

**Honeywell**

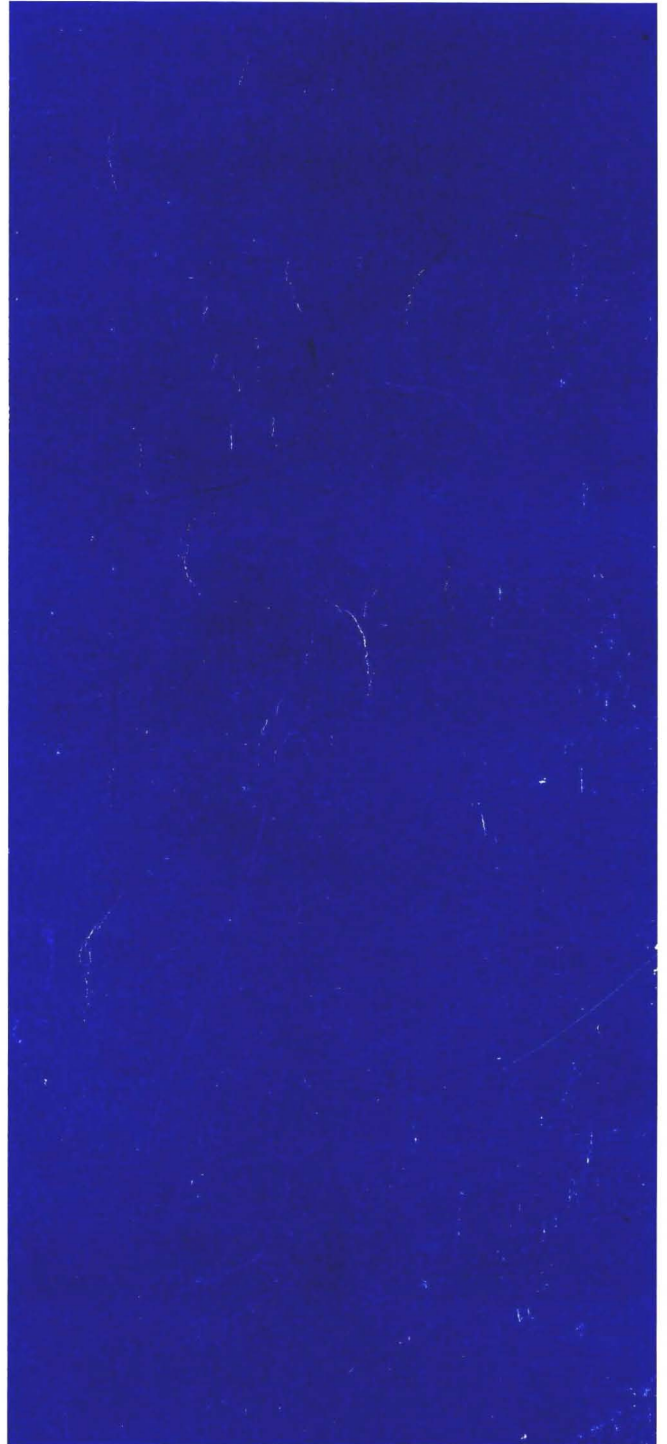
MODELS 2040  
THROUGH 2070  
PROGRAMMERS'  
REFERENCE MANUAL

SERIES 2000

---

HARDWARE

---



# Honeywell

## MODELS 2040 THROUGH 2070 PROGRAMMERS' REFERENCE MANUAL

SERIES 2000

**SUBJECT:**

The Central Processor Hardware of Series 2000 Models 2040, 2040A, 2050, 2050A, 2060 and 2070; the Easycoder Assembly Language; Summary Information Concerning the Programming of Series 2000 Peripherals.

**SPECIAL INSTRUCTIONS:**

Peripheral devices used in Series 2000 systems are thoroughly documented in their own manuals. A complete listing of manuals is contained in the Honeywell Publications Price Catalog, Order No. AB81.

**DATE:**

May 1973

**ORDER NUMBER:**

AG28, Rev. 0

## PREFACE

This manual constitutes a programmer's reference source of detailed information concerning the central processor hardware of Series 2000 models. The EasyCoder Assembly Language, used with the Series 2000 Basic Programming System and the Series 2000 operating system – OS/2000, is also defined. In addition, this volume contains information concerning the programming of Series 2000 peripheral devices and the Scientific Unit and Scientific Sub-processor. The hardware information presented herein is equally applicable for the programmer using the following operating systems: Extended Mod 1 (MSR), Mod 1, or Mod 4. It is recommended that the user obtain software publications applicable to his operating system; refer to the Honeywell Publications Price Catalog, Order No. AB81, for a complete list of available publications.

The equipment characteristics reported herein remain subject to change to allow the introduction of design improvements.

## CONTENTS

	Page
Section I	
Series 2000 Components .....	1-1
Central Processor .....	1-1
Consoles .....	1-2
Standard Processing Mode .....	1-2
Interrupt Processing Mode .....	1-3
External Interrupts .....	1-3
Internal Interrupt .....	1-3
Addressing Modes .....	1-4
Item-Mark Trapping Mode .....	1-4
Processing Power .....	1-4
Peripheral Interface .....	1-5
Peripheral Control .....	1-6
Peripheral Data Transfer Operation .....	1-6
Peripheral Addresses and Unit Loads .....	1-7
Read/Write Channel .....	1-8
Peripheral Equipment .....	1-9
Punched Card Equipment .....	1-9
High-Speed Printers .....	1-9
Print Buffer .....	1-10
Magnetic Tape Units .....	1-10
1200-BPI Recording Density .....	1-10
1600-BPI Recording Density .....	1-10
Dynamic Tape Addressing .....	1-11
IBM Magnetic Tape Compatibility .....	1-12
EBCDIC Code Translation .....	1-12
Disk Pack Drives .....	1-12
Write Protect Capability .....	1-14
Dynamic Disk Addressing .....	1-14
Central Processor Finished .....	1-14
Eight-Bit Transfer .....	1-15
Random Access Drums .....	1-15
High-Speed Disk File .....	1-15
Angular Position Indicator .....	1-15
Paper Tape Equipment .....	1-16
Data Communication Equipment .....	1-16
Consoles .....	1-17
Visual Information Projection (VIP) Devices .....	1-18
Teller Terminal Equipment .....	1-20
Features and Power Modules .....	1-20
Advanced Programming .....	1-20
Program Interrupt .....	1-20
Edit Instruction .....	1-20
Storage Protection .....	1-21
Extended Multiprogramming and Eight-Bit Transfer .....	1-21
Scientific Unit and Scientific Subprocessor .....	1-21
High-Resolution Clock .....	1-21
Expanded Instruction Package .....	1-22
Section II	
The Central Processor .....	2-1
Main Memory .....	2-1



CONTENTS (cont)

	Page
Section II (cont)	
Memory Cycle .....	2-3
Control Memory .....	2-4
Address Registers .....	2-5
Read/Write Counters .....	2-5
Arithmetic Unit .....	2-8
Control Unit .....	2-9
Input/Output Traffic Control .....	2-9
Data Transfer Rates .....	2-9
Memory Access Distribution .....	2-10
Memory Access Distribution of the Type 2041	
Processor .....	2-12
Memory Access Distribution of the Type 2041A	
Processor .....	2-13
Memory Access Distribution of the Type 2051C	
Processor .....	2-13
Memory Access Distribution of the Type 2051A	
Processor .....	2-14
Memory Access Distribution of the Type 2061	
Processor .....	2-16
Memory Access Distribution of the Type 2071	
Processor .....	2-17
Interlocking Read/Write Channels .....	2-17
Variable-Speed Read/Write Channels .....	2-17
Buffered Sectors .....	2-18
Buffered Sector Operation .....	2-18
Buffered Mode .....	2-18
Direct-Access Mode .....	2-19
Buffered Sector Restrictions .....	2-19
Programming Considerations .....	2-19
Extended I/O Indicator .....	2-19
Testing Peripheral Control Unit Busy Status .....	2-19
Escape Codes .....	2-20
Storage Protection Feature .....	2-21
Index Registers .....	2-21
Central Processor Modes .....	2-21
Internal Interrupt .....	2-22
Violations of Storage Protection .....	2-23
Proceed Indicator .....	2-25
Extended Multiprogramming and Eight-Bit Transfer .....	2-26
Storage Protection with Base Relocation .....	2-26
External Interrupt Masking .....	2-27
Instruction Timeout .....	2-27
Eight-Bit Transfer Capability .....	2-28
Privileged SCR Instruction .....	2-29
Privileged BCT Instruction .....	2-29
High-Resolution Clock .....	2-30
Accounting Timer Register .....	2-30
External Interrupt Mode .....	2-30
SCR and LCR Instructions .....	2-31
High-Resolution Clock Allow .....	2-31
Interrupt Processing .....	2-31
External Interrupt .....	2-31
Internal Interrupt .....	2-32
Interrupt Programming .....	2-33
Peripheral Control Interrupt .....	2-35

CONTENTS (cont)

	Page
Section III	
Data Format .....	3-1
Variable Field Length .....	3-1
Instruction Format .....	3-2
Operation Code .....	3-2
A- and B-Addresses .....	3-2
Variant Character .....	3-3
Summary .....	3-3
Organization of Data in Main Memory .....	3-4
Fields .....	3-4
Items .....	3-5
Records .....	3-6
Summary .....	3-6
Magnetic Tape Data Format .....	3-7
Punched Card Format .....	3-9
Disk Format .....	3-10
Data Conventions .....	3-10
Track Format .....	3-11
Record Format .....	3-11
Address Mark .....	3-11
Header Area .....	3-11
Data Area .....	3-14
Track-Linking Record .....	3-15
Section IV	
Addressing .....	4-1
Basic Concepts .....	4-1
Registers Used in Addressing .....	4-3
Sequence Register (SR) .....	4-3
Change Sequence Register (CSR) .....	4-3
External Interrupt Register (EIR) .....	4-3
Internal Interrupt Register (IIR) .....	4-4
A-Address Register (AAR) .....	4-4
B-Address Register (BAR) .....	4-4
Summary .....	4-4
Addressing Modes .....	4-5
Two-Character Addressing Mode .....	4-5
Three-Character Addressing Mode .....	4-6
Four-Character Addressing Mode .....	4-8
Address Modification .....	4-8
Index Registers .....	4-9
Index Register Map .....	4-9
Three-Character Address .....	4-10
Indirect Addressing .....	4-10
Indexed Addressing .....	4-11
Four-Character Address .....	4-12
Indirect Addressing .....	4-12
Indexed Addressing .....	4-13
Treatment of Addresses Larger Than a Memory's	
Maximum Address .....	4-15
Potential Addresses Within Address Register Range .....	4-15
Potential Addresses Outside Address Register Range .....	4-15
Explicit Addressing, Implicit Addressing, and Chaining .....	4-16
Section V	
Easycoder Programming .....	5-1
Introduction .....	5-1
The Symbolic Language .....	5-3
The Assemblers .....	5-3

CONTENTS (cont)

	Page
Section V (cont)	
Coding Form .....	5-5
Card Number (Card Columns 1-5) .....	5-5
Type (Card Column 6) .....	5-6
Mark (Card Column 7) .....	5-7
Location (Card Columns 8-14) .....	5-8
Operation Code (Card Columns 15-20) .....	5-12
Operands .....	5-12
Additional Coding Rules .....	5-14
Address Codes .....	5-14
Absolute .....	5-14
Symbolic .....	5-15
Self Reference .....	5-15
Relative .....	5-16
Out-of-Sequence .....	5-17
Blank .....	5-17
Literals .....	5-18
Decimal Literals .....	5-18
Binary Literals .....	5-19
Octal Literals .....	5-19
Alphanumeric Literals .....	5-20
Area Defining Literals .....	5-21
Address Literals .....	5-22
Variant Character .....	5-23
Input/Output Control Characters .....	5-23
Address Modification Codes .....	5-24
Indexed .....	5-24
Indirect .....	5-25
Section VI	
Data Formatting Statements .....	6-1
Introduction .....	6-1
Define Constant with Word Mark — DCW .....	6-2
Numeric Constants .....	6-2
Decimal Constants .....	6-2
Binary Constants .....	6-2
Octal Constants .....	6-3
Alphanumeric Constants .....	6-4
Blank Constants .....	6-4
Floating-Point Constants .....	6-5
Define Constant — DC .....	6-5
Reserve Area — RESV .....	6-6
Define Symbolic Address — DSA .....	6-7
Define Area — DA .....	6-7
Easycoder C, D, and OS/2000 Options .....	6-10
Section VII	
Assembly Control Statements .....	7-1
Introduction .....	7-1
Program Header — PROG .....	7-2
Segment Header — SEG .....	7-4
Execute — EX .....	7-5
Transfer — XFR .....	7-6
Origin — ORG .....	7-7
Modular Origin — MORG .....	7-9
Literal Origin — LITORG .....	7-10
Set Address Mode — ADMODE .....	7-12
Equals — EQU .....	7-13
Control Equals — CEQU .....	7-14

CONTENTS (cont)

	Page
Section VII (cont)	
Memory Dump — HSM .....	7-15
Skip — SKIP .....	7-16
Suffix — SFX .....	7-16
Repeat — REP .....	7-17
Generate — GEN .....	7-17
Set Line Number — SETLIN .....	7-18
Set Out-of-Sequence Base — XBASE .....	7-19
Range — RANGE .....	7-20
Clear — CLEAR .....	7-21
End — END .....	7-22
Section VIII	
Instructions .....	8-1
Introduction .....	8-1
Arithmetic Operations .....	8-3
Binary Addition .....	8-3
Binary Subtraction .....	8-3
Decimal Addition .....	8-6
True Add .....	8-6
Complement Add .....	8-6
Decimal Subtraction .....	8-7
Indicators .....	8-8
Multiplication .....	8-8
Division .....	8-10
Arithmetic .....	8-13
Add — A .....	8-15
Subtract — S .....	8-16
Binary Add — BA .....	8-18
Binary Subtract — BS .....	8-19
Zero and Add — ZA .....	8-20
Zero and Subtract — ZS .....	8-22
Multiply — M .....	8-23
Divide — D .....	8-25
Logic .....	8-27
Extract — EXT .....	8-28
Half Add — HA .....	8-29
Substitute — SST .....	8-30
Compare — C .....	8-32
Branch — B .....	8-34
Branch on Condition Test — BCT .....	8-35
Branch on Character Condition — BCC .....	8-39
Branch if Character Equal — BCE .....	8-42
Branch on Bit Equal — BBE .....	8-44
Control .....	8-47
Set Word Mark — SW .....	8-48
Set Item Mark — SI .....	8-49
Clear Word Mark — CW .....	8-50
Clear Item Mark — CI .....	8-51
Halt — H .....	8-52
No Operation — NOP .....	8-54
Move Characters to Word Mark — MCW .....	8-55
Load Characters to A-Field Word Mark — LCA .....	8-56
Store Control Registers — SCR .....	8-58
Load Control Registers — LCR .....	8-60
Change Addressing Mode — CAM .....	8-62
Change Sequencing Mode — CSM .....	8-66
Extended Move — EXM .....	8-67

CONTENTS (cont)

	Page
Section VIII (cont)	
Move and Translate — MAT .....	8-70
Move Item and Translate — MIT .....	8-74
Load Index/Barricade Register — LIB .....	8-79
Store Index/Barricade Register — SIB .....	8-83
Table Lookup — TLU .....	8-84
Move or Scan — MOS .....	8-87
Interrupt Control .....	8-93
Store Variant and Indicators — SVI .....	8-94
Restore Variant and Indicators — RVI .....	8-98
Monitor Call — MC .....	8-100
Resume Normal Mode — RNM .....	8-101
Editing .....	8-105
Move Characters and Edit — MCE .....	8-106
Input/Output .....	8-111
Input/Output Control Operations .....	8-112
Selecting RWC Assignments for Use in PDT	
Instructions .....	8-112
Considerations in Selecting RWC Assignments .....	8-112
Device Data Transfer Rate .....	8-113
The Processor Being Used .....	8-115
Input/Output Sector to Which Device is	
Connected .....	8-115
Upward Compatibility .....	8-115
Peripheral Data Transfer — PDT .....	8-116
Escape Code (CE) .....	8-128
Peripheral Control and Branch — PCB .....	8-139
Types of Test and Control Operations .....	8-140
Appendix A	
Octal Notation .....	A-1
Octal-Decimal Conversion Procedure .....	A-3
Appendix B	
Miscellaneous Tables .....	B-1
Appendix C	
Instruction Summary .....	C-1
Instructions Formats and Timing .....	C-1
Appendix D	
Scientific Unit and Scientific Subprocessor .....	D-1
Floating-Point Data Format .....	D-1
Floating-Point Numerical Representation .....	D-2
Floating-Point Registers .....	D-4
Indicators .....	D-4
Automatic Formatting in Arithmetic Operations .....	D-5
Prenormalization .....	D-5
Equalization .....	D-5
Postnormalization .....	D-6
Instruction Formats .....	D-6
Programming Considerations .....	D-7
Symbology for Execution Timings .....	D-7
Timing Notes .....	D-8
Data Moving Instructions .....	D-10
Floating-Point Arithmetic Instructions .....	D-13
Data Conversion Instructions .....	D-17
Control Instructions .....	D-20
Binary Integer Arithmetic Instruction .....	D-25

## ILLUSTRATIONS

		Page
Figure 1-1.	Type 220-3 Console .....	1-2
Figure 1-2.	Type 220-6 Console .....	1-2
Figure 1-3.	Type 220-8 Console .....	1-2
Figure 1-4.	Main Memory Size .....	1-5
Figure 1-5.	Main Memory Speed .....	1-5
Figure 1-6.	Peripheral Simultaneity .....	1-5
Figure 1-7.	Basic Input/Output Data Path .....	1-7
Figure 1-8.	Address Assignments and Unit Loads Available in Series 2000 Processors .....	1-8
Figure 1-9.	Data Path During Card Read Operation .....	1-8
Figure 1-10.	Customer Inquiry Handling via Typical Communications Network .....	1-19
Figure 2-1.	Logical Division of Series 2000 Central Processor .....	2-1
Figure 2-2.	Main Memory Functions .....	2-2
Figure 2-3.	One Memory Position .....	2-2
Figure 2-4.	Representation of Characters in Magnetic Core Storage .....	2-2
Figure 2-5.	Typical Control Register Function .....	2-4
Figure 2-6.	Data Flow Between Main Memory and Arithmetic Unit .....	2-8
Figure 2-7.	Control Unit Activities .....	2-9
Figure 2-8.	Input/Output Traffic Control Activities .....	2-10
Figure 2-9.	Data Transfer Intervals During One Peripheral Operation .....	2-11
Figure 2-10.	Logical Decision Performed by Input/Output Traffic Control .....	2-12
Figure 2-11.	Memory Access Distribution in the Type 2041 Processor .....	2-12
Figure 2-12.	Memory Access Distribution in the Basic Type 2041A Processor .....	2-13
Figure 2-13.	Memory Access Distribution in the Type 2051C Processor and Type 2041A Processor with PM1A40 .....	2-14
Figure 2-14.	Memory Access Distribution in the Basic Type 2051A Processor .....	2-14
Figure 2-15.	Memory Access Distribution in the Type 2051A Processor with PM1A50 .....	2-15
Figure 2-16.	Memory Access Distribution in the Type 2071 Processor and Type 2051A Processor with PM1A50 and PM1B50 .....	2-16
Figure 2-17.	Memory Access Distribution in the Type 2061 Processor and Type 2041A Processor with PM1A40 and PM1B40 .....	2-16
Figure 2-18.	Sample Coding for External Interrupt Routine .....	2-34
Figure 2-19.	Sample Coding for Internal Interrupt Routine .....	2-35
Figure 2-20.	Interrupt Signal Generated by Peripheral Control .....	2-36
Figure 3-1.	Conversion of Symbolic Tag to Absolute Memory Addresses .....	3-2
Figure 3-2.	Series 2000 Instruction Formats .....	3-3
Figure 3-3.	Symbolic Representation of Series 2000 Instructions .....	3-4
Figure 3-4.	Consecutive Storage Locations in Main Memory .....	3-4
Figure 3-5.	Data Field Format in Main Memory .....	3-5
Figure 3-6.	Two Item Formats in Main Memory .....	3-5
Figure 3-7.	Record Format in Main Memory .....	3-6
Figure 3-8.	Character Representation on 7-Track Magnetic Tape .....	3-7
Figure 3-9.	Data Format on Magnetic Tape .....	3-8
Figure 3-10.	Punched Card Codes .....	3-9
Figure 3-11.	Relationship Between Items and Records .....	3-10
Figure 3-12.	Relationship Between Items, Records, and Blocks .....	3-11
Figure 3-13.	Data Conventions of Honeywell Mass-Storage Disk Devices .....	3-12
Figure 3-14.	Flag Character Format .....	3-13
Figure 3-15.	Address Field Format .....	3-13
Figure 3-16.	Data Area Format .....	3-14
Figure 3-17.	Track-Linking Record .....	3-15

ILLUSTRATIONS (cont)

	Page
Figure 4-1.	Typical Add Instruction ..... 4-1
Figure 4-2.	Extraction of Data Fields in Typical Add Instruction ..... 4-2
Figure 4-3.	Series 2000 Index Register Map ..... 4-9
Figure 4-4.	Extraction of Three-Character Indirect Address ..... 4-11
Figure 4-5.	Extraction of Indexed Address in Three-Character Mode ..... 4-12
Figure 4-6.	Extraction of Indirect and Indexed Four-Character Addresses ... 4-14
Figure 4-7.	Series 2000 Instruction Format 1 ..... 4-16
Figure 4-8.	Series 2000 Instruction Format 2 ..... 4-17
Figure 4-9.	Series 2000 Instruction Format 3 ..... 4-17
Figure 5-1.	Relationship of Source Program, Assembler, and Object Program ..... 5-2
Figure 5-2.	Two-Character Address Assembly ..... 5-3
Figure 5-3.	Three-Character Address Assembly ..... 5-4
Figure 5-4.	Four-Character Address Assembly ..... 5-4
Figure 5-5.	Easycoder Coding Form ..... 5-5
Figure 5-6.	Assembly of Indexed Address in Three-Character Addressing Mode ..... 5-25
Figure 5-7.	Assembly of Indexed Address in Four-Character Addressing Mode ..... 5-25
Figure 5-8.	Assembly of Indirect Address in Three-Character Addressing Mode ..... 5-26
Figure 5-9.	Assembly of Indirect Address in Four-Character Addressing Mode ..... 5-26
Figure 8-1.	True Add Examples ..... 8-6
Figure 8-2.	Complement Add Examples ..... 8-7
Figure 8-3.	A- and B-Fields in Multiply Operation ..... 8-9
Figure 8-4.	Factor Locations in Divide Operation ..... 8-11
Figure 8-5.	Changing Addressing Modes via CAM Instruction ..... 8-65
Figure 8-6.	MAT Operation ..... 8-73
Figure 8-7.	MIT Operation ..... 8-79
Figure 8-8.	Basic Storage Protection ..... 8-80
Figure 8-9.	Storage Protection with Base Relocation ..... 8-80
Figure 8-10.	LIB Variant Character ..... 8-81
Figure 8-11.	TLU Operation ..... 8-88
Figure 8-12.	C4 Variant for 9-Track Tape Units ..... 8-132
Figure 8-13.	Format of Type 243 PDT C3 Variant ..... 8-136
Figure D-1.	Floating-Point Data Format in Main Memory ..... D-1
Figure D-2.	Floating-Point Accumulator Data Format ..... D-2
Figure D-3.	Decimal Data Format in Main Memory ..... D-18

TABLES

Table 1-1.	Punched Card Equipment ..... 1-9
Table 1-2.	High-Speed Printers ..... 1-10
Table 1-3.	Magnetic Tape Units ..... 1-11
Table 1-4.	Disk Pack Drives and Disk Subsystems ..... 1-13
Table 1-5.	Disk Pack Drive Features ..... 1-14
Table 1-6.	Random Access Drum Units ..... 1-15
Table 1-7.	High-Speed Disk File ..... 1-16
Table 1-8.	Paper Tape Equipment ..... 1-16
Table 1-9.	Data Communication Equipment ..... 1-17
Table 1-10.	Console Equipment ..... 1-18
Table 1-11.	Visual Information Projection Devices ..... 1-18

TABLES (cont)

	Page
Table 1-12.	Teller Terminal Equipment ..... 1-20
Table 1-13.	Model 2040A Power Modules ..... 1-22
Table 1-14.	Model 2050A Power Modules ..... 1-22
Table 2-1.	Size of Control Memory Registers ..... 2-4
Table 2-2.	Control Memory Registers ..... 2-6
Table 2-3.	Controls/Devices Connectable to Buffered Sectors ..... 2-20
Table 2-4.	Clock Characteristics ..... 2-30
Table 2-5.	Summary of Interrupt/Allow Function Control and Test Operations ..... 2-37
Table 3-1.	Summary of Internal Data Formats ..... 3-6
Table 4-1.	Index Register Addresses in Three-Character Addressing Mode ..... 4-11
Table 4-2.	Index Register Addresses in Four-Character Addressing Mode ..... 4-13
Table 4-3.	Active Address Bits in Series 2000 Single-Character Processors ..... 4-14
Table 5-1.	Set I Punctuation Indicators ..... 5-7
Table 5-2.	Set II Punctuation Indicators (Easycoder C, D, and OS/2000) ... 5-8
Table 6-1.	Data Formatting Statements ..... 6-1
Table 7-1.	Assembly Control Statements ..... 7-1
Table 8-1.	Symbology Used in Series 2000 Instruction Descriptions ..... 8-2
Table 8-2.	Series 2000 Add and Subtract Operations ..... 8-3
Table 8-3.	Binary Addition Table ..... 8-3
Table 8-4.	Algebraic Signs in Decimal Addition ..... 8-6
Table 8-5.	Decimal Arithmetic Sign Conventions ..... 8-8
Table 8-6.	Multiply Sign Conventions ..... 8-9
Table 8-7.	Divide Sign Conventions ..... 8-11
Table 8-8.	SENSE Switch Test Conditions for BCT Instruction ..... 8-36
Table 8-9.	Indicator Test Conditions for BCT Instruction ..... 8-37
Table 8-10.	BCT Instruction Variant Characters ..... 8-38
Table 8-11.	BCC Test Conditions ..... 8-41
Table 8-12.	Control Register Contents Stored by SCR Instruction ..... 8-58
Table 8-13.	Control Registers Stored by SCR Instruction ..... 8-59
Table 8-14.	Control Register Contents Loaded by LCR Instruction ..... 8-61
Table 8-15.	Modes Specified by Variant Character in CAM Instruction ..... 8-63
Table 8-16.	Extended Move Conditions ..... 8-68
Table 8-17.	Size of Information Units in MIT Operation ..... 8-75
Table 8-18.	Correspondence Between LIB Setting and Barricade Location ..... 8-81
Table 8-19.	Move or Scan Conditions ..... 8-89
Table 8-20.	Information Stored by SVI Instruction ..... 8-94
Table 8-21.	Information Restored by RVI Instruction ..... 8-98
Table 8-22.	Special Characters in MCE Instruction ..... 8-107
Table 8-23.	Minimum RWC Capacity Requirements for Series 200/2000 Peripheral Devices ..... 8-113
Table 8-24.	Description of PDT I/O Control Character C1 (RWC Assignment) ..... 8-117
Table 8-25.	Escape Codes ..... 8-129
Table 8-26.	Description of PDT I/O Control Character C2 (Peripheral Control Designation) ..... 8-129
Table 8-27.	Summary of PDT I/O Control Characters ..... 8-132
Table 8-28.	C3 Coding for Type 209 and 209-2 Paper Tape Readers ..... 8-136
Table 8-29.	C3 Coding for Type 210 Paper Tape Punch ..... 8-137
Table 8-30.	C3 Coding for Type 222 Printers ..... 8-137
Table 8-31.	C3 Coding for Type 270A Random Access Drum ..... 8-137



TABLES (cont)

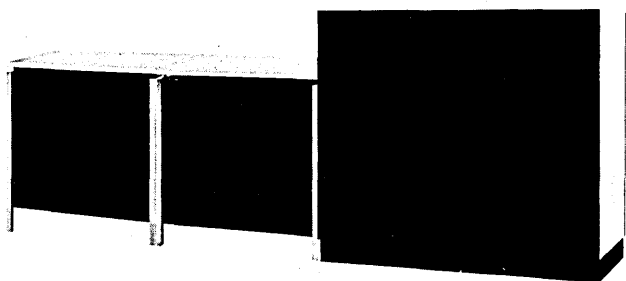
	Page
Table 8-32.	Summary of PDT I/O Control Characters for Type 286 Multiline Communication Controller ..... 8-138
Table 8-33.	Type 286-1, -2, -3 Line Control Instructions ..... 8-138
Table 8-34.	Summary of PCB I/O Control Characters ..... 8-142
Table 8-35.	Summary of PCB I/O Control Characters for Type 286 Multiline Communication Controller ..... 8-152
Table 8-36.	PCB Control Characters C5 through C15 for Type 286-4, -5, -6, -7 Line Control Instructions ..... 8-154
Table A-1.	Binary-Octal Equivalents ..... A-1
Table A-2.	Decimal-Octal Conversion Table ..... A-2
Table B-1.	Control Register Designations ..... B-1
Table B-2.	Extended Move (EXM) Conditions ..... B-2
Table B-3.	Branch on Condition Test (BCT) SENSE Switch Conditions ..... B-3
Table B-4.	Branch on Condition Test (BCT) Indicator Conditions ..... B-4
Table B-5.	Branch on Character Condition (BCC) Conditions ..... B-5
Table B-6.	Series 2000 Character Codes ..... B-7
Table B-7.	Binary, Octal, and Decimal Equivalents ..... B-8
Table B-8.	Powers of 2 ..... B-8
Table B-9.	Move or Scan Variants ..... B-9
Table C-1.	Instruction Summary — Timing Formulas for Models 2040 ..... C-4
Table C-2.	Instruction Timings for Models 2040A, 2050, and 2060 ..... C-8
Table C-3.	Instruction Timings for Models 2050A and 2070 ..... C-11
Table C-4.	Timings for Decimal Multiply and Divide — Model 2040 ..... C-16
Table D-1.	Floating-Point Numerical Representation of Mantissas ..... D-3
Table D-2.	Floating-Point Numerical Representation of Exponents ..... D-3
Table D-3.	Execution Timings in Memory Cycles ..... D-9
Table D-4.	Numerical Representation of Decimal Word Data ..... D-19

SECTION I  
SERIES 2000 COMPONENTS

Series 2000 is a family of modularly designed, compatible data processing systems. Each model within the system consists of two basic elements: a central processor, and an array of peripheral devices connected to that processor. Most peripheral equipment can be attached to any processor and the number of connectable devices is limited only by certain individual power and circuitry restrictions.

The processing power of any of the central processors discussed in this manual can be increased at any time by the addition of peripheral devices and/or optional hardware features. The components of a Series 2000 system discussed in this section include: (1) the central processor; (2) the processor's interface with the peripherals; (3) the peripherals; and (4) the expansion of processing power through the addition of optional hardware features.

CENTRAL PROCESSOR



The central processor is the computing and control center of a Series 2000 model; instructions processed within the central processor control the operations of the entire computer. A Series 2000 processor is functionally divided into three units; storage, control, and arithmetic. The storage unit provides magnetic core storage for both the program instructions and the data to be processed according to these instructions; it is also used to contain the resultant data. The control unit directs the operation of the entire computer by selecting, interpreting, and controlling the execution of all program instructions. It controls not only the flow of information within the central processor but also the flow of data

between the central processor and all peripheral equipment. The arithmetic unit performs such operations as addition, subtraction, multiplication, division, and comparison, as directed by the control unit.

## CONSOLES

The primary communication medium between the operator and the central processor is the Type 220 console, of which three versions are available. In the Type 220-3 Console (Figure 1-1) and the Type 220-6 Console (Figure 1-2), most control functions, including that of direct access to the processor, are performed by means of a console typewriter. This typewriter can also be used as a peripheral device, operating under program control, or as a logging typewriter by which the operator can make essential notes about the program in progress. A console control panel contains power switches, SENSE switches, and certain check condition indicators.

The Type 220-8 Visual Information Control Console (VICC) includes a keyboard and control panel, a display screen, and a console control (Figure 1-3). The basic console can be expanded to include a second display screen, a serial printer, a remote display, and a display switch. The 220-8 performs all of the control functions of the 220-3 and 220-6, and provides Series 2000 systems with vastly increased operator flexibility.

A Type 220 Console is required on all Series 2000 Systems. The Type 220-3 (or, optionally, the Type 220-8) is required on the Type 2041 Central Processor. The Type 220-6 (or, optionally, the Type 220-8) is required on the Type 2041A, 2051 and 2061 Central Processors. The Type 220-6A (or, optionally, the Type 220-8) is required on the Type 2051A Central Processor. The Type 220-8 is standard on the Type 2071 Central Processor.

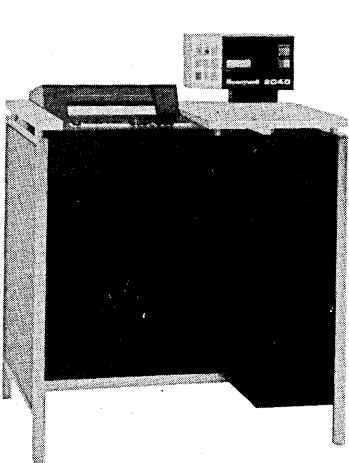


Figure 1-1.  
Type 220-3 Console

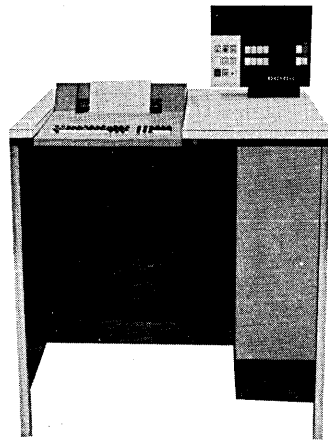


Figure 1-2.  
Type 220-6 Console

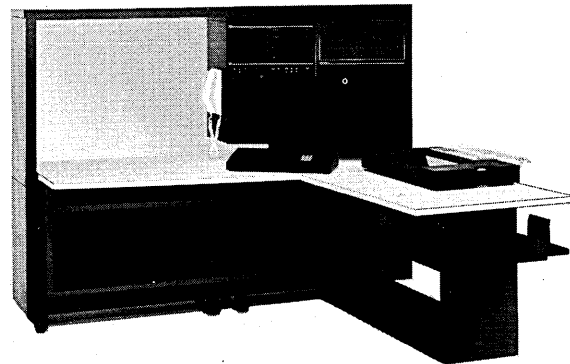


Figure 1-3.  
Type 220-8 Console  
(VICC)

## STANDARD PROCESSING MODE

The central processor performs arithmetic and logical operations as directed by the instructions of an internally stored program. These instructions are read into memory from an input medium such as punched cards, magnetic tape, punched paper tape, disk, or drum. Control circuitry within the processor then selects, interprets, and executes these instructions.

Normally, the instructions are executed sequentially. Branch instructions are provided, however, which make it possible to skip over a group of instructions or otherwise change the sequence of the program.

## INTERRUPT PROCESSING MODE

Sequential instruction execution is changed temporarily whenever the central processor is interrupted. Any one of four sources can "demand" access to the central processor by generating an interrupt signal, which turns on a central processor interrupt indicator. Once an interrupt indicator is detected as being on, a hardware response is made: information concerning the current status of the processor (including the setting of the sequence register) is stored, and a branch is made to a stored routine that identifies and services the demand. Thus, programmed tests need not be made to detect the presence of an interrupt condition — the entire process of detecting and responding to an interrupt signal is an automatic hardware function. After the stored service routine has been executed, control is returned to the interrupted routine at the point where the interruption occurred and the previous status is restored. Two kinds of interrupts can occur in the system: external interrupts and an internal interrupt. A detailed description of interrupt functions and programming for interrupt processing is presented in Section II.

### External Interrupts

The three sources of external interrupts are:

1. Peripheral Control — The control connected to any Series 2000 peripheral device can generate an interrupt signal under program control. For instance, a data communication controller which services one or a number of communication lines and devices may generate a real-time demand on central processor time to handle a customer inquiry from a remote terminal. The current operations of the processor are temporarily interrupted so that the inquiry may be serviced. A routine to read the inquiry and to answer the question from a stored customer file is automatically executed, and a response is sent back to the terminal.
2. Console — The operator can interrupt the central processor by pressing the INTERRUPT button on the console. The source of such "on-site" interrupts is made available to the program by the execution of a single instruction at the beginning of the interrupt service routine.
3. Program Instruction — One instruction in the Series 2000 repertoire, the Monitor Call instruction, is used to generate an interrupt condition. For programming convenience, the activation (or "calling") of the monitor program can be accomplished by means of this instruction.

### Internal Interrupt

When Storage Protection is in effect, an internal interrupt condition, caused by certain violations of a protected memory area or attempts to address nonexistent memory locations, can also occur. Internal interrupts are of lower priority than external interrupts, so that a

processor executing an external interrupt service routine does not respond to an internal interruption until the routine is completed. Processing of internal interrupts is described in Section II.

### ADDRESSING MODES

Due to the binary addressing system used in referencing the individual core storage locations within the central processor, an address portion of a machine-language instruction can occupy two, three, or four characters of memory. The number of character positions employed is controlled by two instructions: the assembly control statement ADMODE, and the Change Addressing Mode (CAM) instruction. Any main memory address can be referenced in any addressing mode by having the central processor prefix the address expressed in the instruction with a binary value previously set in an address register. Thus, the programmer has the ability to set the address registers to some high module, switch to the two-character addressing mode, and still continue to address that module. This utilization of the smallest number of character positions to express any main memory address results in a reduction in the amount of memory required for a particular program.

### ITEM-MARK TRAPPING MODE

The item-mark trapping mode, which can be set via the CAM instruction, causes the processor to treat and execute any instruction containing an item-marked op code as if it were a Change Sequencing Mode (CSM) instruction, which results in a transfer of control to an instruction stored at a prespecified location. This processing mode is used extensively in Liberator systems and can also be used to control program branching.

### PROCESSING POWER

The power of any processor within Series 2000 can be defined as the total effect of its main memory size, its internal speed, and its degree of peripheral simultaneity.

Main memory size, for the Series 2000 Models 2040 through 2070, ranges from a minimum of 49,152 character locations to a maximum 1,048,576 locations. Figure 1-4 illustrates the modular main memory sizes of the various processor types.

The internal speed of a processor is measured in terms of a memory cycle (i. e., the time required to read and restore the contents of a unit location). The unit location used by the Type 2041 is a single, 6-bit character location. The unit location of the Types 2041A, 2051C, and 2061 is two successive character locations. The unit location of the Types 2051A and 2071 is four successive character locations. The processors that manipulate more than one location at a time are called multicharacter processors. Memory cycle speeds range from 1.6 microseconds per single-character fetch to 1 microsecond per four-character fetch (see Figure 1-5).

Peripheral simultaneity is a key feature of Series 2000 processors. Among the processors described in this manual, from 8 to 16 simultaneous input/output operations can be performed concurrently with internal computing (see Figure 1-6).

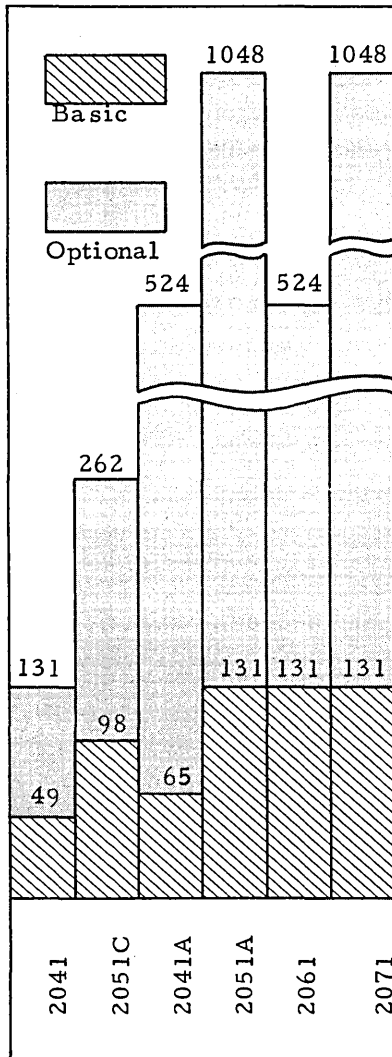


Figure 1-4. Main Memory Size

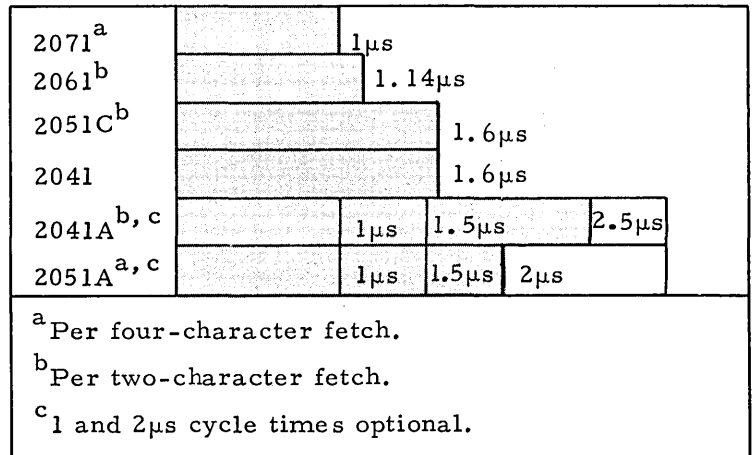


Figure 1-5. Main Memory Speed

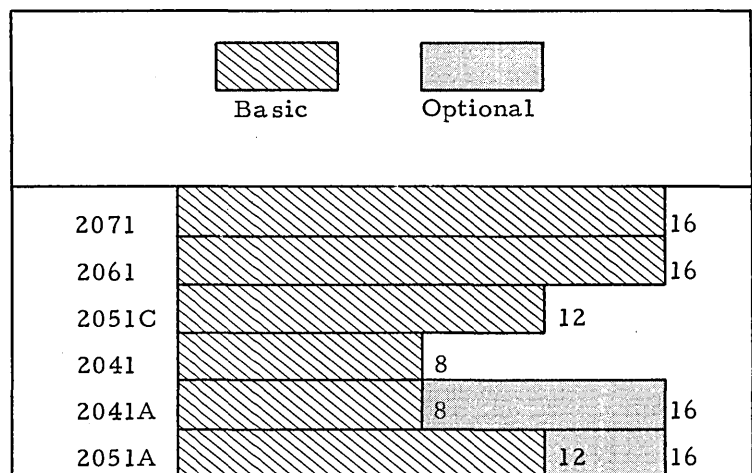


Figure 1-6. Peripheral Simultaneity

## PERIPHERAL INTERFACE

The array of peripheral devices available with Series 2000 processors include: consoles, punched card equipment, high-speed printers, magnetic tape units, paper tape equipment, direct-access devices (disk and drum), MICR reader-sorters, Visual Information Projection units, and various data communication controllers, a front-end network processor, and remote terminals. Also included are computer-to-computer adapters, a time-of-day clock, and a standard interval timer and selector.

Information is transferred between any one of these devices and the central processor by means of a single stored-program instruction — the Peripheral Data Transfer instruction described in Section VIII. By coding various control characters in this instruction, the programmer specifies the direction of data transfer (into or out of the processor), the specific device involved in the transfer, the data path over which information is to be transferred, and any other information necessary to define the input/output operation (e. g., the number of lines to be spaced during printer operations). The actual communication with the central processor is not made by the particular peripheral device but by the peripheral control connected to that device.

### PERIPHERAL CONTROL

A peripheral control regulates the transfer of data between a processor and a peripheral device. The control compensates for the difference in the data transfer rates of the processor and the peripheral device by temporarily storing each character of transmitted information until either the processor or the device is ready to receive the character. The control also converts each character into the code used by the intended recipient (e. g., the card reader control converts a character from Hollerith code to the internal six-bit code of the central processor). As each character is transferred to the control, it is also checked for accuracy by the control. One particularly significant feature of the peripheral control is that it operates independently of the central processor and requires access to the main memory only when information transfers are performed. In particular, all of the previously mentioned activities of the control — temporarily storing, converting, and checking the information — do not involve the central processor in any way. When each character of information is transferred, one main memory cycle is allocated for the transfer.

Some peripheral devices require one peripheral control per device (e. g., a card reader). Other devices can be connected in multiple fashion to a single peripheral control (e. g., up to eight 1/2-inch magnetic tape units can be directed by a single control). The number of Series 2000 devices connectable to a peripheral control is shown in Tables 1-1 through 1-13 on the following pages. The information listed under "Unit Loads" and "Address Assignments" in these tables is used in determining the number of peripheral controls that can be connected to a Series 2000 processor. This connection is described below.

### PERIPHERAL DATA TRANSFER OPERATION

One of the major features of Series 2000 is the degree of peripheral simultaneity that can be achieved by the various processors. The Type 2041 processor can perform up to eight peripheral operations simultaneously; the other processors can perform as many as sixteen simultaneous peripheral operations. While all these operations are being executed, the central processor continues its internal processing. The ability to perform simultaneous peripheral

operations derives from an internal unit of the central processor – the input/output traffic control – that guarantees a peripheral control access to main memory when data is to be transferred. The manner in which the traffic control does this is explained in Section II. The data path used by the traffic control to transfer data (see Figure 1-7) is described below.

#### Peripheral Addresses and Unit Loads

When installed in a Series 2000 computer system, peripheral controls (and their associated devices) are permanently connected to the system. Each control is assigned one or two addresses, depending on the number of directions in which it can transfer data. It is by these peripheral addresses that the controls are designated in input/output instructions. For example, a card reader and its associated control can transfer data in only one direction – into the central processor. The reader control is therefore assigned one address by which it is always designated in an instruction. A combination card reader/card punch and control can transfer data in two directions – into and out of the processor. It is thereby assigned two addresses: one address is used to specify an input (card read) operation, while the other is used to specify an output (card punch) operation.

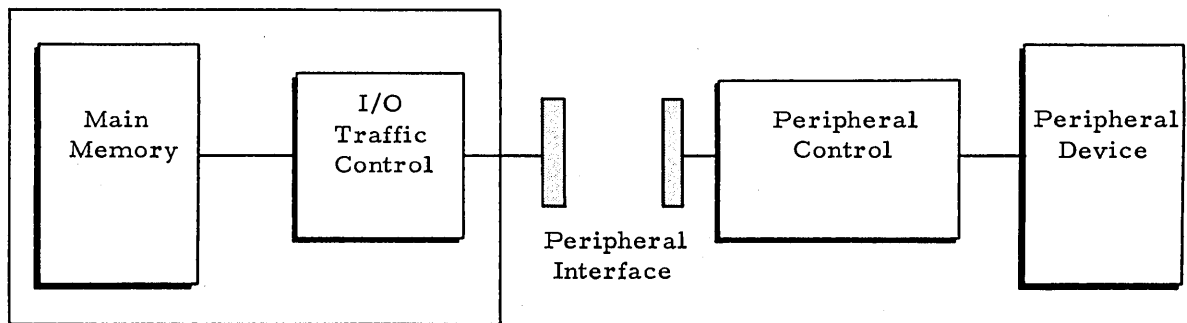


Figure 1-7. Basic Input/Output Data Path

The number of peripheral controls which a Series 2000 processor can accommodate depends upon four factors: (1) the number of "unit loads" of power required by the controls to be connected; (2) the number of unit loads available from the processor; (3) the number of peripheral addresses required to operate the controls; and (4) the number of address assignments that the processor provides. A peripheral control may require either one or two unit loads of power and either one or two addresses. The numbers of unit loads and address assignments available with each Series 2000 processor are shown in Figure 1-8. The numbers of unit loads and address assignments required by each peripheral control are shown in the summary tables of the peripheral equipment (Tables 1-1 through 1-13).



## Read/Write Channel

Note that the permanent connection established in Figure 1-7 is incomplete: there is no connection across the peripheral interface. The input/output data path is completed by one or more "read/write channels," inserted in the data path when the input/output instruction is executed. (More than one read/write channel is sometimes necessary in order to accommodate the high data transfer rates of some devices.) A read/write channel is not permanently connected to any peripheral control but is assigned by the programmer or operating system to specify the data path between a control and the processor.

Type	Address Assignments/Unit Loads	
2071	80	
2061	48	
2051	32	
2041	32	
2041A	32	48
2051A	32	80



	Basic		Optional
---	-------	---	----------

Figure 1-8. Address Assignments and Unit Loads Available in Series 2000 Processors

When the programmer codes an input/output instruction, he specifies among other things the address of the peripheral control that is to send or receive data and the read/write channel(s) over which the data transfer is to take place. When the instruction is executed, the specified read/write channel is automatically inserted in the peripheral interface. For example, Figure 1-9 shows the data path formed during the execution of an input/output instruction in which the programmer specifies that the card reader control is to transfer data over read/write channel 2 (RWC2). The specified channel remains in the interface only for the duration of the card read operation. When the data transfer terminates, RWC2 is automatically removed from the interface and is available for reassignment by another instruction.

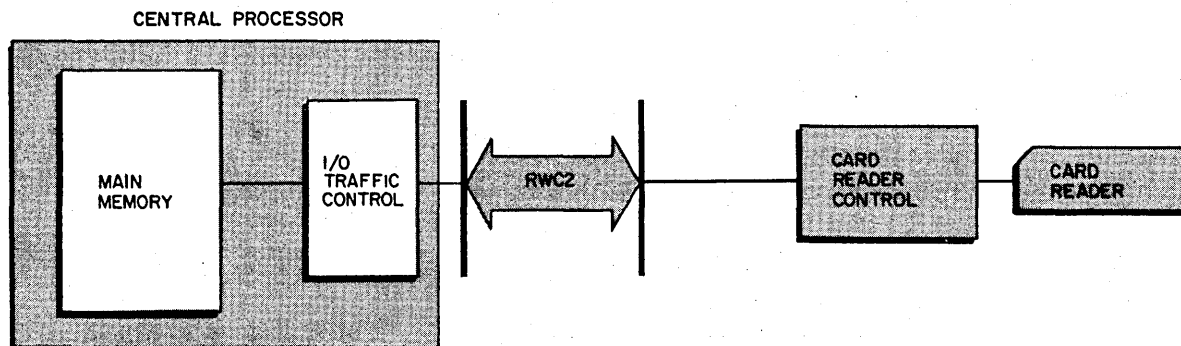


Figure 1-9. Data Path During Card Read Operation

Read/write channels are the key to the achievable simultaneity in a Series 2000 model: the number of read/write channels associated with a particular processor determines the number of peripheral operations that can be performed simultaneously by the processor (see Figure 1-6).

## PERIPHERAL EQUIPMENT

Series 2000 includes a wide variety of peripheral devices not only of different kinds, but also on several performance levels for the same kind.

### PUNCHED CARD EQUIPMENT

Seven different punched card units are offered: five card readers, a card punch, and a card reader/punch. Table 1-1 lists the card devices available within Series 2000. Note that a card device requires either one or two address assignments depending on the number of functions the device performs.

Table 1-1. Punched Card Equipment

Device		Read/Punch Speed	No. Devices Per Control	No. Unit Loads Required by Control & Device	Address Assignments
Type	Function				
123 <sup>a</sup>	Card Reader	400 cards/minute	1	1	1
123-2 <sup>a</sup>	Card Reader	600 cards/minute	1	1	1
123-4 <sup>a</sup>	Card Reader	1050 cards/minute	1	1	1
223	Card Reader	800 cards/minute	1	1	1
223-2	Card Reader	1050 cards/minute	1	1	1
214-1	Card Punch	100-400 cards/minute	1	1	1
214-2	Card Reader/Punch	Read: 400 cards/minute Punch: 100-400 cards/minute	1	1	2

<sup>a</sup> Available only on Models 2040, 2040A, and 2050A integrated controls.

### HIGH-SPEED PRINTERS

Ten types of printers (see Table 1-2) produce printed reports, listings, etc., at speeds that vary from 300 to 1,300 lines per minute. Processed information is printed from any programmer-assigned area in memory. A single program instruction — the Move Characters and Edit instruction — allows the programmer to punctuate the output data, suppress zeros, and insert identifying symbols in the data prior to printing.

## Print Buffer

With the addition of the Print Buffer (Feature 036) to the 222-3, 222-3N, 222-4, 222-6, or 222-7 printer, the amount of central processor memory cycles required for data transfer to the printer is reduced to less than 1%. Thus, more than 99% of the central processor time is available for program execution.

Table 1-2. High-Speed Printers

Type <sup>a</sup>	Print Speed (lines/minute)	No. Printers Per Control	No. Unit Loads Required	Address Assignments
112 <sup>b</sup>	300	1	1	1
112-3 <sup>b</sup>	650	1	1	1
122-3 <sup>b</sup>	650	1	1	1
122-4 <sup>b</sup>	950	1	1	1
122-6 <sup>b</sup>	1100	1	1	1
222-3	650-1, 300	1	1	1
222-3N	650	1	1	1
222-4	950-1, 266	1	1	1
222-6	1100	1	1	1
222-7	300	1	1	1

<sup>a</sup>All printers can have 120 (standard) or 132 (optional) print positions.

<sup>b</sup>Available only on Models 2040, 2040A, and 2050A integrated controls.

## MAGNETIC TAPE UNITS

Magnetic tape is a compact, highly versatile medium for the storage of programs and data files. A complete family of units is available with the Series 2000 processors (see Table 1-3). These tape units transfer data at speeds ranging from 7,200 to 224,000 characters per second.

### 1200-BPI Recording Density

The 1200-bits-per-inch recording density (Feature 054) provides the Type 204B-9 Magnetic Tape Unit with the capability of reading and writing data at a density of 1200 bits per inch (bpi) on high-resolution 1/2-inch magnetic tape. The 1200-bpi recording density enables the 204B-9 to achieve a transfer rate of 144,000 characters per second. Similarly, the 1200-bpi recording density (Feature 055) allows the 204B-7 to operate at 43,200 characters per second.

### 1600-BPI Recording Density

A 1600-bits-per-inch recording density is standard on the Type 204D-1, -3, and -5; 204D-3A and -5A; and 204F-1, -3, and -5 tape units. This density, in conjunction with tape

transport speeds of 35, 70, or 105 inches per second, enables maximum transfer rates of 74,600, 149,300, or 224,000 characters per second when the compatibility mode is specified.

Dynamic Tape Addressing

Feature 056, Dynamic Tape Addressing, allows the changing of tape unit addresses while the processor is in the RUN mode; i. e., entering the STOP mode is unnecessary. This feature is available for use with all 1/2-inch magnetic tape controls.

Table 1-3. Magnetic Tape Units

Type	Data Transfer Rate (characters/second)	No. Devices Per Control	No. Unit Loads Re- quired by Control & Devices	Address Assign- ments
1/2-Inch Magnetic Tape Units (Seven-Track)				
204B-1 204B-2	} 7,200/19,980	1-8	2	2
204B-3 204B-4	} 16,000/44,500	1-8	2	2
204B-5	24,000/66,700	1-8	2	2
204B-7	20,000/28,800 (or 7,200/ 28,800 or 20,000/43,200 or 28,800/43,200)	1-8	2	2
204B-8	44,500/64,000 (or 16,000/ 64,000)	1-8	2	2
204B-9	66,700/96,000 (or 24,000/ 96,000 or 66,700/144,000 or 96,000/144,000)	1-8	2	2
1/2-Inch Magnetic Tape Units (Nine-Track)				
204D-1 <sup>a</sup>	37,300/74,600	1-8	2	2
204D-3 <sup>a</sup>	74,600/149,000	1-8	2	2
204D-5 <sup>a, b</sup>	112,000/224,000	1-8	2	2
204D-3A	74,600/149,300	1-8	2	2
204D-5A <sup>b</sup>	112,000/224,000	1-8	2	2
204F-1	37,300/74,600	1-8	2	2
204F-3	74,600/149,300	1-8	2	2
204F-5 <sup>b</sup>	112,000/224,000	1-8	2	2
<sup>a</sup> As-available only. <sup>b</sup> Restricted to use on buffered sector. Not available on Model 2040.				

## IBM Magnetic Tape Compatibility

IBM Magnetic Tape Compatibility (Features 050 and 051 on the seven-track tape units; Feature 052 on the nine-track tape units) enables the processing of tapes that have been written at appropriate densities by IBM tape units or to write tapes to be read by these units. This capability includes end-of-file mark recognition (tape-mark sensing), and the ability to translate between IBM even-parity BCD code and the Honeywell Series 200/2000 central processor code. Feature 052 is applicable to nine-track units in seven-track compatibility mode and seven-track tape units connected to the 203D tape control.

## EBCDIC Code Translation

On Type 203D and 203F Tape Controls with nine-track tape units, Feature 1052, an EBCDIC Code Translator, enables hardware conversion of a subset of the IBM eight-level Extended Binary Coded Decimal Interchange Code (EBCDIC) to the Series 200/2000 six-bit central processor code, and vice versa.

## DISK PACK DRIVES

Honeywell disk pack drives combine the unlimited shelf storage of magnetic tape with the fast direct access of disk storage. With the use of removable disk packs, data can be recorded, stored indefinitely (like magnetic tape), and rapidly reinserted in an on-line drive. The various disk pack drives are listed in Table 1-4.

Various features are available for use with the disk pack drives. Table 1-5 lists the features and indicates whether the feature is standard or optional. When a feature is optional to a disk pack drive, the appropriate feature number is given.

Table 1-4. Disk Pack Drives and Disk Subsystems

Single-Spindle Units					
Type	Data Storage Capacity Per Drive (characters)	Data Transfer Rate (characters/second)	Max. Drives Per Control	Unit Loads Required	Address Assignments
258	4.6 million	208,333	8	1	2
258B	4.6 million	147,500	8	1	2
259	9.2 million	208,333	8	1	2
259B	9.2 million	147,500	8	1	2
273	18.4 million	208,333	8	1	2
Multispindle Units					
Type	Data Storage Capacity Per Drive (characters)	Data Transfer Rate (characters/second)	Max. Spindles Per Control	Unit Loads Required	Address Assignments
274	147.2 million	208,333	9	1	2
278-5	175 million	416,000	5	1	2
278-6	210 million	416,000	6	1	2
278-7	245 million	416,000	7	1	2
278-8	280 million	416,000	8	1	2
278-9	280 million	416,000	8 <sup>1</sup>	1	2
Disk Subsystems					
Type	Data Storage Capacity Per Subsystem (characters)	Data Transfer Rate (characters/second)	No. Drives Per Subsystem	Unit Loads Required	Address Assignments
275-2	36.8 million	208,333	2	1	2
277-2	128 million	167,000-500,000 <sup>2</sup>	2	1	2
279-2	251.4 million	167,000-500,000 <sup>2</sup>	2	1	2
Disk Subsystem Expansion					
Type	Data Storage Capacity Per Additional Drive (characters)	Data Transfer Rate (characters/second)	No. Drives That Can Be Added	Unit Loads Required	Address Assignments
275	18.4 million	208,333	6	1	2
277	64 million	167,000-500,000 <sup>2</sup>	6	1	2
279	125.7 million	167,000-500,000 <sup>2</sup>	6	1	2
<sup>1</sup> The 278-9 has a spare spindle off-line. <sup>2</sup> The 277 and 279 transfer rate is dependent upon central processor RWC allocation.					

Table 1-5. Disk Pack Drive Features

	258	258B	259	259B	273
Feature					
Write Protect Capability	074	074	074	074	074
Dynamic Disk Addressing	076	076	076	076	076
Central Processor Finished	079	079	079	079	079
Standby Eight-Bit Transfer	-	-	-	-	-
	274	275-2	277-2	278-5 through 278-9	279-2
Feature					
Write Protect Capability	Std	074	074	Std	Std
Dynamic Disk Addressing	Std	076	076	Std	076
Central Processor Finished	Std	079	-	Std	-
Standby <sup>a</sup> Eight-Bit Transfer	-	-	Std 077	-	Std 077
<sup>a</sup> Functionally Identical to Central Processor Finished.					

Write Protect Capability

A Write Protect Capability (Feature 074) allows the operator to protect individual disk pack drives from inadvertent writing via an alternate-action push-button switch on each disk drive. In the permit mode the drive operates as specified; in the protect mode, an attempt to write results in a protect violation status in the disk control address register. This register can be interrogated by the programmer.

Dynamic Disk Addressing

Feature 076 (Dynamic Disk Addressing) allows the changing of disk addresses while the processor is in the RUN mode; i. e., entering the STOP mode is unnecessary.

Central Processor Finished

The Central Processor Finished capability (Feature 079) allows the programmer to issue a special Peripheral Control and Branch (PCB) instruction to the disk pack drive. Upon receipt

of this instruction, power is automatically removed from the drive and a visual indicator is illuminated. This feature minimizes setup time.

Eight-Bit Transfer

The eight-bit transfer capability allows data to be transferred between a disk control and a central processor in an eight-bit transfer mode. Data may also be transferred in the standard Series 200/2000 six-bit transfer mode.

RANDOM ACCESS DRUMS

The Series 2000 drum storage capability features a drum control that can direct from one to eight magnetic drums, each capable of storing 2.6 million characters of information (see Table 1-6). Thus, a single drum subsystem can have a total capacity of over 20 million characters. Any record stored on the drum can be located in 27 milliseconds (average) and can be transferred at the rate of 111,000 characters per second.

Table 1-6. Random Access Drum Units

Type	Data Storage Capacity Per Drum	Data Transfer Rate	No. Drums Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
270A-1 through 270A-8	2.6 million characters	111,000 characters/second	1-8	1	2

HIGH-SPEED DISK FILE

The high-speed disk file is a fixed-head storage device that offers high speed performance with fast access time. Up to four devices can be operated with a single control, and thus a control's capacity may amount to over 16.8 million characters. Any record stored on the disk files can be located in 8.6 milliseconds (average).

Angular Position Indicator

Feature 072 (Angular Position Indicator) provides for optimum addressing of the Type 266 High-Speed Disk File. Information is provided at any given time as to the current position relative to 360 degrees of rotation. Under heavy load conditions with many demands waiting to be executed, the average access time of the Disk Files may be substantially reduced.



Table 1-7. High-Speed Disk File

Type	Data Storage Capacity Per Disk File	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
266	4.2 million characters	300,000 char./sec.	1-4	1	2

### PAPER TAPE EQUIPMENT

Paper tape is an ideal medium for recording data that originates at locations distant from a central Series 2000 installation and, as such, becomes particularly significant in data communication networks. A variety of standard commercial codes may be used with this relatively inexpensive medium. Three paper tape devices are offered in Series 2000 (see Table 1-8).

Table 1-8. Paper Tape Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices <sup>a</sup>	Address Assignments
Type	Function				
209	Paper Tape Reader	600 frames/second	1	2	1
209-2	Paper Tape Reader	600 frames/second	1	2	1
210	Paper Tape Punch	120 frames/second	1	2	1

<sup>a</sup>The total load requirements for the combination of a 209 (or 209-2) reader and a 210 punch is 2 unit loads.

### DATA COMMUNICATION EQUIPMENT

To communicate between a central site (e.g., home office) and remote locations (e.g., branch office, warehouse, etc.) the Series 2000 system must have communication equipment. The two types of communication equipment which allow remote communication with a Series 2000 processor are hardwired communication controllers and a front-end network processor.

Honeywell provides two types of hardwired communication controllers:

1. The Type 281 Single-Line Communication Controller
  - a. Type 281-1 (asynchronous)
  - b. Type 281-2 (synchronous)
2. The Type 286 Multiline Communication Controller
  - a. Type 286-1, -2, -3 (character mode)
  - b. Type 286-4, -5, -6, -7 (message mode)

The single-line communication controller interfaces with only one communication line (either leased private, switched network, or direct connect). The multiline communication controller interfaces with multiple lines simultaneously. Depending on the type, from one to 63 lines can be serviced.

The DATANET 2000 Front-End Network Processor can handle from 1 to 120 asynchronous or synchronous lines simultaneously. Unlike hardwired controllers, the DATANET 2000 relieves the Series 2000 processor of the overhead required to support communications. The DATANET 2000 is responsible for code translation, line discipline, core or disk queuing, automatic retransmission upon detection of error, etc.

The characteristics of both hardwired controllers and the front-end processor are given in Table 1-9.

Table 1-9. Data Communication Equipment

Device		Maximum Transmission Rate Per Line	No. Lines Per Controller	Unit Loads Per Controller	Address Assignments
Type	Function				
Hardwired Communication Controllers					
281-1	Asynchronous Single-Line Communication Controller	1800 bits/second	1	1	2
281-2	Synchronous Single-Line Communication Controller	50,000 bits/second	1	2	2
286-1, -2, -3	Character Mode Multiline Communication Controller	4800 bits/second/line	1-63	2	2
286-4, -5, -6, -7	Message Mode Multiline Communication Controller	9600 bits/second/line	1-63	2	2
Front-End Network Processor					
DATANET 2000	Front-End Network Processor	10,800 bits/second/line	1-120	1	2

A major requirement of many communication networks (e.g., inquiry handling or message switching applications) is fast access to a stored file. Files may sometimes be stored in main memory, but for large files main memory storage is economically unfeasible. File storage units (i.e., the disk pack drives or drum units) fulfill the requirements of these applications.

A typical data communication network is shown in Figure 1-10. The pertinent components of this system are: (1) a Series 2000 central processor; (2) a Type 273 Disk Pack Drive; (3) a Type 281 Communication Controller, (4) two data sets<sup>1</sup>; and (5) the remote terminal.

A VIP Single Display Station is used in this example: a keyboard by which the inquiry is transmitted to the central processor, and a CRT which displays the answer to the inquiry in readable form.

### CONSOLES

Characteristics of the Type 220 consoles, described previously, are listed in Table 1-10.

<sup>1</sup> A data set is required to convert the data signals used by the communication control to signals acceptable for transmission over communication lines and vice versa.

Table 1-10. Console Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
Type	Function				
220-3, -6	Operator's Console	Typing speed (input); or 10 char./sec. (output)	n/a	1	2
220-8	Visual Information Control Console	Typing speed (input); 30 char./sec. (printer output); 480 char./sec. (display output).	n/a	1	2

VISUAL INFORMATION PROJECTION (VIP) DEVICES

Cathode-ray tube (CRT) display units — for businesses requiring instantaneous visual access to data stored in computer files — are available to the Series 2000 user. These devices are on-line to the computer, either locally via direct physical connection or from remote locations via communication facilities. An eight-bit code (seven-bit ASCII plus parity) is used for synchronous transmission and a 10-bit code (seven-bit ASCII plus parity and start and stop bits) for asynchronous transmission.

Both single-station and multistation systems are available; both types of systems offer a wide range of screen sizes. Keyboards include both alphanumeric typewriter layout and block-numeric keys as standard features.

Table 1-11. Visual Information Projection Devices

Device		Data Transfer Rate	No. VIP Devices Per Communication Controller	No. Unit Loads Required by Control & Devices	Address Assignments
Type	Function				
2323	Single-Display Station	1200-2400 bps	1	not applicable	not applicable
2317	Multistation Display System	1200-2400 bps	2-36	not applicable	not applicable
765	Asynchronous Single/Dual/Cluster Configurations	2000-2400 bps	1-20	n/a	n/a
775	Synchronous Single/Dual/Cluster Configurations	2000-2400 bps	1-20	n/a	n/a
785	Synchronous Single Station	2000-2400 bps	1-8	n/a	n/a

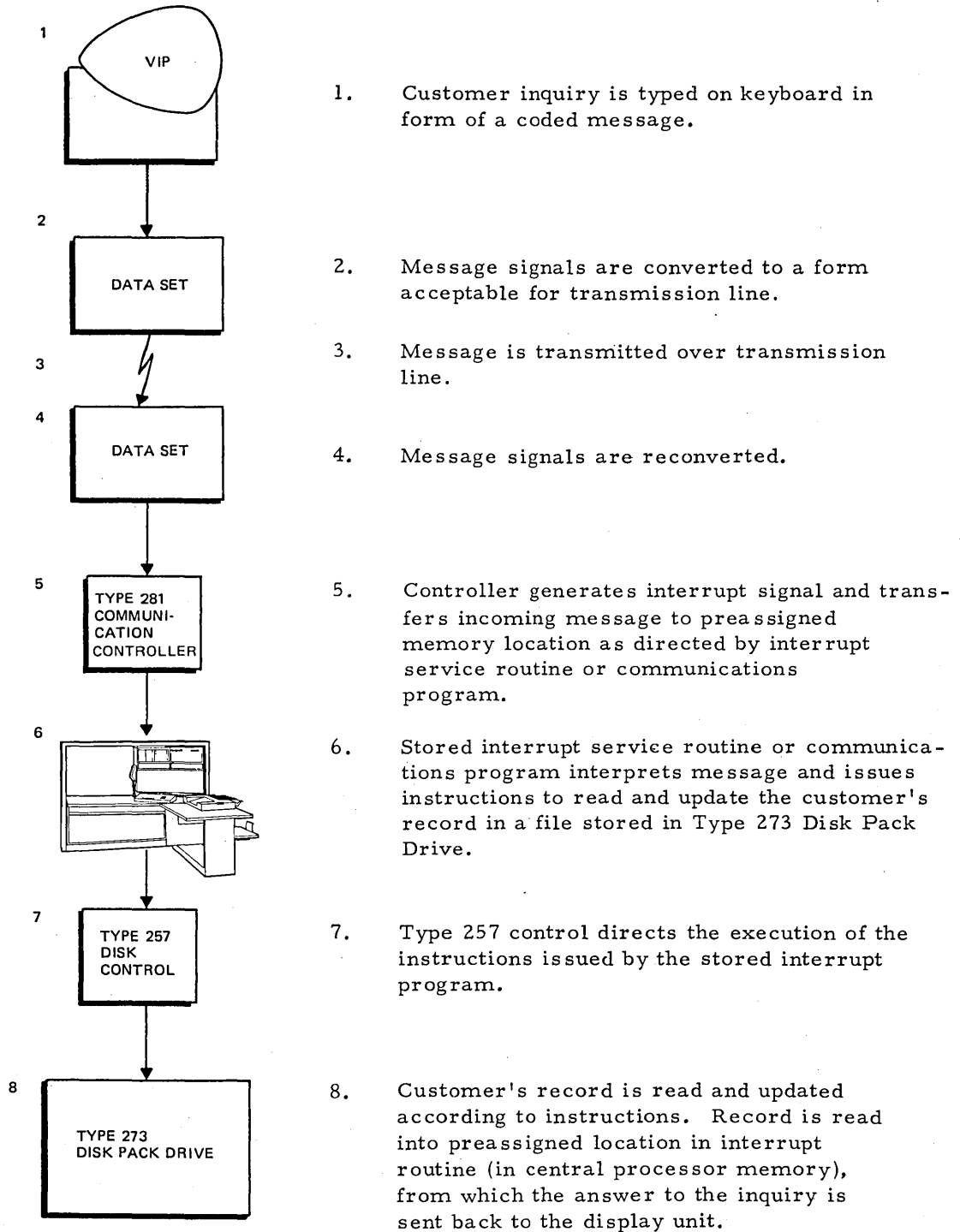


Figure 1-10. Customer Inquiry Handling via Typical Communications Network

## TELLER TERMINAL EQUIPMENT

Honeywell Teller Terminal equipment permits more efficient banking procedures through over-the-counter, on-line processing of all teller-assigned transactions. The Type 7330 and 7340 Teller Terminal is used by the teller for all his bank transactions. The teller terminal transmits transaction information to the computer. Data is transmitted asynchronously via a modified ASCII-type code, permitting combinations of similarly coded terminal devices to share common networks. This code consists of a start bit, seven data bits, an even parity bit, and a stop bit. Specifications of the Type 7330 and 7340 are shown in Table 1-12.

Table 1-12. Teller Terminal Equipment

Device		Data Transfer Rate	Maximum No. Terminals Per Interface Unit	No. Unit Loads Required by Transceiver & Devices	Address Assignments
Type	Function				
7330 & 7340	Teller Terminal	120 characters/record	10	not applicable	not applicable

## FEATURES AND POWER MODULES

The various features that enhance Series 2000 systems are described in the following paragraphs. These features, which were available as options in Series 200 models, are standard in most Series 2000 models.

Power modules for Model 2040A and 2050A systems are described in Tables 1-13 and 1-14, respectively.

### ADVANCED PROGRAMMING

Advanced Programming provides the Series 2000 user with additional program instructions, the ability to modify instruction addresses via indexed or indirect addressing, and a "read reverse" capability with magnetic and paper tape units.

### PROGRAM INTERRUPT

Program Interrupt is standard. A detailed description of program interruption, including conditions that must be present for an interrupt to occur, processor activities that are automatically performed when the interrupt takes place, and the programming of interrupt service routines, is given in Section II.

### EDIT INSTRUCTION

With a comprehensive instruction, Move Characters and Edit, processed information is edited before being converted to an output medium (e. g., a printed document) by the suppression

of unwanted characters and symbols and the insertion of identifying symbols such as the dollar sign, decimal point, and asterisk.

### STORAGE PROTECTION

This feature allows a programmer-specified portion of the main memory (and the contents thereof) to be shielded from accidental alteration by programs running concurrently in the memory. An attempt to violate the protection of this area results in an "internal" processor interruption. The program or programs running in the protected memory area have 15 additional index registers at their disposal; these registers can also be used by programs in the unprotected (or "open") memory area if desired. Storage Protection is described in Section II.

### EXTENDED MULTIPROGRAMMING AND EIGHT-BIT TRANSFER

Processing capabilities are greatly increased by the Extended Multiprogramming and Eight-Bit Transfer feature. In addition to the capabilities supplied by the Storage Protection feature, extended multiprogramming provides storage protection with memory address relocation (or base relocation), interrupt masking, and instruction timeout. The eight-bit transfer capability gives the various models increased flexibility by allowing either eight-bit or six-bit information transfers between certain peripheral controls and main memory. The Extended Multiprogramming and Eight-Bit Transfer capabilities are described in Section II.

### SCIENTIFIC UNIT AND SCIENTIFIC SUBPROCESSOR

The Scientific Subprocessor and Scientific Unit are two functionally-identical units that add 14 floating-point instructions to the Series 2000 repertoire.

The Scientific Subprocessor is standard with Model 2070 processor and is optionally available, as PM3A50, with Model 2050A systems.

The Scientific Unit, as Feature 1100A, is optionally available with Models 2040, 2050, and 2060. As PM3A40, it is optional in the Model 2040A.

The Scientific Subprocessor and Scientific Unit are described in Appendix D.

### HIGH-RESOLUTION CLOCK

The High-Resolution Clock is standard on the Type 2051C, 2051A, 2061, and 2071 processors. A similar feature, the Accounting Timer, is available as PM4A40 with a Type 2041A processor. These features allow elapsed processing time, the time the processor spends in the RUN mode with access to memory, to be carefully measured and maintained.

Under program control, the count can be stored in main memory and printed out on the console. An automatic interrupt is generated if a given program is still accessing memory when the full range of the clock is reached; this interrupt can be used, also under program control, to store the time and reset the timer.

Table 1-13. Model 2040A Power Modules

Power Module	Description
PM1A40	Replaces second I/O sector with a buffered sector
PM1B40	Adds buffered third sector (Power Module PM1A40 must also be present.)
PM2A40	1.5 microseconds memory cycle time (per two characters)
PM2B40	1.0 microsecond memory cycle time (per two characters)
PM3A40	Scientific Unit
PM4A40	Accounting Timer

Table 1-14. Model 2050A Power Modules

Power Module	Description
PM1A50	Replaces second I/O sector with two buffered sectors
PM1B50	Adds buffered fourth and fifth sectors (Power Module PM1A50 must also be present.)
PM2A50	1.5 microseconds memory cycle time (per four characters).
PM2B50	1.0 microsecond memory cycle time (per four characters).
PM3A50	Scientific Subprocessor

### EXPANDED INSTRUCTION PACKAGE

This capability enhances the instruction repertoire of the processor and affords increased compatibility with competitive equipment. This feature provides two additional instructions — Move or Scan (MOS) and Table Lockup (TLU) — and also includes the "S" (Special) mode of processing. The "S" mode of processing, which is implemented by the variant character of the Change Addressing Mode (CAM) instruction, enables the processor to manipulate the Add, Subtract, Zero and Add, Zero and Subtract, and Branch if Character Equal instructions in a special way. All of the above instructions are described in Section VIII of this manual.

SECTION II  
THE CENTRAL PROCESSOR

A Series 2000 central processor is logically divided into five basic units (see Figure 2-1): a main memory, a control memory, an arithmetic unit, a control unit, and an input/output traffic control.

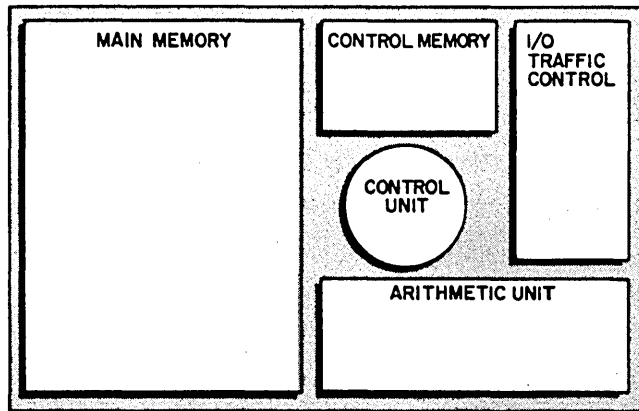


Figure 2-1. Logical Division of Series 2000 Central Processor

MAIN MEMORY

The main memory contains from 49,152 to 1,048,576<sup>1</sup> character locations of magnetic core storage that are used to store program instructions and data during a program run (see Figure 2-2). Every character location is identified by a unique numeric address. This means that an instruction can designate the exact storage locations that contain the data needed for a particular operation.

---

<sup>1</sup>Only access to memory of up to 524,288 characters is described in this manual.



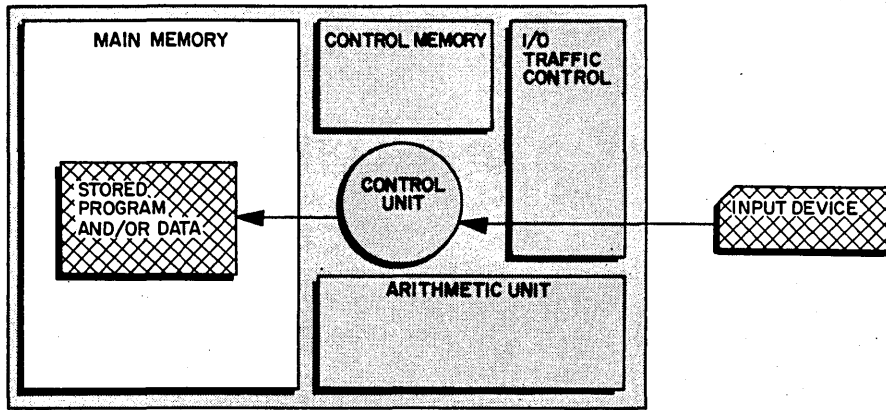


Figure 2-2. Main Memory Functions

Figure 2-3 shows one character position of memory with the name of each core shown to the right. Each core can be individually magnetized to represent either a 1 or a 0, depending upon its polarity. Moving from bottom to top in Figure 2-3, the first six cores are used for data storage, the seventh and eighth cores are used to define the limits of storage areas (these two cores are frequently referred to as "punctuation" bits), and the ninth core is used for parity checking.

CORE	FUNCTION
○	PARITY BIT (P)
○	ITEM-MARK BIT (IM)
○	WORD-MARK BIT (WM)
	} PUNCTUATION BITS
○	B BIT
○	A BIT
	} ZONE BITS
○	8 BIT
○	4 BIT
○	2 BIT
○	1 BIT
	} DATA BITS

Figure 2-3. One Memory Position

		CHARACTER							
		Ø	4	9	B	M	,	(	F
BIT CONFIGURATION	P	○	○	○	○	○	○	○	○
	IM	○	○	○	○	○	○	○	○
	WM	○	○	○	○	○	○	○	○
	B	○	○	○	○	●	●	●	○
	A	○	○	○	●	●	●	●	○
	8	○	○	○	○	○	○	○	○
	4	○	○	○	○	○	○	○	○
	2	○	○	○	○	○	○	○	○
	1	○	○	○	○	○	○	○	○

Figure 2-4. Representation of Characters in Magnetic Core Storage

Figure 2-4 shows how typical numeric, alphabetic, and special characters are stored in the main memory. Shaded circles represent cores containing 1-bits. Bits 1, 2, 4, and 8 in each character position can be combined to represent the decimal values 0 through 9. This four-bit representation of decimal numbers is known as binary-coded decimal (BCD). Alphabetic and special characters are represented by a combination of the numeric (1, 2, 4, and 8) and the A- and B-cores. The A- and B-cores correspond to zone punches on cards: the A-bit represents a 12-punch, the B-bit represents an 11-punch, a combination of the A- and B-bits represents a 0-punch. A listing of the main memory formats for all valid Series 2000 characters appears in Appendix B.

The word-mark bit (WM) is used to define logical storage fields in the memory. Information is rarely stored in the memory as single, independent characters; instead, adjacent character positions are usually grouped to form storage fields. As described in Section III, the wordmark bit is instrumental in defining the size of such fields.

Consecutive storage fields are frequently grouped together to form a unit of information called an item. As its name implies, the item-mark bit (IM) is used to define the size of an item in the main memory (see Section III).

A unit of information that is to be transferred between the main memory and a peripheral device is called a record. A record can be of any length, from one character up to virtually the maximum number of characters in the memory. Both the word-mark and item-mark bits are used in defining the size of a record (see Section III).

The parity bit (P) is used in conjunction with an automatic error-detection technique known as parity checking. Every memory position must be represented in the central processor by an odd number of 1-bits. (Punctuation bits are excluded from this rule except in the multicharacter processors.) Whenever a character is moved from one location to another it is automatically checked to determine if an odd number of data 1-bits have been moved. In Figure 2-4, the characters 0, 9, B, M, and ( are represented by an even number of ones in the data bit positions. Circuitry within the central processor automatically adds a one in the parity bit positions of these characters to provide the required odd bit count.

#### MEMORY CYCLE

The time interval required by a processor to read or write the contents of a unit location is termed memory cycle time. For the processors described in this manual, memory cycle time ranges from 1.6 microseconds per character (Model 2040) down to 1.0 microsecond per four contiguous characters (Model 2070).

## CONTROL MEMORY

The control memory is a high-speed storage unit consisting of up to 64 control registers. (The number of registers actually available depends on the system configuration.) Normally, control registers contain the addresses of instructions and data being processed during a program run. One such register, called the A-address register, is illustrated in Figure 2-5. In this example, the A-address register contains an address (206) designating a main memory location, which in turn contains a unit of information (the decimal digit 7).

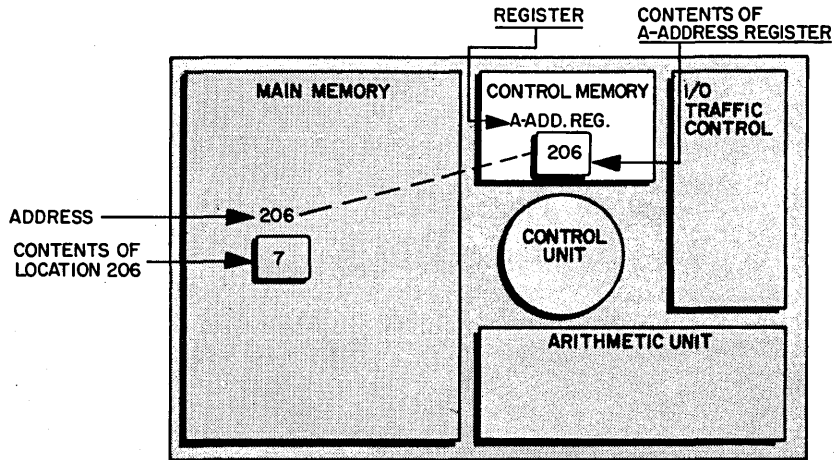


Figure 2-5. Typical Control Register Function

In single-character Series 2000 processors that do not include the Scientific Unit (Feature 1100A), each control register is only as large as it need be to contain the largest, or "highest," main memory address in the user's processor. When the Scientific Unit is included in the system, each control register is 18 bits long. Thus, a processor whose main memory capacity is 49,152 characters contains control memory registers that are each 16 bits long (16 bits allow 65,536 addresses), while the control registers of a processor containing 131,072 characters of memory storage are each 17 bits long (see Table 2-1). In a multi-character processor with up to 524,288 memory locations, 19 control register bits are active.

Table 2-1. Size of Control Memory Registers

MAIN MEMORY CAPACITY (Characters)	49,152	65,536	131,072	262,144	524,288
SIZE OF CONTROL MEMORY REGISTER (Bits)	16	16	17	18	19

Control registers can be addressed either by programmed instruction or from the operator's control panel or console. For instance, an instruction can change the course of a program by manipulating the contents of the control register that governs program sequence; the operator can interrogate a control register to determine the exact location at which the program has halted, etc. When a register is addressed by programmed instruction, it is specified by means of a variant character in the instruction. A register is addressed from the control panel or console by using the register's octal address. The functional name of each register and the variant character which specifies the register are listed in Table 2-2.

### ADDRESS REGISTERS

The A- and B-address registers, the two sequence registers, and the interrupt registers are used to address main memory during the loading and execution of instructions. A detailed description of these registers is presented in Section IV, "Addressing."

### READ/WRITE COUNTERS

Data is transferred between the main memory and a peripheral device via a read/write channel (described in Section I). Two location counters are associated with a read/write channel: a starting location counter and a current location counter. When a peripheral transfer is to be performed, the address at which the transfer is to begin is stored in both counters. Then, as each successive character is transferred, the contents of the current location counter are incremented (or decremented) by one so that when the transfer is completed, this counter contains the address of the character position immediately following (or preceding) the position of the last character transferred. In transfers rated at less than 167 KC, the current location counter is one beyond the record-marked location. For higher transfer rates, consult the appropriate peripheral reference manual.

The availability of starting and current addresses associated with an input/output area greatly simplifies the manipulation of variable-length records.

Table 2-2. Control Memory Registers

Mnemonic Designation	Function	Variant Character
Registers Standard in All Processors		
AAR	A-Address Register	67
BAR	B-Address Register	70
SR	Sequence Register	77
CLC1	Read/Write Channel 1 - Current Location Counter	01
CLC2	Read/Write Channel 2 - Current Location Counter	02
CLC3	Read/Write Channel 3 - Current Location Counter	03
SLC1	Read/Write Channel 1 - Starting Location Counter	11
SLC2	Read/Write Channel 2 - Starting Location Counter	12
SLC3	Read/Write Channel 3 - Starting Location Counter	13
WR1	Work Register 1 <sup>a</sup>	--
WR2	Work Register 2 <sup>a</sup>	--
WR3	Work Register 3 <sup>a</sup>	--
CSR	Change Sequence Register	64
EIR	External Interrupt Register	66
IIR	Internal Interrupt Register	76
CLC1'	Read/Write Channel 1' - Current Location Counter	05
SLC1'	Read/Write Channel 1' - Starting Location Counter	15
CLC4	Read/Write Channel 4 - Current Location Counter	21
CLC5	Read/Write Channel 5 - Current Location Counter	22
CLC6	Read/Write Channel 6 - Current Location Counter	23
CLC4'	Read/Write Channel 4' - Current Location Counter	25
SLC4	Read/Write Channel 4 - Starting Location Counter	31
SLC5	Read/Write Channel 5 - Starting Location Counter	32
SLC6	Read/Write Channel 6 - Starting Location Counter	33
SLC4'	Read/Write Channel 4' - Starting Location Counter	35
CLC5'	Read/Write Channel 5' - Current Location Counter	26
CLC6'	Read/Write Channel 6' - Current Location Counter	27
SLC5'	Read/Write Channel 5' - Starting Location Counter	36
SLC6'	Read/Write Channel 6' - Starting Location Counter	37
CLC8	Read/Write Channel 8 - Current Location Counter	00
CLC9	Read/Write Channel 9 - Current Location Counter	20
SLC8	Read/Write Channel 8 - Starting Location Counter	10
SLC9	Read/Write Channel 9 - Starting Location Counter	30
CLC8'	Read/Write Channel 8' - Current Location Counter	04
CLC9'	Read/Write Channel 9' - Current Location Counter	24
SLC8'	Read/Write Channel 8' - Starting Location Counter	14
SLC9'	Read/Write Channel 9' - Starting Location Counter	34

Table 2-2 (cont). Control Memory Registers

Mnemonic Designation	Function	Variant Character
Registers Standard in All Processors		
CLC2'	Read/Write Channel 2' - Current Location Counter	06
CLC3'	Read/Write Channel 3' - Current Location Counter	07
SLC2'	Read/Write Channel 2' - Starting Location Counter	16
SLC3'	Read/Write Channel 3' - Starting Location Counter	17
Scientific Unit/Scientific Subprocessor		
AC0	Floating-Point Accumulator 0 <sup>b</sup>	-- -- --
AC1	Floating-Point Accumulator 1 <sup>b</sup>	-- -- --
AC2	Floating-Point Accumulator 2 <sup>b</sup>	-- -- --
AC3	Floating-Point Accumulator 3 <sup>b</sup>	-- -- --
Registers on Multicharacter Processors Only		
WR4	Work Register 4 <sup>a</sup>	--
WR5	Work Register 5 <sup>a</sup>	--
WR6	Work Register 6 <sup>a</sup>	--
WR7	Work Register 7 <sup>a</sup>	--
ATR	Accounting Timer Register	54
<p><sup>a</sup>These registers are available only to the processor and must not be addressed by the program.</p> <p><sup>b</sup>These registers (accumulators) can be addressed only by the instructions included in the scientific unit or subprocessor (see Appendix D).</p>		

## ARITHMETIC UNIT

Arithmetic and logical operations are performed by a configuration of components commonly referred to as the arithmetic unit. Basically, this unit is composed of an adder, capable of performing both binary and decimal arithmetic, and two operand storage registers.<sup>1</sup> In single-character processors, each component is capable of storing a single six-bit character. The adder and operand storage registers in the multicharacter processors can store two or four characters at a time. In general terms, an arithmetic or logic operation is performed as follows (see Figure 2-6):

1. An instruction in the stored program specifies the type of operation to be performed and the main memory storage locations of the data to be manipulated.
2. The operands are transferred to the operand storage registers a character or a word at a time, beginning with the rightmost character in each operand.
3. Each pair of characters (or, in the multicharacter processors, each pair of multicharacters) that enters the storage registers is combined in the adder. The result is stored in the main memory as specified by the program instruction. If a carry is generated, it is stored in the adder and combined with the next higher-order pair of characters (or multicharacters).

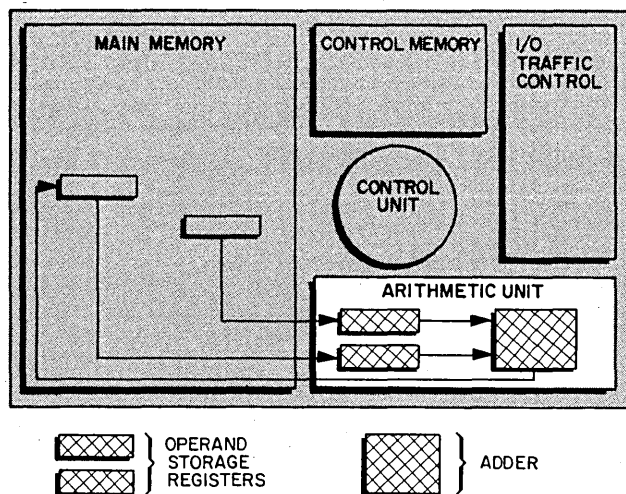


Figure 2-6. Data Flow Between Main Memory and Arithmetic Unit

<sup>1</sup>The contents of these registers are not accessible to the programmer.

## CONTROL UNIT

The control unit is the hub of central processor activities (see Figure 2-7). Its major function is to select, interpret, and execute all of the instructions in the stored program. In carrying out these instructions, the control unit coordinates the various activities of receiving data from input devices, transferring data within the central processor, and transferring processed data to the output units. The main memory addresses used by the control unit in performing these tasks are stored in the registers of the control memory.

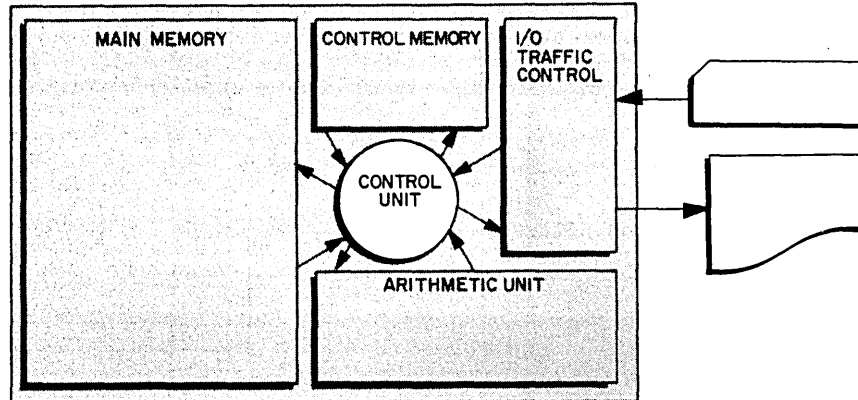


Figure 2-7. Control Unit Activities

### INPUT/OUTPUT TRAFFIC CONTROL

The input/output traffic control is, as its name implies, the unit which regulates the flow (or "traffic") of data transferred during input/output activities. It works in conjunction with the central processor control unit to allocate central processor time to input/output operations and to identify the peripheral controls which are to use that time to transfer data (see Figure 2-8).

The I/O traffic control enables from 8 to 16 simultaneous input/output operations (depending on the processor type) to occur concurrently with the internal computations of the processor. This simultaneity is achieved by the traffic control's allocation of consecutive memory access offerings to either peripheral controls or the central processor.

### DATA TRANSFER RATES

With single-character processors, and with the unbuffered sectors of multicharacter processors, one character can be transferred to or from main memory with each access to memory. The data transfer rate of a read/write channel or time slot is therefore totally dependent upon the number of memory accesses it is granted. For example, if a read/write channel or time slot is granted access to main memory 83,000 times per second (once every 12 microseconds) the data transfer rate is 83,000 characters-per-second.



With the buffered sectors of multicharacter processors, however, it is possible to transfer two or four characters to or from memory with each memory access. Buffered sectors provide temporary storage for characters being transferred between main memory and a peripheral control, thus adapting the slower single-character operation of the peripheral control to the faster multicharacter operation of the processor. Because, with buffered sectors, more than one character is transferred at a time, a given transfer rate can be sustained with fewer accesses to memory. The data transfer rate is now dependent on the number of characters transferred simultaneously, as well as on the number of memory accesses granted. For example, on the buffered sector of a two-character processor, a data transfer rate of 83,000 characters-per-second can be sustained with only 41,500 memory accesses per second (one access each 24 microseconds), since two characters are transferred each time.

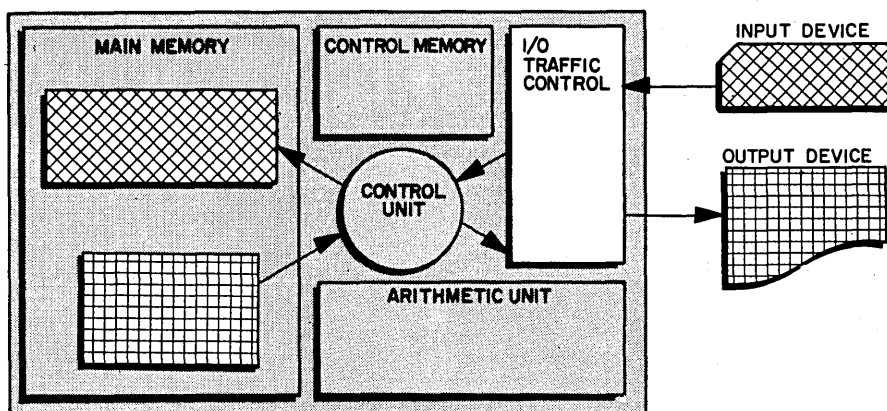


Figure 2-8. Input/Output Traffic Control Activities

### MEMORY ACCESS DISTRIBUTION

Every peripheral data transfer involves some factor that prevents the device being used from transferring data at a rate comparable to that of the central processor. Usually this factor is mechanical — moving a card through the read station or a magnetic tape past the read/write head — although in data communication it is the bit rate of the communication line. Therefore, a peripheral device requires access to the central processor to transfer information to or from the main memory during only a fraction of the time that the operation is proceeding. The periods in which the central processor is actually interrupted for data transfer are spaced over the duration of the peripheral operation (see Figure 2-9).

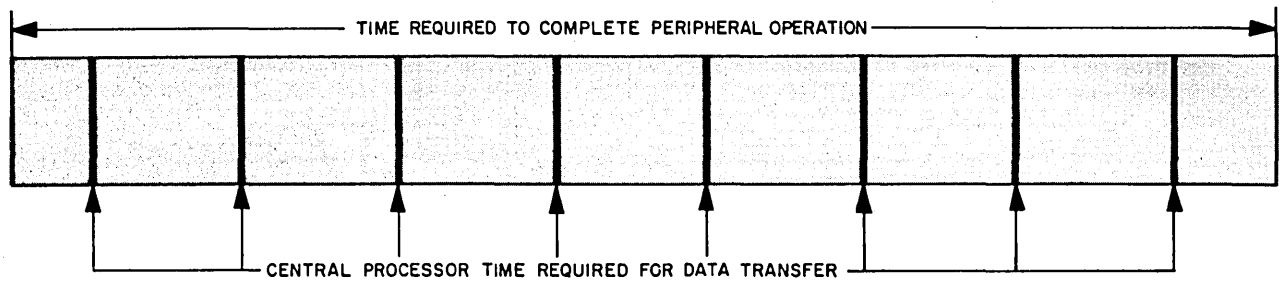


Figure 2-9. Data Transfer Intervals During One Peripheral Operation

When a peripheral operation is in progress but is not using main memory (the gray areas in Figure 2-9), another peripheral control may gain access to the main memory. This second memory access can in turn give way to a third access by another control before the original operation requires access to the memory again, etc. In other words, peripheral operations can occur simultaneously with one another. The periods of time peripheral controls do not require main memory access to transfer data are given to the central processor for its internal activities. It is the function of the I/O traffic control to direct the sharing of main memory access by the various peripheral devices and the central processor.

As described in Section I, in order for an I/O operation to proceed, the programmer or operating system must specify a read/write channel in initiating the peripheral instruction. This read/write channel completes the path between main memory and the control for the peripheral device being addressed.

The rate at which each peripheral control must transfer data over a programmer-assigned read/write channel(s) depends on the mechanical characteristics of the device connected to the control. Thus, the transfer intervals shown in Figure 2-9 are specified according to the device being used. For instance, the transfer rate for the disk pack drive is considerably faster than that for the card punch; therefore, the disk pack drive will require access to the main memory more frequently than the card punch. The I/O traffic control monitors and honors the requests for access to the main memory. The I/O traffic control decides how the memory access offerings should be used — by a read/write channel or by the processor — as shown in Figure 2-10.

The traffic control offers consecutive memory access offerings to read/write channels, one memory access offering per channel. If there is a demand on a particular channel when the memory access is offered, the channel is granted one access to the main memory. During this cycle, one, two, or four characters are transferred to or from memory, depending on the processor type, sector, and transfer mode. If the channel does not require access to memory when it is offered, the cycle is given to the central processor for internal data processing.

A sector always has six time slots. In single-character processing, each read/write channel is permanently associated with a particular time slot. With a multicharacter processor, read/write channels have no permanent time slot assignments; when an I/O instruction is issued, an available time slot is associated with the RWC specified. Memory accesses that are not offered to the I/O traffic control are given unconditionally to the central processor.

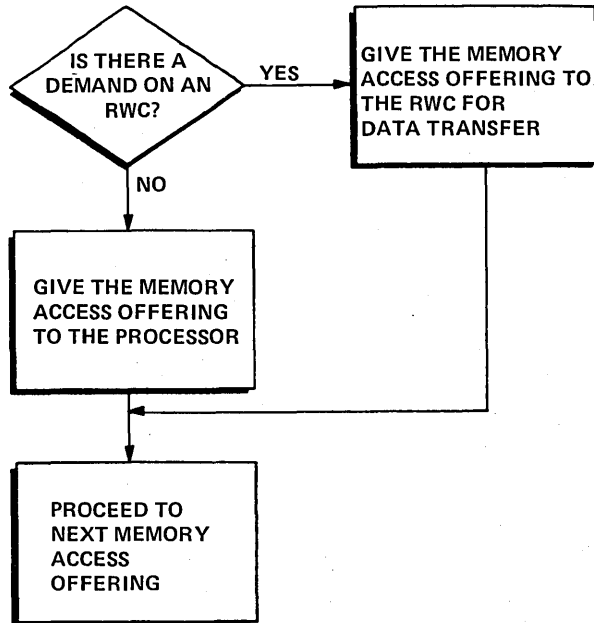


Figure 2-10. Logical Decision Performed by Input/Output Traffic Control

Memory Access Distribution of the Type 2041 Processor

In the Type 2041 processor, access to memory is offered for input/output operations each 1.5 microseconds. The eight read/write channels (four in sector 1 and four in sector 2) are each offered access to memory once every 12 microseconds as shown in Figure 2-11.

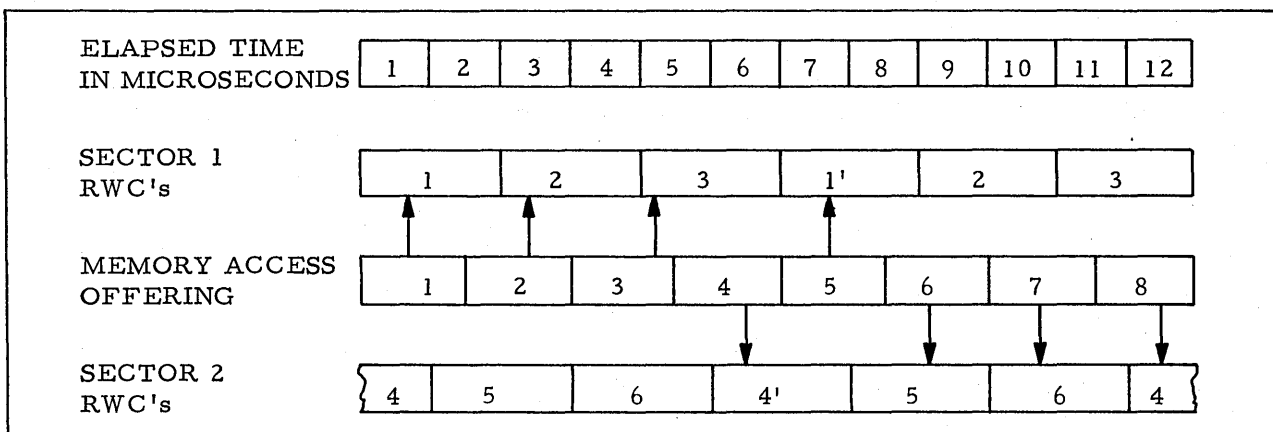


Figure 2-11. Memory Access Distribution in the Type 2041 Processor

When a read/write channel is offered access to memory, it may transfer a single character to or from main memory. This establishes the basic transfer rate of a read/write channel at 83,000 characters-per-second because the sequential offering of access to memory is made 83,000 times each second.

Memory Access Distribution of the Type 2041A Processor

In the basic Type 2041A processor (without input/output power modules), access to memory is alternately offered to sector 1 and sector 2, as shown in Figure 2-12. In a 12-microsecond period read/write channels 2, 3, 5, and 6 are each offered access to memory twice; thus each of these read/write channels has a transfer rate of 167,000 characters per second. Read/write channels 1, 1', 4, and 4' are each offered access to memory once in each 12-microsecond period; thus each has a transfer rate of 83,000 characters per second.

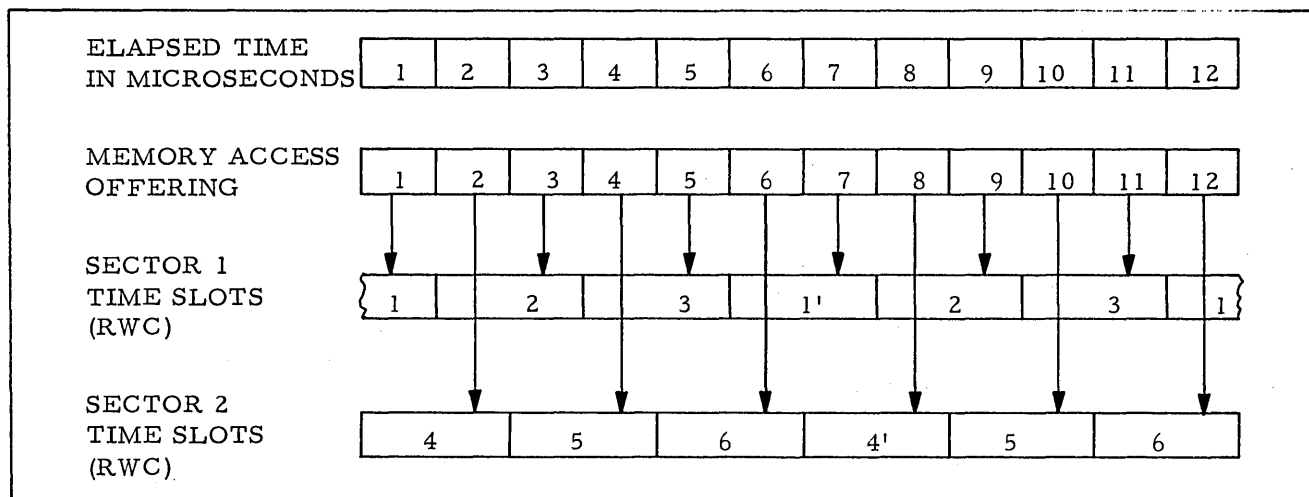


Figure 2-12. Memory Access Distribution in the Basic Type 2041A Processor

The memory access distribution of the Type 2041A processor with PM1A40 is identical to the Type 2051C, and the memory access distribution of the Type 2041A processor with both PM1A40 and PM1B40 is identical to the Type 2061.

Memory Access Distribution of the Type 2051C Processor

In the Type 2051C processor (and the Type 2041A processor with PM1A40), sector 1 (unbuffered) is offered alternate memory access offerings as shown in Figure 2-13. In a 12-microsecond period each sector 1 time slot receives one access to memory and may transfer a single character; thus each sector 1 time slot sustains a data transfer rate of 83,000 characters per second.

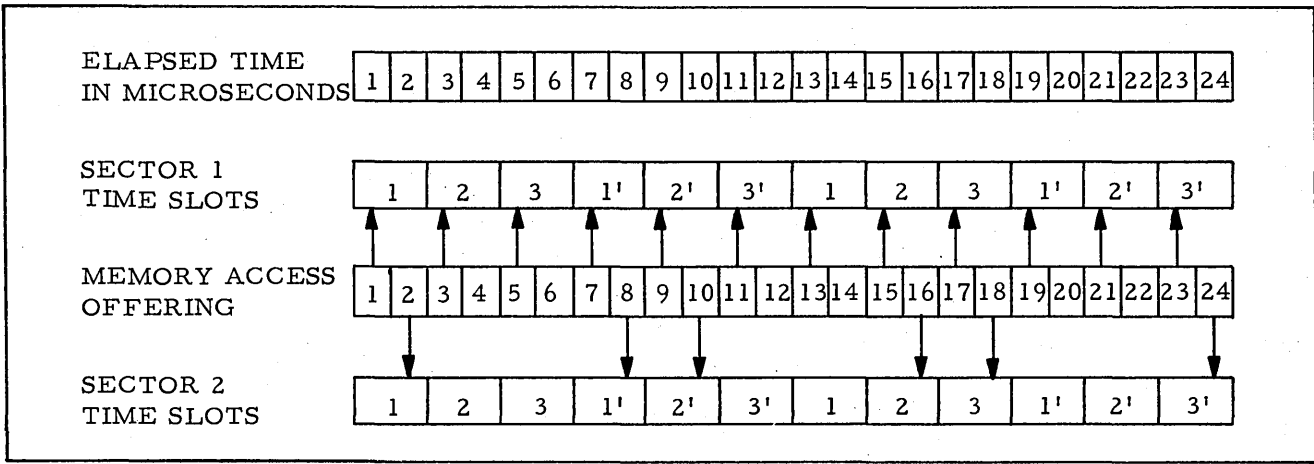


Figure 2-13. Memory Access Distribution in the Type 2051C Processor and Type 2041A Processor with PM1A40.

The time slots of sector 2 (buffered) can each transfer two characters with each access to memory. Each sector 2 time slot gains access to memory once every 24 microseconds; thus each is capable of a data transfer rate of 83,000 characters per second.

Memory Access Distribution of the Type 2051A Processor

In the basic Type 2051A processor (without PM1A50 and/or PM1B50), sector 1 (unbuffered) is offered alternate memory access offerings as shown in Figure 2-14. In a 12-microsecond period, each sector 1 time slot gains one memory access offering; thus each sector 1 time slot sustains a data transfer rate of 83,000 characters per second.

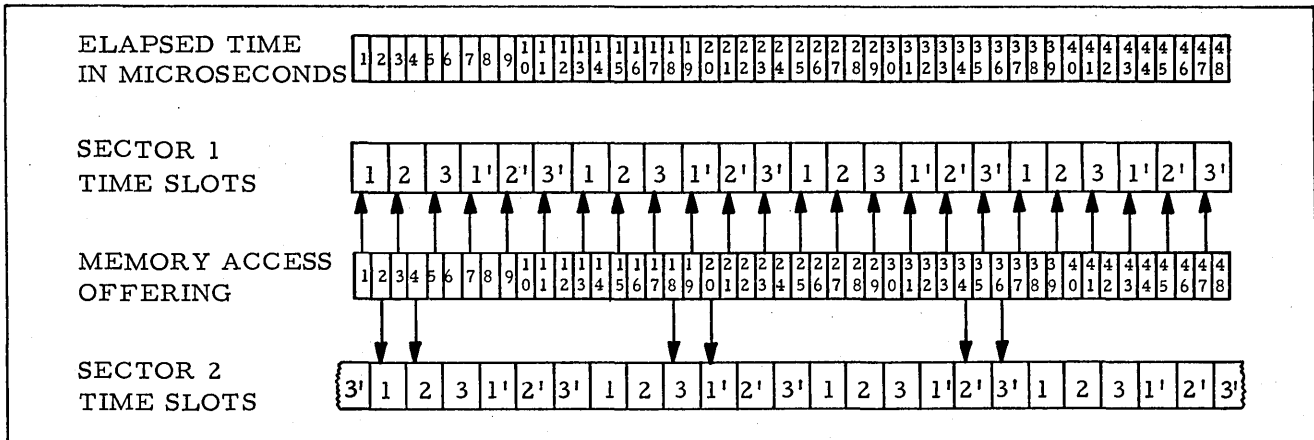


Figure 2-14. Memory Access Distribution in the Basic Type 2051A Processor

The time slots of sector 2 (buffered) can each transfer four characters with each access to memory. As they gain one access to memory every 48 microseconds, each sector 2 time slot has a data transfer rate of 83,000 characters per second.



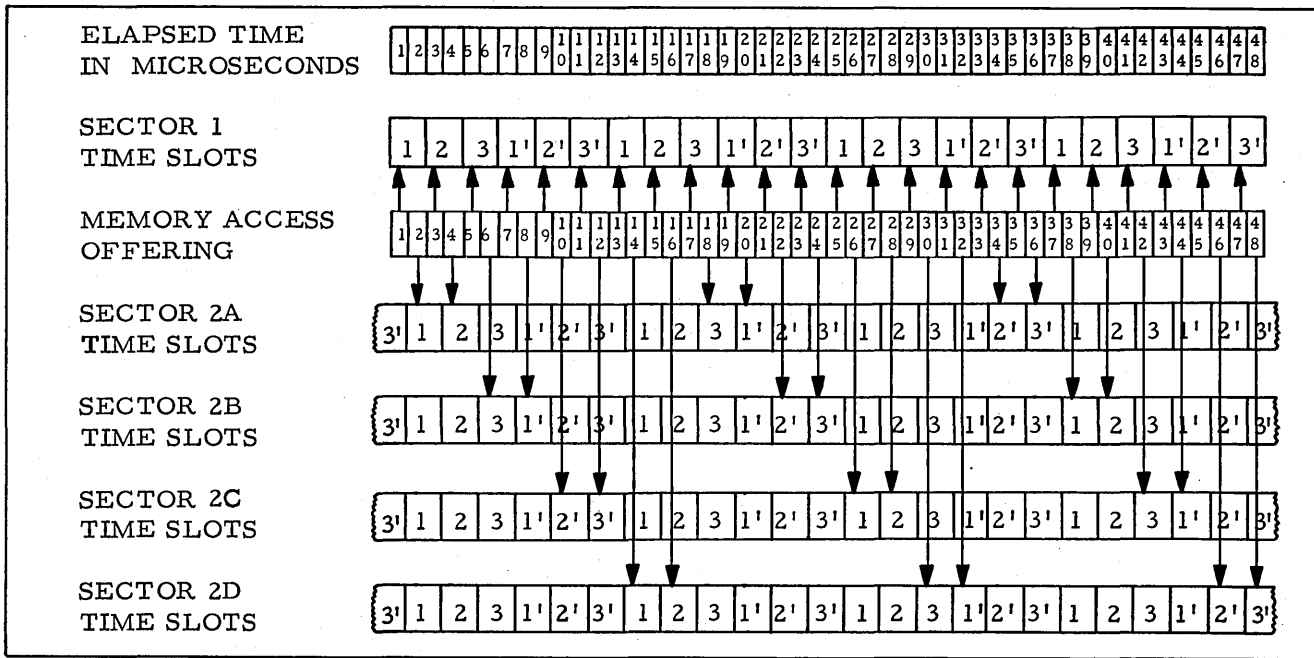


Figure 2-16. Memory Access Distribution in the Type 2071 Processor and Type 2051A Processor with PM1A50 and PM1B50

Memory Access Distribution of the Type 2061 Processor

In the Type 2061 processor (and the Type 2041A processor with both PM1A40 and PM1B40), sector 1 (unbuffered) is offered alternate memory access offerings as shown in Figure 2-17. In a 12-microsecond period, each sector 1 time slot gains one access to memory; thus each sector 1 time slot sustains a data transfer rate of 83,000 characters per second.

The time slots of sectors 2A and 2D (both buffered) can each transfer two characters with each access to memory. Each time slot in sectors 2A and 2D gains access to memory once every 24 microseconds; thus each time slot in sectors 2A and 2D also has a data transfer capability of 83,000 characters per second.

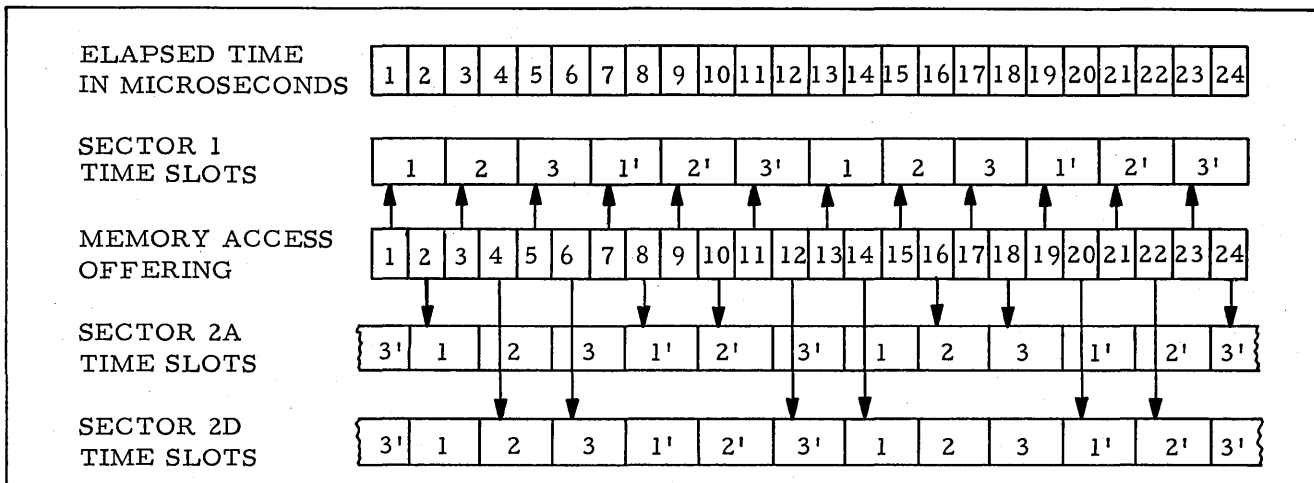


Figure 2-17. Memory Access Distribution in the Type 2061 Processor and Type 2041A Processor with PM1A40 and PM1B40

## Memory Access Distribution of the Type 2071 Processor

In the Type 2071 processor, sector 1 (unbuffered) receives alternate memory access as shown in Figure 2-16. In a 12-microsecond period each sector 1 time slot receives one access to memory and may transfer a single character; thus each sector 1 time slot has a data transfer rate of 83,000 characters per second.

Sectors 2A, 2B, 2C, and 2D (all buffered) divide the remaining accesses to memory. Each time slot in these sectors is capable of transferring four characters at a time, and each receives access to memory once each 48 microseconds. Therefore, each time slot has a data transfer rate of 83,000 characters per second.

## INTERLOCKING READ/WRITE CHANNELS

In order to achieve data transfer rates in single-character processors higher than those attainable with a single read/write channel, it is necessary to interlock two or more read/write channels. In this manner, data transfer rates from 167,000 to 500,000 characters per second are attainable. The same instruction that initiates the data transfer operation specifies whether channels are to be interlocked. When this procedure is used, all of the cycles normally offered to the interlocked channels are made available to the single data transfer operation. The transfer rate thus provided is equal to the sum of the rates attainable individually with the interlocked channels. When the operation is completed, memory cycle allocation returns to normal and channels are again offered cycles at the normal intervals. Programming procedures for channel interlocking are described in Section VIII.

## VARIABLE-SPEED READ/WRITE CHANNELS

In multicharacter processors, transfer rates higher than those attainable with a single time slot are attained by associating two or more time slots with a single read/write channel ("time slots" are the memory cycles offered to a given sector).<sup>1</sup> An I/O instruction specifies the read/write channel that is to be used and the transfer rate required; the I/O traffic control then assigns to the read/write channel the appropriate number of time slots. In this manner, in a typical I/O sector, transfer rates from 167,000 characters per second (using two time slots) to 500,000 characters per second (using all six time slots in the sector) can be attained without affecting the other read/write channels.

---

<sup>1</sup>In certain cases, both a primary channel and a corresponding auxiliary channel are made busy. However, no more than two RWCs are ever made busy by a single I/O instruction on a multicharacter processor.



The most significant advantage of the variable-speed read/write channel capability is that the read/write channels not made busy by a high-speed transfer are available for other peripheral operations. In comparison, in a single-character processor, a 250,000-character-per-second transfer would require the interlocking of several channels; on a multicharacter processor, one primary channel would be used. Other read/write channels would be available for use in other operations; e.g., three 83,000-character-per-second transfers. Note that the transfer rate of a single sector cannot exceed 500,000 characters per second.

## BUFFERED SECTORS

In multicharacter processors replacing a standard second sector with two or four buffered sectors provides additional computation time and a higher I/O transfer capability. This is achieved through the use of intermediate storage buffers for I/O transfer operations. Accumulation of characters in these buffers permits multiple-character transfers into and out of main memory; multiple-character transfers maintain a given transfer rate with decreased frequency of access to main memory.

### Buffered Sector Operation

Data transfer operations on a buffered sector may be executed in either the buffered mode or the direct-access mode. Both modes may be used simultaneously on the same sector.

#### BUFFERED MODE

In the buffered mode, a two-character or four-character buffer is associated with each time slot in the sector. Transfer of data from peripheral control units to these buffers is in the normal, single-character manner. However, since the buffer associated with the active time slot is allowed to accumulate more than one character before access to main memory is required, data transfer from the buffer to main memory is on a multiple-character basis.

As stated previously, this multiple-character transfer capability compensates for the less frequent memory access of buffered sectors, such that, on multicharacter processors, the transfer rate of each time slot is the same as that of time slots in unbuffered sectors (83,333 characters per second). Direct-access mode, described below, is an exception.

## DIRECT-ACCESS MODE

It may be desired to use the time slots in a buffered sector for single-character transfers, just as if the sector were not buffered. This mode of operation is called the direct-access mode, and is used to accommodate peripheral controls or devices that cannot operate in the buffered mode. However, this mode of operation results in a lower effective transfer rate. Since buffered sectors are sharing the memory cycles normally allocated to sector 2, and hence receiving less frequent access to memory, data transfer rates in direct-access mode will be slower in relation to the number of buffered sectors. Therefore, each time slot (and consequently each read/write channel combination) will handle one-half its specified rate on the 2-character processors and one-quarter its specified rate on the 4-character processors (see Table 8-24). Furthermore, interlocking more than two time slots is illegal in this mode.

In view of the foregoing, serious consideration should be given to connecting a peripheral control or device to Sector 1 rather than using direct-access mode on a buffered sector.

### Buffered Sector Restrictions

The use of multiple-character transfers in the buffered mode prohibits the use of certain peripheral controls and devices. The lower transfer rate of buffered sector time slots in the direct-access mode prohibits the use of certain control units and devices in this mode. Table 2-3 indicates which controls/devices can be operated on a buffered sector in the buffered and direct-access modes.

### Programming Considerations

#### EXTENDED I/O INDICATOR

The Extended I/O Indicator is loaded and stored by RVI and SVI instructions, respectively (see Section VIII). This indicator is turned OFF by the INITIALIZE button.

#### TESTING PERIPHERAL CONTROL UNIT BUSY STATUS

Since buffered-mode operation involves the intermediate storage of data characters between the peripheral control unit and main memory, it is highly improbable that a data transfer operation can be completed before the peripheral control becomes "not busy." Therefore, the testing of the peripheral control alone or the receipt of a peripheral control interrupt is not sufficient to guarantee completion of data transfer; the specific read/write channel used in the transfer operation must always be tested. In other words, (1) all PCB instructions used to test for the completion of a peripheral data transfer operation should include the specific RWC designation (control character C1), and (2) all external interrupts should be immediately followed by a PCB instruction that includes the specific RWC designation.

## ESCAPE CODES

When buffered sectors are in use, escape codes must be used to designate any of the sectors 2A, 2B, 2C, 2D (i.e., this cannot be accomplished using the sector bits of control character C2). The escape codes provided for this purpose are listed in Table 8-25. Note that these escape codes not only designate the sector to which a read/write channel is to be assigned, but also indicate whether the I/O transfer operation is to be in direct-access or buffered mode.

NOTE: On processors with two or more buffered sectors, a control character that references sector 2 will be interpreted as sector 2A, buffered mode.

Table 2-3. Controls/Devices Connectable to Buffered Sectors

Peripheral Control/Device	Buffered Mode	Direct Mode
Type 203B-1 Tape Controls	Yes	No
Type 203B-2 Tape Control	Yes	No
Type 203B-4 Tape Control	Yes	No
Type 203B-6 Tape Control	Yes	No
Type 203D-1 Tape Control	Yes	No
Type 203D-3 Tape Control	Yes	No
Type 203D-5 Tape Control	Yes	No
Type 203F1 Tape Control	Yes	No
Type 203F3 Tape Control	Yes	No
Type 203F5 Tape Control	Yes	No
Type 209 Paper Tape Reader	No	Yes
Type 209-2 Paper Tape Reader	No	Yes
Type 210 Paper Tape Punch	Yes	Yes
Type 212 On-Line Adapter	No	Yes
Type 212-1 Memory-to-Memory Adapter	No	Yes
Type 212-2 Central Processor Memory-to-Memory Transfer Unit	No	Yes
Type 213-3 Interval Timer with Interval Selector	Yes	No
Type 213-4 Time-of-Day Clock	Yes	No
Type 208-1 Card Reader-Reader/Punch Control	Yes	Yes
Type 220-6 Console	No	Yes
Type 220-8 Visual Information Control Console	No	Yes
Type 222-3 Printer	Yes	No
Type 223N Printer	Yes	No
Type 222-4 Printer	Yes	No
Type 222-6 Printer (Feature 036 required)	Yes	No
Type 222-7 Printer	Yes	No
Type 223 Card Reader	Yes	No
Type 223-2 Card Reader	Yes	No
Type 232 MICR Reader-Sorter	Yes	No
Type 233-2 MICR Reader-Sorter Control for B103	No	Yes
Type 236-1 High-Speed Document Reader-Sorter Control	No	Yes
Type 257 Disk Pack Drive Control	No	Yes
Type 257-1 Disk Pack Drive Control	Yes	Yes
Type 257-3 Disk Pack Drive Control	Yes	No
Type 257B-1 Disk Pack Drive Control	No	Yes
Type 260 Disk Pack Drive Control	Yes	No
Type 260-1 Disk File Control	Yes	No
Type 274 Disk Pack Drive	Yes	No
Type 275-2 Disk Storage Subsystem	Yes	No
Type 277-2 Disk Storage Subsystem	Yes	No
Type 279-2 Disk Storage Subsystem	Yes	No
Type 281-1, -2 Single-Line Communication Controllers	No	Yes

## STORAGE PROTECTION FEATURE

The Storage Protection feature allows the main memory to be logically divided into two distinct areas: a protected area and an unprotected (or "open") area. When storage protection is in effect, the contents of the protected area are shielded from unintentional interference by any program operating in the standard (noninterrupt) mode (whether residing in the protected or unprotected area). The protected area is specified as follows:

1. The programmer sets the lower boundary of the area with a Load Index/Barricade Register (LIB) instruction specifying the number of a 4,096 - character memory bank. The LIB instruction places this number in the index/barricade register. The lower boundary of the protected area is the leftmost (lowest) core storage location within this bank.
2. The upper boundary of the protected area is always the highest location in main memory.

The loading of the index/barricade register merely sets the low-order boundary of the protected area. In order to put storage protection into effect, the following must be present:

1. The programmer must have turned the protect indicator on by issuing a Restore Variant and Indicators (RVI) instruction specifying the protect indicator.
2. The processor must be in the standard (noninterrupt) mode.

### INDEX REGISTERS

The Storage Protect feature provides the user with an additional 15 index registers (Y1 through Y15), which are located in the leftmost 60 locations of the 4,096-character bank specified by the current contents of the index/barricade register. Thus, these index registers are relocated whenever the contents of the index/barricade register are altered by an LIB instruction. These 15 registers are usable whenever the index/barricade register is loaded with a proper bank number and are not dependent upon whether storage protection is in effect. Instructions whose address portions are indexed by these registers must be assembled and executed in the four-character addressing mode. The high-order bit of the five-bit address modifier in a four-character address distinguishes index registers X1 through X15 from Y1 through Y15.

### CENTRAL PROCESSOR MODES

The central processor can operate in any one of three modes:

1. The standard mode,
2. The external interrupt mode, or
3. The internal interrupt mode.

## Internal Interrupt

When storage protection is in effect (i.e., the protect indicator is on and the processor is operating in the standard mode), certain operations are defined as violations of that protection. These violations are discussed below. A violation causes a violation indicator to be set which, in turn, causes an internal interrupt to occur at the next opportunity. The "next opportunity" means that moment when all of the following conditions are present:

1. The processor is in the RUN mode (i.e., automatically executing stored-program instructions under the control of the sequence register),
2. The processor is about to extract an op code,
3. A memory cycle is allocated to the processor,
4. The processor is in the standard mode (i.e., not in external or internal interrupt mode), and
5. No peripheral or control panel interrupt signal is being received.

When an internal interrupt occurs, the contents of the sequence register and the internal interrupt register are interchanged and the central processor enters the internal interrupt mode. The status of the processor indicators are not stored automatically; therefore, the programmer must perform this function with a Store Variant and Indicators (SVI) instruction. The SVI instruction also clears the violation indicator so that an internal interrupt will not occur when a return is made to the standard mode. While in the internal interrupt mode, any external interrupt will cause the processor to switch to the external interrupt mode.

If an external interrupt occurs while the processor is in the internal interrupt mode, the 1-bit of the character stored by V5 of the SVI instruction indicates the condition. If it is desired to revert to the standard rather than the internal interrupt mode after servicing the external interrupt, this bit should be changed to 0 before executing the RVI instruction.

Note that three basic differences exist between the external interrupt mode and the internal interrupt mode:

1. A unique control memory location, the internal interrupt register (IIR), contains the address of the subroutine which services the internal interrupt,
2. The processor is subject to being interrupted by an external interrupt while still in the internal interrupt mode, but the reverse is not true,
3. No processor indicators are stored or altered (the address mode is not changed) upon entering the internal interrupt mode.

## VIOLATIONS OF STORAGE PROTECTION

The following operations, which constitute violations of storage protection, fall into two general categories: address violations and op code violations..

1. An attempt to transfer information internally (i.e., not via a PDT instruction) to memory locations within the protected area. This includes any attempt to modify index registers Y1 through Y15. However, no violation occurs when information is transferred internally from the protected area (including index registers Y1 through Y15). An internal transfer violation is detected when all of the following conditions are present:
  - a. The bank and sector bits in the A- or B-address register following instruction extraction are equal to or greater than the corresponding bits stored in the index/barricade register.
  - b. A protected location is addressed as a result location,
  - c. The protect indicator is on,
  - d. The program in control is operating in the standard mode, and
  - e. The instruction is not a PDT.

The above conditions are checked as the instruction is being executed. If all of these conditions are met, the internal interrupt address violation indicator is set, and the instruction proceeds to normal completion except that no information is transferred into memory (i.e., the write cycle is inhibited). The next opportunity for the internal interrupt to occur is at the extraction of the next op code. After the internal interrupt mode is entered, the internal interrupt register contains the address of the op code following the instruction which caused the violation, and the A- and B-address registers continue to increment or decrement, as appropriate.

2. An attempt to extract a PDT instruction (input or output) whose effective A-address references a protected memory location. Since the PDT instruction is one of the operations normally prohibited when storage protection is in effect (see 4., below), the proceed indicator must be set in order for the instruction to be extracted beyond the op code. Assuming that the proceed indicator is set, the starting address of the PDT operation is examined for address violation. Once it is determined that the effective A-address references a protected address, no operation is performed (i.e., the specified read/write channel is not tested and the specified peripheral control is not addressed), the internal interrupt address violation indicator is set, the sequence register is advanced to the next op code, and an internal interrupt occurs.

Note that a PDT instruction is checked for possible violation during the extraction phase, while a nonperipheral instruction is checked during its execution phase (see 1., above). If a PDT instruction passes this test during extraction, it is free to be executed and thereby cause data to be transferred. If the information being transferred extends into the protected area, no address violation is detected. To insure that this will not occur, the user must set a record mark immediately prior to the protected area.<sup>1</sup>

As mentioned previously, storage protection (and the checking functions related to it) are in effect only when the processor is operating in the standard mode. However, violations of the protected area by PDT instructions executed in either of the two interrupt modes can be detected if the proceed indicator is set on.

---

<sup>1</sup>If communication devices are being used, two consecutive locations should contain record marks.

3. An attempt to read from a main memory location whose address is greater than the main memory capacity actually present in the machine but within the addressing capacity of the memory address register (MAR).<sup>1</sup> Such an addressing attempt results in a parity error which normally causes the machine to halt. If storage protection is in effect and a parity error occurs, a check is made to determine whether the error occurred above the lower boundary of the protected area. If so, the storage protection hardware assumes that out-of-range addressing has been attempted,<sup>2</sup> no halt occurs, nor is data transferred; instead, the internal interrupt address violation indicator is set, instruction execution is prematurely terminated, and an internal interrupt occurs.

An attempt to reference an address greater than the addressing capacity of the memory address register results in a memory "wraparound." (see page 4-15).

4. An attempt to execute a privileged op code. A privileged op code is one that is (a) not defined for the Series 2000; (b) not recognized on the particular processor; (c) an instruction format violation in any floating-point instruction; or (d) prohibited when storage protection is in effect. The privileged op codes in category (d) are:

H (Halt)

LCR (Load Control Registers)

PDT (Peripheral Data Transfer)

PCB (Peripheral Control and Branch)

SVI (Store Variant and Indicators)

RVI (Restore Variant and Indicators)

RNM (Resume Normal Mode)

LIB (Load Index/Barricade Register)

The above op codes are "privileged" in the sense they are allowed to be executed in either of the interrupt modes but are prohibited in the standard mode while storage protection is in effect (one exception to this is discussed under "Proceed Indicator" below). Such op codes are categorized by their capability of altering the monitor's knowledge of the status of the system or causing some action that is intolerable under certain conditions (e.g., a halt during transfer of data from a communications device). Since an undefined op code or one that is not installed on the user's processor would normally cause a halt due to a program check, such usage has the same effect as that of a privileged op code.

NOTE: The Extended Multiprogramming feature provides additional "privileged" op codes.

---

<sup>1</sup>For example, a MAR with 15 active bits can address up to 32,768 locations; a MAR with 16 active bits can address up to 65,536 locations. A 49,152-character memory would require 16 active bits, thus making it possible to store an address which is beyond the actual memory size.

<sup>2</sup>The final responsibility for determining whether the parity check actually indicates out-of-range addressing rests with the programmer.

NOTE: Op code "00" is defined as an Internal Interrupt Call, and falls within the category of privileged op codes.

If a privileged op code is extracted when storage protection is in effect, the op code violation indicator is turned on, the sequence register is set back to the location of the op code, the operation is terminated, and an internal interrupt occurs. Once the internal interrupt mode is entered, the programmer has two choices: (1) if he wishes to execute the privileged instruction, he must set the proceed indicator (see below) and issue a Resume Normal Mode (RNM) command;<sup>1</sup> (2) if he wishes to bypass the privileged instruction, he must set the internal interrupt register to the location of the next sequential op code and issue a Resume Normal Mode instruction.<sup>2</sup>

### PROCEED INDICATOR

The proceed indicator can be turned on by the Restore Variant and Indicators (RVI) instruction. Turning this indicator on permits the execution of one privileged instruction in the standard mode without op code checking or item-mark trapping being performed. The indicator is turned off following the extraction of any op code in the standard mode. It can also be turned off in either of the interrupt modes by a Store Variant and Indicators (SVI) instruction.

The proceed indicator can also be used to force the checking of the A-address of a PDT instruction executed in either the internal or external interrupt mode. Thus, turning on this indicator prior to the extraction of a PDT instruction in a nonstandard (interrupt) mode results in the same address violation check as though it were extracted in the standard mode with storage protection in effect. If the effective A-address is found to reference a protected area, the actions described below are performed.

1. When the violation occurs in the internal interrupt mode:
  - a. The internal interrupt address violation indicator is set.
  - b. Further extraction of the instruction is not performed and the sequence register is set to the location of the next sequential op code.
  - c. An internal interrupt does not occur since the processor is already in the internal interrupt mode. Instead, the condition of the internal interrupt address violation indicator must be tested by the programmer after he has stored the status of the indicator via an SVI instruction. The SVI instruction also clears the indicator so that it will not cause an internal interrupt to occur when the standard mode is entered later.
2. When the violation occurs in the external interrupt mode:
  - a. The external interrupt address violation indicator is set.

---

<sup>1</sup>The instruction will still not be executed if it involves an address violation.

<sup>2</sup>If the internal interrupt register (which is currently set at the location of the privileged op code) is not advanced to the next op code, the return to normal mode results in the privileged op code again being extracted, thus causing an endless loop.



- b. Further extraction of the instruction is not performed and the sequence register is set to the location of the next sequential op code.
- c. An internal interrupt does not occur since this is impossible while in the external interrupt mode. Instead, the condition of the external interrupt address violation indicator must be tested by the programmer according to the method described in 1. c, on Page 2-25.

#### EXTENDED MULTIPROGRAMMING AND EIGHT-BIT TRANSFER

Extended multiprogramming provides a processor with five basic capabilities required in a multiprogramming environment and one feature required for upward compatibility. These are:

1. Base relocation,
2. Storage protection with base relocation,
3. Interrupt masking,
4. Instruction timeout,
5. 8-bit transfer capability, and
6. Privileged BCT and SCR Instructions.

#### STORAGE PROTECTION WITH BASE RELOCATION

In a processor equipped with extended multiprogramming, storage protection operates in either of two ways: with or without base relocation. Storage protection without base relocation operates as described above.

The storage protection offered by extended multiprogramming is made possible by using base relocation in conjunction with storage protection. Base relocation is in effect when the relocation indicator is set (via the SVI and RVI instructions) and the processor is in the standard (noninterrupt) mode.

Storage protection with base relocation places a barrier above and below the area of memory where the active program is to operate, to prevent it from altering the contents of the rest of memory. The lower barrier is specified by the contents of the base relocation register (BRR), which is loaded and stored via Load Index/Barricade Register (LIB) and Store Index/Barricade Register (SIB) instructions. When relocation is in effect, the BRR is loaded with the bank address of the lowest memory bank (4,096 characters) available to standard mode programs. The address in the BRR is added to each processor memory address transmitted to memory by a standard mode program. This prevents a standard mode program from writing into a memory bank below that specified by the BRR. The upper barrier is specified by the contents of the index barricade register (IBR), as augmented by the base relocation address. (The IBR contains the number of 4,096-character memory banks that are available to a program. Adding the contents of the BRR to the contents of the IBR gives the effective ("relocated") address of the

index barricade.) When storage protection is in effect and an attempt is made to write into memory at an address greater than that stored in the IBR (as augmented), a protection violation occurs resulting in an internal interrupt.

A monitor program keeps track of the locations of the various programs stored in memory and, via the settings of the BRR and the IBR, can relocate reference to any number of 4,096-character banks of memory. Thus, while there may be any number of programs stored in memory, only one program is active at any one time and all other programs are protected from the active program when storage protection is in effect. When, as the result of an interrupt, the monitor program activates a different program, it simply alters the settings of the BRR and the IBR to make available a different portion of memory.

Since all memory references are relocated via the BRR when relocation is in effect, index registers X1 through X15 effectively reside in the 4,096-character bank of memory specified by the BRR. The location of index registers Y1 through Y15 also dependent on the setting of the relocation indicator. When relocation is activated, the Y index registers are also located in the 4,096-character bank specified by the BRR, where they become identical to index registers X1 through X15. When relocation is in effect, each program stored, including the monitor program, has its own set of 15 index registers when it is the active program. The index registers always reside in the memory area occupied by the active program.

#### EXTERNAL INTERRUPT MASKING

Each input/output (I/O) sector has associated with it a 1-bit mask. This mask is stored and set by Store Variant and Indicators (SVI) and Restore Variant and Indicators (RVI) instructions, respectively. When the mask for a sector is a 0, interrupts from sources in that sector are accepted and processed in the manner specified in "Interrupt Processing". When the mask for a sector is a 1, then interrupts are held until the mask is altered or the interrupt function is reset. Control panel and Monitor Call interrupts are never masked. Depression of the INITIALIZE button on the console causes all mask bits to be reset.

#### INSTRUCTION TIMEOUT

It is possible for an instruction in a program to enter an infinite extraction or execution loop which would prevent a monitor program from servicing an interrupt within a specified time. To prevent this from occurring, a timeout function is provided which allows a maximum time limit to be placed on the extraction and the execution of any one instruction when the processor is in the standard mode. This function guarantees that a monitor program will, at some specified time, regain control of the system.

The instruction timer is reset to 0 and begins timing every time the processor starts to extract or execute a new instruction. If the timeout allow function is on, the protect indicator is set, and the processor is in the standard mode when the time interval elapses, then the instruction being extracted or executed is terminated and an internal interrupt occurs.

The timeout function is enabled by a timeout allow function which is set and reset by the SVI and RVI instructions. Refer to Section VIII for SVI and RVI instructions.

#### EIGHT-BIT TRANSFER CAPABILITY

Central processors equipped with this capability can transfer data between peripheral controls and memory in either six- or eight-bit format, as specified in the Peripheral Data Transfer (PDT) instruction.

1. The six-bit mode is the standard data transfer mode used in Series 2000 central processors. In this mode, only data is transferred between memory and peripheral controls. Punctuation is preserved in memory.
2. The eight-bit mode is used in those applications where an eight-bit transfer is desired between the central processor and a peripheral control. In this mode of operation, data and punctuation are transferred between the central processor and peripheral controls. Record marks in memory do not terminate data transfer in this mode.

When in the eight-bit mode, the number of eight-bit character transfers to be performed is determined by a three-character count field in the PDT instruction or by control characters associated with the PDT peripheral controls.

The high-order bit of the C3 control character in a PDT instruction is a multivariant bit which conditions the peripheral control in its interpretation of the remainder of the instruction. When this bit is a 0, all additional control characters beyond C3 are ignored by the control. When the high-order bit of C3 is a 1, additional control characters are present and will be accepted by the peripheral control. In this case, the format of the PDT instruction becomes:  
op code/A address/C1, C2, C3, C4, C5, C6, C7.

Control character C4 is always present when the multivariant bit (bit 6 of C3) is a 1. When the extended bit (bit 5 of C3) is a 1, control characters C5, C6, and C7 are present. When the extended bit is a 0 control characters C5, C6, and C7 are ignored. The high-order bit of C4 determines the data transfer mode; 1 specifies eight-bit mode and a 0 specifies six-bit mode. Because eight-bit mode data transfers are not affected by record marks, data transfer is delimited by the setting of the extended bit in the C3 control character. If this bit is a 0, all data transfers previously terminated by a record mark are now terminated by transferring the number of characters specified in the record header area. If it is a 1 all data transfers previously terminated by a record mark are now terminated by transferring the number of characters specified by the count field (C5, C6, and C7) of the PDT instruction.

## PRIVILEGED SCR INSTRUCTION

When a processor is in the standard mode with the storage protection indicator ON and the proceed indicator OFF, the detection of an SCR instruction having a variant character of octal 00 through octal 37 will set the op code violation indicator and cause an internal interrupt to occur at the next opportunity.

The following status is specified at the conclusion of the trapped SCR instruction.

1. The internal interrupt register (IIR) contains the address of the privileged op code.
2. The A-address register (AAR) contains the A-address of the previous instruction.
3. The contents of the main memory locations specified by the A-address are undisturbed.
4. The variant register contains the variant character of the privileged SCR instruction.

All SCR instructions are identically executed if the proceed indicator is ON.

## PRIVILEGED BCT INSTRUCTION

When a processor is in the standard mode with the storage protection indicator ON, the proceed indicator OFF, and the BCT privileged indicator ON, the detection of a BCT testing the status of any SENSE switch (i.e., having an octal variant of 01 through 17 or 21 through 37) will set the op code violation indicator and cause an internal interrupt to occur at the next opportunity.

The following status is specified at the conclusion of the trapped BCT instruction.

1. The internal interrupt register (IIR) contains the address of the privileged op code.
2. The A-address register (AAR) contains the A-address of the privileged BCT instruction.
3. The contents of the main memory locations specified by the A-address are undisturbed.
4. The variant register contains the variant character of the privileged BCT instruction.

All BCT instructions are identically executed if the proceed indicator is ON.

All BCT privileged indicators can be set or reset under program control only by the RVI instruction.

## HIGH-RESOLUTION CLOCK

A high-resolution clock capability is standard on the multicharacter processors except the Type 2041A. With Honeywell software (OS/2000 or Mod 4) or with individual programming, the high-resolution clock can be used to provide accurate central processor job accounting.

When activated, the high-resolution clock counts elapsed central processor processing time and generates an interrupt after timing out. Elapsed central processor processing time is defined as the time elapsed with the processor in the RUN state and having access to memory. In order to exclude counting I/O time, the clock is inhibited during buffer cycles.

The clock's count of elapsed time is contained in a control memory register known as the accounting timer register (ATR). The current count can be stored by an SCR instruction. In addition, the register can be loaded to any value by an LCR instruction. When the full count is reached (see Table 2-4), an External Interrupt (EI) demand is set. This demand is reset and its indicator cleared when it is stored by an SVI instruction.

Table 2-4. Clock Characteristics

Processor	Resolution (memory cycles)	Timeout (minutes)
2041A	256	5.60
2041A with PM2A40	256	3.36
2041A with PM2B40	256	2.24
2051C	256	3.58
2051A	256	4.48
2051A with PM2A50	256	3.36
2051A with PM2B50	256	2.24
2061	256	2.55
2071	256	2.24

### ACCOUNTING TIMER REGISTER

The accounting timer register is assigned to location (54<sub>g</sub>) in control memory. Its mnemonic designation is ATR. All bits of the register are used.

### EXTERNAL INTERRUPT MODE

When the timer goes beyond its full count, the resulting overflow generates an EI demand. In EI mode, the timer continues to count elapsed time without pause. The interrupt source can be identified with the SVI instruction. The item-mark bit of the sixth character stored indicates a high-resolution clock interrupt. This indicator is cleared when stored.

## SCR AND LCR INSTRUCTIONS

SCR and LCR instructions to the ATR execute in the normal manner. Turning the clock on does not reset the ATR. It is the programmer's responsibility to reset the ATR to zero or any other desired value via an LCR instruction. The SCR and LCR instructions should be executed in four-character mode, as all bits of the ATR are used by the clock.

## HIGH-RESOLUTION CLOCK ALLOW

An LIB instruction is used to turn the clock on or off. The required format is F/A/B/V. Bit 2 of the C3 variant controls the on/off state as follows:

000X10	Turn on the clock
000X00	Turn off the clock

In addition to controlling the on/off state, the LIB instruction in this format executes normally. The SID instruction is not influenced.

## INTERRUPT PROCESSING

The execution of main-program instructions by the processor can be interrupted by an external interrupt source and/or by an internal interrupt source.

## EXTERNAL INTERRUPT

An external interrupt signal can be generated by any or all of three sources:

1. The operator's control panel or console;
2. The Monitor Call instruction; or
3. A peripheral control.

The first two sources interrupt the processor directly: in the case of the control panel or console, the operator simply presses the INTERRUPT button; the Monitor Call instruction interrupts the processor when it is executed. However, a peripheral control interrupts program sequence as directed by the settings of two programmable storage functions contained within the control.

The interrupt signal sets indicators to show the source (whether 1., 2., or 3., above) and the type (external) of interruption. These indicators can be stored and then tested by programmed instruction as described later in this section. The processor acts upon the interrupt signal when the following conditions are present:

1. The processor is in the RUN mode (i.e., the processor is executing, without manual intervention, stored-program instructions under control of SR).
2. The processor is not in the external interrupt mode.

3. An instruction op code is about to be extracted.
4. A memory cycle is allocated to the processor.

It should be noted that condition 3. above does not cause an extensive delay if a processor is attempting to extract a Peripheral Data Transfer (PDT) instruction and the specified read/write channel or peripheral control is "busy." The attempt to issue a PDT instruction to a busy read/write channel or peripheral control does not "stall" the central processor. Rather, the instruction is "re-extracted": SR is set back to the address of the PDT op code, so that condition 3. recurs after the channel or control is found busy.

When the central processor is interrupted, it performs the following functions:

1. Stores the current status of the arithmetic, comparison, address mode, and trap mode indicators in the auxiliary indicators register (AIR).
2. Clears the arithmetic indicators.
3. Enters the three-character, non-trap mode.
4. Interchanges the contents of SR and EIR and branches to the instruction whose op code address was previously stored in EIR.
5. Enters the external interrupt mode.

The interrupt signal is maintained until one of the following steps is taken:

1. A PDT instruction is issued to the peripheral control.
2. The Interrupt function for the peripheral control is turned off.
3. The central processor is initialized.

### INTERNAL INTERRUPT

An internal interrupt signal is caused by a "violation" of storage protection. Processor indicators are set by the internal interrupt signal to show the cause (e.g., op code violation) and the type (internal) of interruption. These indicators can be stored and then tested by programmed instruction as described later in this section.

The processor reacts to the internal interrupt signal when the conditions described on the preceding page are present (i.e., the processor is in the RUN mode, is not in the external interrupt mode, is about to extract an op code, and is presently allocated a memory cycle) plus one additional condition: the processor must not only not be in the external interrupt mode but also must not be in the internal interrupt mode. Thus, the following levels of interrupt priority exist in the affected processors.

1. If the processor is in the non-interrupt (standard) mode, normal program sequence can be interrupted by either an external or an internal source.
2. If the processor is in the internal interrupt mode, program sequence can be interrupted only by an external interrupt source.
3. If the processor is in the external interrupt mode, program sequence can not be interrupted.<sup>1</sup>

The processor responds to an internal interrupt signal as follows:

1. The contents of SR and IIR are interchanged, and the program branches to the instruction whose op code address was previously stored in IIR.
2. The processor enters the internal interrupt mode.

Note that the status of the arithmetic, comparison, address mode, and trap mode indicators are not stored in AIR automatically when the processor responds to an internal interrupt signal. The storing (and subsequent restoring) of the contents of these indicators is the responsibility of the internal interrupt program.

### INTERRUPT PROGRAMMING

Three of the four interrupt control instructions perform basic functions in an interrupt routine:

1. The Store Variant and Indicators instruction (SVI) stores two types of information: (a) information which must be preserved for subsequent return to the interrupted program (e.g., indicator settings, variant register contents, etc.); and (b) information required to identify the interrupt source.
2. The Restore Variant and Indicators instruction (RVI) restores the pertinent information stored by the SVI instruction before returning to the interrupted program.
3. The Resume Normal Mode instruction (RNM) returns the processor to the interrupted program, unless the sector bits of SR have been modified.

The fourth interrupt control instruction – Monitor Call (MC) – causes an external interruption and, therefore, is not coded in the interrupt routine itself.

Other instructions are required in the interrupt routine to store and exercise control over address register contents, as shown in Figures 2-18 and 2-19. The interrupt routine in these figures are assumed to be executed in the same sector as the interrupted program; if not, or if interrupt processing modifies the sector bits in SR, the appropriate sector bits must be stored upon entering the routine and restored when exiting.

---

<sup>1</sup>Interrupt signals generated by any or all of the three external sources (peripheral control, control panel or console, or Monitor Call instruction) may continue to occur while the processor is in the external interrupt mode. The priority in which the interrupts are accommodated is determined by the program (i.e., according to the programmer-established sequence of interrupt source tests).



For proper re-entry to the interrupted program, the same set of indicators stored by the SVI instruction should be restored by the RVI. Since the RVI instruction prepares the processor to re-enter the interrupted program, it should be followed immediately by the RNM instruction. Note that the A- and B-address register settings at the time of the interrupt should also be restored before re-entering the interrupted program. The external interrupt coding shown in Figure 2-18 exploits the ability to restore the address registers automatically by storing their contents in the address fields of the RNM instruction. This technique requires that variant bit  $V_2$  of the RVI instruction be a 0 in order to ensure that the RNM instruction is executed in the maximum address mode of the machine. In an internal interrupt routine, on the other hand, the indicators associated with the  $V_2$  must be stored and restored by the SVI/RVI instructions. Therefore, since the address mode of executing the RNM instruction may not be maximum, the address fields of this instruction must not be coded. Instead, the address register settings must be stored in memory and restored by means of LCR instructions, as shown in Figure 2-19.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION	LOCATION	OPERATION CODE	OPERANDS	
				1-4	5-8
1		AAR	CEQU	#1C67	A-ADDRESS REGISTER
2		BAR	CEQU	#1C70	B-ADDRESS REGISTER
3		MAX	CEQU	#1C60	MAXIMUM ADD. MODE FOR C.P. IS 4
4		ALLS	CEQU	#1C75	INDICATORS STORED
5		ALLR	CEQU	#1C35	INDICATORS RESTORED
6			ADMODE	4	SET MAXIMUM ADDRESSING MODE
7		RESTOR	RVI	ENTER+2, ALLR	RESTORE INDICATORS MAXIMUM
8		EXIT	RNM	0, 0	EXIT WITH AAR + BAR RESTORED
9		ENTER	SVI	ALLS	ENTER AND STORE INDICATORS
10			DCW	#5	RESERVE STORAGE FOR INDICATORS
11			CAM	MAX	ENTER MAXIMUM ADDRESS MODE
12			SCR	EXIT+4, AAR	SAVE AAR
13			SCR	EXIT+8, BAR	SAVE BAR
14					
15					
16					EXTERNAL INTERRUPT ROUTINE
17					
18					
19					
20		B	RESTOR		BRANCH TO RESTOR AND EXIT

Figure 2-18. Sample Coding for External Interrupt Routine

The first example (see Figure 2-18) shows the initial and final coding to be used in an external interrupt routine. It is assumed that the address of the location tagged ENTER was previously stored in EIR, so that the presence of an external interrupt signal results in the automatic branch to the location tagged ENTER. It is assumed that the four-character addressing mode is the maximum addressing mode of the processor for which this routine is written.

NOTE: If the interrupt routine is not in the maximum addressing mode prior to branching to the location tagged RESTOR, a Change Addressing Mode instruction - CAM/MAX - must precede the RVI instruction so that the complete contents of any necessary control memory locations may be restored.

Figure 2-19 shows the initial and final coding written for an internal interrupt routine. It is assumed that the address of the location tagged START was previously stored in IIR and that the maximum addressing mode of the processor is the four-character mode.

The initial and concluding instructions in an internal routine are similar to those in an external interrupt routine, except that the SVI instruction must store the indicators associated with bit  $V_2$  and must not store the contents of the auxiliary indicators register (AIR). All other pertinent indicators are stored by the SVI instruction and are subsequently restored by the RVI instruction at the conclusion of the routine.

### EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	IIR	MIR	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7	8	9	10 11 12 13 14 15	16 17 18 19 20 21	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
1			AAR	CEQU	#1C67 A- ADDRESS REGISTER
2			BAR	CEQU	#1C70 B-ADDRESS REGISTER
3			MAX	CEQU	#1C60 MAXIMUM ADD. MODE FOR CP 1 S. 4
4			INDS	CEQU	#1C73 ALL BUT AIR INDICATORS
5			INDR	CEQU	#1C33 ALL BUT AIR AND INT. INDICATORS
6			SAVEA	DCW	*4C TEMPORARY STORAGE FOR AAR
7			SAVEB	DCW	*4C TEMPORARY STORAGE FOR BAR
8				ADMODE	4 SET MAXIMUM ADDRESSING MODE
9			RESTOR	LCR	SAVEA, AAR RESTORE AAR
10				LCR	SAVEB, BAR RESTORE BAR
11				RVI	START+2, INDR RESTORE ALL BUT AIR AND INT. IND.
12				RNM	EXIT
13			START	SVI	INDS ENTER AND STORE ALL BUT AIR IND.
14				DCW	#5 STORAGE FOR ALL BUT AIR IND.
15				CAM	MAX ENTER MAXIMUM ADDRESSING MODE
16				SCR	SAVEA, AAR SAVE AAR
17				SCR	SAVEB, BAR SAVE BAR
18					
19					
20					
21					
22					
23					
24			B	RESTOR	BRANCH TO RESTOR AND EXIT

Figure 2-19. Sample Coding for Internal Interrupt Routine

### PERIPHERAL CONTROL INTERRUPT

This description pertains to most Series 200/2000 peripheral controls; exceptions are noted in the various hardware manuals describing individual peripheral devices.

Generally, a peripheral control's interrupt facility includes two interrelated functions: the Allow function and the Interrupt function. Certain controls have more than one set of functions (e.g., two sets for disk controls, but one set for magnetic tape controls). When a peripheral control becomes ready to accept a PDT instruction (i.e., reaches a "not-busy" status), it transmits a signal to turn on the Interrupt function, but this signal must be complemented by one from the Allow function (turned on by a PCB instruction) in order to complete the interrupt signal for transmission to the central processor (see Figure 2-20). When the Interrupt function is turned on, the interrupt signal is repeated continuously until the central processor is interrupted or the signal is turned off.

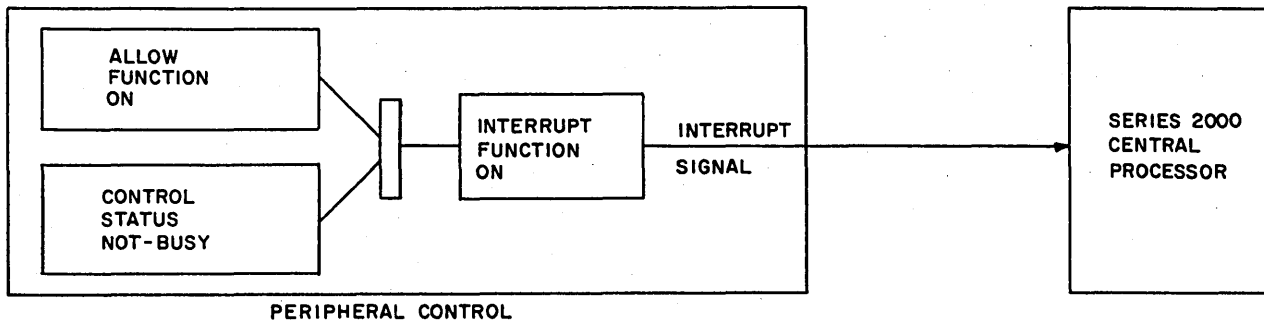


Figure 2-20. Interrupt Signal Generated by Peripheral Control

The interrupt facility for a peripheral control can be activated or deactivated simply by turning the Allow function on or off, respectively. If the Allow function is off at the time the peripheral control becomes not busy and all error information is stored, the interrupt signal can be neither completed nor transmitted. Another method of inhibiting the interrupt facility is to turn off the Interrupt function; this function will not be turned on again until the control completes another PDT instruction. Note that if an interrupt has occurred and the Allow function has then been turned off, the Allow function should not be turned on again until either the Interrupt function has been turned off or a PDT instruction has been initiated by the control; otherwise, an interrupt occurs immediately.

There are various methods of turning the Allow or Interrupt function on or off. The Allow function can be turned on or off by a PCB instruction; similarly, the Interrupt function can be tested or turned off by a PCB instruction. Also, when the peripheral control receives a PDT instruction, its Interrupt function is turned off automatically; at completion of the PDT, a pulse is sent to turn on the Interrupt function. In any situation, both functions are turned off by initializing the central processor.

Specific PCB C3 characters for individual controls are listed in Tables 8-34 through 8-36. The C3 character in a PCB instruction may be used either to control or to test the status of a peripheral control's interrupt facility. The general formats of the C3 characters relating to interrupt control and test are:

- 1110x0 - Turn off the Allow function.
- 1110x1 - Turn on the Allow function.
- 1111x0 - Turn off the Interrupt function.
- 1111x1 - Branch to A if the Interrupt function is on.

The two-bit, shown here as x, is normally 0 if the control being addressed contains only one set of Interrupt/Allow functions. If two sets of functions are present, this bit is set to identify the particular set being tested or controlled. All of these C3 characters result in a branch to A if the device addressed is not operable. Table 2-5 summarizes Interrupt/Allow control and test operations for most peripheral controls; exceptions are noted in individual device manuals.

More than one control character can be used to specify multiple control and/or test operations in a PCB instruction. However, care must be taken in the use of certain combinations of these characters. For example, it is entirely possible for an interrupt to occur between extractions of control characters. In such a case, if control characters for "Branch on Interrupt" and "Turn Off Interrupt" were specified (in that order), the Interrupt function might be turned off without being acknowledged.

Table 2-5. Summary of Interrupt/Allow Function Control and Test Operations

Control/Test Operations	Resulting Effects	
	Allow Function	Interrupt Function
<u>Manual</u>		
INITIALIZE Button	Turned off	Turned off
<u>Program - PCB Control Char.<sup>a</sup></u>		
70	Turned off	None
71	Turned on	None
74	None	Turned off
75	None	Branch to A if on
<u>Peripheral Control</u>		
Upon receipt of PDT	None	Turned off
When PDT completed	None	Turned on if Allow on
<sup>a</sup> All of these PCB control characters will result in a branch to A if the device addressed is not operable.		



## SECTION III DATA FORMAT

### VARIABLE FIELD LENGTH

Information is stored in the main memory in fields. A field is, by definition, any group of characters that is treated as a unit. Series 2000 computers permit fields of any length, from one character up to the maximum number of characters in the memory. This means that an instruction or data field occupies only that number of core storage locations actually needed.

The use of variable-length fields requires that there be a method of indicating the actual lengths of instruction fields and data fields. This requirement is fulfilled by the word-mark bit mentioned in Section II. The word-mark bit performs the following functions:

1. It terminates the retrieval of an instruction.
2. It terminates the execution of an instruction.
3. It defines the size of a data field.

Throughout this manual, the presence of a word mark will be indicated by a circle around the character with which it is associated. The following points should be noted regarding the use of word marks:

1. Word marks can be set and cleared by programmed instructions.
2. Word marks are set by the same routine that loads a program and data into the main memory. Usually, word-mark assignments remain unchanged throughout the execution of a program.
3. An instruction is terminated by a word mark in the storage position immediately following its last (rightmost) character.
4. A data field is terminated by a word mark associated with its high-order (leftmost) character.<sup>1</sup>

---

<sup>1</sup>The footnote on page 3-4 describes an exception.

## INSTRUCTION FORMAT

An instruction is a coded statement that orders the computer to perform a fundamental operation. A set of instructions suitably combined to perform a specific task is called a program or routine.

As will be shown in Section V, the task of coding the instructions in a program is greatly simplified by the use of the Easycoder symbolic programming system. The Easycoder Assembly Program converts the symbolic coding written by the programmer into a machine language that is acceptable to the internal logic of the machine.

### OPERATION CODE

Basic to all instructions is an operation code, usually referred to as an op code, that defines the fundamental operation to be performed. The programmer specifies an op code by using a predefined mnemonic configuration; e.g., BA is the op code that specifies a "binary add" operation, MCW is the op code that specifies a "move characters to word mark" operation. The Easycoder Assembly Program automatically converts a mnemonic op code into a single-character, machine-language op code and sets the word-mark bit in the character position in which it is stored.

### A- AND B-ADDRESSES

Most instructions also have two address portions, designated as the A address and the B address. The address portions indicate the starting locations of the operand fields in the main memory. Using the Easycoder language, the programmer can specify memory locations by means of symbolic addresses or "tags" (see Section V).

The Easycoder Assembly Program automatically assigns absolute memory addresses to the symbolic addresses appearing in a program (see Figure 3-1). Thus, the programmer can manipulate operands without regard to their actual storage locations in memory.

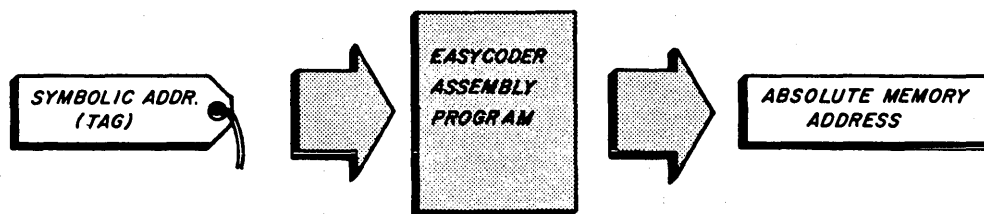


Figure 3-1. Conversion of Symbolic Tag to Absolute Memory Addresses

Because of the modular design of Series 2000 computers, the programmer has the facility to specify whether a two-, three-, or four-character absolute address will be assigned to each symbolic address used in the program. In any case, the absolute addresses assigned by the assembly program are interpreted as pure binary numbers (see Section IV).

### VARIANT CHARACTER

The variant character is used to modify the op code of an instruction. For example, the op code of a Branch on Condition Test instruction (BCT) specifies the fundamental operation "branch if a tested condition is met." The condition or restriction that must be met before the branch can occur is specified by the variant character. A table of valid variant characters is presented in Appendix B.

### SUMMARY

Figure 3-2 shows the six basic formats in which machine-language instructions may appear. Since the maximum number of characters in an instruction depends upon whether two-, three-, or four-character addressing is being used, shaded boxes in the illustration indicate the format of an instruction without specifying the number of characters in each part. These formats are representative of all instructions except those associated with input/output and translate operations.<sup>1</sup> For the sake of direct comparisons, Figure 3-3 illustrates each of the formats defined in Figure 3-2 as a symbolic entry on the programmer's coding form.

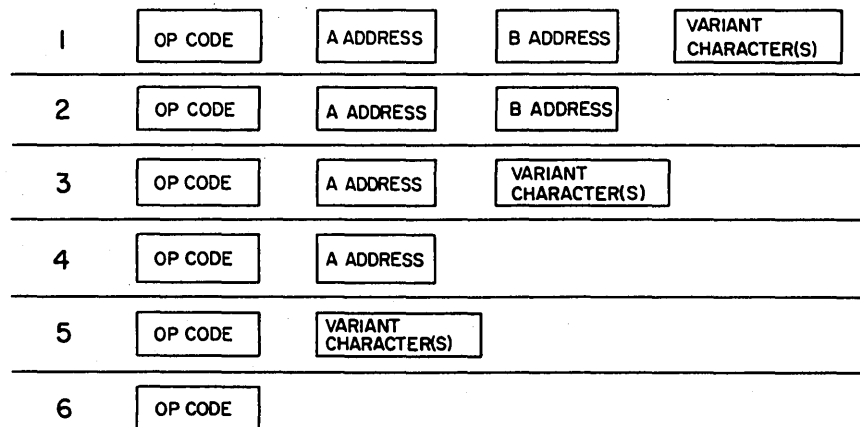


Figure 3-2. Series 2000 Instruction Formats

<sup>1</sup>The format of an input/output instruction is a modification of format 3 shown in Figure 3-2. Specifically, the variant characters of the instruction are replaced by a field of one or more control characters which define the input/output operation in terms of data path, direction of data flow, control unit designation, etc. The format of a translate instruction is a modification of format 1 shown in Figure 3-2. In Section VIII, Series 2000 instructions are described in terms of their individual formats.



# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
1			BCE	P6, LABEL, 06	FORMAT 1
2					
3			A	ITEM, TOTAL	FORMAT 2
4					
5			BCT	BZRO, 03	FORMAT 3
6					
7			SW	WORK	FORMAT 4
8					
9			CAM	60	FORMAT 5
10					
11			S		FORMAT 6
12					
13					
14					
15					

Figure 3-3. Symbolic Representation of Series 2000 Instructions

## ORGANIZATION OF DATA IN MAIN MEMORY

Data may be stored in the main memory in any of the following variable-length formats:

- Field
- Item
- Record

### FIELDS

Consider the eight consecutive storage locations shown in Figure 3-4. To indicate to the machine that these eight characters are to be treated as a field, their left and right boundaries must be defined. The left boundary is normally defined by setting a word mark in position 990. The right boundary is normally defined by specifying storage address 997 in the instruction that will manipulate the field.<sup>1</sup> The eight-character group shown in Figure 3-5 is thus defined as a field.

STORAGE ADDRESS →	990	991	992	993	994	995	996	997
CONTENTS →	7	3	6	6	9	5	2	9

Figure 3-4. Consecutive Storage Locations in Main Memory

<sup>1</sup> Although this is the conventional method of defining fields, the Extended Move (EXM) and Move or Scan (MOS) instructions (see Section VIII) permit a field to be defined by a word mark at either the left or the right boundary. The opposite boundary is then specified in the instruction.

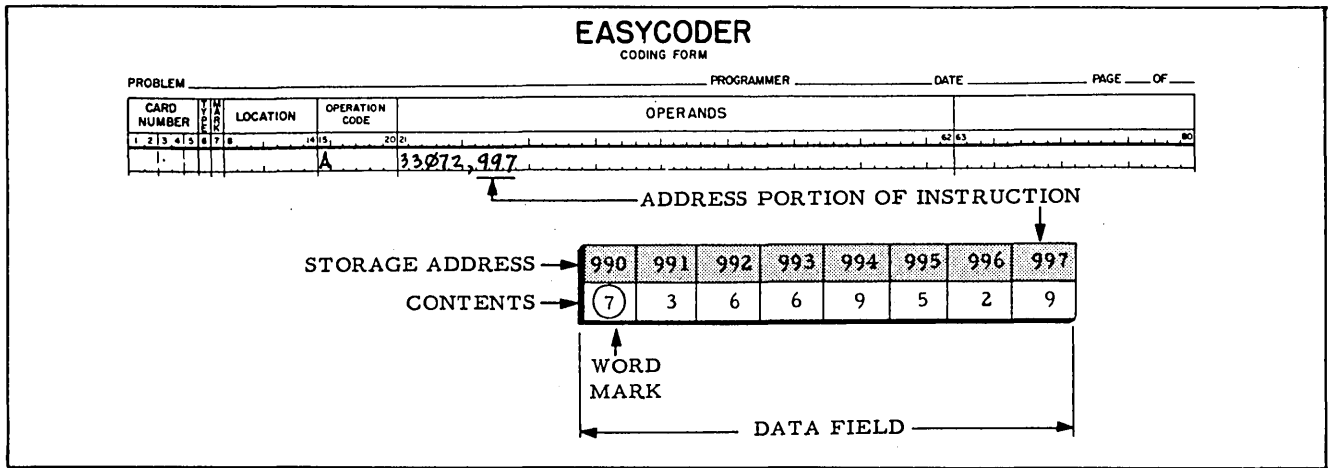


Figure 3-5. Data Field Format in Main Memory

**ITEMS**

An item consists of one or more consecutive storage locations whose boundaries can be defined in either of two ways:

1. The leftmost character position can be defined in the instruction that will operate on the item and the rightmost character position defined by an item mark; or
2. The rightmost character position can be defined in the instruction that will operate on the item and the leftmost character position defined by an item mark.

NOTE: An item mark is illustrated in this manual by underlining the character with which it is associated. Fields within an item are defined by word marks.

There are only two instructions that manipulate items — Move Item and Translate, and Extended Move. In the Move Item and Translate instruction, the leftmost character of an item is addressed and the rightmost character contains an item mark. In the Extended Move instruction, several different item boundaries can be specified by the variant character of the instruction.

Two items, each containing three data fields, are shown in Figure 3-6.

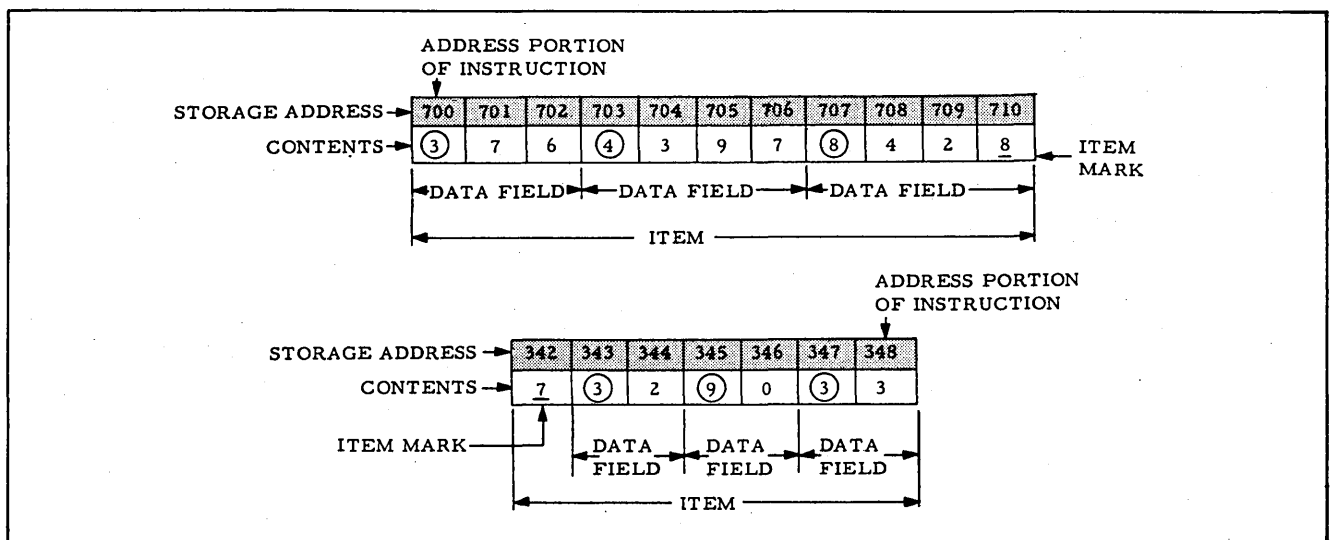


Figure 3-6. Two Item Formats in Main Memory

## RECORDS

A record is any unit of information that is to be transferred between the main memory and a peripheral device. A record can be of any length, from one character up to the maximum number of characters in the memory. It can contain any number of items and fields. The rightmost limit of a record is defined by a record mark in the character position following the last character in the record (see Figure 3-7).

NOTE: A record mark is illustrated by combining the word-mark and item-mark symbols. The address of the leftmost character in a record is specified in the instruction that operates on the record.

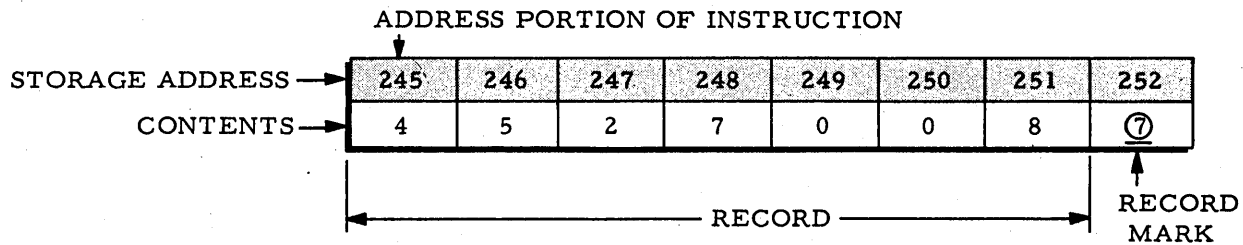


Figure 3-7. Record Format in Main Memory

## SUMMARY

The foregoing data format conventions are summarized in Table 3-1.

Table 3-1 Summary of Internal Data Formats

DATA FORMAT	BOUNDARY DEFINITION		INSTRUCTION USED TO SET MARK (See Section 8)
	LEFTMOST CHARACTER	RIGHTMOST CHARACTER	
FIELD <sup>a</sup>	Word Mark      ⊗	Address portion of instruction	Set Word Mark
ITEM	Address portion of instruction	Item mark <u>X</u>	Set Item Mark
	Item Mark <u>X</u>	Address portion of instruction	
RECORD	Address portion of instruction	Record mark      ⊗  (in character position following last character of record) <sup>b</sup>	BOTH Set Word Mark and Set Item Mark

<sup>a</sup> The Extended Move (EXM) and Move on Scan (MOS) instructions are exceptions to the field format shown (see page 3-4 and note).

<sup>b</sup> A record can also be moved internally (i.e., from one main memory area to another) by means of the Extended Move instruction (see Section VIII). In this case, the character containing the record mark is considered as part of the record. This instruction can specify either the right or left boundary of the record to be moved.

## MAGNETIC TAPE DATA FORMAT

In many applications, a major input and output medium for a Series 2000 computer is magnetic tape. The standard Series 2000 magnetic tape system uses 1/2-inch tape as the recording medium.

Information is stored on 1/2-inch magnetic tape in variable-length groups of characters called records. The tape is divided lengthwise into seven or nine recording tracks. A line of bit positions across the tape, one position for each track, is called a frame. On a 7-track tape, the seven bits in a frame correspond to the six information bits and one parity bit found in a character position in the main memory. On 9-track tape, in the standard packing mode, four main memory characters (27 bits) are written into three tape frames of eight data channels, and one parity channel. Notice that no tracks are provided for the storage of punctuation bits on tape. Unlike main memory records, which are delimited by record-mark punctuation, tape records are separated from each other by a band of blank tape, which is called an interrecord gap. The representation of a memory character position on 7-track magnetic tape is shown in Figure 3-8. (The punctuation bits have been moved within the main memory character position in order to simplify the diagram.)

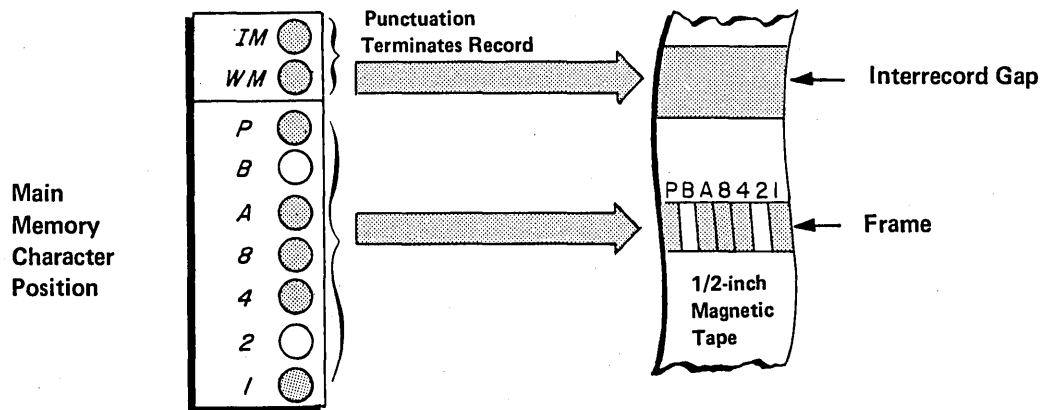


Figure 3-8. Character Representation on 7-Track Magnetic Tape

Characters recorded on magnetic tape are transferred from the main memory without parity bits. At the time of recording, the magnetic tape control generates parity bits as required. The programmer may specify either odd- or even-parity recording:<sup>1</sup> in the odd-parity mode the bit count in each frame is odd; in the even-parity mode the bit count is even.

<sup>1</sup> Feature 052 required on Type 204D-1, -1A, -3, -3A, -5 and -5A; and 204F-1, -3, and -5 Magnetic Tape Units.

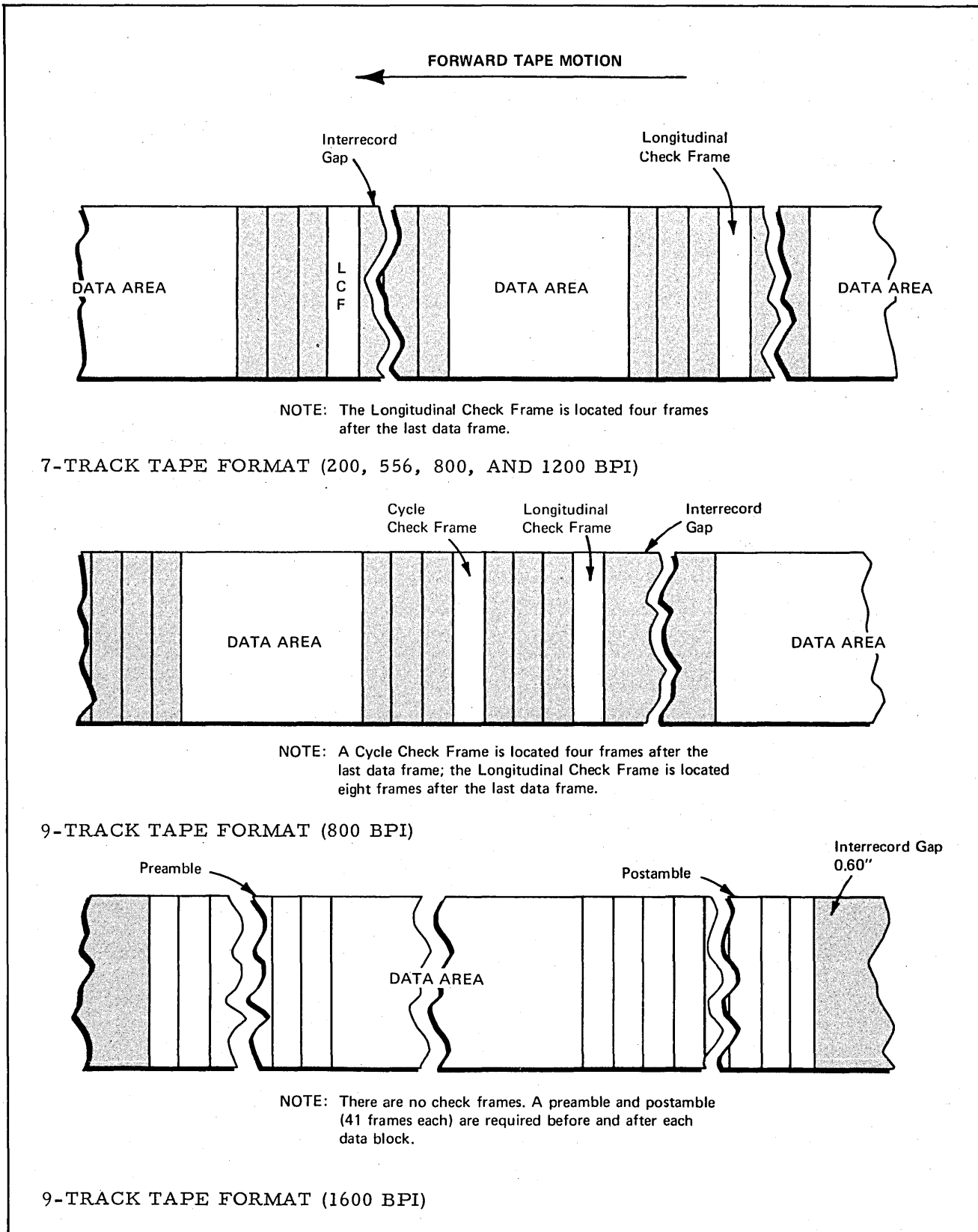


Figure 3-9. Data Format on Magnetic Tape

In addition to parity bits, which are used for frame checking, the magnetic tape control also generates a longitudinal check frame which is used for track checking purposes. A check frame is automatically appended to each record stored on tape.

Recall that a record stored in memory is delimited by a record mark in the character position following the last character in the record. When a record is transferred to tape, the contents of the character position containing the record mark are not included as part of the record. On the other hand, if a record mark is sensed in memory when information is being read in from tape, the record mark will terminate the record and the character position containing the record mark will receive a character from the tape. Although data transfer from the tape is terminated by the record mark, tape motion continues until an interrecord gap is sensed. No punctuation marks are altered in any way as a result of tape read/write operations initiated by a program. Figure 3-9 illustrates the data format on 7- and 9-track tape.

### PUNCHED CARD FORMAT

Punched cards provide a convenient means of entering data into the machine. The cards used for this purpose are either standard 12-row, 80-column cards or 12-row, 51-column cards. Each card column may contain a decimal digit, an alphabetic character, or a special symbol such as a slash or an asterisk (see Figure 3-10).

Numeric information is represented using the card punch positions labeled 0 through nine. Alphabetic information is represented by a combination of numeric punches and zone punches. There are three zone punch positions: the 12-zone at the top edge of the card, the 11-zone just below the 12-zone position, and the 0-zone labeled as row 0 on the card. The 11- and 12-zones are not labeled because the top edge of the card is reserved for printed headings. Notice that row 0 can represent numeric value when it is the only row punched, or a zone punch if used with another value.

In addition to Hollerith code, cards may be punched or read in the direct transcription mode as an optional feature. Each punch position on the card is individually significant in this mode, a punch representing a 1-bit and the absence of a punch representing a 0-bit.

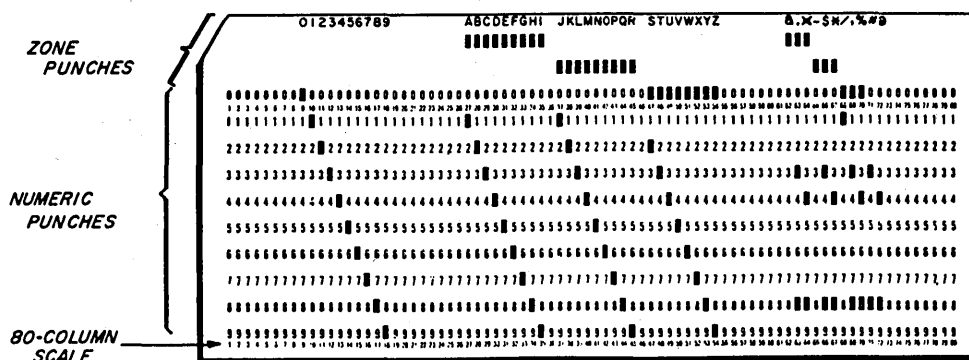


Figure 3-10. Punched Card Codes

## DISK FORMAT

### DATA CONVENTIONS

In disk processing, the basic unit of information is referred to as an item. An item is a logical unit of data, the smallest logical unit of data operated on by programmed instructions. For instance, it may be a single policy in an insurance policy file, or an employee's account in a master payroll file.

A record is a physical unit of data defined as the data written between two interrecord gaps on a track. A single record is the smallest physical unit of data that is operated on by programmed instructions. The number of items contained in a record is determined by the user. An item can be a portion of a record; equal to a record; or composed of more than one record, in which case the item is split between records. For example, if the record size is 250 characters, and the item size is 100 characters, two records contain five items.

RECORD 0 250 Characters			I R G	RECORD 1 250 Characters		
ITEM 0 100 char	ITEM 1 100 char	ITEM 2 50 char		ITEM 2 50 char	ITEM 3 100 char	ITEM 4 100 char

Figure 3-11. Relationship Between Items and Records

A block is defined as the sum of records that are transferred to or from main memory by a single data transfer operation. A block is a physical unit of data. It can contain one or more records, and its size is determined by the user. A block may be contained entirely on one track, or it may begin on one track and end on another. Since the contents of a block are transferred to or from memory, a buffer should be at least as large as a block. A block must contain a whole number of items.

BLOCK 0 1080 Characters						BLOCK 1 1080 Characters								
RECORD 0 540 Characters			I R G	RECORD 1 540 Characters			I R G	RECORD 2 540 Characters			I R G	RECORD 3 540 Characters		
ITEM 0	ITEM 1	ITEM 2		ITEM 3	ITEM 4	ITEM 5		ITEM 6	ITEM 7	ITEM 8		ITEM 9	ITEM 10	ITEM 11
180	180	180		180	180	180		180	180	180		180	180	180

Figure 3-12. Relationship Between Items, Records, and Blocks

A file is a collection of logically related items. A file is the largest unit of information that can be stored or retrieved by the operating system. For example, on a single volume, a single disk pack, there may be an inventory file, a payroll file and a customer-records file.

A volume is a physical unit of peripheral storage, as a disk pack, a reel of tape, or a deck of punched cards.

### TRACK FORMAT

Each track is composed of an index mark, which signifies the beginning of the track, and a variable number of data records, followed by a track-linking record which, if present, is the last logical record on the track. The track-linking record may be located anywhere on the track, as long as it logically follows all the data records on that track. The 277/279 disk drives contain a home address record that defines the track and cylinder in actual use or operation. Each one of these areas is separated by an interrecord gap. Refer to Figure 3-13.

### RECORD FORMAT

#### Address Mark

A record consists of three areas: the address mark, the header area, and the data area.

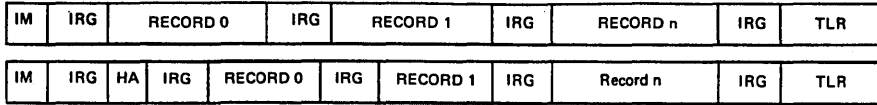
An eight-character Address Mark is automatically written during file formatting. Subsequent data transfer operations use that address mark to locate the beginning of each record. However, the address mark is never transferred to main memory.

#### Header Area

The header area of a record is formatted during a special file formatting procedure and is used later during file processing. The header contains: (1) a flag; (2) the address of the record; (3) the length of the data area in characters; and (4) the checking codes for the header area information.

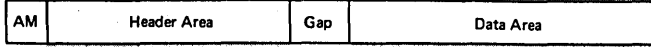


**TRACK**

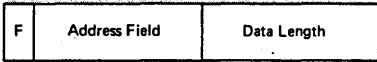


(277/279 Disk Pack Drives)

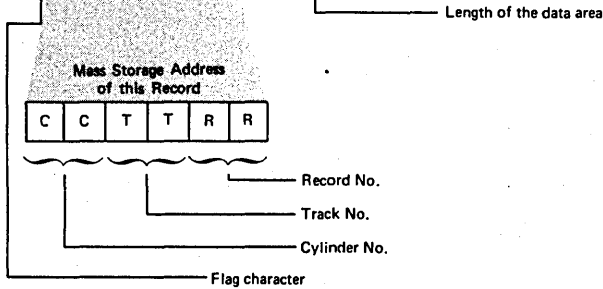
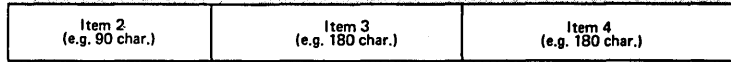
**RECORD**



**RECORD HEADER AREA**



**RECORD DATA AREA**



- IM = Index mark. Defines the beginning point of a track.
- IRG = Interrecord gap.
- TLR = Track-linking record. Last record written on a track. Gives mass storage address of next sequential record.
- AM = Address mark. Defines the beginning of each record.
- HA = Home address. Defines the track and cylinder being used. It is a record containing 5 bytes.

**Record** – A track area occupied by both data and the identifying fields associated with that data. This is the area originally laid out by the formatting program.

There are two types of information in a file: data information and track-linking information. Thus, the records used to contain this information are referred to as data records and track-linking records, respectively.

**Data Area** – Information transferred between the Series 2000 processor and a disk device by means of a data transfer instruction.

**Address Mark and Header Area** – The two identifying fields associated with each record.

The flag (F) is a 1-character field used for control purposes.

**Cylinder Number (CC)** – This 2-character field specifies (in binary) a cylinder number within a device.

**Track Number (TT)** – This 2-character field specifies (in binary) the track number within a cylinder on which the record is situated.

**Record Number (RR)** – This 2-character field identifies the binary record number within a track.

**EXAMPLE OF LOGICAL RELATION OF ITEMS, RECORDS, and BLOCKS:**

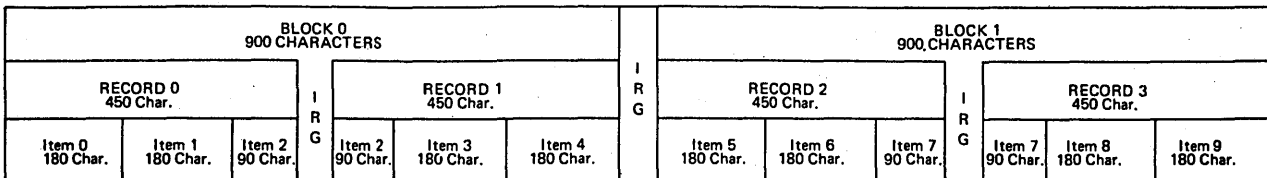


Figure 3-13. Data Conventions of Honeywell Mass-Storage Disk Devices

The flag is a one-character field. Five bits are significant. The remaining bit is not applicable here. See Figure 3-14.

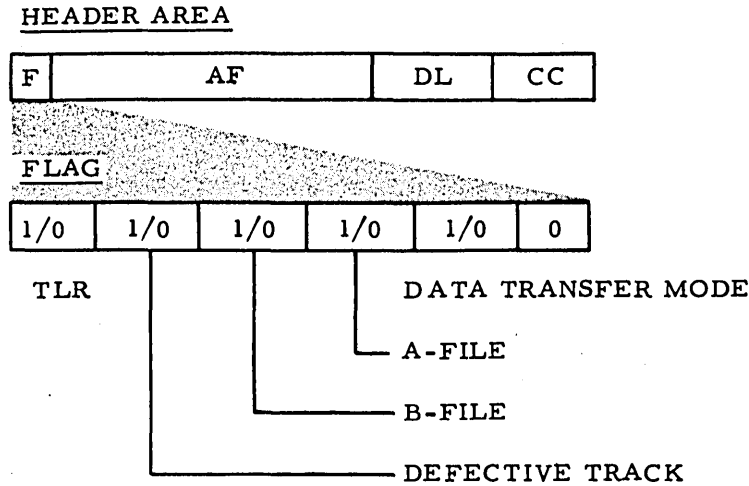


Figure 3-14. Flag Character Format

- Track-Linking Record (TLR) — The high-order bit of this six-bit character contains the value "0" for a data record or the value "1" for a track-linking record.
- Defective Track — This bit position is normally a "0." It is a "1" when the associated track has been designated as defective.
- A-File and B-File — These bits are used for file protection.
- Data Transfer Mode — This bit position is a "0" for six-bit transfer mode.

The address field consists of six characters. It specifies the cylinder, track, and record number of each record. See Figure 3-15.

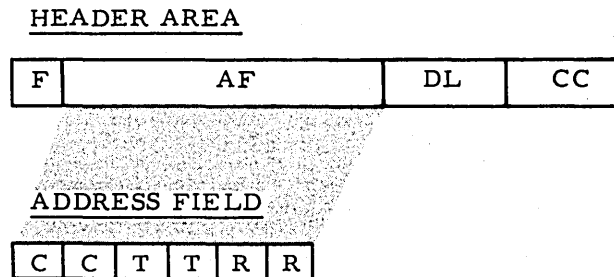


Figure 3-15. Address Field Format

- Cylinder Number (CC) — This two-character field specifies (in binary) a cylinder number within a device.
- Track Number (TT) — This two-character field specifies (in binary) the track number within a cylinder on which the record is situated.
- Record Number (RR) — This two-character field identifies the binary record number within a track.

The data length field of two characters specifies (in binary) the number of characters contained in the data area of the record, not including the checking code within that data area.

The checking code is a two-character field, which is automatically calculated and interpreted by the control for purposes of error detection. The checking code used with the 277/279 disk drives is a seven-byte field which is automatically calculated and interpreted by the control for the purpose of error detection and correction.

Data Area

The data area contains the information that is transferred to or from the central processor. The length of the data area (exclusive of check characters) is determined by the user and is specified in the data length of the header area for that record.

The data area is automatically stored in the following manner. If the total data area is less than 256 characters, it is stored in field 1 (see Figure 3-16) and two check characters are appended to the field. For example, the data area for 100 characters is 102 characters long: data field 1 contains 100 characters and a two-character checking code is appended to this field. With the 277/279 disk drives, the first data field can occupy 1 to 4,096 characters. Each of the other data fields, if any, must be 4,096 characters long. The check characters EDAC are seven bytes long and are appended to each data field by the control (see lower portion of Figure 3-17).

If the data length is greater than 256 characters, the overflow from one or more 256-character fields is stored in data field 1. Each of the data fields other than data field 1 must be 256 characters long. Two check characters are appended to each data field by the control unit. For example, the data area for 300 characters is 304 characters long. Data field 1 contains 44 characters, and data field 2 contains 256 characters. Two check characters are appended to each field, bringing the total to 304 positions.

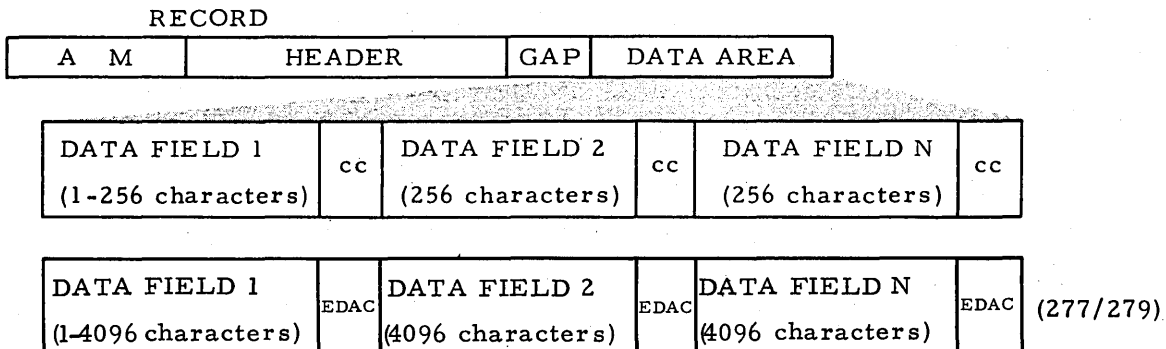


Figure 3-16. Data Area Format

### TRACK-LINKING RECORD

The basic function of the track-linking record is to allow contiguous processing of a series of records that extends from one track to another track on the same cylinder. The track-linking record can also be used to handle overflow records and for alternate track recording. The format of a track-linking record is identical to that of a normal record, except for the data area: the track-linking record data area must contain the address of the next record to be sought.

#### DATA AREA FORMAT

C	C	T	T	R	R
---	---	---	---	---	---

Figure 3-17. Track-Linking Record



SECTION IV  
ADDRESSING

BASIC CONCEPTS

The main memory storage locations that contain the instructions and data of a program are identified to the machine by their particular main memory addresses. Every character storage location in the main memory is directly addressable.

An instruction is stored in a field of from 1 to 12 characters, depending on the format of the instruction and the mode of address assembly (two-, three-, or four-character). Figure 4-1 illustrates how a typical seven-character Add instruction appears when stored in the main memory. (Recall that enclosing a character in a circle indicates that a word mark is associated with it.)

An instruction is addressed by specifying the op code (leftmost) location of the instruction. For instance, the address of the Add instruction in Figure 4-1 is 524. The machine reads an instruction from left to right until it senses a word mark. For example, the extraction of the Add instruction (Figure 4-1) is stopped by the word mark associated with the op code of the next instruction in sequence.

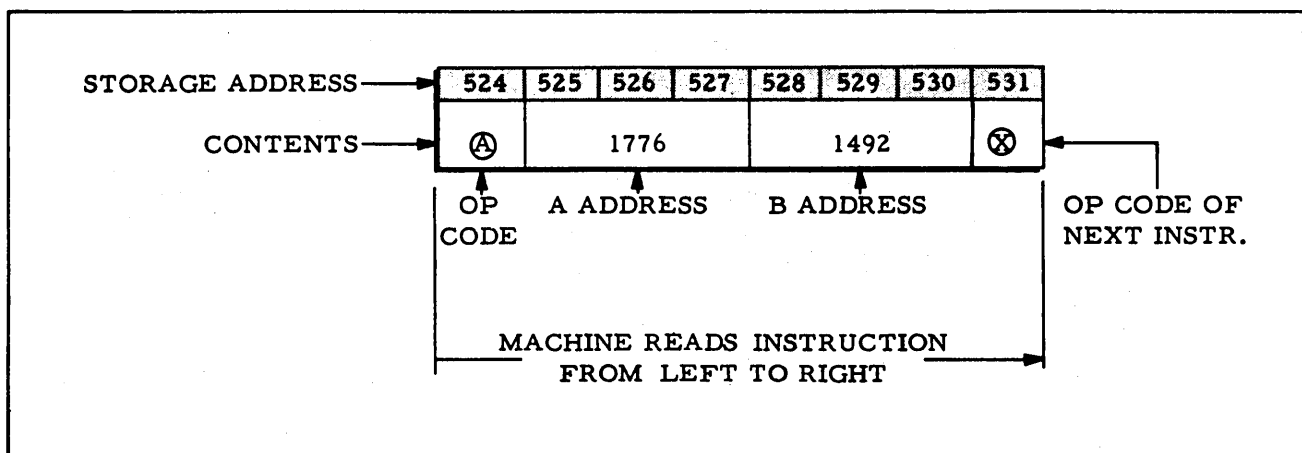
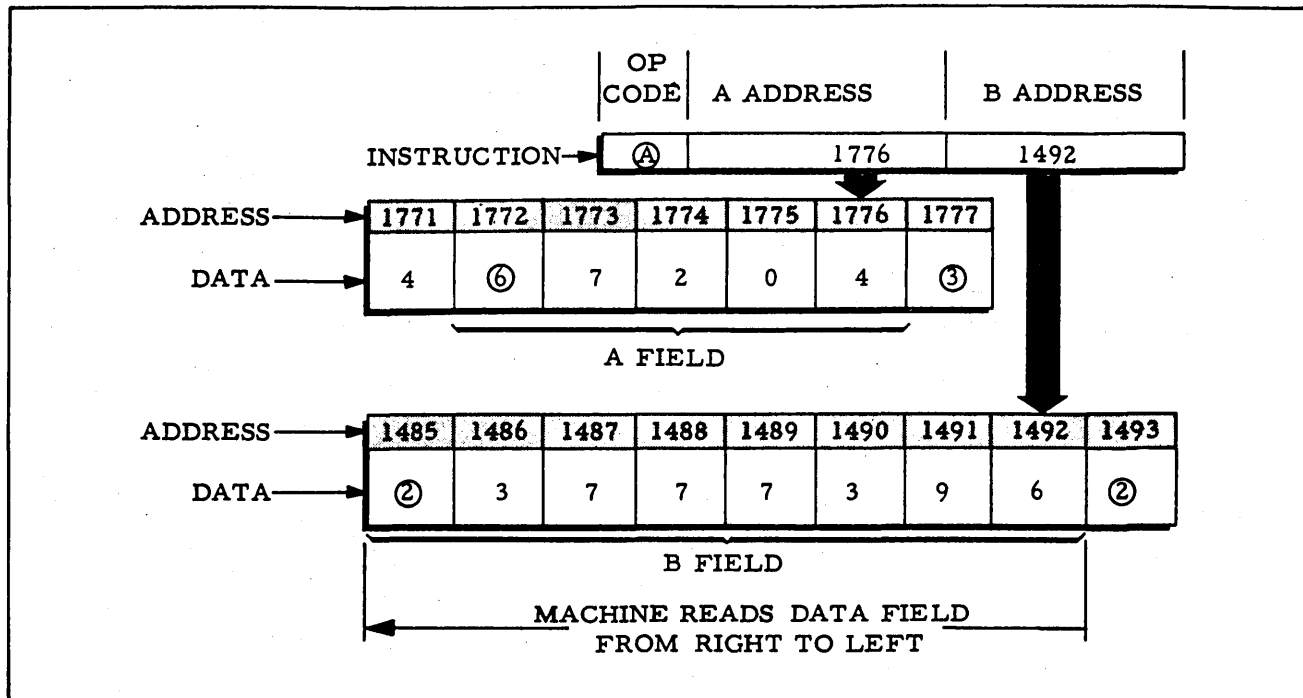


Figure 4-1. Typical Add Instruction

As mentioned in Section III, a data field is normally defined in the following manner: the leftmost location in the field is indicated by a word mark; the rightmost location is specified in the A or B address of an instruction. The machine reads a data field from right to left until it

senses the word mark associated with the leftmost character in the field.<sup>1</sup> For example, the A and B addresses in the instruction shown in Figure 4-1 could specify the data fields shown in Figure 4-2.<sup>2</sup>



The direction in which the machine reads any of the above-mentioned groups is compatible with the manner in which the contents of the group are manipulated. For instance, a field used in an arithmetic operation is read from right to left because such operations combine fields character by character, starting with the low-order or "units" position in each field. Similarly, an instruction is read from left to right because the machine must interpret the op code before it can manipulate the operand(s).

## REGISTERS USED IN ADDRESSING

The processing of a stored-program instruction consists of two phases: the retrieval (or "extraction") of the instruction from main memory storage, and the execution of the instruction. Six control memory registers are used to address the main memory during instruction processing. Four registers — SR, CSR, EIR, and IIR — are related to the sequential selection of instructions in a program; the other two registers — AAR and BAR — control the transfer of information from one storage location to another by containing the address portions of an instruction.

### SEQUENCE REGISTER (SR)

SR contains the address of the next sequential instruction character to be extracted from the memory during a program run. The contents of SR are incremented by one as each instruction character is extracted, so that SR contains the address of the next instruction's op code when one instruction has been completely extracted.

### CHANGE SEQUENCE REGISTER (CSR)

The address of an op code can be stored in CSR.<sup>1</sup> A Change Sequencing Mode instruction will interchange the contents of SR and CSR and thereby cause the program to branch to the instruction whose op code address was stored in CSR. At this point in the program CSR will contain the address of the op code following the Change Sequencing Mode instruction. In order to return to this op code (i. e., to the initial sequence of instructions), another Change Sequencing Mode instruction can be issued.

### EXTERNAL INTERRUPT REGISTER (EIR)

EIR, like CSR, can be used to store the address of an op code.<sup>1</sup> This address and the contents of SR will be interchanged automatically when an external interrupt signal is received. (Recall that an external interrupt signal can be generated by a peripheral control, by the control panel or console, or by the Monitor Call instruction.) In order to return to the normal sequence of instructions that was interrupted, a Resume Normal mode instruction can be issued.

---

<sup>1</sup> A Load Control Registers instruction can be used to store the desired op code address.



## INTERNAL INTERRUPT REGISTER (IIR)

The address of an op code can also be stored in IIR.<sup>1</sup> When Storage Protection is in effect, certain operations are considered as "violations" of storage protection (e.g., the attempt to initiate a data transfer from a peripheral control to a starting location in the protected memory area). An internal interrupt signal is generated when such a violation occurs, and the contents of IIR and SR are automatically interchanged. The Resume Normal Mode instruction is used to return to the interrupted program.

## A-ADDRESS REGISTER (AAR)

AAR normally contains the A-address portion of an instruction (i. e., the storage address of the rightmost character of the A-operand data). This address is loaded into AAR during the extraction phase of processing. In the execution of instructions whose operands are fields or rightmost-addressed fields or items, the contents of AAR are decremented by one as each character in the A field is manipulated.<sup>2</sup> The contents of AAR are incremented by one as each character in a record or leftmost-addressed field or item is extracted.<sup>3</sup>

## B-ADDRESS REGISTER (BAR)

Normally the B-address portion of an instruction is loaded into BAR during the extraction phase. During the execution of most instructions, the contents of BAR are decremented by one as each character in the B field is extracted.<sup>2</sup> If the B operand is a record or a leftmost-addressed item, the contents of BAR are incremented by one as each character is extracted.<sup>3</sup>

## SUMMARY

The foregoing information can be summarized as four easily remembered rules:

1. An instruction is read from left to right. As each character in the instruction is read, the contents of the sequence register are incremented by one.
2. A field is read from right to left.<sup>2</sup> As each character in a field is read, the contents of the corresponding address register are decremented by one.
3. A record is read from left to right.<sup>3</sup> As each character in a record is read, the contents of the corresponding current location counter are incremented by one.

---

<sup>1</sup> A Load Control Registers instruction can be used to store the desired op code address.

<sup>2</sup> A field can also be moved internally from left to right by means of the Extended Move (EXM) or Move or Scan (MOS) instructions (see Section VIII). In this case, the address register is incremented.

<sup>3</sup> A record can also be moved internally from right to left by means of the Extended Move or Move or Scan instructions. In this case, the address register is decremented.

4. An item can be read either from left to right or from right to left. As each character in an item is read, the contents of the corresponding address register are incremented by one if reading from left to right, or decremented by one if reading from right to left.

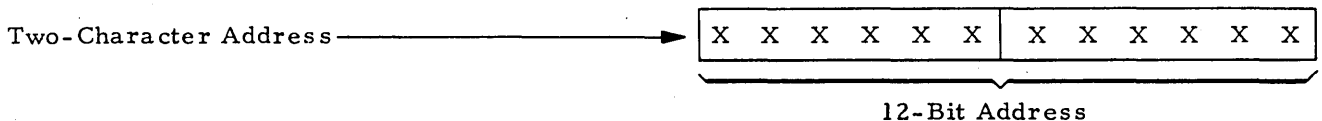
Recall that in the Type 2041, a control memory register is only as large as it need be to contain the largest main memory address in a user's processor (see Table 2-2), so that the size of the user's control registers ranges from 12 to 19 bits in length. The programmer should keep this fact in mind while reading the following description of addressing modes.

## ADDRESSING MODES

As stated at the beginning of this section, an instruction is stored in a field of from 1 to 12 characters, depending on the instruction's format and the programmed addressing mode. The op code is stored as a single six-bit character. Variant characters or I/O control characters, if any, are each stored as single characters. The number of character locations in which each address portion is stored depends on the addressing mode selected by the programmer. This selection is made by means of a Change Addressing Mode instruction with which the programmer specifies the two-, three-, or four-character addressing mode. A significant feature of the Series 2000 addressing technique is that the entire memory is directly addressable.

### TWO-CHARACTER ADDRESSING MODE

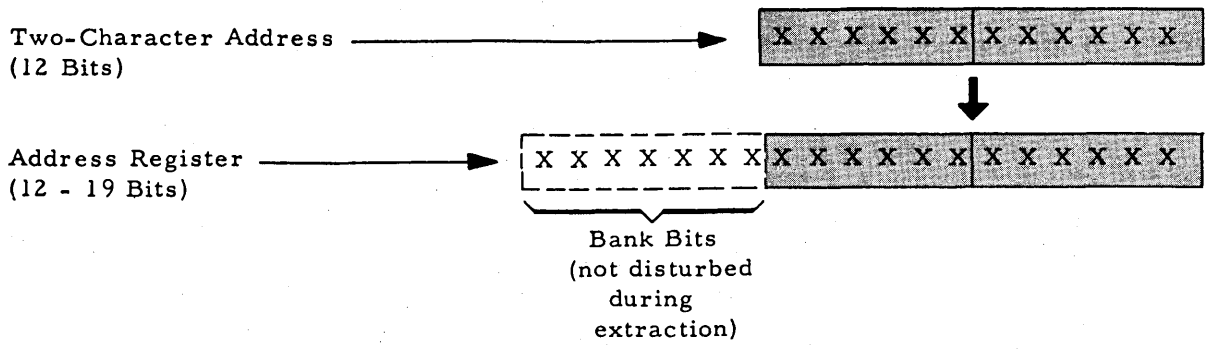
An operand address written in the two-character addressing mode is stored in two consecutive character locations in memory. The stored address (a continuous 12-bit binary number) represents the address of a main memory location in the range  $0 - 4,095_{10}$ .



During the extraction phase of instruction processing, the two-character address is placed in the rightmost 12 bit positions of the address register (AAR or BAR). Any bits in the register to the left of the two-character address are called "bank bits." Previous values in the bank bit positions of the register are not disturbed during instruction extraction.<sup>1</sup>

---

<sup>1</sup>The entire contents of an address register (bank bits + two-character address bits) are affected during the extraction of an instruction whose extraction path "duplicates A" (described in Section IV). Extraction of all other two-character addresses affects only the rightmost 12 bits.



When the instruction is executed, the entire contents of the address register are interpreted as the operand address. Previous values in the bank bit positions, not disturbed during the extraction phase, are used to form the address of the operand during the execution phase. Thus, the bank bit values imply a base address to which the 12-bit address is added to form the actual operand address. If the bank bit values are all 0's, the 12-bit address is the actual operand address.

For example, a two-character A address specifying location  $4,000_{10}$  is extracted and placed in AAR. The second bank bit in AAR (bit position 14) contains a residual value of "1", representing a base address of  $8,192_{10}$ . When the instruction is executed, the entire contents of AAR ( $8,192_{10} + 4,000_{10}$ ) specify the address of the A operand — location  $12,192_{10}$ .

As the contents of the address register are incremented or decremented during "internal" execution, bank bits are not disturbed.<sup>1</sup> If the 12-bit address in the rightmost positions of the register becomes zero, a borrow from the first bank bit does not occur. Thus, the portion of memory which is addressable by a two-character address is the 4,096-character "bank" specified by the base address.

Indexed and indirect addressing (see below) cannot be performed in the two-character addressing mode.

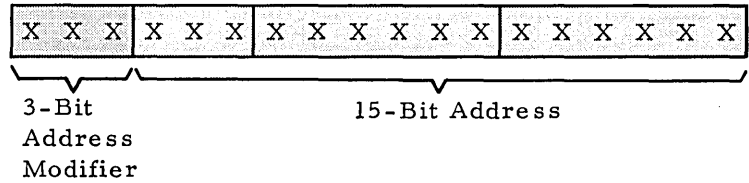
THREE-CHARACTER ADDRESSING MODE

An operand address written in the three-character addressing mode is stored in three consecutive character locations of the memory. The rightmost 15 bits of the stored address represent the address of a main memory location in the range  $0 - 32,767_{10}$ . The leftmost three

<sup>1</sup>"Internal execution" is defined as the incrementing or decrementing of address register contents during memory cycles allocated to the central processor. When peripheral transfer operations are performed, using memory cycles allocated to read/write channels, incrementing and decrementing of address register contents affect all bits of the register. Thus, addressing during peripheral transfer operations is continuous throughout the memory.

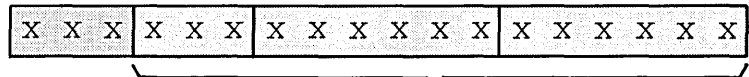
bits, referred to as the "address modifier," specify whether the address is direct, indirect, or indexed (see "Address Modification").

3-Character Address

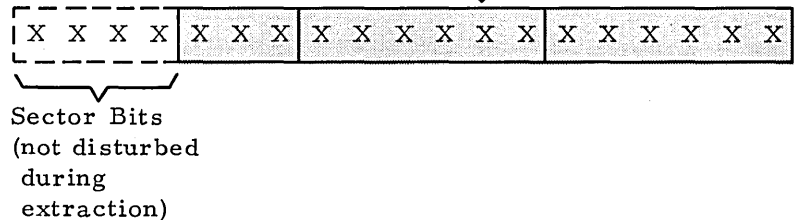


During the extraction phase, the 15-bit address is placed in the rightmost bit positions of the operand address register. Any bits in the register to the left of these bit positions are called "sector bits." Previous values in the sector bit positions of the register are not disturbed during instruction extraction.<sup>1</sup>

3-Character Address  
(15 Address Bits)



Address Register  
(15 - 19 Bits)



When the instruction is executed, the entire contents of the address register are interpreted as the operand address. Previous values in the sector bit positions, not disturbed during the extraction phase, are used to form the address of the operand during the execution phase. Thus, the sector bit values imply a base address to which the 15-bit address is added to form the actual operand address. If the sector bit values are all 0's, the 15-bit address is the operand address.

For example, a three-character A address specifying location  $12,000_{10}$  is extracted and placed in AAR. The first sector bit in AAR (bit position 16) contains the value "1", representing a base address of  $32,768_{10}$ . When the instruction is executed, the entire contents of AAR ( $32,768_{10} + 12,000_{10}$ ) specify the address of the A operand — location  $44,768_{10}$ .

As the contents of the address registers are incremented or decremented during "internal" execution, sector bits are not disturbed. If the 15-bit address in the rightmost locations of the address register becomes zero, a borrow from the first sector bit does not occur. Thus, the

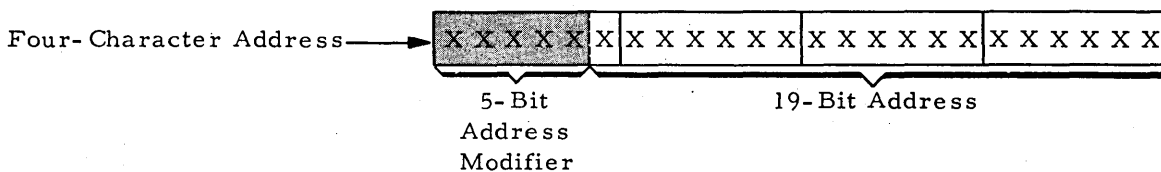
<sup>1</sup>The entire contents of an address register (sector bits + 15-bit address) are affected during the extraction of an instruction whose extraction path "duplicates A" (described in Section IV). Extraction of all other three-character addresses affects only the rightmost 15 bits in the register.

largest portion of memory which is addressable by a three-character address is the 32,768-character "sector" specified by the base address.

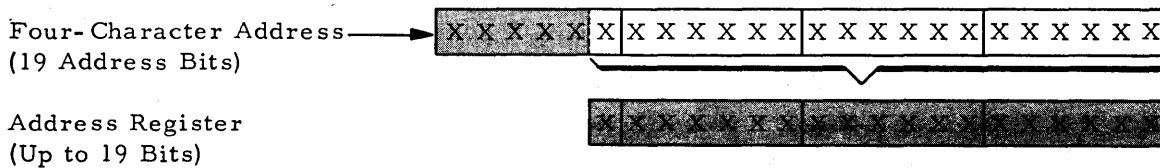
Addressing is continuous throughout the entire memory when a peripheral transfer operation is performed, as in the two-character mode.

#### FOUR-CHARACTER ADDRESSING MODE

An operand address written in the four-character addressing mode is stored in four consecutive character locations. The rightmost 19 bits represent a main memory address in the range 0 - 524,287<sub>10</sub>. The leftmost five bits — the "address modifier" — specify whether the address is direct, indirect, or indexed (see "Address Modification," below).



The 19-bit address is placed in the address register during the extraction phase. Thus, the entire contents of the address register are affected during the extraction of a four-character address.



The entire contents of the register are interpreted as the operand address when the instruction is executed. As the contents of the operand address registers (AAR and BAR) are incremented or decremented during execution, all bits in the register are affected. Thus, addressing is continuous throughout the entire range of available memory (up to 524,288 locations) in the four-character addressing mode.

#### ADDRESS MODIFICATION

Indirect and indexed addressing can be used to modify three- or four-character addresses. These addressing forms are represented by the configuration of the "address modifier" as described below and are interpreted by the processor during the extraction phase.

## INDEX REGISTERS

Index registers are used to store values to be used for address modification during instruction execution. A Series 2000 processor can contain up to 120 index registers. Figure 4-3 shows the memory areas utilized by the largest possible complement of index registers in a Series 2000 memory. The portion of a processor's index register complement usable by a program at any given time varies with the program's location in main memory and the addressing mode in use. Thirty index registers are simultaneously available to a program.

LOCATION 0			
X1-X15	X1-X6	X1-X6	X1-X6
Sector 0	Sector 1	Sector 2	Sector 3
X1-X6	X1-X6	X1-X6	X1-X6
Sector 4	Sector 5	Sector 6	Sector 7
X1-X6	X1-X6	X1-X6	X1-X6
Sector 8	Sector 9	Sector 10	Sector 11
X1-X6	X1-X6	X1-X6	X1-X6
Sector 12	Y1-Y15 <sup>(a)</sup>	Sector 14	Sector 15
	Sector 13		
LOCATION 524, 287			
<sup>a</sup> Registers Y1-Y15 can be positioned, under program control, in the first 60 locations of any 4,096-character bank of memory. If these registers are positioned in the first bank of a 32,768-character sector, they replace the group of six index registers in that sector.			

Figure 4-3. Series 2000 Index Register Map

### Index Register Map (Figure 4-3)

Registers X1-X6 are available to instructions executed in the three-character mode. These registers are located in the first 25 positions (locations 0 through 24) of the 32,768-character sector in which the instruction is stored. Since there can be as many as sixteen 32,768-character sectors in a Series 2000 main memory, up to 96 index registers are supplied for use in a three-character addressing mode.

Index Registers X1-X15, located in the first 60 character positions of memory, are available to instructions executed in the four-character addressing mode. The placement of these registers is independent of the location of the instruction whose address(es) is indexed. Registers Y1-Y15, located in the first 60 positions of a "protected" memory area, are available to all programs operating in the four-character addressing mode.<sup>1</sup> The specific bank at which the protected memory area begins is specified by use of the Load Index/Barricade Register instruction (see Section VIII).

### THREE-CHARACTER ADDRESS

The address modifier of a three-character address (i.e., the leftmost three bits of the stored address) specifies whether the address is direct (000), indirect (111), or indexed (001 through 110).

#### Indirect Addressing

In previous examples and illustrations in this section, an address portion of an instruction always specifies the address of a data field in the main memory. This manner of addressing an operand is commonly referred to as direct, or "first-level," addressing. In some instances, instead of specifying the location of a data field directly, it is more useful to be able to specify the storage location of another address, which in turn specifies the location of the desired data field. This manner of locating an operand is referred to as indirect, or "second-level."

A three-character indirect address is specified by an address modifier of all "1's" and refers to the leftmost storage location of another main memory address. The referenced address can itself be direct, indirect, or indexed as specified by its address modifier. Thus, an indirect address can specify another indirect address, and so on through any number of levels, or it can specify an indexed address. The method of coding an indirect address is illustrated in Section V.

Figure 4-4 shows the extraction of an Add instruction in which indirect addressing is specified in the A-address and direct addressing is specified in the B-address. Note that the A-address (indirect) references the leftmost location of another main memory address. This address, in turn, specifies the location of the rightmost character in the A-field. Note further that if the address modifier of location 1027 were not "000", the remainder of the stored address would be interpreted as an indexed or indirect address.

---

<sup>1</sup>Programs operating in the unprotected portion of memory can read the contents of Y1-Y15 but cannot write into these registers.

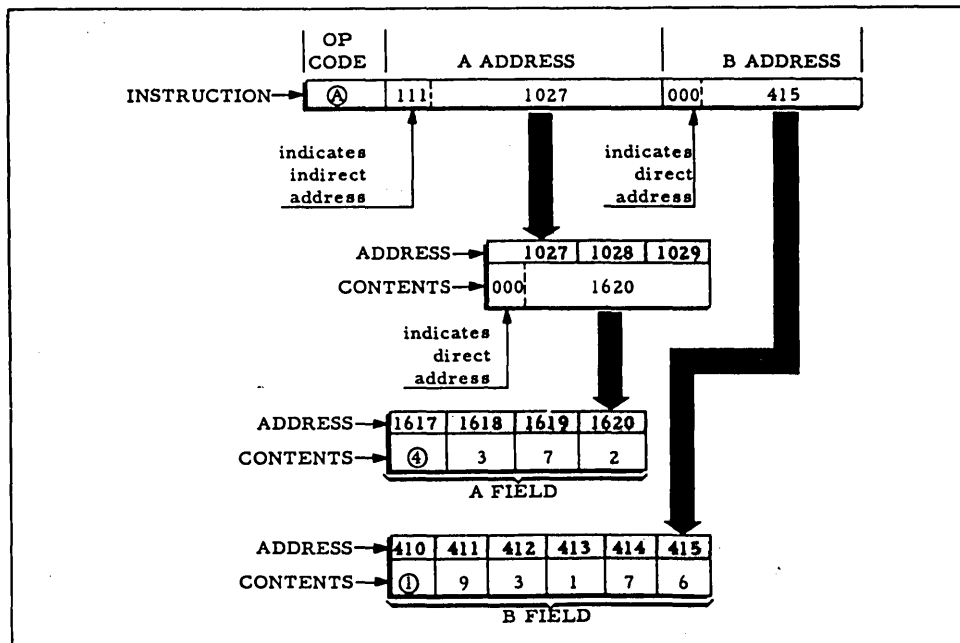


Figure 4-4. Extraction of Three-Character Indirect Address

### Indexed Addressing

When indexed addressing is performed in the three-character mode, the rightmost 15-bit contents of an index register are automatically added (in binary) to the 15-bit address field in an instruction. Three variables must be defined in any indexing operation: (1) the index register to be used, (2) the address to be modified, and (3) the factor (referred to as an augment) to be added to the address. The index register to be used is specified in the address modifier of an address field (see Table 4-1). The address to be modified can be stored in the same address field or it can be stored in the designated index register. If the address to be modified is stored in an address field, the augment is stored in the designated index register and vice versa.

Table 4-1. Index Register Addresses in Three-Character Addressing Mode

Index Register	Address Modifier	Storage Field	Address
X1	001	2 - 4 (+n)	4 (+n)
X2	010	6 - 8 (+n)	8 (+n)
X3	011	10 - 12 (+n)	12 (+n)
X4	100	14 - 16 (+n)	16 (+n)
X5	101	18 - 20 (+n)	20 (+n)
X6	110	22 - 24 (+n)	24 (+n)

n = first location of the 32,768-character sector in which the instruction is stored.

The modification of an address occurs in its respective address register. For instance if the B-address portion of an instruction is indexed, the modification is performed in BAR. This means that neither the original instruction stored in the main memory nor the contents of the index register is altered in any way.



Normal programming, such as a load or a move operation, can be used to store a value in an index register. Similarly, the contents of an index register can be changed by using an instruction such as Binary Add or Binary Subtract. Note that since the index registers are located in the main memory, they can be used as normal storage locations when they are not being used for indexing operations.

Figure 4-5 illustrates how the Add instruction would be extracted if indexed addressing were specified in the A-address portion of the instruction. The method of coding an indexed address is illustrated in Section V.

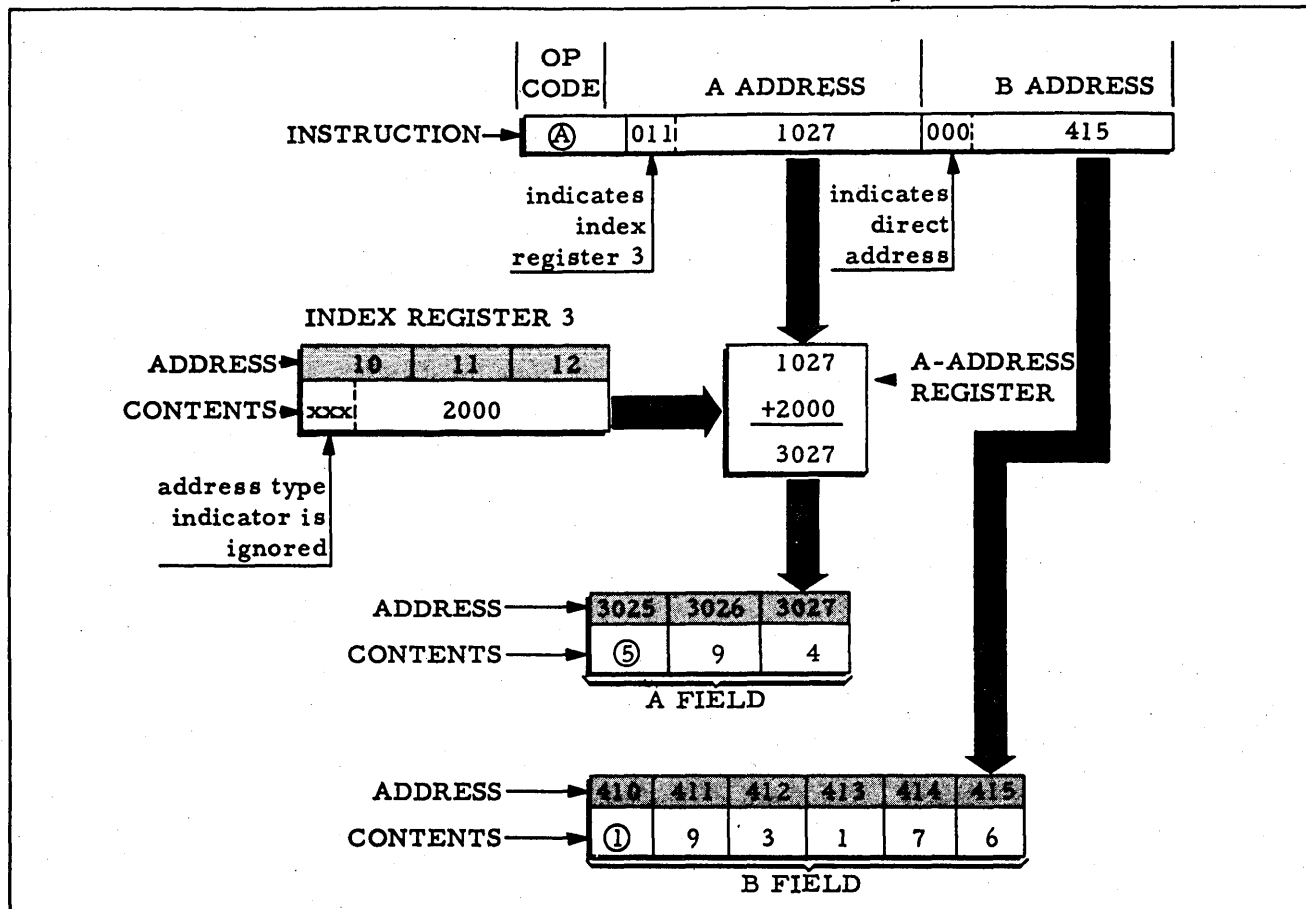


Figure 4-5. Extraction of Indexed Address in Three-Character Mode

#### FOUR-CHARACTER ADDRESS

The address modifier in a four-character address consists of the leftmost five bits of the address. The configuration of these bits specifies whether the address is direct (00000), indirect (10000), or indexed (00001 through 11111, excluding 10000).

#### Indirect Addressing

Indirect addressing in the four-character addressing mode is performed similarly to that in the three character mode, except that:

1. a five-bit address modifier whose configuration is 10000 specifies indirect addressing; and
2. A four-character address is extracted.

The method of coding a four-character indirect address in Easycoder assembly language is identical to that used for a three-character indirect address (see Section V).

### Indexed Addressing

Four-character indexed addresses to be modified by index registers X1 through X15 are specified by an address modifier whose configuration is 00001 through 01111, respectively. Index registers Y1 through Y15, when present, are specified by the configurations 10001 through 11111 (see Table 4-2). Register locations are shown in Figure 4-3.

Table 4-2. Index Register Addresses in Four-Character Addressing Mode

Index Register	Address Modifier	Storage Field	Address
X1	00001	1-4	4
X2	00010	5-8	8
X3	00011	9-12	12
X4	00100	13-16	16
X5	00101	17-20	20
X6	00110	21-24	24
X7	00111	25-28	28
X8	01000	29-32	32
X9	01001	33-36	36
X10	01010	37-40	40
X11	01011	41-44	44
X12	01100	45-48	48
X13	01101	49-52	52
X14	01110	53-56	56
X15	01111	57-60	60
Y1	10001	Same as above, only relative to the 4,096- character memory bank designated by the Load Index/Barricade Register instruction	
Y2	10010		
Y3	10011		
Y4	10100		
Y5	10101		
Y6	10110		
Y7	10111		
Y8	11000		
Y9	11001		
Y10	11010		
Y11	11011		
Y12	11100		
Y13	11101		
Y14	11110		
Y15	11111		

When indexed addressing is performed in the four-character mode, the contents of the specified index register are added (in binary) to the address field of the instruction. However, only the number of active address bits of the index register and the address field are combined (i. e., only the number of bits which are required to address the entire memory of the user's processor). In a single-character processor, the number of active address bits corresponds to the size of a control memory register (see Table 4-3); in a multicharacter processor, all control register bits are active, regardless of main memory size.



## TREATMENT OF ADDRESSES LARGER THAN A MEMORY'S MAXIMUM ADDRESS

In all processors except a multicharacter processor with maximum memory, it is possible to specify in instructions direct addresses that are larger than the address of the processor's highest memory location.

Likewise, it is possible in any Series 2000 processor, by the use of indexed addressing, to specify addresses and address modifiers whose sums are potentially greater than the address of the memory's highest location. For example, consider the case where, in a machine having a 49,152-character memory, an instruction contains the address 49,000 and the address is indexed using a register which contains the value 1,000. Obviously, the sum of 49,000 and 1,000 is greater than the memory's highest address, 49,151.

Situations such as the ones just cited are handled differently, depending upon the relationship between the potential address and the memory size involved and whether or not the Storage Protect feature is in effect. In particular, such situations can be categorized according to whether the potential address is larger or smaller than the range of addresses representable by active address-register bits.

### Potential Addresses Within Address Register Range

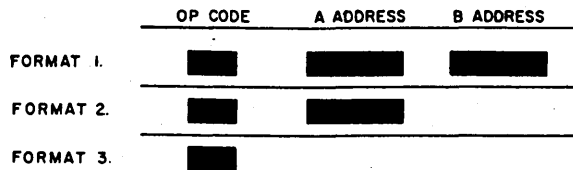
In a multicharacter processor without storage protection in effect, encountering a simple, direct address, or the potential sum of an indexed address and index register contents, which lies between the address of the highest actual memory location and the address registers' upper limit, causes the processor to stop. Results are unspecified for the other Series 2000 processors. Any Series 2000 processor with Storage Protection in effect, upon encountering an address of the type described above, will perform the following actions: the internal interrupt (II) address violation indicator is set, the instruction is terminated prematurely, and an internal interrupt is generated.

### Potential Addresses Outside Address Register Range

In any Series 2000 processor, if a simple direct address, or the potential sum of an indexed address and index register contents, is greater than the largest address representable by active address-register bits, the resultant address is formed modulo the number of locations addressable with the active address bits; i.e., a memory "wraparound" occurs. For example, in a 49K Model 2040, a total of 65,536 locations can be addressed by 16 active address bits. If, in such a machine, an address of 48,000 is indexed by the value 27,000, the resultant effective address will be  $48,000 + 27,000 - 65,536$ , or 9464.

## EXPLICIT ADDRESSING, IMPLICIT ADDRESSING, AND CHAINING

Consider the three instruction formats illustrated below.



Format 1 corresponds to the instructions used in the preceding illustrations. The significant feature of this format is that the addresses of both the A and the B data fields are explicitly specified in the instruction. For this reason the data fields are said to be "explicitly addressed." In general, whenever the programmer writes the address of a data field on his coding sheet, he is explicitly addressing that data field (see Figure 4-7).

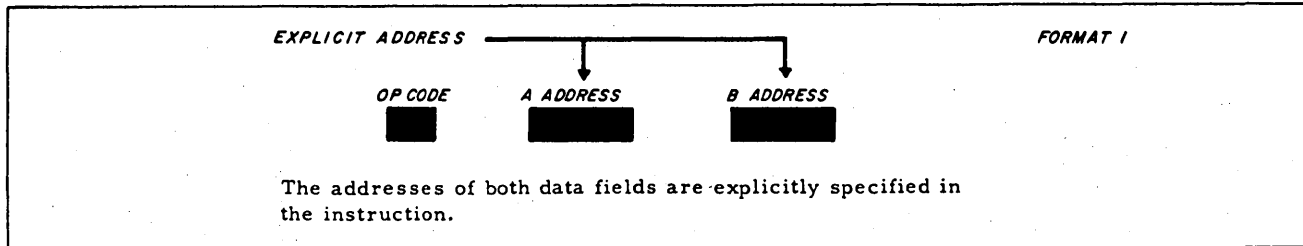


Figure 4-7. Series 2000 Instruction Format 1

Format 2 has two possible interpretations (see Figure 4-8):

1. Ten Series 2000 instructions coded in format 2 cause the A-address to be loaded into both AAR and BAR.<sup>1</sup> Thus, although the B-address portion of the instruction is omitted, the B-field is explicitly addressed by the A-address portion. The extraction path of these instructions is said to "duplicate A" (see Appendix C), since the contents of AAR are duplicated in BAR.
2. The A-address of 19 instructions is loaded into AAR only, leaving BAR undisturbed. An omitted B address in any of these instructions implies that the previous contents of BAR will be used as the address of the B field. For this reason the B-field is said to be "implicitly addressed," and the extraction path of these instructions "preserves B" (see Appendix C).

<sup>1</sup>The entire contents of AAR are loaded into BAR during extraction, so that all bit positions in BAR are identical to those in AAR. Recall that this is the only operation that affects bank bits and sector bits in two-character mode and sector bits in three-character mode.

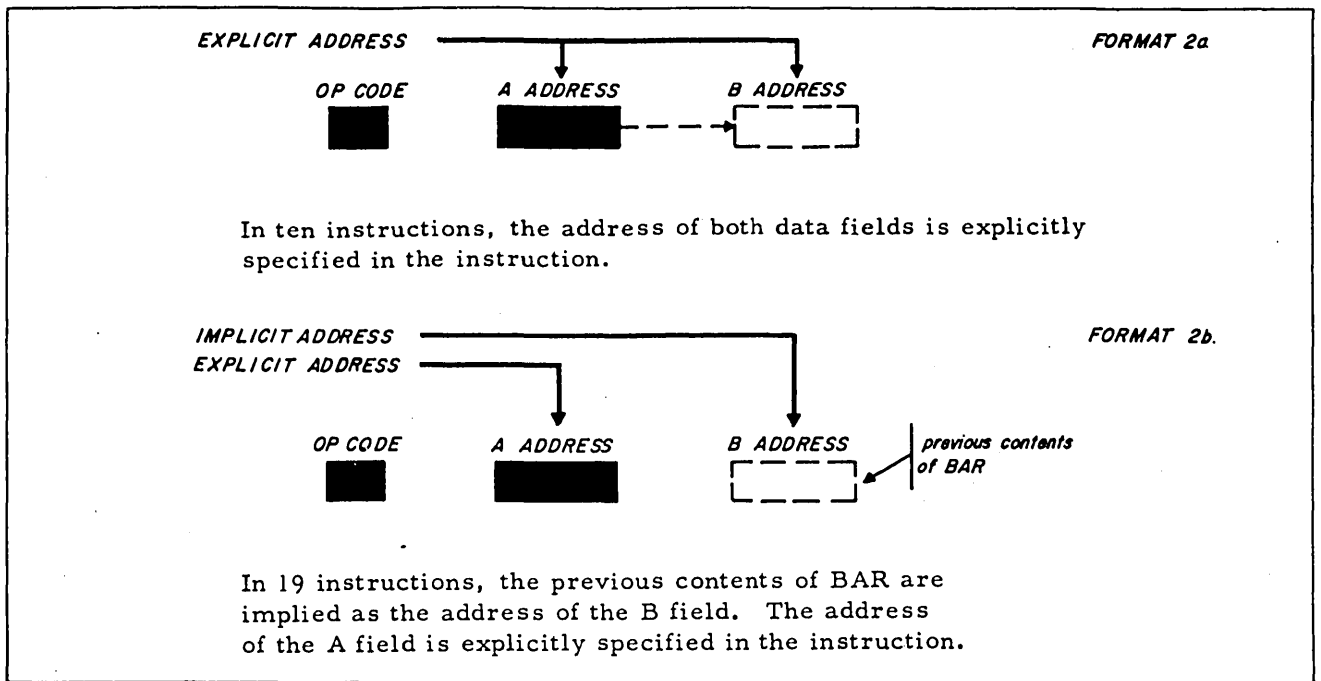


Figure 4-8. Series 2000 Instruction Format 2

In format 3, both data fields are implicitly addressed. The previous contents of AAR are used as the address of the A field, and the previous contents of BAR are used as the address of the B field (see Figure 4-9).

Implicit addressing is extremely useful in situations where it is desired to perform a series of operations on data fields that are in consecutive storage locations. The use of implicit addressing reduces both the time required to perform the operations and the number of memory locations required to store the instructions.

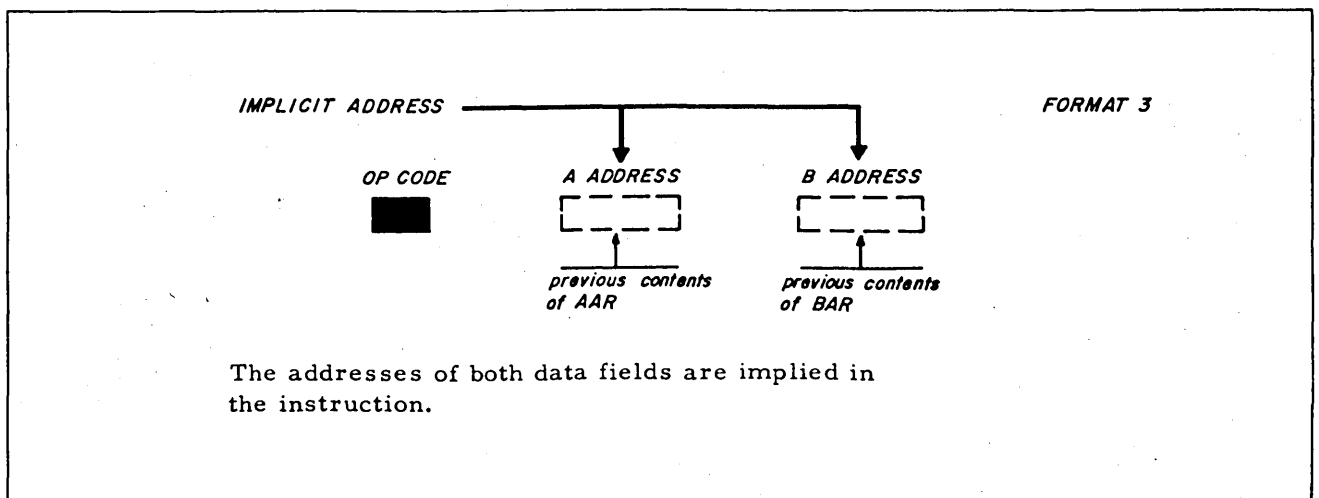


Figure 4-9. Series 2000 Instruction Format 3

As an example, assume that three 10-character fields stored in sequence are to be added to three other sequential fields. First, examine how this operation would be performed using explicit addressing. Upon completion of the first instruction, AAR contains 890 and BAR contains 690. These are the same values that appear in the A- and B-address portions of the second

(A)	900	700
(A)	890	690
(A)	880	680

instruction. Similarly, upon completion of the second instruction, AAR and BAR contain 880 and 680 — the A and B addresses of the third instruction. Since in each case AAR and BAR contain the addresses used in the next instruction, it is unnecessary to write these addresses in the instruction. In other words, this operation could be performed using implicit addressing in the second and third instructions.

(A)	900	700
(A)		
(A)		

Connecting instructions together so that the contents of AAR, BAR, and the variant register (see below) at the conclusion of one instruction satisfy the requirements of the next instruction is called "chaining." Using explicit addressing in the three-character addressing mode, 21 storage locations are required to store the instructions above and the operation takes 187 microseconds to complete on a Type 2041 processor. If the instructions were "chained," nine storage locations would be used and 168 microseconds would be required to complete the operation.

Instructions which require a variant character can also be chained by using the previous contents of the variant register. (The variant register is a single-character, internal register into which the variant character of an instruction is loaded during extraction.) The extent of chaining variant characters (i. e., the number of acceptable instruction formats in which the previous contents of the variant register can be used) varies with the processor being used.

Variant characters can be chained by an instruction coded in any format (i. e., format 1, 2, or 3). The previous contents of the variant register are not normally distributed by the processing of an instruction which does not contain a variant character (see the instruction Branch, Move Characters and Edit, and Move and Translate for exceptions).

Chaining is not limited to sequential operations having the same op code. The only condition that must be met is that an instruction must leave the contents of AAR, BAR, and, if required, the variant register such that they satisfy the addressing requirements of the next instruction in sequence.

To enable the programmer to chain instructions wherever possible, the description of each instruction (see Section VIII) includes a table showing the contents of the address registers after the instruction has been executed. Also, Appendix C denotes whether each instruction in the machine complement can or cannot be chained.





SECTION V  
EASYCODER PROGRAMMING

INTRODUCTION

The preparation of Series 2000 programs is greatly simplified by the use of EasyCoder — a concise, easy-to-use programming system. Specifically, EasyCoder relieves the programmer of many time-consuming duties associated with writing a program in actual machine language. It makes it unnecessary, for example, to maintain a careful record of the storage address assigned to each instruction. In addition, it allows the programmer to employ meaningful symbolic tags (e.g., TAX, FICA, and TOTAL) rather than absolute memory addresses to specify data. In situations where a stored program must be relocated or modified, EasyCoder can be used to perform the required alterations automatically.

EasyCoder includes a number of assembly systems; these systems are:

- EasyCoder A: Part of the Series 200/Basic Programming System. EasyCoder A operates in a system having a minimum main memory size of 4,096 characters. (Additional memory, however, may be used to advantage.) For additional information refer to EasyCoder A Assembly System (Order No. BC28).

NOTE: A counterpart of EasyCoder A — EasyCoder A (P) — is available for use in a paper tape environment. The main memory requirements are identical to those of EasyCoder A. See EasyCoder A (P) Assembly System (Order No. BD42) for more information.

- EasyCoder B: Also part of the Series 200/Basic Programming System. EasyCoder B operates in a system having a minimum main memory size of 8,192 characters. (Additional memory may be used to advantage, however.) See EasyCoder B Assembly System (Order No. BA08) for additional information.
- EasyCoder C: Part of the Series 200/Operating System - Mod 1. EasyCoder C operates in a system having a minimum of 12,288 characters of main memory. (Additional memory, however, may be used to advantage.) For additional information refer to EasyCoder Assemblers C and D (Order No. BA26).
- EasyCoder D: Part of the Series 200/Operating System - Mod 1. EasyCoder D operates in a system having a minimum of 16,384 characters of main memory. (Additional memory however, may be used to advantage.) For additional information see EasyCoder Assemblers C and D (Order No. BA26).

- OS/2000 Easycoder: Part of Series 200/2000 Operating System/2000. OS/2000 Easycoder operates in an OS/2000 partition having a minimum of 32K and a maximum of 256K memory. OS/2000 Easycoder implements operation codes with implied variants in addition to the standard instruction set. For additional information refer to OS/2000 Easycoder Assembler, (Order No. AH31).

Each assembly system includes two basic elements: the Easycoder symbolic language and an Easycoder Assembler. The Easycoder language is used to write the symbolic program (the source program) while the assembler is the programming element that translates the source program into the actual machine-language program (the object program).

To prepare a program in Easycoder symbolic language, the programmer uses an Easy-coder Coding Form (see Figure 5-5) and enters each symbolic instruction or definition on a separate line. As a general rule, the instructions are written in the order in which they are to be executed. (However, the instructions must be in the proper sequence prior to assembly.) After the symbolic program has been written, each line of symbolic coding is punched into a separate source-program card. These cards are the input data which will be processed by an Easycoder assembler.

The assembler accepts the source-program cards and automatically produces a corresponding machine-language object program. It converts mnemonic operation codes into machine language codes, assigns absolute storage addresses to instructions and symbolic operand references, and completely assembles the final program, storing it on punched cards, disk units, or magnetic tape. Another output of the assembler may be a complete printed summary of the symbolic source program and the corresponding machine-language entries. Figure 5-1 illustrates the relationship of the source program, assembler and object program.

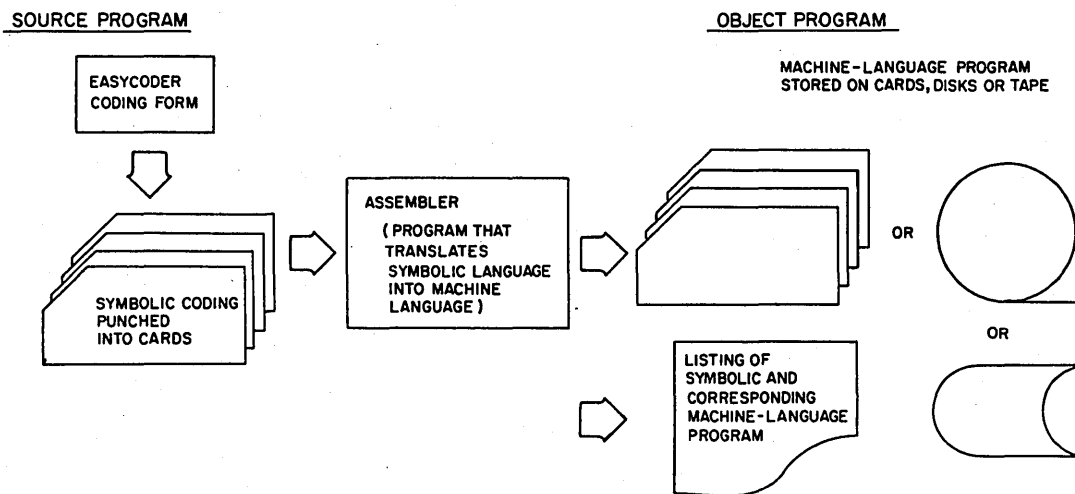


Figure 5-1. Relationship of Source Program, Assembler, and Object Program

## THE SYMBOLIC LANGUAGE

The Easycoder symbolic language is composed of a set of mnemonic operation codes and a set of rules for defining memory areas, addressing operands, and entering constants. The mnemonic operation codes are predefined abbreviations for machine-language operation codes and, in general, provide an easily remembered description of each instruction. For example, SI is the Easycoder mnemonic for the Set Item Mark instruction, and BCC is the mnemonic for the Branch on Character Condition instruction. The set of rules includes special mnemonics for defining work areas in the main memory and for defining programmer-specified constants.

The statements used in writing an Easycoder program can be classified into three groups:

1. Data formatting statements make it possible to reserve areas and store constants without regard to their actual locations in memory. Data formatting statements are described in Section VI.
2. Assembly control statements are used by the programmer to control the assembly of his program. Assembly control statements are described in Section VII.
3. Data processing statements are the actual machine instructions to be executed in the object program. Section VIII contains a description of the data processing statements employed by Series 2000 Central Processors.

## THE ASSEMBLERS

The assembler element of each Easycoder assembly system translates the symbolic source program (written on the Easycoder Coding Form and subsequently punched into a source-program card deck) into machine-language entries, placing the resultant object program on either punched cards or magnetic tape. In addition to the object-program output, the assembler may also produce a printed listing containing the symbolic source program and the corresponding object-program entries (see Figures 5-2, 5-3, and 5-4).

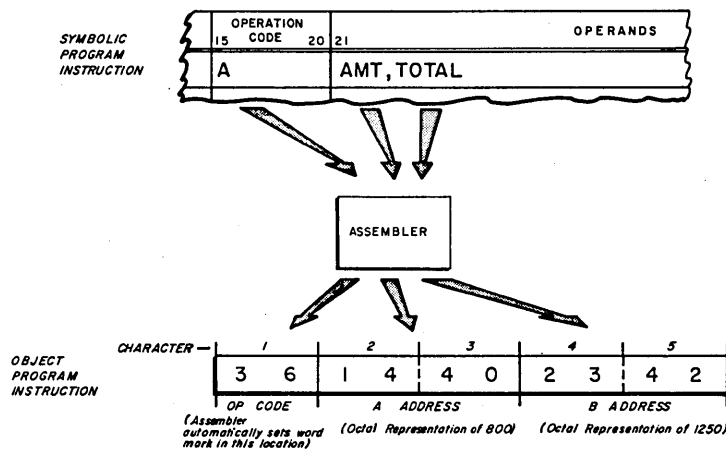


Figure 5-2. Two-Character Address Assembly

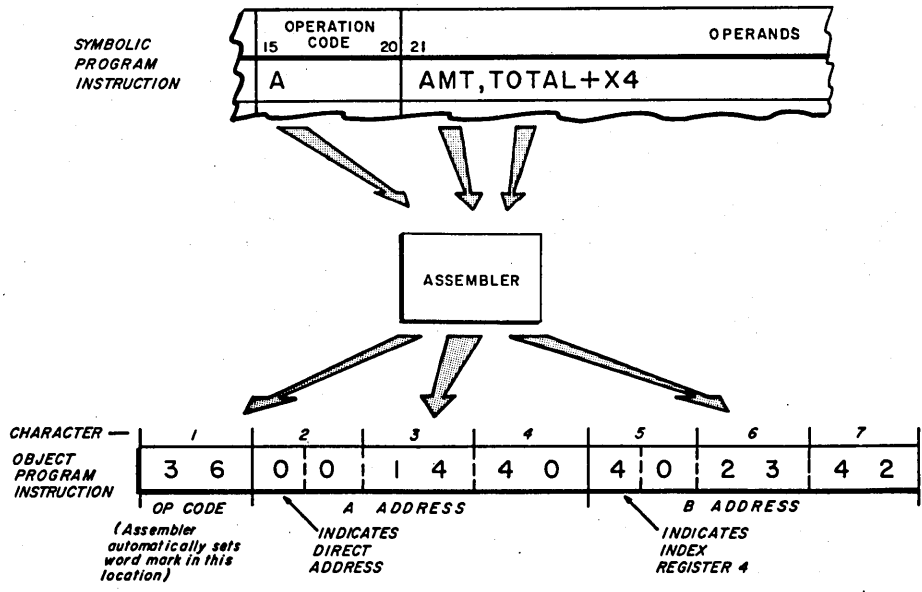


Figure 5-3. Three-Character Address Assembly

Figure 5-2 illustrates how an assembler assembles an object-program instruction using 2-character address assembly. Assume that the tag AMT is assigned to memory location 800 and that the tag TOTAL is assigned to memory location 1250. Figure 5-3 shows how the assembler assembles an object-program instruction using 3-character address assembly. Four-character addresses are assembled as shown in Figure 5-4. Assume that, in Figures 5-3 and 5-4, the tags are assigned the same values as in Figure 5-2.

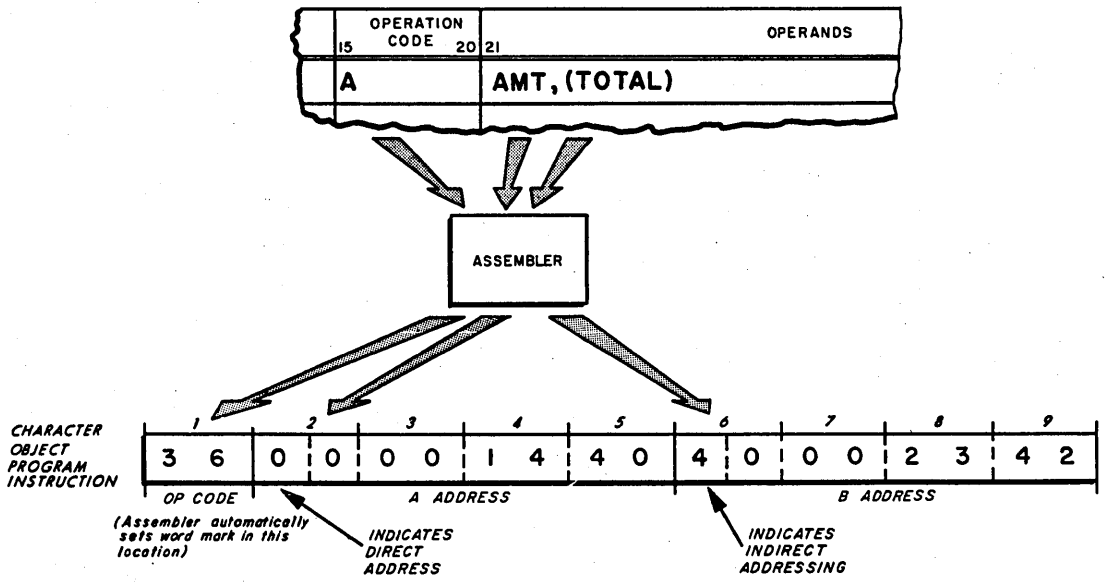


Figure 5-4. Four-Character Address Assembly

# CODING FORM

Programs are written on the EasyCoder Coding Form (Figure 5-5). This form is composed of fixed-format fields for coding such entries as card number, location, and operation code, and a variable-format field for operand addresses and comments. The numbers associated with each subdivision, or field, on the coding form indicate the card columns into which the characters written by the programmer are to be punched.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	LOCATION	OPERATION CODE	OPERANDS	
			1-14	15-80
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				

1645 REV 0

Figure 5-5. EasyCoder Coding Form

CARD NUMBER (Card Columns 1-5)

This five-character field is divided into three parts: the first two characters are used for page numbering, the next two for line numbering, and the last character for insertions. The page entry provides the proper sequencing of coding forms. The line number entry is used for the sequential numbering of instructions on each coding form. The single-character insertion entry permits one or more lines of coding to be inserted between existing lines. For example, to insert a line of coding between lines 16 and 17 of page 8, the following coding should be used.

CARD NUMBER					TYPE
PAGE	LINE		INS		
1	2	3	4	5	
8	1	6			
8	1	6	5		
8	1	7			

NOTE: The number 5, which appears in column 5 above, is optional. An insertion may be made using any decimal, alphabetic, or special character. Provided that the characters are in ascending order of value (beginning with 0), multiple insertions may be made between any two instructions.

TYPE (Card Column 6)

For all instructions and constants, this column remains blank. However, the programmer can enter lines of descriptive information, called remarks lines, anywhere in the source program. Such a line, containing only descriptive data within columns 8 through 80, is identified by an asterisk (\*) in column 6. Information inserted in this manner, while it remains as part of the source program, does not appear in the object program; it does, however, appear in the program listing.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER		OPERANDS
1	2	3
1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33
34	35	36
37	38	39
40	41	42
43	44	45
46	47	48
49	50	51
52	53	54
55	56	57
58	59	60
61	62	63
64	65	66
67	68	69
70	71	72
73	74	75
76	77	78
79	80	81
82	83	84
85	86	87
88	89	90
91	92	93
94	95	96
97	98	99
100	101	102

1 \* SPECIFY CONTROL CONSTANTS

2

EasyCoder C, D, and OS/2000 Options

For EasyCoder C or D users, this column may also contain the letter T to designate a temporary remarks card, or the letter D to designate a data card. If the programmer wishes to enter remarks lines anywhere in the source program but does not want these remarks to become a permanent part of the source program, a T instead of an asterisk (\*) is placed in column 6. Remarks lines inserted in this manner are used only on the first assembly (i.e., when the program is being "inserted"), and are subsequently deleted from the symbolic program tape by the assembler. A temporary (like a permanent) remarks statement, while it appears in the program listing, does not appear in the object program.

A letter D in column 6 indicates a data card. All data cards must be contained in segments consisting only of data cards. In addition, any data card (or group of data cards) must be immediately preceded by a SEG card and immediately followed by either an EX, XFR, or END card. When a data card is encountered by an assembler, columns 8 through 80 are reproduced, unaltered, on the binary run tape or machine-language punched deck.

The TYPE column is also used by EasyCoder C, D, and OS/2000 for macro call statements. The letter C indicates a continued card; L indicates the last (and possibly only) card of the macro call statement.

## EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6 7	8	14 15	20 21	62 63	80
1		C		SAMPLE	A, B,	MACRO NAMED
2		C			C, D,	SAMPLE IS CALLED
3		L			E,	WITH 5 PARAMS
4						
5		L		SAMPLE	A, B, C, D, E,	EQUIV CALL
6						

### MARK (Card Column 7)

This field, used in conjunction with data formatting operations (described in Section VI), serves to set up required punctuation. Two sets of punctuation indicators are available; set I may be employed with all EasyCoder assembly systems (A, B, C, and D); set II, however, may only be used with EasyCoder C and D and OS/2000. Both punctuation sets are described below.

Set I, consisting of a blank ( $\Delta$ ), an L, and an R, establishes the position of the item mark when defining an item (see Table 5-1). Word marking for this first set depends upon the class of instruction, as determined by the contents of the op code field.

NOTE: When an L is used and the leftmost (high-order) character is automatically word marked, a record mark will result.

Table 5-1. Set I Punctuation Indicators

Column 7 Contents	Resultant Item Mark Setting	
	Leftmost (High-order) Character	Rightmost (Low-order) Character
$\Delta$	Blank	Blank
L	Item Mark	Blank
R	Blank	Item Mark

### EasyCoder C, D, and OS/2000 Options

Set II, designed for use with the EasyCoder Assemblers C, D, and OS/2000, can be employed in situations that warrant unusual punctuation requirements. With this set (listed in Table 5-2), any one punctuation indicator controls the complete punctuation setting for the particular instruction or constant; there is no implicit word mark as in the first set. In other words, this second set of punctuation is not dependent upon the class of instructions.



Table 5-2. Set II Punctuation Indicators (Easycoder C, D, and OS/2000)

Column 7 Contents	Resultant Punctuation Setting	
	Leftmost (High-order) Character	Rightmost (Low-order) Character
A	Word Mark	Blank
B	Item Mark	Blank
C	Record Mark	Blank
D	Blank	Word Mark
E	Blank	Item Mark
F	Blank	Record Mark
G	Item Mark	Item Mark
H	Item Mark	Word Mark
I	Item Mark	Record Mark
J	Word Mark	Item Mark
K	Word Mark	Word Mark
M	Word Mark	Record Mark
N	Blank	Blank
P	Record Mark	Word Mark
S	Record Mark	Item Mark
T	Record Mark	Record Mark

#### LOCATION (Card Columns 8-14)

The location field can contain an absolute memory address or a symbolic tag, or it can be left blank. An absolute memory address (expressed as a decimal number) specifies that the instruction or data will be stored in that location. No leading zeros are necessary when writing an absolute decimal number. Moreover, this type of entry does not affect the allocation of any subsequent instructions.

Symbolic tags provide simple, meaningful symbolic references for storage locations, constants, and instructions that are referenced elsewhere in the program. All symbolic tags written in the location field are assigned absolute addresses during assembly. When an entry is assigned a symbolic tag, the contents of the entry can then be referenced by that tag. This means that the programmer can reference data via a symbolic tag and need not be concerned with its actual main memory address. One to six characters make up a symbolic tag (Easycoder D, however, can process tags of up to ten characters in length; see "Easycoder D Options" below). These characters can be alphabetic (A to Z) or numeric (0 to 9); the first character of the tag, however, must be alphabetic.

If the location field entry is made beginning in column 8, the following rules apply:

1. An absolute memory address assigned to an instruction refers to the leftmost character in the instruction.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	M	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16 17 18 19 20 21 22
1					0	A	FICA, TAX	36, the octal value of an add is located in absolute 0; assuming two-character address assembly, the B-Address is located in absolute 3 and 4.

2. An absolute memory address assigned to a constant or reserved area refers to the rightmost character in the field.

5					1235	RESV	30	The reserved area includes absolute 1206 through absolute 1235.
---	--	--	--	--	------	------	----	---

3. If a symbolic tag is assigned to an instruction, the address assigned to the tag will be the address of the leftmost character in the instruction.

## EASYCODER

CODING FORM

8					TAG	A	FICA, TAX	36, the octal value of an add is located at "TAG", a relative address. Assuming a two-character address assembly, the B-Address is located at "TAG+3" and "TAG+4".
---	--	--	--	--	-----	---	-----------	--

4. If a symbolic tag is assigned to a constant or reserved area, the address assigned to the tag will be the rightmost character in the field.

13					WORK	RESV	30	The reserved area includes relative addresses "WORK-29" to "WORK".
----	--	--	--	--	------	------	----	--

These address assignment conventions can be reserved by leaving column 8 blank and entering the first character in column 9. In this case, the following rules apply:

1. An absolute memory address assigned to an instruction refers to the rightmost character in the instruction.

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	M	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16 17 18 19 20 21 22
1					5	A	FICA, TAX	36, the octal value for add is located at absolute 1, assuming two-character addressing.

2. An absolute memory address assigned to a constant or reserved area refers to the leftmost character in the field.







Easycoder C, D, and OS/2000 (Operands Field: Card Columns 21-80)

For users of Easycoder C, D, or OS/2000, the operands field extends to column 80. Remarks may be entered following the terminating blank. One or both operands can be bypassed during assembly by writing one or two leading commas, respectively, in the operands field. Such a comma, or commas, must be left-justified in the operands field and must be followed immediately (i.e., without intervening blanks) by any remaining entries, other than remarks.

Easycoder D and OS/2000 Options

If the alternate coding format is used (i.e., the location field contains tags of from seven to ten characters in length), the operands field occupies card columns 25-80. The method of coding entries and remarks remains the same.

Examples

The first sample instruction causes the contents of the field whose rightmost character is stored in memory location 50 to be added algebraically to the contents of the field designated by the tag TOTAL.

The second instruction tests the indicator specified by variant character 3 and branches to the address tagged EQUAL if the indicator is on.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	VARIANTS	LOCATION	OPERATION CODE	OPERANDS	
				1-15	20-21
1			A	50, TOTAL	
2			BCT	EQUAL, 03	
3			ZA	TOTAL, TMP+X3	
4			MCW	TOTAL-7+X6, GROSS	
5			A	AMT, (SUM-2)	

The third line of coding above shows an instruction in which the B-address is indexed. The instruction causes the contents of field tagged TOTAL to be placed in the field designated by the tag TMP as modified by the contents of index register X3.

The fourth line of coding shows relative addressing and indexing being performed on the A-address. The instruction causes the address, -7, (tagged TOTAL) to be modified by the contents of index register X6. The resultant address specifies a field whose contents are then

placed in the field tagged GROSS. Assuming that TOTAL corresponds to memory location 540 and index register X6 contains a value of 80, the resultant address of this instruction would be 613.

The last line of coding above illustrates an instruction with indirect addressing on the B-address. The contents of the field tagged AMT are added algebraically to the contents of the field whose address is stored in the field tagged SUM-2.

ADDITIONAL CODING RULES

1. Comments and remarks can appear on any line following the last entry on that line and separated from it by a blank space. These notes will be printed on the program listing but will not be assembled as object-program entries. As mentioned previously, any line of coding containing only comments must be designated by an asterisk (\*) or the letter T in column 6.
2. Any number of blank spaces may be used between the comma which terminates the A-operand and the first character of the B-operand. Similarly, any number of spaces may be used between the comma that terminates the B-operand and a variant character.

ADDRESS CODES

Several types of address codes are valid in the operands field of an EasyCoder statement. These codes are defined and illustrated below.

ABSOLUTE

The actual address of a character position in the main memory can be represented as a decimal number; leading 0's can be omitted. The sample instruction causes the contents of the field whose rightmost character location is 32 to be moved to the field whose rightmost character location is 4000.

EASYCODER  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS	
				14 15	20 21
1			MCW	32, 4000	62 63
2					





In the second entry, the notation,  $*+9$ , refers to the rightmost character of the instruction stored immediately to the right of the MCW instruction (assuming that two-character address assembly has been specified). The instruction following the MCW instruction will be moved to the field tagged WORK when the MCW instruction is executed.

### RELATIVE

Relative addressing, or address arithmetic as it is frequently called, can be used with all absolute addresses, symbolic addresses, and the self-reference symbol (\*) (these three types of address codes are referred to as addressing "elements"). By using relative addressing, the programmer can refer to a source-program entry that is stored a specified number of locations away from a particular address. A relative address is specified by appending one or more address modifiers, each consisting of a sign and an addressing element, to another addressing element. The address modifier designates a memory location relative to the location specified by the basic addressing element. For example, the instruction below causes the contents of the field 100 characters beyond the field tagged INT to be added algebraically to the contents of the field 10 characters before the sum of the addresses defined by the tags AMTPD and ERROR.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	V	M	A	LOCATION	OPERATION CODE	OPERANDS																																																																									
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
					A	INT+100,AMTPD+ERROR-10																																																																									

The number of symbolic tags required to write a program can be greatly reduced by the use of relative addressing. The programmer decides how many and which fields in a program to tag and which to reference by relative addressing.

A certain amount of caution is required in the use of relative addressing. First of all, relative addresses are not automatically corrected as a result of subsequent insertions or deletions in the source program. The programmer must remember to adjust manually the address modifiers affected by such changes. Secondly, if relative addressing is used to refer to an operand address in another instruction, care must be taken to insure that the address is referenced by its rightmost character. For example, the A-address of the instruction shown below could be referred to elsewhere in the program as INST+2 or INST+3, depending on whether two- or three-character address assembly were specified.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER		Y M D P R	LOCATION	OPERATION CODE	OPERANDS	
1	2				3	4
			INST	A	SUBT, TOTAL	

## OUT-OF-SEQUENCE

The valid address codes also include the special symbol apostrophe (printer ' ; keypunch 8, 2; octal 12). This symbol is an element whose value is equal to the current value of the out-of-sequence base (OSB). It is followed by an address modifier to specify the address of the desired operand. The OSB is set by means of the XBASE instruction.

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER		Y M D P R	LOCATION	OPERATION CODE	OPERANDS	
1	2				3	4
				A	WORK, '+15	

In the sample statement above, assume that the out-of-sequence base (OSB) has been set to 600 (by the XBASE instruction). The data in the field tagged WORK will be added to the data in the field whose rightmost location is 615 (600 + 15). The result will then be stored in the field whose rightmost location is 615.

## BLANK

There are two conditions for which a blank operand field is valid:

1. The instruction does not require an operand (e.g., the Halt and No Operation instructions).
2. The operands are implicitly addressed: the A-operand is specified by the contents of the A-address register (AAR); the B-operand is specified by the contents of the B-address register (BAR).

If either or both operand addresses are to be supplied by other instructions (as illustrated below in the description of address literals), the affected operands should be represented by zeros; they should not be left blank.

# LITERALS<sup>1</sup>

The purpose of a literal is to allow the programmer to write in the operands field of a symbolic program statement the actual data (as opposed to the address of the field containing the data) to be operated on by an instruction. Easycoder B users can code all literals, except binary, with a maximum length of 40 characters; a binary literal can be coded with a maximum length of six characters. For users of Easycoder C, D, or OS/2000, the maximum length of any literal can be 63 characters.

The assembler automatically assigns a storage field for each literal and inserts its address (i.e., the address of its rightmost character) in the operands field of the instruction in which it appears. In effect, for every literal appearing in the source program, the assembler generates a constant containing the value of the literal, with a word mark in the leftmost character position.

NOTE: If the constant generated from a literal occupies from one to five storage locations, it is assigned a storage address only once in the program, regardless of the number of times the literal appears in the source program. (For Easycoder C, D, or OS/2000 the constant is assigned a storage address only once in the program if it occupies from one to six storage locations.) A constant that exceeds five characters (six for Easycoder C, D, or OS/2000) is assigned a storage address each time the corresponding literal appears in the source program. The latter condition can be avoided by using a DCW statement whenever a long literal is to be used more than once in the source program.

## Decimal Literals

Decimal literals are specified by writing a plus or minus sign followed by the value of the literal. When the literal is assigned to a storage field, the assembler places the sign in the zone bits of the units position of the resulting constant. Unsigned decimal values can be coded as alphanumeric literals.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS										
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
1														
2														
3														
4														

<sup>1</sup>Not available with Easycoder A.

The statement above illustrates the use of a decimal literal. The instruction causes the value 24 to be subtracted from the contents of the field tagged ACCUM.

### Binary Literals

A binary literal is represented as a decimal entry in the operands field of a symbolic instruction. The assembler automatically converts the decimal entry into a binary value and stores it (right-justified) in the storage field. The programmer must specify the number of six-bit characters used to store this value.

A binary literal is coded by writing an = sign, followed by a number which specifies how many six-bit characters should be used to store the resulting binary value, followed by the letter B, followed by the decimal representation of the desired binary literal.

NOTE: If the decimal representation of the binary literal is preceded by a minus sign, the assembler will store the binary literal in two's-complement form.

The first instruction below causes the binary equivalent of 50 (expressed as a continuous 12-bit binary value) to be added to the contents of the field tagged BEGIN+2. The second instruction has been included to illustrate how a binary literal can be used in address modification. In effect, the first instruction modifies the A-address of the second instruction by a value of +50. The third instruction causes the binary equivalent of 2,688 (expressed as a 12-bit binary value) to be moved to the field tagged IND7.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8		14 15	20 21	62 63	80
1			BA	#2B50,	BEGIN+2
2		BEGIN	MCW	ITEMA,	TOTAL
4			MCW	#2B2688,	IND7
5					

### Octal Literals

Octal literals are coded in octal notation (see Appendix A). The programmer must specify the number of six-bit characters required to store an octal literal.

NOTE: Since every octal digit can be represented as three bits, each six-bit character used to store an octal literal contains two octal digits. For example, an octal literal composed of eight octal digits can be stored in a four-character field.

An octal literal is coded in the same format as a binary literal except that the letter B used in the binary literal is replaced by the letter C. The constant stored by the assembler is always left-justified in the storage field.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER		MARK	LOCATION	OPERATION CODE	OPERANDS									
1	2					3	4	5	6	7	8	14	15	20
1				HA	#3C7777, MASK									
2														

The A operand in the above statement is a four-digit octal literal. The assembler will store it left-justified in a three-character field, as 777700.

#### Alphanumeric Literals

An alphanumeric literal is specified by writing the @ symbol before and after the value of the literal. This type of literal can contain blanks, decimal, alphabetic, and special characters (excluding the @ symbol).

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER		MARK	LOCATION	OPERATION CODE	OPERANDS									
1	2					3	4	5	6	7	8	14	15	20
				MCW	@ACCOUNTS PAYABLE, 10/19/65@, PRINT									

The statement above illustrates the use of an alphanumeric literal. The instruction causes the information contained within the @ symbols to be moved to the field tagged PRINT.

#### EASYCODER C, D, and OS/2000 OPTIONS

In addition to the form specified above, users of EasyCoder C, D, or OS/2000 have available to them three other methods of coding alphanumeric literals.

1. A number sign (#) is followed by a number from 1 through 63 which specifies the number of characters in the literal; this number is, in turn, followed by the letter A and the literal.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	T	M	A	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8					14 15 20 21		62 63 80
						MCW	#14A6 LBS @ 21F/LB, PRINT

In the above example there are 14 characters in the literal. The instruction causes these 14 characters to be moved to the field tagged PRINT.

2. If it is desired to set an item mark (in addition to a word mark) in the leftmost position of the literal constant field, a number sign (#) is followed by a number from 1 through 63 which specifies the number of characters in the literal; following this number is the letter L and the literal (see the first example below).

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	T	M	A	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8					14 15 20 21		62 63 80
1						MCW	#6L1965/A, STORE
2							
3						MCW	#6R1965/A, STORE

3. If it is desired to set an item mark in the rightmost position of the literal constant field, a number sign (#) is followed by a number from 1 through 63 which specifies the number of characters in the literal; following this number is the letter R and the literal (see the second example above).

NOTE: In form (1), alphanumeric literals of six characters or less are stored in a literal table and duplicates are eliminated. The duplicates are not, however, eliminated in forms (2) and (3).

### Area Defining Literals

An area defining literal may be used to define and reserve a working area in memory without using a separate data formatting statement. The address which defines the area is written as a symbolic tag. The size of the area to which the literal address refers is specified as a decimal value following the literal address and separated from it by a # symbol.



# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	V	M	P	R	LOCATION	OPERATION CODE	OPERANDS	
							14 15	20 21
1								
2					MCW	+AMT, MODIF, +3		
3					MODIF A	Ø, TOTAL		

## VARIANT CHARACTER

A variant character can be expressed as one alphanumeric character, as two octal digits, or as a symbolic tag.<sup>1</sup> It is written following the operand entries and separated from the last entry by a comma. Octal representation of valid characters are listed in Appendix B.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	V	M	P	R	LOCATION	OPERATION CODE	OPERANDS	
							14 15	20 21
1								
2					BCT	OFLOW, 5Ø		
3					BCC	NEG, SUM, Ø6		
4								

The first instruction above tests an indicator specified by the variant character. If the indicator is on, the instruction causes the program to branch to the address tagged OFLOW. As might be expected, the octal digits 50 represent the overflow indicator. The second instruction causes the single character at the location tagged SUM to be examined for a particular bit configuration as specified by the variant. In this case the variant 06 specifies that the character should be examined for a negative sign. If the desired bit configuration is present, the program branches to the address tagged NEG.

## INPUT/OUTPUT CONTROL CHARACTERS

Input/output control characters can be used only in conjunction with input/output instructions (see Section VIII). One or more of these characters may be written following the A-address entry in an input/output instruction, each preceded by a comma. Input/output control characters may be coded as single alphanumeric characters, as pairs of octal digits, or as symbolic tags.

<sup>1</sup> A symbolic tag, composed of at least two characters, may be used to represent (1) a variant character, or (2) a group of input/output control characters. The number of I/O control characters that may be represented varies from one to six (using either Easycoder A or B) or from one to four (using Easycoder C, D, or OS/2000). The symbolic tag must be defined before it is used in the input/output instruction; the Control Equals statement (CEQU) is generally used for this purpose.





is used because an operand address cannot be introduced with a plus or a minus sign. Thus, the effective A-address of the MCW instruction will be a value six less than that stored in index register X1 (i.e., if index register X1 contains 126, the effective A-address is 120).

Three- or four-character address assembly must be specified (see ADMODE) whenever indexed addressing is to be performed. When the assembler translates an indexed address into a machine-language entry (see Figures 5-6 and 5-7), the translated index register designator is automatically inserted into the address modifier bit positions of the assembled address.

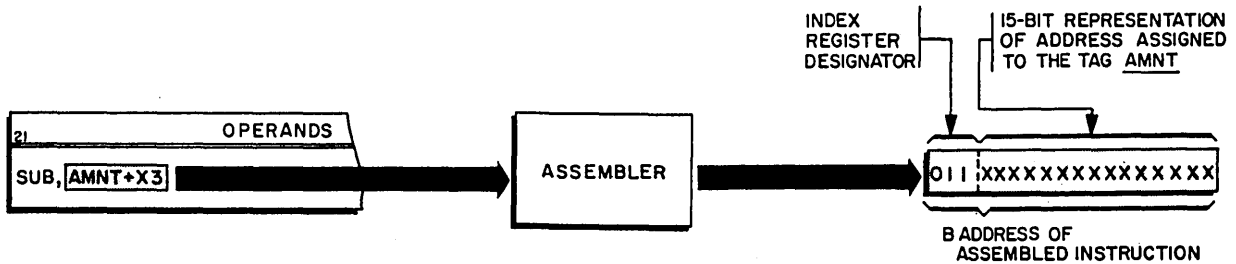


Figure 5-6. Assembly of Indexed Address in Three-Character Addressing Mode

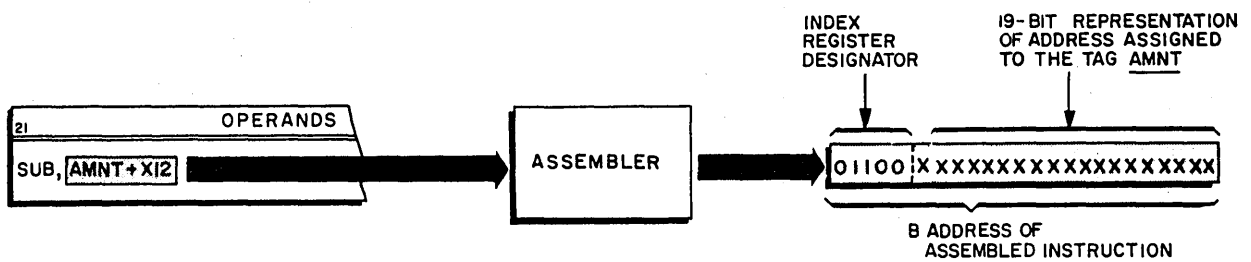


Figure 5-7. Assembly of Indexed Address in Four-Character Addressing Mode

### INDIRECT

An indirect address is specified by enclosing the address (either symbolic or absolute) in parentheses.<sup>1</sup> For example, in the sample instruction below, the parentheses around the tag

<sup>1</sup>The left parenthesis corresponds to keypunch symbol % (card code 0, 8, 4), octal 74; the right parenthesis to keypunch symbol □ (card code R, 8, 4), octal 34.

DATA indicate to the assembler that DATA refers to the leftmost character of a field containing another address. This second address may be a direct, an indexed, or another indirect address. If it is direct or indexed, it specifies the rightmost character of a data field. If it is indirect, it specifies the leftmost character of a field containing another address.

## EASYCODER

CODING FORM

PROBLEM _____										PROGRAMMER _____										DATE _____										PAGE _____ OF _____									
CARD NUMBER		M A R		LOCATION				OPERATION CODE				OPERANDS																											
1	2	3	4	5	6	7	8	14	15	20	21															62	63	80											
1												MCW (DATA), WORK																											
2																																							

Three- or four-character address assembly must be specified (see ADMODE) whenever indirect addressing is to be used. When the assembler translates an indirect address into a machine-language entry (see Figures 5-8 and 5-9), a binary value of 111 (three-character mode) or 10000 (four-character mode) is automatically inserted into the address modifier bit positions of the assembled address.

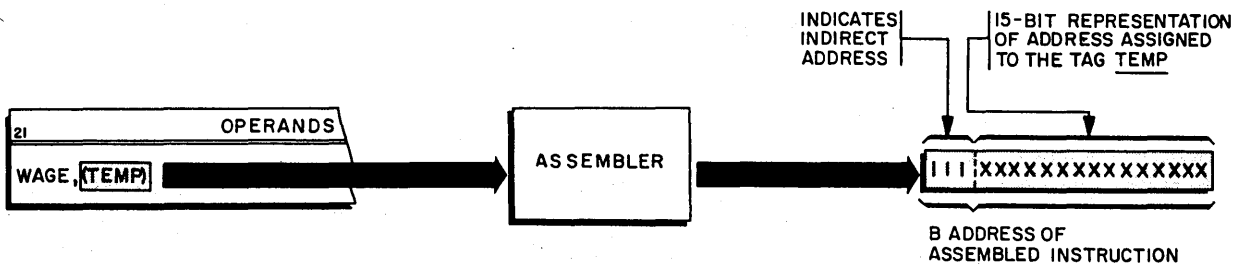


Figure 5-8. Assembly of Indirect Address in Three-Character Addressing Mode

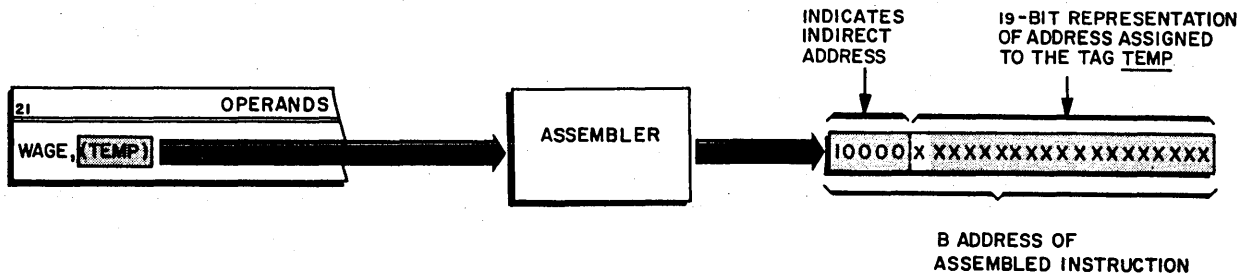


Figure 5-9. Assembly of Indirect Address in Four-Character Addressing Mode

SECTION VI  
DATA FORMATTING STATEMENTS

INTRODUCTION

A value or quantity which must remain fixed or which must be used repeatedly in a program is called a constant. A work area is an area in memory which is reserved for input data, cumulative processing, or output data. By employing data formatting statements, constants can be stored and work areas can be reserved without regard to their actual locations in memory. For instance, the programmer can use a data formatting statement to reserve an 80-character card input area and assign it a symbolic address such as CARDIN, without knowing the actual address of the field. Similarly, a data formatting statement makes it possible to store a constant, such as 2000, and to refer to it by a symbolic tag, such as CON3, without regard to the address at which the constant is stored. Table 6-1 lists the five data formatting statements used with EasyCoder symbolic language.

Table 6-1. Data Formatting Statements

Mnemonic Operation Code	Function
DCW	Define Constant with Word Mark
DC	Define Constant without Word Mark
RESV	Reserve Area
DSA	Define Symbol Address
DA	Define Area*

\*NOTE: The Define Area statement cannot be employed with the EasyCoder A Assembly System.

Although data formatting statements are coded in the same format as most symbolic machine instructions (data processing statements), they are not treated as instructions by an assembler. Instead they are treated as definitions which cause the assembler to perform certain activities but which are not executed during a program run. Since data formatting statements are not executed during a program run, they should not be written in the body of the symbolic program.

Define Constant with Word Mark - DCW

By use of the DCW statement, a constant can be automatically stored in a field reserved by the assembler. In storing the constant, the assembler automatically sets a word mark in the leftmost character position of the storage field. Item marking may be specified as in Table 5-1. An L in column 7 thus results in a record mark with a DCW statement.

NOTE: If Easycoder C, D, or OS/2000 is being used, and if unusual high- and low-order punctuation is required, the programmer may use a Set II punctuation indicator as shown in Table 5-2.

The constant can be assigned a tag. If the tag is left-justified in the location field, it is assigned to the address of the rightmost character of the constant. If the tag is indented one column, it is assigned to the address of the leftmost character of the constant.

NUMERIC CONSTANTS

Numeric constants may take any one of three forms: binary, octal, or decimal. For Easycoder A and B, octal and decimal constants can be coded with a maximum length of 40 characters, while the coding associated with a binary constant is limited to a maximum of six characters. However, the Easycoder C, D, and OS/2000, the maximum length of the storage field which can be occupied by a numeric constant is 63 characters.

Decimal Constants

Signed decimal constants are specified by writing a plus or a minus sign in the first column of the operands field, followed by the value of the constant. When the constant is assigned to a storage field, the assembler places the sign in the zone bits of the rightmost character of the constant.<sup>1</sup> Unsigned decimal constants are written left-justified in the operands field.

**EASYCODER**

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8				
		DEC	DCW	+22

The statement above shows the decimal value of +22 defined as a decimal constant.

Binary Constants

A binary constant is actually written as a decimal entry (maximum value of 999999) which is then automatically converted to a binary value by the assembler. The binary value is stored

<sup>1</sup> See the description of sign codes.

(right-justified) in the constant field. The stated maximum value of 999999 for binary constants is for Easycoder Assemblers A and B only.

To code a binary constant the programmer writes the following: (1) a # sign (in the first column of the operands field); (2) for Easycoder A or B, a number from 1 to 6 which designates the number of six-bit characters needed to store the resulting binary value (for Easycoder C, D, or OS/2000, a number from 1 to 63); (3) the letter B; and (4) the decimal representation of the desired binary constant. Note that if the decimal representation of the binary constant is preceded by a minus sign, the assembler stores the binary constant in twos-complement form.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS										
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
		CON3	DCW	#2B50										

The statement above shows the binary equivalent of 50 defined as a binary constant to be stored in two consecutive character locations.

#### Octal Constants

Octal constants are coded in octal notation (see Appendix A). To code an octal constant the programmer writes the following: (1) a # sign (in the first column of the operands field); (2) a number (not to exceed 20 for Easycoder A and B); not to exceed 63 for Easycoder C, D, and OS/2000, which specifies the number of six-bit characters required to store the octal constant;<sup>1</sup> (3) the letter C; (4) the constant value. Note that the value stored by the assembler is always left-justified in the storage field.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS										
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
		OCT7	DCW	#2C777										

In the statement above, the octal value of 7777 is shown defined as an octal constant to be stored in two consecutive character locations.

<sup>1</sup> Recall that an octal digit can be represented as three bits; thus each six-bit character used to store an octal constant contains two octal digits. For example, an octal constant composed of six octal digits can be stored in a three-character field.

ALPHANUMERIC CONSTANTS

Alphanumeric constants may be coded in one of three ways:

1. Constants (including special symbols and blanks) may be written with the constant value enclosed in @symbols (see the first entry below).
2. If the @symbol is required in the constant, this constant is enclosed in any unused character other than blank, +, -, #, (and F, for Easycoder D and OS/2000 or the digits 0 through 9 (see the second entry below).
3. A number sign (#) is followed by a number from 1 through 56 which specifies the number of alphanumeric characters contained in the constant; this number is, in turn, followed by the letter A and the alphanumeric constant (see the third entry below).<sup>1</sup>

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y A R	W E R	LOCATION	OPERATION CODE	OPERANDS									
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
			COST	DCW	@\$2,128.60@									
2														
3			TIME	DCW	@3:00PM@									
4														
5			DATE	DCW	#4A1972									

NOTE: The maximum number of alphanumeric characters which can be contained in the constant, of course, depends on the number of card columns available in the operands field. Thus it should be remembered that methods 1 and 2, above, require two card columns to format the constant, while method 3 requires either three or four columns.

BLANK CONSTANTS

The DCW statement may be used to reserve a field of blanks with a word mark in the leftmost character position of the field. The programmer writes a # symbol (in the first column) followed by a decimal value (from 1 to 40 for Easycoder A or B, from 1 to 63 for Easycoder C, D, or OS/2000) which indicates the number of blank storage positions desired.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y A R	W E R	LOCATION	OPERATION CODE	OPERANDS									
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
			BLANK	DCW	#21									

<sup>1</sup> This third method of coding alphanumeric constants is applicable only when using Easycoder C, D, or OS/2000.

The DCW statement above defines a 21-character blank field. The address assigned to this field by the assembler will be inserted in an object-program instruction whenever the tag BLANK appears in another symbolic-program entry.

### FLOATING-POINT CONSTANTS

A floating-point constant is written as a decimal entry which is then automatically converted by the assembler to a fixed-length floating-point value, viz., a six-character binary mantissa followed by a two-character power-of-two exponent.

To code a floating-point constant the programmer writes the following:

1. The letter F.
2. A decimal number, the mantissa, which may be signed or unsigned and which may contain a maximum of 11 digits with or without a decimal point.
3. The letter E.
4. A decimal number, the exponent, which must be between 0 and 616, inclusive, and may be signed or unsigned.

If an exponent of zero is desired, the letter E and the decimal number which follows it are not required.

NOTE: If the mantissa and/or the exponent is preceded by a minus sign, the assembler stores the corresponding value in twos-complement form.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	OPERATION	LOCATION	OPERANDS	
				1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16 17 18 19 20 21
1		DCW	FCON1	F4.359E2	
2		DCW	FCON2	F+4359E-1	
3		DCW	FCON3	F1	
4		DCW	FCON4	F-.0001	
5		DCW	FCON5	F-1E-4	

The first two entries above (FCON1 and FCON2) result in the same floating-point value when converted by the assembler. FCON1 uses a decimal point while FCON2 arrives at the same result by using a negative exponent. This is also true for FCON4 and FCON5.

Define Constant - DC

The DC statement is functionally the same as the DCW statement, the only exception being the absence of automatic word marking. This statement may thus be used in place of the DCW



statement if a constant is to be stored without a word mark in its leftmost character position. The programmer, however, may still specify item marking as shown in Table 5-1.

NOTE: If Easycoder C, D, or OS/2000 is being used, and if unusual high- and low-order punctuation is required, the programmer may use a Set II punctuation indicator as shown in Table 5-2.

**Reserve Area - RESV**

Use of the RESV statement enables the programmer to reserve an area of memory. Unlike the DC and DCW statements (which cause data to be loaded into an area reserved by the assembler), the RESV statement does not normally alter the contents of the area defined. Rather, it simply sets aside a storage area to which the programmer can refer by a symbolic tag. The reserved area can be cleared to 0's by means of the CLEAR statement. The number of characters in the reserved area must be specified in the operands field of the RESV statement.

NOTE: When used with Easycoder A or B, the RESV statement must contain a nonzero value in the operands field.

A symbolic tag may be written in the location field. If the tag is left-justified, it is assigned to the rightmost location of the reserved area. If the tag is indented one column, it is assigned to the leftmost location of the reserved area.

When used with Easycoder C, D, or OS/2000, the RESV statement cannot only reserve a specified area but can also load that area with a particular character. The character to be loaded into each location of the reserved area is coded in the op code field immediately following a comma and the mnemonic code. If the mnemonic RESV is followed only by a comma, the reserved area is cleared to blanks.

NOTE: There is no automatic word marking for the reserved areas, nor may column 7 of the RESV statement be used with Easycoder A or B to set punctuation. However, if Easycoder C, D, or OS/2000 is being used, the programmer may use a Set I or II punctuation indicator.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8	14 15 20 21		62 63 80
1	STORE	RESV	30
2	CARD	RESV,	080

The first statement above reserves 30 consecutive character positions that can be addressed via the tag STORE. Note that by referring to the reserved area via a symbolic tag, the programmer need not know its actual location in memory. The second RESV statement, assembled by Easycoder C, D, or OS/2000 reserves 80 consecutive locations and clears the reserved area to 0's.

**Define Symbolic Address - DSA**

The DSA statement can be used to store one or two addresses, or two addresses and a variant character, as a constant. Any valid address can be stored as a constant; the length of each address is determined by the current addressing mode (each address will be two, three, or four characters long).

An item mark may be specified as shown in punctuation Set I Table 5-1. In addition, the DSA statement automatically places a word mark in the leftmost character position of the constant (thus an L in column 7 results in a record mark in this position).

NOTE: If EasyCoder C, D, or OS/2000 is being used, and if unusual high- and low-order punctuation is required, the programmer may use a Set II punctuation indicator as shown in Table 5-2.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y E R	LOCATION	OPERATION CODE	OPERANDS	
1	2 3 4 5 6 7 8		14 15 20 21		62 63 80
		CODE	DSA	ITEM-5	
2					
3		STAR	DSA	ARG,*,A	

The first statement above permits the address of the field five characters before the field tagged ITEM to be referred to in the program by the tag CODE.

The second statement allows the stored constant consisting of the address assigned to ARG, the address assigned to the self-reference indicator \*, and the variant character A (i.e., octal 21) to be referred to by the tag STAR.

**Define Area - DA<sup>1</sup>**

A specified area within the main memory can be defined and reserved by using the DA statement. In addition to defining an area, the DA statement can also define fields and subfields within the reserved area. This statement can also define two or more contiguous areas if these areas are identical in format. In other words, the programmer uses a DA statement to provide the assembler with the following basic information:

1. The number (n) and size (s) of the reserved area(s). (Both n and s can be represented by numbers up to 4,095, depending upon the amount of memory available.)
2. The index register (Xm or Ym) to be associated with each reference to a field or subfield within the reserved area(s) (optional).

<sup>1</sup>The Define Area statement cannot be employed with the EasyCoder A Assembly System.

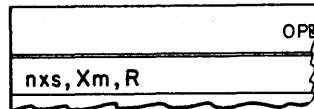
3. The character R which will place a record mark one position to the right of the rightmost reserved area (optional).

NOTE: Additional parameters may be employed with Easycoder C, D, and OS/2000.

A DA statement consists of a heading line which defines an area(s), plus one or more subsequent lines of coding which define the fields and subfields within the area(s). The heading line can contain a symbolic tag (but not an absolute address) in the location field. If this tag begins in column 8, it refers to the rightmost location of the entire area, exclusive of the record mark (if present); if the tag starts in column 9, it refers to the leftmost location of the entire area. Item marks may be specified in column 7 of the heading line by using Set I punctuation indicators as shown in Table 5-1.

NOTE: The list of punctuation indicators specified in Set II, Table 5-2, cannot be used with DA statements.

The operands field in the heading line has the following format:



If a single 80-character area is to be defined, the value of nxs is 1x80. If four identical 80-character areas are to be defined, the value of nxs is 4x80.

The DA statement can be indexed by writing an index register designator (from X1 through X15 or from Y1 through Y15)<sup>1</sup> following the area definition. All references to the field and subfields defined in the DA statement will be automatically indexed by the specified index register, but references to the tag assigned to the entire area will not be indexed. For example, the statement on the next page indicates that all references to the fields and subfields in the 113-character area tagged BUFFER will be indexed by the index register X2; references to the tag BUFFER, however, will not be indexed.

Note that the area definition nxs does not include an allowance for the character position containing the record mark, although this position (if any) is also reserved. For example 4x80 will cause 320 character positions to be reserved. If a record mark is placed one position to the right of the last area, a total of 321 character positions is reserved.

The index register applied to a field or subfield can be changed from that specified in the DA statement by designating a different register in the operands field of an instruction which

---

<sup>1</sup> Index registers X1 through X6 are used with Easycoder B, while index registers X1 through X15 and Y1 through Y15 can be used with Easycoder C, D, or OS/2000.

references the field or subfield. The effect of indexing on a field or subfield can be cancelled by writing X0 as the index register designator in the references in which indexing is not wanted.

As stated above, the heading line may be followed by one or more lines of coding which define fields and subfields within the reserved area(s). As many of these lines as necessary may be used, and these fields and subfields may be defined in any order desired. Positions within each reserved area are numbered sequentially from left to right, starting with one. The coding lines which define fields and subfields must have blank op code fields; each such line may contain a symbolic tag in the location field, if desired.

Fields and subfields are specified as follows:

**Fields:** The lowest and highest positions of the field are written in that order in the operands field, separated by a comma. (If a one-character field is desired, its position number must be written twice in the operands field, separated by a comma.) A word mark is automatically placed in the leftmost position of the field in memory. Item marks may be specified as shown in Table 5-1.

**Subfields:** For a subfield, only the rightmost position is specified. Word marks are not set; however, item marks may be specified as shown in Table 5-1.

**NOTE:** The list of punctuation indicators specified in Set II can not be used with DA statements.

The assembler does not normally clear the defined area. However, the programmer has the option of clearing the area to a specified character by placing a comma and the desired character after the mnemonic code DA in the op code field. The presence of only a comma after the op code implies that the area will be cleared to blanks. When the defined area is cleared, all punctuation is also cleared before setting the "field" punctuation.

The sample coding below illustrates what a DA statement might look like.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	I	X	A	R	LOCATION	OPERATION CODE	OPERANDS							
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
01					BUFFER	DA	4	X28, X2, R						
02					NAME			1, 20						
03					DATE			23, 28						
04					AGE			21, 22						
05					YEAR			28						
06					MONTH			26						

The heading line specifies the following information:

1. Four consecutive, identical areas, each 28 characters long, will be reserved.
2. The tags NAME, DATE, AGE, YEAR, and MONTH, when referred to in symbolic instructions, will be indexed by index register X2.
3. A record mark will be set in the rightmost character position of the entire 113-character reserved area.
4. The entire 113-character area can be referred to via the tag BUFFER. (This tag refers to the leftmost position of the area because it is indented. It is not automatically indexed by index register X2.)

Lines two, three, and four define fields. Word marks will be set in positions 1, 21, and 23 in each of the four identical areas. Lines five and six define subfields: position 28 indicates the year within the date, while position 26 indicates the month within the date.

Easycoder C, D, and OS/2000 Options

When used with Easycoder C, D, or OS/2000, the DA statement may make use of the following parameters (in addition to the n, s, Xm, and R parameters).

1. The character P: Coding this character in the heading line of a DA statement causes the special character 72g, together with an item mark, to be placed at the end of each area as an additional character.
2. The character G: Coding this character in the heading line causes the special character 32g, together with a record mark, to be placed one position to the right of the last area.
3. The character H: Coding this character in the heading line instructs the assembler to associate the index register (Xm or Ym) with each reference to the tag in the location field of the DA statement, as well as with each reference to a field or subfield within the reserved area(s).

NOTE: If a symbolic tag is used, it is not automatically indexed by the specified index register (Xm or Ym) unless parameter H is employed. This parameter is meaningless if no index register is specified.

The format of a DA statement heading line employing all parameters is illustrated below.

**EASYCODER**

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8	14 15 20 21		62 63 60
1		tag	DA nXs, Xm, R, P, G, H
2			

SECTION VII  
ASSEMBLY CONTROL STATEMENTS

INTRODUCTION

Assembly control statements provide programmer control over the assembly of the source program. These statements resemble data formatting statements in that they are treated as definitions. They control such functions as the addressing mode to be used in assembling specified instructions, the assignment of absolute locations to symbolic tags, etc. Used only during the assembly process, assembly control statements are never executed as instructions in the object program. The precise function of each assembly control statement depends upon the assembly system employed.

A summary of the assembly control statements available with Easycoder A, B, C, D, and OS/2000, together with the page where each statement is defined, may be found in Table 7-1. In addition, the heading of each statement in this section includes a table which indicates the assembly systems that may use that particular statement.

Table 7-1. Assembly Control Statements

Easycoder A	Easycoder B	Easycoder C	Easycoder D	OS/2000 Easycoder
Assembly Control Statements	Assembly Control Statements	Assembly Control Statements	Assembly Control Statements	Assembly Control Statements
Program Header	Program Header	Program Header	Program Header	Program Header
		Segment Header	Segment Header	Segment Header
Execute	Execute	Execute	Execute	Execute
		Transfer	Transfer	Transfer
Origin	Origin	Origin	Origin	Origin
Modular Origin	Modular Origin	Modular Origin	Modular Origin	Modular Origin
	Literal Origin	Literal Origin	Literal Origin	Literal Origin
Admode	Admode	Admode	Admode	Admode
Equals	Equals	Equals	Equals	Equals
Control Equals	Control Equals	Control Equals	Control Equals	Control Equals
Memory Dump		Skip	Skip	Skip
		Suffix	Suffix	Suffix

Table 7-1 (cont). Assembly Control Statements

EasyCoder A	EasyCoder B	EasyCoder C	EasyCoder D	OS/2000 EasyCoder
Assembly Control Statements	Assembly Control Statements	Assembly Control Statements	Assembly Control Statements	Assembly Control Statements
Clear	Clear	Repeat Generate Set Line Number Set Out-of-Sequence Base Clear	Repeat Generate Set Line Number Set Out-of-Sequence Base Clear	Repeat Generate Set Line Number Set Out-of-Sequence Base Clear Range
End	End	End	End	End

Program Header PROG
------------------------

A	B	C	D	OS/2000

The program header must be the first entry in a symbolic program. This statement is coded as follows for the various assembly systems.

EASYCODER A

The letters PROG must be written in the op code field, and the operands field must contain a name which identifies the program. (This name will appear in the program listing.) Optionally, an "S" can be placed in column 6; this action specifies that a check is to be made on the card number sequence of the input deck.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8 14 15 20 21 62 63 80		
	PROG	SERIES

In the sample statement above, SERIES is specified as the program name, while the letter S in column 6 designates that a sequence check is desired.



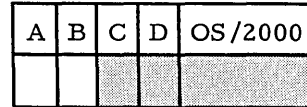
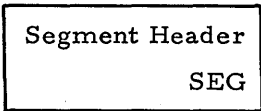


If the programmer desires to use the alternate card format, which allows room for tags consisting of up to ten characters, column 75 of the program header card must contain the letter A. The PROG card itself, however, is never coded in the alternate format: the letters PROG always appear in the op code field (columns 15 through 18), while the name of the program always appears beginning in column 21.

NOTE: If the alternate format is specified, all cards following the program header, up to and including the END card, must be coded in the alternate format.

OS/2000 EASYCODER

As used in OS/2000 EasyCoder, the program header provides program identification; in addition, however, this serves as an "action director" to supersede default and job control language-specified options. For this reason, the programmer should refer to the Honeywell publication OS/2000 EasyCoder Assembler, Order No. AH31.



Programs written for EasyCoder C, D, or OS/2000 may be divided into two or more segments, each of which is loaded into memory and executed as a unit. It is the function of the SEG statement to define the beginning of each segment (memory load). Use of the SEG statement is optional, however. If used, a SEG statement must follow the program header, each Execute statement and each Transfer statement. If it is desired to omit this statement, it must be omitted from the entire program; in this case the assembler generates segment identifications (starting with 01).

EASYCODER C, D, and OS/2000

The letters SEG must be placed in the op code field, while the operands field must contain a two-character segment identification. This segment identification becomes appended to the program name to form a unique search code.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	9 10 11 12	13 14 15 16 17 18 19 20 21	22 23 24 25 26 27 28 29 30	31 32 33 34 35 36 37 38 39 40
1			SEG	AA
2				

In the example above, AA could represent the first segment of a program, in which case this entry would follow the program header.

Execute
EX

A	B	C	D	OS/2000

The end of a memory load is indicated by an EX statement. When the coding inserted by the assembler for the EX statement is encountered during the loading process, a branch to the location specified in the operands field results. This operation enables portions of the program to be executed before the entire program has been loaded. The coding to be executed must appear prior to the EX statement.

### EASYCODER A

The letters EX must be written in the op code field; the operands field contains a direct address, either absolute or symbolic. (If an EX statement is written with a blank operands field, the machine will halt when it encounters the corresponding coding during the loading operation.)

To resume the loading operation, the last instruction in the portion of the program executed must be a Branch instruction which provides re-entry to the load routine. In addition, the first instruction of the executed routine should be an SCR (Store Control Registers) instruction which stores the contents of the B-address register in the A-address of the return Branch instruction.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	W	R	LOCATION	OPERATION CODE	OPERANDS																							
							1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1					EX	SEC 3																							
2																													
3																													

The sample statement above illustrates an EX statement with a symbolic address in the operands field. When the corresponding coding is encountered during the loading operation, program loading is temporarily halted and the portion of the program beginning at the location tagged SEC 3 is executed.

## EASYCODER B

The letters EX must be written in the op code field; the operands field contains a direct address, either absolute or symbolic. (If an EX statement is written with a blank operands field, the machine will halt when it encounters the corresponding coding during the loading operation.)

To resume the loading operation, the last instruction in the portion of the program executed must be a Branch instruction which provides re-entry to the load routine. In addition, the first instruction of the executed routine should be an SCR (Store Control Registers) instruction which stores the contents of the B-address register in the A-address of the return Branch instruction.

Besides causing a branch to the programmer's coding, use of the EX statement causes any literals used in the memory load to be loaded and the literal table to be cleared. If a LITORG statement (see below) does not precede the EX statement, literals are allocated immediately following the in-line coding for the memory load.

- NOTES:
1. Following an EX statement, a new segment number is generated as explained above in the description of the program header.
  2. With Easycoder B, the total of the numbers of Execute, Literal Origin, and End statements must not exceed 31.

See the sample statement given above for Easycoder A.

## EASYCODER C, D, and OS/2000

The letters EX must be written in the op code field; the operands field must contain a direct address, either absolute or symbolic. When used with these assemblers, the EX statement enables a program to be loaded and executed one segment at a time. Each segment except the last must end with either an EX or an XFR statement. When an EX statement is encountered, all literals preceding the EX statement which have not been allocated to memory are allocated in sequence, and the literal table is cleared.

Note that it is the responsibility of the programmer to provide re-entry to the load routine. The methods of returning to the applicable loader are described in the pertinent Honeywell publication - e.g., Card Loader-Monitor B (Order No. BA95), Tape Loader-Monitor C (Order No. BB20), or OS/2000 Supervisor Components (Order No. AH23).

See the sample statement given above for Easycoder A.

Transfer
XFR

A	B	C	D	OS/2000

For Easycoder C, D, and OS/2000 users, the end of a memory load may be indicated by an XFR statement instead of an EX statement. Both statements perform essentially the same functions; the one exception is that use of the XFR statement does not result in the allocation of literals or in the clearing of the literal table.

When the coding inserted by the assembler for the XFR statement is encountered during the loading process, a branch to the location specified in the operands field results. This operation enables portions of the program to be executed before the entire program has been loaded.

### EASYCODER C, D, and OS/2000

The letters XFR must be written in the op code field; the operands field must contain a direct address, either absolute or symbolic. Use of this statement enables a program to be loaded and executed one segment at a time. Each segment except the last must end with either an XFR or an EX statement.

NOTE: It is the responsibility of the programmer to provide re-entry to the load routine.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS	
1			XFR	SEC4	
2					

The sample statement above illustrates an XFR statement with a symbolic address in the operands field. When the corresponding coding is encountered during the loading operation, program loading is temporarily halted and the portion of the program beginning at the location tagged SEC4 is executed.

Origin  
ORG

A	B	C	D	OS/2000

The ORG statement is used to modify the normal memory allocation process of assembly. This statement can be inserted anywhere in the source program to indicate to the assembler that all subsequent coding (instructions, constants, work areas, etc.) should be assigned sequential memory locations starting with the location whose address is specified in the operands field.

A program is normally allocated memory space beginning at location 0. If it is desired to assign memory space starting at some location other than 0, an ORG statement must be inserted in the program immediately following the program header.

EASYCODER A

The letters ORG are written in the op code field, and an address (either absolute or symbolic) is written in the operands field. (If the address is symbolic, the tag must appear in the location field of a previous source-program entry.) The address specified in the operands field is assigned the tag (if any) in the location field; if this tag appears, it must not be indented.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS	
			14 15 20 21	62 63 80
1	ORG		750	
2				
3	ORG	ORTAG		
4				
5				

The first statement above indicates to the assembler that all subsequent entries should be assigned sequential addresses beginning with location 750. The second statement directs the assembler to assign to all subsequent entries sequential addresses beginning with the address that is assigned to the tag ORTAG. (ORTAG must appear in the location field of a previous source-program entry.)

EASYCODER B

The letters ORG are written in the op code field, and an address (either absolute or symbolic) is written in the operands field. (If the address is symbolic, the tag must appear in the location field of a previous source-program entry.) The address specified in the operands field is assigned the tag (if any) in the location field; if this tag appears, it must not be indented.

NOTE: When the BRT punched-card format is specified, an ORG statement must be included immediately following the PROG statement with an address of 1,000 (decimal) or above.

See the sample statements given above for EasyCoder A.

EASYCODER C, D, and OS/2000

The letters ORG are written in the op code field, and an address (either absolute or symbolic) is written in the operands field. If the address is symbolic, the tag must appear in the location field of another (not necessarily previous) source-program entry. A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the ORG statement not been present.

NOTE: Care must be taken so that the address in the operands field is a decimal number of 1,000 or above if Card Loader-Monitor B is used to load the object program. If Tape Loader-Monitor C or Drum Bootstrap-Loader C is used, this decimal number must be 1,340 or above. For OS/2000 the object program must start at 190 or higher. The ORG statement has additional functions for relocatable code (see OS/2000 Easycoder Assembler, Order No. AH31).

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8	14 15 20 21	14 15	22 23 24 25 26 27 28 29 30
		IDENT ORG	7800
2			

In the example above, assume that the instruction preceding the ORG statement was assigned to locations 5000 through 5007. The next instruction would normally begin at location 5008. The tag IDENT, since it begins in column 9, is thus assigned to location 5008, and the next instruction is stored beginning at location 7800.

Modular Origin MORG
------------------------

A	B	C	D	OS/2000

The modular origin statement is similar to the ORG statement described above. The MORG statement indicates to the assembler that all subsequent entries should be assigned sequential addresses starting with the next available location whose address is a multiple of the number written in the operands field of the MORG statement. The entry in the operands field must represent a power of two (e.g., 2, 4, 8, 16, 32, ..... 4,096, etc.).

EASYCODER A and B

The letters MORG are written in the op code field, and a number (a power of two) is placed in the operands field.



A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the LITORG statement not been present.

- NOTES: 1. In the absence of a LITORG statement, all of the generated coding associated with a memory load is allocated immediately following the in-line coding.
2. With Easycoder B, the total of the number of Execute, Literal Origin, and End statements must not exceed 31.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER		Y 1 P E R K	W A R K	LOCATION	OPERATION CODE	OPERANDS									
1	2			3	4	5	6	7	8	14	15	20	21	62	63
1				LIT	LITORG	1550									

In the LITORG statement above, the assembler is directed to assign sequential addresses - starting with location 1550 - to all previously encountered literals. This location is also tagged LIT, since the tag begins in column 8.

### EASYCODER C, D, and OS/2000

The op code field must contain the letters LITORG, while the operands field contains an address (either absolute or symbolic). If a symbolic tag is used, it must have appeared in the location field of another, not necessarily previous, entry. Like the EX statement, the LITORG statement causes the literal table to be cleared. Locations below the loader (EasyCoder C and D) or the Supervisor Communications Region (OS/2000 EasyCoder) must not be used.

A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the LITORG statement not been present.

- NOTE: In the absence of a LITORG statement, all of the generated coding associated with a memory load - except for a memory load terminated by an XFR statement - is allocated immediately following the in-line coding.

### EASYCODER CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER		Y 1 P E R K	W A R K	LOCATION	OPERATION CODE	OPERANDS									
1	2			3	4	5	6	7	8	14	15	20	21	62	63
1				LIT	LITORG	1750									
2															
3				IDENT	LITORG	2000									





The assembler upon encountering the first statement above will assemble the address portions of all subsequent instructions as two-character addresses. The second statement, if encountered later in the same source program, will cause the assembler to change to three-character address assembly.

EASYCODER C, D, and OS/2000

The letters ADMODE are placed in the op code field. The operands field contains either the numbers 2, 3, 4, or a symbolic tag to denote whether all subsequent instructions are to be assembled in the two-, three-, or four-character addressing mode. If a symbolic tag is used, it must have been previously defined to have a value of 2, 3, or 4. If an ADMODE statement is not included at the beginning of the source program, three-character addressing is assumed by the assembler. (It should be a general rule, however, to include an ADMODE statement at the outset of every program.) See the sample statements given above for Easycoder A and B.

Equals
EQU

A	B	C	D	OS/2000

The EQU statement assigns the symbolic tag written in the location field to the address (absolute or symbolic) written in the operands field. This statement thus makes it possible to use different symbolic tags in different parts of the source program to refer to the same memory location.

EASYCODER A and B

The location field contains a symbolic tag, while the op code field contains the letters EQU. The operands field contains the address to which the symbolic tag in the location field is to be assigned. (Each symbolic tag written in the operands field must appear in the location field of a previous source-program entry.)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	9	10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1			TITLE EQU NAME	
2			QUAN EQU AMT-20	

The first EQU statement above causes the assembler to assign the tag TITLE the same location assigned the tag NAME. Thus, the programmer can use either of these two tags to refer to the contents of this location. The second statement employs relative addressing. The assembler will assign the tag QUAN to the location specified by address arithmetic as AMT-20.

EASYCODER C, D, and OS/2000

The location field contains a symbolic tag, while the op code field contains the letters EQU. The operands field contains the address to which the symbolic tag is to be assigned. A symbolic tag written in the operands field must appear in the location field of another (not necessarily previous) source program entry.

See the sample statement given above for Easycoder A and B.

Control Equals CEQU
------------------------

A	B	C	D	OS/2000

The CEQU statement assigns the symbolic tag written in the location field to the value written in the operands field. It is frequently used to assign a tag (containing a minimum of two characters) to a variant character or to a group of input/output control characters.

EASYCODER A and B

The location field contains a symbolic tag, while the op code field contains the letters CEQU. The operands field contains an octal value; this entry is coded as an octal constant and may contain up to 12 octal digits. The symbolic tag in the location field is assigned to this entry.

NOTE: A description of octal constants may be found under the heading "Define Constant with Word Mark - DCW"

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	MARKER	LOCATION	OPERATION CODE	OPERANDS
1		OFLOW	CEQU	#1C50
2			BCT	SUB2, OFLOW

The sample coding above illustrates how a symbolic tag can be used in place of a variant character. The CEQU statement directs the assembler to equate the tag OFLOW to the octal value 50. The second line of coding contains a branch instruction which specifies that a program should branch to the location tagged SUB2 if the condition specified by the variant character tagged OFLOW is present.

The location field contains a symbolic tag, while the op code field contains the letters CEQU. The entry in the operands field must be a decimal, binary, octal, or alphanumeric constant (the octal format is most commonly used). Regardless of the constant used, however, the resultant value must not exceed four characters in length.

- NOTES: 1. Instructions which refer to the tag defined by the CEQU statement must not precede the CEQU statement.
2. A description of constants may be found under the heading "Define Constant with Word Mark - DCW"

See the sample statement given above for Easycoder A and B.

Memory Dump
HSM

A	B	C	D	OS/2000

The HSM statement may be used with Easycoder A to produce a punched card deck containing the Memory Dump routine. This card deck can be loaded into memory to obtain a printed listing of the contents of any portion of main memory. This statement must be coded immediately preceding the CLEAR and END statements in the source program (see below). The total number of HSM, CLEAR, and END statements must not exceed 10.

EASYCODER A

If the punched card deck (containing the Memory Dump routine) is to be loaded into a specific memory area, the start of this area can be specified by a tag in the location field of the HSM statement. A blank location field causes the Memory Dump routine to be loaded into the area following the location assigned to the last character in the object program. The letters HSM must be written in the op code field. The operands field contains the addresses of the first (low) and last (high) locations in the memory area whose contents are to be listed by the Memory Dump routine.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	Y	N	P	R	LOCATION	OPERATION CODE	OPERANDS				
							14 15	20 21			
1	2	3	4	5	6	7	8	14 15	20 21	62 63	80
						HSM		START	STOP+3		
2											

The HSM statement above specifies that the area whose contents are to be listed begins at the location tagged START and ends three locations beyond the location tagged STOP. As the location field is blank, the Memory Dump routine will be stored in the area following the location assigned to the last character in the object program.

Skip  
SKIP

A	B	C	D	OS/2000

Easycoder assemblers normally single-space an assembly listing and skip to the head of the next form when a page becomes filled. The SKIP statement enables the programmer to control the vertical spacing of the assembly listing by causing as many as 15 lines to be skipped.

EASYCODER C, D, and OS/2000

The letters SKIP are placed in the op code field. The operands field contains either a number from 1 to 15 (to indicate the total number of lines to be skipped) or the letter H (which causes the printer to skip to the head of the next form).

NOTE: The assembler automatically skips to the head of the form for each new segment.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	SKIP	9

In the sample coding above, the assembler is directed to skip 9 lines on the program listing.

Suffix  
SFX

A	B	C	D	OS/2000

The SFX statement directs the assembler to append the single-character suffix in the operands field to each tag of five characters or less contained in the following coding. This technique enables the programmer to assign unique tags for each segment of a program and thus guard against double definition of a tag between distinct segments of a program. When inter-segment referencing within a program is required, six-character tags may be assigned.

This operation continues until the occurrence of another SFX statement with a blank operands field, or until the END statement is encountered.

EASYCODER C, D, and OS/2000

The letters SFX are placed in the op code field. A single-character suffix is written in the operands field.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	Y 1 P 2 R 3	M 4 R 5	L 6 O C A T I O N	O P E R A T I O N C O D E	O P E R A N D S	
					14 15 20 21	62 63 80
1				SFX	E	
2			TOTAL	A	FICA+TOTAX-20	

In the above example, the assembler interprets the Add instruction following the SFX statement as: TOTALE A FICAE+TOTAXE-20.

Repeat  
REP

A	B	C	D	OS/2000

This statement directs the assembler to repeat the following data formatting statement the number of times specified in the operands field. The number of times a statement is repeated includes the original statement and may not exceed 63. The assembler repeats the statement without variation, except that any entry in the location field is not repeated.

EASYCODER C, D, and OS/2000

The letters REP are written in the op code field. The operands field designates the number of times the following statement is to be repeated (including the original statement).

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	Y 1 P 2 R 3	M 4 R 5	L 6 O C A T I O N	O P E R A T I O N C O D E	O P E R A N D S	
					14 15 20 21	62 63 80
1				REP	6	
2			OCT56	DCW	#2C6	

In the sample statement above, REP is employed to define six identical constants of octal value 6000.

Generate  
GEN

A	B	C	D	OS/2000

This statement directs the assembler to generate the instruction which follows a specified number of times, incrementing or decrementing the operands of the instruction as specified by the operands field of the GEN statement. The GEN statement can apply to machine instructions

with formats containing a single address, both addresses, a single address and one variant character, or both addresses and one variant character (only one variant character is allowed).

EASYCODER C, D, and OS/2000

The letters GEN are written in the op code field. The operands field contains the parameter specifying the number of times the statement (which follows) is to be generated, including the original statement. This number is followed by a modifier for each operand in the model statement. These modifiers specify the increment (from 0 to +63) or decrement (from -63 to 0) to be applied to each of the operands each time the statement is generated. There must be a modifier for each operand in the model statement (including the variant character, if any), and the modifiers must appear in the same order as the operands. If no modification is desired, 0 is entered as the modifier.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	T Y P E	M A R K	LOCATION	OPERATION CODE	OPERANDS																																																																										
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
1				GEN	10	,	+4	,	+6	,	0																																																																				
2			SWC	BCE	SEL	,	TABLE	,	8																																																																						
3																																																																															
4			TABLE	RESV	60																																																																										

In the example above, the GEN statement generates a series of 10 instructions that will branch to a location SEL, SEL+4, SEL+8, ..... or SEL+36, provided that an 8 is present in the first character of the corresponding item in a table containing ten 6-character items. The tag SWC is assigned to the leftmost character of the first generated instruction. The GEN statement itself must not be tagged.

NOTE: The second BCE instruction generated by the example is BCE/SEL+4, TABLE +6, 8; the third instruction generated is BCE/SEL+8, TABLE +12, 8; and so on. The tenth instruction generated is BCE/SEL+36, TABLE+54, 8.

Set Line Number SETLIN
---------------------------

A	B	C	D	OS/2000

This instruction is used to control the generation of line numbers by the assembler.

EASYCODER C, D, and OS/2000

The letters SETLIN are written in the op code field, while the first five columns of the operand field contain the desired line number. The assembler replaces the contents of the line

number generation counter with the number in the operands field of the SETLIN statement. This statement is effective only when the assembler is generating line numbers. It is important to note that all of the first five columns in the operands field must be punched with a decimal number (i.e., leading 0's are required).

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	Y	H	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1-5	6-80
1	05	01	0			SETLIN	00080	
2	05	02	0			B	00	
3								

In the example above, the SETLIN statement causes the instruction which follows it (B/00) to be assigned a line number of 00080.

Set Out-of-Sequence Base  
XBASE

A	B	C	D	OS/2000

The XBASE statement establishes the out-of-sequence base (OSB). As its name implies, the OSB is a base address for the storage of out-of-sequence coding. Such coding may be allocated or referred to (1) by means of the address code ' (apostrophe) in the location field or (2) by means of the address code ' (apostrophe) in the operands field.

### EASYCODER C, D, and OS/2000

The letters XBASE are written in the op code field. The operands field contains the value (absolute or symbolic) to which the assembler is directed to set the out-of-sequence base (OSB). If a symbolic tag appears in the operands field it must have appeared in the location field of a previous source-program entry.

## EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	Y	H	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1-5	6-80
1						XBASE	500	
2					'275	DCW	@CON@	
3								



In the above example, the out-of-sequence base (OSB) is set to 500 by the XBASE statement. When the second entry is encountered, the assembler assigns the rightmost character of the constant CON to location 775 (500 + 275).

Range
RANGE

A	B	C	D	OS/2000

RANGE statements are used in conjunction with the OS/2000 Supervisor's Call/Cancel Controller. A RANGE statement is required in every root program and in every subprogram to specify the range of consecutive main memory locations required by the program unit. A RANGE statement can appear anywhere within a program unit.

OS/2000 EASYCODER

A RANGE statement is coded as shown below.

RANGE is entered in columns 15 through 19.

The A-address, which begins in column 21, is the direct address (relative to the program) of the lowest main memory location to be used by the program unit. The A-address is followed by a comma.

The B-address is the direct address (relative to the program) of the highest main memory location to be used by the program unit.

The programmer must insure that the RANGE statement delimits a memory area adequate for the program unit as well as any data storage areas, tables, buffers, etc. required by the program unit.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS										
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
			RANGE	*	FIN									
2														
3														

The OS/2000 Supervisor's Call/Cancel Controller uses the difference between the resolved values of the RANGE statement's two operands to ascertain the size of the memory area required by the program unit. At execution time, this size governs whether a program unit can be loaded into memory and where the program unit is loaded.

The macro calls \$CALL, /CALL, \$CANCL, and \$EXIT provide the linkage between the EasyCoder program units and the Supervisor's Call/Cancel Controller. This subject is described in detail in the Honeywell publication OS/2000 Supervisory Components, Order No. AH23.

Clear CLEAR
----------------

A	B	C	D	OS/2000

The CLEAR statement enables the programmer to specify an area of memory which is to be cleared of punctuation before the object program is loaded. The data bits are also cleared to 0's or to a given character. It is not necessary to clear areas which will be used to store the object program.

EASYCODER A

The op code field contains the letters CLEAR, while the operands field contains the addresses (either absolute or symbolic) of the first (low) and last (high) locations in an area to be cleared. If a comma is written immediately following the second address, the character written in the column after the comma is loaded into all locations in the cleared area. If two addresses are written in the operands field and are not followed by a comma and a character, the specified area is cleared to 0's.

A number of CLEAR statements may be written (in sequence) immediately preceding the END statement, provided that the total number of HSM, CLEAR, and END statements does not exceed ten.

NOTE: The 80-character loading area specified in the END statement must never be cleared.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1	CLEAR	CAMT, EAMT	
2			
3	CLEAR	334, 379, J	

The first CLEAR statement above specifies that the area beginning at the location tagged CAMT and ending at the location tagged EAMT is to be cleared to zeros. The second CLEAR statement clears the area beginning at location 334 and ending at 379 to 46 J's.

EASYCODER B

The op code field contains the letters CLEAR, while the operands field contains the addresses (either absolute or symbolic) of the first (low) and last (high) locations in an area to be cleared. If a comma is written immediately following the second address, the character written in the column after the comma is loaded into all locations in the cleared area. If two

addresses are written in the operands field and are not followed by a comma and a character, the specified area is cleared to 0's.

A maximum of nine CLEAR statements may be included in a program. In addition, no coding may appear between the last symbolic CLEAR statement and the END statement.

NOTE: The loading area specified in the END statement must never be cleared.

See the sample statements given above for Easycoder A.

### EASYCODER C, D, and OS/2000

The op code field contains the letters CLEAR, while the operands field contains the addresses (either absolute or symbolic) of the first (low) and last (high) locations in an area to be cleared. If a comma is written immediately following the second address, the character written in the column after the comma is loaded into all locations in the cleared area. If two addresses are written in the operands field and are not followed by a comma and a character, the specified area is cleared to 0's. As many CLEAR statements as necessary can be included in a program.

NOTE: The programmer must exercise caution in the physical placement of the CLEAR statement, as the clearing is performed by the Loader at the time the CLEAR statement is encountered.

See the sample statements given above for Easycoder A.

End
END

A	B	C	D	OS/2000

The last source program statement must be the END statement, which indicates to the assembler that the end of the source program has been reached.

### EASYCODER A

The location field may contain an address (either absolute or symbolic) which specifies the initial location in an 80-character loading area. If the location field is left blank, the assembler automatically reserves an 80-character loading area following the location assigned to the last character in the object program.

The op code field contains the letters END. If it is desired to execute the object program immediately after loading, the operands field must contain the address (either absolute or symbolic) at which the object program is to begin.

# EASYCODER

CODING FORM

PROBLEM _____		PROGRAMMER _____		DATE _____		PAGE _____ OF _____	
CARD NUMBER	Y W P R	LOCATION	OPERATION CODE	OPERANDS			
1 2 3 4 5 6 7 8		14 15	20 21	62 63	80		
		END	OBJECT				

The END statement above specifies that the object program (beginning at the address tagged OBJECT) is to be executed immediately after loading. Since the location field is blank, the assembler will reserve an 80-character loading area following the location assigned to the last character in the object program.

## EASYCODER B

The method of coding this statement depends on which output format has been specified in the program header statement.

1. Output in self-loading format: The location field may contain an address (either absolute or symbolic) which specifies the initial location in an 80-character loading area. If the location field is left blank, the assembler automatically assigns an 80-character loading area following the location assigned to the last character in the object program.

The op code field contains the letters END, while the operands field contains the address (either absolute or symbolic) to which the Loader branches when loading has been completed.

- NOTES:
1. The programmer should ensure that the loading area does not span two 4K memory banks.
  2. During the loading process, the object program must not use the loading area. However, the area may be used following program loading.
  3. When literals are used, the programmer must specify a loading area that does not coincide with the memory area occupied by literals.

2. Output in BRT format: the op code field contains the letters END, while the operands field contains the address (either absolute or symbolic) to which the Loader branches when loading has been completed. When BRT format is specified, all other fields of the END instruction are ignored by the assembler.

- NOTES:
1. The loading area is automatically assigned by the Loader.
  2. With EasyCoder B, the total of the numbers of Execute, Literal Origin, and End statements must not exceed 31.



## SECTION VIII INSTRUCTIONS

### INTRODUCTION

A Series 2000 computer operates under the direction of instructions in the stored program. For descriptive purposes, these instructions are classified into six functional categories: (1) Arithmetic; (2) Logic; (3) Control; (4) Interrupt Control; (5) Editing; and (6) Input/Output.

All instructions are described in the following standard format:

Title:	The title describes the instruction. It appears in the left-hand margin of a page, along with the mnemonic operation code used in the Easycode symbolic programming language and the octal value of the instruction's machine-language code.  If an instruction is included in an optional feature, that feature number accompanies the title.
Format:	This is a tabular representation of all formats which may be used when coding the instruction.
Function:	The function of the instruction is described in terms of each format in which it can be coded.
Word Marks:	The effect of word marks with regard to data fields is specified.
Address Registers after Operation:	The contents of the address registers are indicated for each of the instruction's formats.
Notes:	This is additional information pertaining to the operation.
Examples:	Practical applications of the instruction in its various formats are described and illustrated as symbolic program entries.

Formulas for calculating instruction execution times are presented in Appendix C.

Table 8-1 lists the abbreviations and symbols used in the description of the instructions. These symbols used only with specific instructions are preceded by the title of the instruction to which they pertain.

Table 8-1. Symbology Used in Series 2000 Instruction Descriptions

SYMBOL	MEANING
A	A address of the instruction
B	B address of the instruction
N <sub>i</sub>	Number of characters in the instruction
N <sub>a</sub>	Number of characters in the A field
N <sub>b</sub>	Number of characters in the B field
N <sub>w</sub>	Number of characters in the A or B field, whichever is smaller
NXT	Address of next sequential instruction
JI	Address of next instruction if a branch occurs
A <sub>p</sub>	The previous setting of the A-address register (AAR)
B <sub>p</sub>	The previous setting of the B-address register (BAR)
Divide	
N <sub>dd</sub>	Number of digits in the dividend
Move and Translate	
N <sub>ct</sub>	Number of characters translated
Move Item and Translate	
N <sub>ut</sub>	Number of information units translated
CSR <sub>p</sub>	Previous contents of the change sequence register (CSR)
NA <sub>u</sub>	Number of six-bit character locations occupied by each A-item information unit (1 or 2)
NB <sub>u</sub>	Number of six-bit character locations occupied by each B-item information unit (1 or 2)
Load Control Registers	
(A)	Contents of the field specified by the A address.
Table Lookup	
L <sub>ta</sub>	The location in the table immediately to the left of the argument (or short field) that terminated the search.

## ARITHMETIC OPERATIONS

Series 2000 add operations (binary addition, decimal addition) treat the B-operand as the augend and the A-operand as the addend. The subtract operations (binary subtraction, decimal subtraction) treat the A-operand as the subtrahend and the B-operand as the minuend. The result of each operation is stored in the B-field. These elements are summarized in Table 8-2, where a character enclosed in parentheses indicates the contents of that field.

Table 8-2. Series 2000 Add and Subtract Operations

Addition	Subtraction
( B ) augend	( B ) minuend
+ ( A ) addend	- ( A ) subtrahend
-----	-----
( B ) sum	( B ) difference

### BINARY ADDITION

The Binary Add instruction combines the corresponding bits of the augend and addend and produces a binary sum which is stored in the B-field. This process can be most readily analyzed on a column-by-column basis. For any column in the addition, three variables are significant to the sum: the augend digit, the addend digit, and the carry from the next lower-order column. For any column, the result is fully expressed by a sum digit (1 or 0) and either a carry or no-carry to the next higher-order column. Table 8-3 lists all the possible combinations of these variables.

Table 8-3. Binary Addition Table

Previous Carry	0	0	0	0	1	1	1	1
Augend	0	1	1	0	0	1	1	0
Addend	0	1	0	1	0	1	0	1
Sum	0	0	1	1	1	1	0	0
Carry	0	1	0	0	0	1	1	1

### BINARY SUBTRACTION

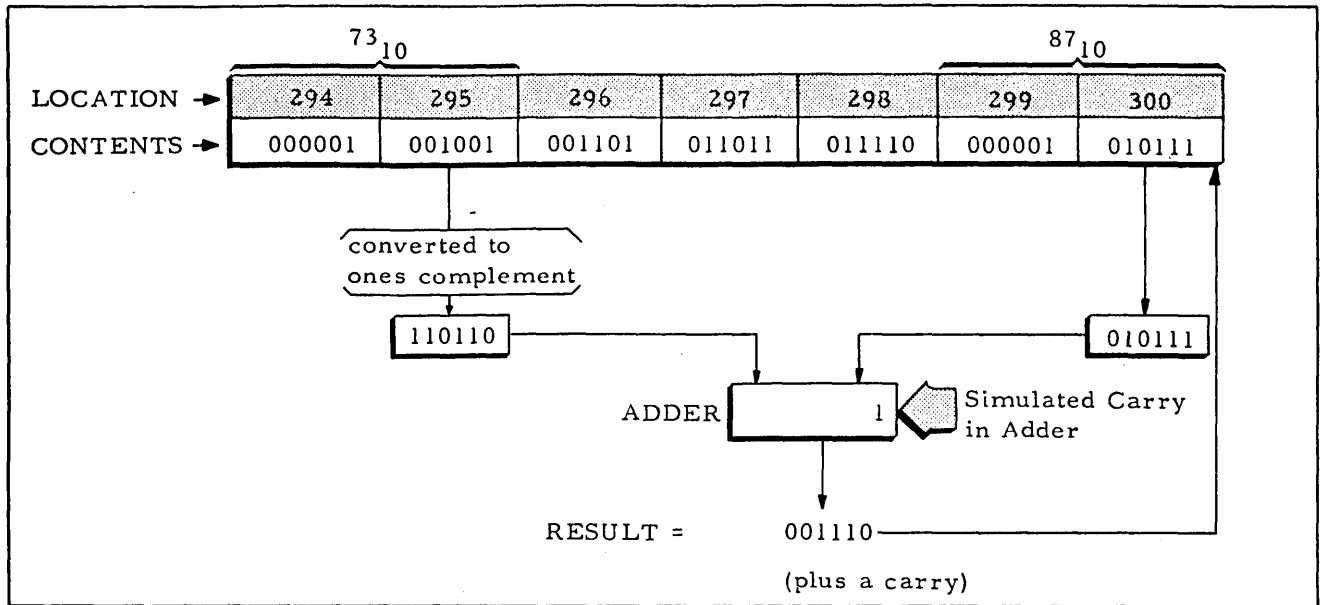
The Binary Subtract instruction performs, in effect, twos-complement arithmetic.<sup>1</sup> When this instruction is executed, each six-bit character of the subtrahend is converted to its ones complement<sup>2</sup> and added to the corresponding character in the minuend, adding from right to left.

<sup>1</sup>The twos complement of a binary number is formed by subtracting the number from a field of all one bits and adding one to the low-order digit of the difference.

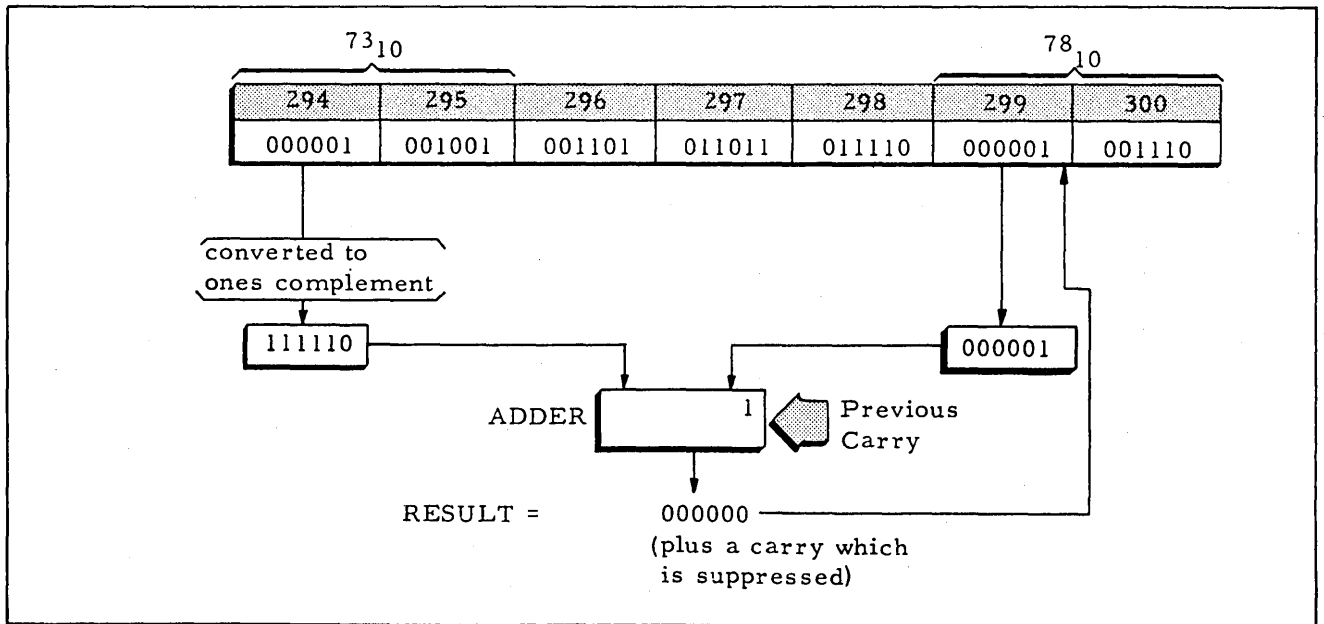
<sup>2</sup>The ones complement of a binary number is formed by subtracting the number from a field of all one bits.







First Addition



Second Addition

The result of the operation (14<sub>10</sub>) is stored in the B field as shown below.

73 <sub>10</sub>	14 <sub>10</sub>					
294	295	296	297	298	299	300
000001	001001	001101	011011	011110	000000	001110

## DECIMAL ADDITION

The Add instruction performs either a true add or a complement add, depending upon the algebraic signs of the operands. The sign of an operand is determined by the combination of zone bits in the units position of that field. The four possible zone bit configurations and the signs they represent are shown in Table 8-4.

Table 8-4. Algebraic Signs in Decimal Addition

SIGN	ZONE BITS		SIGN	ZONE BITS	
	B-Bit	A-Bit		B-Bit	A-Bit
+	0	0	-	1	0
	1	1			
	0	1			

### True Add

A true add is performed if the signs of the A and B fields are alike. The result of the addition is stored in the B field with the same zone bit configuration that was originally in the B field (see Figure 8-1). Zone bits in all B-field locations (except for the units position) are set to zeros. A-field zone bits (except for the units position) are ignored.

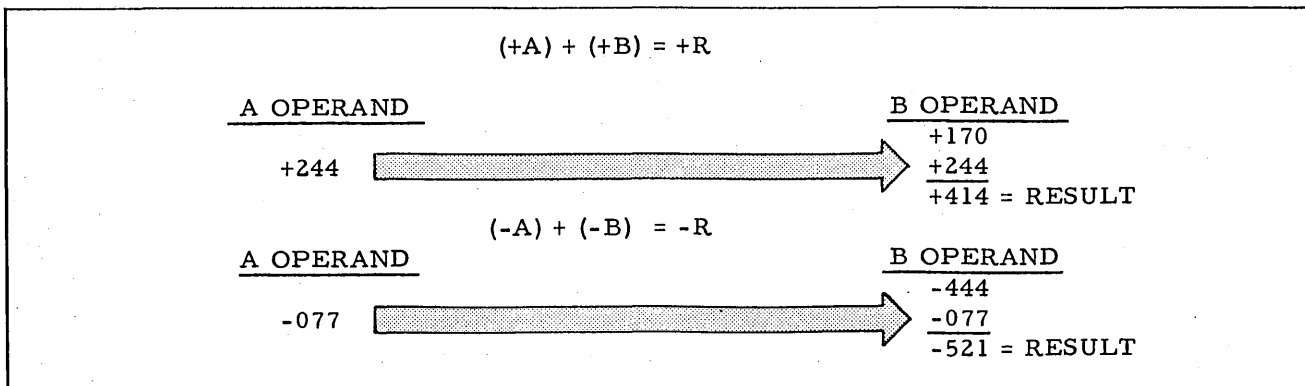


Figure 8-1. True Add Examples

### Complement Add

If the operand signs are not alike, the instruction performs a complement add: the A operand is converted to its tens complement<sup>1</sup> and added to the B operand. The machine automatically initiates a test to determine whether a carry was generated by the high-order addition.

<sup>1</sup>The tens complement of a decimal number is formed by subtracting the number from a field of all nines and adding one to the low-order digit of the difference.

The presence of a carry indicates that the result in the B-field is a true answer, and the operation is terminated with the normalized sign of the B-field as the sign of the result (see Figure 8-2).<sup>1</sup> B-field zone bits (except for the units position) are set to 0's.

The absence of a carry indicates that the A-operand was algebraically larger than the B-operand and that the result is stored in its tens-complement form. A recomplement cycle is performed automatically to convert the result of its true form. The sign of the result is changed during this recomplement cycle. Figure 8-2 illustrates complement add operations with and without recomplementation.

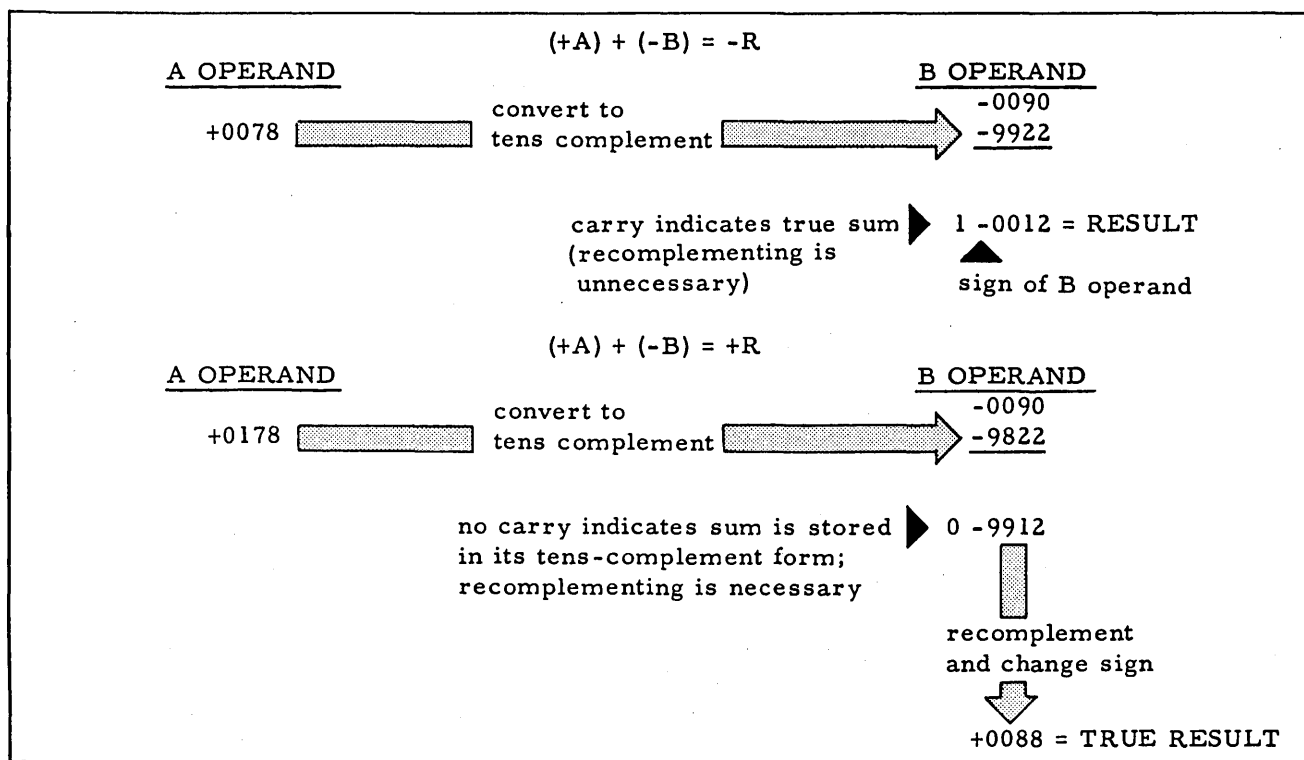


Figure 8-2. Complement Add Examples

### DECIMAL SUBTRACTION

The Subtract instruction is analogous to the Add instruction with the exception that before the operands are combined, the sign of the A-operand is changed. Thus, if the initial sign of the A-operand is equal to that of the B-operand, the operands are combined by a complement add. If, on the other hand, the initial sign of the A-operand is not equal to that of the B-operand, the operands are combined by a true add.

A summary of decimal arithmetic operations is presented in Table 8-5.

<sup>1</sup>Normalized signs are expressed by the following zone bit configurations: plus = 01, minus = 10.

Table 8-5. Decimal Arithmetic Sign Conventions

OPERATION	A-FIELD SIGN	B-FIELD SIGN	TYPE OF ADD	SIGN OF RESULT
ADD	+	+	True	+ (Bit configuration of B)
		-	Complement	Normalized sign of the operand of greater value (- = 10, + = 01)
	-	+	Complement	
		-	True	-
SUBTRACT	+	-	True	-
		+	Complement	Normalized sign of the operand of greater value (- = 10, + = 01)
	-	-	Complement	
		+	True	+ (Bit configuration of B)

INDICATORS

Two indicators are set at the completion of every decimal add and subtract operation: the overflow indicator and the zero balance indicator. If a result is greater than the size of the B-field, the overflow indicator is turned on; if such a carry is not generated, the indicator is unchanged.<sup>1, 2</sup> The zero balance indicator signifies either a zero or a nonzero sum. When a decimal operation produces a result equal to 0 (regardless of sign), the zero balance indicator is turned on; when the result of the operation does not equal 0, the indicator is turned off (the indicator is always turned on at the beginning of execution of a decimal add or subtract instruction).

These indicators are also set by decimal multiply and divide operations. The overflow indicator is turned on when a Decimal Divide instruction is performed in which the divisor is equal to 0. The zero balance indicator is turned on if the product of a decimal multiply operation is equal to 0.

The settings of these indicators can be tested by a Branch on Condition Test instruction. This instruction automatically turns off the overflow indicator.<sup>2</sup> The zero balance indicator is not changed by the branch instruction to test it, but is changed only by the next decimal arithmetic instruction.

MULTIPLICATION

The Multiply instruction causes the signed decimal integer in the A field (the multiplicand) to be multiplied by the signed decimal integer (the multiplier) which is stored in the leftmost locations of the B field. The signed product is stored, right-justified, in the B field.

<sup>1</sup>Only a "true add" operation can turn the overflow indicator on (see Table 8-5).

<sup>2</sup>The overflow indicator is turned off only when tested by the BCT instruction.

The B field must be large enough to insure an adequate number of locations for the development and storage of the product. Its length is therefore defined as the number of locations in the multiplier, plus the number of locations in the multiplicand, plus one (see Figure 8-3).

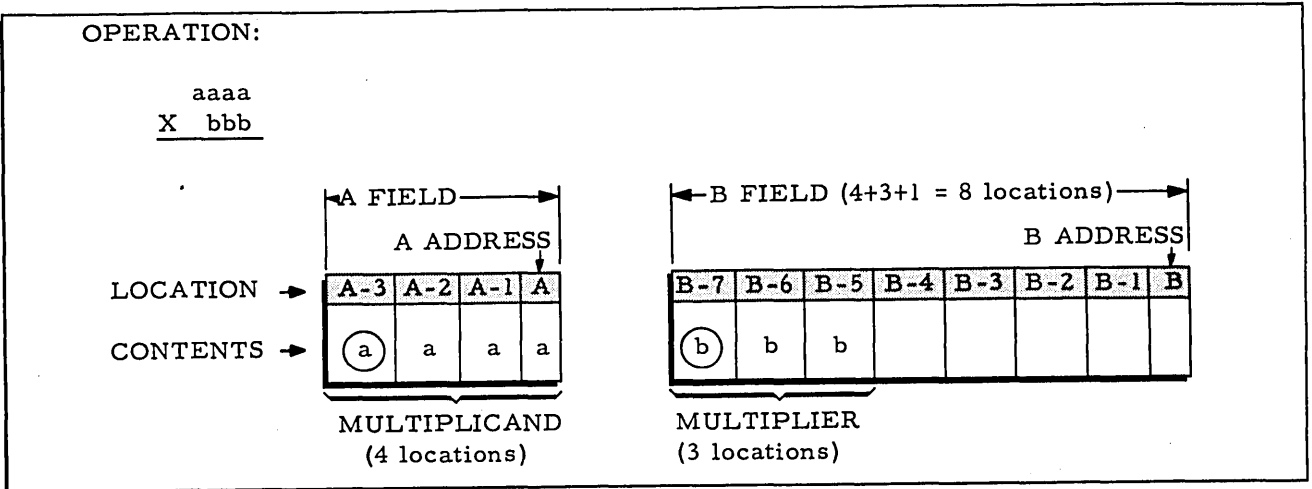


Figure 8-3. A- and B-Fields in Multiply Operation

Word marks are required in the leftmost locations of the multiplicand and the multiplier. All other locations in the B field must be clear of word marks. As shown in Figure 8-3, the rightmost location of the multiplier is defined as  $B - N_a - 1$ , where B is the B address and  $N_a$  is the number of locations in the A field.

The zone bits in the units positions of the multiplier and the multiplicand indicate the signs of the operands. The signs of these factors indicate the sign of the product according to the algebraic sign conventions shown in Table 8-6. The sign of the product is expressed in its normalized form (minus = 10, plus = 01).

Table 8-6. Multiply Sign Conventions

Sign of Multiplicand	+	-	+	-
Sign of Multiplier	+	-	-	+
Sign of Product	+	+	-	-

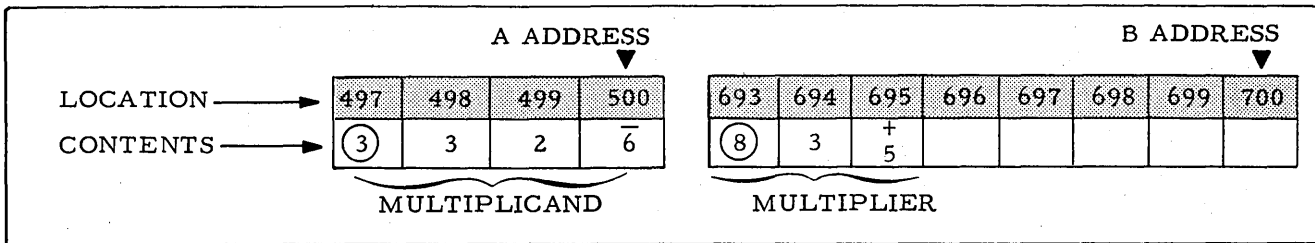
Consider the following Decimal Multiply instruction.

**EASYCODER**  
CODING FORM

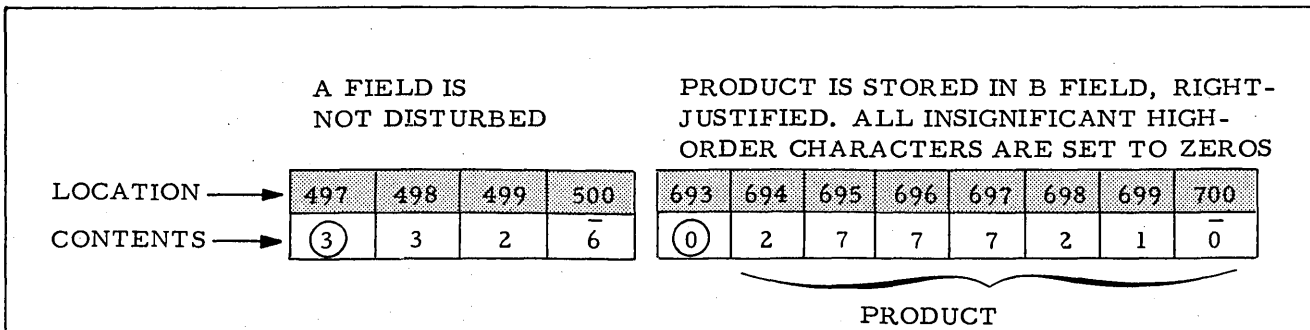
PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS										
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
			M	500,700										

Location 500 is the rightmost location of a four-character field. Location 700 is the rightmost location of an eight-character field. Location 695 (i.e.,  $700 - 4 - 1$ ) is the rightmost location of the multiplier.



The data in the A-field is multiplied by the data in the field whose rightmost location is 695, and the product is stored, right-justified, in the B-field. All B-field zone bits are cleared to 0's (except in the units position, which contains the sign of the product). At the end of the operation, the multiplier is no longer present in the leftmost positions of the B-field, since all B-field locations to the left of the most significant digit of the product are set to 0's. Thus, the multiplier should be preserved in another storage field if it is to be used more than once. The result of the multiply operation is shown below.



### DIVISION

The Divide instruction causes the signed decimal integer in the A-field (the divisor) to be divided into the signed decimal integer whose leftmost location is the B-address of the instruction (the dividend). The quotient is developed and stored in the leftmost locations of the B-field, and the remainder is stored in the rightmost locations of the B-field.<sup>1</sup> To insure an adequate number of storage locations for the development of the quotient, the length of the B-field is determined by adding 1 to the sum of the number of character locations in the divisor and dividend (see Figure 8-4).

The leftmost location of the dividend is defined by the B-address of the Divide instruction. The rightmost location (i. e., the units position) is the first character location to the right of the B-address to have one of its zone bits not equal to 0. As shown in Figure 8-4, all B-field locations to the left of the dividend must contain 0's prior to the divide operation.

A word mark is required in the leftmost location of the divisor. The dividend may or may not contain a word mark.

<sup>1</sup>Note that the B "field" in a divide operation does not define the B-operand but is a group of storage locations within which the B-operand (the dividend) is contained.

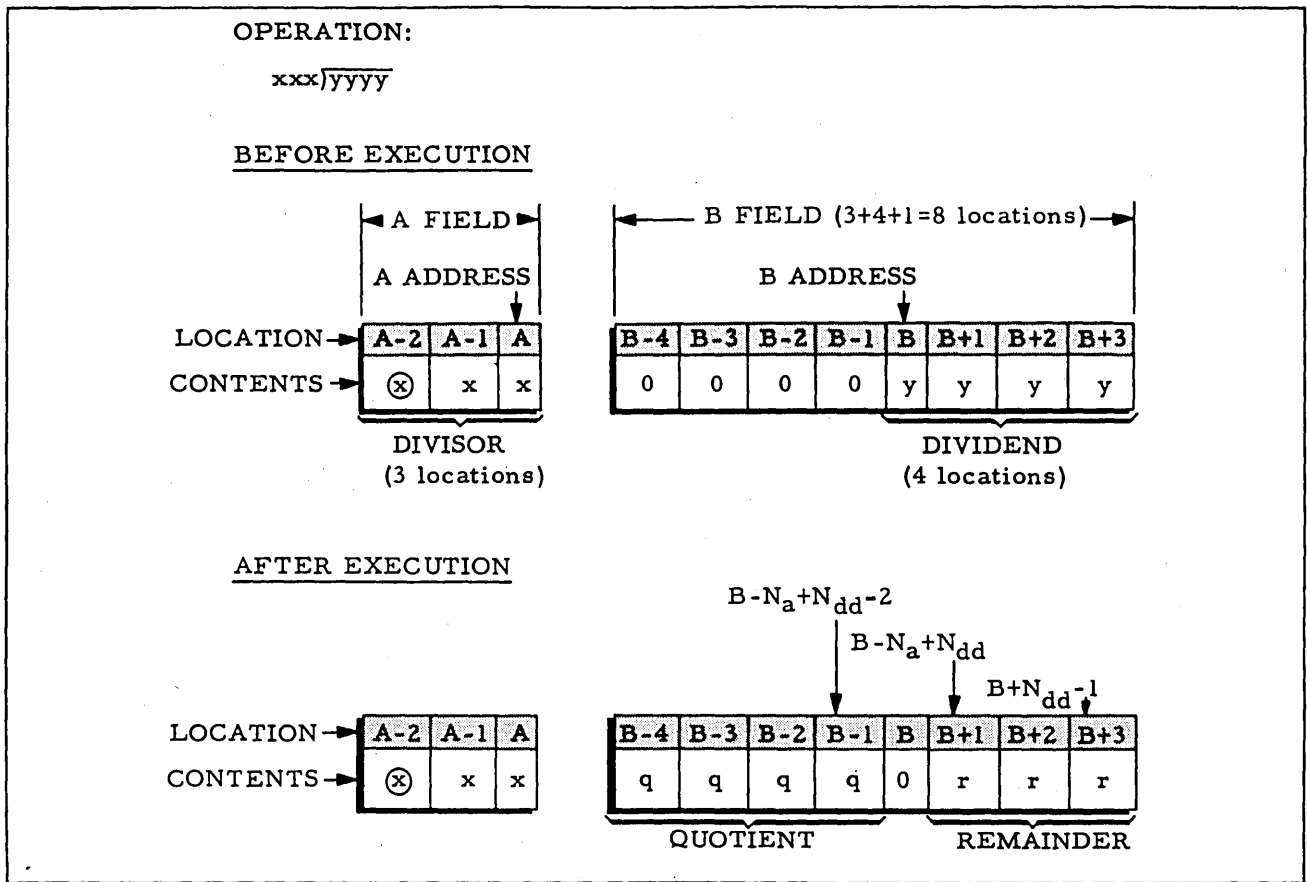


Figure 8-4. Factor Locations in Divide Operation

The signs of the operands are indicated by the zone bits in the units positions of the divisor and dividend. Algebraic sign control is used to determine the sign of the quotient (see Table 8-7). The sign of the quotient is expressed in its normalized form (minus = 10, plus = 01). The sign of the remainder is always the same as that of the dividend (in value if not in bit configuration); its form is normalized if the sign of the dividend is normalized.

Table 8-7. Divide Sign Conventions

Sign of Divisor	+	+	-	-
Sign of Dividend	+	-	+	-
Sign of Remainder	+	-	+	-
Sign of Quotient	+	-	-	+

Since the presence of a signed digit in the dividend specifies its rightmost location, the units position of the dividend must contain a normalized sign and the zone bits of all other dividend characters must be 0.



When division is completed, the signed decimal quotient is stored in the leftmost locations of the B-field; the units position of the quotient is in location  $B - N_a + N_{dd} - 2$ , where  $N_a$  is the number of locations in the A-field and  $N_{dd}$  is the number of locations in the dividend. The signed decimal remainder appears in locations  $B+N_{dd}-1$ ,  $B+N_{dd}-2$ , etc. through location  $B-N_a+N_{dd}$ . The character location separating the quotient and the remainder is cleared to 0 (see Figure 8-4).

In the following example, the divisor is a two-character field whose rightmost location is location 450 and the dividend is a four-character integer whose leftmost location is location 950.

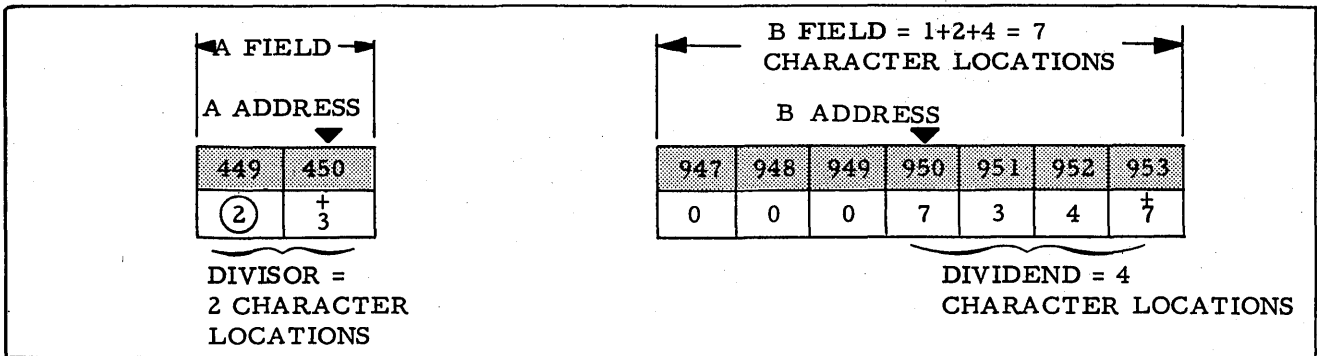
## EASYCODER

CODING FORM

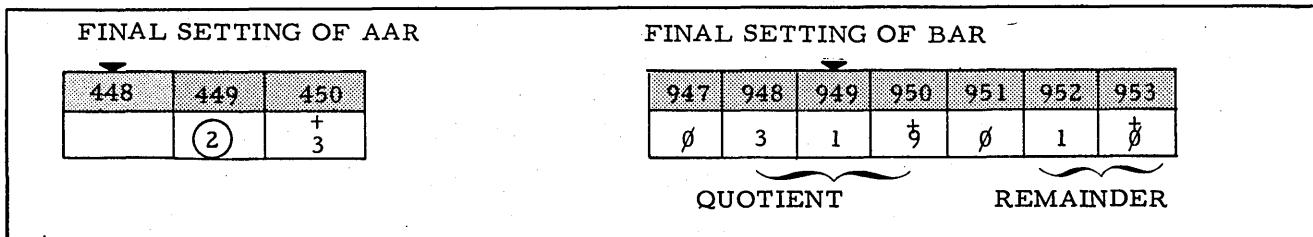
PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	V 1 2 3 4 5 6 7 8	M 9 10 11 12 13 14 15	R 16 17 18 19 20 21 22 23	LOCATION	OPERATION CODE	OPERANDS	
1					D	450,950	
2							
3							

The contents (+23) of the A-field are divided into the contents of the field (+7347) whose leftmost location is 950. The rightmost boundary of the dividend is determined by the first character location (location 953) to the right of location B whose zone bits are nonzero. This units position of the dividend therefore contains the sign of the dividend.



The quotient (+319) is stored in the leftmost character locations of the B-field. The units position of the quotient (location 950) is equal to  $B - N_a + N_{dd} - 2$ , or  $950 - 2 + 4 - 2$ . The remainder is stored in the rightmost locations of the B-field; its leftmost location (location 952) is equal to  $B - N_a + N_{dd}$ , or  $950 - 2 + 4$ ; its rightmost location (location 953) is equal to  $B + N_{dd} - 1$ , or  $950 + 4 - 1$ . The result of the operation is shown below.



## ARITHMETIC

- 8-15 ● ADD
- 8-16 ● SUBTRACT
- 8-18 ● BINARY ADD
- 8-19 ● BINARY SUBTRACT
- 8-20 ● ZERO AND ADD
- 8-22 ● ZERO AND SUBTRACT
- 8-23 ● MULTIPLY
- 8-25 ● DIVIDE



A	ADD	$36_8$
---	-----	--------

FORMAT	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

### FUNCTION

- Format a: The signed decimal data in the A field is added algebraically to the signed decimal data in the B field. The result is stored in the B field.
- Format b: The signed decimal data in the A field is added to itself. The result is stored in the A field.
- Format c: The signed decimal data specified by the contents of the A-address register (AAR) is added algebraically to the signed decimal data specified by the contents of the B-address register (BAR). The result is stored in the B field.

### WORD MARKS

- Format a: The B operand must have a defining word mark. It is this word mark that terminates the operation. The A operand must have a word mark only if it is shorter than the B operand. In this case, transfer of data from the A operand stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.
- Format b: The A operand must have a defining word mark.
- Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

### ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	A-N <sub>a</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>

### NOTES

- The algebraic sign control for the add operation is shown below.

A-FIELD SIGN	+	-	+	-
B-FIELD SIGN	+	-	-	+
TYPE OF ADD	True	True	Comp	Comp
SIGN OF RESULT	Sign of B field		Normalized sign of A or B field, whichever is greater (- = 10, + = 01)	

2. All zone bits in the result field are set to 0's except for the units position (i. e., the sign of the result).
3. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate nondecimal data. Particularly, if the four numeric bits of any character have a binary numeric value of 12 or more (octal 14, 15, 16, and 17), the character is treated as if it were a 0, though its zone bits are retained. The two remaining cases (octal 12 and 13) are unspecified.
4. The overflow and zero balance indicators are set by an add operation.
5. When the central processor is in the "S" mode of processing, the zone bits are not changed in any character other than the units position of the B-field.

**EXAMPLE**

Add Bond Deductions to Total Deductions.

<u>Description</u>	<u>Tag</u>
Bond Deductions	BDED
Total Deductions	TDED

**EASYCODER**

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	M	P	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8		14 15	20 21	62 63
					A		BDED, TDED

S	SUBTRACT	37 <sub>8</sub>
---	----------	-----------------

**FORMAT**

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

**FUNCTION**

**Format a:** The signed decimal data in the A-field is subtracted algebraically from the signed decimal data in the B-field. The result is stored in the B-field.

**Format b:** The signed decimal data in the A-field is subtracted from itself. The result is stored in the A-field. If the A-field sign is minus, the result is a minus zero. If the A-field sign is plus, the result is a plus zero (with normalized sign).

**Format c:** The signed decimal data specified by the contents of the A-address register (AAR) is subtracted algebraically from the signed decimal data specified by the contents of the B-address register (BAR). The result is stored in the B-field.

## WORD MARKS

Format a: The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand. In this case, transmission of data from the A-operand stops after the A-operand word mark is sensed. If the A-field is longer than the B-field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A-operand must have a defining word mark.

Format c: The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand.

## ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>
<u>Format a:</u>	<u>NXT</u>	<u>A-N<sub>w</sub></u>	<u>B-N<sub>b</sub></u>
<u>Format b:</u>	<u>NXT</u>	<u>A-N<sub>a</sub></u>	<u>A-N<sub>a</sub></u>
<u>Format c:</u>	<u>NXT</u>	<u>A<sub>p</sub>-N<sub>w</sub></u>	<u>B<sub>p</sub>-N<sub>b</sub></u>

## NOTES

1. Algebraic sign control for the subtract operation is summarized below.

A-FIELD SIGN	+	-	+	-
B-FIELD SIGN	+	-	-	+
TYPE OF ADD	Comp	Comp	True	True
SIGN OF RESULT	Normalized sign of A or B field, whichever is greater (- = 10, + = 01)		Sign of B field	

2. All zone bits in the result field are set to 0's except for the units position (i.e., the sign of the result).
3. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate nondecimal data. Particularly, if the four numeric bits of any character have a binary numeric value of 12 or more (octal 14, 15, 16, and 17), the character is treated as if it were a 0, though its zone bits are retained. The two remaining cases (octal 12 and 13) are unspecified.
4. The overflow and zero balance indicators are set by a subtract operation.
5. When the central processor is in the "S" mode of processing, the zone bits are not changed in any character other than the units position of the B-field.

## EXAMPLE

Subtract the contents of the five-character fields starting at location 940, 945, 950, and 955 from the contents of the eight-character fields starting at locations 648, 656, 664, and 672.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION	LOCATION	OPERATION CODE	OPERANDS	
				A	B
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80	
1	S		955,672		
2	S				
3	S				
4	S				

<b>BA</b>	BINARY ADD	34 <sub>8</sub>
-----------	------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

FUNCTION

- Format a: The data in the A-field is added in binary fashion, character by character, to the data in the B-field. The result is stored in the B field.
- Format b: The data in the A field is added, character by character, to itself. The result is stored in the A field.
- Format c: The data specified by the contents of the A-address register (AAR) is added, character by character, to the data specified by the contents of the B-address register (BAR). The result is stored in the B field.

WORD MARKS

- Format a: The B operand must have a defining word mark. It is this word mark that terminates the operation. The A operand must have a word mark only if it is shorter than the B operand. In this case the transmission of data from the A field stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.
- Format b: The A operand must have a defining word mark.
- Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	A-N <sub>a</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>





Format b: The A-operand must have a defining word mark.

Format c: The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	A-N <sub>a</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>

NOTES

1. The overflow and zero balance indicators are not set by a binary subtract operation.
2. Formats a. and c. can produce negative results. A negative result is stored in the B-field in its twos-complement form. In this case, the absolute numerical value of the result can be obtained by recomplementing the result stored in the B-field and adding 1. A negative result is detected only if the programmer provides appropriate coding to ascertain whether or not operands will produce such a result.

EXAMPLE

Zero the field starting at location TOTAL.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Z M P R	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8		14 15	20 21	62 63	80
		BS	TOTAL		

NOTE: Zone bits as well as numeric bits are cleared to 0 by this operation.

ZA	ZERO AND ADD	16 <sub>8</sub>
----	--------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	██████	██████████	██████████
b.	██████	██████████	
c.	██████		

## FUNCTION

- Format a: The data in the A-field is transferred, character-by-character, right to left, to the B-field. Zone bits in the B-field are set to 0 in all positions except the units position. The sign of the result field is based on the sign of the A-field (see note 1). If a high-order character of the A-field is transferred before the operation terminates, the remaining B-field characters are cleared to 0's.
- Format b: The data in the A-field is converted to an all-numeric format; i.e., the zone bits of all positions in the field except the units position are set to 0. The result remains in the A-field. The sign of the A-field is not changed by the operation (see note 1).
- Format c: The data specified by the contents of the A-address register (AAR) is transferred to the field specified by the contents of the B-address register (BAR). Zone bits in the B-field are set to 0 in all positions except the units position. The sign of the result field is based on the sign of the A-field (see note 1). If the high-order character of the A-field is transferred before the operation terminates, the remaining B-field characters are cleared to 0's.

## WORD MARKS

- Format a: The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand. In this case, transfer of data from the A-operand stops after the A-operand word mark is sensed. If the A-field is longer than the B-field, the high-order characters of the A-field that exceed the field length defined by the B-operand word mark are not processed.
- Format b: The A-operand must have a defining word mark.
- Format c: The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand.

## ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	A-N <sub>a</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>

## NOTES

1. A plus sign in the units position of the result field is always expressed in its normalized form (01).
2. B-field punctuation is not changed by this operation.
3. This instruction does not set the overflow and zero balance indicators.
4. When the central processor is in the "S" mode of processing and the four numeric bits of any character have a value of 14g or more, the character is treated as if it were a 0. The zero balance indicator is set or reset accordingly.

## EXAMPLE

Transfer the contents of the field tagged ORATE to the field tagged NRATE, setting all zone bits in NRATE (except in the units position) to 0's.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER		M A R K	L O C A T I O N	O P E R A T I O N C O D E	O P E R A N D S	
1	2				3	4
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49
50	51	52	53	54	55	56
57	58	59	60	61	62	63
64	65	66	67	68	69	70
71	72	73	74	75	76	77
78	79	80	81	82	83	84
85	86	87	88	89	90	91
92	93	94	95	96	97	98
99	100	101	102	103	104	105

<b>ZS</b>	<b>ZERO AND SUBTRACT</b>	17 <sub>8</sub>
-----------	--------------------------	-----------------

**FORMAT**

	O P C O D E	A A D D R E S S	B A D D R E S S
a.			
b.			
c.			

**FUNCTION**

**Format a:** The data in the A-field is transferred to the B-field and the sign is changed. Zone bits in the B-field are set to 0's in all positions except the units position. If the high-order character of the A-field is transferred before the operation terminates, the remaining B-field characters are cleared to 0's.

**Format b:** The data in the A-field is converted to an all-numeric format; i. e., the zone bits of all positions in the field except the units position are set to 0. The result remains in the A-field with its sign reversed.

**Format c:** The data specified by the contents of the A-address register (AAR) is transferred with the opposite sign to the field specified by the contents of the B-address register (BAR). Zone bits in the B-field are set to 0 in all positions except the units position. If the high-order character of the A-field is transferred before the operation terminates, the remaining B-field characters are cleared to 0's.

**WORD MARKS**

**Format a:** The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand. In this case, transfer of data from the A-operand stops after the A-operand word mark is sensed. If the A-field is longer than the B-field, the high-order characters of the A-field that exceed the field length defined by the B-operand word mark are not processed.

**Format b:** The A-operand must have a defining word mark.

**Format c:** The B-operand must have a defining word mark. The A-operand must have a word mark only if it is shorter than the B-operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	A-N <sub>a</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>

NOTES

1. A plus sign in the units position of the result field is always expressed in its normalized form (01).
2. B-field punctuation is not changed by this operation.
3. This instruction does not set the overflow and zero balance indicators.
4. When the central processor is in the "S" mode and the four numeric bits of any character have a value of 14<sub>8</sub> or more, the character is treated as if it were a 0. The zero balance indicator is set or reset accordingly.

EXAMPLE

Change the sign of the data in the field tagged PROFIT.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
1			ZS	PROFIT	
2					
3					
4					

**M** MULTIPLY 26<sub>8</sub>

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

FUNCTION

Format a: The signed decimal integer in the A-field is multiplied by the signed decimal integer in the leftmost locations of the B-field. The product is stored, right-justified, in the B-field.

Format b: The signed decimal integer in the A-field is multiplied by the signed decimal integer in the leftmost locations of the field specified by the contents of the B-address register (BAR). The product is stored, right-justified, in the B-field.

Format c: The signed decimal integer in the field specified by the contents of the A-address register (AAR) is multiplied by the signed decimal integer in the leftmost locations of the field specified by the contents of BAR. The product is stored, right-justified, in the B-field.

## WORD MARKS

Formats a, b, and c:

Word marks are required in the high-order locations of both the A- and B-fields. All other B-field locations must be clear of word marks.

## ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>
<u>Format a:</u>	NXT	A-N <sub>a</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	B <sub>p</sub> -N <sub>b</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>a</sub>	B <sub>p</sub> -N <sub>b</sub>

## NOTES

- The A-address of a Decimal Multiply instruction specifies the units position of the multiplicand. The B-address specifies a location which is N<sub>a</sub>+1 locations to the right of the multiplier, since the B-field must contain the multiplier plus enough additional locations (to the right of the multiplier) to provide for the development of the product. Thus, the total number of character locations in the B-field must be one greater than the sum of the number of characters in the multiplicand and the multiplier. For example, in a multiplication operation involving a three-character multiplier and a five-character multiplicand, nine positions (5+3+1) must be provided in the B-field.
- Algebraic sign control for the multiply operation is shown below. The sign of the product is expressed in its normalized form (-=10, +=01).

Sign of Multiplicand	+	-	+	-
Sign of Multiplier	+	-	-	+
Sign of Product	+	+	-	-

- The product is stored (right-justified) in the entire B-field, with the unused high-order positions of the B-field cleared to 0's. As a result of the operation, the multiplier (initially stored in the B-field) is destroyed. Therefore, if the multiplier is to be used more than once, it should be preserved in another storage field.
- The zero balance indicator is turned on if the product of the multiply operation is equal to 0; otherwise, the indicator is turned off by the operation.
- This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate nondecimal data. Particularly, if the four numeric bits of a character have a binary numeric value of 12 or more (octal 14, 15, 16, or 17), the character is treated as if it were a 0. The two remaining cases (octal 12 and 13) are unspecified.
- If the A- and B-operands overlap, then the results are unspecified.

EXAMPLE

Multiply the five-character field tagged CAND by the three-character field whose rightmost character location is six (5+1) less than the location tagged PROD. Store the result, right-justified, in PROD.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21
		M	CAND, PROD

D	DIVIDE	27 <sub>8</sub>
---	--------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	██████	████████	████████
b.	██████	████████	
c.	██████		

FUNCTION

Format a: The signed decimal integer whose leftmost location is B is divided by the signed decimal integer in the A-field. The quotient is stored in the leftmost locations of the B-field; the remainder is stored in the rightmost locations of the B-field.

Format b: The signed decimal integer whose leftmost location is specified by the contents of the B-address register (BAR) is divided by the signed decimal integer in the A-field. The quotient is stored in the leftmost locations of the B-field; the remainder is stored in the rightmost locations of the B-field.

Format c: The signed decimal integer whose leftmost location is specified by the contents of the B-address register (BAR) is divided by the signed decimal integer in the field specified by the contents of the A-address register (AAR). The quotient is stored in the leftmost locations of the B-field; the remainder is stored in the rightmost locations of the B-field.

WORD MARKS

Formats a, b, and c:

The A-operand (the divisor) must contain a word mark. The B-field may contain a word mark.

ADDRESS REGISTERS AFTER OPERATION (WHEN DIVISOR IS NOT EQUAL TO ZERO)

	SR	AAR	BAR	
<u>Format a:</u>	NXT	$A-N_a$	$B-N_a+N_{dd}-3$	} = Tens position of quotient field
<u>Format b:</u>	NXT	$A-N_a$	$B_p-N_a+N_{dd}-3$	
<u>Format c:</u>	NXT	$A_p-N_a$	$B_p-N_a+N_{dd}-3$	

When the divisor is equal to 0, the contents of the address registers are unspecified (see note 1).

NOTES

1. If the divisor is equal to plus or minus zero, the overflow indicator is turned on, division is not performed, and no memory locations are changed.
2. The length of the B-field is determined by adding 1 to the sum of the number of character locations in the divisor and the dividend (B-field length = 1+length of divisor + length of dividend).
3. The A-field (divisor) can be signed or unsigned; if it is unsigned, the divisor is assumed to be positive.
4. The dividend must contain a normalized sign (-=10, +=01) in the units position. The zone bits of all other characters in the dividend must be 0's. The proper signing of the dividend is therefore insured if the dividend is moved into the B-field by a Zero and Add instruction.
5. All high-order locations of the B-field which are not occupied by the dividend must contain 0's when division begins. These 0's can be automatically inserted if the Zero and Add instruction is used to move the dividend into the B-field as mentioned above.
6. The sign of the quotient follows algebraic sign rules as shown below. The sign of the remainder is the original sign of the dividend.

Sign of divisor	+	+	-	-
Sign of dividend	+	-	+	-
Sign of remainder	+	-	+	-
Sign of quotient	+	-	-	+

7. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate nondecimal data. Particularly, if the four numeric bits of a character have a binary numeric value of 12 or more (octal 14, 15, 16, or 17), the character is treated as if it were a 0. The two remaining cases (octal 12 and 13) are unspecified.
8. If the A- and B-operands overlap, then the results are unspecified.

EXAMPLE

Divide the four-character integer whose leftmost location is location 1000 by the three-character field whose rightmost location is location 500. Store the quotient in the leftmost locations of the field at 1000, and store the remainder in the rightmost locations of this field.

$$N_a \text{ (number of characters in divisor)} = 3 \quad N_{dd} \text{ (number of characters in dividend)} = 4$$

$$B \text{ (B address)} = 1000^1$$

$$\text{Units position of quotient } (B - N_a + N_{dd} - 2) = 1000 - 3 + 4 - 2 = \text{location } 999$$

$$\text{Units position of remainder } (B + N_{dd} - 1) = 1000 + 4 - 1 = \text{location } 1003$$

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8		14 15 20 21	62 63 80
	D	500, 1000	

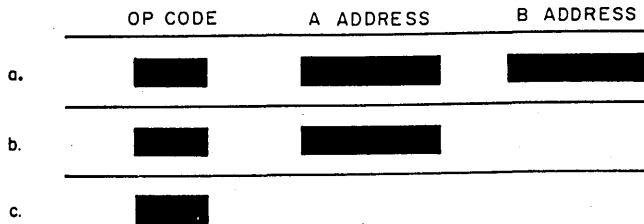
<sup>1</sup> Note that the B address is merely the leftmost location of the dividend, which is located in turn in a field large enough to contain the quotient and remainder.

**LOGIC**

- 8-28 ● EXTRACT
- 8-29 ● HALF ADD
- 8-30 ● SUBSTITUTE
- 8-32 ● COMPARE
- 8-34 ● BRANCH
- 8-35 ● BRANCH ON CONDITION TEST
- 8-39 ● BRANCH ON CHARACTER CONDITION
- 8-42 ● BRANCH IF CHARACTER EQUAL
- 8-44 ● BRANCH ON BIT EQUAL



FORMAT



FUNCTION

Format a: The data in the A field is combined bit-by-bit with the data in the B field, according to the following rules. The result is stored in the B field.

BIT IN A FIELD	BIT IN B FIELD	BIT IN RESULT FIELD
1	1	1
1	0	0
0	1	0
0	0	0

Format b: The data in the A field is combined bit-by-bit with the data specified by the contents of the B-address register (BAR), according to the rules stated above. The result is stored in the B field.

Format c: The data specified by the contents of the A-address register (AAR) is combined bit-by-bit with the data specified by the contents of BAR, according to the rules stated above. The result is stored in the B field.

WORD MARKS

Formats a, b, and c:

A word mark is required for the shorter of the two operands. The operation terminates when this word mark is sensed.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>w</sub>
<u>Format b:</u>	NXT	A-N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>

**EXAMPLE**

Remove all zone bits in the field tagged BASE by combining the contents of BASE with the contents of the field tagged CON. Each character in CON must have the following format:

Bit position    B A 8 4 2 1  
 Contents        0 0 1 1 1 1

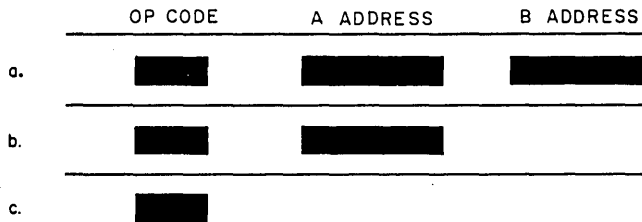
**EASYCODER**  
 CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	V 1 E R	M A C R O	LOCATION	OPERATION CODE	OPERANDS									
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
				EXT	CON, BASE									
2														

HA	HALF ADD (Exclusive OR)	30 8
----	----------------------------	---------

**FORMAT**



**FUNCTION**

**Format a:** The data in the A field is combined bit-by-bit with the data in the B field, according to the following rules. The result is stored in the B field.

BIT IN A FIELD	BIT IN B FIELD	BIT IN RESULT FIELD
1	1	0
1	0	1
0	1	1
0	0	0

**Format b:** The data in the A field is combined bit-by-bit with the data specified by the contents of the B-address register (BAR), according to the rules stated above. The result is stored in the B-field.

**Format c:** The data specified by the contents of the A-address register (AAR) is combined bit-by-bit with the data specified by the contents of BAR, according to the rules stated above. The result is stored in the B-field.

WORD MARKS

Formats a, b, and c:

A word mark is required for the shorter of the two operands. The operation terminates when this word mark is sensed.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>w</sub>
<u>Format b:</u>	NXT	A-N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>

EXAMPLE

Clear all the numeric bits in the field tagged SEVEN to 0's by combining the contents of SEVEN with the contents of the field tagged TOO. Do not change the zone bits in SEVEN. (The contents of each character in TOO are 00xxxx, where x equals the corresponding bit in SEVEN.)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	Z	W	LOCATION	OPERATION CODE	OPERANDS																																																																									
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
				HA		TOO, SEVEN																																																																									

SST	SUBSTITUTE	32 <sub>8</sub>
-----	------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	████	████	████
b.	████	████	████	
c.	████	████		
d.	████			

FUNCTION

Format a: The single character specified by the A address is compared bit-by-bit with the variant character and is moved to the location specified by the B address, according to the following rules:



2. Move the numeric portion of the character at location 256 to location 656.  
A variant of octal 17 provides the required variant bit configuration  
(i.e., 001 111).

## EASYCODER

CODING FORM

PROBLEM _____										PROGRAMMER _____										DATE _____										PAGE _____ OF _____									
CARD NUMBER		TYPE	LOCATION		OPERATION CODE		OPERANDS																																
1	2		3	4	5	6	7	8	14	15	20	21																											
								SST		256	656	17																											

C	COMPARE	33 <sub>8</sub>
---	---------	-----------------

### FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	██████████
b.	████	████████	
c.	████		

### FUNCTION

- Format a:** The data in the B field is compared bit-by-bit with the data in the A field. The comparison turns on indicators that can be interrogated by subsequent Branch instructions. The indicators are reset by the next Compare instruction.
- Format b:** The data specified by the contents of the B-address register (BAR) is compared bit-by-bit with the data in the A field. This operation turns on indicators which can be tested by subsequent Branch instructions. The indicators are reset by the next Compare instruction.
- Format c:** The data specified by the contents of BAR is compared bit-by-bit with the data in the field specified by the contents of the A-address register (AAR). The comparison turns on indicators that can be interrogated by subsequent Branch instructions. The indicators are reset by the next Compare instruction.

### WORD MARKS

#### Formats a, b, and c:

The word mark associated with the B-operand terminates the operation. The A-operand must have a word mark only if it is shorter than the B-operand. In this case, transmission of data from the A-field stops after the A-operand word mark is sensed, and the remaining characters of the B-operand are compared with 0's. If the A-operand is longer than the B-operand, the characters of the A-operand that exceed the field length defined by the B-operand word mark are not processed.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>b</sub>
<u>Format b:</u>	NXT	A-N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>b</sub>

NOTES

1. All characters that can appear in storage can be compared. The ascending order of characters is listed in Appendix B.
2. Both fields must have exactly the same bit configurations to be equal. For example, plus zero is not equal to minus zero.
3. Comparison results and associated branch conditions are listed below.

COMPARISON RESULT	BRANCH CONDITION
B < A	Low Compare
B = A	Equal Compare
B ≤ A	Low or Equal Compare
B > A	High Compare
B ≠ A	Unequal Compare
B ≥ A	High or Equal Compare

EXAMPLE

Compare item number with 4000. If item number equals 4000, continue the program in sequence; otherwise, branch to location NITEM.

<u>Description</u>	<u>Tag</u>
Item Number	ITEM
4000	CON4

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63
1			C	CON4, ITEM
2			BCT	NITEM, 45

B	BRANCH	65 <sub>8</sub>
---	--------	-----------------

FORMAT

OP CODE	A ADDRESS	B ADDRESS	VARIANT
[REDACTED]	[REDACTED]		

FUNCTION

The Branch instruction causes the program to branch to the location specified by the A address and to store the contents of the sequence register (SR) in the B-address register (BAR). It is used to interrupt normal program sequence and to continue the program at any desired point, without testing for specific conditions. Thus, this instruction is frequently referred to as an "unconditional branch."

WORD MARKS

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
JI (A)	A	NXT

NOTES

1. The A address is placed in AAR during the extraction of this instruction, preserving any active high-order bits in AAR. When the instruction is executed, the entire contents of AAR specify the address to which the program branches. Also, the entire contents of SR are stored in BAR during the execution phase.
2. The contents of the variant register are unspecified following the execution of this instruction. Therefore an instruction requiring a variant character must not be chained following a Branch instruction.

EXAMPLE

Select the next instruction from the location tagged SUB6.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	VARIANT	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8		14 15 20 21		62 63 80
			B	SUB6

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	████████	████████	████
b.	████			

FUNCTION

Format a: The variant character specifies a condition indicator or a SENSE switch to be tested. If the condition being tested is present, the program branches to the location specified by the A address and the contents of the sequence register (SR) are stored in the B-address register (BAR). If the condition specified by the variant character is not present, the program continues in sequence. Tables 8-8 and 8-9 list the valid variant characters and the conditions they test.

Format b: If the condition specified by the previous variant character is present, the program branches to the location specified by the contents of the A-address register (AAR) and the contents of SR are stored in BAR. If the conditions being tested is not present, the program continues in sequence. Tables 8-8 and 8-9 list the valid variant characters and the conditions they test.

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTER AFTER OPERATION

	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B	NO BRANCH
			P	
Format b:	JI (A <sup>P</sup> )	A <sup>P</sup>	NXT	BRANCH
	NXT <sup>P</sup>	A <sup>P</sup>	B	NO BRANCH
		P	P	

NOTES

1. If the overflow indicator is tested and an overflow condition exists, the indicator is automatically turned off as a result of being tested. In all other cases, the indicator tested is not affected as a result of the test.
2. The comparison indicators are:
  - a. set by the Compare instruction;
  - b. set by the Table Lookup instruction;
  - c. stored (and cleared) by the Store Variant and Indicators instruction;



Table 8-8. SENSE Switch Test Conditions for BCT Instruction

Variant Character (Octal)	Branch On
00	Unconditional
01	SENSE Switch 1 On
02	SENSE Switch 2 On
03	SENSE Switches 1 <u>and</u> 2 On
04	SENSE Switch 3 On
05	SENSE Switches 1 <u>and</u> 3 On
06	SENSE Switches 2 <u>and</u> 3 On
07	SENSE Switches 1, 2, <u>and</u> 3 On
10	SENSE Switch 4 On
11	SENSE Switches 1 <u>and</u> 4 On
12	SENSE Switches 2 <u>and</u> 4 On
13	SENSE Switches 1, 2, <u>and</u> 4 On
14	SENSE Switches 3 <u>and</u> 4 On
15	SENSE Switches 1, 3, <u>and</u> 4 On
16	SENSE Switches 2, 3, <u>and</u> 4 On
17	SENSE Switches 1, 2, 3, <u>and</u> 4 On
20	Unconditional
21	SENSE Switch 5 On
22	SENSE Switch 6 On
23	SENSE Switches 5 <u>and</u> 6 On
24	SENSE Switch 7 On
25	SENSE Switches 5 <u>and</u> 7 On
26	SENSE Switches 6 <u>and</u> 7 On
27	SENSE Switches 5, 6, <u>and</u> 7 On
30	SENSE Switch 8 On
31	SENSE Switches 5 <u>and</u> 8 On
32	SENSE Switches 6 <u>and</u> 8 On
33	SENSE Switches 5, 6, <u>and</u> 8 On
34	SENSE Switches 7 <u>and</u> 8 On
35	SENSE Switches 5, 7, <u>and</u> 8 On
36	SENSE Switches 6, 7, <u>and</u> 8 On
37	SENSE Switches 5, 6, 7, <u>and</u> 8 On

**NOTE:** When testing for a multiple SENSE switch condition, a branch occurs only if all of the specified conditions are met.

Table 8-9. Indicator Test Conditions for BCT Instruction

Variant Character (Octal)	Branch On
40	Do not branch
41	$B < A$ (Low Compare)
42	$B = A$ (Equal Compare)
43	$B \leq A$ (Low or Equal Compare)
44	$B > A$ (High Compare)
45	$B \neq A$ (Unequal Compare)
46	$B \geq A$ (High or Equal Compare)
47	Unconditional
50	Overflow ✓
51	Overflow <u>or</u> $B < A$
52	Overflow <u>or</u> $B = A$
53	Overflow <u>or</u> $B \leq A$
54	Overflow <u>or</u> $B > A$
55	Overflow <u>or</u> $B \neq A$
56	Overflow <u>or</u> $B \geq A$
57	Unconditional
60	Zero Balance
61	Zero Balance <u>or</u> $B < A$
62	Zero Balance <u>or</u> $B = A$
63	Zero Balance <u>or</u> $B \leq A$
64	Zero Balance <u>or</u> $B > A$
65	Zero Balance <u>or</u> $B \neq A$
66	Zero Balance <u>or</u> $B \geq A$
67	Unconditional
70	Overflow <u>or</u> Zero Balance
71	Overflow <u>or</u> Zero Balance <u>or</u> $B < A$
72	Overflow <u>or</u> Zero Balance <u>or</u> $B = A$
73	Overflow <u>or</u> Zero Balance <u>or</u> $B \leq A$
74	Overflow <u>or</u> Zero Balance <u>or</u> $B > A$
75	Overflow <u>or</u> Zero Balance <u>or</u> $B \neq A$
76	Overflow <u>or</u> Zero Balance <u>or</u> $B \geq A$
77	Unconditional

**NOTE:** When testing for a multiple indicator condition, a branch occurs if any one of the specified conditions is met.

- d. restored by the Restore Variant and Indicators instruction;
  - e. restored by the Resume Normal Mode instruction if coming out of the external interrupt mode (but not out of internal interrupt mode);
  - f. stored when an external interrupt occurs.
3. The A-address (if any) is placed in AAR during the extraction of this instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i.e., if the condition being tested is present), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are stored in BAR during the execution phase of the instruction.
4. Consider the variant character in its six-bit form  $V_6 V_5 V_4 V_3 V_2 V_1$ . The following chart may be used to determine the variant character to be used in a BCT instruction.

Table 8-10. BCT Instruction Variant Characters

$V_6$	$V_5$	$V_4$	$V_3$	$V_2$	$V_1$
00 = Test SENSE Switches 1 through 4		SENSE Switch 4	SENSE Switch 3	SENSE Switch 2	SENSE Switch 1
01 = Test SENSE Switches 5 through 8		SENSE Switch 8	SENSE Switch 7	SENSE Switch 6	SENSE Switch 5
1 = Test Zero Balance, Overflow, or Compare	Zero Balance	Overflow	High Compare	Equal Compare	Low Compare

- 5. SENSE switches 5 through 8 are not available with the Model 2040 processor.
- 6. The BCT op code (when testing a SENSE switch) is a "privileged" op code that has special significance when Extended Multiprogramming is in effect.



Format c: The single character specified by the contents of BAR is examined for a condition specified by the variant character of a previous instruction. If the condition is present, the program branches to the location specified by the A-address, and the contents of SR are stored in BAR. If the condition is not present, the program continues in sequence. The valid variant characters and the condition each represents are listed in Table 8-11.

Format d: The single character specified by the contents of BAR is examined for a condition specified by the variant character of a previous instruction. If the condition is present, the program branches to the location specified by the contents of the A-address register (AAR), and the contents of SR are stored in BAR. If the condition is not present, the program continues in sequence. The valid variant characters and the condition each represents are listed in Table 8-11.

## WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

## ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>	
<u>Format a:</u>	JI (A) NXT	A A	NXT B-1	BRANCH NO BRANCH
<u>Format b:</u>	JI (A) NXT	A A	NXT B-1	BRANCH NO BRANCH
<u>Format c:</u>	JI (A) NXT	A A	NXT B <sub>p</sub> -1	BRANCH NO BRANCH
<u>Format d:</u>	JI (A <sub>p</sub> ) NXT	A <sub>p</sub> A <sub>p</sub>	NXT B <sub>p</sub> -1	BRANCH NO BRANCH

Table 8-11. BCC Test Conditions

Variant Character (Octal)	Character Condition
X0	No condition.
X1	The A bit of the character at B is 1.
X2	The B bit of the character at B is 1.
X3	The B and A bits of the character at B are 11.
X4	The B and A bits of the character at B are 00.
X5	The character at B contains a positive sign (the B and A bits are 01).
X6	The character at B contains a negative sign (the B and A bits are 10).
X7	The B and A bits of the character at B are 11 (same as X3 above).
0X	No condition.
1X	The word-mark bit of the character at B is 1 (either a word mark or a record mark is present).
2X	The item-mark bit of the character at B is 1 (either an item mark or a record mark is present).
3X	The character at B contains a record mark.
4X	The character at B contains no punctuation mark.
5X	The character at B contains a word mark only, not an item mark.
6X	The character at B contains an item mark only, not a word mark.
7X	This is a special case; see note 2.

- NOTES: 1. An X represents any octal digit. If both octal digits specify "no condition" (i. e., 00), the branch occurs unconditionally. If only one digit is 0, the branch occurs if the condition specified by the other digit is met. If both octal digits specify conditions, the branch occurs if both conditions are met. The variant character 7X is an exception to these rules, as described in note 2.
2. The 7X variant is interpreted as follows:
- a. If X is 0, the branch is an unconditional branch.
  - b. If X is any digit other than 0, the branch occurs if either the condition specified by the rightmost digit is met or the character at B contains a word mark.



and the contents of SR are stored in BAR. If the bit configurations are unequal, the program continues in sequence.

Format c: The single character specified by the contents of BAR is compared to the variant character specified in a previous instruction. If the bit configurations of the two characters are equal, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the bit configurations are unequal, the program continues in sequence.

Format d: The single character specified by the contents of BAR is compared to the variant character specified in a previous instruction. If the bit configurations of the two characters are equal, the program branches to the location specified by the contents of the A-address register (AAR), and the contents of SR are stored in BAR. If the bit configurations are unequal, the program continues in sequence.

## WORD MARKS

Formats a, b, c, and d:

A word mark in the location tested has no effect on the instruction.

## ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
<u>Format a:</u>	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
<u>Format b:</u>	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
<u>Format c:</u>	JI (A)	A	NXT	BRANCH
	NXT	A	B <sub>p</sub> -1	NO BRANCH
<u>Format d:</u>	JI (A <sub>p</sub> )	A <sub>p</sub>	NXT	BRANCH
	NXT	A <sub>p</sub>	B <sub>p</sub> -1	NO BRANCH

## NOTES

1. The A-address (if any) is placed in AAR during the extraction of the BCE instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i. e., if the condition being tested is present), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are placed in BAR during the execution phase.
2. When the central processor is in the "S" mode, execution of the BCE instruction sets the comparison indicators to show whether B>V, B=V, or B<V.

## EXAMPLES

1. Determine if the character stored in the location tagged LABEL+3 is equal to 6. If so, branch to the location tagged P6; otherwise continue the program in sequence.



# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	M	P	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8				14 15	20 21	62 63 80
					BCE	P6, LABEL+3, 6

2. Determine if any character position in the seven-character field tagged PART contains the letter Q. If so, branch to the location tagged RETRO; otherwise continue the program in sequence.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	M	P	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8				14 15	20 21	62 63 80
1					BCE	RETRO, PART, Q
2					BCE	
3					BCE	
4					BCE	
5					BCE	
6					BCE	
7					BCE	

<b>BBE</b>	BRANCH ON BIT EQUAL	56 <sub>8</sub>
------------	---------------------	-----------------

### FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.				
b.				
c.				
d.				

### FUNCTION

**Format a:** The single character specified by the B-address is combined bit-by-bit with the variant character, according to the rules shown below. If the result (the logical product) is not equal to 0, the program branches to the location specified by the A-address, and the contents of the sequence register (SR) are stored in the B-address register (BAR). If the result is equal to 0, the program continues in sequence.

Bit in B Character	Bit in Variant Character	Bit in Result Field
1	1	1
1	0	0
0	1	0
0	0	0

Format b: The single character specified by the B-address is combined bit-by-bit with the variant character specified in a previous instruction, according to the rules shown above. If the result is not equal to 0, the program branches to the location specified by the A-address, and the contents of SR are stored in BAR. If the result is equal to 0, the program continues in sequence.

Format c: The single character specified by the contents of BAR is combined bit-by-bit with the variant character specified in a previous instruction, according to the rules shown above. If the result is not equal to 0, the program branches to the location specified by the A-address, and the contents of SR are stored in BAR. If the result is equal to 0, the program continues in sequence.

Format d: The single character specified by the contents of BAR is combined bit-by-bit with the variant character specified in a previous instruction, according to the rules shown above. If the result is not equal to 0, the program branches to the location specified by the contents of the A-address register (AAR), and the contents of SR are stored in BAR. If the result is equal to 0, the program continues in sequence.

#### WORD MARKS

Formats a, b, c, and d:

Word marks are not tested by this instruction and have no effect on the operation.

#### ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
<u>Format a:</u>	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
<u>Format b:</u>	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
<u>Format c:</u>	JI (A)	A	NXT	BRANCH
	NXT	A	B <sub>p</sub> -1	NO BRANCH
<u>Format d:</u>	JI (A <sub>p</sub> )	A <sub>p</sub>	NXT	BRANCH
	NXT	A <sub>p</sub>	B <sub>p</sub> -1	NO BRANCH

NOTES

1. The logical product formed by this instruction is tested but is not stored. Main memory locations are not disturbed by this operation.
2. The A-address (if present) is placed in AAR during the extraction of the instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i. e., if the logical product does not equal 0), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are placed in BAR during the execution phase.
3. Since this instruction results in a branch. If any bit product is not equal to 0, only one bit at a time should be tested. Other bits can be checked by branching to additional BBE instructions.

EXAMPLE

Branch to the location tagged BIT8 only if the character at the location tagged MAR contains a 1 in both the B- and the eight-bit positions. Otherwise, continue in sequence. This example requires two BBE instructions to test the two bits in question; location BIT8 is reached only if both tests are met.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	W	X	LOCATION	OPERATION CODE	OPERANDS								
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					BBE	BITB, MAR, 40								
					§	§								
					§	§								
				BITB	BBE	BIT8, MAR, 10								
				§	§	§								
				§	§	§								
				BIT8	- - -	- - - - -								

## CONTROL

- 8-48 ● SET WORD MARK
- 8-49 ● SET ITEM MARK
- 8-50 ● CLEAR WORD MARK
- 8-51 ● CLEAR ITEM MARK
- 8-52 ● HALT
- 8-54 ● NO OPERATION
- 8-55 ● MOVE CHARACTERS TO WORD MARK
- 8-56 ● LOAD CHARACTERS TO A-FIELD WORD MARK
- 8-58 ● STORE CONTROL REGISTERS
- 8-60 ● LOAD CONTROL REGISTERS
- 8-62 ● CHANGE ADDRESSING MODE
- 8-66 ● CHANGE SEQUENCING MODE
- 8-67 ● EXTENDED MOVE
- 8-70 ● MOVE AND TRANSLATE
- 8-74 ● MOVE ITEM AND TRANSLATE
- 8-79 ● LOAD INDEX/BARRICADE REGISTER
- 8-83 ● STORE INDEX/BARRICADE REGISTER
- 8-84 ● TABLE LOOKUP
- 8-87 ● MOVE OR SCAN

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

FUNCTION

Format a: A word mark is set at the location specified by each address. The data and item-mark bits at each location are undisturbed.

Format b: A word mark is set at the location specified by the A address. The data and item-mark bits at this location are undisturbed.

Format c: Word marks are set at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and item-mark bits at each location are undisturbed.

WORD MARKS

Formats a, b, and c:

Word marks are set as described above.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-1	B-1
<u>Format b:</u>	NXT	A-1	A-1 ✓
<u>Format c:</u>	NXT	A <sub>p</sub> -1	B <sub>p</sub> -1

NOTE

The extraction of this instruction when coded in format a. automatically terminates when the last character of the B address is loaded into BAR. Therefore, a word mark is not required in the location following the B address.



ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-1	B-1
Format b:	NXT	A-1	A-1
Format c:	NXT	A <sub>p</sub> -1	B <sub>p</sub> -1

NOTE

The extraction of this instruction when coded in format a. automatically terminates when the last character of the B address is loaded into BAR. Therefore, a word mark is not required in the location following the B address.

EXAMPLE

Set item marks in the locations tagged ENT and ENT+80

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8		14 15 20 21		62 63 80
			SI	ENT, ENT + 80

CW	CLEAR WORD MARK	23 <sub>8</sub>
----	-----------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

FUNCTION

- Format a: The locations specified by the A and B addresses are cleared of word marks. The data and item-mark bits at these locations are undisturbed.
- Format b: The word mark at the location specified by the A address is cleared. The data and item-mark bits at this location are undisturbed.
- Format c: Word marks are cleared at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and item-mark bits at these locations are undisturbed.

WORD MARKS

Formats a, b, and c:

Word marks are cleared as defined above.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-1	B-1
<u>Format b:</u>	NXT	A-1	A-1
<u>Format c:</u>	NXT	A <sub>p</sub> -1	B <sub>p</sub> -1

EXAMPLE

Clear the word marks at locations 400 and 435.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
1			CW	400, 435
2				

CI	CLEAR ITEM MARK	21 <sub>8</sub>
----	-----------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		



FUNCTION

Format a: Item marks are cleared from the locations specified in the A and B addresses. The data and word-mark bits at these locations are undisturbed.

Format b: The item mark at the location specified by the A address is cleared. The data and word-mark bits at this location are undisturbed.

Format c: Item marks are cleared at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and word-mark bits at these locations are undisturbed.

WORD MARKS

Formats a, b, and c:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-1	B-1
<u>Format b:</u>	NXT	A-1	A-1
<u>Format c:</u>	NXT	A <sub>p</sub> -1	B <sub>p</sub> -1

EXAMPLE

Clear the item mark in location REC.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
			CI	REC

H	HALT	45 <sub>8</sub>
---	------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT I
a.	████			
b.	████	████		
c.	████	████	████	
d.	████	████	████	████

## FUNCTION

Format a: This instruction causes the machine to stop. Pressing the RUN button causes the program to resume with the next instruction in sequence.

Format b: The contents of the sequence register (SR) are stored in the B-address register (BAR); the A address of the instruction is transferred to SR; then the machine stops. Pressing the RUN button causes the program to resume with the instruction specified in the A address. This format is usually referred to as a "halt and branch" instruction.

Format c: This instruction causes the machine to stop. Pressing the RUN button causes the program to resume with the next instruction in sequence. The address portions can be used to indicate control information such as a halt identification number (see note 2).

Format d: This instruction causes the machine to stop. Pressing the RUN button causes the program to resume with the next instruction in sequence. The address portions and the variant character can be used to indicate control information such as halt identification number (see note 2).

## WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

## ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>
Format a:	NXT	A <sub>p</sub>	B <sub>p</sub>
Format b:	JI (A)	A	NXT
Format c:	NXT	A	B
Format d:	NXT	A	B

## NOTES

1. If a Halt instruction (in any format) is executed during a peripheral operation, transfer continues until it is completed.
2. Formats c. and d. are useful when a program contains a number of halts. By assigning a number or symbol in the A and B addresses to each halt, the programmer can later identify a particular halt by displaying the contents of AAR and/or BAR. Although the contents of the variant register cannot be displayed through the console or control panel, format d. can be used to store a variant character which can subsequently be used by the program.
3. The H op code is a "privileged" op code that has special significance when Storage Protection is in effect.

EXAMPLES

1. Stop the machine and specify that when the RUN button is pressed, the next instruction will be selected from the location tagged RES.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER					MARK	LOCATION	OPERATION CODE	OPERANDS		
1	2	3	4	5				6	7	8
							H		RES	

2. Identify the halt at the end of a job as follows:  
 A address =9  
 B address =9

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER					MARK	LOCATION	OPERATION CODE	OPERANDS		
1	2	3	4	5				6	7	8
							H	9	9	

<b>NO P</b>	NO OPERATION	40 <sub>8</sub>
-------------	--------------	-----------------

FORMAT

OP CODE            A ADDRESS            B ADDRESS

---



FUNCTION

This instruction performs no operation. This op code can be substituted for the op code of any instruction to make that instruction ineffective.

WORD MARKS

Program operation resumes at the next op code identified by a word mark.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A <sub>P</sub>	B <sub>P</sub>

NOTES

1. This instruction is commonly used in program modification to cause the machine to skip over specific instructions.



ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>w</sub>	B-N <sub>w</sub>
<u>Format b:</u>	NXT	A-N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>

NOTE

Item marks initially stored in B-field locations will be cleared if the corresponding A-field characters do not include item marks.

EXAMPLE

Move the following A fields and store them in sequential B fields as shown.

	<u>Description</u>	<u>A field</u>	<u>B field</u>
	Unit Number	150-155	800-805
	Rack Number	160-168	806-814
	Part Number	173-180	815-822
	Pin Number	185-187	823-825

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	T Y P E	M A R K	LOCATION	OPERATION CODE	OPERANDS	
1				MCW	187, 825	
2				MCW	180	
3				MCW	168	
4				MCW	155	
5						

LCA	LOAD CHARACTERS TO A-FIELD WORD MARK	15 <sub>8</sub>
-----	--------------------------------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████	████
b.	████	████	
c.	████		

FUNCTION

Format a: The data and punctuation bits in the A field are transferred to the B field.

Format b: The data and punctuation bits in the A field are transferred to the field specified by the contents of the B-address register (BAR).

Format c: The data and punctuation bits in the field specified by the contents of the A-address register (AAR) are transferred to the field specified by the contents of BAR.

WORD MARKS

Formats a, b, and c:

The A operand must have a defining word mark (or record mark). The operation terminates when this mark is transferred to the B field.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-N <sub>a</sub>	B-N <sub>a</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	B <sub>p</sub> -N <sub>a</sub>
<u>Format c:</u>	NXT	A <sub>p</sub> -N <sub>a</sub>	B <sub>p</sub> -N <sub>a</sub>

NOTES

1. This instruction (in any format) is the only instruction that always moves both a field and its defining punctuation mark.
2. All punctuation (word marks, item marks, and record marks) initially stored in B-field locations will be cleared if the corresponding A-field characters do not include identical punctuation.
3. The B address must never fall within the A field. The A address may fall within the B field, however, if desired.

EXAMPLE

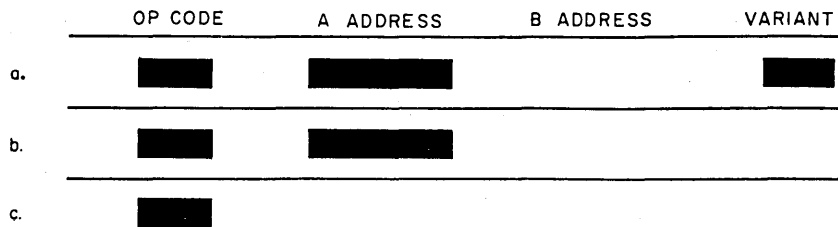
Move both the data bits and the defining word mark of the field tagged TWX to the field tagged RATE.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1			LCA		TWX, RATE	
2						

FORMAT



FUNCTION

Format a: The contents of the control memory register specified by the variant character are stored in the field whose units position is defined by the A address of this instruction. The method of storing these contents depends on the addressing mode being used, as shown in Table 8-12. The valid variant characters and the control register each character represents are listed in Table 8-13.

Table 8-12. Control Register Contents Stored by SCR Instruction

Addressing Mode	Amount of Control Register Stored <sup>a</sup>
Two-Character	Low-order two characters (12 bits).
Three-Character	Low-order 15 bits; the high-order three bits of the field specified by the A-address are cleared to 0's.
Four-Character	The entire contents of the control register plus sufficient high-order 0's to make up 18 bits. On multicharacter processors, the entire bits of the control register plus five high-order 0's. <sup>b</sup>

<sup>a</sup> All bit positions not required to address the largest memory address in a user's system are set to 0's in the A-field.

<sup>b</sup> The five high-order bits of the high-order character are reset to 0's only in the multicharacter processors. In other processors, the entire six bits of the high-order character remain unchanged.

**Format b:** The contents of the control memory register specified by the variant character in a previous instruction are stored in the field whose units position is defined by the A address of this instruction. The number of bits stored depends on the addressing mode being used, as shown in Table 8-12. The valid variant characters and the control register each character represents are listed in Table 8-13.

**Format c:** The contents of the control memory register specified by the variant character in a previous instruction are stored in the field whose units position is defined by the contents of the A-address register (AAR). The number of bits stored depends on the addressing mode being used, as shown in Table 8-12. The valid variant characters and the control register each character represents are listed in Table 8-13.

Table 8-13. Control Registers Stored by SCR Instruction

Variant Character (Octal)	Control Register	Variant Character (Octal)	Control Register	Variant Character (Octal)	Control Register
00	CLC8	20	CLC9	54	ATR
01	CLC1	21	CLC4	64	CSR
02	CLC2	22	CLC5	66	EIR
03	CLC3	23	CLC6	67	AAR
04	CLC8'	24	CLC9'	70	BAR
05	CLC1'	25	CLC4'	76	IIR
06	CLC2'	26	CLC5'	77	SR
07	CLC3'	27	CLC6'		
10	SLC8	30	SLC9		
11	SLC1	31	SLC4		
12	SLC2	32	SLC5		
13	SLC3	33	SLC6		
14	SLC8'	34	SLC9'		
15	SLC1'	35	SLC4'		
16	SLC2'	36	SLC5'		
17	SLC3'	37	SLC6'		

**WORD MARKS**

**Formats a, b, and c:**

A-operand punctuation neither affects nor is affected by this instruction.

**ADDRESS REGISTERS AFTER OPERATION**

**Formats a, b, and c:**

SR	AAR	BAR
NXT	A <sub>p</sub>	B <sub>p</sub>

**NOTES**

1. If AAR is specified by the variant character (octal 67), the previous address in AAR (not the A address retrieved from this instruction) is stored in the location specified by the A address.
2. The control memory register actually designated by the variant character 67<sub>8</sub> is a work register (not AAR). During the extraction of an SCR or LCR



instruction (see below). AAR is used to reference the main memory. Prior to this, the previous contents of AAR are stored in the work register; at the end of the instruction, the contents of the work register are restored in AAR.

3. In a processor equipped with the Scientific Unit (or Scientific Subprocessor), this instruction must not be used to store the contents of the floating-point accumulators.
4. The SCR op code (when storing a read/write counter) is a "privileged" op code that has special significance when Extended Multiprogramming is in effect.

EXAMPLE

Store the contents of BAR in the A-address of the Branch instruction tagged EXIT. (The processor is assumed to be in the three-character addressing mode.)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
				A	B
1					
2		SUB	SCR	EXIT+3,70	
3					
4					
5					
6					
7		EXIT	B	0	

**LCR** LOAD CONTROL REGISTERS 25<sub>8</sub>

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	████████		████
b.	████	████████		
c.	████			

FUNCTION

Format a: The contents of the field whose units position is specified by the A address are loaded into the control register specified by the variant character. The contents of the A field is another main memory address. The method of loading this address into the specified control register depends on the addressing mode being used, as shown in Table 8-14.

Table 8-14. Control Register Contents Loaded by LCR Instruction

Addressing Mode	Amount of Memory Address Loaded
Two-Character	Two-character (12-bit) address is loaded into the low-order two character locations of the register. All other bits in the register (if any) are not disturbed (i. e., the bank bits are protected).
Three-Character	15-bit address is loaded into the low-order 15-bit locations of the register. All other bits in the register (if any) are not disturbed (i. e., the sector bits are protected).
Four-Character	For processors other than the multicharacter processors, an address up to 18 bits long is loaded into the register; the number of bits loaded depends on the size of main memory (see Table 2-2). The multicharacter processors always load 19 bits. <u>Thus, programs written for any other processor, and that are to be compatible with the multicharacter processors must correctly set up the bit to the left of the stored 18-bit address before executing a four-character LCR instruction.</u>

Variant characters and their associated control registers are the same as those specified for the Store Control Registers instruction (see Table 8-13).<sup>1</sup>

**Format b:** The contents of the field whose units position is specified by the A address are loaded into the control register specified by the variant character in a previous instruction. The method of loading the contents of this field (another main memory address) depends on the addressing mode being used, as shown in Table 8-14. Variant characters and their associated control registers are the same as those specified for the Store Control Registers instruction.

**Format c:** The main memory address whose units position is specified by the contents of the A-address register (AAR) is loaded into the control register specified by the variant characters in a previous instruction. The method of loading this address into the specified register depends on the addressing mode being used, as shown in Table 8-14. Variant characters and their associated control registers are the same as those specified for the Store Control Registers instruction.

#### WORD MARKS

##### Formats a, b, and c:

A-operand punctuation neither affects nor is affected by this instruction.

<sup>1</sup> If the variant-dependent privileged LCR capability has been allowed by the execution of an LIB instruction, and the LCR variant character is 64, 67, 70, or 77, and the Storage Protection Indicator is ON in the base mode, and the proceed allow is OFF, the LCR will not cause an Op Code violation and no internal interrupt will be generated.

ADDRESS REGISTERS AFTER OPERATION

Formats a, b, and c:

SR	AAR	BAR	
NXT	(A)	B <sub>p</sub>	VARIANT = 67 <sub>8</sub>
NXT	A <sub>p</sub>	(A)	VARIANT = 70 <sub>8</sub>
(A)	A <sub>p</sub>	B <sub>p</sub>	VARIANT = 77 <sub>8</sub>
NXT	A <sub>p</sub>	B <sub>p</sub>	ALL OTHERS

NOTES

1. If SR is specified by the variant character (77<sub>8</sub>), the next instruction is selected from the location whose address is stored in the field specified by the A-address of the Load Control Registers instruction. In all other cases, the program continues in sequence.
2. The LCR op code is a "privileged" op code that has special significance when Storage Protection is in effect.
3. In a processor equipped with the Scientific Unit (or Subprocessor), this instruction must not be used to load the floating-point accumulators.

EXAMPLE

Load the address stored in the location tagged SUB1 into the change sequence register (CSR).

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y M D	W E R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63	80
			LCR	SUB1,64	

<b>CAM</b>	CHANGE ADDRESSING MODE	42 <sub>8</sub>
------------	------------------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	██████			██████
b.	██████			

FUNCTION

Format a: The Change Addressing Mode instruction is used to specify the following conditions, as designated by the variant character:

1. The addressing mode (two-, three-, or four-character) in which the processor is to interpret the address portions of all subsequent instructions (see note 1).
2. The processing mode (standard mode or "trap" mode) in which all subsequent instructions are to be processed. (See note 3 for a description of the trap mode.)
3. The "S" mode of processing in which several Series 2000 instructions are defined in a special manner (see note 3).

The variant characters and the mode(s) each character represents are listed in Table 8-15.

Format b:

Table 8-15. Modes Specified by Variant Character in CAM Instruction

Variant Character (Octal)	Mode(s)
20 00 or 40 60	Two-character, standard mode Three-character, standard mode Four-character, standard mode
24 04 or 44 64	Two-character, trap mode Three-character, trap mode Four-character, trap mode
30 10 70	Two-character, "S" mode Three-character, "S" mode Four-character, "S" mode
34 14 74	Two-character, trap, "S" mode Three-character, trap, "S" mode Four-character, trap, "S" mode

WORD MARKS

*BIT 2 = Diagnostic Mode 3200*

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

Formats a and b:

SR	AAR	BAR
NXT	A <sub>p</sub>	B <sub>p</sub>

NOTES

1. The CAM instruction is used in conjunction with the ADMODE assembly control statement to specify addressing mode. The ADMODE statement directs the Assembly Program to assemble the address portions of all subsequent source program instructions as two-, three-, or four-character addresses. The CAM instruction directs the processor to interpret the address portions of all subsequent object program instructions as

two-, three-, or four-character addresses. Thus, an address assembled in the three-character addressing mode (via an ADMODE statement) must be processed during a program run as a three-character address for proper execution; the processor is placed in the three-character addressing mode during object program execution by the CAM instruction.

2. The ability to change addressing modes within a program makes it possible to save both time and memory space and provides greater programming flexibility. Extraction and execution time is saved when a smaller addressing mode is used, due to the elimination of the extra memory cycles necessary for a larger address (in characters). Memory space may be conserved by storing frequently used subroutines in the two-character addressing mode (see example).

The larger addresses are necessary to address larger continuous portions of memory. For instance, a two-character address can specify only memory locations within a 4,096-character bank of main memory. A three-character address can refer to any location in a 32,768-character sector. A four-character address can directly address any location in the entire memory (from location  $0_{10}$  to location  $524,287_{10}$ ).

3. When the processor is in the trap mode of instruction execution, any instruction whose op code contains an item mark (or record mark) is both extracted and executed as if it were a Change Sequencing Mode instruction, regardless of the op code that is actually present. The A-address, B-address, and variant character (if any) of the instruction are delivered to AAR, BAR, and the variant register, respectively. The "trapped" op code is not executed; a Change Sequencing Mode instruction (CSM) is executed in its place. The CSM instruction causes a branch to the location stored in the change sequence register (CSR); this location is the beginning of a routine to interpret and execute the instruction whose op code was trapped.

The trap mode is used effectively by the Liberator conversion programs to replace the seldom used instructions of competitive systems when converting the programs of these systems to Series 2000 language. Such instructions are replaced by routines when the trapped op codes are executed as CSM op codes.

4. In addition to specifying the standard or trap modes, the CAM instruction is used to specify the "S" mode of processing. When the processor is in the "S" mode the A, S, ZA, ZS, and BCE instructions are implemented in a special manner. The particular differences that result from the "S" mode of processing are indicated in the notes for each instruction.



FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████			
b.	████	████████		
c.	████	████████	████████	
d.	████	████████	████████	████

FUNCTION

Format a: The contents of the sequence register (SR) and the change sequence register (CSR) are interchanged, and the program branches to the address which was previously stored in CSR.

Format b: The contents of SR and CSR are interchanged, and the program branches to the address which was previously stored in CSR. The A address is loaded into the A-address register (AAR).

Format c: The contents of SR and CSR are interchanged, and the program branches to the address which was previously stored in CSR. The A and B addresses are loaded into AAR and BAR, respectively.

Format d: The contents of SR and CSR are interchanged, and the program branches to the address which was previously stored in CSR. The A and B addresses and the variant character are loaded into AAR, BAR, and the variant register, respectively.

WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	CSR	AAR	BAR
<u>Format a:</u>	JI (contents of CSR)	NXT	A <sub>P</sub>	B <sub>P</sub>
<u>Format b:</u>	JI (contents of CSR)	NXT	A	B <sub>P</sub>
<u>Format c:</u>	JI (contents of CSR)	NXT	A	B
<u>Format d:</u>	JI (contents of CSR)	NXT	A	B

NOTES

1. The Load Control Registers instruction can be used to set up the contents of CSR.
2. When the "trap" mode of instruction execution is specified by the Change Addressing Mode instruction, any subsequent instruction whose op code contains an item mark or a record mark is retrieved and executed as if it were a Change Sequencing Mode instruction. An instruction which is "trapped" in this manner must conform to one of the valid formats of the CSM instruction.

EXAMPLE

Store the absolute address tagged CHANGE in CSR via a Load Control Registers instruction. Later, alter the program sequence by branching to the instruction tagged CHANGE. Provide for the ultimate return to normal programming sequence by storing the contents of SR in CSR.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS	
1			LCR	CHANGE, 64	
2					
3					
4					
5					
6					
7					
8					
9			CSM		
10					
11					
12					
13					
14					
15					

EXM	EXTENDED MOVE	10 <sub>8</sub>
-----	---------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	████████	████████	████
b.	████	████████	████████	
c.	████	████████		
d.	████			



## FUNCTION

Format a: The contents of the A field are moved to the B field in the manner specified by the variant character (see Table 8-16). The programmer specifies how the move operation is to be performed by selecting the desired conditions from the table and encoding the resulting two octal digits as the variant character of the instruction.

Format b: The contents of the A field are moved to the B field in the manner specified by the variant character of a previous instruction (see Table 8-16).

Format c: The contents of the A field are moved to the field specified by the contents of the B-address register (BAR) in the manner specified by the variant character of a previous instruction (see Table 8-16).

Format d: The contents of the field specified by the contents of the A-address register (AAR) are moved to the field specified by the contents of BAR in the manner specified by the variant character of a previous instruction (see Table 8-16).

Table 8-16. Extended Move Conditions

Variant Character (Octal)	Condition
X1	Move A-field <u>data bits</u> to corresponding bit positions in B field.
X2	Move A-field <u>word-mark bits</u> to corresponding bit positions in B field.
X3	Move A-field <u>data and word-mark bits</u> to corresponding bit positions in B field.
X4	Move A-field <u>item-mark bits</u> to corresponding bit positions in B field.
X5	Move A-field <u>data and item-mark bits</u> to corresponding bit positions in B field.
X6	Move A-field <u>word-mark and item-mark bits</u> to corresponding bit positions in B field.
X7	Move A-field <u>data, word-mark and item-mark bits</u> to corresponding bit positions in B field.
0X	Move <u>one character</u> from A to B. The A- and B-address registers are <u>decremented</u> by one.
1X	Move <u>one character</u> from A to B. The A- and B-address registers are <u>incremented</u> by one.
2X	Move characters from <u>right to left</u> (A and B addresses specify rightmost characters in operand fields). Terminate the operation when the first A-field <u>word mark</u> is sensed.
3X	Move characters from <u>left to right</u> (A and B addresses specify leftmost characters in operand fields). Terminate the operation when the first A-field <u>word mark</u> is sensed.

Table 8-16 (cont). Extended Move Conditions

Variant Character (Octal)	Condition
4X	Move characters from <u>right to left</u> . Terminate the operation when the <u>first A-field item mark</u> is sensed.
5X	Move characters from <u>left to right</u> . Terminate the operation when the <u>first A-field item mark</u> is sensed.
6X	Move characters from <u>right to left</u> . Terminate the operation when the <u>first A-field record mark</u> is sensed.
7X	Move characters from <u>left to right</u> . Terminate the operation when the <u>first A-field record mark</u> is sensed.

PUNCTUATION MARKS

Formats a, b, c, and d:

The A field must have a defining punctuation mark, except when the variant character specifies a one-character transfer.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
<u>Format a:</u>	NXT	A-N <sub>a</sub>	B-N <sub>a</sub>	VARIANT = (0, 2, 4, or 6)X
	NXT	A+N <sub>a</sub>	B+N <sub>a</sub>	VARIANT = (1, 3, 5, or 7)X
<u>Format b:</u>	NXT	A-N <sub>a</sub>	B-N <sub>a</sub>	VARIANT = (0, 2, 4, or 6)X
	NXT	A+N <sub>a</sub>	B+N <sub>a</sub>	VARIANT = (1, 3, 5, or 7)X
<u>Format c:</u>	NXT	A-N <sub>a</sub>	B <sub>p</sub> -N <sub>a</sub>	VARIANT = (0, 2, 4, or 6)X
	NXT	A+N <sub>a</sub>	B <sub>p</sub> +N <sub>a</sub>	VARIANT = (1, 3, 5, or 7)X
<u>Format d:</u>	NXT	A <sub>p</sub> -N <sub>a</sub>	B <sub>p</sub> -N <sub>a</sub>	VARIANT = (0, 2, 4, or 6)X
	NXT	A <sub>p</sub> +N <sub>a</sub>	B <sub>p</sub> +N <sub>a</sub>	VARIANT = (1, 3, 5, or 7)X

NOTES

- Here is an example of a typical variant bit configuration: V = 110011. This configuration, encoded in octal notation as 63, specifies that A-field data and word-mark bits are to be moved to the B-field from right to left until the first record mark is sensed in the A-field.
- Consider the variant character in its six-bit form, V<sub>6</sub>V<sub>5</sub>V<sub>4</sub>V<sub>3</sub>V<sub>2</sub>V<sub>1</sub>. If V<sub>1</sub> = 0, A-operand data bits are not transferred and data bits in the B-field remain unchanged.
- If V<sub>2</sub> = 0, A-operand word-mark bits are not transferred and B-operand word-mark bits remain unchanged.

4. If  $V_3 = 0$ , A-operand item-mark bits are not transferred and B-operand item-mark bits remain unchanged.
5. The character containing the terminating punctuation is moved in the same manner as the rest of the field.

**EXAMPLES**

1. Move the data bits of the single character in the location 26 beyond that tagged TEMP to the location tagged WORK, and decrement the A- and B-address registers.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	ITEM MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8		14 15 20 21		62 63 80
			EXM	TEMP+26, WORK, 01

2. Move only the data bits in the field tagged RESV to the field tagged WORK. Move the data from right to left, and terminate the operation when the first item mark in the A field is sensed.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	ITEM MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8		14 15 20 21		62 63 80
			EXM	RESV, WORK, 41

<b>MAT</b>	MOVE AND TRANSLATE	60 8
------------	--------------------	---------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT 1	VARIANT 2
a.					
	OP CODE	A ADDRESS	B ADDRESS	C ADDRESS	
b.					

FUNCTION

Format a: The MAT instruction translates characters from one six-bit configuration to another by means of a stored "translation table." The instruction can be used to translate any number of consecutive characters in the memory.

The A address specifies the location of the rightmost character in the field to be translated. The B address specifies the location into which the translated equivalent of the rightmost A-field character will be moved.

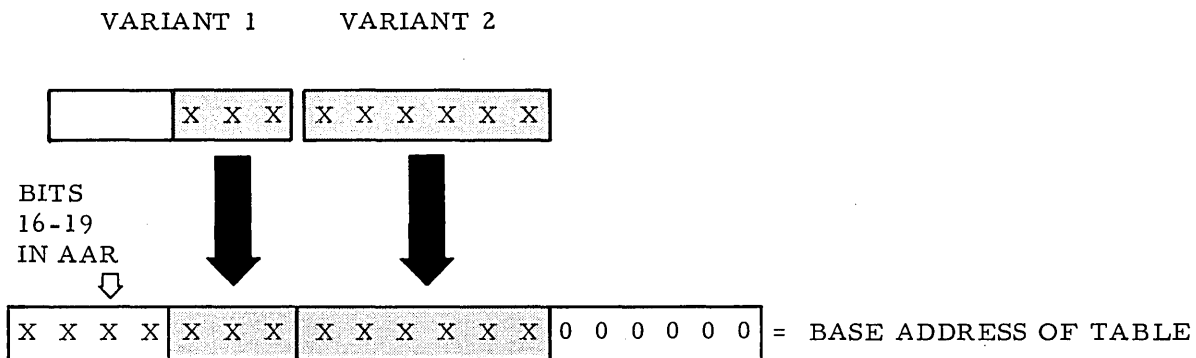
The operation normally terminates when an A-field word mark is sensed. The operation is also terminated if a character is transferred from a word-marked location within the translation table.

The address within the translation table which contains the translated equivalent of an A-field character is formed by combining the A-field character with the two

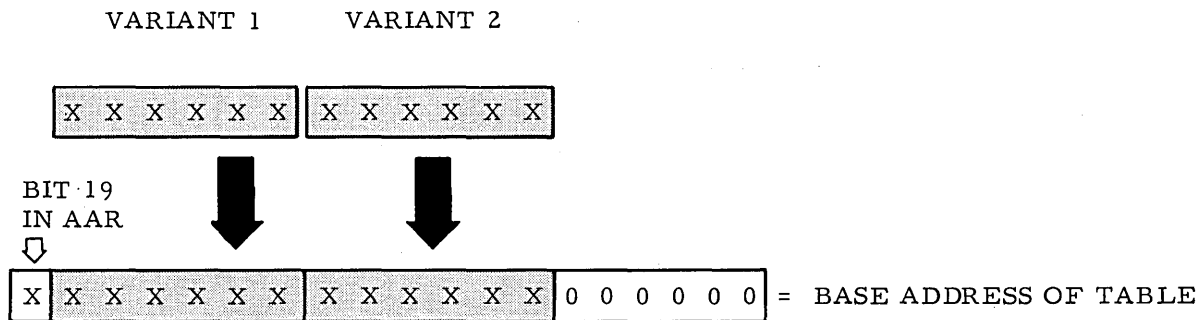
octal variant characters. The method of combining these three characters depends on the addressing mode being used, as described below.

The leftmost, or base, address of the translation table is formed by combining variants 1, 2, and a zero character as shown below. If the processor is in the two- or three-character addressing mode, the leftmost three bits of variant 1 are ignored and the corresponding bit positions (i.e., the sector bits) in the base address (bits 16, 17, 18 and 19) are taken from the contents of the A-address register (AAR). If the processor is in the four-character addressing mode (see below), the entire six-bit contents of variant 1 form bits 13-18 of the base address, while the leftmost (nineteenth) bit, if present, is taken from the contents of AAR.

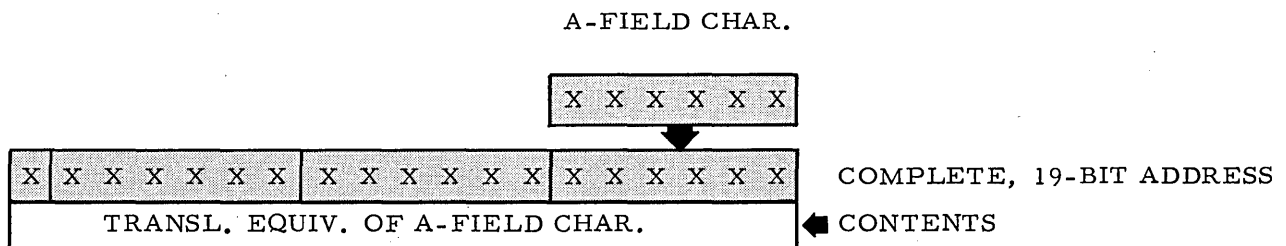
Two- or Three-Character Addressing Mode



Four-Character Addressing Mode



A character in the A field is translated when it is appended to the variant characters (in place of the zero character) to form a complete, 19-bit address. This complete address contains the translated equivalent of the appended A-field character (see below).



Note that because of the positions of variant 1 and variant 2 in the complete address, the base address of the table will always be a multiple of 64. This is compatible with translation requirements since each A-field character can have any of 64 bit configurations (see note 5).

It is a simple task to store the desired equivalent values in a translation table. For instance, assume that a character set which is to be translated into Honeywell code represents the letter A by the bit configuration 110001. Since this bit configuration represents a binary value of 49, the desired Honeywell equivalent (i.e., 010001) should be stored 49 locations beyond the base address of the translation table.

Format b: This is an alternate and simpler format for coding the MAT instruction. In this format, a "C address" replaces the variant characters used in format a. to define the base address of the table. Thus, format b. relieves the programmer of dealing with modulo-64 addresses and converting to octal notation each time a MAT instruction is coded.

The C address is a symbolic tag that is contained in the location field of another source-program entry (e.g., a RESV statement). Once the absolute base address of the table is defined as described for format a., the C address is equated to that address and used in its stead whenever a MAT instruction using the same table is coded again in the program.

Example 2 shows how a C address can be used to define the base address of the translation table.

### WORD MARKS

#### Formats a and b:

The A field must have a defining word mark. It is this word mark that normally stops the operation. The operation will also be terminated if a character is transferred from a word-marked location within the translation table.

### ADDRESS REGISTERS AFTER OPERATION

#### Formats a and b:

SR	AAR	BAR
NXT	A-N <sub>ct</sub>	B-N <sub>ct</sub>

### NOTES

1. This instruction cannot be chained.
2. The contents of the variant register following a move and translate operation are unspecified. Therefore, an instruction requiring a variant character must not be chained after an MAT instruction.
3. Item-mark bits as well as data bits are transferred from the translation table to the B field.
4. Word marks initially stored in the B field remain unchanged. They do not affect the execution of this instruction.
5. The base address of the translation table must always be a multiple of 64. The Easycoder Assembly Program automatically stores the table in this manner when directed by a MORG assembly control statement containing an operand of 64.

EXAMPLES

- Figure 8-6 shows how A-field data is moved to the B field via a translation table.

Translate the contents of the field tagged EXCODE using the stored translation table whose base address is  $256_{10}$  ( $=400_8$ ). Store the translated equivalent in the field tagged EQUIV.

A Address: EXCODE (absolute value = location 630)

B Address: EQUIV (absolute value = location 900)

Variant 1: 00  
Variant 2: 04 } = base address of table (location 256)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER		Y	M	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7
				MAT	EXCODE, EQUIV, 00, 04	

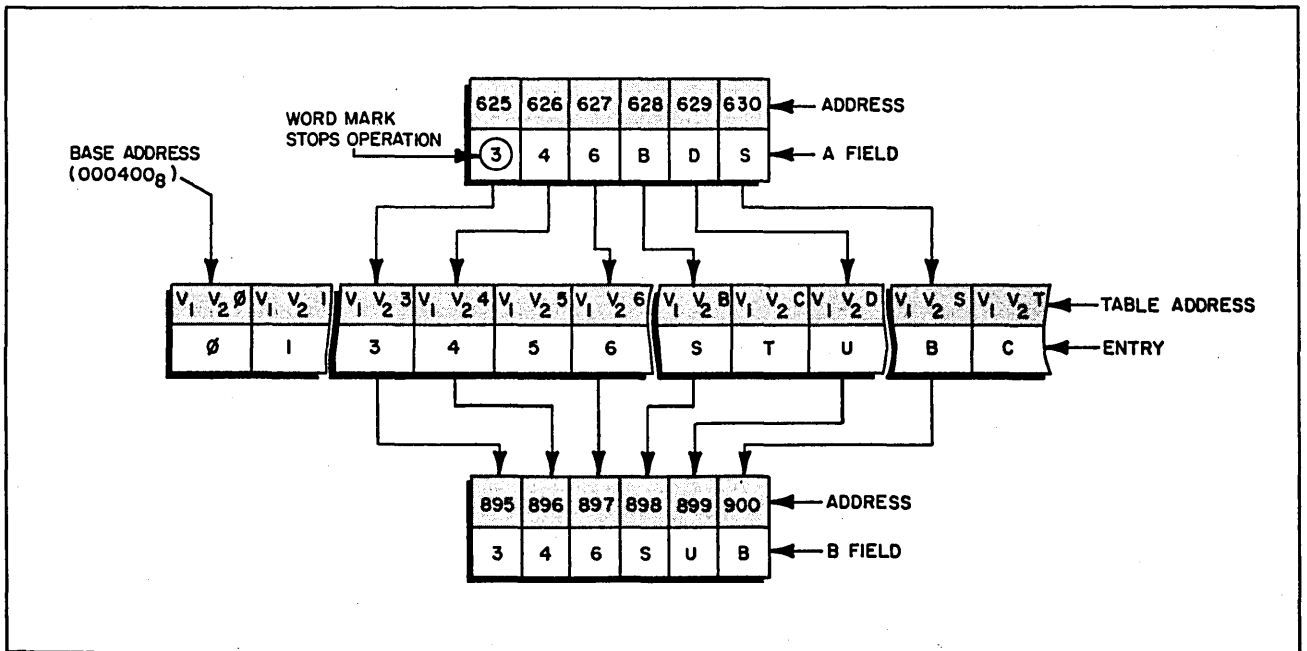


Figure 8-6. MAT Operation

- The following coding shows how the preceding MAT instruction can be coded using a C address. The translation table is set up with a base address of  $256_{10}$  by means of an ORG statement and two DC statements. The ORG statement directs the Assembly Program to load subsequent coding into memory locations beginning at location  $256_{10}$ . The first DC statement defines an alphanumeric constant 40 characters long (i.e., the maximum

size of an alphanumeric constant). These characters are the first 40 characters of a 64-character translation table. The second DC statement defines the remaining 24 characters of the table.

When the MAT instruction is executed, the absolute address equated to the tag MATAB1 is used as the table's base address as in example 1.

## EASYCODER

CODING FORM

PROBLEM _____		PROGRAMMER _____		DATE _____		PAGE ____ OF ____	
CARD NUMBER	VARIANTS	LOCATION	OPERATION CODE	OPERANDS			
				1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16	17 18 19 20 21 22 23 24	25 26 27 28 29 30 31 32
1			ORG	256			
2		MATAB1	DCW	*Δ 1234567890!@%Δ\$ /STUVWXYZA, #23-JKLMNOP*			
3			DC	@QR\$*□Δ\$ ABCDEFGHIA. %!Δ@			
4			}				
5							
6							
7							
8			MAT	EXCODE, EQUIV, MATAB1			

<b>MIT</b>	MOVE ITEM AND TRANSLATE	62 <sub>8</sub>
------------	-------------------------	-----------------

### FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT 1	VARIANT 2	VARIANT 3
a.						
	OP CODE	A ADDRESS	B ADDRESS	C ADDRESS	VARIANT	
b.						

### FUNCTION

**Format a:** The Move Item and Translate instruction is used to translate any information unit (up to 12-bit code) to another information unit of up to 12 bits (e.g., to Series 200 six-bit character code) by the use of a stored translation table. Any number of consecutive information units stored in the memory can be translated.

The A address is the leftmost address of the item to be translated. The B address is the leftmost address of the item into which the translated equivalent of the A item will be moved. The MIT instruction translates the data contents in the A item and moves the translated results, left to right, to the B item.

The operation normally terminates when an item mark is sensed in the A item. The operation will also be terminated if a word-marked character is encountered in the translation table.

An information unit up to six bits in length is stored in one six-bit character location in the memory. Any information unit greater than six bits (7 through 12 bits) is stored in two successive six-bit character locations. Thus, an information unit consisting of up to six bits is considered as a six-bit character, and a unit of from 7 to 12 bits is considered as a 12-bit character.

The sizes of the information units involved in the operation are specified by variant 3, as shown in Table 8-17.

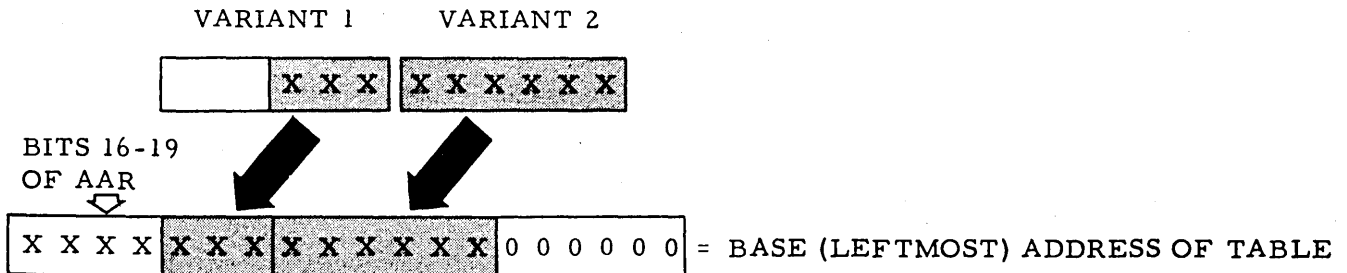
Table 8-17. Size of Information Units in MIT Operation

Variant 3 (Octal)	Operation
00	Translate each <u>six-bit</u> character in the A item. Move the translated equivalent to a <u>six-bit</u> character location in the B item.
01	Translate each <u>12-bit</u> character in the A item. Move the translated equivalent to a six-bit character location in the B item.
02	Translate each <u>six-bit</u> character in the A item. Move the translated equivalent to two character locations (12 bits) in the B item.
03	Translate each <u>12-bit</u> character in the A item. Move the translated equivalent to two character location ( <u>12 bits</u> ) in the B item.

The desired equivalent of an A-item information unit is taken from the stored translation table and moved to the B item. Thus, if the desired equivalent is a six-bit character, each table entry occupies one six-bit character location in the table. If the desired equivalent is a 12-bit character, each table entry occupies two consecutive six-bit character locations in the table. Consequently, variant 3 implicitly specifies the size of each table entry when it explicitly specifies the size of the B-item information unit.

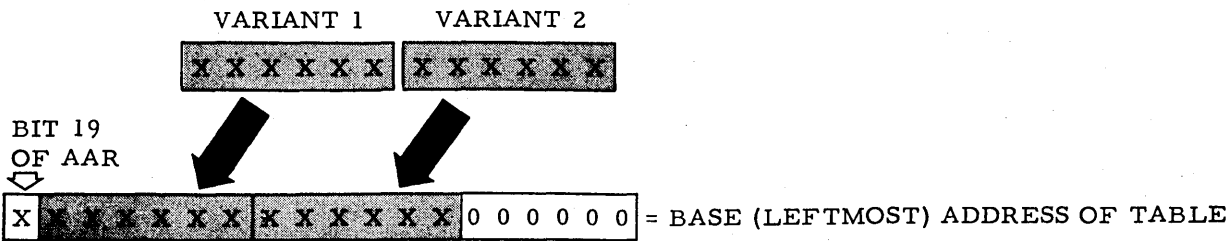
The leftmost, or base, address of the translation table is formed by combining octal variants 1, 2, and a zero character as shown below. If the processor is in the two- or three-character addressing mode, the leftmost three bits of variant 1 are ignored and the corresponding bit positions (i. e., the sector bits) in the base address of the table are taken from the contents of the A-address register (AAR). If the processor is in the four-character addressing mode, the entire six-bit contents of variant 1 form bits 13-18 of the base address, and the nineteenth bit, if present, is taken from the contents of AAR.

Two- or Three-Character Addressing Mode



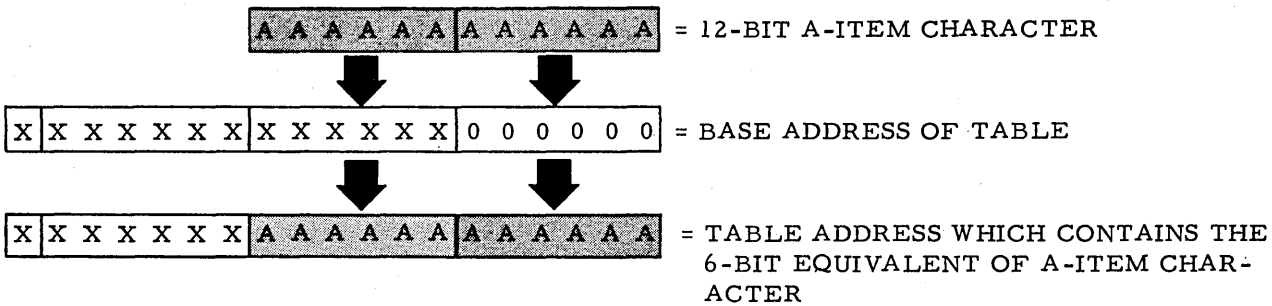


Four-Character Addressing Mode



The address within the translation table which contains the translated equivalent of an A-item character (6- or 12-bit) is formed by superimposing the A-item character over the base address of the table. The method of superposition depends on the size of each table entry (whether 6 or 12 bits), as described below.<sup>1</sup>

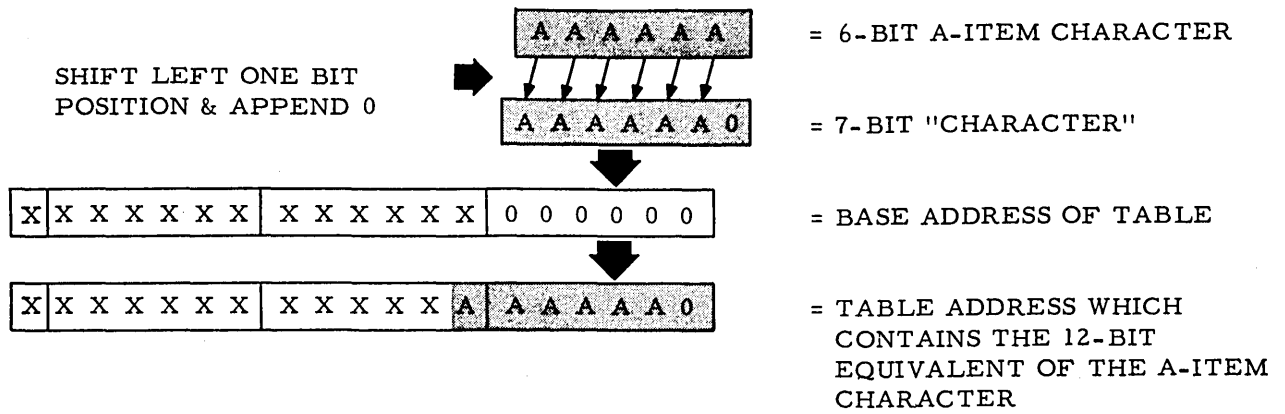
If each table entry is a six-bit character (variant 3 = 00 or 01), the 6- or 12-bit A-item character is superimposed over the rightmost bit positions of the base address. The illustration below shows a 12-bit A-item character being superimposed over the base address, where A = an A-item bit and X = a base address bit.



If each table entry is a 12-bit character (variant 3 = 02 or 03), the 6- or 12-bit A-item character is first shifted one bit position to the left, forming a 7- or 13-bit "character." The rightmost bit position of the character is set to 0. The "character" is then superimposed over the base address to form the table address of the translated equivalent of the A-item character. The shift operation is used to double the referenced table address, since each table entry is stored in two, rather than one, six-bit character locations. The resultant address is the address of the leftmost of the two successive six-bit character locations in the table.

The illustration below shows how a 6-bit A-item character is shifted one bit position to the left and then superimposed over the translation table's base address to form the table address of its equivalent; A = an A-item bit, and X = a base address bit.

<sup>1</sup> Superposition is performed by placing a 1 bit in every position of the table address in which a 1 existed in either the A-item character or the base address or both. This is the "logical inclusive OR" function.



Format b: This is an alternate format for coding the MIT instruction. As in the MAT instruction, a symbolic tag replaces the variant characters used to define the base address of the table in format a. The tag is contained in the location field of another source-program entry which equates the tag to the base address of the table.

The second example of coding an MAT instruction shows the method by which a translation table is stored in memory so that the leftmost location of the table can be used as a symbolic address. This is identical to the method used for format b. of the MIT instruction.

### PUNCTUATION MARKS

Formats a and b:

The A item must contain an item mark. It is this punctuation mark that normally stops the operation. If the A-item information units are 12-bit characters, the terminating item mark may appear in either of the two six-bit character locations.

The operation will also be terminated if a character (6- or 12-bit) is encountered in a word-marked location in the translation table. In this case, neither the word-marked character nor any subsequent characters are moved to the B item; instead, a sequence change is performed (see note 5).

### ADDRESS REGISTERS AFTER OPERATION

Formats a and b:

SR	CSR	AAR	BAR	
NXT	CSR <sub>p</sub>	$A+(N_{A_u})(N_{ut})$	$B+(N_{B_u})(N_{ut})$	} ITEM MARK IN A ITEM STOPS OPERATION
J1 (contents of CSR)	NXT	$A+(N_{A_u})(N_{ut})$	$B+(N_{B_u})(N_{ut})-1$	
				} WORD MARK IN TABLE STOPS OPERATION

### NOTES

1. This instruction cannot be chained.
2. The last six-bit character referenced in the translation table (whether word-marked or not) is left in the variant register following the move item and translate operation.

3. Item-mark bits as well as data bits are transferred from the translation table to the B item.
4. Word marks initially stored in the B item remain unchanged. They do not affect the execution of this instruction.
5. A data control character (e.g., a case-shift character in a teletype code), rather than a translated equivalent to be transferred to the B item, can be stored in a word-marked location in the table. When this word-marked location is sensed, the character in that location is not moved; rather, the contents of SR and CSR are interchanged, providing entry to the routine whose beginning address was previously stored in CSR. Since the word-marked character is stored in the variant register (see note 2), that character can be stored by a Store Variant and Indicators instruction and subsequently tested for identification in the routine.
6. The base address of the translation table must be a multiple of at least 64, due to the positions of variants 1 and 2 in the total 19-bit address. This requirement is sufficient only for the translation of six-bit to six-bit codes. If other than six-bit codes are involved in the translation, the base address of the table must be a multiple of X (where X is the product of the number of codes defined by active bits in the A field entries times the number of characters in each table entry). In other words, the base address of the table must be a multiple of the table size itself. The MORG assembly control statement can be used to assign memory locations to the translation table, starting with the next available memory location whose address is a multiple of 64, 128, 256, etc., as determined by the size of the table.

EXAMPLE

Figure 8-7 shows how eight-bit code is translated to Series 2000 six-bit character code by means of a stored translation table. Each eight-bit information unit is stored in two consecutive six-bit character locations in the A item tagged EIGHT.

Translate the data contents of the item tagged EIGHT using the translation table whose base address is location  $512_{10}$  ( $1000_8$ ). Store the translated values (six-bit characters) in the item tagged SIX.

A Address: EIGHT (absolute value = location 800)  
B Address: SIX (absolute value = location 650)  
Variant 1: 00  
Variant 2: 10  
Variant 3: 01

} = the address of table (location 512)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	W	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8				14 15 20 21		62 63 80
				MIT		EIGHT, SIX, 00, 10, 01

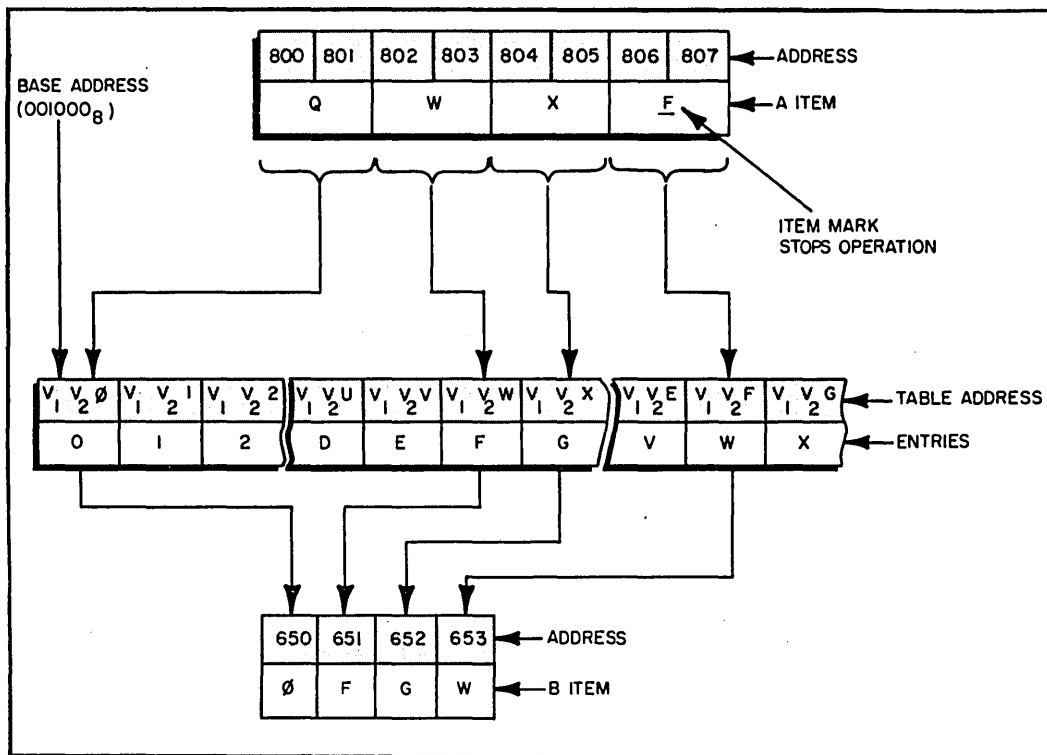


Figure 8-7. MIT Operation

LIB	LOAD INDEX/BARRICADE REGISTER	77 <sub>8</sub>
-----	-------------------------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	████████		
b.	████	████████	████████	
c.	████	████████	████████	████████

FUNCTION

Format a: Basic storage protection is provided by this instruction format; the character(s) at the location(s) specified by the A-address is loaded into the index/barricade register (IBR), specifying the bank address of the lowest main memory bank that is to be protected. The leftmost location of the specified bank is the leftmost location of the protected memory area. (The rightmost location of the protected area is the rightmost location of memory.) For processors other than the multi-character processors, the single-character contents of A are loaded into IBR. In a multicharacter processor, a seventh bit, the rightmost bit of the contents of A-1, is loaded into IBR. The correspondence between the number loaded into IBR and the position of the barricade is shown in Table 8-18. The base relocation register (BRR) is automatically cleared to 0.

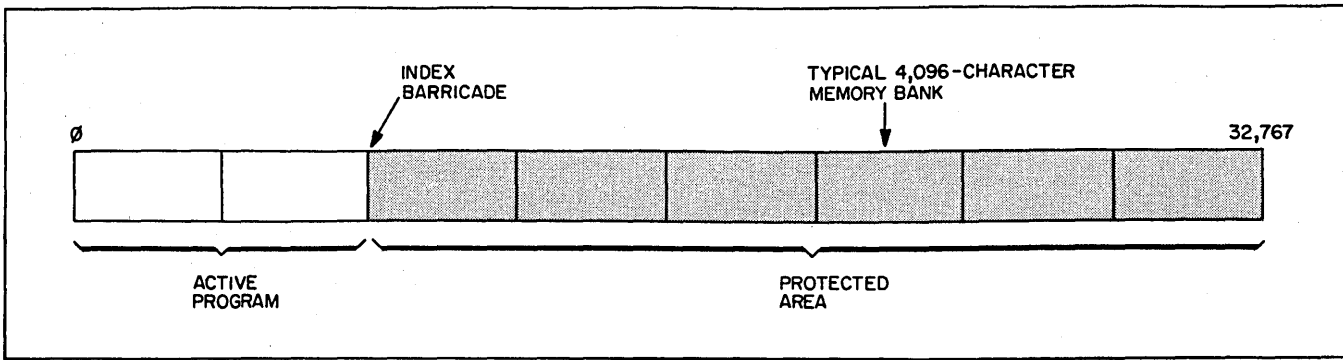


Figure 8-8. Basic Storage Protection

Format b: Storage protection with base relocation is provided by this instruction format; the index/barricade register (IBR) is loaded in the same manner as for basic storage protection (format a. above), but the barricade is relocated relative to the base relocation address. Consequently, when storage protection is in effect, data cannot be delivered to memory locations above the barricade or below the base relocation address unless processing is in the interrupt mode. The character(s) at the location(s) specified by the B address is loaded into the base relocation register (BRR), specifying the bank address of the lowest main memory bank available to a standard (noninterrupt) mode program. The number of main memory locations so designated augments all memory references made in the standard mode. For processors other than the multicharacter processors, the single-character contents of B are loaded into BRR. In a multicharacter processor, a seventh bit, the rightmost bit of the contents of B-1, is loaded into BRR. The character(s) specified by the A address is loaded into the IBR. The barricade is established at the leftmost location in the specified bank (as augmented by the base relocation address); in other words, standard mode programs are prohibited from augmented address equal to or higher than the leftmost location in the specified bank (as augmented). For processors other than the multicharacter processors, the single-character contents of A are loaded into IBR. In a multicharacter processor, a seventh bit, the rightmost bit of the contents of A-1, is loaded into IBR. The correspondence between the number loaded into IBR and the position of the barricade is shown in Table 8-18.

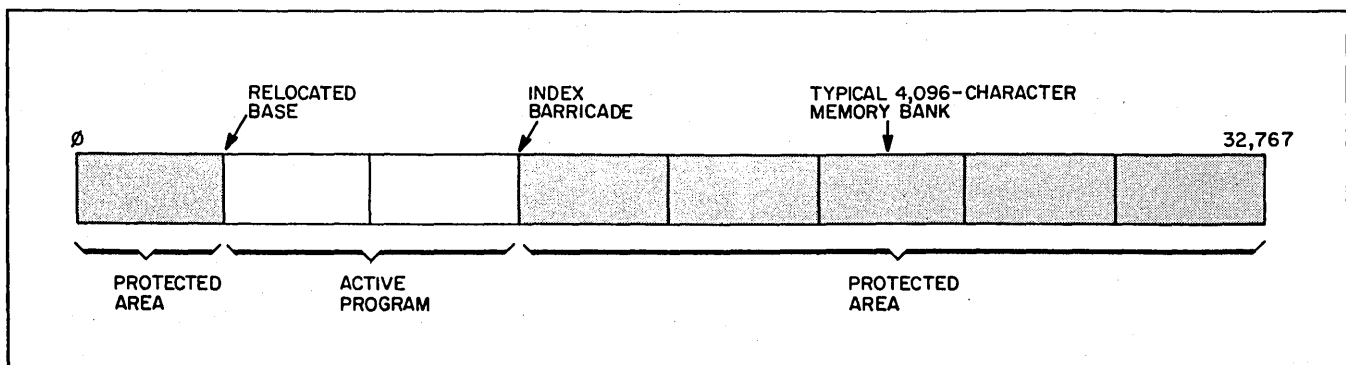


Figure 8-9. Storage Protection with Base Relocation

Format c: As explained in format b (above), storage protection with base relocation is provided by this instruction format. In addition, the high-resolutions clock is activated and the variant-dependent privileged LCR instruction is allowed as specified by the variant character (see Figure 8-10).

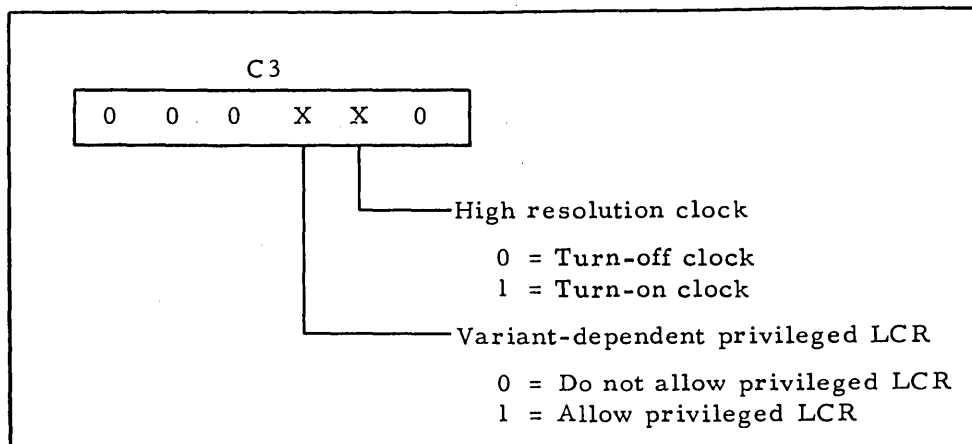


Figure 8-10. LIB Variant Character

Table 8-18. Correspondence Between LIB Setting and Barricade Location

Contents of LIB		Number of Memory Locations to the Left of the Barricade	Contents of LIB		Number of Memory Locations to the Left of the Barricade
Octal	Decimal		Octal	Decimal	
00	0	0	34	28	114,688
01	1	4,096	35	29	118,784
02	2	8,192	36	30	122,880
03	3	12,288	37	31	126,976
04	4	16,384	40	32	131,072
05	5	20,480	41	33	135,168
06	6	24,576	42	34	139,264
07	7	28,672	43	35	143,360
10	8	32,768	44	36	147,456
11	9	36,864	45	37	151,552
12	10	40,960	46	38	155,648
13	11	45,056	47	39	159,744
14	12	49,152	50	40	163,840
15	13	53,248	51	41	167,936
16	14	57,344	52	42	172,032
17	15	61,440	53	43	176,128
20	16	65,536	54	44	180,224
21	17	69,632	55	45	184,320
22	18	73,728	56	46	188,416
23	19	77,824	57	47	192,512
24	20	81,920	60	48	196,608
25	21	86,016	61	49	200,704
26	22	90,112	62	50	204,800
27	23	94,208	63	51	208,896
30	24	98,304	64	52	212,992
31	25	102,400	65	53	217,088
32	26	106,496	66	54	221,184
33	27	110,592	67	55	225,280

Table 8-18 (cont). Correspondence Between LIB Setting and Barricade Location

Contents of LIB		Number of Memory Locations to the Left of the Barricade	Contents of LIB		Number of Memory Locations to the Left of the Barricade
Octal	Decimal		Octal	Decimal	
70	56	229,376	1 34	92	376,832
71	57	233,472	1 35	93	380,928
72	58	237,568	1 36	94	385,024
73	59	241,664	1 37	95	389,120
74	60	245,760	1 40	96	393,216
75	61	249,856	1 41	97	397,312
76	62	253,952	1 42	98	401,408
77	63	258,048	1 43	99	405,504
1 00	64	262,144	1 44	100	409,600
1 01	65	266,240	1 45	101	413,696
1 02	66	270,336	1 46	102	417,792
1 03	67	274,432	1 47	103	421,888
1 04	68	278,528	1 50	104	425,984
1 05	69	282,624	1 51	105	430,080
1 06	70	286,720	1 52	106	434,176
1 07	71	290,816	1 53	107	438,272
1 10	72	294,912	1 54	108	442,368
1 11	73	299,008	1 55	109	446,464
1 12	74	303,104	1 56	110	450,560
1 13	75	307,200	1 57	111	454,656
1 14	76	311,296	1 60	112	458,752
1 15	77	315,392	1 61	113	462,848
1 16	78	319,488	1 62	114	466,944
1 17	79	323,584	1 63	115	471,040
1 20	80	327,680	1 64	116	475,136
1 21	81	331,776	1 65	117	479,232
1 22	82	335,872	1 66	118	483,328
1 23	83	339,968	1 67	119	487,424
1 24	84	344,064	1 70	120	491,520
1 25	85	348,160	1 71	121	495,616
1 26	86	352,256	1 72	122	499,712
1 27	87	356,352	1 73	123	503,808
1 30	88	360,448	1 74	124	507,904
1 31	89	364,544	1 75	125	512,000
1 32	90	368,640	1 76	126	516,096
1 33	91	372,736	1 77	127	520,192

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>
<u>Format a:</u>	NXT	A-2	B <sub>p</sub>
<u>Format b:</u>	NXT	A-2	B-2

NOTES

1. The 15 additional index registers (Y1 through Y15) which are included in the Storage Protect and Extended Multiprogramming features are located in the first 60 character locations to the right of the barricade position specified by this instruction. These locations can be used as normal storage locations when they are not being used for indexing operations.
2. The LIB op code is a "privileged" op code that has special significance when Storage Protection is in effect.
3. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.
4. The LIB instruction is not interpreted by EasyCoder Assembler A, B, or C.

EXAMPLE

Assuming that there are 131,072 storage locations in the main memory, set up the memory in such a way that the "open" memory area consists of locations 0 through 65,535 and the protected memory area consists of locations 65,536 through 131,072. The single octal character "20" is contained in the location tagged MP2.

**EASYCODER**

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	W	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	9	10	11 12 13 14 15	16 17 18 19 20 21	22 23 24 25 26 27 28 29 30
			LIB	MP 2	

<b>SIB</b>	STORE INDEX/BARRICADE REGISTER	76 <sub>8</sub>
------------	--------------------------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	██████████	
b.	████	██████████	██████████

FUNCTION

Format a: Basic storage protection is provided by this instruction format; the contents (up to seven bits) of the index/barricade register (IBR) are stored in the character location(s) specified by the A-address. All high-order bit positions in A which are not used to specify the contents of the index/barricade register are cleared to 0's. In a multicharacter processor only, the seventh bit in IBR is stored in the rightmost bit position of location A-1 and the five remaining bit positions in A-1 are cleared to 0's.

Format b: Storage protection with base relocation is provided by this instruction format; the contents of the index/barricade register (IBR) are stored in the same manner as for basic storage protection (format a, above); in addition, the contents of the base relocation register (BRR) are also stored. The contents (up to seven bits) of BRR are stored in the character location(s) specified by the B-address. All high-order bit positions in B which are not used to specify the contents of the base relocation register are cleared to 0's. In a multicharacter processor only, the seventh bit in BRR is stored in the rightmost bit position of location B-1 and the five remaining



bit positions in B-1 are cleared to 0's. The contents (up to seven bits) of the index/barricade register are stored in the character location(s) specified by the A-address. All high-order bit positions in A which are not used to specify the contents of the index/barricade register are cleared to 0's. In a multicharacter processor only, the seventh bit in IBR is stored in the rightmost bit position of location A-1 and the five remaining bit positions in A-1 are cleared to 0's.

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
<u>Format a:</u>	NXT	A-2	B <sub>p</sub>
<u>Format b:</u>	NXT	A-2	B-2

NOTES

1. The SIB instruction is not interpreted by Easycoder Assembler A, B, or C.
2. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.

EXAMPLE

Store the contents of the index/barricade register in the single character location tagged PROT.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63
		SIB	PROT

TLU	TABLE LOOKUP	57 <sub>8</sub>
-----	--------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	████████	████████	████
b.	████	████████	████████	
c.	████	████████		
d.	████			

## FUNCTION

Format a: A table in memory is a series of fields, each of which normally contains an argument of a function and the corresponding value of the function (see notes 1 and 2). The Table Lookup instruction initiates a search in a stored table for an argument which bears a specified relationship to a search argument, which is stated in the instruction (see illustration below).

The B address specifies the rightmost location of the stored table, the A address specifies the location of the search argument, and the variant character specifies a relationship (equal to, higher than, etc.) between the desired argument in the table and the search argument. The table is searched from right to left until this relationship is found or until a table field is found which is shorter than the search argument. Then comparison indicators are turned on and the search terminates.

Format b: Search the table whose rightmost location is specified by B for an argument which bears to the search argument specified by A a relationship specified by the variant character of a previous instruction. When this relationship is found or when a table field is found which is shorter than the search argument, turn on comparison indicators and terminate the search.

Format c: Search the table whose rightmost location is specified by the contents of the B-address register (BAR) for an argument which bears to the search argument specified by A a relationship specified by the variant character of a previous instruction. When this relationship is found or when a table field is found which is shorter than the search argument, turn on comparison indicators and terminate the search.

Format d: Search the table whose rightmost location is specified by the contents of BAR for an argument which bears to the search argument specified by the contents of the A-address register (AAR) a relationship specified by the variant character of a previous instruction. When this relationship is found or when a table field is found which is shorter than the search argument, turn on comparison indicators and terminate the search.

## WORD MARKS

Formats a, b, c, and d:

The A operand (the search argument) must have a defining word mark. Each table field must also have a defining word mark.

## ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>
<u>Format a:</u>	NXT	A-N <sub>a</sub>	L <sub>ta</sub>
<u>Format b:</u>	NXT	A-N <sub>a</sub>	L <sub>ta</sub>
<u>Format c:</u>	NXT	A-N <sub>a</sub>	L <sub>ta</sub>
<u>Format d:</u>	NXT	A <sub>p</sub> -N <sub>a</sub>	L <sub>ta</sub>

## NOTES

1. Each value in the table is normally stored immediately to the left of the corresponding argument, and each pair (argument plus value) constitutes a field in the table. However, if the values in the table are longer than

three characters, it is advisable to store them in another part of memory and to store their 2- or 3-character addresses in the table instead. Since the timing of the TLU instruction depends on the number of characters searched in the table, it is desirable to minimize the length of the table.

2. The Branch on Condition Test instruction can be used after Table Lookup to branch to a routine which moves the located value to a work area. Note that at the completion of the TLU instruction, the B-address register (BAR) contains the address of the desired value (or the address of a location containing the address of the desired value, in the case where the values are too long for efficient storage in the table).
3. The variant characters which specify the desired relationships between the search argument and the argument to be located in the table are as follows:

Variant Character (Octal)	Relationship Which Terminates Search
01	Stored Argument < Search Argument
02	Stored Argument = Search Argument
03	Stored Argument ≤ Search Argument
04	Stored Argument > Search Argument
05	Stored Argument ≠ Search Argument
06	Stored Argument ≥ Search Argument

4. The length of each argument in the table must be equal to the length of the search argument. Note that a short table field (e.g., one which contains a short argument or which contains no value) can be used to terminate the search, which leaves the comparison indicators set to the condition "Stored Argument > Search Argument."
5. The Table Lookup instruction is not interpreted by EasyCoder Assembler A, B, or C.
6. EasyCoder Assembler D:
  - a. Only the generic op code (TLU) can be used; i.e., specific op codes as in 9. below cannot be used.
  - b. Since format b. requires the use of a specific op code, EasyCoder Assembler D does not permit the use of this format.
7. The Mod 2 assemblers:
  - a. Format a must use the generic op code (TLU) along with an explicit variant character.
  - b. Format b must use a specific op code (e.g., LEH) in order to supply the omitted variant character.
  - c. Formats c and d always use the variant character from the previous contents of the variant register. Therefore, the op code used should agree with the one used previously or be the generic form (TLU).

EXAMPLE

- Figure 8-10 shows how a stored table is searched for an argument which bears a specified relationship to a search argument.

Search the table tagged TABLE1 for the value which corresponds to the argument (557) stored in the field tagged ARGMNT.

A Address: ARGMNT (absolute value = location 609)

B Address: TABLE1 (absolute value = location 149)

Variant 1: 02 = (Stored Argument = Search Argument)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS	
			1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
1	TLU		ARGMNT, TABLE 1, 02	
2				
3				

MOS	MOVE OR SCAN	13 <sub>8</sub>
-----	--------------	-----------------

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	████	██████████	██████████	████
b.	████	██████████	██████████	
c.	████	██████████		
d.	████			

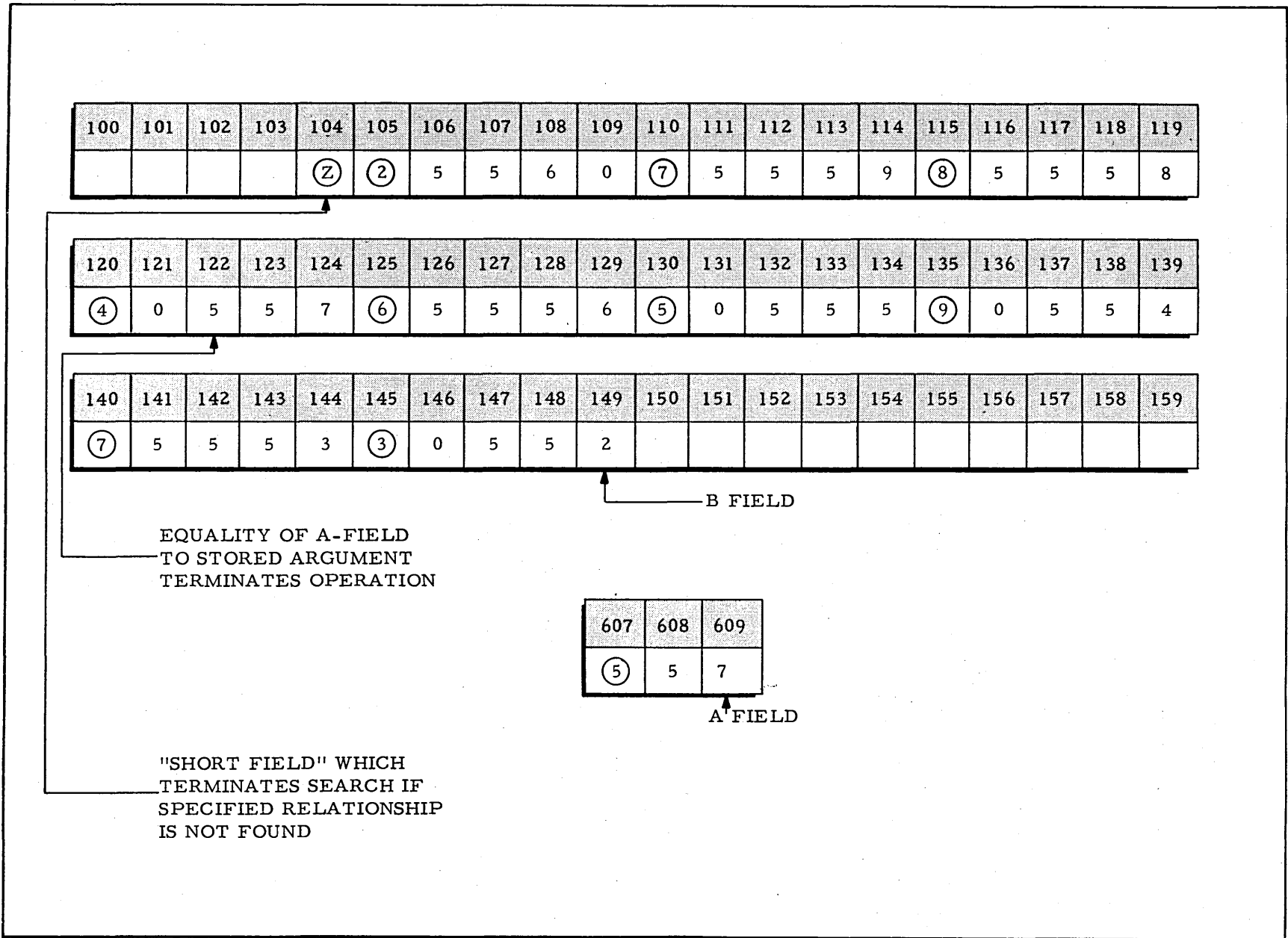


Figure 8-11. TLU Operation

FUNCTION

- Format a: The contents of the A-field are moved to the B-field in the manner specified by the variant character (see Table 8-19). The programmer specifies how the move operation is to be performed by selecting the desired conditions from the table and encoding the resulting two octal digits as the variant character of the instruction.
- Format b: This format is valid in symbolic coding only when a specific op code is used to indicate the omitted variant character. The resultant machine-language format and functions are the same as those described for format a.
- Format c: The contents of the A-field are moved to the field specified by the contents of the B-address register (BAR) in the manner specified by the variant character of a previous instruction (see Table 8-19).
- Format d: The contents of the field specified by the contents of the A-address register (AAR) are moved to the field specified by the contents of BAR in the manner specified by the variant character of a previous instruction (see Table 8-19).

Table 8-19. Move or Scan Conditions

Variant Character (Octal)	Condition
X0	No information is moved. The A- and B-address registers are incremented or decremented in accordance with the high-order digit of the variant character.
X1	Move A-field <u>numeric</u> bits to corresponding bit positions in B field.
X2	Move A-field <u>zone</u> bits to corresponding bit positions in B field.
X3	Move A-field <u>data and item-mark</u> bits to corresponding bit positions in B field.
X4	Move A-field <u>word-mark</u> bits to corresponding bit positions in B field.
X5	Move A-field <u>numeric and word-mark</u> bits to corresponding bit positions in B field.
X6	Move A-field <u>zone and word-mark</u> bits to corresponding bit positions in B field.
X7	Move A-field <u>data, word-mark, and item-mark</u> bits to corresponding bit positions in B field.
0X	Move <u>one character</u> from A to B. The A- and B-address registers are <u>decremented</u> by one.
1X	Move characters from <u>left to right</u> (A and B addresses specify leftmost characters in operand fields). Terminate the operation when the first A- or B-field <u>word mark</u> is sensed.
2X	Move characters from <u>right to left</u> (A and B addresses specify the rightmost characters in operand fields). Terminate the operation when the first A-field <u>word mark</u> is sensed.

Table 8-19 (cont). Move or Scan Conditions

Variant Character (Octal)	Condition
3X	Move characters from <u>left to right</u> . Terminate the operation when the control character "@" (72 <sub>8</sub> ) is sensed in the A field.
4X	Move characters from <u>right to left</u> . Terminate the operation when the first B-field <u>word mark</u> is sensed.
5X	Move characters from <u>left to right</u> . Terminate the operation when the control character ";" (32 <sub>8</sub> ) with a <u>word mark</u> is sensed in the A field.
6X	Move characters from <u>right to left</u> . Terminate the operation when the first A- or B-field <u>word mark</u> is sensed.
7X	Move characters from <u>left to right</u> . Terminate the operation when either the control character ";" (32 <sub>8</sub> ) with a <u>word mark</u> or control character "@" (72 <sub>8</sub> ) is sensed in the A field.

WORD MARKS

Formats a, b, c, and d:

Word marks and control characters affect the operation of the instruction as described in the table above.

ADDRESS REGISTERS AFTER OPERATION

	<u>SR</u>	<u>AAR</u>	<u>BAR</u>	
<u>Format a:</u>	NXT	A-1	B-1	VARIANT = 0X
	NXT	A+N <sub>w</sub>	B+N <sub>w</sub>	VARIANT = 1X
	NXT	A-N <sub>a</sub>	B-N <sub>a</sub>	VARIANT = 2X
	NXT	A+N <sub>a</sub>	B+N <sub>a</sub>	VARIANT = (3, 5, or 7)X
	NXT	A-N <sub>b</sub>	B-N <sub>b</sub>	VARIANT = 4X
	NXT	A-N <sub>w</sub>	B-N <sub>w</sub>	VARIANT = 6X
<hr/>				
<u>Format b:</u>	NXT	A-1	B-1	VARIANT = 0X
	NXT	A+N <sub>w</sub>	B+N <sub>w</sub>	VARIANT = 1X
	NXT	A-N <sub>a</sub>	B-N <sub>a</sub>	VARIANT = 2X
	NXT	A+N <sub>a</sub>	B+N <sub>a</sub>	VARIANT = (3, 5, or 7)X
	NXT	A-N <sub>b</sub>	B-N <sub>b</sub>	VARIANT = 4X
	NXT	A-N <sub>w</sub>	B-N <sub>w</sub>	VARIANT = 6X

<u>Format c:</u>	NXT	A-1	B -1	VARIANT = 0X
	NXT	A+N <sub>w</sub>	B <sub>p</sub> +N <sub>w</sub>	VARIANT = 1X
	NXT	A-N <sub>a</sub>	B <sub>p</sub> -N <sub>a</sub>	VARIANT = 2X
	NXT	A+N <sub>a</sub>	B <sub>p</sub> +N <sub>a</sub>	VARIANT = (3, 5, or 7)X
	NXT	A-N <sub>b</sub>	B <sub>p</sub> -N <sub>b</sub>	VARIANT = 4X
	NXT	A-N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>	VARIANT = 6X

---

<u>Format d:</u>	NXT	A <sub>p</sub> -1	B -1	VARIANT = 0X
	NXT	A <sub>p</sub> +N <sub>w</sub>	B <sub>p</sub> +N <sub>w</sub>	VARIANT = 1X
	NXT	A <sub>p</sub> -N <sub>a</sub>	B <sub>p</sub> -N <sub>a</sub>	VARIANT = 2X
	NXT	A <sub>p</sub> +N <sub>a</sub>	B <sub>p</sub> +N <sub>a</sub>	VARIANT = (3, 5, or 7)X
	NXT	A <sub>p</sub> -N <sub>b</sub>	B <sub>p</sub> -N <sub>b</sub>	VARIANT = 4X
	NXT	A <sub>p</sub> -N <sub>w</sub>	B <sub>p</sub> -N <sub>w</sub>	VARIANT = 6X

### NOTES

1. The character containing the terminating punctuation and/or control characters is moved or scanned in the same manner as the rest of the field.
2. The variant characters and the corresponding mnemonic op codes which they represent are contained in Appendix B.
3. The Move or Scan instruction is not interpreted by the EasyCoder Assembler A, B, C, or D.





## INTERRUPT CONTROL

- 8-94 ● STORE VARIANT AND INDICATORS
- 8-98 ● RESTORE VARIANT AND INDICATORS
- 8-100 ● MONITOR CALL
- 8-101 ● RESUME NORMAL MODE

FORMAT

OP CODE	A ADDRESS	B ADDRESS	VARIANT
████			████

FUNCTION

The SVI instruction is used to store information regarding the current status of the processor when an interrupt condition occurs. The instruction stores the designated information in up to six consecutive locations following its own variant character.

Each bit in the six-bit variant character ( $V_6 V_5 V_4 V_3 V_2 V_1$ ) represents processor control registers or indicators whose contents are to be stored in a single character location. The programmer specifies the amount of information to be stored by selecting the desired entries from Table 8-20 and encoding the resulting bit configuration as two octal digits.

Table 8-20. Information Stored by SVI Instruction

Variant Character	Information Stored
$V_6 V_5 V_4 V_3 V_2 V_1$	
X X X X X 1	The contents of the variant register. The setting of the BCT privileged indicator.
X X X X 1 X	The settings of the arithmetic, comparison, address mode, and item-mark trap mode indicators. This information is stored in seven bit positions of the character location — the six data bit positions and the item-mark bit position.  The arithmetic and comparison indicators are cleared when their contents have been stored.
X X X 1 X X	The contents of the auxiliary indicators register (AIR). The contents of the arithmetic, comparison, address mode, and item mark trap mode indicators are stored automatically in this register upon the occurrence of an external interrupt. Upon executing an RNM instruction to return to either standard or internal interrupt mode, the specified indicators are reset automatically using the contents of this register. The contents of this register can be changed by using the RVI instruction.  The auxiliary arithmetic and comparison indicators are cleared when their contents have been stored.
X X 1 X X X	The settings of the indicators associated with the scientific unit and subprocessor, the sector interrupt masks, and the extended I/O capacity. The scientific indicators are cleared when their contents have been stored.

Table 8-20 (cont). Information Stored by SVI Instruction

Variant Character	Information Stored
$V_6 V_5 V_4 V_3 V_2 V_1$	
<p>X 1 X X X X</p>	<p>The settings of the protect, proceed, floating-point error, instruction timeout allow, S-mode, and relocation indicators and (if the processor is in the <u>external</u> interrupt mode) the setting of the <u>internal</u> interrupt (II) mode indicator.</p> <p>The protect, proceed, floating-point error, and instruction time out allow indicators are cleared when their contents are stored.</p>
<p>1 X X X X X</p>	<p>The settings of the interrupt source indicators and the instruction timeout indicator. The stored settings of the interrupt source indicators can be tested to determine the status of the processor as follows:</p> <ol style="list-style-type: none"> <li>1. Whether the processor is in the <u>external</u> interrupt mode, the <u>internal</u> interrupt mode, or the standard processing mode.</li> <li>2. The source of the interruption if the processor is in the external interrupt mode; any of three sources can be determined — a peripheral control, the console, or the Monitor Call instruction.</li> <li>3. Whether an external interrupt (EI) address violation has occurred (if the processor is in the <u>external</u> interrupt mode).</li> <li>4. Whether an op code violation has occurred (if the processor is in the <u>internal</u> interrupt mode).</li> <li>5. Whether an internal interrupt (II) address violation has occurred (if the processor is in the internal interrupt mode).</li> <li>6. Whether the extraction and execution of an instruction (in the standard mode) has exceeded the maximum time limit (if so, an internal interrupt will have occurred).</li> <li>7. Whether Scientific Unit or Subprocessor is in error.</li> </ol> <p>The indicators referred to in 3 through 6, above, as well as those which identify the control panel (or console) and the Monitor Call instruction source, are cleared when their contents are stored.</p>

## WORD MARKS

A word mark is required in the location following the variant character to terminate the extraction of the SVI instruction. Other word marks (if any) in the locations in which information is stored are ignored and unaffected. Program operation resumes with the next word-marked location following the stored information (the next sequential op code).

## ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A P	B P

Variant Bit	Stored Character Location Bits						
	I/M Bit	B Bit	A Bit	8 Bit	4 Bit	2 Bit	1 Bit
$V_1$	Testing (BCT) of SENSE switches privileged: 1 = yes; 0 = no. <div style="text-align: center; margin-top: 10px;">Contents of variant register</div>						
$V_2$	Trap-mode: 1 = yes; 0 = no.	Address mode: 01 = 2-character; 00 = 3-character; 11 = 4-character.		Overflow: 1 = yes; 0 = no.           *	Zero balance: 1 = yes; 0 = no.           *	$A \leq B$ : 1 = yes; 0 = no.           *	$A = B$ : 1 = yes; 0 = no.           *
$V_3$	Contents of AIR (identical to information stored by $V_2$ , above) <div style="text-align: center; margin-top: 10px;">* * * *</div>						
$V_4$	Extended I/O indicator 1 = on; 0 = off.	MPO: 1 = yes; 0 = no.           *	DVC: 1 = yes; 0 = no.           *	EXO: 1 = yes; 0 = no.           *	Sector 1 interrupt mask: 1 = on; 0 = off.	Sector 2 interrupt mask: 1 = on; 0 = off.	Sector 3 interrupt mask: 1 = on; 0 = off.
$V_5$		Protect indicator: 1 = on; 0 = off.           *	Instruction timeout allow 1 = on; 0 = off.           *	"S" mode	Proceed indicator: 1 = on; 0 = off.           *	Relocation indicator: 1 = on; 0 = off.	In external interrupt mode only: 1 = II indicator on; otherwise, 0.
$V_6$	Processor is in external interrupt mode						
	High-resolution clock interrupt: 1 = yes; 0 = no.           *	EI address violation: 1 = yes; 0 = no.           *	Monitor Call: 1 = yes; 0 = no.           *	Control panel or console interrupt: 1 = yes; 0 = no.           *	Peripheral interrupt: 1 = yes; 0 = no.	1	II Mode indicator: 1 = on; 0 = off.
	Processor is in internal interrupt mode.						
	Floating-point error indicator: 1 = on; 0 = off.           *	II Address violation: 1 = yes; 0 = no.           *	Op code violation: 1 = yes; 0 = no.           *	Instruction timeout indicator: 1 = yes; 0 = no.           *	0	0	1
* = Indicators that are cleared when their contents are stored.							

NOTES

1. Only the number of characters specified by the variant character are stored. They are stored in the order listed in Table 8-20: the contents of the variant register (if specified) are stored in the location immediately following the SVI instruction, etc., using those locations actually required to store the requested information.
2. Item-mark and data bit positions which are not used to store information are cleared to 0's.
3. The format in which information is stored by the SVI instruction is shown in the preceding table. Indicators which are cleared (i. e., set to 0) when their contents are stored are indicated by an asterisk (\*).
4. The current status of the arithmetic, comparison, address mode, and trap mode indicators are not stored in the auxiliary indicators register (AIR) when an internal interrupt occurs. The contents of AIR should therefore not be stored by an SVI instruction in the internal interrupt mode, for the contents of AIR would be meaningless at the time of internal interruption.
5. The SVI op code is a "privileged" op code that has special significance when Storage Protection is in effect.
6. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.
7. The method of coding interrupt service routines is described in Section II "Interrupt Processing."
8. The contents of the variant register are not altered by the execution of this instruction; i. e., the variant character of the SVI instruction is not stored therein.

EXAMPLE

Store the following information in the three successive memory locations which immediately follow the variant character of the instruction:

1. The contents of the variant register;
2. The contents of the auxiliary indicators register (AIR); and
3. The settings of the interrupt source indicators.

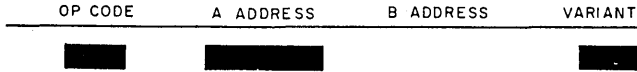
The op code of the SVI instruction is tagged STORE, so that the locations of the stored information are STORE+2, STORE+3, and STORE+4.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	V	A	R	LOCATION	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7	8
				STORE	SVI	45	

FORMAT



FUNCTION

Up to five consecutive characters (previously stored via an SVI instruction) are loaded into the processor control registers and/or indicators specified by the variant character. Characters are retrieved from left to right, beginning with the character specified by the A address.

The low-order five bits of the variant character specify the registers and/or indicators whose contents are to be restored. The programmer specifies the amount of information to be restored by selecting the desired entries from Table 8-21 and encoding the resulting bit configurations as two octal digits.

Table 8-21. Information Restored by RVI Instruction

Variant Character	Information Restored
$V_6 V_5 V_4 V_3 V_2 V_1$	
0 X X X X 1	The contents of the variant register, and the setting of the BCT privileged indicator.
0 X X X 1 X	The settings of the arithmetic, comparison, address mode, and item-mark trap mode indicators. This information is stored in the six data bits and the item-mark bit of a character location.
0 X X 1 X X	The contents of the auxiliary indicators register (AIR). Upon returning from external interrupt mode to either internal interrupt or standard mode, the contents of this register are moved automatically to the indicators specified above for $V_2$ .
0 X 1 X X X	The setting of the indicators associated with the scientific unit or subprocessor and the sector interrupt masks.
0 1 X X X X	The settings of the protect, proceed, instruction time-out allow, S mode, and relocation indicators and (if the processor is in the <u>external</u> interrupt mode) the setting of the internal interrupt (II) mode indicator.

WORD MARKS

Word marks neither affect nor are affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A <sub>p</sub>	B <sub>p</sub>

NOTES

1. Each entry in the right-hand column of Table 8-21 is retrieved from a single character location. Only the number of characters corresponding to the selected table entries are retrieved by the RVI instruction.
2. The RVI op code is a "privileged" op code that has special significance when Storage Protection is in effect.
3. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.
4. The format in which information is stored by an SVI instruction is shown in Table 8-20. Note that the information contained in the last character location is not restored by the RVI instruction.
5. The method of coding interrupt service routines is described in Section II, "Interrupt Processing."
6. The protect and proceed indicators, when present in the user's system, are not turned on automatically by the computer but instead must be turned on by programmed instructions, as follows: (1) a one-bit is set in the bit position which, when restored by the RVI instruction, indicates the status of the indicator; and (2) an RVI instruction with a V<sub>5</sub> bit of 1 in the variant character is executed, thereby turning on the appropriate indicator.
7. Unless the contents of the variant register are explicitly restored by this instruction, they are not altered by its execution; i.e., the variant character of the RVI instruction is not stored in the variant register.

EXAMPLE

Restore the contents of the variant register and auxiliary indicators register (AIR) that were previously stored by the SVI instruction example.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	V A R I A N T	L O C A T I O N	O P E R A T I O N C O D E	O P E R A N D S	
				1 2 3 4 5 6 7 8	14 15 20 21 62 63 80
1			RVI	STORE+2,05	
2					
3					
4					
5					
6					
7					
8					
9					



FORMAT

OP CODE	A ADDRESS	B ADDRESS
██████		

FUNCTION

The Monitor Call instruction causes the processor to enter the external interrupt mode (if the processor is not already in that mode). The following activities are automatically performed:

1. The EI interrupt source indicators are set to show that the Monitor Call instruction is the source of interruption, and the processor enters the external interrupt mode;
2. The settings of the arithmetic, comparison, address mode, and item-mark trap mode indicators are stored in the auxiliary indicators register (AIR);
3. The arithmetic indicators are cleared;
4. The contents of the sequence register (SR) and the external interrupt register (EIR) are interchanged, and the program branches to the instruction whose op code address was previously stored in EIR;
5. The processor switches to the three-character, non-trap mode.

WORD MARKS

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

SR	EIR	AAR	BAR
JI (con- tents of EIR)	NXT	A <sub>p</sub>	B <sub>p</sub>

NOTES

1. If this instruction is issued in the external interrupt mode, the results are unspecified.
2. The interrupt source indicators can be stored via an SVI instruction (see page 8-94). Their stored contents can then be interrogated by programmed instruction to determine the interrupt source.

EXAMPLE

Interrupt the central processor and branch to MONTOR, the location of the monitor program. The address tagged MONTOR, was previously stored in EIR.

# EASYCODER

CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_ OF \_\_\_\_

CARD NUMBER	Y	M	P	R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8					14 15	20 21	62 63 80
1						LCR	MONTOR, 66
2						}	
3							
4							
5						MC	
6							

<b>RNM</b>	RESUME NORMAL MODE	41 <sub>8</sub>
------------	--------------------	-----------------

FORMAT.

	OP CODE	A-ADDRESS	B-ADDRESS
a.			
b.			
c.			

FUNCTION

Format a: The RNM instruction causes an exit from the program being executed in the interrupt mode (external or internal) to the program which was interrupted. The activities performed depend on the type of interrupt mode in which the instruction is issued.

When the RNM instruction is issued in the external interrupt mode:

1. The EI mode indicators are turned off;
2. The arithmetic, comparison, address mode, and item-mark trap mode indicators are restored to the status specified by the auxiliary indicators register (AIR);
3. The A and B addresses of the RNM instruction are stored in the A- and B-address registers (AAR and BAR), respectively; and
4. The contents of the sequence register (SR) and the external interrupt register (EIR) are interchanged, and the program branches to the instruction whose op code address was initially stored in EIR when the external interrupt occurred.

When the RNM instruction is issued in the internal interrupt mode:

1. The II mode indicator is turned off;
2. The A and B addresses of the RNM instruction are stored in AAR and BAR, respectively; and
3. The contents of SR and the internal interrupt register (IIR) are interchanged, and the program branches to the instruction whose op code address was initially stored in IIR when the internal interrupt occurred.

Format b: This format operates like format a. except that the B address of the RNM instruction is not stored in BAR. The previous contents of BAR are not changed.

Format c: This format operates like format a. except that no instruction addresses are stored. The previous contents of AAR and BAR are not affected by this format.

WORD MARKS

Formats a, b, and c:

Word marks are not affected by this instruction.

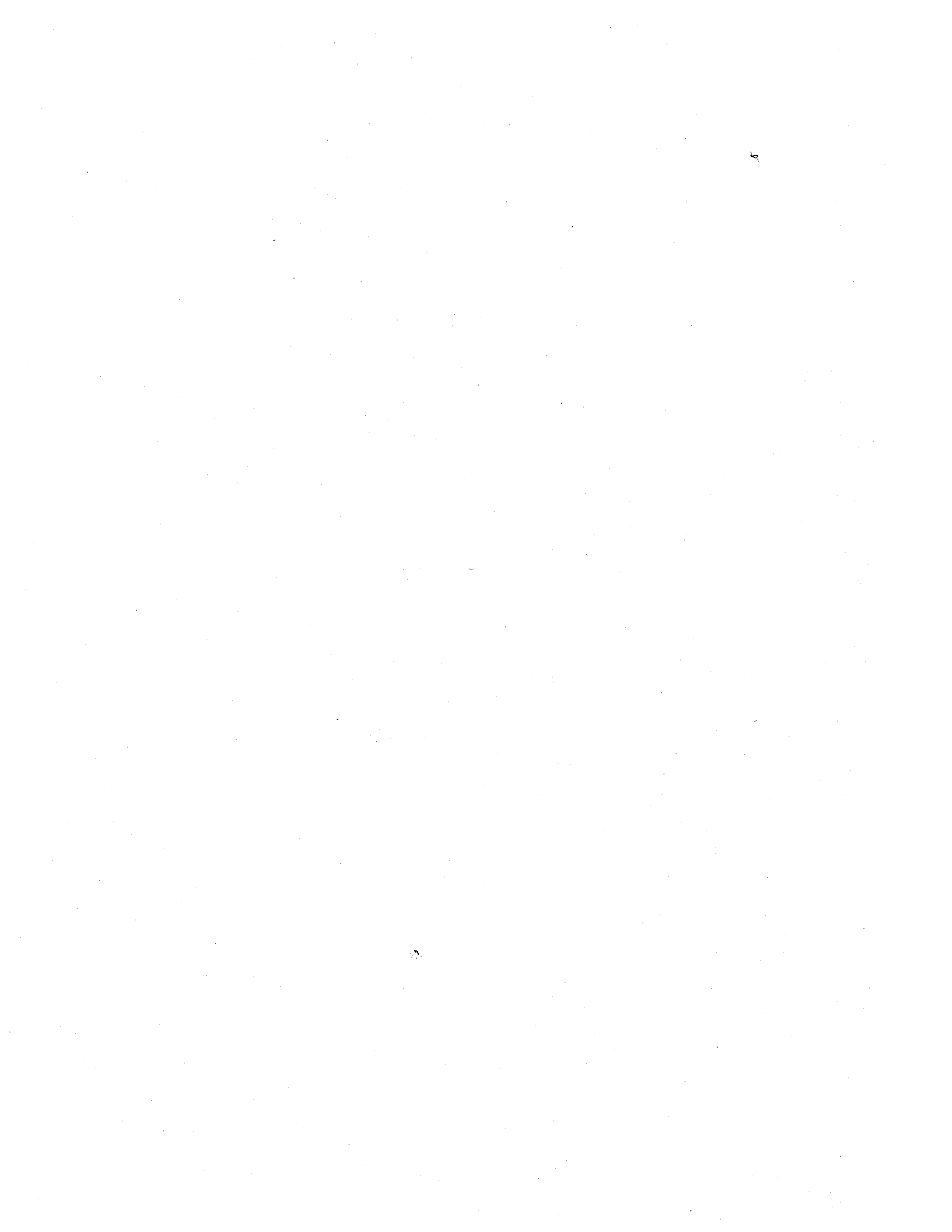
ADDRESS REGISTERS AFTER OPERATION

	SR	EIR	IIR	AAR	BAR	
<u>Format a:</u>	NXT	address of op code following RNM instruction	n/a	A	B	RNM ISSUED IN EXTERNAL INTERRUPT MODE
	NXT	n/a	address of op code following RNM instruction	A	B	RNM ISSUED IN INTERNAL INTERRUPT MODE
<u>Format b:</u>	NXT	address of op code following RNM instruction	n/a	A	B <sub>p</sub>	RNM ISSUED IN EXTERNAL INTERRUPT MODE
	NXT	n/a	address of op code following RNM instruction	A	B <sub>p</sub>	RNM ISSUED IN INTERNAL INTERRUPT MODE
<u>Format c:</u>	NXT	address of op code following RNM instruction	n/a	A <sub>p</sub>	B <sub>p</sub>	RNM ISSUED IN EXTERNAL INTERRUPT MODE
	NXT	n/a	address of op code following RNM instruction	A <sub>p</sub>	B <sub>p</sub>	RNM ISSUED IN INTERNAL INTERRUPT MODE

NOTES

1. The address of the instruction which follows the RNM instruction is stored in the appropriate interrupt register (EIR or IIR) when the RNM instruction is executed. This register therefore contains the address of the first instruction executed in the interrupt routine when the next interrupt of the same type occurs. This instruction should be an SVI instruction, which should be the first instruction executed in any interrupt service routine.





EDITING

8-106 ● MOVE CHARACTERS AND EDIT

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	████	████████	████████
b.	████	████████	
c.	████		

FUNCTION

Format a: The MCE instruction is used to insert identifying symbols and punctuation and to suppress unwanted 0's in a data field. The A-field of an MCE instruction contains the information to be edited. The B-field contains an edit control word which provides a framework for the edit operation. When an MCE instruction is executed, the data in the A-field is moved to the B-field where it is punctuated and formatted according to the edit control word already stored in that field.

NOTE: An LCA instruction can be used to load the control word into the field where the edited information will eventually go. For instance, if the edited information is to be printed, the control word should be loaded into the print image area and the address of this area should be used as the B-address of the MCE instruction.

Editing is performed according to the following rules:

RULE 1. Any character in the Series 2000 character set can be used in the edit control word. Those characters having special meanings are listed in Table 8-22. Any other character, if included in the edit control word, remains in the edited result in the position where written.

RULE 2. A word mark in the high-order position of the B-field controls the edit operation.

RULE 3. The number of replaceable characters in the edit control word must be at least as large as the number of characters in the A-field.

RULE 4. Data is transferred from the A-field character-by-character, from right to left. If a zero suppression symbol is not sensed in the edit control word, the edit operation terminates when the B-field word mark is sensed. A zero suppression symbol causes the edited result field to be scanned from left to right. During this scan, high-order 0's and commas are automatically replaced by blanks (unless an asterisk appears immediately to the left of the zero suppression symbol — see rule 5). Zero suppression is terminated by any of the following:

- a. a decimal digit from 1 through 9,
- b. a decimal point, or
- c. the location that initially contained the zero suppression symbol.

RULE 5. An asterisk immediately to the left of the zero suppression symbol in the control word causes high-order zeros and commas to be replaced by asterisks instead of blanks in a zero suppression operation. High-order blanks are also replaced by asterisks.

RULE 6. A dollar sign immediately to the left of the zero suppression symbol in the control word is replaced with an A-field character and causes the edited result to be rescanned following the zero suppression operation. During this scan, the dollar sign is "floated" to the left of the high-order significant digit in the edited result.

RULE 7. If the sign of the A-field is positive, any location in the B-field containing C, R, CR, or - (octal 23, 51, 75, and 40, respectively) has its contents replaced by a blank. Replacement terminates at the first 0 or blank in the B-field, but is resumed after transfer of the high-order significant digit of the A-field.

Table 8-22. Special Characters in MCE Instruction

CONTROL CHARACTER <sup>a</sup>	FUNCTION
b (blank)	Blanks are replaced with A-field characters such that the rightmost character in the A-field replaces the rightmost blank in the edit control word and all higher-order A-field characters replace successively higher-order blanks.
0 (zero)	This symbol specifies zero suppression. Its location in the control word is interpreted as the rightmost limit of zero suppression. It is replaced with an A-field character.
(decimal point)	The decimal point remains in the edited field in the position where written.
, (comma)	Commas remain in the edited field where written unless zero suppression is specified (see rule 4). Commas in control word positions to the left of the high-order character transferred from the A-field are replaced by blanks.
C <sub>R</sub> , CR (credit) 0̄ (minus) <u>NOTE:</u> 0̄ is printed as a minus symbol.	The credit or minus symbol is undisturbed if the sign in the units position of the A-field is negative. If the sign is positive, the credit (or minus) symbol is blanked out. A credit (or minus) symbol transferred from the A-field is not subject to sign control.
37 <sub>8</sub>	An octal 37 is replaced by a blank in the edited field.
* (asterisk)	The asterisk remains in the edited field in the position where written unless it appears immediately to the left of the zero suppression symbol (see rule 5).
\$ (dollar sign)	The dollar sign remains in the edited field in the position where written unless it appears immediately to the left of the zero suppression symbol (see rule 6).

<sup>a</sup>If assembling under Easycoder C with a Type 222 Printer, 0̄ will assemble as 57<sub>8</sub> and be printed as 1/2. To edit a minus symbol, a dash (-) must be used; it assembles as 40<sub>8</sub>.



Format b: The data contents of the A field are edited and stored in the field specified by the contents of the B-address register (BAR) according to the rules outlined above.

Format c: The data field specified by the contents of the A-address register (AAR) are edited and stored in the field specified by the contents of BAR according to the rules outlined above.

### WORD MARKS

#### Formats a, b, and c:

Both the A field and the B field must have defining word marks. The A-field word mark terminates the transfer of data from the A field. The B-field word mark terminates the edit operation if no zero suppression symbol is sensed in the edit control word or if automatic dollar sign insertion is specified in conjunction with zero suppression. The B-field word mark is erased after terminating the edit.

If zero suppression is specified, a word mark is automatically set in the location containing the zero suppression symbol. When this word mark is sensed during the reverse scan associated with the zero suppression operation, it is erased and, if automatic dollar sign insertion is not called for, the edit operation terminates.

#### Address Registers After Operation

Unspecified unless floating dollar sign is processed. After processing of the terminating location of zero suppression, this location and successive locations to the left are examined until one is found that contains a blank. The \$ character is stored in this location and the operation terminates, leaving the B-address register one location to the left of the \$.

### NOTES

1. The zone bits in the units position of the A-field are cleared to 0 when moved to the B-field. Therefore the value of the character in the units position in the A-field may change when moved to the B-field. For example, an F in the units position of the A-field will appear as a 6 in the result field.
2. Floating dollar sign insertion and automatic asterisk insertion cannot be performed in the same edit operation.
3. The contents of the variant register are unspecified following the execution of this instruction. Therefore, an instruction requiring a variant character cannot be chained following an MCE instruction.

EXAMPLES<sup>1</sup>

Data Field (A Field)	① 00009̄
Control Word (B Field)	ⓑ bb, bb0. bb&&0̄
Result of Edit	. 99 -

Example 1.

Data Field (A Field)	② 5454986
Control Word (B Field)	ⓑ bb&bb&bbb
Result of Edit	254 54 986

Example 2.

Data Field (A Field)	① 00450 <sup>†</sup>
Control Word (B Field)	Ⓢ b, bb0. bb&CR*
Result of Edit	\$ 4.50 *

Example 3.

Data Field (A Field)	① 0897445
Control Word (B Field)	ⓑ bbb, b\$0. bb
Result of Edit	\$8, 974. 45

Example 4.

Data Field (A Field)	① 010450
Control Word (B Field)	ⓑ b, b*0. bb
Result of Edit	***104. 50

Example 5.

---

<sup>1</sup>The character (378) is shown as an ampersand (&) in these examples. However, the ampersand is not the only equivalent of 37<sub>8</sub> as shown in Table B-6.



## INPUT/OUTPUT

- 8-116 ● PERIPHERAL DATA TRANSFER
- 8-139 ● PERIPHERAL CONTROL AND BRANCH

## INPUT/OUTPUT CONTROL OPERATIONS

Effective control over data transfers between the central processor and peripheral units and over the peripheral units themselves is maintained by the use of two basic instructions: Peripheral Data Transfer (PDT), and Peripheral Control and Branch (PCB). The PDT instruction is used to initiate data transfer operations and certain other related operations, such as backspace magnetic tape and advance the printer form.

The PCB instruction can perform four distinct functions: (1) it initiates strictly mechanical (non-data transfer) operations such as magnetic tape rewinds and card rejections; (2) it causes a program branch to be performed contingent upon the settings of peripheral condition indicators; (3) it changes the operational mode of a peripheral control; and (4) it allows a peripheral control to interrupt (or directs the control not to interrupt) the central processor when data transfer is completed.

Detailed programming and operating information for Series 200/2000 peripheral devices is provided in separate publications. The remainder of this section is a summary of the PDT and PCB instructions, based on the assumption that the user is familiar with the contents of the applicable documents. In all applicable cases, the coding summary for a device is followed by a reference to the specific Honeywell manual or information bulletin where additional information can be found.

### SELECTING RWC ASSIGNMENTS FOR USE IN PDT INSTRUCTIONS

As described below, the first control character (C1) in a PDT instruction is referred to as the "read/write channel assignment." This six-bit character specifies the read/write channel(s) selected to complete the data path. When coding a PDT instruction, the programmer may refer to Table 8-24 to select an RWC assignment. The following discussion concerns the considerations involved in selecting RWC assignments and the correspondence between achievable data transfer rates and RWC assignments.

#### Considerations in Selecting RWC Assignments

At least four factors must be considered when selecting an RWC assignment. These factors are: (1) the data transfer rate of the device being addressed; (2) the processor being used; (3) the I/O sector to which the device is attached; and (4) the necessity of being upward compatible.

DEVICE DATA TRANSFER RATE

The first consideration in selecting an RWC assignment is the rated speed at which the device being addressed transfers data to or from main memory. The one or more RWC's assigned to an operation must receive memory accesses often enough to keep up with the I/O data transfer rate of the device. For example, the RWC assignment used in a PDT instruction which addresses a Type 258 Disk Pack Drive must designate a data transfer capacity high enough to keep pace with the device's 208,000-character-per-second transfer rate.

However, due to mechanical tolerances, some devices may transfer data at instantaneous rates higher than their nominal transfer rates. In a few such cases, the devices require an RWC assignment having a greater data handling capacity than would be required if the nominal data transfer rate were maintained. As an example, a Type 204B-5 tape drive using a density of 556 bits per inch requires an RWC assignment having a data handling capacity of 167,000 characters per second, even though the nominal transfer rate for this device is less than 83,300 characters per second.

Table 8-23 lists the minimum RWC capacity requirements for each Series 2000 peripheral device.

Table 8-23. Minimum RWC Capacity Requirements for Series 200/2000 Peripheral Devices

Device	Minimum RWC Capacity Required
112 Printer	167 KC
112-3 Printer	167 KC
122-3 Printer	167 KC
122-4 Printer	167 KC
122-6 Printer	167 KC
123 Card Reader	83.3 KC
123-2 Card Reader	83.3 KC
123-4 Card Reader	83.3 KC
204B-1, -2 Magnetic Tape Units	83.3 KC
204B-3, -4 Magnetic Tape Units	83.3 KC
204B-5 Magnetic Tape Unit (200 bpi)	83.3 KC
204B-5 Magnetic Tape Unit (556 bpi)	167 KC
204B-7 Magnetic Tape Unit	83.3 KC
204B-8 Magnetic Tape Unit (200/556 bpi)	83.3 KC
204B-8 Magnetic Tape Unit (800 bpi)	167 KC
204B-9 Magnetic Tape Unit (200/556 bpi)	83.3 KC
204B-9 Magnetic Tape Unit (800, 1200 bpi)	167 KC
204D-1 Magnetic Tape Unit	83.3 KC
204D-3 Magnetic Tape Unit (800 bpi)	83.3 KC
204D-3 Magnetic Tape Unit (1600 bpi)	167 KC
204D-5 Magnetic Tape Unit (800 bpi)	167 KC
204D-5 Magnetic Tape Unit (1600 bpi)	250 KC
204D-3A Magnetic Tape Unit (800 bpi)	83.3 KC
204D-3A Magnetic Tape Unit (1600 bpi)	167 KC
204D-5A Magnetic Tape Unit (800 bpi)	167 KC

Table 8-23 (cont). Minimum RWC Capacity Requirements for Series 200/2000 Peripheral Devices

Device	Minimum RWC Capacity Required
204D-5A Magnetic Tape Unit (1600 bpi)	250 KC
204F-1 Magnetic Tape Unit	83.3 KC
204F-3 Magnetic Tape Unit (800 bpi)	83.3 KC
204F-3 Magnetic Tape Unit (1600 bpi)	167 KC
204F-5 Magnetic Tape Unit (800 bpi)	167 KC
204F-5 Magnetic Tape Unit (1600 bpi)	250 KC
209 Paper Tape Reader	83.3 KC
209-2 Paper Tape Reader	83.3 KC
210 Paper Tape Punch	83.3 KC
212 On-Line Adapter	167 KC
212-1 Central Processor Adapter	167 KC
212-2 CP Memory-to-Memory Transfer Unit	167 KC
213-3 Interval Timer	
213-4 Time-of-Day Clock	83.3 KC
214-1 Card Punch	83.3 KC
214-2 Card Reader Punch -- Read	83.3 KC
214-2 Card Reader Punch -- Punch	83.3 KC
220-3, -6 Consoles	83.3 KC
220-8 VICC Console	
222 Printers (all versions) <sup>1</sup>	167 KC
223 Card Reader	83.3 KC
223-2 Card Reader	83.3 KC
232 MICR Reader-Sorter and Control	83.3 KC
233-2 MICR Control for B103	83.3 KC
236 High-Speed MICR Reader-Sorter	
243 Optical Document Reader	83.3 KC
258 Disk Pack Drive	250 KC
258B Disk Pack Drive	167 KC
259 Disk Pack Drive	250 KC
259B Disk Pack Drive	167 KC
266 High-Speed Disk File	333 KC
270A Drum	167 KC
273 Disk Pack Drive	250 KC
274 Disk Pack Drive	250 KC
275-2 Disk Pack Drive	250 KC
277-2 Disk Pack Drive	167 KC
278-5 Disk Pack Drive	500 KC
278-6 Disk Pack Drive	500 KC
278-7 Disk Pack Drive	500 KC
278-8 Disk Pack Drive	500 KC
278-9 Disk Pack Drive	500 KC
279-2 Disk Pack Drive	167 KC
281 Single-Line Communication Controllers (all versions) <sup>2</sup>	83.3 KC
286-1, -2, -3 Multiline Communication Controllers	83.3 KC
286-4, -5, -6, -7 Message-Mode Multiline Communication Controllers <sup>3</sup>	83.3 KC
287-1 USASCII AUTODIN Communication Controller <sup>2</sup>	83.3 KC

<sup>1</sup>When a 222-3, -4, -6, or -7 printer is equipped with the Print Buffer (Feature 036), the transfer rate must be either 83.3 KC or 167 KC.

<sup>2</sup>The 281-2F and 287-1 controllers require exclusive assignment of two 83.3 KC RWC's when operating in the full-duplex mode.

<sup>3</sup>The maximum RWC capacity that can be assigned to a 286-4, -5, -6, or -7 is 167 KC.

## THE PROCESSOR BEING USED

Some Series 2000 processors are available with a basic and an expanded I/O configuration. These I/O configurations include different numbers of RWC's. Therefore, the identity of the processor being used and whether it is an expanded configuration must be known in order to determine which RWC assignments are available for use.

The multicharacter processors are equipped with "variable-speed" read/write channels, and no more than two RWC's (a primary and the corresponding auxiliary) are ever made busy by a single RWC assignment. RWC's not made busy by a high-speed transfer are available for use in other operations.

An additional capability of the multicharacter processors makes variable-speed read/write channels even more attractive: in these processors, it is possible to assign RWC's outside their "home" sectors. For example, an RWC normally used only in sector 1 can be assigned to an I/O transfer operation in sector 2. Such assignments of RWC's are accomplished by means of the "sector bits" of I/O control character C2, or through the use of "escape codes." This facility enables the programmer to transfer RWC's temporarily to a sector performing several low-speed operations from another sector in which one or two operations are using the sector's entire data handling capacity.

## INPUT/OUTPUT SECTOR TO WHICH DEVICE IS CONNECTED

Each input/output sector in a Series 2000 processor has a maximum total data transfer capacity of 500,000 characters per second.

In general, the RWC assigned to an operation should be associated with the sector to which the addressed device is connected. However, as indicated above, this rule can be circumvented to advantage in multicharacter processors by the proper selection of C2 sector bits or escape codes.

## UPWARD COMPATIBILITY

Because of the manner in which upward compatibility has been consistently implemented in Series 2000 processors, very little consideration need be given to this factor when selecting RWC assignment codes. The one case where such consideration must be given is when assigning a primary RWC for which there is no corresponding auxiliary channel in the processor being programmed to an operation faster than 83,000 characters per second. An example of such a case is the assignment of the single channel RWC 2 to a drum read operation (102,000 characters per second) to be performed in a basic Type 2040A processor. In the basic processor, RWC 2 can handle transfer rates up to 167,000 characters per second. However, in an expanded Type 2040A (with PM1A40) RWC's 2 and 2' can handle only 83,000 characters per



second apiece unless they are interlocked. Thus, if the attempt were made to run the basic 2040A program on an expanded 2040A, the RWC 2 alone would not be able to handle the drum's transfer rate.

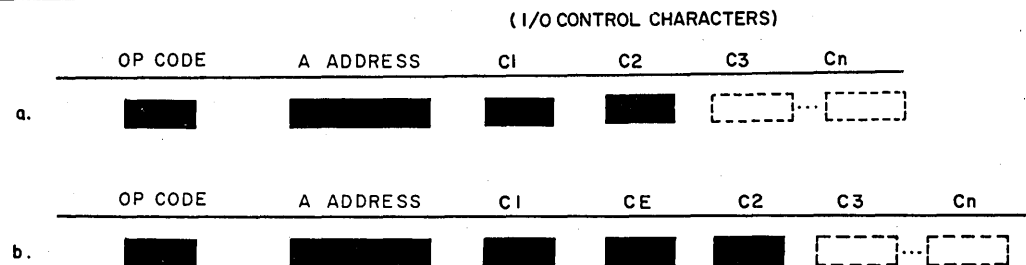
In order to avoid such problems, the following general rule should be followed:

The RWC assignment in a PDT instruction addressing a device which operates between 83,300 and 167,000 characters per second should be such that it would interlock the primary channel and its auxiliary if the program were run in a processor equipped with both channels; i. e., its high-order digit should be 5 or 7.

Clearly, there is no need to specify the "interlock" assignment if the device runs slower than 83,300 characters per second. Rather, in the interest of making more RWC's available for use in other operations, it is wise in such cases to specify the single-channel assignment.

PDT	PERIPHERAL DATA TRANSFER	66 <sub>8</sub>
-----	--------------------------	-----------------

FORMAT



FUNCTION

Format a: The PDT instruction causes data to be transferred between a peripheral device and the main memory area whose leftmost<sup>1</sup> location is designated by the A address. Data transfer is terminated according to the data medium employed. Input/output control characters specify the data path through which the transfer is to be accomplished, as indicated in Tables 8-24 and 8-26.

Format b: Data is transferred between a peripheral device and the main memory area whose leftmost<sup>1</sup> location is designated by the A address. Data transfer is terminated according to the data medium employed. Input/output control characters and an escape code specify the data path through which the transfer is to be accomplished, as indicated in Tables 8-24, 8-25, and 8-26.

---

<sup>1</sup> Rightmost if Read Reverse is specified.

Table 8-24. Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2041		
Data Handling Capacity (characters per second)	RWC's Interlocked and Made Busy (Note 8)	RWC Assignment Code
<u>Sector 1</u>		
83,000	<u>1</u>	11
83,000	<u>1</u> '	15
167,000	<u>1</u> , <u>1</u> '	51
167,000	<u>2</u>	52
167,000	<u>3</u>	53
250,000	<u>1</u> ', <u>3</u>	55
333,000	<u>2</u> , <u>3</u>	56
500,000	<u>1</u> , <u>1</u> ', <u>2</u> , <u>3</u>	54
<u>Sector 2</u>		
83,000	<u>4</u>	31
83,000	<u>4</u> '	35
167,000	<u>4</u> , <u>4</u> '	71
167,000	<u>5</u>	72
167,000	<u>6</u>	73
250,000	<u>4</u> ', <u>6</u>	75
333,000	<u>5</u> , <u>6</u>	76
500,000	<u>4</u> , <u>4</u> ', <u>5</u> , <u>6</u>	74
Type 2041A (Basic)		
<u>Sector 1</u>		
167,000	<u>2</u>	52
167,000	<u>3</u>	53
333,000	<u>2</u> , <u>3</u>	56
500,000	<u>1</u> , <u>1</u> ', <u>2</u> , <u>3</u>	54
83,000	<u>1</u>	11
83,000	<u>1</u> '	15
167,000	<u>1</u> , <u>1</u> '	51
250,000	<u>1</u> ', <u>3</u>	55
<u>Sector 2</u>		
83,000	<u>4</u>	31
83,000	<u>4</u> '	35
167,000	<u>4</u> , <u>4</u> '	71
167,000	<u>5</u>	72
167,000	<u>6</u>	73
250,000	<u>4</u> ', <u>6</u> (note 9)	75
333,000	<u>5</u> , <u>6</u> (note 9)	76
500,000	<u>4</u> , <u>4</u> ', <u>5</u> , <u>6</u> (note 9)	74

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2041A (with PM1A40)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2041A (with PM1A40 and PM1B40)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'
<u>Sector 2D</u>				
83,000 (167,000)	8	22	1, 5	2 (2, 2')
83,000 (167,000)	9	23	1, 5	3 (3, 3')
83,000	8'	26	5	2'
83,000	9'	27	5	3'
250,000	8	60	2, 5	1, 2, 2'
167,000	8 (8, 8')	62	5	2, 2'
167,000	9 (9, 9')	63	5	3, 3'
500,000	9	64	2, 5	1, 2, 3, 1', 2', 3'
250,000	9	65	2, 5	1', 3, 3'
333,000	9	66	2, 5	2, 3, 2', 3'
None		00	4	Not applicable
None		77	3	Not applicable

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2051C				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2051A (Basic)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2051A (with PM1A50)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'
<u>Sector 2D</u>				
83,000 (167,000)	8	22	1, 5	2 (2, 2')
83,000 (167,000)	9	23	1, 5	3 (3, 3')
83,000	8'	26	5	2'
83,000	9'	27	5	3'
250,000	8	60	2, 5	1, 2, 2'
167,000	8 (8, 8')	62	5	2, 2'
167,000	9 (9, 9')	63	5	3, 3'
500,000	9	64	2, 5	1, 2, 3, 1', 2', 3'
250,000	9	65	2, 5	1', 3, 3'
333,000	9	66	2, 5	2, 3, 2', 3'
None		00	4	Not applicable
None		77	3	Not applicable

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2051A (with PM1A50 and PM1B50)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'



Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2051A (with PM1A50 and PM1B50)(cont)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 2B</u>				
250,000	2'	40	2, 5	1, 2, 2'
167,000	1'	41	5	1, 1'
167,000	2'	42	5	2, 2'
167,000	3'	43	5	3, 3'
500,000	3'	44	2, 5	1, 2, 3, 1', 2', 3'
250,000	3'	45	2, 5	1', 3, 3'
333,000	3'	46	2, 5	2, 3, 2', 3'
<u>Sector 2C</u>				
167,000	4'	01	5	1, 1'
167,000	5' (5', 9')	02	1, 5	2, 2'
167,000	6' (6', 4')	03	1, 5	3, 3'
500,000	6'	04	2, 5	1, 2, 3, 1', 2', 3'
250,000	6'	05	2, 5	1', 3, 3'
333,000	6'	06	2, 5	2, 3, 2', 3'
250,000	5'	07	2, 5	1, 2, 2'
<u>Sector 2D</u>				
83,000 (167,000)	8 (8, 8')	22	1, 5	2 (2, 2')
83,000 (167,000)	9 (9, 9')	23	1, 5	3 (3, 3')
83,000	8'	26		2'
83,000	9'	27	5	3'
250,000	8	60	2, 5	1, 2, 2'
167,000	9'	61	5	1, 1'
167,000	8 (8, 8')	62	1, 5	2, 2'
167,000	9 (9, 9')	63	1, 5	3, 3'
500,000	9	64	1, 5	1, 2, 3, 1', 2', 3'
250,000	9	65	2, 5	1', 3, 3'
333,000	9	66	2, 5	2, 3, 2', 3'
None		77	3, 5	Not applicable
None		00	4, 5	Not applicable

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2061				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'
<u>Sector 2D</u>				
83,000 (167,000)	8	22	1, 5	2 (2, 2')
83,000 (167,000)	9	23	1, 5	3 (3, 3')
83,000	8'	26	5	2'
83,000	9'	27	5	3'
250,000	8	60	2, 5	1, 2, 2'
167,000	8 (8, 8')	62	5	2, 2'
167,000	9 (9, 9')	63	5	3, 3'
500,000	9	64	2, 5	1, 2, 3, 1', 2', 3'
250,000	9	65	2, 5	1', 3, 3'
333,000	9	66	2, 5	2, 3, 2', 3'
None		00	4	Not applicable
None		77	3	Not applicable

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2071				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 1</u>				
83,000	1	11		1
83,000 (167,000)	2	12	1	2 (2, 2')
83,000 (167,000)	3	13	1	3 (3, 3')
83,000	1'	15		1'
83,000	2'	16	6	2'
83,000	3'	17	6	3'
250,000	2	50		1, 2, 2'
167,000	1 (1, 1')	51		1, 1'
167,000	2 (2, 2')	52		2, 2'
167,000	3 (3, 3')	53		3, 3'
500,000	3	54	2	1, 2, 3, 1', 2', 3'
250,000	3	55	2	1', 3, 3'
333,000	3	56	2	2, 3, 2', 3'
<u>Sector 2A</u>				
83,000	4	31		1
83,000 (167,000)	5	32	1	2 (2, 2')
83,000 (167,000)	6	33	1	3 (3, 3')
83,000	4'	35		1'
83,000	5'	36	6	2'
83,000	6'	37	6	3'
250,000	5	70	2, 6	1, 2, 2'
167,000	4 (4, 4')	71		1, 1'
167,000	5 (5, 5')	72		2, 2'
167,000	6 (6, 6')	73		3, 3'
500,000	6	74	2	1, 1', 2, 2', 3, 3'
250,000	6	75	2	1', 3, 3'
333,000	6	76	2	2, 2', 3, 3'

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

Type 2071 (cont)				
Data Handling Capacity (characters per second)	Read/Write Counter	RWC Code	RWC Code Notes	Time Slot Used on RWC-Assigned Sector
<u>Sector 2B</u>				
250,000	2'	40	2, 5	1, 2, 2'
167,000	1'	41	5	1, 1'
167,000	2'	42	5	2, 2'
167,000	3'	43	5	3, 3'
500,000	3'	44	2, 5	1, 2, 3, 1', 2', 3'
250,000	3'	45	2, 5	1', 3, 3'
333,000	3'	46	2, 5	2, 3, 2', 3'
<u>Sector 2C</u>				
167,000	4'	01	5	1, 1'
167,000	5' (5', 9')	02	1, 5	2, 2'
167,000	6' (6', 4')	03	1, 5	3, 3'
500,000	6'	04	2, 5	1, 2, 3, 1', 2', 3'
250,000	6'	05	2, 5	1', 3, 3'
333,000	6'	06	2, 5	2, 3, 2', 3'
250,000	5'	07	2, 5	1, 2, 2'
<u>Sector 2D</u>				
83,000 (167,000)	8 (8, 8')	22	1, 5	2 (2, 2')
83,000 (167,000)	9 (9, 9')	23	1, 5	3 (3, 3')
83,000	8'	26		2'
83,000	9'	27	5	3'
250,000	8	60	2, 5	1, 2, 2'
167,000	9'	61	5	1, 1'
167,000	8 (8, 8')	62	1, 5	2, 2'
167,000	9 (9, 9')	63	1, 5	3, 3'
500,000	9	64	1, 5	1, 2, 3, 1', 2', 3'
250,000	9	65	2, 5	1', 3, 3'
333,000	9	66	2, 5	2, 3, 2', 3'
None		77	3, 5	Not applicable
None		00	4, 5	Not applicable

Table 8-24 (cont). Description of PDT I/O Control Character C1  
(RWC Assignment)

NOTES

1. If the extended I/O indicator is off, the RWC's are interlocked for a 167 KC rate; the primary RWC is used for address storage and the primary and auxiliary RWC's are both made busy.
2. These codes may be used only with peripheral controls requiring data transfer rates greater than 167 KC. These codes, therefore, are not legal with escape codes of 14, 15, 16, and 17, unless otherwise specified in the manual for the peripheral control.
3. Code used only for PCB (control and unconditional branch) instruction.
4. Code used only for PCB (peripheral control test) instruction.
5. Codes associated with sectors 2B, 2C, and 2D must be accompanied by a C2 or CE character specifying a sector.
6. These codes are legal only when the extended I/O feature is installed.
7. Data handling capacity of a buffered sector of the 2041A, 2051C, and 2060 is reduced by 50% when a direct-access escape code (14 or 17) is used. Data handling capacity of a buffered sector of the 2051A or 2071 is reduced by 75% when a direct-access escape code (14, 15, 16, or 17) is used.
8. On the 2041 and basic 2041A processors, underlined numbers identify the RWC whose corresponding starting and current location counters are used in the operation.
9. Uses RWC 6 for address storage during data transfer. RWC 2 cannot be active while RWC 5 is active, nor can RWC 3 be active while RWC 6 is active.

ESCAPE CODE (CE)

The escape code is part of format b. of the PDT instruction. When an escape code is included in a PDT instruction, the read/write channel(s) designated by C1 is assigned to an I/O operation in the sector indicated by the escape code (the sector bits of the C2 control character are ignored). The addressed device must be connected to the sector designated by the escape code. See Table 8-25 for sector designations of escape codes.

Table 8-25. Escape Codes

Escape Code	Sector Designated	Processor Affected					
		2041	2041A	2051C	2051A	2061	2071
10	Sector 1		X	X	X	X	X
11	Test RWC only		X	X	X	X	X
12	Sector 2 <sup>1</sup>		X	X	X	X	X
14	Sector 2A Direct-Access		X	X	X	X	X
15	Sector 2B Direct-Access				5		X
16	Sector 2C Direct-Access				5		X
17	Sector 2D Direct-Access		3		4	X	X
54	Sector 2A Buffered Mode		2	X	4	X	X
55	Sector 2B Buffered Mode				5		X
56	Sector 2C Buffered Mode				5		X
57	Sector 2D Buffered Mode		3		4	X	X

<sup>1</sup> A control character that references Sector 2 will be interpreted as Sector 2A, buffered (escape code 54).  
<sup>2</sup> Applicable only when PM1A40 is present.  
<sup>3</sup> Applicable only when PM1A40 and PM1B40 are present.  
<sup>4</sup> Applicable only when PM1A50 is present.  
<sup>5</sup> Applicable only when PM1A50 and PM1B50 are present.

Table 8-26. Description of PDT I/O Control Character C2 (Peripheral Control Designation)

Control Character	Description
C2	<p>This six-bit character specifies the type of data transfer (input or output), the sector (if there is no escape code), and the logical address of the peripheral control to be used in the data transfer.</p> <p style="text-align: center;">C2</p> <p style="text-align: center;">X X X X X X</p> <p style="text-align: right;">Peripheral Control Address Bits</p> <p style="text-align: right;">Sector Bits</p> <p style="text-align: right;">Input/Output Bit</p> <p><b>Input/Output Bit:</b> This bit specifies the direction of data transfer when a peripheral control capable of both reading and writing is involved in the transfer. When such a bidirectional control is used,</p> <p style="padding-left: 40px;">0 = transfer data from memory to the peripheral control (output),</p> <p style="padding-left: 40px;">1 = transfer data to memory from the peripheral control (input).</p> <p>For buffered operations, the I/O bit must agree with the actual direction of data transfer. For direct-access operations, the input/output bit is used together with the peripheral control address bits to identify the peripheral control involved in the operation.</p>

Table 8-26 (cont). Description of PDT I/O Control Character C2  
(Peripheral Control Designation)

Control Character	Description																						
C2 (cont)	<p><u>Sector Bits:</u> If there is no escape code, these bits specify the sector in which the peripheral control is connected. They are specified as follows:</p> <p style="padding-left: 40px;">Sector 1 = 00<sup>1</sup> Sector 2 = 10<sup>1</sup></p> <p><u>Peripheral Control Address Bits:</u> These three bits, in conjunction with the input/output bit, identify the address of the peripheral control involved in the operation. It is recommended that the following octal configurations be used for control character C2 in order to provide uniformity among Series 200 and Series 2000 installations:</p> <table border="0" style="width: 100%; margin-left: 40px;"> <thead> <tr> <th style="text-align: left;"><u>Peripheral Control</u></th> <th style="text-align: left;"><u>Octal Address</u><sup>2</sup></th> </tr> </thead> <tbody> <tr> <td>Magnetic Tape Control</td> <td>00 (output)</td> </tr> <tr> <td></td> <td>40 (input)</td> </tr> <tr> <td>Paper Tape Reader or Card Reader<sup>3</sup></td> <td>41</td> </tr> <tr> <td>Paper Tape Punch or Card Punch<sup>3</sup></td> <td>01</td> </tr> <tr> <td>Printer</td> <td>02</td> </tr> <tr> <td>Type 212 On-Line Adapter</td> <td>42</td> </tr> <tr> <td>Console</td> <td>07 (output)</td> </tr> <tr> <td></td> <td>47 (input)</td> </tr> <tr> <td>Disk Control</td> <td>04 (output)</td> </tr> <tr> <td></td> <td>44 (input)</td> </tr> </tbody> </table>	<u>Peripheral Control</u>	<u>Octal Address</u> <sup>2</sup>	Magnetic Tape Control	00 (output)		40 (input)	Paper Tape Reader or Card Reader <sup>3</sup>	41	Paper Tape Punch or Card Punch <sup>3</sup>	01	Printer	02	Type 212 On-Line Adapter	42	Console	07 (output)		47 (input)	Disk Control	04 (output)		44 (input)
<u>Peripheral Control</u>	<u>Octal Address</u> <sup>2</sup>																						
Magnetic Tape Control	00 (output)																						
	40 (input)																						
Paper Tape Reader or Card Reader <sup>3</sup>	41																						
Paper Tape Punch or Card Punch <sup>3</sup>	01																						
Printer	02																						
Type 212 On-Line Adapter	42																						
Console	07 (output)																						
	47 (input)																						
Disk Control	04 (output)																						
	44 (input)																						
<p><sup>1</sup>In the Types 2041A (equipped with PM1A40 and PM1B40), 2051A (equipped with PM1A50 and PM1B50), 2051C, 2061, and 2071, 10 designates sector 2A.</p> <p><sup>2</sup>These recommended addresses are made up of (1) the input/output bit and (2) the peripheral control and address bits. In Series 2000 systems in which sector designations apply, the specification of the sector bits may alter these addresses.</p> <p><sup>3</sup>In Series 2000 installations containing a card reader/punch unit, these recommended addresses apply. However, if the installation contains a second card reader, the reader portion of the card reader punch should be assigned the address 43<sub>g</sub> and the second card reader assigned the address 41<sub>g</sub>.</p>																							

Additional Parameters (C3 through Cn)

The specific use of these control characters is dependent upon the type of peripheral device addressed. A summary of coding for these characters may be found in Tables 8-27 through 8-33.

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or item marks. However, record marks may terminate the data transfer, depending upon the device used and the operation performed (see the specific Honeywell Publications).

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A	B <sub>p</sub>

NOTES

1. If either the read/write channel or the peripheral control (specified by C1 and C2, respectively) is found "busy" during the extraction of a PDT instruction, the instruction is re-extracted: the contents of SR are set back to the address of the PDT op code, and the extraction process begins again. This process allows the processor to respond to interrupt signals that may occur while the PDT instruction is awaiting the availability of a read/write channel or peripheral control.
2. The PDT op code is a "privileged" op code that has special significance when Storage Protection is in effect (see Section II).
3. Format b. of the PDT instruction is applicable only to multicharacter processors.
4. Unspecified central processor activity can occur when an attempt is made to execute a PDT instruction having a read/write channel assignment (C1) of zero. It is therefore imperative that every PDT instruction contain some legal RWC assignment.
5. Control character C1 of a PDT instruction is stored in the variant register.
6. When buffered sectors are included in a processor, additional programming considerations apply to the PDT instruction (see Section II).

EXAMPLE

Read a card into the 80-character image area tagged CREAD. Use RWC2 and assume that the card reader control is assigned to the logical address of octal 41. Note that the data transfer rate in a card reading operation is less than 83,000 characters per second.

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	TYPE	M	R	A	P	LOCATION	OPERATION CODE	OPERANDS						
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
							PDT	CREAD, 12, 41						
1														
2														



C4						
6	5	4	3	2	1	
0	0	0	0	0	0	Record Mark Termination: 4 x 3 (9-track) or 1 x 1 (7-track).
0	0	0	0	0	1	Record Mark Termination: 2 x 1 (9-track).
0	0	0	0	1	0	Record Mark Termination: ASCII Subset (9-track).
0	0	0	1	0	0	Record Mark Termination: EBCDIC Subset (9-track).
0	0	1	0	0	0	File Mark Search: 9-track 4 x 3 (17 <sub>8</sub> ); 7-track (17 <sub>8</sub> ).
0	0	1	0	0	1	File Mark Search: 9-track 2 x 1 (23 <sub>8</sub> ).
0	1	0	0	0	0	Count Field Termination: 4 x 3 (9-track) or 1 x 1 (7-track).
0	1	0	0	0	1	Count Field Termination: 2 x 1 (9-track).
0	1	0	0	1	0	Count Field Termination: ASCII Subset (9-track).
0	1	0	0	1	1	Count Field Termination: Load Mode (1 x 1) (9-track).
0	1	0	1	0	0	Count Field Termination: EBCDIC Subset (9-track).

Figure 8-12. C4 Variant for 9-Track Tape Units

Table 8-27. Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
CARD	READ	X X	X X	none	none	none	none
	PUNCH	X X	X X	none	none	none	none
See: <u>Type 223/223-2 Card Reader</u> (Order No. BC39), <u>Type 214-1 Card Punch</u> (Order No. BC03), <u>Type 214-2 Card Reader/Punch</u> (Order No. BC04),							
PAPER TAPE	READ	X X	X X	See Table 8-28	none	none	none
	PUNCH	X X	X X	See Table 8-29	none	none	none
See: <u>Types 209, 209-2, and 210 Paper Tape Equipment</u> (Order No. BC42)							
PRINTER	PRINT	X X	X X	See Table 8-30	none	none	none
See: <u>Type 222 Printers</u> (Order No. BC75), <u>Type 222-7 Printer</u> (Order No. BJ23)							
MAGNETIC TAPE 1/2-INCH	READ FORWARD	X X	X <sup>1</sup> X	6 D <sup>3</sup> (D=tape drive, 0 - 7) <sup>6</sup>	See Figure 8-12	none	none
	READ REVERSE (Feature 010 or 011)	X X	X <sup>1</sup> X	2 D <sup>4</sup> (D=tape drive, 0 - 7) <sup>6</sup>	See Figure 8-12	none	none
	WRITE	X X	X <sup>2</sup> X	2 D <sup>5</sup> (D=tape drive, 0 - 7) <sup>6</sup>	See Figure 8-12	none	none
	SPACE FORWARD	X X	X <sup>1</sup> X	4 D (D=tape drive, 0 - 7) <sup>6</sup>	See Figure 8-12	none	none
	BACKSPACE	X X	X <sup>1</sup> X	0 D (D=tape drive 0 - 7) <sup>6</sup>	See Figure 8-12	none	none
	ERASE	X X	X <sup>2</sup> X	0 D (D=tape drive, 0 - 7) <sup>6</sup>	See Figure 8-12	none	none
See: <u>Type 204B Series Magnetic Tape Unit</u> (Order No. BC38), <u>Type 204D-1, -3, -5 Magnetic Tape Units and Controls</u> (Order No. BK02), <u>Type 204D-1A, -3A, -5A Magnetic Tape Units</u> (Order No. AJ22), <u>Type 204F-1, -3, -5 Magnetic Tape Units</u> (Order No. AJ23).							

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4** ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
RANDOM ACCESS DRUM	SEARCH AND READ	X X	X <sup>1</sup> X	See Table 8-31	$\overbrace{0 T \quad \quad \quad T T}^{9\text{-bit track address numbered } 0 - 777 \text{ (octal)}}$		SS Sector address numbered 0 - 47 (octal)
	READ	X X	X <sup>1</sup> X	See Table 8-31	none	none	none
	SEARCH AND WRITE	X X	X <sup>2</sup> X	See Table 8-31	$\overbrace{0 T \quad \quad \quad T T}^{9\text{-bit track address numbered } 0 - 777 \text{ (octal)}}$		SS Sector address numbered 0 - 47 (octal)
	WRITE	X X	X <sup>2</sup> X	See Table 8-31	none	none	none
	READ ADDRESS REGISTER	X X	X <sup>1</sup> X	See Table 8-31	none	none	none
See: <u>Type 270A Random Access Drum and Control</u> (Order No. BA06)							
DISK DEVICES <sup>8</sup>	LOAD ADDRESS REGISTER	X X	X <sup>2</sup> X	0 4	none	none	none
	STORE ADDRESS REGISTER	X X	X <sup>1</sup> X	0 4	none	none	none
	WRITE INITIAL	X X	X <sup>2</sup> X	0 0 or 1 0*	none	none	none
	EXTENDED WRITE INITIAL	X X	X <sup>2</sup> X	2 0 or 3 0*	none	none	none
	WRITE	X X	X <sup>2</sup> X	0 1 or 1 1*	none	none	none
	EXTENDED WRITE	X X	X <sup>2</sup> X	2 1 or 3 1*	none	none	none
	SEARCH AND WRITE	X X	X <sup>2</sup> X	0 2 or 1 2*	none	none	none
	EXTENDED SEARCH AND WRITE	X X	X <sup>2</sup> X	2 2 or 3 2*	none	none	none
	SEARCH AND WRITE NEXT	X X	X <sup>2</sup> X	0 3 or 1 3*	none	none	none
	EXTENDED SEARCH AND WRITE NEXT	X X	X <sup>2</sup> X	2 3 or 3 3*	none	none	none
	SEARCH AND READ	X X	X <sup>1</sup> X	0 2 or 1 2*	none	none	none
	EXTENDED SEARCH AND READ	X X	X <sup>1</sup> X	2 2 or 3 2*	none	none	none
	SEARCH AND READ NEXT	X X	X <sup>1</sup> X	0 3 or 1 3*	none	none	none
	EXTENDED SEARCH AND READ NEXT	X X	X <sup>1</sup> X	2 3 or 3 3*	none	none	none
	READ INITIAL	X X	X <sup>1</sup> X	0 0 or 1 0*	none	none	none
	EXTENDED READ INITIAL	X X	X <sup>1</sup> X	2 0 or 3 0*	none	none	none
READ	X X	X <sup>1</sup> X	0 1 or 1 1*	none	none	none	
EXTENDED READ	X X	X <sup>1</sup> X	2 1 or 3 1*	none	none	none	
<p>* Reading/writing is verified.</p> <p>** C4 is used if C3 high-order bit is 1, specifying an eight-bit transfer.</p> <p>See: <u>Direct-Access Devices and Controls</u> (Order No. BC45)</p>							

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
TYPE 220-3, -6 CONSOLE	READ (NO CARRIAGE RETURN)	X X	X <sup>1</sup> X	0 0	none	none	none
	READ (CARRIAGE RETURN)	X X	X <sup>1</sup> X	0 1	none	none	none
	WRITE (NO CARRIAGE RETURN)	X X	X <sup>2</sup> X	0 0	none	none	none
	WRITE (CARRIAGE RETURN)	X X	X <sup>2</sup> X	0 1	none	none	none
See: Control Panels and Consoles (Models 200 through 4200), (Order No. BC05)							
TYPE 220-8 VICC CONSOLE	READ (no cursor/ carriage return)	X X	X <sup>1</sup> X	0 0	none	none	none
	READ (cursor/carriage return)	X X	X <sup>1</sup> X	0 1	none	none	none
	WRITE (no cursor/ carriage return)	X X	X <sup>2</sup> X	0 0	none	none	none
	WRITE (cursor/carriage return)	X X	X <sup>2</sup> X	0 1	none	none	none
	Cursor/carriage return and line feed (after display)	X X	X X	4 1	X X	X X	X X
	Line erase before display	X X	X X	4 2	X X	X X	X X
	Line erase before display, cursor return and line feed after display	X X	X X	4 3	X X	X X	X X
	Form feed - clear screen and page return before display. (Printer - carriage returns and line feed)	X X	X X	4 4	X X	X X	X X
	Page return - position cur- sor to home (line 1, character position 1) before display. (Printer - carriage return and line feed)	X X	X X	7 0	X X	X X	X X
	Select printer only y = 0: 64 columns y = 4: 80 columns	X X	X X	X X	y 2	X X	X X
	Select interactive display only y = 0: 64 columns y = 4: 80 columns	X X	X X	X X	y 4	X X	X X
	Select status display only y = 0: 64 columns y = 4: 80 columns	X X	X X	X X	y 5	X X	X X
	Select both interactive display and printer	X X	X X	X X	y 0	X X	X X
	Select status display and printer	X X	X X	X X	y 1	X X	X X
	Go to line address given by C5 variant 5 low- order bits with value 01 <sub>g</sub> -30 <sub>g</sub> (line 1 thru line 24) (Printer - carriage return, line feed)	X X	X X	X X	X X	01-30	X X
Blink first word of line containing cursor	X X	X X	X X	X X	4 0	X X	
Go to line address given by C5 variant 5 low- order bits and blink first word of that line	X X	X X	X X	X X	41-77	X X	
See: Type 220-8 Visual Information Control Console (Order No. AJ77)							

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
ON-LINE ADAPTER	TRANSFER ID character to Series 200 memory.	X X	X X	4 X (X=unused)	none	none	none
	ACCEPT the H-800/1800 instruction defined in the ID register. <sup>7</sup>	X X	X X	0 0	none	none	none
	ACCEPT the H-800/1800 instruction defined in the ID register, and cause the H-800/1800 to branch to U+3 or U+5. <sup>7</sup>	X X	X X	0 4	none	none	none
	DO NOT ACCEPT the H-800/1800 instruction defined in the ID register; rather, cause the H-800/1800 program to branch to U+6 or U+7 (read or write error). <sup>7</sup>	X X	X X	1 U (U = any value from 1-7, octal)	none	none	none
	SET the device busy indicator. <sup>7</sup>	X X	X X	3 X (X=unused)	none	none	none
See: Model 212 On-Line Adapter (Order No. BM06)							
TYPE 281 SCCC	RECEIVE	X X	X <sup>1</sup> X	none	none	none	none
	TRANSMIT	X X	X <sup>2</sup> X	none	none	none	none
TIME-OF-DAY CLOCK	TRANSFER TIME TO MEMORY	X X	X X	none	none	none	none
CENTRAL PROC. ERROR ADAPTER	TRANSFER DATA	X X	X X	none	none	none	none
See: Type 212-1 Central Processor Adapter (Order No. BB31)							
232 and 233-2 MICR READER- SORTERS	TRANSFER DATA	X X	X X	none	none	none	none
See: Type 233-2 MICR Control (Order No. BC11)							
OPTICAL DOCU- MENT READER AND CONTROL	READ	X X	X X	See Figure 8-14	none	none	none
See: Type 243 Optical Document Reader (Order No. BJ18)							
<p>NOTES:</p> <ol style="list-style-type: none"> <li>1. The high-order bit must be 1.</li> <li>2. The high-order bit must be 0.</li> <li>3. Odd parity is assumed. If even parity is required, the first octal character should be 7.</li> <li>4. Odd parity is assumed. If even parity is required, the first octal character should be 3.</li> <li>5. Odd parity and short gap are assumed. The first octal character should be 3 for even parity, short gap; 6 for odd parity, long gap; 7 for even parity, long gap.</li> <li>6. D (tape drive) = 0-3 when the instruction is issued to the Type 203B-5 Tape Control. D = 0 or 1 when the instruction is issued to the Type 203C-7 Tape Control.</li> <li>7. This operation issues initiating and concluding device-ready responses.</li> <li>8. Summary applies to all disk devices except 277/279. For information on these devices see: <u>Direct-Access Devices and Controls</u> (Order No. BC45).</li> </ol>							

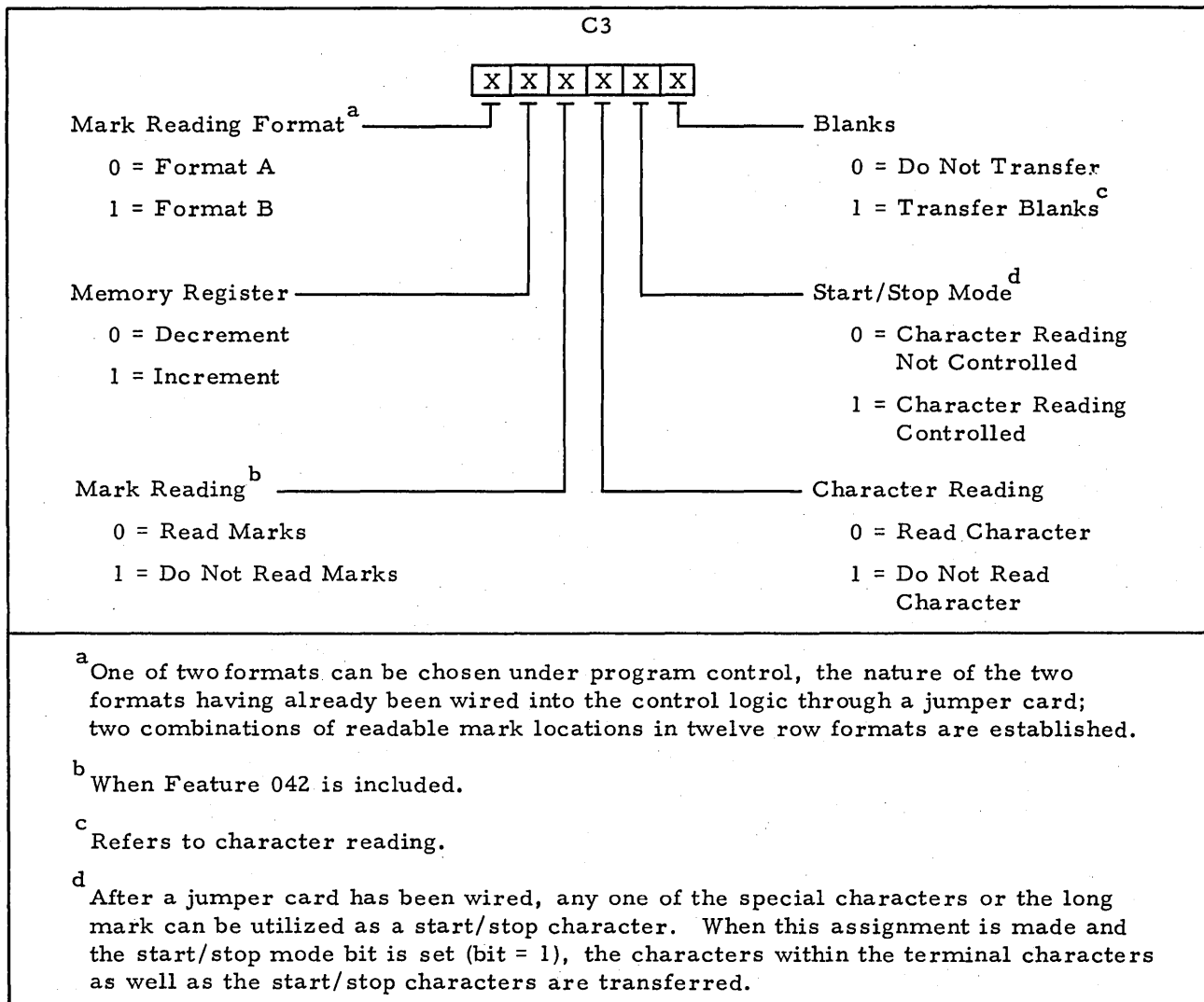


Figure 8-13. Format of Type 243 PDT C3 Variant

Table 8-28. C3 Coding for Type 209 and 209-2 Paper Tape Readers

VALUE	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
1	Not used	One character per frame	Sense end of record	Check odd parity	Read Forward	Increment CLC
0	Not used	Two characters per frame	Do not sense end of record	Check even parity	Read Reverse (Feature 010 or 011)	Decrement CLC

Table 8-29. C3 Coding for Type 210 Paper Tape Punch

VALUE	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
1	Not used	One character per frame	Not used	Compute odd parity	00 = Do not punch parity 01 = Parity bit in channel six 10 = Parity bit in channel seven 11 = Parity bit in channel eight	
0	Not used	Two characters per frame	Not used	Compute even parity		

Table 8-30. C3 Coding for Type 222 Printers

Type 222 Printers <sup>1</sup>	
C3	INTERPRETATION
00nnnn	Print, then space the number of lines specified by nnnn (0 - 15).
01nnnn	Print, then space to channel one of the format tape (HOF) if channel two of the format tape (EOF) is sensed; otherwise, space the number of lines specified by nnnn (0 - 15).
11nnnn	Do not print; space the number of lines specified by nnnn (0 - 15).
100xxx 101xxx	Print, then space to channel xxx. Do not print; space to channel xxx.
000 001 010 011 100 101 110 111	Channel 3 Channel 4 Channel 5 Channel 1 (Head of form) Channel 6 Channel 7 Channel 8 Channel 1 (Head of form)
<sup>1</sup> Control characters are the same with or without the presence of the Print Buffer (Feature 036) in the printer.	

Table 8-31. C3 Coding for Type 270A Random Access Drum

VALUE	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
1	Override	Increment drum address register	This is a Read Address Register instruction	Drum file designation 0 - 7 (octal)		
0	Do not override	Do not increment drum address register	This is not a Read Address Register instruction			

Table 8-32. Summary of PDT I/O Control Characters for Type 286 Multiline Communication Controller

INPUT/OUTPUT OPERATION		A ADDRESS	PDT I/O CONTROL CHAR.		
			C1	C2	C3
TYPE 286-1, -2, -3 MLCC	<u>FIRST DATA TRANSMISSION</u> PDT	LOC (specifies "line 0" in 286)	X X	X <sup>1</sup> X	none
	<u>RECEIVE DATA</u> PDT	LOC+2 (specifies line ad- dress in 286)	X X	X <sup>1</sup> X	none
	<u>TRANSMIT DATA</u> PDT	LOC+2 (specifies line ad- dress in 286)	X X	X <sup>2</sup> X	none
	<u>LINE CONTROL</u> PDT	LOC (specifies address of line to be controlled)  NOTE: The line con- trol transmission PDT instructions are listed in Table 8-33, below.	X X	X <sup>2</sup> X	none
TYPES 286-4, -5, -6, -7 MESSAGE-MODE MLCC	<u>TRANSMIT</u> (Load/test state only)	Leftmost character of field from which data is transferred.	X X	X <sup>2</sup> X	Section address or line number, 00 <sub>g</sub> - 63 <sub>g</sub> .
	<u>RECEIVE</u> (Load/test state only)	Leftmost character of field to which data is transferred.	X X	X <sup>1</sup> X	Section address or line number, 00 <sub>g</sub> - 63 <sub>g</sub> .
	<u>ASSIGN RWC AND LOAD SLC</u> (Initialized or off-line state only)	Leftmost character of 5-character status field storing interrupt information.	X X	X X	none
NOTES: 1. The high-order bit must be 1. 2. The high-order bit must be 0.					

Table 8-33. Type 286-1, -2, -3 Line Control Instructions

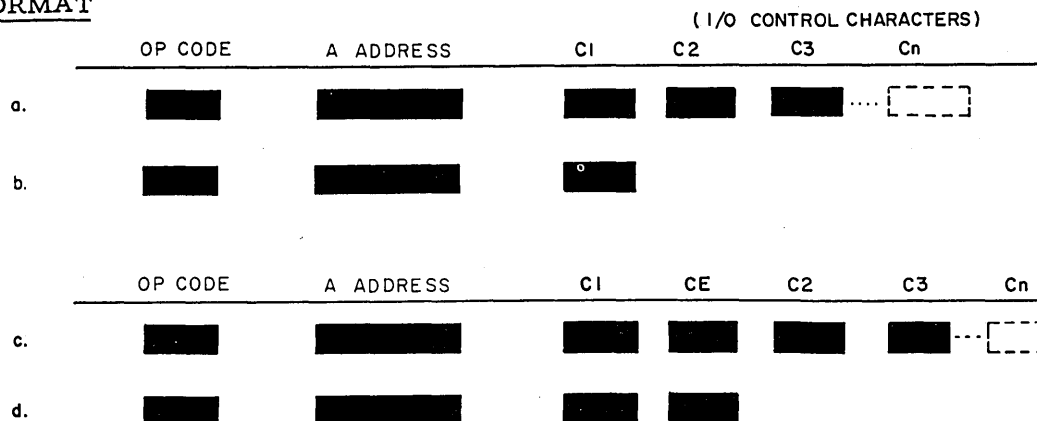
CODE <sup>1</sup> (OCTAL)	INSTRUCTION	DESCRIPTION
10	Transmit last character	Inform the 286 that the last character has been sent from the central processor, and place the control unit in the receive mode for that line (after transmitting last char- acter).
60	Receive clear	Reset the bits of the logic character in the 286 memory. (This instruction should be given when power is first turned on.)

Table 8-33 (cont). Type 286-1, -2, -3 Line Control Instructions

CODE <sup>1</sup> (OCTAL)	INSTRUCTION	DESCRIPTION
30	Inhibit 285 (service request)	Turn off the interrupt capability of a line that is requesting service (either input or output).
50	Transmit idle character	Repeat the previously provided character indefinitely, without interrupts.
40	Transmit	Stop the line from repeating character and cause an interrupt.
74	Move Longitudinal Redundancy Check (LRC) Character	Move the LRC character from the LRC register to the data buffer register (Feature 087).
34	Special Strobe	Activate the special strobe line to a Type 285 adapter via the Type 286 control.
NOTE: The control code is stored in location LOC+1. (The low-order two bits of this code must be 0.)		

PCB	PERIPHERAL CONTROL AND BRANCH	64 <sub>8</sub>
-----	-------------------------------	-----------------

FORMAT





## TYPES OF TEST AND CONTROL OPERATIONS

The Peripheral Control and Branch instruction can initiate four types of operations: (1) strictly mechanical peripheral device operations; (2) test and branch operations; (3) mode change operations; and (4) peripheral interrupt operations.

1. A mechanical operation is a non-data transfer operation such as rewind magnetic tape or seek a disk pack drive cylinder.
2. A test and branch operation tests the status of a peripheral control and/ or a read/write channel(s).<sup>1</sup> If the condition being tested (e. g., peripheral control busy, error in last card punched) is present, a program branch is performed.
3. A mode change operation conditions the addressed peripheral control to operate in a specific mode. For instance, the card reader control can be conditioned to reject illegally punched cards, to generate a busy signal if illegally punched cards are read, or both, depending upon the control characters of the PCB instruction.
4. A peripheral interrupt operation directs a peripheral control to change the setting of an interrupt function or an allow interrupt function (see Appendix D).

Control character C1 designates a read/write channel or combination of channels whose busy status is to be tested. If an RWC busy test is not desired, C1 must contain 0's. C2 designates the logical address of the peripheral control to be tested or actuated. The coding of this character is the same as its coding for a PDT instruction (see Table 8-26).

Control characters C3 through Cn designate the control and test operations. Any number of control characters may follow C2, each one designating a different operation. If control characters within a single instruction designate conflicting operations (e. g., punch Hollerith code and punch direct transcription mode), the control character to the left is cancelled by a conflicting control character to the right within the same instruction. If multiple test operations are specified within a single instruction, a branch will occur if any of the conditions tested is present. The specific use of characters C3 through Cn is dependent upon the type of peripheral device addressed. Tables 8-34 through 8-36 summarize the coding of these characters.

## FUNCTION

Format a: The read/write channel or channel combination specified by C1 is tested for busy status.<sup>2</sup> If it is busy, a branch is made to the instruction at A. If the RWC is not busy (or if C1 is 00g), the operation(s) specified by characters C3 through Cn is performed on the peripheral control specified by C2. This peripheral control must be connected to the input/output sector implied by the value of C1 (or, in a multicharacter processor, by the sector bits of C2).

---

<sup>1</sup> On multicharacter processors, time slots are also tested.

<sup>2</sup> On multicharacter processors, the sector designated by C2 is also interrogated to determine whether it has currently available a sufficient number of unassigned time slots to support the transfer rate implied by C1.

Format b: The read/write channel or channel combination specified by C1 is tested for busy status.<sup>1</sup> If it is busy, a branch is made to A. If the RWC is not busy, the instruction following the PCB is executed. Note that, in a multicharacter processor, PCB instruction in this format does not guarantee that an I/O transfer can be accomplished: the sector to be used must also be tested (i. e., format a, c, or d must be used).

Format c: The read/write channel or channel combination specified by C1 is tested for busy status. Also, the sector designated by CE is interrogated to determine whether it has currently available a sufficient number of unassigned time slots to support the I/O data transfer rate implied by C1. If the specified RWC is not busy and the designated sector can handle the data rate implied by C1, the operation(s) specified by characters C3 through Cn is performed on the peripheral control specified by C2, and the program continues in normal sequence. Otherwise, a branch is made to the instruction at A. The CE character must designate the I/O sector to which the peripheral control specified by C2 is connected.

Format d: The read/write channel or channel combination specified by C1 is tested for busy status. Also, the sector designated by CE is interrogated to determine whether or not it has currently available a sufficient number of unassigned time slots to support the I/O data transfer rate implied by C1 (see Table 8-24). If the specified RWC(s) is not busy and the designated sector can handle the data rate implied by C1, the program continues in sequence. Otherwise, a branch is made to the instruction at A.

#### PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or record marks.

#### ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR	
NXT	A	B P	NO BRANCH
JI (A)	A	NXT	BRANCH

#### NOTES

1. Formats c. and d. are applicable only to the multicharacter processors. In order to produce a meaningful result, when checking for termination of transfer, C1 must specify the proper RWC and test for device busy.
2. The PCB op code is a "privileged" op code that has special significance when Storage Protection is in effect (see Section II).
3. Control character C1 of a PCB instruction is stored in the variant register.

<sup>1</sup> On multicharacter processors, the "home" sector of the read/write channel or channel combination (see Table 8-24) is interrogated to determine whether it has currently available a sufficient number of unassigned time slots to support the transfer rate implied by C1.

4. When buffered sectors are involved, additional programming considerations apply to the PCB instruction (see Section II).

EXAMPLE

In the following example, assume that the logical address of the card reader control is octal 41.

See the card reader control to read Hollerith code (C3 = 27) and to reject automatically all cards with hole-count errors (C4 = 21). If the device is inoperable, branch to the location tagged STOP. (Note that since an RWC is not to be tested, C1 must contain 0's.)

**EASYCODER**  
CODING FORM

PROBLEM \_\_\_\_\_ PROGRAMMER \_\_\_\_\_ DATE \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

CARD NUMBER	Y	X	R	LOCATION	OPERATION CODE	OPERANDS	
						14 15	20 21
1				PCB	STOP	00, 41, 27, 21	
2							
3							
4							
5							
6							
7							
8							

Table 8-34. Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS			
		C1	C2	C3 through Cn	
TYPE 214-1 CARD PUNCH	Branch to A address if device busy	X X	X X	1 0	
	Branch to A address if punch-check error	X X	X X	4 1	
	Branch to A address if device unavailable. If available, set control unit to:	Punch Hollerith code <sup>4</sup>	X X	X X	2 7
		Punch special code	X X	X X	2 6
		Punch direct transcription code (feature 064)	X X	X X	2 5
		Generate busy signal if punch-check error	X X	X X	2 3
		Offset-stack cards with punch-check error	X X	X X	2 1
	Offset-stack the card currently at the punch station	X X	X X	3 1	

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS					
		C1	C2	C3 through Cn			
TYPE 214-1 CARD PUNCH	Turn the control allow function OFF	X X	X X	7 0			
	Turn the control allow function ON	X X	X X	7 1			
	Turn the control interrupt function OFF	X X	X X	7 4			
	Branch to A address if the control interrupt function is ON	X X	X X	7 5			
See: Type 214-1 Card Punch (Order No. BC03)							
TYPE 214-2 CARD READER/PUNCH	Branch to A address if device busy	X X	X <sup>3</sup> X	1 0			
	Branch to A address if cycle-check or punch-check error	X X	X <sup>3</sup> X	4 1			
	Branch to A address if illegal punch	X X	X <sup>3</sup> X	4 2			
	Branch to A address if device unavailable. If available, set control unit to:	X X	X <sup>3</sup> X	2 7			
	Terminate punch-feed read operations, operate in Hollerith mode, and accept all other cards <sup>4</sup>						
	Read or punch special code				X X	X <sup>3</sup> X	2 6
	Read or punch direct transcription code (Feature 064)				X X	X <sup>3</sup> X	2 5
	Generate busy signal if illegal punch				X X	X <sup>3</sup> X	2 4
	Generate busy signal if cycle-check or punch-check error				X X	X <sup>3</sup> X	2 3
	Offset-stack cards with illegal punches				X X	X <sup>3</sup> X	2 2
	Offset-stack cards with cycle-check or punch-check error				X X	X <sup>3</sup> X	2 1
	Operate in punch-feed read mode				X X	X <sup>3</sup> X	2 0
	Offset-stack the card currently at the punch station				X X	X <sup>3</sup> X	3 1
	Turn the control allow function OFF	X X	X <sup>3</sup> X	7 0			
Turn the control allow function ON	X X	X <sup>3</sup> X	7 1				
Turn the control interrupt function OFF	X X	X <sup>3</sup> X	7 4				
Branch to A address if the control interrupt function is ON	X X	X <sup>3</sup> X	7 5				
See: Type 214-2 Card Reader/Punch (Order No. BC04)							

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS				
		C1	C2	C3 through Cn		
TYPE 223, 223-2 CARD READER	Branch to A address if device busy		X X	X X	1 0	
	Branch to A address if cycle-check error		X X	X X	4 1	
	Branch to A address if illegal punch		X X	X X	4 2	
	Branch to A address if device unavailable. If available, set control unit to:	Read Hollerith code and accept all error cards <sup>4</sup>		X X	X X	2 7
		Read special code		X X	X X	2 6
		Read direct transcription code (Feature 044)		X	X X	2 5
		Offset-stack cards with cycle-check error		X X	X X	2 1
		Offset-stack cards with illegal punches		X X	X X	2 2
		Generate busy signal if cycle-check error		X X	X X	2 3
		Generate busy signal if illegal punch		X X	X X	2 4
		Offset-stack previously read card. Branch to A address if instruction issued too late (>30 milliseconds).		X X	X X	3 1
		Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1	
	Turn the control interrupt function OFF		X X	X X	7 4	
	Branch to A address if the control interrupt function is ON		X X	X X	7 5	
See: <u>Type 223, 223-2 Card Reader (Order No. BC39)</u>						
TYPE 209 and 209-2 PAPER TAPE READERS	Branch to A address if device busy		X X	X X	1 0	
	Branch to A address if parity error		X X	X X	4 0	
	Branch to A address if device unavailable. If available, set control unit to:	Rewind the tape (reverse direction)		X X	X X	3 0
		Run out the tape (forward direction)		X X	X X	3 2
	Turn the control allow function OFF		X X	X X	7 0	
	Turn the control allow function ON		X X	X X	7 1	
	Turn the control interrupt function OFF		X X	X X	7 4	
	Branch to A address if the control interrupt function is ON		X X	X X	7 5	
See: <u>Types 209, 209-2, and 210 Paper Tape Equipment (Order No. BC42)</u>						

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 210 PAPER TAPE PUNCH	Branch to A address if device busy	X X	X X	1 0
	Branch to A address if tape-low condition is true	X X	X X	6 0
	Turn the control allow function OFF	X X	X X	7 0
	Turn the control allow function ON	X X	X X	7 1
	Turn the control interrupt function OFF	X X	X X	7 4
	Branch to A address if the control interrupt function is ON	X X	X X	7 5
See: <u>Types 209, 209-2, and 210 Paper Tape Equipment (Order No. BC42)</u>				
TYPE 222 PRINTERS	Branch to A address if device busy	X X	X X	1 0
	Branch to A address if print error	X X	X X	4 0
	Branch to A address if paper is moving	X X	X X	2 0
	Branch to A address if busy or paper is moving	X X	X X	3 0
	Branch to A address if end of form	X X	X X	0 1
	Branch to A address if channel eight	X X	X X	0 2
	Turn the control allow function OFF	X X	X X	7 0
	Turn the control allow function ON	X X	X X	7 1
	Turn the control interrupt function OFF	X X	X X	7 4
	Branch to A address if the control interrupt function is ON	X X	X X	7 5
See: <u>Type 222-3, -4, -5, -6 Printers (Order No. BC75)</u> , or <u>Type 222-7 Printer and Control (Order No. BJ23)</u> .				
NOTE: Control characters are the same with or without the presence of the Print Buffer (Feature 036) in the printer.				
MAGNETIC TAPE UNITS 1/2-INCH	Rewind	X X	X <sup>2</sup> X	2 D (D=tape drive, 0 - 7) <sup>5</sup>
	Rewind and release	X X	X <sup>1</sup> X	2 D (D=tape drive, 0 - 7) <sup>5</sup>
	Branch to A address if read busy	X X	X <sup>1</sup> X	0 D (D=tape drive, 0 - 7) <sup>5</sup>
	Branch to A address if write busy	X X	X <sup>2</sup> X	0 D (D=tape drive, 0 - 7) <sup>5</sup>
	Branch to A address if read/write error	X X	X X	4 D (D=tape drive, 0 - 7) <sup>5</sup>

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
MAGNETIC TAPE UNITS 1/2-INCH (cont)	Branch to A address if beginning of tape	X X	X <sup>1</sup> X	6 D (D=tape drive, 0 - 7) <sup>5</sup>
	Branch to A address if end of tape	X X	X <sup>2</sup> X	6 D (D=tape drive, 0 - 7) <sup>5</sup>
	Turn the control allow function OFF	X X	X <sup>3</sup> X	7 0
	Turn the control allow function ON	X X	X <sup>3</sup> X	7 1
	Turn the control interrupt function OFF	X X	X <sup>3</sup> X	7 4
	Branch to A address if the control interrupt function is ON	X X	X <sup>3</sup> X	7 5
See: <u>Type 204B Series Magnetic Tape Units (Order No. BC38), Type 204D-1, -3, -5 Magnetic Tape Units (Order No. BK02), Type 204D-1A, -3A, -5A Magnetic Tape Units (Order No. AJ22), Type 204F-1, -3, -5 Magnetic Tape Units (Order No. AJ23).</u>				
TYPE 270A RANDOM ACCESS DRUM	Branch to A address if control unit busy <sup>6</sup>	X X	X X	0 X or 1 X (X=unused)
	Branch to A address if error indicator is ON	X X	X X	4 X (X=unused)
	Turn the control allow function OFF	X X	X X	7 0
	Turn the control allow function ON	X X	X X	7 1
	Turn the control interrupt function OFF	X X	X X	7 4
	Branch to A address if the control interrupt function is ON	X X	X X	7 5
See: <u>Type 270A Random Access Drum and Control (Order No. BA06)</u>				
TYPE 220-3 AND -6 CONSOLES	Branch to A address if device busy	X X	X <sup>2</sup> X	1 0
	Turn the allow function OFF	X X	X <sup>2</sup> X	7 0
	Turn the allow function ON	X X	X <sup>2</sup> X	7 1
	Turn the data termination interrupt function OFF	X X	X <sup>2</sup> X	7 4
	Branch to A address if data termination interrupt function is ON	X X	X <sup>2</sup> X	7 5
	Turn the interrupt function OFF	X X	X <sup>2</sup> X	7 6
	Branch to A address if interrupt function is ON	X X	X <sup>2</sup> X	7 7
See: <u>Control Panels and Consoles (Models 200 through 4200), (Order No. BC05)</u>				
TYPE 220-8 VICC CONSOLE	Branch to A address if device busy	X X	X <sup>2</sup> X	1 0
	Branch to A address if interactive display not available	X X	X <sup>2</sup> X	1 5
	Branch to A address if status display not available	X X	X <sup>2</sup> X	1 6

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS			
		C1	C2	C3 through Cn	
TYPE 220-8 VICC CONSOLE	Branch to A address if printer not available	X X	X <sup>2</sup> X	1 7	
	Branch to A address if console error detected	X X	X <sup>2</sup> X	3 0	
	Branch to A address if illegal line address	X X	X <sup>2</sup> X	3 1	
	Branch to A address if parity error on cursor command	X X	X <sup>2</sup> X	3 2	
	Place the console control in a not busy state	X X	X <sup>2</sup> X	3 4	
	Turn the allow function OFF	X X	X <sup>2</sup> X	7 0	
	Turn the allow function ON	X X	X <sup>2</sup> X	7 1	
	Turn the data termination interrupt junction OFF	X X	X <sup>2</sup> X	7 4	
	Branch to A address if data termination interrupt function is ON				
See: <u>Type 220-8 Visual Information Control Console</u> (Order No. AJ77)					
TYPE 212 ON-LINE ADAPTER	Branch to A address if device busy	X X	X <sup>3</sup> X	0 X or 1 X (X=unused)	
	Branch to A address if data transfer is in progress	X X	X <sup>3</sup> X	7 X (X=unused)	
	Branch to A address if error or incomplete indicator is set	X X	X <sup>3</sup> X	4 X	
	Branch to A address if parity error is stored	X X	X <sup>3</sup> X	5 X (X=unused)	
	Branch to A address if incomplete error is stored	X X	X <sup>3</sup> X	6 X (X=unused)	
	Place control character C4 in the ID register if data transfer is not in progress	X X	X <sup>3</sup> X	C3: 2 X (X=unused)  C4: octal character to be placed in ID register	
	Branch to A address unconditionally, and clear the ID register	X X	X <sup>3</sup> X	3 X (X=unused)	
See: <u>Model 212 On-Line Adapter</u> ; (Order No. BM06)					
DISK DEVICES <sup>10</sup>	Branch to A address if specified device is busy; otherwise, set control unit to:	Seek out the cylinder (specified by C5 and C6) in the pack (specified by C4).	X X	X <sup>2</sup> X	C3: 2 D (D=device address, 0 - 7)  C4: 00  C5 and C6: 0000 to 0143 for the Type 258, 0000 to 0312 for the Type 259, 0000 to 0177 for Types 261 and 262.
	Restore the specified device to cylinder zero.		X X	X <sup>1</sup> X	3 D (D=device address, 0-7)



Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
10 DISK DEVICES (cont)	Branch to A address if <u>control busy</u> .	X X	X <sup>2</sup> X	1 0
	Branch to A address if <u>device busy</u> .	X X	X <sup>2</sup> X	C3: 0 D C4: 00 (D=device address, 0-7)
	Branch to A address if a <u>general exception</u> condition occurred during the preceding PDT instruction.	X X	X <sup>2</sup> X	5 0
	Branch to A address if the <u>TLR</u> flag is set.	X X	X <sup>2</sup> X	6 0
	Set control unit to <u>override</u> setting of <u>FORMAT WRITE PERMIT</u> switch.	X X	X <sup>2</sup> X	4 0
	Turn control allow function OFF.	X X	X <sup>2</sup> X	7 0
	Turn control allow function ON.	X X	X <sup>2</sup> X	7 1
	Turn drive allow function OFF.	X X	X <sup>2</sup> X	7 2
	Turn drive allow function ON.	X X	X <sup>2</sup> X	7 3
	Turn control interrupt function OFF. <sup>8</sup>	X X	X <sup>2</sup> X	7 4
	Branch to A address if control interrupt function is ON.	X X	X <sup>2</sup> X	7 5
	Turn drive interrupt function OFF.	X X	X <sup>2</sup> X	7 6
	Branch to A address if device interrupt function is ON.	X X	X <sup>2</sup> X	7 7
See: <u>Direct-Access Devices and Controls (Order No. BC45)</u>				
TYPE 281 SINGLE-LINE COMMUNICATION CONTROLLER	Branch to A address if device busy	X X	X <sup>3</sup> X	1 0
	Branch to A address if parity error	X X	X <sup>3</sup> X	4 0
	Branch to A address if error other than parity error	X X	X <sup>3</sup> X	5 0
	Branch to A address if the 281 is in transmit mode and requesting data for transmission	X X	X <sup>3</sup> X	6 0
	Branch to A address if the 281 is in receive mode and requesting that central processor take received data	X X	X <sup>3</sup> X	6 1
	Turn the allow function OFF	X X	X <sup>7</sup> X	7 0
	Turn the allow function ON	X X	X <sup>7</sup> X	7 1
	Turn the interrupt function OFF	X X	X <sup>7</sup> X	7 4
	Branch to A address if allow and interrupt functions are ON	X X	X <sup>7</sup> X	7 5

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 213-3 INTERVAL TIMER	Branch to A address if device busy (Feature 071)	0 0	X <sup>7</sup> X	1 0
	Turn the allow function OFF	0 0	X <sup>7</sup> X	7 0
	Turn the allow function ON	0 0	X <sup>7</sup> X	7 1
	Turn the allow function ON (Feature 071)	0 0	X <sup>7</sup> X	7 3 (C4 - C6 specify time interval)
	Turn the interrupt function OFF	0 0	X <sup>7</sup> X	7 4
	Branch to A address if interrupt function is ON	0 0	X <sup>7</sup> X	7 5
	Turn the interrupt function OFF (Feature 071)	0 0	X <sup>7</sup> X	7 6
	Branch to A address if interrupt function is ON (Feature 071)	0 0	X <sup>7</sup> X	7 7
See: <u>Type 213-3 Interval Timer and Feature 071 Interval Selector</u> (Order No. BA49)				
TYPE 213-4 TIME OF DAY CLOCK	Branch to A address if device busy	X X	X X	1 0
TYPE 212-1 CENTRAL PROCESSOR ADAPTER	Branch to A address if device busy	X X	X <sup>7</sup> X	0 X or 1 0
	Branch to A address if device busy, and reserve	X X	X <sup>7</sup> X	C3: 2 0 C4: 0 0
	Branch to A address if reserve action by this central processor was not successful	X X	X <sup>7</sup> X	C3: 2 0 C4: 0 0 C5: 6 1
	Branch to A address if 212-1 is <u>not</u> set for data transfer (initiator)	X X	X <sup>7</sup> X	6 1
	Branch to A address if 212-1 is set for data transfer (responder)	X X	X <sup>7</sup> X	6 4
	Turn the allow function OFF	X X	X <sup>7</sup> X	7 0
	Turn the allow function ON	X X	X <sup>7</sup> X	7 1
	Turn the interrupt function OFF	X X	X <sup>7</sup> X	7 4
Branch to A address if allow and interrupt functions are ON	X X	X <sup>7</sup> X	7 5	
See: <u>Type 212-1 Central Processor Adapter</u> (Order No. BB31)				

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
Branch to A address if control busy		X X	X X	1 0
Select stacker designated; Branch to A address if:  1. the reader-sorter is not ready; or  2. the 10-millisecond stacker selection period has elapsed; or  3. The leading edge of the document to be sorted has not passed the reading station; or  4. the leading edge has passed the reading station and a PDT instruction has not yet been issued; or  5. the reader-sorter is performing an automatic reject on the document in question	Stacker 0	X X	X X	2 0
	Stacker 1	X X	X X	2 1
	Stacker 2	X X	X X	2 2
	Stacker 3	X X	X X	2 3
	Stacker 4	X X	X X	2 4
	Stacker 5	X X	X X	2 5
	Stacker 6	X X	X X	2 6
	Stacker 7	X X	X X	2 7
	Stacker 8	X X	X X	3 0
	Stacker 9	X X	X X	3 1
	Stacker X	X X	X X	3 2
Stacker Y	X X	X X	3 3	
Reject Stacker	X X	X X	3 7	
Start feed. Branch to A address if feed cannot be started due to:  1. the reader-sorter not being ready; or  2. proper restart procedures not followed		X X	X X	3 4
Stop feed. Branch to A address if sorter-reader is not ready		X X	X X	3 5
Set pocket-light control. Branch to A address if:  1. the reader-sorter is not ready; or  2. a pocket-light control PCB is already in process		X X	X X	3 6
Branch to A address if:	Amount field error	X X	X X	4 0
	Process control field error	X X	X X	4 1
	Account field error	X X	X X	4 2
	Transit field error	X X	X X	4 3
	Auxiliary on-us field error	X X	X X	4 4
	Device error	X X	X X	5 0
	Passed document condition	X X	X X	5 1
Operate in normal mode		X X	X X	6 0
Operate in short-document mode		X X	X X	6 1

TYPE 232 and 233-2 MICR READER-SORTERS

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
Type 232 and 233-2 MICR READER - SORTERS (cont)	Branch to A address if on-us field is complete	X X	X X	6 2
	Branch to A address if last document was a control document	X X	X X	6 3
	Branch to A address if end-of-file	X X	X X	6 4
	Advance batch counter one digit. Branch to A address if the sorter-reader is not stopped or the batch counter is currently being advanced	X X	X X	6 5
	Turn allow function OFF	X X	X X	7 0
	Turn allow function ON	X X	X X	7 1
	Turn interrupt function OFF	X X	X X	7 4
	Branch to A address if interrupt function is ON	X X	X X	7 5
See: <u>Type 233-2 MICR Control (Order No. BC11)</u>				
TYPE 243 OPTICAL DOCUMENT READER AND CONTROL	Device Busy Test	X X	X X	1 0
	Reject	X X	X X	2 0
	Accept A	X X	X X	2 1
	Accept B	X X	X X	2 2
	Alternate Accept	X X	X X	2 3
	Feed Stop	X X	X X	3 0
	Feed Start	X X	X X	3 1
	Feed Ready Test	X X	X X	3 2
	Device Error	X X	X X	3 3
	One Character Unreadable	X X	X X	3 4
	Multicharacter Unreadable	X X	X X	3 5
	Blank Document	X X	X X	3 6
	Double Feed Test	X X	X X	3 7
	Start Interrupt Test	X X	X X	4 0
	Interrupt #1	X X	X X	4 1
	Interrupt #2	X X	X X	4 2
	Unfinished Reading Interrupt Test	X X	X X	4 3
	Hopper Empty	X X	X X	4 7
Interrupt Allow Function Reset	X X	X X	7 0	
Interrupt Allow Function Set	X X	X X	7 1	

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 243 OPTICAL DOCUMENT READER AND CONTROL (cont)	Interrupt Function Reset	X X	X X	7 4
	Interrupt Function Set	X X	X X	7 5
	Unreadable Mark Detect <sup>9</sup>	X X	X X	4 4
	Blank Document A <sup>9</sup>	X X	X X	4 5
	Blank Document B <sup>9</sup>	X X	X X	4 6

NOTES:

1. The high-order bit must be 1.
2. The high-order bit must be 0.
3. The high-order bit is set to 1 for input operations and to 0 for output operations.
4. This control character should precede all other control characters that set the control to perform a certain action. It is the programmer's responsibility to set the control to the desired mode of operation at the beginning of the run.
5. D (tape drive) = 0-3 when the instruction is issued to the 203B-5 Tape Control. D = 0 or 1 when the instruction is issued to the 203C-7 Tape Control.
6. As the drum control does not permit reading from one drum while writing on another, it is considered busy if either a read or write operation is in progress. (The value of the high-order bit in C2 is thus immaterial in this case.)
7. The high-order bit is ignored.
8. The interrupt functions of both the control and the disk device are automatically turned on when a "not busy" status is reached by the control or the disk device, respectively.
9. Requires Feature 042.
10. Summary applies to all disk devices except 277 and 279. For information on these devices see Direct Access Devices and Controls (Order No. BC45).

Table 8-35. Summary of PCB I/O Control Characters for Type 286 Multiline Communication Controller

OPERATION		PCB I/O CONTROL CHARACTERS			
		C1	C2	C3	C4 through Cn
TYPE 286-1, -2, -3 MLCC	Branch to A address if device busy. If not busy, set the 286 to stop scanning and continue the program in sequence	X X	X X	1 0	none
	Turn the allow function OFF	X X	X X	7 0	none
	Turn the allow function ON	X X	X X	7 1	none
	Branch to A address if the interrupt was due to the 286 requesting service	X X	X X	7 5	none

Table 8-35 (cont). Summary of PCB I/O Control Characters for Type 286  
Multiline Communication Controller

OPERATION		PCB I/O CONTROL CHARACTERS			
		C1	C2	C3	C4 through Cn
TYPE 286-4, -5, -6, -7 MESSAGE-MODE MLCC	Branch to A address if device busy	X X	X X	1 0	none
	Branch to A address if parity error	X X	X X	4 0	none
	Branch to A address if the interrupt was due to the 286 requesting service	X X	X X	7 5	none
	Turn the allow function ON	X X	X X	7 1	none
	Turn the allow function OFF	X X	X X	7 0	none
	Set the 286 to the load/test state	X X	X X	2 5	none
	Provide line orientation for load/test operation	X X	X X	4 1	none
	Turn the load/test state and line orientation OFF	X X	X X	2 4	none
	Turn the interrupt function OFF	X X	X X	7 4	none
	Release the RWC(S) assigned to the 286	X X	X X	2 7	none
	Set the halt/continue indicator to halt	X X	X X	2 0	none
	Set the halt/continue indicator to continue	X X	X X	2 1	none
	Turn the parity error indicator and the parity error interrupt function OFF	X X	X X	2 6	none
	Request the address of the next transfer that is to take place from the line designated by C4, and branch to the A address	X X	X X	3 6	C4: 00 to 77
	Abort the present instruction to the line designated by C4, generate an interrupt, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 3	C4: 00 to 77
	Abort the present instruction to the line designated by C4, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 2	C4: 00 to 77
	Reset synchronization for the line designated by C4, and branch to the A address	X X	X X	3 7	C4: 00 to 77
	Activate the special strobe line to the 285 adapter designated by C4, and branch to the A address	X X	X X	3 4	C4: 00 to 77
	Deliver to the 286 the information specified by C5 et seq. for the next instruction to the line designated by C4, and branch to the A address	X X	X X	3 0	C4: 00 to 77 (See Table 8-36 for C5 et seq.)
	Deliver to the 286 the information specified by C5 et seq. for the next instruction to the line designated by C4; then abort the present instruction to that line, generate an interrupt, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 3	C4: 00 to 77 (See Table 8-36 for C5 et seq.)
Deliver to the 286 the information specified by C5 et seq. for the next instruction to the line designated by C4, then abort the present instruction to that line, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 2	C4: 00 to 77 (See Table 8-36 for C5 et seq.)	

Table 8-36. PCB Control Characters C5 through C15 for Type 286-4, -5, -6, -7  
Line Control Instructions

Control Character	Configuration (Octal)		Description																			
C5 C6 C7	<p style="text-align: center;"><u>C5</u> XX</p> <p style="text-align: center;">most significant six bits</p>	<p style="text-align: center;"><u>C6</u> XX</p> <p style="text-align: center;">middle six bits</p>	<p style="text-align: center;"><u>C7</u> XX</p> <p style="text-align: center;">least significant six bits</p>	Address to be loaded into RWC counters (SLC and CLC) prior to data transfer.																		
C8 C9	<p style="text-align: center;"><u>C8</u> XX</p> <p>Bits <u>6</u> and <u>5</u> specify mode of operation of the line:</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;"><u>6</u></td> <td style="text-align: center;"><u>5</u></td> <td></td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>- Inhibit</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>- Receive</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>- Transmit</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>- Transmit Repeat</td> </tr> </table>	<u>6</u>	<u>5</u>		0	0	- Inhibit	0	1	- Receive	1	0	- Transmit	1	1	- Transmit Repeat	<p style="text-align: center;"><u>C9</u> XX</p> <p>Bit <u>6</u> is the response bit</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;">0</td> <td>- no interrupt is allowed at termination of the instruction.</td> </tr> <tr> <td style="text-align: center;">1</td> <td>- an interrupt is allowed.</td> </tr> </table>	0	- no interrupt is allowed at termination of the instruction.	1	- an interrupt is allowed.	Control characters which specify line action; they are loaded into the next instruction section of memory.
<u>6</u>	<u>5</u>																					
0	0	- Inhibit																				
0	1	- Receive																				
1	0	- Transmit																				
1	1	- Transmit Repeat																				
0	- no interrupt is allowed at termination of the instruction.																					
1	- an interrupt is allowed.																					
<p>Bit <u>4</u> is the Allow Timer bit</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;">0</td> <td>- Timer is not allowed</td> </tr> <tr> <td style="text-align: center;">1</td> <td>- Timer is allowed</td> </tr> </table>	0	- Timer is not allowed	1	- Timer is allowed	<p>Bits <u>4</u> and <u>5</u> are not used and must be zero.</p>																	
0	- Timer is not allowed																					
1	- Timer is allowed																					
<p>Bits <u>3</u> and <u>2</u> specify character parity</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;"><u>3</u></td> <td style="text-align: center;"><u>2</u></td> <td></td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>no parity</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>generation or checking is performed</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>even parity</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>odd parity</td> </tr> </table>	<u>3</u>	<u>2</u>		0	0	no parity	0	1	generation or checking is performed	1	0	even parity	1	1	odd parity	<p>Bit <u>3</u> is the block parity bit</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;">0</td> <td>- block parity is not used</td> </tr> <tr> <td style="text-align: center;">1</td> <td>- block parity is used</td> </tr> </table>	0	- block parity is not used	1	- block parity is used		
<u>3</u>	<u>2</u>																					
0	0	no parity																				
0	1	generation or checking is performed																				
1	0	even parity																				
1	1	odd parity																				
0	- block parity is not used																					
1	- block parity is used																					
<p>Bit <u>1</u> is the character transfer bit</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;">0</td> <td>- one six-bit character transfer per line character</td> </tr> <tr> <td style="text-align: center;">1</td> <td>- two six-bit character transfers per line character</td> </tr> </table>	0	- one six-bit character transfer per line character	1	- two six-bit character transfers per line character	<p>Bit <u>2</u> is the command termination bit</p> <table style="margin-left: 40px;"> <tr> <td style="text-align: center;">0</td> <td>- character recognized is the last one transferred</td> </tr> <tr> <td style="text-align: center;">1</td> <td>- one more data transfer is made to or from the CP after the character recognized and before command termination.</td> </tr> </table>	0	- character recognized is the last one transferred	1	- one more data transfer is made to or from the CP after the character recognized and before command termination.													
0	- one six-bit character transfer per line character																					
1	- two six-bit character transfers per line character																					
0	- character recognized is the last one transferred																					
1	- one more data transfer is made to or from the CP after the character recognized and before command termination.																					

Table 8-36 (cont). PCB Control Characters C5 through C15 for Type 286-4, -5, -6, -7 Line Control Instructions

Control Character	Configuration (Octal)		Description
C8 C9 (cont)	Bit <u>1</u> defines block parity check bit  0 - check bit will be the half add sum of the parity bit of the preceding characters in the message.  1 - block parity character will have same parity generated or checked as the data characters.		
C10 C11	<u>C10</u> XX	<u>C11</u> XX	Eight bits (the low-order two bits of C10 and all six bits of C11) contain the first recognition character.
C12 C13	<u>C12</u> XX	<u>C13</u> XX	Eight bits (the low-order two bits of C12 and all six bits of C13) contain the second recognition character.
C14 C15	<u>C14</u> XX	<u>C15</u> XX	Eight bits (the low-order two bits of C14 and all six bits of C15) contain the SIT character for asynchronous lines.





APPENDIX A  
OCTAL NOTATION

Octal notation is a convenient shorthand method of writing pure binary numbers. In Series 2000 programming it is used to represent such binary values as main memory addresses, variant characters, I/O control characters, and constants.

If a binary value is divided into groups of three bits, proceeding from right to left, each group may be replaced by its octal equivalent as indicated in Table A-1.

Table A-1. Binary-Octal Equivalents

3-BIT BINARY GROUP	OCTAL EQUIVALENT
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Example 1.

The binary value

011111000101001110

when divided into three-bit groups

011 111 000 101 001 110

has an octal equivalent of

3 7 0 5 1 6

Example 2.

The binary value

1010100111010

when divided into three-bit groups

1 010 100 111 010

has an octal equivalent of

1 2 4 7 2

Table A-2. Decimal-Octal Conversion Table

LOW-ORDER OCTAL DIGIT	DECIMAL INCREMENT																												LOW-ORDER OCTAL DIGIT				
	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30	31	40	41		50	60	70	
0	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120	128	136	144	152	160	168	176	184	192	0	0	0	0	0	0	0	
1	001	009	017	025	033	041	049	057	065	073	081	089	097	105	113	121	129	137	145	153	161	169	177	185	193	1	1	1	1	1	1	1	
2	002	010	018	026	034	042	050	058	066	074	082	090	098	106	114	122	130	138	146	154	162	170	178	186	194	2	2	2	2	2	2	2	
3	003	011	019	027	035	043	051	059	067	075	083	091	099	107	115	123	131	139	147	155	163	171	179	187	195	3	3	3	3	3	3	3	
4	004	012	020	028	036	044	052	060	068	076	084	092	100	108	116	124	132	140	148	156	164	172	180	188	196	4	4	4	4	4	4	4	
5	005	013	021	029	037	045	053	061	069	077	085	093	101	109	117	125	133	141	149	157	165	173	181	189	197	5	5	5	5	5	5	5	
6	006	014	022	030	038	046	054	062	070	078	086	094	102	110	118	126	134	142	150	158	166	174	182	190	198	6	6	6	6	6	6	6	
7	007	015	023	031	039	047	055	063	071	079	087	095	103	111	119	127	135	143	151	159	167	175	183	191	199	7	7	7	7	7	7	7	
0000	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30	0000	0	0	0	0	0	0	0
0200	31	32	33	34	35	36	37	40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57	60	0200	0	0	0	0	0	0	0	
0400	62	63	64	65	66	67	70	71	72	73	74	75	76	77	100	101	102	103	104	105	106	107	110	111	0400	0	0	0	0	0	0	0	
0600	113	114	115	116	117	120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137	140	141	142	0600	0	0	0	0	0	0	0	
0800	144	145	146	147	150	151	152	153	154	155	156	157	160	161	162	163	164	165	166	167	170	171	172	173	0800	0	0	0	0	0	0	0	
1000	175	176	177	200	201	202	203	204	205	206	207	210	211	212	213	214	215	216	217	220	221	222	223	224	1000	0	0	0	0	0	0	0	
1200	226	227	230	231	232	233	234	235	236	237	240	241	242	243	244	245	246	247	250	251	252	253	254	255	1200	0	0	0	0	0	0	0	
1400	257	260	261	262	263	264	265	266	267	270	271	272	273	274	275	276	277	300	301	302	303	304	305	306	1400	0	0	0	0	0	0	0	
1600	310	311	312	313	314	315	316	317	320	321	322	323	324	325	326	327	330	331	332	333	334	335	336	337	1600	0	0	0	0	0	0	0	
1800	341	342	343	344	345	346	347	350	351	352	353	354	355	356	357	360	361	362	363	364	365	366	367	370	1800	0	0	0	0	0	0	0	
2000	372	373	374	375	376	377	400	401	402	403	404	405	406	407	410	411	412	413	414	415	416	417	420	421	2000	0	0	0	0	0	0	0	
2200	423	424	425	426	427	430	431	432	433	434	435	436	437	440	441	442	443	444	445	446	447	450	451	452	2200	0	0	0	0	0	0	0	
2400	454	455	456	457	460	461	462	463	464	465	466	467	470	471	472	473	474	475	476	477	500	501	502	503	2400	0	0	0	0	0	0	0	
2600	505	506	507	510	511	512	513	514	515	516	517	520	521	522	523	524	525	526	527	530	531	532	533	534	2600	0	0	0	0	0	0	0	
2800	536	537	540	541	542	543	544	545	546	547	550	551	552	553	554	555	556	557	560	561	562	563	564	565	2800	0	0	0	0	0	0	0	
3000	567	570	571	572	573	574	575	576	577	600	601	602	603	604	605	606	607	610	611	612	613	614	615	616	3000	0	0	0	0	0	0	0	
3200	620	621	622	623	624	625	626	627	630	631	632	633	634	635	636	637	640	641	642	643	644	645	646	647	3200	0	0	0	0	0	0	0	
3400	651	652	653	654	655	656	657	660	661	662	663	664	665	666	667	670	671	672	673	674	675	676	677	3400	0	0	0	0	0	0	0		
3600	702	703	704	705	706	707	710	711	712	713	714	715	716	717	720	721	722	723	724	725	726	727	730	3600	0	0	0	0	0	0	0		
3800	733	734	735	736	737	740	741	742	743	744	745	746	747	750	751	752	753	754	755	756	757	760	761	762	3800	0	0	0	0	0	0	0	
4000	764	765	766	767	770	771	772	773	774	775	776	777	1000	1001	1002	1003	1004	1005	1006	1007	1010	1011	1012	1013	4000	0	0	0	0	0	0	0	
4200	1015	1016	1017	1020	1021	1022	1023	1024	1025	1026	1027	1030	1031	1032	1033	1034	1035	1036	1037	1040	1041	1042	1043	1044	4200	0	0	0	0	0	0	0	
4400	1046	1047	1050	1051	1052	1053	1054	1055	1056	1057	1060	1061	1062	1063	1064	1065	1066	1067	1070	1071	1072	1073	1074	1075	4400	0	0	0	0	0	0	0	
4600	1077	1100	1101	1102	1103	1104	1105	1106	1107	1110	1111	1112	1113	1114	1115	1116	1117	1120	1121	1122	1123	1124	1125	1126	4600	0	0	0	0	0	0	0	
4800	1130	1131	1132	1133	1134	1135	1136	1137	1140	1141	1142	1143	1144	1145	1146	1147	1150	1151	1152	1153	1154	1155	1156	1157	4800	0	0	0	0	0	0	0	
5000	1161	1162	1163	1164	1165	1166	1167	1170	1171	1172	1173	1174	1175	1176	1177	1200	1201	1202	1203	1204	1205	1206	1207	1210	5000	0	0	0	0	0	0	0	
5200	1212	1213	1214	1215	1216	1217	1220	1221	1222	1223	1224	1225	1226	1227	1230	1231	1232	1233	1234	1235	1236	1237	1240	1241	5200	0	0	0	0	0	0	0	
5400	1243	1244	1245	1246	1247	1250	1251	1252	1253	1254	1255	1256	1257	1260	1261	1262	1263	1264	1265	1266	1267	1270	1271	1272	5400	0	0	0	0	0	0	0	
5600	1274	1275	1276	1277	1300	1301	1302	1303	1304	1305	1306	1307	1310	1311	1312	1313	1314	1315	1316	1317	1320	1321	1322	1323	5600	0	0	0	0	0	0	0	
5800	1325	1326	1327	1330	1331	1332	1333	1334	1335	1336	1337	1340	1341	1342	1343	1344	1345	1346	1347	1350	1351	1352	1353	1354	5800	0	0	0	0	0	0	0	
6000	1356	1357	1360	1361	1362	1363	1364	1365	1366	1367	1370	1371	1372	1373	1374	1375	1376	1377	1400	1401	1402	1403	1404	1405	6000	0	0	0	0	0	0	0	
6200	1407	1410	1411	1412	1413	1414	1415	1416	1417	1420	1421	1422	1423	1424	1425	1426	1427	1430	1431	1432	1433	1434	1435	1436	6200	0	0	0	0	0	0	0	
6400	1440	1441	1442	1443	1444	1445	1446	1447	1450	1451	1452	1453	1454	1455	1456	1457	1460	1461	1462	1463	1464	1465	1466	1467	6400	0	0	0	0	0	0	0	
6600	1471	1472	1473	1474	1475	1476	1477	1500	1501	1502	1503	1504	1505	1506	1507	1510	1511	1512	1513	1514	1515	1516	1517	1520	6600	0	0	0	0	0	0	0	
6800	1522	1523	1524	1525	1526	1527	1530	1531	1532	1533	1534	1535	1536	1537	1540	1541	1542	1543	1544	1545	1546	1547	1550	1551	6800	0	0	0	0	0	0	0	
7000	1553	1554	1555	1556	1557	1560	1561	1562	1563	1564	1565	1566	1567	1570	1571	1572	1573	1574	1575	1576	1577	1600	1601	1602	7000	0	0	0	0	0	0	0	
7200	1604	1605	1606	1607	1610	1611	1612	1613	1614	1615	1616	1620	1621	1622	1623	1624	1625	1626	1627	1630	1631	1632	1633	1634	7200	0	0	0	0	0	0	0	
7400	1635	1636	1637	1640	1641	1642	1643	1644	1645	1646	1647	1650	1651																				

## OCTAL-DECIMAL CONVERSION PROCEDURE

Consider the decimal number to be converted as a base and an increment. Locate the base (the next lower number which is evenly divisible by 200) in the margin of the lower chart and the increment in the body of the upper chart. The intersection of the row and column thus defined contains the high-order digits of the octal equivalent. The low-order digit appears in the margins of the upper chart opposite the increment. For example, to convert 7958 to octal, the base is 7800 and the increment is 158. Locate 158 in the upper chart and read down this column to the 7800 row below. The high-order octal result is 1742. Then read out to the margin of the upper chart to obtain the low-order digit of 6. Append (do not add) this digit to 1742 for an octal equivalent of 17,426.

To convert an octal number to decimal, locate the high-order digits in the body of the lower chart and the low-order digit in the margin of the upper chart. Then perform the converse of the above operation.



APPENDIX B  
MISCELLANEOUS TABLES

Table B-1. Control Register Designations

CONTROL REGISTER	VARIANT CHARACTER (LCR & SCR Instructions)
CLC8	00
CLC1	01
CLC2	02
CLC3	03
CLC8'	04
CLC1'	05
CLC2'	06
CLC3'	07
SLC8	10
SLC1	11
SLC2	12
SLC3	13
SLC8'	14
SLC1'	15
SLC2'	16
SLC3'	17
CLC9	20
CLC4	21
CLC5	22
CLC6	23
CLC9'	24
CLC4'	25
CLC5'	26
CLC6'	27
SLC9	30
SLC4	31
SLC5	32
SLC6	33
SLC9'	34

Table B-1 (cont). Control Register Designations

CONTROL REGISTER	VARIANT CHARACTER (LCR & SCR Instructions)
SLC4'	35
SLC5'	36
SLC6'	37
AC0	--
AC1	--
AC2	--
AC3	--
ATR	54
CSR	64
EIR	66
AAR	67
BAR	70
IIR	76
SR	77

Table B-2. Extended Move (EXM) Conditions

CONDITIONS	VARIANT BITS					
	V <sub>6</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>3</sub>	V <sub>2</sub>	V <sub>1</sub>
<b>Type of Move</b>						
1. A-field data bits → B	X	X	X	X	X	1
2. A-field word-mark bits → B	X	X	X	X	1	X
3. A-field item-mark bits → B	X	X	X	1	X	X
<b>Direction of Move</b>						
1. right to left	X	X	0	X	X	X
2. left to right	X	X	1	X	X	X
<b>Termination of Move</b>						
1. automatic after single-character move	0	0	X	X	X	X
2. A-field word mark	0	1	X	X	X	X
3. A-field item mark	1	0	X	X	X	X
4. A-field record mark	1	1	X	X	X	X

Table B-3. Branch on Condition Test (BCT) SENSE Switch Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
00	Unconditional
01	SENSE Switch 1 On
02	SENSE Switch 2 On
03	SENSE Switches 1 <u>and</u> 2 On
04	SENSE Switch 3 On
05	SENSE Switches 1 <u>and</u> 3 On
06	SENSE Switches 2 <u>and</u> 3 On
07	SENSE Switches 1, 2, <u>and</u> 3 On
10	SENSE Switch 4 On
11	SENSE Switches 1 <u>and</u> 4 On
12	SENSE Switches 2 <u>and</u> 4 On
13	SENSE Switches 1, 2, <u>and</u> 4 On
14	SENSE Switches 3 <u>and</u> 4 On
15	SENSE Switches 1, 3, <u>and</u> 4 On
16	SENSE Switches 2, 3, <u>and</u> 4 On
17	SENSE Switches 1, 2, 3, <u>and</u> 4 On
20	Unconditional
21	SENSE Switch 5 On
22	SENSE Switch 6 On
23	SENSE Switches 5 <u>and</u> 6 On
24	SENSE Switch 7 On
25	SENSE Switches 5 <u>and</u> 7 On
26	SENSE Switches 6 <u>and</u> 7 On
27	SENSE Switches 5, 6, <u>and</u> 7 On
30	SENSE Switch 8 On
31	SENSE Switches 5 <u>and</u> 8 On
32	SENSE Switches 6 <u>and</u> 8 On
33	SENSE Switches 5, 6, <u>and</u> 8 On
34	SENSE Switches 7 <u>and</u> 8 On
35	SENSE Switches 5, 7, <u>and</u> 8 On
36	SENSE Switches 6, 7, <u>and</u> 8 On
37	SENSE Switches 5, 6, 7, <u>and</u> 8 On

NOTE: When testing for a multiple SENSE switch condition, a branch occurs only if all of the specified conditions are met.



Table B-4. Branch on Condition Test (BCT) Indicator Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
40	Do not branch
41	$B < A$ (Low Compare)
42	$B = A$ (Equal Compare)
43	$B \leq A$ (Low or Equal Compare)
44	$B > A$ (High Compare)
45	$B \neq A$ (Unequal Compare)
46	$B \geq A$ (High or Equal Compare)
47	Unconditional
50	Overflow
51	Overflow <u>or</u> $B < A$
52	Overflow <u>or</u> $B = A$
53	Overflow <u>or</u> $B \leq A$
54	Overflow <u>or</u> $B > A$
55	Overflow <u>or</u> $B \neq A$
56	Overflow <u>or</u> $B \geq A$
57	Unconditional
60	Zero Balance
61	Zero Balance <u>or</u> $B < A$
62	Zero Balance <u>or</u> $B = A$
63	Zero Balance <u>or</u> $B \leq A$
64	Zero Balance <u>or</u> $B > A$
65	Zero Balance <u>or</u> $B \neq A$
66	Zero Balance <u>or</u> $B \geq A$
67	Unconditional
70	Overflow <u>or</u> Zero Balance
71	Overflow <u>or</u> Zero Balance <u>or</u> $B < A$
72	Overflow <u>or</u> Zero Balance <u>or</u> $B = A$
73	Overflow <u>or</u> Zero Balance <u>or</u> $B \leq A$
74	Overflow <u>or</u> Zero Balance <u>or</u> $B > A$
75	Overflow <u>or</u> Zero Balance <u>or</u> $B \neq A$
76	Overflow <u>or</u> Zero Balance <u>or</u> $B \geq A$
77	Unconditional

NOTE: When testing for a multiple indicator condition, a branch occurs if any one of the specified conditions is met.

Table B-5. Branch on Character Condition (BCC) Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
00	Unconditional
01	A bit is 1
02	B bit is 1
03	B and A bits are 11
04	B and A bits are 00
05	B and A bits are 01 (Positive sign)
06	B and A bits are 10 (Negative sign)
07	B and A bits are 11 (same as 03)
10	Word-mark bit is 1
11	Word-mark bit is 1, A bit is 1
12	Word-mark bit is 1, B bit is 1
13	Word-mark bit is 1, B and A bits are 11
14	Word-mark bit is 1, B and A bits are 00
15	Word-mark bit is 1, Positive sign
16	Word-mark bit is 1, Negative sign
17	Word-mark bit is 1, B and A bits are 11
20	Item-mark bit is 1
21	Item-mark bit is 1, A bit is 1
22	Item-mark bit is 1, B bit is 1
23	Item-mark bit is 1, B and A bits are 11
24	Item-mark bit is 1, B and A bits are 00
25	Item-mark bit is 1, Positive Sign
26	Item-mark bit is 1, Negative Sign
27	Item-mark bit is 1, B and A bits are 11
30	Record mark
31	Record mark, A bit is 1
32	Record mark, B bit is 1
33	Record mark, B and A bits are 11
34	Record mark, B and A bits are 00
35	Record mark, Positive sign
36	Record mark, Negative sign
37	Record mark, B and A bits are 11
40	No punctuation (Word-mark and Item- mark bits are 00)
41	No punctuation, A bit is 1
42	No punctuation, B bit is 1
43	No punctuation, B and A bits are 11
44	No punctuation, B and A bits are 00
45	No punctuation, Positive sign
46	No punctuation, Negative sign
47	No punctuation, B and A bits are 11
50	Word mark only
51	Word mark only, A bit is 1
52	Word mark only, B bit is 1
53	Word mark only, B and A bits are 11
54	Word mark only, B and A bits are 00
55	Word mark only, Positive sign
56	Word mark only, Negative sign
57	Word mark only, B and A bits are 11

Table B-5 (cont). Branch on Character Condition (BCC) Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
60	Item mark only
61	Item mark only, A bit is 1
62	Item mark only, B bit is 1
63	Item mark only, B and A bits are 11
64	Item mark only, B and A bits are 00
65	Item mark only, Positive sign
66	Item mark only, Negative sign
67	Item mark only, B and A bits are 11
70	Unconditional
71	Word mark <u>or</u> A bit is 1
72	Word mark <u>or</u> B bit is 1
73	Word mark <u>or</u> B and A bits are 11
74	Word mark <u>or</u> B and A bits are 00
75	Word mark <u>or</u> Positive sign
76	Word mark <u>or</u> Negative sign
77	Word mark <u>or</u> B and A bits are 11

Table B-6. Series 2000 Character Codes

Key Punch	Card Code	Central Processor Code	Octal	High Speed Printer	Key Punch	Card Code	Central Processor Code	Octal	High Speed Printer
0	0	000000	00	0	0̄ or -	X, 0 or X <sup>(1)</sup>	100000	40	-
1	1	000001	01	1	J	X, 1	100001	41	J
2	2	000010	02	2	K	X, 2	100010	42	K
3	3	000011	03	3	L	X, 3	100011	43	L
4	4	000100	04	4	M	X, 4	100100	44	M
5	5	000101	05	5	N	X, 5	100101	45	N
6	6	000110	06	6	O	X, 6	100110	46	O
7	7	000111	07	7	P	X, 7	100111	47	P
8	8	001000	10	8	Q	X, 8	101000	50	Q
9	9	001001	11	9	R	X, 9	101001	51	R
#	8, 2	001010	12	'		X, 8, 2	101010	52	#
@	8, 3	001011	13	=	\$	X, 8, 3	101011	53	\$
Space	8, 4	001100	14	:	*	X, 8, 4	101100	54	*
	Blank	001101	15	Blank		X, 8, 5	101101	55	"
	8, 6	001110	16	> <sup>(2)</sup>		X, 8, 6	101110	56	≠ <sup>(2)</sup>
& or 0	8, 7	001111	17	&	- or 0̄	X or X, 0 <sup>(1)</sup>	101111	57	1/2 or ! <sup>(2)</sup> <sup>(3)</sup>
	R, 0 or R <sup>(1)</sup>	010000	20	+		8, 5	110000	60	< <sup>(2)</sup>
A	R, 1	010001	21	A	/	0, 1	110001	61	/
B	R, 2	010010	22	B	S	0, 2	110010	62	S
C	R, 3	010011	23	C	T	0, 3	110011	63	T
D	R, 4	010100	24	D	U	0, 4	110100	64	U
E	R, 5	010101	25	E	V	0, 5	110101	65	V
F	R, 6	010110	26	F	W	0, 6	110110	66	W
G	R, 7	010111	27	G	X	0, 7	110111	67	X
H	R, 8	011000	30	H	Y	0, 8	111000	70	Y
I	R, 9	011001	31	I	Z	0, 9	111001	71	Z
	R, 8, 2	011010	32	;		0, 8, 2	111010	72	@
	R, 8, 3	011011	33	.	,	0, 8, 3	111011	73	,
□	R, 8, 4	011100	34	)	%	0, 8, 4	111100	74	(
	R, 8, 5	011101	35	%		0, 8, 5	111101	75	( <sub>R</sub> )
	R, 8, 6	011110	36	■		0, 8, 6	111110	76	□ <sup>(2)</sup>
& or 0	R or R, 0 <sup>(1)</sup>	011111	37	? <sup>(2)</sup>		0, 8, 7	111111	77	¢ <sup>(2)</sup>

<sup>(1)</sup> Special Code. The second (alternative) card-code/central processor code equivalency is in effect when control character 26 is coded in a card read or punch PCB instruction.

<sup>(2)</sup> Indicates symbol which will be printed by a printer which has a 63-character drum (Type 222 printers).

<sup>(3)</sup> The exclamation point replaces the one-half symbol on a type roll containing the Mark II character font.

Table B-7. Binary, Octal, and Decimal Equivalents

BIN.	OCT.	DEC.	BIN.	OCT.	DEC.
0	0	0	100000	40	32
1	1	1	100001	41	33
10	2	2	100010	42	34
11	3	3	100011	43	35
100	4	4	100100	44	36
101	5	5	100101	45	37
110	6	6	100110	46	38
111	7	7	100111	47	39
1000	10	8	101000	50	40
1001	11	9	101001	51	41
1010	12	10	101010	52	42
1011	13	11	101011	53	43
1100	14	12	101100	54	44
1101	15	13	101101	55	45
1110	16	14	101110	56	46
1111	17	15	101111	57	47
10000	20	16	110000	60	48
10001	21	17	110001	61	49
10010	22	18	110010	62	50
10011	23	19	110011	63	51
10100	24	20	110100	64	52
10101	25	21	110101	65	53
10110	26	22	110110	66	54
10111	27	23	110111	67	55
11000	30	24	111000	70	56
11001	31	25	111001	71	57
11010	32	26	111010	72	58
11011	33	27	111011	73	59
11100	34	28	111100	74	60
11101	35	29	111101	75	61
11110	36	30	111110	76	62
11111	37	31	111111	77	63

Table B-8. Powers of 2

n	2 <sup>n</sup>
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1 024
11	2 048
12	4 096
13	8 192
14	16 384
15	32 768
16	65 536
17	131 072
18	262 144
19	524 288
20	1 048 576
21	2 097 152
22	4 194 304
23	8 388 608
24	16 777 216

Table B-9. Move or Scan Variants

MOVE OPERATION CODES

Mnemonic Op Code	Statement Name	Corresponding Move or Scan Variant
MLC	Move Left Characters	63
MLN	Move Left Numerics	61
MLW	Move Left Word Marks	64
MLZ	Move Left Zones	62
MLCA	Move Left Characters to A-Field Word Mark	23
MLCB	Move Left Characters to B-Field Word Mark	43
MLCS	Move Left Character Single	03
MLCW	Move Left Characters and Word Marks	67
MLNA	Move Left Numerics to A-Field Word Mark	21
MLNB	Move Left Numerics to B-Field Word Mark	41
MLNS	Move Left Numeric Single	01
MLNW	Move Left Numerics and Word Marks	65
MLWA	Move Left Word Marks to A-Field Word Mark	24
MLWB	Move Left Word Mark to B-Field Word Mark	44
MLWS	Move Left Word Mark Single	04
MLZA	Move Left Zones to A-Field Word Mark	22
MLZB	Move Left Zones to B-Field Word Mark	42
MLZS	Move Left Zone Single	02
MLZW	Move Left Zones and Word Marks	66
MLCWA	Move Left Characters and Word Mark to A-Field Word Mark	27
MLCWB	Move Left Characters and Word Mark to B-Field Word Mark	47
MLCWS	Move Left Characters and Word Mark Single	07
MLNWA	Move Left Numerics and Word Mark to A-Field Word Mark	25
MLNWB	Move Left Numerics and Word Mark to B-Field Word Mark	45
MLNWS	Move Left Numeric and Word Mark Single	05
MLZWA	Move Left Zones and Word Mark to A-Field Word Mark	26
MLZWB	Move Left Zones and Word Mark to B-Field Word Mark	46
MLZWS	Move Left Zones and Word Mark Single	06
MRC	Move Right Characters	13

Table B-9 (cont). Move or Scan Variants

MOVE OPERATION CODES

Mnemonic Op Code	Statement Name	Corresponding Move or Scan Variants
MRN	Move Right Numerics	11
MRW	Move Right Word Marks	14
MRZ	Move Right Zones	12
MRCG	Move Right Characters to A-Field Group Mark- Word Mark	53
MRCM	Move Right Characters to A-Field Record Mark or Group Mark-Word Mark	73
MRCR	Move Right Characters to A-Field Record Mark	33
MRCW	Move Right Characters and Word Mark to A- or B- Field Word Mark	17
MRNG	Move Right Numerics to A-Field Group Mark-Word Mark	51
MRNM	Move Right Numerics to A-Field Record Mark or Group Mark-Word Mark	71
MRNR	Move Right Numerics to A-Field Record Mark	31
MRNW	Move Right Numerics and Word Mark to A- or B- Field Word Mark	15
MRWG	Move Right Word Marks to A-Field Group Mark- Word Mark	54
MRWM	Move Right Word Marks to A-Field Record Mark or Group Mark-Word Mark	74
MRWR	Move Right Word Marks to A-Field Record Mark	34
MRZG	Move Right Zones to A-Field Group Mark-Word Mark	52
MRZM	Move Right Zones to A-Field Record Mark or Group Mark-Word Mark	72
MRZR	Move Right Zones to A-Field Record Mark	32
MRZW	Move Right Zones and Word Mark to A- or B-Field Word Mark	16
MRCWG	Move Right Characters and Word Marks to A-Field Group Mark-Word Mark	57
MRCWM	Move Right Characters and Word Marks to A-Field Record Mark-Group Mark-Word Mark	77
MRCWR	Move Right Characters and Word Marks to A-Field Record Mark	37
MRNWG	Move Right Numerics and Word Marks to A-Field Group Mark-Word Mark	55
MRNWM	Move Right Numerics and Word Marks to A-Field Record Mark-Group Mark-Word Mark	75

Table B-9 (cont). Move or Scan Variants

MOVE OPERATION CODES

Mnemonic Op Code	Statement Name	Corresponding Move or Scan Variants
MRNWR	Move Right Numerics and Word Marks to A-Field Record Mark	35
MRZWG	Move Right Zones and Word Marks to A-Field Group Mark-Word Mark	56
MRZWM	Move Right Zones and Word Marks to A-Field Record Mark-Group Mark-Word Mark	76
MRZWR	Move Right Zones and Word Marks to A-Field Record Mark	36
<u>SCAN OPERATION CODES</u>		
SCNL	Scan Left to A- or B-Field Word Mark	60
SCNR	Scan Right to A- or B-Field Word Mark	10
SCNLA	Scan Left to A-Field Word Mark	20
SCNLB	Scan Left to B-Field Word Mark	40
SCNLS	Scan Left Single Position	00
SCNRG	Scan Right to A-Field Group Mark-Word Mark	50
SCNRM	Scan Right to A-Field Record Mark or Group Mark-Word Mark	70
SCNRR	Scan Right to A-Field Record Mark	30
NOTE: Any move or scan variant with X3 or X7 format moves item marks as well as data and/or word marks.		





APPENDIX C  
INSTRUCTION SUMMARY

INSTRUCTIONS FORMATS AND TIMING

Each Series 2000 single-character processor instruction is described in terms of its operation code, formats, and timing formulas in Table C-1. Table C-2 lists the instruction timings in memory cycles for the Models 2040A, 2050, and 2060. Timing formulas for the Models 2050A and 2070 are given in Table C-3.

The formulas given in both tables provide execution time in memory cycles. Equivalent expressions for symbols used in Tables C-1 and C-4 are as follows:

SYMBOL	MEANING
A	Address of A-operand field.
B	Address of B-operand field.
h	The sum of the values of the multiplier digits which are less than or equal to five, plus the sum of the elevens complements of all digits whose values are greater than five.
$N_a$	Number of characters in the A-operand field.
$N_{aw}$	Number of words in the A-operand field.
$N_b$	Number of characters in the B-operand field.
$N_{bw}$	Number of words in the B-operand field.
$N_{b1}$	Number of words that the A field occupies in the B field, whether or not the operands have been modified by an arithmetic operation.
$N_{b2}$	Number of words in the B-operand field excluding $N_{b1}$ .
$N_c$	Number of control characters in the instruction.
$N_{cn}$	Number of control characters following control character 3 (C3).
$N_{dd}$	Number of digits in the dividend.
$N_i$	Number of characters in the instruction.
$N_{ia}$	Number of words in the item to be translated.
$N_{ib}$	Number of words in the result item.
$N_{ic}$	Number of translation units (6-bit or 12-bit characters) to be translated.
$N_j$	Number of character locations bypassed to reach the next sequential op code.

SYMBOL	MEANING
$N_m$	Number of characters moved.
$N_{mr}$	Number of digits in the multiplier.
$N_q$	Number of digits in the quotient ( $=N_{dd} - Z_{ld} - N_a + Z_{1a} + 1$ ).
$N_r$	Number of characters referenced.
$N_{sc}$	Number of characters scanned.
$N_{st}$	Number of characters stored.
$N_w$	Number of characters in the A- or B-operand field, whichever is shorter.
$N_{wj}$	Number of words bypassed to reach the next sequential op code.
$N_{ws}$	Number of words stored.
$n$	Number of items in the table or the number of times the A operand is compared against some portion of the B operand.
$Q_i$	The value of the " $i^{th}$ " digit of the quotient.
$s$	Sum of all multiplier digits.
SUM	Sum of the upwards-rounded values of all multiplier digits divided by 2.
V	Variant character.
W	Number of memory words used to store the data involved.
$W_i$	Number of four-character words used to store one more than the total number of characters in the instruction.
$W_{mr}$	Number of words in the multiplier.
$X_0$	Zero if no second scan (zero suppression); one if the scan is performed.
$Y_0$	Zero if no third scan (dollar-sign insertion); one if the scan is performed.
Z	Number of characters scanned during zero suppression.
$Z_{1a}$	Number of leading zeros in the A-operand field.
$Z_{law}$	Number of words containing leading zeros in the A-operand field.
$Z_{ld}$	Number of leading zeros in the dividend.
$Z_{mr}$	Number of 0's in the multiplier.
$Z_{ta}$	Number of trailing 0's (i.e., consecutive low-order zeros) in the A-operand field.
$Z_{taw}$	Number of words containing trailing zeros in the A-operand field.
$Z_w$	Number of words scanned during zero suppression.
$Z_z$	Zero if $Z_{1a} = 0$ ; one if $Z_{1a} \neq 0$ .
\$	Number of characters scanned during dollar-sign insertion.
$\$w$	Number of words scanned during dollar-sign insertion.

NOTE: The timing formulas presented in Tables C-1, C-2, C-3, and C-4 are based on the use of direct addressing. If address modification is used, the formulas in Tables C-1 and C-4 for the Model 2040 should be modified as follows:

1. Indirect Addressing – Add one memory cycle for each character extracted as a result of indirect addressing.
2. Indexed Addressing – Add three memory cycles for each indexed address.

Likewise, the use of address modification requires that the formulas in Tables C-2 and C-3 for the Models 2040A, 2050, 2050A, 2060, and 2070 be modified as follows:

1. Indirect Addressing – Add 1.16 memory cycles for each indirect address formed plus one memory cycle for each word extracted as a result of indirect addressing.
2. Indexed Addressing – Add 3.167 memory cycles if one address is indexed, 5.16 memory cycles if both addresses are indexed.

Table C-1. Instruction Summary – Timing Formulas for Model 2040\*

Op Code				Function	Timing (Memory Cycles)	Format	Extraction Path <sup>1</sup>	Required Word Marks	Terminated By:	Can Instruction Be Chained?
Mnemonic	Op- tal	Card Code	Key Punch							
ARITHMETIC INSTRUCTIONS										
A	36	R, 8, 6		Decimal Add	$N_i+2N_w+2N_b$ (no recomplement) <sup>3</sup> $N_i+2N_w+4N_b$ (recomplement) <sup>3</sup>	a. A/A, B b. A/A c. A/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
S	37	R or R, 0 <sup>2</sup>	&	Decimal Subtract	$N_i+2N_w+2N_b$ (no recomplement) <sup>3</sup> $N_i+2N_w+4N_b$ (recomplement) <sup>3</sup>	a. S/A, B b. S/A c. S/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
BA	34	R, 8, 4	□	Binary Add	$N_i+1+N_w+2N_b$	a. BA/A, B b. BA/A c. BA/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
BS	35	R, 8, 5		Binary Subtract	$N_i+1+N_w+2N_b$	a. BS/A, B b. BS/A c. BS	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
ZA	16	8, 6		Zero and Add	$N_i+1+N_w+N_b$	a. ZA/A, B b. ZA/A c. ZA/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
ZS	17	8, 7		Zero and Sub- tract	$N_i+1+N_w+N_b$	a. ZS/A, B b. ZS/A c. ZS/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
M	26	R, 6	F	Decimal Multiply	See Table C-4.	a. M/A, B b. M/A c. M/	Preserves B.	A and B fields.	Both word marks.	Yes.
D	27	R, 7	G	Decimal Divide	See Table C-4.	a. D/A, B b. D/A c. D/	Preserves B.	A operand (divisor).	A-operand word mark.	Yes.
LOGIC INSTRUCTIONS										
EXT	31	R, 9	I	Extract (Logical Product)	$N_i+1+3N_w$	a. EXT/A, B b. EXT/A c. EXT/	Preserves B.	Smaller oper- and.	Word mark of smaller operand.	Yes.
HA	30	R, 8	H	Half Add (Exclusive Or)	$N_i+1+3N_w$	a. HA/A, B b. HA/A c. HA/	Preserves B.	Smaller oper- and.	Word mark of smaller operand.	Yes.
SST	32	R, 8, 2		Substitute	$N_i+4$	a. SST/A, B, V b. SST/A, B c. SST/A d. SST/	Preserves B.	None.	Single- character operation.	Yes.
C	33	R, 8, 3		Compare	$N_i+2+N_w+N_b$	a. C/A, B b. C/A c. C/	Preserves B.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.
B	65	0, 5	V	Branch (Unconditional)	$N_i+2$	a. B/A	Bypasses B.	None.	n/a	No.
BCT	65	0, 5	V	Branch on Condition Test	$N_i+2$	a. BCT/A, V b. BCT/	Bypasses B.	None.	n/a	Yes.
BCC	54	X, 8, 4	*	Branch on Character Condition	$N_i+4$	a. BCC/A, B, V b. BCC/A, B c. BCC/A d. BCC	Preserves B.	None.	Single- character operation.	Yes.
BCE	55	X, 8, 5		Branch if Character Equal	$N_i+4$	a. BCE/A, B, V b. BCE/A, B c. BCE/A d. BCE/	Preserves B.	None.	Single- character operation.	Yes.
BBE	56	X, 8, 6		Branch on Bit Equal	$N_i+4$	a. BBE/A, B, V b. BBE/A, B c. BBE/A d. BBE/	Preserves B.	None.	Single- character operation.	Yes.
CONTROL INSTRUCTIONS										
SW	22	R, 2	B	Set Word Mark	$N_i+3$ <sup>4</sup>	a. SW/A, B b. SW/A c. SW/	Duplicates A.	None.	n/a	Yes.
SI	20	R, 0 or R <sup>3</sup>	&	Set Item Mark	$N_i+3$ <sup>4</sup>	a. SI/A, B b. SI/A c. SI/	Duplicates A.	None.	n/a	Yes.
CW	23	R, 3	C	Clear Word Mark	$N_i+3$	a. CW/A, B b. CW/A c. CW/	Duplicates A.	Word marks are cleared.	n/a	Yes.
CI	21	R, 1	A	Clear Item Mark	$N_i+3$	a. CI/A, B b. CI/A c. CI/	Duplicates A.	None.	n/a	Yes.
H	45	X, 5	N	Halt	$N_i+2$	a. H/ b. H/A c. H/A, B d. H/A, B, V	Preserves B.	None.	n/a	No.
NOP	40		-	No Operation	$N_i+2$ <sup>3</sup>	a. NOP/	Bypasses A and B.	None.	n/a	No.
MCW	14	8, 4		Move Characters to Word Mark	$N_i+1+2N_w$	a. MCW/A, B b. MCW/A c. MCW	Preserves B.	Smaller operand.	Word mark of smaller operand.	Yes.

Table C-1 (cont). Instruction Summary – Timing Formulas for Model 2040\*

Op Code				Function	Timing (Memory Cycles)	Format	Extraction Path <sup>1</sup>	Required Word Marks	Terminated By:	Can Instruction Be Chained?
Mne- monic	Oc- tal	Card Code	Key Punch							
CONTROL INSTRUCTIONS (cont)										
LCA	15	Blank	Space	Load Characters to A-Field Word Mark	$N_i+1+2N_a$	a. LCA/A, B b. LCA/A c. LCA/	Preserves B.	A operand.	A-operand word mark.	Yes.
SCR	24	R, 4	D	Store Control Registers	$N_i+5$	a. SCR/A, V b. SCR/A c. SCR/	Bypasses B.	None.	n/a	Yes.
LCR	25	R, 5	E	Load Control Registers	$N_i+5$	a. LCR/A, V b. LCR/A c. LCR/	Bypasses B.	None.	n/a	Yes.
CAM	42	X, 2	K	Change Addressing Mode	$N_i+2^3$	a. CAM/V b. CAM/	Bypasses A and B.	None.	n/a	Yes.
CSM	43	X, 3	L	Change Sequencing Mode	$N_i+3^3$	a. CSM/ b. CSM/A c. CSM/A, B d. CSM/A, B, V	Preserves B.	None.	n/a	Yes.
EXM	10	8	8	Extended Move	$N_i+1+2N_a$	a. EXM/A, B, V b. EXM/A, B c. EXM/A d. EXM/	Preserves B.	See page 8-67	See page 8-67	Yes.
MAT	60	8, 5		Move and Translate	$N_i+3N_a$	a. MAT/A, B, V, V <sub>2</sub> b. MAT/A, B, C	See page 8-70	A operand.	Word mark in A operand or in table.	No.
MIT	62	0, 2	S	Move Item and Translate	$N_i+N_a+2N_{ic}$	a. MIT/A, B, V <sub>1</sub> , V <sub>2</sub> , V <sub>3</sub> b. MIT/A, B, C, V <sub>1</sub>	See page 8-74	None.	A-operand item mark or word mark in table.	No.
LIB	77	0, 8, 7		Load Index/Barricade Register	$N_i+3$ $N_i+5$	a. LIB/A b. LIB/A/B	Preserves B.	None.	Single-character operation.	Yes.
SIB	76	0, 8, 6		Store Index/Barricade Register	$N_i+3$ $N_i+5$	a. SIB/A b. SIB/A/B	Preserves B.	None.	Single-character operation.	Yes.
TLU	57	R, 6	F	Table Lookup	$N_i+1+(N_a)_b+N_b$	a. TLU/A, B, V b. TLU/A, B c. TLU/A d. TLU	Preserves B.	A operand.	A-operand word mark.	Yes.
MOS	13	8, 3	#	Move or Scan	$N_i+1+3(N_m)$ (Move) $N_i+1+3(N_{sc})$ (Scan)	a. MOS/A, B, Y b. MOS/A, B c. MOS/A d. MOS	Preserves B	See page 8-87	See page 8-87	Yes.
INTERRUPT CONTROL INSTRUCTIONS										
SVI	46	X, 6	O	Store Variant and Indicators	$N_i+2+N_{st}+N_j$	a. SVI/V	Bypasses A and B.	See page 8-94	See page 8-94	No.
RVI	67	0, 7	X	Restore Variant and Indicators	$N_i+2+N_r^3$	a. RVI/A, V	Restores A and bypasses B.	None.	Word mark of next instruction.	No.
MC	44	X, 4	M	Monitor Call	$N_i+2^3$	a. MC/	Bypasses A and B.	None.	Word mark of next instruction.	No.
RNM	41	X, 1	J	Resume Normal Mode	$N_i+3^5$	a. RNM/A, B b. RNM/A c. RNM/	Preserves B.	None.	n/a	No.
EDITING INSTRUCTION										
MCE	74	0, 8, 4	%	Move Characters and Edit	$N_i+1+N_a+2N_b+2Z+2S$	a. MCE/A, B b. MCE/A c. MCE/	Preserves B.	A operand and B operand (see page 8-106)	See page 8-106	No.
INPUT/OUTPUT INSTRUCTIONS										
PDT	66	0, 6	W	Peripheral Data Transfer	$(N_i-N_{c1}+1) + (N_{cn}+3)$ + data transfer time.	a. PDT/A, C <sub>1</sub> , .. C <sub>n</sub>	Bypasses B.	None.	Record, mark in memory or unit record length.	No.

Table C-1 (cont). Instruction Summary – Timing Formulas for Model 2040\*

Op Code				Function	Timing (Memory Cycles)	Format	Extraction Path <sup>1</sup>	Required Word Marks	Terminated By:	Can Instruction Be Chained?
Mne- monic	Octal	Card Code	Key Punch							
INPUT/OUTPUT INSTRUCTIONS (cont)										
PCB	64	0, 4	U	Peripheral Control and Branch	$(N_i - N_c + 1) + N_c$	a. PCB/A, C <sub>1</sub> ...C <sub>n</sub>	Bypasses B.	None.	n/a	No.
<p>*All information given in this table, other than timing formulas, is applicable to the multicharacter processors. See Appendix D for information concerning Scientific Unit and Subprocessor.</p> <p><sup>1</sup>The extraction path of the various instructions is defined as follows:  <u>Preserves B</u> – The previous contents of BAR are used as the B address when the instruction is coded in the format Op Code/A.  <u>Duplicates A</u> – The contents of AAR are used as the B address when the instruction is coded in the format Op Code/A.  <u>Bypasses B</u> – The contents of BAR are not used in any format.  <u>Bypasses A and B</u> – The contents of AAR and BAR are not used in any format.</p> <p><sup>2</sup>The second (alternate) card code is in effect when control character 26 is coded in a Card Read or Punch PCB instruction.</p> <p><sup>3</sup>Subtract one memory cycle from this formula if the instruction is executed in a Type 2041 processor.</p> <p><sup>4</sup>Subtract one memory cycle from this formula if the instruction is issued in a Type 2041 processor in the format Op Code/A, B.</p> <p><sup>5</sup>Subtract one memory cycle from the formula if the instruction is executed in a Type 2041 processor.</p>										

The following symbols and meanings should be used to interpret Table C-2 only.

SYMBOL	MEANING
$N_a$	Length of A-operand.
$N_b$	Length of B-operand.
$N_w$	Length of shorter of the two operands.
T	0 if terminating punctuation is sensed in the A-field; 2 if terminating punctuation is sensed in the B-field and not in the A-field.
X	Number of characters scanned in Edit, second pass.
Y	Number of characters scanned in Edit, third pass.
n	Number of times the A-field is scanned.
$N_{tb}$	Number of characters read out of translation table.
$N_m$	Number of characters moved.
$N_s$	Number of characters scanned.
$k_1$	0.5 if $N_w$ is odd and $N_b$ is even. 0.0 if $N_w$ and $N_b$ are even. 0.1 if $N_w$ is even and $N_b$ is odd. 1.5 if $N_w$ and $N_b$ are odd.
$k_2$	0.5 if $N_w$ is odd and $N_b$ is even, or $N_w$ is even and $N_b$ is odd. 1.0 if $N_w$ and $N_b$ are odd. 0.0 if $N_w$ and $N_b$ are even.
$k_3$	1.5 if $N_w$ is odd. 0.0 if $N_w$ is even.
$k_4$	0.5 if $N_a$ is odd and $N_b$ is even, or $N_a$ is even and $N_b$ is odd. 1.0 if $N_a$ and $N_b$ are both odd. 0.0 if $N_a$ and $N_b$ are both even.
$k_5$	0.0 if $N_s$ is even. 0.5 if $N_s$ is odd.
$k_6$	0.0 if X and Y are even. 1.0 if X is odd and Y is even, or X is even and Y is odd. 2.0 if X and Y are odd.
$k_7$	1 if B-field is odd. 0 if B-field is even.
$N_{ap}$	Number of character pairs in the A-field (upward rounded).
$N_{bp}$	Number of character pairs in the B-field (upward rounded).



Multiply

$N_{dap}$	Number of pairs of multiplicand digits excluding trailing zeroes.
$N_{dbp}$	Number of pairs of multiplier digits excluding trailing zeroes.
$N_{mp}$	Number of digit pairs in multiplier excluding trailing zeroes ( $N_{bp} - Z_{mrp}$ ).
SUM	(The sum of all even multiplier digits plus all other multiplier digits individually increased by one) divided by two.
$Z_{mrp}$	Number of trailing zero pairs in multiplier (downward rounded)
$Z_{tap}$	Number of trailing zero pairs in multiplicand (downward rounded).

Divide

$N_{dap}$	Number of divisor digit pairs ( $N_a - Z_a$ ), upward rounded.
$N_{qt}$	Number of characters in the quotient, where $N_{qt} = N_b - Z_b - N_a + Z_a + 1$
$Z_a$	Number of leading divisor zeroes.
$Z_b$	Number of leading dividend zeroes.
$Z_{ap}$	Number of leading zero pairs in the divisor (upward rounded).
$Z_{bp}$	Number of leading zero pairs in the dividend (upward rounded).

Table C-2. Instruction Timings for Models 2040A, 2050, and 2060

Mnemonic Op Code	Instruction Name	Timing (Memory Cycles)
Fixed-Point Arithmetic Instructions		
A	Decimal Add	$.5N_w + N_b + k_1$ ①
S	Decimal Subtract	$.5N_w + N_b + k_1$ ①
BA	Binary Add	$.5N_w + N_b + k_1$
BS	Binary Subtract	$.5N_w + N_b + k_1$
ZA	Zero and Add	$.5N_w + 5N_b + k_2$
ZS	Zero and Subtract	$.5N_w + 5N_b + k_2$
M	Decimal Multiply	If $N_a$ or $N_b = 0$ : $2 + 2N_{ap} + 2N_{bp}$ If $N_a$ and $N_b = 0$ : $2 + 2N_{ap} + 2N_{bp} + \text{SUM} \cdot N_{mp}$ $+ 2 \left[ N_{ap} - Z_{mrp} \right] N_{mp} + N_{mp}$

Table C-2 (cont). Instruction Timings for Models 2040A, 2050, and 2060

Mnemonic Op Code	Instruction Name	Timing (Memory Cycles)
D	Decimal Divide	If divisor=0: $1+2N_{dap}$ If $N_a > N_b$ (indivisible): $4+4N_{dap}$ If $N_a - Z_a > N_b - Z_b$ (indivisible): $5+2N_{dap} + 2Z_{ap} + 3Z_{bp}$ If divisible [A] 0: $3+2N_{dap} + 2Z_{ap} + 3Z_{bp} + N_{qt}$ $(15N_{dap} + 2) + .6N_{qt} \cdot N_{dap}$
Logic Instructions		
EXT	Extract	$1.5N_w + k_3$
HA	Half Add	$1.5N_w + k_3$
SST	Substitute	3
C	Compare	$.5N_w + .5N_b + k_2 + 1$
B or BCT	Branch	1 (Includes test of eight SENSE switches)
BCC	Branch on Character Condition	2
BCE	Branch on Character Equal	2
BBE	Branch on Bit Equal	2
Control Instructions		
SW	Set Word Mark	2
SI	Set Item Mark	2
CW	Clear Word Mark	2
CI	Clear Item Mark	2
H	Halt	1
NOP	No Operation	1
MCW	Move Characters to Word Mark	$N_w^{(2)}$
LCA	Load Characters to A-field Word Mark	$N_a^{(2)}$
SCR	Store Control Registers	3
LCR	Load Control Registers	3

Table C-2 (cont). Instruction Timings for Models 2040A, 2050, and 2060

Mnemonic Op Code	Instruction Name	Timing (Memory Cycles)
CAM	Change Addressing Mode	1
CSM	Change Sequencing Mode	1 When item-mark trapping is in effect, any item-marked op code is treated as CSM.
EXM	Extended Move	$N_a^{(2)}$
MAT	Move and Translate	$2N_a + k_7$
MIT	Move Item and Translate	If V=00: $2N_a + k_7$ If V=02: $3N_{ap}$ If V=01: $3N_a$ If V=03: $3N_{ap}$
TLU	Table Lookup	$n [ .5N_a + .5N_b + k_4 ]$
MOS	Move or Scan	Move= $1.5N + k_3$
Interrupt Control Instructions		
MC	Monitor Call	1 (Not including the time used in honoring the interrupt signal.) <sup>3</sup>
SVI	Store Variant and Indicators	.5 (number of characters stored plus number skipped over to reach next op code) + 1
RVI	Restore Variant and Indicators	.5 (number of characters referenced) + 1
RNM	Resume Normal Mode	1
Edit Instruction		
MCE	Edit	$N_a + k_7 + N_b + X + Y + k_6$
Input/Output Instructions		
PCB	Peripheral Control and Branch	Timing proceeds normally until the control character designating the read/write channel is extracted. Subsequent characters must be transferred to the I/O traffic control of the sector designated by the sector bits of the control character. This can occur only during the memory cycle set aside for that sector. Therefore timings vary.  If an external interrupt signal is received during a PDT in which the RWC of the peripheral control is busy, extraction of the PDT must begin again.
PDT	Peripheral Data Transfer	
<sup>1</sup> Add $.5N_b$ when the sign of the result differs from the original sign of the B-field. <sup>2</sup> Add one memory cycle if the terminating field has an odd number of characters. <sup>3</sup> The time used in honoring a peripheral interrupt, control panel interrupt, or monitor call signal is three processor-allocated memory cycles.		

The following symbols and meanings should be used to interpret Table C-3 only.

SYMBOL	MEANING
$N_a$	Length of A-operand.
$N_b$	Length of B-operand.
$N_w$	Length of shorter of the two operands.
$n$	Number of times the A-field is scanned.
$k_1$	$2 \left( \left\lfloor \frac{N_b}{4} \right\rfloor \right)$ if sign of result differs from original sign; otherwise 0.
ZS	Number of characters between zero suppression signal and B-field word mark.
\$	Number of characters between zero suppression symbol and most significant non-zero digit plus one. \$ = 0 if "float dollar sign" is not involved.
$N_q$	Number of characters in a quotient where $N_q = N_{dd} - Z_{1b} - N_a + Z_{1a} + 1$ .
$Z_{1a}$	Number of leading divisor zeros.
$Z_{1b}$	Number of leading dividend zeros.
$N_{dd}$	Number of dividend digits including leading zeros.
P	0.00556 probability of intermediate remainder correction due to subtraction.
$N_{ddr}$	The remainder of $\frac{N_{dd} - Z_{1b}}{4}$ .
$N_{ar}$	The remainder of $\frac{N_a}{4}$ .
$N_{br}$	The remainder of $\frac{N_b}{4}$ .
K1	0 if $N_{ar} + N_{br}$ is greater than 4; otherwise K1 = 1.
K2	$(N_{ar} + N_{br})$ or 4, whichever is smaller.
$\sum_{q=1}^q$	Value of sum of quotient digits (e. g., for quotient = 578, $\sum_{q=1}^q = 20$ ).

Table C-3. Instruction Timings for Models 2050A and 2070

Mnemonic Op Code	Instruction Name	Timing Formulas (in Memory Cycles)	Notes
Fixed-Point Arithmetic Instructions			
A	Decimal Add	$\left( \left\lfloor \frac{N_w}{4} \right\rfloor \right) + 2 \left( \left\lfloor \frac{N_b}{4} \right\rfloor \right) + k_1$	Note 2.
S	Decimal Subtract	Same as decimal add.	Note 2.
BA	Binary Add	$\left( \left\lfloor \frac{N_w}{4} \right\rfloor \right) + 2 \left( \left\lfloor \frac{N_b}{4} \right\rfloor \right)$	Note 2.

Table C-3 (cont). Instruction Timings for Models 2050A and 2070

Mnemonic Op Code	Instruction Name	Timing Formulas (in Memory Cycles)	Notes
Fixed-Point Arithmetic Instructions (cont).			
BS	Binary Subtract	Same as binary add.	Note 2.
ZA	Zero and Add	$(\lfloor \frac{N_w}{4} \rfloor) + (\lfloor \frac{N_b}{4} \rfloor)$	
ZS	Zero and Subtract	Same as zero and add.	
M	Decimal Multiply	If A and B are not equal to 0: $4 + 2 \lfloor \frac{N_a}{4} \rfloor + 3 \lfloor \frac{N_b}{4} \rfloor - K1$ $+ 14 [ \lfloor \frac{N_a}{4} \rfloor \cdot (\lfloor \frac{N_b}{4} \rfloor - 1) ]$ $+ (\lfloor \frac{N_a}{4} \rfloor - 1) (2N_{br} + 6)$ $+ 2N_{br} + K2$ If A=0 or B=0: $2 + 2 \lfloor \frac{N_a}{4} \rfloor + 2 \lfloor \frac{N_b}{4} \rfloor$	Note 6.
D	Decimal Divide	If divisor $\neq 0$ and $(N_{dd} - Z_{1b}) \geq (N_a - Z_{1a})$ : $T = \lfloor \frac{N_a}{4} \rfloor + \lfloor \frac{N_{dd}}{4} \rfloor + 2 (\lfloor \frac{N_a - Z_{1a}}{4} \rfloor)$ $+ \lfloor \frac{N_q}{4} \rfloor + \sum_{q=1}^q N_q (3) (\lfloor \frac{N_a - Z_{1a}}{4} \rfloor)$ $+ [ N_q \cdot P (\lfloor \frac{N_a - Z_{1a}}{4} \rfloor) ]$ $+ (2N_q - 1) + 3.$ If divisor = 0: $T = \frac{N_a}{4} + 1.$ If $N_{dd} - Z_{1b} < N_a - Z_{1a}$ (dividend < divisor): $T = \lfloor \frac{N_a}{4} \rfloor + 5$ $+ \frac{N_{dd}}{4} + 2 (\lfloor N_{dd} - Z_{1b} \rfloor)$ $+ N_{ddr}$	Notes 7, 8.

Table C-3 (cont). Instruction Timings for Models 2050A and 2070

Mnemonic Op Code	Instruction Name	Timing Formulas (in Memory Cycles)	Notes
Logic Instructions			
EXT	Extract	$3 \left( \left\lceil \frac{N_w}{4} \right\rceil \right)$	
HA	Half Add	Same as extract.	
SST	Substitute	3	
C	Compare	$\left( \left\lceil \frac{N_w}{4} \right\rceil \right) + \left( \left\lceil \frac{N_b}{4} \right\rceil \right) + 1$	
B	Branch (unconditional)	1	Includes test of 8 sense switches. Note 4.
BCT	Branch on Condition Test	1	Includes test of 8 sense switches. Note 4.
BCC	Branch on Character Condition	2	
BCE	Branch on Character Equal	2	
BBE	Branch on Bit Equal	2	
Control Instructions			
SW	Set Word Mark	2	Note 1.
SI	Set Item Mark	2	Note 1.
CW	Clear Word Mark	2	Note 1.
CI	Clear Item Mark	2	Note 1.
H	Halt	1	Note 5.
NOP	No Operation	1	
MCW	Move Characters to Word Mark	$2 \left( \left\lceil \frac{N_w}{4} \right\rceil \right)$	
LCA	Load Characters to A-Field Word Mark	$2 \left( \left\lceil \frac{N_a}{4} \right\rceil \right)$	
SCR	Store Control Registers	2	Note 4.
LCR	Load Control Registers	2	
CAM	Change Addressing Mode	1	

Table C-3 (cont). Instruction Timings for Models 2050A and 2070

Mnemonic Op Code	Instruction Name	Timing Formulas (in Memory Cycles)	Notes
Control Instructions (cont).			
CSM	Change Sequencing Mode	1	Note 3.
EXM	Extended Move	$2 \left( \left\lceil \frac{N_a}{4} \right\rceil \right)$	
MAT	Move and Translate	$2 \left( \left\lceil \frac{N_a}{4} \right\rceil \right) + N_a$	Note 4.
MIT	Move Item and Translate	V = 00: $2 \left( \left\lceil \frac{N_a}{4} \right\rceil \right) + N_a$ V = 01: $\left( \left\lceil \frac{N_a}{4} \right\rceil \right) + N_a + \left( \left\lceil \frac{N_a}{2} \right\rceil \right)$	Note 4.
MOS	Move or Scan	$3 \left( \left\lceil \frac{N_w}{4} \right\rceil \right)$	
TLU	Table Look-up	$n \left[ \left( \left\lceil \frac{N_a}{4} \right\rceil \right) + \left( \left\lceil \frac{N_b}{4} \right\rceil \right) \right]$	
LIB	Load Index/Barri- cade Register	2	
SIB	Store Index/Barri- cade Register	2	
Interrupt Control Instructions			
MC	Monitor Call	1	Not including the time used in honoring the interrupt signal. Note 10.
RVI	Restore Variant and Indicators	3	Typical (all indicators re-stored). Note 4.
SVI	Store Variant and Indicators	3	Typical (all indicators stored). Note 4.
RNM	Resume Normal Mode	1	Note 4.
Input/Output Instructions			
PCB	Peripheral Control and Branch	7	Notes 4, 9.
PDT	Peripheral Data Transfer	10	Notes 4, 9.

Table C-3 (cont). Instruction Timings for Models 2050A and 2070

Mnemonic Op Code	Instruction Name	Timing Formulas (in Memory Cycles)	Notes
Edit Instructions			
MCE	Edit	$\left(\left\lceil \frac{N_w}{4} \right\rceil\right) + 3 \left(\left\lceil \frac{N_b}{4} \right\rceil\right)$ $+ \left(\left\lceil \frac{ZS}{4} \right\rceil\right) + \left(\left\lceil \frac{\$}{4} \right\rceil\right)$	
<p><sup>1</sup> Subtract one memory cycle from this formula when the format is op code/A address.</p> <p><sup>2</sup> If the format is op code/A address, the formula is <math>2 \left(\left\lceil \frac{N_a}{4} \right\rceil\right)</math>.</p> <p><sup>3</sup> When item mark trapping is in effect, any item-marked op code is treated as CSM.</p> <p><sup>4</sup> This is a privileged instruction, used when storage protection is in effect.</p> <p><sup>5</sup> Instruction terminates prematurely if any table access finds a word mark.</p> <p><sup>6</sup> Assumes all multiplier digits are 5.</p> <p><sup>7</sup> Assumes all quotient digits will be 5.</p> <p><sup>8</sup> <math>P = 0.00556</math> (probability of odd back cycles) should be considered to = 0.</p> <p><sup>9</sup> Timing proceeds normally until the control character designating the read/write channel is extracted. Subsequent characters must be transferred to the I/O traffic control of the sector addressed in the instruction. This can occur only during the memory cycle set aside for that sector. Therefore, timings vary.</p> <p>If an external interrupt signal is received during a PDT in which the RWC of the peripheral control is busy, extraction of the PDT must begin again.</p> <p><sup>10</sup> The time used in honoring a peripheral interrupt, control panel or console interrupt, or monitor call interrupt is three processor-allocated memory cycles.</p>			



Table C-4. Timings for Decimal Multiply and Divide - Model 2040

Function	Timing (Memory Cycles)
Multiply	$N_i + 5 + 2N_a + 2Z_{ta} + 5N_{mr} - Z_{mr} + s(N_a - Z_{ta}) + 2(N_a - Z_{ta})(N_{mr} - Z_{mr})$
Divide	$N_i + 4 + 2N_a \text{ if divisor} = 0$ $N_i + 17.5 + 4.5N_a + 15.5Z_{la} + 12.5N_{dd} + 15N_a(N_{dd} - N_a + Z_{la}) \text{ if } (N_a - Z_{la}) \leq (N_{dd}) \text{ and divisor } \neq 0$ $N_i + 7 + 4N_a \text{ if } (N_a - Z_{la}) > (N_{dd})$

APPENDIX D  
SCIENTIFIC UNIT AND SCIENTIFIC SUBPROCESSOR

The scientific unit is available as Feature 1100A with the Type 2041, 2051, and 2061 processors, and as power module PM3A40 with the Type 2041A processor. The functionally-identical scientific subprocessor is standard with the Type 2071 processor and is available as power module PM3A50 with the Type 2051A processor. Both the scientific unit and the scientific subprocessor provide the following types of scientific instructions:<sup>1</sup>

1. Floating-point load and store.
2. Floating-point arithmetic.
3. Decimal-to-binary and binary-to-decimal conversion.
4. Floating-point test and branch.
5. Binary integer arithmetic.
6. Mantissa shift.

FLOATING-POINT DATA FORMAT

A floating-point number is represented by a fixed-length, 48-bit word. The high-order 36 bits contain a fraction, the mantissa. The low-order 12 bits contain an exponent of base 2. The value of a floating-point number is the product of the mantissa and 2 raised to the indicated exponent. As explained below, a Series 2000 floating-point word is capable of expressing numbers in the range  $\pm 2^{-2048}$  to  $\pm 2^{+2047}$ , or approximately  $\pm 10^{\pm 616}$ . In main memory, a floating-point word occupies a field of eight consecutive character positions, as shown in Figure D-1.

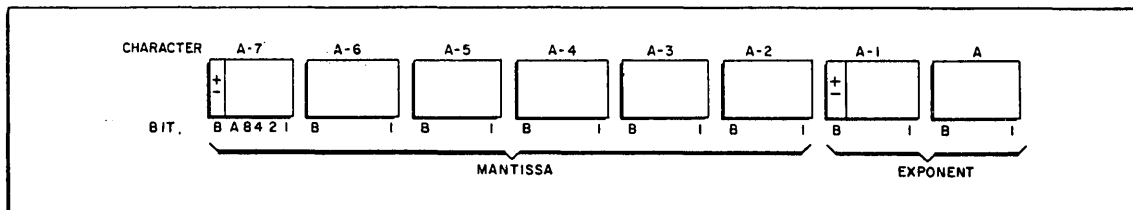


Figure D-1. Floating-Point Data Format in Main Memory

Four floating-point accumulators are reserved in control memory to contain operands and results of floating-point operations. The accumulators are explicitly addressed in the floating-point instructions by the octal digits 0, 1, 2, and 3. Each accumulator is composed of three specific, 18-bit, control memory registers, as explained below. Only the low-order 12 bits of

<sup>1</sup>None of these are interpreted by EasyCoder Assembler A, B, or C.



12-bit, binary integer whose high-order bit is 0. A negative exponent is a 12-bit, binary, twos-complement integer whose high-order bit, by definition, is 1.

Table D-1. Floating-Point Numerical Representation of Mantissas

Sign Bit	Implied Binary Point					Mantissa Value
Bit Position:	36	35	34-----2	1		
Bit Value:	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-34}$	$2^{-35}$	
0	1	1-----1	1		$+1 \cdot 2^{-35}$	
		⋮				
0	1	0-----0	0		$+1/2$	
0	0	1-----1	1		$+1/2 \cdot 2^{-35}$	
		⋮				
0	0	0-----0	1		$+2^{-35}$	
0	0	0-----0	0		$+0$	
1	1	1-----1	1		$-2^{-35}$	
		⋮				
1	1	0-----0	0		$-1/2$	
1	0	1-----1	1		$-(1/2 + 2^{-35})$	
		⋮				
1	0	0-----0	0		$-1$	

Table D-2. Floating-Point Numerical Representation of Exponents

Sign Bit	Implied Binary Point					Exponent Value
Bit Position:	12	11	10-----2	1		
Bit Value:	$2^{11}$	$2^{10}$	$2^9$	$2^1$	$2^0$	
0	1	1-----1	1		$+2047$	
		⋮				
0	0	0-----0	1		$+1$	
0	0	0-----0	0		$+0$	
1	1	1-----1	1		$-1$	
1	1	1-----1	0		$-2$	
		⋮				
1	0	0-----0	0		$-2048$	

Floating-point arithmetic instructions deliver results with normalized mantissas. For positive numbers, a normalized mantissa has a 1 immediately following the implied binary point (i. e., the high-order two bits are 01). For negative numbers, a normalized mantissa has a 0 immediately following the implied binary point (i. e., the high-order two bits are 10). In Table D-1, normalized mantissas are shaded. A normal 0 is defined as a floating-point word whose mantissa and exponent are both +0. An unnormalized zero is one whose mantissa is 0 but whose exponent is nonzero.

### FLOATING-POINT REGISTERS

The four addressable floating-point accumulators occupy the following locations in control memory:

Accumulator Address	Control Memory Address (Operator's Control Panel Only)		
	High-Order Mantissa	Low-Order Mantissa	Exponent
0	43	42	41
1	47	46	45
2	53	52	51
3	57	56	55

NOTE: In program instructions, the floating-point accumulators may be addressed only via the octal digits 0, 1, 2, and 3 in the floating-point instructions. The instructions LCR and SCR must not be used to address these accumulators. At the control panel, the operator may address these locations with the addresses in the above table.

A "pseudo-accumulator" is provided, which always contains a normal 0. The pseudo accumulator is addressed by the octal digit 7. Any floating-point number may be normalized by adding it to the normal 0 in accumulator 7. Note that the pseudo-accumulator should not be specified as the result location in any floating-point instruction, because the result data would be lost.

The scientific unit and subprocessor also include a low-order result register (LOR). The LOR may contain a low-order sum, difference, or product, or the remainder of a division operation. In effect, the LOR provides an additional 36 bits of mantissa precision. The LOR is not addressed explicitly in the floating-point arithmetic instructions, as are the accumulators. However, instructions are provided to load and store the contents of the LOR.

### INDICATORS

Three indicators are present in the scientific unit and subprocessor:

Exponent Overflow (EXO):	Activated when a base-2 exponent exceeds +2047. The correct mantissa and an exponent that is 4096 less than the correct exponent are delivered to the result accumulator.
	NOTE: If an exponent is less than -2048, a normal 0 is delivered automatically.
Divide Check (DVC):	Activated when a divisor is equal to zero. This indicator causes the division operation to be skipped and there is no accumulator alteration.
Multiply Overflow (MPO):	Activated when the product of a Binary Integer Multiply instruction exceeds 23 bits in length, 23 and the sign. Other high-order bits lost.

These indicators can be tested by a Floating Test and Branch on Indicator instruction. Indicators are reset to zero by this test or by system initialization. In addition, these indicators can be cleared and stored by an SVI instruction and restored by an RVI instruction. However, they do not initiate interrupts.

### AUTOMATIC FORMATTING IN ARITHMETIC OPERATIONS

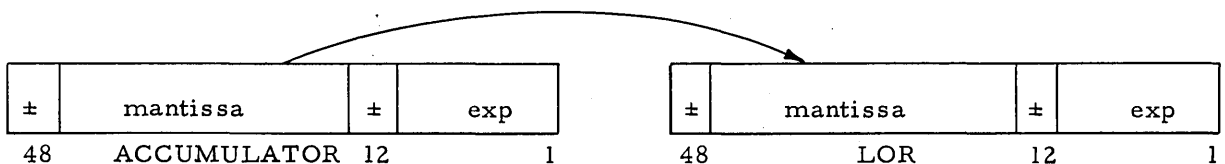
Floating-point arithmetic instructions accept either normalized or unnormalized operands. The scientific unit automatically shifts operands in order to perform arithmetic operations, and automatically normalizes results of arithmetic operations. The three types of automatic formatting are described below.

#### Prenormalization

In a floating divide operation, an unnormalized divisor is prenormalized. The mantissa is left-shifted until normalized, and the exponent is decreased by one for each bit position shifted. In the scientific subprocessor, the dividend is prenormalized. In the scientific unit, it is the programmer's responsibility (e.g., by a Floating Point Add to the zero accumulator) to normalize the dividend.

#### Equalization

In floating add and subtract operations, the mantissa of the operand with the smaller exponent is right-shifted, and the exponent is increased by one for each bit position shifted, until the exponents of the two operands are equal. Bits are shifted from the low-order mantissa position of the accumulator (bit 13) into the high-order mantissa position of the LOR (bit 47), as shown below.



## Postnormalization

The results of floating add, subtract, multiply, and divide operations are normalized. If the tentative result is unnormalized, the mantissa is left-shifted until normalized, and the exponent is decreased by one for each bit position shifted. For results in which mantissa overflow occurred, the mantissa is right-shifted one bit position and the exponent is increased by one. Note that postnormalization may restore bits that were shifted into the LOR by equalization.

## INSTRUCTION FORMATS

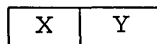
Only four operation codes are associated with the 14 scientific instructions. The Binary Mantissa Shift instruction has the mnemonic BMS (octal code 04). The Binary Integer Multiply instruction has the mnemonic BIM (octal code 05). All the remaining floating-point instructions use one or both of the following op codes:

<u>Name</u>	<u>Mnemonic</u>	<u>Octal Code</u>
Floating Memory to Accumulator	FMA	07
Floating Accumulator to Accumulator	FAA	06

The full formats of the floating-point instructions are given below:

	<u>OP CODE</u>	<u>A ADDRESS</u>	<u>B ADDRESS</u>	<u>VARIANT 1</u>	<u>VARIANT 2</u>
FMA:	■	■		■	■
FAA:	■			■	■

The first six-bit instruction variant usually addresses the floating-point accumulators used in an operation. In subsequent instruction description, this variant is abbreviated



where octal digits X and Y are the accumulator addresses given on Page D-4. The accumulator X addressed in the high-order three variant bits is usually the source of a floating-point operand. The accumulator Y addressed in the low-order three variant bits is usually the destination of a floating-point result. The second instruction variant is a six-bit octal character which defines the particular floating-point instruction (e. g. , Floating Multiply).

The memory-to-accumulator format is used in those instructions that require a main memory address in addition to floating-point accumulator references. In instruction descriptions, the A address of an instruction is abbreviated by the letter A. The A address may define the main memory location of an 8-character, floating-point operand, or it may specify a branch address. The accumulator-to-accumulator format is used in those instructions that require only floating-point accumulator references.

In addition to the full instruction formats described above, each form of a floating-point instruction using the FMA or FAA format is assigned a unique assembly-language mnemonic, which also generates the 07 or 06 octal op code. When an instruction is coded using its unique mnemonic, the second variant is automatically generated and is not written in the operands field by the programmer. In summary, the floating-point instructions may be coded in two equivalent forms:

1. The full form, which contains an FMA or FAA mnemonic op code, an A address if appropriate, and two variants.
2. The unique form, which contains a unique mnemonic op code, an A address if appropriate, and one variant.

Both forms are described for each instruction in the following sections.

### PROGRAMMING CONSIDERATIONS

For instructions in the FMA format, the A address is processed by the central processor in the usual manner, using the A-address register (AAR). The description of each instruction gives the address register settings after the operation. During instruction extraction, the two variants of FMA and FAA instructions are transmitted directly to the scientific unit or sub-processor. The variant register is unspecified following these instructions. In the extraction or restoration of operands in memory, the scientific unit or subprocessor neither recognizes nor alters punctuation bits.

### SYMBOLOLOGY FOR EXECUTION TIMINGS

A:	A address of the instruction.
B:	B address of the instruction.
X:	Floating-point accumulator addressed in the high-order three bits of an instruction variant (usually the source of an operand).
Y:	Floating-point accumulator addressed in the low-order three bits of an instruction variant (usually the destination of a result).
(A):	Floating-point word contained in the main memory field from location A through location A-7.
(X) or (Y):	Floating-point word contained in accumulator X or Y.
LOR:	Low-order result register.
(LOR):	Floating-point word contained in LOR.
A <sub>p</sub> :	Previous setting of A-address register.
B <sub>p</sub> :	Previous setting of B-address register.
D:	One if there is a two-bit overflow into LOR; otherwise zero.
JI:	Address of next instruction if branch occurs.
NXT:	Next sequential instruction.
N <sub>n</sub> :	Number of bit positions shifted for automatic formatting.
N <sub>1</sub> :	Number of binary ones in a multiplier.



$N_s$ :	Number of shifts.
[ ]	"Smallest integer greater than".
W:	Number of memory words used to store the data involved.
X-:	In the first variant of an instruction, only the high-order three bits specifying accumulator X are significant.
-Y:	In the first variant of an instruction, only the low-order three bits specifying accumulator Y are significant.
SP:	Single-precision.
DP:	Double-precision.
K:	Total number of one bits in the multiplier.
M:	Number of shifts (Binary Mantissa Shift instruction).
SR:	Sequence register.
N:	Number of prenormalization, postnormalization, and equalization steps.
$N_{add}$ :	Difference in values of X and Y exponents.
$N_{mult}$ :	Number of shifts needed to postnormalize the result.
$N_{div}$ :	Number of shifts needed to normalize the mantissa of the operand that requires the most shifts to normalize.
$N_{bms}$ :	Number of shifts specified in the instruction.

#### TIMING NOTES

Table D-3 specifies execution timing in memory cycles on the various Series 2000 central processors. Remember that total instruction time includes extraction as well as execution. Total timings in memory cycles can be determined by adding the following:

For the Type 2041: the number of characters in the instruction.

For the Types 2041A, 2051C, and 2061: the number of characters in the instruction divided by two (minimum), or the number of characters in the instruction divided by the two plus one (maximum), depending on the position of the first character of the instruction (in an odd or even memory module).

For the Types 2051A and 2071:  $\frac{L_I + Q}{4} + 1 + \text{the number of indexing operations} + \text{the number of indirect operations.}$

where  $L_I$  = the number of characters extracted

$Q = 1$  if 3-character mode FA format and A address is either indexed or indirect

$Q = 0$  in all other cases.

NOTE: If  $\frac{L_I + Q}{4}$

results in a fraction, it should be rounded up to the next larger whole number.

Table D-3. Execution Timings in Memory Cycles

Instruction	Mnemonic	2041	2041A, 2051C, 2061	2051A, 2071
Store Floating Accumulator:				
Memory-Accumulator	FMA	11	12	3
Accumulate-Accumulator	FAA	4	5	2
Load Floating Accumulator:				
Memory-Accumulator	FMA	11	12	3
Accumulate-Accumulator	FAA	4	5	2
Load Low-Order Result:				
Memory-Accumulator	FMA	10	10	3
Accumulate-Accumulator	FAA	3	3	2
Store Low-Order Result:				
Memory-Accumulator	FMA	10	11	3
Accumulate-Accumulator	FAA	3	4	1
Floating-Point Arithmetic Instructions				
Floating Add:				
Memory-Accumulator	FMA	$13 + \frac{N}{6}$	$13 + \frac{N}{4}$	$3 + \frac{N_{\text{add}}}{8}$
Accumulate-Accumulator	FAA	$7 + \frac{N}{6}$	$7 + \frac{N}{4}$	$2 + \frac{N_{\text{add}}}{8}$
Floating Subtract:				
Memory-Accumulator	FMA	$13 + \frac{N}{6}$	$13 + \frac{N}{4}$	$3 + \frac{N_{\text{add}}}{8}$
Accumulate-Accumulator	FAA	$7 + \frac{N}{6}$	$7 + \frac{N}{4}$	$2 + \frac{N_{\text{add}}}{8}$
Floating Multiply:				
Memory-Accumulator	FMA	$18 + \frac{K}{6} + \frac{N}{6}$	$21 + \frac{K}{4} + \frac{N}{4}$	$(8 \text{ to } 17) + \frac{N_{\text{mult}}}{8}$
Accumulate-Accumulator	FAA	$12 + \frac{K}{6} + \frac{N}{6}$	$16 + \frac{K}{4} + \frac{N}{4}$	$(7 \text{ to } 16) + \frac{N_{\text{mult}}}{8}$
Floating Divide:				
Memory-Accumulator	FMA	$25 + \frac{N}{6}$	$31 + \frac{N}{4}$	$(10 \text{ to } 14) + \frac{N_{\text{div}}}{8}$
Accumulate-Accumulator	FAA	$18 + \frac{N}{6}$	$26 + \frac{N}{4}$	$(8 \text{ to } 12) + \frac{N_{\text{div}}}{8}$

Table D-3 (cont). Execution Timings in Memory Cycles

Instruction	Mnemonic	2041	2041A, 2051C, 2061	2051A, 2071
Conversion Instructions				
Decimal-Binary	FMA	20 + D	24	5 + (1/4 times the number of leading zeros) + (1/2 times the number of nonleading zeros)
Binary-Decimal	FMA	21	24	13
Control Instructions				
Floating Test and Branch On Accumulator Condition:	FMA			
No Branch		3	3	1
Branch		4	5	2
Floating Test and Branch On Indicator:	FMA			
No Branch		2	2	1
Branch		3	4	2
Binary Mantissa Shift	BMS	$4 + \frac{M}{6}$	$5 + \frac{M}{4}$	$2 + \frac{N_{bms}}{8}$
Binary Integer Multiply	BIM	$20 + \frac{K}{6}$	$21 + \frac{K}{4}$	7 to 13

DATA MOVING INSTRUCTIONS

STORE FLOATING ACCUMULATOR

FORMAT

FMA: FMA/A, X-, 00 or TAM/A, X-  
 FAA: FAA/XY, 00

FUNCTION

FMA: (X) is stored in memory locations A through A-7.  
 (X) is unaltered.  
 FAA: (X) is stored in accumulator Y.

REGISTERS AFTER OPERATION

AAR	BAR
A-8	B <sub>p</sub>

NOTE:

1. No normalization occurs.

EXAMPLE

Store the contents of floating accumulator 1 in the main memory field whose rightmost character is tagged RESULT.

CARD NUMBER	Y P R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
1		FMA	RESULT, 10, 00	OR
2		TAM	RESULT, 10	

LOAD FLOATING ACCUMULATOR

FORMAT

FMA: FMA/A, -Y, 02 or TMA/A, -Y  
 FAA: FAA/XY, 02 or TAA/XY

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is loaded into accumulator Y.  
 FAA: (X) is loaded into accumulator Y.

REGISTERS AFTER OPERATION

	AAR	BAR
FMA:	A-8	B <sub>p</sub>
FAA:	A <sub>p</sub>	B <sub>p</sub>

NOTE

1. No normalization occurs.

EXAMPLES

1. Load the floating-point word stored in memory locations DELTA-7 through DELTA into floating accumulator 0.

CARD NUMBER	Y P R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
1		FMA	DELTA, 00, 02	OR
2		TMA	DELTA, 00	

2. Load the contents of accumulator 3 into accumulator 0.

CARD NUMBER	Y P R	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63 80
1		FAA	30, 02	OR
2		TAA	30	

STORE LOW-ORDER RESULT

FORMAT

FMA: FMA/A, 00, 07 or TLM/A  
 FAA: FAA/-Y, 07 or TLA/-Y

FUNCTION

FMA: (LOR) is stored in memory locations A through A-7.  
 FAA: (LOR) is stored in accumulator Y.

REGISTERS AFTER OPERATION

	AAR	BAR
FMA:	A-8	B <sub>p</sub>
FAA:	A <sub>p</sub>	B <sub>p</sub>

NOTE

1. No normalization occurs.

EXAMPLES

1. Store the contents of the LOR in the main memory field whose rightmost character is tagged RESULT.

CARD NUMBER	T V P R E K	M A R K	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8		14 15	20 21	62 63 80
1			FMA	00, 07	OR RESULT
2			TLM		RESULT

2. Store the contents of the LOR in accumulator 2.

CARD NUMBER	T V P R E K	M A R K	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8		14 15	20 21	62 63 80
1			FAA	02, 07	OR
2			TLA	02	

LOAD LOW-ORDER RESULT

FORMAT

FMA: FMA/A, 00, 01 or TML/A  
 FAA: FAA/X-, 01 or TAL/X-

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is loaded into the LOR.  
 FAA: (X) is loaded into the LOR.

REGISTERS AFTER OPERATION

	AAR	BAR
FMA:	A-8	B <sub>p</sub>
FAA:	A <sub>p</sub>	B <sub>p</sub>

NOTE

1. No normalization occurs.

EXAMPLES

1. Load the floating-point word stored in memory locations STORE-7 through STORE into the LOR.

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1	FMA	STORE, 00, 01	OR
2	TML	STORE	

2. Load the contents of accumulator 2 into the LOR.

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1	FAA	20, 01	OR
2	TAL	20	

FLOATING-POINT ARITHMETIC INSTRUCTIONS

FLOATING ADD

FORMAT

FMA: FMA/A, XY, 10 or AMA/A, XY  
 FAA: FAA/XY, 10 or AAA/XY

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is added to (X), and the sum is stored in accumulator Y. The low-order sum is stored in LOR.

FAA: (X) is added to (Y), and the sum is stored in accumulator Y. The low-order sum is stored in LOR.

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B <sub>P</sub>	The low-order result of the addition. The sign bit of LOR = 0. The exponent of LOR = the exponent of the high-order result minus 35.
FAA:	A <sub>p</sub>	B <sub>P</sub>	Same as above.

NOTES

1. Equalization, and postnormalization occur if required.
2. X and Y may specify the same accumulator.
3. An exponent overflow indication may be given.
4. A result with a zero mantissa is returned as a normal zero.

EXAMPLE

Add the three floating-point numbers stored in sequential fields beginning in location DATA. Store the sum in the eight-character field whose rightmost character is tagged SUM.

CARD NUMBER	OPERATION	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	FMA	DATA+7	01, 02	load first no. into accumulator 1
2	FMA	DATA+15	11, 10	add second no.
3	FMA	DATA+23	11, 10	add third no.
4	FMA	SUM	10, 00	store sum

**FLOATING SUBTRACT**

FORMAT

FMA: FMA/A, XY, 11 or SMA/A, XY  
 FAA: FAA/XY, 11 or SAA/XY

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is subtracted from (X); i. e., its twos complement is added to (X). The result is stored in accumulator Y. The low-order result is stored in the LOR.

FAA: (Y) is subtracted from (X). The result is stored in accumulator Y, and the low-order result is stored in the LOR.

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B <sub>p</sub>	Low-order difference. Sign bit = 0. Exponent = high-order exponent minus 35.
FAA:	A <sub>p</sub>	B <sub>p</sub>	Same as above.

NOTES

1. Equalization, and postnormalization occur if required.
2. X and Y may specify the same accumulator.
3. An exponent overflow indication may be given.
4. A result with a zero mantissa is returned as a normal zero.

EXAMPLE

1. Subtract the floating-point word in locations DATA-7 through DATA from the contents of accumulator 3 and store the result in accumulator 1.

CARD NUMBER	T M P R	LOCATION	OPERATION CODE	OPERANDS										
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
			FMA	DATA	31	11				OR				
2			SMA	DATA	31									

FLOATING MULTIPLY

FORMAT

FMA: FMA/A, XY, 13 or MAM/A, XY  
 FAA: FAA/XY, 13 or MAA/XY

FUNCTION

FMA: (X) is multiplied by the floating-point word in memory locations A through A-7. The high-order product is stored in accumulator Y. The low-order product is stored in LOR.  
 FAA: (X) is multiplied by (Y). The high-order product is stored in accumulator Y. The low-order product is stored in LOR.

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B <sub>p</sub>	Low-order product. Sign bit = 0. Exponent = high-order exponent minus 35.
FAA:	A <sub>p</sub>	B <sub>p</sub>	Same as above.



NOTES

1. X and Y may specify the same accumulator.
2. Postnormalization occurs if required.
3. An exponent overflow indication may be given.
4. If either operand is equal to zero, the results in both accumulator and LOR are normal zeros.

EXAMPLE

1. Multiply the floating-point word in accumulator 2 by the floating-point word in accumulator 0, and store the product in accumulator 0.

CARD NUMBER	Y	M	A	R	K	LOCATION	OPERATION CODE	OPERANDS						
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
							FAA	20	13	OR				
							MAA	20						

FLOATING DIVIDE

FORMAT

- FMA: FMA/A, XY, 12 or DMA/A, XY  
 FAA: FAA/XY, 12 or DAA/XY

FUNCTION

- FMA: The floating-point word in locations A through A-7 is divided by (X). The quotient is stored in accumulator Y. The remainder is stored in LOR.  
 FAA: (Y) is divided by (X). The quotient is stored in accumulator Y. The remainder is stored in LOR.

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B <sub>p</sub>	Contains the remainder. The absolute value of the remainder mantissa is less than the absolute value of the mantissa of the normalized divisor. The sign of the remainder is equal to the sign of the dividend. The exponent of the remainder is equal to the exponent of the dividend minus 35, and plus one if the absolute value of the dividend mantissa is greater than the absolute value of the mantissa of the normalized divisor.
FAA:	A <sub>p</sub>	B <sub>p</sub>	Same as above.



FUNCTION

The 11-character main memory field whose low-order character position is A (see Figure D-3) is treated as a signed decimal integer. That is, each character represents a decimal digit (see Table D-4). The sign of the integer is given by the zone bits of the units position (character A), as follows: 10 = negative; anything else = positive. The decimal integer is converted to a 36-bit binary integer and stored in the mantissa portion of (Y); the exponent of (Y) is set to +35.

REGISTERS AFTER OPERATION

AAR	BAR	LOR
A-11	B P	Low-order result of conversion (see note 2 below). Sign bit = 0. Exponent = high-order exponent minus 35.

NOTES

1. The zone bits of the 10 high-order decimal characters are ignored. If the middle two data bits of any character are 11, that character is interpreted as a zero. (Octal 12 and 13 are unspecified.)
2. Because an 11-digit decimal number has a range of  $\pm 99,999,999,999$  and a 36-bit binary twos-complement number has a range of approximately  $\pm 34,359,738,368$ , mantissa overflow of up to two bits is possible. If mantissa overflow occurs, the low-order one or two bits are shifted into LOR. Accumulator Y then contains the high-order result of conversion, with an exponent of 36 or 37. Note that when a low-order result is shifted into LOR, the high-order result is automatically normalized.\*

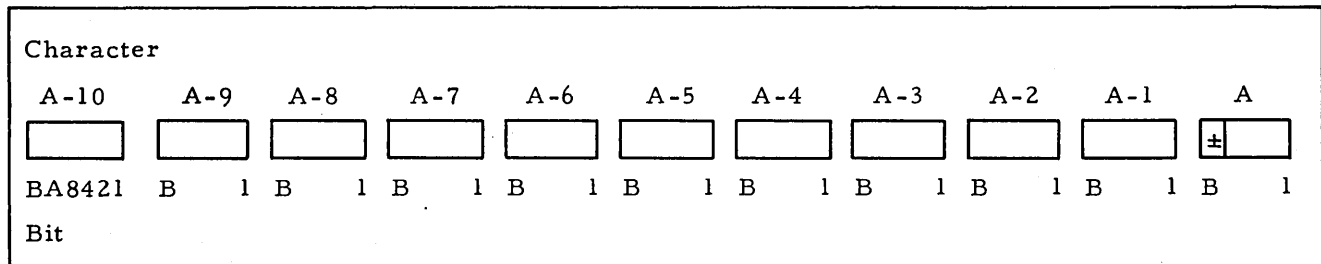


Figure D-3. Decimal Data Format in Main Memory

\*With two exceptions:

- (-34, 359, 738, 368) will translate to (110...0) with an exponent of 36.
- (-68, 719, 476, 736) will translate to (110...0) with an exponent of 37.

Table D-4. Numerical Representation of Decimal Word Data

Decimal Digit	A-10	.....	A-2	A-1	A		Value
					Sign		
Positive	001001	.....	001001	001001	01	1001	+99,999,999,999
	.	.....	.	.	.	.	.
	.	.....	.	.	.	.	.
	.	.....	.	.	.	.	.
	000000	.....	000000	000000	01	0010	+2
	000000	.....	000000	000000	01	0001	+1
	000000	.....	000000	000000	01	0000	+0
Negative	000000	.....	000000	000000	10	0000	-0
	000000	.....	000000	000000	10	0001	-1
	000000	.....	000000	000000	10	0010	-2
	.	.....	.	.	.	.	.
	.	.....	.	.	.	.	.
	001001	.....	001001	001001	10	1001	-99,999,999,999

EXAMPLE

Convert 899,473 to a binary integer in the mantissa portion of accumulator 0.

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1	DEC	DCW	+00000899473
2	FMA	DEC, 00, 03	

BINARY TO DECIMAL CONVERSION

FORMAT

FMA/A, X-, 06 or BTD/A, X-

FUNCTION

The mantissa portion of (X) is converted from a twos-complement binary integer to a signed decimal integer. The decimal integer is stored in the 11-character main memory field whose low-order character is location A.

REGISTERS AFTER OPERATION

AAR	BAR
A-11	B <sub>p</sub>

NOTES

1. If the binary integer is negative, the zone bits of the units character (location A) are set to 10. If the binary integer is positive, the zone bits of the units character are set to 01. The zone bits of the other 10 characters are set to 00.
2. The exponent is accumulator X is ignored and unaltered.

EXAMPLE

1. Convert the mantissa portion of the floating-point word in accumulator 3 to a signed decimal integer. Store the decimal integer in the main memory field whose rightmost character is tagged DEC.

CARD NUMBER	TYPE	M	P	R	LOCATION	OPERATION CODE	OPERANDS							
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					DEC	DCW	#110000000000000000000000							
						BTD	DEC 30							

CONTROL INSTRUCTIONS

FLOATING TEST AND BRANCH ON ACCUMULATOR CONDITION

FORMAT

FMA/A, XC, 04 or FBA/A, XC

FUNCTION

The mantissa portion of (X) is tested for the condition specified by C, the low-order octal digit of variant I:

- C = 0      no branch
- C = 1      (X) = 0
- C = 2      (X) < 0
- C = 3      (X) ≤ 0
- C = 4      (X) > 0
- C = 5      (X) ≥ 0
- C = 6      (X) ≠ 0
- C = 7      unconditional branch

If the condition specified by C is satisfied, program control branches to location A.

REGISTERS AFTER OPERATION

AAR	BAR	SR	
A	B p	NXT	NO BRANCH
A	NXT	JI(A)	BRANCH

NOTE

1. (X) must be normalized.

EXAMPLE

Subtract the floating-point word in accumulator 1 from the floating-point word in accumulator 0. If the difference is less than or equal to zero, branch to location LESS.

CARD NUMBER	TYPE	M	X	R	LOCATION	OPERATION CODE	OPERANDS							
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
1						SAA	01	floating subtract						
2						FBA	LESS, 13	test and branch						

FLOATING TEST AND BRANCH ON INDICATOR

FORMAT

FMA/A, 0D, 05 or FBI/A, 0D

FUNCTION

The indicator(s) specified by D, the low-order octal digit of variant 1, are tested. If any of the indicators is set, control branches to location A.

- D = 0 no branch
- D = 1 multiply overflow
- D = 2 exponent overflow
- D = 3 exponent overflow and multiply overflow
- D = 4 divide check
- D = 5 divide check and multiply overflow
- D = 6 Divide check and exponent overflow
- D = 7 divide check, exponent overflow, and multiply overflow

REGISTERS AFTER OPERATION

AAR	BAR	SR	
A	B p	NXT	NO BRANCH
A	NXT	JI(A)	BRANCH

NOTE

1. All indicators tested are reset.

EXAMPLE

Multiply the floating-point word in accumulator 1 by the floating-point word in accumulator 2. If exponent overflow occurs, store the contents of the sequence register and accumulator 2, replace the contents of accumulator 2 with the largest positive floating-point number, and continue.

CARD NUMBER	T V P R	M A R R	LOCATION	OPERATION CODE	OPERANDS	
					1 2 3 4 5 6 7 8	14 15 20 21 62 63 80
1			SEQREG	DCW	#4C 00000000	
2			ACC	DCW	#8C0000000000000000	
3			MAX	DCW	#8C37777777777777	
4				FAA	12, 13	floating multiply
5			TEST	FBI	OVER, 02	test for exponent overflow
6						
7						
8						
9			OVER	SCR	SEQREG, 77	store sequence register
10				FMA	ACC, 20, 00	store accumulator
11				FMA	MAX, 02, 02	load accumulator with max. value
12				B	TEST+7	return (in four-char. mode)

BINARY MANTISSA SHIFT

FORMAT

BMS/XM, V

FUNCTION

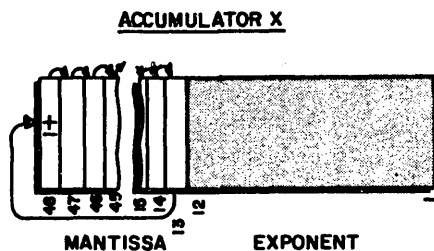
In a single-precision shift, the mantissa portion of (X) is shifted by the number of bit positions specified by variant 2 ( $0 \leq V \leq 63$ ). In a double-precision shift, the mantissa portions of (X) and (LOR) are treated as a single register and shifted the number of bit positions specified by variant 2. The exponent portions of (X) and (LOR) are never shifted. A shift operation may be of either the rotate or the arithmetic type, in the left or right direction. In a rotate shift, bits shifted off the end of a "register" (mantissa of X or mantissas of X and LOR) are moved end-around to the opposite end of the register. That is, no bits are lost in a rotate shift. In an arithmetic shift, bits shifted off the end of a register are lost. Note that in an arithmetic shift, the sign positions of accumulator X and LOR are protected; i. e., bits are shifted around these positions. In a right arithmetic shift, the sign bit is duplicated in the vacated bit positions. In a left arithmetic shift, vacated bit positions are filled with zeros.

M, the low-order octal digit of variant 1, specifies the mode of shifting, as illustrated as follows.

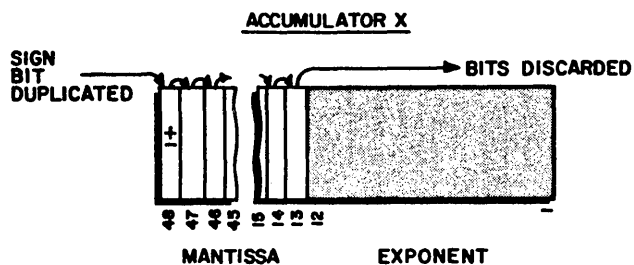




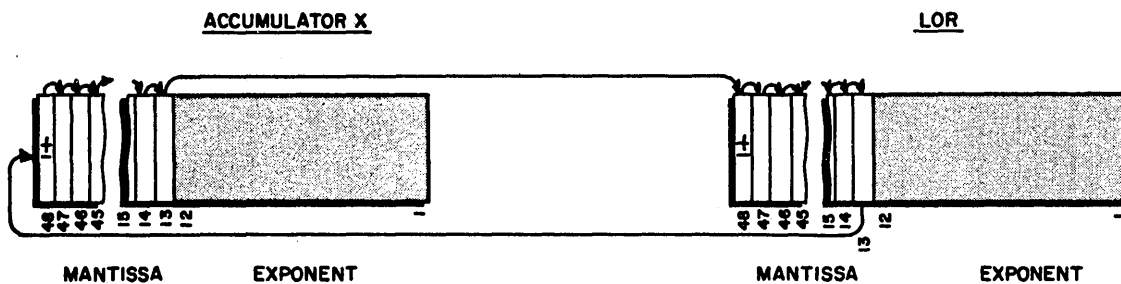
M=1: RIGHT, ROTATE, SINGLE-PRECISION SHIFT



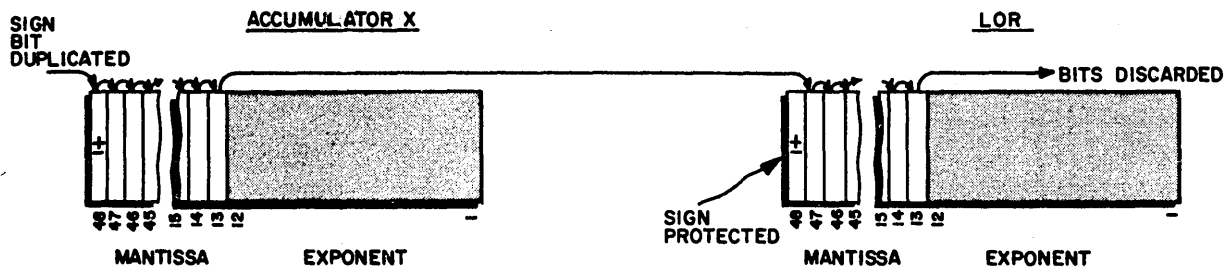
M=5: RIGHT, ARITHMETIC, SINGLE-PRECISION SHIFT



M=3: RIGHT, ROTATE, DOUBLE-PRECISION SHIFT



M=7: RIGHT, ARITHMETIC, DOUBLE-PRECISION SHIFT





NOTES

1. If the product exceeds 23 bits, a multiply overflow indication is given and the low-order 23 bits are delivered to the field specified by the B address. The 24th bit delivered is the proper sign bit. Any high-order bits are lost.
2. The product is not shifted in any way.

EXAMPLE

Multiply the binary equivalent of  $735_{10}$  by the binary equivalent of  $899_{10}$ .

CARD NUMBER				MARK	LOCATION	OPERATION CODE	OPERANDS																								
1	2	3	4					5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1					INT1	DCW	#4B735																								
2					INT2	DCW	#4B899																								
3					BIM	INT1, INT2	product is delivered to INT2																								

COMPUTER GENERATED INDEX

2040 INSTRUCTION SUMMARY - TIMING FORMULAS FOR MODELS 2040.  
C-4  
TIMINGS FOR DECIMAL MULTIPLY AND DIVIDE - MODEL 2040.  
C-16

2040A INSTRUCTION TIMINGS FOR MODELS 2040A, 2050, AND 2060.  
C-8  
MODEL 2040A POWER MODULES. 1-22

2041 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041 PROCESSOR.  
2-12

2041A MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041A PROCESSOR.  
2-13

2050 INSTRUCTION TIMINGS FOR MODELS 2040A, 2050, AND 2060.  
C-8

2050A INSTRUCTION TIMINGS FOR MODELS 2050A AND 2070. C-11  
MODEL 2050A POWER MODULES. 1-22

2051A MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051A PROCESSOR.  
2-14

2051C MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051C PROCESSOR.  
2-13

2060 INSTRUCTION TIMINGS FOR MODELS 2040A, 2050, AND 2060.  
C-8

2061 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2061 PROCESSOR.  
2-16

2070 INSTRUCTION TIMINGS FOR MODELS 2050A AND 2070. C-11

2071 MEMORY ACCESS DISTRIBUTION IN THE TYPE 2071 PROCESSOR.  
2-16  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2071 PROCESSOR.  
2-17

209 C3 CODING FOR TYPE 209 AND 209-2 PAPER TAPE READERS.  
8-136

209-2 C3 CODING FOR TYPE 209 AND 209-2 PAPER TAPE READERS.  
8-136

210 C3 CODING FOR TYPE 210 PAPER TAPE PUNCH. 8-137

222 C3 CODING FOR TYPE 222 PRINTERS. 8-137

243 FORMAT OF TYPE 243 PDT C3 VARIANT. 8-136

270A C3 CODING FOR TYPE 270A RANDOM ACCESS DRUM. 8-137

286 SUMMARY OF PCB I/O CONTROL CHARACTERS FOR TYPE 286  
MULTILINE COMMUNICATION CONTROLLER. 8-152  
SUMMARY OF PDT I/O CONTROL CHARACTERS FOR TYPE 286  
MULTILINE COMMUNICATION CONTROLLER. 8-138

286-1 TYPE 286-1, -2, -3 LINE CONTROL INSTRUCTIONS. 8-138

286-4 PCB CONTROL CHARACTERS C5 THROUGH C15 TYPE 286-4, -5, -6,  
-7 LINE CONTROL INSTRUCTIONS. 8-154

7-TRACK CHARACTER REPRESENTATION ON 7-TRACK MAGNETIC TAPE. 3-7

9-TRACK C4 VARIANT FOR 9-TRACK TAPE UNITS. 8-132

ABSOLUTE  
ABSOLUTE. 5-14  
CONVERSION OF SYMBOLIC TAG TO ABSOLUTE MEMORY ADDRESS.  
3-2

ACCESS  
C3 CODING FOR TYPE 270A RANDOM ACCESS DRUM. 8-137  
MEMORY ACCESS DISTRIBUTION IN THE TYPE 2071 PROCESSOR.  
2-16  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041 PROCESSOR.  
2-12  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041A PROCESSOR.  
2-13

ACCESS (CONT)  
2-13  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051A PROCESSOR.  
2-14  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051C PROCESSOR.  
2-13  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2061 PROCESSOR.  
2-16  
MEMORY ACCESS DISTRIBUTION OF THE TYPE 2071 PROCESSOR.  
2-17  
MEMORY ACCESS DISTRIBUTION. 2-10  
RANDOM ACCESS DRUMS. 1-15

ACCOUNTING  
ACCOUNTING TIMER REGISTER. 2-30

ACCUMULATOR  
FLOATING-POINT ACCUMULATOR DATA FORMAT. D-2

ACTIVE  
ACTIVE ADDRESS BITS IN SERIES 2000 SINGLE-CHARACTER  
PROCESSORS. 4-14

ACTIVITIES  
INPUT/OUTPUT TRAFFIC CONTROL ACTIVITIES. 2-10

ADD  
ADD. 8-15  
COMPLEMENT ADD EXAMPLES. 8-7  
EXTRACTION OF DATA FIELDS IN TYPICAL ADD INSTRUCTION.  
4-2  
SERIES 2000 ADD AND SUBTRACT OPERATIONS. 8-3  
TYPICAL ADD INSTRUCTION. 4-1

ADDRESS  
ACTIVE ADDRESS BITS IN SERIES 2000 SINGLE-CHARACTER  
PROCESSORS. 4-14  
ADDRESS ASSIGNMENTS AND UNIT LOCATIONS AVAILABLE IN SERIES  
2000 PROCESSORS. 1-8  
ADDRESS CODES. 5-14  
ADDRESS FIELD FORMAT. 3-13  
ADDRESS LITERALS. 5-22  
ADDRESS MODIFICATION CODES. 5-24  
ADDRESS MODIFICATION. 4-8  
ADDRESS REGISTERS. 2-5  
ADDRESSING MODES. 1-4 4-5  
ADDRESSING. 4-1  
ASSEMBLY OF INDEXED ADDRESS IN FOUR-CHARACTER ADDRESSING  
MODE. 5-25  
ASSEMBLY OF INDEXED ADDRESS IN THREE-CHARACTER ADDRESSING  
MODE. 5-25  
ASSEMBLY OF INDIRECT ADDRESS IN FOUR-CHARACTER ADDRESSING  
MODE. 5-26  
ASSEMBLY OF INDIRECT ADDRESS IN THREE-CHARACTER  
ADDRESSING MODE. 5-26  
CHANGING ADDRESSING MODES VIA CAM INSTRUCTION. 8-65  
CONVERSION OF SYMBOLIC TAG TO ABSOLUTE MEMORY ADDRESS.  
3-2  
DEFINE SYMBOLIC ADDRESS. 6-7  
DYNAMIC DISK ADDRESSING. 1-14  
DYNAMIC TAPE ADDRESSING. 1-11  
EXPLICIT ADDRESSING, IMPLICIT ADDRESSING, AND CHAINING.  
4-16  
EXTRACTION OF INDEXED ADDRESS IN THREE-CHARACTER MODE.  
4-12  
EXTRACTION OF INDIRECT AND INDEXED FOUR-CHARACTER  
ADDRESSES. 4-14  
EXTRACTION OF THREE-CHARACTER INDIRECT ADDRESS. 4-11  
FOUR-CHARACTER ADDRESS ASSEMBLY. 5-4  
FOUR-CHARACTER ADDRESSING MODE. 4-8  
FOUR-CHARACTER ADDRESS. 4-12  
INDEX REGISTER ADDRESSES IN FOUR-CHARACTER ADDRESSING  
MODE. 4-13  
INDEX REGISTER ADDRESSES IN THREE-CHARACTER ADDRESSING  
MODE. 4-11  
PERIPHERAL ADDRESSES AND UNIT LOCATIONS. 1-7  
POTENTIAL ADDRESSES OUTSIDE ADDRESS REGISTER RANGE.  
4-15  
POTENTIAL ADDRESSES WITHIN ADDRESS REGISTER RANGE. 4-15  
REGISTERS USED IN ADDRESSING. 4-3  
SET ADDRESS MODE. 7-12  
THREE-CHARACTER ADDRESS ASSEMBLY. 5-4  
THREE-CHARACTER ADDRESSING MODE. 4-6  
THREE-CHARACTER ADDRESS. 4-10  
TWO-CHARACTER ADDRESS ASSEMBLY. 5-3  
TWO-CHARACTER ADDRESSING MODE. 4-5

A- AND B-ADDRESSES  
A- AND B-ADDRESSES. 3-2

A-ADDRESS REGISTER  
A-ADDRESS REGISTER. 4-4

A-FIELD  
A-FIELD AND B-FIELD IN MULTIPLY OPERATION. 8-5  
LOAD CHARACTERS TO A-FIELD WORD MARK. 8-56

ALGEBRAIC  
ALGEBRAIC SIGNS IN DECIMAL ADDITION. 8-6

ALPHANUMERIC  
ALPHANUMERIC CONSTANTS. 6-4  
ALPHANUMERIC LITERALS. 5-20

COMPUTER GENERATED INDEX

ANGULAR  
 ANGULAR POSITION INDICATOR, 1-15

AREA  
 DATA AREA FORMAT, 3-14  
 DEFINE AREA, 6-7  
 RESERVE AREA, 6-6

AREA DEFINING  
 AREA DEFINING LITERALS, 5-21

ARITHMETIC  
 ARITHMETIC OPERATIONS, 8-3  
 ARITHMETIC UNIT, 2-8  
 ARITHMETIC, 8-14  
 AUTOMATIC FORMATTING IN ARITHMETIC OPERATIONS, D-5  
 BINARY INTEGER ARITHMETIC INSTRUCTION, D-25  
 DATA FLOW BETWEEN MAIN MEMORY AND ARITHMETIC UNIT, 2-8  
 DECIMAL ARITHMETIC SIGN CONVENTIONS, 8-8  
 FLOATING-POINT ARITHMETIC INSTRUCTIONS, D-13

ASSEMBLER  
 RELATIONSHIP OF SOURCE PROGRAM, ASSEMBLER, AND OBJECT PROGRAM, 5-2

ASSEMBLERS  
 THE ASSEMBLERS, 5-3

ASSEMBLY  
 ASSEMBLY CONTROL STATEMENTS, 7-1  
 ASSEMBLY OF INDEXED ADDRESS IN FOUR-CHARACTER ADDRESSING MODE, 5-25  
 ASSEMBLY OF INDEXED ADDRESS IN THREE-CHARACTER ADDRESSING MODE, 5-25  
 ASSEMBLY OF INDIRECT ADDRESS IN FOUR-CHARACTER ADDRESSING MODE, 5-26  
 ASSEMBLY OF INDIRECT ADDRESS IN THREE-CHARACTER ADDRESSING MODE, 5-26  
 FOUR-CHARACTER ADDRESS ASSEMBLY, 5-4  
 THREE-CHARACTER ADDRESS ASSEMBLY, 5-4  
 TWO-CHARACTER ADDRESS ASSEMBLY, 5-3

ASSIGNMENT  
 ADDRESS ASSIGNMENTS AND UNIT LOADS AVAILABLE IN SERIES 2000 PROCESSORS, 1-8  
 DESCRIPTION OF PDI I/O CONTROL CHARACTER C1 (RWC ASSIGNMENT), 8-117  
 SELECTING RWC ASSIGNMENTS FOR USE IN PDI INSTRUCTIONS, 8-112

BARRICADE  
 CORRESPONDENCE BETWEEN LIB SETTING AND BARRICADE LOCATION, 8-81

BASE  
 STORAGE PROTECTION WITH BASE RELOCATION, 2-26 8-80

BCC  
 BCC TEST CONDITIONS, 8-41  
 BRANCH ON CHARACTER CONDITION (BCC) CONDITIONS, B-5

BCT  
 BCT INSTRUCTION VARIANT CHARACTERS, 8-38  
 BRANCH ON CONDITION TEST (BCT) INDICATOR CONDITIONS, B-4  
 BRANCH ON CONDITION TEST (BCT) SENSE SWITCH CONDITIONS, B-3  
 INDICATOR TEST CONDITIONS FOR BCT INSTRUCTION, 8-37  
 PRIVILEGED BCT INSTRUCTION, 2-25  
 SENSE SWITCH TEST CONDITIONS FOR BCT INSTRUCTION, 8-36

BINARY  
 BINARY ADDITION, 8-3  
 BINARY CONSTANTS, 6-2  
 BINARY INTEGER ARITHMETIC INSTRUCTION, D-25  
 BINARY LITERALS, 5-19  
 BINARY SUBTRACTION, 8-3  
 BINARY, OCTAL, AND DECIMAL EQUIVALENTS, 8-8

BINARY ADD  
 BINARY ADD, 8-18

BINARY SUBTRACT  
 BINARY SUBTRACT, 8-19

BINARY-OCTAL  
 BINARY-OCTAL EQUIVALENTS, A-1

BITS  
 ACTIVE ADDRESS BITS IN SERIES 2000 SINGLE-CHARACTER PROCESSORS, 4-14

B-ADDRESS REGISTER  
 B-ADDRESS REGISTER, 4-4

B-FIELD  
 A-FIELD AND B-FIELD IN MULTIPLY OPERATION, 8-9

BLANK  
 BLANK CONSTANTS, 6-4  
 BLANK, 5-17

BLOCKS  
 RELATIONSHIP BETWEEN ITEMS, RECORDS, AND BLOCKS, 3-11

BRANCH  
 BRANCH ON CHARACTER CONDITION (BCC) CONDITIONS, B-5  
 BRANCH ON CONDITION TEST (BCT) INDICATOR CONDITIONS, B-4  
 BRANCH ON CONDITION TEST (BCT) SENSE SWITCH CONDITIONS, B-3  
 BRANCH, 8-34  
 PERIPHERAL CONTROL AND BRANCH, 8-139

BRANCH IF CHARACTER EQUAL  
 BRANCH IF CHARACTER EQUAL, 8-42

BRANCH ON BIT EQUAL  
 BRANCH ON BIT EQUAL, 8-44

BRANCH ON CHARACTER CONDITION  
 BRANCH ON CHARACTER CONDITION, 8-39

BRANCH ON CONDITION TEST  
 BRANCH ON CONDITION TEST, 8-35

BUFFER  
 PRINT BUFFER, 1-10

BUFFERED  
 BUFFERED MODE, 2-18  
 BUFFERED SECTOR OPERATION, 2-18  
 BUFFERED SECTOR RESTRICTIONS, 2-19  
 BUFFERED SECTORS, 2-18  
 CONTROLS/DEVICES CONNECTABLE TO BUFFERED SECTORS, 2-20

BUSY  
 TESTING PERIPHERAL CONTROL UNIT BUSY STATUS, 2-19

CAM  
 CHANGING ADDRESSING MODES VIA CAM INSTRUCTION, 8-65  
 MODES SPECIFIED BY VARIANT CHARACTER IN CAM INSTRUCTION, 8-63

CAPABILITY  
 EIGHT-BIT TRANSFER CAPABILITY, 2-28  
 WRITE PROTECT CAPABILITY, 1-14

CAPACITY  
 MINIMUM RWC CAPACITY REQUIREMENTS FOR SERIES 200/2000 PERIPHERAL DEVICES, 8-113

CARD  
 DATA PATH DURING CARD READ OPERATION, 1-8  
 PUNCHED CARD EQUIPMENT, 1-9  
 PUNCHED CARD FORMAT, 3-9

CARD NUMBER  
 CARD NUMBER, 5-5

CHAINING  
 EXPLICIT ADDRESSING, IMPLICIT ADDRESSING, AND CHAINING, 4-16

CHANGE ADDRESSING MODE  
 CHANGE ADDRESSING MODE, 8-62

CHANGE SEQUENCE REGISTER  
 CHANGE SEQUENCE REGISTER, 4-3

CHANGE SEQUENCING MODE  
 CHANGE SEQUENCING MODE, 8-66

CHANNEL  
 INTERLOCKING READ/WRITE CHANNELS, 2-17  
 READ/WRITE CHANNEL, 1-8  
 VARIABLE-SPEED READ/WRITE CHANNELS, 2-17

CHARACTER  
 BRANCH ON CHARACTER CONDITION (BCC) CONDITIONS, B-5  
 CHARACTER REPRESENTATION ON 7-TRACK MAGNETIC TAPE, 3-7  
 DESCRIPTION OF PDI I/O CONTROL CHARACTER C1 (RWC ASSIGNMENT), 8-117  
 DESCRIPTION OF PDI I/O CONTROL CHARACTER C2 (PERIPHERAL), 8-129  
 FLAG CHARACTER FORMAT, 3-13  
 LIB VARIANT CHARACTER, 8-81  
 MODES SPECIFIED BY VARIANT CHARACTER IN CAM INSTRUCTION, 8-63  
 SERIES 2000 CHARACTER CODES, B-7  
 VARIANT CHARACTER, 3-3 5-23

CHARACTERISTICS  
 CLOCK CHARACTERISTICS, 2-30

CHARACTERS  
 BCT INSTRUCTION VARIANT CHARACTERS, 8-38  
 INPUT/OUTPUT CONTROL CHARACTERS, 5-23  
 LOAD CHARACTERS TO A-FIELD WORD MARK, 8-56  
 MOVE CHARACTERS AND EDIT, 8-106  
 MOVE CHARACTERS TO WORD MARK, 8-55  
 REPRESENTATION OF CHARACTERS IN MAGNETIC CORE STORAGE, 2-2  
 SPECIAL CHARACTERS IN MCE INSTRUCTION, 8-107  
 SUMMARY OF PDI I/O CONTROL CHARACTERS, 8-133

CLEAR  
 CLEAR, 7-21

CLEAR ITEM  
 CLEAR ITEM MARK, 8-51

CLEAR WORD  
 CLEAR WORD MARK, 8-50

CLOCK  
 CLOCK CHARACTERISTICS, 2-30  
 HIGH-RESOLUTION CLOCK ALLCN, 2-31  
 HIGH-RESOLUTION CLOCK, 1-21 2-30

CODE  
 ADDITIONAL CODING RULES, 5-14  
 ADDRESS CODES, 5-14  
 ADDRESS MODIFICATION CODES, 5-24  
 C3 CODING FOR TYPE 209 AND 209-2 PAPER TAPE READERS, 8-136  
 C3 CODING FOR TYPE 210 PAPER TAPE PUNCH, 8-137  
 C3 CODING FOR TYPE 222 PRINTERS, 8-137  
 C3 CODING FOR TYPE 270A RANDOM ACCESS DRUM, 8-137  
 EASYCODER CODING FORM, 5-5

COMPUTER GENERATED INDEX

CODE (CCNT)  
 EBCDIC CODE TRANSLATION. 1-12  
 ESCAPE CODES. 2-20 8-129  
 OPERATION CODE. 3-2 5-12  
 PUNCHED CARD CODES. 3-9  
 SAMPLE CODING FOR EXTERNAL INTERRUPT ROUTINE. 2-34  
 SAMPLE CODING FOR INTERNAL INTERRUPT ROUTINE. 2-35  
 SERIES 2000 CHARACTER CODES. 8-7

COMMUNICATION  
 CUSTOMER INQUIRY HANDLING VIA TYPICAL COMMUNICATIONS NETWORK. 1-19  
 DATA COMMUNICATION EQUIPMENT. 1-16 1-17  
 SUMMARY OF PCB I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-152  
 SUMMARY OF PDT I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-138

COMPARE  
 COMPARE. 8-32

COMPATIBILITY  
 IBM MAGNETIC TAPE COMPATIBILITY. 1-12

COMPLEMENT  
 COMPLEMENT ADD EXAMPLES. 8-7

COMPLEMENT ADD  
 COMPLEMENT ADD. 8-6

COMPONENTS  
 SERIES 2000 COMPONENTS. 1-1

CONCEPTS  
 BASIC CONCEPTS. 4-1

CONSECUTIVE  
 CONSECUTIVE STORAGE LOCATIONS IN MAIN MEMORY. 3-4

CONSIDERATIONS  
 PROGRAMMING CONSIDERATIONS. 2-15 D-7

CONSOLE  
 CONSOLE EQUIPMENT. 1-18  
 CONSOLES. 1-17 1-2

CONSTANT  
 ALPHANUMERIC CONSTANTS. 6-4  
 BINARY CONSTANTS. 6-2  
 BLANK CONSTANTS. 6-4  
 DECIMAL CONSTANTS. 6-2  
 DEFINE CONSTANT WITH WORD MARK. 6-2  
 DEFINE CONSTANT. 6-5  
 FLOATING-POINT CONSTANTS. 6-5  
 NUMERIC CONSTANTS. 6-2  
 OCTAL CONSTANTS. 6-3

CONTENTS  
 CONTROL REGISTER CONTENTS LOADED BY LCR INSTRUCTION. 8-61  
 CONTROL REGISTER CONTENTS STORED BY SCR INSTRUCTION. 8-58

CONTROL  
 ASSEMBLY CONTROL STATEMENTS. 7-1  
 CONTROL DESIGNATION). 8-129  
 CONTROL EQUALS. 7-14  
 CONTROL INSTRUCTIONS. D-19  
 CONTROL MEMORY REGISTERS. 2-6  
 CONTROL REGISTER CONTENTS LOADED BY LCR INSTRUCTION. 8-61  
 CONTROL REGISTER CONTENTS STORED BY SCR INSTRUCTION. 8-58  
 CONTROL REGISTER DESIGNATIONS. E-1  
 CONTROL REGISTERS STORED BY SCR INSTRUCTION. 8-59  
 CONTROL UNIT. 2-9  
 CONTROL. 8-47  
 DESCRIPTION OF PDT I/O CONTROL CHARACTER C1 (RWC ASSIGNMENT). 8-117  
 DESCRIPTION OF PDT I/O CONTROL CHARACTER C2 (PERIPHERAL. 8-129  
 INPUT/OUTPUT CONTROL CHARACTERS. 5-23  
 INPUT/OUTPUT CONTROL OPERATIONS. 8-112  
 INPUT/OUTPUT TRAFFIC CONTROL ACTIVITIES. 2-10  
 INPUT/OUTPUT TRAFFIC CONTROL. 2-9  
 INTERRUPT CONTROL. 8-93  
 INTERRUPT SIGNAL GENERATED BY PERIPHERAL CONTROL. 2-36  
 LOGICAL DECISION PERFORMED BY INPUT/OUTPUT TRAFFIC CONTROL. 2-12  
 PCB CONTROL CHARACTERS C5 THROUGH C15 TYPE 286-4, -5, -6, -7 LINE CONTROL INSTRUCTIONS. 8-154  
 PERIPHERAL CONTROL AND BRANCH. 8-139  
 PERIPHERAL CONTROL INTERRUPT. 2-35  
 PERIPHERAL CONTROL. 1-6  
 SIZE OF CONTROL MEMORY REGISTERS. 2-4  
 SUMMARY OF INTERRUPT/ALLOW FUNCTION CONTROL AND TEST OPERATIONS. 2-37  
 SUMMARY OF PDT I/O CONTROL CHARACTERS. 8-133  
 TEST AND CONTROL OPERATIONS. 8-140  
 TESTING PERIPHERAL CONTROL UNIT BUSY STATUS. 2-19  
 TYPE 286-1, -2, -3 LINE CONTROL INSTRUCTIONS. 8-138  
 TYPICAL CONTROL REGISTER FUNCTION. 2-4

CONTROL CHARACTERS  
 PCB CONTROL CHARACTERS C5 THROUGH C15 TYPE 286-4, -5, -6, -7 LINE CONTROL INSTRUCTIONS. 8-154

CONTROL CHARACTERS (CONT)  
 SUMMARY OF PCB I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-152  
 SUMMARY OF PCB I/O CONTROL CHARACTERS. 8-142  
 SUMMARY OF PDT I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-138

CONTROLLER  
 SUMMARY OF PCB I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-152  
 SUMMARY OF PDT I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-138

CONTROLS/DEVICES  
 CONTROLS/DEVICES CONNECTABLE TO BUFFERED SECTORS. 2-20

CONVENTIONS  
 DATA CONVENTIONS OF HONEYWELL MASS-STORAGE DISK DEVICES. 3-12  
 DATA CONVENTIONS. 3-10  
 DECIMAL ARITHMETIC SIGN CONVENTIONS. 8-8  
 DIVIDE SIGN CONVENTIONS. 8-12  
 MULTIPLY SIGN CONVENTIONS. 8-9

CONVERSION  
 CONVERSION OF SYMBOLIC TAG TO ABSOLUTE MEMORY ADDRESS. 3-2  
 DATA CONVERSION INSTRUCTIONS. C-17  
 DECIMAL-OCTAL CONVERSION TABLE. A-2  
 OCTAL-DECIMAL CONVERSION PROCEDURE. A-3

CORE  
 REPRESENTATION OF CHARACTERS IN MAGNETIC CORE STORAGE. 2-2

COUNTERS  
 READ/WRITE COUNTERS. 2-5

CYCLE  
 EXECUTION TIMINGS IN MEMORY CYCLES. D-9  
 MEMORY CYCLE. 2-3

DATA  
 BASIC INPUT/OUTPUT DATA PATH. 1-7  
 DATA AREA FORMAT. 3-14  
 DATA COMMUNICATION EQUIPMENT. 1-16 1-17  
 DATA CONVENTIONS OF HONEYWELL MASS-STORAGE DISK DEVICES. 3-12  
 DATA CONVENTIONS. 3-10  
 DATA CONVERSION INSTRUCTIONS. C-17  
 DATA FIELD FORMAT IN MAIN MEMORY. 3-5  
 DATA FLOW BETWEEN MAIN MEMORY AND ARITHMETIC UNIT. 2-8  
 DATA FORMAT ON MAGNETIC TAPE. 3-8  
 DATA FORMAT. 3-1  
 DATA FORMATTING STATEMENTS. 6-1  
 DATA MOVING INSTRUCTIONS. D-10  
 DATA PATH DURING CARD READ OPERATION. 1-8  
 DATA TRANSFER INTERVALS DURING ONE PERIPHERAL OPERATION. 2-10  
 DATA TRANSFER RATES. 2-9  
 DECIMAL DATA FORMAT IN MAIN MEMORY. D-18  
 EXTRACTION OF DATA FIELDS IN TYPICAL ADD INSTRUCTION. 4-2  
 FLOATING-POINT ACCUMULATOR DATA FORMAT. D-2  
 FLOATING-POINT DATA FORMAT IN MAIN MEMORY. D-1  
 MAGNETIC TAPE DATA FORMAT. 3-7  
 NUMERICAL REPRESENTATION OF DECIMAL WORD DATA. D-19  
 ORGANIZATION OF DATA IN MAIN MEMORY. 3-4  
 PERIPHERAL DATA TRANSFER OPERATION. 1-6  
 PERIPHERAL DATA TRANSFER. 8-116  
 SUMMARY OF INTERNAL DATA FORMAT. 3-6

DECIMAL  
 ALGEBRAIC SIGNS IN DECIMAL ADDITION. 8-6  
 BINARY, OCTAL, AND DECIMAL EQUIVALENTS. 8-8  
 DECIMAL ADDITION. 8-6  
 DECIMAL ARITHMETIC SIGN CONVENTIONS. 8-8  
 DECIMAL CONSTANTS. 6-2  
 DECIMAL DATA FORMAT IN MAIN MEMORY. D-18  
 DECIMAL LITERALS. 5-16  
 DECIMAL SUBTRACTION. 8-7  
 NUMERICAL REPRESENTATION OF DECIMAL WORD DATA. D-19  
 TIMINGS FOR DECIMAL MULTIPLY AND DIVIDE - MODEL 2040. C-16

DECIMAL-OCTAL  
 DECIMAL-OCTAL CONVERSION TABLE. A-2

DECISION  
 LOGICAL DECISION PERFORMED BY INPUT/OUTPUT TRAFFIC CONTROL. 2-12

DENSITY  
 1200-BPI RECORDING DENSITY. 1-10  
 1600-BPI RECORDING DENSITY. 1-10

DESCRIPTION  
 DESCRIPTION OF PDT I/O CONTROL CHARACTER C1 (RWC ASSIGNMENT). 8-117  
 DESCRIPTION OF PDT I/O CONTROL CHARACTER C2 (PERIPHERAL. 8-129  
 SYMBOLOGY USED IN SERIES 2000 INSTRUCTION DESCRIPTIONS. 8-2

DESIGNATION  
 CONTROL DESIGNATION). 8-129

COMPUTER GENERATED INDEX

DESIGNATION (CONT)  
 CONTROL REGISTER DESIGNATIONS. E-1

DEVICES  
 DATA CONVENTIONS OF HONEYWELL MASS-STORAGE DISK DEVICES. 3-12  
 MINIMUM R/W CAPACITY REQUIREMENTS FOR SERIES 200/2000 PERIPHERAL DEVICES. 8-113  
 VISUAL INFORMATION PROJECTION DEVICES. 1-18

DIRECT-ACCESS  
 DIRECT-ACCESS MODE. 2-19

DISK  
 DATA CONVENTIONS OF HONEYWELL MASS-STORAGE DISK DEVICES. 3-12  
 DISK FORMAT. 3-10  
 DISK PACK DRIVE FEATURES. 1-14  
 DISK PACK DRIVES AND DISK SUBSYSTEMS. 1-13  
 DISK PACK DRIVES. 1-12  
 DYNAMIC DISK ADDRESSING. 1-14  
 HIGH-SPEED DISK FILE. 1-15 1-6

DISTRIBUTION  
 MEMORY ACCESS DISTRIBUTION IN THE TYPE 2071 PROCESSOR. 2-16  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041 PROCESSOR. 2-12  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041A PROCESSOR. 2-13  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051A PROCESSOR. 2-14  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051C PROCESSOR. 2-13  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2061 PROCESSOR. 2-16  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2071 PROCESSOR. 2-17  
 MEMORY ACCESS DISTRIBUTION. 2-10

DIVIDE  
 DIVIDE SIGN CONVENTIONS. 8-12  
 DIVIDE. 8-25  
 FACTOR LOCATIONS IN DIVIDE OPERATION. 8-11  
 TIMINGS FOR DECIMAL MULTIPLY AND DIVIDE - MODEL 2040. C-16

DIVISION  
 DIVISION. 8-10  
 LOGICAL DIVISION OF SERIES 2000 CENTRAL PROCESSOR. 2-1

DRIVE  
 DISK PACK DRIVE FEATURES. 1-14  
 DISK PACK DRIVES AND DISK SUBSYSTEMS. 1-13  
 DISK PACK DRIVES. 1-12

DRUM  
 C3 CODING FOR TYPE 270A RANDOM ACCESS DRUM. 8-137  
 RANDOM ACCESS DRUMS. 1-15

DUMP  
 MEMORY DUMP. 7-15

EASycODER  
 EASycODER CODING FORM. 5-5  
 EASycODER C, D, AND OS/2000 OPTIONS. 6-10  
 EASycODER PROGRAMMING. 5-1  
 SET II FUNCTION INDICATORS (EASycODER C, D, AND OS/2000). 5-8

EBCDIC  
 EBCDIC CODE TRANSLATION. 1-12

EDIT  
 EDIT INSTRUCTION. 1-20  
 EDITING. 8-105  
 MOVE CHARACTERS AND EDIT. 8-106

END  
 END. 7-22

EQUALIZATION  
 EQUALIZATION. D-5

EQUALS  
 CONTROL EQUALS. 7-14  
 EQUALS. 7-13

EQUIPMENT  
 CONSOLE EQUIPMENT. 1-18  
 DATA COMMUNICATION EQUIPMENT. 1-16 1-17  
 PAPER TAPE EQUIPMENT. 1-16  
 PERIPHERAL EQUIPMENT. 1-9  
 PUNCHED CARD EQUIPMENT. 1-9  
 TELLER TERMINAL EQUIPMENT. 1-20

EQUIVALENTS  
 BINARY-OCTAL EQUIVALENTS. A-1  
 BINARY, OCTAL, AND DECIMAL EQUIVALENTS. B-8

ESCAPE  
 ESCAPE CODES. 2-20 8-129

ESCAPE CODE  
 ESCAPE CODE. 8-128

EXAMPLES  
 COMPLEMENT ADD EXAMPLES. 8-7

EXECUTION  
 EXECUTION TIMINGS IN MEMORY CYCLES. D-9  
 SYMBOLIC FOR EXECUTION TIMING. D-7

EXM  
 EXTENDED MOVE (EXM) CONDITIONS. E-2

EXPANDED  
 EXPANDED INSTRUCTION PACKAGE. 1-22

EXPLICIT  
 EXPLICIT ADDRESSING, IMPLICIT ADDRESSING, AND CHAINING. 4-16

EXPONENTS  
 FLOATING-POINT NUMERICAL REPRESENTATION OF EXPONENTS. D-3

EXTERNAL  
 EXTERNAL INTERRUPT MASKING. 2-27  
 EXTERNAL INTERRUPT MODE. 2-30  
 EXTERNAL INTERRUPT. 2-31  
 SAMPLE CODING FOR EXTERNAL INTERRUPT ROUTINE. 2-34

EXTERNAL INTERRUPT REGISTER  
 EXTERNAL INTERRUPT REGISTER. 4-3

EXTRACT  
 EXTRACT. 8-28

EXTRACTION  
 EXTRACTION OF DATA FIELDS IN TYPICAL ADD INSTRUCTION. 4-2  
 EXTRACTION OF INDEXED ADDRESS IN THREE-CHARACTER MODE. 4-12  
 EXTRACTION OF INDIRECT AND INDEXED FOUR-CHARACTER ADDRESSES. 4-14  
 EXTRACTION OF THREE-CHARACTER INDIRECT ADDRESS. 4-11

FACTOR  
 FACTOR LOCATIONS IN DIVIDE OPERATION. 8-11

FEATURE  
 DISK PACK DRIVE FEATURES. 1-14  
 FEATURES AND POWER MODULES. 1-20  
 STORAGE PROTECTION FEATURE. 2-21

FIELD  
 ADDRESS FIELD FORMAT. 3-13  
 DATA FIELD FORMAT IN MAIN MEMORY. 3-5  
 EXTRACTION OF DATA FIELDS IN TYPICAL ADD INSTRUCTION. 4-2  
 FIELDS. 3-4  
 VARIABLE FIELD LENGTH. 3-1

FILE  
 HIGH-SPEED DISK FILE. 1-15 1-6

FLAG  
 FLAG CHARACTER FORMAT. 3-13

FLOATING-POINT  
 FLOATING-POINT ACCUMULATOR DATA FORMAT. D-2  
 FLOATING-POINT ARITHMETIC INSTRUCTIONS. D-13  
 FLOATING-POINT CONSTANTS. 6-5  
 FLOATING-POINT DATA FORMAT IN MAIN MEMORY. D-1  
 FLOATING-POINT NUMERICAL REPRESENTATION OF EXPONENTS. D-3  
 FLOATING-POINT NUMERICAL REPRESENTATION OF MANTISSAS. D-3  
 FLOATING-POINT NUMERICAL REPRESENTATION. D-2  
 FLOATING-POINT REGISTERS. D-4

FLOW  
 DATA FLOW BETWEEN MAIN MEMORY AND ARITHMETIC UNIT. 2-8

FORMAT  
 ADDRESS FIELD FORMAT. 3-13  
 DATA AREA FORMAT. 3-14  
 DATA FIELD FORMAT IN MAIN MEMORY. 3-5  
 DATA FORMAT ON MAGNETIC TAPE. 3-8  
 DATA FORMAT. 3-1  
 DECIMAL DATA FORMAT IN MAIN MEMORY. D-18  
 DISK FORMAT. 3-10  
 FLAG CHARACTER FORMAT. 3-13  
 FLOATING-POINT ACCUMULATOR DATA FORMAT. D-2  
 FLOATING-POINT DATA FORMAT IN MAIN MEMORY. D-1  
 FORMAT OF TYPE 243 PDT C3 VARIANT. 8-136  
 INSTRUCTION FORMAT. 3-2  
 INSTRUCTION FORMATS. D-6  
 INSTRUCTIONS FORMATS AND TIMING. C-1  
 MAGNETIC TAPE DATA FORMAT. 3-7  
 PUNCHED CARD FORMAT. 3-9  
 RECORD FORMAT IN MAIN MEMORY. 3-6  
 RECORD FORMAT. 3-11  
 SERIES 2000 INSTRUCTION FORMAT 1. 4-16  
 SERIES 2000 INSTRUCTION FORMAT 2. 4-17  
 SERIES 2000 INSTRUCTION FORMAT 3. 4-17  
 SERIES 2000 INSTRUCTION FORMATS. 3-3  
 SUMMARY OF INTERNAL DATA FORMAT. 3-6  
 TRACK FORMAT. 3-11  
 TWO ITEM FORMATS IN MAIN MEMORY. 3-5

FORMATTING  
 AUTOMATIC FORMATTING IN ARITHMETIC OPERATIONS. D-5  
 DATA FORMATTING STATEMENTS. 6-1

FORMULAS  
 INSTRUCTION SUMMARY - TIMING FORMULAS FOR MODELS 2040. C-4

FOUR-CHARACTER  
 ASSEMBLY OF INDEXED ADDRESS IN FOUR-CHARACTER ADDRESSING MODE. 5-25

COMPUTER GENERATED INDEX

**FOUR-CHARACTER (CCNT)**  
 ASSEMBLY OF INDIRECT ADDRESS IN FOUR-CHARACTER ADDRESSING MODE. 5-26  
 EXTRACTION OF INDIRECT AND INDEXED FOUR-CHARACTER ADDRESSES. 4-14  
 FCLR-CHARACTER ADDRESS ASSEMBLY. 5-4  
 FOUR-CHARACTER ADDRESSING MODE. 4-8  
 FCLR-CHARACTER ADDRESS. 4-12  
 INDEX REGISTER ADDRESSES IN FCLR-CHARACTER ADDRESSING MODE. 4-13

**FUNCTION**  
 MAIN MEMORY FUNCTIONS. 2-2  
 SUMMARY OF INTERRUPT/ALLOW FUNCTION CONTROL AND TEST OPERATIONS. 2-37  
 TYPICAL CONTROL REGISTER FUNCTION. 2-4

**HALF ADD**  
 HALF ACC. 8-29

**HALT**  
 HALT. 8-52

**HEADER**  
 PROGRAM HEADER. 7-2  
 SEGMENT HEADER. 7-4

**HIGH-RESOLUTION**  
 HIGH-RESOLUTION CLOCK ALLOW. 2-31  
 HIGH-RESOLUTION CLOCK. 1-21 2-30

**HIGH-SPEED**  
 HIGH-SPEED DISK FILE. 1-15 1-6  
 HIGH-SPEED PRINTERS. 1-10 1-9

**IBM**  
 IBM MAGNETIC TAPE COMPATIBILITY. 1-12

**IMPLICIT**  
 EXPLICIT ADDRESSING, IMPLICIT ADDRESSING, AND CHAINING. 4-16

**INDEX**  
 ASSEMBLY OF INDEXED ADDRESS IN FOUR-CHARACTER ADDRESSING MODE. 5-25  
 ASSEMBLY OF INDEXED ADDRESS IN THREE-CHARACTER ADDRESSING MODE. 5-25  
 EXTRACTION OF INDEXED ADDRESS IN THREE-CHARACTER MODE. 4-12  
 EXTRACTION OF INDIRECT AND INDEXED FOUR-CHARACTER ADDRESSES. 4-14  
 INDEX REGISTER ADDRESSES IN FCLR-CHARACTER ADDRESSING MODE. 4-13  
 INDEX REGISTER ADDRESSES IN THREE-CHARACTER ADDRESSING MODE. 4-11  
 INDEX REGISTERS. 2-21 4-9  
 SERIES 2000 INDEX REGISTER MAP. 4-9

**INDEX/BARRICADE**  
 LOAD INDEX/BARRICADE REGISTER. 8-79  
 STORE INDEX/BARRICADE REGISTER. 8-83

**INDICATOR**  
 ANGULAR POSITION INDICATOR. 1-15  
 BRANCH ON CONDITION TEST (BCT) INDICATOR CONDITIONS. 8-4  
 EXTENDED I/C INDICATOR. 2-15  
 INDICATOR TEST CONDITIONS FOR BCT INSTRUCTION. 8-37  
 INDICATORS. 8-8 D-4  
 PROCEED INDICATOR. 2-25  
 RESTORE VARIANT AND INDICATORS. 8-98  
 SET 1 PUNCTUATION INDICATORS. 5-7  
 SET 11 PUNCTUATION INDICATORS (EASYSYCODER C, D, AND OS/2000). 5-8  
 STORE VARIANT AND INDICATORS. 8-94

**INDIRECT**  
 ASSEMBLY OF INDIRECT ADDRESS IN FOUR-CHARACTER ADDRESSING MODE. 5-26  
 ASSEMBLY OF INDIRECT ADDRESS IN THREE-CHARACTER ADDRESSING MODE. 5-26  
 EXTRACTION OF INDIRECT AND INDEXED FOUR-CHARACTER ADDRESSES. 4-14  
 EXTRACTION OF THREE-CHARACTER INDIRECT ADDRESS. 4-11

**INFORMATION**  
 SIZE OF INFORMATION UNITS IN MIT OPERATION. 8-75  
 VISUAL INFORMATION PROJECTION DEVICES. 1-18

**INPUT/OUTPUT**  
 BASIC INPUT/OUTPUT DATA PATH. 1-7  
 INPUT/OUTPUT CONTROL CHARACTERS. 5-23  
 INPUT/OUTPUT CONTROL OPERATIONS. 8-112  
 INPUT/OUTPUT TRAFFIC CONTROL ACTIVITIES. 2-10  
 INPUT/OUTPUT TRAFFIC CONTROL. 2-9  
 INPUT/OUTPUT. 8-111  
 LOGICAL DECISION PERFORMED BY INPUT/OUTPUT TRAFFIC CONTROL. 2-12

**INQUIRY**  
 CUSTOMER INQUIRY HANDLING VIA TYPICAL COMMUNICATIONS NETWORK. 1-19

**INSTRUCTION**  
 BCT INSTRUCTION VARIANT CHARACTERS. 8-38  
 BINARY INTEGER ARITHMETIC INSTRUCTION. D-25  
 CHANGING ADDRESSING MODES VIA CAM INSTRUCTION. 8-65  
 CONTROL INSTRUCTIONS. D-19

**INSTRUCTION (CCNT)**  
 CONTROL REGISTER CONTENTS LOADED BY LCR INSTRUCTION. 8-61  
 CONTROL REGISTER CONTENTS STORED BY SCR INSTRUCTION. 8-56  
 CONTROL REGISTERS STORED BY SCR INSTRUCTION. 8-59  
 DATA CONVERSION INSTRUCTIONS. C-17  
 DATA MOVING INSTRUCTIONS. D-10  
 EDIT INSTRUCTION. 1-20  
 EXPANDED INSTRUCTION PACKAGE. 1-22  
 EXTRACTION OF DATA FIELDS IN TYPICAL ADD INSTRUCTION. 4-2  
 FLOATING-POINT ARITHMETIC INSTRUCTIONS. D-13  
 INDICATOR TEST CONDITIONS FOR BCT INSTRUCTION. 8-37  
 INSTRUCTION FORMAT. 3-2  
 INSTRUCTION FORMATS. D-6  
 INSTRUCTION SUMMARY - TIMING FORMULAS FOR MODELS 2040. C-4  
 INSTRUCTION SUMMARY. C-1  
 INSTRUCTION TIMEOUT. 2-27  
 INSTRUCTION TIMINGS FOR MODELS 2040A, 2050, AND 2060. C-8  
 INSTRUCTION TIMINGS FOR MODELS 2050A AND 2070. C-11  
 INSTRUCTIONS FORMATS AND TIMING. C-1  
 INSTRUCTIONS. 8-1  
 MODES SPECIFIED BY VARIANT CHARACTER IN CAM INSTRUCTION. 8-63  
 PCB CONTROL CHARACTERS C5 THROUGH C15 TYPE 286-4, -5, -6, -7 LINE CONTROL INSTRUCTIONS. 8-154  
 PRIVILEGED BCT INSTRUCTION. 2-29  
 PRIVILEGED SCR INSTRUCTIONS. 2-29  
 SCR AND LCR INSTRUCTIONS. 2-31  
 SELECTING RWC ASSIGNMENTS FOR USE IN PDT INSTRUCTIONS. 8-112  
 SENSE SWITCH TEST CONDITIONS FOR BCT INSTRUCTION. 8-36  
 SERIES 2000 INSTRUCTION FORMAT 1. 4-16  
 SERIES 2000 INSTRUCTION FORMAT 2. 4-17  
 SERIES 2000 INSTRUCTION FORMAT 3. 4-17  
 SERIES 2000 INSTRUCTION FORMATS. 3-3  
 SPECIAL CHARACTERS IN MCE INSTRUCTION. 8-107  
 SYMBOLIC REPRESENTATION OF SERIES 2000 INSTRUCTIONS. 3-4  
 SYMBOLOGY USED IN SERIES 2000 INSTRUCTION DESCRIPTIONS. 8-2  
 TYPE 286-1, -2, -3 LINE CONTROL INSTRUCTIONS. 8-138  
 TYPICAL ADD INSTRUCTION. 4-1

**INTEGER**  
 BINARY INTEGER ARITHMETIC INSTRUCTION. D-25

**INTERFACE**  
 PERIPHERAL INTERFACE. 1-5

**INTERLOCKING**  
 INTERLOCKING READ/WRITE CHANNELS. 2-17

**INTERNAL**  
 INTERNAL INTERRUPT. 2-22 2-32  
 SAMPLE CODING FOR INTERNAL INTERRUPT ROUTINE. 2-35  
 SUMMARY OF INTERNAL DATA FORMAT. 3-6

**INTERNAL INTERRUPT REGISTER**  
 INTERNAL INTERRUPT REGISTER. 4-4

**INTERRUPT**  
 EXTERNAL INTERRUPT MASKING. 2-27  
 EXTERNAL INTERRUPT MODE. 2-30  
 EXTERNAL INTERRUPT. 2-31  
 INTERNAL INTERRUPT. 2-22 2-32  
 INTERRUPT CONTROL. 8-93  
 INTERRUPT PROCESSING MODE. 1-3  
 INTERRUPT PROCESSING. 2-31  
 INTERRUPT PROGRAMMING. 2-33  
 INTERRUPT SIGNAL GENERATED BY PERIPHERAL CONTROL. 2-36  
 PERIPHERAL CONTROL INTERRUPT. 2-35  
 PROGRAM INTERRUPT. 1-20  
 SAMPLE CODING FOR EXTERNAL INTERRUPT ROUTINE. 2-34  
 SAMPLE CODING FOR INTERNAL INTERRUPT ROUTINE. 2-35

**INTERRUPT/ALLOW**  
 SUMMARY OF INTERRUPT/ALLOW FUNCTION CONTROL AND TEST OPERATIONS. 2-37

**INTERVALS**  
 DATA TRANSFER INTERVALS DURING ONE PERIPHERAL OPERATION. 2-10

**ITEM**  
 MOVE ITEM AND TRANSLATE. 8-74  
 TWO ITEM FORMATS IN MAIN MEMORY. 3-5

**ITEM-MARK**  
 ITEM-MARK TRAPPING MODE. 1-4

**ITEMS**  
 ITEMS. 3-5  
 RELATIONSHIP BETWEEN ITEMS AND RECORDS. 3-10  
 RELATIONSHIP BETWEEN ITEMS, RECORDS, AND BLOCKS. 3-11

**LANGUAGE**  
 THE SYMBOLIC LANGUAGE. 5-3

**LCR**  
 CONTROL REGISTER CONTENTS LOADED BY LCR INSTRUCTION. 8-61



COMPUTER GENERATED INDEX

LCR (CONT)  
 SCR AND LCR INSTRUCTIONS. 2-31

LENGTH  
 VARIABLE FIELD LENGTH. 3-1

LIB  
 CORRESPONDENCE BETWEEN LIB SETTING AND BARRICADE LOCATION. 8-81  
 LIB VARIANT CHARACTER. 8-81

LINE  
 PCB CONTROL CHARACTERS C5 THROUGH C15 TYPE 286-4, -5, -6, -7 LINE CONTROL INSTRUCTIONS. 8-154  
 TYPE 286-1, -2, -3 LINE CONTROL INSTRUCTIONS. 8-138

LITERAL  
 ADDRESS LITERALS. 5-22  
 ALPHANUMERIC LITERALS. 5-20  
 AREA DEFINING LITERALS. 5-21  
 BINARY LITERALS. 5-19  
 DECIMAL LITERALS. 5-18  
 LITERAL ORIGIN. 7-10  
 LITERALS. 5-18  
 OCTAL LITERALS. 5-19

LOAD  
 LOAD CHARACTERS TO A-FIELD WORD MARK. 8-56  
 LOAD INDEX/BARRICADE REGISTER. 8-79

LOAD CONTROL REGISTERS  
 LOAD CONTROL REGISTERS. 8-66

LOADED  
 ADDRESS ASSIGNMENTS AND UNIT LOADS AVAILABLE IN SERIES 2000 PROCESSORS. 1-8  
 CONTROL REGISTER CONTENTS LOADED BY LCR INSTRUCTION. 8-61  
 PERIPHERAL ADDRESSES AND UNIT LOADS. 1-7

LOCATION  
 CONSECUTIVE STORAGE LOCATIONS IN MAIN MEMORY. 3-4  
 CORRESPONDENCE BETWEEN LIB SETTING AND BARRICADE LOCATION. 8-81  
 FACTOR LOCATIONS IN DIVIDE OPERATION. 8-11  
 LOCATION. 5-8

LOGIC  
 LOGIC. 8-27

LOGICAL  
 LOGICAL DECISION PERFORMED BY INPUT/OUTPUT TRAFFIC CONTROL. 2-12  
 LOGICAL DIVISION OF SERIES 2000 CENTRAL PROCESSOR. 2-1

MAGNETIC  
 CHARACTER REPRESENTATION ON 7-TRACK MAGNETIC TAPE. 3-7  
 DATA FORMAT ON MAGNETIC TAPE. 3-8  
 IBM MAGNETIC TAPE COMPATIBILITY. 1-12  
 MAGNETIC TAPE DATA FORMAT. 3-7  
 MAGNETIC TAPE UNITS. 1-10 1-11  
 REPRESENTATION OF CHARACTERS IN MAGNETIC CORE STORAGE. 2-2

MANTISSAS  
 FLOATING-POINT NUMERICAL REPRESENTATION OF MANTISSAS. D-3

MAP  
 SERIES 2000 INDEX REGISTER MAP. 4-9

MARK  
 CLEAR ITEM MARK. 8-51  
 CLEAR WORD MARK. 8-50  
 DEFINE CONSTANT WITH WORD MARK. 6-2  
 LOAD CHARACTERS TO A-FIELD WORD MARK. 8-56  
 MARK. 5-7  
 MOVE CHARACTERS TO WORD MARK. 8-55  
 SET ITEM MARK. 8-49  
 SET WORD MARK. 8-48

MASKING  
 EXTERNAL INTERRUPT MASKING. 2-27

MASS-STORAGE  
 DATA CONVENTIONS OF HONEYWELL MASS-STORAGE DISK DEVICES. 3-12

MAT  
 MAT OPERATION. 8-73

MCE  
 SPECIAL CHARACTERS IN MCE INSTRUCTION. 8-107

MEMORY  
 CONSECUTIVE STORAGE LOCATIONS IN MAIN MEMORY. 3-4  
 CONTROL MEMORY REGISTERS. 2-6  
 CONVERSION OF SYMBOLIC TAG TO ABSOLUTE MEMORY ADDRESS. 3-2  
 DATA FIELD FORMAT IN MAIN MEMORY. 3-5  
 DATA FLOW BETWEEN MAIN MEMORY AND ARITHMETIC UNIT. 2-8  
 DECIMAL DATA FORMAT IN MAIN MEMORY. D-18  
 EXECUTION TIMINGS IN MEMORY CYCLES. D-9  
 FLOATING-POINT DATA FORMAT IN MAIN MEMORY. D-1  
 MAIN MEMORY FUNCTIONS. 2-2  
 MAIN MEMORY SIZE. 1-5  
 MAIN MEMORY SPEED. 1-5  
 MAIN MEMORY. 2-1  
 MEMORY ACCESS DISTRIBUTION IN THE TYPE 2071 PROCESSOR. 2-16  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041 PROCESSOR.

MEMORY (CONT)  
 2-12  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041A PROCESSOR. 2-13  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051A PROCESSOR. 2-14  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051C PROCESSOR. 2-13  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2061 PROCESSOR. 2-16  
 MEMORY ACCESS DISTRIBUTION OF THE TYPE 2071 PROCESSOR. 2-17  
 MEMORY ACCESS DISTRIBUTION. 2-10  
 MEMORY CYCLE. 2-3  
 MEMORY DUMP. 7-15  
 ONE MEMORY POSITION. 2-2  
 ORGANIZATION OF DATA IN MAIN MEMORY. 3-4  
 RECORD FORMAT IN MAIN MEMORY. 3-6  
 SIZE OF CONTROL MEMORY REGISTERS. 2-4  
 TWO ITEM FORMATS IN MAIN MEMORY. 3-5

MIT  
 MIT OPERATION. 8-79  
 SIZE OF INFORMATION UNITS IN MIT OPERATION. 8-75

MODE  
 ASSEMBLY OF INDEXED ADDRESS IN FOUR-CHARACTER ADDRESSING MODE. 5-25  
 ASSEMBLY OF INDEXED ADDRESS IN THREE-CHARACTER ADDRESSING MODE. 5-25  
 ASSEMBLY OF INDIRECT ADDRESS IN FOUR-CHARACTER ADDRESSING MODE. 5-26  
 ASSEMBLY OF INDIRECT ADDRESS IN THREE-CHARACTER ADDRESSING MODE. 5-26  
 BUFFERED MODE. 2-18  
 DIRECT-ACCESS MODE. 2-19  
 EXTERNAL INTERRUPT MODE. 2-30  
 EXTRACTION OF INDEXED ADDRESS IN THREE-CHARACTER MODE. 4-12  
 FOUR-CHARACTER ADDRESSING MODE. 4-8  
 INDEX REGISTER ADDRESSES IN FOUR-CHARACTER ADDRESSING MODE. 4-13  
 INDEX REGISTER ADDRESSES IN THREE-CHARACTER ADDRESSING MODE. 4-11  
 INTERRUPT PROCESSING MODE. 1-3  
 ITEM-MARK TRAPPING MODE. 1-4  
 SET ADDRESS MODE. 7-12  
 STANDARD PROCESSING MODE. 1-2  
 THREE-CHARACTER ADDRESSING MODE. 4-6  
 TWO-CHARACTER ADDRESSING MODE. 4-5

MODEL  
 INSTRUCTION SUMMARY - TIMING FORMULAS FOR MODELS 204C. C-4  
 INSTRUCTION TIMINGS FOR MODELS 2040A, 2050, AND 2060. C-8  
 INSTRUCTION TIMINGS FOR MODELS 2050A AND 2070. C-11  
 MODEL 204CA POWER MODULES. 1-22  
 MODEL 205CA POWER MODULES. 1-22  
 TIMINGS FOR DECIMAL MULTIPLY AND DIVIDE - MODEL 2040. C-16

MODES  
 ADDRESSING MODES. 1-4 4-5  
 CENTRAL PROCESSOR MODES. 2-21  
 CHANGING ADDRESSING MODES VIA CAM INSTRUCTION. 8-65  
 MODES SPECIFIED BY VARIANT CHARACTER IN CAM INSTRUCTION. 8-63

MODIFICATION  
 ADDRESS MODIFICATION CODES. 5-24  
 ADDRESS MODIFICATION. 4-8

MODULAR  
 MODULAR ORIGIN. 7-9

MODULES  
 FEATURES AND POWER MODULES. 1-20  
 MODEL 204CA POWER MODULES. 1-22  
 MODEL 205CA POWER MODULES. 1-22

MONITOR CALL  
 MONITOR CALL. 8-100

MOVE  
 DATA MOVING INSTRUCTIONS. D-10  
 EXTENDED MOVE (EXM) CONDITIONS. E-2  
 MOVE AND TRANSLATE. 8-70  
 MOVE CHARACTERS AND EDIT. 8-106  
 MOVE CHARACTERS TO WORD MARK. 8-55  
 MOVE ITEM AND TRANSLATE. 8-74  
 MOVE OR SCAN CONDITIONS. 8-89  
 MOVE OR SCAN VARIANTS. 8-9  
 MOVE OR SCAN. 8-87

MULTILINE  
 SUMMARY OF PCB I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-152  
 SUMMARY OF PDT I/O CONTROL CHARACTERS FOR TYPE 286 MULTILINE COMMUNICATION CONTROLLER. 8-138

MULTIPLICATION  
 MULTIPLICATION. 8-8

COMPUTER GENERATED INDEX

- MULTIPLY
  - A-FIELD AND B-FIELD IN MULTIPLY OPERATION. 8-9
  - MULTIPLY SIGN CONVENTIONS. 6-9
  - MULTIPLY. 8-23
  - TIMINGS FOR DECIMAL MULTIPLY AND DIVIDE - MODEL 2040. C-16
- MULTIPROGRAMMING
  - EXTENDED MULTIPROGRAMMING AND EIGHT-BIT TRANSFER. 1-21
  - 2-26
- NETWORK
  - CUSTOMER INQUIRY HANDLING VIA TYPICAL COMMUNICATIONS NETWORK. 1-19
- NO OPERATION
  - NO OPERATION. 8-54
- NOTATION
  - OCTAL NOTATION. A-1
- NUMERIC
  - NUMERIC CONSTANTS. 6-2
- NUMERICAL
  - FLOATING-POINT NUMERICAL REPRESENTATION OF EXPONENTS. D-3
  - FLOATING-POINT NUMERICAL REPRESENTATION OF MANTISSAS. D-3
  - FLOATING-POINT NUMERICAL REPRESENTATION. D-2
  - NUMERICAL REPRESENTATION OF DECIMAL WORD DATA. D-19
- OBJECT
  - RELATIONSHIP OF SOURCE PROGRAM, ASSEMBLER, AND OBJECT PROGRAM. 5-2
- OCTAL
  - BINARY, OCTAL, AND DECIMAL EQUIVALENTS. E-8
  - OCTAL CONSTANTS. 6-3
  - OCTAL LITERALS. 5-19
  - OCTAL NOTATION. A-1
- OCTAL-DECIMAL
  - OCTAL-DECIMAL CONVERSION PROCEDURE. A-3
- OPERANDS
  - OPERANDS. 5-12
- OPERATION
  - A-FIELD AND B-FIELD IN MULTIPLY OPERATION. 8-9
  - ARITHMETIC OPERATIONS. 8-3
  - AUTOMATIC FORMATTING IN ARITHMETIC OPERATIONS. D-5
  - BUFFERED SECTOR OPERATION. 2-18
  - DATA PATH DURING CARD READ OPERATION. 1-8
  - DATA TRANSFER INTERVALS DURING ONE PERIPHERAL OPERATION. 2-10
  - FACTOR LOCATIONS IN DIVIDE OPERATION. 8-11
  - INPUT/OUTPUT CONTROL OPERATIONS. 8-112
  - MAT OPERATION. 8-73
  - MIT OPERATION. 8-79
  - OPERATION CODE. 3-2 5-12
  - PERIPHERAL DATA TRANSFER OPERATION. 1-6
  - SERIES 2000 ADD AND SUBTRACT OPERATIONS. 8-3
  - SIZE OF INFORMATION UNITS IN MIT OPERATION. 8-75
  - SUMMARY OF INTERRUPT/ALLOW FUNCTION CONTROL AND TEST OPERATIONS. 2-37
  - TEST AND CONTROL OPERATIONS. 8-140
  - TLL OPERATION. 8-88
- OPTIONS
  - EASYCODER C, D, AND OS/2000 OPTIONS. 6-10
- ORGANIZATION
  - ORGANIZATION OF DATA IN MAIN MEMORY. 3-4
- ORIGIN
  - LITERAL ORIGIN. 7-10
  - MODULAR ORIGIN. 7-9
  - ORIGIN. 7-7
- OS/2000
  - EASYCODER C, D, AND OS/2000 OPTIONS. 6-10
  - SET II PUNCTUATION INDICATORS (EASYCODER C, D, AND OS/2000). 5-8
- OUT-OF-SEQUENCE
  - OUT-OF-SEQUENCE. 5-17
- PACK
  - DISK PACK DRIVE FEATURES. 1-14
  - DISK PACK DRIVES AND DISK SUBSYSTEMS. 1-13
  - DISK PACK DRIVES. 1-12
- PACKAGE
  - EXPANDED INSTRUCTION PACKAGE. 1-22
- PAPER
  - PAPER TAPE EQUIPMENT. 1-16
- PAPER TAPE
  - C3 CODING FOR TYPE 209 AND 209-2 PAPER TAPE READERS. 8-136
  - C3 CODING FOR TYPE 210 PAPER TAPE PUNCH. 8-137
- PATH
  - BASIC INPUT/OUTPUT DATA PATH. 1-7
  - DATA PATH DURING CARD READ OPERATION. 1-8
- PERIPHERAL
  - DATA TRANSFER INTERVALS DURING ONE PERIPHERAL OPERATION. 2-10
  - DESCRIPTION OF PDT I/O CONTROL CHARACTER C2 (PERIPHERAL. 8-129
  - INTERRUPT SIGNAL GENERATED BY PERIPHERAL CONTROL. 2-36
- PERIPHERAL (CONT)
  - MINIMUM RWC CAPACITY REQUIREMENTS FOR SERIES 200/2000 PERIPHERAL DEVICES. 8-113
  - PERIPHERAL ADDRESSES AND UNIT LOADS. 1-7
  - PERIPHERAL CONTROL AND BRANCH. 8-139
  - PERIPHERAL CONTROL INTERRUPT. 2-35
  - PERIPHERAL CONTROL. 1-6
  - PERIPHERAL DATA TRANSFER OPERATION. 1-6
  - PERIPHERAL DATA TRANSFER. 8-116
  - PERIPHERAL EQUIPMENT. 1-5
  - PERIPHERAL INTERFACE. 1-5
  - PERIPHERAL SIMULTANEITY. 1-5
  - TESTING PERIPHERAL CONTROL UNIT BUSY STATUS. 2-19
- POSITION
  - ANGULAR POSITION INDICATOR. 1-15
  - ONE MEMORY POSITION. 2-2
- POSTNORMALIZATION
  - POSTNORMALIZATION. D-6
- POWER
  - FEATURES AND POWER MODULES. 1-20
  - MODEL 2040A POWER MODULES. 1-22
  - MODEL 2050A POWER MODULES. 1-22
  - PROCESSING POWER. 1-4
- POWERS OF 2
  - POWERS OF 2. B-8
- PRENORMALIZATION
  - PRENORMALIZATION. D-5
- PRINT
  - PRINT BUFFER. 1-10
- PRINTERS
  - C3 CODING FOR TYPE 222 PRINTERS. 8-137
  - HIGH-SPEED PRINTERS. 1-10 1-9
- PROCEDURE
  - OCTAL-DECIMAL CONVERSION PROCEDURE. A-3
- PROCEED
  - PROCEED INDICATOR. 2-25
- PROCESSING
  - INTERRUPT PROCESSING MODE. 1-3
  - INTERRUPT PROCESSING. 2-31
  - PROCESSING POWER. 1-4
  - STANDARD PROCESSING MODE. 1-2
- PROCESSOR
  - ACTIVE ADDRESS BITS IN SERIES 2000 SINGLE-CHARACTER PROCESSORS. 4-14
  - ADDRESS ASSIGNMENTS AND UNIT LOADS AVAILABLE IN SERIES 2000 PROCESSORS. 1-8
  - CENTRAL PROCESSOR FINISHED. 1-14
  - CENTRAL PROCESSOR MODES. 2-21
  - CENTRAL PROCESSOR. 1-1
  - LOGICAL DIVISION OF SERIES 2000 CENTRAL PROCESSOR. 2-1
  - MEMORY ACCESS DISTRIBUTION IN THE TYPE 2071 PROCESSOR. 2-16
  - MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041 PROCESSOR. 2-12
  - MEMORY ACCESS DISTRIBUTION OF THE TYPE 2041A PROCESSOR. 2-13
  - MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051A PROCESSOR. 2-14
  - MEMORY ACCESS DISTRIBUTION OF THE TYPE 2051C PROCESSOR. 2-13
  - MEMORY ACCESS DISTRIBUTION OF THE TYPE 2061 PROCESSOR. 2-16
  - MEMORY ACCESS DISTRIBUTION OF THE TYPE 2071 PROCESSOR. 2-17
  - THE CENTRAL PROCESSOR. 2-1
- PROJECTION
  - VISUAL INFORMATION PROJECTION DEVICES. 1-18
- PROTECT
  - WRITE PROTECT CAPABILITY. 1-14
- PROTECTION
  - BASIC STORAGE PROTECTION. 8-80
  - STORAGE PROTECTION FEATURE. 2-21
  - STORAGE PROTECTION WITH BASE RELOCATION. 2-26 8-80
  - STORAGE PROTECTION. 1-21
  - VIOLATIONS OF STORAGE PROTECTION. 2-23
- PUNCH
  - C3 CODING FOR TYPE 210 PAPER TAPE PUNCH. 8-137
  - PUNCHED CARD EQUIPMENT. 1-9
  - PUNCHED CARD FORMAT. 3-9
- PUNCHED CARD
  - PUNCHED CARD CODES. 3-9
- PUNCTUATION
  - SET I PUNCTUATION INDICATORS. 5-7
  - SET II PUNCTUATION INDICATORS (EASYCODER C, D, AND OS/2000). 5-8
- RANDOM
  - C3 CODING FOR TYPE 270A RANDOM ACCESS DRUM. 8-137
  - RANDOM ACCESS DRUMS. 1-15
- RANGE
  - POTENTIAL ADDRESSES OUTSIDE ADDRESS REGISTER RANGE. 4-15
  - POTENTIAL ADDRESSES WITHIN ADDRESS REGISTER RANGE. 4-15

COMPUTER GENERATED INDEX

RANGE (CONT)  
 RANGE, 7-20

RATES  
 DATA TRANSFER RATES, 2-9

READ  
 DATA PATH DURING CARD READ OPERATION, 1-8

READERS  
 C3 CODING FOR TYPE 209 AND 209-2 PAPER TAPE READERS, 8-136

READ/WRITE  
 INTERLOCKING READ/WRITE CHANNELS, 2-17  
 READ/WRITE CHANNEL, 1-8  
 READ/WRITE COUNTERS, 2-5  
 VARIABLE-SPEED READ/WRITE CHANNELS, 2-17

RECORD  
 1200-BPI RECORDING DENSITY, 1-10  
 1600-BPI RECORDING DENSITY, 1-10  
 RECORD FORMAT IN MAIN MEMORY, 3-6  
 RECORD FORMAT, 3-11  
 RECORDS, 3-6  
 RELATIONSHIP BETWEEN ITEMS AND RECORDS, 3-10  
 RELATIONSHIP BETWEEN ITEMS, RECORDS, AND BLOCKS, 3-11  
 TRACK-LINKING RECORD, 3-15

REGISTER  
 ACCOUNTING TIMER REGISTER, 2-30  
 CONTROL REGISTER CONTENTS LOADED BY LCR INSTRUCTION, 8-61  
 CONTROL REGISTER CONTENTS STORED BY SCR INSTRUCTION, 8-58  
 CONTROL REGISTER DESIGNATIONS, 8-1  
 INDEX REGISTER ADDRESSES IN FOUR-CHARACTER ADDRESSING MODE, 4-13  
 INDEX REGISTER ADDRESSES IN THREE-CHARACTER ADDRESSING MODE, 4-11  
 LOAD INDEX/BARRICADE REGISTER, 8-79  
 POTENTIAL ADDRESSES OUTSIDE ADDRESS REGISTER RANGE, 4-15  
 POTENTIAL ADDRESSES WITHIN ADDRESS REGISTER RANGE, 4-15  
 SERIES 2000 INDEX REGISTER MAP, 4-9  
 STORE INDEX/BARRICADE REGISTER, 8-83  
 TYPICAL CONTROL REGISTER FUNCTION, 2-4

REGISTERS  
 ADDRESS REGISTERS, 2-5  
 CONTROL MEMORY REGISTERS, 2-6  
 CONTROL REGISTERS STORED BY SCR INSTRUCTION, 8-59  
 FLOATING-POINT REGISTERS, D-4  
 INDEX REGISTERS, 2-21 4-9  
 REGISTERS USED IN ADDRESSING, 4-3  
 SIZE OF CONTROL MEMORY REGISTERS, 2-4

RELATIONSHIP  
 RELATIONSHIP BETWEEN ITEMS AND RECORDS, 3-10  
 RELATIONSHIP BETWEEN ITEMS, RECORDS, AND BLOCKS, 3-11  
 RELATIONSHIP OF SOURCE PROGRAM, ASSEMBLER, AND OBJECT PROGRAM, 5-2

RELATIVE  
 RELATIVE, 5-16

RELOCATION  
 STORAGE PROTECTION WITH BASE RELOCATION, 2-26 8-80

REPEAT  
 REPEAT, 7-17

REQUIREMENTS  
 MINIMUM RWC CAPACITY REQUIREMENTS FOR SERIES 200/2000 PERIPHERAL DEVICES, 8-113

RESERVE  
 RESERVE AREA, 6-6

RESTORE  
 RESTORE VARIANT AND INDICATORS, 8-98

RESTRICTIONS  
 BUFFERED SECTOR RESTRICTIONS, 2-19

RESUME NORMAL MODE  
 RESUME NORMAL MODE, 8-101

ROUTINE  
 SAMPLE CODING FOR EXTERNAL INTERRUPT ROUTINE, 2-34  
 SAMPLE CODING FOR INTERNAL INTERRUPT ROUTINE, 2-35

RULES  
 ADDITIONAL CODING RULES, 5-14

RWC  
 DESCRIPTION OF PDT I/O CONTROL CHARACTER C1 (RWC ASSIGNMENT), 8-117  
 MINIMUM RWC CAPACITY REQUIREMENTS FOR SERIES 200/2000 PERIPHERAL DEVICES, 8-113  
 SELECTING RWC ASSIGNMENTS FOR USE IN PDT INSTRUCTIONS, 8-112

SCAN  
 MOVE OR SCAN CONDITIONS, 8-89  
 MOVE OR SCAN VARIANTS, B-9  
 MOVE OR SCAN, 8-87

SCIENTIFIC  
 SCIENTIFIC UNIT AND SCIENTIFIC SUBPROCESSOR, 1-21 D-1

SCR  
 CONTROL REGISTER CONTENTS STORED BY SCR INSTRUCTION, 8-58

SCR (CONT)  
 CONTROL REGISTERS STORED BY SCR INSTRUCTION, 8-59  
 PRIVILEGED SCR INSTRUCTIONS, 2-25  
 SCR AND LCR INSTRUCTIONS, 2-31

SECTOR  
 BUFFERED SECTOR OPERATION, 2-18  
 BUFFERED SECTOR RESTRICTIONS, 2-19  
 BUFFERED SECTORS, 2-18  
 CONTROLS/DEVICES CONNECTABLE TO BUFFERED SECTORS, 2-20

SEGMENT  
 SEGMENT HEADER, 7-4

SELF REFERENCE  
 SELF REFERENCE, 5-15

SENSE  
 BRANCH ON CONDITION TEST (BCT) SENSE SWITCH CONDITIONS, B-3  
 SENSE SWITCH TEST CONDITIONS FOR ECT INSTRUCTION, 8-36

SEQUENCE REGISTER  
 SEQUENCE REGISTER, 4-3

SERIES 2000  
 ACTIVE ADDRESS BITS IN SERIES 2000 SINGLE-CHARACTER PROCESSORS, 4-14  
 ADDRESS ASSIGNMENTS AND UNIT LOADS AVAILABLE IN SERIES 2000 PROCESSORS, 1-8  
 LOGICAL DIVISION OF SERIES 2000 CENTRAL PROCESSOR, 2-1  
 SERIES 2000 ADD AND SUBTRACT OPERATIONS, B-3  
 SERIES 2000 CHARACTER CODES, B-7  
 SERIES 2000 COMPONENTS, 1-1  
 SERIES 2000 INDEX REGISTER MAP, 4-9  
 SERIES 2000 INSTRUCTION FORMAT 1, 4-16  
 SERIES 2000 INSTRUCTION FORMAT 2, 4-17  
 SERIES 2000 INSTRUCTION FORMAT 3, 4-17  
 SERIES 2000 INSTRUCTION FORMATS, 3-3  
 SYMBOLIC REPRESENTATION OF SERIES 2000 INSTRUCTIONS, 3-4  
 SYMBOLOLOGY USED IN SERIES 2000 INSTRUCTION DESCRIPTIONS, 8-2

SERIES 200/2000  
 MINIMUM RWC CAPACITY REQUIREMENTS FOR SERIES 200/2000 PERIPHERAL DEVICES, 8-113

SET I  
 SET I PUNCTUATION INDICATORS, 5-7

SET II  
 SET II PUNCTUATION INDICATORS (EASYSOFT C, D, AND OS/2000), 5-8

SET ITEM  
 SET ITEM MARK, 8-49

SET LINE NUMBER  
 SET LINE NUMBER, 7-18

SET OUT-OF-SEQUENCE BASE  
 SET OUT-OF-SEQUENCE BASE, 7-19

SET WORD  
 SET WORD MARK, 8-48

SETTING  
 CORRESPONDENCE BETWEEN LIE SETTING AND BARRICADE LOCATION, 8-81

SIGN  
 DECIMAL ARITHMETIC SIGN CONVENTIONS, 8-8  
 DIVIDE SIGN CONVENTIONS, 8-12  
 MULTIPLY SIGN CONVENTIONS, 8-9

SIGNAL  
 INTERRUPT SIGNAL GENERATED BY PERIPHERAL CONTROL, 2-36

SIGNS  
 ALGEBRAIC SIGNS IN DECIMAL ADDITION, 8-6

SIMULTANEITY  
 PERIPHERAL SIMULTANEITY, 1-5

SINGLE-CHARACTER  
 ACTIVE ADDRESS BITS IN SERIES 2000 SINGLE-CHARACTER PROCESSORS, 4-14

SKIP  
 SKIP, 7-16

SOURCE  
 RELATIONSHIP OF SOURCE PROGRAM, ASSEMBLER, AND OBJECT PROGRAM, 5-2

SPEED  
 MAIN MEMORY SPEED, 1-5

STATEMENTS  
 ASSEMBLY CONTROL STATEMENTS, 7-1  
 DATA FORMATTING STATEMENTS, 6-1

STATUS  
 TESTING PERIPHERAL CONTROL UNIT BUSY STATUS, 2-19

STORAGE  
 BASIC STORAGE PROTECTION, 8-80  
 CONSECUTIVE STORAGE LOCATIONS IN MAIN MEMORY, 3-4  
 REPRESENTATION OF CHARACTERS IN MAGNETIC CORE STORAGE, 2-2  
 STORAGE PROTECTION FEATURE, 2-21  
 STORAGE PROTECTION WITH BASE RELOCATION, 2-26 8-80  
 STORAGE PROTECTION, 1-21  
 VIOLATIONS OF STORAGE PROTECTION, 2-23

STORE  
 STORE INDEX/BARRICADE REGISTER, 8-83

COMPUTER GENERATED INDEX

STORE (CONT)  
 STORE VARIANT AND INDICATORS. 8-94  
 STORE CONTROL REGISTERS  
 STORE CONTROL REGISTERS. 8-58  
 STORED  
 CONTROL REGISTER CONTENTS STORED BY SCR INSTRUCTION.  
 8-58  
 CONTROL REGISTERS STORED BY SCR INSTRUCTION. 8-59  
 SUBPROCESSOR  
 SCIENTIFIC UNIT AND SCIENTIFIC SUBPROCESSOR. 1-21 D-1  
 SUBSTITUTE  
 SUBSTITUTE. 8-30  
 SUBSYSTEMS  
 DISK PACK DRIVES AND DISK SUBSYSTEMS. 1-13  
 SUBTRACT  
 SERIES 2000 ADD AND SUBTRACT OPERATIONS. 8-3  
 SUBTRACT. 8-16  
 SUBTRACTION  
 BINARY SUBTRACTION. 8-3  
 DECIMAL SUBTRACTION. 8-7  
 SUFFIX  
 SUFFIX. 7-16  
 SWITCH  
 BRANCH ON CONDITION TEST (BCT) SENSE SWITCH CONDITIONS.  
 8-3  
 SENSE SWITCH TEST CONDITIONS FOR BCT INSTRUCTION. 8-36  
 SYMBOLIC  
 CONVERSION OF SYMBOLIC TAG TO ABSOLUTE MEMORY ADDRESS.  
 3-2  
 DEFINE SYMBOLIC ADDRESS. 6-7  
 SYMBOLIC REPRESENTATION OF SERIES 2000 INSTRUCTIONS.  
 3-4  
 SYMBOLIC. 5-15  
 THE SYMBOLIC LANGUAGE. 5-3  
 SYMBOLOGY  
 SYMBOLOGY FOR EXECUTION TIMING. D-7  
 SYMBOLOGY USED IN SERIES 2000 INSTRUCTION DESCRIPTIONS.  
 8-2  
 TABLE  
 DECIMAL-OCTAL CONVERSION TABLE. A-2  
 TABLE LOOKUP  
 TABLE LOOKUP. 8-84  
 TABLES  
 MISCELLANEOUS TABLES. B-1  
 TAG  
 CONVERSION OF SYMBOLIC TAG TO ABSOLUTE MEMORY ADDRESS.  
 3-2  
 TAPE  
 C4 VARIANT FOR 9-TRACK TAPE UNITS. 8-132  
 CHARACTER REPRESENTATION ON 7-TRACK MAGNETIC TAPE. 3-7  
 DATA FORMAT ON MAGNETIC TAPE. 3-8  
 DYNAMIC TAPE ADDRESSING. 1-11  
 IBM MAGNETIC TAPE COMPATIBILITY. 1-12  
 MAGNETIC TAPE DATA FORMAT. 3-7  
 MAGNETIC TAPE UNITS. 1-10 1-11  
 PAPER TAPE EQUIPMENT. 1-16  
 TELLER  
 TELLER TERMINAL EQUIPMENT. 1-20  
 TERMINAL  
 TELLEK TERMINAL EQUIPMENT. 1-20  
 TEST  
 BCC TEST CONDITIONS. 8-41  
 BRANCH ON CONDITION TEST (BCT) INDICATOR CONDITIONS.  
 8-4  
 BRANCH ON CONDITION TEST (BCT) SENSE SWITCH CONDITIONS.  
 8-3  
 INDICATOR TEST CONDITIONS FOR BCT INSTRUCTION. 8-37  
 SENSE SWITCH TEST CONDITIONS FOR BCT INSTRUCTION. 8-36  
 SUMMARY OF INTERRUPT/ALLOW FUNCTION CONTROL AND TEST  
 OPERATIONS. 2-37  
 TEST AND CONTROL OPERATIONS. 8-140  
 TESTING PERIPHERAL CONTROL UNIT BUSY STATUS. 2-19  
 THREE-CHARACTER  
 ASSEMBLY OF INDEXED ADDRESS IN THREE-CHARACTER ADDRESSING  
 MODE. 5-25  
 ASSEMBLY OF INDIRECT ADDRESS IN THREE-CHARACTER  
 ADDRESSING MODE. 5-26  
 EXTRACTION OF INDEXED ADDRESS IN THREE-CHARACTER MODE.  
 4-12  
 EXTRACTION OF THREE-CHARACTER INDIRECT ADDRESS. 4-11  
 INDEX REGISTER ADDRESSES IN THREE-CHARACTER ADDRESSING  
 MODE. 4-11  
 THREE-CHARACTER ADDRESS ASSEMBLY. 5-4  
 THREE-CHARACTER ADDRESSING MODE. 4-6  
 THREE-CHARACTER ADDRESS. 4-10  
 TIMEOUT  
 INSTRUCTION TIMEOUT. 2-27  
 TIMER  
 INSTRUCTION SUMMARY - TIMING FORMULAS FOR MODELS 2040.  
 TIMER (CONT)  
 C-4  
 INSTRUCTIONS FORMATS AND TIMING. C-1  
 SYMBOLOGY FOR EXECUTION TIMING. C-7  
 TIMING NOTES. D-8  
 TIMINGS  
 EXECUTION TIMINGS IN MEMORY CYCLES. D-9  
 INSTRUCTION TIMINGS FOR MODELS 2040A, 2050, AND 2060.  
 C-8  
 INSTRUCTION TIMINGS FOR MODELS 2050A AND 2070. C-11  
 TIMINGS FOR DECIMAL MULTIPLY AND DIVIDE - MODEL 2040.  
 C-16  
 TLU  
 TLU OPERATION. 8-88  
 TRACK  
 TRACK FORMAT. 3-11  
 TRACK-LINKING  
 TRACK-LINKING RECORD. 3-15  
 TRAFFIC  
 INPUT/OUTPUT TRAFFIC CONTROL ACTIVITIES. 2-10  
 INPUT/OUTPUT TRAFFIC CONTROL. 2-5  
 LOGICAL DECISION PERFORMED BY INPUT/OUTPUT TRAFFIC  
 CONTROL. 2-12  
 TRANSFER  
 DATA TRANSFER INTERVALS DURING ONE PERIPHERAL OPERATION.  
 2-10  
 DATA TRANSFER RATES. 2-9  
 EIGHT-BIT TRANSFER CAPABILITY. 2-28  
 EIGHT-BIT TRANSFER. 1-15  
 EXTENDED MULTIPROGRAMMING AND EIGHT-BIT TRANSFER. 1-21  
 2-26  
 PERIPHERAL DATA TRANSFER OPERATION. 1-6  
 PERIPHERAL DATA TRANSFER. 8-116  
 TRANSFER. 7-6  
 TRANSLATE  
 MOVE AND TRANSLATE. 8-70  
 MOVE ITEM AND TRANSLATE. 8-74  
 TRANSLATION  
 EBCDIC CODE TRANSLATION. 1-12  
 TRAPPING  
 ITEM-MARK TRAPPING MODE. 1-4  
 TRUE ADD  
 TRUE ADD. 8-6  
 TWO-CHARACTER  
 TWO-CHARACTER ADDRESS ASSEMBLY. 5-3  
 TWO-CHARACTER ADDRESSING MODE. 4-5  
 UNIT  
 ADDRESS ASSIGNMENTS AND UNIT LOCATIONS AVAILABLE IN SERIES  
 2000 PROCESSORS. 1-8  
 ARITHMETIC UNIT. 2-8  
 C4 VARIANT FOR 9-TRACK TAPE UNITS. 8-132  
 CONTROL UNIT. 2-9  
 DATA FLOW BETWEEN MAIN MEMORY AND ARITHMETIC UNIT. 2-8  
 MAGNETIC TAPE UNITS. 1-10 1-11  
 PERIPHERAL ADDRESSES AND UNIT LOCATIONS. 1-7  
 SCIENTIFIC UNIT AND SCIENTIFIC SUBPROCESSOR. 1-21 D-1  
 SIZE OF INFORMATION UNITS IN MIT OPERATION. 8-75  
 TESTING PERIPHERAL CONTROL UNIT BUSY STATUS. 2-19  
 VARIABLE  
 VARIABLE FIELD LENGTH. 3-1  
 VARIABLE-SPEED  
 VARIABLE-SPEED READ/WRITE CHANNELS. 2-17  
 VARIANT  
 BCT INSTRUCTION VARIANT CHARACTERS. 8-38  
 C4 VARIANT FOR 9-TRACK TAPE UNITS. 8-132  
 FORMAT OF TYPE 243 PDT C3 VARIANT. 8-136  
 LIB VARIANT CHARACTER. 8-81  
 MODES SPECIFIED BY VARIANT CHARACTER IN CAM INSTRUCTION.  
 8-63  
 MOVE OR SCAN VARIANTS. 8-9  
 RESTORE VARIANT AND INDICATORS. 8-98  
 STORE VARIANT AND INDICATORS. 8-94  
 VARIANT CHARACTER. 3-3 5-23  
 VIOLATIONS  
 VIOLATIONS OF STORAGE PROTECTION. 2-23  
 VISUAL  
 VISUAL INFORMATION PROJECTION DEVICES. 1-18  
 WORD  
 DEFINE CONSTANT WITH WORD MARK. 6-2  
 LOAD CHARACTERS TO A-FIELD WORD MARK. 8-56  
 MOVE CHARACTERS TO WORD MARK. 8-55  
 NUMERICAL REPRESENTATION OF DECIMAL WORD DATA. D-19  
 WRITE  
 WRITE PROTECT CAPABILITY. 1-14  
 ZERO AND ADD  
 ZERO AND ADD. 8-20  
 ZERO AND SUBTRACT  
 ZERO AND SUBTRACT. 8-22



HONEYWELL INFORMATION SYSTEMS

Publications Remarks Form\*

TITLE: SERIES 2000  
MODELS 2040 THROUGH 2070  
PROGRAMMERS' REFERENCE MANUAL

ORDER No.: JANUARY 1973  
DATED: AG28, REV. 0

ERRORS IN PUBLICATION:

[Empty box for reporting errors in publication]

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

[Empty box for providing suggestions for improvement to publication]

*(Please Print)*

FROM: NAME \_\_\_\_\_  
COMPANY \_\_\_\_\_  
TITLE \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

DATE: \_\_\_\_\_

CUT ALONG LINE

\*Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here.

CUT ALONG LINE

FIRST CLASS  
PERMIT NO. 39531  
WELLESLEY HILLS,  
MASS. 02181

**Business Reply Mail**  
Postage Stamp Not Necessary if Mailed in the United States

POSTAGE WILL BE PAID BY:

**HONEYWELL INFORMATION SYSTEMS**  
60 WALNUT STREET  
WELLESLEY HILLS, MASS. 02181

ATTN: PUBLICATIONS, MS 050

**Honeywell**

The Other Computer Company:  
**Honeywell**

HONEYWELL INFORMATION SYSTEMS

6764  
3473

Printed in U.S.A.

In the U.S.A.: 200 Smith Street, MS 061, Waltham, Massachusetts 02154  
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario

AG28, Rev. 0