

**Honeywell**

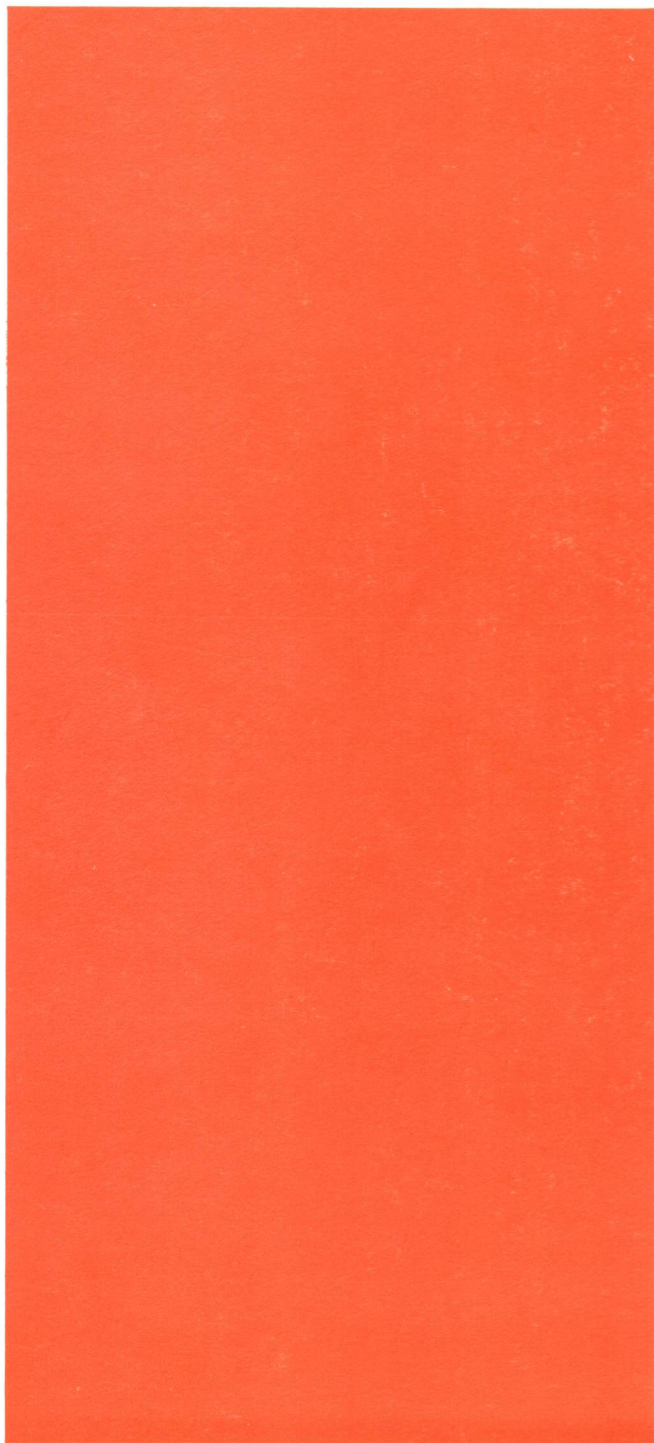
**DATANET 355/6600  
MACRO ASSEMBLER PROGRAM**

**SERIES 60 (LEVEL 66)/6000**

---

**SOFTWARE**

---



# Honeywell

## DATANET 355/6600 MACRO ASSEMBLER PROGRAM

SERIES 60 (LEVEL 66)/6000

SUBJECT:

Language Characteristics, Programming Methods, Instructions, Pseudo-Operations, Input/Output Operations.

SPECIAL INSTRUCTIONS:

This manual replaces DATANET 355 Macro Assembler Program, Order No. BB98, for Series 6000 System users. Order No. BB98 must be used by Series 600 System users and by Series 6000 System users on prior software releases.

SOFTWARE SUPPORTED:

Series 60 Level 66 Software Release 2  
Series 6000 Software Release H

DATE:

December 1975

ORDER NUMBER:

DD01, Rev. 0

## PREFACE

The DATANET FNP Macro Assembler Program manual is intended as a guide for the programmer using the Symbolic Macro Assembler (MAP). It contains all the information needed by the programmer to code an assembly language program on the coding form.

The manual is addressed to programmers experienced in assembly language programming. It assumes some knowledge and experience in the use of indirect address modification, hardware indicators, faults, interrupts and recovery routines, macro operations, pseudo-operations, and other features normally encountered in a very flexible instruction repertoire. It also assumes that the programmer is familiar with the TWOs complement number system as used in a sign-number machine.

FUNCTIONAL LISTING OF PUBLICATIONS  
for  
SERIES 60 (LEVEL 66) and SERIES 6000 SYSTEMS

FUNCTION	APPLICABLE REFERENCE MANUAL	ORDER NO.
	<u>TITLE</u>	
	<u>Series 60 (Level 66)/Series 6000:</u>	
Hardware reference:		
Series 60 Level 66 System	Series 60 Level 66 Summary Description	DC64
Series 6000 System	Series 6000 Summary Description	DA48
DATANET 355 Processor	DATANET 355 Systems Manual	BS03
DATANET 6600 Processor	DATANET 6600 Systems Manual	DC88
Operating system:		
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	DD19
Job Control Language	Control Cards Reference Manual	DD31
Table Definitions	System Tables	DD14
I/O Via MME GEINOS	I/O Programming	DB82
System initialization:		
System Startup	System Startup	DD33
System Operation	System Operation Techniques	DD50
Communications System	GRTS/355 and GRTS/6600 Startup Procedures	DD05
Communications System	NPS Startup	DD51
DSS180 Subsystem Startup	DSS180 Startup (Series 6000 only)	DD34
Data management:		
File System	File Management Supervisor	DD45
Integrated Data Store (I-D-S)	I-D-S/I Programmer's Guide	DC52
Integrated Data Store (I-D-S)	I-D-S/I User's Guide	DC53
File Processing	Indexed Sequential Processor	DD38
File Input/Output	File and Record Control	DD07
File Input/Output	Unified File Access System (UFAS) (Series 60 only)	DC89
I-D-S Data Query System	I-D-S Data Query System Installation	DD47
I-D-S Data Query System	I-D-S Data Query System User's Guide	DD46
Program maintenance:		
Object Program	Source and Object Library Editor	DD06
System Editing	System Library Editor	DD30
Test system:		
Online Test Program	Total Online Test System (TOLTS)	DD39
Test Descriptions	Total Online Test System (TOLTS) Test Pages	DD49
Error Analysis and Logging	Honeywell Error Analysis and Logging System (HEALS)	DD44
Language processors:		
Macro Assembly Language	Macro Assembler Program (GMAP)	DD08
COBOL-68 Language	COBOL	DD25
COBOL-68 Usage	COBOL User's Guide	DD26
JOVIAL Language	JOVIAL	DD23
FORTRAN Language	FORTRAN	DD02
Macro Assembly Language	DATANET 355/6600 Macro Assembler Program	DD01
Generators:		
Sorting	Sort/Merge Program	DD09
Merging	Sort/Merge Program	DD09

## FUNCTION

## APPLICABLE REFERENCE MANUAL

ORDER  
NO.

	TITLE	
	Series 60 (Level 66)/Series 6000:	
Simulators:		
DATANET 355/6600 Simulation	DATANET 355/6600 Simulator	DD32
Service and utility routines:		
Loader	General Loader	DD10
Utility Programs	Utility	DD12
Utility Programs	UTL2 Utility Routine (Series 60 only)	DC91
Media Conversion	Bulk Media Conversion	DD11
System Accounting	Summary Edit Program	DD24
FORTTRAN	FORTTRAN Subroutine Libraries	DD20
FNP Loader	DATANET 355/6600 Relocatable Loader	DD35
Service Routines	Service Routines	DD42
Software Debugging	Debug and Trace Routines	DD43
Time Sharing systems:		
Operating System	TSS General Information	DD22
System Programming	TSS Terminal/Batch Interface	DD21
System Programming	TSS System Programmer's Reference Manual	DD17
BASIC Language	Time Sharing BASIC	DD16
FORTTRAN Language	FORTTRAN	DD02
Text Editing	Time Sharing Text Editor	DD18
dataBASIC Language	dataBASIC System Language Manual	DD95
dataBASIC Loading	dataBASIC Load/Unload System	DD96
Remote communications:		
DATANET 30/305/355/6600 FNP	Remote Terminal Supervisor (GRTS)	DD40
DATANET 355/6600 FNP	Network Processing Supervisor (NPS)	DD48
DATANET 700 RNP	RNP/FNP Interface	DB92
Transaction processing:		
User's Procedures	Transaction Processing System User's Guide	DD41
Handbooks:		
System-operator communication	System Console Messages	DD13
Error Messages, Abort Codes	Error Messages and Abort Codes	DC97
Pocket guides:		
Control Card Formats	Control Cards and Abort Codes	DD04
FORTTRAN	FORTTRAN Pocket Guide	DD82

## CONTENTS

		Page
Section I	Introduction. . . . .	1-1
	Relocatable and Absolute Assemblies. . . . .	1-2
	Source Program Input . . . . .	1-3
	\$ SNUMB Control Card. . . . .	1-3
	\$ 355MAP Control Card . . . . .	1-3
	\$ LIMITS Control Card . . . . .	1-4
	\$ UPDATE Control Card (optional). . . . .	1-5
	\$ ALTER Control Card. . . . .	1-5
	\$ ENDJOB Control Card . . . . .	1-5
	***EOF Control Card . . . . .	1-6
	Typical Deck Setups. . . . .	1-6
	Assembler Outputs. . . . .	1-6
	Binary Decks. . . . .	1-7
	Relocatable Object Decks. . . . .	1-8
	Preface Cards . . . . .	1-8
	Relocatable Text Cards. . . . .	1-10
	Absolute Object (Binary) Decks. . . . .	1-11
	Absolute Text Card. . . . .	1-11
	Transfer Card . . . . .	1-12
	Assembly Listing. . . . .	1-12
	Full Listing Format . . . . .	1-13
	Preface Card Listing. . . . .	1-15
	SYMDEF, SYMREF, Labeled Common. . . . .	1-15
	Blank Common. . . . .	1-15
	Symbolic Reference Table. . . . .	1-15
	Error Flags . . . . .	1-16
Section II	Assembly Language Programming . . . . .	2-1
	Symbolic Coding Form . . . . .	2-1
	Location Field. . . . .	2-3
	E/O/8 Field . . . . .	2-3
	Operation Field . . . . .	2-3
	Variable Field. . . . .	2-3
	Comments Field. . . . .	2-4
	Identification Field. . . . .	2-5
	Summary of Symbolic Card Format . . . . .	2-5
	Language Structure . . . . .	2-5
	Character Set . . . . .	2-5
	Program Symbols . . . . .	2-5
	Types of Program Symbols. . . . .	2-6
	Expressions in General . . . . .	2-6
	Elements. . . . .	2-6
	Terms and Operators . . . . .	2-7
	Asterisk Used as An Element . . . . .	2-7
	Algebraic Expressions . . . . .	2-8
	Evaluation of Algebraic Expressions . . . . .	2-8
	Boolean Expressions . . . . .	2-8
	Evaluation of Boolean Expressions . . . . .	2-9
	Relocatable and Absolute Expressions. . . . .	2-10
	Special Relocatable Expressions . . . . .	2-11
	Literals . . . . .	2-12
	Decimal Literals. . . . .	2-13
	Octal Literals. . . . .	2-14

CONTENTS (cont)

	Page
Alphanumeric Literals . . . . .	2-14
SACI Literals . . . . .	2-15
Instruction Literals. . . . .	2-15
Variable Field Literals . . . . .	2-16
Nonmemory Reference Instruction Literals. . . . .	2-16
 Section III	
Processor Instructions. . . . .	3-1
Representation of Information. . . . .	3-1
Single-Precision Data . . . . .	3-1
Double-Precision Data . . . . .	3-1
Alphanumeric Data . . . . .	3-2
Number System. . . . .	3-2
Instructions . . . . .	3-4
Memory Reference Instruction. . . . .	3-4
Nonmemory Reference Instructions. . . . .	3-4
Group 1 Nonmemory Instructions. . . . .	3-5
Group 2 Nonmemory Instructions. . . . .	3-5
Processor Registers. . . . .	3-6
Faults . . . . .	3-6
Program Interrupts . . . . .	3-9
Processor Indicators . . . . .	3-12
Address Formation. . . . .	3-14
Basic Level Effective Address Formation	
Rules. . . . .	3-15
Word Addressing - Basic Level. . . . .	3-16
Character Addressing - Basic Level . . . . .	3-17
Indirect Level Effective Address Formation	
Rules. . . . .	3-21
Word Addressing - Indirect Level . . . . .	3-21
Character Addressing - Indirect Level. . . . .	3-22
Processor Instructions . . . . .	3-22
Processor Instruction Description . . . . .	3-28
Effective Address and Memory Location	
Symbols . . . . .	3-28
Register Symbols . . . . .	3-29
Register Positions and Content Symbols . . . . .	3-29
Other Processor Symbols. . . . .	3-29
Memory Reference Instructions . . . . .	3-30
Load Instructions. . . . .	3-30
Store Instructions . . . . .	3-32
Add Instructions . . . . .	3-34
Subtract Instructions. . . . .	3-37
Multiply Instructions. . . . .	3-39
Divide Instructions. . . . .	3-40
Boolean Instructions . . . . .	3-41
Compare Instructions . . . . .	3-43
Transfer Instructions. . . . .	3-45
Input/Output Instructions. . . . .	3-48
Nonmemory Reference Instructions. . . . .	3-49
Group 1, Immediate Add Instructions. . . . .	3-49
Group 1, Immediate Load Instructions . . . . .	3-52
Group 1, Interrupt Control Instructions. . . . .	3-53
Group 1, Immediate Boolean Instructions. . . . .	3-54
Group 1, Immediate Compare Instructions. . . . .	3-55
Group 2, Data Movement Shift Instructions. . . . .	3-57
Group 2, Data Movement Normalize	
Instructions. . . . .	3-62
Group 2, Data Movement Copy Instructions . . . . .	3-63
Group 2, Interrupt Control Instructions. . . . .	3-65
Group 2, Miscellaneous Instructions. . . . .	3-65
 Section IV	
Pseudo-Operations . . . . .	4-1

CONTENTS (cont)

	Page
Control Pseudo-Operations. . . . .	4-4
ON/OFF Switch Type Control Pseudo-Operation . . . . .	4-4
DETAIL ON/OFF - Detail Output Listing . . . . .	4-5
LIST ON/OFF - Control Output Listing. . . . .	4-5
PCC ON/OFF - Print Control Cards. . . . .	4-5
REF ON/OFF - References . . . . .	4-6
REFMA ON/OFF - Reference Macro Operation. . . . .	4-6
PARITY ON/OFF - ASCII Parity Control. . . . .	4-7
PMC ON/OFF - Print Macro Expansion. . . . .	4-8
PUNCH ON/OFF - Control Card Output. . . . .	4-8
EDITP - Edit Print Lines. . . . .	4-9
Control Pseudo-Operations. . . . .	4-9
EJECT - Restore Output Listing. . . . .	4-9
REM - Remarks . . . . .	4-9
* In Column One --Remarks . . . . .	4-10
LBL - Label . . . . .	4-10
TTL - Title . . . . .	4-11
TTLS - Subtitle . . . . .	4-11
CPR - Copyright . . . . .	4-11
ABS - Output Absolute Text. . . . .	4-12
FUL - Output Full Binary Text . . . . .	4-12
TCD - Punch Transfer Card . . . . .	4-13
HEAD - Heading. . . . .	4-13
DCARD - Punch BCD Card. . . . .	4-15
END - End of Assembly . . . . .	4-15
OPD - Operation Definition. . . . .	4-16
OPSYN - Operation Synonym . . . . .	4-18
Location Counter Pseudo-Operation. . . . .	4-19
USE - Use Multiple Location Counters. . . . .	4-19
BEGIN - Origin of a Location Counter. . . . .	4-19
ORG - Origin Set by Programmer. . . . .	4-20
LOC - Location of Output Text . . . . .	4-20
Symbol - Defining Pseudo-Operations. . . . .	4-21
EQU - Equal To. . . . .	4-21
FEQU - Special FORTRAN Equivalence. . . . .	4-21
BOOL - Boolean. . . . .	4-22
SET - Symbol Redefinition . . . . .	4-22
MIN - Minimum . . . . .	4-23
MAX - Maximum . . . . .	4-23
SYMDEF - Symbol Definition. . . . .	4-23
SYMREF - Symbol Reference . . . . .	4-24
NULL - Null . . . . .	4-25
EVEN - Force Location Counter Even. . . . .	4-25
ODD - Force Location Counter Odd. . . . .	4-25
EIGHT - Force Location Counter to a Multiple of 8. . . . .	4-25
BASE - Force Location Counter to a Multiple Power of 2. . . . .	4-26
Data Generating Pseudo-Operations. . . . .	4-26
OCTAL - Octal . . . . .	4-27
DEC - Decimal . . . . .	4-27
BCI - Binary Coded Decimal Information. . . . .	4-29
VFD - Variable Field Definition . . . . .	4-29
ASCII, ASCIIIC, ACI, ACIC - ASCII Coded Information. . . . .	4-31
SACI - Symbolic ASCII Information . . . . .	4-32
DUP - Duplicate Cards . . . . .	4-35
Memory Allocation Pseudo-Operations. . . . .	4-36
BSS - Block Started by Symbol . . . . .	4-36
BFS - Block Followed by Symbol. . . . .	4-36
BLOCK - Block Common. . . . .	4-36
LIT - Literal Pool Origin . . . . .	4-37



CONTENTS (cont)

	Page
Conditional Pseudo-Operations. . . . .	4-37
IFE - If Equal. . . . .	4-38
IFG - If Greater Than . . . . .	4-38
IFL - If Less Than. . . . .	4-39
INE - If Not Equal. . . . .	4-40
Special Word Format Pseudo-Operations. . . . .	4-40
MARK - Specify Symbol in Location Field . . . . .	4-40
ARG - Argument - Generate Zero Operation Code Computer Word. . . . .	4-41
TTLDAT - Title Date . . . . .	4-41
DATE - Current Date . . . . .	4-41
NONOP - Undefined Operation . . . . .	4-41
ZERO - Generate One Word with Two Subfields . . . . .	4-42
MAXSZ - Maximum Size of Assembly. . . . .	4-42
IND - Generate One Word for Indirect Addressing . . . . .	4-43
Data Control Word Format Pseudo-Operations . . . . .	4-44
ICW - I/O Control Word Generator. . . . .	4-44
DCW - I/O Control Word Generator. . . . .	4-45
Macro Pseudo-Operations. . . . .	4-45
Definition of the Macro Prototype . . . . .	4-46
MACRO - Macro Identification . . . . .	4-47
END - End Macro Prototype . . . . .	4-47
Using a Macro Operation . . . . .	4-50
Pseudo-Operations Used Within Macro Prototypes . . . . .	4-52
CRSM ON/OFF - Created Symbols . . . . .	4-53
ORGCSM - Origin Created Symbols . . . . .	4-53
IDRP - Indefinite Repeat. . . . .	4-53
DELM - Delete Macro Named . . . . .	4-54
Implementation of System Macro Operations . . . . .	4-55
PUNM - Punch Macro Prototypes . . . . .	4-55
LODM - Load System Macro Operations . . . . .	4-55
Notes and Example on Defining a Prototype . . . . .	4-56
Program Linkage Pseudo-Operations. . . . .	4-57
CALL - Call Subroutines . . . . .	4-57
SAVE - Save--Return Linkage Data. . . . .	4-59
RETURN - Return--from Subroutines . . . . .	4-61
ETC - Continuation. . . . .	4-62
System (Built-In) Symbols. . . . .	4-62
Section V	
Input/Output Operations . . . . .	5-1
Peripheral Control Word. . . . .	5-1
Direct Channel Programming . . . . .	5-2
Indirect Channel Programming . . . . .	5-2
Program Interrupt Control. . . . .	5-2
Status . . . . .	5-2
IOM Faults . . . . .	5-3
Intercomputer Adapter (ICA). . . . .	5-4
DATANET FNP Interface . . . . .	5-4
Central System Interface. . . . .	5-5
Interrupt Cell Assignment Switches . . . . .	5-5
Port Assignment for FNP's . . . . .	5-5
Mailbox Addresses. . . . .	5-5
Processor Fault Switches . . . . .	5-5
Emergency Interrupt Cell Number. . . . .	5-6
PCW Mailbox. . . . .	5-6
DATANET FNP Control Word Formats. . . . .	5-6
Peripheral Control Word (PCW). . . . .	5-6
List Indirect Control Word (LICW). . . . .	5-7
Data Control Word (DCW). . . . .	5-7
Configuration Status Format. . . . .	5-9

CONTENTS (cont)

	Page
Active Status Format . . . . .	5-10
Status Word Format . . . . .	5-11
Central System Control Word Formats . . . . .	5-12
Peripheral Control Word (PCW) in Central System Mailbox. . . . .	5-12
Direct Interface Adapter (DIA) . . . . .	5-14
DATANET FNP Interface . . . . .	5-14
Central System Interface. . . . .	5-15
DATANET FNP Control Word Formats. . . . .	5-16
Peripheral Control Word (PCW). . . . .	5-16
List Indirect Control Word (LICW). . . . .	5-17
Central System Address Extension . . . . .	5-19
Configuration Status Format. . . . .	5-19
Status Indirect Control Word (Status ICW). . . . .	5-20
Status Word Format in DATANET FNP Memory . . . . .	5-20
Central System Control Word Formats . . . . .	5-22
Peripheral Control Word (PCW). . . . .	5-22
Test and Bootload Indirect Control Word (Test ICW/Bootload ICW) . . . . .	5-23
High Speed Line Adapter (HSLA) . . . . .	5-23
General Information . . . . .	5-23
PCW Format. . . . .	5-24
Command PCW0, PCW1. . . . .	5-25
Command PCW2, PCW3. . . . .	5-25
Control Words . . . . .	5-25
Indirect Control Word . . . . .	5-25
Base Address Word . . . . .	5-26
Mask Register Word. . . . .	5-26
Control Word Memory Map (Example for Channel 06). . . . .	5-26
Character Control Table . . . . .	5-27
Character Control Character Addressing. . . . .	5-27
Status. . . . .	5-29
Binary Synchronization Status. . . . .	5-31
Low Speed Line Adapter (LSLA). . . . .	5-31
General Information . . . . .	5-31
Control Words . . . . .	5-32
PCW Format. . . . .	5-33
Commands, PCW0, PCW1. . . . .	5-33
Indirect Control Word . . . . .	5-33
Status. . . . .	5-34
Command Characters in DATANET FNP Memory. . . . .	5-35
Status Characters in DATANET FNP Memory . . . . .	5-36
Peripheral Subsystem Adapter (PSA) . . . . .	5-38
General Information . . . . .	5-38
PSA Word Formats. . . . .	5-38
Connect PCW (Operational Mode) . . . . .	5-38
Interrupt Multiplex Word (IMW) . . . . .	5-39
SSCW or SCW (SSCW, SCW). . . . .	5-39
List Pointer Word (LPW). . . . .	5-40
PSA Mailbox. . . . .	5-40
Instruction Data Control Word (IDCW) . . . . .	5-40
Data Control Word (DCW). . . . .	5-41
Logical Channel DCW List . . . . .	5-42
Status Word Formats. . . . .	5-42
PSA Error Summary. . . . .	5-44
Service Codes - MPC to PSA . . . . .	5-45
Service Codes - PSA to MPC . . . . .	5-45
MPC Commands . . . . .	5-46
Special Controller Commands. . . . .	5-47
MPC Device Status. . . . .	5-47
IOM Channel Status . . . . .	5-48

CONTENTS (cont)

	Page
Document Handler Channel (DHC) . . . . .	5-49
General Information . . . . .	5-49
MRS200/DRD200 . . . . .	5-50
Peripheral Control Word (PCW) . . . . .	5-50
Status Word Format . . . . .	5-51
DRD236/DHUL600 . . . . .	5-52
Load External Format (LDEX) . . . . .	5-52
Store External Format (STEX) . . . . .	5-53
Peripheral Control Word (PCW) . . . . .	5-53
Character Control Word (CCW) . . . . .	5-55
CCW Address Formation . . . . .	5-56
Queue Status Word (QSW) . . . . .	5-56
Terminate Status Word . . . . .	5-57
Binary Synchronous Channel (BSC) . . . . .	5-58
Transmit Mode . . . . .	5-59
Receive Mode . . . . .	5-60
Control Words . . . . .	5-60
Status Words . . . . .	5-61
Computer Monitor Adapter (CMA) . . . . .	5-64
Configuration Patching . . . . .	5-64
Channel Number Patch . . . . .	5-64
Interrupt Level Patch . . . . .	5-64
Indirect Control Word Base Address Patch . . . . .	5-64
"Dead-Man" Timer Duration Patch . . . . .	5-64
Data Response Timer Duration Patch . . . . .	5-64
CMA Control Words . . . . .	5-65
Peripheral Control Word (PCW) . . . . .	5-65
Indirect Control Word (ICW) . . . . .	6-66
Data Transfer . . . . .	5-67
Status Words . . . . .	5-67
System Monitoring . . . . .	5-69
Control Console Adapter (CCA) . . . . .	5-70
General Information . . . . .	5-70
Card Reader . . . . .	5-71
General Information . . . . .	5-71
Line Printer . . . . .	5-72
General Information . . . . .	5-72
Timer and Switch Channel . . . . .	5-73
General Information . . . . .	5-73
Common Peripheral Status Format . . . . .	5-74
DATANET FNP General Memory Map . . . . .	5-75
Interrupt Cells . . . . .	5-75
Processor Fault Vectors . . . . .	5-83
IOC Fault Status Locations . . . . .	5-84
Coding Examples . . . . .	5-84
BCD Addition . . . . .	5-85
BCD Subtraction . . . . .	5-86
Data Movement . . . . .	5-88
Binary to Binary Coded Decimal Conversion Routine . . . . .	5-89
Character Transliteration . . . . .	5-90
Appendix A	Standard Character Set . . . . . A-1
Appendix B	Conversion Tables . . . . . B-1
	Octal-Decimal Integer Table . . . . . B-1
	Octal-Decimal Fraction Table . . . . . B-5
Appendix C	Table of Powers of Two . . . . . C-1
Appendix D	Table of Binary-Decimal Equivalents . . . . . D-1

CONTENTS (cont)

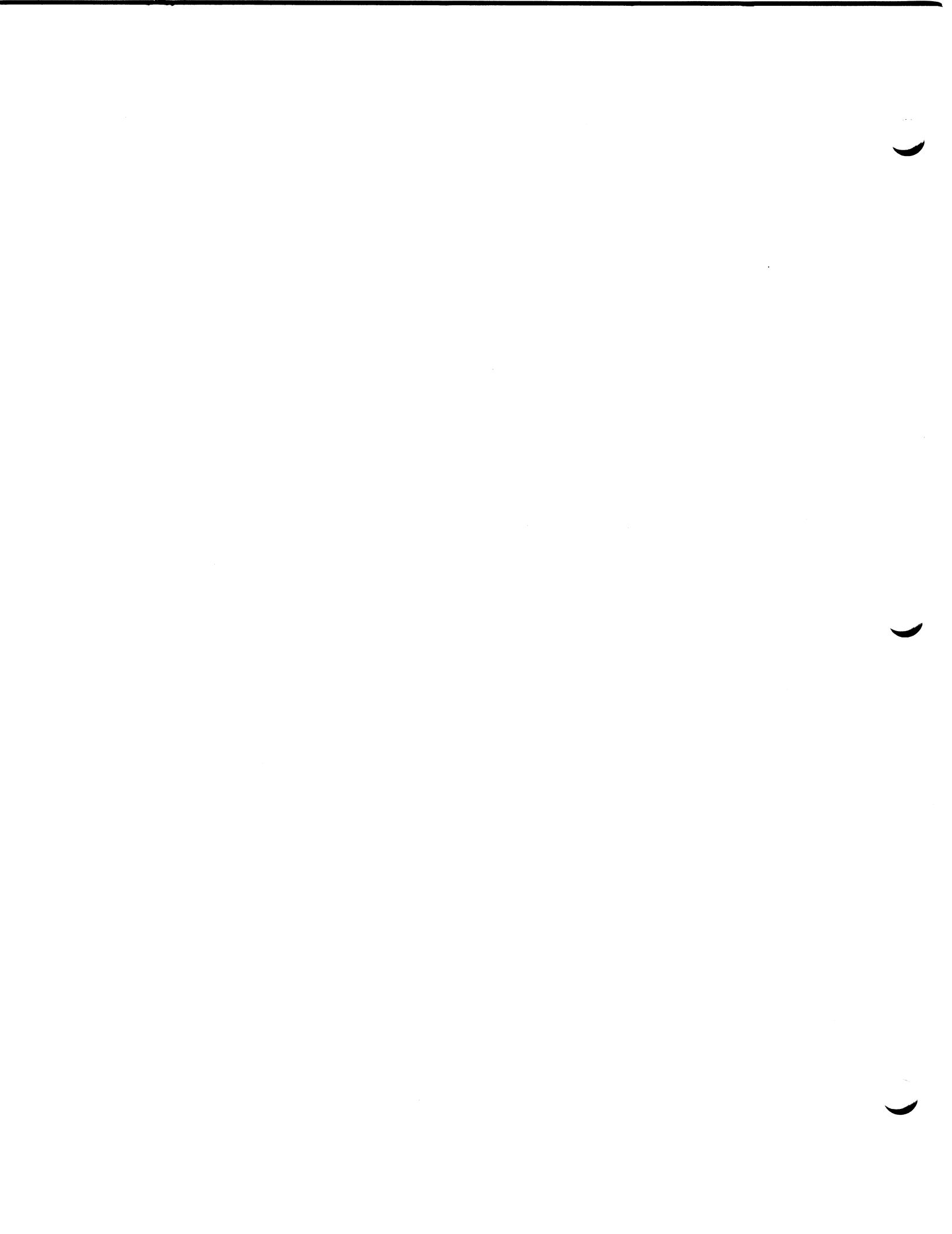
	Page
Appendix E            The Twos Complement Number System . . . . .	E-1
Index . . . . .	i-1

ILLUSTRATIONS

Figure 2-1	Symbolic Coding Form. . . . .	2-2
Figure 3-1	Ranges of Fixed-Point Numbers . . . . .	3-3
Figure 3-2	Effective Address Formation Examples, Character Addressing . . . . .	3-19
Figure 3-3	Character Address Addition Rules. . . . .	3-20
Figure 3-4	Indirect Level Word Addressing. . . . .	3-21
Figure 3-5	Indirect Level Character Addressing . . . . .	3-22
Figure 3-6	Memory Reference, Effective Address Formation Diagram. . . . .	3-36
Figure 3-7	Nonmemory Reference, Effective Address Formation Diagram. . . . .	3-51
Figure 4-1	Symbolic ASCII Symbols. . . . .	4-34
Figure 5-1	Character Control Character Addressing. . . . .	5-28
Figure 5-2	DATANET FNP Interrupt Cells . . . . .	5-76
Figure 5-3	DATANET FNP Interrupt Vectors . . . . .	5-77

TABLES

Table 3-1	Processor Faults. . . . .	3-7
Table 3-2	Memory Map - Interrupts . . . . .	3-11
Table 3-3	Indicators and Their Relation to the Bit Positions of a Memory Location . . . . .	3-12
Table 3-4	Instruction Repertoire by Functional Class. . . . .	3-23
Table 3-5	Instruction Repertoire (Alphabetical) . . . . .	3-26
Table 3-6	Instruction Repertoire (Octal Codes). . . . .	3-27
Table 3-7	Memory Reference, Fractional Add (FA) Function. . . . .	3-36
Table 3-8	Nonmemory Reference, Fractional Add (FA) Function . . . . .	3-51
Table 4-1	Pseudo-Operations by Functional Class . . . . .	4-2



## SECTION I

### INTRODUCTION

The DATANET 355/6600 Assembly Program (hereafter called MAP) is a program which translates symbolic machine language into absolute or relocatable binary machine instructions convenient for programmer use. The symbolic language is sufficiently like machine language to permit the programmer to utilize all the facilities of the computer which would be available to him if he were to code directly in machine language.

An assembler resembles a compiler in that it produces machine language programs. It differs from a compiler in that the symbolic language used with an assembler is closely related to the language used by the computer, while the source language used with a compiler resembles the technical language in which problems are stated by human beings.

Compilers have several advantages over assemblers. The language used with the compiler is easier to learn and is oriented toward the problem to be solved. The user of a compiler usually does not need an intimate knowledge of the inner workings of the computer. Programming is faster. Finally, the time required to obtain a finished, working program is greatly reduced since there is less chance for the programmer to make mistakes. The assembler compensates for its disadvantages by offering those programmers, who need a great degree of flexibility in writing their programs, that flexibility which is not currently found in compilers.

The MAP assembler gives the programmer some of the conveniences of a compiler and the flexibility of an assembler. The ability to design desired macros in order to provide convenient shorthand notations plus the use of all DATANET FNP machine instructions, as well as a complete set of pseudo-operations provides the programmer with a very powerful and flexible tool. The output options enable him to obtain binary text in relocatable as well as absolute formats.

The MAP assembler is implemented in the classic format of Macro assemblers, with several variations. There are two passes over the external text: the first pass allows for updating and/or merging of an alter package to a previously prepared assembly input. The alter package consists of changes to be made to the previous assembly under control of alter cards. During Pass 1, all symbols are collected and assigned their absolute or relocatable values relative to the current location counter. Macro prototypes are processed and placed in the macro skeleton table, immediately ready for expansion. All macro calls, therefore, are expanded in Pass 2, allowing the macro skeleton table to be destroyed prior to Pass 2.

Machine operation codes, pseudo-operations, and macro names are all carried in the operation table during Pass 1. This implies that all operation codes, machine or pseudo, along with macro names are looked up during Pass 1, and that the general operation table is destroyed at the end of Pass 1. At the end of Pass 1, the symbol table is sorted; and a complete readjustment of symbols by their relative location counter is performed.

The preface card(s) is punched at the beginning of Pass 2. All instructions are generated during Pass 2. This is accomplished by performing a scan over the variable fields and address modifications. This information is then combined with the operation code from Pass 1 by using a Boolean OR function. Apparent errors are flagged.

The symbolic cross-reference table is created as the variable fields are scanned and expanded. The final edit of the symbol table is done at the end of Pass 2. Generative pseudo-operations are processed with the conversion being done in Pass 2. Pseudo-operations are available to control punching of binary cards and printing images of source cards. Images of source cards in error will be printed, regardless of control pseudo-operations. Undefined symbols and error conditions are noted at the end of the printer listing.

The classic format of a variable field symbolic assembly program is used throughout MAP. Typically, a symbolic instruction consists of four major divisions; location field, operation field, variable field, and comments field.

The location field normally contains a name by which other instructions may refer to the instruction named. The operation field contains the name of the machine operation, pseudo-operation, or macro-operation. The variable field normally contains the location of the operand. The comments field exists solely for the convenience of the programmer and plays no part in the assembly process. An identification field is provided to give a means of identifying the location of a card within a deck.

#### RELOCATABLE AND ABSOLUTE ASSEMBLIES

The MAP assembler processes the following types of inputs:

1. Source programs written originally in assembler language
2. Compressed source decks (COMDK) for item 1
3. Correction (ALTER) cards for item 1

The normal mode of the assembler in processing input subprograms of the types indicated above is relocatable. Each subprogram in the job stream is handled individually and is assigned memory locations normally beginning with zero and extending to the upper limits required for that subprogram. Since a job stream may contain many such subprograms, it is apparent that they cannot all be loaded into a memory area starting with location zero: they must be loaded into different memory areas. Furthermore, they must be movable (relocatable) among the areas. For relocatable programs, the Assembler provides: (1) delimiters identifying each subprogram; (2) information specifying that the subprogram is relocatable; (3) the length of the subprogram and (4) relocation control bits for each assembled word. Refer to the DATANET 355/6600 Relocatable Loader reference manual for a detailed description of the MAP relocation scheme.

Subprogram delimiters are the assembler output cards. The \$ OBJECT card heads the subprogram assembly and the \$ DKEND card ends the assembly. An assembly is designated as relocatable on a card-to-card basis by a unique punched code on each binary card. The subprogram length is punched in the preface card(s) which immediately follows the \$ OBJECT card of each subprogram. The relocation control bits are grouped together in the binary card and are referenced by the Relocatable Loader while it is loading the subprogram into absolute memory locations.

The assembler designates that the assembly output is absolute on a card-to-card basis by punching a unique code value in each card. This value causes the Relocatable Loader to regard all addresses on a card as actual (physical) memory address. Each absolute subprogram assembly begins with a \$ OBJECT card and terminates with the \$ DKEND card as in the relocatable assemblies. The normal assembler operating mode is relocatable. It is set absolute by the programmer use of the ABS pseudo-operation.

### SOURCE PROGRAM INPUT

The following is the minimum required deck setup for the assembly of a program written in MAP. Refer to the Control Cards Reference Manual for more detailed information on system control cards.

1	8	16
<hr/>		
\$	SNUMB	
\$	IDENT	
\$	355MAP	Options
	.	} Source Deck
	.	
\$	UPDATE	
	.	} Alter Deck (optional)
	.	
\$	ENDJOB	
***EOF		

### \$ SNUMB Control Card

1	8	16
<hr/>		
\$	SNUMB	Job Identifier,Urgency

where:

Job Identifier is from one to five characters and must be present on the card.

Urgency is a number from 1 to 63 and represents the relative importance of the job. If omitted, a value based on the system resources needed for the job is calculated.

### \$ 355MAP Control Card

1	8	16
<hr/>		
\$	355MAP	Options

This card is used to call the assembler into memory from external storage. The variable field specifies assembler output options. The following options are available with MAP (standard options are underlined):

LSTOU - A listing of the assembled object program output will be prepared.



- NLSTOU - No listing of the assembled object program output will be prepared.
- DECK - A binary object program deck will be prepared as output.
- NDECK - No binary object program deck will be prepared as output.
- GMAC - The GRTS System Macro operations will be loaded into the MAP macro prototype area.
- NGMAC - The GRTS System Macro operations will not be loaded.
- ON5 - Print all source images regardless of any pseudo-operations that might otherwise result in their not being printed.
- DUMP - Slave memory dump will be produced if activity terminates abnormally.
- NDUMP - A panel dump of program registers, slave prefix, and upper SSA (Slave Service Area) will be produced if activity terminates abnormally.
- COPY - The binary object program deck produced by MAP will be copied to the \*B file, and SIM (DATANET 355/6600 Simulator Program) will be called upon the completion of the assembly.
- NCOPY - No binary object program deck will be copied to the \*B file, and MAP will return control to GCOS upon the completion of the assembly.
- COMDK - A compressed deck of the source program will be prepared as output.
- NCOMDK - No compressed deck of the source program will be prepared as output.
- NXEC - If the COPY option is specified and an assembly error is encountered, NXEC causes the COPY option to be ignored.
- SYMTAB - Prepare a listing of the Symbol Reference Table (if one has been built) even though NLSTOU is specified.

\$ LIMITS Control Card

1	8	16	
\$	LIMITS	20,32000,,10000	

This control card assigns non-standard activity limits for the MAP assembly. The first parameter is the processor run time for the activity in hundredths of an hour. The second parameter is the memory required for the activity. The fourth parameter is the maximum number of printed lines of SYSOUT. If not specified this value is 5120.

\$ UPDATE Control Card (optional)

1	8	16	73	80
_____			_____	
\$	UPDATE	LIST	Identification	

The \$ UPDATE control card is used when supplying alter input to the assembler. The operand field is used when a listing of the \$ ALTER cards is required (LIST option). Any \$ ALTER card out of order is listed as an alter file error. The Identification field is optional and can be used to place an identifier in columns 73-80 of each source card contained in the Alter File (A\*).

\$ ALTER Control Card

1	8	16
_____		
\$	ALTER	M,N

The \$ ALTER control card is used to make correction to the source card or COMDK input. Source cards can be inserted or deleted, or can replace other source cards from the input file to MAP by specifying the respective alter numbers in the operand field.

The operand field contains alter numbers M,N taken from a previous assembly listing of the job to which changes are to be made. The alter numbers are consecutive card numbers starting with 00001 and increasing by one for each source input card.

When it is desired to insert cards into a deck, the N subfield is not used. In this case, the source cards following this \$ ALTER card or other \$ control card are inserted just prior to the card corresponding to alter number N.

When it is desired to delete and/or replace one or more cards from a deck, the N subfield is given as shown above. When M and N are equal, card M is deleted. When N identifies a card following M, all cards, M through N, are deleted. In addition, any cards following the \$ ALTER card up to but not including the next \$ ALTER card (or other control card) will be inserted in place of the deleted cards.

\$ ENDJOB Control Card

1	8	16
_____		
\$	ENDJOB	Not Used

The \$ ENDJOB card is used to indicate that the job is a candidate for allocation and execution, provided errors are not detected.

\*\*\*EOF Control Card

1            8            16  
-----

\*\*\*EOF

This must be the last card of every job.

TYPICAL DECK SETUPS

Assemble with no compressed deck or binary deck produced.

1            8            16  
-----

```
$        SNUMB
$        IDENT
$        355MAP        NCOMDK,NDECK
          .        }        Source Deck
          .        }
          .        }
$        ENDJOB
***EOF
```

Assemble a compressed deck with alters and produce a new compressed deck.

1            8            16  
-----

```
$        SNUMB
$        IDENT
$        355MAP        COMDK
          .        }        Compressed Source Deck
          .        }
          .        }
$        UPDATE        LIST
$        ALTER
          .        }        Alter Deck
          .        }
          .        }
$        ENDJOB
***EOF
```

ASSEMBLER OUTPUTS

The MAP assembler outputs, based on the options specified on the \$ 355 MAP control card, are binary decks, assembly listing, and compressed decks.

## Binary Decks

Unless the NDECK option is specified on the \$ 355 MAP card, the Assembler punches a binary deck in relocatable, or absolute mode. The first card generated by MAP for every object program is a \$ OBJECT card. The \$ OBJECT control card has the following format:

1	8	16	61	67	73	80
\$	OBJECT	Optional Comment, Sequence Options	Time of Assembly or blank	Date of Assembly or TTL Date	Optional Label	

The Optional Comment Sequence Option subfield is either a product of the second subfield of the LBL pseudo-operation (described in the section on pseudo-operations) or the entries may be inserted by the programmer. When a sequence checking option is not specified, the Optional Label subfields of all cards in a \$ OBJECT deck are sequence checked and the activity deleted in case of an error. When an error is detected, a message is printed on the execution report. The following sequence checking option, SEQ, is assumed if no option is specified:

SEQ - Check sequence and delete the activity if an error occurs.  
CKSEQ - Check sequence and flag errors but do not delete the activity.  
NSEQ - No sequence check.

Time of Assembly is in hours and thousandths of hours in the form XX.XXX. This time appears in the page heading of the associated listing. If a TTL card with a TTL date entry is present in the assembly, this field is a blank.

Date of Assembly subfield is in the form mmddy. If a TTL card with a TTL date entry is present in the assembly, the TTL date from that card is placed in this field by the assembler.

The Optional Label subfield is a product of the first subfield of the LBL pseudo-operation. It is an alphanumeric identification number designating the object program or subprogram. If not specified, it is produced starting at 0000000.

The last card generated by MAP for every object program is the \$ DKEND control card. It has the following format:

1	8	16	61	67	73	80
\$	DKEND	blank	Time of Assembly	Date of Assembly	Optional Label	

The assembler places the time of assembly in columns 61-66 in the form xx.xxx hours, and the date of assembly in columns 67-72 in the form mmddy. The Optional Label field is the same field as described under the \$ OBJECT control card with an appropriate numeric sequence number.

## Relocatable Object Decks

Relocatable object decks contain, in addition to the \$ OBJECT and \$ DKEND cards, cards with preface information and text information.

### Preface Cards

Preface cards provide the Relocatable Loader with pertinent size and linkage information, such as:

- Length of the subprogram text region
- Length of Blank Common area required, if any
- Total number of SYMDEF, SYMREF, and Labeled Common symbols as well as the symbols themselves.
- The relative entry value of the region length for each symbol.

The contents of each word on the Preface cards is as follows:

Words 1-2

0	23	89	1112	1718	35
100	n1	101	n2	n3	

Bits 0-2 and 9-11 identify the card as a binary preface card.

Subfield n1 contains the number of bits required to express the total number of Labeled Common and SYMREF symbols referenced within the subprogram. This number is calculated as follows:

$$n1 = \log(N+1)$$

where N is the count of symbols and the range of n1 is  $5 \leq n1 \leq 14$ .

Subfield n2 contains the count of words on the Preface card beginning with word 5.

Subfield n3 is the length of the subprogram text.

Words 3-4

Checksum of columns 1-3 and 7-72

Words 5-6

0	1718	2021	35
A	M	N	

Subfield A contains the length of Blank Common area required by the subprogram.

Subfield M, if non-zero, specifies loading of the subprogram should start at the next available address which is a multiple of the modulo of two selected. The values for M are as follows:

M	Modulo value
1	8
2	16
3	32
4	64
5	128
6	256
7	not used

If M is equal to zero, the subprogram is loaded starting at the next even location.

Subfield N contains two times the number of SYMDEF, SYMREF, and Labeled Common symbols contained on the Preface card(s). If N > 20, additional Preface cards are required. On additional Preface cards, words 5 and 6 are not changed. Words 7 through 46 contain symbol entries as follows:

Words n, n+1

0	56	1112	1718	2324	2930	35
C1	C2	C3	C4	C5	C6	

Words n+2, n+3

0	1718	2021	3233	35
A	M		K	

The first two words (n,n+1) of each entry is a symbol. The second two words (n+2, n+3) describe that symbol completely as to its usage by the subprogram being loaded. The value K (bits 33-35) defines the type of symbol and thus, the meaning of the other fields involved:

- K=0 - The symbol is a primary SYMDEF. Subfield A contains a value equal to the position of the symbol relative to the beginning of the subprogram. M is not used.
- K=1 - The symbol is a secondary SYMDEF. Subfield A contains a value equal to the position of the symbol relative to the beginning of the subprogram. M is not used.
- K=5 - The symbol is a SYMREF. Subfield A is zero. M is not used.
- K=6 - The symbol is the name of a Labeled Common region. Subfield A contains the length of the region and cannot be zero. If M is non-zero, the region is assigned beginning at the next available address that is a multiple of the modulo of two selected. (Same meaning as in words 5-6.) If M is zero, the region is assigned beginning at the next even location.

## Relocatable Text Card

Text cards contain information required to execute the desired function. This information is formatted to give the Relocatable Loader the necessary parameters to form a useful block of data, or to form executable instructions in memory.

A relocatable text card contains the following:

Words 1-2

0	23	89	1112	1718	35
010 <sup>10</sup>	n1	101	n2	n3	

Bits 0-2, 9-11 define the card as a column binary relocatable text card.

Subfield n1 indicates the symbol (obtained from the Preface card) relative to which this text is to be loaded. If n1 is zero, the text is loaded relative to the primary program region. Where  $1 \leq n1 \leq$  number of Labeled Common symbols in the Preface card, n1 then specifies the symbolic address relative to which the text is to be relocated.

Subfield n1 contains a count of the number of instructions associated with this control word. The count does not include the five words of relocation data and is not necessarily a count of the words on the card.

Subfield n1 is the relative loading address under the load counter specified by n3.

Words 3-4

Checksum of columns 1-3, 7-72.

Words 5-9

Relocation data. Each of words 5-8 contains nine 2-bit relocation identifiers. Word 9 contains three identifiers. Each 2-bit identifier contains relocation information for each instruction or data word in the text of the card as follows:

- 00 - Absolute - no relocation applied.
- 01 - Relocatable - relocate relative to the load address of the subprogram.
- 10 - Blank Common - relocate relative to the beginning of the Blank Common region.
- 11 - Special Relocation - relocate relative to the preface entry encoded in the word (SYMREF or Labeled Common).

Words 10-48 contain instructions and data (maximum of 39 words per card). If the number of available instructions or data words on the card are not completely used by the n specified in the control word (words 1-2), and at least four words are left vacant on the card, then new control words (see format of words 1-2) may appear after the last utilized word. The new control word indicates a new word count n2 and a new loading address n3. The loading is then continued with the new address and with the relocation data continuously retrieved from words 5-9. The new control words do not have relocation bits associated with them. This process may be repeated as often as necessary to fill the card.

Absolute Object (Binary) Decks

An absolute object deck consists of one or more absolute text cards and a transfer card. Absolute text cards provide the Loader with binary text and an absolute starting location to be used in assigning memory locations to the contents of the card.

Absolute Text Card

This card contains the following:

Words 1-2

0	23	89	1112	1718	35
001	n1	101	n2	n3	

Bits 0-2, 9-11 identify the card as a column binary absolute text card.

Subfield n1 is zero  
 Subfield n2 contains a word count  
 Subfield n3 contains an absolute address

Words 3-4

Checksum of columns 1-3, 7-72.

Words 5-48

Instructions and text (maximum of 44 words per card). If the number of instructions or data words is not complete, and at least four words are available, words 1 and 2 may be repeated after the last word. These words contain a new word count n2 and a new loading address n3.



## Transfer Card

The Transfer Card is generated automatically, as the last card of an absolute subprogram assembly, by the END pseudo-operation. The Transfer Card specifies to the Loader the entry location for the program. The Transfer Card contains the following:

Words 1-2

0	23	89	1112	1718	35
000	n1	101	n2	n3	

Bits 0-2, 9-11 identify the card as an absolute transfer card.

Subfield n1 is zero

Subfield n2 is zero

Subfield n3 contains the transfer address

Words 3-48

Not used.

## Assembly Listing

Each assembler listing consists of the following parts:

1. The contents of all preface cards (primary SYMDEF symbols, secondary SYMDEF symbols, SYMREF symbols, Labeled Common symbols -- from the BLOCK pseudo-operation). This section is omitted from an absolute assembly.
2. The sequence of instructions in order of input to the assembler.
3. The symbolic reference table.

## Full Listing Format

Each instruction word produced by the assembler is printed on a 132-character line. The line contains the following items for each such word of all symbolic cards:

1. Error Flags - One character for each error type (See "Error Codes" in this section).
2. Octal location of the assembled word.
3. Octal representation of the assembled word.
4. Relocation bits for the assembled word. If the word is a memory reference instruction with instruction counter modification, the relocation bits (0) are replaced with the address of the operand. This value is placed in columns 27-31 right adjusted, with blanks for leading zeros. (See "Relocatable and Absolute Expressions" in Section II of this manual.)

5. Reproduction of the symbolic card, including the comments and identification fields, exactly as coded.
6. Mapping data. Information to indicate the applicabilty of each record listed.

The format of the full listing is:

Fields	A	B	C	D	E	F	G	H	I
Columns	1-6	8-12		15-23		26-31	35-39	41-120	122-132
				⏟				⏟	
				Machine Instruction				Source Card Image	

A = Error flags  
 B = Relative absolute location  
 C = Tag  
 D = Operation code  
 E = Displacement field  
 F = Relocation bits  
 G = Alter statement number  
 H = Source card image  
 I = Compressed deck and Alter Deck Mapping Data

For field I, the data for each record has one of the following formats:

xxxxx	= unmodified from input COMDK
xxxxxRyyyyy	= xxxxx replaced (R) by yyyyy
Nyyyyy	= new (N) alter number yyyyy
xxxxM,xxxxNDyyyyy	= alters from xxxxm to xxxxn deleted (D) by yyyyy
(blank)	= not present in either COMDK or A* alter file

The mapping data is printed on the listing if the following conditions apply:

- No output compressed deck is being produced
- An A\* alter file is present for the assembly

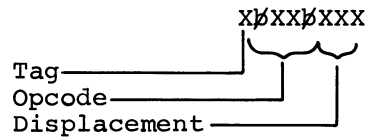
These data on the assembly listing make it feasible to develop software from a reference base without working from successively new compressed decks.

Several variations appear for the machine instruction (fields C, D, E - bits 15-23). These are summarized below:

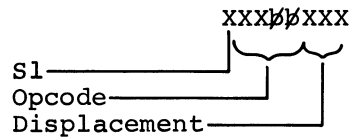
Type of Machine  
Word

Listing Format

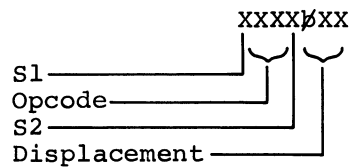
Memory Reference Instruction



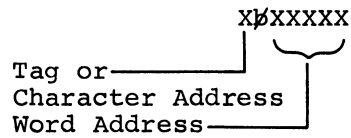
Nonmemory Reference Instruction  
Group 1



Nonmemory Reference Instruction  
Group 2



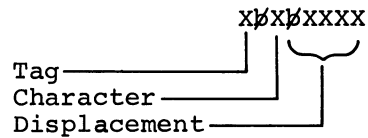
IND/ZERO Pseudo-Operation



Data Generating Pseudo-Operation

XXXXXX

IND/A,T, Character



## Preface Card Listing

The listing of the preface information is in a self-explanatory format, with each major subdivision of preface symbols preceded by a heading. The order is the same as that of the card(s) produced.

### SYMDEF, SYMREF, Labeled Common

All SYMDEF symbols are listed with their name and octal location within the listing, in ascending location order. All primary SYMDEFs are listed first and then all secondary SYMDEFs.

The Labeled Common and SYMREF symbols are numbered sequentially 1 through n, where the number represents the special relocation entry number employed in referencing those special symbols.

### Blank Common

Prior to the listing of the special symbols, the assembler enters a statement of the amount of Blank Common memory requested by the subprogram. The statement format is self-explanatory.

### Symbolic Reference Table

The symbol table contains all symbols used, the octal values (normally the location value), and the alter number of all instructions that reference the symbol. The table format is as follows:

<u>Octal</u>	<u>Symbol</u>	<u>References by Alter No.</u>
2233	CNTR	216 159 164 216

The example above indicates that symbol CNTR has been assigned the value 2233 (octal) and is referenced in three places. The first alter number (216) is the point in the instruction listing where the symbol is defined.

## Error Flags

The following list comprises the MAP error flags for individual instructions and pseudo-operations:

<u>Error</u>	<u>Flag</u>	<u>Cause</u>
Undefined	U	Undefined symbol(s) appear in the variable field.
Multidefined	M	Multiple-defined symbol(s) appear in the location and/or variable field.
Address	A	Illegal value or symbol appears in the variable field. Also used to denote the lack of a required field.
Index	X	Illegal index or address modifier.
Relocation	R	Relocation error; expression in the variable field will produce a relocatable error upon loading.
Phase	P	Phase error: this implies undetected machine errors or symbols defined in Pass 2 with a different value than defined in Pass 1.
Even	E	Inappropriate character in column 7.
Conversion	C	Error in conversion of a subfield of a data generating pseudo-operation. Illegal character.
Location	L	Error in the location field.
Operation	O	Illegal operation.
Table	T	An assembly table overflowed preventing the proper processing of this card. Table overflow error information appears at the end of the listing.

MAP also prints out the following error messages.

SYMBOL TABLE OVERFLOW  
MACRO EXPANSION TABLE OVERFLOW  
MACRO PROTOTYPE TABLE OVERFLOW  
NO END CARD ON INPUT FILE  
SYMBOL REFERENCE TABLE OVERFLOW  
EXECUTION NOT POSSIBLE, NO SYMDEF  
TOO MANY CARDS TO BE DUPLICATED  
OPERATION TABLE OVERFLOW  
UNEXPECTED EOF ON INTERMEDIATE FILE  
NOT ENOUGH CARDS TO BE SKIPPED  
NEXEC OPTION SPECIFIED Fatal Error...Simulation Deleted  
ERROR IN MACRO EXPANSION Assembler will attempt to Recover

## SECTION II

### ASSEMBLY LANGUAGE PROGRAMMING

Program statements (source statements) written in assembler language consist of three types:

1. Machine Instructions
2. Pseudo-Operations
3. Macro Instructions

Machine instructions are one-for-one symbolic representations of the DATANET FNP instructions. The assembler produces one machine instruction in the object program for each machine instruction statement in the source program.

Pseudo-operations are so called because of their similarity to machine operations in an object program. Unlike machine operations, which perform some part of a task directly concerned with solving a problem, pseudo-operations work indirectly on the problem by performing machine conditioning functions and by directing the assembler in the preparation of machine coding.

A Macro instruction statement causes the assembler to retrieve a specially coded symbolic routine, modify the routine according to the information in the Macro statement, and insert the modified routine into the source program for translation into machine language.

#### SYMBOLIC CODING FORM

A source program is a sequence of source statements that are punched onto cards. These statements are written on the standard coding form (Figure 2-1). Each line of coding on the form is punched into one card.

The body of the form contains six fields as follows:

1. Location field in columns 1-6
2. E/O/8 (Even/Odd/Eight) field in column 7.
3. Operation field in columns 8-15
4. Variable field in columns 16-31
5. Comments field in columns 32-72
6. Identification field in columns 73-80



### Location Field

This field may (in machine instructions or Macro's) contain a symbol or may be blank. In certain pseudo-operations, described later, this field has a special use.

### E/O/8 Field

This 1-character field allows the programmer to specify that this generated machine word falls in an even location, an odd location, or a location which is a multiple of eight. If this field is left blank, the instruction will be located in the next available location.

### Operation Field

The operation field can contain from zero to six characters selected from the set 0-9, A-Z, and the period (.). The operation field may contain: (1) a legal DATANET FNP machine instruction, (2) an assembler pseudo-operation, or (3) a programmer macro-operation code. The character group must begin in column eight (left justified) and must be followed by at least one blank.

A blank field or the special code ARG is interpreted as a zero operation and the operation field is all zeros in the assembly coding. Anything appearing in the operation field which is not in (1), (2), or (3) above is an illegal operation and results in an error flag in the assembly listing.

### Variable Field

The variable field can contain one or more subfields separated by commas. The number and type of subfields vary depending on the machine-operation instruction, pseudo-operation, or macro-operation.

The subfields within the variable field of machine instructions depend on the instruction type. All subfields, regardless of number, are comma separated. For memory reference machine instructions the format is:

16

---

Displacement Address, Tag, Character Address



where:

The displacement address may be an absolute decimal number or an absolute expression

The tab may be 1, 2, 3, I or \*

The character address may be B.0, B.1, C.0, C.1 or C.2. These character addresses have the following definitions:

<u>Symbol</u>	<u>Symbol Synonym</u>	<u>Value</u>	<u>Meaning</u>
B.0	0.2	2	9-bit data transmission, character 0
B.1	1.2	3	9-bit data transmission, character 1
C.0	0.3	4	6-bit data transmission, character 0
C.1	1.3	5	6-bit data transmission, character 1
C.2	2.3	6	6-bit data transmission, character 2

For nonmemory reference machine instructions Group 1 the format is:

16

Displacement Address, Character Address

For Group 1, the displacement address and character address, is specified by the same conventions listed for the memory reference instructions.

For nonmemory reference machine instructions Group 2 the format is:

16

Shift Count or Null

For Group 2, the shift count is a decimal number specifying the number of bit positions to shift, rotate, etc, data in the operational registers.

The variable subfields used with pseudo-operations are described individually with the pseudo-operation. Subfields used with macro-operations are substitutable arguments which, in themselves, may be instructions, operand addresses, modifier tags, pseudo-operations, or other macro-operations. These subfields are discussed with the macro-operations.

The first character of the variable field must begin in column 16. The end of the variable field is determined by the first blank character encountered (except for data generating pseudo-operations). If any subfield is null (no entry given when one is needed), it is interpreted as zero.

#### Comments Field

The comments field is for the convenience of the programmer and plays no part in the assembly process. Programmer comments follow the variable field and are separated from that field by at least one blank column.

## Identification Field

This optional field is used by the programmer for instruction identification and sequencing.

## Summary of Symbolic Card Format

The following is a breakdown of the card columns normally used.

<u>Card Columns</u>	<u>Subfield</u>	<u>Contents</u>
1-6	Location	Symbol, blank, *
7	Even/odd/eight	E, O, 8
8-13	Operation	Machine Instructions, Pseudo-operation, macro-operation
14-15	Blank	
16-blank	Variable field	Address - Tag for machine instruction. Special fields for macro-operations and pseudo-operations (see individual descriptions).
Blank-72	Comments	Optional comments (Separated from variable field by at least one blank).
73-80	Identification	Optional

When columns 1-16 are blank, the symbolic card is treated as a remarks card.

## LANGUAGE STRUCTURE

### Character Set

Assembler language statements may be written using the character set shown in Appendix A.

### Program Symbols

A symbol is a string of from one to six nonblank characters, at least one of which is nonnumeric, and the first of which is nonzero. The characters may be taken from the set A-Z, 0-9, and the period (.). Symbols can appear in the location and variable fields of the assembler coding form. Symbols are also known as location symbols and symbolic addresses. Symbols are defined by:

1. Their appearance in the location field of an instruction, pseudo-operation, or Macro-operation
2. Their use as the name of a subprogram in a CALL pseudo-operation
3. Their appearance in the SYMREF pseudo-operation.

Every symbol used in a program must be defined exactly once, except for those symbols which are initially defined and redefined by the SET pseudo-operation. An error will be indicated by the assembler if any symbol is referenced but never defined, or if any symbol is defined more than once.

The following are examples of permissible symbols:

A	A1000	ELXP3	A.....
Z	FIRST	.XP3	B.707
B1	ALOG10	ADDTO	1234X
ERR	BEGIN	ERROR	3.141P

### Types of Program Symbols

Symbols are classified into four types:

1. Absolute--A symbol which refers to a specific number.
2. Common--A symbol which refers to a location in common memory. These locations are defined by the use of the BLOCK pseudo-operation.
3. Relocatable--A symbol which appears in the location field of an instruction. Symbols that appear in the location field of symbol defining pseudo-operations are defined as the same type as the symbol in the variable field.
4. SYMREF--A symbol which appears in the variable field of a SYMREF pseudo-operation; it is considered to be defined external to the subprogram being assembled, and is accorded special handling by the loader.

### EXPRESSIONS IN GENERAL

In writing symbolic instructions, the use of symbols only in the allowable subfields presents the programmer with too restrictive a language. Therefore, in the notation of subfields of machine instructions, and in the variable fields of pseudo-operations (and by following specific rules), the use of expressions as well as symbols is permitted. Before discussing expressions, it is necessary to describe the building blocks used to construct them. These building blocks are elements, terms, and operators.

#### Elements

The smallest component of a complete expression is an element. An element consists of a single symbol or an integer less than  $2^{17}$ . (The asterisk may also be used as an element.)

## Terms and Operators

A term is a string composed of elements and operators. It may consist of one element or, generally speaking,  $n$  elements separated by  $n - 1$  operators of the type  $*$  and  $/$ , where  $*$  indicates multiplication and  $/$  indicates division. If a term does not begin with an element or end with an element, then a null element will be assumed. It is not permissible to write two operators in succession or to write two elements in succession.

Examples of terms are:

M	MAN*T	7*Y
436	BETA/3	A*B*C/X*Y*Z
START	4*AB/ROOT	ONE*TWO/THREE

## Asterisk Used as an Element

An asterisk ( $*$ ) may be used as an element in addition to being used as an operator. When it is used as an element, it refers to the location of the instruction in which it appears.

For example, the instructions

		TRA	2
	TEMP	BSS	1
	AB	LDA	-1
are equivalent to			
		TRA	AB-*
	TEMP	BSS	1
	AB	LDA	TEMP-*

and represent a transfer to the second instruction following the transfer instruction and an accumulator load from the location preceding the load instruction. There is no ambiguity between this usage of the asterisk as an element and its use as the operator for multiplication since the position of the asterisk always makes clear what is meant. Thus,  $**M$  means "the location of this instruction multiplied by the element  $M$ ." The  $**$  means "the location of this instruction times the null element" and would be equal to zero. The notation  $*-*$  means "the location of this instruction minus the location of this instruction." (See "Algebraic Expressions" and "Boolean Expressions".)

## Algebraic Expressions

An algebraic expression is a string composed of terms separated by the operators + (addition) and - (subtraction). Therefore, an expression may consist of one term or, more generally speaking, n terms separated by n - 1 operators of the type + and -. It is permissible to write two operators, +, and -, in succession and the assembler will assume a null element between the two operators. If no initial term or final term is stated, it will be assumed to be zero, except when the divisor is zero, in which case the divisor is assumed to be 1. An expression may begin with the operator + or -. However, if not explicitly given, + will be assumed. Examples of permissible algebraic expressions are:

A	B+4		CY*DY+EX/FY-100
SINE	7		-EXP*FUNC/LOGX+XYZ/10-SINE
XYZ	-99	-X/Y	*+5*X (the first asterisk refers to the instruction location)
A-3	-88	X*Y	--(equivalent to zero minus zero)

## Evaluation of Algebraic Expressions

An algebraic expression is evaluated as follows: first, each symbolic element is replaced by its numerically defined value; then, each term is computed from left-to-right in the order of its occurrence. In division, the integral part of the quotient is retained; the remainder is immediately discarded. For example, the value of the term  $7/3*3$  is 6. In the evaluation of an expression, division by zero is equivalent to division by one and is not regarded as an error. After the evaluation of terms, they are combined in a left-to-right order with the initial term of the expression assumed to be zero followed by a plus operator. If there is no final term, a null term will be used. At the completion of the expression evaluation, the assembler reduces the result by modulo  $2^n$  where n is the number of binary bits in the field being defined. Grouping by parentheses is not permitted, but this restriction may often be circumvented.

## Boolean Expressions

A Boolean expression is defined similarly to an algebraic expression except that the operators \*, /, +, or - are interpreted as Boolean operators. The meaning of these operators is defined below.

1. The expression that appears in the variable field of a BOOL pseudo-operation uses Boolean operators.
2. The expression that appears in the octal subfield of the variable field of a VFD pseudo-operation uses Boolean operators.

## Evaluation of Boolean Expressions

A Boolean expression is evaluated following the same procedure used for an algebraic expression except that the operators are interpreted as Boolean.

In a Boolean expression, the form operators +, -, \*, and / have Boolean meanings, rather than their normal arithmetic meanings, as follows:

<u>Operator</u>	<u>Meaning</u>	<u>Definition</u>
+	OR, INCLUSIVE OR, union	$0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 1$
-	EXCLUSIVE OR symmetric difference	$0 - 0 = 0$ $0 - 1 = 1$ $1 - 0 = 1$ $1 - 1 = 0$
*	AND, intersection	$0 * 0 = 0$ $0 * 1 = 0$ $1 * 0 = 0$ $1 * 1 = 1$
/	1's complement, NOT	$/0 = 1$ $/1 = 0$

Although / is a unary operation involving only one term, by convention A/B is taken to mean A\*/B. This is not regarded as an error by the assembler. Thus, the table for / as a two-term operation is:

$0/0 = 0$	$1/0 = 1$
$0/1 = 0$	$1/1 = 0$

Other conventions are:

+A = A+ = A	
-A = A- = A	
*A = A* = 0	(possible error--operand missing)
A/ = A/0 = A	

## Relocatable and Absolute Expressions

Expression evaluation can result in either relocatable or absolute values. There are three types of relocatable expressions; program relocatable (R), Blank Common relocatable (C), and Labeled Common relocatable (L). The rules by which the assembler determines the relocation validity of an expression are complex, and the presence of multiple location counters compounds the problem. Certain of the principal pseudo-operations impose restrictions as to the type of expression that is permissible; these are described separately under each of the affected pseudo-operations:

BEGIN	BOOL	DUP	FEQU	MIN	SET
BFS	BSS	EQU	MAX	ORG	

The following rules summarize the conditions and restrictions governing the admissibility of relocation:

1. Division involving a relocatable element(s) is not valid.
2. Multiplication of two relocatable elements is not valid.
3. The asterisk (\*) symbol (implying current location counter) is a relocatable element.
4. When the result of the evaluation of an expression is an absolute element, the expression is absolute.
5. When the result of the evaluation of an expression is a relocatable element, the expression is relocatable.
6. When the result of the evaluation of an expression is the sum or difference of a relocatable element and an absolute element, the expression is relocatable.
7. When the result of the evaluation of an expression is the difference between two relocatable elements, the expression is absolute.

As the result of the evaluation of an expression:

1. The sum of two or more relocatable elements is not valid.
2. The product of an absolute element and a relocatable element is not valid.
3. A negative relocatable element is not valid.
4. The difference of two different types of relocatable elements is not valid.

These rules are not a complete set of determinants but do serve as a basis for establishing a method of defining relocation admissibility of an expression.

Let Rr denote a program-text relocatable element, Rc denote a Blank Common element, and Rl denote a Labeled Common element. Next, take any expression and process it as follows:

1. Replace all absolute elements with their respective values.
2. Replace any relocatable element with the proper Ri, where i = r, c, or l. This yields a resulting expression involving only numbers and the terms Rr, Rl, and Rc.
3. Discard all terms in which all elements are absolute.
4. Evaluate the resulting expression. If it is zero or numeric, the original expression is absolute; if it is explicitly Rr, Rc, or Rl, then the original expression is normal relocatable. Blank Common, relocatable, or Labeled Common relocatable, respectively.
5. If the resulting expression is not as given in 4 above, it is a relocation error and/or an invalid expression.

In the illustrative examples following, assume ALPHA and BETA to be normal relocatable elements (Rr), GAMMA and DELTA to be Blank Common relocatable elements (Rc), and EPSILON and ZETA to be Labeled Common relocatable elements (R). Let N and K be absolutely equivalent to 5 and 8, respectively.

1.  $4*ALPHA-7-4*BETA$   
reduces to  
 $4*Rr-4*Rl=0$ ,  
thus indicating a valid absolute expression.
2.  $N*ALPHA+8*GAMMA+21 - K*DELTA$   
reduces to  
 $5*Rr+8*Rc-8*Rc=5Rr$ ,  
thus indicating an invalid expression.
3.  $EPSILON+N-ZETA$   
reduces to  
 $Rl+5-Rl=5$ ,  
thus indicating a valid absolute expression.
4.  $ALPHA-GAMMA+DELTA+7$   
reduces to  
 $Rr-Rc+Rc=Rr$ ,  
thus indicating a valid relocatable expression.

### Special Relocatable Expressions

All symbols defined as other than equal to some number (A EQU 4) are defined relative to some explicit or implied location counter (USE, BLOCK) and are subject to adjustment at the end of Pass 1. Therefore, they are considered to be relocatable in Pass 1, even in an absolute assembly.



Thus, special action must be taken, if they are to be referenced and used in Pass 1 by certain pseudo-operations--those which call for an expression evaluation for the determination of some count subfield, the result of which must be absolute. As an example, consider

```
BCI          6,HOLLERITH TEXT
DUP          5,2
```

Normally, the count fields in the above are nonvariant and there is no problem. Consider however

```
M           BCI          N,HOLLERITH TEXT
           DUP          N,M-1
```

The assembler is equipped to handle expressions in these count fields, provided the result is absolute. But, since M in the above example is a location symbol, and its value relative to the origin of the USE is all that is known in Pass 1, a relocation error would result. The solution to this problem is simply to define some symbol at the first available location of the counter in question. It has a value of zero relative to the origin of that counter and may be used as follows:

```
FIRST      USE          CTR
           NULL
           .
           .
           .
M          BCI          N,HOLLERITH TEXT
           DUP          N,M-FIRST-1
```

The result of this expression is now absolute, and truly represents the Pass 1 value of the symbol M (less 1).

### LITERALS

A literal in a subfield is defined as being the data to be operated on rather than an expression which points to a location containing the data. In MAP, literals are permitted in the nonmemory reference instructions and the following pseudo-operations:

```
CALL      ICW          ZERO
DCW       IND
```

All other uses of literals will result in a warning flag. The nonmemory reference literals are of an immediate type and do not cause entries to be made in the literal pool.

The assembler retains pseudo-operation literals by means of a table called a literal pool. When a pseudo-operation literal appears, the assembler prepares a constant which is equivalent in value to the data in the literal subfield. This constant is then placed in the literal pool, providing an identical constant has not already been so entered. If the constant is placed in the literal pool, it is assigned an address; and this address then replaces the data in the literal subfield, the constant being retained in the pool. If the constant is already in the literal pool, the address of the identical constant replaces the data in the literal subfield.

The assembler processes six types of literals: (1) decimal, (2) octal, (3) ASCII, (4) alphanumeric, (5) instruction, and (6) variable field. The appearance of an equal sign in column 16 of the variable field, instructs the assembler that the subfield immediately following is a literal. For pseudo-operation literals (except CALL) all types of literals are permitted. Instruction and variable field literals are not permitted for nonmemory reference instructions. The CALL pseudo-operation is restricted to decimal, octal, and alphanumeric literals where the character count is less than seven.

The instruction and variable field literals are placed in the literal pool. Because they cannot be evaluated until Pass 2 of the assembly; no attempt is made to check for duplicate entries into the pool. For alphanumeric literals with 7 or more characters and Symbolic ASCII (SACI) literals, no evaluation is done until Pass 2; however, space is reserved in the literal pool.

### Decimal Literals

1. Integers -- A decimal integer is a signed or unsigned string of digits. It is differentiated from the other decimal types by the absence of a decimal point, the letter B, the letter E, and the letter D.
2. Single-Precision Floating-Point -- A floating-point number is distinguished by the presence of an E, a decimal point, or both. A floating-point number consists of two parts: a principal part and an exponent. The presence of the exponent is optional. The principal part is a signed or unsigned decimal number with a decimal point in any position of the number, or with an assumed decimal point at the right-hand end of the number. If there is no exponent part, the decimal point may not be assumed, but must be present.

The exponent part follows the principal part and consists of the letter E followed by a signed or unsigned decimal integer.

3. Double-Precision Floating-Point -- The format of the double-precision floating-point number is identical with the single-precision format with two exceptions:
  1. There must always be an exponent.
  2. The letter E must be replaced by the letter D.

The assembler will ensure that all double-precision numbers begin in even memory locations. Ambiguity of storage assignment as to even or odd will always cause the assembler to force the first half of double-precision word pairs to even locations; it will then issue a warning in the printout listing.

4. Fixed-Point -- A fixed-point quantity possesses the same characteristics as the floating-point -- with one exception: it must have a third part present. This is the binary scale factor denoted by the letter B followed by a signed or unsigned integer. The binary point is initially assumed at the left-hand end of the word between bit positions 0 and 1. It is then adjusted by the binary scale factor, designated with plus implying a shift to the right and with minus, a shift to the left. Double-precision fixed-point follows the rules of double-precision floating-point with addition of the binary scale factor.

Examples of decimal literals are:

=-10	Integer
*=26.44167E-1	Single-precision floating-point
=1.27743675385D0	Double-precision floating-point
=22.5B5	Fixed-point

### Octal Literals

The octal literal consists of the character 0 followed by a signed or unsigned octal integer. The octal integer may be from one to six digits in length plus the sign. The assembler will store it in a word, right-justified. The word will be stored in its real form and will not be complemented if there is the presence of a minus one. The sign applies to bit 0 only.

Examples of octal literals are:

=01257  
=0-37742

### Alphanumeric Literals

The alphanumeric, or Hollerith literal, consists of the letters H or nH, where n is a character count followed by the data. If there is no count specified, a literal of exactly one 6-bit character is assumed to follow the letter H. If a count exists, the n characters following the character H are to be used as the literal. If the value n is not a multiple of three, the last partial word will be left-justified and filled in with blanks. The value n can range from 1 through 53. (Embedded blanks do not terminate scanning of the cards by the assembler.)

Examples of alphanumeric literals are:

=HA  
=HG  
=4HCONE (  represents a blank)  
=7HTHE

## SACI Literals

The Symbolic ASCII (SACI) literals consist of the letter A or nA where n is a field count, followed by the data. The maximum value of n is 26 and the SACI literal must be contained on one card (cannot be continued with the ETC pseudo-operation). In nonmemory reference literals, no count is specified; a literal of exactly one Symbolic ASCII field (see the SACI pseudo-operation for the permitted fields) is assumed to follow the letter A. If a count exists, the n fields following the character A are to be used as the literal. If the value of n is not a multiple of 2, the last partial word is left-justified and zero filled. The fields are comma separated and a blank will terminate the scanning of the card by the assembler, (blanks may be assembled by using the symbol SP). If the PARITY pseudo-operation is ON, parity is generated for each character.

Examples of Symbolic ASCII literals:

<u>Literal</u>	<u>Interpreted Value</u>
=A\$	044
=ASP	040
=AEOT	004
=4ACR,LNF,?,DEL	015012, 077177
=7AT,LH,LE,SP,E,LN,LD	124150, 145040, 105156, 144000

See Figure 4-1 for the symbolic ASCII symbols.

## Instruction Literals

The instruction literal consists of the equal sign (=) character followed by the letter M. This is followed in turn by an operation code, one blank, and a variable field. (The embedded blank does not terminate scanning of the card in this instance.) Only the machine instructions and one pseudo-operation (ARG) are legal in an instruction literal.

Examples of instruction literals are:

=MARG/BETA-ALPHA  
=MTRA/ALPHA-BETA

Pseudo-operations containing instruction literals cannot use address modification, since, if a modifier is encountered, it is assumed to be part of the instruction literal.

## Variable Field Literals

The variable field literal begins with the character V. Subfields are separated by commas. Each subfield is preceded by a count of bits for that subfield and a slash (/). The total bit count for one literal subfield must not exceed 18. The subfields of a variable field literal may be one of three types: algebraic, Boolean, alphanumeric. See the "VFD (Variable Field Definition)" pseudo-operation for the detailed description of use of variable field data. The variable field format is the same for both the variable field literal and the VFD pseudo-operation.

Examples of variable field literals are:

```
=V10/895,5/37,H6/C,15/ALPHA
=V18/ALPHA,012/235,6/0
```

Pseudo-operations containing variable field literals cannot use any of the forms of a tag modifier.

## Nonmemory Reference Instruction Literals

When a literal is used in a nonmemory reference instruction, the value of the literal is not stored in the literal pool but is truncated to a 9-bit value and placed in the displacement field of the instruction. Normally a literal represents an 18-bit number. For the nonmemory, floating-point, Hollerith, and SACI literal bits 0 through 8 of the literals are placed in the displacement field. For all other literals, bits 9 through 17 are placed in the displacement field.

Examples of nonmemory reference literals:

<u>Coded Literal</u>	<u>Assembled Instruction</u>
ILA =100	673144
IANA =077	022077
IERA =0777531	322531
ILA =2B13	673040
ICMPA =H?	422017
ICMPA =AETX	422003

NOTE: If parity was specified as ODD and is ON, the last example would assemble as 422203.

### SECTION III

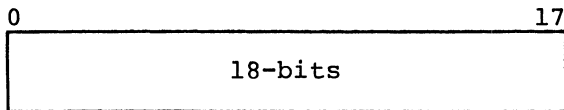
#### PROCESSOR INSTRUCTIONS

##### REPRESENTATION OF INFORMATION

The DATANET FNP deals with four basic data sizes: 6 bits, 9 bits, 18 bits, and 36 bits. The 6-bit and 9-bit sizes are called alphanumeric data (characters); the 18-bit and 36-bit sizes are called single-precision data and double-precision data, respectively.

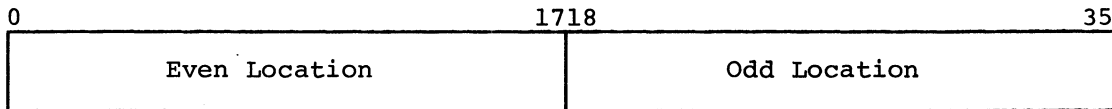
##### Single-Precision Data

The single-precision data word occupies one memory location and consists of 18 bits arranged as follows:



##### Double-Precision Data

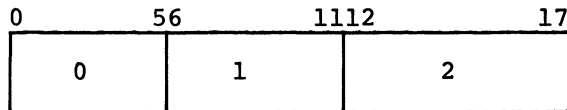
The double-precision data word consists of 36 bits arranged as shown below. This word occupies two consecutive memory locations, an even location and the next higher odd location (a word pair).



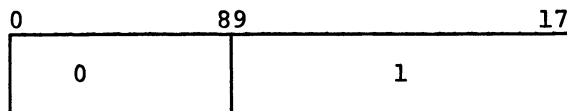
## Alphanumeric Data

Alphanumeric data consists of 6-bit or 9-bit characters. A single-precision word (one memory location) contains either two 9-bit characters or three 6-bit characters as shown below. The three 6-bit characters are numbered 0, 1 and 2; the two 9-bit characters are numbered 0 and 1.

### 6-Bit Characters:



### 9-Bit Characters:



## NUMBER SYSTEM

Instructions can be divided into two groups according to the way in which the operand is interpreted: the "logical instruction" and the "algebraic instruction" group.

For the logical group, operands and results are regarded as unsigned, positive binary numbers. In the case of addition and subtraction, the occurrence of any overflow is reflected by the carry out of the most significant (leftmost) bit position:

**Addition:** If the carry out of the leftmost bit position equals 1, then the result is above the range.

**Subtraction:** If the carry out of the leftmost bit position equals 0, then the result is below the range.

In the case of comparisons, the Zero and Carry Indicators show the relation.

For the algebraic group, operands and results are regarded as signed, binary numbers, the leftmost bit being used as a sign bit, (a 0 being plus and 1 minus). When the sign is positive all the bits represent the absolute value of the number; and when the sign is negative, they represent the 2's complement of the absolute value of the number.

In the case of addition and subtraction the occurrence of an overflow is reflected by the carry into and out of the leftmost bit position (the sign position). If the carry into the leftmost bit position does not equal the carry out of that position then overflow has occurred. If overflow has been detected and if the sign bit equals 0, the resultant is below range; if with overflow, the sign bit equals 1, the resultant is above range.

An explicit statement about the location of the binary point is necessary only for multiplication and division; for addition, subtraction, and comparison it is sufficient to assume that the binary points are aligned. In the DATANET FNP, multiplication and division are implemented for 2's complement fractional numbers.

In integer arithmetic, the location of the binary point is assumed to be at the right of the least-significant bit position; that is, depending on the precision, to the right of bit position 17 or 35. The general representation of a fixed-point integer is then:

$$-a_n 2^n + a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \dots + a_1 2^1 + a_0 2^0$$

where  $a_n$  is the sign bit.

In fractional arithmetic, the location of the binary point is assumed to be at the left of the most-significant bit position, that is, to the left of bit position 1. The general representation of a fixed-point fraction is:

$$-a_0 2^1 + a_1 2^2 + \dots + a_{n-1} 2^{-(n-1)} + a_n 2^{-n}$$

The number ranges for the various cases of precision, interpretation, and arithmetic are listed in Figure 3-1.

		Precision		
Interpretation	Arithmetic	Address Field (X1,X2,X3)	Single Word (A,Q,Y)	Double Word (AQ, Y-Pair)
Algebraic	Integer	-	$-2^{17} \leq N \leq (2^{17}-1)$	$-2^{35} \leq N \leq (2^{35}-1)$
	Fractional	-	$-1 \leq N \leq (1-2^{-17})$	$-1 \leq N \leq (1-2^{-35})$
Logical	Integer	$0 \leq N \leq (2^{15}-1)$	$0 \leq N \leq (2^{18}-1)$	$0 \leq N \leq (2^{36}-1)$
	Fractional	-	$0 \leq N \leq (1-2^{-18})$	$0 \leq N \leq (1-2^{-36})$

Figure 3-1. Ranges of Fixed-Point Numbers

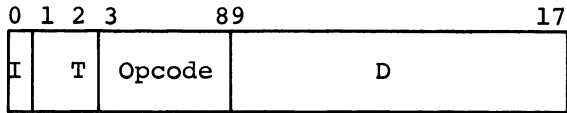


## INSTRUCTIONS

There are two basic types of DATANET FNP instructions: (1) memory reference instructions, and (2) nonmemory reference instructions.

### Memory Reference Instruction

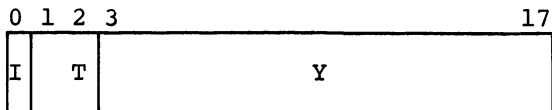
The memory reference instruction has the following format in physical memory.



where:

- I = Indirect Bit: when on (=1), the effective address is computed from the indirect word.
- T = Tag Field: used to specify address modification using one of three index registers (X1, X2, X3) or the instruction counter (IC).
- Opcode = Operation Code: One of the legal FNP memory reference operation codes.
- D = Displacement

The basic method of forming effective addresses consists of adding the 9-bit displacement field (D) to the complete address obtained from one of the three index registers or the instruction counter. The displacement (D) is treated as a 9-bit number in the 2's complement form. This allows the effective address to be greater-than, or less-than, the whole address in the base register (X1, X2, X3, or IC). When indirect addressing is specified (I=1), the effective address is used as the address of an indirect word with the following format:



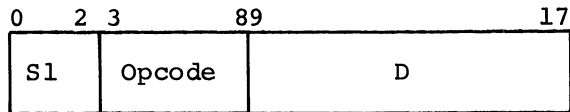
This differs from the instruction word, in that Y is an address field rather than a displacement field, and no base address is needed to form a full 15-bit address. The I specifies further indirect addressing.

### Nonmemory Reference Instructions

The nonmemory reference instructions are those instructions that contain the data to be used with the program addressable registers in the displacement field. There are two groups of nonmemory reference instructions, with each group having primary operation code(s) common to the group. The specific instruction within the group is determined by using other fields of the instruction word as suboperation codes.

Group 1 Nonmemory Instructions

The Group 1 nonmemory reference instructions have the following format:

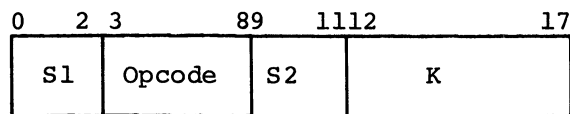


where:

- S1 = Suboperation Code: used to determine the specific instruction in the group.
- Opcode = Operation Code: Codes octal 73,22,52, or 12 signify that this is nonmemory reference Group 1 and that the displacement and tag fields are to be handled in a different manner from the Group 2 instructions.
- D = Displacement: This field is the operand and is handled differently for each instruction. See "Nonmemory Reference Instructions" for descriptions of the instructions in this group.

Group 2 Nonmemory Instructions

The nonmemory reference instructions in Group 2 have the following format:



where:

- S1, S2 = Suboperation Codes: These two codes form a prefix and a suffix to the operation codes and are used to determine the instruction within the group.
- Opcode = Operation Code: Code octal 33 signifies that the instruction is a Group 2 nonmemory reference instruction.
- K = Operation Value: This field is used for such functions as shift counts.

## PROCESSOR REGISTERS

The program-accessible registers are as follows:

<u>Name</u>	<u>Mnemonic</u>	<u>Length (bits)</u>
Accumulator	AQ	36
Three index registers	X1, X2, X3	18
Instruction counter	IC	15
Indicator register	I	8
Input/output channel select register	S	6

The AQ register is used as follows:

- In fixed-point operations, as an operand register for double-precision operations.
- In fixed-point operations, as operands for single-precision operations where each AQ half serves independently of the other. The values are then called A-register (AQ bits 0-17) and Q-register (AQ bits 18-35).

The index registers are used for address modification.

The indicator register is a generic term for all program-accessible indicators within the processor. The name is used where the set of indicators appears as a register, that is, a source or destination of data.

The instruction counter holds the address of the next instruction to be executed.

The input/output channel select register specifies the input/output channel for the programmed input/output operations.

## FAULTS

Faults (internal interrupts) are included in the processor to provide for program intervention when certain system errors or other events occur. Eight types of faults are provided. When specific faults occur, program control is automatically transferred to fixed memory locations. The eight types of faults are shown in Table 3-1.

As shown in the table, two groups of faults are defined: those which unconditionally abort the instruction in execution and those which do not. The faults which do not cause an unconditional abort (non-abort faults) are recognized under the same conditions as program interrupts except that they have higher priority.

Table 3-1. Processor Faults

Fault Vector Octal Memory Location	Fault Name	Priority	Unconditional Abort
00440	Power shutdown beginning	7	no
00441	Power-on restart	4	yes
00442	Memory parity error	1	yes on instruction and indirect cycles; no on operand cycles
00443	Illegal operation code	2	yes
00444	Overflow	5	no
00445	Illegal store operation	3	yes
00446	Divide check	6	no
00447	Illegal program interrupt	8	no

The recognition of a fault causes the processor to execute a hardware-forced TSY instruction using the contents of the memory location assigned to the specific fault, as an indirect word. The memory location can be program loaded with the starting address of a fault processing routine.

The instruction counter will always be the value present when the fault occurred, namely the address of the faulty instruction. The forced TSY fault response will increment that value by 1 for storage in memory.

Two faults have associated indicator and inhibit bits in the Indicator Register: the Memory Parity Error and the Overflow faults. The other faults have no indicators. The faults with indicators operate as follows: If the fault occurs and is not inhibited, the jump to the fault vector occurs, and the indicator bit is set. If the fault occurs and is inhibited, the jump to the fault vector does not occur, and the indicator bit is set. In either case, the program receives indication of the occurrence of the fault.

The processor faults shown in Table 3-1 are described as follows:

Power Shutdown Beginning Fault -- This fault is triggered by an external signal to the DATANET FNP indicating impending power off. This fault can be inhibited by a manual switch on the Operation and Maintenance Panel.

Power-On Restart Fault -- This fault is triggered by an external signal to the DATANET FNP indicating the power-on has occurred. The computer initializes and transfers to the fault vector location. This fault can be disabled by a manual switch on the Operation and Maintenance Panel.

Memory Parity Error Fault -- This fault is triggered when a parity error occurs during a read from memory by the processor. This fault can be program inhibited by setting the Parity Fault Inhibit bit ON with the LDI instruction.

Illegal Operation Code Fault -- When the processor detects an illegal operation code, this fault will occur. All unused operation codes are illegal.

Overflow Fault -- This fault is generated by an overflow during an arithmetic operation. This fault can be program inhibited by setting the Overflow Fault Inhibit bit ON with the LDI instruction.

Illegal Memory Operation -- The illegal memory operation fault will be generated for the following reasons:

- Memory controller time out (hardware error)
- Illegal command to memory controller (hardware error)
- Out of bounds address
- Any attempt to alter memory in a protected region
- A character address of seven (7)

Divide Check Fault -- This fault is generated when a division cannot be carried out for one of the reasons specified with each divide instruction.

Illegal Program Interrupt Fault -- The illegal program interrupt fault may be generated in two ways:

1. The processor attempted to answer an interrupt when there was no interrupt present (hardware error).
2. The processor answered a valid interrupt, but the contents of the word in memory containing the interrupt sublevels were all zeros. This can happen if the processor or IOM stores zero in one of the interrupt sublevel locations after an interrupt has been set.

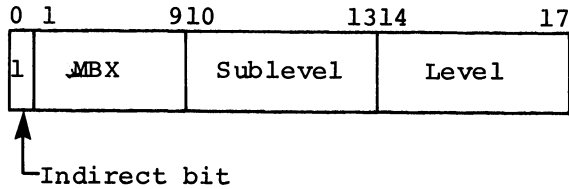
NOTE: A convenient way to determine if an interrupt was present in a word is to look at bits 16 and 17 of the interrupt sublevel locations. If they are both 1, then an interrupt in that level has been answered.

Any faults which happen during a CIOC, LDEX, or STEx instruction are answered by the processor, even though the IOM participated in the cycle. The control strobes from the memory controller will not go to the IOM under those conditions.



Bits 16 and 17 of the Interrupt Sublevel Word are both set to one each time that word is accessed by the Memory Controller in response to an interrupt. These bits may only be set to zeros by program. This feature is available as a diagnostic aid. If an interrupt is answered and the Interrupt Sublevel Word is found to contain all zeros in bits zero to 15, an Illegal Program Interrupt fault is generated.

After the scan of the Interrupt Sublevel Word has detected a one and the bit position (sublevel number) of the one is determined, the processor is forced by hardware to execute a TSY instruction with two levels of indirect address modification. The first indirect vector address is formed with the format:



This indirect word points to the Interrupt Vector location. No memory cycle is involved in obtaining either the TSY instruction or the first level indirect word.

The second level indirect address is obtained from the memory location specified by the Interrupt Vector Address just described. This location is normally program loaded with the starting address of an interrupt service routine.

Since the TSY is wired into the interrupt mechanism; the only variable for the operating program is the vector to the service routine. This means that a program cannot NOP (no operation) an interrupt (as might be done for a pseudo mask). This must be done by a "spring loaded" transfer, that is, a vector out to an immediate return.

The interrupt levels may be inhibited (masked), but not individual sublevels within an Interrupt Sublevel Word.

Table 3-2 shows the memory map of the Interrupt Vector and Interrupt Sublevel word locations.

Table 3-2. Memory Map - Interrupts

Octal Address	Sublevel No. (Decimal)	Level No. (Decimal)	
00000	0	0	} Interrupt Vectors
00001	0	1	
00002	0	2	
00003	0	3	
00004	0	4	
00005	0	5	
00006	0	6	
00007	0	7	
00010	0	8	
00011	0	9	
00012	0	10	
00013	0	11	
00014	0	12	
00015	0	13	
00016	0	14	
00017	0	15	
00020	1	0	} Interrupt Sublevel Words
00021	1	1	
.	.	.	
.	.	.	
.	.	.	
00377	15	15	
00400	Not applicable	0	
.	Not applicable	.	
.	Not applicable	.	
00417	Not applicable	15	

The program can enable or inhibit any of the 16 interrupt levels via the instruction SIER (Set Interrupt level Enable Register). Inhibiting an interrupt level postpones its recognition until the level is enabled.

The recognition of all interrupts may be inhibited via the instruction INH (Inhibit Interrupts). Recognition of any interrupt is postponed until the execution of an ENI (Enable Interrupts) instruction.

Recognition of any enabled interrupt or non-abort fault occurs at the completion of the instruction execution in progress when the interrupt or fault was set, with the following exceptions:

- All jump (transfer) instructions.
- ENI, INH, LDI, and STI instructions.



## PROCESSOR INDICATORS

The processor indicators give the programmer information about the present state of the processor and the program it is executing. The indicators are set automatically by the processor and, in general, indicate the results after the execution of the present instruction. The Indicators can be regarded as individual bit positions in an 8-bit Indicator Register (IR). An indicator is set to the ON or OFF state by certain events in the processor, or by certain instructions. The ON state corresponds to a binary one in the respective bit position of the IR; the OFF state corresponds to a zero.

The description of each machine instruction includes those indicators that may be affected by the instruction and the conditions under which a setting of the indicators to a specific state occurs. If the conditions stated are not satisfied, the state of the indicator remains unchanged.

The instruction set includes the LDI and STI instructions which transfer data between a memory location and the Indicator Register. The indicators and their relation to the bit positions of the memory location word are shown in Table 3-3. For the purposes of these instructions, the Indicator Register and the I/O Channel Select Register are treated as one register. In other operations, these registers are functionally separate.

Table 3-3. Indicators and Their Relation to the Bit Positions of a Memory Location

Bit Position	Indicator
0	Zero
1	Negative
2	Carry
3	Overflow
4	Interrupt inhibit
5	Parity fault inhibit
6	Overflow fault inhibit
7	Parity error
8	Not used
9	
10	
11	Input/output channel Select Register
12	
13	
14	
15	
16	
17	

The following describes the individual indicators:

**Zero Indicator** -- The zero indicator is used to test for zero or non-zero operands or resultants. It is set by instructions that change the contents of a processor register (AQ, Xn) or adder, and by the comparison instructions. The indicator is set ON when the new contents of the affected register or adder output contains all binary zeros otherwise the indicator is set OFF.

The zero indicator is tested by the Transfer on Zero (TZE) and the Transfer on Not Zero (TNZ) instructions.

Negative Indicator -- The negative indicator is used to test for negative or positive operands or resultants. It is affected by instructions that change the contents of a processor register (AQ) or adder, and by comparison instructions. The indicator is set ON when the contents of bit position 0 of this register or adder output is a binary 1; otherwise it is set OFF. The contents of the index registers do not affect this indicator.

The negative indicator is tested by the Transfer on Minus (TMI) and Transfer on Plus (TPI) instructions.

Carry Indicator -- The carry indicator is used to determine if an operation has generated a carry out of the two most significant bits (bit positions 0 and 1). This is not an arithmetic overflow. The carry indicator is affected by left shifts, additions, subtractions, and comparisons. The indicator is set ON when a carry is generated out of bit position 0; otherwise it is set OFF. On arithmetic shifts to the left, a carry is produced whenever the number involved is changed in sign during the shift.

The Transfer on No Carry (TNC) instruction tests the state of the carry indicator.

Overflow Indicator -- The overflow indicator is used to determine if the resultant of an arithmetic operation has exceeded the word length of the computer.

The overflow indicator is set ON if there is a carry out of either the most significant bit (bit position 0) or the next most significant bit (bit position 1) but not both. It is affected by the arithmetic instructions, but not by compare or Boolean instructions.

Since it is not set to OFF otherwise, the Overflow Indicator reports any overflow that has occurred since it was last set OFF by one of the instructions Transfer on Overflow (TOV) or Load Indicator Register (LDI).

The TOV instruction tests the status of the overflow indicator and sets it OFF.

Overflow Fault Inhibit Indicator -- If the overflow fault inhibit indicator is ON, then the setting ON of the overflow indicator does not cause an overflow fault trap to occur. The overflow fault inhibit indicator can be set ON or OFF only by the instruction LDI. Clearing of the overflow fault inhibit indicator to the enabled state does not generate a fault from a previously set overflow indicator. The status of the overflow fault inhibit indicator does not affect the setting, testing, or storing of the overflow indicator.

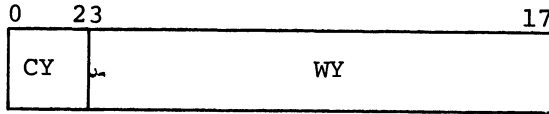
Parity Error Indicator -- The parity error indicator is set to ON when a parity error is detected during the access of words from memory. It may be set to ON or OFF by the LDI instruction.

Parity Fault Inhibit Indicator -- When the parity fault inhibit indicator is ON, the setting of the parity error indicator does not cause a parity error fault trap to occur. When the parity fault inhibit indicator is OFF, such a trap will occur. The parity fault inhibit indicator can be set to ON or OFF only by the LDI instruction. Clearing of the parity inhibit indicator to the unmasked state does not generate a fault from a previously set parity error indicator. The status of the parity fault inhibit indicator does not affect the setting, testing, or storing of the parity error indicator.

Interrupt Inhibit Indicator -- The interrupt inhibit indicator is affected by the Load Indicator Register (LDI), Interrupt Inhibit (INH) and Enable Interrupt (ENI) instructions. If interrupts are inhibited, this indicator is turned on.

ADDRESS FORMATION

The DATANET FNP memory is addressed by an 18-bit two-part address as follows:



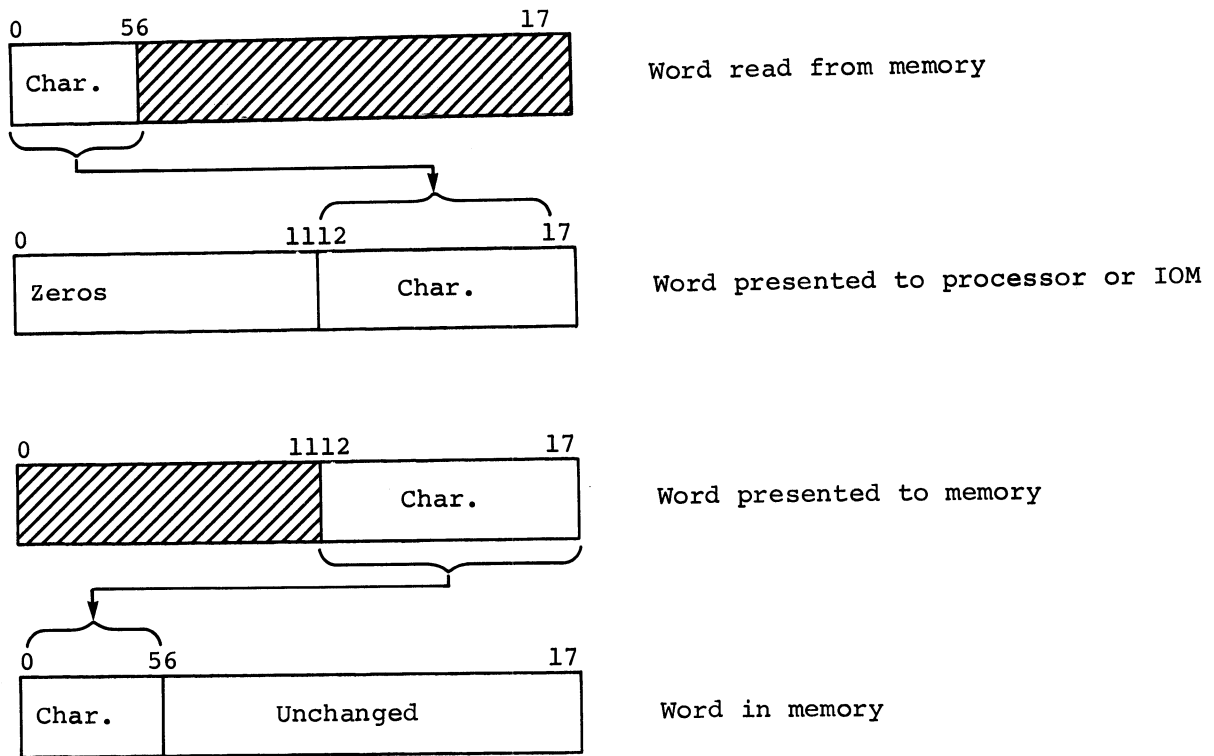
The CY field is the character address and the WY field is the word address. The character and word addresses are used for all memory references. The 15-bit WY specifies one of 32,768 (18-bit) words (maximum) in the normal binary fashion. The 3-bit CY specifies the desired character by a code assigned to each character.

The addressable characters, their CY codes, and fractional interpretations are:

<u>Fractional Interpretation</u>	<u>CY Value</u>	<u>Character Addressed</u>
0/3	100	6-bit character number 0 (bits 0-5)
1/3	101	6-bit character number 1 (bits 6-11)
2/3	110	6-bit character number 2 (bits 12-17)
0/2	010	9-bit character number 0 (bits 0-8)
1/2	011	9-bit character number 1 (bits 9-17)
Full word	000	entire 18-bit word

The character addressing feature is valid only for single-precision (18-bit) memory references.

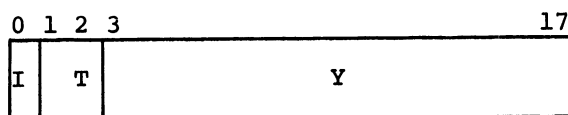
When characters are addressed, they are transferred to and from the memory right justified. Unused bits are zeros for operations from memory and are ignored for operations to memory. An example of a load and store operation upon 6-bit character No. 0 (CY = 100) is:



### Basic Level Effective Address Formation Rules

The basic method of forming an effective address (Y\*\*) consists of adding the 9-bit instruction displacement (D) field to a selected base register (X1, X2, X3, or IC). The displacement field is treated as a 9-bit two's complement number to allow the effective address to be greater than, or less than, the base address.

When the instruction specifies indirect addressing (I bit=1), the effective address is calculated as above to form the address for an indirect word with the format:



The Y field of the indirect word is an address field, and no displacement has to be added to form the effective address. The indirect bit in this word specifies whether or not continued indirect addressing is to be performed. The T field specifies which of the three index registers is to be used for further indirect modification. If the T field in the indirect word is 00, no modification is performed and the Y field becomes the effective address (Y\*\*). The basic address formation rules are:

	Instruction Word		Indirect Word	
	<u>I=0</u>	<u>I=1</u>	<u>I=0</u>	<u>I=1</u>
T=00	Y**=IC+D	Y*=C(IC+D)	Y**=Y	Y*=C(Y)
T=01	Y**=X1+D	Y*=C(X1+D)	Y**=X1+Y	Y*=C(X1+Y)
T=10	Y**=X2+D	Y*=C(X2+D)	Y**=X2+Y	Y*=C(X2+Y)
T=11	Y**=X3+D	Y*=C(X3+D)	Y**=X3+Y	Y*=C(X3+Y)
NOTE: Y** = Effective operand address Y* = Address of indirect word				

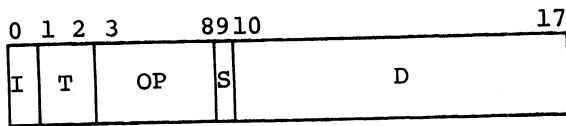
There is no limitation on the number of indirect levels at modification.

The memory reference instructions can be divided into two types: single-precision and double-precision. All single-precision instructions can address an 18-bit word, a 6-bit character, or a 9-bit character. Double-precision instructions can address only word pairs as operands.

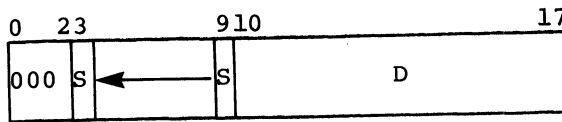
When forming the effective address for a single-precision instruction, the data size to be used is determined by the index register specified (if any) for address modification. Each of the three 18-bit index registers (X1, X2, and X3) contains two parts, a 15-bit word address part (WXn) and a 3-bit character address part (CXn), in the same way that the memory address has a character part and a word part. When the processor selects the index register for address modification, the CXn field of the index register is examined. If that 3-bit field contains any of the character address codes, the resultant memory reference will involve a character. If the CXn field of the selected index register contains the code for a full word address (CXn=000), the resultant memory reference will involve a full 18-bit word. If no index register is selected, word addressing is always used. The IC has only 15 bits; these 15 bits correspond to a word address. The nonexistent 3-bit C field of the IC is treated as if it contained the word code (000).

#### WORD ADDRESSING - BASIC LEVEL

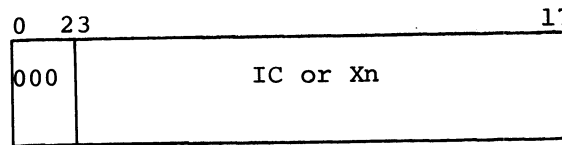
If the effective address is the address of a full word, it is formed by adding the D field of the instruction to the word part of the specified base register. The D field is expanded into a 15-bit quantity by extending the sign of D (bit 9) six places to the left. The addition of the expanded D field and the IC is limited to 15 bits so that carries out of bit 3 will not affect CY\*.



Instruction word in memory



Expanded D field



Instruction counter or Index Register

The effective address is formed by examining the C field of the selected base register (IC or Xn, n = 1,2,3). If that field is 000, the effective address is formed as shown above.

Since the IC is a 15-bit register, it can only be used for word addressing.

#### CHARACTER ADDRESSING - BASIC LEVEL

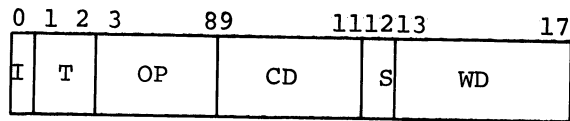
In character addressing, memory is considered as containing 18-bit words divided into two 9-bit characters, or three 6-bit characters. The address of 6-bit character number-0 of location Z would be  $Z + 0/3$ . Similarly the address of 9-bit character number-0 of location Z would be  $Z + 0/2$ . The character and word address fields of a memory address correspond to the fraction and word number.

When the FNP computes an effective address for a character, it treats the quantities involved as if they were integer and fractional addresses. For example, if  $C(Xn) = (Q+0/3)$ , the effective address computed for an LDA instruction with a displacement of  $(1+2/3)$  (referencing Xn), would be  $(Q+0/3) + (1+2/3) = (Q+1+2/3)$ .

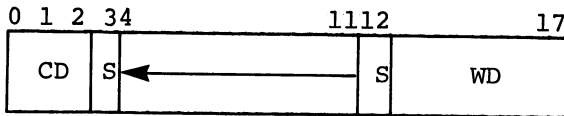
A Load A Register (LDA) instruction with a displacement field of  $(-1+2/3)$  referencing Xn would have an effective address of  $(Q+0/3) + (-1+2/3) = (Q-1+2/3)$ . This allows forward and backward addressing of a character string.

The effective address in the character addressing mode is formed by treating the D field of the instruction as having a 3-bit character (fractional) part and a 6-bit word part. The 6-bit word part is expanded into a 15-bit signed number by extending bit 12 of the instruction word nine places to the left.

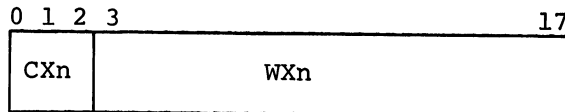
The resultant 15-bit number is added to the 15-bit word portion of the selected index register. The 3-bit character field (C) of the expanded displacement, and the 3-bit C field of the selected index register together determine the C field of the effective address. For example, if the C field of the displacement has a value 101 (i.e., 6-bit character number 1), which is interpreted as 1/3, and the C field of the index register has a value 101 (1/3), the C field of the effective address will be 110 (2/3), which is 6-bit character number 2. The expansion of the D field is:



Instruction in memory



Expanded D field



Selected index register (X1,X2,X3)

If the C fields of the D field and the selected index register combine to form a fraction greater than, or equal to 1, the word address portion of the effective address is increased by one. This can happen for example, when an LDA instruction with a D field interpreted as (1+2/3) references X1, for which the contents are interpreted as (Q+1/3). The effective address is then (Q+1/3) + (1+2/3) = (Q+2+0/3).

A list of examples showing the effective address formation for various combinations of C values is shown in Figure 3-2. This figure shows both the machine operation (in octal) and its fractional interpretation.

When the effective address is formed by the character addition process, carries from the word field addition do not affect the character addition. However, carries from the character field addition go "end around" to the word field addition, as in the case of (2/3+1/3) = 3/3 = (1+0/3).

A summary of the character addition rules, in the form of an addition table is shown in Figure 3-3. This figure shows the actual machine numbers (in octal) and their fractional interpretation.

CX, WX	D Field of Instruction Word	Expanded D Field C, (expansion)W	Effective Address CY*, WY*	Meaning
5,07642 (07642 + 1/3)	403 + (3 + 0/3)	4,(000)03 =	5,07645 (07645 + 1/3)	Six-bit character No. 1 of location 07645
5,07642 (07642 + 1/3)	603 + (3 + 2/3)	6,(000)03 =	4,07646 (07646 + 0/3)	Six-bit character No. 0 of location 07646
3,07642 (07642 + 1/2)	203 + (3 + 0/2)	2,(000)03 =	3,07645 (07645 + 1/2)	Nine-bit character No. 1 of location 07645
3,07642 (07642 + 1/2)	302 + (3 + 1/2)	3,(000)03 =	2,07646 (07646 + 0/2)	Nine-bit character No. 0 of location 07645
5,07642 (07642 + 1/3)	677 + (-1 + 2/3)	6,(777)77 =	4,07642 (07642 + 1/3)	Six-bit character No. 0 of location 07642
5,07642 (07642 + 1/3)	577 + (-1 + 1/3)	5,(777)77 =	6,07641 (07641 + 2/3)	Six-bit character No. 2 of location 07641
3,07642 (07642 + 1/2)	377 + (-1 + 1/2)	3,(777)77 =	2,07642 (07642 + 0/2)	Nine-bit character No. 0 of location 07642
3,07642 (07642 + 1/2)	277 + (-1 + 0/2)	2,(777)77 =	3,07641 (07641 + 1/2)	Nine-bit character No. 1 of location 07641

Figure 3-2. Effective Address Formation Examples, Character Addressing



C Field of Expanded D

Fractional	Octal	Word		0	1	0/2	2	1/2	3	0/3	4	1/3	5	2/3	6	Illegal	7
		Word	Double Word														
Word	0	-	-	0	7	-	7	-	7	-	7	-	7	-	7	-	7
Double Word	1	-	-	7	7+	-	7+	-	7+	-	7	-	7+	-	7+	-	7
0/2	2	-	-	7	7+	0/2	2	1/2	3	-	7	-	7+	-	7+	-	7
1/2	3	-	-	7	7+	1/2	3	0/2+1	2+	-	7	-	7+	-	7+	-	7
0/3	4	-	-	7	7	-	7	-	7	0/3	4	1/3	5	2/3	6	-	7
1/3	5	-	-	7	7	-	7+	-	7+	1/3	5	2/3	6	0/3+1	4+	-	7
2/3	6	-	-	7	7+	-	7+	-	7+	2/3	6	0/3+1	4+	1/3+1	5+	-	7
Illegal	7	-	-	7	7+	-	7+	-	7+	-	7	-	7+	-	7+	-	7

Notes

(1) Illegal combinations result in an octal 7 being generated as the effective character address. Octal 7 causes an Illegal Store Operation Fault.

(2) + indicates an end-around carry.

Figure 3-3. Character Address Addition Rules

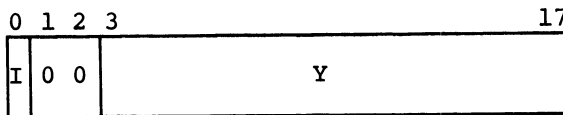
# Indirect Level Effective Address Formation Rules

## WORD ADDRESSING - INDIRECT LEVEL

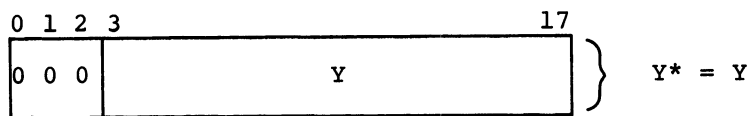
In indirect level addressing, the source of the address is the indirect word instead of the instruction, as in basic level addressing. A difference from basic level addressing is that the indirect word contains an address field, not a displacement field. The address field of the indirect word may be used with or without index register modification. Figure 3-4 shows the effective address formation for word addressing without and with index register modification respectively.

### Without Index Register Modification

Indirect word in memory:

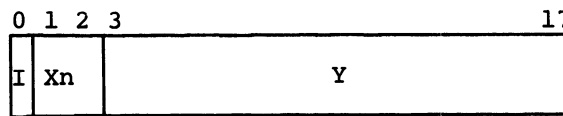


Expanded Y Field:

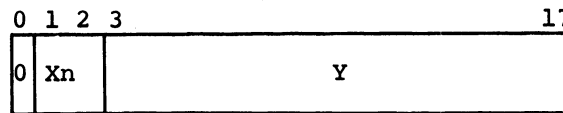


### With Index Register Modification

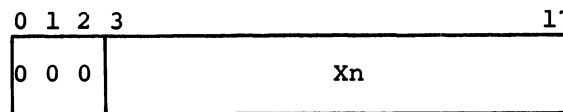
Indirect word in memory:



Expanded Y field:



Index Register:



$$Y^* = Y + X_n$$

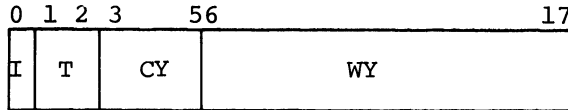
Figure 3-4. Indirect Level Word Addressing

CHARACTER ADDRESSING - INDIRECT LEVEL

The character addressing mechanism may also be used during indirect addressing. If the T field of the indirect word references an index register the C field of which is non-zero, the 15-bit address field of the indirect word is broken into a 3-bit C field and a 12-bit signed word field, as shown in Figure 3-5. The effective address is then calculated using the same fractional address rules as in basic level addressing.

Y-Field Expansion

Indirect word in memory



Expanded indirect word:

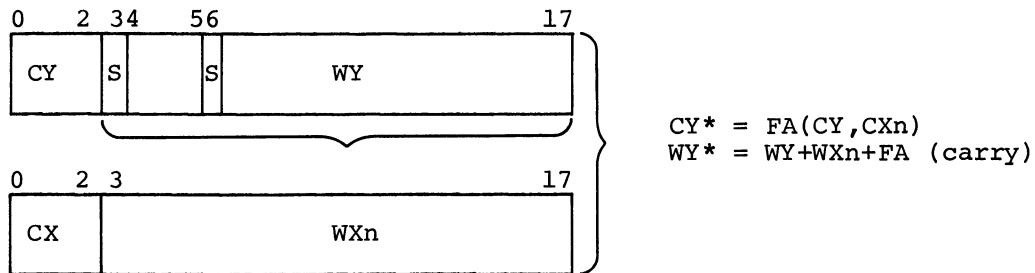


Figure 3-5. Indirect Level Character Addressing

PROCESSOR INSTRUCTIONS

Processor instructions (machine instructions) written for the assembler consist of a symbol (or blanks) in the location field, a 3- to 6-character alphanumeric code representing a DATANET FNP operation in the operation field, and an operand address (symbolic or numeric) plus a possible modifier tag in the variable field. Legal symbols used in the location fields, and as operand addresses in the variable fields, have been previously described.

The standard machine mnemonics are entered left justified in the operation field. Mnemonics by functional class are listed in Table 3-4 showing Execution Time and octal code. Table 3-5 contains an alphabetical listing of mnemonics. Table 3-6 contains a listing of mnemonics by octal codes.

Table 3-4. Instruction Repertoire by Functional Class

Memory Reference Instructions

<u>Octal Code</u>	<u>Instruction</u>	<u>Execution Time (Microseconds)</u>
<u>Data Movement - Load</u>		
07	LDA Load A	2.0
47	LDQ Load Q	2.0
04	LDAQ Load AQ	2.4
43,03,41	LDXn Load Xn (n = 1,2,3)	2.0
44	LDI Load Indicator register	2.0
<u>Data Movement - Store</u>		
17	STA Store A	2.1
57	STQ Store Q	2.1
14	STAQ Store AQ	2.1
53,13,50	STXn Store Xn (n = 1,2,3)	2.1
54	STI Store Indicator register	2.1
56	STZ Store Zero	2.1
<u>Arithmetic - Addition</u>		
06	ADA Add to A	2.0
46	ADQ Add to Q	2.0
15	ADAQ Add to AQ	2.4
42,02,40	ADCXn Add character address to Xn	2.0
16	ASA Add stored to A	2.5
76	AOS Add one to storage	2.5
<u>Arithmetic - Subtraction</u>		
26	SBA Subtract from A	2.0
66	SBQ Subtract from Q	2.0
24	SBAQ Subtract from AQ	2.4
36	SSA Subtract stored from A	2.5
<u>Arithmetic - Multiplication</u>		
01	MPF Multiply fraction	6.7
<u>Arithmetic - Division</u>		
21	DVF Divide fraction	7.5
<u>Boolean Operations</u>		
34	ANA AND to A	2.0
32	ANSA AND to storage A	2.5
37	ORA OR to A	2.0
72	ORSA OR to storage A	2.5
35	ERA EXCLUSIVE OR to A	2.0
62	ERSA EXCLUSIVE OR to storage A	2.5
<u>Comparison</u>		
27	CMPA Compare with A	2.0
67	CMPQ Compare with Q	2.0
63,23,61	CMPXn Compare with Xn	2.0
20	SZN Set zero and negative indicators from memory	2.0
31	CANA Comparative AND with A	2.0


Table 3-4 (cont). Instruction Repertoire by Functional Class

<u>Octal Code</u>	<u>Instruction</u>	<u>Execution Time (Microseconds)</u>
<u>Transfer of Control</u>		
71	TRA	Transfer unconditionally 1.0
10	TSY	Transfer and store (IC) in Y 2.2
74	TZE	Transfer on zero 1.0
64	TNZ	Transfer on not zero 1.0
75	TMI	Transfer on minus 1.0
65	TPL	Transfer on plus 1.0
45	TNC	Transfer on no carry 1.0
55	TOV	Transfer on overflow 1.0
<u>Input/Output</u>		
60	CIOC	Connect input/output channel 3.3
30	LDEX	Load external (I/O) channel 3.0
70	STEX	Store external (I/O) channel 3.1

Nonmemory Reference Instructions

<u>Octal Code</u>	<u>Instruction</u>	<u>Execution Time (Microseconds)</u>
GROUP 1		
<u>Immediate Load</u>		
673	ILA	Immediate load A 1.3
473	ILQ	Immediate load Q 1.3
<u>Immediate Add</u>		
773	IAA	Immediate add to A 1.3
573	IAQ	Immediate add to Q 1.3
173,273, 373	IACXn	Immediate add character address to Xn 1.3
<u>Immediate Boolean</u>		
022	IANA	Immediate AND to A 1.3
122	IORA	Immediate OR to A 1.3
322	IERA	Immediate EXCLUSIVE OR to A 1.3
<u>Immediate Compare</u>		
422	ICMPA	Immediate compare with A 1.3
222	ICANA	Immediate comparative AND to A 1.3
<u>Interrupt Control</u>		
012	RIER	Read Interrupt Level Enable Register 2.2
412	RIA	Read Interrupt Address 2.5
052	SIER	Set Interrupt Level Enable Register 2.3
452	SIC	Set Interrupt Cells 2.5

Table 3-4 (cont). Instruction Repertoire by Functional Class

<u>Octal Code</u>	<u>Instruction</u>	<u>Execution Time (Microseconds)</u>	
GROUP 2			
<u>Data Movement</u>			
7333	CQA Copy Q register into A register	1.3	
6333	CAQ Copy A register into Q register	1.3	
4332, 0332,4333	CAXn Copy A register into Xn register	1.3	
2332, 3332,3333	CXnA Copy Xn register into A register	1.3	
<u>Interrupt Control</u>			
3331	INH Interrupt inhibit mode ON	1.3	
7331	ENI Interrupt enable mode ON	1.3	
4331	DIS Delay until interrupt	1.3	
2331	NOP No operation	1.3	
<u>Shifts</u>			
0337	ARS A right shift	$1.2 + 0.25N^1$ 	
4337	QRS Q right shift		
0335	LRS Long right shift		
0336	ALS A left shift		
4336	QLS Q left shift		
0334	LLS Long left shift		
2337	ARL A right logic		
6337	QRL Q right logic		
2335	LRL Long right logic		
2336	ALR A left rotate		
3336	ALP A left parity rotate		
6336	QLR Q left rotate		
7336	QLP Q left parity rotate		
2334	LLR Long left rotate		
1336	NRM Normalize		$1.4 + 0.35N^1$
1334	NRML Normalize long		$1.4 + 0.25N^1$

<sup>1</sup>N = Number of bits shifted

Table 3-5. Instruction Repertoire (Alphabetical)

Mnemonics		
ADA	IACXn	QLS
ADAQ	IANA	QRL
ADCXN	IAQ	QRS
ADQ	ICANA	RIA
ALP	ICMPA	RIER
ALR	IERA	SBA
ALS	ILA	SBAQ
ANA	ILQ	SBQ
ANSA	INH	SEL
AOS	IORA	SIC
ARS	LDAQ	SSA
ASA	LDEX	STA
CANA	LDI	STAQ
CAQ	LDQ	STEX
CAXn	LDXn	STI
CIOC	LLR	STQ
CMPA	LLS	STXn
CMPQ	LRL	STZ
CMPXn	LRS	SZN
CQA	MPF	TMI
CXnA	NOP	TNC
DIS	NRM	TNZ
DVF	NRML	TOV
ENI	ORA	TPL
ERA	ORSA	TRA
ERSA	QLP	TSY
IAA	QLR	TZE

Table 3-6. Instruction Repertoire (Octal Codes)

Memory Reference				Nonmemory Reference					
				<u>Group 1</u>		<u>Group 2</u>			
<u>Octal</u>		<u>Octal</u>		<u>Octal</u>		<u>Octal</u>		<u>Octal</u>	
00	----	40	ADCX3	073	SEL	0330	----	4330	----
01	MPF	41	LDX3	173	IACX1	0331	----	4331	DIS
02	ADCX2	42	ADCX1	273	IACX2	0332	CAX2	4332	CAX1
03	LDX2	43	LDX1	373	IACX3	0333	----	4333	CAX2
04	LDAQ	44	LDI	473	ILQ	0334	LLS	4334	----
05	----	45	TNC	573	IQQ	0335	LRS	4335	----
06	ADA	46	ADQ	673	ILA	0336	ALS	4336	QLS
07	LDA	47	LDQ	773	IAA	0337	ARS	4337	QRS
10	TSY	50	STX3	012	RIER	1330	----	5330	----
11	----	51	----	112	----	1330	----	5331	----
12	Group 1	52	Group 1	212	----	1332	----	5332	----
13	STX2	53	STX1	312	----	1333	----	5333	----
14	STAQ	54	STI	412	RIA	1334	NRML	5334	----
15	ADAQ	55	TOV	512	----	1335	----	5335	----
16	ASA	56	STZ	612	----	1336	NRM	5336	----
17	STA	57	STQ	712	----	1337	----	5337	----
20	SZN	60	CIOC	052	SIER	2330	----	6330	----
21	DVF	61	CMPX3	152	----	2331	NOP	6331	----
22	Group 1	62	ERSA	252	----	2332	CX1A	6332	----
23	CMPX2	63	CMPX1	352	----	2333	----	6333	CAQ
24	SBAQ	64	TNZ	452	SIC	2334	LLR	6334	----
25	----	65	TPL	552	----	2335	LRL	6335	----
26	SBA	66	SBQ	652	----	2336	ALR	6336	QLR
27	CMPA	67	CMPQ	752	----	2337	ARL	6337	QRL
30	LDEX	70	STEX	022	IANA	3330	----	7330	----
31	CANA	71	TRA	122	IORA	3331	INH	7331	ENI
32	ANSA	72	ORSA	222	ICANA	3332	CX2A	7332	----
33	Group 2	73	Group 1	322	IERA	3333	CX3A	7333	CQA
34	ANA	74	TZE	422	ICMPA	3334	----	7334	----
35	ERA	75	TMI	522	----	3335	----	7335	----
36	SSA	76	AOS	622	----	3336	ALP	7336	QLP
37	ORA	77	----	722	----	3337	----	7337	----



## Processor Instruction Description

The descriptions of the DATANET FNP instruction set are arranged by functional class including data movement, arithmetic, Boolean operations, comparison, transfer of control, and miscellaneous operations. For the description of the machine instructions it is assumed that the reader is familiar with the general structure of the processor, the representation of information, the data formats, and the method of address modifications, as presented in the preceding paragraphs.

A fixed format is presented for the description of each machine instruction. The format includes the mnemonic and name of the instruction, a summary, indicators, and pertinent notes.

A heading identifies the mnemonic and name of the instruction. The octal code is shown in the word format and includes any suboperation codes.

The summary presents the change in the status of the processor that is effected by the execution of the instruction. The operation is described in a shorthand, symbolic form. If reference is made to the status of an indicator, then it is the status of this indicator before the operation is executed.

Only those indicators for which the status can be changed by the execution of this instruction are listed. Indicators that are not listed are not affected. In most instances, a condition for setting ON as well as one for setting OFF is stated. If only one of the two is stated, then this indicator remains unchanged. Unless explicitly stated otherwise, the conditions refer to (for example) the contents of registers as existing after the execution of the instruction.

Notes exist only in those instances where the summary is not sufficient for an understanding of the operation.

The following abbreviations and symbols are used for the description of the machine operation.

### EFFECTIVE ADDRESS AND MEMORY LOCATION SYMBOLS

- Y\* = The effective address of the respective instruction
- Y pair = A symbol denoting that the effective address Y designates a pair of memory locations (36 bits) with successive addresses, the lower one being even. When effective address is even, then it designates the pair (Y, Y + 1), and when it is odd, then the pair (Y - 1, Y). In any case the memory location with the lower (even) address contains the most significant part of a double-precision number, or the first of a pair of instructions.
- D = Displacement contained in the D field of the instruction.

## REGISTER SYMBOLS

A	=	Accumulator register (18 bits)
Q	=	Quotient register (18 bits)
AQ	=	Combined accumulator-quotient register (36 bits)
X	=	Index register n (n = 1,2,3) (18 bits)
CX	=	Positions 0,1,2 of Xn
WX	=	Positions 3 . . . 17 of X
IC	=	Instruction counter (15 bits)
I	=	Indicator register (8 bits)
S	=	Input/output channel selection register
Z	=	Temporary pseudo-result of a nonstore comparative operation

## REGISTER POSITIONS AND CONTENT SYMBOLS

R <sub>i</sub>	=	The i <sup>th</sup> position of R
R <sub>i...j</sub>	=	The positions i through j of R

R stands for any of the registers listed above as well as for a memory location or a pair of memory locations.

C(R)	=	The contents of the full register R
C(R) <sub>i</sub>	=	The contents of the i <sup>th</sup> position of R
C(R) <sub>i...j</sub>	=	The contents of the positions i through j of R

When the description of an instruction states a change only for a part of a register or memory location, then it is always understood that the part of the register or memory location which is not mentioned remains unchanged.

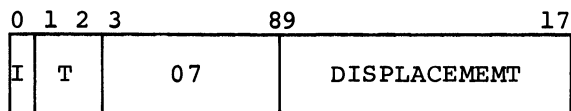
## OTHER PROCESSOR SYMBOLS

$\longrightarrow$	=	Replaces
::	=	Compare with
AND	=	The Boolean connective AND (symbol $\wedge$ )
OR	=	The Boolean connective OR (symbol $\vee$ )
$\equiv$	=	The Boolean connective NON-EQUIVALENCE (EXCLUSIVE OR)

# Memory Reference Instructions

## LOAD INSTRUCTIONS

LDA - Load A



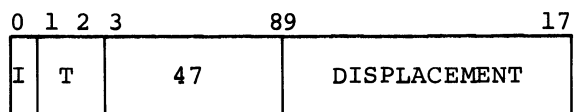
### Summary

$C(Y) \rightarrow C(A)$

### Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF  
Negative        If  $C(A)_0 = 1$ , then ON; otherwise OFF

LDQ - Load Q



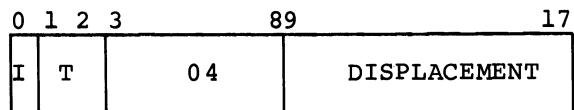
### Summary

$C(Y) \rightarrow C(Q)$

### Indicators

Zero            If  $C(Q) = 0$ , then ON; otherwise OFF  
Negative        If  $C(Q)_0 = 1$ , then ON; otherwise OFF

LADQ - Load Q



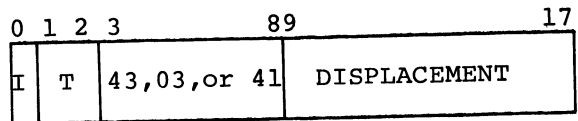
### Summary

$C(Y\text{-pair}) \rightarrow C(AQ)$

### Indicators

Zero            If  $C(AQ) = 0$ , then ON; otherwise OFF  
Negative        If  $C(AQ)_0 = 1$ , then ON; otherwise OFF

LDXn - Load Xn (n = 1,2,3)



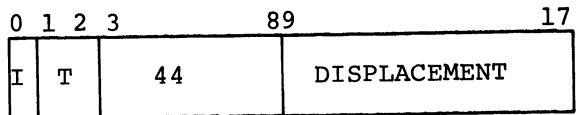
Summary

C(Y) (Bits 0-17) → C(Xn)

Indicators

Zero If C(Xn) = 0, then ON; otherwise OFF

LDI - Load Indicator Register



Summary

C(Y) (Bits 0-17,12-17) → C(X)

Indicators

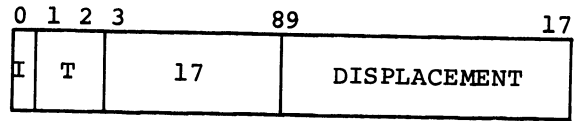
If corresponding bit in C(Y) is 1, then ON; otherwise OFF

NOTE: The relation between bit positions of C(Y) and the indicators is as follows:

<u>Bit Position</u>	<u>Indicators</u>
0	Zero
1	Negative
2	Carry
3	Overflow
4	Interrupt Inhibit
5	Parity fault Inhibit
6	Overflow fault Inhibit
7	Parity Error
8	} Not used
.	
.	
11	
12	} Input/output channel select register
.	
.	
.	
17	

STORE INSTRUCTIONS

STA - Store A



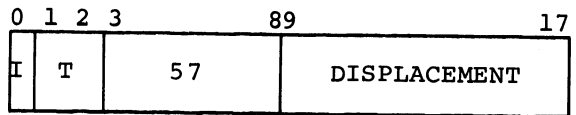
Summary

C(A) → C(Y)

Indicators

None affected

STQ - Store Q



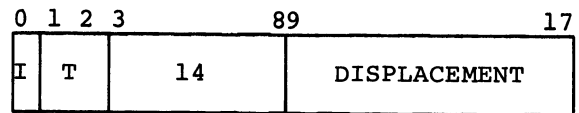
Summary

C(Q) → C(Y)

Indicators

None affected

STAQ - Store AQ



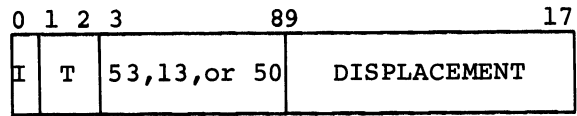
Summary

C(AQ) → C(Y-pair)

Indicators

None affected

STXn - Store Xn (n = 1,2,3)



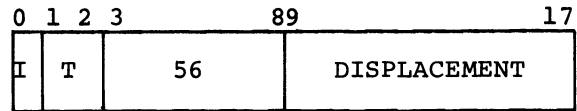
Summary

C(Xn) → C(Y) (Bits 0-17)

Indicators

None affected

STZ - Store Zero



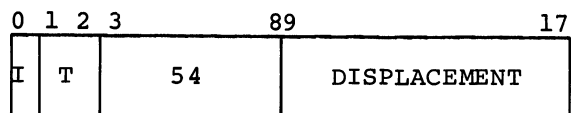
Summary

00...0 → C(Y)

Indicators

None affected

STI - Store Indicator Register



Summary

C(I) (Bits 0-7,12-17) → C(Y) (Bits 0-7,12-17)  
 0 → C(Y) (Bits 8-11)

Indicators

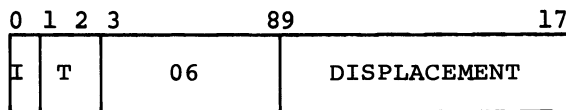
The relation between bit positions of C(Y) and the indicators is as follows:

<u>Bit Position</u>	<u>Indicators</u>
0	Zero
1	Negative
2	Carry
3	Overflow
4	Interrupt Inhibit
5	Parity fault Inhibit
6	Overflow fault Inhibit
7	Parity Error
8	Not used
.	
.	
11	
12	Input/output channel select register
.	
.	
.	
17	

The ON state corresponds to a ONE bit, the OFF state to a ZERO bit.

ADD INSTRUCTIONS

ADA - Add to A



Summary

C(A) + C(Y) → C(A)

Indicators

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If C(A) <sub>0</sub> = 1, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of A <sub>0</sub> is generated, then ON; otherwise OFF

ADQ - Add to Q

0	1	2	3	89	17
I	T	46		DISPLACEMENT	

Summary

$$C(Q) + C(Y) \longrightarrow C(Q)$$

Indicators

Zero	If $C(Q) = 0$ , then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$ , then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of $Q_0$ is generated, then ON; otherwise OFF

ADAQ - ADD to Q

0	1	2	3	89	17
I	T	15		DISPLACEMENT	

Summary

$$C(AQ) + C(Y\text{-pair}) \longrightarrow C(AQ)$$

Indicators

Zero	If $C(AQ) = 0$ , then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$ , then ON; otherwise OFF
Overflow	If range of AQ exceeded, then ON
Carry	If a carry out of $AQ_0$ is generated, then ON; otherwise OFF

ADCXn - Add Character Address to Xn (n = 1,2,3)

0	1	2	3	89	17
I	T	42,02, or 40		DISPLACEMENT	

Summary

$$FA [C(Xn), C(Y)] \longrightarrow C(X)$$

Indicators

Zero	If $C(Xn) = 0$ , then ON; otherwise OFF
------	---

The Fractional Add (FA) function is a special addition for character address (fractional) handling. The rules for forming the result:  $[FA C(Xn), CY]$  are shown in Figure 3-6 and Table 3-7. The entries in the table of FA function are the resulting octal contents of the C portion of the effective address.



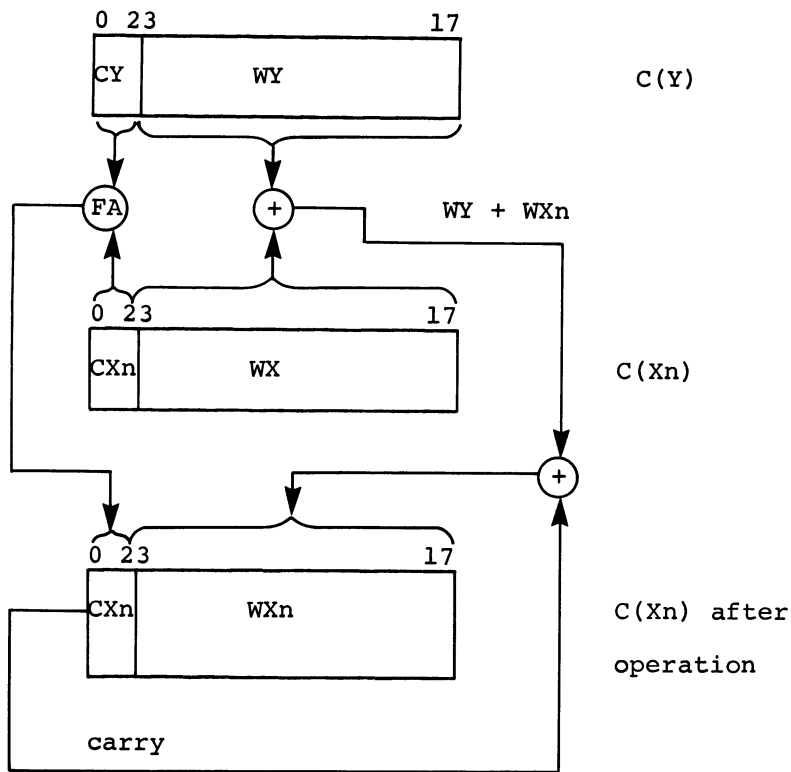


Figure 3-6. Memory Reference, Effective Address Formation Diagram

Table 3-7. Memory Reference, Fractional Add (FA) Function

C(Y) \ C(Xn)	C(Y)							
	Word 0	1	(0/2) 2	(1/2) 3	(0/3) 4	(1/3) 5	(2/3) 6	Illegal 7
Word 0	0	7	7	7	7	7	7	7
Word 1	7	7	7	7	7	7	7	7
(0/2) 2	7	7	2	3	7	7	7	7
(1/2) 3	7	7	3	2+carry	7	7	7	7
(0/3) 4	7	7	7	7	4	5	6	7
(1/3) 5	7	7	7	7	5	6	4+carry	7
(2/3) 6	7	7	7	7	6	4+carry	5+carry	7
Illegal 7	7	7	7	7	7	7	7	7

NOTE: 7 indicates an illegal combination which results in an illegal memory operation fault on the first memory reference.

ASA - Add A to Storage

0	1	2	3	89	17
I	T	16		DISPLACEMENT	

Summary

$$C(A) + C(Y) \longrightarrow C(Y)$$

Indicators

Zero            If  $C(Y) = 0$ , then ON; otherwise OFF  
 Negative       If  $C(Y)_0 = 1$ , then ON; otherwise OFF  
 Overflow       If range of Y is exceeded, then ON  
 Carry           If a carry out of  $Y_0$  is generated, then ON; otherwise OFF

AOS - Add One to Storage

0	1	2	3	89	17
I	T	76		DISPLACEMENT	

Summary

$$C(Y) + 0...01 \longrightarrow C(Y)$$

Indicators

Zero            If  $C(Y) = 0$ , then ON; otherwise OFF  
 Negative       If  $C(Y)_0 = 1$ , then ON; otherwise OFF  
 Overflow       If range of Y is exceeded, then ON  
 Carry           If a carry out of  $Y_0$  is generated, then ON; otherwise OFF

SUBTRACT INSTRUCTIONS

SBA - Subtract from A

0	1	2	3	89	17
I	T	26		DISPLACEMENT	

Summary

$$C(A) - C(Y) \longrightarrow C(A)$$

Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF  
 Negative       If  $C(A)_0 = 1$ , then ON; otherwise OFF  
 Overflow       If range of A is exceeded, then ON  
 Carry           If a carry out of  $A_0$  is generated, then ON; otherwise OFF

SBQ - Subtract from Q

0	1	2	3	89	17
I	T	66		DISPLACEMENT	

Summary

$$C(Q) - C(Y) \longrightarrow C(Q)$$

Indicators

Zero	If $C(Q) = 0$ , then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$ , then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of $Q_0$ is generated, then ON; otherwise OFF

SBAQ - Subtract from AQ

0	1	2	3	89	17
I	T	24		DISPLACEMENT	

Summary

$$C(AQ) - C(Y\text{-pair}) \longrightarrow C(AQ)$$

Indicators

Zero	If $C(AQ) = 0$ , then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$ , then ON; otherwise OFF
Overflow	If range of AQ exceeded, then ON
Carry	If a carry out of $AQ_0$ is generated, then ON; otherwise OFF

SSA - Subtract Stored from A

0	1	2	3	89	17
I	T	36		DISPLACEMENT	

Summary

$$C(A) - C(Y) \longrightarrow C(Y)$$

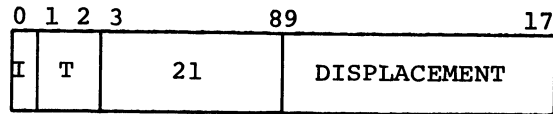
Indicators

Zero	If $C(Y) = 0$ , then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$ , then ON; otherwise OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of $Y_0$ is generated, then ON; otherwise OFF



# DIVIDE INSTRUCTIONS

## DVF - Divide Fraction



### Summary

$C(AQ) + C(Y)$ ; fractional quotient  $\longrightarrow C(A)$   
remainder  $\longrightarrow C(Q)$

### Indicators

#### If division takes place:

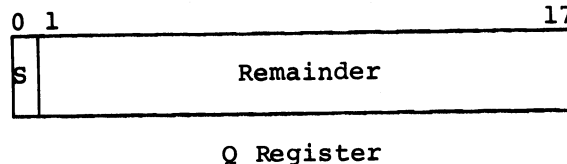
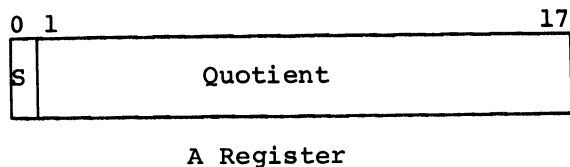
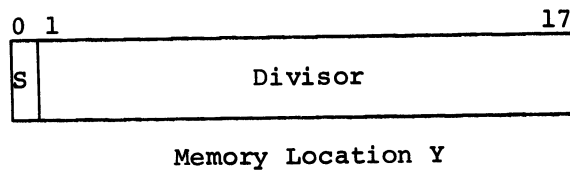
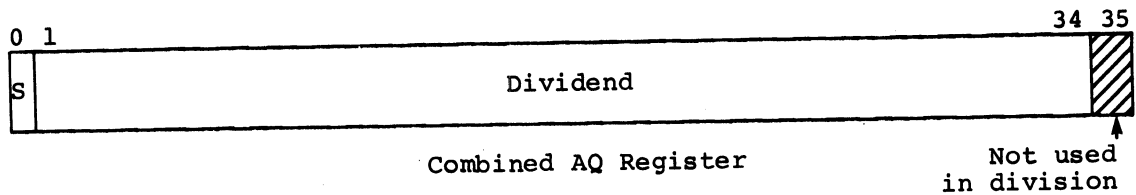
Zero                    If  $C(A) = 0$ , then ON; otherwise OFF  
Negative                If  $C(A)_0 = 1$ , then ON; otherwise OFF

#### If no division takes place:

Zero                    If divisor = 0, then ON; otherwise OFF  
Negative                If dividend < 0, then ON; otherwise OFF

### Notes

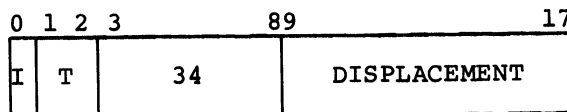
1. A 35-bit fractional dividend (including sign) is divided by an 18-bit fractional divisor (including sign) to form an 18-bit quotient (including sign) and an 18-bit remainder (including sign). Bit position 17 of the remainder corresponds to bit position 34 of the dividend. The remainder sign is equal to the dividend sign unless the remainder is zero.



2. If  $|dividend| \geq |divisor|$ , or if  $divisor = 0$ , then the division itself does not take place. Instead, a divide fault trap occurs; the divisor  $C(Y)$  remains unchanged, and  $C(AQ)$  is the dividend.

BOOLEAN INSTRUCTIONS

ANA - AND to A



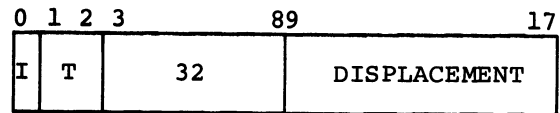
Summary

$$C(A)_i \text{ AND } C(Y)_i \longrightarrow C(A)_i \text{ for all bit } i = 0, 1, \dots, 17$$

Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(A)_0 = 1$ , then ON; otherwise OFF

ANSA - AND to Storage A



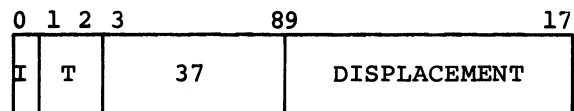
Summary

$C(A)_i \text{ AND } C(Y)_i \longrightarrow C(Y)_i$  for all bit  $i = 0, 1, \dots, 17$

Indicators

Zero            If  $C(Y) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(Y)_0 = 1$ , then ON; otherwise OFF

ORA - OR to A



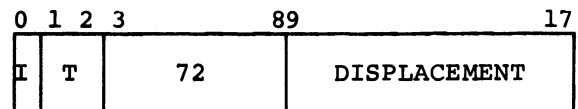
Summary

$C(A)_i \text{ OR } C(Y)_i \longrightarrow C(A)_i$  for all bit  $i = 0, 1, \dots, 17$

Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(A)_0 = 1$ , then ON; otherwise OFF

ORSA - OR to Storage A



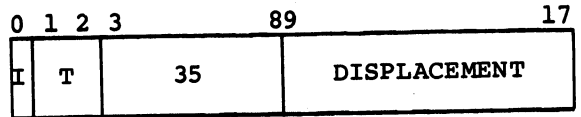
Summary

$C(A)_i \text{ OR } C(Y)_i \longrightarrow C(Y)_i$  for all bit  $i = 0, 1, \dots, 17$

Indicators

Zero            If  $C(Y) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(Y)_0 = 1$ , then ON; otherwise OFF

ERA - EXCLUSIVE OR to A



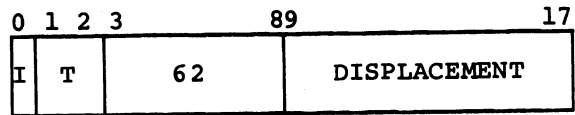
Summary

$$C(A)_i \neq C(Y)_i \rightarrow C(A)_i \text{ for all bit } i = 0, 1, \dots, 17$$

Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(A)_0 = 1$ , then ON; otherwise OFF

ERSA - EXCLUSIVE OR to Storage A



Summary

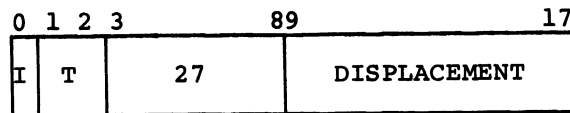
$$C(A)_i \neq C(Y)_i \rightarrow C(Y)_i \text{ for all bit } i = 0, 1, \dots, 17$$

Indicators

Zero            If  $C(Y) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(Y)_0 = 1$ , then ON; otherwise OFF

COMPARE INSTRUCTIONS

CMPA - Compare with A



Summary

Comparison  $C(A) : : C(Y)$

Indicators

Algebraic Comparison - (Signed fixed-point)

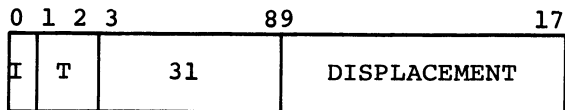
Zero ↓	Negative ↓	Carry ↓	Relation	Sign
0	0	0	$C(A) > C(Y)$	$C(A)_0 = 0, C(Y)_0 = 1$
0	0	1	$C(A) > C(Y)$	
1	0	1	$C(A) = C(Y)$	$C(A)_0 = C(Y)_0$
0	1	0	$C(A) < C(Y)$	
0	1	1	$C(A) < C(Y)$	$C(A)_0 = 1, C(Y)_0 = 0$



Logic Comparison - (Unsigned fixed point)

<u>Zero</u>	<u>Carry</u>	<u>Relation</u>
0	0	$C(A) < C(Y)$
1	1	$C(A) = C(Y)$
0	1	$C(A) > C(Y)$

CANA - Compare AND with A



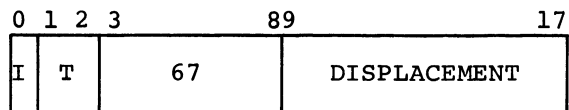
Summary

$$Z_i = C(A)_i \text{ AND } C(Y)_i \text{ for all bits } i = 0, 1, \dots, 17$$

Indicators

Zero            If  $Z = 0$ , then ON; otherwise OFF  
 Negative        If  $Z_0 = 1$ , then ON; otherwise OFF

CMPQ - Compare with Q



Summary

Comparison  $C(Q) :: C(Y)$

Indicators

Algebraic Comparison

<u>Zero</u>	<u>Negative</u>	<u>Carry</u>	<u>Relation</u>	<u>Sign</u>
0	0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0, C(Y)_0 = 1$
0	0	1	$C(Q) > C(Y)$	
1	0	1	$C(Q) = C(Y)$	$C(Q)_0 = C(Y)_0$
0	1	0	$C(Q) < C(Y)$	
0	1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1, C(Y)_0 = 0$

Logic Comparison

<u>Zero</u>	<u>Carry</u>	<u>Relation</u>
0	0	$C(Q) < C(Y)$
1	1	$C(Q) = C(Y)$
0	1	$C(Q) > C(Y)$

CMPXn - Compare with Xn

	0	1	2	3		89		17
I		T	63,23,or 61			DISPLACEMENT		

Summary

Comparison C(Xn) :: C(Y)

Indicators

Zero            If C(Xn) = C(Y), then ON; otherwise OFF

SZN - Set Zero and Negative Indicators from Storage

	0	1	2	3		89		17
I		T	20			DISPLACEMENT		

Summary

Test the number C(Y)

Indicators

<u>Zero</u>	<u>Negative</u>	<u>Relation</u>
0	0	Number C(Y) > 0
1	0	Number C(Y) = 0
0	1	Number C(Y) < 0

TRANSFER INSTRUCTIONS

TRA - Transfer Unconditionally

	0	1	2	3		89		17
I		T	71			DISPLACEMENT		

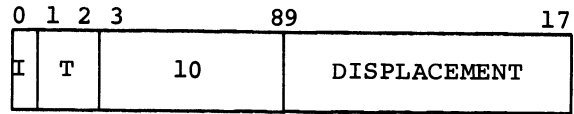
Summary

Y → C(IC)

Indicators

None affected

TSY - Transfer and Store IC in Y



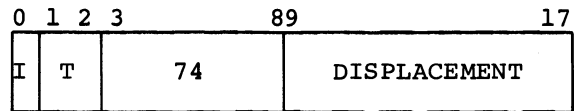
Summary

$C(IC) + 1 \longrightarrow C(Y), Y + 1 \longrightarrow C(IC)$

Indicators

None affected

TZE - Transfer on Zero



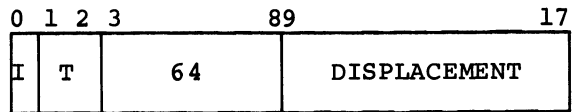
Summary

If zero indicator is ON, then  $Y \longrightarrow C(IC)$

Indicators

None affected

TNZ - Transfer on Not Zero



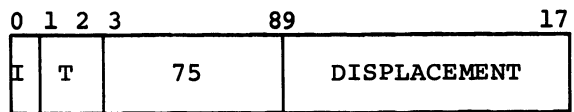
Summary

If zero indicator OFF, then  $Y \longrightarrow C(IC)$

Indicators

None affected

TMI - Transfer on Minus



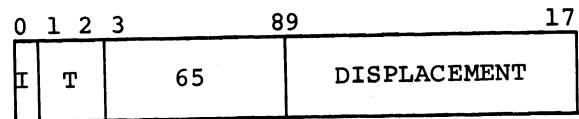
Summary

If negative indicator ON, then  $Y \longrightarrow C(IC)$

Indicators

None affected

TPL - Transfer on Plus



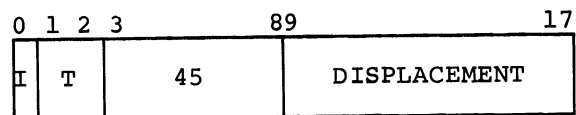
Summary

If negative indicator OFF, then  $Y \rightarrow C(IC)$

Indicators

None affected

TNC - Transfer on No Carry



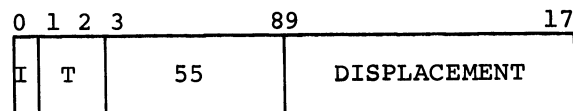
Summary

If carry indicator OFF, then  $Y \rightarrow C(IC)$

Indicators

None affected

TOV - Transfer on Overflow



Summary

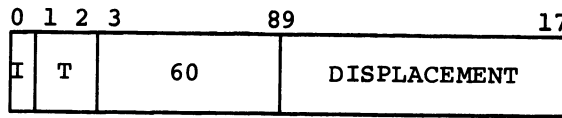
If overflow indicator ON, then  $Y \rightarrow C(IC)$  and the overflow indicator is turned OFF

Indicators

Overflow      Set OFF

## INPUT/OUTPUT INSTRUCTIONS

### CIOC - Connect Input/Output Channel



#### Summary

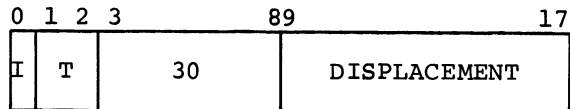
The contents of the effective address C(Y\*) and the contents of the I/O channel select register are transmitted to the input/output controller (IOM).

#### Indicators

None affected

NOTE: Y\* (effective address) is the address of a Peripheral Control Word (PCW) for the input/output channel specified by the contents of the I/O channel select register. The IOM accesses the double-precision (36-bit) PCW at Y\* and sends it, or portions thereof, to the channel indicated.

### LDEX - Load External Register



#### Summary

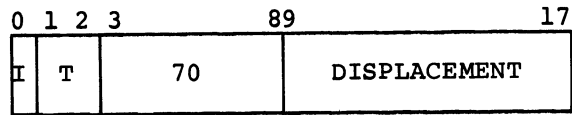
The contents of the effective address (Y\*) and the contents of the I/O channel select register are transmitted to the input/output controller (IOM).

#### Indicators

None affected

NOTE: Y\* (effective address) is the address of a data word accessed by the IOM and sent to the input/output channel specified by the I/O channel select register.

STEX - Store External Register



Summary

The effective address C(Y\*) and the contents of the input/out channel select register are sent to the input/output controller (IOM).

Indicators

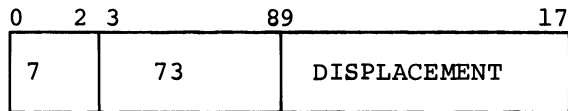
None affected

NOTE: The IOM stores the data from the selected input/output channel at the address specified by the effective address Y\*.

Nonmemory Reference Instructions

GROUP 1, IMMEDIATE ADD INSTRUCTIONS

IAA - Immediate Add to A



Summary

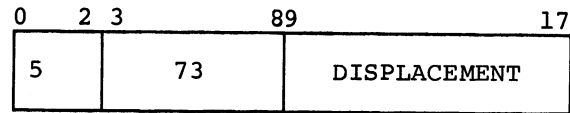
$C(A) + D \text{ (Bits 9-17)} \longrightarrow C(A)$

Indicators

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If C(A) <sub>0</sub> = 1, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON
Carry	If carry out of A <sub>0</sub> is generated, then ON; otherwise OFF

NOTE: The D field (9-17) is treated as a 2's complement number. Before the add occurs, the sign bit, Y<sub>9</sub>, is extended nine places to the left to form an 18-bit, signed operand.

IAQ - Immediate Add to Q



Summary

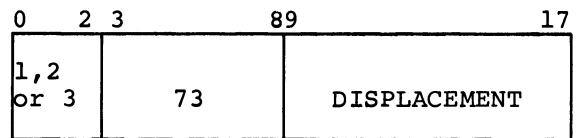
$$C(Q) + D \text{ (Bits 9-17)} \longrightarrow C(Q)$$

Indicators

- Zero            If  $C(Q) = 0$ , then ON; otherwise OFF
- Negative      If  $C(Q)_0 = 1$ , then ON; otherwise OFF
- Overflow      If range of Q is exceeded, then ON
- Carry          If carry out of  $Q_0$  is generated, then ON; otherwise OFF

NOTE: The D field (9-17) is treated as a 2's complement number. Before the add occurs, the sign bit,  $Y_9$ , is extended nine places to the left to form an 18-bit signed operand.

IACXn - Immediate Add Character Address to Xn            (n = 1,2,3)



Summary

$$FA [C(Xn), D \text{ (Bits 9-17)}] \longrightarrow C(Xn)$$

Indicators

- Zero            If  $C(Xn) = 0$ , then ON; otherwise OFF

NOTE: This instruction increases (or decreases) X by the number of characters and/or words specified in the D field of the instruction.

The Fractional Add (FA) function, is a special addition for character address (fractional) handling. The rules for forming the result:  $(FA C(Xn), D)$  are shown in the following diagram shown in Figure 3-7 and Table 3-8. The entries in Table 3-8 are the resulting octal contents of the C position of the effective address.

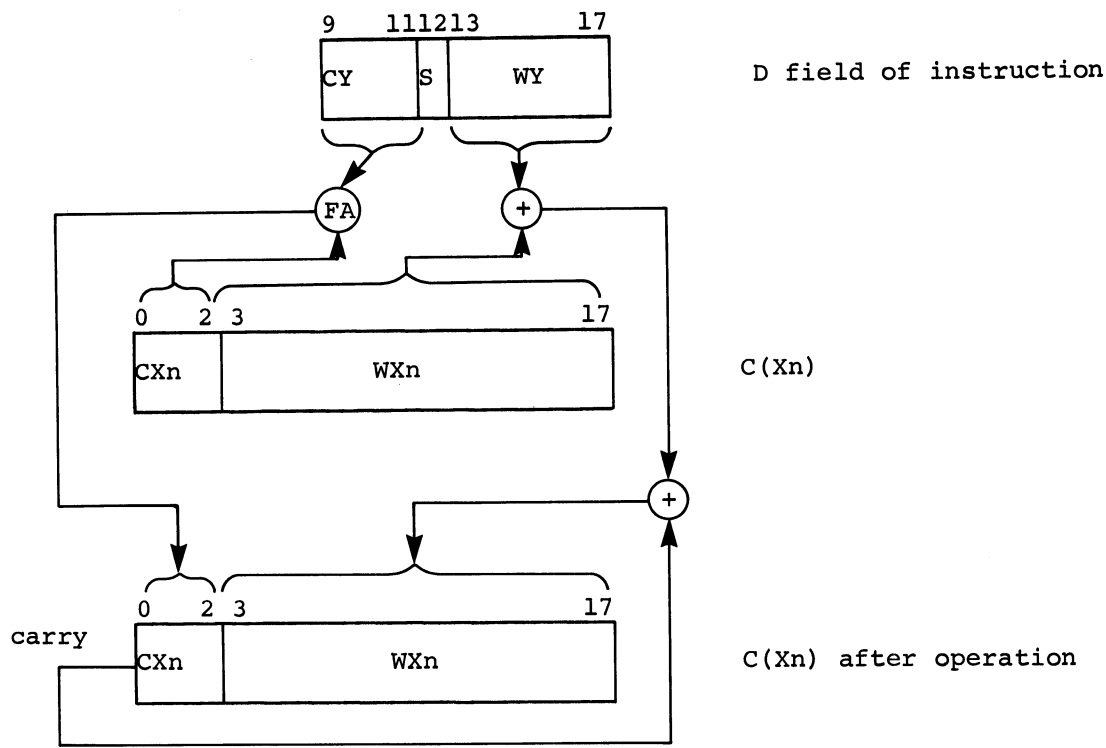


Figure 3-7. Nonmemory Reference, Effective Address Formation Diagram

Table 3-8. Nonmemory Reference, Fractional Add (FA) Function

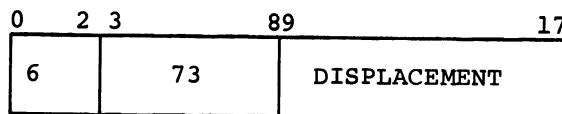
D9..11	Word	0	1	(0/2) 2	(1/2) 3	(0/3) 4	(1/3) 5	(2/3) 6	Illegal 7
	CXn								
Word	0	0	7	7	7	7	7	7	7
	1	7	7	7	7	7	7	7	7
(0/2)	2	7	7	2	3	7	7	7	7
(1/2)	3	7	7	3	2+carry	7	7	7	7
(0/3)	4	7	7	7	7	4	5	6	7
(1/3)	5	7	7	7	7	5	6	4+carry	7
(2/3)	6	7	7	7	7	6	4+carry	5+carry	7
Illegal	7	7	7	7	7	7	7	7	7

NOTE: 7 indicates an illegal combination which results in an illegal memory operation fault on the first memory reference.



GROUP 1, IMMEDIATE LOAD INSTRUCTIONS

ILA - Immediate Load A



Summary

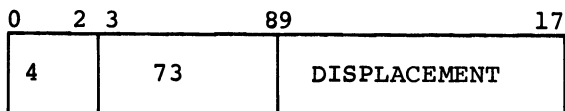
Y (Bits 9-17) → C(A)

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
Negative        If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

NOTE: The Y field (bits 9-17) is treated as a 2's complement number. When the load occurs the sign bit, Y<sub>9</sub>, is extended to A<sub>0</sub>.

ILQ - Immediate Load



Summary

Y (Bits 9-17) → C(Q)

Indicators

Zero            If C(Q) = 0, then ON; otherwise OFF  
Negative        If C(Q)<sub>0</sub> = 1, then ON; otherwise OFF

NOTE: The Y field (bits 9-17) is treated as a 2's complement number. When the load occurs, the sign bit, Y<sub>9</sub>, is extended to Q<sub>0</sub>.

GROUP 1, INTERRUPT CONTROL INSTRUCTIONS

SEL - Select Input/Output Channel

0	2 3	89	17
0	73	CHANNEL NO.	

Summary

Y (Bits 12-17) → input/output select register

Indicators

None affected

NOTE: The contents of the input/output select register selects the channel for the CIOC, STEx, and LDEX instructions.

RIER - Read Interrupt Level Enable Register

0	2 3	89	17
0	12	000	

Summary

The contents of the interrupt level enable register are transferred to the C(A) bits 0-15. The C(A) bits 16-17 are set to 0.

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
 Negative        If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

RIA - Read Interrupt Address

0	2 3	89	17
4	12	000	

Summary

The address corresponding to the highest priority interrupt cell, which is ON and enabled, is transferred to C(A) bits 10-17. The C(A) bits 1-9 are set to 0. The C(A) bit 0 is set to 1. The interrupt cell is then reset.

If no interrupt cells are ON and enabled, the C(A) are set to the location of the Illegal Program Interrupt Fault Vector with the Indirect bit set; C(A) = 400447 (octal).

Indicators

Zero            - set OFF  
 Negative        - set ON

NOTE: The level of the interrupt cell occupies C(A) bits 14-17. The channel number of the interrupt cell occupies C(A) bits 10-13.

SIER - Set Interrupt Level Enable Register

0	2 3	89	17
0	52	000	

Summary

The C(A) bits 0-15 are transferred to the 16-bit interrupt level enable register.

Indicators

None affected

NOTE: When a bit in the interrupt level enable register is 1, the program interrupt level corresponding to that bit is enabled.

SIC - Set Interrupt Cells

0	2 3	89	13 14	17
4	52	00	CELL NO.	

Summary

The C(A) register (bits 0-15) is set by an OR operation with the interrupt cells on the level specified by Y (bits 14-7).

Indicators

None affected

GROUP 1, IMMEDIATE BOOLEAN INSTRUCTIONS

IANA - Immediate AND to A

0	2 3	89	17
0	22	Y	

Summary

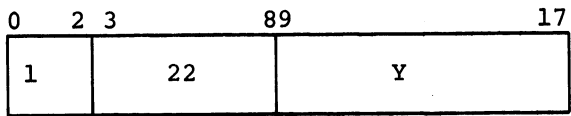
$C(A)_i \text{ AND } Y_i \longrightarrow C(A)_i$  for all  $i = 0, 1, 2, \dots, 17$

Indicators

Zero            If  $C(A)_0 = 0$ , then ON; otherwise OFF  
 Negative        If  $C(A)_0 = 1$ , then ON; otherwise OFF

NOTE: The value of Y (9-17) is treated as a 2's complement number. When the AND occurs, the value of Y9 (sign) is extended nine places to the left to form an 18-bit signed operand.

IORA - Immediate OR to A



Summary

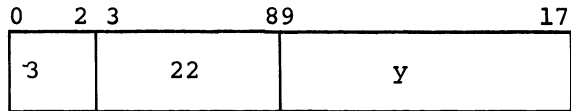
$C(A)_i \text{ OR } Y_i \rightarrow C(A)_i$  for all  $i = 0, 1, \dots, 17$

Indicators

Zero            If  $C(A) = 0$ , then ON, otherwise OFF  
 Negative        If  $C(A)_0 = 1$ , then ON, otherwise OFF

NOTE: The value of Y (9-17) is treated as a 2's complement number. When the OR occurs, the value of Y9 (sign) is extended nine places to the left to form an 18-bit signed operand.

IERA - Immediate EXCLUSIVE OR to A



Summary

$C(A)_i \neq Y_i$  for all bits  $i = 0, 1, \dots, 17$

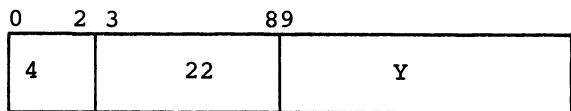
Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(A)_0 = 1$ , then ON; otherwise OFF

NOTE: The value Y (9-17) is treated as a 2's complement number. When the EXCLUSIVE OR occurs, the value of Y9 (sign) is extended nine places to the left to form an 18-bit signed operand.

GROUP 1, IMMEDIATE COMPARE INSTRUCTIONS

ICMPA - Immediate Compare A



Summary

Comparison  $C(A) : : Y$

## Indicators

### Algebraic Comparison

<u>Zero</u>	<u>Negative</u>	<u>Carry</u>	<u>Relation</u>	<u>Sign</u>
0	0	0	$C(A) > Y$	$C(A)_0 = 0, Y9 = 1$
0	0	1	$C(A) > Y$	$C(A)_0 = Y9$
1	0	1	$C(A) = Y$	
0	1	0	$C(A) < Y$	
0	1	1	$C(A) < Y$	$C(A)_0 = 1, Y9 = 0$

### Logic Comparison

<u>Zero</u>	<u>Carry</u>	<u>Relation</u>
0	0	$C(A) < Y$
1	1	$C(A) = Y$
0	1	$C(A) > Y$

NOTE: The value of Y (9-17) is treated as a 2's complement number. When the Comparison occurs, the value of Y9 (sign) is extended nine places to the left to form an 18-bit signed operand.

ICANA - Immediate Comparative AND with A

0	2 3	89	17
2	22	Y	

### Summary

$$Z_i = C(A)_i \text{ AND } Y_i \text{ for all bits } i = 0, 1, \dots, 17$$

### Indicators

Zero            If  $Z = 0$ , then ON; otherwise OFF  
Negative        If  $Z_0 = 1$ , then ON; otherwise OFF

NOTE: The value of Y (9-17) is treated as a 2's complement number. When the COMPARATIVE AND occurs, the value Y9 (sign) is extended nine places to the left to form an 18-bit signed operand.

GROUP 2, DATA MOVEMENT SHIFT INSTRUCTIONS

ARS - A Right Shift

0	2 3	89	1112	17
0	33	7	COUNT	

Summary

Shift right C(A) by Y (12-17) positions; fill vacated positions with C(A)<sub>0</sub>.

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
 Negative        If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

QRS - Q Right Shift

0	2 3	89	1112	17
4	33	7	COUNT	

Summary

Shift right C(Q) by Y (12-17) positions; fill vacated positions with C(Q)<sub>0</sub>.

Indicators

Zero            If C(Q) = 0, then ON; otherwise OFF  
 Negative        If C(Q)<sub>0</sub> = 1, then ON; otherwise OFF

LRS - Long Right Shift

0	2 3	89	1112	17
0	33	5	COUNT	

Summary

Shift right C(AQ) by Y (12-17) positions; fill vacated positions with C(AQ)<sub>0</sub>.

Indicators

Zero            If C(AQ) = 0, then ON; otherwise OFF  
 Negative        If C(AQ)<sub>0</sub> = 1, then ON; otherwise OFF

ALS - A Left Shift

0	2 3	8 9	1 1 1 2	1 7
0	3 3	6	COUNT	

Summary

Shift left C(A) by Y (12-17) positions, fill vacated positions with zeros.

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
 Negative       If C(A)<sub>0</sub> = 1, then ON; otherwise OFF  
 Carry          If C(A)<sub>0</sub> ever changes during the shift, then ON; otherwise OFF

QLS - Q Left Shift

0	2 3	8 9	1 1 1 2	1 7
4	3 3	6	COUNT	

Summary

Shift left C(Q) by Y (12-17) positions, fill vacated positions with zeros.

Indicators

Zero            If C(Q) = 0, then ON; otherwise OFF  
 Negative       If C(Q)<sub>0</sub> = 1, then ON; otherwise OFF  
 Carry          If C(Q)<sub>0</sub> ever changes during the shift, then ON; otherwise OFF

LLS - Long Left Shift

0	2 3	8 9	1 1 1 2	1 7
0	3 3	4	COUNT	

Summary

Shift left C(AQ) by Y (12-17) positions; fill vacated positions with zeros.

Indicators

Zero            If C(AQ) = 0, then ON; otherwise OFF  
 Negative       If C(AQ)<sub>0</sub> = 1, then ON; otherwise OFF  
 Carry          If C(AQ)<sub>0</sub> ever changes during the shift, then ON; otherwise OFF

ARL - A Right Logic

0	2 3	89	1112	17
2	33	7	COUNT	

Summary

Shift right C(A) by Y (12-17) positions; fill vacated positions with zeros.

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
 Negative       If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

QRL - Q Right Logic

0	2 3	89	1112	17
6	33	7	COUNT	

Summary

Shift right C(Q) by Y (12-17) positions; fill vacated positions with zeros.

Indicators

Zero            If C(Q) = 0, then ON; otherwise OFF  
 Negative       If C(Q)<sub>0</sub> = 1, then ON; otherwise OFF

LRL - Long Right Logic

0	2 3	89	1112	17
2	33	5	COUNT	

Summary

Shift right C(AQ) by Y (12-17) positions; fill vacated positions with zeros.

Indicators

Zero            If C(AQ) = 0, then ON; otherwise OFF  
 Negative       If C(AQ)<sub>0</sub> = 1, then ON; otherwise OFF



ALR - A Left Rotate

0	2 3	89	1112	17
2	33	6	COUNT	

Summary

Rotate C(A) by Y (12-17) positions; enter each bit leaving position zero into position 17.

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
 Negative       If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

ALP - A Left Parity Rotate

0	2 3	89	1112	17
3	33	6	COUNT	

Summary

Rotate C(A) by Y (12-17) positions, enter each bit leaving position zero into position 17.

Indicators

Zero            If the number of 1's leaving position 0 is even, then ON; otherwise OFF  
 Negative       If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

QLR - Q Left Rotate

0	2 3	89	1112	17
6	33	6	COUNT	

Summary

Rotate C(Q) by Y (12-17) positions; enter each bit leaving position zero into position 17.

Indicators

Zero            If C(Q) = 0, then ON; otherwise OFF  
 Negative       If C(Q)<sub>0</sub> = 1, then ON; otherwise OFF

QLP - Q Left Parity Rotate

0	2 3	89	1112	17
7	33	6	COUNT	

Summary

Rotate C(Q) by Y (12-17) positions, enter each bit leaving position zero into position 17.

Indicators

Zero            If the number of 1's leaving position 0 is even, then ON; otherwise OFF  
 Negative        If  $C(Q)_0 = 1$ , then ON; otherwise OFF

LLR - Long Left Rotate

0	2 3	89	1112	17
2	33	4	COUNT	

Summary

Rotate C(AQ) by Y (12-17) positions; enter each bit leaving position zero into position 35.

Indicators

Zero            If  $C(AQ) = 0$ , then ON; otherwise OFF  
 Negative        If  $C(AQ)_0 = 1$ , then ON; otherwise OFF

GROUP 2, DATA MOVEMENT NORMALIZE INSTRUCTIONS

NRM - Normalize

0	2 3	89	1112	17
1	33	6	00	

Summary

C(A) normalized → C(A)

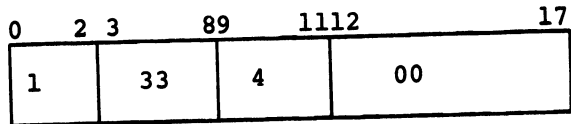
Indicators

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If C(A) <sub>0</sub> = 1, then ON; otherwise OFF
Overflow	Set OFF

Notes

1. If the overflow indicator is OFF, this instruction shifts the C(A) left until the C(A)<sub>0</sub> ≠ C(A)<sub>1</sub>. The C(X1) are decremented for each position shifted.
2. If the overflow indicator is ON, the C(A) are shifted right one position, and then the sign bit C(A)<sub>0</sub> is inverted to reconstitute the original sign. Furthermore, the overflow indicator is set OFF and the C(X1) are incremented.
3. This instruction can be used to correct overflows.
4. If C(A) = 0, then no shift occurs.
5. X1 is decremented or incremented using the full adder (18-bit field), not the character address addition used with other index register operations.

NRML - Normalize Long



Summary

C(AQ) normalized → C(AQ)

Indicators

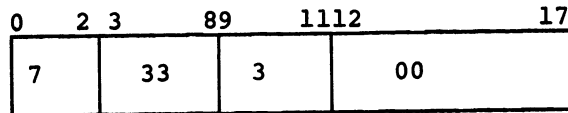
Zero            If C(AQ) = 0, then ON; otherwise OFF  
 Negative        If C(AQ)<sub>0</sub> = 1, then ON; otherwise OFF  
 Overflow        Set OFF

Notes

1. If the overflow indicator is OFF, this instruction shifts the C(AQ) left until C(AQ)<sub>0</sub> ≠ C(A)<sub>1</sub>. The C(X1) are decremented for each position shifted.
2. If the overflow indicator is ON, the C(AQ) are shifted right one position, and then the sign bit C(AQ)<sub>0</sub> is inverted to reconstitute the original sign. Furthermore, the overflow indicator is set OFF and the C(X1) are incremented.
3. This instruction can be used to correct overflows.
4. If C(AQ) = 0, then no shift occurs.
5. X1 is decremented or incremented using the full adder (18-bit field), not the character address addition used with other index register operations.

GROUP 2, DATA MOVEMENT COPY INSTRUCTIONS

CQA - Copy AQ into A



Summary

C(Q) → C(A), C(Q) does not change

Indicators

Zero            If C(A) = 0, then ON; otherwise OFF  
 Negative        If C(A)<sub>0</sub> = 1, then ON; otherwise OFF

CAQ - Copy A into Q

0	2 3	89	1112	17
6	33	3	00	

Summary

$C(A) \longrightarrow C(Q)$ ,  $C(A)$  does not change

Indicators

Zero            If  $C(Q) = 0$ , then ON; otherwise OFF  
Negative        If  $C(Q)_0 = 1$ , then ON; otherwise OFF

CAXn - Copy A into Xn            (n = 1,2,3)

0	2 3	89	1112	17
4,0,4	33	2,2,3	00	

Summary

$C(A) \longrightarrow C(X_n)$ ,  $C(A)$  does not change

Indicators

Zero            If  $C(X_n) = 0$ , then ON; otherwise OFF

CXnA - Copy Xn into A            (n = 1,2,3)

0	2 3	89	1112	17
2,3,3	33	2,2,3	00	

Summary

$C(X) \longrightarrow C(A)$ ,  $C(X_n)$  does not change

Indicators

Zero            If  $C(A) = 0$ , then ON; otherwise OFF

GROUP 2, INTERRUPT CONTROL INSTRUCTIONS

INH - Interrupt Inhibit

0	2 3	89	1112	17
3	33	1	00	

Summary

Program execute interrupts are inhibited

Indicators

Interrupt inhibit indicator is turned ON

ENI - Enable Interrupt

0	2 3	89	1112	17
7	33	1	00	

Summary

Program execute interrupts are enabled

Indicators

Interrupt inhibit indicator is turned OFF

GROUP 2, MISCELLANEOUS INSTRUCTIONS

DIS - Delay Until Interrupt Signal

0	2 3	89	1112	17
4	33	1	00	

Summary

No operation takes place; the processor waits for a program interrupt signal.

Indicators

None affected

NOTE: An interrupt may occur even if the Interrupt inhibit indicator is turned ON.

NOP - No Operation

0	2	3	89	1112	15	16	17
2	33	1	00	X	X		

Summary

No operation takes place.

Indicators

None affected

NOTE: XX are optional sync bits for test and diagnostics only. These bits must be zero for all other users.

## SECTION IV

### PSEUDO-OPERATIONS

Pseudo-operations are so called because of their similarity to machine operations in an object program. In general, machine operations are produced by computer instructions and perform some task, or part of a task, directly concerned with solving the problem at hand. Pseudo-operations work indirectly on the problem by performing machine conditioning functions, such as allocating memory, and by directing the assembler in the preparation of machine coding. A pseudo-operation affecting the assembler may generate several, one, or no words in the object program.

Pseudo-operations are grouped, in this section, according to function:

<u>FUNCTIONAL GROUP</u>	<u>PRINCIPAL USES</u>
Control	Selects printout options for the assembly listing, directs punchout for absolute/relocatable binary program decks, and selects format for the absolute binary deck.
Location counter	Provides programmer control for single or multiple location counters.
Symbol definition	Defines assembler source program symbols by means other than listing in the location field of the coding form.
Data generation	Produces binary data words for the assembly program.
Memory allocation	Provides programmer control for the use of memory.
Special	Generates zero operation code instructions and continued subfields for selected pseudo-operations.
Macro	Begins and ends macro prototypes; provides assembler generation of macro-argument symbols; and repeats substitution of arguments within macro prototypes.
Conditional assembly	Provides conditional assembly of variable numbers of input words based upon the subfield entries of these pseudo-operations.
Program linkage	Generates standard system subroutine calling sequences and return (exit) linkages.



Table 4-1 contains a complete listing of pseudo-operation mnemonic codes.

Table 4-1. Pseudo-Operations by Functional Class

Pseudo-Operation  
Mnemonic

Functions

Control Pseudo-Operations

ABS	Output Absolute Text
CPR	Copyright
DCARD	Punch BCD Card
DETAIL ON/OFF	Detail Output Listing
EDITP ON/OFF	Edit Print Lines
EJECT	Restore Output Listing
END	End of Assembly
FUL	Output Full Binary Text
HEAD	Heading
LBL	Label
LIST ON/OFF	Control Output Listing
OPD	Operation Definition
OPSYN	Operation Synonym
PARITY ON/OFF	ASCII Parity Control
PCC ON/OFF	Print Control Cards
PMC ON/OFF	Print Macro Expansion
PUNCH ON/OFF	Control Card Output
REFMA ON/OFF	Reference Macro Operation
REF	References
REM	Remarks
TCD	Punch Transfer Card
TTL	Title
TTLS	Subtitle
*	Remarks

Location Counter Pseudo-Operations

BEGIN	Origin of a Location Counter
LOC	Location of Output Text
ORG	Origin Set by Programmer
USE	Use Multiple Location Counters

Symbol Defining Pseudo-Operations

BASE	Force Location Counter to a Multiple Power of 2
BOOL	Boolean
EIGHT	Force Location Counter to Multiple of 8
EQU	Equal To
EVEN	Force Location Counter Even
FEQU	Special FORTRAN Equivalence
MIN	Minimum
MAX	Maximum
NULL	Null
ODD	Force Location Counter Odd
SET	Symbol Redefinition
SYMDEF	Symbol Definition
SYMREF	Symbol Reference

Table 4-1. (cont) Pseudo-Operations

Data Generating Pseudo-Operations

ASCII,ACI,ASCIIC,ACIC	ASCII Coded Information
BCI	Binary Coded Decimal Information
DEC	Decimal
DUP	Duplicate Cards
OCT	Octal
SACI	Symbolic ASCII Information
VFD	Variable Field Definition

Memory Allocation Pseudo-Operations

BFS	Block Followed by Symbol
BLOCK	Block Common
BSS	Block Started by Symbol
LIT	Literal Pool Origin

Conditional Pseudo-Operations

IFE	If Equal
IFG	If Greater Than
IFL	If Less Than
INE	If Not Equal

Special Word Format Pseudo-Operations

ARG	Argument--Generate Zero Operation Code Computer Word
DATE	Current Date
IND	Generate Word for Indirect Addressing
MARK	Specify Symbol in Location Field
MAXSZ	Maximum Size of Assembly
NONOP	Undefined Operation
TTLDAT	Title Date
ZERO	Generate Word with Two Subfields

Data Control Word Format Pseudo-Operations

DCW	I/O Control Word Generator
ICW	I/O Control Word Generator

Macro Pseudo-Operations

CRSM ON/OFF	Created Symbols
DELM	Delete Macro Named
ENDM	End Macro Prototype
IDRP	Indefinite Repeat
LODM	Load System Macro Prototypes
MACRO	Macro Identification
ORGCSM	Origin Created Symbols
PUNM	Punch Macro Prototypes

Program Linkage Pseudo-Operations

CALL	Call Subroutines
SAVE	Save-Return Linkage Data
RETURN	Return from Subroutines



## DETAIL ON/OFF - Detail Output Listing

12	8	16
Blanks	DETAIL	ON (Normal mode)
Blanks	DETAIL	OFF

Some pseudo-operations generate no binary words; however, several of them generate more than one. The generative pseudo-operations are: ACI, ACIC, ASCII, ASCIIC, ICW, DCW, OCT, DEC, BCI, DUP, CALL, SAVE, RETURN, and VFD. The DETAIL pseudo-operation provides control over the amount of listing detail generated by the generative pseudo-operations.

The use of the DETAIL OFF pseudo-operation causes the assembly listing to be abbreviated by eliminating all but the first word generated by any of the above pseudo-operations. In the case of the DUP pseudo-operation, only the first iteration will be listed. The DETAIL ON pseudo-operation causes the assembler to resume the listing which had been suspended by a DETAIL OFF pseudo-operation.

If the assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

## LIST ON/OFF - Control Output Listing

12	8	16
Blanks	LIST	ON (Normal mode)

The LIST pseudo-operation with OFF in the variable field causes the normal listing to change as follows: LIST OFF will appear in the listing; thereafter, only instructions which are flagged in error appear. If the assembly ends in the LIST OFF mode, only the error messages appear.

The LIST pseudo-operation with ON in the variable field causes the normal listing, which was suspended by a LIST OFF pseudo-operation, to be resumed. If the assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

## PCC ON/OFF - Print Control Cards

1	8	16
Blanks	PCC	OFF (Normal mode)

The PCC pseudo-operation affects the listing of the following pseudo-operations:

DETAIL	LIST	TTL	PMC	EDITP
EJECT	ABS	TTLS	PUNCH	PARITY
LBL	REF	CRSM	IDRP	REFMA
INE	IFE	IFG	IFL	

PCC ON causes the affected pseudo-operations to be printed. PCC OFF causes the affected pseudo-operations to be suppressed; this is the normal mode at the beginning of the assembly. If the assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

The LBL, TTL, and TTLS operations are not affected if the alter number is less than four (1, 2, or 3).

#### REF ON/OFF - References

1	8	16	
Blanks	REF	ON	(Normal mode)

The REF pseudo-operation controls the assembler in making entries into the symbol reference table and controls the listing of nonreferenced symbols. REF ON (the normal mode) causes the assembler to begin making entries to the symbol reference table. REF OFF causes the assembler to stop making entries to the symbol reference table. If the assembler is already in a specified ON/OFF mode, another request for the same mode is ignored.

The entry LNRSM (list nonreferenced symbols) can also be used as a subfield of the variable field, to list nonreferenced symbols when the assembler is in the REF ON mode. The variable field scan is terminated when either an ON, OFF or RESTORE subfield is encountered. Therefore, these entries should always be last when used in a series of subfields.

#### Examples:

1. REF ON or the absence of a REF pseudo-operation causes a listing of only referenced symbols, and references to those symbols.
2. REF LNRSM, ON or REF LNRSM causes listing of all symbols and references.
3. REF OFF causes listing of all symbols, but no references. (REF LNRSM, OFF has the same effect because the LNRSM entry is only effective when the assembler is in the REF ON mode.)

#### REFMA ON/OFF - Reference Macro Operation

1	8	16	
Blanks	REFMA	OFF	(Normal mode)

The REFMA ON pseudo-operation causes the assembler to create a separate symbol reference table for the macro operations. Each entry of this table consists of a macro name and the alter number(s) at which the name is referenced. If a macro name is present but not referenced, it will not appear in the table.

For a macro name to be referenced, REFMA ON must be specified prior to defining the macro operation. However, since the MAP macro operations are loaded automatically by the assembler before this REFMA pseudo-operation appears, the LODM pseudo-operation must be used to load these macro operations again if it is required to reference them. REFMA OFF causes the assembler to stop referencing macro operations.

Examples:

1. To reference GRTS system macro operations:

```
REFMA  ON
LODM   .GRTM
```

All macro operations under the name .GRTM will be referenced until REFMA OFF is encountered.

2. To reference MAP system macro operations:

```
REFMA  ON
LODM   .GMAC
```

All macro operations under the name .GMAC (MAP Macro operations) will be referenced until REFMA OFF is encountered.

3. To reference program macro operations:

```
REFMA  ON
SPLL  MACRO
      #1      1
      STA     #3-*
      ILQ     0
      ENDM    SPLL
```

The symbolic name of the macro operation (in the location field of the macro identification) must be unique for the program in which the REFMA pseudo-operation is used. The use of this name in the location field at any other instruction, pseudo-operation, or macro operation will result in a multidefined symbol error.

PARITY ON/OFF - ASCII Parity Control

1	8	16	
Blanks	PARITY	OFF	(Normal mode)

The PARITY ON pseudo-operation causes the assembler to generate parity on ASCII characters used in literals, and any used in the Symbolic ASCII pseudo-operation (SACI). PARITY OFF (normal mode) suppresses the parity generation. If the assembler is already in a specified ON/OFF mode, another request for the same mode is ignored.

The entry of ODD or EVEN (EVEN is the normal mode) in a subfield in the variable field specifies the way parity is to be generated when the assembler is in the PARITY ON mode. The variable field scan is terminated when either an ON, OFF, or RESTORE subfield is encountered; therefore, these entries should always be last when used in a series of subfields.

NOTE: The SAVE and RESTORE function applies only to the ON/OFF mode for the pseudo-operation. EVEN and ODD remain in the last mode specified until changed by a different PARITY EVEN/ODD pseudo-operation.

Examples:

PARITY ON causes EVEN parity to be generated for all ASCII literals and SADI subfields.

PARITY ODD,ON causes odd parity to be generated for ASCII characters in literals and SADI subfields.

PARITY EVEN,ON causes even parity to be generated for ASCII characters in literals and SADI subfields.

PARITY OFF or the absence of the PARITY pseudo-operation causes the suppression of parity generation.

PMC ON/OFF - Print Macro Expansion

1	8	16	
<hr/>			
Blanks	PMC	OFF	(Normal mode)

The PMC pseudo-operation causes the assembler to list or suppress all instructions generated by a macro call. PMC ON causes the assembler to print all generated instructions. PMC OFF causes the assembler to suppress all but the macro call. If the assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

PUNCH ON/OFF - Control Card Output

1	8	16	
<hr/>			
Blanks	PUNCH	ON	(Normal mode)

Subject to the DECK/NDECK option of the \$ 355MAP card, the normal mode of the assembler is to punch binary cards for everything it assembles. If PUNCH is used in the operation field with OFF in the variable field, the binary deck will not be punched, beginning at the point the assembler encounters the PUNCH pseudo-operation.

These conventions hold true for both the output binary deck, and the load file counterpart, in the case of assemble and execute activities.

If the assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

## EDITP - Edit Print Lines

<u>1</u>	<u>8</u>	<u>16</u>	
Blanks	EDITP	OFF	(Normal mode)

The EDITP pseudo-operation has a special application. It is used when the program includes the character ? (17) and/or ! (77) punched somewhere on a symbolic card. In normal operation these characters have special meaning to the printer subsystem and may cause character shifting, line suppression, slewing, or buffer overflow. As such, an EDITP ON instruction, causes the output routine to issue printer commands which will treat these as non-special characters. The assembler will then remain in this mode until an EDITP OFF pseudo-operation is encountered.

## CONTROL PSEUDO-OPERATIONS

### EJECT - Restore Output Listing

<u>1</u>	<u>8</u>	<u>16</u>	
Blanks	EJECT	Column 16 must be blank	

The EJECT pseudo-operation causes the assembler to position the printer paper at the top of the next page, to print the title(s), and then print the next line of output on the second line below the title(s).

### REM - Remarks

<u>1</u>	<u>8</u>	<u>16</u>	
Blanks	REM	Remarks and comments can start in	
	or	column 12 or later	
remarks			

The REM pseudo-operation causes the contents of this line of coding to be printed on the assembly listing (just as the comments appear on the coding sheet). However, for readability, columns 8-10 are replaced by blanks before printing.

REM is provided for the convenience of the programmer; it has no other effect upon the assembly.



\* In Column One--Remarks

12            8            16

\*Remarks and comments in columns 2-80

A card containing an asterisk (\*) in column 1 is taken as a remark card. The contents of columns 2-80 are printed on the assembly listing just as they appear on the coding sheet; the asterisk has no other effect on the assembly program.

LBL - Label

1            8            16

Blanks    LBL            X,Y

where:

X=null or up to 8 alphabetic and numeric characters.

Y=null or up to 42 alphabetic and numeric characters.

LBL causes the assembler to serialize the binary cards using columns 73-80, (columns 79 and 80 for full binary cards). The LBL pseudo-operation allows the programmer to specify a left-justified alphabetic label for the identification field and begin serialization with some initial serial number other than zero. The LBL pseudo-operation also allows the programmer to specify up to 42 characters of comments on the \$ OBJECT card of the binary deck. The comment, if present, begins in column 16 of the \$ OBJECT card. The following conditions apply:

1. If the first subfield is null, the assembler discontinues serialization of the binary deck.
2. If the first subfield is not blank, serialization begins with the characters appearing in the first subfield; the characters are left-justified and filled in with terminating zeros up to the position(s) used for the sequence number. Serialization is incremented until the rightmost nonnumeric character is encountered, at which time the sequence recycles to zero.
3. If no LBL pseudo-operation appears in the symbolic deck, the assembler begins serializing with 00000000.
4. If the second subfield is blank, the assembler inserts blanks in the variable field of the \$ OBJECT card.
5. If the second subfield is not blank, the characters in this subfield are inserted on the \$ OBJECT card in columns 16 through column 57.



In the absence of a G or H, the H is assumed. When present, the G or H must be the first subfield.

Subfields other than G or H must be 4-digit numerics defining the years in which the program was copyrighted. No more than four years may be listed within one use of the CPR pseudo-operation.

For example:

```
CPR G, 1969, 1970
CPR 1970, 1971
```

will produce the following copyright notices:

```
Copyright 1969, 1970 by the General Electric Company
Copyright 1970, 1971 by Honeywell Information Systems Inc.
```

#### ABS - Output Absolute Text

```
1      8      16
```

---

```
Blanks  ABS          Column 16 must be blank
```

The ABS pseudo-operation causes the assembler to produce an output of absolute binary text.

The normal mode of the assembler is relocatable; however, if absolute text is required for a given assembly, the ABS pseudo-operation should appear in the deck before any instructions or data. It may be preceded only by listing pseudo-operations. It may, however, appear repeatedly in an assembly interspersed with the FUL pseudo-operation. It should be noted that the pseudo-operations affecting relocation are considered errors in an absolute assembly.

Pseudo-operations that will be in error if used in an absolute assembly are:

```
BLOCK          SYMDEF          SYMREF
```

For a description of the absolute binary card format, see the definition for the FUL pseudo-operation.

#### FUL - Output Full Binary Text

```
1      8      16
```

---

```
Blanks  FUL          Column 16 must be blank
```

The FUL pseudo-operation is used to specify absolute assembly and the FUL format for absolute binary text.

The FUL pseudo-operation has the same effect and restrictions on the assembler as ABS, except for the format of the binary text output. The format of the text is of continuous information with no address identification; that is, the absolute binary cards are punched with program instructions in columns 1-78 (52 words). Such cards can be used in self-loading operations or other environments where control words are not required on the binary card. Sequence numbers are punched in columns 79-80 starting with 00 and continuing to 99 with rollover to 00.

#### TCD - Punch Transfer Card

1	8	16
Blanks	TCD	An expression in the variable field or a symbol

In an absolute assembly, the binary transfer card, produced at the end of the deck as a result of the END card, directs the loading program to cease loading and turn control over to the program at the point specified by the transfer card. Sometimes it is desirable to cause a transfer card to be produced before encountering the end of the deck. This is the purpose of the TCD pseudo-operation. Thus, a binary transfer card is produced, generating a transfer address equivalent to the value of the expression in the variable field.

TCD is an error in the relocatable mode.

#### HEAD - Heading

1	8	16
Blanks	HEAD	From 1 to 7 subfields in the variable field, each containing a single, nonspecial character used as a heading character

In programming, it is sometimes desirable to combine two programs, or sections of the same program, that use the same symbols for different purposes. The HEAD pseudo-operation makes such a combination possible by prefixing each symbol of five or fewer characters with a heading character. This character must not be one of the special characters; that is, it must be one of the characters A-Z, 0-9, or the period (.). Using different heading characters, in different program sections later to be combined for assembly, removes any ambiguity as to the definition of a given symbol.

The effect of the HEAD pseudo-operation is to cause every symbol of five or less characters, appearing in either the location field or the variable field, to be prefixed by the current HEAD character. The current HEAD character applies to all symbols appearing after the current HEAD pseudo-operation and before the next HEAD or END pseudo-operation.

Deheading is accomplished by a zero or blanks in the variable field. To understand more thoroughly the operation of the heading function, it is necessary to know that the assembler internally creates a six-character symbol by right-justifying the characters of the symbol and filling in leading zeros. Thus, if the assembler is within a headed program section and encounters a symbol of five or fewer characters, it inserts the current HEAD character into the high-order, leftmost character position of the symbol. Each symbol, with its inserted HEAD character, then can be placed in the assembler Symbol Table as unique entries and assigned their respective location values.

It is also possible to head a program section with more than one character. This is done by using the pseudo-operation HEAD in the operation field with from two to seven heading characters in the variable field, separated by commas. The effect of a multiple heading is to define each symbol of that section once for each heading character. Thus, for example, if the symbols SHEAR, SPEED, and PRESS are headed by

HEAD	X,Y,Z
------	-------

nine unique symbols

XSHEAR	XSPEED	XPRESS
YSHEAR	YSPEED	YPRESS
ZSHEAR	ZSPEED	ZPRESS

are generated and placed in the Assembler Symbol Table. This allows regions by HEADX, HEADY, or HEADZ to obtain identical values for the symbols SHEAR, SPEED, and PRESS.

Cross-referencing among differently headed sections can be accomplished by the use of six-character symbols or by the use of the dollar sign (\$). Six-character character symbols are immune to HEAD; therefore, they provide a convenient method of cross-referencing among differently headed regions.

When a symbol within a headed section is also to be a SYMDEF symbol, it must be a six-character symbol (immune to HEAD).

To allow the programmer more flexibility in cross-referencing, the MAP assembler language includes the use of the dollar sign (\$) to denote references to an alien-headed region.

If the programmer wishes to reference a symbol of less than six characters in another program section, he merely prefixes the symbol by the HEAD character for that respective section, separating the HEAD character from the body of the symbol by a dollar sign (\$).

To reference from a headed region into a region that is not headed (zero heading), the programmer can use either the heading character zero and the dollar sign (0\$) preceding the symbol; or, if the symbol is the initial value of the variable field, then the appearance of only the leading dollar sign will cause the zero heading to be attached to the symbol.

Example:

START	LDA	A-*	Initial instruction (no heading)
	---		
	---		
	TRA	B\$SUM-*	Transfer to new headed section
A	BSS	1	} Section headed B
	HEAD	B	
SUM	LDA	\$A-*	
	---		
	---		
	TRA	0\$START+2-*	
	END		

The LDA \$A-\* could have been written as LDA 0\$A-\*, as they both mean the same.

DCARD - Punch BCD Card

1            8            16

---

Blanks DCARD        N,M Two subfields in the variable field

The first subfield (N) contains a decimal integer (limited only by the size of available memory), and the second subfield (M) contains a single BCD character used as a decimal data identifier. The assembler punches the next N cards after the DCARD pseudo-operation with the specified BCD identifier in column one of each of these N cards and with the BCD information taken from the corresponding source cards on a one-for-one basis.

There are no restrictions on the BCD information that can be placed in columns 2-72 of the source cards. (One of the significant uses of DCARD is to generate 355SIM control cards.)

The DCARD has the further effect of suppressing the normal automatic generation of a \$ OBJECT and \$ DKEND card.

END - End of Assembly

1            8            16

---

Blanks END        Blanks or an expression in the variable field  
or a  
symbol

The END pseudo-operation signals the assembler that it has reached the end of the symbolic input deck; it must be present as the last physical card encountered by the assembler.

If a symbol appears in the location field, it is assigned the next available location.

In a relocatable assembly, the variable field must be blank; in an absolute assembly, the variable may contain an expression. In relocatable decks, the starting location of the program will be an entry location and the location specified is given to the loader by a special control card used with the loader. Absolute programs require a binary transfer card which is generated by the END pseudo-operation. The transfer address is obtained from the expression in the variable field of the END card.

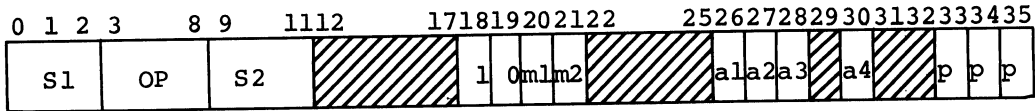
### OPD - Operation Definition

1	8	16
New Oper- ation code	OPD	One or more subfields, separated by commas, in the variable field. The subfields define the bit configuration of the new operation code

The OPD pseudo-operation can be used to define or redefine machine instructions to the assembler. This allows programmers to add operation codes to the Assembler Table of Operation Codes during the assembly process. This is extremely useful and powerful in defining new instructions or special bit configurations.

The variable field subfields are bit-oriented and have the same general form as described under the VFD pseudo-operation. In addition, the variable field considered in its entirety, requires the use of either of two specific 36-bit formats for defining the operation.

1. The normal instruction format
2. The input/output operation format



where:

S1 = Suboperation field 1 (MBZ) for Memory Reference Instructions)

OP = Primary operation field (bits 3 through 8 of instruction)

S2 = Suboperation field 2 (MBZ for Memory and Nonmemory, Group 1, Reference Instructions)

m = Modifier tag type (0=allowed; 1=not allowed)

m1 : register modification

m2 : indirect addressing

a = Address field conditions (0=not required; 1=required)

a1 : address required/not required

a2 : address required even

a3 : address must be absolute

a4 : character field allowed/not 1/0)

p = Octal assembly listing format (x represents one octal digit)

000 : x xx xxx Memory Reference Instructions

001 : xxx xxx Nonmemory, Group 1 Instructions

010 : xxxx xx Nonmemory, Group 2 Instruction

011 : x xxxxx IND and ZERO pseudo-operations

100 : xxxxxx Data generating pseudo-operations

101 : x x xxxx IND pseudo-operations with character sub-field

To illustrate the use of OPD, assume one wished to define the current machine instruction, Load A (LDA). Using the preceding format and the octal notation (as described under the VFD pseudo-operation), can be coded as:

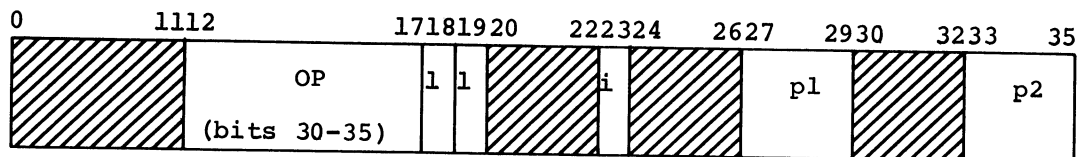
	<u>1</u>	<u>8</u>	<u>16</u>
or	LDA	OPD	03/0,06/07,03/0,6/,02/2,6/,03/5,1/,01/1,2/,03/0
or	LDA	OPD	012/0070,6/,02/2,6/,03/5,02/1,5/
	LDA	OPD	018/007000,018/401240

or in other forms, providing the bit positions of the instruction-defining format are individually specified to the assembler.

NOTE: "036" is illegal for OPD since eighteen bit octal fields are the maximum lengths possible.



The input/output operation defining format and subfields for types 0 and 1 is as follows:



where:

OP = New operation code for bit positions 30-35

i = Type of input/output command

0 : OP            000000

                  0000kk

1 : OP da        000000

                  00dakk

p1 = Listing format of first word (bits 0-17). (See preceding instruction format for description)

p2 = Listing format of second word (bits 18-35). (See preceding instruction format for description)

As an example of the use of OPD to generate an input/output command (using the above format for the variable field and defining the bits according to the rules for VFD), assume one wanted to generate the exact command, Write Tape Binary (WTB). This could be written as

1	8	16
WTB	OPD	12/,06/15,02/3,04/1,06/4,06/2

or in various other bit oriented forms.

OPSYN - Operation Synonym

1	8	16
A sym- bol or opera- tion code	OPSYN	A mnemonic operation code in the variable field

The OPSYN pseudo-operation is used for equating either a newly defined symbol or a presently defined operation to some operation code already in the operation table of the assembler. The operation code may have been defined by a prior OPD or OPSYN pseudo-operation; in any case, it must be in the assembler operation table. The new symbol to be defined is entered in the location field and the operation code that must be in the assembler operation table is entered in the variable field. The new symbol must be defined (entered into the operation table) by the OPSYN pseudo-operation code before it is used as an operation code.

## LOCATION COUNTER PSEUDO-OPERATION

### USE - Use Multiple Location Counters

1            8            16

---

Blanks    USE            A single symbol, blanks or the word PREVIOUS in  
the variable field

The assembler provides the ability to employ multiple location counters via the USE pseudo-operations. This pseudo-operation causes the assembler to place succeeding instructions under control of the location counter represented by the symbol in the variable field. Each location counter begins with the value of zero, and its size is determined as being the highest value assumed by it (that is, occupied by some instruction assembled under it). This is not always the last instruction under the USE, as an ORG may have occurred within it. At the completion of the first pass through the symbolic program, the length of each USE will be a known value, and the order of memory allocation will be implied by the order of first presentation to the assembler. Thus, the origin of each location counter may be computed based on the origin and size of the one preceding it. There is an assumed location counter, called the blank USE, implied in all assemblies, which has a natural origin of zero.

Automatic determination of a counter origin may be overridden with the BEGIN pseudo-operation. In this case, the chain of location counters will be made, completely ignoring those counters which have an associated BEGIN. In more general terms, then, the origin of a non-begin location counter is taken as one more than the highest value taken by the next prior non-begin counter. The first of these non-begin counters has an origin of zero, by definition. The location counter in control at the time that a USE is encountered is suspended at its current value and preserved as the PREVIOUS counter. It may be called back into operation at any later point in the program without confusion as to its current state, and will begin counting at the address one higher than the last location counted.

If the word PREVIOUS appears in the variable field, the assembler reactivates the location counter just before the present one. It is not possible to go back more than one level via the USE PREVIOUS command.

### BEGIN - Origin of a Location Counter

1            8            16

---

Blanks    BEGIN            Two subfields in the variable field

The BEGIN pseudo-operation is used to arbitrarily specify the origin of a given location counter. As such, it will not be tied into the chain of location counters as described in USE. Its origin, however, may be an expression involving some symbol, or symbols, defined under another location counter, in which case it will be linked to the chain at the specified point. The user must beware of overlaying code with this pseudo-operation.

The location counter symbol is specified in the first subfield and is given the value specified by the expression found in the second subfield. Any symbol appearing in the second subfield must be previously defined and must appear under one location counter. The BEGIN pseudo-operation may appear anywhere in the deck; it does not invoke the counter, a USE pseudo-operation must be given to bring a location counter into effect.

#### ORG - Origin Set by Programmer

1	8	16
Blanks	ORG	An expression in the variable field or a symbol

The ORG pseudo-operation is used to change the next value of a counter, normally assigned by the assembler, to a desired value. If ORG is not used, the counter is initially set to zero.

All symbols appearing in the variable field must be previously defined. If a symbol appears in the location field, it is assigned the value of the variable field. If the result of the evaluation of a variable field expression is absolute, the instruction counter is reset to the specified value, relative to the current location counter. If an expression result is relocatable, the current location counter is suspended and the counter to which the expression is relocated will be invoked with the value given by the expression.

#### LOC - Location of Output Text

1	8	16
Blanks	LOC	An expression in the variable field

The LOC pseudo-operation functions identically to the ORG pseudo-operation, with one exception; it has no effect on the loading address when the assembler is punching binary text. That is, the value of the location counter is changed to that given by the variable field expression, but the loading will continue to be consecutive. This provides a means of assembling code in one area of memory, although execution will occur at some other area of memory.

All symbols appearing in the variable field of this pseudo-operation must be previously defined.

The sole purpose of the LOC pseudo-operation is to allow program coding to be loaded in one area of memory and then to be subsequently moved to another area for execution.

## SYMBOL-DEFINING PSEUDO-OPERATIONS

The flexibility in program writing can be augmented by defining symbols to the assembler with pseudo-operations. This symbol definition capability is used for (1) equating symbols, or (2) defining parameters used frequently by the program that are subject to change.

It should be noted that they do not generate any machine instructions or data, but are available merely for the convenience of the programmer.

### EQU - Equal To

1            8            16

---

Symbol EQU            An expression in the variable field

The EQU pseudo-operation defines the symbol in the location field to have the value of the expression appearing in the variable field. The symbol in the location field assumes the same mode as that of the expression in the variable field; absolute or relocatable. (See "Relocatable and Absolute Expressions".)

All symbols appearing in the variable field must be previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot appear in the variable field.

If the asterisk (\*) appears in the variable field denoting the current location counter value, it is given the value of the next sequential location not yet assigned by the assembler; this with respect to the unique location counter presently in effect.

### FEQU - Special FORTRAN Equivalence

1            8            16

---

Symbol FEQU            A symbol in the variable field

The FEQU pseudo-operation equates the symbol in the location field with the symbol in the variable field; the latter yet undefined.

The restrictions for this pseudo-operation are: (1) the variable field may not contain an expression, (2) the symbol in the variable field may not subsequently appear in either field of another FEQU pseudo-operation, and (3) if HEAD characters are in effect, both symbols (or neither symbol) must be able to be headed.

As implemented, both symbols are essentially held in abeyance until the variable field symbol is defined. At that point, both symbols take on the same value and characteristics, and are available for normal functions.

It should be noted that the symbol in the variable field does not have to be undefined. Nor does it have to be a symbol. It may be a number, or the current location counter value symbol (\*). However, in these cases, FEQU acts just as EQU, and the location symbol is immediately defined with the indicated value.

#### BOOL - Boolean

1	8	16
<hr/>		
Symbol	BOOL	A Boolean expression in the variable field

The BOOL pseudo-operation defines a constant of 18 bits and is similar to EQU except that the evaluation of the expression in the variable field assumes Boolean operators. By definition, all integral values are assumed to be octal; in error otherwise. The symbol in the location field will always be absolute, and the presence of any expression other than absolute in the variable field will be considered an error. (See "Relocatable and Absolute Expressions".)

All symbols appearing in the variable field must be previously defined.

#### SET - Symbol Redefinition

1	8	16
<hr/>		
Symbol	SET	An expression in the variable field

The SET pseudo-operation permits the redefinition of a symbol previously defined to the assembler. This feature is useful in macro expansions where it may be undesirable to use created symbols (CRSM).

All symbols entered in the variable field must be previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot be used in the variable field.

The symbol in the location field is given the value of the expression in the variable field. The SET pseudo-operation may not be used to define or redefine a relocatable symbol. (See "Relocatable and Absolute Expressions".)

When the symbol occurring in the location field is defined by other than a previous SET, the current SET pseudo-operation is ignored and flagged as an error.

The last value assigned to a symbol by SET affects only subsequent inline coding instructions using the redefined symbol.

## MIN - Minimum

1	8	16
Symbol	MIN	A sequence of expression, separated by commas in the variable field--all of the same type; that is, a relocatable <u>or</u> absolute

The MIN pseudo-operation defines the symbol in the location field as having the minimum value among the various values of all relocatable or all absolute expressions contained in the variable field.

All symbols appearing in the variable field must be previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot be used in the variable field.

## MAX - Maximum

The MAX pseudo-operation is coded in the same format as MIN above. It defines the symbol in the location field as having the maximum value of the various expressions contained in the variable field.

All symbols appearing in the variable field must be previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot be used in the variable field.

## SYMDEF - Symbol Definition

1	8	16
Blanks	SYMDEF	Symbols separated by commas in the variable field

The SYMDEF pseudo-operation is used to identify symbols which appear in the location field of a subprogram when these symbols are referred to from outside the subprogram (by SYMREF). Also, the programmer must provide a unique SYMDEF for use by the Loader to denote each subprogram entry point for the loading operations. The symbols used in the variable field of a SYMDEF entry are called SYMDEF symbols. Multiple defined SYMDEF symbols cannot occur since the assembler ignores the current definition if it finds the same symbol previously entered in the SYMDEF table.

The appearance of a symbol in the variable field of a SYMDEF entry indicates that:

1. The symbol must appear in the location field of only one of the instructions within the subroutine in which SYMDEF occurs.
2. The assembler places each SYMDEF symbol and its relative address in the preface card.
3. At load time, the loader builds a table of SYMDEF symbols to be used for linkage with SYMREF symbols.

It is possible to classify SYMDEF symbols as primary and secondary. A secondary SYMDEF symbol is denoted by a minus sign in front of the symbol. The Loader will provide linkage for a secondary SYMDEF symbol only after linkage is made to a primary SYMDEF within the same subprogram. Secondary SYMDEF symbols are appropriate when using the system subroutine library and generating routines for accessing the library. Secondary SYMDEF symbols are normally defined as secondary entries to subroutines within a subprogram library. The use of primary and secondary SYMDEF symbols is also described in the DATANET 355/6600 Relocatable Loader manual.

SYMREF - Symbol Reference

1	8	16
Blanks	SYMREF	A sequence of symbols separated by commas entered in the variable field

The SYMREF pseudo-operation is used to denote symbols that are used in the variable field of a subprogram, but are defined in a location field external to the subprogram. Symbols used in the variable field of a SYMREF entry are called SYMREF symbols.

When a symbol appears in the variable field of a SYMREF entry, the following items apply:

1. The symbol should occur in the variable field of at least one pseudo-operation (CALL, IND, ZERO, VFD, DCW, ICW), within the subroutine.
2. At assembly time, the assembler enters the SYMREF symbol in the preface card of the assembled deck and places a special entry number in the variable fields of all pseudo-operations (in the referenced subroutine) that contain the symbol.
3. At loading, the Loader associates the SYMREF symbol with a corresponding SYMDEF symbol and places the appropriate address in all pseudo-operations that have been given the special entry number.

Symbols appearing in the variable field of a SYMDEF instruction must not appear in the location field of any entry within the subroutine in which SYMREF is used.

Example:

<u>Base Program or Subprogram</u>			<u>Referencing Subprogram</u>		
	SYMREF	PROG		SYMREF	PROG
PROG	SAVE	1,2		.	
	LDA	TEMP-*		TSY	PRG-*,I
	.			.	
	.			.	
	RETURN	PROG	PRG	IND	PROG
TEMP	BSS	1			
	.				
	.				

NULL - Null

1	8	16
<hr/>		
Symbol	NULL	The variable field is not interpreted

The NULL pseudo-operation acts as an NOP machine instruction during assembly. No actual words are assembled. A symbol on a NULL operation is defined as the current value of the location counter.

EVEN - Force Location Counter Even

1	8	16
<hr/>		
Symbol	EVEN	The variable field is not interpreted
	or	
	blanks	

The EVEN pseudo-operation effects the same result as the E in column 7. If the location counter is odd, an NOP operation is generated, thereby making it even. If there is a symbol in the location field, it is defined at the even address.

ODD - Force Location Counter Odd

1	8	16
<hr/>		
Symbol	ODD	The variable field is not interpreted
	or	
	blanks	

The ODD pseudo-operation has the same effect as an O in column 7. If the location counter is even, an NOP operation is generated, thereby making it odd. If there is a symbol in the location field, it is defined at the odd address.

EIGHT - Force Location Counter to a Multiple of 8

1	8	16
<hr/>		
Symbol	EIGHT	The variable field is not interpreted
	or	
	blanks	

The EIGHT pseudo-operation has the same effect as an 8 in column 7. If the location counter is not a multiple of 8, a TRA n is generated, where the value of n is the number of locations skipped, and the location counter is bumped by n. If there is a symbol in the location field, it is defined at the mod-8 address.



## BASE - Force Location Counter to a Multiple Power of 2

1	8	16
Symbol	BASE	One subfield in the variable field
or		
blanks		

The BASE pseudo-operation is used to force the location counter to a multiple of a power of two. If the location counter is not a multiple of this number, a TRA n is generated, where the value of n is the number of locations to the next location that is a multiple of the base selected. If there is a symbol in the location field, it is defined at the appropriate address.

The subfield contains one of the following values; 8, 16, 32, 64, 128, 256. These are the only legitimate values for the variable field. All others are ignored.

### Example:

Assume the location ALPHA must start at a multiple of sixty-four. Force this condition as follows:

ALPHA BASE 64

With the location counter at 1123 (octal) this would print out (without column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
01123	071055	0
01200	ALPHA BASE	64

NOTE: In each of the four pseudo-operations, (EVEN, ODD, EIGHT, and BASE) the origin of the location counter is also forced to a related address. For EVEN and ODD, it is forced even; for EIGHT, it is forced to a multiple of eight; and for BASE, it is forced to a multiple of a power of two.

### DATA GENERATING PSEUDO-OPERATIONS

The assembler provides eight pseudo-operations which can be used to generate data in the program at the time of assembly. These are BCI, OCT, DEC, VFD, ASCII, ASCIIC, ACI, ACIC and SACI. All are word oriented except VFD which is bit oriented. The pseudo-operation DUP does not generate data, but through its repeat capability, causes symbolic instructions and pseudo-operations to be iterated.

OCTAL - Octal

	<u>1</u>	<u>8</u>	<u>16</u>
Symbol	OCT	One or more subfields separated by commas appearing in the variable field; each one containing a signed or unsigned octal integer	
or			
blanks			

The OCT pseudo-operation is used to program octal integer data into an assembled program. The OCT pseudo-operation causes the assembler to generate n locations of OCT data where the variable field contains n subfields (n-1 commas). Consecutive commas in the variable field cause the generation of a zero data word, as does a comma followed by a terminal blank. Up to six octal digits plus the leading sign may make up the octal number.

The OCT configuration is considered true and is not complemented on negatively signed numbers. The sign applies only to bit 0. All assembly program numbers are right-justified, retaining the integer form.

Example:

OCT 1,-4,7701,+3,, -77731,04

If the current location counter is set at 506, the instruction above is printed out as follows (less the column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
00506	000001	0	OCT 1,-4,7701,+3,,
00507	400004	0	-77731,04
00510	007701	0	
00511	000003	0	
00512	000000	0	
00513	477731	0	
00514	000004	0	

DEC - Decimal

	<u>1</u>	<u>8</u>	<u>16</u>
Symbol	DEC	One or more subfields in the variable field, separated by commas, each containing a decimal entry	
or			
blanks			

The assembler allows four types of decimal information that can be specified for conversion to binary data. The various types are uniquely defined by the syntax of the individual subfields of the DEC pseudo-operation. The basic types are single-precision, fixed-point numbers; single-precision, floating-point numbers; double-precision, fixed-point numbers; and double-precision, floating-point numbers. All fixed-point numbers are right-justified in the assembled binary words; floating-point numbers are left-justified to bit position eight with the binary point between positions 0 and 1 of the mantissa.

Example (Single Precision):

GAMMA      DEC      3,-1,6.,.2E1,1B9,1.2E1B14,-4

This instruction prints out the following data words (without column headings), assuming that GAMMA is located at 1041.

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
01041	000003	0	GAMMA DEC 3,-1,6.,.2E1,
01042	777777	0	1B9,1.2E1B14,
01043	006600	0	-4
01044	004400	0	
01045	000400	0	
01046	000140	0	
01047	777774	0	

The presence of the decimal point and or the E scale factor implies floating-point, while the added B (binary scale) implies fixed-point binary numbers. The absence of all of these elements implies integers.

DEC            -1B9,-1.,1000

With the location counter at 1050, the subfields above generate:

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
01050	777400	0	DEC -1B9,-1.,1000
01051	001000	0	
01052	001750	0	

Example (Double Precision):

BETA      DEC      .3D0,0.D0,1.2D1B68,1D-1

The location counter is at the address BETA (1060); the subfields above generate the following double words:

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
01060	776463	0	BETA DEC .3D0,0.D0,
01061	146314	0	1.2D1B68,1D-1
01062	400000	0	
01063	000000	0	
01064	000000	0	
01065	000140	0	
01066	772631	0	
01067	463146	0	

## BCI - Binary Coded Decimal Information

1	8	16
Symbol	BCI	Two subfields in the variable field: a count
or		subfield and a data subfield
blanks		

The BCI pseudo-operation is used to enter binary-coded decimal (BCD) character information into a program.

The first subfield is numeric and contains a count that determines the length of the data subfield. The count specifies the number of 3-character machine words to be generated; thus, if the count field contains n, the data subfield contains 3n characters of data. The maximum value for n is 18. The minimum value for n is 1.

The second subfield contains the BCD characters, three per machine word.

### Example:

BETA BCI 6,NO ERROR CONDITION

Again, assume the location counter set at 506 (location of BETA); the above instruction prints out (less column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
00506	454620	0	BETA BCI 6,NO ERROR
00507	255151	0	CONDITION
00510	465120	0	
00511	234645	0	
00512	243163	0	
00513	314645	0	

## VFD - Variable Field Definition

1	8	16
Symbol	VFD	One or more subfields in the variable field
or		separated by commas
blanks		

The VFD pseudo-operation is used to define the data word in terms of individual bits.

In considering the definition of a VFD subfield, it is understood that the unit of information is a single bit (in contrast with the unit of information in the BCI pseudo-operation which is six bits). Each VFD subfield is one of four types: an algebraic expression, a Boolean expression, alphanumeric (H or R), or ASCII (A). Each subfield contains a conversion-type indicator and a bit count; the maximum value is 18 bits. The bit count is an unsigned integer which defines the length of the subfield; it is separated from the data subfield by a slash (/). If the bit count is immediately preceded by an O, H, R, or A, the variable-length data subfield is either Boolean, alphanumeric, or ASCII. In the absence of any type indicators, the data subfield is algebraic. A Boolean subfield contains an expression that is evaluated using the Boolean operators (\*,/,+,-).

R is an alphanumeric indicator which specifies right adjustment of the argument. Unused bit positions are zero filled. R is used only in a VFD pseudo-operation.

The contents of the ASCII subfields can be any ASCII symbol listed in Figure 4-1, or any three digit octal number in the range 000 through 777. (See the "SACI pseudo-operation" in this section.)

The data subfield is evaluated according to its form: algebraic, Boolean, alphanumeric, or ASCII. An 18-bit field results. The low-order n bits of the algebraic or Boolean expression determines the resultant field value; whereas for the alphanumeric subfield the high-order n bits are used for H, and low-order n bits are used for R. For an ASCII subfield the high-order bits are used.

If the required subfields cannot be contained on one card, they must be continued by the use of the ETC pseudo-operation. This is done by terminating the variable field of the VFD pseudo-operation with a comma. The next subfield is then given as the beginning expression in the variable field of an ETC card. If necessary, subsequent subfields may be continued onto following ETC cards in the same manner. Except for the H type alphanumeric, the scanning of the variable field is terminated upon encountering the first blank character.

The VFD may generate more than one machine word; if the sum of the bit counts is not a multiple of a discrete machine word, the last partial string of bits is left-justified and the word completed with zeros.

Examples:

1. Assume the address ALPHA is packed in the first word, decimal 3 in the next six bits, the literal letter B in the next six bits, and an octal 77 in the last six bits of the second word, as follows:

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
01053	000731	1 VFD 18/ALPHA,6/3,H6/B,
01054	032277	0 06/77

The total number of bits under a VFD need not be a multiple of a full word, nor is the total field (sum of all subfields) restricted to one word. The total field width, however, for a single subfield is 18 bits.

2. Consider a program to generate a six-word identifier for a table. Assume  $n$  is the word length of the table and is equal to 12. You wish to place twice the length of the table in the first 12 bits, the name of the table in the next 60 bits, the location of the table (where TABLE is a program relocatable symbol equal to 2351 (octal)) in the next 18 bits, zero in the next eight bits, and -1 in the next six bits--all in a six-word key.

With the location counter at 1054.

```
VFD      12/2*12,H18/PRE,H18/SSU,H18/RE,H6/,18/TABLE,
          8/,6/-1
```

will generate

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
01054	003047	0	VFD 12/2*12,H18/PRE,H18/
01055	512562	0	SSU,H18/RE,H6/,18/
01056	626451	0	TABLE,8/,6/-1
01057	252020	0	
01060	002351	1	
01061	001760	0	

where 1 specifies the relocatability of TABLE.

With the location counter at 5221,

```
VFD      A9/A,A9/252,A18/CR,LNF,A12/A,B,O6/77
```

will generate

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
05521	301252	0	VFD A9/A,A9/252,A18/CR,LNF,
05522	015212	0	A12/A,B,O6/77
05523	301377	0	

assuming a PARITY pseudo-operation appeared before this statement and indicated ODD parity.

#### ASCII, ASCIIC, ACI, ACIC - ASCII Coded Information

	<u>1</u>	<u>8</u>	<u>16</u>
Symbol	ASCII	One or more subfields separated by commas	
or	ASCIIC		
blank	ACI		
	ACIC		

The ASCII and ASCIIC pseudo-operations are used to enter lower case ASCII character information.

The ACI and ACIC pseudo-operations are used to enter upper case ASCII character information.

In the ASCII and ACI pseudo-operations, the first subfield is numeric and specifies the number of two-character machine words to be generated. If the count is n, the data field contains 2n characters. The maximum value for n is 28, the minimum is one.

In the ASCIIC and ACIC pseudo operation, the first subfield specifies the number of two-character machine words (as in ASCII and ACI). A second subfield contains a decimal number that specifies the number of characters to be assembled. This number is converted to binary and is inserted in the first half of the first assembled word.

NOTE: This number is not examined by the assembler. It is converted and stored only.

The second subfield in ASCII, ACI and the third subfield of ASCIIC, ACIC contain the ASCII characters, two per machine word.

Example:

```
BETA  ASCII 4, NO ERROR
      ASCIIC 5,8, NO ERROR
```

Assume the location counter is set at 506 (location of BETA); the instructions above would print out (less column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
00506	156157	0	BETA ASCII 4,NO ERROR
00507	040145	0	
00510	162162	0	
00511	157162	0	
00512	010156	0	ASCIIC 5,8,NO ERROR
00513	157040	0	
00514	145162	0	
00515	162157	0	
00516	162040	0	

SACI - Symbolic ASCII Information

<u>1</u>	<u>8</u>	<u>16</u>
Symbol	SACI	One or more subfields separated by commas
or		
blank		

The SACI pseudo-operation provides the ability to assemble all of the ASCII character set using the character itself, a descriptive symbol, or any three-digit octal number. In addition, parity can be generated for each ASCII character symbol processed, if the PARITY pseudo-operation is used. Parity is not generated for the octal representation of subfields used in place of the symbols.

Each subfield should contain one of the characters or symbols listed in Figure 4-1, or any octal subfields in the range 000 through 777. A symbol must be used in place of an actual comma (,) or blank ( ) since these characters are used as field delimiters when scanning the variable field.

NOTE: Each octal subfield must contain exactly three characters.

Two SACI subfields are stored for each word generated. If an odd number of subfields is specified, only the upper half of the last word is used: the lower half contains zeros.

Example:

```
MESG SACI SYN,SYN,SYN,SYN,SOH,H,B, ,B, ,STX,ETX
MESG SACI CR,LNF,6,0,0,SP,D,I,S,C,O,N,N,E,C,T,S
```

Assuming the PARITY pseudo-op has not appeared, or is set to OFF, and the location counter is set at 2103 (location of MESG); the instructions above would print out as follows (less column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>			
02103	026026	0	MESG	SACI	SYN,SYN,SYN,SYN,
02104	026026	0			SOH,H,B, ,B, ,
02105	001110	0			STX,ETX
02106	102100	0			
02107	102100	0			
02110	002003	0			
02111	015012	0	MESS	SACI	CR,LNF,6,0,0,SP,
02112	066060	0			D,I,S,C,O,N,N,E,
02113	060040	0			C,T,S
02114	104111	0			
02115	123103	0			
02116	117116	0			
02117	116105	0			
02120	103124	0			
02121	123000	0			

Assuming the PARITY pseudo-operation appeared and is set to EVEN,ON and the location counter is set at 2103 (location of MESG); the above would print out as follows (less column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>			
02103	226226	0	MESG	SACI	SYN,SYN,SYN,SYN,
02104	226226	0			SOH,H,B, ,B, ,
02105	201110	0			STX,ETX
02106	102300	0			
02107	102300	0			
02110	202003	0			
02111	215012	0	MESS	SACI	CR,LNF,6,0,0,SP,
02112	066060	0			D,I,S,C,O,N,N,E,
02113	060240	0			C,T,S
02114	104311	0			
02115	123303	0			
02116	317116	0			
02117	116305	0			
02120	303324	0			
02121	123000	0			



The following is an example of the octal digit subfield in the SACI pseudo-operation:

```
MESG SACI A,B,C,101,102,103,777
```

Assuming the PARITY pseudo-operation has appeared and is set to ODD,ON and the location counter is set at 4271 (location of MESG); the above would print out as follows (less column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>					
04271	301302	0	MESG	SACI	A,B,C,101,102,103,777		
04272	103101	0					
04273	102103	0					
04274	777000	0					

<u>Symbol</u>	<u>Value</u>	<u>Symbol</u>	<u>Value</u>	<u>Symbol</u>	<u>Value</u>	<u>Symbol</u>	<u>Value</u>
NULL	000	SP	040		100	GRA	140
SOH	001	!	041	A	101	a	141
STX	002	"	042	B	102	b	142
ETX	003	#	043	C	103	c	143
EOT	004	\$	044	D	104	d	144
ENQ	005	%	045	E	105	e	145
ACK	006	&	046	F	106	f	146
BELL	007	'	047	G	107	g	147
BSP	010	(	050	H	110	h	150
HT	011	)	051	I	111	i	151
LNF	012	*	052	J	112	j	152
VT	013	+	053	K	113	k	153
FF	014	,	054	L	114	l	154
CR	015	-	055	M	115	m	155
SO	016	.	056	N	116	n	156
SI	017	/	057	O	117	o	157
DLE	020	0	060	P	120	p	160
DC1	021	1	061	Q	121	q	161
DC2	022	2	062	R	122	r	162
DC3	023	3	063	S	123	s	163
DC4	024	4	064	T	124	t	164
NAK	025	5	065	U	125	u	165
SYN	026	6	066	V	126	v	166
ETB	027	7	067	W	127	w	167
CAN	030	8	070	X	130	x	170
EM	031	9	071	Y	131	y	171
SUB	032	:	072	Z	132	z	172
ESC	033	;	073	[	133	LBR	173
FS	034	<	074	\	134	VTL	174
GS	035	=	075	]	135	RBR	175
RS	036	>	076	^	136	TILDE	176
US	037	?	077	_	137	DEL	177

Figure 4-1. Symbolic ASCII Symbols

## DUP - Duplicate Cards

1	8	16
Symbol	DUP	Two subfields in the variable field separated
or		by a comma
blanks		

The DUP pseudo-operation provides an easy method for generating tables and or data. It causes the assembler to duplicate a sequence (range) of instructions, or pseudo-operations, a specified number of times.

The first subfield in the variable field is an absolute expression which defines the count. The value of the count field specifies the number of cards following the DUP pseudo-operation that are included in the group to be duplicated. The value in the count field must be a decimal integer less than, or equal to, ten.

The second subfield is an absolute expression which specifies the number of iterations. The value in the iteration field specifies the number of times the group of cards, following the DUP pseudo-operation, is to be duplicated. This value can be any positive integer less than  $2^{18}$  minus 1. The groups of duplicated cards appear in the assembled listing immediately behind the original group.

If either the count field or the iteration field contains zero or is null, the DUP pseudo-operation is ignored.

If a symbol appears in the location field of the pseudo-operation, it is given the address of the next location to be assigned by the assembler.

If an odd or even address is specified for an instruction within the range of a DUP pseudo-operation, the instruction is placed in an odd or even address and a filler used when needed. The filler is an NOP instruction.

All symbols appearing in the variable field of the DUP pseudo-operation must be previously defined. Any symbols appearing in the location field of the instructions being duplicated are defined only on the first iteration, thus avoiding multiple-defined symbols. The SET pseudo-operation would be the exception to this rule.

The only instructions or pseudo-operations which may not appear in the range of a DUP operation are END, MACRO, and DUP. An ETC pseudo-operation can not appear as the first card after the range of a DUP operation.

## MEMORY ALLOCATION PSEUDO-OPERATIONS

These pseudo-operations are used to reserve specified memory areas for use as storage areas or work areas.

### BSS - Block Started by Symbol

1	8	16
Symbol	BSS	A permissible expression in the variable field
or		defines the amount of storage to be reserved
blanks		

The BSS pseudo-operation is used by the programmer to reserve an area of memory for working or data storage. The variable field contains an expression that specifies the number of locations the assembler must reserve in the program.

If a symbol is entered in the location field, it is assigned the value of the first location in the block of reserved storage. If the expression in the variable field contains symbols, they must be previously defined and must yield an absolute result. No binary cards are generated by this pseudo-operation.

### BFS - Block Followed by Symbol

1	8	16
Symbol	BFS	A permissible expression in the variable field
or		defines the amount of storage to be reserved
blanks		

The BFS pseudo-operation is identical to BSS, with one exception. If a symbol appears in the location field, it is assigned the value of the first location after the block of reserved storage has been assigned.

### BLOCK - Block Common

1	8	16
Symbol	BLOCK	A symbol in the variable field

The purpose of the BLOCK pseudo-operation is to specify that program data which follows the block entry is to be assembled in the LABELED COMMON region of the user program under the symbol appearing in the variable field. BLOCK, in effect, is another location counter external to the text of the program.

The symbol in the variable field specifies the label of the COMMON area to be assembled. If the variable field is left blank, the normal FORTRAN BLANK COMMON is specified, and data following the BLOCK pseudo-operation is assembled relative to the unlabeled (BLANK COMMON) memory area of the user program. It is not possible to assemble data or instructions into BLANK COMMON. Storage labeling and reservation only is permitted.

The pseudo-operations which take the program out of BLOCK mode and into some other mode are:

BLOCK (for some other LABELLED COMMON)

USE

ORG/LOC, where the value of the expression is relocatable

END

It should be noted that BLOCK does not cause the assembler to make the current USE location counter PREVIOUS. As such, a USE PREVIOUS following a BLOCK will cause the location counter in effect prior to the last USE, to be invoked. A maximum of 63 labeled COMMON areas are permitted in a program.

### LIT - Literal Pool Origin

1	8	16
Symbol	LIT	Column 16 must be blank
or		
blanks		

The LIT pseudo-operation causes the assembler to punch and print all previously developed literals. If the LIT entry occurs in the middle of the program, the literals up to that point are punched or printed starting with the first available location after LIT; the literal pool is reinitialized as if the assembly had just begun.

If there are literals remaining in the pool when the END card is encountered, the origin of the literal pool is one location past the final word defined by the program. The maximum number of LIT pseudo-operations allowed in a program is 63.

### CONDITIONAL PSEUDO-OPERATIONS

The pseudo-operations INE, IFE, IFL, and IFG add flexibility to variable-length or conditional expansions of the macro prototype. When used within a macro call, the conditional pseudo-operation affects cards within the macro operation itself. The use of these pseudo-operations, however, is not limited to macro operations; they may be used in a subprogram to effect conditional assembly of segments of the program.

The programmer must avoid using noncomparable elements within these pseudo-operations. The first comma encountered in the variable field is considered as separating the first subfield from the second subfield (the fields to be compared). Symbols used in the variable field will normally have been previously defined. On the other hand, one of the primary uses of conditional pseudo-operations is to test if a symbol has been defined at a given point in an assembly. Consequently, undefined symbols within a conditional are not flagged in the left margin of the listing. If the symbol is never defined within the assembly, the symbol will be listed as undefined at the end of the listing.

Formats of conditional pseudo-operations follow.

## IFE - If Equal

1	8	16	
Blanks	IFE	X, Y, n or %aaa	Two or three subfields in the variable field. X is compared with Y; if equal, the next n cards are assembled. aaa is a defined symbol

The IFE pseudo-operation provides for conditional assembly of the next n cards, depending on the relationship of the first two subfields of the variable field. The next n cards are assembled if, and only if, the expression or alphanumeric string in the first subfield is equal to the expression or alphanumeric string in the second subfield. If the compared subfields are not equal, the next n cards are bypassed. Resumption begins at card n+1. The n is specified in the third subfield and is assumed to be one, if not specified.

Two types of comparisons are possible in the subfields of the IFE pseudo-operation. The first is an algebraic comparison after the expression has been evaluated. The second is an alphanumeric comparison and the relation is the collating sequence. Alphanumeric strings in the variable field of IFE are denoted by placing the subfield within apostrophe marks. If either the first or second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

If the first character of the condition span argument is the percent symbol (%), the rest of the field is assumed to be a symbol defined in a succeeding line of coding. If the condition fails (i.e., the conditional coding is skipped), the assembler scans forward to the line which contains the symbol and resumes assembly with that line. If the symbol is not detected prior to the end of the program, an error flag is printed. If the conditional assembly occurs within a macro skeleton, and the symbol is not detected prior to the end of the macro operation, an error flag is printed and assembly resumes with the first line following the macro operation.

## IFG - If Greater Than

1	8	16	
Blanks	IFG	X, Y, n or %aaa	Two or three subfields in the variable field. If X is greater than Y, the next n cards are assembled. aaa is a defined symbol

The IFG pseudo-operation provides for conditional assembly of the next n cards, depending on the value of the first two subfields of the variable field. The next n cards are assembled if, and only if, the expression or alphanumeric string in the first subfield is greater than the expression or alphanumeric string in the second subfield. If the first subfield is not greater, the next n cards are bypassed. Resumption begins at card n+1. The n is specified in the third subfield and is assumed to be one if not present.

Two types of comparisons are possible in the subfields of the IFG pseudo-operation. The first is a straight numeric comparison after the expression has been evaluated. The second is an alphanumeric comparison, using the relation of the collating sequence. Alphanumeric strings in the variable field of the IFG are denoted by placing the subfield within apostrophe marks. If either the first or the second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

If the first character of the condition span argument is the percent symbol (%), the rest of the field is assumed to be a symbol defined in a succeeding line of coding. If the condition fails (i.e., the conditional coding is skipped), the assembler scans forward to the line which contains the symbol and resumes assembly with that line. If the symbol is not detected prior to the end of the program, an error flag is printed. If the conditional assembly occurs within a macro skeleton, and the symbol is not detected prior to the end of the macro operation, an error flag is printed and assembly resumes with the first line following the macro operation.

#### IFL - If Less Than

1	8	16	
Blanks	IFL	X, Y, n or %aaa	Two or three subfields in the variable field. If X is less than Y, the next n cards are assembled. aaa is a defined symbol

The IFL pseudo-operation provides for conditional assembly of the next n cards, depending on the value of the first two subfields of the variable field. The next n cards are assembled if, and only if, the expression or alphanumeric string in the first subfield is less than the expression or alphanumeric string in the second subfield. If the first subfield is not less, the next n cards are bypassed. Resumption begins at card n+1. The n is specified in the third subfield and is assumed to be one if not present.

Two types of comparisons are possible in the subfields of the IFL pseudo-operation. The first is a straight numeric comparison after the expression has been evaluated. The second is an alphanumeric comparison, using the relation of the collating sequence. Alphanumeric strings in the variable field of IFL are denoted by placing the subfield within apostrophe marks. If either the first or second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

If the first character of the condition span argument is the percent symbol (%), the rest of the field is assumed to be a symbol defined in a succeeding line of coding. If the condition fails (i.e., the conditional coding is skipped), the assembler scans forward to the line which contains the symbol and resumes assembly with that line. If the symbol is not detected prior to the end of the program, an error flag is printed. If the conditional assembly occurs within a macro skeleton, and the symbol is not detected prior to the end of the macro operation, an error flag is printed and assembly resumes with the first line following the macro operation.

## INE -If Not Equal

<u>1</u>	<u>8</u>	<u>16</u>	
Blanks	INE	X, Y, n or %aaa	Two or three subfields in the variable field. X is compared with Y; if not equal, the next n cards are assembled. aaa is a defined symbol

The INE pseudo-operation provides for conditional assembly of the next n cards, depending on the relationship of the first two subfields of the variable field. The value of the expression in the first subfield is compared to the value of the expression in the second subfield. If they are not equivalent, the next n cards are assembled, where n is specified in the third subfield; otherwise, the next n cards are bypassed, resumption beginning at the (n+1)th card. If the third subfield is not present, n is assumed to be one.

Two types of comparisons are possible in the subfields of the INE pseudo-operation. The first is an algebraic comparison after the expression has been evaluated. The second is an alphanumeric comparison, using the relation of the collating sequence. Alphanumeric strings in the variable field of INE are denoted by placing the subfield within apostrophe marks. If either the first or second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

If the first character of the condition span argument is the percent symbol (%), the rest of the field is assumed to be a symbol defined in a succeeding line of coding. If the condition fails (i.e., the conditional coding is skipped), the assembler scans forward to the line which contains the symbol and resumes assembly with that line. If the symbol is not detected prior to the end of the program, an error flag is printed. If the conditional assembly occurs within a macro skeleton, and the symbol is not detected prior to the end of the macro operation, an error flag is printed and assembly resumes with the first line following the macro operation.

## SPECIAL WORD FORMAT PSEUDO-OPERATIONS

### MARK - Specify Symbol in Location Field

<u>1</u>	<u>8</u>	<u>16</u>	
Symbol	MARK	Blanks or comment	in the variable field

The MARK pseudo-operation allows a symbol to be specified in the location field without having this symbol entered in the symbol table. The pseudo-operation is explicitly provided to allow the definition of the extent of a block of conditional code within a macro skeleton without creating multiple defined symbols. No word of coding is generated by this pseudo-operation. The same symbol may appear in several MARK pseudo-operations in the same program without generating error flags.

ARG - Argument--Generate Zero Operation Code Computer Word

1            8            16

---

Symbol ARG            One to three subfields in the variable field

The use of ARG pseudo-operation field causes the assembler to generate a binary word with bit configuration in the general instruction format. The operation code 00 is placed in the operation field. The variable field is interpreted in the same manner as a standard machine instruction.

TTLDAT - Title Date

1            8            16

---

Blanks TTLDAT  
or  
Symbol

The use of the TTLDAT pseudo-operation causes the date in columns 63-68 of the first TTL card to be assembled at the current available address in BCI format. If there is no date on the first TTL card, the characters NO TTL are assembled instead.

DATE - Current Date

1            8            16

---

Blanks DATE            Column 16 must be blank

The DATE pseudo-operation is used to enter the current date into a program. The six-character current date in the form mmddy is inserted into an assembled program at this point.

Example:

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>	
01021	000601	0	DATE
01022	050607	0	

This example shows the results of a DATE pseudo-operation assembled on 6/15/67.

NONOP - Undefined Operation

When an NONOP pseudo-operation is encountered, NONOP is looked up in the operation table and used in place of the undefined operation. NONOP is initially set as an error routine, but OPD, OPSYN or MACRO pseudo-operation can be used to redefine NONOP.



## ZERO - Generate One Word with Two Subfields

1            8            16

---

Symbol    ZERO            Two subfields in the variable field  
or  
blanks

The ZERO pseudo-operation is used for the definition of a value to be loaded into index registers. The assembler generates a binary word divided into two subfields. The first subfield is the address field: stored in bit positions 3-17. The second subfield is the character address field (optional): it can be a numeric or symbolic value and is stored in bit positions 0-2.

### Examples:

Assume it is necessary to load an index register with the character address of the starting location for the data buffer, BUFR. Assume also that the buffer contains six-bit data characters. Using the ZERO pseudo-operation, this word is defined:

```
ZERO        BUFR,C.0
```

With the location counter at 2057 (octal) and the location 1004 (octal) assigned for BUFR, this would print out (without column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
02057	4 01004	1    ZERO    BUFR,C.0

If instead of six-bit data characters the buffer contained nine-bit data characters, the following example creates a word suitable for use in character addressing:

```
ZERO        BUFR,B.0
```

With the location counter at 2057 (octal) and the location 1004 (octal) assigned for BUFR, this would print out:

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
02057	2 01004	1    ZERO    BUFR,B.0

## MAXSZ - Maximum Size of Assembly

1            8            16

---

Blank    MAXSZ            A decimal number in the variable field

The decimal number represents an estimate of the largest number of assembled instructions and data in the program or subprogram. The variable field number is evaluated, saved, and printed at the end of the assembly listing. It may then be compared with the actual size of the assembly.

MAXSZ pseudo-operation is provided as a convenience and may be inserted anywhere in the coding.

IND - Generate One Word for Indirect Addressing

1	8	16
Symbol	IND	Three subfields in the variable field, separated
or		by commas
blanks		

The IND pseudo-operation is used for the definition of a value to be used in indirect addressing. The assembler generates a binary word divided into three subfields. The first subfield is an address field placed into bit positions 3-17. The second subfield is a tag field having the same format as in memory reference instructions and is stored in bit positions 0-2. If a third subfield exists, it specifies a character address subfield. By definition, the first subfield becomes a 12-bit displacement subfield and the character address is stored in bit positions 3-5. Both the second and third subfields are optional.

Examples:

1. Assume a program has a Memory Reference Instruction whose effective address points to the location ALPHA, and the desired data is the location of BETA. The following IND pseudo-operation accomplishes this function.

ALPHA      IND      BETA

With the location counter at 4032 (octal) and the symbol BETA having the value 11236 (octal), the following would print out (without column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
04032	0 11236	1    ALPHA   IND   BETA

2. Consider the program situation where one wishes to use character addressing with an indirect word. Assume a Memory Reference Instruction points to location GAMMA indirectly. The following use of the IND pseudo-operation displays the use of this method of character addressing.

GAMMA    IND    0,1,C.0

With the location counter at 1530 (octal) the following would print out (without column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
01530	1 4 0000	0    GAMMA   IND   0,1,C.0

NOTE: With any type of character addressing, an index register must always be specified. Furthermore, the index register referenced must contain a compatible character address in bit positions 0-2. That is, an instruction using six-bit characters must reference an index register with six-bit character addressing.

# DATA CONTROL WORD FORMAT PSEUDO-OPERATIONS

## ICW - I/O Control Word Generator

<u>1</u>	<u>8</u>	<u>16</u>
Symbol	ICW	Four subfields in the variable field, separated by commas
or		
blanks		

The ICW pseudo-operation provides a simple way of generating Indirect Control Words for I/O programming. The assembler generates two words and forces the first word to always start in an even location.

The first subfield is an address subfield that points to the first word where data will be loaded or stored (depending on the particular I/O operation). It may be an absolute or relocatable value.

The second subfield is a character address subfield that specifies the size of data transmission fields and the starting position for character transmissions. The allowable symbols in this subfield are:

<u>Symbol</u>	<u>Value</u>	<u>Meaning</u>
W.1	0	18 bit data transmission
W.2	1	36 bit data transmission
B.0	2	9 bit data transmission, char 0
B.1	3	9 bit data transmission, char 1
C.0	4	6 bit data transmission, char 0
C.1	5	6 bit data transmission, char 1
C.2	6	6 bit data transmission, char 2
I	7	Indirect Idle, no data transmission

The third subfield specifies a tally count. The count is the number of individual data transmissions required to send or receive a block of data. This subfield is optional. If no subfield is present, the field is set to zero.

The fourth subfield is used to set the exhaust bit in the Indirect Control Word. It must be an absolute expression having the value zero or one. If the exhaust bit is equal to one, neither the address fields or tally are incremented during an I/O operation. This is an optional field and is assumed to be zero if absent.

### Example:

Assume it is desired to transmit a block of data to a six bit character channel device. An ICW pseudo-operation may be used to create an Indirect Control Word for the transmission:

```
ICW  BUFFER,C.0,160
```

With the location counter at 1163 (octal) and the location 3046 (octal) assigned for BUFFER, this would print out (without column headings):

<u>Location</u>	<u>Contents</u>	<u>Relocation</u>
01163	233100	0
01164	4 03046	1 ICW BUFFER,C.0,160
01165	000240	0

## DCW - I/O Control Word Generator

1	8	16
Symbol	DCW	Three subfields in the variable field, separated by commas
or		
blanks		

The DCW pseudo-operation is similar to the ICW pseudo-operation. This pseudo-operation generates an I/O control word for a direct I/O channel. Direct channels normally transmit data to and from memory 36 bits at one time. Thus for direct channels, the character address subfield in the ICW pseudo-operation is a fixed value. This pseudo-operation, therefore, eliminates the character address subfield in the ICW.

The first subfield is an address subfield. It should point to the first word of a data buffer which begins in an even location. This address can be absolute or relocatable.

The second subfield specifies a tally count. This count is the number of word pairs to be transferred for this I/O operation. This subfield is optional and if absent, zero is assumed.

The third subfield is used to set the exhaust bit. The subfield must be an absolute expression having the value zero or one. This is an optional field and is assumed to be zero if not present.

### MACRO PSEUDO-OPERATIONS

Programming applications frequently involve:

1. Coding of a repeated pattern of instructions that within themselves contain variable entries at each iteration of the pattern.
2. Basic coding patterns subject to conditional assembly at each occurrence.

The macro pseudo-operation provides a shorthand notation for handling this special type of operation. Having once determined the iterated pattern, the programmer can, within the MACRO pseudo-operation, designate selectable fields of any instruction of the pattern as variable. Thereafter, by coding a single macro entry, the entire pattern can be used as many times as needed, substituting different parameters for the selected subfields on each pass.

When the iterated pattern is defined by a name, this name becomes the operation code of the macro entry.

As a generative operation, the macro operation causes n card images (where n is normally greater than one) to be generated; these may have substitutable arguments. The macro entry is known as the prototype or skeleton, and the card images that may be defined are almost unrestricted as to type:

1. Any processor instruction
2. Almost any assembler pseudo-operation
3. Any previously defined macro operation

Card images of these types are subject to the same conditions and restrictions when generated by the macro processor as though they had been produced directly as inline coding.

To use the macro prototype, once named, the programmer enters the macro operation code in the operation field and arguments in the variable field of the MACRO entry. The arguments comprise variable-field subfields and refer directly to the argument pointers specified in the fields of the card images of the prototype. By suitably selecting the arguments in relation to their use in the prototype, the programmer causes the assembler to produce inline coding variations of the n card images defined within the prototype.

The effect of a macro operation is the same as an open subroutine it produces inline coding to perform a predefined function. The inline code is inserted in the normal flow of the program so that the generated instructions are executed inline with the rest of the program each time the macro operation is used.

An important feature in specifying a prototype is the use of macro operations within a given prototype. The assembler processes such "nested" macro operations at expansion time only. The nesting of one macro definition within another prototype is not permitted. If macro operation codes are arguments, they must be used in the operation field for recognition. Thus, the macro entry must be defined before its appearance as an argument; that is, the prototype must be available to the assembler before encountering a demand for its use.

#### Definition of the Macro Prototype

The definition of a macro prototype is made up of three parts:

1. Creation of a heading card that assigns the prototype a name.
2. Generation of the prototype body of n card images with their substitutable arguments.
3. Creation of a prototype termination card.

## MACRO - MACRO Identification

<u>1</u>	<u>8</u>	<u>16</u>
Symbol	MACRO	The variable field may contain blanks or any number of options separated by commas

The MACRO pseudo-operation code defines a macro operation by symbolic name. The symbol in the location field conforms to standard symbol formation rules and defines the name of a macro call whose prototype is given on the next n lines. The prototype definition continues until the assembler encounters the proper ENDM pseudo-operation. The name of the macro call is a required entry. If the symbol is identical with an operation code already in the table, the macro operation is used as a new definition for that operation code. It is entered in the assembler operation table with a pointer to the associated prototype that is entered in the Macro Prototype Table.

The following options can be specified in the variable field of the MACRO pseudo-operation:

<u>Symbol</u>	<u>Meaning</u>
C	Comments option. This option instructs the assembler to save columns 1 through 72 of every macro prototype card within the range of this macro definition. This enables comments that are included in the macro definition to be printed in the macro expansion.
M	Multiple definition option. This option is used to suppress the multiple definition flag normally given to any MACRO pseudo-operation that redefines an existing operation code mnemonic.

### Example:

```
LDM  MACRO C,M
```

### END - End Macro Prototype

<u>1</u>	<u>8</u>	<u>16</u>
Blanks	ENDM	A symbol in the variable field

The symbol in the variable field of the ENDM pseudo-operation is the symbolic name of the macro operation as defined in the location field of the corresponding MACRO pseudo-operation (heading card). Every macro prototype must contain both the terminal ENDM pseudo-operation and the MACRO pseudo-operation.

Thus, every macro prototype has the form

Heading card	{	OPNAME	MACRO
			----
			----
			----
Prototype body	{		.
			:
			.
			----
			----
Terminal card		ENDM	OPNAME

where OPNAME represents the prototype name that is placed in the Assembler Operation Table.

The prototype body contains a sequence of standard source-card images (of the types listed earlier) that otherwise would be repeated frequently in the same source program. Thus, for example, if the iterated coding pattern

```

LDA      A+5 - *
LDQ      B+5 - *
STA      C - *
STQ      D - *
.
.
LDA      U - *
LDQ      Y - *
STA      BETA - *
STQ      ALPHA - *
.
.
LDA      W+X - *
LDQ      Y+Z - *
STA      GAMMA - *
STQ      NEXT1 - *

```

appeared in a subprogram, it can be represented by the following prototype body (preceded by the required prototype name):

1	8	16	
<hr/>			
LDM	MACRO		Macro prototype with substitutable
	LDA	#1-*	arguments in the variable field
	LDQ	#2-*	
	STA	#3-*	
	STQ	#4-*	
	ENDM	LDM	

The previous coding examples could then be represented by the macro operation LDM as follows:

```

LDM      (A+5), (B+5), C, D
- - -
- - -
LDM      U, V, BETA, ALPHA
- - -
- - -
LDM      (W+X), (Y+Z), GAMMA, NEXT1

```

The assembler recognizes substitutable arguments by the presence of the number-sign (#) identifier. Having sensed this identifier, it examines the next one or two digits. (Sixty-three is the maximum number of arguments usable in a single prototype.)

Macro prototype arguments may appear in the location field, in the operation field, in the variable field, and coincidentally in combinations of these fields within a single card image. Substitutions that can be made in these fields are:

1. Location field--any permissible location symbol (see comments below)
2. Operation field--all machine instructions, all pseudo-operations (except the MACRO pseudo-operation) and previously defined macro operations
3. Variable field--any allowable expression followed by an admissible modifier tag and separated from the expression by a delimiting comma.

In general, anything appearing to the right of the first blank in the variable field is copied into the generated card image. For example, a substitutable argument appearing in the comments field of a card image--that is, separated from the variable field by one or more blanks--is not interpreted by the assembler (except in the case of the ACI, ASCII, ACIC, ASCIIIC, BCI, REM, TTL, and TTLS pseudo-operations). This means that only pertinent information in the location, operation, and variable fields is recognized, that internal blanks are not allowed in these fields, and that the first blank in these fields causes field termination.

When specifying a symbol in a location field of an instruction within a prototype, the programmer must be aware that this macro operation is used only once. On the second use, the same symbol is redefined causing a multiple-defined symbol. Consequently, the use of location symbols within the prototype is discouraged. Alternatively, for cases where repeated use of a prototype is necessary, two techniques are available: (1) use of Created Symbols and (2) placement of a substitutable argument in the location field and a unique symbol in the argument of the macro operation each time the prototype is used. These techniques are described below under the caption "Using a Macro Operation".

The location field, operation field, and variable field can contain text and arguments that can be linked by entering the substitutable argument (for example, AB#3) directly in the text, with no blanks or special symbols preceding or following the entry. Linking is especially useful in the operation field and in the partial subfields of the variable field. (Refer to the description of ACI, ACIC, ASCII, ASCIIIC, BCI, REM, TTL, and TTLS immediately following.) As an example of the first use, consider a machine instruction such as LD(R) where R can assume the designators A, Q, AQ, and X1-X3.

The prototype NAME

```

NAME          MACRO
-----
-----
LD#2
-----      A,#1
-----
ENDM          NAME

```

contains a partial operation field argument; when the inline coding is generated, LD#2 becomes LDA, LDQ, etc., as designated by the argument used in the macro operation.



The ACI, ACIC, ASCII, ASCIIC, BCI, REM, TTL, and TTLS pseudo-operations used within the prototype are scanned in full for substitutable arguments. The variable field of these pseudo-operations can contain blanks and argument pointers. The following illustrates a typical use:

```

ALPHA                MACRO
                    -----
                    -----
                    -----
NOTE#1              REM          IGNORE#2#ERRORS#ON#3
                    -----
                    -----
                    ENDM          ALPHA

```

An asterisk (\*) type remarks card cannot appear in a macro prototype.

### Using a Macro Operation

Use of a macro operation can be divided into two basic parts; definition of the prototype and writing the macro operation. The first part has been described on the preceding pages; writing the macro operation to call upon the prototype is the process of using the MACRO pseudo-operation.

The macro operation card is made up of two basic fields; the operation field that contains the name of the prototype being referenced and the variable field that contains subfield arguments relating to the argument pointers of the prototype on a sequential, one-to-one basis. For example, the defined prototype LDM, mentioned earlier, can be called for expansion by the macro entry

```
LDM          (A+5), (B+5), C, D
```

where the variable field arguments, separated by commas and taken left-to-right, correspond with the prototype pointers #1 through #4. These arguments are then substituted in the corresponding positions of the prototype to produce a sequence of instructions using these arguments in the assigned location, operation, and variable fields of the prototype body.

The maximum number of macro call arguments is 63; arguments greater than 63 are treated modulo 64. For example, the 70th argument is the same as the 6th argument and is so recognized by the assembler. Each such argument can be a literal, a symbol, or an expression (delimited by commas) that conforms to the restrictions imposed upon the field of the machine instruction or pseudo-operation within the prototype, where the argument will be inserted.

The following conditions and restrictions apply to the expansion of macro operations:

1. Anything appearing in the location field of a prototype card image, whether text or a substitutable argument, causes generation to begin in column 1 for that text or argument.
2. Location field text generated from an argument pointer (in a prototype location field) producing a resultant field extending beyond column 8 causes the operation field to begin in the next position after the generated text. Normally, the operation field begins in column 8.

3. Operation field text generated from an argument pointer (in a prototype operation field) producing a resultant field extending beyond column 16 causes the variable field to start in the next position after the generated text. Normally, the variable field will begin in column 16.
4. The variable field may begin after the first blank that terminates the operation field but not later than column 16 in the absence of the condition in 3 above.
5. No generated card image may have more than 72 characters recorded; that is, the capacity of one card image cannot be exceeded (columns 73-80 are not part of the card image).
6. No argument string of alphanumeric characters can exceed 57 characters.
7. Up to 63 levels of macro nesting are permitted.

An argument can also be declared null by the programmer when writing the macro operation; however, it must be declared explicitly null. Explicitly null arguments may be specified in either of two ways; by writing the delimiting commas in succession with no spaces between the delimiters, or by terminating the argument list with a comma with the next normal argument of the list omitted. (Refer to the "CRSM pseudo-operation".) A null argument means that no characters are inserted in the generated card image wherever the argument is referenced. When a macro operation argument relates to an argument pointer and the pointer requires the argument to have multiple entries or contains blanks, the corresponding argument must be enclosed within parentheses, with the parenthetical argument set off by the normal comma delimiters. The parenthetical argument may contain commas as separators. Examples of prototype card images that require the use of parentheses in the macro call are pseudo-operations such as IDRP, VFD, BCI, and REM, as well as the variable field of any entry where the address and tag may be one argument.

It is also possible to enclose an argument within brackets, making them subarguments; in the case blanks are ignored as part of the argument. For example the macro call of the macro operation named ABC may be written as

```

ABC      [A,
ETC      24,
ETC      2*D]
```

and is equivalent to

```

ABC      (A,24,2*D)
```

even though numerous blanks occur after the arguments A, and 24,. Thus, the assembler packs everything it finds within brackets and suppresses all blanks therein. The above manner of writing the macro call provides additional flexibility in programming one subargument per card with an ETC pseudo-operation, the blanks no longer being significant.

It may happen that the argument list of a macro operation extends beyond the capacity of one card. In this case, the ETC pseudo-operation is used to extend the list to the next card. In using ETC, the last argument entry of the macro operation is delimited by a following comma, and the first entry of the ETC card is the next argument in the list. Within the prototype, as many ETC cards as required may be used for internal macro operations, or VFD pseudo-operations.

Pseudo-Operations Used Within Macro Prototypes

Need for Prototype Created Symbols. A macro prototype, in which an argument pointer is used in the location field, requires that a new symbol be specified each time the prototype is called. In addition, where a nonsubstitutable symbol is used in a prototype location field, the macro operation may be used only once without incurring an assembler error flag on the second and all subsequent calls to that prototype (multiple-defined symbol). Primarily, to avoid the former task of having to repeatedly define new symbols when using the macro operation and to enable repeated use of a prototype with a location field symbol (nonsubstitutable), the created symbol concept is provided.

Use of Created Symbols. Created symbols are of the type .xxx. where xxx runs from 001 through 999, thus making possible up to 999 created symbols for an assembly. The periods are part of the symbol. The assembler generates a created symbol only if an argument in the macro operation is implicitly null; that is, only if the macro operation defines fewer arguments than given in the related macro prototype, or if the designator # is used as an argument. Explicitly null arguments will not generate.

Example:

Assume a macro prototype of the form

NAME	MACRO	
	-----	#1, #2
#4	-----	X-*
#5	-----	ALPHA, #3
	-----	#4-*
	TMI	#5-*
	EBDN	NAME

with five arguments, 1 through 5. The macro operation NAME in the form

NAME            A,2,,,B

specifies the third and fourth arguments as explicitly null; consequently, no created symbols are provided. The expansion of the operation is

	-----	A,2
	-----	X-*
B	-----	ALPHA (Unless a specified modification
	-----	is given, IC modification will be
		assumed.)
	TMI	B-*

The macro operation card image

NAME            A,2,

indicates the third argument is explicitly null, while arguments four and five are implicitly null. Consequently, created symbols are provided for arguments four and five, but not for three. This is shown in the expansion of the macro operation:

	-----	A,2
.011.	-----	X-*
.012.	-----	ALPHA, (Unless a specified modification
	-----	.011.-* given, IC modification will be
		.012 assumed.)
	TMI	

A created symbol may be requested for the third argument by omitting the last comma. To change an explicitly null argument to an implicitly null argument, insert the # designator in an explicitly null position. Thus, for the preceding example

NAME                   A,2,,#,B

the fourth argument becomes implicitly null and a created symbol will be generated.

#### CRSM ON/OFF - Created Symbols

1           8           16

---

Blanks CRSM           ON   Normal mode

Created symbols are generated only within macro prototypes. They can be generated for argument pointers in the location, operation, and variable fields of instructions or pseudo-operations that use symbols. Accordingly, the CRSM pseudo-operation affects only coding produced by the expansion of macro operations. CRSM ON causes the assembler to initiate or resume the creation of symbols; CRSM OFF terminates the symbol creation if CRSM ON was previously in effect. If the assembler is already in the specified mode, the pseudo-operation is ignored.

#### ORGCSM - Origin Created Symbols

1           8           16

---

Blanks ORGCSM       One expression in the variable field

The variable field of the ORGCSM pseudo-operation entry is evaluated and becomes the new starting value between the periods of the created symbols.

#### IDRP - Indefinite Repeat

1           8           16

---

Blanks IDRP       #3   An argument number or blanks in the variable field, depending on the IDRP of the IDRP pair

The purpose of the IDRP pseudo-operation is to provide an iteration capability within the range of the macro prototype by letting the number of grouped variables in an argument pointer determine the iteration count.

The IDRP pseudo-operation must occur in pairs, thus delimiting the range of the iteration within the macro prototype. The variable field of the first IDRP must contain the argument number that points to the particular argument used to determine the iteration count and the variables to be affected. The variable field of the second IDRP must be blank.

At expansion time, the programmer denotes the grouping of the variables (subarguments) of the iteration by placing them in parentheses, as the nth argument, where n is the argument value contained in the initial IDRP variable field entry.

IDRP is limited to use within the macro prototype, and nesting is not permitted. However, as many disjoint IDRP pairs may occur in one macro operation as is desired.

For example, given the macro skeleton

```

NAME          MACRO
              .
              .
              .
              IDRP          #2
              ADA          #2-*
              IDRP
              .
              .
              ENDM          NAME

```

the macro CALL (with variables X1,X2, and X3)

```

A             NAME          Q+2,(X1,X2,X3),B

```

generates

```

A             .
              .
              .
              ADA          X1-*
              ADA          X2-*
              ADA          X3-*
              .

```

In this example, arguments #1 and #3, Q+2, and B respectively, are used in the skeleton ahead of and after the appearance of the IDRP range-iteration pair.

#### DELM - Delete Macro Named

1	8	16
Symbol	DELM	A symbol in the variable field
or		
Blanks		

The DELM pseudo-operation deletes the macro named in the variable field from the macro prototype area, and disables the corresponding operation table entry. With this pseudo-operation, systems that require many macro prototypes, or that have minimal memory allocation at assembly time, can re-use this memory for redefining or defining new macro operations. Redefinition of a deleted macro name does not produce an M (multiply-defined) error flag on the assembly listing.

## Implementation of System Macro Operations

The assembler can load a unique set (or sets) of macro operations under control of a pseudo-operation. This permits the language processors to uniquely identify the standard system macro operations required for the assembly of their programs.

System macro operations are, by definition, located on the System File on a mass storage device. They are written by the System Editor, in System Loadable Format, as a freestanding system program. The catalog name is the same used by MAP in the loading operation. For proper implementation, the MASTER option of the System Editor parameters card must be specified. It may be in absolute or relocatable System Loadable Format.

This implementation technique permits any unit, or related group of MAP users to define and implement a unique set of system macro operations.

### PUNM - Punch Macro Prototypes

1            8            16

---

Blanks    PUNM            The variable field is not examined

This pseudo-operation causes the assembler, in pass one, to scan the operation table for all macro operations defined. It then appends the definitions to the end of the prototype table and constructs a control word specifying the length of this area and the number of macro operations defined therein.

At the beginning of pass two, this information is punched in relocatable binary instruction cards, along with \$ OBJECT, preface, and \$ DKEND cards. The primary SYMDEF of this deck will arbitrarily be .MACR.

In the normal preparation of system macro operations, it is not desirable to include the MAP system macro operations. For this reason, the assembly of a set of system macro operations should have NGMAC specified on the \$ 355MAP card.

### LODM - Load System Macro Operations

1            8            16

---

Blanks    LODM            Two subfields in the variable field

The LODM pseudo-operation causes the assembler to issue MME GECALL for a set of system macro operations. The name used in the GECALL sequence is the symbol taken from the first subfield of the variable field of the LODM pseudo-operation. Macro operations thus loaded are appended to (not overlaid) the Macro Prototype Table. They are defined and made available for immediate use. If a macro prototype is redefined by this operation, the LODM operation is flagged with an M. If the user wishes to suppress the M flag, the second subfield must contain an M.

Example:

LODM .GRTM,M

Notes and Examples on Defining a Prototype

The following examples show some of the ways in which macro operations may be used.

1. Field substitution

Prototype definition:

```
ADDTO      MACRO
            LDA      #1 - *
            ADA      #2
            STA      #3 - *
            ENDM     ADDTO

Use:
            ADDTO    A,(1,1),B+5
```

2. Linkage of text and arguments

Prototype definition:

```
INCX      MACRO
            IACIACX #3      #2
            INE      #1,'1'
            TRA      #1 - *
            ENDM     INCX

Use:
            INCX      LOCA,1,1
or
            INCX      1,1,1
```

3. Argument in a BCI pseudo-operation

Prototype definition:

```
ERROR     MACRO
            TSY      DIA-*
            ZERO     #1
            BCI      10,ERROR,#1/CONDITION/IGNORED
            ENDM     ERROR

Use:
            ERROR     5
```

4. Macro operation in a prototype

Prototype definition:

```
TEST      MACRO
            LDA      #1-*
            CMPA     #2-*
            #3       #4-*
            ERROR    #5
            ENDM     TEST

Use:
            TEST     A,B,TZE,ALPHA,3
```

## 5. Indefinite Repeat

Prototype definition (for generating a symbol table):

SYMGEN	MACRO	
	IDRP	#1
#1	BCI	2,#1
	IDRP	
	ENDM	SYMGEN
Use:	SYMGEN	(LABEL,TEST,ERROR,MACRO)

## 6. Subroutine macro CALL

Prototype definition:

DOO	MACRO	
K	SET	0
	IDRP	#2
K	SET	K+1
	IDRP	
	TSY	#1-*
	TRA	1+K
	IDRP	#2
	ZERO	#2
	IDRP	
	ENDM	DOO
Use:	DOO	SRT, (ARG1,ARG2,ARG3)

## PROGRAM LINKAGE PSEUDO-OPERATIONS

### CALL - Call Subroutines

1            8            16

Symbol CALL      Subfields in the variable field with contents and  
or                    delimiters as described below  
blanks

The CALL pseudo-operation generates the standard subroutine calling sequence.

The first subfield in the variable field of the operation is separated from the next n subfields by a left parenthesis. This subfield contains the symbol that identifies the subroutine being called. It is possible to modify this symbol by separating the symbol and the modifier with a comma. (In a relocatable assembly the symbol entered in this subfield is treated as if it were entered in the variable field of a SYMREF operation.)

The next n subfields are separated from the first subfield by a left parenthesis and from subfield n+1 by a right parenthesis. Thus the next n subfields are contained in parentheses and are separated from each other by commas. The contents of these subfields are arguments used in the subroutine being called.



The next  $m$  subfields are separated from the previous subfields by a right parenthesis and from each other by commas. These subfields are used to define locations for error returns from the subroutine. If no error returns are needed, then  $m=0$ .

The last subfield contains an identifier for the operation. This identifier is used when a trace of the program is made. The identifier may be an expression contained in apostrophes. Thus the last subfield is separated from the previous subfields by an apostrophe. If the last subfield is omitted, the assembly program provides an identifier (the assigned alter number of the CALL pseudo-operation itself.)

In the following examples, the calling sequences generated by the pseudo-operation are listed below the CALL pseudo-operation. For clarification, AAAAA defines the location of the CALL pseudo-operation; SUB is the name of the subroutine called; MOD is an address modifier; A1 through An are arguments; E1 through Em define error returns; E.I. is an identifier; and .NAME. is the location of the first SYMDEF defined in the routine. The number sequences 1,2,...,n and 1,2,...,m designate argument positions only.

Relocatable mode:

```

AAAAA CALL SUB,MOD(A1,A2,...,An)E1,E2,...,Em'E.I.'
AAAAA TSY 2,I
      TRA 4+n+m
      IND SUB,MOD
      ZERO .NAME.
      ZERO E.I.
      ZERO A1
      ZERO A2
      .
      .
      ZERO An
      IND Em
      .
      .
      IND E2
      IND E1

```

Absolute mode:

```

AAAAA CALL SUB,MOD(A1,A2,...,An)E1,E2,...,Em'E.I.'
AAAAA TSY 2,I
      TRA 4+n+m
      IND SUB,MOD
      ZERO .NAME.
      ZERO E.I.
      ZERO A1
      ZERO A2
      .
      .
      ZERO An
      IND Em
      .
      .
      IND E2
      IND E1

```

If the variable field of the CALL cannot be contained on a single line of the coding sheet, it may be continued in succeeding lines by using the ETC pseudo-operation. This is done by terminating the variable field of the CALL operation with a comma. The next subfield is then the first subfield of the ETC pseudo-operation. Subsequent subfields may be continued in following lines in the same manner.

When a CALL to an external subprogram appears within a headed section, the external subprogram must be identified by a six-character symbol (immune to HEAD).

If a CALL is being used to access an internally defined subroutine, the subroutine must be placed before the CALL in the program deck. Also, a SYMDEF pseudo-operation with the symbol identifying the subroutine in its variable field must be placed before the CALL in the program deck. Starting the subroutine with a SAVE pseudo-operation automatically provides the SYMDEF.

SAVE - Save--Return Linkage Data

1	8	16
Symbol	SAVE	Blanks or subfields separated by commas in the variable field--as described below

The SAVE pseudo-operation produces instructions necessary to save specified index registers and the contents of the indicator register.

The symbol in the location field of the SAVE operation is used for reference by the RETURN operation. (This symbol is treated by the assembler, in the relocatable mode, as if it had been coded in the variable field of a SYMDEF operation.

The subfields in the variable field, if present, each contain an integer 1-3. Thus, each subfield specifies one index register to be saved.

When the SAVE variable field is blank, the following coding is generated:

```

BBBBB   BCI      2,NAME
         IND
         TRA      4
         LDI      2
         TRA     BBBB-*,I
         ZERO
         STI     -1

```

The instructions generated by the SAVE pseudo-operation are listed below. Example 1 is in the relocatable mode, and Example 2 is in the absolute mode. The symbols il through in are integers 1-3.

NAME, as selected by the assembler, is the first SYMDEF defined in the routine. This may be accomplished explicitly with SYMDEF pseudo-operation, or implicitly with SAVE.

BBBBB is a symbol that must be present; it is always a primary SYMDEF.

Examples:

1.   BBBBB   SAVE     $i_1, i_2, \dots, i_n$   
  
          BCI     2, NAME  
BBBBB    IND  
          TRA      $4+2n$   
          LDX( $i_1$ )  $2+n$   
          LDX( $i_2$ )  $2+2$   
          .  
          .  
          .  
          LDX( $i_n$ )  $2+n$   
          LDI      $2+n$   
          TRA      $-(3+n)$   
          ZERO     $X(i_1)$   
          ZERO     $X(i_2)$   
          .  
          .  
          .  
          ZERO     $X(i_n)$   
          ZERO  
          STI     -1  
          STX( $i_1$ )  $-(2+n)$   
          STX( $i_2$ )  $-(2+n)$   
          .  
          .  
          .  
          STX( $i_n$ )  $-(2+n)$

2.   BBBBB   SAVE     $i_1, i_2, \dots, i_n$   
  
          IND  
BBBBB    TRA      $4+2n$   
          LDX( $i_1$ )  $2+n$   
          LDX( $i_2$ )  $2+n$   
          .  
          .  
          .  
          LDX( $i_n$ )  $2+n$   
          LDI      $2+n$   
          TRA      $-(3+n)$   
          ZERO     $X(i_1)$   
          ZERO     $X(i_2)$   
          .  
          .  
          .  
          ZERO     $X(i_n)$   
          ZERO  
          STI     -1  
          STX( $i_1$ )  $-(2+n)$   
          STX( $i_2$ )  $-(2+n)$   
          .  
          .  
          .  
          STX( $i_n$ )  $-(2+n)$

## RETURN - Return--From Subroutines

1	8	16
Symbol	RETURN	One or two subfields in the variable field
or		
blanks		

The RETURN pseudo-operation is used for exit from a subroutine. The pseudo-operations generated by a RETURN pseudo-operation must make reference to a SAVE operation within the same subroutine. This is effected by the first subfield of RETURN, which must always be present. This symbol must be defined in the location field of a SAVE pseudo-operation.

The second subfield is optional and, if present, specifies the particular error return to be made; if the second subfield contains the value k, then the return is made to the kth error return.

In the examples following, the assembled instructions generated by RETURN are listed below the RETURN operation. For both examples the assembler generates the same group of instructions when in either the relocatable or absolute mode.

### Examples:

1. RETURN BBBB  
TRA 1,I  
IND BBBB+2
2. RETURN BBBB,K  
STA 8  
LDA 8,I  
ERA 8  
IAA -k  
ASA 7,I  
LDA 3  
TRA 1,I  
IND BBBB+2  
IND  
IND BBBB,I  
TRA 0,I  
IND BBBB

## ETC - Continuation

<u>1</u>	<u>8</u>	<u>16</u>
Blanks	ETC	Variable field contains information overflow of previous card

The ETC pseudo-operation permits the continuation of a variable field that exceeds the space limitations of one card. The variable field of the previous card must be terminated by a comma to indicate continuation. There is no limit as to the number of ETC cards utilized. The ETC pseudo-operation can be used with any macro, VFD, and CALL pseudo-operation; it cannot be used with the DUP pseudo-operation. ETC can not be used to continue a macro operation.

If the ETC pseudo-operation is used within a macro skeleton and within an IDRP loop, the IDRP loop must not intervene between the ETC and the subject instruction or another connecting ETC. For example:

```
VFD #6018/777001,  
IDRP #1  
ETC 09/#1  
IDRP #1
```

Note that this example violates both parts of the restriction.

## SYSTEM (BUILT-IN) SYMBOLS

It is possible to include additional permanently defined system symbols in the assembler. This is done by a reassembly of the Macro Assembler and by placing the necessary information in the required tables.

## SECTION V

### INPUT/OUTPUT OPERATIONS

Program control of input/output operations is by the use of the Connect Input/Output Channel (CIOC), Load External Channel (LDEX), Store External Channel (STEX), and Select Input/Output Channel (SEL) machine instructions. These instructions transfer data between memory and the channels on the Input/Output Multiplexer (IOM).

The CIOC instruction always accesses a double-precision (36-bit) Peripheral Control Word (PCW) and sends it, or portions thereof, to the channel indicated by the I/O Channel Select Register. If the channel has a 6-, 9-, or 18-bit interface, it uses only part of the word.

The LDEX and STEX instructions are used to transfer data to and from channels operating in a static mode. The channel for the transfer is designated by the I/O Channel Select Register. The I/O Channel Select Register is loaded by the SEL (Select I/O Channel) instruction.

The instructions are executed as follows:

1. The processor decodes the LDEX, STEX, or CIOC.
2. The processor calculates the effective address by the normal method.
3. When the effective address is generated, the processor sends a request for transfer directly to the IOM and presents the address and channel number to the IOM for that transfer. The processor indicates the type of transfer required (CIOC,LDEX,STEX).
4. When the IOM is ready, it initiates a memory cycle using the address supplied by the processor. When the cycle is complete, the IOM releases the waiting processor which resumes its program. The processor is assigned top priority in accessing the IOM so that the processor waits at most, one memory cycle.

#### PERIPHERAL CONTROL WORD

Program control of dynamic channels is accomplished by the connect sequence. In the connect sequence the IOM pulls a Peripheral Control Word (PCW) from memory into the IOM registers. The format of the PCW varies from channel to channel; but, in general, it specifies the device command to be executed and other information needed by the channel. The PCWs for specific channels are described in this section.

## DIRECT CHANNEL PROGRAMMING

Channels that operate in a direct mode (maintain address and tally outside memory) are normally high performance, high data-rate channels.

After initiation by the CIOC instruction, the IOM pulls the PCW from a memory location. The IOM places the required portions of the PCW on the output lines to the selected channel. The device interprets the PCW and requests the first Data Control Word (DCW) and stores it in the channel. The DCWs may be in a sequential list or a threaded list determined by the channel.

Once initiated, the channel requests transfers asynchronously from the program until the DCW(s) is exhausted. Status information is normally obtained from this type of high performance channel in the form of a pre-assigned status location that the channel updates in memory immediately after completing the data transaction. The channel normally signals the end of the transaction by issuing a Set Interrupt Cell request.

## INDIRECT CHANNEL PROGRAMMING

The indirect channel operates much the same as the direct channel using an Indirect Control Word (ICW). In the indirect mode, the ICW is obtained from a memory location by the IOM each time a channel requests an indirect data transfer, thus relieving the channel of the addressing and tally counting.

The ICW must be set up by the program prior to issuing the CIOC instruction to the channel. Once started, the indirect device operates independent of the program, signalling the end of a data transfer by a Set Interrupt Cell request.

Status for an indirect channel is normally placed in memory, using a status ICW, after the data transfer and before the Set Interrupt Cell request is issued by the channel.

## PROGRAM INTERRUPT CONTROL

The program interrupt is the main method of communication between input/output and the program. The interrupt enable register has 16 bits, one for each of 16 interrupt levels. There are 16 interrupts per level for an effective 256 program interrupts. The interrupts are enabled in groups of 16. The Set Interrupt Enable Register (SIER) and Read Interrupt Enable Register (RIER) instructions provide program access to the Interrupt Enable Register.

## STATUS

For most devices, status is stored in the same manner as data, using an ICW or DCW. The Store Status and Set Interrupt Cell may be requested simultaneously by the channel. Status word formats vary with the types of channels and are specified with the individual channel in this section.

Static status information may be obtained by the program from a channel at any time by using the STEX instruction.

## IOM FAULTS

The IOM detected faults result in a status in octal locations 420-447 (as determined by the channel number, modulo 16) store sublevel and a program interrupt (level 0, determined by the channel, modulo 16).

The fault status word has the following format:

0	78	1011	1314	17
MBZ	Data Command	Interrupt Command	Fault Type	

where:

Data Commands may be

<u>Bits</u> 8-10	<u>Meaning</u>
000	None
001	Load
010	Store
011	ADD
100	Subtract
101	AND
110	OR
111	Fault

Interrupt Commands may be

<u>Bits</u> 11-13	<u>Meaning</u>
000	None
001	Unconditional
010	Conditional or TRO (Tally Run Out)
011	Conditional or PTRO (Pre-Tally Run Out)
100	Conditional or Data Negative
101	Conditional or Zero
110	Conditional or Overflow
111	Fault

<u>Bits</u> 14-17	<u>Fault</u> <u>Type</u>
0000	None
0100	Program Fault
1000	Memory Parity Error
1100	Illegal Command to IOM
1010	Adder/Bus Parity
1001	Indirect Channel Detected Parity
1101	Direct Channel Detected Parity
1111	IOM Bus Priority Break



The following code combinations cause an Illegal Channel Request Fault:

<u>Data Command Code</u>	<u>Interrupt Command Code</u>
7	X
X	7
0	0
0	2
0	3
0	4
0	5
0	6

X = Any code

### INTERCOMPUTER ADAPTER (ICA)

The Intercomputer Adapter (ICA) links the input/output bus of the DATANET FNP Input/Output Multiplexer (IOM) with the Central System controller. The ICA transfers data and control information back and forth between the DATANET FNP memory and the Central System memory.

The DATANET FNP generates a list of control words that define the location of data in its memory and the location in the Central System memory that is to reserve the data, sets up the Peripheral Control Word mailbox, and executes a Connect Input/Output Channel (CIOC) instruction to the ICA.

The Central Processor (program) sets up a mailbox with instructions to interrupt the ICA and then executes a CIOC to the ICA.

### DATANET FNP Interface

IOM Channel (patchable in ICA)	4
Interrupt Vectors (patchable in ICA)	
Fault (level 0)	100
Terminate (level 2)	102
Specials (all level 3)	03, 23, 43, 63, 103, ..., 363
Mailboxes	
IOC Fault Status	424
Status ICW (patchable in ICA)	452, 453
PCW (Mailbox - patchable in ICA)	454, 455
List ICW structure	

## Central System Interface

Up to four DATANET FNPs may be configured on a single Central System.

### INTERRUPT CELL ASSIGNMENT SWITCHES

<u>FNP Number</u>	<u>Switch Settings (for C/S with IOM)</u>
0	3
1	7
2	13
3	17

### PORT ASSIGNMENT FOR FNPs

The DATANET FNP must be assigned a higher port priority than the C/S processor unit, but a lower port priority than the C/S IOM.

### MAILBOX ADDRESSES

The Central System mailbox address for FNP-0 is

$$600_8 * A + 1400_8$$

where: A is the number of C/S IOMs configured.

Add  $100_8$  for each additional FNP configured.

### PROCESSOR FAULT SWITCHES

Central System processor fault switches should be set by the algorithm:

$$600_8 * A + 100_8 * B + 1400_8 \quad (\text{for FNP-0})$$

where: A is the number of C/S IOMs configured and  
B is the number of FNPs configured

Add  $40_8$  for each additional C/S processor unit configured.

EMERGENCY INTERRUPT CELL NUMBER

The emergency interrupt cell number is derived from the interrupt cell assignment switches on the ICA panel. It depends on the number of FNPs configured.

<u>FNP Number</u>	<u>Switch Setting</u>	<u>Emergency Interrupt Cell Number</u>
0	3	19
1	7	23
2	13	29
3	17	33

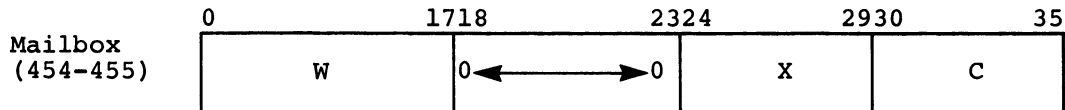
PCW MAILBOX

The FNP data PCW mailbox is set by switches on the ICA configuration panel. It may vary for different configurations, but is usually set at 454.

DATANET FNP Control Word Formats

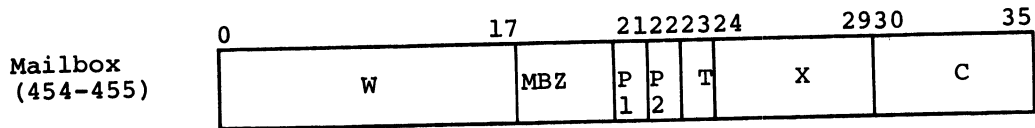
PERIPHERAL CONTROL WORD (PCW)

Model AA2



COMMAND	C-FIELD (Opcode)	W-FIELD	X-FIELD	NOTES
Connect	67	Central System address sent with connect	Not used	Only commands allowable if ICA <u>is</u> busy
Set Execute Cell	73	Central System address (used only for port select.)	Central System interrupt cell to be set	
None	Any except 67 or 73	DATANET FNP address of List ICW	Not used	PCW interpretation if ICA is <u>not</u> busy

Model AB1

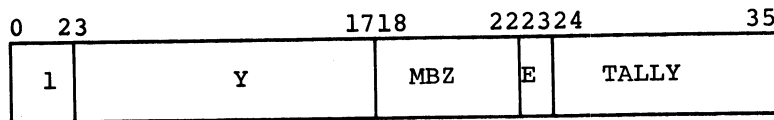


where:

- C } = Same as on previous page
- W } = Same as on previous page
- X } = Same as on previous page
- P1 = Odd parity for bits 0-17
- P2 = Odd parity for bits 18-35 including bit 21
- T = Test/normal mode; 0 = normal mode, 1 = test mode

LIST INDIRECT CONTROL WORD (LICW)

Model AA2



where:

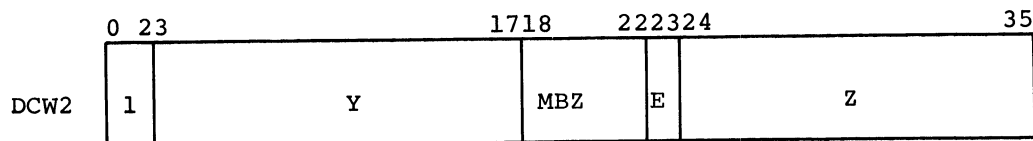
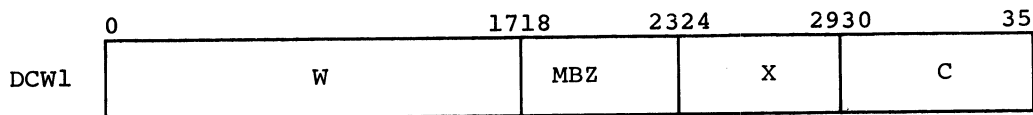
- Bits 0-2 = 001
- Y = Starting address of DCW list
- MBZ = Must be zero
- E = Exhaust bit for Tally field
- TALLY = Number of 36-bit words to transfer

Model AB1

Same as for Model AA2

DATA CONTROL WORD (DCW)

Model AA2

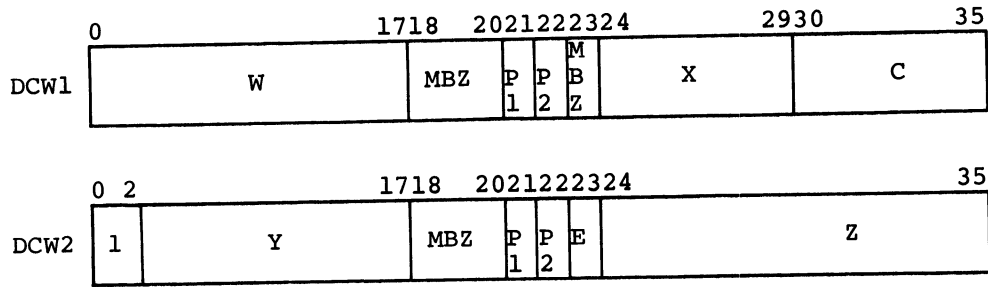


where:

C, W, X, Y, Z = Defined in following table  
 E, MBZ = Same as on previous page

Command	C-FIELD (Opcode)	W-FIELD (C/S Address)	X-FIELD	Y-FIELD (FNP Address)	Z-FIELD
EXECUTE	66	Used for port select only	Not used	Loc for XEC data	Not used
CONNECT	67	Specifies addr. sent with CON.	Not used	Not used	Not used
DISCONNECT	70	Not used	Not used	Not used	Not used
SXC 355 (Level 3)	71	Not used	Specifies channel number	Not used	Not used
JUMP	72	Not used	Not used	Specified addr. of new List ICW	Not used
Set Execute Cell	73	Used for port select only	No. of int. cell to be set	Not used	Not used
CONFIG.	74	Bits 16-17 used only	Not used	First loc. of config. data	Tally
DATA XFER to Central System	75	Central System starting address	Not used	DATANET FNP starting address	Tally
DATA XFER from Central System	76	Central System starting address	Not used	DATANET FNP starting address	Tally
DATANET FNP WRAPAROUND	77	Not used	Not used	DATANET FNP starting address	Tally (must be even)
READ/CLEAR Central System, or to STORE DATANET FNP	65	Central System starting address	Not used	DATANET FNP starting address	Tally

Model AB1



where:

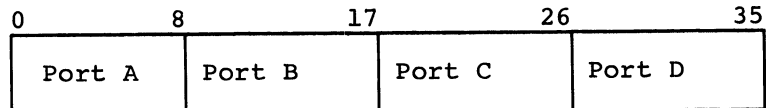
- E, MBZ = Same as on previous page
- P1 = Odd parity for bits 0-17
- P2 = Odd parity for bits 18-35 including bit 21
- C,W,X,Y,Z = Same as above, except for the following

Command	C	W	X	Y	Z
Illegal Command	77	-	-	-	-
DATANET FNP WRAPAROUND	66	Not used	Not used	DATANET FNP starting address	Tally (must be even)

CONFIGURATION STATUS FORMAT

System Controller Port Configurations (W<sub>16-17</sub> = 00)

Format (Loc. Y):



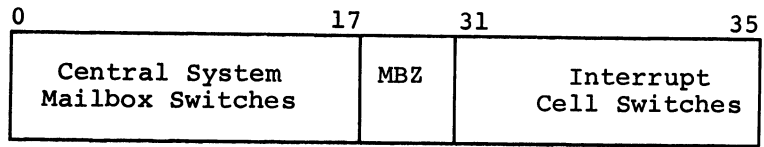
Bits

Function

- 0-2, 9-11, 18-20, 27-29 Indicates ICA configuration panel logical port number assignments.
- 3, 12, 21, 30 Indicates ports are interlaced (if = 1)
- 4, 13, 22, 31 Indicates ports are enabled (if = 1)
- 5, 14, 23, 32 Indicates whether a System Initialize signal is accepted (if = 1) or ignored (if = 0)
- 6-8, 15-17, 24-26, 33-35 Indicates memory size in each port.

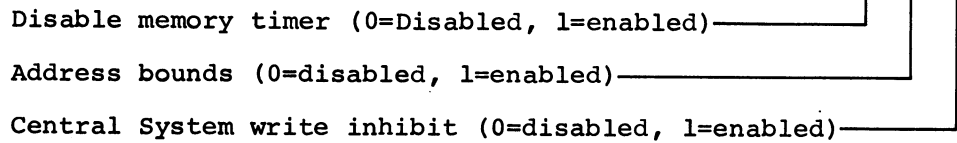
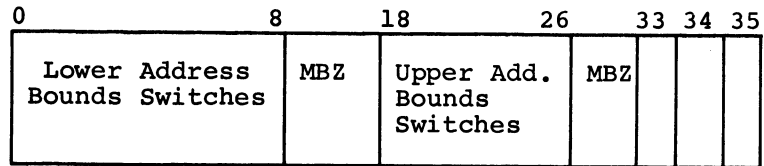
Central System Mailbox and Interrupt Cell Switches ( $W_{16-17} = 01$ )

Format (Loc. Y):



Lower/Upper Address Bounds ( $W_{16-17} = 10$ )

Format (Loc. Y):



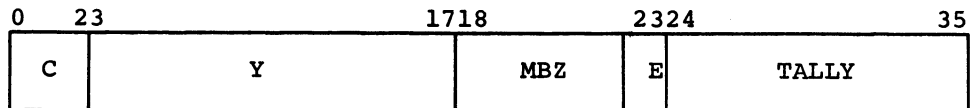
Zeros ( $W_{16-17} = 11$ )

Format (Loc. Y):

Always stored as a word of zeros.

ACTIVE STATUS FORMAT

Status Indirect Control Word (ICW)



- C field - 001 (indirect 36)
- Y field - DATANET FNP memory address where status will be stored
- E-bit - Exhaust bit (set to one when tally exhausted)
- TALLY field - Number of 36-bit words to be transferred.

STATUS WORD FORMAT

Model AA2

BIT	FUNCTION (Indication)	
0-2 } 3-5 } 6-8 } 9-11 }	Logical number assigned to physical ports A, B, C, D, respectively on ICA configuration panel	
12-18		MBZ
19		Odd tally for Opcode 77 from DATANET FNP
20		Illegal CON from DATANET FNP
21	Illegal Opcode from DATANET FNP	
22	List ICW TRO (Tally Run Out)	
23	List ICW did not specify 36-bit word	
24	Address < Lower Boundary	
25	Address > Upper Boundary	
26	Central System Write inhibit	
27	Central System Test command while busy	
28	Illegal Opcode from Central System	
29	Central System INA	
30	E bit set in List ICW	
31	Parity error (Central System)	
32 } 33 } 34 } 35 }	Illegal action from Central system	

Model AB1

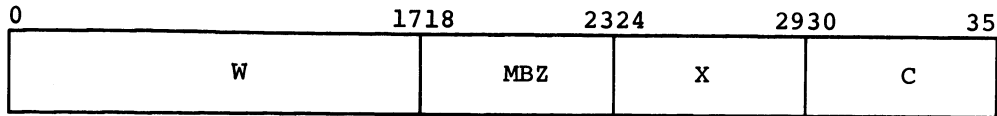
BIT	FUNCTION (Indication)	
0-2 } 3-5 } 6-8 } 9-11 }	Logical number assigned to physical ports A, B, C, D, respectively on ICA configuration panel	
12-14		MBZ
15		FNP Parity
16-18		MBZ
19	Odd tally for Opcode 66 from DATANET FNP	
20	Illegal CON from DATANET FNP	
21	Illegal Opcode from DATANET FNP	
22	List ICW TRO (Tally Run Out)	
23	List ICW did not specify 36-bit word	
24	Address < Lower Boundary	
25	Address > Upper Boundary	
26	Central System Write Inhibit	
27	Central System Test command while busy	
28	Illegal Opcode from Central System	
29	Central System INA	
30	E bit set in List ICW	
31	Parity error (Central System)	
32 } 33 } 34 } 35 }	Illegal action from Central system	



Central System Control Word Formats

PERIPHERAL CONTROL WORD (PCW) IN CENTRAL SYSTEM MAILBOX

Model AA2

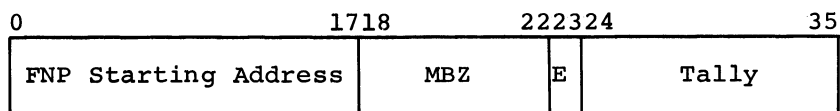


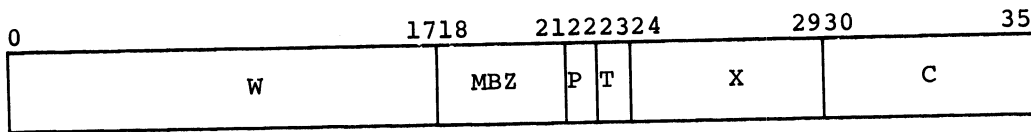
where:

W,X,C are as follows:

COMMAND	C-FIELD (Opcode)	W-FIELD C/S Address	X-FIELD
SXC DATANET FNP (Level 3)	71	Not used	Cell number
BOOTLOAD	72	Specifies Central System "Boot ICW" address <sup>1</sup>	Not used
SXC Central System	73	Specifies Central System address (used for port select only)	Cell number
TEST DATA XFER to Central System	75	Specifies Central System "Test ICW" address <sup>1</sup>	Not used
TEST DATA XFER to DATANET FNP	76	Specifies Central System "Test ICW" address <sup>1</sup>	Not used
Central System Wraparound	77	Specifies Central System address containing wraparound data	Not used

<sup>1</sup>TEST ICW and BOOT ICW format:





where:

P = Odd parity  
T = Test mode bit

W,X,C are as follows:

COMMAND	C-FIELD (Opcode)	W-FIELD C/S Address	X-FIELD
Store Emergency	40	Not used	Not used
Store Execute Register	41	Loaded into AR6	Loaded into Execute Register
Increment AR6	42	Not used	Not used
C/S Wraparound	74	C/S Address	Not used
Illegal Command	77	--	--

Actual Status Word Format

The Central System status word is stored in location = mailbox + 3

BIT	FUNCTION (Indication)
0-14	MBZ
15	
16-17	
18	FNP INA
19	Odd tally for Opcode 66 from FNP
20	Illegal CON from FNP
21	Illegal Opcode from FNP
22	List ICW TRO
23	List ICW did not specify 36-bit word
24	Address < Lower Boundary
25	Address > Upper Boundary
26	Central System Write Inhibit
27	Central System Test command while busy
28	Illegal Opcode from Central System
29	Central System INA
30	E bit set in List ICW
31	Parity error (Central System)
32	Illegal action from Central System
33	
34	
35	

## DIRECT INTERFACE ADAPTER (DIA)

The Direct Interface Adapter (DIA) provides a data and control information link between the DATANET FNP Input/Output Multiplexer (IOM) and the Central System IOM Direct Channel Adapter (DCA).

Data is transferred between the DIA and DCA on a 36-bit bidirectional interface and between the DIA and DATANET FNP on a 36-bit direct or indirect transfer mode.

### DATANET FNP Interface

I/O Channel Number = n

where: n is a switch settable value in the range 0 to 17 octal.

### Interrupt Vectors

Fault = n\*16 (level 0)

Terminate = n\*16 + t;

where: t is a switch settable value. The switches labeled TERMINATE INTERRUPT LEVEL specify the value of t. It can be set in the range 0 to 17 octal but must always be set to the value 2 for operation with GRTS software.

Special = x\*16 + s

where: s is a switch settable value. The switches labeled SPECIAL INTERRUPT LEVEL specify the value of s. It can be set in the range 0 to 17 octal, but must always be set to the value 3 for operation with GRTS software. The symbol x varies from 0 to 17 octal giving the sixteen different special interrupt vector addresses. If s is set to the value 3 (i.e. level 3), the special interrupt vectors would be assigned at the following locations: 03, 23, 43, 63, 103, 123, 143, 163, 203, 223, 243, 263, 303, 323, 343, 363.

### Mailboxes

IOM Fault Status = 400 + n

DIA Mailbox Base Address = m

where: m value is a switch settable value in the range 000 to 774 in increments of four. This address must always be 454 or above to prevent conflict with locations already assigned to interrupt vectors, interrupt cells, IOM fault status words, processor fault vectors, and the DATANET FNP timer mailbox. GRTS software further restricts the mailbox base address setting to be less than 500 octal. Thus the value of m must be one of the following for operation with GRTS software: 454, 460, 464, 470, 474.

PCW Mailbox = m, m+1

Status ICW Mailbox = m+2, m+3

## Central System Interface

Central System IOM Channel Number - Varies for each system.

### Interrupt Level Numbers

Fault = Level 1 (Fixed in hardware)

Terminate = Level 3 (Fixed in software)

Special = 7 used for the following three functions:

- (1) For terminating T and D commands (the switches labeled TERMINATE INTERRUPT LEVEL should be set to the value 7).
- (2) For reporting an emergency condition such as when the Central System attempts to access a DATANET FNP DIA in the masked state (the switches labeled EMERGENCY INTERRUPT LEVEL should be set to the value 7)
- (3) For reporting a GRTS software failure. (This interrupt LEVEL is fixed to the value 7 in GRTS software.)

When a special interrupt is received by GCOS, the DATANET FNP is considered to be down.

### Mailboxes

Central System IOM Fault Status is stored via the IOM Fault Channel mailbox as is the case for any other IOM channel.

Central System IOM Channel Mailbox is not used by the hardware since this channel is a Direct Channel.

Central System DIA Mailbox Base Address is set using the switches labeled PCW MAILBOX ADDRESS. This value varies with the number of IOMs configured in the Central System. Use the following algorithm to determine the proper value for FNP #0:

Mailbox Base = 1400 + 600 \*A

where: A = No. of IOM's configured.

DATANET FNP Control Word Formats

PERIPHERAL CONTROL WORD (PCW)

0	23	17	18	20	21	22	23	24	26	27	29	30	35
001	Y	0	0	0	0	P	P	M	0	0	0	X	C
					1	2							

where:

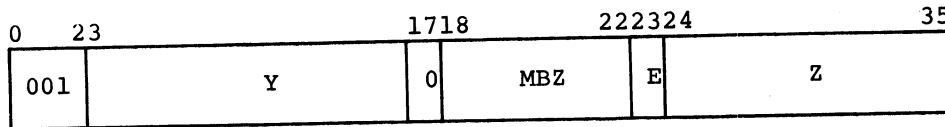
- Bits 0-2 = 001 Specifies an indirect 36 character address.
- Bit 21 = P1 Signifies the parity bit to make bits 0-17 of this word odd parity.
- Bit 22 = P2 Signifies the parity bit to make bits 18-35 of this word odd parity.
- Bit 23 = M<sup>1</sup> Specifies the DIA channel should be masked if this bit is equal to one. All other fields are ignored if this bit is one. If this bit is equal to zero, the DIA channel is unmasked and the operation code (C-field) is interpreted and executed.

Fields C,X,Y are as follows:

COMMAND	C-FIELD (Opcode)	Y-FIELD	X-FIELD	NOTES
Interrupt Central System	73	Not used	Central System IOM interrupt level no. to be set	
Start DIA List Service	Any legal Opcode except 73	DATANET FNP address of LIST ICW	Not used	PCW interpreted only if DIA is <u>not</u> busy

<sup>1</sup>This assumes the CIOC instruction is addressing the DIA PCW mailbox. If it is not, bit 23 is not used in the PCW mailbox.

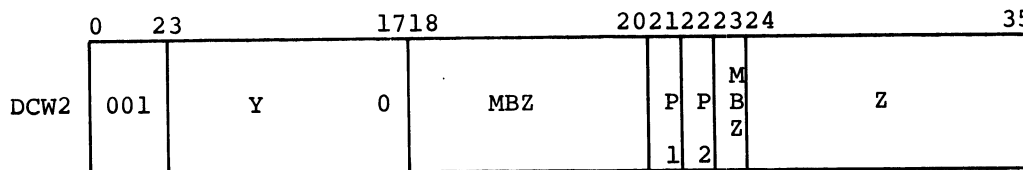
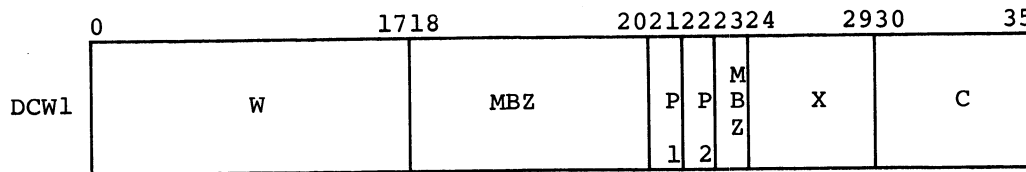
LIST INDIRECT CONTROL WORD (LICW)



where:

- Bits 0-2 = 001 Specifies an indirect 36 character address.
- Bits 3-17 = Y DATANET FNP address where DCW list begins. This must be an even location.
- Bit 23 = E Signifies the tally (Z-field) has exhausted. This bit is set to one when the tally is decremented from one to zero. Once the E bit is set, the Y and Z fields are not incremented or decremented further on subsequent accesses.
- Bits 24-35 = Z Specifies the number (tally) of 36-bit words to be processed in the DCW list.

Command Data Control Word (Command DCW)



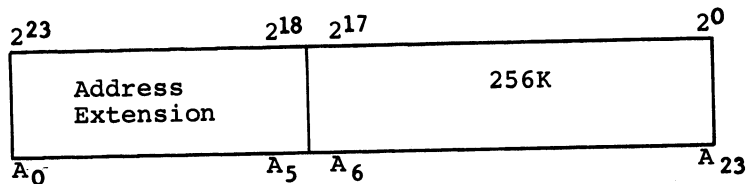
where:

- Bits 0-2 (DCW2) = 001 Specifies a direct 36 character address.
- Bit 21 = P1 Signifies the parity bit used by software to make bits 0-17 have odd parity.
- Bit 22 = P2 Signifies the parity bit used by software to make bits 18-35 have odd parity.

Fields C,W,X,Y,Z are as follows:

COMMAND	C-FIELD (Opcode)	W-FIELD (C/S Address)	X-FIELD	Y-FIELD FNP-ADDRESS	Z-FIELD
READ CLEAR Central System; or to STORE	65	Central System starting address	Extended address bits A0-A5	DATANET FNP starting address	Tally
DISCONNECT	70	Not used	Not used	Not used	Not used
INTERRUPT DATANET FNP (on level specified by SPECIAL INT. switch)	71	Not used	Specifies channel number in bits 26-29	Not used	Not used
JUMP	72	Not used	Not used	Specifies addr. of new List ICW	Not used
INTERRUPT Central System	73	Not used	Specifies interrupt level no. in bits 27-29	Not used	Not used
CONFIGURATION STATUS	74	Specifies starting configuration format in bits 16-17	Not used	Loc. of config. data	Tally
DATA XFER to Central System	75	Central System starting address	Extended address- bits A0-A5	DATANET FNP starting address	Tally
DATA XFER to DATANET FNP	76	Central System starting address	Extended address- bits A0-A5	DATANET FNP starting address	Tally

CENTRAL SYSTEM ADDRESS EXTENSION



Address extension bits  $A_0-A_5$  are obtained from the x-field of the DCW. This allows addressing 16K of Central System memory.

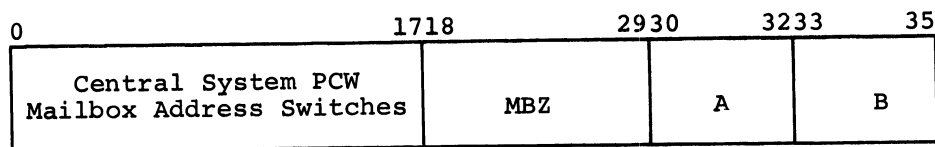
CONFIGURATION STATUS FORMAT

Zeros ( $W_{16-17} = 00$ )

Format (Loc Y): always stored as a word of zeros.

Central System Mailbox and Interrupt Level Switches ( $W_{16-17} = 01$ )

Format (Loc Y):

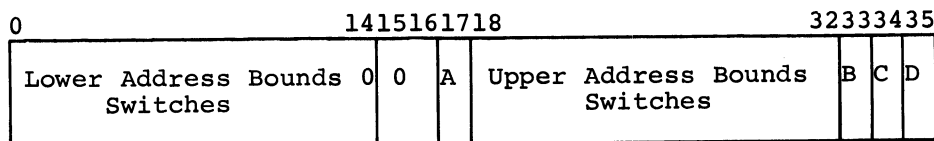


where:

- Bits 30-32 = A Specifies Test Data Terminate Level to be set on Central System IOM.
- Bits 33-35 = B Specifies Emergency Interrupt Level to be set on Central System IOM.

Lower/Upper Address Bounds ( $W_{16-17} = 10$ )

Format (Loc Y):



where:

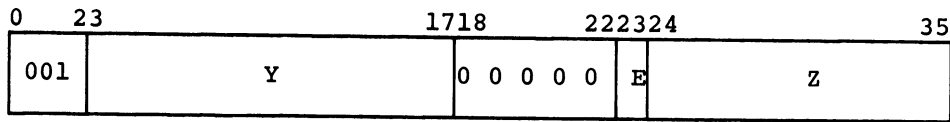
- Bit 17 = A Signifies bootload command is enabled if equal one or disabled if equal zero.
- Bit 33 = B Signifies memory timer is disabled if equal zero or enabled if equal one.
- Bit 34 = C Signifies address bounds check is disabled if equal zero or enabled if equal one.
- Bit 35 = D Signifies Central System Write Inhibit is disabled if equal zero or enabled if equal one.



Zeros ( $W_{16-17} = 11$ )

Format (Loc Y): always stored as a word of zeros.

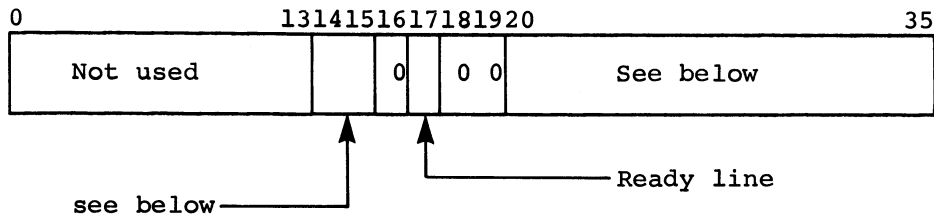
### STATUS INDIRECT CONTROL WORD (STATUS ICW)



where:

- Bits 0-2 = 001 Specifies an indirect 36 character address.
- Bits 3-17 = Y DATANET FNP memory address where status will be stored by the DIA.
- Bit 23 = E Signifies the tally (Z-field) has exhausted. This bit is set to one by hardware when the tally is decremented from one to zero. Once the E bit is set, the Y and Z fields are not incremented or decremented further on subsequent accesses.
- Bits 24-35 = Z Specifies the number of 36-bit status stores which can be made in the DATANET FNP.

### STATUS WORD FORMAT IN DATANET FNP MEMORY



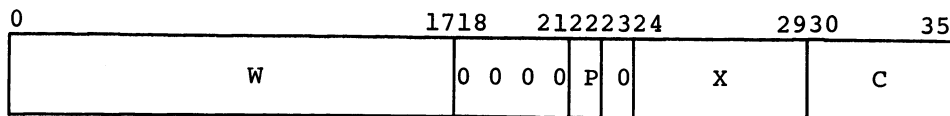
BIT	FUNCTION (INDICATION)
0-13	Zero (not used)
14	DIA internal parity error (Data transfers)
15	FNP software parity error (mailbox or DCW pair)
16	Zero
17	When equal to one, indicates the Central System
18	Direct Channel is in operational mode
19	Zero
20	Zero
21	Illegal connect from DATANET FNP
22	Illegal Opcode from DATANET FNP (or control word
23	Parity error if bit-15 equals one)
24	List ICW tally runout
25	DATANET FNP address field did not specify direct
26	36-bit address in control word
27	Address less than lower bound switches
28	Address greater than upper bound switches
29	Central System write inhibit violation
30	Central System test command while busy
31	Illegal Opcode from Central System
32	Central System not available
33	E bit set in List ICW
34	Parity error detected by DIA in Central System IOM
35	Parity error detected in Central System <sup>1</sup>
35	System fault detected in Central System IOM

<sup>1</sup>Status bits 32-34 indicate parity error in data transfers as follows:

- Bit 32 = C/S IOM to DCA
- Bit 33 = DCA to C/S IOM
- Bit 34 = DIA to DCA

Central System Control Word Formats

PERIPHERAL CONTROL WORD (PCW)



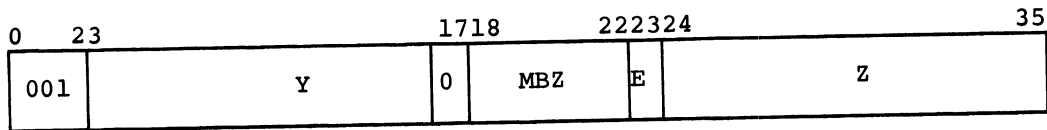
where:

Bit 22 = P Signifies the parity bit used by software to make this word contain an odd number of bits. Set to one for odd parity.

W,X,C are as follows:

COMMAND	C-FIELD (Opcode)	W-FIELD (C/S Address)	X-FIELD
INTERRUPT DATANET FNP (on level specified by SPECIAL INT. switch)	71	Not used	Specifies channel number in bits 26-29
BOOTLOAD FNP	72	Specifies Central System Boot ICW address	Extended address bits - A0-A5
INTERRUPT Central System	73	Not used	Specifies interrupt level no. in bits 27-29
TEST DATA Transfer to Central System	75	Specifies Central System Test ICW address	Extended address bits A0-A5
TEST DATA Transfer to DATANET FNP	76	Specifies Central System Test ICW address	Extended address bits A0-A5

TEST AND BOOTLOAD INDIRECT CONTROL WORD (Test ICW/Bootload ICW)



where:

Bits 0-2 = 001 Specifies a direct 36 character address  
Bits 3-17 = Y Specifies the starting address in DATANET FNP for data transfer. The starting address in the Central System is the word following this control word.  
Bit 23 = E Tally exhaust bit.  
Bits 24-35 = Z Specifies the number of 36-bit words to be transferred.

HIGH SPEED LINE ADAPTER (HSLA)

The High-Speed Line Adapter (HSLA) is a multilane communications controller with up to 32 concurrently operating lines. It handles both synchronous and asynchronous character-oriented communication terminals operating at various transmission rates -- 75 to 50,000 bits per second.

The HSLA controller bus provides the interface between DATANET FNP subchannel units. Data transfer between the controller and the character-buffered subchannels is parallel by character and between the subchannel and the terminals is in serial bit form.

General Information

I/O Channel - 6-10 (octal):

The channel assignment is hardware patchable.

HSLA Subchannel - 1000-1777 (HSLA #1)  
Comm. Regions - 2000-2777 (HSLA #2)  
3000-3777 (HSLA #3)

Interrupt Vectors:

Level 4 (all) - active subchannels 0-15  
Level 5 (all) - active subchannels 16-31  
Level 6 (all) - configuration subchannels 0-15  
Level 7 (all) - configuration subchannels 16-31

PCW Format

PCW BIT	COMMAND PCW0	COMMAND PCW1	CONFIGURATION PCW2 (ASYNCHRONOUS)	CONFIGURATION PCW3 (SYNCHRONOUS)	BISYNC. PCW3 (SYNCHRONOUS)
0-1	=00	=01	=10	=11	=11
2-5	COMMAND	COMMAND	COMMAND	COMMAND	
6	← (NOT USED) →				
7-11	SUBCHANNEL NUMBER (0-31)				
12				{ LPR (Lat. Parity receive) LPS (Lat. Parity send) LPO (Lat. Parity odd) ACW (Alt. Cont. Word) CCT (CCT enable) -- (Spare)                 }	
13					
14	NOT				
15	← →				
16	USED				
17					
18-23	← NOT USED BY HSLA →				
24		(Reserved for subchannel broadside cmds)	2 Stop Bits (IF=1) Not used	(Reserved for subchannel)	(24) If=1, CRC-16 polynomial; If=0, CCITT polynomial (27) If=1, EBCDIC code; If=0, ASCII code (28) If=1, Transp. data If=0, Non-Transp. data (29) If=1, Timer is enabled
25					
26					
27		RECEIVE MODE			
28	NOT	SEND MORE	110 bps	Bit 8 synchr. char.	
29		WRAPAROUND	134.5 bps	Bit 7 synchr. char.	
30		DATA TERMINAL READY	150 bps	Bit 6 synchr. char.	
31		REQUEST TO SEND	300 bps	Bit 5 synchr. char.	
32		MAKE BUSY	1050 bps	Bit 4 synchr. char.	
33	USED	SUPERVISORY SEND	1200 bps	Bit 3 synchr. char.	
34		CALL REQUEST	1800 bps	Bit 2 synchr. char.	
35		SPARE	75/600 bps (option)	Bit 1 synchr. char.	

Command PCW0, PCW1

Opcode 2 3 4 5	Command
0 0 0 0	No command sent. (req'd to send broadside commands in PCW1)
0 0 0 1	Subchannel input status request
0 0 1 0	Subchannel output status request
0 0 1 1	Subchannel configuration status request
0 1 0 0	Set subchannel mask bit (in mask register)
0 1 0 1	Reset subchannel mask bit (in mask register)
0 1 1 0	Switch subchannel Receive data buffer
0 1 1 1	Switch subchannel Send data buffer
1 0 0 0	Initialize (HSLA all subchannels)
1 0 0 1	Store mask register (in subchannel 0 ICW table, loc's 12 and 13)
1 0 1 0	(not used)
1 0 1 1	(not used)
1 1 0 0	Resync (Restart sync search and continue until sync is established)
1 1 0 1	Transmit line break (approx. 600 msec.)
1 1 1 0	(not used)
1 1 1 1	(not used)

Command PCW2, PCW3

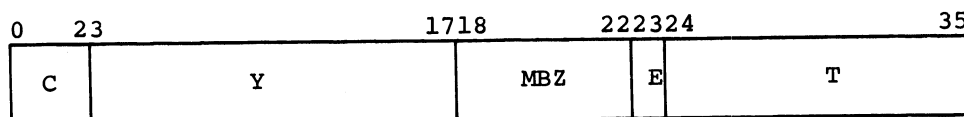
Opcode 2 3 4 5	Command
1 1 0 0	5-bit character (asynchronous)
1 1 0 1	6-bit character (asynchronous)
1 1 1 0	7-bit character (asynchronous)
1 1 1 1	8-bit character (asynchronous)

Character length is ignored for BSC subch. It is wired for 8 bits per character.

Control Words

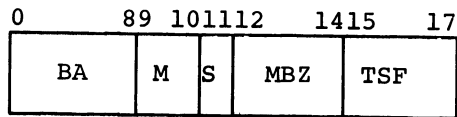
Relative address (octal)	Function
0-1	Receive ICW primary
2-3	Receive ICW secondary
4-5	Send ICW primary
6-7	Send ICW secondary
10-11	Base Address used (word 10) and spare (word 11)
12-13	Mask register (subch. 0 only, not used elsewhere)
14-15	Active ICW, status
16-17	Config. status mailbox

Indirect Control Word



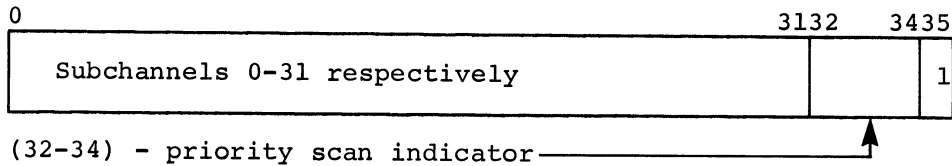
C - Character position  
Y - Absolute memory address  
E - Exhaust bit  
T - Tally

Base Address Word



- BA (0-8) - base address (of CCT)
- M (9-10) - modifier
- S (11) - short table indicator
- MBZ(12-14) - must be zero
- TSF(15-17) - table switch field (for switching tables in transparent mode)

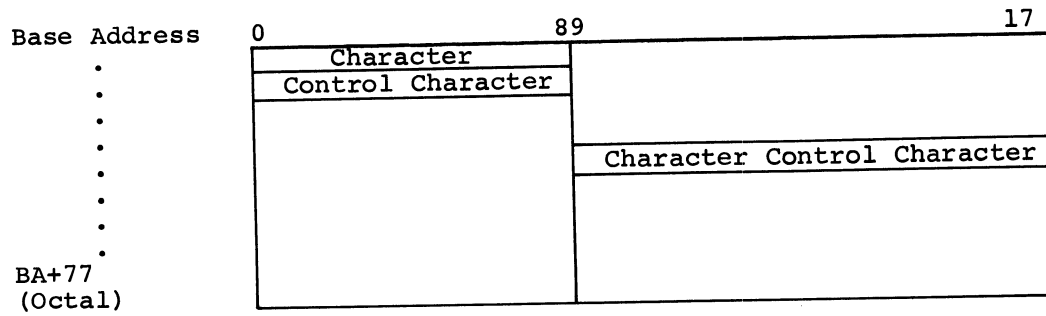
Mask Register Word



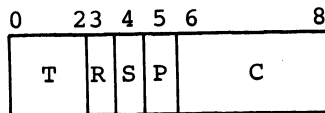
Control Word Memory Map (Example for Channel 06)

ICW Address octal	Subchannel Number		Interrupt Vector Locations	
	decimal	octal	active	configuration
01000-01017	0	0	00004	00006
1020- 1037	1	1	024	026
1040- 1057	2	2	044	046
1060- 1077	3	3	064	066
1100- 1117	4	4	104	106
1120- 1137	5	5	124	126
1140- 1157	6	6	144	146
1160- 1177	7	7	164	166
1200- 1217	8	10	204	206
1220- 1237	9	11	224	226
1240- 1257	10	12	244	246
1260- 1277	11	13	264	266
1300- 1317	12	14	304	306
1320- 1337	13	15	324	326
1340- 1357	14	16	344	346
1360- 1377	15	17	364	366
1400- 1417	16	20	005	007
1420- 1437	17	21	025	027
1440- 1457	18	22	045	047
1460- 1477	19	23	065	067
1500- 1517	20	24	105	107
1520- 1537	21	25	125	127
1540- 1557	22	26	145	147
1560- 1577	23	27	165	167
1600- 1617	24	30	205	207
1620- 1637	25	31	225	227
1640- 1657	26	32	245	247
1660- 1667	27	33	265	267
1700- 1717	28	34	305	307
1720- 1737	29	35	325	327
1740- 1757	30	36	345	347
1760- 1777	31	37	365	367
Channel 07 = 2000 - 2777				
Channel 10 = 3000 - 3777				

## Character Control Table



The Character Control Character (CCC) has the following format:



T = Table switch field  
R = Resync  
S = Switch buffers  
P = Parity inhibit  
C = Command

The Command field can be as follows:

000 = normal character, store  
001 = terminate+1, store  
010 = terminate+2, store  
011 = terminate non-store  
100 = marker, set status bit only, store  
101 = marker, interrupt+1, store  
110 = marker, no store  
111 = marker, interrupt non-store

## Character Control Character Addressing

Figure 5-1 contains a block diagram of Character Control Character addressing.



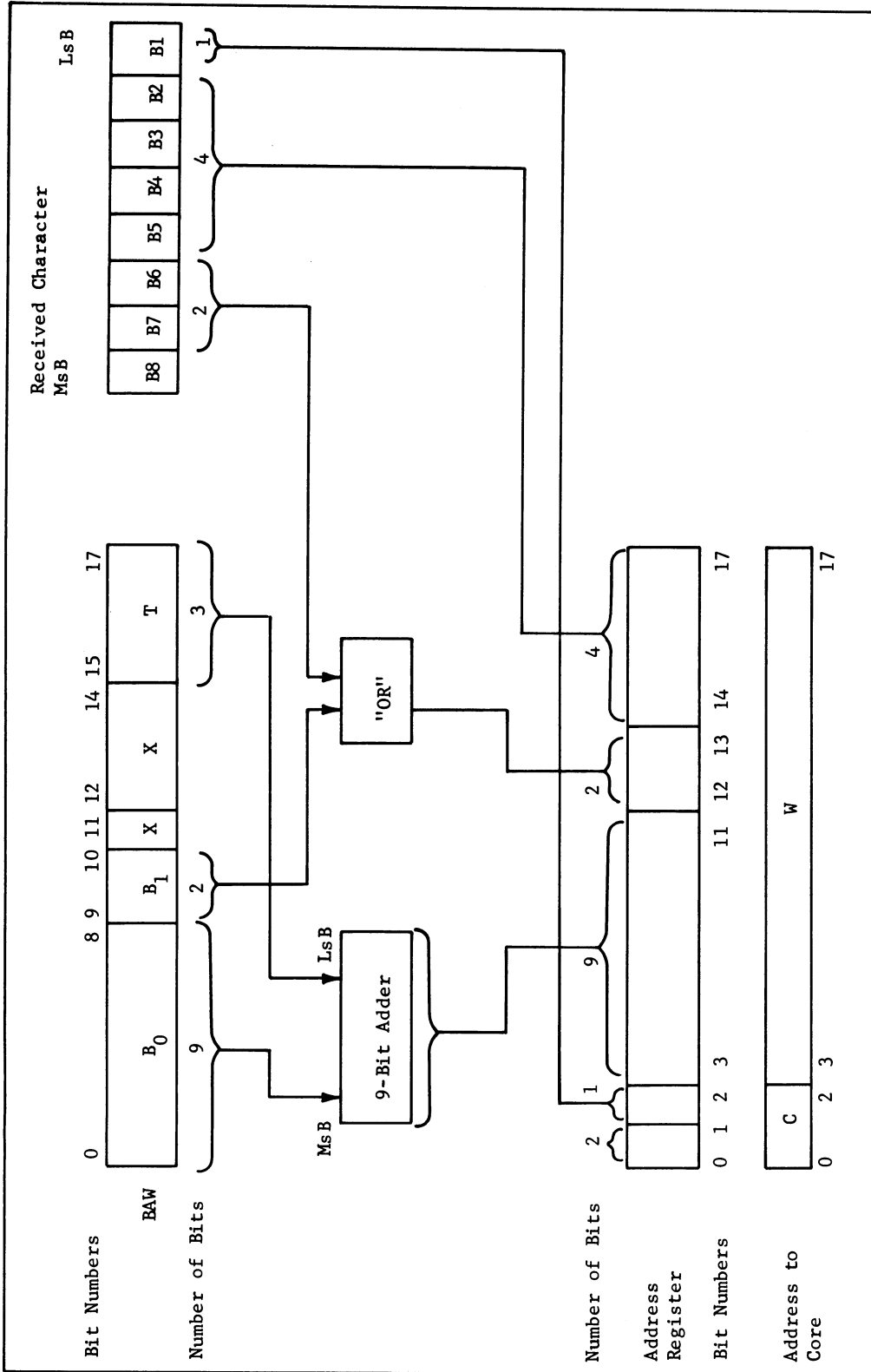


Figure 5-1. Character Control Character Addressing

Status

Status words for active and configuration status are as follows:

BIT	ACTIVE STATUS	CONFIGURATION	
0	0 = send, 1 = receive	= 1 (configuration)	
1	<sup>1</sup> normal marker character received	sync. (if=1), async. (if=0)	
2	<sup>1</sup> delayed marker character received	spare	
3	<sup>1</sup> terminate character received	} See Subchannel type	
4	alternate buffer is active (if=1)		
5	switch buffers after status store (if=1)		
6	TYO (if = 1)		
7	TY1 (if = 1)		
8	<sup>1</sup> lateral parity error		
9	<sup>1</sup> cmd. sent to unimplemented subch.		
10	<sup>1</sup> change in date set status occurred		
11	(spare - MBZ)		
12	transfer timing error (if = 1)		
13	(spare - MBZ)		
14	(spare - MBZ)		
15	No stop bit received (async only)		(spare)
16	DLO (data line occupied ACU)		(spare)
17	PWI (power indicator - ACU)	(spare)	
18	data set ready	(spare)	
19	clear to send	(spare - MBZ)	
20	carrier detect	(spare - MBZ)	
21	supervisory receive	5-bit character (async. only)	
22	ACR (abandon call and retry-ACU)	6-bit character	
23	data set status lead up (ACU)	7-bit character	
24	ring indicator	8-bit character	
25	line break	use two stop bits (if bit 1=1)	
26	(spare)	(spare)	
27	receive mode	(spare)	
28	send mode	(spare)	
29	wraparound mode	} See Configuration Status	
30	data terminal ready		
31	request to send		
32	make busy		
33	supervisory transmit		
34	call request (ACU)		
35	(spare - reserved for subchannel)		

<sup>1</sup>Used with receive status only

The Channel Types are:

Bit 3 4 5 6 7 8	Subchannel Type
0 0 0 0 0 0	Illegal
0 0 0 0 0 1	General purpose S/C
0 0 0 0 1 0	General purpose S/C (with ACU)
0 0 0 0 1 1	Dual synchronous ASCII S/C
0 0 0 1 0 0	Dual synchronous ASCII S/C (with ACU)
0 0 0 1 0 1	Dual asynchronous S/C (EIA)
0 0 0 1 1 0	Not to be used (reserved for SLA 355)
0 0 0 1 1 1	Dual asynchronous S/C (VCA)
0 0 1 0 0 0	General purpose S/C (MIL188)
0 0 1 0 0 1	Wideband S/C (TELPAC)
0 0 1 0 1 0	High level data link control S/C (ADCCP)
0 0 1 0 1 1	Dual synchronous S/C (MIL188)
0 0 1 1 0 0	Bi-synchronous S/C (BSC)
0 0 1 1 0 1	Voice answerback S/C (generator)
0 0 1 1 1 0	Voice answerback S/C (receiver)
0 0 1 1 1 1	Spare
0 1 0 0 0 0	Dual asynchronous S/C (MIL188)
0 1 0 0 0 1	High level data link control S/C (ADCCP) (wide band)
0 1 0 0 1 0	Spare
1 1 1 1 1 1	

The Configuration Status is:

Configuration Status Bit	Asynchronous	Synchronous
28	110 bps	bit 8
29	134.5 bps	bit 7
30	150 bps	bit 6
31	300 bps	bit 5
32	1050 bps	bit 4
33	1200 bps	bit 3
34	1800 bps	bit 2
35	75/600 bps	bit 1

} sync character

BINARY SYNCHRONIZATION STATUS

BSC Config Status Bit	Status
0	Must = 1 (sync.)
3-8 (octal)	Subchannel type = 14
24	1 = CRC-16 Polynomial 0 = CCITT Polynomial
27	1 = EBCDIC Code 0 = ASCII Code
28	1 = Transparent Data 0 = Non-Transparent Data
29	1 = Timer Enabled

LOW SPEED LINE ADAPTER (LSLA)

The Low-Speed Line Adapter (LSLA) is a communications controller that provides time division multiplexing, developing a message frame composed of a number of 8-bit characters. It handles up to 52 low-speed terminals operating at speeds up to 110 bits per second, or 26 terminals at speeds up to 150 bits per second, or 17 terminals at speeds up to 300 bits per second. Terminals with different transmission speeds can be mixed on a single LSLA.

General Information

- I/O Channel (through LSLA channel card) - 11-16 (octal)
- Interrupt Vectors
 

Channel Number	11	12	13	14	15	16
IOM Detected Error	220	240	260	300	320	340
Active	221	241	261	301	321	341
Configuration	222	242	262	302	322	342
- Control Words (character control does not apply) (see Note 1)
  1. PCW's
  2. ICW's
- Character Control - does not apply

- Status (via status ICW's) (see Note 2)

NOTES: 1. LSLA uses PCW0 and PCW1 only, as follows:

PCW0: bits 0-5 and 23 are read.

PCW1: bits 0-5, 23, and 27-31 are read.

2. Status same as HSLA status, except as follows:

CONFIGURATION STATUS - only bits 0, 3-8, 15, 23, 28-35 are used. Channel type (bits 3-8) is coded 06.

ACTIVE STATUS - only bits 0, 4-7, 10, 12, 18-20, 27-31 are used.

Control Words

Relative Address (octal)	Function
0-1	Primary Receive ICW
2-3	Secondary Receive ICW
4-5	Primary Send ICW
6-7	Secondary Send ICW
10-13	Not Used
14-15	Active Status ICW
16-17	Configuration Status Mailbox

Channel Number	Control Word Block Address Range (octal)
11	00500 - 00517
12	00520 - 00537
13	00540 - 00557
14	00560 - 00577
15	00600 - 00617
16	00620 - 00637

PCW Format

PCW Bit	Command PCW0 Active	Command PCW1 Active-Broadside
0-1	00	01
2-5	See Command PCW0	See Command PCW1
6-22	Not used	Not used
23	Mask bit	Mask bit
24-26	Not used	Not used
27	Not used	Receive mode
28	Not used	Send Mode
29	Not used	Wraparound
30	Not used	Data Terminal Ready
31	Not used	Request to send
32-35	Not used	Not used

Commands PCW0, PCW1

Octal	Opcode				Command
	2	3	4	5	
0	0	0	0	0	No command sent (Req'd to send broadside commands in PCW1)
1	0	0	0	1	Subchannel input status request
2	0	0	1	0	Subchannel output status request
3	0	0	1	1	Subchannel configuration status request
4	0	1	0	0	(not used)
5	0	1	0	1	(not used)
6	0	1	1	0	Switch subchannel receive data buffer
7	0	1	1	1	Switch subchannel send data buffer
10	1	0	0	0	Initialize
11	1	0	0	1	(not used)
12	1	0	1	0	(not used)
13	1	0	1	1	(not used)
14	1	1	0	0	Resync (Restart sync search and continue until sync is established)
15	1	1	0	1	(not used)
16	1	1	1	0	(not used)
17	1	1	1	1	(not used)

Indirect Control Word

0	23	1718	222324	35
C	Y	MBZ	E	T

C - Character position  
Y - Absolute memory address  
E - Exhaust bit  
T - Tally

Status

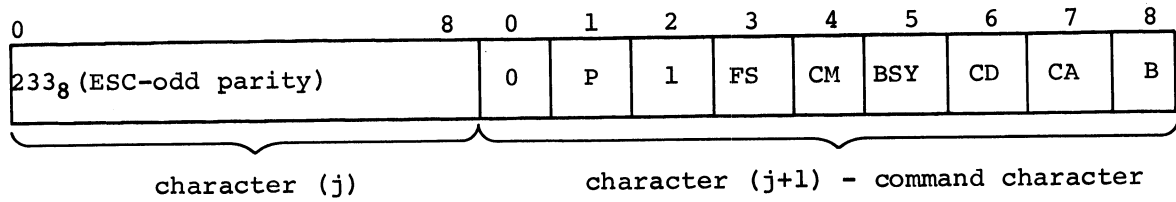
BIT	ACTIVE STATUS	CONFIGURATION
0	0 = send, 1 = receive	=1 (configuration)
1	MBZ	Sync. (=1)
2	MBZ	MBZ
3	MBZ	0
4	Alternate buffer is active (if=1)	0
5	Switch buffers after status store (if=1)	0 Subchannel type (06)
6	TY0 (if=1)	1
7	TY1 (if=1)	1
8	MBZ	0
9	MBZ	MBZ
10	Data set status change (Receive only) <sup>1</sup>	MBZ
11	MBZ	MBZ
12	Transfer timing error (if=1)	MBZ
13	MBZ	MBZ
14	MBZ	MBZ
15	MBZ	Two send ICWs (=1)
16	MBZ	MBZ
17	MBZ	MBZ
18	Data set ready	MBZ
19	Clear to send	MBZ
20	Carrier detect	MBZ
21	MBZ	MBZ
22	MBZ	MBZ
23	MBZ	8-bit character (=1)
24	MBZ	MBZ
25	MBZ	MBZ
26	MBZ	MBZ
27	Receive mode	MBZ
28	Send mode	0
29	Wraparound mode	0
30	Data terminal ready	0
31	Request to send	1 ASCII "SYN"
32	MBZ	0 character 026 (octal)
33	MBZ	1
34	MBZ	1
35	MBZ	0

<sup>1</sup>Rules for Data Set Status change interrupt:

If Data Set Ready changes state; or if Data Set Ready is sent, and either Clear to Send or Carrier Detect changes state; an Active status interrupt occurs and Receive status is stored with bit 10 set to one.

Command Characters in DATANET FNP Memory

Type I - device control (no ACU):

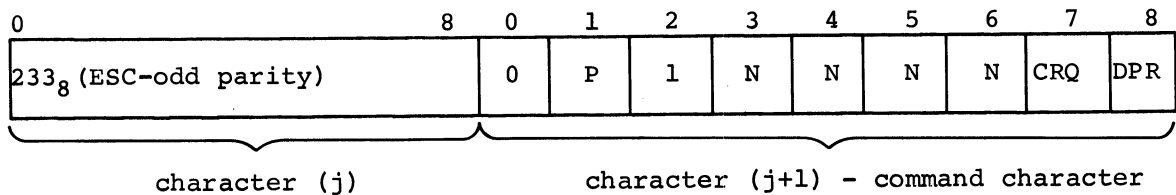


Command character decode:

- P - odd parity
- CD - data terminal ready
- CM - answer control (for Bell 103E)
- CA - request to send
- BSY - busy
- FS - frequency select
- B - line break transmit

The command consists of ESC (odd parity) and the next "non-fill" character. Fill characters (037 octal) may occur between the ESC and the command character.

Type I - device control (with ACU):

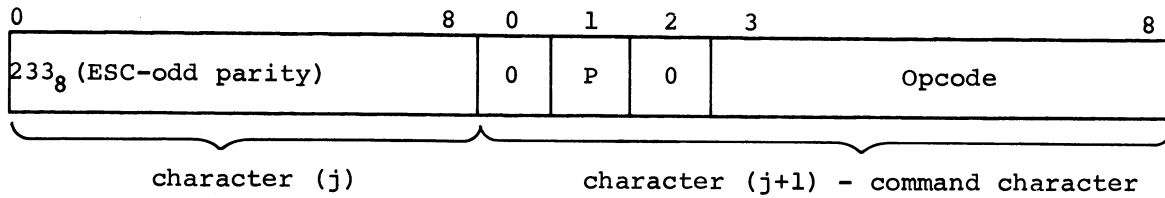


Command character decode:

- P - odd parity
- NNNN - number bits for ACU (binary)
- DPR - digit present
- CRQ - call request



Type II - special control:

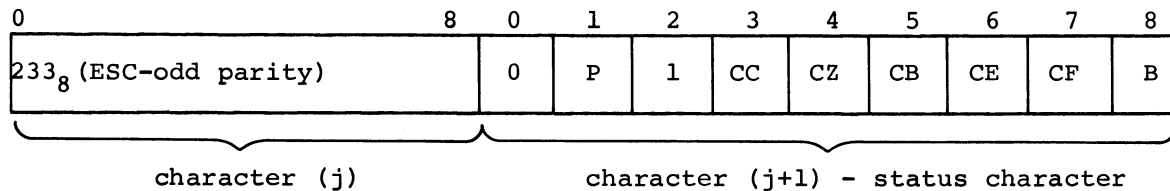


Command character decode (P = odd parity):

- 40 - Error count command<sup>2</sup>
- 41 - Spare
- 44 - Low speed wraparound reset
- 45 - Low speed wraparound set<sup>1</sup>
- 50 - High speed wraparound<sup>1,2</sup>
- 51 - Configuration mode cmd<sup>1,2</sup>
- 54 - Disable protect<sup>2</sup>
- 55 - Channel status request

Status Characters in DATANET FNP Memory

Type I - channel modem status (no ACU):



Status character decode:

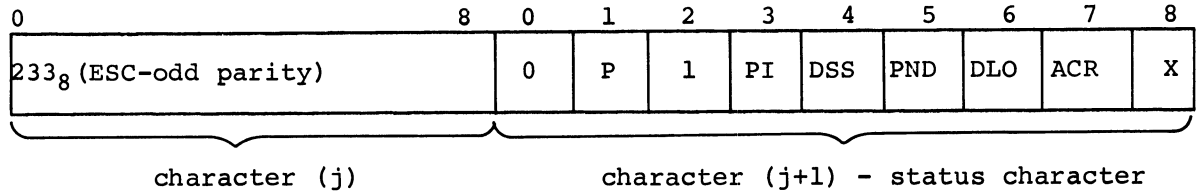
- P - odd parity
- CC - data set ready
- CZ - restraint
- CB - clear to send
- CE - ring
- CF - carrier detect
- B - line break

---

<sup>1</sup>Disable protect required preceding this command. For example: ESC, DAP, ESC, CONFIG.

<sup>2</sup>Command must be sent in T and D time slot.

Type I - channel status (with ACU):

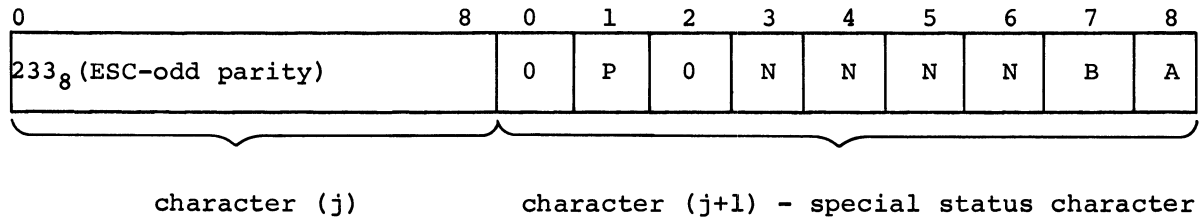


Status character decode:

- P - odd parity
- PI - power indicator
- PND - present next digit
- DSS - data set status
- ACR - abandon call - retry
- DLO - data line occupied
- X - not used except by T and D

These commands consist of ESC (odd parity) and the next "non-fill" character. Fill characters (037 octal) may occur between the ESC and the command character.

Type II - special status (via T and D channel)



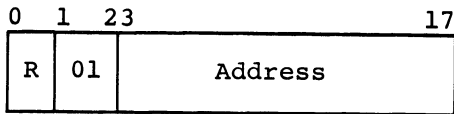
Status character decode:

- P - odd parity
- NNNN - error count (binary)
- BA - (=00) for error count





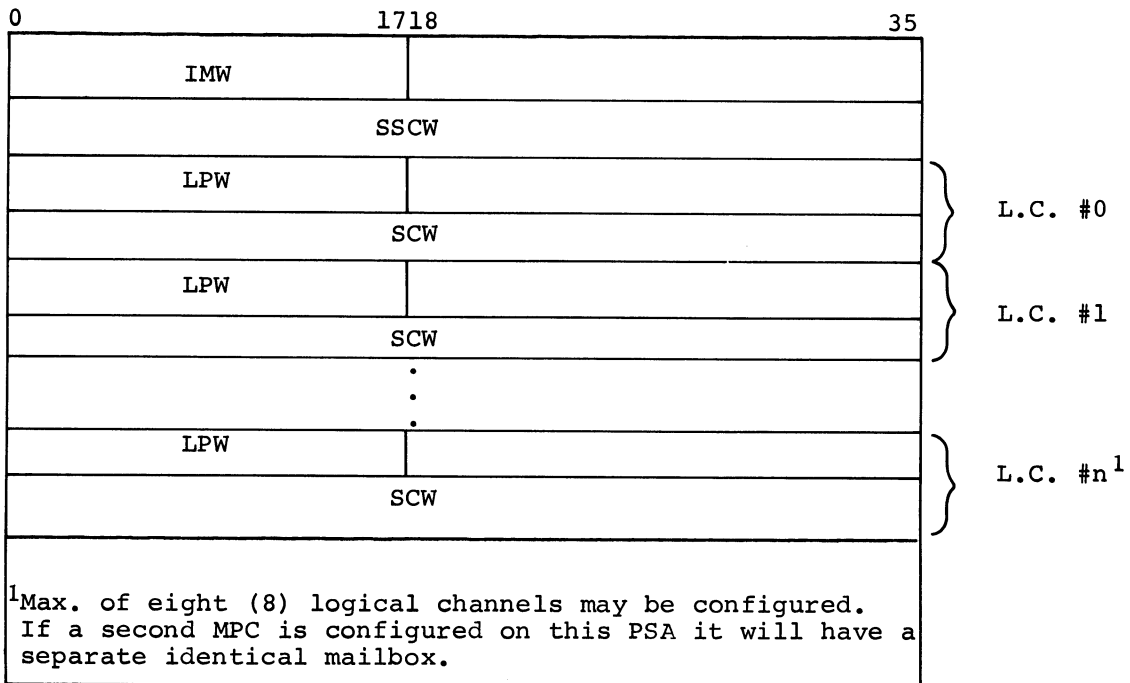
LIST POINTER WORD (LPW)



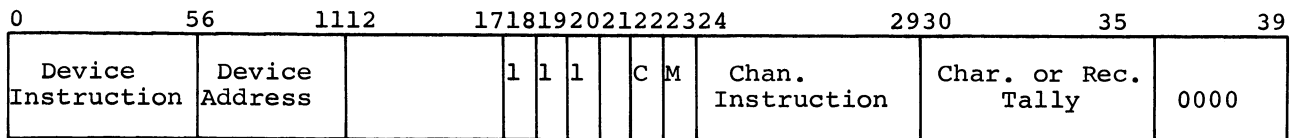
Bit 0 (Restricted) - When set to 1 restricts DCW List Services to one IDCW.

PSA MAILBOX

- Address in Base Address Register points to starting location of mailbox area.



INSTRUCTION DATA CONTROL WORD (IDCW)



Byte 0  
(0-7)

Byte 1  
(8-15)

Byte 2  
(16-23)

Byte 3  
(24-31)

Byte 4  
(32-39)

Bits 0-5 - MPC System Command

Bits 6-11 - Device Code

First IDCW will establish Device Code. Subsequent IDCW Device Codes will be ignored. This makes it impossible for a program to switch devices within a list.

Bits 12-17 - Ignored

Bits 18-20 - Must be on. (This distinguishes DCW from an IDCW.)

Bit 21 - Ignored

Bit 22 - Continue

Indicates this is not last IDCW in list. Upon completion of this IDCW, a "Move Pointer" Service Code will be issued to obtain new IDCW.

Bit 23 - Marker Bit (Ignored if Continue Bit (22) not on.) Upon completion of IDCW, MPC will issue Service Codes to Store Marker Status and Set Marker Interrupt. MPC will then issue a Move Pointer Service Code to obtain a new IDCW.

Bits 24-29 - Channel Instruction

Must be the following:

- 00 - Unit Record Transfer
- 02 - Peripheral Action (No Data transfer: e.g., Request Status, Release, Restore)
- 2(X) - Command Extension Modifiers
- 4(X) - Special Controller Commands

If 2(X):

- 21 - Inhibit Automatic Retry
- 22 - Inhibit Alternate Track Logic and E-D-C Logic
- 23 - Special Permission Execution
  - o Read Override RPS Queue
  - o Write Override RPS Queue
- 24 - EDAC Override (190); Check Char. Override (181)
- 25 - Read and perform error correction on data before transferring data to EUS (PSA).

If 4(X):

- 40 - Special Controller Command Device Address Field Bits 6-11) MBZ.

Bits 30-35 - Record Tally or Character

Contains number of times device instruction is to be reissued by Controller. (Chan. Inst. - 02)

DATA CONTROL WORD (DCW)

0	23		17	18	21	22	23	24	32	34	35
001	Data Address	0	MBZ	Cmd.	Tally						

Bits 22 23 Identify Type of DCW

- 0 0 = IOTD = I/O Transfer and Disconnect
- 0 1 = IOTP = I/O Transfer and Proceed
- 1 0 = TDCW = Transfer DCW
- 1 1 = IONTTP = I/O Non-transfer and Proceed

LOGICAL CHANNEL DCW LIST

Separate DCW lists for each logical channel configured.

0	1718	35
IDCW		
DCW		
DCW		
DCW		
.		
.		
.		
.		
DCW		

- Logical Channel LPW in PSA mailbox points to starting address of DCW list.

STATUS WORD FORMATS

Terminate/Marker Status - First Word (5 Bytes from MPC)

0	1	2	56	1112	15161718	2324	2930	35	39
T	P	Major Status	Substatus	Software Status	I A	IOM/Chan. Status	MBZ	Record Residue	0000

Byte 0 (0-7)	Byte 1 (8-15)	Byte 2 (16-23)	Byte 3 (24-31)	Byte 4 (32-39)
-----------------	------------------	-------------------	-------------------	-------------------

- Bit 0 - Set by MPC when status stored (Software flag).
- Bit 1 - Power Bit ( = 1 if MPC power off).
- Bit 2-5 - MPC Major Status
- Bit 6-11 - MPC Substatus
- Bit 12-15 - Software Status;  
Stored as zeros by MPC. Can be used by software to indicate software detected errors to Slave program after hardware has stored the Status Word.
- Bit 16 - Initiate Interrupt  
Status stored during Instruction Sequence; No Data Transfer.
- Bit 17 - Abort Bit  
Stored as "0" by MPC. Set to 1 by software if this transaction caused program to abort.

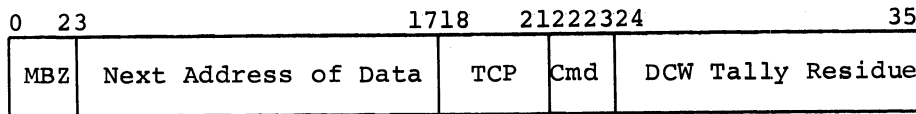
Bit 18-23 - IOM/Channel Status  
 Divided into two independent three bit segments:

Bits 18-20 = Channel detected user faults  
 21-23 = IOM detected user faults reported to PSA

Bits 24-29 - MBZ

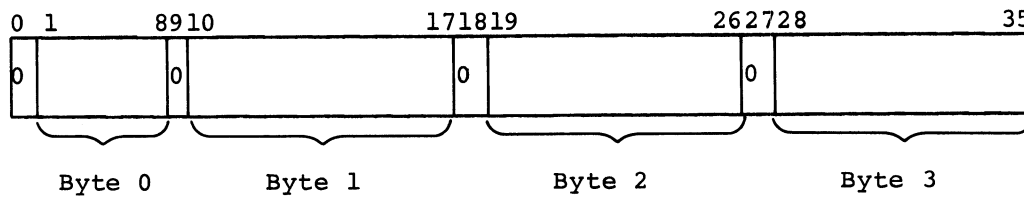
Bits 30-35 - Residue Record Count  
 (Illegal for disk subsystem. Will always be stored as zeros.)

Terminate/Marker Status - Second Word (36 Bits from PSA)



Bit 0-2 - MBZ  
 Bit 3-17 - Contents of DCW Register  
 Bit 18-21 - Terminate Character Position Code  
 Bit 22-23 - DCW Cmd.  
 Bit 24-35 - Contents of Tally Register

Special Status (ASCII Format)



Bits 1-8 (Byte 0) - MPC sends this character all zeros.  
 Bits 10-17 (Byte 1) - MPC Device # (MPC Controller is device #0).  
 Bits 19-26 (Byte 2) } Type of Special Interrupt:  
 Bits 28-35 (Byte 3) }     01 - Pack Change  
                                   02 - Device Released  
                                   03 - Power off

NOTE: The Special Status Store will be followed by a Special Interrupt (Level 1).



PSA ERROR SUMMARY

FAULTS	ACTION	FAULT BYTE SENT TO MPC <sup>2</sup>							Meaning		
		0	1	2	3	4	5	6		7	
Data Parity Error, Internal PSIC	Terminate later if Data Service.  Terminate now if not Data Service.	1	0	0	1	0	0	0	0	0	Terminate
Connect while busy (Unexpected PCW)	Terminate when LC is active.	1	0	0	0	1	0	0	0	0	
Data Parity Error. PSI	Terminate later.	1	0	1	1	0	0	0	0	0	
Parity error, <sup>1</sup> FNP IOM to or from PSIC treated as a FNP IOM detected fault.											
Illegal Service Code	Terminate now.	0	1	0	0	0	0	0	0	0	
Parity error during Service Code sequence.	Terminate now.	0	1	0	0	0	0	0	0	0	
Illegal DCW	Terminate now.	1	0	0	1	1	0	0	0	0	Terminate
Service Code for not busy or masked logical channel.	Terminate now.	1	1	0	0	0	0	0	0	0	Abort
(PSIC is masked for these conditions)											
FNP IOM Detected Fault	Terminate now.	1	1	0	0	0	0	0	0	0	
Masking PCW	Terminate when LC # is active.	1	1	0	0	0	0	0	0	0	Abort
Tally Runout with Terminate from MPC	Termination in process.	0	0	0	0	0	0	0	0	0	PSIC Tally Runout

<sup>1</sup>Detected on FNP AB1 and 6600 Models

<sup>2</sup>Fault Byte - Bits 0 and 1

00 - Special IOM Instruction

01 - PSA Detected Illegal SVC or P.E. (MPC should Retry.)

10 - Status Store and Terminate cycle should follow.

11 - L. C. Masked - Abort (No Status and Term. cycle)

SERVICE CODES - MPC TO PSA

<u>MEANING</u>	<u>DATA BITS</u>		<u>HEXI-</u>
	<u>0123</u>	<u>4567</u>	<u>DEC.</u>
Initiate new channel program.	0000	0001	1
Move pointer and initiate Command Transfer.	0000	0010	2
Backup pointer and initiate Command Transfer.	0000	0011	3
Data Transfer, Read Binary	0000	0100	4
Data Transfer, Read ASCII	0000	0101	5
Data Transfer, Write Binary	0000	0110	6
Data Transfer, Write ASCII	0000	0111	7
Store Special Status	0000	1000	8
Store Terminate Status	0000	1001	9
Set Terminate Interrupt	0000	1101	D
Set Marker Interrupt	0000	1110	E
Set Special Interrupt	0000	1111	F
All other codes are illegal.			

SERVICE CODES - PSA TO MPC

Connect	0010	0000
Disconnect	0001	0000

## MPC COMMANDS

Sent to MPC in Bits 0-5 of IDCW.

<u>OCTAL</u>	<u>HEX.</u>	<u>DESCRIPTION</u>
00	00	Request Status
04	04	Read Non-Standard
10	08	Boot C/S
11	09	Boot ITR
16	0E	Write Control Reg.
17	0F	Format
21	11	Read EDAC
22	12	Read Status Register
23	13	Read ASCII
25	15	Read
26	16	Read Control Register
27	17	Read Header
30	18	Execute DLI
31	19	Write
32	1A	Write ASCII
33	1B	Write and Compare
34	1C	Seek
36	1E	Special Seek
37	1F	Preseek
40	20	Reset Status
42	22	Restore
72	3A	Set Standby
76	3E	Release
77	3F	Reserve

SPECIAL CONTROLLER COMMANDS

00	Suspend
02	Read Mem. ASCII
04	Read Lock
06	Initiate Read
12	Write Mem. ASCII
14	Write Lock
16	Initiate Write
20	Release
22	Read Mem. Binary
32	Write Mem. Binary
34	Cond. Write Lock

MPC DEVICE STATUS

Major and Substatus fields of first Status Word.

MAJOR  
STATUS            SUBSTATUS  
2345678 9 10 11

0000 - READY

    000 000 No Substatus  
    000 0XX Retrys  
    001 0XX Device in T and D

0001 - BUSY

    000 000 Positioning  
    100 000 Alternate Channel

0010 - ATTENTION

    000 001 Write Inhibit  
    000 010 Seek Incomplete  
    001 000 Device Fault  
    010 000 Device in Standby  
    100 000 Device Off-Line

0011 - DATA ALERT

    000 001 Transfer Timing  
    000 010 Parity  
    000 100 Invalid Seek Addr.  
    0X1 000 Header Verification  
    X1X 000 Cyclic Check  
    1X0 000 Compare Alert

0100 - END-OF-FILE

    000 000 Good Track  
    000 0X1 Last Consec. Block  
    000 01X Block Count Limit  
    000 100 Def. Track-Alt. Assg.  
    001 000 Def. Track-No Alt.  
    010 000 Alt. Track Det.

0101 - INSTRUCTION REJECT

000 001 Invalid Opcode  
000 010 Invalid Device Code  
000 100 IDCW Parity  
001 000 Inv. Inst. Seq.

1010 - MPC DEVICE ATTENTION

000 001 Config. Error  
000 010 Multiple Device  
000 011 Device No. Error  
001 011 CA Error and OPI Down  
001 100 Alert EN-1  
001 110 CA Alert (No EN-1)

1011 - MPC DEVICE DATA ALERT

000 001 Transmission Parity  
000 010 Inconsistent Cmd.  
000 011 Sum Check Error  
000 100 Byte Lockout  
001 110 EDAC Parity  
010 001 Sector Size Error  
010 010 Non-Standard Sect. Size  
010 011 Search Alert (#1st)  
010 100 Cyclic Code (#1st)  
010 101 Search Alert (#1st)  
010 110 Sync Byte #Hex 19  
011 010 EDAC Corr. #Last Sect.  
011 011 EDAC Corr. -B. C. L.  
011 100 EDAC Uncorr.

1101 - MPC COMMAND REJECT

000 001 Illegal Procedure  
000 010 Illegal L. C. #  
000 011 Illegal Suspend  
000 100 Continue Bit Not Set

IOM/CHANNEL STATUS

Bits 18-23 of first Status Word

18 19 20

0 0 1 Connect While Busy  
0 1 0 PSA Internal P. E.  
0 1 1 Illegal DCW  
1 1 0 Transmission P. E.

21 22 23

0 0 0 (Not Used)

DOCUMENT HANDLER CHANNEL (DHC)

General Information

The Document Handler Channel interfaces document reader/sorters to the FNP IOM. The reader/sorter may be one of four types used primarily in the banking industry.

Model Number	Description
4WDHC 600 AC1	Document Handler Ch. MRS200 or DRD200
4WDHC 601 AA1	Document Handler Ch. DRD236 or DHU1600

DHC Comm. Region Description

Chan. No.	Option	Device	Term. Status ICW (Base)	Data ICW (+2)	Q Status ICW (+4)	Int. Vec. (Term.)
12	DHC	Doc. Hdlr. #1	540	542	544	242
13	DHC	Doc. Hdlr. #2	550	552	544	262
14	DHC	Doc. Hdlr. #3	560	562	564	302
15	DHC	Doc. Hdlr. #4	570	572	574	322
11	DHC	Doc. Hdlr. #5	600	602	604	222
7	DHC	Doc. Hdlr. #6	610	612	614	162

- The options above have the capability of reading MICR or OCR characters.

MICR (4 Bit Char.) - Magnetic Ink Character Recognition

OCR (7 Bit Char.) - Optical Character Recognition

- The DHC provides the only method of inputting OCR or MICR information from documents to Central System.

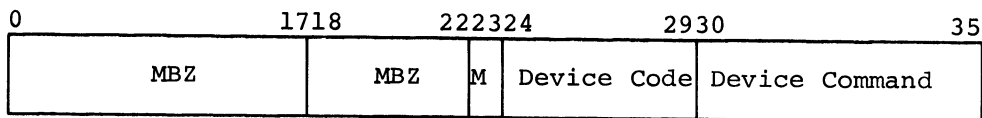
Reader/Sorter Characteristics:

- MRS200 - MICR (Magnetic Ink Character Recognition) Reader/Sorter
  - E13B MICR Font
  - 1200 Documents per minute
  - 12 pockets
- DRD200 - Optical Document Reader/Sorter
  - COC5 Optical Bar Font
  - Mark Sense Recognition (Optional)
  - 1200 Documents per minute
  - 2 pockets

- DRD236 - High-speed MICR Reader/Sorter
  - E13B MICR Font
  - 1625 Documents per minute
  - Up to 32 pockets
- DHU1600- Same as DRD236

MRS200/DRD200

PERIPHERAL CONTROL WORD (PCW)



where:

M = Mask (logical "1" Masks)  
 Device Code used for Pocket Select  
 Device commands are:

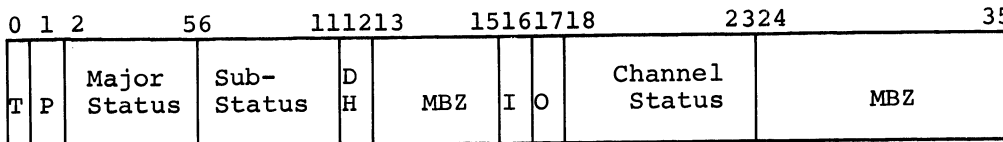
Operation	Octal Code	Device Code
Feed Continuous	41	00
Read Document	01	00
Pocket Document	43	XX
Stop Feed	61	00
Request Status	00	00
Reset Status	40	00
Remote Manual Halt	62	00

XX = Device Code pocket selection. Table shows order in which the pockets are arranged on the Reader Sorter.

Device Code	Pocket
12	Special Sort
00	0
01	1
02	2
03	3
04	4
05	5
06	6
07	7
10	8
11	9
13	Reject

- Initiate Status (Bit 16) - ICW Base Address + 4 (No Interrupt)
- Configuration and patch plug for channel number and ICW address is installed on the FNP backpanel pins associated with the LA WWB.

STATUS WORD FORMAT



where:

T (Terminate Bit) - A one means Peripheral went busy after command or status was stored.

P (Power Bit) - A one means no power on Peripheral.

I (Initiate Bit) - A one means Peripheral did not go busy after command.

Channel Status:

- 000000 - Normal (No Errors)
- 000001 - Transmission Parity Error
- 000010 - Tally Runout Error
- 000100 - Channel Detected Peripheral Fault
- 010000 - Connect While Busy
- 100000 - Bus Write Parity (AB1) not used on DHC
- DH - Must be a one for Document Handler Channel (Echo Bit)

Major Status and Substatus

Major Status	Substatus										
	2	3	4	5	6	7	8	9	10	11	
Ready	0	0	0	0	0	0	0	0	0	0	
Attention	0	0	1	0	0	0	X	X	X	1	Feeder/Pocket Alert
					0	0	X	X	1	X	Manual Halt
					0	0	X	1	X	X	Document Jam
					0	0	1	0	0	0	Feed Alert
					0	1	0	0	0	1	Last Batch
Data Alert	0	0	1	1	0	0	0	0	0	1	Transfer Timing Alert
					X	X	X	X	1	X	Multiple Feed
					X	X	X	1	0	X	Late/No Read Command
					X	X	1	X	0	X	No Pocket Command
					X	1	X	X	0	X	TCD Alert



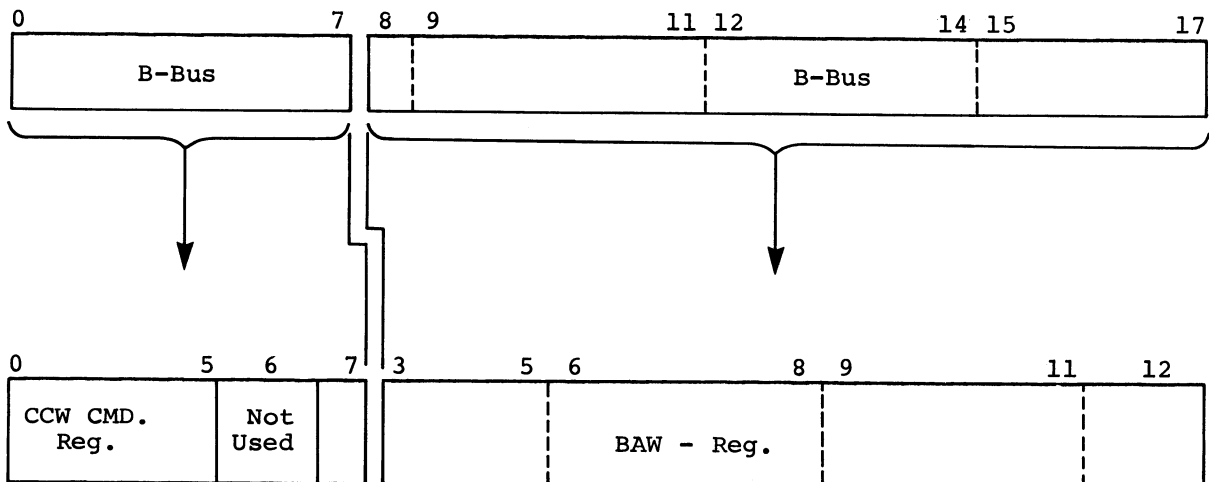
Major Status		Substatus			
Command	0 1 0 1	X 0 0 0	X 1	Invalid Opcode	
Reject		X 0 0 0	1 X	Invalid Device Code	
Special Substatus: Document committed		1 X X X X X			
Other Status Conditions in Bit (0-17) Special Echo Line (Bit 12)					

DRD236/DHUL600

- DHC601 connects the Reader/Sorter to FNP
- Requires computer clock pulse (SCCP) furnished by oscillator on IOM Backpanel (4-Megahertz)
- Uses a base address word (BAW) to reference a control character word (CCW) table.
- CCW used to control Data Store, Q-Status Word and Q-Status Store. (Q symbols)
- BAW loaded by LDEX instruction prior to issuing connect.

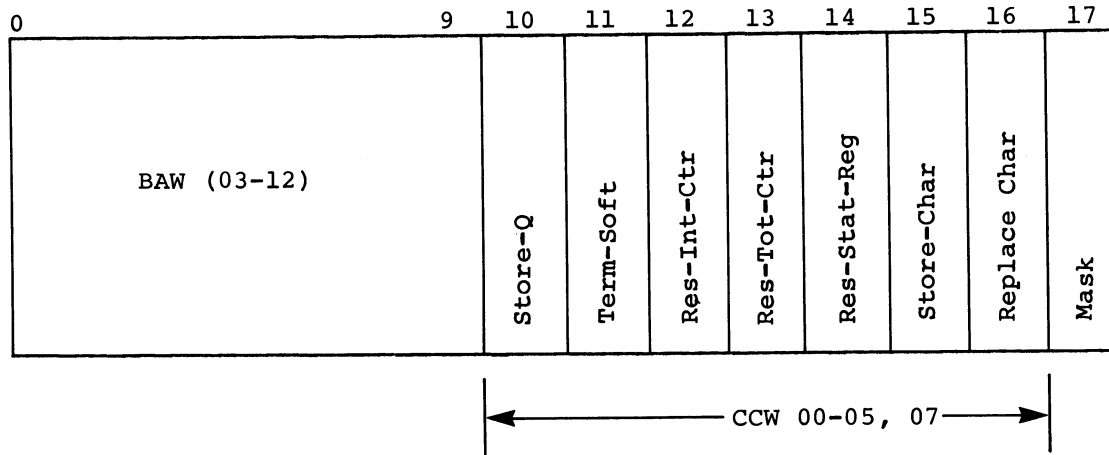
LOAD EXTERNAL FORMAT (LDEX)

- Loads the BAW with B Bus bits 8-17, and the CCW command register with B Bus 0-7.



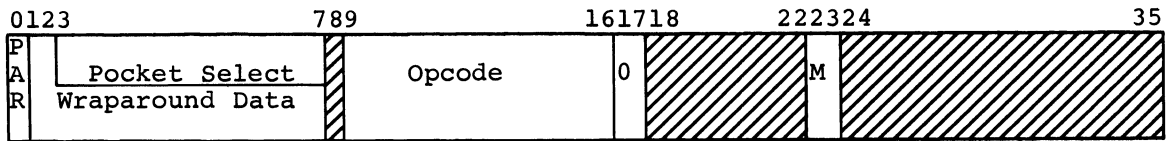
STORE EXTERNAL FORMAT (STEX)

- Reads the contents of the BAW across C Bus 0-9 and the contents of the CCW Command Register across C Bus 10-16, plus mask bit 17.



PERIPHERAL CONTROL WORD (PCW)

- Defines operation to be performed by the Peripheral or T and D Wrap.



Bit Position	Description
0	Odd Parity bit on even word only.
1-2	Wraparound data when in Wraparound Mode (Bit 16) can be used to simulate either MICR or OCR character.
3-7	Select Pocket (00-31) if used with Bit 14. Turn on Pocket light is used with Bit 9.
8	MBZ - Reserved for Opcode Extension
9-16	Opcode Field (Bit Position): 9 - Sorter Pocket Light (with Bit 3-7) <sup>2</sup> 10 - Batch Count Plus On <sup>2</sup> 11 - Start - Feed A (MICR) <sup>1</sup> 12 - Start - Feed B (OCR) <sup>1</sup> 13 - Stop Feed 14 - Pocket Select (with Bits 3-7) 15 - Request Status <sup>2</sup> 16 - Wraparound <sup>1</sup>
17-22	Must Be Zero
23	M = Mask Bit
24-35	MBZ
<sup>1</sup> Exclusive Bits - Only one on at a time. <sup>2</sup> May be 'OR'ed with other instruction bits.	

CHARACTER CONTROL WORD (CCW)

- A Double Word located in a table that is used for command, status, and replacement character.

0	78	1112	1718	252627	35
CCW Command	MBZ	CCW Status	MBZ	P	Replace Character

CCW Command		
CCW Bit	Mnemonic	Definition
00	Don't Store	If one, character not stored
01	Store-Q and Q Status	If one, queue character and Q status is stored.
02	Term-soft	Terminate via software
03	Res-Int-Ctr	Reset Interval Counter
04	Res-TOT-Ctr	Reset Total Counter
05	Res-Stat-Reg	Reset Status Register
06	-	MBZ
07	Replace	Replace character just read with CCW Replace character

CCW Status (Used and Interpreted by Software)	
Status Bit	Definitions
12	Garbage
13	Ignore (can't read) Character
14	Dash
15	Plus
16	Not Used
17	Blank

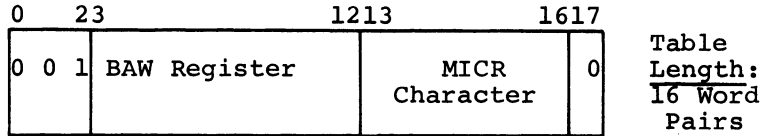
P - Bit 26 contains odd one parity on Odd Word.

Replace Character - 4 Bit MICR  
 - 7 Bit OCR

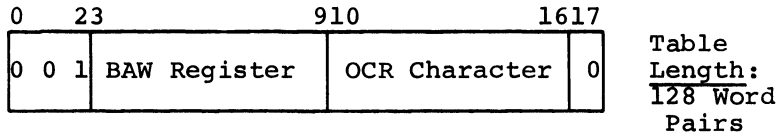
CCW ADDRESS FORMATION

- The character read (OCR or MICR) is appended to the BAW to reference a Double Word in CCW Table.

1. FEED-A-Format

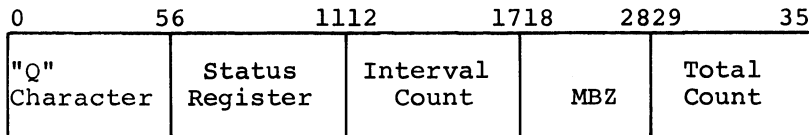


2. FEED-B Format



QUEUE STATUS WORD (QSW)

- Used to form a Status Table
- Stored Indirect double precision to the location specified by Channel ICW Base Address +4.

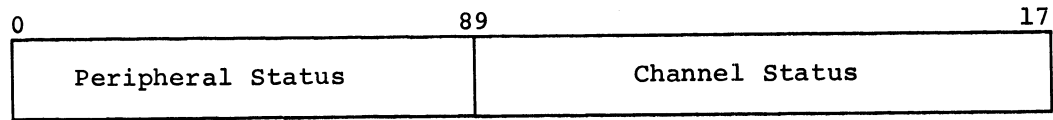


QSW Bits	Definition
0-5	Contain the "Q" character to be stored, either read or CCW replace character.
6-11	Contain in the CCW Status Register (stored from CCW and software interpreted).
12-17	Contain the Interval Count.
18-28	Must be Zero.
29-35	Contain the total count.

NOTE: Count and Status Registers are cleared by channel after each document is read.

TERMINATE STATUS WORD

- Stored Indirect single precision to location specified by Channel ICW Base Address +2.
- No special status on DHC601



Peripheral Status

Nine Status bits indicate peripheral status.

Bit Position	Code	Definition	Interrupt
0	SNOTR	Sorter Not Ready	Yes
1	RTERM	Read Terminate	Yes
2	--	Not Used	
3	STLP	Too Late to Pocket	No
	SDDL	Double Document Detect	No
5	SJAM	Document Jam	No
6	BTDL	Batch Ticket Detect	No
		Level	
7	BCEL	Batch Number Count	No
		Enable	
8	SMRT	Document Missort	No

The first two bit conditions act as major status and will cause an interrupt. The rest of the status conditions act as sub-status conditions and will not cause an interrupt.

Channel Status

The following status bits indicate Channel Status.

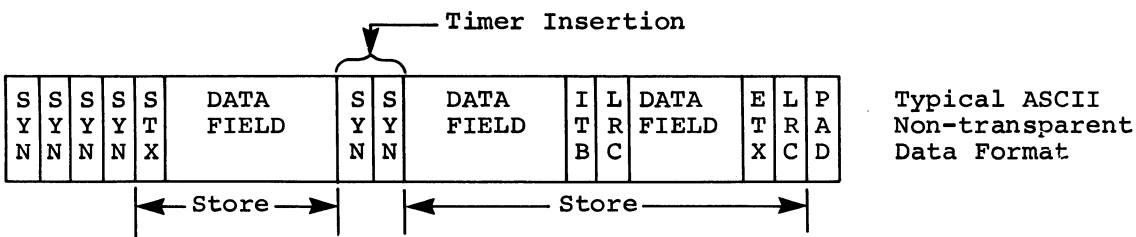
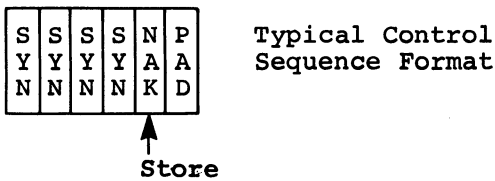
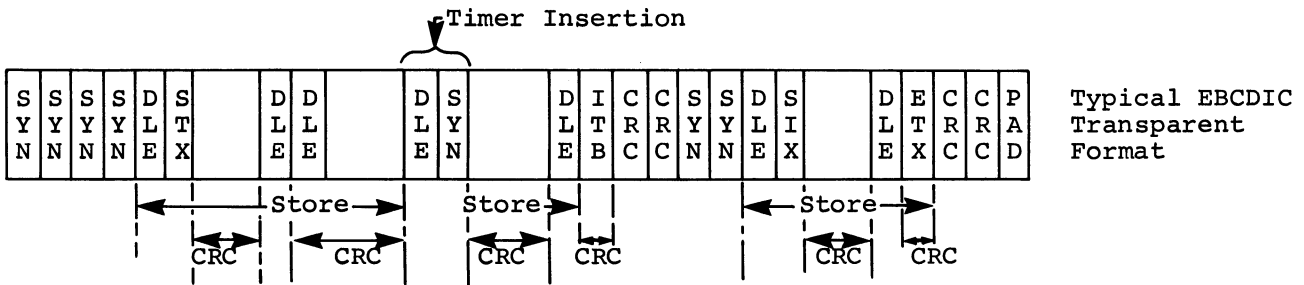
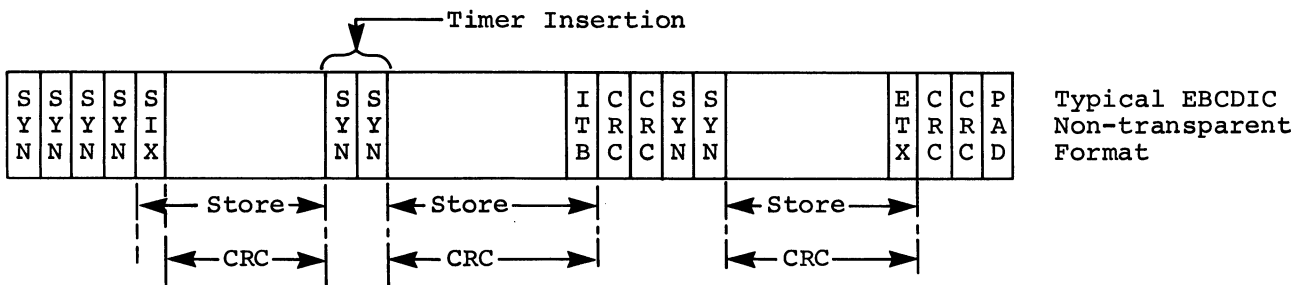
Bit Position	Code	Definition	Interrupt
9	STAST	Status Stored	Yes
10	TY0	Tally Runout	Yes
11	--	Not used	
12	TY1	Pre-Tally Runout	Yes
13	WRAP	Wraparound Mode	Yes
14	XFER	Transfer Timing Error	Yes
15	STERM	Software Terminate	Yes
16	NODOC	No More Documents to Read	Yes
17	PEPCW	PCW Parity Error	Yes

BINARY SYNCHRONOUS CHANNEL (BSC)

The BSC subchannel operates with either ASCII or EBCDIC code with either transparent or non-transparent data. When properly configured by the FNP program, the BSC subchannel is capable of transmitting and receiving data in any one of the following operational configurations:

- Non-transparent operation/ASCII code
- Non-transparent operation/EBCDIC code
- Transparent operation/ASCII code
- Transparent operation/EBCDIC code

The general formats of the message transmission blocks and control sequences transmitted and received in each of these configurations are:



## Transmit Mode

In general, this subchannel is used as a normal synchronous channel. The software sets up the data, data buffer areas and control words to configure the subchannel.

In setting up the data area, software provides the data field including control characters. The hardware supplies the synchronizing sequence, error control and pad character. For the non-transparent ASCII mode the error control (Longitudinal Redundancy Check Character) must be generated and checked (in receive mode) by software. For the other modes the error control is provided entirely by hardware.

There is one second timer used in the transmit mode and a three second timer for receive mode. If the message takes longer than one second to transmit, the hardware inserts two synchronizing characters (or DLE SYN for transparent operation) to maintain message synchronization. The receiving end deletes these characters from the message.

The hardware recognizes the control characters in the message to determine when to insert the CRC (Cyclic Redundancy Check) bytes.

For non-transparent ASCII, the operation of the channel is similar to an HSC355 or HSS355 subchannel. Software takes care of the LRC and terminates the transfer properly with the correct DCW data count.

For non-transparent EBCDIC, the hardware recognizes SOH or STX as start of message. AN ITB is recognized but does not cause a termination. When ETX or ETB is recognized, the hardware appends the CRC bytes and pad byte (all 1's) to the message and terminates whether the DCW is expired or not. This operation is also terminated with an expired DCW, whether an ETX or ETB is the last byte in the buffer or not. An ENQ character is recognized as an abort character and causes termination without CRC's and pad. The following control sequence characters are recognized by the hardware in non-transparent EBCDIC mode: NAK, ENQ, EOT, DLE, DLE'70', DLE/, and DLE. In the transmit mode, a pad character is added by the hardware and the operation is terminated.

For transparent operations (ASCII or EBCDIC) the hardware recognizes DLE STX as a start character and DLE ETX or DLE ETB as the termination character. The DLE ITB is recognized but does not cause a terminate. The DLE ENQ combination is recognized as an abort condition.



## Receive Mode

As in the transmit mode, software sets up the data buffers and control words to configure the channel. For non-transparent ASCII, the character-control-table (CCC) must be specified by software and the LRC calculation and checking is the responsibility of the software. Character parity, however, is provided by the hardware.

For non-transparent EBCDIC and transparent operations (ASCII or EBCDIC), the hardware terminates upon recognizing the message terminate characters or by DCW tally run out (see Transmit Mode).

In the receive mode with the timer running (3 sec.), the hardware stores status if the timer runs out without receiving any message synchronization characters, but does not terminate data reception.

## Control Words

PCW0 may be used to command the HSLA. There are no changes to its format.

PCW1 is used to set up the BCS Subchannel. The format is:

<u>PCW1 Bit</u>	<u>Definition</u>
0-1	Must be = 01
2-3	Must be = 00 (part of operation code)
4-5	Part of operation code
6	Not used
7-11	Subchannel number (0-31)
12-26	Not used
27	1 = Receive mode, 0 = Disable receive mode
28	1 = Send mode, 0 = Disable send mode
29	(for T and D)
30	1 = Set Data Terminal Ready, 0 = Reset DTR
31	1 = Set Request to Send, 0 = Reset RTS
32	(for T and D)
33-34	Not used
35	(for T and D)

PCW3 is used to configure the subchannel for operation with the correct code (ASCII or EBCDIC) transparent or non-transparent data and with the correct cyclic check polynomial (CRC-16 for EBCDIC and CCITT for ASCII). The format is:

<u>PCW3 Bit</u>	<u>Definition</u>
0-1	Must be - 11
2-3	Must be - 01
4-6	Not used
7-11	Subchannel number (0-31)
12	LPR (lateral parity-check or receive)
13	LPS (lateral parity - generation send)
14	1 = Odd lateral parity, 0 = Even lateral parity
15	2SB (two send buffers - ICW's)
16	1 = Use char. control table, 0 = Don't use
17-19	Not used
20-23	Not used
24	1 = CRC-16 Polynomial ( $X^{16}+X^{15}+X^2+1$ ) 0 = CCITT Polynomial ( $X^{16}+X^{12}+X^5+1$ )
25-26	Not used
27	1 = EBCDIC code (8 bits), 0 = ASCII code (7+1 parity)
28	1 = Transparent mode, 0 = Non-transparent
29	1 = Enable Transmit and Rec. Timers, 0 = Disable timers
30-35	Not used

#### Status Words

There are two types of status stored, Active and Configuration. They may both be obtained by software command. In addition, prior to all software interrupts, Active status is stored by the hardware. Status stores may be initiated by the HSLA or by the subchannel. Configuration status is obtained by software command only.

## Active Status

<u>Bit</u>	<u>Definition</u>
0	1 = Receive status, 0 = Send status
1	1 = Normal marker character received
2	1 = Delayed marker character received
3	1 = Terminate character received
4	0 = Active primary buffer, 1 = Active alternate buffer
5	1 = Buffers will be switched after status store
6	1 = Tally is zero
7	1 = Tally is one
8	1 = Lateral parity error detected somewhere in message received
9	1 = Command sent to unimplemented subchannel
10	1 = Change in data set (modem) status has occurred
11	Not used
12	1 = Transfer timing error
13-17	Not used
18	1 = Data Set Ready line is up, 0 = DSR is down
19	1 = Clear To Send line is up, 0 = CTS is down
20	1 = Carrier Detect line is up, 0 = CD is down
21-23	Not used
24	1 = Ring Indicator is up, 0 = RI is down
25	Not used
26	1 = Receive block terminate
27	1 = Receive mode
28	1 = Send mode
29	(for T and D)
30	1 = Data Terminal Ready is up, 0 = DTR is down
31	1 = Request To Send is up, 0 = RTS is down
32	(for T and D)
33	1 = CRC error was detected
34	1 = Receive time out occurred (3 sec.)
35	(for T and D)

## Configuration Status

<u>Bit</u>	<u>Definition</u>
0	1 = Configuration status
1	1 = Synchronous subchannel
2	Not used
3-8	Subchannel type number = 14 (octal)
9-11	Not used
12	LPR (Receive lateral parity check)
13	LPS (Send lateral parity generate), 7+1 codes
14	1 = Odd lateral parity, 0 = Even lateral parity
15	1 = Two send buffers, 0 = One send buffer
16	1 = Use CCT, 0 = Don't use CCT on receive
17-19	Not used
20-23	Not used
24	1 = CRC-16 Polynomial, 0 = CCITT Polynomial
25-26	Not used
27	1 = EBCDIC code (8 bits), 0 = ASCII code (7+1)
28	1 = Transparent mode, 0 = Non-transparent
29	1 = Timers enabled, 0 = Timers disabled
30-35	Not used

## COMPUTER MONITOR ADAPTER (CMA)

The Computer Monitor Adapter (CMA) provides a data and monitor link between dual DATANET FNP systems used in message switching. It provides a communication path between the two systems for constant update of each systems communication tables and status.

### Configuration Patching

#### CHANNEL NUMBER PATCH

Patchable to any six-bit code

#### INTERRUPT LEVEL PATCH

Send and Receive interrupts patchable to any of 16 program interrupt levels.

#### INDIRECT CONTROL WORD BASE ADDRESS PATCH

Patchable in range of 460 to 770 (octal)

- The ICW Base Address is the location of the Receive ICW.
- The ICW Base Address +2 is the location of the Send ICW.
- The ICW Base Address +4 is the location of the Receive Status ICW.
- The ICW Base Address +6 is the location of the Send Status ICW.

#### "DEAD-MAN" TIMER DURATION PATCH

Patchable for 250 ms, 500 ms or 1.0 second.

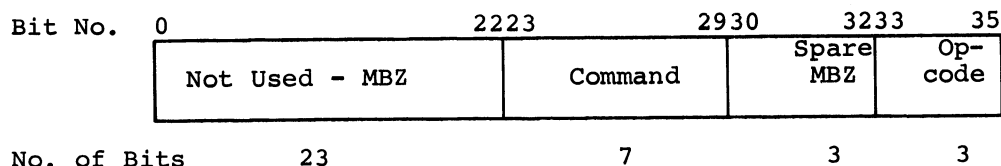
#### DATA RESPONSE TIMER DURATION PATCH

Patchable for 125 ms or 250 ms.

CMA Control Words

PERIPHERAL CONTROL WORD (PCW)

The general format of the PCW used for control of the CMA is:



The seven command bits in the PCW are used to send broadside commands to the CMA:

<u>Bit</u>	<u>Function</u>
23	Mask Channel
24	Timer Control
25	Receive Mode
26	Send Mode
27	Wraparound Mode
28	Switch Control
29	Parity Test

The 3 Opcode bits are encoded (bit 33 = MSB) as follows:

<u>Opcode Value (octal)</u>	<u>Function</u>
0	(no op)
1	Timer reset
2	Request Send Status
3	Request Receive Status
4 to 7	(spare)

When a PCW is sent to the CMA, the command bits result in the following actions by the CMA.

<u>Bit</u>	<u>Action by CMA</u>
23	When a 1, the channel is masked and initialized and does not request service of the IOM; when a 0, the channel is unmasked and may request IOM service as necessary for normal operation.
24	When a 1, the CMA dead-man timer is enabled; when a 0, the timer is stopped.
25	When a 1, the CMA receive mode logic is enabled; when a 0, it is disabled.
26	When a 1, the CMA send mode logic is enabled; when a 0, it is disabled.

- 27 When a 1, the CMA external interface lines are wrapped back on one another; when a 0, the external interface lines of the CMA are returned to their normal operational state.
- 28 When a 1, the Switch Control lead to the Line Transfer Device is turned on (indicating "permission" to be placed on-line); when a 0, the Switch Control lead is turned off.
- 29 When a 1, the CMA send logic is forced to generate incorrect parity (even) on data bytes to be sent while in the wraparound mode. When a 0, normal odd parity is generated. This bit is interpreted by the CMA only when in the wraparound mode (PCW bit 27 = 1).

The Opcodes in the PCW sent to the CMA result in the following actions by the CMA.

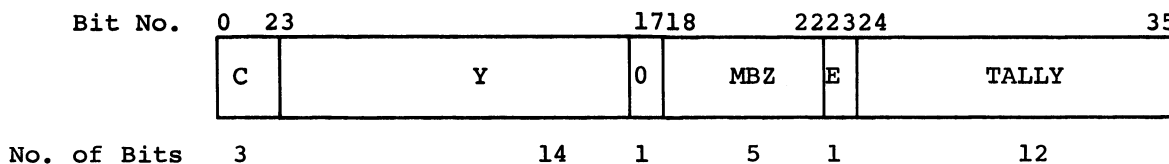
Opcode	Action by CMA
0	No action caused by this Opcode. Used in conjunction with broadside commands not requiring any Opcode.
1	Causes the "dead-man" timer to be reset to the beginning of its timing cycle. This Opcode is ignored if sent when the timer is disabled by command bit 24.
2	Causes the CMA to store Send Status and issue a Send Status Interrupt.
3	Causes the CMA to store Receive Status and issue a Receive Status Interrupt.
4-7	(spare - reserved for future use)

#### INDIRECT CONTROL WORD (ICW)

Indirect data transfers between the CMA and the FNP memory are effected by the IOM through the use of ICW's - one for send data, one for receive data, one for send status and one for receive status. The ICW's are stored in FNP memory locations based on the ICW Base Address.

The memory address of the appropriate ICW is presented to the IOM by the CMA when an indirect data transfer is requested by the CMA.

All four ICW's have the same format:



C - Byte size should be 010 or 011 (9-bit characters) for Send ICW and Receive ICW. Should be 000 (18-bit word) for Send Status ICW and Receive Status ICW.

Y - Address of first data word bit 17 is forced to zero by channel.

E - Exhaust bit. If a one, tally has run out.

TALLY - Number of accesses to memory.

### Data Transfer

- Data transmission initiated by FNP program.
- Program sets up transmit buffer and Send ICW.
- Program issues PCW to CMA to set Send mode.
- Channel issues "Send Mode On" indicator to other system and starts 250 ms "Send Mode Response" timer.
- If other system does not respond, first CMA resets Send mode, stores Send status and issues a Send status interrupt.
- If other system responds with "Receive Mode On" indication within timer period, CMA requests the first data byte.
- Data reception requires FNP program to set up receive buffer and Receive ICW.
- Program issues a PCW to CMA to set Receive mode.

### Status Words

Two types of status are stored by the CMA - Send Status and Receive Status. These status words are placed on the least significant bit positions of the C Bus (corresponding to the bit positions in an 18-bit FNP word):

C Bus Bit	Send Status	Receive Status
17	Line Switch Mode <sup>1</sup>	Line Switch Mode
16	Line Switch Position <sup>1</sup>	Line Switch Position
15	System Confidence (other system) <sup>1</sup>	System Confidence (other system)
14	Switch Control <sub>1</sub> (other system) <sup>1</sup>	Switch Control (other system)
13	Switch Control	Switch Control
12	Wraparound Mode	Wraparound Mode
11	Send Mode	Send Mode
10	Receive Mode	Receive Mode
9	Timer Control	Timer Control
8	Send PTRO <sup>2</sup>	Receive PTRO <sup>3</sup>
7	Send TRO <sup>2</sup>	Receive TRO <sup>2</sup>



C Bus Bit	Send Status	Receive Status
6	Parity error on byte sent	Parity error on byte received
5	Transmission terminated by receiving system <sup>2</sup>	Send Mode (other system) <sup>4</sup>
4	Data parity error (IOM to CMA)	Data parity error (CMA to IOM)
3	PCW parity error <sup>2</sup>	
2	Send mode response timeout <sup>2</sup>	

<sup>1</sup>Causes a Send Status interrupt whenever change in state occurs.

<sup>2</sup>Causes a Send Status interrupt whenever bit changes to one.

<sup>3</sup>Causes a Receive Status interrupt whenever bit changes to one.

<sup>4</sup>Causes a Receive Status interrupt whenever change in state occurs.

The functions of the Send Status word bits are:

<u>Bit</u>	<u>Function</u>
17	When a 1, indicates that the line transfer switch is in the automatic mode of operation. When a 0, indicates that the line transfer switch is in the manual mode of operation.
16	When a 1, indicates that the system is connected to the communication lines. When a 0, indicates that the system is not connected to the communication lines (other system online).
15	When a 1, the System Confidence signal from the other system CMA is on. When a 0, it is off.
14	When a 1, the Switch Control signal from the other system CMA is on. When a 0, it is off.
13	When a 1, the Switch Control signal ( <u>this</u> CMA) is on. When a 0, it is off.
12	When a 1, the CMA is in the Wraparound Mode. When a 0, it is not.
11	When a 1, the CMA's Send logic is enabled. When a 0, it is not.
10	When a 1, the CMA's Receive logic is enabled. When a 0, it is not.
9	When a 1, the dead-man timer is enabled. When a 0, it is not.
8	When a 1, a Send pre-tally run out has occurred.
7	When a 1, a Send tally run out has occurred.
6	When a 1, the other system CMA detected a parity error on a byte sent to it. (Status reported at end of Send operation).
5	When a 1, the system turned its Receive mode off.
4	When a 1, a data byte containing a parity error was received from the IOM. (Status reported at end of Send operation).

- 3 When a 1, a PCW containing a parity error was received from the IOM. (This PCW is not executed).
- 2 When a 1, the other system did not respond with a Receive mode on indication within the 250 millisecond interval after Send mode was set. Did not respond with a Data Received strobe after a Data Available strobe. (Resets Send mode and cause status store).

The functions of the Receive Status word bits are:

<u>Bit</u>	<u>Function</u>
17-9	(same as for Send status word)
8	When a 1, a pre-tally run out occurred on the last byte stored indirect by the CMA.
7	When a 1, a Receive tally run out has occurred.
6	When a 1, a parity error was detected on a data byte received from the other system. This is not reported until the data transfer operation is completed.
5	When a 1, indicates that the other system Send mode is on. When a 0, indicates that the other system Send mode is off.
4	When a 1, indicates that a data byte transferred from the CMA to the IOM was received by the IOM with a parity error. This status is not reported until the data transfer operation is terminated.

### System Monitoring

Monitoring by CMA hardware timer which must be periodically reset by FNP program. If timer runs out, is disabled by program, or system loses power, the System Confidence signal is turned off.

System Confidence signal is turned on (and maintained) by FNP program issue of PCW to enable dead-man timer. This causes CMA to turn on signal and start a one-second timer (hardware). This timer must be reset by program (PCW) within each one second time period.

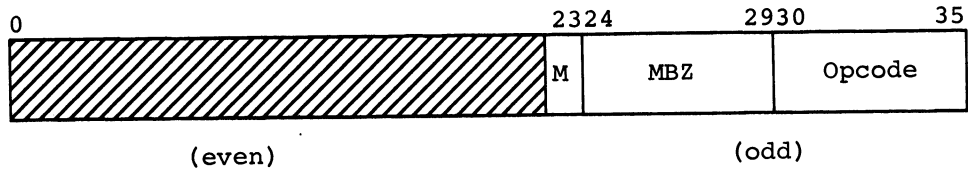


CARD READER

General Information

- I/O Channel number = 1 (hardware patchable)
  - Interrupt vectors (hardware patchable)
- IOM detected fault - 20  
 Special - 21  
 Terminate - 22
- Control words

1. PCW



where:

Bit 23 = M - Mask bit  
 Bits 30-35 = Opcode as follows

Op-code	Command	MAP Mnemonic
00	Request status	REQS
40	Reset status	RESS
01	Read card binary	RCB
02	Read card decimal	RCD
03	Read card mixed	RCM

2. ICW Addresses

Status - 464, 465 Indirect 36  
 Data - 466, 467 Indirect 6

• Status

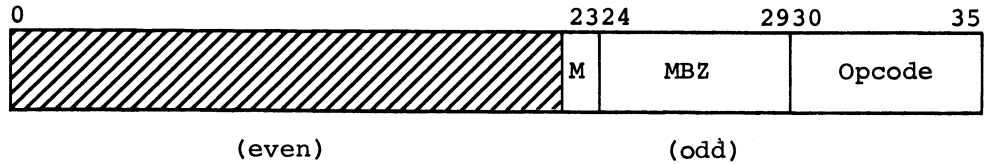
	Major Status					Substatus					
	Bits	2	3	4	5	6	7	8	9	10	11
Ready		0	0	0	0	0	0	0	0	0	0
Device Attention		0	0	1	0	0	0	0	0	0	1
						0	0	0	0	1	0
						0	0	0	1	0	0
						0	0	1	0	0	0
						0	1	0	0	0	0
						1	0	0	0	0	0
						1	0	0	0	0	0
Data alert		0	0	1	1	0	0	0	0	0	1
						0	0	0	0	1	0
Command reject		0	1	0	1	X	X	X	X	X	X
Busy		1	0	0	0	X	X	X	X	X	X

LINE PRINTER

General Information

- I/O Channel number = 2 (hardware patchable)
- Interrupt Vectors (octal) (hardware patchable)
  - Fault (IOM detected) = 40
  - Special = 41
  - Terminate = 42
- Control Words

1. PCW



where:

Bit 23 = M  
Bits 30-35 = Opcode as follows

Op-code	Command	MAP Mnemonic
00	Request status	REQS
40	Reset status	RESS
10	Write printer, non-edited, no slew	WPR
11	Write printer, non-edited, slew 1	WPR1
12	Write printer, non-edited, slew 2	WPR2
13	Write printer, non-edited, slew to TOP	WPR20
30	Write printer, - edited, no slew	WPRE
31	Write printer, - edited, slew 1	WPRE1
32	Write printer, - edited, slew 2	WPRE2
33	Write printer, - edited, slew to TOP	WPRE20
61	Slew 1 line	SLW1
62	Slew 2 lines	SLW2
63	Slew to top of form	SLW20

2. ICW Addresses (hardware patchable)

Status - 470, 471 Indirect 36  
Data - 472, 473 Indirect 6

- Status

Major Status					Substatus						
Bits	2	3	4	5	6	7	8	9	10	11	
Ready	0	0	0	0	0	0	0	0	0	0	
Attention	0	0	1	0	0	0	X	X	X	1	out of paper
					0	0	X	X	X	X	manual halt
					0	0	X	1	X	X	VFU tape alert
					0	0	1	X	X	X	check
Data alert	0	0	1	1	X	X	X	0	1	X	alert before printing
					0	0	0	0	1	1	transfer timing alert
					X	X	X	1	0	0	alert during printing
					0	X	1	X	X	X	paper low
					X	1	X	X	X	X	slew alert
					1	X	X	X	X	X	top of page echo
Command reject	0	1	0	1	X	X	0	0	0	1	invalid opcode
					X	1	0	0	0	X	slew alert on last slew command
					1	X	0	0	0	X	top page echo on last slew

TIMER AND SWITCH CHANNEL

General Information

- I/O Channel no. = 77 (octal) (hardware patchable)
- Interrupt vectors (hardware patchable)
  - IOM detected fault - 360
  - Interval timer runout - 361
  - Elapsed timer rollover - 362
- Mailboxes (hardware patchable)
  - 450 - Interval timer
  - 451 - Elapsed timer
- PCW - Only mask bit (23) is used
  - Affects interval timer only (elapsed timer unaffected)
  - If mask bit = 1, interval timer turned OFF.
  - If mask bit = 0, interval timer turned ON.
- Status - None reported
- Other - To read maintenance panel data switches, use STEX to channel 77 (octal)

COMMON PERIPHERAL STATUS FORMAT

The status word has the following format:

0	1	2	56	1112	15161718	2324	3233	35	
1	P	Major Status	Substatus	MBZ	1	A	Status Detected by Channel	MBZ	CPR

- Bit 0 - A one indicates a status has been stored.
- Bit 1 P - A one indicates that the device does not have power on or there is no device attached.
- Major Status - The major status received from the peripheral subsystem.
- Substatus - The substatus received from the peripheral subsystem.
- Bit 16 I - Set if the device instruction is not accepted by the peripheral. The peripheral subsystem does not go busy as a result of this instruction.
- Bit 17 A - Stored as zero by the hardware - reserved for software use.
- Status Detected by Channel (Bits 18-23)
  - Bit 18 - Not used.
  - Bit 19 - Connect While Busy - Error flag denoting that a connect was received while the channel was busy with a previous operation.
  - Bit 20 - Control Word Error - An illegal ICW count of zero was sensed in a PCW while the channel was busy with a previous operation.
  - Bit 21 - Character Position Error - An illegal character position count (6 or 7) was sensed in processing a PCW that called for data transfer.
  - Bit 22 - Tally Runout Error - Tally runout condition was sensed during data service mode.
  - Bit 23 - Transmission Parity Error - Channel detected parity error on character from peripheral subsystem.
- CPR
  - Bits 33-35 - The channel returns the character position residue to the DATANET FNP. This residue is the character number into which the next data character would be transmitted if the transaction were continued. This channel always transmits a 36-bit word but indicates the last actual character + 1 (modulo 6) as the character position residue.
- MBZ - Must be zero

DATANET FNP GENERAL MEMORY MAP

Address (octal)	Function
00000 } 00377 }	Interrupt vectors
00400 } 00417 }	Interrupt cells
00420 } 00437 }	IOM fault status
00440 } 00447 }	Processor fault vectors
00450 } 00777 }	I/O Comm. Region
01000 } 01777 }	HSLA #1 I/O Comm. Region
02000 } 02777 }	HSLA #2 I/O Comm. Region
03000 } 03777 }	HSLA #3 I/O Comm. Region
04000 } 77777 }	Program area

(optional program area)

Interrupt Cells

There are 256 interrupt cells available in the DATANET FNP, divided into 16 levels with 16 cells each. Levels correspond to words in memory, with cells being equivalent to bit positions (0-15) within the level. Levels 0-15 are located in 400 (octal) through 417 (octal) respectively.

Masking is by level only, and interrupt service (answering) priority is by cells within a level, then by level (level 0 highest).

The interrupt vector location corresponding to any cell (bit) within a level is found as follows:

$$(\text{interrupt vector address})_8 = (\text{bit position} \times 20_8) + (\text{level})_8$$

Conversely, if an interrupt vector location is known, the corresponding level is equal to the four least significant bits of the vector address, and the bit position (cell) is equal to the next four bits of the address.

Figure 5-2 shows a general interrupt cell map. Interrupt vectors are shown in Figure 5-3.





Address (octal)	Interrupt Level (decimal)	Function
00000	0	Console { fault special (request) terminate
001	1	
002	2	
003	3	ICA Special #0
004	4	HSLA #1 { active subchannel 0 active subchannel 16 config. subchannel 0 config. subchannel 16
005	5	
006	6	
007	7	
010	8	Reserved for HSLA #2
011	9	
012	10	
013	11	
014	12	Reserved for HSLA #3
015	13	
016	14	
017	15	
020	0	Card Reader { fault special terminate
021	1	
022	2	
023	3	ICA Special #1
024	4	HSLA #1 { active subchannel 1 active subchannel 17 config. subchannel 1 config. subchannel 17
025	5	
026	6	
027	7	
030	8	Reserved for HSLA #2
031	9	
032	10	
033	11	
034	12	Reserved for HSLA #3
035	13	
036	14	
037	15	
040	0	Line Printer { fault special terminate
041	1	
042	2	
043	3	ICA Special #2
044	4	HSLA #1 { active subchannel 2 active subchannel 18 config. subchannel 2 config. subchannel 18
045	5	
046	6	
00047	7	

Figure 5-3. DATANET FNP Interrupt Vectors

Address (octal)	Interrupt Level (decimal)	Function	
00048	8	Reserved for HSLA #2	
051	9		
052	10		
053	11		
054	12	Reserved for HSLA #3	
055	13		
056	14		
057	15		
060	0	ICA Special #3	
061	1		
062	2		
063	3		
064	4	HSLA #1	{ active subchannel 3 active subchannel 19 config. subchannel 3 config. subchannel 19
065	5		
066	6		
067	7		
070	8	Reserved for HSLA #2	
071	9		
072	10		
073	11		
074	12	Reserved for HSLA #3	
075	13		
076	14		
077	15		
100	0	ICA	fault
101	1	Unassigned	
102	2	ICA	terminate
103	3	ICA Special #4	
104	4	HSLA #1	{ active subchannel 4 active subchannel 20 config. subchannel 4 config. subchannel 20
105	5		
106	6		
107	7		
110	8	Reserved for HSLA #2	
111	9		
112	10		
113	11		
114	12	Reserved for HSLA #3	
115	13		
116	14		
00117	15		

Figure 5-3 (cont). DATANET FNP Interrupt Vectors

Address (octal)	Interrupt Level (decimal)	Function
00120	0	Unassigned
121	1	
122	2	
123	3	ICA Special #5
124	4	HSLA #1 { active subchannel 5 active subchannel 21 config. subchannel 5 config. subchannel 21
125	5	
126	6	
127	7	
130	8	Reserved for HSLA #2
131	9	
132	10	
133	11	
134	12	Reserved for HSLA #3
135	13	
136	14	
137	15	
140	0	HSLA #1 fault
141	1	Unassigned
142	2	
143	3	ICA Special #6
144	4	HSLA #1 { active subchannel 6 active subchannel 22 config. subchannel 6 config. subchannel 22
145	5	
146	6	
147	7	
150	8	Reserved for HSLA #2
151	9	
152	10	
153	11	
154	12	Reserved for HSLA #3
155	13	
156	14	
157	15	
160	0	HSLA #2 fault
161	1	Unassigned
162	2	
163	3	ICA Special #7
164	4	HSLA #1 { active subchannel 7 active subchannel 23 config. subchannel 7 config. subchannel 23
165	5	
166	6	
00167	7	

Figure 5-3 (cont). DATANET FNP Interrupt Vectors

Address (octal)	Interrupt Level (decimal)	Function
00170	8	Reserved for HSLA #2
171	9	
172	10	
173	11	
174	12	Reserved for HSLA #3
175	13	
176	14	
177	15	
200	0	HSLA #3            fault
201	1	Unassigned
202	2	
203	3	ICA Special #8
204	4	HSLA #1            { active subchannel 8 active subchannel 24 config. subchannel 8 config. subchannel 24
205	5	
206	6	
207	7	
210	8	Reserved for HSLA #2
211	9	
212	10	
213	11	
214	12	Reserved for HSLA #3
215	13	
216	14	
217	15	
220	0	LSLA                fault
221	1	{ active configuration
222	2	
223	3	ICA Special #9
224	4	HSLA #1            { active subchannel 9 active subchannel 25 config. subchannel 9 config. subchannel 25
225	5	
226	6	
227	7	
230	8	Reserved for HSLA #2
231	9	
232	10	
233	11	
234	12	Reserved for HSLA #3
235	13	
236	14	
00237	15	

Figure 5-3 (cont). DATANET FNP Interrupt Vectors

Address (octal)	Interrupt Level (decimal)	Function
00240	0	Unassigned
↑ 241	1	
242	2	
243	3	ICA Special #10
244	4	HSLA #1 { active subchannel 10 active subchannel 26 config. subchannel 10 config. subchannel 26
245	5	
246	6	
247	7	
250	8	Reserved for HSLA #2
251	9	
252	10	
253	11	
254	12	Reserved for HSLA #3
255	13	
256	14	
257	15	
260	0	Unassigned
261	1	
262	2	
263	3	ICA Special #11
264	4	HSLA #1 { active subchannel 11 active subchannel 27 config. subchannel 11 config. subchannel 27
265	5	
266	6	
267	7	
270	8	Reserved for HSLA #2
271	9	
272	10	
273	11	
274	12	Reserved for HSLA #3
275	13	
276	14	
277	15	
300	0	Unassigned
301	1	
302	2	
303	3	ICA Special #12
304	4	HSLA #1 { active subchannel 12 active subchannel 28 config. subchannel 12 config. subchannel 28
305	5	
306	6	
↓ 00307	7	

Figure 5-3 (cont). DATANET FNP Interrupt Vectors

Address (octal)	Interrupt Level (decimal)	Function	
00310	8	Reserved for HSLA #2	
311	9		
312	10		
313	11		
314	12	Reserved for HSLA #3	
315	13		
316	14		
317	15		
320	0	Unassigned	
321	1		
322	2		
323	3	ICA Special #13	
324	4	HSLA #1	{ active subchannel 13 active subchannel 29 config. subchannel 13 config. subchannel 29
325	5		
326	6		
327	7		
330	8	Reserved for HSLA #2	
331	9		
332	10		
333	11		
334	12	Reserved for HSLA #3	
335	13		
336	14		
337	15		
340	0	Unassigned	
341	1		
342	2		
343	3	ICA Special #14	
344	4	HSLA #1	{ active subchannel 14 active subchannel 30 config. subchannel 14 config. subchannel 30
345	5		
346	6		
347	7		
350	8	Reserved for HSLA #2	
351	9		
352	10		
353	11		
354	12	Reserved for HSLA #3	
355	13		
356	14		
00357	15		

Figure 5-3 (cont). DATANET FNP Interrupt Vectors

Address (octal)	Interrupt Level (decimal)	Function
00360	0	Timer channel fault
361	1	Interval timer runout
362	2	Elapsed timer rollover
363	3	ICA Special #15
364	4	HSLA #1 { active subchannel 15 active subchannel 31 config. subchannel 15 config. subchannel 31
365	5	
366	6	
367	7	
370	8	Reserved for HSLA #2
371	9	
372	10	
373	11	
374	12	Reserved for HSLA #3
375	13	
376	14	
00377	15	

Figure 5-3 (cont). DATANET FNP Interrupt Vectors

Processor Fault Vectors

Absolute Address (octal)	Function
0040	Power off
0041	Power on
0042	Memory Parity
0043	Illegal Operation Code
0044	Overflow
0045	Illegal Memory Operation
0046	Divide Check
0047	Illegal Program Interrupt



IOC Fault Status Locations

Address (octal)	Function
00420	I/O Chan. 0 (console)
↑ 1	I/O Chan. 1 (card reader)
2	I/O Chan. 2 (line printer)
3	Not used
4	I/O Chan. 4 ICA
5	Not used
↓ 6	I/O Chan. 6 (HSLA)
00427	Not used (reserved for HSLA #2)
00430	Not used (reserved for HSLA #3)
↑ 1	I/O Channel 11 (LSLA)
2	Not used (reserved for LSLA #2)
3	Not used (reserved for LSLA #3)
4	Not used (reserved for LSLA #4)
5	Not used (reserved for LSLA #5)
↓ 6	Not used (reserved for LSLA #6)
00437	Timer Channel

CODING EXAMPLES

The following pages show examples of coding techniques for performing typical program functions. These examples:

1. Illustrate the use of address modification variations for indexing, indirection, and character addressing.
2. Demonstrate operations performed on characters.

The list of examples is by no means complete in that it does not present all of the processor capabilities; however, the examples serve as convenient references for programmers newly acquainted with the DATANET FNP.

Each example is self-contained and self-explanatory. In most cases, questions that may be raised can be answered by referring to the descriptions of particular instructions or pseudo-operations.

## BCD Addition

The following example illustrates the addition of two words containing BCD integers. The example limits the result to 999.

```
01      LDI   OVMASK-*  INHIBIT OVERFLOW FAULTS
02      LDA   A-*      TO ADD C=A+B IN BCD
03      ADA   B-*      COMPUTE A+B
04      ADA   OCTL66-*  ADD OCTAL 66 TO EACH DIGIT TO FORCE CARRIES
05      STA   C-*
06      ANA   OCTL60-*  EXTRACT OCTAL 60 FROM EACH NON-CARRY
07      ERSA  C-*      SUBTRACT OCTAL 60 FROM EACH NON-CARRY
08      ARL   3        SUBTRACT
09      ERA   MINUS1-*  OCTAL 06
10      IAA   1        FROM EACH
11      ASA   C-*      NON-CARRY
      .
      .
      .
12 OVMASK OCT   004000  OVERFLOW FAULT INHIBIT INDICATOR
13 OCTL66 OCT   666666  OCTAL 66s
14 OCTL60 OCT   606060  OCTAL 60s
15 MINUS1 DEC   -1      MINUS ONE
```

- 01 Inhibits an overflow fault from occurring during add operations.
- 02 Places the number in A into the accumulator.
- 03 Adds the number in B to the accumulator. Column V in the table following, shows the possible results for any digit. It should be noted that there are 19 possible results, indicated by lines 0-18.
- 04 Forces any carries into the units position of the next digit. Column W contains the 20 possible results for each digit position. The additional possibility (line 19) arises from the fact that there can be a carry of one into a digit.
- 05 Stores the intermediate result in C.
- 06 Extracts an octal 60 from each non-carry digit. The results are indicated in Column X. The digits that did not force a carry (lines 0-9) result in an octal 60, the digits that had a carry into the next digit (lines 10-18) result in 00.
- 07 Performs an exclusive OR of the contents of the accumulator with the contents of C. This in effect subtracts octal 60 from each digit that did not have a carry (lines 0-9). The results are indicated in Column Y.
- 08 Shifts the octal 60s to the right three places.
- 09
- 10 Negates the contents of the accumulator.
- 11 Adds to memory of the contents of the accumulator with the contents of C. This in effect subtracts a 06 from each digit that did not have a carry. The results are indicated in Column Z.

ADDITION RESULTS					
LINE	V	W	X	Y	Z
0	00	66	60	6	00
1	01	67	60	7	01
2	02	70	60	10	02
3	03	71	60	11	03
4	04	72	60	12	04
5	05	73	60	13	05
6	06	74	60	14	06
7	07	75	60	15	07
8	10	76	60	16	10
9	11	77	60	17	11
10	12	00	00	0	00
11	13	01	00	1	01
12	14	02	00	2	02
13	15	03	00	3	03
14	16	04	00	4	04
15	17	05	00	5	05
16	20	06	00	6	06
17	21	07	00	7	07
18	22	10	00	10	10
19	--	11	00	11	11

### BCD Subtraction

The following is an example of subtracting one BCD number from another BCD number. The contents of A must be equal to or greater than the contents of B.

```

01      LDI   OVMASK-*  INHIBIT OVERFLOW FAULTS
02      LDA   A-*      [ TO SUBTRACT C=A-B IN BCD
03      SBA   B-*      [ COMPUTE A-B
04      STA   C-*
05      ANA   OCTL60-*  EXTRACT OCTAL 60 FROM EACH BORROW
06      ERSA  C-*      SUBTRACT OCTAL 60 FROM EACH BORROW
07      ARL   3         [ SUBTRACT
08      ERA   MINUS1-* [ OCTAL 06
09      IAA   1         [ FROM EACH
10      ASA   C-*      [ BORROW
      .
      .
11 OVMASK  OCT   004000  OVERFLOW FAULT INHIBIT INDICATOR
12 OCTL60  OCT   606060  OCTAL 60s
13 MINUS1  DEC   -1      MINUS ONE

```

- 01 Inhibits an overflow fault from occurring during the subtract operations.
- 02 Loads the accumulator with the contents of A.
- 03 Subtracts the contents of B from the accumulator. The possible results for each digit are indicated in Column W.
- 04 Stores the intermediate result in C.
- 05 Extracts an octal 60 from each digit that required a borrow. The possible results of this instruction are indicated in Column X, line 0-19 (10-19 refer to those which result in octal 60.)

- 06 An exclusive OR to storage, in effect subtracts the octal 60s in the accumulator from the corresponding digit in C. The possible results for each digit are displayed in Column Y.
- 07 Shifts the octal 60s in the accumulator right three places.
- 08
- 09 Negates the contents of the accumulator.
- 10 An add to storage, is in effect a subtraction of 06 from each digit that required a borrow, the result being placed in C. Column Z of the table reflects the possible results for each digit.

SUBTRACTION RESULTS				
LINE	W	X	Y	Z
0	11	0	11	11
1	10	0	10	10
2	07	0	07	07
3	06	0	06	06
4	05	0	05	05
5	04	0	04	04
6	03	0	03	03
7	02	0	02	02
8	01	0	01	01
9	00	0	00	00
10	77	60	17	11
11	76	60	16	10
12	75	60	15	07
13	74	60	14	06
14	73	60	13	05
15	72	60	12	04
16	71	60	11	03
17	70	60	10	02
18	67	60	7	01
19	66	60	6	00

## Data Movement

Loop Termination - (move 100 nine-bit characters from block A to block B).

### 1. Index register as a counter

	1	8	16
		LDX1	COUNT-*
		LDX2	BLKA-*
		LDX3	BLKB-*
Z		LDA	0,2,.B.0
		STA	0,3,B.0
		IACX3	0,B.1
		IACX2	0,B.1
		IACX1	-1
		TNZ	Z-*
		TRA	--
COUNT	DEC		100
BLKA	ZERO		BLOCKA,B.0
BLKB	ZERO		BLOCKB,B.0
BLOCKA	BSS		50
BLOCKB	BSS		50

NOTE: The negative indicator is not set during the execution of index register instructions. Thus, the zero indicator must be used when testing the result of an arithmetic operation with an index.

### 2. Memory cell as a counter

	1	8	16
		ILA	-COUNT
		STA	CNTR-*
		LDX2	BLKA-*
		LDX3	BLKB-*
Z		LDA	0,2,B.0
		STA	0,3,B.0
		IACX3	0,B.1
		IACX2	0,B.1
		AOS	CNTR-*
		TMI	Z-*
		TRA	--
COUNT	EQU		100
CNTR	BSS		1
BLKA	ZERO		BLOCKA,B.0
BLKB	ZERO		BLOCKB,B.0
BLOCKA	BSS		50
BLOCKB	BSS		50

3. Character position as terminator

1	8	16
	LDX2	BLKA-*
	LDX3	BLKB-*
Z	LDA	0,2,B.0
	STA	0,3,B.0
	IACX3	0,B.1
	IACX2	0,B.1
	CMPX2	END-*
	TNZ	Z-*
	TRA	--
BLKA	ZERO	BLOCKA,B.0
BLKB	ZERO	BLOCKB,B.0
END	ZERO	BLOCKA+50,B.0
BLOCKA	BSS	50
BLOCKB	BSS	50

Binary to Binary Coded Decimal Conversion Routine

FUNCTION - This routine converts any binary number having a decimal value of 32767 or less into its BCD equivalent.

CALLING SEQUENCE - TSY BINBCD-\*

RETURN

CONDITIONS - ON ENTRY - The accumulator register should contain the number to be converted.

- ON EXIT - The combined AQ register contains the BCD result right adjusted, with blanks for leading zeros.

BINBCD ZERO (RETURN)

STX1 BNBSV1-\*

SAVE REGISTERS

STX2 BNBSV2-\*

LDX1 BNBTAB-\*

LOCATION OF CONVERSION TABLE

LDX2 BNBBF-\*

LOCATION OF OUTPUT BUFFER

BNB10 NULL

ILQ 0 CLEAR COUNTER

BNB20 NULL

SBA 0,1 SIMULATE A DIVIDE

TMI BNB30-\*

END OF THE GREAT DIVIDE

IAQ 1 BUMP QUOTIENT

TRA BNB20-\*

BNB30 NULL

ADA 0,1 ADD BACK TO REALITY

ALS 3 SHIFT REMAINDER

STQ 0,2,C.0

SAVE CONVERTED DIGIT

IACX1 1 BUMP TABLE ADDRESS

IACX2 0,C.1 BUMP BUFFER POINTER

CMPX2 BNBEND-\*

IS THE CONVERSION DONE...

TNZ BNB10-\*

NO. DO IT AGAIN

LDX2 BNBBF-\*

YES. INSERT BLANKS FOR LEADING ZEROS.

ILQ BLNK

```

BNB40  NULL
      LDA      0,2,C.0
              GET NEXT DIGIT
      TNZ      BNB50-*
              NON-ZERO--FINI
      STQ      0,2,C.0
              REPLACE AUTT FOR BLANK
      IACX2    0,C.1 BUMP BUFFER POINTER
      CMPX2    BNB40-*
              NO. GET SOMEMORE
      STZ      -1,2,C.2
              SINGLE ZERO

BNB50  NULL
      LDAQ     BNBBUF-*
              ANSWER TO THE AQ REG
      LDX2     BNBSV2-*
              RESTORE REGISTERS
      LDX1     BNBSV1-*
      TRA     BINBCD-*,I EXIT
      BLNK    BOOL    000020    A BCD BLANK
      BNBSV1  ZERO    0        INDEX REGISTER ONE SAFE STORE
      BNBSV2  ZERO    0        INDEX REGISTER TWO SAFE STORE
      BNBBF   ZERO    BNBBUF,C.1
              TEMPORARY BUFFER TALLY
      BNBEND  ZERO    BNBBUF+2,C.0
              END OF THE TEMPORARY BUFFER
      BNBUFEBCI 2, 00000
              TEMPORARY BUFFER
      BNBTAB  ZERO    *+1     BINARY TO BCD CONVERSION TABLE
              DEC      10000,8000,6400,5120,4096

```

### Character Transliteration

The following example illustrates a method of transliterating each character of a card image that has been punched in the Standard Character set to the octal value of the corresponding character in the upper case ASCII set. The card origin is at IMAGE and the converted card image is stored starting at BUFFER.

```

01          LDX1  BCDBF-*    STARTING LOCATION OF BCD CARD IMAGE
02          LDX2  ACIBF-*    STARTING LOCATION OF ASCII CARD IMAGE
03          ILQ   80        INITIALIZE LOOP COUNT
04  LOOP    LDX3  0,1,C.0    LOAD A BCD CHARACTER
05          LDA   TRANS-*,I  CONVERT CHARACTER TO ASCII
06          STA   0,2,B.0    STORE CONVERTED CHARACTER IN BUFFER
07          IACX1 0,C.1
08          IACX2 0,B.1
09          IAQ   -1
10          TNZ   LOOP-*
          .
          .
          .
11  BCDBF   ZERO   IMAGE,C.0
12  ACIBF   ZERO   BUFFER,B.0
13  TRANS   IND   TABLE,3
14  IMAGE   BSS   27        BCD CARD IMAGE
15  BUFFER  BSS   40        ASCII CARD IMAGE
16  TABLE  OCT   060      BCD OCTAL REPRESENTATION 00 CHARACTER 0
17          OCT   061      01 1
18          OCT   062      02 2
19          OCT   063      03 3
20          OCT   064      04 4
21          OCT   065      05 5

```

22	OCT	066	06	6
23	OCT	067	07	7
24	OCT	070	10	8
25	OCT	071	11	9
26	OCT	133	12	[
27	OCT	043	13	#
28	OCT	100	14	@
29	OCT	072	15	:
30	OCT	076	16	>
31	OCT	077	17	?
32	OCT	040	20	Ø
33	OCT	101	21	A
34	OCT	102	22	B
35	OCT	103	23	C
36	OCT	104	24	D
37	OCT	105	25	E
38	OCT	106	26	F
39	OCT	107	27	G
40	OCT	110	30	H
41	OCT	111	31	I
42	OCT	046	32	&
43	OCT	056	33	.
44	OCT	135	34	]
45	OCT	050	35	(
46	OCT	074	36	<
47	OCT	134	37	\
48	OCT	136	40	↑
49	OCT	112	41	J
50	OCT	113	42	K
51	OCT	114	43	L
52	OCT	115	44	M
53	OCT	116	45	N
54	OCT	117	46	O
55	OCT	120	47	P
56	OCT	121	50	Q
57	OCT	122	51	R
58	OCT	055	52	-
59	OCT	044	53	\$
60	OCT	052	54	*
61	OCT	051	55	)
62	OCT	073	56	;
63	OCT	047	57	'
64	OCT	053	60	+
65	OCT	057	61	/
66	OCT	123	62	S
67	OCT	124	63	T
68	OCT	125	64	U
69	OCT	126	65	V
70	OCT	127	66	W
71	OCT	130	67	X
72	OCT	131	70	Y
73	OCT	132	71	Z
74	OCT	137	72	←
75	OCT	054	73	,
76	OCT	045	74	%
77	OCT	075	75	=
78	OCT	042	76	"
79	OCT	041	77	!





APPENDIX A

STANDARD CHARACTER SET

STANDARD CHARACTER SET	INTERNAL MACHINE CODE	OCTAL	HOLLERITH CARD CODE	ASCII PSEUDO-OP CODE	ACI PSEUDO-OP CODE
0	000000	00	0	060	060
1	000001	01	1	061	061
2	000010	02	2	062	062
3	000011	03	3	063	063
4	000100	04	4	064	064
5	000101	05	5	065	065
6	000110	06	6	066	066
7	000111	07	7	067	067
8	001000	10	8	070	070
9	001001	11	9	071	071
[	001010	12	2-8	133	133
#	001011	13	3-8	043	043
@	001100	14	4-8	100	100
:	001101	15	5-8	072	072
>	001110	16	6-8	076	076
?	001111	17	7-8	077	077
␣	010000	20	(blank)	040	040
A	010001	21	12-1	141	101
B	010010	22	12-2	142	102
C	010011	23	12-3	143	103
D	010100	24	12-4	144	104
E	010101	25	12-5	145	105
F	010110	26	12-6	146	106
G	010111	27	12-7	147	107
H	011000	30	12-8	150	110
I	011001	31	12-9	151	111
&	011010	32	12	046	046
.	011011	33	12-3-8	056	056
]	011100	34	12-4-8	135	135
(	011101	35	12-5-8	050	050
<	011110	36	12-6-8	074	074
\	011111	37	12-7-8	134	134
↑	100000	40	11-0	136	136
J	100001	41	11-1	152	112
K	100010	42	11-2	153	113
L	100011	43	11-3	154	114
M	100100	44	11-4	155	115
N	100101	45	11-5	156	116
O	100110	46	11-6	157	117
P	100111	47	11-7	160	120
Q	101000	50	11-8	161	121
R	101001	51	11-9	162	122
-	101010	52	11	055	055
\$	101011	53	11-3-8	044	044
*	101100	54	11-4-8	052	052
)	101101	55	11-5-8	051	051
;	101110	56	11-6-8	073	073
'	101111	57	11-7-8	047	047
+	110000	60	12-0	053	053
/	110001	61	0-1	057	057
S	110010	62	0-2	163	123
T	110011	63	0-3	164	124
U	110100	64	0-4	165	125
V	110101	65	0-5	166	126
W	110110	66	0-6	167	127
X	110111	67	0-7	170	130
Y	111000	70	0-8	171	131
Z	111001	71	0-9	172	132
←	111010	72	0-2-8	137	137
,	111011	73	0-3-8	054	054
%	111100	74	0-4-8	045	045
=	111101	75	0-5-8	075	075
"	111110	76	0-6-8	042	042
!	111111	77	0-7-8	041	041



APPENDIX B

CONVERSION TABLES

OCTAL-DECIMAL INTEGER TABLE

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

Octal	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

Octal	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

Octal 0400 to 0777  
Decimal 0256 to 0511

Octal	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

Octal 1400 to 1777  
Decimal 0768 to 1023

Octal	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

OCTAL-DECIMAL INTEGER TABLE (Cont.)

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

Octal	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

Octal	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

Octal	2400 to 2777
Decimal	1280 to 1535

Octal	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

Octal	3400 to 3777
Decimal	1792 to 2047

Octal	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

OCTAL-DECIMAL INTEGER TABLE (Cont.)

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

Octal	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

Octal	0	1	2	3	4	5	6	7
5000	2500	2501	2502	2503	2504	2505	2506	2507
5010	2508	2509	2510	2511	2512	2513	2514	2515
5020	2516	2517	2518	2519	2520	2521	2522	2523
5030	2524	2525	2526	2527	2528	2529	2530	2531
5040	2532	2533	2534	2535	2536	2537	2538	2539
5050	2540	2541	2542	2543	2544	2545	2546	2547
5060	2548	2549	2550	2551	2552	2553	2554	2555
5070	2556	2557	2558	2559	2560	2561	2562	2563
5100	2564	2565	2566	2567	2568	2569	2570	2571
5110	2572	2573	2574	2575	2576	2577	2578	2579
5120	2580	2581	2582	2583	2584	2585	2586	2587
5130	2588	2589	2590	2591	2592	2593	2594	2595
5140	2596	2597	2598	2599	2600	2601	2602	2603
5150	2604	2605	2606	2607	2608	2609	2610	2611
5160	2612	2613	2614	2615	2616	2617	2618	2619
5170	2620	2621	2622	2623	2624	2625	2626	2627
5200	2628	2629	2630	2631	2632	2633	2634	2635
5210	2636	2637	2638	2639	2640	2641	2642	2643
5220	2644	2645	2646	2647	2648	2649	2650	2651
5230	2652	2653	2654	2655	2656	2657	2658	2659
5240	2660	2661	2662	2663	2664	2665	2666	2667
5250	2668	2669	2670	2671	2672	2673	2674	2675
5260	2676	2677	2678	2679	2680	2681	2682	2683
5270	2684	2685	2686	2687	2688	2689	2690	2691
5300	2692	2693	2694	2695	2696	2697	2698	2699
5310	2700	2701	2702	2703	2704	2705	2706	2707
5320	2708	2709	2710	2711	2712	2713	2714	2715
5330	2716	2717	2718	2719	2720	2721	2722	2723
5340	2724	2725	2726	2727	2728	2729	2730	2731
5350	2732	2733	2734	2735	2736	2737	2738	2739
5360	2740	2741	2742	2743	2744	2745	2746	2747
5370	2748	2749	2750	2751	2752	2753	2754	2755
5380	2756	2757	2758	2759	2760	2761	2762	2763
5390	2764	2765	2766	2767	2768	2769	2770	2771
5400	2772	2773	2774	2775	2776	2777	2778	2779
5410	2780	2781	2782	2783	2784	2785	2786	2787
5420	2788	2789	2790	2791	2792	2793	2794	2795
5430	2796	2797	2798	2799	2800	2801	2802	2803
5440	2804	2805	2806	2807	2808	2809	2810	2811
5450	2812	2813	2814	2815	2816	2817	2818	2819
5460	2820	2821	2822	2823	2824	2825	2826	2827
5470	2828	2829	2830	2831	2832	2833	2834	2835
5500	2836	2837	2838	2839	2840	2841	2842	2843
5510	2844	2845	2846	2847	2848	2849	2850	2851
5520	2852	2853	2854	2855	2856	2857	2858	2859
5530	2860	2861	2862	2863	2864	2865	2866	2867
5540	2868	2869	2870	2871	2872	2873	2874	2875
5550	2876	2877	2878	2879	2880	2881	2882	2883
5560	2884	2885	2886	2887	2888	2889	2890	2891
5570	2892	2893	2894	2895	2896	2897	2898	2899
5600	2900	2901	2902	2903	2904	2905	2906	2907
5610	2908	2909	2910	2911	2912	2913	2914	2915
5620	2916	2917	2918	2919	2920	2921	2922	2923
5630	2924	2925	2926	2927	2928	2929	2930	2931
5640	2932	2933	2934	2935	2936	2937	2938	2939
5650	2940	2941	2942	2943	2944	2945	2946	2947
5660	2948	2949	2950	2951	2952	2953	2954	2955
5670	2956	2957	2958	2959	2960	2961	2962	2963
5700	2964	2965	2966	2967	2968	2969	2970	2971
5710	2972	2973	2974	2975	2976	2977	2978	2979
5720	2980	2981	2982	2983	2984	2985	2986	2987
5730	2988	2989	2990	2991	2992	2993	2994	2995
5740	2996	2997	2998	2999	3000	3001	3002	3003
5750	3004	3005	3006	3007	3008	3009	3010	3011
5760	3012	3013	3014	3015	3016	3017	3018	3019
5770	3020	3021	3022	3023	3024	3025	3026	3027
5780	3028	3029	3030	3031	3032	3033	3034	3035
5790	3036	3037	3038	3039	3040	3041	3042	3043
5800	3044	3045	3046	3047	3048	3049	3050	3051
5810	3052	3053	3054	3055	3056	3057	3058	3059
5820	3060	3061	3062	3063	3064	3065	3066	3067
5830	3068	3069	3070	3071	3072	3073	3074	3075
5840	3076	3077	3078	3079	3080	3081	3082	3083
5850	3084	3085	3086	3087	3088	3089	3090	3091
5860	3092	3093	3094	3095	3096	3097	3098	3099
5870	3100	3101	3102	3103	3104	3105	3106	3107

Octal	4400 to 4777
Decimal	2304 to 2559

Octal	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2					

OCTAL-DECIMAL INTEGER TABLE (Cont.)

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

Octal	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

Octal	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

Octal 6400 to 6777  
Decimal 3328 to 3583

Octal	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

Octal 7400 to 7777  
Decimal 3840 to 4095

Octal	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

OCTAL-DECIMAL FRACTION TABLE

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046



OCTAL-DECIMAL FRACTION TABLE (Cont.)

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL-DECIMAL FRACTION TABLE (Cont.)

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949



APPENDIX C

TABLE OF POWERS OF TWO

$2^n$	$n$	$2^{-n}$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5



APPENDIX D

TABLE OF BINARY - DECIMAL EQUIVALENTS

Maximum Decimal Integral Value	Number of Decimal Digits	Number of Bits	Maximum Decimal Fractional Value
1		1	.5
3		2	.75
7		3	.875
15	1	4	.937 5
31		5	.968 75
63		6	.984 375
127	2	7	.992 187 5
255		8	.996 093 75
511		9	.998 046 875
1 023	3	10	.999 023 437 5
2 047		11	.999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383	4	14	.999 938 964 843 75
32 767		15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071	5	17	.999 992 370 605 468 75
262 143		18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 367 187 5
1 048 575	6	20	.999 999 046 325 683 593 75
2 097 151		21	.999 999 523 162 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 607		23	.999 999 880 790 710 449 218 75
16 777 215	7	24	.999 999 940 395 355 244 609 375
33 554 431		25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727	8	27	.999 999 992 549 419 403 076 171 875
268 435 455		28	.999 999 996 274 709 701 538 085 937 5
536 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823	9	30	.999 999 999 068 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367		35	.999 999 999 970 896 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75
281 474 976 710 655		48	

This chart provides the information necessary to determine:

- a. The number of bits needed to represent a given decimal number. Use columns one and three or four and three.
- b. The number of bits needed to represent a given number of decimal digits (all nines). Use columns two and three.
- c. The maximum decimal value represented by a given number of bits, use columns one and three or three and four.



APPENDIX E

THE TWOS COMPLEMENT NUMBER SYSTEM

First consider a simple example of twos complement numbers, namely integers of three bits each, numbering the bits 0, 1, and 2, respectively, from left to right. Then the integer xyz represents the decimal quantity  $-4x+2y+z$ :

hence    011 represents +3  
           010 represents +2  
           001 represents +1  
           000 represents +0  
           111 represents -1  
           110 represents -2  
           101 represents -3  
 and       100 represents -4

Thus each decimal integer from -4 to 3 has a unique representation as a twos complement number. Bit 0 also serves as the sign-bit, since it is 0 for all positive numbers and 1 for all negative numbers. Note that 000 is a positive number.

We perform the addition  $abc+xyz$  as though  $abc$  and  $xyz$  were signless binary integers from 0 to 7, ignoring any carry out of bit 0 of the sum. If the true sum is not an integer from -4 to 3, then we have an overflow. We observe that the carry out of bit 0 = the carry out of bit 1 if, and only if, there is no overflow. In the case when  $a \neq x$ , we cannot have an overflow, since the sum ranges from -4 to 2. It follows that  $a + x = 1$  and that the carries must be equal, since we have  $0+1 = 1$  with carry 0 and  $1+1 = 0$  with carry 1. In the case when  $a = x$ , we have no overflow if, and only if, bit 0 of the sum =  $x$ . We have this equality if, and only if, the carries are equal, since we have  $0+0+0 = 0$  with carry 0 and  $1+1+1 = 1$  with carry 1. We conclude that our overflow test is a valid one. The following examples are illustrations of twos complement addition:

CARRIES	00	11	00	01	11	10
abc	110=-2	110=-2	010=+2	010=+2	110=-2	110=-2
xyz	001=+1	011=+3	001=+1	011=+3	111=-1	101=-3
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
abc+xyz	111=-1	001=+1	011=+3	101=-3	101=-3	011=+3
REMARKS	NO OVF.	NO OVF.	NO OVF.	OVF.	NO OVF.	OVF.



Say that  $uvw$  is the ones complement of  $xyz$  (and vice versa) if  $uvw+xyz = 111$ . Hence  $u+x = v+y = w+z = 1$ . Say that the quantity  $uvw+001$  is the twos complement of  $xyz$ , observing that its decimal value is:

$$\begin{aligned} -4u+2v+w+1 &= -4(1-x)+2(1-y)+(1-z)+1 \\ &= -(-4x+2y+z), \end{aligned}$$

or minus the value of  $xyz$ . For this reason we call  $xyz$  a twos complement number. We perform the subtraction  $abc-xyz$  by the triple addition  $abc+uvw+001$  (in effect, by adding  $abc$  and  $uvw$  with a forced carry of 1 into the low order bit 2). We use the same overflow test as for addition. Note that  $000-000 = 000$  (no overflow) and that  $000-100 = 100$  (overflow). Hence 000 is its own twos complement, and 100 does not have a proper twos complement. Note the conspicuous absence of a -0 from the twos complement system on the previous page.

We may generalize the above discussion to include twos complement integers of  $n$  bits each. The integer

$$X_0X_1X_2\dots X_{n-2}X_{n-1}$$

represents the decimal quantity as:

$$-2^{n-1}X_0+2^{n-2}X_1+2^{n-3}X_2+\dots+2X_{n-2}=X_{n-1}$$

The same rules as above hold for addition, overflow, complementation, and subtraction. There are several choices for  $n$ :

- $n = 8$  for exponent fields
- $n = 18$  for address fields
- $n = 18$  for single-precision integers
- $n = 36$  for double-precision integers

The use of twos complement numbers offers many advantages.

1. It eliminates housekeeping before and after addition and subtraction in the computer hardware.
2. It permits addition and subtraction modulo  $2^n$ , since we may always consider a number to be signless.
3. It permits addition of a quantity to a field of a word, without any need to worry about the sign-bit. (In the sign-magnitude system, one would add the quantity if the sign were positive, and subtract the quantity if the sign were negative.)
4. It makes zero a unique positive number.
5. It is compatible with index register arithmetic.

The user must always remember that the computer is a twos complement machine, especially when converting programs that were originally written for a machine with sign-magnitude or ones complement arithmetic. For example, the sign magnitude convention of "changing sign" corresponds to the twos complement convention of "negation" (or "complementation").

INDEX

- \$ 355MAP
  - \$ 355MAP Control Card 1-3
- \$ ALTER
  - \$ ALTER Control Card 1-5
- \$ DKEND
  - \$ DKEND card 1-2
  - \$ DKEND control card 1-7
- \$ ENDJOB
  - \$ ENDJOB Control Card 1-5
- \$ LIMITS
  - \$ LIMITS Control Card 1-4
- \$ OBJECT
  - \$ OBJECT card 1-2
- \$ SNUMB
  - \$ SNUMB Control Card 1-3
- \$ UPDATE
  - \$ UPDATE Control Card 1-5
- \*\*\*EOF
  - \*\*\*EOF Control Card 1-6
- A\*
  - Alter File (A\*) 1-5
- ABORT
  - unconditional abort 3-6
- ABS
  - ABS - Output Absolute Text 4-12
  - ABS pseudo-operation 4-12
- ABSOLUTE 2-6
  - ABS - Output Absolute Text 4-12
  - absolute binary cards 4-13
  - Absolute Object (Binary) Deck 1-11
  - Absolute Text Card 1-11
  - absolute text cards 1-11
  - RELOCATABLE AND ABSOLUTE ASSEMBLIES 1-2
  - Relocatable and Absolute Expression 2-10
- Accumulator 3-6
- ACI
  - ACI and ACIC pseudo-operations 4-32
- ACI (cont)
  - ASCII ASCIIC ACI ACIC - ASCII Coded Information 4-31
- ACIC
  - ACI and ACIC pseudo-operations 4-32
  - ASCII ASCIIC ACI ACIC - ASCII Coded Information 4-31
- ACTIVE
  - ACTIVE STATUS FORMAT 5-10
- ADA
  - ADA - Add to A 3-34
- ADAPTER
  - COMPUTER MONITOR ADAPTER (CMA) 5-64
  - CONTROL CONSOLE ADAPTER (CCA) 5-70
  - DIRECT INTERFACE ADAPTER (DIA) 5-14
  - HIGH SPEED LINE ADAPTER (HSLA) 5-23
  - INTERCOMPUTER ADAPTER (ICA) 5-4
  - LOW SPEED LINE ADAPTER (LSLA) 5-31
  - PERIPHERAL SUBSYSTEM ADAPTER (PSA) 5-38
- ADAQ
  - ADAQQ- ADD to Q 3-35
- ADCXN
  - ADCXn - Add Character Address to Xn 3-35
- ADD
  - ADA - Add to A 3-34
  - ADAQ - ADD to Q 3-35
  - ADCXn - Add Character Address to Xn 3-35
  - ADD INSTRUCTIONS 3-34
  - ADQ - Add to Q 3-35
  - AOS - Add One to Storage 3-37
  - ASA - Add A to Storage 3-37
  - Fractional Add 3-50
  - Fractional Add (FA) function 3-35
  - GROUP 1 IMMEDIATE ADD INSTRUCTIONS 3-49
  - IAA - Immediate Add to A 3-49
  - IACXn - Immediate Add Character Address to Xn 3-50
  - IAQ - Immediate Add to Q 3-50
- Addition 3-2
  - Arithmetic - Addition 3-23
  - BCD Addition 5-85

ADDITION (cont)  
 Character Address Addition Rules 3-20

ADDRESS  
 ADCXn - Add Character Address to Xn 3-35  
 ADDRESS FORMATION 3-14  
 Base Address Word 5-26, 5-38  
 Basic Level Effective Address Formation Rules 3-15  
 CCW ADDRESS FORMATION 5-56  
 character address 3-14  
 Character Address Addition Rules 3-20  
 effective address (Y\*\*) 3-15  
 EFFECTIVE ADDRESS AND MEMORY LOCATION SYMBOLS 3-28  
 Effective Address Formation Examples 3-19  
 IACXn - Immediate Add Character Address to Xn 3-50  
 INDIRECT CONTROL WORD BASE ADDRESS PATCH 5-64  
 Indirect Level Effective Address Formation Rules 3-21  
 indirect vector address 3-10  
 MAILBOX ADDRESSES 5-5  
 RIA - Read Interrupt Address 3-53  
 word address 3-14

ADDRESSING  
 CHARACTER ADDRESSING - BASIC LEVEL 3-17  
 CHARACTER ADDRESSING - INDIRECT LEVEL 3-22  
 Character Control Character Addressing 5-27  
 forward and backward addressing 3-17  
 WORD ADDRESSING - BASIC LEVEL 3-16  
 WORD ADDRESSING - INDIRECT LEVEL 3-21

ADMISSIBILITY  
 admissibility of relocation 2-10

ADQ  
 ADQ - Add to Q 3-35

ALGEBRAIC  
 Algebraic Expressions 2-8  
 algebraic instruction 3-2  
 Evaluation of Algebraic Expressions 2-8

ALLOCATION  
 Memory allocation 4-1  
 MEMORY ALLOCATION PSEUDO-OPERATIONS 4-36

ALP  
 ALP - A Left Parity Rotate 3-60

ALPHANUMERIC  
 Alphanumeric Data 3-2  
 Alphanumeric Literals 2-14

ALR  
 ALR - A Left Rotate 3-60

ALS  
 ALS - A Left Shift 3-58

ALTER  
 Alter File (A\*) 1-5

ANA  
 ANA - AND to A 3-41

ANSA  
 ANSA - AND to Storage A 3-42

AOS  
 AOS - Add One to Storage 3-37

AQ  
 AQ register 3-6  
 CQA - Copy AQ into A 3-63  
 SBAQ - Subtract from AQ 3-38  
 STAQ - Store AQ 3-32

AREA  
 Length of Blank Common area 1-8

ARG 2-3  
 ARG - Argument--Generate Zero Operation Code Computer Word 4-41  
 ARG pseudo-operation 4-41

ARITHMETIC  
 Arithmetic - Addition 3-23  
 Arithmetic - Division 3-23  
 Arithmetic - Multiplication 3-23  
 Arithmetic - Subtraction 3-23

ARL  
 ARL - A Right Logic 3-59

ARS  
 ARS - A Right Shift 3-57

ASA  
 ASA - Add A to Storage 3-37

ASCII  
 ASCII and ASCIIIC pseudo-operations 4-31  
 ASCII ASCIIIC ACI ACIC - ASCII Coded Information 4-31  
 PARITY ON/OFF - ASCII Parity Control 4-7  
 SACI - Symbolic ASCII Information 4-32

ASCIIIC  
 ASCII and ASCIIIC pseudo-operations 4-31  
 ASCII ASCIIIC ACI ACIC - ASCII Coded Information 4-31

ASSEMBLER  
 ASSEMBLER OUTPUTS 1-6

ASSEMBLY  
 Assembly Listing 1-12

## ASSEMBLY (cont)

Conditional assembly 4-1  
Date of Assembly 1-7  
END - End of Assembly 4-15  
MAXSZ - Maximum Size of Assembly  
4-42  
RELOCATABLE AND ABSOLUTE ASSEMBLIES  
1-2  
Time of Assembly 1-7

## ASSIGNMENT

PORT ASSIGNMENT FOR FNPs 5-5

## ASSIGNMENT SWITCHES

INTERRUPT CELL ASSIGNMENT SWITCHES  
5-5

## ASTERISK

asterisk (\*) in column 1 4-10  
Asterisk Used as an Element 2-7

## BACKWARD

forward and backward addressing  
3-17

## BASE

BASE - Force Location Counter to a  
Multiple Power of 2 4-26  
Base Address Word 5-26, 5-38  
BASE pseudo-operation 4-26  
INDIRECT CONTROL WORD BASE ADDRESS  
PATCH 5-64

## BASIC LEVEL

Basic Level Effective Address  
Formation Rules 3-15  
CHARACTER ADDRESSING - BASIC LEVEL  
3-17  
WORD ADDRESSING - BASIC LEVEL 3-16

## BCD

BCD Addition 5-85  
BCD Subtraction 5-86  
DCARD - Punch BCD Card 4-15

## BCI

BCI - Binary Coded Decimal  
Information 4-29  
BCI pseudo-operation 4-29

## BEGIN

BEGIN - Origin of a Location  
Counter 4-19  
BEGIN pseudo-operation 4-19

## BFS

BFS - Block Followed by Symbol 4-36  
BFS pseudo-operation 4-36

## BINARY

absolute binary cards 4-13  
Absolute Object (Binary) Deck 1-11  
BCI - Binary Coded Decimal  
Information 4-29  
Binary Decks 1-7  
BINARY SYNCHRONIZATION STATUS 5-31  
BINARY SYNCHRONOUS CHANNEL (BSC)  
5-58

## BINARY (cont)

Binary to Binary Coded Decimal  
Conversion Routine 5-89  
column binary relocatable text card  
1-10  
FUL - Output Full Binary Text 4-12

## BITS

Relocation bits 1-12

## BLANK

Blank Common 1-15  
length of Blank Common 1-9  
Length of Blank Common area 1-8

## BLOCK

BFS - Block Followed by Symbol 4-36  
BLOCK - Block Common 4-36  
BLOCK pseudo-operation 4-36  
BSS - Block Started by Symbol 4-36

## BOOLEAN

BOOL - Boolean 4-22  
BOOL pseudo-operation 2-8, 4-22  
Boolean Expressions 2-8  
BOOLEAN INSTRUCTIONS 3-41  
Boolean Operations 3-23  
Evaluation of Boolean Expressions  
2-9  
GROUP 1 IMMEDIATE BOOLEAN  
INSTRUCTIONS 3-54

## BSC

BINARY SYNCHRONOUS CHANNEL (BSC)  
5-58

## BSS

BSS - Block Started by Symbol 4-36  
BSS pseudo-operation 4-36

## BUILT-IN

SYSTEM (BUILT-IN) SYMBOLS 4-62

## CALL

CALL - Call Subroutines 4-57  
CALL pseudo-operation 4-57

## CANA

CANA - Compare AND with A 3-44

## CAQ

CAQ - Copy A into Q 3-64

## CARD

\$ 355MAP Control Card 1-3  
\$ ALTER Control Card 1-5  
\$ DKEND card 1-2  
\$ DKEND control card 1-7  
\$ ENDJOB Control Card 1-5  
\$ LIMITS Control Card 1-4  
\$ OBJECT card 1-2  
\$ SNUMB Control Card 1-3  
\$ UPDATE Control Card 1-5  
\*\*\*EOF Control Card 1-6  
absolute binary cards 4-13  
Absolute Text Card 1-11  
absolute text cards 1-11  
CARD READER 5-71

CARD (cont)

- column binary relocatable text card 1-10
- DCARD - Punch BCD Card 4-15
- DUP - Duplicate Cards 4-35
- NO END CARD ON INPUT FILE 1-16
- NOT ENOUGH CARDS TO BE SKIPPED 1-16
- PCC ON/OFF - Print Control Cards 4-5
- Preface Card 1-8
- Preface Card Listing 1-15
- PUNCH ON/OFF - Control Card Output 4-8
- Relocatable Text Card 1-10
- Reproduction of the symbolic card 1-13
- Summary of Symbolic Card Format 2-5
- TCD - Punch Transfer Card 4-13
- TOO MANY CARDS TO BE DUPLICATED 1-16
- transfer card 1-11, 1-12

CARRY

- Carry Indicator 3-13
- TNC - Transfer on No Carry 3-47
- Transfer on No Carry (TNC) 3-13

CAXN

- CAXn - Copy A into Xn 3-64

CCA

- CONTROL CONSOLE ADAPTER (CCA) 5-70

CCW

- CCW ADDRESS FORMATION 5-56
- CHARACTER CONTROL WORD (CCW) 5-55

CELL

- EMERGENCY INTERRUPT CELL NUMBER 5-6
- INTERRUPT CELL ASSIGNMENT SWITCHES 5-5
- Interrupt Cells 5-75
- SIC - Set Interrupt Cells 3-54

CENTRAL

- Central System Control Word Formats 5-12, 5-22
- Central System Interface 5-5, 5-15

CHANNEL

- BINARY SYNCHRONOUS CHANNEL (BSC) 5-58
- CHANNEL NUMBER PATCH 5-64
- CIOC - Connect Input/Output Channel 3-48
- Control Word Memory Map (Example for Channel 06) 5-26
- DIRECT CHANNEL PROGRAMMING 5-2
- DOCUMENT HANDLER CHANNEL (DHC) 5-49
- I/O Channel Select Register 3-12
- Input/output channel select 3-6
- input/output channel select register 3-6
- LOGICAL CHANNEL DCW LIST 5-42
- SEL - Select Input/Output Channel 3-53
- TIMER AND SWITCH CHANNEL 5-73

CHARACTER

- ADCXn - Add Character Address to Xn 3-35
- character address 3-14
- Character Address Addition Rules 3-20
- CHARACTER ADDRESSING - BASIC LEVEL 3-17
- CHARACTER ADDRESSING - INDIRECT LEVEL 3-22
- Character Control Character Addressing 5-27
- Character Control Table 5-27
- CHARACTER CONTROL WORD (CCW) 5-55
- Character Set 2-5
- Character Transliteration 5-90
- Command Characters 5-35
- IACXn - Immediate Add Character Address to Xn 3-50
- IND/A T Character 1-14
- Status Characters 5-36

CHECK

- Divide Check Fault 3-8

CIOC

- CIOC - Connect Input/Output Channel 3-48

CLASS

- Instruction Repertoire by Functional Class 3-23
- Mnemonics by functional class 3-22

CMA

- CMA Control Words 5-65
- COMPUTER MONITOR ADAPTER (CMA) 5-64

CMPA

- CMPA - Compare with A 3-43

CMPQ

- CMPQ - Compare with Q 3-44

CMPXN

- CMPXn - Compare with Xn 3-45

CODE

- ARG - Argument--Generate Zero Operation Code Computer Word 4-41
- ASCII ASCIIC ACI ACIC - ASCII Coded Information 4-31
- BCI - Binary Coded Decimal Information 4-29
- Binary to Binary Coded Decimal Conversion Routine 5-89
- CODING EXAMPLES 5-84
- Illegal Operation Code Fault 3-8
- SERVICE CODES - MPC TO PSA 5-45
- SERVICE CODES - PSA TO MPC 5-45
- SYMBOLIC CODING FORM 2-1, 2-2

COLUMN

- asterisk (\*) in column 1 4-10
- column binary relocatable text card 1-10

COMDK 1-4

COMMAND  
 Command Characters 5-35  
 Command PCW0 PCW1 5-25  
 Command PCW2 PCW3 5-25  
 Commands PCW0 PCW1 5-33  
 MPC COMMANDS 5-46  
 SPECIAL CONTROLLER COMMANDS 5-47

COMMENT  
 Comments Field 2-4  
 Optional Comment Sequence Option 1-7

COMMON 2-6  
 Blank Common 1-15  
 BLOCK - Block Common 4-36  
 Labeled Common 1-9  
 length of Blank Common 1-9  
 Length of Blank Common area 1-8  
 SYMDEF SYMREF and Labeled Common symbols 1-8  
 SYMDEF SYMREF Labeled Common 1-15  
 COMMON PERIPHERAL STATUS FORMAT 5-74

COMPARATIVE  
 ICANA - Immediate Comparative AND with A 3-56

COMPARE  
 CANA - Compare AND with A 3-44  
 CMPA - Compare with A 3-43  
 CMPQ - Compare with Q 3-44  
 CMPXn - Compare with Xn 3-45  
 COMPARE INSTRUCTIONS 3-43  
 GROUP 1 IMMEDIATE COMPARE INSTRUCTIONS 3-55  
 ICPA - Immediate Compare A 3-55

Comparison 3-23

COMPUTER  
 ARG - Argument--Generate Zero Operation Code Computer Word 4-41  
 COMPUTER MONITOR ADAPTER (CMA) 5-64

CONFIGURATION  
 Configuration Patching 5-64  
 CONFIGURATION STATUS FORMAT 5-9, 5-19  
 System Controller Port Configurations 5-9

CONNECT  
 CIOC - Connect Input/Output Channel 3-48  
 CONNECT PCW (OPERATIONAL MODE) 5-38

CONSOLE  
 CONTROL CONSOLE ADAPTER (CCA) 5-70

CONTENT  
 REGISTER POSITIONS AND CONTENT SYMBOLS 3-29

CONTINUATION  
 ETC - Continuation 4-62

CONTROL 4-1  
 \$ 355MAP Control Card 1-3  
 \$ ALTER Control Card 1-5  
 \$ DKEND control card 1-7  
 \$ ENDJOB Control Card 1-5  
 \$ LIMITS Control Card 1-4  
 \$ SNUMB Control Card 1-3  
 \$ UPDATE Control Card 1-5  
 \*\*\*EOF Control Card 1-6  
 Central System Control Word Formats 5-12, 5-22  
 Character Control Character Addressing 5-27  
 Character Control Table 5-27  
 CHARACTER CONTROL WORD (CCW) 5-55  
 CMA Control Words 5-65  
 Control Word Memory Map (Example for Channel 06) 5-26  
 Control Words 5-25, 5-32  
 DATA CONTROL WORD (DCW) 5-7, 5-41  
 DATA CONTROL WORD FORMAT PSEUDO-OPERATIONS 4-44  
 DATANET FNP Control Word Formats 5-6, 5-16  
 DCW - I/O Control Word Generator 4-45  
 GROUP 1 INTERRUPT CONTROL INSTRUCTIONS 3-53  
 GROUP 2 INTERRUPT CONTROL INSTRUCTIONS 3-65  
 ICW - I/O Control Word Generator 4-44  
 Indirect Control Word 5-25, 5-33  
 INDIRECT CONTROL WORD (ICW) 5-66  
 INDIRECT CONTROL WORD BASE ADDRESS PATCH 5-64  
 INSTRUCTION DATA CONTROL WORD (IDCW) 5-40  
 Interrupt Control 3-25  
 LIST INDIRECT CONTROL WORD (LICW) 5-7, 5-17  
 LIST ON/OFF - Control Output Listing 4-5  
 ON/OFF Switch Type Control Pseudo-Operation 4-4  
 PARITY ON/OFF - ASCII Parity Control 4-7  
 PCC ON/OFF - Print Control Cards 4-5  
 PERIPHERAL CONTROL WORD 5-1  
 PERIPHERAL CONTROL WORD (PCW) 5-6, 5-16, 5-22, 5-50, 5-53, 5-65  
 PROGRAM INTERRUPT CONTROL 5-2  
 PUNCH ON/OFF - Control Card Output 4-8  
 Transfer of Control 3-24

CONTROL WORD  
 PERIPHERAL CONTROL WORD (PCW) 5-12

CONTROLLER  
 SPECIAL CONTROLLER COMMANDS 5-47  
 System Controller Port Configurations 5-9

CONTROL PSEUDO-OPERATIONS 4-4, 4-9

CONVERSION  
 Binary to Binary Coded Decimal  
 Conversion Routine 5-89

COPY 1-4  
 CAQ - Copy A into Q 3-64  
 CAXn - Copy A into Xn 3-64  
 CQA - Copy AQ into A 3-63  
 CXnA - Copy Xn into A 3-64  
 GROUP 2 DATA MOVEMENT COPY  
 INSTRUCTIONS 3-63

COPYRIGHT  
 CPR - Copyright 4-11

COUNTER  
 BASE - Force Location Counter to a  
 Multiple Power of 2 4-26  
 BEGIN - Origin of a Location  
 Counter 4-19  
 EIGHT - Force Location Counter to a  
 Multiple of 8 4-25  
 EVEN - Force Location Counter Even  
 4-25  
 instruction counter 3-6  
 Location counter 4-1  
 LOCATION COUNTER PSEUDO-OPERATION  
 4-19  
 ODD - Force Location Counter Odd  
 4-25  
 USE - Use Multiple Location  
 Counters 4-19

CPR  
 CPR - Copyright 4-11  
 CPR pseudo-operation 4-11

CQA  
 CQA - Copy AQ into A 3-63

CREATED  
 CRSM ON/OFF - Created Symbols 4-53  
 ORGCSM - Origin Created Symbols  
 4-53

CRSM  
 CRSM ON/OFF - Created Symbols 4-53  
 CRSM pseudo-operation 4-53

CURRENT  
 DATE - Current Date 4-41

CXNA  
 CXnA - Copy Xn into A 3-64

DATA  
 Alphanumeric Data 3-2  
 DATA CONTROL WORD (DCW) 5-7, 5-41  
 DATA CONTROL WORD FORMAT  
 PSEUDO-OPERATIONS 4-44  
 Data Generating Pseudo-Operation  
 1-14  
 DATA GENERATING PSEUDO-OPERATIONS  
 4-26  
 Data generation 4-1  
 Data Movement 3-25, 5-88  
 Data Movement - Load 3-23  
 Data Movement - Store 3-23

DATA (cont)  
 DATA RESPONSE TIMER DURATION PATCH  
 5-64  
 Data Transfer 5-67  
 Double-Precision Data 3-1  
 GROUP 2 DATA MOVEMENT COPY  
 INSTRUCTIONS 3-63  
 GROUP 2 DATA MOVEMENT NORMALIZE  
 INSTRUCTIONS 3-62  
 GROUP 2 DATA MOVEMENT SHIFT  
 INSTRUCTIONS 3-57  
 INSTRUCTION DATA CONTROL WORD  
 (IDCW) 5-40  
 Relocation data 1-10  
 SAVE - Save--Return Linkage Data  
 4-59  
 Single-Precision Data 3-1

DATANET FNP  
 DATANET FNP Control Word Formats  
 5-6, 5-16  
 DATANET FNP GENERAL MEMORY MAP 5-75  
 DATANET FNP Interface 5-4

DATE  
 DATE - Current Date 4-41  
 Date of Assembly 1-7  
 DATE pseudo-operation 4-41  
 TTLDAT - Title Date 4-41

DCARD  
 DCARD - Punch BCD Card 4-15  
 DCARD pseudo-operation 4-15

DCW  
 DATA CONTROL WORD (DCW) 5-7, 5-41  
 DCW - I/O Control Word Generator  
 4-45  
 DCW pseudo-operation 4-45  
 LOGICAL CHANNEL DCW LIST 5-42

DEC  
 DEC - Decimal 4-27  
 DEC pseudo-operation 4-27

DECIMAL  
 BCI - Binary Coded Decimal  
 Information 4-29  
 Binary to Binary Coded Decimal  
 Conversion Routine 5-89  
 DEC - Decimal 4-27  
 Decimal Literals 2-13

DECK 1-4  
 Absolute Object (Binary) Deck 1-11  
 Binary Decks 1-7  
 Relocatable Object Deck 1-8  
 TYPICAL DECK SETUPS 1-6

DEFINITION  
 Definition of the Macro Prototype  
 4-46  
 OPD - Operation Definition 4-16  
 Symbol definition 4-1  
 SYMDEF - Symbol Definition 4-23  
 VFD - Variable Field Definition  
 4-29

DELAY  
   DIS - Delay Until Interrupt Signal 3-65

DELETE  
   DELM - Delete Macro Named 4-54

DELIMITER  
   Subprogram delimiter 1-2

DELM  
   DELM - Delete Macro Named 4-54  
   DELM pseudo-operation 4-54

DEVICE  
   MPC DEVICE STATUS 5-47

DHC  
   DOCUMENT HANDLER CHANNEL (DHC) 5-49

DIA  
   DIRECT INTERFACE ADAPTER (DIA) 5-14

DIRECT  
   DIRECT CHANNEL PROGRAMMING 5-2  
   DIRECT INTERFACE ADAPTER (DIA) 5-14

DIS  
   DIS - Delay Until Interrupt Signal 3-65

DIVIDE  
   Divide Check Fault 3-8  
   DIVIDE INSTRUCTIONS 3-40  
   DVF - Divide Fraction 3-40

DIVISION  
   Arithmetic - Division 3-23

DOCUMENT HANDLER  
   DOCUMENT HANDLER CHANNEL (DHC) 5-49

DOUBLE-PRECISION  
   Double-Precision Data 3-1  
   Double-Precision Floating-Point 2-13

DRD236/DHU1600 5-52

DUMP 1-4

DUP  
   DUP - Duplicate Cards 4-35  
   DUP pseudo-operation 4-35

DUPLICATE  
   DUP - Duplicate Cards 4-35  
   TOO MANY CARDS TO BE DUPLICATED 1-16

DVF  
   DVF - Divide Fraction 3-40

E/O/8  
   E/O/8 Field 2-3

EDIT  
   EDITP - Edit Print Lines 4-9

EDITP  
   EDITP - Edit Print Lines 4-9  
   EDITP pseudo-operation 4-9

EFFECTIVE  
   Basic Level Effective Address Formation Rules 3-15  
   effective address (Y\*\*) 3-15  
   EFFECTIVE ADDRESS AND MEMORY LOCATION SYMBOLS 3-28  
   Effective Address Formation Examples 3-19  
   Indirect Level Effective Address Formation Rules 3-21

EIGHT  
   EIGHT - Force Location Counter to a Multiple of 8 4-25  
   EIGHT pseudo-operation 4-25

EJECT  
   EJECT - Restore Output Listing 4-9  
   EJECT pseudo-operation 4-9

ELEMENT  
   Asterisk Used as an Element 2-7  
   Elements 2-6

EMERGENCY  
   EMERGENCY INTERRUPT CELL NUMBER 5-6

ENABLE  
   Enable Interrupt (ENI) 3-14  
   ENI (Enable Interrupts) 3-11  
   ENI - Enable Interrupt 3-65  
   RIER - Read Interrupt Level Enable Register 3-53  
   SIER (Set Interrupt level Enable Register) 3-11  
   SIER - Set Interrupt Level Enable Register 3-54

END  
   END - End Macro Prototype 4-47  
   END - End of Assembly 4-15  
   END pseudo-operation 1-12, 4-15  
   NO END CARD ON INPUT FILE 1-16

ENDM  
   ENDM pseudo-operation 4-47

ENI  
   Enable Interrupt (ENI) 3-14  
   ENI (Enable Interrupts) 3-11  
   ENI - Enable Interrupt 3-65

EOF  
   UNEXPECTED EOF ON INTERMEDIATE FILE 1-16

EQU  
   EQU - Equal To 4-21  
   EQU pseudo-operation 4-21

EQUAL  
   EQU - Equal To 4-21



EQUIVALENCE  
 FEQU - Special FORTRAN Equivalence  
 4-21

ERA  
 ERA - EXCLUSIVE OR to A 3-43

ERROR  
 Error Flag 1-12  
 Error Flags 1-16  
 ERROR IN MACRO EXPANSION 1-16  
 error message 1-16  
 MAP error flags 1-16  
 Memory Parity Error 3-7  
 Memory Parity Error Fault 3-8  
 Parity Error Indicator 3-13  
 PSA ERROR SUMMARY 5-44

ERSA  
 ERS - EXCLUSIVE OR to Storage A  
 3-43

ETC  
 ETC - Continuation 4-62  
 ETC pseudo-operation 4-62

EVALUATION  
 Evaluation of Algebraic Expressions  
 2-8  
 Evaluation of Boolean Expressions  
 2-9

EVEN  
 EVEN - Force Location Counter Even  
 4-25  
 EVEN pseudo-operation 4-25

EXAMPLE  
 CODING EXAMPLES 5-84  
 Control Word Memory Map (Example  
 for Channel 06) 5-26  
 Effective Address Formation  
 Examples 3-19

EXCLUSIVE  
 ERA - EXCLUSIVE OR to A 3-43  
 ERS - EXCLUSIVE OR to Storage A  
 3-43  
 IERS - Immediate EXCLUSIVE OR to A  
 3-55

EXECUTION  
 EXECUTION NOT POSSIBLE NO SYMDEF  
 1-16

EXPANSION  
 ERROR IN MACRO EXPANSION 1-16  
 MACRO EXPANSION TABLE OVERFLOW 1-16  
 PMC ON/OFF - Print Macro Expansion  
 4-8

EXPRESSION  
 Algebraic Expressions 2-8  
 Boolean Expressions 2-8  
 Evaluation of Algebraic Expressions  
 2-8  
 Evaluation of Boolean Expressions  
 2-9

EXPRESSION (cont)  
 EXPRESSIONS IN GENERAL 2-6  
 Relocatable and Absolute Expression  
 2-10  
 Special Relocatable Expressions  
 2-11

EXTERNAL  
 LDEX - Load External Register 3-48  
 LOAD EXTERNAL FORMAT (LDEX) 5-52  
 STEX - Store External Register 3-49  
 STORE EXTERNAL FORMAT (STEX) 5-53

FA  
 Fractional Add (FA) function 3-35

FAULT  
 Divide Check Fault 3-8  
 FAULTS 3-6  
 Faults (internal interrupts) 3-6  
 forced TSY fault 3-7  
 Illegal Operation Code Fault 3-8  
 Illegal Program Interrupt Fault 3-8,  
 3-10  
 IOC Fault Status Locations 5-84  
 IOM FAULTS 5-3  
 Memory Parity Error Fault 3-8  
 Overflow Fault 3-8  
 Overflow Fault Inhibit Indicator  
 3-13  
 Overflow faults 3-7  
 Parity Fault Inhibit 3-8  
 Parity Fault Inhibit Indicator 3-13  
 Power Shutdown Beginning Fault 3-7  
 Power-On Restart Fault 3-7  
 PROCESSOR FAULT SWITCHES 5-5  
 Processor Fault Vectors 5-83  
 Processor Faults 3-7

FEQU  
 FEQU - Special FORTRAN Equivalence  
 4-21  
 FEQU pseudo-operation 4-21

FIELD  
 Comments Field 2-4  
 E/O/8 Field 2-3  
 Identification Field 2-5  
 Location Field 2-3  
 MARK - Specify Symbol in Location  
 Field 4-40  
 Operation Field 2-3  
 Variable Field 2-3  
 Variable Field Literals 2-16  
 VFD - Variable Field Definition  
 4-29

FILE  
 Alter File (A\*) 1-5  
 NO END CARD ON INPUT FILE 1-16  
 UNEXPECTED EOF ON INTERMEDIATE FILE  
 1-16

Fixed-Point 2-14

FLAG  
 Error Flag 1-12  
 Error Flags 1-16

**FLAG (cont)**  
 MAP error flags 1-16

**FLOATING-POINT**  
 Double-Precision Floating-Point 2-13  
 Single-Precision Floating-Point 2-13

**FNP**  
 PORT ASSIGNMENT FOR FNPs 5-5

**FORCE**  
 BASE - Force Location Counter to a Multiple Power of 2 4-26  
 EIGHT - Force Location Counter to a Multiple of 8 4-25  
 EVEN - Force Location Counter Even 4-25  
 ODD - Force Location Counter Odd 4-25

**FORCED**  
 forced TSY fault 3-7

**FORM**  
 SYMBOLIC CODING FORM 2-1, 2-2

**FORMAT**  
 ACTIVE STATUS FORMAT 5-10  
 Actual Status Word Format 5-13  
 Central System Control Word Formats 5-12, 5-22  
 COMMON PERIPHERAL STATUS FORMAT 5-74  
 CONFIGURATION STATUS FORMAT 5-9, 5-19  
 DATA CONTROL WORD FORMAT PSEUDO-OPERATIONS 4-44  
 DATANET FNP Control Word Formats 5-6, 5-16  
 format of the full listing 1-13  
 Full Listing Format 1-12  
 Listing Format 1-14  
 LOAD EXTERNAL FORMAT (LDEX) 5-52  
 PCW Format 5-33  
 PSA Word Formats 5-38  
 SPECIAL WORD FORMAT PSEUDO-OPERATIONS 4-40  
 STATUS WORD FORMAT 5-11, 5-51  
 STATUS WORD FORMATS 5-42  
 STORE EXTERNAL FORMAT (STEX) 5-53  
 Summary of Symbolic Card Format 2-5

**FORMATION**  
 ADDRESS FORMATION 3-14  
 Basic Level Effective Address Formation Rules 3-15  
 CCW ADDRESS FORMATION 5-56  
 Effective Address Formation Examples 3-19  
 Indirect Level Effective Address Formation Rules 3-21

**FORTRAN**  
 FEQU - Special FORTRAN Equivalence 4-21

**FORWARD**  
 forward and backward addressing 3-17

**FRACTION**  
 DVF - Divide Fraction 3-40  
 MPF - Multiply Fraction 3-39

**FRACTIONAL**  
 Fractional Add 3-50  
 Fractional Add (FA) function 3-35

**FUL**  
 FUL - Output Full Binary Text 4-12  
 FUL pseudo-operation 4-12

**FUNCTION**  
 Fractional Add (FA) function 3-35

**FUNCTIONAL**  
 Instruction Repertoire by Functional Class 3-23  
 Mnemonics by functional class 3-22

**GENERATION**  
 Data generation 4-1

**GENERATOR**  
 DCW - I/O Control Word Generator 4-45  
 ICW - I/O Control Word Generator 4-44

**GMAC** 1-4

**GROUP 1**  
 GROUP 1 IMMEDIATE ADD INSTRUCTIONS 3-49  
 GROUP 1 IMMEDIATE BOOLEAN INSTRUCTIONS 3-54  
 GROUP 1 IMMEDIATE COMPARE INSTRUCTIONS 3-55  
 GROUP 1 IMMEDIATE LOAD INSTRUCTIONS 3-52  
 GROUP 1 INTERRUPT CONTROL INSTRUCTIONS 3-53  
 Group 1 Nonmemory Instructions 3-5

**GROUP 2**  
 GROUP 2 DATA MOVEMENT COPY INSTRUCTIONS 3-63  
 GROUP 2 DATA MOVEMENT NORMALIZE INSTRUCTIONS 3-62  
 GROUP 2 DATA MOVEMENT SHIFT INSTRUCTIONS 3-57  
 GROUP 2 INTERRUPT CONTROL INSTRUCTIONS 3-65  
 GROUP 2 MISCELLANEOUS INSTRUCTIONS 3-65  
 Group 2 Nonmemory Instructions 3-5

**HEAD**  
 HEAD - Heading 4-13  
 HEAD pseudo-operation 4-13

**HEADING**  
 HEAD - Heading 4-13

HIGH SPEED  
HIGH SPEED LINE ADAPTER (HSLA) 5-23

HSLA  
HIGH SPEED LINE ADAPTER (HSLA) 5-23

I/O  
DCW - I/O Control Word Generator 4-45  
I/O Channel Select Register 3-12  
ICW - I/O Control Word Generator 4-44

IAA  
IAA - Immediate Add to A 3-49

IACXN  
IACXn - Immediate Add Character Address to Xn 3-50

IANA  
IANA - Immediate AND to A 3-54

IAQ  
IAQ - Immediate Add to Q 3-50

IC  
TSY - Transfer and Store IC in Y 3-46

ICA  
INTERCOMPUTER ADAPTER (ICA) 5-4

ICANA  
ICANA - Immediate Comparative AND with A 3-56

ICMPA  
ICMPA - Immediate Compare A 3-55

ICW  
ICW - I/O Control Word Generator 4-44  
ICW pseudo-operation 4-44  
INDIRECT CONTROL WORD (ICW) 5-66

IDCW  
INSTRUCTION DATA CONTROL WORD (IDCW) 5-40

IDENTIFICATION  
Identification Field 2-5  
MACRO - MACRO Identification 4-47

IDRP  
IDRP - Indefinite Repeat 4-53  
IDRP pseudo-operation 4-53

IERA  
IERA - Immediate EXCLUSIVE OR to A 3-55

IF GREATER THAN  
IFG - If Greater Than 4-38

IF LESS THAN  
IFL - If Less Than 4-39

IF NOT EQUAL  
INE - If Not Equal 4-40

IFE  
IFE pseudo-operation 4-38

IFG  
IFG - If Greater Than 4-38  
IFG pseudo-operation 4-38

IFL  
IFL - If Less Than 4-39  
IFL pseudo-operation 4-39

ILA  
ILA - Immediate Load A 3-52

ILLEGAL  
Illegal Memory Operation 3-8  
Illegal Operation Code Fault 3-8  
Illegal Program Interrupt Fault 3-8, 3-10

ILQ  
ILQ - Immediate Load 3-52

IMMEDIATE  
GROUP 1 IMMEDIATE ADD INSTRUCTIONS 3-49  
GROUP 1 IMMEDIATE BOOLEAN INSTRUCTIONS 3-54  
GROUP 1 IMMEDIATE COMPARE INSTRUCTIONS 3-55  
GROUP 1 IMMEDIATE LOAD INSTRUCTIONS 3-52  
IAA - Immediate Add to A 3-49  
IACXn - Immediate Add Character Address to Xn 3-50  
IANA - Immediate AND to A 3-54  
IAQ - Immediate Add to Q 3-50  
ICANA - Immediate Comparative AND with A 3-56  
ICMPA - Immediate Compare A 3-55  
IERA - Immediate EXCLUSIVE OR to A 3-55  
ILA - Immediate Load A 3-52  
ILQ - Immediate Load 3-52  
IORA - Immediate OR to A 3-55

IMPLEMENTATION  
Implementation of System Macro Operations 4-55

IMW  
INTERRUPT MULTIPLEX WORD (IMW) 5-39

IND  
IND - Generate One Word for Indirect Addressing 4-43  
IND pseudo-operation 4-43

IND/A  
IND/A T Character 1-14

IND/ZERO  
IND/ZERO Pseudo-Operation 1-14

## INDEFINITE

IDRP - Indefinite Repeat 4-53

## INDEX

index registers 3-6

## INDICATOR

Carry Indicator 3-13  
indicator register 3-6, 3-12  
Indicator Register (IR) 3-12  
Interrupt Inhibit Indicator 3-14  
LDI - Load Indicator Register 3-31  
Load Indicator Register (LDI) 3-13  
Negative Indicator 3-13  
Overflow Fault Inhibit Indicator 3-13  
Overflow Indicator 3-13  
Parity Error Indicator 3-13  
Parity Fault Inhibit Indicator 3-13  
PROCESSOR INDICATORS 3-12  
STI - Store Indicator Register 3-34  
SZN - Set Zero and Negative Indicators from Storage 3-45  
Zero Indicator 3-12

## INDIRECT

Indirect Control Word 5-25, 5-33  
INDIRECT CONTROL WORD (ICW) 5-66  
INDIRECT CONTROL WORD BASE ADDRESS PATCH 5-64  
indirect vector address 3-10  
LIST INDIRECT CONTROL WORD (LICW) 5-7, 5-17

## INDIRECT ADDRESSING

IND - Generate One Word for Indirect Addressing 4-43

## INDIRECT CHANNEL

INDIRECT CHANNEL PROGRAMMING 5-2

## INDIRECT LEVEL

CHARACTER ADDRESSING - INDIRECT LEVEL 3-22  
Indirect Level Effective Address Formation Rules 3-21  
WORD ADDRESSING - INDIRECT LEVEL 3-21

## INE

INE - If Not Equal 4-40  
INE pseudo-operation 4-40

## INH

INH (Inhibit Interrupts) 3-11  
INH - Interrupt Inhibit 3-65  
Interrupt Inhibit (INH) 3-14

## INHIBIT

INH (Inhibit Interrupts) 3-11  
INH - Interrupt Inhibit 3-65  
Interrupt Inhibit (INH) 3-14  
Interrupt Inhibit Indicator 3-14  
Overflow Fault Inhibit Indicator 3-13  
Parity Fault Inhibit 3-8  
Parity Fault Inhibit Indicator 3-13

## INPUT

NO END CARD ON INPUT FILE 1-16  
SOURCE PROGRAM INPUT 1-3  
types of inputs 1-2

## INPUT/OUTPUT 3-24

CIOC - Connect Input/Output Channel 3-48  
Input/output channel select 3-6  
input/output channel select register 3-6  
INPUT/OUTPUT INSTRUCTIONS 3-48  
INPUT/OUTPUT OPERATIONS 5-1  
input/output program interrupt levels 3-9  
SEL - Select Input/Output Channel 3-53

## INSTRUCTION

ADD INSTRUCTIONS 3-34  
algebraic instruction 3-2  
BOOLEAN INSTRUCTIONS 3-41  
COMPARE INSTRUCTIONS 3-43  
DIVIDE INSTRUCTIONS 3-40  
GROUP 1 IMMEDIATE ADD INSTRUCTIONS 3-49  
GROUP 1 IMMEDIATE BOOLEAN INSTRUCTIONS 3-54  
GROUP 1 IMMEDIATE COMPARE INSTRUCTIONS 3-55  
GROUP 1 IMMEDIATE LOAD INSTRUCTIONS 3-52  
GROUP 1 INTERRUPT CONTROL INSTRUCTIONS 3-53  
Group 1 Nonmemory Instructions 3-5  
GROUP 2 DATA MOVEMENT COPY INSTRUCTIONS 3-63  
GROUP 2 DATA MOVEMENT NORMALIZE INSTRUCTIONS 3-62  
GROUP 2 DATA MOVEMENT SHIFT INSTRUCTIONS 3-57  
GROUP 2 INTERRUPT CONTROL INSTRUCTIONS 3-65  
GROUP 2 MISCELLANEOUS INSTRUCTIONS 3-65  
Group 2 Nonmemory Instructions 3-5  
INPUT/OUTPUT INSTRUCTIONS 3-48  
instruction counter 3-6  
INSTRUCTION DATA CONTROL WORD (IDCW) 5-40  
Instruction Literals 2-15  
Instruction Repertoire by Functional Class 3-23  
INSTRUCTIONS 3-4  
LOAD INSTRUCTIONS 3-30  
logical instruction 3-2  
Machine instruction 2-1  
Macro instruction statement 2-1  
Memory Reference Instruction 1-14, 3-4  
Memory Reference Instructions 3-30  
memory reference machine instructions 2-3  
MULTIPLY INSTRUCTIONS 3-39  
Nonmemory Reference Instruction 1-14  
Nonmemory Reference Instruction Literals 2-16

INSTRUCTION (cont)  
 Nonmemory Reference Instructions 3-4  
 nonmemory reference machine instructions 2-4  
 Processor Instruction Description 3-28  
 PROCESSOR INSTRUCTIONS 3-22  
 STORE INSTRUCTIONS 3-32  
 SUBTRACT INSTRUCTIONS 3-37

Integers 2-13

INTERCOMPUTER  
 INTERCOMPUTER ADAPTER (ICA) 5-4

INTERFACE  
 Central System Interface 5-5, 5-15  
 DATANET FNP Interface 5-4  
 DIRECT INTERFACE ADAPTER (DIA) 5-14

INTERMEDIATE  
 UNEXPECTED EOF ON INTERMEDIATE FILE 1-16

INTERNAL  
 Faults (internal interrupts) 3-6

INTERRUPT  
 DIS - Delay Until Interrupt Signal 3-65  
 EMERGENCY INTERRUPT CELL NUMBER 5-6  
 Enable Interrupt (ENI) 3-14  
 ENI (Enable Interrupts) 3-11  
 ENI - Enable Interrupt 3-65  
 Faults (internal interrupts) 3-6  
 GROUP 1 INTERRUPT CONTROL INSTRUCTIONS 3-53  
 GROUP 2 INTERRUPT CONTROL INSTRUCTIONS 3-65  
 Illegal Program Interrupt Fault 3-8, 3-10  
 INH (Inhibit Interrupts) 3-11  
 INH - Interrupt Inhibit 3-65  
 input/output program interrupt levels 3-9  
 INTERRUPT CELL ASSIGNMENT SWITCHES 5-5  
 Interrupt Cells 5-75  
 Interrupt Control 3-25  
 Interrupt Inhibit (INH) 3-14  
 Interrupt Inhibit Indicator 3-14  
 INTERRUPT LEVEL PATCH 5-64  
 INTERRUPT MULTIPLEX WORD (IMW) 5-39  
 interrupt sequence 3-9  
 interrupt service routine 3-10  
 Interrupt Sublevel 3-11  
 Interrupt Sublevel Word 3-9  
 Interrupt Vector 3-11  
 Interrupt Vector location 3-9  
 Memory Map - Interrupts 3-11  
 PROGRAM INTERRUPT CONTROL 5-2  
 program interrupts 3-6, 3-9  
 RIA - Read Interrupt Address 3-53  
 RIA - Read Interrupt Level Enable Register 3-53  
 SIC - Set Interrupt Cells 3-54

INTERRUPT (cont)  
 SIER (Set Interrupt level Enable Register) 3-11  
 SIER - Set Interrupt Level Enable Register 3-54

IOC  
 IOC Fault Status Locations 5-84

IOM  
 IOM FAULTS 5-3

IOM/CHANNEL  
 IOM/CHANNEL STATUS 5-48

IORA  
 IORA - Immediate OR to A 3-55

IR  
 Indicator Register (IR) 3-12

LABEL  
 LBL - Label 4-10  
 Optional Label 1-7

LABELED  
 Labeled Common 1-9  
 SYMDEF SYMREF and Labeled Common symbols 1-8  
 SYMDEF SYMREF Labeled Common 1-15

LADQ  
 LADQ - Load Q 3-30

LANGUAGE  
 LANGUAGE STRUCTURE 2-5

LBL  
 LBL - Label 4-10  
 LBL pseudo-operation 4-10

LDA  
 LDA - Load A 3-30

LDEX  
 LDEX - Load External Register 3-48  
 LOAD EXTERNAL FORMAT (LDEX) 5-52

LDI  
 LDI - Load Indicator Register 3-31  
 Load Indicator Register (LDI) 3-13

LDQ  
 LDQ - Load Q 3-30

LDXN  
 LDXn - Load Xn 3-31

LEFT  
 ALP - A Left Parity Rotate 3-60  
 ALR - A Left Rotate 3-60  
 ALS - A Left Shift 3-58  
 LLR - Long Left Rotate 3-61  
 LLS - Long Left Shift 3-58  
 QLP - Q Left Parity Rotate 3-61  
 QLR - Q Left Rotate 3-60  
 QLS - Q Left Shift 3-58

**LENGTH**  
 length of Blank Common 1-9  
 Length of Blank Common area 1-8  
 Length of the subprogram text region 1-8

**LEVEL**  
 input/output program interrupt levels 3-9  
 INTERRUPT LEVEL PATCH 5-64  
 RIER - Read Interrupt Level Enable Register 3-53  
 SIER (Set Interrupt level Enable Register) 3-11  
 SIER - Set Interrupt Level Enable Register 3-54

**LICW**  
 LIST INDIRECT CONTROL WORD (LICW) 5-7, 5-17

**LINE**  
 EDITP - Edit Print Lines 4-9  
 HIGH SPEED LINE ADAPTER (HSLA) 5-23  
 LINE PRINTER 5-72  
 LOW SPEED LINE ADAPTER (LSLA) 5-31

**LINKAGE**  
 Program linkage 4-1  
 PROGRAM LINKAGE PSEUDO-OPERATIONS 4-57  
 SAVE - Save--Return Linkage Data 4-59

**LIST**  
 LIST INDIRECT CONTROL WORD (LICW) 5-7, 5-17  
 LIST ON/OFF - Control Output Listing 4-5  
 LIST POINTER WORD (LPW) 5-40  
 LNRSM (list nonreferenced symbols) 4-6  
 LOGICAL CHANNEL DCW LIST 5-42

**LISTING**  
 Assembly Listing 1-12  
 DETAIL ON/OFF - Detail Output Listing 4-5  
 EJECT - Restore Output Listing 4-9  
 format of the full listing 1-13  
 Full Listing Format 1-12  
 LIST ON/OFF - Control Output Listing 4-5  
 Listing Format 1-14  
 Preface Card Listing 1-15

**LIT**  
 LIT - Literal Pool Origin 4-37  
 LIT pseudo-operation 4-37

**LITERAL**  
 Alphanumeric Literals 2-14  
 Decimal Literals 2-13  
 Instruction Literals 2-15  
 LIT - Literal Pool Origin 4-37  
 LITERALS 2-12  
 Nonmemory Reference Instruction Literals 2-16

**LITERAL (cont)**  
 Octal Literals 2-14  
 SACI Literals 2-15  
 Variable Field Literals 2-16

**LLR**  
 LLR - Long Left Rotate 3-61

**LLS**  
 LLS - Long Left Shift 3-58

**LNRSM**  
 LNRSM (list nonreferenced symbols) 4-6

**LOAD**  
 Data Movement - Load 3-23  
 GROUP 1 IMMEDIATE LOAD INSTRUCTIONS 3-52  
 ILA - Immediate Load A 3-52  
 ILQ - Immediate Load 3-52  
 LADQ - Load Q 3-30  
 LDA - Load A 3-30  
 LDEX - Load External Register 3-48  
 LDI - Load Indicator Register 3-31  
 LDQ - Load Q 3-30  
 LDXn - Load Xn 3-31  
 LOAD EXTERNAL FORMAT (LDEX) 5-52  
 Load Indicator Register (LDI) 3-13  
 LOAD INSTRUCTIONS 3-30  
 LODM - Load System Macro Operations 4-55

**LOADER**  
 Relocatable Loader 1-3, 1-10

**LOADING**  
 loading of the subprogram 1-9

**LOC**  
 LOC - Location of Output Text 4-20  
 LOC pseudo-operation 4-20

**LOCATION**  
 BASE - Force Location Counter to a Multiple Power of 2 4-26  
 BEGIN - Origin of a Location Counter 4-19  
 EFFECTIVE ADDRESS AND MEMORY LOCATION SYMBOLS 3-28  
 EIGHT - Force Location Counter to a Multiple of 8 4-25  
 EVEN - Force Location Counter Even 4-25  
 Interrupt Vector location 3-9  
 IOC Fault Status Locations 5-84  
 LOC - Location of Output Text 4-20  
 Location counter 4-1  
 LOCATION COUNTER PSEUDO-OPERATION 4-19  
 Location Field 2-3  
 MARK - Specify Symbol in Location Field 4-40  
 Octal location 1-12  
 ODD - Force Location Counter Odd 4-25  
 USE - Use Multiple Location Counters 4-19

LODM  
 LODM - Load System Macro Operations 4-55  
 LODM pseudo-operation 4-55

LOGIC  
 ARL - A Right Logic 3-59  
 LRL - Long Right Logic 3-59  
 QRL - Q Right Logic 3-59

LOGICAL  
 LOGICAL CHANNEL DCW LIST 5-42  
 logical instruction 3-2

LONG  
 LLR - Long Left Rotate 3-61  
 LLS - Long Left Shift 3-58  
 LRL - Long Right Logic 3-59  
 LRS - Long Right Shift 3-57  
 NRML - Normalize Long 3-63

LOW SPEED  
 LOW SPEED LINE ADAPTER (LSLA) 5-31

LPW  
 LIST POINTER WORD (LPW) 5-40

LRL  
 LRL - Long Right Logic 3-59

LRS  
 LRS - Long Right Shift 3-57

LSLA  
 LOW SPEED LINE ADAPTER (LSLA) 5-31

LSTOU 1-3

MACHINE  
 Machine instruction 2-1  
 memory reference machine instructions 2-3  
 nonmemory reference machine instructions 2-4

MACRO 4-1  
 Definition of the Macro Prototype 4-46  
 DELM - Delete Macro Named 4-54  
 END - End Macro Prototype 4-47  
 ERROR IN MACRO EXPANSION 1-16  
 Implementation of System Macro Operations 4-55  
 LODM - Load System Macro Operations 4-55  
 MACRO - MACRO Identification 4-47  
 MACRO EXPANSION TABLE OVERFLOW 1-16  
 Macro instruction statement 2-1  
 MACRO PROTOTYPE TABLE OVERFLOW 1-16  
 MACRO pseudo-operation 4-47  
 MACRO PSEUDO-OPERATIONS 4-45  
 PMC ON/OFF - Print Macro Expansion 4-8  
 Pseudo-Operations Used Within Macro Prototypes 4-52  
 PUNM - Punch Macro Prototypes 4-55  
 REFMA ON/OFF - Reference Macro Operation 4-6

MACRO (cont)  
 Using a Macro Operation 4-50

MAILBOX  
 MAILBOX ADDRESSES 5-5  
 PCW MAILBOX 5-6  
 PSA MAILBOX 5-40

MAP  
 Control Word Memory Map (Example for Channel 06) 5-26  
 DATANET FNP GENERAL MEMORY MAP 5-75  
 MAP error flags 1-16  
 Memory Map - Interrupts 3-11

MARK  
 MARK - Specify Symbol in Location Field 4-40  
 MARK pseudo-operation 4-40

MASK  
 Mask Register Word 5-26

MAX  
 MAX - Maximum 4-23  
 MAX pseudo-operation 4-23

MAXIMUM  
 MAX - Maximum 4-23  
 MAXSZ - Maximum Size of Assembly 4-42

MAXSZ  
 MAXSZ - Maximum Size of Assembly 4-42  
 MAXSZ pseudo-operation 4-43

MEMORY  
 Control Word Memory Map (Example for Channel 06) 5-26  
 DATANET FNP GENERAL MEMORY MAP 5-75  
 EFFECTIVE ADDRESS AND MEMORY LOCATION SYMBOLS 3-28  
 Illegal Memory Operation 3-8  
 Memory allocation 4-1  
 MEMORY ALLOCATION PSEUDO-OPERATIONS 4-36  
 Memory Map - Interrupts 3-11  
 Memory Parity Error 3-7  
 Memory Parity Error Fault 3-8  
 Memory Reference Instruction 1-14, 3-4  
 Memory Reference Instructions 3-30  
 memory reference machine instructions 2-3

MESSAGE  
 error message 1-16

MIN  
 MIN - Minimum 4-23  
 MIN pseudo-operation 4-23

MINIMUM  
 MIN - Minimum 4-23

MINUS  
 TMI - Transfer on Minus 3-46

MINUS (cont)  
 Transfer on Minus (TMI) 3-13

MNEMONICS  
 Mnemonics by functional class 3-22

MODE  
 CONNECT PCW (OPERATIONAL MODE) 5-38

MONITOR  
 COMPUTER MONITOR ADAPTER (CMA) 5-64

MONITORING  
 System Monitoring 5-69

MOVEMENT  
 Data Movement 3-25, 5-88  
 Data Movement - Load 3-23  
 Data Movement - Store 3-23  
 GROUP 2 DATA MOVEMENT COPY  
 INSTRUCTIONS 3-63  
 GROUP 2 DATA MOVEMENT NORMALIZE  
 INSTRUCTIONS 3-62  
 GROUP 2 DATA MOVEMENT SHIFT  
 INSTRUCTIONS 3-57

MPC  
 MPC COMMANDS 5-46  
 MPC DEVICE STATUS 5-47  
 SERVICE CODES - MPC TO PSA 5-45  
 SERVICE CODES - PSA TO MPC 5-45

MPF  
 MPF - Multiply Fraction 3-39

MRS200/DRD200 5-50

MULTIPLE  
 BASE - Force Location Counter to a  
 Multiple Power of 2 4-26  
 EIGHT - Force Location Counter to a  
 Multiple of 8 4-25  
 USE - Use Multiple Location  
 Counters 4-19

MULTIPLEX  
 INTERRUPT MULTIPLEX WORD (IMW) 5-39

MULTIPLICATION  
 Arithmetic - Multiplication 3-23

MULTIPLY  
 MPF - Multiply Fraction 3-39  
 MULTIPLY INSTRUCTIONS 3-39

NCOMDK 1-4

NCOPY 1-4

NDECK 1-4

NDUMP 1-4

NEGATIVE  
 Negative Indicator 3-13  
 SZN - Set Zero and Negative  
 Indicators from Storage 3-45

NGMAC 1-4

NLSTOU 1-4

NONMEMORY  
 Group 1 Nonmemory Instructions 3-5  
 Group 2 Nonmemory Instructions 3-5  
 Nonmemory Reference Instruction  
 1-14  
 Nonmemory Reference Instruction  
 Literals 2-16  
 Nonmemory Reference Instructions  
 3-4  
 nonmemory reference machine  
 instructions 2-4

NONOP  
 NONOP - Undefined Operation 4-41  
 NONOP pseudo-operation 4-41

NONREFERENCED  
 LNRSN (list nonreferenced symbols)  
 4-6

NOP  
 NOP - No Operation 3-66

NORMALIZE  
 GROUP 2 DATA MOVEMENT NORMALIZE  
 INSTRUCTIONS 3-62  
 NRM - Normalize 3-62  
 NRML - Normalize Long 3-63

NRM  
 NRM - Normalize 3-62

NRML  
 NRML - Normalize Long 3-63

NULL  
 NULL - Null 4-25  
 NULL pseudo-operation 4-25

NUMBER  
 CHANNEL NUMBER PATCH 5-64  
 EMERGENCY INTERRUPT CELL NUMBER 5-6  
 NUMBER SYSTEM 3-2

NXEC 1-4  
 NXEC OPTION SPECIFIED 1-16

OBJECT  
 Absolute Object (Binary) Deck 1-11  
 object program 2-1  
 Relocatable Object Deck 1-8

OCT  
 OCT pseudo-operation 4-27

OCTAL  
 OCTAL - Octal 4-27  
 Octal Literals 2-14  
 Octal location 1-12  
 Octal representation 1-12

ODD  
 ODD - Force Location Counter Odd  
 4-25



ODD (cont)  
   ODD pseudo-operation 4-25

ON/OFF  
   CRSM ON/OFF - Created Symbols 4-53  
   DETAIL ON/OFF - Detail Output Listing 4-5  
   LIST ON/OFF - Control Output Listing 4-5  
   ON/OFF Switch Type Control Pseudo-Operation 4-4  
   PARITY ON/OFF - ASCII Parity Control 4-7  
   PCC ON/OFF - Print Control Cards 4-5  
   PMC ON/OFF - Print Macro Expansion 4-8  
   PUNCH ON/OFF - Control Card Output 4-8  
   REF ON/OFF - References 4-6  
   REFMA ON/OFF - Reference Macro Operation 4-6

ON5 1-4

OPD  
   OPD - Operation Definition 4-16  
   OPD pseudo-operation 4-16

OPERATION  
   ARG - Argument--Generate Zero Operation Code Computer Word 4-41  
   Boolean Operations 3-23  
   Illegal Memory Operation 3-8  
   Illegal Operation Code Fault 3-8  
   Implementation of System Macro Operations 4-55  
   INPUT/OUTPUT OPERATIONS 5-1  
   LODM - Load System Macro Operations 4-55  
   NONOP - Undefined Operation 4-41  
   NOP - No Operation 3-66  
   OPD - Operation Definition 4-16  
   Operation Field 2-3  
   OPERATION TABLE OVERFLOW 1-16  
   OPSYN - Operation Synonym 4-18  
   REFMA ON/OFF - Reference Macro Operation 4-6  
   Using a Macro Operation 4-50

OPERATIONAL  
   CONNECT PCW (OPERATIONAL MODE) 5-38

OPERATORS  
   Terms and Operators 2-7

OPSYN  
   OPSYN - Operation Synonym 4-18  
   OPSYN pseudo-operation 4-18

OPTION  
   NXEC OPTION SPECIFIED 1-16  
   Optional Comment Sequence Option 1-7

OPTIONAL  
   Optional Comment Sequence Option 1-7

OPTIONAL (cont)  
   Optional Label 1-7

OR  
   ERA - EXCLUSIVE OR to A 3-43  
   ERSA - EXCLUSIVE OR to Storage A 3-43  
   IERA - Immediate EXCLUSIVE OR to A 3-55  
   IORA - Immediate OR to A 3-55  
   ORA - OR to A 3-42  
   ORSA - OR to Storage A 3-42  
   SSCW OR SCW (SSCW SCW) 5-39

ORA  
   ORA - OR to A 3-42

ORG  
   ORG - Origin Set by Programmer 4-20  
   ORG pseudo-operation 4-20

ORGCSM  
   ORGCSM - Origin Created Symbols 4-53  
   ORGCSM pseudo-operation 4-53

ORIGIN  
   BEGIN - Origin of a Location Counter 4-19  
   LIT - Literal Pool Origin 4-37  
   ORG - Origin Set by Programmer 4-20  
   ORGCSM - Origin Created Symbols 4-53

ORSA  
   ORSA - OR to Storage A 3-42

OUTPUT  
   ABS - Output Absolute Text 4-12  
   ASSEMBLER OUTPUTS 1-6  
   DETAIL ON/OFF - Detail Output Listing 4-5  
   EJECT - Restore Output Listing 4-9  
   FUL - Output Full Binary Text 4-12  
   LIST ON/OFF - Control Output Listing 4-5  
   LOC - Location of Output Text 4-20  
   PUNCH ON/OFF - Control Card Output 4-8

OVERFLOW  
   MACRO EXPANSION TABLE OVERFLOW 1-16  
   MACRO PROTOTYPE TABLE OVERFLOW 1-16  
   OPERATION TABLE OVERFLOW 1-16  
   Overflow Fault 3-8  
   Overflow Fault Inhibit Indicator 3-13  
   Overflow faults 3-7  
   Overflow Indicator 3-13  
   SYMBOL REFERENCE TABLE OVERFLOW 1-16  
   SYMBOL TABLE OVERFLOW 1-16  
   TOV - Transfer on Overflow 3-47  
   Transfer on Overflow (TOV) 3-13

PARITY  
   ALP - A Left Parity Rotate 3-60  
   Memory Parity Error 3-7

PARITY (cont)  
 Memory Parity Error Fault 3-8  
 Parity Error Indicator 3-13  
 Parity Fault Inhibit 3-8  
 Parity Fault Inhibit Indicator 3-13  
 PARITY ON/OFF - ASCII Parity  
 Control 4-7  
 QLP - Q Left Parity Rotate 3-61  
 PARITY ON pseudo-operation 4-7

PATCH  
 "DEAD-MAN" TIMER DURATION PATCH  
 5-64  
 CHANNEL NUMBER PATCH 5-64  
 DATA RESPONSE TIMER DURATION PATCH  
 5-64  
 INDIRECT CONTROL WORD BASE ADDRESS  
 PATCH 5-64  
 INTERRUPT LEVEL PATCH 5-64

PATCHING  
 Configuration Patching 5-64

PCC  
 PCC ON/OFF - Print Control Cards  
 4-5  
 PCC pseudo-operation 4-5

PCW  
 CONNECT PCW (OPERATIONAL MODE) 5-38  
 PCW Format 5-33  
 PCW MAILBOX 5-6  
 PERIPHERAL CONTROL WORD (PCW) 5-6,  
 5-12, 5-16, 5-22, 5-50, 5-53, 5-65

PCW0  
 Command PCW0 PCW1 5-25  
 Commands PCW0 PCW1 5-33

PCW1  
 Command PCW0 PCW1 5-25  
 Commands PCW0 PCW1 5-33

PCW2  
 Command PCW2 PCW3 5-25

PCW3  
 Command PCW2 PCW3 5-25

PERIPHERAL  
 COMMON PERIPHERAL STATUS FORMAT  
 5-74  
 PERIPHERAL CONTROL WORD 5-1  
 PERIPHERAL CONTROL WORD (PCW) 5-6,  
 5-12, 5-16, 5-22, 5-50, 5-53, 5-65  
 PERIPHERAL SUBSYSTEM ADAPTER (PSA)  
 5-38

PLUS  
 TPL - Transfer on Plus 3-47  
 Transfer on Plus (TPL) 3-13

PMC  
 PMC ON/OFF - Print Macro Expansion  
 4-8  
 PMC pseudo-operation 4-8

POINTER  
 LIST POINTER WORD (LPW) 5-40

POOL  
 LIT - Literal Pool Origin 4-37

PORT  
 PORT ASSIGNMENT FOR FNPs 5-5  
 System Controller Port  
 Configurations 5-9

POSITIONS  
 REGISTER POSITIONS AND CONTENT  
 SYMBOLS 3-29

POWER  
 BASE - Force Location Counter to a  
 Multiple Power of 2 4-26  
 Power Shutdown Beginning Fault 3-7

POWER-ON  
 Power-On Restart Fault 3-7

PREFACE  
 Preface Card 1-8  
 Preface Card Listing 1-15

PRIMARY  
 primary SYMDEF 1-9

PRINT  
 EDITP - Edit Print Lines 4-9  
 PCC ON/OFF - Print Control Cards  
 4-5  
 PMC ON/OFF - Print Macro Expansion  
 4-8

PRINTER  
 LINE PRINTER 5-72

PROCESSOR  
 OTHER PROCESSOR SYMBOLS 3-29  
 PROCESSOR FAULT SWITCHES 5-5  
 Processor Fault Vectors 5-83  
 Processor Faults 3-7  
 PROCESSOR INDICATORS 3-12  
 Processor Instruction Description  
 3-28  
 PROCESSOR INSTRUCTIONS 3-22  
 PROCESSOR REGISTERS 3-6  
 state of the processor and the  
 program 3-12

PROGRAM  
 Illegal Program Interrupt Fault 3-8,  
 3-10  
 input/output program interrupt  
 levels 3-9  
 object program 2-1  
 PROGRAM INTERRUPT CONTROL 5-2  
 program interrupts 3-6, 3-9  
 Program linkage 4-1  
 PROGRAM LINKAGE PSEUDO-OPERATIONS  
 4-57  
 Program statements 2-1  
 Program Symbols 2-5  
 source program 2-1  
 SOURCE PROGRAM INPUT 1-3

PROGRAM (cont)  
state of the processor and the  
program 3-12  
Types of Program Symbols 2-6

PROGRAMMER  
ORG - Origin Set by Programmer 4-20

PROGRAMMING  
DIRECT CHANNEL PROGRAMMING 5-2  
INDIRECT CHANNEL PROGRAMMING 5-2

PROTOTYPE  
Defining a Prototype 4-56  
Definition of the Macro Prototype  
4-46  
END - End Macro Prototype 4-47  
MACRO PROTOTYPE TABLE OVERFLOW 1-16  
Pseudo-Operations Used Within Macro  
Prototypes 4-52  
PUNM - Punch Macro Prototypes 4-55

PSA  
PERIPHERAL SUBSYSTEM ADAPTER (PSA)  
5-38  
PSA ERROR SUMMARY 5-44  
PSA MAILBOX 5-40  
PSA Word Formats 5-38  
SERVICE CODES - MPC TO PSA 5-45  
SERVICE CODES - PSA TO MPC 5-45

PSEUDO-OPERATION 2-1  
ABS pseudo-operation 4-12  
ACI and ACIC pseudo-operations 4-32  
ARG pseudo-operation 4-41  
ASCII and ASCIIIC pseudo-operations  
4-31  
BASE pseudo-operation 4-26  
BCI pseudo-operation 4-29  
BEGIN pseudo-operation 4-19  
BFS pseudo-operation 4-36  
BLOCK pseudo-operation 4-36  
BOOL pseudo-operation 2-8, 4-22  
BSS pseudo-operation 4-36  
CALL pseudo-operation 4-57  
CONDITIONAL PSEUDO-OPERATIONS 4-37  
CONTROL PSEUDO-OPERATIONS 4-4, 4-9  
CPR pseudo-operation 4-11  
CRSM pseudo-operation 4-53  
DATA CONTROL WORD FORMAT  
PSEUDO-OPERATIONS 4-44  
Data Generating Pseudo-Operation  
1-14  
DATA GENERATING PSEUDO-OPERATIONS  
4-26  
DATE pseudo-operation 4-41  
DCARD pseudo-operation 4-15  
DCW pseudo-operation 4-45  
DEC pseudo-operation 4-27  
DELM pseudo-operation 4-54  
DETAIL pseudo-operation 4-5  
DUP pseudo-operation 4-35  
EDITP pseudo-operation 4-9  
EIGHT pseudo-operation 4-25  
EJECT pseudo-operation 4-9  
END pseudo-operation 1-12, 4-15  
ENDM pseudo-operation 4-47  
EQU pseudo-operation 4-21

PSEUDO-OPERATION (cont)  
ETC pseudo-operation 4-62  
EVEN pseudo-operation 4-25  
FEQU pseudo-operation 4-21  
FUL pseudo-operation 4-12  
HEAD pseudo-operation 4-13  
ICW pseudo-operation 4-44  
IDRP pseudo-operation 4-53  
IFE pseudo-operation 4-38  
IFG pseudo-operation 4-38  
IFL pseudo-operation 4-39  
IND pseudo-operation 4-43  
IND/ZERO Pseudo-Operation 1-14  
INE pseudo-operation 4-40  
LBL pseudo-operation 4-10  
LIT pseudo-operation 4-37  
LOC pseudo-operation 4-20  
LOCATION COUNTER PSEUDO-OPERATION  
4-19  
LODM pseudo-operation 4-55  
MACRO pseudo-operation 4-47  
MACRO PSEUDO-OPERATIONS 4-45  
MARK pseudo-operation 4-40  
MAX pseudo-operation 4-23  
MAXSZ pseudo-operation 4-43  
MEMORY ALLOCATION PSEUDO-OPERATIONS  
4-36  
MIN pseudo-operation 4-23  
NONOP pseudo-operation 4-41  
NULL pseudo-operation 4-25  
OCT pseudo-operation 4-27  
ODD pseudo-operation 4-25  
ON/OFF Switch Type Control  
Pseudo-Operation 4-4  
OPD pseudo-operation 4-16  
OPSYN pseudo-operation 4-18  
ORG pseudo-operation 4-20  
ORGCSM pseudo-operation 4-53  
PCC pseudo-operation 4-5  
PMC pseudo-operation 4-8  
PROGRAM LINKAGE PSEUDO-OPERATIONS  
4-57  
Pseudo-Operations Used Within Macro  
Prototypes 4-52  
PUNCH pseudo-operation 4-8  
REF pseudo-operation 4-6  
REM pseudo-operation 4-9  
RETURN pseudo-operation 4-61  
SACI pseudo-operation 4-32  
SAVE pseudo-operation 4-59  
SET pseudo-operation 4-22  
SPECIAL WORD FORMAT  
PSEUDO-OPERATIONS 4-40  
SYMBOL-DEFINING PSEUDO-OPERATIONS  
4-21  
SYMDEF pseudo-operation 4-23  
SYMREF pseudo-operation 4-24  
TCD pseudo-operation 4-13  
TTL pseudo-operation 4-11  
TTLDAT pseudo-operation 4-41  
TTLS pseudo-operation 4-11  
USE pseudo-operations 4-19  
VFD pseudo-operation 2-8, 4-29  
ZERO pseudo-operation 4-42

PUNCH  
DCARD - Punch BCD Card 4-15

PUNCH (cont)  
 PUNCH ON/OFF - Control Card Output 4-8  
 PUNCH pseudo-operation 4-8  
 PUNM - Punch Macro Prototypes 4-55  
 TCD - Punch Transfer Card 4-13

PUNM  
 PUNM - Punch Macro Prototypes 4-55

QLP  
 QLP - Q Left Parity Rotate 3-61

QLR  
 QLR - Q Left Rotate 3-60

QLS  
 QLS - Q Left Shift 3-58

QRL  
 QRL - Q Right Logic 3-59

QRS  
 QRS - Q Right Shift 3-57

QSW  
 QUEUE STATUS WORD (QSW) 5-56

QUEUE  
 QUEUE STATUS WORD (QSW) 5-56

READ  
 RIA - Read Interrupt Address 3-53  
 RIER - Read Interrupt Level Enable Register 3-53

READER  
 CARD READER 5-71

REDEFINITION  
 SET - Symbol Redefinition 4-22

REF  
 REF ON/OFF - References 4-6  
 REF pseudo-operation 4-6

REFERENCE  
 Memory Reference Instruction 1-14, 3-4  
 Memory Reference Instructions 3-30  
 memory reference machine instructions 2-3  
 Nonmemory Reference Instruction 1-14  
 Nonmemory Reference Instruction Literals 2-16  
 Nonmemory Reference Instructions 3-4  
 nonmemory reference machine instructions 2-4  
 REF ON/OFF - References 4-6  
 REFMA ON/OFF - Reference Macro Operation 4-6  
 SYMBOL REFERENCE TABLE OVERFLOW 1-16  
 Symbolic Reference Table 1-15  
 SYMREF - Symbol Reference 4-24

REFMA  
 REFMA ON/OFF - Reference Macro Operation 4-6  
 REFMA ON PSEUDO-OPERATION 4-6

REGION  
 Length of the subprogram text region 1-8

REGISTER  
 AQ register 3-6  
 I/O Channel Select Register 3-12  
 index registers 3-6  
 indicator register 3-6, 3-12  
 Indicator Register (IR) 3-12  
 input/output channel select register 3-6  
 LDEX - Load External Register 3-48  
 LDI - Load Indicator Register 3-31  
 Load Indicator Register (LDI) 3-13  
 Mask Register Word 5-26  
 PROCESSOR REGISTERS 3-6  
 REGISTER POSITIONS AND CONTENT SYMBOLS 3-29  
 REGISTER SYMBOLS 3-29  
 RIER - Read Interrupt Level Enable Register 3-53  
 SIER (Set Interrupt level Enable Register) 3-11  
 SIER - Set Interrupt Level Enable Register 3-54  
 STEX - Store External Register 3-49  
 STI - Store Indicator Register 3-34

RELOCATABLE 2-6  
 column binary relocatable text card 1-10  
 RELOCATABLE AND ABSOLUTE ASSEMBLIES 1-2  
 Relocatable and Absolute Expression 2-10  
 Relocatable Loader 1-3, 1-10  
 Relocatable Object Deck 1-8  
 Relocatable Text Card 1-10  
 Special Relocatable Expressions 2-11

RELOCATION  
 admissibility of relocation 2-10  
 Relocation bits 1-12  
 Relocation data 1-10

REM  
 REM - Remarks 4-9  
 REM pseudo-operation 4-9

REMARKS  
 REM - Remarks 4-9

REPEAT  
 IDRP - Indefinite Repeat 4-53

REPertoire  
 Instruction Repertoire by Functional Class 3-23

REPRESENTATION  
 Octal representation 1-12

REPRESENTATION (cont)  
 REPRESENTATION OF INFORMATION 3-1

REPRODUCTION  
 Reproduction of the symbolic card  
 1-13

RESPONSE  
 DATA RESPONSE TIMER DURATION PATCH  
 5-64

RESTART  
 Power-On Restart Fault 3-7

RESTORE  
 EJECT - Restore Output Listing 4-9

RETURN  
 RETURN - Return--From Subroutines  
 4-61  
 RETURN pseudo-operation 4-61

RIA  
 RIA - Read Interrupt Address 3-53

RIER  
 RIER - Read Interrupt Level Enable  
 Register 3-53

RIGHT  
 ARL - A Right Logic 3-59  
 ARS - A Right Shift 3-57  
 LRL - Long Right Logic 3-59  
 LRS - Long Right Shift 3-57  
 QRL - Q Right Logic 3-59  
 QRS - Q Right Shift 3-57

ROTATE  
 ALP - A Left Parity Rotate 3-60  
 ALR - A Left Rotate 3-60  
 LLR - Long Left Rotate 3-61  
 QLP - Q Left Parity Rotate 3-61  
 QLR - Q Left Rotate 3-60

ROUTINE  
 Binary to Binary Coded Decimal  
 Conversion Routine 5-89  
 interrupt service routine 3-10

RULES  
 Basic Level Effective Address  
 Formation Rules 3-15  
 Character Address Addition Rules  
 3-20  
 Indirect Level Effective Address  
 Formation Rules 3-21

SACI  
 SACI - Symbolic ASCII Information  
 4-32  
 SACI Literals 2-15  
 SACI pseudo-operation 4-32

SAVE  
 SAVE - Save--Return Linkage Data  
 4-59  
 SAVE pseudo-operation 4-59

SBA  
 SBA - Subtract from A 3-37

SBAQ  
 SBAQ - Subtract from AQ 3-38

SBQ  
 SBQ - Subtract from Q 3-38

SCW  
 SSCW OR SCW (SSCW SCW) 5-39

SECONDARY  
 secondary SYMDEF 1-9

SEL  
 SEL - Select Input/Output Channel  
 3-53

SELECT  
 I/O Channel Select Register 3-12  
 Input/output channel select 3-6  
 input/output channel select  
 register 3-6  
 SEL - Select Input/Output Channel  
 3-53

SEQUENCE  
 interrupt sequence 3-9  
 Optional Comment Sequence Option  
 1-7

SERVICE  
 interrupt service routine 3-10  
 SERVICE CODES - MPC TO PSA 5-45  
 SERVICE CODES - PSA TO MPC 5-45

SET  
 Character Set 2-5  
 ORG - Origin Set by Programmer 4-20  
 SET - Symbol Redefinition 4-22  
 SET pseudo-operation 4-22  
 SIC - Set Interrupt Cells 3-54  
 SIER (Set Interrupt level Enable  
 Register) 3-11  
 SIER - Set Interrupt Level Enable  
 Register 3-54  
 SZN - Set Zero and Negative  
 Indicators from Storage 3-45

SHIFT  
 ALS - A Left Shift 3-58  
 ARS - A Right Shift 3-57  
 GROUP 2 DATA MOVEMENT SHIFT  
 INSTRUCTIONS 3-57  
 LLS - Long Left Shift 3-58  
 LRS - Long Right Shift 3-57  
 QLS - Q Left Shift 3-58  
 QRS - Q Right Shift 3-57  
 Shifts 3-25

SHUTDOWN  
 Power Shutdown Beginning Fault 3-7

SIC  
 SIC - Set Interrupt Cells 3-54

SIER  
 SIER (Set Interrupt level Enable Register) 3-11  
 SIER - Set Interrupt Level Enable Register 3-54

SIGNAL  
 DIS - Delay Until Interrupt Signal 3-65

SINGLE-PRECISION  
 Single-Precision Data 3-1  
 Single-Precision Floating-Point 2-13

SIZE  
 MAXSZ - Maximum Size of Assembly 4-42

SKIPPED  
 NOT ENOUGH CARDS TO BE SKIPPED 1-16

SOURCE  
 source program 2-1  
 SOURCE PROGRAM INPUT 1-3

SSA  
 SSA - Subtract Stored from A 3-38

SSCW  
 SSCW OR SCW (SSCW SCW) 5-39

STA  
 STA - Store A 3-32

STAQ  
 STAQ - Store AQ 3-32

STATE  
 state of the processor and the program 3-12

STATEMENT  
 Macro instruction statement 2-1  
 Program statements 2-1

STATUS 5-2, 5-29, 5-34  
 ACTIVE STATUS FORMAT 5-10  
 Actual Status Word Format 5-13  
 BINARY SYNCHRONIZATION STATUS 5-31  
 COMMON PERIPHERAL STATUS FORMAT 5-74  
 CONFIGURATION STATUS FORMAT 5-9, 5-19  
 IOC Fault Status Locations 5-84  
 IOM/CHANNEL STATUS 5-48  
 MPC DEVICE STATUS 5-47  
 QUEUE STATUS WORD (QSW) 5-56  
 Status Characters 5-36  
 STATUS WORD FORMAT 5-11, 5-51  
 STATUS WORD FORMATS 5-42  
 Status Words 5-67  
 TERMINATE STATUS WORD 5-57

STEX  
 STEX - Store External Register 3-49  
 STORE EXTERNAL FORMAT (STEX) 5-53

STI  
 STI - Store Indicator Register 3-34

STORAGE  
 ANSA - AND to Storage A 3-42  
 AOS - Add One to Storage 3-37  
 ASA - Add A to Storage 3-37  
 ERSA - EXCLUSIVE OR to Storage A 3-43  
 ORSA - OR to Storage A 3-42  
 SZN - Set Zero and Negative Indicators from Storage 3-45

STORE  
 Data Movement - Store 3-23  
 SSA - Subtract Stored from A 3-38  
 STA - Store A 3-32  
 STAQ - Store AQ 3-32  
 STEX - Store External Register 3-49  
 STI - Store Indicator Register 3-34  
 STORE EXTERNAL FORMAT (STEX) 5-53  
 STORE INSTRUCTIONS 3-32  
 STQ - Store Q 3-32  
 STXn - Store Xn 3-33  
 STZ - Store Zero 3-33  
 TSY - Transfer and Store IC in Y 3-46

STQ  
 STQ - Store Q 3-32

STRUCTURE  
 LANGUAGE STRUCTURE 2-5

STXN  
 STXn - Store Xn 3-33

STZ  
 STZ - Store Zero 3-33

SUBFIELDS  
 ZERO - Generate One Word with Two Subfields 4-42

SUBLEVEL  
 Interrupt Sublevel 3-11  
 Interrupt Sublevel Word 3-9

SUBPROGRAM  
 Length of the subprogram text region 1-8  
 loading of the subprogram 1-9  
 Subprogram delimiter 1-2

SUBROUTINES  
 CALL - Call Subroutines 4-57  
 RETURN - Return--From Subroutines 4-61

SUBSYSTEM  
 PERIPHERAL SUBSYSTEM ADAPTER (PSA) 5-38

SUBTITLE  
 TTLS - Subtitle 4-11

SUBTRACT  
 SBA - Subtract from A 3-37

SUBTRACT (cont)  
SBAQ - Subtract from AQ 3-38  
SBQ - Subtract from Q 3-38  
SSA - Subtract Stored from A 3-38  
SUBTRACT INSTRUCTIONS 3-37

SUBTRACTION 3-2  
Arithmetic - Subtraction 3-23  
BCD Subtraction 5-86

SUMMARY  
PSA ERROR SUMMARY 5-44  
Summary of Symbolic Card Format 2-5

SWITCH  
ON/OFF Switch Type Control  
Pseudo-Operation 4-4  
PROCESSOR FAULT SWITCHES 5-5  
TIMER AND SWITCH CHANNEL 5-73

SYMBOL  
BFS - Block Followed by Symbol 4-36  
BSS - Block Started by Symbol 4-36  
CRSM ON/OFF - Created Symbols 4-53  
EFFECTIVE ADDRESS AND MEMORY  
LOCATION SYMBOLS 3-28  
LNRSM (list nonreferenced symbols)  
4-6  
MARK - Specify Symbol in Location  
Field 4-40  
ORGCSM - Origin Created Symbols  
4-53  
OTHER PROCESSOR SYMBOLS 3-29  
Program Symbols 2-5  
REGISTER POSITIONS AND CONTENT  
SYMBOLS 3-29  
REGISTER SYMBOLS 3-29  
SET - Symbol Redefinition 4-22  
Symbol definition 4-1  
SYMBOL REFERENCE TABLE OVERFLOW  
1-16  
SYMBOL TABLE OVERFLOW 1-16  
SYMBOL-DEFINING PSEUDO-OPERATIONS  
4-21  
SYMDEF - Symbol Definition 4-23  
SYMDEF SYMREF and Labeled Common  
symbols 1-8  
SYMREF - Symbol Reference 4-24  
SYSTEM (BUILT-IN) SYMBOLS 4-62  
Types of Program Symbols 2-6

SYMBOLIC  
Reproduction of the symbolic card  
1-13  
SACI - Symbolic ASCII Information  
4-32  
Summary of Symbolic Card Format 2-5  
SYMBOLIC CODING FORM 2-1, 2-2  
Symbolic Reference Table 1-15

SYMDEF  
EXECUTION NOT POSSIBLE NO SYMDEF  
1-16  
primary SYMDEF 1-9  
secondary SYMDEF 1-9  
SYMDEF - Symbol Definition 4-23  
SYMDEF pseudo-operation 4-23

SYMDEF (cont)  
SYMDEF SYMREF and Labeled Common  
symbols 1-8  
SYMDEF SYMREF Labeled Common 1-15

SYMREF 1-9, 2-6  
SYMDEF SYMREF and Labeled Common  
symbols 1-8  
SYMDEF SYMREF Labeled Common 1-15  
SYMREF - Symbol Reference 4-24  
SYMREF pseudo-operation 4-24

SYMTAB 1-4

SYNCHRONIZATION  
BINARY SYNCHRONIZATION STATUS 5-31

SYNCHRONOUS  
BINARY SYNCHRONOUS CHANNEL (BSC)  
5-58

SYNONYM  
OPSYN - Operation Synonym 4-18

SYSTEM  
Central System Control Word Formats  
5-12, 5-22  
Central System Interface 5-5, 5-15  
Implementation of System Macro  
Operations 4-55  
LODM - Load System Macro Operations  
4-55  
NUMBER SYSTEM 3-2  
SYSTEM (BUILT-IN) SYMBOLS 4-62  
System Controller Port  
Configurations 5-9  
System Monitoring 5-69

SZN  
SZN - Set Zero and Negative  
Indicators from Storage 3-45

TABLE  
Character Control Table 5-27  
MACRO EXPANSION TABLE OVERFLOW 1-16  
MACRO PROTOTYPE TABLE OVERFLOW 1-16  
OPERATION TABLE OVERFLOW 1-16  
SYMBOL REFERENCE TABLE OVERFLOW  
1-16  
SYMBOL TABLE OVERFLOW 1-16  
Symbolic Reference Table 1-15

TCD  
TCD - Punch Transfer Card 4-13  
TCD pseudo-operation 4-13

TERMINATE  
TERMINATE STATUS WORD 5-57

TERMS  
Terms and Operators 2-7

TEXT  
ABS - Output Absolute Text 4-12  
Absolute Text Card 1-11  
absolute text cards 1-11  
column binary relocatable text card  
1-10

TEXT (cont)  
 FUL - Output Full Binary Text 4-12  
 Length of the subprogram text region 1-8  
 LOC - Location of Output Text 4-20  
 Relocatable Text Card 1-10

TIME  
 Time of Assembly 1-7

TIMER  
 TIMER AND SWITCH CHANNEL 5-73

TIMER DURATION  
 "DEAD-MAN" TIMER DURATION PATCH 5-64  
 DATA RESPONSE TIMER DURATION PATCH 5-64

TITLE  
 TTL - Title 4-11  
 TTLDAT - Title Date 4-41

TMI  
 TMI - Transfer on Minus 3-46  
 Transfer on Minus (TMI) 3-13

TNC  
 TNC - Transfer on No Carry 3-47  
 Transfer on No Carry (TNC) 3-13

TNZ  
 TNZ - Transfer on Not Zero 3-46  
 Transfer on Not Zero (TNZ) 3-12

TOV  
 TOV - Transfer on Overflow 3-47  
 Transfer on Overflow (TOV) 3-13

TPL  
 TPL - Transfer on Plus 3-47  
 Transfer on Plus (TPL) 3-13

TRA  
 TRA - Transfer Unconditionally 3-45

TRANSFER  
 Data Transfer 5-67  
 TCD - Punch Transfer Card 4-13  
 TMI - Transfer on Minus 3-46  
 TNC - Transfer on No Carry 3-47  
 TNZ - Transfer on Not Zero 3-46  
 TOV - Transfer on Overflow 3-47  
 TPL - Transfer on Plus 3-47  
 TRA - Transfer Unconditionally 3-45  
 transfer card 1-11, 1-12  
 Transfer of Control 3-24  
 Transfer on Minus (TMI) 3-13  
 Transfer on No Carry (TNC) 3-13  
 Transfer on Not Zero (TNZ) 3-12  
 Transfer on Overflow (TOV) 3-13  
 Transfer on Plus (TPL) 3-13  
 Transfer on Zero (TZE) 3-12  
 TSY - Transfer and Store IC in Y 3-46  
 TZE - Transfer on Zero 3-46

TRANSLITERATION  
 Character Transliteration 5-90

TSY  
 forced TSY fault 3-7  
 TSY - Transfer and Store IC in Y 3-46

TTL  
 TTL - Title 4-11  
 TTL pseudo-operation 4-11

TTLDAT  
 TTLDAT - Title Date 4-41  
 TTLDAT pseudo-operation 4-41

TTLS  
 TTLS - Subtitle 4-11  
 TTLS pseudo-operation 4-11

TYPE  
 ON/OFF Switch Type Control Pseudo-Operation 4-4  
 types of inputs 1-2  
 Types of Program Symbols 2-6

TYPICAL  
 TYPICAL DECK SETUPS 1-6

TZE  
 Transfer on Zero (TZE) 3-12  
 TZE - Transfer on Zero 3-46

USE  
 USE - Use Multiple Location Counters 4-19  
 USE pseudo-operations 4-19

VARIABLE  
 Variable Field 2-3  
 Variable Field Literals 2-16  
 VFD - Variable Field Definition 4-29

VECTOR  
 indirect vector address 3-10  
 Interrupt Vector 3-11  
 Interrupt Vector location 3-9  
 Processor Fault Vectors 5-83

VFD  
 VFD - Variable Field Definition 4-29  
 VFD pseudo-operation 2-8, 4-29

WORD  
 Actual Status Word Format 5-13  
 ARG - Argument--Generate Zero Operation Code Computer Word 4-41  
 Base Address Word 5-26, 5-38  
 CMA Control Words 5-65  
 Control Words 5-25, 5-32  
 IND - Generate One Word for Indirect Addressing 4-43  
 INTERRUPT MULTIPLEX WORD (IMW) 5-39  
 Interrupt Sublevel Word 3-9  
 LIST POINTER WORD (LPW) 5-40  
 Mask Register Word 5-26



WORD (cont)

PSA Word Formats 5-38  
QUEUE STATUS WORD (QSW) 5-56  
SPECIAL WORD FORMAT  
PSEUDO-OPERATIONS 4-40  
STATUS WORD FORMAT 5-11, 5-51  
STATUS WORD FORMATS 5-42  
Status Words 5-67  
TERMINATE STATUS WORD 5-57  
word address 3-14  
WORD ADDRESSING - BASIC LEVEL 3-16  
WORD ADDRESSING - INDIRECT LEVEL  
3-21  
ZERO - Generate One Word with Two  
Subfields 4-42

XN

ADCXn - Add Character Address to Xn  
3-35  
CAXn - Copy A into Xn 3-64  
CMPXn - Compare with Xn 3-45  
CXnA - Copy Xn into A 3-64  
IACXn - Immediate Add Character  
Address to Xn 3-50  
LDXn - Load Xn 3-31  
STXn - Store Xn 3-33

Y\*\*

effective address (Y\*\*) 3-15

ZERO

ARG - Argument--Generate Zero  
Operation Code Computer Word 4-41  
STZ - Store Zero 3-33  
SZN - Set Zero and Negative  
Indicators from Storage 3-45  
TNZ - Transfer on Not Zero 3-46  
Transfer on Not Zero (TNZ) 3-12  
Transfer on Zero (TZE) 3-12  
TZE - Transfer on Zero 3-46  
ZERO - Generate One Word with Two  
Subfields 4-42  
Zero Indicator 3-12  
ZERO pseudo-operation 4-42





# HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE

SERIES 60 (LEVEL 66)/6000 DATANET 355/6600  
MACRO ASSEMBLER PROGRAM

ORDER NO.

DD01, REV. 0

DATED

DECEMBER 1975

## ERRORS IN PUBLICATION

## SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION



Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below.

FROM: NAME \_\_\_\_\_

DATE \_\_\_\_\_

TITLE \_\_\_\_\_

COMPANY \_\_\_\_\_

ADDRESS \_\_\_\_\_

\_\_\_\_\_

CUT ALONG LI

CUT ALONG LINE

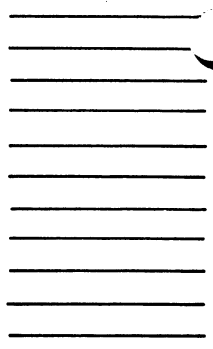
FOLD ALONG LINE

FIRST CLASS  
PERMIT NO. 39531  
WALTHAM, MA  
02154

Business Reply Mail  
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS  
200 SMITH STREET  
WALTHAM, MA 02154



CUT ALONG LINE

FOLD ALONG LINE

ATTENTION: PUBLICATIONS, MS 486

**Honeywell**

CUT ALONG LINE

# Honeywell

## Honeywell Information Systems

In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154  
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5  
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

15332, 2376, Printed in U.S.A.

DD01, Rev. 0