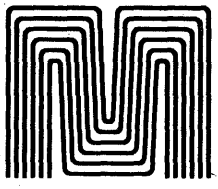
MICROPROGRAMMING HANDBOOK

Microdata



Microdata Corporation
644 East Young Street
Santa Ana, California

PRINTED IN U.S.A.

PART I

MICROPROGRAMMED COMPUTER PRIMER

PART II

APPLICATION OF THE
MICROPROGRAMMED COMPUTERS

PART III

MICRO 800 USERS MANUAL

PART IV

MICRO 810 FIRMWARE MANUAL

PART V

SYSTEM DESIGN PROCEDURES USING
MICROPROGRAMMING

PART VI

PRODUCT CATALOG

FOREWORD

This is the first and only handbook on microprogramming. It has been written and published by Microdata Corporation, the company which has pioneered the practical application of microprogramming in the minicomputer field. Its purpose is to introduce the computer user to this powerful concept, to illustrate its many clear-cut advantages in computing and control applications and to provide detailed instructions to the system designer for the most economical and efficient application of microprogramming technology.

Microdata believes, as do many other knowledgeable individuals and organizations, that the microprogrammable computer architecture will emerge as the dominant concept in the small computer area. The inevitability of microprogramming rests on fundamental advantages to both the computer manufacturer and to the computer user.

To the manufacturer, a microprogrammable architecture permits development and production of a single system of compatible hardware which can be program tailored to fit a much wider range of requirements than can be met by conventional software-oriented machines. From a design standpoint, only microprogramming permits full and extensive use of commercial MSI and LSI devices which result in higher performance for a given cost than in conventional designs. These benefits are, of course, passed directly to the user.

Microdata recognized the inherent practical advantages in the microprogramming concept for minicomputers at a very early date. Accordingly, the MICRO 800 series of computers was introduced early in 1969, and to date hundreds of these machines have been delivered. Acceptance of this product and its concept has prompted recent introduction of the MICRO 1600 series which builds on the MICRO 800 technology but which offers significant improvements in performance at a lower cost. These computers are unique in the field and offer users a set of advantages which cannot be obtained elsewhere.

This publication is offered as an aid to users and potential users of computers who, at some point, will avail themselves of microprogramming. Comments and additions by readers who wish to help expand upon the growing body of knowledge in this field are encouraged and solicited by Microdata.

TABLE OF CONTENTS

INTRODUCTION

The story of Microdata Corporation's success is the story of microprogramming, a unique element which is the secret of the significant advantages of the company's advanced minicomputers over fixed-instruction machines.

The major difference between Microdata's products and conventional minicomputers is the skillful incorporation of microprogrammed control memories as a major adjunct to the usual basic elements of any computer—control unit, main memory, arithmetic/logic unit and input/output.

The advantages are manyfold. Ease of programming using the widest possible choice of language selection is a major gain. In turn, this permits the use of low-skill (and lower salaried) programmers to operate the equipment.

Microprogramming also means higher speed with much more efficient use of the main memory of the computer.

In many cases, the storage capacity of the main memory is increased because the programs used in conventional minicomputers to perform certain operational instructions are stored in the control memory, thus freeing storage capacity in the main memory for the purpose it was intended—problem-solving.

Inherently, microprogramming gives the user unequalled flexibility in accordance with the design philosophy of Microdata Corporation. This flexibility is extremely important to the user because the computer can be tailored to his specific needs, no matter how complex or simple, and can be changed at will.

By strictly adhering to this philosophy, Microdata Corporation has set new industry standards for performance at minimum cost, unequalled memory efficiency and the availability of a wide variety of languages from which to choose. In short, Microdata has reached a pinnacle in the only meaningful measurement of computer performance—the ability to solve specific problems accurately and efficiently in terms of time and therefore cost to the user.

Microprogrammable computers also have ripped away many barriers to broader application of minicomputers. The way is clear for use of Microdata's products in business and scientific applications because of the ease and flexibility of programming techniques.

A number of factors have contributed to these advances by Microdata Corporation, including modern facilities geared to volume production, exploitation of the most advanced technologies and concepts available in the industry, and the field-proven reliability of hundreds of the company's minicomputers.

PART I

MICROPROGRAMMED COMPUTER PRIMER

INTRODUCTION

December 1945, ENIAC, the first electronic high-speed stored program general purpose computer was completed. Six years later Professor M.V. Wilkes of Cambridge University coined the word microprogramming to describe computer instructions that carry out numerous information transfers in a single execution cycle. Cost-performance improvements as a result of 25 years of advancement in computer technologies have been almost overwhelming. In 1965 it became practical and possible to build computers with control units driven by microprograms. The concept was not exploited on a widespread basis until recently. In large and medium scale computers microprogramming provides the capability to emulate other computers, and to maintain upward/downward compatibility over a wide range of models within a computer series.

The small or so called minicomputer incorporating microprogramming now exploits the advances in semiconductor and memory technologies with microprogramming far beyond the larger model computers. Full advantage of new low cost memories are realized only by users of small microprogrammed computers. The spectrum of applications between the special purpose computer, where the entire program is implemented in a microprogram, to the general purpose computer implemented by microprogram can be selected by the user to achieve a meaningful price/performance ratio for the application.

ORGANIZATION OF THE MICROPROGRAMMED COMPUTER

The organization of the microprogrammed computer can best be described after we first review the organization of its predecessor, the fixed instruction stored program general purpose computer.

The Fixed Instruction Computer

In simple terms the fixed instruction stored program computer is built around a storage and retrieval scheme, typically a magnetic core memory. The structure and information paths of a computer are represented in the simplified block diagram (Figure 1).

As defined in most textbooks, the five elements comprising a digital computer are: memory, arithmetic unit, input, output and control unit.

Memory: Modern computer memories are implemented using high speed semiconductors or magnetic core memory systems. These memories are high-speed random access devices of which information, usually in a binary form, is written or read from any addressed section of the memory.

Arithmetic Unit: In many instances is referred to as the arithmetic and logic unit (ALU). As the name implies it performs the arithmetic operations on data transferred within the computer, the memory, the input and the output.

Input/Output: Communication with a wide variety of devices in the language of the operator are made possible by transfer channels referred to as the input and output sections of a computer. Devices connected to the input/output of a computer referred to as computer peripherals include elementary switches and indicator lamps, typewriters, magnetic or paper tape units, line printers, analog converters, cathode ray tube displays (TV type devices), card readers and punches, communication lines, etc.

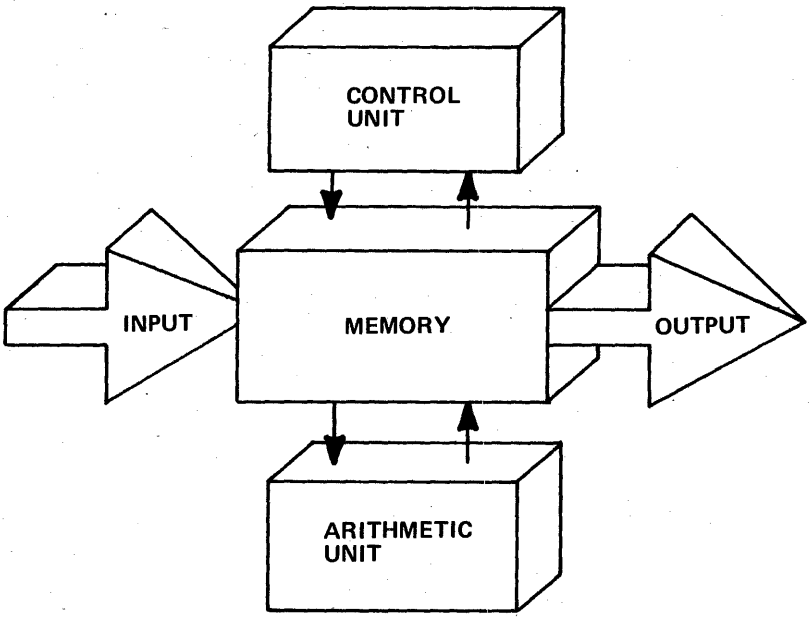


Figure 1. Simplified Block Diagram
Fixed Instruction Stored Program General Purpose Computer

In addition to man communication type devices the input/output of a computer may be connected to intermediate storage devices for mass memory requirements. Such mass memory devices include but are not limited to magnetic disc storage systems, magnetic drums, and a larger scale computer memory.

Control Unit: The control unit may be referred to as the "brain" portion of any computer because it coordinates all units of the computer in timed logical sequence. The control unit of a small fixed instruction computer receives sequences of instructions from memory. These sequences, called programs, reside in the memory and are referred to as "software." The control unit is closely synchronized to the memory cycle speed and execution time of each fixed instruction is usually a multiple of the memory speed.

The Microprogrammed Computer

Four of the elements of the microprogrammed computer are nearly identical to the fixed instruction computer. The significant difference is in the control unit ("Brain"). The basic control sequences of a microprogrammed computer originate in a separate "control memory," usually a read-only memory (ROM) which operates at speeds many times faster than the main memory section of the computer. Thus the simplified block diagram (Figure 2) of the microprogrammed computer has one more element than the fixed instruction computer.

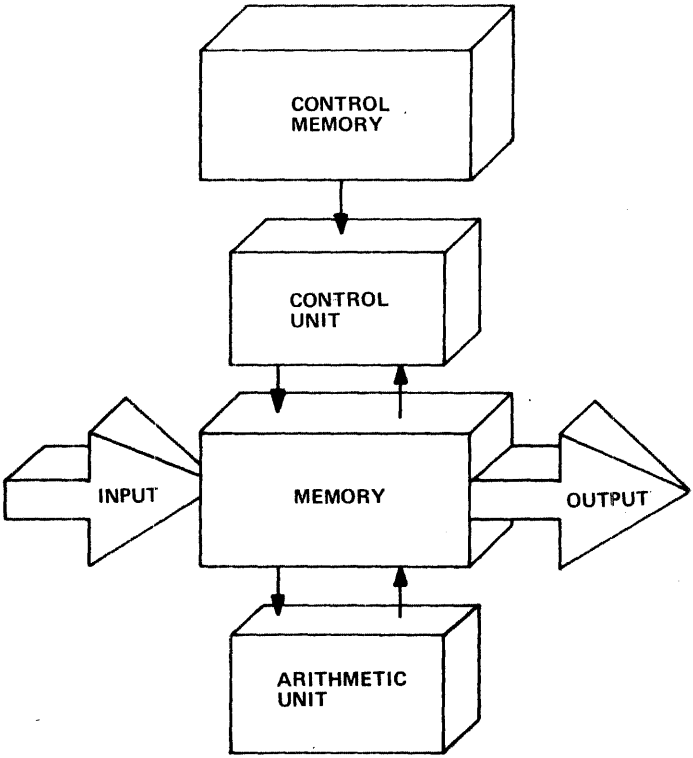


Figure 2. Simplified Block Diagram
Microprogrammed Computer

Memory: The random access main memory of the microprogrammed computer differs little from the fixed instruction computer. It is implemented with magnetic core or semiconductor systems in similar sizes and speeds to the fixed instruction computer. The basic difference is the timing and control of the memory system. The control unit of the microprogrammed computer is clocked to a significantly higher speed separate memory system. Hence, the main memory speed is essentially independent of the processor speed and is operated in a manner similar to an input/output device.

Arithmetic Unit: The arithmetic and logic unit in a microprogrammed computer operates on fixed data lengths, typically 8 bits. The speed of the unit is 10 to 50 times faster than fixed instruction computer arithmetic units operating on smaller portions of arithmetic problems at each step. Microcommands are much more intimately related to the computer architecture and to bit patterns. This allows high versatility in problem solution and minimizes the restrictions usually encountered at the software level.

Input/Output: Microprogrammed computers provide extremely fast elementary I/O capabilities. Data paths are fixed length, typically 8 bits, and the I/O control functions are simple elements sequenced by high speed control memory firmware. This permits special I/O systems to be designed for the users' requirements. The microprogrammed computer offers all of the I/O capabilities found in fixed instruction computers coupled with the unique advantage of providing only the capabilities needed, and the versatility to be changed when required.

Control Unit: The control unit of the microprogrammed computer is simple and straightforward. It operates and controls all elements of the computer system including two levels of memory. Because it is more basic than the control units in fixed instruction computers it provides capability to solve problems in an added dimension. The control unit is programmable, not fixed. Programs operating upon the control unit are called microprograms, and are referred to as firmware. These programs are as easy to write and implement as is software in the fixed instruction computer.

If we refer to the control unit of any computer as the "Brain," then the microprogrammed computer control unit could be referred to as a brain ingredient, which we can readily adjust to suit our needs.

Control Memory: The control memory is the element that most dramatically distinguishes the microprogrammed computer. The control memory contains the stored sequence of control functions that dictate end user architecture of the microprogrammed computer. These stored sequences are called "microprograms" or "firmware" corresponding to fixed instruction computer sequences called "programs" or "software."

The control memory has been called many other names including, read-only store (ROS), read-only memory (ROM) and control store. Terminology relating to the control memory of microprogrammed computers is most complex because of many misnomers coined by computer and semiconductor manufacturers. Present terminology that relates to the mechanization of control memory are:

ROM: Read-Only Memory: Any memory system in which the bit patterns of each word are fixed, and unalterable.

In application, few ROM's can be modified after manufacture. Those ROM's that can, may be called modifiable. To make any change requires a hardware modification such as adding or deleting diodes in a diode matrix ROM or rerouting of wires in a core ROM.

BROM: Bipolar Read Only Memory: Large scale integration (LSI) bipolar devices are used for volume manufacture. Original setup masking is expensive. Cost for manufactured elements is low.

PROM: Programmable Read Only Memory: A semiconductor diode array is programmed by fusing or burning out diode junctions. Cost for setup is minimal. Manufacturing cost is moderate to high. The PROM is usually used for final shake down of a system prior to investing in the BROM setup.

AROM: Alterable Read Only Memory: A true misnomer. The AROM is actually a read-write memory that is used for initial checkout of firmware. The firmware is typically loaded into the AROM via a paper tape input device. Once loaded the AROM operates the control unit as does any ROM control memory. The advantage of the AROM is programming within a few minutes rather than a manufacturing process. Cost is high; however, the devices are used indefinitely for checkout and analysis of numerous firmware implementations.

COST AND PERFORMANCE ADVANTAGES OF THE MICROPROGRAMMED COMPUTER

Fixed instruction minicomputers are basically application sensitive. Even with numerous models to choose from only a few offer good price performance for any specific application. Even more important to note is the fact that if a specific fixed instruction computer offers the best price performance for a given application at one level of complexity it may offer less relative value as the complexity changes.

Typically, to increase the performance of the fixed instruction computer the main memory (usually core memory) is increased in size.

When all the smoke settles the performance of any computer is measured by its ability to solve a specific problem within a given period of time.

For most project managers the selection of a minicomputer is a traumatic experience. He is exposed to numerous technical concepts, specifications and a variety of salesmen and skilled technicians from companies with one goal—to sell him their solution to his technical problem. If a thorough up-to-date evaluation was performed with all minicomputer manufacturers the evaluation could cost him more than the project implementation. The prime criteria for selection of the appropriate minicomputer is time and cost of implementation over the entire project life. In this light, the microprogrammed minicomputer offers an answer to this enigma. The user selects the cost/performance lines between three elements; hardware, firmware, and software for his specific application.

One of the primary purposes of this "Microprogramming Handbook", is to educate and illustrate for the user the capabilities of specific product lines and to assist these cost/performance trade-off selections.

The following comparison chart illustrates five capability levels comparing one of the more popular fixed instruction minicomputers, referred to as brand X, and a microprogrammable minicomputer, the MICRO 1600. Each level represents computer problem solving capability with corresponding notation on price, memory use and relative speed (micro vs. fixed). Within

any capability level numerous trade-offs between control memory size and core memory size can be established for the MICRO 1600.

For example, level number 4 shown in the comparison chart represents a computer capability for a time-sharing system employing high-level interpretive language and executive programs. Implementation of floating point arithmetic and executive subroutines in firmware thus expands the ROM from 768 words to 8,192 words. As a result, the MICRO 1600 cost is reduced approximately 15 percent and execution time is improved by a factor of approximately 20.

This comparison clearly illustrates that as the size of the control memory increases advantages result in price and relative speed. In addition, programming costs and implementation time can be significantly reduced once the users' needs are established in firmware. Now, with the availability of supporting systems from Microdata, firmware development is in the same dimension in price and turn-around time normally associated with fixed instruction computers. The result: computer users can benefit from microprogramming along with the computer manufacturer.

	Microprogrammed Computer (MICRO 1600)				Fixed Instruction Computer (Brand X)	
Level	Core Memory Size	Control Memory Size	System Price	Relative Speed	System Price	Core Memory Size
1.	8K X 8 4K X 8	512 X 16 1024 X 16	$5,910 $5,420	1:2 2:1	$6,250	4K X 16
2.	16K X 8 12K X 8	512 X 16 2048 X 16	$8,610 $7,690	1:2 5:1	$8,950	8K X 16
3.	32K X 8 24K X 8	512 X 16 1024 X 16	$14,010 $11,470	1:2 10:1	$14,350	16K X 16
4.	48K X 8 24K X 8	768 X 16 8192 X 16	$19,770 $16,750	2:3 15:1	$19,750	24K X 16
5.	65K X 8 32K X 8	1024 X 16 12K X 16	$25,170 $22,250	1:1 20:1	$27,000	32K X 16

THE MICRO 1600 MICROPROGRAMMABLE COMPUTER

The term microprogram, its associated terms microprogrammable and microprogrammed is used to denote programmable sub steps of general purpose processor instructions.

The MICRO 1600, however, is organized to use its basic instructions (called commands) either as sub steps of a general purpose processor instruction set, or directly for application programs. All classes of microprograms used in the MICRO 1600 are called firmware, which may be considered as a mix of hardware and software. The MICRO 1600 read only memory has a fixed hardware design except for the firmware patterns in the memory matrix. Much less original design effort is necessary for firmware in comparison to hardware since only the pattern need be checked out. With electrically-alterable read only memories and high-capacity bipolar read only memories, firmware is as flexible as software and retains the inherent speed advantage of microprogramming.

Microprogram Function Summary

Figure 3 illustrates the basic functional MICRO 1600 units and their interrelation in the processor. There is no direct one-to-one correspondence between the functions in Figure 3 and the hardware implementation in the MICRO 1600 because some of the functional elements are dispersed on more than one board. All of the essential data and control paths are shown, with data shown as solid lines and control as broken lines. No data passes through the control portion of the computer.

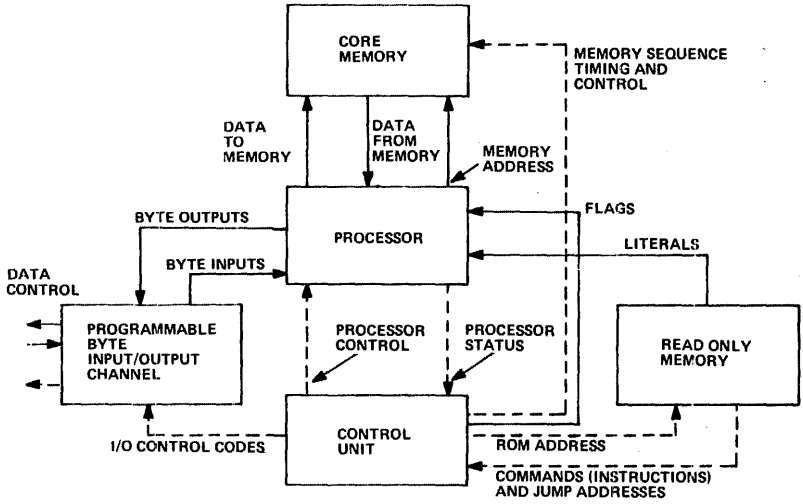


Figure 3. Functional Block Diagram of the MICRO 1600

Processor

The basic processor functions are as follows:

- Arithmetic (Add, Subtract).
- Logical ("OR," "Exclusive OR," "AND").
- Shift.
- Load Registers With Literals from ROM.
- Load or Add to Files With Literals From ROM.
- Transfer Data to and from Core Memory.
- Transfer Data to and From Byte I/O.
- Compare Data in Files With Literals from ROM.
- Provide and Update Address Value to Core Memory.

The processor consists of the following basic functional elements:

- Arithmetic/Logic Unit.
- File Registers.
- Core Memory Address Registers.
- Operand Register.
- Memory Buffer Register.
- I/O Register.
- Interconnecting Logic.

The processor is set up to do its various functions by the control unit. It provides the control unit directly with zero, negative and overflow condition status. Other status functions are tested using compare commands of bit test with literal commands.

Control Unit

The basic control functions are as follows:

- Processor Command Decoding and Control.
- Data Steering:
  - Files to Arithmetic/Logic Unit (ALU).
  - Input to ALU.
  - Operand Register to ALU.
  - Input to ALU.
  - Literals From ROM to Files or Registers.
  - Memory to Processor.
  - ALU to Files and Registers.
- Instruction Skipping Based on Processor Conditions.
- Advancing ROM Addresses.
- Jumping to ROM Addresses.
- Fetching and Holding Commands from ROM.
- I/O Control Code Generation.
- Core Memory Transfer Timing.
  - Full or Half Cycle.
  - Read or Write.

Command Execution

In the microprogrammable computer, the instruction fetch, decode, execute, and distribution functions are not divided into distinct, separate steps as they are in most fixed instruction computers. Instead, the various functions go on simultaneously during the time between clock pulses. Sufficient time is allowed for all functions to settle between clocks. Reading of instructions from ROM is done on a lookahead basis. The instructions are clocked into the ROM register where all other functions, such as decoding, steering, and processing are done (and results are entered into designated registers) on the next clock.

Because of this, the effective execution time for most instruction is 200 nanoseconds, and 400 nanoseconds for those involving skips or jumps because of the lookahead function.

Control Memory

The Control Memory contains 16-bit words which consist of commands, or literals. The literals are used to initialize files or registers, to add to files, for comparison test purposes, or for control memory address jumping.

Core Memory

The core memory stores 8-bit data words from the processor. Read and write cycles can be either full or half cycle. The memory address is provided by the processor. Timing pulses are provided by the control function. Data, pointers, and flags are stored in the core memory. If the microprogram is a general purposes processor implementation, then the core memory also is used to store instructions.

Programmable Byte I/O Channel

There is a high degree of flexibility in microprogramming of I/O. Data is transferred into and out of the processor under the direction of the control unit. Output data is transferred directly from the processor's output register. Input data transferred via the input bus can be directly copied into files or registers by microcode. A large number of peripheral devices can be connected to the computer and serviced one at a time through the byte I/O channel.

COMPARISON OF A MICROPROGRAMMABLE COMPUTER TO A GENERAL PURPOSE FIXED INSTRUCTION COMPUTER

In the general purpose fixed instruction computer, the instructions are stored in core memory along with data. Both instructions and data can be altered by the program. In a microprogrammable computer, the instructions are stored in a read only memory along with permanent (or constant) data. Only variable data, pointer, and flags are stored in core memory.

Instruction Repertoire

In the general purpose fixed instruction computer there is usually a limited instruction repertoire with variations of instruction, and memory reference instructions having limited addressing modes.

In the microprogrammable computer there is usually a smaller number of instructions which are more compact and specialized than the fixed instructional computer. Memory addressing and I/O functions usually are built up by assembling a group of micro instructions. The micro instructions are closely related to the internal architecture and I/O structure of the basic computer.

Instruction Speeds

Microprogrammable computers are faster than fixed instruction computers for the following reasons:

1. Instruction execution times are from 5 to 30 times faster in a microprogrammed computer.
2. File registers can be used for data storage, and pointers, where core is required in a fixed instruction computer, thus program execution time can be sped up by avoiding memory access cycles.
3. Subroutines are closely tailored to specific requirements and data word lenghts, thus improving computer efficiency and speed.
4. Input/output routines can be simplified for the application to increase I/O speed.
5. Special time-consuming algorithms (math, logic, etc.), which are not available in the general purpose processor can be easily incorporated into a microprogrammed processor.

Additional comparisons between a general purpose processor and a microprogrammable processor are included in Table 1.

Table 1. Comparison of Microprogrammed Computer to General Purpose Software Programmed Computer

Function	General Purpose	Microprogrammed MICRO 1600
Arithmetic and logic operations	• memory reference/ register reference	• register reference
	• conditions automatically set	• conditions set when enabled
	• usually 12 or 16 bits	• 8 bits
	• specific registers are used	• general purpose file registers
	• execution time 2-10 microseconds	• 200 nanoseconds
Shift Operations	• multiple bits at a time	• single bit at a time
	• left/right	• left/right
	• limited types of shift	• unlimited types of shift
	• usually 16 bits	• 8 bits
	• specific registers only	• any file registers
Conditional Skips	• forward/reverse	• forward
	• to multiply locations	• to one location
	• fixed registers used and tested	• any file register can be tested
	• program conditions tested	• basic conditions tested
Jumps/Return Jumps	• programmable locations	• programmable locations
	• return jump, automatic address set up	• set up return jump address with microcode
Memory Accesses	• referred to as part of Memory Reference Instruction	• set up memory address registers, initiate transfer in microcode
	• address in instruction	• address in any file register
Memory Addressing	• 16K to 65K Bytes core memory	• 65K Bytes core memory
	• control-fixed	• control variable, ROM; 256 x 16 expandable to 16,386 x 16
I/O	• instruction designates destination and source	• data transfer and timing controlled by microcode
Interrupts	• automatic hardware function	• microcode test, and handling
Concurrent I/O	• optional, referred to as direct multiplex channel or 3 cycle data break	• implemented directly in microcode
DMA	• external memory access	• external memory access
Indexing	• specific register(s) assigned	• index in any file register
Program	• software	• firmware
Execution Time	• microseconds	• nanoseconds

GLOSSARY

A

ACCESS, IMMEDIATE — Ability to obtain data from or place data in a storage device, or register directly without serial delay, usually in a relatively short time.

ACCESS, PARALLEL — Obtaining data from or placing data into storage where time required is dependent on simultaneous transfer of all elements of a word from a given location.

ACCESS, RANDOM — (1) Obtaining data from or placing data into storage where time required is independent of location of information most recently obtained or stored; (2) device in which random access, as defined in defintiion 1, can be achieved without time penalty.

ACCESS, SERIAL — Obtaining data from or placing data into storage where time required is dependent on necessity for waiting while nondesired storage locations are processed.

ACCUMULATOR — (1) Register and associated equipment in arithmetic unit of computer in which arithmetical and logical operations are performed; (2) unit in a digital computer where numbers are accumulated. Often the accumulator stores one operand and on receipt of any second operand, it forms and stores result.

ACCURACY — Degree of exactness of an approximation or measurement. Accuracy normally denotes absolute quality of computed results; precision refers to the amount of detail used in representing those results.

ADDER — Device which forms, as output, the sum of two or more numbers presented as inputs. Often no data retention feature is included; the output signal remains only as long as the input signals are present.

ADDRESS — (1) Identification, represented by a name, label, or number, for registers or location in storage. Addresses are also a part of an instruction word along with commands, tags, and other symbols; (2) part of an instruction which specifies an operand.

ADDRESS, ABSOLUTE — Address which indicates exact storage location where the referenced operand is to be found or stored in the actual machine code address numbering system.

ADDRESS, BASE — (1) Number which appears as an address in a computer instruction, but which serves as base, index, initial or starting point for subsequent addresses to be modified; (2) number used in symbolic coding in conjunction with relative address.

ADDRESS, DIRECT — Address which indicates the location where referenced operand is to be found or stored with no reference to index register or B-Box.

ADDRESS, EFFECTIVE — (1) Modified address; (2) address actually considered to be used in particular execution of computer instruction.

ADDRESS, IMMEDIATE — Instruction address in which address part of instruction is operand.

ADDRESS, INDEXED — Address that is to be modified or has been modified by index register or similar device.

ADDRESS, INDIRECT — Address in computer instruction which indicates location of address of referenced operand.

ADDRESS PART — Part of instruction word that defines address of register or location.

ADDRESS, RELATIVE — Address to which base address must be added to find machine address.

ADDRESS, SYMBOLIC — Label, alphabetic or alphameric, used to specify storage location in context of a particular program. Programs are often first written using a symbolic address in some convenient code, which are translated into absolute addresses by assembly program.

ADDRESS, VARIABLE — See address, indexed.

ADP — Automatic Data Processing.

ALGEBRA, BOOLEAN — Process of reasoning or deductive system of theorems using symbolic logic, and dealing with classes, propositions, or on-off circuit elements. It employs symbols to represent operators such as AND, OR, NOT, EXCEPT, IF . . . THEN, etc., to permit mathematical calculation. (Named for George Boole, English mathematician [1815-1864]).

ALGOL — ALGOrithmic Language. See language, algorithmic.

ALGORITHMIC — Constructive calculating process usually assumed to lead to solution of problem in finite number of steps.

ALLOCATION, STORAGE — Process of reserving blocks of storage to specified blocks of information.

ALPHAMERIC — Contraction of alphanumeric and alphabetic-numeric. Characters which include letters of the alphabet, numerals, and other sucy symbols as punctuation or mathematical symbols.

ALU — Arithmetic and Logical Unit.

ANALOG — Representation of numerical quantities by means of physical variables: translation, rotation, voltage, or resistance. Contrasted with digital.

ANALYSIS, NUMERICAL — Study of methods of obtaining useful quantitative solutions to mathematical problems, regardless of whether an analytic solution exists, and study of errors and bounds on errors in obtaining such solutions.

ANALYSIS, SYSTEMS — Examination of an activity, procedure, method, technique, or business to determine what must be accomplished and how necessary operations may best be accomplished.

ANALYST — Person skilled in definition and development of techniques for solving problems; especially those techniques for solutions on computer.

ANALYZER — Computer routine to analyze program written for the same or a different computer. Computer (usually analog) designed and used primarily for solving many types of different equations.

APPLICATION — System or problem to which a computer is applied. Reference is often made to an application as being either computational type, wherein arithmetic computations predominate, or data processing type, wherein data handling operations predominate.

ARGUMENT — (1) Independent variable: in looking up quantity in a table, number or any numbers which identify location of desired value; or in mathematical function, variable which when certain value is substituted for it, value of function is determined; (2) operand in an operation on one or more variables.

ARITHMETIC, FLOATING POINT — Calculation which automatically accounts for location of radix point. Usually accomplished by handling number as signed mantissa times radix raised to an integral exponent.

ARITHMETIC SECTION — See unit, arithmetic.

AROM — Electrically Alterable Read Only Memory.

ASSEMBLE — (1) To integrate subroutines that are supplied, selected, or generated into main routine, by means of preset parameters, by adapting, or changing relative and symbolic addresses to absolute form, or by placing them in storage; (2) to operate, or perform functions of an assembler.

ASSEMBLER — Computer program which operates on symbolic input data to produce machine instructions by carrying out such functions as: translation of symbolic operation codes into computer operating instructions; assigning locations in storage for successive instructions; or computation of absolute addresses from symbolic addresses. An assembler generally translates input symbolic codes into machine instructions item for item, and produces as output the same number of instructions or constants which were defined in the input symbolic codes.

ASYNCHRONOUS — Lack of time coincidence in set of repeated events where the term is applied to computer to indicate that execution of one operation is dependent on a signal that previous operation is completed.

ATLAS — Abbreviated Test Language for Avionics Systems.

AUTOMATION — (1) Implementation of processes by automatic means; (2) theory, art, or technique of making a process more automatic; (3) investigation, design, development, application of methods of rendering processes automatic, self-moving, or self-controlling.

B

BASIC — Beginner's All-purpose Symbolic Instruction Codes. A simple, easy to learn, machine independent, conversational computer language.

BAUD — (1) Unit of signalling speed equal to number of code elements per second; (2) unit of signalling speed equal to twice the number of Morse code dots continuously sent per second.

BINARY — Characteristic, property, or condition in which there are but two possible alternatives: binary number system using 2 as its base and using only digits zero and one.

BIT — (1) Abbreviation of binary digit; (2) single character in binary number; (3) single pulse in group of pulses; (4) unit of information capacity of a storage device. Capacity in bits is the logarithm to the base two of the number of possible states of the device.

BIT, PARITY — Check bit that indicates whether total number of binary "1" digits in a character or word (excluding parity bit) is odd or even. If a "1" parity bit indicates an odd number of "1" digits, then a "0" bit indicates an even number. If total number of "1" bits, including parity bit, is always even, system is called an even parity system. In an odd parity system, total number of "1" bits, including parity bit, is always odd.

BLOCK — (1) Group of computer words considered as a unit by virtue of their being stored in successive storage locations; (2) set of locations or tape positions in which a block of words is stored or recorded; (3) circuit assemblage which functions as a unit: circuit building block of standard design, and logic block in sequential circuit.

BOOTSTRAP — Technique for loading first instructions of a routine into storage; then using these instructions to bring in the rest of the routine; usually involves either entering of a few instructions manually or use of a special console key.

BRANCH — Selection of one, two, or more possible paths in flow of control based on some criterion. Instructions which mechanize this concept are sometimes called branch instructions, but the terms transfer of control and jump are more widely used.

BRANCHPOINT — Point in a routine where one of two or more choices is selected under control of routine.

BREAKPOINT — Point in computer program at which conditional interruption, to permit visual check, printing out, or other analysis. Breakpoints are usually used in debugging operations.

BROM — Bipolar Read Only Memory.

BUFFER — (1) Internal portion of data processing system serving as intermediary storage between two storage or data handling systems with different access times or formats; usually to connect an input or output device with main or internal high-speed storage; (2) logical OR circuit; (3) an isolating component designed to eliminate reaction of a driven circuit on circuits driving it: buffer amplifier; (4) diode.

BUS — (1) Circuit over which data or power is transmitted, often one which acts as a common connection among a number of locations; (2) communications path between two switching points.

BYTE — (1) Generic term to indicate measurable portion of consecutive binary digits: an 8-bit or 6-bit byte; (2) group of binary digits usually operated upon as a unit.

C

CAPACITY, CHANNEL — (1) Maximum number of binary digits or elementary digits to other bases which can be handled in a particular channel per unit time; (2) maximum possible information transmission rate through channel at specified error rate. Channel capacity may be measured in bits per second or bauds.

CAPACITY, STORAGE — Number of elementary pieces of data that can be contained in storage device. Frequently defined in terms of characters in a particular code or words of fixed size.

CARD, PUNCH — Heavy stiff paper of constant size and shape, suitable for punching in a pattern that has meaning and that can be handled mechanically. Punched holes are sensed electrically by wire brushes, mechanically by metal fingers, or photoelectrically by photocells.

CARRY — (1) Signal, or expression, produced as result of arithmetic operation on one digit place of two or more numbers expressed in positional notation and transferred to next higher place for processing there; (2) signal or expression as defined above which arises in adding, when the sum of two digits in the same digit place equals or exceeds base of the number system in use. If a carry-into-a-digit place will result in a carry-out of the same digit place, and if the normal adding circuit is bypassed when generating this new carry, it is called a high speed carry, or "standing on nines" carry. If the normal adding circuit is used in such a case, the carry is called a cascaded carry. If a carry resulting from the addition of carries is not allowed to propagate (when forming the partial product in one step of a multiplication process) process is called a partial carry. If it is allowed to propagate, the process is called a complete carry. If a carry generated in the most significant digit place is sent directly to least significant place (when adding two negative numbers using nine complements) that carry is called an end-around carry; (3) signal or expression in direct subtraction, as defined in (1) above which arises when the difference between the digits is less than zero. Such a carry is frequently called a borrow; (4) action of forwarding a carry; (5) command directing a carry to be forwarded.

CELL — (1) Storage for one unit of information, usually one character or one word; (2) location specified by whole or part of address and possessed of the faculty of store. Specific terms such as column, field, location, and block are preferable when appropriate.

CHAD — Small piece of paper tape or punch card removed when punching a hole to represent information.

CHADLESS — Type of punching of paper tape in which each chad is left fastened by about a quarter of the circumference of the hole, at the leading edge. This

mode of punching is useful where it is undesirable to destroy information written or printed on punched tape or it is undesirable to produce chads. Chadless punched paper tape must be sensed by mechanical fingers, for the presence of chad in the tape would interfere with reliable electrical or photoelectric reading of the paper tape.

CHAIN – (1) Any series of items linked together; (2) routine consisting of segments which are run through computer in tandem, only one being within computer at any one time and each using output from previous program as its input.

CHANNEL – (1) Path along which information, particularly a series of digits or characters, may flow; (2) one or more parallel tracks treated as a unit; (3) in a circulating storage, a channel is one recirculating path containing fixed number of words stored serially by word; (4) path for electrical communication; (5) band of frequencies used for communication.

CHARACTER – (1) One symbol of a set of elementary symbols such as those corresponding to typewriter keys. Symbols usually include decimal digits 0 through 9, letters A through Z, punctuation marks, operation symbols, and any other single symbols which computer may read, store, or write; (2) electrical, magnetic, or mechanical profile used to represent character in a computer, and its various storage and peripheral devices. Character may be represented by a group of other elementary marks, such as bits or pulses.

CHARACTER, BINARY CODED – One element of a notation system representing alphameric character such as deciminal digits, alphabetic letters, and punctuation marks by predetermined configuration of consecutive binary digits.

CHARACTER, ILLEGAL – Character or combination of bits which is not accepted as a valid representation by the machine design or by a specific routine. Illegal characters are commonly detected and used as an indication of machine malfunction.

CHARACTER, REDUNDANT – Character specifically added to a group of characters to ensure conformity with certain rules which can be used to detect computer malfunction.

CHART, FLOW – Graphic representation of the major steps of work in process. Illustrative symbols may represent documents, machines, or actions taken during process. The area of concentration is on where or who does what rather than how it is to be done.

CHART, LOGICAL FLOW – Detailed solution of work order in terms of the logic, or built-in operations and characteristics, of a specific machine. Concise symbolic notation is used to represent information and describe input, output, arithmetic, and logical operations involved. Chart indicates types of operations by use of a standard set of block symbols. Coding process normally follows the logical flow chart.

CHECK – Process of partial or complete testing of the correctness of machine operations, the existence of certain prescribed conditions within the computer, or the correctness of the results produced by a program. A check of any of these conditions may be made automatically by the equipment or may be programmed.

CHECK, PARITY – Summation check in which binary digits, in character or word, are added, modulo 2, and the sum checked against a single, previously computed parity digit: a check which tests whether number of ones in a word is odd or even.

CHECK-SUM – Check in which groups of digits are summed, usually without regard for overflow, and that sum checked against a previously computed sum to verify that no digits have been changed since the last summation.

CHECK, VALIDITY – Check based on known limits or on given information or computer results: a calendar month will not be numbered greater than 12; a week does not have more than 168 hours.

CIRCUIT – (1) System of conductors and related electrical elements through which electrical current flows; (2) communications link between two or more points.

CLEAR – To erase the contents of storage device by replacing the contents with blanks, or zeros.

CLOCK, REAL TIME – Clock which indicates passage of actual time, in contrast to a fictitious time set up by the computer program, such as elapsed time in the flight of a missile, wherein a 60-second trajectory is computed in 200 actual milliseconds, or a 0.1 second interval is integrated in 100 actual microseconds.

COBOL – Common Business Oriented Language.

CODE – (1) System of symbols for meaningful communication; (2) system of symbols for representing data or instructions in a computer or tabulating machine; (3) to translate program for the solution of a problem on a given computer into a sequence of machine language or pseudo instructions and addresses acceptable to that computer; (4) machine language program.

CODE, BINARY – (1) Coding system in which encoding of any data is done through use of bits, 0 or 1; (2) a code for the ten decimal digits, 0 through 9, in which each is represented by its binary, radix 2, equivalent: straight binary.

CODE, COMPUTER – (1) System of combinations of binary digits used by a given computer; (2) repertoire of instructions.

CODE, ERROR CORRECTING – Error-detecting code in which forbidden pulse combination produced by gain or loss of a bit indicates which bit is wrong.

CODE, ERROR DETECTING – Code in which errors produce forbidden combinations. A single error-detecting code produces a forbidden combination if a digit gains or loses a single bit. A double error-detecting code produces a forbidden combination if digit gains or loses either one or two bits.

CODE, INSTRUCTION – List of symbols, names, and definitions of instructions which are intelligible to a given computer or computing system.

CODE, MICRO -- (1) System of coding making use of suboperations not ordinarily accessible in programming: coding that makes use of parts of multiplication or division operations; (2) list of small program steps. Combinations of these steps, performed automatically in a prescribed sequence from a macro-operation (multiply, divide, and square root).

CODE, STRAIGHT LINE – Repetition of sequence of instructions, with or without address modification, by explicitly writing instructions for each repetition. Generally straight line coding will require less execution time and more space than equivalent loop coding. If number of repetitions is large, this type of coding is tedious unless a generator is used. Feasibility of straight line coding is limited by required space and difficulty of coding a variable number of repetitions.

CODE, SYMBOLIC – Code which expresses programs in source language: by referring to storage locations and machine operations by symbolic names and addresses which are independent of their hardware determined names and addresses.

CODING – Ordered list in computer code or pseudo code, of successive computer instructions representing successive computer operations for solving a specific problem.

COLLATE – To merge two or more ordered sets of data or cards to produce one or more ordered sets that still reflect the original ordering relations. The collation process is the merging of two sequences of cards, each ordered on some mutual key, into a single sequence ordered on the same key.

COLUMN – (1) Character or digit position in a positional information format, particularly one in which characters appear in rows, and rows are placed one above another: the rightmost column in a five decimal place table, or in a list of data; (2) character or digit position in a physical device, such as punch card or a register, corresponding to a position in a written table or list: the rightmost place in a register; or the third column in an eighty column punch card.

COMMAND – (1) Electronic pulse, signal, or set of signals to start, stop, or continue some operation. It is incorrect to use command as a synonym for instruction; (2) portion of an instruction word which specifies operation to be performed.

COMMENT – Expression which explains or identifies a particular step in a routine, but which has no effect on the operation of the computer in performing instructions for the routine.

COMPARE – To examine representation of a quantity to discover its relationship to zero, or to examine two quantities usually for the purposes of discovering identity or relative magnitude.

COMPATIBILITY, EQUIPMENT – Characteristic of computers by which one computer may accept and process data prepared by another computer without conversion or code modification.

COMPILE – To produce a machine language routine from a routine written in source language by selecting appropriate subroutines from a subroutine library, as directed by the instructions or other symbols of the original routine, supplying the linkage which combines the subroutines into a workable routine and translating the subroutines and linkage into machine language. The compiled routine is then ready to be loaded into storage and run: the compiler does not usually run the routine it produces.

COMPILER – Computer program more powerful than an assembler. In addition to its translating function which is generally the same process as that used in an assembler, it is able to replace items of input with series of instructions (subroutines). Thus, where an assembler translates item for item, and produces as output the same number of instructions or constants which were put into it, a compiler will do more. Program which results from compiling is a translated and expanded version of the original.

COMPLEMENT – (1) Quantity expressed to the base N, which is derived from a given quantity by a particular rule; frequently used to represent the negative of the given quantity; (2) a complement on N, obtained by subtracting each digit of the given quantity from N-1, adding unity to the least significant digit, and performing all resultant carrys: the twos complement of binary 11010 is 00110; the tens complement of decimal 456 is 544; (3) a complement of N-1, obtained by subtracting each digit of the given quantity from N-1: the ones complement of binary 11010 is 00101; the nines complement of decimal 456 is 543.

COMPUTER – Device capable of accepting information, applying prescribed processes to that information, and supplying the results of these processes. It usually consists of input and output devices, storage, arithmetic, and logical units, and a control unit.

COMPUTER, ANALOG – Computer which represents variables by physical analogies. Any computer which solves problems by translating physical conditions such as flow, temperature, pressure, angular position, or voltage into related mechanical or electrical quantities and uses mechanical or electrical equivalent circuits as an analog for the physical phenomenon being investigated. Computer which generally uses an analog for each variable and produces analogs as output. Thus an analog computer measures continuously whereas a digital computer counts discretely.

COMPUTER, DIGITAL – Computer which processes information represented by combinations of discrete or discontinuous data as compared with an analog computer for continuous data. A device for performing sequences of arithmetic and logical operations, not only on data but its own program. A stored

program digital computer capable of performing sequences of internally stored instructions, as opposed to such calculators as card-programmed calculators, on which the sequence is impressed manually.

COMPUTER, FIXED PROGRAM — Computer in which the sequence of instructions are permanently stored or wired, and performs automatically. Not subject to change either by the computer or the programmer except by rewiring or changing the storage input.

COMPUTER, GENERAL PURPOSE — Computer designed to solve a large variety of problems: a stored program computer which may be adapted to any of a very large class of applications.

COMPUTER, SOLID STATE — Computer built primarily from solid state electronic circuit elements.

COMPUTER, SPECIAL PURPOSE — Computer designed to solve a specific class or narrow range of problems.

COMPUTER, STORED PROGRAM — Computer capable of performing sequences of internally stored instructions, usually capable of modifying those instructions as directed by the instructions.

COMPUTER, WIRED PROGRAM — Computer in which instructions that specify the operations are specified by the placement and interconnection of wires. Wires are usually held by a removable control panel, allowing flexibility of operation, but the term is also applied to permanently wired machines which are then called fixed program computers.

CONDITIONAL TRANSFER OF CONTROL — Computer instruction which when reached in a program will cause the computer either to continue with the next instruction in the original sequence or to transfer control to another stated instruction, depending on a condition regarding some property of numbers which has then been determined.

CONFIGURATION — Group of machines which are interconnected and are programmed to operate as a system.

CONJUNCTION — Logical operation which makes use of the AND operator or logical product.

CONSOLE — Portion of the computer which may be used to control the machine manually, correct errors, determine the status of machine circuits, registers and counters, determine contents of storage, and manually revise storage contents.

CONSTANT(S) — Quantities or messages present in the machine and available as data for the program and which usually are not subject to change.

CONTENT(S) — Data contained in any storage medium. Quite prevalently, the symbol ( ) is used to indicate the contents of: (M) indicates the contents of the storage location whose address is M; or (T_2) may indicate the contents of the tape on input-output unit two.

CONTROL — (1) Part of a digital computer or processor which determines the execution and interpretation of instructions in proper sequence, including decoding of each instruction and application of the proper signals to the arithmetic unit and other registers in accordance with the decoded information; (2) one or more of the components in any mechanism responsible for interpreting and carrying out manually-initiated directions. Sometimes it is called manual control; (3) in some business applications, a mathematical check; (4) in programming, instructions which determine conditional jumps are often referred to as control instructions; time sequence of execution of instructions is called the flow of control.

CONTROL, MANUAL — Direction of a computer by means of manually operated switches.

CONTROL, MASTER — Application-oriented routine usually applied to the highest level of a subroutine hierarchy.

CONTROL, NUMERICAL — Descriptive of systems in which digital computers are used for the control of operations, particularly of automatic machines wherein the operation control is applied at discrete points in the operation or process.

CONTROL, PROGRAM — Descriptive of system in which a computer is used to direct an operation or process and automatically hold or make changes in the operation or process on the basis of a prescribed sequence of events.

CONVERSION — (1) Process of changing information from one form of representation to another, such as from the language of one type of machine to that of another or from tape to print; (2) process of changing from one data processing method to another, or from one type of equipment to another: conversion from punch card equipment to magnetic tape equipment.

CONVERSION, BINARY TO DECIMAL — Process of converting a number written to base of two to the equivalent number written to base of ten.

CONVERSION, DECIMAL TO BINARY — Process of converting a number written to base of ten, or decimal, into the equivalent number written to base of two, or binary.

CONVERT — (1) To change numerical information from one number base to another; (2) to transfer information from one recorded medium to another.

CONVERTER — Device which converts representation of information, or which permits changing the method for data processing from one form to another: a unit which accepts information from punch cards and records the information on magnetic tape, possibly including editing facilities.

COPY — To reproduce information in a new location, replacing whatever was previously stored there, usually leaving information unchanged at the original location.

COPY, HARD — A printed copy of machine output: printed reports, listings, documents, summaries.

COUNTER — Device, register, or location in storage for storing numbers or number representations which permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary value.

COUNTER, PROGRAM — Register which holds the identification of the instruction word to be executed next in time sequence, following present operation. Register often a counter which is incremented to the address of the next sequential storage location, unless transfer or other special instruction is specified by the program.

CPU — Central Processing Unit.

CROSS ASSEMBLER — A symbolic language translator that operates on one type of computer to produce machine code for another type of computer.

CROSSTALK — (1) Unwanted signals in a channel which originate from one or more other channels in the same communication system; (2) signals electrically coupled from another circuit, usually undesirably, but sometimes useful.

CYBERNETICS — Technology involved in the comparative study of the control and intracommunication of information-handling machines and nervous systems of animals and man to understand and improve communication.

CYCLE — (1) Same as loop (1); (2) a nonarithmetic shift in which digits dropped off at one end of a word are returned at the other end in circular fashion: cycle left and cycle right; (3) to repeat a set of operations indefinitely or until a

stated condition is met. The set of operations may be subject to variation on each repetition, as by address changes obtained by programmed computation or by use of devices such as an index register; (4) occurrence, phenomena, or interval of space or time that recurs regularly and in the same sequence: the interval required for completion of one operation in a repetitive sequence of operations.

CYCLE, STORAGE — (1) Periodic sequence of events occurring when information is transferred to or from the storage device of a computer; (2) storing, sensing, and regeneration form parts of storage sequence.

D

DATA — General term denoting any or all facts, numbers, letters, and symbols, or facts that refer to or describe an object, idea, condition, situation, or other factors. Connotes basic elements of information which can be processed or produced by a computer. Sometimes data is considered to be expressible only in numerical form, but information is not so limited.

DATA, RAW -- Data which has not been processed. Such data may or may not be in machine-sensible form.

DATA-REDUCTION — Process of transforming masses of raw data, usually gathered by automatic recording equipment, into useful, condensed, or simplified intelligence.

DATA-REDUCTION, ON-LINE — Processing of information as rapidly as the information is received by the computing system or as rapidly as it is generated by the source.

DECADE — Group or assembly of ten units: a counter which counts to ten in one column or a resistor box which inserts resistance quantities in multiples of powers of 10.

DECIMAL, BINARY CODED — Decimal notation in which the individual decimal digits are represented by a pattern of ones and zeros: in the 8-4-2-1 coded decimal notation, the number twelve is represented as 0001 0020 for 1 and 2, respectively, whereas in pure or straight binary notation it is represented as 1100.

DECISION — Computer operation to determine if a certain relationship exists between words in storage or registers, and taking alternative courses of action, affected by conditional jumps or equivalent techniques. The process consists of making comparisons by use of arithmetic to determine the relationship of two terms (numeric, alphabetic or a combination of both): equal, greater than, or less than.

DECISION, LOGICAL — Choice or ability to choose between alternatives. Basically this amounts to an ability to answer yes or no with respect to certain fundamental questions involving equality and relative magnitude: in an inventory application, it is necessary to determine whether or not there has been an issue or a given stock item.

DECODE — (1) To apply a code to reverse some previous encoding; (2) to determine meaning of individual characters or groups of characters in a message; (3) to determine the meaning of an instruction from the set of pulses which describes the instruction, command, or operation to be performed.

DECODER — (1) Device which determines the meaning of a set of signals and initiates a computer operation based thereon; (2) matrix of switching elements which selects one or more output channels according to the combination of input signals present.

DECREMENT — (1) Quantity by which a variable is decreased; (2) specific part of an instruction word in some binary computers — a set of digits.

DEFINITION – (1) Resolution and sharpness of an image, or the extent to which an image is brought into sharp relief; (2) degree with which a communication system reproduces sound images or messages.

DELAY – (1) Time after the close of a reporting period before information pertaining to that period becomes available. Delay may also cover the time to process data and to report; (2) retardation of the flow of information in a channel for a finite period of time.

DELIMITER – A character which limits a string of characters, and therefore cannot be a member of the string.

DENSITY, CHARACTER – Number of characters stored per unit of length: on some magnetic tape drives, 800 or 1600 bits can be stored serially, linearly, and axially per inch.

DENSITY, PACKING – Number of units of useful information contained within a given linear dimension, usually expressed in units per inch: the number of binary digit magnetic pulses or number of characters stored on tape or drum per linear inch on a single track by a single head.

DESIGN, LOGICAL – (1) Planning of a data processing system before detailed entineering design; (2) synthesizing of a network of logical elements to perform a specified function; (3) result of (1) and (2), frequently called the logic of a computer or of a data processing system.

DEVICE, INPUT – Mechanical unit designed to bring data to be processed into a computer: a card reader, a tape reader, or a keyboard.

DEVICE, OUTPUT – The part of a machine which translates the electrical impulses representing data processed by the machine into permanent results such as printed forms, punched cards, and magnetic writing on tape.

DIAGRAM – (1) Schematic representation of a sequence of subroutines designed to solve a problem; (2) coarser and less symbolic representation than a flow chart, frequently including descriptions in words; (3) schematic or logical drawing showing the electrical circuit or logical arrangements within a component.

DIAGRAM, BLOCK – (1) Graphic representation of the hardware in a computer system. A block diagram indicates the paths along which information and control flows between the various parts of a computer system, not to be confused with the term flow chart; (2) coarser and less symbolic representation than a flow chart.

DICTIONARY – List of code names used in a routine or system; their intended meaning in that routine or system.

DIGIT – Sign or symbol used to convey a specific quantity of information either by itself or with other numbers of its set; 2, 3, 4, and 5 are digits; the base or radix must be specified and each digit's value assigned.

DIGITAL – Pertaining to utilization of discrete integral numbers in a given base to represent all the quantities that occur in a problem or calculation. It is possible to express in digital form all information stored, transferred, or processed by a dual state condition: on-off, open-closed, and true-false.

DIRECTORY – File containing the layout for each field of the described record. A directory describes the layout of a record within a file.

DISK, MAGNETIC – Storage device on which information is recorded on the magnetizable surface of a rotating disk. A magnetic disk storage system is an array of such devices, with associated reading and writing heads which are mounted on movable arms.

DUMP, STORAGE – Listing of contents of a storage device, or parts of it.

DUPLEX — Twin, pair, or two-in-one situation: channel providing simultaneous transmission in both directions or a second set of equipment to be used in event of failure of the primary or either device.

E

EDP — Electronic Data Processing.

ENCODE — (1) To apply a code, frequently one consisting of binary numbers, to represent individual characters or groups of characters in a message; (2) to substitute letters, numbers, or characters for other numbers, letters, or characters, usually to intentionally hide the meaning of the message except to certain individuals who know the enciphering scheme.

ENCODER — Device capable of translating from one method of expression to another method of expression; translating a message into a series of binary digits.

END OF FILE — Termination or point of completion of a quantity of data.

ENTRY — (1) Statement in a programming system. In general, each entry is usually written on one line of a coding form and punched on one card; some systems permit a single entry to overflow several cards; (2) item of a list.

EQUIPMENT, OFF-LINE — Peripheral equipment or devices not in direct communication with the central processing unit of a computer.

EQUIPMENT, ON-LINE — System and peripheral equipment or devices in which the operation of such equipment is under control of the central processing unit, in which information reflecting current activity is introduced into the data processing system as soon as it occurs, directly in-line with the main flow of transaction processing.

EQUIPMENT, PERIPHERAL — Auxiliary machines which may be placed under central computer control: card readers, card punches, magnetic tape feeds, and high-speed printers. Peripheral equipment may be used on-line or off-line depending upon computer design and job requirements.

ERROR — (1) General term referring to any deviation of a computed or a measured quantity from the theoretically correct or true value; (2) part of the error due to a particular identifiable cause: a truncation error, or a rounding error. In a restricted sense, that deviation due to unavoidable random disturbances, or to the use of finite approximations to what is defined by an infinite series; (3) amount by which the computed or measured quantity differs from the theoretically correct or true value.

ERROR, ABSOLUTE — Magnitude of the error disregarding the algebraic sign or (if a vectorial error) disregarding its direction.

ERROR, INHERITED — Error in initial values, especially the error inherited from previous steps in the step-by-step integration. This error could also be the error introduced by the inability to make exact measurements of physical quantities.

ERROR, ROUNDING — Error resulting from rounding off a quantity by deleting the less significant digits and applying some rule of correction to the part retained: 0.2751 can be rounded to 0.275 with a rounding error of .0001.

ERROR, TRUNCATION — Error resulting from the use of only a finite number of terms of an infinite series, or from approximation of operations in the infinitesimal calculus by operations in calculus of finite differences. Frequently convenient to define truncation error, by exclusion, as any error generated in computation not due to rounding, initial conditions, or mistakes. A truncation error would thus be that deviation of a computed quantity from the theoretically correct value that would be present even in the hypothetical situation in which no mistakes were made, all given data were exact, no inherited error, and infinitely many digits retained in all calculations.

EXECUTE — To interpret a machine instruction and perform the indicated operation(s) on the operand(s).

EXIT — A way of momentarily interrupting or leaving a repeated cycle of operations in a program.

EXPRESSION — Any symbol or group of symbols representing a variable, or group of variables, possibly combined by symbols representing operators to a set of definitions and rules.

F

FETCH — To obtain data from storage.

FIELD — Assigned area in a record to be marked with information.

FIELD, CONTROL — A constant location where information for control purposes is placed; e.g., in a set of punch cards, if columns 79 and 80 contain various codes which control whether or not certain operations will be performed on any particular card, then columns 79 and 80 constitute a control field.

FILE — Organized information directed toward some purpose; may or may not be sequenced according to a key contained in each record.

FLAG — (1) Bit of information attached to a character or word indicating boundary of a field; (2) indicator used to tell some later part of a program that some condition occurred earlier; (3) indicator used to identify the members of several intermixed sets.

FORTRAN — FORmula TRANslator. Programming language designed for problems which can be expressed in algebraic notation allowing for exponentiation up to three subscripts. The FORTRAN compiler is a routine for a given machine which accepts a program written in FORTRAN source language and produces a machine language routine object program. FORTRAN II added considerably to the power of the original language by giving it the ability to define and use almost unlimited hierarchies of subroutines, all sharing a common storage region if desired. Later improvements have added the use of Boolean expressions, and some capabilities for inserting symbolic machine language sequences within a source program.

G

GAP — (1) Space or time interval used as an automatic sentinel to indicate the end of a word, record, or file of data on a tape: a word gap at the end of a word, a record or item gap at the end of a group of words, a file gap at the end of a group of records or items; (2) absence of information for a specified length of time or space on a recording medium, contrasted with marks and sentinels which are the presence of specific information to achieve a similar purpose. Marks are used primarily internally in variable word length machines. Sentinels achieve similar purposes either internally or externally, but sentinels are programmed, not inherent in the hardware; (3) space between the reading or recording head and the recording medium, such as tape, drum, or disk.

GAP, RECORD — Interval of space or time associated with a record to indicate or signal the end of the record.

GATE, AND — Signal circuit with two or more input wires in which the output wire gives a signal only if all input wires receive coincident signals.

GATE, OR — Electrical gate or mechanical device which implements the logical OR operator. An output signal occurs whenever there are one or more inputs on a multi-channel input. An OR gate performs the function of the logical "inclusive OR Operator."

GENERATE – To produce or prepare a specific term in accordance with a specific and defined rule or program.

GENERATOR, PROGRAM – Program which permits a computer to write other programs automatically. Two types: 1. the character controlled generator, which operates like a compiler in that it takes entries from a library tape, but unlike a simple compiler in that it examines control characters associated with each entry, and alters instructions found in the library according to the directions contained in control characters. 2. Pure generator is a program that writes another program. When associated with an assembler, a pure generator is usually a program section called into storage by the assembler from a library tape, which then writes one or more entries in another program. Most assemblers are also compilers and generators. The entire system is usually referred to as an assembly system.

GENERATOR, RANDOM NUMBER – Machine routine or hardware designed to produce a random number or series of random numbers to specified limitations.

GENERATOR, REPORT – Technique for producing complete data processing reports giving only description of the desired content and format of output reports, and certain information concerning the input file.

H

HANDLING, DATA – Same as processing, data (2).

HARDWARE – The physical equipment or devices forming a computer and peripheral equipment.

HEAD – Device which reads, records, or erases information in a storage medium, usually a small electromagnet used to read, write or erase information on a magnetic drum or tape or the set of perforating or reading fingers and block assembly for punching or reading holes in paper tape or cards.

HOLLERITH – System of encoding alphanumeric information onto cards, synonymous with punch cards.

HOUSEKEEPING – Administrative or overhead operations necessary to maintain control of a situation: involves setting up of constants and variables to be used in the program.

HYSTERESIS – (1) Lagging in the response of a unit of a system behind an increase or a decrease in the strength of a signal; (2) phenomenom demonstrated by materials which make their behavior a function of the history of their environment.

I

IMAGE – Exact duplicate array of information or data stored in (or in transit to) a different medium.

IMAGE, CARD – Representation in storage of the holes punched in a card, so that the holes are represented by one binary digit and the unpunched spaces are represented by the other binary digit.

INDEX – Symbol or a number identifying a particular quantity in an array of similar quantities: X5 is the fifth item in an array of X's.

INDICATORS – Devices registering conditions such as high or equal conditions resulting from a computation. Sequence of operations within a procedure may be varied according to the position of an indicator.

INPUT – (1) Information or data transferred or to be transferred from an external storage medium into internal storage of the computer; (2) describing the routines which direct input as defined in (1) or the devices from which such information is available to the computer; (3) device or collective set of devices necessary for input as defined in (1).

INPUT-OUTPUT — General term for the equipment used to communicate with a computer and the data involved in the communication.

INQUIRY — Technique whereby the interrogation of computer storage may be initiated at a keyboard.

INSTRUCTION — (1) Set of characters which defines an operation together with one or more addresses, or no address, and which, as a unit, causes the computer to perform the operation on the indicated quantities. "Instruction" is preferable to the terms "command" and "order"; "command" is reserved for a specific portion of the instruction word: the part which specifies the operation which is to be performed. Order is reserved for the ordering of the characters, implying sequence, or the order of the interpolation, or the order of the differential equation; (2) the operation or command to be executed by a computer, together with associated addresses, tags and indices.

INSTRUCTION, MACRO — (1) Instruction consisting of a sequence of micro instructions inserted into the object routine for performing a specific operation; (2) more powerful instructions which combine several operations in one instruction.

INSTRUCTION, MICRO — Small, single, short, add, shift or delete type of command.

INSTRUCTION, SYMBOLIC — Instruction in assembly language directly translatable into a machine code.

INTELLIGENCE, ARTIFICIAL — Study of computer techniques to supplement human capabilities. As man has invented and used tools to increase his physical powers, he now is beginning to use artificial intelligence to increase his mental powers. In a more restricted sense, the study of techniques for more effective use of digital computers by improved programming techniques.

INTERFACE — Common boundary between automatic data processing systems or parts of a single system.

INTERLACE — To assign successive storage locations: on a magnetic drum, usually to reduce access time.

INTERPRETER — (1) Punch card machine which will take a punch card with no printing on it, read the information in the punched holes, and print a translation in characters in specified rows and columns; (2) executive routine which as computation progresses translates a stored program expressed in machine-like pseudo code into machine code and performs indicated operations, by subroutines, as translated. An interpreter is essentially a closed subroutine which operates successively on an indefinitely long sequence of program parameters, the pseudo instructions, and operands. It may usually be entered as a closed subroutine and left by a pseudo-code exit instruction.

INTERRUPT — To temporarily disrupt the normal operation of a routine by a special signal from the computer. Normal operation can normally be resumed from that point later.

ITEM — (1) Set of one or more fields containing related information; (2) unit of correlated information relating to a single person or object; (3) contents of a single message.

ITERATIVE — Procedure or process which repeatedly executes a series of operations until some condition is satisfied. Can be implemented by a loop in routine.

J

JAM, CARD — A pile-up of cards in a machine.

K

KEY – (1) A group of characters which identifies or is part of a record or item; any entry in a record or item can be used as a key for collating or sorting; (2) marked lever manually operated for copying a character: a typewriter, paper tape perforator, card punch, manual keyboard, digitizer or manual word generator; (3) lever or switch on a computer console for manually altering computer action.

KEYPUNCH – (1) A special device to record information in cards or tape by punching holes in the cards or tape to represent letters, digits, and special characters; (2) to operate a device for punching holes in cards or tape.

L

LABEL – Symbols used to identify or describe an item, record, message, or file. It may be the same as the address in storage.

LANGUAGE – System for representing and communicating information or data between people, or between people and machines. A system consists of a carefully defined set of characters and rules for combining them into larger units, such as words or expressions, and rules for word arrangement or usage to achieve specific meanings.

LANGUAGE, ALGORITHMIC – Arithmetic language by which numerical procedures may be precisely presented to a computer in a standard form. Language is intended as a means of directly presenting any numerical procedure to any suitable computer for which a compiler exists, and also to communicate numerical procedures among individuals. The language itself results from international cooperation to obtain a standardized algorithmic language.

LANGUAGE, COMMON MACHINE – Machine-sensible information representation common to a related group of data processing machines.

LANGUAGE, COMMON BUSINESS ORIENTED – Specific language by which business data processing procedures may be precisely described in a standard form, intended not only to present any business program to any suitable computer for which a compiler exists, but as a means of communicating such procedures among individuals.

LANGUAGE, INTERNATIONAL ALGEBRAIC – Forerunner of ALGOL.

LANGUAGE, MACHINE – (1) Language designed for interpretation and use by a machine without translation; (2) system for expressing information which is intelligible to a specific machine (a computer or class of computers). Such a language may include instructions which define and direct machine operations, and information to be recorded by or acted upon by these machine operations; (3) set of instructions expressed in the number system basic to a computer, together with symbolic operation codes with absolute addresses, relative addresses, or symbolic addresses.

LANGUAGE, OBJECT – Language which is output of an automatic coding routine. Usually object language and machine language are the same, but a series of steps in an automatic coding system may involve object language of one step serving as a source language for the next step.

LANGUAGE, PROBLEM ORIENTED – (1) Language designed for convenience of program specification in a general problem area rather than for easy conversion to machine instruction code. (Components of such language may bear little resemblance to machine instructions.); (2) machine-independent language where one need only state the problem, not the how of solution.

LANGUAGE, PROGRAM – Language used by programmers to write computer routines.

LANGUAGE, SOURCE – Original form in which a program is prepared before machine processing.

LENGTH, RECORD — Number of characters necessary to contain all the information in a record.

LENGTH, WORD — Number of characters in a machine word. In a given computer, the number may be constant or variable.

LIBRARY — Collection of information available to a computer, usually on magnetic tapes.

LIBRARY, ROUTINE — Collection of standard, proven routines and subroutines by which problems may be solved.

LIBRARY, SUBROUTINE — Standard and proven subroutines kept on file for use at any time.

LINE, ACOUSTIC DELAY — Delay line using a medium providing acoustic delay as mercury or quartz delay lines.

LIST, ASSEMBLY — Printed list, the byproduct of an assembly procedure. It lists in logical instruction sequence details of a routine showing the coded and symbolic notation next to the actual notations established by the assembly procedure. Highly useful in the debugging of a routine.

LIST, PUSH DOWN — List of items where the last item entered is the first item of the list, and the relative position of the other items is "pushed back" by one item.

LIST, PUSH UP — List of items where each item is entered at the end of the list, and the other items maintain their same relative position in the list.

LOAD — (1) To put data into a register or storage; (2) to put a magnetic tape onto a tape drive, or to put cards into a card reader.

LOAD-AND-GO — Automatic coding procedure which compiles the program, creating machine language, and proceeds to execute the created program. Such procedures are usually part of a monitor.

LOCATION — Storage position in the main internal storage which stores one computer word and which is usually identified by an address.

LOGIC — (1) Science dealing with criteria or formal principles of reasoning and thought; (2) systematic scheme which defines the interactions of signals in the design of an automatic data processing system; (3) principles and application of truth tables and interconnection between logical elements required for arithmetic computation in an automatic data processing system.

LOGIC, SYMBOLIC — (1) Study of formal logic and mathematics by special written language which avoids the ambiguity and inadequacy of ordinary language; (2) mathematical concepts, techniques, and languages as used in (1), whatever their particular application or context.

LOOK UP TABLES — See table.

LOOP — (1) Self-contained series of instructions in which the last instruction can modify and repeat itself until a terminal condition is reached. Productive instructions in the loop generally manipulate the operands, while bookkeeping instructions modify the productive instructions, count the number of repetitions. A loop may contain any number of conditions for termination. The equivalent can be achieved by the technique of straight line coding, whereby the repetition of productive and bookkeeping operations is accomplished by explicitly writing the instructions for each repetition; (2) communications circuit between two private subscribers or between subscriber and local switching center.

LOW-ORDER — Pertaining to the weight or significance assigned to the digits of a number: in the number 123456, the lower order digit is six. The three low-order bits of a binary word are another example.

LPM — Lines Per Minute.

MAINTENANCE, FILE — Periodic file modification to incorporate changes occurring during a given period.

MAINTENANCE, PREVENTIVE — Maintenance of a computer system to keep equipment in operating condition and prevent failures during productive runs.

MAINTENANCE, REMEDIAL — Maintenance performed by contractor following equipment failure: performed as required, on an unscheduled basis.

MALFUNCTION — Failure in the operation of the hardware of a computer.

MASKING — (1) Process of extracting a nonword group or a field of characters from a word or string of words; (2) process of setting internal program controls to prevent transfers that otherwise would occur upon setting of internal machine latches.

MATRIX — (1) Array of quantities in a prescribed form. In mathematics, usually capable of being subject to mathematical operation by an operator or another matrix; (2) array of coupled circuit elements: diodes, wires, magnetic cores, and relays, capable of performing a specific function such as conversion from one numerical system to another. The elements are usually arranged in rows and columns. A matrix is a particular type of encoder or decoder.

MESSAGE — (1) Group of words, variable in length, transported as a unit; (2) transported item of information.

MICROCOMMAND — A word obtained from the control store that exercises elementary control over the various system elements within a basic machine cycle.

MICROPROGRAM — (1) Program of analytic instructions which the programmer constructs from the basic subcommands of a digital computer; (2) sequence of pseudo commands translated by hardware into machine subcommands; (3) means of building various analytic instructions as needed from the subcommand structure of a computer; (4) plan for obtaining maximum utilization of the abilities of a digital computer by efficient use of the subcommands of the machine.

MISTAKE — Human failing: faulty arithmetic, use of incorrect formula, or incorrect instructions: sometimes called gross errors to distinguish from rounding and truncation errors. Computers malfunction and humans make mistakes. Computers do not make mistakes and humans do not malfunction, in this sense.

MIT — Master Instruction Tape. See tape, master instruction.

MNEMONIC — Pertaining to the assisting of human memory: a mnemonic term, usually an abbreviation, that is easy to remember (mpy for multiply and acc for accumulator).

MODIFY — (1) To alter a portion of an instruction to make its interpretation and execution other than normal. Modification may or may not permanently change the instruction or affect only the current execution. Most frequent modification is that of the effective address through use of index registers; (2) to alter a subroutine according to a defined parameter.

MODULE — (1) Interchangeable plug-in item containing components; (2) an incremental block of storage or other building block for expanding the computer capacity.

MONITOR — To supervise and verify the correct operation of a program during its execution, usually by a diagnostic routine used from time to time to answer questions about the program.

MONITOR ROUTINE — See routine, executive.

MULTIPLEX — The process of transferring data from several storage devices operating at relatively low transfer rates to one storage device operating at a high transfer rate so that the high-speed device is not obliged to wait for the low-speed devices.

MULTIPROGRAMMING — Technique for handling numerous routines or programs simultaneously by an interweaving process.

N

NANOSECOND — One-thousandth of a millionth of a second; 10^{-9} seconds.

NOISE — Meaningless extra bits or words which must be ignored or removed from the data when the data are used.

NORMALIZE — (1) To adjust the exponent and fraction of a floating point quantity so that the fraction lies in the prescribed normal standard range; (2) to reduce a set of symbols or numbers to a normal or standard form.

NOTATION — (1) Act, process, or method of representing facts or quantities by a system or set of marks, signs, figures, or characters; (2) system of such symbols or abbreviations used to express technical facts or quantities; as mathematical notations; (3) annotation; note.

NOTATION, SYMBOLIC — Method of representing a storage location by one or more figures.

NUMBER — (1) The, or a total, aggregate, or amount of units; (2) a figure or word, or a group of figures or words, representing graphically an arithmetical sum; a numeral, as the number 45; (3) numeral by which a thing is designated in a series, as a pulse number; (4) single member of a series designated by consecutive numerals, as a part number; (5) character, or a group of characters, uniquely identifying or describing an article, process, condition, document, or class; (6) to count, enumerate; (7) to distinguish by a number.

NUMBER, BINARY — A number, usually consisting of more than one figure, representing a sum, in which the individual quantity represented by each figure is based on a radix of two. The figures used are 0 and 1.

NUMBER, DECIMAL — A number, usually of more than one figure, representing a sum, in which the quantity represented by each figure is based on the radix of ten. The figures are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

NUMBER, HEXADECIMAL — Number, usually of more than one figure, representing a sum in which the quantity represented by each figure is based on a radix of sixteen.

NUMBER, SYMBOLIC — Numeral, used in writing routines, for referring to a specific storage location; such numerals are converted to actual storage addresses in the final assembling of the program.

O

OCTAL — Pertaining to eight; usually a number system of base or radix eight: in octal notation, octal 214 is 2 times 64, plus 1 times 8, plus 4 times 1, and equals decimal 140. Octal 214 in binary-coded-octal is represented as 010, 001, 100; octal 214, as a straight binary number is written 10001100. Note that binary coded octal and straight binary differ only in the use of commas; in the example shown, the initial zero in the straight binary is dropped.

OFF-LINE — Descriptive of a system and peripheral equipment or devices in which the operation of peripheral equipment is not under the control of the central processing unit.

ON-LINE — Descriptive of a system and of peripheral equipment or devices in which the operation of such equipment is under control of the central processing

unit, and in which information reflecting current activity is introduced into the data processing system as soon as it occurs. Thus, directly in-line with the main flow of transaction processing.

OPEN-ENDED — Quality by which addition of new terms, subject headings, or classifications does not disturb the preexisting system.

OPERAND — Quantity entering or arising in an instruction. An operand may be an argument, a result, a parameter, or an indication of the location of the next instruction, as opposed to the operation code or symbol itself. It may be the address portion of an instruction.

OPERATION, HOUSEKEEPING — General term for the operation performed for a machine run before actual processing begins. Examples of housekeeping operations are establishing controlling marks, setting up auxiliary storage units, reading in first record for processing, initializing, setup verification operations, and file identification.

OPERATION, PARALLEL — The performance of several actions, usually of a similar nature, simultaneously through provision of individual similar or identical devices for each such action. Particularly flow or processing of information. Parallel operation is performed to save time over serial operation. Parallel operation usually requires more equipment.

OPERATION, REAL TIME — Use of computer as an element of a processing system in which times of occurrence of data transmission are controlled by other portions of the system, or by physical events outside the system, and cannot be modified for convenience in computer programming. Such an operation either proceeds at the same speed as the events being simulated or at a sufficient speed to analyze or control simultaneous external events.

OPERATION, SEQUENTIAL — Performance of actions one after the other in time. The actions referred to are of a large scale as opposed to the smaller scale operations referred to by the term serial operation. For an example of sequential operation consider Ax(BxC). The two multiplications indicated follow each other sequentially. However, the processing of the individual digits in each multiplication may be either parallel or serial.

OPERATION, SERIAL — Flow of information through a computer in time sequence using only one digit, word, line or channel at a time.

OPERATOR — (1) A mathematical symbol which represents a mathematical process to be performed on an associated operand; (2) the portion of an instruction which tells the machine what to do; (3) a machine operator.

OPERATOR, AND — (1) Logical operator which has the property that if P is a statement and Q is a statement, then P AND Q is true if both statements are true, false if either is false or both are false. Truth is normally expressed by the value 1, falsity by 0. The AND operator is often represented by a centered dot (P·Q), by no sign (PQ), by an inverted "u" or logical product symbol ($P \cap Q$), or by the letter "X" or multiplication symbol (PxQ). Note that the letters AND are capitalized to differentiate between the logical operator AND the conjunction; (2) the logical operation which makes use of the AND operator or logical product.

OPERATOR, EXCLUSIVE OR — A logical operator which has the property that if P and Q are two statements, then the statement P*Q, where the * is the Exclusive OR operator, is true if either P or Q, but not both are true, and false if P and Q are both false or both true, according to the following table, wherein the figure 1 signifies a binary digit or truth.

P	Q	P*Q	
0	0	0	(even)
0	1	1	(odd)
1	0	1	(odd)
1	1	0	(even)

The Exclusive OR is the same as the Inclusive OR, except that the case with both inputs true yields no output: P*Q is true if P or Q are true, but not both. Primarily used in compare operations.

OPERATOR, INCLUSIVE OR — Logical operator which has the property that P or Q is true, if P or Q or both is true; when the term OR is used alone, as in OR-gate, the inclusive OR is usually implied.

OPERATOR, OR — Logical operator which has the property such that if P or Q are two statements, then the statement P or Q is true or false varies according to the following possible combinations:

P	Q	P or Q
False	True	True
True	False	True
True	True	True
False	False	False

ORDER — (1) Defined successive arrangement of elements or events. Losing favor as a synonym for instructions, due to ambiguity; (2) to sequence or arrange in a series; (3) weight or significance assigned to a digit position in a number.

ORIGIN — The absolute storage address in relative coding to which addresses in a region are referenced.

OUTPUT — (1) Information transferred from internal storage of a computer to secondary or external storage, or to any device outside of the computer; (2) routines which direct (1); (3) device or collective set of devices necessary for (1); (4) to transfer from internal storage on to external media.

OVERFLOW — (1) The condition which arises when the result of an arithmetic operation exceeds the capacity of the storage space allotted in a digital computer; (2) digit arising from this condition if a mechanical or programmed indicator is included (otherwise digit may be lost.

OVERLAY — Technique for bringing routines into high-speed storage from some other form of storage during processing, so that several routines will occupy the same storage locations at different times. Overlay is used when the total storage requirements for instructions exceed the available main storage.

OVERPUNCH — To add holes in a card column that already contain one or more holes.

P

PANEL, CONTROL — (1) Interconnection device, usually removable, which employs removable wires to control the operation of computing equipment. Used on punch card machines to carry out functions controlled by the user. On computers it is used primarily to control input and output functions; (2) device or component of some data processing machines that permits the expression of instructions in a semifixed computer program by the insertion of pins, plugs, or wires into sockets, or hubs in the device, in a pattern to represent instructions, thus making electrical interconnections which may be sensed by the data processing machine.

PARALLEL — (1) To handle simultaneously in separate facilities; (2) to operate on two or more parts of a word or item simultaneously.

PARAMETER — (1) Quantity in a subroutine whose value specifies or partly specifies the process to be performed. It may be given different values when the subroutine is used in different main routines or in different parts of one main routine, but which usually remains unchanged throughout any one such use; (2) quantity used in a generator to specify machine configuration, designate subroutines to be included, or otherwise to describe the desired routine to be generated; (3) constant or a variable in mathematics, which remains constant

during some calculation; (4) definable characteristic of an item, device, or system.

PASS — Complete cycle of reading, processing and writing: a machine run.

PATCH — (1) Section of coding inserted into a routine to correct a mistake or alter the routine, often not inserted into the actual sequence of the routine being corrected, but placed somewhere else, with an exit to the patch and a return to the routine provided; (2) to insert corrected coding.

PERIOD, PERFORMANCE — Period of 30 consecutive calendar days during which a newly installed computer is being tested for acceptance by the U.S. Government. Such a period does not include equipment time used for data purification, file conversion, and similar preparatory operations or those hours of operation rescheduled as a result of equipment failure.

PING-PONG — Programming technique of using two magnetic tape units for multiple reel files and switching automatically between the two units until the complete file is processed.

PLOTTER — Visual display or board in which a dependent variable is graphed by an automatically controlled marker as a function of one or more variables.

POINTER, BINARY — Radix pointer in a binary number system: the dot that marks the position between the integral and fractional, or units and halves in a binary number.

POINT, LOAD — Preset point at which magnetic tape is initially positioned under the read-write head to start reading or writing.

POINT, RADIX — Dot delineating the integer digits from the fractional digits of a number; specifically, the dot that delineates the digital position, involving the zero exponent of the radix from the digital position involving the minus-one exponent of the radix. The radix point is often identified by the name of the system (binary point, octal point, or decimal point). In the writing of any number in any system, if no dot is included, the radix point is assumed to follow the rightmost digit.

PRE-EDIT — To edit the input data previous to the computation.

PRECISION — (1) Degree of exactness with which a quantity is stated; (2) degree of discrimination or amount of detail: a 3 decimal digit quantity discriminates among 1000 possible quantities. A result may have more precision than it has accuracy: the true value of pi to 6 significant digits is 3.14159; the value 3.14162 is precise to 6 figures, given to 6 figures, but is accurate only to about 5.

PRIMITIVE — Primitive usually pertains to the lowest level of a machine instruction or lowest unit of language translation.

PROBLEM, BENCHMARK — Routine used to determine the speed performance of a computer. One method is to use one-tenth of the time required to perform nine complete additions and one complete multiplication. A complete addition or a complete multiplication time includes the time required to procure two operands from storage, perform the operation and store the result, and the time required to select and execute the required number of instructions.

PROCESS — General term covering such terms as assemble, compile, generate, interpret, and compute.

PROCESS, ITERATIVE — A process for calculating a desired result by means of a repeating cycle of operations, which comes closer and closer to the desired result; e.g., the arithmetical square root of N may be approximated by an iterative process using additions, subtractions, and divisions only.

PROCESSING, AUTOMATIC DATA — Processing performed by a system of electronic or electrical machines so interconnected and interacting as to reduce to

a minimum the need for human assistance or intervention. Synonymous with (ADP) and related to (system, automatic data processing).

PROCESSING, BATCH — Technique by which terms to be processed must be coded and collected into groups before processing.

PROCESSING, DATA — (1) Preparation of source media which contain data or basic elements of information, and the handling of such data according to precise rules or procedures to accomplish such operations as classifying, sorting, calculating, summarizing, and recording; (2) production of records and reports.

PROCESSING, ELECTRONIC DATA — Data processing performed largely by electronic equipment.

PROCESSING, INFORMATION — A less restrictive term than data processing, encompassing the complete scientific and business operations performed by a computer.

PROCESSING, PARALLEL — The operation of a computer so that programs for more than one run are stored simultaneously in its storage, and executed concurrently.

PROCESSING, REAL TIME — Processing of information or data in a sufficiently rapid manner so that the results of the processing are available in time to influence the process being monitored or controlled.

PROCESSOR — (1) Generic term which includes assembly, compiling, and generation; (2) shorter term for automatic data processor or arithmetic unit.

PROGRAM — (1) Complete plan for the solution of a problem, more specifically the complete sequence of machine instructions and routines necessary to solve a problem; (2) to plan the procedures for solving a problem. This involves the analysis of the problem, preparation of a flow diagram, preparing details, testing, and developing subroutines, allocation of storage locations, specification of input and output formats, and incorporation of a computer run into a complete data processing system.

PROGRAM, CONTROL — Sequence of instructions which prescribes the steps to be taken by a computer system or any other device.

PROGRAM, GENERAL — Program expressed in computer code designed to solve a class of problems, or specializing on a specific problem when appropriate parametric values are supplied.

PROGRAM, OBJECT — Program which is the output of an automatic coding system. Often the object program is a machine language program ready for execution, but it may well be in an intermediate language. Contrasted with (program, source).

PROGRAM, SOURCE — Computer program written in a language designed for ease of expression of a class of problems or procedures, by humans: symbolic or algebraic. A generator, assembler, translator, or compiler routine is used to perform the mechanics of translating the source program into an object program in machine language. See program, object, above.

PROGRAMMING, INTERPRETIVE — Writing of programs in pseudo machine language, which is precisely converted by the computer into actual machine language instructions before being performed by the computer.

PROGRAMMING, MICRO — Technique of using a special set of instructions for an automatic computer that consists only of basic elemental operations which the programmer may combine into higher level instructions, which he may then program using the higher level instructions only: if a computer has only basic instructions for adding, subtracting, and multiplying, the instruction for dividing would be defined by microprogramming.

PROGRAMMING, SYMBOLIC — Use of arbitrary symbols to represent addresses in order to facilitate programming.

PROM — Programmable Read Only Memory. Integrated circuit array that is manufactured with a pattern of all logical zeros or ones and has a specific pattern written into it by a special hardware programmer.

PSEUDO-OPERATION — An operation which is not part of the computer's operation repertoire as realized by hardware; hence an extension of the set of machine operations.

PSEUDO-RANDOM — Property of satisfying one or more of the standard criteria for statistical randomness but being produced by a definite calculation process.

PUNCH, CARD — Machine which punches cards in designated locations to store data which can be conveyed to other machines or devices by reading or sensing the holes.

R

RADIX — Quantity of characters for use in each of the digital positions of a numbering system. In the more common numbering systems the characters are some or all of the Arabic numerals:

System Name	Character	Radix
Binary	(0,1)	2
Octal	(0,1,2,3,4,5,6,7)	8
Decimal	(0,1,2,3,4,5,6,7,8,9)	10

Unless otherwise indicated, the radix of any number is assumed to be 10. For positive identification of a radix 10 number, the radix is written in parentheses as a subscript to the expressed number: $126_{(10)}$. The radix of any nondecimal number is expressed in similar fashion: $11_{(2)}$ and $5_{(8)}$. Synonymous with base.

RANDOM ACCESS — See access, random.

RATE, BIT — Rate at which binary digits, or pulses representing them, pass a given point on a communications line or channel.

RATE, CLOCK — Time rate at which pulses are emitted from the clock. The clock rate determines the rate at which logical or arithmetic gating is performed with a synchronous computer.

RATE, ERROR — Total amount of information in error, due to the transmission media, divided by the total amount of information received.

RATE, SAMPLING — Rate at which measurements of physical quantities are made: if it is desired to calculate the velocity of a missile and its position is measured each millisecond, then the sampling rate is 1,000 measurements per second.

RATIO, SIGNAL-TO-NOISE — Ratio of the amount of signals conveying information to the amount of signals not conveying information.

READ — (1) To sense information contained in some source; (2) the sensing of information contained in some source.

READ-IN — To sense information contained in some source and transmit this information to an internal storage.

READ, NONDESTRUCTIVE — Reading of the information in a register without changing that information.

READ-OUT — To sense information contained in some internal storage and transmit this information to a storage external to the computer.

READ, CARD — (1) Mechanism that senses information punched into cards; (2) input device consisting of a mechanical punch card reader and related electronic circuitry which transcribes data from punch cards to working storage or magnetic tape.

READER, CHARACTER — Specialized device which can convert data represented in one of the type fonts or scripts read by human beings directly into machine language. Such a reader may operate optically, or if the characters are printed in magnetic ink, the device may operate magnetically or optically.

READER, HIGH-SPEED — Reading device capable of being connected to a computer to operate online without seriously holding up the computer. A card reader reading more than 250 cards per minute would be called a high-speed reader. A reader which reads punched paper tape at a rate greater than 50 characters per second could also be called a high-speed reader.

READER, MAGNETIC TAPE — Device capable of sensing information recorded on a magnetic tape in the form of a series of magnetized spots.

READER, PAPER TAPE — Device capable of sensing information punched on a paper tape in the form of a series of holes.

RECORD, UNIT — (1) Separate record that is similar in form and content to other records; (2) sometimes a piece of nontape auxiliary equipment (card reader, printer or console typewriter).

REGISTER — Hardware device used to store bits or characters. A register is usually constructed of elements such as transistors or tubes and usually contains approximately one word of information. Common programming usage demands that a register have the ability to operate upon information and not merely store information; hardware usage does not make the distinction.

REGISTER, INDEX — A register which contains a quantity which may be used to modify addresses. B-register.

REGISTER, SHIFT — Register in which the characters may be shifted one or more positions to the right or left. In a right shift, the rightmost characters are lost. In a left shift, the leftmost characters are lost.

RELIABILITY — (1) A measure of the ability to function without failure; (2) the amount of credence placed in a result.

RERUN — To repeat all or part of a program on a computer.

RESTART — To go back to a specific planned point in a routine, usually in the case of machine malfunction, for the purpose of rerunning the portion of the routine in which the error occurred. The length of time between restart points in a given routine should be a function of the mean free-error time of the machine itself.

RESTORE — To return an index register, a variable address, or other computer word to its initial or preselected value.

RETRIEVAL, INFORMATION — Recovering of desired information or data from a collection of graphic records.

RETURN — Mechanism providing for a return in the usual sense, in particular a set of instructions at the end of a subroutine which permit control to return to the proper point in the main routine.

ROUND — Deletion of the least significant digit(s) with or without modifications to reduce bias.

ROUTINE — Set of coded instructions arranged in proper sequence to direct the computer to perform a desired operation or sequence of operations, or a subdivision of a program consisting of two or more instructions that are functionally related (a program). See subroutine and program.

ROUTINE, DIAGNOSTIC — Routine used to locate a malfunction in a computer, or to aid in locating mistakes in a computer program. Thus, any routine specifically designed to aid in debugging or trouble shooting.

ROUTINE, EXECUTIVE — Routine which controls loading and relocation of routines and in some cases makes use of instructions which are unknown to the general programmer. Effectively, an executive routine is part of the machine itself.

ROUTINE, FLOATING POINT — Set of subroutines which cause a computer to execute floating point arithmetic. These routines may be used to simulate floating point operations on a computer with no built-in floating point hardware.

ROUTINE, HOUSEKEEPING — Initial instructions in a program which are executed only one time: clear storage.

ROUTINE, INTERPRETIVE — Routine that decodes and executes instructions written as pseudocodes, contrasted with a compiler which decodes the pseudocodes into a machine language routine to be executed at a later time. The essential characteristic of an interpretive routine is that a particular pseudo code operation must be decoded each time it is executed.

RUN — Performance of one program on a computer, thus the performance of one routine, or several routines linked so that they form an automatic operating unit, during which manual manipulations by the computer operator are minimal.

S

SCALE — A range of values frequently dictated by the computer word-length or routine at hand.

SCAN — To examine every reference or every entry in a file routinely as a part of a retrieval scheme; occasionally, to collate.

SCREEN — (1) Surface in an electrostatic cathode ray storage tube where electrostatic charges are stored, and by means of which information is displayed or stored temporarily; (2) to make preliminary selection from a set of entities, selection criteria being based on a given set of rules or conditions.

SEARCH — To examine a series of items for any that have a desired property or properties.

SEARCH, BINARY — Search in which the series of items is divided into two parts, one of which is rejected, and the process repeated on the unrejected part until the item with the desired property is found. This process usually depends upon the presence of a known sequence in the series.

SEGMENT — (1) To divide a routine in parts, each consisting of an integral number of subroutines, and each part capable of being completely stored in the internal storage and containing the necessary instructions to jump to other segments; (2) that portion of a routine too long to fit into internal storage which is short enough to be stored entirely in the internal storage. Such a segment contains the coding necessary to call in other segments automatically. Routines which exceed internal storage capacity may be automatically divided into segments by a compiler.

SELECT — (1) To take the alternative A if the report on a condition is of one state, and alternative B if the report on the condition is of another state; (2) to choose a needed subroutine from a file of subroutines.

SELECTOR — Device which interrogates a condition and initiates one of several alternate operations.

SENSE — (1) To examine, particularly relative to a criterion; (2) to determine the present arrangement of some element of hardware, especially a manually-set switch; (3) to read punched holes or other marks.

SENSING, MARK — Technique for detecting special pencil marks entered in special places on a punch card and automatically translating the marks into punched hole.

SEQUENCE — (1) To put a set of symbols into an arbitrarily defined order: to select A if A is greater than or equal to B, or select B if A is less than B; (2) arbitrarily defined order of a set of symbols: an orderly progression of items of information or of operations in accordance with some rule.

SEQUENCE, CALLING — Instructions used for linking a closed subroutine with a main routine: standard linkage and a list of the parameters.

SEQUENCE, CONTROL — Normal order of selection of instructions for execution. In some computers one of the addresses in each instruction specifies the control sequence. In most computers, the sequence is consecutive except where a transfer occurs.

SEQUENCE, RANDOM NUMBER — Unpredictable array of numbers produced by change, and satisfying one or more of the tests for randomness.

SERIAL — (1) Handling of one after the other in a single facility, such as transfer or store in a digit-by-digit time sequence, or to process a sequence of instructions one at a time (sequentially); (2) time sequence transmission of, storage of, or logical operations on the parts of a word, with the same facilities for successive parts. Related to operation, serial and contrasted with parallel (2).

SERIAL-PARALLEL — (1) Combination of serial and parallel (serial by character, parallel by bits comprising the characters; (2) descriptive of a device which converts a serial input into a parallel output.

SET — (1) To place a storage device in a prescribed state; (2) to place a binary cell in the one state; (3) a collection of elements having some feature in common or which bear a certain relation to one another: all even numbers, geometrical figures, terms in a series, a group of irrational numbers, all positive even integers less than 100 may be a set or a subset.

SET, CHARACTER — Agreed set of representations (characters) from which selections are made to denote and distinguish data. Each character differs from all others, and the total number of characters in a given set is fixed: a set may include the numerals 0 to 9, the letters A to Z, punctuation marks and a blank or space. Clarified by alphabet.

SHIFT — To move the characters of a unit of information columnwise right or left. For a number, this is equivalent to multiplying or dividing by a power of the base of notation. See below.

SHIFT, ARITHMETIC — To multiply or divide a quantity by a power of the number base: if binary 1101, which represents decimal 13, is arithmetically shifted twice to the left, the result is 110100, which represents 52, which is also obtained by multiplying 13 by 2 twice; on the other hand, if the decimal 13 were to be shifted to the left twice, the result would be the same as multiplying by 10 twice, or 1300.

SHIFT, CYCLIC — Shift in which the digits dropped-off at one end of a word are returned at the other in a circular fashion: if register holds eight digits, 23456789, the result of a cyclic shift two columns to the left would be to change the contents of the register to 45678923.

SIMULATION — (1) The representation of physical systems and phenomena by computers, models or other equipment: an imitative type of data processing in which an automatic computer is used as a model of some entity; a chemical process. Information enters the computer to represent the factors entering the real process, the computer produces information that represents the results of the process, and the processing done by the computer represents the process itself; (2) in computer programming, the technique of setting up a routine for one computer to make it operate as nearly as possible like some other computer.

SIMULATOR – (1) Computer or model representing a system or phenomenon which mirrors or maps the effects of various changes in the original, enabling the original to be studied, analyzed, and understood by means of the behavior of the model; (2) a program or routine corresponding to a mathematical model or representing a physical model; (3) a routine executed by one computer but which imitates the operations of another computer.

SOFTWARE – The totality of programs and routines used to extend the capabilities of computers, such as compilers, assemblers, narrators, routines, and subroutines. Contrasted with hardware.

SORT – To arrange items of information according to rules dependent upon a key or field contained in the items or records: to digital-sort is to sort first the keys on the least significant digit, and to resort on each higher order digit until the items are sorted on the most significant digit.

SORT, MERGE – To produce a single sequence of items, ordered according to some rule, from two or more previously unordered sequences, without changing the items in size, structure, or total number. More than one pass may be required for a complete sort, but items are selected during each pass on the basis of the entire key.

STORAGE – (1) The term preferred to memory; (2) pertaining to a device in which data can be stored and from which it can be obtained at a later time. The means of storing data may be chemical, electrical or mechanical; (3) a device consisting of electronic, electrostatic, electrical, hardware or other elements into which data may be entered, and from which data may be obtained as desired; (4) the erasable storage in any given computer. See memory.

STORAGE, BUFFER – (1) Synchronizing element between two different forms of storage, usually between internal and external; (2) input device in which information is assembled from external or secondary storage and stored ready for transfer to internal storage; (3) output device into which information is copied from internal storage and held for transfer to secondary or external storage. Computation continues while transfers between buffer storage and secondary or internal storage or vice versa take place; (4) device which stores information temporarily during data transfers. See buffer.

STORAGE, DISK – Storage of data on the surface of magnetic disks. See disk, magnetic and storage, magnetic disk.

STORAGE, MAGNETIC CORE – Storage device in which binary data are represented by the direction of magnetization in each unit of an array of magnetic material, usually in the shape of o-rings, but also in other forms such as wraps on bobbins. Synonymous with core storage.

STORAGE, MAGNETIC DISK – Storage system consisting of magnetically coated disks, on the surface of which information is stored in the form of magnetic spots arranged to represent binary data. These data are arranged in circular tracks around the disks and are accessible to reading and writing heads on an arm which can be moved mechanically to the desired disk and then to the desired track on that disk. Data from a given track are read or written sequentially as the disk rotates. See storage, disk.

STORAGE, PARALLEL – Storage of data in which all bits, characters, or words are essentially equally available in space, without time being one of the factors. When words are in parallel, the storage is said to be parallel by words; when characters within words, or binary digits within words or characters, are dealt with simultaneously, not one after the other, the storage is parallel by characters, or parallel by bit.

STORAGE, PROGRAM – Portion of the internal storage reserved for the storage of programs, routines, and subroutines. In many systems protection devices are used to prevent inadvertent alteration of the contents of the program storage. Contrasted with storage, temporary.

STORAGE, TEMPORARY — Portion of the internal storage reserved for the data upon which operations are being performed. Synonymous with working space and storage; contrasted with storage, program.

STORE — (1) To transfer an element of information to a device from which the unaltered information can be obtained at a later time; (2) to retain data in a device from which it can be obtained at a later time.

SUBPROGRAM — Part of a larger program which can be converted into machine language independently. See microprogram.

SUBROUTINE — (1) Set of instructions necessary to direct the computer to carry out a well defined mathematical or logical operation; (2) subunit of a routine. A subroutine is often written in relative or symbolic coding even when the routine to which it belongs is not; (3) portion of a routine that causes a computer to carry out a well-defined mathematical or logical operation; (4) routine arranged so that control may be transferred to it from a master routine and so that, at the conclusion of the subroutine, control reverts to the master routine (usually called closed subroutine); (5) single routine may simultaneously be both a subroutine with respect to another routine and a master routine with respect to a third. Control is usually transferred to a single subroutine from more than one place in the master routine; the reason for using the subroutine is to avoid having to repeat the same sequence of instructions in different places in the master routine. See routine.

SUBROUTINE, CLOSED — Subroutine not stored in the main path of the routine. Such a subroutine is entered by a jump operation; provision is made to return control to the main routine at the end of the operation. The instructions related to the entry and reentry function constitute a linkage.

SUBROUTINE, STATIC — A subroutine which involves no parameters other than the addresses of the operands.

SUBSET — (1) A set contained within a set; (2) a subscriber apparatus in a communications network.

SUBTRAHEND — The number or quantity which is subtracted from another number, called the minuend, giving a result usually called the difference, or sometimes called the remainder.

SUM, LOGICAL — A result, similar to an arithmetic sum, obtained in the process of ordinary addition, except that the rules are such that a result of one is obtained when either one or both input variables is a one, and an output of zero is obtained when the input variables are both zero. The logical sum is the name given the result produced by the inclusive or operator.

SYMBOL, LOGICAL — Sign used as an operator to denote the particular operation to be performed on the associated variables.

SYNTAX — The rules governing sentence structure in a language, or statement structure in a language such as that of a compiler.

SYSTEM — Assembly of procedures, processes, methods, routines, or techniques united by regulated interaction to form an organized whole.

SYSTEM, INFORMATION — Network of all communication methods within an organization. Information may be derived from many sources other than a data processing unit: telephone, personal contact, or by studying an operation.

SYSTEM, INFORMATION RETRIEVAL — System for locating and selecting, on demand, certain documents or other graphic records relevant to a given information requirement from a file. Examples of information retrieval systems are classification, indexing, and machine searching systems.

SYSTEM, NUMBER — (1) Systematic method for representing numerical quantities in which any quantity is represented as the sequence of coefficients of the

successive powers of a particular base with an appropriate point. Each succeeding coefficient from right to left is associated with and usually multiplies the next higher power of the base. The first coefficient to the left of the point is associated with the zero power of the base. For example, in decimal notation 371.426 represents $(3x10^2)+(7x10^1)+(1x10^0)+(4x10^{-1})+(2x10^{-2})+(6x10^{-3})$; (2) following are names of the number systems with bases 2 through 20: 2, binary; 3, ternary; 4, quaternary; 5, quinary; 6, senary; 7, septenary; 3, octal, or octonary; 9, novenary; 10, decimal; 11, undecimal; 12, duodecimal; 13, terdenary; 14, quaterdenary; 15, quindenary; 16, sexadecimal, or hexadecimal; 17, septendecimal; 18, octodenary; 19, novemdenary; 20, vicenary. 32, duosexadecimal, or duotricinary; and 60, sexagenary. The Binary, Octal, Decimal, and Sexadecimal systems are widely used in computers.

SYSTEM, OPERATING — Integrated collection of service routines for supervising the sequencing of programs by a computer. Operating systems may perform debugging, input-output, accounting, compilation, and storage assignment tasks.

T

TABLE — Collection of data in a form suitable for ready reference, frequently as stored in sequenced machine locations or written in the form of an array of rows and columns for easy entry and in which an intersection of labeled rows and columns serves to locate a specific piece of data or information.

TABLE, FUNCTION — (1) Two or more sets of information so arranged that an entry in one set selects one or more entries in the remaining sets; (2) a dictionary; (3) a device constructed of hardware, or a subroutine, which can either decode multiple inputs into a single output or encode a single input into multiple outputs; (4) a tabulation of the values of a function for a set of values of the variable.

TABLE LOOK UP (TLU) — Obtaining a function value corresponding to an argument, stated or implied, from a table of function values stored in the computer. Also, the operation of obtaining a value from a table.

TABLE, TRUTH — Representation of a switching function, or truth function, in which every possible configuration of argument values 0, 1, or true-false is listed, and beside each is given the associated function value 0-1 or true-false. The number of configurations is 2^N, where N is the number of arguments, unless the function is incompletely specified: don't care conditions. An example of a truth table for the AND-function and the OR-function (inclusive) is:

VARIABLE		AND	OR
A	B	AB	A+B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

TAG — Unit of information whose composition differs from that of other members of the set so that it can be used as a marker or label. A tag bit is an instruction word that is also called a sentinel.

TAPE, MAGNETIC — Tape or ribbon of any material impregnated or coated with magnetic or other material on which information may be placed in the form of magnetically polarized spots.

TAPE, PAPER — Strip of paper capable of storing or recording information. Storage may be in the form of punched holes, partially punched holes, carbonization or chemical change of impregnated material, or imprinting. Some paper tapes, such as punched paper tapes, are capable of being read by the input device of a computer or a transmitting device by sensing the pattern of holes which represent coded information.

TAPE, PUNCH — Tape, usually paper, upon which data may be stored in the form of punched holes. Hole locations are arranged in columns across the width of the tape. There are usually 5 to 8 positions (channels) per column, with data represented by a binary coded decimal system. All holes in a column are sensed simultaneously in a manner similar to that for punch cards.

TIME, ACCESS — (1) Time it takes a computer to locate data or an instruction word in its storage section and transfer it to its arithmetic unit where the required computations are performed; (2) time required to transfer information which has been operated on from the arithmetic unit to the location in storage where the information is to be stored.

TIME, EXECUTION — The portion of an instruction cycle during which the actual work is performed or operation executed: the time required to decode and perform an instruction. See below.

TIME, INSTRUCTION — Portion of an instruction cycle during which the control unit is analyzing the instruction and setting up to perform the indicated operation. Same as time, execution.

TIME, LATENCY — (1) Time lag between completion of instruction staticizing and the initiation of the movement of data from its storage location; (2) rotational delay time from a disc file or a drum file.

TIME-SHARING — Use of a device for two or more purposes during the same overall time, accomplished by interspersing component actions in time.

TIME, SWITCHING — (1) Time interval between the reference-time, or time at which the leading edge of switching or driving pulse occurs, and the last instant at which the instantaneous voltage response of a magnetic cell reaches a stated fraction of its peak value; (2) time interval between the reference time and the first instant at which the instantaneous integrated voltage response reaches a stated fraction of its peak value.

TIME, TURN-AROUND — Time required to reverse the direction of transmission in a communication channel.

TRACE — Interpretive diagnostic technique which provides an analysis of each executed instruction and writes it on an output device as each instruction is executed.

TRACK — Path along which information is recorded on a storage device: the track on a drum or tape.

TRANSFER — (1) Conveyance of control from one mode to another by means of instructions or signals; (2) conveyance of data from one place to another; (3) instruction for transfer; (4) to copy, exchange, read, record, store, transmit, transport, or write data; (5) instruction which provides the ability to break the normal sequential flow of control.

TRANSFER OPERATION — See operation, transfer.

TRAP — (1) Special form of a conditional breakpoint activated by the hardware itself, by conditions imposed by the operating system, or by a combination of the two. Traps are an outgrowth of switch-controlled halts or jumps. Internal triggers or traps often exist in a computer. Since these are usually set only by unexpected or unpredictable occurrences and since the execution time and number of instructions for testing them can be burdensome, these triggers usually cause an automatic transfer of control, or jump to a known location, to record in other standard locations the location from which the transfer occurred and the cause of the transfer. Some trapping features can also be enabled or inhibited under program control: an overflow trap; (2) routine to determine indirectly the setting of internal triggers in the computer.

TROUBLE-SHOOT — To seek the cause of a malfunction or erroneous program behavior to remove the malfunction.

TRUNCATE — To drop digits of a number of terms of a series, lessening precision: the number 3.14159265 is truncated to five figures in 3.1415, whereas one may round off to 3.1416.

U

UNDERFLOW — (1) Condition which arises when a machine computation yields a result which is smaller than the smallest possible quantity which the machine is capable of storing; (2) a condition in which the exponent plus the excess becomes negative in a floating point arithmetic operation.

UNIT — Portion or subassembly of a computer which constitutes the means of accomplishing some inclusive operation or function.

UNIT, ARITHMETIC — Portion of the hardware of a computer in which arithmetic and logical operations are performed. The arithmetic unit generally consists of an accumulator, special registers for the storage of operands and results, supplemented by shifting and sequencing circuitry for implementing multiplication, division, and other desired operations.

UNIT, ASSEMBLY — (1) Device which performs the function of associating and joining several parts or piecing together a program; (2) a portion of a program capable of being assembled into a larger whole program.

UNIT, CONTROL — Computer segment which directs the sequence of operations, interprets the coded instructions, and initiates the proper commands to the computer circuits preparatory to execution.

UNIT, PAPER TAPE — Mechanism which handles punched paper tape and usually consists of a paper tape transport, sensing and recording or perforating heads and associated electrical and electronic equipments.

UNIT, READ PUNCH — Input-output unit of a computing system which punches computed results into cards, reads input information into the system, and segregates output cards. The read-punch unit generally consists of a card feed, a read station, a punch station, another read station, and output card stackers.

UNIT, TAPE — Device consisting of a tape transport, controls, a set of reels and a length of tape capable of recording and reading information on and from the tape, at the request of the computer under the influence of a program.

UPDATE — (1) To put into a master file the changes required by current information or transactions; (2) to modify an instruction so that the address numbers are increased by a stated amount each time the instruction is performed.

V

VALIDITY — Correctness: especially degree of closeness by which iterated results approach the correct result.

VALIDITY CHECK — See check, validity.

VARIABLE — (1) Quantity which can assume any of the numbers of some set of numbers; (2) condition, transaction, or event which changes or may be changed as a result of processing additional data through the system.

VECTOR — Quantity having magnitude and direction, in contrast with a scalar which has quantity only.

VERIFIER — Device on which a record can be compared or tested for identity character-by-character with a retranscription or copy as it is being prepared.

VERIFY — To check a transcribing operation by a compare operation. It usually applies to transcriptions which can be read mechanically or electrically.

VOCABULARY — List of operating codes or instructions available to the programmer for writing the program for a given problem for a specific computer.

VOCABULARY, SOPHISTICATED — Advanced and elaborate set of instructions. Some computers can perform only the more common mathematical calculations such as addition, multiplication, and subtraction. A sophisticated vocabulary computer can go beyond this and perform such operations as linearize, extract square root, and select highest number.

W

WORD — Ordered set of characters which occupies one storage location and is treated by the computer circuits as a unit and transferred as such. Ordinarily a word is treated by the control unit as an instruction, and by the arithmetic unit as a quantity. Word lengths may be fixed or variable.

WORD, CONTROL — Word, usually the first or last of a record, or first or last word of a block, which carries indicative information for the following words, records, or blocks.

WORD, DATA — Word which may be primarily regarded as part of the information manipulated by a program. A data word may be used to modify a program instruction or be arithmetically combined with other data words.

WORD, INFORMATION — Ordered set of characters bearing at least one meaning and handled by a computer as a unit, including separating and spacing, which may be contrasted with instruction words. See word, machine.

WORD-LENGTH, VARIABLE — Having the property that a machine word may have a variable number of characters, applicable either to a single entry whose information content may be changed from time to time, or to a group of functionally similar entries whose corresponding components are of different lengths.

WORD, MACHINE — A unit of information of a standard number of characters which a machine regularly handles in each transfer: a machine may regularly handle numbers or instruction in units of 36 binary digits; this is then the machine word. See word, information.

WRITE — (1) To transfer information, usually from main storage, to an output device; (2) to record data in a register, location, or other storage device.

Z

ZERO — Numeral normally denoting lack of magnitude. In many computers there are distinct representations for plus and minus zero.

ZONE — (1) Portion of internal storage allocated for a particular function or purpose; (2) three top positions of 12, 11 and 0 on certain punch cards. In these positions, a second punch can be inserted so that with punches in the remaining positions — 1 to 9 — alphabetic characters may be represented.

ZONE, NEUTRAL — Area in space or an interval of time in which a state of being other than the implementing state exists: a range of values in which no control action occurs or a brief period between words when certain switching action takes place. Similar to dead band.

PART II

APPLICATION OF THE MICROPROGRAMMED COMPUTERS

INTRODUCTION

There are four classes of applications which are established for Microprogrammed computers. Each class contains several sub classes which are implemented by control unit programming (firmware) variation.

Any class, augmentation of, or variation of, represents a computer architecture different from one another each offering specific advantages to the intended end application.

General Purpose Computers

- General Purpose Computers With Standard Instruction Set.
- General Purpose Computers With Added Special Instructions.
- General Purpose Computers With Background for Special Data Processing or Input/Output Functions.
- General Purpose Computer With Addition of Special Microprogram Which is Entered and Exits From the Software Program, and Remains Active for a Relatively Long Period of Time.

Special Purpose Computers

- Special Instruction Set.
- Direct Application Microprogram.
- Direct Sequence of Subroutines.
- Interlaced Microprogram Instructions and/or Subroutines With Partial Processing.
- Subroutine Branching According to System States.

Emulator Computer

- Duplication or Approaching Equal Functional Capability With a Preexisting Fixed Instruction Stored Program General Purpose Computer.
- Duplication or Approaching Equal Functional Capability With a Preexisting Special Purpose Computer.

Language Processor

- Direct Execution of High Level Language Statements.
- Partial Execution of High Level Language Statements.

With such a large selection of organizations to choose from, use of a microprogrammable computer provides a very useful method for arriving at the most cost-effective processing or control system, including development, hardware, programming and operating costs.

CLASSES OF APPLICATION

General Purpose Computers

- **General Purpose Computer with Standard Instruction Set.**

In this class of computers the microprogram is designed to fetch instructions from core memory and to execute them by microprogram subroutines. Once started the microprogram continues to loop back on itself, looking for and executing instructions until it sees a halt instruction, or gets into an input mode, and waits for a character. The instruc-

tions share the core memory with data and flags. The coding of the instructions in core bears no particular relationship in format to the micro-commands.

The general flow of firmware functions for the General Purpose Computer is shown in Figure 4.

All operations, including arithmetic, logical, control, shift, branching, jumps, input/output, and register transfer are programmed into micro-program subroutines.

An example of General Purpose firmware is described in detail in Part IV "MICRO 810 Firmware Manual".

- **General Purpose Computer with added special instructions.**

Firmware for a general purpose computer will contain several unused operation codes which can be used for additional instructions. The simplest

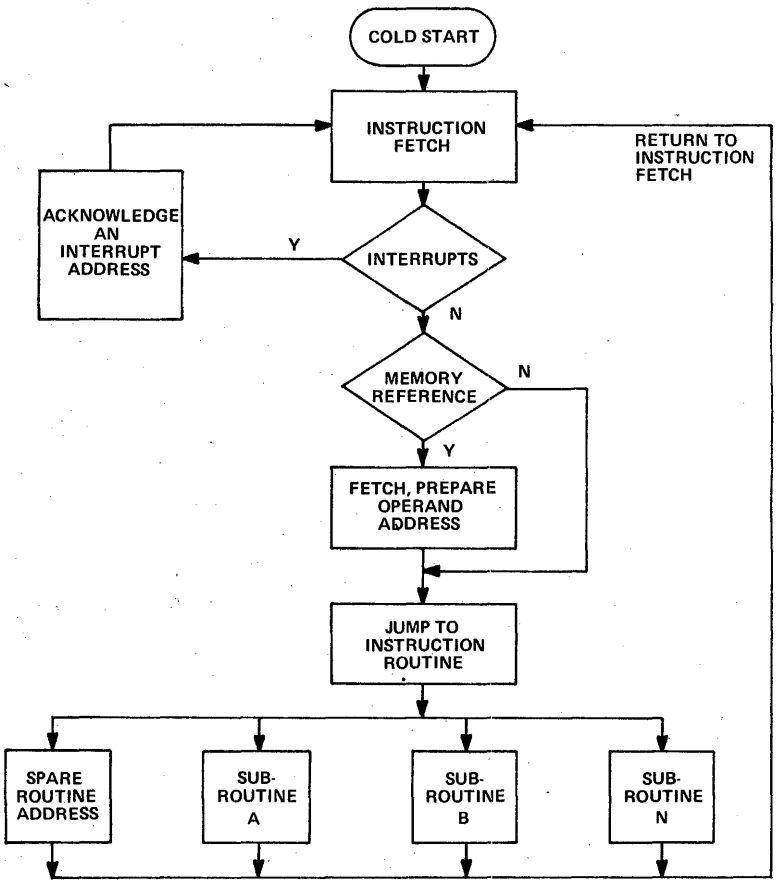


Figure 4. Firmware Function Flow

way to add instructions is to make use of a spare operation code which can easily be converted to a jump instruction to enter a new firmware routine. The new instruction can be either a memory reference or non memory reference instruction. Multiple instructions can be added by using sub-operation codes. Typical instructions which may be added are as follows:

Floating Point Arithmetic.
BCD Arithmetic.
Data Block Manipulation Routines.
Error Code Generation and Checking.
Push Down Stacks and Related Functions.
Special Input/Output Routines for Greater Speed, Increased Functional Complexity, or Simplified Interfaces.
Curve Fitting Routines and Interpolation.
Square, Square Root, and Other Related Functions.
Table Search.
Character Test and Manipulation.
Communications Handshaking.
Filtering and Spectrum Analysis Operations.
Pattern Manipulation and Recognition Functions.

The capacity to add instructions of these types tremendously increases throughput capability and processing power of any General Purpose computer.

The procedure for adding instructions is to define the instruction algorithm, flow chart, and microcode, then to do a timing analysis of the routine to see if it is equal to or less than the maximum permissible interrupt time. If not, the routine must be subdivided to do only a portion of the operation each time it is entered, or to allow testing of interrupts at scheduled times during the routine.

- **General Purpose Computer with Background Microprogram.**

Microprograms can be added to the general purpose computer which run continuously, or on command, and perform some function independent of the software, or indirectly related to the software. These programs are periodically entered as interrupt routines although they don't divert the software program like a normal interrupt does. One example of this is the concurrent input/output routine of the MICRO 810. This firmware transfers a block of data between interface devices and core memory. The concurrent input/output operation is set up and initiated by software, but proceeds independent of the software until the complete block of data has been transferred. Another example is the communications multiplexing function of the MICRO 820 Series computers. This firmware handles up to 32 low speed asynchronous communications lines with character assembly and disassembly performed by firmware. A character queue, and status flags are maintained by the multiplexing firmware to provide a link to the software program. The multiplexer firmware is controlled by the software by means of programmable rates and configurations, enable and disable functions, buffer assignments, and setting or resetting of control flags. Once set up, however, the multiplexer firmware proceeds independently of any specific instructions from the software program. Sampling rates are timed by hardware rate generators.

Other typical background microprograms which fit into this category are as follows:

Analog Data Channel Scanning and Input, or Analog Time Series Sampling.
Matrix Manipulations.
Mapping Functions.
Coordinate Conversions.
Output of Memory Map to Large-Scale Lamp Display.
Statistical Functions, Such as Determining Average, Standard Deviation, and Trends of Large Blocks of Data.
Continuous Data String Manipulations and Code Conversions.

- **General Purpose Computer with Special Microprogram.**

Occasionally there is a requirement for high processing rate (requiring dedicated uninterrupted microprogramming) combined with software flexibility. This combination may be achieved by placing a general purpose instruction set, and a special microprogram instruction set in the same computer. The general purpose or software instruction set is used for relatively slow functions, such as system initialization, monitoring console parameters, updating displays, determination of system states, implementing of relatively slow but complex system control functions, and message preparation.

The microprogram routine is used for high-speed and/or complex data input/output, computation, and control functions. The general procedure for this type computer system is to perform all software functions necessary to set up the microprogram for some segment of its entire job, and then turn complete program control over to the special microprogram until the segment is complete. At this time the special microprogram returns control to the software program. A typical application for this approach is machine tool control. The machine control function involves position sampling, polynomial curve fitting, system control computations, control outputs, timing, status monitoring, and other functions depending on the machine function complexity. Use of microprogramming provides for large increases in processing rate which are necessary to maintain precise control, with complex curves, at specified machine rates.

The software sets up the curves and process rates for a machine processing segment. These curves and rates are interpolated by the microprogram.

Other examples besides machine tool control are as follows:

Sampling a large block of high speed data which occurs in a burst.
Spectrum analysis or filtering with frequency parameters set up by software program.
Contour plotter controller.

Special Purpose Computers

- **Special Instruction Set**

For many applications a standard software instruction set, such as the MICRO 820 may be more sophisticated than needed. Such features as multiple addressing modes, variable word length, concurrent I/O, etc.,

may not be needed. In this case it is possible and desirable to create a special instruction set which will increase throughput rates, make better use of core memory, and provide an instruction set tailored to a specific need. The general organization for this firmware is the same as for the MICRO 820 firmware. However, functions may be deleted or modified, such as testing for interrupts, operand addressing, etc.

Typical applications for a special purpose software instruction set are as follows:

Compiler or Interpreter.
Special Communications Processor.
Automatic Tester.
Sequence Controller.
Business Processor.
Batch Terminal.
Inventory System.
Data collection/reduction system.

- **Direct Application Microprogram.**

In this case, the application program is completely written at the firmware level. This type of program is suitable for dedicated applications, where the processing is relatively simple, but very high processing rates are required, a permanent program is desired, or simplified interface hardware is used, which requires microprogramming for the interface control and data transfer sequences.

Direct application microprograms may occur in one of many different general structures. Three of these which will be described are as follows:

Direct Sequence of Instructions and/or Subroutines.
Interlaced Subroutines with Processing Status Flags and Partial Processing During Each Entry to a Routine.
Branching to Subroutines Dependent on System States.

Each of these will be discussed briefly in the following paragraphs.

In many applications a combination of any two or all three of these methods may be used.

Direct Sequence of Instructions and/or Subroutines.

This approach is the simplest, and potentially the fastest, if it fits the application. The flow diagram for this approach is shown in Figure 5.

The sequence of instruction execution is always the same. The loop may be free running for very simple applications, or it may be initialized by a real time clock where time precision is required.

A typical example of this organization is a dedicated communication line processor where the computer samples and updates a large number of full duplex, serial, asynchronous data lines. The firmware does sampling, character assembly and disassembly, and loads a buffer when a character is assembled. The data is then transferred to another device. A program such as this must be able to handle maximum line load conditions without loss of data. Some of the functions, such as loading the buffer could be spread out over a full character time to smooth out the work load, but then the

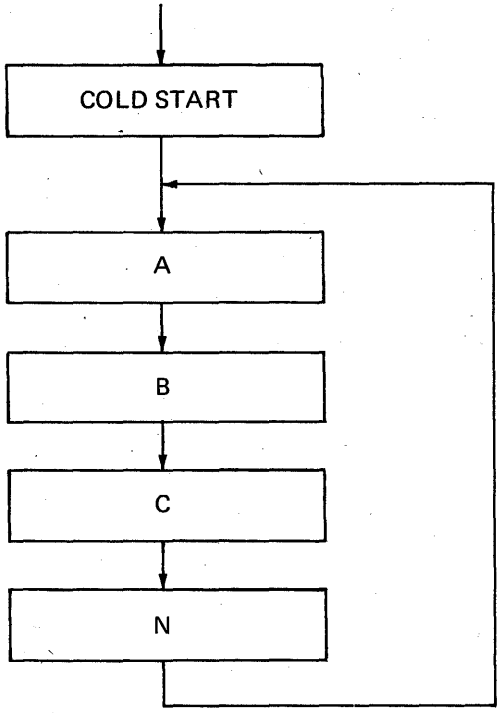


Figure 5. Subroutines or Instructions

program would become more complex, and would become category 2. Statistical averaging shows that the possibility of all lines being active, and in both bit and character sync, is extremely remote. A system like this could handle a line rate times line quantity product which has a theoretical peak instantaneous load of at least 130% of the processing time available and not lose nearly as much data due to processing time limitations as due to random line errors, because the probability of an instantaneous load approaching even 100% is very remote.

Other examples of the direct sequence approach are as follows:

Low Speed Sequence Controller.
Dedicated Synchronous Data Line Concentrator.
Dedicated Device Controller.
High-Speed Status Monitor.
On-Line Performance Monitor.
Auxiliary High-Speed Processor.

Interlaced Microprogram Subroutines with Processing Status Flags and Partial Processing during each entry to a Routine.

Many direct application microprograms involve a number of slow-speed peripheral devices which could be serviced by the microprogram on a part-time basis, or handle data formatted to cause load peaking. Each time a

device, or data value is looked at by the microprogram some different phase of the process may occur, or many times no processing is required at all. The phase may depend on the previous phase, or on the time interval, or a status flag. The microprogram for this class of organization has an execution, or main loop routine which goes from one routine to the next, in sequence and tests status flags to see if the subroutine is to be entered and what processing is required. The general flow is in Figure 6 and the expansion of one functional step is in Figure 7.

In Figure 6 each circle represents a subroutine status, retrieval, test, entry, update and storage function. The boxes represent the routines which are entered from the main loop.

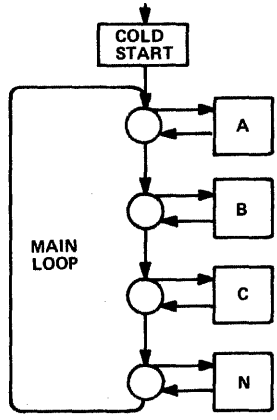


Figure 6. General Flow for Interlaced Subroutines

As can be seen from Figure 7. processing time must be expended to fetch, test, and store flags, pointer, and data, and this reduces the overall processing capacity. However, this approach allows time spreading of the work load which in most cases makes up for the loss in average processing capacity by a large increase in peak load capacity. The two improvements to interlacing are increased peak load capacity and increased overall throughput capacity.

For example, to process a string of serial characters, load peaking comes when a character has been assembled, and when a block has been assembled. In each case there is a time gap until the next character is assembled. Therefore, the work load can be spread out over a number of bit sample times. It can be partitioned according to line number to simplify subroutine organization. When a message block has been assembled, there is even more time until the next block is assembled, so that the time for character checking, buffer moving, etc., can sometimes be spread out over an entire message block. Another requirement might be a code conversion on an assembled character. This could be broken down into subroutines with only a portion being executed at each time interval.

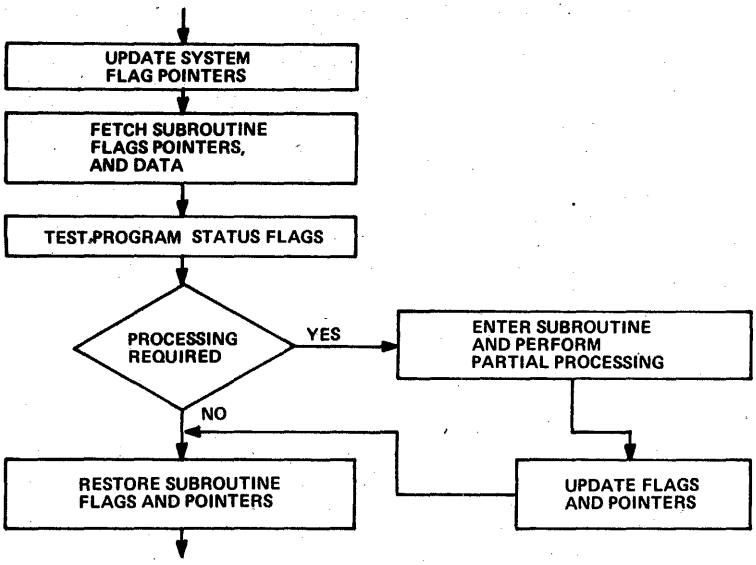


Figure 7. Expansion of One Functional Element of Interlaced Routine Flow Chart

A typical situation which must be handled by interlacing to achieve high throughput is as follows:

In Figure 8 is a block diagram of a microprogrammed peripheral controller.

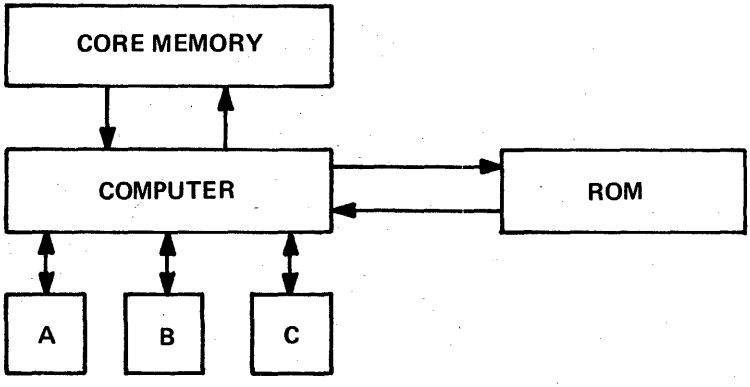


Figure 8. Peripheral Controller Block Diagram

The three devices must run concurrently to achieve maximum throughput. Each of the devices has operations which can be broken up into sub-operations as shown in Figure 9.

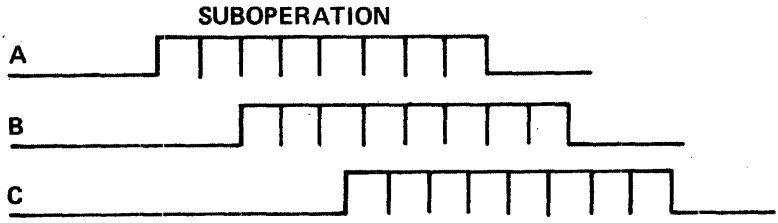


Figure 9. Simplified Processing Profiles

Device B could start as soon as device A has completed some of the sub-operations. Therefore the sub-operations are interlaced. If the devices are asynchronous, and the correspondence between sub-operations is not on a one-to-one basis, the subroutine status tests may at times indicate no processing for one cycle of the microprogram.

Typical applications for interlaced subroutines are as follows:

Batch Processing Terminals.
On-Line Inventory and Audit Systems.
Process Controllers.
General Purpose Communications Terminals.
Monitoring Systems.

Subroutine Branching According to System States.

In some programs branching into subroutines may be a function of the state of a peripheral device or time, or the settings on a control panel. In many of these cases it is not necessary to fetch the status, data or flags of each subroutine in sequence to see if it is to be processed.

For example in a particular machine control application, the processing functions depend on machine temperature, RPM, etc.

For many of these parameters, a truth table may be prepared, which indicates the next program state as a function of the previous and present system states. Then the executive routine tests the states, and determines which subroutines to execute next. Typical examples where this method of microprogramming applies are as follows:

Power Plant Control.
Petroleum System Control.
Chemical Processing Plant.
Interactive Systems.
Numerical Machine Tool Control.
Medical and Laboratory Instrumentation Control.

Emulator Computer

In the truest sense all applications of the microprogrammed computer can be considered emulation. However, as defined here, the emulator computer is the microprogrammed computer with its firmware allowing functional duplication of another computer. Direct emulation of a preexisting general purpose or special purpose computer is practical only if an advantage results. Usually a cost advantage is realized if the preexisting computer is several yers old. In many cases a speed advantage will result.

Many parameters need be considered to determine feasibility and efficiency of a microprogrammed computer emulating any specific general purpose or special purpose computer. Essentially these parameters are:

Complexity and Number of Logical Elements.
Word Size and Number of Hardware Registers.
Maximum Main Memory (Core) Size and Word Length.
Execution Time Required Per Operation.
Input/Output Requirements.

Detailed knowledge of both the preexisting computer and the microprogrammed computer is needed to properly evaluate the feasibility and fit of emulation.

Language Processors

The instruction set configuration of a special purpose computer which is to be programmed at the assembler language level is usually a "hostile" environment to the implementation of compiler level languages. The microprogrammed processor permits the configuration of a minicomputer architecture which is efficient in a compiler language environment. In essence, the utilization of an assembler may be minimized and the compiler statements are in effect interpreted more directly.

For purpose of illustration the implementation of a BASIC compiler in the MICRO 820 computer will be discussed. The MICRO 820 has a general purpose instruction repertoire with conventional assembler and utility software. A single-user BASIC has been developed for the MICRO 820 computer. This BASIC compiler is written in the MICRO 820 assembler language. The early version of the BASIC was installed in the MICRO 820, occupying approximately 7,500 bytes of core memory. A subsequent version of the MICRO 820 architecture is being augmented with special firmware routines such as floating point and other firmware routines. By doubling the micro memory from 768 words to 1,536 words of microcommands, the storage requirement of the compiler in core memory is reduced approximately 66 percent, or from 7,500 bytes to 2,500 bytes. As a result, greater working storage is available for the user and the compile time for the processor is sharply decreased.

This improvement in processor efficiency becomes more significant as the system is extended to perform time share BASIC. An important capability in the implementation of time share BASIC is an operating system which permits the computer to look like a single machine to multiple users. Microdata's time-sharing operating system (MICROshare) initially resides in approximately 4,096 bytes of core memory. Through microprogramming the performance of MICROshare can be sharply increased by con-

verting various features of MICROshare from software (1 user per 8-bit instruction) into firmware (200 ns per 16-bit instruction). When a time-sharing system is under control of a high-performance operating system, it provides for the efficient transfer and execution of programs and files in mass storage (disc memory). System response time is sharply increased; core usage is significantly minimized.

The MICRO 1600 is designed to accommodate all the functions of the MICRO 800 product line. This includes direct function processors, special purpose computers which may or may not require architectural augmentation and compiler language processors. The MICRO 1600 provides a new dimension in the minicomputer field as a compiler language processor. Large arrays of micromemories can be conveniently implemented. The control memory in the MICRO 1600 can be addressed up to 16K X 16. It permits the effective implementation of higher level languages such as BASIC, COBOL, FORTRAN, SNOBOL, ATLAS or equivalent.

APPLICATION EXAMPLES

Automatic Test System

MICRO 811 computers are used to control all functions contained in automatic facilities for routine testing and detailed trouble-shooting of printed circuit boards (Figure 10).

The MICRO 811, intended primarily for testing boards used in the MICRO 800 computer, generates stimulus functions and measures corresponding responses of any circuit boards which are digital in nature. Memory boards which are primarily analog are handled on a special tester.

Components of the automatic test system are the MICRO 811 computer with 8K memory, instruction repertoire and input/output line driver and receiver. The card test unit includes stimulus, response and control boards, power supply, 480-pin patch board receiver, 10 test characters and interface cable.

Software includes a Microdata board test control program, board test tape generator, board test tape, control board and data board. Other options are available for special-purpose uses.

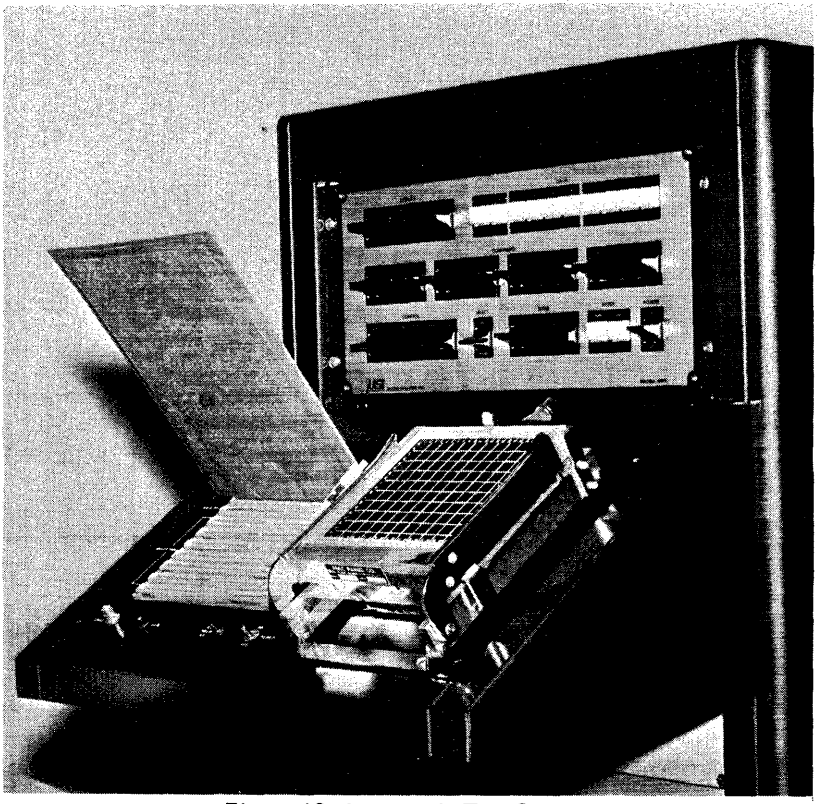


Figure 10. Automatic Test System

Floating Point Processor (Special Purpose)

An ideal use of the MICRO 800 computer is as a floating point processor, since the machine is an extremely high performance processor with the facility for creating specialized instruction sets at the micro step level.

The machine can be mechanized by microprogramming, thus achieving floating point operations at high processing and throughput rates.

As a floating point processor, the MICRO 800 operates on variable word length floating point data. These word lengths may be specified – and changed at any time – to be 8-128-bit fraction plus 8 bits for sign and exponent. Floating point operations use four operating accumulator registers, each 136 bits long, which can be maintained either in core memory or in a special high-speed scratchpad memory.

Data is transferred between accumulator registers and file registers at a high rate of speed by using the microprogram. Maintaining the accumulators in core memory results in low hardware cost, but processing speed is somewhat slower than if the slightly more costly high-speed scratchpad memory is used.

The floating point processor can be integrated into a system in a variety of configurations, each of which has a slightly different equipment requirement, a different mode of operation, requires a different microprogram and yields a different throughput rate.

These configurations are: a peripheral processor to an existing computer; a separate, complete, self-contained floating point computer; a dual processor, sharing memory with a standard processor or computer, or a combined floating point processor and general purpose integer processor such as the MICRO 810.

Fast Fourier Transform Processor (Special Purpose)

MICRO 800 computers are being used to perform spectral analyses of electrical signals using the computational technique known as fast Fourier transform.

Using specially designed fast Fourier transform read-only memories, the MICRO 800 and other components of the system sample and digitize input signals at uniformly spaced time intervals, performs the spectral analysis and processes the results to construct outputs of a specified form.

The output is displayed on one of three devices – an oscilloscope, slow X-Y plotter or fast X-Y plotter. The displays are driven by two 8-bit digital-to-analog converters in a number of modes, including small-interval stairstep, recurrent and single-cycle.

Several functions are displayed, including input signal frame, power spectrum, log power spectrum, amplitude spectrum and phase spectrum.

The system features a special resolution of one part in 200 over the signal input bandwidth and an amplitude error of less than 10%.

The MICRO 800 computers used in the system are configured with a 4096-word core memory, real time clock, power fail protect, I/O expander with 32 inputs and 32 outputs, and ADC-DAC unit with power supply.

Multilane Parking Facility Computer

Multilane parking facilities associated with large modern buildings are relatively complex and are now being automated with various technologies.

The microprogrammed computer provides a significant reduction in the amount of interface hardware, and provides for the permanence of fixed hardwired control systems. Microprogramming provides this capability in all functions:

Fee Calculation.
Customer I.D. Card Validation.
Audit Calculations and Printouts.
Automobile Counts by Lane.
Lane and Area Count Totalizations.
Violation Detections.
Fee Display Update.
Real Time Clock.
Input Customer I.D. Data.

To keep the interfaces simple, all data including treadle pulses, I.D. card information, local data entry and loop detector pulses enter the computer in bit serial form. Display data is on a common bus, with select lines to control distribution.

All data assembly, accumulation, evaluation, storage, retrieval, and control functions are done within the processor, eliminating the requirement for special external hardware to do counting, data assembly, detection logic, and arithmetic functions.

In Figure 1 is a general block diagram showing the types of data going in and out of the processor.

The ticket machines, treadles, loops, and fee displays are in remote locations from the computer and the printer, keyboard, etc., are nearby. The data from the ticket machines consists of contact closures detecting the presence of a ticket, or indicating output, and taking of a ticket. The ticket machine reader inputs serial data which is organized similar to a serial teletype message. This information consists of entry and exit time, or customer I.D.

In the lanes are loop detectors and treadles. Loop detectors input contact closures when they are crossed. The treadle detectors input a series of closures to indicate direction of travel.

For generation of time of day clock, external time of day pulses are used instead of the internal computer clock to maintain time synchronism with the local power company.

Fee display is output in digit serial BCD form accompanied by display select codes, to minimize the number of wires to the display units.

For this example, which represents a medium size parking facility, the local keyboard, printer, and punch is a teletype.

All of the items shown are mounted in the basic computer cabinet.

A system of this type will handle 10-20 lanes with typical numbers of devices such as 25 treadles, 50 loops, and 10 ticket machines.

In a program like this the core memory is used to store data tables, flags, input and output maps, partially processed data, messages, clock, fee totals, lane totals, and area totals. No program is stored in core because the entire program is in firmware.

Data Communications Application, Special Purpose Concentrator

The MICRO 800 computer with a dedicated microprogram used as a concentrator connects a large number of local data terminals to a small group of dedicated trunk line modems on a time share basis. All data messages handled have fixed formats.

The data concentrator is designed to function as a complete data and control interface, performing the following functions:

Data Source Scanning and Queueing.
Modem Poll Monitor and Response.

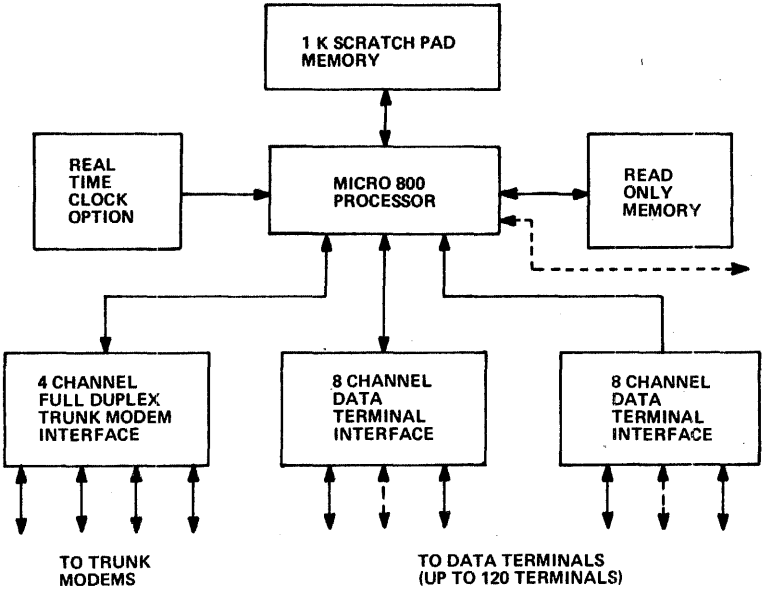


Figure 11. Concentrator Block Diagram

Data Routing Control.
Control Character Examining and Processing.
Header Identification and Stripping.
Hand Shaking With Trunk Modems.
Data Transfer.
Supervisory Data Processing.
Canned Status Message Generation.
Addition of Header Information.
Parity and Block Character Check.
Character Bit Stripping and Adding.

All of these operations are performed with a maximum throughput delay of 3 characters.

The interfaces to the data terminals and trunk modems is in bit serial form, thus simplifying the interface hardware.

The concentrator operates on the 2400 baud synchronous data with the trunk modems and simultaneously provides data clocks to the terminals.

A block diagram of the concentrator is shown in Figure 11. There are two interface types, the trunk modem interface and the data terminal interface. The scratchpad memory is used to store pointers, transfer instructions, flags, request queues, and as a data buffer. All programming is in the read only memory.

Within the MICRO 800, the arithmetic/logic unit is used for character recognition, character shifting, conditional branching, parity and block character checking, bit stripping, I-D to address conversion, queueing preparation and evaluation, code conversion, and other miscellaneous character processing functions.

The MICRO 800 file registers are used for storage of data immediately after it is read in from one of the modules or before reading it out; for storage of status, and control words, for storage of indices, for storage of outputs from the arithmetic unit, and as operational registers for the arithmetic, logic and control functions performed by the MICRO 800.

The firmware instructions are organized in sequences similar to core memory programs with the capability to execute nested subroutines, conditional branching, and various arithmetic control and logic functions necessary to efficiently perform identical functions on multiple data paths with asynchronous timing between paths.

The real time clock option is used to generate an internal timing interrupt at approximately 2500 cps. which controls all bit and character processing cycles within the concentrator. The 2500 cps. rate ensures that no data bit changes at 2400 cps. will be missed by the system.

Numerical Control of Vertical Machining Center

A MICRO 800 computer is being used as the complete numerical control system for a vertical machining center utilizing some innovative machine tool programming techniques.

Consisting of a vertical mill, an automatic tool changer and a digital control system with its associated panels, the mill is completely hydraulic with options for high accuracy laser positioning feedback.

The MICRO 800 positions the table, saddle and spindle (X, Y and Z axis) and controls the direction and speed of rotation of the spindle. The microprogramming feature of the MICRO 800 is used to perform the feedback control of the position and velocity of the axis.

Both linear and circular contouring are provided with a positioning accuracy of 200 micro-inches and velocity of the tool with respect to the workpiece of 0.01 to 200 inches per minute.

The MICRO 800 also controls an automatic tool changer containing 20 tools. All motions are initiated and confirmed by the computer to achieve the necessary sequences.

Machining operations are specified through choice of a manual or tape preparation panel.

The manual panel permits moving the mill in a very simple manner and also provides for entry of tool dimensions used for offset and length compensation.

The tape preparation panel permits programming the machine operations in a sort of "graphical APT" manner. Canned sequences such as drill, bore, tap, mill, etc., are specified along with all pertinent data without regard to tool dimensions. Workpiece dimensions are specified in absolute, relative or trigonometric form. Contours also are specified.

When the computer has validated the requested operation, it assumes control of the machining and can initiate, abort, terminate, test, accept or reject through the tape panel. If accepted by the operator, the operation is preserved on magnetic tape for later use.

After completion of the first workpiece, additional copies are made by merely replaying the cassette magnetic tape with the MICRO 800 control system in the automatic mode. The cassette can be removed from the controller for future use.

Vibration Analyzer (Special Purpose)

The MICRO 800 computer is being used as the heart of a vibration analysis system operating with six channels of frequency shifters and filters, a high-speed multiplexer and analog-to-digital converter, a specially designed control panel and 13 other digital-to-analog converters.

Input to the system is from vibration sensors or other noise sources for which power spectral density plots are desired. Frequency range for analysis is 4 Hz to 6 KHz. Output data, both linear and decibel, is plotted on up to 12 X-Y plotters, and analysis of all six channels is done concurrently.

Using customized firmware, the MICRO 800 computer operates the panel, controls frequency shifting through a voltage controlled oscillator, performs data averaging and maintains system timing.

In addition, the computer calculates both linear and logarithmic (decibels) power spectra, controls the X-Y recorders and can measure the period of an external signal and convert it to frequency (4 Hz to 8 KHz) with an accuracy of 0.1% of indicated frequency over the entire range.

Interface for Campus Central Processor, Satellite Computers

MICRO 800 computers are in use at a major university as the key ingredients of remote terminals interfacing satellite computers at various campus locations to a large-scale central computer (Figure 12).

These "smart terminals" — versatile displays ranging from elegant to not so elegant — provide straightforward interfacing to other computers which handle specific kinds of communications.

Use of the MICRO 800 in this application has eliminated the need for a large amount of specialized hardware at remote sites, and provides an abundance of flexible programming capability through the use of micro-programmed firmware.

With its 220 nanosecond microcommand time and the ability to put input/output and interface functions into firmware provides a far greater throughput rate than is possible with core memory.

A safety factor is provided, too. Storage is fixed in the read-only control memory, insuring that no one, no matter how inexperienced can modify or destroy programs. Storage can be modified according to need by simply exchanging boards.

The MICRO 800 also gives the university a "do-it-yourself" computer capability. Computer center engineers can economically tailor the performance characteristics of the computer in firmware to suit the specific needs of each terminal location.

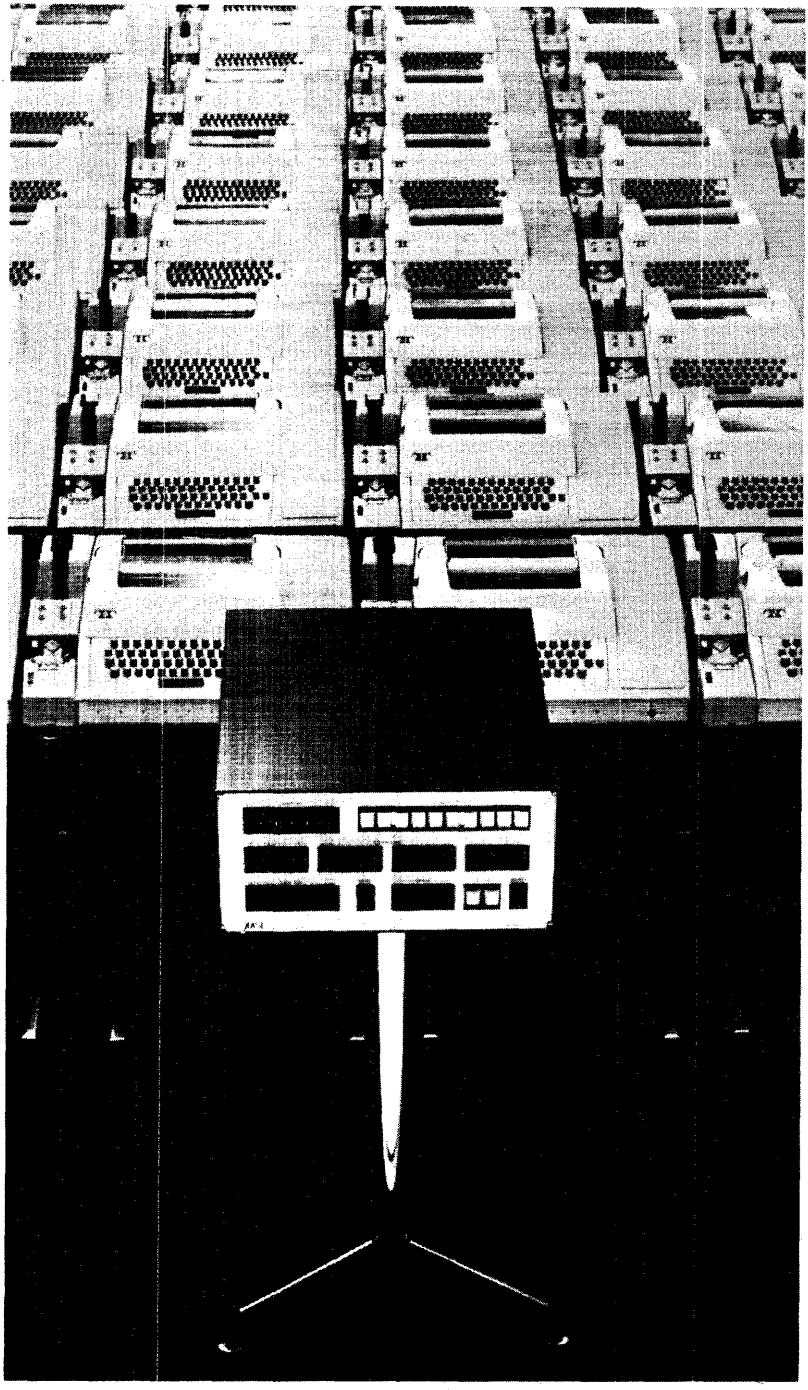
Eventually, the university plans to interface all existing campus computers to its large-scale central processor.

The MICRO 800 represents a general solution to the university's vast number of applications because of its flexibility. Among these applications are interactive display systems and automated systems, which, without the MICRO 800, would have required two completely different sets of hardware.

Figure 12. Campus Interface System

PART III

MICRO 800 USERS MANUAL



CHAPTER 1

SYSTEM DESIGN FEATURES

MICRO 800 is a byte-oriented microprogrammed computer designed for dedicated applications. The functional, mechanical and electrical design of the computer provides a set of functional elements which can be tailored to specific application requirements. The MICRO 800 is a basic set of hardware which, with modification, can be expanded to a series of machines.

The design concepts embodied in the MICRO 800 provide a unique combination of features unavailable in other computer systems. These include:

Microprogramming

The MICRO 800 incorporates a set of commands which exert powerful micro-control over the machine's data manipulation paths and control. Command sequences which form microprograms are stored in a read-only storage. The MICRO 800 can be programmed to emulate instructions of general or special purpose computers or to perform specific applications.

Speed

The machine features a 1.1 microsecond core memory cycle time and a 220 nanosecond command execution time. This speed permits rapid emulation of macro instructions and can be used to minimize interface hardware by applying the speed of the machine to interface functions.

Modularity

The modular electrical and mechanical design has all the flexibility needed to apply the MICRO 800 to a wide range of applications. The modular design of the core memory read-only storage, processor options, and input/output elements permits expansion of the system as required. The compact 8¾-inch-high enclosure has a number of spare circuit board slots and ample power for system and peripheral interfaces even when the processor is fully expanded.

Low Cost

The MICRO 800 uses TTL monolithic integrated circuits, including a large number of the medium scale integration type for savings in parts and assembly time. The use of a read-only memory for control further reduces the number of circuits that might otherwise be required to provide similar functional capability. Packaging and powering of the MICRO 800 is designed for significant cost savings.

Software

Programs for the MICRO 800 include an assembler written in FORTRAN for use on large-scale computers, utility programs for generating the read-only memory maps, processor and memory diagnostics, and a simulator program for checking our microprograms. See Chapter 6, "Programming Systems."

GENERAL CHARACTERISTICS

The advanced features and operating characteristics include:

- Memory addressing to 32K.
- 1024, 4096 or 8192 byte memory modules.
- 32,768 bytes of memory in basic 8¾-inch-high cabinet.
- 1.1 microsecond memory speed (full cycle).
- 8 or 9 bit memory bytes for efficient character handling.
- Direct memory access (DMA) option.
- 16 general-purpose eight-bit file registers.
- Up to 1024 words of read only storage in 256 word modules with optional expansion capability to 2048 words.
- 220 nanosecond microcommand execution time.
- 15 basic commands.
- Three versions of control consoles.
- TTL integrated circuitry.
- Operating temperature range 0°C to 50°C.
- Dimensions: 8¾ inches high, 19 inches wide, 23 inches deep.
- Power: 115/230 vac, 50-60 cycle.
- Four versions of read only memory.

SYSTEM ORGANIZATION

The MICRO 800 is a bus organized machine built around a file of 16 programmable registers and employing microprogrammed control. The basic elements of the machine are shown in the block diagram of Figure 13.

The machine executes 15 basic commands with many variations. All commands are 16 bits in length and are in one of three formats. MICRO 800 programs, which are known as microprograms, are placed in a read-only memory and thereafter become a part of the machine's hardware. The program can be changed by replacing the printed circuit boards containing the read-only memory. The commands read out of the read-only memory control all aspects of the operation of the basic machine and are executed in a single machine clock cycle.

The eight-bit arithmetic/logic unit performs all manipulation of data, including: addition, subtraction, logical AND, logical OR, logical exclusive OR, and one-bit left and right shifts. The output of the logic network is the A-bus which is the input to the files and other machine registers. All byte data movement is performed over this bus. The output of the file is one of the inputs to the arithmetic/logic unit; the other is the B bus. Inputs to this bus are determined by the command, its options, and the I/O mode. Bus inputs are the true output of the T register, the complement output of the T register, the input bus and the eight-bit literal contained in some commands.

The memory data and address busses communicate between the core memory modules, the processor and the DMA. Either the processor or the DMA may operate with the memory, with the DMA having top priority.

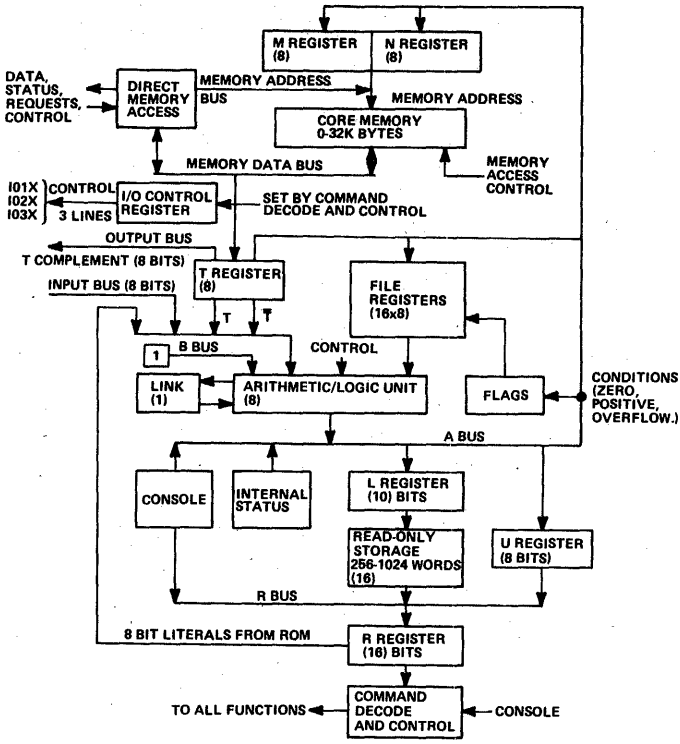


Figure 13. MICRO 800 Block Diagram

The registers, file, arithmetic/logic unit and bussing are organized onto two identical "data" printed circuit boards—a four-bit slice of the machine on each board. All command decoding, control, clock generation and memory timing are located on a single "control" board. Each 256 words of diode read-only storage requires a single board and the core memory a pair of boards. The fusable diode, and bipolar ROM's contain up to 2048 instructions on one board.

REGISTERS AND FILE

There are eight registers and 16 file registers, each of which has a specific use in the processor, while the file is used for general storage and flags.

T Register

The eight-bit T register serves as the operand register for most of the operate class commands, and as a buffer register for output and memory operations. Both the true and complement output of the T register can be gated to the B-bus as an operand. When both the contents of T and its complement are selected as operands, the effective operand is all 1-bits; if neither is selected the operand is all 0-bits. The T register can be loaded

from core memory on a read instruction, directly from read-only memory using a load T instruction or from a file register by designating T as the destination register of an operate class command. All programmed outputs including both control and data bytes go out via the T register.

M Register

The eight-bit M register contains the seven high order bits of the processor memory address. This register is gated onto the memory address bus at all times except when a DMA operation is in process. The M register can be loaded directly from ROM using a load M command, or can be loaded by designating M as the destination register of an operate class command. The M register is cleared on a load N command.

N Register

The eight-bit N register contains the eight low order bits of the processor memory address. This register is gated onto the memory address bus at all times except when a DMA memory operation is in process. The N register can be loaded directly from ROM using a load N command, or by being designated as the destination register of an operate class command.

L Register

The 10-bit L register is the machine's program counter and contains the read-only storage address of the next command to be executed, unless altered by a jump command. The eight low order bits of the L register are a counter which is incremented by one at each clock time when the processor is running unless there is a command execution delay imposed. L is loaded by a load L command, or as a destination register of an operate class command.

U Register

The eight-bit U register is used to modify the output of the read-only storage. For commands with 0's in the four high order bits of 1's in bit 15 and the three low order bits, the contents of the U register is inclusive-ORed with the eight high order bits of the read-only memory output as it is gated into the R register. This allows for dynamic modification and changing of operation codes and file register designators. U is loaded by a load U command or as a destination register of an operate class command.

R Register

The 16-bit register holds the present command being executed. Its output is decoded and controls the operation of the processor at each clock time.

LINK Register

The one-bit LINK register holds the adder's high order carry from add, subtract, and compare commands and the shifted off end bit from the shift command.

I/O Control Register

This three-bit register generates the control signals for the I/O bus. Seven separate control signals can be developed by decoding of the register outputs. It is loaded and cleared by a control command, placing the timing of

I/O control signals under command control. There are three output modes and four input modes. The high order bit of the register is the input flag. When this bit is a 1-bit the input bus is substituted for the T register when it is selected and the input bus is the source of data when executing an external I/O control command.

File Registers

The file consists of 16 eight-bit operational registers. All commands except the load register with OP code (1) specify a file register to be operated on or to provide an operand or both. All file registers are functionally identical except for file register 0 which contains eight flags, and cannot be used for general storage. The flags of file register 0 are given in Table 2.

Table 2. File Register 0 Flags

BIT	FLAG
0	– Overflow Result Condition
1	– Negative Result Condition
2	– Zero Result Condition
3	– Concurrent I/O Request Line
4	– Internal Interrupt
5	– I/O Reply Line
6	– Serial Teletype
7	– External Interrupt Line

CORE MEMORY

The magnetic core memory is organized into pluggable modules of 4096 or 8192 bytes. The memory is addressed at the byte level and each byte contains 8 or 9 bits. The ninth bit is devoted to the memory parity bit option. Memory may be expanded up to four modules (32,768 bytes) within the basic 8¾-inch cabinet.

The memory is operated in read/write and full/half cycle operations. The full-cycle memory timing is five 220 ns clock cycles (1.1 microseconds); the half-cycle timing in the system is three clock cycles (660 ns). For a read operation, the accessed data is placed in the T register two clock cycles after the start of the memory operation. Full cycle regeneration of the data in the memory does not require the use of the T register and T may be modified by the microprogram before completion of the restore part of the cycle.

The four memory modules plug into the memory address and data busses which run vertically on the back-plane. A spare board slot wired for access options which can include a DMA I/O channel and a special DMA peripheral controller.

CONTROL MEMORY

The read-only memory provides the storage for commands and constants of the microprogram. Its output is gated into the R register where it controls the operation of the machine at the next clock time.

The read-only memory is organized into modules of 256 words contained on a single printed circuit board. Each of the four possible read-only memory boards receives an address from the L register via the read only memory address bus, and the selected board gates its addressed contents onto the read-only memory data bus where it is entered into the R register.

The memory is constructed of diodes with a diode being placed at the proper coordinates for 1-bits in the commands. The commands are designed to use 0-bits as the normal case to reduce the number of diodes on the board; on the average, about one-third of the total bits contain 1's.

The read-only memory is always accessed for the next command while the current command is being executed. This lookahead achieves faster command execution time. When the sequence of command execution is altered by a jump or skip, an additional cycle must be taken to perform an access before the next command is executed. When the machine is halted, the L register contains the address of the first command to be executed when operation is started.

ARITHMETIC FUNCTIONS

The MICRO 800 uses a 2's complement binary number system. The registers and memory cells are 8 bits in length. For convenience of programming, entering data, printing out, and preparing punched paper tape, the 8 bits are organized into two hexadecimal digits. The hexadecimal digits, with their decimal and binary equivalents, are as follows:

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Throughout this document hexadecimal numbers are identified with single quotes:

'33'
'AA'

For additional functions, the two numbers are added directly with the carry out of the most significant bit going to LINK, and overflow setting the overflow bit, if designated in the command.

For subtraction, one number is converted to a 2's complement and added to the other.

For single byte operations, with a 2's complement number system, the range of numbers is as follows:

Binary	Hexadecimal	Decimal	
01111111	'7F'	+127	POSITIVE
00000001	'01'	+ 1	
00000000	'00'	0	
11111111	'FF'	- 1	NEGATIVE
11111110	"FE"	- 2	
10000000	'80'	-128	

↑ Sign bit

Examples of Arithmetic Functions:

Addition: A + B = C

Example #1

Decimal	Hexadecimal		Binary
3	'03'		00000011
+5	'05'		00000101
8	'008'	Link = 0	000001000
	↑ Link	Overflow = 0	↑ Link

Example #2

Decimal	Hexadecimal		Binary
65	'41'		01000001
+82	+'52'		01010010
147	'093'	Link = 0	010010011
↑ Beyond normal range of +127	↑ Link	Overflow = 1	↑ Link ↑ Sign becomes negative

On example #2 the overflow occurred because the range of positive numbers was exceeded. LINK was 0 because the carryout of the add was 0 even though overflow occurred.

Example #3

Decimal	Hexadecimal 2's Complement	Binary (2's Complement)
- 93	'A3'	10100011
+(-105)	+'97'	+10010111
-198	'13A'	100111010
Overflow occurs because -198 exceeds the maximum negative number.	↑ Link = 1 Overflow = 1	↑ Link ↑ Effective 8 bit result is a positive number.

Example #4	Decimal	Hexadecimal 2's Complement	Binary (2's Complement)
	45	'2D'	00101101
	+(-62)	+'C2'	11000010
	-17	'OEF' Link = 0	011101111
	↑	↑ Overflow = 0	↑
	No overflow, within number range.	Link	Link

Example #5	Decimal	Hexadecimal 2's Complement	Binary (2's Complement)
	77	'4D'	01001101
	+(-27)	+'E5'	+11100101
	+50	+'132'	100110010
		↑	↑
	No overflow within number range.	Link = 1	Link = 1
	Link = 1 No overflow		

In general, arithmetic overflow occurs whenever the number range (+127 to −128) of the MICRO 800 is exceeded on an arithmetic operation. As can be seen in the examples, the link bit may be set even though an overflow did not occur. This is the result of using a 2's complement number system.

To mechanize overflow detection in the MICRO 800 use is made of the fact that when there is an overflow, the carry into the most significant bit does not equal the carry out of the most significant bit. This can be shown as follows:

Overflow Examples:

Decimal	Hexadecimal	Binary
127	'7F'	01111111
+ 1	'01'	00000001
128	'080'	010000000
↑		↑↑
Overflow because the positive range was exceeded.		The carry into bit 7 = 1 The carry out of bit 7=0 Therefore overflow occurred. Link = 0

Decimal	Hexadecimal	Binary
126	'7E'	01111110
+ 1	'01'	00000001
127	'07F'	01111111
		0 carry in
		0 carry out

No overflow because positive range not exceeded.

Carry into bit 7 = carry out of bit 7.
Therefore no overflow.

Decimal	Hexadecimal	Binary
-93	'A3'	10100011
+(-105)	+'97'	+10010111
-198	'13A'	100111010
	Link	Carry into bit 7 = 0
		Carry out of bit 7 = 1

Overflow

Therefore overflow occurred.

Decimal	Hexadecimal	Binary
77	'4D'	01001101
-27	+'E5'	11100101
+50	+'132'	100110010
	Link	Carry into bit 7 = 1
		Carry out of bit 7 = 1

No overflow

Therefore no overflow.

Decimal	Hexadecimal	Binary
93	'5D'	01010010
+105	+ 69	01101001
198	0C6	011000110
		01

Overflow

Carry in does not = carry out. Therefore overflow occurred.

For 2's complement, the number is first converted to 1's complement, then 1 is added.

Example – 2's complement of '35'

	'35' hex =	00110101	binary
2's comp.		11001010	ones complement
	'CB' hex=	11001011	ones complement +1

STATUS AND CONDITION FLAGS

Internal Status

Eight internal status bits are provided to designate a particular internal interrupt condition. When any of the internal status bits are a 1-bit, the internal interrupt flag (bit 4) in file register 0 is also a 1-bit. This flag is tested by the microprogram to detect the presence of the internal interrupt condition. The internal status bits are entered via the A-bus into the selected file register by a control command, at which time the status bits are cleared. The eight internal status bits have the assignments given in Table 3.

Table 3. Internal Status Bits

BIT	INTERNAL STATUS
0	Console Interrupt
1	DMA Termination
2	Real-Time Clock Interrupt
3	(Spare)
4	Memory Parity Error Interrupt
5	(Spare)
6	Console Halt Switch
7	Power Fail/Restart Interrupt

All the internal status bits except the console interrupt and halt are associated with processor options and may be reassigned for special applications.

Condition Flags

The overflow, negative and zero conditions resulting from an operation involving the arithmetic/logic unit may be stored in file register 0 (see Table 3). The condition flags are updated for command 7 and for commands 8, 9, B – F if bit 4 is a 1-bit. These condition flags can be tested by the microprogram for implementing various conditional operations. Definition of the condition flags is as follows:

Overflow – The Overflow condition flag stores the arithmetic overflow condition during an add, subtract or copy command. The overflow condition flag stores the shifted off end bit during a shift command. Arithmetic overflow occurs, when the result exceeds the range of the computer's 8-bit registers.

Negative – The Negative condition flag stores the high order bit of the result on the A-bus, since the 2's complement number system uses the most significant bit as the sign bit.

Zero – The zero condition flag stores the zero test condition of the result on the A-bus. When the link control (bit 7) of the operate commands is a 1-bit, the zero condition flag may not be set to indicate a zero result unless it is already set; it may be reset to indicate a non-zero result. This provides

a linked zero test over multiple bytes of a variable byte operation. For a detailed description of linked zero test, refer to the description of the Add command.

COMMAND TIMING

Each command is executed in a single clock cycle time, although execution may be delayed because of core memory or read-only memory operations. The system clock rate is 4.55 mHz, and the clock cycle 220 nanoseconds.

Memory Busy Delays

If the memory is busy (because of processor or DMA operation) at the time a read or write memory command or a command which will modify the M or N registers is to be executed, execution is delayed until the memory operation is completed. These commands are executed on the last clock of the memory half or full cycle. If a DMA request is pending at the time a read or write memory command is to be executed, execution is delayed to give the DMA memory priority.

Memory Data Delays

Operate class commands which select the contents of either the T register or its complement during the first two cycles of a processor memory read operation are executed during the third cycle of the read operation. This allows time for the accessed byte to be placed in the T register.

The memory delays are explained in more detail in the description of the memory command.

Read-Only Memory Delays

An extra cycle is required for command execution because of the look-ahead nature of the read-only memory for the following conditions:

- Jump command.
- Test If zero command when a skip occurs.
- Test If not zero command when a skip occurs.
- Compare command when a skip occurs.
- Operate class commands which have the L register designated.

CHAPTER 2

MICROCOMMAND REPERTOIRE

This section contains descriptions of all MICRO 800 commands. With each description is a diagram showing the format of the command and its operation code, given in hexadecimal. Above each diagram is the command's mnemonic code and the name of the command. Under each diagram is a description of the command, followed by a list of the registers and indicators that can be affected by the command. The timing of each command is one clock cycle (220 ns) unless the L register is designated as the destination of the result, in which case the command execution time is two cycles.

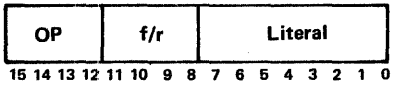
COMMAND FORMATS

There are three basic command formats. Each command is 16 bits in length and is contained in a single read-only memory location.

The formats are literal commands, operate commands and execute commands.

Literal Commands

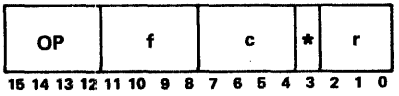
The literal class commands have the following format:



In this format the operation code occupies the four high order bits. Bits 11-8 contain either a file register designator (f) or a register or control group designator (r). Bits 7-0 contain an eight-bit literal which is transferred as an operand to the B-bus.

Operate Commands

The operate class commands have the following format:



In this format the operation code occupies the four high order bits. Bits 11-8 contain a file register designator (f) which specifies one of the 16 file registers to be used in command execution. Bits 7-4 contain control option bits (c) which are unique to the specific command. When bit 3 is one, the result of an operate class command is inhibited from being placed in the designated file register. Symbolically, this is specified to the program assembler by appending an * to the command mnemonic. The register designator (r) in bits 2-0 specifies a processor register destination to receive the result of the operation.

Since there is only one file register selected at a time, the only file register that can receive the result of a particular operate command is the same file register selected for the operand. The register's identifier is added as a second character of the command mnemonic. The register codes (Table 4) are:

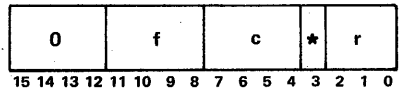
Table 4. Register Designators for Operate Commands

Designator	Mnemonic	Register
0		none
1	T	T Register
2	M	M Register
3	N	N Register
4	L	L Register-addresses: 000-0FF and 200-2FF
5	K	L Register-addresses: 100-1FF and 300-3FF
6	U	U Register
7	S	U Register ORed into command (except for Control command)

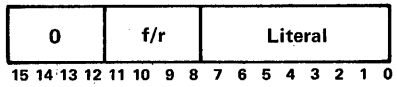
Execute Command

The execute command causes the contents of the U register to be ORed with the eight high order bits of the command to form an effective command. This operation is also performed when r=7 for the operate class commands. The execute command has zero-bits in the four high order bits. The remainder of the command has the format required for the effective command to be executed.

Formats for Execute Commands



If U contains Operate command OP code.



If U contains Literal command OP code.

Literal Commands

The literal commands, listed by OP code are as follows:

OP Code	Command
1	Load Register
2	Load File
3	Add to File
4	Test Zero
5	Test Not Zero
6	Compare

The literal commands are used to load constants into various MICRO 800 registers, to test for bit configurations and data values in file registers, and to load or add constants to file registers. Eight of the 16 bits are used as command, and the other 8 are available as data.

Operate Commands

The operate commands, listed by OP code are as follows:

OP Code	Command
7	Control
8	Add
9	Subtract
A	Memory
B	Copy
C	OR
D	EXCLUSIVE OR
E	AND
F	SHIFT

The operate commands are used to control the flow of data in or out and through the MICRO 800 computer, and to perform the arithmetic and logic functions in the computer.

With this powerful command set it is possible to implement all of the data handling and control functions of a larger computer.

TERMS AND SYMBOLS USED IN THE COMMAND DESCRIPTIONS

(f_1)	Contents of file 1.
$(f_1) \rightarrow T$	Contents of file 1 to T register.
....	Indeterminate value or function.
'AA'	Hexadecimal number in flow chart.
X'AA'	Hexadecimal constant in assembly language statement.

Affected Register States

For each command certain registers are modified. These are described in examples as affected registers.

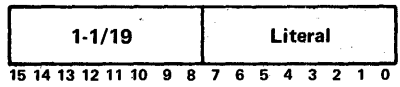
$\wedge$	LOGICAL AND
$\vee$	LOGICAL OR
$\not\vee$	LOGICAL EXCLUSIVE OR

L '024'	Effective Address of L register as used in examples. (Because of the lookahead feature of the MICRO 800, the actual L address is one higher than indicated in the examples.)

MICROCOMMANDS—FORMATS, DESCRIPTIONS, AND EXAMPLES

The formats of the examples for each command have been selected to facilitate explanation of that particular command. Because of the differences in characteristics and utilization of the various commands, and associated data patterns, the example formats are different for each command category.

Command	Mnemonic	OP Code
Load T	LT	11



The contents of the eight-bit literal field are placed in the T register. The condition flags and LINK register are not affected.

This command is used to provide constant data values, bit patterns for comparison tests, masks, and input/output control codes, which are most conveniently used in the T Register.

The T register is also modified by designation as destination register in operate commands.

Example: Load T with hexadecimal value 'AA'

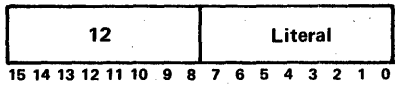
L	Machine Code	Assembly Language	Flow Chart Notation
'024'	'11AA'	LT X'AA'	'AA' → T

Affected Register States:

Register	Before	After
L	'024'	'025'
T	....	'AA'

Command Execution Time – 220 nanoseconds.

Command	Mnemonic	OP Code
Load M	LM	12



The contents of the eight-bit literal field are placed in the M register. The condition flags and LINK register are not affected.

This command is used to set the M register for accessing dedicated core locations. The M register is also modified by designation as destination register in operate commands.

Example: Load M with page address hexadecimal value '55'

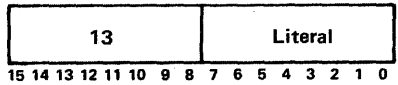
L	Machine Code	Assembly Language	Flow Chart Notation
'134'	'1255'	LM X'55'	'55' → M

Affected Register States:

Register	Before	After
L	'134'	'135'
M	....	'55'

Command Execution Time – 220 nanoseconds.

Command	Mnemonic	OP Code
Load N	LN	13



The contents of the eight-bit literal field are placed in the N register and the M register is cleared. The condition flags and LINK register are not affected.

This command is used to set the N register for accessing dedicated core locations. If the location is in page 0 of core ('0000'–'00FF') only this command is required to set both the M and N registers, since M is automatically cleared. If M is not to be page 0, then N must first be set, followed by M.

Example: Load N with address hexadecimal value "F" and set M = '00'

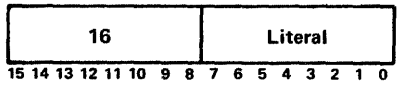
L	Machine Code	Assembly Language	Flow Chart Notation
'235'	'13FF'	LN X'FF'	'FF' —►N '00' —►M

Affected Register States:

Register	Before	After
L	'235'	'236'
M	----	'FF'
N	----	'00'

Command Execution Time: 220 nanoseconds.

Command	Mnemonic	OP Code
Load U	LU	16



This command is used to place specific command codes into the U register, which is used in conjunction with general function EXECUTE class commands. The U register can also be modified by being designated as the destination register in an operate command. The differences in utilization of these two approaches for modifying the U register are described in a later paragraph which discusses U register applications.

Whenever the U register is modified it is necessary to place at least one command between the modifying command and a command which uses the U register as an input. Otherwise an undefined value of U may be used.

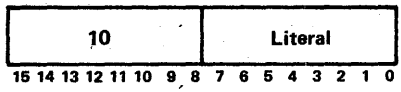
Example: Load U with hexadecimal value '84'

L	Machine Code	Assembly Language	Flow Chart Notation
'155'	'1684'	LU X'84'	'84'⟶U

Affected Register States:

Register	Before	After
L	'155'	'156'
U	---	'84'

Command	Mnemonic	OP Code
Load Zero Control	LZ	10



When this command is executed, a pulse called CGOX of approximately 200 nanoseconds width is generated. CGOX is available on the I/O and option board connectors of the MICRO 800. During CGOX, the literal value is on the A-bus, which is available to the option board. An 8 bit control latch can be set on the option board by this command and used for any purpose, such as enabling counters, interrupts, or control lines.

On I/O boards, a literal value must be first placed in T, and then strobed out with CGOX. CGOX can be used without the literal to initiate special I/O sequences.

Example: Set bits 1 and 2 of special control latch on option board using Load Zero Control.

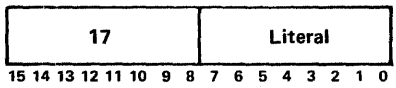
Bit pattern 00000110 – '06'

L	Machine Code	Assembly Language	Flow Chart Notation
'055'	'1006'	LZ X'06'	'06'⟶Z

Affected Register States:

Register	Before	After
L	'055'	'056
Special	———	'06'

Command	Mnemonic	OP Code
Load Seven Control	LS	17



The eight bits of the literal perform control functions as described below.

1700 – No operation.

1701 – Enable serial teletype. The serial teletype input is gated into bit 6 of file register 0. The serial TTY value is available all the time.

1704 – Disable external interrupts: Recognition of external interrupts is inhibited.

1708 – Enable external interrupts: Recognition of external interrupts is enabled.

Note: Commands 1704 and 1708 are meaningful only when the option board has been installed in the MICRO 800, and a modification has been made to the computer backplane. These commands set and reset an interrupt input enable latch on the option board. Without the option board the external interrupt line is always enabled.

1710 – Disable real time clock: The real-time clock and interrupt are disabled.

1720 – Enable real time clock: The real-time clock and interrupt are enabled.

Note: These commands are meaningful only when the option board containing the real time clock is installed. When the clock is enabled it is preset to its wired value. Each time the real time clock cycles, it sets internal status bit 2, which remains set until sampled by the microprogram.

1740 – Spare.

1780 – Halt: The processor is halted.

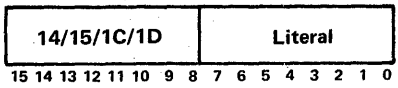
When the processor halts, all clocks stop, except for clock 6, and the L register remains at the next value after the halt command. Depressing the run switch will start the program at the next instruction after the halt command.

Command Execution Time – 220 nanoseconds.

Non-conflicting commands can be executed simultaneously. For example, enable external interrupts can be combined with enable real time clock. The bits of the literal parts of the commands are ORed to produce the hexadecimal code.

Example:	Machine Code	Literal Bits
Enable Interrupts	1708	0000 | 1000
Enable Real Time Clock	1720	0010 | 0000
Composite Command	1728	0010 | 1000

Command	Mnemonic	OP Codes
Jump (Also called Load L)	JP	14, 15, 1C, 1D

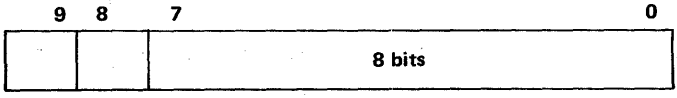


The contents of the eight-bit literal are placed in the eight low order bits of the L register; the content of bit 8 is placed in L_8 and the content of bit 11 is placed in L_9. The location of the next command to be executed is at the address specified by the new contents of the L register. The execution time of the command is two cycles. The jump operation codes for the four 256-word pages in read-only memory are as follows:

14 – Jump to locations 000-0FF (page 0)
15 – Jump to locations 100-1FF (page 1)
1C – Jump to locations 200-2FF (page 2)
1D – Jump to locations 300-3FF (page 3)

In order to fully explain this command, a detailed description of the L register follows:

L Register Organization



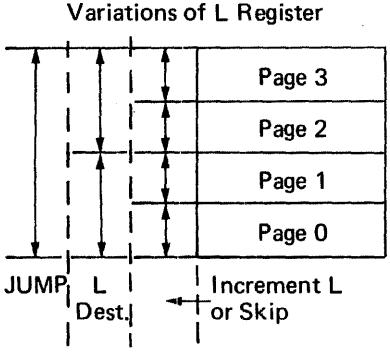
Bits 0 to 7 act somewhat like a counter in that they are incremented like a counter after each command execution except conditional skips, jumps, or operate commands containing L or K as a destination. If the L count is at XFF, and the next command causes L to be incremented, the L count will go to X00, with no indication of a carry. If a command causes L to skip, L will go from XFF to X01.

To change pages, it is necessary to change bit 8 or 9. Bit 9 can be changed only with a jump (literal to L) command. With the jump command, any part of L can be reached.

Bit 8 can be changed with either a jump command or by designating the L register as the destination register in an operate command.

As shown in Table 4, a destination designator of 4 or 5 affects the L register. The designator 4 causes bit 8 to reset, and 5 causes bit 8 to set. In the assembly language mnemonics, a 4 is labeled L, and a 5 is labeled K.

The various methods of changing L are shown in the following read-only map outline.



L Register

Page		Address Within Page
1	1	
1	0	
0	1	
0	0	

Since L is always addressing the next command to be executed, any condition, such as a skip, jump, or L destination results in a clock cycle skip because the "next" command must be discarded for a new "next" command.

Examples:

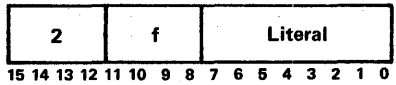
L	Machine Code	Assembly Language	Flow Chart Notation
1) Jump to page 0 location '33'			
'021'	'1433'	JP X'033'	'033'→L Sometimes just shown as a line from one block to another in flow chart.
2) Jump to page 2 location '46'			
'150'	'1C46'	JP X'246'	'246'→L
3) Jump to page 3 location '31'			
'230'	'1D31'	JP X'331'	'331'→L

L Register States:

Example	Before	After
1	'021'	'033'
2	'150'	'246'
3	'230'	'331'

Command Execution Time — 440 nanoseconds.

Command	Mnemonic	OP Code
Load File	LF	2f



The contents of the eight-bit literal field are placed in the file register designated by f. File register 0 cannot be loaded by this command. The condition flags and LINK register are not affected.

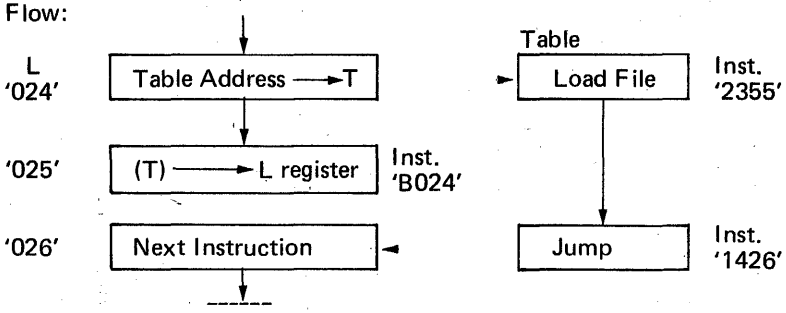
This command is used for initializing or clearing file registers. It is also used for setting relative and absolute jump addresses into files. It can also be used as part of a table look-up routine. Another application is for setting indirect return addresses into files.

A brief description of a table look-up technique follows:

The table look-up function can be implemented using a combination of load file, jump, and operate class (L destination) commands.

A table of values is stored in the ROM which are accessed by jumping to a selected command using an operate class command with an L destination. The selected command is a load file command. After the load file command there must be a jump command to get back to the program routine.

Flow:



If, because of a large table, it is necessary to conserve memory locations in the ROM, a number of load file commands could be grouped with each jump command. This will temporarily tie up as many files as load file commands.

Example of load file command:

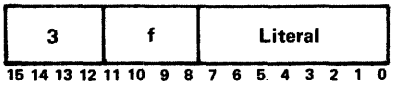
Load file 3 with '55'

L	Machine Code	Assembly Language	Flow Chart Notation	
'025'	'2355'	LF 3, X'55'	'55'	f3

Affected Register States:

Register	Before	After
L	'025'	'026'
file 3	———	'55'

Command	Mnemonic	OP Code
Add to File	AF	3f



The contents of the eight-bit literal field are added to the contents of the file register designated by f and the sum replaces the original contents of the file register. Subtraction is performed by placing the 2's complement of the number in the literal field. The condition flags and LINK register are not affected. File 0 may not be selected by this command.

This command is used whenever it is desired to add a number other than 1 (in which case the operate class add is used) to a file register. Specific cases are where a file is used for a pointer or to update the U register and changes of 2 or greater are required. Another use is to clear out higher order bits from a register. This command can also be used to set a flag bit in a file without resetting the other flag bits.

Examples:

1) All '2A' to file 3 which contains '31'
2) Subtract '03' from file 5 which contains '54'
3) Set flag bit 6 in file 9 which has flag bit 1 set

Example Number	L	Machine Code	Assembly Language	Flow Chart Notation
1)	'015'	'332A'	AF 3,X'2A'	$(f_3)+'2A' \rightarrow f_3$
2)	'105'	'35FD' ①	AF 5,X'FD'	$(f_5)-'03' \rightarrow f_5$
3)	'250'	'3940' ②	AF 9,X'40'	$(f_9)+'40' \rightarrow f_A$

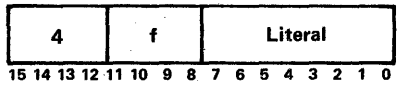
① 2's complement of '03'
② Hexadecimal equivalent of bit 6 = 1

Affected Register States:

Example Number	Register	Before	After
1)	L	'015'	'016'
	file 3	'31'	'5B'
2)	L	'105'	'106'
	file 5	'54'	'51'
3)	L	'250'	'251'
	file 9	'02'	'42'

Execution Time — 220 nanoseconds.

Command	Mnemonic	OP Code
Test If Zero	TZ	4f



If, for all the 1-bits of the literal field, the corresponding bits of the file register designated by f are 0-bits, the next command is skipped. The condition flags, LINK register and the file register are not affected. If the skip is taken, the timing of the command is two clock cycles.

This is a conditional branch type of command designed to test for the following conditions or functions existing in the referenced file register: negative or positive number, odd or even number, interrupt or internal status bits, sense switch bits, condition flags set or not set, teletype input bit set or not set. Since all of the selected bits must be 0, this is a logical AND type function. If a test bit is 0, the corresponding bit in the file does not affect the skip.

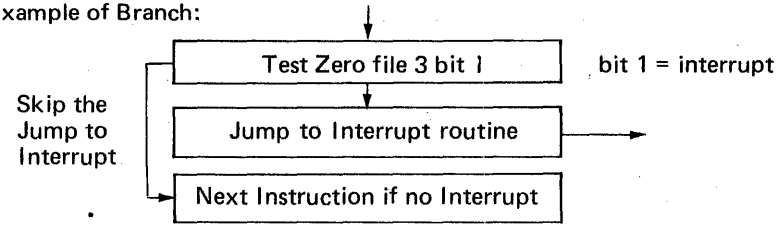
Bit Pattern Examples:

File Register Test Zero Literal	10001000 00111000	No Skip
File Register Test Zero Literal	11100111 00011000	Skip
File Register Test Zero Literal	10110000 01001010	Skip
File Register Test Zero Literal	00010000 00010000	No Skip

Since all bits tested must be 0, this command is good for testing for the occurrence of any of a number of possibilities, such as testing for the presence of any of 3 interrupt flags.

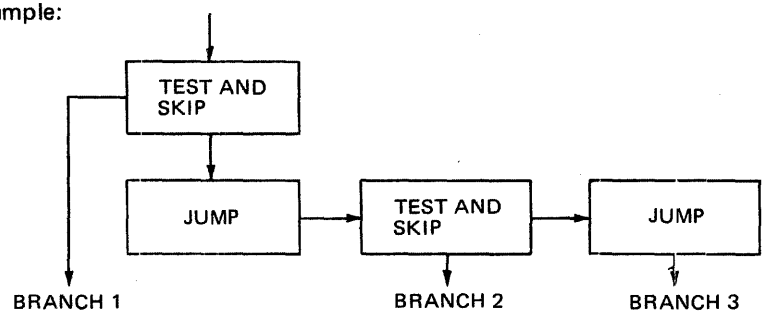
The conditional skip can be used for branching, or for simply skipping one instruction for certain conditions. For branching, the skip is followed by a jump command.

Example of Branch:



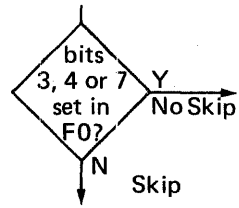
A three-way branch can be implemented with two test and skip commands and two jump commands.

Example:



Example: Skip if bits 3, 4, and 7 are not set in file 0.

L	Machine Code	Mnemonic	Flow Chart Notation
'00E'	'4098'	TZ F0,X'98'	



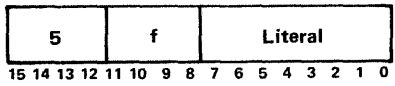
Affected Register States:

	Register	Before	After	
Case 1	L	'00E'	'010'	Skip
	F0	'43'	'43'	
Case 2	L	'00E'	'00F'	No Skip
	F0	'80'	'80'	

Command Execution Time – 220 nanoseconds – No Skip.
– 440 nanoseconds with Skip.

This timing applies to test not zero, and compare, as well.

Command	Mnemonic	OP Code
Test If Not Zero	TN	5f



If, for any bit of the literal field which is a 1-bit, the corresponding bit of the file register designated by f is also a 1-bit, the next command is skipped. The condition flags, LINK register and file register are not affected. If the skip is taken the timing of the command is two clock cycles.

This command differs from the test zero command in two ways. First it skips on 1's instead of 0's, and it skips on any 1 as opposed to all 0's on the test zero instruction.

If both tests (zero and not zero) were reduced to one bit comparisons, the only variation would be that one command produces the opposite result of the other. The choice would then be if a jump was wanted if the tested bit was 1, or 0.

If multiple bits are tested, the test not zero is the MAX TERM, and test zero is the MIN TERM logic equivalent.

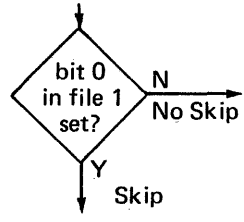
Bit Pattern Examples for test not zero:

File Register	01101100	Skip
Test Not Zero Literal	00110001	
File Register	01000001	No Skip
Test Not Zero Literal	00011010	
File Register	01100110	Skip
Test Not Zero Literal	01101000	
File Register	11100111	No Skip
Test Not Zero Literal	00010000	

Example: Skip if bit 0 in file 1 = 1

L	Machine Code	Mnemonic
'01C'	'5101'	TN 1,X'01'

Flow Chart Notation

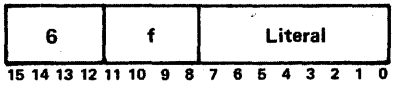


Affected Register States:

	Register	Before	After	
Case 1	L	'01C'	'01E'	Skip
	F_1	'01'	'01'	
Case 2	L	'01C'	'01D'	No Skip
	F_1	'80'	'80'	

Command Execution Time – 220 nanoseconds – No Skip.
440 nanoseconds – Skip.

Command	Mnemonic	OP Code
Compare	CP	6f



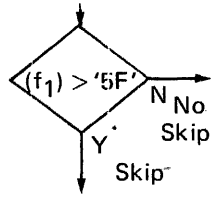
If the sum of the contents of the file register designated by f and the contents of the eight-bit literal is greater than 2^8-1, the next command is skipped. The condition flags, and file register are not affected. If the skip is taken the timing of the command is two clock cycles. The LINK stores the carry out of the adder. File 0 may not be selected by this command.

This command is used for looping control, and for data value testing. It is also used to test OP codes in instructions for selection of a particular class of OP codes, such as memory reference, having OP code (MICRO 810) greater than 5, for example. To test if the content of a file register exceeds a selected number, the 1's complement is placed in the literal part of the compare command.

Example: Skip if (f_1) > '5F'

L	Machine Code	Mnemonic
'014'	'61A0'	CP 1,X'A0'

Flow Chart Notation

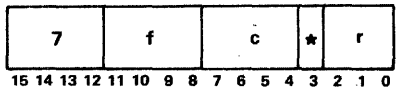


Affected Register States:

	Register	Before	After	
Case 1	L	'014'	'016'	No Skip
	F_1	'52'	'52'	
Case 2	L	'014'	'015'	Skip
	F_1	'66'	'66'	

Command Execution Time — 220 nanoseconds — No Skip.
440 nanoseconds — Skip.

Command	Mnemonic	OP Code
Control	K	7f



This command is used to control special data flow operations, and input/output functions. The prime functions are as follows:

- Enter sense switches from panel to selected file register.
- Shift selected file right 4 bit places.
- Enter internal status to selected file register.
- Set and clear the 3 input/output control flip flops (IOXX).

A secondary function for some of the prime functions is that data can simultaneously be moved from a file, or the input bus ANDed with the selected file, to a register. File 0 may be selected by the shift right 4 function only. These functions will be explained in detail in the following paragraphs. This command unconditionally updates the arithmetic condition flags in file 0.

The prime functions of this command are determined by the value of the c field as follows:

c	Operation	Explanation
0	— No Operation	
1	— Enter Sense Switches:	The status of the four console sense switches are placed in the four high order bits of the file register designated by f. The four low order bits are set to 1-bits. The status can also be placed in the designated destination register.
2	— Shift File Right 4:	The four high order bits of the file register designated by f are placed in four low order bits of the file register. The four high bits are set to 1-bits. The result can also be transferred to the designated destination register.
3	— Unused	
4	— Enter Internal Status:	The eight internal status bits are placed in the file register designated by f, and the designated destination register. The internal interrupt flag in file 0 is reset by this command, along with the console interrupt, real time clock, memory parity, and power fail/restart. Console step is reset upon release of the console switch and spare bits are controlled according to their individual implementation in hardware.
5	— Unused	
6	— Unused	

7	— Enter Console Switches:	The contents of the eight low order console command switches are ANDed with eight low order bits of the next command. File register 0 and destination register 0 must be selected to prevent any modification of the file or register during the execution of the Control command. The command physically preceding this operation must not cause a read-only memory delay.
8	— Clear I/O Mode:	The I/O Control register is cleared. Data from the designated file or the input bus ANDed with the designated file can be transferred to the designated file register and register (r).
9-F	— Set I/O Mode:	The I/O Control register is loaded with the three low order bits of c placing it in one of seven I/O bus or serial teletype modes. These modes are described in Section 4. Data from the designated file or the input bus ANDed with the designated file can be transferred to a designated file register and register (r).

Affected: F, I/O Control, Condition Flags, r

For all values of c, except 0, 3, 5, 6, or 7, source data is placed in the designated file, if bit 3 = 0 and in the designated destination register. Destination r = 7 is undefined for this command. In other words, the U register is not used.

Examples:

C = 1 Enter sense switches into file 1

L	Machine Code	Mnemonic	Flow Chart Notation
'005'	'7110'	K 1,1	(SSW)→f_1

Affected Register Status:

	Register	Before	After
Case 1	L	'005'	'006'
	file 1	———	'9F'
	Sense SW (Binary)	1001	1001
	File 0 (Bits 2-0)	———	010
Case 2	L	'005'	'006'
	file 1	———	'2F'
	Sense SW (Binary)	0010	0010
	File 0 (Bits 2-0)	———	000

C = 2 Shift file 1 right 4

L	Machine Code	Mnemonic	Flow Chart Notation
'012'	'7120'	K 1,2	F_1 SR4 → F_1

Affected Register States:

Register	Before	After
L	'012'	'013'
file 1	'E0'	'FE'
file 0 (Bits 2-0)	---	010

C = 4 Enter internal status to file 1

L	Machine Code	Mnemonic	Flow Chart Notation
'1E3'	'7140'	K 1,4	Status→f_1

Affected Register Status:

Register	Before	After
L	'1E3'	'1E4'
file 1	---	'45'
Status	'45'	'40'
file 0 (Bits 2-0)	---	000

Note: Sense switch 4 can be tested by testing negative condition flag after entering SSW to file 0.

C = 7 Enter console switches

This requires two commands, the first being the enter console switches, followed by a load file, if the switch settings are to go into a file; a load register if switch settings are to go into a register, or an operate command if switches are to modify the command. A load file operation will be used for the example. The load file literal must be FF to duplicate the switch settings into the file.

Example: Enter console switches into f5.

L	Machine Code	Mnemonic	Flow Chart Notation
'112'	'7070'	K 0,7	f5 ∧ CSW→f5
'113'	'25FF'	LF 5, X'FF'	

Affected Register Status:

Register	Before	After
L	'112'	'114'
file 5	---	'A5'
Console SW	'A5'	'A5'
file 0 (Bit 2-0)	---	010

This command cannot be executed via the front panel because it requires a dynamic situation, and two separate functions entered on the front panel.

C = 8-F Input/Output control

When c equals 8-F, the operations are associated with external input/output, and the three low order bits of c are placed in the I/O Control register. On the same operation, data can be moved from the designated file register or the input bus ANDed with the designated file register as determined by the current contents of the I/O Control register, to the designated file or destination register. The data source is specified as follows:

I/O Control Register Mode	Source
0-3	Designated file register.
4-7	Input bus A designated file register.

The values 4-7 correspond to the 103X control flip flop. This flip flop must be set in order to transfer data from the input bit to the computer internal registers. Other than this restriction, the three I/O control register bits can be used in any manner desired at the microprogramming level of the MICRO 800 and as long as standard I/O interface modules are not used.

For purposes of standardization of common interface modules, and implementation of standard I/O software instructions, a convention for I/O codes has been adopted as shown inTable 5.

Table 5. MICRO 810/820 Standard I/O Control Codes

c Field (Hex)	I/O Mode	IOXX 3 2 1	Control Activity	
8	0	0 0 0	None	
9	1	0 0 1	Control Output (COXX/)	Output Codes
A	2	0 1 0	Data Output (DOXX/)	Output Codes
B	3	0 1 1	Space Serial Teletype	Output Codes
C	4	1 0 0	Concurrent Acknowledge (CACK/)	
D	5	1 0 1	I/O Acknowledge (IACK/)	Input Codes
E	6	1 1 0	Data Input (DIXX/)	Input Codes
F	7	1 1 1	Spare	Input Codes

Note that the I/O mode is directly represented as the 3 least significant bits of the c field.

Standard Output Functions:

The two output codes COXX, DOXX represent a two-byte output sequence, where the first byte is for control, and the second byte is for data. A device select control byte is first put in the T register (which is also the output bus) and then COXX is set and reset. Then a data value is placed in T and DOXX is set and reset.

Standard Input Functions:

COXX and DIXX control codes are used for data input routines. A device select control byte is first placed in T, and COXX is set and reset. Then DIXX is set, data is input while DIXX is set and then DIXX is reset.

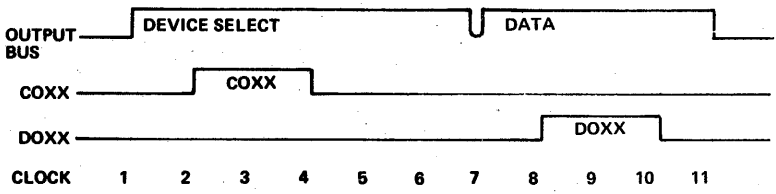
While DIXX is set, data can be entered two different ways:

1) Operate commands involving T get the input bus instead of T as long as I03X is set. These commands are ADD, OR, COPY, EXCLUSIVE OR, AND. Any of these can be used to input data while DIXX is set as long as T complement is not selected.

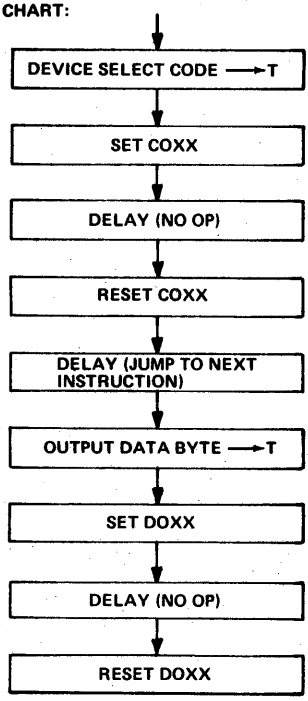
2) The control command with the c field = 8-F causes the input bus to be ANDed with the selected file register as long as I03X is set. This method allows inputting on the same command that resets DIXX (providing the selected file has first been set to 'FF').

I/O Examples:

1) Generate following output wave form:



FLOW CHART:

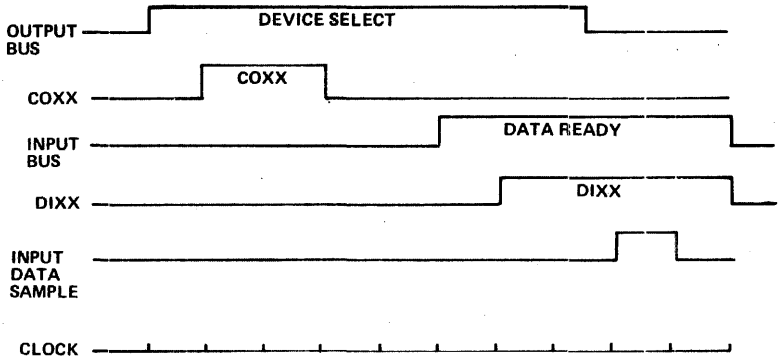


I/O CONTROL MACHINE CODES

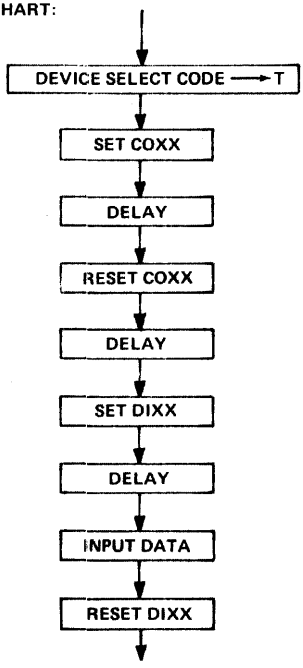
'7090'
'1000'
'7080'
JUMP CAUSES 2 CLOCK DELAY

'70A0'
'1000'
'7080'

2) Input data according to following wave form:



FLOW CHART:



I/O CONTROL MACHINE CODES

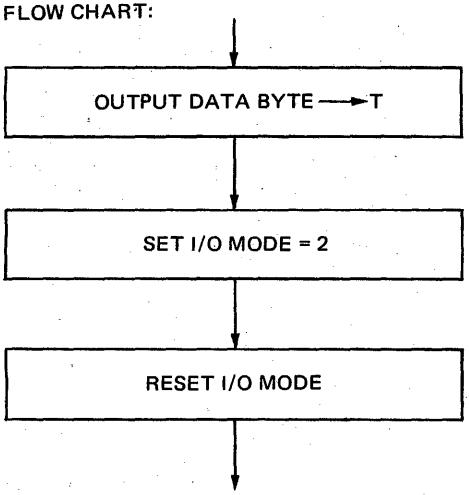
SET COXX	'7090'
DELAY	'1000'
RESET COXX	'7080'
DELAY	Jump to next inst. 2 clock delay
SET DIXX	'70E0'
DELAY	Jump to next inst. 2 clock delay
INPUT DATA	Operate class command
RESET DIXX	'7080'

For a very simple interface having only 3 data registers to set, a single byte sequence will suffice for outputting data.

3) Output a byte to interface Latch No. 2, where only 3 interface latches exist in the system, using the simple interface technique mentioned above.

FLOW CHART:

I/O CONTROL MACHINE CODES

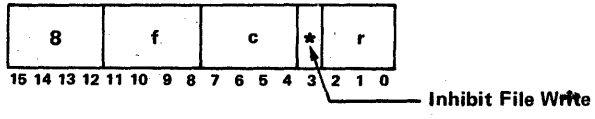


On an input cycle it is necessary to wait at least one clock cycle after generating DIXX to input data. The I/O controls are set in time at the completion of the control command. An input on the next clock would attempt to transfer data before the interface unit has the correct response data ready for input.

c field = B which is I/O mode 3 is used to set the serial teletype mode to SPACE, which ties up the I/O channel.

c field = D which is I/O mode 5 is used to acknowledge interrupts.

Command	Mnemonic	OP Code
Add	A	8f



The selected operand is added to the contents of the file register designated by f. The sum is placed in the file register (f), if * is a 0-bit, and in the register designated by r. The state of the carry out of the high order bit of the adder is placed in LINK. File 0 may not be selected by this command. The c field controls selection of the operand, incrementing the result and modification of the condition flags as follows:

c-bits
7 6 5 4

1 x x x — Link Control: The content of LINK is added to the sum. The zero condition flag can be reset but cannot be set, providing a linked zero test over multiple bytes. A linked zero over multiple bytes functions as follows: Assume a 2-byte add is to be performed. Two file registers contain a 16-bit number to be added to another 16-bit number in core memory. The add is performed one byte at a time, with the LINK used for carry into the second add. On the first byte addition the condition flags are modified. If the result of the first byte addition is not zero, then of course the entire addition results in a non-zero condition, so that the zero condition flag should not be set on the second byte add even if its result is zero. On the other hand, if the first add produces a zero condition, the second may not, therefore the zero condition flag should be resettable on the second byte add.

The add function can be used to move data from a file to another register by not selecting any input in the c field.

x 1 x x — Add One: One is added to the sum.

x x 1 x — Select T: The contents of the T register or the input bus are selected as the operand. If the T register is not selected, the operand is zero.

x x x 1 — Modifying Condition Flags: The condition flags are updated according to the result.

Eight different examples have been selected to illustrate various c states, data values, and destination registers. Since the L register advances 1 unless it is the destination, its state will not be shown in the affected register state chart. File 1 will be used in all examples.

The various functions selected for each example are shown in Tables 6, 7, 8 and 9.

Table 6.

The general form of the examples is —

Add the contents of file 1 to one or more of the following:

Link, 1, T

Destination register choices are

T, F_1, or N

Link is always updated.

Condition flags are updated on selected examples.

Table 7.

Add command uses file 1 for all examples.
Table of functions selected for each example.

Example	c Field					Destination		
	Add Link	Add 1	Select T	Modify Cond. Flags	Hexa-decimal Code for c Field	Selected Register Symbol	Binary Code	Hexa-decimal Code
1. Add (file 1) to (T), put result in T and f_1, and update condition flags.	0	0	1	1	3	T, f_1	0001	1
2. Add (file 1) to (T), put result in T, update condition flags.	0	0	1	1	3	T	1001	9
3. Add (file 1) to T, put result in N, update condition flags.	0	0	1	1	3	N	1011	B
4. Add (file 1) to T, +1, put result in f_1 and N.	0	1	1	0	6	N, f_1	0011	3
5. Add (file 1) to (LINK), put result in f_1.	1	0	0	0	8	f_1	0000	0
6. Add one to f_1 and put result in f_1, update C.	0	1	0	1	5	f_1	0000	0
7. Add (f_1) to T and (LINK). Put result in f_1.	1	0	1	0	A	T, f_1	0000	0
8. Add (file 1) to (T) plus 1. Put result in T, f_1.	0	1	1	0	6	T, f_1	0001	1

The coding for the 8 Addition examples is shown below.

Table 8.

Example	Machine Code (Hex)	Assembly Language Mnemonics	Flow Chart Notation
1	8131	AT 1, T, C	(f_1) + (T) ⟶ T, f_1, C
2	8139	AT* 1, T, C	(f_1) + (T) ⟶ T, C
3	813B	AN* 1, T, C	(f_1) + (T) ⟶ N, C
4	8163	AN 1, I, T	(f_1) + (T) +1 ⟶ N, f_1
5	8180	A 1, L	(f_1) + (L) ⟶ f_1
6	8150	A 1, I, C	(f_1) + 1 ⟶ f_1
7	81A0	A 1, L, T	(f_1) + T + (L) ⟶ f_1
8	8161	AT 1, I, T	(f_1) + (T) + 1 ⟶ T, f_1

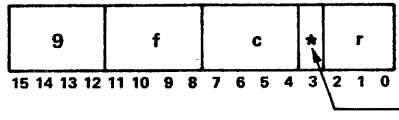
NOTE: If both Link and 1 are selected as inputs, they are ORed instead of added, thus the effective input is 1 regardless of the value of L.

Command Execution Time – 220 nanoseconds.

Table 9. Affected Register State Chart

Example		File	T	Link	N	Conditions Zero	Neg	Ovflow
1	Before	'65'	'9B'	-----	-----		-----	
	After	00	00	1	-----	1	0	0
2	Before	'65'	'15'	-----	-----		-----	
	After	'65'	'7A'	0	-----	0	0	0
3	Before	'65'	'65'	-----	-----		-----	
	After	'65'	'65'	0	'CA'	0	1	1
4	Before	'65'	'00'	-----	-----		-----	
	After	'66'	'00'	0	'66'		-----	
5	Before	'00'	-----	1	-----		-----	
	After	'01'	-----	0	-----		-----	
6	Before	'FF'	-----	-----	-----		-----	
	After	'00'	-----	1	-----	1	0	0
7	Before	'00'	'00'	1	-----		-----	
	After	'01'	'00'	0	-----		-----	
8	Before	'01'	'01'	-----	-----		-----	
	After	'03'	'03'	0	-----		-----	

Command	Mnemonic	OP Code
Subtract	S	9f



The complement of the selected operand plus one is added to the contents of the file register designated by f. The difference is placed in the file register (f) if * is a 0-bit, and in the register designated by r. The result is a 2's complement subtraction. The state of the carry out of the high order bit of the adder is placed in LINK. File 0 may not be selected by this command. The c field controls selection of the operand, incrementing the result, and modification of the condition flags as follows:

c-bits 7 6 5 4	Operation
1 x x x	Link control: The content of LINK is added to the sum. Selection of the LINK inhibits the automatic addition of one. The zero condition flag cannot be set, providing a linked zero test over multiple bytes. Refer to the add description for details on linked zero test.
x 1 x x	Inhibit add one: If link control is not selected, one is automatically added to the result to produce a 2's complement subtraction. This control bit inhibits this addition, providing a 1's complement subtraction.
x x 1 x	Select T: This complement of the contents of the T register are selected as the operand to the adder. If not selected, the operand consists of a 1-bit in each bit position.
x x x 1	Modify Condition Flags: The condition flags are updated according to the result.

Affected: F, LINK, Condition Flags, r

If the input bus is enabled (103X), this command will yield an unpredictable result because the complement of the input bus is not available.

Examples:

1. Subtract zero from file 1.

   $(f_1) - 0 \longrightarrow f_1$

Machine Code	Mnemonic
'9100'	S 1

Affected register states:

Register	Before	After
Link	---	1
file 1	'00'	'00'

Even though 0 is subtracted from 0, since 2's complement adding is used there is a carry of 1 all through the adder to the Link.

2. Subtract T, 1 from file 1
   Destination T Update condition flags

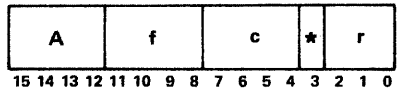
Machine Code	Mnemonic	Flow Chart Notation
'9179'	ST* 1,D,T,C	$(f_1) - T\text{-}1 \longrightarrow T,C$

Affected register states:

Register	Before	After
f_1	'31'	'31'
T	'31'	'FF' ← 2's complement for -1
L	———	0
C	———	0 1 0 (Zero, Neg, Overflow)

Command execution time – 220 nanoseconds.

Command	Mnemonic	OP Code
Read Memory	R	Af
Write Memory	W	Af



The primary function of this command is to initiate a core memory cycle in which one byte is transferred between the T register and core memory. The address in core is determined by the contents of the M and N registers. File 0 may not be selected by this command.

The lower two bits of the c field determine whether the memory operation is read or write and whether the operation is a full or half cycle.

The c-bits control the type of memory operation as follows:

c-bits 7 6 5 4	Memory Access Operation
x x 1 x	Half Cycle: If this bit is a 1-bit, a half cycle memory operation is performed; otherwise a full cycle operation is selected.
x x x 1	Write: If this bit is a 1-bit, a write memory operation is performed; otherwise a read operation is selected.

A full cycle takes 5 clock times.

A half cycle takes 3 clock times.

A full cycle read leaves the data in core unchanged.

A full cycle write causes the old data to be cleared so the new value is unaffected by the old.

A half cycle read leaves all ones in the core location.

A half cycle write ANDS the data to be written with the data already in core.

If a half cycle write into a particular memory cell was preceded by a half cycle read, the data value gets stored without modification since it is ANDed with all 1's, left from the previous half cycle read.

A secondary function of this command is to simultaneously move data between registers while initiating the memory cycle.

The contents of the file register designated by f is unaltered, incremented, or decremented as controlled by the c field. The result is placed in the file register (f) if * is a 0-bit, and in the register designated by r. At the same time, a read (R) or write (W) memory operation is initiated as controlled by bit 4. If the operation is a memory read, the T register is cleared and the accessed data is set into the T register after two clock cycle times. Data to be written into memory must be placed in the T register during or before the write memory command, if the operation is a half cycle write, and by the first clock cycle time after the write memory command on a full cycle write. The condition flags and LINK are not affected. Execution of the memory command is delayed if the memory is in a busy condition from a previous R or W command or DMA operation.

The bits of the c field control the transfer of data from the file register as follows:

c-bits 7 6 5 4	Operation
0 0 x x	Transfer: The contents of the file register are transferred unaltered.
0 1 x x	Decrement: The contents of the file register minus one are routed as specified. If the M register is selected as the destination and the content of LINK is a 1-bit, the contents of the file register are transferred without being decremented. This provides a decrement with link control when M is the destination.
1 0 x x	Add Link: The content of LINK is added to the contents of the file register, and the sum is transferred as specified.
1 1 x x	Increment: The contents of the file register plus one are transferred as specified.

This data transfer feature permits setting up one of the registers directly involved with the memory access (M, N, or T) at the same time the memory cycle is initiated. There are some timing restrictions pertaining to modification of M, N, or T registers during a memory cycle. Some of the functions have logic interlocks to prevent errors, and some do not. These restrictions must be carefully considered with respect to data errors, and unexpected program time delays. The restrictions are as follows:

1) Attempting to change M, or N while a memory cycle is in progress stops the computer clock until the memory cycle is over. No data errors result. Either M or N can be changed by the command initiating the memory cycle without causing delay.

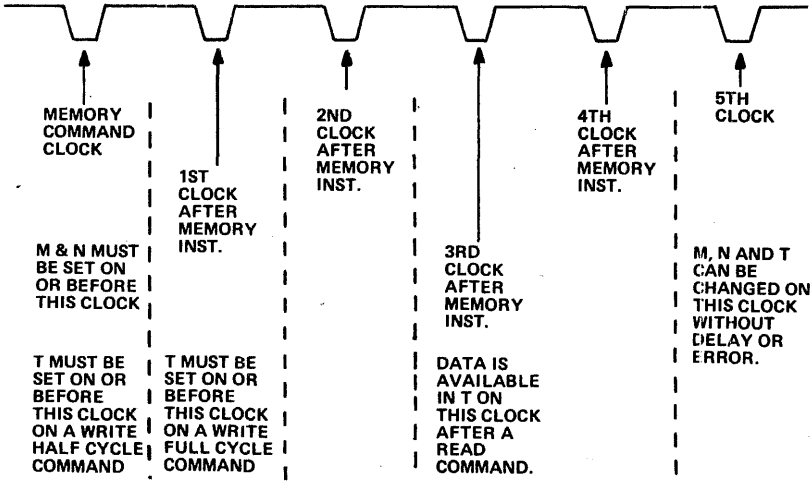
2) Accessing T during a read cycle causes the clock to stop until the new data value from core is correctly in T. This causes delay but no data error.

3) Changing T during a write cycle will not cause delay but it may cause a data error.

The memory access restrictions are specifically defined in the following chart:

	Full Cycle Read	Full Cycle Write	Half Cycle Read	Half Cycle Write
Delay from changing M and N	Up to 4 clocks	Up to 4 clocks	Up to 2 clocks	Up to 2 clocks
Delay due to T access	Up to 2 clocks	0	Up to 2 clocks	0 clocks
Data in T available (on Read)	2nd clock after memory command		2nd clock after memory command	
T must be loaded by (on Write)		1st clock after memory cycle command		Memory Cycle Command
T must stay loaded until (on Write)		4 clocks after memory command		2 clocks after memory command

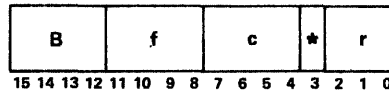
Timing Diagram for Memory Accesses:



Examples:

Example	Machine Code: op	Machine Code: file	Machine Code: c	Machine Code: dest	Mnemonics	c Field Binary Functions and Codes for Memory Commands	c Field Hex. Code	General Description
1) Full cycle write (file 1) + 1 ⟶ N, f_1	A	1	D	3	WN 1, I	Increment 1 1 — Full cycle write 0 1	D	Full cycle write memory is initiated and N register is updated as well as f_1.
2) Half cycle read (file 2) ⟶ M, f_2	A	2	2	2	RM 2, H	Transfer 0 0 — Half cycle read 1 0	2	Half cycle read memory is initiated while M register is updated directly from f_2.
3) Half cycle write (file 2) + (Link) ⟶ M, f_2	A	2	B	2	WM 2, L, H	Add Link 1 0 — Half cycle write 1 1	B	Half cycle write memory is initiated while file 2 and M are updated by adding (LINK).
4) Full cycle write (file 3) ⟶ T, f_3	A	3	1	1	WT 3	Transfer 0 0 — Full cycle write 0 1	1	Full cycle write memory is initiated, T is updated from f_3 on the same command.
5) Half cycle read (f_1) − 1 ⟶ N followed (f_3) + (T) ⟶ T, f_3	A 8	1 3	6 2	B 1	Inhibit file write RN* 1, D, H AT 3, T	Decrement 0 1 — Half cycle read 1 0 —	6 —	Half cycle read memory is initiated, followed by T register access on the next instruction. This will cause a program delay until the third clock.
6) Half cycle write followed by loading T (f3) ⟶ T, f_3	A 8	0 3	3 0	0 1	W 0, H AT 3	Transfer 0 0 — Half cycle write 1 1 —	3 —	Half cycle write memory is initiated, followed by loading T on next instruction. No time delay occurs, but data written into memory may be incorrect.
7) Full cycle read, decrement (file 1) and transfer to M (f_1) − 1 ⟶ M, f_1	A	1	4	2	RM 1, D	Decrement 0 1 — Full cycle read 0 0 —	4	A full cycle read is initiated. (f_1) is decremented and transferred to M. If (LINK) = 1 the contents of the file are transferred without being decremented.

Command	Mnemonic	OP Code
Copy	C	Bf



The selected operand is placed in the file register designated by f, if * is a 0-bit, and in the register designated by r. The LINK is not affected. The c field controls selection of the operand, incrementing the operand, and modification of condition flags as follows:

c-bits 7 6 5 4	Operation
1 x x x	Link Control: The content of LINK is added to the sum. The zero condition flag can be reset but cannot be set, providing a linked zero test over multiple bytes.
x 1 x x	Add One: One is added to the sum.
x x 1 x	Select T: The contents of the T register or Input bus are selected as the operand. If the T register is not selected, the operand is zero.
x x x 1	Modify condition flags: The condition flags are updated according to the result.

Affected: F, Condition Flags, r

This command is used to transfer T to a selected file register, with the option of incrementing or adding LINK while transferring. It is also used for inputting data, because when the input control flip flop (103X) is set during an input mode, operate commands selecting T get the input bus instead.

The command can be used to test the condition of T by selecting f0 as the file register (which is unaffected) and setting the modify condition flag in the c field.

The command can also be used to clear one file and another selected register by not selecting any input in the c field.

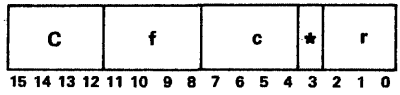
Command Execution Time – 220 nanoseconds.

File register 1 is used for all examples except setting condition flag example.

Examples of Copy Command:

Examples	Machine Code op	Machine Code file	Machine Code dest	c field for Copy Commands: Link	Add 1	Select T	Mod. Cond. Flags	Hex. Code	Destination for Copy Commands: Selected Registers	Binary Code	Hex. Code	Mnemonics	General Discussion
(T) ⟶ f_1	B	1	2 0	0	0	1	0	2	f_1	0000	0	C 1,T	(T) is transferred, unaltered to file 1.
(T) + 1 ⟶ f_1, N	B	1	6 3	0	1	1	0	6	f_1, N	0011	3	CN 1,I,T	(T) is incremented and transferred to file 1, and to the N register.
(T) + (LINK) ⟶ f_1	B	1	A 0	1	0	1	0	A	f_1	0000	0	C 1,T,L	(T) is added to (LINK) and transferred to f_1.
0 ⟶ f_1, N	B	1	0 3	0	0	0	0	0	f_1, N	0011	3	CN 1	File 1 and N registers are cleared because no input is selected.
(T) ⟶ f_0, C Set Condition Flags	B	1	3 0	0	0	1	1	3	f_0	0000	0	C 0,T,C	Condition flags are set according to the state of (T). File 0 can't be loaded by this instruction so is unchanged.
Set DIXX	7	0	E 0									K 0,X'E'	The input flip flop is set by the DIXX command, so the copy T command transfers the Input bus to file 1 and to T.
Delay	1	0	0 0									LZ X'00'	
(T) ⟶ f_1, T	B	1	2 1	0	0	1	0	2	f_1,T	0001	1	CT 1,T	
Reset DIXX	7	0	8 0									K 0,8	

Command	Mnemonic	OP Code	Symbol
OR	0	Cf	AVB



The selected operand is logically inclusive-ORed on a bit-for-bit basis with the contents of the file register designated by f and the result is placed in the file register, if * is a 0-bit, and in the register designated by r. The LINK is not affected. The c field controls selection of the operand and modification of the condition flags as shown below:

c-bits 7 6 5 4	Operation
1 x x x	Link control: The zero condition flag can be reset but cannot be set, providing a linked zero test over multiple bytes. See the description of the add command for a detailed description of linked zero test.
x 1 x x	Select complement T: The complement of the contents of the T register is selected as the operand. If the T register is also selected, the effective operand contains a 1-bit in each bit position.
x x 1 x	Select T: The contents of the T register or Input bus are selected as the operand. If neither the T register nor the complement of the T register is selected, the operand is zero.
x x x 1	Modify Condition Flags: The condition flags are updated according to the result.

Affected: F, Condition Flags, r

If both complement T and $\overline{T}$ are selected, the operand is all 1's. If the input bit is enabled (103X), complement T must not be selected.

This command is used for the general function of logical ORing as needed in a microprogram. It also has the following specific applications: Setting flag bits without disturbing other bits (with the OR function it doesn't matter if the flag is already set since there is no carry); moving data from a file to another register by not selecting any operand; setting all 1's in a file register and/or one other selected register by selecting both T and $\overline{T}$ complement as operands; combining two numbers into one byte, such as for assembling hexadecimal digits into multiple digit numbers after the digits have been input to the computer as a string.

Bit pattern example of OR function:

	Binary	Hexadecimal
file 1	01101000	'68'
T	00110100	'34'
Result	01111100	'7C'

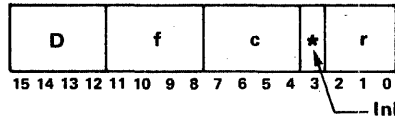
Command Execution Time — 220 nanoseconds.

File register 1 is used for all examples.

Examples of OR command:

Flow Chart Notation	Machine Code op	Machine Code file	Machine Code dest	c field for OR commands: Link	Select Comp. T	Select T	Mod. Cond. Flags	Hex. Code	Destination for OR command results: Selected Registers	Binary Code	Hex. Code	Mnemonics	General Discussion
$(f_1) \vee (T) \longrightarrow T$	C	1	2 9	0	0	1	0	2	T	1001	9	OT* 1, T	OR (file 1) with (T), inhibit file write put result in T.
$(f_1) \vee 0 \longrightarrow N, f_1$	C	1	0 3	0	0	0	0	0	N, f_1	0011	3	ON 1	Move (file 1) to N by ORing with 0 and putting result in N.
$(f_1) \vee (T) \longrightarrow f_1$	C	1	2 0	0	0	1	0	2	f_1	0000	0	O 1, T	OR (file 1) with (T) and put result in file 1.
$(f_1) \vee (T), (\overline{T}) \longrightarrow N$	C	1	6 B	0	1	1	0	6	N	1011	B	ON* 1,T,F	Set N = FF (all ones) by ORing (f_1) with T, $\overline{T}$ and putting result in N.
$(f_1) \vee (T), (\overline{T}) \longrightarrow f_1$	C	1	6 0	0	1	1	0	6	f_1	0000	0	O 1,T,F	Set f_1 = FF by ORing f_1 with T, $\overline{T}$ and putting result in f_1.
$(f_1) \vee (T) \longrightarrow$ Link, C	C	1	B 8	1	0	1	1	B	none	1000	8	O* 1,T,L,C	Perform conditional test on $(f_1) \vee (T)$ without changing f_1 or T. Select L to perform linked zero test with a previous command.

Command	Mnemonic	OP Code	Symbol
Exclusive OR	X	Df	A ∧ B



The selected operand is logically exclusive-ORed on a bit for bit basis with the contents of the file register designated by f and the result is placed in the file register, if * is a 0-bit, and in the register designated by r. The LINK is not affected. The c field controls selection of the operand and modification of the condition flags as shown below:

c-bits 7 6 5 4	Operation
1 x x x	Link Control: The zero condition flags can be reset but cannot be set, providing a linked zero test over multiple bytes. See the description of the Add command for a detailed description of linked zero test.
x 1 x x	Select Complement T: The complement of the contents of the T register is selected as the operand. If the T register is also selected, the effective operand contains a 1-bit in each bit position.
x x 1 x	Select T: The contents of the T register or input bus are selected as the operand. If neither the T register nor the complement of the T register is selected, the operand is zero.
x x x 1	Modify Condition Flags: The condition flags are updated according to the result.

Affected: F, Condition Flags, r

If both T and $\overline{T}$ are selected, this command produces the one's complement of the value in the file register. If the input bus is enabled (I03X), complement T must not be selected.

This command is used for the following functions: general purpose exclusive OR; data comparison; ones complementing; and flipping selected bits such as controls and status flags.

Bit pattern example of exclusive OR.

	Binary	Hexadecimal
file 1	01101100	'6C'
T	00011010	'1A'
Result	01110110	'76'

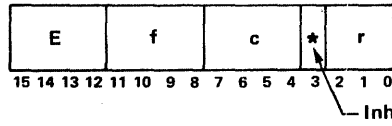
Command execution time – 220 nanoseconds.

File register 1 is used for all examples.

Examples of Exclusive OR command:

Example Flow Chart Notation	Machine Code op	file	dest	c field for OR commands: Link	Select Comp. T	Select T	Mod. Cond. Flags	Hex. Code	Destination for Exclusive OR command results: Selected Registers	Binary Code	Hex. Code	Mnemonics	General Discussion
$(f_1) \forall (T) \longrightarrow T$	D	1	2 9	0	0	1	0	2	T	1001	9	XT* 1, T	Exclusive OR (file 1) with (T) inhibit file write, put result in T.
$(f_1) \forall 0 \longrightarrow N, f_1$	D	1	0 3	0	0	0	0	0	N, f_1	0011	3	XN 1	Move (file 1) to N by exclusive ORing with 0 (same result as OR), put result in N.
$(f_1) \forall (T) \longrightarrow f_1$	D	1	2 0	0	0	1	0	2	f_1	0000	0	X 1, T	Exclusive OR (file 1) with (T) and put result in file 1.
$(f_1) \forall (T), (\overline{T}) \longrightarrow T$	D	1	6 B	0	1	1	0	6	N	1001	9	XT* 1,T,F	Produce ones complement of (f_1) and place result in T.
$f_1 \forall (T), (\overline{T}) \longrightarrow f_1$	D	1	6 0	0	1	1	0	6	f_1	0000	0	X 1,T,F	Produce ones complement of (f_1) and put it back into f_1.
$(f_1) \forall (T) \longrightarrow$ Link, C	D	1	B 8	1	0	1	1	B	none	1000	8	X* 1, T, L,C	Perform conditional test and linked zero test on $(f_1) \forall (T)$ without changing (f_1) or (T).

Command	Mnemonic	OP Code	Symbol
And	N	Ef	A ∧ B



The selected operand is logically ANDed on a bit-for-bit basis with the contents of the file register designated by f and the result is placed in the file register, if * is a 0-bit, and in the register designated by r. The LINK is not affected. The c field controls selection of the operand and modification of the condition flags as shown below:

c-bits 7 6 5 4	Operation
1 x x x	Link control: The zero condition flag can be reset but cannot be set, providing a linked zero test over multiple bytes. See the description of the add command for a detailed description of a linked zero test.
x 1 x x	Select complement T: The complement of the contents of the T register is selected as the operand. If the T register is also selected, the effective operand contains a 1-bit in each bit position.
x x 1 x	Select T: The contents of the T register or Input bus are selected as the operand. If neither the T register nor the complement of the T register is selected, the operand is zero.
x x x 1	Modify condition flags: The condition flags are modified by execution of the command. Updated according to the result.

Affected: F, Condition Flags, r

If both T and $\overline{T}$ are selected and And command moves the data, unchanged from the selected file register to the designated destination register. If the input bus is enabled (103X), complement T must not be selected.

The and command is used for the following functions: General purpose anding of files and T; resetting selected flag or status bits, without disturbing other flags; and masking out parts of a byte.

File register 1 is used for all examples.

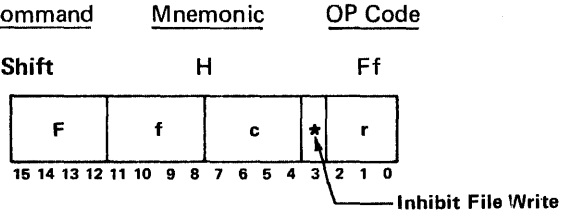
Examples of And Command:

Example Flow Chart Notation	Machine Code op	 file	 dest	c field for And commands Link	 Select Comp. T	 Select T	 Mod. Cond. Flags	 Hex. Code	Destination for And command results Selected Registers	 Binary Code	 Hex. Code	Mnemonics	General Discussion
$(f_1) \wedge (T) \longrightarrow f_1$	E	1	2 0	0	0	1	0	2	f_1	0000	0	N 1,T	(f_1) is anded with (T). The result is put into f_1.
$(f_1) \wedge 0 \longrightarrow N, f_1$	E	1	0 3	0	0	0	0	0	N, f_1	0011	3	NN 1	(f_1) is anded with 0. The result (which is 0) is put into N, and f_1.
$(f_1) \wedge (T) \longrightarrow T$	E	1	2 9	0	0	1	0	2	T	1001	9	NT* 1,T	(f_1) is anded with (T). The result is put in T and inhibited from f_1.
$(f_1) \wedge (T), (\overline{T}) \longrightarrow N$	E	1	6 B	0	1	1	0	6	N	1011	B	NN* 1,T,F	(f_1) is anded with (T), $(\overline{T})$ which is same as anding with FF (all ones). Result is put in N and inhibited from f_1.
$(f_1) \wedge (\overline{T}) \longrightarrow f_1$	E	1	4 0	0	1	0	0	4	f_1	0000	0	N 1,F	(f_1) is anded with $(\overline{T})$. The result is put into f_1.
$(f_1) \wedge (T) \longrightarrow$ Link, C	E	1	B 8	1	0	1	1	B	none	1000	8	N 1,T, L,C	(f_1) is anded with (T). The result is not put in any register. Only the condition flags are set. Use of link results in multi byte zero test.

Bit pattern examples of the and function.

	Binary	Hexadecimal
file 1	01101011	'6B'
T	10101101	'AD'
Result	00101001	'29'
file 1	01000010	'42
T	10111111	'BF'
Result	00000010	'02'
	Reset a flag	
file 1	10100101	'A5'
T	11010011	'D3'
(Select $\overline{T}$)	(00101100)	('2C')
Result	00100100	'24'
file 1	10100101	'A5'
T, $\overline{T}$	11111111	'FF'
Result	10100101	'A5'

Command Execution Time – 220 nanoseconds.



The contents of the file register designated by f is shifted left or right one bit position and placed in the file register, if * is a 0-bit, and in the register designated by r. The high order or low order bit which is shifted off is placed in LINK and in the overflow flag if the modify condition flag is selected. The c field controls the direction of shift, entry of an end bit, and modification of the condition flags as follows:

c-bit 7 6 5 4	Operation
1 x x x	Link control: The content of the LINK is inserted into the vacated low order or high order bit position. The zero condition flag can be reset but cannot be set, providing a linked zero test over multiple bytes. See the description of the add command for a detailed description of the linked zero test.
x 1 x x	Insert 1: A 1-bit is unconditionally inserted into the vacated low order or high order bit position; otherwise a 0-bit is inserted unless the contents of LINK is selected.

c-bit 7 6 5 4	Operation
x x 1 x	Shift right: if bit 5 is a 1-bit, the operation is a right shift; otherwise a left shift is performed.
x x x 1	Modify condition flags: The zero and negative flags are updated according to the result. The content of the bit shifted out is placed in the overflow flag.

Affected: F, LINK, Condition Flags, r

This command provides great flexibility for various shifting functions mechanized by microprogramming. These are as follows:

- Left or right shifting;
- End around carry or no end around carry;
- Arithmetic or logical shifts;
- Multiple byte shift register implementations in either file registers or core memory;
- Pattern rotations by successive shifting of 8 files one bit at a time and assembling into a 9th file;
- Set or reset link bit by shifting with no destination register.

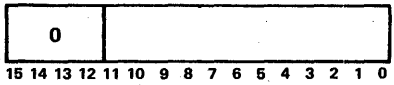
Bit pattern examples of shift command. All examples are for shift (f_1) and put result back in f_1.

Instruction	Sequence Number	file 1 Binary	Link	file 1 Hexa-decimal	Condition Flags
Shift Right	before	01101001	0	'69'	---
	after	00110100	1	'34'	---
Shift Left	before	01101001	1	'69'	---
	after	11010010	0	'D2'	---
Shift Right Enter Link	before	00111000	1	'38'	---
	after	10011100	0	'9C'	---
Shift Left Enter 1	before	10001010	0	'8A'	---
	after	00010101	1	'15'	---
Shift Left Modify Condition Flag	before	11001011	0	'CB'	---
	after	10010110	1	'96'	011
Shift Right Modify Condition Flag	before	00000001	0	'01'	---
	after	00000000	1	'00'	101

Instruction codes for bit pattern examples of shift command.
These examples are the same except for additional Destination Registers.

Example	Flow Chart Notation	Machine Code op	Machine Code field	Machine Code c	Machine Code dest	c field: Insert Link	c field: Insert 1	c field: Shift Right	c field: Mod. Cond. Flags	c field: Hex. Code	Destination for Shift Command results: Selected Registers	Destination for Shift Command results: Binary Code	Destination for Shift Command results: Hex. Code	Mnemonics	General Discussion
Shift right result to f_1, T.	$(f_1)@_R \rightarrow f_1,T$	F	1	2	1	0	0	1	0	2	f_1,T	0001	1	HT 1,R	(file 1) is shifted right one bit, link, or 1 are not inserted. The result is put in T and f_1.
Shift left result to f_1.	$(f_1)@_L \rightarrow F_1$	F	1	0	0	0	0	0	0	0	f_1	0000	0	H 1	(file 1) is shifted left one bit, link or 1 are not inserted. The result is put in f_1.
Shift right insert link result to f_1, N.	$(f_1)@_R$+LK$\rightarrow f_1,N$	F	1	A	3	1	0	1	0	A	f_1,N	0011	3	HN 1,R,L	(file 1) is shifted right one bit, (Link) is inserted in vacated left hand bit. Result is put in f_1 and N.
Shift left insert 1 result to f_1, M.	$(f_1)@_L$+1$\rightarrow f_1,M$	F	1	4	2	0	1	0	0	4	f_1, M	0010	2	HM 1, I	(file 1) is shifted left. 1 is inserted into the vacated right hand bit. Result is put in f_1 and M.
Shift left modify cond. flag. Result to f_1.	$(f_1)@_L \rightarrow f_1,C$	F	1	1	0	0	0	0	1	1	f_1	0000	0	H 1,C	(file 1) is shifted left. The result is put into file 1. Condition flags are modified.
Shift right Modify cond. flag. Result to f_1.	$(f_1)@_R \rightarrow F_1,C$	F	1	3	0	0	0	1	1	3	f_1	0000	0	H 1,R,C	(file 1) is shifted right. The result is put into file 1. Condition flags are modified.

Command	Mnemonic
Execute	E



The eight-bit contents of the U register are ORed with the eight high order bits of the execute command to form an effective command. This provides a means of partially modifying the contents of a read only storage location. The ORing is performed before the output of the read only storage is gated into the R register. The meaning of bits present in positions 0-11 is dependent upon the desired effective operation code after the modification. Due to the lookahead feature of the read-only memory, the new contents of the U register are not available until after one machine cycle following the transfer of data to it.

The execute command provides a means for program modification of a command. This capability is used for many different functions, three of which are as follows:

- Indexing of file registers in a program loop.
- Having a general purpose instruction which may take on different specific functions, such as load a register, add to the register, AND with the register, etc., depending on program variables.
- Selection of alternate file registers depending on program variables.

Sometimes a combination of two of the above is used.

The U register can be set with the load U command, or by being designated as the destination register of an operate class command, such as Add, Copy, etc.

For file register indexing, a separate file register is designated as an index register. It is loaded with an initial value, then incremented, with the result being put in U each time through the loop, until the loop is exited.

Examples of execute commands:

U register	'84'	
Execute Command	'0021'	This command is stored in ROM ET 0, 2
Effective Command	'8421'	$\begin{cases} (f_4) + (T) \longrightarrow f_4, T \\ AT \quad 4, T \end{cases}$

Incrementing the U register value leaves the command the same, but changes the file register number to 5. If this continued to file F, the next increment would change the command to a subtract.

U Register	'F1'	
Execute Command	'0020'	This command is stored in ROM E 0, 2
Effective Command	'F120'	Shift Right file 1 H 1, R

The meaning of the c field of the lower two hexadecimal digits in the execute command changes with the OP code value in the U register. Therefore the c field is left as a digit in the MNEMONIC for the execute command.

Commands can also be modified by the U register by using the operate commands with a 7 in the destination register. This method is advantageous if there are two variable functions to be done in one loop, with one U register setting. For example, a program may be indexing through a set of files where it is necessary to add to a file, and shift the same file in the same program loop. This could be mechanized as follows:

$(f_F) + 1 \longrightarrow U, f_F$

--- NOP

$(f_0) + (T) \longrightarrow f_0$, Destination = 7 (OR U with command)

$(F_0) @_R \longrightarrow F_0$, Destination = 7

The coding for this is:

	Machine Code	Mnemonic
	'8F46'	AU F, I
another command	----	----
	'8027'	AS 0, T Add to file 0
	'F027'	HS 0, R Shift file 0

Assume U = '04' after the first command.

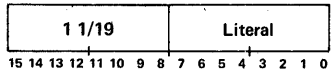
The effective commands following are:

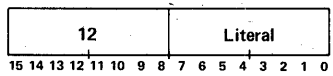
'8427'	Add to file 4
'F427'	Shift file 4 right

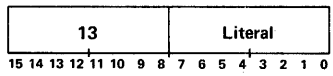
This method of command modification has the limitation of no destination register since the destination register code position is tied up selecting U as a modifier to the command. The execute command does not have this restriction.

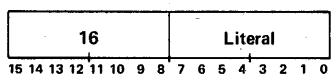
COMMAND REFERENCE TABLE

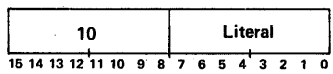
Command	Mnemonic	Operation Code	Comments
Load T	LT	1 1/19 | Literal	
Load M	LM	12 | Literal	
Load N	LN	13 | Literal	
Load U	LU	16 | Literal	
Load Zero	LZ	10 | Literal	
Load Seven	LS	17 | Literal	

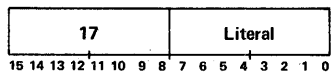




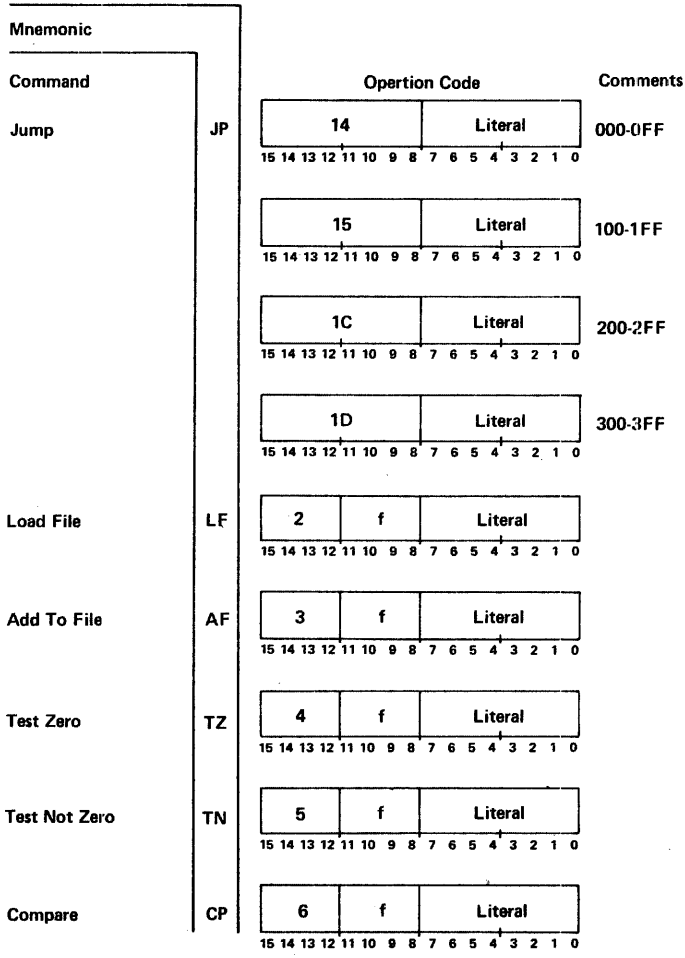


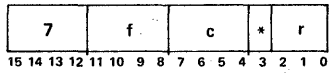
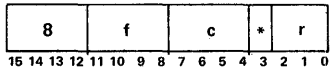
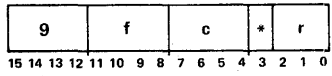


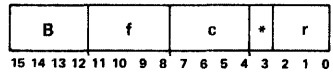
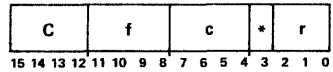


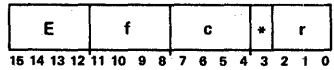
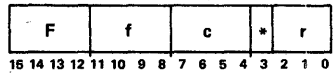


1	7	0	0	No Op	
1	7	0	1	Enable Serial TTY	
1	7	0	2	Reset T_8	
1	F	0	2	Set T_8	
1	7	0	4	Disable	External Interrupts
1	7	0	8	Enable	
1	7	1	0	Disable	Real Time Clock
1	7	2	0	Enable	
1	7	4	0	Load Protect Bit	
1	7	8	0	Halt	


Mnemonic
Command
Opertion Code
Comments
Jump
JP
14
Literal
000-0FF
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15
Literal
100-1FF
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1C
Literal
200-2FF
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1D
Literal
300-3FF
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Load File
LF
2
f
Literal
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Add To File
AF
3
f
Literal
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Test Zero
TZ
4
f
Literal
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Test Not Zero
TN
5
f
Literal
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Compare
CP
6
f
Literal
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Mnemonic		Operation Code	Comments	Operand Field
Command				
Control	K			
		0	No Op	
		1	Enter Sense SW	
		2	Shift Right 4	
		4	Enter Internal Status	
		7	Enter Console SW	
		8	Clear I/O	
		9	Set COXX (in MICRO 810/820)	
		A	Set DOXX (in MICRO 810/820)	
		B	Space Serial TTY	
		C	Set CACK (in MICRO 810/820)	
		D	Set IACK (in MICRO 810/820)	
		E	Set DIXX (in MICRO 810/820)	
		F	Spare	
Add	Ar*			
		1 x x x	Link	L
		x 1 x x	Add 1	I
		x x 1 x	Select T	T
		x x x 1	Modify Condition Flags	C
Subtract	Sr*			
		1 x x x	Link	L
		x 1 x x	Decrement	D
		x x 1 x	Select T	T
		x x x 1	Modify Condition Flags	C

Mnemonic Command		OperationCode	Comments	Operand Field
Memory	Wr* Rr*			
		1 x x x	Link	L
		x 1 x x	Decrement	D
		1 1 x x	Increment	I
		x x 1 x	Half Cycle Operation	H
		x x x 1	Write Operation (supplied by OP Code)	
Copy	Cr*			
		1 x x x	Link	L
		x 1 x x	Add 1	I
		x x 1 x	Select T	T
		x x x 1	Modify Condition Flags	C
OR	Or*			
		1 x x x	Link	L
		x 1 x x	$\overline{T}$	F
		x x 1 x	T	T
		x x x 1	Modify Condition Flags	C
Exclusive OR	Xr*			
		1 x x x	Link	L
		x 1 x x	$\overline{T}$	F
		x x 1 x	T	T
		x x x 1	Modify Condition Flags	C

Mnemonic Command		Operation Code	Comments	Operand Field
AND	Nr*			
		1 x x x	Link	L
		x 1 x x	$\overline{T}$	F
		x x 1 x	T	T
		x x x 1	Modify Condition Flags	C
Shift	Hr*			
		1 x x x	Link	L
		x 1 x x	Insert 1	I
		x x 1 x	Shift R	R
		x x x 1	Modify Condition Flags	C

CPU MICRO COMMAND REPERTOIRE

	Code	Mnemonic	Name	Operation
Literal Class Commands	0XXX	E	Execute	0X is ORed with U Register
	10XX	LZ	Load Zero	No Operation
	11XX	LT	Load T	XX replaced contents of T
	12XX	LM	Load M	XX replaces contents of M
	13XX	LN	Load N	XX replaces N & M is cleared
	14XX	JP	Jump	to page 0
	15XX	JP	Jump	to page 1
	1CXX	JP	Jump	to page 2
	1DXX	JP	Jump	to page 3
	16XX	LU	Load U	XX replaces contents of U
	17XX	LS	Load Seven	Internal Controls
	2fXX	LF	Load File (f)	f = File number
	3fXX	AF	Add to File	f = File number
	4fXX	TZ	Test if zero	Skip on no bits match
	5fXX	TN	Test if zero	Skip on Any bits match
	6fXX	CP	Compare	Skip on f + XX 2^8-1
	Code	Mnemonic	Name	c Field (Binary)
Operate Class Commands	7fC*r	K	Control	0000 No Operation 0001 Enter Sense Switches 0010 Shift Right Four Bits 0100 Enter Internal Status 0111 Enter Console Switches 1000 Clear I/O Mode 1001 Control Output 1010 Data Output 1011 Space Serial TTY 1100 Concurrent Acknowledge 1101 Interrupt Acknowledge 1110 Data Input 1111 Spare
	8fC*r	A	Add	0001 Modify Flags 0010 File + T 0100 Sum + 1 1000 Sum + Link Bit
	9fC*r	S	Subtract	0001 Modify Flags 0010 File + T complement 0100 Inhibit Increment 1000 Difference + Link
	AfC*r	R/S	Read/Write Memory	00XX Transfer 01XX Decrement 10XX Add Link 11XX Increment XX1X Half Cycle XXX1 Write (Not Read)
If * = 0, result of operation is placed in file (f).	BfC*r	C	Copy	XXX1 Modify Flags XX1X Select T X1XX Select + 1 1XXX Select Link
	CfC*r	O	OR	XXX1 Modify Flags XX1X Select T X1XX Select T complement 1XXX Linked Zero Test
	DfC*r	X	Exclusive OR	Same as OR
	EfC*r	N	AND	Same as OR
	FfC*r	H	Shift	XXX1 Modify Flags XX1X Shift Right X1XX Insert ONE 1XXX Insert Link

CHAPTER 3

INPUT/OUTPUT

GENERAL DESCRIPTION

The CPU provides an extremely fast, elementary input/output capability. The data paths and control functions are simple elements that are sequenced from the control memory with flexible disciplines. The fact that the I/O element is very fast, 220 ns/step, microprograms (firmware) in the control memory can implement facilities with a high degree of versatility in timing, data paths and I/O capabilities such as priority interrupts, fully buffered data channels, macroprogrammable transfers, and special purpose communication multiplexer channels. This basic I/O element called the "Byte I/O Bus" is described in the following paragraphs. In addition, the direct memory occurs (DMA) and serial data interface are described.

BYTE I/O BUS

The byte I/O facility allows for data transfers over a party-line I/O bus under microprogram control. This I/O facility consists of a byte input bus, a byte output bus, and a three-bit I/O control register.

The I/O control register is loaded by bits 6-4 of the control command. The contents of the I/O control register define an I/O bus mode. The I/O control register outputs may be decoded to form individual control signals defining the type of transfer being performed on the byte I/O bus and the state of the serial interface output. Of the eight possible states of the I/O control register, one represents no activity on the bus, three are output modes, and four are input modes. One of the output modes removes the MARKing current from the serial interface output a SPACE to be output.

The byte I/O control modes are given in Table 10.

Table 10. Byte I/O Control Modes

Control Command

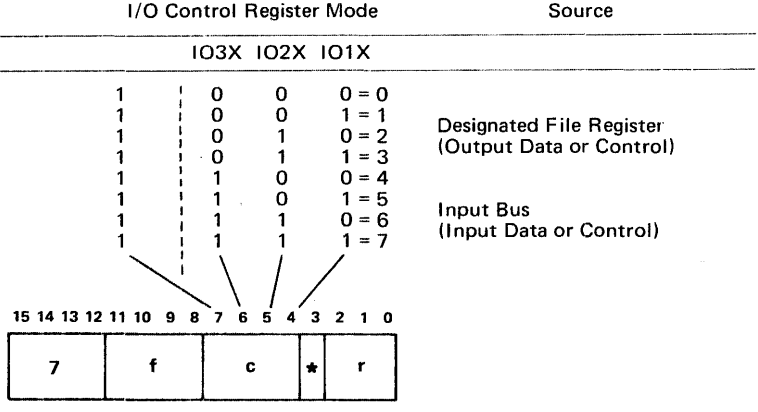
7	f	c	*	r
15 14 13 12	11 10 9 8	7 6 5 4	3	2 1 0

	Bits 7 6 5 4		Hex	Mode	Control Activity
	0 0 0 0		0		No Operation
	0 0 0 1		1		Enter Sense Switches
	0 0 1 0		2		Shift "f" Right Four Places
	0 1 0 0		4		Enter Internal Status
	0 1 1 1		7		Enter Console Switches (0-7)
OUTPUT FUNCTIONS	1 0 0 0	I/O CONTROL	8	0	Clear I/O Mode
	1 0 0 1		9	1	SPARE (*)
	1 0 1 0		A	2	SPARE (*)
	1 0 1 1		B	3	Space Serial Interface
INPUT FUNCTIONS	1 1 0 0		C	4	SPARE (*)
	1 1 0 1		D	5	SPARE (*)
	1 1 1 0		E	6	SPARE (*)
	1 1 1 1		F	7	SPARE

*These functions are used in the MICRO 810 and 820 I/O systems.

When the c field equals hexadecimal 8-F, the operations are associated with external input/output, and the three low order bits of c are placed in the I/O control register.

This three-bit register generates the control signals for the I/O bus by a decoding of the register outputs. It is loaded and cleared by a control command and therefore the timing of I/O control signals is under command control. There are three output modes and four input modes. The high order bit of the register is the input flag. When this bit is a 1-bit the input bus is substituted for the T register inputs, thus providing a source of data when executing an external I/O control command. On the same operation, data can be moved from the designated file register or the input bus, as determined by the current contents of the I/O control register, to the designated file or destination register. The data source is specified as follows:



Mode	Control Activity	Comments
0	Clear I/O Mode:	The I/O control register is cleared. Data from the designated file or Input bus can be transferred to the designated file register and register (R).
1-7	Set I/O Mode:	The I/O Control register is loaded with the three low order bits of c placing it in one of seven I/O bus or serial interface modes. These modes are described above. Data from the designated file or Input bus can be transferred to a designated file register and register (r).

NOTE: Once an I/O control register mode has been SET, an I/O clear mode must be executed to change the I/O control register mode of operation.

Internal Status — Interrupt

Eight internal status bits are provided to designate a particular internal interrupt condition. When any of the internal status bits are a 1-bit, the internal interrupt flag (bit 4) in the file register 0 is also a 1-bit. This flag bit is tested by the microprogram to detect the presence of the internal interrupt condition. The internal status bits are entered via the A bus into the selected file register by a control command. The eight internal status bits have the assignments given as follows:

Internal Status Bits

	Internal Status	
Bit	Without Processor Option Bd	With Processor Option bd
0	Console Interrupt	Console Interrupt
1	SPARE (DMA)*	SPARE (DMA)*
2	SPARE	Real Time Clock Interrupt
3	SPARE	SPARE
4	SPARE	Memory Parity Error Interrupt
5	SPARE	SPARE
6	Console Halt Switch	Console Halt Switch
7	SPARE	Power Fail/Restart Interrupt

*Not available as SPARE if DMA is installed.

All the internal status bits except the console interrupt and halt are associated with processor options and may be reassigned for special applications.

Bus Lines

The byte I/O bus consists of

- input data lines
- input control lines
- output data lines
- output control lines

The electrical implementation of the input and output bus lines is shown in Figure 14.

Input Lines

The data lines are an input to the B bus gating. The control lines are input to bits of file register 0. The input lines are ground TRUE signals which are properly terminated at the processor. If the bus is carried out of the basic enclosure it also must be terminated at the remote end. Each peripheral device gates information onto the bus by means of open collector type 944 DTL drive circuits. Up to 15 drivers may be connected to each line.

The logic level on the twisted pairs are:

One — 0 Volts
Zero — +3 Volts

*Typical Byte I/O Control Modes
(MICRO 810/820)

Mode	Control Activity	Term
0	None	None
1	Control Output	COXX/
2	Data Output	DOXX/
3	Space Serial Interface	SP1X/
4	Interrupt Acknowledge	IACK/
5	Concurrent Acknowledge	CACK/
6	Data Input	DIXX/
7	Spare	SP3X/

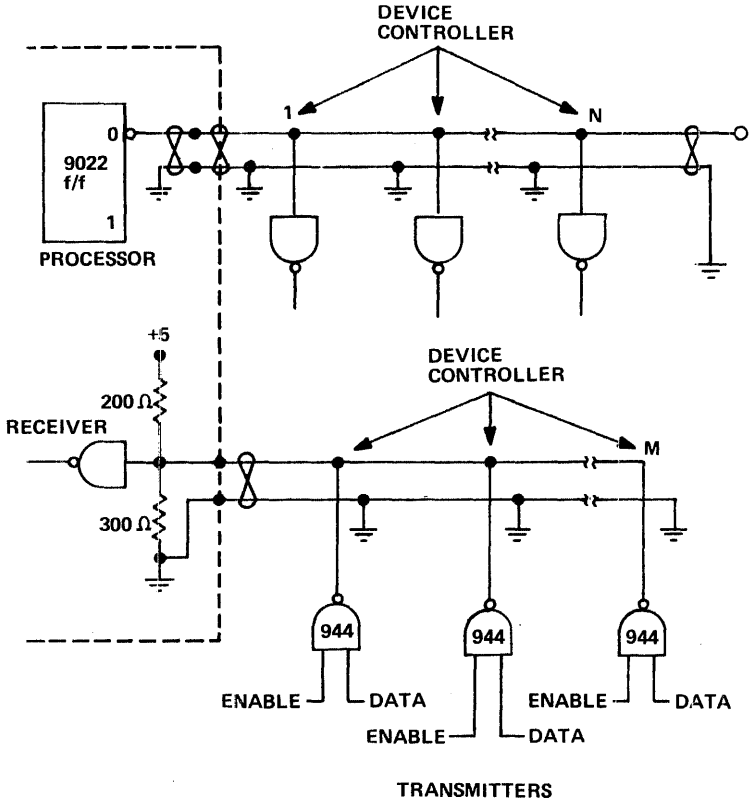


Figure 14. Bus Lines

Output Lines

The output data lines originate with the FALSE output of the T register. The output control lines originate with the I/O control register. If all peripheral devices on the bus are local to the enclosure, and the bus does not leave the enclosure, then the bus is standard logic levels and no DTL drivers and terminations are used. It may be necessary to repower the signals. If the bus leaves the enclosure, an I/O control board is required to provide type 944 DTL output drivers and decoding the control register. The cable length can be up to 30 feet in length and must be terminated at the remote end. Up to 15 receivers can be accommodated. The levels on the twisted pairs are:

One – 0 Volts
Zero – +3 Volts

Control Lines In	Typical Use in the System
External Interrupt (EINT/):	A peripheral device makes this line low to request an interrupt of the macroprogram. The microprogram must respond with an I/O acknowledge (mode 5)* signal. This line is bit 7 of the file register 0 where a 1-bit indicates an external interrupt request.
I/O Reply (ERPY/):	A peripheral device makes this line low in response to an I/O operation when closed-loop operation is required. This line is bit 5 of the file register 0.
I/O Request (ECIO/):	A peripheral device makes this line low in order to request a concurrent data transfer. The microprogram must respond with an I/O acknowledge (mode 5)* signal. This line is bit 3 of the file register 0.

File Register 0 Flags

Bit	Flag
0	– Overflow Result Condition
1	– Negative Result Condition
2	– Zero Result Condition
3	– Concurrent I/O Request Line** or (SPARE)
4	– Internal Interrupt
5	– I/O Reply Line** or (SPARE)
6	– Serial Interface
7	– External Interrupt Line** or (SPARE)

**If a standard CPU interface is not used, these Flags may be used as SPARE bits.

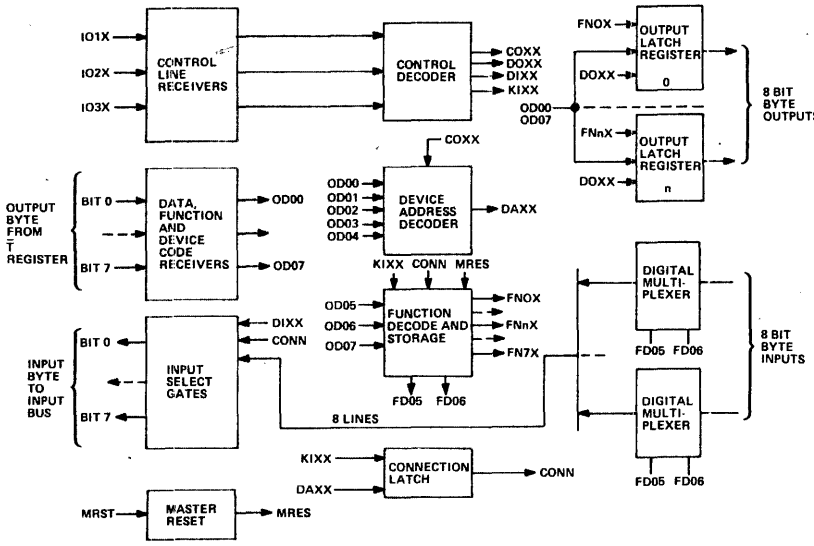


Figure 15. I/O Interface Block Diagram

Since the function code is only 3 bits instead of 4 it is effectively multiplied by 2 when put into the device and function code word.

Description of functional block diagram (Figure 15).

The control decoder receives the IOXX lines from the control line receivers and first decodes them into COXX, DOXX, and DIXX. These three are ORed to produce KIXX which is used to set and reset function and connect latches.

The device address decoder becomes active whenever the boards address appears on the 0D00-0D04 lines. DAXX is active only when COXX is active. Otherwise DAXX would become active every time the device address appeared on the output data lines.

The function latches set or reset every time there is a KIXX pulse. The output functions FN0X, etc., are not enabled unless CONN is active. The functions are used to enable the output latch.

The connection latch is set when the board detects its device address and COXX is active. It is reset on the trailing edge of the next DIXX or DOXX pulse.

The connection latch enables the functions and the input selection gate.

The input selection gates place the input data onto the input bus during DIXX whenever the CONN latch has been set indicating that this board has been addressed.

The output latches are updated to the values on the OD0X lines during DOXX whenever the corresponding function code FNNX is active.

Serial Interface

The processor contains a serial interface capable of communicating with a full duplex teletype. The input from the teletype appears as bit 6 of file register 0 where a 1-bit indicates that the teletype is transmitting a SPACE. The output to the teletype normally transmits a 20 milliampere MARKing current which can be keyed off to send a SPACE signal by placing the I/O control register in mode 3. Character assembly and disassembly, including all timing and synchronization, are performed by microprogramming.

The serial interface is standard. A teletype or CRT wired for 4-wire full duplex 20 milliampere operation may be directly connected to the cable provided with the machine. Other types of serial I/O devices also may use this condition.

Direct Memory Access

The direct memory access (DMA) interface allows for direct connection to the memory address, data and control busses. Within the machine enclosure there is a circuit board slot which is reserved for the DMA. This board may contain a channel to which a number of peripheral devices are connected, or a device controller which has direct memory access capability. Generally the DMA system will be customized for special applications.

The maximum data transfer rate is 909,000 bytes per second. The DMA I/O takes precedence over the processor for memory operations. The DMA must supply its own address control.

Typical Byte I/O Interface

To illustrate byte I/O programming, a typical interface has been selected which has minimum functions for transferring bytes in and out of the computer. A more complex device, such as a tape controller, or card reader, using the byte I/O function would contain logic similar to this for transferring control, status, and data between the controller and the MICRO 800.

The byte I/O interface described contains the following basic functions.

- Line receivers and drivers
- Device address decoder
- Function latch and decoder
- Connection latch
- Input multiplexer
- Input selection gates
- Output latches
- Control decoder

For the following examples assume that the device code is 00001. This results in the following device and function codes:

Function Code		Device and Function Code	
Binary	Hex	Binary	Hex
000	0	000 0 0001	01
001	1	001 0 0001	21
010	2	010 0 0001	41
011	3	011 0 0001	61

For summary purposes the logic terms used in the I/O interface example (which are standard for MICRO 800 interfaces) are defined in Table 11.

Table 11. Definition of Terms in I/O Interface Block Diagram

COXX	FUNCTION AND DEVICE CODE OUTPUT CONTROL PULSE
DOXX	DATA OUTPUT CONTROL PULSE
DIXX	DATA INPUT CONTROL PULSE
KIXX	INTERFACE CLOCK PULSE FORMED BY ORing COXX, DOXX, and DIXX
DAXX	DETECTED DEVICE ADDRESS ENABLED BY COXX
0D00-0D07	OUTPUT DATA LINES RECEIVES FROM MICRO 800 T COMPLEMENT REGISTER
FN0X-FN7X	LATCHED AND DECODED FUNCTIONS ENABLED BY CONN.
FD05-FD06	LATCHED BUT UNDECODED FUNCTION BITS
CONN	CONNECT LATCH INDICATING THAT THE I/O BOARD HAS RECEIVED ITS DEVICE CODE WITH COXX.
MRES	MASTER RESET FROM MICRO 800
I01X-I03X	3 BITS FROM CONTROL OUTPUT REGISTER
DIG MUX	DIGITAL MULTIPLEXER

See Figure 16 for I/O signal source.

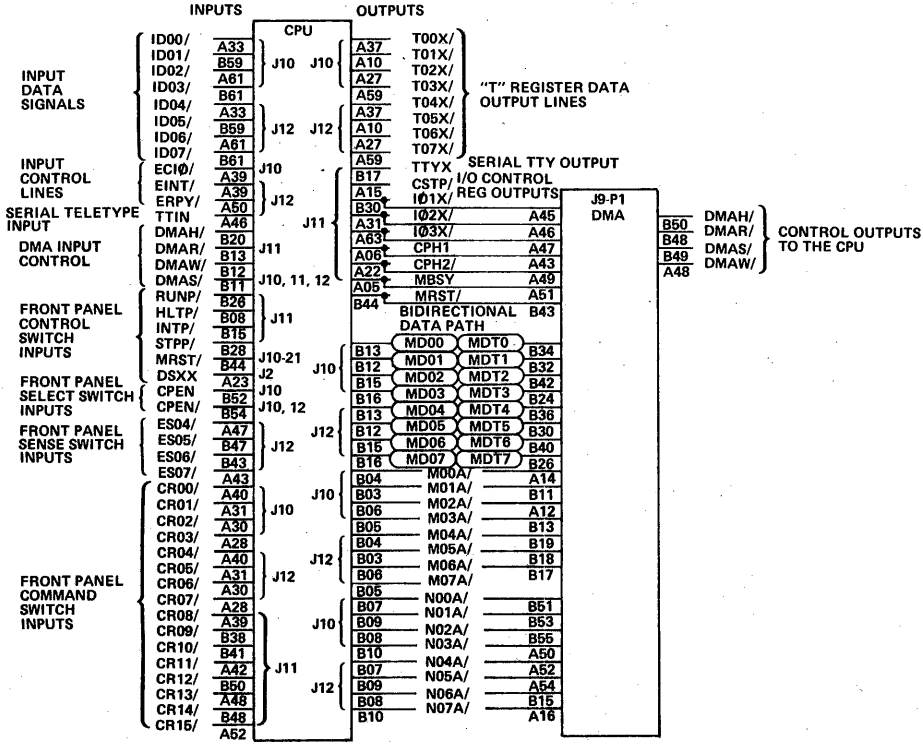
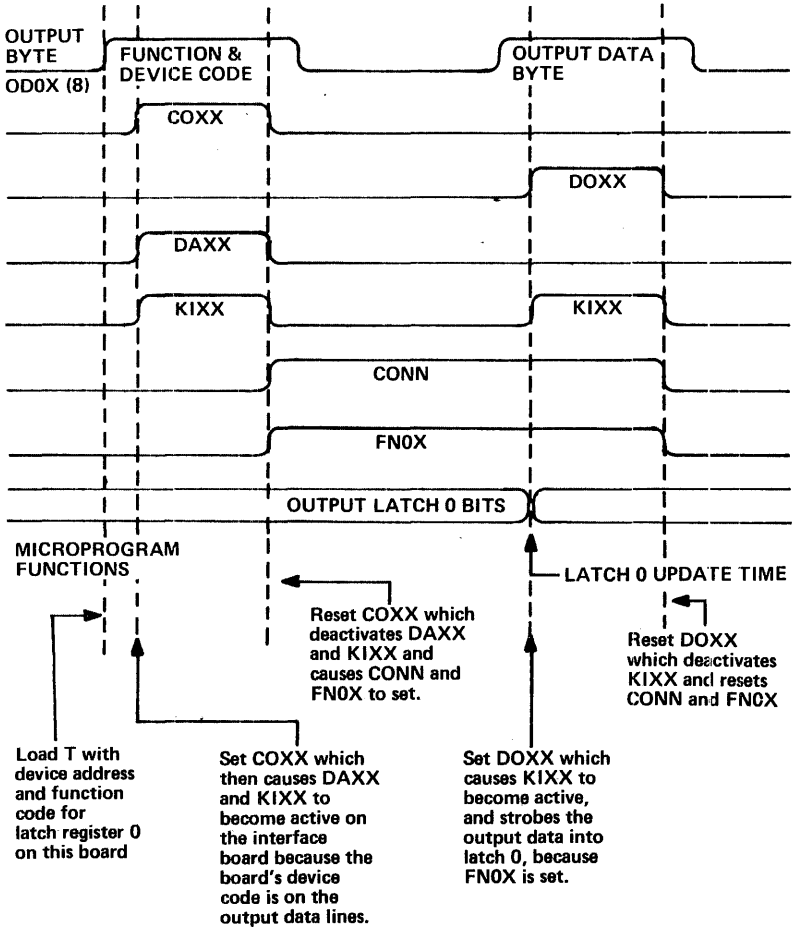


Figure 16. CPU Input/Output Signals

EXAMPLES OF I/O MICROPROGRAMMING

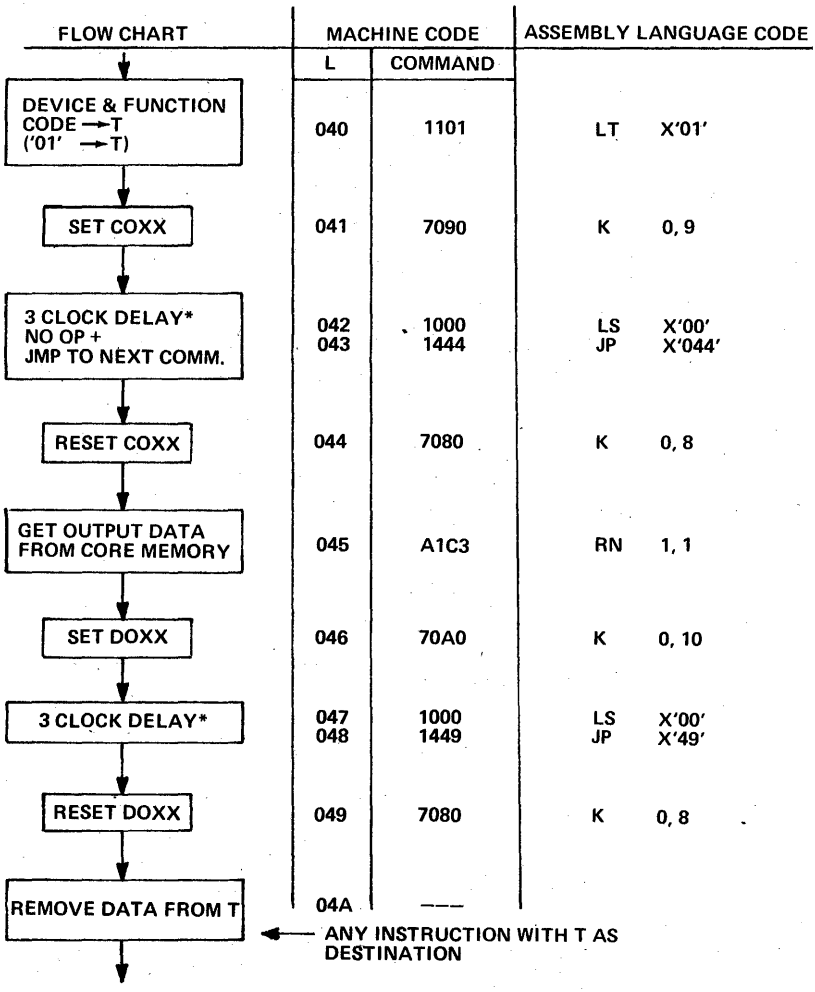
Example 1. For the first Input/Output example the timing of events and the microprogram routine are described for outputting a byte from the MICRO 800 to latch 0 in the interface board with device code 01.

Timing Diagram: Output a byte to latch register 0.



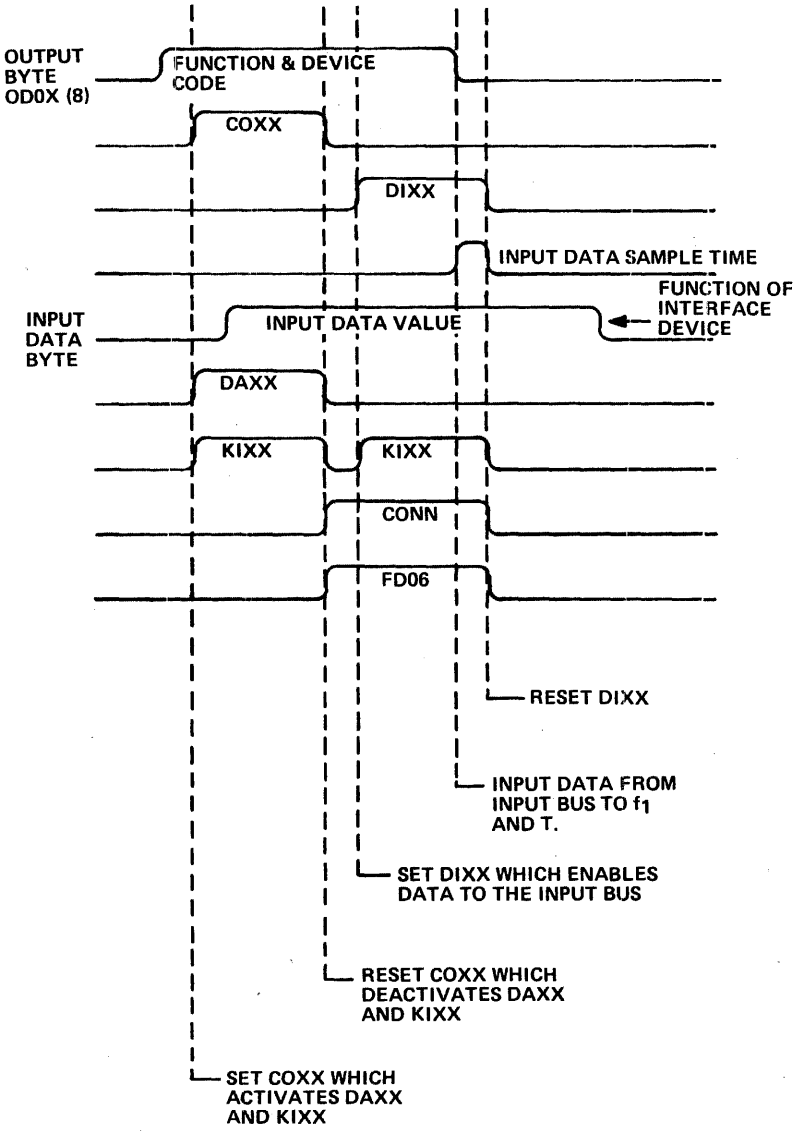
Microprogram: For outputting a data byte from the MICRO 800 to device 1, byte 0.

Example 1.

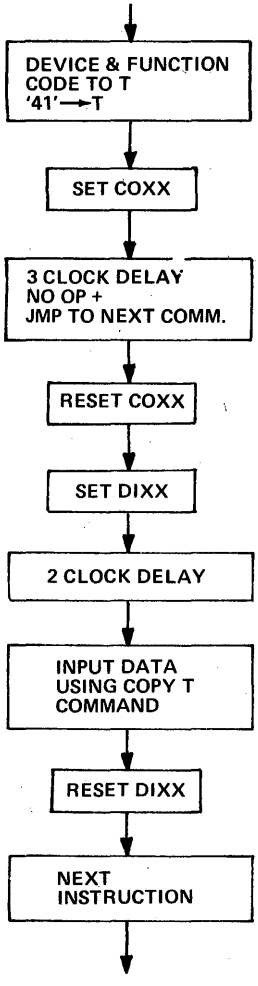


*This is the standard delay in the MICRO 810 to generate an 880 ns COXX and DOXX. It could be shorter if the interface is in the computer. Housekeeping can be done on delay clocks.

Example 2. For the second input/output example, the timing of events and the microprogram are described for inputting a byte from input byte 2 of device 01 to file register 1 and T.



Microprogram for example 2 inputting a data byte from device 01, byte 2 to f_1 and T.



FLOW CHART	MACHINE CODE		ASSEMBLY LANGUAGE CODE	
	L	COMMAND		
DEVICE & FUNCTION CODE TO T '41'→T	060	1141	LT	X'41'
SET COXX	061	7090	K	0, 9
3 CLOCK DELAY NO OP + JMP TO NEXT COMM.	062	1000	LZ	X'00'
	063	1464	JP	X'64'
RESET COXX	064	7080	K	0, 8
SET DIXX	065	70E0	K	0, 14
2 CLOCK DELAY	066	1467	JP	X'67'
INPUT DATA USING COPY T COMMAND	067	B121	CT	1, T
RESET DIXX	068	7080	K	0, 8
NEXT INSTRUCTION	069	ANY INSTRUCTION CAN BE NEXT		

Example 3. Special Input Function

To achieve minimum input time and still achieve one clock delay after setting DIXX use the following:

K	0, 14	Set DIXX
LF	K, X'FF'	Set file 1 = all ones and generate 1 clock delay
K	1, 11	Reset DIXX and simultaneously 'and' the input bus with (file 1)

Example 4. High speed multiple byte output to a special interface. Output bytes from files 1, 2, 3, and 4 to a 32 bit register on a special interface unit is an I/O connector. Use DOXX followed by a load zero command (CGOX). DOXX is used to distinguish from input command, followed by 4 file to T commands:

K	0, 10	DOXX Set
LZ	X '00'	CGOX
AT	1	Transfer files to T
AT	2	
AT	3	
AT	4	
K	0, 8	Reset DOXX

For this a very simple interface can be designed to transfer 32 bits of data from the MICRO 800 to an interface in only 1.54 microseconds.

CHAPTER 4

CENTRAL PROCESSOR OPTIONS

In addition to the option hardware, proper firmware must be provided to implement system action and response. This firmware may be designed specially for a given application. Standard firmware for each option described below is available.

Real-Time Clock

The real-time clock option provides an internal interrupt at a crystal-controlled timing rate. This may be used at the macroprogramming level for a real-time clock. The timing is derived from the processor internal clock which is divided down by some integer number less than 2^{13}, as determined by optional strapping on the option board.

When the timing signal occurs, it provides an internal interrupt by setting condition flag bit 4 and bit 2 of the internal status byte. The timing signal internal interrupt may be disabled and enabled by commands 1710 and 1720 respectively. The microprogram must detect the internal interrupt and take appropriate action. Special real-time clock interrupt handling firmware is available.

Power-Fail/Automatic Restart

The power-fail and automatic restart option provides the following:

1. An internal interrupt by setting condition flag bit 5 and bit 7 of the internal status byte upon detection of loss of primary power.
2. A machine reset when the computer is halted after loss of primary power.
3. A machine reset for 200 milliseconds after power is applied.
4. Automatic switch to run mode after the power-on reset period.
5. Power-restart interrupt immediately after automatic switch to run mode.

A power-fail interrupt detected while the machine is in the run mode can be used to cause the machine registers to be stored and to bring the processor to a halt. The automatic machine reset that follows the halt and the one following power-on prevents any spurious operations in the core memory. At power-on, the machine reset clears the L register causing the machine to start at read-only memory location 0. The power-fail interrupt which occurs at this time can be detected and treated as a restart interrupt to cause a restoring of the machine registers. Standard power-fail/automatic restart interrupt firmware is available.

CHAPTER 5

OPERATOR CONTROLS

CONSOLES

Two control console options are available: system console and basic console. These consoles differ in their number of displays and controls. This range of consoles permits the user to tailor the cost to meet the control and display capability required for a particular application. The system console is shown in Figure 17.

System Console

The system console provides complete control and display facilities. It is primarily used for maintenance, system and firmware checkout. This console provides for display of the MICRO 800 registers in addition to the functions of the basic console. The features include:

- Run and halt indicators
- Display of A-bus
- Display of M, N, and L registers
- Display of output of read-only memory
- Four sense switches
- Six control switches, including run, step, interrupt, clock reset, and save
- Manual command execution
- Power on-off

Basic Console

The basic console provides minimal control capability and is designed for dedicated system application where operator control is not required. The features include:

- Run and halt indicators

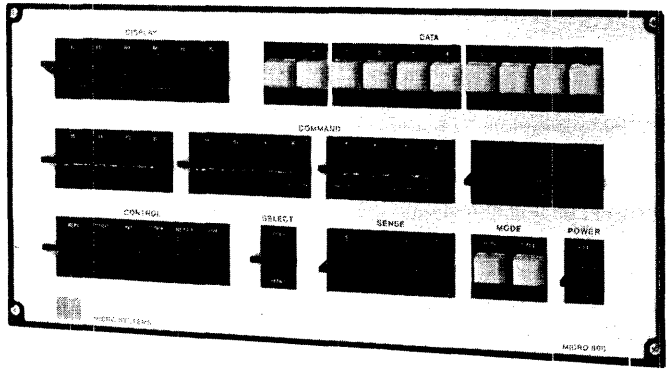


Figure 17. System Console

- Four sense switches
- Six control switches, including run, step, interrupt, clock, reset and save
- Power on-off

DISPLAYS

Run Lamp

The run lamp is illuminated when the processor is running.

Halt Lamp

The halt lamp is illuminated when the power is on and the process is not running.

SWITCHES

Display Selector

These seven interlocked switches select the register or bus to be displayed on the system console. The displays which can be selected are: L register, M register, N register, eight high order bits of the read-only memory output, eight low order bits of the read-only memory and the A bus. When the machine is halted the output of the read-only memory is the same as the contents of the R register, and is the next command to be executed.

Command

These 16 alternate action switches are substituted for the read-only storage on the system console when the SELECT switch is in the PANEL position. Depressing the CLOCK switch causes the command set on the switches to be executed. The command may also be executed repeatedly by depressing the RUN switch. These switches are used to gate registers to the A bus display and for entering data into the fjle and registers.

Select

This alternate action switch selects the console panel command switches (PANEL) or the read only memory (ROM) as the command to be executed next. This switch is not available on the basic console.

Sense

The four alternate action sense switches are available on both consoles. The state of these switches may be transferred to a file register or machine register by the control command. These switches may be used to provide manual control of micro level and macro level programs.

Run

This momentary contact switch places the processor in the run mode causing it to execute microcommands.

Step

This momentary contact switch places the processor in the run mode and as long as the switch is depressed causes an internal interrupt. The halt internal interrupt is bit 7 of the internal status. This switch is normally microprogrammed to cause a processor halt. Since the processor is forced to run when the switch is depressed, the machine can be microprogrammed to cause a single macro instruction to be executed.

Interrupt

This momentary contact switch places the processor in the run mode and causes an internal interrupt. The console interrupt is bit 0 of the internal status. This switch is normally microprogrammed to cause a console interrupt.

Clock

This momentary contact switch causes the processor to execute a single microcommand. If the processor is running at the time the switch is depressed, the processor will come to a forced halt following the current microcommand execution.

Reset

This momentary contact switch halts the processor and clears the L register, I/O control register and other control flip flops. The reset is made available to I/O devices. Since the current microcommand execution will not be completed, the computer should not be stopped by this switch.

Save

This alternate action switch is the same as the RESET switch but can be set on providing a continuous reset. If this switch is on at the time the power is turned on or off the contents of the memory will not be lost or altered.

OPERATING PROCEDURES – SYSTEM CONSOLE

Execution of Commands from the front panel of the System Console

Most microcommands can be executed from the front panel by using the command switches to simulate read only memory. These commands can be used to check-out most of the MICRO 800 logic, and also to gain familiarity with the microcommand set. The following list of commands is a minimum that should be tried out when first becoming acquainted with the MICRO 800.

For the examples all command switch settings and displays are shown in hexadecimal.

1. Loading and stepping the L register
   a. Load L
      1) Set CLOCK, RESET
      2) Set 'SELECT' to panel
      3) Select L display
      4) Set the following commands into the command switches, and press the CLOCK switch once for each

Setting Switches	Display
14AA	0AA
1455	055
15FF	1FF
1C11	211
1DEE	3EE

b. Step L

1) Set SELECT to ROM

2) Set RESET

3) Select L display

4) Each time the CLOCK switch is pressed, the L count should increment, skip, or jump. If no ROM board is plugged in, the L count will step.

2. Test M and N

1) Set SELECT to PANEL

2) Display to M or N

3) Set the following command into the command switches and press the clock switch once for each.

1255	Load M	M = 55
13AA	Load N	N = AA, M = 0

Try other values and repeat.

3. Test ROM and L register (with 810 firmware).

1) Set SELECT to ROM

2) Set RESET

3) Select L, or R2 or R1

4)

L	R2	R1	
000	BF	02	} Repeatedly press the CLOCK
001	2B	00	
002	2A	00	
003	40	10	

After this, the L value depends on computer register states, because of conditional skips and jumps.

4. Test the T register

1) Set SELECT to PANEL

2) Set DISPLAY to D (A bus)

3) Set the following sequences into the command switches and press the CLOCK switch.

11AA	CLOCK	Load T
B020		Display T = AA with copy T
1155	CLOCK	Load T
B020		Display T = 55 with Copy T

Try other values and repeat.

5. Test the File Registers

a. Load and Read each File

1) Set SELECT to PANEL

2) Set DISPLAY to D (A bus)

3) Set the following sequences into the command switches and press the CLOCK switch.

21AA	CLOCK	Load file 1 with 'AA'
C100		OR 0 with file 1 (Display file 1)

OBSERVE 'AA'

Repeat with file numbers 2-F and different data patterns.

b. Load all files first, and then read back.

1) SELECT to PANEL

2) DISPLAY to D (A bus)

3) Set command switches to 2111, press CLOCK, change command switches to 2222, press CLOCK. Repeat up to 2FFF.

4) Display file 1 with 8100 or C100, and repeat for 8200, 8300, etc., to 8F00.

c. Test Add to File

1) SELECT to PANEL

2) DISPLAY to D

3) Set command switches to 2100 (clear file 1), press CLOCK.

4) Set command switches to 3101 (Add 1 to file 1). Display will be at 01 before CLOCK is pressed. Each time CLOCK is pressed, display will increment.

5) Repeat for different file values and increment sizes (3102, etc.).

6. Test various Arithmetic, Logic and Shift Commands

1) SELECT to PANEL

2) DISPLAY to D

a. ADD

1101	CLOCK	01 $\longrightarrow$ T
2101	CLOCK	01 $\longrightarrow$ f_1
8120		$(f_1)+(T) \longrightarrow f_1$ Initial display=02

Each time CLOCK is pressed display will increment.

1101	CLOCK	01 $\longrightarrow$ T
2101	CLOCK	01 $\longrightarrow$ f_1
8121		$(f_1)+(T) \longrightarrow f_1$, T Initial display=02

Each time CLOCK is pressed display value will double.

2100	CLOCK	
8140		(file 1)+1 $\longrightarrow$ file 1 Display = 01

Repeat with different initial values in f_1 and T.

Change destination register to M and N and display these directly while repeating above tests.

b. SUBTRACT

1101	CLOCK	01 ⟶ T
21FF	CLOCK	FF ⟶ f_1
9120		(f_1)-(T) ⟶ f_1 Display = FE

Each time CLOCK is pressed display value will decrement.

Repeat for other values in f_1 and $\overline{T}$.

c. Logic Functions

	11AA	CLOCK	AA ⟶ T
	21CC	CLOCK	CC ⟶ f_1
OR	C120	C140	Display result of logic function
EXOR	D120	D140	
AND	E120	E140	

Table of Values:

c field = 2	OR	EX OR	AND
T	10101010	10101010	10101010
f_1	11001100	11001100	11001100
Display	11101110	01100110	10001000

c field = 4			
$\overline{T}$	01010101	01010101	01010101
f_1	11001100	11001100	11001100
	11011101	10011001	01000100

d. Shift

1) 2101 CLOCK 01 ⟶ file 1
F100 Display bit shifted left 1

Each time CLOCK is pressed the bit will shift left one place.

2) 2100 CLOCK 00 ⟶ f_1
F100 CLOCK Clears link
2101 CLOCK 01 ⟶ f_1
F18D Display bit shifted left 1

Repeated pressing of CLOCK will cause a left shift with end around carry.

F120 Causes Right Shift
F140 Causes Left Shift insert ones.

7. Load and Read Memory

This requires setting M and N, loading T, and executing a write memory command to load. To read, set M and N, execute a read memory command, and a B020 to display T. Set SELECT on PANEL and display on D.

Load core location 0210 with AA.

1310	10 → N, 0 → M
1202	02 → M
11AA	AA → T
A010	Write memory

Read core location 0010

1310	10 → N, 0 → M
A000	Read memory
B020	Display T

8. Enter Sense Switches

Set SELECT to PANEL

DISPLAY to D

Set command switches to 7010, sense switch settings will appear on display as follows:

Binary	
X X X X	1 1 1 1
switch settings	all 1's

9. Shift file right 4

Set SELECT to PANEL

DISPLAY to D

21A0	CLOCK	A0 → f_1
7120		Shift right 4

Display = FA

10. Test U Register

The lower 4 bits of the U register can easily be loaded and tested by observing its effect on a file display command. First load all files in sequence with the file number for a value. Then load U with a Cf value, followed by an execute command, with 0000 set on the command switches. This will cause the value of file f to be displayed.

1) Set SELECT to PANEL
2) DISPLAY to D
3) Load files

2101	CLOCK	01 ⟶ f_1
2202	CLOCK	02 ⟶ f_2

(repeat for all files up to F)

4) Load U

16C1	CLOCK	C1 ⟶ U

5) Set 0000 on command switches
Display 01 from file 1

6) Load U

1CC2	CLOCK	C2 ⟶ U

7) Set 0000 on command switches
Display 02 from file 2

Repeat 6 and 7 for all files to F.

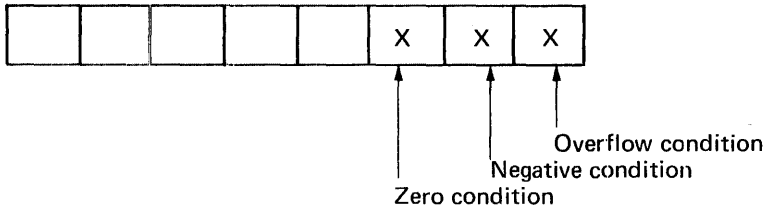
11. Set and Display Condition Flags and Link

The flags can be displayed by the command C000 (file 0 ∨ 0⟶file 0), and the link can be displayed by B080. Copy link⟶no destination, which displays the link as LSB on the A bus.

1) Set SELECT to PANEL
2) DISPLAY to D
3) Depress CLOCK for first two instructions only:

Load File	2101		2100	zero condition
Add to file	8110		8100	
Display link	B080	Link = 0	B080	Link = 0
Display flags	C000	All flags = 0	C000	Flag = 1
	2180	Negative	217F	Overflow
	8110	cond.	8150	
	B080	Link = 0	B080	Link = 0
	C000	Flag = 1	C000	Flag = 1
	21FF	Zero cond.		
	8150			
	B080	Link = 1		
	C000	Flag = 1		

Condition Flag Display (File zero)



Explanation of Codes

8110	Add 0 to file 1	Update Condition flags
8150	Add 1 to file 1	Update Condition flags
B080	Copy Link to no destination except A bus	
C000	OR file 0 with no operand	

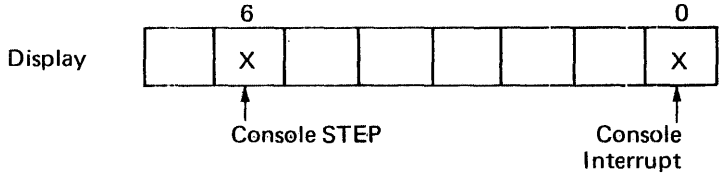
12. External Internal Status

This instruction will demonstrate sensing of the console STEP, and console interrupt inputs.

1) Set SELECT to PANEL
2) DISPLAY to D

Command switches

7040



Press console STEP and interrupt switches, and observe changes in bits 0 and 6.

CHAPTER 6

PROGRAMMING SYSTEMS FOR MICRO 800 FIRMWARE DEVELOPMENT

The programming systems for the MICRO 800 computer permits the user to develop special application firmware at a cost and turnaround time that is now comparable to software development in competitively priced fixed instruction computers. This chapter describes the assemblers, operating systems, simulator and use of the Alterable Read Only Memory system which are used as standard aids in microprogramming. In addition, procedures for checkout and debugging of microprograms are provided.

AP800 CROSS ASSEMBLER

AP800 is a symbolic assembly program for the MICRO 800 computer. The assembler provides for symbolic addressing and mnemonics for machine and assembler instructions. This program is written in FORTRAN IV and may be adapted to many computer systems. The MICRO 800 source program is entered by punch cards and the output of the assembler includes an assembly listing, read only storage diode map, and an object program card deck.

MAP800 CROSS ASSEMBLER

MAP800 is a two pass symbolic assembly program which allows for assembly of MICRO 800 microprograms on the MICRO 810 or MICRO 820 computer. It is designed to operate using an ASR 33 Teletype with paper tape reader and punch. Output consists of an assembly listing and an object program paper tape for use by the MICRO 800 simulator program, SIM800.

The assembly language includes the following features:

Address Arithmetic – Decimal and hexadecimal numbers, symbolic addresses, and arithmetic expressions.

Listing Control – The format of the listing may be controlled with comment cards included.

Diagnostics – Diagnostics for source program errors included in the output listing.

Option Flags – Single letter flags to signify options to microcommands.

SYMBOLIC LANGUAGE

The source language is a sequence of symbolic commands, called statements. Each statement is written on a single line and may consist of from one to four entries: a name field, an operation field, an operand field, and a comments field.

Name Field

The name field entry is always a symbol. The first character of a symbol is alphabetic or a period; subsequent characters may be

alphabetic, numeric, or a period. A name entry is usually optional. When an asterisk, *, appears as the first character the remainder of the line is considered as comment. The type of command determines the legal content of the name field.

Operation Field

The operation field entry is a mnemonic operation code specifying the machine command or assembler instruction. The field begins with the first non blank character following the name field in paper tape or with column 8 in cards. All machine command mnemonics are two characters except those of the operate class where no destination register is designated. The operate class commands have a basic single letter mnemonics. If the result of the operation is to be sent to a machine register then the register identifier character, r, is appended as the second character of the mnemonic. Register identifier characters are shown below. An asterisk, *, is appended to the mnemonic if the result of the operation is not to be placed in the designated file register. Some of the mnemonics accepted by the assembler are commonly used forms of other commands.

Register Designator	r	Register
0		None.
1	T	T Register.
2	M	M Register.
3	N	N Register.
4	L	L Register Addresses: 000-0FF and 200-2FF.
5	K	L Register Addresses: 100-1FF and 300-3FF.
6	U	U Register.
7	S	U Register ORed in command (Except for K command).

Operand Field

The operand field entries provide the file register designators, literals, and option bits for the machine commands. The operand field may start anywhere following the operation field. When punched in cards, column 14 is the normal starting column. It is terminated by the first blank. One or more operands, separated by commas may be written, depending on the needs of the command. All entries in the operand field, except the single character option bit identifiers for the operate class commands, are expressions. An expression is a symbol, decimal number, or hexadecimal number, or a combination of these terms made by + and — operators.

The following single character option identifiers, designators and literals may appear in the operand field.

L — Link Control.
I — Add one or insert one on Shift.
D — Decrement one.
T — T register operand.
F — Complement of T register operand.
H — Half cycle memory operation (otherwise full cycle).
R — Right shift (otherwise left shift).
C — Set condition flags.

f – File register designator (0-15)
c – Option code (0-15)
n – Literal (8, 9, or 10 bit)

Comments Field

Comments describing the information about the program may be inserted between the end of the operand field and column 72. All characters, including spaces, may be used in writing a comment. If the listing is printed on a teletype, only the first 53 characters of the source line are printed.

MACHINE COMMANDS

Machine commands are expressed by a one or two character mnemonic code in the operation field. The required operands depend on the command type. The four syntax types are described below. Examples of the method of writing machine commands in the assembly language are shown in the sample listing in section 5.

Load Register Commands (Command 1)

All commands of this syntax type have two character mnemonics beginning with L, except for the jump command. The second character is the register identifier character. The operand field of all commands of this type except jump must contain a single operand which is an expression, whose value is less than 1024 and greater or equal to -256. It is evaluated modulo 256. The jump commands must contain an operand expression which has a positive value less than 1024.

Literal-File Commands

The commands of this syntax group (commands 2-6), have two character mnemonics and require two operands. The first operand is an expression which designates a file register (f) and must be in the range 0-15. The second operand (n) is an expression which must be less than 1024 and greater than or equal to -256. It is evaluated modulo 256

Execute and Control Commands

The commands of this syntax group have operation code mnemonics identical to those of the next group, and require two operands. The first operand is an expression which designates a file register (f) and must be in the range 0-15. The second operand (c) is an expression which designates the option bits (7-4) and must be in the range 0-15.

Operate Class Commands other than Execute and Control

The commands of this syntax group have basic operation code mnemonics which are a single character. If the result of the operation is to be routed to a machine register the designator of that register is appended as a second character of the mnemonic. If the result is not to be placed in the designated file register, an * is appended to the mnemonic.

OPERAND FIELD EXPRESSIONS

Expressions in the operand field are made up of one or more terms which are connected by + and – arithmetic operators. No parenthetical expressions are allowed. Each term of the expression represents a value. Values

Command	Mnemonics	Operand Field
Load T	LT	n
Load M	LM	n
Load N	LN	n
Load U	LU	n
Load Zero Control	LZ (L)	n
Load Seven Control	LS	n
Jump	JP	n
Load File	LF	f,n
Add to File	AF	f,n
Test If Zero	TZ	f,n
Test If Not Zero	TN	f,n
Compare	CP	f,n
Execute	Er*	f,c
Control	Kr*	f,c
Add	Ar*	f,L,I,T,C
Increment	Ir*	f,L,C
Subtract	Sr*	f,L,D,T,C
Decrement	Dr*	f,L,C
Copy	Cr*	f,L,I,T,C
Read	Rr*	f,L,I,D,H
Write	Wr*	f,L,I,D,H
Logical OR	Or*	f,L,F,T,C
Move	Mr*	f,L,C
Exclusive-OR	Xr*	f,L,F,T,C
Logical AND	Nr*	f,L,F,T,C
Shift	Hr*	f,L,I,R,C

may be assigned by the assembler program (symbols), or there may be inherent in the term itself (constants). The range of values depends on the operand and the instruction.

Symbols

A symbol is composed of one to three characters in MAP800, or one to six characters in AP800. The first character must be alphabetic or period; subsequent characters may be numeric, alphabetic, or period. Imbedded blanks are not allowed and the assembler stops scanning the symbol with the first character which is not alphanumeric or a period. All symbols, except the special symbol *, used in an operand field, must be defined by a single appearance in the name field of statement within the program.

Special Symbol

The special symbol * represents the momentary values of the assembler's location counter. It may be used as any other symbol in an expression but must never appear in the name field.

ALPHABETIC LIST OF COMMANDS

Command	Mnemonic	Operation Code	Page
AND	N	Ef	118
Add	A	8f	103
Add To File	AF	3f	92
Compare	CP	6f	96
Control	K	7f	96
Copy	C	Bf	112
Exclusive-OR	X	Df	116
Execute	E	0	123
Jump	JP	14,15,1C,1D	89
Load File	LF	2f	91
Load T	LT	11,19	85
Load M	LM	12	85
Load N	LN	13	86
Load U	LU	16	86
Load Seven Control	LS	17	88
Load Zero Control	LZ (L)	10	87
OR	O	Cf	114
Read	R	Af	108
Shift	H	Ff	120
Subtract	S	9f	106
Test if Zero	TZ	4f	93
Test if Not Zero	TN	5f	94
Write	W	Af	108

Constants

The values of the constant terms are not assigned by the assembler program but are inherent in the terms. There are two types of constant terms: decimal and hexadecimal.

a. Decimal Constant

A decimal constant is an unsigned decimal number. The value must be less than 65,536.

b. Hexadecimal Constant

A hexadecimal constant is an unsigned hexadecimal number of up to four characters written as a sequence of hexadecimal digits. The digits are enclosed in single quotation marks and preceded by the letter X. Each hexadecimal digit represents a four-bit binary number. The characters A through F are used to identify the hexadecimal integers 10 through 15.

ASSEMBLER INSTRUCTIONS

Seven assembler instructions are included for control of the assembly process and the output listing.

ORG – Set Location Counter

The ORG assembler instruction alters the setting of the location counter. The name field entry, if any, will be assigned the value of the program

counter after it is altered. The operand field of ORG must contain an expression whose value will be placed in the location counter. All symbols in the expression must have been previously defined when the instruction is first encountered. The next command which places object code in the program is forced to begin a new object card.

EQU – Equate Symbol

The EQU assembler instruction is used to define a symbol by assigning to it the value of the operand field. Any symbols appearing in the expression must have been previously defined when the instruction is first encountered. A name field entry must be present.

DC – Define Constant

The DC assembler instruction is used to create any microcommand for which a symbolic representation does not exist. Each statement specified only one constant. The constant is written as an expression and is assembled as a 16-bit word in storage.

END – End Assembly

The END assembler instruction terminates the assembly of a program and must be the last statement in a source program.

The next three descriptions are available only in the AP800 version.

IDENT – Program Identification

The IDENT assembler instruction is used to identify the start of a program and to supply the program name which is located in the operand field. The IDENT must be the first statement in a source program.

SPACE – Space Listing

The SPACE assembler instruction causes one or more blank lines to be inserted into the listing. The name field is disregarded by the assembler. The operand field contains an expression specifying the number of blank lines. If the spacing is beyond the end of the current page, the listing begins at the top of the next page.

EJECT – Start New Listing Page

The EJECT instruction causes the next line of the listing to appear at the top of the next page. The name and operand fields are disregarded by the assembler.

ASSEMBLY LISTING AND DIODE MAP

The output listing from the assembler contains the memory address, and contents of words in the object program. The source statement is printed side-by-side with the object code.

FORMAT FOR AP800

Printer Columns	Contents
8 - 11	Error flags
15 - 17	Storage address
21 - 24	Storage contents
31 - 110	Source statement

ERROR FLAGS

A – Address Error

This error occurs when an address expression in the operand field is incorrectly written or the value is out of range for one of the operands. An error flag will occur for each operand in error or out of range.

F – Flag Error

This error occurs when an operate class command has an option flag in the operand field which is not allowed for the command or is unrecognizable.

M – Multidefined Symbol Error

This error occurs when the symbol in the name field has been previously defined by appearing in the name field of another instruction.

N – Name Field Error

This error occurs when the symbol in the name field starts with a character other than alphabetic or period, or contains a non alphanumeric or non period character.

O – Operation Mnemonic Error

This error occurs when the assembler does not recognize the contents of the operation field starting in column 8. A zero value is assembled to allow patching.

U – Undefined Symbol Error

This error occurs when the symbol encountered in an expression of the operand field is not defined by an appearance in the name field.

DIODE MAP FOR AP800

The read only memory diode map is printed if a control card following the END card contains a 1, 2 or 3 in column 1. The digit specifies the number of diode maps to be printed. The diode map for each 256 word read only memory board is placed on three pages of the assembly listing. The format of the map is the same as the physical layout of the ROM board. An X on the map indicates a 1-bit and that a diode is to be placed at the position of the X, while an 0 indicates a 0-bit and no diode.

Each of the 64 lines of the diode map for a board contains the diodes for four words. The address of the first word is printed at the left of the map. The four words are interleaved so that the same bit position in each of the four words are grouped together and printed as a cluster at four diode positions. The 16 bit positions are printed across the page and the sum of the number of diodes on the line is placed at the right of the map.

SAMPLE LISTING

In order to illustrate assembly language programming, three examples are included in this manual (Figures 18, 19 and 20). The first is a set of unrelated commands assembled by AP800 showing how to write various commands. The second is a listing of a portion of the MICRO 810 firmware assembled by MAP800. The third example is a sample coding sheet with a portion of a program on it.

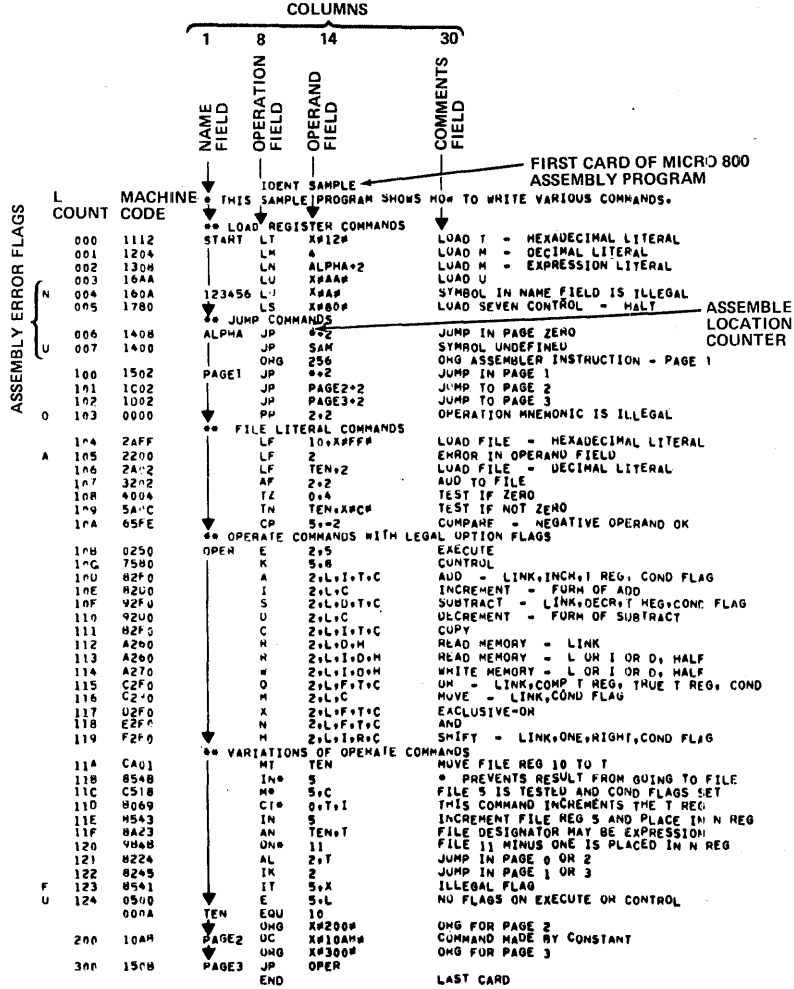
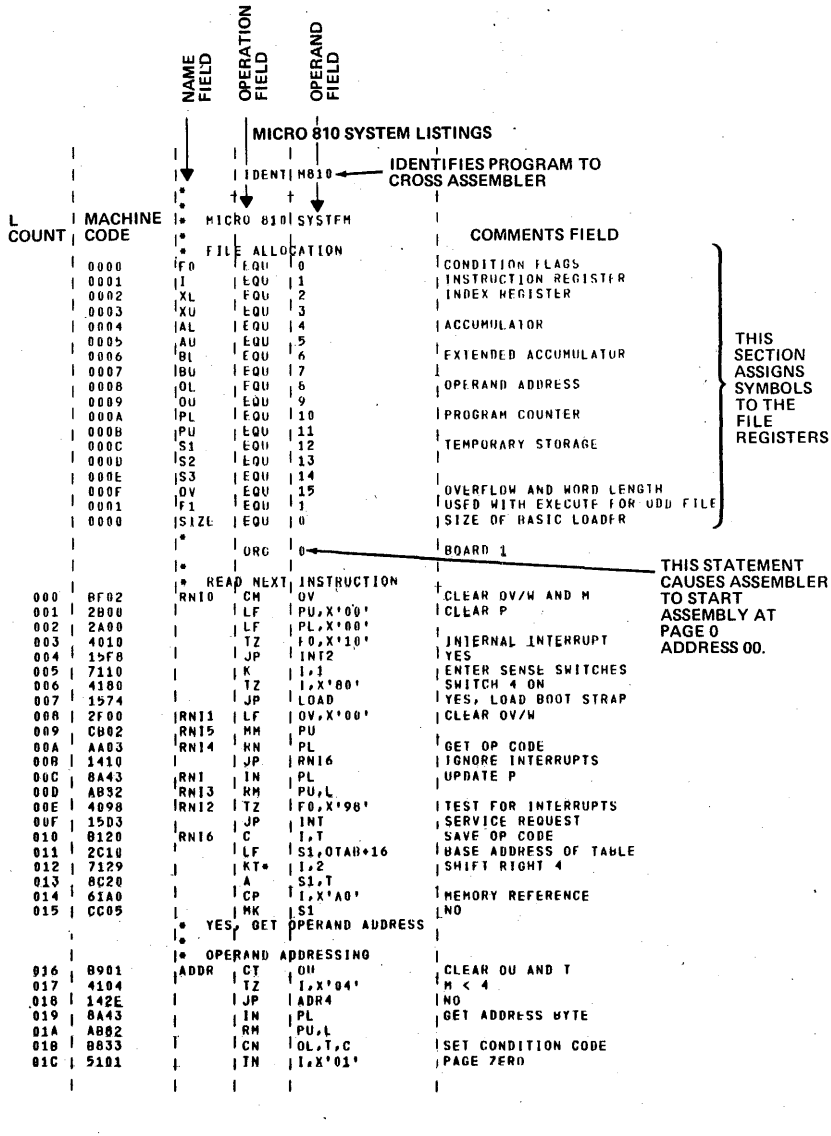


Figure 18. Sample Listing




```
                *  MICRO 810 SYSTEM
                *
                *  FILE ALLOCATION
0000            F0      EQU     0       CONDITION FLAGS
0001            I       EQU     1       INSTRUCTION REGISTER
0002            XL      EQU     2       INDEX REGISTER
0003            XU      EQU     3
0004            AL      EQU     4       ACCUMULATOR
0005            AU      EQU     5
0006            BL      EQU     6       EXTENDED ACCUMULATOR
0007            BU      EQU     7
0008            OL      EQU     8       OPERAND ADDRESS
0009            OU      EQU     9
000A            PL      EQU     10      PROGRAM COUNTER
000B            PU      EQU     11
000C            S1      EQU     12      TEMPORARY STORAGE
000D            S2      EQU     13
000E            S3      EQU     14
000F            OV      EQU     15      OVERFLOW AND WORD LENGTH
0001            F1      EQU     1       USED WITH EXECUTE FOR ODD FILE
0000            SIZE    EQU     0       SIZE OF BASIC LOADER
                *       ORG     0       BOARD 1
                *
                *  READ NEXT INSTRUCTION
000   BF02      RNI0    CM      OV      CLEAR OV/W AND M
001   2B00              LF      PU,X'00'  CLEAR P
002   2A00              LF      PL,X'00'
003   4010              TZ      F0,X'10'  INTERNAL INTERRUPT
004   15F8              JP      INT2    YES
005   7110              K       I,1     ENTER SENSE SWITCHES
006   4180              TZ      I,X'80'  SWITCH 4 ON
007   1574              JP      LOAD    YES, LOAD BOOT STRAP
008   2F00      RNI1    LF      OV,X'00'  CLEAR OV/W
009   CB02      RNI5    MM      PU
00A   AA03      RNI4    RN      PL      GET OP CODE
00B   1410              JP      RNI6    IGNORE INTERRUPTS
00C   8A43      RNI     IN      PL      UPDATE P
00D   AB32      RNI3    RM      PU,L
00E   4098      RNI2    TZ      F0,X'98'  TEST FOR INTERRUPTS
00F   15D3              JP      INT     SERVICE REQUEST
010   8120      RNI6    C       I,T     SAVE OP CODE
011   2C10              LF      S1,OTAB+16  BASE ADDRESS OF TABLE
012   7129              RT*     I,2     SHIFT RIGHT 4
013   8C20              A       S1,T
014   61A0              CP      I,X'A0'  MEMORY REFERENCE
015   CC05              MK      S1      NO
                *  YES, GET OPERAND ADDRESS
                *
                *  OPERAND ADDRESSING
016   B901      ADDR    CT      OM      CLEAR OU AND T
017   4104              TZ      I,X'04'  M < 4
018   142E              JP      ADR4    NO
019   8A43              IN      PL      GET ADDRESS BYTE
01A   AB82              RM      PU,L
01B   B833              CN      OL,T,C  SET CONDITION CODE
01C   5101              IN      I,X'01'  PAGE ZERO
```

Figure 19. MICRO 810 System Listings

MICRO 800 SERIES SYMBOLIC CODING FORM

PROGRAM NAME: SAMPLE | AUTHOR: | PROJECT NUMBER: M800 | DATE: | PAGE OF: | CARD NUMBER:

NAME	OPERATION	OPERAND FIELD	COMMENTS
	ORG	0	
*			
*	READ NEXT	INSTRUCTION	
RNIO	CM	OV	CLEAR OV/M AND M
	LF	PU,X'00'	CLEAR P
	LF	PL,X'00'	
	TZ	FO,X'10'	INTERNAL INTERRUPT
	JP	INIT2	YES
	K	I,1	ENTER SENSE SWITCHES
	TZ	I,X'80'	SWITCH 4 ON
	JP	LOAD	YES LOAD BOOT STRAP
RNI1	LF	OV,X'00'	CLEAR OV/M
RNI5	MM	PU	

FORM 9-133

Figure 20. MICRO 800 Series

OPERATION PROGRAM CARD DECK FROM AP800

The assembly program generates a deck of cards which contain the binary object code, if a control card following the END card contains a 0 in column 2. All information punched on the cards is in Hollerith code, with a single hexadecimal digit (four binary bits) punch in each column. This format allows easy visual reading of the cards after they are interpreted and permits rapid patching or generation of patches to the deck. Each card contains 16 program words. If all 16 words are zero, the card is not punched.

The cards have two fields as follows:

Columns 1-4 — Load address.
Columns 5-68 — Object code, four columns per word

The format of the binary paper tape created by MAP800 is described under Simulator Operating System.

SIMULATOR OPERATING SYSTEM (SOS) AND SIMULATOR PROGRAM (SIM800)

INTRODUCTION

The Simulator Operating System (SOS) is an on-line executive system for controlling the operations of the MICRO 800 simulator (SIM800) and incorporates teletype control of debug, console, and executive functions. The teletype is used rather than any console operations except for the console interrupt, which is used to cause control to return to SOS while the simulator is operating. SIM800 and SOS are always loaded into the MICRO 810 or 820 as a single program because all simulator operations are controlled by SOS.

The following is a list of the features available to the user:

Display and change the content of a simulated read only memory location.

Display and change the content of a simulated core memory location.

Two breakpoints for microprogram debugging.

Display and change the content of a simulated MICRO 800 element.

Display the content of all simulated MICRO 800 elements.

Simulate execution of a microprogram.

Load a formatted program tape into simulated read only memory.

Load a formatted tape into simulated core memory.

Punch the content of simulated read only memory into paper tape.

Punch the content of simulated core memory into paper tape.

INSTRUCTIONS FOR USE

This section provides instructions for using the SOS program.

Loading the SOS and SIM800 by the bootstrap and basic loaders

The SOS is loaded into memory via the basic paper tape loader. This basic loader is in the bootstrap format (1 data byte per frame of tape) and is spliced onto the front of the SOS tape. The splice is made so that the last frame of the loader is followed immediately with the leader of the SOS tape. The microprogrammed bootstrap loader loads the basic loader and transfers control to it. Then the basic loader loads the SOS and, after a successful load, transfers control to the SOS. Following is a procedure for loading a formatted paper tape through the teletype via the bootstrap and basic loaders.

1. Place the TTY in the off-line mode, place the reader control lever to the "free" position and enable the teletype reader. Type control and Q.
2. Place the TTY in the on-line mode and insert the SOS tape in the reader with the first rub-out character at the read station. Set the reader control lever in the stop (center) position.
3. Set the front panel sense switches as follows:

   Sense switch 1: off for serial TTY interface, on for parallel TTY interface.

   Sense switch 2: must be off.

   Sense switch 3: must be off.

   Sense switch 4: must be on. This selects the bootstrap loader whenever the run switch is selected and was preceded by a reset.
4. Press the reset and the run switches and the system will wait for the teletype reader to be started.
5. Press the TTY reader lever to the start position. When the basic loader is loaded and operating properly, the teletype page printer mechanism will chatter whenever a record separator passes the read station. This is caused by the issuance of reader off and reader on codes between records.

If a checksum error is found, the message 'CE' is typed and the system will halt. Another attempt to properly load the record may be accomplished by backing up the tape to the previous record separator, placing the reader control lever in the stop (center) position, and pressing the run switch on the front console. When the SOS is properly loaded, control will transfer to it, the teletype bell will ring, and an equal sign will be typed.

Loading the SOS and SIM800 by the R Operator of TOS

Unroll about 30 inches of the program tape to bypass the basic loader and locate the leader (any frame with channel 8 present) of the formatted tape. Insert the tape into the reader with any part of the leader at the read station and set the reader control lever to center position. Typing an R will start the loading. A checksum is calculated for each record loaded and if it doesn't equal the checksum read with the record, the letters 'CE'

will be typed and control will return to the standard teletype operating system program (TOS). By backing up the tape to the previous separator and typing an R, another attempt may be made to load the tape.

SOS Operators

All operations which are performed by SOS are initiated by typing a single alphabetic character which designates one of 13 operators. These operators are described in detail in Section 3 and are summarized in Appendix A.

The SOS program is ready to accept an operator designator character at any time after ringing the bell and typing an equal sign. If a character other than a legal operator designator is typed, SOS will reject the character, ring the bell, and type an equal sign again.

NOTE: For the purposes of this manual, all references to the teletype carriage return are as shown; (CR).

Hexadecimal Input/Output

All data and addresses are displayed and entered in hexadecimal. The 16 hexadecimal digits are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The hexadecimal values may not be signed. When entering a two-digit memory cell value or a four-digit memory address, no spaces or other than hexadecimal characters may be in the digit string. SOS assumes that the hexadecimal digit string is terminated when it receives the first non-hexadecimal character; therefore, it will not act on an input until the digit string is terminated. If more than the required number of digits are entered, SOS will take the last two or four as required. Leading zero digits need not be typed. If the first non-hexadecimal character is not a space, comma, or carriage return (CR), the data or address value is ignored and the operation is terminated. However, before termination, all valid hexadecimal data or address values that were accepted are retained. When more than one address or data value is typed they may be separated by either a comma or a space. For clarity in this document only commas are shown. When an operator requires an address, it will ignore leading spaces, i.e.:

W ssss, eeee (CR)

Console Interrupt

The console interrupt is used to interrupt the simulation of a microprogram or to abort the I, O, R, or W operator and return control to SOS. The user should be careful if the simulator is interrupted because complete simulation of the current command may not be complete but the K,L register will be pointing to the next sequential location.

If the console interrupt is activated when control is residing in SOS (waiting for an operator), an exit is made to the resident TOS. When using the serial teletype interface, the exit is not taken until one character is typed on the keyboard to force completion of the IBS instruction.

Halt and Error Returns

If a microcommand halt (1780) is detected, control will return to SOS and an H followed by the content of the K,L register plus one will be typed.

During the simulation of microprograms, various undefined microcommands and system timing violations are checked for and if detected will cause an error return to SOS. The letter E and a three digit error number will be typed, followed by the content of the K,L register plus one, and control will return to SOS. A list of the error codes and their meaning are contained in Appendix B.

OPERATORS

Card Read: C

The C operator causes SOS to load a program card deck into simulated ROS. The format of the cards must be as described in the AP800 Assembly Program manual. Loading is terminated and control is returned to SOS, when a card is read containing a blank in column 5. If a blank card is read, any character other than a hexadecimal character is read, or a card reader malfunction occurs; the message ERR will be typed and control will return to SOS. Loading may continue, by correcting the error condition, backing up one card, starting the reader, and typing a C. Since no information goes through the reader when a blank card is read or when a pick failure occurs, it is not necessary to back up one card.

Display: Dn

The D operation causes the contents of the simulated system element n to be typed out followed by a dash. At this time the contents of the element may be changed by typing in one or two hexadecimal digits. When a comma or space is typed after the data or after the dash, the contents of the next element in sequence will be displayed. The various simulated system elements (n) and their meaning are listed below in sequence. If a (CR) is typed, or if a space or comma is typed after the contents of the panel switches (P) has been displayed, this operator is terminated. All examination must be completed on one line of type.

List of values for "n", in order of their appearance:

0 . . 9 A . . F	Files 0 through F
T	T Register
M	M Register
N	N Register
K	(L Register Bits 9, 8)
L	L Register (Bit 7-0)
U	U Register
Z	Link flip-flop (1 bit)
Q	R Register (Bits 15-8)
R	R Register (Bits 7-0)
S	Internal Status Register
I	Input bus
O	I/O Control Register (3 bits)
P	Panel command switches (7-0)

Display: D (CR)

This mode of the D operator causes all of the simulator system elements to be typed out on two lines. A single space is provided between each element and there is a double space after every fourth element. Sixteen files are contained on line one with thirteen additional elements being displayed on line two in the following manner.

D (CR)

```
00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
01 02 03 04  05 06 07 08  09 0A 0B 0C  0D
```

End of Tape: E

The E operation punches an end of tape record consisting of a zero record size and an execution address of zero. This ensures that tapes punched by SOS will not contain a load and go address. Following the punching of this record, six inches of trailer will be punched automatically.

Go To: G ssss, tttt, uuuu (CR)

The G operation causes SOS to set trap operations for read only memory locations tttt and uuuu, and to start simulation at read only memory location ssss. If a (CR) is typed after G, simulation starts at the location contained in the K,L register. If a (CR) is typed after ssss, no traps are set, and if a (CR) is typed after tttt only one trap is set. All traps set are automatically cleared when either one is reached or control is transferred to SOS, signalled by the ringing of the teletype bell and the printing of an equal sign. Upon return from a trap, a T, followed by the contents of the K,L registers, is typed out. At this time the command located at the trap location has not been executed. A trap at location zero is not permitted as this value is used by SOS to indicate that a trap has not been set.

Input: I

The I operator causes SOS to load a MICRO 800 program tape into simulated read only memory in the same manner as the R operator loads a formatted tape into core memory. The tape may be created by the O operator of SOS or by the MAP800 assembler.

Leader/Trailer: L

The L operator will cause the paper tape punching device to punch six inches of tape containing channel eight punches only.

Memory: M ssss,

The M operator causes the contents of the simulated memory location specified by ssss to be typed out followed by a dash. At this time the contents of the memory location may be changed by typing in two hexadecimal digits. When a space or comma is typed after the data or after the dash, the contents of the next sequential location is typed by SOS. A (CR) terminates this operator. The actual amount of simulated core memory will vary depending on the size of the actual memory and the amount of simulated read only memory desired. Standard configuration is 768 words of read only memory and 256 bytes of core memory.

Output: O ssss, eeee (CR)

The O operator causes the contents of the simulated read only memory area starting with ssss and ending with eeee to be written on the standard output device in the same format as with the W operator. Each record will contain 64_{10} commands from read only memory except the last record which will contain a number of commands equal to the total number module 64_{10}. Typing a (CR) following the second address will start the operation.

Print ROM: P ssss,

The P operator causes the simulated read only memory address specified by ssss to be typed out on a new line followed by the contents of that location. A dash is typed after the value to indicate that it may be changed by typing in one to four hexadecimal digits. When a space or comma is typed after the new data or after the dash, the next sequential read only storage address and its contents are typed by SOS on a new line. A (CR) terminates this operator.

Read: R

The R operator causes SOS to load a formatted tape into simulated core memory. This operation can be configured for any standard input device, but normally the device will be the teletype paper tape reader. The tape must be inserted in the reader with the leader (any frame with channel 8 present) placed at the read station before the R is typed. When the loader encounters an end of tape record the loading process is terminated and controls are transferred to SOS. If an end of tape record is not read, loading will continue until the computer is halted or until the console interrupt is activated. A checksum is calculated for each record loaded and if it doesn't equal the checksum read with the record, the letters 'CE' will be typed and control will return to SOS. By backing up the tape to the previous separator and typing an R, another attempt may be made to load the tape.

Time: T

The T operator causes SOS to print the letters I, M, and E, followed by a four digit hexadecimal number and a dash. This number represents the total number of machine cycles accumulated through simulation since the last reset or preset. The counter may be reset or preset by typing in one to four hexadecimal digits before typing a carriage return to terminate the operation.

Write: W ssss, eeee (CR)

The W operation causes the contents of the simulated memory area starting with ssss and ending with eeee to be written on the standard output device, normally the teletype punch. Each record of the output will contain 128_{10} data bytes except the last record which will contain a number of bytes equal to the total byte count module 128_{10}. Typing a (CR) following the second address will start the operation.

Zero Flags: Z

The Z operation causes SOS to reset flags used by the simulator for error detection and to simulate the functions performed by the reset switch on

the front panel. File zero will be cleared, all internal status bits will be cleared, and the K,L register, I/O control register, and the value of the input bus will be set to zero. This operator should be used before setting up parameters and starting a simulation sequence.

PROGRAM TAPE FORMAT

The binary paper tape format (Figure 21), can be generated by the two pass assembler, and by the output and write subroutines of SOS. This format allows for variable length records of up to 64_{10} sixteen-bit micro-commands, or 128_{10} eight-bit bytes, a record load address, and a record checksum. Each record contains a count of the number of data bytes and the 15 bit address at which data is to be loaded. The record is loaded sequentially starting with this address. When there is a discontinuity in the loading addresses, a new record is started so that a load address may be specified. The last byte of each record is a checksum which is the summation of the byte count, load address, and data bytes formed on an eight-bit basis with overflow added into the least significant bit of the sum.

A byte count of zero signifies an end of tape record and if present will be the last record read. The paper tape reader will be stopped and control is returned to SOS.

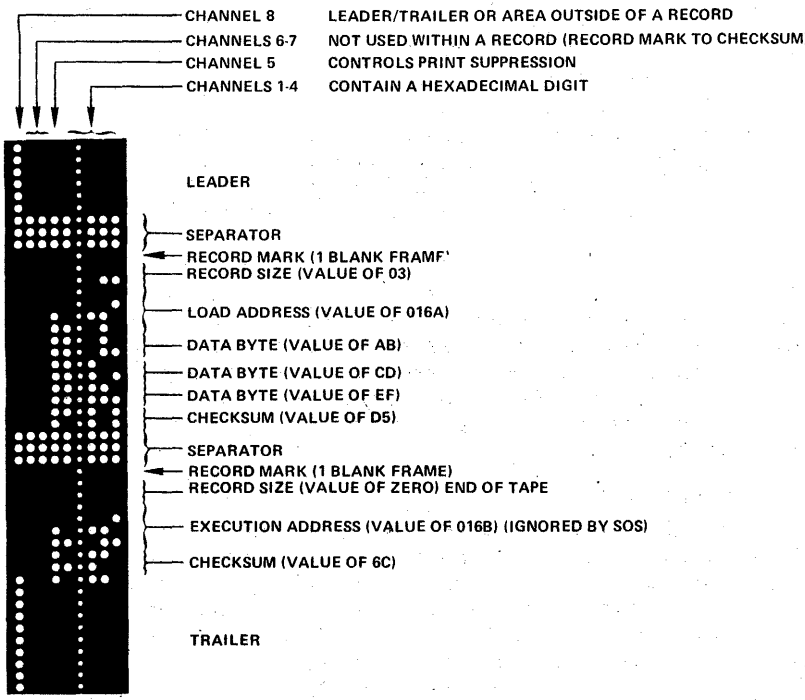


Figure 21. Binary Paper Tape Format

APPENDIXES

APPENDIX A

SUMMARY OF SOS OPERATORS

Underlined items are typed out by SOS:

C	Read a program card deck into simulated ROM.
D1 xx-, xx-nn (CR)	Display content of File 1, leave File 1 unaltered and display content of File 2, change the content to nn and terminate the operation.
D (CR)	Display the content of all simulated elements. Line one contains the 16 files and line two contains 13 additional elements.
E	Write an end of tape record into formatted paper tape.
G (CR)	Simulation starts at the location contained in the K,L register.
G ssss (CR)	Simulation starts at location ssss.
G ssss, tttt (CR)	Simulation starts at location ssss, a trap is set for location tttt.
G ssss, tttt, uuuu (CR)	Simulation starts at location ssss, traps are set for locations tttt and uuuu.
G, tttt (CR)	Simulation starts at the location contained in the K,L register, a trap is set for location tttt.
G, tttt, uuuu (CR)	Simulation starts at the location contained in the K,L register, traps are set for locations tttt and uuuu.
I	Input a formatted program tape to simulated read only memory. After loading, control returns to SOS.
L	Punch six inches of paper tape leader (channel 8 only).
M ssss, xx-nn, xx- (CR)	Display the contents of simulated memory location ssss and change the contents to nn. Display the contents of location ssss+1, leave the location unaltered and terminate the operation. This operation must be completed on one line of type.
O ssss, tttt (CR)	Output the contents of simulated read only memory from locations ssss through tttt into formatted paper tape.

P ssss, ssss xxxx-, ssss xxxx-nnnn (CR)	Print the content of simulated read only memory location ssss, leave the location unaltered and display the content of location ssss+1. Change the content of ssss+1 to nnnn and terminate the operation.
R	Read a formatted paper tape into simulated core memory. After loading, control returns to SOS.
TIME xxxx- O (CR)	Display the number of machine cycles accumulated during simulation. Reset the time to zero and terminate the operation.
W ssss, tttt (CR)	Write the contents of simulated core locations ssss through tttt into formatted paper tape.
Z	Zero simulator error flags and reset the simulated MICRO 800 system.

APPENDIX B

SIM800: ERROR MESSAGES

#	Meaning
001	U-Register timing – can't use U during first cycle following its setting.
002	Console command switches – Command preceding 707X control command causes an ROM delay.
003	Memory write full cycle – attempt to set T during second, third or fourth cycle following the memory command.
004	Memory read – T is set without being selected, during the first or second cycle following the memory command.
005	Attempt to load literal with an undefined register destination of 8, 9, A, B, E, or F. Destination 9 is undefined because the memory spare bit option is not simulated.
006	Attempt to load or add literal into file register zero.
007	Attempt to use undefined C-bit combinations 3, 5, or 6 in a control command.
008	Console command switches – file register zero not selected in 707X control command.
009	Address in M and N exceeds available simulated memory.
010	Memory write half cycle – attempt to set T during first or second cycle following the memory command.
011	Execute command found after U-register OR-ed into instruction.
012	Undefined B-bus operand – usually resulting from selection of complement T when the input bus (103X) is enabled.

ALTERABLE READ-ONLY MEMORY OPERATING SYSTEM (AROS)

INTRODUCTION

The Alterable Read-Only Memory Operating System (AROS) is a program which permits on-line control, loading and dumping of firmware code using the teletypewriter and/or card reader. The program is used in conjunction with Microdata's Alterable Read-Only Memory System (AROM). The AROM system described in Part VI "Product Catalog" is a valuable tool for checkout of firmware systems. It is particularly useful in real-time firmware or I/O oriented applications that require precise timing to be analyzed which cannot be done with the simulator system.

The features of the AROS program include the following:

Loading of the AROM system (memory) with firmware code in the form of formatted punched cards or punched paper tapes.

Display and/or change of operator designated AROM locations using the teletypewriter.

Listing and/or dumping of AROM on teletypewriter and punched paper tape.

INSTRUCTIONS FOR USE

This section provides instructions for using the AROS program.

Loading AROS by the bootstrap and basic loaders

The AROS is loaded into memory via the basic paper tape loader. This basic loader is in the bootstrap format (1 data byte per frame of tape) and is spliced onto the front of the AROS tape. The splice is made so that the last frame of the loader is followed immediately with the leader of the AROS tape. The microprogrammed bootstrap loader loads the basic loader and transfers control to it. Then the basic loader loads the AROS and, after a successful load, transfers control to the AROS. Following is a procedure for loading a formatted paper tape through the teletype via the bootstrap and basic loaders.

1. Place the TTY in the off-line mode, place the reader control lever to the "free" position and enable the teletype reader. Type control and Q.
2. Place the TTY in the on-line mode and insert the AROS tape in the reader with the first sub-out character at the read station. Set the reader control lever in the stop (center) position.
3. Set the front panel sense switches as follows:

   Sense switch 1: off for serial TTY interface, on for parallel TTY interface.

   Sense switch 2: must be off.

   Sense switch 3: must be off.

   Sense switch 4: must be on. This selects the bootstrap loader whenever the run switch is selected and was preceded by a reset.

4. Press the reset and the run switches and the system will wait for the teletype reader to be started.
5. Press the TTY reader lever to the start position. When the basic loader is loaded and operating properly, the teletype page printer mechanism will chatter whenever a record separator passes the read station. This is caused by the issuance of reader off and reader on codes between records.

If a checksum error is found, the message "CE" is typed and the system will halt. Another attempt to properly load the record may be accomplished by backing up the tape to the previous record separator, placing the reader control lever in the stop (center) position, and pressing the run switch on the front console. When the AROS is properly loaded, control will transfer to it, the teletype bell will ring, and an at sign (@) will be typed.

Loading AROS by the R Operator of TOS

Unroll about 30 inches of the program tape to bypass the basic loader and locate the leader (any frame with channel 8 present) of the formatted tape. Insert the tape into the reader with any part of the leader at the read station and set the reader control lever to center position. Typing an R will start the loading. A checksum is calculated for each record loaded and if it doesn't equal the checksum read with the record, the letters "CE" will be typed and control will return to TOS. By backing up the tape to the previous separator and typing an R, another attempt may be made to load the tape.

AROS Operators

All operations which are performed by AROS are initiated by typing a single alphabetic character which designates one of 10 operators.

The AROS program is ready to accept an operator designator character at any time after ringing the bell and typing at sign (@). If a character other than a legal operator designator is typed, AROS will reject the character, ring the bell, and type an at sign (@) again.

NOTE: For the purposes of this manual, all references to the teletype carriage return are shown as; (CR).

Hexadecimal Input/Output

All data and addresses are displayed and entered in hexadecimal. The 16 hexadecimal digits are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The hexadecimal values may not be signed. When entering a four digit data value or a four digit memory address, no spaces or other than hexadecimal characters may be in the digit string. AROS assumes that the hexadecimal digit string is terminated when it receives the first non-hexadecimal character. Therefore, it will not act on an input until the digit string is terminated. If more than the required number of digits are entered, AROS will take the last four as required.

Leading zero digits need not be typed. If the first non-hexadecimal character is not a space, comma, or carriage return (CR), the data or address value is ignored and the operation is terminated. However, before termination, all valid hexadecimal data or address values that were accepted are retained. When more than one address is typed they may be separated

by either a comma or a space. For clarity in this document only commas are shown. When an operator requires an address, it will ignore leading spaces, i.e.:

W ssss, eeee (CR)

Console Interrupt

The console interrupt may be used to terminate the D, R, V, and W operations, return control to AROS and type a carriage return, line feed, bell, and at sign (@). If the console interrupt is activated when control is residing in AROS (waiting for an operator), an exit is made to the resident TOS. When using the serial teletype interface, the exit is not taken until one character is typed on the keyboard to force completion of the IBS instruction.

OPERATORS

Card Read: C

The C operator causes AROS to load a program card deck into reference ROS. The format of the cards must be as described in the AP800 Assembly Program manual. Loading is terminated and control is returned to AROS, when a card is read containing a blank in column 5. If a blank card is read, any character other than a hexadecimal character is read, or a card reader malfunction occurs; the message ERR will be typed and control will return to AROS. Loading may continue, by correcting the error condition, backing up one card, starting the reader, and typing a C. Since no information goes through the reader when a blank card is read or when a pick failure occurs, it is not necessary to back up one card.

Dump: D ssss, eeee (CR)

The D operation causes the contents of AROM to be dumped on the teletype printer starting with the address ssss and ending with the address eeee. AROS types the four digit address at the left margin followed by eight 16-bit words of AROM. This operation is terminated when the contents of the last AROM location has been typed, or the console interrupt is activated. Typing a (CR) after the second address will start the operation.

End of Tape: E

The E operation punches an end of tape record consisting of a zero record size, a zero address, and a zero checksum followed by six inches of tape containing channel eight punches only.

Leader: L

The L operator will cause the punching device to punch six inches of tape containing channel eight punches only.

Print Reference ROS: P ssss,

The P operator causes the reference ROS address specified by ssss to be typed out on a new line followed by the contents of that location. A dash is typed after the value to indicate that it may be changed by typing in one to four hexadecimal digits. When a comma or space is typed, after the new data or after the dash, the next sequential reference ROS address and its contents are typed on a new line. A (CR) terminates the operation.

Read: R

The R operator causes AROS to load a formatted tape into reference ROS. The tape must be inserted into the teletype reader with the leader (any frame with channel 8 present) placed at the read station before the R is typed. When the loader encounters an end of tape record, the loading process is terminated and control is returned to AROS. If an end of tape record is not read, loading will continue until the reader is empty or until the console interrupt is activated. A checksum is calculated for each record loaded, and if it doesn't equal the checksum read with the record, the letters 'CE' will be typed and control will return to AROS. By backing up the tape to the previous separator and typing an R, another attempt may be made to load the tape.

Transfer: T ssss, eeee (CR)

The T operation causes a clock of reference ROS starting with location ssss and ending with location eeee to be transferred to the corresponding locations in AROM. The operation is started by typing a (CR) following the second address and is terminated when the contents of the last location specified is transferred. There is no verification or check of the data written made by this operator.

Verify: V ssss, eeee (CR)

The V operation causes a block of AROM starting with location ssss and ending with location eeee to be read and compared with the corresponding locations in reference ROS. All variances will be displayed along with their associated address. The operation is started by typing a (CR) following the second address. Termination occurs when the last specified location is checked and a message is typed or the console interrupt is activated.

Write: W ssss, eeee (CR)

The W operation causes the contents of reference ROS starting with location ssss and ending with location eeee to be written on the standard output device, normally the teletype punch. Each record of the output will contain 64_{10} 16-bit words, except the last record, which will contain the number of words equal to the total word count modulo 64_{10}. Typing a (CR) following the second address will start the operation.

Zero: Z ssss, eeee (CR)

The Z operator causes the reference ROS locations starting with ssss and ending with eeee to be set to zero. Typing a (CR) following the second address will start the operation.

PROGRAM TAPE FORMAT

The binary paper tape format (Figure 22) can be generated by the MAP800 assembler, by the O operator of the simulator and by the W operator of AROS. This format allows for variable length records of up to 64_{10} 16-bit words (punched as 128 bytes), a record load address (address X 2), and a record checksum. Each record contains a byte count of the number of data bytes and the address at which loading is to start. The last byte of each record is a checksum which is the summation of the byte count, load address, and data bytes formed on an eight bit basis with overflow added into the least significant bit of the sum.

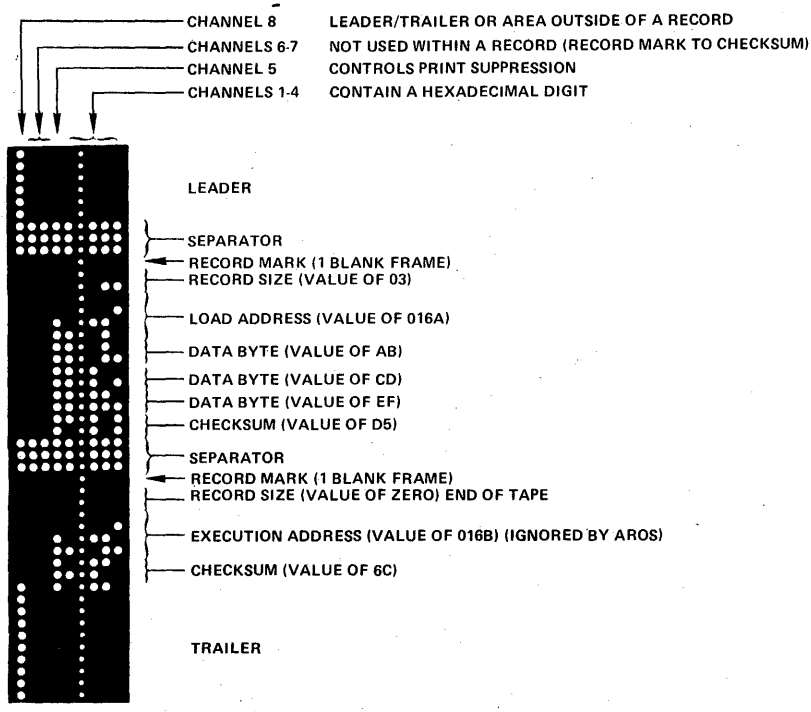


Figure 22. Binary Paper Tape Format

SUMMARY OF AROS OPERATORS

Underlined items are typed out by AROS:

C	Read a program card deck into reference ROS.
D ssss, eeee (CR)	Dump the contents of AROM locations ssss through eeee onto the teletype printer. Each line will contain an address and up to eight 16-bit values.
E	Write an end of tape record into formatted paper tape.
L	Punch six inches of paper tape leader (channel 8 only).
P ssss, ssss xxxx-, ssss xxxx-nnnn (CR)	Print the content of reference ROS location ssss, leave the location unaltered and display the content of location ssss+1. Change the content of ssss+1 to nnnn and terminate the operation.

R	Read a formatted paper tape into reference ROS. After loading, control returns to AROS.
T ssss, eeee (CR)	Transfer the block of reference ROS from ssss to eeee to the corresponding locations in AROM.
V ssss, eeee (CR) LOC ROM REF ssss xxxx yyyy Verify Completed	Verify the block of AROM from ssss to eeee to the corresponding locations in reference ROS. An error is indicated at location ssss.
W ssss, eeee (CR)	Write the contents of reference ROS locations ssss through eeee into formatted paper tape.
Z ssss, eeee (CR)	Set the contents of reference ROS locations ssss through eeee to zero.

PROGRAM CHECKOUT AND DEBUGGING

After a program has been written and assembled, the program debugging phase begins. Depending on the size and complexity of the program, and the care used in preparing the program, this phase may be routine, requiring only a few hours, or it may require many days.

The simulator is very useful for debugging because commands can be easily modified to correct errors or to help in finding errors. This also applies to the Alterable Read Only Storage (AROS).

Programs can also be checked and modified quite easily even if they have already been put in diode read only memory.

This discussion of checkout and debugging is divided into four sections:

General Checkout Procedures
Checkout with Simulator
Checkout with AROS
Checkout with Diode Read Only Storage

General Checkout Procedures

There are a number of programming errors which might possibly occur, and are sometimes very difficult to detect. These are the kind that represent valid program commands as far as the assembler and simulator are concerned, thus are not flagged as errors by these two programs. Being aware of the typical errors and their effect on a program helps considerably in locating them.

Some of the error types can definitely cause any one of the symptoms, and these should be checked out first. The procedures for detection and checkout of error symptoms differ for use of the simulator, alterable read only, or diode board, and for that reason will be discussed separately.

Simulator

The simulator is useful for checking internal programs for correct sequences, correctness of results of algorithms, math routines, etc. and for

input output sequences. Since the simulator does not run in real time, it is limited in its ability to test the entire program in normal operation. Also, since it is simulated, it is not possible to step through the program by means of the clock switch and observe the L count and ROS outputs. With the simulator, the ROS can be checked using the teletype, and all files, etc. can be set up using the teletype. Then breakpoints can be placed in the routines and the program can be started at convenient points to test individual routines, or combinations of routines, after the breakpoint is reached.

Some of the more common errors and error symptoms are listed in Table 12.

The reason why all of these are mentioned is that they become the base for establishment of a growing check list which should always be referred to during program checkout. As errors are found in different categories not on the list, they are added to the list. For certain phases of a checkout process, such as checking individual subroutines, obviously all of the error categories don't apply so only selected ones need to be considered.

Many times hours and even days are wasted trying to track down an apparent error cause when a few minutes spent going through the check list would show that a few other items could cause the same symptom. The diagnostic effectiveness of the check list is increased by putting it in the form of a table which relates errors to symptoms, or symptoms to errors. For most cases this table is applicable to checkout with the simulator, AROS, or diode board.

One big advantage of firmware checkout over software checkout is that firmware errors don't cause the program to destroy itself, thus wiping out the error symptoms.

The program error check list relating symptom to error takes on the form shown in the example of Table A. The X's indicate the most likely relations between program errors and symptoms, although under certain conditions any one of the symptoms might be caused by any of the program error types. The various files, registers and flags are tested to see if the routines operated correctly. Once error symptoms are detected, the program errors can be tracked down by the relationships illustrated in Table A.

In Table A the general functions such as algorithms, flow charts, and transfer of flow chart information to coding can introduce errors causing any of the listed symptoms. Therefore, these parts require special checkout on paper before committing to read only storage. One method which proves quite successful in many cases is to define the algorithm and flow chart, and do the coding in MICRO 810, 811 or 820 software as close in format as possible to the firmware coding, and check out these routines first before committing to firmware. This works satisfactorily except for the real time limitation in high speed operations.

Use of the Simulator to Check Subroutines

Two simple subroutines have been selected to illustrate use of the simulator for checkout. The first routine sets files 1-E to 'AA', and the second routine does a simple 8-bit positive number multiply.

The simulator operators to be used for these two examples are as follows:

a. DN — Display files, registers, and flags.

b. G ssss, tttt, uuuu (CR) — Execute a program starting at ssss, with traps at tttt, and uuuu.

c. P ssss — Prints out and permits loading of ROS commands starting at location ssss.

d. Z — Resets flags used by the simulator.

e. D (CR) — Display all Files and Registers.

Routine 1 — Set files 1-E to 'AA'

The U register is used for file indexing. File F is used to contain, and update the U register value. The machine code for this program is as follows:

Example 1. Set files I-E=AA

L Counter Address	Command	Operation
000	2FB0	U Reg. Code to File F.
001	8F46	Update File F and U Reg.
002	11AA	Set T=AA
003	0020	Execute Command (Effectively copy T)
004	6F42	Compare for last file value
005	1401	Jump to repeat loop
006	1780	Halt*

*For demo only, usually a jump or subroutine exit.

Simulator Operations

1. Z — to initialize the simulator.

2. P 000,

000	xxxx	2FB0,
001	xxxx	8F46,
002	xxxx	11AA,
003	xxxx	0020,
004	xxxx	6F42,
005	xxxx	1401,
006	xxxx	1780 (CR)

This part is printed out by simulator.

New commands are typed in followed by comma until last command.

3. G 000 (CR) execute program without traps.

   For correct operation program halts and prints out an H followed by 0007 which is L register +1.

4. Use of D_1, followed by commas, will cause the teletype to print out the content of the files:

   D_1 AA-, AA-, AA, etc.

Typical errors and symptoms:

1. 001 – 8F06 instead of 8F46 – File F not incremented. Program will never exit from loop to halt instruction. No files will be loaded with AA.
2. 004 – 6F41 – File F incremented once too often. Program will loop one extra time, setting file F=AA, which will then cause additional loops storing T into memory at locations determined by M & N. Then program will repeat loading AA into files. Program will never exit loop.
3. 005 – 1400 or 1402 – File F will either be reinitialized every time or nonincremented, so loop will never be exited.

Routine 2 – 8 bit positive number multiply.

x * y→Z_U, Z_L
file 2 = X
file 3 = Y and Z_L
file 4 = Z_U
file 5 = Shift Count

Machine Code

L Counter Address	Command	Operation
000	2508	Shift count = 8
001	C201	Move X to T Register
002	2400	Clear Z_U
003	4301	Test Y for odd/even
004	8420	Add T to T_U
005	F420	Shift Z_U
006	F3A0	Shift Z_L
007	9550	Decrement Shift Count
008	5004	Zero Condition Test
009	1403	Jump to repeat loop
00A	1780	Halt*

*For demo only.

Simulator

1. Z – to initialize this simulator.

2. P 000,
   000 xxxx 2508,
   001 xxxx C201,

   Complete until entire program up to 00A 1780 is loaded.

3. D2, xx-02
   D3, xx-04
   example of 2 x 4 = 8
   x = 2
   y = 4
   $Z_L = 8, Z_U = 0$
4. G 000 (CR) Execute, with no traps.
5. Results

   H 00B Halt location +1

   D3, 08, 00

   Z_L Z_U

Typical errors and symptoms:

1. 008 – 4004 instead of 5004

   The requirement is to skip on zero shift count which would seem like Test Zero is correct. However, the zero condition flag is being tested. This must be =1 when shift cound is 0. A 4004 would cause program not to loop.

2. 007 – 9540 Condition flag not updated.

   Subroutine will never exit because zero condition flag will never be set.

   If flag had been set when routine was entered, exiting would occur on first pass.

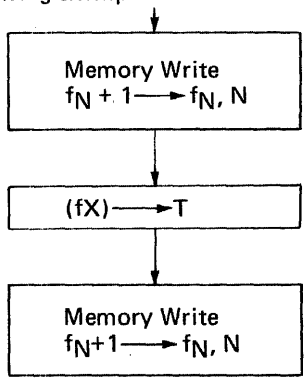
3. 006 – F320 Link not entered.

   With this error, the program would loop properly and exit to the halt, but the Z_L value in file 3 would always be 0.

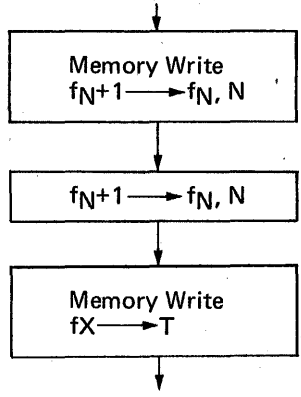
As larger and more complex subroutines and entire programs consisting of many subroutines are checked out, more of the error sources included in Table A must be considered.

Many times, if a timing error for memory access or I/O is found, it can be corrected without addition of instructions requiring relocation by changing the order of instructions or changing a no-op to a jump to next instruction to increase a delay factor.

Consider the following example:



Assume that this is a programming error because the value in fX is not supposed to be stored until the 2nd memory write cycle shown. The routine could be changed to the following:



The same number of instructions are required, but instruction 2 which causes modification of N will cause a delay until the first memory cycle is complete, thus causing f(X) to go into memory on the 2nd cycle. Changes of this type are particularly important when the program being checked is in diode read only memory.

Checkout of an applications microprogram can be facilitated by preparation of simple programs for display of registers and core memory and placing these in the upper part of the read only memory.

Also checkout of short firmware subroutines is facilitated by using a MICRO 810, 811, or 820 having an additional ROS which is electrically alterable by the program. Then the software programs can be used to test core memory and to display most of the file registers.

Checking Subroutines with the Alterable Read Only Storage

An alterable read only storage (AROS) has the advantages of running in real time as well as ease of command modification.

Programs can be checked out by manually clocking one step at a time while testing the L counter for proper looping, by preparing and testing one subroutine at a time using halt instructions to break up loops, and test partial routine functions. Real time I/O operations can be tested by looping on I/O subroutines, or looping on small groups of subroutines. When the individual routines have been checked, it becomes much easier to assemble and to test the entire program.

Checkout of Programs in Diode Read Only Storage

Programs in diode read only memory should first be manually clocked to see if the L counter follows the correct branching paths, and to check each command in read only storage. File registers are checked at various points in the routine by switching to front panel control and setting command switches to Cf00 and display to D. To bypass loops, the L count is set to the next instruction after the loop. Those items in Table 12 causing all possible symptoms to occur should be checked first. This includes the

diode map, instruction op codes and functions to flow charts, to coding. When stepping through a program, I/O timing cannot be tested in real time, nor can omissions of U register modification delay be detected, therefore these two areas should be thoroughly checked on the flow charts and coding sheets.

To facilitate checkout with diode boards, temporary halt, or loop instructions can be put in the program, and easily changed after the subroutines have been checked out.

Many times in the firmware development phases it is possible to correct an error or omission by placing a jump instruction to an unused part of read only storage, programming the fix there, and jumping back to the first correct instruction after the error. These detours or patches can then be eliminated in the firmware production phase after the firmware program has been checked out.

Table 12. Program Error Check List

← Programming Error

Error Symptom ↓

Incorrect functional definition in program.	Algorithm error (logical, arithmetic, etc.)	Flow chart organiztion error.	Error in, or omission of flow chart function.	Incorrect transfer of function from flow chart to code sheet.	Omission of a command from a routine.	Unexpected conditional timing constraints or processing time load.	Memory access time delays unaccounted for.	Delay after changing U register absent.	Lack of provision for crossing page boundaries (in core or ROS).	Inadvertent double use of file register.	Unexpected conditions causing subroutine to be incorrect or incomplete.	
X	X	X	X	X	X	X	X	X			X	Incorrect or Missing I/O Data
X	X	X	X	X	X	X	X	X		X	X	Incorrect Results but Correct Files Modified
X	X	X	X	X	X			X		X		Incorrect Files Modified
X	X	X	X	X	X	X						Program Hangs up in a Loop
X	X	X	X	X	X							Program Fails to Loop in a Subroutine
X	X	X	X	X	X							Program Exits a Subroutine Loop too early or too late
X	X	X	X	X	X				X			Incorrect Core Memory Storage Locations
X	X	X	X	X	X				X			Core Data/Flags Destroyed
X	X	X	X	X	X		X	X			X	Incorrect Data/Flags Stored
X	X	X	X	X	X				X			Incorrect or No Return From Subroutine
X	X	X	X	X	X				X			Program Never gets to Correct Subroutine
X	X	X	X	X	X	X				X		Intermittent Program Errors
X	X	X	X	X	X		X			X		Program Does Not Enter a Loop According to Expected Flags or Status
X	X	X	X	X	X		X			X		Program Enters Loop when Conditions Say it Should Not
X	X	X	X	X	X	X						Incorrect or Lack of I/O Control Pulses
X	X	X	X	X	X				X			Program Stays in One Page of ROS
X	X	X	X	X	X				X			Program Follows Unexpected Meaningless Path Through Routines
X	X	X	X	X	X				X			Program Jumps to an Unused ROS Area
X	X	X	X	X	X	X		X				Timing Errors

Table 12. Program Error Check List (Continued)

Error Symptom ↓ / Programming Error →	Unexpected overflow condition.	Incorrect skip condition (mainly double inversion).	Improper I/O timing.	Destination register error or omission.	Error in time delay routine calculation.	Not changing jump addresses after routing relocation, or selecting wrong address.	Miscounting for loop exit test.	Omission of file write inhibit.	Incorrect literal in conditional jump command.	Incorrect or omitted term in c field of command.	Errors or omissions in initialization functions.	Inadvertent modification of flag bits in a file.	Incorrect timing, sequences, truth tables, or complement in interface.
Incorrect or Missing I/O Data	X		X	X	X					X			X
Incorrect Results but Correct Files Modified	X			X				X		X		X	
Incorrect Files Modified						X	X						
Program Hangs up in a Loop				X	X	X			X	X			
Program Fails to Loop in a Subroutine		X							X	X			
Program Exits a Subroutine Loop too early or too late							X	X	X	X			
Incorrect Core Memory Storage Locations				X			X	X		X			
Core Data/Flags Destroyed				X			X	X		X			
Incorrect Data/Flags Stored	X			X						X			
Incorrect or No Return From Subroutine				X		X			X				
Program Never gets to Correct Subroutine				X		X							
Intermittent Program Errors					X							X	
Program Does Not Enter a Loop According to Expected Flags or Status									X	X		X	
Program Enters Loop when Conditions Say it Should Not									X	X		X	
Incorrect or Lack of I/O Control Pulses			X		X					X			X
Program Stays in One Page of ROS													
Program Follows Unexpected Meaningless Path Through Routines						X							
Program Jumps to an Unused ROS Area						X							
Timing Errors					X		X						X

Table 12. Program Error Check List (Continued)

Programming Error

Error Symptom	Incorrect setting of sense switches.	Not accounting for special results, such as 1's introduced by shift right 4 command.	Selection of wrong entry point for multiple entry subroutines.	Incorrect subroutine jump pointer usage.	Incorrect accounting for internal flags.	Use of incorrect command op code.	Not setting M register after N register.	Diodes or bits incorrectly placed in ROS.	Unplanned modification of link bit or a condition flag between command which sets it and command which uses it.
Incorrect or Missing I/O Data						X		X	
Incorrect Results but Correct Files Modified		X	X			X		X	X
Incorrect Files Modified		X	X			X		X	
Program Hangs up in a Loop	X		X			X		X	
Program Fails to Loop in a Subroutine			X			X		X	
Program Exits a Subroutine Loop too early or too late						X		X	
Incorrect Core Memory Storage Locations			X			X	X	X	
Core Data/Flags Destroyed			X		X	X	X	X	
Incorrect Data/Flags Stored			X			X		X	
Incorrect or No Return From Subroutine				X		X		X	
Program Never gets to Correct Subroutine		X		X		X		X	
Intermittent Program Errors						X		X	X
Program Does Not Enter a Loop According to Expected Flags or Status	X				X	X		X	
Program Enters Loop when Conditions Say it Should Not						X		X	X
Incorrect or Lack of I/O Control Pulses						X		X	
Program Stays in One Page of ROS						X		X	
Program Follows Unexpected Meaningless Path Through Routines						X		X	
Program Jumps to an Unused ROS Area		X		X		X		X	
Timing Errors						X		X	

CHAPTER 7

TECHNIQUES AND EXAMPLES

TECHNIQUES FOR EFFICIENT MICROPROGRAMMING

In many aspects microprogramming is similar to assembly language software programming of small computers. There are basic arithmetic, logic, I/O, control, and memory functions. Programs are organized with executives and subroutines. Jumps and return jumps can be made. The basic differences are as follows:

- There are no variable addressing modes at the microcommand level. Memory accesses must be programmed on a step-by-step basis, with commands to set memory address, and to transfer data to and from T, which is the memory transfer register.
- Execution of commands is much faster than in a software machine.
- I/O functions must be programmed on a step-by-step basis, including setting up device connect codes in T, and programming input and output strobe pulses.
- Return jumps must be set up by storing return addresses in a file register.
- Arithmetic shift, control and logic functions are all register oriented, and are limited in scope, such as shift one bit position, add 8 bits, 8 bit logic, skip only one location, etc.
- The command or instruction memory is semi-permanent read only memory with a limited capacity, so that much care must be taken to conserve the number of commands or instructions in the program.
- The commands or instructions are much more intimately related to the machine architecture, and to bit patterns, therefore some knowledge of logic Boolean algebra, and small computer organization is highly desirable, and is applied to the programs.
- Interrupts are monitored by status sampling rather than hardware interrupts as found in software programmed machines.
- All commands or instructions are single word (16 bits) and relate to files, or register.
- Commands are organized in such a manner as to make is possible sometimes to do more than one function on a command, and this is necessary many times to conserve commands.
- The flexibility of programmable alteration commands is not as great as with software programs. A special register, called the U register, is necessary for this function.
- There are two levels of high-speed storage – the file register and core memory. The files are general purpose at the microprogramming level.
- There are special commands in microcode not normally found in software commands, such as shift right 4, load zero, and literal to register, which simplify many functions.
- There are certain timing constraints related to I/O, memory, skips, jumps, and U register applications, which must be taken into account when preparing microprograms.

Even with all of the above constraints, it is possible to have microprograms which are 10-50 times as fast as equivalent software programs and which require the same or fewer instructions than a software program.

In order to make full use of the power of microprogramming a large number of techniques are possible to reduce the number of instructions, and/or to reduce execution time.

The following techniques are discussed in this next section:

1. Generation of delays for memory accesses, U register applications, and input/output.
2. Double functions on a single command.
3. Uses, setting and testing of Link.
4. Uses of U register.
5. Setting and using of condition flags.
6. Use of loops vs straight line programming.
7. Small general purpose subroutines.
8. Use of shift right 4 instruction (generated with and without U).
9. Use of files for flags, counters, and reference data.
10. Organization of Op codes, file, and core allocations to reduce instructions.
11. Saving diodes by selection of instructions and files.
12. Saving jump instructions when branching.
13. Reducing two branches to one by multifunction commands, and commands which become effective No Ops in one branch.
14. Interlacing vs cascading of routines.
15. Uses of inhibit file write.
16. Moving data from file to register.

1. **Generation of delays for memory accesses, U register applications and input/output.**

Each of these items requires a delay of 1 to 3 clock times after the command. The desirable thing to do is some required function which provides the delay with no error. For example, on a memory write, T must not be written into for 4 clock times. On the 32-bit input example (#2) the write memory command is followed by reset DIXX, a skip test, and a jump. None of these affect T, so the entire memory delay is achieved with no loss of execution time. The memory time is then reduced from 1.1 μs to .22 μs. Also in this same example, the one clock delay after DIXX, prior to data input is achieved by advancing the byte address counter, thus avoiding a No Op. Most of the input and output delays can be generated by updating program counters, and addresses, etc. Microprogram Example No. 10 contains many of this type command. Microprogram Example No. 12A shows an example of placing a memory access command after updating U to provide a delay without a No Op.

2. **Double functions on a single command.**

The following double functions can be done, and should always be used when possible:

a. Clear both a file and register with a copy 0 command. Similar techniques can be used to set both equal to 01, OR, FF.

b. Update a file or register on a memory command. (This does not have to be a memory register.)

c. Update a file, or register on an I/O control command. (Output moves only.)

3. **Uses, setting and testing Link.**

Link is used to indicate carry for an arithmetic function, or the shifted out bit on a shift function. It is used for multibyte arithmetic, shifting, or memory address incrementing.

Link can be preset by shifting a file, with inhibit file write. If link is to be set specifically to 1 or 0, it may be accomplished by subtracting zero or adding zero to any selected file regardless of its contents. For sign extension on a shift, link is preset to whatever value is in the end bit of the designated file.

The state of link can be tested without disturbing a file by executing a shift right command with the following c field functions: inhibit file write, enter link, and update the condition flags. The link appears in the MSB which sets or resets the negative condition flag. If the condition flags must be saved, then link can be entered into MSB or LSB of a file, and tested. Link can also be tested by entering into a file using the copy command as well as the shift command.

If link is used in a routine, care must be taken to avoid setting or resetting it on a function before the time it is to be tested.

4. **Uses of the U register.**

The U register is used for file indexing, and command modification. It is ORed with the upper 8 bits of the execute command or operate commands (except control) which select destination register value 7. Typical modifications are as follows:

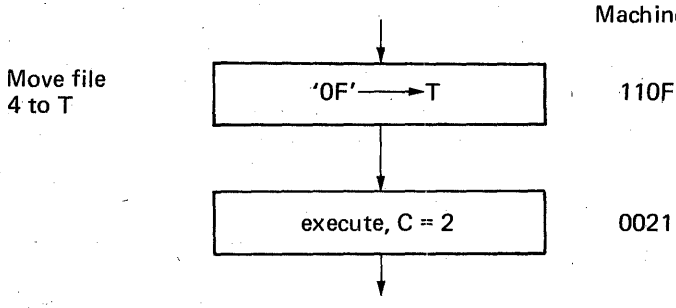
a. Execute 0020

The 2 in the c field selects T for add, subtract, logic functions, and copy. Therefore the 0020 can be used for multi-purpose command execution, by loading U with the desired Op code, and file register number.

For moving, loading or clearing a group of files, the Op code will remain fixed, and only the file number will change. In this case, the Op code for copy ('B') or move ('C') can be used with a 0 for the file number.

When U has been set, the new value does not become effective until the second clock. Sometimes two entirely different functions can be implemented using U. For example, if it is necessary to move the

upper 4 bits or alternately the lower 4 bits of a file to the T register, this can be done as follows:



Case one: move upper 4 (U) = 74

with c = 2 this becomes 7421 shift right 4⟶T

Case two: Move lower 4 U = E4

with C = 2 this becomes E421
And f_4 with T⟶T

If a number of different functions are to be done to a register in one pass through a loop, the operate command with destination code 7 is used. This can not be used if a destination register is required.

5. **Setting and using condition flags.**

The three condition flags are overflow, negative, zero. The condition flags remain unchanged unless the c field in an operate command is set for updating condition flags, or a control command is executed. The zero condition flag is used to test for arithmetic zero conditions, and for end of a subroutine loop. Condition flags can be set without changing files. Some of the techniques are as follows:

a. fa + 0⟶C by inhibiting file write, and adding 0, the condition flags for a file state can be set.

b. T⟶f0, C by copying T and inhibiting file write, the condition flags for a T state can be tested.

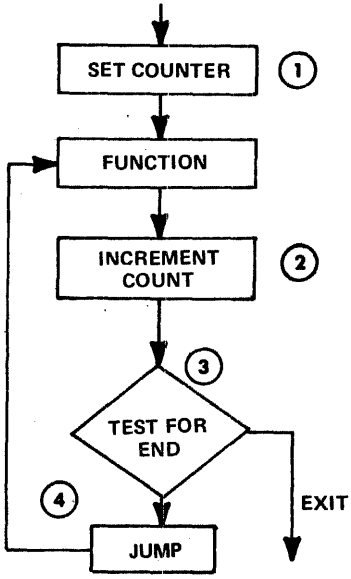
c. enter sense switches to f0⟶C Sense switch 4 can be used to set the negative condition flag without affecting any register.

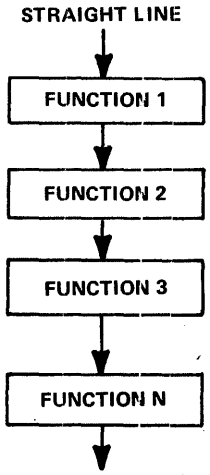
d. fA + T⟶C setting C for normal add function.

e. Copy Link⟶C Set negatitive, and zero condition flags.

6. **Use of loops vs straight line programming**

The two main factors of consideration are execution time and number of commands. If the number of commands using a straight line approach is five or less, there are no command savings using a loop because four commands are required to set up the loop as shown:





The loop takes much longer than the straight line approach. A typical loop is shown in Example 7. In this routine there would be nine functional commands per input byte for a total of 36 for four bytes. Using a loop reduces the command count to 12 commands. The straight line approach takes 7.94 us instead of 10.56 us as in Example 7. Therefore if time were very critical it might be desirable to use the straight line approach.

7. **Small general purpose subroutines.**

To reduce the total number of commands in a microprogram, subroutines can be used in a manner similar to software programs.

To jump to a routine on the same page requires 2 or 3 instructions, one for the return address, one for the jump, and usually one to set a flag, pointer, etc., for the subroutine. Therefore if the subroutine requires only 4 or 5 instructions it is not worth making as a standard. If the routine, such as a general purpose I/O routine requires 10 or so instructions and is used more than once, then it is definitely of value to make the routine general purpose.

8. **Use of shift right 4 command.**

This command is used to transfer the upper four bits of a file to the lower four in the file and/or to a destination register. The upper four are replaced with 1's, which may or may not have to be cleared. To clear the 1's, simply add '10' to the file after shifting. If the value is an Op code to be tested, the 1's can be treated as a constant. If the result is to be subtracted from another value obtained by similar means, the 1's will cancel.

9. **Use of file register for flags, counters, and reference data.**

File registers are used for routine control words as well as data. When it is necessary to conserve files, flags, etc., are sometimes stored in core between routines so that file register meanings may change during a microprogram. Also files can sometimes serve a dual function by judicious location of flags. In Example 19, there is a subroutine which must perform differently on alternate passes. On one pass there is an effective shift right 4 leaving 1's to be cleared. One file contains a flag to indicate which pass it is. This flag is also placed in bit position 4; therefore the file content can be added to the file containing 1's to be cleared, thus serving a dual function. Also a file assigned to update U can be used as the loop program counter.

10. **Organization of Op codes, file register numbers, and core memory addresses to minimize commands.**

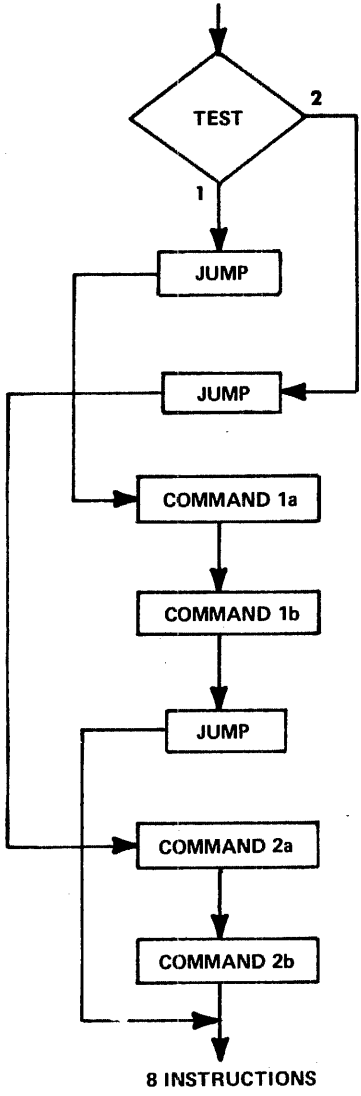
Many times it is possible to use particular files to make their addresses correspond to memory addresses, such as in Example 12A. This will save both files and commands. Also locating a block of data in one page saves an instruction. Use of file F for an instruction which may be either a shift or add will minimize instructions, as shown in Example 19.

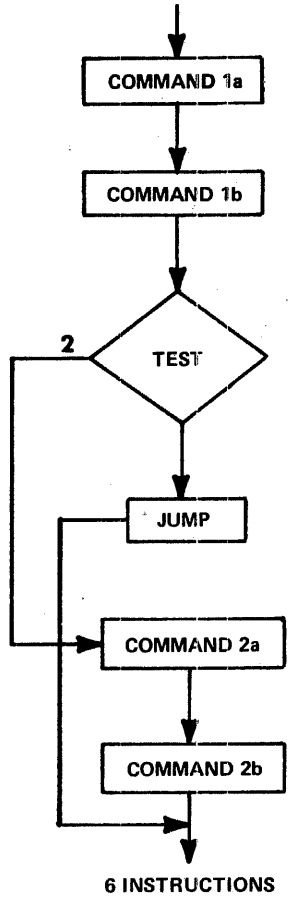
11. **Saving of diodes by selection of files and instructions.**

If possible files used very often should have numbers which have the least number of diodes. If there is a choice of TZ, TN, or using condition flags vs. testing the file directly, the method which requires the fewest diodes should be used, particularly if there are very many ROM's to be built using discrete diodes.

12. **Saving jump instructions when branching.**

This example shows that if there are two branches, each having two or more commands, doing one of the branches first reduces the number of commands by two.

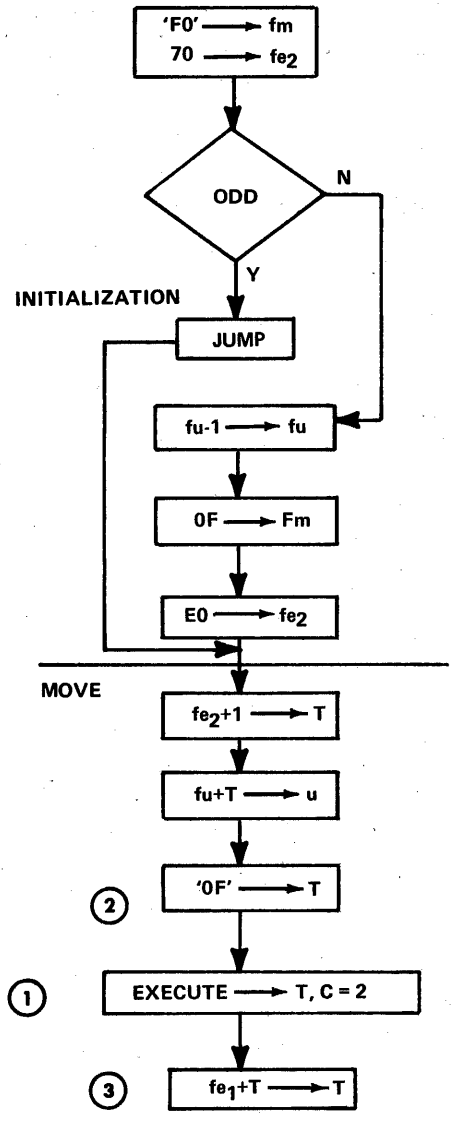




13. **Reducing two branches to one by multi-function commands which become effective No Ops in one branch.**

Many times a function varies with program state, such as moving upper or lower half of a byte in BCD manipulations. Sometimes widely varying functions can be combined by organizing the routine for the worst case function, and having some of its steps become effective no ops for the simpler functions.

This is illustrated in Example 19.



The odd state is for moving the upper byte. The even for the lower byte. If odd, the pertinent state when entering 'move' is

$fe_2 = '70'$

With this stage the value in U becomes 7f

Control — Selected file register

This causes a shift R 4 at ① with result to T, which nullifies command ②

'0F' → T

If the state is even, the state of fe_2, entering the move is E0. This causes U to become Ef which is the And function. This causes the contents of f to be Anded with (f) with result to T. In this case the '0F' loaded in T causes selection of only the lower half of (f). The next instruction ③ fe_1+T⟶T adds '10' to T if in the odd state, which clears the 1's resulting from the shift R. If in the even state, fe_1 contains '00' so command ③ is an effective no OP.

14. **Interlacing vs. cascading of subroutines.**

What this means is entering a subroutine and remaining until an operation is complete, vs. doing parts of routines, and moving on to subsequent routines before finishing. Cascading results in the fewest instructions, but can drastically reduce throughput, if the routines are time paced by external devices, such as card readers, serial teletypes, line printers, in which case the microprogram must wait for data to be supplied by the interface. For example, teletype lines should be monitored by the microprogram on a bit sample basis instead of assembling an entire character. More commands are required to store and fetch pointers and status bits and to test for status, but the throughput improvements are worth the extra coding, and sometimes an absolute necessity.

15. **Use of inhibit file write.**

Inhibit file write is used for the following functions:

a. Setting registers without changing the content of a file.

b. Presetting Link using shift or arithmetic functions.

c. Presetting the condition flags without changing the state of a file.

16. **Moving data from a file to a register.**

Normally data is moved from a file to a register using the OR function because it doesn't affect link. If the state of link is not needed, the move can be implemented using the Add 0 to file with a savings of one diode and always resetting Link.

MICROPROGRAMMING EXAMPLES

The following Microprogramming Examples illustrate basic microprogramming techniques. Many routines, such as the 8-bit positive number multiply have been simplified from standard routines by omitting such capabilities as handling negative numbers as well as positive numbers. For a more detailed description of typical subroutines, and an entire program, refer to Part IV—MICRO 810 firmware reference manual.

Most of the routines do not contain the linkages to an executive program, such as setting return addresses, etc., because these vary with the type of executive in which the routine may be used.

Some of the routines were selected only as examples to illustrate certain microprogramming techniques, and may not use the simplest possible algorithm.

The examples are done in flow chart and assembly language coding, along with comments. For normal programming, the comments are not usually as detailed as these examples. Execution times are included to illustrate the high processing rates possible using microprogramming. Machine code is included for the first 15 examples.

The names of the example subroutines are as follows:

1. Multiply 2 Positive 8 Bit Numbers
2. Subroutine Jumps
3. Time Delay Routine
4. Input Data from 4 External Registers
5. Load 8 Successive File Registers from 8 Successive Core Locations
6. 16-bit Addition, Core to File Register
7. Input a 32-Bit Word From an External Device to Core Memory
8. 16-Bit Right Shift with End Around Carry
9. A ORed with B, Result to A
10. Update a 10 BCD Digit Display From Core
11. Clear a Block of Core Memory
12. Read and Write Between 8 Files and 8 Consecutive Core Locations
13. Output From 8 Files to 8 Shift Registers
14. Input From 8 Shift Registers to 8 Files.
15. Input a Block of Data to Core From an A to D Converter
16. BCD to Binary Conversion
17. Binary to BCD Conversion
18. General Purpose Multiple File Shift Routine
19. Hexadecimal to ASCII Conversion Routine
20. General Purpose Code Conversion by Table Lookup
21. Binary Multiply (16 bits)
22. Generate Cyclic Redundancy Code for one 8-Bit Data Byte
23. Generate ASCII Parity

MICROPROGRAM EXAMPLE NO. 1

Multiply Two Positive Numbers $X * Y = z$

Specific Considerations

- Each number 8 bits maximum including sign.
- Result to occupy two 8-bit file registers.
- Numbers to be in file registers before multiply routine.

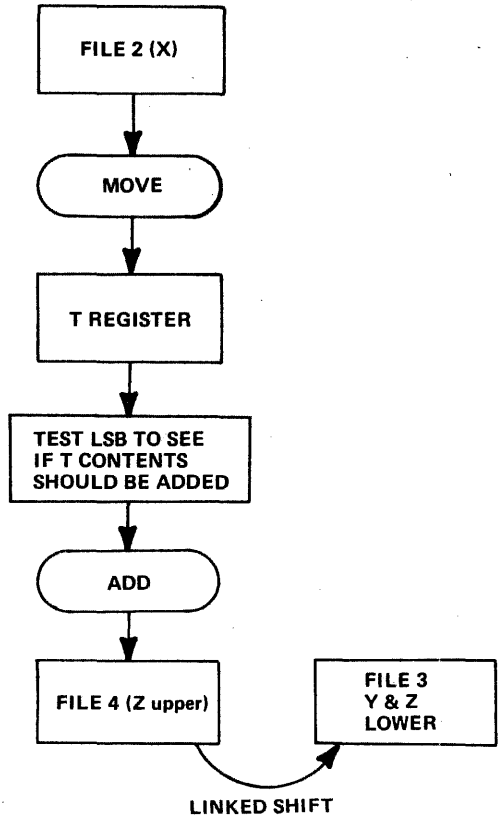
General Approach

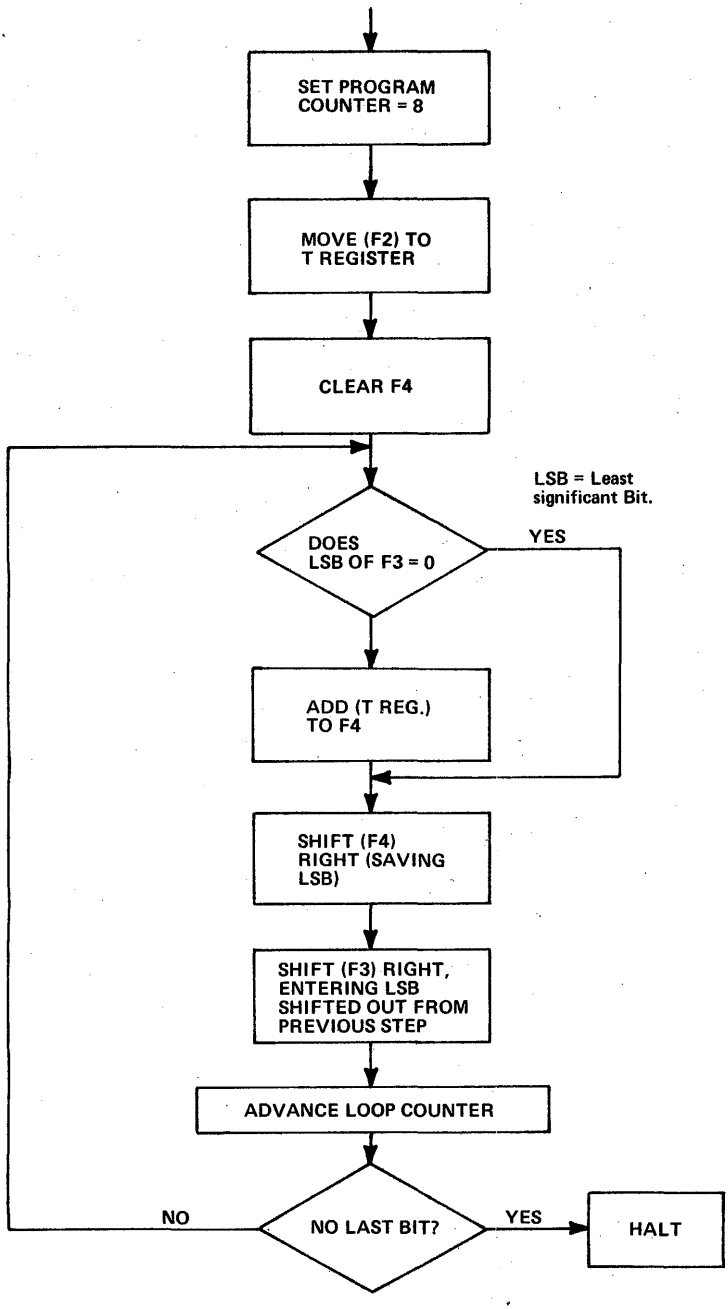
Use Add and Shift Algorithm.

File Register Assignments

F2 = X
F3 = Y, and Z Lower
F4 = Z Upper
F5 = Loop Counter

Data Flow





Functional Flow Chart for Multiply

Program for Multiply routine:

Machine Code		Assembly Language			
L	Command	Name	Operation	Operand	Comments
000	2508		LF	5, X'08'	Set Loop Ctr = 8
001	C201		MT	2	Move X to T Reg.
002	2400		LF	4, X'00'	Clear ZU
003	4301	ADD	TZ	3, X'01'	Y Bit 0 = 1
004	8420		A	4, T	Add X to Z
005	F420		H	4, R	Shift Z_U
006	F3A0		H	3, L, R	Shift Z_L
007	9550		D	5, C	Decrement Ctr
008	5004		TN	0, X'04'	Loop Ctr = 0
009	1403		JP	ADD	Jump Loop
00A	1780		LS	X '80'	Halt

For Simulator:

1. Load ROS: P000, 2508, C201, etc.
2. Data Values: Set file 2, f_3
   D2, type in X
   D3, type in Y
3. Execute: G0000 CR
4. Display results with D2, D3, D4.

BINARY BIT BY BIT EXAMPLE OF MULTIPLY

DECIMAL VALUES
X = 89
Y = 106
Z = 9434

Binary Values

									Initially Y, this ends up as Z lower								
X	0	1	0	1	1	0	0	1									
Y	0	1	1	0	1	0	1	0									
ADDO	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	← Least significant bit is tested each time to determine if X should be added or not.
1. SHIFT	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	
ADDX	0	1	0	1	1	0	0	1	0	0	1	1	0	1	0	1	
2. SHIFT	0	0	1	0	1	1	0	0	1	0	0	1	1	0	1	0	
ADDO	0	0	1	0	1	1	0	0	1	0	0	1	1	0	1	0	
3. SHIFT	0	0	0	1	0	1	1	0	0	1	0	0	1	1	0	1	
ADDX	0	1	1	0	1	1	1	0	1	0	0	1	1	0	1	1	
4. SHIFT	0	0	1	1	0	1	1	1	1	0	1	0	0	1	1	0	
ADDO	0	0	1	1	0	1	1	1	1	0	1	0	0	1	1	0	
5. SHIFT	0	0	0	1	1	0	1	1	1	1	0	1	0	0	1	1	
ADDX	0	1	1	1	0	1	0	0	1	1	0	1	0	0	1	1	
6. SHIFT	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	1	
ADDX	1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	1	
7. SHIFT	0	1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	
ADDO	0	1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	
8. SHIFT	0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	0	

FINAL RESULT = 9434

This program loops 8 times.

Execution time = 14.74 microseconds.

MICROPROGRAM EXAMPLE NO. 2

Subroutine Jumps

Return jumps to subroutines can easily be implemented in microprograms. Two examples are shown below. One is for return jumps to programs on the same page, and the other is for return jumps to another page. A page is 256 locations.

a. Return Jump to Routine on same page (or pair of pages).

```
034F   2A0A                    LF   V, 10
0350   2B52                1 (LF   W, CVB1) <-- Loading Return Address in W
0351   1D61                   (JP   CVB4)  <-- Jumping
0352   210F   (CVB1)      2  LF   OP, X '0F'
                              ----
       Jump                   ----
       Address                ----
0361   51FF   (CVB4)          TN   OP, X'FF'
0362   CB05                4 (MK   W)   <-- Return Jumping
```

(CVB1) Return Jump Address

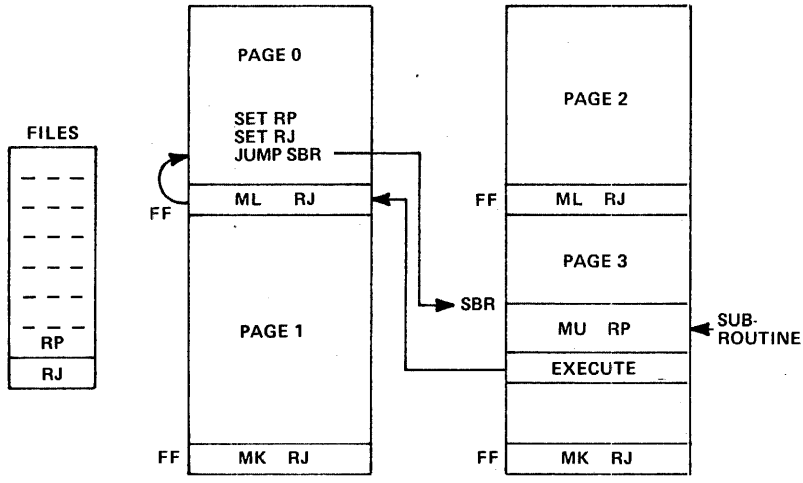
To do a return jump to the same page (or pair of pages), the address of the next command after the jump command 3 is loaded into a temporary file register, called W in this example. 1 Then the jump is made to the first command of the subroutine 2. The return jump command 4 moves the return address (stored in W for this example) into L or K. (K is simply L with the page bit set to 1.) This command causes L to jump to the programmed return jump location.

b. General Return Jump

To jump to any location in the read only memory requires an additional step besides that described in example a. It is necessary to have an additional return address for page identification. One way to mechanize a general scheme for return jumping to subroutine is to have a pointing command on each page and to use an indirect jumping technique.

This is illustrated by the following read only memory map. The indirect jump location is at the same address on each page (FF for this example).

Two files are assigned for return addresses, one contains the page, and the other the return address on the page. Both of these must be set



prior to making the jump. RP is the page pointer. If for a number of commands there is no multiple re-entry points, or multiple nesting across page boundaries, RP can be set, and left set for a number of commands.

The return jump to originating PAGE is accomplished using the execute command with the U register. Since the intermediate jump locations are all at XFF, it is only necessary to load U with the X (or page identifier) from RP. This is mechanized as follows:

RP = file E
RJ = file F

Page 0 for Jump Command

015	2E14	LF		RP, X'14'	Return Page to RP
016	2F18	LF		RJ, X'18'	Return Address
017	1D41	JP		SUB	
018		Next command after subroutine			
341	2104	SUB	LF	1, X'04'	Any command may be here
350	CE01		MU	RP	Set Page into U
351	8000		A	0	No Op delay, to use U
352	00FF		ES*	0, 15	Execute to interpret RP value as page jump command

Execute command:

0	0	F	F	Execute
1	4			in U Register
1	4	F	F	effective command

Jump to Page 0 location FF

at Page 0 location FF

0FF	CF04	ML	RJ

This loads L with return address in RJ.

MICROPROGRAM EXAMPLE NO. 3

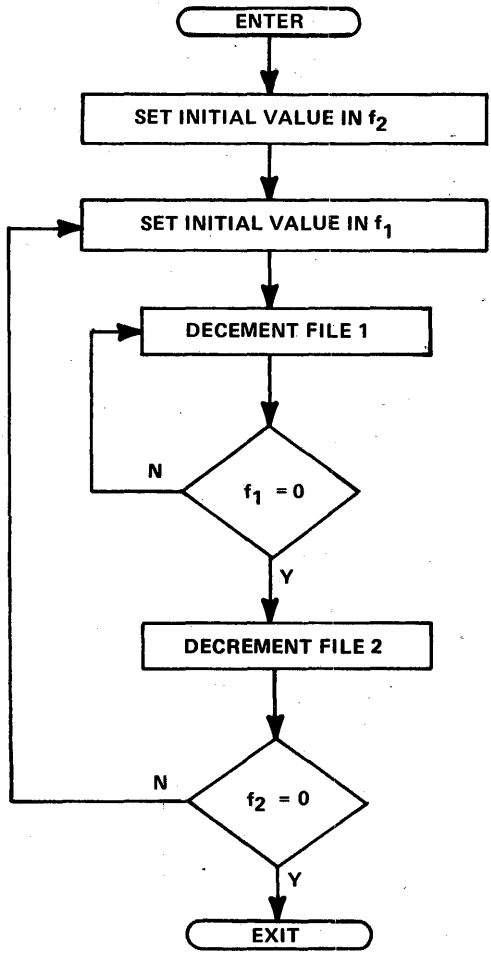
Time Delay Routine

Nested loop program to generate a time delay, such as can be used to sample serial teletype data.

Specific Considerations

- Two nested loops, with file 1 assigned to inner loop and file 2 assigned to outer loop.
- File 0, zero condition flag, is used to indicate zero count for both loops.

Functional Flow Chart:



Program for Time Delay Routine:

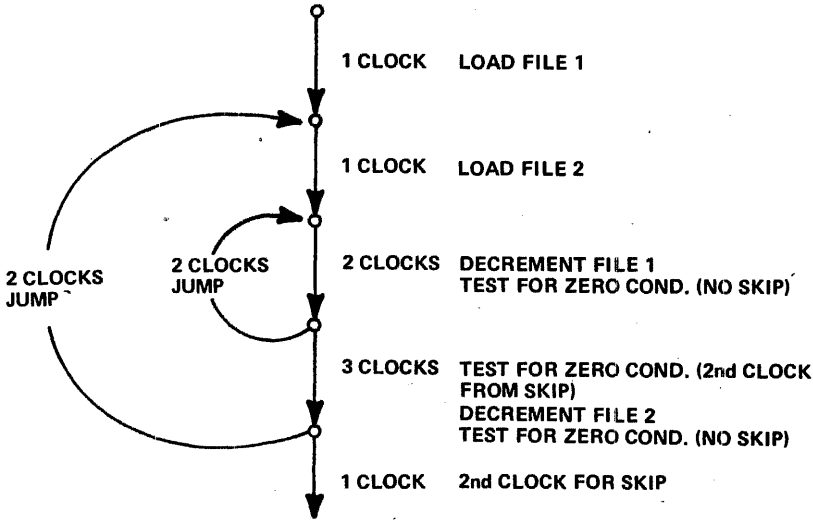
Machine Code		Assembly Language			
L	Command	Name	Operation	Operand	Comments
000	22 ②		LF	2, X'②'	Set outer loop
001	21 ①	LP2	LF	1, X'①'	Set inner loop
002	9150	LP1	D	1, C	Decrement inner loop file 1 Set C
003	5004		TN	0, X'04'	Zero count?
004	1402		JP	LP1	Jump inner loop
005	9250		D	2, C	Decrement outer loop file 2 Set C
006	5004		TN	0, X'04'	Zero count
007	1401		JP	LP2	Jump outer loop

② outer link count
① inner link count

Calculation of delay:

The delay of this routine can be calculated by preparing a flow graph with the number of clock times for each branch in the graph. The graph for this routine is as follows:

Flow Graph for Time Delay Routine:



Number of clock times, $C = 8 + 8 (m - 1) + 4m (n - 1)$

$= 4m (1 + n)$

$t = .22\ C$ microseconds $= .88m (1 + n)$

where

m = outer loop counts
n = inner loop counts

This equation is valid for $1 < m, n < 255$.

If m or n = 0, their effective value becomes 256.

Examples of clock time calculations:

m	n	C	t (microseconds)
1	1	8	1.76
1	2	12	2.64
2	1	16	3.52
2	2	24	5.28

Example of derivation of m and n:

Calculate m and n for a time delay of 20 milliseconds = 20,000 microseconds.

Solution:

$.88m (1 + n) = 20{,}000$

pick m = 20,000 = 142 decimal = '8E' hexadecimal

then $.88 \times 142 (1 + n) = 20{,}000$

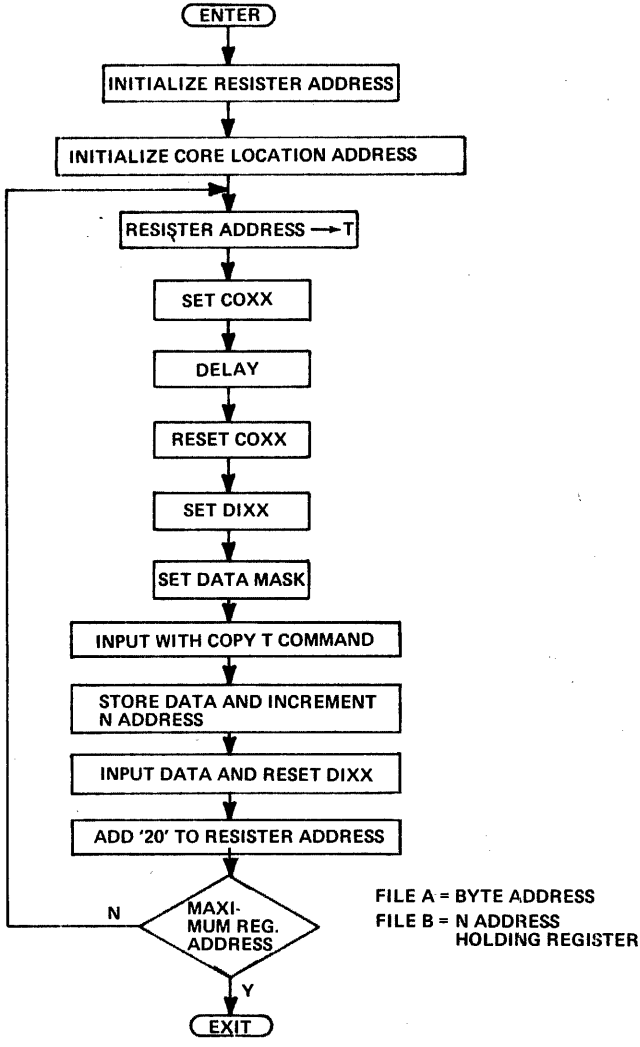
$$n = \frac{20{,}000}{.88 \times 142} - 1 = 160\text{-}1 = 159 \text{ decimal} = \text{'9F' hex.}$$

MICROPROGRAM EXAMPLE NO. 4

Data Input from 4 External Registers

Input data from 4 registers (at device '08', '28', '48', '68') to core locations '0200', '0201', '0202', '0203'.

Flow Chart:



Program for Input Date Byte Routine:

Machine Code		Assembly Language			
L	Command	Name	Operation	Operand	Comments
000	2A08		LF	10, X'08'	Set Register Address
001	1202		LM	X'02'	Set M Address register = '02'
002	2BFF		LF	11, X'FF'	Set N Address register = Int. Add. -1
003	CA01	ADD	MT	10	Register Address to T
004	7090		K	0, 9	Set COXX
005	1000		LZ	X'00'	No Op Delay*
006	7080		K	0, 8	Reset COXX
007	70E0		K	0, E	Set DIXX
008	21FF		LF	1, X'FF'	Set Data Mask
009	ABD3		WN	11, I	Update N, start a write
00A	7181		KT	1, 8	Input to T, reset DIXX
00B	3A20		AF	10, X'20'	Update register address
00C	6A80		CP	10, X'80'	Skip if (f_A) > 68
00D	1403		JP	ADD	Jump Loop
00E	Next command				

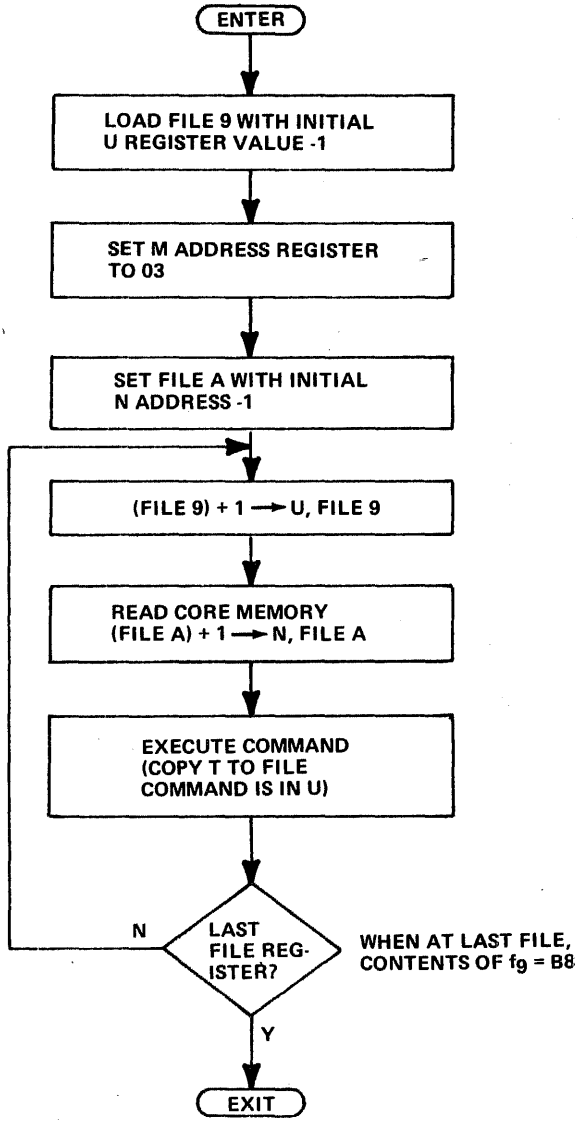
*If LZ is used for a special interface, it may not be usable as a No Op.

MICROPROGRAM EXAMPLE NO. 5

Load 8 successive file registers (f_1–f_8) from 8 successive core locations (0301–0308)

Use the execute command for loading files. The U register will be loaded with a value which has a Copy T as an Op code. Use file 9 to contain and update U register values. File 9 will also act as a loop counter. Use file A to contain and update N address register value.

Flow Chart:



Program for Loading 8 Successive Files from Core:

Machine Code		Assembly Language			
L	Command	Name	Operation	Operand	Comments
000	29B0		LF	9, X'B0'	Initial U value -1
001	1203		LM	X'03'	M address
002	2A00		LF	10, X'00'	Initial N address -1
003	8946	LP1	AU	9, I	Update file 9 and U register
004	AAC3		RN	10, I	Read memory and update N, and file 10
005	0020		E	0, 2	Copy T to file register 1 to 8 in sequence
006	6948		CP	9, X'48'	(fg) > B7
007	1403		JP	LP1	Jump Loop
008	Next command				

Effective command at 005:

Execute	0020	
U register	B1	
Effective command	B120	Copy T to file 1

MICROPROGRAM EXAMPLE NO. 6

16 Bit add (core to file)

This routine adds the contents of files A_U, A_L to a 16 bit word in core memory at the address contained in O_U, O_L and places the result in A_U, A_L.

File designations:

Temp. register S = f_1

Data in files A_U = f_4, A_L = f_5

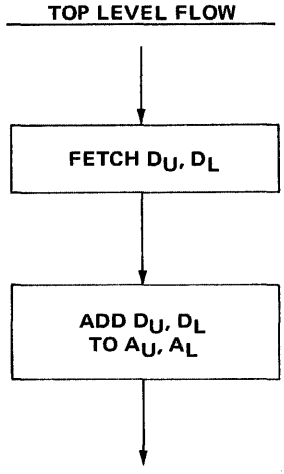
Core memory address in O_U = f_8, O_L = f_9

Result in file A_U = f_4, A_L = f_5

Memory Location:

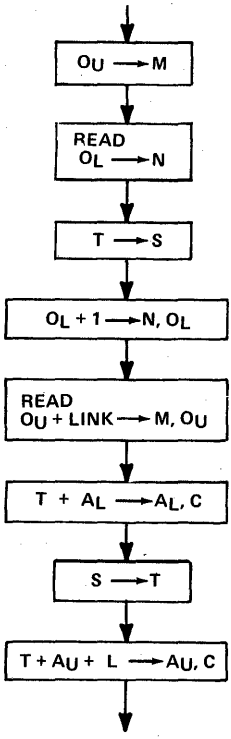
Data in D_U and D_L (successive bytes in core)

The condition flags are set by this routine to indicate negative result, overflow, or linked zero test over multiple bytes.



This routine has 8 microcommands, and takes 2.86 microseconds* to execute. There is an effective 3 clock delay after the 1st memory command, due to changing N and selecting T, and a 2 clock delay after 2nd memory command due to selecting T.

*Not including return jump.



MACHINE ADD. CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
000 C802	ADD	MM	OU	Move upper address byte to M.
001 A903		RN	OL	Read upper data byte, move lower address byte to N (data goes to T).
002 B120		C	S, T	Save upper data byte in S.
003 8943		IN	OL	Move incremented lower address byte to M.
004 A882		RM	OU, L	Read lower data byte. Move upper address byte + (Link) to M. Data goes to T.
005 8530		A	AL, T, C	Add (T) to lower byte of A, set condition flags.
006 C101		MT	S	Move lower data byte from S to T.
007 84B0		A	AU, T, L, C	Add upper data bytes + Link. Set condition flags. Linked 0 test.

MICROPROGRAM EXAMPLE NO. 7

Input a 32 bit word from an external device to core memory.

This routine causes the data in a 32-bit word to be partitioned into 4 bytes which are input to 4 consecutive core locations designated by O_U and O_L.

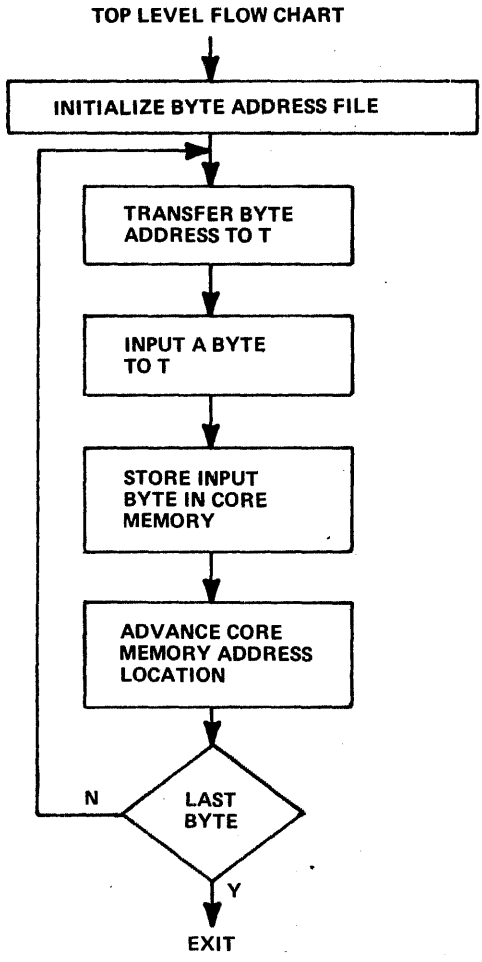
File Designations:

- Core memory address for data is in $O_U = f_8$, $O_L = f_9$.
  Byte address is in $F_B = f_B$.

Byte Addresses: 01, 21, 41, 61.

Memory Locations:

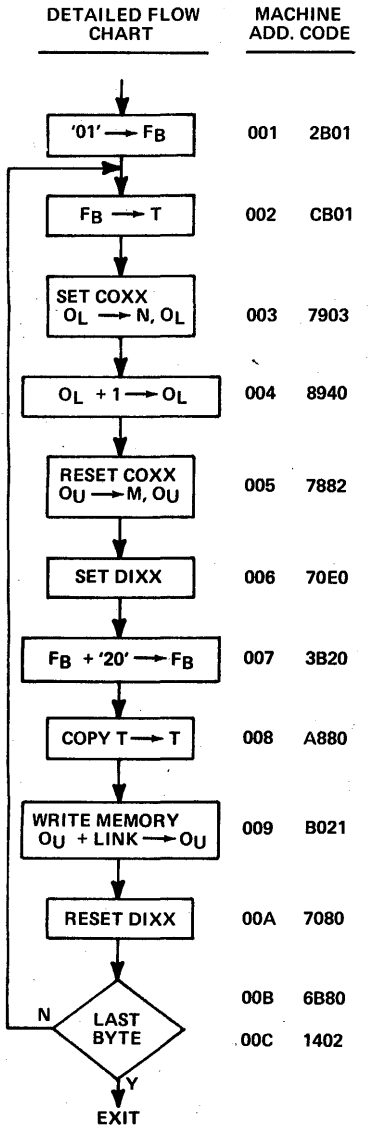
4 successive bytes starting with the 1st location in O_U, O_L.



In order to save microcommands some of the functions shown in the top level flow chart are dispersed and combined with other functions as shown in the detailed flow chart.

The write memory command is deliberately placed before the data point command in the detailed flow chart to allow memory to start prior to changing T.

This routine has 12 microcommands and takes 10.56 μs to execute, which includes all I/O and memory access timing, but does not include return jump.



DETAILED FLOW CHART	MACHINE ADD.	MACHINE CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'01' → FB	001	2B01	INP	LF	FB,X'01'	Initialize byte address file.
FB → T	002	CB01	NXT	MT	FB	Move byte address to T.
SET COXX OL → N, OL	003	7903		KN	OL, 9	Set COXX and N address register.
OL + 1 → OL	004	8940		I	OL	Delay and update OL (1)
RESET COXX OU → M, OU	005	7882		KM	OU, 8	Reset COXX and set M address register
SET DIXX	006	70E0		K	0, E	Set DIXX.
FB + '20' → FB	007	3B20		AF	FB,X'20'	Delay and advance byte address.
COPY T → T	008	A880		W	OU, L	Store input byte and update OU (2)
WRITE MEMORY OU + LINK → OU	009	B021		CT	0, T	Input byte to T.
RESET DIXX	00A	7080		K	0, 8	Reset DIXX.
N LAST BYTE	00B	6B80		CP	FB, X '80'	Test for last byte. Jump back of more bytes to be input.
	00C	1402		JP	NXT	
Y EXIT					Exit	

(1) The state of Link from this command must be saved for updating OU.

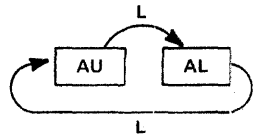
(2) Input (Link) from update of OL.

MICROPROGRAM EXAMPLE NO. 8

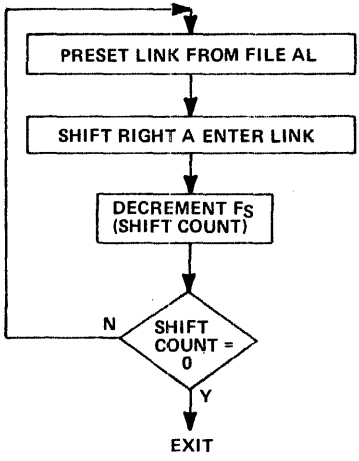
16 bit right shift with end around carry with the shift count in file register S.

File Designations:

- Data to be shifted in files A_U, A_L
- Shift count in file S.



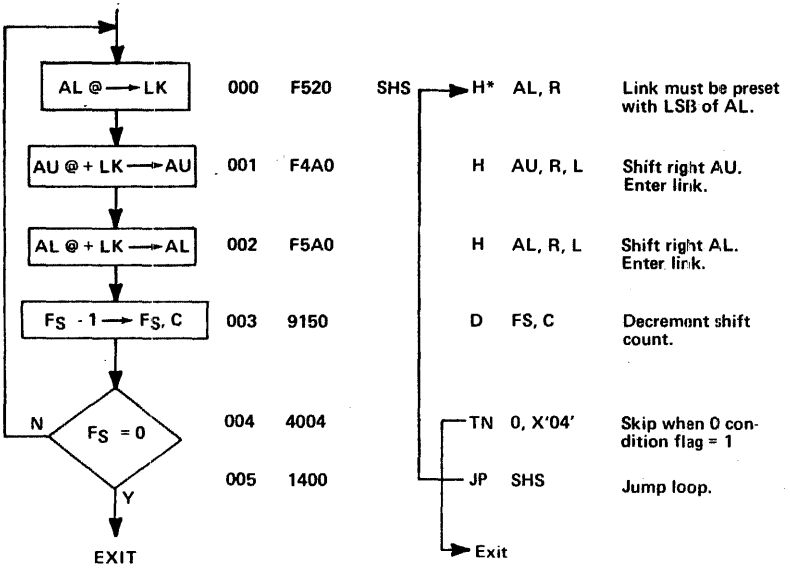
FLOW CHART



File S = f_1
A_U = f_4
A_L = f_5

This subroutine has 6 commands. The execution time is 1.54 N* microseconds, where n = number of bit positions shifted.

*Not including return jump.



The number of bytes shifted can be increased by adding one command per byte which is .22 ns/byte per loop additional time.

MICROPROGRAM EXAMPLE NO. 9

A ORed with B to A

Logic Symbol

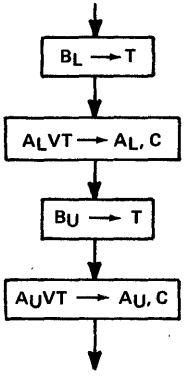
$A \vee B \longrightarrow A$

In this routine the contents of A_U and A_L is logically ORed on a bit-by-bit basis with the content of B_U and B_L. The result is placed in A_U, A_L.

File Register Designations:

Data Files $A_U = f_4$, $A_L = f_5$

Files $B_U = f_6$, $B_L = f_7$



MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
000	C701	OR	MT	BL	Move B → T
001	C530		O	AL, T, C	OR AL with T
002	C601		MT	BU	Move BU → T
003	C4B0		O	AU, T, C, L	OR AU with T

The last operand includes L to provide a linked zero test over multiple bytes.

This routine has 4 commands and takes .88 microseconds, not including return jump.

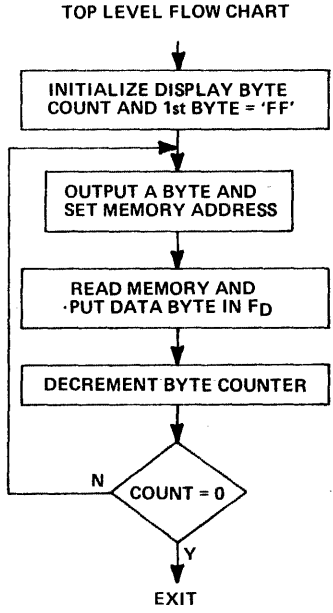
MICROPROGRAM EXAMPLE NO. 10

Update 10 BCD digit display from core.

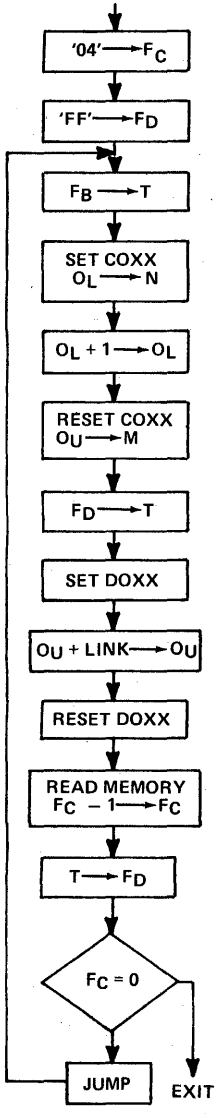
For this routine a 5-byte packed BCD image of the digital display is maintained at all times in core. This image is updated by other programs. Periodically this routine is utilized to transfer the image out to the display lamps. The routine uses the standard COXX, DOXX procedures, which output a device and function code, strobed by COXX, followed by a data value (in this case two packed BCD digits) strobed by DOXX. Two digits are updated by each output byte.

Data Characteristics:

- 2 digit packed BCD per byte in core in consecutive locations.
- Data sequenced to display one byte at a time, display logic automatically sequences through latches.
- Data sequencer enabled by 1st byte containing all 1's, and disabled by last data byte.
- Core location addresses in $O_U = f_6$, $O_L = f_7$.
- Display output byte address is in $F_B = f_B$.
- Standard I/O logic is used which automatically disconnects after each byte is transferred.
- Display byte count is in $F_C = f_C$.
- Data from memory is temporarily held in $F_D = f_D$.



This routine has 14 commands and takes 13.42 microseconds to execute.



DETAILED FLOW CHART	MACHINE ADD.	MACHINE CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'04'→FC	000	2C04	DSP	LF	FC, X'04'	Initialize byte count.
'FF'→FD	001	2DFF		LF	FD, X'FF'	Set 1st output byte = 'FF'
FB→T	002	CB01	RPT	MT	FB	Move byte address to T.
SET COXX OL→N	003	7793		KN	OL, 9	Set COXX and N register.
OL + 1→OL	004	8740		I	OL	Delay and update OL.
RESET COXX OU→M	005	7682		KM	OU, 8	Reset COXX, and set M register.
FD→T	006	CD01		MT	FD	Move output byte to T.
SET DOXX	007	70A0		K	0, 10	Set DOXX.
OU + LINK→OU	008	8680		A	OU, L	Delay and update OU.
RESET DOXX	009	7080		K	0, 8	Reset DOXX.
READ MEMORY FC − 1→FC	00A	AC40		R	FC, D	Read memory and decrement byte count.
T→FD	00B	BD20		C	FD, T	Transfer output byte, just read from core to FD.
FC = 0	00C	4C03		TZ	FC, X'03'	Test for byte count = 0.
JUMP / EXIT	00D	1402		JP	RPT	Jump loop.
			Exit			

MICROPROGRAM EXAMPLE NO. 11

Clear a block of core memory.

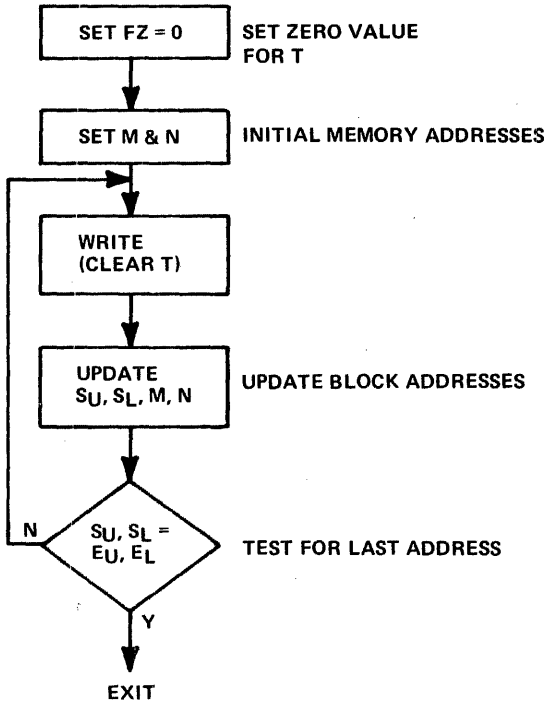
This routine causes a selected block of core memory to be set to all zeros.

File Register Designations:

Starting of current address $S_U = f_8$, $S_L = f_9$

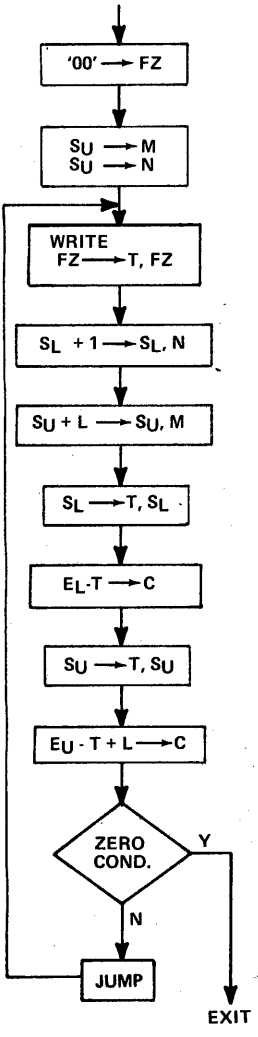
Ending address $E_U = f_A$, $E_L = f_B$

Zero value in FZ = f_1



On a write memory command, data in T is stored in the memory location set by M and N.

This routine has 12 commands. It takes 3.52 microseconds to clear the first byte, plus 3.08 microseconds for each additional byte. Clearing a fixed length block in one page takes only 1.1 μs per additional byte.



DETAILED FLOW CHART	MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'00' → FZ	000	2100	CLR	LF	FZ, X'00'	Set zero value for T.
SU → M	001	C802		MM	SU	Initial value to M.
SU → N	002	C903		MN	SL	Initial value to N.
WRITE FZ → T, FZ	003	A111	NXT	WT	FZ	Write zero into core.
SL + 1 → SL, N	004	8943		IN	SL	Increment 16 bit memory address.
SU + L → SU, M	005	8882		AM	SU, L	
SL → T, SL	006	C901		MT	SL	Subtract
EL-T → C	007	9B38		S*	EL, T,C	SL from EL
SU → T, SU	008	C801		MT	SU	Subtract
EU - T + L → C	009	9AB8		S*	EU, T,L,C	SU from EU
ZERO COND.	00A	5004		TN	0,X'04'	Last byte cleared.
JUMP	00B	1403		JP	NXT	Jump loop.

MICROPROGRAM EXAMPLE NO. 12A

Read 8 consecutive core locations into 8 consecutive file registers.

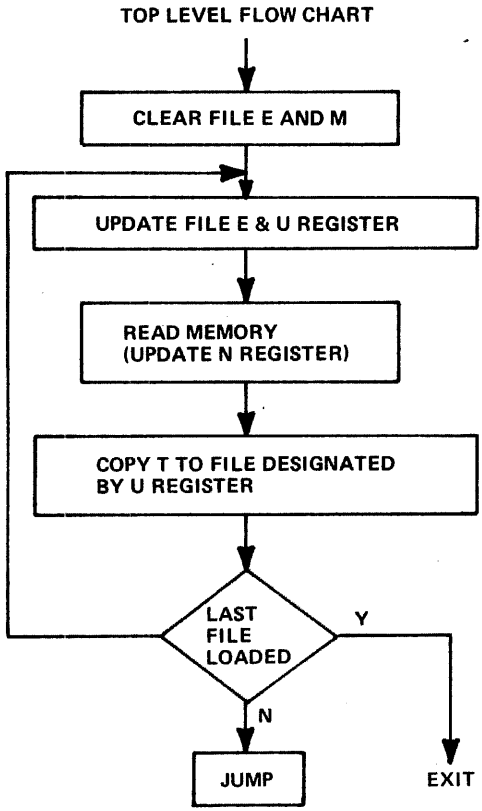
This routine is used to move a block of data from core to the files.

File Designations:

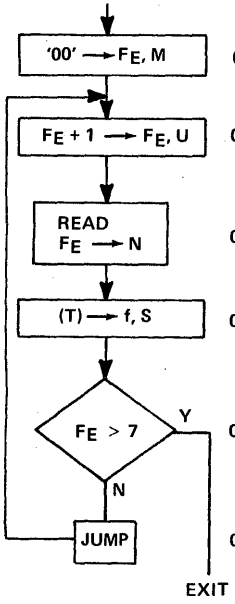
Files 1-8	to receive data
File E	Memory address and file index. U register is used to index through the files.

Dedicated Core Locations:

All on page 0, with N = 01, 02, 03, 04. . . . . 08.



6 commands are required. Execution time is 14.08 μs.



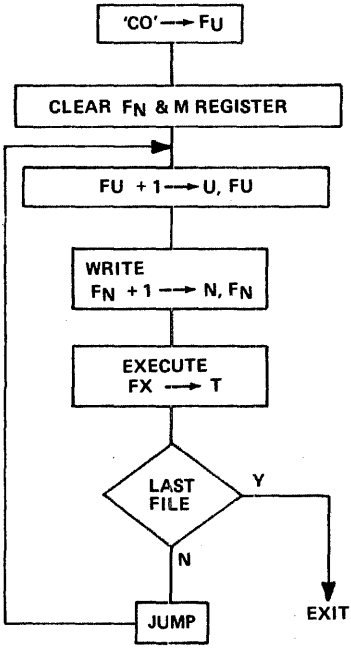
DETAILED FLOW CHART	MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'00' → FE, M	000	BE02	LFR	CM	14	Copy 0 to FE and M to clear both.
FE + 1 → FE, U	001	8E46	NXT	IU	14	Increment FE and put result in U and FE.
READ FE → N	002	AE03		RN	14	Read memory. Update N with (FE).
(T) → f, S	003	B027		CS	0, T	Copy T to file designated by (U).
FE > 7	004	6EF8		CP	14,X'F8'	Test for last file.
JUMP	005	1401		JP	NXT	Jump loop.

MICROPROGRAM EXAMPLE NO. 12B

Write 8 consecutive files into 8 consecutive core locations.

This routine is similar to 7a except for use of a write command and a move to T command, which requires the execute command to have T as a destination. File U (f_E) contains the Op code for a move, so it can't be used for the memory address if N = 01, 02, etc.

7 commands are required. Execution time is 10.78 μs.



DETAILED FLOW CHART	MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER.	OPERAND
	000	2EC0	STM	LF	FU, X 'CO'
	001	BB02		CM	FN
	002	8E46	NXT	IU	FU
	003	ABD3		WN	FN, I
	004	0001		ET	0, 0
	005	6E38		CP	FU, X'38'
	006	1402		JP	NXT

MICROPROGRAM EXAMPLE NO. 13A

Output from 8 files to 8 shift registers.

8a. File to register bit order the same.

This routine provides the microprogramming for utilization of the minimum number of logic chips to get 64 lines out from the computer. These lines can be used to drive displays, printers, etc.

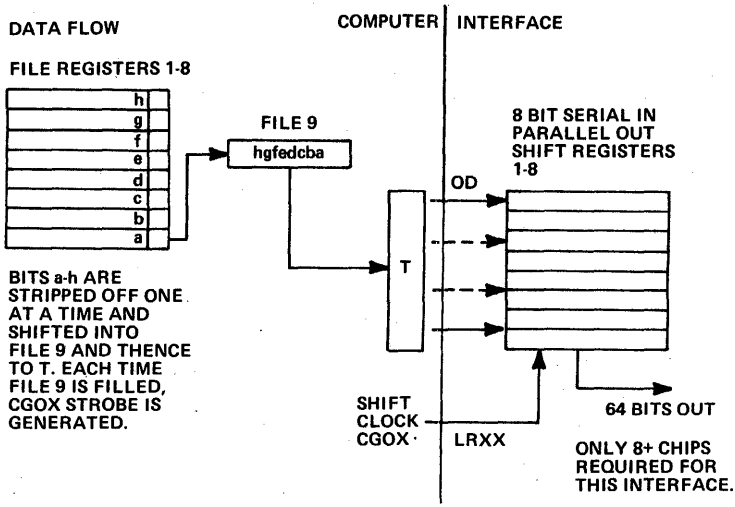
This routine is used where the order of bits shifted out is important or where the number of output shift registers is less than 8 so there is no symmetry.

The next Example (8b) shows much simpler coding to interface with 8 shift registers without pattern rotation.

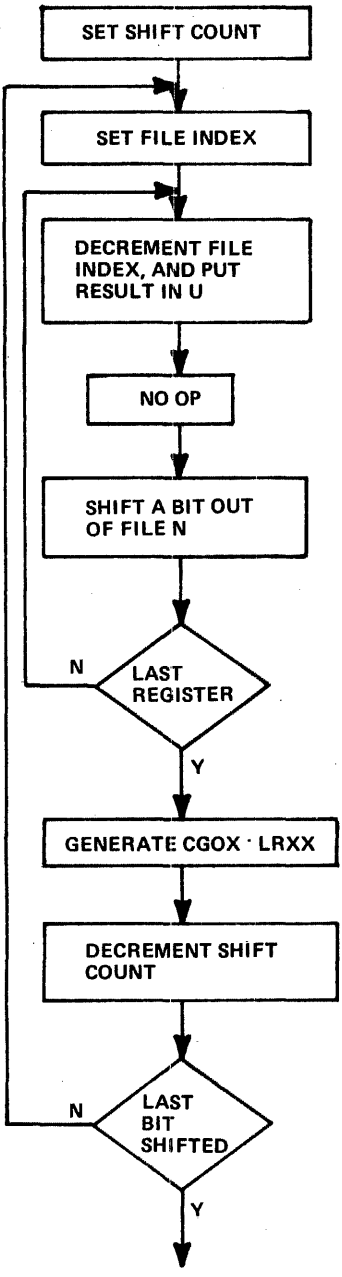
File Allocations:

Files 1-8	Data
File 9	Shift assembly register
File E	File index register
File F	Shift count register

Since this is a minimum hardware interface, the load zero command (CGOX) will be used to strobe the data directly out of T.



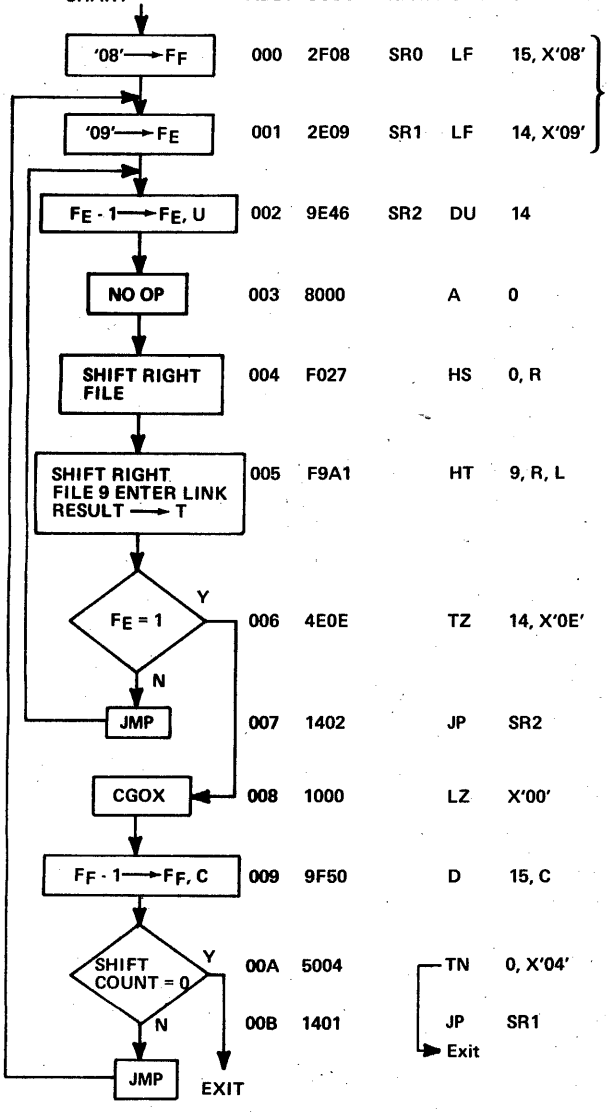
TOP LEVEL FLOW CHART



This routine has 12 commands.

It takes 107.36 microseconds to execute this routine.

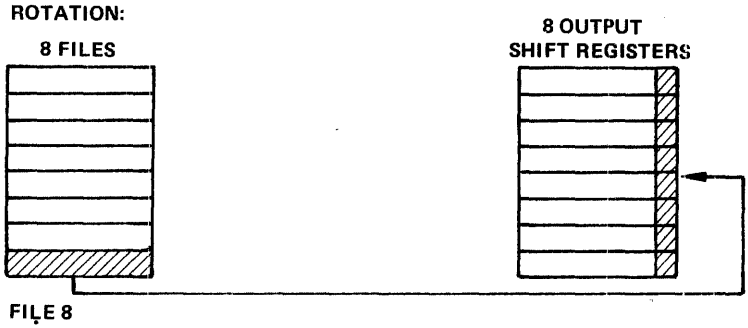
This routine used in conjunction with routine 7 for loading core to files requires 19 commands total, and 118.14 microseconds to output 8 core locations to 8 output bytes with an 8-chip interface.


DETAILED FLOW CHART
MACHINE ADD. CODE
ASSEMBLY LANGUAGE NAME OPER OPERAND
COMMENTS
'08' → FF
000 2F08 SR0 LF 15, X'08'
'09' → FE
001 2E09 SR1 LF 14, X'09'
Set shift count and file index.
FE - 1 → FE, U
002 9E46 SR2 DU 14
Decrement index → U
NO OP
003 8000 A 0
Simple No Op (Add to file 0)
SHIFT RIGHT FILE
004 F027 HS 0, R
Shift right file selected by index LSB to link.
SHIFT RIGHT FILE 9 ENTER LINK RESULT → T
005 F9A1 HT 9, R, L
Shift file 9 right, enter link, result to T.
FE = 1
Y
N
006 4E0E TZ 14, X'0E'
Output byte assembled.
JMP
007 1402 JP SR2
Jump back to byte assembly.
CGOX
008 1000 LZ X'00'
Generate CGOX-LRXX strobe.
FF - 1 → FF, C
009 9F50 D 15, C
Decrement shift count.
SHIFT COUNT = 0
Y
N
00A 5004 TN 0, X'04'
Shift count = 0.
00B 1401 JP SR1
Jump back to next bit shift.
Exit
JMP
EXIT

MICROPROGRAM EXAMPLE NO. 13B

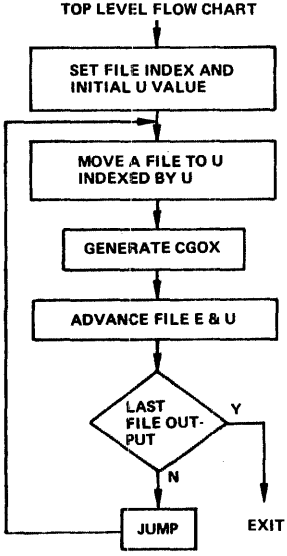
File to register; with hardware rotation of bit pattern.

In most cases, such as for updating digital displays, etc., it doesn't matter if the pattern in the 8 file registers is "rotated" with respect to the 8 output shift registers. In the example below, file 8 becomes disassembled into 1 bit in each of the 8 output shift registers. By changing the connection of wires to the display, the effective rotation can be cancelled. By allowing for rotation, the microprogram becomes much simpler than the example in 8a.

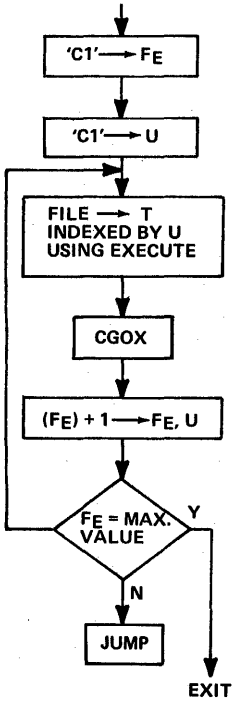


File Register Designations:

f_1 - f_8	output data
f_E	file index



This routine requires 7 instructions, and takes 10.78 microseconds to execute. So there is a tremendous time savings over the 8a example which requires pattern rotation by the microprogram.



DETAILED FLOW CHART	MACHINE ADD.	MACHINE CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'C1' → FE	000	16C1		LU	X'C1'	Initial index value to U*.
'C1' → U	001	2EC1	OUT	LF	14, X'C1'	Initial index value to file E.
FILE → T INDEXED BY U USING EXECUTE	002	0001	NXT	ET	0, 0	Move file selected by index to T which is the output bus.
CGOX	003	1000		LZ	X'00'	Generate COGX with load zero command.
(FE) + 1 → FE, U	004	8E46		IU	14	Advance index file and U.
FE = MAX. VALUE	005	6E37		CP	14, X'37'	Test for last file output and skip if complete.
JUMP	006	1402		JP	NXT	Jump loop.

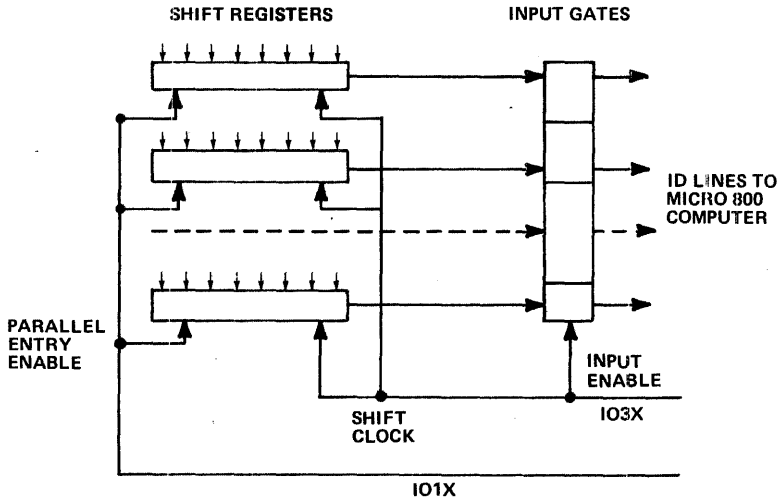
*In this routine F_E and U are updated after the execute command to avoid an extra delay which is required after updating U. In this case the delay is accomplished by the test and jump instruction.

MICROPROGRAM EXAMPLE NO. 14

Input from 8 shift registers to 8 files in MICRO 800.

This routine is somewhat similar to routine 13B except that data is input. The shift registers in the interface are parallel in, serial out.

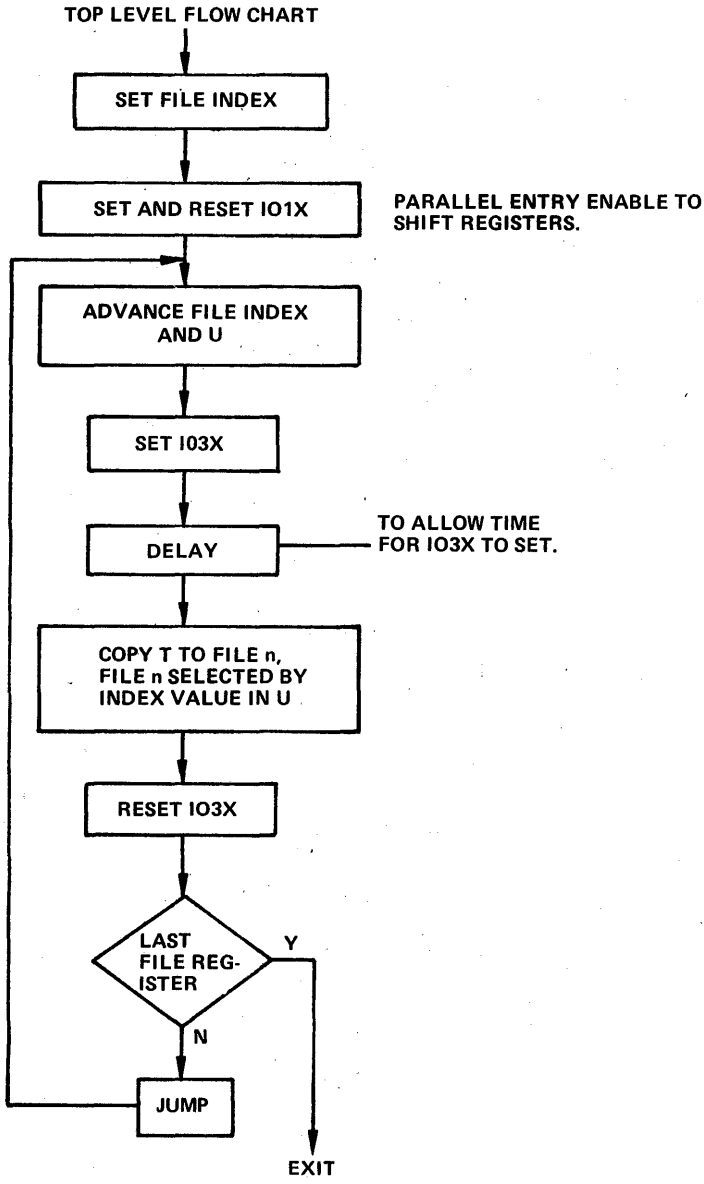
Interface Block Diagram:



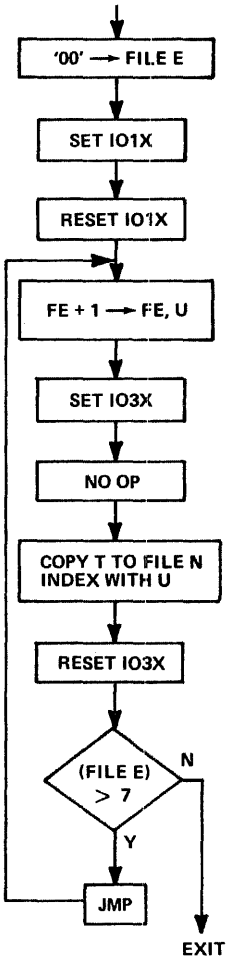
File Register Designations:

file 1 - file 8 data file registers

file E file index



This routine has 10 instructions and takes 14.52 microseconds to execute.



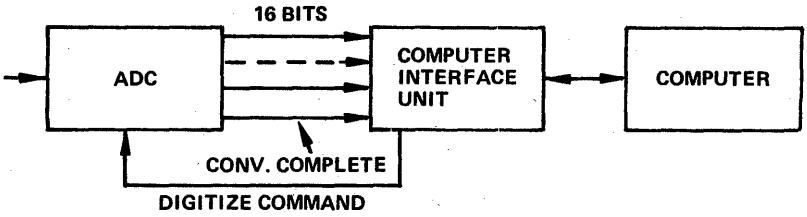
DETAILED FLOW CHART	MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'00' → FILE E	000	2E00	IPT	LF	14, X'00'	Initialize index file.
SET IO1X	001	7090		K	0, 9	Generate parallel entry enable strobe.
RESET IO1X	002	7080		K	0, 8	
FE + 1 → FE, U	003	8E46	NXT	IU	14	Update file index and U.
SET IO3X	004	70C0		K	0, 12	Set input enable flip flop.
NO OP	005	8000		A	0	Convenient No Op for time delay.
COPY T TO FILE N INDEX WITH U	006	B027		CS	0, T	Copy T → f_n indexed.
RESET IO3X	007	7080		K	0, 8	Reset input enable flip flop.
(FILE E) > 7	008	6EF8		CP	14, X'F8'	Are 8 files loaded.
JMP	009	1403		JP	NXT	No, Jump loop.

MICROPROGRAM EXAMPLE NO. 15

Input block of data to core from A to D converter.

This routine shows a method for inputting a series of 16-bit data words from an ADC. The sample rate is controlled by the read time clock option. The data words are placed in consecutive core locations. A software flag is set when the sample data block is complete.

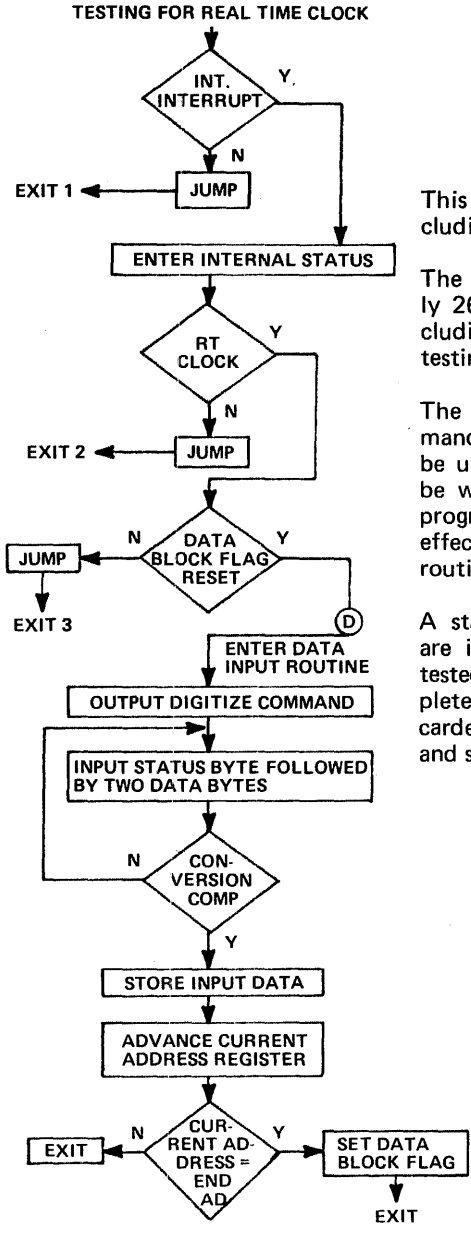
Block Diagram:



File Register Designations:

$S_U = f_4$, $S_L = f_5$	Starting (or current), address in data block.
$E_U = f_6$, $E_L = f_7$	End address in data block.
$F_F = f_F$	Bit 0 software flag.
$F_E = f_E$	Input routine file index.
$D_U = f_2$, $D_L = f_3$	Temporary files for input data.
$F_S = f_1$	Input status file.
$F_B = f_B$	Byte address file.
'FF' and COXX =	Digitize Command.

The microprogram tests the input status byte for conversion complete before inputting data.

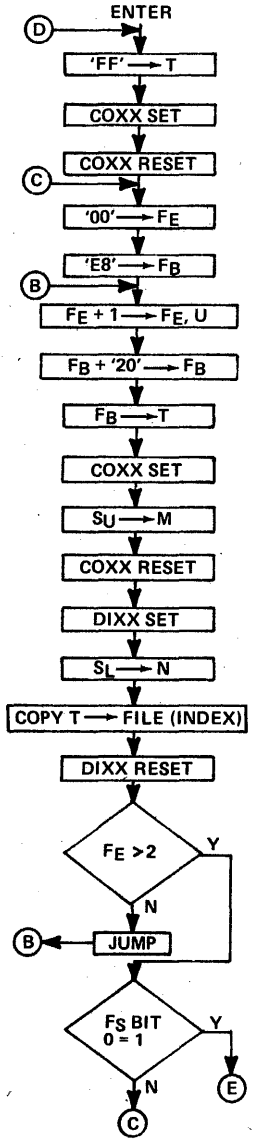


This routine has 40 commands including the real time clock test.

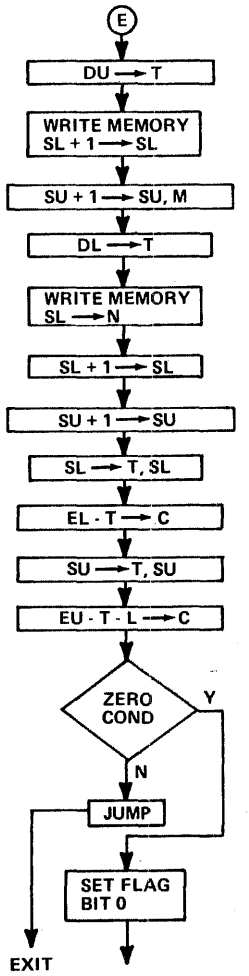
The execution time is approximately 26 microseconds per sample, including time for conversion, and testing real time clock.

The time delay from digitize command to conversion complete could be used for housekeeping if it can be worked in at that time in the program. This would result in an effective time reduction for this routine.

A status byte and two data bytes are input and then status byte is tested. If conversion is not complete, the two input bytes are discarded, and another sample of data and status is taken.



DETAILED FLOW CHART	MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'FF' → T	000	11FF	ADC	LT	X'FF'	Load T with digitize command function.
COXX SET	001	7090		K	0, 9	Digitize command strobe.
COXX RESET	002	7080		K	0, 8	
'00' → FE	003	2E00	EN1	LF	FE, X'00'	Initialize file index.
'E8' → FB	004	2BE8		LF	FB, X'E8'	Initialize byte address.
FE + 1 → FE, U	005	8E46	EN2	IU	FE	Increment file index.
FB + '20' → FB	006	3B20		AF	FB, X'20'	Advance byte address.
FB → T	007	CB01		MT	FB	Byte address to T.
COXX SET	008	7090		K	0, 9	COXX set.
SU → M	009	C402		MM	SU	Delay, and set M.
COXX RESET	00A	7080		K	0, 8	COXX reset.
DIXX SET	00B	70E0		K	0, 14	DIXX set.
SL → N	00C	C503		MN	SL	Delay, and set N
COPY T → FILE (INDEX)	00D	B027		CS	0, T	Input data byte.
DIXX RESET	00E	7080		K	0, 8	DIXX reset.
FE >2	00F	6EFD		CP	FE, X'FD'	Next byte input.
JUMP	010	1405		JP	EN2	More bytes to input.
FS BIT 0 = 1	011	5101		TN	FS, X'01'	Conversion complete.
	012	1403		JP	EN1	Take another sample of status and data bytes.



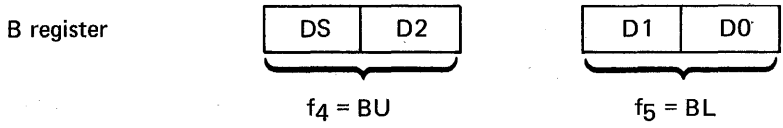
MACHINE ADD.	CODE	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
013	C201		MT	DU	Move most significant byte to T for storage.
014	A5D0		W	SL, I	Store most significant byte and increment SL.
015	8482		AM	SU, L	Update SU, M.
016	C301		MT	DL	Store least significant byte, update N.
017	A513		WN	SL	Store least significant byte, update N.
018	8540		I	SL	Increment SL.
019	8482		AM	SU, L	Update SU, M (M for access delay)
01A	C501		MT	SL	Compare SL, SU to EL, EU to see if input block is complete. This is a linked zero test over multiple bytes.
01B	9738		S*	EL, T, C	
01C	C401		MT	SU	
01D	96B8		S*	EU, T, C, L	
01E	5004		TN	0, X'04'	Test zero condition flag for end of block.
01F	1422		JP	EXIT	Continue to input and store data on next real time clock.
020	3101		AF	FF, X'01'	Set block complete flag bit.

Notice in this routine that after the two write commands, M is deliberately made the destination register of a command, to generate a delay prior to modifying T.

MICROPROGRAM EXAMPLE NO. 16

Conversion of 3 digit BCD plus sign into Binary.

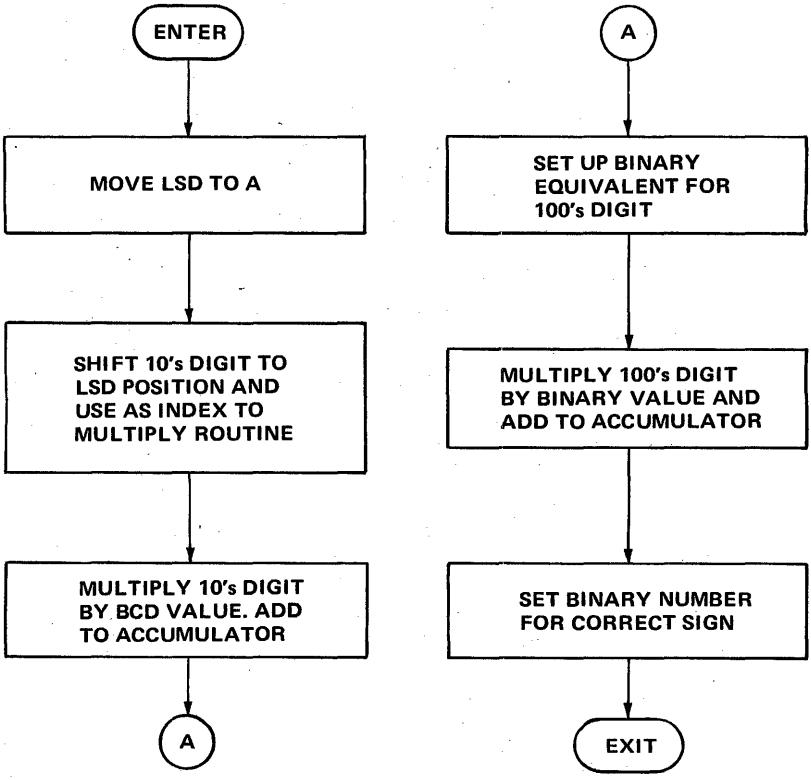
Given 3 digits in the registers BU and BL. Binary result will be in AU and AL.



other files used:

Op	= file 1	Digit value
V	= file A	Power of 10 Binary
W	= file B	Return Address

The basic technique is to multiply each BCD digit by its power of 10 expressed in binary, and to add each converted digital value in an accumulator. The top level flow is as follows:



BCD to Binary Program:

Name	Operation	Operand	Comments
CB	LF	AL,X'0F'	Set Mask for lower BCD Digit.
	MT	BL	Move Lower 2 Digits to T.
	N	AL, T	Mask, select lower Digit of AL.
	C	AU	Clear A upper.
	C	OP, T	Copy lower 2 digits from T to Op.
	K	OP, 2	Shift OP right 4, move 2nd digit to LSD.
	LT	X'0F'	Load Mask in T.
	N	OP, T	Mask out all but 2nd digit.
	LF	V, 10	Put Binary value for 10 in V.
	LF	W, CB1	Load Return Address into W.
	JP	CB4	Jump to Multiply Routine.
CB1	LF	OP,X'0F'	Set Mask in Op for 100's digit.
	MT	BU	Move 100's digit to T.
	N	OP, T	Mask out all but 100's digit.
	LF	V, 100	Put Binary value for 100 in V.
	LF	W, CB2	Load Return Address into W.
	JP	CB4	Jump to Multiply Routine.
CB2	TN	BU,X'80'	Test for Sign bit in B.
	JP	CB3	Exit if Positive Sign.
	X	AL, T, F	Ones Complement AL.
	I	AL	Add one for 2's complement.
	X	AU, T, F	Ones complement AU.
	A	AU, L	Add carry for 2's complement.
CB3	MU	RP	Set up Page Jump.
	A	0	No OP after changing U.
	ES*	0, 15	Execute implements gen. Page Jump.
*		MULTIPLY ROUTINE	
CB4	TN	OP,X'FF'	Test to see if Op has reached 0.
	MK	MK W	Return from Multiply Routine.
	MT	MT V	Move power of 10 binary to T.
	A	AL, T	Add power of 10 to accumulator.
	A	AU, L	Add carry to AU.
	D	OP	Decrement Op.
	JP	CB4	Jump Loop until Multiply over.

The multiply routine selected for this example (at CB4) is designed for minimum commands rather than minimum execute time. The multiply routine execution time is dependent on the size of the digit being converted.

The BCD digit is put into one register, and the power of 10 in another register. The BCD digit is decremented once each time the binary value for the power of 10 is added to the accumulator. When the digit is decremented to 0, the loop is exited. The average number of times through the loop per digit is 4. This is 35 clock times or about 7 microseconds.

The total average conversion time for 3 digit BCD numbers to binary is about 22 microseconds.

MICROPROGRAM EXAMPLE NO. 17

Binary to BCD Conversion.

Convert a positive binary number with a value equal to or less than 999 (decimal) into a 3-digit packed BCD integer.

Conversion Algorithm: Binary number will be successively divided by powers of 10 (starting with 100) with quotient equal to BCD value, and remainder to be divided by next lower power of 10.

$$\frac{\text{Initial Binary Number}}{100} = Q_1 \qquad R_1$$

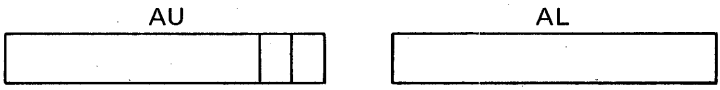
Q_1: 100's digit

$$\frac{R_1}{10} = Q_2 + R_2$$

Q_2: 10's digit; R_2: 1's digit

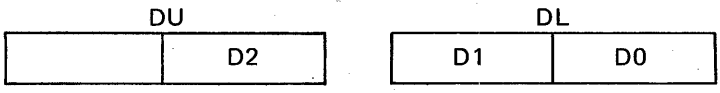
File Register Assignments:

1. Binary number is initially in A_U and A_L.



2 upper bits — 8 lower bits

2. BCD result is in D_U and D_L.



most significant digit — middle digit — least significant digit

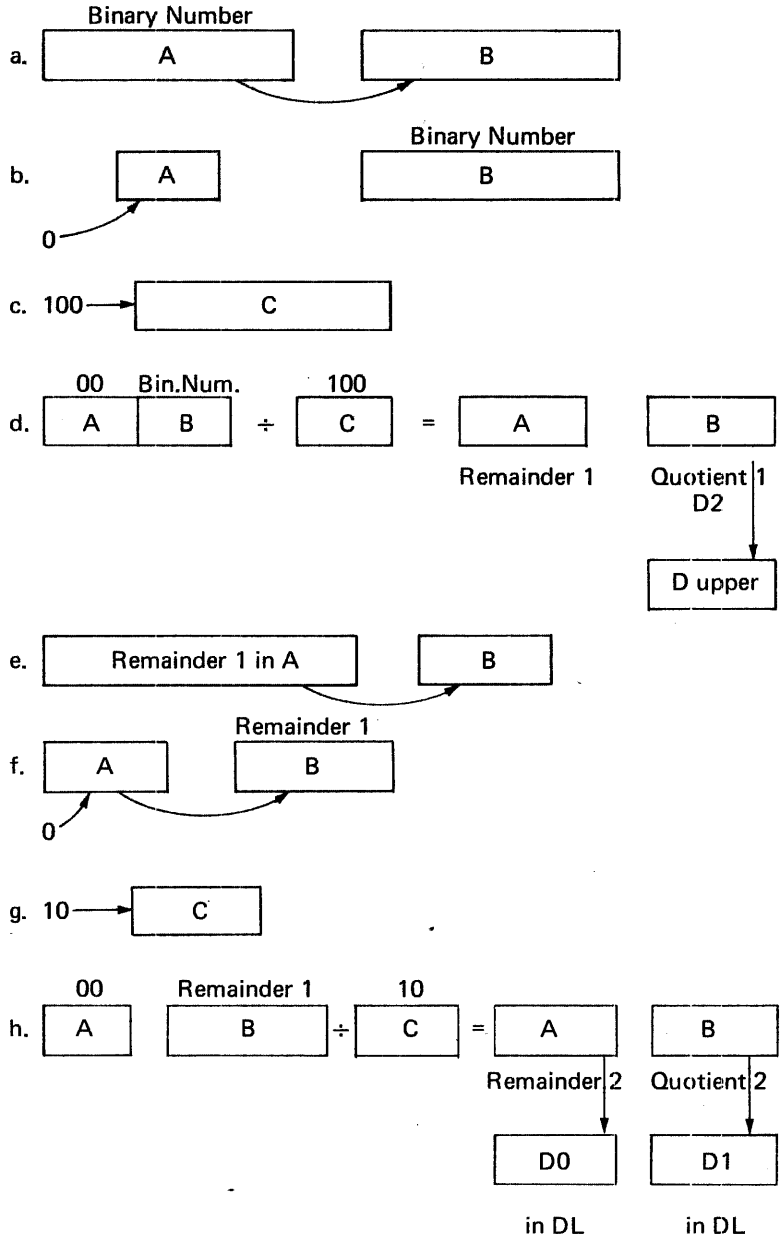
3. A_U, A_L, B_U, B_L, C_U, C_L are used for dividing registers as follows:

   a. A and B are an extended accumulator containing the dividend, C contains the divisor.

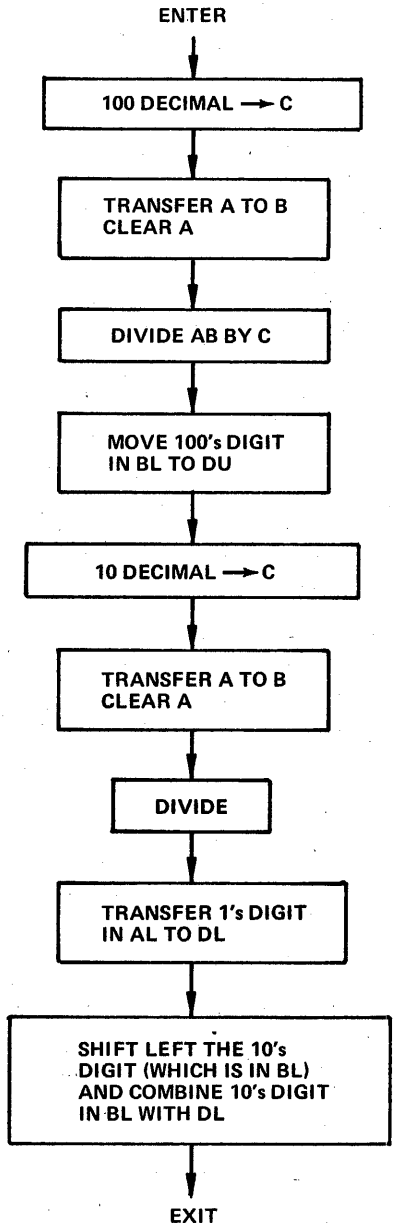
   b. After the divide, the quotient is in B, and the remainder is in A.

   c. Prior to the divide, the content of A is moved to B, and A is cleared.

4. The flow of data through the registers is as follows:

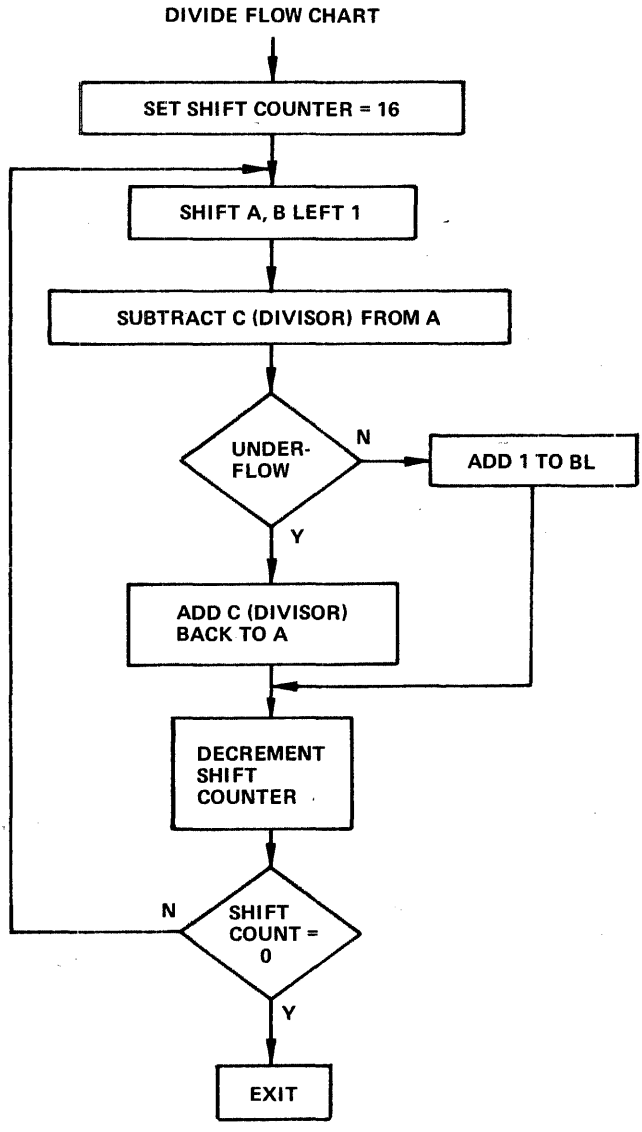


Binary to BCD conversion routine flow chart:



This routine (including the two divides), takes 47 commands, and approximately 150 microseconds to execute.

The divide routine used for this example is for positive binary integers only. It is implemented with a shift and subtract algorithm.



This divide algorithm will actually handle larger numbers than occurring in this example but is the simplest routine from a command count standpoint. For numbers the size used in this example, the divide operation could be speeded up by shifting right 6 times before starting to subtract the divisor.

Assembly Language Program to

Convert Positive Binary, 10 Bit Integer in A to 3 Digit Packed BCD Integer in D.

Uses simplified Divide Routine.

Name	Operation	Operand	Comments
CV	LF	CU, 0	Clear C upper.
	LF	CL, 100	100's coefficient to CL.
	LF	W, CV1	Set return address.
	JP	CV3	Jump to divide set up routine.
CV1	MT	BL	Move most significant digit to DU.
	C	DU, T	
	LF	CL, 10	10's coefficient to CL.
	LF	W, CV2	Set return address.
	JP	CV3	Jump to divide set up routine.
CV2	MT	AL	Move least significant digit to DL.
	C	DL, T	
	H	BL	Shift the 10's digit left one digit position.
	H	BL	
	H	BL	
	HT	BL	
	0	DL, T	Move middle digit to DL.
	MK	Y	Return.
CV3	MT	AL	Move (A) to B.
	C	BL, T	
	MT	AU	
	C	BU, T	
	C	AL	Clear A.
	C	AU	
	LF	RJ, CV4	Set return address.
	JP	DV	Jump to divide routine.
CV4	MK	W	Return to binary to BCD.

The calling sequence for this routine is

```
LF      Y, RET
JP      CV
```

Divide routine is on the same page as conversion routine.

Assembly Language Program for Divide Routine

Divide $\frac{AB}{C}$ Quotient in B, Remainder in A

Name	Operation	Operand	Comments
DV	LF	V,X'10'	Set shift counter = 16 decimal.
DV1	H	BL	Shift left 1.
	H	BU, L	
	H	AL, L	
	H	AU, L	
	MT	CL	Subtract divisor.
	S	AL, T, C	
	MT	CU	
	S	AU, T, L, C	
	TN	0,X'02'	Test for Underflow.
	JP	DV2	
	MT	CL	Add C to A.
	A	AL, T, C	
	MT	CU	
	A	AU, T, L, C	
DV3	D	V	Decrement shift CTR.
	TZ	V,X'FF'	Test for zero count.
	JP	DV1	Repeat loop.
	MK	RJ	Return.
DV2	I	BL	Add 1 bit to BL.
	JP	DV3	Jump to decrement shift counter.

MICROPROGRAM EXAMPLE NO. 18

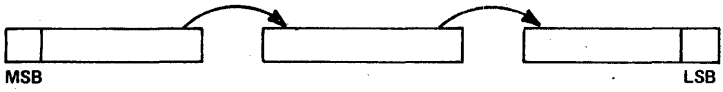
General purpose multiple file shift routine.

This routine provides a general purpose capability for shifting a group of contiguous file registers with a number of variations as indicated below.

The following items are program variable:

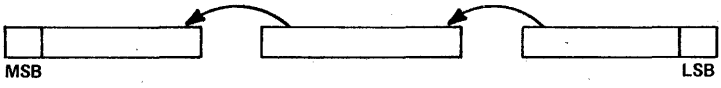
- Number of bytes 1-8, always starting with file 1.
- Number of positions shifted 1 to 256.
- Direction left or right.
- Enter one of following into vacated bit: 0, 1, LSB, MSB; which provides the capability for arithmetic or logic shifts with sign extension, end around carry, clearing, or setting to 1's.

RIGHT SHIFT



For a right shift, entering MSB causes sign extension and LSB causes end around carry.

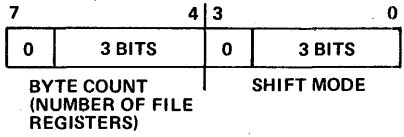
LEFT SHIFT:



For a left shift, entering MSB causes end around carry, while LSB causes odd/even extension.

File Register Designations:

File 1-8 Shift registers as selected by the instruction.
File 9 Byte count, and shift mode.

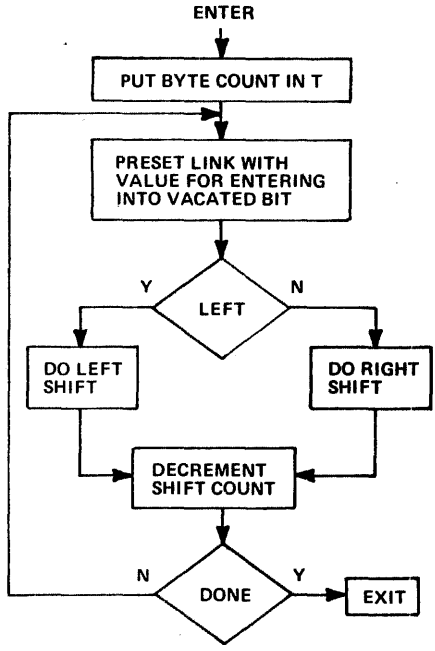


Shift Mode	Direction	Enter into vacated bit
000	L	enter 0
001	L	enter 1
010	L	enter LSB
011	L	enter MSB
100	R	enter 0
101	R	enter 1
110	R	enter LSB
111	R	enter MSB

File A Shift count

File B File index (f_U)

TOP LEVEL FLOW CHART



Presetting Link

Link is preset by one of the following:

1. Shifting right file 9 to preset link with 0 of 1.
2. Shifting left file 1 to preset link with MSB.
3. Shifting right the highest numbered file of the shift register to preset link with LSB.

In all cases, inhibit file write is used to preserve the value in the file.

For the actual right or left shift, the execute command is used, with the file register number in U.

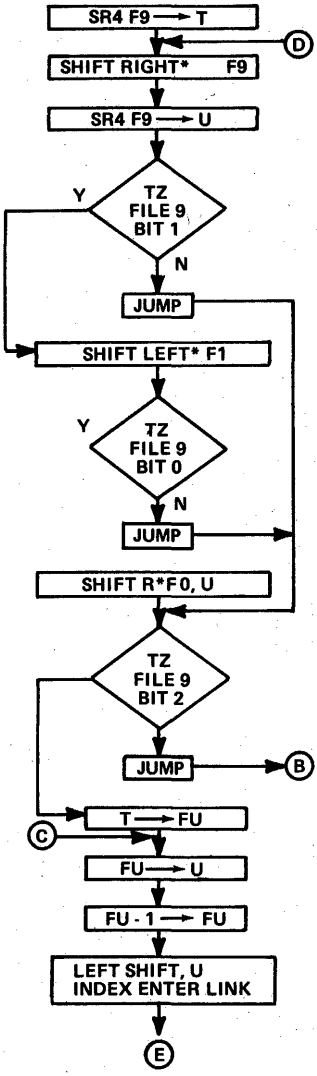
The byte count in file 9 is shifted right 4 and placed in T and U at the beginning of the program. The all 1's left in the upper 4 bits can be left there because they conveniently form the Op code for shift. T is used to hold the maximum file register number for reference purposes.

Since link is used extensively for holding shifted out bits for the next shift command, special care was taken in preparing the program to avoid commands other than the shift commands which affect link.

This routine has 29 commands.

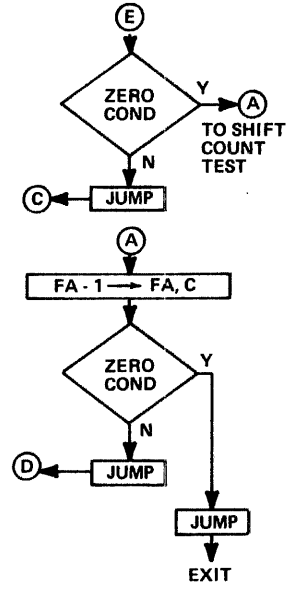
The execution time is approximately

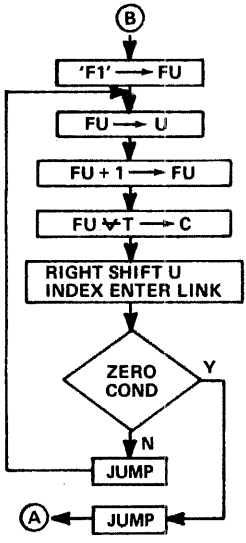
[5.94 + 1.32 x (byte count)] x (bit count) microseconds
For example 1 8 bytes, 4 bits
Time = 66 microseconds
For example 2 2 bytes, 1 bit
Time = 8.58 microseconds.



DETAILED FLOW CHART	NAME	OPER	OPERAND	COMMENTS
	SR	KT*	9, 2	Set byte count in T.
	SR1	H*	9, R	Preset link with 1 or 0.
		KU*	9, 2	Set byte count and shift instruction in U.
		TZ	9, X'02'	Test for link to be preset or constant.
		JP	SR2	Jump to shift routine.
		H*	1	Preset MSB
		TZ	9, X'01'	MSB or LSB.
		JP	SR2	Jump to shift routine.
		E*	0, R	Preset LSB.
	SR2	TZ	9, X'04'	Test for right or left shift.
		JP	SR4	Jump to right shift.
		C	11, T	Initialize file index.
	SR3	MU	11	File index to U.
		AF	11, X'FF'	Decrement file index.
		E	0, L	Left shift, enter link, file index.

DETAILED FLOW CHART





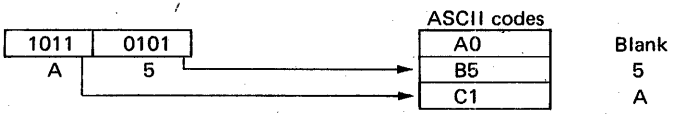
ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
	TZ	11, X'0F'	All files shifted.
	JP	SR3	Shift additional files.
SR6	D	10, C	Decrement shift count.
	TN	0, X'04'	Zero count zero.
	JP	SR1	No.
	JP	EXIT	Done.
SR4	LF	11, X'F1'	Initialize file index.
SR5	MU	11	File index to U.
	AF	11, X'01'	Increment file index.
	X	11, T, C	Test for FU = (T).
	E	0, R, L	Right shift, enter link, file index.
	TN	0, X'04'	Test for last file.
	JP	SR5	Shift more files.
	JP	SR6	Shift count test.

MICROPROGRAM EXAMPLE NO. 19

Hexadecimal to ASCII Conversion Routine.

This routine converts an 8 bit binary number (which is also 2 hexadecimal digits) into two ASCII characters, and also generates an ASCII equivalent for a space. The 3 characters are assembled for sequencing to an output device for print out.

Data Flow:



Typical print out sequence:

A5 F0 D3 C4 . . . . .

Data values and flags are maintained and updated in dedicated locations in core memory. If new characters are ready for output before converted characters are printed out, any queueing will be provided by a different routine. This routine will provide a flag to indicate when it's ready to receive a new character, and sets a flag for output request. Output is done by another routine, which monitors the output request flag of this routine and resets it after outputting a character.

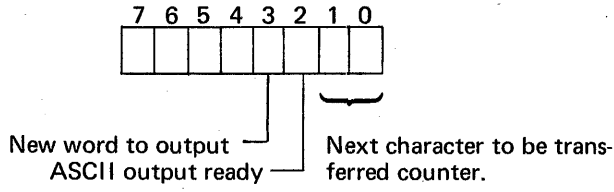
Core Memory Requirements:

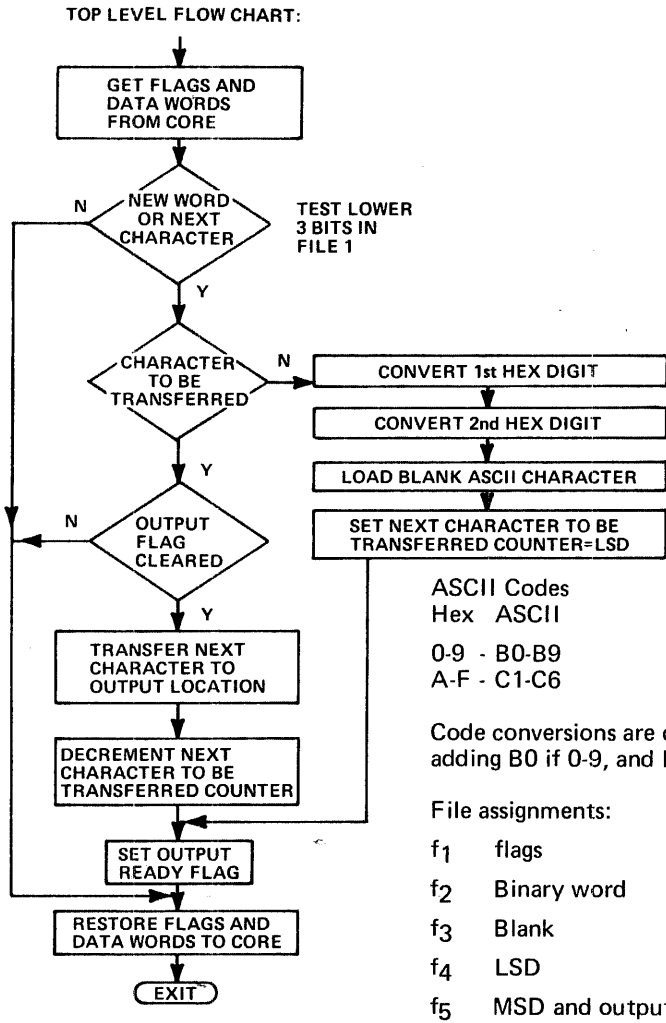
File register		Core	
	1	0001	Flags
	2	0002	Binary word
	3	0003	ASCII for blank
	4	0004	ASCII for least significant digit
	5	0005	ASCII for most significant digit and for output

Next character to be transferred counter
MSD 11
LSD 10
Blank 01
None 00

Zero count here and in bit 2 indicates ready for new character.

Flag word:





Command Count 53.

Execution time for conversion of character is approximately 20 micro-seconds.

A. Routines already described.

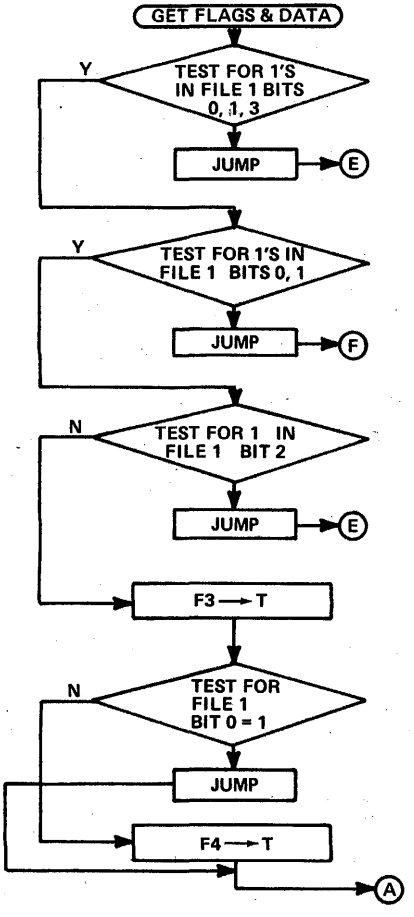
1. Get flags and data words from core.

   This subroutine is the same as subroutine example 7a with the one modification to change the file count from 8 to 5.6 commands required.

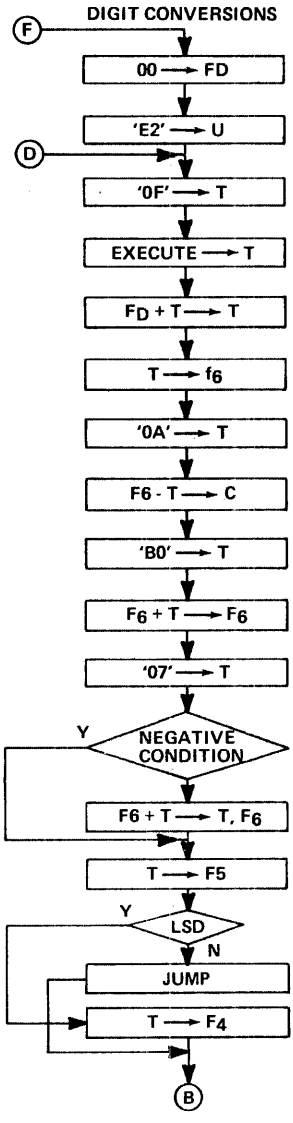
2. Restore flags and data words to core.

   This routine is similar to example 7b except that the file count is changed from 8 to 5. 8 commands required.

B. Detailed flow charts for remaining routines:

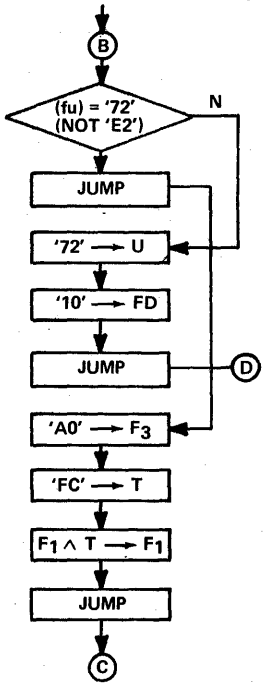


NAME	OPER	OPERAND	COMMENTS
DTA	TN	F1, X'0B'	Test for data to be processed.
	JP	END	
	TN	F1, X'03'	Test for character to be transferred or to be converted.
	JP	CHR	
	TZ	F1, X'04'	Test for output flag cleared.
	JP	END	
	MT	F3	Transfer BLANK.
	TZ	F1, X'01'	Test for LSB-BLANK.
	JP	*+2	Bypass LSD.
	MT	F4	Transfer LSD.
	JP	DEC	Decrement counter.



ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
CHR	LF	FD, X'00'	Clear MSD-LSD flag.
	LU	X'E2'	Set U for And command.
MSK	LT	X'0F'	Mask for LSD.
	ET	0, 2	And with T or shift right 4 depending on (U).
	AT	FD, T	Add (fd) to clear 1's resulting from shift right 4.
	C	F_6, T	Move T to f_6.
	LT	X'0A'	Subtract 0A to test for hex digit 0-9 or A-F.
	S*	F_6, T, C	
	LT	X'B0'	ASCII conversion value added to hex number for 0-9.
	A	F6, T	
	LT	X'07'	Additional ASCII conversion for A-F.
	TN	0, X'02'	Test for value 0-9 or A-F.
	AT	F_6, T	Add final conversion value.
	C	F_5, T	Copy T to f_5.
	TZ	FD, X'10'	Test for least significant digit.
	JP	*+2	Leave in f_5 if MSD.
	C	F_4, T	Move to f_4 if LSD.

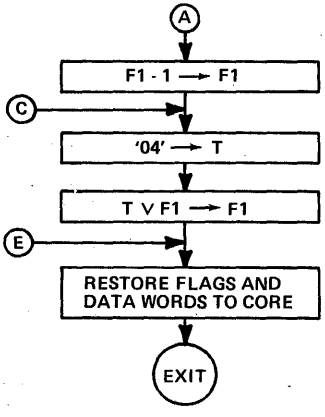
DETAILED FLOW CHART



ASSEMBLY LANGUAGE

NAME	OPER	OPERAND	COMMENTS
	TZ	fu, X'10'	Test bit 4 to indicate '72' vs 'E2'.
	JP	BLK	Jump to load ASCII for blank.
	LU	X'72'	Set U for control command to do SR4.
	LF	FD, X'10'	Set fd for MSD.
	JP	MSK	
BLK	LF	F3, X'A0'	ASCII for Blank.
	LT	X'FC'	Set bits 0 and 1 in f1=0 to clear next character to be transferred counter.
	N	F1, T	
	JP	SET	

DETAILED FLOW CHART



ASSEMBLY LANGUAGE

NAME	OPER	OPERAND	COMMENTS
DEC	D	F1	Decrement next character to be transferred counter.
SET	LT	X'04'	Set ASCII output ready flag.
	O	F1, T	
END			

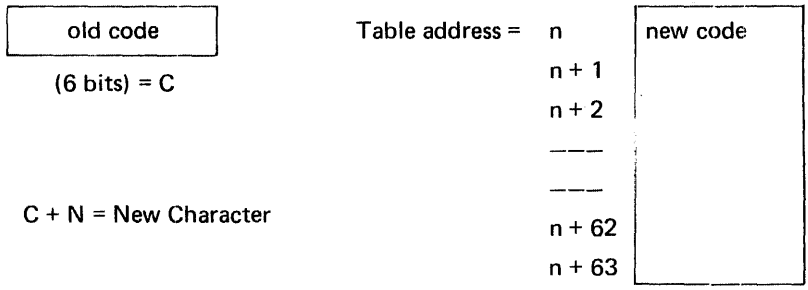
MICROPROGRAM EXAMPLE NO. 20

General Purpose Code Conversion by Table Translation.

This routine will convert a string of characters from any one of 64 characters into any of 64 other characters (character capacity easily changed). The translation table which is in core memory can be loaded with any desired code.

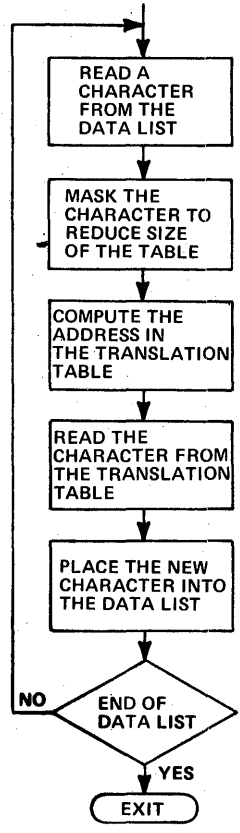
The general approach is to use the character as a displacement value and index into a table to obtain the corresponding character. This type of code conversion is useful where there is no simple mathematical relationship between the two character sets (as with BCD to ASCII, for instance).

Table organization in core:

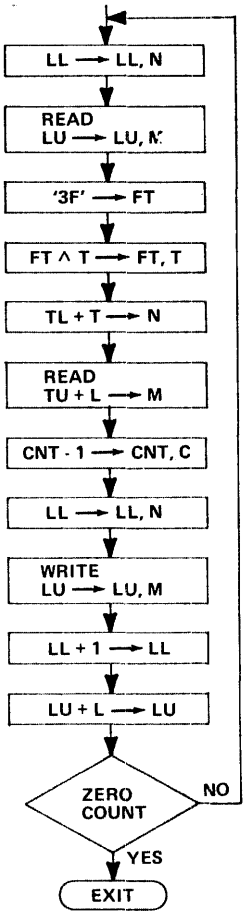


File Assignments:

LL	=	Lower 8-bits of data list address.
LU	=	Upper 7-bits of data list address.
TL	=	Lower 8-bits of translation table address.
TU	=	Upper 7-bits of translation table address.
CNT	=	Number of characters in data list.
FT	=	Mask to limit the table to 64 entries.



This routine uses 13 commands, and takes 4.18 microseconds per character for translation.



ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
TRN	MN	LL	Get a character from the data list.
	RM	LU	
	LF	FT, X'3F'	Set a mask for 64 characters.
	NT	FT, T	Remove unwanted high order bits.
	AN*	TL, T	Add the value of the character with the base address of the table to obtain the new character.
	RM*	TU, L	
	D	CNT, C	Reduce character count.
	MN	LL	Place the translated character back into the data list.
	WM	LU	
	I	LL	Move the data list pointer to the next character.
	A	LU, L	
	TN	0, X'04'	End of List.
	JP	TRN	No, get the next character.

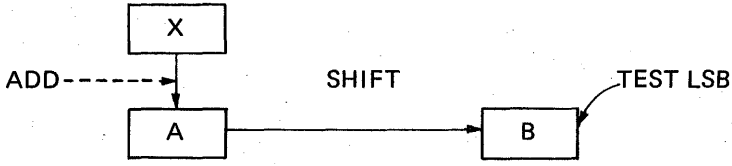
MICROPROGRAM EXAMPLE NO. 21

Binary Multiply (16 bits)

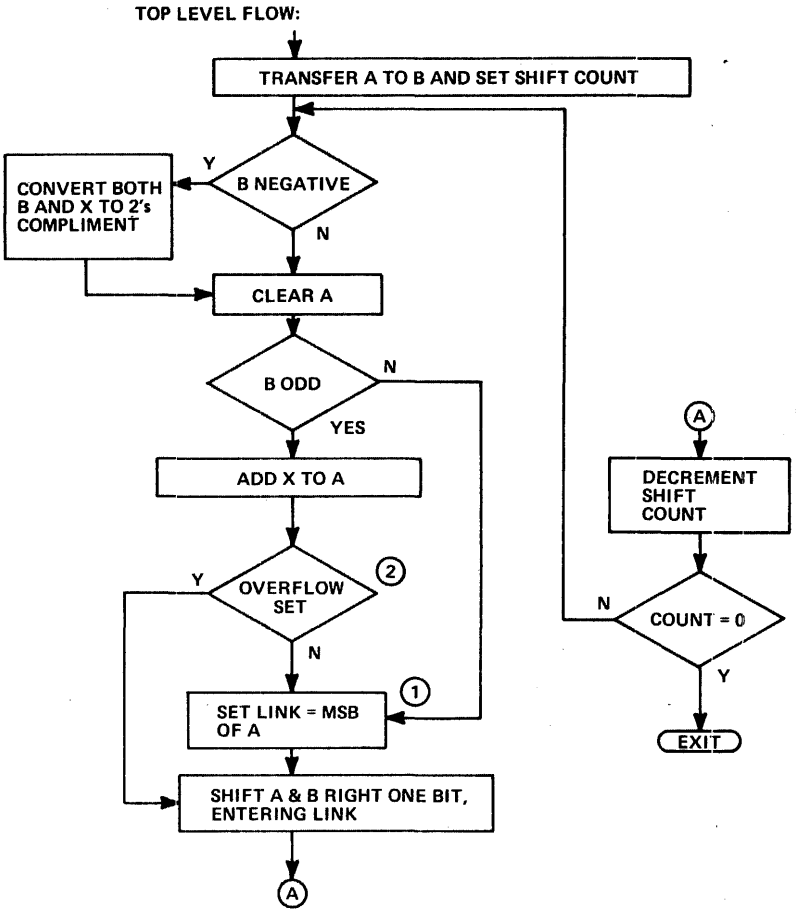
This routine multiplies two 16 bit positive or negative numbers. The two byte operand in X is multiplied by the contents of A and the result is placed in the 32 bit A - B registers. The multiply is an integer type, and the 30 bit resultant magnitude occupies the 30 low order bits of A and B, and a double sign bit occupies the two high order bits.

This example is the same as the routine used in the MICRO 810 firmware except for deletion of memory referencing, concurrent I/O servicing, and linking to the 810 program.

The basic algorithm for this routine consists of testing the LSB of B, and adding X to A whenever LSB of B = 1; then shifting the accumulation right one place, as well as shifting B right one place. Then the next LSB of B is tested. This is repeated until all parts of A have been tested.



To simplify programming, A is first transferred to B, then A is cleared. The contents of A are not tested for sign until after it has first been transferred to B. This is only for convenience of programming. If B is negative, both numbers are 2's complemented. If X is negative, the sign is maintained by sign extension, during shifting.



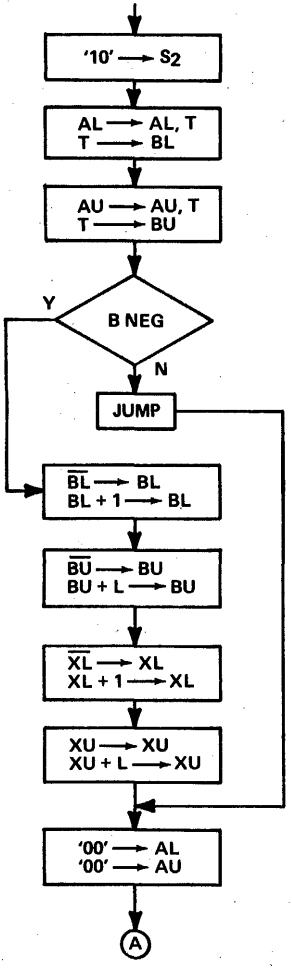
file registers
AU, AL Multiplicand (1st)
BU, BL Multiplicand (2nd)
XU, XL Multiplier
S_2 Shift Count

AU, AL
BU, BL Product

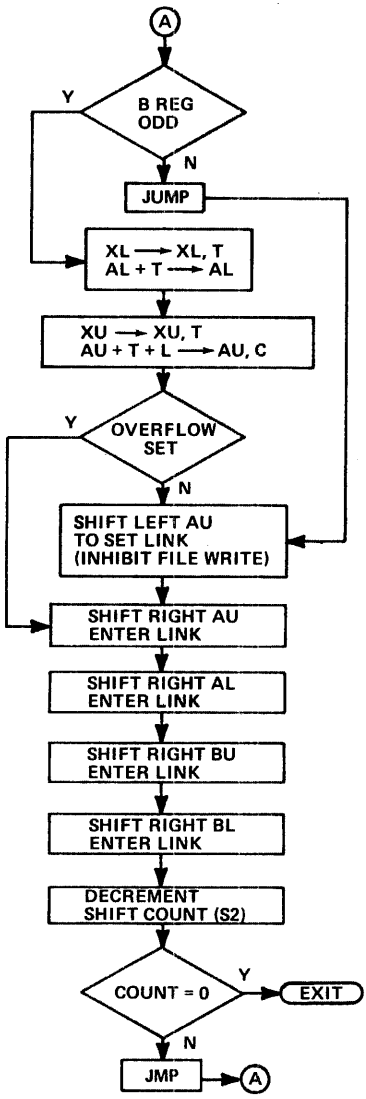
① Link is set to provide for sign extension of the partial accumulation.

② If there is overflow, link is already set to the correct sign value, which may not = MSB of A.

This routine has 32 commands, and takes the following approximate time:
Max. 60 microseconds; Average 54 microseconds.



NAME	OPER	OPERAND	COMMENTS
MUL	LF	S2, X'10'	Set shift count for 16 bits.
	MT	AL	Move A register to B register.
	C	BL, T	
	MT	AU	
	C	BU, T	
	TN	BU, X'80'	Test MSB of BU for negative condition.
	JP	ML3	Bypass complementing.
	X	BL, T, F	2's complement B and X by exclusive ORing with all 1's using T, T as operand and adding 1 to B and X.
	I	BL	
	X	BU, T, F	
	A	BU, L	
	X	XL, T, F	
	I	XL	
	X	XU, T, F	
	A	XU, L	
ML3	LF	AL, X'00'	Clear A after transferring A to B.
	LF	AU, X'00'	



DETAILED FLOW CHART	NAME	OPER	OPERAND	COMMENTS
	ML1	TN	BL, X'01'	Test B for odd.
		JP	ML2	Bypass addition function if B even.
		MT	XL	Add X to (A) and put result in A. Set condition flag for overflow test.
		A	AL, T	
		MT	XU	
		A	AU, T,L,C	
		TN	0, X'01'	Test for overflow.
	ML2	H*	AU	Set link for sign entry.
		H	AU, R, L	Shift A, B right one bit, entering contents of link.
		H	AL, R, L	
		H	BU, R, L	
		H	BL, R, L	
		D	S_2, C	Decrement shift count and set condition flag.
		TN	0, X'04'	Test for zero condition.
		JP	ML1	More bits to be shifted.

MICROPROGRAM EXAMPLE NO. 22

Generate Cyclic Code for one 8 bit data byte.

This routine generates the CRC 16 cyclic redundancy code used in bi-synchronous communication.

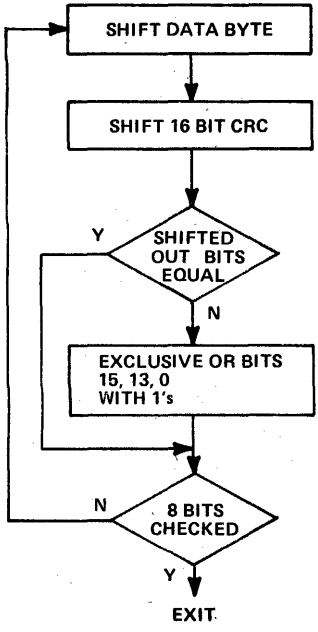
The byte operand in S_1 is entered into the 16 bit cyclic code contained in the A register. The polynomial used for generating the cyclic code is $X^{16} + X^{15} + X^2 + 1$.

The general algorithm is to shift the 16 bit code in A, and to exclusive OR bits 15, 13, and 0 with the result of a comparison between the least significant bits of the cyclic code in A and the least significant bit of S_1 shifted once for each comparison.

This is a microprogram rendition of the feedback shift registers which are used to implement polynomial divisions for generating cyclic codes.

At the beginning of a character string, A should be cleared.

For each 8 bit data byte the top level flow is as follows:



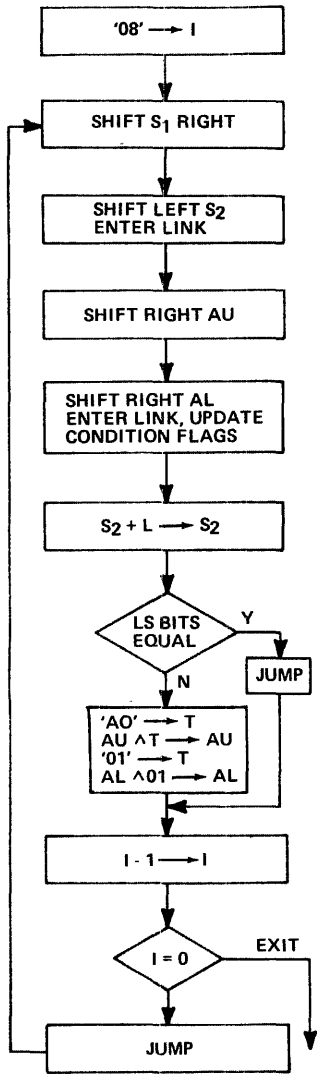
This routine takes 15 commands and takes the following approximate time:

t max.	30 microseconds
t avg.	28 microseconds

file registers

AU, AL	CRC code
S_1	Data byte
S_2	Save Link
I	Shift Counter

This routine is the same as that used in the MICRO 820 except for the omission of memory referencing and linking to the main firmware.

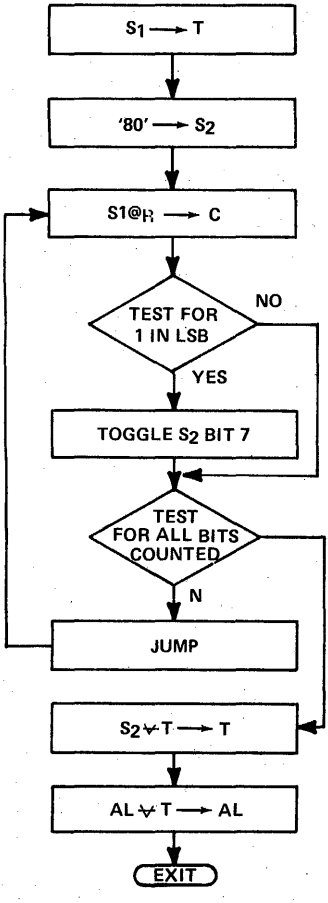


DETAILED FLOW CHART	ASSEMBLY LANGUAGE NAME	OPER	OPERAND	COMMENTS
'08' → I		LF	I, X'08'	Set bit count.
SHIFT S_1 RIGHT	SSR	H	S_1, R	Save LSB of S_1.
SHIFT LEFT S_2 ENTER LINK		H	S_2, L	
SHIFT RIGHT AU		H	AU, R	Shift right CRC word.
SHIFT RIGHT AL ENTER LINK, UPDATE CONDITION FLAGS		H	AL, L,R,C	
S_2 + L → S_2		A	S_2, L	Add link to saved LSB to compare.
LS BITS EQUAL		TN	S_2, X'01'	Equal?
N		JP	NQL	Bypass exclusive OR function.
'A0' → T		LT	X'A0'	Exclusive OR the polynomial function into A.
AU ∧T → AU		X	AU, T	
'01' → T		LT	X'01'	
AL ∧01 → AL		X	AL, T	
I - 1 → I	NQL	D	I, C	Decrement bit counter.
I = 0		TN	O, X'04'	Last bit.
EXIT				
JUMP		JP	SSR	Repeat.

MICROPROGRAM EXAMPLE NO. 23

Generate ASCII Parity.

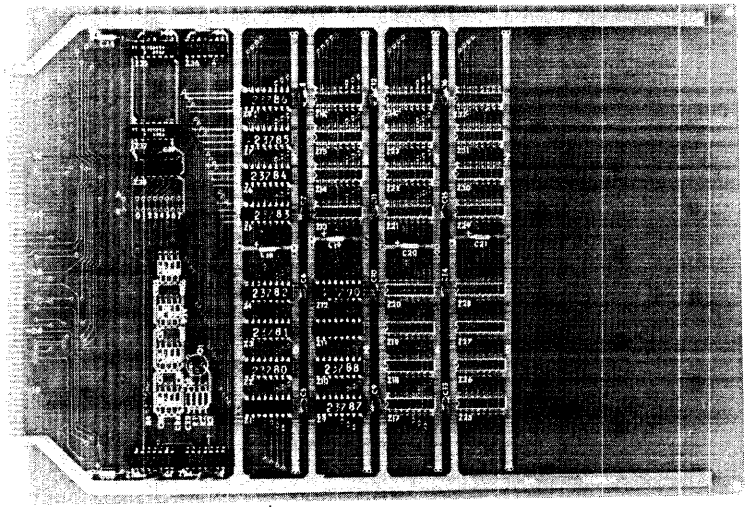
This routine will generate and attach an odd parity bit to bit 7 of a character contained in file S_2. It will also generate a block longitudinal parity LRC for this character, by exclusive ORing with an LRC being accumulated in AL. This routine is the same as used in the 820 except for omission of memory referencing and linking with the main 820 firmware. Parity is generated by shifting and testing the bits in S_1 and toggling a bit in S_2 for each bit = 1 is S_1.



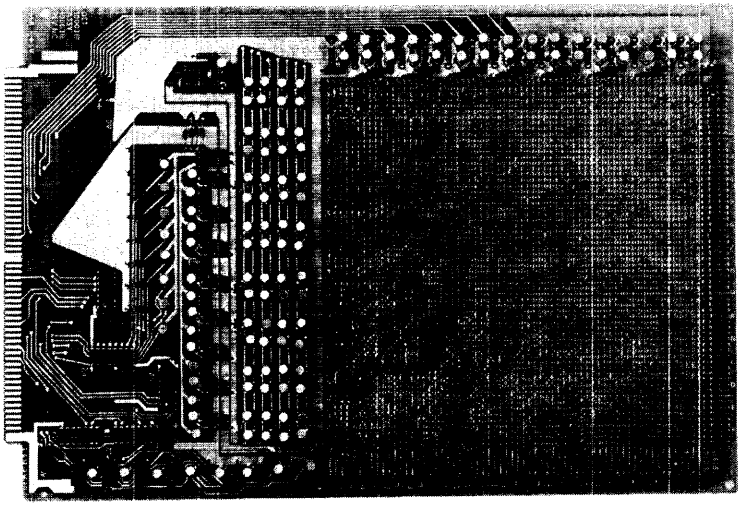
NAME	OPER	OPERAND	COMMENTS
PAR	MT	S_1	Save character.
	LF	S_2, X'80'	Set initial parity.
	H	S_1, R, C	Shift out a bit.
	TZ	0, X'01'	Test for bit = 1.
	AF	S_2, X'80'	Toggle parity bit.
	TN	0, X'04'	Test for all bits shifted out.
	JP	PAR + 1	Repeat.
	XT	S_2, T	Attach odd parity.
	X	AL, T	Generate LRC.

PART IV

MICRO 810 FIRMWARE MANUAL



Semiconductor Read-Only Memory Expandable from 768 Words to 2,048 Words.



Diode Matrix 256-Word Read-Only Memory.

INTRODUCTION

The basic steps for development of a general purpose computer architecture using a microprogrammed computer are as follows:

1. Functional Definition
   - Input/Output Characteristics.
   - Operating Registers Assignments (Accumulator, Index, Program Counter, etc.).
   - Word Length (Fixed and Variable).
   - Core Memory Addressing Modes for Jumps and Operand Fetching.
   - Instruction Repertoire.
   - Instruction and Data Formats (Number of Bytes, Sign Extension, Op Codes, etc.).
   - Interrupt System (External/Internal).
   - Desired Instruction Execution Times.
   - Bootstrap Load Technique.
2. Hardware Modification (if any).

   Modifications or additions may be required (particularly in the interface) to achieve the desired specs. For example if a 16-bit I/O path were required in the emulator, an I/O expander would be required on the MICRO 800. For the MICRO 810 emulation, no hardware changes are required, since the byte I/O scheme is a direct mechanism of the MICRO 800 byte I/O channel.
3. Analysis and Selection Algorithms.

   Definition of subroutines, organization of routine hierarchy and preparation of top level flow chart.
4. Detailed derivation of each algorithm to be used.
5. Preparation of detailed flow charts for each subroutine.
6. Assembly language coding.
7. Assembly of program, diode map generation, and checkout.

To illustrate these steps, annotation flow charts and the assembly language program for the *original version of the MICRO 810 except for compare, multiply, and divide instructions are included, along with a summary of the 810 processor characteristics which affect the firmware.

The MICRO 810 is an example of an emulation. Its characteristics as related to the microprogram are described in the following paragraphs. The first step in development is to define the basic functions.

MICRO 810 Functions

Six operational registers:

- Accumulator (A) – 16 bits.
- Auxiliary accumulator (B) – 16 bits.
- Index register (X) – 16 bits.
- Program counter (P) – 15 bits.
- Overflow (O) – 1 bit.
- Word length control (W) – 2 bits.

Extensive, powerful instruction set including 89 individual operations:

- Multiply and divide (2).
- Control (17).
- Multi-bit arithmetic and logical shifts (12).
- Conditional jumps (16).
- Input/Output (8).
- Inter-register (16).
- Memory reference including jump, compare and variable word length operations – (18).

Eight operand addressing modes including:

- Direct to page 0 (first 256 bytes).
- Direct relative to P (±128 bytes).
- Indirect to page 0 (first 256 bytes).
- Indirect relative to P (±128 bytes).
- Indexed (to 32,768 bytes).
- Indexed with bias (to 32,768 bytes).
- Extended address (to 32,768 bytes).
- Literal.

Multi-precision 1, 2, 3, or 4 byte load, store, and arithmetic operations.
Flexible I/O facilities including:

- programmed transfers to/from A and B registers and memory to byte I/O.
- concurrent buffered I/O.
- serial I/O channel for local teletype.

Expandable priority interrupt system
Processor options which include:

- real-time clock.
- power-fail detect and automatic restart.
- memory parity detect and interrupt.

Built-in bootstrap loader in non-volatile read only store.

*(Later MICRO 810 versions have modified interrupt, concurrent I/O and control firmware.)

To provide all of this capability only 710 micro instructions were required. This leaves capability for addition of 314 additional microinstructions for special functions.

FILE REGISTER ASSIGNMENTS

The MICRO 810 contains six operational registers which are accessible to the programmer. These operational registers occupy nine of the 16 file registers of the basic MICRO 800 hardware; the remaining seven hardware registers are not accessible by the MICRO 810 instructions although specially designed macros could make use of these at the micro-level.

A REGISTER (file registers 4 and 5)

The 16-bit A register is the accumulator with which most operations are performed. The A register holds the upper portion of 24- or 32-bit data words and all of 8- and 16-bit data words. The A register may be shifted by itself or in conjunction with the B register.

B REGISTER (file registers 6 and 7)

The 16-bit B register is the auxiliary accumulator and is used mainly as an extension of the accumulator to hold the lower 16 bits of 24- and 32-bit data. The B register may be shifted by itself or in conjunction with the A register.

X REGISTER (file registers 2 and 3)

The 16-bit X register is an index register used in address modification. It can communicate directly with memory, be incremented, and compared with the A register.

P REGISTER (file registers A and B)

The 15-bit P register is the program counter which holds the address of next memory instruction to be executed.

W REGISTER (bit 2 of file register F)

The 2-bit W register holds the word length mode. It is loaded by a control instruction and sets the byte length of the operand for all variable word length instructions.

O REGISTER (bits 1, 0 of file register F)

The one-bit O register holds the overflow flag. The overflow is set by arithmetic instructions when an overflow occurs, by execution of a control instruction, or by the compare instruction. It may be reset by execution of a control instruction or by a conditional jump instruction that tests for an overflow condition.

Files 8, 9 are for the operand address.

Files C, D, E are used for temporary storage.

File 0 is for condition flags.

File 1 is the instruction register.

The file register assignments are completely accomplished by micro-programming. There are no internal wiring modifications to convert a MICRO 800 to a MICRO 810 other than the arrangement of matrix diodes on the read only memory boards.

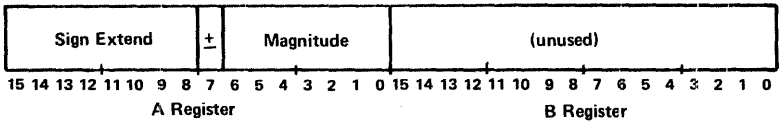
INFORMATION FORMATS

The basic element of information is an 8-bit byte in which the bit positions are numbered from 7 through 0, left to right. Both instructions and data occupy a variable number of bytes for maximum storage efficiency. A word is a 16-bit element of information consisting of two bytes. The accumulator and index register both hold a 16-bit word.

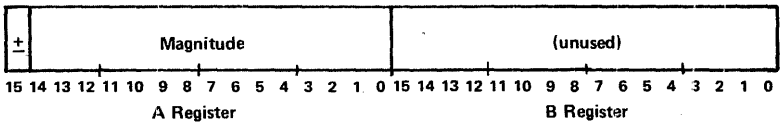
DATA FORMAT

Data in the MICRO 810 is variable precision of 8, 16, 24, or 32-bit length. Negative numbers are represented in 2's complement.

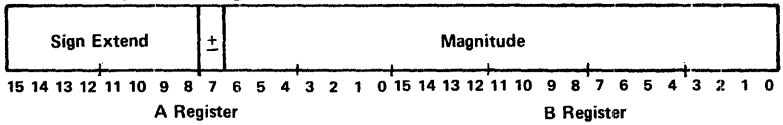
8 Bits (1 Byte) -- Range: $+2^7-1$ to -2^7



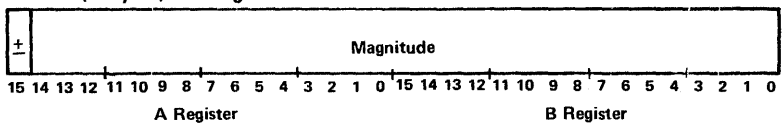
16 Bits (2 Bytes) -- Range: $+2^{15}-1$ to -2^{15}



24 Bits (3 Bytes) -- Range: $+2^{23}-1$ to -2^{23}



32 Bits (4 Bytes) -- Range: $+2^{31}-1$ to -2^{31}

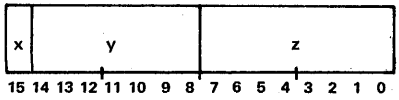


To have variable word length operations, the microprogram must test the instruction Op code, bit 3 to see if variable word length is specified. It must then test file register F, bit 2 for which word length is set. Then the instruction is carried out by the microprogram according to the settings of these two bits. Testing and variable word length execution are done in the designated memory reference microprogram subroutine.

ADDRESS WORD FORMAT FOR MEMORY REFERENCE INSTRUCTIONS

A 16-bit address word containing a 15-bit memory address and an index flag as shown below. The address may be a direct or indirect address as dictated by the instruction operation code. The value of the address word

is equal to the contents of bits 14-0 and is equal to the contents of bits 14-0 plus the contents of the x register if bit 15 is a 1-bit.



In the operand address subroutine, the address is determined by the microprogram and placed into the operand address register.

INSTRUCTION FORMAT

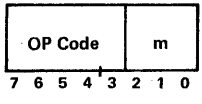
Instruction formats are one to five bytes, but in all cases the first contains an eight-bit operation code which defines the operation class, the suboperation code, and any modifiers. Succeeding byte(s) contain such information as:

Single byte absolute or relative address.
Double byte address word.
Single byte shift count.
Single byte I/O function and device address.
1, 2, 3, or 4 byte literal data.

OPERAND ADDRESSING MODES

The memory reference instructions defined in the following section each have eight possible modes of addressing an operand in memory. The number of bytes in the instruction format varies with the mode. The additional bytes of the instruction contain addresses, partial addresses, or data (literals).

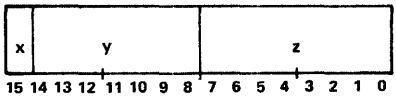
The basic memory reference instruction is one byte containing two fields as follows:



The 5-bit operation code defines the basic instructions; the 3-bit m field specifies the address mode. Additional bytes contain the address of an operand, an indirect address, a base address, or a literal depending on the addressing mode. The effective operand address is the memory location specified after all indirect and/or index modifications have been performed.

For variable word length instructions, such as Load A versus Load Variable, bit 3 is used to indicate whether variable word length is to be used. The microprogram tests this bit. For fixed word length instructions, such as multiply/divide, bit 3 indicates the instruction type.

When an indirect address mode is specified, the location of the indirect address word is the first byte of a two-byte word having the format shown below:



Indirect Address Word Format

For indirect addressing, the microprogram fetches the first referenced word, which points it to the actual address word, to which may be added the contents of the index register.

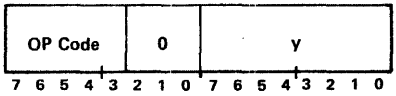
Bit 7 of the first byte (x) defines whether or not the indirect address word will be modified by the contents of the index register:

If x = 0, the 15-bit number formed by y and z is the effective operand address.

If x = 1, the 15-bit number formed by y and z is a base address to which is added the contents of the X register. The result is the effective operand address.

The individual addressing modes and the memory reference instruction format for that mode are defined below. The microprogram has a subroutine called operand addressing which examines the subsequent bytes of memory reference instructions, and uses this information to determine the operand address.

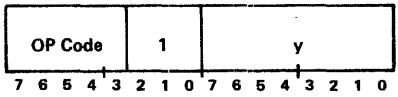
DIRECT PAGE 0 (m=0)



The effective operand address is given by the contents of the second byte of the instruction (y) with seven high order zero bits appended. This mode provides direct addressing of operands in the first 256 memory locations.

The microprogram clears the upper byte of the operand address register, and places byte y in the lower byte of the operand address register.

DIRECT RELATIVE (m=1)

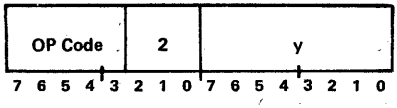


The effective operand address is given by the sum of the contents of the second byte (y) with its high order sign bit (bit 7) extended and the contents of the P register. The contents of the P register at the time the addition is performed is the address of the memory location following y. This

mode provides for addressing from 127 locations ahead to 128 locations behind the memory location of the next instruction.

The microprogram sets the P register to the next instruction location, adds the byte in y to p and places the result in the operand address register.

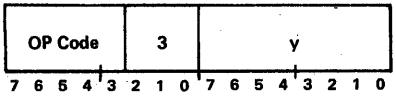
INDIRECT PAGE 0 (m=2)



An indirect address word is specified by the contents of the second byte (y) of the instruction with seven high order zero bits appended. The 2-byte indirect address word addressed is located in the first 256 memory locations. The effective operand address is given by the contents of the indirect address word if the index flag (bit 15) is a 0-bit, or by the sum of the contents of the indirect address word and the X register if the index flag (bit 15) is a 1-bit.

The microprogram fetches the two byte address from page 0 designated by byte Y. It adds the contents of the index register (if bit 15=1), and places the result in the operand address register.

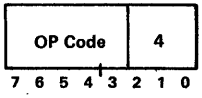
INDIRECT RELATIVE (m=3)



An indirect address word is specified by the sum of the contents of the second byte (y) with its high order bit (bit 7) extended and the contents of the P register. The contents of the P register at the time the addition is performed is the address of the memory location following y. The effective operand address is given by the contents of indirect address word if the index flag (bit 15) is a 0-bit or by the sum of the contents of the indirect address word and the X register if the index flag (bit 15) is a 1-bit.

The microprogram advances the P counter to the next instruction location, adds the content of byte y, fetches the 2 byte address from the resultant location, adds content of index (if bit 15=1) and places the result in the operand address register.

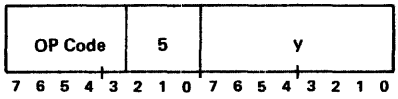
INDEXED (m=4)



The effective operand address is given by the contents of the X register.

The microprogram loads the content of X into the operand address register.

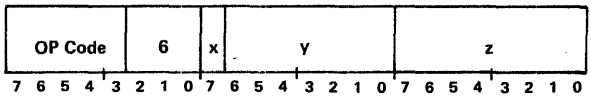
INDEXED WITH BIAS (m=5)



The effective operand address is given by the sum of the contents of the X register and the contents of the second byte (y) of the instruction.

The microprogram adds the content of X to byte Y, and places the result in the operand address register.

EXTENDED ADDRESS (m=6)



A 16-bit address word is located in the second and third byte of the instruction. The effective operand address is given by the contents of the address word if the index flag bit in bit 15 is an 0-bit, or by the sum of the contents of the address word and the X register if the index flag is a 1-bit.

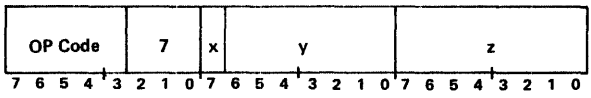
The microprogram takes bytes Y, and Z and adds the contents of index if bit X=1 and places the result in the operand address register.

LITERAL (m=7)

The effective operand address is given by the contents of the P register. The operand is located in from 1-4 bytes following the first byte of the instruction, depending upon the operand precision. The P register is incremented for each operand byte accessed. The Jump and Return Jump memory referencing instructions do not have a literal mode.

The microprogram places the contents of the P register into the operand address register.

JUMP/RETURN JUMP INDIRECT EXTENDED ADDRESS (m=7)



A 16-bit direct address word is located in the second and third bytes of the instruction. This word addresses an indirect address word located at

the address given by the contents of the second and third bytes if bit 15 of the address word is a 0-bit or by the sum of the contents of the second and third bytes and the X register if the index flag bit in bit 15 is a 1-bit.

The effective jump address is given by the contents of the indirect address word if the index flag in bit 15 of the indirect address word is a 0-bit, or by the sum of the contents of the indirect word and the X register if the index flag bit in bit 15 of the indirect address word is a 1-bit.

The microprogram tests to see if mode = 7, and the command is a jump or return jump. If all of these conditions are so, the microprogram fetches the bytes Y, Z (with index if bit X=1) and places them in the operand address register.

MICRO 810 INSTRUCTIONS

OPERATION CODE	MNEMONIC	INSTRUCTION NAME
CONTROL (one byte)		
00	HLT	Halt
01	TRP	Trap
02	ESW	Enter Sense Switches
03	PMP	Protect Memory Page
04	DIN	Disable Interrupt System
05	EIN	Enable Interrupt System
06	DRT	Disable Real Time Clock
07	ERT	Enable Real Time Clock
08	RO1	Reset Overflow and Set Word Length to 1
09	RO2	Reset Overflow and Set Word Length to 2
0A	RO3	Reset Overflow and Set Word Length to 3
0B	RO4	Reset Overflow and Set Word Length to 4
0C	SO1	Set Overflow and Set Word Length to 1
0D	SO2	Set Overflow and Set Word Length to 2
0E	SO3	Set Overflow and Set Word Length to 3
0F	SO4	Set Overflow and Set Word Length to 4
34	NOP	No Operation
CONDITIONAL JUMP (2 bytes)		
10 XX	JOV	Jump if Overflow Set
11 XX	JAZ	Jump if A Equal to Zero
12 XX	JBZ	Jump if B Equal to Zero
13 XX	JXZ	Jump if X Equal to Zero
14 XX	JAN	Jump if A Negative
15 XX	JXN	Jump if X Negative
16 XX	JAB	Jump if A Equals B
17 XX	JAX	Jump if A Equals X
18 XX	NOV	Jump if Overflow not Set
19 XX	NAZ	Jump if A not Equal to Zero
1A XX	NBZ	Jump if B not Equal to Zero
1B XX	NXZ	Jump if X not Equal to Zero
1C XX	NAN	Jump if A not Negative
1D XX	NXN	Jump if X not Negative
1E XX	NAB	Jump if A not Equal to B
1F XX	NAX	Jump if A not Equal to X

Where: XX is a relative jump address (plus or minus hex 7F from the first byte after the jump instruction.

OPERATION CODE	MNEMONIC	INSTRUCTION NAME

SHIFT (2 byte instruction)

20 XX	LLA	Logical Left A
21 XX	LLB	Logical Left B
22 XX	LLL	Logical Left Long
24 XX	LRA	Logical Right A
25 XX	LRB	Logical Right B
26 XX	LRL	Logical Right Long
28 XX	ALA	Arithmetic Left A
29 XX	ALB	Arithmetic Left B
2A XX	ALL	Arithmetic Left Long
2C XX	ARA	Arithmetic Right A
2D XX	ARB	Arithmetic Right B
2E XX	ARL	Arithmetic Right Long

Where: XX is shift count.

INPUT/OUTPUT (2 and 4 byte instruction)

30 00		IBS	Input Byte Serially
31 XX		IBA	Input Byte to A
32 XX		IBB	Input Byte to B
33 XX	AAAA	IBM	Input Byte to Memory
38 00		OBS	Output Byte Serially
39 XX		OBA	Output Byte from A
3A XX		OBB	Output Byte from B
3B XX	AAAA	OBM	Output Byte from Memory

Where: XX is a 3-bit function code and 5-bit device address. AAAA is a core memory address.

REGISTER OPERATE (one byte)

Group 1		
40	ORA	OR B with A
41	XRA	Exclusive – OR B with A
42	ORB	OR A with B
43	XRB	Exclusive – OR A with B
44	INX	Increment X
45	DCX	Decrement X
46	AWX	Add Word Length to X
47	SWX	Subtract Word Length from X
Group 2		
48	INA	Increment A
49	INB	Increment B
4A	OCA	One's Complement A
4B	OCB	One's Complement B
4C	TAX	Transfer A to X
4D	TBX	Transfer B to X
4E	TXA	Transfer X to A
4F	TXB	Transfer X to B

OPERATION CODE	MNEMONIC	INSTRUCTION NAME
MEMORY REFERENCE (1, 2, 3, 4, 5 byte)		
60	JMP	Jump
68	RTJ	Return Jump
70	IWM	Increment Word in Memory
78	DWM	Decrement Word in Memory
80	LDX	Load X
88	STX	Store X
90	MUL	Multiply
98	DIV	Divide
A0	ADA	Add to A
A8	ADV	Add Variable
B0	SBA	Subtract from A
B8	SBV	Subtract Variable
C0	CAP	Compare A
C8	CPV	Compare Variable
D0	ANA	And
D8	ANV	And Variable
E0	LDA	Load A
E8	LDV	Load Variable
F0	STA	Store A
F8	STV	Store Variable

INTERRUPTS

The MICRO 810 has firmware to process both external and internal interrupts. The firmware tests for interrupts, acknowledges them, and executes a return jump to the designated software routine for each interrupt channel.

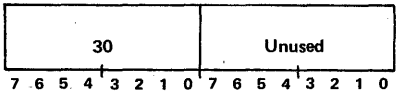
CONCURRENT I/O

The concurrent I/O allows for block transfers between the external device on the Byte I/O bus and memory at a maximum rate of 20,000 bytes per second. The transfers are fully automatic, and once started proceed without program intervention. Concurrent I/O takes priority over instruction execution and forces momentary sequence breaks during execution of long instructions such as multiply, divide and shifts to insure that concurrent I/O displays are not excessive.

SERIAL INPUT/OUTPUT INSTRUCTIONS

Two instructions are provided for bit serial transfers of data between the A register and a serial I/O device. In the MICRO 810, these instructions are standardly timed to transfer bits at the rate of 110 bits/second for interface with a serial teletype. However, the timing can be easily altered by a simple change of firmware to handle another type of serial device.

IBS INPUT BYTE SERIALLY



An eight-bit byte is assembled from the serial teletype interface and placed in the eight low order bits of the A register. The eight high order bits of A remain unchanged. The execution time of this instruction terminates when a complete teletype character has been received. The instruction must be accessed before the start of the teletype input for proper assembly of the character. Sampling of the teletype line and assembly of bits is done by a microprogram subroutine, which includes its own delay routine to time out the bits as shown below.

OBS OUTPUT BYTE SERIALLY

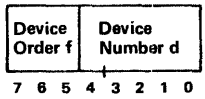
The eight low order bits of the A register are disassembled and output serially as a teletype character to the serial teletype interface. The eight low order bits of A will be set to one. The eight high order bits remain unchanged. The execution of this instruction terminates when a complete byte has been transmitted.

Affected: A

BYTE INPUT/OUTPUT INSTRUCTIONS

Byte programmed input/output operations provide transfers of data, control and status over the Byte I/O channel. This multiplex channel permits intermixed program and concurrent I/O transfers. More than one device on the bus may be operating in a concurrent block transfer mode at the same time. A maximum of 32 devices may normally be addressed on the Byte I/O bus.

The second byte of the instruction is a control byte which provides a three-bit device order and a five-bit device number as follows: The microprogram causes the second byte to be placed on the output bus, and generates a control output strobe called COXX. In the output mode, the data is placed on the output bus and strobed out with DOXX. For input, data on the input bus is strobed in by DIXX.



Byte input/output is basically a two-phase operation. First the control byte is placed on the output bus before the actual transfer of data. All devices examine the transmitted device number. The device whose assigned number is the same as contained in the control word accepts the control byte and sets for a subsequent data byte transfer. The second phase consists of the input or output of a single byte. When a device order does not require a data transfer, the second byte is disregarded by the device controller.

TOP LEVEL FLOW CHART

The purpose of the top level flow chart is to define the microprogram subroutines, and their interrelationship. This flow chart shows all of the basic paths that the microprogram can follow as it goes through its repetitive looping operation.

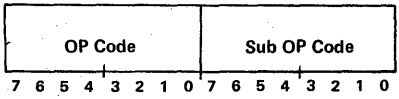
The top level flow chart can be divided into six major areas for discussion purposes.

- Instruction fetching
- Interrupt and Concurrent I/O Processing
- Operand Addressing
- Nonmemory Reference Instruction Execution
- Memory Reference Instruction Execution
- Bootstrap Load

Instruction Fetching

MICRO 810 instructions, stored in core, contain from 1 to 5 bytes, depending on the instruction. During the instruction fetch routine, only the first byte is fetched from core. This byte contains the basic Op code of the instruction, which identifies whether the instruction is memory reference or not, and what the specific instruction is.

First byte format.



The Op code identifies the class of instruction for nonmemory reference instructions, and the type of instruction for memory references.*

The sub Op code identifies the type instruction for nonmemory reference, and the address mode, and fixed versus variable word length for the memory reference instructions.

The Op codes are organized so that all memory reference instructions have Op codes 6. The microprogram makes use of this fact when testing to see if the instruction is memory reference.

During the instruction fetch subroutine, the Op code is tested for memory reference, and a jump table number is set up to jump into the subroutine corresponding to the Op code.

Other things done during instruction fetch are testing for interrupt, and advancing the program counter.

The instruction fetch routine contains a cold start portion which initializes the program counter, tests for internal interrupts, and tests for bootstrap load.

*On some of the memory reference instructions the sub Op code is also required to indicate type of instruction.

The instruction fetch routine has many different entry points, which are a function of the state of the P register as determined by the previous subroutine that the microprogram executed.

Interrupt Processing

If there is an internal or external interrupt, the microprogram services it immediately. Servicing consists of acknowledging the interrupt, inputting the device address (if external), and jumping to the interrupt routine, or transferring a data byte if concurrent I/O. When this is done, the microprogram returns to the instruction fetch cycle. At this time, the interrupt routine address will be in the program counter.

Operand Addressing

This microprogram subroutine prepares the absolute address of the operand of a memory reference instruction, and places it in the operand address register. The address modes are identified in the sub Op code. Address information is contained in the 2nd and 3rd bytes of the instruction.

The addressing modes are as follows:

1. Direct Page 0 (1st 256 bytes)

   The second byte is placed directly in the operand address register by the microprogram.

2. Direct Relative

   The second byte is added to the P counter, and the result is placed in the operand address register.

3. Indirect Page 0 (1st 256 bytes)

   The address indicated by the second byte is fetched from Page 0 and added with the contents of the index register (if bit 15 is set), and placed in the operand address register. If bit 15 is not set, the address is placed directly in the operand address register.

4. Indirect Relative

   The second byte is added to the P counter. This address is used to fetch the indirect address, which is added to the content of the index register (if bit 15 is set), and placed in the operand address register. If bit 15 is not set, the indirect address is placed directly in the operand address register.

5. Indexed

   The address in the index register is transferred to the operand address register.

6. Indexed With Bias

   The 2nd byte is added to the index register and placed in the operand address register.

7. Extended Address (Absolute Address)

   The 2nd and 3rd bytes of the instruction are added to the index register (if bit 15 is set) and placed in the operand address register. If bit 15 is not set, the 2nd and 3rd bytes are placed directly in the operand address register.

8. Literal

   The P counter is incremented and placed in the operand address register.

Non-memory Reference Instruction

The non-memory reference instructions consist of the following:

- Conditional Jumps
- Input/Output a byte of data (Parallel or Serial)
- Control Operations
- Register Shifts
- Register Operations

Since none of these involve an operand to be fetched from memory, the operand addressing function is bypassed by the microprogram.

Memory Reference Instructions

The memory reference instructions are grouped as follows:

- Load, Add, And, Subtract
- Store
- Unconditional Jump
- Return Jump
- Increment or Decrement Word in Memory
- Compare
- Multiply, Divide

The operand for each of these operations is fetched from the address location contained in the operand address register.

Bootstrap Load

This microprogram is entered from the cold start part of the instruction fetch routine. It loads a program load routine which is on paper tape.

Detailed Flow Charts

The next step after preparing the top level flow chart is to prepare the detailed flow charts for the individual subroutines. At this time it is necessary to have a detailed definition of the procedures, equations, and algorithms to be executed in each subroutine. The basic microprogramming approaches must be identified, such as use of the U register, combining multiple functions into the same routine, a definition of microprogram jump and return jump procedures.

There is no set rule for the detail level of symbology to be used in microprogram flow charts. The general considerations for detail level are as follows:

1. Ease of identifying and defining procedures.
2. Ability to communicate program organization and steps to others.
3. Ease of coding program from flow charts.

To provide a detailed description of the MICRO 810 firmware selected, detailed flow charts, comments, and functional grouping indications are included in the following pages, along with a table of symbol definitions to facilitate reading the charts. Microcode addresses are included on the flow charts to facilitate relating the steps in the flow chart to the instructions in the assembly listing.

Glossary of Flow Chart Symbols for MICRO 810 Firmware

A. File Registers

F_0	File 0	Flag Register.
I	File 1	Instruction Register (for first byte of instruction).
XL XU	File 2 File 3	Upper and Lower Bytes of Index Register.
AL AU	File 4 File 5	Upper and Lower Bytes of A Register.
BL BU	File 6 File 7	Upper and Lower Bytes of B Register.
OL OU	File 8 File 9	Upper and Lower Bytes of Operand Address Register.
PL PU	File A File B	Upper and Lower Bytes of Program Counter Register.
S_1	File C	Temporary, Always Used for Subroutine Return Address.
S_2 S_3	File D File E	Temporary.
OV/W	File F	Overflow and Word Length.
F1	File 1	Used for execute command reference Register for selecting odd file. This does not actually select file 1 because of the U register modification.
U_L, U_u		Designates a command selecting U register modification with File 0 reference, and modified by U register.

B. Other Registers

T	T Register
U	U Register
L	L Register (also referred to as K in assembly language)
m,M	M Register Upper Memory Address Register
n,N	N Register Lower Memory Address Register
L,LK	Also defined as LINK
C	Update condition Flags.

C. Miscellaneous Mnemonics

SS4
RN1
OP

D. Symbols for Constants

1. Constants to go into U Register for Instruction Modification

LDAL	Loading A using 'B4' op code which is COPY T to A.
ANAL	AND A using 'C4' op code which is AND to A.
SBAL	Subtract A using '94' op code which is Subtract from A.
ADAL	Add A using '84' op code which is ADD to A.
LDXL	Load X using 'B2' op code which is Copy T to X.
STAL	Store A using 'A4' op code which is Memory op code.
STXL	Store X using 'A2' op code which is Memory op code.
STBL	Store B using 'A6' which is memory op code.

2. Jump Table Constants

OTAB = '10'	Main table jump reference to location 100. The '10' is used to clear the upper 4 ones in the op code which has been shifted right 4 places.
JTBL = '4E'	Jump base reference constant used in conditional Jump routine, to go to the selected conditional jump subroutine.
CTBL = '15'	Jump base reference constant used in control routine for jumping to selected control function.

E. Miscellaneous Symbols

SR4	Shift Right 4
SS4	Sense Switch 4
CTL	Control Subroutine
CJ	Conditional Jump Subroutine
SH	Shift Routine
IO	Input Output Routine
REG	Register Operate Routine
SP	Spare
RNI	Read Next Instruction
JMP	Jump
RTJ	Return Jump
IND 1 INDX	Entry points to perform indexing
ADDR, ADRO	Entry points in operand addressing routine
OP	Op Code
SOF	Set Overflow
SET	Set Mask
OCK	Test for Overflow set
LDA	Load A
ANA	And A
SBA	Subtract A
ADA	Add to A
MR1	Memory Reference Entry from LDX
LDX	Load X
STA	Store A
IWM	Increment Word in Memory
M	Address Mode (sometimes M Register)
@	Shift
V	OR Logic Symbol
∧	And Logic Symbol
∀	Exclusive OR Logic Symbol

F. Microprogram Command Symbols

'00' ⟶ OV, m	Load OV and m registers with '00' to clear them.
Pu ⟶ Pu, m	Move content of Pu to m (back to Pu is immaterial but saves a diode).
PL ⟶ P1, n (READ)	Initiate a read memory cycle and also move content of PL to n Register.
I SR4 ⟶ T	Shift right file 4 and put result in T.
PL+1 ⟶ PL, n	Increment (PL) and put result in n and PL.
W∧ T ⟶ T	'AND' (W) with (T) and put result in T.
U_L (F) T ⟶ U_L, C	General Purpose Command. UL Selectable file by U register. F Selectable command by U register. T Operand, Up date condition flags.
NOP N	No Operation.
JP*+1	Jump to next location (2 clock delay).
U_u ⟶ T (Write)	Execute command with memory op code in U register, T destination and write bit set in C field.
BL(F)T ⟶ BL	Execute command selecting B register, with variable op code in U register. T register operand.
S_1@+1 ⟶ S_1, U	Shift file S_1 right, enter 1 into vacated bit, place result in S_1 and U.
U_L+1 ⟶ U_L	Incrementing selected register with file address modified by content of U Register.
CTBL ⟶ SI	Load file S_1 with constant identified as CTBL.
$\overline{U}_L$ ⟶ U_L	Complement selected file.
I∀T, $\overline{T}$ ⟶ LK	Exclusive OR (I) with T and $\overline{T}$ thus complementing (I).
BL@+LK ⟵ BL	Shift (BL) left, enter (LINK).
U_u@+LK ⟶ U_u	Shift selected file right, enter link. File designated by contents of U.
S_1 ⟶ L	Move (S_1) to L (a jump command).
I@ ⟵ T	Shift (I) left, result to T.

On the flow charts, the machine code address of each instruction is placed next to the box containing the instruction, as close as possible. Since jump instructions are not shown in the boxes, a dot is placed in the flow line having the jump and identified with the machine code address for the jump instruction. When the jump destination is indicated with (INT)←—, the machine code address of the jump destination is placed by the circle as follows 1F8 (INT2)←—

The flow charts are shown in Figures 23 through 39.

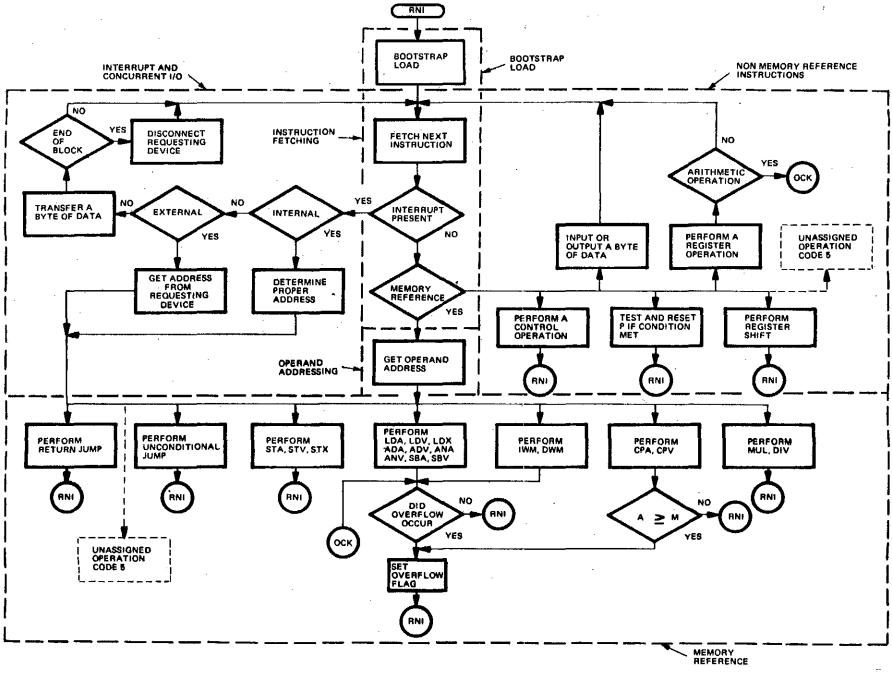


Figure 23. 810 Top Level Flow Chart

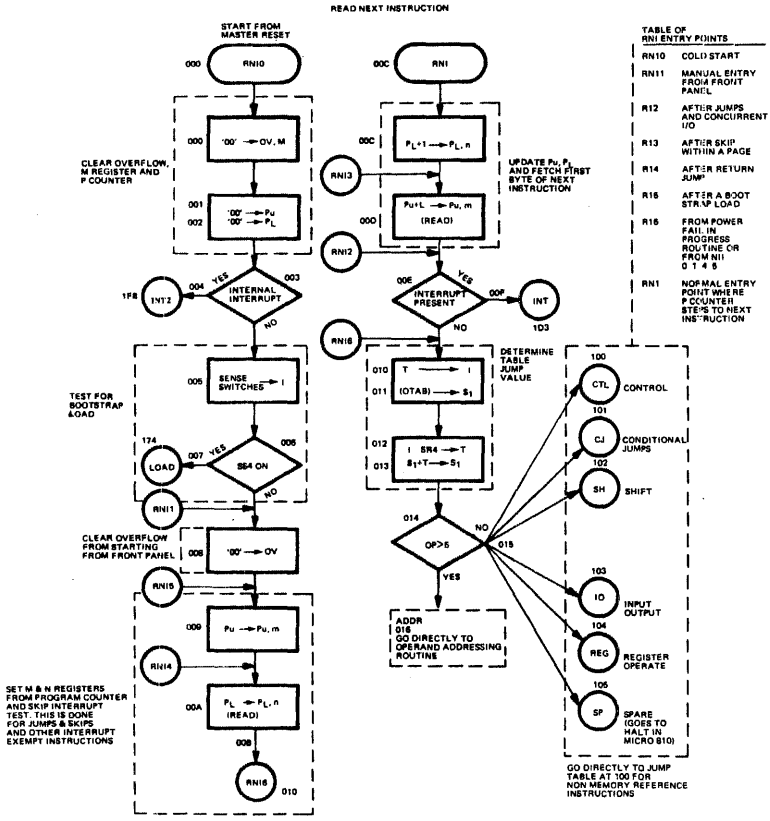


Figure 24. Read Next Instruction

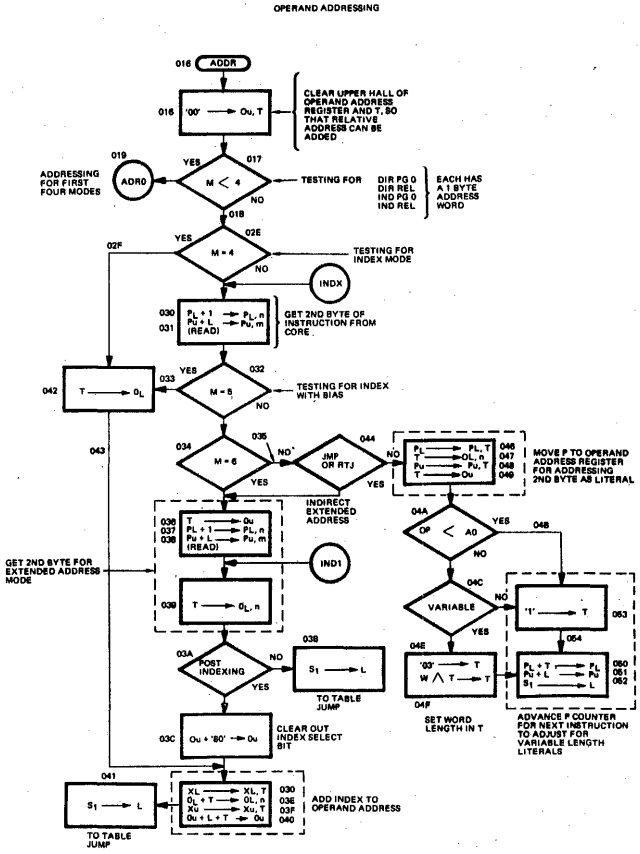


Figure 25. Operand Addressing

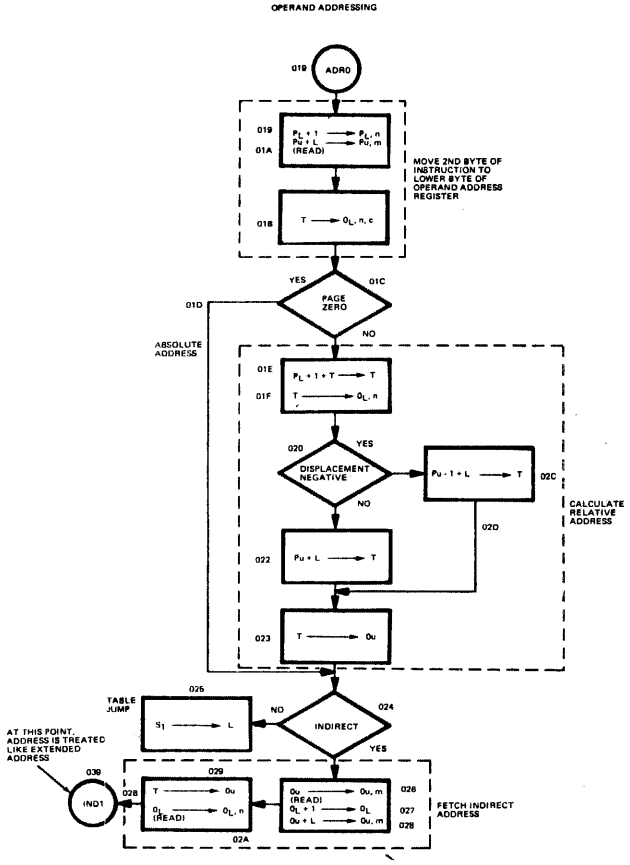


Figure 26. Operand Addressing (Continued)

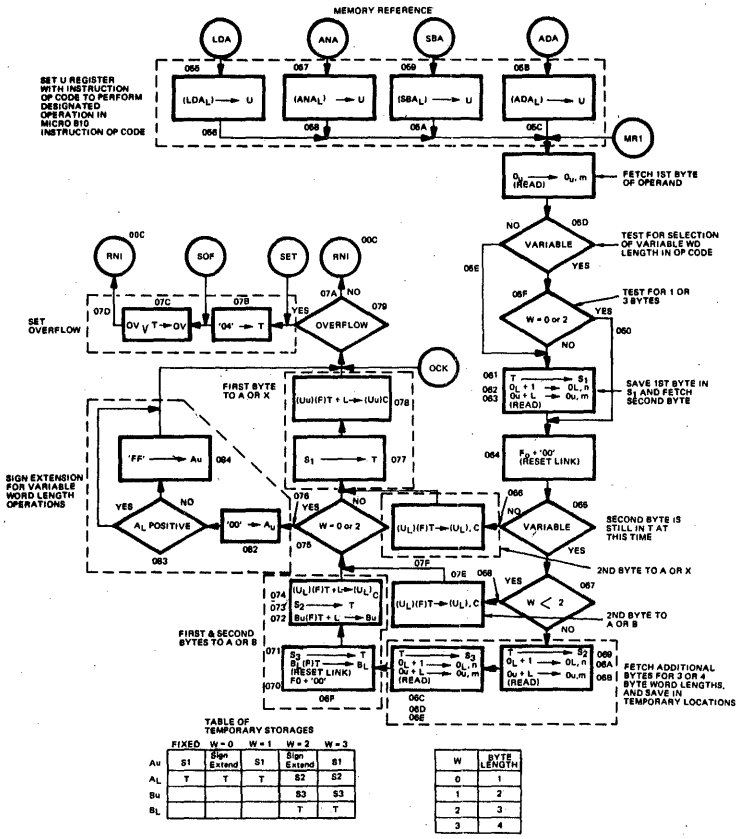


Figure 27. Memory Reference

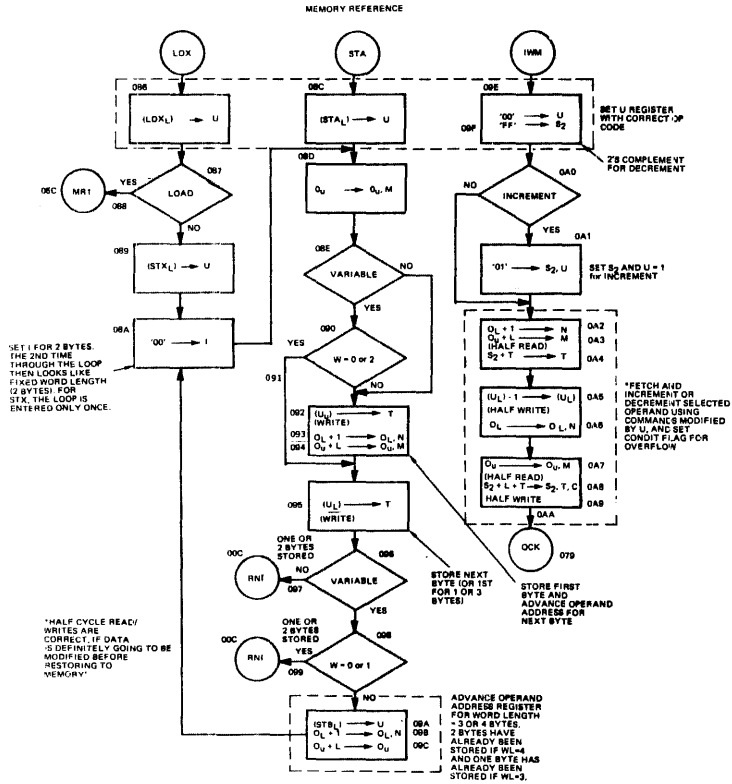


Figure 28. Memory Reference (Continued)

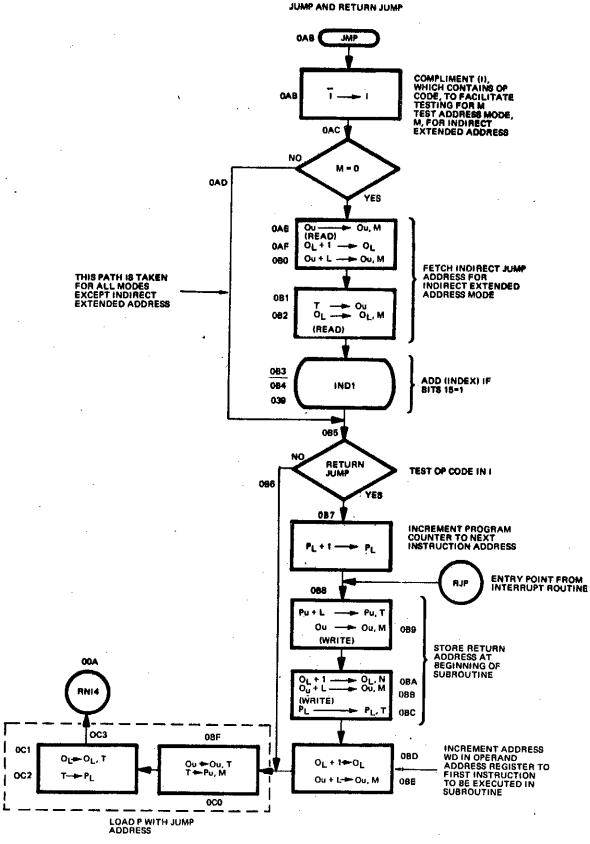


Figure 29. Jump and Return Jump

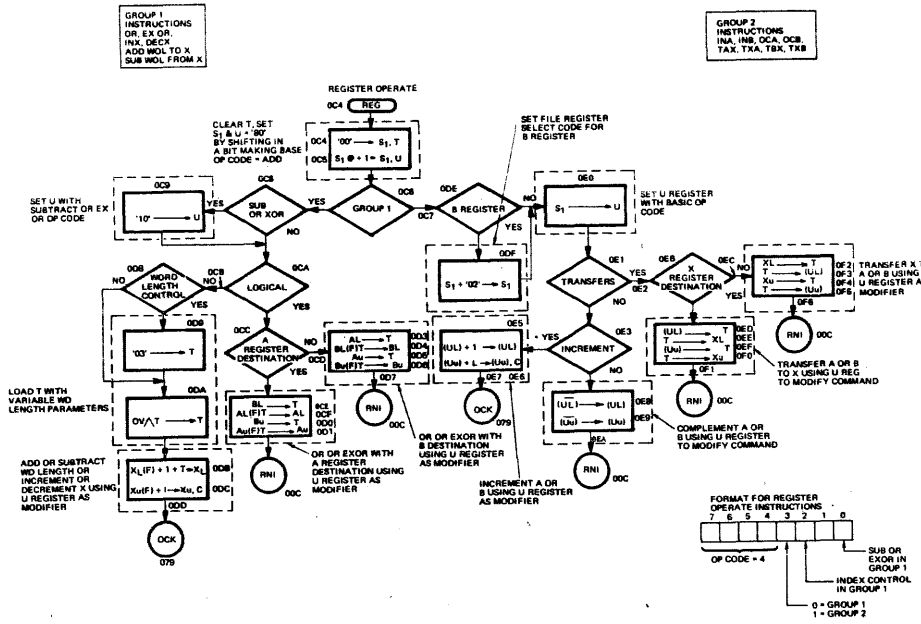


Figure 30. Register Operate

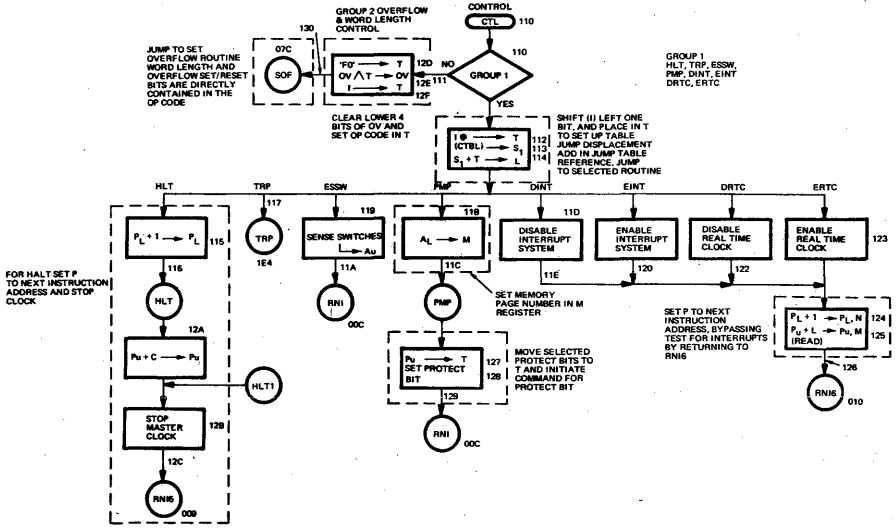


Figure 31. Control

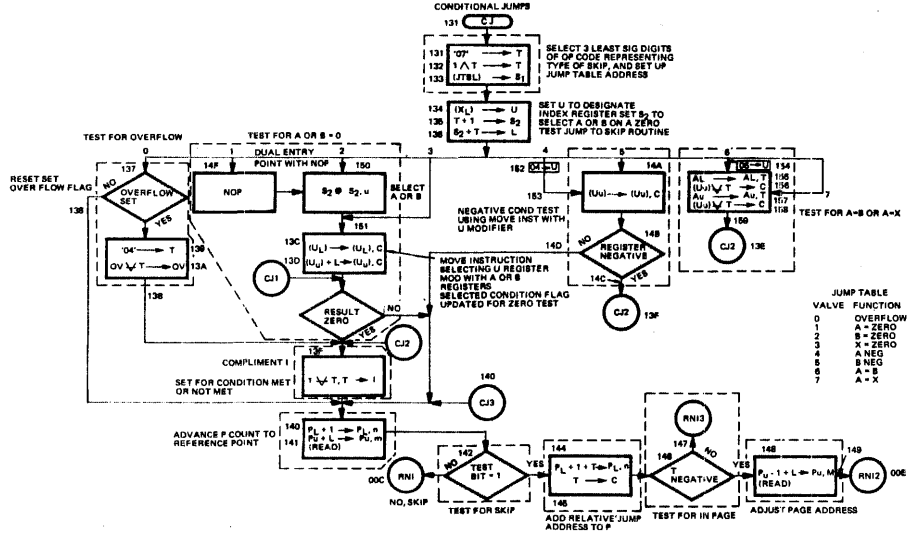


Figure 32. Conditional Jumps

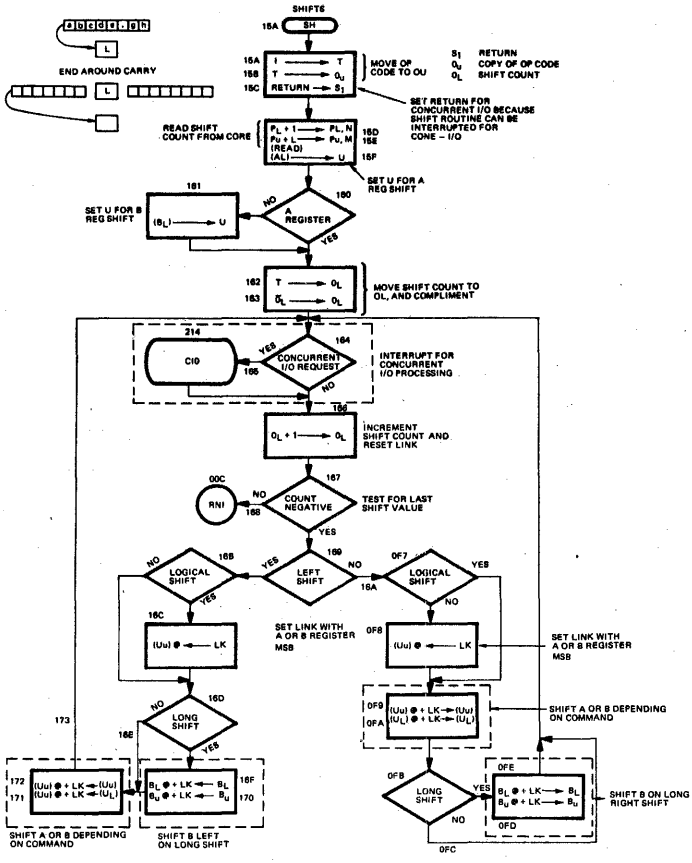


Figure 33. Shifts

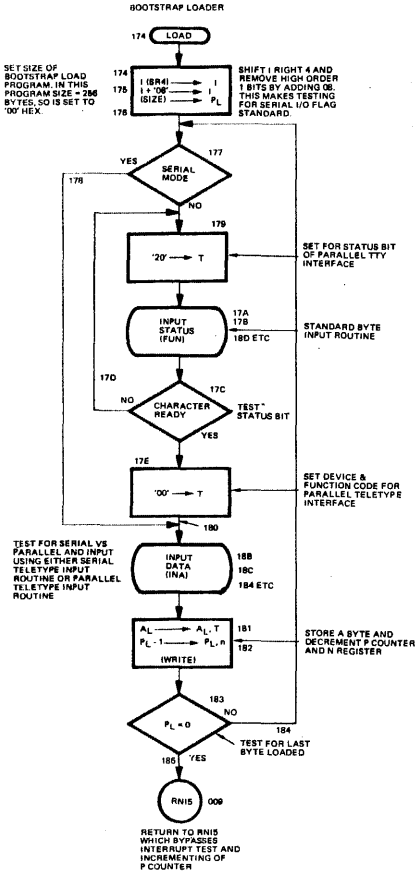


Figure 34. Bootstrap Loader

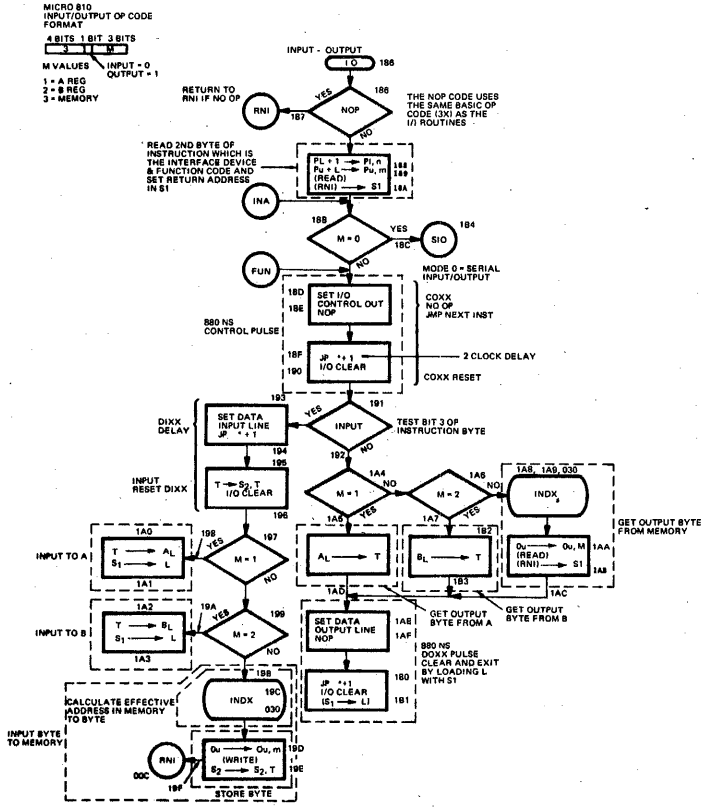


Figure 35. Input-Output

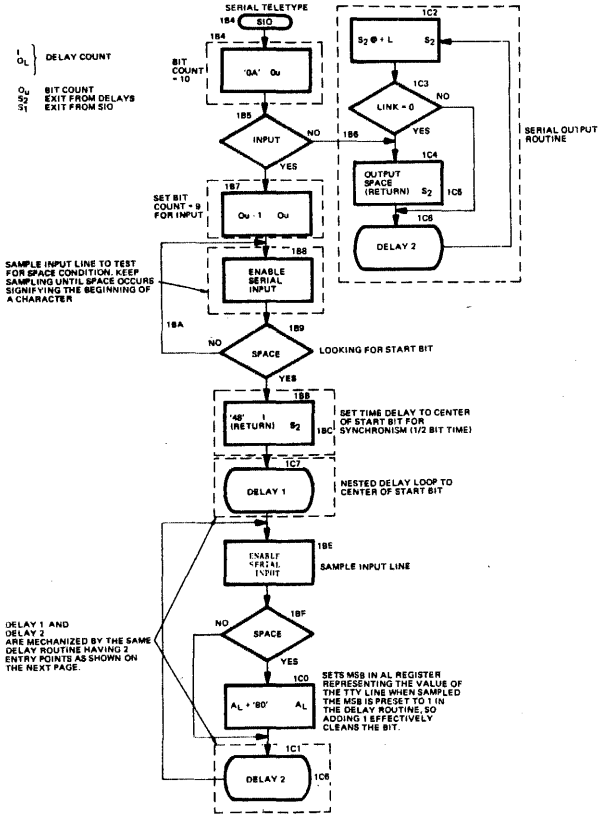


Figure 36. Serial Teletype

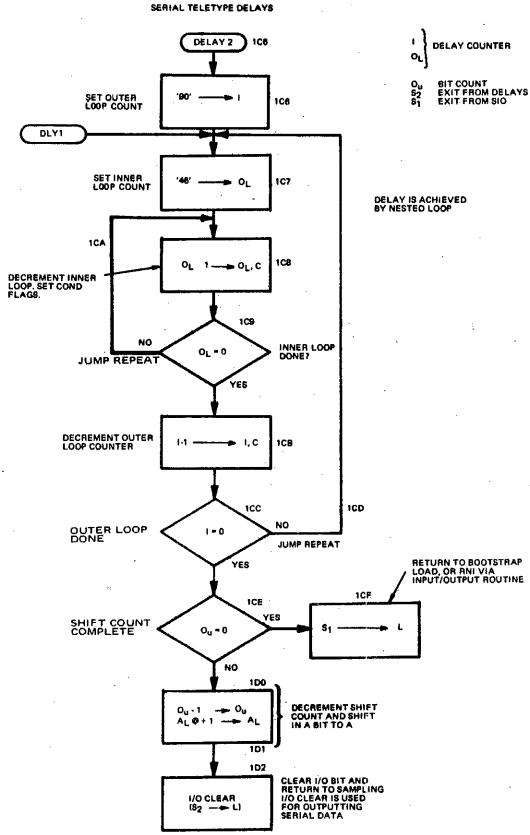


Figure 37. Serial Teletype Delays

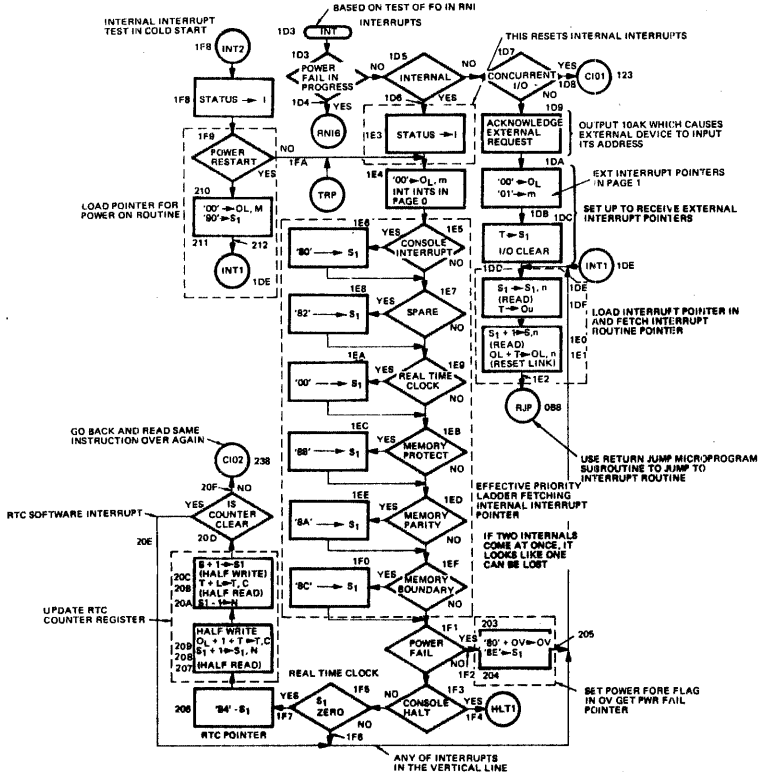


Figure 38. Interrupts

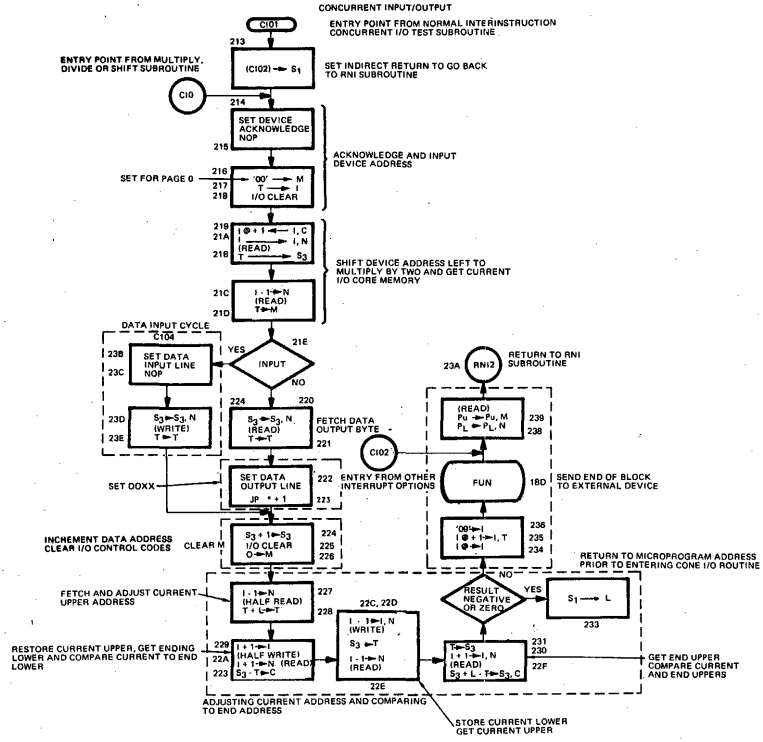


Figure 39. Concurrent Input/Output

MICRO 810 ASSEMBLY LISTINGS

The assembly language program with machine code and comments is included for reference from the flow charts. To illustrate the flow of micro commands for 810 operations, the dotted line flow is for a load A register direct relative address mode instruction.

Load A Direct Relative Address Mode

For this example, the op code in MICRO 810 machine language is:

```
0200    E1
0201    18
```

The E signifies load,

The 1 in binary is 0001

```
                  |   |
Fixed Word Length     Direct Relative
```

The 18 specifies a relative address 18 hex from the P count of the next instruction, which is 0202 + 18 = 021A.

In the RNI loop the op code, E1 is fetched and tested for memory reference. E 5 means memory reference. Therefore the operand address mode is entered. The 1 says direct relative, so the relative address 18 is fetched from core and added to 0202 and the result, 021A, is placed in the operand address register.

Then the microcommand jumps, via the jump table at 100, to the memory reference routine, entering at LDA. The 1 in the Op code signifies fixed word length (two bytes) so two bytes are fetched from core, starting at the location in the operand address register (021A) and placed in the A register. Then the microprogram returns to RNI to advance the P counter and fetch the next instruction.

The sequence of both of these examples can be seen by following the solid or dotted flow lines on the listing.

FUNCTION FLOW EXAMPLES OF A MICRO 810 INSTRUCTION

Load A direct relative

Machine Code of MICRO 810 Instruction Stored in Memory:

```
01FF    34      No op
0200    E1      Load A Dir. Rel.
0201    18      Rel. Address
```

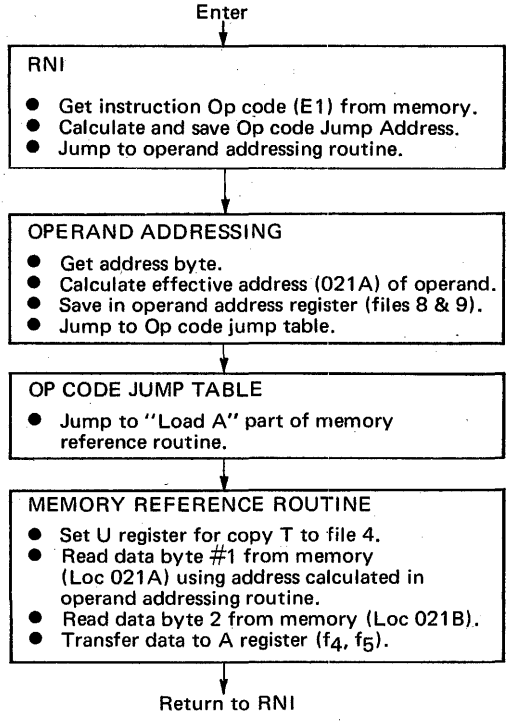
The instruction is located at P=0200 in core memory. For the example it is assumed that the previous instruction was a no Op, and there were no interrupts, or concurrent I/O requests. Therefore, the read next instruction routine will be entered at RNI.

The MICRO 810 instruction bit configuration is as follows:

E		1
1110	0	001
"Load" Op Code	Fixed Word Length	Mode 1 Direct Relative

The relative address '18' is a positive displacement. This instruction will cause a 16-bit number located at 021A to be loaded into the A register (files 4 and 5).

The basic functions (omitting tests and skips) for implementation of this instruction within the MICRO 800 are shown in the following flow chart:



The sequence of micro instructions is traced out in the following coding which was lifted from the MICRO 810 Firmware reference manual.

```
                        IDENT M810
                        *
                        *  MICRO 810 SYSTEM
                        *
                        *  FILE ALLOCATION
                0000    F0      EQU   0          CONDITION FLAGS
                0001    I       EQU   1          INSTRUCTION REGISTER
                0002    XL      EQU   2          INDEX REGISTER
                0003    XU      EQU   3
                0004    AL      EQU   4          ACCUMULATOR
                0005    AU      EQU   5
                0006    BL      EQU   6          EXTENDED ACCUMULATOR
                0007    BU      EQU   7
                0008    OL      EQU   8          OPERAND ADDRESS
                0009    OU      EQU   9
                000A    PL      EQU   10         PROGRAM COUNTER
                000B    PU      EQU   11
                000C    S1      EQU   12         TEMPORARY STORAGE
                000D    S2      EQU   13
                000E    S3      EQU   14
                000F    OV      EQU   15         OVERFLOW AND WORD LENGTH
                0001    F1      EQU   1          USED WITH EXECUTE FOR ODD FILE
                0000    SIZE    EQU   0          SIZE OF BASIC LOADER
                        *
                                ORG   0          BOARD 1
                        *
                        *  READ NEXT INSTRUCTION
            000 BF02    RNI0    CM    OV         CLEAR OV/W AND M REGISTERS
            001 2800            LF    PU,X'00'   CLEAR P COUNTER UPPER
            002 2A00            LF    PL,X'00'   CLEAR P COUNTER LOWER
            003 4010            TZ    F0,X'10'   INTERNAL INTERRUPT
            004 15F8            JP    INT2       YES, JUMP TO INTERRUPT ROUTINE
            005 7110            K     I,1        ENTER SENSE SWITCHES
            006 4180            TZ    I,X'80'    SWITCH 4 ON
            007 1574            JP    LOAD       YES, LOAD BOOT STRAP PROGRAM
            008 2F00    RNI1    LF    OV,X'00'   CLEAR OV/W REGISTER
EXAMPLE     009 C802    RNI5    MM    PU         MOVE P UPPER TO M REGISTER
            00A AA03    RNI4    RN    PL         GET OP CODE (FIRST BYTE OF INSTRUCTION)
            00B 1410            JP    RNI6       IGNORE INTERRUPTS (FOR SOME INSTRUCTIONS)
LOAD A      00C 8A43    RNI     IN    PL         UPDATE P BY INCREMENTING IT
DIRECT      00D A882    RNI3    RM    PU,L       FETCH INSTRUCTION BYTE
RELATIVE    00E 4098    RNI2    TZ    F8,X'98'   TEST FOR INTERRUPTS
            00F 15D3            JP    INT        SERVICE REQUEST BY JUMP TO INT, ROUTINE
            010 8120    RNI6    C     I,T        SAVE OP CODE STILL INT AFTER FETCH
            011 2C10            LF    S1,OTAB+16 BASE ADDR+16 TO CLEAR ONES IN SHIFTED OP
            012 7129            KT=   I,2        SHIFT RIGHT 4
            013 8C20            A     S1,T       ADD BASE ADDRESS TO SHIFTED OP
            014 61A0            CP    I,X'A0'    MEMORY REFERENCE IF OP .GT. 5F
            015 CC05            MK    S1         NO, GO DIRECTLY TO JUMP TABLE
                        *  YES, GET OPERAND ADDRESS
                        *
                        *  OPERAND ADDRESSING
            016 B901    ADDR    CT    OU         CLEAR OU AND T
            017 4104            TZ    I,X'04'    M .LT. 4 (FIRST 4 ADDRESSING MODES)
            018 142E            JP    ADR4       NO, MODE .GT. 4
            019 8A43            IN    PL         GET ADDRESS BYTE FOR PAGE ZERO OR RELATIVE
            01A A882            RM    PU,L
            01B 8833            CN    OL,T,C     SET CONDITION CODE FOR SIGN OF DISPLACEMENT
            01C 5101            TN    I,X'01'    PAGE ZERO ADDRESS MODE
            01D 1424            JP    ADR2       YES, JUMP TO INDIRECT TEST
            01E 8A69            AT=   PL,I,T     ADD RELATIVE VALUE
            01F 8823            CN    OL,T       TRANSFER RELATIVE VALUE TO OL AND N
            020 4002            TZ    F0,X'02'   DISPLACEMENT NEGATIVE (C SET AT 018)
            021 142C            JP    ADR3       YES, JUMP TO NEG. DISPLACEMENT CALCULATION
            022 8889            AT=   PU,L       ADD CARRY FOR PAGE BOUNDARY
            023 8920    ADR1    C     OU,T       TRANSFER RESULT TO OU
            024 5102    ADR2    TN    I,X'02'    INDIRECT ADDRESS MODE
            025 CC05            MK    S1         NO, EXIT TO JUMP TABLE
        (B) 026 A902            RM    OU         READ UPPER BYTE OF INDIRECT ADDRESS
            027 8840            I     OL         ADVANCE POINTER TO LOWER BYTE
            028 8982            AM    OU,L
            029 8920            C     OU,T       GET UPPER ADDRESS BYTE (READ AT 026)
            02A A803            RN    OL         READ LOWER BYTE OF INDIRECT ADDRESS
            02B 1439            JP    IND1       GO CHECK FOR POST INDEXING
            02C 9889    ADR3    ST=   PU,L       BORROW FROM UPPER ADDRESS
            02D 1423            JP    ADR1       GO TO INDIRECT ADDRESS ROUTINE
            02E 5103    ADR4    TN    I,X'03'    M .EQ. 4  INDEX MODE
            02F 1442            JP    ADR7       YES, GO TO INDEX FUNCTION
            030 8A43    INDX    IN    PL         ADVANCE P COUNTER
            031 A882            RM    PU,L       GET 2ND BYTE OF INSTRUCTION FROM CORE
            032 5102            TN    I,X'02'    M .EQ. 5  INDEXED WITH BIAS
            033 1442            JP    ADR7       YES
            034 4101            TZ    I,X'01'    M .EQ. 6  EXTENDED ADDRESS
            035 1444            JP    LIT        NO
            036 8920    ADR5    C     OU,T       GET UPPER ADDRESS BYTE (READ AT 031)
            037 8A43            IN    PL         ADVANCE P COUNTER
            038 A882            RM    PU,L       GET 3RD BYTE FROM CORE
            039 8823    IND1    CN    OL,T       TRANSFER 3RD BYTE TO OL
            03A 5980            TN    OU,X'80'   INDEXED (BIT 15 .EQ. 1)
            03B CC05            MK    S1         NO, EXIT
            03C 3980            AF    OU,X'80'   REMOVE BIT BY CARRY OUT, LEAVING A ZERO
            03D C201    ADR6    MT    XL         ADD X TO ADDRESS FOR INDEXING
            03E 8823            AN    OL,T       MOVE X INTO OPERAND ADDRESS REGISTER
            03F C301            MT    XU
            040 89A0            A     OU,L,T
            041 CC05            MK    S1         EXIT TO JUMP TABLE
            042 8820    ADR7    C     OL,T       GET BIAS (T .EQ. 0, WHEN M .EQ. 4)
            043 143D            JP    ADR8
            044 6190    LIT     CP    I,X'98'    JMP,RTJ,IBM, OR OBM (TEST NON LITERAL MODE)
            045 1436            JP    ADR5       YES
            046 CA01            MT    PL         LITERAL MODE
            047 8823            CN    OL,T                  } MOVE P TO OPERAND
            048 CB01            MT    PU                    } ADDRESS REGISTER
            049 8920            C     OU,T
            04A 6160            CP    I,X'60'    FIXED WORD LENGTH INSTRUCTION
            04B 1453            JP    ADR9       YES
            04C 5108            TN    I,X'08'    VARIABLE WORD LENGTH MODE
            04D 1453            JP    ADR9       YES
```

```
                04E  1103                LT    X'03'          SET MASK TO SELECT WORD LENGTH
                04F  EF29                NT*   OV,T           WORD LENGTH TO T REGISTER
                050  8A20          ADR8  A     PL,T           ADJUST P FOR NEXT INSTRUCTION
                051  8B80                A     PU,L
                052  CC05                MK    S1             EXIT TO JUMP TABLE
                053  1101          ADR9  LT    X'01'          1 TO T FOR ADDING 1 TO P
                054  1450                JP    ADR8           WITH FIXED WORD LENGTH TYPE
                                   *
                                   *  MEMORY REFERENCE
                055  16B4          LDA   LU    X'B4'          SET U WITH LOAD (COPY) OP CODE
                056  145C                JP    MR1            GO TO READ OPERANDS
                057  16E4          ANA   LU    X'E4'          SET U WITH LOGICAL AND OP CODE
                058  145C                JP    MR1            GO TO READ OPERANDS
                059  1694          SBA   LU    X'94'          SET U WITH SUBTRACT OP CODE
                05A  145C                JP    MR1            GO TO READ OPERANDS
                05B  1684          ADA   LU    X'84'          SET U WITH ADD OP CODE
                05C  A902          MR1   RM    OU             READ BYTE FROM MEMORY
                05D  5108                TN    I,X'08'        VARIABLE WORD LENGTH
                05E  1461                JP    MR2            NO, (FIXED LENGTH OPERANDS)
                05F  5F01                TN    OV,X'01'       W .EQ 0 OR 1 (2 BYTES MAXIMUM)
                060  1464                JP    MR3            YES
                061  BC20          MR2   C     S1,T           GET AN OPERAND
                062  8843                IN    OL             ADVANCE OPERAND ADDRESS AND
                063  A982                RM    OU,L           READ NEXT BYTE FROM MEMORY
                064  8000          MR3   A     F0             RESET LINK FOR COPY (LOAD) FUNCTION
                065  5108                TN    I,X'08'        VARIABLE WORD LENGTH
                066  1480                JP    MR8            NO
                067  5F02                TN    OV,X'02'       W .LT. 2 (2 BYTES MAXIMUM)
                068  147E                JP    MR7            YES
                069  BD20                C     S2,T           GET AN OPERAND  } FETCH 2ND AND 3RD
                06A  8843                IN    OL             }               } OR 3RD AND 4TH
                06B  A982                RM    OU,L           }               } OPERANDS DEPENDING
                06C  BE20                C     S3,T           GET AN OPERAND  } ON WORD LENGTH
                06D  8843                IN    OL             }               }
                06E  A982                RM    OU,L           }               }
                06F  8000                A     F0             RESET LINK FOR COPY (LOAD) FUNCTION
                070  0620                E     BL,2           OPERATE ON BL (FUNCTION IN U)
                071  CE01                MT    S3             MOVE OPERAND TO T
                072  07A0                E     BU,10          OPERATE ON BU (FUNCTION IN U)
                073  CD01                MT    S2             MOVE OPERAND TO T
                074  00B0                E     F0,11          OPERATE ON AL
                075  5F01          MR4   TN    OV,X'01'       W .EQ. 0 OR 2 (1 OR 2 BYTES)
                076  1482                JP    MR9            YES
                077  CC01          MR5   MT    S1             MOVE OPERAND TO T
                078  01B0                E     F1,11          OPERATE ON AU OR XU (FUNCTION IN U)
                079  5001          OCK   TN    F0,X'01'       OVERFLOW SET
RETURN          07A  140C                JP    RNI            NO
TO RNI          07B  1104          SET   LT    X'04'          SET MASK
                07C  CF20                SOF   O              SET BIT IN OV
                07D  140C                JP    RNI
                07E  0030          MR7   E     F0,3           OPERATE ON AL (FUNCTION IN U)
                07F  1475                JP    MR4
                080  0020          MR8   E     F0,2           OPERATE ON AL OR XL (FUNCTION IN U)
                081  1477                JP    MR5
                082  2500          MR9   LF    AU,X'00'       CLEAR AU            } SIGN EXTENSION
                083  4480                TZ    AL,X'80'       RESULT POSITIVE     } FOR VARIABLE
                084  C560                O     AU,T,F         FF TO AU            } WORD LENGTH TYPE
                085  1479                JP    OCK
                086  16B2          LDX   LU    X'B2'          SET U WITH LOAD X (COPY) OP CODE
                087  5108                TN    I,X'08'        STORE
                088  145C                JP    MR1            NO, GO READ OPERANDS
                089  16A2                LU    X'A2'          SET U WITH STORE X OP CODE
                08A  2100          ST4   LF    I,X'00'        CLEAR I FOR STORE OPERATION
                08B  148D                JP    ST1            GO STORE OPERANDS
                08C  16A4          STA   LU    X'A4'          SET U WITH STORE A OP CODE
                08D  C902          ST1   MM    OU
                08E  5108                TN    I,X'08'        VARIABLE
                08F  1492                JP    ST2            NO
                090  5F01                TN    OV,X'01'       W .EQ. 0 OR 2
                091  1495                JP    ST3            YES
                092  0111          ST2   ET    F1,1           STORE UPPER BYTE USING EXECUTE WITH U MOD.
                093  8843                IN    OL             INCREMENT OPERAND ADDRESS REGISTER TO 2ND
                094  8982                AM    OU,L           OPERAND BYTE
                095  0011          ST3   ET    F0,1           STORE LOWER BYTE
                096  5108                TN    I,X'08'        VARIABLE
                097  140C                JP    RNI            NO
                098  5F02                TN    OV,X'02'       W .EQ. 0 OR 1
                099  140C                JP    RNI            YES
                09A  16A6                LU    X'A6'          SET U WITH STORE R OP CODE
                09B  8843                IN    OL             INCREMENT OPERAND ADDRESS REGISTER
                09C  8980                A     OU,L
                09D  148A                JP    ST4            GO STORE R REGISTER
                09E  1600          IWM   LU    X'00'          CLEAR U
                09F  CD60                O     S2,T,F         SET FOR DECREMENT
                0A0  5108                TN    I,X'08'        TEST FOR INCREMENT
                0A1  BD46                CU    S2,I           SET FOR INCREMENT
                0A2  884B                IN*   OL
                0A3  A9AA                RM*   OH,L,H        HALF READ OPERAND TO T REGISTER
                0A4  8C29                AT*   S2,T           +1 OR -1 FOR INCREMENT OR DECREMENT
                0A5  AC77                WS    S1,D,H         WRITE AND DECR S2 IF AN INCREMENT WAS DONE
                0A6  C803                MN    OL
                0A7  A922                RM    OU,H           HALF READ UPPER BYTE TO T
                0A8  8DB1                AT    S2,L,T,C       ADD CARRY TO UPPER BYTE AND SET COND. FLG.
                0A9  A030                W     F0,H           HALF WRITE
                0AA  1479                JP    OCK            CHECK FOR OVERFLOW
                                   *
                                   *  JUMP AND RETURN JUMP
                0AB  D160          JMP   X     I,T,F          COMPLEMENT INSTRUCTION REGISTER
                0AC  4107                TZ    I,X'07'        M .EQ. 7  EXTENDED INDIRECT
                0AD  14B5                JP    JM1            NO
                0AE  A902                RM    OU             READ UPPER BYTE OF INDIRECT ADDRESS
                0AF  8840                I     OL             INCREMENT OPERAND
                0B0  8982                AM    OU,L           ADDRESS REGISTER
                0B1  B920                C     OU,T           GET HIGH BYTE WHICH IS IN T
                0B2  A803                RN    OL             READ LOWER BYTE OF INDIRECT ADDRESS
                0B3  2C18                LF    S1,PTR3        SET INDIRECT RETURN
                0B4  1439                JP    IND1           CHECK FOR POST INDEXING
```

```
0B5  4108      JM1   TZ    I,X'08'          RETURN JUMP
0B6  14BF            JP    JM2              NO
0B7  8A40            I     PL               ADJUST P FOR NEXT INSTRUCTION
0B8  8881      RJP   AT    PU,L             AFTER RTJ INSTRUCTION
0B9  A912            WM    OU               STORE PU          } STORE PROGRAM COUNTER
0BA  8843            IN    OL                                  AT FIRST TWO LOCATIONS
0BB  A992            WM    OU,L             STORE PL          } OF ROUTINE CALLED BY
0BC  CA01            MT    PL                                  RTJ, TRP, OR INTERRUPT
0BD  8840            I     OL                    SET OPERAND ADDRESS TO
0BE  8982            AM    OU,L                  FIRST INSTRUCTION IN
0BF  C901      JM2   MT    OU                    CALLED SUBROUTINE AND
0C0  8822            CM    PU,T                  PLACE THE VALUE INTO THE
0C1  C801            MT    OL                    PROGRAM COUNTER TO BEGIN
0C2  BA20            C     PL,T                  EXECUTION OF THE SUBROUTINE
0C3  140A            JP    RNI4             RETURN TO RNI
                *
                *  REGISTER OPERATE
0C4  9C01      REG   CT    S1               CLEAR T AND S2
0C5  FC66            MU    S1,I,R           LOAD U WITH ADD OP CODE (80)
0C6  4108            TZ    I,X'08'          GROUP1
0C7  14DE            JP    REG3             NO
0C8  4101            TZ    I,X'01'          SUB OR XOR INSTRUCTIONS
0C9  1610            LU    X'10'            YES
0CA  4104            TZ    I,X'04'          INDEX CONTROL INSTRUCTION
0CB  14D8            JP    REG2             YES
0CC  4102            TZ    I,X'02'          A REG DESTINATION INSTRUCTION
0CD  14D3            JP    REG1             NO
0CE  C601            MT    BL               B OR A TO A, USING U REG. MOD
0CF  C427            OS    AL,T                   OR
0D0  C701            MT    BU               B XOR A TO A, USING U REG. MOD
0D1  C527            OS    AU,T
0D2  140C            JP    RNI
0D3  C401      REG1  MT    AL               A OR B TO B, USING U REG. MOD
0D4  C427            OS    BL,T                   OR
0D5  C501            MT    AU               A XOR B TO B, USING U REG. MOD
0D6  C727            OS    BU,T
0D7  140C            JP    RNI
0D8  4102      REG2  TZ    I,X'02'          WORD LENGTH CONTROL
0D9  1103            LT    X'03'            YES, SET MASK FOR WORD LENGTH BITS
0DA  EF29            NT*   OV,T              WITH AND COMMAND
0DB  8267            AS    XL,I,T           ADD OR SUBTRACT WORD LENGTH, INCREMENT
0DC  8397            AS    XU,L,C           OR DECREMENT X (DEPENDING ON U REGISTER)
0DD  1479            JP    OCK              CHECK FOR OVERFLOW
0DE  4101      REG3  TZ    I,X'01'          B REGISTER TO BE MOVED OR MODIFIED
0DF  3C02            AF    S1,X'02'         YES
0E0  CC06            MU    S1               SET U WITH BASIC OP CODE
0E1  4104            TZ    I,X'04'          INTER REGISTER TRANSFERS
0E2  14EB            JP    RBQ5             YES
0E3  4102            TZ    I,X'02           COMPLEMENT A OR B REGISTER
0E4  14E8            JP    RBQ4             YES
0E5  0440            E     AL,4             ADD 1 TO INCREMENT A OR B
0E6  0590            E     AU,9             ADD CARRY TO UPPER BYTE
0E7  1479            JP    OCK              CHECK FOR OVERFLOW
0E8  D467      REG4  XS    AL,T,F           1'S COMPLEMENT A OR B REGISTER
0E9  D567            XS    AU,T,F
0EA  140C            JP    RNI
0EB  4102      REG5  TZ    I,X'02'          X REGISTER SOURCE FOR TRANSFER
0EC  14F2            JP    RBQ6             YES
0ED  0401            ET    AL               A OR B TO T       }
0EE  B220            C     XL,T             T TO X            } TRANSFER A OR B TO X
0EF  0501            ET    AU               A OR B TO T       } DEPENDING ON U REGISTER
0F0  B320            C     XU,T             T TO X            }
0F1  140C            JP    RNI
0F2  C201      REG6  MT    XL               X TO T            }
0F3  B427            CS    AL,T             T TO A OR B       } TRANSFER X TO A OR B
0F4  C301            MT    XU               X TO T            } DEPENDING ON U REGISTER
0F5  B527            CS    AU,T             T TO A OR B       }
0F6  140C            JP    RNI
                *
                *  RIGHT SHIFTS
0F7  4908      SR    TZ    OU,X'08          LOGICAL SHIFT
0F8  F10F            HS*   F1               NO, SET LINK WITH SIGN
0F9  F1A7            HS    F1,L,R           RIGHT 1           } SHIFT RIGHT SELECTED
0FA  F0A7            HS    F0,L,R           RIGHT 1           } REGISTER (A OR B)
0FB  9902            TN    OU,X'02'         LONG SHIFT
0FC  1564            JP    SH1              NO
0FD  F7A0            H     BU,L,R           RIGHT 1           } SHIFT RIGHT
0FE  F6A0            H     BL,L,R           RIGHT 1           } B REGISTER
0FF  1564            JP    SH1              REPEAT SHIFTS
                *
                      ORG   256              BOARD 2
                *
                *  OP CODE JUMP TABLE
100  1510      OTAB  JP    CTL              CONTROL
101  1531            JP    CJ               CONDITIONAL JUMPS
102  155A            JP    SH               SHIFTS
103  1586            JP    IO               INPUT/OUTPUT
104  14C4            JP    REG              REGISTER OPERATE
105  1C00            JP    SP.              SPARE INSTRUCTION OP CODE
106  14AB            JP    JMP              JUMP AND RETURN JUMP
107  149E            JP    IWM              INCREMENT AND DECREMENT MEMORY
108  1486            JP    LDX              LOAD AND STORE X
109  1C01            JP    MUL.             MULTIPLY/DIVIDE
10A  145B            JP    ADA              ADD
10B  1459            JP    SBA              SUBTRACT
10C  1C02            JP    CPA.             COMPARE
10D  1457            JP    ANA              AND
10E  1455            JP    LDA              LOAD A
10F  148C            JP    STA              STORE A
                *
                *  CONTROL
110  4108      CTL   TZ    I,X'08'          TEST FOR GROUP 1 OR 2
111  152D            JP    GP2              OVERFLOW AND WORD LENGTH EXIT
112  F109            MT*   I                SET UP JUMP TABLE VALUE
113  2C15            LF    S1,CTBL          ADD WITH BASE ADDRESS
114  8C25            AK    S1,T             TABLE JUMP
115  8A40      CTBL  I     PL               HALT (INCREMENT P COUNTER)
```

```
116  152A          JP    HLT          JMP TO HALT ROUTINE
117  15E4          JP    TRP          TRAP INSTRUCTION (SAME AS CONSOLE INT.)
118  1485    PTR3  JP    JM1          IND FROM ADDR TO JUMP (NOT PART OF CONTROL)
119  7510          K     AU,1         ENTER SENSE SWITCHES
11A  140C          JP    RNI
11B  C402          MM    AL           PROTECT MEMORY PAGE
11C  1527          JP    PMP
11D  1704          LS    X'04'        DISABLE INTERRUPT SYSTEM
11E  1524          JP    EC1
11F  1708          LS    X'08'        ENABLE INTERRUPT SYSTEM
120  1524          JP    EC1
121  1710          LS    X'10'        DISABLE REAL TIME CLOCK
122  1524          JP    EC1
123  1720          LS    X'20'        ENABLE REAL TIME CLOCK
124  8A43    EC1   IN    PL           SET P TO NEXT INSTRUCTION ADDRESS
125  AB82          RM    PU,L         AND FETCH INSTRUCTION BYTE
126  1410          JP    RNI6         BY PASS INTERRUPT CHECK
127  C701    PMP   MT    BU           SELECTED PROTECT BITS TO T
128  1740          LS    X'40'        SET PROTECT STATUS
129  140C          JP    RNI
12A  8B80    HLT   A     PU,L         ADD CARRY TO ADJUST P UPPER FOR NEXT INSTR.
12B  1780    HLT1  LS    X'80'        STOP CLOCK
12C  1409          JP    RNI5
12D  11F0    GP2   LT    X'F0'        SET MASK (TO SAVE UPPER HALF OF OV/W)
12E  EF20          N     OV,T         CLEAR OV/W STATUS
12F  C101          MT    I            PUT OV/W SETTING INTO T
130  147C          JP    SOF          GO SET NEW STATUS FOR OV/W
                *
                *  CONDITIONAL JUMPS
131  1107    CJ    LT    X'07'        MASK FOR CONDITION
132  E129          NT*   I,T          REMOVE OP CODE
133  2C4E          LF    S1,JTBL      BASE TABLE ADDRESS
134  1602          LU    X'02'        SET FOR X REGISTER
135  BC60          C     S2,I,T       SET TO SELECT A, OR B ON ZERO TEST
136  8C25          AK    S1,T         DO A TABLE JUMP
137  5F04    J0    TN    OV,X'04'     OVERFLOW TEST
138  1540          JP    CJ3          NO
139  1104          LT    X'04'        OVERFLOW RESET BIT TO T
13A  DF20          X     OV,T         RESET OVERFLOW BY TOGGLING
13B  153F          JP    CJ2
13C  C017    J3    MS    F0,C         TEST LOW BYTE   } TEST A OR B WITH
13D  C197          MS    F1,L,C       TEST HIGH BYTE  } LINKED ZERO TEST
13E  4004    CJ1   TZ    F0,X'04'     RESULT ZERO
13F  D160    CJ2   X     I,T,F        YES, FLIP TEST BIT BY COMPLEMENT
140  8A43    CJ3   IN    PL           GET DISPLACEMENT WHICH IS 2ND
141  AB82          RM    PU,L         BYTE OF INSTRUCTION
142  5108          TN    I,X'08'      CONDITION MET
143  140C          JP    RNI          NO
144  8A63          AN    PL,I,T       ADD DISPLACEMENT
145  B030          C     F0,T,C       LOOK AT T
146  5002          TN    F0,X'02'     T NEGATIVE
147  140D          JP    RNI3         NO
148  AB42          RM    PU,D         ADJUST PAGE IF BOUNDARY CROSSED
149  140E          JP    RNI2
14A  C117    J5    MS    F1,C         LOOK AT AU OR XU FOR SIGN TEST
14B  4002          TZ    F0,X'02'     NEGATIVE
14C  153F          JP    CJ2          YES
14D  1540          JP    CJ3          NO
                *
                *  CONDITIONAL JUMP TABLE
14E  1537    JTBL  JP    J0           OVERFLOW
14F  1000          L     X'00'        NOP
150  FD06          HU    S2           SET FOR A OR B
151  153C          JP    J3
152  1604          LU    X'04'        SET FOR A
153  154A          JP    J5
154  1606          LU    X'06'        SET FOR B
155  C401    J7    MT    AL
156  D03F          XS*   F0,T,C       COMPARE LOWER   } TEST FOR A=B OR A=X
157  C501          MT    AU                          } DEPENDING ON U REG.
158  D1BF          XS*   F1,L,T,C     COMPARE UPPER   }
159  153E          JP    CJ1          TEST RESULT OF COMPARISON
                *
                *  SHIFTS
15A  C101    SH    MT    I            SAVE OP CODE IN OPERAND
15B  8920          C     OU,T         ADDRESS REGISTER
15C  2C66          LF    S1,SH2       SET ADDR FOR CONCURRENT I/O TEST
15D  8A43          IN    PL           GET SHIFT COUNT (2ND BYTE IN INSTRUCTION)
15E  AB82          RM    PU,L
15F  1604          LU    X'04'        SET U FOR SHIFTING A REGISTER
160  4901          TZ    OU,X'01'     TEST FOR A OR B SHIFT
161  1606          LU    X'06'        SET U FOR SHIFTING B REGISTER
162  8820          C     OL,T         MOVE SHIFT COUNT TO OL
163  D860          X     OL,T,F       AND COMPLEMENT IT FOR LOOP CONTROL.
164  4008    SH1   TZ    F0,X'08'     CONCURRENT I/O REQUEST
165  1C14          JP    C10          YES (SERVICE CONC I/O DURING SHIFT)
166  8840    SH2   I     OL           ADD 1 TO COUNT AND RESET LINK
167  5880          TN    OL,X'80'     COUNT NEGATIVE
168  140C          JP    RNI          NO
169  4904          TZ    OU,X'04'     LEFT SHIFT
16A  14F7          JP    SR           NO, JUMP TO RIGHT SHIFT ROUTINE
16B  5908          TN    OU,X'08'     LOGICAL SHIFT
16C  F10F          HS*   F1          YES, SET LINK WITH LOW ORDER BIT
16D  5902          TN    OU,X'02'     LONG SHIFT
16E  1571          JP    SL1          NO
16F  F680          H     BL,L         LEFT 1          } SHIFT B LEFT FOR
170  F780          H     BU,L         LEFT 1          } LONG SHIFT
171  F08F    SL1   HS    F0,L         LEFT 1          } SHIFT A, OR B LEFT,
172  F187          HS    F1,L         LEFT 1          } DEPENDING ON U
173  1564          JP    SH1          REPEAT SHIFTS
                *
                *  BOOTSTRAP LOADER
174  7120    LOAD  K     I,2          SHIFT RIGHT (RIGHT JUSTIFY OP CODE)
175  3108          AF    I,X'08'      REMOVE BITS BY CAUSING CARRY ON UPPER BITS
176  2A00          LF    PL,SIZE      SET LOADER SIZE .EQ. 256
177  5101    LOD1  TN    I,X'01'      SERIAL MODE
178  157F          JP    LOD3         YES
179  1120    LOD5  LT    X'20'        SET FOR STATUS IN
```

```
17A  2C7C          LF   S1,LOD2        SET RETURN
17B  1580          JP   FUN            GET STATUS
17C  5402    LOD2  TN   AL,X'02'       CHARACTER READY
17D  1579          JP   LOD5           NO
17E  1100          LT   X'00'          SET FOR DATA IN
17F  2C81    LOD3  LF   S1,LOD4        SET RETURN
180  158B          JP   INA            GET DATA
181  C401    LOD4  MT   AL             SET DATA IN T
182  AA53          WN   PL,D           STORE BYTE
183  4AFF          TZ   PL,X'FF'       DONE LOADING
184  1577          JP   LOD1           NO
185  1409          JP   RNI5           YES
                   *
                   *  INPUT-OUTPUT
186  4104    IO    TZ   I,X'04'        NOP
187  140C          JP   RNI            YES
188  8443          IN   PL             GET DEVICE ADDRESS WHICH IS
189  AB82          RM   PU,L           SECOND BYTE OF INSTRUCTION
18A  2CA3          LF   S1,IOK5        RETURN TO RNI
18B  5103    INA   TN   I,X'03'        SERIAL MODE
18C  1584          JP   SIO            YES
18D  7090    FUN   K    F0,9          CONTROL OUT          }
18E  1000          L    X'00'         NOP                  } COXX CONTROL
18F  1590          JP   IO1           STROBE               } STROBE
190  7080    IO1   K    F0,8          CLEAR
191  4108          TZ   I,X'08'        INPUT
192  1544          JP   OUT            NO
193  70E0          K    F0,14         DATA IN              }
194  1595          JP   IO2           STROBE               } DIXX INPUT
195  BD21    IO2   CT   S2,T          GET DATA             } STROBE
196  7080          K    F0,8          CLEAR                }
197  5102          TN   I,X'02'       M .EQ. 1             } TEST FOR INPUT TO A
19A  15A0          JP   IO4           YES                  } OR INPUT TO B
199  5101          TN   I,X'01'       M .EQ. 2             } OR INPUT TO MEMORY
19A  15A2          JP   IO5
19B  2C9D          LF   S1,IO3
19C  1430          JP   INDX          GET STORE ADDRESS    } SET ADDR FOR INPUT TO
19D  A912    IO3   WM   OU                                 } MEMORY AND STORE BYTE
19E  CC01          MT   S2            STORE BYTE           }
19F  140C          JP   RNI
1A0  B420    IO4   C    AL,T          PUT BYTE IN A
1A1  CC05    IO4A  MK   S1
1A2  B620    IO5   C    BL,T          PUT BYTE IN B
1A3  140C    IOK5  JP   RNI
1A4  5102    OUT   TN   I,X'02'       M .EQ. 1             } TEST FOR OUTPUT FROM
1A5  15AD          JP   IO7           YES                  } A, B, OR MEMORY
1A6  5101          TN   I,X'01'       M .EQ. 2             }
1A7  15B2          JP   IO10          YES
1A8  2CAA          LF   S1,IO6
1A9  1430          JP   INDX          GET OUTPUT ADDRESS   } FETCH OUTPUT
1AA  A902    IO6   RM   OU                                 } BYTE FROM MEMORY
1AB  2C9F          LF   S1,IO4-1      SET RETURN
1AC  15AE          JP   IO8
1AD  C401    IO7   MT   AL             A TO T
1AE  70A0    IO8   K    F0,10         OUTPUT               }
1AF  1000          L    X'00'         NOP                  } DOXX OUTPUT
1B0  15B1          JP   IO9           STROBE               } STROBE
1B1  7C85    IO9   KK   S1,8          CLEAR AND EXIT       }
1B2  C601    IO10  MT   BL             B TO T
1B3  15AE          JP   IO8
                   *
                   *  SERIAL TELETYPE
1B4  290A    SIO   LF   OU,X'0A'       SET BIT COUNT
1B5  4108          TZ   I,X'08'        INPUT
1B6  15C4          JP   SOUT           NO
1B7  9940          D    OU             ADJUST BIT COUNT FOR INPUT SAMPLING
1B8  1701    SIO1  LS   X'01'          ENABLE SERIAL TTY (INPUT A SAMPLE)
1B9  5040          TN   F0,X'40'       START BIT
1BA  15B8          JP   SIO1           NO, REPEAT SAMPLE
1BB  2148          LF   I,X'48'        SET DELAY COUNT (220 MS)
1BC  2DBE          LF   S2,SIO1        SET DELAY RETURN
1BD  15C7          JP   DLY1
1BE  1701    SIO1  LS   X'01'          ENABLE SERIAL TTY
1BF  4040          TZ   F0,X'40'       SPACE
1C0  3480          AF   AL,X'80'       YES, REMOVE BIT
1C1  15C6          JP   DLY2           GO, DELAY
1C2  FD80    SIOO  W    S2,L           GET LINK BIT
1C3  5D01          TN   S2,X'01'       CURRENT BIT, A ZERO
1C4  7080    SOUT  K    F0,11          YES, SPACE
1C5  2DC2          LF   S2,SIOO        SET DELAY RETURN
1C6  2190    DLY2  LF   I,X'90'        SET DELAY COUNT (220 MS)
1C7  2846    DLY1  LF   OL,X'46'
1C8  9850    DLI   D    OL,C           REDUCE LOW COUNTER
1C9  5004          TN   F0,X'04'       COUNTER ZERO
1CA  15C8          JP   DL1            NO
1CB  9150          D    I,C            REDUCE UPPER COUNTER
1CC  5004          TN   F0,X'04'       COUNTER ZERO
1CD  15C7          JP   DLY1           NO
1CE  590F          TN   OU,X'0F'       BIT COUNTER ZERO
1CF  CC05          MK   S1             YES, EXIT
1D0  9940          D    OU             REDUCE BIT COUNTER
1D1  F460          H    AL,I,R         SHIFT LOW BIT TO LINK
1D2  7C85          KK   S2,8           CLEAR AND EXIT (MARK)
                   *
                   *  INTERRUPTS
1D3  4F80    INT   TZ   OV,X'80'       POWER FAIL IN PROGRESS
1D4  1410          JP   RNI6           YES
1D5  4010          TZ   F0,X'10'       INTERNAL
1D6  15E3          JP   INT0           YES
1D7  4008          TZ   F0,X'08'       CONCURRENT I/O
1D8  1C13          JP   CIO1           YES
1D9  7D00    EXT   K    F0,13         ACKNOWLEDGE          }
1DA  2800          LF   OL,X'00'       CLEAR OL             } IOAK INTERRUPT
1DB  1201          LM   X'01'          SET FOR PAGE 1       } ACKNOWLEDGE
1DC  BC20          C    S1,T           GET ADDRESS          } STROBE
1DD  7080          K    F0,8           CLEAR                }
1DE  AC03    INT1  RN   S1
1DF  B920          C    OU,T           GET UPPER ADDRESS
```

```
1E0   ACC3            RN    S1,I
1E1   8823            AN    OL,T              GET LOWER ADDRESS AND RESET LINK
1E2   14B8            JP    RJP               DO A RETURN JUMP
1E3   7140    INTO    K     I,4               GET INTERNAL STATUS
1E4   B802    TRP     CM    OL                CLEAR OL AND M
1E5   4101    INT3    TZ    I,X'01'           CONSOLE INTERRUPT OR TRAP
1E6   2C80            LF    S1,X'A0'          YES
1E7   4102            TZ    I,X'02'           SPARE
1E8   2C82            LF    S1,X'A2'          YES
1E9   4104            TZ    I,X'04'           REAL TIME CLOCK
1EA   2C00            LF    S1,X'00'          YES
1EB   4108            TZ    I,X'08'           MEMORY PROTECT
1EC   2C88            LF    S1,X'88'          YES
1ED   4110            TZ    I,X'10'           MEMORY PARITY
1EE   2C8A            LF    S1,X'8A'          YES
1EF   4120            TZ    I,X'20'           MEMORY BOUNDARY
1F0   2C8C            LF    S1,X'8C'          YES
1F1   4180            TZ    I,X'80'           POWER FAIL
1F2   1C03            JP    PWRF              YES
1F3   4140            TZ    I,X'40'           CONSOLE HALT
1F4   152B            JP    HLT1              YES
1F5   4C80            TZ    S1,X'80'          REAL TIME CLOCK
1F6   15DE            JP    INT1              NO
1F7   1C06            JP    INT4
1F8   7140    INT2    K     I,4               GET INTERNAL STATUS
1F9   5180            TN    I,X'80'           POWER RESTART
1FA   15E4            JP    TRP               NO
1FB   1C10            JP    INT5
                *
                *  INDIRECT POINTERS
1FC   1C38    PTR4    JP    CIO2              INDIRECT FROM CIO OR IO TO CIO2
1FD   1C7E    PTR1    JP    MUL3              INDIRECT FROM CIO TO MULTIPLY
1FF   1C9B    PTR2    JP    DIV3              INDIRECT FROM CIO TO DIVIDE
                *
                        ORG   512               BOARD 3
                *
                *  SECONDARY OP CODE TABLE
200   1780    SP.     LS    X'80'             SPECIAL (ERROR HALT)
201   1C62    MUL.    JP    MUL               MULTIPLY/DIVIDE
202   1C40    CPA.    JP    CPA               COMPARE
                *
                *  INTERRUPT OPTIONS  (PWFL/RESTART AND RTC)
203   3F80    PWRF    AF    OV,X'80'          SET FLAG FOR POWER FAIL
204   2C8E            LF    S1,X'8E'
205   15DE            JP    INT1
206   2C84    INT4    LF    S1,X'84'          SET COUNTER ADDRESS
207   ACE3            RN    S1,I,H            GET LOWER HALF OF COUNTER
208   8879            AT*   OL,I,T,C           ADD 1 AND SET COND CODE
209   A030            W     F0,H              PUT BACK
20A   AC68            RN*   S1,D,H            GET UPPER HALF OF COUNTER
20B   8891            CT    F0,L,T,C          ADD CARRY AND SET COND CODE
20C   ACF0            W     S1,I,H            PUT BACK
20D   4004            TZ    F0,X'84'          COUNTER ZERO
20E   15DE            JP    INT1              YES, GO TO SERVICE ROUTINE
20F   1C38            JP    CIO2              NO, GO RE-FETCH INSTRUCTION
210   2800    INT5    LF    OL,X'50'          CLEAR OL
211   2C90            LF    S1,X'90'          SET ADDRESS
212   15DE            JP    INT1
                *
                *  CONCURRENT INPUT-OUTPUT
213   2CFC    CIO1    LF    S1,PTR4           INDIRECT RETURN ADDRESS FROM
                *                               CONCURRENT I/O, ENTERED
                *                               FROM NORMAL INTERRUPT/CONC I/O
                *                               TEST ROUTINE
214   70D0    CIO     K     F0,13             ACKNOWLEDGE REQUEST
215   1000            L     X'00'             NOP                          } CONCURRENT
216   1200            LM    X'00'             SET FOR PAGE ZERO            } I/O ACKNOWLEDGE
217   B120            C     I,T               GET ADDRESS                  } STROBE
218   7080            K     F0,8              CLEAR
219   F150            W     I,I,C             ADJUST AND REMOVE I/O FLAG BY SHIFTING
21A   A103            RN    I
21B   8E20            C     S3,T              GET CURRENT ADDRESS LOWER
21C   A148            RN*   I,D
21D   B022            CM    F0,T              GET CURRENT ADDRESS UPPER
21E   5001            TN    F0,X'01'          INPUT (TEST OVERFLOW COND. FLAG.)
21F   1C38            JP    CIO4              YES
220   AE03            RN    S3                READ OUTPUT BYTE FROM MEMORY
221   B020            C     F0,T              WAIT FOR DATA (DELAY)
222   70A0            K     F0,10             OUTPUT                       } DOXX STROBE FOR
223   1C24            JP    CIO3              DELAY                        } CONCURRENT OUTPUT
224   9E40    CIO3    I     S3                ADJUST CURRENT LOWER         } CONCURRENT OUTPUT
225   7080            K     F0,8              CLEAR
226   1200            LM    X'00'             SET FOR PAGE ZERO (CONC I/O POINTER)
227   A16B            RN*   I,D,H            GET CURRENT ADDRESS UPPER
228   B0A1            CT    F0,L,T            ADJUST (ADD CARRY)
229   A1F0            W     I,I,H             PUT BACK
22A   A1CB            RN*   I,I              GET ENDING LOWER
22B   9E38            S*    S3,T,C            COMPARE LOW BYTES
22C   A153            WN    I,D               STORE CURRENT LOWER
22D   CE01            MT    S3
22E   A148            RN*   I,D               GET CURRENT UPPER
22F   8E20            C     S3,T
230   A1C3            RN    I,I               GET ENDING UPPER
231   9EB0            S     S3,L,T,C          COMPARE HIGH BYTES
232   4006            TZ    F0,X'06'          RESULT .LT. 0 (LINKED ZERO TEST)
233   15A1            JP    I04A              GET TO SECOND PAGE TO EXIT
234   F120            W     I,R               ADJUST DEVICE ADDRESS        } END OF BLOCK
235   F161            WT    I,I,R             PUT IN FUNCTION CODE         } DISCONNECT
236   2109            LF    I,X'09'           OUTPUT FROM 'A' COMMAND      } DISCONNECT
237   158D            JP    FUN               DISCONNECT DEVICE
238   CA03    CIO2    MN    PL                GET CURRENT INSTRUCTION
239   AB02            RM    PU
23A   140E            JP    RNI2
23B   70E0    CIO4    K     F0,14             INPUT                        } CONCURRENT DATA
23C   1000            L     X'00'             NOP                          } INPUT STROBE
23D   AE13            WN    S3                STORE INPUT DATA             } INPUT STROBE
23E   B021            CT    F0,T              GET INPUT BYTE               } INPUT STROBE
23F   1C24            JP    CIO3
```

PART V

SYSTEM DESIGN PROCEDURES USING MICROPROGRAMMING

INTRODUCTION

Computer system design is greatly simplified by adherence to a basic sequence of activities. Each step is essential to the overall success as thoroughly as possible to simplify subsequent steps and to reduce the amount of revision to previous steps. Many of the procedures listed below appear to be removed from the computer considerations because they deal with the system as a whole. However, it turns out that to obtain *full* advantage of the cost savings and system enhancement capabilities of a microprogrammable processor it is absolutely necessary to start considering the computer characteristics right at the beginning during the preliminary system functional definition phase.

Outline of System Definition Procedures

1. System Functional Definition:
   - Operations
   - Inputs and Outputs
   - Control Functions
   - Basic Functional Units/Tasks
2. System Configuration Definitions:
   - System Block Diagram
   - Basic Data Flow Definition
   - Subunit Functional Definitions
3. Detailed System Performance Specification:
   - Data Rates
   - Accuracies
   - Data Processing Functions
   - Data Formats
   - Number of Channels
   - Characteristics of Peripheral Devices
4. Interface Specifications:
   - Number of Lines
   - Data Rates
   - Interface Procedures
   - Status Lines
   - Control Lines
   - Control Codes
   - Device Addresses
5. Program Specifications:
   - Processing Functions
   - Data Rates
   - Data Characteristics
   - General Subroutine Definition
   - Mathematical Function Definition
   - Nonmathematical Process Definition
   - Input and Output Data Content and Formats

6. Tradeoff Analysis:

    Software
    Firmware
    Hardware

7. Processor and Interface Hardware Specifications:

    Architecture
    Number of Lines

8. Software/Firmware Program Specifications.

9. Detailed Program Functions, Analysis and Definition:

    Top Level flow of System Program
    Algorithm Selection and Definition
    Memory Allocations
    Interface Address and Functions Assignments
    Subroutine Hierarchy Definition
    Determination of Data Tables, Pointers, etc.
    Coding, Assembly
    Preparation of Diode Map
    Prepare Read Only Memory
    Prepare Software Programs (if any)
    System Checkout

These steps are considered only in their relation to the programming requirements. There are many other steps related to hardware design and component selection that are not covered here.

To illustrate the preceding points a generalized example of a computer system has been selected. This system would typically be used in a monitor and control system. It has the following functions:

Multichannel Analog Input

Dual Channel DAC Output

High Speed Paper Tape Reader for Entering Programs Locally

Communications Channel for Remote Status Reports

High Speed Printer for Local Status and Data Printout

Status Switch Closure Monitor

Control Relay Output

Operating Mode Control and Status Display Panel

Core Memory for Data and Storage Instruction

Real Time Clock and Power Fail Detect Option

Computer

Read Only Memory

1. System Functional Definition

In this section the following functions are defined for the example system:

a. Operational characteristics of system to be controlled:

- Block Diagrams
- Graphs
- Transfer Functions for Control equations
- Timing Diagrams for Response Time
- Sequence Diagrams for Control Algorithms

b. Function of each Analog Input Channel:

- Range
- Rates
- Accuracy
- Relation of Data to System Operation
- Signal Profile

c. Function of each Analog Control Channel:

- Range
- Rates
- Accuracy
- Signal Profile
- Effect of Data on System Operation

d. Definition of Status Switches:

- Functions
- Rates to be Monitored
- Meaning

e. Control Relay Functional Definition

- Latch vs. Non Latch
- Effect of Each Relay on System Operation

f. Communications Requirements

- Message Characteristics
- Data Rates
- Hand Shaking Procedure
- Formats

g. Panel Control and Display Functions:

- Number and Meaning of Control Switches
- Quantity, Type and Meaning of Status Displays

h. Printer

- Message Formats
- Printout rate
- Message Line Size

2. System Configuration Definition

The System Block Diagram for the controller is as shown in Figure 40 with basic data flow indicated on the block diagram as well as subunit functional definitions.

3. Detailed System Performance Specifications

Typical factors which affect the programming are as follows:

- ADC Conversion Accuracy (Number of Bits)
- ADC Sample Rate, and Conversion Time
- DAC Update Rate
- Code Conversions
- Scaling Requirements
- Curve Fitting Characteristics
- Transfer Function Calculations
- Averaging
- Communication Link Requirements
  - Rates
  - Formats
  - Controllers
  - Handshaking
  - Polling Procedures
- Printout Message Requirements
- Processing Variations Relative to Status and Control Panel Inputs
- Control Point Output Requirements
- Initialization of Cold Start Requirements

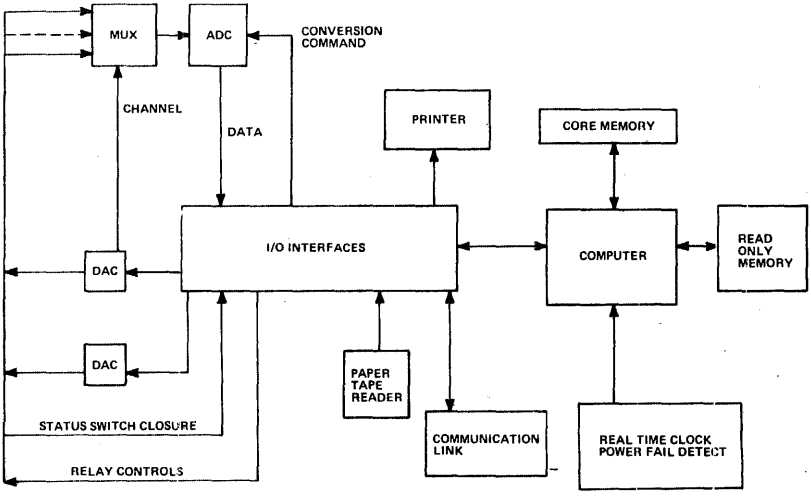


Figure 40. System Example Block Diagram

4. Interface Performance Specifications

After the peripheral hardware has been selected and defined in detail, the specifications for the interface to the computer can be defined. This consists of identifying data, status, and control lines from each peripheral device. Line groupings for each category are established, so they can be most efficiently organized to match the byte I/O characteristics of the computer control and data transfer. Timing and sequence requirements for each interface are also defined. This information is used to help determine the degree of hardware vs. microprogramming to be used for the interface.

5. Program Specifications

The program specs define all processing functions. They include a list of all functional subroutines, data processing rates, organization of the executive routine, tables or lists of input and output data categories, and definition of the mathematical, logical, and algorithmic processes to take place, and the order in which these processes occur.

A typical list of routines might be as follows:

- Application Routines
- Cold Start
- Main Loop
- Determine Next Processing State
- Output Analog Control Parameters to DAC's
- Linear Interpolation
- Calculate Basic Control Parameters
- Sample Console Settings
- Sample Analog Parameters and Convert to System Units
- Compute System RPM
- Update System Status Display
- Process Interrupts
- Communications Routine
- Status Message Printout Routine
- Paper Tape Reader Input Routine
- Code Conversion Routine
- System Status Monitor Routine
- Relay Control Update Routine
- Utility Routines (If Microprogram Is Used)
- Multiply
- Store X
- Load X
- Divide
- BCD to B in Any
- B in any to BCD
- Shift Left N bits
- Shift Right N bits
- Square Root
- Input/Output
- Printout
- Integrate
- Data Average

The general organization of these routines is defined at this stage of analysis, along with an estimate and definition of core memory requirements for flags, buffers, partially processed data, console and status switch memory maps, and system status information.

Also, the processing time for the various routines are estimated and defined along with an estimate of micro instruction requirements.

6. Tradeoffs

Before the detailed hardware and program specifications are tied down it is necessary to conduct a tradeoff analysis to assure that the cost/performance requirements for the system are being met. Here the tradeoff is related to application of hardware, firmware, and software to the various internal and interface functions of the computer. The areas of cost reduction to be considered are as follows:

- Interface Hardware Complexity
- New hardware Design Requirements
- Microprogram Size
- Core Memory Requirements
- Complexity of Peripheral Devices
- Availability of Existing Programs
- Program Development Times

A large number of factors must be included in the tradeoff analysis. The most important ones related to program development are listed below:

- Overall data throughput requirements including peak and average data loads.

- Variability of program functions, including operating modes, data formats, status combinations, processing states, number of I/O channels, operating ranges, etc.

- Permanence of program structure, once defined, and need to avoid having to load program on site.

- Speed and complexity of peripheral devices and processing functions.

- Existing standard interfaces, and the extent of microprogramming required for these interfaces.

- Number of systems to be developed and available development time (affecting nonrecurring costs ratio, and development staffing requirements).

- Special processing requirements with high speed or complexity in the fields of arithmetic, logic data manipulation, character assembly, control functions, hand shaking, etc.

- Overall program size.

- Existing standard firmware and software routines which are applicable to the system.

- Operating complexity, maintenance and training requirements.

- System reliability, including failure rates, and equipment redundancy requirements, which may dictate the requirement for self contained hardware functions.

The result of the tradeoff study will be the following:

- Use of sophisticated interfaces not requiring firmware, or use of extremely simple interfaces which do require firmware. (Tradeoff factors: Read only memory capacity for interface functions, speed of data transfer, interface control sequences, available process time.)

- Use of software program for entire operation.

- Use of software program with special I/O or processing routines added to microprogram.

- Development of special instruction set for the application.

- Combined use of special firmware, special hardware interfaces, and special hardware processing functions such as hardware multiply/divide.

Typical functions which may be completely or partially done by two or three of the following: Software, firmware or hardware, depending on data processing rates, hardware complexity, system throughput requirements, read only memory capacity, thus must have tradeoff analysis applied for selection.

- Serial data character assembly/disassembly
- Card reader control and data transfer
- Binary to BCD or ASCii conversion
- BCD to binary Conversion
- Multiply or divide
- Digital filtering
- Magnetic tape controller functions
- High-speed line printer control
- ADC control and data input
- Message Switching
- Remote monitor functions
- Synchronous modem control
- Image scanning
- Disc controller
- Error detection, and code generation
- Table lookup
- Communications line polling/handshaking
- Console parameter input/scaling

Tradeoff Examples:

Example 1

Firmware can be used to interface with a card reader having minimum readout electronics. However if the firmware must monitor the high-speed stroke pulses from the card reader to synchronize with the reader data lines, the firmware becomes too tied down to service other peripherals. Therefore the card reader interface should have some character synch. even with firmware if multiple peripheral devices must operate simultaneously.

Example 2

Display lamps could be scanned by firmware to avoid using latches to hold display parameters. In a system of any size this will tie up the computer considerably, and the cost of the firmware may be as much as the latches.

Example 3

Firmware can be used to control a disk without using DMA except for character shifting for transfer to and from the track. However if there is a requirement to simultaneously interface with the disk and another peripheral device, even firmware may not be fast enough.

7. Hardware Specs

The hardware specs of interest here are for the interfaces and special processing functions and relate to the programming requirements. They include the following:

- Definition of standard interfaces, including complete identification of data input and output channels, control line functions, status lines, device and function codes, and timing requirements for dynamic data or control lines.

- Definition of special interfaces including all of the factors for standard interfaces plus special control sequences and special data input/output sequences which must be microprogrammed. These definitions must be in terms of the standard control and byte transfer functions of the computer.

- Definition of special processing hardware units, such as hardware multiply/divide, buffers, fast fourier processor, digital filter, etc. Again, the basic interest for this document is the programrning required to transfer data and initiate the special processor operation.

8. Software or Firmware Program Specifications

These include a detailed functional description of all subroutines, executive routine, data, control, status words, memory requirements, data tables, flags, pointers, etc.

9. Detailed Program Functions Analysis Definitions and Programming

The general steps to be followed in the programming phase should be adhered to simplify the entire task and to assure the best program results.

- Top level flow chart
- Detailed algorithm definition
- Memory allocations (data, flags, pointers, etc.)
- Interface address and function tabulation
- Definition of subroutine, hierarchy (looping, branching, nesting).
- Preparation of tables and formats for data, status, flags, pointers, scale factors, address pointers.
- Top level flow charts for subroutines.
- File register assignments.
- Detail subroutine subcharts.
- Coding, assembly, checkout, etc.

These steps are illustrated in the emulator example which follows and in the microprogram subroutine examples in the microprogrammers manual.

The last step consists of converting the flow chart functions into routines that are ready for implementation in hardware to yeild the system firmware. These steps include translating the MICRO 800 instructions selected for each routing into the mnemonic or machine language code, loading them into an operating system, and eliminating any errors that may have been made during the previous steps. Microdata Corporation furnishes a software program (Simulator Operating System) for use on one of the 800 series computers which simulates the user's microprogram and provides operator control for debugging and evaluation procedures. The completed program is printed in the form of a diode map to simplify the placement of diodes on the read only memory circuit boards which contain the complete microprogram.

Microprogramming Aids

The software aids for microprogramming, furnished by Microdata Corporation are briefly described in Figure 41. Several methods are available to convert the microprogram source statements to the final diode map for hardware implementation. These methods incorporate different programs according to the processing equipment available to the user. For instance, the MAP800 program is used with a MICRO 811 computer to enter source statements and assemble the listings. The AP800 program is used on a large-scale computer to produce an object program. Variations in methods also permit selection of media for recording and communicating the program information including punched cards, paper tape, printed documents, etc.

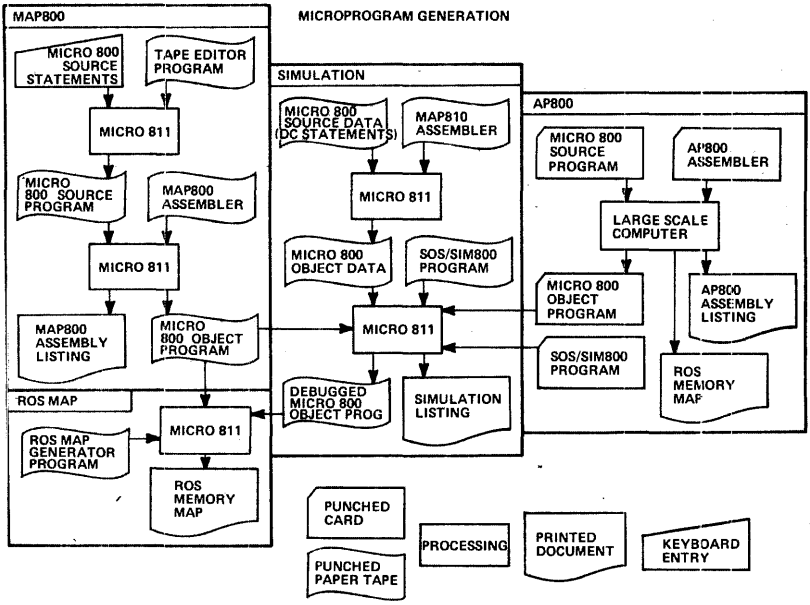


Figure 41. Microprogramming Generation

The final step in the process is the implementation of the microprogram by loading the signal diodes on the ROM circuit boards. This process consists of inserting diodes in the board at locations designated by the diode map and corresponding to the logical 1's in the machine language code. The absence of a diode indicates a logical 0. When the complete microprogram has been implemented in diodes on the ROM boards, the "new" computer is assembled by inserting these boards into the standard MICRO 800 enclosure which houses the hardware components furnished by Microdata Corporation.

PART VI

PRODUCT CATALOG

MICRO 400 COMPUTER

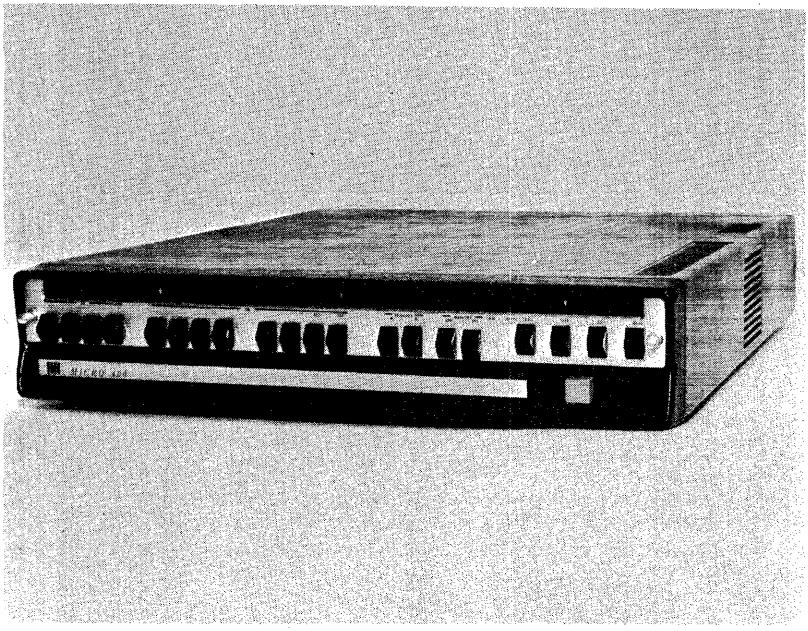
The MICRO 400 is a programmable, high-speed, general-purpose computer designed for the large-volume user or original equipment manufacturer. Although small and low-priced, the MICRO 400 is remarkably powerful.

Architectural simplicity is fundamental in the MICRO 400 and hardware packaging allows the user to easily incorporate basic equipment modules for his application. A comprehensive set of interfaces is available for peripheral, communications and utility devices.

The input/output structure uses a standard programmable data channel and MICRObus, a single bus organization which provides direct access for all memory and system control devices and for the central processing unit.

Extensive standard support software is provided, including a symbolic assembler for preparation of source programs in symbolic notation.

The MICRO 400 features 1.6 microsecond cycle time, 400 nanosecond access time, basic memory module sizes ranging from 1024 to 8,192 words of core memory direct addressing to 4,096 words and operates up to 32 I/O devices. The machine weighs 23 pounds complete and uses 3.5 inches of rack space.



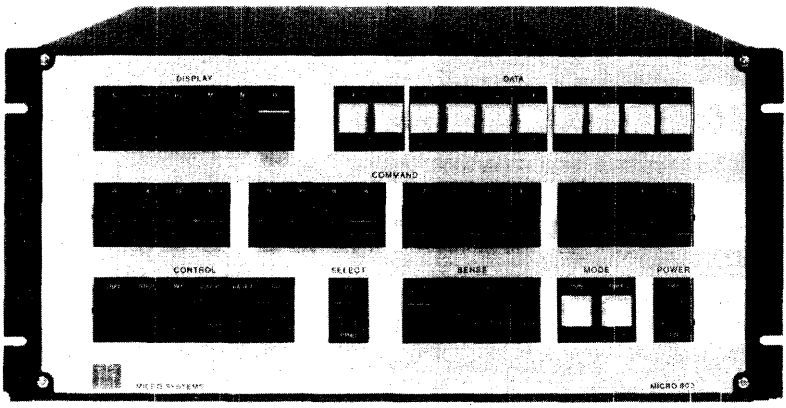
MICRO 800 COMPUTER

The MICRO 800 is a high-speed microprogrammed computer whose flexibility, functional modularity and system-oriented packaging make it ideally suited for dedicated volume applications.

The MICRO 800's flexibility permits the computer system to be expanded or reduced to the exact configuration needed for any application. For example, the computer can be used without a core memory as an inexpensive controller or data concentrator. When memory is required for storage of variable parameters, tables or data, high-speed core memory may be added to the system.

The MICRO 800 also can be microprogrammed to emulate other general or special-purpose computers enabling the software of these machines to be compatible with the MICRO 800. In such a case, additional interface hardware can be furnished to provide plug-to-plug compatibility with other computers.

In addition to low unit cost, the MICRO 800 system also can reduce overall system cost. The high-speed execution of firmware routines allows the processor logic to be time-shared to minimize input/output interface hardware.



Microprogramming also provides exceptionally high performance with an unusually small amount of internal hardware. The basic computer consists of two identical data boards, each of which is a 4-bit slice of the computer's data paths and registers, and a single control board which provides command decoding and timing.

Main frame options including memory parity, power fail/automatic restart, real-time clock and input/output interfaces are implemented on card modules which plug into the basic MICRO 800 enclosure.

With its 1.1 microsecond core memory cycle time and 220 nanosecond command execution time, the MICRO 800 is the fastest machine in its class. Core memory is expandable from 0 to 32,768 bytes in 4,096 byte or 8192 byte increments. A 1,024 byte core memory also is available for small, inexpensive systems. Weight is 75 pounds.

MICRO 810 COMPUTER

The MICRO 810 is a general purpose computer which is a microprogrammed adaptation of the MICRO 800. Microprogrammed subroutines, configured in the read only memory, interpret macro instructions of programs stored in the core memory.

A powerful macro level computer, the MICRO 810 also retains all the modular and functional advantages of the MICRO 800.

The MICRO 810 has available considerably larger programs than most machines in its class, combined with ease of programming and programming flexibility. Some of the advantages of the MICRO 800 can be obtained by adding problem-oriented instructions or firmware subroutines to the MICRO 810. Multiply/divide instructions are standard.

The MICRO 810 features 1.1 microsecond cycle time and 220 nanosecond execution time in the ROM. Core memory is field-expandable to 32,768 Bytes (8, 9 or 10 bits). Extra memory bits may be used for memory parity and special applications. A 1024-byte by 9-bit core memory also is available. Weight is 75 pounds.

MICRO 820 COMPUTER

Featuring a comprehensive instruction repertoire and powerful input/output facility, the MICRO 820 is a high-speed, microprogrammed general purpose computer capable of handling a wide variety of applications.

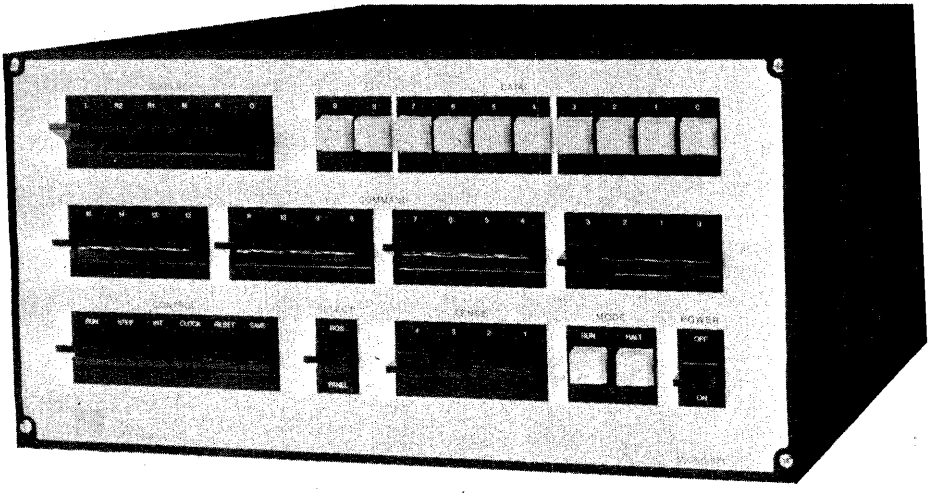
Use of high-speed read-only memories for macro control greatly reduces the number of CPU circuits which otherwise would be required to provide the powerful instructions of the MICRO 820.

A superior price/performance ratio is achieved in the MICRO 820 by efficient core memory usage and ease of programming.

The MICRO 820 system is designed to accommodate additional standard and special firmware inexpensively, permitting the user to specify augmented capabilities such as multiply/divide instructions, BCD arithmetic, floating point arithmetic, trigonometric and transcendental functions and fully buffered communications multiplexers.

Among features of the MICRO 820 are variable precision operation, character/string manipulation and stack processing. A complete line of peripheral options is available to achieve almost unlimited flexibility in application of the MICRO 820.

Core memory is expandable to 32,768 bytes in the basic 8¾-inch cabinet using 4,096 and 8,192 plug-in memory modules. Cycle time is 1.1 microsecond in core memory and 220 nanosecond execution time in the ROM.


RUN
STEP
INT
CLOCK
RESET
SAVE
MODE
RUN
HALT
POWER
OFF
ON

MICRO 1600 COMPUTER

Newest and most advanced of Microdata Corporation's families of computers is the MICRO 1600, a companion product line to the MICRO 800 which provides significant performance improvements in both speed and function.

Both the 1600 and 800 are functionally compatible, enabling established MICRO 800 users to use the 1600 directly without redevelopment of firmware, software or system peripherals or interfaces.

However, new and revised firmware can achieve significant performance improvements at both the micro and macro levels of programming.

The MICRO 1600 is an economical machine with unequalled flexibility which can be tailored to fit almost any application. Modular design of core memory, processor, microprogram control memory and input/output modules provides easy, economical expansion of all functional areas of the computer.

Extra space and power in the basic enclosure permits growth from a minimum to a fully expanded configuration without the need for special or expansion enclosures. User-designed interfaces can be installed in the computer cabinet.

The widest range of hardware, firmware and software options in the industry is available to augment the MICRO 1600.

Improved features of the MICRO 1600 are higher speed, processor options which are part of the CPU, additional general-purpose registers, control memory expansion to 16,384 words, core memory expansion to 65,000 words, dual processor capability, memory data buffer, data output buffer, memory address link bit and expanded control panel facilities. This is accomplished through maximum use of the most advanced MSI and LSI technology.

Control memory cycle time is 1 microsecond, 200 nanosecond command execution rate.


Microdata
1600

FIRMWARE TRAINING SYSTEM

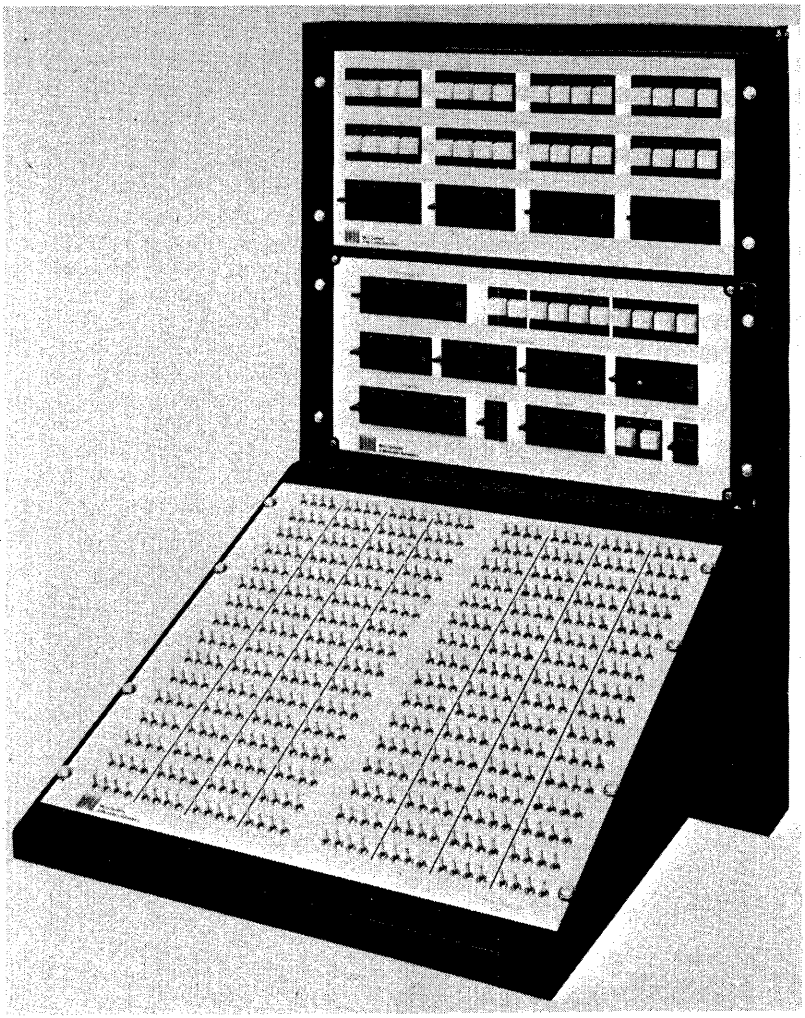
The firmware trainer is a valuable tool for classroom teaching of microprogramming techniques. Small firmware routines can be quickly set up and checked out with the aid of the comprehension switch panel layout and the built-in visual display. Firmware alterations and corrections are made quickly and efficiently, permitting the student to concentrate on the problem rather than the hardware.

The system consists of a MICRO 800 computer with a utility read-only memory, a switch matrix read-only memory, a 4096 byte magnetic core memory, a TTY/display controller and an I/O display panel.

The MICRO 800 computer includes a special interface wired to a panel with 512 switches. Each switch connects a diode to the computer to designate a logical 1 for binary values of the microprogram command sequence. A maximum of 32 commands may be used at one time on the panel.

As an aid in demonstration and training activities, the preprogrammed utility ROM is included to facilitate input/output functions without expending instructions on the ROM switch panel. Six utility routines are included to permit display and recording of data obtained during execution of microprograms.

A 30-page operations manual and 50 copies of the microprogramming handbook are included with the firmware trainer system. Price for the system is $10,000.



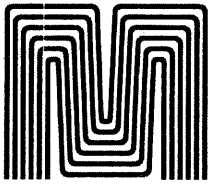
ALTERABLE READ-ONLY MEMORY SYSTEM

Designed for use with the MICRO 800 series of computers, Microdata Corporation's Alterable Read-Only Memory System for test and debugging of microprograms in a real-time environment permits implementation of firmware on a level comparable to software and gives the user a wide range of application flexibility.

Using the concept of dynamic microprogramming, the system operates at full control memory speed of 220 nanoseconds command execution time. The basic capacity of the system is 1K by 16, but can be expanded to 2K by 16.

A supporting software package called the Alterable Read-Only Memory Operating System is included, and a card reader is optional. The software package permits loading of the machine from a variety peripheral devices and permits the operator to examine and alter the contents at will.

June 1971



MICRODATA CORPORATION

COMMENT AND EVALUATION SHEET
Microprogramming Handbook

YOUR EVALUATION OF THIS HANDBOOK WILL BE WELCOMED BY MICRODATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM

NAME: _____

BUSINESS
ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINE AND STAPLE

STAPLE

FOLD HERE

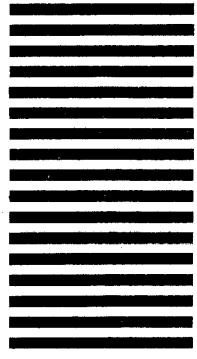
First Class
Permit No. 1972
Santa Ana
California 92711

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE U.S.A.

TRIM HERE

Postage Will Be Paid By

MICRODATA CORPORATION
644 East Young Street
Santa Ana, California 92705



STAPLE