request) or

SPB    MØRC07  (Priority Request)
       Error Return
       Normal Return
```

8.1.21 Multiple Output Distributor Driver (MDR)

MDR is initiated by the MØD Interrupt to execute requests made by MØR. To output a request, the command word from the driver table is placed in the A-Register. The data is transferred to the location specified by the group address of the command word. The Multiple Output Distributor is then ready to accept another request.

All output commands are initiated from the Priority Driver Table before information is taken from the Non-Priority Driver Table.

For timed requests, a timer is activated. When the requested time has expired, MDR executes the unlatch command.

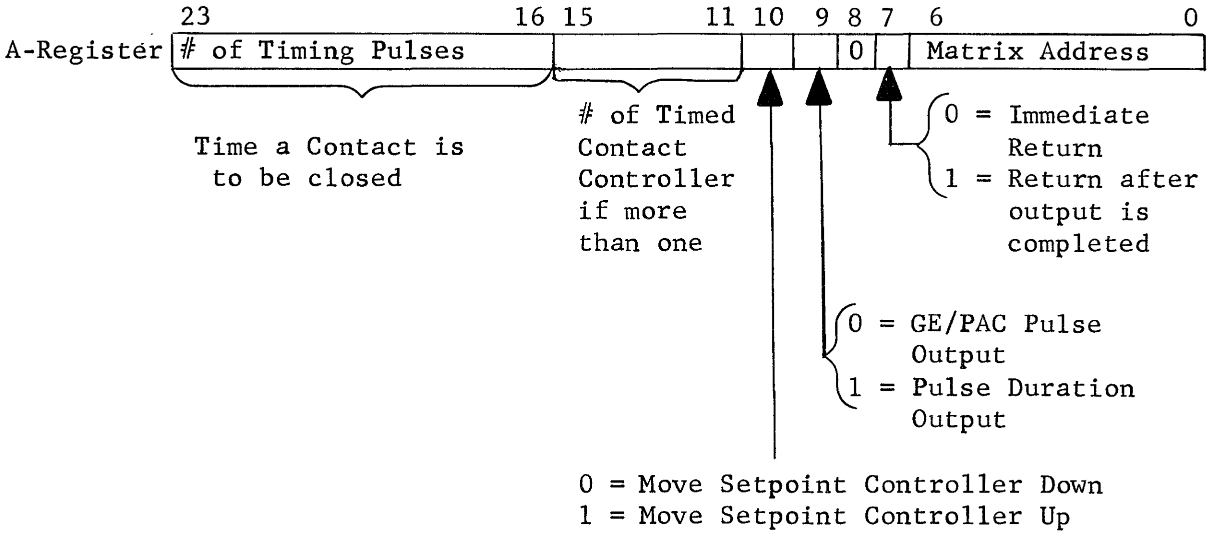
For pulsed requests, a pulsed timer is started. When it expires, the contacts are opened.

8.1.22 Timed Contact Output Request Subroutine (TCØ) - (Local Terminals Only)

The Timed Contact Output Request Subroutine supplies the communication link for making requests through the Timed Contact Output Controller. The TCØ Output Command Word is stored in the Normal or Priority Output Driver Table.

Timed Contact Output Request

TCØ Command Word



23
16 15
11 10
9 8 7
6
0
A-Register
# of Timing Pulses
0
Matrix Address
Time a Contact is to be closed
# of Timed Contact Controller if more than one
0 = Immediate Return
1 = Return after output is completed
0 = GE/PAC Pulse Output
1 = Pulse Duration Output
0 = Move Setpoint Controller Down
1 = Move Setpoint Controller Up

SPB TCØC01 (Normal Request) or SPB TCØC02 (Priority Request)

ERROR RETURN

NORMAL RETURN

A-REGISTER CONTENTS	ERROR
0	Driver List Full
2	Requested Group has failed on 2 consecutive overloads.
4	Invalid Matrix Address
8	Timer Failure

8.1.23 Timed Contact Output Driver (TCD)

TCD uses the Timed Contact Output Controller for addressing and controlling timed contact output groups from the Arithmetic Unit. To output a request, TCD places the command word in the A-Register. The data is transferred to the location specified by the group address of the command word. The Timed Contact Output Controller is then ready to accept another request.

All output commands are initiated from the Priority Driver Table before information is taken from the Normal Driver Table.

8.1.24 Scan Request Subroutine (SCR) - (Local Terminals Only)

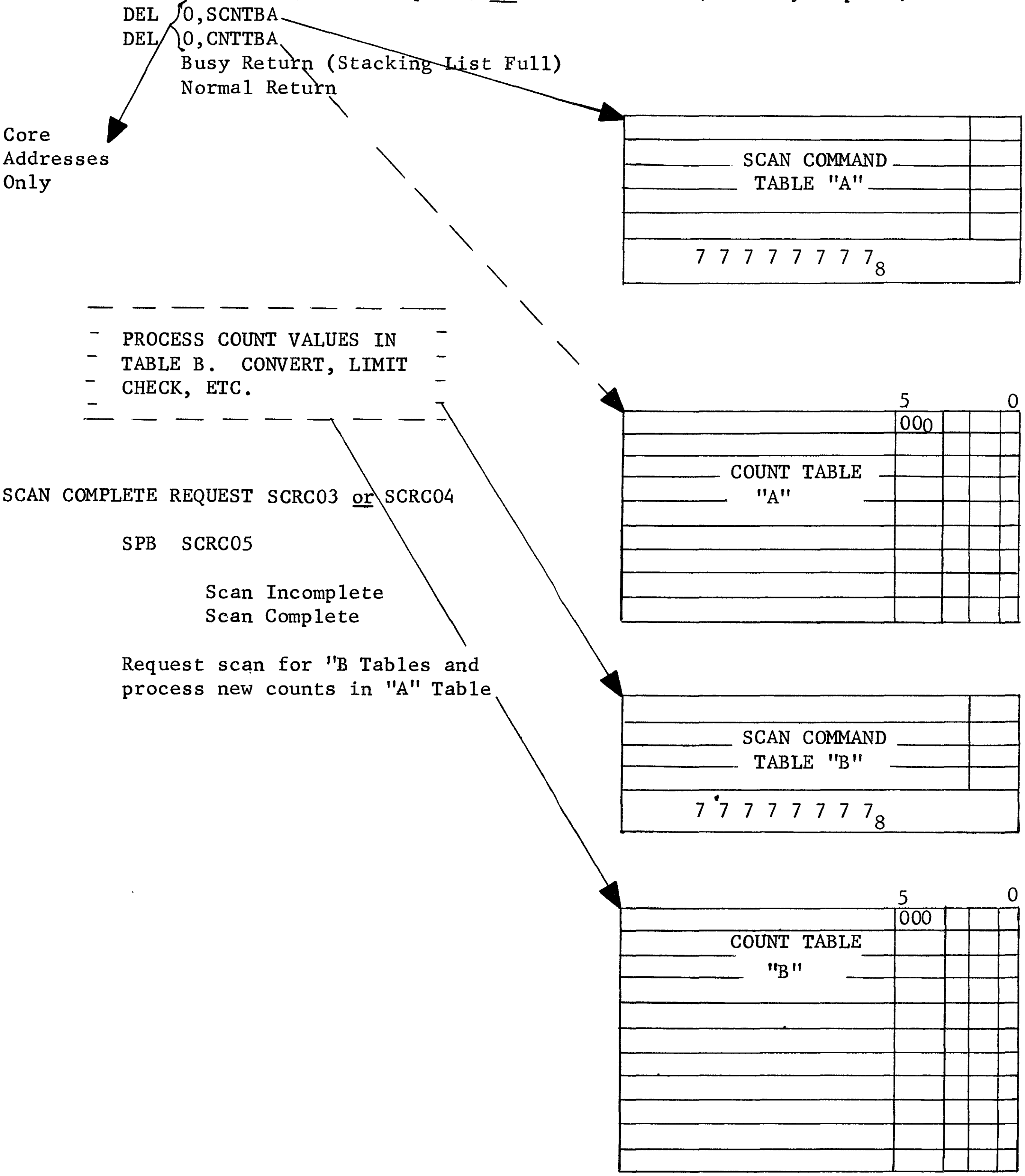
The Scan Request Subroutine stores the addresses of the Scanner Command Word and Count Value Tables in a stacking table. The Scanner Commands are executed by the Scan Driver. SCR process normal or priority, buffered and non-buffered scan requests. A normal request is processed on a first in/first out basis. Priority requests are processed on a last in/first out basis. The system has the option of storing time of the completed scan as last word of scan request count table.

8.1.24 Scan Request Subroutine (SCR)- cont'd.

Buffered requests permit the Scanner to be driven near maximum speed by allowing a functional program to process one set of count values while another group of analog values is being scanned for the same program.

```
SPB  SCRC03   (Normal Request) or SPB  SCRC04   (Priority Request)
DEL  0,SCNTBA
DEL  0,CNTTBA
     Busy Return (Stacking List Full)
     Normal Return
```

Core Addresses Only



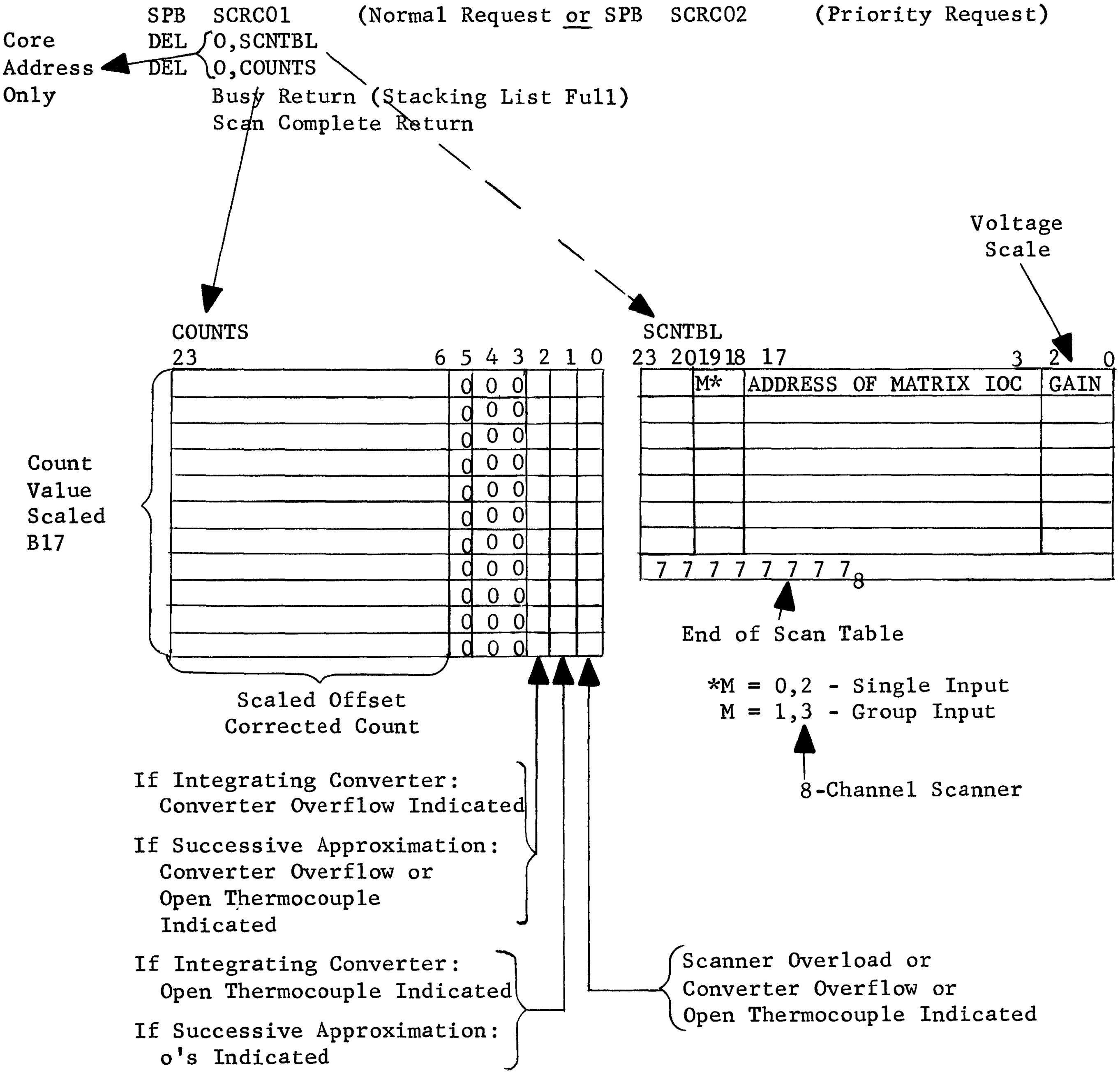
PROCESS COUNT VALUES IN TABLE B. CONVERT, LIMIT CHECK, ETC.

SCAN COMPLETE REQUEST SCRC03 or SCRC04

```
SPB  SCRC05

         Scan Incomplete
         Scan Complete
```

Request scan for "B Tables and process new counts in "A" Table

A non-buffered scan request locks out the calling program until the analog scan request is processed.



```
          SPB  SCRC01     (Normal Request or SPB  SCRC02     (Priority Request)
Core      DEL  0,SCNTBL
Address   DEL  0,COUNTS
Only           Busy Return (Stacking List Full)
               Scan Complete Return
```

Core Address Only
COUNTS
23
6 5 4 3 2 1 0
0 0 0
Count Value Scaled B17
Scaled Offset Corrected Count
SCNTBL
23 20 19 18 17 3 2 0
M*
ADDRESS OF MATRIX IOC
GAIN
Voltage Scale
7 7 7 7 7 7 7 7
8
End of Scan Table
*M = 0,2 - Single Input
M = 1,3 - Group Input
8-Channel Scanner
If Integrating Converter: Converter Overflow Indicated
If Successive Approximation: Converter Overflow or Open Thermocouple Indicated
If Integrating Converter: Open Thermocouple Indicated
If Successive Approximation: o's Indicated
Scanner Overload or Converter Overflow or Open Thermocouple Indicated

SCNTBL

23	22 — 20	19 — 3	2 — 0
G	Thermocouple Reference Junction Temperature Index		GAIN

G = 0 (No Gain Optimization)
G = 1 (Gain Optimization on Single API or Single Channel Scanner)

(Reference: Scan Command Word, Programming Techniques Manual)

8.1.25 Scan Driver (SND)

The Scan Driver outputs Scanner Commands to a Driver Table. They are sent to the Scanner by an ØDL or ØUT Command. The count values are returned by an IDL or IN Instruction and stored in the specified table. The count values are converted, offset corrected, and scaled before storing. After all requested points have been scanned, the calling function is turned on.

8.1.26 Scan Offset Program (SCF)

The analog readings of the shorted pair are obtained for each voltage scale. This new offset is calculated from the weighted average of the current reading and four previous offset values for each voltage scale.

8.1.27 Corrective Action Diagnostic (CAD)

CAD performs peripheral, drum, disc, multiple output distributor, scanner, and timed-contact output corrective actions for the GE/PAC Monitor. Corrective action is taken for the following peripherals:

100 CPM Card Punch
350 CPM Card Reader
300 LPM Printer

70 CPM Card Reader
Output Typers
I/O Typers
Paper Tape Punch
Paper Tape Reader

Corrective actions are as listed:

1. Substitute an alternate output peripheral when an output peripheral fails.
2. Permit operator recovery for the 350 CPM Card Reader and 100 CPM Card Punch.
3. Reset device flags and switches for the failed device.
4. Turn on any programs "locked out" for the failed device.
5. Type alarm messages indicating the exact location of the failure, when possible.

After the necessary actions are completed, CAD turns itself off and exits to the ECP.

8.1.28 Initialization Routine (INZ)

INZ initializes the start-up conditions for a Monitor System. Initial storage, switches, and variable locations are set for Monitor and Compatible Programs.

Each system should add its own system initialization to this routine.

To initialize an all-core system, the programmer must follow the outlined steps:

1. Branch to the starting location of the Initialize Routine (37_8).
2. Press Console → B.
3. Turn key to "Automatic".
4. Press "Step" button.
5. Reset the "API Lockout" switch.

For a drum/core system, follow these steps:

1. Turn keyswitch to the "Manual" position.
2. Set "API Lockout" switch to "Inhibit" position.
3. Turn "Off/On" switch to "Initialize" position.
4. Press "Program Load" button.
5. Press "Demand" button.
6. Turn keyswitch to "Auto" position.
7. Press "Step" button.
8. Press "Demand" button.
9. Raise all console switches.
10. Press "Demand" button.
11. Set "API Lockout" switch to "Permit" position.

The system time can be seen counting right-justified in the A-Register.

8.1.29 Console Switch On-Line Operator Program (ØPR)

This program is used to communicate with the on-line service routines. Each routine is requested by using the Console Switches on the GE/PAC Console. (Reference: GE/PAC 4000 Operator System Console Reference Manual, Library Control No. YPG29.) The following routines operate under ØPR42:

8.1.29.1 On-Line Memory Change with Punch Option (MCG)

Types and/or changes memory contents.
Records program changes by typewriter and paper tape.

8.1.29.2 On-Line Loader (LDR)

Transfers information stored on paper tape into memory.
Compares memory contents with tape information.
Relocates tape information when storing into memory.
Loads pre-assembled library subroutines.

8.1.29.3 On-Line Dump (ØLD)

Punches memory contents on paper tape.
Types memory contents on the Console Typewriter.

8.1.29.4 On-Line Clock (CLK)

Resets the system time.
Updates the program execution times and auxiliary time counters based on the new system time.

8.1.29.5 Program Status (PST)

Initiates or stops the execution of functional programs operating under MONITOR control.
Locks out any functional program preventing it from being turned on.
An option, which permits a program to be turned on at a designated time, is also available. Without this option, a program is turned on immediately.

8.1.29.6 Peripheral In/Out of Service (PØS)

Removes peripheral devices from service without an alarm notification.

Restores failed or out-of-service peripherals to service.

8.1.29.7 On-Line Paper Tape Duplicator (DUP)

Duplicates paper tapes.

8.1.29.8 Extract Load Tape Program (XLT)

Produces a bi-octal tape from the PAL output symbolic tape.

8.1.29.9 PAL GE/PAC Assembler (PAL)

Assembles symbolic programs punched on paper tape or cards. Types an assembly listing.
Produces a bi-octal paper tape.

8.1.29.10 PAL Correction Program

Updates a PAL or FORTRAN language program by reading a tape containing changes to the program. Inserting, deleting, and replacing instructions are permitted. The revised program may be typed and resequenced by tens.

8.1.29.11 Demand Scan (DSN)

Assists in calibrating analog points by demand scanning of one analog point at a time.

8.1.29.12 Controller Change (CCG)

Changes the value of an analog point on the Analog Scan Controller.

8.1.30 On-Line Operator System (I/O Typer) ØPR43

This program calls on-line service routines requested by the I/O Typewriters. The following routines may be called under ØPR43:

8.1.30.1 On-Line Memory Change with Punch Option for I/O Typer (MCG43)

Changes memory locations through the I/O Typer.
Types the contents of memory locations.
Punches a tape record of any changed location.

8.1.30.2 On-Line Paper Tape Loader - I/O Typer (LDR43)

Transfers information stored on paper tape into memory.
Compares memory contents with tape information.
Relocates tape information when storing into memory.
Loads pre-assembled library subroutines.

8.1.30.3 On-Line I/O Typer Dump (ØLD43)

Punches memory contents in the bi-octal format on the paper tape punch.

Types or prints memory contents in the octal format on the I/O Typer or Printer.

8.1.30.4 Clock (CLK43)

Resets the system time.
Updates the program execution times and auxiliary time counters based on the new system time.
Establishes the date in the Monitor program.
Types a record of the time and date change.

8.1.30.5 Program Status (PST43)

Initiates or stops the execution of functional programs under MONITOR control.
Locks out any functional program preventing it from being turned on.
Supplies a time option which permits a program to be turned on at a specified time. Without this option, a program is turned on immediately.
Types a record of all program status changes.

8.1.30.6 Peripheral In/Out of Service (PØS43)

Removes peripheral devices from service.
Restores failed or out-of-service peripherals to service.
Types a record of the service performed.

8.1.30.7 Paper Tape Duplicator (DUP43)

Duplicates paper tapes.
Types an actal checksum of the tape characters duplicated.
Issues error messages when parity or device failures are detected.

8.1.30.8 Extract Load Tape Program (XLT)

Produces a bi-octal tape from the PAL output symbolic tape.

8.1.30.9 PAL GE/PAC Assembler (PAL)

Assembles symbolic programs punched on paper tape or cards, types an assembly listing, and produces a bi-octal paper tape.

8.1.30.10 PAL Correction Program

Updates a PAL or FORTRAN language program by reading a tape containing changes to a program.
Inserting, deleting, and replacing instructions are permitted.
The revised program may be typed and re-sequenced by tens.

8.1.30.11 GE/PAC FORTRAN Compiler

Compiles GE/PAC programs written in the FORTRAN language.
FORTRAN statements may be read from paper tape or cards.
From the input, a paper tape is created for listing on the flexowriter or assembly by the PAL Assembler.

8.1.31 Find/Restore Working Core Area Subroutines (FMR)

The Find Working Core Area Subroutine is used to find space for reading data from paper tape, scanning analog points, transferring a program segment, building an output data table, transferring an untested program of the Free-Time System, etc. When a space is found, this area is set to unavailable, occupied status.

The Restore Working Core Area Subroutine releases an area by setting it unoccupied an available.

To find a working core area,

```
LDA  Number of Locations
SPB  FMRC01
     Unavailable Return
     Normal Return with the location of the free
     core area in the A-Register
```

The "Unavailable Return" is taken when a working core area is not available. The system programmer, at this time, should set a delay and then initiate another FMRC01 subroutine call.

To release a working core area,

```
DLD  A-Register with the Number of Locations
     Q-Register with the starting core location
SPB  RMRC01
     Normal Return
```

8.1.32 Run, Stop System Subroutine (RMP)

RMP requests transfers for system subroutines from drum or disc into a working core area. After transfer, RMP branches to the first location of the subroutine. When the subroutine has completed its functions, RMP releases the working core areas and returns to the calling program.

To request a system subroutine,

```
Set program area unavailable.  (See MAP)
DLD  A-Register with the drum location
     Q-Register with the number of words
SPB  RMPC01
     Busy Return
     Normal Return
```

8.1.32 Run, Stop System Subroutine (RMP) - cont'd.

To release its working core area, the subroutine must know its own size and its present core location.

```
DLD  A-Register with No. of Words
     Q-Register with Core Location
LDX  Return,2
SPB  SMPC01
```

If the requested program is busy with another request, return is made at the busy return. If either FMRC01 or DTRC02 is busy, a quarter second delay is initiated in the calling program, and the request is re-initiated after the delay.

APPENDIX A

MONITOR PSEUDO-OPS

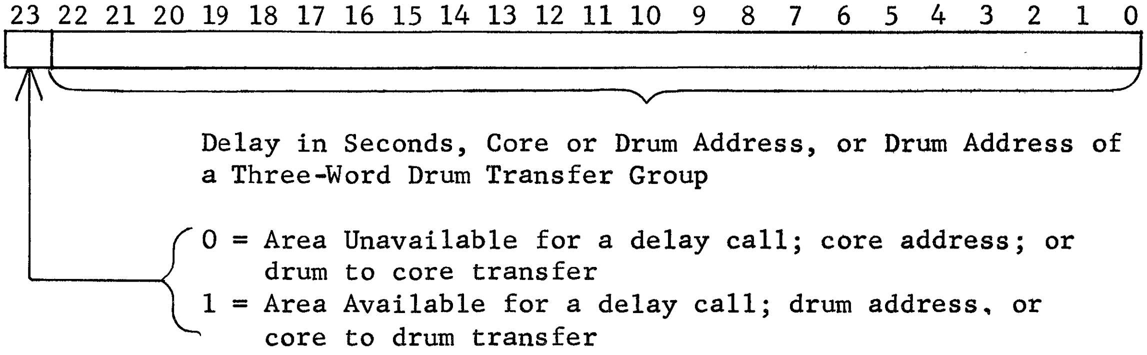
Pseudo-ops defined by MONITOR are:

BCD - Binary to 6-Bit BCD
BCN - Binary to Non-Edited Character
CLK - Clock
DEL - Delay and Drum/Core Addresses
DFE - Floating Point to E Type Floating Point Decimal
DFP - Floating Point to Fixed-Point Decimal
DFX - Binary to Fixed-Point Decimal
FBB - Binary to 4-Bit BCD
FØR - Peripheral Device No. and Message Area No. for Output Calls.
ØCT - Binary to Octal (integers)
PRG - Maintains REGSTG and RSX Tables of ECP
SIZ - Size of Program

The BCD, BCN, CLK, DFE, DEP, DFX, FBB, and ØCT pseudo-ops are used with the Output Program as format words. Refer to the Output Program, subcategories of 1.14, for explanation of these pseudo-ops. The DEL, FØR, PRG, and SIZ pseudo-ops are used in subroutine calls and are explained below:

The DEL pseudo-op is used in the Scan Request, Input Request, Output Request, 1st word of the 3-word Drum Transfer Request, and Set Program Delay Subroutine Calls. Coding examples and format are as follows:

```
DEL    0,  15*SECND          DEL    1,FMTB
```

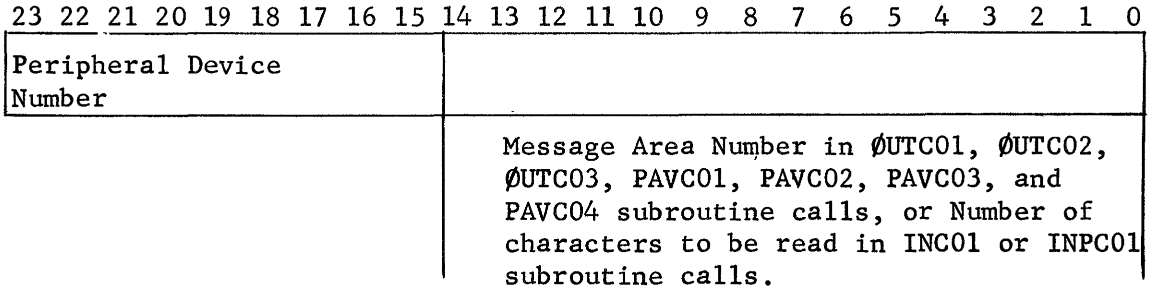


Delay in Seconds, Core or Drum Address, or Drum Address of a Three-Word Drum Transfer Group

0 = Area Unavailable for a delay call; core address; or drum to core transfer
1 = Area Available for a delay call; drum address, or core to drum transfer

The FØR pseudo-op is used to specify the peripheral device number and message area number for output calls, or peripheral device number and number of characters to be read for input calls.

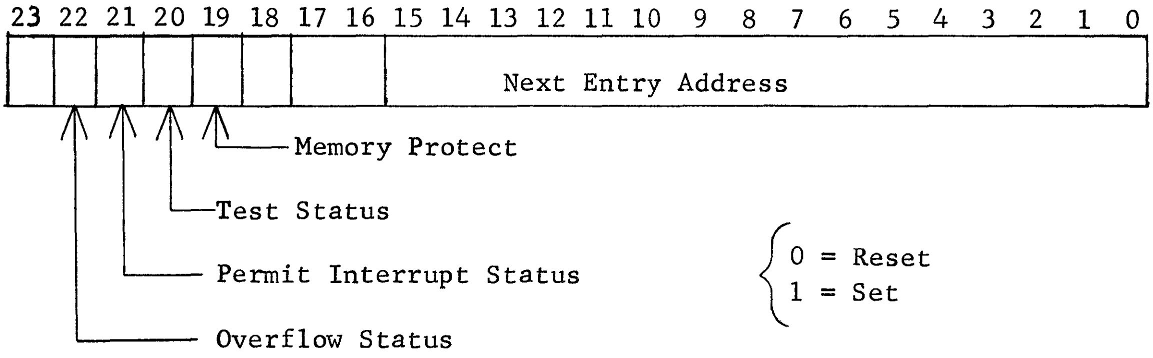
```
FØR LTYPER,  0          FØR CRRDR,  22
```



Message Area Number in ØUTC01, ØUTC02, ØUTC03, PAVC01, PAVC02, PAVC03, and PAVC04 subroutine calls, or Number of characters to be read in INC01 or INPC01 subroutine calls.

The PRG pseudo-op is used to maintain the REGSTG and RSX Tables of ECP and by the Turn Program Off Subroutine. It contains the flip-flop status of overflow, permit interrupt, test, next entry location, and memory fence.

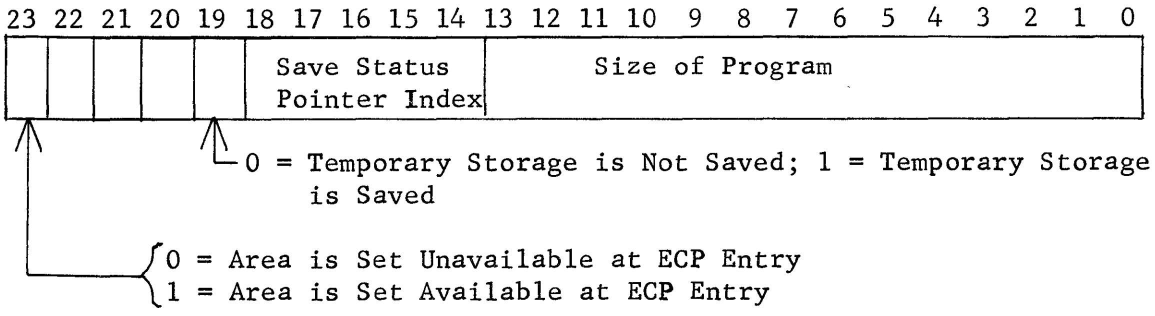
```
PRG  0, 1, 0, START
PRG  0, 1, 0, START,0
```



For a Drum/Core system, if the next entry address is set to zero, the program begins at its first location.

The SIZ pseudo-op specifies the required parameters for the second word of the three-word ECP Drum Transfer Group.

SIZ 1, 1, 3, /400 (This pseudo-op tells the area availability status, Bit 23; save status, Bit 19; save status pointer index. Bits 18-14; and Size of the Program.)


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Save Status Pointer Index
Size of Program
0 = Temporary Storage is Not Saved; 1 = Temporary Storage is Saved
0 = Area is Set Unavailable at ECP Entry
1 = Area is Set Available at ECP Entry

APPENDIX B

COMMUNICATION CALLS

		Page
DELC01	Set Program Delay for A Specified Time Period	1-9
DELC02	Sets Program Area Unoccupied and Available During Delay	1-9
DTRC01	Drum Transfer Request (Immediate Return)	1-12
DTRC02	Drum Transfer Request (Drum Transfer Completed Return)	1-12
FMRC01	Find A Working Core Area	1-46
FRPC01	Find Register Pointer	1-13
INC01	Non-Buffered Input Request	1-31
INPC01	Buffered Input Request	1-29
INPC02	Read Completed Request	1-30
MAP01	Core Map Maintenance - Set Area Occupied	1-11
MAP02	Core Map Maintenance - Set Area Unoccupied	1-11
MAP03	Core Map Maintenance - Set Area Unavailable	1-11
MAP04	Core Map Maintenance - Set Area Available	1-11
MØRC01	Normal Non-Priority Multiple Output Request	1-33
MØRC02	Normal Priority Multiple Output Request	1-33
MØRC03	Timed Non-Priority Multiple Output Request	1-34
MØRC04	Timed Priority Multiple Output Request	1-34
MØRC05	Unlatch A Previous Timed Request	1-34
MØRC06	Pulsed Non-Priority Multiple Output Request	1-35
MØRC07	Pulsed Priority Multiple Output Request	1-35
ØFFC01	Turn Program Off Request	1-8
ØFFC02	Set Program Area Unoccupied and Available During the Time Program Is Turned Off	1-8
ØUTC01	Peripheral Availability and Data Area Output Request	1-16
ØUTC02	Normal Output Request	1-16
ØUTC03	Priority Output Request	1-17
PAVC01	Peripheral Availability Request with Alternate Selection	1-14
PAVC02	Data Area Availability Request	1-15
PAVC03	Output Request which Places Request in Stacking Table	1-15
PAVC04	Priority Request (Places Request at Top of the List)	1-14
RMPC01	Request A System Subroutine	1-46
RMRC01	Release A Working Core Area	1-46
SCRC01	Normal Non-Buffered Scan Request	1-40
SCRC02	Priority Non-Buffered Scan Request	1-40
SCRC03	Normal Buffered Scan Request	1-39
SCRC04	Priority Buffered Scan Request	1-39
SCRC05	Scan Completed Request	1-39
SMPC01	Release Working Core Area of A System Subroutine	1-47
TCØC01	Normal Timed Contact Output Request	1-37
TCØC02	Priority Timed Contact Output Request	1-37
TPNC01	Turn Program On Request	1-10
TPNC02	Turn Program On Request If Program Is Currently Turned Off	1-10
TPNC03	Specifies Next Running Program	1-10

APPENDIX C

MONITOR SYMBOLS

The following symbols are defined by Monitor, and should not be defined by the programmer:

ALERT - Peripheral Activate Flag
The peripheral device priority number corresponds to the associated bit.
Example: Device #3 = Bit 3 of Flag

ALTFLG - No Alternate Device Flag
This flag shows that an output message for the requested output peripheral has no working alternate, therefore, the output message was destroyed.
Example: Device priority number i corresponds to bit i.

ALTTBL - Primary Alternate Device Priority Numbers for Peripheral Devices
The alternate for device priority number i is found in ALTTBL+i.
Example: Priority Device #5 = ALTTBL+5

ALTTB2 - Secondary Alternate Peripheral Substitution Table. Provides another alternate table in the event a secondary alternate peripheral device is required.

AUXTM - Table of Auxiliary Time Counters

AVLMAP - Available Area Map

BAD - Peripheral Device Failure Flag
The peripheral device priority number corresponds to the associated bit.
Example: Device #4 = Bit 4 of Flag

CØRMAP - Occupied Area Map

DMCRNØ - Three Times the Running Program Number

DTAREA - Data Area Bit Word
Area$_i$ corresponds to Bit$_{i-1}$
Example: Area 2 = Bit 1
Area 3 = Bit 2

DVCØDE - Device Codes
A table of hardware addresses and classification codes for peripheral devices arranged in descending order.
Example: #0 = Highest Priority Device

FAILUR - Peripheral or Hardware Failure Device Flags

ØØS - Peripheral Device Out-of-Service Flag
The peripheral device number corresponds to the associated bit.

PRIØNØ - One Times the Running Program Number

PRØCFG - Output Peripheral Device-in-Progress Flag
The peripheral device number corresponds to the associated bit.

PRØG - Program Execution Times
The table starts with Program No. 1.

PRØGNØ - Eight Times the Running Program Number

SECND - One second in real-time counts: 1 = One second system; 2 = One-half second system; 4 = One-quarter second system, etc.

SELECT - Peripheral Action Flag
The peripheral device number corresponds to the associated bit.

APPENDIX C

MONITOR SYMBOLS

STSMAP - Occupied Save Status Area Map

TIME - Time in System Counts (Cleared at Midnight)

WAITFG - Output Peripheral Message Completion Flag
The peripheral device number corresponds to the associated bit.

WAITRQ - Number of Words Reserved in the Stacking Table for Output Peripheral Message (each output request requires two words in the table).
This table is indexed by device priority number.

ZFER - Number of Drum Transfer Requests Waiting for a Program
This table is referenced by program number.
Example: Program 7 = XFER+7

APPENDIX D

MONITOR ASSEMBLY CHECKLIST

In building a Monitor system, the programmer must define the specific parameters and system options. The following checklists are used to specify these parameters. When the checklists are completed, send them accompanied by the interrupt assignment for the system to the Programming Librarian.

By judicious selection of system options, it is possible to acquire a Monitor which contains only those capabilities that are needed for each application. The size of Monitor varies with the combination of options selected.

Additional copies of the Checklist may be obtained upon request from the Programming Library.

Library Control No. ____________
Programmer's Name ____________
Date ____________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
1. IS YOUR SYSTEM FOR (check One) 1. AU1 (GE/PAC 4040) Drum/Disc Core 2. AU1 (GE/PAC 4040) All-Core 3. AU2 (GE/PAC 4050/4060) Drum/Disc Core 4. AU2 (GE/PAC 4050/4060) All-Core			
2. QUASIS - Check one or more depending on the system's need. Single-word floating point (AU2) □ Double-word floating point (AU2) □ Single word with Multiply Step (AU1) □ Single word without Multiply Step (AU1 or AU2) □ Double word floating point with Multiply Step (AU1) □ Is floating point being used in your system? YES____ NO____			
3. NUMBER OF COUNTS USED BY SYSTEM PROGRAMS Number of System Program to be turned on when a system count becomes negative. This program must be written by the System Programmer Option count. System calendar updating YES____ NO____	AA APRØGM	EQL EQL	
4. TOTAL NUMBER OF FUNCTIONAL PROGRAMS	B	EQL	
5. NUMBER OF FUNCTIONAL PROGRAMS HAVING SAVE STATUS. (Include Monitor Programs listed in 11.)	BB	EQL	
6. NUMBER OF 8-WORD REGISTER STORAGE BLOCKS (Include Monitor Programs listed in 11.)	BBB	EQL	

Library Control No. ___________
Programmer's Name ___________
Date ___________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
7. STARTING ADDRESS OF THE FLOATING WORKING CORE AREA (Starts after MONITOR and system permanent core and must be a multiple of 100_8.)	BASE	EQL	/
8. MAXIMUM CORE ADDRESS	MAXC	EQL	/
MAXIMUM DRUM/DISC ADDRESS	MAXD	EQL	/
9. NUMBER OF ON-LINE PERIPHERAL DEVICES Peripheral devices are typewriters, printers, card readers, card punches, paper tape punches, magnetic tapes, etc.	E	EQL	
10. STACKING TABLES, 1 per peripheral device			
One stacking table is required for each peripheral device (printer,	F_0	EQL	
magnetic tape, typewriter, or punch). The size of each stacking	F_1	EQL	
table must be a power of two (F_i). The number of requests in each	F_2	EQL	
stacking table represents the number of concurrent outputs that	F_3	EQL	
can be requested on a peripheral. Each request requires two entries.	F_4	EQL	
The actual size of the stacking table is equal to F_i. For four	F_5	EQL	
concurrent requests on device 2, the stacking table must have eight	F_6	EQL	
locations. The following peripherals (input) require no stacking	F_7	EQL	
tables: (Assign zero for these peripherals).	F_8	EQL	
	F_9	EQL	
70 CPM Card Readers	F_{10}	EQL	
Paper Tape Readers	F_{11}	EQL	
350 CPM Card Readers	F_{12}	EQL	
Recommended sizes for the other stacking tables are:	F_{13}	EQL	
Paper Tape or Card Punches - 4	F_{14}	EQL	
Printers 8			
I/O Typers 8	F_{15}	EQL	
Output Typers 8 or 16			
Magnetic Tapes 4			

Library Control No. ________
Programmer's Name ________
Date ________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
11. PROGRAM NUMBERS			
Output Program	G	EQL	
I/O Driver	IØDPRG	EQL	
Scan Offset Program	SCF	EQL	
Input Program	KNMBR	EQL	
Corrective Action Program	L	EQL	
Console Switch Operator Program	S	EQL	
I/O Typer Operator Program	SS	EQL	
12. SIZE OF SYSTEM INITIALIZATION Do not include Monitor initialization area.	SINZTL	EQL	
13. MESSAGE TYPE OR STOP OPTION FOR DRUM/DISC FAILURE Device Address of typer for Drum/Disc Alarm	ITCTYP	EQL	/
14. NUMBER OF 50 OR 60-CYCLE PULSES IN ONE TIME COUNT INTERVAL This EQL defines the length of the time count interval for the system. The system time may be kept in seconds, multiples of seconds, or fractions of seconds. For example, in a 60-cycle system, a time count interval of 1/4 seconds, NCYCLE would be fifteen. NCYCLE would be sixty for a one-second interval. Check 50 or 60-Cycle Pulse Required □ 50-CYCLE □ 60-CYCLE	NCYCLE	EQL	

Library Control No. ________
Programmer's Name ________
Date ________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
15. OUTPUT PROGRAM INFORMATION Number of Lines Per Page of Printer Output (Specify lines for each printer.)	LPAGE	EQL	
16. ECP OPTIONS REGISTER POINTER TABLE YES ___ Check this option if there are NO ___ Programs within your system which have single word register storage or 8-word shared register storage.	RSX		
PRIORITY TABLE OPTION YES ___ If program priority changing is NO ___ desired, this table must be included in your Monitor DOES YOUR SYSTEM HAVE FIXED WORKING CORE AREAS? YES ___ NO ___	PRGTBL		
17. TURN PROGRAM ON S.R. CALL YES ___ This optional call, TPNC03, NO ___ specifies which program to run next.			
18. UNOCCUPIES, AVAILABLE OPTION FOR SET PROGRAM DELAY AND TURN PROGRAM OFF SUBROUTINES DELC02 Call YES ___ NO ___ ØFFC02 Call YES ___ NO ___			

Library Control No. ________
Programmer's Name ________
Date ________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE

19. DEVICE CODES - Priority Order #0-#15

Priority levels must be established for each peripheral device. Devices which must be activated as soon as possible upon request are placed in the highest priority.

The 16-character device name is used for alarming actions.

If the FORTRAN Submonitor is desired, fill in its device number. (System Option #20) FORTRAN peripherals are divided into three categories-Input, Printing, and Punching Peripherals. Each category begins with Device No. 0.

Mtr. Priority No.	16-Char. Device Name for CAD	Device Address	Alt. #1	Alt #2	FORTRAN SUBMTR NO.
#0					
#1					
#2					
#3					
#4					
#5					
#6					
#7					
#8					
#9					
#10					
#11					
#12					
#13					
#14					
#15					

20. IS THE PAGE CONTROL OPTION DESIRED FOR TYPEWRITERS? YES___ NO ___

The lines per page is usually 58; lines to top of next page is usually 8.

Device Prior.#	Lines Per Page	Lines to Top of Next Page

21. IS THE ASCII CODE TO BE USED IN YOUR SYSTEM? YES ___ NO ___	ASWØRD		

22. CORRECTIVE ACTION DIAGNOSTIC PROGRAM

Priority Number of Device for Operator Messages

Check if corrective action messages should be printed in red. YES ___ NO ___

Check is special Multiple Output corrective action. YES ___ NO ___

Note: If special action is required, define your system needs.

DTYPER	EQL	

Library Control No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
1. INPUT Give the program number of the program which will be turned on if the Input Demand Button is pressed for the following peripheral devices:				PROGRAM NUMBER
I/O TYPER #1				
I/O TYPER #2				
Paper Tape Reader #1				
Paper Tape Reader #2				
70 CPM Reader #1				
70 CPM Reader #2				
400 CPM Reader #1				
400 CPM Reader #2				
If no program is associated with the Input Demand Button, place N/A for that device.				
2. ON-LINE OPERATOR SYSTEM (Console Switches) Priority Number of Console Typewriter Priority Number of Console Punch Priority Number of Console Reader This option is used for on-line debugging. The programs listed under the OPR System may be called during system operation.		CTYPER CPUNCH RDR	EQL EQL EQL	
*A. Memory Change - displays or changes the contents of memory.				
*B. On-Line Loader - enters new programs or data from paper tape.				
*C. On-Line Dump - displays large blocks of memory				

* Requires the Octal Output Routine.

Library Control No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
2. ON-LINE OPERATOR SYSTEM (Console Switches) - cont'd. D. On-Line Clock - updates the system time or initiates the system time according to the time of day. Option - Date YES ___ NO ___				
E. Program Status - On/Off/Lockout - turns on, turns off, or "locks out" system functional programs. Option - Turn on Program at a specified time. YES ___ NO ___				
F. Device In/Out of Service - removes peripheral devices from service when making repairs or changing paper or ribbon and restores them to service upon completion of repair.				
G. Paper Tape Duplicator - duplicates a paper tape.				
H. Extract Load Tape Program - produces a bi-octal paper tape from a PAL output symbol tape.				
I. GE/PAC FORTRAN Compiler - Compiles GE/PAC programs written in the GE/PAC FORTRAN language.				
J. Demand Scan - scans one analog point at a time.				
K. Controller Change - changes the value of an analog point.				

Library Control No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
3. ON-LINE OPERATOR SYSTEM (I/O Typer)				
Priority Number of Console Punch		CPUNCH	EQL	
Priority Number of Console Reader		RDR	EQL	
Device Priority Numbers of I/O Typers which may communicate with ØPR.			EQL	
The following programs may be called under the I/O Typer ØPR.				
A. On-Line Memory Change - displays or changes memory.				
B. On-Line Paper Tape Loader - enters new programs or data from paper tape.				
C. On-Line I/O Typer Dump - types and punches memory contents.				
D. On-Line Clock - resets the system time and updates the program execution times and auxiliary time counters. Option - Date YES ___ NO ___				
E. Program Status - On/Off/Lockout - initiates, stops or locks out the execution of functional programs under MONITOR control. Option - Turn on a program at a specified time. YES ___ NO ___				
F. Peripheral In/Out of Service - removes peripheral devices from services or restores failed or out-of-service peripherals to service.				
G. Paper Tape Duplicator - duplicates paper tapes issuing error messages when parity or device failures are detected.				
H. PAL GE/PAC Assembler - assembles symbolic program(s) punched on paper tape or cards.				
Symbolic Table Size		TBLSZE	EQL	
Indicate line spaces, if different than 112		PRMTR	EQL	
I. PAL Correction Program - Updates a PAL or FORTRAN Language Program.				
J. GE/PAC FORTRAN Compiler - Compiles GE/PAC Programs written in the GE/PAC FORTRAN language.				

Library Control No. ________
Programmer's Name ________
Date ________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
4. AUXILIARY TIME COUNTERS (Number of Auxiliary Time Counters) This option must be chosen if there are functional programs in the system which initiate themselves at regular time intervals and are also initiated by system or operator demands. The regular time interval should be saved in an auxiliary time counter.		C	EQL	
5. FLOATING POINT CALCULATIONS AND OUTPUT (Requires BDC41) Select this option for making calculations in floating-point and converting to decimal fixed point or to code in FORTRAN.				
6. BINARY E TYPE FLOATING POINT (Requires BDC41) Select this option for converting binary floating point data to decimal floating point.				
7. DECIMAL FIXED-POINT OUTPUT (Requires BDC41) This option converts binary data to fixed-point decimal format for output. CAD requires this routine.				
8. OCTAL OUTPUT This option must be used to output binary data in octal format. The Memory Change, On-Line Dump and On-Line Loader Routines require this option.				
9. ALPHA-NUMERIC CHARACTER OUTPUT (6-Bit BCD) If alphanumeric messages are required using 6-Bit BCD data (four characters per word), this option must be checked. The On-Line OPR and CAD require this option.				

Library Control No. ________
Programmer's Name ________
Date ________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
10. NON EDITED CHARACTER OUTPUT (One Character Per Word) This option is used for non-standard character formats. It is required by the OPR Systems, Memory Change, and On-Line Dump.				
11. TIME OUTPUT Select this option to convert system time counts to hours and minutes or hours, minutes and seconds in decimal. CAD requires this option.				
12. 4-BIT BCD CONVERSION This option converts binary data to four-bit BCD information.				
13. MULTIPLE OUTPUT DISTRIBUTOR Select this option if your system has a Multiple Output Distributor. Each multiple output request requires two entries in the appropriate driver table				
Number of Multiple Output Groups per Controller		MXGRP0 MXGRP1 MXGRP2 MXGRP3	EQL EQL EQL EQL	
Total Number of MØD Controllers in System		MTØTAL	EQL	
Device Address for Controller #1		MØD0	EQL	
Device Address for Controller #2		MØD1	EQL	
Device Address for Controller #3		MØD2	EQL	
Device Address for Controller #4		MØD3	EQL	
Size of Normal Multiple Output Driver Table		MØDNMB	EQL	
Size of Priority Multiple Output Driver Table		MØTNMB	EQL	
Option-Time Latching YES ___ NO ___ This option requires 2 extra API's - 60-Cycle DMT and 60-Cycle DMT Echo.				
Number of Latched Output Timers (Maximum of 24)		$TMNBP_i$	EQL	

Library Control No. __________
Programmer's Name __________
Date __________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
13. MULTIPLE OUTPUT DISTRIBUTOR (cont'd)				
Location of 60-Cycle DMT for MØ Timing		TMLØC$_i$	EQL	
Option-Pulsed Motor YES ___ NO ___ This Option requires 3 extra API's 60-Cycle DMT, 60-Cycle DMT Echo, and a non-inhibitable ØFS.				
Location of 60-Cycle DMT for MØ Pulsing		PULØC$_i$	EQL	
14. TIMED CONTACT OUTPUT CONTROLLER				
Select this option if your system has a Timed Contact Output Controller. Each timed contact output request requires two entries in the appropriate driver table.				
TCØ Controller Address		TCØ	EQL	
Number of Timed Contact Output Groups		MXTCGP	EQL	
Size of Normal Timed Contact Output Driver Table		TCØNMB	EQL	
Size of Priority Timed Contact Output Driver Table (Recommended Size - 8 or 16)		TCINMB	EQL	
15. ANALOG SCAN				
Scan Controller Address		SCAN	EQL	
Largest Group Mode Used (Number of Channels)		SCNGRP	EQL	
Scan Command Words for Offset Points		SFSCW0	EQL	/
(The Scan Offset Program is required		SFSCW1	EQL	/
Include this program on your ECP Information Sheet.)		SFSCW2	EQL	/
Check Type of Amplifier Preston High _____ Preston Low _____ Vidar Integrating Converter _____				

Library Control No. ________
Programmer's Name ________
Date ________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
15. ANALOG SCAN (cont'd) Options: Add the Reference Block Temperature for thermocouples. YES ___ NO ___ Store time of day as the last word of the Count Table YES ___ NO ___ Gain Optimization (Used only for a single channel scanner) YES ___ NO ___		SFSCW3 SFSCW4 SFSCW5 SFSCW6 SFSCW7	EQL EQL EQL EQL EQL	/ / / / /
Check if buffered scanning is desired. Buffered requests permit a functional program to process one set of count values while the Driver is scanning another set.				
Is interruption of the current scan request for a priority scan request required? (AU1 single API) YES ___ NO ___				
16. OFF-LINE DRUM/DISC CORE MEMORY CHANGE This option displays or changes memory off-line				
17. CHARACTER SET (Check One) 1. 4201A - Peripheral Buffer				
2. 4201B - Peripheral Buffer				
3. ASCII				
18. FIND/RESTORE WORKING CORE AREA This option searches for a working core area and sets it unavailable or restores an area by setting it available.				

Library Control No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
19. RUN, STOP SYSTEM SUBROUTINE (Requires Option 18) Check this option for finding space for a system subroutine stored on drum or disc and runs it in working core.				
20. FORTRAN SUBMONITOR Check this option to communicate to MONITOR through FORTRAN. FORTRAN Submonitor also requires option 18 and 19. Check the desired options: Input □ Decimal to Binary □ Card Input □ Output □ I/O Availability □ Drum Read/Write □ Subroutine Linkage □ Computed GO TO □ Assigned GO TO □ Data Link □ If the free-time system is required, all options with the exception of data link, must be included.				
21. INTERRUPTABLE SYSTEM SUBROUTINES If there are subroutines within the system which must run with interrupt permitted (due to length) and are used by more than one functional program, they must follow special entry and exit conventions. When this is the case, this option must be selected.				

Library Control No. ____________
Programmer's Name ____________
Date ____________________________

		INITIAL F/F STATUS												
PROG. PRIORITY NO.	NAME	OVRF	PAIF	TSTF	TMFF	DRUM/DISC ADDRESS	*(a) AREA STATUS	*(b) SAVE STATUS	# WORDS OF PROG.	SAVE STATUS AREA ADDRESS	# WORDS OF SAVE STATUS AREA	*(c) REG. STORAGE	*(d) ITC & DTD INTERRUPT RETURN	CHECK IF PROGRAMS RESIDE IN PERM. CORE***
1														
2														
3														
4														
5														
6														
7														
8														

*Place a checkmark (✓) in the corresponding column marked with an asterisk if:

a) Area status is available
b) Save status is required
c) 8-word register storage is required
d) Interrupt Time Counter and Drum/Disc Transfer Complete Interrupts return to ECP.
** Applies to Drum/Disc Systems. Insert starting core address in the drum/disc address column for these programs.

APPENDIX E

AUDIT CODES

Following is a list of Monitor Audit Codes and the Psuedo-Ops they define. For a more detailed discussion of Psuedo-Op subroutines, refer to the appropriate Monitor User's Manual section shown opposite each mnemonic. See page 27 (Appendix C) of Process Assembler Language Manual for additional information relative to Audit Codes.

Section/Page Reference	Psuedo-Op	
1.14.3	BCD	*DEF /40000000, 30, 28, 29, 31
1.14.4	BCN	*DEF /50000000, 38
1.14.6	CLK	*DEF /60000000, 63, 25, 26, 28, 29, 31
A-1	DEL	*DEF /00000000, 16, 9, 0, 0
1.14.7	DFE	*DEF /10000000, 63, 38, 37, 28, 34, 33, 32, 21
1.14.8	DFP	*DEF /00000000, 63, 38, 37, 36, 35, 34, 33, 32, 31
1.14.1	DFX	*DEF /20000000, 63, 38, 37, 28, 34, 33, 32, 31
1.14.2	FBB	*DEF /70000000, 63, 38, 37, 34, 32, 31
A-1	FØR	*DEF /00000000, 27, 3, 0, 0
1.14.5	ØCT	*DEF /30000000, 38, 37, 34, 31
A-1	PRG	*DEF /00000000, 63, 13, 14, 15, 1, 39
A-2	SIZ	*DEF /00000000, 16, 39, 40, 3

Assigned Monitor Audit Codes

25	*DEF /1, 18
26	*DEF /1, 15
27	*DEF /777, 15
28	*DEF /37, 7
29	*DEF /77, 1
30	*DEF /777, 12
31	*DEF /1, 0
32	*DEF /1, 1
33	*DEF /1, 2
34	*DEF /17, 3
35	*DEF /17, 7
36	*DEF /1, 11
37	*DEF /7, 12
38	*DEF /77, 15
39	*DEF /1, 19
40	*DEF /37, 14

GE PAC 4000

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER

TASC INSTRUCTION REFERENCE MANUAL

GENERAL ELECTRIC COMPANY
PHOENIX, ARIZONA

Library Control No. YPF18M

This manual published by:
PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER SECTION
P. O. BOX 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

IN THE DEVELOPMENT OF THE SOFTWARE DESCRIBED, THE GENERAL ELECTRIC COMPANY RESERVES THE RIGHT TO MODIFY THE DESIGN FOR REASONS OF IMPROVED PERFORMANCE AND OPERATIONAL FLEXIBILITY.

CONTENTS

INTRODUCTION

GE/PAC TASC (Tabular Sequence Control) is a two-pass assembler program which provides efficient symbolic programming capabilities for coding sequence routines. GE/PAC TASC is advantageous to any assignment which requires a sequential progression of steps to control a continuous process. These steps may be in fixed sequence or alternate predetermined sequences depending upon the process at specified times and/or conditions. It is unlimited in the number of steps permitted, subject only to the size and flexibility of the computer being used and relationship of other functions requiring memory.

GE/PAC TASC operates from a table which provides the variables which are distinctive to a particular plant. Initial coding and final documentation of the sequences are in an understandable language. Minimum training is required for personnel to effectively use the language, either for initial coding or updating operating programs. When completed, the final documentation is expressed in terms easily understood by plant personnel without computer programming background.

GENERAL

GE/PAC TASC (TAS03) compiles TASC language programs and saves the data on an Output File Tape. It also updates programs on an Input File Tape as specified by the control and correction cards. Changes to existing programs are accomplished on a match/merge basis.

The following processing operations may be performed by TAS03 using the control cards described herein:

1. Assemble one or more new programs, creating a new Output file tape or adding to a current tape
2. Modify existing programs on an Input file tape
3. Delete existing programs from an Input file tape
4. Prepare a common EQL tape from several assemblies
5. Prepare a point summary tape from input cards

Programs written in the TASC language may be interspersed with PAL, COOL, FORTRAN, etc. language programs on an Output file tape. Interspersing is possible because each program has its own project-program number. TASC will copy TSR programs; TSR will copy TASC.

1 STATEMENT FORMAT

TASC language cards include both TASC statements and PAL commands. Input information is written on the "TASC Coding Form" (Figure 1). Each line on the coding form represents one instruction to the assembler. All GE/PAC instructions are valid except those concerned with input/output. The only valid PAL pseudo operations are BRT, CON, DCN, DEF, and EQL. The format and restrictions or conventions for PAL records are contained in the "Process Assembler Language" manual. The coding form is composed of fourteen fields defined as follows:

1.1 LOCATION FIELD - Columns 1 thru 6

This field is used to identify the instruction location. A name written in this field becomes associated with the instruction written on the same line. Any reference to the instruction may be made by that name. Names used in this field must consist of six of fewer characters, the first of which must start in column one. A decimal is considered as an alphabetic character in this context.

COLUMN 7 IS NOT USED

1.2 OP CODE FIELD - Columns 8 thru 10

This field contains a two or three character operation code which identifies the operation to be executed. The legal operations are defined in Section 3 (page 9 through 54). Refer to the "Process Assembler Language" manual (YPG12M) for straight-line coding techniques.

1.3 POINT IDENT FIELD - Columns 11 thru 16

This field contains a six-character alphanumeric symbol which is the name of the point in the point summary table.

1.4 STATUS FIELD - Column 17

A single character in this field designates the desired status of the input:

H = High	Ø = Open
L = Low	C = Closed

1.5 VALUE FIELD - Columns 18 thru 23

This field is used to enter the numeric value required by the specific TASC op-code. Formats are:

XXXXXX, XXXX.X, XXX.XX, XX.XXX, X.XXXX, and .XXXXX

The number is right justified; leading zeros not required.

1.6 TRY FIELD - Columns 24 and 25

This field designates the number of times to ask a question or execute a designed path of coding. Right justified; leading zeros not required.

1.7 DELAY FIELD - Columns 26 thru 29

A value (minutes, seconds, or fractional seconds) in this field designates the length of time for a program delay. Formats are: XXXM, XXXS, X.XS, or X.XX. The last format is assumed seconds. The decimal point must appear in column 27 when used. Right justified; leading zeros not required.

1.8 A FIELD - Columns 30 and 31

This field contains the previous breakpoint number.

1.9 B FIELD - Columns 32 and 33
This field contains the current breakpoint number.

1.10 TYPE FIELD - Column 34

An alphabetic character appearing in this field defines the type of alarm specified by a system. Acceptable codes are A or P which correspond to 0 and 15 in output.

1.11 ABNORMAL RETURN FIELD - Columns 35 thru 40

This field contains the symbolic location of the path for the analyzing subroutine when the condition of input is not as specified. This field is also known as the Branch Field.

1.12 COMMENTS FIELD - Columns 41 thru 69

This field is used at the descretion of the programmer to make remarks associated with each instruction.

1.13 KEY FIELD - Column 70

A character in this field designates the language or a change for the line of coding on which it appears. The two characters used when coding in TASC are:
8 = TASC language
0 = Delete

1.14 IDENTIFICATION FIELD - Columns 71 thru 80

The project-program number is written in columns 71 thru 75. Sequence numbers appear in columns 76 thru 80.

GENERAL ELECTRIC
PROCESS COMPUTER SECTION
PHOENIX, ARIZONA

TASC
TABULAR SEQUENCE CONTROL CODING FORM

Project Name

Program Name

Page of Date

Programmer

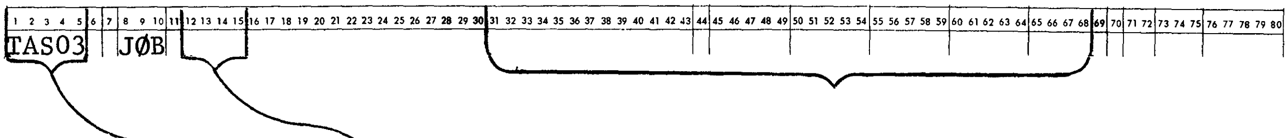
	SYMBOLIC LOCATION		OP CODE	POINT IDENT	S	VALUE	TRY	DELAY	A	B	TYPE	ABNORMAL RETURN	COMMENTS	KEY	Proj. #	Prog. #	SEQ. #	
	1 2 3 4 5 6	7	8 9 10	11 12 13 14 15 16	17	18 19 20 21 22 23	24 25	26 27 28 29	30 31	32 33	34	35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69	70	71 72	73 74 75	76 77 78 79 80	
0														8				0
1														8				1
2														8				2
3														8				3
4														8				4
5														8				5
6														8				6
7														8				7
0														8				0
1														8				1
2														8				2
3														8				3
4														8				4
5														8				5
6														8				6
7														8				7
0														8				0
1														8				1
2														8				2
3														8				3
4														8				4
5														8				5
6														8				6
7														8				7

FIGURE 1

2 CARD FORMATS & DESCRIPTIONS

The card formats used with GE/PAC TASC are described in the following paragraphs. These card descriptions illustrate the information required on each. Information which remains constant is printed on the coding line; other requirements are bracketed.

2.1 The JOB control card is the first card in every input deck processed. Columns 31-68 are printed as the heading on the printer output unless/until an HDG control card is encountered.

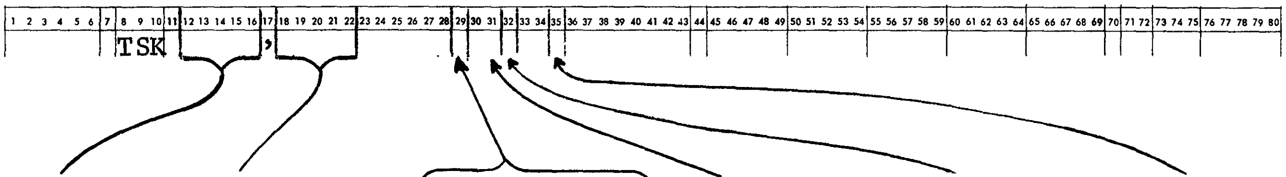


ID for Operator

Input File Tape # for corrections or additions

Date, Programmer Name, Project Name, etc.

2.2 The TSK control card precedes each program of TASC language cards.



Proj-Prog # of input cards and/ or tape

Resequence and start with this number if col. 29 is S. Change Proj-Prog to this number and resequence starting at 00010 if col. 29 is L. If blank, resequence and start with 00010 if col. 29 is S. (Ignored if col. 29 is blank.)

S=Resequence
L=Change Proj-Prog #
Blank = Do not resequence or change Proj-Prog #

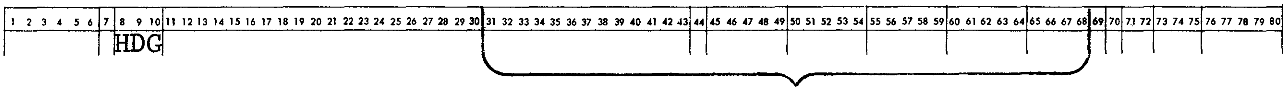
Blank=Clear EQL tables. Character = Save previous EQL tables.

Blank= No EQL tape input. Character = Use system EQL tape.

Blank = No EQL tape output. Character= Write system EQL tape

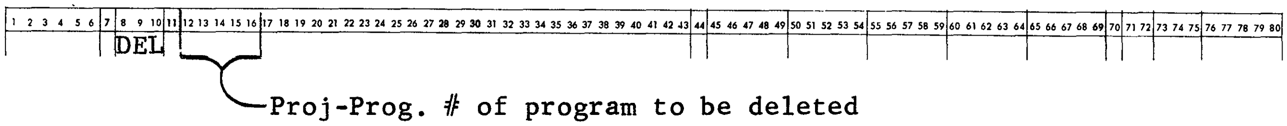
Note: It is possible to resequence and change Project-Program number on initial assemblies.

2.3 The HDG control card is used to change the heading on subsequent printer output. It could be used preceding each assembly if required.



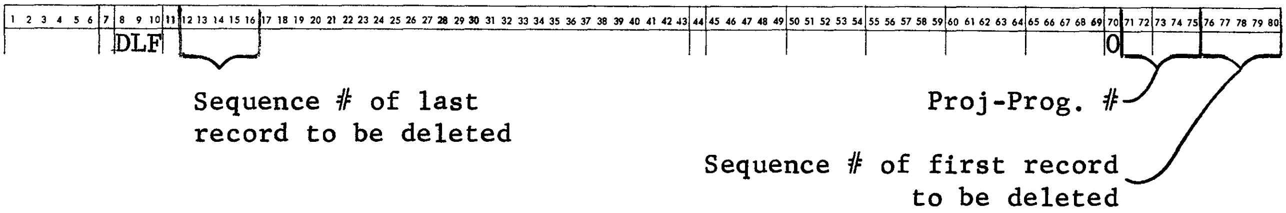
New heading for each page of output

2.4 The DEL control card is used to delete a program from the Input File Tape.



Proj-Prog. # of program to be deleted

2.5 The DLF control card is a special card for block deletion of magnetic tape records.

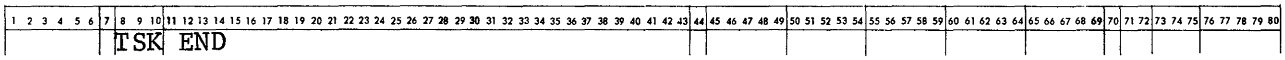


Sequence # of last record to be deleted

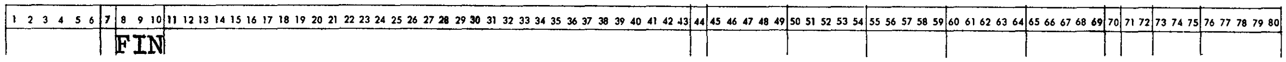
Proj-Prog. #

Sequence # of first record to be deleted

2.6 The TSK END card indicates the end of a point summary table generation or a TASC assembly. It must appear at the end of each assembly to indicate the end of pass 1.



2.7 The FIN control card indicates the end of the processing run. It appears following the last TSK END control card.

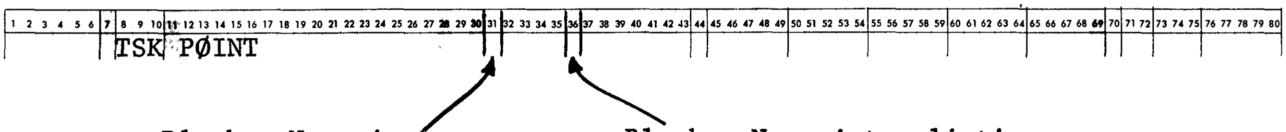


2.8 Point Summary Cards

The point summary enables the programmer to refer to input points, output points, bits, and clocks by their identification number instead of location when coding in TASC.

Point summary cards are used to create the point summary table on magnetic tape which is used by all TASC assemblies for a given system. When the tape is created, the table is also loaded on drum for use by the compiler. On subsequent processing runs, it is called by the first program to be assembled and is retained on drum throughout the processing run.

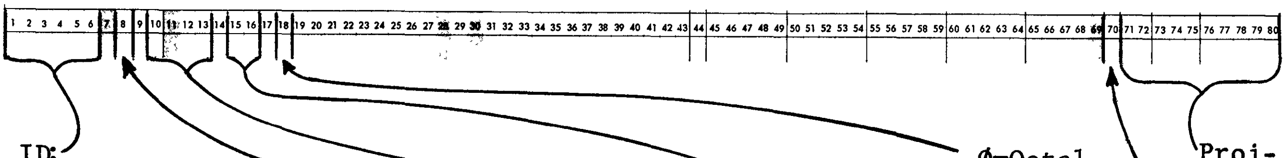
2.8.1 The TSK POINT control card is used to create a point summary tape from point summary input cards and precedes the point summary cards. The point summary table is placed on drum during card processing.



Blank = No point summary input tape to update.
Character = Update previous point summary tape.

Blank = No printer listing.
Character = List point summary on printer.

2.8.2 Input Format



Point ID:
Any combination of alphas, numerics, and blanks to be used as identification throughout system.

Type:
A = Analog
B = Bit
C = Clock
D = Digital
E = Conditional Alarm

Group No:
4-character group no. octal or decimal as specified by col. 18

Position No:
2 character position no. octal or decimal as specified by col. 18

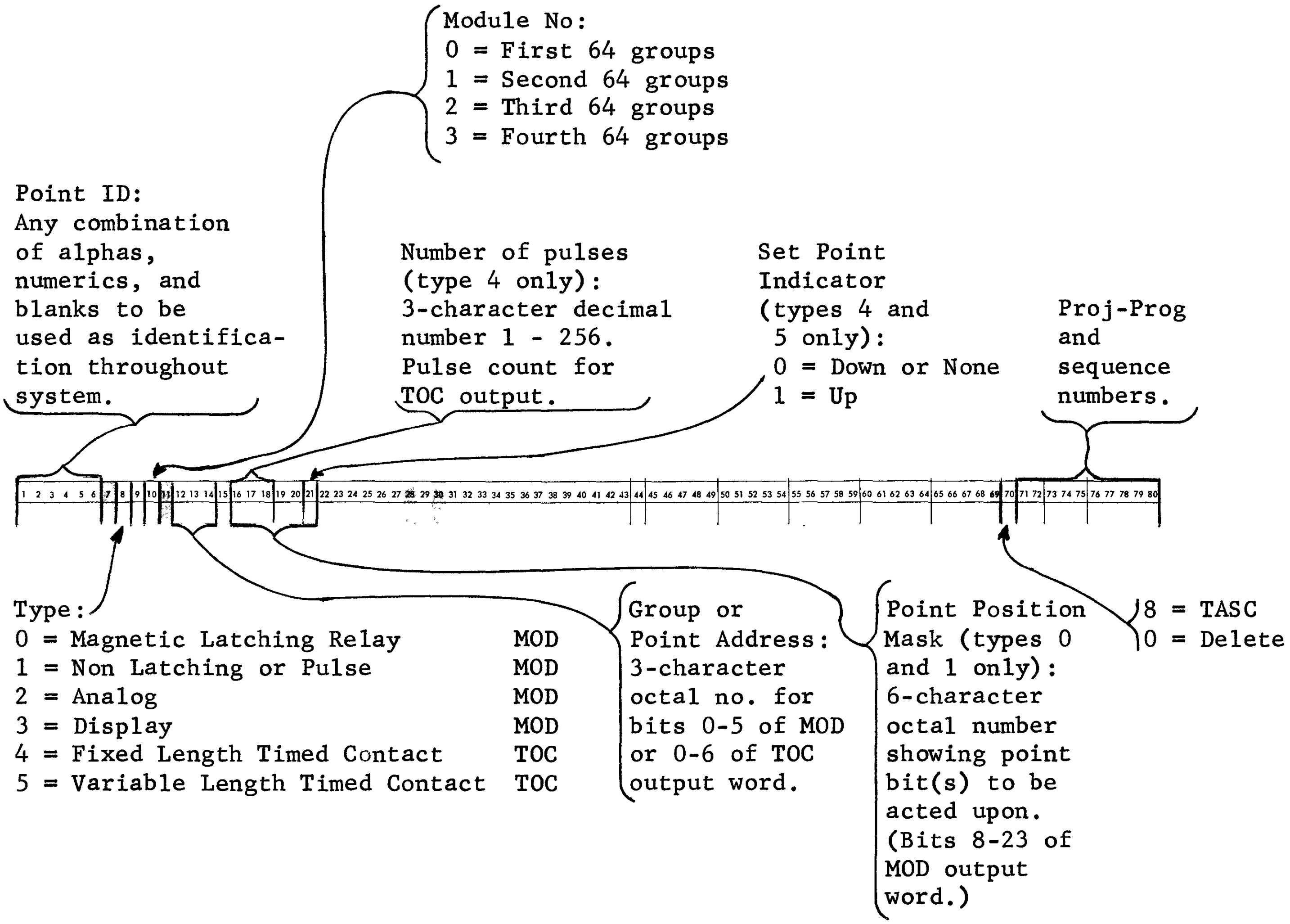
Ø=Octal
D=Decimal

8=TASC
0=Delete

Proj-Prog. and seq. nos.

Note: All fields on point summary cards are right justified except Point ID, which is left justified. In all numeric fields leading zeros are not required, and if zero is desired, the field may remain blank.

2.8.3 Output Format



Module No:
0 = First 64 groups
1 = Second 64 groups
2 = Third 64 groups
3 = Fourth 64 groups

Point ID:
Any combination of alphas, numerics, and blanks to be used as identification throughout system.

Number of pulses (type 4 only):
3-character decimal number 1 - 256.
Pulse count for TOC output.

Set Point Indicator (types 4 and 5 only):
0 = Down or None
1 = Up

Proj-Prog and sequence numbers.

Type:

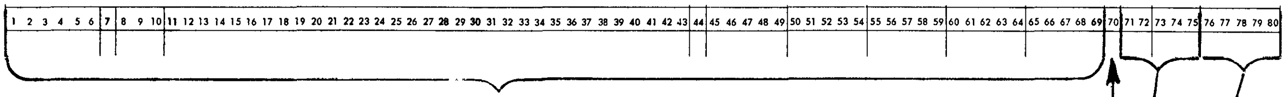
0 =	Magnetic Latching Relay	MOD
1 =	Non Latching or Pulse	MOD
2 =	Analog	MOD
3 =	Display	MOD
4 =	Fixed Length Timed Contact	TOC
5 =	Variable Length Timed Contact	TOC

Group or Point Address:
3-character octal no. for bits 0-5 of MOD or 0-6 of TOC output word.

Point Position Mask (types 0 and 1 only):
6-character octal number showing point bit(s) to be acted upon.
(Bits 8-23 of MOD output word.)

8 = TASC
0 = Delete

2.9 TASC Language cards are used to enter or remove a single entry in the point summary or a program.



Refer to STATEMENT FORMAT (page 1) for proper usage.
Blank if single record delete.

Initial assembly: card to be processed.
8-Corrections: card to be inserted or to replace existing record.
0-Corrections: delete an existing record (See special card for block deletions on page 5.)

Project-Program number

Proj-Prog. # must agree with number on TSK control card and sequence numbers must be in ascending order. (On point summary cards point ID, cols. 1-6, must also be in ascending order.)

3 TASC INSTRUCTIONS

ADA - ASSIGN DRUM ADDRESS

PURPOSE

To assign drum storage for sequence table information.

REMARKS

This Op Code must be used at the beginning of each assembly. Ending address of a memory block may also be specified. Successive ADA records (up to 8) may be used for different drum areas.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field	X			ADA
Point Identity	X			Starting address. Six octal characters right justified. Leading zeros not required.
Status Field			X	
Value Field		X		Ending Address. Six octal characters right justified. Leading zeros not required. Must be last location in memory block. (See ATE)
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

AQ - ANALOG QUESTION

PURPOSE

To determine the status of one or more analog inputs or clocks.

REMARKS

May or may not alarm
May or may not contain outputs
May or may not print conditional alarm

CODING REQUIREMENTS	Mandatory	Optional	Not used	FIRST RECORD CONVENTIONS
Symbolic Location		X		
Op Code Field	X			AQ
Point Identity	X			Point ID of analog input or clock
Status Field	X			H for High; L for Low
Value Field	X			Actual High or Low Limit Decimal, right justify
Try Field		X		Number of times to try if delay and retrial is desired.
Delay Field		X		Amount of delay between retrials. See DLY for restrictions.
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch		X		Symbolic location to branch when no answer

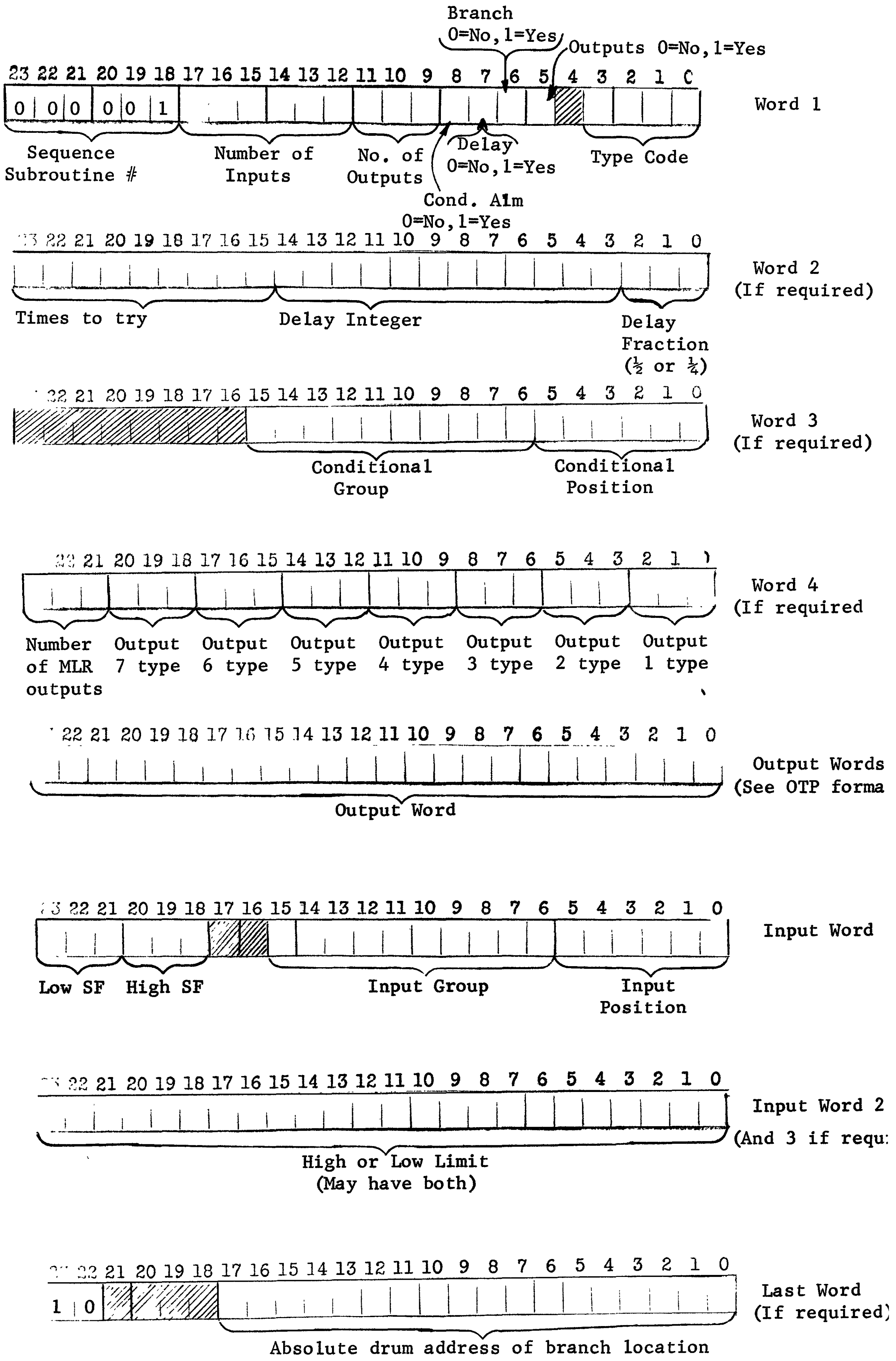
AQ (cont'd)

PURPOSE

Second and succeeding records

REMARKS

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field			X	
Point Identity	X			Point ID of additional input, output, or conditional alarm. Not required for second limit on same point
Status Field	X			H for High; L for Low Desired status for MLR outputs Not required for conditional alarm or other outputs
Value Field	X			Actual High or Low Limit for inputs. Decimal, right justify. Not required for other records.
Try Field			X	
Delay Field		X		Delay for MLR output
A Field			X	
B Field			X	
Type Field			X	
Branch			X	



NOTE: Output and Input words are repeated as many times as necessary

AQM - ANALOG QUESTION-MAGNETIC

PURPOSE

To test the status of an analog input and adjust a piece of equipment using an MLR output.

REMARKS

See flow diagram for usage.

CODING REQUIREMENTS	Mandatory	Optional	Not used	FIRST RECORD CONVENTIONS
Symbolic Location		X		
Op Code Field	X			AQM
Point Identity	X			Point ID of analog input
Status Field	X			H for High; L for Low
Value Field	X			Actual High or low limit Decimal, right justified
Try Field	X			Number of times to try in external loop
Delay Field	X			Delay for external loop. See DLY for restrictions.
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch		X		Symbolic location to branch to when no answer

AQM (cont'd):

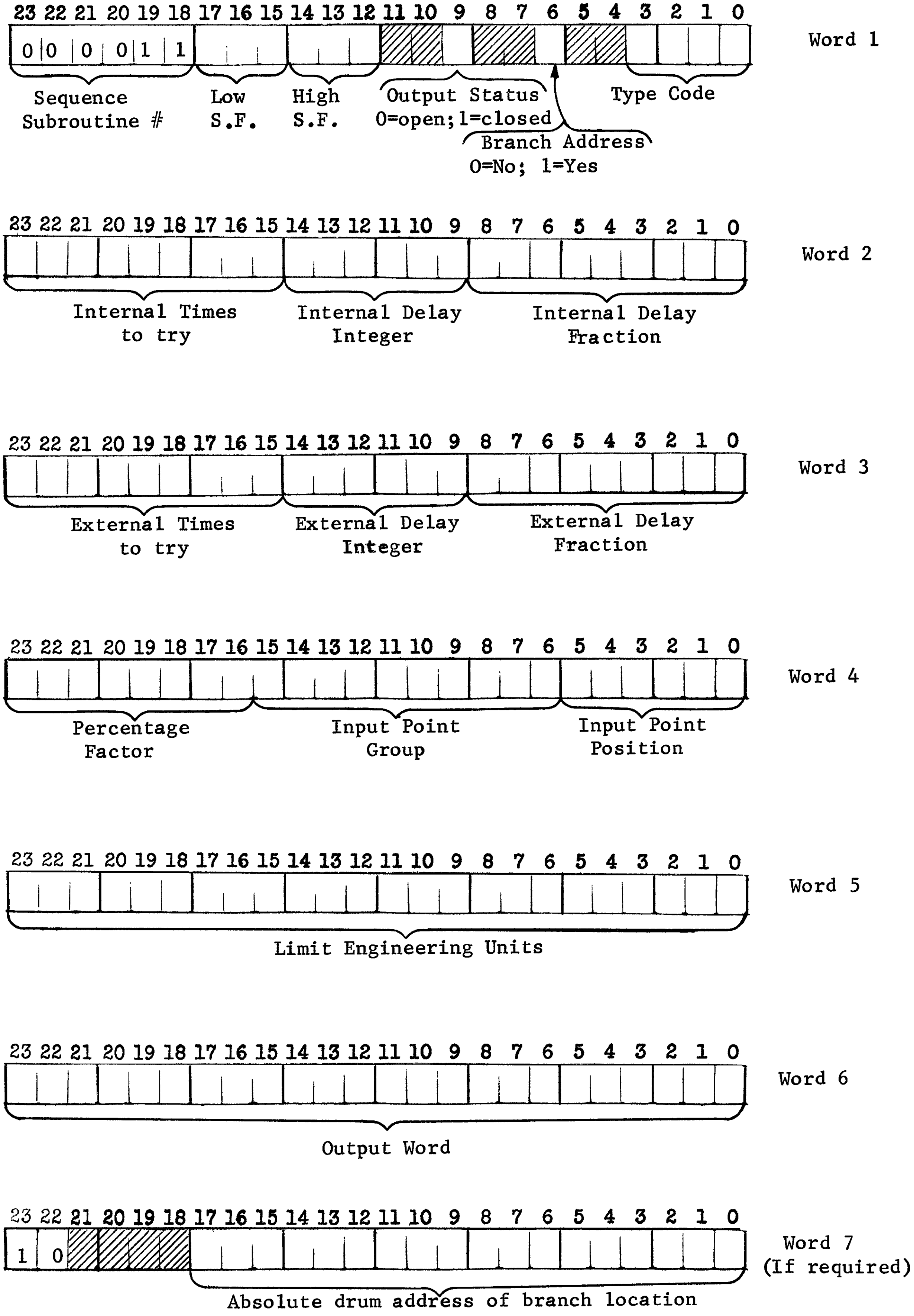
PURPOSE

Second record

REMARKS

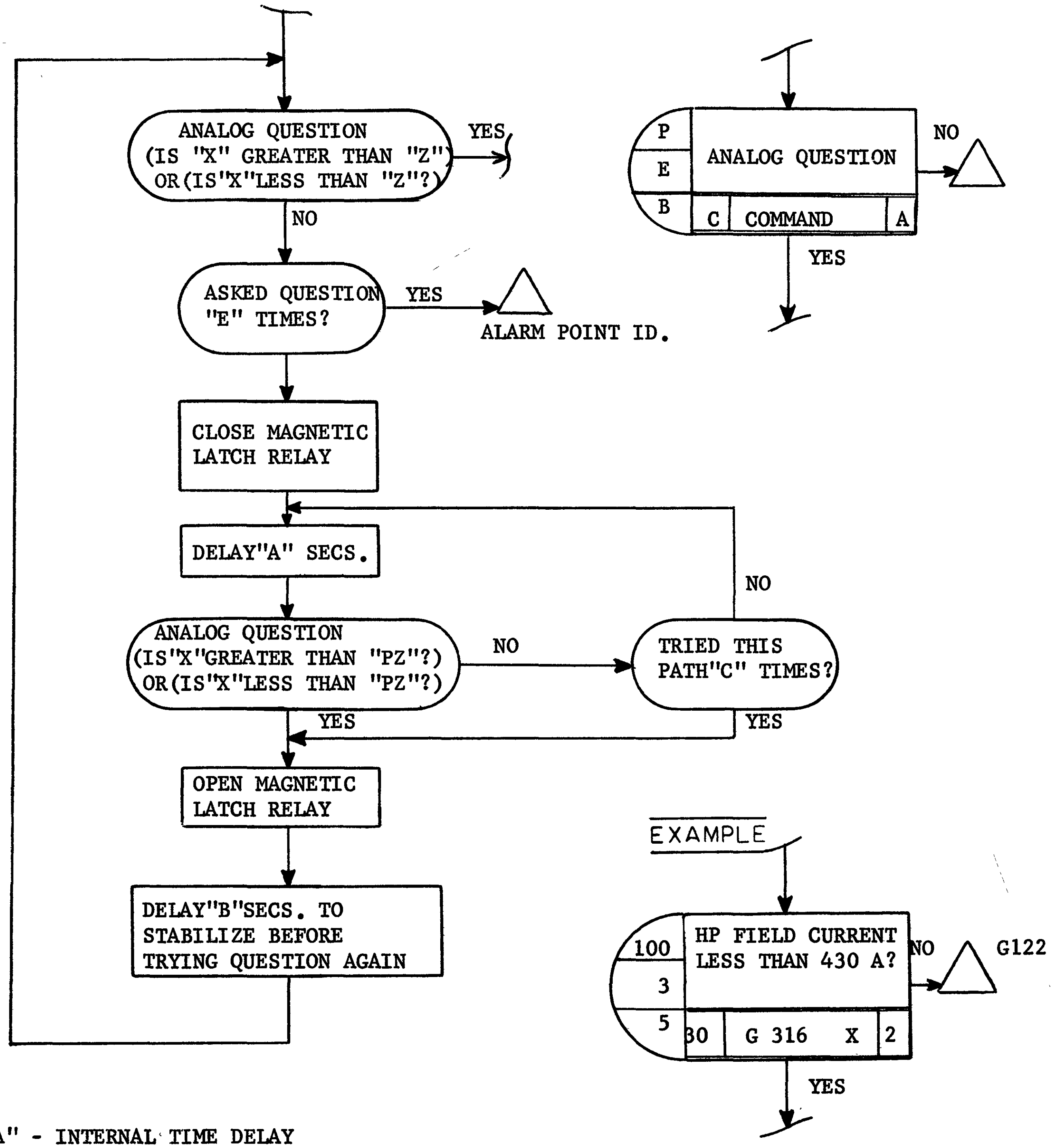
CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field			X	
Point Identity	X			Point ID of MLR output
Status Field	X			Desired status of output: O= Open, C = Closed
Value Field	X			Percentage value for limit in internal loop. Decimal, right justified
Try Field	X			Number of times to try in internal loop.
Delay Field	X			Delay forinternal loop See DLY for restrictions
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

AQM


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Word 1
0 0 0 0 1 1
Sequence Subroutine #
Low S.F.
High S.F.
Output Status 0=open; 1=closed
Branch Address 0=No; 1=Yes
Type Code
Word 2
Internal Times to try
Internal Delay Integer
Internal Delay Fraction
Word 3
External Times to try
External Delay Integer
External Delay Fraction
Word 4
Percentage Factor
Input Point Group
Input Point Position
Word 5
Limit Engineering Units
Word 6
Output Word
Word 7 (If required)
1 0
Absolute drum address of branch location

COMPLEX ANALOG QUESTION (ASK & DO)

EQUIVALENT FLOW DIAGRAM SYMBOL


ANALOG QUESTION (IS "X" GREATER THAN "Z") OR(IS"X"LESS THAN "Z"?)
YES
NO
ASKED QUESTION "E" TIMES?
YES
ALARM POINT ID.
CLOSE MAGNETIC LATCH RELAY
DELAY"A" SECS.
ANALOG QUESTION (IS"X"GREATER THAN "PZ"?) OR(IS"X"LESS THAN "PZ"?)
NO
TRIED THIS PATH"C" TIMES?
NO
YES
YES
OPEN MAGNETIC LATCH RELAY
DELAY"B"SECS. TO STABILIZE BEFORE TRYING QUESTION AGAIN
P
E
B
ANALOG QUESTION
C
COMMAND
A
NO
YES
EXAMPLE
100
3
5
HP FIELD CURRENT LESS THAN 430 A?
30
G 316
X
2
NO
G122
YES

NOTES:

1. "A" - INTERNAL TIME DELAY
2. "B" - EXTERNAL DELAY BETWEEN QUESTION TRIALS.
3. "C" - MAXIMUM NUMBER OF TIMES TO ASK QUESTION IN INTERNAL LOOP
4. "E" - MAXIMUM NUMBER OF TIMES QUESTION CAN BE ASKED IN EXTERNAL LOOP WITHOUT ALARM.
5. "P" - PERCENT OF "Z" (ANALOG LIMIT) REQUIRED TO OPEN RELAY.

ATE - ASSIGN TABLE ANALYZER ENTRY

PURPOSE To inform the assembler of the entry address for the Table Analyzer or Sequence Executive Routine. Also block size, number of filler words at end of block, and the filler word to be used. See also BNB and BRT.

REMARKS

This Op Code must be used at the beginning of each assembly. The block is the portion of sequence table brought into core at one time to be operated on (usually 64 or 128 words.)

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field	X			ATE
Point Identity	X			Table Analyzer Entry Address. Five octal characters right justified.
Status Field			X	
Value Field	X			Number of filler words to leave at end of block. Two decimal characters right justified.
Try Field			X	
Delay Field	X			Block Size. Two decimal characters right justified. Must be power of 2 to be compatible with Monitor.
A Field			X	
B Field	X			Filler word. First two octal characters.
Type Field			X	
Branch	X			Filler word. Last six octal characters.

BEG - BEGIN PROGRAM

PURPOSE

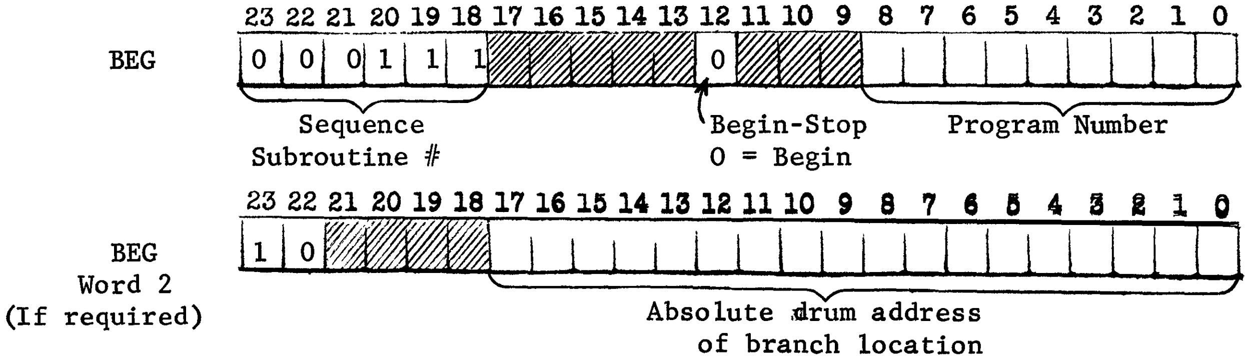
To turn a program on.

REMARKS

Used to set the appropriate program clock to zero and initiate program entry.

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			BEG
Point Identity		X		Alphanumeric program name. For documentation purposes only.
Status Field			X	
Value Field	X			Program Number. Decimal number, right justified



	Mandatory	Optional	Not used	CONVENTIONS
Branch		X		Symbolic location to branch to next.

BNB - BRANCH TO NEXT BLOCK

PURPOSE

To enable starting of routines or sections at the beginning of a block and to leave space for additions or corrections at the end of the current block.

REMARKS

This command will generate a BRU command to the first word of the next block and fill the remainder of this block with filler words.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field	X			BNB
Point Identity			X	
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

BP - BREAK POINT COMMAND

PURPOSE

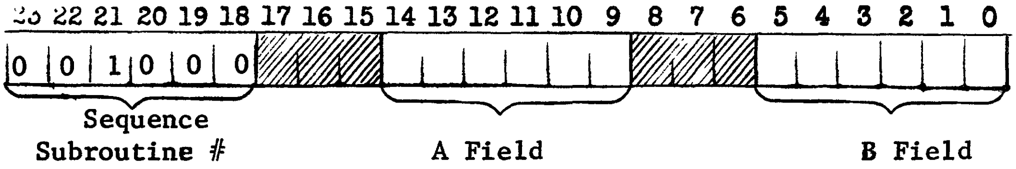
To indicate the current breakpoint.

REMARKS

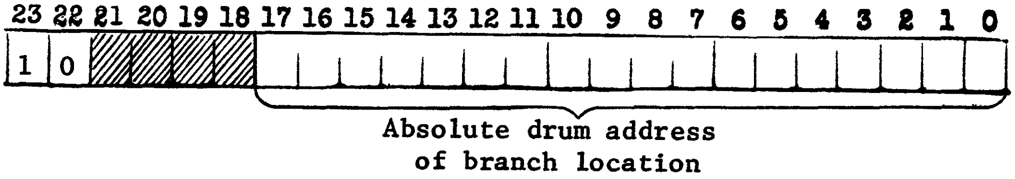
CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			BP

BP



BP
Word 2
(If required)



	Mandatory	Optional	Not used	CONVENTIONS
A Field	X			Previous Breakpoint Number. Two decimal characters.
B Field	X			Current Breakpoint Number. Two decimal characters.
Type Field			X	
Branch		X		Symbolic location to branch to next.

PURPOSE

To furnish linkage between TASC coding and straight line or PAL coding.

REMARKS

May go to the first word of PAL coding or skip as many words as desired. The number of words of PAL coding must be specified to determine the ability to fit in this block.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			BRS
Point Identity			X	
Status Field			X	
Value Field	X			Number of words of PAL coding. Include all generated words for multi-word constants.
Try Field		X		Increment past first word to enter PAL coding.
Delay Field			X	
A Field			X	
B Field			X	

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRS	1	1	▨	▨	▨	▨																		

Absolute drum address
of here + 1 + indicated increment

BRT - BRANCH TO TASC

PURPOSE

To furnish linkage between straight line (PAL) coding and TASC coding.

REMARKS

May have branch specified if desired. Generates SPB in machine language to special entry of the Table Analyzer.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		Symbolic location is used for PAL references only.
Op Code Field	X			BRT
Point Identity		X		Cols 12-17 (PAL Operand) Symbolic location to branch to next.

BRT

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	0	0														

Address of Table Analyzer Specified by ATE

BRT
Word 2
(If required)

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	▨	▨	▨	▨																		

Absolute drum address of branch location

BRU - BRANCH UNCONDITIONALLY

PURPOSE

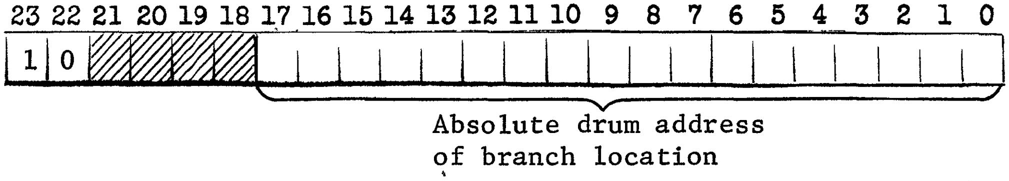
To symbolically branch to another line of logic.

REMARKS

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			BRU
Point Identity			X	
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	

BRU



	Mandatory	Optional	Not used	CONVENTIONS
Branch	X			Symbolic location to branch to.

CQ - CLOCK QUESTION
CQS - CLOCK QUESTION AND SET

PURPOSE: To interrogate a clock with respect to a previously set time, the current time, and the time differential specified. CQS is also used to set the current time in the clock on one path.

REMARKS: To be meaningful, an STC command must be previously executed to establish a stored time. A conditional alarm may be printed on one path if desired.

CODING REQUIREMENTS	Mandatory	Optional	Not used	FIRST RECORD CONVENTIONS
Symbolic Location		X		
Op Code Field	X			CQ or CQS
Point Identity	X			Point ID of clock
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field	X			Time differential. See DLY for restrictions.
A Field			X	
B Field			X	
Type Field			X	
Branch	X			Symbolic address of alternate path

CQ and CQS (cont'd)

PURPOSE

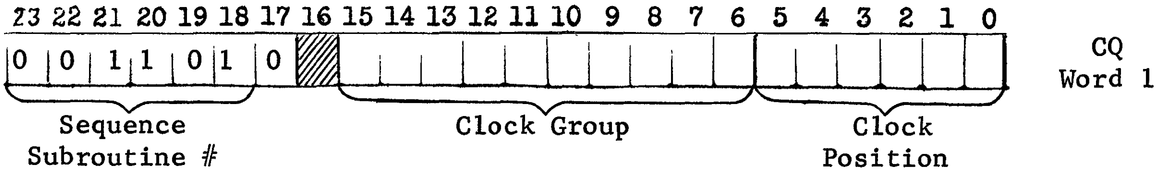
Second record if conditional alarm is required

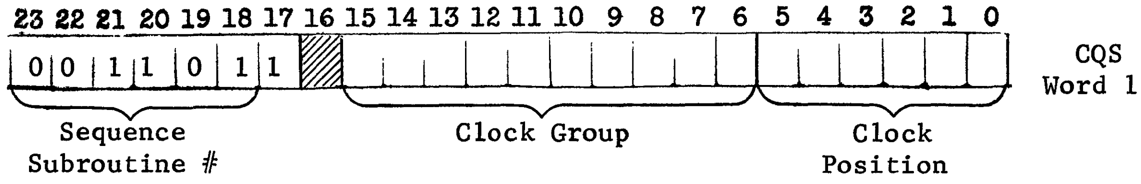
REMARKS

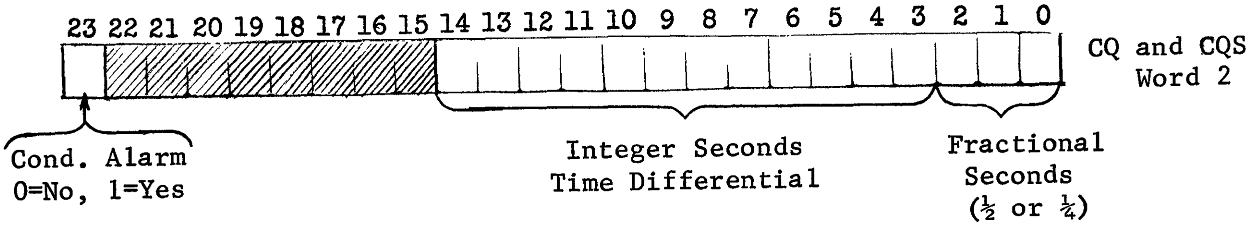
CODING REQUIREMENTS

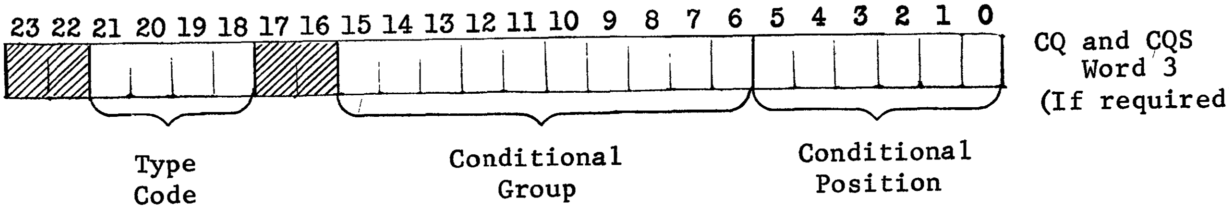
	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field			X	
Point Identity	X			Point ID of Conditional Alarm
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch			X	

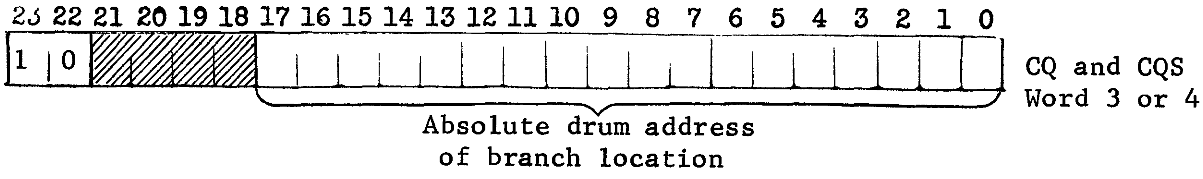
CQ or CQS


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 0 1 0
Sequence Subroutine #
Clock Group
Clock Position
CQ Word 1


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 0 1 1
Sequence Subroutine #
Clock Group
Clock Position
CQS Word 1


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Cond. Alarm
0=No, 1=Yes
Integer Seconds Time Differential
Fractional Seconds (½ or ¼)
CQ and CQS Word 2


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Type Code
Conditional Group
Conditional Position
CQ and CQS Word 3 (If required


23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0
Absolute drum address of branch location
CQ and CQS Word 3 or 4

DLY - DELAY COMMAND

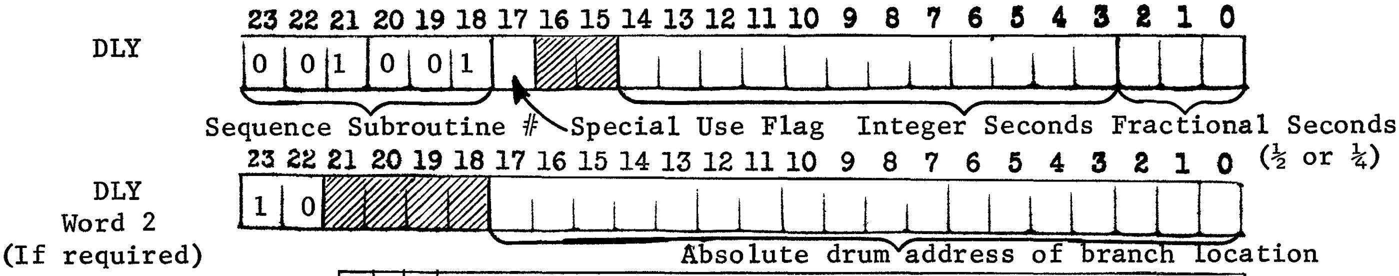
PURPOSE

To set an indicated delay for the program being processed.

REMARKS

DLY is a function of the ECP cycle time.
Delay value may only have the following forms:
XXXS, XXXM, X.XX, X.XS or X.XM

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			DLY
Point Identity			X	
Status Field		X		Special use flag Blank=zero, 1=one in output word
Value Field			X	
Try Field			X	
Delay Field	X			Specified delay. See above for restrictions S=Seconds, M=Minutes. X.XX is assumed to be seconds.



Branch		X		Symbolic location to branch to next.

DQ - DIGITAL QUESTION

PURPOSE

To determine the status of one or more contact inputs or bits.

REMARKS

May or May not alarm
May or may not contain outputs
May or may not print conditional alarm

CODING REQUIREMENTS	Mandatory	Optional	Not used	FIRST RECORD CONVENTIONS
Symbolic Location		X		
Op Code Field	X			DQ
Point Identity	X			Point ID of contact input or bit
Status Field	X			Desired status: o = open orreset, c = closed or set
Value Field			X	
Try Field		X		Number of times to try if delay and re-trial is desired
Delay Field		X		Amount of delay between re-trials See DLY for restrictions
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch		X		Symbolic location to branch to if no answer is found

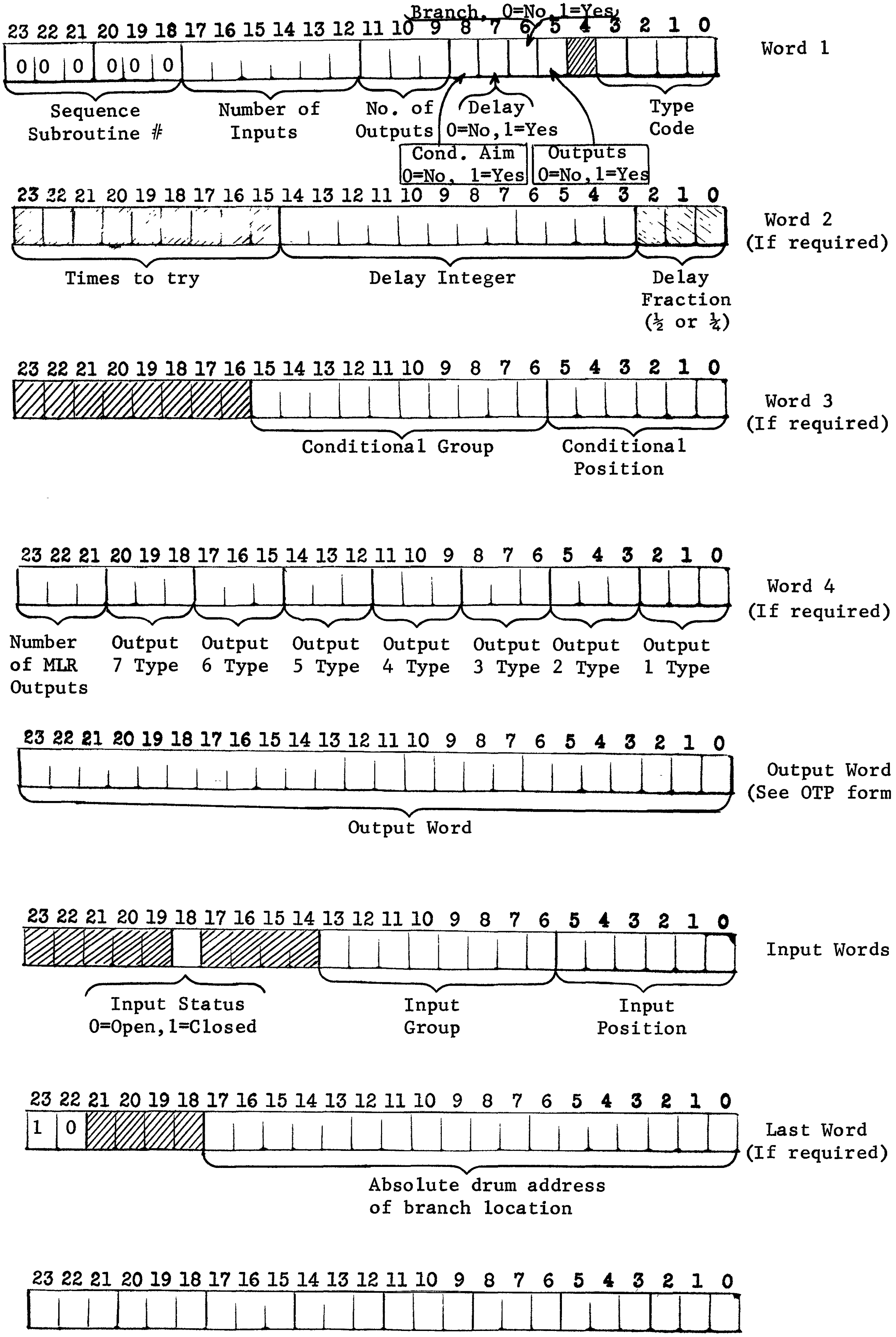
DQ - (cont'd)

PURPOSE

Second and succeeding records

REMARKS

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field			X	
Point Identity	X			Point ID of contact input; or bit for additional inputs; point ID of output point; conditional alarm.
Status Field	X			Desired status for additional inputs or for MLR output. Not required for conditional alarm or other outputs
Value Field		X		Count value for analog output
Try Field			X	
Delay Field		X		Delay for MLR output
A Field			X	
B Field			X	
Type Field			X	
Branch			X	



NOTE: Output and Input Words are repeated as many times as necessary.

DQG - DIGITAL QUESTION GROUP

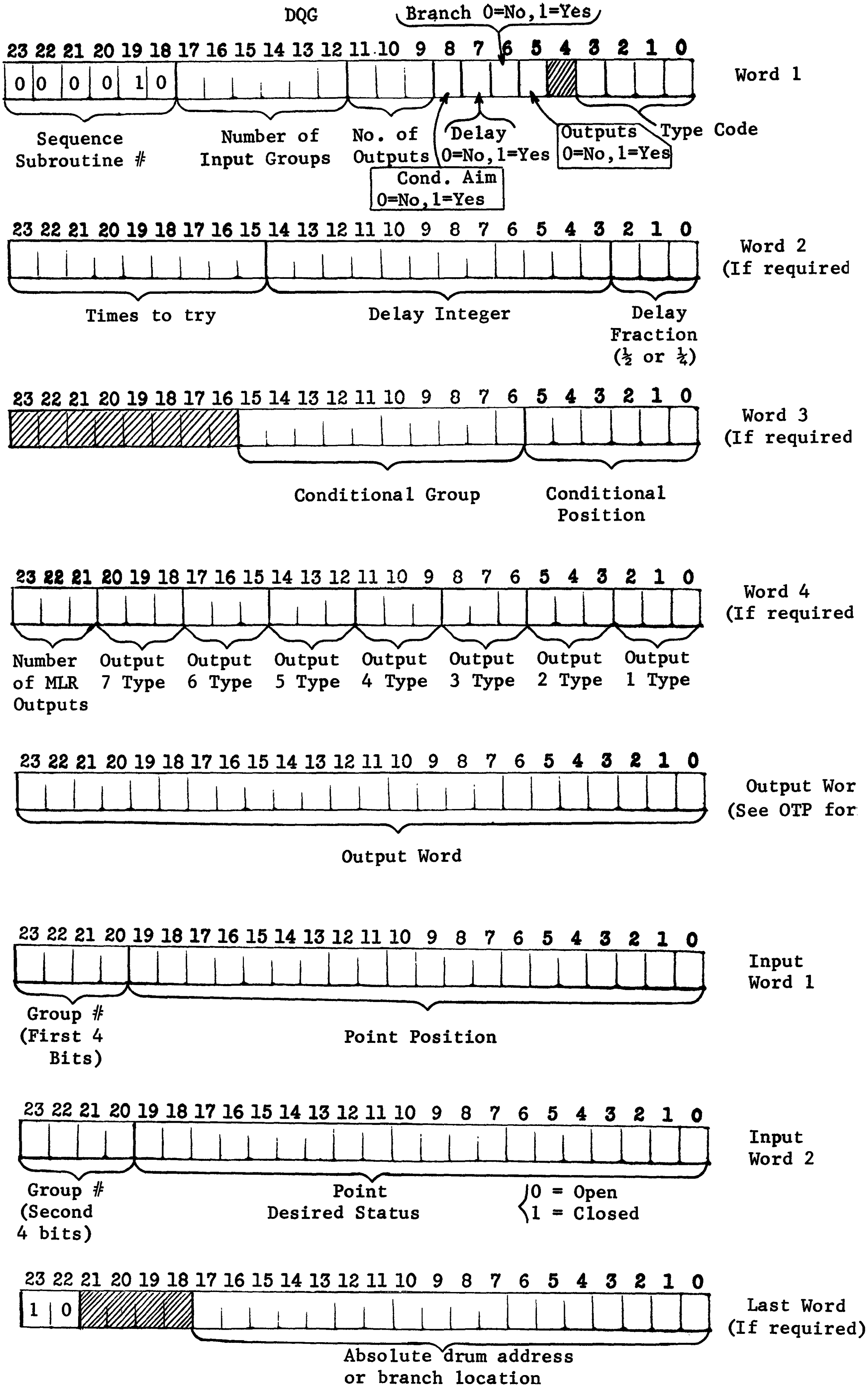
PURPOSE

To determine the status of several contact inputs or bits where they are known to appear at several points in the same group.

REMARKS

See DQ for second and succeeding record formats.

CODING REQUIREMENTS	Mandatory	Optional	Not used	FIRST RECORD CONVENTIONS
Symbolic Location		X		
Op Code Field	X			DQG
Point Identity	X			Point ID of contact input or bit
Status Field	X			Desired Status
Value Field			X	
Try Field		X		Number of times to try if delay and re-trial is desired
Delay Field		X		Amount of delay between re-trials. See DLY for restrictions.
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch		X		Symbolic location to branch to when no answer.



NOTE: Output and Input Words are repeated as many times as necessary.

END - END PROGRAM

PURPOSE

To signify the end of the symbolic information for this assembly.

REMARKS

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field	X			END
Point Identity			X	
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

EQT - EQUATE TASC SYMBOL

PURPOSE

To provide symbol equating from one assembly to another.

REMARKS

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location	X			TASC symbol to be equated
Op Code Field	X			EQT
Point Identity	X			Six octal characters of absolute drum address of symbol. Right justified.
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

ERC - ERASE CLOCK

PURPOSE

To erase the time set in an indicated clock.

REMARKS

The clock is assigned a group and position number by the point summary. The clock status may be interrogated by AQ, CQ, or CQS.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			ERC
Point Identity	X			Point ID of clock to be erased

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERC	0	0	0	1	1	0	1	1																
	Sequence Subroutine #						See SET & RMV		Point Group										Point Position					

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERC Word 2 (If required)	1	0	▨	▨	▨	▨																		
							Absolute drum address of branch location																	

	Mandatory	Optional	Not used	
Branch		X		Symbolic location to branch to next.

GOX - GO "X" TIMES

PURPOSE

To provide a loop counter.

REMARKS

Proper use of this command enables clearing of the counter, controlling the number of times through a loop, and alarming upon completion of the specified number of times. A counter must be cleared before starting to count.

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			GOX
Point Identity		X		Point ID of conditional alarm May be used only if try ≠ 0
Value Field	X			Counter number. Two decimal characters, right justified.
Try Field.	X			Number of times to go through command. Decimal, right justified. Set to zero to clear counter.

GOX Word 1

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	▨	▨																
Sequence Subroutine #								Cond. Alm. Ind. 0=No 1=Yes	Counter No.						No. of times								

GOX Word 2 (If required)

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨	▨	▨	▨	▨	▨	▨	▨																
								Conditional Group										Conditional Position					

GOX Word 3 (If required)

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		▨	▨	▨	▨																		
						Absolute drum address of branch location																	

	Mandatory	Optional	Not used	CONVENTIONS
Branch			X	Symbolic location to branch to. Must be used only if try ≠ 0.

NOTE: Only word 1 will appear when clearing counter

INS - INSERT IN SCAN

PURPOSE

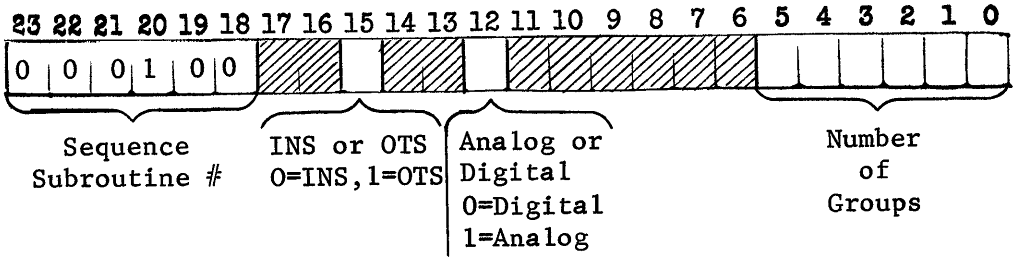
To insert points in scan for alarming.

REMARKS

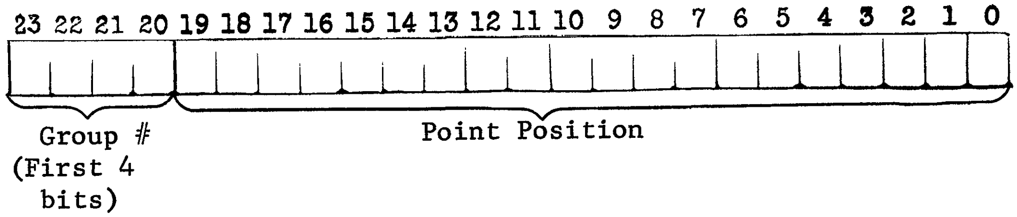
May be used to act upon several points with one INS command. Analog and digital points must be in separate commands.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			INS Required only on first line if several points being acted upon.
Point Identity	X			Point ID of point being acted upon.
Status Field	X			Required for digital points only: O=Open, C=Closed. May optionally contain an A to indicate analog points for documentation.
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch		X		Symbolic location to branch to next

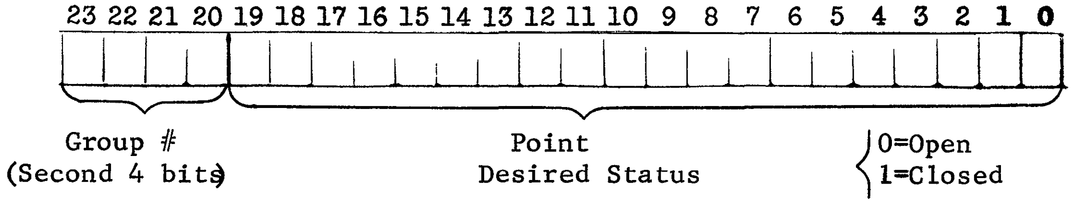
INS



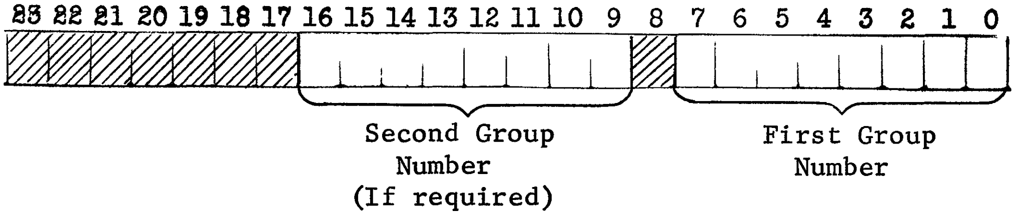
Word 1



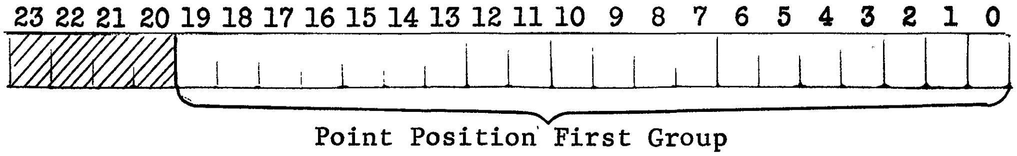
INS Digital Word A



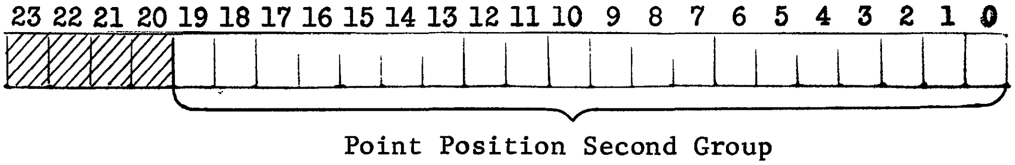
INS Digita Word B



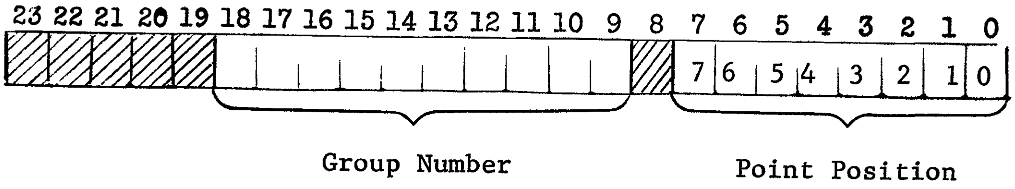
OTS Digita Word A



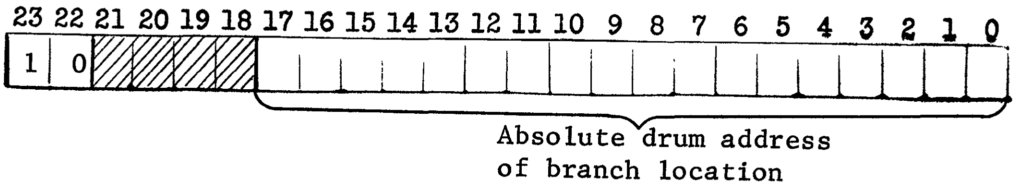
OTS Digita Word B



OTS Digital Word C (If require



INS & OTS Analog Word A



Last Word (If require

NOTE: Words A, B and C are repeated as many times as necessary for the number of groups.

ØTP-OUTPUT COMMAND

PURPOSE

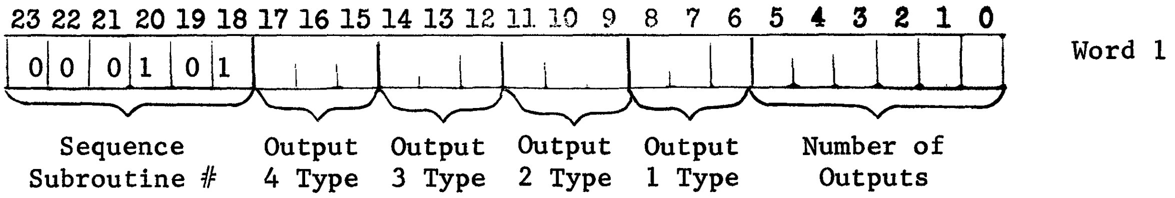
To provide output signals.

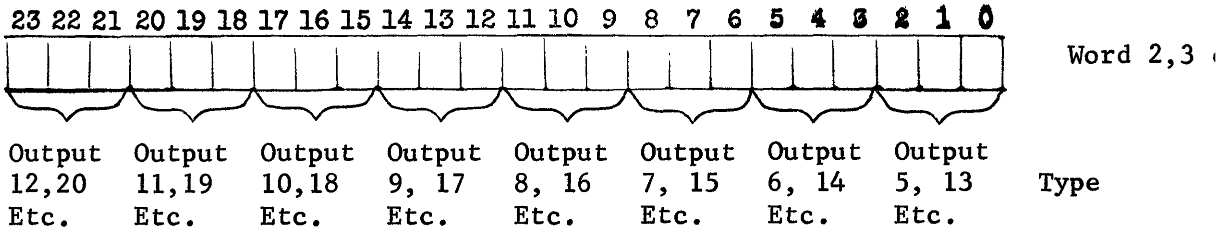
REMARKS

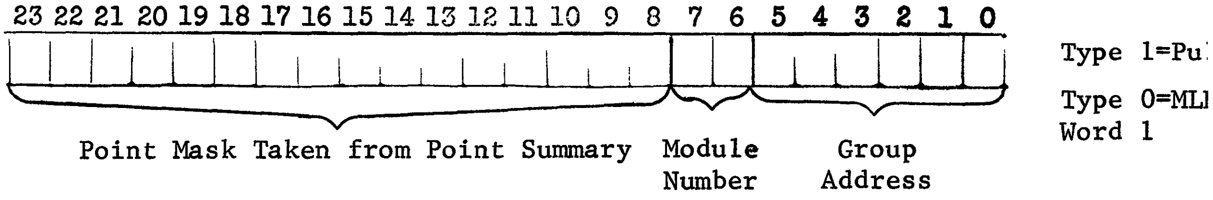
May be used to give several outputs with one ØTP command. Output types may be mixed as required.

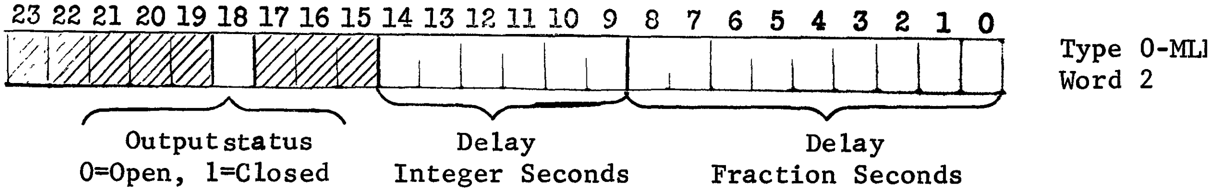
CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			ØTP Required only on first record if several outputs are to be given.
Point Identity	X			Point ID of output point.
Status Field		X		Used to indicate status desired on MLR outputs.
Value Field		X		Used for output counts on analog output number of pulses on TOC output and the four characters on display output. Always decimal, right justified.
Try Field			X	
Delay Field		X		Used to indicate delay on MLR outputs See DLY for restrictions.
A Field			X	
B Field			X	
Type Field			X	
Branch		X		Symbolic location to branch to next

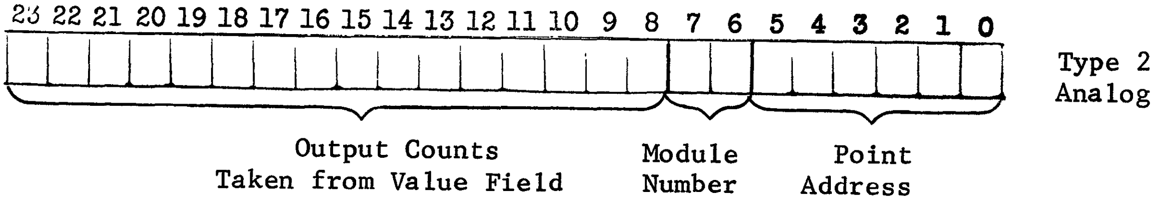
ØTP

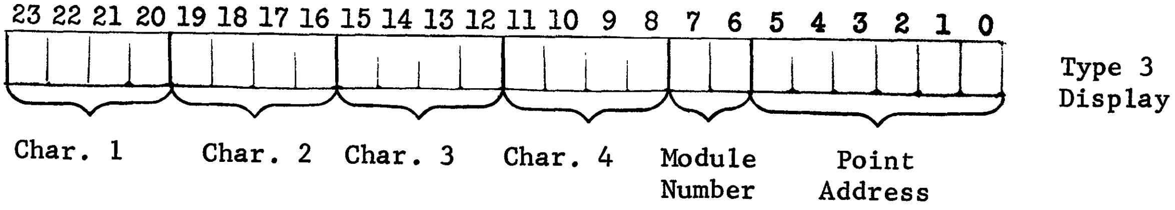


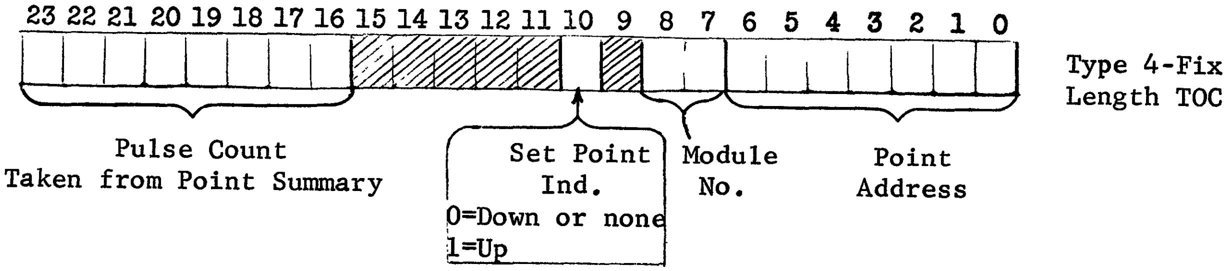


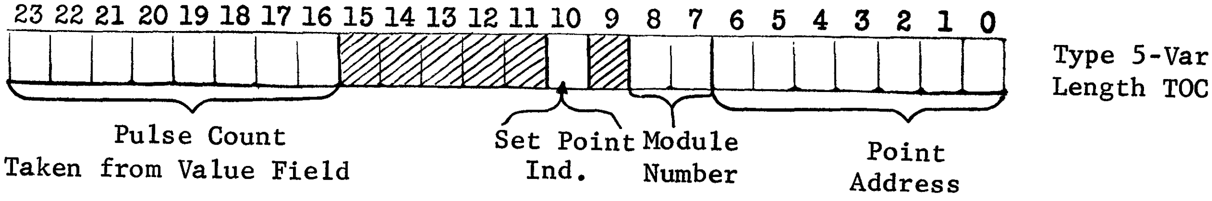












NOTE: Point or Group Address, Module Number, and Set Point Indicator taken from Point Summary.

OTS - REMOVE FROM SCAN

PURPOSE

To remove points from scan for alarming.

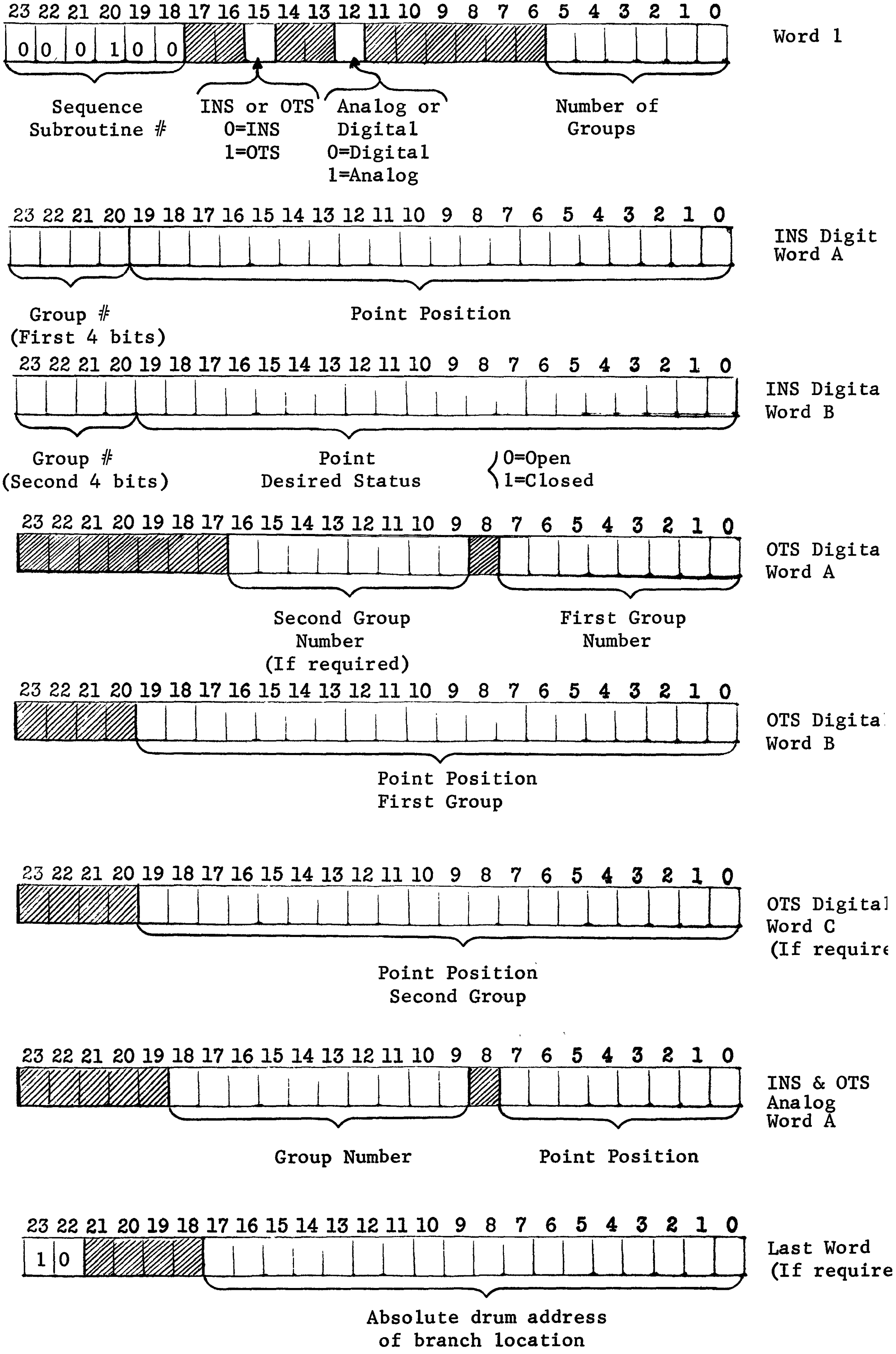
REMARKS

May be used to act upon several points with one ØTS command. Analog and digital points must be in separate commands.

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			ØTS Required only on first line.
Point Identity	X			Point ID of point being acted upon
Status Field		X		May be used to indicate analog or digital (A or D) for documentation.
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch		X		Symbolic location to branch to next.

ØTS



NOTE: Words A, B, and C are repeated as many times as necessary for the number of groups.

PRB - PRINT BLACK

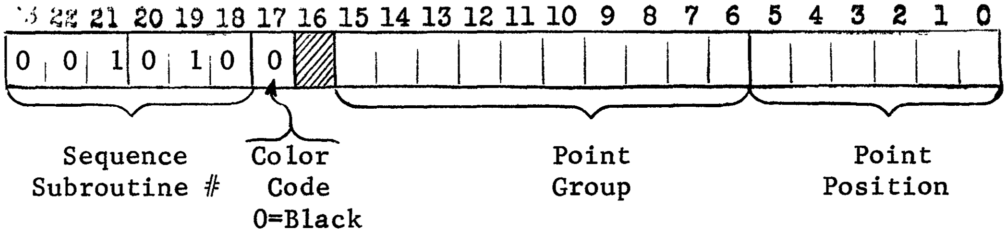
PURPOSE

To print a message and value in black.

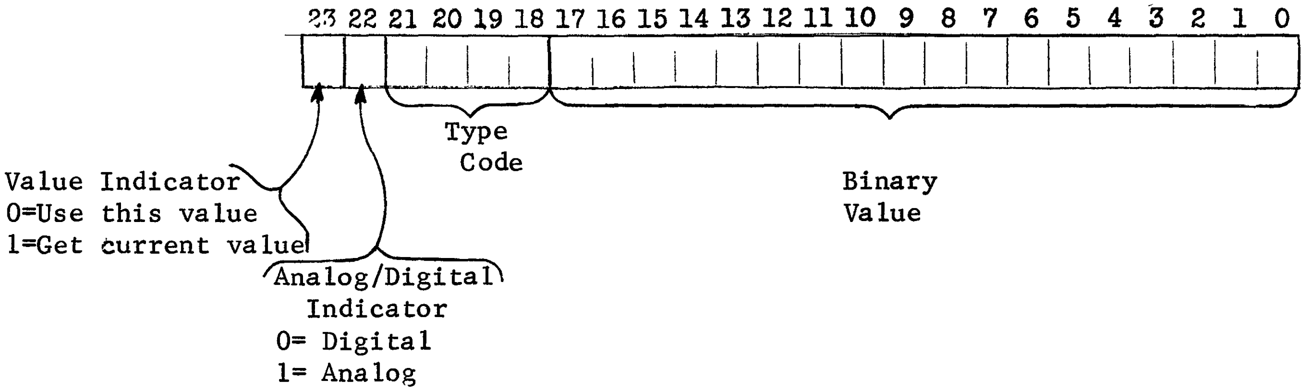
REMARKS

PRB will accept analog, digital, or conditional alarm points assigned by the point summary.

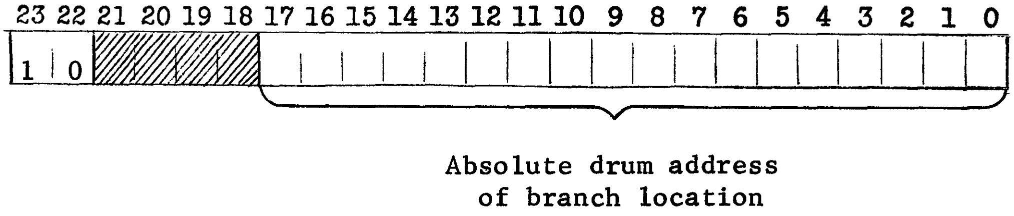
CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			PRB
Point Identity	X			Point ID of point to be printed
Status Field			X	
Value Field		X		Constant value to be printed. If left blank, the actual value of the analog or digital point is to be printed. Value is decimal, right justified.
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch		X		Symbolic location to branch to next.



PRB Word 1



PRB Word 2



PRB Word 3
(If req

PRR - PRINT RED

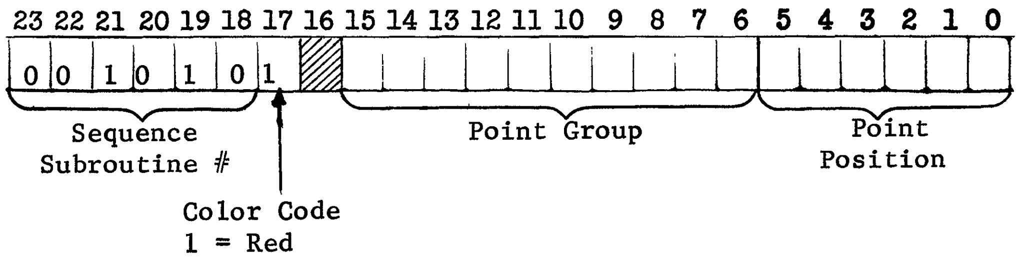
PURPOSE

To print a message and value in red.

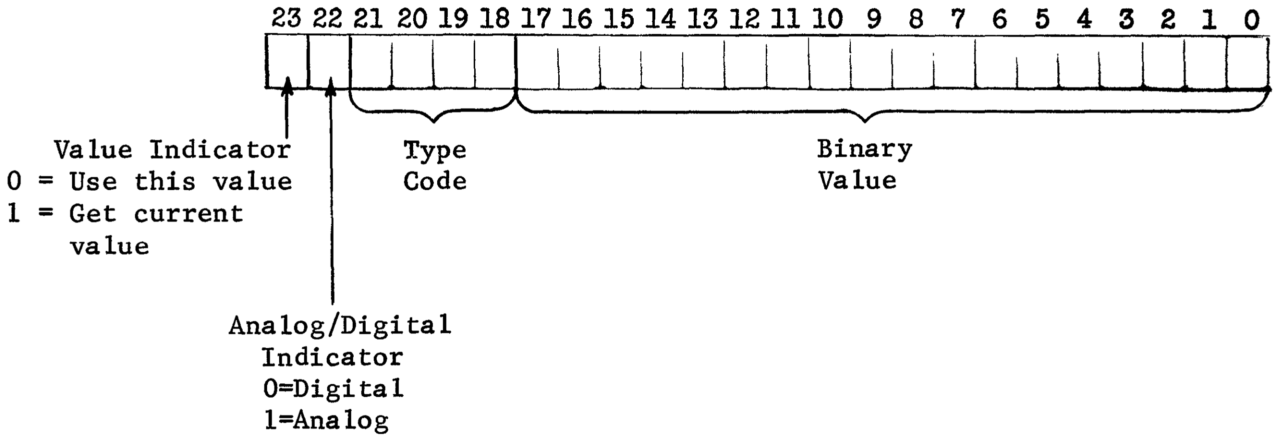
REMARKS

PRR will accept analog, digital, or conditional alarm points assigned by the point summary.

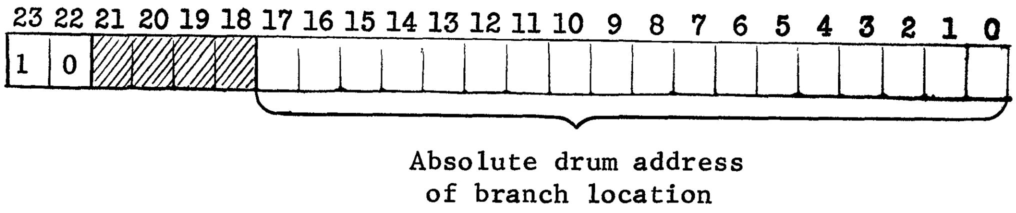
CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			PRR
Point Identity	X			Point ID of point to be printed.
Status Field			X	
Value Field		X		Constant value to be printed or blank
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field	X			Indicate type of alarm: A or P
Branch		X		Symbolic location to branch to next.



PRR Word 1



PRR Word 2



PRR Word 3 (If requi

RMV - REMOVE BIT

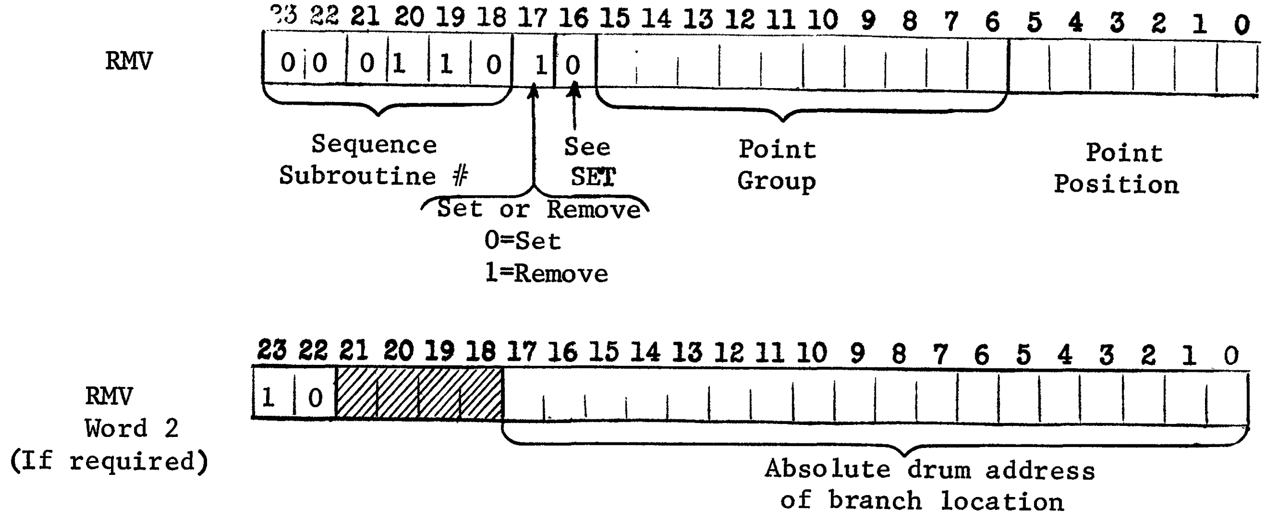
PURPOSE

To set an indicated bit to a "zerd'.

REMARKS

The bit is assigned a group and position number by the point summary list. The bit status may be interrogated by the DQ.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			RMV
Point Identity	X			Point ID of bit to be removed.



Branch		X		Symbolic location to branch to next.

SET - SET BIT

PURPOSE

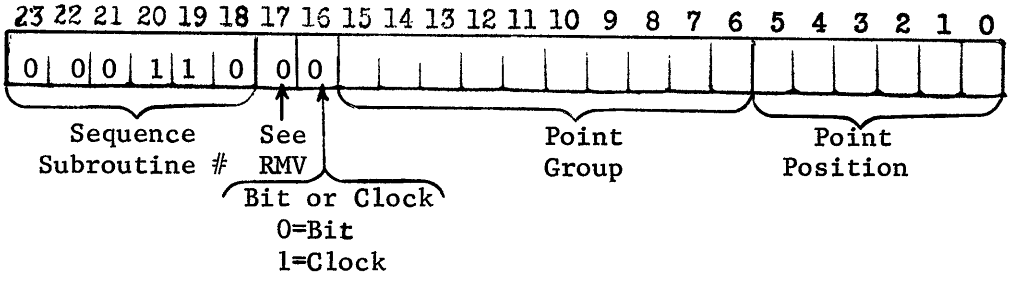
To set an indicated bit to a "one".

REMARKS

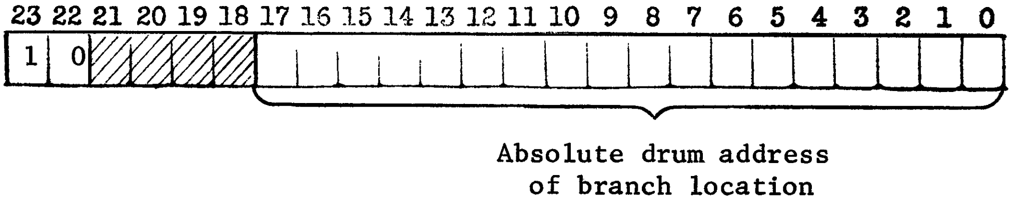
The bit is assigned a group and position number by the point summary list. The bit status may be interrogated by the DQ.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			SET
Point Identity	X			Point ID of bit to be set.

SET



SET
Word 2
(If required)



Branch			X	Symbolic location to branch to next.

SKP — SKIP n LOCATIONS

PURPOSE To provide space in-line for future additions or corrections.

REMARKS SKP will fill the desired number of words with filler words. The number of words skipped does not include the words to remain at the end of a block if the SKP overflows a block. It is assumed that the SKP command is preceded by a BRU command.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field	X			SKP
Point Identity			X	
Status Field			X	
Value Field	X			Number of words to be skipped. Two decimal characters right justified. May not be greater than block size specified in ATE
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

SLW - SLEW PAGE

PURPOSE

To slew printer paper to next page.

REMARKS

This command is used to make better documentation by enabling routines or sections to start printing on a new page.

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location			X	
Op Code Field	X			SLW
Point Identity			X	
Status Field			X	
Value Field			X	
Try Field			X	
Delay Field			X	
A Field			X	
B Field			X	
Type Field			X	
Branch			X	

SPL - SAFE POINT LAST

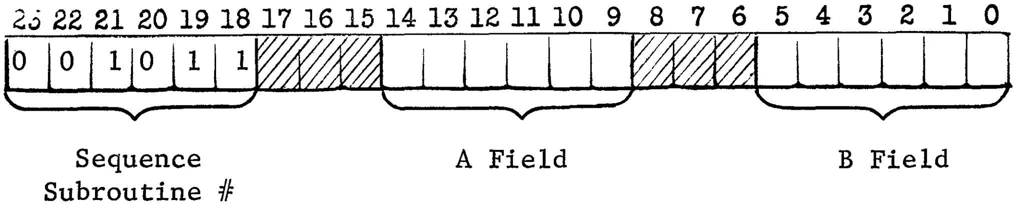
PURPOSE

To indicate the last safe stopping point number.

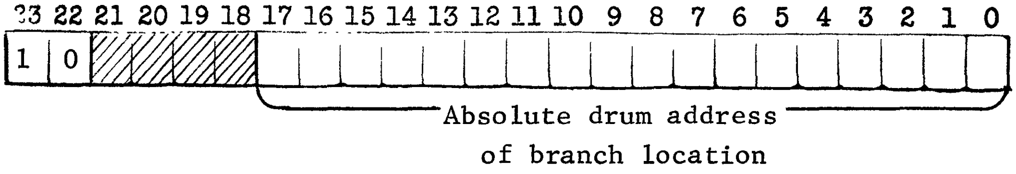
REMARKS

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			SPL

SPL



SPL
Word 2
(If required)



	Mandatory	Optional	Not used	CONVENTIONS
A Field	X			Breakpoint Number. Two decimal characters.
B Field	X			Safepoint Number. Two decimal characters.
Type Field			X	
Branch		X		Symbolic location to branch to next.

SPN - SAFE POINT NEXT

PURPOSE

To indicate the next safe stopping point number.

REMARKS

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			SPN

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPN	0	0	1	1	0	0	▨	▨	▨							▨	▨	▨						
	Sequence Subroutine #									A Field									B Field					

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPN Word 2 (If required)	1	0	▨	▨	▨	▨																		
							Absolute drum address of branch location																	

	Mandatory	Optional	Not used	CONVENTIONS
A Field	X			Breakpoint Number. Two decimal characters.
B Field	X			Safe Point Number. Two decimal characters.
Type Field			X	
Branch		X		Symbolic location to branch to next.

STC - SET TIME IN CLOCK

PURPOSE

To set an indicated clock to the present time.

REMARKS

The clock is assigned a group and position number by the point summary. The clock status may be interrogated by AQ, CQ, or CQS.

CODING REQUIREMENTS

	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			STC
Point Identity	X			Point ID of clock to be set

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STC	0	0	0	1	1	0	0	1																
	Sequence Subroutine #						See SET & RMV		Point Group										Point Position					

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STC Word 2 (If required)	1	0	▨	▨	▨	▨																		
							Absolute drum address of branch location																	

	Mandatory	Optional	Not used	CONVENTIONS
Branch		X		Symbolic Location to branch to next.

STP - STOP PROGRAM

PURPOSE

To turn a program off.

REMARKS

Used to set the appropriate program clock off.
It is assumed next re-entry is set elsewhere.

CODING REQUIREMENTS	Mandatory	Optional	Not used	CONVENTIONS
Symbolic Location		X		
Op Code Field	X			STP
Point Identity		X		Alphanumeric program name. For documentation purposes only.
Status Field			X	
Value Field	X			Program number. Decimal number, right justified.

STP

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	▨	▨	▨	▨	▨	1	▨	▨	▨									

Sequence Subroutine # (bits 23–18)

Begin-Stop 1=Stop (bit 12)

Program Number (bits 8–0)

STP
Word 2
(If required)

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	▨	▨	▨	▨																		

Absolute drum address of branch location (bits 17–0)

Branch		X		Symbolic location to branch to next.

4 OPERATING PROCEDURES

1. Load blank tapes on magnetic tape drives 2 and 3
2. Place cards in reader and initialize
3. Load GE/PAC TASC master tape on drum using BLT01
4. Load OPR02 and call GE/PAC TASC using OPR02 operating procedures

Operator action for typed messages:

Message	Action
GE/PAC TASC	To indicate that blank tapes are mounted on tape units 2 and 3 and that cards are in reader and ready to process, change switch 0.
FIRST CARD NOT JOB	Cards are not positioned correctly in card reader or JOB card is missing. Correct and change switch 0.
OFILE NO ON T2 IS	Place the tape reel number mounted on tape drive 2 in switches 8-19 and change switch 0. Number will be typed for verification. If number is correct, raise switches 1-19 and change switch 0. If number is incorrect, place proper number in switches 8-19 and change switch 0. Number will be typed and program will stop for verification again. Proceed as above for correct or incorrect number.
MOUNT TAPE XXXX ON T1	Mount previous Ofile tape requested on tape drive 1 and change switch 0.
WRONG TAPE MOUNTED	Wrong tape was mounted above. Mount correct tape and change switch 0.
CNTRL CD ERR	Control card is not HDG, TSK, DEL or FIN, or DEL with no input tape. Correct control card, reload card reader and change switch 0.
MOUNT POINT SUMMARY TAPE ON T5	Mount previous point summary for this system on tape drive 5 and change switch 0.
MOUNT BLANK TAPE ON T4	Mount blank tape for point summary output and change switch 0.

OPERATING PROCEDURES (cont'd)

Message	Action
REMOVE AND LABEL TAPE ON T4	Remove and label tape as point summary for this system.
XXXXXX OUT OF SEQ IGNORE XXXXXX DUPE IGNORE XXXXXX TYPE CODE ERROR - NOT ON DRUM	Point summary input card is incorrect as indicated. Return these messages to programmer.
MOUNT EQL TAPE ON T4	Mount previous system EQL on tape drive 4 and change switch 0.
MOUNT BLANK ON T4	Mount blank tape for new EQL on tape drive 4 and change switch 0.
REMOVE AND LABEL EQL ON T4	Remove and label tape as EQL for this system.

Actions taken for peripheral messages are listed in HPS01 - High Speed Peripheral Package writeup.

APPENDIX A - ERROR CODES

The assembler performs validity tests on each instruction. When errors or suspected errors are detected, one of the following indicators will appear on the output listing. The following error codes apply to TASC language coding:

CODE	DEFINITION	CAUSE
P	Point ID Error	1. Input, output, or conditional alarm point indicated is not in point summary table. 2. Wrong type of point indicated for this command (DQ calling for analog point, etc.).
P2	Second Line Point ID	For commands which have special format for second line-either of above reasons.
S	Second Record Missing	For commands which must have more than one line.
T	Too Many Points	Too many points have been indicated for a multi-point command.
U	Undefined Symbol	1. A symbol appeared in branch field which was not defined by appearing in location field of command or EQT. 2. Symbol table was full when the symbol was defined.
The following PAL error codes apply when coding in the PAL language:		
L	Location Field Error	1. First character of the label is not alphabetic, (See Table I, page 25 in PAL manual). 2. Using the DEF pseudo-op when: - the mnemonic assigned is a GE/PAC machine operation. - requesting "Extra Operands" definition when mnemonic has been previously defined as machine-typed operation. - there is an illegal audit code number. 3. Location Field is blank when a symbol is required. 4. Location Field contains a symbol when not allowed.
Ø	Operation Field Error	1. The Op-code not part of the language or was not added to the table through the DEF pseudo-op. This often occurs when definition was attempted but was

		illegal. Consequently, it was not added to the operation table. 2. This op-code cannot be GENerated.
I	Illegal operand	1. Blank operand when an operand is required. 2. Operand not blank when it should have bee 3. One or more required operands missing. 4. Too many operands. 5. Operand value too large. 6. Negative operand value in an instruction that will not accept one. 7. Illegal constant.
X	Index Word Error	1. Index word 1 or 2 specified. 2. Required index missing. 3. Specified index word is greater than seven.
U	Undefined Symbol	Occurs only when a symbol appears in the Operand Field and: 1. It never appeared in the Location Field or on the Common Symbol Tape. 2. It appeared in the Location Field, but the symbol table was full at that time.
C	Illegal Character	A character, not associated with the assembler language, was found in one of the following fields: 1. Location 2. Op-Code 3. Operand (Refer to Table II, page 25 in PAL manual)
M	Multiply-defined Symbol	1. Symbol in Location Field was flagged because: - it has appeared in the same field on a previous record - it appeared on the requested EQL tape w a value unequal to the one being assign - it was saved from a previous assembly with a value unequal to the one being assigned. 2. Any record which references a multiply-defined symbol in the Operand Field will also be flagged.
2	Second Pass definition of symbol different from First Pass	
R	Relative Operand Error	Operand value was relative and should be absolute.
F	Tables full	

GE/PAC 4000

FREE-TIME SYSTEM USER'S MANUAL

December 1965 Revised
Library Control No. YPG19M

This manual published by:
PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. BOX 2918
PHOENIX, ARIZONA - 85002

IN THE DEVELOPMENT OF THE SOFTWARE DESCRIBED, THE GENERAL ELECTRIC COMPANY RESERVES THE RIGHT TO MODIFY THE DESIGN FOR REASONS OF IMPROVED PERFORMANCE AND OPERATIONAL FLEXIBILITY.

GENERAL ELECTRIC
PROCESS COMPUTERS

LE	
FREE-TIME SYSTEM USER'S MANUAL	Library Control No. YPG19M

REVISION CONTROL SHEET

APPROVED BY: F R Weinberger DATE: 11-8-65

REV.	RECORD OF CHANGE	DATE	REV.	RECORD OF CHANGE	DATE
A	Page 2-1; 2.1 - I/O Typer	11/8/65	B	Page 3-1; 3.1 and 3.1.2	12/7/65
	Page 3-4; 3.2 - #6			3-2; elimin. last para.	
	Page 3-6; para. 1			3-4; changed 5 & 6	
	Page 3-7; 3. - para. 3			3-5; elimin. last para.	
	Page 3-8; 1., 2., & 3			3-6; 4 types of control cards	
	Page 3-9; Real-Time, para. 1			3-7; added 4.	
	Page 3-16; Compile, para. 1			3-9; chg. in 1st para. under LOAD	
	Page 3-18; Example 8			3-10; changed para. 3	
	Page 3-23; para. 2			3-11; retitled decks	
	Page 3-24; title change			3-12; changed para. 2	
	Page 3-25; para. 2			3-23; added 1. & paras. 4 & 5 para. 2 changed	
	Page 3-26; para. 2			3-25; added para. 3	
	Page 3-28; Examples 20 & 21			3-26; changed example 18b	
	Page 3-30; para. 2, 3, & 4			3-30; changed I/O	
	Page 3-31; paras. 1 & 2			4-2; eliminated 4.4	
	Page 4-2; para. 4.3			C-1; added A	
	Page A-1; #5			C-3; added C.	
	Page B-1; additional DUMP			F R Weinberger	12/7/6
	Page B-2; PERM. & KEEP added				
	Page C-2; 12. & added 20				
/8/65	F R Weinberger				
	Pages 1, 2, & 3 added	12/7/65			
	2-1; para. 2.2 changed				

					PAG
A	O. L. Jones O. L. Jones 11/65				1
B	Orma Jones 12/65				REISSUE

CONTENTS

INTRODUCTION

This manual describes the capabilities and usage of the Free-Time System and the Language Processing and Debugging System for the
* GE/PAC 4060 computer. Refer to the appropriate manual for further information on FØRTRAN, PAL, and MØNITØR. (See references below.)

The combined purpose of the Free-Time and Language Processing and Debugging Systems is to provide the user with the ability to compile, test, and execute functional programs in a real-time environment. Existing service programs may be initiated by a control card. New programs may be tested and entered into the overall system in easy stages, starting with an untested program and arriving at an operating real-time program.

This system provides "load and go" or "compile and go" with debugging at the symbolic level, using the names of FORTRAN variables. The dynamic relocation of programs and automatic allocation of storage frees the programmer from any concern other than the successful compilation and testing of his program.

References

YPG31M	MONITOR USERS MANUAL
YPG14M	FORTRAN REFERENCE MANUAL
YPG12M	PROCESS ASSEMBLER LANGUAGE MANUAL

*Revised

GENERAL PURPOSE

The primary purpose of a GE/PAC process control computer is to react automatically to real-time process conditions. Even though this reaction may use most of the available time when the process is undergoing a change, some available time is usually present. This time, referred to as free-time, will normally be plentiful and can be used for computations which are independent of the process. These free-time functions must not disturb the reaction time of the real-time system.

Purpose of Free-Time System

An operator, using the Free-Time System, may request the execution of any free-time program not related to the process and some real-time programs which are related to the process. Execution is requested via a control card describing the desired program. Real-time programs are designated by their program-priority numbers in the Process System. Free-time programs are designated by the first six characters of their name.

Purpose of LPD System

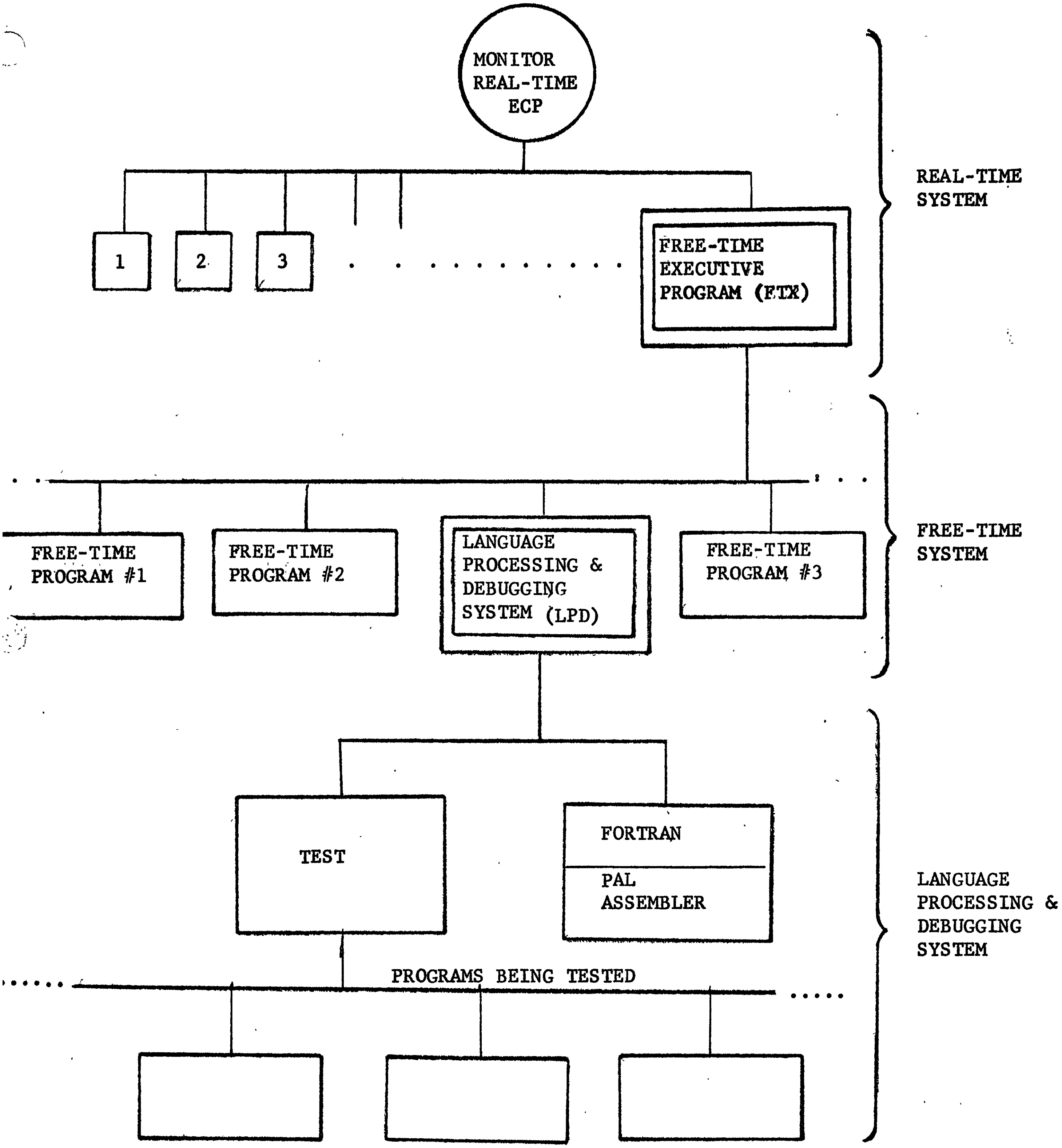
The purpose of the Language Processing and Debugging (LPD) System is to provide the user with the ability to compile, test, and maintain several libraries of programs during the free-time available in the GE/PAC 4060. Both real-time and free-time programs may be processed and maintained by LPD. Provisions also exist to maintain the system common symbol table and the FORTRAN subroutine library. All of these functions are requested by control cards.

- Language Processing - compiles FORTRAN and PAL language programs and enters them into the free-time, real-time, or undebugged library on request.

- Testing - tests the compiled programs under simulated conditions.

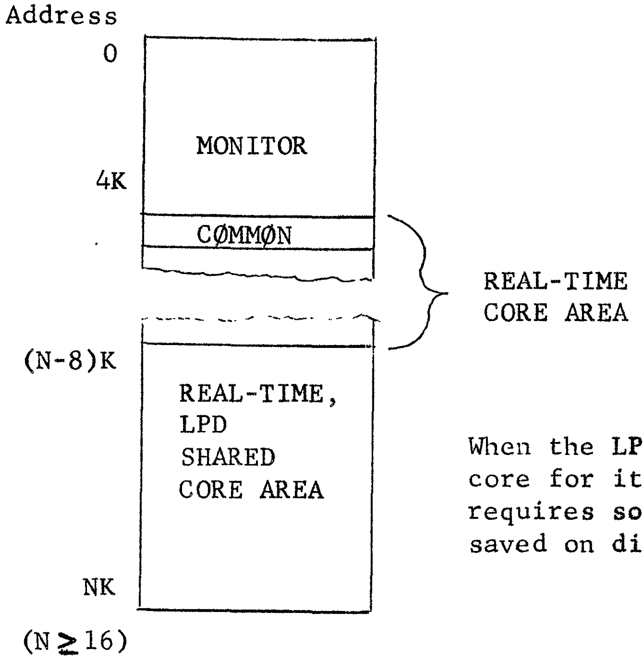
- System Protection - protects the system from the actions of untested programs by using the memory and input/output protection of the GE/PAC 4060.

PROGRAM ORGANIZATION



CORE ORGANIZATION

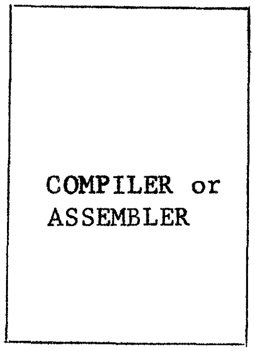
1. GENERAL



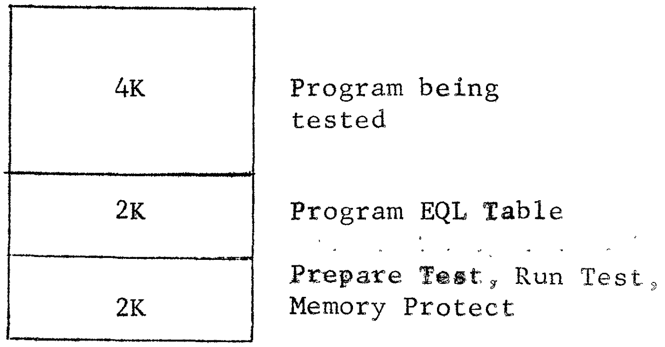
When the LPD is running, it uses this area of core for its programs. If the real-time system requires some or all of this area, this area is saved on disc in a special save area.

2. LPD usage of last 8K

A. Language Processing



B. Testing (numbers are approximate)



DISC ORGANIZATION

Address 0	12K	MONITOR, SUB-MONITOR, OPR
	22K	LPD PROGRAMS
	8K	LPD Save Area
	400	FORTRAN Library Catalog (100 programs, maximum)
	800	Free-Time Library Catalog (200 programs, maximum)
	800	Undebugged Library Catalog (200 programs, maximum)
	2K	System EQL Table
46K	L	Real-Time Library
	M	FORTRAN Library
	N	Free-Time Library
	Ø	Undebugged Library
	P	Temporary Disc Area (LPD Scratch Area)
N		

1. SYSTEM CAPABILITIES

The System has the ability to:

- Request, through control cards, the execution of programs associated with the process (real-time), or programs independent of the process (free-time).
- Compile and/or test programs during on-line operation of the GE/PAC 4060.
- Add or remove programs from the real-time, free-time, or FORTRAN Subroutine libraries.
- Permit modification to the list of common symbols.
- Protect the system against destruction by untested programs.

1.1 SYSTEM PROTECTION

System protection prevents an untested program from destroying information outside its own area or from disturbing the rest of the system.

Any untested program may legally:

- Branch to any location within its own area.
- Store into its own area or permanent core area.

MONITOR and Input/Output Subroutine requests are trapped and interpreted before they are executed, because untested programs must not have access to such devices as the Multiple Output Distributor, Time Output Contact, Scanner, etc.

2. SYSTEM REQUIREMENTS

2.1 Hardware Requirements

Input to the Free-Time and Language Processing and Debugging System is accomplished by punched cards; output by punched cards and a line printer. The following devices are required and have the minimum listed capabilities:

GE/PAC 4060	- 16K Core 0.5 million words of bulk storage
Card Reader	- 350 cpm
Card Punch	- 100 cpm
Line Printer	- 300 lpm
Input/Output Typewriter	

Operator response to the system is through the I/O Typewriter.

2.2 Software Requirements

* The MONITOR used with the Free-Time System contains communication for the peripheral devices listed above.

* Programs in all libraries must conform to the following rules:

* ● All programs must be assembled with a starting location of zero (no ØRG or DCW).

● All programs must maintain the following order:

a. Instruction area, including constants.
b. Storage area - must be BSS's (block reservation).
c. Library routines, including programmer-defined subprograms.

● All storage areas not relative to the program must be in permanent core or bulk storage.

● All input/output requests must be made through standard MONITOR communication.

*Revised

3. SYSTEM DESCRIPTION

3.1 Free-Time Executive Program (FTX)

The Free-Time Executive Program is under direct control of the MONITOR System. It is called via the Real-Time Executive Program by an Input Demand Button on the Card Reader. The Free-Time Executive Program finds the name of the program to be executed by reading a JØB card. When the JØB Card calls for a free-time program, the Free-Time Executive Program makes a request for MONITOR to operate the program.

* If the JØB Card requests a valid real-time program, a request is made to the Real-Time Executive Program to execute the program. Some real-time programs may not be called by a JØB Card. For example, the Output Program is only turned on by the Output Request Subroutine; the Input Program is initiated by the Input Driver, etc.** When a request is made which violates the MONITOR system rules, the JØB Card is ignored, and an error message is printed.

3.1.2 JØB Control Card

The JØB Control Card identifies the beginning of the card deck associated with a job. The JØB card determines the program to be called, and, if it is a free-time program, specifies a heading line for printer output.

* The JØB card is divided into three fields. All fields except the last are terminated by a comma. The last field is terminated by two blank columns. Single blank columns are ignored and may be used anywhere for clarity.

The first field of the JØB card must contain only the word, JØB. The second field contains the program name or number and is interpreted as follows:

- If the first character is numeric, the field is assumed to contain three or less digits which are the real-time program number.
- If the first character is alphabetic, the first six characters of the field are used as the name of a free-time program.
- A comma terminates the program name.

*Revised
**See Appendix A.

The third field is an identification field and is used as the printer output heading for a free-time program. Single blank columns are preserved in this field and the field may not extend past column 69.

The following rules must be observed when preparing a JØB Card:

- The first character for a free-time program must be alphabetic.

- Single blank columns are permitted for ease in reading and are ignored.

- The characters following the comma after the program name are used as the heading line for printer output.

- The card is terminated by two consecutive blank columns or at column 69 (end of heading).

Any JØB card which does not follow the prescribed format is ignored by the Free-Time Executive Program, and the JØB Card is printed, followed by an error message. *

3.1.3 JØB Control Card Examples

```
Example 1:
          Free-Time Program with six letter name and no printer heading
JØB, LØADER

Example 2:
          Free-Time Program with name less than six letters and printer heading
JØB,DUMP,PERFØRMANCECALCS

Example 2A:
          Spaces added for clarity
JØB, DUMP, PERFØRMANCE CALCS

Example 3:
          Abbreviated Form
JØB, CURVEF
          is the same as
JØB, CURVE FIT RØUTINE
          The first six characters (spaces deleted) make up the program name

Example 4:
          Real-Time Program - by Monitor-assigned priority number
JØB, 34
```

3.2 Language Processing and Debugging (LPD) System

The LPD System is used to compile, test and add new programs to the over-all program system.

The LPD System has the ability to:

1. Compile and assemble programs.

2. Compile and execute directly ("compile and go").

3. "Load and go."

4. Output compiled programs to bulk memory or to binary cards.

5. Test programs stored in bulk memory. Octal corrections are permitted. Snapshot dumps, and starting and ending run addresses may be specified.

* 6. Specify a bulk memory area in which storing of information is permitted and the availability of each input/output device.

7. Maintain programs in bulk memory which are:

   a. run under control of the System MONITOR.
   b. run by the Free-Time System.
   c. available to the Debugging System.

8. Maintain a FORTRAN library of subroutines in bulk memory.

9. Maintain a table of symbols and their values or locations which have been specified as "common" and to provide the ability to:

   a. add the common symbols, from a program which has been compiled, to the common symbol table.
   b. remove the entire common symbol table.
   c. load a new common symbol table from binary cards.

*Revised

The LPD System consists of:

1. A series of programs, each designed to do a particular function, such as compile, test, read cards, punch cards, print and check card validity.

2. A sequencing program which processes control cards and chooses the programs to accomplish the desired function.

Programs may be entered into the system by one of two methods:

* ● Compiling FORTRAN language statements.

* ● Loading binary cards produced by the GE/PAC language processor.

The program name or number and the particular library into which this program is being placed are specified on the compile or load control card preceding the input deck.

Program libraries exist for the following:

* (1). Real-Time Programs (numbered programs, 1, 2, 3, ...) executed directly by MONITOR. The system status of these programs must be specified when the program is entered in this library.

(2). Free-Time Programs (six character alphanumeric names, LOADER, DUMP, etc.) executed by FTX. Free-time programs may not directly initiate the execution of (turn on) real-time programs.

(3) Either real or free-time programs, which have not been tested, executed by LPD.

*Revised

3.2.1 Control Cards

The order of parameters on control cards is immaterial, since the names of the fields uniquely identify the action to be taken. However, the first field must identify the type of Control Card.

Single blank columns may be used for formating and clarity. They are ignored by the LPD System. Two blank columns in a row signify the end of information on a card.

* Four types of control cards exist for the LPD System.

1. Actions to be taken by the LPD System.

   a. FIN - signifies the end of a job. All programs in temporary status are removed and control is returned to FTX which will process the next JØB.

   b. DIAGNOSTIC COMPILE - initiates a FORTRAN compilation and no object program is produced.

   c. COMPILE - initiates a FORTRAN compilation or a PAL assembly, and transfers the object program to bulk memory.

   d. LOAD - loads the following binary program or symbol table card deck into bulk memory.

   e. TEST - transfers a free-time untested program to core for execution by LPD.

   f. REMOVE - removes from the system a program or symbol table which has been entered by LOAD or COMPILE.

   The REMOVE card can be followed only by an action control card. The FIN can be followed only by a JØB card.

   Each of these cards, except REMOVE and FIN, are followed by either language cards to be compiled, binary deck to be loaded, or cards giving TEST operating conditions or data.

2. The TEST operating condition cards are CORRECT, DUMP, and EXECUTE. These cards modify the program, specify the breakpoints for dumping, and supply additional information for processing enviornment.

*Revised

2. a. CORRECT - corrects the program as it exists in core using symbolic or numeric addresses and octal contents.

   b. DUMP - gives locations and limits for snapshot dumps.

   c. EXECUTE - executes the program between given locations.

3. The System Protection cards are BULK STORE, INPUT DEVICES, OUTPUT DEVICES, and PROGRAM. These cards specify the memory and input/output protection during the testing of a program, and defines those real-time programs which may be legally initiated by that program.

   a. BULK STORE - specifies the bulk area into which the program being tested may write.

   b. INPUT DEVICES - specifies the input devices which the program being tested may use.

   c. OUTPUT DEVICES - specifies the output devices which the program being tested may use.

   d. PROGRAM - allows the program being tested to initiate the execution of programs in the Real-Time Library through the Real-Time Executive Program.

   All of the above cards have a variable format with fields separated by commas. Each field is uniquely identified by the first six characters or by the prepositions "TO," "FROM," or "AT." All fields, except the first field (card name), may appear in any order. If there are less than six characters in a program name, blanks are added for the remaining characters to form a unique six-character identifier. (Any field may exceed six characters.)

   Numeric addresses are assumed to be relative to the beginning of the program except on the BULK STØRE Card.

* 4. Other control cards are:

   a. END COMPILATION - indicates the end of a FORTRAN deck.

   b. END TEST - indicates the end of a program test deck.

   c. REAL TIME STATUS - specifies the operating parameters for a PERMANENT program running under the Real-Time Executive Program.

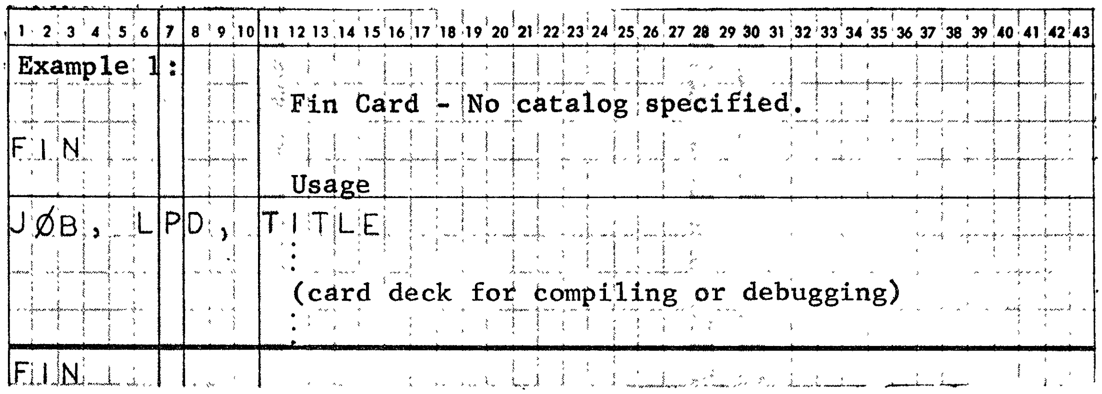
*Revised

3.2.2 Control Card Usage

"Action-to-be-taken" Examples

FIN

FIN signifies "end of job" and removes all programs in the system that are in temporary (initial testing) status.



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Example 1:
                          Fin Card - No catalog specified.
FIN
                          Usage
JØB, LPD, TITLE
                          :
                          (card deck for compiling or debugging)
                          :
FIN
```

An option on the FIN cards allows the programmer to determine what programs are in the system and how much bulk storage is used by them.

If a catalog is requested, the following items are listed.

1. Names (Numbers) and sizes for:
   - a. Real-Time Programs in "permanent" status.**
   - b. Free-Time Programs in "permanent" status.
   - c. Programs in the FORTRAN Library.
   - d. Programs (either Real-Time or Free-Time) in "keep" status.
2. Total bulk memory required for:
   - a. Real-Time Programs
   - b. Free-Time Programs
   - c. FORTRAN Library Subroutines
   - d. Programs in "KEEP" Status.
3. Available Bulk Memory

**Programs in PERMANENT status are under direct control of the Free-Time or Real-Time Executive Programs. Programs in KEEP status may be run only under the LPD System.

```
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Example 2:
          FIN Card - Catalog Specified
FIN,CATALØG
```

When a FIN Card is missing or another card error results in ignoring the FIN Card, a subsequent JØB Card may appear to be out of sequence. If this happens, the JØB Card is treated as a FIN Card, and the job is ignored.

LOAD

The LOAD Control Card loads a binary program deck and symbol table deck into bulk memory. Programs are identified by the program number (three digits), or the first six characters (excluding blanks) of the program name. If no name or number is given, the LPD system assigns the name TEMP01 (temporary status) to the program. An attempt to load subsequent programs with no name stated, prior to a FIN Card, causes these programs to be ignored. The status of the program may be specified either KEEP or PERMANENT. If no status is specified, the program is placed in temporary status and is removed by a FIN card.

Programs in the KEEP state are retained in bulk memory following a FIN Card and may only be called by the LPD program via a TEST Card. Programs in the PERMANENT state are also retained in bulk memory following a FIN card, but they may be initiated either by the Free-Time Executive Program via a JØB Card or by the Real-Time Executive Program via a program turn-on.

Programs placed in PERMANENT status must be identified by:

1. a number for a real-time functional program operating under the Real-Time (MONITOR) Executive Program. (This LOAD Card must be followed by a REAL TIME STATUS Card).

2. a symbolic name for a free-time library program which is called with a JØB Card.

REAL TIME STATUS

A PERMANENT program operating under control of the Real-Time Executive Program must have its operating parameters specified on a Real-Time Status Card. The following states are assumed if no status is specified on the REAL TIME STATUS Card:

1. Interrupts will be permitted to occur.

2. The program's core area is available for use by higher priority programs.

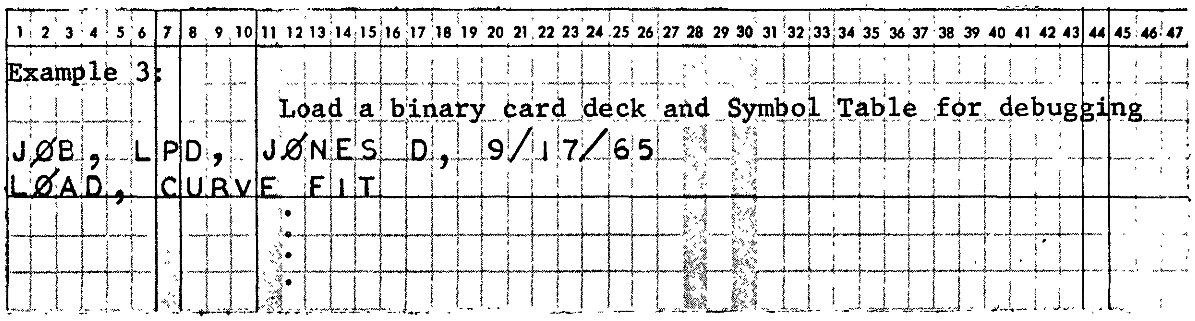
3. The program can be initiated by a JØB Card.

Operating conditions other than those shown above may be specified as follows:

1. INHIBITED
2. UNAVAILABLE
3. NØ JØB

Every LØAD or CØMPILE card specifying a PERMANENT real-time (numbered) program must be followed by a REAL TIME STATUS Card.

An additional status, FORTRAN LIBRARY, is used for programs which were compiled as FORTRAN LIBRARY subroutines. These subroutines are loaded without symbol tables.

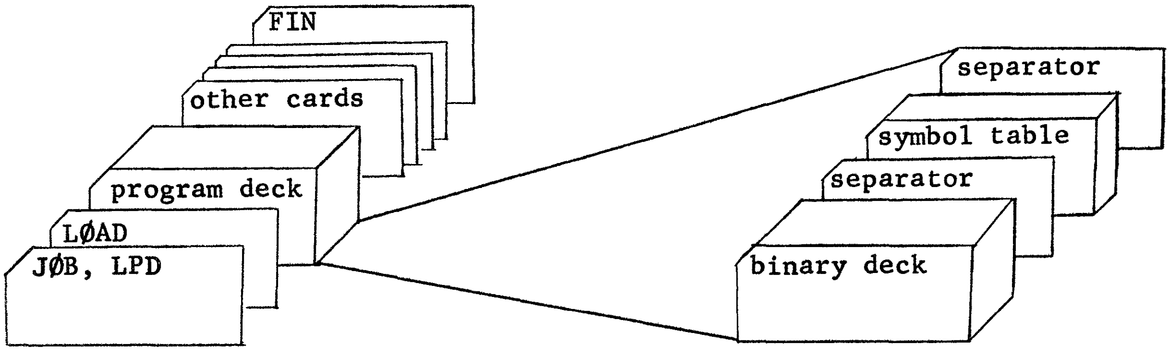


Example 3: Load a binary card deck and Symbol Table for debugging

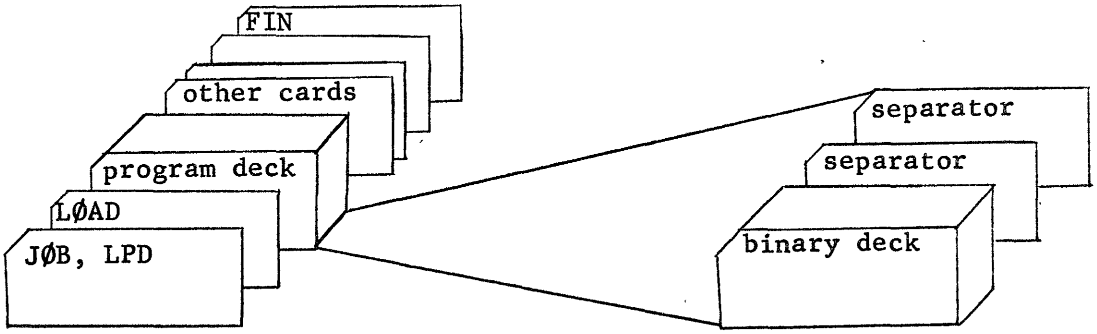
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
JØB, LPD, JØNES D, 9/17/65
LØAD, CURVE FIT
          .
          .
          .
```

The program is in temporary status and will be removed from the system by a FIN card.

* To load a binary deck produced by the Free-Time System.



* To load a binary deck from some other source.



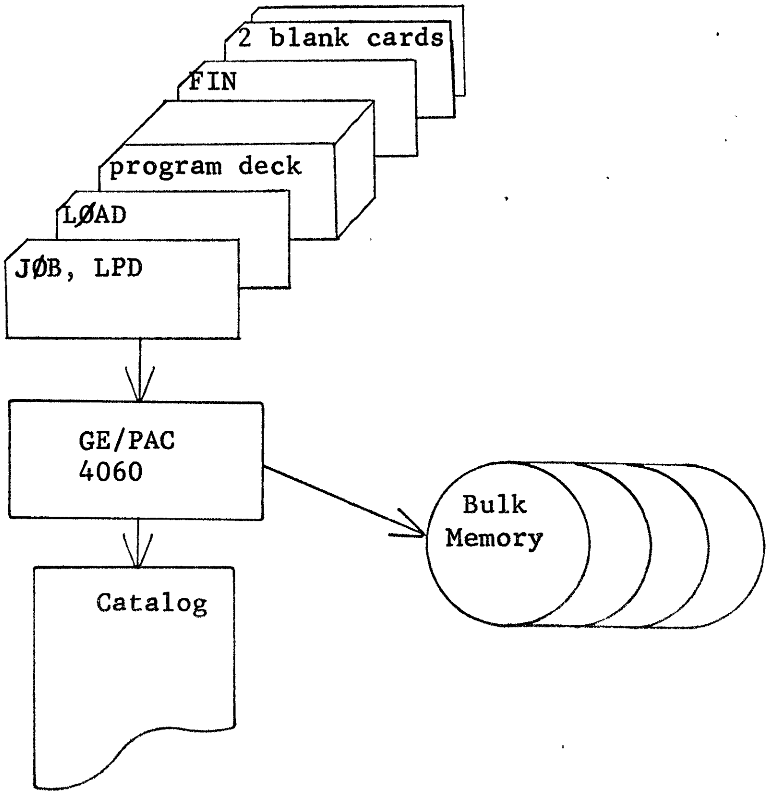
* A symbol table will be expected by the loader. If none exists, an extra blank card must be included.

Example 4: Load a program for subsequent debugging and list the catalog

```
JØB, LPD, JØNES D. 9/21/65
LØAD, CURVE FIT, KEEP
          .
          .
          .
          (Binary Deck)
          .
          .
FIN, CATALØG
```

The program is placed in KEEP status. It is not removed by the FIN Card and need not be reloaded for subsequent testing.

A listing of all free-time, real-time, and FORTRAN Library programs and the bulk storage allocation is produced on the printer.



Example 4a

Load a finished program into the free-time library. The program is to remain in bulk memory for regular use by Programmers.

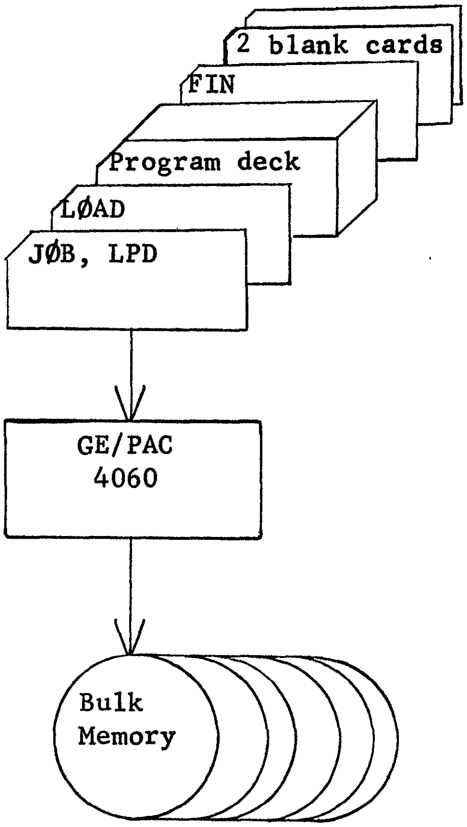
```
JØB, LPD, JØNES D. 9/27/65
LØAD, CURVE FIT, PERMANENT
          :
          (Binary Deck)
          :
FIN
```

This program (CURVE FIT) may be initiated by a JØB Card.

Example 4b

Load a finished program into the real-time library. The program will remain in bulk storage and can be initiated by the Real-Time Executive Program. The program will not allow itself to be interrupted by automatic priority interrupt, nor will it allow its core area to be used by another program during its execution. The program cannot be initiated by a JØB Card.

```
JØB, LPD, J.SMITH 10/4
LØAD, 41, PERMANENT
REAL TIME STATUS,41, UNAVAILABLE, INHIBITED, NØ JØB
          :
          :
          (Binary Deck)
          :
          :
FIN
```

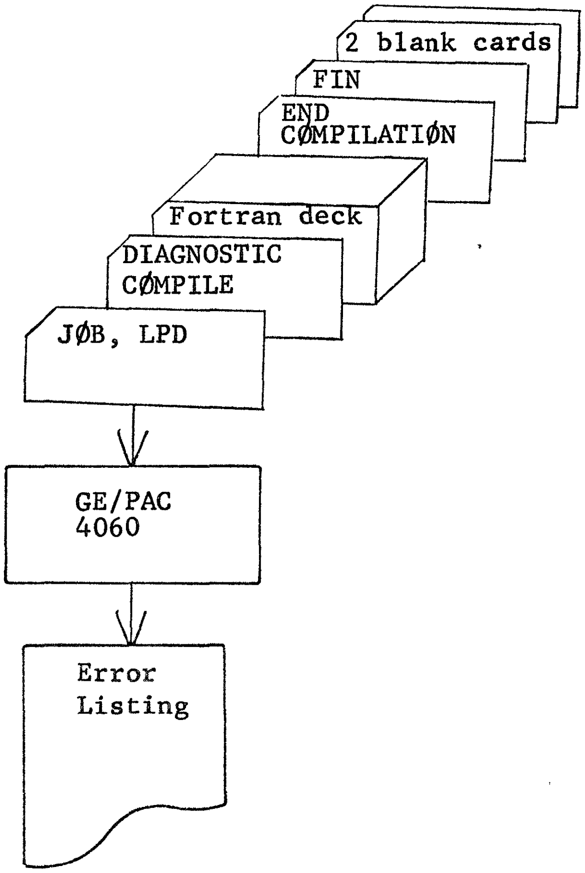


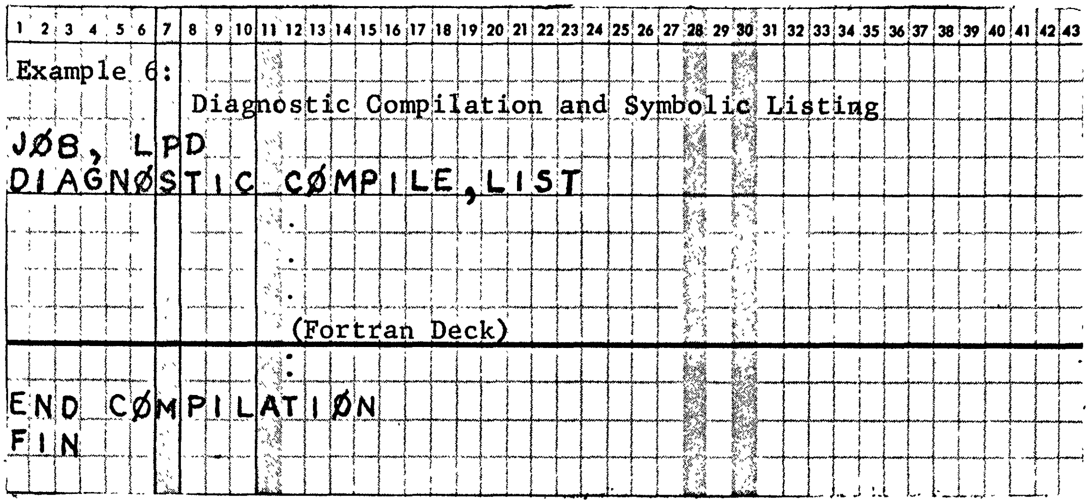
A new symbol table is loaded if the program name is SYMBOL TABLE.

DIAGNOSTIC COMPILE

The FORTRAN language cards following this card are compiled using the common symbol table. All statements containing errors are listed with the proper error indicators. Generated programs are not saved. The LIST option produces a printout of all input statements. The common symbol table is not revised to include the common symbols from this program. The FØRTRAN deck must be followed by an END COMPILATION Card.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
Example 5:
           Diagnostic (Errors only) Compilation
JØB, LPD, DEMAND LØG
DIAGNØSTIC  CØMPILE
           .
           .
           .
           .
            (Fortran Deck)
           .
           .
END CØMPILATIØN
FIN
```





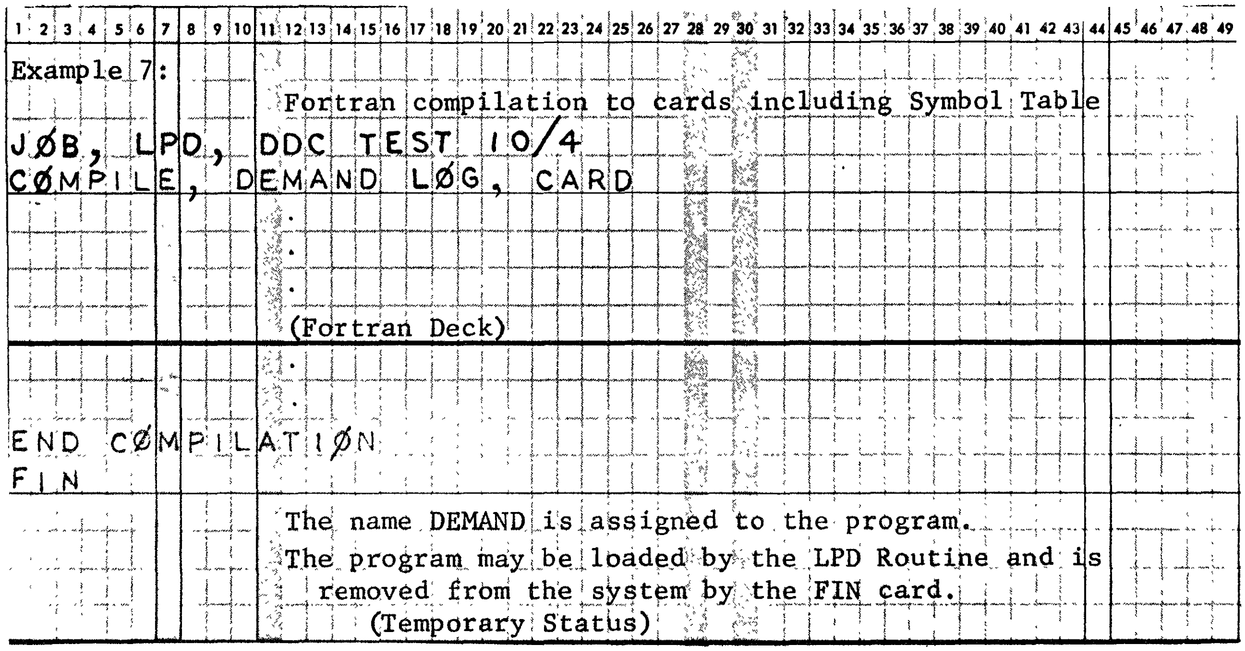
```
Example 6:
          Diagnostic Compilation and Symbolic Listing
JØB, LPD
DIAGNØSTIC CØMPILE,LIST
              .
              .
              .
             (Fortran Deck)
              :
END CØMPILATIØN
FIN
```

COMPILE

The FORTRAN language cards following this card are compiled using the common symbol table. The generated program is placed in bulk memory and is identified by the program number (three digits), or the first six characters of the program name. Compilations may be made in either the temporary, KEEP, FORTRAN LIBRARY, or PERMANENT status. If the CØMPILE card specifies a PERMANENT Real-Time Program, it must be followed by a REAL-TIME STATUS CARD.

If no status is specified, temporary status is assigned and the program is removed when the FIN card is encountered. If more than one status is specified, the program is assigned KEEP Status. If no program name is given, the name TEMP01 is assigned and the program is given temporary status. An attempt to compile subsequent programs with no name, prior to a FIN Card, results in compiling and listing these programs, but none are saved.

All FORTRAN decks must be followed by an END COMPILATION card.



```
Example 7:
           Fortran compilation to cards including Symbol Table
JØB, LPD, DDC TEST 10/4
CØMPILE, DEMAND LØG, CARD
           .
           .
           .
          (Fortran Deck)
           .
           .
END CØMPILATIØN
FIN
           The name DEMAND is assigned to the program.
           The program may be loaded by the LPD Routine and is
             removed from the system by the FIN card.
                (Temporary Status)
```

*Revised

The CARD option produces a program deck and a symbol table including only symbols which have been used. This deck consists of:

1. The binary program deck (including all subroutines which have been obtained from the FORTRAN subroutine library),

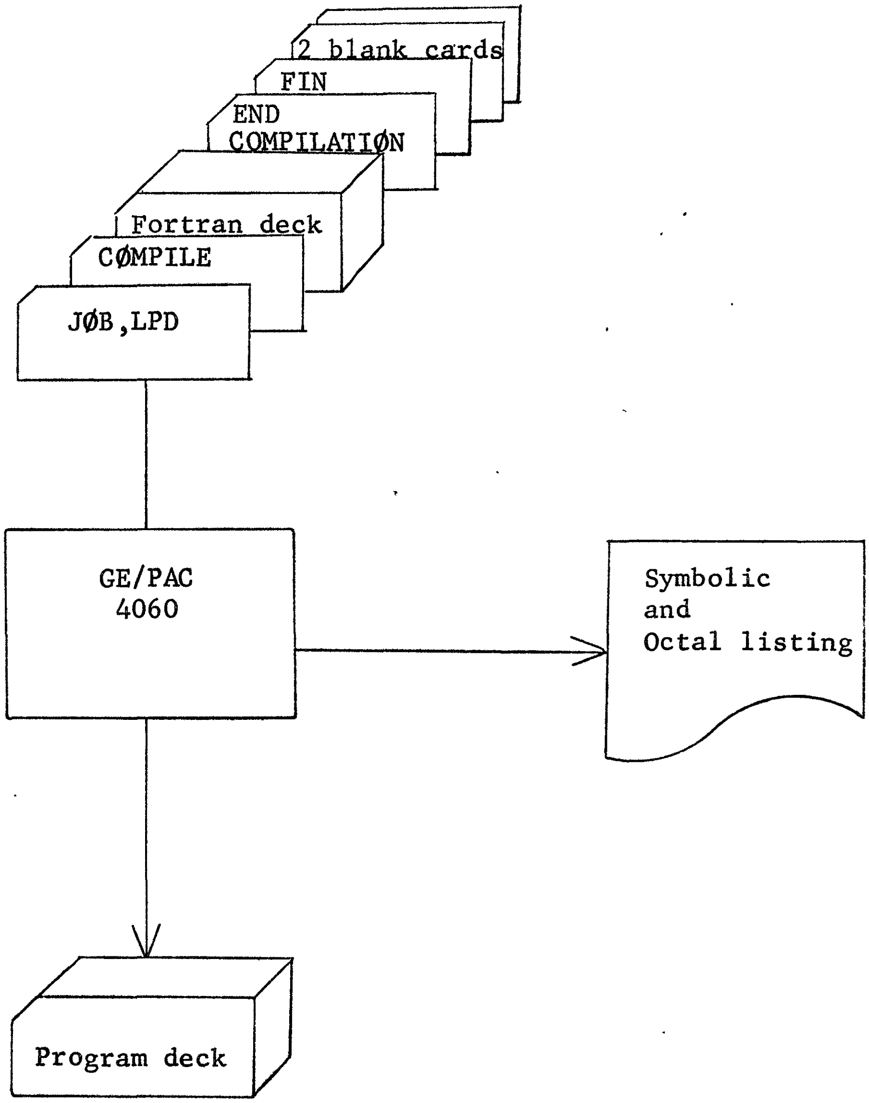
2. a separator card,

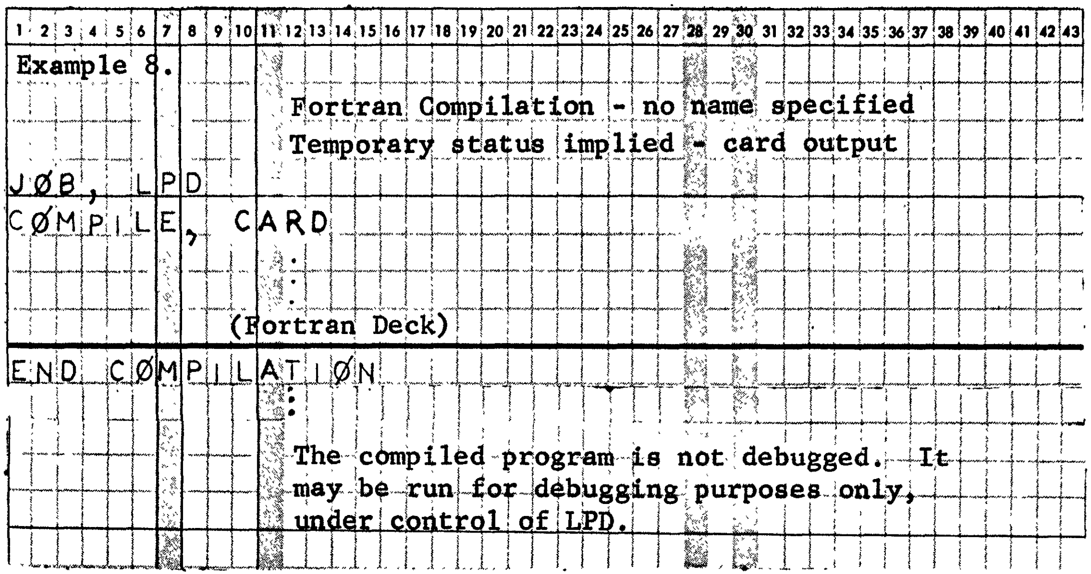
3. the symbol table used by this program,

4. separator card.

Each card in the deck will contain the six-character or three-digit name and a sequence number in the last ten columns. This deck may be loaded by using a LOAD card.

COMPILATION





Example 8.

Fortran Compilation - no name specified
Temporary status implied - card output

```
JØB, LPD
CØMPILE, CARD
          :
          :
        (Fortran Deck)
END CØMPILATIØN
          :
```

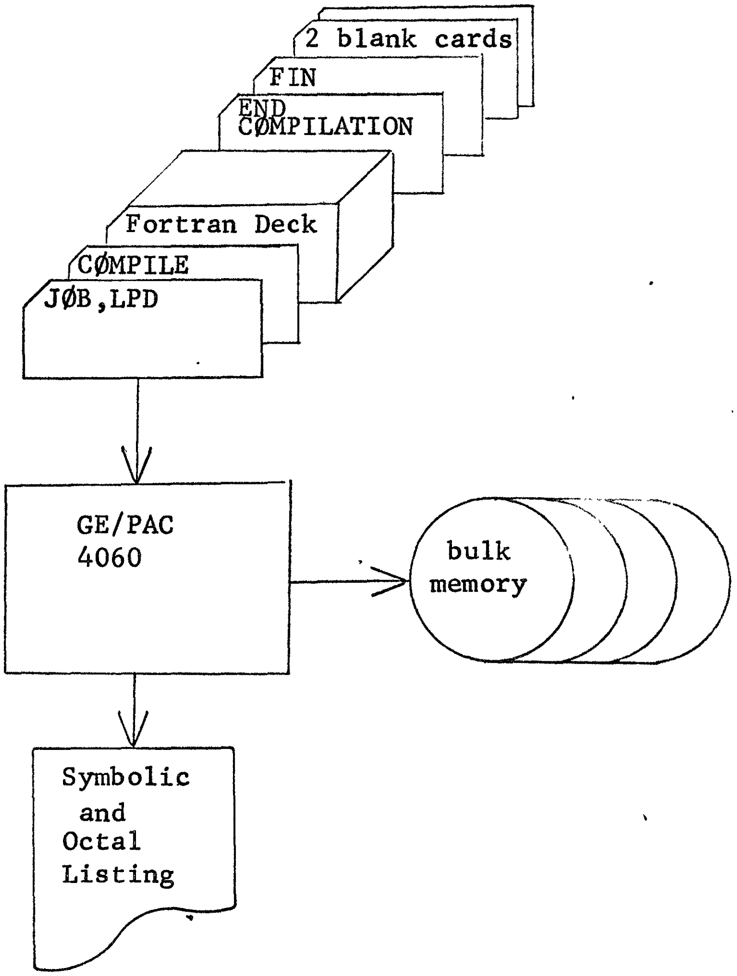
The compiled program is not debugged. It may be run for debugging purposes only, under control of LPD.

Example 9:

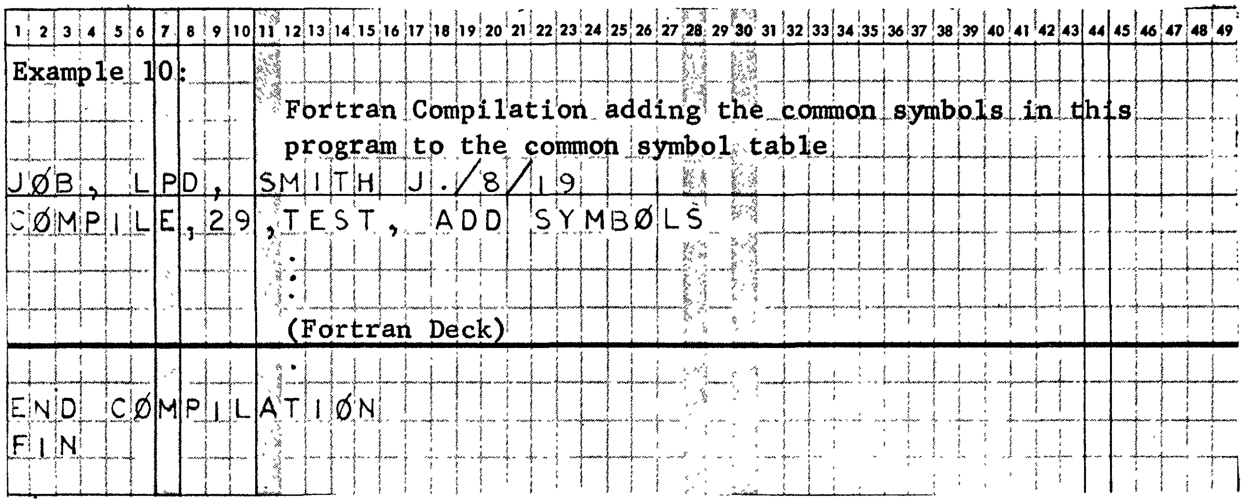
Fortran Compilation - program to remain in bulk storage

```
JØB, LPD, 7/13/65 RUN
CØMPILE, KEEP, DEMAND LØG
          .
          .
          .
          (Fortran Deck)
          .
END CØMPILATIØN
FIN
```

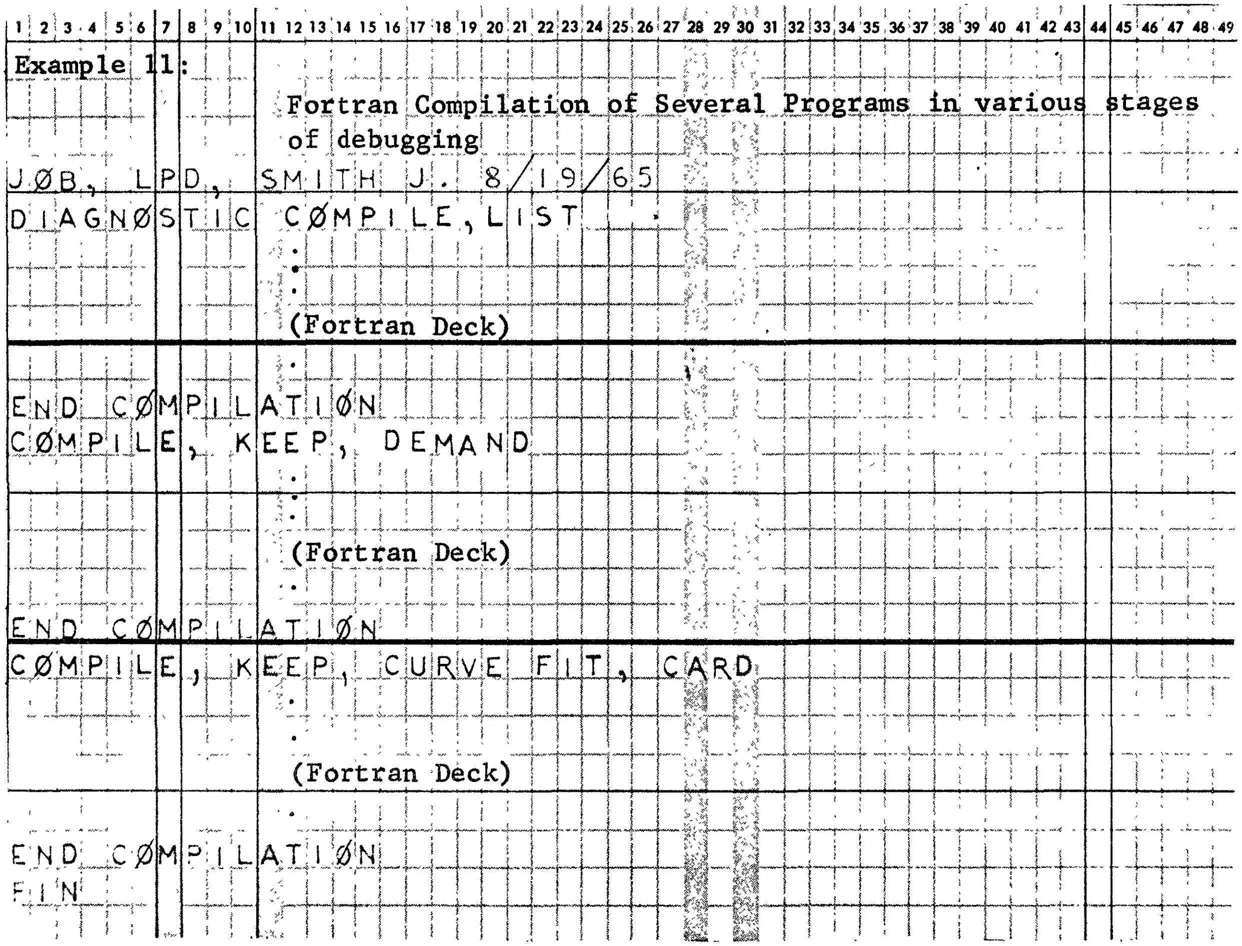
The program is not debugged and may only be called by LPD. It is not removed from the System by the FIN card.



The ADD SYMBOLS option of the COMPILE Card adds the common symbols from the program being compiled to the common symbol table. When the same common symbol name is defined in the common symbol table before compilation and in the program being compiled, only the value from the program being compiled is saved.



```
Example 10:
          Fortran Compilation adding the common symbols in this
          program to the common symbol table
JØB, LPD, SMITH J./8/19
CØMPILE,29,TEST, ADD SYMBØLS
          .
          .
          .
          (Fortran Deck)
          .
END CØMPILATIØN
FIN
```



```
Example 11:
          Fortran Compilation of Several Programs in various stages
          of debugging
JØB, LPD, SMITH J. 8/19/65
DIAGNØSTIC CØMPILE,LIST
          .
          .
          .
          (Fortran Deck)
          .
END CØMPILATIØN
CØMPILE, KEEP, DEMAND
          .
          .
          (Fortran Deck)
          .
END CØMPILATIØN
CØMPILE, KEEP, CURVE FIT, CARD
          .
          .
          (Fortran Deck)
          .
END CØMPILATIØN
FIN
```

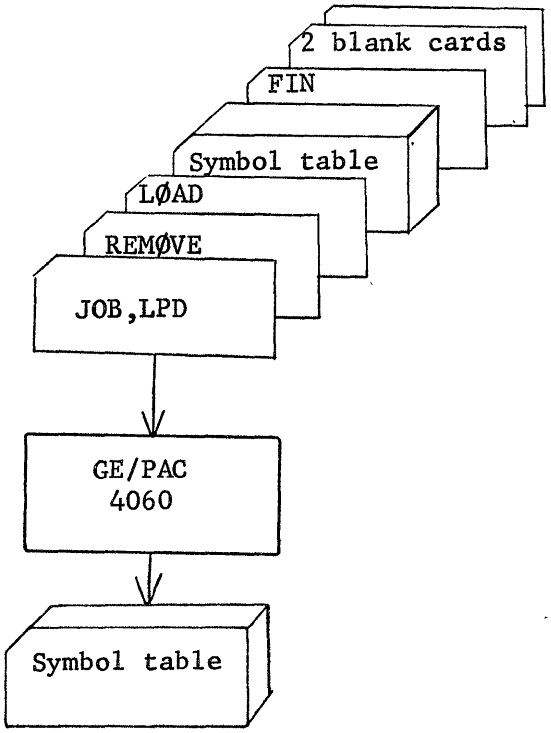
REMOVE

Using the REMOVE card, programs may be removed from any library (real-time, free-time, or FORTRAN Subroutine Library) or the common symbol table may be cleared. Specify the program by either the three-digit number or the six-character alpha-numeric name and the status(PERMANENT or KEEP).

When a REMOVE, SYMBOL TABLE is requested, the symbol table is dumped on binary cards.

Before an existing program or the symbol table is replaced, it must be removed from the system by a REMOVE card.

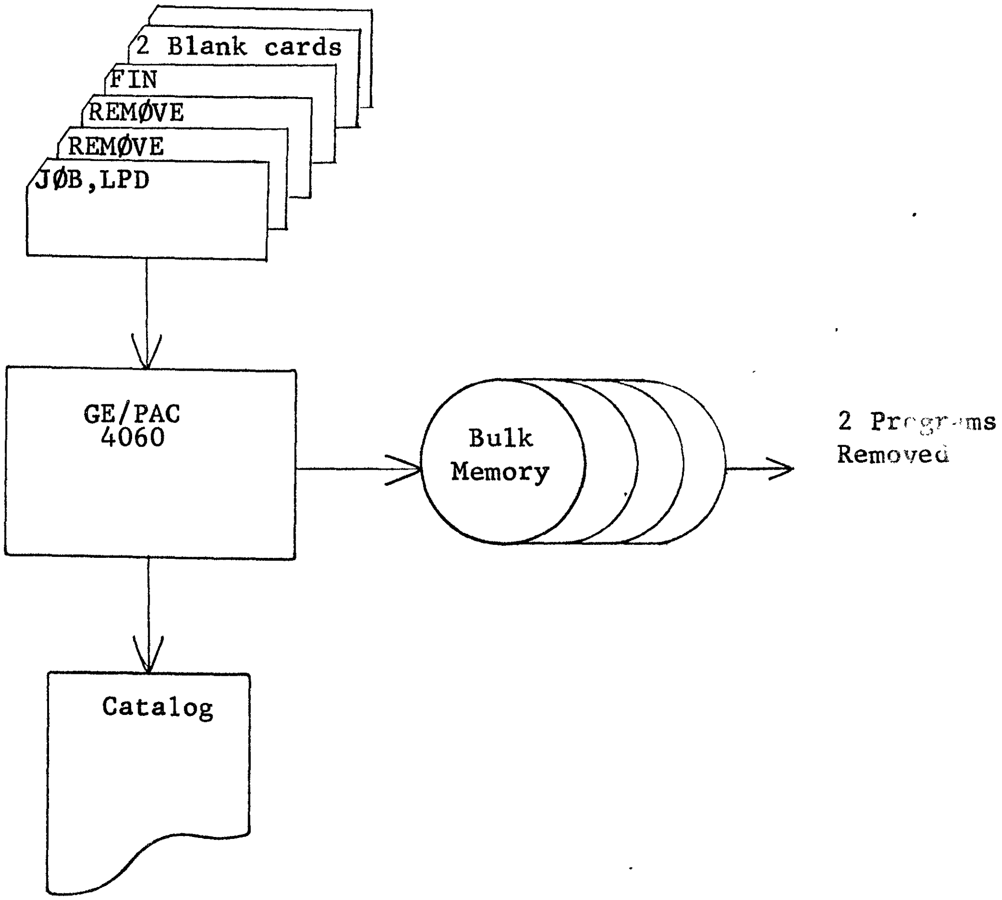
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Example 12:
          Load a symbol table in place of the current common symbol table.
JØB,LPD
REMØVE,SYMBØL TABLE
LØAD,SYMBØL TABLE
          .
          .
          (Symbol Table)
          .
          .
FIN
```



Example 13:

Remove Free-Time Program "Least Squares" and real-time program No. 41 from the system

```
JØB,LPD
REMØVE,LEAST SQUARES,KEEP
REMØVE, 41,PERMANENT
FIN, CATALØG
```



TEST

The program named on the TEST card, is tested as follows:

* 1. The program is called from bulk memory.

2. The operating conditions, as specified in the system protection cards, are set.

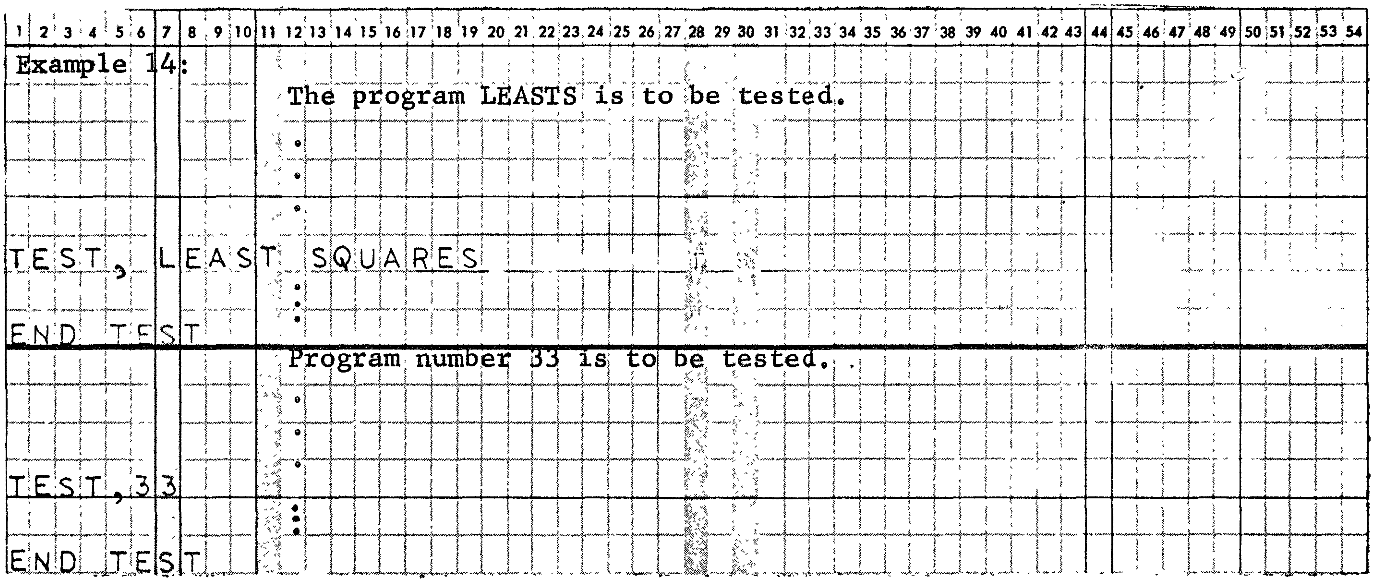
3. The program is modified according to the CORRECT cards.

4. Traps are placed in the program to initiate snapshot dumps.

5. The program is executed.

All cards specifying conditions for the first program segment must follow the TEST card, but precede the first EXECUTE card. Similarly, all cards for the second segment must follow the first EXECUTE Card and precede the second EXECUTE Card, etc. The program name may be either symbolic or numeric. If no program name is given, the program name TEMP01 (temporary status) is assumed.

The test deck must be followed by an END TEST card.



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Example 14:
          The program LEASTS is to be tested.
          .
          .
          .
TEST, LEAST SQUARES
          :
END TEST
          Program number 33 is to be tested.
          .
          .
          .
TEST,33
          :
END TEST
```

* A 4K core area (4096 locations) is available for testing a program. An attempt to test a larger program results in loading and testing the first 4K of the program.

* The program symbol table allows a maximum of 600 symbols. All symbols exceeding 600 are dropped after compilation, and are not available for symbolic references during a TEST.

*Revised

Test Operating Condition Examples

EXECUTE

* The EXECUTE Card initiates the running of a program segment. It gives the first and last locations to be executed during the debugging of this segment. When an EXECUTE Card is encountered, the program is executed starting at the FROM location and continuing until:

1. an invalid Branch Instruction is encountered,
2. the TØ location is executed,
3. the maximum time limit specified on the EXECUTE card is exceeded, or
4. a MONITOR Turn Program Off request is encountered.

The maximum allowable running time for each segment of the test is specified on the EXECUTE Card. If no time is specified, the maximum time is set to one minute.

If the FROM Statement is not given, execution starts at the first location of the program. When the TØ statement is not given, the segment is terminated by either the maximum real-time being exceeded, a program turn off, or an invalid branch instruction.

Symbolic or numeric locations (relative to the beginning of the program) may be used in the EXECUTE Card.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54

Example 16a:

Execute a program from its origin to statement $20. The maximum real time allowed for the test is one minute.

```
        :
EXECUTE, TØ $20
```

Example 16b:

Execute a program from the 10th location after the beginning of the program to symbolic location GØBACK. The maximum real time allowed for the test is 3 minutes.

```
        :
EXECUTE, FRØM 10, TØ GØBACK, 3 MINUTES
```

Example 16c:

Execute a program from its origin until it turns itself off (See Monitor Turn Program Off request), but for not longer than two minutes. The program will remain in bulk memory after the run.

```
JØB, LPD, 9/26/65 LEWIS W.
LØAD, KEEP, ØPTIMUM CØNTRØL
        :
        (Binary Deck)
        :
TEST, ØPTIMUM CØNTRØL
EXECUTE, 2 MINUTES
END TEST
FIN
```

*Revised

Data cards which are read by the program being tested must follow the EXECUTE card for the segment(s) containing the READ statements.

CORRECT

The CORRECT card permits octal corrections (only) to be made to a segment while it is being tested. It also permits insertion of data
* in a symbolic common location. Numeric addresses are assumed to be relative to the beginning of the program. Corrections are sequential, starting with the location following "AT". Corrections are right-justified if less than eight digits in length and two consecutive commas are interpreted as "skip one location."

In card example 17a given below, there is one eight digit correction, one correction with four leading zeros and four digits, followed by a skip and a word of zeros. The last comma could be removed without affecting the correction.

A maximum of 27 corrections and "skips" may be entered on each card. Corrections are retained only until the END TEST Card is encountered.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
```

Example 17a: Explained above.

```
CORRECT, AT ARRAY, 24660000, 4371,,0,
```

Example 17b: Change the 44th (octal) location from the beginning of the program to the instruction "branch to here plus 3 locations" (BRU * + 3)

```
CORRECT, AT /44, 14040003
```

* Since a program includes only instruction and data areas as compiled, the programmer must provide any additional area needed during debugging unless the program is less than 4K.

*Revised

DUMP

The DUMP Card displays the contents of the specified areas or location on the printer before the designated instruction is executed. The format of the DUMP is either floating or fixed point decimal (integer) depending on whether the FORTRAN definition of the beginning label is floating or fixed. If the definition is neither floating or fixed (example: $TEMP), the format is octal. The format inferred by the label may be overridden by using a stated option. Permissible options are INTEGER, REAL, and OCTAL. This allows the programmer to follow the data formats as declared at compilation time.

All DUMP requests are removed at the end of each program segment. Therefore, all dumps desired between EXECUTE Cards must be specified. When more than twenty DUMP Cards are entered between EXECUTE Cards, only the first twenty are used.

Example 18a:

Display the contents of the 64-word Table, DIGITS, in floating point format when the program reaches statement 20.

```
:
DUMP, AT $20, FRØM DIGITS, 64 WØRDS
:
```

Example 18b:

Display the contents of the locations from MARGIN to VAR inclusive when the program reaches the 163rd (octal) location from the beginning. The dump is in fixed-point format.

```
:
DUMP, FRØM MARGIN, TØ VAR, AT /163
```

Example 18c:

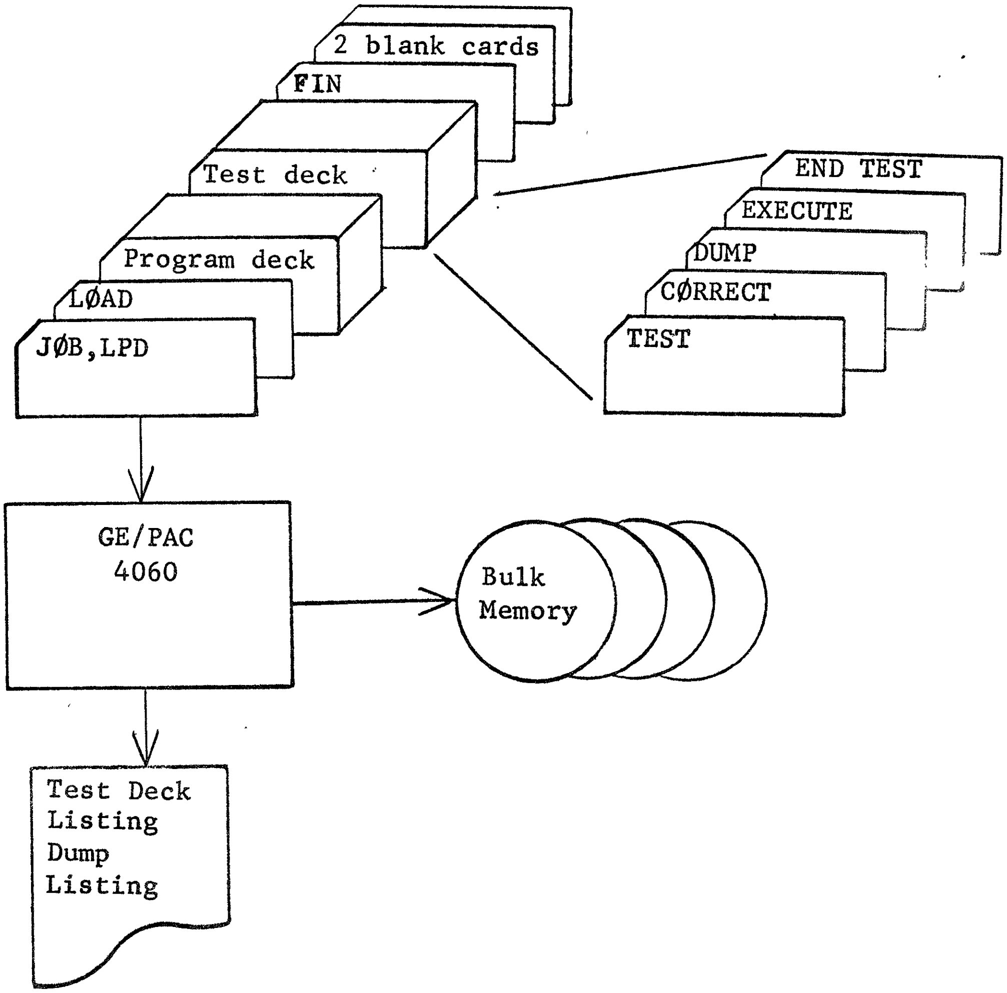
When the program reaches the 10th (decimal) location from the beginning, display the octal contents at the temporary storage location ITEM

```
:
DUMP, ITEM ,AT 10, ØCTAL
:
```

Example 19:

Execute a program from its origin to statement 20. The test must not run longer than 5 minutes. Test data is entered into "ATABLE" and the program's temporary storage is displayed in floating point format on the printer when the program reaches the 5th location after statement 33.

```
JØB,LPD,K.JØRDAN CØMPILATIØN NØ.5 6/28/66
LØAD, ØPTIMUM CØNTRØL, KEEP
        :
        (Binary Deck)
        :
TEST,ØPTIMUM CØNTRØL
CØRRECT,AT ATABLE,25320000,24240000,0,22610000
DUMP,AT $33+5, FRØM $TEMP, 24 WORDS,REAL
EXECUTE,TØ $20, 5 MINUTES
END TEST
FIN
```



"Compile and Go"

Example 20:

* Compile the FORTRAN program LEAST SQUARES and punch a program deck. Enter a change at the symbolic location SUMSQ. Execute the program from statement 3, to 4 locations after statement 10. When the program reaches the 4th location after statement 10, display on the printer the contents of symbolic location SIGMA in floating point (SIGMA is a FORTRAN floating point symbol). Also display the contents of the 150 word table starting at SUMA.

Enter a new value in SUMSQ and re-run the program, this time from statement 20 to statement 10. When the program reaches statement 30, display the contents of the locations from TABLE to BTABLE (inclusive) in floating point format.

Upon completion, remove the program "LEAST SQUARES" from the System

```
JØB,LPD, ØPTIMUM CØNTRØL TEST 3/7/66
CØMPILE, LEAST SQUARES,CARD
          :
        (Fortran Deck)
          :
END CØMPILATIØN
TEST, LEAST SQUARES
CØRRECT, AT SUMSQ, 22620000
DUMP, AT $10+4, FRØM SUMA, 150 WØRDS, INTEGER
DUMP, AT $10+4, SIGMA
EXECUTE, FRØM $3, TØ $10+4, 7 MINUTES
CØRRECT, AT SUMSQ,22734000
DUMP, AT $30, FRØM TABLE, TØ BTABLE
EXECUTE, FRØM $20, TØ $10, 3 MINUTES
END TEST
FIN
```

"Test where program is in KEEP Status (in bulk memory)"

Example 21:

A program RØØTFINDER is in bulk memory in KEEP status. Execute the program from statement 1 to the end. When the program reaches statement 45, display the contents of the 48-word floating point table, RØØTS. Change the contents of the 135th location from the beginning to a store instruction and repeat the test. The program will remain in bulk storage after the test.

```
JØB,LPD, TEST FØR CØMPLEX RØØTS 5/18
TEST, RØØTFINDER
DUMP, AT $45, FRØM RØØTS, 48 WØRDS
EXECUTE, FRØM $1
CØRRECT, AT /135,32300145
DUMP, AT $45, FRØM RØØTS, 48 WØRDS
EXECUTE, FRØM $1
END TEST
FIN
```

*Revised

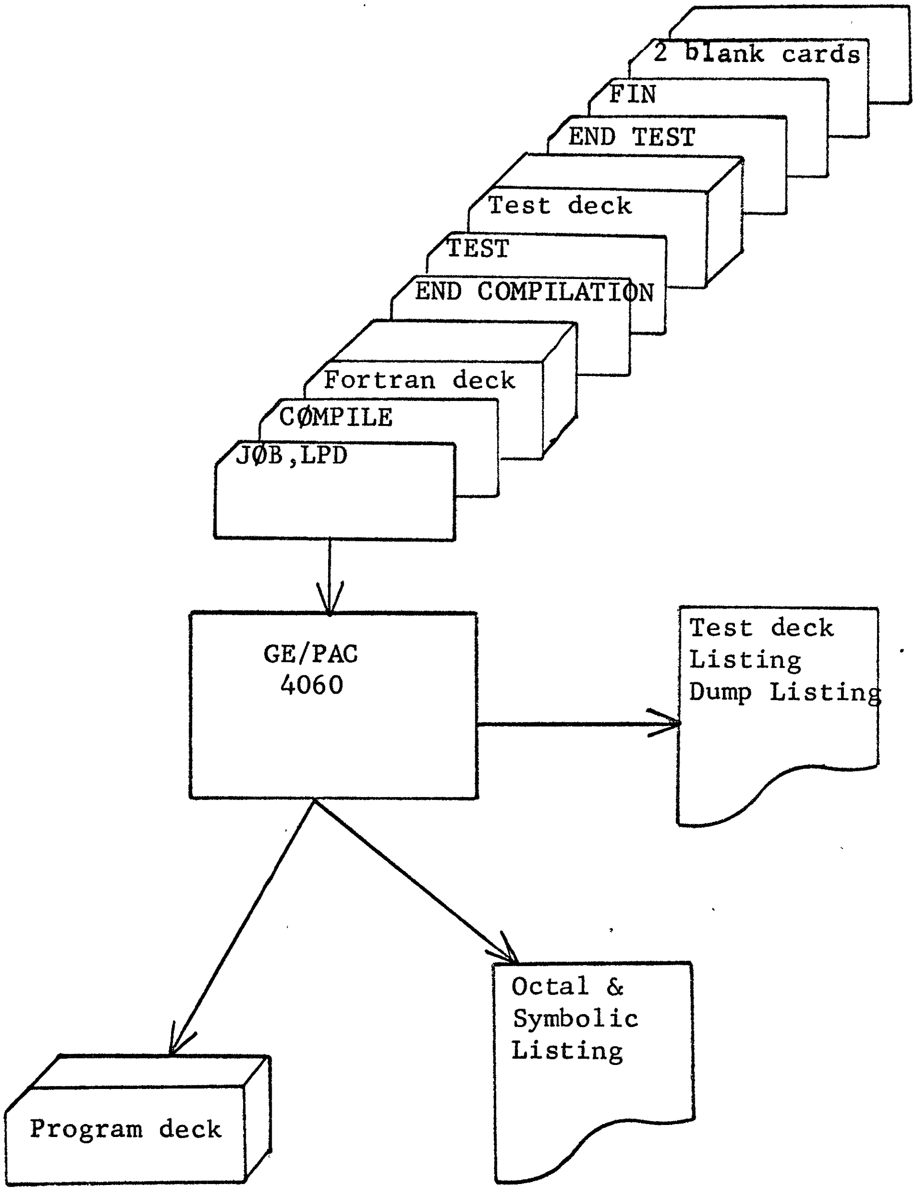


Figure for Example 20.

System Protection Examples

If no system protection cards are included in the deck, the program
* will only be allowed to store in its own area or in COMMON, and branch within its own area.

BULK STORE

The BULK STORE allows the program being debugged to transfer data from core to the designated area in bulk memory. Symbolic labels must be absolute bulk storage locations. Numeric addresses must be entered as octal. Bulk store parameters are effective until either a new BULK STØRE or the END TEST Card is encountered.

```
 1 2 3 4 5 6 7 8 9 10 11 ...
Example 22:
          The program being executed may transfer to bulk memory only
          between the specified locations. Symbolic locations are absolute
          labels
          :
BULK STØRE, FRØM ATABLE, TØ BTABLE
          :
          :
BULK STØRE, FRØM /167700, TØ /172300
          :
```

* Input/Output During Testing

These cards are used to specify which I/O devices may be used by the program being tested. They are effective until the END TEST card is encountered. Device numbers (i) are FORTRAN device numbers. If not stated, device 0 is assumed. All devices of one type must be specified on one card.

INPUT DEVICES

This card is effective until a new INPUT DEVICES card is encountered.

1. READi specifies a card reader, paper tape reader, or input typer.
2. LINKi specifies data link input.

OUTPUT DEVICES

This card is effective until a new OUTPUT DEVICES card is encountered.

1. PRINTi specifies an output typer or printer.
2. PUNCHi specifies a card or paper tape punch.
3. LINKi specifies data link output.

*Revised

Example 23:

The program being executed may use only the specified input and output devices, i.e. card reader, printer, and card punch.

:

```
INPUT   DEVICES,READ1,LINK1
ØUTPUT  DEVICES,PRINT1 ,PUNCH2
```

PROGRAM

The PROGRAM Card tells which real-time programs may be turned on by the program being tested. Programs to be turned on are specified by their program number.

Programs which are considered a part of MONITOR (e.g. Input, Output) may not be turned on. (See Appendix A.) Program card parameters are effective until either a new PROGRAM Card or the END TEST Card is encountered.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
```

:

Example 24:

The program being executed may initiate the execution of only the specified real-time programs (Nos. 23 and 24)

:

```
PROGRAM,  23,24
```

:

Violation of any of the above system protection restrictions by program is noted by an error printout (See 3.2.4 - Card Errors). The card in error is ignored.

Example 25:

The program, 25, is to be debugged with the following restrictions.

1. It may request transfers to bulk storage only between octal addresses 463000 and 471200 inclusive.
2. It may only make input requests through the card reader.
3. It may only make output requests of the printer.
4. It may initiate the execution of real-time programs 4, 5 and 15 only.

Branch instructions are inserted at the 6th and 7th locations after statement 10.

When the program reaches statement 56, the contents of 130 locations will be displayed on the printer in fixed point (integer) format.

The program will be executed from its beginning to statement 60, with a maximum time limit of 5 minutes.

```
JØB, LPD, FRØST J. 8/15 TEST RUN
CØMPILE, 25
          :
          (Fortran Deck)
          :
END CØMPILATIØN
TEST, 25
BULK STØRE, FRØM /463000, TØ /471200
INPUT DEVICES, READ1
ØUTPUT DEVICES, PRINT1
PRØGRAM, 4, 5, 15
CØRRECT, AT $10+6,34040016, 14077773
DUMP,FRØM ATABLE, 130 WØRDS,INTEGER, AT $56
EXECUTE, TØ $60, 5 MINUTES
END TEST
FIN
```

4. OPERATING INSTRUCTIONS

4.1 Card Handling

Review all card decks to insure that they are in correct order. For instance, a COMPILE request should have:

1. JØB Card
2. CØMPILE Card
3. FØRTRAN Deck or Decks
4. END CØMPILATIØN Card
5. FIN Card

A LØAD request should have:

1. JØB Card
2. LØAD Card
3. Binary Deck
4. Separator Card

If there is a symbol table to be loaded, there must also be a symbol table deck and another separator card.

If the program is not being loaded in the KEEP or temporary status, there must be no symbol deck. A program in the KEEP or temporary status with no symbol deck must have an additional separator card added at the end of the deck. Note: A blank card may be substituted for a separator card.

A TEST deck should have:

1. JØB Card
2. TEST Card
3. CØRRECT Cards (Optional)
4. DUMP Cards (Optional)
5. System Protection Cards (Optional)
6. EXECUTE Cards
7. END TEST Card
8. FIN

Place the card deck in the input hopper and press the demand button on the card reader.

4.2 Operating Requirements

Refer to the operating instructions for the card reader, printer, and card punch.

A run may be aborted at any time that the LPD System is not reading cards by pressing the Input Demand Button on the card reader. This action simulates reading of a FIN Card and the JOB is terminated.

* If LPD is reading cards, turn the card reader off. This simulates peripheral failure, and action should be taken according to paragraph 4.3.

* Other free-time programs can be aborted in the same manner by checking the input demand flag, or by peripheral failure on an input request.

4.3 Peripheral Failures

When there is a malfunction on any of the peripheral equipment, the system types the appropriate error message:

```
LINK2  FAILURE
READ1  FAILURE
PUNCH1 FAILURE
```

The operator notifies the system if the device will be repaired. If a repair is not possible, the Free-Time System turns itself off and the JØB must be restarted after the repair has been made.

*Revised

APPENDIX A

RESTRICTIONS

* 1. Program or symbol names should not begin with the following identifiers:

KEEP
PERMAN (ENT)
ADDSYM (BØLS)
FØRTRA(N)
TEMP01
SYMBØL
CARD
AT
TØ
FROM
REAL
INTEGER
ØCTAL

* 2. The first six letters of any two program or symbol names should not be the same.

3. No temporary locations (BSS) are permitted within the program, for instance, those produced by PAL coding. All temporary locations (BSS) are placed at the end of the program by FORTRAN.

4. Information placed after column 69 on the control card is ignored.

5. The following Monitor functional programs may not be called using the JØB card or by a Turn Program On request:

a. Output Program
b. Input Program
c. Corrective Action Diagnostic
d. Free-Time Executive Program
e. Operator Routines.

*Revised

LIST OF CONTROL CARDS CONTAINING ALL OPTIONS

```
BULK STØRE, FRØM /135260, TØ DRUMAD
CØMPILE, CURVE FIT,KEEP,CARD, ADD SYMBØLS
CØMPILE, SQUARE RØØT, FØRTRAN LIBRARY
CØMPILE,23, PERMANENT
CØRRECT,AT DØG+5,2451,35030011,0,22,,44020106
CØRRECT,14040003 ,AT /64
DIAGNØSTIC COMPILE,LIST
DUMP, ITEM, AT 10, ØCTAL
DUMP,AT $16, FRØM /55,TØ $TEMP,ØCTAL
DUMP,FRØM DØG,TØ CAT, AT 25,INTEGER
DUMP,TØ $30,FRØM /162,AT $10,REAL
END COMPILATION
END TEST
EXECUTE
EXECUTE, FRØM START,TØ /240, 10 MINUTES
EXECUTE,TØ /531, FRØM 10
EXECUTE, TØ $12, 5 MINUTES
EXECUTE, FRØM $20
```

*Revised

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
FIN

FIN, CATALØG

INPUT DEVICES,READ ,LINK1

JØB,LPD, LIST OF CØNTRØL CARDS WITH ØPTIØNS

JØB, 15

LØAD,LEAST SQUARES,PERMANENT

LØAD,27,KEEP

ØUTPUT DEVICES, PRINT1 ,PUNCH1

PRØGRAM, 2,31,3

REAL TIME STATUS, 12, INHIBITED, UNAVAILABLE, NØ JØB

REMØVE,41, PERMANENT

REMØVE, MILL SETUP, KEEP

TEST,TRAJECTØRY

TEST,14
```

APPENDIX C

ERROR MESSAGES

A. System Protection Errors

Messages are printed when an error is located within a program which violates the system protection rules as defined by the test system correction cards.

When an invalid instruction is found, an error message, "ILLEGAL XXXXX, relative address, instruction, contents of the Index Register or Device or 0", is printed. XXXXX may be a STORE INST, BRANCH INST, I/O INST, MTR ENTRY, BULK TRANS, ØØMED INST, RPTED INST, or INPUT REQST.

Examples:

```
1. ILLEGAL STORE INST. 00076 32340174 00000012

2. ILLEGAL I/O INST.   00043 25430000 00002400

3. ILLEGAL BRANCH INST. 00042 14040163 00000000
```

B. Control Card Errors

Messages are printed when control cards do not contain valid information, or when the deck following the control card does not match the control card.

Checking begins with the first JØB Card and all cards following are tested for correct format. An error is indicated by the appropriate message following the listing of the card in error. No processing is accomplished on this card. Subsequent cards are processed in the normal manner.

1. No program name or number on the JØB Card.

```
     JØB
     ILLEGAL PROGRAM REQUEST
```

2. Program name starting with a non-alphabetic character (free time program).

```
     JØB, 6DUMP
     ILLEGAL PROGRAM REQUEST
```

3. Program name not in the library (free time or real time).

```
     JØB, SZYGXK
     NOT IN LIBRARY
         or
     JØB, 125369
     NOT IN LIBRARY
```

4. The second field on the FIN Card is not CATALOG.

```
FIN, CXTALOG
ILLEGAL FIELD
```

5. The second field on the DIAGNOSTIC COMPILE Card is not LIST.

```
DIAGNOSTIC COMPILE, JUST
ILLEGAL FIELD
```

6. Program name is the same as another field on the card. (See Appendix A).

```
COMPILE, PERMANENT, CARD, PERMANENT
ILLEGAL FIELD
```

7. Unidentifiable or excessive fields on any card

```
REMOVE, CARDEX, PERMANENT, LIST
ILLEGAL FIELD
```

8. A non-FORTRAN deck following a COMPILE card.

```
SYNTAX ERROR on every card
```

9. A non-binary deck following a LOAD card.

```
FORMAT ERROR YPK0300335
```

Columns 70-80 of the first card in error will be printed.

10. Missing separator cards.

```
FORMAT ERROR
```

11. Slashes, numbers greater than seven, or alphabetics in the CORRECTION field of the CORRECT Card.

```
CORRECT, AT ALPHA, 0493602K8
ILLEGAL FIELD

CORRECT, AT BETA, /04567
ILLEGAL FIELD
```

12. The given symbol does not exist in the Symbol Table.

```
DUMP, AT XGMPX, FROM A, TO B
ILLEGAL FIELD
```

13. A missing field or a blank field where no blank field is permitted (no status was specified).

```
REMOVE, 41
ILLEGAL FIELD
```

14. Program number of compiled or loaded program too big.

```
LOAD, 99362, PERMANENT
ILLEGAL PROGRAM NAME
```

15. Attempting to add a program which already exists without using a remove card.
    LOAD, 15, PERMANENT
    ILLEGAL PROGRAM NAME

16. Attempting to remove a non-existent program
    REMOVE, 92, PERMANENT
    ILLEGAL PROGRAM NAME

17. Attempting to remove an operating real-time program.
    REMOVE, 21, PERMANENT
    PROGRAM ON

18. Overloading bulk memory
    NO TEMPORARY BULK STORAGE AVAILABLE

19. Attempting to compile a FORTRAN program with too many statements.
    NOT ENOUGH TEMPORARY BULK STORAGE FOR COMPILATION

20. Missing FIN Card.
    NO FIN CARD

21. Too many DUMP Cards between EXECUTE Cards or an unidentified card.
    ILLEGAL CARD

C. Programming Errors

Messages are printed following the listing of a compiled and assembled program to call attention to errors made by the programmer. These errors may cause the program to be loaded incorrectly, resulting in an invalid run. XXXXXX in the messages denotes program or subroutine name.

1. Requesting a FORTRAN subroutine which is not in the FORTRAN library. No action taken.

   XXXXXX SUBROUTINE REQUESTED, NOT IN FORTRAN LIBRARY

2. Requests for library subroutines exceed 100. Remainders are noted in error messages and ignored.

   TOO MANY LIB CALLS. REQUEST FOR XXXXXX NOT INCLUDED

3. Attempt to include a function or subroutine with a program to be added to the FORTRAN library. The subprogram is ignored.

   FORTRAN LIBRARY PROGRAM XXXXXX HAS A SUBPROGRAM

4. More than one status given for a program which has a name or number. Program is added to undebugged library.

   TOO MANY STATUS FIELDS SPECIFIED. ASSUME KEEP STATUS

5. A program without a name or number has a status specified. Program is given name TEMP01 and is added to undebugged library.

   STATUS SPECIFIED, NØ NAME GIVEN. ASSUME TEMP STATUS.

6. A program deck was requested, but the card punch failed during the punchout.

   PUNCH BAD OR OUT OF SERVICE. PROGRAM DECK INCOMPLETE

7. Number of labels referenced in program exceeds 600. The remainder are ignored and cannot be referenced symbolically on TEST deck cards.

   TOO MANY LABELS. EXCESS LABELS CANNOT BE REFERENCED

8. The program exceeds the scratch bulk storage area. The program is deleted.

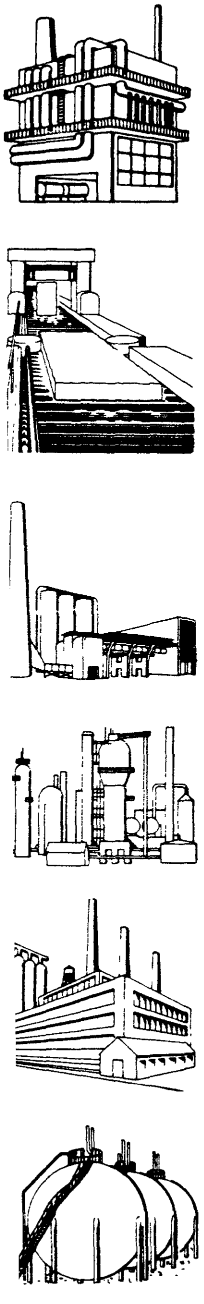
   PROGRAM BEING ASSEMBLED EXCEEDS BULK SCRATCH AREA.

APPENDIX D

ILLUSTRATIONS

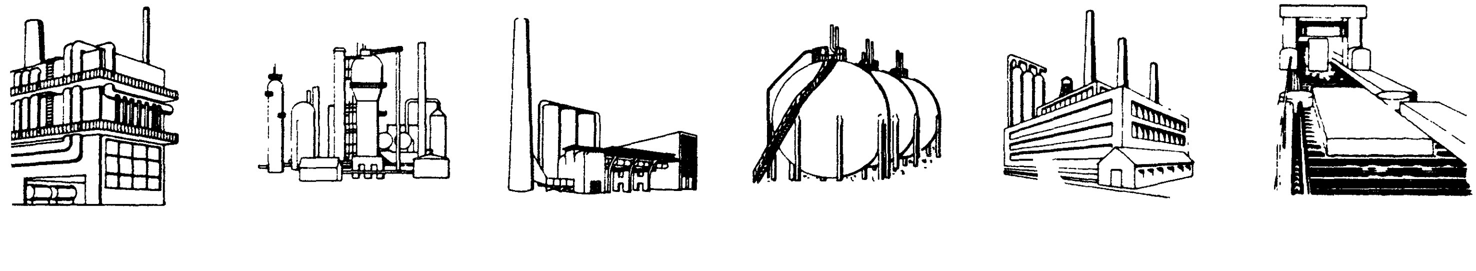
GE-PAC® 4000

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER



FREE-TIME SYSTEM USER'S MANUAL

(FTS02)



PROCESS COMPUTER
BUSINESS SECTION
PHOENIX, ARIZONA

GENERAL ELECTRIC

Library Control No. YPG58M

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P.O. Box 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

CONTENTS

INTRODUCTION

This manual describes the capabilities and usage of the General Free-Time System for the GE/PAC* 4050-4060 computers.

The purpose of the General Free-Time System is to provide the user with the ability to compile, test, and execute functional programs in a real-time environment. Existing service programs may be initiated by a control card. New programs may be tested and entered into the overall system in easy stages, starting with an untested program and arriving at an operating real-time program.

This system provides "load and go" or "compile and go" with testing at the symbolic level, using the names of FORTRAN variables. The dynamic relocation of programs and automatic allocation of storage frees the programmer from any concern other than the successful compilation and testing of his program.

The GE/PAC family of programming manuals consists of separate, complete booklets, each of which deals with a specific subject. The FREE-TIME SYSTEM MANUAL is the sixth in this series of publications and can be identified as such by the use of the numeral six prefix in paragraph headings and subheadings.

Reference is made in the test to the following publications which are available through the Programming Library:

YPG51M - MONITOR USER'S MANUAL,
YPG14M - FORTRAN REFERENCE MANUAL,
YPG12M - PROCESS ASSEMBLER LANGUAGE MANUAL

* Registered Trademark of the General Electric Company.

6.1 PURPOSE

6.1.1 GENERAL PURPOSE

The primary purpose of the GE/PAC Process Control Computer is to react automatically to real-time process conditions. However, there is normally a considerable amount of time, called free-time, available for computations which are independent of the real-time process. The Free-Time System is designed to make use of this time without disturbing the operation of the real-time system.

6.1.2 PURPOSE OF FREE-TIME SYSTEM

An operator using the Free-Time System may request the execution of any free-time program not related to the process and some real-time programs which are related to the process. Execution is requested via a control card describing the desired program. Real-time programs are designated by their program-priority numbers in the Process System. Free-Time programs are designated by the first six characters of their name.

6.1.3 PURPOSE OF THE LANGUAGE PROCESSING AND TEST FUNCTION

The purpose of the Language Processing and Test (LPT) Function within the Free-Time System is to provide the user with the ability to compile, test, and maintain several libraries of programs. This is done during the free-time which is available even while the GE/PAC is on-line. Both real-time and free-time programs may be processed and maintained by LPT. Provisions also exist to maintain the system common symbol table and the FORTRAN subroutine library. All of these functions are requested by control cards.

1. Language Processing - Compiles FORTRAN and PAL language programs and enters them into the free-time, real-time, or TEST library on request.

2. Testing - Tests the compiled programs under real-time conditions.

3. System Protection - Protects the system from the actions of untested programs by using the memory and input/output protection hardware.

6.2 SYSTEM CAPABILITIES

The System has the ability to:

1. Request, through control cards, the execution of real-time programs associated with the process, or free-time programs independent of the process,

2. Compile and/or test programs during on-line operation of the GE/PAC,

3. Add or remove programs from the real-time, free-time, or FORTRAN subroutine libraries,

4. Add or remove tables from the data table library (option),

5. Permit modification to the list of common symbols,

6. Protect the system against disruption by untested programs.

6.3 SYSTEM PROTECTION

System protection prevents an untested program from destroying information outside its own area or from disturbing the rest of the system.

Any untested program is allowed to:

1. Branch to any location within its own area,

2. Store into its own area or free-time permanent core area.

Monitor and Input/Output subroutine requests are trapped and interpreted before they are executed, because untested programs must not have access to such devices as the Multiple Output Distributor, Time Output Contact, Scanner, etc.

6.4 SYSTEM REQUIREMENTS

6.4.1 HARDWARE

Input to the Free-Time System is accomplished by punched cards; output is by optional punched cards or printing. Certain devices are required for a minimum system. These are:

1. GE/PAC 4050,4060 - 16K Core Minimum

2. Bulk Storage Device

3. Card Reader

4. Input/Output Typewriter

For optimum system communication, a second printing device is recommended. This could be an Anelex line printer, a Selectric output typewriter, or a Model B output typewriter. Also, if it is desired to maintain binary card decks for back-up of all programs, a card punch is a required peripheral device. (See COMPILE Control Card, P. 26.)

6.4.2 SOFTWARE

The Monitor used with the Free-Time System contains communication for the peripheral devices chosen.

Programs in all libraries must conform to the following rules:

1. All programs must be assembled with a starting location of zero. (The ORG or DCW pseudo-ops are considered illegal by the assembler.)

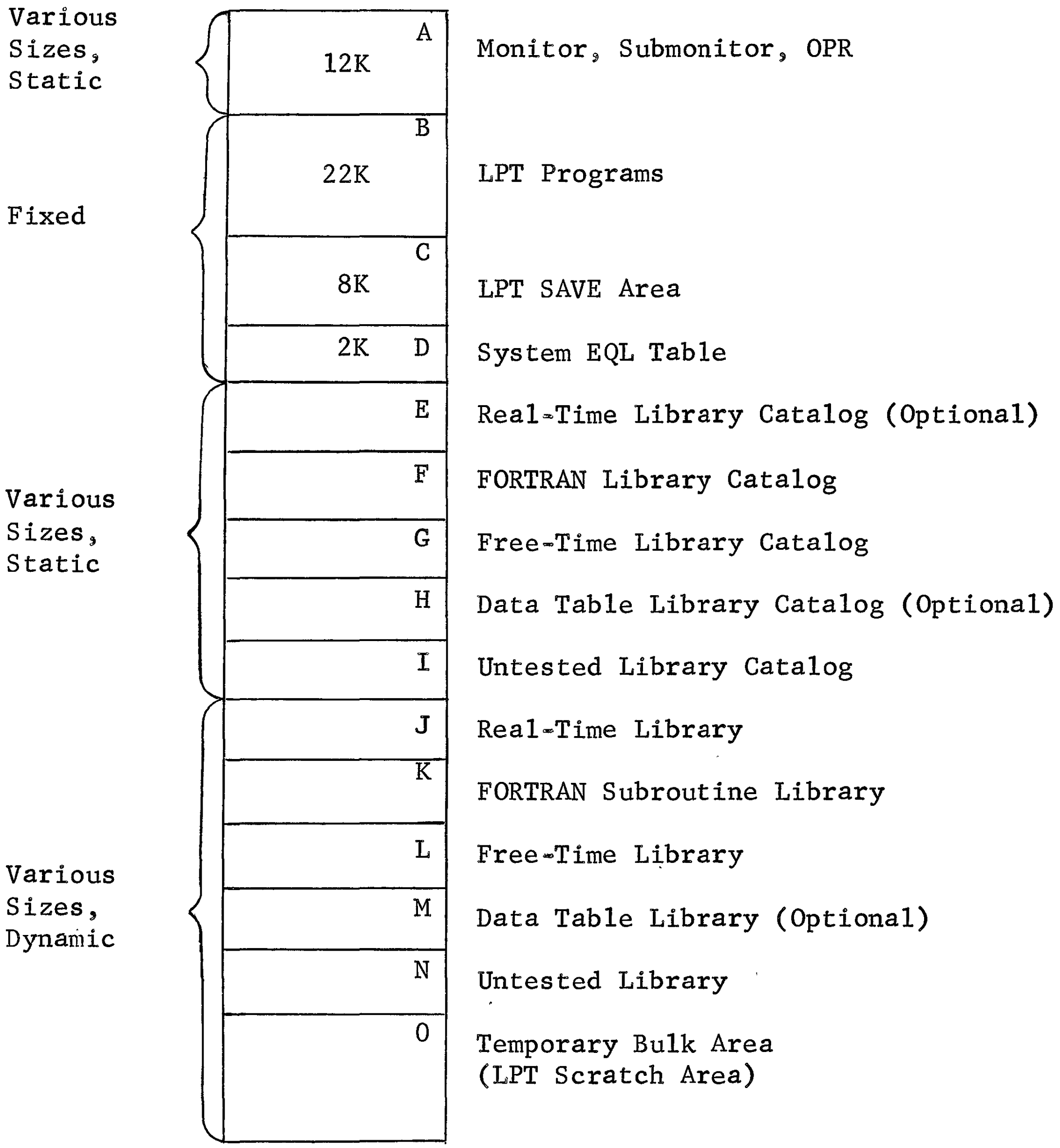
2. All storage areas not relative to the program must be in permanent core or bulk storage;

3. All input/output requests must be made through standard Monitor or FORTRAN communication.

6.5 SYSTEM ORGANIZATION

6.5.1 BULK ORGANIZATION

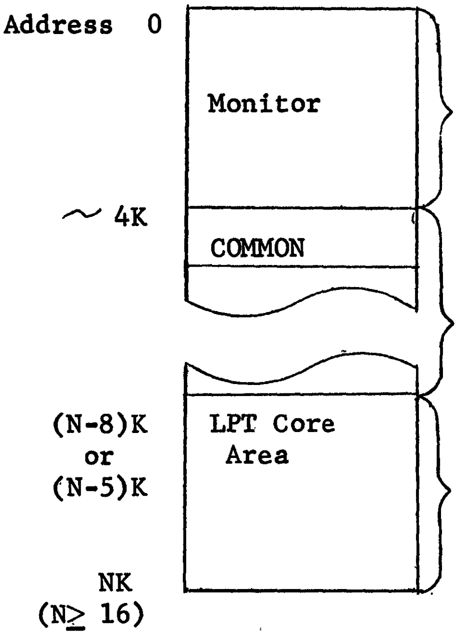
Sizes shown below are approximate. They are given to indicate the relative size of the units within the Bulk Organization.



Various Sizes, Static	12K	A	Monitor, Submonitor, OPR
Fixed	22K	B	LPT Programs
	8K	C	LPT SAVE Area
	2K	D	System EQL Table
Various Sizes, Static		E	Real-Time Library Catalog (Optional)
		F	FORTRAN Library Catalog
		G	Free-Time Library Catalog
		H	Data Table Library Catalog (Optional)
		I	Untested Library Catalog
Various Sizes, Dynamic		J	Real-Time Library
		K	FORTRAN Subroutine Library
		L	Free-Time Library
		M	Data Table Library (Optional)
		N	Untested Library
		0	Temporary Bulk Area (LPT Scratch Area)

6.5.2 CORE ORGANIZATION

1. GENERAL



The Monitor shown includes all peripheral devices needed by the Free-Time System plus a full Submonitor. It does not include those drivers associated with Process I/O, nor does it include permanent core system functions.

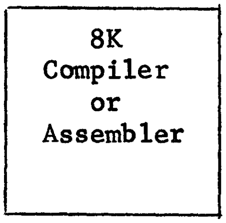
Real-Time Core Area

When the LPT is running, this area of core is used for its programs.

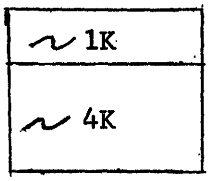
NK = Maximum Core Area

2. LPT USAGE OF CORE

A. Language Processing



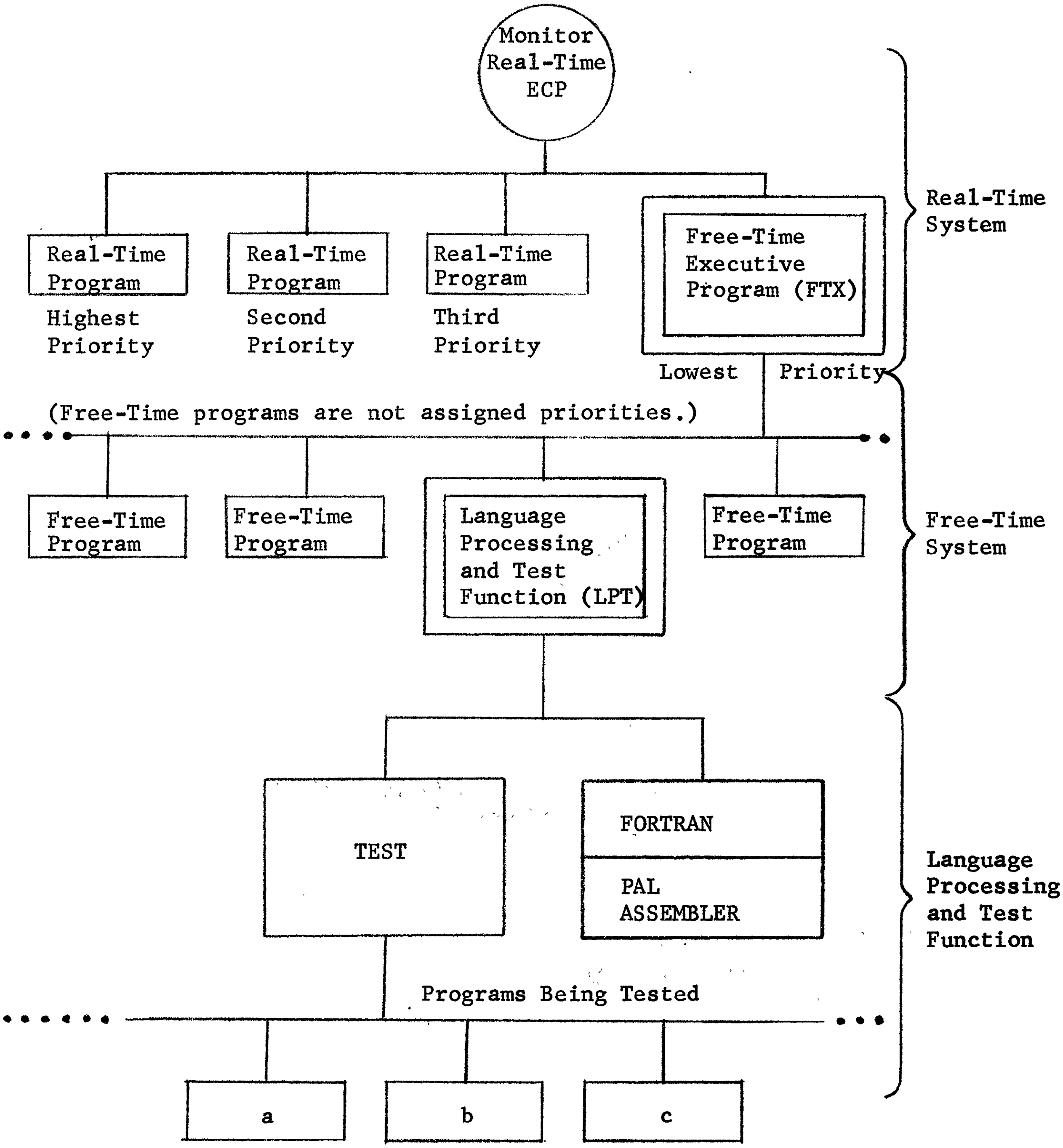
B. Testing (numbers are approximate)



Memory Protect Software

Program Being Tested

6.5.3 PROGRAM ORGANIZATION



6.6 SYSTEM DESCRIPTION

6.6.1 FREE-TIME EXECUTIVE PROGRAM (FTX)

The Free-Time Executive Program is under direct control of the Monitor system (as outlined on P. 9). It is called via the Real-Time Executive Program by the INPUT DEMAND Button on the card reader. The Free-Time Executive Program finds the name of the program to be executed by reading a JOB Card. When the JOB Card calls for a free-time program, the Free-Time Executive Program makes request for Monitor to operate the program.

If the JOB Card requests a valid real-time program, a request is made to the Real-Time Executive Program to execute the program. Some real-time programs may not be called by a JOB Card. For example, the Output Program is only turned on by the Output Request Subroutine; the Input Program is initiated by the Input Driver, etc.[1] When a request is made which violates the Monitor system rules, the JOB Card is ignored, and an error message is printed.

6.6.2 JOB CONTROL CARD

The JOB Control Card identifies the beginning of the card deck associated with a job. The JOB Card specifies the program to be turned on, and, if it is a free-time program, specifies a heading line for printer output.

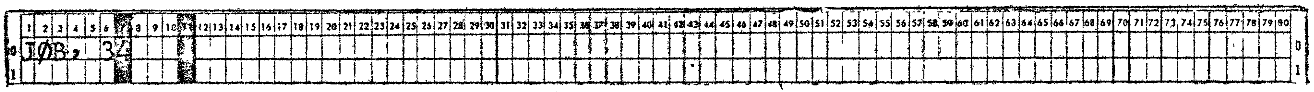
The JOB Card contains three fields, separated by commas. The last field terminates in Column 80. Single blank columns are ignored except in the heading, and may be used anywhere for clarity.

The first field of the JOB Card must contain only the word "JØB".

The second field contains the program name or number and is interpreted as follows:

1. If the first character is numeric, the field is assumed to contain three or fewer digits, which is the real-time program number.

EXAMPLE 1:



JØB, 34

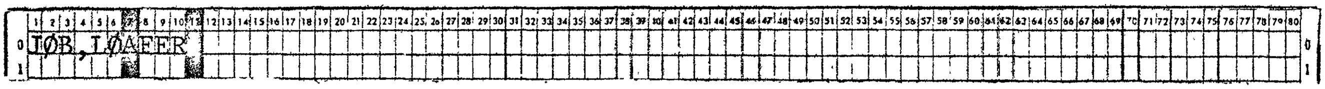
Real-Time Program - Called by Monitor Priority Number.

[1] Refer to Appendix A.

2. If the first character is alphabetic, the first six characters of the field are used as the name of a free-time program.

   If the program referred to is a free-time program, the third field is used as the printer output heading for the job.

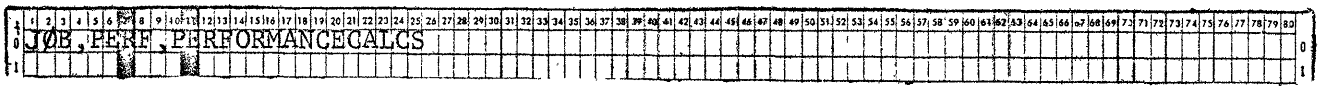
EXAMPLE 2:



```
JØB,LØAFER
```

Free-Time Program with Six Letter Name and No Printer Heading.

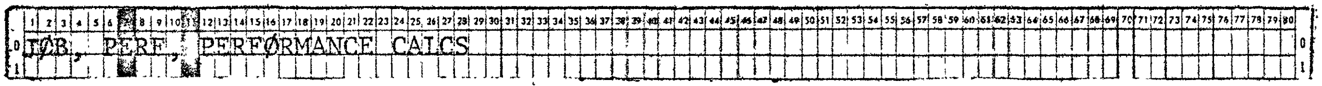
EXAMPLE 3:



```
JØB,PERF,PERFORMANCECALCS
```

Free-Time Program with Name Less Than Six Characters and Printer Heading.

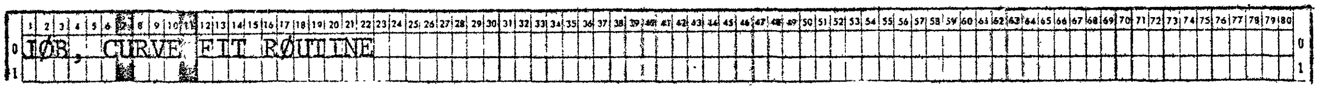
EXAMPLE 4:



```
JØB, PERF, PERFØRMANCE CALCS
```

Spaces Added for Clarity Using Example 3.

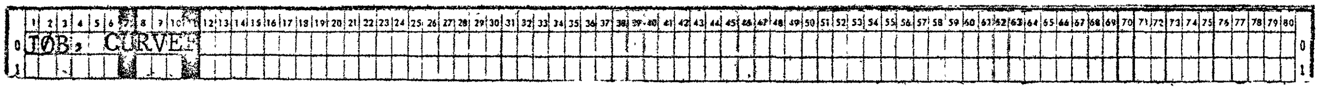
EXAMPLE 5:



```
JØB, CURVE FIT RØUTINE
```

Free-Time Program With A Name of More Than Six Characters.
The first six characters (spaces deleted) make up program name.

EXAMPLE 6:



```
JØB, CURVE
```

Abbreviated Form of Example 5.

Any JOB Card which does not follow the prescribed format is ignored by the Free-Time Executive Program, and the card is printed, followed by an error message.

6.7 LANGUAGE PROCESSING AND TEST (LPT) FUNCTION

6.7.1 GENERAL INFORMATION

The LPT Function is used to compile, test, and add new programs to the over-all program system.

The LPT Function has the ability to:

1. Compile and assemble, then:

   a. Output the program to bulk,
   b. Execute the program ("compile and go"), and
   c. Output a binary program deck (Card punch necessary).

2. Load a program from binary cards, then:

   a. Output the program to bulk, and/or
   b. Execute the program - "load and go" (Card punch necessary).

3. Test programs stored in bulk memory. The programmer may:

   a. Make octal corrections,
   b. Take snapshot dumps, starting and ending run addresses,
   c. Specify a bulk memory area in which storing of information is permitted,
   d. Specify the availability of each input/output device.

4. Maintain programs in bulk memory which are:

   a. Run under control of the System Monitor,
   b. Run by the Free-Time System,
   c. Available to the TEST System.

5. Maintain a library of FORTRAN subroutines in bulk memory.

6. Maintain a library of data tables (optional).

7. Maintain a table of symbols and their values or locations which have been specified as "common", and to provide the ability to:

   a. Add or remove symbols from the common symbol table,
   b. Remove the entire common symbol table (Card punch necessary).
   c. Load a new common symbol table from binary cards (Card punch necessary).

The LPT Function consists of:

1. A series of programs, each designed to do a particular function, such as read cards, check card validity, compile, test, punch cards, and print.

2. A sequencing program which processes control cards and chooses the programs to accomplish the desired function.

Programs may be entered into the Free-Time System by one of two methods:

1. Compiling FORTRAN language statements.

2. Loading binary cards produced by the GE/PAC language processor.

The program name or number and the particular library into which this program is being placed are specified on the COMPILE or LOAD Control Card preceding the input deck.

Program libraries exist for the following:

1. Real-time programs (numbered programs, 1, 2, 3, ...) executed directly by Monitor. The system status of these programs must be specified when the program is entered in this library.

2. Free-time programs (six character alphanumeric names, LOAFER, CURVEF, etc.) executed by FTX.

3. FORTRAN subroutines which may be called by any program when that program is compiled.

4. Data tables (numbered) which may be retrieved by any program (option).

5. Untested real- or free-time programs, executed by LPT under surveillance of memory - protect hardware and software.

6.7.2 LPT CARDS, GENERAL DESCRIPTION

Single blank columns may be used in coding for formatting and clarity. They are ignored by the LPT Function. Two blank columns in a row signify the end of information on a card.

Four types of control cards exist for the LPT Function - Action, TEST Operating Condition, System Protection, and END. These cards are explained in detail, with examples, as they arise in the text.

6.7.2.1 Action Control Cards

Action Control Cards indicate actions taken by the LPT Function:

a. FIN - signifies the end of a job. All programs in temporary status are removed, and control is returned to FTX which will process the next JOB.
b. LOAD - loads the following binary program or symbol table card deck into bulk storage.
c. REMOVE - removes from the system a program or symbol table which has been entered by LOAD or COMPILE.
d. COMPILE - initiates a FORTRAN compilation or a PAL assembly, and transfers the object program to bulk storage. With a card punch, a binary deck may be produced.
e. TEST - transfers a free-time untested program to core for execution by LPT.

The REMOVE Card can be followed only by an action control card. The FIN Card can be followed only by a JOB Card.

Each of the others is followed by either language cards to be compiled, a binary deck to be loaded, or cards giving TEST operating conditions or data.

6.7.2.2 TEST Operating Condition Cards

TEST Operating Condition Cards are CORRECT, DUMP, and EXECUTE. These cards modify the program, specify the breakpoints for dumping, and supply additional information for processing environment.

a. CORRECT - corrects the program as it exists in core, using symbolic or numeric addresses and octal contents.
b. DUMP - gives locations and limits for snapshot dumps.
c. EXECUTE - executes the program between given locations.

6.7.2.3 System Protection Cards

System Protection Cards are BULK STORE, INPUT, OUTPUT, and PROGRAM. These cards specify the memory and input/output protection during the testing of a program, and define those real-time programs which may be legally initiated by that program.

a. BULK STORE - specifies the bulk area into which the program being tested may write.
b. INPUT - specifies the input devices which the program being tested may use.
c. OUTPUT - specifies the output devices which the program being tested may use.
d. PROGRAM - allows the program being tested to initiate the execution of programs in the Real-Time Library through the Real-Time Executive Program.

All of the Type 2 and Type 3 cards listed above have a variable format with fields separated by commas. Each field is uniquely identified by the first six characters or by the prepositions AT, FROM or TO. If there are less than six characters in a program name, blanks are added for the remaining characters to form a unique six-character identifier. (Any field may exceed six characters.)

Numeric addresses are assumed to be relative to the beginning of the program except on the BULK STORE Card.

6.7.2.4 END Cards

END Cards are:

a. END COMPILATION - indicates the end of a FORTRAN (symbolic) deck.
b. END TEST - indicates the end of a program test deck.

Both of these cards must be followed by an Action Control Card.

Because of constant reference to these cards throughout the manual, they are noted as they are first encountered. There is no separate section explaining these two cards.

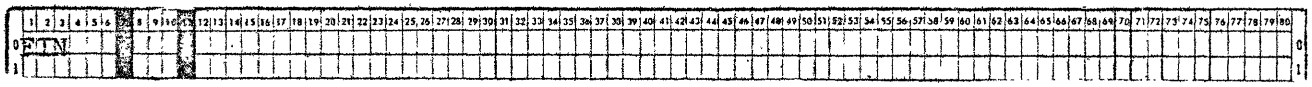
6.7.3 LPT CARD USAGE

6.7.3.1 Action Control Cards (Action To Be Taken)

FIN Control Cards

FIN signifies "end of job" and removes all programs in the system that are in temporary (initial testing) status.

EXAMPLE 7:

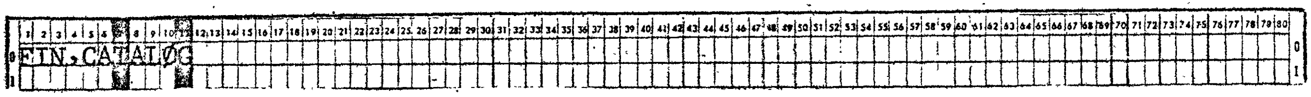


FIN

FIN Control Card - No Catalog Specified.

An option on the FIN Cards allows the programmer to determine what programs are in the system and how much bulk storage is used by them.

EXAMPLE 8:



```
FIN,CATALØG
```

FIN Control Card - Catalog Specified.

If a catalog is requested, the following items are listed:

1. Names (Numbers) and sizes for:

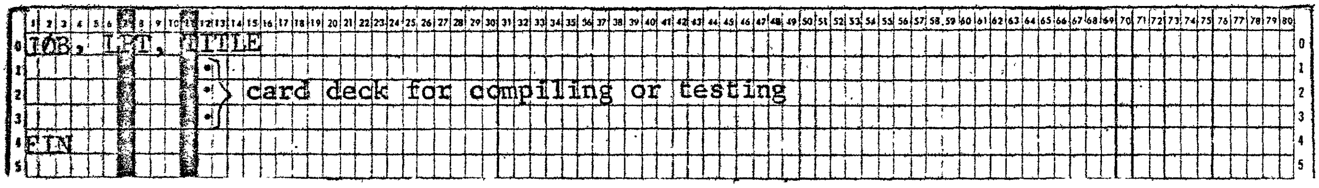
   a. Real-time programs in "permanent" status.[2]
   b. Free-time programs in "permanent" status.
   c. Subroutines in the FORTRAN library.
   d. Data tables.
   e. Programs (either real-time or free-time) in the TEST library.

2. Total bulk memory required for:

   a. Real-time programs
   b. Free-time programs
   c. FORTRAN library subroutines
   d. Data tables
   e. Programs in the TEST library

3. Remaining bulk storage.

EXAMPLE 9:

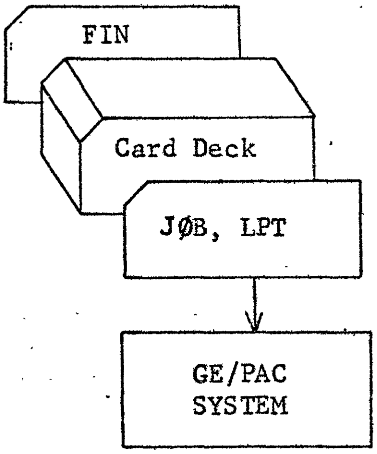


```
JØB, LPT, TITLE
           •
           • } card deck for compiling or testing
           •
FIN
```

FIN Control Card Used with Card Deck.

[2] Programs in PERMANENT status are under direct control of the Free-Time or Real-Time Executive Programs. Programs in the TEST library may be run only under the LPT Function

The following illustration depicts Example 9:



When a FIN Card is missing or another card error results in ignoring the FIN Card, a subsequent JOB Card may appear to be out of sequence. If this happens, the JOB Card is treated as a FIN Card, and the job is ignored.

LOAD Control Card

* The Load Control Card loads a binary program deck and symbol table deck into bulk storage

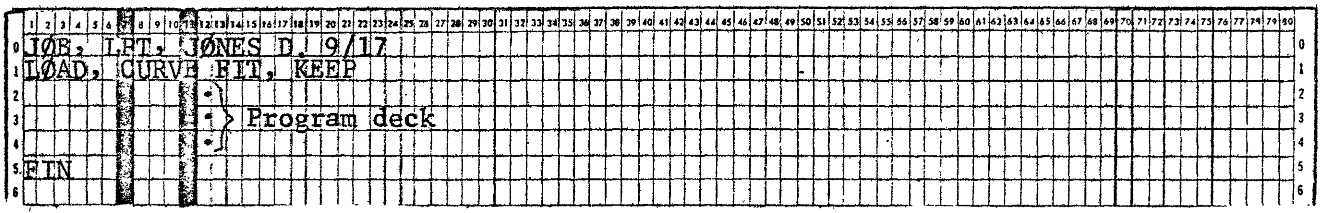
The first field on the card contains the word "LOAD"

The second field on the card is the program or data table number (three digits), the first six characters (excluding blanks) of the program name, or the word "SYMBOLS" In Example 10, the letters CURVEF represent the program name. If no name is given, the LPT Function assigns the program to the TEST library in temporary status. An attempt to load subsequent programs with no name stated, prior to a FIN Card, causes these programs to be ignored. The FIN Card causes removal of such temporary programs.

The third field indicates the status of the program as either PERMANENT, KEEP, DATA, or FORTRAN (per Example 10). If no status is specicified, the program is placed in the TEST library in temporary status and is removed by a FIN Card. This status indicates what happens to the material once action, as directed by the control card, has been completed. For example, the status KEEP places the material into the TEST library on bulk storage, to be called only by the TEST Card. Status is discussed following Example 10 and its illustrations.

* Revision 1/67

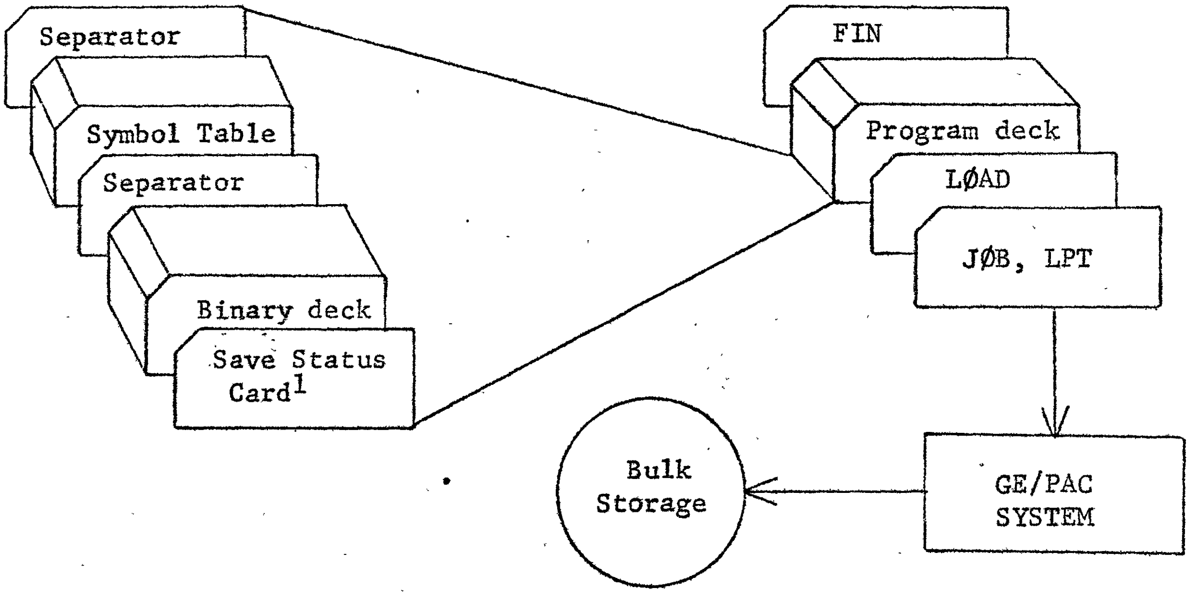
EXAMPLE 10:



```
JØB, LPT, JØNES D. 9/17
LØAD, CURVE FIT, KEEP
            .
            . } Program deck
            .
FIN
```

Load A Binary Deck and Symbol Table

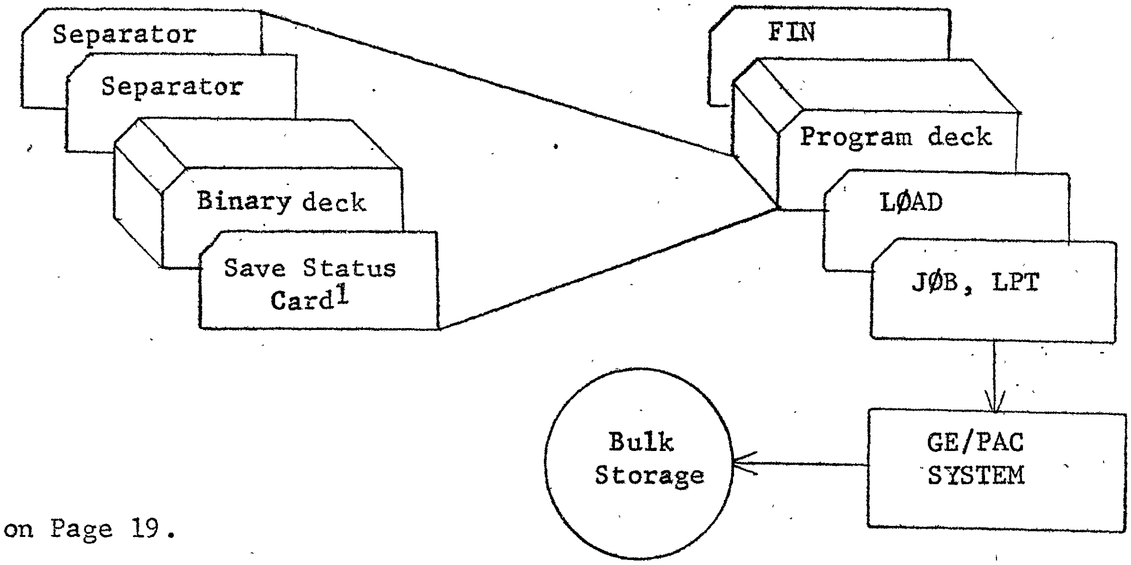
The following illustration depicts a binary LOAD deck produced by the Free-Time System (with optional card-punch) and its storage (from Example 10).



The following illustration depicts a binary LOAD deck from some source other than the Free-Time System. A symbol table is expected by the loader. If none exits, as per illustration, an

[1] The SAVE STATUS CARD contains an 8-digit octal number in Columns 1 through 8, stating the size of the Save Status Area for this program. It is punched by the system when binary cards are requested. It must be hand-processed and added to the deck if the source is other than the Free-Time System.

extra blank (separator) card must be included. This figure also applies to the coding from Example 10.

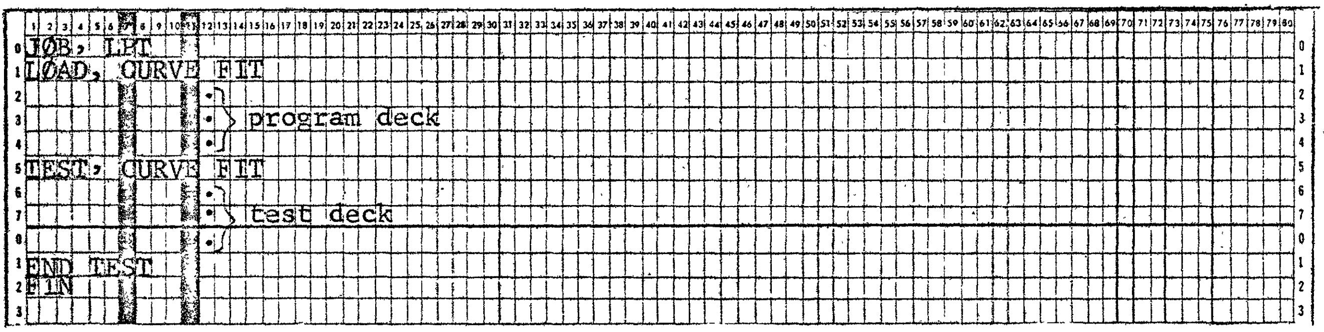


[1] See Footnote on Page 19.

No Status Specified

If no status is specified on the LOAD Card, the program is placed in the TEST library in temporary status and is removed by a FIN Card (per Example 11).

EXAMPLE 11:

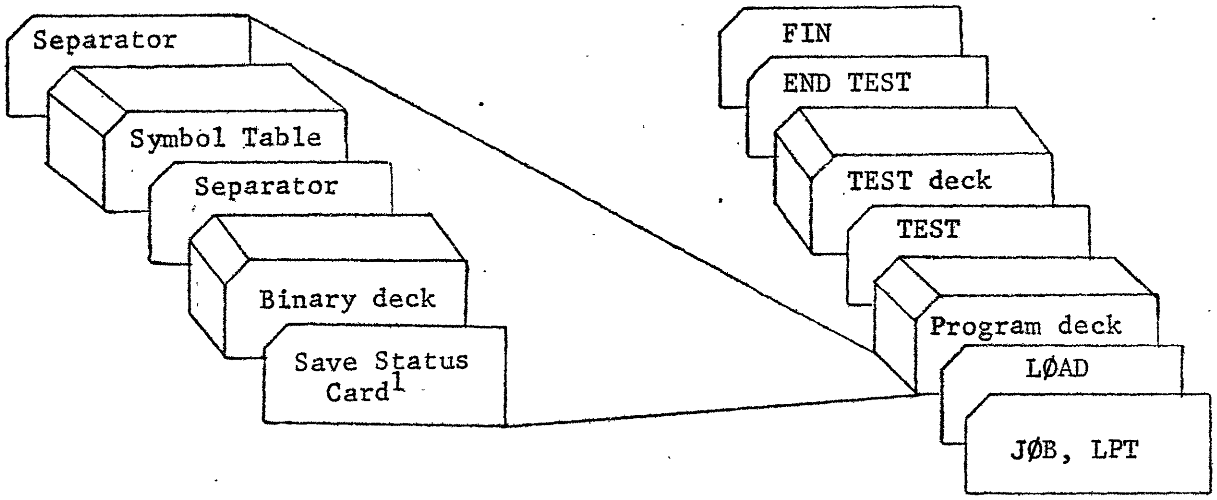


```
0 JØB, LPT
1 LØAD, CURVE FIT
2            .
3            . } program deck
4            .
5 TEST, CURVE FIT
6            .
7            . } test deck
0            .
1 END TEST
2 FIN
3
```

Load A Binary Card Deck and Symbol Table for Testing.

(The program is in temporary status and is removed from the system by the FIN Card. Note the use of the END TEST Card to halt the testing.)

The figure below illustrates the LOAD deck for Example 11:

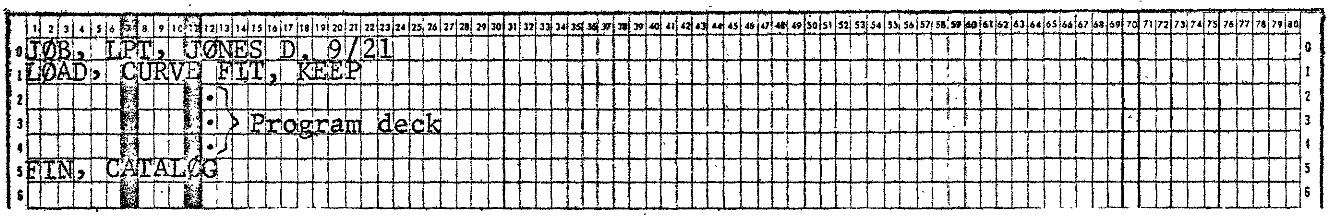


[1] See Footnote on Page 19.

KEEP Status

Programs in KEEP status are retained in the TEST library following a FIN Card and may only be called by the LPT Function via a TEST Card.

EXAMPLE 12:

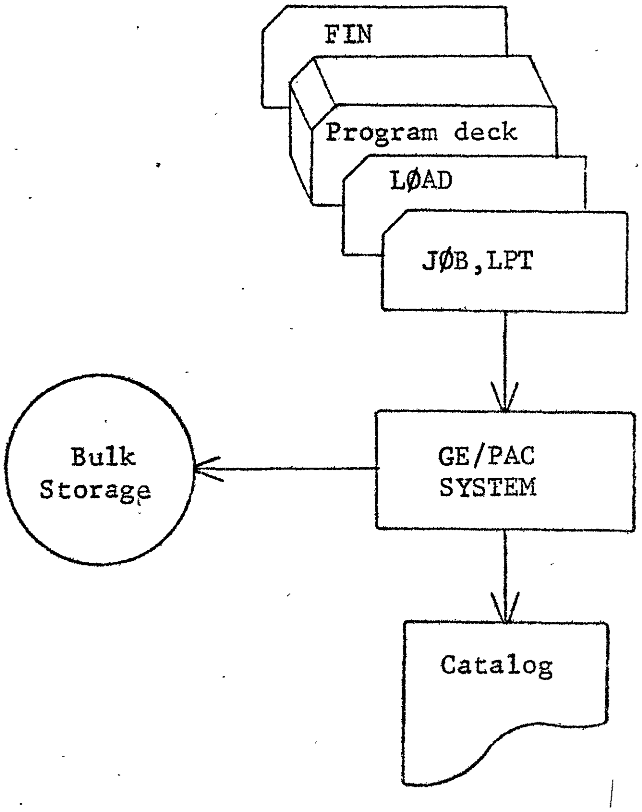


```
JØB, LPT, JØNES D, 9/21
LØAD, CURVE FIT, KEEP
           •
           • } Program deck
           •
FIN, CATALØG
```

To Load A Program for Subsequent Testing and Catalog Listing.

In Example 12, the program is placed in KEEP status. It is not removed by the FIN Card and need not be reloaded for subsequent testing. A listing of all program libraries and the bulk storage allocation (the catalog) is produced on the printer.

The following illustration depicts the deck, its storage, and the catalog printout:



PERMANENT Status

Programs in the PERMANENT status are also retained in bulk storage following a FIN Card. They may be initiated either by the Free-Time Executive Program via a JOB Card or by the Real-Time Executive Program via a program turn-on.

Programs placed in PERMANENT status must be identified by:

1. A number, for a real-time functional program operating under the Real-Time Monitor Executive Program, or

2. A symbolic name, for a free-time library program which is called with a JOB Card.

A PERMANENT program operating under control of the Real-Time Executive Program must have its operating parameters specified on the LOAD Card. The following states are assumed if no status is specified on the card:

1. The program's core area is available for use by higher-priority programs.

2. The program can be initiated by a JOB Card.

3. The program contains FORTRAN I/O requests subroutines, and/or functions.

Operating conditions other than those shown above may be specified as follows:

1. UNAVAILABLE

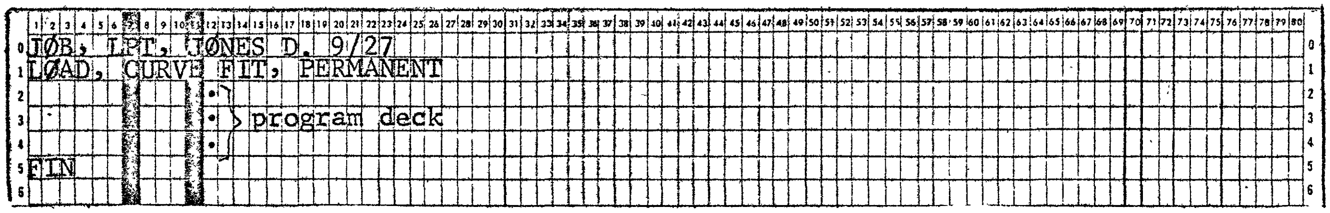
2. NØ JØB

3. NØ SUBRØUTINES

Every LOAD Card specifying a PERMANENT real-time (numbered) program must specify those conditions which will differ from the stated assumptions.

The following examples give the coding for loading finished programs with PERMANENT status into free-time and real-time libraries.

EXAMPLE 13:



```
JØB, LPT, JØNES D. 9/27
LØAD, CURVE FIT, PERMANENT
           •
           • } program deck
           •
FIN
```

To Load A Finished Program into The Free-Time Library.

In Example 13, the program is to remain in bulk storage for regular use by programmers. Following this job, the program may be initiated by a JOB Card.

In Example 14, the program will remain in bulk storage and can be initiated by the Real-Time Executive Program. The program will not allow its core area to be used by another program during its execution. The program cannot be initiated by a JOB Card.

EXAMPLE 14:

```
JØB, LPT, J.SMITH 10/4
LØAD, 41,PERMANENT,UNAVAILABLE,NØ JØB
          .
          . } Program deck
          .
FIN
```

To Load A Finished Program into The Real-Time Library.

FORTRAN Status

An additional status, FORTRAN, is used for programs which were compiled as FORTRAN subroutines. These subroutines are loaded without symbol tables (per illustration, p.20).

To load a binary deck into the FORTRAN subroutine library, use the following instruction: LØAD, (program name), FØRTRAN.

When it is desired to replace the removed system symbols, the binary deck produced by REMOVE, SYMBOLS (p.25) is preceded by the instruction LØAD, SYMBØLS.

REMOVE Control Card

By using the REMOVE Control Card, programs may be removed from any library, or the common symbol table may be cleared.

The first field of the REMOVE Card contains the word "REMØVE".

The second field of the card contains either the three-digit number or the six-character alpha-numeric name of the program, or the word "SYMBØLS".

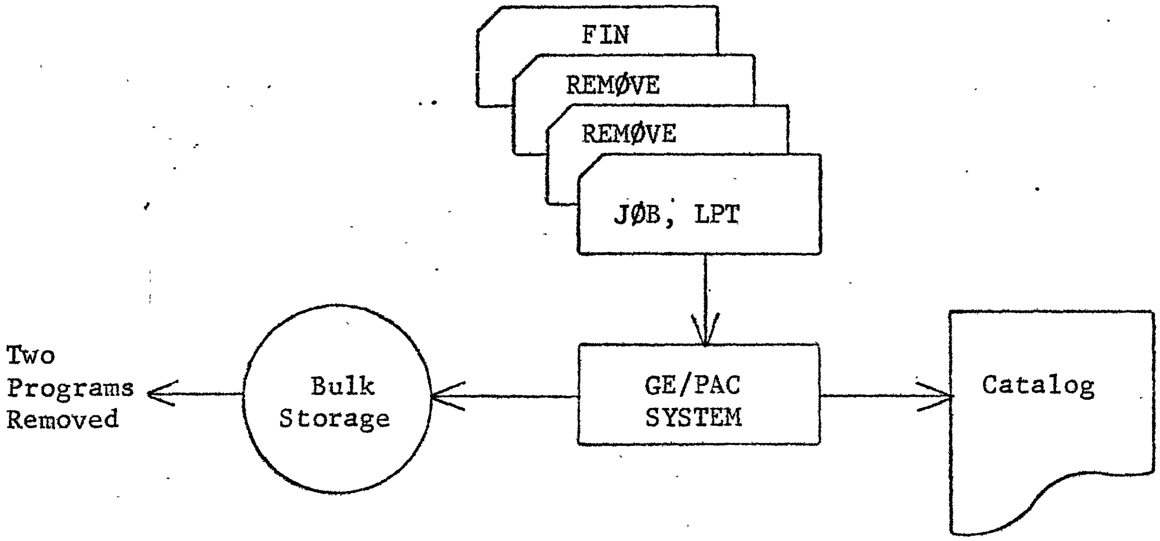
The third field contains the status of the program (PERMANENT, KEEP, DATA, or FORTRAN).

EXAMPLE 15:

```
JØB,LPT
REMØVE,LEAST SQUARES,KEEP
REMØVE, 41,PERMANENT
FIN, CATALØG
```

Remove TEST Program "LEASTS" and Real-Time Program Number 41 from The System and Print a Catalog.

The following illustration depicts Example 15:



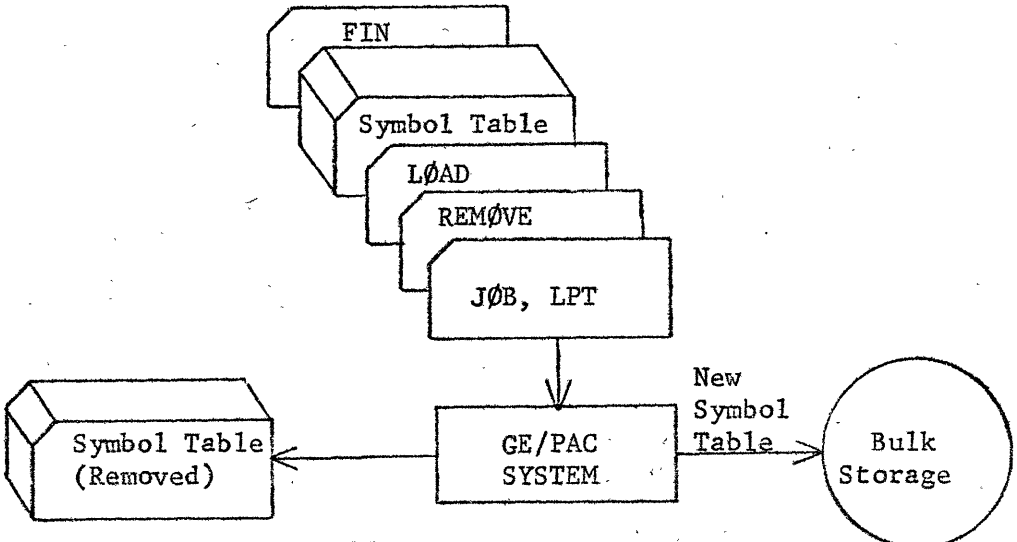
When REMOVE, SYMBOLS is requested, the symbol table dumped on binary cards. This is an option - a card punch is necessary.

EXAMPLE 16:

```
JØB, LPT
REMØVE,SYMBØLS
LØAD,SYMBØLS
           .
           .  } Symbol table
           .
FIN
```

Load A Symbol Table In Place of The Current Common Symbol Table

As shown in Example 16 and the illustration below, before an existing program or the symbol table is replaced, it must be removed from the system by a REMOVE Card.



COMPILE Control Card

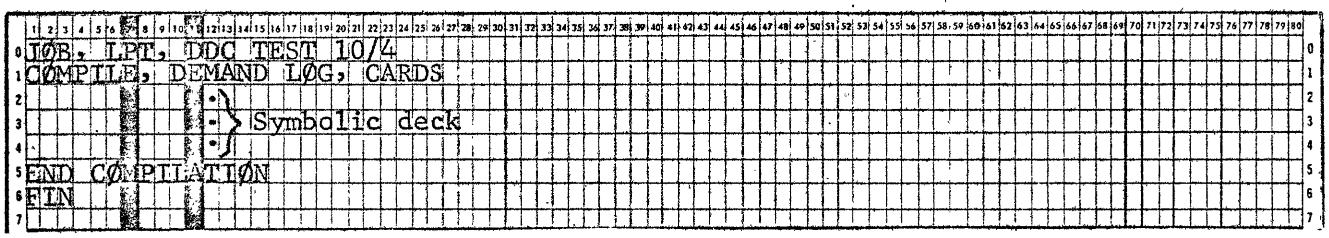
The FORTRAN (symbolic) language cards following the COMPILE Control Card are compiled using the common symbol table. The generated program is placed in bulk storage and is identified by the program number or data table number (three digits), or the first six characters (excluding blanks) of the program name. This identification is the second field of the card. If no name or number is given, the LPT Function assigns the program to the TEST library in temporary status. An attempt to load subsequent programs with no name stated, prior to a FIN Card, causes these programs to be ignored. The FIN Card causes removal of such temporary programs.

The third field of the card is the status of the program and may be specified either PERMANENT, KEEP, DATA, or FORTRAN. If no status is specified, the program is placed in the TEST library in temporary status and is removed by a FIN Card. Explanation of status, in relationship to the COMPILE Card, with examples, follows.

All symbolic decks beginning with a COMPILE Control Card must be followed by an END COMPILATION Card. All cards in the symbolic deck must have a key in Column 70 indicating whether they are PAL or FORTRAN statements.[1] The symbolic card preceding the END COMPILATION Card must be an END Card (symbolic) with a 7 in Column 70.

No Status Specified

EXAMPLE 17:



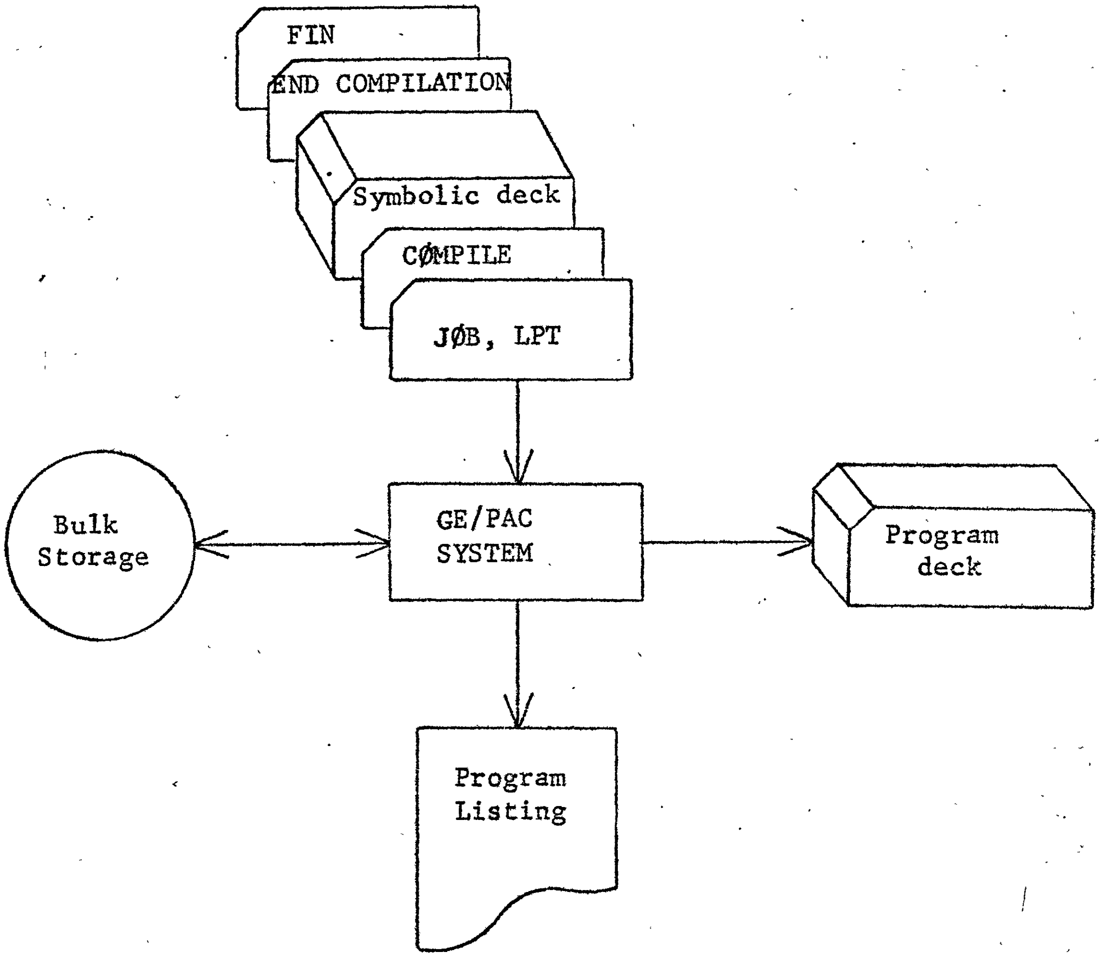
```
0 JØB, LPT, DDC TEST 10/4
1 CØMPILE, DEMAND LØG, CARDS
2            ·}
3            ·} Symbolic deck
4            ·}
5 END CØMPILATIØN
6 FIN
7
```

Symbolic Compilation to Cards (Program Deck).

In Example 17, the name DEMAND is assigned to the program. Since PERMANENT, KEEP, DATA, or FORTRAN is not given in the third field, the program is placed in temporary status and is removed by the FIN Card.

[1] The numeral "6" indicates PAL.
The numeral "7" indicates FORTRAN.

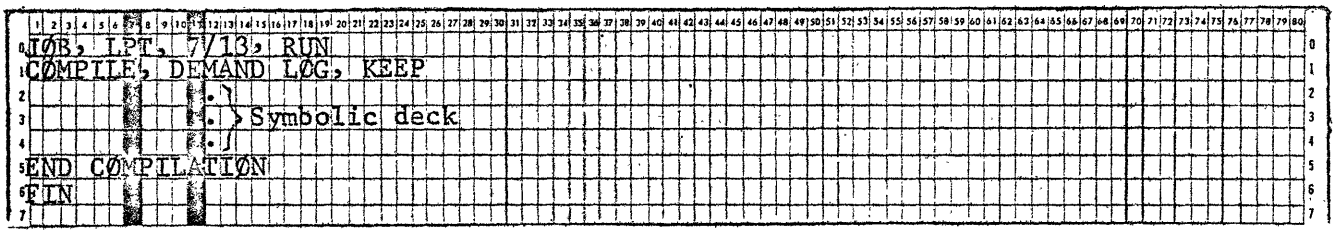
The following illustration depicts Example 17:



KEEP Status

Programs in KEEP status are retained in the TEST library following a FIN Card and may be called only by the LPT Function via a TEST Card.

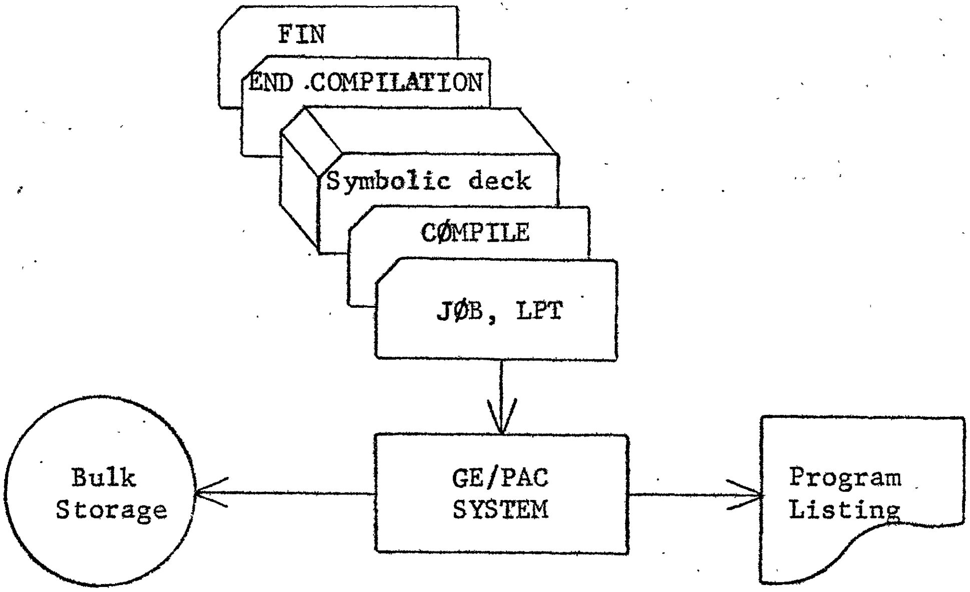
EXAMPLE 18:



```
JØB, LPT, 7/13, RUN
CØMPILE, DEMAND LØG, KEEP
           .
           . }Symbolic deck
           .
END CØMPILATIØN
FIN
```

FORTRAN Compilation - Program to Remain in Bulk Storage

The following illustration depicts Example 18:



Example 18 shows a program which is not tested and may only be called by LPT. It is not removed from the System by the FIN Card.

PERMANENT Status

Programs in PERMANENT status are also retained in bulk storage following a FIN Card, but they are initiated either by the Free-Time Executive Program via a JOB Card or by the Real-Time Executive Program via a program turn-on.

Programs placed in PERMANENT status must be identified by:

1. A number for a real-time functional program operating under the Real-Time Monitor Executive Program, or

2. A symbolic name for a free-time library program which is called with a JOB Card.

A PERMANENT program operating under control of the Real-Time Executive Program must have its operating parameters specified on the COMPILE Card. The following states are assumed if no status is specified on the card:

1. The program's core area is available for use by higher-priority programs.

2. The program can be initiated by a JOB Card.

3. The program contains FORTRAN I/O requests subroutines, and/or functions.

Operating conditions other than those shown above may be specified as follows:

1. UNAVAILABLE

2. NØ JØB

3. NØ SUBRØUTINES

Every COMPILE Control Card specifying a PERMANENT real-time (numbered) program must specify those conditions which will differ from the stated assumptions.

FORTRAN Status

An additional status, FORTRAN, is used for programs which are to be compiled as FORTRAN subroutines. These may be called by any program.

NO LIST Option

The NO LIST option suppresses symbolic listing of both input statements and the generated PAL coding in order to increase speed of compilation.

CARDS Option

The CARDS option produces a deck which consists of:

1. A binary program deck (including all subroutines which have been obtained from the FORTRAN subroutine library).

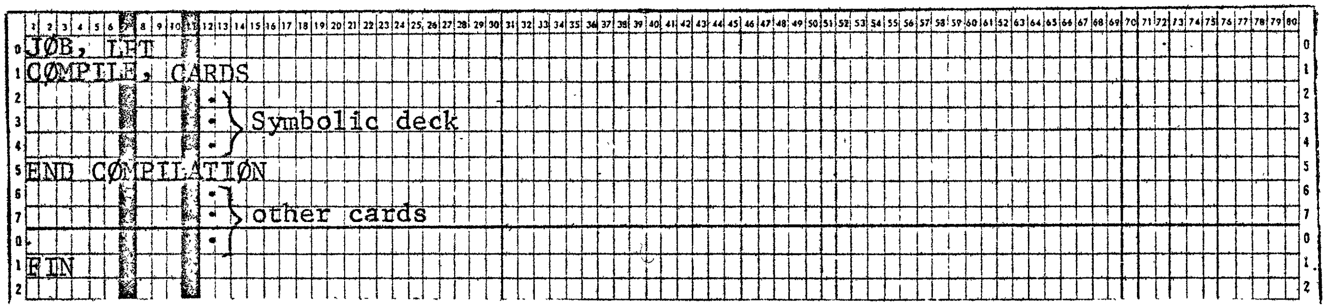
2. A separator (blank) card.

3. The symbol table used by this program.

4. A separator (blank) card.

Each card in the deck will contain the six-character or three-digit name and a sequence number in the last ten columns. This deck may be loaded using a LOAD Card.

EXAMPLE 19:

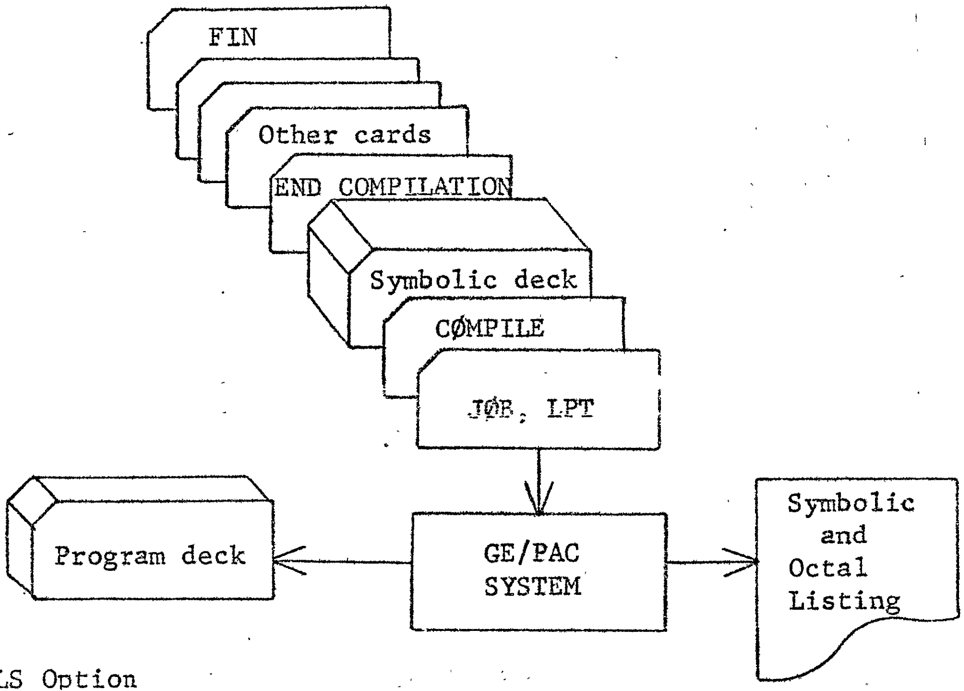


```
JØB, LPT
CØMPILE, CARDS
          .
          . } Symbolic deck
          .
END CØMPILATIØN
          .
          . } other cards
          .
FIN
```

FORTRAN (Symbolic) Compilation - No Name Specified.

Since no name is specified, temporary status is implied. A card output is specified. The compiled program is not tested. It may be loaded and run for testing purposes only, under control of LPT.

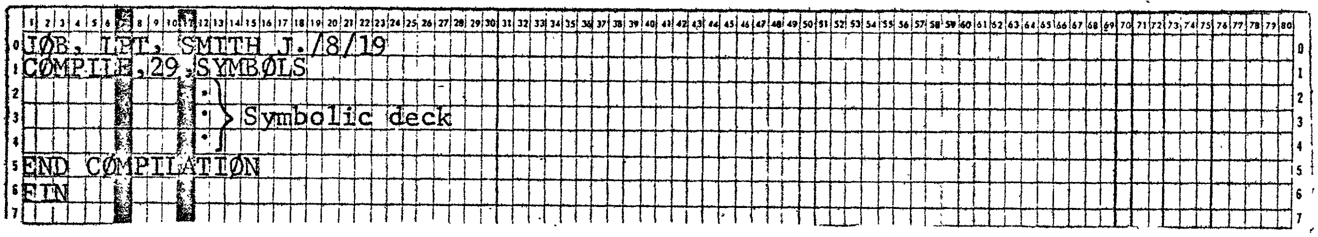
The following illustration depicts Example 19:



SYMBOLS Option

The SYMBOLS option of the COMPILE Card adds the common symbols to the common symbol table from the program being compiled. When the same name is defined in the common symbol table, both before compilation and in the program being compiled, only the value from the program being compiled is saved.

EXAMPLE 20:

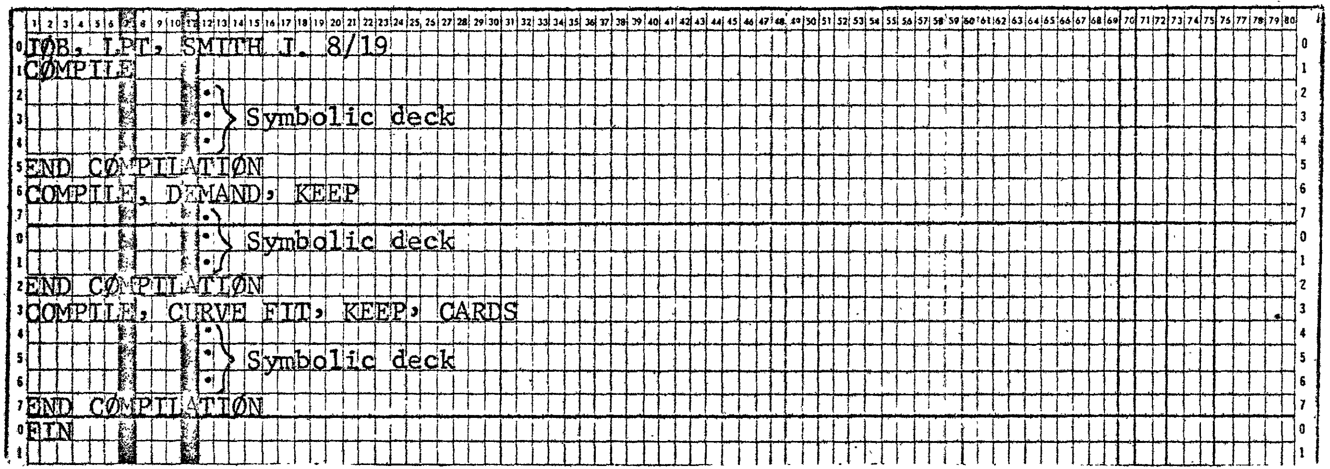


```
JØB, LPT, SMITH J./8/19
CØMPILE,29,SYMBØLS
           .
           . } Symbolic deck
           .
END CØMPILATIØN
FIN
```

FORTRAN Compilation Adding The Common Symbols in Program 29 to the Common Symbol Table.

The following example shows a batch compilation of several programs:

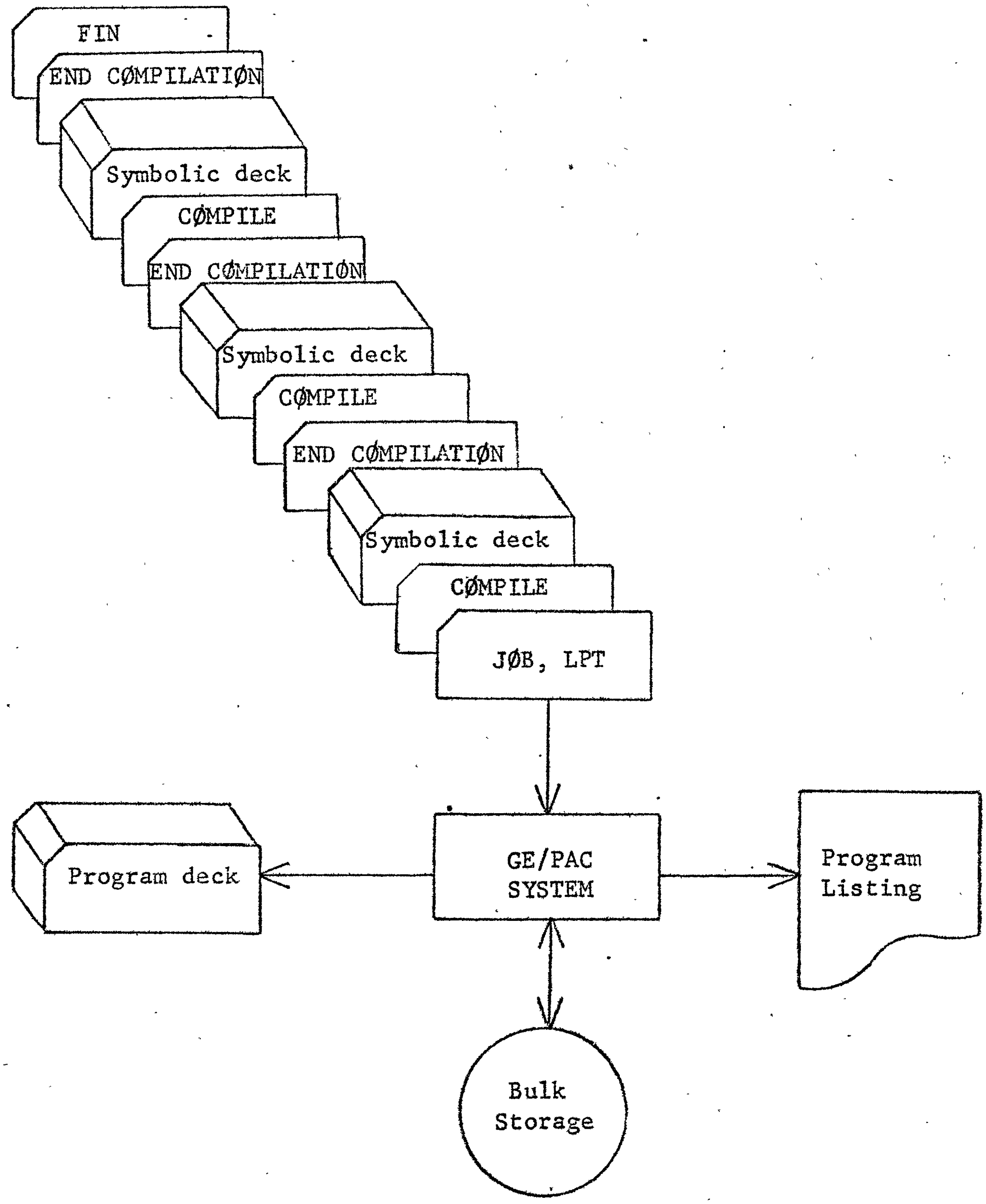
EXAMPLE 21:



```
JØB, LPT, SMITH J. 8/19
CØMPILE
           .
           . } Symbolic deck
           .
END CØMPILATIØN
COMPILE, DEMAND, KEEP
           .
           . } Symbolic deck
           .
END CØMPILATIØN
COMPILE, CURVE FIT, KEEP, CARDS
           .
           . } Symbolic deck
           .
END CØMPILATIØN
FIN
```

FORTRAN Compilation of Several Programs in Various Stages of Testing.

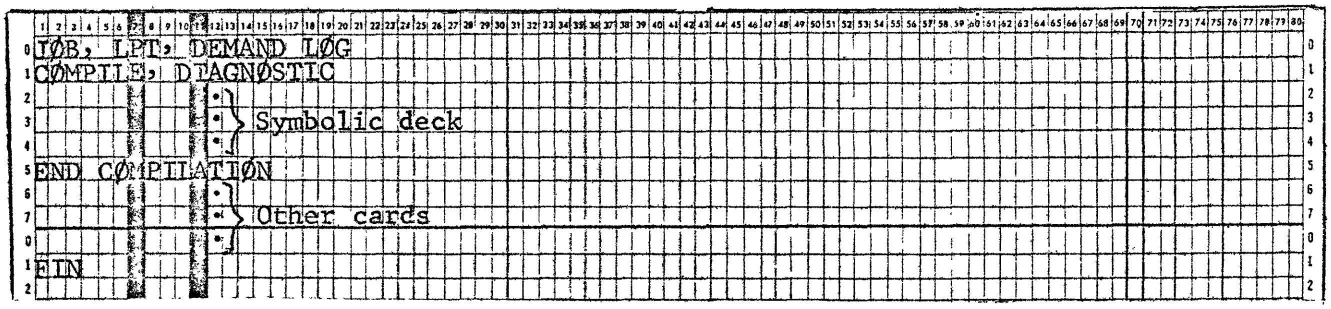
The following illustration depicts Example 21:



DIAGNOSTIC Option

Using the option with DIAGNOSTIC in the second field, generated programs are not saved, and the compiler produces a printout of all input statements. All statements containing errors are indicated with the proper error message. The common symbol table cannot be revised to include the common symbols from this program. The FORTRAN (symbolic) deck is followed by the END COMPILATION Card.

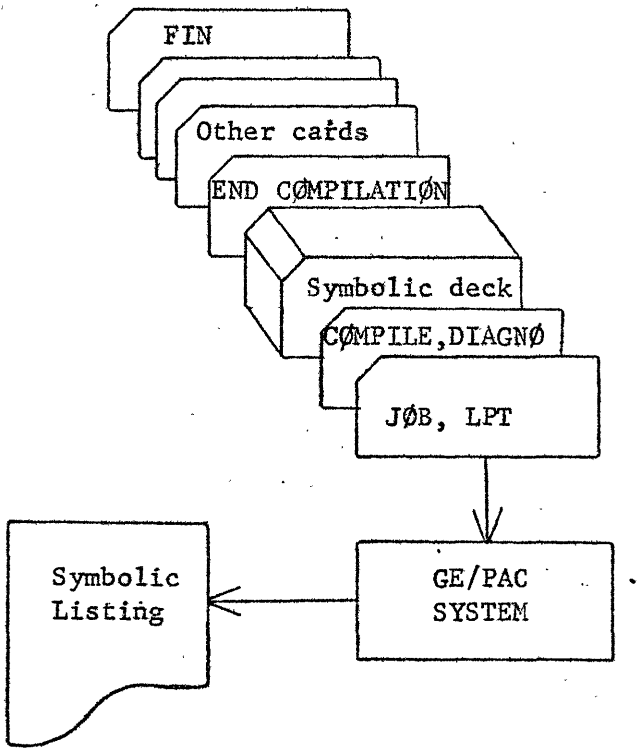
EXAMPLE 22:



```
JØB, LPT, DEMAND LØG
CØMPILE, DIAGNØSTIC
            •
            • } Symbolic deck
            •
END CØMPILATIØN
            •
            • } Other cards
            •
FIN
```

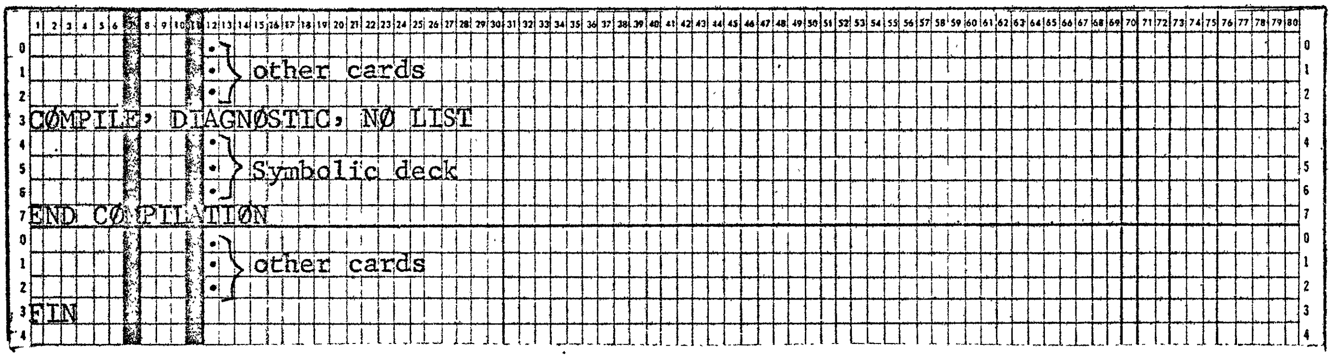
Compilation (Diagnostic) with Symbolic Listing.

The following illustration depicts Example 22:



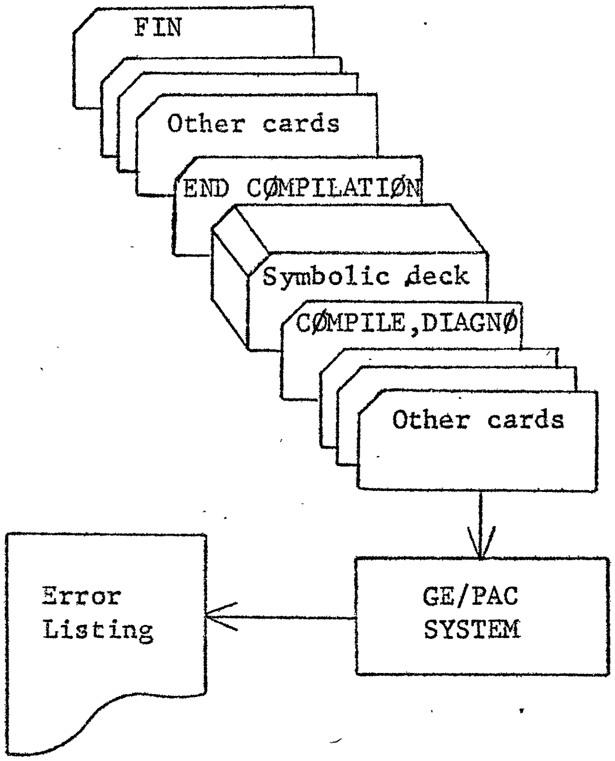
CØMPILE, DIAGNØSTIC, NØ LIST suppresses all printing other than statements in error plus corresponding error messages. The following coding example and illustration show the use of this option.

EXAMPLE 23:



```
           .
           . } other cards
           .
CØMPILE, DIAGNØSTIC, NØ LIST
           .
           . } Symbolic deck
           .
END CØMPILATIØN
           .
           . } other cards
           .
FIN
```

Compilation (Diagnostic) and No Program Listing

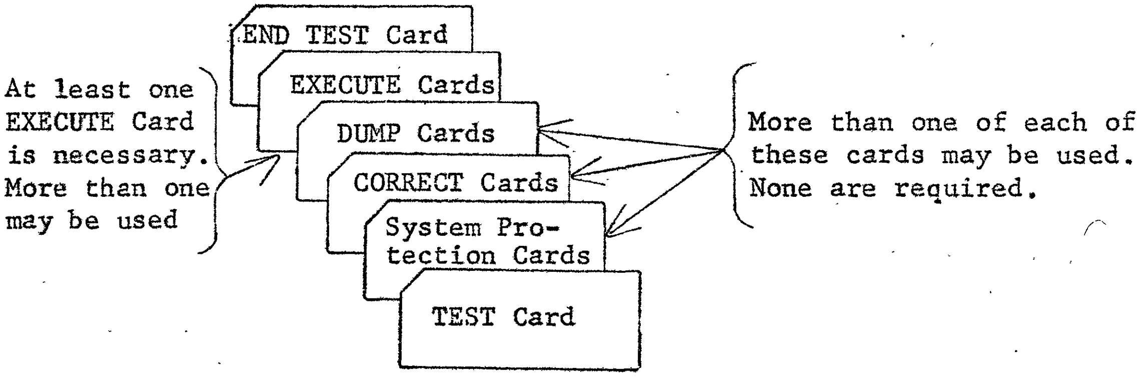


TEST Control Card

The TEST Control Card has a first field, which contains the word "TEST", and a second field which is the three-digit or six-character name of the program to be tested.

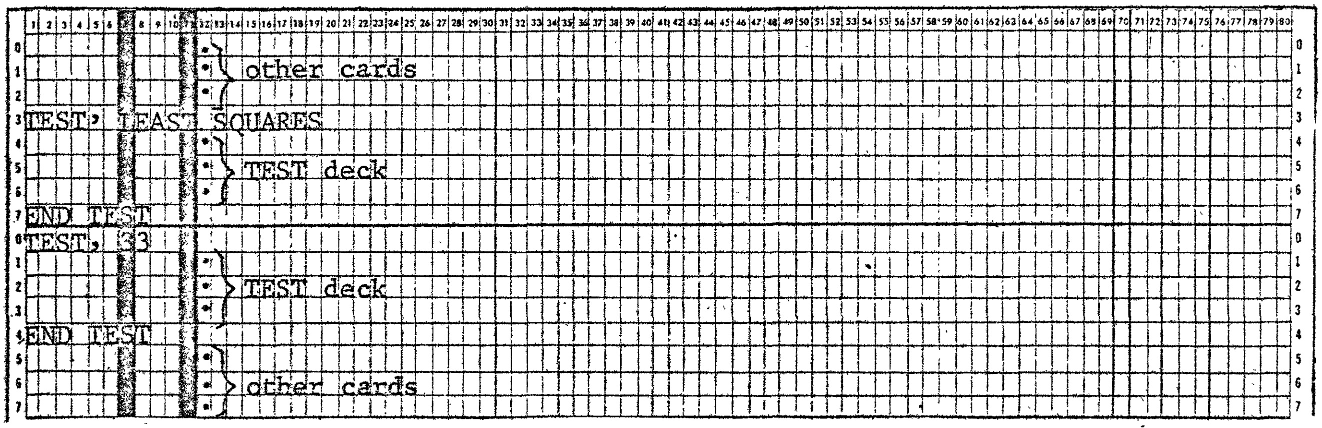
The program named on the TEST Card is tested as follows:

1. The program is called from bulk storage.
2. The operating conditions are set using:
   a. System Protection Cards - Section 6.7.3.3
   b. CORRECT Cards for modification - Section 6.7.3.2
   c. DUMP Cards to initiate snapshot dumps - Section 6.7.3.2
3. The program is executed. (EXECUTE Card - Section 6.7.3.2)
4. The TEST is terminated by an END TEST Card.


At least one EXECUTE Card is necessary. More than one may be used
END TEST Card
EXECUTE Cards
DUMP Cards
CORRECT Cards
System Protection Cards
TEST Card
More than one of each of these cards may be used. None are required.

The program named on the TEST Card may be either symbolic or numeric. If no program name is given, the temporary program is assumed. The TEST Deck must be followed by an END TEST Card.

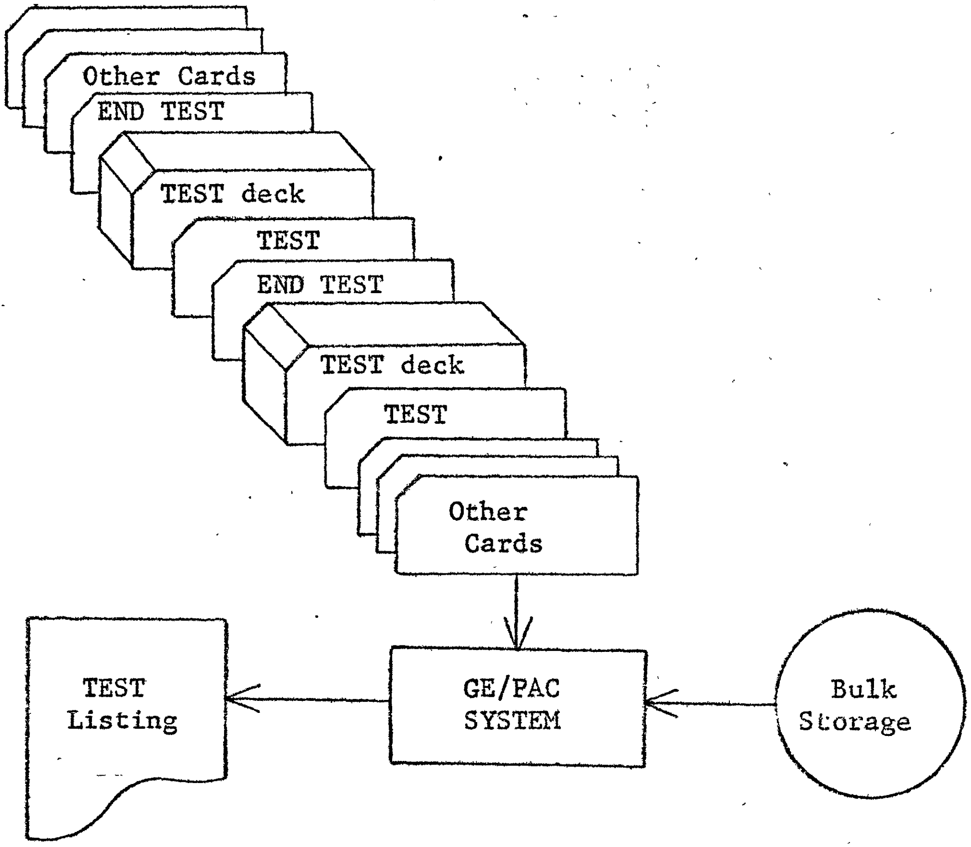
EXAMPLE 24:



```
           .
           . } other cards
           .
TEST, LEAST SQUARES
           .
           . } TEST deck
           .
END TEST
TEST, 33
           .
           . } TEST deck
           .
END TEST
           .
           . } other cards
           .
```

Programs 31 And LEAST SQUARES Are To Be Tested.

Example 24 shows two tests performed following and prior to other portions of the LPT Function; therefore, neither JOB nor FIN Cards are indicated. The following illustration depicts Example 24:



A 4K core area (4032 locations) is available for testing a program. An attempt to test a larger program results in loading and testing only the first 4K of the program.

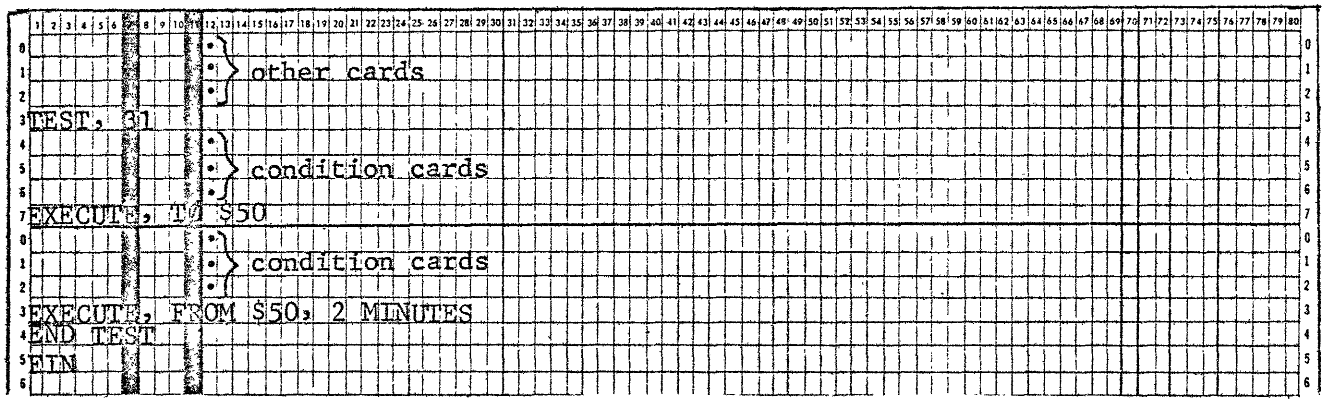
The program symbol table allows a maximum of 640 symbols. All symbols exceeding 640 are dropped after compilation and are not available for symbolic references during a TEST.

A TEST segment is defined by the starting and ending locations stated on the EXECUTE Card. If no starting address is specified, the segment begins at Location Zero.

Only one program at a time may be tested. However, each program may have more than one TEST segment. These TEST segments can be tested consecutively using only one TEST Control Card, with condition cards and EXECUTE Cards initializing segment testing.

All cards specifying conditions for the first TEST segment of the program must follow the TEST Control Card, but precede the first EXECUTE Card. Similarly, all condition cards for the second TEST segment of that program must follow the first EXECUTE Card and precede the second EXECUTE Card, etc. (See Example 25 and its illustration.)

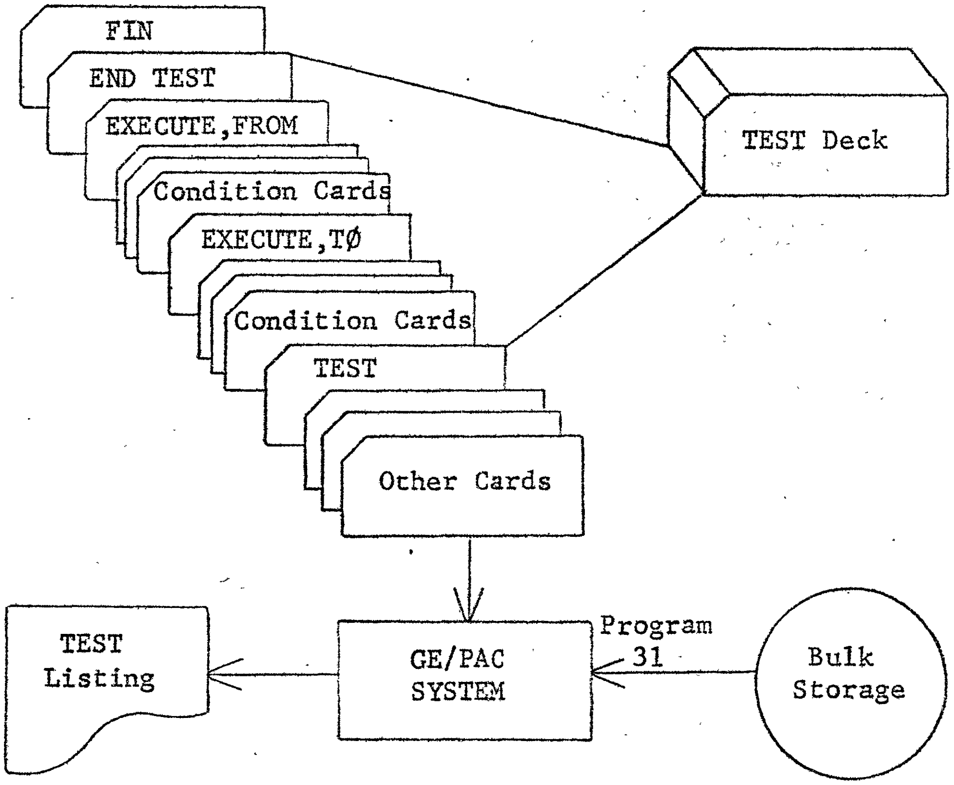
EXAMPLE 25:



```
           .
           . } other cards
           .
TEST, 31
           .
           . } condition cards
           .
EXECUTE, TØ $50
           .
           . } condition cards
           .
EXECUTE, FROM $50, 2 MINUTES
END TEST
FIN
```

Segments of Program 31 Are To Be Tested

The following illustration depicts Example 25:



6.7.3.2 TEST Operating Condition Cards (Test Parameter Cards)

EXECUTE Card

The EXECUTE Card initiates the running of a program segment. It gives the first (FROM) and last (TO) locations to be executed during the TEST of this segment.

If the FROM statement is not given, execution starts at the first location of the program. When the TO statement is not given, the segment is terminated by either the maximum time being exceeded, a program turn-off, or an invalid branch instruction.

The maximum allowable running time for each segment of the TEST is specified on the EXECUTE Card in minutes. If no time is specified (per Example 27, following), the maximum real-time allowed is set to one minute.

When an EXECUTE Card is encountered, the program is executed starting at the FROM location and continuing until:

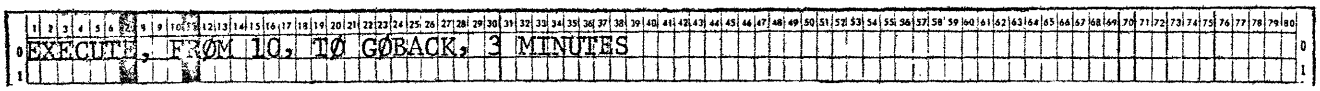
1. An invalid branch instruction is encountered.

2. The TO location is executed.

3. The maximum time limit specified on the EXECUTE Card is exceeded.

4. A Monitor Turn Program Off request is encountered. (Refer to YPG51M - MONITOR USER'S MANUAL.)

EXAMPLE 26:



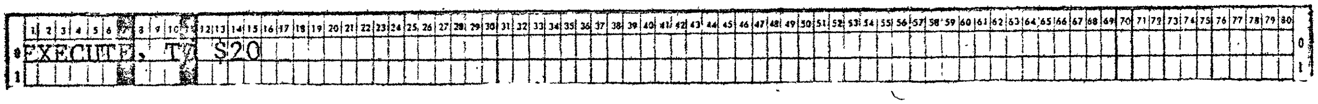
```
EXECUTE, FRØM 10, TØ GØBACK, 3 MINUTES
```

Execute A Program From the 10th Location After The Beginning Of The Program To The Symbolic Location GØBACK

(The maximum real-time allowed for the test is three minutes.)

The following example shows coding which executes a program from its origin to Statement $20:

EXAMPLE 27:



```
EXECUTE, TØ $20
```

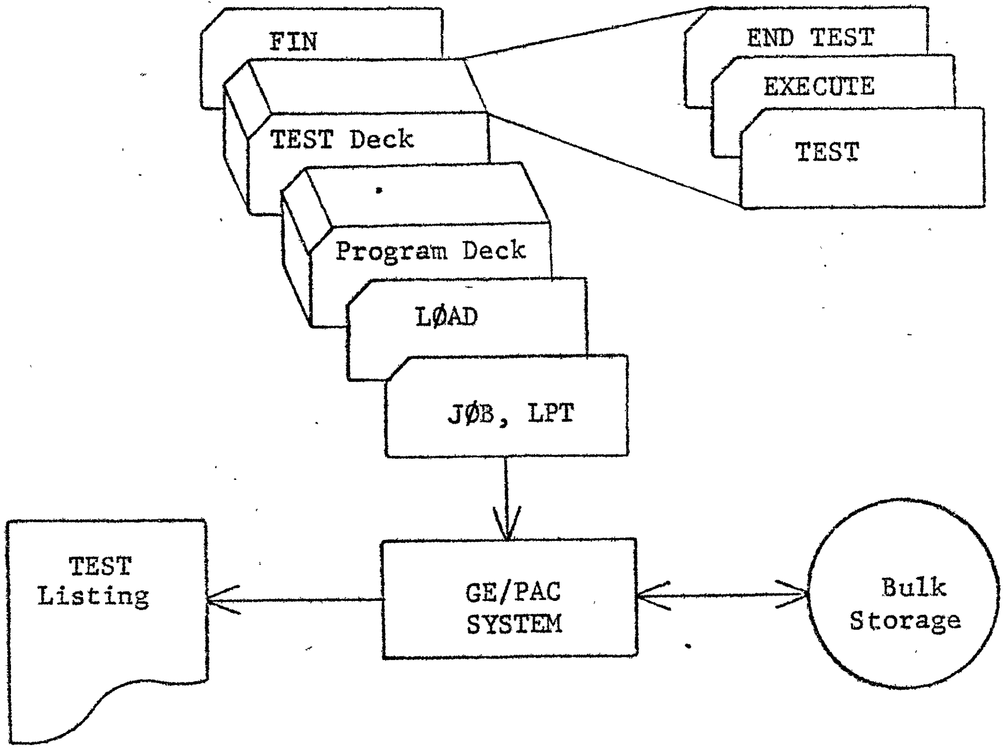
Since no time is specified, the maximum real-time allowed for the test is one minute.

In the example below (Example 28), the EXECUTE Card is shown in use with other cards and a program deck.

EXAMPLE 28:

```
JØB, LPT, 9/26, LEWIS W.
LØAD,ØPTIMUM CØNTRØL, KEEP
         .
         .  } Program deck
         .
TEST,ØPTIMUM CØNTRØL
EXECUTE, 2 MINUTES
END TEST
FIN
```

This coding loads and executes a program from its origin until it turns itself off, but for not longer than two minutes. If an invalid branch is encountered within the deck, the program will be turned off. Because of KEEP status, the program will remain in bulk storage after the run. The following illustration depicts Example 28:

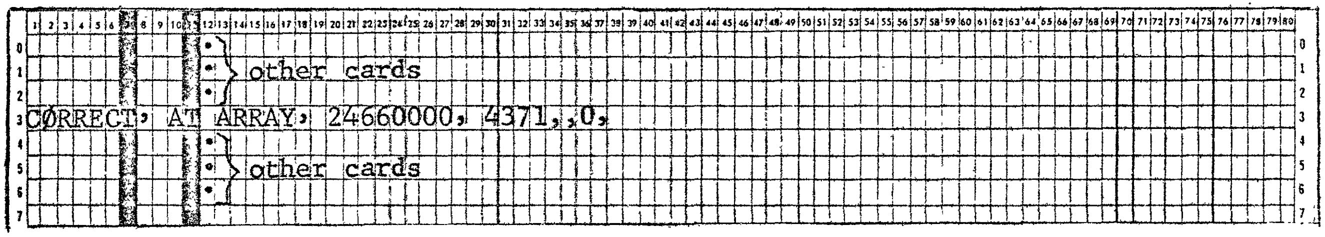


Data cards which are read by the program being tested must follow the EXECUTE Card for the segment(s) containing the READ statements.

CORRECT Card

The CORRECT Card permits octal corrections (only) to be made to a segment while it is being tested. It also permits insertion of data in a symbolic common location. Numeric addresses are assumed to be relative to the beginning of the program. Corrections are sequential, starting with the location following AT. Corrections are right-justified if less than eight digits at length, and two consecutive commas are interpreted as "skip one location".

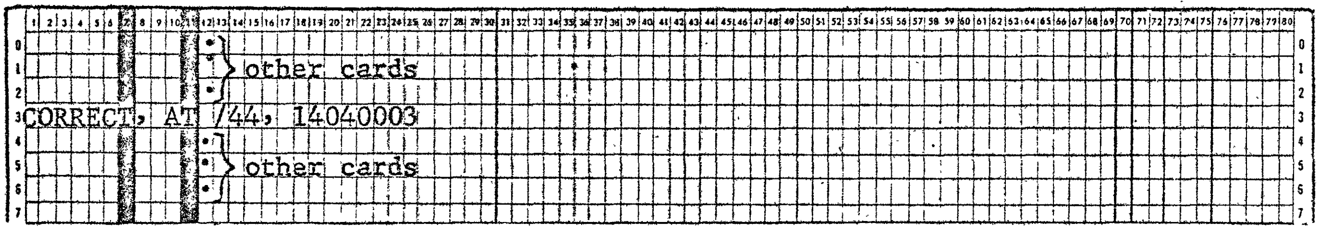
EXAMPLE 29:



```
            •
            • } other cards
            •
CØRRECT, AT ARRAY, 24660000, 4371,,0,
            •
            • } other cards
            •
```

In Example 29, there is one eight-digit correction, one correction with four leading zeros and four digits, followed by a skip and a word of zeros. The last comma can be removed without affecting the correction.

EXAMPLE 30:



```
            •
            • } other cards
            •
CORRECT, AT /44, 14040003
            •
            • } other cards
            •
```

This example changes the 44th (octal) location from the beginning of the program to the instruction "branch to here plus three locations" (BRU * + 3).

A maximum of 10 corrections and "skips" may be entered on each card. Corrections are retained only until the END TEST Card is encountered. A maximum of 15 CORRECT Cards may be entered between EXECUTE Cards.

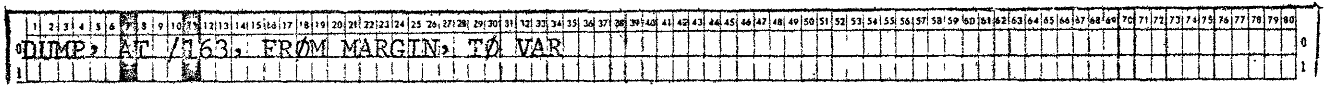
Since a program includes only instruction and data areas as compiled, the programmer may use the additional area up to 4K as needed for patches during TEST (ing).

DUMP Card

The DUMP Card displays on the printer the contents of the area specified between the locations prefixed by the prepositions FROM and TO as they existed prior to the execution of the designated instruction (AT). The format of the dump (Refer to Appendix E) is either floating- or fixed-point decimal (integer) depending on whether the FORTRAN definition of the beginning (FROM) labels is floating or fixed. If the definition is neither floating- nor fixed-point (example: $TEMP), the format is octal. The format inferred by the label may be overridden by using a stated option. Permissible options are INTEGER, REAL, and OCTAL. This allows the programmer to follow the data formats as declared at compilation time.

All DUMP requests are removed at the end of each program segment. Therefore, all dumps desired between EXECUTE Cards must be specified. When more than five DUMP Cards are entered between EXECUTE Cards, only the first five are used.

EXAMPLE 31:

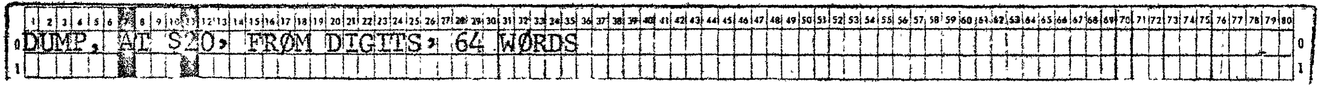


```
DUMP, AT /163, FRØM MARGIN, TØ VAR
```

Example 31 shows coding to display the contents of the locations from MARGIN to VAR inclusive when the program reaches the 163rd (octal) location from the beginning. The dump is in fixed-point format.

Instead of the preposition TO, the number of words desired may be used (per Example 32, below). When a FORTRAN statement number is to be specified as the symbolic location for any of the prepositions, the number must be preceded by a $ (dollar-sign).

EXAMPLE 32:

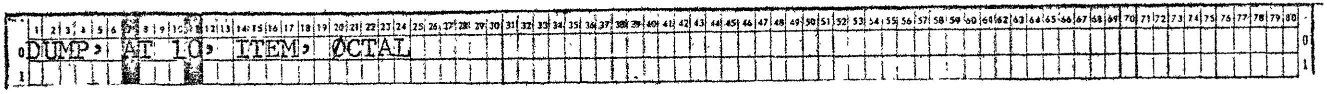


```
DUMP, AT S20, FRØM DIGITS, 64 WØRDS
```

Example 32 shows coding to display the contents of the 64-word table, DIGITS, in floating-point format, when the program reaches Statement 20.

When only one item is desired, the preposition FROM is not necessarily included (as in Example 33).

EXAMPLE 33:

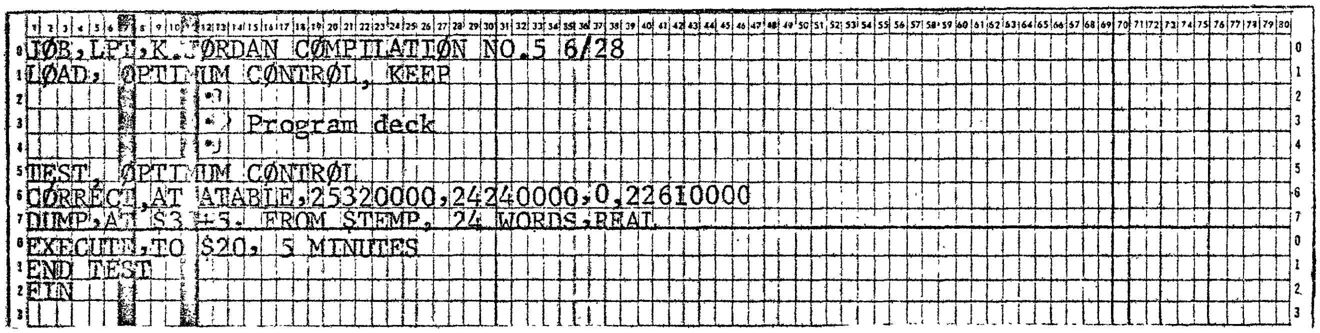


```
DUMP, AT 10, ITEM, ØCTAL
```

When the program reaches the 10th (decimal) location from the beginning, display the octal contents at the temporary storage location ITEM. (Example 33)

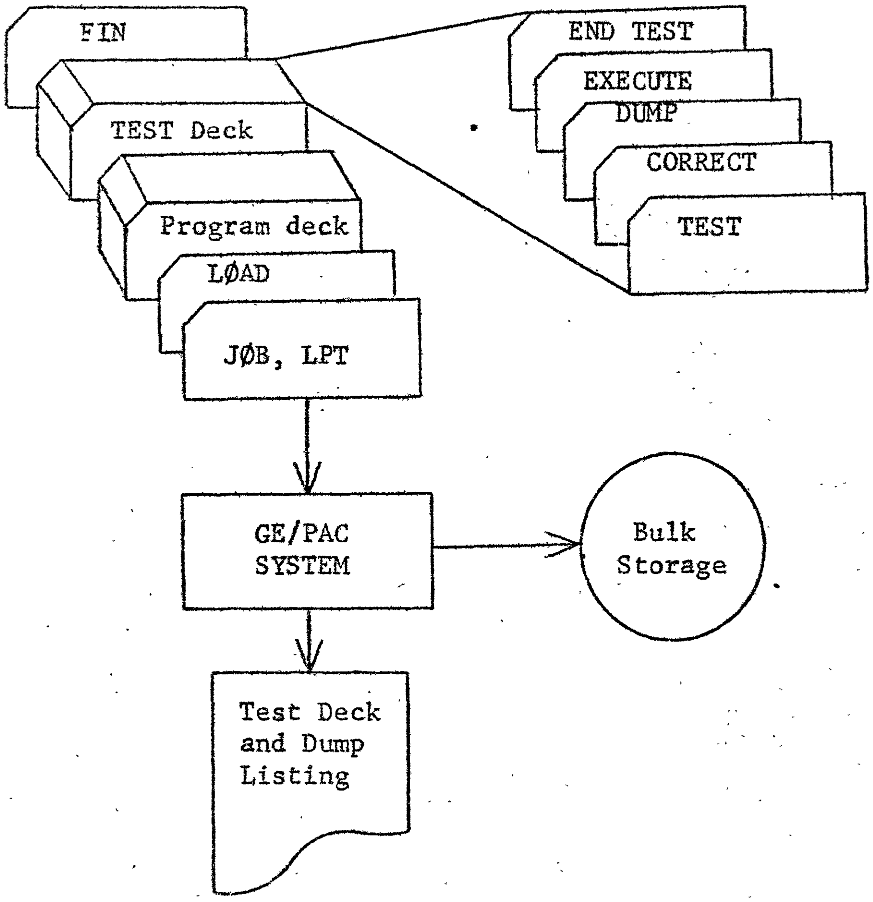
Example 34 below executes a program from its origin to Statement 20. The TEST must not run longer than five minutes. TEST data is entered into ATABLE, and the temporary storage of the program is displayed in floating-point format on the printer when the program reaches the fifth location after Statement 33.

EXAMPLE 34:



```
JØB,LPT,K.JØRDAN CØMPILATIØN NO.5 6/28
LØAD, ØPTIMUM CØNTRØL, KEEP
           •)
           •) Program deck
           •)
TEST, ØPTIMUM CØNTRØL
CØRRECT,AT ATABLE,25320000,24240000,0,22610000
DUMP,AT $33+5, FROM $TEMP, 24 WORDS,REAL
EXECUTE,TO $20, 5 MINUTES
END TEST
FIN
```

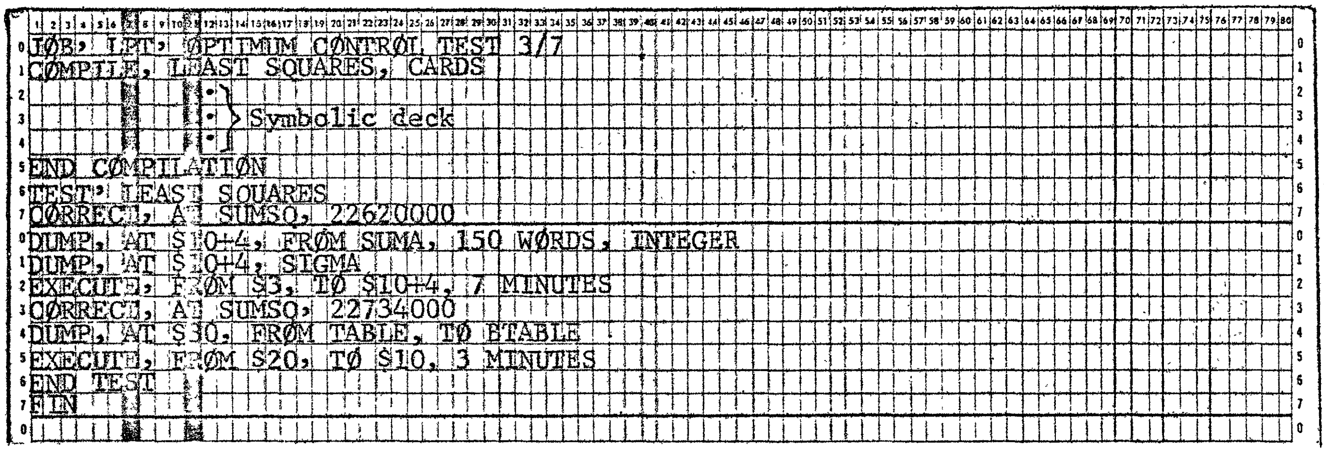
The following illustration depicts Example 34:



Example 35 ("compile and go") below shows coding that performs the following:

1. Compile the FORTRAN (symbolic) program LEAST SQUARES and punch a program deck.
2. Enter a change at the symbolic location SUMSQ.
3. Execute the program from Statement 3 to four locations after Statement 10.
4. When the program reaches the fourth location after Statement 10, display on the printer the contents of symbolic location SIGMA in floating-point (SIGMA is a FORTRAN floating-point symbol).
5. Display the contents of the 150-word table starting at SUMA.
6. Enter a new value in SUMSQ and re-run the program from Statement 20 to Statement 10.
7. When the program reaches Statement 30, display the contents of the locations from TABLE to BTABLE (inclusive) in floating-point format.
8. Upon completion, remove program LEAST SQUARES from the System.

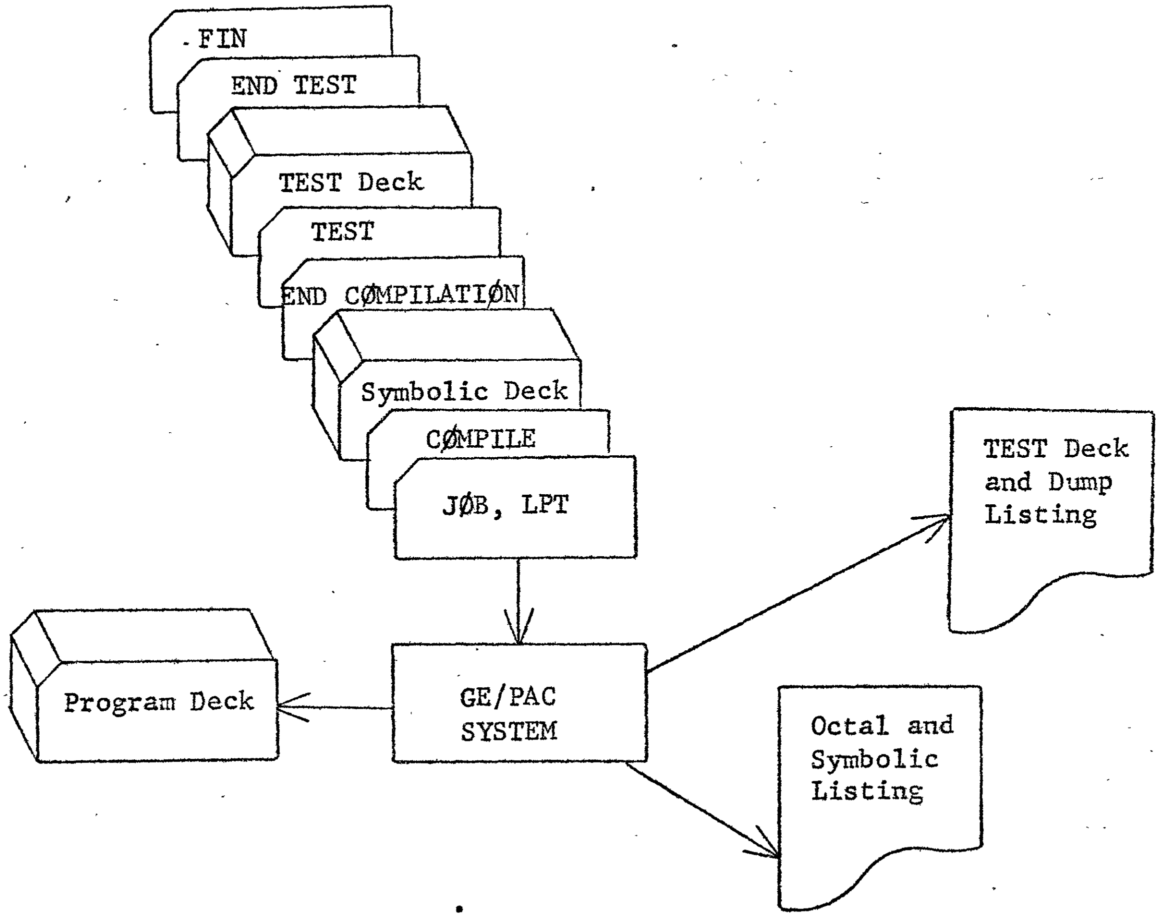
EXAMPLE 35:



```
JØB, LPT, ØPTIMUM CØNTRØL TEST 3/7
CØMPILE, LEAST SQUARES, CARDS
           •}
           • }Symbolic deck
           •}
END CØMPILATIØN
TEST, LEAST SQUARES
CØRRECT, AT SUMSQ, 22620000
DUMP, AT $10+4, FRØM SUMA, 150 WØRDS, INTEGER
DUMP, AT $10+4, SIGMA
EXECUTE, FRØM $3, TØ $10+4, 7 MINUTES
CØRRECT, AT SUMSQ, 22734000
DUMP, AT $30, FRØM TABLE, TØ BTABLE
EXECUTE, FRØM $20, TØ $10, 3 MINUTES
END TEST
FIN
```

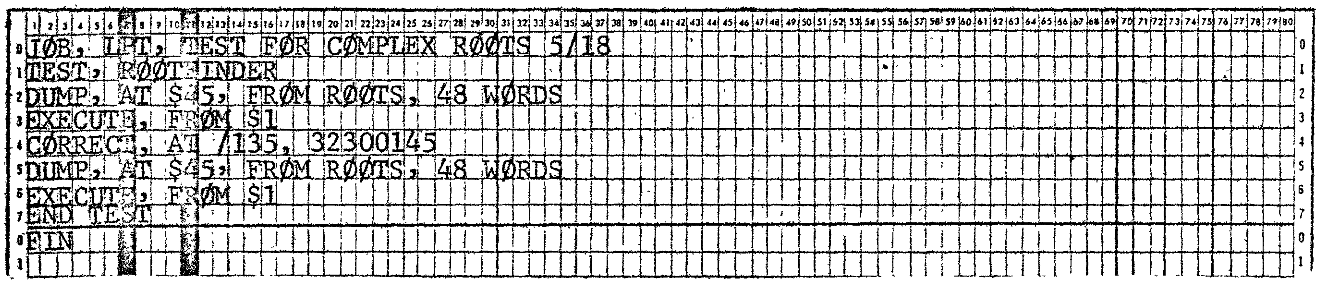
Compile and Go.

The following illustration depicts Example 35:



In Example 36, a program, ROOTFINDER, is in bulk storage in KEEP status. Execute the program from Statement 1 to the end. When the program reaches Statement 45, display the contents of the 48-word floating-point table, ROOTS. Change the contents of the 135th location from the beginning to a store instruction and repeat the test. The program will remain in bulk storage after the test.

EXAMPLE 36:



```
JØB, LPT, TEST FØR CØMPLEX RØØTS 5/18
TEST, RØØTFINDER
DUMP, AT $45, FRØM RØØTS, 48 WØRDS
EXECUTE, FRØM $1
CØRRECT, AT /135, 32300145
DUMP, AT $45, FRØM RØØTS, 48 WØRDS
EXECUTE, FRØM $1
END TEST
FIN
```

Test Where Program is in KEEP Status (Bulk Storage).

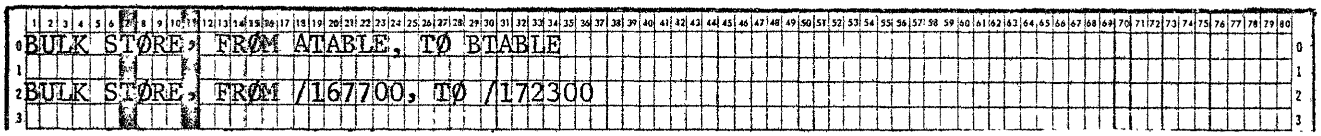
6.7.3.3 System Protection Cards

If no system protection cards are included in the deck, the program will only be allowed to store in its own area or in COMMON, and branch within its own area.

BULK STORE Cards

The BULK STORE Card allows the program being tested to transfer data from core to the designated area in bulk storage. Symbolic labels must refer to absolute bulk storage locations. Numeric addresses must be entered as octal. Bulk store parameters are effective until either a new BULK STORE Card or the END TEST Card is encountered. The system must reserve a bulk storage area in order for the programmer to know where on bulk certain locations exist. Example 37 shows coding for the BULK STORE using symbolic and numeric addresses.

EXAMPLE 37:



```
0 BULK STØRE, FRØM ATABLE, TØ BTABLE
1
2 BULK STØRE, FRØM /167700, TØ /172300
3
```

The program being executed may transfer to bulk storage only between the specified locations. Symbolic names must refer to absolute locations.

Input/Output During Testing (INPUT Card, OUTPUT Card)

These cards are used to specify which I/O devices may be used by the program being tested. They are effective until the END TEST Card is encountered.

A peripheral may have more than one identifying number within the system. Device numbers "i" represent the FORTRAN Submonitor number of the input/output device as indicated the Monitor check list. If not stated, Device "O" is assumed.

All devices of one type must be specified on one card, and no more than six devices may be placed on a card.

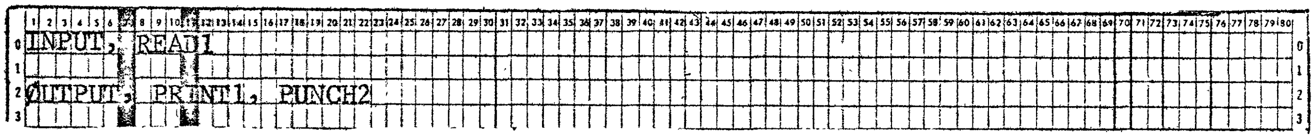
INPUT Card

This card is effective until a new INPUT Card or END TEST Card is encountered. READi specifies a card reader, paper tape reader, or input typer. ("i" represents a number assigned to each individual input device.)

OUTPUT Card

This card is effective until a new OUTPUT Card or END TEST Card is encountered. PRINTi specified an output typer or printer. PUNCHi specifies a card or paper tape punch. ("i" represents a number assigned to each individual output device.)

EXAMPLE 38:



```
INPUT, READ1

ØUTPUT, PRINT1, PUNCH2
```

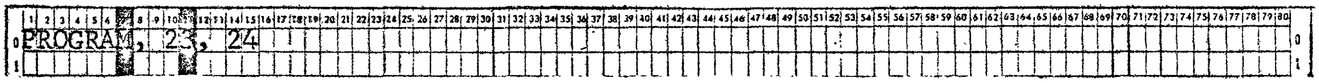
The program being executed may use only the specified input and output devices, i.e. card reader for input, printer and card punch for output.

PROGRAM Card

The PROGRAM Card tells which real-time programs may be turned on by the program being tested. Programs to be turned on are specified by their program number. A maximum of four programs are allowed to be specified per PROGRAM Card.

Programs which are considered a part of Monitor (e.g. Input, Output) may not be turned on. (See Appendix A.) Program card parameters are effective until either a new PROGRAM Card or the END TEST Card is encountered.

EXAMPLE 39:



```
PROGRAM, 23, 24
```

Initiation of Real-Time Programs Numbers 23 and 24.

Violation of any of the above system protection restrictions by the program is noted by an error printout; the card in error is ignored.

In Example 40, the program, 25, is to be tested with the following restrictions:

1. It may request transfers to bulk storage only between octal addresses 463000 and 471200 inclusive.

2. It may only make input requests through the card reader.

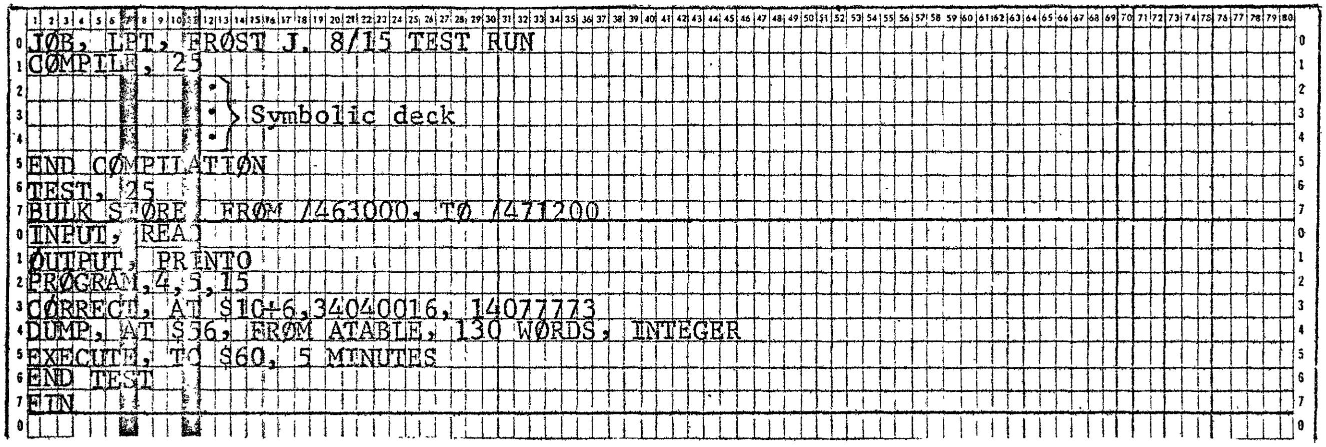
3. It may only make output requests of the printer.

4. It may initiate the execution of Real-Time Programs 4,5, and 15 only.

Branch instructions are inserted at the 6th and 7th locations after Statement 10.

When the program reaches Statement 56, the contents of 130 locations starting with ATABLE will be displayed on the printer in fixed-point (integer)format. The program will be executed from its beginning to Statement 60, with a maximum time limit of five minutes.

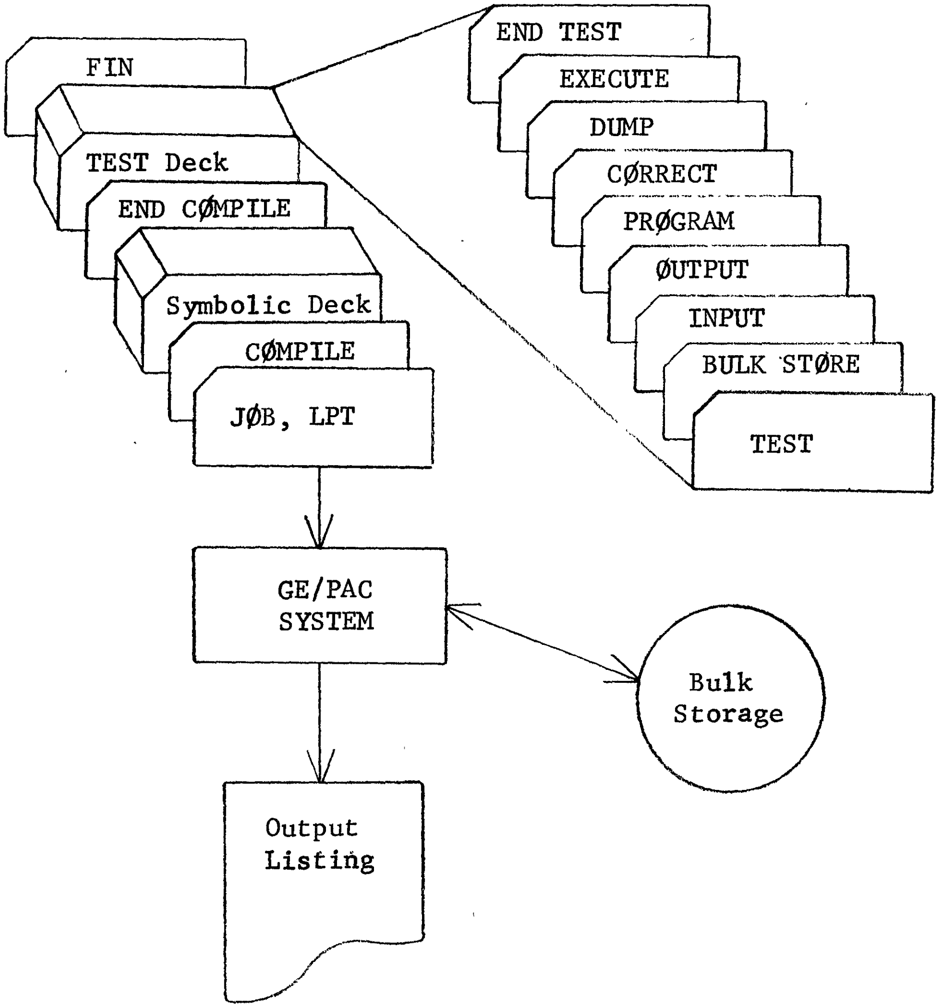
EXAMPLE 40:



```
JØB, LPT, FRØST J. 8/15 TEST RUN
CØMPILE, 25
           .
           . } Symbolic deck
           .
END CØMPILATIØN
TEST, 25
BULK STØRE, FRØM /463000, TØ /471200
INPUT, READ
ØUTPUT, PRINTO
PRØGRAM,4,5,15
CØRRECT, AT $10+6,34040016, 14077773
DUMP, AT $56, FRØM ATABLE, 130 WØRDS, INTEGER
EXECUTE, TO $60, 5 MINUTES
END TEST
FIN
```

To Compile and Test Program Number 25.

The illustration below depicts Example 40:



6.8 OPERATING INSTRUCTIONS

6.8.1 CARD HANDLING

Review all card decks to insure that they are in correct order so that an invalid run may be avoided. For instance, a COMPILE request should have:

1. JOB Card.
2. COMPILE Card.
3. FORTRAN (symbolic) deck or decks.
4. END COMPILATION Card.
5. FIN Card.

A LOAD request should have:

1. JOB Card.
2. LOAD Card.
3. Binary (program) deck.
4. FIN Card

A TEST deck should have:

1. JOB Card.
2. TEST Card.
3. CORRECT Cards (optional).
4. DUMP Cards (optional).
5. System Protection Cards (optional).
6. EXECUTE Cards.
7. END TEST Card.
8. FIN Card.

Place the desired card deck in the input hopper and press the INPUT DEMAND Button on the card reader.

6.8.2 OPERATING REQUIREMENTS

Refer to respective operating instructions furnished with each system for the card reader, printer, and card punch.

A run may be aborted at any time that the LPT System is not reading cards by pressing the INPUT DEMAND Button on the card reader. This action simulates failure of the card reader, and the job is terminated.

If LPT is reading cards, turn the card reader off. This simulates peripheral failure, and action should be taken according to Paragraph 6.8.3.

Other free-time programs can be aborted in the same manner by checking the input demand flag, or by peripheral failure on an input request.

6.8.3 PERIPHERAL FAILURE

When there is a malfunction on any of the peripheral equipment, Corrective Action Diagnostic types an appropriate error message.

The operator notifies the system if the device will be repaired. If a repair is not possible, the Free-Time System turns itself off and the job must be restarted after the repair has been made.

APPENDIX A

RESTRICTIONS

1. FREE-TIME SYSTEM RESTRICTIONS

   A. Program or symbol names should not begin with the following identifiers:

      1) KEEP
      2) PERMAN (ENT)
      3) FORTRA (N)
      4) SYMBOL (S)
      5) CARDS
      6) LPT
      7) AT
      8) TO
      9) FROM
      10) REAL
      11) INTEGE (R)
      12) OCTAL
      13) DIAGNO (STIC)

   Appendix D contains a list of all symbols preloaded in the System EQL Table.

   B. The first six letters of any two program or symbol names should not be the same.

   C. The following Monitor functional programs may not be called using the JOB Control Card or by a Turn Program On request:

      1) Output Program.
      2) Input Program.
      3) Corrective Action Diagnostic.
      4) Free-Time Executive Program.
      5) Operator Routines.

2. GE/PAC PROCESS FORTRAN RESTRICTIONS

   A. Since all bulk allocation is accomplished by the system, the statements BEGIN PROGRAM AT and SEGMENT are not allowed.

   B. Normally, the compiler generates the data areas as referenced by the FORTRAN statements. However, certain labels for variables are reserved for the system. Reference to these labels in a FORTRAN statement causes the compiled program to access these system variables. They are treated as if they had been DEFINED (refer to FORTRAN REFERENCE MANUAL). A list of these labels is included in Appendix D.

3. PAL LANGUAGE RESTRICTIONS.

   A. Since both bulk and core storage allocation is done by the system, the ORG and DCW pseudo-operations are not allowed by the assembler. If used, they are flagged as illegal operations and are ignored. Each program is assembled relative to Location 0.

   B. When the op-code and/or operand written by the programmer causes the assembler to generate an illegal operation, or if the assembler cannot understand the instruction as written, the assembler replaces the offending instruction with one which has no effect (BRU*+1).

APPENDIX B

LIST OF CONTROL CARDS CONTAINING EXAMPLES OF ALL OPTIONS

GENERAL ELECTRIC
PROCESS COMPUTER BUSINESS SECTION
PHOENIX, ARIZONA

GE/PAC* LANGUAGE STATEMENT CODING FORM

Project Name
Program Name
Page of Date
Programmer

PROCESSOR KEYS
0-DELETE
6-PAL
7-FORTRAN

LOCATION · OP CODE · OPERAND · KEY · Proj. # · Prog. # · Sequence #

```
BULK STØRE, FRØM /135260, TØ DRUMAD

CØMPILE
CØMPILE, TRAJECTØRY
CØMPILE, CURVE FIT, KEEP, CARDS
CØMPILE, 23, PERMANENT, SYMBØLS, NØ LIST
CØMPILE, SQUARE RØØT, FØRTRAN, CARDS
CØMPILE, DIAGNØSTIC
CØMPILE, DIAGNØSTIC, NØ LIST

CØRRECT, AT DØG+5, 2451, 35030011, 0, 22,, 44020106
CØRRECT, AT /64, 14040003

DUMP, ITEM, AT 10, ØCTAL
DUMP, AT $16, FRØM /55, TØ STEMP, ØCTAL
DUMP, AT 25, FRØM DØG, TØ CAT, INTEGER
DUMP, AT $10, FRØM /162, TØ $30, REAL

END CØMPILATIØN
END TEST

EXECUTE
EXECUTE, FRØM START, TØ /240, 10 MINUTES
EXECUTE, FRØM 10, TØ /531
EXECUTE, TØ $12, 5 MINUTES
EXECUTE, FRØM $20

FIN
FIN, CATALØG

INPUT, READ

JØB, LPT, LIST ØF CØNTRØL CARDS WITH ØPTIØNS
JØB, L5

LØAD
LØAD, TRAJECTØRY
LØAD, 27, KEEP
LØAD, LEAST SQUARES, PERMANENT
LØAD, 27, PERMANENT
LØAD, INPUT, FØRTRAN
LØAD, SYMBØLS

ØUTPUT, PRINT1, PUNCH1

PRØGRAM, 2, 31, 3

REMØVE, MILL SETUP, KEEP
REMØVE, LEAST SQUARES, PERMANENT
REMØVE, 41, PERMANENT
REMØVE, SØRTF, FØRTRAN
REMØVE, SYMBØLS

TEST
TEST, TRAJECTØRY
TEST, 14
```

APPENDIX C

ERROR MESSAGES

1. SYSTEM PROTECTION ERRORS

Messages are printed when an error is located within a program which violates the system protection rules as defined by the TEST System protection cards.

When an invalid instruction is found, an error message, "ILLEGAL XXXXX, relative address, instruction, contents of the Index Register or Device or 0," is printed. XXXXX may be a STORE INST, BRANCH INST, IN/OUT INST, MTR ENTRY, BULK TRANS, OOM REQUEST, RPT REQUEST, INPUT REQUEST, OUTPUT REQUEST, or DUMP ADDRESS.

EXAMPLES:

```
A.  ILLEGAL STORE INST      00076 32340174 00000012
B.  ILLEGAL IN/OUT INST     00043 25430000 00002400
C.  ILLEGAL BRANCH INST     00042 14040163 00000000
```

An additional error message, ILLEGAL BRANCH - OUT OF BOUNDS TO / (absolute address), is printed when an attempt is made to branch above the maximum location allowed or below the minimum location allowed.

2. PROGRAMMING ERRORS

MESSAGE	CAUSE	ACTION TAKEN
ILLEGAL CARD FIELD	a. No program I.D. on JOB Card.	Card is skipped.
	b. Program not in library.	Card is skipped.
	c. Undefined field on a card.	Card is skipped.
	d. Attempt to load or compile an already-existant program.	Program is not loaded.
	e. More than one status on COMPILE or LOAD Card.	Program is not loaded.
	f. Non-existant label on a TEST Control Card.	Card is skipped.
	g. Alpha, or non-octal digit, in the CORRECT Card.	Card is skipped.
	h. More than four program numbers specified on a PROGRAM Card.	Excess are ignored.
ILLEGAL CONTROL CARD	a Missing field on a control card	Card is skipped.
	b. Too many of a particular type card - DUMP or CORRECT, for example.	Card is skipped.
	c. JOB Card is read, not following a FIN Card.	The JOB following this card is lost. Must be reloaded in card reader.

APPENDIX C - contd

MESSAGE	CAUSE	ACTION TAKEN
NOT ENOUGH BULK	The scratch area remaining is too small to contain the present: a. Compilation, or b. Load.	Program is not compiled or loaded.
ILLEGAL LIB REQUEST	a. A requested subroutine did not follow the program and was not in the FORTRAN library.	a. Subroutine cannot be appended to program. b. Program not included in any library.
	b. More than 100 LIB requests	Excess are ignored.
	c. Program being compiled for FORTRAN library has a subroutine.	Program not included in any library.
TOO MANY SYMBOLS	a. More than 640 references in this program.	Excess not included.
	b. More than 1088 common symbols.	System symbol table is not updated
NO SYSTEM SYMBOLS	A REMOVE, SYMBOLS was not followed by a LOAD, SYMBOLS.	Program cannot be assembled.
PROGRAM TURNED ON	A request to remove a real-time program found the program running.	Request ignored.

APPENDIX D

LIST OF SYSTEM EQLS

The following labels are preloaded into the system EQL table:

	TYPE OF	
LABEL	ENTRY	VALUE OR MEANING
CRDRDR	EQL	Priority Number of Card Reader
PRTR	EQL	Priority Number of Printer
SECND	EQL	Number of Time Counts in One Second
CTYPER	EQL	Priority Number of the Console Typer
CPUNCH	EQL	Priority Number of the Card Punch
FTSK01	EQL	Disc Address of the Free-Time SAVE Area
ECPC01	BRU	MONITOR BRANCH VECTOR
MAP01	BRU	See appropriate Monitor subroutine write-up for usage.
MAP02	BRU	
MAP03	BRU	
MAP04	BRU	
SRGC01	BRU	
SRGC02	BRU	
RRGC01	BRU	
RRGC02	BRU	
FRPC01	BRU	
GADC01	BRU	
RMRC01	BRU	
PAVC02	BRU	
FMRC01	BRU	
RMPC01	BRU	
ØUTC01	BRU	
ØUTC02	BRU	
ØUTC03	BRU	
PAVC03	BRU	
PAVC04	BRU	
INPC01	BRU	
INPC02	BRU	
DELC01	BRU	
ØFFC01	BRU	
TPNC01	BRU	
TPNC02	BRU	
PAVC01	BRU	
DTRC01	BRU	
DTRC02	BRU	
INC01	BRU	
PNHC01	BRU	
DRLC01	BRU	
RRLC01	BRU	
DLRC01	BRU	Data Link Branch Vector
DLRC02	BRU	

APPENDIX D - contd

LABEL	TYPE OF ENTRY	VALUE OR MEANING
E01F19	BRU	Convert A Card Image Subroutine
E02F19	BRU	Convert One Column Subroutine
$INPUT	BRU	Submonitor
$ØTPUT	BRU	Branch Vector
$DLDAT	BRU	
$DLLØC	BRU	
$DLHLT	BRU	
$READ	BRU	
$PRINT	BRU	
$PUNCH	BRU	
$INTRP	BRU	
$DMYSU	BRU	
$ØUTRP	BRU	
$BLKØT	BRU	
$BLKIN	BRU	
$BLKTP	BRU	
$IØTST	BRU	
$HLTIN	BRU	
$BLKHL	BRU	
$HLØUT	BRU	
$CGØTØ	BRU	
$AGØTØ	BRU	
QSTRAP	BRU	System Trap - BRU*
ZERØ	CØN D	Table of Integer Constants
ØNE	CØN D	
TWØ	CØN D	
THREE	CØN D	
FØUR	CØN D	
FIVE	CØN D	
SIX	CØN D	
SEVEN	CØN D	
EIGHT	CØN D	
NINE	CØN D	
TEN	CØN D	
IZERØ	CØN D	Table of Integer Constants for FORTRAN
IØNE	CØN D	
ITWØ	CØN D	
ITHREE	CØN D	
IFØUR	CØN D	
IFIVE	CØN D	
ISIX	CØN D	
ISEVEN	CØN D	
IEIGHT	CØN D	
ININE	CØN D	
ITEN	CØN D	

LABEL	TYPE OF ENTRY	VALUE OR MEANING
TENX0	CØN F	Table of Floating-Point. Powers of Ten.
TENX1	CØN F	
TENX2	CØN F	
TENX3	CØN F	
TENX4	CØN F	
TENX5	CØN F	
TENX6	CØN F	
TENX7	CØN F	
TENX8	CØN F	
TENX9	CØN F	
MSKR3	CØN Ø	Table of Right-Bit Masks. Number Denotes Bit Position of Highest Bit.
MSKR5	CØN Ø	
MSKR6	CØN Ø	
MSKR7	CØN Ø	
MSKR8	CØN Ø	
MSKR11	CØN Ø	
MSKR13	CØN Ø	
MSKR14	CØN Ø	
MSKR15	CØN Ø	
MSKR16	CØN Ø	
MSKR17	CØN Ø	
MSKL5	CØN Ø	Table of Left-Bit Masks. Number Denotes Number of Left-Bits In Mask.
MSKL9	CØN Ø	
MSKL18	CØN Ø	
D24	CØN D	Decimal 24
TIME	CØN	Time In Counts Since Midnight.
ITIME	CØN	Time In Counts Since Midnight (FORTRAN).
CØUNT	CØN	Diagnostic Count Table.
ICØUNT	CØN	Diagnostic Count Table (FORTRAN).
PRØGNØ	BSS	Running Program Number - Times 8.
PRIØNØ	BSS	Running Program Number - Times 1.
DMCRNØ	BSS	Running Program Number - Times 3.
PRØG	BSS	Table of Next Execution Times For All Real-Time Programs.
IPRØG	BSS	Table of Next Execution Times For All Real-Time Programs (FORTRAN).
AUXTM	BSS	Table of Auxiliary Time Counts.
LAUXTM	BSS	Table of Auxiliary Time Counts (FORTRAN).
LCNT	BSS	Number of Current Lines To Be Printed.
LPAGE	CØN	Number of Lines To Be Printed Per Page.
PAGENØ	BSS	Current Page Number.
IPAGNØ	BSS	Current Page Number (FORTRAN).
VCNT	BSS	Current Vertical Page Control.
PTHB1	BSS	Current Heading.
ØFFØ1	EQL	Free-Time Turn-Off Flag.

APPENDIX D - contd

LABEL	TYPE OF ENTRY	VALUE OR MEANING
ALTFLG	BSS	Alternate Device Availability.
XFER	BSS	Number of Bulk Transfers Waiting For Each Program.
DRMLØC	BSS	Three-Word Pointer For Each Real-Time Program.
SIZE	BSS	
CØRLØC	BSS	
SAVTBL	BSS	Table of Pointers for SAVE Status.
SECDAY	CØN	Number of Time Counts In 24 Hours.
ISECDY	CØN	Number of Time Counts In 24 Hours (FORTRAN).
LKØUT	CØN Ø	Constant Which Is Used To Lock-Out A Program.
DERO1	EQL	Derelativize An Address Subroutine.
DMDFLG	BSS	Flag Word For Demands From Various Demand Buttons
DMD2	BSS	Flag Word For Second Demands.
RLIST1	BSS	Input Buffer For Card Reader.
INPTB1	EQL	Punch Output Buffer.
CØMMØN	EQL	Start of FORTRAN Common Storage Area.

APPENDIX E

DUMP FORMAT

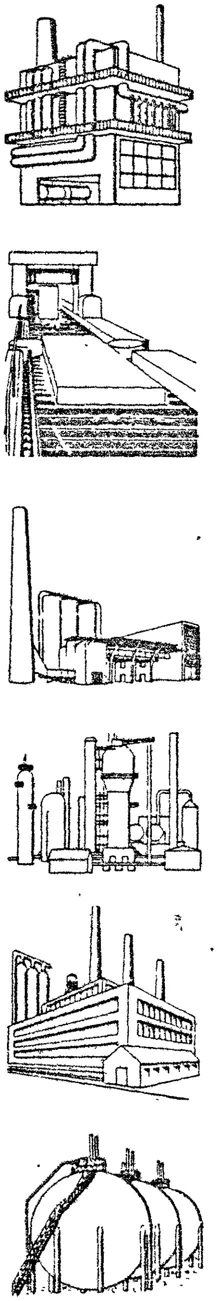
This coding will produce the dump which follows:

```
Where Ø = Octal
      X = Decimal
      E = Exponent
      0 = Zero
```

```
DUMP, AT WØRK, FRØM VALUE, 15 WØRDS
DUMP, AT WØRK, FRØM LABEL, TØ NEXT
DUMP, AT WØRK, $TEMP, 20 WØRDS, ØCTAL
```

```
DUMP AT WØRK

    A          Q          P          I          X2         X3         X4         X5         X6         X7
ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ

FROM VALUE
0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX
0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX 0.XXXXXE XX

FROM LABEL
  XXXXX      XXXXX      XXXXX      XXXXX      XXXXX      XXXXX      XXXXX      XXXXX      XXXXX      XXXXX
  XXXXX      XXXXX      XXXXX      XXXXX      XXXXX      XXXXX1

FROM $TEMP
ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ
ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ
ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ   ØØØØØØØØ
```

[1] = NEXT

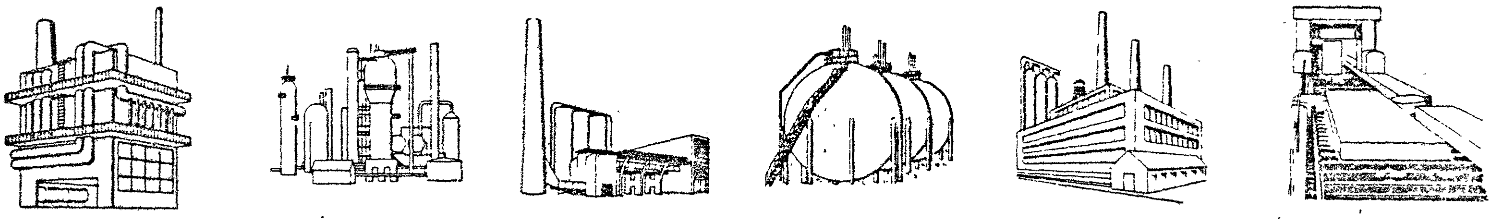


GE-PAC® 4020

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER

MONITOR MANUAL

PRELIMINARY



PROCESS COMPUTER
BUSINESS SECTION
PHOENIX, ARIZONA

GENERAL ELECTRIC

Library File No. YPG53M

Revised January 1967

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. Box 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

CONTENTS

*Revised 12/66

CONTENTS

INTRODUCTION

The objective of the MONITOR MANUAL is to present various ways of using the Monitor System in conjunction with GE/PAC* 4020 Process Automation Computers. Use of this manual is predicated upon, and presupposes the reader's familiarity with, real-time system requirements, GE/PAC programming techniques, and Process Assembler Language.

Designed especially for the user, the manual explains what Monitor does and what the programmer must do to insure its successful operation. The publication also discusses individual subprograms which comprise the Monitor System, and includes the communications links which must be provided in the functional programs to allow necessary exchanges of data between Monitor and the outside world. A detailed analysis of Monitor logic is not included in this manual; rather, it is covered in the various Monitor program write-ups which are available through the Programming Library.

Examples used throughout the text showing various calling sequences are not to be construed as being the only method of using such calls. It would take more space than is practicable to allow in such a text to show all the possibilities of each subroutine call.

*Registered Trademark of the General Electric Company

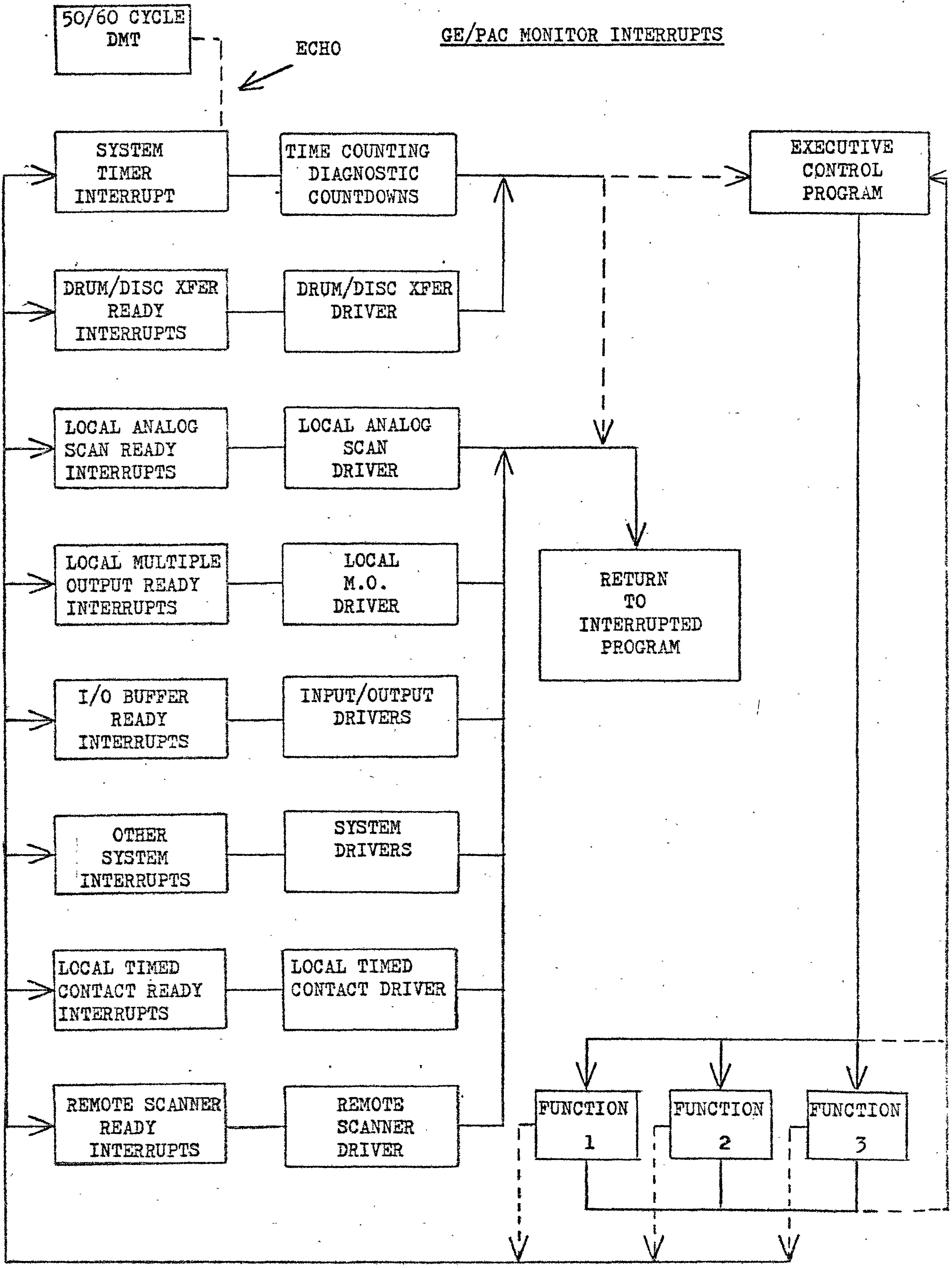
GENERAL

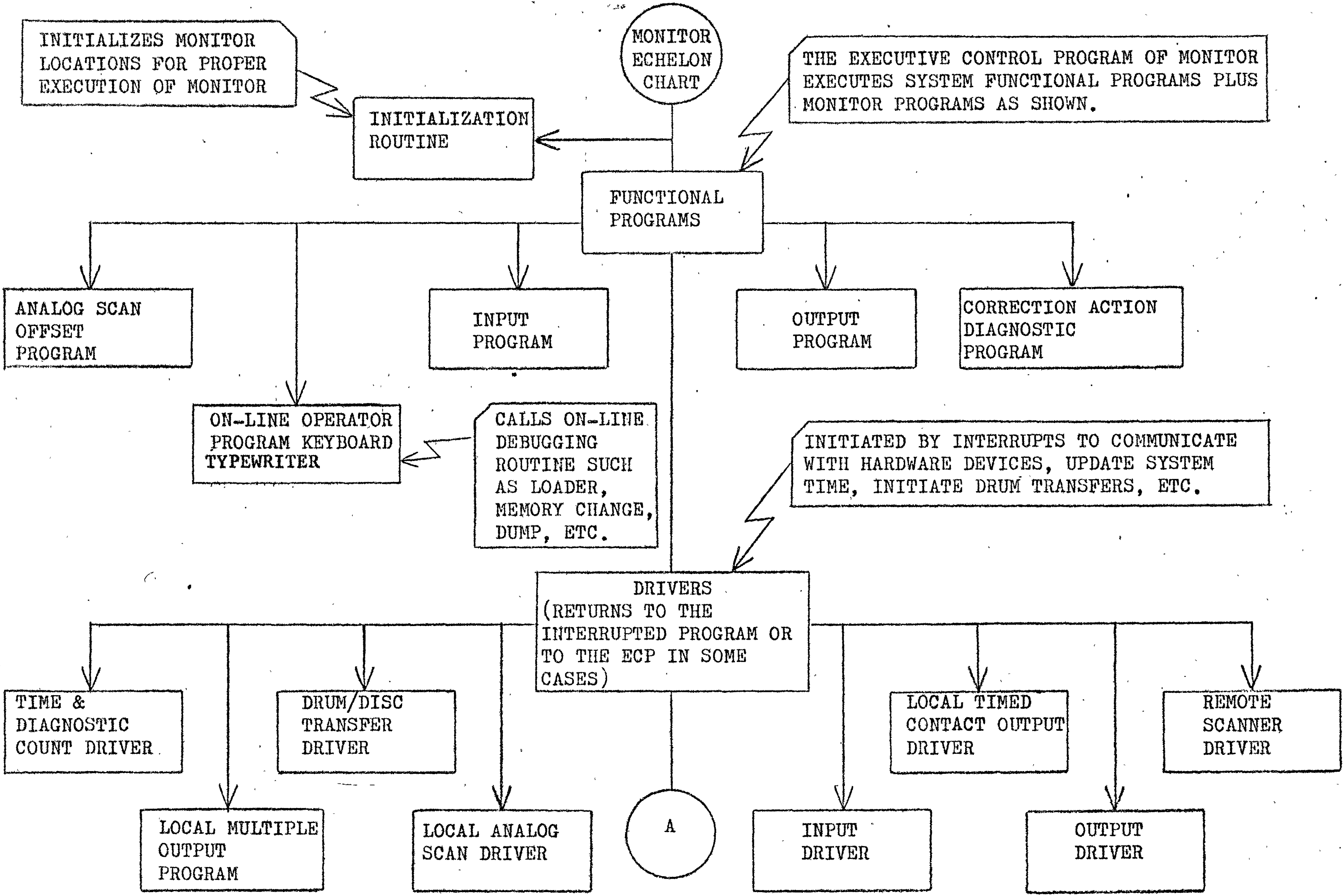
A real-time process is characterized by the occurrence of many events, some continuous, others random in nature. Events may occur simultaneously. A digital computer, however, is a serial device; that is, it performs its program options serially, one by one. Therefore, the matching of the digital computer and a real-time process requires a control system which coordinates the requirements and characteristics of both.

A GE/PAC Monitor is an operating system composed of a library of subprograms which provide the basis for a process computer system. It is the framework to which the specific functional programs are added. The Monitor accomplishes the timing and scheduling operations, input/output, internal data transfer, timed contact, multiple output, corrective action, initialization, etc. Many options are available which may or may not be included in a tailored Monitor System.

The user requests a tailored Monitor by checking the desired options on the Monitor Checklist, a copy of which is included in this manual (see Appendix D). Additional copies of the Checklist may be obtained upon request from the Programming Library.

GE/PAC MONITOR INTERRUPTS


50/60 CYCLE DMT
ECHO
SYSTEM TIMER INTERRUPT
TIME COUNTING DIAGNOSTIC COUNTDOWNS
EXECUTIVE CONTROL PROGRAM
DRUM/DISC XFER READY INTERRUPTS
DRUM/DISC XFER DRIVER
LOCAL ANALOG SCAN READY INTERRUPTS
LOCAL ANALOG SCAN DRIVER
RETURN TO INTERRUPTED PROGRAM
LOCAL MULTIPLE OUTPUT READY INTERRUPTS
LOCAL M.O. DRIVER
I/O BUFFER READY INTERRUPTS
INPUT/OUTPUT DRIVERS
OTHER SYSTEM INTERRUPTS
SYSTEM DRIVERS
LOCAL TIMED CONTACT READY INTERRUPTS
LOCAL TIMED CONTACT DRIVER
REMOTE SCANNER READY INTERRUPTS
REMOTE SCANNER DRIVER
FUNCTION 1
FUNCTION 2
FUNCTION 3


MONITOR ECHELON CHART
INITIALIZATION ROUTINE
FUNCTIONAL PROGRAMS
ANALOG SCAN OFFSET PROGRAM
INPUT PROGRAM
OUTPUT PROGRAM
CORRECTION ACTION DIAGNOSTIC PROGRAM
ON-LINE OPERATOR PROGRAM KEYBOARD TYPEWRITER
DRIVERS (RETURNS TO THE INTERRUPTED PROGRAM OR TO THE ECP IN SOME CASES)
TIME & DIAGNOSTIC COUNT DRIVER
DRUM/DISC TRANSFER DRIVER
LOCAL TIMED CONTACT OUTPUT DRIVER
REMOTE SCANNER DRIVER
LOCAL MULTIPLE OUTPUT PROGRAM
LOCAL ANALOG SCAN DRIVER
A
INPUT DRIVER
OUTPUT DRIVER

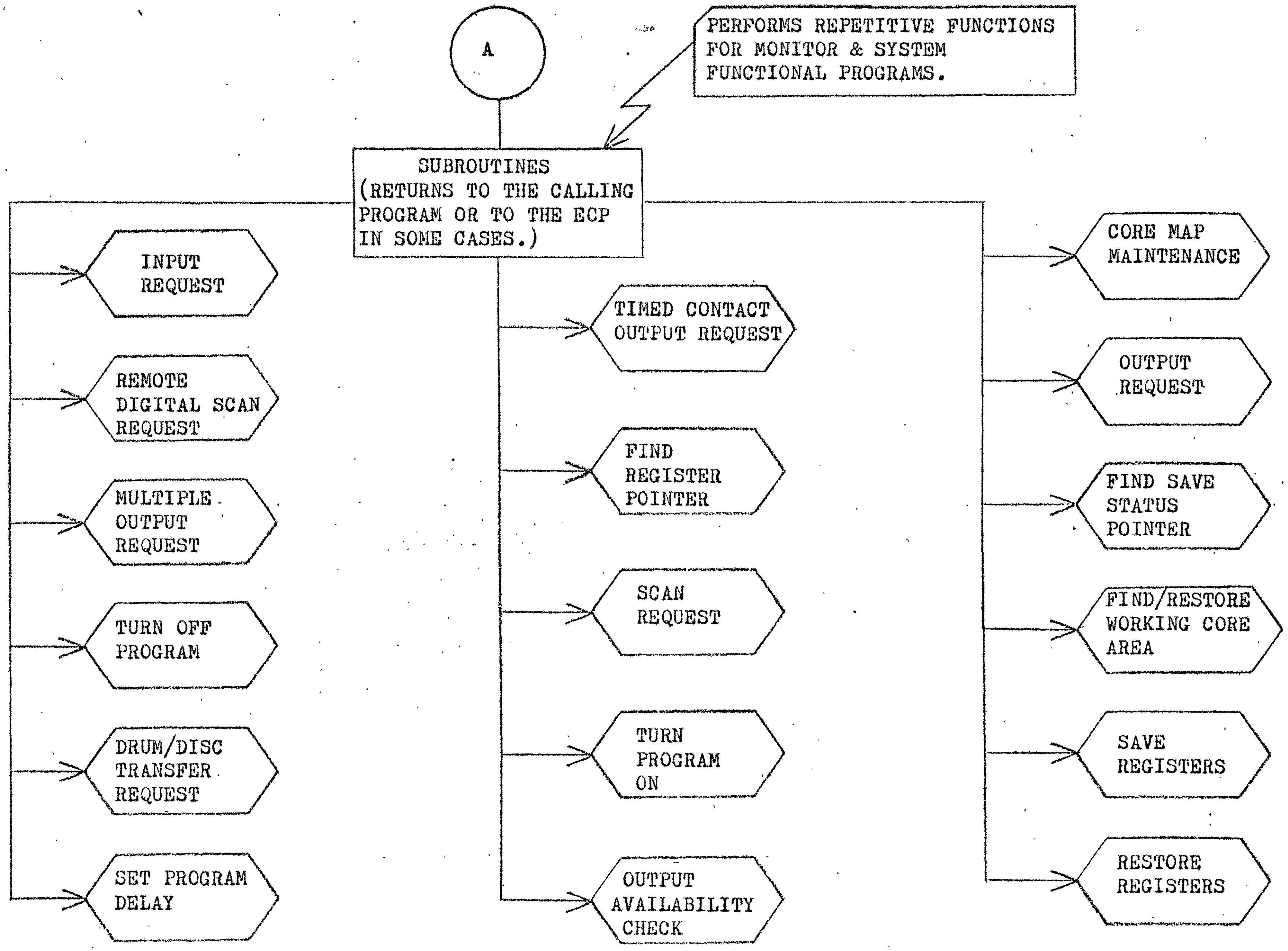
INITIALIZES MONITOR LOCATIONS FOR PROPER EXECUTION OF MONITOR

THE EXECUTIVE CONTROL PROGRAM OF MONITOR EXECUTES SYSTEM FUNCTIONAL PROGRAMS PLUS MONITOR PROGRAMS AS SHOWN.

CALLS ON-LINE DEBUGGING ROUTINE SUCH AS LOADER, MEMORY CHANGE, DUMP, ETC.

INITIATED BY INTERRUPTS TO COMMUNICATE WITH HARDWARE DEVICES, UPDATE SYSTEM TIME, INITIATE DRUM TRANSFERS, ETC.

contd on page xi


A
PERFORMS REPETITIVE FUNCTIONS FOR MONITOR & SYSTEM FUNCTIONAL PROGRAMS.
SUBROUTINES (RETURNS TO THE CALLING PROGRAM OR TO THE ECP IN SOME CASES.)
INPUT REQUEST
REMOTE DIGITAL SCAN REQUEST
MULTIPLE OUTPUT REQUEST
TURN OFF PROGRAM
DRUM/DISC TRANSFER REQUEST
SET PROGRAM DELAY
TIMED CONTACT OUTPUT REQUEST
FIND REGISTER POINTER
SCAN REQUEST
TURN PROGRAM ON
OUTPUT AVAILABILITY CHECK
CORE MAP MAINTENANCE
OUTPUT REQUEST
FIND SAVE STATUS POINTER
FIND/RESTORE WORKING CORE AREA
SAVE REGISTERS
RESTORE REGISTERS

1. SYSTEM DESCRIPTION

Monitor consists of the component programs described below.

1.1 EXECUTIVE CONTROL PROGRAM (ECP)

The Executive Control Program runs permitted and schedules the execution of programs based on priority, execution time, and core availability.

System programs are executed in priority order by comparing the programs' next execution time (PRØG$_n$) with the current time (TIME). The highest priority program for which the execution time is equal to or less than the current time is executed. When the execution time is current for a program, the ECP requests a transfer from drum or disc to core providing:

1. The program is not in core.
2. The program is not presently being transferred.
3. A core area is available.

After the transfer has been completed, the program is initiated. If there is no available core area for that program, ECP tests the execution time for the next lower priority program.

There are two levels of priority program search. The first allows the system program to indicate that the program X should be run next. The ECP would initiate the program if in core, or initiate the transfer into core, if there is room. This search is initiated by a TPNC03 call.

The second Program Priority search beginning level is at the top of the program list which will result from one of the following actions:

MØD (Multiple Output Distributor - Alternate Return Only)
TCØ (Timed Contact Output - Alternate Return Only)
ITC (Interrupt Time Counter Interrupt)
DTD (Drum/Disc Transfer Interrupts)
SND (Scan Complete Interrupts)

If a program has a "turned off" or "locked out" code in its PRØG location, it is not executed until a time for execution is assigned.

There are three classifications of register storage for functional programs. They are:

1. Programs which have no register storage of their own.
2. Programs which have their own 8-word block of register storage.
3. Programs which share an 8-word block of register storage with other functional programs.

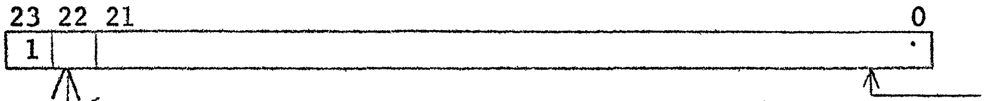
The current ECP priority order is defined in the PRGTBL Table. The latter is included in the Monitor ØFile tape and allows the programmer to change the priority of a program after the initial Monitor Assembly.

NOTE: Any direct branches to the ECP must be done in the inhibited state. The RSX Table is eliminated when all system programs have their own 8-word register storage block.

REGISTER POINTER TABLE (RSX)

This table contains an index to the 8-word storage block for programs having full register storage or sharing storage. For programs with no register storage, the table contains the flip-flop status and the program's next entry location.

A. Format for Shared or Single 8-Word Register Storage

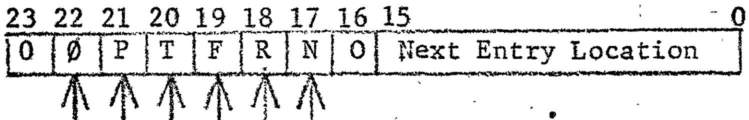


8-Word Register Block Index

0 = Return to ECP after ITC Timer or Drum/Disc Transfer Complete Interrupts.

1 = Return to Program after ITC Timer or Drum/Disc Transfer Complete Interrupts.

B. Format for No Register Storage



N = 1: Negative Relative Entry Address (Bits 0-15)
0: Positive Relative Entry Address (Bits 0-15)
For an all-core system, N is always zero.

R = 1: Absolute Permanent Core Location
0: Relative Address (Drum/Core)
For an all-core system, R is always zero.

F = 1: Set Memory Fence; 0: Reset

T = 1: Set Test Flip-Flop; 0: Reset

P = 1: Set Permit Interrupt; 0: Reset

Ø = 1: Set Overflow; 0: Reset

NOTE: Each program will have one word, in numerical order, in the RSX Table beginning with ECP (Program Number zero).

Examples:

		23	22	21	20	19	18	17	16	15	0	
*	RSX	1	0								0	ECP
*		0	Ø	P	T	F	R	N	0	NEXT ENTRY LOCATION		Program Number 1
		1	0								1	Program Number 2
		1	1								1	Program Number 3
		1	0								2	Program Number 4
*		0	Ø	P	T	F	R	N	0	NEXT ENTRY LOCATION		Program Number 5

Register Storage Table (REGSTG) stored in permanent core:

	23	22	21	20	19	18	17	16	0	
AREG	REGISTER CONTENTS AT NEXT ENTRY									First 8-word Register Storage Block (ECP)
QREG	REGISTER CONTENTS AT NEXT ENTRY									
PREG	0	Ø	P	T	F	R	N	NEXT ENTRY ADDRESS		
X3REG										
X4REG										
X5REG										
X6REG										
X7REG										
										Second 8-word Register Storage Block Programs 2 and 3
										Third 8-word Register Storage Block Program 4

*Revised 12/66

The other tables assisting the ECP in performing its functions are:

PRØG - PROGRAM EXECUTION TIME TABLE

TIME [00001433] (SYSTEM TIME COUNTS)

Program #			
1	PRØG	40000000	PROGRAM OFF
2		00001342	PROGRAM CURRENTLY RUNNING
3		00001465	PROGRAM DELAYED
4		40000001	LOCKED OUT
5		40000000	OFF
6		40000000	OFF
7		00000000	PROGRAM ON

DRUM/DISC TRANSFER CONTROL TABLE

<table>
<tr><td></td><td></td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17 15</td><td>14</td><td>13 0</td></tr>
<tr><td rowspan="4">PROGRAM No. n Transfer Group</td><td>DRMLØC</td><td>0</td><td colspan="8">BEGINNING DRUM/DISC ADDRESS</td></tr>
<tr><td>SIZE</td><td>A</td><td>C</td><td>T</td><td>N</td><td>F</td><td>W</td><td>DRUM/DISC CONTROLLER INDEX #</td><td>0</td><td>SIZE OF PROGRAM*</td></tr>
<tr><td></td><td colspan="7">23 16</td><td colspan="2">15 0</td></tr>
<tr><td>CØRLØC</td><td colspan="7">RESERVED FOR FUTURE USE</td><td colspan="2">BEGINNING CORE ADDRESS</td></tr>
<tr><td colspan="2">DRMLØC+3(n+1)
SIZE+3(n+1)
CØRLØC+3(n+1)</td><td colspan="9">Disc/Drum Transfer Control Group for Program #n+1</td></tr>
<tr><td colspan="2">CØRLØC+3K</td><td colspan="9">Disc/Drum Transfer Control Group For Program #K</td></tr>
</table>

A - Area availability on entry from ECP
0 = Unavailable
1 = Available

C - Core Status
0 = Program is not in core
1 = Program is in core

T - Transfer Status
0 = Program is not in transfer
1 = Program is in transfer or has requested DTRC02 transfer from/to bulk.

*For programs with adjacent save status, this size figure includes save status and program. For programs with disjoint save status, this size figure is for program only.

N - Current Area Status
0 = Program is running with core area unavailable
1 = Program is running with core area available

F - Fortran Available Subroutine Usage Status
0 = Program does not use subroutine while running available
1 = Program uses subroutines while running available

W - Fixed working core status
0 = Program can run anywhere in working core
1 = Program must run from fixed working core area

For permanent.core programs

DRMLØC = 00000000_8

SIZE = 20000000_8

or

$$\underbrace{200XXXXX_8}_{\text{Size for documentation only}}$$

$$\text{CØRLØC} = \underbrace{000LLLLL_8}_{\text{permanent core starting core address of program}}$$

The Save Status Area Control Table is used for programs requiring temporary storage to be saved in an unprotected area on drum before overwriting. This feature is called Save Status. The ECP transfers the temporary storage of programs having Save Status to drum before another program is transferred in its place. However, when a functional program is turned off, its temporary storage is not automatically saved on drum.

Save Status is determined by the Find Save Status Pointer Subroutine (FSSC01).

<u>SAVE STATUS AREA CONTROL TABLE</u>

	23	22 0
$SAVTBL_n$	0	STARTING DRUM/DISC ADDRESS

	23	22	17 15	14	13 0
$SAVSIZ_n$	S	RESERVED FOR FUTURE USE	DRUM/DISC CONTROLLER INDEX #	0	SIZE OF SAVE STATUS FOR BOTH ADJACENT AND DISJOINT AREAS

S ← Save Status Classification

1 = Adjacent Save Status
0 = Disjoint Save Status

*

*Revised 12/66

	23 — 16	15 — 0
SAVLØC$_n$	RESERVED FOR FUTURE USE	BEGINNING CORE ADDRESS

The determination of save status is by bit testing and counting of the STSBTS table.

Bit Status

0 = Program does not have Save Status
1 = Program has Save Status

Example:

There are 50 programs in a system of which 10 have save status.

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8*	7	6	5	4	3	2	1	0
STSBTS	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0

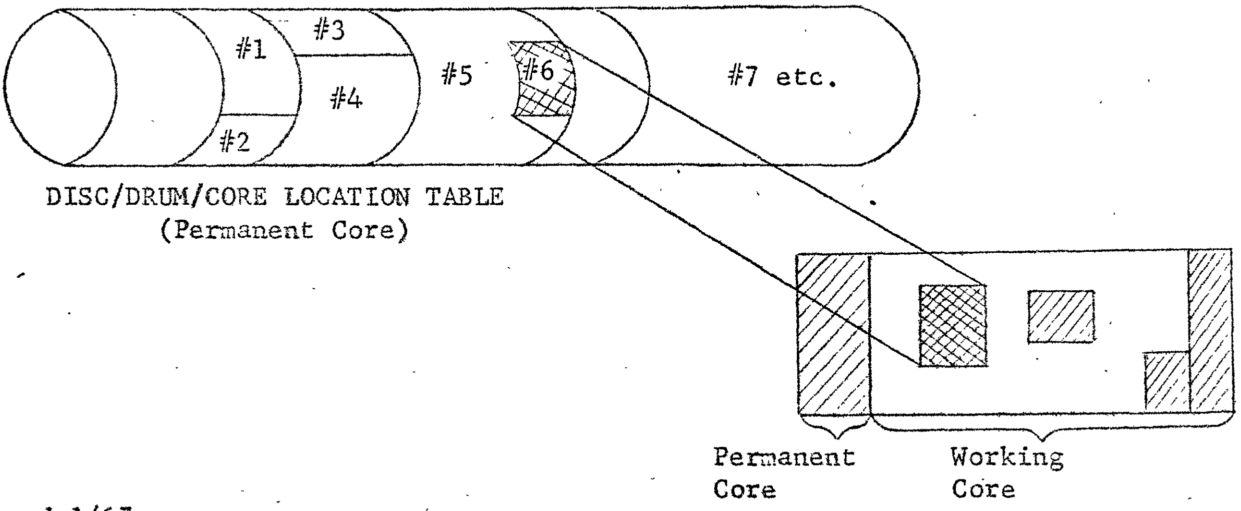
* Locate Save Status Programs by Counting from The Left of Bit 8

Bits 7–0: Summation of Bits In Previous Words

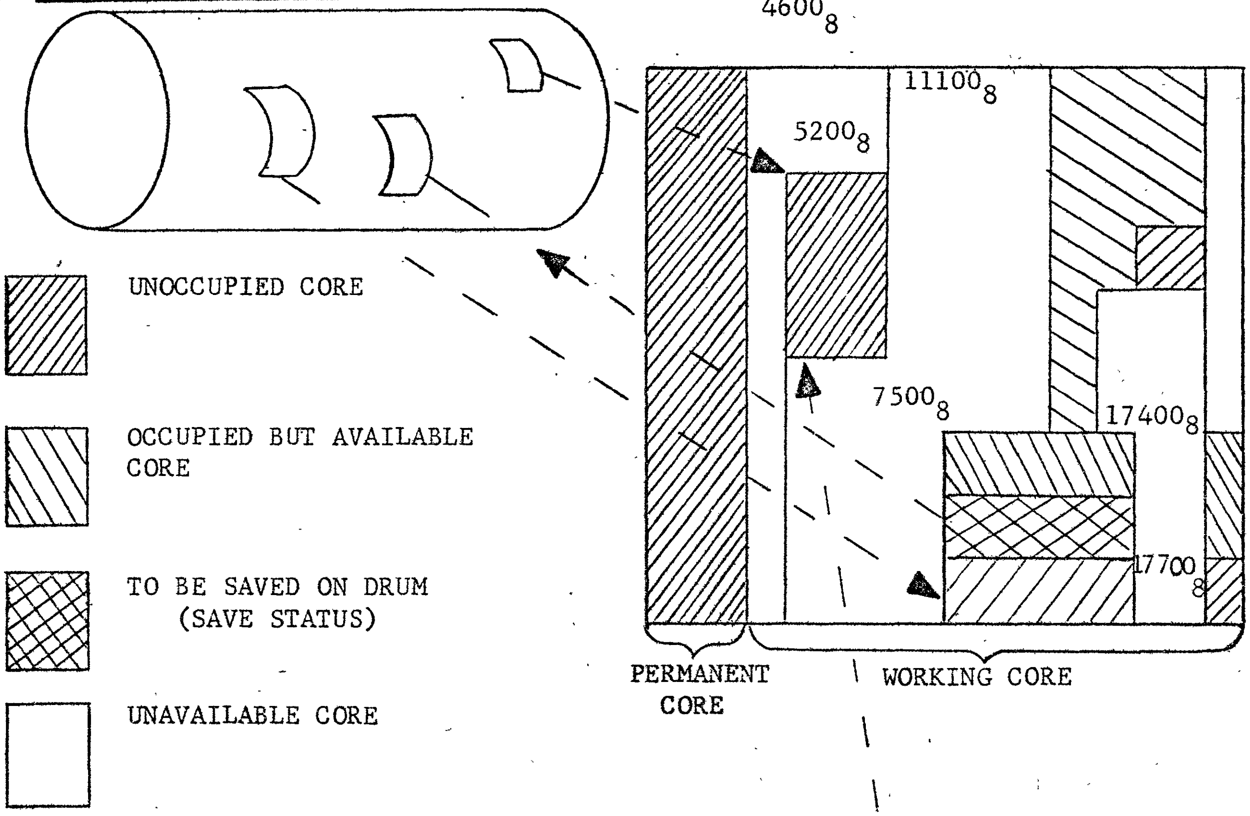
PROGRAMS HAVING SAVE STATUS	ASSOCIATED 3 WORD XFER GROUP USED	
2	0	(Group Consists of SAVTBL, SAVSIZ, and SAVLØC)
6	1	
10	2	
14	3	
22	4	
28	5	
35	6	
43	7	
49	8	
50	9	

The Find Save Status Subroutine (FSS) must be used by all systems having Save Status.

ECP DISC/DRUM/CORE COMMUNICATIONS



ECP DRUM/DISC/CORE COMMUNICATIONS



Each bit in the following tables represents 64 core locations 100_8 (standard block size).

OCCUPIED AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CØRMAP	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1

UNUSED

1 = Core Area Occupied by a Functional Program

AVAILABLE AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AVLMAP	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

UNUSED

1 = Core Area Unavailable for Overwriting

OCCUPIED SAVE STATUS AREA MAP

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STSMAP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

UNUSED

1 = Occupied by save status

* 1.1.1 SAVE REGISTERS SUBROUTINE (SRG)

SRG saves the register contents and/or next entry location for a functional program. For functional programs having full register storage, A, Q, and X3 through X7-Registers for the interrupted program are transferred to the Register Storage Table. Otherwise, only the next entry point to the functional program is saved.

* 1.1.2 RESTORE REGISTERS SUBROUTINE (RRG)

RRG returns to an interrupted program at the designated entry point. The contents of register storage are transferred to the register locations for programs having full register storage. These values represent the contents of the various registers for the running program at the time of its last interrupt.

1.2 TIME AND DIAGNOSTIC COUNT DRIVER (ITC)

ITC uses two interrupts to control the system timing. The first (non-inhibitable) interrupt occurs each 16 2/3 MS on 60-cycle systems, or each 20 MS on 50-cycle systems. Upon interrupt, a DMT counter is decreased by one. When the counter equals minus one, the second (inhibitable) interrupt is triggered.

The second interrupt causes entry to ITC. The DMT counter is initialized by adding the number of 16 2/3 or 20 MS intervals in a time count. This number, NCYCLE, must be evenly divisible into fifty/sixty and is specified by the system programmer at assembly time. NCYCLE determines the length of time represented by one time count (one second, ½ second, ¼ second, etc.). ITC increases the time of day (TIME) by one. The time of day is cleared at midnight and the calendar is updated. ITC also adjusts all program and auxiliary time counters at the beginning of each day. Auxiliary time counters are used by functional programs which require initiation at set time intervals (one minute, two minutes, etc.).Typical programs which may use auxiliary time counter are the Scan or Performance Calculations.

ITC also tests for interrupt driven device failures on the Peripheral Buffer, Output Distributor, Scanner, etc. Each device is assigned a specified CØUNT which is used for counting the time required for activating a particular device. This location (CØUNT) represents the maximum number of time count intervals in which action should be completed. If a CØUNT becomes negative, the device has failed and the Corrective Action Program is turned on. The system programmer may also count for special timing functions. When a system count becomes negative, the ITC turns on a system program (APRØGM), which handles diagnostic functions.

When an interrupt occurs during the execution of (1) interruptable system subroutine, (2) Executive Control Program, (3) TIM/TOM Local Analog Scan Driver, or (4) a program having single-word register storage only, ITC returns immediately to the interrupted subroutine.

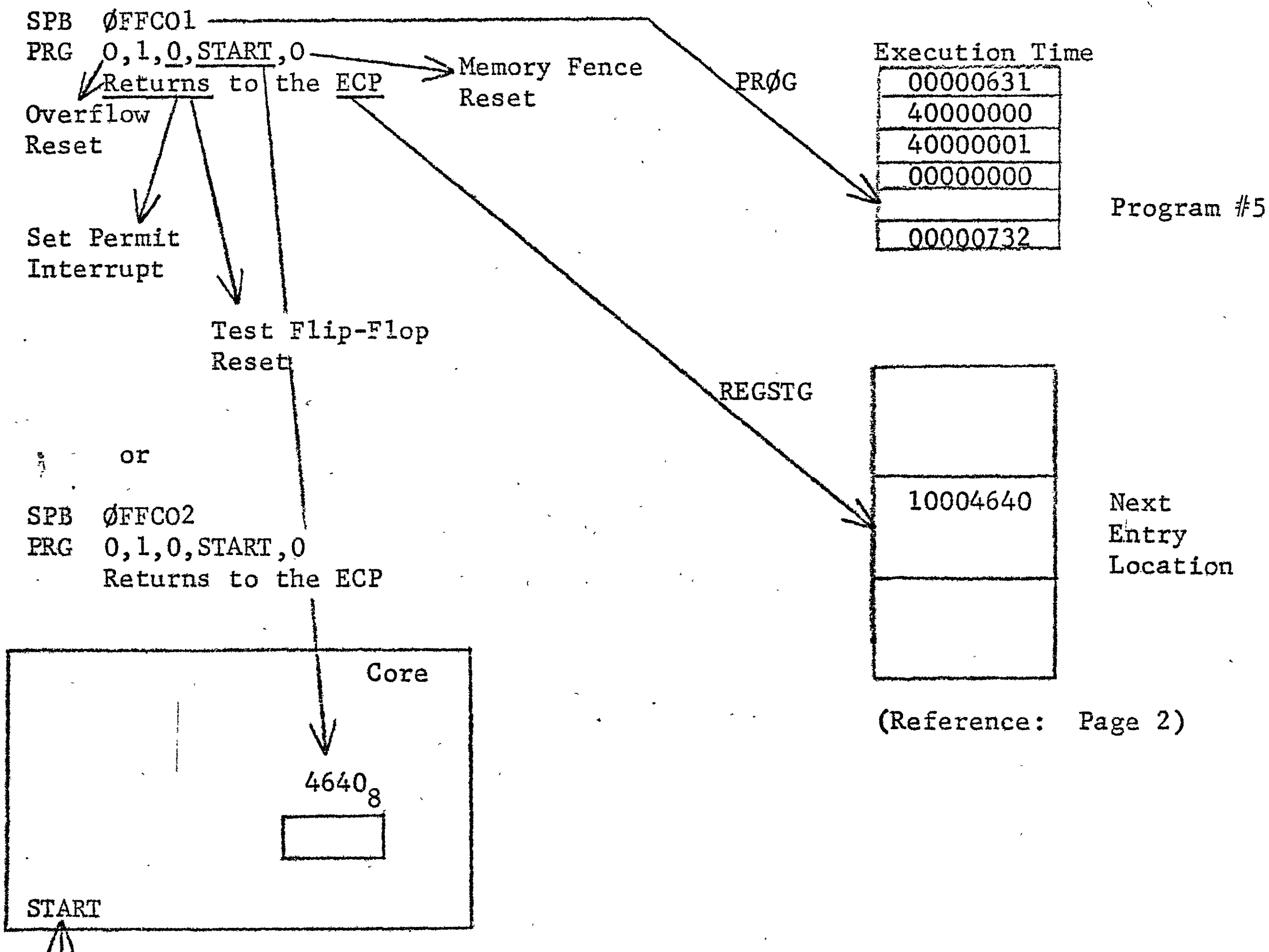
In all other cases, ITC returns control to the interrupted program or to the ECP, depending on Bit 22 being set in the Register Storage Table for systems having RSX. For those without RSX, ITC returns to ECP.

1.3 TURN PROGRAM OFF SUBROUTINE (ØFF)

The Turn Program Off Subroutine stops the ECP from initiating the execution of functional programs. Programs are turned off by placing the "off" constant, 40000000_8, as the next execution time. Only a running program may turn itself off.

After the "off" constant is stored, control is transferred to the ECP.

To communicate with OFF, use the following calling sequences:



```
SPB  ØFFCO1
PRG  0,1,0,START,0
     Returns to the ECP

     or

SPB  ØFFCO2
PRG  0,1,0,START,0
     Returns to the ECP
```

(Reference: Page 2)

Must be relative to start or zero (drum/disc systems only).

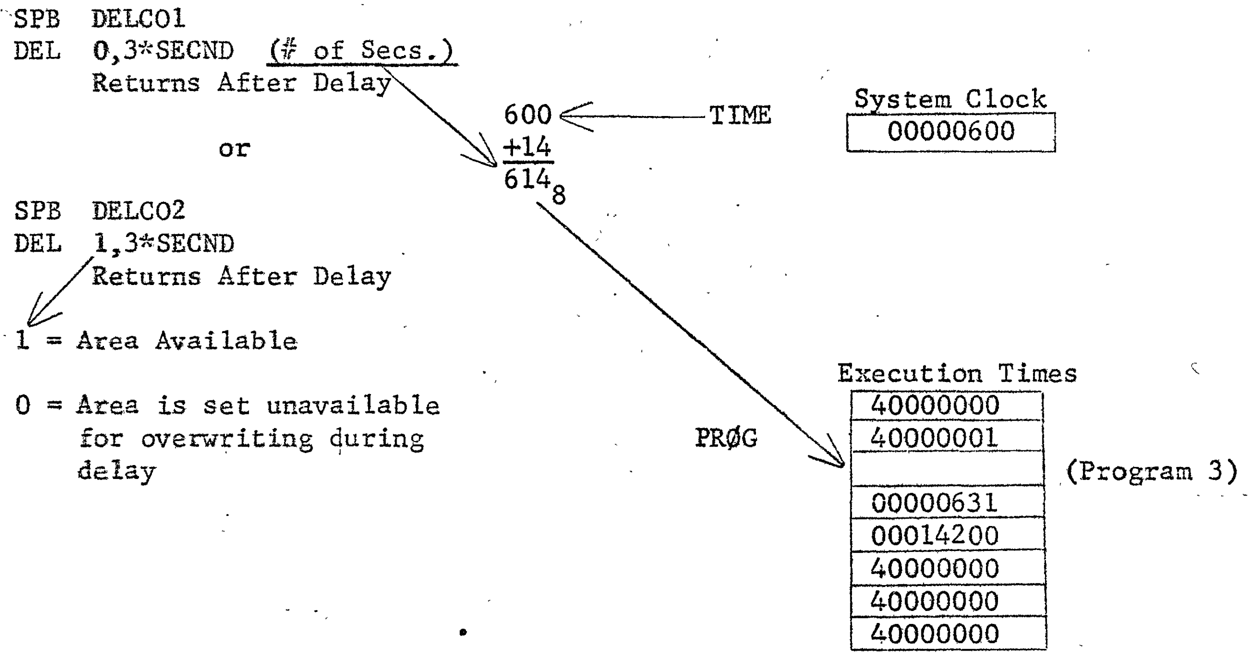
The ØFFCO2 call gives the programmer the ability to set the program area unoccupied and available while the program is turned off. Save status is not saved on drum/disc. Programs that are transferred from drum to core each time executed could effectively use this call.

1.4 SET PROGRAM DELAY SUBROUTINE (DEL)

DEL delays the execution of a functional program that is running currently for a specified time period. The delay, in time counts, is added to the current time and stored in PRØG for the calling function. Only a running program can delay itself.

The A, Q and X3 through X7-Registers are saved for those programs having register storage.

To set a delay, use the following calling sequence:



```
SPB  DELCO1
DEL  0,3*SECND  (# of Secs.)
     Returns After Delay

             or

SPB  DELCO2
DEL  1,3*SECND
     Returns After Delay

1 = Area Available

0 = Area is set unavailable
    for overwriting during
    delay
```

Execution Times
40000000
40000001
00000631
00014200
40000000
40000000
40000000

The above example shows a 1/4-second system.

DELCO2 call gives the programmer the ability to set the program area unoccupied and available during long delays. Save status will not be saved on drum/disc. This call should be used by programs which run at long time intervals. Setting the area unoccupied cuts down ECP map search time. Programs that are transferred from drum to core each time executed would also use this call effectively.

1.5 TURN PROGRAM ON SUBROUTINE (TPN)

The Turn Program On Subroutine is used to change the execution time of functional programs. The execution time and the program number are given in the calling sequence.

After the new execution time is stored for a program, control is returned to the calling function.

Programs which are "locked out", next time of execution 40000001_8, may not be turned on by TPN.

TPN returns with all ones in the A-Register when a request is made to turn a program on which is "locked out".

```
Turn Program 3 on
     LDZ
     SPB  TPNC01
     CØN  G,HLØG
          Return
```

	Execution Times	
	40000000	
PRØG	40000001	
		(Program 3)
	00000631	
	00014200	
	40000000	
	40000000	
	40000000	

```
HLØG  EQL  3
```

The TPNC01 call can turn a program on immediately or set any execution time desired in the PRØG Table.

To enter an execution time only if the program has the off constant in the PRØG Table:

```
     LDA  Execution Time or LDZ
     SPB  TPNC02
     CØN  G,HSCAN
          Return

HSCAN EQL  8
```

The following calling sequence allows the system programmer to specify which program will run next. If the program is in core, it will be entered immediately; if it is on drum, its transfer will be initiated.

```
     LDZ
     SPB  TPNC03
     CØN  G,MSCAN

MSCAN EQL  7
```

At the next ECP entry, the MSCAN program is turned on. If this program has the "lockout" constant in its PRØG location, the request is ignored.

An example of when to use the TPNC03 subroutine call would be: If program #X decides that another program must run next, program X makes a TPNC03 call and either delays or turns itself off. The use of this calling sequence is only required when a program is dynamic and changing and the computer speed is not fast enough to handle functions in their normal sequence.

1.6 MAP MAINTENANCE SUBROUTINE (MAP)

MAP is used to update the core map tables (CØRMAP and AVLMAP). Core areas may be set occupied, unoccupied, available, or unavailable. The following tables are maps of working core:

```
SPB  MAP01 - Set Area Occupied or
SPB  MAP02 - Set Area Unoccupied or
SPB  MAP03 - Set Area Unavailable or
SPB  MAP04 - Set Area Available
     Return to the calling program
```

UNOCCUPIED CORE AREA MAP TABLE CØRMAP

CØRMAP	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In this table, "one" bits indicate that the core area represented is occupied by a functional program whether its area is available or not.

AVAILABLE CORE AREA MAP TABLE AVLMAP

AVLMAP	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

UNUSED

In this table, "one" bits indicate that the core area is unavailable for use.

*Each bit represents 64K words of working core area. For example, if k = 1, each bit represents 64 words of core. The working core area of a 12K system starts at fixed location 12000_8 **in the above example.**

*Revised 12/66

In our example, five programs are in core occupying areas:

12000_8	-	12677_8	(Word 1, Bits 0 - 6)
13300_8	-	14577_8	(Word 1, Bits 11 - 21)
15400_8	-	17677_8.	(Word 2, Bits 4 - 22)
23200_8	-	24677_8	(Word 4, Bits 2 - 14)
25000_8	-	25777_8	(Word 4, Bits 16 - 23)

NOTE: The program in core area 12000_8 - 12677_8 is running with its core area available.

The beginning of the map table is the upper right hand bit of the table and the end is the lower left hand bit.

1.7 DRUM/DISC TRANSFER REQUEST SUBROUTINE (DTR)

DTR requests transfers between drum or disc and core memory. The core address of the 3-word transfer command and the program number plus the three words of the transfer group are stored in the Driver Table. DTR may return to the calling program immediately after the request is stored (DTRC01) or after the completion of the actual transfer, DTRC02.

```
SPB  DTRC01
LDK  DRMXFR         (May be Indexed)
     Error Return   (Driver Table Full)
     Normal Return
```

The Bulk Transfer Driver Program will process disc transfers as though all disc storage (within a controller) is continuous addressing. The programmer is not concerned with data or programs physically split between disc platters.

DRMXFR = Symbolic Address of (3) Word Transfer Command Group

FORMAT OF (3) WORD TRANSFER GROUP

23	22 16	15	14 0
D	Drum or Disc Address		
Controller No.			Number of Words
		Core Address	

D = Direction of Transfer
0 = Drum to Core
1 = Core to Drum

BUILDING PROCEDURE FOR (3) WORD TRANSFER GROUP

```
DEL  D, Bulk Address
FØR  Controller No., Number of Words
LDA  Symbolic Core Address
```

1.8 DRUM/DISC TRANSFER DRIVER (DTD)

DTD initiates transfers between core memory and drum or disc. It is entered from the Drum/Disc Transfer Complete Interrupt.

Following each interrupt, a transfer is initiated by an ØUT Drum/Disc Command. Return from DTD is to the interrupted program or the ECP.

1.9 FIND REGISTER POINTER SUBROUTINE (FRP)

FRP determines a program's type of register storage. It gives the starting address of the 8-word storage block or the next entry address for the specified program.

```
LDA  Program Number
SPB  FRPC01
     Returns with the address in the A-Register and Test
     Flip-Flop Status
```

The test flip-flop is set if the program has full register storage. Otherwise, it is reset.

See Register Pointer Table under Paragraph 1.1.

1.10 FIND SAVE STATUS SUBROUTINE (FSS)

FSS determines whether programs have save status. For those programs with save status, it gives the index to the 3-word save status transfer group (SAVTBL).

```
LDA  Program Number
SPB  FSSC01
     Returns with the index to the 3-word transfer group in the A-Regis-
     ter and Test Flip-Flop Status.
```

The test Flip-Flop is set if the program has save status. Otherwise, it is reset.

* 1.11 ØUC41C/B OUTPUT AVAILABILITY CHECK SUBROUTINE

ØUC is used whenever the programmer desires to check the availability of an output device, a specific fixed data area, or to locate an available floating data area. If the request returns to the "available return", the calling function may assemble its message in the specified area prior to making the actual output request.

A data area may be assigned to each output message representing the drum, disc, or permanent core area containing the message. When an output availability check is made and the area is available, the area is set unavailable and is set free again when the last character of the output message is completed.

The data area number is used as a tag to identify a message area. The tag may apply to an individual message or a buffer which that message occupies. Until the driver removes the data area tag, no other program can use this tag.

Fixed Data Areas

Fixed Data Areas are disjointed designated areas in core or bulk where output data words may be stored. The system must allocate, and system programs must check for, the specific area to be used.

Floating Data Areas

The Floating Data Areas are designated contiguous fixed areas in core (all-core) or on bulk (bulk-core) where output data words may be stored. The system program may use the first available area.

NOTE: On systems where format words are built, they may be stored with the data words in the same data area.

For a fixed area request, the Q-Register will contain data of an unknown nature if the available return is made. The A-Register will contain the number of the data area. For floating data areas, the address of the area to be used will be in the Q-Register upon return to the calling function. Bit 23 will indicate whether this is a core or bulk address. If this is a bulk address, Bit 23 is set and the Q-Register can be stored, without change, as the first word of the bulk transfer group. The A-Register contains the data area number.

For every "available return", where the data area was found, there must be an associated output request. The calling function must realize that an area remains reserved until output is accomplished, therefore, if an area is requested it must be used or it will remain reserved and unused indefinitely. The calling function can release a reserved area by resetting the data area's bit in the data area availability flag, DTAREA. For example, to release data area 13:

```
ØØM  DTAREA
RBK  13
```

* DTAREA, the data area availability flag, is illustrated below:

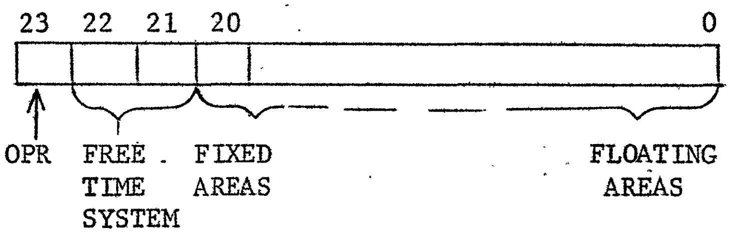


Figure 1. FORMAT OF DTAREA

Data areas 21-23 are reserved for the use of the OPR and Free-Time System. A predetermined number of areas are reserved as fixed areas for the system. The remaining areas are considered to be floating areas, available on a first come, first served basis. The number of floating areas is equated to DTANMB when the system's Monitor is built.

ØUC determines the address of the data area by using the system's check list words DTFBSE and DTFSZE. DTFBSE defines the beginning address of the first floating area (area number 0) and DTFSZE is equal to the size of a floating area. DTFSZE is defined by the system, but must be a multiple of 64.

Calling Sequence

```
p-1      LDA  WORD
p        SPB  ØUCC01
p+1           Unavailable Return
p+2           Normal Return
               .
               .
               .
WØRD     ØUF  0,CKAREA,FDAREA,DVCHCK,0,
              DVPNØ,0,0,DAREA
```

Where:

DAREA, Bits 0-4, specifies the area to be checked if a specific area check is requested.

DVPNØ, Bits 14-18, is the device priority number of the device to be checked ($0 \leq n \leq 23$).

DVCHCK, Bit 20 of the ØUF word controls the device check. 0 = no device check, 1 = perform a device check.

FDAREA, Bit 21 of the ØUF word controls the type of area check to be performed. If Bit 21 is set, this check is for an available floating area. If Bit 21 is reset, this is a specific area check.

CKAREA, Bit 22 of the ØUF word controls the area check. 0 = no area check, 1 = perform an area check.

Upon entry, ØUC examines Bit 20 of the ØUF word. If Bit 20 is set, a device check is performed. The particular peripheral, defined in Bits 14-18 of the ØUF word, are examined for failure, serviceability, and availability.

* Revised 12/66

* This test is performed by examining the BAD and ØØS flags and the device's stacking table respectively. BAD, the peripheral failure flag, indicates which peripheral has actually failed, and ØØS, the out-of-service flag, indicates which peripherals are presently out-of-service (removed for repairs, preventive maintenance, etc.).

A request stacking table exists for each device and this table is checked to determine if there is space for this request. This check is merely a determination of space at this time and does not reserve that space. Therefore, the calling function cannot rely on this space remaining available until a request is issued. When the actual output request is made, the table is re-examined and if space is not available, ØUR, the Output Request Program initiates a delay until space is available.

If Bit 22 of the ØUT word is set, an area availability check is performed. After determining that an area check is required, ØUC examines Bit 21 to determine the type of check that is required. If Bit 21 equals 1, ØUC is to find a floating area. A floating area is found by searching the data availability word, DTAREA. Once found, the area number is placed in the A-Register and its address in the Q-Register. If this is a bulk address, Bit 23 of the Q-Register is set. If an area is not available at this time, a return is made to the calling function with an unavailable indication in the A-Register.

If a peripheral has failed, is out-of-service, has a full stacking table or has no area available at this time, control is transferred to the "unavailable return" with the proper indication set in the A-Register (see Figure 2).

Contents of A	Reason	Action
0	Stacking Table Full	Delay and try again
1	Device Failure	Request an alternate device or bypass output
2	Device out-of-service	Request an alternate device or bypass output
4	No data area available	Delay and try again

Figure 2. REASONS AND INDICATIONS OF UNAVAILABILITY

The calling function must then take an appropriate action depending upon this indication. The indications are such that bit testing or a count of significant zero's can be used to determine exact reason for unavailability.

For bulk systems, the floating areas are located on bulk. For multiple bulk systems, the floating areas must be on one bulk controller. In an all-core system, the floating areas are designated core areas.

The call to ØUC destroys the contents of the A- and Q-Registers and Index Register Number Two.

* Revised 12/66

* ILLUSTRATIVE EXAMPLES:

1. Request for Device Check Only

```
                LDA  WØRD
STEP1           SPB  ØUCC01
STEP1           BRU  NØDEV
                     Normal Return
                 ⋮
TYPER           EQL  3
                 ⋮
WØRD            ØUF  0,0,0,1,0,TYPER,0,0,0
```

In the previous example, an availability check is requested for device number three. If device three is not available for any reason, the return will be to STEP1, the unavailable return, and the calling program branches to its corrective routine based upon the indication within the A-Register. If the device is available, the return is to STEP1+1 and normal processing continues.

2. Request for Specific Data Area, No Device Check

```
                LDA  WØRD
                SPB  ØUCC01
RETURN          BRU  UNAVLB
                     Normal Return
                 ⋮
WØRD            ØUF  0,1,0,0,0,0,0,0,DAREA6
                 ⋮
DAREA6          EQL  6
```

In this example, Bit 20 of the ØUF word is 0 indicating that a device check is not requested. There is,however, a request for data area number 6. If area number 6 is available the return will be RETURN+1 with 6 in the A-Register. If this area is not available the return will be to location RETURN.

3. Request for Any Available Area, No Device Check

```
                LDA  WØRD
                SPB  ØUCC01
XRTN01          BRU  NØAREA
                     Normal Return
                 ⋮
WØRD            ØUF  0,1,1,0,0,0,0,0,0,
```

In this example, Bits 22 and 21 of the ØUF word indicate that the calling function is willing to use any area that is available. If an area is available, the return is to XRTN01+1 with the number of the available area in the A-Register and the address of the floating data area in the Q-Register. If an area is not available, return is to XRTN01 where the responsibility is returned to the calling function.

4. Check Device Number 6, Check Data Area Number 3

```
                LDA  WØRD
                SPB  ØUCC01
                     Unavailable Return
                     Normal Return
                 ⋮
WØRD            ØUF  0,1,0,1,0,6,0,0,3
```

* Revised 12/66

* If successful on search for both, the return will be to p+2. In this case the data area is reserved. However, even though the stacking table is not full at this time, it may be full by the time the ØURC01 request is made.

1.12 ØUR41C/B - OUTPUT REQUEST SUBROUTINE

The ØUR call provides automatic peripheral substitution if the requested peripheral is bad or out-of-service. ØUR also checks the availability of the stacking table for the peripheral or its selected substitute. If the stacking table for the requested peripheral or its alternate is full, a delay is initiated which lasts until the table can contain this request. Neither of these actions, automatic peripheral substitution or stacking table full delay, requires special coding or further action on the part of the calling function.

If an alternate device cannot be found, an alternate unavailable indication (77777777_8) is placed in the A-Register and return is made to the calling function. Since this return is the same as a normal return the calling function must immediately test for a minus one condition.

If the peripheral or its alternate is available and the stacking table can contain the request, ØUR stacks the request, sets the appropriate bit in ALERT, the request-for-peripheral flag, indicating that a request does exist for this device, and turns the Output Program "ON".

The contents of A, Q and X2 are destroyed by this routine.

Calling Sequence

```
p        SPB  ØURC01
p+1      LDK  LABEL(,X)
p+2           Return
```

Where: LABEL(,X) is an address and can be absolute in the first 16K of core, ± 8K relative to the LDK instruction. An index can contain the starting location of an FMR area, and LABEL is the increment from the beginning of the area. "X" can be Registers 3-7. The format of the Label Table is:

```
LABEL    ØUF  PRINT,CØNTRØL,AREANØ,DELAY[1],FUNC,DVPNØ,ØUTMØD,BULKD[1],BULKF[1]
         DEL  A,FØRMAT
         CØN  G, DAREA (or BSS  1, if data area is not used)
         DEL  A, DATA
```

Where:

DAREA, is the number of the data area being used ($0 \leq n \leq 23$).

DATA, is the address of the first word in the data area.

0 for data words in core (Address can be relative or absolute).
1 for data words on bulk (Address must be absolute).

FØRMAT, is the address of the first format word.

A, if A = 0, format words are in core (Address can be relative or absolute
if A = 1, format words are on bulk (Address must be absolute).

PRINT, is Bit 23 of the OUF word. If Bit 23 is set, page line count control is desired. If Bit 23 is reset, control is not desired. This bit applies to printing and typing devices only.

[1] These parameters are utilized in bulk-core systems only.

* CØNTRL, is Bit 22 of the ØUF word. It indicates whether ASCII/CPC or EIA-RS244 code is desired. If Bit 22 is set EIA-RS244 output conversion is desired. If Bit 22 is reset, ASCII/CPC code is desired.

AREANØ, is Bit 21 and indicates whether data areas are used in output request. If Bit 21 is set, the data areas are used. If Bit 21 is reset, data areas are not used.

DELAY, is Bit 20 and indicates delay status, if Bit 20 is set, delay is opposite of present availability status; if Bit 20 is reset, area availability status is not changed.

FUNC, is Bit 19 of the ØUF word. If Bit 19 is set, output is to be punched. If Bit 19 is reset, a printed output is desired. This bit is necessary when either the normal or alternate device is an ASR teletype.

DVPNØ, Bits 14 to 18 are the device priority number.

ØUTMØD, is Bit 13 and when set indicates this message is to be output in binary. Binary output is allowed on card punches and devices with a paper tape punching function only. If the bit is reset the message is output as symbolic characters.

BULKD, Bits 5 to 9 are the bulk controller index to the data words.

BULKF, Bits 0 to 4 are the bulk controller index to the format words.

1. A and Q Established by ØUR Call

```
          LDA  WØRD
          SPB  ØUCC01
          BRU  UNAVL
               Normal Return
           .
           .
           .
          SPB  ØURC01
          LDK  LABEL
          TNM
          BTR  NØALTS
           .
           .
           .
WORD      ØUF  0,1,1,0,0,PRNTR,0,0,0
           .
           .
           .
LABEL     ØUF  0,0,1,0,0,PRNTR,0,0,0
          DEL  0,FØRM1
          CØN  G,DAREA
          DEL  0,DATA
           .
           .
           .
PRNTR     EQL  3
```

The data area number, as well as the data address LABEL + 2 and LABEL + 3, must be stored after making the ØUC call. The DEL word in the Label Table specifies a core address for FORM1, the location of the format table. The request is for device number three,(PRINTER). The return from ØUR is to the TNM instruction. This instruction tests the contents of the A-Register

* upon return. If A-Register is minus one (all one's) the peripheral or its substitute is not available at this time. If this fact is not important, the test need not be made. This example is for a line printer utilizing page control and printing in ASCII (American Standard Code for Information Interchange).

2. No Data Area Used

```
          SPB  ØURC01
          LDK  LABEL
          TNM
          BTR  NØALTS
           ⋮
LABEL     ØUF  0,0,0,0,1,PUNCH,1,0,0
          DEL  0,FØRM1
          BSS  1
          DEL  0,DATA
```

This example is for a binary record on a card punch with no data areas. The teletype punch is an alternate so the fifth field is a one bit.

3. Fixed Area Message

```
          LDA  WØRD
          SPB  ØUCC01
          BRU  UNAVL
               Normal Return
           ⋮
          SPB  ØURC01
          LDK  LABEL
          TNM
          BTR  NØALTS
           ⋮
WØRD      ØUF  0,1,0,0,0,3,0,0,17
           ⋮
LABEL     ØUF  0,1,1,0,1,3,0,0,0
          DEL  0,MESS1
          CØN  0,17
          DEL  0,DATA
```

This example is similar to Example 1 of Section 1.11 except that LABEL+2 and LABEL+3 need not be updated. The data area and data address in this example are fixed. Since the fifth parameter of the ØUF word is a 1, this device has a punching function. Output will be EIA.

* 1.13 ØUP41C/B - OUTPUT PROGRAM

Monitor outputs information by using a group of related routines and subroutines within the Output Program. The Output Program is turned on after receiving an output request from the Output Request Subroutine.

The stored format word is decoded into separate fields. Format words which are stored on drum or disc are transferred to a 64-word core buffer for processing. Data which is stored on drum or disc is placed in a (64) word core buffer. Both buffers are within the Output Program's temporary storage (save) area.

There are two of these (128) word data-format areas, allowing data and format words for two devices to be saved. The highest priority device outputting uses one (128) word buffer. The other buffer is shared by the remaining devices which have messages to be output. The data-format word buffer is not released until the devices output buffer has been either filled or end-of-message is reached.

The proper conversion routine, as indicated in Bits 23-21, is then entered.

The following routines operate exclusively under control of the Output Program:

*BTD - Binary to Decimal Conversion Subroutine for the following output formats:

a) DFP - Decimal Floating Point
b) DFE - E-Type Floating Point
c) DFX - Decimal Fixed Point

*ØCT - Binary to Octal Conversion
BCD - Eight-bit BCD Conversion
*BCN - Binary Character
*CLK - Print Clock Time
*FBB - Four-Bit BCD Conversion

The following subroutines are also contained within the Output Program to assist in placing the converted information in the Driver Table, printing error messages, etc. They are:

UPD - Update Index Pointer to Data Word
SCD - Store Character Executive

Subroutines: within SCD:

STR - Store Character Subroutine
*PCC - Page Control Subroutine

*Indicates Optional Subroutine

Error Typeouts indicate the following:

Incorrect format and data words are indicated by printing an F* (for format), an L* (for large), or D* (for data) instead of the desired information. Unnormalized floating point numbers are an example of an incorrect data word.

Numeric values which cannot be displayed are indicated by printing or punching an L* for numbers which are too large. Numbers which are too small are printed as zero.

The End of Message word is all bits set (77777777_8).

* Every table of format words must contain an End of Message word which is generated by:

```
CØN Ø,77777777
```

The repeat factor is used when more than one data word is associated with a format word. For example, if four data words use the same format, three is placed in this field. If only one data word is used, the repeat factor should be zero.

The multiplying factor positions the decimal point for typing in logs.

Example:

LOG HEADINGS

Pressure of Boiler "A"	Temperature of Steel Furnace
$X10^{-3}$ lbs.	$X10^{+2}$ degrees

Multiplying Factor
To Be Used:
in Format Word
Column 1, -3
Column 2, +2

Color control can only be achieved by using one of the following three format words:

A: BCD
B: CLK
C: FBB

This would mean that if within a message the programmer wants to change color control, a format word may need to be inserted for that purpose only.

*Revised 12/66

If a minus sign is requested, it will be printed if the value is negative. Hence, iif no sign is requested and the value is negative, the absolute value will be typed. Plus signs are never typed.

There is double word output capability in BTD. The maximum number of characters that can be output is 31. (Total width of field, preceding spaces plus characters.) The maximum number of characters for DFP and DFE will be 15. The maximum number of whole numbers that can be indicated in DFX is 9. This limit is caused by the binary scale factor limit of 31.

BCN, binary output can only be output to a punching device.

1.14 OUTPUT DRIVER (ØUD41C/B)

ØUD41C/B is entered from the channel-ready interrupt. All registers are saved for the interrupted program in the register storage area for each device, reserved for drivers.

If this is the channel-ready interrupt and the end-of-message flag, EOMFG, is set, then an actual end-of-message exists. Under these conditions, the EOMFG and PROCFG flags are reset and, if a data area was used for this message, it is set available.

If EOMFG is not set, the interrupted system is returned to following the restoration of the registers and resetting of MESSFG, the buffer availability flag.

For I/O devices, a channel-busy check is made on the channel-ready interrupt. If busy, this indicates that an operator has initiated a request using the operator demand function of that device. If there is a functional program associated with this I/O device and that program is currently off, the driver turns on the program with an information word in the A-Register on subsequent entry. In addition, a flag is set indicating that the demand function has occurred. This information word specifies the device priority number of the input device and whether the device is a keyboard, paper tape reader, or card reader.

Interrupt Entries

Data Ready Interrupt - TØM "Output Buffer Address."
Channel Ready Interrupt - ØUDEii

Where ii is the system's output interrupt index number.

NOTE: The 4020 buffer may contain output packed in several modes, hence, several Channel-ready interrupts will be generated and several TØM words used.

The format words shown in the following conversion routines include an example of the Monitor Pseudo-operation. These pseudo-op instructions can only be used when assembling with GE/PAC Monitor or the GE/PAC Monitor EQL tape.

1.14.1 BINARY TO DECIMAL CONVERSION (BTD41C/B)

BTD41C/B converts binary floating point numbers or binary fixed-point numbers to decimal fixed point characters and stores them in the driver table of the selected device.

A. FLOATING POINT TO FIXED POINT DECIMAL

	23	22	21	20 16	15 12	11	10 7	6 3	2	1	0
DFP	0	0	0	Total width of field	Repeat Factor		Multiplying Factor N	Number of Frac. Digits			

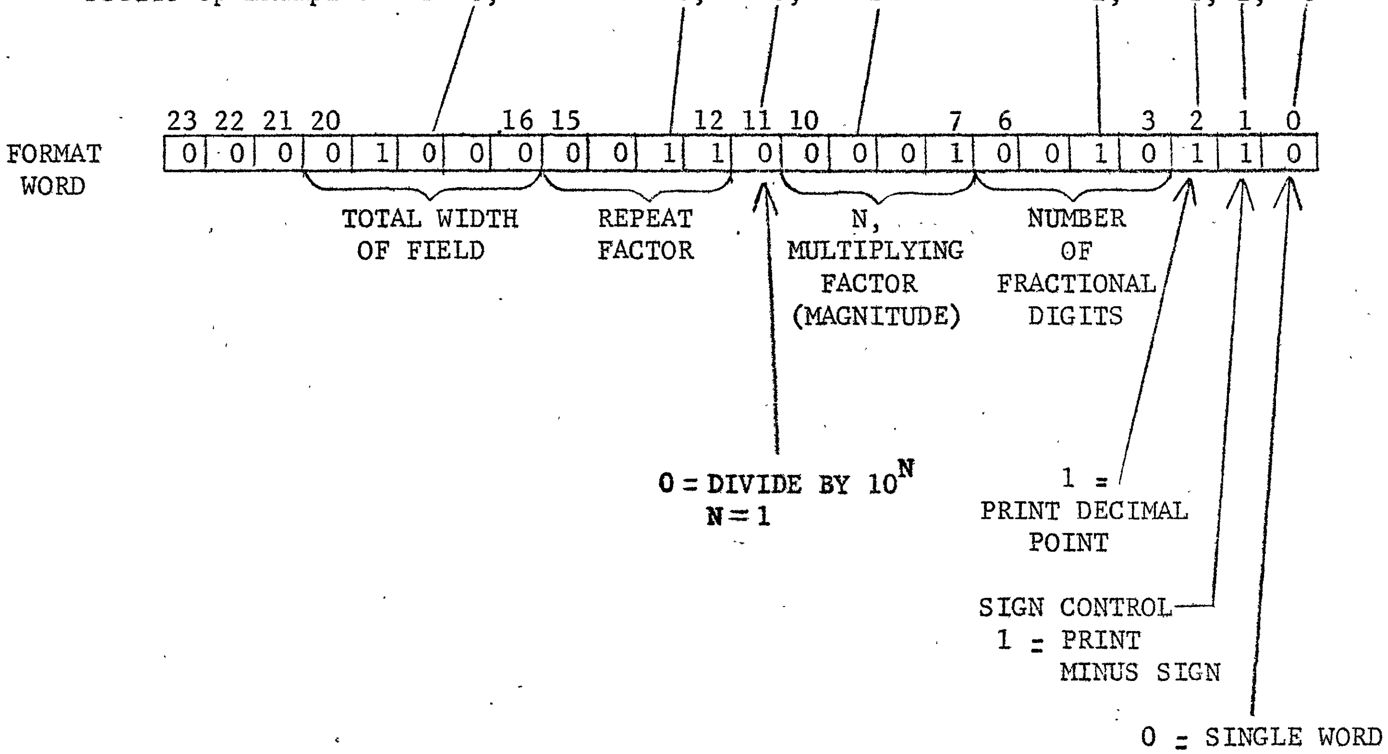
0 = Divide by 10^N
1 = Multiply by 10^N

0 = No decimal point printed
1 = Print decimal point

0 = No sign printed
1 = Print minus sign

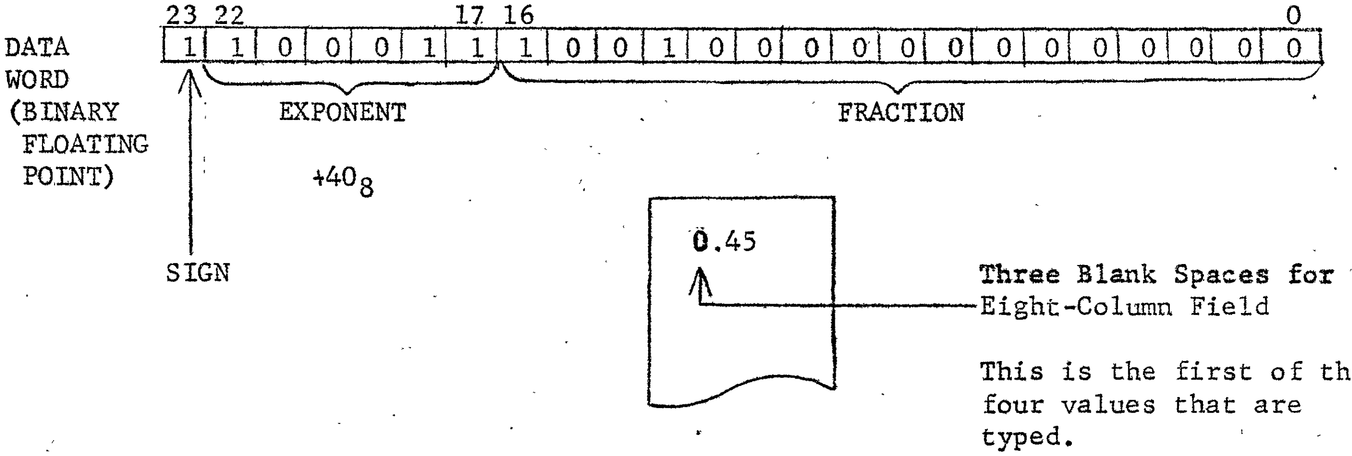
0 = Single word
1 = Double word

Pseudo-op Example: DFP 8, 3, 0, 1 2, 1, 1, 0



*

This pseudo-operation specifies output of four values with a multiplying factor of 10^{-1} and two fractional digits in an 8-column field. It prints the decimal point and minus sign.



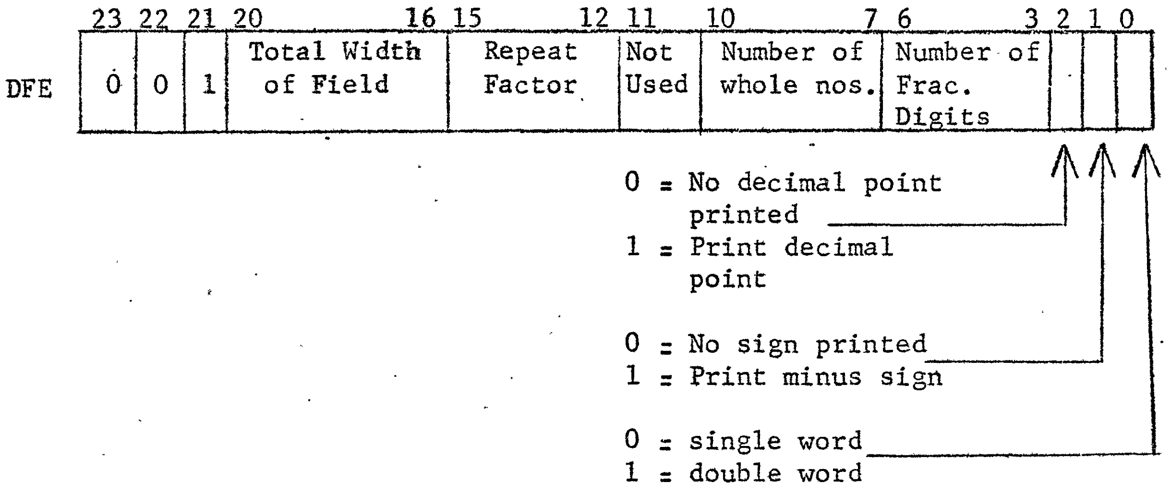
*

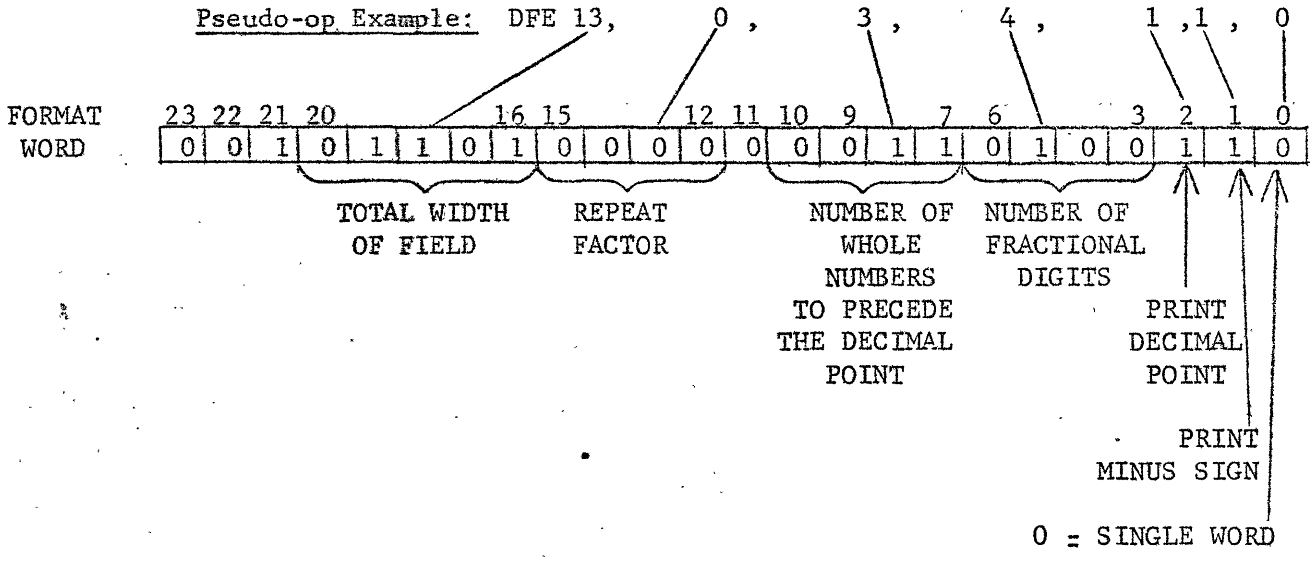
Three Blank Spaces for Eight-Column Field

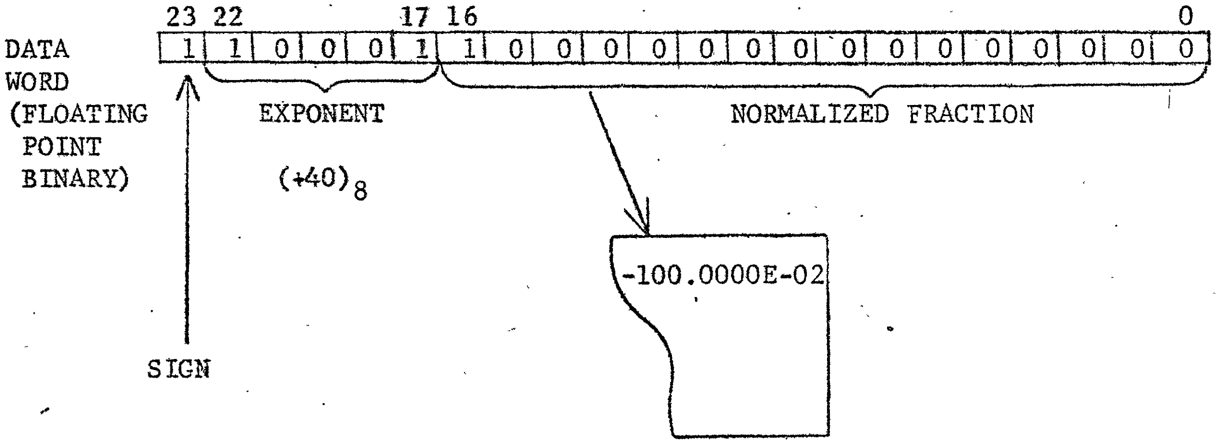
This is the first of the four values that are typed.

*Revised 12/66

B. FLOATING POINT TO FLOATING POINT DECIMAL




Pseudo-op Example: DFE 13, 0, 3, 4, 1,1, 0



C. BINARY TO FIXED POINT DECIMAL

	23	22	21	20 16	15 12	11 7	6 3	2	1	0
* DFX	0	1	0	TOTAL WIDTH OF FIELD	REPEAT FACTOR	BINARY SCALE FACTOR	NUMBER OF FRACTIONAL DIGITS			

Bit 2:
0 = No decimal point printed
1 = Print Decimal point

Bit 1:
0 = No sign printed
1 = Print minus sign

Bit 0:
* 0 = single word
1 = double word

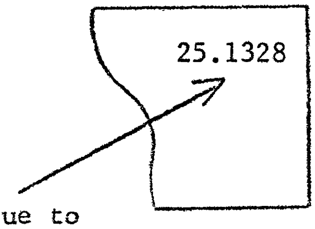
Pseudo-op Example:

DFX 10 , 0 , 5 , 4 , 1 , 1 , 1

	23		21	20				16	15			12	11				7	6			3	2	1	0
FORMAT WORD	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0
				TOTAL WIDTH OF FIELD (10)					REPEAT FACTOR (0)				BINARY SCALE FACTOR (5)					NUMBER OF FRACTIONS (4)				DECIMAL POINT (1)	MINUS SIGN (1)	SINGLE WORD (1)

One value, scaled B5, is typed in a ten-column field. The values are printed with decimal point and four fractional digits. The sign is printed if the value is negative.

	23																							0
DATA WORD	0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
						▲ BINARY SCALE FACTOR																		



25.1328

Three spaces precede the value to make a ten-column field.

*Revised 12/66

1.14.2 BINARY TO OCTAL CONVERSION (OCT41C/B)

ØCT41C/B converts binary integers to octal integers.

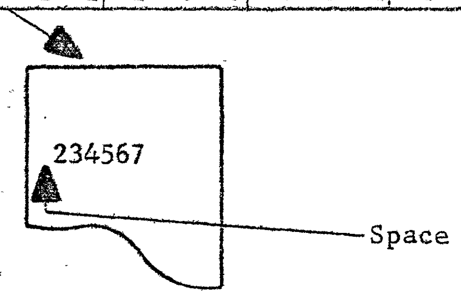
	23	22	21	20 · 16	15 12	11				7	6 3	2	1	0
ØCT	0	1	1	Total width of Field	Repeat Factor	0	0	0	0	0	Number of Octal Chars.	0	0	0

Pseudo-op Example: ØCT 7, 1, 6

	23 21	20 16	15 12	11 7	6 3	2 1 0
Format Word	0 1 1	0 0 1 1	0 0 0 1	0 0 0 0 0	0 1 1 0	0 0 0
		TOTAL WIDTH OF FIELD	REPEAT FACTOR		NUMBER OF CHARACTERS	

Six low order octal digits are typed in a seven-column field. The next data word would be processed in the same manner. (Repeat Factor 1).

	23							0
First Data Word (Binary Integer)	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1



1.14.3 BINARY TO EIGHT-BIT BCD CONVERSION (BCD41C/B)

BCD41C/B converts binary information to eight-bit BCD characters.

	23			20 12	11 7	6 2	1	0
BCD	1	0	0	Number of Characters	Number of Leading Spaces	Format Control (See Page 30)		

CHARACTER MODE (bit 1)
0=3 Per Word Packing
1=1 Per Word Packing

COLOR CONTROL (bit 0)
0=Black
1=Red

FORMAT CONTROL

Line and Control Codes, Printer

Code	Action
00	No Action
01*	Print one line with data in printer output buffer. Single line advance.
02*	Print one line with data in printer output buffer. Double line advance.
10*	Print-line slew to channel 1 - normally top of form
11*	Print-line slew to channel 2
12*	Print-line slew to channel 3
13*	Print-line slew to channel 4
14*	Print-line slew to channel 5
15*	Print-line slew to channel 6
16*	Print-line slew to channel 7
17*	Print-line slew to channel 8
20	Use first character of Data String for Control (Fortran usage)

* The following action is taken when control is in the first format word.

An 01 has no effect, that is, no action will be taken.
An 02 causes one blank line to be printed.
An 10-17 causes a slew to the specified channel.

Card Control Characters - Card Punch

Code	Action
00	No action
01*	Punch one card with data in the card output buffer.
02*	Punch one card with data in the card output buffer. Punch one blank card.
20	Use first character of Data String for Control (Fortran usage)

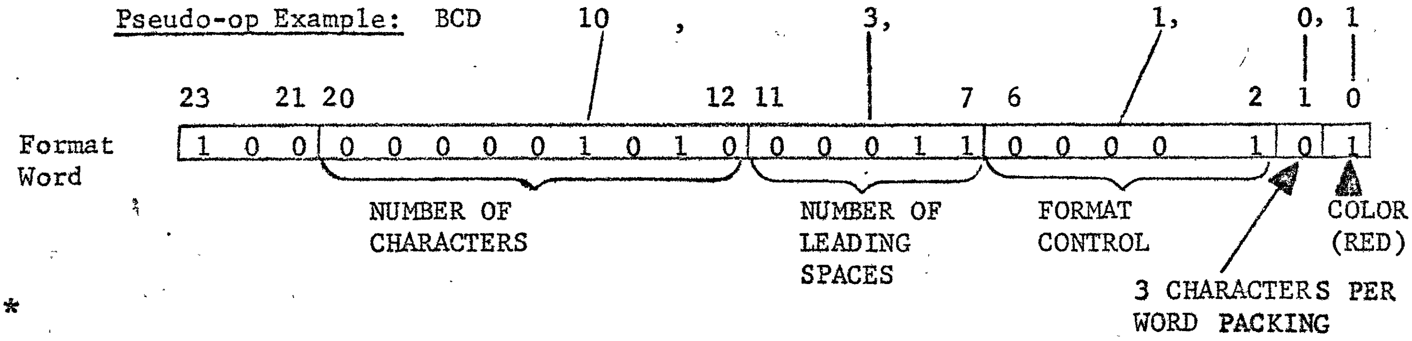
* The following action is taken when control is in first format word.

An 01 has no effect, that is, no action is taken.
An 02 causes one blank card to be punched.
An 10-17 is ignored.

	Leading Carriage Returns - Other Devices
Code	Action
00	No action
01*	One inserted carriage return
02*	Two inserted carriage returns
10	Advance to top of form
20	Use first character of Data String for Control (FORTRAN Usage)

* The following action is taken when control is in the first format word.

An 01 has no effect, that is no action will be taken.
An 02 causes one carriage return to be executed.
The necessary line feeds are inserted for teletypes.
A 10 causes an effective "advance to top of form" through carriage returns.
A 11-17 is ignored.

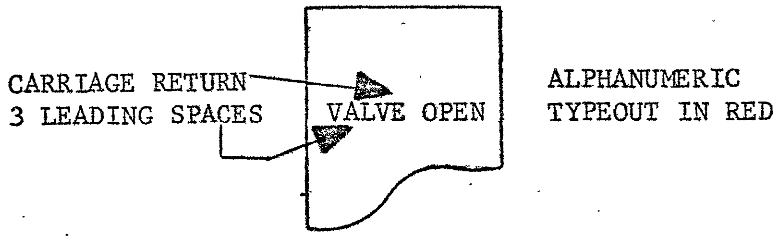


*

CON A, 10 VALVE OPEN (pseudo-op generating data words)

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA WORDS	0	1	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0
	0	1	0	1	0	1	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0
	0	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0
	0	1	0	0	1	1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

*Revised 12/66



1.14.4 BINARY CHARACTER OUTPUT (BCN41C/B)

BCN41C/B places binary information directly into the dirver table for the selected punching device.

	23	22	21	20 ... 12	11											0
BCN	1	0	1	Number of words	0	0	0	0	0	0	0	0	0	0	0	0

Pseudo-op Example:

BCN 3

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA WORDS	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0	0	0
	0	1	0	0	1	1	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0
	1	0	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	0	1	0	1	0	0	1

* If output is to a card punch the first six columns are punched with odd columns using data Bits 23-12 and even columns using Bits 00-11.

1.14.5 CLOCK OUTPUT (CLK41C/B)

CLK41C/B converts the time of day from system time counts to decimal hours, minutes, and seconds. It is then printed as four or six decimal digits.

	23	22	21	20	19	18	17	16	15	14	13	12	11 ... 7	6 ... 2	1	0
CLK	1	1	0	0	0		0		0	0	0	0	Number of Leading Spaces	FORMAT CONTROL	0	

Requested Time
= 0, message time, (no data word)
= 1, event time (data word)

0 = HR/MIN Printout (XXXX)
1 = HR/MIN/SEC Printout (XXXXXX)

(Line and Card Control)

Color Control
0 = Black
1 = Red

FORMAT CONTROL

	Line and Control Codes, Printer
Code	Actions
00	No action
01*	Print one line with data in printer output buffer-single line advance.
02*	Print one line with data in printer output buffer-double line advance.
10	Print line slew to channel 1 - normally top of form
11	Print line slew to channel 2
12	Print line slew to channel 3
13	Print line slew to channel 4
14	Print line slew to channel 5
15	Print line slew to channel 6
16	Print line slew to channel 7
17	Print line slew to channel 8

* Action taken when control is in the first format word:
An 01 will have no effect, that is no action will be taken.
An 02 will cause one blank line to printed first.
An 10-17 will cause a slew to the specified channel.

	Card Control Characters - Card Punch
Code	Action
00	No action
01*	Punch one card with data in the card output buffer
02*	Punch one card with data in the card output buffer Punch one blank card.

* Action taken when control is in the first format word.
An 01 will have no effect, that is, no action will be taken.
An 02 will cause one blank card to be punched.
An 10-17 will be ignored.

	Leading Carriage Returns - Other Devices
Code	Action
00	No action
01*	One inserted carriage returns**
02*	Two inserted carriage returns**
10*	Advance to top of page

* Action taken when control is in the first format word.
An 01 will have no effect, that is no action will be taken.
An 02 will cause one carriage return** to be executed.
An 10 will cause an effective advance to top of page through carriage returns**.
** The necessary line feeds will be inserted for teletypes.
Codes 11-17 will be ignored.

Psuedo-op Example:

CLK 0, 0, 3, 1, 0

23	22	21	20 19	18	17	16	15 12	11 7	6 2	0
1	1	0	0 0	0	0	0	0 0 0 0	0 0 0 11	00001	0

0 = Message Time (no data word)
1 = Event Time (data word)

0 = Hr/Min Printout (XXXX)
1 = Hr/Min/Sec Printout (XXXXXX)

Number of Leading Spaces

Form Control

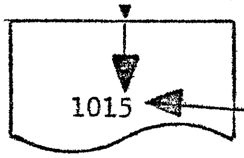
Color Control
0 = Black
1 = Red

The above example types or punches time in hours and minutes preceded by a carriage return and three spaces using black ribbon.

1/4 SEC, COUNTS

* Data Word (TIME) 000000100100000010010000

Time



black typeout
hours & minutes only

1.14.6 BINARY TO FOUR-BIT BCD (FBB41C/B)

FBB41C/B converts binary data, left justified, to six 4-bit BCD characters per word.

* FBB

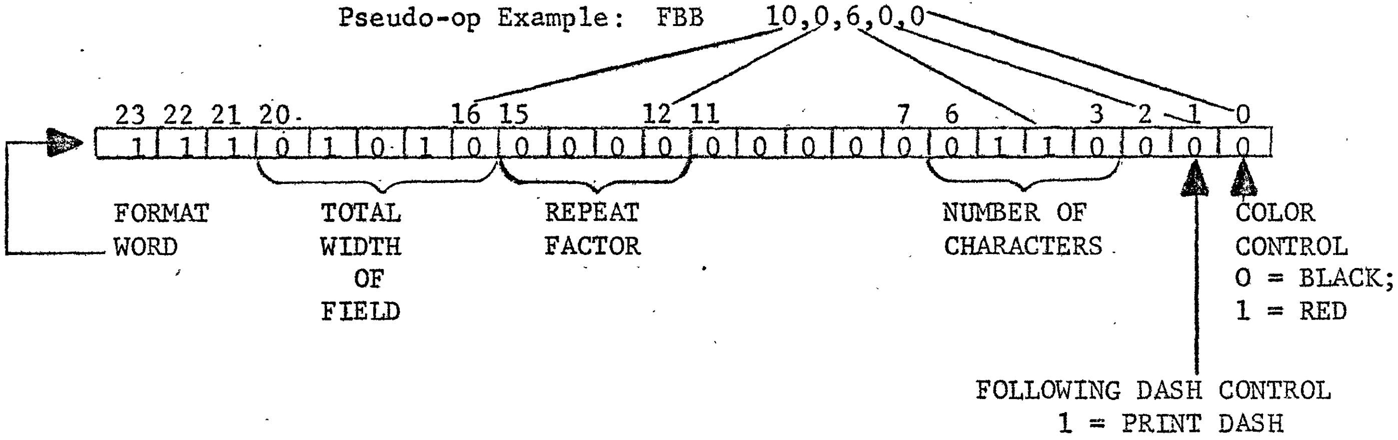
23	22	21	20 16	15 12	11					6	5 3	2	1	0
1	1	1	Total width of field	Repeat Factor	0	0	0	0	0	0	Number of Chars.	0		

Following Dash Control
= 0, No dash
= 1, Print dash

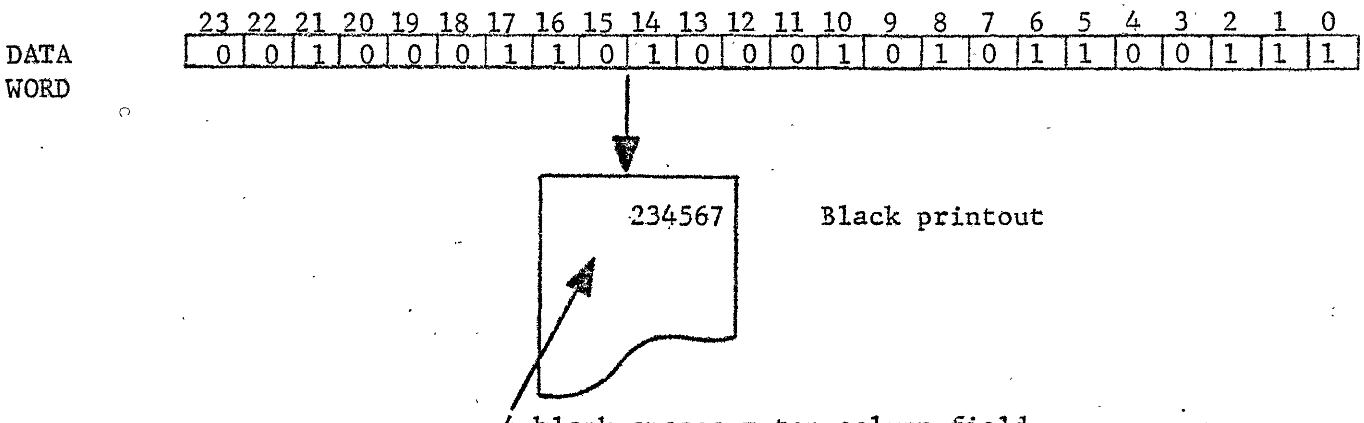
Color Control
= 0, Print black
= 1, Print red

*Revised 12/66

Pseudo-op Example: FBB 10,0,6,0,0



One value, consisting of 6 characters, is typed in black.



4 blank spaces = ten column field

Note: If the repeat factor is used, each data word must have the same number of characters.

1.15 INR41C/B - INPUT REQUEST SUBROUTINE

INR accepts input requests from the paper tape reader, card reader, I/O typer, or teletypes. Input from more than one peripheral may be requested. However, each device is limited to one request at any time. Every request must be tested for completion by using the INRC02 call.

If an INRC01 request is made when the device is available, the read is initiated and the calling program is entered immediately at the "available return" location. At this time the calling program may perform any processing desired.

At any time following the input request, the calling function must request a test for completion by using the INRC02 call. Upon making a completion request, the functional program is locked out until the read is completed or until a failure is discovered.

Upon entering INR, the availability of the device requested in the INF word is tested for whether it has failed, is out-of-service or is currently being used. If any of these conditions exist, INR returns to the unavailable return with the reason for unavailability in the A-Register.

If the device is available, the input request is stored, the device is activated, and the exit is to the normal return.

The symbolic address INPTBL, in the DEL word, points to the list control word. Input characters are stored starting at this address plus one and continue for n number of words.

When specifying NRCHAR, the number of characters to be read, the system programmer must make allowances for all characters. In the case of the paper tape reader, this table must allow for the media- disable code ETX. If the device is a keyboard, this list must allow for any operational errors, carriage returns and correction characters.

* Calling Sequence

Request

```
p       SPB  INRC01
p+1     LDK  LABEL(,X)
p+2          Unavailable Return
p+3          Normal Return
```

Where: The operand LABEL(,X) is the same as for OURC01. The format of the Label Table is:

```
LABEL   INF  CØDE,PCKMØD,FUNC,DVPNØ,NCHAR
        DEL  0,INPTBL
```

Test for Completion

```
p-1     LDK  DVPNØ
p       SPB  INRC02
p+1          error return
p+2          read complete
```

NOTE: The requesting program must have full 8-word register storage.

* Revised 12/66

Where: INPTBL is the label assigned to the input data list. INPTBL must be a core address (relative or absolute) and is the address of the second word of the input area. The first word is the number of characters read, which is inserted by Monitor. *

```
*                   BSS  1  CHARACTERS READ

    INPTBL          BSS  1  LIST CONTROL WORD

                    BSS  N
```

N = Number of Words for Input Data.

CØDE - is Bit 22 of the INF word and indicates the input mode. If Bit 22 is set, the input mode is binary and if Bit 22 is reset, the input mode is symbolic.

FUNC - is Bit 19 of the INF word and indicates whether this device has a keyboard or non-keyboard function. 0 keyboard input function and 1 non-keyboard input. **

PCKMØD - is Bit 20 of the INF word and indicates the desired packing mode. Packing is a function of the device. If Bit 20 is reset, the packing mode is one per word. If Bit 20 is set, packing occurs. *

DEVICE	Packing Mode	
	Binary Input	Symbolic Input
Paper Tape	4	1,3
Cards	1,2	1,3
Keyboard	N/A	1,3

DVPNØ - is the input device priority number (Bits 14 to 18)

NCHAR - is the maximum number of characters allowed for this read (number of BSS words reserved in memory, BSS N of previous example). See Input Buffer Size under General Information. (Bits 0-13).

INPUT BUFFER SIZE

A. Binary Input

1. Card - Record Size, 40 or 80 words - Input Data plus two words for character total and list control word. (Number of words depends on packing mode.) When entered in 40 words, it is unpacked to 80 words if single-word packing is desired. *

* Revised 12/66
** Revised 1/67

* 2. Paper Tape - Number of characters divided by four, plus three words (ETX and the two above).

* B. Symbolic Input

1. Card - 80 words Input Data plus two words, for character total and list control word. (One character per word packing).

   40 words Input Data plus two words for character total and list control word. (Three character per word packing).

2. Paper Tape - Record Size plus three words (ETX, device address and list control word). (Record Size = Number of characters or Number of characters divided by 3 plus one if remainder, depending on packing mode.)

3. Keyboard - Number of characters (including carriage return) plus two words plus an allowance for error characters, (divided by three, if three characters per word packing is desired).

PROGRAM COMMUNICATION

INR41C/B returns to the "unavailable return" of the calling sequence with the reason unavailable in the A-Register. (Refer to Table 1.)

Contents of A-Register	Reason
0	Device busy
1	Device failed
2	Device out-of-service
4	Alarm line 3 (Media problem)

Table 1. Unavailable Indications

The INRC02 call, test for completion, returns to the normal return when the read is complete or the error return if the read was in error. The error indication is in the A-Register of the calling program. Table 2 describes these error conditions. These errors are detected by the Input Driver and alarmed by the Input Program. The actual corrective action is the responsibility of the system programmer.

Upon returning under normal conditions or if a data error occurs, the A-Register of the calling program contains the number of words accepted.

* Revised 12/66

Contents of A-Register	Cause	Possible Action
Bit 19	Keyboard Input Message Time Aborted	Reinitiate request
Bit 20	Input Buffer Overflow	Reinitiate request
Bit 21	Alarm line 4 (Data error)	Have record repositioned, then reinitiate request
Bit 22	Alarm line 2 (Mechanical or electrical failures	Terminate until repairs are made
Bit 23	Alarm line 1 (Operator intervention)	Delay and reinitiate request

NOTE: There will be no conversion or filtering on error returns except for alarm line 4. (Data error)

Table 2. Error Indications

Examples:

* 1. Request for input using a card reader:

```
p          SPB  INRC01
p+1        LDK  LABEL
p+2        BRU  RDERR
p+3             Normal Return
            •
            •
            •
LABEL      INF  0,1,1,READER,80
           DEL  0,INBUF
READER     EQL  3
           BSS  1
INBUF      BSS  1
           BSS  40
```

In this calling sequence, the request is for input of 80 characters[1] which are to be packed. Packing is three per word, left justified. The right-most position of the 27th word is zero-filled. RDERR1 is the address of a system-defined error routine. If the requested device was unavailable, the return to the calling program is to the branch to RDERR1. At RDERR1 the system program must perform some appropriate corrective action, depending upon the nature of the error. The reason for error is contained in the A-Register of the requesting program (see Table 1).

[1] If the request has specified any number of characters other than 80 (such as 27, 40, or 160) the actual number of characters accepted is 80 characters (one full card).

Test for completion:

At any time following the input request, the calling function must make a test for completion using the INRC02 call.

```
          LDR  READER
          SPB  INRC02
          BRU  RDERR2
```

After a successful read complete, the requesting program can expect to find 27 words of data starting at location INBUF+1.

If the read had been in error the requesting program branches to RDERR2 for analysis and corrective action depending upon the nature of the error. The actual error can be determined by examining the bits in the A-Register upon return to the requesting program (refer to Table 2).

* 2. Request for input using a card reader:

```
p         SPB  INRC01
p+1       LDK  LABEL
p+2       BRU  RDERR1
p+3            Normal Return
           :
LABEL     INF  1,1,1,READER,80
          DEL  0,INBUF
```

This request is the same as example number one except the input is binary. This effects the packing mode, causing packing to be two characters per word. The results are located from INBUF+1 to INBUF+40. The test for completion is the same.

* 3. Request for input using a keyboard:

```
p         SPB  INRC01
p+1       LDK  LABEL
p+2       BRU  KEYR1
p+3            Normal Return
           :
LABEL     INF  0,1,0,KEYBD,30
          DEL  0,KEYIN
KEYBD     EQL  4               Where "n" is the number of characters
          BSS  1               that might be typed incorrectly plus the
KEYIN     BSS  1               DELETE character (<—). The number of
          BSS  10+n            characters should include a carriage return.
```

* This request for device number four indicates that thirty characters are to be inputted through the keyboard into buffer KEYIN. Packing is three characters per word. However, the size of the buffer is 10 words + n. The buffer must be able to accommodate the entire record plus any operational errors and correction characters that may be entered. All characters are accepted and placed in the buffer, three per word. If more than 30 characters are entered,these additional

characters are ignored. After the read is completed, the input program edits and packs the data in the desired mode. The number of characters read is the BSS word at KEYIN-1. The number of words packed is in the A-Register of the calling program upon return to that program. In this example, the words should be in KEYIN+1 to KEYIN+10. If less than or more than 30 characters were accepted, this is reflected in the "number of words" found in the A-Register upon return to the requesting program. A test for completion must follow the input request.

```
          LDK  KEYBD
          SPB  INRCO2
          BRU  KEYER2
```

* 4. Request for input, keyboard:

```
          SPB  INRCO1
          LDK  LABEL
          BRU  KEYER1
               Normal Return
           :
LABEL     INF  0,0,0,KEYBD,30
          DEL  0,KEYIN
           :
          LDK  KEYBD
          SPB  INRCO2
          BRU  KEYER2
               Normal Return
```

This request is the same as the previous request except that packing is not desired. The characters, therefore, are stored one per word starting at KEYIN+1.

NOTE: An INF word of INF 1,x,x,x,x (binary input) constitutes an illegal request for a keyboard. The result of such a request depends upon the packing mode. The input program assumes that the characters are symbolic and edits and packs them accordingly.

* 5. Request for input, paper tape reader:

```
          SPB  INRCO1
          LDK  LABEL
          BRU  PTERR1
               Normal Return
           :
LABEL     INF  1,1,1,PAPERD,60
          DEL  0,PTRBUF
           :
PAPERD    EQL  5
          BSS  1
PTRBUF    BSS  1
          BSS  16
```

This request is for 60 binary characters. Packing has been requested and is four characters per word. In fact, binary input from paper tape is always packed four characters per word whether it is specified or not. The 7th bit of the input character is deleted when packed.

Test for completion:

```
          LDK  PAPERD
          SPB  INRC02
          BRU  PTERR2
```

* 6. Request for input, paper tape reader:

```
          SPB  INRC01
          LDK  LABEL
          BRU  PTERR1
               Normal Return
           .
           .
           .
LABEL     INF  0,1,1,PAPERD,60
          DEL  0,PTRBUF
```

This request is the same as the previous request except that input is symbolic. Symbolic input is always packed three per word.

Test for completion:

```
          LDK  PAPERD
          SPB  INRC02
          BRU  PTERR2
```

1.16 INPUT DRIVER (IND41C/B)

The Input Driver non-inhibitable data-ready interrupt will be a TIM instruction.

GE/PAC 4020

Data Ready Interrupt - TIM "Input Buffer Address"
* Channel Ready Interrupt - SPB INDEii

ii = system's input interrupt index number

Upon receipt of the channel-ready interrupt, the Input Program is turned on. If this device is I/O in nature, its output function is tested to determine if there is something in the buffer to output. If it is, an enable command is issued for the output channel and the diagnostic count for that device is set.

For input devices, a channel busy check is made on the channel-ready interrupt. If busy, this indicates that an operator has initiated a request using the operator demand function of that device. If there is a functional program associated with this I/O device and that program is currently off, the driver turns on the program with an information word in the A-Register on subsequent entry. In additon, a flag is set indicating that the demand function has occurred. This information word specifies the device priority number of the input device and whether the device is a keyboard, paper tape reader or card reader.

1.17 INPUT PROGRAM (INP41C/B)

The input program is turned ON by the Input Driver upon receipt of the channel ready interrupt.

Upon entry, the Input Program INP searches for requested input devices in priority number. When a device with "read complete" is found, its read status is checked to determine if input occurred error-free.

If an error was found while reading, the reason for the error is placed in the A-Register of the calling program and this program is turned ON if a test for completion had been initiated.

If the read was error-free or has only a data error, the INF word for this request is decoded and a check is made to determine the type of conversion needed. If the device is a card reader, the characters can be stored as read or converted from Hollerith to ASCII. If the device is an IBM Selectric, the CPC code is converted to ASCII. All illegal characters are converted to a special error character.

If input is through a keyboard, the data is filtered and then packed as requested.

The total number of words, after packing, are in the A-Register of the calling program which is turned on at this time.

The Input Program returns to either the error return or read complete return of the "test for completion" call of the requesting program. The test for completion calling sequence is:

```
p-1  LDK  DVPNØ
p    SPB  INRC02
p+1       error return
p+2       read complete
```

Where: DVPNØ is the device priority number

If the return is "read complete", the A-Register of the calling program contains the total number of words in the buffer. If the return is an "error return", the A-Register contains an error indication. If the error was a data error, it will also contain the number of words. These indications are summarized below:

Contents of A-Register	Cause	Possible Action
Bit 19	Keyboard Input Message Time Aborted	Initiate Request
Bit 20	Input Buffer Overflow	Reinitiate Request
Bit 21	Alarm line 4 (Data error)	Have record repositioned initiate request.
Bit 22	Alarm line 2 (Mechanical or electrical failures)	Terminate until repairs are made
Bit 23	Alarm line 1 (Operator intervention)	Delay and Initiate Request

NOTE: There will be no conversion or filtering on error returns except for Alarm line 4, (data error).

1.18 MULTIPLE OUTPUT REQUEST SUBROUTINE LOCAL AND REMOTE (MOR46)

The Multiple Output Distributor is used for addressing and controlling decimal, binary, or analog data output groups from the Arithmetic Unit. MØR46 supplies the communication link to the Local or Remote driver for obtaining these outputs.

A group status word is needed if the unaffected bits on an output must be set to the same position as the last output for that group.

The status word table contains one word of data for each group in that MØD defined by the programmer as having a current status word. Each word contains a group address and the current status of the contacts for that group. Group status words may be arranged in any order within the table. If bit 7 of the Q-Register equals zero, the index used by the calling program is an index to the status table.

Two words are required to request output of a particular group.

Example:

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	10
A-Register	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0		6		
Q-Register	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0		4		
	Status Tables																7	6					
1000	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	1	0	3		2		
1001	0	1	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	0	2		7		
1002	1	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	0	0	1		4		
1003	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	2		1		
1004	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	1	0	1		3		
1005	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0		6		
1006	1	0	1	0	1	0	1	0	0	0	1	1	0	0	1	1	0	0	2		2		
1007	1	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	11		5		
1010	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0		4		

Explanation:

In the mask word (Q-Register), bit 7 is reset meaning that this group has a current status word and bits 0-5 of the A-Register contain an index to the status table, which is 06. Since bits 17-23 of the Q-Register are set, the corresponding bits of the A-Register are output as shown. The remaining bits (8-16) are output as shown in the group status word for group 22 (index 06), with one exception.

Explanation: contd

Any bit set in the A-Register 8-16, is or'ed into the status word and output. The request and driver programs do not check this and it should be considered a system program error, (see bit 14 of A-Register in the foregoing example).

LOCAL REQUEST

The first word (A-Register) contains a group address or an index to the status word table. Bit 7 of the second word (Q-Register), if set, indicates that the first word contains a group address and has no status word. If bit 7 is reset, it implies that the first word contains an index to the status word table and there is status saved for this group, The second word contains an index to the correct multiple output distributor. If a group has a status table, the status words are initially set by the system programmer.

REMOTE REQUEST

Bit 23 of the first word indicates the mode of the request, whether normal or repeat. If the repeat mode is selected, the driver sets the repeat bit on its output request. The Remote Scanner Controller saves and sends the output request back to the driver for verification. The driver then resets the repeat bit and sends the output, knowing the Remote Scanner Controller will compare this request with the saved command. If they agree, the output is made, otherwise, an error condition exists. This first word also states the type of formats used, either 8-bit (8 bits of data plus 5 bits for group address) or 10-bit (10 bits of data plus 3 bits for group address). The 10-bit format words have only one operational delay time. This is four milliseconds. The 8-bit format words have two, 76 or 4 milliseconds. Bits 5-0 of the first word contain a group address if bit 7 of the Q-Register is set. Otherwise, they contain an index to the status word table if bit 7 is reset.

In addition, the second word contains an index to the correct multiple output distributor.

NORMAL OUTPUT - LOCAL OR REMOTE

For Local and Remote requests, all bits set in the first word are output. For normal requests, the first word contains group output in bits 8-23; operational delay time in bit 7, alternate (1) or immediate (0) return to calling program in bit 6 and a group number or status table index in bits 0-5.

The second word contains a mask of the bits to be changed in the status word bits 8-23, bit 7 indicates whether this group has a status word, and bits 0-6 indicate the Multiple Output Distributor being used.

NORMAL OUTPUT - LOCAL OR REMOTE - contd

The current status is updated for those groups which have status. There is no updating for groups without status. In either case, the output command and program number are stored in the appropriate Priority or Non-priority Driver Table.

TIMED-LATCH OUTPUT - LOCAL ONLY

In the case of the timed-latch request, the first word indicates which positions of the output group to latch. The second word gives the time required between the latch and unlatched commands.

When a timed request is made, MØR46 tests the timer availability flag. If a timer location is free, MØR46 places the timer number and output command in the appropriate Priority or Non-priority Driver Table. It also saves the program number, time of latching and contact status for driver.

For a request to unlatch a previous request, no information is stored in the Driver Table. The unlatch flag is set for the output driver which aborts the request before the request time expires. Three flags are used for Timed Latching Output:

1. Unlatch Flag Word - set on completion or abort.
2. Timer Available Flag Word - set when timer in use.
3. Timer Active Flag Word - set when timer is timing.

PULSED OUTPUT - LOCAL ONLY

For a pulsed request, the first word states the direction of the pulses. The second word contains the number of pulses to be sent out. The output command, number of pulses and program number are stored by MØR46 in the Priority or Non-priority Driver Table for pulsed request.

OUTPUT OPTIONS

The operation delay time is the time required to transfer a command word from the A-Register to the MØD Command Register and to initiate the transfer of the data portion of the command word to the output function specified by the matrix address. The time to complete these transfers is either four milliseconds, seventy-five milliseconds or forty microseconds, depending upon the output function for a local request. For a remote request, the transmission time must also be added to get total time.

There are two types of returns which are specified by bit 6 of the first word. The Immediate Return returns to the normal exit of the calling sequence after the output demand is stored in the Driver Table. The other return locks out the calling function (which is turned on by the Output Driver) until the output is completed. Control is then returned to the normal exit of the calling sequence.

SYSTEM COMMUNICATION

Calling Sequence

NORMAL REQUEST - LOCAL

p - 2 Place the change word in the A-Register (First Word)

Bits 23-8 - Data:

1 = contact to be closed
0 = contact to be opened or remain unchanged.

Bit 7 - Operation Delay Time:

1 = 40 microseconds output or
75 milliseconds output
0 = 4 milliseconds output

Bit 6 - Ready Signal:

1 = Return when output is complete
0 = Immediate Return

Bits 5-0 - Multiple Output Group Number if Bit 7 Change Mask Word is set, or

* Bits 5-0 - Index to the Status Word Table if Bit 7 of Change Mask Word is **reset.**

p - 1 Place Change Mask Word in the Q-Register (Second Word).

Bits 23-8 - Data:

1 = Bit to be set or reset as in A-Register.
0 = Bit to remain as in status if group has status word.

Bit 7 - Status

1 = Group has no status word.
0 = Group has a status word.

Bits 6-0 - Index to Multiple Output Distributor

```
P     SPB  MØRC01 Non-Priority Request
                  or
P     SPB  MØRC02 Priority Request

P+1   ERROR RETURN

P+2   NORMAL RETURN
```

***Revised 12/66**

NORMAL REQUEST - REMOTE

P - 2 Place change word in the A-Register (First Word)

Bit 23 - Repeat bit

0 = Normal mode
1 = Repeat mode

Bit 22 - Unassigned

Bit 21 - Size of Data Field

0 = 8-bit format word
1 = 10-bit format word

Bit 20 - Operational Delay Time

For 8-bit format words

Bit 20 = 0 = 4 millisecond output
= 1 = 75 millisecond output

For 10-bit format words

Bit 20 must be zero for a 4 millisecond output only

Bit 19-7 - Data and Group Address

For 8-bit format words

Bits 19-12 - Data

1 = Contact to be closed
0 = Contact to be opened or remain unchanged

Bits 11-7 - Group Address

For 10-bit format words

Bits 19-10 - Data

1 = Contact to be closed
0 = Contact to be opened or remain unchanged

Bits 9-7 - Group Address

Bit 6 - Ready Signal

1 = Return when output is complete
0 = Immediate Return

Bits 5-0 - Multiple Output Group Number if Bit 7 of Change Mask word is set, or

Bits 5-0 - Index to the Status Word Table if Bit 7 of Change Mask word is reset.

P - 1 Place Change Mask Word in the Q-Register (Second Word).

Bits 23-20 - Unassigned

For 8-bit format

Bits 19-12 - Data

0 = Bit to be unaffected
1 = Bit to be changed

Bits 11-8 - Unassigned

For 10-bit format

Bits 19-10 - Data

0 = Bit to be unaffected
1 = Bit to be changed

Bits 9-8 - Unassigned

Bit 7 - Status

1 = Group has no status word.
0 = Group has a status word.

Bits 6-0 - Index to Multiple Output Distributor

P SPB MØRC01 Non-Priority Request
or
P SPB MØRC02 Priority Request

P+1 ERROR RETURN

P+2 NORMAL RETURN

TIMED REQUEST - LOCAL ONLY

P-2 Place the latch command in the A-Register (First Word).

Bits 23-8 - Data

1 = Contact to be latched
0 = Contact to remain unchanged

Bit 7 - Operator Delay Time

1 = 40 microseconds output or
75 milliseconds output
0 = 4 milliseconds output

Bit 6 - Ready Signal

1 = Return when output is completed
0 = Immediate Return

Bits 5-0 - Multiple Output Group Number if Bit 7 of Timer Word is set, or

Bits 5-0 - Index to the Status Word Table if Bit 7 of Pulsed Word is reset.

P-1 Place the Timer Word in the Q-Register (Second Word)

Bits 23-8 - Number of counts for relay to remain latched.

(1 count = 16 2/3 milliseconds for 60-cycle systems)
(1 count = 20 milliseconds for 50-cycle systems)

Bit 7 - Status

1 = Group has no status word
0 = Group has a status word

Bits 6-0 - Index to the Multiple Output Distributor

```
P     SPB  MØRC03  Non-Priority request
                          or
P     SPB  MØRC04  Priority request
                          or
P     SPB  MØRC05  Unlatch a previous request (Immediate return only)
```

P+1 ERROR RETURN

P+2 NORMAL RETURN

PULSED REQUEST

P-2 Place pulse command in the A-Register.

Bits 23-8 - Data

1 = Contacts to be closed
0 = Contacts to remain open

Bits 7 - Operation Delay Time

1 = 40 microseconds output or
75 milliseconds output
0 = 4 milliseconds output

Bit 6 - Ready Signal

1 = Return when output is complete
0 = Immediate Return

Bits 5-0 - Multiple Output Group Number if Bit 7 of Pulse Word is set, or

Bits 5-0 - Index to the status word if bit 7 of pulsed word is reset.

P-1 Place Pulse Word in Q-Register (Second Word).

Bits 23-8 - Number of pulses to be sent out.

Bit 7 - Status

1 = Group has no status word*
0 = Group has a status word*

*NOTE: A pulsed request does not need status.

```
P     SPB  MØRC06  Non-Priority Request
                        or
P     SPB  MØRC07  Priority Request

P+1   ERROR RETURN

P+2   NORMAL RETURN
```

PROGRAM COMMUNICATION

MØR46 returns to the "Error Return" of the calling sequence when one of the following conditions exist:

A-Register Contents	Error Condition
0	Driver Table Full or No Timer Available for Timed Requests.
1	Requested Group has failed on two previous consecutive overloads.
2	Output failed on two consecutive overloads (Alternate Return).
4	Invalid Multiple Output Group Address (Status, Index, Max. for groups with status) or Multiple Output Distributor Index).
8	Multiple Output Distributor Timer Failure, Remote Scanner Failure, or Communications Coupler failed.
16	Invalid Remote Output Requested.

1.19 MULTIPLE OUTPUT DISTRIBUTOR DRIVER - LOCAL (MDR)

MDR is initiated by the MØD Interrupt to execute requests made by MØR. To output a request, the command word from the driver table is placed in the A-Register. The data is transferred to the location specified by the group address of the command word. The Multiple Output Distributor is then ready to accept another request.

All output commands are initiated from the Priority Driver Table before information is taken from the Non-Priority Driver Table.

For timed requests, a timer is activated. When the requested time has expired, MDR executes the unlatch command.

For pulsed requests, a pulsed timer is started. When it expires, the contacts are opened.

Timed Contact Output Request Subroutine-Local and Remote (TCØ42)

The Timed Contact Output Controller is used for addressing and controlling timed contact output groups from the Arithmetic Unit. TCØ42 supplies the communication link for obtaining these outputs.

The output command is stored with the program number in the Normal or Priority Output Driver Table. In a normal request, the new output command is placed in the Normal Driver Table. For a priority request, the new output command is stored in the Priority Driver Table. All priority requests are serviced prior to Normal Driver Table requests.

There are two types of returns (specified by bit 7 of the Command Word) following a request for output. Return 1 returns to the normal exit of the calling sequence after the output command has been stored in the driver table. Return 2 locks out the calling program until the output is completed. The calling program is then turned on and control is returned to the normal exit of the calling sequence.

SYSTEM COMMUNICATION

Calling Sequence (Local Controllers)

```
p-1  Place the TCØ Command Word in the A-Register:

     Bits 23-16 - Number of timing pulses for contact closure time or
                  stepping.
          15-11 - Index # of Controller
             10 - Position for Setpoint Controller
                  0 = Move Setpoint Down; 1 = Move Setpoint Up
              9 - 0 = GE/MAC*PULSE OUTPUT
                  1 = PULSE DURATION OUTPUT
              7 - 0 = Normal Return (Immediate)
                  1 = Alternate Return (After Operation is Completed)
              6 - 0 = Matrix Address

     p          SPB     TCØC01              Normal Request
                     or
     p          SPB     TCØC02              Priority Request
     p+1                Busy/Error Return
     p+2                Normal Return
```

Calling sequence (Remote Controllers)

```
p-1  Place the TCØ Command Word in the A-Register

     Bits 23-0 = Normal Mode
             1 = Repeat Mode
          22-17-Number of timing pulses for contact closure time or stepping
          16-0 = Move Setpoint Down
             1 = Move Setpoint Up
          15-0 = GE/MAC Pulse Output
             1 = Pulse Duration Output
          14-8-Index # of Controller
           7-0 = Normal Return (Immediate)
             1 = Alternate Return (after operation is completed)
           6-0-Matrix Address

p                SPB      TCØC03            Normal Request
                     or
                 SPB      TCØC04            Priority Request
p+1                 Busy/Error Return
p+2                 Normal Return
```

* Trademark of General Electric Company

PROGRAM COMMUNICATION

TCØ42 returns to the "busy/error return" of the calling sequence when one of the following conditions exist:

A-Register	Error
0	Driver Table Full
2	Output Failed on Two Consecutive Overloads
4	Invalid Matrix Address
8	TCØ Timer Failure (Deadman), Remote Scanner, or Communications coupler failed.

Timed Contact Output Driver (LØCAL) - TCD

TCD uses the Timed Contact Output Controller for addressing and controlling timed contact output groups from the Arithmetic Unit. To output a request, TCD places the command word in the A-Register. The data is transferred to the location specified by the group address of the command word. The Timed Contact Output Controller is then ready to accept another request.

All output commands are initiated from the Priority Driver Table before information is taken from the Normal Driver Table.

1.20 SCAN REQUEST SUBROUTINE (SCR) - (LOCAL AND REMOTE)

The Scan Request Subroutine stores the addresses of the Scanner Command Word and Count Value Tables in a stacking table. The Scanner Commands are executed by the Scan Driver. SCR processes normal or priority, buffered and non-buffered scan requests. A normal request is processed on a first in/first out basis. Priority requests are processed on a last in/first out basis. The system has the option of storing time of the completed scan as last word of scan request count table.

SYSTEM COMMUNICATION

Calling sequence

Non-buffered Scan Request

```
p     SPB   SCRC06   Normal Reuest or
p     SPB   SCRC07   Priority Request
p+1   DEL   0,SCPTR
p+2         Busy/error return
p+3         Scan Complete return

      SCPTR = address of Analog Scan request table
```

Buffered Scan Request

```
p     SPB   SCRC08   Normal request or
p     SPB   SCRC09   Priority request
p+1   DEL   0,SCPTR
p+2         Busy/error return
            Normal return
              .
              .
              .
              .
              .
              .
```

To Determine When a Buffered Scan Request has been Completed

```
p     SPB   SCRC05
p+1         Scan Incomplete return
p+2         Scan Complete return
```

Analog Scan Request Table Layout

SCPTR table example

Scanner$_A$ index number
Core address of scan command words
Core address of count value table

Scanner$_B$ index number
Core address of scan command words
Core address of count value table

-
- groups of three for
- each analog scanner
-

77777777_8 End of request indicator

The last word of the individual scanner's Scan Command Word Table must also be a 77777777_8.

Example of Scanner Index number:

Four local and two remote analog scanners have index numbers 0-5. These index numbers are the systems programs link or identification with the correct scanner.

PROGRAM COMMUNICATION

A scan request returns to the "busy return" of the calling sequence with zero in the A-Register when:

1. The individual scanner's stacking table of requests is full or

2. The program currently has a buffered scan in progress or

3. The system is currently processing the maximum number of scans from the system stacking table.

It returns to the "busy return" with a 77777777_8 in the A-Register if none of the requested scanners are good.

If anSCRC05 request is made after a buffered scan is complete, its return is to p-2. The A-Register contains a minus one (77777777_8) if any scanners have failed and a zero if all scanners are good.

The program must check SCNBAD to determine which of his requested scanners have failed. These scanners could be both local and remote. If scans from several scanner controllers were requested, the scans are completed on the working scanners before returning. If a request is made and the scanner has already failed, the request will not be stacked for the failed scanner. To place a bad analog scanner back in service, the SCNBAD bit must be reset.

GENERAL INFORMATION

The locations of the Scan Command Word Table, Count Table, and the number of the calling program are saved in the scan stacking tables.

A flag is set indicating that a scan is in progress for this program (if the call is for buffered scan). The flag is reset by the Scan Driver when the scanning is completed. A program cannot request a second scan until a first buffered scan is complete.

A scanner can only be called once per request. The scanner index numbers in the Analog Scan Request Table, SCPTR, need not be in order.

For each Remote Scan Command Word, it is possible to request one of two modes. The first is an analog input only. The second is an analog input followed by a digital input in the counts table. This option is shown by bit 18 in the Scan Command Word.

A remote or local 1 API priority scan request interrupts a normal scan. However, a local priority 3 API scan request allows a normal scan in progress to finish before the priority is serviced.

The thermocouple reference count correction along with the offset correction and scaling is completed before storing a value in the counts table.

The System Programmer must update up to eight thermocouple reference count values for each analog scanner in the system. The label for this is:

RETMXX - where XX is the Analog Scanner Index number. The first analog scanner will be #00.

SCAN COMMAND WORD FORMATS

A. LOCAL SCANNERS

1. Successive approximation Converter (high and/or low level)

23	22 20	19 18	17 15	14 11	10 7	6 3	2 0
	THERMOCOUPLE REFERENCE BLOCK TEMPERATURE INDEX		GROUP ADDRESS W	MATRIX ADDRESS M/N	POINT ADDRESS P/Q	MUST BE ZERO	VOLTAGE SCALE CONTROL

0 = No gain optimization desired
1 = Gain optimization desired (applies only to single channel, single API scanner)

00 = Single Input Mode
01 = Group Input Mode (2 or 4 points per group
10 = Single Input Mode
11 = Group Input Mode (8 points per group)

2. Integrating Converter

23	22 20	19 18	17 15	14 11	10 7	6 5	4 3	2 0
	THERMOCOUPLE REFERENCE TEMPERATURE INDEX		GROUP ADDRESS W	MATRIX ADDRESS M/N	POINT ADDRESS P/Q	OPERATION CONTROL	INTEGRATING TIME CONTROL	VOLTAGE SCALE CONTROL

0 = No gain optimization desired
1 = Gain optimization desired (applies only to single channel, single API scanner)

00 = Single Input Mode
01 = Group Input Mode (2 or 4 points per group)
10 = Single Input Mode
11 = Group Input Mode (8 points per group)

B. REMOTE SCANNERS

Successive approximation converter

23	22	21 19	18	17 12	11	10 9	8 7	6 5	4 3	2 0
R				RESERVED FOR FUTURE USE	GROUP ADDRESS W	MATRIX ADDRESS M	MATRIX ADDRESS N	POINT ADDRESS P	POINT ADDRESS	VOLTAGE SCALE CONTROL GAIN

0 = Analog and Digital Input
1 = Digital Input Not Saved

Thermocouple Reference Block Temperature Index

0 = No Gain Optimization
1 = Use Gain Optimization

0 = Normal Mode
1 = Repead Mode (The Remote Scanner will save and send the output request back to the Driver for verification before starting input)

Buffered Scan Request

Buffered scanning is an option which may be selected at the time a request is made.

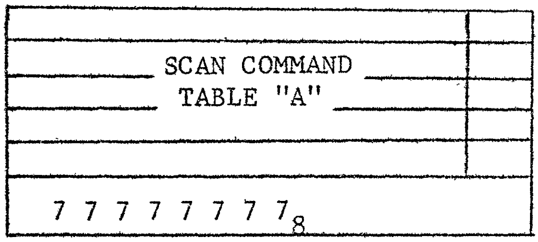
A request to test for buffered scan completion can be made by making an SCRC05 call. If the scan is complete, the program gets a "p+2" return, otherwise an incomplete return (p+1) results.

When the scan request is stored in the request tables, return is made immediately to the normal return.

At this time the functional program processes the count values placed in the alternate count table by a previous scan request. In this way, a functional program can process one set of count values while the scan driver is placing count values in an alternate list and drive the scanner at close to maximum speed.

```
        SPB   SCRC08   (Normal Request) or SPB   SCRC09   (Priority Request)
        DEL    0, SCPTR
                Busy Return (Stacking List Full)
                Normal Return
```

Core
Addresses
Only

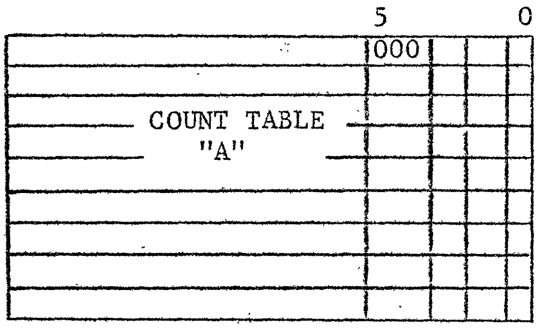


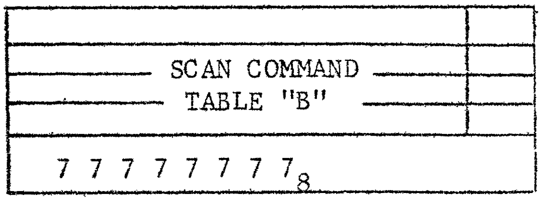
SCAN COMPLETE REQUEST SCRC08 or SCRC09

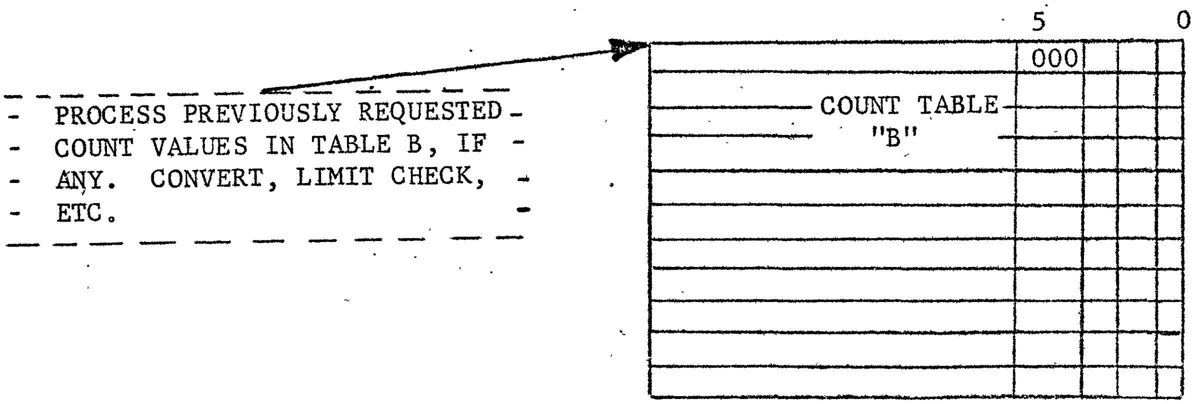
SPB SCRC05

Scan Incomplete
Scan Complete

Request scan for "B Tables and process new counts in "A" Table





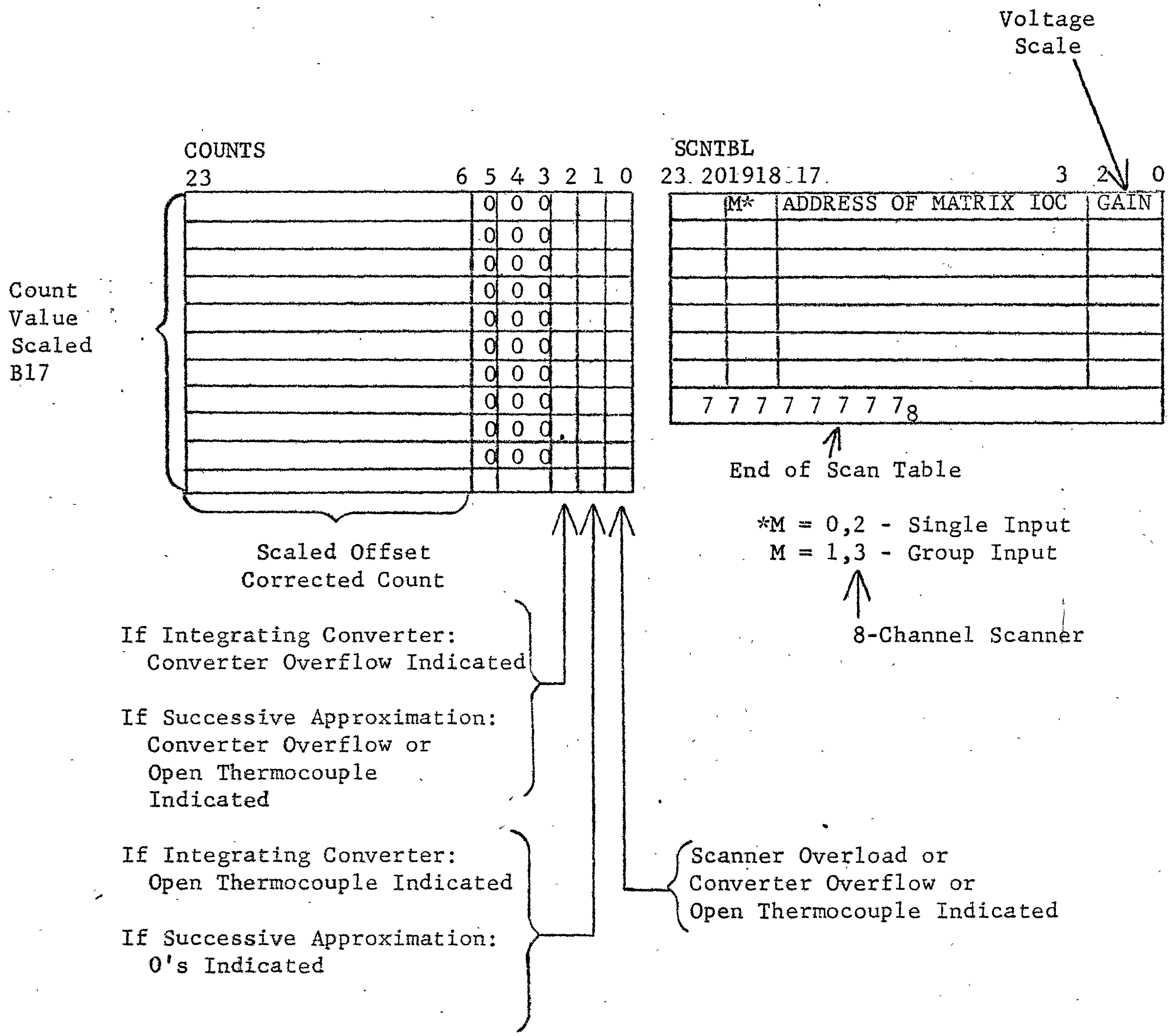

PROCESS PREVIOUSLY REQUESTED COUNT VALUES IN TABLE B, IF ANY. CONVERT, LIMIT CHECK, ETC.
5
0
000
COUNT TABLE
"B"

Non-Buffered Scan Request

When the scanning is completed, return is made to the "Scan Complete Return". Local and remote (1) API are normal requests processed on a first in/first out basis. Local 3 API priority requests are processed on a last in/first out basis. Remote and local 1 API priority requests are processed on a first in/first out basis. There are separate tables for normal and priority (1) API scans.

A non-buffered scan request locks out the calling program until the analog scan request is processed.

```
              SPB  SCRC06    (Normal Request or SPB  SCRC07    (Priority Request)
Core          DEL  0, SCPTR
Address <-
Only               Busy Return (Stacking List Full)
                   Scan Complete Return
```



Voltage Scale
COUNTS
23
6 5 4 3 2 1 0
0 0 0
Count Value Scaled B17
Scaled Offset Corrected Count
If Integrating Converter: Converter Overflow Indicated
If Successive Approximation: Converter Overflow or Open Thermocouple Indicated
If Integrating Converter: Open Thermocouple Indicated
If Successive Approximation: 0's Indicated
Scanner Overload or Converter Overflow or Open Thermocouple Indicated
SCNTBL
23 20 19 18 17 3 2 0
M*
ADDRESS OF MATRIX IOC
GAIN
7 7 7 7 7 7 7 7₈
End of Scan Table
*M = 0,2 - Single Input
M = 1,3 - Group Input
8-Channel Scanner

1.21 ANALOG SCAN DRIVER (SND) LOCAL

The Scan Driver outputs Scanner Commands from a Driver Table. They are sent to the Scanner by a TØM or ØUT Command. The count values are returned by a TIM or IN Instruction and stored in the specified table. The count values are converted, offset corrected, and scaled before storing. After all requested points have been scanned, the calling function is turned on.

1.22 SCAN OFFSET PROGRAM (SCF)

The analog readings of the shorted pair are obtained for each voltage scale. This new offset is calculated from the weighted average of the current reading and previous offset values for each voltage scale.

1.23 REMOTE DIGITAL INPUT REQUEST SUBROUTINE (RDS41)

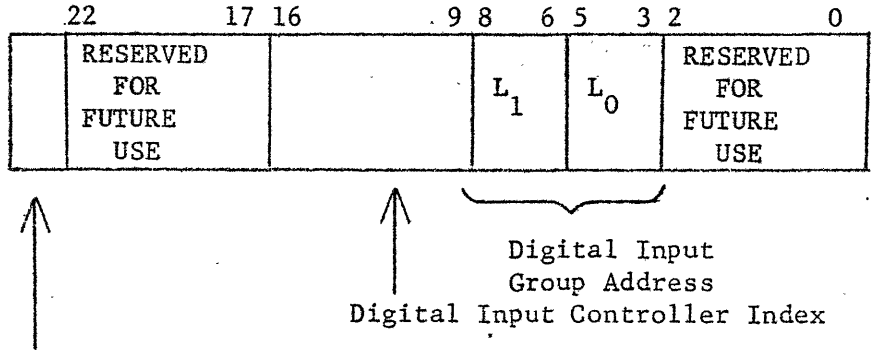
RDS41 is used to request digital inputs from digital input controllers on systems having remote scanners. The system programs requesting digital input are "locked out" while their digital input requests are being processed.

SYSTEM COMMUNICATION

Calling Sequence

```
p-1  Place the DIS Command Word in the A-Register
p    SPB  RDSC01
p+1       Busy/Error Return
p+2       Normal Return
```

The format of the DIS Command Word is shown below:



0 = Normal Mode
1 = Repeat Mode
(The remote scanner will save and send the output request back to the driver for verification before starting input)

The digital Input Controller index indicates which controller is being requested.

e.g. Two controllers in system

1st has index # = 0
2nd has index # = 1 .

The number of data contacts read per digital input group (L_1L_0) has been reduced to eleven plus validity.

PROGRAM COMMUNICATION

RDS41 returns to "busy/error return" of the calling sequence when one of the following conditions exist:

A-Register	Error
0	Driver Table full
8	Digital Input Controller, remote scanner, or communications coupler has failed.

RDS41 returns to the "normal return" of the calling sequence with the following word in the A-Register:

23	13	12 11	10 0
RESERVED FOR FUTURE USE	VALIDITY INDICATOR		DATA CONTACTS

I/O Typer Input Demand or Character Ready
From the Addressed 4307 Remote Scanner

Bit 11 will always be reset at this time. For more information on the use of this bit, see TSD41 Remote Scanner Driver Section.

RDS41 is a permanent core subroutine which runs inhibited. It restores the original interrupt condition upon return to the calling program. RDSC01 is a common label defined on the system EQL tape.

A program using this subroutine must have full eight-word register storage.

1.24 CORRECTIVE ACTION DIAGNOSTIC (CAD)

CAD performs peripheral, drum, disc, multiple output distributor, scanner, and timed-contact output corrective actions for the GE/PAC Monitor. Corrective action is taken for the following peripherals:

4221C	Fixed Carriage Output Typer (15 CPS)
4223C	Long Carriage Output Typer (10 CPS)
4233C	Teletype (10 CPS)
4253C	Paper Tape Punch (120 FPS)
4262C	Line Printer (300 LPM)
4270C	I/O Typer (15.5 CPS)
4282A	Card Punch (100 CPM)
4224C	Card Reader (CR 10)
4213C	Digitronics paper tape readers (100, 200, 300, FPS)

Corrective actions are as listed:

1. Substitute an alternate output peripheral when an output peripheral fails.
2. Permit operator recovery for the card reader, card punch and printer.
3. Reset device flags and switches for the failed device.
4., Turn on any programs "locked out" for the failed device.
5. Type alarm messages indicating the exact location of the failure, when possible.

After the necessary actions are completed, CAD turns itself off and exits to the ECP.

1.25 INITIALIZATION ROUTINE (INZ)

INZ initializes the start-up conditions for a Monitor System. Initial storage, switches, and variable locations are set for Monitor Programs.

Each system should add its own system initialization to this routine.

To initialize an all-core system, the programmer must follow the outlined steps:

1. Branch to the starting location of the Initialize Routine (37_8).

2. Press Console → B.
3. Turn key to "Automatic".
4. Press "Step" button.
5. Reset the "API Lockout" switch.

For a drum/disc core system; will be defined later.

1.26 FIND/RESTORE WORKING CORE AREA SUBROUTINES (FMR)

The Find Working Core Area Subroutine is used to find space for reading data from paper tape, scanning analog points, transferring a program segment, building an output data table, transferring an untested program of the Free-Time System, etc. When a space is found, this area is set to unavailable, occupied status.

The Restore Working Core Area Subroutine releases an area by setting it unoccupied and available.

To find a working core area,

```
LDA  Number of Locations
SPB  FMRC01
     Unavailable Return
     Normal Return with the location of the free
     core area in the A-Register
```

The "Unavailable Return" is taken when a working core area is not available. The system programmer, at this time, should set a delay and then initiate another FMRC01 subroutine call.

To release a working core area,

```
DLD  A-Register with the Number of Locations
     Q-Register with the starting core location
SPB  RMRC01
     Normal Return
```

1.27 Remote Scanner Driver (RSD41)

The Remote Scanner Driver is used to operate the Model 4307 GE/PAC Remote Scanner Controller through the Model 4306 Communications Coupler. It permits:

1. Analog scanning from remote Analog Input Controllers

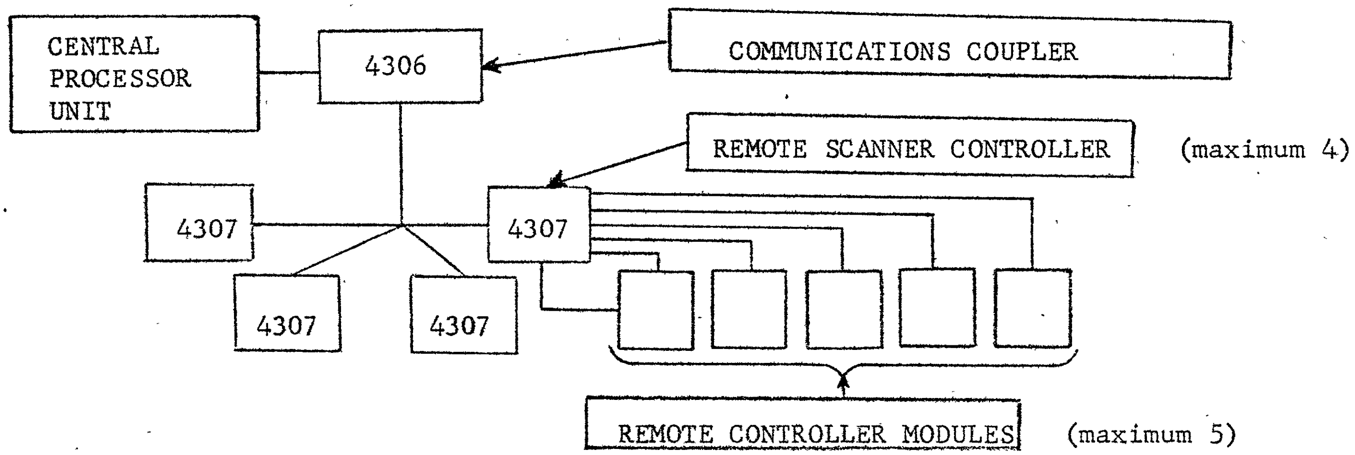
2. Digital scanning from remote Digital Input Controllers

3. Digital and analog outputs to remote Multiple Output Controllers

4. Timed contact outputs to remote Timed Output Controllers

5. Peripheral buffer inputs and outputs to remote 4201B Peripheral Buffers

FUNCTIONAL DIAGRAM



Only one of each module is allowed for each Remote Scanner Controller.

a. Model 4100 Analog Input Controller and 4130 A/D Converter.
b. Model 4400 Digital Input Controller.
c. Model 4300 Multiple Output Controller.
d. Model 4302 Timed Output Controller.
e. Model 4201B Peripheral Buffer

1.28 ANALOG INPUTS

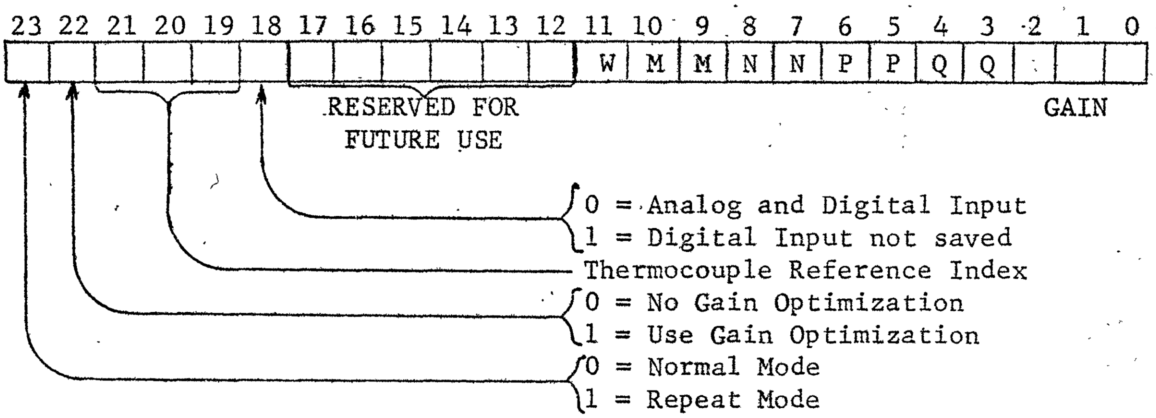
The Analog Input function operates similar to the Analog Controller operation if it were attached to the central processor except that it is slower in operation, and its scanner command format is compressed, and merged with the module address. Automatic group advance type contoller is not allowed. The scanner is single-channel successive approximation only. Matrix overload and converter overflow is handled separately and their indication is returned as non-data responses.

The system programmer has two options for each scan command word presented to Monitor, which are indicated in Bit 18.

1) Analog Input only or
2) Analog and Digital Inputs

If bit 18 is not set, the analog value is stored followed by the digital value.

ANALOG REQUEST FORMAT



OBJECTIVES

1. To input the analog count value from the scanner.
2. To store the value in the specified count table if the point is not "locked out" of scan.
3. To scale all count values to correspond with the lowest voltage scale.
4. To subtract the offset voltage (counts) from the raw count.
5. To add the thermocouple reference block temperature voltage (counts) to the raw count.
6. To gain optimize upon request when converter overflows.
7. To optionally store the time in system counts as the last word of the scan count value table.
8. To repeat a scan command output for verification of line transmission if requested.
9. To store a digital count value upon request after the analog value in the specified count table if correct expected response formats are received.
10. To notify the calling program after two consecutive transmission or non-expected response errors with a specified bit set in the A-Register.
11. To turn on a non-buffered calling program when all scans have been completed with a scanner failed code in the A-Register if required.

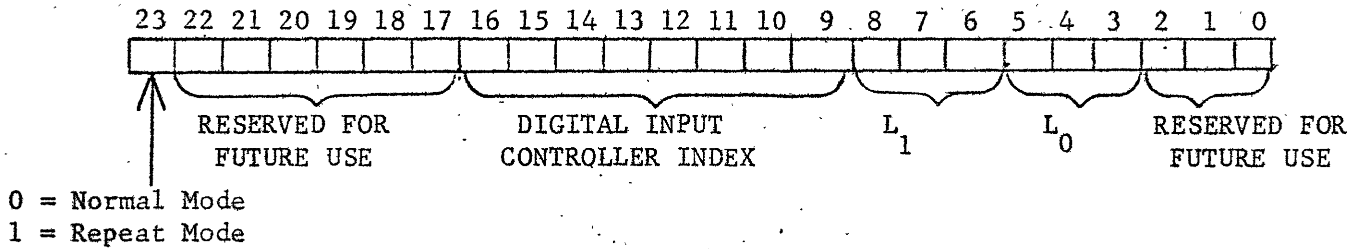
1.29 DIGITAL INPUTS

The Digital Input function must by necessity work in a different manner than when attached to a central processor due to transmission out and back. To conserve line time it operates in two modes:

1) In conjunction with the Analog Scanner
2) Separately

The number of contacts read per digital input group (L_1L_0) is reduced to eleven plus validity. The driver does not check the validity bit.

DIGITAL REQUEST FORMAT



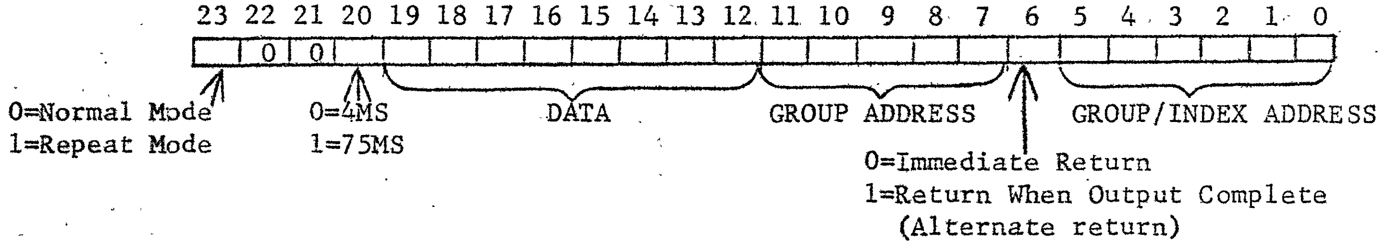
OBJECTIVES

1. To input the digital count value from the remote scanner.
2. To store the value in the program's A-Register within his register storage block.
3. To repeat a digital output for verification of line transmission if requested.
4. To turn on the calling program after digital request has been completed.
5. To notify the calling program after two consecutive transmissions on non-expected response errors with a specified bit set in the A-Register.

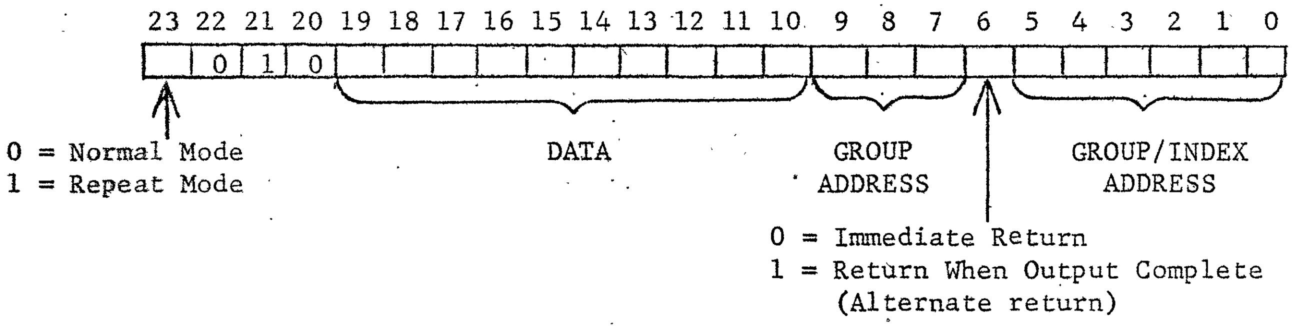
1.30 DIGITAL AND ANALOG OUTPUTS

The Analog and Digital Output function operates similar to the present Multiple Output Driver, except that the format is changed. Digital Outputs will be either eight bits per word with 32 groups addressable or 10 bits per word with eight groups addressable.

8-BIT REQUEST FORMAT



10-BIT REQUEST FORMAT



NOTE: The use and position of Bit 6 is kept the same as in the local output. The Group/Index address in Bits 0-5 has the same usage as in local outputs. If there is no status word table, this number is a Group Address. See MØR44 - Multiple Output Request Subroutine, for further information.

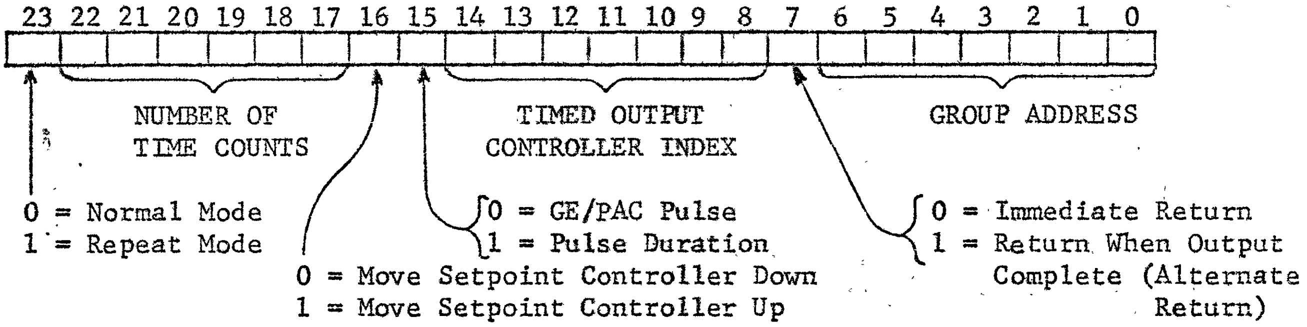
OBJECTIVES

1. To output analog, binary or decimal contact status.
2. To request a normal output only.
3. To repeat an analog, binary, or decimal contact output for verification of line transmission.
4. To turn on the calling program after output has been completed on an alternate return request.
5. To notify the calling program after two consecutive transmissions or non-expected response errors with a specified bit set in the A-Register.
6. To notify the calling program after two consecutive overload errors with a specified bit set in the A-Register.

1.31 TIMED CONTACT OUTPUTS

The timed contact output function is similar to the operation of a timed contact output controller when connected to a central processor except for reduced speed and a compressed format word merged with the module address. There are only 32 group addresses and the count is restricted to 6 Bits.

TIMED OUTPUT REQUEST FORMAT



OBJECTIVES

1. To output timed contact closures for a specified time period.
2. To repeat a timed contact output for verification of line transmission if requested.
3. To turn on the calling program after output has been completed on an alternate return request.
4. To notify the calling program after two consecutive transmissions or expected response errors with a specified bit set in the A-Register.
5. To notify the calling program after two consecutive overload errors with a specified bit set in the A-Register.

1.32 PERIPHERAL BUFFER INPUTS AND OUTPUTS

* The peripheral I/O functions operate as did the 4201B Peripheral Buffer which was connected directly to the central process, except that speeds are slower and the device types restricted. The Devices offered will be the following:

	a)	Model 4221B	Output Typer
	b)	Model 4223B	Long Carriage Typer
*	c)	Model 4270B	I/O Typer
	d)	Model 4253A	Paper Tape Punch
	e)	Model 4240B	Card Reader (70 CPM)
	f)	Model 4213A	High Speed Paper Tape Reader

Peripheral Buffer input/output Monitor requests continue to be handled in the same manner. No repeat of remote I/O requests are allowed. If the peripheral buffer is not ready, the driver will do other requests or polling until it becomes ready.

1.33 PRIORITIES AND INDEXING

Two Remote Scanner Driver entries, from interrupts, are required for each Communications Coupler. The Remote Scanner Modules have priorities within a Communications Coupler. A normal request from a higher priority module will be processed before a priority request from a lower priority module in the same communication coupler. These module Priorities must be defined at the time the Monitor is built.

Modules, both local and remote, have indices which must be specified at assembly time. The indices are for a table of pointer words. The pointer words indicate the location of a module's associated stacking tables.

Example:

Four local and two remote modules of the same type have index numbers 0-5. These index numbers are the system program's link or identification with the correct module.

Index words must also be specified at assembly time giving the 4307 each remote module is associated with.

The digital input controllers, multiple output controllers, and timed output controllers are handled in a similar manner.

A Communications Coupler communicates with only one module at a time. A two-second diagnostic count will be set to check for interrupt failure. Each Communications Coupler has the following interrupts:

DMT		NON-INHIBITABLE
TIM		NON-INHIBITABLE
SPB (TIM ECHO)	RESPONSE	INHIBITABLE (TABLE FULL)
SPB	TRANSMIT	INHIBITABLE

*Revised 12/66
**Revised 1/67

1.34 POLLING DIGITAL SCAN

This driver does a sampling (polling) of each 4307 at timed intervals. This is required to bring in change detection and I/O typer input demand. It is also required to keep the driver active while waiting for a TCØ, MØ or PB ready signal. It is accomplished by digital scan of group #00 of the Digital Controller.

If there is no digital controller on the 4307, the digital scan request to a dummy group #00 will still bring in the required information.

CHANGE DETECTION

A table of words called CHGDET is set up with one word for each Communications Coupler. The remote scanners are also assigned a priority. When a change detection occurs, a bit is set in the correct word (communications coupler) using this priority as an index. Also, a system program which is designated (CHDPRG), is turned on. This system program must then determine which remote scanner had the Change Detection, reset the bit and take the appropriate action.

Repeat

The "R" repeat bit is used in the following manner. Each remote analog input, digital input, multiple output, and timed output request, must define whether this request is to be repeated. If the "R" option is selected, the driver sets the "R" bit on its output request command. The 4307 saves and sends the output request back to the driver for verification. The driver then sends the output request again without the "R" bit, knowing the 4307 will compare this request with the saved command. If an error occurred, a non-data response reply will be returned to the driver with the error transmission bit set.

Transmission Speed

Line speed can be selected by a plug-in jumper at 150, 300, 600, 1200 and 1800 bits per second. Usually the 1800 is selected for general remote scanner application.

1.35 SYSTEM RESTRICTIONS

1. Only one I/O typer can be used for each remote scanner.
2. Only (4) Remote Scanners can be connected to a single Model 4306 Communications Coupler.
3. A maximum of (5) modules, one of each type, can be connected to a single Model 4307 Remote Scanner.
4. Only (1) peripheral or module can be accessed at a time on a given Model 4306 Communications Coupler (Party Line).
5. Each analog scan automatically gives a digital input, during the time the analog scan is being accomplished. The digital group address, however, is the same as the N.P.Q. specified for the analog point. (A program option exists, which allows the programmer the choice of saving this digital input.)
6. Only single-channel, successive approximation scanners can be used.
7. Only one digital input group can be requested in a single request by a program. The number of contact points per group is 11.
8. No repeat of remote peripheral buffer requests will be allowed.
9. No Pulsed or Timed Multiple Outputs can be used.
10. Teletypes can be used as remote output peripherals only.
11. A single system can have no more than 24 peripheral devices.

NOTE: See Remote Scanner Driver (RSD41), for more information.

1.36 COMPUTER OPERATOR SUBSYSTEM

This subsystem is functionally equivalent to that which has traditionally been known as OPR, but its organization is somewhat different and its capabilities somewhat expanded. Two distinct versions are available, one for core-only memory systems, and a second for core-bulk memory systems.

The subsystem consists of an executive function and a group of programs which are designed to perform the specific requested functions.

The executive function operates within the framework of Monitor as a functional program and is activated (turned on) by operator demand from an I/O keyboard-printer device. The executive communicates with the operator via the I/O keyboard-printer to determine which of the available functions is requested.

It then establishes communication between the requested program and Monitor so that the requested program may then continue the communication with the operator to determine the specific parameters required to perform the task requested. The executive is then free to accept requests for other functions. The number of concurrently operating functions is limited by the number of positions in the Monitor priority table made available to the Utility Subsystem.

The specific functions which are available under this subsystem are:

a. LOADING of binary data from any specified input device to any specified memory medium.
b. DUMPING of binary data from any specified memory medium to any specified non-printing output device. Output format is to be acceptable to LOADING function.
c. DUMPING of octal data from any specified memory medium to any specified printing device.
d. COMPARING binary data from any specified input device against contents of any specified memory medium with documentation of discrepancies on any specified printing device.
e. Translating octal data from any specified input device to binary data output to any specified non-printing output device. Output format is to be acceptable to LOADING function.
f. Processing Library calls in program input files. This function accepts from any specified input device the binary output of the PAL assembler. It requests input of required library subroutines and appends them to the assembled program. The output consists of binary data in format acceptable by LOADER function, and it may be directed to any specified non-printing output device.
g. MEMORY CHANGING and DOCUMENTING - This function documents and/or changes the contents of any designated memory location. Documentation of changes is in octal (version 1) or in decimal (version 2) on the same I/O keyboard-printer device through which the parameters are communicated.
h. Setting system CLOCK and CALENDAR.
i. Changing PROGRAM STATUS (On, Off, Locked Out).
j. Changing PERIPHERAL STATUS (in or out of service).

k. Media conversion - This function accepts input from any specified input device and reproduces it on a record by record basis on any specified output device. Input records must not be longer than the maximum permissible record in the specified output medium. If the input device is a keyboard, standard "delete previous character" and "delete record" characters are recognized so that the function can serve as an effective paper tape or card preparation tool.

l. Establish MEMORY PROTECTION STATUS for a designated functional program. This function provides capability for setting up memory protection parameters for a program to be debugged. It includes standard trapping subroutines for protection-violation and API stall traps.

   These subroutines document the occurrence of traps, and when feasible cause re-entry to the offending program for further execution. If re-entry is infeasible the offending program is turned off.

In addition to the basic configuration and the required memory space, hardware requirements for each of the described functions are evident from the functions themselves.

APPENDIX A

MONITOR PSEUDO-OPS

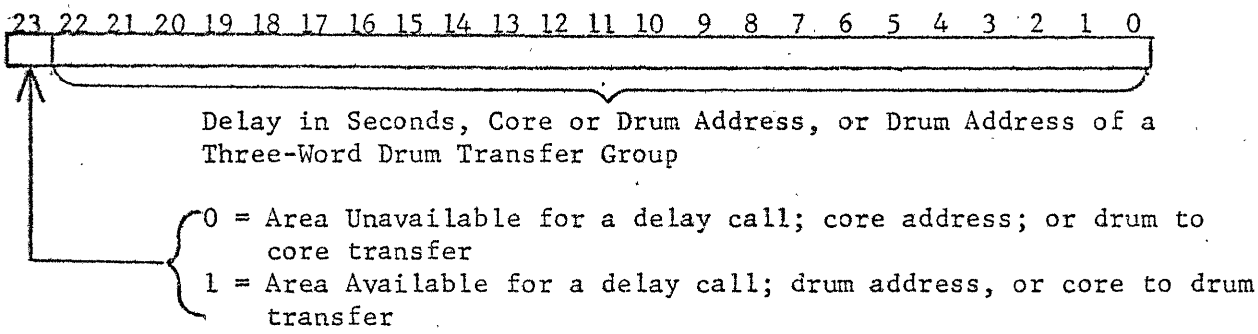
Pseudo-ops defined by MONITOR are:

BCD	- Binary to 8-Bit BCD
BCN	- Binary Character Output
CLK	- Clock
DEL	- Delay and Drum/Core Addresses
DFE	- Floating Point to E Type Floating Point Decimal
DFP	- Floating Point to Fixed-Point Decimal
DFX	- Binary to Fixed-Point Decimal
FBB	- Binary to 4-Bit BCD
FØR	- Drum/Disc Controller Index and Number of Word in bulk request
INF	- Input Request Information word
ØCT	- Binary to Octal (integers)
ØUF	- Output Availability Check and Information Word
PRG	- Maintains REGSTG and RSX Tables of ECP
SZE	- Size of Program

The BCD, BCN, CLK, DFE, DFP, DFX, FBB, and ØCT pseudo-ops are used with the Output Program as format words. Refer to the Output Program, subcategories of 1.14, for explanation of these pseudo-ops. The DEL, FØR, INF, PRG, ØUF, and SZE pseudo-ops are used in subroutine calls and are explained below:

The DEL pseudo-op is used in the Scan Request, Input Request, Output Request, 1st word of the 3-word Drum Transfer Request, and Set Program Delay Subroutine Calls. Coding examples and format are as follows:

```
              DEL      0, 15*SECND                  DEL         1,FMTB
```



Delay in Seconds, Core or Drum Address, or Drum Address of a Three-Word Drum Transfer Group

0 = Area Unavailable for a delay call; core address; or drum to core transfer

1 = Area Available for a delay call; drum address, or core to drum transfer

The FØR pseudo-op is used to specify the drum/disc controller index and the number of words in a DTRC01 or DTRC02 request.

```
              FØR  0,/300              FØR PDISC, WTØTAL
```

23 22 21 20 19 18 17 16 15	14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Drum/Disc controller index	Number of words to transfer

The INF pseudo-op is used in Input Request. A coding example and format definition follows:

INF   0, 1, 1, 4, 25

*

23	22	21	20	19	18 14	13 0
0	0	0	1	1	0 0 1 0 0	0 0 0 0 0 0 0 0 0 1 1 0 0 1

Bit 22 indicates that the input mode is symbolic characters. (Set indicates binary)
Bit 20 indicates that packing will occur.

*

Device	Packing Mode	
	Binary Input	Symbolic Input
Paper Tape	1,4	1,3
Cards	1,2	1,3
Keyboard	NA	1,3

(Reset indicates one per word packing.)

Bit 19 indicates paper tape input. (Reset indicates keyboard input. This bit has meaning for teletypes only.)
Bits 14-18 is the input device priority number.
Bits 0-13 are the maximum number of characters to be read.

The ØUF pseudo-op is used in Output Availability Check and Output Request. A coding example and format definition follows:

ØUF 0, 1, 0, 0, 0, 3, 0, 0, 17

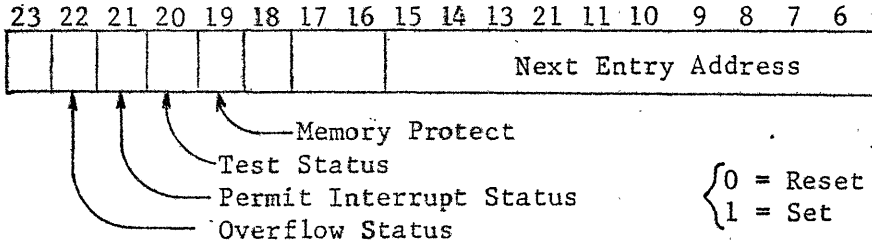
23	22	21	20	19	18 14	13	12 10	9 5	4 0
0	1	0	0	0	0 0 0 1 1	0	Reserved for future use	0 0 0 0 0	1 0 0 0 1

Bit 23 indicates no page control is desired for this message.
Bit 22 indicates EIA - RS 244 output conversion is desired. (Reset for ASCII/CPC)
Bit 21 indicates that a data area is not used in this message.
Bit 20 indicates that the area availability status will not be changed. (Set indicates delay in opposite of present stauts)
Bit 19 indicates that the message is to be typed. (Set for punching. Has meaning only on teletypes.)
Bit 14-18 indicates the device priority number.
Bit 13 indicates this message is output as symbolic characters. (Set indicates binary)
Bits 5-9 is the bulk controller index to the data words.
Bits 0-4 is the bulk controller index to the format words.

*Revised 12/66

The PRG pseudo-op is used to maintain the REGSTG and RSX Tables of ECP and by the Turn Program Off Subroutine. It contains the flip-flop status of overflow, permit interrupt, test, next entry location and memory fence.

PRG 0, 1, 0, START, 0



For a Drum/Core system, if the next entry address is set to zero, the program begins at its first location.

The SZE pseudo-op specifies the required parameters for the second word of the three-word ECP Drum Transfer Group and the second word of the three-word save status tables.

ECP
Drum Transfer Group

SZE 1,0,0,0,/400

(This psuedo-op tells the area availability status, Bit 23; Fortran status, Bit 19; fixed core status, bit 18; drum/disc controller index bits 17-15; and size of program, bits 13-0

	23	22	21	20	19	18	17——15	14	13——0
SIZE	A	C	T	N	F	W	Drum/Disc Controller Index Number	0	*Size of Program

A - Area availbility on entry from ECP
0 = unavailable
1 = available
C - Core status
0 = program not in core
1 = program in core
T - Transfer Status
0 = program not in transfer
1 = program in transfer or requested DTRC02 transfer to or from bulk

* For programs with adjacent save status, this size figure includes save status and programs. For programs with disjoint save status, this size figure is for program only.

N - Current Area Status
0 = program running with core area unavailable
1 = program running with core area available
F - FORTRAN Available Subroutine Usage Status
0 = program does not use subroutine while running available
1 = program uses subroutine while running available
W - Fixed Working Core Status
0 = Program can run anywhere in working core
1 = Program must run from fixed working core area.

APPENDIX B

COMMUNICATION CALLS

		Page
DELC01	Set Program Delay for A Specified Time Period	10
DELC02	Sets Program Area Unoccupied and Available During Delay	10
DTRC01	Drum Transfer Request (Immediate Return)	13
DTRC02	Drum Transfer Request (Drum Transfer Completed Return)	13
FMRC01	Find a Working Core Area	66
FRPC01	Find Register Pointer	14
FSSC01	Find Save Status Pointer	14
INRC01	Input Request	36,39,40,41,42
INRC02	Read Completed Request	36,38,40,41,42,43
MAP01	Core Map Maintenance - Set Area Occupied	12
MAP02	Core Map Maintenance - Set Area Unoccupied	12
MAP03	Core Map Maintenance - Set Area Unavailable	12
MAP04	Core Map Maintenance - Set Area Available	12
MØRC01	Normal Non-Priority Multiple Output Request	48,50
MØRC02	Normal Priority Multiple Output Request	48,50
MØRC03	Timed Non-Priority Multiple Output Request	51
MØRC04	Timed Priority Multiple Output Request	51
MØRC05	Unlatch A Previous Timed Request	51
MØRC06	Pulsed Non-Priority Multiple Output Request	52
MØRC07	Pulsed Priority Multiple Output Request	52
ØFFC01	Turn Program Off Request	9
ØFFC02	Set Program Area Unoccupied and Available During the Time Program Is Turned Off	9
ØUCC01	Output Availability Check	16,17,18,19
ØURC01	Output Request	20
RDSC01	Remote Digital Input Request	63
RMRC01	Release A Working Core Area	66
SCRC06	Normal Non-Buffered Scan Request	56
SCRC07	Priority Non-Buffered Scan Request	56
SCRC08	Normal Buffered Scan Request	56
SCRC09	Priority Buffered Scan Request	56
SCRC05	Scan Completed Request	56
TCØC01	Normal Timed Contact Output Request (Local)	54
TCØC02	Priority Timed Contact Output Request (Local)	54
TCØC03	Normal Timed Contact Output Request (Remote)	54
TCØC04	Priority Timed Contact Output Request (Remote)	54
TPNC01	Turn Program On Request	11
TPNC02	Turn Program On Request If Program Is Currently Turned Off	11
TPNC03	Specifies Next Running Program.	11

APPENDIX C

MONITOR SYMBOLS

The following symbols are defined by Monitor, and should not be defined by the programmer:

ALERT - Output Peripheral Activate Flag

The peripheral device priority number corresponds to the associated bit.

Example: Device #3 = Bit 3 of Flag

ALTFLG - No Alternate Device Flag

This flag shows that an output message for the requested output peripheral has no working alternate, therefore, the output message was destroyed.

Example: Device priority number i corresponds to bit i.

ALTTBL - Primary Alternate Device Priority Numbers for Peripheral Devices. The alternate for device priority number i is found in ALTTBL+i.

Example: Priority Device #5 = ALTTBL+5

ALTTB2 - Secondary Alternate Peripheral Substitution Table. Provides another alternate table in the event a secondary alternate peripheral device is required.

AUXTM - Table of Auxiliary Time Counters

AVLMAP - Available Area Map

BAD - Peripheral Device Failure Flag

The Peripheral device priority number corresponds to the associated bit.

Example: Device #4 = Bit 4 of Flag

CØRMAP - Occupied Area Map

DMCRNØ - Three Times the Running Program Number

DTAREA - Data Area Bit Word

Area$_i$ corresponds to Bit $_i$.

Example: Area 2 = Bit 2

Area 3 = Bit 3

DVCØDE - Device Codes

A table of hardware addresses and classification codes for peripheral devices arranged in descending order.

Example: #0 - Highest Priority Device

EOMFG - A one-word flag indicating that end-of-message has been reached for device i.

FAILUR - Peripheral or Hardware Failure Device Flags

MESSFG - A one-word flag indicating the availability of the output buffer for device i.

ØØS - Peripheral Device Out-of-Service Flag

PRIØNØ - One Times the Running Program Number

PRØCFG - A one-word flag set when the first buffer of a message is ready for output.

PRØG - Program Execution Times

The table starts with Program No. 1.

PRØGNØ - Eight Times the Running Program Number

SECND - One second in real-time counts: 1 = One second system; 2 = One-half second system; 4 = One-quarter second system, etc.

STSMAP - Occupied Save Status Area Map

TIME - Time in System Counts (Cleared at Midnight)

XFER - Number of Drum Transfer Requests Waiting for a Program

This table is referenced by program number.

Example: Program 7 = XFER+7

APPENDIX D

MONITOR ASSEMBLY CHECKLIST

In building a Monitor system, the programmer must define the specific parameters and system options. The following checklists are used to specify these parameters. When the checklists are completed, send them accompanied by the interrupt assignment for the system to the Programming Librarian.

By judicious selection of system options, it is possible to acquire a Monitor which contains only those capabilities that are needed for each application. The size of Monitor varies with the combination of options selected.

Additional copies of the Checklist may be obtained upon request from the Programming Library.

Library File No.________
Programmer's Name________
Date________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
1. IS YOUR SYSTEM FOR (Check One) □ 1. (GE/PAC 4020) Drum/Disc Core □ 2. (GE/PAC 4020) All-Core □			
2. QUASI'S - Check one or more depending on the system's need. Single-word floating point □ Double-word floating point □ Single/Double-word floating point □ Is floating point being used in your system? YES____ NO____ If yes, is hardware floating point being used in your system? YES____ NO____			
3. NUMBER OF COUNTS USED BY SYSTEM PROGRAMS	AA	EQL	
Number of System Program to be turned on when a system count becomes negative.	APRØGM	EQL	
This program must be written by the System Programmer Option count. System calendar updating YES____ NO ____			
4. TOTAL NUMBER OF FUNCTIONAL PROGRAMS (Include Monitor Programs listed in 11.)	B	EQL	
* 5. NUMBER OF FUNCTIONAL PROGRAMS HAVING SAVE STATUS.	BB	EQL	
6. NUMBER OF 8-WORD REGISTER STORAGE BLOCKS All Monitor Programs require 8-word register storage blocks.	BBB	EQL	

* Revised 1/67

Library File No. __________
Programmer's Name __________
Date __________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
7. STARTING ADDRESS OF THE FLOATING WORKING CORE AREA (Starts after MONITOR and system permanent core and must be a multiple of 100_8.)	BASE	EQL	/
* 8. A. MAXIMUM CORE ADDRESS	MAXC	EQL	/
B. BULK CONTROLLER DATA:			
1) Bulk Controller 1:			
a. Controller Address	BULK1	EQL	/
b. Pointer Address	BULK1P	EQL	/
c. Maximum Address	MAXD1	EQL	/
d. Check which bulk device is used:			
Drum_____ Disc_____			
If disc:			
State number of drives:	DRIVE1	EQL	
Is Seek Ahead Option desired?			
YES _____ NO_____			
2) Bulk Controller 2:			
a. Controller Address	BULK2	EQL	/
b. Pointer Address	BULK2P	EQL	/
c. Maximum Address	MAXD2	EQL	/
d. Check which bulk device is used:			
Drum_____ Disc_____			
If disc:			
State number of drives:	DRIVE2	EQL	
Is Seek Ahead Option desired?			
YES _____ NO _____			
C. ARE BUFFERED BULK CALLS (DTRC03,DTRC04) DESIRED?			
YES _____ NO _____			

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
9. NUMBER OF ON-LINE PERIPHERAL DEVICES Peripheral devices are typewriters, printers, card readers, card punches, paper tape punches, magnetic tapes, etc. EACH I/O DEVICE REQUIRES TWO DEVICE NUMBERS.	E	EQL	
10. STACKING TABLES, one per each peripheral device number. One stacking table is required for each peripheral device (printer, magnetic tape, typewriter, or punch). The size of each stacking table must be a power of two (F_i). The number of requests in each stacking table represents the number of concurrent outputs that can be requested on a peripheral. Each request requires four entries. The actual size of the stacking table is equal to F_i. For four concurrent requests on Device 2, the stacking table must have sixteen locations. The following peripherals (input) require no stacking tables: (Assign zero for these peripherals). Card Readers Paper Tape Readers Keyboard Input Recommended sizes for the other stacking tables are: Paper Tape or Card Punches - 8 Printers 16 I/O Typers 16 Output Typers 16 or 32 Magnetic Tapes 8 Data Edit Displays 8	F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12} F_{13} F_{14} F_{15}	EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL EQL	
11. PROGRAM NUMBERS			
* Output Program	ØUP	EQL	
* Scan Offset Program	SCF	EQL	
Input Program	INP	EQL	
Corrective Action Program	CAD	EQL	
* Keyboard/I/O Typer Operator Program	ØPX	EQL	

* Revised 1/67

Library File No. ______
Programmer's Name ______
Date ______

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
* 12. FREE-TIME SYSTEM Is Free-Time System desired? YES ___ NO ___ If so, refer to the Free-Time System checksheet.	FTX	EQL	
13. SIZE OF SYSTEM INITIALIZATION Do not include Monitor initialization area.	SINZTL	EQL	
14. NUMBER OF 50 OR 60-CYCLE PULSES IN ONE TIME COUNT INTERVAL This EQL defines the length of the time count interval for the system. The system time may be kept in seconds, multiples of seconds, or fractions of seconds. For example, in a 60-cycle system a time count interval of 1/4 seconds, NCYCLE would be fifteen. NCYCLE would be sixty for a one-second interval. Check 50 or 60-Cycle Pulse Required 50-CYCLE ___ 60-CYCLE ___	NCYCLE	EQL	
* 15. OUTPUT PROGRAM INFORMATION A. Number of floating data areas	DTANMB	EQL	
B. Drum - disc address of floating area or core address on all-core systems	DTFBSE	EQL	
C. Sizes of floating data areas (Multiple of 64)	DTFSZE	EQL	
D. Number of data area (Multiple of 24)	DTØTAL	EQL	
16. ECP OPTIONS A. REGISTER POINTER TABLE YES ___ NO ___ Check this option if there are programs within your system which have single word register storage or 8-word shared register storage. B. PRIORITY TABLE OPTION YES ___ NO ___ If program priority changing is desired, this table must be included in your Monitor. DOES YOUR SYSTEM HAVE FIXED WORKING CORE AREAS? YES ___ NO ___			

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
* 17. TURN PROGRAM ON SUBROUTINE CALL YES ____ This optional call, TPNC03, NO ____ specifies which program is to run next.			
18. UNOCCUPIED, AVAILABLE OPTION FOR SET PROGRAM DELAY SUBROUTINE DELC02 Call YES ____ NO ____			
* 19. CORRECTIVE ACTION DIAGNOSTIC PROGRAM A. Priority Number of Device for Operator Messages B. Should corrective action messages be printed in red? YES ____ NO ____ C. Is special Multiple Output corrective action required? YES ____ NO ____ Note: If special action is required, define your system needs.	DTYPER	EQL	
** 20. OUTPUT PAGE CONTROL Is the Page Control Option desired for printing devices? YES ____ NO ____ If so, furnish:			

Device Priority Number	Lines per Page	Lines to Top of Next Page

The lines per page is usually 50; lines to the top of the next page is usually 10.

* Revised 1/67
** Added 1/67

Library File No. ______________
Programmer's Name ______________
Date ______________

SYSTEM PARAMETERS (All EQL's must have an assigned value)

* 21. DEVICE CODES - Local and Remote

Monitor Priority Number	R=REMOTE L=LOCAL	Device Type @	If input device, give associated program @@	If Local, P.B. Hardware Address. If Remote, C.C. Hardware Address.	18-Character Device Name for CAD	Device Address	Alternate Device		FORTRAN SUBMONITOR NUMBER
							#1	#2	
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									

@ Examples:

I/O Typer	= I/O	Card Punch	= CP
Paper Tape Punch	= PTP	Keyboard Send/Receive Teletypewriter	= KSR
Card Reader	= CR	Automatic Send/Receive Teletypewriter	= ASR
Paper Tape Reader	= PTR	Receive Only Teletypewriter	= RO
Fixed Carriage Output	= OP	Long Carriage Typer	= LC

@@ This is the program which will be turned on if the Input Demand/Break Key is pressed. If no program is associated with the Input Demand/Break Key, place an N/A for the device.

* Revised 1/67

SYSTEM PARAMETERS (ALL EQL's must have an assigned value.)	SYMBOL	EQL	VALUE
* 21. DEVICE CODES - Local and Remote - Contd Priority levels must be established for each peripheral device. Devices which must be activated as soon as possible upon request are placed in the highest priority. Each I/O requires two consecutive priority numbers with output first. The 18-character device name is used for alarming. If the FORTRAN Submonitor is used in the system, fill in its device number. FORTRAN peripherals are divided into three categories: Input, Printing, and Punching Peripherals. Each category begins with Device Number 0.			

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
1. ON-LINE COMPUTER OPERATOR SUBSYSTEM a. Device Priority Number of Input Devices which may communicate with the subsystem and their output devices. The following programs may be called using the Computer Operator Subsystem: * b. Number of Program Priority Numbers to be controlled by the Operator Executive. * c. Number of programs to be run as ØPR programs		ØPRSUB ØPRNMB		
* A. Octal Memory Change - changes and/or documents memory. Check if punch option required.	YES NO ___ ___			
B. Binary Loader - enters new programs or data				
C. Binary Dump - punches memory contents.				
D. Printed Dump - Prints memory contents.				
* E. Clock - resets the system time and calendar and updates the program execution times and auxiliary time counters. Check if the calendar option required.	YES NO ___ ___			
F. Program Status - On/Off/Lockout - initiates, stops or locks out the execution of functional programs under MONITOR control.				
G. Peripheral In/Out of Service - removes peripheral devices from services or restores failed or out-of-service peripherals to service.				

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
* 2. AUXILIARY TIME COUNTERS (Number of Auxiliary Time Counters) This option must be chosen if there are functional programs in the system which initiate themselves at regular time intervals and are also initiated by system or operator demands. The regular time interval should be saved in an auxiliary time counter.		C	EQL	
* 3. FLOATING POINT OUTPUT Select this option for making calculations in floating-point and converting to decimal fixed point or to code in FORTRAN.				
4. BINARY E TYPE FLOATING POINT OUTPUT Select this option for converting binary floating point data to decimal floating point.				
5. DECIMAL FIXED-POINT OUTPUT This option converts binary data to fixed-point decimal format for output.				
6. OCTAL OUTPUT This option must be used to output data in octal format.				
7. ALPHA-NUMERIC CHARACTER OUTPUT (8-Bit BCD) If alphanumeric messages are required using 8-Bit BCD data (three or one character (s) per word), this option must be checked.				

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
8. BINARY CHARACTER OUTPUT				
9. TIME OUTPUT Select this option to convert system time counts to hours and minutes or hours, minutes and seconds in decimal.				
10. 4-BIT BCD OUTPUT This option converts binary data to four-bit BCD information.				

Library File No. ____________
Programmer's Name ____________
Date ____________

* SYSTEM OPTIONS

11. MULTIPLE OUTPUT DISTRIBUTOR

MØD Controller Class Index No. @	L=LOCAL R=REMOTE	NNDTSi NORMAL/TIMED Non-Priority Driver Table Size @@	NPDTSi NORMAL/TIMED Priority Driver Table Size @@	PNDTSi Pulsed Non-Priority Driver Table Size @@@	PPDTSi Pulsed Priority Driver Table Size @@@
0					
1					
2					
3					
4					
5					
6					
7					

@ Example of Class Index Number:

Four local and two remote MØD controllers will have Index Numbers 0-5.
These index numbers are the system programs' link or identification with the correct MØD.

@@ NORMAL/TIMED Driver Tables require two locations for each request.

@@@ Pulsed Driver Tables require four locations for each request.

* Revised 1/67

SYSTEM OPTIONS

* 11. MULTIPLE OUTPUT DISTRIBUTOR - Contd

Library File No. ____________

Programmer's Name ____________

Date ____________

List the Groups for each MØD that will have a current status word:

MØD Controller Index No.	Group Numbers
0	
1	
2	
3	
4	
5	
6	
7	

			IF LOCAL		
MØD Controller Class Index No.	MØDi Hardware Address; If Local, MØD Controller. If Remote, Communication Coupler.	MØDGPi Total No. of Groups	PULØCi Pulsed DMT Interrupt Location	TMLØCi Timer DMT Interrupt Location	TMNBRi Maximum No. of Concurrent Timed Requests Being Processed @@@
0					
1					
2					
3					
4					
5					
6					
7					

@@@ Each Timer Request requires three locations in a table.

Maximum number of timers = 24 per controller

* Revised 1/67

SYSTEM OPTIONS			Library File No.			
* 12. TIMED CONTACT OUTPUT			Programmer's Name			Date
TCØ Controller Class Index @	R=REMOTE L=LOCAL	Number of TCØ Groups On Controller	If Local, Controller Address. If Remote, C.C. Hardware Address.	Size of Driver Table @@ Priority	 Normal	For Use by Library Only
Number 0						LR00T EQL CC00T A00T TØNP00 TØNS00 TØPP00 TØPS00
Number 1						LR01T EQL CC01T A01T TØNP01 TØNS01 TØPP01 TØPS01
Number 2						LR02T EQL CC02T A02T TØNP02 TØNS02 TØPP02 TØPS02
Number 3						LR03T EQL CC03T A03T TØNP03 TØNS03 TØPP03 TØPS03

@ These class index numbers are the system programs' link or identification with the correct TCØ.

Example of TCØ Index Numbers:

Four local and two remote TCØ Controllers will have index class numbers 0-5.

@@ Each TCØ request requires two locations in a Driver Table.

* Revised 1/67

SYSTEM OPTIONS	Library File No.
* 12. TIMED CONTACT OUTPUT - Contd	Programmer's Name Date

TCØ Controller Class Index @	R=REMOTE L=LOCAL	Number of TCØ Groups On Controller	If Local, Controller Address. If Remote, C.C. Hardware Address.	Size of Driver Table @@		For Use by Library Only
				Priority	Normal	
Number 4						LR04T EQL CC04T A04T TØNP04 TØNS04 TØPP04 TØPS04
Number 5						LR05T EQL CC05T A05T TØNP05 TØNS05 TØPP05 TØPS05
Number 6						LR06T EQL CC06T A06T TØNP06 TØNS06 TØPP06 TØPS06
Number 7						LR07T EQL CC07T A07T TØNP07 TØNS07 TØPP07 TØPS07

@ These class index numbers are the system programs' link or identification with the correct TCØ.
Example of TCØ Index Number:
Four local and two remote TCØ Controllers will have index class numbers 0-5.

@@ Each TCØ request requires two locations in a Driver Table.

* Revised 1/67

SYSTEM OPTIONS	Library File No.	
* 13. ANALOG SCANNER	Programmer's Name	Date

Scanner Class Index @	R=REMOTE L=LOCAL	Amplifier Type	If Local, 3 API or 1 API @@	If Local, Group Mode of Hardware	If Local- Controller Address. If Remote- C.C. Hardware Address.	Size of Driver Table @@@		For Use by Library Only	
						Priority	Normal		
Number 0								LR00S API00 CC00S A00S SNNS00 SNNP00 SNPS00 SNPP00	EQL
Number 1								LR01S API01 CC01S A01S SNNS01 SNNP01 SNPS01 SNPP01	EQL
Number 2								LR02S API02 CC02S A02S SNNS02 SNNP02 SNPS02 SNPP02	EQL

@ These class index numbers are the system programs' link or identification with the correct scanner.
Example of Scanner Index Number:
Four local and two remote analog scanners will have Index Class Numbers 0-5.

@@ One API Scanning allows a priority scan to interrupt a normal scan in progress.

@@@ Each analog request requires one location in a Driver Table.

* Revised 1/67

SYSTEM OPTIONS | Library File No.

* 13. ANALOG SCANNER - contd | Programmer's Name | Date

Scanner Class Index @	R=REMOTE L=LOCAL	Amplifier Type	If Local, 3 API or 1 API @@	If Local, Group Mode of Hardware	If Local- Controller Address. If Remote- C.C. Hardware Address.	Size of Driver Table @@@		For Use by Library Only
						Priority	Normal	
Number 3								LR03S EQL API03 CC03S A03S SNNS03 SNNP03 SNPS03 SNPP03
Number 4								LR04S EQL API04 CC04S A04S SNNS04 SNNP04 SNPS04 SNPP04
Number 5								LR05S EQL API05 CC05S A05S SNNS05 SNNP05 SNPS05 SNPP05

@ These class index numbers are the system programs' link or identification with the correct scanner.
Example of Scanner Index Number:
Four local and two remote analog scanners will have Index Class Numbers 0-5.

@@ One API Scanning allows a priority scan to interrupt a normal scan in progress.

@@@ Each analog request requires one location in a Driver Table.

* Revised 1/67

*

SYSTEM OPTIONS					Library File No.			
13. ANALOG SCANNER - Contd					Programmer's Name			Date
Scanner Class Index @	R=REMOTE L=LOCAL	Amplifier Type	If Local, 3 API or 1 API @@	If Local, Group Mode of Hardware	If Local- Controller Address. If Remote- C.C. Hardware Address.	Size of Driver Table @@@		For Use by Library Only
						Priority	Normal	
Number 6								LR06S EQL API06 CCQ6S A06S SNNS06 SNNP06 SNPS06 SNPP06

@ These class index numbers are the system programs' link or identification with the correct scanner.
Example of Scanner Index Number:
Four local and two remote analog scanners will have Index Class Numbers 0-5.

@@ One API Scanning allows a priority scan to interrupt a normal scan in progress.

@@@ Each analog request requires one location in a Driver Table.

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

SYSTEM OPTIONS

* 13. ANALOG SCANNER - contd

Options apply to all scanners in the system

For Use by Library Only

1. Options: Add the Reference Block Temperature for thermocouples.
YES _____ NO _____

2. Store time of day as the last word of the Count Table.
YES _____ NO _____

3. Gain Optimization (used only for a single-channel single API Scanner)
YES _____ NO _____

4. Is buffered scanning desired? YES _____ NO _____

Buffered requests permit a functional program to process one set of count values while the Driver is scanning another set.

5. What is the maximum number of analog scan requests that can be stacked at one time? ________________________

SNPRØG EQL
TØTSCN EQL

This can be a maximum of 24 and never needs to be more than the number of system programs that can request analog scanning because a program can only request one scan at a time.

6. Is priority scanning required? YES _____ NO _____

* Revised 1/67

Library File No. ____________
Programmer's Name ____________
Date ________________

SYSTEM OPTIONS	CHECK	SYMBOL	EQL	VALUE
* 14. FORTRAN SUBMONITOR Check this option to communicate to MONITOR through FORTRAN. Check the desired options: Input: a. Decimal to Binary □ b. Alphanumeric □ c. Octal □ Output □ a. Printed □ b. Punched □ I/O Availability □ Drum Read/Write □ Subroutine Linkage □ Computed GO TO □ Assigned GO TO □ If the free-time system is required, all options must be included..				
15. INTERRUPTABLE SYSTEM SUBROUTINES If there are subroutines within the system which must run with interrupt permitted (due to length) and are used by more than one functional program, they must follow special entry and exit conventions. When this is the case, this option must be selected.				

* Revised 1/67

Library File No. ______________
Programmer's Name ______________
Date ______________

SYSTEM OPTIONS

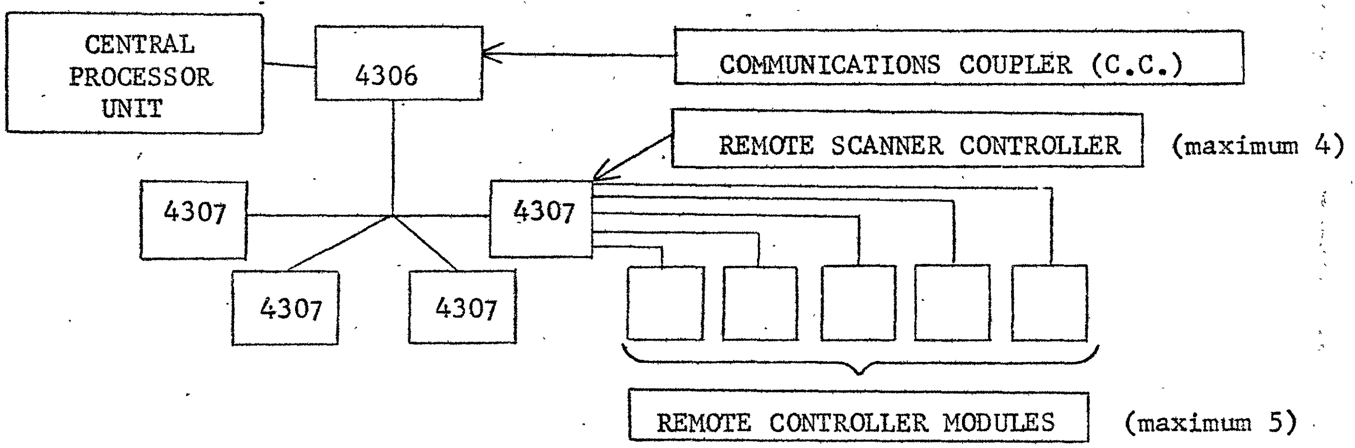
		INITIAL F/F STATUS												
ROG. RIORI-Y NO.	NAME	O V R F	P A I F	T S T F	T M F F	DRUM/DISC ADDRESS	@ (a) AREA STATUS	@ (b) SAVE STATUS	# WORDS OF PROG.	SAVE STATUS AREA ADDRESS	# WORDS OF SAVE STATUS AREA	@ (c) REG. STORAGE	@ (d) ITC & DTD INTERRUPT RETURN	CHECK IF PROGRAM RESIDE IN PERM. CORE@@
1														
2														
3														
4														
5														
6														
7														
8														

@ Place a checkmark (✓) in the corresponding column marked with an "@" sign if:

a) Area status is available
b) Save status is required
c) 8-word register storage is required
d) Interrupt Time Counter and Drum/Disc Transfer Complete Interrupts return to ECP.
@@ Applies to Drum/Disc Systems. Insert starting core address in the drum/disc address column for these programs.

REMOTE SCANNER

* 1. REMOTE SCANNER PACKAGE


CENTRAL PROCESSOR UNIT
4306
COMMUNICATIONS COUPLER (C.C.)
REMOTE SCANNER CONTROLLER
(maximum 4)
4307
4307
4307
4307
REMOTE CONTROLLER MODULES
(maximum 5)

Only one of each module will be allowed for each Remote Scanner Controller

a. Model 4100 Analog Input Controller and 4130 A/D Converter
b. Model 4400 Digital Input Controller
c. Model 4300 Multiple Output Controller
d. Model 4302 Timed Output Controller
e. Model 4201B Peripheral Buffer

Devices Allowed on Peripheral Buffer Module:

1) Fixed Carriage Output Typer, Model No. 4221B.
2) Long Carriage Output Typer, Model No. 4223B.
3) I/O Typer (one per P.B.), Model No. 4270B.
4) Paper Tape Punch, Model No. 4253A.
5) Card Reader (70 CPM), Model No. 4240B.
6) Paper Tape Reader, Model No. 4213B.
7) Teletypewriter, Model No. 4233A.

All modules have class index numbers. This relates the controller or buffer to other controllers or buffers of the same type. These index numbers are the system programs' link or identification with the correct controller or buffer.

Example: Two local and three remote analog scanners will have Class Index Numbers 0-4 in the analog scan section of the Monitor checklist.

A further explanation of the terms used in the checklist is given in the following program writeups:

YPEA7X - Remote Scanner Driver (RSD41)
YPT86X - Analog Scan Request (SCR45)
YPED5X - Multiple Output Request (MØR46)
YPED6X - Timed Output Request (TCØ42)
YPEA6X - Remote Digital Input Request (RDS41)

* Added 1/67

Library File No.

REMOTE SCANNER Programmer's Name Date

* 1. REMOTE SCANNER PACKAGE - contd

Communications Coupler Index Number @ ____________
Hardware Address ____________
Number of Remote Scanners on this Communications Coupler ____________

Priority of Module On This Communications Coupler @@	TYPE MODULE	Remote Scanner Address L_3	Class Index Number of Module	For Use By Library Only
# 0	Polling			CC-POO EQL
1	Polling			01
2	Polling			02
3	Polling			03
4				04
5				05
6				06
7				07
8				08
9				09
10				10
11				11
12				12
13				13
14				14
15				15
16				16
17				17
18				18
19				19
20				20
21				21
22				22
23				23

@ Every Communications Coupler must be assigned an index number in relationship to the other Communications Couplers.

Example: If there are three Communications Couplers in a system their Index Numbers will be 0, 1 and 2.

A copy of this page is required for each Communications Coupler in the System.

@@ Types of modules on a Communications Coupler are:

1. Polling (digital input)
2. Analog Scanner
3. MØ
4. TCØ
5. Peripheral Buffer
6. Digital Scanner

List all the modules on this Communications Coupler in priority order you would like to have them serviced. The Remote Scanner number has no bearing in determining this priority.

The class index number of polling is the same as Digital Input of the same Remote Scanner.

* Added 1/67

Library File No. ____________
Programmer's Name ____________
Date ____________

* REMOTE SCANNER 1. REMOTE SCANNER PACKAGE - contd	For Use By Library Only
1. Does system have Change Detection? YES ____ NO ____	
2. What is program number of Change Detection? ________________________	CHDPRG EQL
3. At what interval is Polling required? (suggest one second) ____________	PØLL EQL
Polling time must be in multiples of system time. Minimum polling time one-half second system = one second. Minimum polling time one-quarter second system = one-half second. Maximum System Time = ½ Second	
4. At what interval is remote Module back-in service initialization required? ______ ________________________	ØNMDTM EQL
Minimum Initialization Time = one minute (must be in multiples of one minute)	
5. Program Number of Remote Corrective Action. ________________________	CAR EQL

* Added 1/67

Library File No.

* REMOTE SCANNER　　Programmer's Name　　Date

2. REMOTE DIGITAL SCANNER

Class Index @	Communications Coupler Hardware Address	Size of Scanner Driver Table @@	For Use By Library Only
Number 0			CC00D EQL A00D DNNS00 DNNP00
Number 1			C001D EQL A01D DNNS01 DNNP01
Number 2			CC02D EQL A02D DNNS02 DNNP02
Number 3			CC03D EQL A03D DNNS03 DNNP03
Number 4			CC04D EQL A04D DNNS04 DNNP04
Number 5			CC05D EQL A05D DNNS05 DNNP05
Number 6			CC06D EQL A06D DNNS06 DNNP06
Number 7			CC07D EQL A07D DNNS07 DNNP07

@ Class Index Number is for Remote Digital Scanners only.

@@ Each digital request requires two locations in a Driver Table.

* Added 1/67

Library File No. ______________
Programmer's Name ______________
Date ______________

* REMOTE SCANNER

3. REMOTE PERIPHERAL BUFFERS

Class Index @	Communications Coupler Hardware Address	Mtr. Priority Numbers of Devices on This P. B.
Number 0		
Number 1		
Number 2		
Number 3		
Number 4		
Number 5		
Number 6		
Number 7		

@ Class Index Number is for Remote Peripheral Buffers only.

* Added 1/67

APPENDIX E

AUDIT CODES

* Following is a list of Monitor Audit Codes and the Pseudo-Ops they define. For a more detailed discussion of Pseudo-Op subroutines, refer to the appropriate Monitor Manual section shown opposite each mnemonic. See Appendix C of Process Assembler Language Manual for additional information relative to Audit Codes.

Section/Page Reference	Pseudo-Op	
1.14.3	BCD	*DEF /40000000, 63, 30, 28, 29, 32, 31
1.14.4	BCN	*DEF /50000000, 30
1.14.5	CLK	*DEF /60000000, 63, 25, 26, 28, 29, 31
1.4	DEL	*DEF /00000000, 16, 9, 0, 0
1.3	DFE	*DEF /10000000, 63, 38, 37, 28, 34, 33, 32, 31
1.3	DFP	*DEF /00000000, 63, 38, 37, 36, 35, 34, 33, 32, 31
1.3	DFX	*DEF /20000000, 63, 38, 37, 28, 34, 33, 32, 31
1.3	FBB	*DEF /70000000, 63, 38, 37, 34, 32, 31
76	FØR	*DEF /00000000, 27, 3, 0, 0
77	INF	*DEF /00000000, 63, 13, 15, 39, 40, 3
1.3	ØCT	*DEF /30000000, 38, 37, 34
77	ØUF	*DEF /00000000, 63, 16, 13, 14, 15, 39, 40, 41, 42, 7
78	PRG	*DEF /00000000, 63, 13, 14, 15, 1, 39
	SZE	*DEF /00000000, 63, 16, 39, 25, 18, 3

Assigned Monitor Audit Codes

25	*DEF /1, 18
26	*DEF /1, 16
27	*DEF /777, 15
28	*DEF /37, 7
29	*DEF /37, 2
30	*DEF /777, 12
31	*DEF /1, 0
32	*DEF /1, 1
33	*DEF /1, 2
34	*DEF /17, 3
35	*DEF /17, 7
36	*DEF /1, 11
37	*DEF /17, 12
38	*DEF /37, 16
39	*DEF /1, 19
40	*DEF /37, 14
41	*DEF /1, 13
42	*DEF /37, 5

* Revised 1/67

SUBJECT: QUESTIONS AND ANSWERS CONCERNING YPG53M - 4020 MONITOR MANUAL (PRELIMINARY)

April 28, 1967

The following questions, concerned with the Real-Time Multiprogramming Operating System found in YPG53M, have been submitted and are answered herin for the purpose of clarifying portions of that document.

1. Page 2, Section B

QUESTION: The "R" bit is specified as "zero" for both drum-core and all-core systems - explain.

ANSWER: The "R" bit explaination should read:

R = 1: Absolute Permanent Core Location
0: Relative Address
For an all-core system, "R" is always "zero".

2. Page 2, Section B

QUESTION: Where is the condition of the FMS bit stored where a program is interrupted?

ANSWER: If single-double or double only for a program, the program must have full register storage.

On p. 3, in the REGSTG Table, Bit 23 of the P-REG is 0 for single or 1 for double.

For drivers that go back to program immediately, Bit 23 of I-Block plus 2 is the indicator: Where I-Block is the same as First 8-Word Register Storage Block (ECP).

3. Page 9, Section 1.3

QUESTION: In what condition does the ØFFC01 subroutine leave the program's area?

ANSWER: Occupied, but available.

4. Page 10-11, Section 1.5

A. QUESTION: If the TPNC02 call is used and the program to be turned on is in delay, what happens?

ANSWER: Return is made to the program with a -1 in the A-Register.

B. QUESTION: Is the request ignored?

ANSWER: Yes.

C. QUESTION: If ignored, is there any indication given that the request was ignored?

ANSWER: Yes, in the A-Register upon return.

5. Page 11

QUESTION: If TPNCO3 call is used to turn on a program that is currently in core, is that program entered immediately, without regard to a higher priority request?

ANSWER: The next time ECP is entered, it goes immediately to that program, without regard to priority. (Only one TPNCO3 call is allowed at a time.)

6. Page 11

A. QUESTION: If a TPNCO3 call is used to turn on a program that is not currently in core, is the request for its transfer serviced immediately or is this request simply entered at the bottom of the transfer request list?

ANSWER: The request is serviced immediately; the transfer goes to the bottom of the list similar to all other bulk transfer requests.

B. QUESTION: When the transfer of the program is completed, is the program entered immediately, without regard to priority?

ANSWER: Yes.

7. Page 13, Section 1.7

QUESTION: Does the DTRCO2 set the calling program's area unavailable until the transfer is complete?

ANSWER: The area is not changed.

8. Page 14, Section 1.10

QUESTION: Is the index to the first 3-word group a "O"?

ANSWER: Yes.

9. Page 15, Section 1.11

A. QUESTION: Is there any limit to the size of a fixed data area?

ANSWER: There is no limit.

B. QUESTION: Must the address of a fixed area be pre-defined to the operating system?

ANSWER: No.

10. Page 16, Section 1.11

A. QUESTION: When using floating data tags, is it always necessary to transfer the data to the area associated with that tag before requesting output?

ANSWER: 1) In an all-core system, the floating data is a core address.
   a. Find the area
   b. Enter the data

2) In a bulk-core system:
   a. Build either before or after making check
   b. Find data area before transfer
   c. Bulk-transfer the data

B. QUESTION: Is it possible to simply check for a floating tag and request output from some other area - a data area in core, for instance?

ANSWER: Yes, but not recommended.

11. Page 19, Section 1.12

QUESTION: If a request is made to output information, formatted BCD, to a card punch, in the symbolic mode, will the output routine perform the ASCII to 80-column card conversion?

ANSWER: Yes. (If already 24-bit binary, use BCN.)

12. Page 28, Section C

QUESTION: Is there any procedure that can be followed to output numbers with negative scale factors?

ANSWER: Floating-point may be used.

13. Page 32, Section 1.14.4

QUESTION: What is the format for the binary cards produced by the output routine?

ANSWER: Bits 23-12 represent the odd columns;
Bits 11-0 represent the even columns.

14. Page 37, Section 1.15

A. QUESTION: Does the location ahead of the input list control word always contain the number of characters read, regardless of how they were packed?

ANSWER: Always the number of characters read on normal return or error returns with data error.

B. QUESTION: Does this refer to the numbers of characters before or after editing?

ANSWER: This is the number of characters after editing.

15. Page 41, Section 1.15.3

QUESTION: What action does the input program take when more than the specified number of characters have been entered? (Assume the input list is large enough to receive all the characters.)

ANSWER: Assume a request for Device Number 4 indicates that 30 characters are to be inputted through the keyboard into buffer KEYIN. If less than or 30 characters are accepted, this is reflected in the "number of words" found in the A-Register upon return to the requesting program's Normal Return. If more than 30 characters are entered, return is to the INRC02 call Error Return. If keyboard input, it is either "Time Abort" or "Overflow Error".

16. Page 41, Section 1.15

A. QUESTION: On a normal return from an INRC02 call, the A-Register contains a count of the "number of words". Is this the number of words occupied by the input message before or after editing?

ANSWER: After editing. (The number of words including partial words after editing.)

B. QUESTION: Are partially occupied words included in this count?

ANSWER: Yes. (Partially filled words are left-justified with zero's on the right end.)

17. Page 38, Section 1.15

QUESTION: Does the input routine edit information received from symbolic paper tape?

ANSWER: No, it just edits out ETX codes.

18. Page 51, Section 1.18

A. QUESTION: If, in a timed output request (MØRC03), a "0" in the A-Register leaves a contact "unchanged", what is the status word used for?

ANSWER: "Bits 23-8 - Data" should state:

1 = Contact to be latched
0 = Contact to remain unchanged or unlatched when no status

B. QUESTION: Is the status word affected by an MØRC03 request?

ANSWER: The status word is updated when the group requested has status.

19. Page 51, Section 1.18

A. QUESTION: When aborting a timed output (MØRC03) request with an MØRC05 request, what information should be placed in the A and Q-Registers?

ANSWER: The same contents of the A and Q-Registers in the MØRC03.

B. QUESTION: Can a single contact involved in a timed request be aborted?

ANSWER: No.

20. Page 51, Section 1.18

QUESTION: If a delayed return from the MØRC03 routine is chosen, will control be returned to the calling program immediately, after the relays have been operated, or after the time count has expired?

ANSWER: It returns after the time count has expired.

B. QUESTION: Does the MØRC03 affect the status of the calling program's area when the delayed return option is selected?

ANSWER: It doesn't touch the area.

21. Page 52, Section 1.18

QUESTION: In a pulsed output request (MØRC07), will a "O" in the A-Register cause a currently closed relay to be opened?

ANSWER: A zero means no action to be taken.

22. Page 53, Section 1.18

QUESTION: What is the duration of a pulse put out through the MØRC07 routine?

ANSWER: It is hardware defined - time base. (Greater than one cycle)

23. Page 53, Section 1.18

QUESTION: Can a pulsed output (MØRC07) request be aborted?

ANSWER: No.

24. Page 53, Section 1.18

QUESTION: Does the MØRC07 routine affect the area occupied by the calling program when the delayed return is chosen?

ANSWER: It doesn't change the area status.

25. Page 56, Section 1.20

QUESTION: Does the non-buffered scan routine affect the area occupied by the calling program?

ANSWER: It doesn't touch the area.

26. Page 59, Section 1.20

QUESTION: In the single input scan mode, when is the 10 code used in Bits 19 and 18 of the control word and when is the 00 code used?

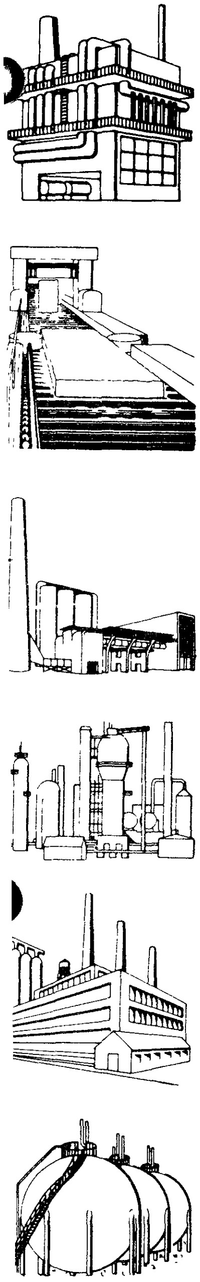
ANSWER: Only Bit 18 must be zero. (Hardware defined) Bit 19 is ignored.

Michael S Purtill

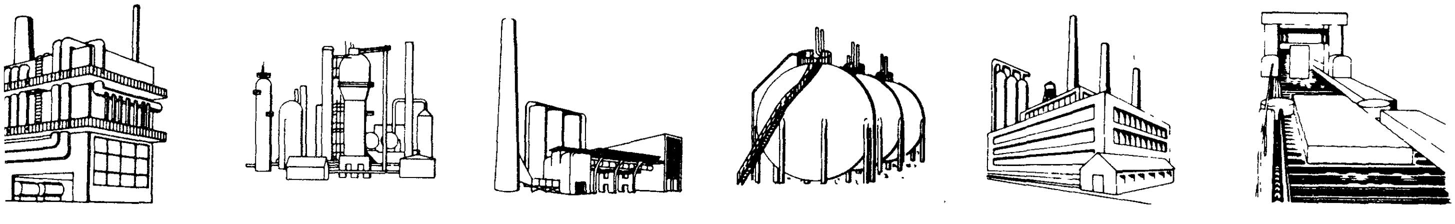
Michael S. Purtill
Technical Writer
Programming Publications

GE-PAC® 4000

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER



PROCESS ASSEMBLER LANGUAGE (ASCII Character Set)



ROCESS COMPUTER
USINESS SECTION
HOENIX, ARIZONA

GENERAL ELECTRIC

Library Control No. YPG52M

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P. O. BOX 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

CONTENTS

INTRODUCTION

This manual defines the standard Process Assembler Language (PAL) for the GE/PAC* 4000 Process Computer System using the American Standard Code for Information Interchange (ASCII). This discussion of PAL is directed toward the programmer familiar with GE/PAC programming concepts and techniques at the assembly language level. It does, however, include examples and references useful to the less experienced programmer.

First, the statement format is defined and PAL techniques are discussed. The pseudo-operations are then outlined and defined followed by assembly preparation and loading techniques.

* Registered Trademark of the General Electric Company

1 STATEMENT FORMAT

Assembly program input information is written on the "GE/PAC Language Statement Coding Form" (Figure 1). Each line on the coding form represents one instruction to the assembler. The coding form is comprised of four fields:

1. Location Field
2. Op Code Field
3. Operand Field
4. Identification Field

1.1 LOCATION FIELD - Columns 1 through 6

The location field is used to associate a name with the instruction or data written on the same line. Any reference to the instruction may be made by that name. Names used in the location field must consist of six or fewer alphanumeric characters; the first of which must be alphabetic and start in column one. A decimal point is considered as an alphabetic character in this context.

1.2 OP CODE FIELD - Columns 8 through 10

This field contains a two or three-character operation code which identifies the instruction to be executed. The legal operations consist of the pseudo operations described in Section 2 and the machine operations outlined in Appendix D.

1.3 OPERAND FIELD - Columns 12 through 69

The operand field may contain a combination of parameters. When evaluated, the parameters define the operand or operands required by the operation code. The following four basic parameter types are permitted in this field:

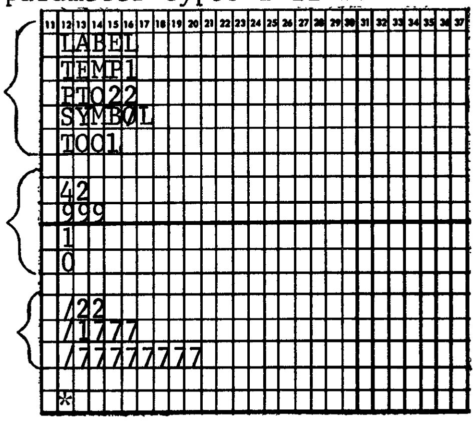
Label - A name composed of six or fewer alphanumeric characters; the first of which must be alphabetic. Any such symbolic parameter corresponds to a name written in the location field of some instruction.

Decimal - A decimal integer value

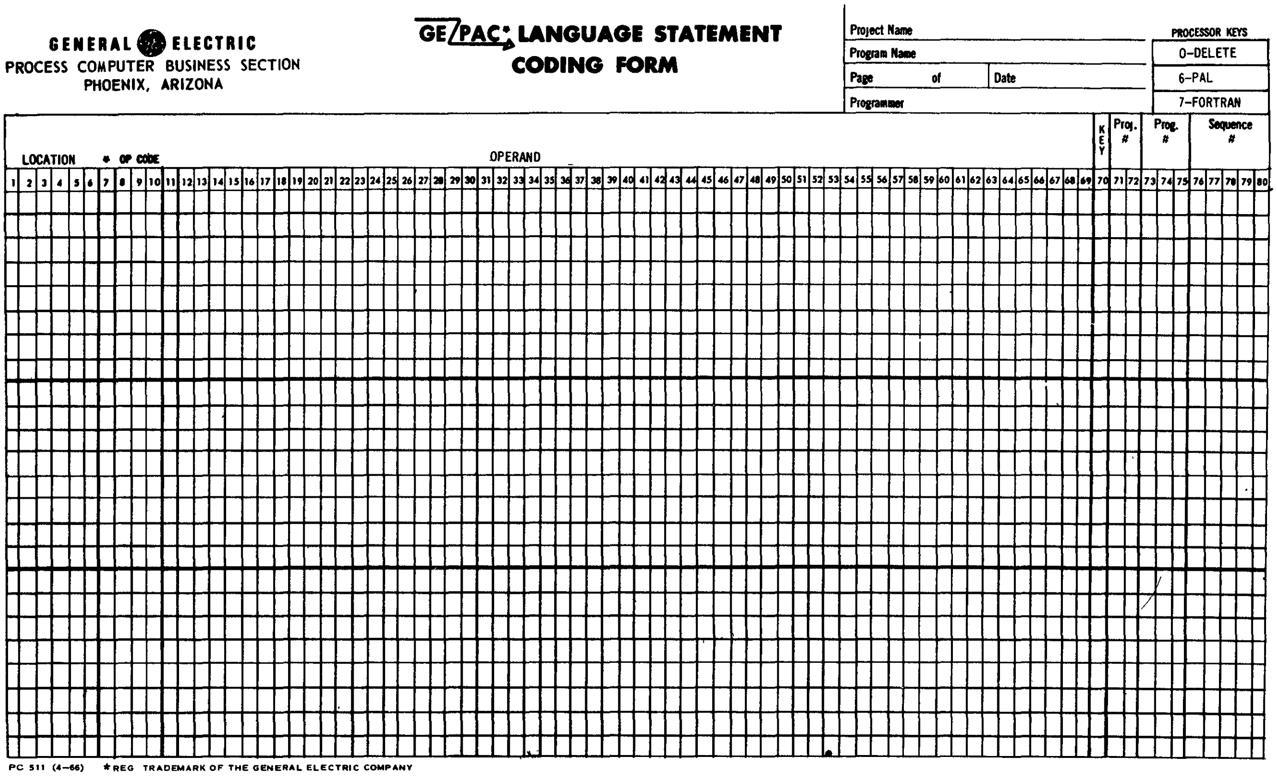
Octal - An octal integer value, preceded by a slash (/)

Present Location - An asterisk (*), which represents the address of this instruction's present memory location

Examples of the four parameter types follow:



```
                  11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
Label                LABEL
                     TEMP1
                     PTO22
                     SYMBØL
                     TOO1

Decimal              42
                     999
                     1
                     0

Octal                /22
                     /1777
                     /77777777

Present Location     *
```



GENERAL ELECTRIC
PROCESS COMPUTER BUSINESS SECTION
PHOENIX, ARIZONA

GE/PAC* LANGUAGE STATEMENT
CODING FORM

Project Name
Program Name
Page of Date
Programmer

PROCESSOR KEYS
0-DELETE
6-PAL
7-FORTRAN

LOCATION * OP CODE OPERAND KEY Proj. # Prog. # Sequence #

PC 511 (4-66) *REG TRADEMARK OF THE GENERAL ELECTRIC COMPANY

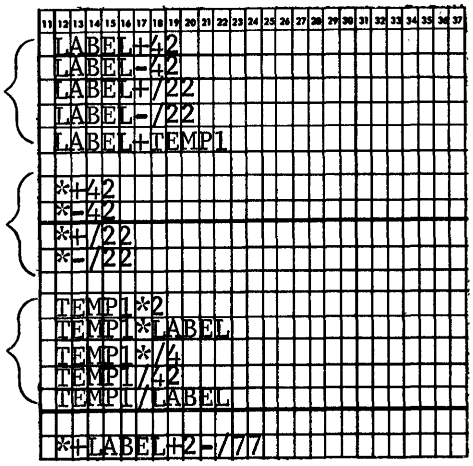
Figure 1 - GE/PAC CODING FORM

Single Operands

A single operand value may be composed of one or a combination of the four parameter types listed on page one. These parameters are combined using the following operators:

+ add	* multiply
- subtract	/ divide

Examples of parameter combinations which form a single operand are illustrated below:



Single operand as a combination of a symbolic and decimal or octal

```
LABEL+42
LABEL-42
LABEL+/22
LABEL-/22
LABEL+TEMP1
```

Combinations with *

```
*+42
*-42
*+/22
*-/22
```

Multiplication and division

```
TEMP1*2
TEMP1*LABEL
TEMP1*/4
TEMP1/42
TEMP1/LABEL
```

All four parameters combined

```
*+LABEL+2-/77
```

The parameters in the preceding examples are combined strictly from left to right. Ordinary precedence rules apply.

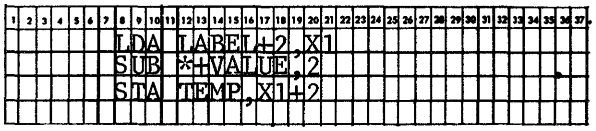
The meaning of asterisk or slash depends upon its relationship to the remaining parameters. For example, an asterisk represents multiplication only if it is connecting two parameters. In all other cases, it represents present location. The slash indicates an octal value when it precedes a numeric parameter; otherwise it is a division sign.

Most assembly language statements in any problem involve only simple operand combinations. Rarely will the multiplication or division capabilities be used. It is important to understand that the combinations of parameters in the operand field, simple or complex, involve the value (usually the location) of the symbol; not the contents of the location referenced.

Multiple Operands

Many computer instructions require more than one operand value. Most memory addressing instructions, for example, may have two operands; one to specify an index register and a second to specify the memory address. Multiple operands are desirable in many other occasions. When more than one operand is required, the comma (,) is used to separate them.

Examples of two-operand instructions are presented in the following:



```
1234567890123456789012345678901234567
       LDA LABEL+2,X1
       SUB *+VALUE,2
       STA TEMP,X1+2
```

The first blank space in the operand field terminates the instruction. Characters appearing after the first blank space are treated as comments.

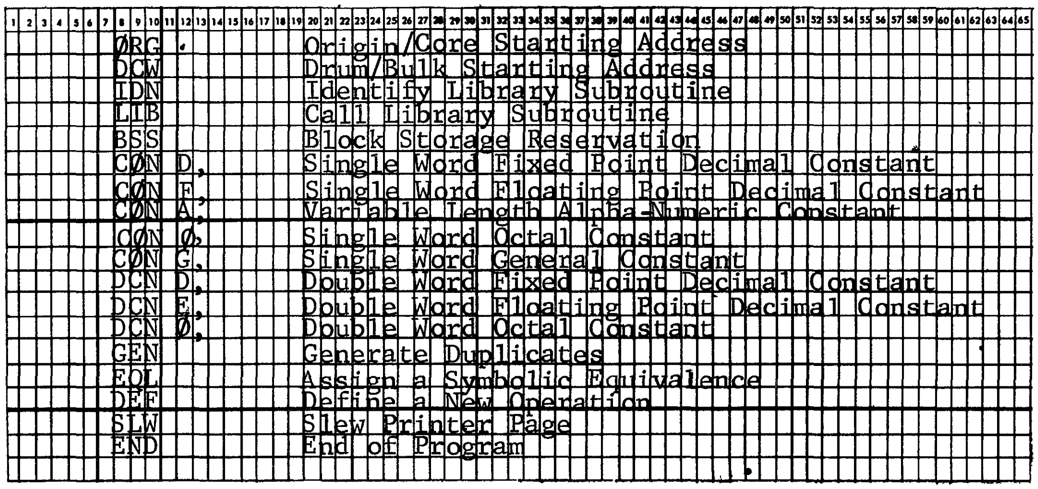
1.4 IDENTIFICATION FIELD - Columns 70 through 80

The last eleven columns are reserved for the identification of each line of coding, being used in some assembly systems, for proper maintenance of the symbolic information.

Information in this field also permits assemblers to differentiate between various languages the programmer has used. The use of this field may vary. The documentation for the various assemblers or other language processing systems describes its use.

2. PSEUDO-OPERATIONS

The pseudo-operations are written on the coding form in the op code field. They direct the assembler in storage assignments, symbol definitions, and generation of constants. All labels appearing in the operand field of pseudo-operations must be predefined (i.e., they must be common symbols or they must appear in the location field of a statement preceding this). A summary of the PAL pseudo-operations follows:



ØRG		Origin/Core Starting Address
DCW		Drum/Bulk Starting Address
IDN		Identify Library Subroutine
LIB		Call Library Subroutine
BSS		Block Storage Reservation
CØN	D,	Single Word Fixed Point Decimal Constant
CØN	F,	Single Word Floating Point Decimal Constant
CØN	A,	Variable Length Alpha-Numeric Constant
CØN	Ø,	Single Word Octal Constant
CØN	G,	Single Word General Constant
DCN	D,	Double Word Fixed Point Decimal Constant
DCN	E,	Double Word Floating Point Decimal Constant
DCN	Ø,	Double Word Octal Constant
GEN		Generate Duplicates
EQL		Assign a Symbolic Equivalence
DEF		Define a New Operation
SLW		Slew Printer Page
END		End of Program

In addition, the following conventions apply:

1. An asterisk in Column 1 identifies the entire line as a comment which will appear on the output assembly, but has no effect on the program.

2. An asterisk in Column 7 of a statement identifies the location field name as a common, absolute symbol.

3. A dash in Column 7 of the statement identifies the location field name as absolute (but not common).

4. Special pseudo-operations are available for use with Monitor (refer to GE/PAC MONITOR MANUAL).

2.1 ØRG - ORIGIN/CORE STARTING ADDRESS DCW - DRUM/BULK STARTING ADDRESS

ØRG and DCW are used to specify, to the loader, the following information:

1. The location in core (ØRG) or bulk (DCW) where the loader is to place the instructions that follow.

2. An indication if this segment should be permitted relocation at load time.

When a segment is designated relocatable the loader accepts, from the operator, bulk and core relocation values which tell the loader to move the program. Placing an asterisk in Column 7 of an ØRG or DCW instruction inhibits this action for the entire program up to another ØRG or DCW instruction.

The ØRG requires one operand, which specifies the core starting location. DCW requires two operands. The first specifies the bulk starting location; the second (optional) specifies a corresponding core location.

In the extreme left column of the listing produced by the assembler is the location of each instruction. This location is a core location initialized to the value of the ØRG operand or the second operand of a DCW.

If a program has no ØRG or DCW, it is assembled as though it were headed with an ØRG 0, permitting relocation.

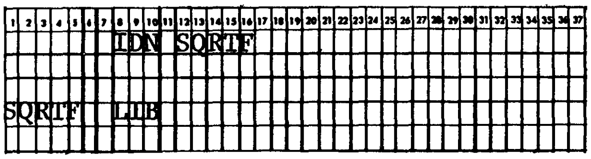
Any symbol used in an operand must appear in the location field of a previous instruction within this program, or must have been communicated as a common symbol from another program. The location field of an ØRG or DCW pseudo-op must remain blank.

2.2 IDN - IDENTIFY LIBRARY SUBROUTINE
LIB - CALL LIBRARY SUBROUTINE

The IDN and LIB pseudo-ops are used with FORTRAN library subroutines. They may be used to build and call other library subroutines as desired.

The appearance of an LIB at the end of a program signals the loader to search for a program by that name (identified by IDN) to load and establish communication between the calling program and the library routine.

Examples:



```
       IDN SQRTF

SQRTF  LIB
```

An IDN appears only once in a library routine, and must be the first statement in the routine. An ØRG or DCW may not appear in the routine. The name of the routine (a Label as defined in paragraph 1.3) appears in Columns 12 and following.

The LIB statements must appear at the end of the calling program. The number of them is limited only by the number that the loader can accommodate, which varies with the loader being used. The library subroutine name is a label appearing in the location field.

2.3 BSS - BLOCK STORAGE RESERVATION

BSS is used to reserve or skip a block of memory. The size of the block is designated in the operand field. Any symbol that is used must be a common symbol communicated from another program or must have appeared in the location field of an earlier instruction within this program.

A symbol written in the location field is entered into the assembler's table of symbolic equivalences. It is entered with the location value corresponding to the first location of the reserved block.

2.4 CØN - CONSTANT

CØN generates program constants. The five types are specified by an alphabetic character in Column 12 as illustrated on page five.

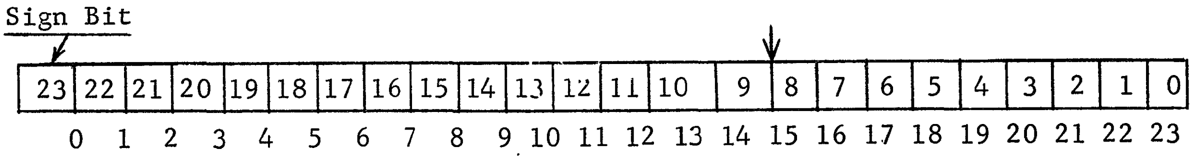
The assembler generates the binary equivalent for a CØN constant. A symbol appearing in the location field of any constant is entered in the assembler table of symbolic equivalences. All references to the constant may be made by that name.

2.4.1 Single-Word Fixed-Point Decimal Constant

The first operand, D, identifies the constant as fixed-point decimal. The second operand presents the value of the constant. A binary scale factor and a power of ten exponent may be expressed in the constant. The binary scale factor indicates the bit position of the binary point relative to the sign bit in the word.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
       CØN D,42
       CØN D,22.4B14
       CØN D,-.224B14E2
```

In the preceding examples 2 and 3, the binary point is assumed as indicated by the arrow illustrated below:



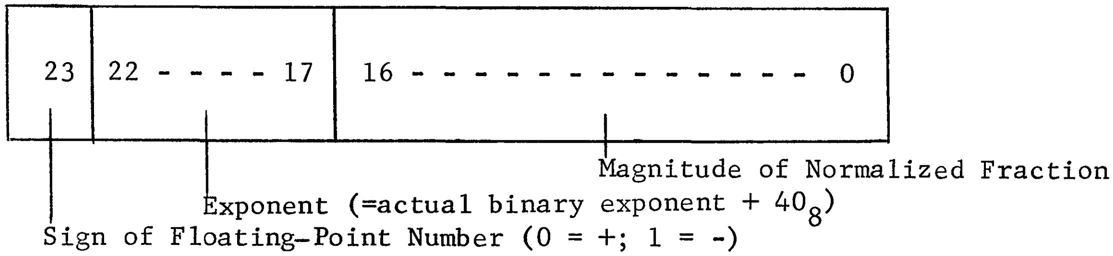
Bits are numbered from right to left; binary points (equivalent to a decimal point) are numbered from left to right. The binary point is considered as being located to the right of any bit. Consequently, if a point is to the right of Bit 23, the data word is considered to be a fraction scaled B0. The data word is considered to be a full integer scaled B23 when the point is to the right of bit zero.

2.4.2 Single-Word Floating-Point Decimal Constant

The first operand, F, in the following example, identifies the constant as a floating-point decimal. The second operand presents the value of the constant. The number following the E indicates the power of ten exponent.

```
CØN F,22.5
CØN F,225E-1
CØN F,.225E2
```

A single-word floating-point constant is formed with the exponent in the six bits following the sign bit and the normalized fraction in the rightmost 17 bits of a word.


23
22 - - - - 17
16 - - - - - - - - - - - - - - 0
Magnitude of Normalized Fraction
Exponent (=actual binary exponent + 40_8)
Sign of Floating-Point Number (0 = +; 1 = -)

2.4.3 Variable Length Alphanumeric Constant

The first operand, A, identifies the constant as an alphanumeric. The second operand presents the number of characters in the constant (between 1 and 52). The third presents the characters of the constant. All characters are translated to the seven-bit ASCII code, an eighth bit (always zero) is appended on the left of each, then they are packed three per word from left to right. If the number of characters requested is insufficient to fill the last word, it is completed with null codes (177_8).

```
CØN A,2,AB           Generates one word; third character null
CØN A,3,AB           Generates one word; third character blank
CØN A,9,ABCDEFGHI    Generates three words
```

2.4. Single-Word Octal Constant

The first operand, Ø, identifies the constant as an octal constant. The second operand presents the value of the constant.

```
CØN Ø,777
CØN Ø,311
CØN Ø,77777777
```

The value of the second operand is converted to binary, right justified, and entered in the program as a single word constant. A plus or minus sign is not usable with CØN Ø.

2.4.5 Single-Word General Constant

The first operand, G, identifies it as a general constant. The second operand specifies the value of the constant. The second operand may be any combination of parameters defined as legal for the operand field. A decimal, a symbolic, and an octal constant are illustrated below:

```
CØN G,25
CØN G,LABEL-6
CØN G,/77777777
```

Decimal constants of this type must be integers. B-scale factors and E multipliers as used in the CØN D pseudo-op are not allowed.

2.5 DCN - DOUBLE-WORD CONSTANT

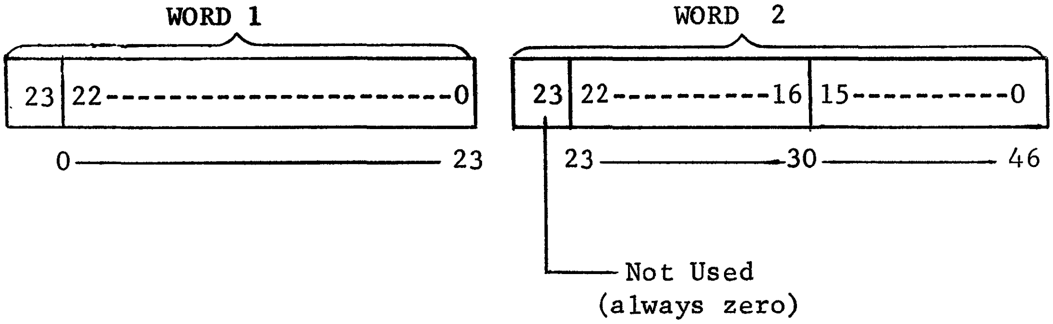
DCN is used to generate double-word program constants. The type of constant and its value are specified in the operand field.

2.5.1 Double-Word Fixed-Point Decimal Constant

The first operand, D, identifies the constant as a fixed-point decimal. The second operand presents the value of the constant. A binary scale factor and a power of ten exponent may be expressed in the constant. The binary point is relative to the sign bit in the first word. The value is placed in two words. The sign of the second word is not used and is set to zero. It is analogous to the sign of the Q-Register which is not used in GE/PAC arithmetic instructions.

```
DCN D,42
DCN D,22.4B30
DCN D,.224B30E2
```

In the second and third examples, the binary point is assumed between Bits 15 and 16 of word two as indicated by the arrow:



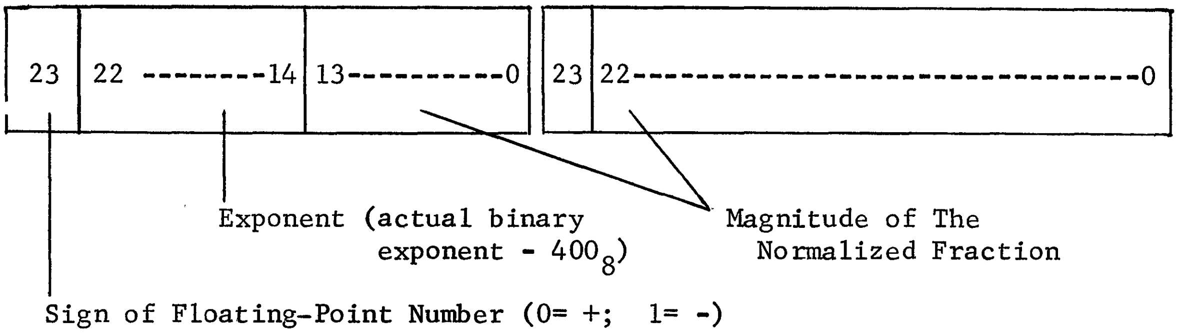
Since Bit 23 of word two is not used, Bit 22 of word two and Bit zero of word one are contiguous. Therefore, binary point 23 is between Bit zero of word one and Bit 22 of word two.

2.5.2 Double-Word Floating-Point Decimal Constant

The first operand, F, identifies the constant as a floating decimal. The second operand presents the value of the constant.

```
DCN F,-232.5
DCN F,2.3252E2
DCN F,2325.E-1
```

The format of the double-word floating-point constant is illustrated below:


23
22 --------14
13----------0
23
22--------------------------------0
Exponent (actual binary exponent - 400_8)
Magnitude of The Normalized Fraction
Sign of Floating-Point Number (0= +; 1= -)

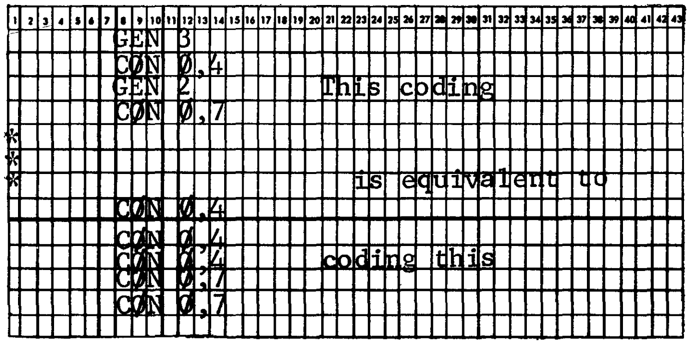
2.5.3 Double-Word Octal Constant

The first operand, Ø, identifies the constant as an octal constant. The value of the constant is written as the second operand. It is converted to binary, right justified, and entered as a double word logical constant (48 bits). Unlike arithmetic instructions, which ignore the sign bit of the Q-Register, GE/PAC logical instructions use the sign bit of Q. Therefore, Bit 23 of the second word of a double word octal constant is utilized.

```
DCN Ø,77
DCN Ø,311
DCN Ø,7777777777777777
```

2.6 GEN - GENERATE DUPLICATES

A GEN instruction operand specifies the total number of times the following instruction will appear. The operand may be a combination of parameters defined as legal for the operand field. Symbols used must be predefined.



```
       GEN 3
       CØN Ø,4
       GEN 2       This coding
       CØN Ø,7
*
*
*                     is equivalent to
       CØN Ø,4
       CØN Ø,4
       CØN Ø,4     coding this
       CØN Ø,7
       CØN Ø,7
```

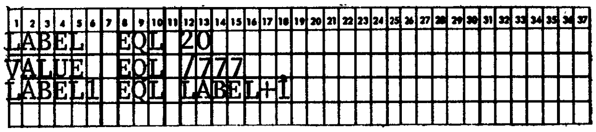
The GEN instruction cannot be used to duplicate the following pseudo-ops:

ØRG
BSS
DCW
CØN A
GEN
EQL
DEF
SLW
END
LIB
IDN

2.7 EQL - ASSIGN A SYMBOLIC EQUIVALENCE

EQL is used to assign a value to a symbol during assembly. This function is useful in sharing storage areas, establishing communication between programs, and assigning parameter values in a form adaptable for changes.

Enter the symbol into the location field and its equivalent value into the operand field. The operand value may be any combination of parameters described in Section 1. There is one restriction; symbols written in the operand field must be previously defined.



```
LABEL  EQL 20
VALUE  EQL 7777
LABEL1 EQL LABEL+1
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

2.8 DEF - DEFINE A NEW OPERATION

DEF permits the programmer to define new operation codes (i.e., codes that are not defined in the PAL language).

DEF is used for the following reasons:

1. An op code may not exist for all GE/PAC computers; therefore it is not included in the basic PAL language. The programmer may define the op code when frequently used in the system.[1]

2. Tables of parameters may consist of more than one value packed within a word. In this case, it is more convenient to write the values separately and allow the assembler to combine them into single words. Otherwise, the values would have to be hand-assembled to one-word constants and entered by using the CØN pseudo-op. DEF permits the programmer to define new operation codes that will accept multiple values as separate operands.

A detailed procedure for using the DEF pseudo-op follows.

[1] For infrequently used operations, it may be easier to insert the undefined instruction as an octal constant.

2.8.1 Op Code Definition

Machine operations (op codes) are written with optional operands separated by commas. The assembled instruction results from combining the values of the operands with the base octal associated with the op code. The base octal is 24 bits long.

Each operand has two format characteristics:

1. Operand width (specified as the largest octal value that may be generated)

2. Position (where the value is placed in the finished word)

Example: The machine op code, ADD, is comprised of base octal 11000000 and two operands. The first operand value is placed into Bits 13-0 (width 37777_8, position 0). Second operand value (Index Register) is placed into Bits 17-15 (width 7, position 15).

To define a new op code all the following must be specified:

1. The base octal
2. The number of operands it will accept[2]
3. Format characteristics for each operand

Defining a new op code requires defining audit codes. An audit code specifies an operand width and position to the assembler. Each audit code has a number associated with it. Sixty-four audit codes are available for the assembler. Audit code numbers 0-24 are reserved; 0-19 are predefined (refer to Appendix C). Codes 25-50 should be avoided by the programmer; they are reserved for PAL and Monitor for future op code additions. Audit code 63 has special use (refer to paragraph 2.8.2). Therefore, only audit code numbers 51-62 may be assigned freely by the programmer. These may be reused within the same program, one definition applying only until the next definition is encountered. It is recommended that the programmer start with 62, numbering in reverse.

Three steps in defining a new operation follow:

1. Determine width and position of all operands.

2. Define audit codes for those widths and positions.

3. Define the op code by assigning to it the audit codes from Step 2 above.

[2] Up to four operands are normally allowed. When more than four are required, information outlined in paragraph 2.8.2 applies.

The steps in detail:

1. For each operand of a new op code, determine its width and the position of its rightmost bit (anchor bit).

Example: Assume a new op code, called AAA, were needed with the following characteristics:

a. A base octal of 12034560

b. Two operands, whose values are placed in Bits 17-15 and Bits 3-0. These operands are described as follows: Operand one has a width of three bits and an achor bit of 15. This width is expressed as the largest number it can contain in octal (in this case, seven is the largest number three bits can contain). The second operand has a width of four bits (/17) and an anchor bit of zero.

2. Define audit codes for each operand as follows:

a. Write the audit code in Columns 1 and 2 of the location field.

b. Write DEF in the op code field

c. In the operand field, write two operands separated by a comma. The first is the width, expressed in octal; the second is the anchor bit position.

Example:

The coding for AAA above follows:

```
        Location      Op Code     Operand

           62           DEF        /7,15
           61           DEF        /17,0



Audit Code                       Width        Anchor Bit
```

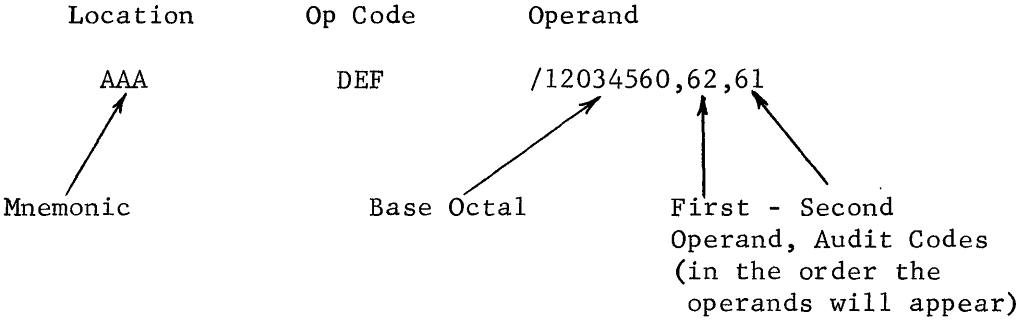
3. When all necessary audit codes are defined, the op code itself may be defined as indicated below:

a. Write the new operation mnemonic in Columns 1 through 3 of the location field.

b. Write DEF in the op code field.

c. Write the base octal and audit codes, separated by commas, in the operand field.

Example:



Location	Op Code	Operand
AAA	DEF	/12034560,62,61

Mnemonic
Base Octal
First - Second Operand, Audit Codes (in the order the operands will appear)

The new op code may now be used (note that a symbolic operand may be used):

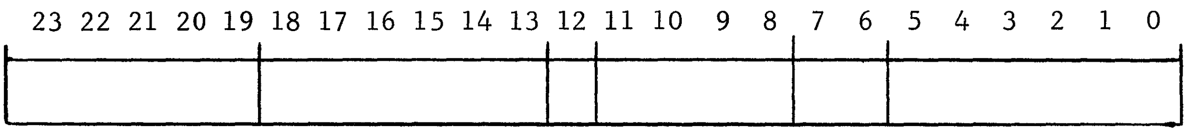
```
A       EQL 3
        AAA 4,2
        AAA A,0          Result     12 [4]  3  4  5  6 [2]
                                    12 [3]  3  4  5  6 [0]
                                        ↑               ↑
                                  1st Operand      2nd Operand
```

Different op codes may share audit codes if they have operands which coincide exactly in width and position (i.e., the programmer need define only one audit code for a particular operand width and position, then that audit code may be used freely for as many op codes as desired).

2.8.2 Extra Operands Option

When generating special tables of data, four operands may be inadequate.

Example: A table of constants is desired where each word is divided as follows; each section is able to take on an independent value.



The normal technique for op codes permits only four operands, which is insufficient (the example requires seven operands). The use of DEF in the Extra Operands Option permits up to 12 operands. The steps for defining an op code when using the Extra Operands Option are nearly identical to those for a normal op code.

1. For each operand, determine its width and the position of its rightmost bit.

Example:

Width (No. of Bits)	Largest Number It Can Contain	Anchor Bit
1	1	23
4	17_8	19
6	77_8	13
1	1	12
4	17_8	8
2	3	6
6	77_8	0

2. Define audit codes for those positions and bit lengths.

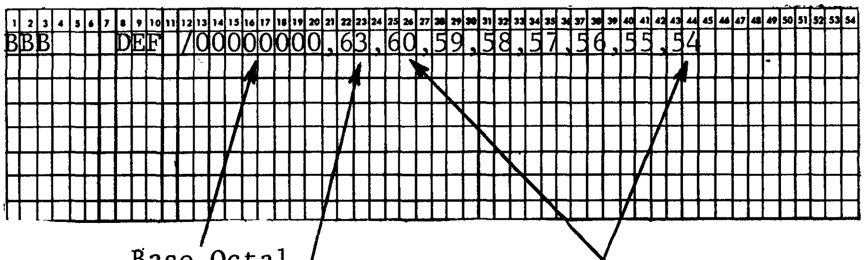
   Example:

```
60     DEF /1,23
59     DEF /17,19
58     DEF /77,13
57     DEF /1,12
56     DEF /17,8
55     DEF /3,6
54     DEF /77,0
```

Note: If an audit code, with identical width and anchor bit for an earlier op code is defined, it need not be defined again, but may be used for this op code.

3. When all necessary audit codes are defined, the op code itself is defined. To signal the Extra Operands Option, place audit code 63 as the first audit code.

Example: (BBB is used as the mnemonic for this op code).



```
BBB    DEF /00000000,63,60,59,58,57,56,55,54
```

Base Octal

Extra Operands Option

Audit Codes (in the same order the operands will appear)

4. Using the new mneomonic:

```
H      EQL /777
       BBB 1,/17,/50,0,5,3,H
       BBB 0,8,1,0,2,1,4
       BBB 0,0,1,0,0,0,0
```

The foregoing is assembled as follows:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	77202777
0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	20021104
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	00020000

2.8.3 General Rules

1. DEF is available only for additions to the language, not for substituting new definitions for permanently defined PAL op codes.

2. Operands of the DEF instruction itself may be written in decimal, octal, or symbolic. However, all symbols must be previously defined. The usual rules for determining if operands are relative or absolute do not apply. All operands are considered absolute.

3. Operands for newly defined op codes follow the usual rules for translation as absolute or relative symbolics (refer to page 18).

4. Audit codes must be numeric.

5. Audit codes 51-62 only may be used. It is recommended that the programmer start with 62 and number in reverse.

6. Audit codes and op codes not reserved for PAL may be redefined within a program as often as the programmer desires.

7. New op codes, audit codes, and their audit code definitions are a part of the common symbol table which may be preserved for subsequent assemblies.

8. The Extra Operands Option may be used for defining all op codes. However, normal use of DEF for op codes requiring four or fewer operands will save space that the assembler can use to store more symbols.

2.8.4 Error Conditions

When an attempt is made to redefine an op code or audit code that has been permanently defined, the permanent definition still applies and the new definition will be flagged as a location error (L). Permanently defined op codes are those that make up the PAL language codes.

When defining an op code, if audit code 63 (Extra Operands Option) is written anywhere except directly after the base octal, an illegal operand error (I) will occur. Definition of the op code is terminated at this point.

When defining an audit code, the nubers 0 through 24 will be flagged as a location error (L). They are reserved for the assembler.

2.9 SLW - SLEW PRINTER PAGE

Upon encountering the SLW pseudo-op, the assembler positions the paper at the top of the next page. Assemblers operating on computers without output page control ignore the SLW instructions.

2.10 END - END OF PROGRAM

END is used to terminate the assembly and must be the last statement in a program. Upon encountering END, the assembler generates control information for the loader.

An operand may be used to indicate the starting point of the program. When this is done, the starting address is also communicated to the loader.

2.11 MONITOR PSEUDO-OPS

Pseudo-ops exist to generate constants for Monitor. They are used to generate parameters for calling sequences and format words for input/output. These pseudo-ops are described in the GE/PAC Monitor Manual and are available only through the Monitor Common Symbol Tape.

3.0 PROCESSING AND OUTPUT

GE/PAC uses a two-pass assembler. During the first pass, the assembler examines the storage allocation and forms a table of symbolic equivalences. Each symbol appearing in the location field is entered into the symbol table. During the second pass, the assembler completely forms each instruction code. For each symbolic operation, the assembler searches a table of permanent operation codes to acquire its corresponding base octal and audit codes for operand treatment. The operand field is scanned, and equivalent values are computed using the symbol table formed during the first pass. The base octal and the operands are then combined using the audit codes. The final step in the assembly of each statement is the preparation of a line of output for listing and direct loading.

3.1 CHARACTER CODES

Appendix A presents the character set associated with PAL. Additional codes applicable to the Model 4233 Teletypewriter, Model 4262 Line Printer, Model 4223 Output Typer (long carriage), Model 4270 I/O Typer, etc. are presented on a separate sheet available from the Programming Library. They are legal as comments in the operand field depending upon the device used for listing at assembly time. Any character may be used in the operand of a CØN A instruction if the character code is legal for the devices that will read and write that character, at assembly and at run time.

3.2 ABSOLUTE AND RELATIVE VALUES

An operand of an instruction which references memory may be specified to be assembled as an absolute or relative address. An absolute address is the true location of the referenced memory cell. A relative address is the difference between the address of the referenced memory cell and the address of the instruction. The result is negative when the referenced address is smaller than the instruction's address. Assembling addresses in their relative form permits a program to operate anywhere in the GE/PAC.

An operand as written by the programmer can consist of one or a combination of the following terms:

An absolute label	Absolute terms
An integer	
A relative label	Relative terms
The *(Present location)	

Combinations are formed using add (+), subtract (-), multiply (*), and divide (/) operators (refer to paragraph 1.3)

An operand is assembled relative if any relative term (a relative label or the asterisk) appears in the combination of terms. An operand is assembled absolute when no relative term appears.

A label usually represents an address in memory. Addresses in the GE/PAC may be referenced relative or absolute. To reference an address absolute, specify as absolute the label associated with that address. Otherwise, it is assumed to be relative by the assembler. A label specified as common to more than one program (refer to paragraph 3.2) is assembled as an absolute label in all programs in which it is referenced. Two additional methods to specify a label as absolute are as follows:

1. Place a dash in Column 7 of the line of coding where the label appears in the location field.

   Example: ALPHA -LDA BETA — ALPHA will be referenced absolute.

2. Use the EQL pseudo-op to equate the label to an absolute operand.

   Example: GAMMA EQL ALPHA+2 — Both ALPHA and 2 are absolute. Therefore, the operand is absolute and GAMMA will be considered absolute.

3.3 COMMON SYMBOLS

A label is specified to be common by placing an asterisk (*) in Column 7 of the line of coding where the label appears in the location field. Each common label and its value is available through a common symbol table to all other programs in a system.

3.4 ASSEMBLER VARIATIONS

Variations in the capabilities of language translators originate from the different operating environment of each. For example, the memory available affects the size of the symbol table. It is important that the characteristics of each translator be examined prior to its use.

3.5 OUTPUT

Each translator of the assembly language possesses its own output characteristics. However, all translators can produce the two basic output requirements:

1. An assembly listing, which includes the original symbolic information plus the assembled instructions in the relative and absolute forms. The output listing will also contain error flags. These are defined in Appendix B.

2. Output for loading in the GE/PAC system.

APPENDIX A

ASCII CHARACTER CODE TRANSLATION

The list of characters in Table I comprises the total ASCII Character Set associated with the assembler language. The octal associated with each character represents the equivalent code produced by the paper tape preparation device for input.

The additional characters listed in Table II may be legal, as comments in the operand field depending upon the device used for listing at assembly time, or in the operand of a CØN A instruction if all devices that read or write that character can recognize it, at assembly and at run time.

TABLE I

CHARACTER	TYPER
A	= 101
B	= 102
C	= 103
D	= 104
E	= 105
F	= 106
G	= 107
H	= 110
I	= 111
J	= 112
K	= 113
L	= 114
M	= 115
N	= 116
O	= 117
P	= 120
Q	= 121
R	= 122
S	= 123
T	= 124
U	= 125
V	= 126
W	= 127
X	= 130
Y	= 131
Z	= 132
0	= 060
1	= 061
2	= 062
3	= 063
4	= 064
5	= 065
6	= 066
7	= 067
8	= 070
9	= 071
+	= 053
-	= 055
/	= 057
*	= 052
.	= 056
,	= 054
space	= 040

TABLE II

CHARACTER	TYPER
(	= 050
)	= 051
=	= 075
"	= 042
$	= 044
'	= 047
#	= 043
@	= 100
&	= 046
%	= 045
?	= 077
!	= 041
>	= 076
<	= 074
[	= 133
]	= 135
:	= 072
;	= 073
\	= 134
∧	= 136
_	= 137

APPENDIX B

ERROR FLAGS

The assembler performs validity tests on each instruction. When errors or suspected errors are detected, one of the following indicators will appear on the output listing.

FLAG	DEFINITION	CAUSE
L	Location Field Error	1. First character of the label is not alphabetic (see Appendix A, Table I). 2. Using the DEF pseudo-op when: a. The mneomonic assigned is a GE/PAC machine operation. b. Requesting "extra operands" definition when mnemonic has been previously defined as machine-typed operation. c. There is an illegal audit code number. 3. Location field is blank when a symbol is required. 4. Location field contains a symbol when not allowed. 5. Label not found in the table, probably due to overflow of the table (some assemblers) on the first pass.
Ø	Operation Field Error	1. The op code not part of the language or was not added to the table through the DEF pseudo-op. This often occurs when definition was attempted but was illegal. Consequently, it was not added to the operation table. 2. This op code cannot be GENerated.
I	Illegal Operand	1. Blank operand when an operand is required. 2. Operand not blank when it should have been. 3. One or more required operands missing. 4. Too many operands. 5. Operand value too large. 6. Negative operand value in an instruction that will not accept one. 7. Illegal constant.
X	Index Word Error	1. Index word 1 or 2 specified. 2. Required index missing. 3. Specified index word is greater than seven.
U	Undefined Symbol	Occurs only when a symbol appears in the operand field and: 1. It never appeared in the location field or on the common symbol tape. 2. It appeared in the location field, but the symbol table was full at that time.
C	Illegal Character	A character, not associated with the assembler language, was found in one of the following fields: 1. Location 2. Op Code 3. Operand (Refer to Appendix A, Table II)

APPENDIX B

ERROR FLAGS

FLAG	DEFINITION	CAUSE
M	Multiply-defined Symbol	1. Symbol in location field was flagged because: a. It has appeared in the same field on a previous record. b. It appeared on the requested EQL tape with a value unequal to the one being assigned. c. It was saved from a previous assembly with a value unequal to the one being assigned. 2. Any record which references a multiply-defined symbol in the operand field will also be flagged.
2	Second Pass Definition of Symbol Different from First Pass	
R	Relative Operand Error	Operand value was relative and should normally be absolute for this operation.
F	Tables Full	

AUDIT CODES

Each machine instruction may have operand values. The assembler has a list of operand-associated numbers called audit codes. These numbers uniquely identify individual operand requirements. Two basic characteristics of audit codes are:

1. Operand width expressed as a mask

2. Anchor position

A maximum of sixty-four audit codes are available. Audit codes 0 through 50 are reserved for the assembler and Monitor; 51 through 62 are available for programmer definition. The first twenty audit codes are defined and listed in the table at the right.

AUDIT NUMBER	OPERAND WIDTH	ANCHOR POSITION	REMARKS
0	0	0	No operand accepted
1	14	0	Full operand. May be absolute or relative.
2	3	15	For instructions with optional tag field. Must be absolute if not blank.
3	14	0	Full operand. Must be absolute.
4	12	0.	Must be absolute.
5	24	0	Pass without audit (any value accepted).
6	3	15	For instructions requiring a tag. Must be absolute.
7	5	0	K bits. Must be absolute.
8	14	0	Value will be negated (TXH). Must be absolute.
9	23	0	Operand may be absolute or relative (DEL).
10	3	15	For STX instruction.
11	14	0	Operand may be relative or absolute and may be negative (INX).
12	5	15	Absolute only
13	1	22	Absolute only
14	1	21	Absolute only
15	1	20	Absolute only
16	1	23	Absolute only
17	23	0	Absolute only
18	3	15	For STR-type instruction.
19	12	0	Absolute only; may be negative.

APPENDIX D

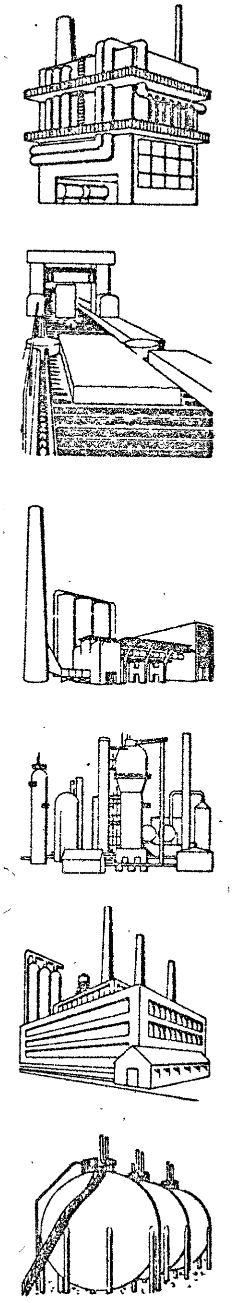
MACHINE OPERATIONS

ABL	APPEND ITEM TO BEGINNING OF LIST
ABT	ABORT DEVICE D's OPERATION
ACT	ACTIVATE DEVICE D's INTERRUPT
ADD	ADD
ADØ	ADD ONE TO BIT K
AEL	APPEND ITEM TO END OF LIST
AFA	ADD FIELD TO A
AKA	ADD K TO A
ANA	AND TO A
BRU	BRANCH UNCONDITIONALLY
BTR	BRANCH IF TSTF RESET
BTS	BRANCH IF TSTF SET
CBK	CHANGE BIT K
CLØ	COUNT LEAST SIGNIFICANT ONES
CLZ	COUNT LEAST SIGNIFICANT ZEROS
CMØ	COUNT MOST SIGNIFICANT ONES
CMZ	COUNT MOST SIGNIFICANT ZEROS
CPL	COMPLEMENT A
DAD	DOUBLE ADD
DLA	(SHIFT) DOUBLE LEFT ARITHMETIC
DLD	DOUBLE LENGTH LOAD
DLL	(SHIFT) DOUBLE LEFT LOGICAL
DMT	DECREMENT MEMORY AND TEST
DRA	(SHIFT) DOUBLE RIGHT ARITHMETIC
DRC	(SHIFT) DOUBLE RIGHT CIRCULAR
DRL	(SHIFT) DOUBLE RIGHT LOGICAL
DST	DOUBLE LENGTH STORE
DSU	DOUBLE SUBTRACT
DVD	DIVIDE
ERA	EXCLUSIVE OR TO A
FAD	FLOATING ADD
FDV	FLOATING DIVIDE
FIX	FIX FLOATING NUMBER
FLØ	FLOAT FIXED NUMBER
FMP	FLOATING MULTIPLY
FMS	FLOATING MODE SHIFT
FSU	FLOATING SUBTRACT
IAI	INHIBIT AUTOMATIC INTERRUPT
IBK	ISOLATE BIT K
IDL	INPUT FROM DEVICE TO LIST
IN	INPUT FROM DEVICE D
INX	INCREMENT X
JCB	JUMP IF CHANNEL IS BUSY
JDR	JUMP IF DATA READY
JND	JUMP IF NO DEMAND
JNE	JUMP IF DEVICE D NOT IN ERROR
JNØ	JUMP IF NO OVERFLOW
JNP	JUMP IF NO PARITY ERROR
JNR	JUMP IF DEVICE D NOT READY
LBM	LOAD BIT MASK
LDA	LOAD THE A-REGISTER
LDB	HIGHSPEED I/O BUFFER
LDF	LOAD FIELD
LDI	LOAD INDIRECT
LDK	LOAD A WITH K
LDØ	LOAD ONE INTO BIT K
LDP	LOAD PLACE
LDQ	LOAD THE Q-REGISTER
LDR	LOAD REGISTERS
LDX	LOAD X WORD
LDZ	LOAD ZEROS INTO A
LMØ	LOAD MINUS ONE

APPENDIX D

MACHINE OPERATIONS

LPR	LOAD PLACE AND RESTORE
LXC	LOAD X WITH COUNT
LXK	LOAD X WITH K
MAQ	MOVE A TO Q
MPY	MULTIPLY
NEG	NEGATE
NØP	NO OPERATION
ØDL	OUTPUT FROM DEVICE TO LIST
ØØM	OPERATE ON MEMORY
ØPR	OPERATE DEVICE D
ØRA	OR TO A
ØUT	OUTPUT TO DEVICE D
PAI	PERMIT AUTOMATIC INTERRUPT
RBK	RESET BIT K
RBL	REMOVE BEGINNING ITEM FROM LIST
RCS	READ CONSOLE SWITCHES
REL	REMOVE ENDING ITEM FROM LIST
REV	RESET TSTF IF BIT K IS EVEN
RNZ	RESET TSTF IF A IS NONZERO
RØD	RESET TSTF IF BIT K IS ODD
RPT	REPEAT INSTRUCTION IN LOCATION 7
RST	RESET TSTF
SBK	SET BIT K
SEL	SELECT DEVICE D
SET	SET TSTF
SEV	SET TSTF IF BIT K IS EVEN
SFA	SUBTRACT FIELD FROM A
SKA	SUBTRACT K FROM A
SLA	SHIFT LEFT ARITHMETIC
SLL	SHIFT LEFT LOGICAL
SNZ	SET TSTF IF A IS NONZERO
SØD	SET TSTF IF BIT K IS ODD
SPB	SAVE PLACE AND BRANCH
SRA	SHIFT RIGHT ARITHMETIC
SRC	SHIFT RIGHT CIRCULAR
SRL	SHIFT RIGHT LOGICAL
SSA	SET STALL ALARM
STA	STORE CONTENTS OF A
STB	STORE HIGHSPEED I/O BUFFER
STF	STORE FIELD
STI	STORE INDIRECT
STQ	STORE.CONTENTS OF Q
STR	STORE REGISTERS
STX	STORE X
SUB	SUBTRACT
TER	TEST EVEN AND RESET BIT K
TES	TEST EVEN AND SET BIT K
TEV	TEST BIT K EVEN
TFE	TEST FIELD EQUAL
TFL	TEST FIELD LESS
TNM	TEST NOT MINUS ONE
TNZ	TEST A NONZERO
TØD	TEST BIT K ODD
TØR	TEST ODD AND RESET BIT K
TØS	TEST ODD AND SET BIT K
TSC	TEST AND SHIFT CIRCULAR
TXH	TEST X HIGH OR EQUAL
TZC	TEST ZERO AND COMPLEMENT
TZE	TEST A ZERO
XEC	EXECUTE

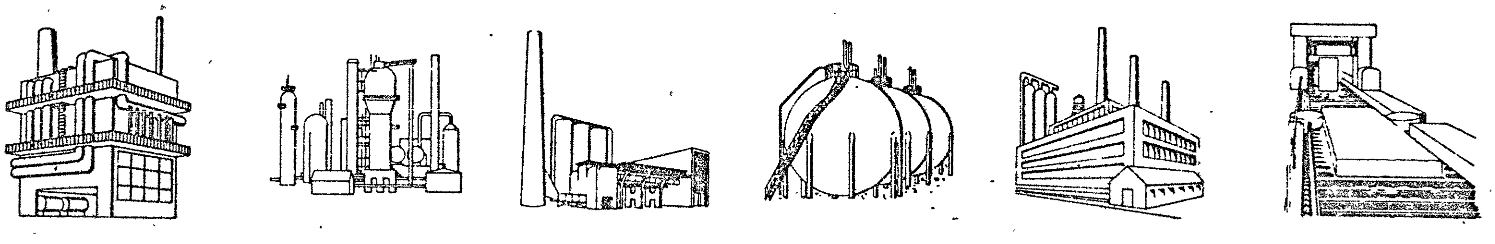


GE PAC 4000

GENERAL ELECTRIC PROCESS AUTOMATION COMPUTER

ON-LINE PROCESS OPTIMIZATION SYSTEM

PRELIMINARY



PROCESS COMPUTER
BUSINESS SECTION
PHOENIX, ARIZONA

GENERAL ELECTRIC

Library Control No. YPG63M

This manual published by:

PROGRAMMING SUPPORT
GENERAL ELECTRIC COMPANY
PROCESS COMPUTER BUSINESS SECTION
P.O. Box 2918
PHOENIX, ARIZONA 85002

For further information concerning the contents of this manual, contact the Programming Library at the above address.

LIST OF EFFECTIVE PAGES

*INSERT LATEST CHANGE PAGES
DESTROY SUPERSEDED PAGES

The total number of pages in this publication is 223, consisting of the following:

Page	Date	Page	Date
i - ii	11/1/66	9-1	11/30/66
*iiia - iva	11/30/66	*9-2A - 9-4A	11/30/66
v - vi	11/1/66	*9-4.1	11/30/66
1-1	11/1/66	*9-5A - 9-6A	11/30/66
2-1 - 2-2	11/1/66	A-1 - A-5	11/1/66
3-1 - 3-2	11/1/66	B-1 - B-2	11/1/66
4-1 - 4-7	11/1/66	*B-3A	11/30/66
*5-1A	11/30/66	B-4 - B-5	11/1/66
5-2	11/1/66	C-1 - C-3	11/1/66
*5-3A - 5-4A	11/30/66	D-1 - D-9	11/1/66
5-5	11/1/66	*D-10A	11/30/66
*5-6A - 5-7A	11/30/66	D-11	11/1/66
5-8 - 5-12	11/1/66	E-1 - E-21	11/1/66
6-1 - 6-7	11/1/66	*E-22A	11/30/66
*6-8A	11/30/66	*E-22.1	11/30/66
6-9 - 6-10	11/30/66	E-23 - E-26	11/1/66
7-1 - 7-9	11/1/66	F-1 - F-15	11/1/66
*7-10A - 7-12A	11/30/66	G-1 - G-3	11/1/66
7-13 - 7-17	11/1/66	H-1 - H-4	11/1/66
*7-18A	11/30/66	I-1 - I-2	11/1/66
7-19 - 7-20	11/1/66	*I-3A - I-7A	11/30/66
*7-21A	11/30/66	I-8	11/1/66
7-22 - 7-29	11/1/66	J-1 - J-2	11/1/66
*7-30A	11/30/66	*K-1 - K-5	11/30/66
7-31 - 7-63	11/1/66	*L-1 - L-3	11/30/66
8-1 - 8-5	11/1/66	GLOSSARY(1-4)	11/1/66
*8-6A	11/30/66	*INDEX (1-7)	11/30/66
8-7 - 8-11	11/1/66		

Note: Revisions are indicated by vertical lines in outer margins except for typographical corrections, orientation, and illustrations.

TABLE OF CONTENTS

Section Page

SECTION 1

INTRODUCTION

SECTION 2

PROBLEM

SECTION 3

METHOD OF SOLUTION

SECTION 4

USING OPO

TABLE OF CONTENTS (cont.)

Section Page

SECTION 5

MODELS

SECTION 6

DATA SETS

SECTION 7

COMMANDS

R

TABLE OF CONTENTS (cont.)

Section Page

SECTION 8

TASK MANAGEMENT FUNCTIONS

SECTION 9

ERROR PROCESSING

APPENDICES

Appendix Page

TABLE OF CONTENTS (cont.)

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

SECTION 1 - INTRODUCTION

1.1 PURPOSE OF THE MANUAL

This manual describes external functional characteristics of the On-Line Process Optimization System (hereinafter referred to as OPO). The relationship between process and OPO is defined. The communication interfaces between the user and OPO are included. Related file management facilities are also described.

It is assumed that the user of this manual has a working knowledge of the GE/PAC MONITOR, process control, and linear programming techniques.

1.2 BRIEF DESCRIPTION OF ON-LINE PROCESS OPTIMIZATION SYSTEM

The OPO is a non-linear optimization system consisting of a set of functional programs designed to operate in conjunction with user-written process monitoring programs and user-written process simulation programs to achieve on-line process optimization. The system may also be used in non-control applications for the purposes of experimentation and verification and for the acquisition of management information. A variety of methods are provided to facilitate communications between users and OPO. A data management subsystem is included to minimize the amount of data which must be furnished by the user each time OPO is utilized.

OPO may be used for process optimization in an open or closed loop application. In all applications the user must supply a set of data which describes the problem and a functional program which simulates the process to be optimized.

SECTION 2 - PROBLEM

2.1 GENERAL PROBLEM

OPO is generally designed to determine settings of controllable independent process variables in such a manner that a process will operate in an optimum manner as defined by some objective function and subject to limiting values on some dependent process variables.

The independent process variables may also be limited to certain ranges of values.

It may be desirable to limit both the independent and dependent process variables. There may be "hard" limits which must not be violated under any circumstances. There may be "soft" limits which can be violated but the magnitude of the violation must be minimized. In later sections of this manual, "hard" limits will be referenced as *bounds* and "soft" limits will be referenced as *targets*.

Some dependent process variables depend in part upon their own value. Such variables are referred to in this manual as *implicit* variables.

Generally, the dependent variables and the objective function are non-linear functions of the independent variable.

2.2 EXAMPLE PROBLEM

To better illustrate the problem, a hypothetical process is discussed in the following paragraph of this section.

A reactor yields three products; X, Y, and Z. The amount of each yield product produced is a non-linear function of the temperature (T) and the pressure (P) at which the reactor is operated and the quantity of feed stock (F) fed into the reactor. The reactor should be operated by methods which tend to maximize the value of the products produced.

The amounts of each of the products which can be produced is limited. There may be marketing restrictions on how much of each product can be sold and there may be an obligation to produce at least some fixed minimum amount of each product. These limits are "hard" limits or *bounds*. There may be "soft" limits or *targets* on some products. For example, it might be possible to sell product X above a certain quantity only at the reduced price.

In addition, there may also be limits on the process variables. The feed stock quantity may not be less than zero and may also be limited by the supply available. Physical limitations of the reactor may place "hard" restrictions on operating temperature and pressure. Excessive maintenance costs due to very high operating temperatures and pressures may impose "soft" restrictions.

SECTION 3 - METHOD OF SOLUTION

3.1 RESTRICTED LP

The solution to the non-linear optimization problem is obtained by solving a series of highly restricted linear optimization problems.

Linearizing the problem about the starting values of the independent variables and applying upper and lower bounds to the independent variables makes it possible to solve a highly restricted LP and gain an improvement in the objective function. The problem can then be relinearized about the new set of independent variable values and another LP solution computed.

The linearization of the problem and evaluation of the dependent variables and objective functions are performed by calling the user-supplied process simulation program which is called a *model*.

Each such LP solution is a *step* toward the solution of the non-linear problem; the sequence of steps made in arriving at the solution of the non-linear problem constitutes a *move*. After each step, tests are made to determine whether a solution to the non-linear problem has been reached.

The bounds generated by the solution program to control the step size are compared with the bounds on the problem supplied by the user. The most restrictive of these bounds is used to control the step size.

3.2 LINEARIZATION

The non-linear dependent variables and objective function must be linearized about the position (value) of the independent variables in order to solve the linear program.

The partial derivatives of the dependent variables are computed at the position of the independent variables and these are used as the coefficients of the LP matrix.

The partial derivatives are computed with the aid of a subprogram which simulates the process being controlled. This program is supplied by the user and is referenced herein as a MODEL.

After the system is linearized the solution program determines the step size permissible. This step size is input by the user or is computed, based on the curvature of the functions. A detailed explanation is given in Section 5.

SECTION 4 - USING OPO

4.1 USER SUPPLIED REQUIREMENTS

In order to use OPO to optimize a process, the user must supply the following three items.

1) A functional program which simulates the process to be optimized. This program, given a set of values of independent process variables, produces numerical values describing the relationship of the dependent process variables and the objective function on the independent variables. This program is the *model* mentioned earlier. Characteristics of a *model* and its interface with OPO are described in Section 5.

2) A set of data which defines the process unit within the OPO system. These data comprise a *unit common data set*. When this data set is installed, all space used by OPO for data associated with that unit is reserved. The unit common data set contains the names of the process variables and may contain restrictions which are used to edit data entered later for that unit.

3) An *optimizer data set* (ODS) which is the actual problem to be solved. It includes the independent variables' starting values, the bounds and targets to be applied to the dependent and independent variables, and all data necessary to control the progress of the solution.

In addition to the above requirements, the user must provide process monitoring programs where the application includes closed-loop optimization.

The Optimizer Data Set (ODS) and the MODEL (M) which simulates the process are logically an entity which is denoted as ODS/M.

The following subsections describe how the user performs the actions required to optimize a process using OPO.

4.2 USER ACTIONS

1) The user writes a MODEL in either FREE-TIME FORTRAN or PAL and installs it in the GE/PAC MONITOR as a real-time functional program.

2) He prepares the input data for unit common and enters it in the system using a Load Unit Common (LUC) Command (refer to Section 7).

3) He prepares the input data for the problem and enters it in the system using a Build Test Data Set (BTD) Command (refer to Section 7).

4) He orders the execution of the optimizing algorithm using the ODS and its associated MODEL (ODS/M) with a SOLVE (SOL) Command (refer to Section 7).

5) If reports and results disseminated during execution of the optimizing algorithm are not sufficient for his needs, the user obtains additional information through the FILE, PRINT, and MATRIX Commands (refer to Section 7).

Assuming the MODEL to be installed in the GE/PAC MONITOR, the OPO responds to the above commands as follows:

1) Unit space is allocated in OPO bulk storage and the unit common data is stored after being processed. These actions are invoked by the LUC Command.

2) Execution of the BTD command results in allocation of ODS space in OPO bulk storage. Before the ODS data is stored, it is subjected to an edit using the restrictions in the unit common data. If any ODS data violates the editing restrictions, the user is so notified on the I/O typer and is given an opportunity to correct the error.

3) Utilizing the bounds, initial values, and other information - including the MODEL's program number - in the ODS, the optimizing algorithm produces an optimum set of independent process variable values. The optimum set of values may be displayed or passed to a functional process monitor program whose program number was specified as part of the ODS data.

4) Additional information, such as the reduced costs, may be displayed on the printer or transmitted to a communication area on the bulk memory device where it may be retrieved by any functional program knowing the location of the data.

4.3 OPO CAPABILITIES

The commands referenced in the preceding paragraphs are elements of the OPO command language. The user directs data input, control actions, communication with other functional programs, execution of the optimizing algorithm, and output through the medium of this language (refer to Section 7). The user presents his commands to OPO in the form of *tasks*. Because OPO is capable of optimizing several process units concurrently and can simultaneously maintain several optimizer data sets and associated MODELS for each unit, a *task* is a sequence of commands directed at a particular ODS/M in a specified unit. Tasks may be presented to the system without regard to the disposition of prior tasks; OPO queues them on the appropriate units and schedules their execution.

Tasks may originate from external or internal sources. The commands have an external format suitable for preparation on punched cards for entry into the system through the card reader. Internal sources such as a process monitoring program generate tasks in an internal format and store them in bulk memory for transmission to the system.

A set of *task management functions* are also provided as part of OPO. These functions are utilized by the user to order OPO to accept new tasks from both internal and external sources, delete tasks from the system, move a task to the front of its queue, and obtain information concerning the status of the system. Those task management functions which may be used by an external user are invoked and furnished with parameters through the GE/PAC MONITOR OPR Subroutine. An internal user initiates execution of a task management function by turning on the functional program containing the task management function and furnishing it with the location of a block of previously prepared parameter values. The task management functions, their parameters, and uses are further described in Section 8.

4.4 DEFINITIONS

Further description of the system requires the definition of additional terms. OPO data or unit common data of a given type that are associated with process variables are termed *vectors*. The remainder of the data, which are not associated with process variables, are *system parameters*. For example, a MODEL program number is a system parameter.

The user has the facility to select a subset of the process variables defined for a unit (refer to Section 7) to be *active* in a particular execution of the optimizing algorithm. Only variables designated as active in an ODS are considered by the optimizing algorithm. Variables that are required by MODEL, but not controllable, would be inactive.

Some dependent process variables may be *implicit* variables. These variables are, in part, functions of themselves and are described by the equation

$$Y_y = F(X, Y_x)$$

where y and x represent dependent and independent sets of process variables respectively. In order for a solution to be meaningful, $Y_y = Y_x$ is a requirement. The methods for handling implicit variables within OPO are described in Section 5.6.

The optimizer data sets and associated MODELs (ODS/Ms) in a unit may consist of no more than one control ODS/M and several test ODS/Ms. The *control ODS/M* in a unit is one that is actively controlling the process unit. The results of an optimization on the control ODS/M are utilized by an on-line functional program for transmission to the process. Only one control ODS/M can exist in a given unit at any one time. Before an ODS/M attains control status, it must pass through a test phase. A test ODS/M is used for experimentation and verification of an ODS and its associated MODEL. A test ODS/M may also be used for the acquisition of management information not normally required to be produced by a control ODS/M. Several test ODS/Ms may exist in a unit simultaneously.

The preceding paragraphs do not describe the full capabilities of OPO which are implied in the following list of principal OPO characteristics.

1) OPO can be used to optimize several process units concurrently.

2) OPO can simultaneously maintain several data sets in each unit.

3) OPO commands are furnished for data input and modification.

4) OPO automatically edits all input data according to user specifications.

5) OPO can perform its functions in response to tasks emanating from internal sources such as process monitoring functional programs in the GE/PAC 4000 Real-Time Library or from external sources via the card reader.

6) OPO can accept input data from both external and internal sources; it can output data to external and internal destinations.

SECTION 5 - MODELS

5.1 MODEL

During the course of an optimization, the OPO optimizing algorithm requires:

1) The distance of the current values of the dependent process variables and the objective function from their respective bounds, and

2) The incremental effects of a change in any independent process variables (x) on the values of the objective function (y_o) and on the values of the dependent process variables (y_i, i = 1,2,...).

In order that this information may be made available, the user must furnish OPO with a MODEL that, given a set of values of the independent process variables (X), computes and returns the corresponding value of the objective function (y_o) and the corresponding values of the dependent process variables (y_i, i = 1,2,...). This function is required of a MODEL. A MODEL may, *at the user's option*, also compute the partial derivatives (incremental changes) and return them to the OPO. This option is further described in Section 5.3.

Other than the required function and the optional function stated above, nothing more of a computing nature is expected of a MODEL. All problems of input, output, violation of bounds and targets are outside of a MODEL's domain. A MODEL is required to properly interface with OPO. The interface is described in Section 5.4.

5.2 MODEL COMPUTATIONAL METHODS

OPO places absolutely no requirements on the manner in which the vector Y ($Y = y_0, y_1, y_2, \ldots$) or the partial derivatives of Y are computed by a MODEL. However, because a MODEL is called many times in the course of an optimization, it should be programmed and coded to execute quickly and efficiently.

For mathematical accuracy and operating efficiency, the user must consider the following points when formulating and programming a MODEL.

1) All dependent process variables must be continuous.

2) If the MODEL incorporates iterative procedures in the computation of Y or the partial derivatives of Y, care must be taken that loose convergence does not lead to inaccurate results.

3) If the user decides not to compute the partial derivatives of Y in his MODEL, he should insure that the computed values of Y(X) are accurate.

4) The dependent variable vectors computed by MODEL must be stored in sequential locations starting in the bulk location specified in the communications area. The first dependent variable (Y) must be the objective function.

5) The independent variables are presented to MODEL as double precision numbers and the dependent variable values must be returned in double precision.

6) When implicit variables exist in a problem they exist in both the independent and dependent variable sets. If there are I implicit variables, these are represented in the first I independent variables ($X_1 - X_I$) and the first I dependent variables following the objective function.

5.3 MODEL TYPES

While a MODEL must be capable of computing Y(X), given X, the user may also choose to compute partial derivatives of Y in his MODEL. Further, he may choose between computing the partials of Y with respect to a given x ($x \varepsilon X$) or computing the partials of Y with respect to all x $\left(\frac{\partial Y(X)}{\partial x_j}, j = 1,2,\ldots\right)$.

Type 0 MODEL

A type 0 MODEL computes only the vector Y(X), given X. The incremental effects are calculated by OPO using the central difference approximation

$$\frac{\partial y_i}{\partial x_j} \sim \frac{y_i(X+\Delta x_j) - y_i(X-\Delta x_j)}{2\Delta x_j}, \; i = 0,1,2,\ldots$$

where $Y = (y_0, y_1, \ldots)$,

$X = (x_1, x_2, \ldots)$,

Y_0 represents the objective function,

and Δx_j represents an incremental change in x_j.

Type 1 MODEL

In addition to being capable of computing Y(X), a type 1 MODEL can also compute the vector

$$\frac{\partial Y(X)}{\partial x} = \left(\frac{\partial Y_0(X)}{\partial x_j}, \frac{\partial Y_1(X)}{\partial x_j}, \ldots \right)$$

for a given j indicated by OPO.

This type of MODEL should be utilized by the user when analytical methods for computing the partials are known and produce more accurate results than the central difference approximation used by OPO and described above. The use of a type 1 MODEL, as opposed to a type 0, may result in more efficient execution since OPO needs to make only one call on a type 1 MODEL to obtain the incremental effect of a change in x_j on Y.

Type 2 MODEL

A type 2 MODEL extends the function of a type 1 in that it is capable of computing, in response to a single call from OPO, the partials of Y with respect to each and every independent variable, i.e., the vectors

$$\frac{\partial Y(X)}{\partial x_1} \quad \frac{\partial Y(X)}{\partial x_2}$$

Here, OPO need make only one call on the MODEL where N calls on a type 1 MODEL would be required (N representing the number of independent process variables). A type 2 Model may be used only when the entire LP Matrix can fit in core.

A type 2 MODEL must also be capable of computing Y(X) when so requested by OPO.

5.4 OPO-MODEL INTERFACE

The interface between OPO and a MODEL consists of five items. Each item and its interpretation are discussed below.

1) MODEL Type Flag (MTF)

OPO calls on a MODEL with MTF set to either zero or to the MODEL's type:

a) When OPO calls for the execution of a type 0 MODEL, OPO always sets MTF to 0.

b) When calling a type 1 MODEL, OPO always sets MTF to 0 or 1.

c) OPO always sets MTF to 0 or 2 when calling a type 2 MODEL.

When a MODEL, regardless of its type, is called by OPO with MTF set to zero, it must perform the following two actions:

a) The MODEL must reset MTF to its type, i.e., a type 0 MODEL leaves MTF alone, a type 1 sets MTF to 1, and a type 2 sets MTF to 2.

b) The MODEL must compute Y(X).

When a type 1 MODEL is called by OPO with MTF set to 1, the MODEL is expected to compute the vector

$$\frac{\partial Y(X)}{\partial x_j}$$

where j is also set by OPO (refer to 5 below).

When a type 2 MODEL is called by OPO with MTF set to 2, the MODEL is expected to generate a complete LP matrix exclusive of the right-hand-side.

2) Bulk Controller Number (NBC)

Numerical values are communicated between OPO and a MODEL via bulk memory areas assigned by OPO. OPO sets NBC with the number of the controller in charge of the designated bulk memory device. A MODEL does not alter the contents of NBC.

3) Independent Process Variable Communication Area (LOCX)

OPO sets LOCX with the bulk memory address of the first independent process variable to be used by the MODEL in its current computations. The remainder of the independent variable values follow the first in double precision locations.

The called MODEL utilizes the contents of NBC and LOCX to locate X.

4) Dependent Process Variable Communication Area (LOCY)

OPO sets LOCY with the Bulk Memory address of the first location in the area where the MODEL is to deposit its computed results.

A type 0 MODEL stores the vector Y(X) in the area.

A type 1 MODEL, if called with MTF set to zero, also stores Y(X) in the area. If the MODEL is called with MTF set to 1, it stores $\frac{\partial Y(X)}{\partial x_j}$ in the area.

A type 2 MODEL, if called with MTF set to zero, stores Y(X) in the area. If called with MTF set to 2, the MODEL stores $\frac{\partial Y(X)}{\partial x_1}$, $\frac{\partial Y(X)}{\partial x_2}$,... in the area.

See Appendix K for MODEL result storage rules.

5) Index Value (JMODL)

This parameter, set by OPO, is utilized only by a type 1 MODEL. The value of JMODL specifies the independent process variable with respect to which partial derivatives of Y are to be taken, i.e., $\frac{\partial Y(X)}{\partial x_{JMODL}}$.

These five items are contained in common storage at location OPOMOD. Each item occupies one full word.

5.5 STEP LIMITS

When a type 0 MODEL is used, it is possible for the optimizer to determine the curvature of the dependent variables with respect to each independent variable and to limit the size of the step that may be taken by the solution before re-linearization is necessary.

The step limit is determined for each X as follows:

$$ES_x = \frac{\text{Min}}{\text{ally}} \; ES_{x'}, \; d_X\sqrt{\frac{dy}{\frac{Y(X=dx) \; - \; Y(x-dx)}{2} - Y(X)}}$$

where X_s is the step limit, dx is the increment used in computing partial derivatives with respect to X, and dy is user input for this calculation.

When type 1 or 2 models are used, step limits cannot be calculated as above and the user input values must be read.

The user may input both absolute and fractional step limits. In this case, the step limit used for the calculation is the largest of the two.

$$ES_x = \text{Max} \left(S_x, \; F_x \; (X) \right)$$

5.6 IMPLICIT VARIABLES

Implicit variables take the form $Y_y = f(X,Y_x)$. For a solution to be meaningful the condition $Y_y = Y_x$ must be satisfied.

This condition is satisfied in OPO by forcing the difference $Y_y - Y_x$ to be equal to zero.

Y_x is a member of the independent variable set and the difference $Y_y - Y_x = f(X,Y_x) - Y_x$ is a member of the dependent variable set.

If there are I implicit variables in the problem $(Y_y - Y_x)$ must be the first I members of the dependent set following the objective function $(Y_i - Y_I)$.

5.7 EXAMPLE MODEL

To illustrate the functions of a MODEL and its interface with OPO, an example of a MODEL which simulates a simple process is described below.

The process unit is a hypothetical reactor which yields three products: X, Y, and Z. The amount of each yield product is a function of temperature (T), pressure (P), and feed quantity (F).

$$X(F,P,T) = \{4(.004P + .78)\ [T-300)\ (\frac{300}{T^2})]\}\ F$$

$$Y(F,P,T)\quad \{.865 \times 10^{-3}\ (-.002P + 1.11)[(T-300)(\frac{300}{T})^{-1.5}]\}\ F$$

$$Z(F,P,T) = F - X(F,P,T) - Y(F,P,T)$$

The objective of the optimization is to determine the temperature and pressure settings that will maximize the total value of the reactor products. The objective function is

$$V(P,T) = 100\ X(F,P,T) + 125\ Y(F,P,T) - 55Z^2(F,P,T).$$

For the purposes of illustration, a type 1 MODEL is presented. To prepare a type 1 MODEL, the equations for the partial derivatives of the dependent process variables with respect to each of the independent process variables are required. These are:

$$\frac{\partial V(F,P,T)}{\partial F} = 100\ \frac{\partial X(F,P,T)}{\partial F} + 125\ \frac{\partial Y(F,P,T)}{\partial F} - 100\ Z(F,P,T)\ \frac{\partial Z(F,P,T)}{\partial F}$$

$$\frac{\partial V(F,P,T)}{\partial P} = 100\ \frac{\partial X(F,P,T)}{\partial P} + 125\ \frac{\partial Y(F,P,T)}{\partial P} - 110\ Z(F,P,T)\ \frac{\partial Z(F,P,T)}{\partial P}$$

$$\frac{\partial V(F,P,T)}{\partial T} = 100\ \frac{\partial X(P,T)}{\partial T} + 125\ \frac{\partial Y(P,T)}{\partial T} - 100\ Z(P,T)\ \frac{\partial Z(P,T)}{\partial T}$$

$$\frac{\partial X(F,P,T)}{\partial F} = 4(.004P + .78)[(T-300)(\frac{300}{T^2})]$$

$$\frac{\partial X(F,P,T)}{\partial P} = F(.016(T-300)\ (\frac{300}{T^2}))$$

$$\frac{\partial X(F,P,T)}{\partial T} = F(-1200(.004P + .78)T^{-2}(1-600T^{-1})$$

$$\frac{\partial Y(F,P,T)}{\partial F} = .865 \text{ x } 10^{-3}(-.002P+1.11)[(T-300)\left(\frac{300}{T}\right)^{1.5}$$

$$\frac{\partial Y(F,P,T)}{\partial P} = F\left(-1.73 \text{ x } 10^{-6}(T-300)(\frac{300^{1.5}}{T})\right)$$

$$\frac{\partial Y(F,P,T)}{\partial T} = F(-4.32 \text{ x } 10^{-4}(-\ 002P + 1.11)(\frac{300}{T})^{1.5}\ (1-\frac{900}{T})\ \Big)$$

$$\frac{\partial Z(F,P,T)}{\partial F} = 1.0 - \frac{\partial X(F,P,T)}{\partial F}\quad \frac{\partial Y(F,P,T)}{\partial F}$$

$$\frac{\partial Z(F,P,T)}{\partial P} = \left(\frac{-\partial X(F,P,T)}{\partial P}\quad \frac{\partial Y(F,P,T)}{\partial P}\right)$$

$$\frac{\partial Z(F,P,T)}{\partial T} = \left(\frac{-\partial X(F,P,T)}{\partial T}\quad \frac{\partial Y(F,P,T)}{\partial T}\right)$$

In the Free-Time FORTRAN Program presented below, the following relationship between mathematical symbols and programming symbols will exist:

Mathematical Symbol	Programming Symbol
F	X(1)
P	X(2)
T	X(3)
V(F,P,T)	Y(1)
X(F,P,T)	Y(2)
Y(F,P,T)	Y(3)
Z(F,P,T)	Y(4)

The partial derivatives of V, X, Y, and Z will also be stored in Y(1), Y(2), Y(3), and Y(4), respectively.

An additional note required before presenting the MODEL program: the permanent core symbols, MTF, NBC, LOCX, LOCY, and JMODL, are available to a FORTRAN Program only if the Free-Time FORTRAN compiler is used.

```
   SAMPLE TYPE 1 MODEL
C    MTF,NBC,LOCX,LOCY,JMODL, ARE DEFINED IN OPO COMMON
      DIMENSION X(3),Y(4)
C  OBTAIN VALUES OF INDEPENDENT PROCESS VARIABLES FROM BULK MEMORY
      READ DISC (NBC),(LOCX), X
C  CALCULATE COMMON SUB-EXPRESSION
      T1=X(3)-300.0
C  CALCULATE Y(X)
      Y(2)=(4.0 * (.004*X(2)+.78) * T1 * (300.0/X(3)**2)) * X(1)
      Y(3)=(0.865E-3 * (-.002*X(2)+1.11) * T1 * (300.0/X(3))**1.5)
     1                       * X(1)
      Y(4)=X(1) - Y(2) - Y(3)
C  BRANCH ON TYPE OF CALL
      GO TO (10,20), MTF+1
C  TYPE 0 CALL - SET MTF AND COMPUTE OBJECTIVE FUNCTION
   10 MTF=1
      Y(1)=100.0*Y(2) + 125.0*Y(3) - 55.0*(Y(4)**2)
      GO TO 60
C  TYPE 1 CALL - COMPUTE PARTIALS
C    SAVE Y(4)=Z(P,T)
   20 T2=Y(4)
C  BRANCH ON INDEPFNDENT VARIABLE NUMBER
      GO TO (25,30,40), JMODL
C    COMPUTE PARTIALS WITH RESPECT TO F
   25 Y(2)=4.0 * (.004 * X(2) + .78) * T1 * (300.0/X(3)**2)
      Y(3)=8.65E-4 * (-.002 * X(2) + 1.11) * T1 * (300.0/X(3))**1.5
      Y(4)=1.0 - Y(2) -Y(3)
      GO TO 50
C  COMPUTE PARTIALS WITH RESPECT TO P
   30 Y(2)=(.016 * T1 * (300.0/X(3)**2)) * X(1)
      Y(3)=(-1.73E-6 * T1 * (300.0/X(3))**1.5) * X(1)
      Y(4)=-Y(2)-Y(3)
      GO TO 50
C  COMPUTE PARTIALS WITH RESPECT TO T
   40 Y(2)=(-1200.0 * (.004*X(2)+.78) * X(3)**(-2)
     1                       * (1.0-600.0/X(3))) * X(1)
      Y(3)=(-4.325E-4 * (-.002*X(2)+1.11) * (300.0/X(3))**1.5
     1                       * (1.0-900.0/X(3))) * X(1)
      Y(4)= -Y(2)-Y(3)
C  COMPUTE PARTIAL OF OBJECTIVE FUNCTION
   50 Y(1)=100.0*Y(2) + 125.0*Y(3) - 110.0*T2*Y(4)
C  TRANSMIT RESULTS TO BULK MEMORY
   60 WRITE DISC (NBC),(LOCY), Y
      STOP
      END
```

Figure 5-1 Sample Type 1 Model

SECTION 6 - DATA SETS

6.1 INTRODUCTION

The data furnished by the user to OPO is divided into the following classifications:

1) Unit Common Data Set

   a) Required data
   b) Optional data

2) Optimizer Data Set (ODS)

   a) Required data
   b) Optional data

6.2 UNIT COMMON DATA SET

A unit common data set (unit common) consists of the names of the variables in the process unit, editing restrictions on ODS data, unit operating parameters, and other data pertinent to all ODS/Ms in the unit. Unit common is primarily utilized by the OPO data management facilities to perform file maintenance and to edit ODS input data. Unit common data must be entered in OPO before any ODS is constructed in the unit. The elements of a unit common are named and defined in Appendix B.

A unit common data set is divided into required and optional subsets (which are identified in Appendix B). The required subset consists of the process variable names and several parameters required for proper allocation of the bulk memory area to hold unit common and ODS data sets to be included in the process unit.

Some of the parameters in the optional data subset have default values. These are the parameters which specify the standard operator responses for error classes 1 through 5 (refer to Section 9). The remainder of the parameters are editing restrictions on ODS parameters.

All vectors in the optional subset of unit common are editing restrictions on ODS vectors. Each ODS vector name has two associated unit common vector names: 1) the name of the vector holding lower editing restrictions on the ODS vector, and 2) the name of the upper editing restriction vector. If a user furnishes OPO with data for an ODS vector (either required or optional), the corresponding unit common editing vectors are *not* required. If one or both of the editing vectors *is* furnished with data, the indicated type of editing is applied by OPO to any input ODS data.

6.3 OPTIMIZER DATA SET

An optimizer data set (ODS) consists of data utilized by the OPO optimizing algorithm. The components of an ODS are named and defined in Appendix A.

The required data subset must be furnished by the user in order for the optimizing algorithm to operate. One subset component of special interest is *MODL*. This is the program number of the user supplied process simulation program (MODEL).

The components of the optional data subset are not required for operation of the optimizing algorithm. The majority of these components are tolerance parameters and process variable bounds. If the user supplies data for them, the optimizer will utilize it.

Some components of the optional data subset have default values. If the user does not supply the components, OPO will automatically utilize system-defined values. These data items, which are identified in Appendix A, are required for operation of the algorithm, but it is not required that the user supply them.

ODS data is input to OPO via data statements used in conjunction with a BUILD TEST DATA SET (BTD) command (refer to Section 7.2.3)

6.4 DATA EDITING

OPO can edit for consistency those elements of an ODS for which the user supplies data. In general, editing consists of a comparison of an ODS datum against a lower restriction and an upper restriction as the datum is input into the ODS. Whenever an ODS datum violates an editing restriction, an error diagnostic is emitted and the operator is given an opportunity to correct the error. (Refer to Section 9 for a more detailed discussion of error processing).

The user controls OPO's performance of editing in the following manner:

1) Each ODS vector or parameter that may be edited has corresponding editing vectors or parameters in unit common (refer to Appendix B for the correspondence).

2) If the user supplies data for an editing vector or parameter, then that vector or parameter will be utilized; that is, editing will be performed on the appropriate ODS vector or parameter as it is being loaded.

Appendix B lists the editing parameters and vectors which may be added to a unit common data set in addition to the ODS parameters and vectors which they affect.

6.5 EXAMPLE

To illustrate the basic concepts of the preceding sections, an example of an external OPO job is presented below. The example consists of two parts:

1) Two tasks expressed in external command format.

2) Task management function calls that an external user might make with respect to the tasks.

NOTE

A third part, the program MODEL, appeared in Section 5.

```
BOT,AA,1,1,U                                                                    1
LUC,3,3,0,10128                                                                 2
DCL,UXUB,LXUB,UYUB,UXLB,LXLB,LYLB                                               3
SET,UYUB,1.0, LYLB,0.0                                                          4
PAR,UMXT,1050, UPMN,1.E-4                                                       5
TBX,    UXUB          LXUB          UXLB          LXLB                          6
 FEED   100000.0      99000.0       50000.0       5000.0
 PRES   150.0         149.9         55.0          .55E1
 TEMP   700.0         650.0         3.25E2        300.0
ENT                                                                             7
TBY,
 PRD1
 PRD2                                                                           8
 PRD3
ENT
DEF,UXUB,LXUB,UYUB,UXL3,LXLB,LYLB
END                                                                             9
EOT                                                                             10
BOT,AB,1,1,D1                                                                   11
BTD,                                                                            12
SET,XFML,1.E-2, YFML,1.E-2, YUPB,1.0, YLOB,0., XSTT,0., XUPB,0.,     C
    XLOB,0., XAML,0.,XASL,0., YAML,0.
PAR,MODL,46, RLOC,140300, RBCN,0, RPRG,22                                       13
TBX,    XSTT          XUPB          XLOB          XAML          XASL
 FEED   75000.0       100000.0      30000.0       10000.0       1000.0
 PRES   55.0          150.0         50.0          5.0           5.0E-3
 TEMP   600.0         700.0         300.0         10.0          1.E-2
ENT
TBY,    YAML
 PRD1   1.E-2                                                                   14
 PRD2   2.E-3
 PRD3   6.E-3
ENT
DEF,XFML,YFML,YUPB,YLOB,XSTT,XUPB,XLOB,XAML,XASL,YAML
END                                                                             15
SOL,1                                                                           16
FIL,/17567, XSTT,0, YSTT,64, RDCT,128                                           17
TOP,24                                                                          18
EOT                                                                             19
FIN                                                                             20
```

Figure 6-1 Example Tasks (External Format)

6.5.1 Tasks

Briefly, the function of the first task, AA, is to establish a unit and load the unit common data for that unit. The second task establishes a test ODS, loads the data for the ODS, calls for an execution of the optimizing algorithm, and transmits results to a real-time functional program. The encircled numbers to the right of the commands refer to explanatory notes following the tasks.

The following notes correspond to the encircled reference numbers in the right-hand-margin of the preceding listing.

1) The first task is identified by the characters AA, has priority 1, and is directed at unit 1's unit common data set (refer to BOT Command, Section 7.2.1).

2) The process unit will have 3 independent process variables, 3 dependent process variables, and no implicit process variables. The unit will require no more than 10,128 words of bulk memory storage (refer to LUC Command, Section 7.3.2).

3) Six optional unit common editing vectors are declared (refer to Sections 7.3.1 and 7.3.2 for further discussion of the DCL Command and Appendix B for discussion of unit common editing vectors).

4) The editing vectors UYUB and LYLB are initially loaded with the values 1.0 and 0, respectively. The SET data statement is discussed in Sections 7.3.1 and 7.3.2.

5) Two of the optional unit common parameters, UMXT and UPMN, are loaded with values (refer to Sections 7.3.1, 7.3.2, and Appendix B).

6) Four editing vectors associated with the independent process variables are loaded with data using a tabular input format. The names of the independent process variables are to be FEED, PRES and TEMP. (Refer to Sections 7.3.1 and 7.3.2.)

7) The value card deck following the TBX data statement is terminated by an ENT (END TAB) data statement. (Refer to Sections 7.3.1 and 7.3.2.)

8) The names of the three dependent process variables are to be PRD1, PRD2, and PRD3 (refer to Data Loading in Section 7.3.2).

9) The DEF Command causes the vectors named in the command to be marked as defined in the unit common data set and these named vectors will be used.

10) The END (END DATA) data statement terminates the LUC command. (Refer to Sections 7.3.1 and 7.3.2.)

11) The EOT (END OF TASK) command terminates task AA. (Refer to Section 7.2.2.)

12) Task AB has priority level 1 and is directed at ODS number 1 in unit 1.

13) A test ODS is to be constructed in the unit. (Refer to Section 7.3.3).

14) The real-time functional program number, 46, of the MODEL to be associated with this ODS is loaded into MODL. The OPO optimizing algorithm is to store the optimized independent process variable values in bulk, starting in bulk location 140300_8. The disk is controlled by disk controller number 0. The algorithm is to turn on real-time functional program number 22 which will utilize the results. (Refer to Section 7.6 and Appendix A.)

15) Various ODS vectors are to be loaded with data using the tabular input format. (Refer to Section 3.3.2 and Appendix A.)

16) The DEF command causes the vectors moved in the command to be marked as defined and they may be used by subsequent commands directed at the optimize data set.

17) The END command terminates the BTD command.

18) The optimizing algorithm is to operate on the ODS using its associated MODEL. A scratch basis will be used for the first step. (Refer to Section 7.6.)

19) The contents of the vectors XSTT, YSTT, and RDCT are to be stored in a bulk memory communication area which is specified in the interlock status block located at address 17567_8 in permanent core. (Refer to Section 7.5 and Appendix H.)

20) Real-time functional program number 24 is turned on, presumably to utilize the data placed in the communication area by the preceding FILE command (refer to Section 7.5.1).

21) Task AB is terminated.

22) The FIN card terminates the task sequence.

6.5.2 Task Management Functions

An external user is required to call on the task management function GET TASK in order to enter a task into the system from the card reader. In addition, the example below contains a call on the STATUS function to obtain a report on the physical structure of the unit and the data sets.

The example is in the form of a facsimile of the communication between operator and OPO on the I/O typer. Lines typed by OPO or the MONITOR are denoted by a † in the left-hand-margin; lines typed by the operator, by *. Neither the † or * will actually appear on the I/O typer output. It is assumed that the operator has loaded the tasks from the preceding section in the card hopper and has depressed the input button on the I/O typer.

```
* OPOS
† CARD TASK-1,DELETE-2,MOVE UP-3,STATUS-4
* 1
† END JOB
† END TASK AA
† END TASK AB
```

Here, the two tasks have been executed by OPO and the operator wishes to check the status of the task queues. He again depresses the input button on the I/O typer.

```
*  OPOS
†  CARD TASK-1,DELETE-2,MOVE UP-3,STATUS-4
*  4
†  QUEUE STATUS REPORT   124036PM   10-23-66
†  OPO BULK  /604400  0  /700000
†  UNALLOCATED OPO BULK  /631700  /562500
†  UNIT 1 DESCRIP/16043  UNIT BULK  /606000  /23700
†  QUEUE EMPTY
†  DATA SETS    BASE      LENGTH
†  U            /606000   /1000
†  1            /607000   /3500
†  END REPORT
†  END JOB
```

Task management functions are described more fully in Section 8.

SECTION 7 - COMMANDS

The commands defined in the OPO system provide facilities which enable a user to establish models in the system, order the execution of the optimizing algorithm using the data in any one of the optimizer data sets, and transmit the results produced by the algorithm to any of several destinations, depending upon how the results are to be utilized.

Specifically, the commands may be separated into six groups as follows:

1) TASK DELIMITING COMMANDS
2) INPUT COMMANDS
3) CONTROL COMMANDS
4) COMMUNICATION COMMANDS
5) SOLVE COMMAND
6) OUTPUT COMMANDS

The task delimiting commands demarcate a task for OPO. The input commands are used to establish units and data sets and to enter data into the system through peripheral devices. One of the communication commands allows the user to obtain data for the system from functional programs not incorporated in OPO.

The SOLVE command is used to order an execution of the optimizing algorithm.

Displays of the solution results, in addition to other pertinent information, are accomplished through use of the output commands. If the results are to be utilized by other GE/PAC functional programs, one of the communication commands will accomplish the required transfer of data. Output commands are also used to produce printouts of system information for debugging purposes.

The final group, control commands, furnish facilities for the deletion of units and data sets, installation of control data sets, and synchronization of the executions of tasks with those of external functional programs. A fallback command is also provided which allows a unit to be easily reestablished should it be inadvertently destroyed.

7.1 COMMAND REPRESENTATION

OPO commands and data statements have both an external and an internal representation. The external representations, which are used for tasks entering OPO through the card reader, are card-oriented and designed for readability. The internal representations are utilized for tasks generated by functional programs not incorporated in OPO, e.g. a process monitoring, real-time, functional program. The internal representations are designed for compactness and for ease in generation by functional program.

The external representations, because of their readability, are used throughout the discussion of commands and data statements in this section. The internal representations are not presented in this section, but are found in Appendix D.

7.1.1 General External Representation

All OPO commands and data statements, with the exception of tabular input statements (TBX, TBY, and value statements) share a common card layout. The command and data statement keywords appear in columns 1 through 3. A comma (,) in column 4 separates the keyword and parameter fields. If the parameter field is blank, the comma need not appear. The command and data statement parameters start in column 5 and may extend through column 67. Column 68 is used for command and statement continuation control. The remainder of the card, columns 70 through 80, is used for program identification and sequencing. Figure 7-1 depicts the card fields while Table 7-1 defines the possible contents of the card fields.

7.1.2 Continuation Control

When the parameter field of a single card cannot accommodate all the desired parameters, the command may be continued into the parameter field of another card by placing a "C" in column 68 of the card being continued.

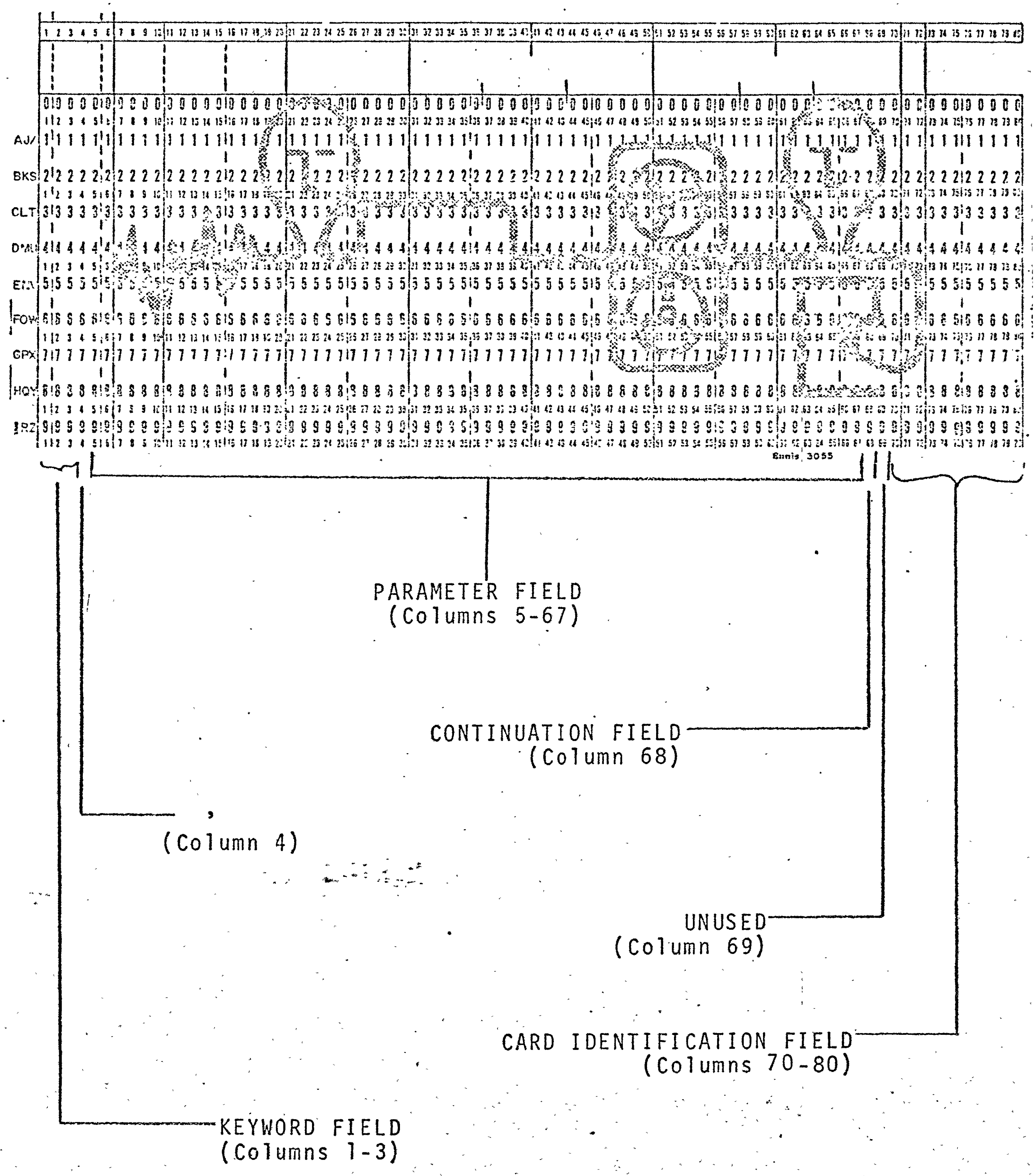
There is no restriction on the number of cards comprising a command or statement. The only restrictions involved in continuation are:

1) A "C" must appear in column 68 of each card, except the last, in the command or data statement.

Table 7-1 Command and Data Statement Format[†]

Columns	Field
1-3	*Keyword Field:* Contains one of the following keywords - BOT, EOT, LUC, BTD, MOD, DCL, PAR, SET, ALT, DEF, ENT, END, FIL, GET, ATD, DTD, DUC, TOP, SAV, SOL, PRT, MAT, or DMP. The keyword field is utilized only in the first card of a command.
4	Column 4 contains a comma (,), separating the keyword and parameter fields.
5-67	*Command and Statement Parameter Field:* Contains the parameters (if any) of the comnabd or data statement separated by commas. The various types of parameters are: 1) vector names, 2) variable names, 3) task identifiers, 4) data set indicators, 5) decimal, floating-point numbers, 6) decimal, integer numbers, and 7) octal, integer numbers. Two consecutive blanks will terminate the parameter field prior to column 67.
68	*Continuation Field:* A "C" is punched in this column of each card that is continued. This "C" should appear in each card of a command or statement except the last card.
70-80	*Card Identification Field:* May be used for a card sequence number, program identification, or left blank.

[†] The format described in this table and in Figure 7-1 does not apply to tabular input statements (TBX, TBY, and Value cards). Their format is described in Section 7.3.1.



[†] Tabular input statements (TBX, TBY, and Value statements) do not conform to this format (refer to Section 7.3.1).

Figure 7-1 External Format Card Fields

2) The keyword field (columns 1-3) and column 4 may not be utilized on any card of the command except the first. Neither can column 4 be utilized on any card in the command.

3) A parameter must not be split between two cards; e.g. a portion of a name must not appear on one card while the remainder of the name appears in the parameter field of the next card.

7.1.3 OPO Character Set

The OPO commands and data statements, in their external representation, utilize the following character set.

Alphabetic	A through Z
Numeric	0 through 9
Special Characters	+ - / . , blank

The elements of OPO commands and data statements are constructed of characters drawn from the above set. Typically, OPO commands and data statements consist of a keyword followed by a list of parameters. Parameters may be classified on the basis of structure as follows: vector names; variable names; task identifiers; data set indicators; activity indicators; task priority; decimal, floating-point numbers; decimal integer numbers; and octal, integer numbers. The form of each type is defined in the subsections below.

7.1.4 Vector Names

Vector names, which are system-defined, consist of exactly four alphabetic characters. The vector names are listed in Appendices A, B, and I.

Examples: XSTT YLTR RDCT

7.1.5 Variable Names

The names of process variables, which are defined by the user, may consist of one to four characters. The comma (,) is the only character which must not be used in a variable name. The first character of a name must be an alphabetic character:

Examples of valid variable names:

PRES
X1
Y/2
AB-4
I6B
Y3/2

Examples of invalid variable names:

4AB
/B3
TEMP1
+AB

7.1.6 Task Identifiers

Task identifiers are defined by the user and must consist of exactly two characters. The comma (,) and the blank are the only two characters which may not be used in a task identifier.

Examples of valid task identifiers:

AA
43
B6
8Z
/A
4.

Examples of invalid task identifiers:

B TSK C,

7.1.7 Data Set Indicators

There are three forms of data set indicators.

Form 1: U

U implies the unit common data set in a unit.

Form 2: C

C implies the control optimizer data set (control ODS) in a unit.

Form 3: Dn , $0<n\leq 9$

Dn implies the nth ODS in a unit, based on the order in which the optimizer data sets were built.

7.1.8 Activity Indicators

There are two forms of activity indicators.

Form 1: A

A implies activate.

Form 2: I

I implies deactivate.

7.1.9 Task Priority

A task priority is a decimal integer n in the range $0 \leq n \leq 99$. Priority is directly proportional to the ascending integer sequence 0,1,..., 98,99.

7.1.10 Decimal, Floating Point Number

Decimal, floating point numbers are represented by a string of up to 11 decimal digits and either a decimal point or an exponent representing a power of ten. The permitted forms are:

nE n.   .n   n.n   nE±e   n.E±e   .nE±e   n.nE±e

n is the base; e is the exponent to the base 10. The range of |E| is 0 through 77. When e is positive, the sign preceding e may be omitted. Each of the forms may be preceded by a + or - sign. Single imbedded blanks will be suppressed.

Examples:

-0.999 999 99999E77   .4   4E
46E-3   3.E+62

7.1.11 Decimal Integer Numbers

A decimal integer number is represented by a string of up to seven decimal digits. Decimal integers are restricted to the range 0 through 8,388,607. Only positive decimal integers are valid.

Examples:

4   62308   994

7.1.12 Octal Integer Numbers

An octal integer is represented by a slash (/) followed by a string of up to eight octal integers. Only positive octal integers are valid. The octal integer range is 0 through 37777777_8.

Examples:

```
/4     /37777777     /602
```

7.2 TASK DELIMITING COMMANDS

The BEGIN TASK (BOT) and END TASK (EOT) commands serve as task delimiters. A task is a sequence of commands directed to a particular data set within a specific unit. This data set may be the unit common, the control ODS, or any one test ODS in the unit.

7.2.1 BEGIN TASK Command

Syntax

Column 1 5

BOT,n_1,n_2,n_3,n_4

n_1 - task identifier

n_2 - task priority

n_3 - unit number; a decimal integer

n_4 - data set indicator

Example:

```
Column     1   5
           BOT,24,46,3,D4
```

A BOT command must be the first command in a task. In addition to denoting the beginning of a task, the BOT command has four associated parameters which describe the task to the OPO. These parameters are:

1) Task Identifier - This user-supplied identifier identifies the task to the OPO and is specified by the user whenever he wishes to move the task or delete it. The identifier should be unique within the unit to which the task is directed.

2) Unit Number - The unit to which the task is to be applied must be identified.

3) Data Set Indicator - The data set indicator specifies which data set within the unit is to be the object of the task. There are three permissible settings for this parameter: unit common (U), control ODS (C), or the letter D followed by n representing the number of a test ODS ($0<n\leq9$).

4) Task Priority - The user assigns his task a priority which is used by the OPO in scheduling the execution of the tasks. A tasks' priority remains constant for the life of the task. The priority value is a decimal integer in the range $0<n\leq99$.

7.2.2 END TASK Command

Syntax

```
Column    1
          EOT
```

The END TASK command serves only as a task delimiter and marks the end of a task.

7.2.3 FIN Control Card

When a series of task are submitted through the card reader, the last task must be followed by a FIN card.

Syntax

```
Column      1
            FIN
```

7.3 INPUT COMMANDS AND FORMATS

Three commands are provided to furnish data input facilities. These are:

LOAD UNIT COMMON	(LUC)
BUILD TEST ODS	(BTD)
MODIFY	(MOD)

The first two commands, LUC and BTD, are used for unit common and ODS establishment, respectively. MOD is used to modify and update data sets after the unit and its data sets are established.

LUC, BTD, and MOD commands physically precede specifications of data. The various types of data cards that comprise the data specifications are described in the following subsection. The input commands and their uses are described in later subsections.

7.3.1 Data Formats

There are six types of data cards: Declare (DCL), System Parameter (PAR), Set (SET), TAB (TBX and TBY), Define (DEF), and Alter (ALT). Two additional statements, the End Data statement (END) and the End Tabular statement (ENT), are used to denote the end of a sequence of data specifications.

The format of each data card is a combination of fixed and free field. Certain information on the cards must be punched in system-defined positions on the cards.

1) DECLARE - The Declare card contains a list of system-defined vector names. It has the following format:

Syntax

```
Column    1   5
          DCL,v1,v2,...
```

v_i - a four character vector name (refer to Appendices A and B).

Example

Column 1 5

```
        DCL,XDEL,YUTR,YDEN,XLTR
```

This card is used in conjunction with the LUC and BTD commands to declare the vectors that will be utilized in unit common and optimizer data sets. Declaration of a vector name causes an area in OPO bulk storage to be allocated to the vector.

Vector names that may appear in a DCL statement are defined in Appendices A and B.

2) SYSTEM PARAMETER - The PAR statement is used to specify the values of system parameters.

Syntax

Column 1 5

$\mathrm{PAR}, p_1, v_1, p_2 v_2, \ldots$

p_i - a four character system parameter name (refer to Appendices A and B).

v_i - a numerical value which may be:

(1) decimal, floating point

(2) decimal, integer

(3) octal, integer

Example

Column 1 5

```
        PAR,MXIT,1050,RLOC,14025,DZRO,1.5E-6
```

The parameter names are system-defined. The permissible values vary with the parameter and its definition (refer to Appendices A and B). PAR statements may be used with a LUC, BTD, or MOD command.

3) SET - The SET statement is used to specify initialization options for data set vectors. The statement appears as

Syntax

```
Column    1   5
              SET,v1,o1,v2,o2,...
```

v_i - a four character vector name

o_i - a set option which may be:

(1) decimal, floating point number,

(2) Dn, where n is a model number, $0<n\leq 10$

(3) A

(4) I

Example

```
Column    1   5
              SET,XUPB,0.,PVAR,A,YLOB,D6,XDEL,2.0E-4
```

There are four SET options defined. The last two options (c and d) apply only to the process variables. These four options are:

a) ODS Number

b) Initialization Value

c) Active

d) Inactive

The ODS number option means that values for the associated vector are to be obtained from the vector of the same name in the specified ODS. This option may be used only when the SET card is under the control of a BTD or MOD command. It may not be used with a LUC command.

The initialization value option is expressed as a double precision, floating point numerical value. All elements in the associated vector are filled with this numerical value. It may be applied to vectors in a unit common or in an ODS.

The active and inactive options apply only to selection of the set of process variables that will be active in the LP problem of a particular ODS. This selection will be discussed in the descriptions of the BTD and MOD commands.

4) TAB (TBX and TBY) - The TBX and TBY cards and their associated value cards provide a tabular input form for loading elements of several vectors with a single record from the system input device. The TAB cards specify vector names and establish a correspondence between card fields and those names. Succeeding value cards contain the data to be loaded into the vectors. Continuation control does not apply to tabular input data statements and value cards.

TBX Card

Syntax

Column	1	9	21	33	45	57
	TBX,	v_1	v_2	v_3	v_4	v_5

v_i - a four character name of a vector associated with independent process variables

Example

Column	1	9	21	33	45	57
	TBX,	XUPB	XLOB	XUTR	XLTR	XFML

TBY Card

Syntax

Column	1	9	21	33	45	57
	TBY,	v_1	v_2	v_3	v_4	v_5

v_i - a four character name of a vector associated with dependent process variables.

Example

Column	1	9	21	33	45	57
	TBY,	YUPB	YLOB	YUTR	YLTR	YFML

Value Cards

Syntax

Column	2	9	21	33	45	57
	x	e_1	e_2	e_3	e_4	e_5

x - a one to four character process variable name
e - a floating point, decimal number

Example

```
Column   2     9           21     33        45      57
         XX4A  +.043692E+13  0.2  -4.8E-2   .5E-5   60.
```

ENT Card

Syntax

```
Column   1
         ENT
```

Example of tabular input:

```
Column   1      9         21        33        45        57
          TBX,  XDEL      XLTR      XFML      XFSL
          PRV1  .556E-6   2.0       .10       .5E-1
          NPTG  .75E-3    2000.0    .02       .3E-3
         ENT
         TBY,   YUPB      YLOB      YUTR      YLTR      YDEL
          MTHG  1500.0    750.0     1495.     762.0     .33E-4
         ENT
```

The vector names (v_1,v_2...) are system-defined. The process variable names (x) are user-supplied and *must* appear on the value cards. The combination of vector name and variable name define a particular field on a particular card; that field contains the datum destined for the element of the vector associated with the variable. A blank field at a variable-vector intersection is interpreted to be zero.

TBX is used when the succeeding variables pertain to the set of independent process variables. TBY is used with variables from the dependent variable set which includes the implicit variable set. The ENT statement terminates a sequence of value cards. The use of these forms is further explained in the subsections on LUC and BTM commands.

5) ALTER - An ALT statement is used to assign a value to a single datum in a vector. There may be more than one such assignment specified in an ALT statement, e.g.,

Syntax

```
Column     1    5
           ALT, c1,r1,v1,c2,r2,v2,...
```

c_i - a four character vector name

r_i - a one to four character variable name

v_i - a decimal, floating point number
or
an activity indicator (A or I)

Example

```
Column     1    5
           ALT, XUPB,XX1A,2.0E-1,XLOB,XX46,4.576E-8,PVAR,XX1A,A
```

6) DEFINE - A DEF statement is used to "define" vectors. A vector will not be utilized by OPO unless its name has been specified in a DEF statement. This statement is included in the system in order that a vector will not be utilized unless it contains meaningful values.

Syntax

```
Column      1     5
            DEF,  v1,v2,...
```

v_i - a four character vector name

Example

```
Column      1     5
            DEF,  XDEL,YASL,XLTR
```

7) END DATA - The END statement carries no data specifications. It is used to mark the end of a sequence of data cards of the types described above and to terminate a LUC, BTD, or MOD command.

Syntax

```
Column      1
            END
```

7.3.2 LOAD UNIT COMMON Command

The LUC command causes the OPO to allocate space for a unit and to prepare for loading of editing data. The data specifications must immediately follow the LUC command. The specifications consist of declarations of vectors, definitions of process variable names, and specification of values for vector elements.

Syntax

```
Column       1   5
             LUC, n1,n2,n3,n4
```

n_1 - number of independent process variables; $n_1 \leq 100$

n_2 - number of dependent process variables; $n_2 \leq 100$

n_3 - number of implicit process variables; $n_3 \leq n_2$

n_4 - number of words on disk required for all ODS and unit common data in the unit

Example

```
Column       1   5
             LUC, 42,73,12,150000
```

The first three are self-explanatory. The fourth is the number of words required to accommodate, in bulk memory, all the data sets that will exist in the unit at any one time. This estimate will be calculated by the user on the basis of formulas given in Appendix G.

Unit Common System Parameters

Unit Common system parameters consist of editing restrictions on ODS system parameters and standard operator responses for error classes 1 through 4. The names and definitions of unit common system parameters are to be found in Appendix B.

The user sets a parameter value by specifying the parameter's name and the desired value in a PAR data statement. The user is not required to furnish values for unit common parameters. The various standard error responses have system-defined default values (see Appendix B) while an unvalued editing parameter implies that editing will not be performed on the corresponding ODS parameter.

Figure 7-5 is an example LUC command and contains a PAR data statement.

Vector Declarations

DCL statements which identify the editing vectors that will be utilized in this unit causes the allocation of the proper amount of space for the specified unit common vectors.

The declaration of a vector does not mean that the vector will be unequivocally utilized by the OPO; the vector must also be specified in a DEF statement before OPO considers it defined and available for use. The declaration of a vector insures only that space will be available for storage of the vector's values at the time they are loaded. The required DEF statement does not have to be specified in the LUC command; it may be placed in the data specifications of a subsequent MOD command.

Unit common vectors do not have to be specified in a DCL statement in order to be allocated bulk memory space. Space will be allocated for a previously unallocated vector whenever its name is encountered in a DCL, SET, TBX, TBY, or ALT data statement used in conjunction with an LUC command.

The vectors in unit common are editing vectors. If an editing vector is not defined (in the OPO sense), the type of editing it represents will not be applied to the optimizer data sets in the unit. All unit common vectors are named and defined in Appendix B.

Initial Value Specification

When the user wishes to load all the elements of a vector with the same datum, he may do so by using a SET statement with the initialization value option.

The initialization value option is the only option that may be used with SET under control of the LUC command.

The statement

```
SET, UXST, 2.30E+2, UXUT, 60.5E-2
```

causes each element of the vectors UXST and UXUT to contain the values 230.0 and 0.605, respectively.

If a vector specified in a SET statement is not currently allocated (e.g. via a DCL statement), it's appearance in the SET statement will result in a bulk storage allocation for it.

Data Loading

Data destined for vectors may be specified using TBX and TBY cards. Because only vectors are loaded using the TAB cards, all data on the accompanying value cards are associated with the process variables. The vectors are divided into two groups: those associated with the independent variables and those associated with the dependent variables. All vectors in a group have the same length. The groups are loaded separately; all data associated with the independent variables are loaded before the dependent variable data or vice versa.

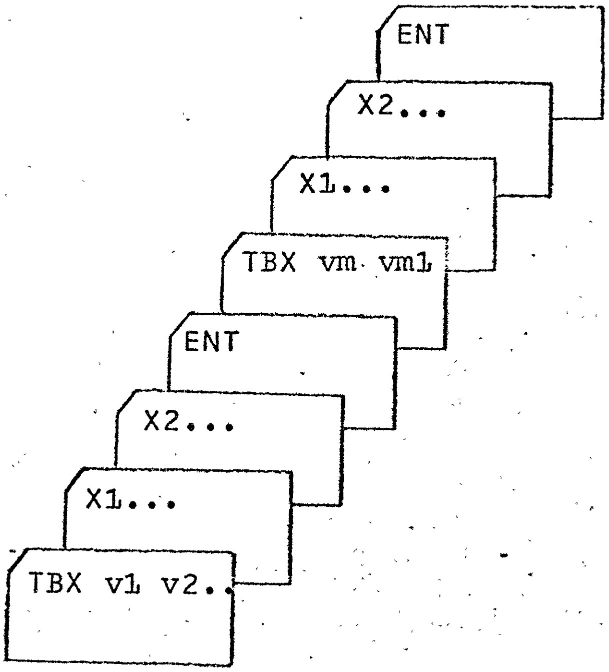


Figure 7-2 Tabular Data Card Sequence

More than one set of TAB and value cards are permitted within a group when there are more vectors to be loaded than can be named on a single TAB card. Variable names are required on each set of value cards.

The variable names on TBX and TBY value cards have special significance when used with the LUC command. The appearance of a variable name on a value card following the first TBX or the first TBY card after an LUC command defines that variable as a process variable. If a TBX card is involved, the variable is an independent process variable; in the case of a TBY card, it is a dependent process variable. A name must appear in the first set of TBX or TBY value cards to be defined as a process variable.

```
Column     1       9          21

           TBX     UXST       LXST
            X1     24         22.
            X2     78         70.
            X3     39         38.
           ENT
           TBX     UYAM       UXAS
            X4     5E-4       25E-6
           ENT
```

Figure 7-3 Variable Name Definitions

In the example in Figure 7-3, the independent process variables for the unit will be X1, X2, and X3; X4 will not be a process variable because it did not appear on a value card following the first TBX. An analogous situation exists with respect to TBY and dependent variables.

Figure 7-4 is an example of tabular input and includes TBX, TBY, value, and ENT statements. The vectors specified in TBX and TBY statements are unit common editing vectors (refer to Appendix B).

```
Column 1    9           21          33          45          57

     TBX    UXST        LXST        UXUB        LXUB        UXLB
      XVR1  1.25        5           1.75        1.20        .75
      XVR2  48.0        43.0        53.0        51.         36.0
       .
       .
       .
      XVRK  10.         8.0         15.         10.         7.5
     ENT
     TBX,   LXLB        UXAM        LXAM        LXAS        UXAS
      XVR1  .33         .5E-4       .25E-4      .28E-5      .14E-5
      XVR2  32.         .876E-5     .42E-5      .5E-6       .25E-6
       .
       .
       .
      XVRK  7.3         .5E-2       .45E-2      .777E-4     .33E-4
     ENT
     TBY    UYUB        LYUB        UYLB        LYLB        UYAM
      YV1   24.0        21.0        21.0        17.0        .4E-2
      YV2   1430.       1400.       1375.0      1250.       4.0
       .
       .
       .
      YVN   63.         60.0        59.0        50.         .5E-3
     ENT
     TBY,   LYAM
      YV1   .2E-2
      YV2   1.0
       .
       .
       .
      YVN   .1E-3
     ENT
```

Figure 7-4 LUC Tabular Input Example

Datum Changes

Another data statement that may be used in conjunction with the LUC command is the ALT statement. Its primary use in this context is to set isolated values in a vector that was initialized through specification in a SET statement with the initialization value option. It may also be used to change values that were loaded using the TBX and TBY cards.

ALT statements used in conjunction with LUC commands must follow the required TAB cards in order that the process variable names will be defined prior to the appearance of the ALT.

If a previously unallocated vector is encountered in an ALT statement, bulk storage will be assigned to the vector as a consequence of the vector's appearance in the ALT statement. That is,

```
Column     1   5
           ALT,UXST,XV1,42.0
```

and

```
Column     1   5
           DCL,UXST
           ALT,UXST,XV1,42.0
```

are equivalent.

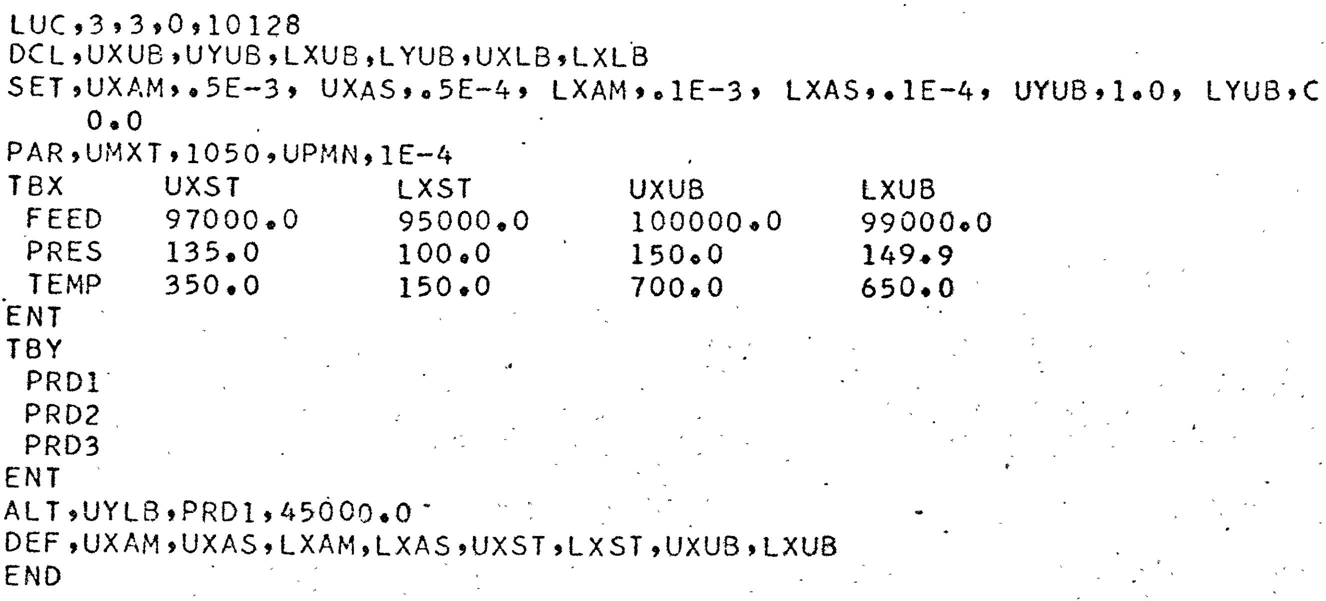
Vector Definition

The final type of data statement that may be used with an LUC command is the DEF statement. All vectors specified in a DEF statement should be fully loaded with data prior to the appearance of the DEF statement. If the user specifies in a DEF statement the name of an editing vector not fully loaded with data, editing will be performed with meaningless restrictions.

Only editing vectors that have appeared in a DEF statement will be used to edit input data destined for an optimizer data set.

Example LUC Command

Figure 7-5 depicts an LUC command which utilizes all of the various types of data statement.



```
LUC,3,3,0,10128
DCL,UXUB,UYUB,LXUB,LYUB,UXLB,LXLB
SET,UXAM,.5E-3, UXAS,.5E-4, LXAM,.1E-3, LXAS,.1E-4, UYUB,1.0, LYUB,C
    0.0
PAR,UMXT,1050,UPMN,1E-4
TBX     UXST          LXST          UXUB          LXUB
 FEED   97000.0       95000.0       100000.0      99000.0
 PRES   135.0         100.0         150.0         149.9
 TEMP   350.0         150.0         700.0         650.0
ENT
TBY
 PRD1
 PRD2
 PRD3
ENT
ALT,UYLB,PRD1,45000.0
DEF,UXAM,UXAS,LXAM,LXAS,UXST,LXST,UXUB,LXUB
END
```

Figure 7-5 Example LUC Command

Several points may be noted in the figure:

1) The DCL data statement does not contain all of the editing vectors specified in the entire command.

2) None of the vector names specified in the SET statement appear in the DCL statement; nevertheless, bulk memory space will be assigned to these vectors as a consequence of their appearance in the SET statement.

3) The SET statement requires two cards, involving the use of continuation control.

4) Two of the vectors in the TBX statement, UXST and LXST, do not appear in the DCL statement. Bulk memory space will be assigned to the two vectors due to their appearance in the TBX statement.

5) The TBY statement contains no vector names. The associated value statements exist solely to define the names of the dependent process variables.

6) UYLB, specified in the ALT statement, appears in no other statement in the command. Bulk storage will be allocated for UYLB because of its specification in the ALT statement.

7) The ALT statement, which specifies a value for the element of UYLB associated with the dependent process variable PRD1, is specified after the dependent process variable names have been defined.

8) Only the editing, or unit common, vectors specified in the DEF statement, will be used in subsequent editing of data destined for an ODS. This statement is based on the assumption that a succeeding MOD command directed at unit common does not 'enable' additional editing vectors.

7.3.3 Build Test ODS

The BTD command prepares the OPO system for the establishment of an optimizer data set. Establishment consists of allocating space for the ODS and loading of data. The data specifications for the ODS are expressed in data statements which follow the BTD command. The process of building a test ODS is similar to loading a unit common data set. However, the two differ in detail.

The optimizing algorithm is not required to operate on the entire set of process variables in the unit. OPO furnishes facilities for designating a subset of the set of process variables to be optimized. Therefore, establishment of an ODS includes setting system parameters, declaring vectors to be utilized in the ODS, loading data into the ODS vectors, and designating the variables to be active in the optimization.

The data loaded into test ODS vectors during execution of a BTD command are subject to editing. The actual application of a type of editing depends upon the definition of the corresponding editing vector in unit common. If an editing vector is defined (specified in an LUC DEF statement) in unit common, the type of editing it represents is applied; otherwise, it is not.

Syntax

```
Column     1   5
           BTD,m
```

m - an ODS data set indicator (C or Dn where 0<n≤9

Examples

```
Column     1   5
           BTD,D4
           BTD
```

BTD Command Parameters

The BTD has a single, optional parameter. This parameter is an ODS specifier and, if specified, implies that the system parameters in the ODS being built are to be initialized with values identical to those of the system parameters in the indicated ODS.

Any system parameter obtained with this option may be changed via a PAR statement following the BTD command.

ODS System Parameters

The system parameters in an ODS may be set using a PAR statement. If the system parameter option appeared in the BTD command, values specified in PAR statements will supersede values obtained as a result of the option's presence.

The system parameters in an ODS are used primarily for holding information required by the optimizing algorithm. Some are utilized by the data management portions of OPO. Some of these parameters have default values; e.g., if the user does not set a parameter, either with the BTD option or in a PAR statement, OPO will set it with a system-specified value. The ODS system parameters are defined in Appendix A and default values are specified where applicable.

Vector Declarations and Definition

The ODS vectors are named and defined in Appendix A. They may be classified as required or optional. The user has the responsibility of ensuring that the right required vectors are fully loaded with data prior to the execution of the optimization algorithm. Required vectors do not have to be specified in either a DCL or a DEF statement; they are both allocated and defined automatically before any data statement in a BTD command is processed.

All optional ODS vectors that are to be eventually utilized must be specified in a DCL statement in order that space be reserved for them in the OPO data files. After they have been loaded with data, they must be specified in a DEF statement in order that they will be utilized by the optimization algorithm. Optional vectors must be declared in a BTD command; they may be loaded and defined either in the same BTD command or in a subsequent MOD command.

Automatic Vector Loading

SET cards are used to specify types of automatic vector loading. In addition, bulk storage will be allocated for previously unallocated vectors encountered in a SET statement.

All SET options listed in Section 7.3.1 may be used in conjunction with the BTD command, including the initialization value option discussed in Section 7.3.2.

Because of the inter-ODS communication allowed by OPO, an ODS number option may be used in conjucntion with the BTD command. When this option is used on a SET statement, e.g.,

```
Column    1   5              1   5
          SET,vname,Dn   or  SET,vname,C
```

the values for the vector vname are obtained from the vector of the same name in ODS number n or in the control ODS. The ODS number option may not be used in the establishment of the first ODS in the unit.

The two remaining options, active and inactive, can be applied only to the name PVAR (process variables). PVAR is defined as the collection of names of all process variables. It is not a vector and it is not explicitly loaded by the user. The name PVAR is used only in the context of selection of active variables. The SET statement

```
Column    1   5
          SET,PVAR,A
```

results in all process variables being set in active status for this ODS. The statement

```
Column     1   5
               SET,PVAR,I
```

implies that all process variables are to be inactive in this ODS.

The ODS number option may also be applied to PVAR. The statement

```
Column     1   5
               SET,PVAR,Dn
```

specifies that the set of active variables in the current ODS is to be identical to that of existing ODS n.

The active-inactive status of any variable may be specified using the ALT statement. This function provides a facility for overriding automatic loading accomplished through the use of the above options.

Data Loading

The TBX and TBY cards, with their associated value cards, are used to load data from the system input device. Their use with the BTD command is identical to their use under the LUC command. Only the vector names that may be specified on the TAB cards are different.

The variable names specified on the value cards are solely for identification purposes. There is no definition implied by their appearance. The use of a name that was not defined during loading of unit common is an error.

Datum Changes

The uses of the ALT card described in connection with the LUC command are permitted with relation to the BTD command. In addition to the specification of data for elements of vectors, variables may be activated or inactivated using the ALT statement.
The statement

```
Column    1   5
          ALT,PVAR,X1,A , PVAR,Y4,I
```

implies that the variable X1 is to be active in the LP problem while Y4 is to be inactive.

Through proper combinations of the SET and ALT statements, the user may minimize the amount of data card preparation required to indicate the active variables. For example, if only a few variables are to be active such as X1, X2, Y1, and Y2, the cards

```
SET,PVAR,I
ALT,PVAR,X1,A,PVAR,X2,A,PVAR,Y1,A
ALT,PVAR,Y2,A
```

result in the proper status settings.

Example BTD Command

Figure 7-6 depicts a BTD command which utilizes the majority of the data statements.

```
BTD
DCL,XDEL,YASL
PAR,MODL,46,RLOC,/40300,RBCN,0, RPRG,22
SET,XAML,.25E-3,YAML,.25E-2,XASL,.3E-4, YFML,.1, PVAR,A,XFSL,.01,  C
   XLOB,0.0,YLOB, 0.0,YUPB,97000.0
TBX,     XSTT          XUPB
 FEED    96000.0       99500.0
 TEMP    200.0         675.0
 PRES    125.0         149.95
ENT
ALT,YUPB,PRD3,90000.0
DEF,YFML,XFSL
END
```

Figure 7-6 Example BTD Command

Several points concerning the example should be noted:

1) Only two vectors are specified in the DCL statement. The remainder of the vectors named in the command were either automatically allocated bulk storage because they are required vectors (e.g. XSTT, XUPB, YUPB) or because they were unallocated when they were encountered in another data statement (e.g YFML, XFSL).

2) The required ODS parameter MODL is loaded with the functional program number (22) of the user's MODEL.

3) The variable names in the value statements associated with the TBX statement do not appear in the same order as they were defined in the example LUC command in Figure 7-5. This is a permissible occurrence.

4) No TBY statement appears as all required and desired ODS vectors associated with dependent process variables have been loaded with data using other means.

5) The DEF statement defines only the two optional ODS vectors YFML and XFSL. The required vectors are automatically defined and their names do not have to appear in a DEF statement. All optional ODS vectors whose names appeared in data statements other than DEF have been allocated but are not defined.

6) Coordinating the example LUC command in Figure 7-5 and the above example, editing will be performed on the ODS vectors XSTT, XUPB, XASL, and XAML. There are no editing violations.

7.3.4 Modify

The MOD command is used primarily to effect changes in existing values in unit common or ODS data. It may also be used to load vectors that were declared in a LUC or BTD command but were not loaded with data at that time.

Syntax

```
Column    1
          MOD
```

All user-supplied data in a unit are subject to modification. System parameters, entire vectors, and particular elements of vectors may all be changed. The composition of the set of variables active in an ODS may also be changed.

OPO contains facilities for inspecting requested modifications to insure that the proposed values satisfy criteria specified by the user for the unit. These criteria are contained in the editing vectors which comprise part of unit common. The actions taken by OPO when a proposed modification violates an editing criterion are described in Section 9, Error Processing.

System Parameter Modifications

The PAR statement is used to change the value of a system parameter.

Vector Element Modification

The ALT statement specifying vector, variable, and new datum may be used to change a particular element in a vector.

Vector elements may also be changed using tab statements with associated value statements. Only the vector elements corresponding to a process variable name on a value card will be affected. For instance, the cards

```
Column    1      9           21
          TBX    XSTT        UXST
           X1    2.0         3.0
           X6                6.0
```

will result in 2.0 being stored in the element of XSTT associated with X1 ($XSTT_{X1}$), 3.0 being stored in $UXST_{X1}$, 6.0 being stored in $UXST_{X6}$, and 0.0 being stored in $XSTT_{X6}$. The remainder of XSTT and UXST would not be affected.

Vector Modification

An entire vector may be furnished with new values using either a SET statement or a TBX or TBY statement with associated value statements. All SET options are permissible for use in conjunction with the MOD command.

Variable Status

Variables are added or removed from the set of variables active in the model with the ALTER card. For instance, the card

```
    Column     1   5
               ALT,PVAR,X1,I,PVAR,Y2,A
```

specifies that X1 is to be removed from the active variable set and Y2 is to be added. The number of variables added does not have to equal the number removed.

Example MOD Command

Figure 7-7 is an example of a MOD command and illustrates some of the techniques of using the command. This example deals with the same vectors used in the example LUC and BTD commands in

Figures 7-5 and 7-6. It is assumed that the MOD command in the example is contained in a task directed at an ODS.

```
MOD
PAR,MODL,48
TBX,     XSTT         XDEL
 PRES    120.0        1.2
 TEMP    165.0        1.5
 FEED    95500.0      45.0
ENT
DEF,XDEL
ALT,XFSL,FEED,.02,YLOB,PRD2,32005.0
SET,YASL,D1
DEF,YASL
END
```

Figure 7-7 Example MOD Command

Some of the pertinent points in the above example are:

1) The MODEL associated with the ODS is being changed in the PAR statement.

2) The TBX statement and its value cards specify a change in all the values of XSTT.

3) Referring to Figure 7-6, it may be seen that XDEL was declared in the BTD command but not loaded with data or defined. The TBX statement and its value statements load XDEL and the succeeding DEF statement defines it. XDEL will subsequently be utilized by the optimizing algorithm.

4) There are two DEF statements in the command. This is permissible.

5) The ALT statement alters the contents of elements of XFSL and YLOB.

6) The SET statement specifies that the contents of the vector YASL in the first ODS in the unit are to be transferred to YASL in the current ODS. The first ODS must have been previously built for this to be a valid statement.

7) The second DEF statement enables YASL and makes it available for use by the optimizing algorithm.

7.4 CONTROL COMMANDS

The five commands in the control group are used for a variety of purposes. Included in the group are commands to delete a unit common or an ODS and to establish a control ODS. Two additional commands provide facilities for ordering the execution of external functional programs and for restoring a unit to a prior state.

7.4.1 ACCEPT TEST ODS Command

The ATD command establishes a test ODS as the control ODS in the appropriate unit. After execution of the command, all tasks directed to the control ODS will apply to the "accepted" ODS.

Syntax

```
Column    1

          ATD
```

The ODS that will assume control status is the one specified in the BOT command of the task containing the ATD command.

Execution of the command does not cause the destruction of any information in the unit. The specified test ODS becomes the control ODS while the superseded control ODS reverts to test ODS status. The ODS number associated with the superseded control ODS prior to its elevation to control ODS status is again in effect after its demotion.

7.4.2 DELETE TEST ODS Command

The DTD command is used to remove a test ODS from its unit. The ODS will no longer exist as far as OPO is concerned. The ODS and unit are specified in the BOT command of the containing task. The command has no parameters.

Syntax

Column	1
	DTD

7.4.3 DELETE UNIT COMMON Command

Execution of the DUC command results in the erasure of all data residing in the data files of the specified unit common. It has an effect on the unit with respect to the OPO similar to the effect of the DTD command on an ODS with respect to the unit. The command may be viewed as the "unit deactivating" command. The command has no parameters.

Syntax

```
Column    1
          DUC
```

7.4.4 TURN ON PROGRAM Command

The TOP command enables the user to turn on a specified functional program during the execution of a task.

Syntax

```
Column    1   5
          TOP,n
```

n - a decimal integer; 0<n≤99

Example

```
Column    1   5
          TOP,42
```

Parameters

Program Number - The number which identifies the functional program to the GE/PAC Monitor must be specified.

Usage

The TOP command will generally be used immediately following SOL and FIL commands. These functions may involve storing of OPO data files in areas accessible to functional programs not incorporated in the OPO. The TOP command is then used to initiate the processing of data stored in this manner.

There are no restrictions on the placement of a TOP command other than it must be contained within a task and cannot appear among the data cards which follow a LUC, BTD, or MOD command.

7.4.5 SAVE Command

The SAV command provides a fallback or fail-safe capability. Execution of the command results in a punch-out (in binary card deck form) of all information in the unit. The unit to be saved is specified on the BOT card of the containing task. The output deck is structured and formatted for reloading by the GE/PAC MONITOR Loader.

Syntax

```
Column    1
          SAV
```

The SAV command is also used to obtain a backup copy of unit information. The backup copy will be loaded by the operator to restore the unit should OPO's data files within the computer system be inadvertently destroyed. The information will be loaded into the same locations in the computer.and the random access memory device that it occupied at the time the SAV command was executed.

7.5 COMMUNICATION COMMANDS

The communications group consists of two commands: FIL and GET. The general function of these two commands is to effect the transfer of data to and from bulk memory files.

The primary use of these commands is for communication, through bulk storage, between the OPO and external functional programs. For example, if data is to be furnished by the OPO for utilization by an external functional program, the data is moved using a FIL command to a designated bulk area known to the functional program. After the move is completed, the data is available to the program. Communication in the reverse direction involves an analogous use of the GET command.

When communication between OPO and a user program is performed through bulk memory, precautions must be taken to prevent conflicts in referencing the bulk area. To protect the data in the OPO and the external functional program, each bulk area so utilized has an associated interlock status block located in core memory. The interlock block and it's use are described in Section 7.5.3.

7.5.1 FILE Command

The FIL command is used to transfer data from the OPO to bulk memory. All vectors in the system, for which the user initially supplies data, may be specified in a FIL command. Rows and columns of an LP matrix are also candidates for filing, in addition to other system-generated vectors which are named and defined in Appendix I.

Syntax

```
Column     1   5
                FIL,m,v1,n1,v2,n2,...
```

m - an octal integer

v_i - a four character vector name

or

MATX,r where r is a one to four character process variable name

n_i - a decimal integer

Example

```
Column     1   5
                FIL,/4065,XDEL,0,XELL,64,MATX,PRV1,128
```

The parameters of the FIL may be divided into two types, interlock status block address and object vector specifiers.

1) <u>Interlock Status Block Address</u> - The interlock status block address is a core memory address of a three word block of storage. The block contains items which, in addition to describing the bulk communication area, are used to protect the area from access by unauthorized programs. The interlock status block concept is discussed in detail in Section 7.5.3.

2) Object Vector Specifier - An object vector specifier identifies the vector or LP matrix row or column to be moved. There is no limit on the number of specifiers that may appear in a single FIL command. The specifier consists of the vector name and an address. This address identifies the specific destination for the vector, relative to the base location of the bulk area.

The sum of a relative address specified in a FIL command and the base location of the bulk communication area (refer to Section 7.5.3) must be a bulk memory sector origin, i.e. a multiple of 64_{10}. Failure of such a sum to be a multiple of 64_{10} results in an error condition (refer to the OPO Operations Manual for error descriptions).

The example FIL command on the preceding page specifies that the contents of the vectors XDEL and XELL (refer to Appendix A) are to be transferred to a bulk communication area. The location of the area (B) is specified in an interlock status block located at location 4065_8 in core memory. The contents of XDEL will be deposited in the communication area, starting at location B+0. The contents of XELL will be stored in the area starting at location B+64. The transmission will not be initiated as long as some other functional program is reading data from the area (refer to Section 7.5.3).

7.5.2 GET Command

The function of the GET command is inverse to that of the FIL command. GET is used to obtain data for the OPO from bulk memory communication areas. Only vectors for which the user may supply data may be specified in a GET command (refer to Appendices A and B). References to the rows and columns of an LP matrix or other system-generated information are illegal. There is no limit to the number of vectors that may be specified in a single GET command.

Syntax

Column 1 5

GET,m,v_1,n_1,v_2,n_2,...

m - an octal integer

v_i - a four character vector name

n_i - a decimal integer

Example

Column 1 5

```
GET,/4070,YLOB,128,YDEL,256
```

The parameters of the GET command are identical in form and usage to those of the FIL command.

1) Interlock Status Block Address - The interlock status block address is a core memory address of a three word block of storage. The block contains items which, in addition to describing the bulk communication area, are used to protect

the area while GET-specified data is being read from the area. Section 7.5.3 describes the interlock status block in detail.

2) Object Vector Specifier - A specifier consists of a vector name and an address. This address identifies the starting location, relative to the base location of the disk area, of the data to be obtained.

The example GET command specifies that the ODS vectors YLOB and YDEL (refer to Appendix A) are to be loaded with data from a bulk communication area. The location of the communication area is contained in the interlock status block situated in core memory at location 4070_8. If B is the base location of the area, sufficient data to fully load YLOB will be fetched from the area starting at location $B+128_{10}$. Location $B+256_{10}$ is the base location of the data to be stored in YDEL. The transmission of data from the communication area to OPO data files will not take place as long as some other functional program is writing in the area (refer to Section 7.5.3).

7.5.3 Interlock Status Block Concept

To insure data protection to the OPO and external functional programs when FIL and/or GET commands are used to communicate between the two, each bulk area utilized in this manner has an associated interlock status block. The interlock status must be set and interrogated by both participants in the communication.

The OPO is not responsible for the allocation of either the interlock block or the bulk communication area. The user must provide for the existence of all desired interlock blocks in core as well as for the areas in bulk memory. In addition, the user is responsible for initialization of the interlock block items.

There are five logical items defined in an interlock block as follows:

1) Bulk Communication Area - The bulk communication area is specified by its base location, its length, and the number of the appropriate bulk controller. These three values are used to determine absolute bulk memory addresses from the relative addresses associated with each vector in the FIL and GET commands and to guard against overflow of the area.

2) Write Status - There are two flags associated with Write Status; the Write Active flag and the Write Waiting flag (off-on flags). Write Active is set when a write operation on the communication area is currently in progress. Write Waiting is set when a write operation is pending, either because a prior write operation is active or because reading operations are currently proceeding.

3) Number of Readers - This item is a counter which is incremented by one just prior to initiation of a reading operation and decremented by one when a read operation is completed.

4) Unit Number - The Unit Number is used to restrict the use of the associated communication area to one unit. This restriction provides protection against overlaying of data obtained from two or more units.

5) Control ODS Only Write - This off-on flag is intended to protect data emanating from a control ODS from being destroyed by test ODS tasks. If the flag is on, a test ODS task may not write in the associated communication area, but may read from the area. Control ODS's and external functional programs may read and write in the area. The physical structure of an interlock status block is given in Appendix J.

Write Status and Number of Readers are used to prevent merging of logical reads and writes which reference the same communication area. Protection is necessary since a logical I/O operation may consist of several physical operations. For example, each vector specified in a FIL command implies at least one physical write operation. Therefore, without protection, an external functional program could attempt to read data not FILed.

The flowcharts in Figures 7-8 and 7-9 depict the actions that must be performed by reading and writing external functional programs in order to properly utilize the interlock items, Write Status and Number of Readers.

The user has the responsibility for synchronizing the OPO and an external functional program. He must insure that the OPO does not GET data from a disk area before the program has placed it there. Similarly, an external functional program must not attempt to utilize data before it has been FILed by the OPO. One method of synchronization involves the TURN ON PROGRAM command (TOP, described in Section 7.4.4. For example, the command sequence

```
Column    1   5
          FIL,/5036,XSTT,0
          TOP,16
```

where 16 represents the name of the utilizing external real-time functional program, insures a proper data transfer of the contents of XSTT from OPO to 16. One method of transferring data from 16 to OPO is for 16 to transfer the data to the communication area and then present an internal task containing the appropriate GET command to the OPO (refer to Section 8.3 and Appendix D).

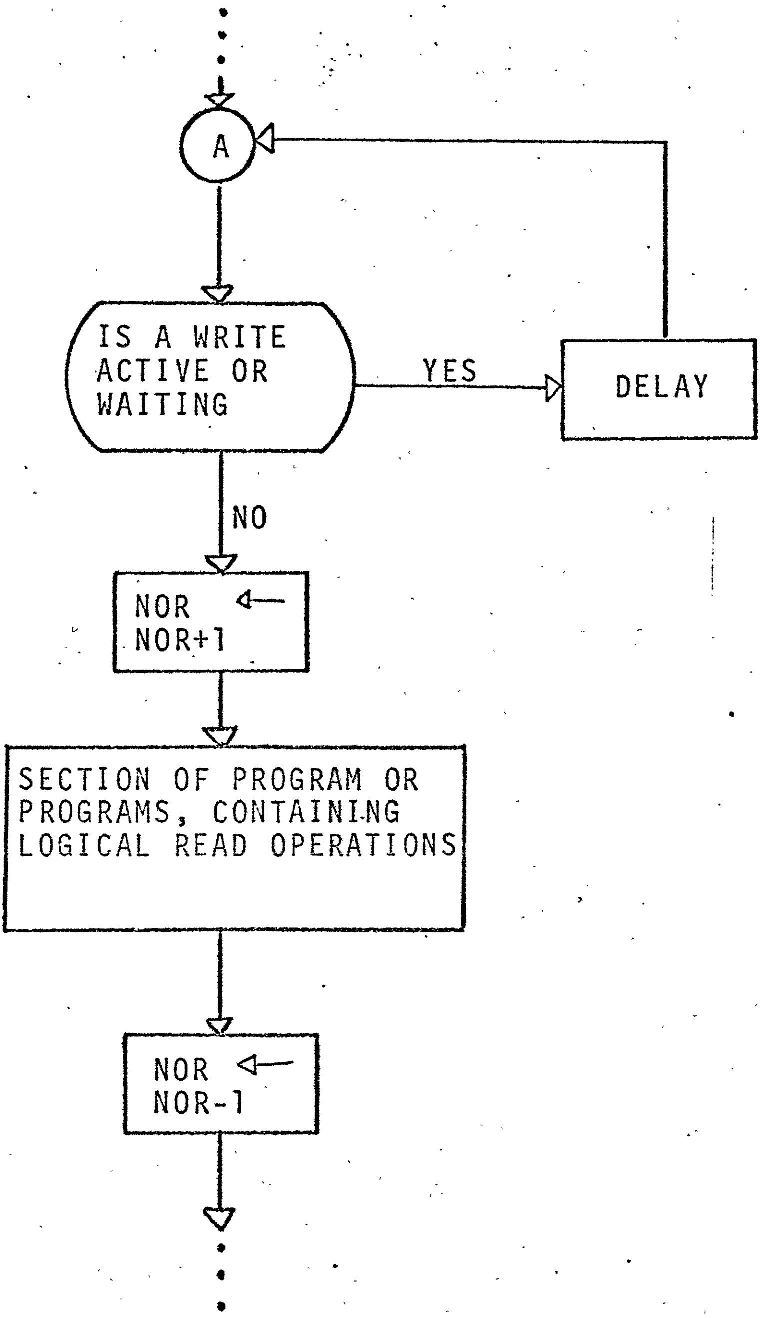

A
IS A WRITE ACTIVE OR WAITING
YES
DELAY
NO
NOR ← NOR+1
SECTION OF PROGRAM OR PROGRAMS, CONTAINING LOGICAL READ OPERATIONS
NOR ← NOR-1

Figure 7-8 Reading Program Interlock Code

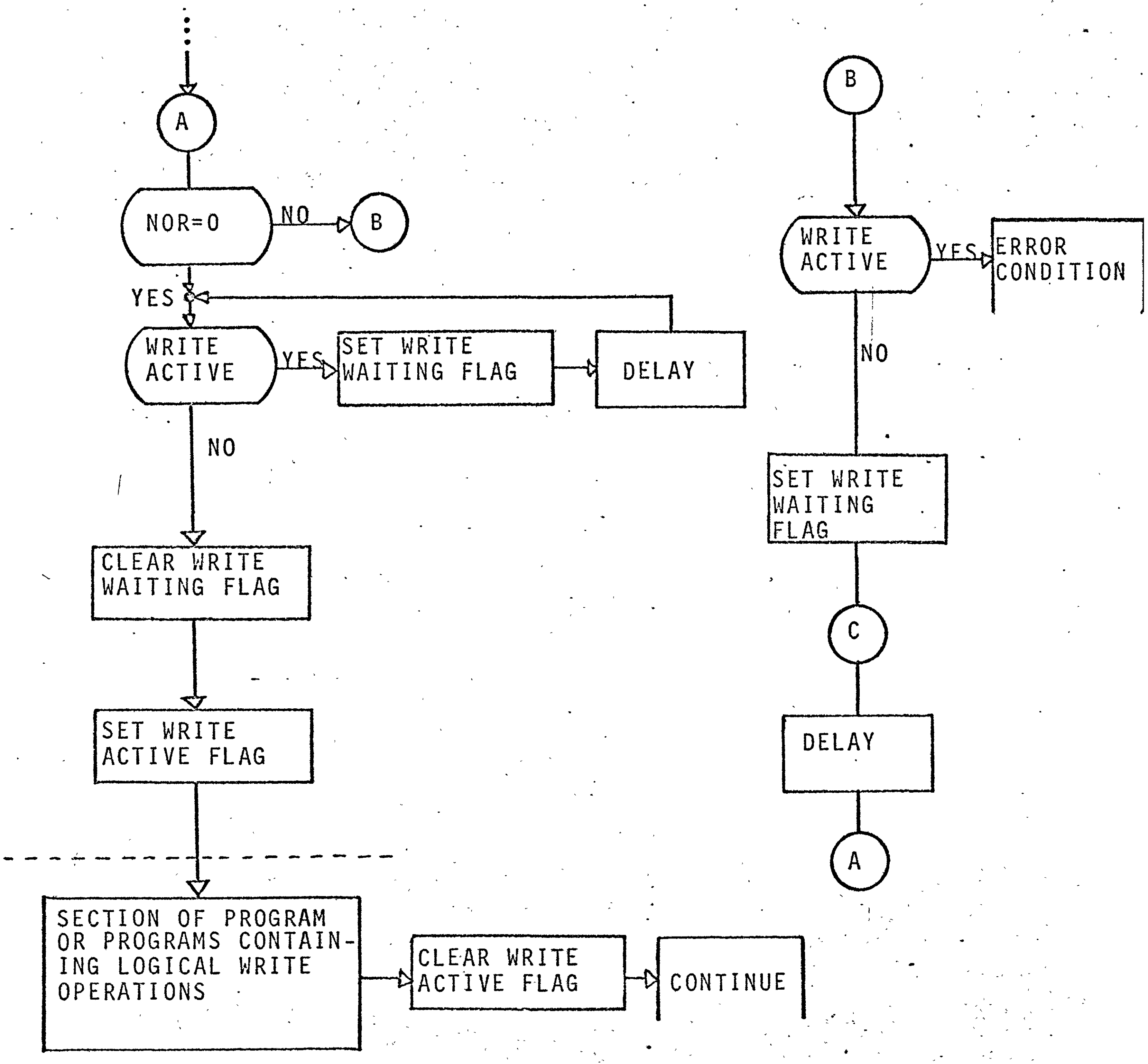


Figure 7-9 Writing Program Interlock Code

7.6 SOLVE COMMAND

7.6.1 Function

The SOL command causes OPO to determine a solution for the optimization problem. This solution is a move from the plant position or starting position to a new recommended optimum position.

Syntax

```
Column      1   5
            SOL,n
```

n - 1 or 2

Example

```
Column      1   5
            SOL,2
```

7.6.2 Parameters

The SOL command has one parameter. This parameter specifies the type of starting basis to be used in the first step of the solution.

If the parameter is 1, a scratch (slack) basis will be used. If the parameter is 2, a basis selected by the user will be used as the starting basis. The user selects a starting basis by specifying a starting position for the move.

This option applies only to the first step of the move.

7.6.3 Consequences

A MOVE is made resulting in the generation of a new recommended position for the process unit. During the MOVE, reports on the progress of the solution will be printed as specified in the reporting specifications set up by the user in the ODS. These reports include iteration reports, subset selection reports, step reports, and move reports. The user may specify, in the ODS parameters (refer to Appendices A and F), the detail to be included in these reports and the frequency at which the reports should be published.

7.7 OUTPUT COMMANDS AND FORMATS

The OPO system produces two general classes of printed output. The first type of output is generated by the solution program in response to controls set by the user in the ODS. The second type of output is generated in response to specific commands which are part of a task. Output formats are described in Appendix F.

7.7.1 Solution Generated Output

Five reports are generated during the solution process; the iteration log, subset selection log, step log, the solution report, and the reduced cost report.

Iteration Log - The iteration log is available for each LP iteration made during the LP solution in a single step. The report is generated, or not, according to the setting of control parameters in the ODS. These parameters can be set or modified by the user.

The iteration log reports the following data:

1) The number of iterations in the move
2) Variable entering basis
3) Variable leaving basis
4) Value of pivot element
5) New value of the objective function
6) Cause of iteration
7). Unit and ODS identification

Subset Select Log - The subset select log is available for each subset select cycle taken. The report is generated or not according to the setting of parameters in the ODS.

The subset select log reports the following data:

1) The number of subset select cycles in step
2) The number of exchanges made in cycle
3) The number of pivot steps in the transformation
4) The objective function value
5) The number of iterations in the move
6) The number of unacceptable pivots rejected
7) Unit and ODS identification

Step Logs - The step log is available for each step in a move. The log is generated or not according to parameters in the ODS.

The step log reports the following data:

1) The number of step completed
2) Objective function value
3) The number of iterations in move
4) The number of subset select cycles
5) The number of unacceptable pivots rejected
6) Unit and ODS identification

Solution Report - The solution report can be output after each step of a move, after each n^th step of a move, and/or at the end of a move. If any solution reports are requested, a report will be made after the last step of the move regardless of reporting interval. Solution report controls are set in the ODS by the user.

The solution report includes the following data for the whole step:

1) Unit and ODS identifiers
2) Step number
3) Time of day
4) Number of steps in move
5) Objective function value
6) The number of iterations in move
7) The number of infeasibilities
8) The number of variables outside the target range
9) Objective function improvement

For each variable the solution report includes the following:

1) Variable name
2) Variable value at optimum
3) Variable status
4) Most restrictive bound values
5) Target values

The variable status information includes whether the variable was in the optimization, whether it is basic, and its status in relation to each of the bounds. Status is reported with respect to outer limits, move limits, step limits, and target values.

The report includes the values of the most restrictive upper and lower bounds on the variable and type (outer limit, move limit, or step limit) of each such bound.

Reduced Cost Report - The reduced cost report is produced after each step or each n^{th} step. If any reduced cost reports are produced, a report is made after the last step of a move regardless of the reporting interval.

Reduced cost report controls are set in the ODS by the user and are independent of the solution report controls.

For each variable the reduced cost report includes:

1). Variable name
2) Variable status
3) D_j for each non-basic variable
4) The effective costs

The reduced cost (D_j) is the net cost of changing the variable value in a positive direction.

The effective costs are the costs used in computing the value of the reduced cost. These costs include any move penalties or target penalties which may be active in the reduced cost computation.

The above reports and the means of selecting and controlling their generation is explained more fully in Appendix F.

7.7.2 Command Generated Output

Output can be generated on the printer in response to commands included in a task. These commands are the MAT, PRT, and DMP commands.

MATRIX Command - The MAT command in a task causes the contents of the LP matrix, in the ODS to which the task is directed, to be output to the printer. This matrix is generated in the solution process. Any time a MAT command is executed, the matrix printed is the one used in the last step made in finding the optimal solution. The printout is in a standardized format (see Appendix F).

Syntax

```
Column    1
          MAT
```

Print Command - The PRT command can be used to print any vector, or vectors, in unit common, in an ODS, or in the LP matrix in an ODS.

Syntax

```
Column      1   5
            PRT,v1,v2,v3,...
```

v_i - a four character vector name

or

MATX,r where r is a one to four character variable name

Example

```
Column      1   5
            PRT,XUPB,XELL,YELL,XSTT,MATX,XVR1
```

The names appearing in the command are the names of the vectors to be printed.

Each vector is printed as a column on the page. If all columns named in the command cannot be printed across a single page, then multiple pages can be used as needed. Appendix F shows the format of output generated by the PRT command.

Each column is headed by the vector name. The vectors which may be printed include all of the model vectors to which the user can input data (refer to Appendices A and B), all working vectors used in the solution (refer to Appendix C), and all rows and columns of the LP matrix.

Dump Command - The DMP command is intended to be used for debugging purposes. It allows the user to obtain hard copy of the contents of an ODS or unit. The hard copy is produced on a high-speed printer.

Syntax

<u>Column</u> 1 5

```
          DMP,n1,n2,n3,...
```

n_i - ODS data set indicators (C or Dn, $0<n\leq9$; refer to Section 7.1)

The command may be used in a unit task (one whose BOT card specifies unit common) or an ODS task (when the BOT card specifies an ODS).

Parameters

The command has no parameters when used in a task directed at an ODS. The contents of unit common and the ODS that is the object of the task will be dumped.

When used in a unit task, DMP may have a list of ODS numbers as parameters. These ODS numbers identify the ODS's to be dumped. Unit common will also be dumped.

If there are no ODS numbers appended to the command in a unit task, *all* ODS's in the unit are dumped in addition to unit common.

Output

All systems parameters and vectors in unit common and specified ODS's will be printed, including system-generated information which was not originally furnished by the user.

A list of unallocated vectors will be produced with an identifying title. Names and contents of declared, but undefined, vectors will be printed, accompanied by an indication of their undefined state.

Format

The contents of unit common and any specified, or implied, ODS's will be printed separately. The unit common title line will contain the identifying number of the involved unit. The contents of an ODS will be preceded by a title identifying the ODS.

The formats for unit common contents and ODS contents are quite similar. System parameter values with their names will be printed immediately after the title lines followed by the vector information printed in a format similar to that used for TAB input. A line of vector names will be printed followed by the vector values aligned with the names. Process variable names will appear in the left column to identify the vector elements. The format of the printout is illustrated in Appendix F.

SECTION 8 - TASK MANAGEMENT FUNCTIONS

The task management functions allow both the operator and internal functional programs to present and manipulate the tasks directed to the OPO. The operator can also initiate the execution of functions to check the format of his input card deck or to give a summary report of the present status of the OPO System.

There are seven task management functions as follows:

1) GET TASK FROM CARD READER
2) DIAGNOSTIC SCAN
3) STATUS
4) MOVE UP TASK
5) DELETE TASK
6) ACCEPT TASK
7) CONTROL TASK INTERRUPT

The first five task management functions listed (1-5) are directly available to the operator. The last four functions listed (4-7) are available to internal functional programs. A brief description of these functions is given below. For detailed information on the operating procedure of those functions available to the operator, the user is directed to Section 2 of the *OPO Operations Manual.*

8.1 FUNCTIONS AVAILABLE TO THE OPERATOR

The five functions available to the operator are accessed through the Operator Linkage Program (OLP01-YPC1B) which operates within the GE/PAC MONITOR I/O Typer OPR System. All communications occur via the I/O Typer. OLP01, acting for OPO, will ask the operator which of the five task management functions is desired. (Refer to Section 8.1.6 for an example.)

8.1.1 GET TASK FROM CARD READER

This function is the required manner in which the operator requests OPO to accept a new job (i.e., one or more tasks) from the card reader. The BEGIN TASK (BOT) card is read and parameters extracted. OLP01, as a functional program, uses the ACCEPT TASK function (refer to Section 8.2.1) to enter the card task into the OPO System.

8.1.2 DIAGNOSTIC SCAN

This function will read the cards of the first task presently residing in the input hopper of the card reader. The format of each card will be checked. When a syntax error is found, a message on the printer will reproduce the card and identify the column where the error was noted. The task is not executed and no editing of values occurs. Since a syntax error will terminate the execution of a card task*, it is desirable to have OPO run a diagnostic scan on a task before a request for its acceptance is made.

*Card tasks will be aborted on a syntax error and the remaining cards automatically given a diagnostic scan because they are essentially free-time applications. Processing the errors found on these tasks may unduly tie-up the OPO and prevent it from achieving its main objective - the on-line, real-time, optimization of the process.

8.1.3 STATUS

A report of the current condition of the task queues and unit descriptors are given on the I/O Typer. This function may be used as a system debugging aid, to determine the progress of particular tasks, and to determine current state and structure of OPO system bulk storage. As the report conforms to present conditions of the system, it may assume many forms. Appendix G shows a representative format of the STATUS report.

8.1.4 MOVE UP TASK

The purpose of this function is to move a previously accepted task to the head of its queue, thereby causing its execution to precede the execution of other tasks in that queue. The operator is required to furnish the task's identifier and the task's unit number. These parameters are called separately by OLP01. If the task is not found in the queue, the operator is notified.

8.1.5 DELETE TASK

This function will remove a task from the system. OLP01 calls for the task's identifier and unit number. The task's queue is searched for the particular task. When located, it is removed, thereby canceling its execution. If it is not located or is being executed, the operator is notified.

8.1.6 Example of Conversation between Operator and OPO

An example of the conversation between the operator and OPO over the I/O typer is given below. The user is referred to the OPO Operations Manual for full operating instructions.

To illustrate, the operator wishes to move the previously accepted task X4 to the head of unit 3's task queue. The items entered by the operator are underlined. After the operator depresses the input button on the I/O typer, the following conversation occurs:

```
READY OPOS;
CARD TASK-1, MOVEUP-2, DELETE-3, DIAG SCAN-4, STATUS-5 2;
UNIT 3;  TASK X4;
END JOB
```

8.2 FUNCTIONS AVAILABLE TO THE INTERNAL FUNCTIONAL PROGRAM

Four task management functions are available to any functional program in the GE/PAC MONITOR System. These functions and their required parameters are described below. The calling sequences are also shown. It should be noted that sufficient space has been allocated in the parameter communications words so that the two characters identifying the task may be in either Common Peripheral or ASCII code. The user must use the code with which his GE/PAC MONITOR normally operates.

8.2.1 ACCEPT TASK

This function is equivalent to the operator's GET TASK FROM CARD READER function (8.1.1). This function requests that OPO accept a new task located in bulk storage.

Accept Task Parameters

Parameters for ACCEPT TASK are loaded into permanent core location OPOQSR.

OPOQSR	BULK ADDR		
OPOQSR+1		17 15 C	
OPOQSR+2	00	0	00000

where BULK ADDR is the bulk address of the new task's Task Status Word (refer to Appendix E) and C is the associated bulk device controller number.

NOTE

(OPOQSR+2)=0 identifies the ACCEPT TASK function.

8.2.2 MOVE UP TASK

The overall effect of this function is exactly the same as the operator's version (8.1.4).

Move Up Task Parameters

Parameters for MOVE UP TASK, which are loaded into OPOQSR, are required in the following format.

OPOQSR		7 0 UNIT#
OPOQSR+1	23 8 TASK ID	
OPOQSR+2	00000	001

where the TASK ID is the task's identifier (two characters, left-justified in the field), the UNIT # is the task's unit number.

NOTE

(OPOQSR+2) = 1 identifies the
MOVE UP TASK function.

8.2.3 DELETE TASK

Again, this function, in effect, is equivalent to the operator's version (8.1.5).

Delete Task Parameters

As before, these parameters are loaded into OPOQSR.

OPOQSR		7 0 UNIT#
OPOQSR+1	23 8 TASK ID	
	00000	003

where the TASK ID is the task's identifier (two characters left-justified in the field), the UNIT# is the task's unit number.

NOTE

(OPOQSR+2) = 3 identifies the DELETE TASK function.

8.2.4 CONTROL TASK INTERRUPT

This function is a composite of ACCEPT TASK, MOVE UP TASK and an interrupt of the test task (i.e. a task not on the control ODS), if one is in progress. It is used to prevent delays on control tasks generated by internal functional programs by interrupting the execution of a test task.

The composition is as follows:

1) A request to accept a new task which must be directed at the current control ODS.

2) At the same time, this new task is to go directly to the head of its task queue.

3) If a test task is in progress, it will be interrupted at the conclusion of the next command and the new control task will be the next task executed when OPO services this queue.

4) If a control task is in progress, the new control task will not interrupt but will merely be moved to the second position in the task queue and be executed once the current control task is completed.

Control Task Interrupt Parameters

The parameters required for this function are quite similar to those of ACCEPT TASK:

OPOQSR	BULK ADDR		
OPOQSR+1		17 15 C	
OPOQSR+2	00	0	00002

where BULK ADDR is the bulk address of the new control task's Task Status Word (refer to Appendix E) and C is the associated bulk device controller number.

NOTE

(OPOQSR+2) = 2 identifies the CONTROL TASK INTERRUPT function.

8.3 CALLING PROCEDURE

To use any of these four functions, the internal functional program must perform three distinct acts.

1) AVAILABILITY TEST

2) PARAMETER LOAD

3) TURN-ON FUNCTION

8.3.1 AVAILABILITY TEST

The functional program must determine if the OPO Task Management Functions are free to honor a request. The functions are available for use whenever the contents of permanent core location

OPOQSR are zero. OPOQSR is non-zero whenever one of the functions is operating. The test should be made with interrupts inhibited to insure proper results.

8.3.2 PARAMETER LOAD

After locating the functions available for use, the parameters, as specified in Sections 8.2.1, 8.2.2, 8.2.3, and 8.2.4 are loaded into OPOQSR.

8.3.3 TURN-ON FUNCTION

Once parameters are loaded, the functional program can turn on the functional program (using the Monitor subroutine TPNC01) which operates the functions. This program (Queue Service Request - YPC1C) has a Monitor program priority number equal to OPOMPN.

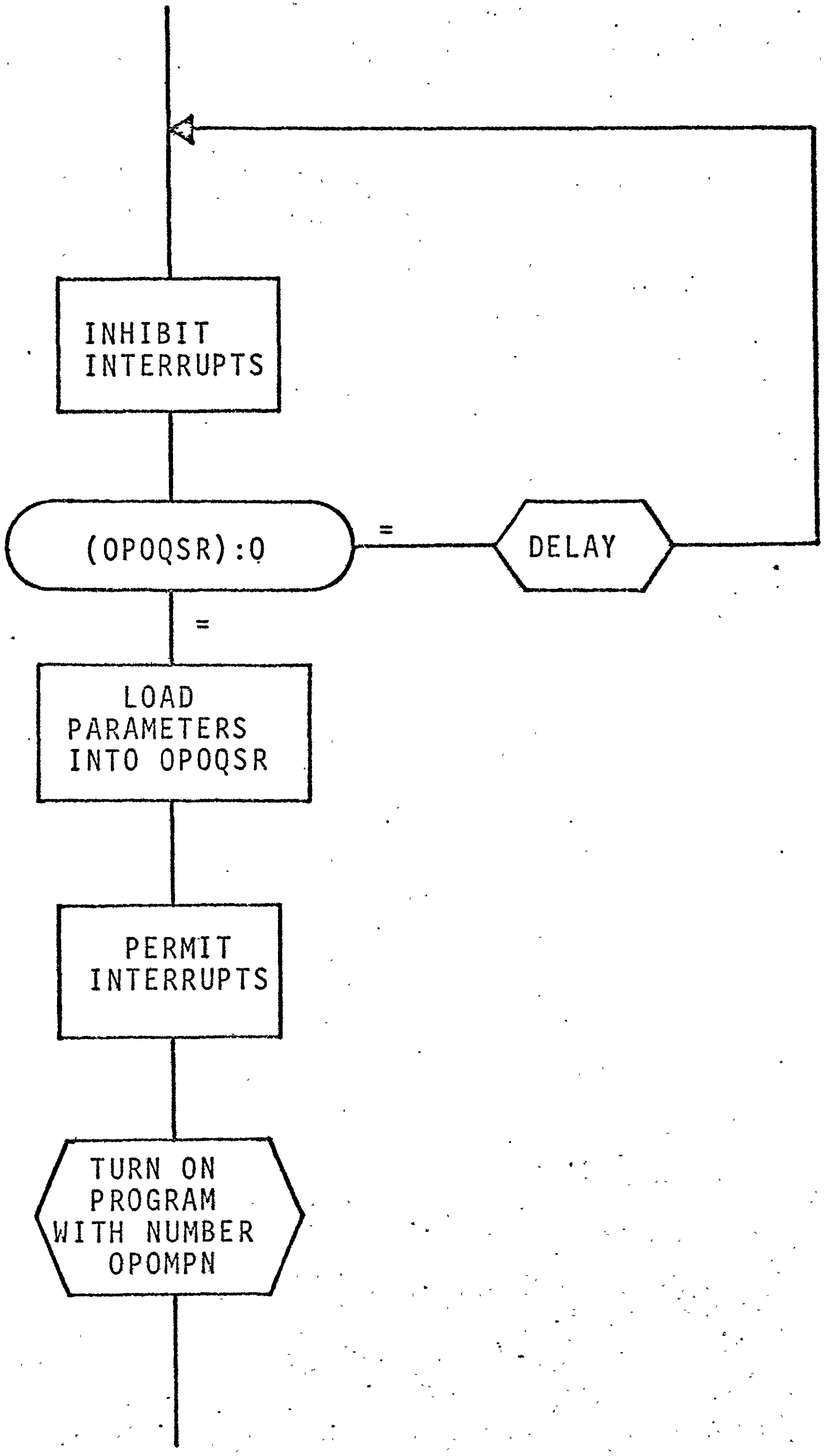

INHIBIT INTERRUPTS
(OPOQSR):0
=
DELAY
=
LOAD PARAMETERS INTO OPOQSR
PERMIT INTERRUPTS
TURN ON PROGRAM WITH NUMBER OPOMPN

Figure 8-1 Queue Service Interlock Flow

8.3.4

In PAL coding statements, the three steps may be performed as follows:

```
        IAI
        LDA OPOQSR              IS FUNCTION F AVAILABLE
        TZE
        BTS *+4                 YES
        SPB DELCO1              NO, DELAY AND TRY AGAIN
        DEL 1, SECND/2
        BRU *-6
        LDA UNITNO              PICK UP
        LDQ TASKID               PARAMETERS
        DST OPOQSR                AND
        LDK F                      LOAD
        STA OPOQSR+2                THEM IN
        PAI
        LDA ZERO
        SPB TPNCO1              TURN-ON FUNCTION
        CON G,OPOMPN

TASKID CON A,2,XL               FOR BOTH COMMON PERIPHERAL
*                               CODE AND ASCII SYSTEMS
UNITNO CON D,3                  TO INDICATE UNIT 3
```

SECTION 9 - ERROR PROCESSING

The basic concept of OPO error processing is to allow the operator to respond to an error in order to specify some corrective action. Allowing the operator to communicate in this manner results in better through-put times for a given task than could be achieved without such communication.

The errors, which are listed in the OPO Operations Manual, are classified according to the subset of responses available to the operator. Table 9-1 gives the whole collection of responses available to the operator. The numbers are the assigned response codes.

Table 9-1 Available Operator Responses

1.	HOLD
2.	ABORT TASK
3.	SKIP
4.	IGNORE
5.	ACCEPT VALUE
6.	ACCEPT RESTRICTION
7.	RESET RESTRICTION
8.	SUBSTITUTE NEW VALUE
9.	STANDARD

There are six error classes each defined by the permissible responses the operator may use. Table 9-2 lists these six error classes and their respective subset of responses. An expanded discussion of these error classes appears in Section 4 of the OPO Operations Manual.

Table 9-2 Permissible Responses for Error Classes

RESPONSE	CLASS					
	1	2	3	4	5	6
HOLD (1)	✓	✓	✓	✓	✓	
ABORT TASK (2)	✓	✓	✓	✓	✓	✓
SKIP (3)	✓	✓				
IGNORE (4)		✓	✓	✓		
ACCEPT VALUE (5)			✓			
ACCEPT RESTRICTION (6)			✓			
RESET RESTRICTION (7)			✓			
SUBSTITUTE NEW VALUE (8)	✓	✓	✓	✓	✓	
STANDARD (9)	✓	✓	✓	✓		

9.1 AVAILABLE OPERATOR RESPONSES

9.1.1 HOLD (1)

This response requests a hold for a defined period of time while the operator determines a course of action. After three consecutive holds by the operator, OPO will assume the standard response (9).

9.1.2 ABORT TASK (2)

Execution of the current task wherein the error occurs is terminated and the task is deleted. If the task is a card task, the remainder of the task will be passed through the card reader. Since special actions may occur as a result of this response, the user is directed to the particular error as listed in the OPO Operations Manual, Section 4.3.

9.1.3 SKIP (3)

Depending on the error, an entire command or one component of a command will be skipped. Processing will continue with the next command or command component. The action taken in a particular case can be determined by referring to Section 4.3 of the OPO Operations Manual.

9.1.4 IGNORE (4)

Again, actual action of this response depends greatly on the particular error in mind. Section 4.3 of the OPO Operations Manual describes the action according to the error. In general, a violating value will be ignored and the original value retained, a vector name may be skipped, or default value incorporated.

9.1.5 ACCEPT VALUE (5)

A violating value is to be accepted in spite of its violation of an editing restriction.

9.1.6 ACCEPT RESTRICTION (6)

The violating value will be ignored and the violated restriction will be accepted in its stead.

9.1.7 RESET RESTRICTION (7)

The violating value will replace the violated editing restriction. This voids the violation.

9.1.8 SUBSTITUTE NEW VALUE (8)

A new value, vector name, variable name, keyword, etc., supplied by the operator, will replace the violating name or value.

9.1.9 STANDARD (9)

The standard response is preset by the user to be synonymous with one of the other responses. The responses HOLD, SUBSTITUTE VALUE and STANDARD may not be the standard for any error class.

The standard response is selected by the user by setting the unit common system parameters STD1, STD2, STD3, and STD4 for error classes 1-4. A standard response cannot be selected for classes 5 and 6.

9.2 OPERATOR NOTIFICATION

When an error is detected, OPO Informs the operator via the I/O typer that the error has occurred and identifies the error by giving the error's corresponding error code. The error code is a three digit number where the first digit identifies the error's class and the other two digits serve to identify the error in that error class.

A complete list of all OPO error conditions and codes is contained in the OPO Operations Manual.

9.2.1 Error Message Components

Each error message will have at least one line of I/O typer output. This line is called the leading line and always assumes the following form:

```
OPO ERROR CODE XYZ UNIT K ODS N TASK AB
```

where XYZ is the error code (X is the error class),

K is the unit number of the unit where the error was detected,

N is the number of the Data Set (D) (U for Unit common and C for Control ODS) upon which the error was detected,

and AB is the identifier of the task being executed when the error was detected.

The leading line may be followed by a group of names or values, at most three to the line, which are pertinent to error processing. Positional meaning to these values will be given and their actual meaning will be found listed, by error code, in Section 4.3 of the OPO Operations Manual.

9.3 OPERATOR RESPONSE

If the error permits an operator response, the word RESPONSE will follow the error notification.

9.3.1 Example

```
OPO ERROR CODE 402 UNIT 2 ODS 3 TASK L2
 LUC   XXPR  1.0E3
RESPONSE
```

The operator will respond with the code of the chosen response. If the response is Substitute Value, OPO will call for the new name (or value, etc.) by typing NEW VALUE. The operator now enters his new value or name. If the Standard response is taken, OPO will inform the operator of the action taken by typing ACTION J, where J is the code corresponding to that action.

```
OPO ERROR CODE 402 UNIT 2 ODS 3 TASK L2
LUC  XXPR 1.0E3
RESPONSE 9; ACTION 4;
```

For full explanations and examples, the user is referred to Section 4 of the OPO Operations Manual.

APPENDICES

APPENDIX A

OPTIMIZER DATA SET

APPENDIX A

OPTIMIZER DATA SET

The following is a list of parameters and vectors which comprise an optimizer data set.

Certain parameters and vectors are required and must be loaded by the user before a solution can be generated.

Some parameters are optional because the system assumes a value if none is input. For these parameters the definition is followed by the default value.

Required Data

Parameters

MODL - the program number of the user-supplied MODEL to be utilized during execution of the OPO optimizing algorithm.

Vectors

XSTT - Starting values of the independent process variables (X).

XUPB - Upper bounds on the values of X.

YUPB - Upper bounds on the values of the dependent and implicit process variables (Y).

XLOB - Lower bounds on the values of X.

YLOB - Lower bounds on the values of Y.

XAML - Absolute move limits on X.

YAML - Absolute move limits on Y.

XASL - Absolute step limits on X.

Optional Data

Parameters

Non-default parameters are:

RLOC - The location of a bulk memory area where results are to be stored by the optimizing algorithm for use by a user's functional program.

RBCN - The number of the controller in charge of the bulk memory device used in connection with RLOC.

RPRG - The program number of the user's functional program to be turned on at the completion of an execution of the optimizing algorithm in order to utilize the results previously stored (refer to RLOC above).

XMPN - The penalty applied to X to prevent small moves (refer to DZRO below) near the optimum.

YMPN - The penalty applied to Y to prevent small moves (refer to DZRO below) near the optimum.

Parameters with default values are:

MXIT - Maximum number of iterations to be made by the optimizing algorithm during one step. Default value is 1000.

MXSP - Maximum number of steps to be made during one move. Default value is 500.

MXIL - Maximum number of iterations to be made without printing the iteration log. Default value is 10.

MXSS - Maximum number of subset select cycles to be made without printing a subset select log. Default value is 10.

MXSL - Maximum number of steps to be taken during a move without printing a step log. Default value is 10.

MXSO - Maximum number of steps to be taken in a move without printing a solution report. Default value is 10.

MXRC - Maximum number of steps to be taken in a move without printing a reduced cost report. Default value is 10.

RPRT - Specifies optional formats of the iteration log. Default value is 31.

PMIN - Minimum acceptable pivot value. Default value is .00001.

DZON - Tolerable amount of constraint violation by any element of Y. .Default value is .0001.

NOTE

DZON will be utilized by the optimizing algorithm only if the optional vector YDZN is not furnished by the user.

AZRO - Coefficients in the LP matrix whose magnitude is less than AZRO will be considered to be zero. Default value is .00001.

XEMX - Estimated maximum number of independent process variables active in the optimization at any given time. Default value is NX.

YEMX Estimated maximum number of dependent process variables active in the optimization at any given time. Default value is NY.

DZRO Coefficients in the reduced cost equation whose magnitude is less than DZRO will be considered to be zero. Default value is .00001.

NOTE

DZRO will be utilized by the optimizing algorithm only if either of the optional parameters XMPN and YMPN is not furnished by the user.

KIMP} DOBJ A move will be terminated if KIMP consecutive steps produce less than DOBJ improvement in the value of the objective function. Default values of KIMP and DOBJ are 5 and .001, respectively.

Vectors

XUTR - Upper target limit on X.

YUTR - Upper target limit on Y.

XLTR - Lower target limit on X.

YLTR - Lower target limit on Y.

XTPN - Penalty applied when X violates its target range.

YTPN - Penalty applied when Y violates its target range.

XFML - Fractional move limit on X.

YFML - Fractional move limit on Y.

YDZN - Limit on allowable constraint violation by each Y. (refer to the optional parameter DZON).

XDEL - Incremental changes in X used in the calculation of partial derivatives and adaptive linearity limits.

NOTE

XDEL is optional if a type 1 or type 2 MODEL is being used, otherwise, it is required.

YDEL - Constants used to compute adaptive linearity limits.

NOTE

The furnishing of data for this vector by the user will result in the calculation of adaptive linearity limits when a type 0 MODEL is being used; the vector will be ignored when a type 1 or type 2 MODEL is used.

XFSL - Fractional step limit on X.

YFSL - Fractional step limit on Y.

YASL - Absolute step limit on Y.

APPENDIX B

UNIT COMMON DATA SET COMPONENTS

APPENDIX B

UNIT COMMON DATA SET COMPONENTS

Required Data

Parameters

The following four parameters are part of the LUC command and have no system-defined names.

1) The number of independent process variables in the process unit.

2) The number of dependent process variables in the process unit.

3) The number of implicit process variables in the process unit.

4) Estimated amount of bulk storage space to be reserved for the process unit.

Optional Data

Parameters

Four unit common optional parameters have default values. These are the standard operator responses to errors in error classes 1 through 4 (refer to Section 9, Error Processing). The parameter values are specified by the user using PAR data statements in conjunction with an LUC command.

STD1 - Standard response for error class 1. Default value is 3.

STD2 - Standard response for error class 2. Default value is 4.

STD3 - Standard response for error class 3. Default value is 4.

STD4 - Standard response for error class 4. Default value is 4.

The remainder of the parameters do not have default values and do not enter into OPO operations unless the user furnishes data for them in the data statements of an LUC or MOD command.

UXMT - Upper editing restriction on MXIT.

UXMS - Upper editing restriction on MXSP.

UPMN - Upper editing restriction on PMIN.

LPMN - Lower editing restriction on PMIN.

UDZN - Upper editing restriction on DZON.

LDZN - Lower editing restriction on DZON.

UAZR - Upper editing restriction on AZRO.

LAZR - Lower editing restriction on AZRO.

UDZR - Upper editing restriction on DZRO.

LDZR - Lower editing restriction on DZRO.

UXMP - Upper editing restriction on XMPN.

LXMP - Lower editing restriction on XMPN.

UYMP - Upper editing restriction on YMPN.

LYMP - Lower editing restriction on YMPN.

Vectors

The vectors listed below are all editing vectors and contain upper and lower restrictions on ODS data. The left-hand column contains the names of the unit common editing vectors; the right-hand column contains the names of the restricted ODS vector corresponding to the editing vectors. The ODS vectors are defined in Appendix A.

Editing Vector Names	Restricted Vector
UXUB LXUB	XUPB
UYUB LYUB	YUPB
UXLB LXLB	XLOB
UYLB LYLB	YLOB
UXUT LXUT	XUTR
UYUT LYUT	YUTR
UXLT LXLT	XLTR
UYLT LYLT	YLTR
UXFM LXFM	XFML
UYFM LYFM	YFML
UXAM LXAM	XAML
UYAM LYAM	YAML
UXST LXST	XSTT

UXDL LXDL	}	XDEL
UYDL LYDL	}	YDEL
UXFS LXFS	}	XFSL
UYFS LYFS	}	YFSL
UXAS LXAS	}	XASL
UYAS LYAS	}	YASL
UXTP LXTP	}	XTPN
UYTP LYTP	}	YTPN
UXMP LXMP	}	XMPN
UYMP LYMP	}	YMPN
UYDZ LYDZ	}	YDZN

APPENDIX C

WORKING VECTORS

APPENDIX C

WORKING VECTORS

The following is a list of vectors generated in the optimizer data set by the solution program. The user cannot input data to these vectors, but he can use their contents. The user can obtain the contents of these vectors by naming them in FIL or PRT commands.

YSTT - Starting value of dependent variables as computed by MODEL, based on XSTT.

XDVL - (XINT-XELL) at start of move.

XHUB - Hard upper bound on X; minimum (XUPB, XINT + move limit).

XHLB - Hard lower bound on X; maximum (XLOB, XINT - move limit).

XHUB - Hard upper bound on Y; minimum (YUPB, YINT + move limit).

XHLB - Hard lower bound on X; maximum (YLOB, YINT - move limit).

XINT - Initial value of X.

XINT - Initial value of Y.

XELL - Effective lower limit on X.

XEUL - Effective upper limit on X.

YELL - Effective lower limit on Y.

YEUL - Effective upper limit on Y.

XSTA - Status of X.

YSTA - Status of Y.

XESL - Effective step limit on X.

YESL - Effective step limit on Y.

XRDC - Reduced cost of X.

YRDC - Reduced cost of Y.

NOTE

The reduced costs of X and Y are non-zero only for the non-basic variables.

APPENDIX D

EXTERNAL COMMAND FORMATS

APPENDIX D

EXTERNAL COMMAND FORMATS

1.0 BEGIN TASK Command (Section 7.2.1)

1.1 Syntax

```
Column      1   5
            BOT,n1,n2,n3,n4
```

n_1 - task identifier

n_2 - task priority

n_3 - unit number

n_4 - data set indicator

(Refer to Section 7.1)

1.2 Example

```
Column      1   5
            BOT,24,46,3,D3
```

2.0 LOAD UNIT COMMON Command (Section 7.3.2)

2.1 Syntax

```
Column      1   5
            LUC,n1,n2,n3,n4
```

n_1 - number of independent process variables; $n_1 \leq 100$

n_2 - number of dependent process variables; $n_2 \leq 100$

n_3 - number of implicit process variables; $n_3 \leq n_2$

n_4 - number of words on disk required for all ODS and unit common data in the unit (refer to Appendix H).

2.2 Example

```
Column     1   5
           LUC,42,73,12,150000
```

3.0 BUILD TEST ODS Command (Section 7.3.3)

3.1 Syntax

```
Column     1   5
           BTD,m
```

m - Dn, where n is an ODS number; $0 \leq n \leq 9$.

m is an optional parameter.

3.2 Examples

```
Column     1   5
           BTD,D4
           BTD
```

4.0 MODIFY Command (Section 7.3.4)

4.1 Syntax

```
Column     1   5
           MOD
```

5.0 DECLARE Data Statement (Section 7.3.1)

5.1 Syntax

Column 1 5

DCL,v_1,v_2,...

v_i - a four character vector name

5.2 Example

Column 1 5

```
DCL,XUPB,YUPB,XLOB,YLOB
```

6.0 SYSTEM PARAMETER Data Statement (Section 7.3.1)

6.1 Syntax

Column 1 5

PAR,p_1,v_1,p_2,v_2,...

p_i - a four character system parameter name

v_i - a numerical value which may be:

(1) decimal, floating point

(2) decimal integer

(3) octal integer

(Refer to Section 7.1)

6.2 Example

Column 1 5

```
PAR,MXIT,1050,RLOC,14025,AZRO,1.5E-6
```

7.0 SET Data Statement (Section 7.3.1)

7.1 Syntax

Column 1 5

$\text{SET},v_1,o_1,v_2,o_2,\ldots$

v_i - a four character vector name

o_i - a set option which may be:

(1) decimal, floating point number

(2) ODS data set indicator

(3) activity indicator

(Refer to Section 7.1)

7.2 Example

Column 1 5

```
SET,XUPB,0.,PVAR,A,YLOB,D6,XTPN,2.0+2
```

8.0 ALTER Data Statement (Section 7.3.1)

8.1 Syntax

Column 1 5

$\text{ALT},c_1,r_1,v_1,c_2,r_2,v_2,\ldots$

c_i - a four character vector name

r_i - a one to four character variable name

v_i - a decimal, floating point number
or
an activity status (A or I)

(Refer to Section 7.1)

8.2 Example

```
Column   1   5
         ALT,XUPB,XX1A,2.0E-1,XLOB,XX46,4.576E-8,PVAR,XX1A,A
```

9.0 TAB Data Statements (Section 7.3.1)

9.1 TBX Data Statement (Section 7.3.1)

9.1.1 Syntax

Column	1	9	21	33	45	57
	TBX,	v_1	v_2	v_3	v_4	v_5

v_i - a four character name of a vector associated with independent process variables

9.1.2 Examples

Column	1	9	21	33	45	57
	TBX,	XUPB	XLOB	XUTR	XLTR	XFML

9.2 TBY Data Statement (Section 7.3.1)

9.2.1 Syntax

Column	1	9	21	33	45	57
	TBY,	v_1	v_2	v_3	v_4	v_5

v_i - a four character name of a vector associated with dependent process variables.

9.2.2 Example

Column	1	9	21	33	45	57
	TBY	YUPB	YLOB	YUTR	YLTR	YFML

9.3 Value Data Statements (Section 7.3.1)

9.3.1 Syntax

Column	2	9	21	33	45	57
	x	e_1	e_2	e_3	e_4	e_5

x - a one to four character process variable name
e - a floating point, decimal number
(refer to Section 7.1)

9.3.2 Example

Column	2	9	21	33	45	57
	XX4A	+.04369922E3	0.2	-4.8E-2	.5E-E	60.

9.4 ENT Data Statement (Section 7.3.1)

9.4.1 Syntax

Column	1 5
	ENT

9.5 Example of Tabular Input

Column	1	9	21	33	45	57
	TBX,	XDEL	XLTR	XFML	XAML	
	PRV1	.556E-6	2.0	.10	.5E-1	
	NPTG	.75E-3	2000.0	.02	.3E-3	
	TBY,	YUPB	YLOB	YUTR	YLTR	YDEL
	MTHG	1500.0	750.0	1495.	762.0	.33E-4
	ENT					

10.0 END DATA Data Statement (Section 7.3.1)

10.1 Syntax

Column	1
	END

11.0 FILE Command (Section 7.5.1)

11.1 Syntax

<u>Column</u>   1   5

FIL,m,v_1,n_1,v_2,n_2,...

m - an octal integer

v_i - a four character vector name
or
MATX,r where r is a one to four character process variable name

n_i - a decimal integer

(refer to Section 7.1)

11.2 Example

<u>Column</u>   1   5

```
FIL,/4065,XDEL,0,XELL,64,MATX,PRV1,128
```

12.0 GET Command (Section 7.5.2)

12.1 Syntax

<u>Column</u>   1   5

GET,m,v_1,n_1,v_2,n_2,...

m - an octal integer

v_i - a one to four character vector name

n_i - a decimal integer

(refer to Section 7.1)

12.2 Example

<u>Column</u>   1   5

```
GET,/4070,YLOB,256,YDEL,64
```

13.0 ACCEPT TEST ODS Command (Section 7.4.1)

13.1 Syntax

```
Column       1

             ATD
```

14.0 DELETE TEST ODS Command (Section 7.4.2)

14.1 Syntax

```
Column       1

             DTD
```

15.0 DELETE UNIT COMMON Command (Section 7.4.3)

15.1 Syntax

```
Column       1

             DUC
```

16.0 TURN ON PROGRAM Command (Section 7.4.4)

16.1 Syntax

```
Column       1   5

             TOP,n
```

n - a decimal integer; $n \leq 99$

16.2 Example

```
Column       1   5

             TOP,42
```

17.0 SAVE Command (Section 7.4.5)

17.1 Syntax

```
Column       1

             SAV
```

18.0 SOLVE Command (Section 7.6)

18.1 Syntax

```
Column       1   5

             SOL,n
```

n - one or two

18.2 Example

```
Column       1   5

             SOL,2
```

19.0 PRINT Command (Section 7.7.2)

19.1 Syntax

Column   1   5

PRT,v_1,v_2,v_3,...

v_i - a four character vector name

or

MATX,r where r is a one to four character variable name

19.2 Example

```
Column       1   5

             PRT,XUPB,XELL,YELL,XSTT,MATX,XVRI
```

20.0 MATRIX Command (Section 7.7.2)

20.1 Syntax

<u>Column</u>   1

MAT

21.0 DUMP Command

21.1 Syntax

<u>Column</u>   1   5

DMP,n_1,n_2,n_3,...

n_i - ODS data set indicators (C or Dn, 0 n 9; refer to Section 7.1)

22.0 END OF TASK Command (Section 7.2.2)

22.1 Syntax

<u>Column</u>   1

EOT

APPENDIX E

INTERNAL TASK FORMATS

APPENDIX E

INTERNAL TASK FORMATS

Internal tasks are generated by real-time functional programs and stored in *bulk memory*. The generating program presents the task to OPO and manipulates it (if necessary) through calls on the task management functions. The task management functions available to the generating program are given in Section 8.2.

There are three primary differences between external and internal tasks that are of concern to the user:

1) The commands and data statements comprising an internal task are represented in an internal format.

2) An internal task is headed by a task status word which may be interrogated by the generating functional program to determine the status of the task.

3) The task management functions are invoked by setting up of parameters and functional program turn-ons.

The meanings of the commands and data statements used in an internal task are identical to their meanings in external tasks.

E.1 COMMAND AND DATA STATEMENT INTERNAL FORMATS

The internal formats for each command and data statement are given in Section E.4. The internal formats are designed to:

1) Occupy less memory space than the external formats.

2) Minimize the effort required for the user to generate an internal task and yet retain a high degree of similarity to the syntax of the external formats.

3) Minimize the number of conversions that must be performed by OPO in processing commands and data statements.

A command expressed in its internal format may be viewed as a parameter block for OPO. The general form of an internal format is:

1) The command or data statement keyword in the first word of the command.

2) The command length in the second word of the command.

3) Command parameters in succeeding words.

E.1.1 Keywords

The keywords (e.g. BOT, DCL, FIL) are expressed in the character code of the system (Common Peripheral Code or ASCII). The keyword is left-justified in the word. Assuming Common Peripheral Code (CPC) to be the character set code of the system, a FIL keyword would appear as 26314320_8 in the internal representation. In ASCII, the keyword would be 106111114_8. The PAL pseudo-op

```
CON A,3,XXX
```

is a convenient way of generating a keyword.

E.1.2 Command Length

The command length, occupying the second word of the command, is the number of words required for storage of the command. The number is expressed as a single precision, binary integer. The command length includes the words required to hold the keyword and the command length itself.

A FIL command requiring 20_{10} words of bulk storage for its parameters will have a command length of 22_{10} words. The first two words of this command in its internal format would appear as:

L	26314320
L+1	00000026

assuming Common Peripheral Codes (CPC).

E.1.3 Parameters

Parameters may be classified as:

1) Vector and variable names

2) Numerical Values

   a) double precision, floating point

   b) single precision integers

      i) task priorities

      ii) unit numbers

3) Data set indicators

4) Activity status indicators

5) Task identifier

Vector and Variable Names

Vector and variable names require two words in an internal command, regardless of the length of the name. The names are expressed in the character code of the system (Common Peripheral Code or ASCII) and are left-justified in the first of the two words.

The vector name XSTT would appear as

67626363
20202020

in a CPC system and as

130123124
124040040

in an ASCII system.

The variable name X2 would appear as

67022020
20202020

in a CPC system and as

130062040
040040040

in an ASCII system.

Numeric Values

All numbers, regardless of precision, require two words in an internal format. Double precision numbers occupy both words while an integer must be right-justified in the second word with first word containing zeros.

All numbers are expressed internally in binary. The PAL pseudo-ops, CON and DCN, provide a convenient way to provide the numbers in the generating program.

Examples:

double precision, floating point, decimal number

```
DCN F,2.325E2
```

single precision, decimal integer

```
CON O,0
CON D,9
```

single precision, octal integer

```
CON O,0
CON O,76
```

Task priorities and unit numbers, subject to the restrictions specified in Section 7.1, are treated as single precision, decimal integers.

Data Set Indicators

The data set indicators (U, C, and Dn; refer to Section 7.1) each require two words in an internal format. The first word always contains zero. The indicator is in the second word.

The indicators U and C are left-justified in the second word. Thus,

```
CON 0,0
CON A,1,U
```

will generate a data set indicator specifying unit common.

The ODS specifier Dn is handled differently. The D is discarded and n is treated as a decimal integer *right-justified* in the second word. The data set indicator D4 (in its external representation) would be generated internally by

```
CON 0,0
CON D,4
```

Activity Status Indicators

The activity status indicators (A and I) require two words in an internal format. They are analagous to the data set indicators U and C. The PAL pseudo-ops

```
CON 0,0
CON A,1,I
```

generates an activity status in its internal form.

Task Identifiers

A task identifier also requires two words in an internal format. The first word is filled with zeros while the identifier is left-justified in the second word and expressed in the character code of the system. The PAL pseudo-ops

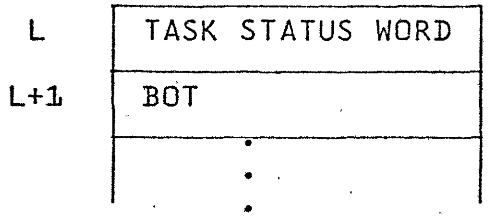
```
CON 0,0
CON A,2,AB
```

generate the internal form of the task identifier AB.

The internal formats of each OPO command and data statement is given in Section E.4.

E.2 TASK STATUS WORD

The task status word (TSW) is a flag word allocated by the user at the head of his task. The TSW must occupy the bulk memory word immediately preceding the BOT command of the task



L	TASK STATUS WORD
L+1	BOT
	⋮

It is the location of the TSW that the user furnishes ACCEPT TASK when presenting the task to OPO.

The structure of a TSW is illustrated in Figure E-1.

23 8	7	6	5	4	3	2	1	0
unused	TO	TS	TE	TD	DI	TI	TA	TC

Figure E-1 Task Status Word Structure

Each of the items in a TSW is a one bit flag having an on-off setting. The items are each defined below.

TASK ACCEPTED (TE; bit 5)

Set (1) - The task has been accepted by ACCEPT TASK and has been entered in the appropriate unit task queue.

Reset (0) - The task has not yet been accepted by ACCEPT TASK and is not yet in a unit's task queue.

TASK STARTED (TS; bit 6)

Set (1) - Execution of the task has been initiated; the first command in the task is either being processed or has been processed.

Reset (0) - Processing of the first command in the task has not been initiated.

TASK DONE (TD; bit 4)

Set (1) - The task has been terminated and is no longer in the OPO unit task queues. The task may have been terminated by deletion, abortion or successful completion.

Reset (0) - The task has not yet been terminated and is still in a unit task queue.

TASK DELETED (TO; bit 7)

Set (1) - Execution of the task was never initiated and the task has been removed from its unit's task queue as the result of a call on the task management function DELETE TASK.

Reset (0) - The task has not been deleted.

TASK ABORTED (TA; bit 1)

Set (1) - The task has been aborted, either as a result of an operator response to an error or because a class 6 error was encountered (refer to Section 9 of this manual and to the OPO Operations Manual).

Reset (0) - The task has not been aborted.

UNIT INACTIVE (TI; bit 2)

The meaning of this flag is defined only when the TASK ABORTED flag is set.

Set (1) - The task has been aborted because it was directed at a unit which is either unallocated or allocated and inactive.

Reset (0) - The task has been aborted, but not because the unit was inactive.

ODS INACTIVE (DI; bit 3)

The meaning of this flag is defined only if the TASK ABORTED flag is set.

Set (1) - The task has been aborted because it was directed at an ODS which was either unallocated or allocated and inactive.

Reset (0) - The task has been aborted but not because the ODS was inactive.

TASK CORRECTED (TC; bit 0)

Set (1) - The task has been corrected in the user's bulk memory task area as the result of an operator response to an error diagnostic. The pertinent point is to note that the task has been altered in some fashion between the time it was entered into OPO and the time that its execution was completed.

Reset (0) - No task correction has occurred.

E.3 TASK MANAGEMENT CALLS

The procedures for the invoking of a task management function by an internal user are given in Sections 8.2 and 8.3 of this manual.

E.4 COMMAND AND DATA STATEMENT INTERNAL FORMATS

This section contains general forms, for each command and data statement, of their internal formats. Component groups are denoted by a solid line along side the words in the appropriate commands. Each component type permitted in a given command appears at least once in the appropriate general form.

E.4.1 BEGIN TASK

word 1	BOT	
2	10_{10}	command length
3	0	
4	n_t	task identifier
5	0	
6	n_p	task priority
7	0	
8	n_u	unit number
9	0	
10	n_m	data set indicator

E.4.2 LOAD UNIT COMMON

word 1	LUC	
2	10_{10}	command length
3	0	
4	n_x	number of independent process variables (single precision, binary integer)
5	0	
6	n_y	number of dependent process variables (single precision, binary integer)
7	0	
8	n_i	number of implicit process variables (single precision, binary integer)
9	0	
10	n_w	bulk storage requirements for unit (single precision, binary integer)

E.4.3 BUILD TEST ODS

word 1	BTD	
2	4	command length
3	0	
4	n	ODS number (single precision, binary integer)

or

word 1	BTD	
2	2	command length

E.4.4 MODIFY

word 1	MOD	
2	2	command length

E.4.5 DECLARE

word 1	DCL	
2		command length
3	v_1	vector name
4		
5	v_2	vector name
6		
7	v_3	vector name
8		
⋮	⋮	

E.4.6 SYSTEM PARAMETER

word 1	PAR	
2		command length
3	p	parameter name (left-justified in first word of a two word blank)
4		
5	f	parameter value (double precision, floating point binary number)
6		
7	p	
8		
9	0	
10	n_m	ODS number (single precision, binary integer)
11	p	
12		
13	0	
14	n	parameter value (single precision binary integer)
⋮	⋮	

E.4.7 SET

word 1	SET	
2		command length
3	v	vector name (left-justified)
4		
5	f	initialization option value (double precision, floating point binary number)
6		
7	v	
8		
9	0	
10	n	ODS number (single precision, binary integer)
11	PVAR	
12		
13	0	
14	a	activity status (A or I in character codes, left-justified)
.	.	
.	.	
.	.	

E.4.8 ALTER

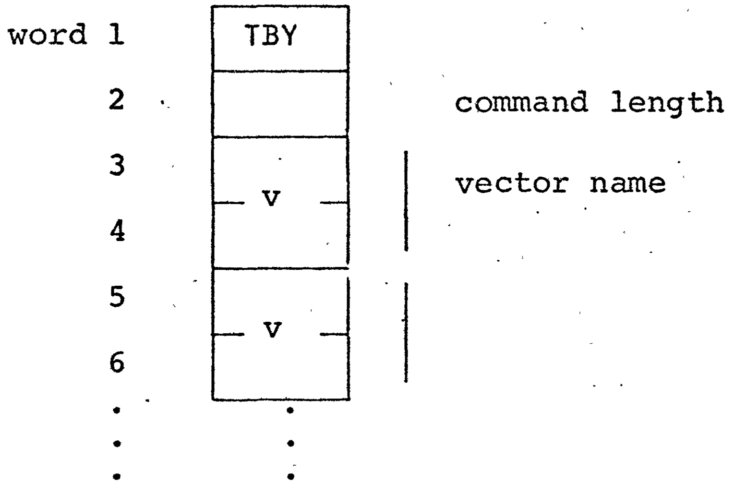
word		
1	ALT	
2		command length
3	v	vector name (left-justified)
4		
5	r	variable name (left-justified)
6		
7	f	value (double precision, floating point binary number)
8		
9	PVAR	
10		
11	r	
12		
13	0	
14	a	activity status (A or I in character codes, left-justified)
⋮	⋮	

E.4.9 Tabular Input

TBX word		
1	TBX	
2		command length
3	v	vector name
4		
5	v	
6		
	⋮	

A maximum of five vector names may be specified in a TBX statement. A TBX statement with no vector names is valid when the statement and its associated value statements are to be used only to define independent process variable names.

TBY



A maximum of five vector names may be specified in a TBY statement. A TBY statement with no vector names is valid when the statement and its associated value statements are to be used only to define independent process variable names.

Value Statement

word 1	0	
2		command length
3	r	variable name
4		
5	f	value (double precision, floating point, binary)
6		
7	f	
8		
	• • •	

The number of values (f) specified in a value statement must be exactly equal to the number of vector names specified in the preceding TBX or TBY statement.

ENT

word 1	ENT	
2	2	command length

E.4.10 DEFINE

word 1	DEF	
2		command length
3	v_1	vector name
4		
5	v_2	
6		
7	v_3	
8		
⋮	⋮	

E.4.11 END

word 1	END	
2	2	command length

E.4.12 FILE

word 1	FIL	
2		command length
3	0	
4	n_I	core memory address of interlock status block (single precision, binary integer)
5	v	vector name (left-justified)
6		
7	0	
8	n_r	relative address in disc area (single precision, binary integer)
9	MATX	
10		
11	r	variable name (left-justified)
12		
13	0	
14	n	relative address in disc area (single precision, binary integer)
⋮	⋮	

E.4.13 GET

word 1	GET	
2		command length
3	0	
4	n	interlock status block address (single precision, binary integer)
5	v	vector name (left-justified)
6		
7	0	relative address in disc area (single precision, binary integer)
8	n	
⋮	⋮	

E.4.14 ACCEPT TEST ODS

word 1	ATD	
2	2	command length

E.4.15 DELETE TEST ODS

word 1	DTD	
2	2	command length

E.4.16 DELETE UNIT COMMON

word 1	DUC	
2	2	command length

E.4.17 TURN ON PROGRAM

word 1	TOP	
2	4	command length
3	0	
4	n	program number (single precision, binary integer)

E.4.18 SAVE

word 1	SAV	
2	2	command length

E.4.19 SOLVE

word 1	SOL	
2	4	command length
3	0	
4	n	starting basis option (0 or 1; single precision, binary integer)

or

word 1	SOL
2	2

E.4.20 PRINT

word 1	PRT	
2		command length
3	v_1	vector name (left-justified)
4		
5	v_2	
6		
7	MATX	
8		
9	r	variable name
10		
⋮	⋮	

E.4.21 MATRIX

word 1	MAT	
2	2	command length

E.4.22 DUMP

word 1	DMP	
2		command length
3	0	ODS data set indicator (single precision, binary integer or C)
4	n_1	
5	0	
6	n_2	
⋮	⋮	

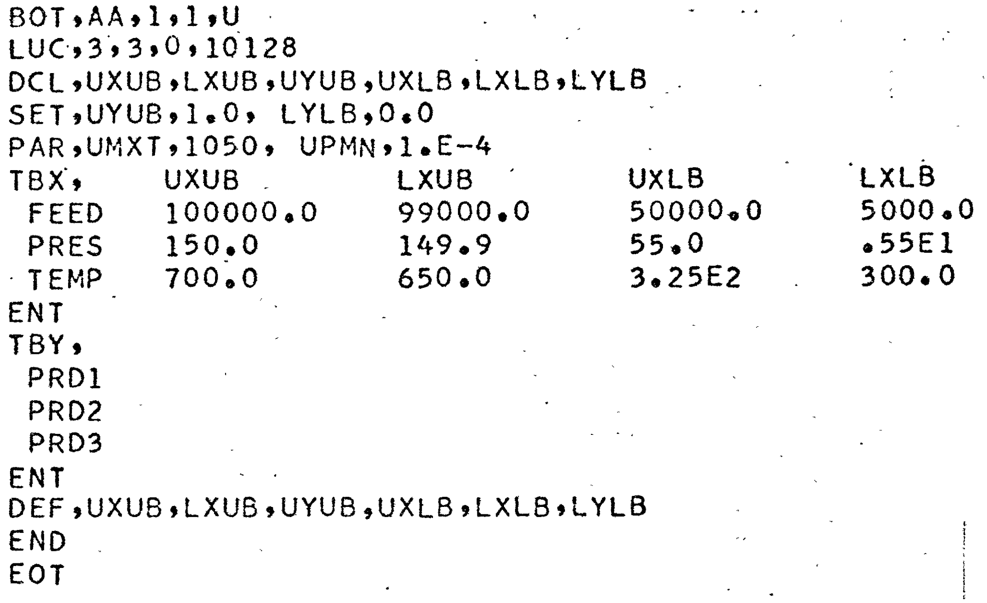
E.4.23 END OF TASK

word 1	EOT	
2	2	command length

E.5 INTERNAL TASK EXAMPLE

An OPO task will be presented in both its external and internal representations. The internal representation will be in the form of PAL pseudo-ops which would generate the appropriate formats is assembled.

The task in its external form is shown in Figure E-2. This is one of the two tasks which appeared in Figure 6-1 of this manual.



```
BOT,AA,1,1,U
LUC,3,3,0,10128
DCL,UXUB,LXUB,UYUB,UXLB,LXLB,LYLB
SET,UYUB,1.0, LYLB,0.0
PAR,UMXT,1050, UPMN,1.E-4
TBX,     UXUB        LXUB        UXLB        LXLB
 FEED    100000.0    99000.0     50000.0     5000.0
 PRES    150.0       149.9       55.0        .55E1
 TEMP    700.0       650.0       3.25E2      300.0
ENT
TBY,
 PRD1
 PRD2
 PRD3
ENT
DEF,UXUB,LXUB,UYUB,UXLB,LXLB,LYLB
END
EOT
```

Figure E-2   OPO Task in External Format

The task in its internal form appears on the following two pages. Some items of interest concerning the internal task are:

1) The use of the pseudo-ops to achieve the proper number of words for a parameter. For example, CON A,5,xxx reserves two words for a name regardless of the character code (Common Peripheral Code or ASCII) used in the system. DCN D,nnn is also very useful.

2) Only the first empty component of the TBY statement must be explicitly zeroed (see card 71). The first empty component in a TBX or TBY statement implies that the remainder of the components are also empty. OPO then proceeds to process value statements.

```
Card
 No.

  1      CON O,0            TASK STATUS WORD
  2      CON A,3,BOT        BOT KEYWORD
  3      CON D,10               COMMAND LENGTH
  4      CON O,0
  5      CON A,2,AA             TASK IDENTIFIER
  6      DCN D,1                TASK PRIORITY
  7      DCN D,1                UNIT NUMBER
  8      CON O,0
  9      CON A,1,U              DATA SET INDICATOR
 10      CON A,3,LUC        LUC KEYWORD
 11      CON D,10               COMMAND LENGTH
 12      DCN D,3                NO. OF INDEPENDENT VARIABLES
 13      DCN D,3                NO. OF DEPENDENT VARIABLES
 14      DCN D,0                NO. OF IMPLICIT VARIABLES
 15      DCN D,10128            UNIT BULK STORAGE REQUIREMENT
 16      CON A,3,DCL        DCL KEYWORD
 17      CON D,14               COMMAND LENGTH
 18      CON A,5,UXUB           VECTOR NAME
 19      CON A,5,LXUB           VECTOR NAME
 20      CON A,5,UYUB           VECTOR NAME
 21      CON A,5,UXLB           VECTOR NAME
 22      CON A,5,LXLB           VECTOR NAME
 23      CON A,5,LYLB           VECTOR NAME
 24      CON A,3,SET        SET KEYWORD
 25      CON D,10               COMMAND LENGTH
 26      CON A,5,UYUB           VECTOR NAME
 27      DCN F,1.0              INITIALIZATION VALUE
 28      CON A,5,LYLB           VECTOR NAME
 29      DCN F,0.0              INITIALIZATION VALUE
 30      CON A,3,PAR        PAR KEYWORD
 31      CON D,10               COMMAND LENGTH
 32      CON A,5,UMXT           PARAMETER NAME
 33      DCN D,1050             PARAMETER VALUE
 34      CON A,5,UPMN           PARAMETER NAME
 35      DCN F,1.E-4            PARAMETER NAME
 36      CON A,3,TBX        TBX KEYWORD
 37      CON D,12               COMMAND LENGTH
 38      CON A,5,UXUB           VECTOR NAME
 39      CON A,5,LXUB           VECTOR NAME
 40      CON A,5,UXLB           VECTOR NAME
 41      CON A,5,LXLB           VECTOR NAME
 42      DCN D,0                EMPTY TBX COMPONENT
 43      CON O,0            VALUE STATEMENT
 44      CON D,14               COMMAND LENGTH
 45      CON A,5,FEED           INDEPENDENT VARIABLE NAME
 46      DCN F,100000.0         VALUE FOR UXUB(FEED)
 47      DCN F,99000.0          VALUE FOR LXUB(FEED)
 48      DCN F,50000.0          VALUE FOR UXLB(FEED)
 49      DCN F,5000.0           VALUE FOR LXLB(FEED)
 50      DCN D,0                EMPTY VALUE STATEMENT COMPONENT
 51      CON D,0            VALUE STATEMENT
 52      CON D,14               COMMAND LENGTH
 53      CON A,5,PRES           INDEPENDENT VARIABLE NAME
 54      DCN F,150.0            VALUE FOR UXUB(PRES)
 55      DCN F,149.9            VALUE FOR LXUB(PRES)
```

Figure E-3   Internal Task Example

```
56      DCN F,55.0                  VALUE FOR UXLB(PRES)
57      DCN F,.55E1                 VALUE FOR LXLB(PRES)
58      DCN D,0                     EMPTY VALUE STATEMENT COMPONENT
59      CON O,0                 VALUE STATEMENT
60      CON D,14                    COMMAND LENGTH
61      CON A,5,TEMP                INDEPENDENT VARIABLE NAME
62      DCN F,500.0                 VALUE FOR UXUB(TEMP)
63      DCN F,650.0                 VALUE FOR LXUB(TEMP)
64      DCN F,3.25E2                VALUE FOR UXLB(TEMP)
65      DCN F,300.0                 VALUE FOR LXLB(TEMP)
66      DCN D,0                     EMPTY VALUE STATEMENT COMPONENT
67      CON A,3,ENT             ENT KEYWORD
68      CON D,2                     COMMAND LENGTH
69      CON A,3,TBY             TBY KEYWORD
70      CON D,12                    COMMAND LENGTH
71      DCN D,0                     EMPTY TBY STATEMENT COMPONENT
72      BSS 8                       4 MORE EMPTY COMPONENTS
73      CON O,0                 VALUE STATEMENT
74      CON D,14                    COMMAND LENGTH
75      CON A,5,PRD1                DEPENDENT VARIABLE NAME
76      BSS 10                      5 EMPTY COMPONENTS
77      CON O,0                 VALUE STATEMENT
78      CON D,14                    COMMAND LENGTH
79      CON A,5,PRD2                DEPENDENT VARIABLE NAME
80      BSS 10                      5 EMPTY COMPONENTS
81      CON O,0                 VALUE STATEMENT
82      CON D,14                    COMMAND LENGTH
83      CON A,5,PRD3                DEPENDENT VARIABLE NAME
84      BSS 10                      5 EMPTY COMPONENTS
85      CON A,3,ENT             ENT KEYWORD
86      CON D,2                     COMMAND LENGTH
87      CON A,3,DEF             DEF KEYWORD
88      CON D,14                    COMMAND LENGTH
89      CON A,5,UXUB                VECTOR NAME
90      CON A,5,LXUB                VECTOR NAME
91      CON A,5,UYUB                VECTOR NAME
92      CON A,5,UXLB                VECTOR NAME
93      CON A,5,LXLB                VECTOR NAME
94      CON A,5,LYLB                VECTOR NAME
95      CON A,3,END             END KEYWORD
96      CON D,2                     COMMAND LENGTH
97      CON A,3,EOT             EOT KEYWORD
98      CON D,2                     COMMAND LENGTH
```

3) Because no vectors were specified in the TBY statement, only the variable name component of the associated value statements will be examined by OPO. Thus, the value components of the value statements need not be explicitly zeroed although the words must be allocated (see cards 73-84).

APPENDIX F

REPORT FORMATS

APPENDIX F

REPORT FORMATS

Iteration Log

The iteration log is printed under control of the report control parameter RPRT. If this parameter contains a one (1), iteration logs are printed as often as specified in the parameter MXIL.

If the user makes no specification, every tenth iteration is logged.

The iteration log will present the following data:

1) KITT, number of iterations in move

2) Index of variable entering basis

3) Index of variable leaving basis

4) Value of the pivot element for the iteration

5) OBJL, value of the objective function after iteration

6) Cause of iteration (0 inversion, 1 optimize)

7) Unit and optimizer data set identification

NOTE

Independent variables are indexed from 1.
Dependent variables are indexed from 1001.

Subset Select Log

Subset select cycles are logged if iterations are logged. They may be specified without iteration logs by placing a two (2) in the report control parameter RPRT. The frequency of logging is specified in MXSS. Subset select cycles will be logged for optimize cycles, but not invert cycles.

The subset select log will present the following data.

1) KSSS , number of subset select cycles in step.

2) KXCH number of exchanges made in cycle.

3) KXBS the number of pivot steps involved in transforming the matrix

4) OBJL the objective function value.

5) KITT the number of iterations in move.

6) KRET number of variables rejected due to unacceptable pivots.

7) UNIT and ODS identification

Step Logs

Step logs will be printed if subset select logs are printed. Step logs may be printed without subset select logs by placing a four (4) in the parameter RPRT. The frequency of printing step logs is controlled by the parameter MXSL.

If the user makes no specifications, every tenth step is logged.

The step logs will present the following data.

1) KSTP number of steps completed in move

2) OBJL objective function value

3) KITT number of iterations in move

4) KSSS numbers of subset select cycles in step

5) KREJ number of variables rejected due to unacceptable pivots.

6) UNIT and ODS identification

Solution Report

The solution report is printed under control of the report control parameter RPRJ. If this parameter contains an eight (8), a solution report is printed at the end of each move and at step intervals specified by the parameter MXSO.

If the user makes no specification, the solution report is printed at the end of each move.

The solution report will present the following data.

1) UNIT identification

2) Optimizer DATA SET identification

3) Time of day

4) KSTP the number of steps in the move

5) OBJL the value of the objective function at the end of the move

6) KITT the number of iterations in the move

7) The number of infeasibilities

8) The number of variables outside the target range

9) The improvement in the objective function

For each variable the solution report will show:

1) Variable label.

2) Optimum value of the variable. For independent variables, this is the value determined by the solution. For dependent variables, this is the value generated by calling MODEL with the optimum values of the independent variables.

3) Status indicates the status of the variable at the optimum with respect to four types of bounds;

a) OL (outer limits)

b) ML (move limits)

c) TG (targets)

d) ST (step limits)

The status indicators are interpreted as follows:

-2 optimum value is less than lower limit

-1 optimum value is at the lower limit

0 optimum value is between lower and upper limits

1 optimum value is at upper limit

2 optimum value is greater than upper limit

9 the variable is not being set by the LP

4) BOUNDS gives the most restrictive limits of feasibility which were applicable for each step. Each bound value is followed by a code which denotes the type of limit which was the bounding value.

1 = outer limit

2 = move limit

3 = step limit

5) Target gives the upper and lower limits of the target range for the variable.

Reduced Cost Report

The reduced cost report is printed under control of the report control parameter RPRT. If this parameter contains a sixteen (16), a reduced cost report is printed at the end of each move and at step intervals specified by the parameter MXRC.

If the user makes no specification, the reduced cost report is printed at the end of each move.

The reduced cost report will show:

1) Variable name

2) Indicator if variable is basic (B)

3) Reduced cost, i.e., the cost of moving a non-basic variable in the positive direction

4) Effective cost, including target penalties, of moving a non-basic variable in the positive direction.

Report Selection

Report	RPRT	Frequency
Iteration Log	1	at each MXIL iterations
Subset Select LOG	1 or 2	at each MXSS subset select cycles
Step Logs	1 or 2 or 4	at each MXSL steps
Solution Report	8	at each MXSO steps and at end of move
Reduced Cost Report	16	at each MXRC steps and at end of move

Any combination of reports may be obtained by setting RPRT to the sum of the settings required to get the individual reports.

Matrix Command Output

The matrix command will cause the printing of a full matrix tableau. For each row and column the following data will be included:

1) Variable identification

NOTE

Independent variables are indexed from 1, dependent variables are indexed from 1001.

2) Upper bound

3) Variable value

For each non-basic variable (column), there will be included the reduced cost (DJ).

When the matrix contains more columns than can fit on a single page, the excess columns are output on additional pages.

Print Command Output

The output generated by a print command is a tabular output of all the vectors named in the print command. Seven vectors can be accommodated on a page. As many pages as necessary will be used to accommodate all the vectors named in the command.

Each page will be headed by a line which includes:

1) Unit identification
2) Data set identification
3) Time of day

Dump Command Output

The output of a dump command is a tabular output of all the vectors in a data set. Seven vectors can be accommodated on a page and as many pages as necessary are used to output the entire data set. Data set parameters are output as vectors of length one (1).

Each data set dump is headed by a line which includes:

1) Unit identification
2) Data set identification
3) Time of day

REPORT FORMAT EXAMPLES

ITERATION LOG

```
INTERATION AAAA  BBBB  CCCC  .DDDDDE+DD  .FFFFFFFE+FF  G  HH  II

            A = NUMBER OF ITERATIONS IN MOVE
            B = INDEX OF ENTERING VARIABLE
            C = INDEX OF LEAVING VARIABLE
            D = VALUE OF PIVOT
            F = NEW OBJECTIVE FUNCTION VALUE
            G = 0,INVERSION 1,OPTIMIZATION
            H = UNIT
            I = ODS
```

SUBSET SELECTION LOG

```
SUBSET SEL AA AA BBBB CCCC  .DDDDDDDE+DD  FFFF  GGGG  HH  II

            A = NUMBER OF SS CYCLES IN STEP
            B = NUMBER OF EXCHANGE MADE IN CYCLE
            C = NUMBER OF PIVOT STEPS IN UPDATE
            D = NEW OBJECTIVE VALUE
            F = NUMBER OF ITERATIONS IN MOVE
            G = NUMBER OF REJECTED PIVOTS
            H = UNIT IDENTIFIER
            I = ODS IDENTIFIER
```

STEP LOG

```
STEP LOG  AAAA  .BBBBBBBBE+BB  CCCCC  DDDD  FFFF GG  HH

            A = NUMBER OF STEPS IN MOVE
            B = OBJECTIVE FUNCTION VALUE
            C = NUMBER OF ITERATIONS IN MOVE
            D = NUMBER OF SS CYCLES IN STEP
            F = NUMBER OF REJECTED PIVOTS
            G = UNIT IDENTIFIER
            H = ODS IDENTIFIER
```

SOLUTION REPORT

```
UNIT --   ODS --   TIME ------   STEP ----   OBJECTIVE------------
INTERATION ----- INF ---  OUT OF PNG ---  OBJ IMP -------------

 VARBL  OPTIMUM            STATUS           BOUNDS                   TARGETS
 LABEL   VALUE          OL ML TG ST     LOWER  *   UPPER   *    LOWER      UPPER

 AAAA  BBBBBBBE+BB      CC DD EE FF   GGGGGGGG H IIIIIIII J  KKKKKKKK  LLLLLLLL

       A = VARIABLE LABEL
       B = OPTIMUM VALUE
       C = OUTER LIMIT STATUS
       D = MOVE LIMIT STATUS
       E = TARGET LIMIT STATUS
       F = STEP LIMIT STATUS
       G = LOWER BOUND VALUE
       H = LOWER BOUND TYPE
       I = UPPER BOUND VALUE
       J = UPPER BOUND TYPE
       K = LOWER TARGET VALUE
       L = UPPER TARGET VALUE
```

REDUCED COST

```
       REDUCED COST  VARBL   *          DJ             DJI

                     AAAA    B    CCCCCCCCCCCC  DDDDDDDDDDDD

                A = VARIABLE LABEL
                B = BASIC,B OR NON-BASIC
                C = REDUCED COST
                D = EFFECTIVE COST
```

MAT COMMAND OUTPUT

```
UNIT --  DATA SET --  TIME ------

LABEL                      AAAA       AAAA       AAAA       AAAA       AAAA

    UPPER BD            .BBBBBE+B  .BBBBBE+B  .BBBBBE+B  .BBBBBE+B  .BBBBBE+B

            VALUE       .CCCCCE+C  .CCCCCE+C  .CCCCCE+C  .CCCCCE+C  .CCCCCE+C

              DJ        .DDDDDE+D  .DDDDDE+D  .DDDDDE+D  .DDDDDE+D  .DDDDDE+D

AAAA .BBBBBE+B .CCCCCE+C .FFFFFE+F  .FFFFFE+F  .FFFFFE+F  .FFFFFE+F  .FFFFFE+F
AAAA .BBBBBE+B .CCCCCE+C .FFFFFE+F  .FFFFFE+F  .FFFFFE+F  .FFFFFE+F  .FFFFFE+F

                A=VARIABLE LABEL
                B=UPPER BOUND ON VARIABLE
                C=VARIABLE VALUE
                D=VARIABLES REDUCED COST
                F=MATRIX ELEMENT VALUE
```

PRINT COMMAND OUTPUT

```
UNIT --  DATA SET --  TIME ------

LABEL  AAAA       AAAA       AAAA       AAAA       AAAA

BBBB   .CCCCCE+C .CCCCCE+C .CCCCCE+C .CCCCCE+C .CCCCCE+C
BBBB   .CCCCCE+C .CCCCCE+C .CCCCCE+C .CCCCCE+C .CCCCCE+C

                  A=VECTOR LABEL
                  B=VARIABLE LABEL
                  C=ELEMENT VALUE
```

DUMP COMMAND OUTPUT

```
UNIT --  DATA SET --  TIME ------

  AAAA       AAAA       AAAA       AAAA       AAAA

 .BBBBBE+B .BBBBBE+B .BBBBBE+B .BBBBBE+B .BBBBBE+B

  CCCC       CCCC       CCCC      CCCC

 .DDDDDE+D .DDDDDE+D .DDDDDE+D .DDDDDE+D
 .DDDDDE+D .DDDDDE+D .DDDDDE+D .DDDDDE+D

                  A=PARAMETER LABEL
                  B=PARAMETER VALUE
                  C=VECTOR LABEL
                  D=VECTOR ELEMENT VALUE
```

APPENDIX G

OPO STATUS REPORT

APPENDIX G

OPO STATUS REPORT

A typical example of the general appearance of the STATUS REPORT is shown below. The numbers on the right are note numbers to explanations of particular items in the report.

QUEUE STATUS REPORT

```
  OPO BULK 0 /00302400 /702000
  UNALLOCATED OPO BULK /00520500 /466700

  UNIT 1  DESCRIP /026420  UNIT BULK /025700
  UNSED UNIT BULK /00342200 /025700
  QUEUE PRIORITY 03  TIME 093848  TASK-IN-PROGRESS C7
       TASKS PRIORITY  TIME    SOURCE      DATA SET
        C7      03     093848  1 /00104307    1 C
        A4      12     094034  CARD READER    U
       DATA-SETS   BASE    LENGTH
        U         /00306400 /017400
        1 C       /00326000 /014200

  UNIT 2  DESCRIP /016432  UNIT BULK /00370100 /041400
  UNSED UNIT BULK /00511100 /004400
  QUEUE EMPTY
       DATA-SETS   BASE    LENGTH
        U         /00410100 /020600

  UNIT 3  DESCRIP /016444  UNIT BULK /00431500 /037600
 UNIT DELETED

  UNIT 4  DESCRIP /016456  UNIT BULK /00471300 /024200
  UNSED UNIT BULK /00430700 /000600
  QUEUE PRIORITY 07  TIME 093619  NO TASK-IN-PROGRESS
       TASKS PRIORITY  TIME    SOURCE      DATA SET
        RP      07     093619  0 /00766470    I
      DATA-SETS    BASE    LENGTH
       U          /00473300 /005400
       1          /00500700 /004100
       2 C        /00505000 /004100

  END REPORT
```

NOTES

1) The single number is the system bulk controller number. The eight octal digit number following is the base address of OPO system bulk storage. The last number is the original size of OPO bulk storage.

2) Wherever two numbers appear, as here, the first will be the base location and the second the size of the area.

3) The permanent core location of Unit 1's descriptor is /016420.

4) The Task Time is the time the task was accepted by OPO.

5) The letter U and C mean Unit Common Data Set and Control Data Set, respectively.

6) Unit 2 has no tasks to execute, therefore, its queue is empty.

7) Unit 3 has been deleted, but it's allocated storage remains in force because of the existence of Unit 4.

8) Although Unit 4 has tasks to process, none is currently in progress.

APPENDIX H

INTERLOCK STATUS BLOCK

APPENDIX H

INTERLOCK STATUS BLOCK

The interlock status block (ISK) structure is as follows:

<table>
<tr><td>L</td><td colspan="6">23 COMMUNICATION AREA BASE LOCATION 0
(CABA)</td></tr>
<tr><td>L+1</td><td>23
WW</td><td>22
WA</td><td>21
CO</td><td>20 18
CABC</td><td>17 7
CAL</td><td>6 0
UN</td></tr>
<tr><td>L+2</td><td colspan="6">23 0
NUMBER OF READERS (NOR)</td></tr>
</table>

Communication Area Base Location (CABA) - The bulk memory address of the first word in the communication area.

Communication Area Bulk Controller (CABC) - The device number of the controller in charge of the bulk memory device holding the communication area.

Write Waiting Flag (WW) - The flag is set (1) when a write operation utilizing the communication area is pending, either because a prior write operation on the area is in progress or because read operations on the area are in progress. The flag is reset (0) when there is no write operation waiting to modify the contents of the communication area.

Write Active Flag (WA) - The flag is set (1) when a write operation is currently modifying the contents of the communication area. The flag is reset (0) where there is no write operation on the area currently in execution.

Unit Number (UN) - If this item is zero, the communication area may be accessed by FIL or GET commands in a task directed at any defined, or active, unit. If the item is non-zero, it contains a unit number. The communication area may be accessed only by FIL and GET commands in tasks directed at the specified unit.

Control ODS Only Write (CO) - If reset (0), the communication area may be accessed by FIL or GET commands in a task directed at any data set - unit common, a test ODS, or the control ODS. If set (1), only FIL and GET commands in tasks directed at a control ODS may access the area.

Communication Area Length (CAL) - The number of words in the communication area.

Number of Readers (NOR) - The number of programs currently reading data from the communication area. Each reading program increments NOR by one just prior to initiating an operation on the area and decrements it by one when the operation is completed. No write operation on the area should be initiated while NOR is non-zero.

Usage

The user is responsible for the allocation in core memory of all required interlock status blocks. An interlock status block may be assigned to permanent core or to working core, at the user's option. However, the ISB addresses specified in FIL and GET commands are absolute addresses. This means that if a specified ISB is relocated in core, its contents will be lost to OPO.

The communication area protection items are only operable if external functional programs accessing the areas properly update them. For instance, if an external functional program initiates a logical read consisting of several physical reads without incrementing the Number of Readers, an OPO FIL command may initiate a write operation between execution of two reads and destroy data desired by the external functional program.

There should be at exactly one ISB for each bulk communication area that the user defines. Sharing an ISB between two or more areas by modifying its contents or providing two or more ISB's for a single area will most probably result in the loss of area protection.

APPENDIX I

STORAGE REQUIREMENTS

APPENDIX I

STORAGE REQUIREMENTS

1.0 CORE STORAGE REQUIREMENTS

The following requirements are based on the size of the largest program in the OPO system.

The minimum amount of core in which OPO can operate is 2000 words. However, for increased efficiency, it is advisable to have approximately 4000 words available for OPO so that more than one system component can reside in core at a given time.

In addition, OPO requires 50 words of common plus 10 words of common for each unit that the user's particular system can handle. The maximum number of units to be permitted is specified when the user's OPO system is installed.

These requirements are in addition to any core required for the Monitor and other programs in the user's system.

2.0 BULK STORAGE REQUIREMENTS

The following bulk storage estimates are based on the vectors required for models and units.

Quantities used in the following discussions are defined as follows:

N_X is the number of independent variables in the unit.

N_Y is the number of dependent variables in the unit.

The following restrictions hold on N_X and N_Y.

$N_X \leq 100$

$N_Y \leq 100$

2.1 OPO System Bulk Storage

OPO System Bulk Storagecontains system constants (such as vector and parameter names) and as a holding area for information which must be temporarily saved. The unit queues are also located in this section of bulk.

Fixed OPO System Bulk Storage: 256 words

Variable OPO System Bulk Storage: $\left\lceil \frac{4.U + 63}{64} \right\rceil \cdot 64$ words where U is the maximum number of units that may be defined in the system and $\lceil a \rceil$ is the integer portion of a.

Unit task queues: U·64 words

Total OPO System Bulk Storage: $256 + \left\lceil \frac{4.U + 63}{64} \right\rceil \cdot 64$
$+ 64U$ words (H-1)

2.2 Unit Common Bulk Storage Requirements

Each defined (LUCed) unit has a unit common bulk area. Each unit common bulk area consists of an area fixed in size. The variable-sized area has lower and upper bounds.

Fixed unit common bulk: 128 words

Variable unit common bulk:

Minimum size

$$\left\{ \left\lceil \frac{N_X + 63}{64} \right\rceil + \left\lceil \frac{N_Y + 63}{64} \right\rceil \right\} \cdot 64 \text{ words}$$

Maximum area

$$\left\{ \left\lceil \frac{N_X + 63}{64} \right\rceil + 20 \left\lceil \frac{2(N_X) + 63}{64} \right\rceil + \left\lceil \frac{N_Y + 63}{64} \right\rceil \right.$$
$$\left. + 22 \left\lceil \frac{2(N_Y) + 63}{64} \right\rceil \right\} \cdot 64 \text{ words}$$

Thus, a given unit common may require from

$$C = \left\{ 2 + \left\lceil \frac{N_X + 63}{64} \right\rceil + \left\lceil \frac{N_Y + 63}{64} \right\rceil \right\} \cdot 64 \text{ words} \qquad \text{(H-2)}$$

To

$$C = c + \left\{ 20 \left\lceil \frac{2(N_X) + 63}{64} \right\rceil + 22 \left\lceil \frac{2(N_Y) + 63}{64} \right\rceil \right\} \cdot 64 \text{ words} \qquad \text{(H-3)}$$

The actual size a given unit common bulk area is given by:

$$AC = C + \left\{ X_c \left\lceil \frac{2(N_X) + 63}{64} \right\rceil + Y_c \left\lceil \frac{2(N_Y) + 63}{64} \right\rceil \right\} \cdot 64 \text{ words} \qquad \text{(H-4)}$$

$0 \leq X_C \leq 20$

$0 \leq Y_C \leq 22$

where X_C and Y_C represent the number of editing vectors allocated in the unit common. X_C represents the number of editing vectors associated with independent process variables and Y_C, the number associated with dependent process variables.

2.3 ODS Bulk Storage Requirements

There may be one to nine optimizer data sets in *each* unit. A single ODS consists of a fixed and a variable area. The variable area is bounded above and below.

Fixed ODS bulk: 128 words

Variable ODS bulk: the formulas for the variable area are lengthy and will be developed in stages.

$$s = 6(N_x + 4)$$

$$t = 2(N_y + 4)$$

$$r = \left\lceil \frac{(\frac{1400-s}{t}) + 1}{2} \right\rceil$$

$$q = \left\lceil \frac{(\frac{1400-s}{t}) - 1}{2} \right\rceil$$

$$p = \left\lceil \frac{N_x - r + q - 1}{q} \right\rceil$$

$$M = \left\{ \left\lceil \frac{s+63}{64} \right\rceil + \left\lceil \frac{r \cdot t+63}{64} \right\rceil + p \left\lceil \frac{q \cdot t+63}{64} \right\rceil + \left\lceil \frac{6(N_x+4) + 63}{64} \right\rceil \right\} \cdot 64 \text{ words} \qquad \text{(H-5)}$$

(M is the bulk storage requirement for the LP matrix)

Minimum variable ODS area (c)

$$d = M + \left\{ 15 \left\lceil \frac{2(N_x) + 63}{64} \right\rceil + 13 \left\lceil \frac{2(N_y) + 63}{64} \right\rceil \right\} .64 \text{ words} \qquad \text{(H-6)}$$

Maximum ODS variable area (D)

$$D = d + \left\{ 6 \left\lceil \frac{2(N_x) + 63}{64} \right\rceil +8 \left\lceil \frac{2(N_y) + 63}{64} \right\rceil \right\} .64 \text{ words} \quad \text{(H-7)}$$

The actual size of a given ODS is:

$$\left. \begin{array}{l} AD = d + \left\{ X_D \left\lceil \frac{2(N_x) + 63}{64} \right\rceil \ +Y_D \left\lceil \frac{2(N_y) + 63}{64} \right\rceil \right\} .64 \text{ words} \\ 0 \leq X_D \leq 6 \\ 0 \leq Y_D \leq 9 \end{array} \right\} \quad \text{(H-8)}$$

where X_D and Y_D represent, respectively, the number of optional ODS vectors associated with independent variables (e.g. XDEL) and the number of optional ODS vectors associated with dependent variables (e.g YTPN) that were allocated by the user in the ODS.

2.4 Total OPO Bulk Storage Requirements

The total requirement can be a widely varying quantity and depends on:

1) The number of process units in the system

2) The number of independent, dependent, and implicit process variables defined in each unit

3) The number of optimizer data sets (ODS) allocated in each unit

4) The number of optional unit common and ODS vectors allocated by the users in the units.

Assuming the user has sufficient data at hand, equations H-1 through H-8 will allow him to determine the bulk storage requirements for any ODS, unit common, unit, or system.

APPENDIX J

HARDWARE AND SOFTWARE (MONITOR) REQUIREMENTS

APPENDIX J

HARDWARE AND SOFTWARE (MONITOR) REQUIREMENTS

Hardware

1) GE/PAC 4000 (8K core minimum, 16K core recommended)

2) I/O Typer

3) Card Reader/Punch

4) Line Printer

5) Bulk Storage Device (Disc, at least 131K)

Software (Monitor)

1) GE/PAC Monitor, I/O Typer OPR System (with calendar option).

2) Find working core storage program, FMR.

3) Run system subroutine program, RMP.

4) Six (6) consecutive functional program numbers.

5) One functional program number for each process simulation program (MODEL).

6) Double precision quasi package.

OPO assumes that the bulk storage device is read and written in blocks of 64 words.

APPENDIX K

MODEL RESULT STORAGE RULES

APPENDIX K

MODEL RESULT STORAGE RULES

When MODEL is called by OPO all communications are accomplished via a five word communications area OPOMOD in permanent core. The fourth word of this communications area contains the location, on bulk, at which MODEL must store its result.

This result storage location must be interpreted by MODEL as follows for each model type.

Type 0 - Assuming M dependent variables, MODEL must compute M+1 result values to be stored beginning at the bulk location specified at OPOMOD+3.

The first value (Y_0) is the value of the objective function. The remaining M values are dependent (constraint) variable values.

This rule applies to Model types 1 and 2 when they are called with the model type flag set to zero.

All the result values must be double precision.

YLOC+0	OBJV
+2	Y_1
+4	Y_2
.	.
.	.
+2M	Y_M

Type 1 - Assuming M dependent variables MODEL must compute M+1 result values to be stored beginning at the bulk location specified at OPOMOD+3.

The results for a type 1 MODEL are partial derivatives of the M+1 dependent variables and the objective function with respect to the independent variable specified in JMODL at OPOMOD+4.

All result values must be double precision.

YLOC+0	∂ OBJV
2	∂Y_1
4	∂Y_2
:	:
2M	∂Y_M

Type 2 - A type two MODEL may be used *only* when the whole LP Matrix will fit in core.

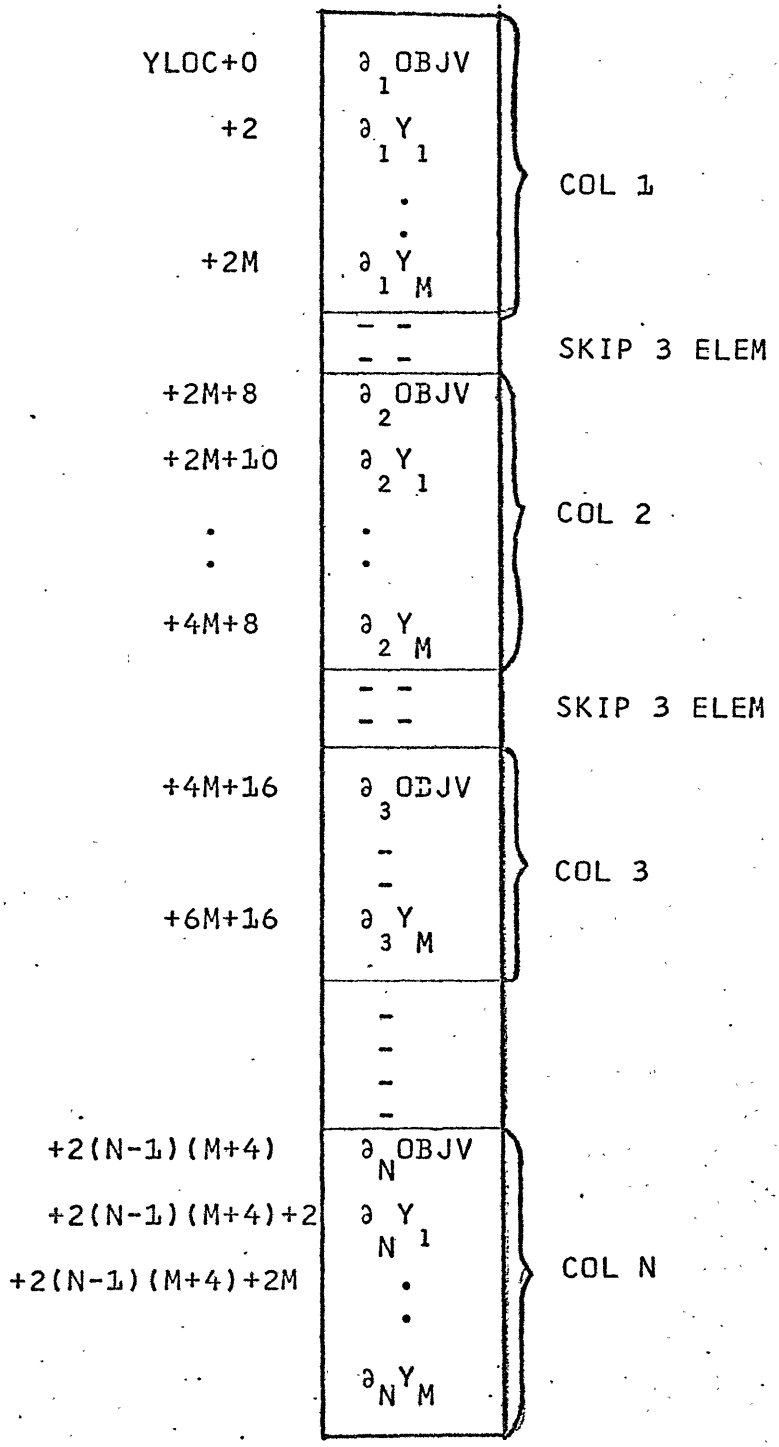
If the active part of the problem has N independent and M dependent variables, and the users OPO system has MATSIZ words available for matrix storage then the matrix will fit core if:

$$(6+2N)(4+M) \leq MATSIZ$$

A type 2 MODEL must compute N double precision vectors, each of length M+1 elements. The i th vector is the set of partial derivatives of the dependent variables with respect to the i th independent variable (refer to type 1 above).

The type 2 MODEL may compute rows and columns *only* for the variables which are active in the problem.

Given the starting location YLOC the MODEL must store the matrix as follows:



APPENDIX L

OPO SYSTEM LIMITATIONS AND RESTRICTIONS

APPENDIX L

OPO SYSTEM LIMITATIONS AND RESTRICTIONS

Problem Size

Maximum problem size is one hundred independent and one hundred dependent variables (100 × 100).

The maximum size problem that can be solved in-core is nominally (22 × 22). This can be increased by increasing the matrix work area above the standard 1400 words at assembly time.

The ratio of trade-off between dependent and independent variables is proportional to the ratio of the number of dependent variables to the number of independent variables.

$\Delta n_z / \Delta n_y = NX/NY$

Data Sets and Units

The maximum number of optimizer data sets per unit is 10.

These may be up to 5 units in the standard system. This may be modified at assembly time.

There is one task queue per unit. The number of tasks waiting in a queue is restricted to 16. This restriction may be relaxed at assembly time.

There may be one and only one control data set per unit.

Model Restrictions

A type two model may be used only when the matrix will fit in core. Refer to Appendix K.

Dependent process variables must be continuous.

Convergence in iterative procedures must be tight.

All arguments and results are double precision.

GLOSSARY

GLOSSARY

ACCEPT TASK - To insert a task into a task queue in preparation for execution of the task.

ACTIVE VARIABLES - The subset of the independent and dependent variables which is to be considered in solving a particular model.

COMMAND - A statement specifying an action to be performed on a data set.

CONTROL DATA SET - A data set which is being used to control the process with which the unit is associated. There can be only one control data set in a unit.

DEPENDENT VARIABLES - The set of variables whose value is determined by the value of the independent variables.

EDITING - A scheme by which limits are applied to quantities input to a data set. The purpose of editing is to detect possible errors.

EXTERNAL FUNCTIONAL PROGRAM - Any functional program that is not incorporated in the OPO.

FILE MAINTENANCE - The process of loading, modifying, or deleting data sets by means of a set of commands grouped under the heading of file maintenance.

IMPLICIT VARIABLES - The set of variables whose value is in part determined by itself. These variables are represented in the independent and dependent sets.

INDEPENDENT VARIABLES - The set of variables which may be changed by OPO in the optimization process.

OPTIMIZER DATA SET - Data needed to optimize process. A collection of vectors (costs, targets, etc.) plus an LP matrix.

MOVE - The process of generating a recommended plant position. A move includes a series of steps from a starting position to the optimum (recommended) position. Each step is made by solving an LP.

OPO - Online Process Optimization.

RECOMMENDED POSITION - A set of values of the independent variables which, according to the results of a solve, will yield an optimum value of the payoff.

MODEL - A user supplied functional program which computes a set of values for the dependent variables when given values of the independent variables.

SOLVE - To generate for a Unit, by means of making a move; a new recommended position. Optional reports may be printed by the solution process.

START POSITION - A set of values of the independent variables which the user wants to use as a starting point for a solve.

STEP - The alteration of the recommended position made by solving an LP on a small region of the solution space which is treated as being linear.

TARGET - A vector of values which are desirable for the independent or dependent variables.

TASK - A string of commands applying to one data set in one unit with an associated priority. Delimited by begin and end task commands.

TASK QUEUE - A list of tasks waiting to be serviced. There is a separate queue for each unit. Queues are serviced according to priority of highest priority task waiting in queue.

TEST DATA SET - An optimizer data set which is not being used to control the process. A unit in test mode.

UNIT - A set of optimize data sets which pertain to a single process (real or not) being controlled.

UNIT COMMON - A set of vectors common to all optimizer data sets in a unit including variable labels and editing vectors.

VECTOR - A set of related quantities which apply to each of the independent or dependent variables.

ALPHABETICAL INDEX