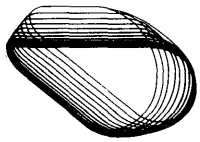


SYSTEM 2400

SOFTWARE

MACHINE CODE AND ASSEMBLY LANGUAGE

MOHAWK DATA SCIENCES CORP.

SYSTEM 2400

MACHINE CODE AND ASSEMBLY LANGUAGE

(Level 03, Revision 03 Only)

SECOND EDITION

CORPORATE HEADQUARTERS · UTICA, NEW YORK 13503

Form No. 1948-U774③ Mohawk Data Sciences Corp. Printed in U.S.A.

REVISION PAGE

EDITION/ ADDENDUM NUMBER	NUMBER OF SHEETS AFFECTED	FORM NUMBER AND DATE	SOFTWARE LEVEL SUPPORTED
EDITION 1	---------	M-1948-1271	Unknown
ADDENDUM 1	---------	M-1948-0372	Unknown
ADDENDUM 2	---------	M-1948-1072	Unknown
ADDENDUM 3	---------	M-1948-0173	03.00
ADDENDUM 4	---------	M-1948-0673	03.00
ADDENDUM 5	6	M-1948-1173	03.02
EDITION 2	---------	PM-1948-0774	03.03

REASON FOR CURRENT ADDENDUM/EDITION:

This edition combines edition 1 of this manual and edition 1 of the 2408 Instruction Set, Form No. PM-2571.

Change bars indicate the latest additions and corrections. An asterisk (*) accompanying the change bar indicates a deletion.

TABLE OF CONTENTS

TABLE OF CONTENTS
(cont'd)

TABLE OF CONTENTS
(cont'd)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(cont'd)

LIST OF TABLES

FOREWORD

This manual describes the instruction repertoire, format, and detailed information for programming the SYSTEM 2400 Processors in Machine Code and SYSTEM 2400 Assembly Language. For conventional purposes, the user should always program SYSTEM 2400 applications in Mohawk Data Language (MDL) or RPG II, both of which are fully supported by MDS.

This manual provides in-depth information to the experienced user regarding the machine-code instruction repertoire for the SYSTEM 2400 Processors. The Software Manual *SYSTEM 2400 Processor Programming in Machine Code* (Form No. M-2269) is prerequisite reading to this document. For the effective use of this manual, familiarity with the following publications will also be particularly helpful:

SYSTEM 2400 Processor Operator Control Panels Hardware Manual (Form No. M-2268)

2406 Systems Console Hardware Manual (Form No. M-1940)

The reader should be familiar with compatible data processing equipments and associated programming systems.

Users that elect to include the 2406 Systems Console within their system and to program applications without using MDL or RPG II can function satisfactorily with the set of machine-level instructions described herein. This document contains the instruction formats and detailed steps for their use within the following functional categories:

- Data move
- Branching
- Compare
- Test
- Input/output
- General Purpose
- Logical
- Binary Arithmetic
- Decimal Arithmetic
- Sequential Editing

- Interrupt
- External Execute
- Instruction Expansion Module A
- Instruction Expansion Module B

As a general rule, instructions preceded by an asterisk (*) may only be used with a 502 Processor. While the instructions not preceded by an asterisk may be used with a 501 or 502 Processor. For detailed information on which processor can execute each instructions, see Appendix E.

ABBREVIATIONS AND CONVENTIONS

The following abbreviations and conventions are used when describing the instruction format and presenting typical examples:

AR	Active Record
B	Buffer
I	Item
f	Denotes an "off" condition
IDT	Item Descriptor Table
L_i	Literal to be ignored
L_f	Fill literal
L_s	Sentinel literal
LSD	Least significant digit
M	Mask
MSB	Most significant bit
MSBY	Most significant byte
n	Null
o	Denotes an "on" condition
OC	Operation Code
OP1-4	Operands 1 through 4
PBIAS	Program Bias
PCB	Program Control Block
P_a	Pointer after an execution
P_b	Pointer before an execution
R	Record
s	Space
SDAT	Storage Descriptor Area Table
z	Zero
=	Equals
≠	Not equal to
>	Greater than
<	Less Than
≥	Equal to or Greater Than
≤	Equal to or Less Than
+	Plus
-	Minus

SUMMARY OF INSTRUCTIONS

BY FUNCTION

	Op Code		Page No.	Instruction	Format				
	Octal	Mnem.			OC	OP1	OP2	OP3	OP4
DATA MOVE	000	M	1-7	Move Item, Left-Align, No Fill	000	AR/I	AR/I		
	001	MR	1-8	Move Item, Right-Align, No Fill	001	AR/I	AR/I		
	003	MED	1-9	Move Item, Edit	003	AR/I	AR/I	AR/I	L
	004	MF	1-10	Move Item, Left-Align, Fill	004	AR/I	AR/I	L	
	005	MRF	1-11	Move Item, Right-Align, Fill	005	AR/I	AR/I	L	
	006	MJ	1-12	Move Item, Left-Justify, Fill	006	AR/I	AR/I	L	
	007	MRJ	1-13	Move Item, Right-Justify, Fill	007	AR/I	AR/I	L	
	050	MPK	1-14	Move, Pack	050	AR/I	AR/I		
	052	MUP	1-15	Move, Unpack	052	AR/I	AR/I		
	140	TRL	1-16	Translate Code	140	AR/I	AR/I	AR/I	
	141	ML	1-17	Move Literal	141	AR/I	L		
BRANCHING	020	NOP	1-19	No Operation	020	address			
	021	GGT	1-20	GOTO Greater Than	021	address			
	022	GLT	1-21	GOTO Less Than	022	address			
	023	GNE	1-22	GOTO Not Equal	023	address			
	024	GE	1-23	GOTO Equal	024	address			
	025	GNL	1-24	GOTO Not Less Than	025	address			
	026	GNG	1-25	GOTO Not Greater Than	026	address			
	027	G	1-26	GOTO Unconditionally	027	address			
	030	GD	1-27	GOTO On Designators	030	M	address		
	031	GS	1-28	GOTO On Switches	031	M	address		

* 502 Mode only

SUMMARY OF INSTRUCTIONS
BY FUNCTION
(continued)

	Op Code		Page No.	Instruction	Format				
	Octal	Mnem.			OC	OP1	OP2	OP3	OP4
BRANCHING	061	GBG	1-29	GOTO Binary Greater Than	061	AR/I	address		
	062	GBL	1-30	GOTO Binary Less Than	062	AR/I	address		
	063	GBN	1-31	GOTO Binary Non-Zero	063	AR/I	address		
	064	GBZ	1-32	GOTO Binary Zero	064	AR/I	address		
	065	GGBE	1-33	GOTO Binary $\geq$ Zero	065	AR/I	address		
	066	GLBE	1-34	GOTO Binary $\leq$ Zero	066	AR/I	address		
	071	GDG	1-35	GOTO Decimal Greater Than	071	AR/I	address		
	072	GDL	1-36	GOTO Decimal Less Than	072	AR/I	address		
	073	GDN	1-37	GOTO Decimal Non-Zero	073	AR/I	address		
	074	GDZ	1-38	GOTO Decimal Zero	074	AR/I	address		
	075	GGDE	1-39	GOTO Decimal $\geq$ Zero	075	AR/I	address		
	076	GLDE	1-40	GOTO Decimal $\leq$ Zero	076	AR/I	address		
	170	GCT	1-41	GOTO On Count	170	AR/I	address		
	172	GTB	1-42	GOTO Table (Indirect Branch)	172	AR/I	AR/I	L	
	173	GRT	1-44	GOTO Return (Branch)	173	B			
	176	GSB	1-45	GOTO Subroutine (Branch)	176	B	address		
COMPARE	044	CB	1-47	Compare Binary	044	AR/I	AR/I		
	046	CD	1-48	Compare Decimal	046	AR/I	AR/I		
	142	CAN	1-49	Compare Alphanumerics	142	AR/I	AR/I		
	144	CL	1-50	Compare Literal	144	AR/I	L		

* 502 Mode only

SUMMARY OF INSTRUCTIONS

BY FUNCTION

(continued)

	Op Code		Page No.	Instruction	Format				
	Octal	Mnem.			OC	OP1	OP2	OP3	OP4
TEST	040	TBS	1-52	Test Binary Sign	040	AR/I			
	042	TDS	1-53	Test Decimal Sign	042	AR/I			
	150	TI	1-54	Test Item	150	AR/I	AR/I		
	151	TL	1-55	Test Literal	151	AR/I	L		
	152	TM	1-56	Test Mask	152	AR/I	M		
	*153	TIM	1-57	Test Item Mask	153	AR/I	AR/I		
INPUT/OUTPUT	100	INS	1-60	Special In	100	AR/I	AR/I		
	104	EF	1-61	External Function On Channel	104	AR/I	AR/I	AR/I	
	105	OTS	1-62	Special Out	105	AR/I	AR/I		
	*106	EFS	1-63	External Function Special	106	AR/I	AR/I	AR/I	
	107	GA	1-64	GOTO On Active Channel	107	AR/I	address		
	110	STC	1-65	Store Channel Control Register	110	AR/I	AR/I		
	*111	STR	1-66	Store Channel Reverse	111	AR/I	AR/I		
	*112	INR	1-67	Initiate Input Reverse	112	AR/I	B		
	114	IN	1-68	Initiate Input On Channel	114	AR/I	B		
	115	OUT	1-69	Initiate Output On Channel	115	AR/I	B		
	*116	OTR	1-70	Initiate Output Reverse	116	AR/I	B		
GENERAL PURPOSE	000	RN	1-72	Rename	000	B/R	B/R		
	*124	STD	1-73	Store Designators	124	AR/I			
	*126	LD	1-74	Load Designators	126	AR/I			

* 502 Mode only

SUMMARY OF INSTRUCTIONS

BY FUNCTION

(continued)

	Op Code		Page		Format				
	Octal	Mnem.	No.	Instruction	OC	OP1	OP2	OP3	OP4
GENERAL PURPOSE	134	STT	1-75	Store Tally Counter	134	AR/I			
	136	LT	1-76	Load Tally Counter	136	AR/I			
	143	H	1-77	Halt	143				
	146	SDI	1-78	Set Display Indicators	146	AR/I			
	147	GAP	1-79	No Operation-Leave Gap	147				
	156	CDI	1-80	Clear Display Indicators	156	AR/I			
	161	LSP	1-81	Load Storage Descriptor Pointer	161	SDP			
	165	LR1	1-82	Load Active Record 1	165	R			
	171	LR2	1-82	Load Active Record 2	171	R			
	175	LR3	1-82	Load Active Record 3	175	R			
LOGICAL	160	X	1-84	OR (Exclusive)	160	AR/I	AR/I	AR/I	
	162	RCK	1-85	Longitudinal Redundancy Check	162	AR/I	AR/I		
	164	O	1-86	OR (Inclusive)	164	AR/I	AR/I	AR/I	
	166	N	1-87	Logical AND	166	AR/I	AR/I	AR/I	
BINARY ARITHMETIC	041	AB	1-89	Add Binary	041	AR/I	AR/I	AR/I	
	045	SB	1-90	Subtract Binary	045	AR/I	AR/I	AR/I	
	051	ALB	1-91	Add Literal Binary	051	AR/I	AR/I	L	
	055	SLB	1-92	Subtract Literal Binary	055	AR/I	AR/I	L	

* 502 Mode only

SUMMARY OF INSTRUCTIONS
BY FUNCTION
(continued)

	Op Code		Page		Format				
	Octal	Mnem.	No.	Instruction	OC	OP1	OP2	OP3	OP4
DECIMAL ARITHMETIC	043	A	1-94	Add Decimal	043	AR/I	AR/I	AR/I	
	047	S	1-95	Subtract Decimal	047	AR/I	AR/I	AR/I	
	053	AL	1-96	Add Literal Decimal	053	AR/I	AR/I	L	
	057	SL	1-97	Subtract Literal Decimal	057	AR/I	AR/I	L	
SEQUENTIAL EDITING	014	CP	1-100	Compress Item, Left-Align, Fill	014	AR/I	AR/I	L_i	L_f
	015	CPR	1-101	Compress Item, Right-Align, Fill	015	AR/I	AR/I	L_i	L_f
	120	APR	1-102	Append, Right-Eliminate	120	AR/I	B	L	
	121	APA	1-103	Append, Advance	121	AR/I	B		
	122	APE	1-105	Append, Left-Eliminate	122	AR/I	B	L	
	130	EXV	1-107	Extract Variable Length Item, Fill	130	B	AR/I	L_s	L_f
	131	EXP	1-111	Extract Previous Item	131	B	AR/I		
	132	EX	1-112	Extract Item	132	B	AR/I		
	133	EXA	1-114	Extract Item, Advance	133	B	AR/I		
INTERRUPT	113	GSI	1-117	GOTO On Service Request	113	AR/I	address		
	117	GCI	1-118	GOTO On Channel Interrupt	117	AR/I	address		
	154	SWS	1-119	Swap States	154				
	155	SIL	1-120	Set Interrupt Lockout	155				
	157	CIL	1-121	Clear Interrupt Lockout	157				
	174	IM	1-122	Interrupt Mask	174	AR/I			
	177	GIR	1-124	Interrupt Branch GOTO	177	B			

* 502 Mode only

SUMMARY OF INSTRUCTIONS
BY FUNCTION
(continued)

	Op Code		Page		Format				
	Octal	Mnem.	No.	Instruction	OC	OP1	OP2	OP3	OP4
EXTERNAL EXECUTE INSTRUCTIONS OC - 145	004	LC	1-127	Load Delta Clock	004	AR/I	----	----	
	014	SEE	1-128	Store External Instruction Error	014	----	----	AR/I	
	015	SCE	1-131	Store Channel Parity Error	015	----	----	AR/I	
	020	MB	1-132	Multiply Binary	020	AR/I	AR/I	AR/I	
	021	MLB	1-133	Multiply Literal Binary	021	AR/I	----	AR/I	L
	022	DB	1-134	Divide Binary	022	AR/I	AR/I	AR/I	
	023	DLB	1-135	Divide Literal Binary	023	AR/I	----	AR/I	L
	024	MD	1-136	Multiply Decimal	024	AR/I	AR/I	AR/I	
	025	MLD	1-137	Multiply Literal Decimal	025	AR/I	----	AR/I	L
	026	DD	1-138	Divide Decimal	026	AR/I	AR/I	AR/I	
	027	DLD	1-139	Divide Literal Decimal	027	AR/I	----	AR/I	L
	030	BTD	1-140	Binary to Decimal	030	AR/I	----	AR/I	
	031	DTB	1-141	Decimal to Binary	031	AR/I	----	AR/I	
	034	SDR	1-142	Store Decimal Remainder	034	----	----	AR/I	
	035	SBR	1-143	Store Binary Remainder	035	----	----	AR/I	

* 502 Mode only

SUMMARY OF INSTRUCTIONS

BY FUNCTION

(continued)

	Op Code		Page No.	Instruction	Format				
	Octal	Mnem.			OC	OP1	OP2	OP3	OP4
INSTRUCTION EXPANSION MODULE B	105	SVP or SAP	1-149	Save (P)	105	AR/I 002	AR/I 002		
	100	SRP	1-150	Store (P)	100	AR/I 002	AR/I		
INSTRUCTION EXPANSION MODULE B	105	ORE	1-153	OR (Exclusive)	105	AR/I 001	AR/I 001		
	105	AND	1-154	Logical AND	105	AR/I 001	AR/I 002		
	105	ORI	1-156	OR (Inclusive)	105	AR/I 001	AR/I 004		
	105	LRC	1-158	Longitudinal Redundancy Check	105	AR/I 001	AR/I 001		
	105	EMA	1-160	Enter Module Accumulator	105	AR/I 001	AR/I 050		
	100	SMA	1-161	Store Module Accumulator	100	AR/I 001	AR/I		

* 502 Mode only

SUMMARY OF INSTRUCTIONS
BY OCTAL SEQUENCE

Octal	Mnemonic	Page No.	Instruction
000	RN	1-72	Rename
000	M	1-7	Move Item, Left-Align, No Fill
001	MR	1-8	Move Item, Right-Align, No Fill
003	MED	1-9	Move Item, Edit
004	MF	1-10	Move Item, Left-Align, Fill
005	MRF	1-11	Move Item, Right-Align, Fill
006	MJ	1-12	Move Item, Left-Justify, Fill
007	MRJ	1-13	Move Item, Right-Justify, Fill
014	CP	1-99	Compress Item, Left-Align, Fill
015	CPR	1-10	Compress Item, Right-Align, Fill
020	NOP	1-19	No Operation
021	GGT	1-20	GOTO Greater Than
022	GLT	1-21	GOTO Less Than
023	GNE	1-22	GOTO Not Equal
024	GE	1-23	GOTO Equal
025	GNL	1-24	GOTO Not Less Than
026	GNG	1-25	GOTO Not Greater Than
027	G	1-26	GOTO Unconditionally
030	GD	1-27	GOTO On Designators
031	GS	1-28	GOTO On Switches
040	TBS	1-52	Test Binary Sign
041	AB	1-89	Add Binary
042	TDS	1-53	Test Decimal Sign
043	A	1-94	Add Decimal

* 502 Mode only.

SUMMARY OF INSTRUCTIONS
BY OCTAL SEQUENCE
(continued)

Octal	Mnemonic	Page No.	Instruction
044	CB	1-47	Compare Binary
045	SB	1-90	Subtract Binary
046	CD	1-48	Compare Decimal
047	S	1-95	Subtract Decimal
*050	MPK	1-14	Move, Pack
051	ALB	1-91	Add Literal Binary
*052	MUP	1-15	Move, Unpack
053	AL	1-96	Add Literal Decimal
055	SLB	1-92	Subtract Literal Binary
057	SL	1-97	Subtract Literal Decimal
*061	GBG	1-29	GOTO Binary Greater Than
*062	GBL	1-30	GOTO Binary Less Than
*063	GNB	1-31	GOTO Binary Non-Zero
*064	GBZ	1-32	GOTO Binary Zero
*065	GGBE	1-33	GOTO Binary $\geq$ Zero
*066	GLBE	1-34	GOTO Binary $\leq$ Zero
*071	GDG	1-35	GOTO Decimal Greater Than
*072	GDL	1-36	GOTO Decimal Less Than
*073	GDN	1-37	GOTO Decimal Non-Zero
*074	GDZ	1-38	GOTO Decimal Zero
*075	GGDE	1-39	GOTO Decimal $\geq$ Zero
*076	GLDE	1-40	GOTO Decimal $\leq$ Zero
100	INS	1-60	Special In
100	SRP	1-150	Store (P)

* 502 Mode only.

SUMMARY OF INSTRUCTIONS
BY OCTAL SEQUENCE
(continued)

Octal	Mnemonic	Page No.	Instruction
100	SMA	1-161	Store Module Accumulator
104	EF	1-61	External Function On Channel
105	OTS	1-62	Special Out
105	SAP or SVP	1-149	Save (P)
105	ORE	1-153	OR (Exclusive)
105	AND	1-154	Logical AND
105	ORI	1-156	OR (Inclusive)
105	LRC	1-158	Longitudinal Redundancy Check
105	EMA	1-160	Enter Module Accumulator
*106	EFS	1-63	External Function Special
107	GA	1-64	GOTO On Active Channel
110	STC	1-65	Store Channel Control Register
*111	STR	1-66	Store Channel Reverse
*112	INR	1-67	Initiate Input Reverse
*113	GSI	1-117	GOTO On Service Request
114	IN	1-68	Initiate Input On Channel
115	OUT	1-69	Initiate Output On Channel
*116	OTR	1-70	Initiate Output Reverse
*117	GCI	1-118	GOTO On Channel Interrupt
120	APR	1-102	Append, Right-Eliminate
121	APA	1-103	Append, Advance
122	APE	1-105	Append, Left-Eliminate

* 502 Mode only.

SUMMARY OF INSTRUCTIONS
BY OCTAL SEQUENCE
(continued)

Octal	Mnemonic	Page No.	Instruction
124	STD	1-73	Store Designators
126	LD	1-74	Load Designators
130	EXV	1-107	Extract Variable Length Item, Fill
131	EXP	1-111	Extract Previous Item
132	EX	1-112	Extract Item
133	EXA	1-114	Extract Item, Advance
134	STT	1-75	Store Tally Counter
136	LT	1-76	Load Tally Counter
140	TRL	1-16	Translate Code
141	ML	1-17	Move Literal
142	CAN	1-49	Compare Alphanumerics
143	H	1-77	Halt
144	CL	1-50	Compare Literal
*145 (004)	LC	1-127	Load Delta Clock
*145 (014)	SEE	1-128	Store External Instruction Error
*145 (015)	SCE	1-131	Store Channel Parity Error
*145 (020)	MB	1-132	Multiply Binary
*145 (021)	MLB	1-133	Multiply Literal Binary
*145 (022)	DB	1-134	Divide Binary
*145 (023)	DLB	1-135	Divide Literal Binary

* 502 Mode only.

SUMMARY OF INSTRUCTIONS
BY OCTAL SEQUENCE
(continued)

Octal	Mnemonic	Page No.	Instruction
*145 (024)	MD	1-136	Multiply Decimal
*145 (025)	MLD	1-137	Multiply Literal Decimal
*145 (026)	DD	1-138	Divide Decimal
*145 (027)	DLD	1-139	Divide Literal Decimal
*145 (030)	BTD	1-140	Binary to Decimal
*145 (031)	DTB	1-141	Decimal to Binary
*145 (034)	SDR	1-142	Store Decimal Remainder
*145 (035)	SBR	1-143	Store Binary Remainder
146	SDI	1-78	Set Display Indicators
*147	GAP	1-79	No Operation-Leave Gap
150	TI	1-54	Test Item
151	TL	1-55	Test Literal
152	TM	1-56	Test Mask
*153	TIM	1-57	Test Item Mask
*154	SWS	1-119	Swap States
*155	SIL	1-120	Set Interrupt Lockout
156	CDI	1-80	Clear Display Indicators
*157	CIL	1-121	Clear Interrupt Lockout
*160	X	1-84	OR (Exclusive)
*161	LSP	1-81	Load Storage Descriptor Pointer

* 502 Mode only.

SUMMARY OF INSTRUCTIONS
BY OCTAL SEQUENCE
(continued)

Octal	Mnemonic	Page No.	Instruction
*162	RCK	1-85	Longitudinal Redundancy Check
*164	O	1-86	OR (Inclusive)
165	LR1	1-82	Load Active Record 1
*166	N	1-87	Logical AND
*170	GCT	1-41	GOTO On Count
171	LR2	1-82	Load Active Record 2
*172	GTB	1-42	GOTO Table (Indirect Branch)
*173	GRT	1-44	GOTO Return (Branch)
*173	IM	1-122	Interrupt Mask
175	LR3	1-82	Load Active Record 3
*176	GSB	1-45	GOTO Subroutine (Branch)
*177	GIR	1-124	Interrupt Branch GOTO

* 502 Mode only.

SECTION I
SYSTEM 2400 MACHINE CODE

This section describes the total SYSTEM 2400 machine - level instruction set.

Each instruction has a variable-length format, with

- an op code, to specify the operation to be performed,

and

- zero-to-four operands, to specify the records, items, buffers, etc., to be operated upon.

The op code and operands are each 1-byte long, with the op code first,

followed by the operands arranged in the prescribed sequence for a given instruction. Op codes and operands are expressed in octal notation.

CONDITION DESIGNATORS

Condition Designators denote

- =, ≠, >, <

and

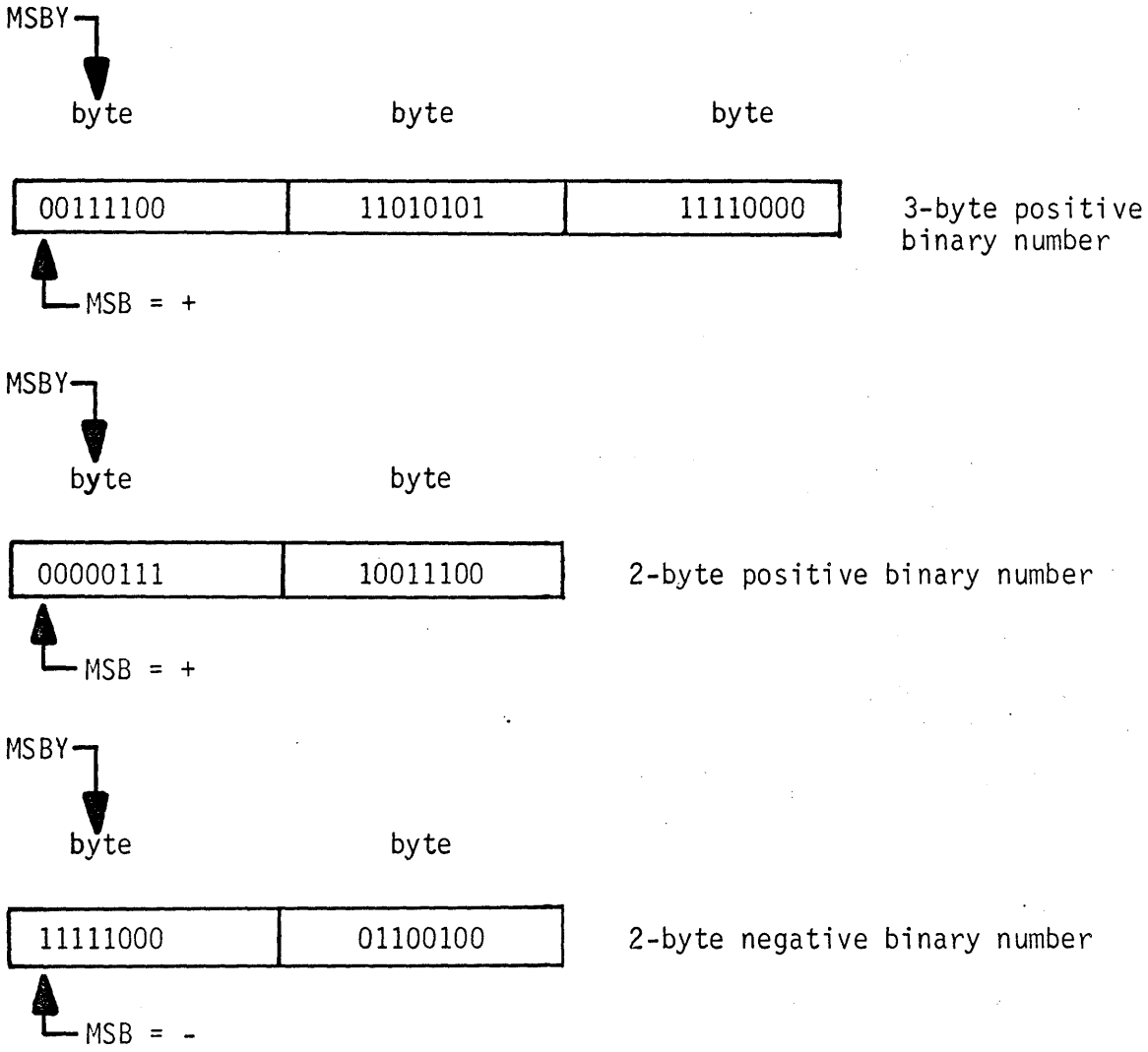
- arithmetic and abnormal-edit errors.

The Compare and the Test instructions establish conditions and set the appropriate internal condition designators, which are used by the Branching instructions to branch from the instruction execution sequence. Many of the Sequential Editing instructions set the Equal designator to indicate when the end of a data trnasfer to or from a working buffer has occurred. They also set the abnormal-edit designator to indicate when the receiving area in a data transfer is too small. Both the arithmetic overflow and the arithmetic error designators denote errors caused by a Binary or a Decimal Arithmetic instruction.

The designators remain set throughout program execution until they are reset by subsequent instructions.

BINARY NUMBER SIGNS

The sign of a binary number is indicated by the most significant bit (MSB) of the most significant byte (MSBY) of the item: 0 = + and 1 = -.

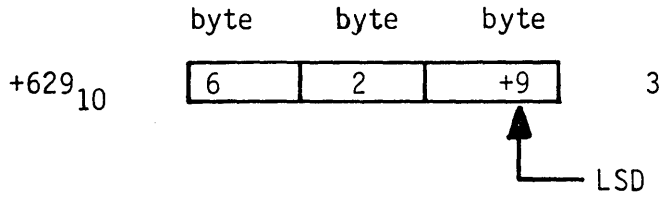


All the negative binary numbers coded into the binary-oriented instructions *must be* in the two's complement form. To convert a binary number

	0 0 1 0 1 0 1 0	into two's complement notation,
change each bit	1 1 0 1 0 1 0 1	to its opposite state (- one's complement)
and add 1	1	to produce its
two's complement.	1 1 0 1 0 1 1 0	

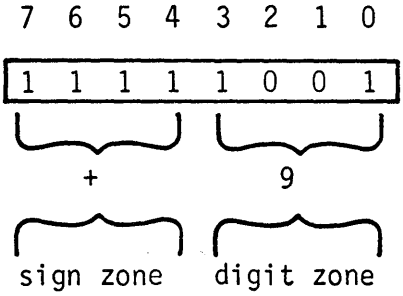
DECIMAL NUMBER SIGNS

In decimal arithmetic, the sign of a number is indicated by the sign of the least significant digit, as shown below.



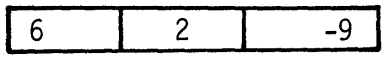
3-byte positive decimal number

The plus sign for the digit is expressed as "1111" in the sign zone (left half) of the LSD (9_{10}) byte (position 7654).



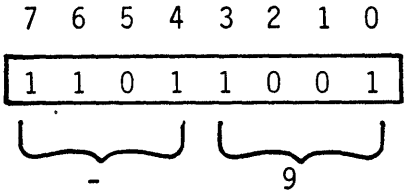
bit configuration for 9_{10}

For a minus number, such as



-629$_{10}$ 3-byte negative decimal number,

The minus sign for the digit is expressed as 1101 in the sign zone of the LSD byte, as shown below.



bit configuration for -9_{10}

In the SYSTEM 2400 EBCDIC character set, the negative numbers correspond to the binary configurations for the letters J. through R, as noted below.

Binary	or	Octal	=	Number	and Corresponds to
1101 0001		321		-1	J
1101 0010		322		-2	K
1101 0011		323		-3	L
1101 0100		324		-4	M
1101 0101		325		-5	N
1101 0110		326		-6	O
1101 0111		327		-7	P
1101 1000		330		-8	Q
1101 1001		331		-9	R

FUNCTIONS AND OPERATIONS

Each machine-code instruction is functionally categorized into one of the following:

- Data Move (1-6): transfer a copy of a complete data string.
- Branching (1-18): conditional or unconditional branching from the normal program sequence.
- Compare (1-46): compare data strings.
- Test (1-51): test for the sign or identity of an item.
- Input/Output (1-59): initiate and control input/output operations.
- General Purpose (1-71): perform various operational functions.
- Logical (1-83): AND, OR, or Exclusive OR.
- Binary Arithmetic (1-88): add and subtract in binary
- Decimal Arithmetic (1-93): add and subtract in decimal.
- Sequential Editing (1-98): manipulate data as it is transferred between peripherals.
- Interrupt (1-116): interpret, control and process events that divert the processor from main program execution.
- External Execute (1-125): instructions added by hardware expansion modules to provide the following functions:
  a. Multiply and Divide
  b. Binary/Decimal Conversion
  c. Delta Clock
  d. Channel Parity Error Determination
- Instruction Expansion Modules A and B (1-144).

Individual instructions are described as follows:

FUNCTIONAL CATEGORY
Descriptive Name of Instruction
Mnemonic Op Code = ABC
Octal Op Code = 123

PURPOSE: Brief explanation of what the instruction does.

FORMAT: Format of the instruction.

OPERATION: Operation of the instruction and the programming details.

EXAMPLE: Typical example that uses the instruction.

The Data Move instructions move a copy of OP1 item to OP2 item, character-by-character; OP1 item remains unchanged. The move is terminated when all of OP1 item is copied or when OP2 item is full. When OP2 item is longer than OP1 item, the excess positions are unchanged by the move operation, unless a fill operation is specified.

Left-justification means omit copying the *leading* nulls, spaces, and zeros and left-align the entry into the OP2 item with a character fill.

Right-justification means omit copying the *trailing* spaces and nulls (not zeros) and right-align the entry into the OP2 item with a character fill.

The Data Move instructions include the following:

- Move Item, Left Align, No Fill (1-7)
- Move Item, Right Align, No Fill (1-8)
- Move Item, Edit (1-9)
- Move Item, Left Align, Fill (1-10)
- Move Item, Right Align, Fill (1-11)
- Move Item, Left Justify, Fill (1-12)
- Move Item, Right Justify, Fill (1-13)
- Move, Pack (1-14)
- Move, Unpack (1-15)
- Translate Code (1-16)
- Move Literal (1-17)

DATA MOVE

Move Item, Left-Align, No Fill

Mnemonic Op Code = M

Octal Op Code = 000

PURPOSE: To copy the data from one item to another, with the content of the receiving item left-aligned.

FORMAT:

OC	OP1	OP2
000	AR/I	AR/I

OPERATION: A copy of the contents of the OP1 item is moved left-aligned into the OP2 item. If OP2 is larger than OP1, the remaining characters are unaffected. If OP1 is larger than OP2, the extra OP1 characters at the right are truncated.

EXAMPLES: No. 1

OC	OP1	OP2
000	102	211

OP1	AsBCDEF	Item 2 of Active Record 1
OP2	RICHARDsG	Item 11 of Active Record 2
OP2 after	AsBCDEFsG	Item 11 of Active Record 2
OP1 after	AsBCDEF	Item 2 of Active Record 1

No. 2

OC	OP1	OP2
000	102	211

OP1	AsBCDEF	Item 2 of Active Record 1
OP2	PQRS	Item 10 of Active Record 2
OP2 after	AsBC	Item 10 of Active Record 2
OP1 after	AsBCDEF	Item 2 of Active Record 1

DATA MOVE

Move Item, Right-Align, No Fill

Mnemonic Op Code = MR

Octal Op Code = 001

PURPOSE: To copy the data from one item to another, with the content of the receiving item right-aligned.

FORMAT:

OC	OP1	OP2
001	AR/I	AR/I

OPERATION: A copy of the contents of the OP1 item is moved right-aligned into the OP2 item. If OP2 is larger than OP1, the remaining characters are unaffected. If OP1 is larger than OP2, the extra OP1 characters at the left are truncated.

EXAMPLES: No. 1

OC	OP1	OP2
001	102	211

OP1	AsBCDEF	Item 2 of Active Record 1
OP2	RICHARDsG	Item 11 of Active Record 2
OP2 after	RIAsBCDEF	Item 11 of Active Record 2
OP1 after	AsBCDEF	Item 2 of Active Record 1

No. 2

OC	OP1	OP2
001	102	210

OP1	AsBCDEF	Item 2 of Active Record 2
OP2	PQRS	Item 10 of Active Record 2
OP2 after	CDEF	Item 10 of Active Record 2
OP1 after	AsBCDEF	Item 2 of Active Record 1

DATA MOVE

Move, Edit

Mnemonic Op Code = MED

Octal Op Code = *003

PURPOSE: To copy the data from one item right-aligned to another item under the control of a third mask item. Any remaining characters are replaced with the specified fill character.

FORMAT:

OC	OP1	OP2	OP3	OP4
003	AR/I	AR/I	AR/I	L

OPERATION: A copy of the contents of the OP1 item is moved right-aligned into the OP3 item under control of the OP2 mask item. Every OP2 mask character that is equal to a null allows an OP1 item character to be moved into OP3. Every OP2 character that is not a null is itself moved to the OP3 item. After the move of all OP1 characters, leading zeros, spaces, commas, and nulls in the OP3 item are replaced by the fill literal specified in OP4. The low-order digit of the result has its four bits replaced with all ones (positive sign convention). The abnormal edit designator is set if the OP2 or OP3 item is smaller than the OP1 item.

EXAMPLE:

OC	OP1	OP2	OP3	OP4
003	101	102	212	134

OP1	0 0 0 4 9 9 8 4 2	Item 1 of Active Record 1
OP2	n n n , n n n , n n n . n n	Item 2 of Active Record 1
OP3 before	X X X X X X X X X X	Item 12 of Active Record 2
OP4	*	Literal asterisk (134 in EBCDIC)
OP3 after	* * 4 , 9 9 8 . 4 2	Item 12 of Active Record 2

DATA MOVE

Move Item, Left-Align, Fill

Mnemonic Op Code = MF

Octal Op Code = 004

PURPOSE: To copy the data from one item to another, with the contents of the receiving item left-aligned and any remaining characters replaced by a specified fill character.

FORMAT:

OC	OP1	OP2	OP3
004	AR/I	AR/I	L

OPERATION: A copy of the contents of the OP1 item is moved left-aligned into the OP2 item. If OP2 is larger than OP1, the remaining characters at the right are replaced by the OP3 character. If OP1 is larger than OP2, the extra OP1 characters at the right are truncated.

EXAMPLE:

OC	OP1	OP2	OP3
004	124	215	000

OP1	GHIJ	Item 24 of Active Record 1
OP2	******	Item 15 of Active Record 2
OP2 after	GHIJnn	Item 15 of Active Record 2
OP1 after	GHIJ	Item 24 of Active Record 1

DATA MOVE

Move Item, Right-Align, Fill

Mnemonic Op Code = MRF

Octal Op Code = 005

PURPOSE: To copy the data from one item to another, with the contents of the receiving item right-aligned and any remaining characters replaced by a specified fill character.

FORMAT:

OC	OP1	OP2	OP3
005	AR/I	AR/I	L

OPERATION: A copy of the contents of the OP1 item is moved right-aligned into the OP2 item. If OP2 is larger than OP1, the remaining characters at the left are replaced by the OP3 character. If OP1 is larger than OP2, the extra OP1 characters at the left are truncated.

EXAMPLE:

OC	OP1	OP2	OP3
005	125	215	133

OP1	4.50	Item 25 of Active Record 1
OP2	******	Item 15 of Active Record 2
OP2 after	SS4.50	Item 15 of Active Record 2
OP1 after	4.50	Item 25 of Active Record 1

DATA MOVE

Move Item, Left-Justify, Fill

Mnemonic Op Code = MJ

Octal Op Code = 006

PURPOSE: To copy the data from one item to another, with the content of the receiving item left-justified and any remaining characters replaced by a specified fill character.

FORMAT:

OC	OP1	OP2	OP3
006	AR/I	AR/I	L

OPERATION: A copy of the contents of the OP1 item is moved into the OP2 item and left-justified. Left-justification means that the leading (leftmost) nulls, spaces, and zeros in the OP1 item are *not* moved to the OP2 item and the remaining characters are left-aligned. Any remaining characters at the right of OP2 are replaced by the OP3 character. If OP1 is larger than OP2, the extra characters at the right are truncated.

EXAMPLE:

OC	OP1	OP2	OP3
006	117	212	116

OP1	sznnAsBCn	Item 17 of Active Record 1
OP2	123456789123	Item 12 of Active Record 2
OP2 after	AsBCn+++++++	Item 12 of Active Record 2
OP1 after	sznnAsBCn	Item 17 of Active Record 1

DATA MOVE

Move Item, Right Justify, Fill

Mnemonic Op Code = MRJ

Octal Op Code = 007

PURPOSE: To copy the data from one item to another, with the content of the receiving item right-justified and any remaining characters replaced by a specified fill character.

FORMAT:

OC	OP1	OP2	OP3
007	AR/I	AR/I	L

OPERATION: A copy of the contents of the OP1 item is moved into the OP2 item and right-justified. Right-justification means that the trailing (rightmost) nulls and spaces in the OP1 item are *not* moved to the OP2 item and the remaining characters are right-aligned. Any remaining characters at the left of OP2 are replaced by the OP3 character. If OP1 is larger than OP2, the extra characters at the right are truncated.

EXAMPLE:

OC	OP1	OP2	OP3
007	120	212	116

OP1	nsAB.Csznns	Item 20 of Active Record 1
OP2	123456789123	Item 12 of Active Record 2
OP2 after	++++nsAB.Csz	Item 12 of Active Record 2
OP1 after	nsAB.Csznns	Item 20 of Active Record 1

DATA MOVE

Move, Pack

Mnemonic Op Code = MPT

Octal Op Code = *050

PURPOSE: To extract the digit portions from a source item and pack them into a destination item, eliminating the sign zones of all source bytes except the rightmost. Any remaining destination item positions are filled with binary zeros.

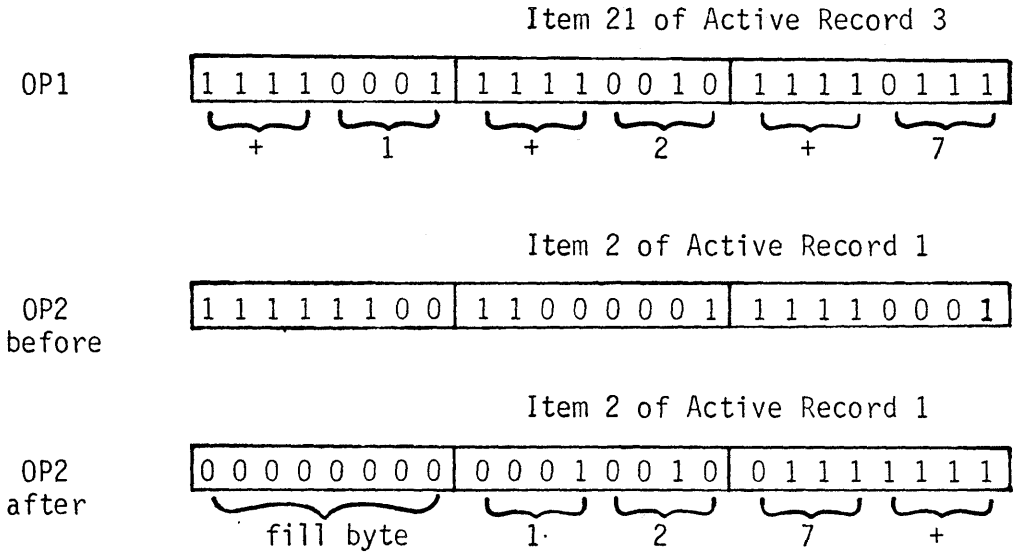
FORMAT:

OC	OP1	OP2
050	AR/I	AR/I

OPERATION: The decimal information contained in OP1 is moved into OP2. The sign zone changes place with the digit zone and the resultant byte is moved right-aligned into OP2. The following transfers move only the digit zone into OP2, thereby packing the digit zones of two source bytes into one destination byte. After the contents of OP1 are packed into OP2, any remaining OP2 bytes are filled with binary zeros.

EXAMPLES:

OC	OP1	OP2
050	321	102



DATA MOVE

Move, Unpack

Mnemonic Op Code = MUP

Octal Op Code = *052

PURPOSE: To unpack packed decimal information from a source item right-aligned into a destination item. Data is moved into the destination item in zoned form. Any remaining OP2 item positions are filled with binary zeros.

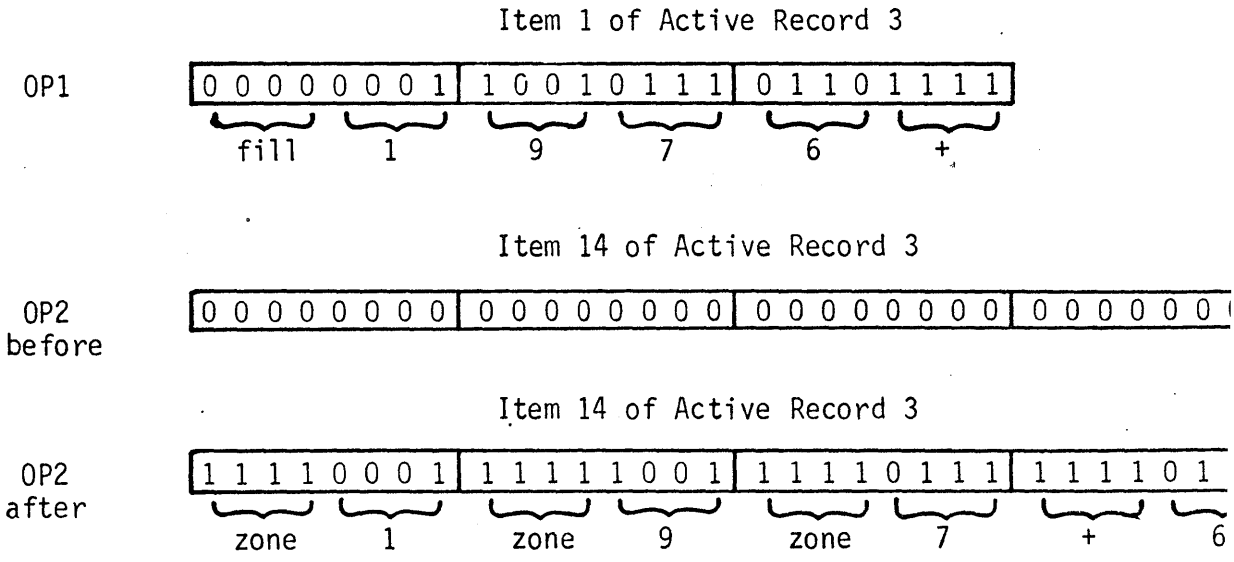
FORMAT:

OC	OP1	OP2
052	AR/I	AR/I

OPERATION: The OP1 item contains packed decimal data. The numerics and the sign of the rightmost OP1 byte are stored into the rightmost OP2 byte after the sign zone and the digit zone have been switched around. For the second OP1 byte, the four zone bits are added to the four lower bits and stored into OP2. Then four zone bits are added to the remaining upper four bits and stored into OP2. In this way, each OP1 byte produces two OP2 bytes. After the contents of OP1 are unpacked into OP2, any remaining byte positions in OP2 are filled with binary zeros.

EXAMPLE:

OC	OP1	OP2
052	301	314



DATA MOVE

Translate Code

Mnemonic Op Code = TRL

Octal Op Code = 140

PURPOSE: To convert characters from one code, such as EBCDIC, to another code, such as USASCII.

FORMAT:

OC	OP1	OP2	OP3
140	AR/I	AR/I	AR/I

OPERATION: Each character of the OP1 item is sequentially translated into its equivalent binary code from the table of character codes in OP2 and sequentially entered into the OP3 item. The example below illustrates the translation process.

EXAMPLE:

OC	OP1	OP2	OP3
140	110	201	322

Item 10 of Active Record 1 is translated into Item 22 of Active Record 3 by means of the conversion table in Item 1 of Active Record 2.

OP1 $A_1\ C_1\ T_1$ — The OP1 item contains the characters ACT in code 1.

OP2 $A_2\ B_2\ C_2\ \ldots\ Z_2\ \ldots$ — The binary value for each OP1 character is added to the first location of OP2, giving the location of the equivalent character in binary of code 2, which is then transferred to OP3.

OP3 $A_2\ C_2\ T_2$ **** — The code 2 equivalent is left-aligned and any extra characters are unaffected.

DATA MOVE

Move Literal

Mnemonic Op Code = ML

Octal Op Code = 141

PURPOSE: To fill an item with a specified character.

FORMAT:

OC	OP1	OP2
141	AR/I	L

OPERATION: The OP2 literal is entered into each position of the OP1 item.

EXAMPLE:

OC	OP1	OP2
141	103	133

OP1 [ABCDEF] Item 3 of Active Record 1

OP2 [$]

OP1 after [$$$$$$] Item 3 of Active Record 1

Literal Character: EBCDIC 133 = $

(See Appendix C)

PROGRAM STARTING ADDRESS (P-BIAS)

All addresses referenced in the "branch to" operands are *relative* to the first address of the program instructions. Core-memory assignments are made after the program is written and the starting address of the program is stored in the PCB. The relative address of each branching instruction is added to the program starting address (P-Bias) during instruction execution.

The Branching instructions include the following:

- No Operation (1-19)
- GOTO Greater Than (1-20)
- GOTO Less Than (1-21)
- GOTO Not Equal (1-22)
- GOTO Equal (1-23)
- GOTO Not Less Than (1-24)
- GOTO Not Greater Than (1-25)
- GOTO Unconditionally (1-26)
- GOTO On Designators (1-27)
- GOTO On Switches (1-28)
- GOTO Binary Greater Than (1-29)
- GOTO Binary Less Than (1-30)
- GOTO Binary Non-Zero (1-31)
- GOTO Binary Zero (1-32)
- GOTO Binary $\geq$ Zero (1-33)
- GOTO Binary $\leq$ Zero (1-34)
- GOTO Decimal Greater Than (1-35)
- GOTO Decimal Less Than (1-36)
- GOTO Decimal Non-Zero (1-37)
- GOTO Decimal Zero (1-38)
- GOTO Decimal $\geq$ Zero (1-39)
- GOTO Decimal $\leq$ Zero (1-40)
- GOTO On Count (1-41)
- GOTO Table (1-42)
- GOTO Return (Branch) (1-44)
- GOTO Subroutine (Branch) (1-45)

BRANCHING

No Operation

Mnemonic Op Code = NOP

Octal Op Code = 020

PURPOSE: No operation. The instruction sequence is not changed.

FORMAT:

OC	Branch to
020	Address

OPERATION: When the instruction is executed, no change to indicator lights, designators, data, or instruction sequence is made.

This instruction may be used temporarily in a sequence of instructions, where it will have no effect. The actual op code may be changed later to 'G', 'GE', 'GNE', 'GLT', or 'GGT', so that subsequent execution of the instruction may actually cause a branch to the specified address.

BRANCHING

GOTO Greater Than

Mnemonic Op Code = GGT

Octal Op Code = 021

PURPOSE: To cause a branch from the instruction execution sequence when a "greater than" condition exists.

FORMAT:

OC	Branch to
021	Address

OPERATION: When this instruction is executed and the GREATER THAN condition designator *is set and* the EQUAL condition designator *is not set*, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.

EXAMPLE:

OC	Branch to	
021	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If conditions are met, then execution is resumed at location 011-234; otherwise, it continues with 010-165.

BRANCHING

GOTO Less Than

Mnemonic Op Code = GLT

Octal Op Code = 022

PURPOSE: To cause a branch from the instruction execution sequence when a "less than" condition exists.

FORMAT:

OC	Branch to
022	Address

OPERATION: When this instruction is executed and the GREATER THAN *and* the EQUAL condition designators *are not set*, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.

EXAMPLE:

OC	Branch to	
022	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If conditions are met, then execution is resumed at location 011-234; otherwise, it continues with 010-165.

BRANCHING

GOTO Not Equal

Mnemonic Op Code = GNE

Octal Op Code = 023

PURPOSE: To cause a branch from the instruction execution sequence when a "not equal" condition exists.

FORMAT:

OC	Branch to
023	Address

OPERATION: When this instruction is executed and the EQUAL condition designator *is not set*, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.

EXAMPLE:

OC	Branch to	
023	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If conditions are met, then execution is resumed at location 011-234; otherwise, it continues with 010-165.

BRANCHING

GOTO Equal

Mnemonic Op Code = GE

Octal Op Code = 024

PURPOSE: To cause a branch from the instruction execution sequence when an "equal" condition exists.

FORMAT:

OC	Branch to
024	Address

OPERATION: When this instruction is executed and the EQUAL designator *is set*, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.

EXAMPLE:

OC	Branch to	
024	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If conditions are met, then execution is resumed at location 011-234; otherwise, it continues with 010-165.

BRANCHING

GOTO Not Less Than

Mnemonic Op Code = GNL

Octal Op Code = 025

PURPOSE: To cause a branch from the instruction execution sequence when a "not less than" condition exists.

FORMAT:

OC	Branch to
025	Address

OPERATION: When this instruction is executed and the EQUAL condition designator is set, or the EQUAL condition designator is not set and the GREATER THAN condition designator is set, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.

EXAMPLE:

OC	Branch to	
025	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-234; otherwise, it continues with 010-165.

BRANCHING

GOTO Not Greater Than

Mnemonic Op Code = GNG

Octal Op Code = 026

PURPOSE: To cause a branch from the instruction execution sequence when a "not greater than" condition exists.

FORMAT:

OC	Branch to
026	Address

OPERATION: When this instruction is executed and the EQUAL condition designator is set, or both the EQUAL and GREATER THAN condition designators are not set, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.

EXAMPLE:

OC	Branch to	
026	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-234; otherwise, it continues with 010-165.

BRANCHING

GOTO Unconditionally

Mnemonic Op Code = G

Octal Op Code = 027

PURPOSE: To cause a branch from the instruction execution sequence when this instruction is executed.

FORMAT:

OC	Branch to
027	Address

OPERATION: When this instruction is executed, the instruction execution sequence is transferred to the "branch to" address.

EXAMPLE:

OC	Branch to	
027	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. Execution is resumed at location 011-234.

BRANCHING

GOTO On Designators

Mnemonic Op Code = GD

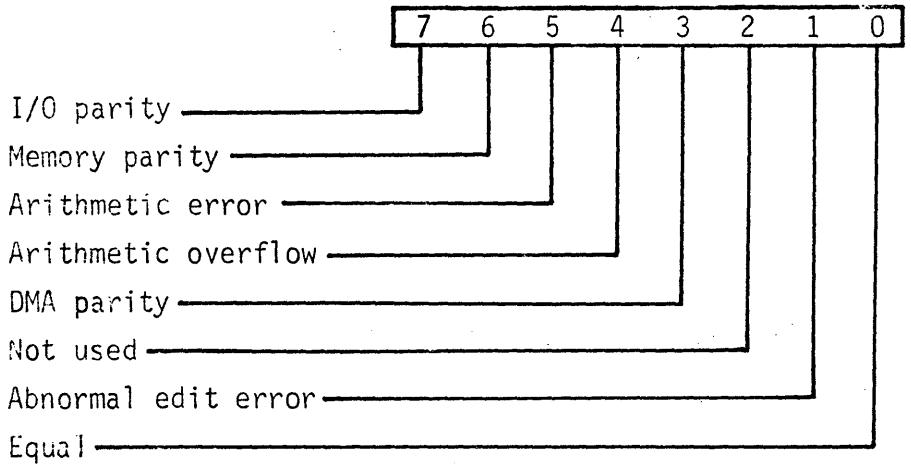
Octal Op Code = 030

PURPOSE: To cause a branch from the instruction execution sequence when a condition designator condition is matched in the bit configuration of the specified mask.

FORMAT:

OC	OP1	Branch to
030	M	Address

OPERATION: When the instruction is executed and a bit in the OP1 binary configuration is matched with one in the designators, the instruction execution sequence is transferred to the "branch to" address; otherwice, the execution sequence continues with the next instruction. The designators are set by the execution of other instructions.



Bit positions of mask with respect to designators.

EXAMPLE:

OC	OP1	Branch to	
030	002	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed.

OP1 00000010 bit configuration of 002

00000011 ABNORMAL EDIT ERROR and EQUAL condition designators are set

set

Since position 1 is matched, regardless of other designator settings, the execution is resumed at location 011-234.

BRANCHING

GOTO On Switches

Mnemonic Op Code = GS

Octal Op Code = 031

PURPOSE: To cause a branch from the instruction execution sequence when any of the operator panel GOTO switch settings is matched by the bit configuration of the specified mask.

FORMAT:

OC	OP1	Branch to
031	M	Address

OPERATION: When the instruction is executed and *any* of the operator panel switch settings matches the bit configuration of OP1, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The settings are manually activated.

EXAMPLE:

OC	OP1	Branch to	
031	120	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. Switch setting B on the operator panel was manually set.

	7	6	5	4	3	2	1	0
OP1	0	1	0	1	0	0	0	0

B

Position 6 and setting B match, which satisfies the requirement of any one position matching any one setting. Execution is resumed at location 011-234.

BRANCHING

GOTO Binary Greater Than

Mnemonic Op Code = GBG

Octal Op Code = *061

PURPOSE: To cause a branch from the instruction execution sequence when a "greater than binary zero" condition exists.

FORMAT:

OC	OP1	Branch to
061	[illegible]	Address

OPERATION: When the instruction is executed and the contents of OP1 are greater than binary zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Binary Ø	(GT) = 1		
	(=) = 1		
OP1 > Binary Ø	(GT) = 1	Branch	
	(=) = 0	performed	
OP1 < Binary Ø	(GT) = 0		
	(=) = 0		

EXAMPLE:

OC	OP1	Branch to	
061	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

GOTO Binary Less Than

Mnemonic Op Code = GBL

Octal Op Code = *062

PURPOSE: To cause a branch from the instruction execution sequence when a "less than binary zero" condition exists.

FORMAT:

OC	OP1	Branch to
062	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are less than binary zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Binary Ø	(GT)	= 1	
	(=)	= 1	
OP1 > Binary Ø	(GT)	= 1	
	(=)	= 0	
OP1 < Binary Ø	(GT)	= 0	Branch
	(=)	= 0	performed

EXAMPLE:

OC	OP1	Branch to	
062	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Binary Non-Zero

Mnemonic Op Code = GBN

Octal Op Code = *063

PURPOSE: To cause a branch from the instruction execution sequence when a "binary non-zero" condition exists.

FORMAT:

OC	OP1	Branch to
063	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are not equal to binary zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues wit with the next instruction. The designators are set as follows:

OP1 = Binary Ø	(GT) = 1		
	(=) = 1		
OP1 > Binary Ø	(GT) = 1	Branch	
	(=) = 0	performed	
OP1 < Binary Ø	(GT) = 0	Branch	
	(=) = 0	performed	

EXAMPLE:

OC	OP1	Branch to	
063	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Binary Zero

Mnemonic Op Code = GBZ

Octal Op Code = *064

PURPOSE: To cause a branch from the instruction execution sequence when a "binary zero" condition exists.

FORMAT:

OC	OP1	Branch to
064	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are equal to binary zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Binary Ø (GT) = 1 Branch
(=) = 1 performed
OP1 > Binary Ø (GT) = 1
(=) = 0
OP1 < Binary Ø (GT) = 0
(=) = 0

EXAMPLE:

OC	OP1	Branch to	
064	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Binary $\geq$ Zero

Mnemonic Op Code = GGBE

Octal Op Code = *065

PURPOSE: To cause a branch from the instruction execution sequence when an "equal to or greater than binary zero" condition exists.

FORMAT:

OC	OP1	Branch to
065	AR/I	Address

OPERATION: When this instruction is executed and the contents of OP1 are equal to or greater than binary zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Binary Ø	(GT) = 1	Branch
	(=) = 1	performed
OP1 > Binary Ø	(GT) = 1	Branch
	(=) = 0	performed
OP1 < Binary Ø	(GT) = 0	
	(=) = 0	

EXAMPLE:

OC	OP1	Branch to	
065	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Binary $\leq$ Zero

Mnemonic Op Code = GLBE

Octal Op Code = *066

PURPOSE: To cause a branch from the instruction execution sequence when an "equal to or less than binary zero" condition exists.

FORMAT:

OC	OP1	Branch to
066	AR/I	Address

OPERATION: When this instruction is executed and the contents of OP1 are equal to or less than binary zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Binary Ø	(GT) = 1	Branch
	(=) = 1	performed
OP1 > Binary Ø	(GT) = 1	
	(=) = 0	
OP1 < Binary Ø	(GT) = 0	Branch
	(=) = 0	performed

EXAMPLE:

OC	OP1	Branch to	
066	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Decimal Greater Than

Mnemonic Op Code = GDG

Octal Op Code = *071

PURPOSE: To cause a branch from the instruction execution sequence when a "greater than decimal zero" condition exists.

FORMAT:

OC	OP1	Branch to
071	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are greater than decimal zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are as follows:

OP1 = Decimal Ø	(GT) = 1		
	(=) = 1		
OP1 > Decimal Ø	(GT) = 1	Branch	
	(=) = 0	performed	
OP1 < Decimal Ø	(GT) = 0		
	(=) = 0		

EXAMPLE:

OC	OP1	Branch to	
071	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

GOTO Decimal Less Than

Mnemonic Op Code = GDL

Octal Op Code = *072

PURPOSE: To cause a branch from the instruction execution sequence when a "less than decimal zero" condition exists.

FORMAT:

OC	OP1	Branch to
072	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are less than decimal zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Decimal Ø	(GT) = 1		
	(=) = 1		
OP1 > Decimal Ø	(GT) = 1		
	(=) = 0		
OP1 < Decimal Ø	(GT) = 0	Branch	
	(=) = 0	performed	

EXAMPLE:

OC	OP1	Branch to	
072	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Decimal Non-Zero

Mnemonic Op Code = GDN

Octal Op Code = *073

PURPOSE: To cause a branch from the instruction execution sequence when a "decimal non-zero" condition exists.

FORMAT:

OC	OP1	Branch to
073	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are not equal to decimal zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Decimal Ø	(GT) = 1		
	(=) = 1		
OP1 > Decimal Ø	(GT) = 1	Branch	
	(=) = 0	performed	
OP1 < Decimal Ø	(GT) = 0	Branch	
	(=) = 0	performed	

EXAMPLE:

OC	OP1	Branch to	
073	AR/I	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Decimal Zero

Mnemonic Op Code = GDZ

Octal Op Code = *074

PURPOSE: To cause a branch from the instruction execution sequence when a "decimal zero" condition exists.

FORMAT:

OC	OP1	Branch to
074	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are equal to decimal zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Decimal Ø	(GT) = 1	Branch
	(=) = 1	performed
OP1 > Decimal Ø	(GT) = 1	
	(=) = 0	
OP1 < Decimal Ø	(GT) = 0	
	(=) = 0	

EXAMPLE:

OC	OP1	Branch to	
074	221	001	060

P-Bias = 010-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 011-060; otherwise, it continues with 010-126.

BRANCHING

GOTO Decimal $\geq$ Zero

Mnemonic Op Code = GGDE

Octal Op Code = *075

PURPOSE: To cause a branch from the instruction execution sequence when an "equal to or greater than zero" condition exists.

FORMAT:

OC	OP1	Branch to
075	AR/I	Address

OPERATION: When this instruction is executed and the contents of OP1 are equal to or greater than decimal zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Decimal Ø	(GT) = 1	Branch
	(=) = 1	performed
OP1 > Decimal Ø	(GT) = 1	Branch
	(=) = 0	performed
OP1 < Decimal Ø	(GT) = 0	
	(=) = 0	

EXAMPLE:

OC	OP1	Branch to	
075	221	001	060

P-Bias = 001-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, execution is resumed at location 010-060; otherwise, it continues with 010-125.

BRANCHING

GOTO Decimal $\leq$ Zero

Mnemonic Op Code = GLDE

Octal Op Code = *076

PURPOSE: To cause a branch from the instruction execution sequence when an "equal to or less than zero" condition exists.

FORMAT:

OC	OP1	Branch to
076	AR/I	Address

OPERATION: When this instruction is executed and the contents of OP1 are equal to or less than decimal zero, the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence continues with the next instruction. The designators are set as follows:

OP1 = Decimal Ø	(GT) = 1	Branch
	(=) = 1	performed
OP1 > Decimal Ø	(GT) = 1	
	(=) = 0	
OP1 < Decimal Ø	(GT) = 0	Branch
	(=) = 0	performed

EXAMPLE:

OC	OP1	Branch to	
076	221	001	060

P-Bias = 001-000

Location 010-122 contains the above instruction and it is being executed. If conditions are met, the execution is resumed at location 011-060; otherwise, it continues with 010-125.

BRANCHING

GOTO On Count

Mnemonic Op Code = GCT

Octal Op Code = *170

PURPOSE: To provide timing-loop or count-down capabilities by testing the OP1 item for binary zero.

FORMAT:

OC	OP1	Branch to
170	AR/I	Address

OPERATION: When the instruction is executed and the contents of OP1 are equal to binary zero, the instruction execution sequence continues with the next instruction. If the contents of OP1 are not equal to binary zero, a binary 1 is subtracted from OP1 and the execution sequence is transferred to the "branch to" address.

EXAMPLES:

OC	OP1	Branch to	
170	202	004	005

P-Bias = 001-000

Location 001-000 contains the above instruction and it is being executed.

No. 1

OP1 before	372	Item 2 of Active Record 2
OP1 after	371	Item 2 of Active Record 2

Branch to location 005-005.

OP1 before	000	Item 2 of Active Record 2
OP1 after	000	Item 2 of Active Record

Continue with 001-004.

OP1 must be 1 byte? accesses only the 1st byte

GOTO Table (Indirect Branch)

Mnemonic Op Code = GTB

Octal Op Code = *072

PURPOSE: To find a "branch to" address by using an index number which leads to the branch address in a table that makes up OP2. OP3 contains the upper table limit.

FORMAT:

OC	OP1	OP2	OP3
172	AR/I	AR/I	L
	index	table	upper limit

OPERATION: The OP2 item is a table of two-byte addresses. The OP1 index number is multiplied by two and added to the beginning OP2 address. The resultant address and the resultant address + 1 form the "branch to" address. When the instruction is executed, the execution sequence is transferred to this "branch to" address.

$OP1 \times 2 + {}^{OP2}_{address}$ = location of upper address

location of upper address + 1 = location of lower address.

If the OP1 index is greater than the OP3 literal (upper limit of table), the branch is not made and the execution sequence continues with the next instruction.

EXAMPLES: No. 1

OC	OP1	OP2	OP3
172	103	307	004

P-Bias = 001-000

Location 001-000 contains the above instruction and it is being executed.

OP1 [002] Item 3 of Active Record 1

OP2			
002-000	XXX	adr. Ø	Item 7 of Active Record 3
-001	XXX		
-002	XXX	adr. 1	
-003	XXX		
-004	005	adr. 2	
-005	221		
-006	XXX	adr. 3	
-007	XXX		
-010	XXX	adr. 4	
-011	XXX		

OP3 [004] upper limit: address 004 of OP2 table

OP1 x 2 + OP2 address = location of upper address

002 x 2 + 002-000 = 002-004 = location of upper address = 005 (from table)

upper address (002-004) + 1 = lower address (002-005) = 221 (from table)

"branch to" address 005-221

+ P-Bias 001-000

next instruction 006-221

Branch to location 006-221.

No. 2

OC	OP1	OP2	OP3
172	214	107	004

P-Bias = 001-000

Location 001-000 contains the above instruction and it is being executed.

OP1 [005] Item 14 of Active Record 2

OP2

002-000	XXX	adr. Ø
-001	XXX	
-002	XXX	adr. 1
-003	XXX	
-004	005	adr. 2
-005	221	
-006	016	adr. 3
-007	117	
-010	XXX	adr. 4
-011	XXX	
-012	any data	

Item 7 of Active Record 1

OP3 [004] upper limit: address 004 of OP2 table

OP1 x 2 + OP2 address = location of upper address

005 x 2 + 002-000 = 002-012 = illegal address (beyond the limit of table, specified in OP3)

Continue with 001-004.

BRANCHING

GOTO Return (Branch)

Mnemonic Op Code = GRT

Octal Op Code = *173

PURPOSE: To return from a subroutine to a main routine return address. The P-address is read from a push-down stack buffer. See the GSB instruction on the preceding page.

FORMAT:

OC	OP1
173	B

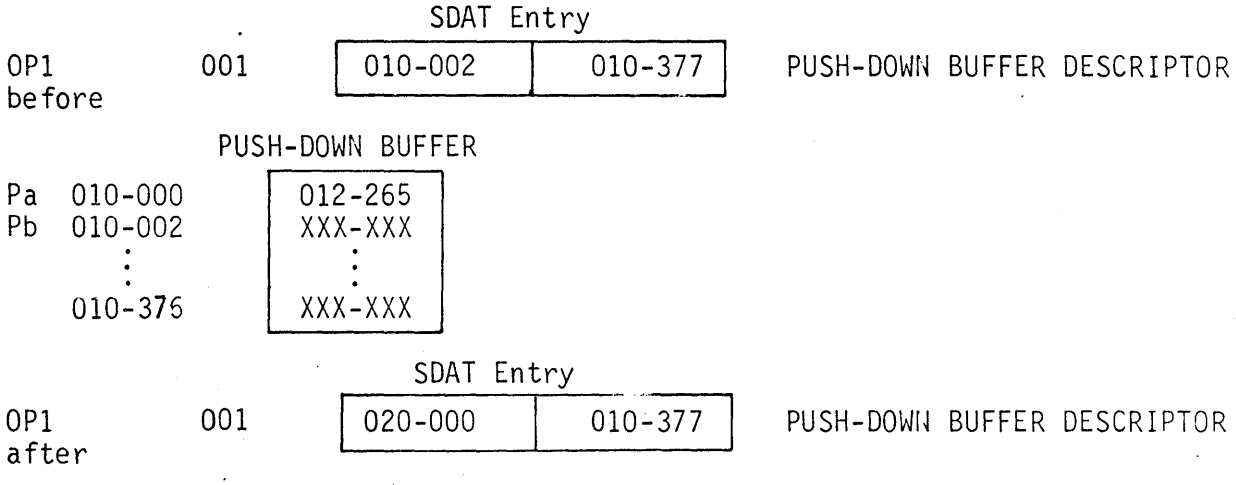
OPERATION: OP1 defines a push-down stack buffer which contains the return address. The push-down stack buffer consists of a set of two-byte entries which define return addresses. The P-address for the return address is located at the current buffer pointer address minus one and minus two. After retrieving the return address, the current buffer pointer is decremented by two. A branch is made to the return address. P-bias is not added to the return address prior to the branch.

EXAMPLE:

OC	OP1
173	001

P-Bias = 012-000

P-address before = 013-077



P-address after = 012-265

BRANCHING

GOTO Return (Branch)

Mnemonic Op Code = GSB

Octal Op Code = *176

PURPOSE: To branch to a subroutine and save the return main routine P-address in a push-down stack buffer. The limitation on nesting is determined by the size of the push-down stack buffer.

FORMAT:

OC	OP1	Branch to
176	B	Address

OPERATION: OP1 defines a push-down stack buffer. When this instruction is executed, the P-address of the next instruction is placed in the current push-down buffer pointer address and current push-down buffer pointer plus one. The push-down buffer consists of two-byte entries (return addresses). The current buffer pointer address is advanced by two and a branch is made to the designated subroutine.

EXAMPLE:

OC	OP1	Branch to	
176	001	001	000

P-Bias = 012-000

001-000

P-address before = 012-261

P-address return = 012-265

P-after 013-000

013-000	OC OP1. . . OPn
	subroutine
013-077	GRT B

SDAT Entry

OP1 before 001 | 010-000 | 010-377 | PUSH-DOWN BUFFER DESCRIPTOR

PUSH-DOWN BUFFER

Pb	010-000	012-265
Pa	010-002	XXX-XXX
	:	:
	010-376	XXX-XXX

OP1 after 001 SDAT Entry

010-002	010-377	PUSH-DOWN BUFFER DESCRIPTOR

ESTABLISH CONDITIONS

The Compare instructions establish relational conditions by setting appropriate internal condition designators, which are utilized by the Branching instructions as branching conditions.

The Compare instructions include the following:

- Compare Binary (1-47)
- Compare Decimal (1-48)
- Compare Alphanumeric (1-49)
- Compare Literal (1-50)

COMPARE

Compare Binary

Mnemonic Op Code = CB

Octal Op Code = 044

PURPOSE: To compare two binary numbers and thereby establish a condition designation of equality: >, =, or <.

FORMAT:

OC	OP1	OP2
044	AR/I	AR/I

OPERATION: The OP1 item is compared to the OP2 item, and their relationship is indicated by the internal condition designator settings below:

GREATER THAN is set when	OP1 > OP2
EQUAL is set when	OP1 = OP2
Both are not set when	OP1 < OP2

COMPARE

Compare Decimal

Mnemonic Op Code = CD

Octal Op Code = 046

PURPOSE: To compare two decimal numbers and thereby establish a condition designation of equality: >, =, or <.

FORMAT:

OC	OP1	OP2
046	AR/I	AR/I

OPERATION: The OP1 item is compared to the OP2 item, and their relationship is indicated by the internal condition designator settings below:

GREATER THAN is set when	OP1 > OP2
EQUAL is set when	OP1 = OP2
Both are not set when	OP1 < OP2

COMPARE

Compare Alphanumerics

Mnemonic Op Code = CAN

Octal Op Code = 142

PURPOSE: To compare all the characters in one item with those of another item and thereby establish a condition designation of equality: = or ≠.

FORMAT:

OC	OP1	OP2
142	AR/I	AR/I

OPERATION: All the characters of OP1 and OP2 are compared, starting with the leftmost character of each item. Null characters are considered equivalent to *any* alphanumeric. OP1 can be larger than OP2, but not the reverse. The result of the comparison is indicated by the condition designators below:

EQUAL is set when OP1 = OP2

EXAMPLES: No. 1

OP1 ABCnX7

OP1 and OP2 are equal because the n and the 5 are equivalent for this comparison.

OP2 ABC5X7

No. 2

OP1 ABCnX7

OP1 and OP2 are equal; OP1 can be >, but not the reverse.

OP2 ABC

No. 3

OP1 ABC

OP1 and OP2 are not equal, since OP2 > OP1.

OP2 ABCnX7

COMPARE

Compare Literal

Mnemonic Op Code = CL

Octal Op Code = 144

PURPOSE: To determine whether all of the characters in an item are the same as a specified literal and thereby establish a condition designation of equality: = or ≠.

FORMAT:

OC	OP1	OP2
144	AR/I	L

OPERATION: All the characters in the OP1 item are compared to the specified literal character to establish whether they are all the same as the literal or the Null character. The result of the comparison is indicated by the condition designator below:

EQUAL is set when OP1 = L or n

EXAMPLE:

OP1	***nn*
OP2	*

OP1 and OP2 are equal because the n and the asterisk are equivalent for this comparison.

ESTABLISH CONDITIONS

The Test instructions establish

- The sign of a binary or a decimal number

  or

- The occurrence of a given character

  or

- A bit correspondence.

The Test instructions are utilized by the Branching instructions as branching conditions.

The Test instructions include the following:

- Test Binary Sign (1-52)
- Test Decimal Sign (1-53)
- Test Item (54)
- Test Literal (1-55)
- Test Mask (1-56)
- Test Item Mask (1-57)

TEST

Test Binary Sign

Mnemonic Op Code = TBS

Octal Op Code = 040

PURPOSE: To examine the sign of a given binary number and thereby establish a + or - condition designation.

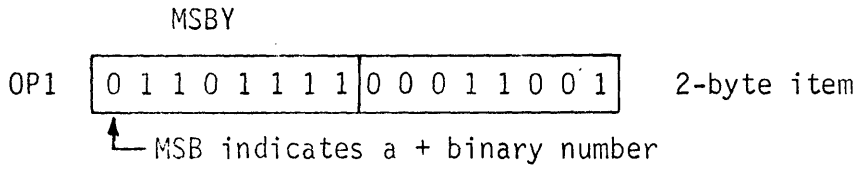
FORMAT:

OC	OP1
040	AR/I

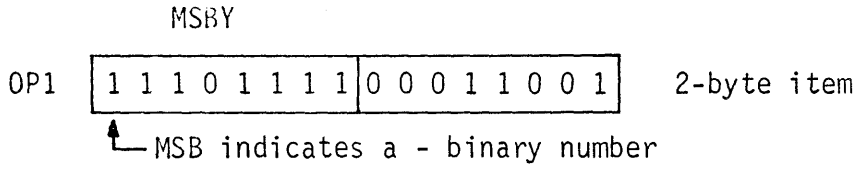
OPERATION: The sign of the OP1 item (binary number) is examined to determine whether it is + or -. The sign is indicated by a 0 (+) or a 1 (-) in the MSB of the MSBY, and the condition designator is set accordingly:

GREATER THAN is set when the MSB = 0 (which means +)

EXAMPLES: No. 1



No. 2



TEST

Test Decimal Sign

Mnemonic Op Code = TDS

Octal Op Code = 042

PURPOSE: To examine the sign of a given decimal number and thereby establish a + or - condition designation.

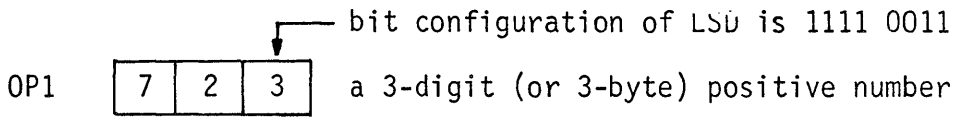
FORMAT:

OC	OP2
042	AR/I

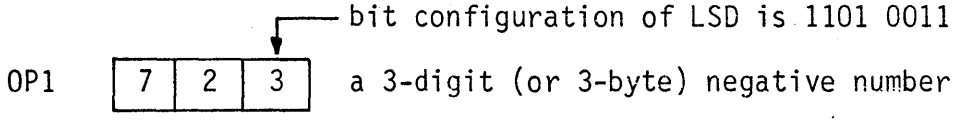
OPERATION: The sign of the OP1 item (decimal number) is examined to determine whether it is + or -. The sign is indicated by the sign zone of the least significant digit of the number, and the condition designator is set accordingly:

GREATER THAN is set when the number is positive.

EXAMPLES: No. 1



No. 2



TEST

Test Item

Mnemonic Op Code = TI

Octal Op Code = 150

PURPOSE: To determine whether the first character in one item appears in any positions of another item and thereby establish a condition designation.

FORMAT:

OC	OP1	OP2
150	AR/I	AR/I

OPERATION: The OP2 item is tested for the occurrence of the leftmost character of the OP1 item, and the condition designator is set accordingly:

EQUAL is set when OP2 contains at least one character corresponding to the first character of OP1.

EXAMPLE:

TEST

Test Literal

Mnemonic Op Code = TL

Octal Op Code = 151

PURPOSE: To determine whether an item contains a specified literal and thereby establish a condition designation.

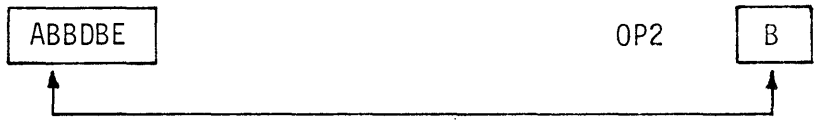
FORMAT:

OC	OP1	OP2
151	AR/I	L

OPERATION: The OP1 item is tested for the occurrence of the specified literal, and the condition designator is set accordingly:

EQUAL is set when OP1 contains the literal at least once.

EXAMPLE:



TEST

Test Mask

Mnemonic Op Code = TM

Octal Op Code = 152

PURPOSE: To determine whether the leftmost character of an item has a bit in any position corresponding to the bits in the mask, and thereby establish a condition designation.

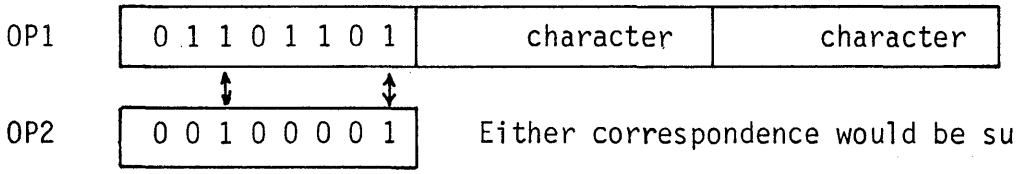
FORMAT:

OC	OP1	OP2
152	AR/I	M

OPERATION: The bit configuration of the leftmost character of the OP1 item is compared with the bit configuration of the specified mask. If there is a bit correspondence with any of the character bit positions, then the condition designator is set accordingly:

EQUAL is set when a bit correspondence exists.

EXAMPLE:



Either correspondence would be sufficient

TEST

Test Item Mask

Mnemonic Op Code = TIM

Octal Op Code = *153

PURPOSE: To scan an item of unknown contents with an item of pre-determined contents and look for a one-bit compare in any bit position.

FORMAT:

OC	OP1	OP2
153	AR/I	AR/I

OPERATION: The leftmost OP1 item character is compared against the OP2 item characters. If any "one" bit in the OP1 scan character matches a "one" bit position in the OP2 character, the instruction is terminated. Otherwise, the instruction is terminated at the end of the OP2 item. EQUAL is set when any "one" bit in the OP1 scan character matches a "one" bit position in the OP2 character.

EXAMPLES:

OC	OP1	OP2
153	204	377

No. 1

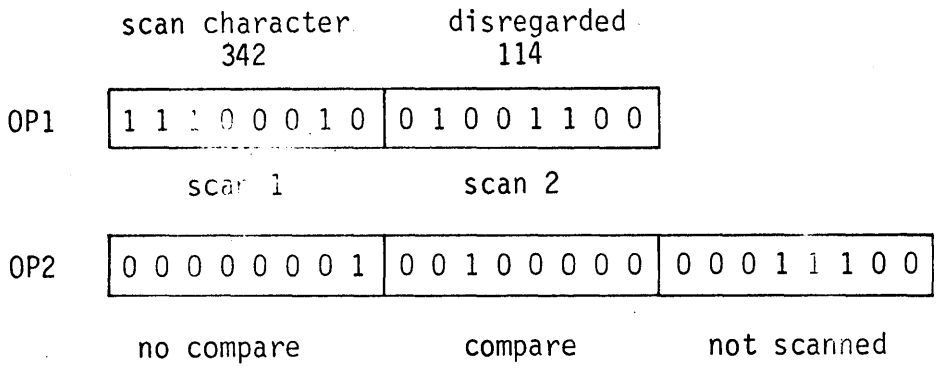
OP1	342	114		Item 4 of Active Record 2
OP2	001	020	034	Item 77 of Active Record 3

	scan character 342	disregarded 114
OP1	1 1 1 0 0 0 1 0	0 1 0 0 1 1 0 0

	scan 1	scan 2	scan 3
OP2	0 0 0 0 0 0 0 1	0 0 0 1 0 0 0 0	0 0 0 1 1 1 0 0
	no compare	no compare	no compare, end of OP2, instruction is terminated

No. 2

OP1	342	114		Item 4 of Active Record 2
OP2	001	040	034	Item 77 of Active Record 3



matched bit position, instruction is terminated and EQUAL is set.

INTERACTION

The I/O instructions initiate, control, and direct the I/O operations and interaction with the peripherals. This activity takes place between the input and output buffers and the I/O channel and its associated peripherals.

GENERAL I/O OPERATIONS

The Initiate Input On Channel and the Initiate Output On Channel instructions are used to initiate the standard input and output operations on a specified I/O Selector Channel and designate the appropriate buffer and peripherals. Hardware registers temporarily store and maintain the count of the characters into and out of the I/O buffers. Channel designators indicate appropriate *active* or *inactive* conditions on the specified channel.

RELATED I/O OPERATIONS AND STATUS

The External Function On Channel instruction is used to transmit operational commands (such as read) to a peripheral and to determine the operational status of the peripheral. GOTO On Active Channel is a branch instruction that is used to branch away from the normal instruction execution sequence if a specified channel is active. The Store Channel Control Register is used to store the current character address derived during the general I/O operations.

SPECIAL I/O OPERATIONS

The Special In and the Special Out instructions are used to perform the I/O operations on the DMA channels.

REVERSE I/O OPERATIONS

The reverse buffering capabilities are used to buffer input/output data in reverse when the connected I/O device is reading or writing in reverse.

The Input/Output instructions include the following:

- Special In (1-60)
- External Function On Channel (1-61)
- Special Out (1-62)
- External Function Special (1-63)
- GOTO On Active Channel (1-64)
- Store Channel Control Register (1-65)
- Store Channel Reverse (1-66)
- Initiate Input Reverse (1-67)
- Initiate Input On Channel (1-68)
- Initiate Output On Channel (1-69)
- Initiate Output Reverse (1-70)

INPUT/OUTPUT

Special In

Mnemonic Op Code = INS

Octal Op Code = 100

PURPOSE: To direct the input data to a given storage area when using a DMA channel.

FORMAT:

OC	OP1	OP2
100	AR/I	AR/I

OPERATION: The input data received through the specified DMA channel interface module in OP1 is stored in the OP2 item, left-aligned.

OP1 item specifies the DMA channel number connected to the DMA interface module.

OP2 item is to receive the input data.

The documentation for the specific interface module provides detailed information concerning the input data.

EXAMPLE:

OC	OP1	OP2
100	241	304

OP1 Item 41 of Active Record 2 contains the DMA channel number.

OP2 Item 4 of Active Record 3 is to receive the input data.

INPUT/OUTPUT

External Function On Channel

Mnemonic Op Code = EF

Octal Op Code = 104

PURPOSE: To transmit operational commands (such as READ EBCDIC) to a peripheral device and determine its status.

FORMAT:

OC	OP1	OP2	OP3
104	AR/I	AR/I	AR/I

OPERATION: The External Function code in the OP2 item is transmitted over the I/O channel specified in the OP1 item to the peripheral device, which replies with a status code that is placed in the OP3 item. If the status is *not* received within a specific time period, one null character is stored in the first status byte and the next instruction is executed. The meanings for the External Function and the status codes are dependent upon the type of peripheral.

NULL characters are sent as command bytes after the end of the OP2 item when OP2 < OP3. The next instruction is executed when OP3 ends.

No status bytes are stored in the status area after the end of the OP3 item when OP2 > OP3. The next instruction is executed when OP2 ends.

Both the function request (FRQ) control line and the data request (DRQ) control line are active on the first command byte. Only the FRQ is active on each succeeding command byte of the OP2 item.

For a complete explanation, see PROCESSOR PROGRAMMING IN MACHINE CODE, Form No. M-2269.

EXAMPLE:

OC	OP1	OP2	OP3
104	103	104	110

The items are all located in Active Record 1.

OP1 Item 3 contains the channel number for the peripheral.

OP2 Item 4 contains the External Function code for a specific peripheral.

OP3 Item 10 is to store the status code from the peripheral.

INPUT/OUTPUT

Special Out

Mnemonic Op Code = OTS

Octal Op Code = 105

PURPOSE: To direct output data from a given storage area when using a DMA channel.

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

OPERATION: The output data from the OP2 item is sent to the DMA channel interface module specified in OP1.

OP1 item specifies the DMA channel number connected to the DMA interface module.

OP2 item contains the output data.

The documentation for the specific interface module provides detailed information concerning the output data.

EXAMPLE:

OC	OP1	OP2
105	120	212

OP1 Item 30 of Active Record 1 contains the DMA channel number.

OP2 Item 12 of Active Record 2 contains the output data.

INPUT/OUTPUT

External Function Special

Mnemonic Op Code = EFS

Octal Op Code = *106

PURPOSE: To prevent issue of a data request for I/O devices that do not require such a signal; otherwise, this instruction is the same as EF.

FORMAT:

OC	OP1	OP2	OP3
106	AR/I	AR/I	AR/I

OPERATION: The EFS instruction prevents the issue of DRQ for I/O devices that may function improperly on receiving such a signal. This instruction guarantees 2408 compatibility via software for all I/O applications that do not require the above signal.

INPUT/OUTPUT

GOTO On Active Channel

Mnemonic Op Code = GA

Octal Op Code = 107

PURPOSE: To cause a branch from the instruction execution sequence when the specified channel is active.

FORMAT:

OC	OP1	Branch to
107	AR/I	Address

OPERATION: If the channel specified in OP1 is active, then the instruction execution sequence is transferred to the "branch to" address; otherwise, the execution sequence is continued.

EXAMPLE:

107	001	001	234

P-Bias = 010-000

Location 010-162 contains the above instruction and it is being executed. If channel 001 (OP1) is active, then execution is resumed at location 011-234; otherwise, it continues with 010-166.

INPUT/OUTPUT

Store Channel Control Register

Mnemonic Op Code = STC

Octal Op Code = 110

PURPOSE: To store the contents of the channel control register.

FORMAT:

OC	OP1	OP2
110	AR/I	AR/I

OPERATION: Stores in the OP2 item the "current character address" of the channel control register specified by the OP1 item. See the operation descriptions of the IN and OUT instructions for the meaning of the "current character address."

EXAMPLE:

OC	OP1	OP2
110	117	212

OP1 Item 17 of Active Record 1 contains the channel number in binary.

OP2 Item 12 of Active Record 2 is to receive the current character address.

INPUT/OUTPUT

Store Channel Reverse

Mnemonic Op Code = STR

Octal Op Code = *111

PURPOSE: To store the last address buffer control work for a specified channel.

FORMAT:

OC	OP1	OP2
111	AR/I	AR/I

OPERATION: The last address buffer control word for the channel specified by OP1 is stored into the item specified by OP2.

NOTE

The 2408 has two buffer control addresses for each individual selector channel. One buffer address is used for forward buffering of data. The other buffer address is used for reverse buffering. Thus, the last address buffer control word is being decremented for every character that is transferred between the I/O device and the processor during reverse buffering. All reverse buffering instructions are mainly used to enable the processor to receive data from a magnetic tape unit reading backwards.

EXAMPLE:

OC	OP1	OP2
111	201	202

OP1 [006] Item 1 of Active Record 2

Channel 6

OP2 before [000 000] Item 2 of Active Record 2

[127 322] control memory address 35

OP2 after [127 322]

INPUT/OUTPUT

Initiate Input Reverse

Mnemonic Op Code = INR

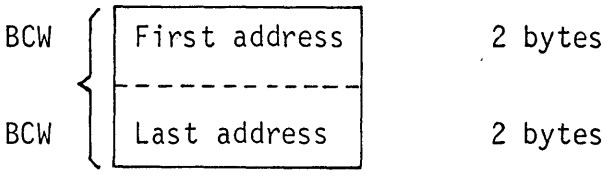
Octal Op Code = *112

PURPOSE: To store data reverse into a specified buffer through a specified channel.

FORMAT:

OC	OP1	OP2
112	AR/I	B

OPERATION: OP2 specifies an input buffer, where the last address is used as the current address, always being decremented after each character transfer. The loading of the buffer is terminated and the channel designator is reset when the first address of the buffer is reached. Data is received through the channel specified in the OP1 item. OP2 defines the buffer in the following manner:



NOTE

The 2408 has two buffer control word addresses for each individual selector channel. One buffer address is used for forward buffering of data. The other buffer address is used for reverse buffering. Thus, the last address buffer control is being decremented for every character that is transferred between the I/O device and the processor during reverse buffering. All reverse buffering instructions are mainly used to enable the processor to receive data from a magnetic tape unit reading backwards.

The I/O parity designator is set if a parity error is detected on the input data.

INPUT/OUTPUT

Initiate Input On Channel

Mnemonic Op Code = IN

Octal Op Code = 114

PURPOSE: To initiate input on a specific channel and designate the input buffer area.

FORMAT:

OC	OP1	OP2
114	AR/I	B

OPERATION: Initiates an input of data from a peripheral device, using the channel specified in the OP1 item and storing the data in the buffer area specified in the OP2 item, which is defined in the SDAT by its *first address* and *last address*. The OP2 buffer descriptor is transferred to a channel control register and a channel designator is set to ACTIVE. In the channel control register, the first and last addresses are entered in the current address register and the last address register, respectively.

The next instructions are executed while the characters are received from the peripheral device and stored in the buffer. After each character is stored in the buffer, the current address register is compared to the last address register and then incremented by one. When they are equal, the input operation is terminated and the channel designator is set to INACTIVE.

Upon termination, the current address register = last character address + 1, and the last address register = last address.

The channel designator is also set to INACTIVE when a Function Acknowledge is received from the peripheral.

The I/O parity condition designator is set when an odd-parity error is detected on input data.

For a complete explanation, see PROCESSOR PROGRAMMING IN MACHINE CODE, Form No. M-2269.

EXAMPLE:

OC	OP1	OP2
114	241	303

OP1 Item 41 of Active Record 2 contains the channel number.

OP2 Item 3 of Active Record 3 contains the input buffer limits.

INPUT/OUTPUT

Initiate Output On Channel

Mnemonic Op Code = OUT

Octal Op Code = 115

PURPOSE: To initiate output on a specific channel and designate the output buffer area.

FORMAT:

OC	OP1	OP2
115	AR/I	B

OPERATION: Initiates an output of data from the output buffer specified in OP2 to a peripheral device on the channel specified in the OP1 item. The OP2 buffer is defined in the SDAT by its *first address* and *last address + 1*. During execution of this instruction, these addresses are transferred to a channel control register and a channel designator is set to ACTIVE. The first and last + 1 addresses are entered in the current address register and the last address register, respectively, of the channel control register.

The next instructions are executed, while the characters are transferred from the buffer to the peripheral, with an odd-parity bit generated for each byte. Before each character transfer, the current address register is compared to the last address register and then incremented by one. When they are equal, the operation is terminated and the channel designator is set to INACTIVE.

The channel designator is also set to INACTIVE when a Function Acknowledge is received from the peripheral. The channel control register is then:

current address register = last address + 2

last address register = last address + 1

EXAMPLE:

OC	OP1	OP2
115	145	114

OP1 Item 45 of Active Record 1 contains the channel number.

OP2 Item 14 of Active Record 1 contains the output buffer limits.

INPUT/OUTPUT

Initiate Output Reverse

Mnemonic Op Code = OTR

Octal Op Code = *116

PURPOSE: To initiate output reverse on a specific channel and to designate the output buffer area.

FORMAT:

OC	OP1	OP2
116	AR/I	B

OPERATION: OP2 specifies an output buffer. Data output through the channel specified by OP1 is initiated. After each character transfer to the peripheral, the last address of the channel buffer control word is decremented until the first address - 1 and last address match. Operations are then terminated by resetting the channel designator. OP2 defines the buffer the same way as in the INR instruction.

The General Purpose instructions include the following:

- Rename (1-72)
- Store Designators (1-73)
- Load Designators (1-74)
- Store Tally Counter (1-75)
- Load Tally Counter (1-76)
- Halt (1-77)
- Set Display Indicators (1-78)
- No Operation - Leave Gap (1-79)
- Clear Display Indicators (1-80)
- Load Storage Descriptor Pointer (1-81)
- Load Active Record 1, 2 or 3 (1-82)

GENERAL PURPOSE

RENAME

Mnemonic Op Code = RN

Octal Op Code = 000

PURPOSE: To change or reset a record or buffer address descriptor in the SDAT.

FORMAT:

OC	OP1	OP2
000	R	R

or

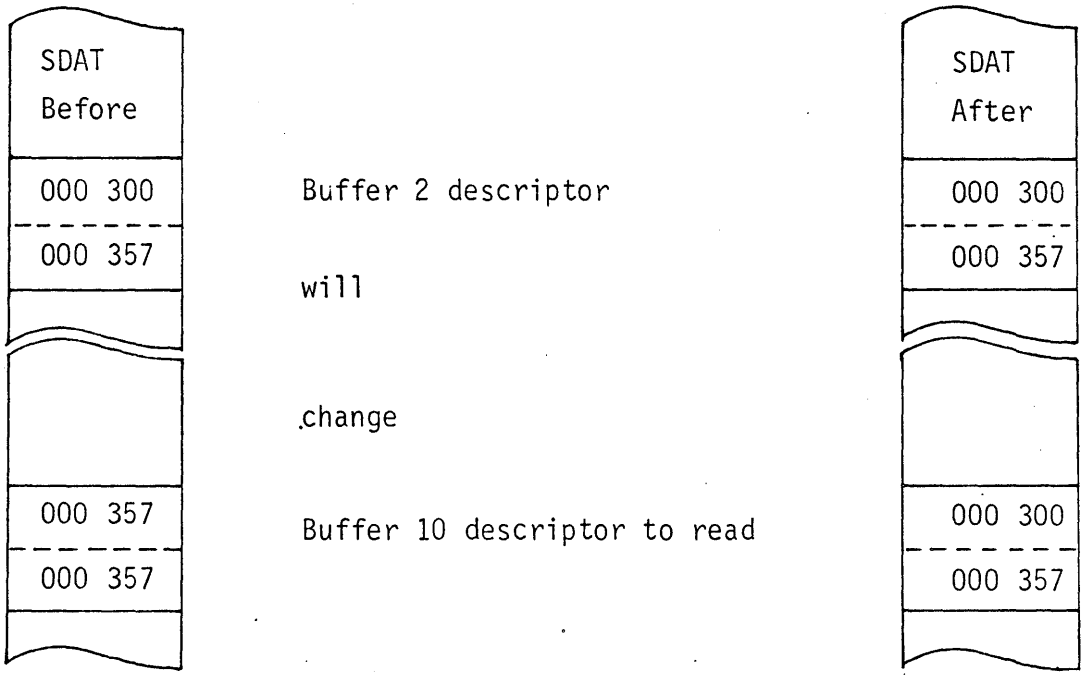
OC	OP1	OP2
000	B	B

OPERATION: The address descriptor in the SDAT for the OP2 record or buffer is replaced by the OP1 record or buffer address descriptor. This instruction is used to restore a buffer descriptor to its original contents following an Extract or Append type of instruction, since the starting address of the descriptor is incremented to maintain a current address pointer during the execution of these instructions and is therefore no longer available. Note that a data move to or from an SDAT entry acts as if the SDAT entry is a 4-byte data item.

EXAMPLE:

OC	OP1	OP2
000	002	010

When this instruction is executed,



thereby restoring (renaming) it to its original value.

GENERAL PURPOSE

Store Designators

Mnemonic Op Code = STD

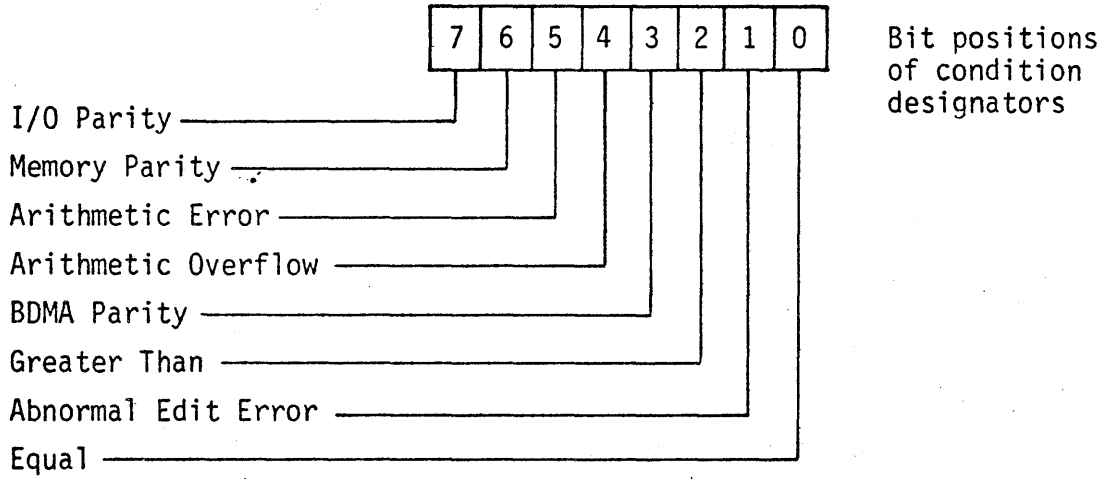
Octal Op Code = *124

PURPOSE: To store the status of the condition designators into a specified item.

FORMAT:

OC	OP1
124	AR/I

OPERATION: One byte of data, representing the present status of the designators, is written into the leftmost byte of the item specified by OP1.



EXAMPLE:

OC	OP1
124	317

OP1 before: 327 — Item 17 of Active Record 3

OP1 after: 101 — Item 17 of Active Record 3

-assuming the Equal and Memory Parity designators are set.

GENERAL PURPOSE

Load Designators

Mnemonic Op Code = LD

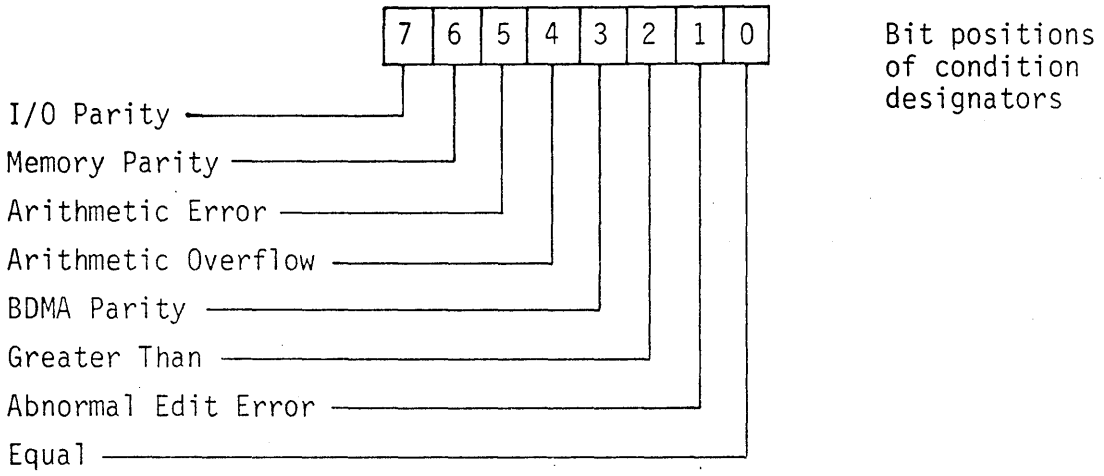
Octal Op Code = *126

PURPOSE: To set the condition designators according to the bit pattern of the specified item.

FORMAT:

OC	OP1
126	AR/I

OPERATION: The leftmost item of OP1 forces the designators to an identical bit pattern.



EXAMPLE:

OC	OP1
126	320

OP1 060 Item 20 of Active Record 3

		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Designators	before	0	0	0	0	0	0	0	0
	after	0	0	1	1	0	0	0	0

The Arithmetic Error and Arithmetic Overflow designators are set.

GENERAL PURPOSE

Store Tally Counter

Mnemonic Op Code = STT

Octal Op Code = *134

PURPOSE: To store the contents of the tally counter into a specified item.

FORMAT:

OC	OP1
134	AR/I

OPERATION: The 16 bit positions of the tally counter are stored into the leftmost two bytes of the OP1 item. See Appendix D for the effect of instruction execution on Tally Counter.

EXAMPLE:

OC	OP1
134	107

Tally Counter =

137	276	

OP1 before

224	011	377	Item 7 of Active Record 1

OP1 after

137	276	377

GENERAL PURPOSE

Load Tally Counter

Mnemonic Op Code = LT

Octal Op Code = *136

PURPOSE: To set the tally counter according to the bit pattern of specified item.

FORMAT:

OC	OP1
136	AR/I

OPERATION: The leftmost two bytes of the item specified by OP1 force the tally counter to an identical bit pattern. See Appendix D for the effect of instruction execution on Tally Counter.

EXAMPLE:

OC	OP1
136	221

OP1 | 131 | 246 | 012 | Item 21 of Active Record 2

Tally Counter

before	0 0 0 0 0 0
after	1 3 1 2 4 6

GENERAL PURPOSE

HALT

Mnemonic Op Code = H

Octal Op Code = 143

PURPOSE: To stop the execution of instructions.

FORMAT:

OC
143

OPERATION: Halts the execution of instructions, but the I/O operations can continue until the buffer terminates. The STOP indicator on the PROCESSOR STATUS panel is illuminated. The instruction execution sequence can be resumed by pressing the RUN switch on the control panel.

GENERAL PURPOSE

Set Display Indicators

Mnemonic Op Code = SDI

Octal Op Code = 146

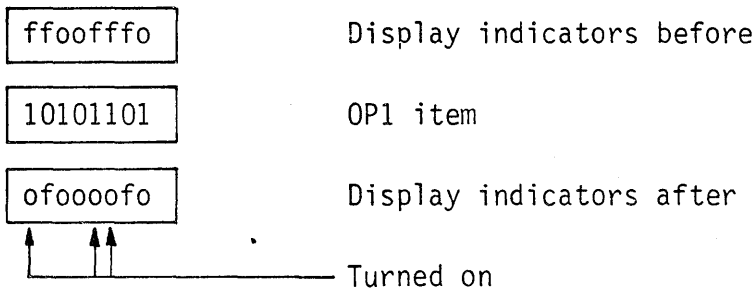
PURPOSE: To turn on one or more display indicators.

FORMAT:

OC	OP1
146	AR/I

OPERATION: The display indicators are turned on by the presence of a one ("1") bit in the corresponding position of the 1-byte item of OP1.

EXAMPLE:



GENERAL PURPOSE

No Operation - Leave Gap

Mnemonic Op Code = GAP

Octal Op Code = *147

PURPOSE: To allow a program delay.

FORMAT:

OC
147

OPERATION: The program is delayed by 1 microsecond (the time it takes to go through a P-sequence).

GENERAL PURPOSE

Clear Display Indicators

Mnemonic Op Code = CDI

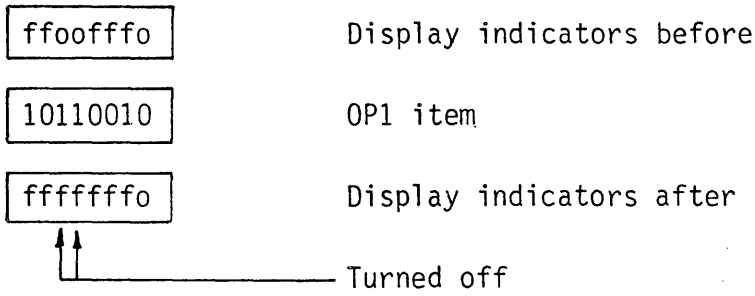
Octal Op Code = 156

PURPOSE: To turn off one or more display indicators.

FORMAT:

OC	OP1
156	AR/I

OPERATION: The display indicators are turned off by the presence of a one ("1") bit in the corresponding position of the 1-byte item of OP1.



GENERAL PURPOSE

Load Storage Descriptor Pointer

Mnemonic Op Code = LSP

Octal Op Code = 161

PURPOSE: To declare another SDAT active.

FORMAT:

OC	OP1
161	SDP

OPERATION: The OP1 storage descriptor pointer is loaded over the current pointer in address 000-000 and $000\text{-}001_8$, thereby activating a new SDAT.

GENERAL PURPOSE

Load Active Record 1, 2, or 3

Mnemonic Op Code = LR1, LR2, or LR3

Octal Op Code = 165, 171, or 175

PURPOSE: To declare a record active.

FORMAT:

OC	OP1
165,171 or 175	R

OPERATION: Loads the active record register with the OP1 record number, which is an item in the SDAT and contains the record descriptor for the record area in storage. A record must be declared active before it can be referenced by any other instructions. Records continue to be "active" until a succeeding LR instruction activates another record in its place (1, 2, or 3).

Appendix B, Programming Active Records, provides additional information for using the Load Active Record instructions.

EXAMPLE:

OC	OP1
171	011

OC 171 Load Active Record 2

OP1 011 with record 11 of the SDAT, which indicates the first address of the record in storage and the first address of its IDT.

LOGICAL

The logical instructions include the following:

- OR (Exclusive) (1-84)
- Longitudinal Redundancy Check (1-85)
- OR (Inclusive)(1-86)
- Logical AND (1-87)

LOGICAL SET

OR (Exclusive)

Mnemonic Op Code = X

Octal Op Code = *160

PURPOSE: To logically OR (Exclusive) two strings of data and store the result into a defined item.

FORMAT:

OC	OP1	OP2	OP3
160	AR/I	AR/I	AR/I

OPERATION: An Exclusive OR function is performed between the data contained in the leftmost bytes of the OP1 and OP2 items. The result is stored left-aligned into the OP3 item. This function continues until one of the three operands is ended.

Exclusive OR Operation:

```
  1 1 0 0
  1 0 1 0
  -------
  0 1 1 0
```

EXAMPLE:

OC	OP1	OP2	OP3
160	201	202	203

OP1	305	1 1 0 0 0 1 0 1	Item 1 of Active Record 2
OP2	147	0 1 1 0 0 1 1 1	Item 2 of Active Record 2
OP3 before	377	1 1 1 1 1 1 1 1	Item 3 of Active Record 2
OP3 after	242	1 0 1 0 0 0 1 0	Item 3 of Active Record 2

LOGICAL SET

Longitudinal Redundancy Check

Mnemonic Op Code = RCK

Octal Op Code = *162

PURPOSE: To perform successive Exclusive OR operations to a string of data and to store the result into a defined item.

FORMAT:

OC	OP1	OP2
162	AR/I	AR/I

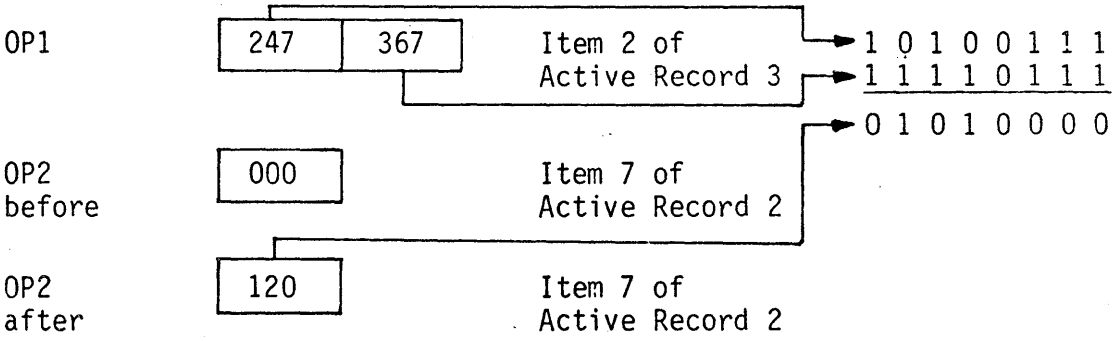
OPERATION: An Exclusive OR function is performed between the leftmost byte of the OP1 item and the second leftmost byte. An Exclusive OR is then performed between this result and the third leftmost byte of the OP1 item. This function continues until the end of the OP1 item is reached. The result is then stored into the OP2 item.

Exclusive OR Operation:

```
  1 1 0 0
  1 0 1 0
  -------
  0 1 1 0
```

EXAMPLE:

OC	OP1	OP2
162	302	207



LOGICAL SET

OR (Inclusive)

Mnemonic Op Code = O

Octal Op Code = *164

PURPOSE: To logically OR (Inclusive) two strings of data and store the result into a defined item.

FORMAT:

OC	OP1	OP2	OP3
164	AR/I	AR/I	AR/I

OPERATION: A Logical OR function is performed between the data contained in the leftmost bytes of the OP1 and OP2 items. The result is stored left-aligned into the OP3 item. This function continues until one of the three operands is ended.

Inclusive OR Operation:

```
1 1 0 0
1 0 1 0
-------
1 1 1 0
```

EXAMPLE:

OC	OP1	OP2	OP3
164	301	202	107

OP1	362	1 1 1 1 0 0 1 0	Item 1 of Active Record 3
OP2	203	1 0 0 0 0 0 1 1	Item 2 of Active Record 2
OP3 before	227	1 0 0 1 0 1 1 1	Item 7 of Active Record 1
OP3 after	363	1 1 1 1 0 0 1 1	Item 7 of Active Record 1

LOGICAL SET
Logical AND
Mnemonic Op Code = N
Octal Op Code = *166

PURPOSE: To logically AND two strings of data and store the result into a defined item.

FORMAT:

OC	OP1	OP2	OP3
166	AR/I	AR/I	AR/I

OPERATION: A logical AND operation is performed between the data contained in the leftmost bytes of the OP1 and OP2 items. The result is stored left-aligned into the OP3 item. This function continues until one of the three operands is ended.

Logical AND Operation:

```
  1 1 0 0
  1 0 1 0
  -------
  1 0 0 0
```

EXAMPLE:

OC	OP1	OP2	OP3
166	102	203	306

OP1	153	0 1 1 0 1 0 1 1	Item 2 of Active Record 1
OP2	355	1 1 1 0 1 1 0 1	Item 3 of Active Record 2
OP3 before	000	0 0 0 0 0 0 0 0	Item 6 of Active Record 3
OP3 after	151	0 1 1 0 1 0 0 1	Item 6 of Active Record 3

During binary arithmetic operations, one of two error indicators may be illuminated on the operator panel and the corresponding condition designator set:

- Arithmetic overflow - when the receiving item is 1 byte too small and the sign of the number is lost (condition designator bit 4).
- Arithmetic Error - when the receiving item is 1 or more bytes too small for the operation (condition designator bit 5).

Whenever either of these conditions occurs, the arithmetic operation is immediately terminated.

Negative binary numbers *must be* coded in two's complement form.

The sign of a binary number is indicated by a 0 (for +) or a 1 (for -) in the MSB of the MSBY.

The Binary Arithmetic instructions include the following:

- Add Binary (1-89)
- Subtract Binary (1-90)
- Add Literal Binary (1-91)
- Subtract Literal Binary (1-92)

BINARY ARITHMETIC

Add Binary

Mnemonic Op Code = AB

Octal Op Code = 041

PURPOSE: To add two binary numbers.

FORMAT:

OC	OP1	OP2	OP3
041	AR/I	AR/I	AR/I

OPERATION: The OP1 item is added to the OP2 item and the sum stored in the OP3 item.

Condition Designators:

Arithmetic overflow is set when the OP3 item is too small by one byte and the sign bit is lost.

Arithmetic error is set when the add operation cannot be completed for all bytes in the OP1 or OP2 items because the OP3 item is too small.

EXAMPLE:

OP1		01111111	11001110	
OP2			01001101	
OP3	00000000	10000000	00011011	after the addition

BINARY ARITHMETIC

Subtract Binary

Mnemonic Op Code = SB

Octal Op Code = 045

PURPOSE: To subtract one binary number from another.

FORMAT:

OC	OP1	OP2	OP3
045	AR/I	AR/I	AR/I

OPERATION: The OP2 item is subtracted from the OP1 item and the difference is stored in the OP3 item.

Condition Designators:

Arithmetic overflow is set when the OP3 item is too small by one byte and the sign bit is lost.

Arithmetic error is set when the subtract operation cannot be completed for all bytes in the OP1 and OP2 item because the OP3 item is too small.

EXAMPLE:

OP1	00001101	01011101	
OP2		01110111	
OP3	00001100	11100110	after the subtraction

BINARY ARITHMETIC

Add Literal Binary

Mnemonic Op Code = ALB

Octal Op Code = 051

PURPOSE: To add a binary number contained in the instruction itself (literal) to a binary number in core storage.

FORMAT:

OC	OP1	OP2	OP3
051	AR/I	AR/I	L

OPERATION: The OP3 binary number is added to the OP1 item and the sum stored in the OP2 item.

Condition Designators:

Arithmetic overflow is set when the OP2 item is too small by one byte and the sign bit is lost.

Arithmetic error is set when the add operation cannot be completed for all bytes in OP1 item because the OP2 item is too small.

EXAMPLE:

OP1	00000001	11001110	
OP2		01001101	
OP3	00000010	00011011	after the addition

BINARY ARITHMETIC

Subtract Literal Binary

Mnemonic Op Code = SLB

Octal Op Code = 055

PURPOSE: To subtract a binary number contained in the instruction itself (literal) from a binary number in core storage.

FORMAT:

OC	OP1	OP2	OP3
055	AR/I	AR/I	L

OPERATION: The OP3 binary number is subtracted from the OP1 item and the difference stored in the OP2 item.

Condition Designators:

Arithmetic overflow is set when the OP2 is too small by one byte and the sign is lost.

Arithmetic error is set when the subtract operation cannot be completed for all bytes in the OP1 item because the OP2 item is too small.

EXAMPLE:

OP1	00000001	11101110	
OP2		00110110	
OP3	00000001	10111000	after the subtraction

DECIMAL ARITHMETIC

During decimal arithmetic operations, one of two error indicators may be illuminated on the operator panel and the corresponding condition designator set:

- Arithmetic overflow - when the receiving item is 1 byte too small and the carry is lost (condition designator bit 4).
- Arithmetic error - when the receiving item is 1 or more bytes too small for the operation (condition designator bit 5).

Whenever either of these conditions occurs, the arithmetic operation is immediately terminated, and the results in the receiving item may *not* be represented in "absolute values" but in "ten's complement" notation. Normally, the sign of the number is corrected and the decimal digits changed to their absolute value after the add operation. A ten's complement number is obtained by:

- subtracting the absolute value

  from

- 10 raised to a power equal to the number of digits in the absolute value.

For example, the ten's complement for 456 is 10^3 = 1000, and then 1000 - 456 = 544.

The sign of a number is indicated by the sign zone of the least significant digit.

The Decimal Arithmetic instructions include the following:

- Add Decimal (1-94)
- Subtract Decimal (1-95)
- Add Literal Decimal (1-96)
- Subtract Literal Decimal (1-97)

DECIMAL ARITHMETIC

Add Decimal

Mnemonic Op Code = A

Octal Op Code = 043

PURPOSE: To add two decimal numbers.

FORMAT:

OC	OP1	OP2	OP3
043	AR/I	AR/I	AR/I

OPERATION: The contents of the OP1 item are added to the contents of the OP2 item and the sum is stored in the OP3 item. The numbers in the OP1 and OP2 items are represented in decimal form.

Condition Designators:

Arithmetic overflow is set when the OP3 item is too small by one byte and the carry is lost.

Arithmetic error is set when the add operation cannot be completed for all bytes in the OP1 or OP2 item because the OP3 item is too small.

EXAMPLE:

OP1		7	1	+4
OP2			4	+3
OP3	0	7	5	+7

after the addition

DECIMAL ARITHMETIC

Subtract Decimal

Mnemonic Op Code = S

Octal Op Code = 047

PURPOSE: To subtract one decimal number from another.

FORMAT:

OC	OP1	OP2	OP3
047	AR/I	AR/I	AR/I

OPERATION: The contents of the OP2 item are subtracted from the OP1 item and the difference is stored in the OP3 item. The numbers in the OP1 and OP2 items are represented in decimal form.

Condition Designators:

Arithmetic overflow is set when the OP3 item is too small by one byte and the carry is lost.

Arithmetic error is set when the subtract operation cannot be completed for all bytes in the OP1 and OP2 item because the OP3 item is too small.

EXAMPLE:

OP1			6	-3	
OP2			8	+3	
OP3	0	1	4	-6	after the subtraction

DECIMAL ARITHMETIC

Add Literal Decimal

Mnemonic Op Code = AL

Octal Op Code = 053

PURPOSE: To add a decimal number contained in the instruction itself (literal) to a decimal number in core storage.

FORMAT:

OC	OP1	OP2	OP3
053	AR/I	AR/I	L

OPERATION: The contents of the OP3 literal are added to the contents of the OP1 item, and the sum is stored in the OP2 item. The OP1 item and OP3 literal are decimal numbers.

Condition Designators:

Arithmetic overflow is set when the OP2 item is too small by one byte and the carry is lost.

Arithmetic error is set when the add operation cannot be completed for all bytes of the OP1 item because the OP2 item is too small.

EXAMPLE:

OP1		4	+3	
OP3			+5	
OP2	0	4	+8	after the addition

DECIMAL ARITHMETIC

Subtract Literal Decimal

Mnemonic Op Code = SL

Octal Op Code = 057

PURPOSE: To subtract a decimal number contained in the instruction itself (literal) from a decimal number in core storag.e

FORMAT:

OC	OP1	OP2	OP3
057	AR/I	AR/I	L

OPERATION: The contents of the OP3 literal are subtracted from the contents of the OP1 item, and the difference is stored in the OP2 item. The OP1 item and OP3 literal are decimal numbers.

Conditional Designators:

Arithmetic overflow is set when the OP2 item is too small by one byte and the carry is lost.

Arithmetic error is set when the subtract operation cannot be completed for all bytes in the OP1 item since the OP3 item is too small.

EXAMPLE:

OP1		4	+6	
OP3			+4	
OP2	0	4	+2	after the subtraction

EDIT IN TRANSMIT

The Sequential Editing instructions can edit large volumes of data as they are transferred from peripheral to peripheral and when the units of data:

- Consist of more than 256 characters (the maximum record size),
- Cannot be assigned a predetermined number of characters,
- Consist of sub units of data that must be handled sequentially,

  or

- Are reduced in size by removing special characters during editing.

The Sequential Editing instructions consist of three groups, which:

- *Compress* data into smaller groups after eliminating nulls and specified characters,
- *Append* portions of data to the build-up of a larger block, and
- *Extract* portions from large data blocks.

COMPRESS

The Compress instructions sequentially copy all of the contents of one record item into another, while excluding null characters and the character specified in the literal operand during the copying operation. In addition, they either left- or right-align the copied characters into the receiving item and, fill in the remaining item locations with the character specified in the second literal operand.

APPEND

The Append instructions sequentially copy record items or all of a record into a buffer, and two of them provide for excluding nulls and a specified character during the copying operation. These instructions are very useful for the construction or build-up of large blocks of data from smaller, well-structured items, such as when writing onto magnetic tape.

EXTRACT

The Extract instructions sequentially copy part or all of a buffer area into a record item, with the fill-in of a specified character in one of the instructions. These instructions are primarily used for extracting record-size portions from the input buffer containing a large data block, such as received from magnetic tape input.

The Sequential Editing instructions include the following:

- Compress Item, Left-Align, Fill (1-100)
- Compress Item, Right-Align, Fill (1-101)
- Append, Right Eliminate (1-102)
- Append, Advance (1-103)
- Append, Left Eliminate (1-105)
- Extract Variable Length Item, Fill (1-107)
- Extract Previous Item (1-111)
- Extract Item (1-112)
- Extract Item, Advance (1-114)

SEQUENTIAL EDITING

Compress Item, Left-Align, Fill

Mnemonic Op Code = CP

Octal Op Code = 014

PURPOSE: To eliminate the null and a specified character during the copying of one item into another, with left-alignment and character fill.

FORMAT:

OC	OP1	OP2	OP3	OP4
014	AR/I	AR/I	L_i	L_f

OPERATION: A copy of the contents of the OP1 item is moved left-aligned into the OP2 item. The null characters and characters which match the OP3 literal character are *not* transferred. Any remaining locations of OP2 are filled with the OP4 literal character. The operation terminates when either all of OP1 is transferred or OP2 becomes full.

EXAMPLE: No. 1

OP1	$XYZn$E$
OP2	ZZZZZZZ
OP3	$
OP4	*
OP2 after	XYZE***

No. 2

OP1	$XYZn$E$
OP2	ZZ
OP3	$
OP4	*
OP2 after	XY

SEQUENTIAL EDITING

Compress Item, Right-Align, Fill

Mnemonic Op Code = CPR

Octal Op Code = 015

PURPOSE: To eliminate the null and a specified character during the copying of one item into another, with right-alignment and character fill.

FORMAT:

OC	OP1	OP2	OP3	OP4
015	AR/I	AR/I	L_i	L_f

OPERATION: A copy of the contents of the OP1 item is moved right-aligned, into the OP2 item. The null characters and characters which match the OP3 literal character are *not* transferred. Any remaining locations of OP2 are filled with the OP4 literal character. The operation terminates when either all of OP1 is transferred or OP2 becomes full.

EXAMPLE: No. 1

OP1	$XYZn$E$
OP2	ZZZZZZZ
OP3	$
OP4	*
OP2 after	***XYZE

No. 2

OP1	$XYZn$E$
OP2	ZZ
OP3	$
OP4	*
OP2 after	ZE

SEQUENTIAL EDITING

Append, Right-Eliminate

Mnemonic Op Code = APR

Octal Op Code = 120

PURPOSE: To select a record item and copy the data into a buffer.

FORMAT:

OC	OP1	OP2	OP3
120	AR/I	B	L

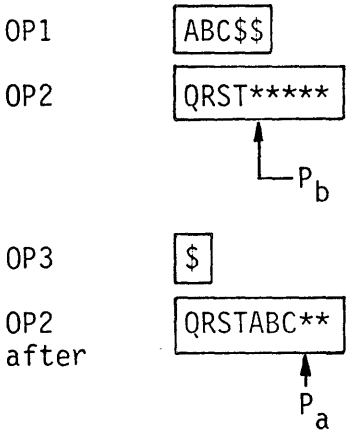
OPERATION: The trailing characters of the OP1 item that match the literal are eliminated. The remainder of OP1 is copied into the OP2 buffer, left-aligned, starting at the current address pointer of the buffer. The current address pointer for the buffer is incremented by the number of characters transferred.

Condition Designators:

EQUAL is set when the end of the OP2 buffer is reached concurrently with the end of the OP1 item. The current address pointer is advanced. The execution of another Append instruction will *not* set the EQUAL or the ABNORMAL EDIT designations.

ABNORMAL EDIT is set when there are more characters from OP1 to be transferred than the OP2 buffer can hold. The current address pointer is *not* advanced. OP2 contains the partial transfer.

EXAMPLE:



Normal operations: *no* designators are set

SEQUENTIAL EDITING

Append, Advance

Mnemonic Op Code = APA

Octal Op Code = 121

PURPOSE: To copy the data from a record item into a buffer.

FORMAT:

OC	OP1	OP2
121	AR/I	B

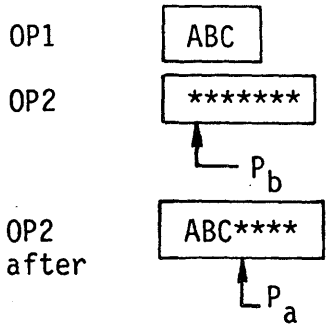
OPERATION: A copy of the OP1 item is moved into the OP2 buffer. The current address pointer for the buffer is incremented by the number of characters transferred.

Condition Designators:

EQUAL is set when the end of the OP2 buffer has been reached concurrently with the end of the OP1 item. The current address pointer is advanced. The execution of another Append instruction will *not* set the EQUAL or the ABNORMAL EDIT designations.

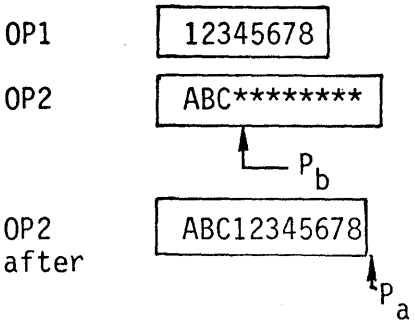
ABNORMAL EDIT is set when there are more characters from OP1 to be transferred than the OP2 buffer can hold. The current address pointer is *not* advanced. OP2 contains the partial transfer.

EXAMPLES: No. 1



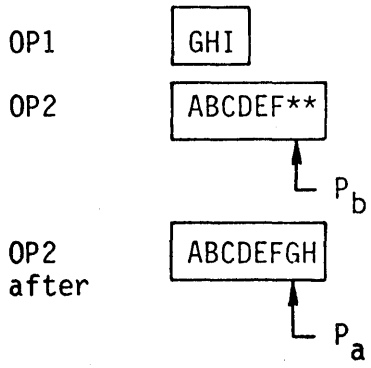
Normal operation: *no* designators are set

No. 2



End of buffer operation: EQUAL is set. P_a points to the location after that of the "8" character.

No. 3



Abnormal operation: ABNORMAL EDIT is set. The entire OP1 item *cannot* be moved into the remaining OP2 buffer space.

SEQUENTIAL EDITING

Append, Left-Eliminate

Mnemonic Op Code = APE

Octal Op Code = 122

PURPOSE: To select a record item and copy the data into a buffer.

FORMAT:

OC	OP1	OP2	OP3
122	AR/I	B	L

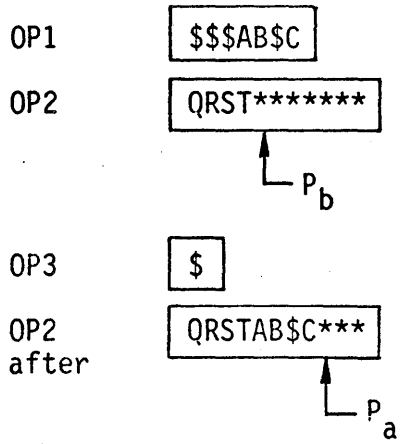
OPERATION: The leading characters of the OP1 item that match the literal are eliminated. The remainder of OP1 is copied left-aligned into the OP2 buffer. The current address pointer for the buffer is incremented by the number of characters transferred.

Condition Designators:

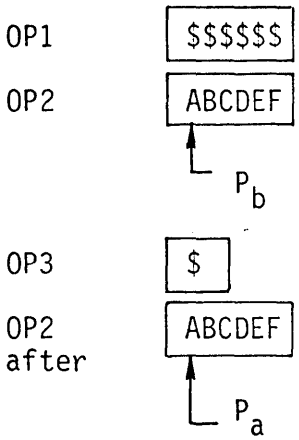
EQUAL is set when the end of the OP2 buffer is reached concurrently with the end of the OP1 item. The current address pointer is advanced. The execution of another Append instruction will *not* set the EQUAL or the ABNORMAL EDIT designations.

ABNORMAL EDIT is set when there are more characters from OP1 to be transferred than the OP2 buffer can hold. The current address pointer is *not* advanced. OP2 contains the partial transfer.

EXAMPLES: No. 1



No. 2



Both are normal operations: *no* designators are set.

SEQUENTIAL EDITING

Extract Variable Length Item, Fill

Mnemonic Op Code = EXV

Octal Op Code = 130

PURPOSE: To select a portion of a buffer, as indicated by a sentinel character, and copy the data into a record item with a character fill.

FORMAT:

OC	OP1	OP2	OP3	OP4
130	B	AR/I	L_s	L_f

OPERATION: A copy from the OP1 buffer area is moved left-aligned into the OP2 item starting at the location in the current address pointer of the buffer and continuing up to the location before the "sentinel" literal (L_s). The remaining locations of OP2 are filled with the literal (L_f). The current address pointer of the buffer is incremented to the address following the sentinel literal. The sentinel is *not* transferred. If the sentinel is the last byte in the buffer, the current address pointer is advanced *only* to the address of the sentinel itself.

Condition Designators:

EQUAL is set when the end of the OP1 buffer is reached by the time the OP2 item is filled. The current address pointer of the buffer is *not* advanced. The remaining positions of the OP2 item are filled-in with the OP4 literal.

ABNORMAL EDIT is set when no sentinel is encountered in the OP1 buffer by the time the OP2 item is filled. The current address pointer of the buffer is *not* advanced. OP2 contains the characters already transferred.

NOTE

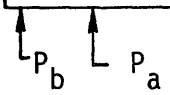
When EQUAL alone is set, the current address pointer is at the last byte of the buffer, which is a sentinel. OP2 will contain only the fill character. Further Extract-Variable instructions will have the same result.

When ABNORMAL EDIT alone is set, OP2 is too short to receive all characters in OP1 before the sentinel character. OP2 contains the partial transfer.

When both the EQUAL and the ABNORMAL EDIT designators are set, OP1 did *not* have any remaining sentinel characters in it.

EXAMPLES: No. 1

OP1 [ABC,1234,,XY]



OP2 [*******]

OP3 [,]

OP4 [n]

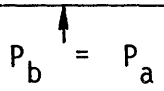
OP2 after [ABCnnnn]

Normal operation: *no* designators are set. The OP1 pointer is moved one location past the comma.

For the follwoing examples, OP3 = [,] and OP4 = [n]

No. 2

OP1 [ABC,1234,,XY]

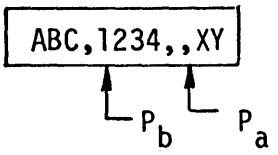


OP2 [***]

OP2 after [123]

Abnormal operation: ABNORMAL EDIT is set. The OP1 pointer is *not* moved. The end of OP2 is reached prior to finding a comma in OP1.

No. 3

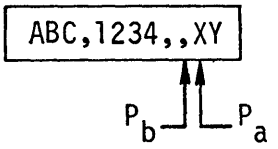
OP1 

OP2 ****

OP2 after 1234

Normal operation: *no* designators are set. The OP1 pointer is moved one location past the first comma encountered.

No. 4

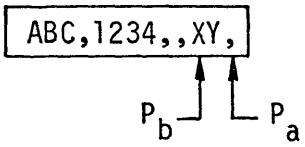
OP1 

OP2 **

OP2 after nn

Normal operation: *no* designators are set. *No* characters are transferred from OP1 to OP2, since the comma is found immediately. The OP1 pointer is moved one position past the comma.

No. 5

OP1 

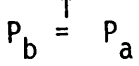
OP2 ***

OP2 after XYn

Normal operation: *no* designators are set. The pointer is at the comma, since it is the last character in OP1.

No. 6

OP1 | ABC,1234,,XY,



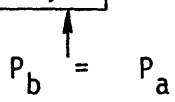
OP2 | ***

OP2 after | nnn

End of buffer operation: EQUAL is set. This implies that all of the characters have been copied from OP1. The pointer is *not* moved.

No. 7

OP1 | ABC,1234,XY



OP2 | ****

OP2 after | XYnn

Abnormal operation: EQUAL and ABNORMAL EDIT are set. *No* comma was found at the end of OP1.

SEQUENTIAL EDITING

Extract Previous Item

Mnemonic Op Code = EXP

Octal Op Code = 131

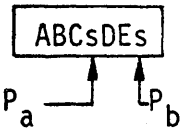
PURPOSE: To select a portion of a buffer and copy the data into a record item.

FORMAT:

OC	OP1	OP2
131	B	AR/I

OPERATION: A copy from the OP1 buffer area is moved right-aligned into the OP2 item and starting from the current address pointer-1 and decrementing it to the last character transferred. The operation is terminated when OP2 is full. *No* designators are set.

EXAMPLE:

OP1 

OP2 

OP2 after 

SEQUENTIAL EDITING

Extract Item

Mnemonic Op Code = EX

Octal Op Code = 132

PURPOSE: To copy the characters from a buffer area into a record item, without incrementing the current address pointer.

FORMAT:

OC	OP1	OP2
132	B	AR/I

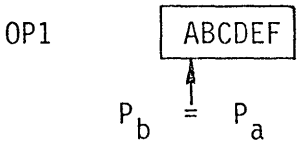
OPERATION: A copy of the OP1 buffer, from the location in the current address pointer, is moved left-aligned into the OP2 item. The current address pointer is *not* changed: $P_a = P_b$. The operation is terminated when the end of OP1 or OP2 is reached.

Condition Designators:

EQUAL is set when the end of the OP1 buffer is reached concurrently with the end of the OP2 item.

ABNORMAL EDIT is set when the end of the OP1 buffer is reached prior to the end of the OP2 item.

EXAMPLES: No. 1

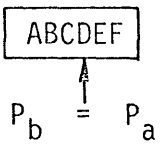


OP2 ***

OP2 after ABC

Normal operation: *no* designators are set.

No. 2

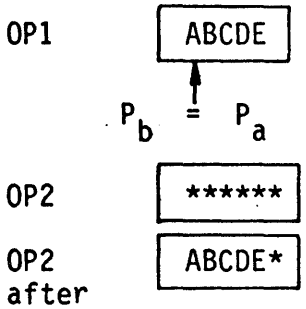


OP2 ***

OP2 after DEF

EQUAL is set.

No. 3



ABNORMAL EDIT is set. There are *not* enough characters in OP1 to fill OP2.

SEQUENTIAL EDITING

Extract Item, Advance

Mnemonic Op Code = EXA

Octal Op Code = 133

PURPOSE: To copy the characters from a buffer area into a record item.

FORMAT:

OC	OP1	OP2
133	B	AR/I

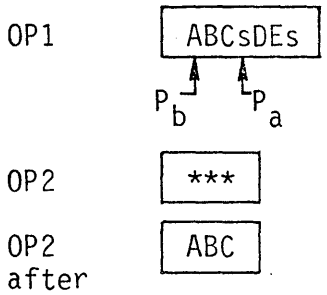
OPERATION: A copy of the OP1 buffer, from the location in the current address pointer, is moved left-aligned into the OP2 item. The current address pointer of the buffer is incremented to the location following the last character transferred. The operation is terminated when either the end of the OP1 buffer is reached or the OP2 item is full.

Condition Designators:

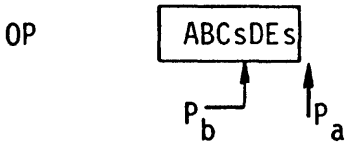
EQUAL is set when the end of the OP1 buffer is reached concurrently with the end of the OP2 item. The current address pointer is advanced. The execution of another Extract Item, Advance instruction will *not* set the EQUAL or ABNORMAL EDIT designators.

ABNORMAL EDIT is set when the end of the OP1 buffer is reached prior to the end of the OP2 item.

EXAMPLES: No. 1

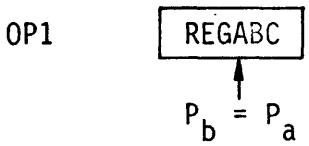


No. 2



End of buffer operation: EQUAL is set. The pointer is adjusted to one more than the last location in OP1.

No. 3



Abnormal operation: ABNORMAL EDIT is set. There are *not* enough characters in OP1 to fill OP2.

Interrupt-related instructions are used to interpret, control and process events (interrupts) that divert the processor from main program execution. The Interrupt instructions include the following:

- GOTO on Service Request (1-117)
- GOTO on Channel Interrupt (1-118)
- Swap States (1-119)
- Set Interrupt Lockout (1-120)
- Clear Interrupt Lockout (1-121)
- Interrupt Mask (1-122)
- Interrupt Branch GOTO (1-124)

Refer to Appendix A, Interrupt Processing, for a detailed description for using the interrupt set of instructions.

INTERRUPT
GOTO On Service Request
Mnemonic Op Code = GSI
Octal Op Code = *113

PURPOSE: To branch to a subroutine if a service request for a specified channel has been raised prior to this instruction.

FORMAT:

OC	OP1	Branch to
113	AR/I	address

OPERATION: OP1 specifies the channel that is checked for a service request (see page A-6). If a service request is stored in its service request storage and the channel specified by OP1 matches the requested one, the execution of the GSI instruction forces a branch to the address specified in OP2 and OP3. If a request is not present, the following instruction is executed.

EXAMPLE:

OC	OP1	Branch to	
113	234	002	300

P-Bias = 010-000

OP1

0	0	0	0	0	1	1	0

Item 34 of Active Record 2

Location 010-070 contains the above instruction and it is being executed. If a service request is stored for channel 6, execution is resumed at location 012-300; otherwise, it continues with 010-074.

INTERRUPT

GOTO On Channel Interrupt (Monitor Interrupt)

Mnemonic Op Code = GCI

Octal Op Code = *117

PURPOSE: To branch to a specified address if a channel interrupt for a specified channel has been monitored.

FORMAT:

OC	OP1	Branch to
117	AR/I	address

OPERATION: If a channel interrupt has been monitored prior to this instruction on a channel specified in OP1, the interrupt is cleared and program execution resumes at the specified address.

Otherwise, the next program instruction is executed. The channel interrupt is set whenever the channel goes inactive, such as when a buffer is terminated.

EXAMPLE:

OC	OP1	Branch to	
117	104	006	230

P-Bias = 010-000

OP1

0	0	0	0	0	1	0	0

Item 4 of Active Record 1

Location 011-224 contains the above instruction and it is being executed. If a monitor interrupt is stored for channel 4, execution is resumed at location 016-230 and the monitor interrupt is cleared; otherwise, execution continues with 011-230.

INTERRUPT

Swap States

Mnemonic Op Code = SWS

Octal Op Code = *154

PURPOSE: To change the processor from worker state to executive state or from executive state to worker state.

FORMAT:

OC

154

OPERATION: The 502 Processor has two general states which are reflected in two different sets of Active Records. The worker state uses core memory locations 000-000 through 000-017 and the executive state uses 000-040 through 000-057 to store the Active Records.

In both states, the Program Control Block is the image of the state's hardware set of Active Records. A Swap States instruction forces the image of the alternate-state core Active Records into the hardware Active Records, thereby eliminating the need to have two different sets of hardware Active Records.

A Swap States instruction is executed immediately if the present state is the worker state. If the present state is the executive state, the SWS instruction is not executed until the instruction following the SWS instruction is performed, so that information can be retrieved from the executive state (such as an interrupt return address).

The hardware keeps track of its current state automatically. The software keeps track by its design. Any power-up, restart, or P-start forces the processor to the worker state.

INTERRUPT

Set Interrupt Lockout

Mnemonic Op Code = SIL

Octal Op Code = *155

PURPOSE: To lockout all interrupts.

FORMAT:

OC

155

OPERATION: This instruction causes all interrupts to be locked out. This condition may only be cleared by a Clear Interrupt Lockout instruction or an Interrupt Branch GOTO instruction. The storage of incoming interrupts of any type is, however, not affected by this instruction.

INTERRUPT

Clear Interrupt Lockout

Mnemonic Op Code = CIL

Octal Op Code = *157

PURPOSE: To enable interrupts.

FORMAT:

OC

157

OPERATION: The interrupt lockout is cleared in the hardware. However, all interrupts locked out prior to this instruction by an Interrupt Mask instruction remain locked out.

INTERRUPT

Interrupt Mask

Mnemonic Op Code = IM

Octal Op Code = *174

PURPOSE: To selectively enable or disable interrupts by a specified mask.

FORMAT:

OC	OP1
174	AR/I

OPERATION: The OP1 item defines a three-byte item mask; Monitor, Service and Special interrupts. Each byte is bit encoded. A "zero" bit enables the processor to honor the interrupt. A "one" bit causes the processor to ignore the interrupt. The normal instruction sequence to change the mask is:

```
SIL
IM    desired-mask
```

The byte and associated bit assignments are as follows:

BIT POSITION	MONITOR BYTE 1 (MSBY)	SERVICE BYTE 2	SPECIAL BYTE 3 (LSBY)
Ø	I/O Channel Ø	I/O Channel Ø	Non-Operational Sub-Op Code
1	I/O Channel 1	I/O Channel 1	Not assigned
2	I/O Channel 2	I/O Channel 2	Delta Clock
3	I/O Channel 3	I/O Channel 3	Not assigned
4	I/O Channel 4	I/O Channel 4	Not assigned
5	I/O Channel 5	I/O Channel 5	Machine Interrupt
6	I/O Channel 6	I/O Channel 6	BDMA Channel 6
7	I/O Channel 7	I/O Channel 7	BDMA Channel 7

EXAMPLE:

OC	OP1
174	201

	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0		
OP1	1	1	0	0	0	1	1	1	1	0	0	1	0	1	1	0	1	0	0	1	1	0	1	0	Binary	Item 1 of Active Record 2
	307								226								132								Octal	

MONITOR INTERRUPTS	Enabled	Disabled
	Channel 3	Channel Ø
	Channel 4	Channel 1
	Channel 5	Channel 2
		Channel 6
		Channel 7

SERVICE INTERRUPTS	Enabled	Disabled
	Channel Ø	Channel 1
	Channel 3	Channel 2
	Channel 5	Channel 4
	Channel 6	Channel 7

SPECIAL INTERRUPTS	Enabled	Disabled
	Non-Operational Sub-Op Code	BDMA Channel 7
	Delta Clock	
	Machine Interrupt	
	BDMA Channel 6	

Interrupt Branch GIR

Mnemonic Op Code = GIR

Octal Op Code = *177

PURPOSE: To return from an interrupt subroutine to the routine prior to the branch, or back into a required routine, while also restoring the P-address needed for program execution. This P-address is read from a push-down stack defined by OP1. The GIR instruction is normally preceded by a Swap States instruction. This instruction is similar to the GRT instruction except interrupts are enabled with this instruction. Only those interrupts which are allowed by the interrupt mask are enabled

FORMAT:

OC	OP1
177	B

OPERATION: OP1 defines a push-down stack that contains the return address. Push-down buffer locations are two bytes long and contain the desired return address. It is located at the current buffer address minus 1 and minus 2

EXAMPLE:

OC	OP1
177	001

P-Bias = 012-000

P-address before = 013-077

P-address after = 012-265

SDAT Entry

OP1 before 001 | 010-002 | 010-377 | PUSH-DOWN BUFFER DESCRIPTOR

PUSH-DOWN BUFFER

Pa	010-000	012-265
Pb	010-002	XXX-XXX
	⋮	⋮
	010-376	XXX-XXX

SDAT Entry

OP1 after 001 | 010-000 | 010-377 | PUSH-DOWN BUFFER DESCRIPTOR

EXTERNAL EXECUTE INSTRUCTION SET

The programmable functions of the 502B Processor are expanded by incorporating additional hardware modules. Modules are available for the following functions:

- Multiply and Divide Decimal or Binary
- Binary-to-Decimal and Decimal-to-Binary conversion
- Delta Clock (interrupting interval timer)
- Parity Error Determination
- External Execute Instruction Error Detection

These functions are implemented by using the External Execute instruction. This instruction contains a sub-op code in addition to the normal operational code (145). The format of the External Execute instruction is as follows:

OC	SUB-OP	OP1	OP2	OP3	OP4
145	XXX	AR/I	AR/I	AR/I	L

The External Execute instructions include the following:

Mnemonic	Operation Code (OC)	Sub-Op Code	Instruction	Page
LC	145	004	Load Delta Clock	1-127
SEE	145	014	Store External Instruction Error	1-128
SCE	145	015	Store Channel Parity Error	1-131
MB	145	020	Multiply Binary	1-132
MLB	145	021	Multiply Literal Binary	1-133
DB	145	022	Divide Binary	1-134
DLB	145	023	Divide Literal Binary	1-135
MD	145	024	Multiply Decimal	1-136
MLD	145	025	Multiply Literal Decimal	1-137
DD	145	026	Divide Decimal	1-138
DLD	145	027	Divide Literal Decimal	1-139
BTD	145	030	Binary to Decimal	1-140
DTB	145	031	Decimal to Binary	1-141
SDR	145	034	Store Decimal Remainder	1-142
SBR	145	035	Store Binary Remainder	1-143

MULTIPLE/DIVIDE INSTRUCTIONS

The operation of multiply and divide instructions is critical regarding operand sizes, since no indication is given if the storage item for the result is too small. The programmer must, therefore, make certain that the storage item has sufficient length.

The following formulas may be used to determine the operand size:

For Multiply:

[maximum number of digits in OP1] + [maximum number of digits in OP2] = [number of digits in product]

For Divide:

[maximum number of digits in OP1] - [maximum number of digits in OP2 (after eliminating leading zeros)] +1 = [number of digits in quotient]

UNUSED OPERANDS

Unused operand must specify valid operands because the processor generates the addresses for these operands, even though they are not used. A value of 000 is always valid.

EXTERNAL EXECUTE

Load Delta Clock

Mnemonic Op Code = LC

Octal Op Code = *145

Octal Sub-Op Code = 004

PURPOSE: To load the Delta Clock with a time interval.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	004	AR/I	---	---

OP1 = 2-byte item containing an initial time interval (in binary)

OP2 = Not used

OP3 = Not used

OPERATION: When this instruction is executed, the 2-byte time interval (OP1 item) is loaded into the Delta Clock. The loading of the clock initializes the clock and the time interval is decremented by one every 100 microseconds (± 0.5%). When the time interval has decremented to zero, a class 3 interrupt occurs and the clock is deactivated. The Delta Clock has a time range of 100 microseconds to 6.5536 seconds. The clock may be deactivated at any time by loading it with a 2-byte binary zero; no interrupt will occur.

EXTERNAL EXECUTE

Store External Instruction Error

Mnemonic Op Code = SEE

Octal Op Code = *145

Octal Sub-Op Code = 014

PURPOSE: To obtain and store the status and error data of Class 3 type interrupts.

FORMAT:

OC	SUB-OP	OP1	OP2	OP3
145	014	---	---	AR/I

OP1 = Not used

OP2 = Not used

OP3 = 14-byte storage item for interrupt status

OPERATION: When this instruction is executed, a 14-byte item containing class 3 type interrupt status and associated error data is stored in the OP3 item as shown below:

	MSBY			LSBY
OP3	Byte 1	Byte 2	. . . . .	Byte 14

OP3 Item	Contents
1 (MSBY)	Sub-Op Code of instruction
2	Interrupt Status byte
3	OP1 Lower Address Limits (bits 8-15)
4	OP1 Lower Address Limits (bits 0-7)
5	OP1 Upper Address Limits (bits 8-15)
6	OP1 Upper Address Limits (bits 0-7)
7	OP2 Lower Address Limits (bits 8-15)
8	OP2 Lower Address Limits (bits 0-7)
9	OP2 Upper Address Limits (bits 8-15)
10	OP2 Upper Address Limits (bits 0-7)
11	OP3 Lower Address Limits (bits 8-15)
12	OP3 Lower Address Limits (bits 0-7)
13	OP3 Upper Address Limits (bits 8-15)
14 (LSBY)	OP3 Upper Address Limits (bits 0-7)

Byte 2 (Interrupt Status Byte) specifies the type of class 3 interrupt that has occurred. The class 3 type interrupts are described as follows:

BYTE 2 Bit Position	Contents
0	Non-operational Sub-Op Code
1	Not assigned
2	Delta Clock Interrupt
3	Not assigned
4	Not assigned
5	Machine Check Interrupt
6	Direct Memory Access Channel (DMA) 6 Interrupt
7	Direct Memory Access Channel (DMA) 7 Interrupt

Non-operational Sub-Op Code (2^0) - This bit is set under the following two conditions:

1. The Sub-Op Code in the instruction causing the interrupt was not in the processor's repertoire of instructions (an illegal instruction) and therefore cannot be executed, or
2. The required hardware modules are not in the processor to execute the Sub-Op Code (unavailable instruction).

In either event, the Sub-Op code is contained in byte 1 and bytes 3 through 14 contain the absolute beginning and ending addresses for the OP1, OP2 and OP3 items of the instruction that caused the interrupt.

The programmer, by software methods, can determine which of the above two conditions caused the interrupt (contents of byte 1) and in turn take appropriate action. If the instruction cannot be executed due to lack of hardware modules, the programmer may elect to include software to perform the same operations as the External Instruction that could not be executed. The recovery from this condition is simplified in that the Sub-Op Code and the absolute address limits of the OP1, OP2 and OP3 items are defined for the instruction that could not be executed.

CAUTION

A unique condition exists when an Execute External instruction containing four operands cannot be executed. This condition causes the Program Pointer (P) to be off by one byte which must be corrected by adding one (+1) to the value of P. See Appendix A, Interrupt Programming for a detailed description of this condition (Class 3 - Special Interrupts).

Delta Clock (2^2) - This bit is set when the Delta Clock counts down to zero (see the Load Delta Clock instruction).

Machine Check (2^5) - This bit is set when a memory, I/O Selector channel or BDMA channel parity error occurs.

BDMA Channel 6 (2^6) - This bit is set by the device attached to BDMA channel 6 when it requires service from the processor.

BDMA Channel 7 (2^7) - This bit is set by the device attached to BDMA channel 7 when it requires service from the processor.

NOTE

Byte 1 of OP3 contains the sub-op code of this instruction (014), unless the interrupt type was a Non-Operational Sub-Op Code (bit 2^0 set in byte 2).

EXTERNAL EXECUTE

Store Channel Parity Error

Mnemonic Op Code = SCE

Octal Op Code = *145

Octal Sub-Op Code = 015

PURPOSE: To obtain and store I/O and DMA channel parity error status.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	015	---	---	AR/I

OP1 = Not used

OP2 = Not used

OP3 = 1-byte storage item for channel parity error status

OPERATION: When this instruction is executed, a 1-byte item containing I/O and DMA channel parity error status is stored in the OP3 item. The contents of the 1-byte item is bit encoded as follows:

Bit Position	Meaning
0 1 2	Specifies the I/O channel on which the parity error occurred
3	Not assigned
4 5 6	110_2 = BDMA channel 6 parity error 111_2 = BDMA channel 7 parity error
7	0 = BDMA data or status parity erro 1 = BDMA address parity error

EXTERNAL EXECUTE

Multiply Binary

Mnemonic Op Code = MB

Octal Op Code = *145

Octal Sub-Op Code = 020

PURPOSE: To multiply a binary number by another.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	020	AR/I	AR/I	AR/I

OP1 = MULTIPLICAND - 1- to 5-byte binary item

OP2 = MULTIPLIER - 1- to 5-byte binary item

OP3 = PRODUCT - 1- to 10-byte binary item

OPERATION: The contents of the OP1 item (multiplicand) are multiplied by the contents of the OP2 item (multiplier) and the product is stored into the OP3 item. The sign of the product is computed algebraically and extended to the MSB of the OP3 item. If the OP3 item is too small to contain the product of OP1 and OP2, the result is indeterminate.

EXTERNAL EXECUTE

Multiply Literal Binary

Mnemonic Op Code = MLB

Octal Op Code = *145

Octal Sub-Op Code = 021

PURPOSE: To multiply a binary number in core storage by a binary number contained in the instruction itself.

OC	SUB-OC	OP1	OP2	OP3	OP4
145	021	AR/I	---	AR/I	L

OP1 = MULTIPLICAND - 1- to 5-byte binary item

OP2 = Not used

OP3 = PRODUCT - 1- to 6-byte binary item

OP4 = MULTIPLIER - 1-byte binary literal

OPERATION: The contents of the OP1 item (multiplicand) are multiplied by the contents of the OP4 item (multiplier) and the product is stored into the OP3 item. The sign of the product is computed algebraically and extended to the MSB of the OP3 item. If the OP3 item is too small to contain the product of OP1 and OP4, the result is indeterminate.

EXECUTE EXECUTE

Divide Binary

Mnemonic Op Code = DB

Octal Op Code = *145

Octal Sub-Op Code = 022

PURPOSE: To divide a binary number by another.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	022	AR/I	AR/I	AR/I

OP1 = DIVIDEND - 1- to 10-byte (not counting leading zeros) binary item

OP2 = DIVISOR - 1- to 5-byte binary item

OP3 = QUOTIENT - 1- to 5-byte binary item

REMAINDER - see Store Binary Remainder (SBR) instruction.

OPERATION: The contents of the OP1 item (dividend) are divided by the contents of the OP2 item (divisor) and the quotient is stored into the OP3 item. The sign of the quotient is computed algebraically and extended to the MSB of the OP3 item. If the OP3 item is too small to contain the quotient of OP1 divided by OP2, the result is indeterminate.

The "remainder" of a divide binary operation may be obtained by using the Store Binary Remainder (SBR) instruction.

EXTERNAL EXECUTE

Divide Literal Binary

Mnemonic Op Code = DLB

Octal Op Code = *145

Octal Sub-Op Code = 023

PURPOSE: To divide a binary number in core storage by a binary number contained in the instruction itself.

FORMAT:

OC	SUB-OP	OP1	OP2	OP3	OP4
145	023	AR/I	---	AR/I	L

OP1 = DIVIDENT - 1- to 6-byte (not counting leading zeros) binary item

OP2 = Not used

OP3 = QUOTIENT - 1- to 5-byte binary item

OP4 = DIVISOR - 1- binary literal

REMAINDER - see Store Binary Remainder (SBR) instruction.

OPERATION: The contents of the OP1 item (dividend) are divided by the contents of the OP4 item (divisor) and the quotient is stored into the OP3 item. The sign of the quotient is computed algebraically and extended to the MSB of the OP3 item. If the OP3 item is too small to contain the quotient of OP1 divided by OP4, the result is indeterminate.

The "remainder" of a divide literal binary operation may be obtained by using the Store Binary Remainder (SBR) instruction.

EXTERNAL EXECUTE

Multiply Decimal

Mnemonic Op Code = MD

Octal Op Code = *145

Octal Sub-Op Code = 024

PURPOSE: To multiply a decimal number by another.

FORMAT:

OC	SUB-OP	OP1	OP2	OP3
145	024	AR/I	AR/I	AR/I

OP1 = MULTIPLICAND - 1- to 12-byte unpacked decimal item

OP2 = MULTIPLIER - 1- to 12-byte unpacked decimal item

OP3 = PRODUCT - 1- to 24-byte unpacked decimal item

OPERATION: The contents of the OP1 item (multiplicand) are multiplied by the contents of the OP2 item (multiplier) and the product is stored into the OP3 item. The sign of the product is computed algebraically and is set into the sign zone of the LSBY of OP3. If the OP3 item is too small to contain the product of OP1 and OP2, the result is indeterminate.

EXTERNAL EXECUTE

Multiply Literal Decimal

Mnemonic Op Code = MLD

Octal Op Code = *145

Octal Sub-Op Code = 025

PURPOSE: To multiply a decimal number in core storage by a decimal number contained in the instruction itself.

FORMAT:

OC	SUB-OP	OP1	OP2	OP3	OP4
145	025	AR/I	---	AR/I	L

OP1 = MULTIPLICAND - 1- to 12-byte unpacked decimal item

OP2 = Not used

OP3 = PRODUCT - 1- to 13-byte unpacked decimal item

OP4 = MULTIPLIER - 1-byte unpacked decimal literal

OPERATION: The contents of the OP1 item (multiplicand) are multiplied by the contents of the OP4 item (multiplier) and the product is stored into the OP3 item. The sign of the product is computed algebraically and is set into the sign zone of the LSBY of OP3. If the OP3 item is too small to contain the product of OP1 and OP4, the result is indeterminate.

EXTERNAL EXECUTE

Divide Decimal

Mnemonic Op Code = DD

Octal Op Code = *145

Octal Sub-Op Code = 026

PURPOSE: To divide a decimal number by another.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	026	AR/I	AR/I	AR/I

OP1 = DIVIDEND - 1- to 24-byte (not counting leading zeros) unpacked decimal item

OP2 = DIVISOR - 1- to 12-byte unpacked decimal item

OP3 = QUOTIENT - 1- to 12-byte unpacked decimal item

REMAINDER - see Store Decimal Remainder (SDR) instruction.

OPERATION: The contents of the OP1 item (dividend) are divided by the contents of the OP2 item (divisor) and the quotient is stored into the OP3 item. The sign of the quotient is computed algebraically and is set into the sign zone of the LSBY of OP3. If the OP3 item is too small to contain the quotient of OP1 divided by OP2, the result is indeterminate.

The "remainder" of a divide decimal operation may be obtained by using the Store Decimal Remainder (SDR) instruction.

EXTERNAL EXECUTE

Divide Literal Decimal

Mnemonic Op Code = DLD

Octal Op Code = *145

Octal Sub-Op Code = 027

PURPOSE: To divide a decimal number in core storage by a decimal number contained in the instruction itself.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3	OP4
145	027	AR/I	---	AR/I	L

OP1 = DIVIDEND - 1- to 13-byte (not counting leading zeros) unpacked decimal item

OP2 = Not used

OP3 = QUOTIENT - 1- to 12-byte unpacked decimal item

OP4 = DIVISOR - 1-byte unpacked decimal item

REMAINDER - see Store Decimal Remainder (SDR) instruction.

OPERATION: The contents of the OP1 item (dividend) are divided by the contents of the OP4 item (divisor) and the quotient is stored in the OP3 item. The sign of the quotient is computed algebraically and is set into the sign zone of the LSBY of OP3. If the OP3 item is too small to contain the quotient of OP1 divided by OP4, the result is indeterminate.

The "remainder" of a divide literal decimal operation may be obtained by using the Store Decimal Remainder (SDR) instruction.

EXTERNAL EXECUTE

Binary to Decimal

Mnemonic Op Code = BTD

Octal Op Code = *145

Octal Sub-Op Code = 030

PURPOSE: To convert a binary number to an unpacked decimal number.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	030	AR/I	---	AR/I

OP1 = 1- to 10-byte binary item

OP2 = Not used

OP3 = 1- to 24-byte unpacked decimal item

OPERATION: The contents of the OP1 item are converted to an unpacked decimal number and stored in the OP3 item. The sign of the OP1 item is set in the sign zone of the LSBY of OP3. If the OP3 item is too small to contain the converted number, the result is indeterminate.

EXTERNAL EXECUTE

Decimal to Binary

Mnemonic Op Code = DTB

Octal Op Code = *145

Octal Sub-Op Code = 031

PURPOSE: To convert an unpacked decimal number to a binary number.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	031	AR/I	---	AR/I

OP1 = 1- to 24-byte unpacked decimal item

OP2 = Not used

OP3 = 1- to 10-byte binary item

OPERATION: The contents of the OP1 item are converted to a binary number and stored in the OP3 item. The sign of the OP1 item is extended to the MSB of the OP3 item. If the OP3 item is too small to contain the converted number, the result is indeterminate.

EXTERNAL EXECUTE

Store Decimal Remainder

Mnemonic Op Code = SDR

Octal Op Code = *145

Octal Sub-Op Code = 034

PURPOSE: To obtain and store the remainder resulting from a decimal divide operation.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	034	---	---	AR/I

OP1 = Not used

OP2 = Not used

OP3 = 1- to 12-byte unpacked decimal item

OPERATION: When this instruction is executed, the remainder, resulting from a decimal divide operation, is stored in the OP3 item. This instruction should immediately follow the execution of the Divide Decimal instruction to insure validity of the remainder. The sign of the remainder is the same as the dividend (see The Divide Decimal instruction) and is set in the LSBY of the remainder. If the OP3 item is too small to contain the remainder, the result is indeterminate.

EXTERNAL EXECUTE

Store Binary Remainder

Mnemonic Op Code = SBR

Octal Op Code = *145

Octal Sub-Op Code = 035

PURPOSE: To obtain and store the remainder resulting from a binary divide operation.

FORMAT:

OC	SUB-OC	OP1	OP2	OP3
145	035	---	---	AR/I

OP1 = Not used

OP2 = Not used

OP3 = 1- to 5-byte binary item

OPERATION: When this instruction is executed, the remainder, resulting from a binary divide operation, is stored in the OP3 item. This instruction should immediately follow the execution of the Binary Divide instruction to insure validity of the remainder. The sign of the remainder is the same as the dividend (see the Binary Divide instruction) and is extended to the MSB of the remainder. If the OP3 item is too small to contain the remainder, the result is indeterminate.

INSTRUCTION EXPANSION MODULES

GENERAL

The programmable functions of the SYSTEM 2400 Processor are expanded by incorporating additional hardware modules. These modules are integrated within the main chassis of the Processor and interface with the logic via the Direct Memory Access (DMA) channels.

Two hardware modules are available with the Processor and are referred to as Instruction Expansion Module A and B (see Figure 1-1).

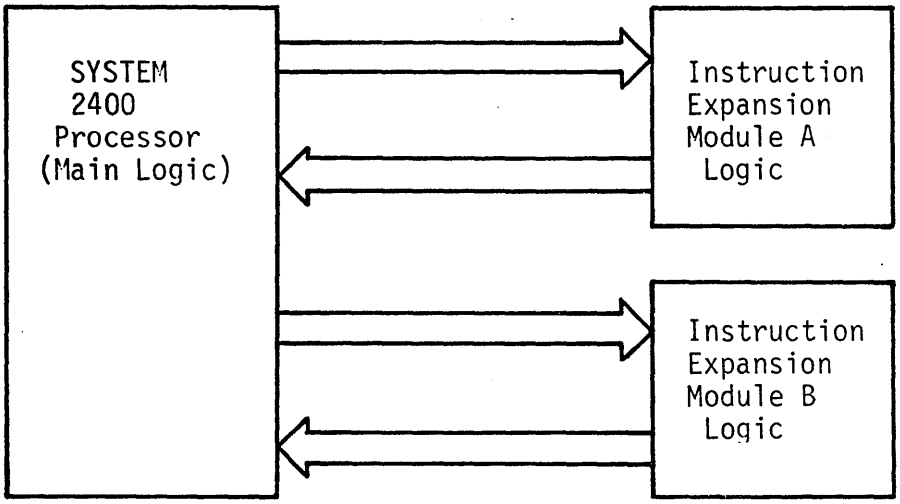


Figure 1-1. SYSTEM 2400 Processor Instruction Expansion Modules

Instruction Expansion Module A (SNAP P Adapter) provides the programmer with a set of instructions to abort the main program, jump to a subroutine, and return to the main program at the point of exit.

Instruction Expansion Module B (Utility Adapter) provides the programmer with a set of instructions to perform the following logical operations:

- Exclusive OR/LRC (Longitudinal Redundancy Check)
- Logical AND, and
- Inclusive OR.
- 16 Bit CRC
- 12 Bit CRC
- Load Utility Adapter

CHANNEL ASSIGNMENTS

Instruction Expansion Modules A and B are connected to Direct Memory Access (DMA) channels 2 and 1, respectively, as shown in Figure 1-2.

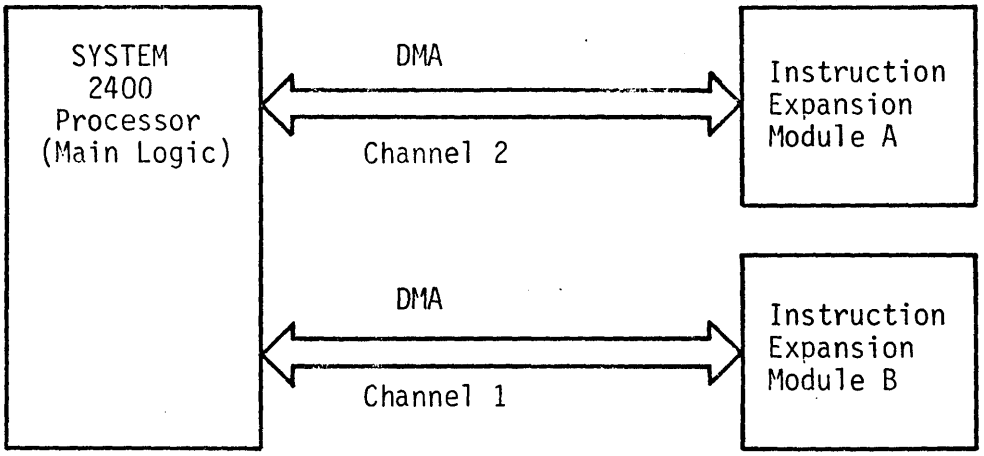


Figure 1-2. Instruction Expansion Modules - Channel Configuration

INFORMATION TRANSFER

The transfer of data and commands between the Processor and connected Instruction Expansion Modules is accomplished using the *Special In* and *Special Out* instructions. These instructions are used to transfer information over the DMA Channels (see Figure 1-3).

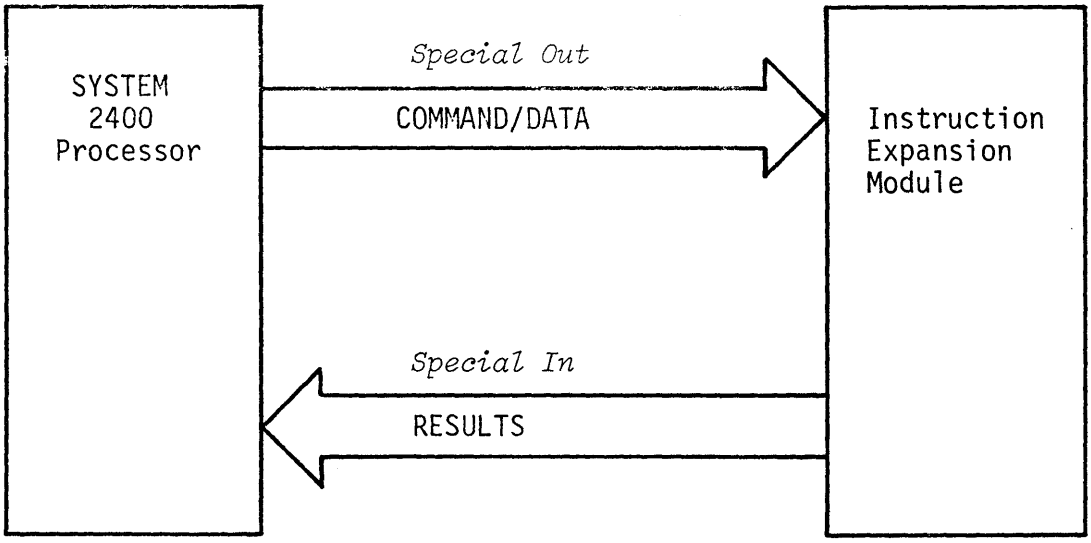


Figure 1-3. Information Transfer

The *Special Out* instruction transfers the command (functions to be performed) and the data to be operated upon by the connected Instruction Expansion Module.

The format of the *Special Out* instruction is shown below:

OC	OP1	OP2
105	AR/I	AR/I

OC = Operational code for the *Special Out* instruction.

OP1 = Operand 1, a 1-byte item containing the DMA channel number over which the data specified by the OP2 item is transferred.

OP2 = Operand 2, a 1- or multi-byte item containing:

- The command code byte (instruction) specifying the function to be performed. The command code must be the first byte transferred in the OP2 item.
- Data bytes conveying the information to be operated upon by the Instruction Expansion Module as specified by the command code byte.

The *Special In* instruction is used to retrieve and store the results (contents) from the Instruction Expansion Module following an operation directed by the command code in a *Special Out* instruction. The format of the *Special In* instruction is shown below:

OC	OP1	OP2
100	AR/I	AR/I

OC = Operational code for the *Special In* instruction.

OP1 = Operand 1, a 1-byte item containing the DMA channel number over which the data is to be received.

OP2 = Operand 2, a multi-byte item into which the contents of the Instruction Expansion Module is to be stored.

Although the format of the *Special Out* and *Special In* instructions remains the same for the various functions performed by the Instruction Expansion Modules, special commands are used to identify each function.

INSTRUCTION EXPANSION MODULE A

GENERAL

Instruction Expansion Module A (SNAP P Adapter)[1] enables the programmer to leave the main program, mump to a subroutine, and return to the main program at the point of exit. This programmed return/jump capability requires that the address of the next instruction (contents of the P-Register) to be executed in main memory be saved before an exit is made to a subroutine.

The contents of the P-Register (saved address) is saved by the Instruction Expansion Module upon execution of the Save P instruction. Execution of the Store P instruction will obtain the saved address and store it in main memory.

The SYSTEM 2400 Assembler provides the macros 'RTN' and 'MDL' to accomplish subroutine linkage without using a DMA channel.

Programmed steps required to implement the return/jump feature via expansion module A are as follows:

Main Program

- Executes a Save P instruction which instructs the Instruction Expansion Module to save the contents of the P-Register.
- Execute a GOTO instruction to jump to the subroutine.

Subroutine

- Execute a Store P instruction to obtain and store the saved address in main memory as the address in a GOTO instruction.
- Execute the subroutine processing instructions.
- Execute a GOTO instruction to exit from the subroutine to a fixed location in memory.

Main Memory

- Execute an instruction to subtract P-bias from the stored return address.
- Execute an instruction to increase the stored return address by plus 3.

---

[1] Appendix G gives an in-depth coverage for users implementing this expansion.

- Execute the GOTO instruction containing the adjusted return address. Return will be to the instruction following the GOTO instruction used to exit from the main program.

The module A instructions include the following:

- Save (P) (1-149)
- Store (P) (1-150)

PROGRAM CALL

Save (P)

Mnemonic Code = SAP or SVP

Octal OP Code = 105

PURPOSE: To obtain and save the contents of the program control register (P).

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

DMA CHANNEL CODE: 002

COMMAND CODE: 002

OPERATION: The output data from the OP2 item is sent to the Instruction Expansion Module connected to the DMA channel specified in OP1.

OP2 is a 1-byte item containing the command code: 002.

The Instruction Expansion Module, upon receipt of the command code, obtains and saves the contents of the P-register. The P-register contains the address of the next instruction to be executed in the program.

EXAMPLE:

OC	OP1	OP2
105	103	104

OP1 Item 3 of Active Record 1 contains the DMA channel number.

OP2 Item 4 of Active Record 1 contains the command code.

PROGRAM CALL

Store (P)

Mnemonic Code = SRP

Octal OP Code = 100

PURPOSE: To obtain the contents of the P-register from the Instruction Expansion Module and store it in main memory.

FORMAT:

OC	OP1	OP2
100	AR/I	AR/I

DMA CHANNEL CODE: 002

OPERATION: The input data received from the Instruction Expansion Module is received via the DMA channel specified in OP1.

OP2 item is to receive the input data.

The input data consists of a 2-byte address. This address is obtained and saved by the Instruction Expansion Module as a result of executing the Save P instruction.

EXAMPLE:

OC	OP1	OP2
100	241	101

OP1 Item 41 of Active Record 2 contains the DMA channel number.

OP2 Item 1 of Active Record 1 is to receive the input data.

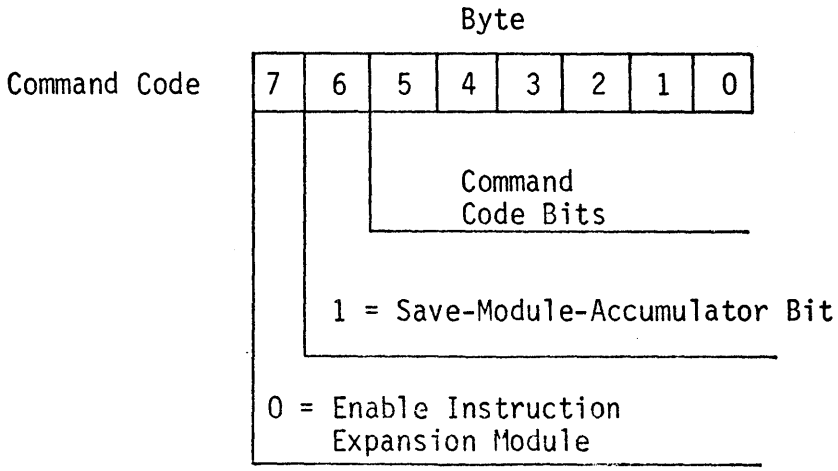
INSTRUCTION EXPANSION MODULE B

GENERAL

Instruction Expansion Module B (Utility Adapter)[1] provides the programmer with a logical set of instructions: exclusive OR, inclusive OR, and AND functions. The logical functions may be performed on two data characters or on a string of data characters (as is the case when computing an LRC character for a string of data characters). The 1-byte result of a logical operation resides in the module accumulator.

Instruction Expansion Module B is able to transfer the data in its 1-byte accumulator to the SYSTEM 2400 Processor via the Store Module Accumulator instruction.

The command codes used in the instructions to direct the Instruction Expansion Module to perform a specific function contain two modifier bits, as shown below:



Save Module Accumulator Bit (2^6) - Informs the Instruction Expansion Module to save the contents of the accumulator. This feature permits logical operations on strings of data characters in excess of 255 bytes (the maximum number of data bytes transferred with a single instruction is 256 with the first byte being the command code) or on a group of single bytes or strings of bytes located in different part of memory. For example, an LRC operation on a string of data characters greater than 255 bytes would require that bit 2^6 be set to a "1" in all subsequent instructions conveying data during this operation. In addition, the contents of the module accumulator may be stored in main memory using the Store Module Accumulator instruction and may be returned to the accumulator in the Instruction Expansion Module using the

[1] Appendix H gives an in-depth coverage for users implementing this expansion.

Enter Module Accumulator instruction, thus allowing more than one subroutine in the main program to utilize the features of the Instruction Expansion Module.

Enable Instruction Expansion Module Bit (2^7) - Bit 2^7 set to a "0" in a command code enables the Instruction Expansion Module and informs the other connected peripherals to deselect it.

The module B instructions include the following:

- OR (Exclusive) 1-153)
- Logical AND (1-155)
- OR (Inclusive) (1-157)
- Longitudinal Redundancy Check (1-159)
- Enter Module Accumulator (1-161)
- Store Module Accumulator (1-162)

LOGICAL SET

OR (Exclusive)

Mnemonic Code = ORE

Octal Op Code = 105

PURPOSE: To logically OR (Exclusive) two or more data characters.

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

DMA CHANNEL CODE: 001

COMMAND CODE: 001

With save-module-accumulator-modifier bit: 101

OPERATION: The output data from the OP2 item is sent to the Instruction Expansion Module connected to the DMA channel specified in OP1.

OP2 is a multi-byte item containing the command code as the leftmost byte in the item, followed by data characters (255 bytes maximum).

The Instruction Expansion Module, upon receipt of the first byte (command code) of the OP2 item, resets its logic and prepares for an exclusive OR operation. The first data character (byte 2 of OP2) is stored in the accumulator. The next data character received is OR'ed with the contents of the accumulator, with the result residing in the accumulator. This procedure is repeated for all data characters in the OP2 item. The result of the exclusive OR operation is obtained and stored in main memory using the Store Module Accumulator instruction.

Exclusive OR Operation:

```
  1 1 0 0
  1 0 1 0
  -------
  0 1 1 0
```

RESTRICTION: A maximum of 255 data characters may be OR'ed with a single instruction. When operating on data strings greater than 255 data characters, the save-module-accumulator-modifier bit of the command code must be set to a "1" in subsequent instructions to prevent the Instruction Expansion Module from resetting its accumulator, thereby destroying its contents.

EXAMPLE: Two-byte OR (Exclusive) operation.

OC	OP1	OP2
105	241	302

OP1 | 001 | Item 41 of active record 2

OP2

b_1	b_2	b_3
001	145	114

Item 2 of active record 3

b_1 = Command code (exclusive OR)
b_2 = 01100101
b_3 = 01001100
Result = 00101001

LOGICAL SET

Logical AND

Mnemonic Code = AND

Octal Op Code = 105

PURPOSE: To logically AND two or more data characters.

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

DMA CHANNEL CODE: 001

COMMAND CODE: 002

With save-module-accumulator-modifier bit: 102

OPERATION: The output data from the OP2 item is sent to the Instruction Expansion Module connected to the DMA channel specified in OP1.

OP2 is a multi-byte item containing the command code as the leftmost byte in the item, followed by data characters (255 bytes maximum).

The Instruction Expansion Module, upon receipt of the first byte (command code) of the OP2 item, resets its logic and prepares for a logical AND operation. The first data character (byte 2 of OP2) is stored in the accumulator. The next data character received is AND'ed with the contents of the accumulator, with the result residing in the accumulator. This procedure is repeated for all data characters in the OP2 item. The result of the logical AND operation is obtained and stored in main memory using the Store Module Accumulator instruction.

Logical AND operation:

```
  1 1 0 0
  1 0 1 0
  -------
  1 0 0 0
```

RESTRICTION: A maximum of 255 data characters may be AND'ed with a single instruction. When operating on data strings greater than 255 data characters, the save-module-accumulator-modifier bit of the command code must be set to a "1" in subsequent instructions to prevent the Instruction

Expansion Module from resetting its accumulator, thereby destroying its contents.

EXAMPLE: Two-byte logical AND operation.

OC	OP1	OP2
105	241	304

OP1 | 001 | Item 41 of active record 2

OP2

b_1	b_2	b_3	
002	145	114	Item 4 of active record 3

b_1 = Command code (logical AND)
b_2 = 01100101
b_3 = 01001100
Result = 01000100

LOGICAL SET

OR (Inclusive)

Mnemonic Code = ORI

Octal Op Code = 105

PURPOSE:

PURPOSE: To logically OR (Inclusive) two or more data characters.

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

DMA CHANNEL CODE: 001

COMMAND CODE: 004

With save-module-accumulator-modifier bit: 104

OPERATION: The output data from the OP2 item is sent to the Instruction Expansion Module connected to the DMA channel specified in OP1.

OP2 is a multi-byte item containing the command code as the leftmost byte, followed by data characters (255 bytes maximum).

The Instruction Expansion Module, upon receipt of the first byte (command code) of the OP2 item, resets its logic and prepares for an inclusive OR operation. The first data character (Byte 2 of OP2) is stored in the module accumulator. The next data character received is OR'ed with the contents of the accumulator, with the result residing in the accumulator. This procedure is repeated for all data characters in the OP2 item. The result of the inclusive OR operation is obtained and stored in main memory using the Save Module Accumulator instruction.

For a complete explanation, see PROCESSOR PROGRAMMING IN MACHINE CODE, Form No. M-2269.

Inclusive OR Operation:

```
  1 1 0 0
  1 0 1 0
  -------
  1 1 1 0
```

RESTRICTION: A maximum of 255 data characters may be OR'ed with a single instruction. When operating on data strings greater than 255 data characters,

the save-module-accumulator-modifier bit of the command code must be set to a "1" in subsequent instructions to prevent the Instruction Expansion Module from resetting its accumulator, thereby destroying its contents.

EXAMPLE: Two-byte OR (inclusive) operation.

OC	OP1	OP2
105	241	303

OP1 | 001 | Item 41 of active record 2

OP2

b_1	b_2	b_3	
004	145	114	Item 3 of active record 3

b_1 = Command code (inclusive OR)
b_2 = 01100101
b_3 = 01001100

Result = 01101101

LOGICAL SET

Longitudinal Redundancy Check

Mnemonic Code = LRC

Octal Op Code = 105

PURPOSE: To generate a longitudinal parity number on a string of data characters (exclusive OR).

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

DMA CHANNEL CODE: 001

COMMAND CODE: 001

With save-module-accumulator-modifier bit: 101

OPERATION: The output data from the OP2 item is sent to the Instruction Expansion Module connected to the DMA channel specified in OP1.

OP2 is a multi-byte item containing the command code as the leftmost byte in the item, followed by data characters (255 bytes maximum).

The Instruction Expansion Module, upon receipt of the first byte (command code) of the OP2 item, resets its logic and prepares for an LRC (exclusive OR) operation. The first data character (byte 2 of OP2) is stored in the module accumulator. The next data character received is OR'ed with the contents of the accumulator, with the result residing in the accumulator. This procedure is repeated for all data characters in the OP2 item. The result of the LRC operation is obtained and stored in main memory using the Store Module Accumulator instruction.

RESTRICTION: A maximum of 255 data characters may be operated upon using a single LRC instruction. When performing an LRC operation on data strings larger than 255 data characters, the save-module-accumulator-modifier bit of the command code must be set to a "1" in subsequent instructions to prevent the Instruction Expansion Module from resetting its accumulator, thereby destroying its contents.

EXAMPLE: Multi-byte LRC operation.

OC	OP1	OP2
105	241	101

OP1 | 001 | Item 41 of active record 2

OP2

b_1	b_2	b_3	b_4
001	145	114	165

b_1 = Command code (LRC)
b_2 = 01100101
b_3 = 01001100
Result = 00101001
b_4 = xxxxxxxx
New Result = xxxxxxxx
b_5 =
.
. etc.
.
. 10011001
b_n = 01110101
Total Result = 11101100
In Accumulator

ENTER/STORE

Enter Module Accumulator

Mnemonic Code = EMA

Octal Op Code = 105

PURPOSE: To enter the accumulator of Instruction Expansion Module B with a 1-byte number.

FORMAT:

OC	OP1	OP2
105	AR/I	AR/I

DMA CHANNEL CODE: 001

COMMAND CODE: 050

OPERATION: The output data from the OP2 item is sent to the Instruction Expansion Module connected to the DMA channel specified in OP1.

OP2 is always a 3-byte item formatted as follows:

	b_1	b_2	b_3
OP2	001	XXX	XXX

b_1 = Command Code (EMA)

b_2 = The 1-byte value to be re-entered into the accumulator.

b_3 = A dummy byte (any value) to position b_2 within the accumulator.

The primary function of this instruction is to re-enter the result of a logical operation which was terminated prior to completion (usually due to other requirements imposed on the Instruction Expansion Module by the program).

RESTRICTION: After the current result of a logical operation has been re-entered in the module accumulator, the operation may be continued. Since the accumulator now contains a value to be operated upon, all subsequent instructions conveying data to the Instruction Expansion Module must have the save-module-accumulator-modifier bit (2^6) set in the command code to avoid clearing of the accumulator.

ENTER/STORE

Store Module Accumulator

Mnemonic Code = SMA

Octal Op Code = 100

PURPOSE: To obtain and store the contents of the Instruction Expansion Module accumulator in main memory.

FORMAT:

OC	OP1	OP2
100	AR/I	AR/I

DMA CHANNEL CODE: 001

OPERATION: The input data from the Instruction Expansion Module is received via the DMA channel specified in OP1.

OP2 item is to receive the input data.

The input data from the module accumulator of the Instruction Expansion Module is a 1-byte item for all logical set instructions.

SECTION II
SYSTEM 2400 ASSEMBLER LANGUAGE

INTRODUCTION

The Assembler language is a symbolic programming language that includes:

- Basic Instructions that provide mnemonics that correspond to machine-language operation codes, and
- Assembler Directives that direct the assembler to perform certain tasks.

In machine-language instructions, octal numbers specify op codes and operands. In the Assembler language, mnemonics specify the operation codes and symbolic names to specify the records, items, buffers, addresses, and literals.

Below is an example written in Assembler language and in machine language:

```
Assembler language MR      ACCNT1,ACCNT2      REPLACE ACCNT NBR
Machine language 001110204
```

In this example, the mnemonic "MR" should bring to mind the Move Right-Aligned, No Fill instruction. Data is moved from "ACCNT1" to "ACCNT2". The comment following the instruction states the purpose of this instruction in the program.

In the machine-language instruction, however, one must consult a table to tell that "001" is the Move Right-Aligned, No Fill instruction. Further, one could not readily discern that item 10 of active record 1 is being moved to item 4 of active record 2. The intent or purpose of the instruction is not clear.

The Assembler language is used as follows:

- To easily define EBCDIC, USASCII, tri-octal, and address constants.
- To assign symbolic names to values which may be changed prior to assembly.
- To reserve unused areas within memory for reference during program execution. (An earlier program may place data into such an area, for example.)
- To provide a self-documented program listing.
- To facilitate program and subroutine linkages (MDL and RTN macro instructions), and

- To reassign a program to any part of memory (relocatability).

Once the program has been written in Assembler language, the program must be converted to equivalent machine coding before it can be executed. This conversion (assembly) process is done by the Assembler program.

The Assembler performs three basic functions:

- Converts the Assembler instructions to their machine-language equivalents.
- Prints a listing of the Assembler instructions with their machine-language equivalents and flags any syntax errors found.
- Writes the machine-language coding on a magnetic tape for "collection" or immediate program loading and execution.

CODING CONVENTIONS

Assembler instructions are coded in 80-character records for punched card compatibility. Each instruction must be coded as follows:

- Columns 1-9 contain the label field.
- Columns 10-15 contain the op-code field.
- Columns 16-35 contain the operand field.
- Columns 36-71 contain the comment field.
- Column 72 is reserved for editing.
- Columns 73 to 80 are not checked, but may contain sequence numbers or comments.

A *label*, if present, *must* start in column 1. At least one space must separate the label (or start-of-card) and op-code fields, the op-code and operand fields, and the operand and comments fields.

NOTE

If an asterisk (*) is coded in column 1, the record is considered a comment and is therefore not translated into machine code. A comment is simply printed on the Assembler listing.

SYMBOLIC NAMES

The Assembler uses symbolic names for:

- Mnemonic op codes;
- Locations (addresses) of instructions;
- Locations of data;
- Records;
- Items within records; and
- Buffers

Symbolic names must follow these rules:

1. A name must contain from 2 to 6 characters.
2. The characters of a name must be the letters of the alphabet (A-Z) or the digits (0-9), in any combination.
3. The first character of a name must be alphabetic.
4. The names SPACE, NULL, HICORE, NXCORE, SDAPE, PBIASE are reserved for special use. (SPACE is octal 100; NULL is octal 000; for others, see "Relocatability", page

Examples:

A123BC - Valid
PAYIN - Valid
5OUT - Invalid; first character must be alphabetic
C - Invalid; must contain from 2 to 6 characters
INP$$ - Invalid; characters must be alphabetic or digits

Programmers should *not* utilize the lower area of core memory in Assembler language programs. Core locations below 100 are currently assigned or reserved for future use as follows:

LIST OF RESERVED MEMORY LOCATIONS

Address (octal)	Content
000,001	SDAT pointer
002,003	I/O function table address
004,005	Address of data record, ACTIVE RECORD 1
006,007	Address of item descriptor table, ACTIVE RECORD 1
010,011	Address of data record, ACTIVE RECORD 2

012,013	Address of item descriptor table, ACTIVE RECORD 2
014-015	Address of data record, ACTIVE RECORD 3
016-017	Address of item descriptor table, ACTIVE RECORD 3
020-021	P-BIAS (program start/restart address)
022-023	Real-time clock
024-037	Interrupt
040-057	Interrupt state SDAT pointer and active registers
060-077	Reserved for future use

BASIC INSTRUCTIONS

For each Peripheral Processor machine instruction there is a corresponding Assembler instruction. Each instruction has from *zero* to *four* operands within it, and each instruction may optionally have a label attached to it. The label must start in column 1.

In Table 2-1 below, the basic instructions are grouped according to number and type of operands. Operands are separated by commas with no intervening spaces. If a 2-byte operand is coded where a 1-byte operand should be, the Assembler flags the statement with an "I" (invalid) and uses the rightmost byte only. If the wrong number of operands is coded, the Assembler flags the statement with a "W" (warning) and uses nulls (000) in place of missing operands.

Table 2-1. Basic Instructions

INSTRUCTION TYPE		MNEMONIC		INSTRUCTION NAME
Type I	Instructions With No Operand (One Byte Total)	H	143	Halt
		GAP	147	No Operation - Leave Gap
		SWS	154	Swap States
		SIL	155	Set Interrupt Lockout
		CIL	157	Clear Interrupt Lockout
Type II	Instructions With One 1-Byte Operand (Two Bytes Total)	TBS	040	Test Binary Sign
		TDS	042	Test Decimal Sign
		STD	124	Store Designators
		LD	126	Load Designators
		STT	134	Store Tally Counter
		LT	136	Load Tally Counter
		SDI	146	Set Display Indicators

Table 2-1. Basic Instructions (Continued)

INSTRUCTION TYPE		MNEMONIC		INSTRUCTION NAME
		CDI	156	Clear Display Indicators
		LSP	161	Load Storage Descriptor Pointer
		LR1	165	Load Active Record 1
		LR2	171	Load Active Record 2
		GRT	173	GOTO Return (Branch)
		IM	174	Interrupt Mask
		LR3	175	Load Active Record 3
		GIR	177	Interrupt Branch GOTO
Type III	Instructions With One 2-Byte Operand (Three Bytes Total)	NOP	020	No Operation
		GGT	021	GOTO Greater Than
		GLT	022	GOTO Less Than
		GNE	023	GOTO Not Equal
		GE	024	GOTO Equal
		GNL	025	GOTO Not Less Than
		GNG	026	GOTO Not Greater Than
		G	027	GOTO Unconditionally
Type IV	Instructions With One 1-Byte Operand and One 2-Byte Operand (Four Bytes Total)	GD	030	GOTO On Designators
		GS	031	GOTO On Switches
		GBG	061	GOTO Binary Greater Than
		GBL	062	GOTO Binary Less Than
		GBN	063	GOTO Binary Non-Zero
		GBZ	064	GOTO Binary Zero
		GGBE	065	GOTO Binary _ Zero
		GLBE	066	GOTO Binary _ Zero
		GDG	071	GOTO Decimal Greater Than
		GDL	072	GOTO Decimal Less Than
		GDN	073	GOTO Decimal Non-Zero
		GDZ	074	GOTO Decimal Zero
		GGDE	075	GOTO Decimal _ Zero
		GLDE	076	GOTO Decimal _ Zero
		GA	107	GOTO On Active Channel
		GSI	113	GOTO On Service Request
		GCI	117	GOTO On Channel Interrupt
		GCT	170	GOTO On Count
		GSB	176	GOTO Subroutine (Branch)

Table 2-1. Basic Instructions (Continued)

INSTRUCTION TYPE		MNEMONIC		INSTRUCTION NAME
Type V	Instructions With Two 1-Byte Operands (Three Bytes Total)	M	000	Move Item, Left-Align, No Fill
		RN	000	Rename
		MR	001	Move Item, Right-Align, No Fill
		CB	044	Compare Binary
		CD	046	Compare Decimal
		MPK	050	Move, Pack
		MUP	052	Move, Unpack
		INS	100	Special In
		OTS	105	Special Out
		STC	110	Store Channel Control Register
		STR	111	Store Channel Reverse
		INR	112	Initiate Input Reverse
		IN	114	Initiate Input On Channel
		OUT	115	Initiate Output On Channel
		OTR	116	Initiate Output Reverse
		APA	121	Append, Advance
		EXP	131	Extract Previous Item
		EX	132	Extract Item
		EXA	133	Extract Item, Advance
		ML	141	Move Literal
		CAN	142	Compare Alphanumerics
		CL	144	Compare Literal
		TI	150	Test Item
		TL	151	Test Literal
		TM	152	Test Mask
		TIM	153	Test Item Mask
		TCK	162	Longitudinal Redundancy Check
Type VI	Instructions With Three 1-Byte Operands (Four Bytes Total)	MF	004	Move Item, Left-Align, Fill
		MRF	005	Move Item, Right-Align, Fill
		MJ	006	Move Item, Left-Justify, Fill
		MRJ	007	Move Item, Right-Justify, Fill
		AB	041	Add Binary
		A	043	Add Decimal
		SB	045	Subtract Binary

Table 2-1. Basic Instructions (Continued)

INSTRUCTION TYPE		MNEMONIC		INSTRUCTION NAME
		S	047	Subtract Decimal
		ALB	051	Add Literal Binary
		AL	053	Add Literal Decimal
		SLB	055	Subtract Literal Binary
		SL	057	Subtract Literal Decimal
		EF	104	External Function On Channel
		EFS	106	External Function Special
		APR	120	Append, Right-Eliminate
		APE	122	Append, Left-Eliminate
		TRL	140	Translate Code
		X	160	OR (Exclusive)
		O	164	OR (Inclusive)
		N	166	Logical AND
		GTB	172	GOTO Table (Indirect Branch)
Type VI	Instructions With Four 1-Byte Operands (Five Bytes Total)	CP	014	Compress Item, Left-Align, Fill
		CPR	015	Compress Item, Right-Align, Fill
		EXV	130	Extract Variable Length Item, Fill
EXPANSION MODULE INSTRUCTIONS				
Type VIII	Instructions with Two 1-Byte Operands (Three Bytes Total)	SRP	100	Store (P)
		SMA	100	Store Module Accumulator
		SAP	105	Save (P)
		SVP	105	Store (P)
		ORE	105	OR (Exclusive)
		AND	105	Logical AND
		ORI	105	OR (Inclusive)
		LRC	105	Longitudinal Redundancy Check
		EMA	105	Enter Module Accumulator

Table 2-1. Basic Instructions (Continued)

INSTRUCTION TYPE		MNEMONIC		INSTRUCTION NAME
		EXTERNAL EXECUTE INSTRUCTIONS		
Type IX	Instructions With One Sub-Op-Code And One 1-Byte Operand (Three Bytes Total)	LC	145 004	Load Delta Clock
	Instructions With One Sub-Op-Code And Three 1-Byte Operands (Five Bytes Total)	SEE	145 014	Store External Instruction Error
		SCE	145 015	Store Channel Parity Error
		MB	145 020	Multiply Binary
		DB	145 022	Divide Binary
		MD	145 024	Multiply Decimal
		DD	145 026	Divide Decimal
		BTD	145 030	Binary to Decimal
		DTB	145 031	Decimal to Binary
		SDR	145 034	Store Decimal Remainder
		SBR	145 035	Store Binary Remainder
	Instructions With One Sub-Op-Code And Four 1-Byte Operands (Six Bytes Total)	MLB	145 021	Multiply Literal Binary
		DLB	145 023	Divide Literal Binary
		MLD	145 025	Multiply Literal Decimal
		DLD	145 027	Divide Literal Decimal

Operands in basic instructions may be in the following formats:

A. A tri-octal character string at least two digits long between 000 and 377 for 1-byte operands, or between 000-000 and 377-377 for 2-byte operands. As shown, a minus sign may be used to separate the digits of each byte.

B. A *label* format, with the value limitations as in A above.

C. A *label + nnnnn* format, where "nnnnn" is a decimal value between 0 and 99999, with the value limitations as in A above.

D. A *label - nnnnn* format, where "nnnnn" is a decimal value between 0 and 99999, with the value limitations as in A above.

E. An *n/label* format, where "n" is a decimal value indicating an active record (see below), and Label is a tri-octal value between 0 and 77.

0 = SDAT Pointer

1 = Active Record 1.

2 = Active Record 2.

3 = Active Record 3.

F. An *n/ii* format, where "n" indicates an active record (as in E above), and "ii" is a 1- or 2-byte octal number between 0 and 77. (For 1-byte operands only.)

G. A single character (except space, +, *, or comma) may be used as a *literal* (1-byte operand). The Assembler will translate the character to its corresponding EBCDIC code.

NOTE

The *first* operand cannot be in this format.

H. A defined character (EBCDIC, USASCII, Decimal or Octal) may be used as a literal (1-byte operand) in a valid instruction operand field or as a constant (1-byte operand) in directives which require operands. The format of the operand is as follows:

U'u' where U indicates that the character between the apostrophes is an USASCII character and is translated into its USASCII code equivalent.

u = USASCII character.

C'c' where C indicates that the character between the apostrophes is an EBCDIC character and is translated into its EBCDIC code equivalent.

c = EBCDIC character.

D'ddd' where D indicates that the number between the apostrophes is a decimal number and is translated into its binary equivalent.

ddd = decimal number (0-255)

O'ooo' where O indicates that the number between the apostrophes is an octal number and is translated into its binary equivalent.

ooo = octal number (0-377)

Example 1: TL FLAG,C'A'

Example 2: RES 0'2ØØ'

I. Labels may be chained up to a maximum of 28 characters in the operand field. The plus (+) and minus (-) signs are valid arithmetic operators within the chain.

NOTE

If a constant is used, it must be located at the end of the chain. Only one constant may be used in a chain.

Example: G GET+GOT-PUT+O'100'

The operand in the example contains 18 characters.

A single asterisk (*) may be used in place of a label in formats B, C, and D. The asterisk is equal to the address of the op-code of the instruction. The value so generated is a 2-byte operand.

An asterisk used in a GO instruction is equal to the op-code address minus the PBIAS.

The permissible Assembler operands and operand formats are summarized in Table 2-2.

LINKAGE MACROS

The Assembler has two linkage macros available which can simulate the "catch-P" function (executing an out-of-line subroutine) or pass parameter addresses between subroutines. Each may optionally have a label.

Move Double Literal (MDL) Macro Statement

Mnemonic: MDL

The MDL statement has two operands:

1. A 1-byte "active-record/item-number" (Formats A, B, C, D, E, F, G, H in Table 2-2).
2. A 2-byte address (Formats A, B, C, D in Table 2-2).

Table 2-2. Summary of Operands

OPERAND FORMATS

A. (i). Self-defining tri-octal: 0 to 377.
   (ii). Self-defining tri-octal: 0 to 377. (Dashes allowed)

B. Label (e.g. "ABC123", or "*").

C. Label+nnnnn (where "nnnnn" is decimal, 0 to 99999).

D. Label-nnnnn (where "nnnnn" is decimal, 0 to 99999).

E. n/LABEL (where "n" = 0, 1, 2, or 3, and "LABEL" = 0 to 77 tri-octal).

F. n/ii (where "n" = 0, 1, 2, or 3, and "ii" = 0 to 77 tri-octal).

G. x (where "x" is a single character except "+", "*", space, or ",").

H. A defined character: EBCDIC, USASCII, Decimal, or Octal.

I. Labels may be joined in an algebraic expression with plus (+) and minus (-) signs. Only one constant may -e used and it must be the last term in the expression.

PERMISSIBLE OPERANDS

a. For Type II basic instructions (see Table 2-1).
   Operand Types: A(i), B, C, D, E, F (Values 0 to 377)

b. For Type III basic instructions (see Table 2-1).
   Operand Types: A(ii), B, C, D

c. For Type IV basic instructions (see Table 2-1).
   First Operand: A(i), B
   Second Operand: A(ii), B, C, D

d. For Type V and VI basic instructions (see Table 2-1).
   First Operand: A(i), B, C, D, E, F
   Subsequent Operands: A(ii), B, C, D, E, F, G

e. For MDL and RTN Linkage Macros.
   First Operand: A(i), B, C, D, E, F
   Second Operand: A(ii), B, C, D

f. For constants in 'DC' when defining addresses (Y or A) and in 'SD'.
   A(ii), B, C, D

g. For BUFF and RES Assembler Directives. B, C, D, H

h. For END and LOAD Assembler Directives. A(ii), B, C, D

i. For ENTRY and EXTRN Assembler Directive. B

J. For EQU Assembler Directive. A(i), A(ii), B, C, D, E, F, G

k. For constants in 'DCF' Assembler Directive. A(i). H, I.

The 2-byte address is moved into the item defined by the first operand. The item is assumed to be two bytes long. The MDL statement is effected by the "141" and "051" (or "055") machine instructions.

Both operands may be external labels.

Example:

To move the address of "DATA" to the 2-byte field "PARM", code the following statement:

```
        MDL     PARM,DATA
```

Return (RTN) Macro Statement

Mnemonic: RTN

The RTN statement has the same format as the MDL statement.

The P-BIAS is subtracted from the 2-byte address and the result is moved into the item defined by the first operand. The item is assumed to be two bytes long.

Typically, the first operand of an RTN statement describes the 2-byte operand of a "G" instruction. Linkage to a routine called "MULT" may be accomplished by the following statements:

```
        RTN     MUEXIT,RETURN
        G       MULT
RETURN  ALB     COUNT,COUNT,001
```

The routine "MULT" exits by executing the "G" instruction whose operand is defined by "MUEXIT". Control is returned to the original routine at the instruction labeled "RETURN".

Both operands may be external labels.

DEFINITION OF CONSTANTS

Data used by the program may be defined by the DC (Define Constant) statement. Seven types of operands may be associated with the DC statement. Only one operand may appear in each DC statement. Optionally, the DC statement may be labeled.

Definition of Tri-Octal Constant

First byte of operand must be the letter "O".

Optionally, a length may be specified as 'Lnn', where 'nn' is a decimal number from 1 to 99. If *no* length is specified, the length is calculated from the constant specified.

The constant itself is specified as (tri-octal) digits enclosed in single quotes. For readability, minus signs may be interspersed with the digits. The constant specified is right-aligned in the constant area. Extra digits in the area are made to be nulls.

Examples:

DC OL3'010-224'

Generates a 3-byte constant with the tri-octal value '000-010-224'.

DC O'112-253-364-111'

Generates a 4-byte constant with the tri-octal value '112-253-364-111'.

NOTE

If *no* constant is specified, the whole constant area is filled with nulls.

Definition of EBCDIC Character Constant

First byte of operand must be the letter "C".

Optionally, a length may be specified, as above for tri-octal constant.

The constant itself is specified as (EBCDIC) characters enclosed in single quotes. The constant specified is left-aligned in the constant area. Extra bytes in the area are assumed to be spaces.

Examples:

DC CL5'PAGE'

Generates a 5-byte constant with the tri-octal value '327-301-307-305-100' ("P-A-G-E-space").

DC C'HDG'

Generates a 3-byte constant with the tri-octal value '310-304-307' ("H-D-G").

NOTE

If no constant is specified, the whole area is filled with EBCDIC spaces.

Definition of USASCII Character Constant

First byte of operand must be the letter "U".

Optionally, a length may be specified as above for tri-octal constant.

The constant itself is specified as characters enclosed in single quotes. Each character is translated to its USASCII equivalent and left-aligned in the constant area. Extra bytes in the area are assumed to be USASCII spaces.

Examples:

DC UL5'PAGE'

Generates a 5-byte constant with the tri-octal value '020-001-007-005-040' (USASCII "P-A-G-E-space").

DC U'HDG'

Generates a 3-byte constant with the tri-octal value '010-004-007' (USASCII "H-D-G").

NOTE

If *no* constant is specified, the whole area is filled with USASCII spaces.

Definition of Binary Constant

First byte of operand must be the letter "D".

Optionally, a length may be specified as above for tri-octal constant.

The constant itself is specified as (decimal) characters enclosed in single quotes. The constant ("nnn") is a decimal number from 0 to 255. The constant specified is right-aligned in the constant area. Extra bytes in the area are made to be nulls.

Example:

DC DL 1'128'

Generates a 1-byte constant with the tri-octal value '200' (128_{10}).

DC DL6'9'

Generates a 6-byte constant with the tri-octal value '000-000-000-000-000-011' (9_{10}).

DC D'255'

Generates a 1-byte constant with the tri-octal value '377' (255_{10}).

Definition of Address Constant

First byte of operand must be the letter "A" or "Y".

Optionally, a length may be specified, as follows:

"L2" indicates one address constant is to be generated.
"L4" indicates two address constants are to be generated.

One or two address constants follow, each in one of the A, B, C, or D formats (see Table 2-2), enclosed by single quotes or brackets. If two constants are specified, a comma separates them.

Examples:

DC A(AXØ4+3)

Generates one 2-byte address constant pointing to the label "AXØ4" plus three bytes.

DC YL4(HERE,THERE)

Generates two 2-byte address constants, the first constant being an address pointing to the label "HERE", the second being an address pointing to the label "THERE".

Definition of Biased Address Constant

The first byte of the operand must be the letter "B" or "J".

Optionally, a length may be specified, as follows:

"L2" indicates one biased address constant is to be generated.

"L4" indicates two biased address constants are to be generated.

One or two address constants follow, each in one of the A, B, C, or D formats (see Table 2-2), enclosed by single quotes or brackets. If two constants are specified, a comma separates them.

"Biased" addresses are identical to ordinary addresses, except that the program P-BIAS has been subtracted from them. A common use for this type of address is as entries in a "jump-table".

Examples:

DC B(AXØ4+3)

Generates one 2-byte constant of an address pointing to the label "AXØ4", plus three bytes, relative to the P-BIAS address.

DC BL4(HERE,THERE)

Generates two 2-byte constants, the first constant being an address pointing to the label "HERE", the second being an address pointing to the label "THERE", both relative to the P-BIAS address.

Definition of Constant Fill

Fill space in memory.

A label may be used optionally.

The Define Constant Fill (DCF) directive allows the programmer to fill a designated number of memory locations with a specific constant (character or number). Four types of operands may be associated with the DCF statements:

Operand	Constant Type
DLnn'ddd'	Decimal
CLnn'C'	EBCDIC
OLnn'OOO'	Octal
ULnn'u'	USASCII

where: nn = specifies a constant length from 1-99.
ddd = specifies a decimal number, 0 to 255.
OOO = specifies a octal number, 0 to 377.
u = specifies a USASCII character.

Examples:

```
DCF     DL80'100 - fills the next 80 locations with the octal value 144.
DCF     CL2'A'   - fills two locations with the octal value 301.
DCF     OL3'123' - fills three locations with the octal value 123.
DCF     UL10'B'  - fills ten locations with the octal value 102.
```

ASSEMBLER DIRECTIVES

The Assembler directives provide additional information during assembly of the program. Note that if each directive requires an operand, the operand must be self-defining (i.e., numeric) or previously defined (if a label). When this rule is broken, the statement is printed during phase 1 of assembly with a "U" flag to denote that the operand is undefined.

The Assembler directives include the following:

- BUFF - Define buffer for logical IOCS
- DBL - Double-space listing
- EJECT - Start a new page in listing
- END - End assembly
- ENTRY - Specify entry point
- EQU or = - Equate or define label
- EXTRN - Specify external label
- LOAD - Set value of current address pointer
- OBJ - Generate object code
- NOBJ - Generate *no* object code
- PUNCH - Punch operand into object code
- REPRO - Reproduce next statement into object code
- RES - Reserve space in memory
- SDP - Define SDAT pointer
- SGL - Single-space listing
- SPACE - Leave a space in listing
- START - Start assembly
- TITLE - Specify a title for the assembly
- NOGEN - Inhibit second and following print lines
- GEN - Cancel previous NOGEN directive
- COM - Specify entry point for AR/I information
- INC - Include operand (library subroutine)
- LIST - Generate listing output

- NLIST - Generate no listing output
- PAGE - Set address pointer to start of next page

BUFF Directive

Define buffer for logical IOCS.

A label is not allowed with this directive.

One operand is required.

Example:

"BUFF 2ØØ"

Reserve 2ØØ bytes for IOCS buffer.

NOTE

If the operand is self-defining, it must be a *decimal* number.

This statement should appear only *once* in an assembly. Its operand is converted to tri-octal and placed in the object output 'UCORE' record. (The Collector reserves this space and provides the necessary linkage parameter to IOCS.) If this directive is omitted, 300 bytes are provided for IOCS. This directive *may* appear before the START directive. The BUFF statement is meaningless in non-relocatable assemblies.

DBL Directive

Double-space listing.

A label or operand is not allowed with this directive.

This statement starts a new page on the assembly listing. Subsequent statements are double-spaced. This directive *may* appear before the START statement.

NOTE

When DBL has not been specified, statements are single-spaced on the listing. The DBL statement itself does *not* appear on the listing.

EJECT Directive

New page on listing.

A label or operand is not allowed with this directive.

This statement starts a new page on the assembly listing. This directive *may* appear before the START statement.

NOTE

The EJECT statement itself does *not* appear on the listing.

END Directive

End assembly.

A label is not allowed with this directive.

One operand is required: the address of the first instruction to be executed in the program. (This operand may be a tri-octal, self-defining term, or a label.)

This operand is placed into memory locations 20 and 21 and used as the program P-BIAS. (The P-BIAS is also placed in the 'UCORE' record by the Assembler.)

ENTRY Directive

Specify entry point.

A label is not allowed with this directive.

One operand is required: it *must* be a label, elsewhere defined.

Each ENTRY statement *must follow* the START statement and *must precede* all basic instructions, DC statements, and EXTRN statement. This statement writes a 'ZZZZENTRY' record in the object output, enabling the specified *label* to be referenced in another assembly as a 2-byte address value. Also, this statement may be used *only* in relocatable programs (see "Relocatability", page

EQU (or =) Directive

Equate or define label.

A label *must* be used with this directive.

One operand is required: it may be in any of the formats A, B, C, D, E, F, or G (see Table 2-2).

The label is assigned the value of the operand. The EQU directive *may* appear before the START statement.

EXTRN Directive

Specify external label.

A label is not allowed with this directive.

One operand is required: it *must* be a *label*.

The EXTRN statement *may* appear anywhere between the START and END statements, but *must* appear after any ENTRY statements. This statement enables the programmer to use the specified label elsewhere in the program as if it were an already-Defined address (2-byte value) or AR/I information (1-byte value). The Collector later determines the value of the label and inserts that value in the object coding where that label is used. This statement may be used only in relocatable programs (see "Relocatability", page 2-31.

LOAD Directive

Set value of current address pointer.

A label is not allowed with this directive.

One operand is required: it may be in any of the formats A, B, C, or D (see Table 2-2).

The LOAD statement resets the current address pointer to the value specified by the operand. The next statement starts at that address. This directive is *ignored* if it occurs before the START statement of the program.

OBJ Directive

Generate object code. (This directive is needed, only if "NOBJ" was specified earlier.)

A label or operand is not allowed with this directive.

The OBJ statement writes object coding on tape (unless suppressed by Sense switches, see "Operating the Assembler" on page 2-34.

NOBJ Directive

Generate *no* object code.

A label or operand is not allowed with this directive.

The NOBJ statement causes object coding to cease being written on tape.

NOTE

When NOBJ has *not* been specified, object coding is written on tape (unless suppressed by Sense switches, see "Operating the Assembler" on page 2-34

PUNCH Directive

Punch operand into object code.

A label is not allowed with this directive.

One operand is required: any character string, enclosed in single quotes.

The operand is left-aligned, padded with spaces, and placed on the object tape as an 80-byte record (unless suppressed by Sense switches, see "Operating the Assembler" on page 2-34 . This directive *may* appear before the START statement.

Example:

```
PUNCH     'ZZZZPROGA'
```

REPRO Directive

Reproduce next statement into object code.

A label or operand is not allowed with this directive.

This statement places columns 1 through 74 of the next statement as an 80-byte record on the object tape (unless suppressed by Sense switches, see "Operating the Assembler" on page 2-34 . The next statement is *not* processed by the Assembler as an instruction. Bytes 75 to 80 of the object record are a check-sum and sequence number, assigned by the Assembler.

RES Directive

Reserve space in memory.

A label may be used optionally.

One operand is required: the length of the field to be reserved at this address.

NOTE

If the operand is self-defining, it must be a decimal number.

This statement increases the current address counter by the value of the operand. *No* data will be loaded into the reserved area. When the program is loaded at execution time, the memory areas reserved by RES statements contain data and instructions left from the previous program.

NOTE

The next instruction that produces object coding starts a new object record. Therefore, RES statements should not be interspersed with DC statements in the program data areas unless necessary.

The RES statement is *ignored* if it appears before the START statement.

SDP Directive

Define SDAT pointer.

A label is not allowed with this directive.

One operand is required: the address of the initial SDAT in the program, in any of the formats A, B, C, or D (see Table 2-2).

This operand is placed into memory locations Ø and 1 and used as the program SDAP (SDAT Pointer) or Active Record Ø. This statement should appear only *once* in an assembly. Its operand is converted to tri-octal and placed in the object output 'UCORE' record. This statement is ignored if it appears before the START statement.

The operand may be an external label.

SGL Directive

Single-space listing. (This directive is needed only if double-spacing was specified earlier.)

A label or operand is not allowed with this directive.

This statement stores a *new page* on the assembly listing. Subsequent statements are single-spaced. This directive *may* appear before the START statement.

SPACE Directive

Leave a space on listing.

A label or operand is not allowed with this directive.

This statement causes a blank line to be left on the assembly listing. This directive *may* appear before the START statement.

NOTE

The SPACE statement itself does *not* appear on the listing.

START Directive

Start assembly.

A label is not allowed with this directive.

If this is to be a relocatable assembly, one of the following operands must be used:

REL - ZZZZUCORE record is generated.

REL,Ø - This operand sets the LO ahd HI fields in the ZZZZUCORE record to zero, regardless of program size.

Basic instructions (and certain directives) are *ignored* when they appear before the START statement. This statement must appear once (and only once) in each assembly and indicates to the Assembler that basic instructions are to be converted to object code from this point on until the END card is reached.

TITLE Directive

Specify a title for the assembly.

A label is not allowed with this directive.

One operand is required: any character string enclosed in single quotes.

This operand (up to 50 characters) is used as a title for the program listing. This statement causes a new page with the specified title to begin. This title is printed at the top of each page of the listing until changed by another TITLE statement. This statement *may* appear before the START statement.

Example:

```
TITLE       'PROGRAM A - - WRITTEN BY J. DOE'
```

NOTE

The TITLE statement itself does *not* appear in the assembly listing.

NOGEN Directive

Inhibit second and following print lines.

A label or operand is not allowed with this directive.

This statement suppresses the additional print lines associated with a particular statement. Only the first line is printed. It may also be used to suppress the second and following print lines of an RTN or MDL pseudo-op or the additional lines associated with a "DC" command.

GEN Directive

Cancel previous NOGEN directive.

A label or operand is not allowed with this directive.

This statement inhibits the effect of any previous NOGEN directive.

COM Directive

Specify entry point for AR/I information.

A label is not allowed with this directive.

One operand is required: it must be a label, elsewhere defined.

This statement is like an ENTRY statement, except COM is used to make active record information or a record number within an SDAT available to other programs. This causes the Collector to pass a 1-byte, unbiased parameter. COM should appear

after the START directive and must precede all basic instructions, DC statements, and EXTRN statements. This statement may be used only in relocatable progrms (see "Entry" directive, page 2-19). (See also "Relocatability", page 2-31.

INC Directive

Includes operand (library subroutine).

A label is not allowed with this directive.

One operand is required; it must be an ENTRY name.

This statement generates an EXTRN object record with a special flag. During the collection phase, this record directs the collector program to include the specified subroutine (ENTRY name) on to the object program tape being collected.

LIST Directive

Generate listing output (This directive is needed only if "NLIST" was specified earlier).

A label or operand is not allowed with this directive.

The LIST statement directs the assembler to begin transfer of the listing output to the device specified via the Sense switches (see "Operating the Assembler" on page 2-34.

NLIST Directive

Generate no listing output.

A label or operand is not allowed with this directive.

The NLIST statement directs the assembler to terminate transfer of the listing output to the device specified via the Sense switches (see "Operating the Assembler" on page 2-34.

NOTE

When NLIST has not been specified, the listing output is generated unless suppressed via the Sense switches (see "Operating the Assembler", page 2-34.

PAGE Directive

Set current address pointer to starting address of the next page.

A label or operand is not allowed with this directive.

NOTE

Each page represents 377_8 bytes of memory. Page boundaries are defined as follows:

Page	Bytes
1	000-000 through 000-377
2	001-000 through 001-377
3	002-000 through 002-377
⋮	⋮ ⋮

GENERATING A STORAGE DESCRIPTOR AREA TABLE (SDAT)

Each entry in an SDAT consists of two addresses. Any entry can be defined by the SD instruction. A storage descriptor area table (or portion thereof) is started by the SSDT directive. Except for comment cards or EQU statements, no statements can be interspersed with SD statements.

SSDT Directive (Start Storage Descriptor Table)

A label is optional on this directive. (Normally, this label is the operand in the SDP directive.)

A self-defining octal constant between Ø and 77 is the only operand. (Normally, this constant is Ø.)

The value of the operand is the value assigned to the label of the first SD statement following the SSDT directive. Thereafter, each SD statement has a value *one greater* than the previous value assigned to its label.

SD Directive (Storage Descriptor)

A label is optional. The value assigned to the label is a record number between 0 and 77 (octal) one greater than that of the previous SD statement. (If the previous statement was an SSDT, the value of the label will be that of the operand of the SSDT.)

Two operands appear in the SD statement: each in one of the formats A, B, C, or D (see Table 2-2).

The SD instruction will generate two 2-byte addresses, as specified by the operands.

GENERATING AN ITEM DESCRIPTOR TABLE (IDT)

Each entry in an SDAT consists of two 1-byte "relative displacements" from a record area describinb the start and end bytes of a field. An entry can be defined by the ID instruction. An item descriptor table (or portion thereof) is started by the SIDT directive. Except for comment cards or EQU statements, no statements can be interspersed with ID statements. Figure 2-1 demonstrates the relationship of labels in the SDAT, IDT, and record area, and their use in the executable instruction section.

SIDT Directive (Start Item Descriptor Table)

A label is optional on this directive. (Normally, this label is referenced as the second address in an SD instruction.)

The first operand is the address of the record being described by this table. (Normally, this address is the first address in the associated SD instruction.) This operand may be in one of the formats A, B, C, or D (see Table 2-2).

The second operand is the "active record/item" number to be assigned to the first item described in the table. Thereafter, each item in the table will have a value, *one greater* than the previous, assigned to the label of the ID defining it.

ID Directive (Item Descriptor)

The four types of ID operands, are described under separate headings below.

GAP DEFINER

A label is not allowed with a "gap" definer.

Two operands, separated by commas, are used:

1. The first is a *minus* sign;
2. The second is a *decimal* number defining the length of the gap.

This type of ID does *not* generate an item descriptor. It signifies that the next item descriptor will have, as its "start" byte, a value reflecting the existence

of the gap defined by this descriptor. The gap is considered to start at the byte immediately after the "end" byte of the previous item (or at Ø if the SIDT statement preceded).

REDEFINE AND LEAVE GAP

A label is not allowed with this type of ID.

Two operands, separated by commas, are used:

1. The first operand is the letter *R* (redefine).
2. The second operand is a *decimal* number defining the length of the gap (it may be "Ø").

This type of ID does *not* generate an item descriptor. It signifies that the next item descriptor will have, as its "start" byte, a value reflecting the existence of the gap defined by this descriptor. The gap is considered to start at the "start" byte of the previous item (or at Ø if the SIDT statement preceded).

DEFINE START AND LENGTH OF ITEM

A label must be used with this type of ID. The value assigned to this label is *one* more than the value assigned to the previous item. (If this is the first ID in the table to describe an item, the value assigned to the label is that of the second operand of the SIDT defining the start of the table.)

Two operands, separated by commas, are used:

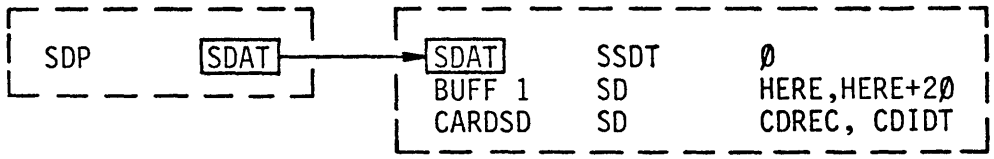
1. The first operand defines the start of the item. It may be a label (format B, C, or D, see Table 2-2) or a *decimal* number indicating the displacement from the start of the recird (from Ø to 255). If a label is used, the displacement is caluclated by subtracting from its value that of the first operand of the SIDT statement starting the table.
2. The second operand is a self-defining *decimal* number from 1 to 256 specifying the length of the item being described.

This type of ID generates a 2-byte item descriptor.

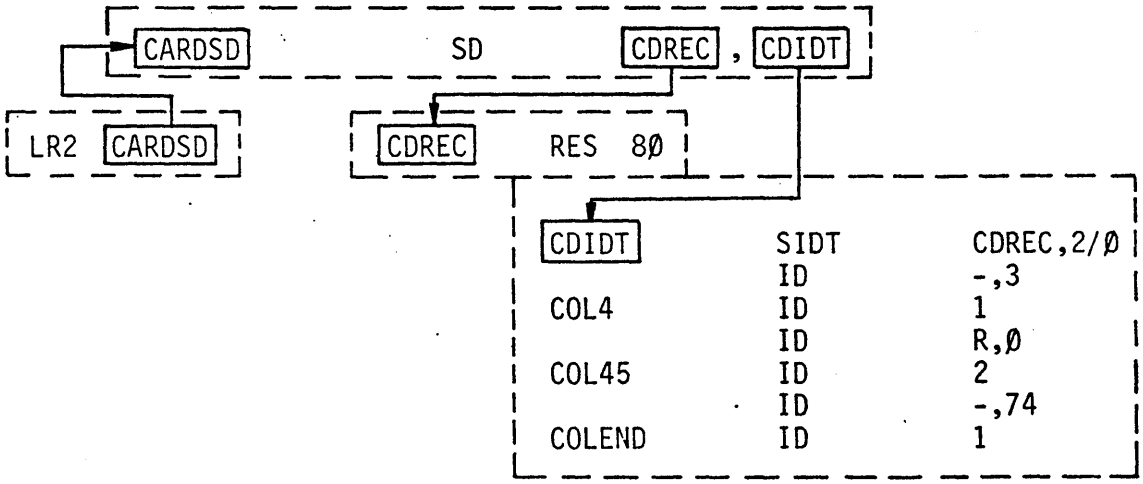
DEFINE LENGTH OF ITEM

A label must be used with this type of ID. The value assigned to this label is *one* more than the value assigned to the previous item. (If this is the first ID in the table to describe an item, the value assigned to the label is that of the second operand of the SIDT defining the start of the table.)

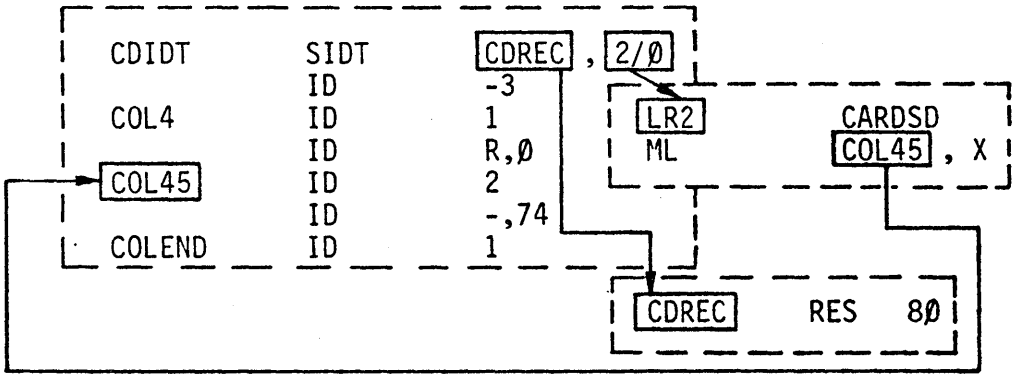
1) The SDP instruction points to the SSDT directive.



2) The first operand of a record SD points to the actual data area; the second points to the SIDT directive; the label is used in the "load active record n". instruction.



3) The first operand of the SIDT directive points to the actual data area: the second operand specifies the active record into which the corresponding SD will be loaded.



4) Labels defined in the IDT (by ID instructions) are used as 1-byte operands in instructions. (In this example, 'COL45" will have a value of "201" and describe columns 4 and 5 of the record in "CDREC".)

Figure 2-1. Use of SDAT and IDT

EDITING SOURCE INPUT

So that anticipated modifications to source programs may be coded but not assembled, the Assembler provides an editing feature. Based on the value of a code in column 72, the Assembler will either treat a particular statement as a comment or as a statement to be assembled. Two types of editing are supported, as described below.

Normal Editing (Switch H)

A. Edit Switch OFF:
   All statements will be assembled, except those with "+" (plus sign) in column 72.

B. Edit Switch ON:
   All statements will be assembled, except those with "-" (minus sign) in column 72.

The programmer should write his program such that anticipated deletions are coded with a minus sign, while anticipated additions are coded with a plus sign. Turning on the Edit switch during assembly gives him the "anticipated" program with all changes.

Three-Program Editing (Switches G and H)

Using this method, "+" or "-" should not appear in column 72.

A. Switch G and H OFF: All statements will be assembled.

B. Switch G ON, H OFF: All statements will be assembled, except those with the digits 1,3,5 or 7 in column 72.

C. Switch GO OFF, H ON: All statements will be assembled, except those with the digits 2, 3, 6, or 7 in column 72.

D. Switch G ON, H ON: All statements will be assembled, except those with the digits 4,5,6 or 7 in column 72.

Conceptually, this method allows the programmer to have three different "versions" of a program with one source input. Column 72, in each case, contains a number representint the decimal sum of the "version numbers" in which the statement will not appear. The versions are numbered: 1 (G OFF; H ON); 2 (G ON; H OFF); and 4 (G ON; H ON).

The following tables indicate whether a particular statement will be assembled (A) or not (N) under the various switch settings.

	NORMAL EDITING CONTENT OF COLUMN 72		
	+	-	Other, except 1 through 7)
H OFF	N	A	A
H ON	A	N	A

	THREE PROGRAM EDITING CONTENT OF COLUMN 72							
	1	2	3	4	5	6	7	(Other, except + or -)
G OFF; H OFF	A	A	A	A	A	A	A	A
F OFF; H ON (Version 1)	A	N	N	A	A	N	N	A
G ON; H OFF (Version 2)	N	A	N	A	N	A	N	A
F ON; H ON (Version 4)	A	A	A	N	N	N	N	A

RELOCATABILITY

Predefined External Labels

When writing a relocatable program (see START directive), certain special predefined external labels may be used:

- HICORE - The address of the highest byte of core.
- NXCORE - The address of the next unused byte of core after collection of all modules.
- PBIASE - The P-BIAS of the main (first) module in a collection.
- SDAPE - The SDAT-POINTER of the main (first module in a collection.

EXTRN statements for these names must not be written. Reference to external labels (both predefined and those defined by EXTRN statements) must be made to the label only, not with "plus" or "minus" notation.

EXTRN, ENTRY and COM Statements

EXTRN statements in an assembly refer to labels defined in other modules. Such labels are referenced by ENTRY and COM statements in the modules in which they are defined.

The collector will gather together all modules with matching EXTRN and ENTRY or COM labels and substitute the proper addresses for each occurrence.

'ZZZZUCORE' Card-Image

The assembler will produce a 'ZZZZUCORE' card-image in every relocatable assembly to communicate to the Collector miscellaneous information about the Assembler's use of core memory.

ERROR FLAGS IN LISTINGS

Each statement in the source program is checked for syntax while being assembled. If an error is found, a letter indicating the type of error is printed on the right side of the listing on the same line as the statement in error. The left side of the listing (where machine coding is printed) will have asterisks printed in the same line as any syntax error. Table 2-3 lists syntax error flags and their respective meanings.

Absolute Non-relocatable Address

Absolute non-relocatable addresses will be generated by the assembler for address type operands by either using a constant or preceding the label or constant with an equal (=) sign in the operand field.

MODES OF OPERATION

The SYSTEM 2400 Assembler provides for both single file and multi-file assembly processing. Sense switch E is used to select the mode of operation:

Switch E ON - Selects multi-file assembly.
Switch E OFF - Selects single file assembly.

Single File Assembly

Single file processing can be performed using source input from punched cards or magnetic tape and requires operator attention between each assembly.

Multi-File Assembly

Multi-file processing can only be performed using source input from magnetic tape. The card reader will be used to read the file control cards necessary to control the multi-file assembly process.

FILE CONTROL

The assembler uses SCOD records for file selection and two consecutive tape marks to indicate the end of tape.

FILE CONTROL DIRECTIVES

The assembler will recognize three File Control directives. These directives are always provided to the assembler via card input. The three directives are described below.

Table 2-3. Assembler Syntax Error Flags

FLAG	MEANING
C	PHASING ERROR: 1. On END card, different number of card-images read in the two phases. 2. Label on this card disagrees with its location in Phase 1.
E	ENTRY ERROR: ENTRY card found after other Assembler instructions (in Phase 1 as well).
F	FORMAT ERROR: 1. Label is too long. 2. Operation code is too long. 3. Invalid SD or ID instruction. 4. ENTRY or EXTRN directive used in unrelocatable program.
I	INVALID OPERAND: 1. Incorrectly specified operand. 2. Incorrect number of operands.
L	ILLEGAL LABEL
M	MULTI-DEFINED LABEL: 1. Label on statement is defined elsewhere. 2. Operand contains reference to multi-defined label.
O	ILLEGAL OPERATION CODE
U	UNDEFINED: 1. Operand contains reference to undefined label. 2. Operand in Assembler directive was not previously defined (in Phase 1).
V	OVERFLOW Label on statement not saved or given value because of overflowing core (in Phase 1). Label on rejected statement also rejected (Phase 2 only).
W	WARNING: 1. Operand value truncated to one byte. 2. Negative address generated. 3. Invalid ID specification in operand. 4. Invalid use of a literal.
X	EXTRN LOST Because of overflowing core.
Y	ILLEGAL ADDRESS-TYPE DC Length must be 2 or 4 bytes.

Select File Call

The Selective File Call directive is used to identify a file to be assembled from magnetic tape input. The card contains the name (XXXXXX) that appears in the $$$$CODXXXXXX record. The name (up to six characters) must be left-justified on the input card starting in column one. The cards identifying the files to be assembled must be ordered in the same manner as their respective files are ordered on the magnetic tape. For information on the $$$$SCOD records, see the Librarian program in the SYSTEM 2400 Utilities Manual, Form No. PM-2601.

Multi-File Call

This directive is used to assemble all files on the magnetic tape, or all remaining files following the last file specified by a Select File Call directive. The card contains a plus sign (+) left-justified in column one.

End

This directive directs the assembler to cease communications with the card reader and must be the last card in the File Control card deck. The card contains the "/*" symbols (slant and asterisk) left-justified in column one.

OPERATING THE ASSEMBLER

The program is executed in two phases:

PHASE 1:

- Source statements are read.
- The Assembler constructs a table of labels used in the program.

In PHASE 2:

- Source statements are read again.
- The Assembler produces object coding and a listing.

To execute the Assembler:

1. Load the Assembler into cord.
2. Turn on any Sense switches needed.
3. If a multi-file assembly, place file control card deck in card reader.

   Press the RUN switch.

   If source statements are being read from a card reader, the Assembler will halt at the end of Phase 1. The operator must place the source statements back into the input hopper, make sure the reader is ready, then press the RUN switch to execute Phase 2.

   If subsequent assemblies are wanted, return to step 2.

Use of Sense switches and indicator lights is detailed in Table 2-4.

TABLE 2-4. Lights and Switches Used by Assembler

USE OF LIGHT	NAME	USE OF SENSE SWITCH
ON, mount new list tape	A	OFF: Normal ON: Restart Assembler
ON, if in listing routine	B	OFF: Listing on 'LP' Device ON: Listing on 'MT' Device[1]
ON, if in input routine	C	OFF: Input on 'MT' Device ON: Input on 'CR' Device
ON, if in object routine	D	OFF: Object Output on 'MT' Device ON: Object Output Suppressed
ON, if I/O error occurs	E	OFF: Single File Assembly ON: Multi-File Assembly
ON during assembly	F	OFF: Normal ON: Listing Suppressed
ON during Phase 2	G	OFF: Normal or Edit PGM2 ON: Edit PGM1 or Edit PGM 4
ON during Phase 1	H	OFF: Normal or Edit PGM1 ON: Edit PGM2 or Edit PGM 4

LP = LINE PRINTER
CR = CARD READER
MT = MAGNETIC TAPE

I/O ERROR: If an I/O error occurs light E is illuminated as well as C, or D to indicate which operation failed. Press RUN to continue the assembly operation and bypass the error condition.

ERROR COUNT: The total number (binary) of errors encountered during the assembly run is displayed in the indicator lights (A-H).

---

[1]With switch B on, when an end-of-tape is detected, two tapemarks are written, the tape is rewound, and the program halts. Mount another list tape and press RUN to complete the assembly.

DEVICE CONFIGURATION

Input - May be on card reader or magnetic tape.

Object Output - May be on magnetic tape or suppressed.

Listing - May be on line printer or magnetic tape or suppressed.

Files on magnetic tape are assigned at the commencement of the run, depending on sense switch settings (to select devices) and the system configuration. First, the listing, if on tape, is assigned to MT3 or *next-highest* available drive. Then, object output, if produced, is assigned to MT2 or *next-highest* available drive. Finally, the input, if on tape, is assigned to MT1 or *next-highest* available drive.

The card reader and line printer are both device "Ø" in their respective classes, when used.

Unless all tape drives are needed for the execution of the Assembler, "MTØ" is *not* used, leaving MTØ free for the master program tape.

If more tape drives are required than exist in the system configuration, the Assembler, having determined this, will return to Step 2 of the operating instructions above.

All possible Assembler configurations are summarized in Table 2-5.

TABLE 2-5. Assembler Configurations

SENSE SWITCHES				SYSTEM CONFIGURATIONS																		
				4 (or more) MT's						3 MT's					2 MT's				1 MT		
B	C	D	F	CRØ	LPØ	MTØ	MT1	MT2	MT3	CRØ	LPØ	MTØ	MT1	MT2	CRØ	LPØ	MTØ	MT1	CRØ	LPØ	MTØ
Ø	Ø	Ø	Ø	i	L	M		0		i	L	M		0	i	L	M	0	i	L	0
Ø	Ø	Ø	1	i		M		0		i		M		0	i		M	0	i		0
Ø	Ø	1	Ø	i	L	M				i	L	M			i	L	M	0	i	L	M
Ø	Ø	1	1	i		M				i		M			i		M		i		M
Ø	1	Ø	Ø		L	M	i	0			L	M	i	0		L	i	0	*	*	*
Ø	1	Ø	1			M	i	0				M	i	0			i	0	*	*	*
Ø	1	1	Ø		L	M	i				L	M	i			L	M	i		L	i
Ø	1	1	1			M	i					M	i				M	i			i
1	Ø	Ø	Ø	i		M		0	L	i		M	0	L	i		0	L	*	*	*
1	Ø	Ø	1	i		M		0		i		M		0	i		M	0	i		0
1	Ø	1	Ø	i		M			L	i		M		L	i		M	L	i		L
1	Ø	1	1	i		M				i		M			i		M		i		M
1	1	Ø	Ø			M	i	0	L			i	0	L	*	*	*	*	*	*	*
1	1	Ø	1			M	i	0				M	i	0			i	0	*	*	*
1	1	1	Ø			M	i		L			M	i	L			i	L	*	*	*
1	1	1	1			M	i					M	i				M	i			i

LEGEND:
i = Input (source)
0 = Output (object)
L = Listing
M = Master Program Tape (MPT)
CR = Card Reader
LP = Line Printer
MT = Magnetic Tape
* = Not enough tape drives; program will restart

OBJECT CODE MAP

Figure 2-2 reflects the layout of an object program on object tape.

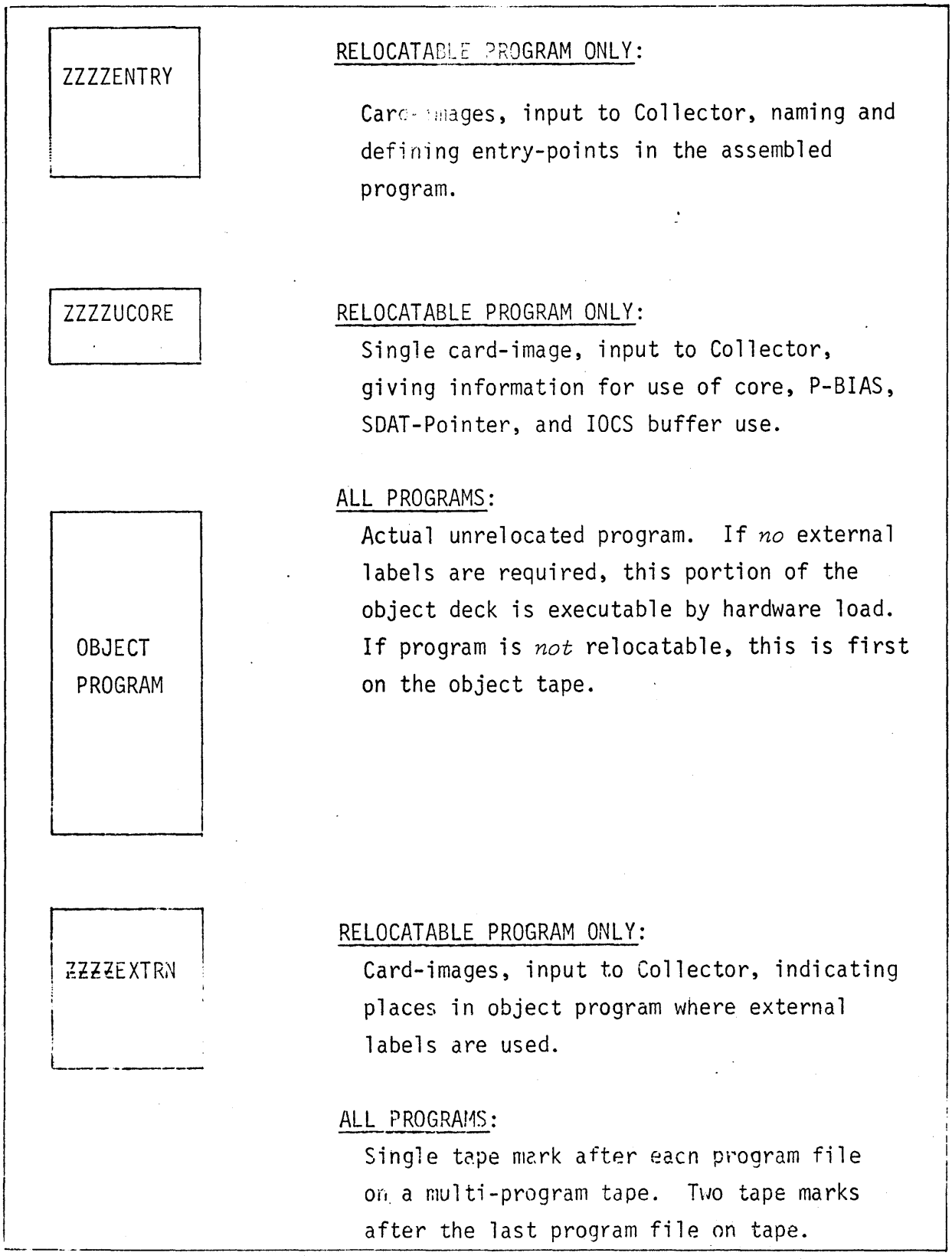


Figure 2-2. Object Code Map

APPENDIX A

INTERRUPT PROGRAMMING

Provisions have been made within the 502 Processor to interrupt the main program by events which occur asynchronously with main program execution. The following events can interrupt the processor:

CLASS 1 - Monitor interrupts: associated with input and output buffering on I/O Selector Channels. The interrupt occurs, if enabled, when an active input or output buffer goes from the active state to the inactive state indicating that the buffer is filled or emptied.

CLASS 2 - Service interrupts: associated with the peripheral devices connected to an I/O Selector Channel. The interrupt occurs, if enabled, when the peripheral device sets its interrupt line to the processor indicating service is required by the device.

CLASS 3 - Special interrupts: primarily associated with the processor hardware itself; machine checks, illegal instructions, and add on hardware modules. The interrupt occurs, if enabled, when a unique condition or an error is detected by the hardware indicating that some action must be taken by the software.

SOFTWARE INTERRUPT LINKAGE

Software interrupt linkage is provided for in the design of the processor's Program Control Block (PCB) (see Figure A-1). When an interrupt occurs, the current instruction being executed in the main program is completed and then the program location pointer (P) is forced to a fixed memory location in the PCB; one for each class of interrupt. Each of the locations in the PCB reserve four bytes for an interrupt linkage instruction which, when executed, provides a branch to an appropriate interrupt handling routine. The interrupt linkage instruction normally is a GOTO Subroutine (GSB) instruction. Execution of the GSB instruction causes P, the return address to the main program, to be saved in a push-down stack buffer as specified by the OP1 item.

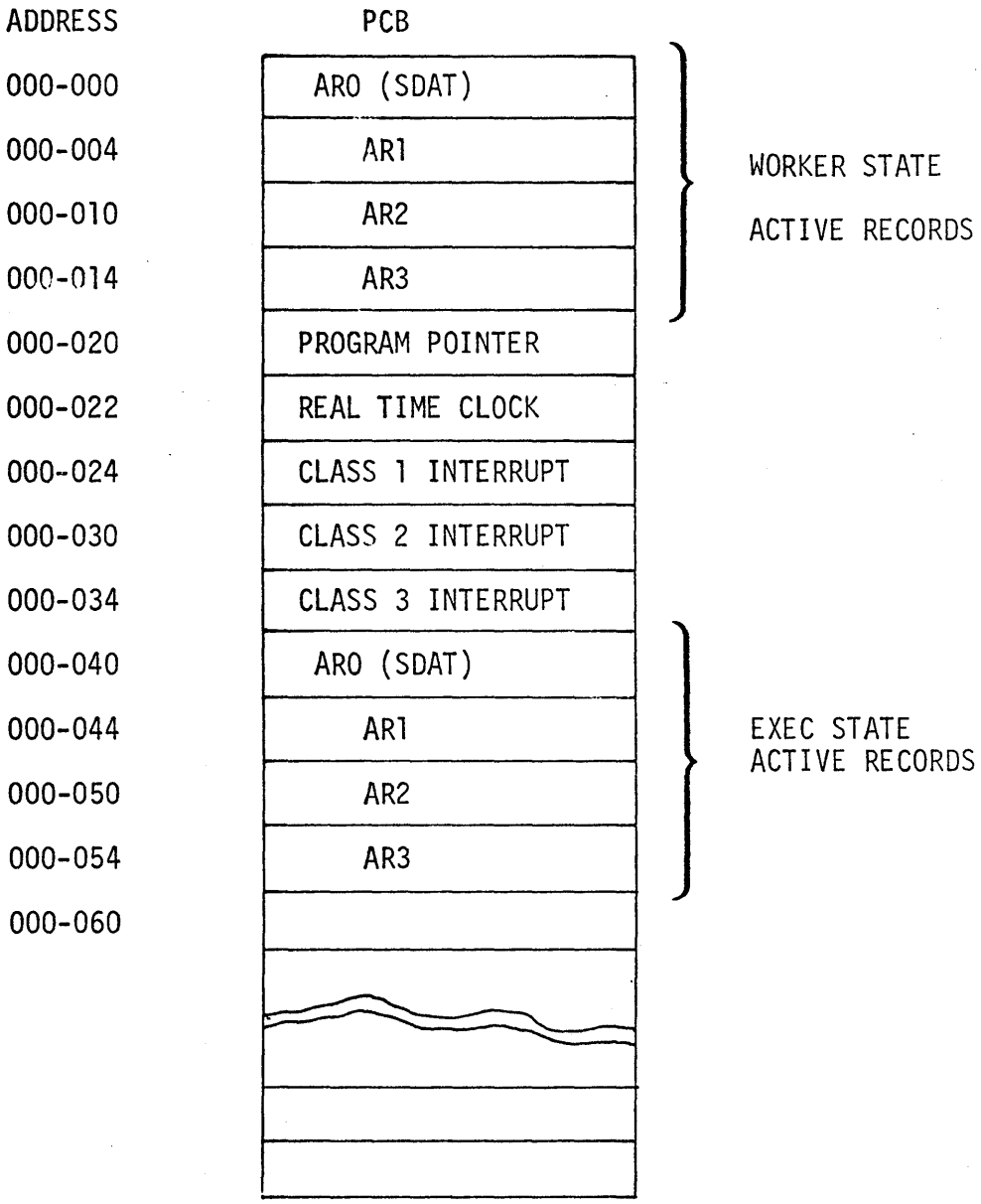


Figure A-1. 2408 Processor-Program Control Block.

Return to the main program from the interrupt handling routine normally is accomplished by the execution of an Interrupt Return GOTO (GIR) instruction, which extracts the return address as specified by the OP1 item from the push-down stack buffer and places it in the P register. This return address is the next instruction executed in the main program.

WORKER/EXECUTIVE STATE

The processor has two states of operation: the Worker State and the Executive State. Each state has a separate set of Active Records in the Program Control Block (see Figure A-1). Address 000-000 through 000-017 contain the Active Records for the Worker State; addresses 000-040 through 000-057 contain the Active Records for the Executive State.

On power-up, restart, and P-start, the processor is forced to the Worker State. The processor is switched to the Executive State by either of the following methods:

Swap States (SWS) Instruction: This instruction switches the processor from its current operating state to the other state.

Interrupt: An interrupt automatically forces the processor to the Executive State.

The swap states condition is effective immediately when the SWS instruction is executed while in the Worker State. However, in the Executive State, one additional instruction is executed before the processor switches to the Worker State to allow the execution of an Interrupt Return GOTO (GIR) instruction which references an AR/I item. For example, the normal exit from an interrupt handling subroutine is the execution of a SWS instruction followed by the GIR instruction.

ENABLE/DISABLE INTERRUPTS

Any or all interrupts may be selectively enabled or disabled by using the Interrupt Mask (IM) instruction. This instruction is normally used at the beginning of the program to enable only those interrupts that will be used by the program. All disabled interrupts are ignored.

SET/CLEAR INTERRUPT LOCKOUT

Interrupt lockout is a condition associated with program instruction execution. When the interrupt lockout condition prevails, interrupts that occur are saved in hardware logic and only when the interrupt lockout condition is cleared does the program honor these interrupts. The interrupt lockout condition may be caused by

any one of the following:

- When power is applied to the processor.
- When a RESTART-RUN operation is initiated.
- When an interrupt (Class 1, 2, or 3) occurs.
- When the Set Interrupt Lockout (SIL) instruction is executed.

If the program is to use interrupts, a Clear Interrupt Lockout (CIL) instruction m7st be executed.

The execution of this instruction allows only those interrupts enabled by the Interrupt Mask (IM) instruction to be honored by the processor. All other interrupts are ignored. When an interrupt does occur, the processor automatically locks out all other interrupts until the interrupt lockout condition is cleared. Normally, interrupt lockout is automatically cleared when exiting the interrupt routine by using the Interrupt Return GOTO instruction. The interrupt lockout condition may also be cleared by executing the Clear Interrupt Lockout (CIL) instruction. If the CIL instruction is used within an interrupt handling routine, the routines must be nested properly and associated designators stored to insure proper operation. Usually the CIL instruction is used following a Set Interrupt Lockout (SIL) instruction. The SIL instruction is used when a portion of the program must be run without interruption. When completed, the CIL instruction clears the interrupt lockout condition, allowing interrupts to be honored.

SAVE CONDITION DESIGNATORS & TALLY COUNTER

Upon entering an interrupt handling routine, the Store Tally Counter (STT) and the Store Designators (STD) instructions are normally the first instructions to be executed. These two instructions save the condition of the tally counter and the designators prior to executing instructions to determine the cause of the interrupt. The interrupt handling routine may use instructions which affect the condition of the tally counter and designators, thereby destroying their content as pertaining to main program (worker state) operation. Prior to exiting the interrupt handling routine, the Load Designators and Load Tally Counter instructions are executed to restore the tally counter and designators to their original condition under the worker state.

Condition Designators

The condition designators are contained within a 1-byte item and are listed as follows:

Bit 2^7 = I/O Parity Error
2^6 = Memory Parity Error
2^5 = Arithmetic Error
2^4 = Arithmetic Overflow Error
2^3 = BDMA Parity Error
2^2 = Greater Than Designator
2^1 = Abnormal Edit Error
2^0 = Equal Designator

The GOTO on Designators (GD) instruction may be used to test the individual bits except for bit 2^2 - Greater Than Designator Set.

Tally Counter

The tally counter is a 2-byte binary counter (see Figure A-2) which counts the number of data bytes moved or compared by instructions accessing AR/I type operands. Only the actual data transferred or compared is counted. Character fills or character eliminates are not. Note that any instruction that affects the tally resets the counter to zero before execution of the instruction. See Appendix D for a detailed list of the instructions that affect the tally counter.

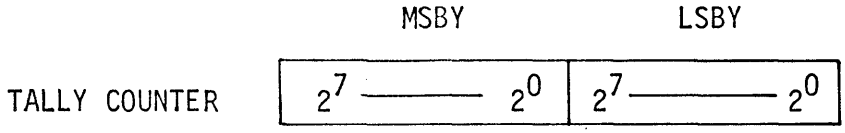


Figure A-2. Tally Counter

CLASS 1 - MONITOR INTERRUPTS

Monitor interrupts are associated with data buffering on I/O Selector channels. A monitor interrupt occurs when an input or output buffer on an I/O Selector channel switches from the active to the inactive state (filled or emptied), provided that monitor interrupts have been enabled and are not locked out. The monitor interrupt can be generated by any one of the eight I/O Selector channels.

When a monitor interrupt occurs, it forces a branch to address 000-024 which normally contains a GSB instruction. This instruction, when executed, saves P and branches to the interrupt handling routine. In the interrupt routine, a series of GOTO On Channel Interrupt (GCI) instructions may be used to determine which I/O Selector channel caused the interrupt. The GCI instruction also clears the interrupt condition for the I/O Selector channel tested.

Figure A-3 gives an example of a Monitor Interrupt handling subroutine. Upon entry, the contents of the tally counter and designators are stored and the I/O Selector channels are tested to determine which channel had a buffer terminate. Upon detecting a channel whose buffer terminated, a branch is made to the processing portion of the subroutine. When processing is completed, the subroutine restores the tally count and designators, swaps states, and returns to the main program.

This subroutine example shows the channels being tested in order of channel priority; channel 7, then channel 6, etc. Any order may be used, but the channels should be tested in an order which gives priority to high-speed buffering devices. For example, if a disk is connected to channel 4, a magnetic tape unit to channel 6 and a card reader to channel 2, then channel 4 should be tested first, channel 6 second and channel 2 third to effectively service the speeds of the three devices.

Note that once an input or output buffer is initiated on a channel, it runs asynchronously with the main program. With a number of buffers initiated on various channels, it is possible to get one interrupt (assume channel 4) that forces the program to the interrupt routine, but prior to testing channel 4, channel 6 also interrupts. Although interrupts are locked out, if channel 6 is tested before channel 4, the interrupt on channel 6 will be honored first. In this case, the channel 6 interrupt is cleared, but not the channel 4 interrupt. On return to the main program, one instruction is executed and then the channel 4 interrupt is honored.

CLASS 2 - SERVICE INTERRUPTS

Class 2 service interrupts are generated by peripheral devices connected to I/O Selector channels. Eight of these interrupts, one for each I/O Selector channel, are available for connected devices. Service interrupts allow a device to interrupt the processor when it requires special action to be performed by the processor. To generate a service interrupt, the peripheral device must be able to set the service request line in the I/O cable connecting it to the I/O Selector channel. If more than one device capable of generating an interrupt is connected to the same I/O cable (channel, the programmer must then request status from the devices to determine which one caused the interrupt.

When a service interrupt occurs, it forces a branch to address 999-030 which normally contains a GSB instruction (see Figure A-1). This instruction, when executed, saves P and branches to the interrupt handling routine. The interrupt

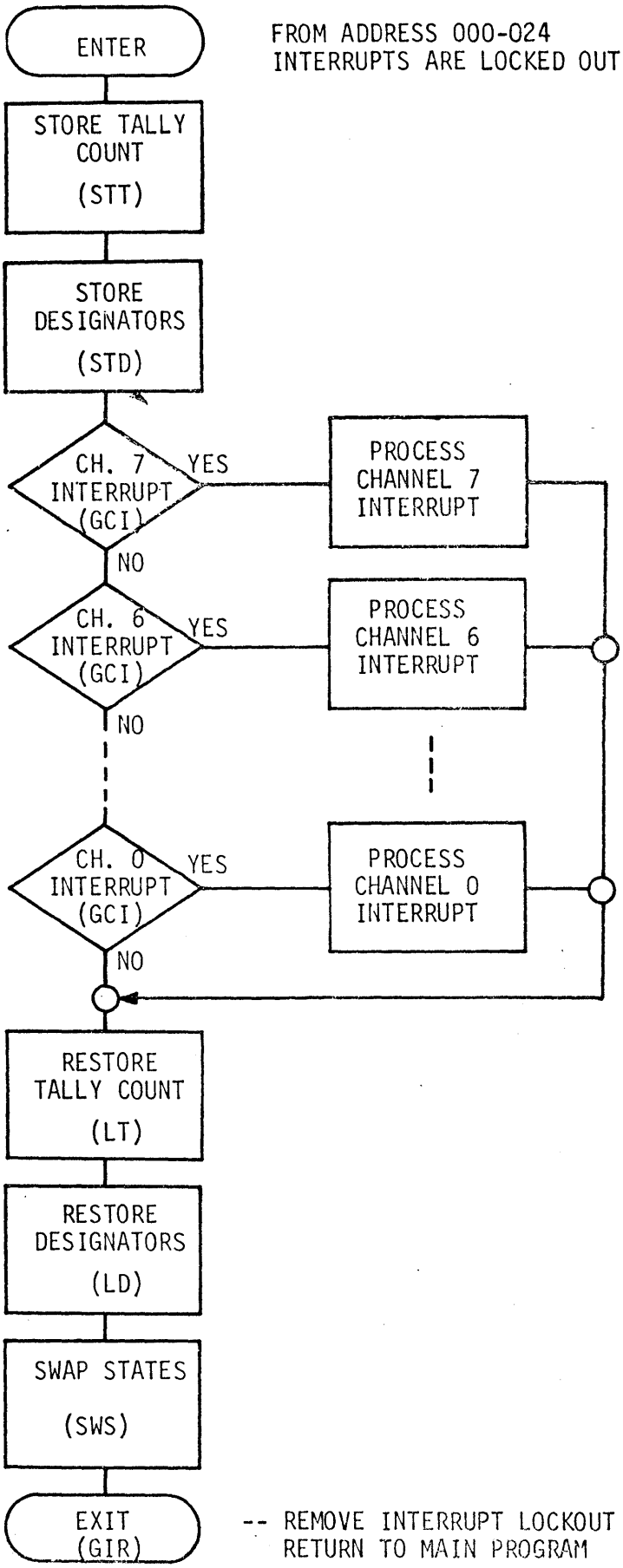


Figure A-3. Monitor Interrupt Processing Flow Diagram (example).

routine should be designed to identify the interrupting device by executing a series of GSI instructions (see Figure A-4), similar to the interrupt routine used for monitor interrupts.

If more than one device on a given channel can generate the service interrupt, the interrupting device is identified by requesting status from each device. When status is requested from the interrupting device, the service request status bit in the status reply will be set and the device will clear its service request on the I/O cable. Class 2 service interrupts are handled similar to monitor interrupts and can be selectively enabled or disabled using the IM instruction.

CLASS 3 - SPECIAL INTERRUPTS

Class 3 interrupts are generated by special conditions internal to the processor itself. Provisions have been made to detect up to eight class 3 interrupts. The interrupt sources are bit position encoded within a 8-bit byte and are obtained by using the Store External Instruction Error (SEE) instruction. The interrupt sources currently being used are as follows:

BIT POSITION	INTERRUPT SOURCE
2^0	Non-Operation Sub-Op Code
2^1	Not assigned
2^2	Delta Clock
2^3	Not assigned
2^4	Not assigned
2^5	Machine Check
2^6	BDMA Channel 6
2^7	BDMA Channel 7

When a class 3 interrupt occurs, it forces a branch to address 000-034 which normally contains a GSB instruction (see Figure A-1). This instruction, when executed, saves P and branches to the interrupt handling subroutine.

Figure A-5 gives an example of a Special Interrupt handling subroutine. Upon entry the contents of the tally counter and designators are stored and then the SEE instruction is executed to obtain the Class 3 Interrupt status which always consists of a 14-byte item as follows:

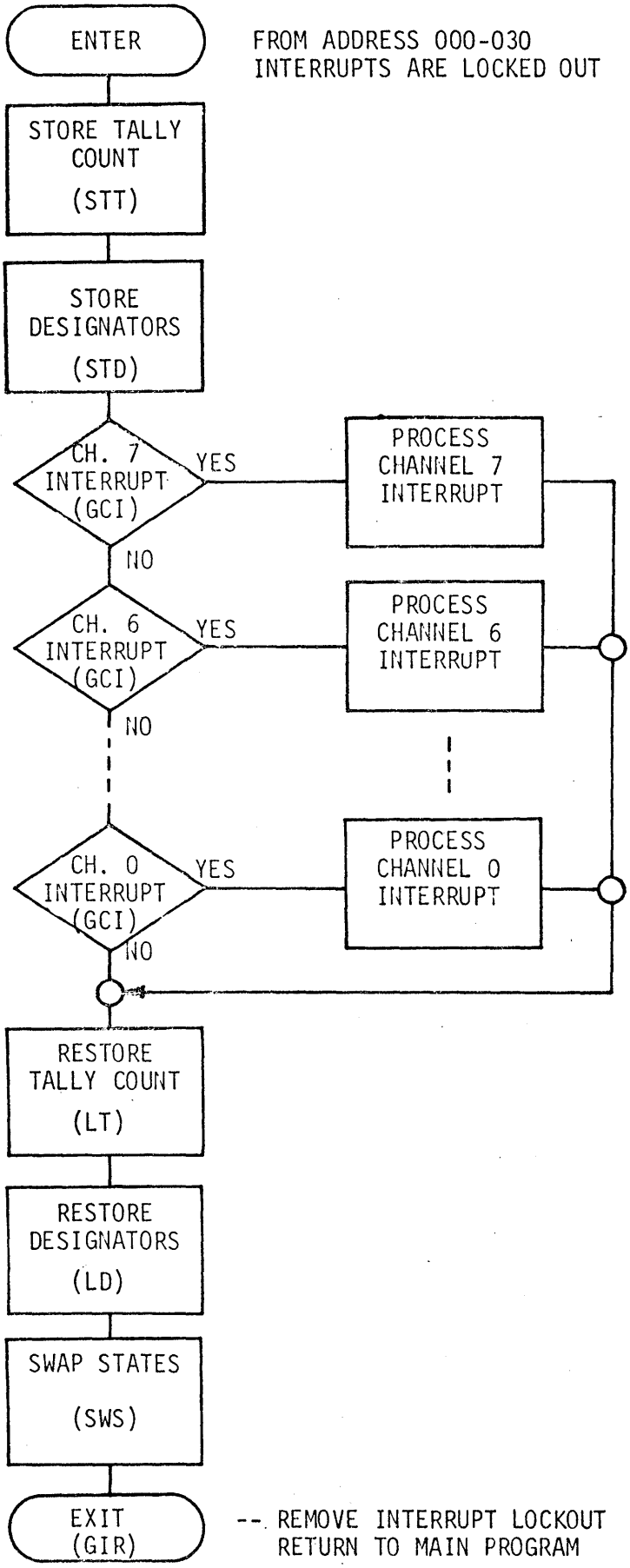
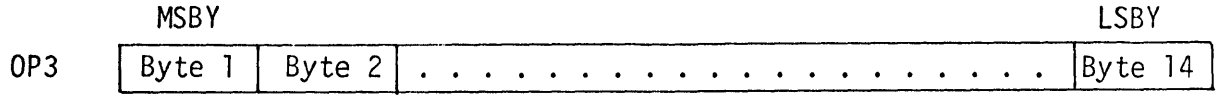


Figure A-4. Service Interrupt Processing Flow Diagram (example).



Byte two contains the bit-encoded interrupt source and is tested by using a series of GOTO On Designator (GD) instructions to determine the type of interrupt that had occurred. The order of testing these bits is a function of program design.

Non-Operational Sub-Op Code (2^0)

Assuming a Non-Operation Sub-Op Code error (bit 2^0 set), a check must first be made to determine if an illegal sub-op code was detected or if the processor lacks the hardware module to execute the instruction. This check can be done by testing the contents of byte 1 in the status item. It contains the sub-op code that could not be executed. If the sub-op code is illegal, error indicators may be displayed and the program halted while still in the interrupt handling routine. Return to the main program at this time without positively identifying what the sub-op was supposed to be may cause program instruction execution to be indeterminate. This point will become apparent in the following discussion relating to lack of hardware modules.

If the sub-op code is legal but cannot be executed due to lack of hardware, the programmer may then include additional software to perform the same operation as the unexecutable sub-op code. The recovery from this condition is simplified in that the sub-op code and the absolute address limits of the OP1, OP2 and OP3 items are defined in the status item (OP3 of SEE instruction). In addition, the return address stored in the push-down stack buffer (see GSB instruction) prior to entering this interrupt handling routine points to the absolute address of the literal for those instructions (sub-op) having four operands. The literal may be obtained by using this address and the return address must then be advanced by 1, such that on return to the main program, instruction execution sequence is in sync with instruction op codes. This situation can best be explained by describing in general what the processor is doing when it executes an Execute External (145) instruction with a sub-op code. Assume the following Execute External instruction is to be executed.

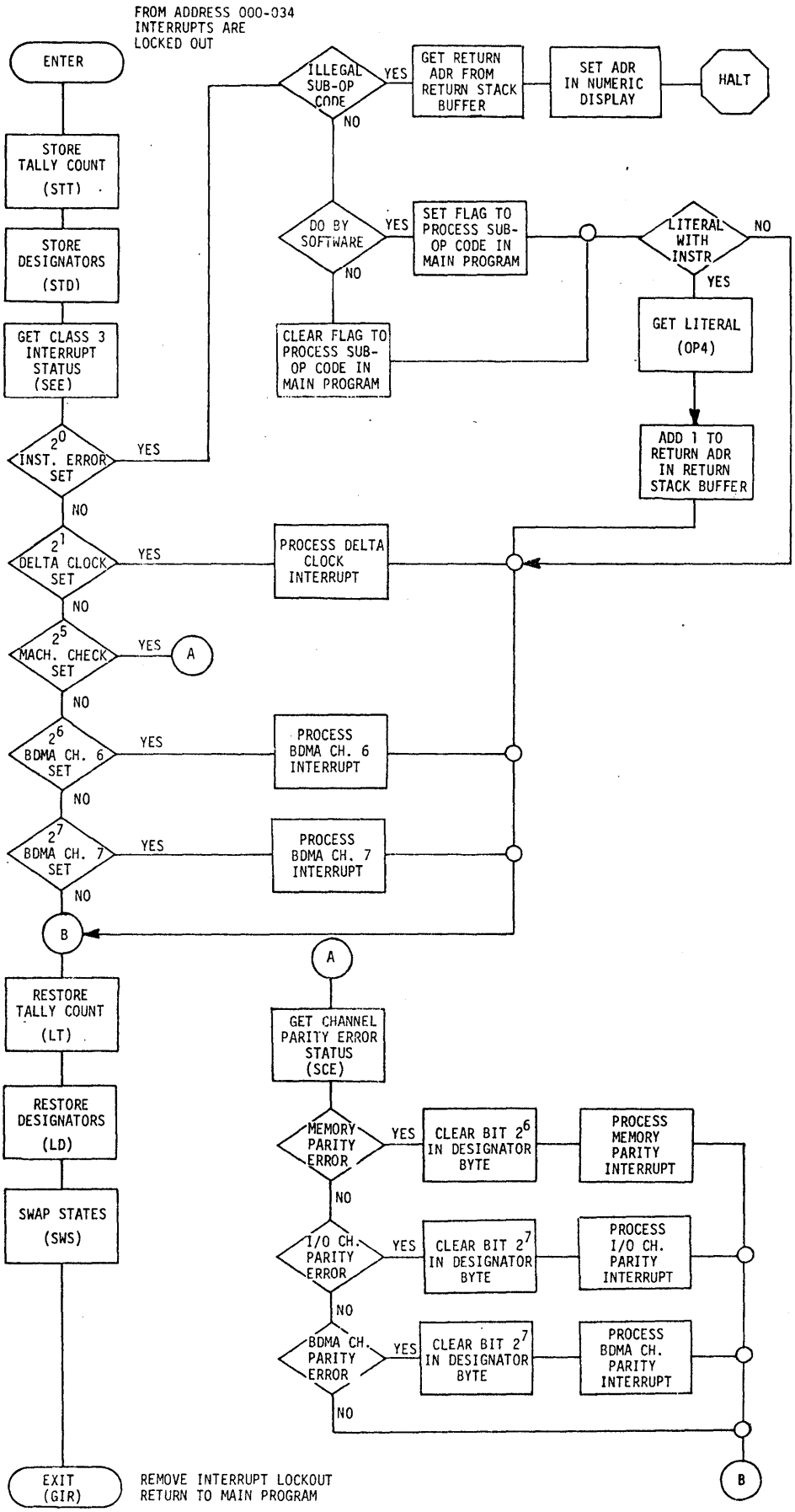
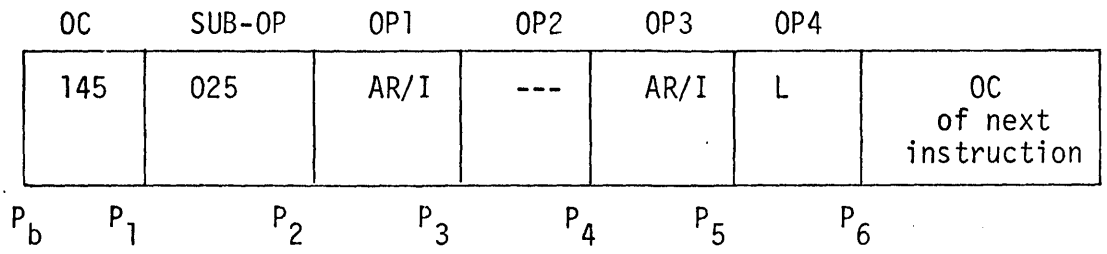


Figure A-5. Special Interrupt Processing Flow Diagram (example).

Multiple Literal Decimal Instruction



P_b - Program Pointer points to beginning of instruction. OC-145 when read specifies an External Instruction. Advance to P_1.

P_1 - Read SUB-OP code (MLD in this example). Advance to P_2.

P_2 - Read OP1 and generate the beginning and ending addresses for operand 1 item. Advance to P_3.

P_3 - Read OP2 and generate the beginning and ending addresses for operand 2 item. Note: This step takes place even though OP2 is not used by the instruction. Advance to P_4.

P_4 - Read OP3 and generate the beginning and ending addresses for operand 3 item. Advance to P_5.

P_5 - At this point the processor passes control to the multiply/divide hardware module. The module reads OP4 (literal) from the instruction and performs the multiply operation. When finished the program pointer is setting at P_6 and control is returned to the processor.

P_6 - The processor reads up the next instruction (OC) to execute, etc.

The execution of an illegal or unavailable instruction is essentially the same as described above except for the following:

1. An illegal instruction is detected when the SUB-OP code is read (P_1). Instruction execution continues and the beginning and ending addresses for operands OP1, OP2 and OP3 are generated advancing the program pointer to P_5. At this time the class 3 interrupt occurs. Operand OP4 is not read.
2. An unavailable instruction (hardware module missing) is detected after operand OP3 has been read and its beginning and ending addresses generated. The program pointer is at P_5. At this time the class 3 interrupt occurs. Operand OP4 is not read.

When the class 3 interrupt occurs, the contents of the program pointer (now at P_5 in example) is saved. The processor switches to the Executive State and the program pointer is forced to 000-034: the location of the next instruction to execute. At address 000-034, the programmer should have a GOTO Subroutine (GSB) instruction. This instruction, when executed, causes the program (worker state) return address (P_5) to be stored into the Push Down Stack Buffer and a branch is made to the programmer's subroutine to determine the cause of the class 3 interrupt Return (GIR) instruction is used. This instruction, when executed, takes the last program pointer address (P_5) from the Push Down Stack Buffer and forces it in the P register as the address of the next instruction to be executed in the worker state. With regard to the example above, the processor uses OP4 as the operation code (OC) for the next instruction to be executed. As a result, the instruction execution sequence is indeterminate.

To recover from the above situation, the programmer must add one (+1) to the return address (P_5) located in the Push Down Stack Buffer prior to return to the main program (worker state). In essence, the program pointer must be advanced to P_6 in the example above.

Although the primary function of being able to detect an illegal or unavailable sub-op code is program recovery, the feature may also be used by the programmer to obtain absolute addresses for specific AR/I operands. All that is required is to execute an Execute External (145) instruction with an illegal sub-op code (i.e., 176 or 177) and specifying up to three AR/I operands. The interrupt handling subroutine executes the SEE instruction to obtain the absolute addresses for the specified AR/I operands. The above technique may also be used to switch the processor from the Worker to the Executive state.

Delta Clock (2^2)

This bit if set indicates that the Delta Clock has counted down to zero (see LC instruction).

Machine Check (2^5)

A machine check interrupt occurs if the processor detects a parity error under the following conditions:

Memory Parity - A parity error was detected while reading a byte from core memory.

I/O Channel Parity - A parity error was detected while receiving a data or status byte from the peripheral device connected to an I/O Selector channel.

BDMA Channel Parity - A parity error was detected while receiving a data or status byte from the peripheral device connected to a BDMA channel, or a parity error was detected in the address furnished by the device connected to the BDMA channel in either a read or write memory cycle.

A GOTO On Designator (GD) instruction may be used to distinguish between the above parity errors. If an I/O or BDMA channel parity error, the Store Channel Parity Error (SCE) instruction may be executed to obtain additional status as to which I/O or BDMA channel caused the interrupt. In addition, the BDMA channel parity error is categorized as a data or status byte parity error or as an address parity error.

NOTE:

1. The following rule must be adhered to when processing a Machine Check interrupt:

   Clear the corresponding bit in the designator byte prior to exiting the interrupt subroutine (see Figure A-5).

   If the bit is not cleared, the processor, upon return to the Worker state, will detect the bit set and automatically generate another class 3 interrupt, forcing the program back to the interrupt handling subroutine.

2. The I/O PAR CK indication on the Processor Panel for the 2408 Processor indicates that a parity error has been detected on a processor I/O channel or a BDMA channel during a status or data transfer from a peripheral device. An address with bad parity transferred from a peripheral device on a BDMA channel also lights the indicator. The programmer, via software methods, may determine the exact cause of the parity error as described above.

APPENDIX B

PROGRAMMING ACTIVE RECORDS

The inherent design of address generation for Active Record Items in the processor logic allows the programmer to load one or more Active Records (AR's) with the execution of one Load Active Record instruction.

Existing documentation describes the Load Active Record instructions on a one-for-one basis:

LR1 Loads Active Record 1 (AR1)
LR2 Loads Active Record 2 (AR2)
LR3 Loads Active Record 3 (AR3) and
LSP Loads the Storage Descriptor Area Pointer (AR0)

The format for the above instructions consists of an op code and one operand which specifies a four-byte item in the Storage Descriptor Area Table (SDAT). The location of the SDAT table is defined by the contents of AR0 which is loaded during initial program load. Active Records 1, 2 and 3 are then loaded from the SDAT table using the LR1, LR2 and LR3 instructions. The SDAP table, whose location is specified by the contents of AR0 (bytes 0 and 1), does not require an Item Descriptor Table (IDT), since AR0 when used as an operand in an instruction always specifies a four-byte item (hardware design). As a result, the LR1, LR2, LR3 and LSP instructions can only load one AR in the PCB if the operand specifies AR0 as the Active Record.

Basically, the LR1, LR2, LR3 and LSP instructions are a move right-aligned instruction. The Operation Code (OC) specifies the beginning location into which the first byte of the item specified by OP1 is moved. The number of bytes moved is determined by the following factors:

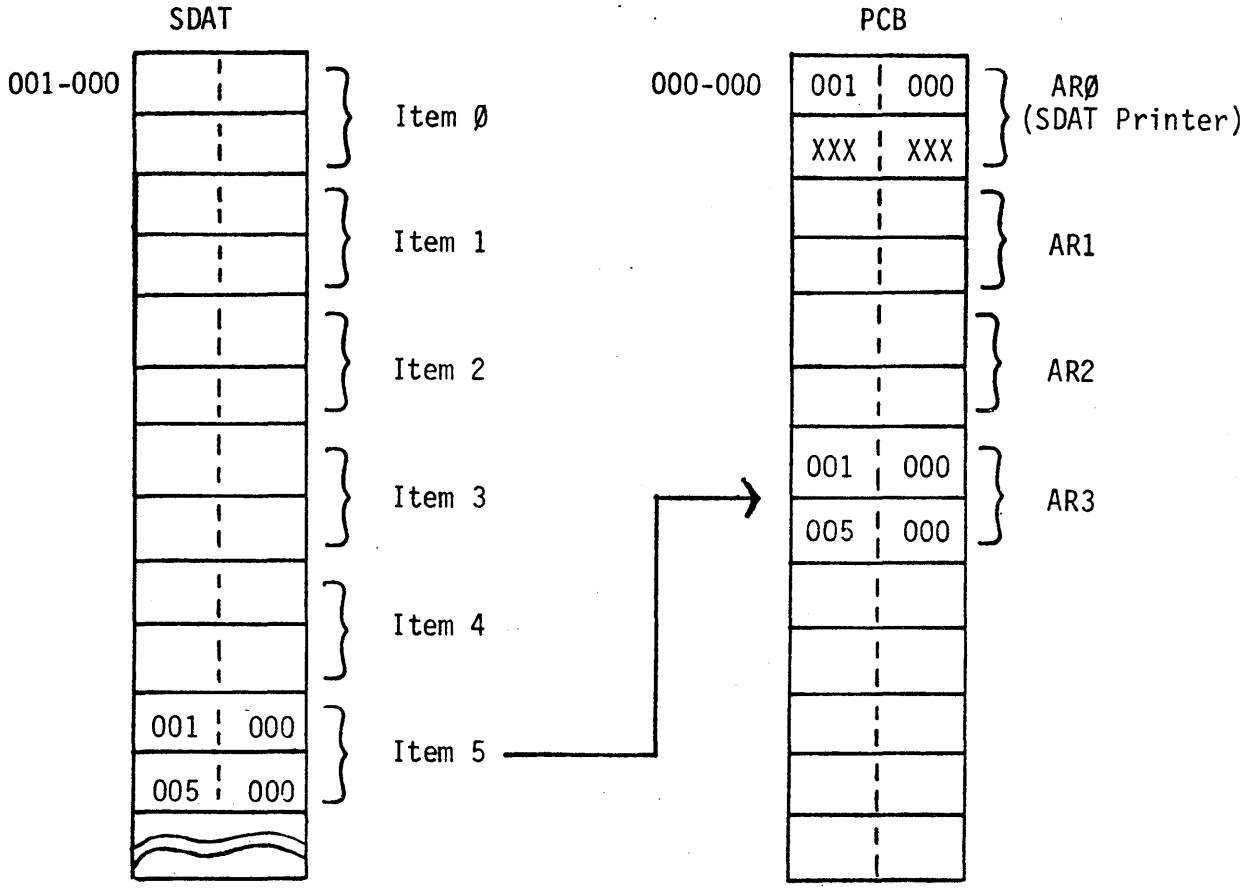
If OP1 specifies AR0 in OP1, then only four bytes are moved.

If OP1 specifies AR1, AR2 or AR3, then the item length determines the number of bytes moved.

Assume the following instruction is being executed:

OC	OP1
LR3	R
175	005

- where R specifies an item in the SDAT table.



When the above instruction is executed, item 5 (Record Descriptor) in the SDAT table is moved right-aligned into AR3 of the PCB. Since the OP1 item in the above instruction specified AR0 as the active record, only 4 bytes are moved.

Assume the next instruction to be executed is also an LR3 instruction and the following conditions exist:

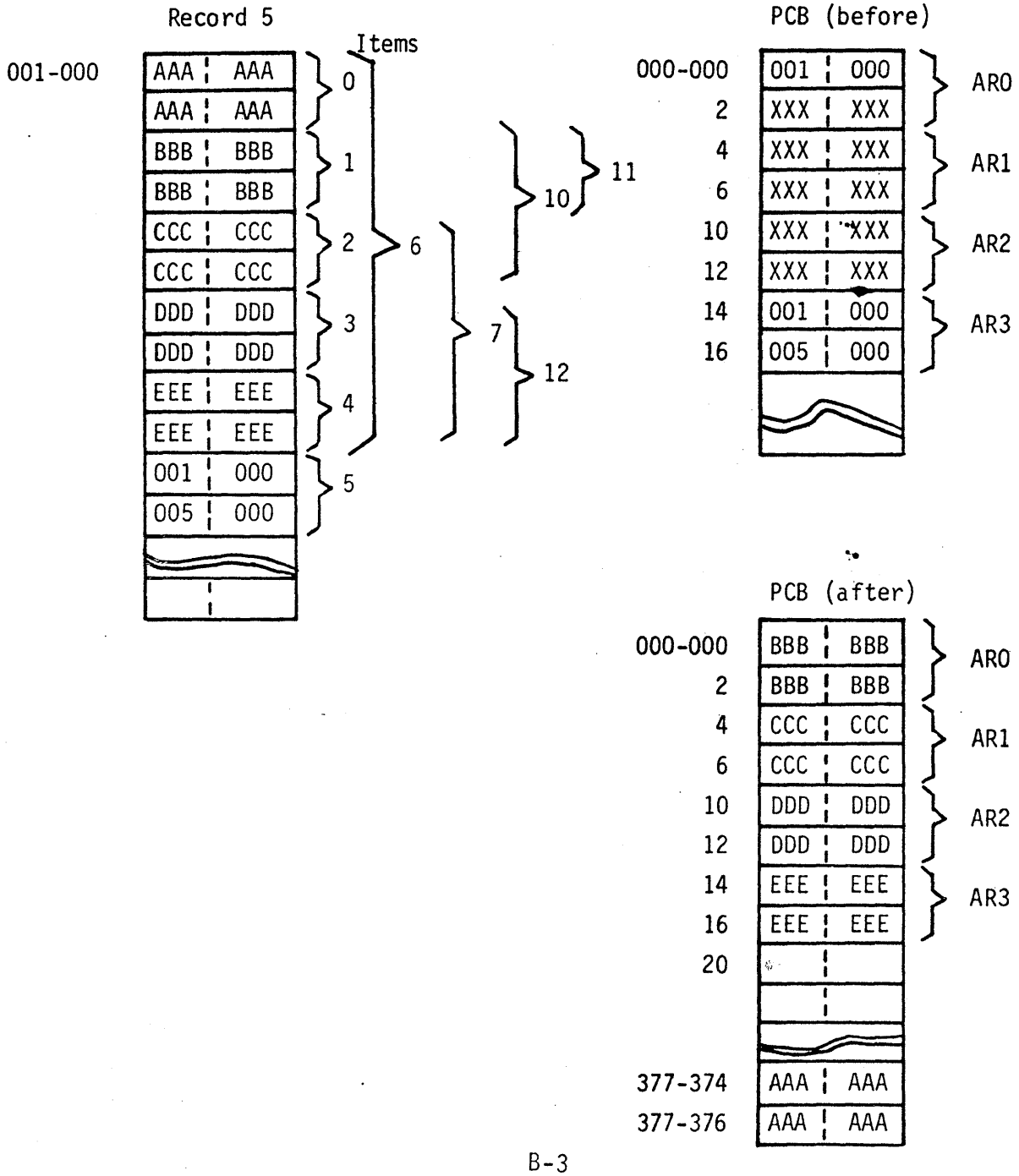
- The record described by item 5 in the SDAT overlays the SDAT itself.
- The IDT for record 5 is located at address 005-000 and contains the item descriptors as illustrated (item 0 through 12), and
- The programmer wishes to load AR0, AR1, AR2, and AR3 with different values (item 6 of record 5) using the one LR3 instruction.

OC OP1

LR3	AR/I
175	306

- where AR/I specifies Active Record 3 and item 6 of record 5.

When the instruction is executed, item 6 of record 5 is moved right-aligned into the PCB until address 000-000 is filled, and the remaining four bytes (AAA, AAA,AAA,AAA) are moved to the last four bytes of memory (377-374 through 377-377 in a 65K memory).



In general, the above philosophy allows loading one or more AR's with one Load Record instruction as follows.

Instruction	Record Item Length (bytes)	AR's Loaded
LR3	4	AR3
LR3	10_8	AR3, AR2
LR3	14_8	AR3, AR2, AR1
LR3	20_8	AR3, AR2, AR1, AR0
LR2	4	AR2
LR2	10_8	AR2, AR1
LR2	14_8	AR2, AR1, AR0
LR1	4	AR1
LR1	10_8	AR1, AR0
LSP	4	AR0

The above discussion was directed toward the processor Worker State. The same philosophy applies to the Exec State.

The above discussion also illustrated the SDAT table with a record overlay. This need not be, it may be a different record or records as long as the overall programming architecture of the processor is followed.

APPENDIX C - EBCDIC CODE

Char.	Bit 7 6 5 4 3 2 1 0	Octal Code	Char.	Bit 7 6 5 4 3 2 1 0	Octal Code
A	1 1 0 0 0 0 0 1	301	6	1 1 1 1 0 1 1 0	366
B	1 1 0 0 0 0 1 0	302	7	1 1 1 1 0 1 1 1	367
C	1 1 0 0 0 0 1 1	303	8	1 1 1 1 1 0 0 0	370
D	1 1 0 0 0 1 0 0	304	9	1 1 1 1 1 0 0 1	371
E	1 1 0 0 0 1 0 1	305	Space	0 1 0 0 0 0 0 0	100
F	1 1 0 0 0 1 1 0	306	¢ ①	0 1 0 0 1 0 1 0	112
G	1 1 0 0 0 1 1 1	307	.	0 1 0 0 1 0 1 1	113
H	1 1 0 0 1 0 0 0	310	<	0 1 0 0 1 1 0 0	114
I	1 1 0 0 1 0 0 1	311	(	0 1 0 0 1 1 0 1	115
J	1 1 0 1 0 0 0 1	321	+	0 1 0 0 1 1 1 0	116
K	1 1 0 1 0 0 1 0	322	| ②	0 1 0 0 1 1 1 1	117
L	1 1 0 1 0 0 1 1	323	&	0 1 0 1 0 0 0 0	120
M	1 1 0 1 0 1 0 0	324	! ③	0 1 0 1 1 0 1 0	132
N	1 1 0 1 0 1 0 1	325	$	0 1 0 1 1 0 1 1	133
O	1 1 0 1 0 1 1 0	326	*	0 1 0 1 1 1 0 0	134
P	1 1 0 1 0 1 1 1	327	)	0 1 0 1 1 1 0 1	135
Q	1 1 0 1 1 0 0 0	330	;	0 1 0 1 1 1 1 0	136
R	1 1 0 1 1 0 0 1	331	¬ ④	0 1 0 1 1 1 1 1	137
S	1 1 1 0 0 0 1 0	342	-	0 1 1 0 0 0 0 0	140
T	1 1 1 0 0 0 1 1	343	/	0 1 1 0 0 0 0 1	141
U	1 1 1 0 0 1 0 0	344	'	0 1 1 0 1 0 1 1	153
V	1 1 1 0 0 1 0 1	345	%	0 1 1 0 1 1 0 0	154
W	1 1 1 0 0 1 1 0	346	—	0 1 1 0 1 1 0 1	155
X	1 1 1 0 0 1 1 1	347	>	0 1 1 0 1 1 1 0	156
Y	1 1 1 0 1 0 0 0	350	?	0 1 1 0 1 1 1 1	157
Z	1 1 1 0 1 0 0 1	351	:	0 1 1 1 1 0 1 0	172
0	1 1 1 1 0 0 0 0	360	#	0 1 1 1 1 0 1 1	173
1	1 1 1 1 0 0 0 1	361	@	0 1 1 1 1 1 0 0	174
2	1 1 1 1 0 0 1 0	362	'	0 1 1 1 1 1 0 1	175
3	1 1 1 1 0 0 1 1	363	=	0 1 1 1 1 1 1 0	176
4	1 1 1 1 0 1 0 0	364	"	0 1 1 1 1 1 1 1	177
5	1 1 1 1 0 1 0 1	365	≠	1 1 1 0 0 0 0 0	340
			Null	0 0 0 0 0 0 0 0	000
+ Sign	1 1 1 1 x x x x		Substitute Codes		
- Sign	1 1 0 1 x x x x		①] ② ! ③ [④ ^		

APPENDIX D

TALLY COUNTER -

INSTRUCTION EXECUTION

TALLY COUNTER -
INSTRUCTION EXECUTION

	Op Code		Instruction	Tally Counter	
	Octal	Mnem.		Clear & Count	Clear Only
DATA MOVE	000	M	Move, Left-Align, No Fill	X	
	001	MR	Move, Right-Align, No Fill	X	
	*003	MED	Move, Edit		X
	004	MF	Move, Left-Align, Fill	X	
	005	MRF	Move, Right-Align, Fill	X	
	006	MJ	Move, Left-Justify Fill	X	
	007	MRJ	Move, Right-Justify Fill	X	
	140	TRL	Translate Code	X	
	141	ML	Move Literal		X
	*050	MPK	Pack	X	
	*052	MUP	Unpack	X	
BRANCHING	021	GGT	GOTO Greater Than	-	-
	022	GLT	GOTO Less Than	-	-
	023	GNE	GOTO Not Equal	-	-
	024	GE	GOTO Equal	-	-
	025	GNL	GOTO If Not Less Than	-	-
	026	GNG	GOTO If Not Greater Than	-	-
	027	G	GOTO Unconditionally	-	-
	030	GD	GOTO On Designators	-	-
	031	GS	GOTO On Switches	-	-

* 502 Mode Only

TALLY COUNTER -
INSTRUCTION EXECUTION
(continued)

	Op Code		Instruction	Tally Counter	
	Octal	Mnem.		Clear & Count	Clear Only
BRANCHING	*061	GBG	GOTO Binary Greater Than	-	-
	*062	GBL	GOTO Binary Less Than	-	-
	*063	GBN	GOTO Binary Non Zero	-	-
	*064	GBZ	GOTO Binary Zero	-	-
	*065	GGBE	GOTO Binary Zero	-	-
	*066	GLBE	GOTO Binary Zero	-	-
	*071	GDG	GOTO Decimal Greater Than	-	-
	*072	GDL	GOTO Decimal Less Than	-	-
	*073	GDN	GOTO Decimal Non Zero	-	-
	*074	GDZ	GOTO Decimal Zero	-	-
	*075	GGDE	GOTO Decimal Zero	-	-
	*076	GLDE	GOTO Decimal Zero	-	-
	*170	GCT	GOTO On Count	-	-
	*172	GTB	GOTO Table (Indirect Branch)	-	-
	*173	GRT	Return GOTO	-	-
	*176	GSB	GOTO Subroutine	-	-
COMPARE	044	CB	Compare Binary	X	
	046	CB	Compare Decimal	X	
	142	CAN	Compare Alphanumerics	X	
	144	CL	Compare Literal	X	

* 502 Mode Only

TALLY COUNTER -
INSTRUCTION EXECUTION
(continued)

	Op Code		Instruction	Tally Counter	
	Octal	Mnem.		Clear & Count	Clear Only
TEST	040	TBS	Test Binary Sign		X
	042	TDS	Test Decimal Sign		X
	150	TI	Test Item	X	
	151	TL	Test Literal	X	
	152	TM	Test Mask	X	
	*153	TIM	Test Item Mask	X	
INPUT/OUTPUT	100	INS	Special In	-	-
	104	EF	External Function On Channel	-	-
	105	OTS	Special Out	-	-
	*106	EFS	External Function Special	-	-
	107	GA	GOTO On Channel Active	-	-
	110	STC	Store Channel Control Register	-	-
	*111	STR	Store Channel Reverse	-	-
	*112	INR	Initiate Input Reverse	-	-
	114	IN	Initiate Input On Channel	-	-
	115	OUT	Initiate Output On Channel	-	-
	*116	OTR	Initiate Output Reverse	-	-
GENERAL PURPOSE	000	RN	Rename	X	
	020	NOP	No Operation	-	-
	*124	STD	Store Designators	-	-

* 502 Mode Only

TALLY COUNTER -
INSTRUCTION EXECUTION
(continued)

	Op Code		Instruction	Tally Counter	
	Octal	Mnem.		Clear & Count	Clear Only
SEQUENTIAL EDITING	014	CP	Compress Item, Left-Align, Fill	X	
	015	CPR	Compress Item, Right-Align, Fill	X	
	120	APR	Append, Right-Eliminate	X	
	121	APA	Append, Advance	X	
	122	APE	Append, Left-Eliminate	X	
	130	EXV	Extract Variable Length Item, Fill	X	
	131	EXP	Extract Previous Item	X	
	132	EX	Extract Item	X	
	133	EXA	Extract Item, Advance	X	
LOGICAL	*160	X	Exclusive OR	-	-
	*162	RCK	Longitudinal Redundancy Check	-	-
	*164	O	Logical OR	-	-
	*166	N	Logical AND	-	-
INTERRUPT	*113	GSI	GOTO On Service Request	-	-
	*117	GCI	GOTO On Channel Interrupt	-	-
	*154	SWS	Swap States	-	-
	*155	SIL	Set Interrupt Lockout	-	-
	*157	CIL	Clear Interrupt Lockout	-	-
	*174	IM	Interrupt Mask	-	-
	*177	GIR	Interrupt Branch GOTO	-	-

* 502 Mode Only

TALLY COUNTER -
INSTRUCTION EXECUTION
(continued)

	Op Code		Instruction	Tally Counter	
	Octal	Mnem.		Clear & Count	Clear Only
GENERAL PURPOSE	*126	LD	Load Designators	-	-
	*134	STT	Store Tally Counter	-	-
	*136	LT	Load Tally Counter	-	-
	143	H	Halt	-	-
	146	SDI	Set Display Indicators	X	-
	*147	GAP	No Operation Leave Gap	-	-
	156	CDI	Clear Display Indicators	X	
	*161	LSP	Load Storage Descriptor Pointer	-	-
BINARY ARITHMETIC	*165	LR1	Load Active Record 1	-	-
	*171	LR2	Load Active Record 2	-	-
	*175	LR3	Load Active Record 3	-	-
	041	AB	Add Binary	X	
	045	SB	Subtract Binary	X	
	051	ALB	Add Literal Binary	X	
	055	SLB	Subtract Literal Binary	X	
DECIMAL ARITHMETIC	043	A	Add Decimal	X	
	047	S	Subtract Decimal	X	
	053	AL	Add Literal Decimal	X	
	057	SL	Subtract Literal Decimal	X	

* 502 Mode Only

TALLY COUNTER -
INSTRUCTION EXECUTION
(continued)

	Op Code		Instruction	Tally Counter	
	Octal	Mnem.		Clear & Count	Clear Only
EXTERNAL EXECUTE INSTRUCTIONS OC - 145	*004	LC	Load Delta Clock	X	
	*014	SEE	Store External Instruction Error		X
	*015	SCE	Store Channel Parity Error		X
	*020	MB	Multiply Binary	X	
	*021	MLB	Multiply Literal Binary	X	
	*022	DB	Divide Binary	X	
	*023	DLB	Divide Literal Binary	X	
	*024	MD	Multiply Decimal	X	
	*025	MLD	Multiply Literal Decimal	X	
	*026	DD	Divide Decimal	X	
	*027	DLD	Divide Literal Decimal	X	
	*030	BTD	Binary to Decimal	X	
	*031	DTB	Decimal to Binary	X	
	*034	SDR	Store Decimal Remainder		X
	*035	SBR	Store Binary Remainder		X

* 502 Mode Only

APPENDIX E

INSTRUCTION EXECUTION TIMES AND PROCESSOR MODELS

The instruction execution times vary, depending upon the instruction and the number of data bytes manipulated. Table E-1 gives the formulae for calculating the execution time of the instructions when a SUDA processor is used.

The following symbology is used for the table:

- A = Number of bytes in the space described by operand A.
- B = Number of bytes in the space described by operand B.
- C = Number of bytes in the space described by operand C.
- S(p,q) = Number of bytes in the shorter of the spaces described by operands "p" and "q".
- L(p,q) = Number of bytes in the longer of the spaces described by operands "p" and "q".
- N = Number of bytes to be eliminated in operand A (in APR, APE instructions).
- X = Number of bytes compared or tested (in CAN, CL, TI, TL instructions)

  In CAN: If equal, X=B;
  If unequal, X=the number of bytes in 'B' compared until inequality is established.

  In TI or TL: If equal, X=the number of bytes in 'A' compared until value is found.
  If unequal, X=A.

  In CL: If equal, X=A.
  If unequal, X=the number of bytes in 'A' compared until inequality is established.

Table E-1. Formulae for Execution Times of the 501A Processor

OP CODE		
OCT	MNE	EXECUTION TIMES (in microseconds)
		DATA MOVE
000	M	30 + 4*S(A,B)
001	MR	30 + 4*S(A,B)
004	MF	32 + 2*S(A,B) + 2*B
005	MRF	32 + 2*S(A,B) + 2*B
006	MJ	32 + 2*S(A,B) + 2*B
007	MRJ	32 + 2*S(A,B) + 2*B
140	TRL	44 + 8*S(A,B)
141	ML	18 + 2*A
		BRANCHING
020	NOP	6 (no jump possible)
021	GGT	10 if jump, 6 if no jump
022	GLT	10 if jump, 6 if no jump
023	GNE	10 if jump, 6 if no jump
024	GE	10 if jump, 6 if no jump
025	GNL	10 if jump, 6 if no jump
026	GNG	10 if jump, 6 if no jump
027	G	10
030	GD	1s if jump, 8 if no jump
031	GS	12 if jump, 8 if no jump
		COMPARE
044	CB	30 + 2*S(A,B) + 4*L(A,B)
046	CD	If signs are alike, 30 + 2*S(A,B) + 4*L(A,B)
		If signs are unlike, 36
142	CAN	30 + 4*X
144	CL	18 + 2*X

Table E-1. Formulae for Execution Times of the 501A Processor (continued)

OP CODE		EXECUTION TIMES (in microseconds)
OCT	MNE	
		TEST
040	TBS	18
042	TDS	18
150	T1	32 + 2*X
151	TL	18 + 2*X
152	TM	20
		INPUT/OUTPUT
100	INS	32 + 2*B
104	EF	46 + 8*B
105	OTS	32 + 2*B
107	GA	26 if jump, 22 if no jump
110	STC	38
114	IN	36
115	OUT	36
		GENERAL
000	RN	46
143	H	2
146	SD1	18
156	CD1	18
161	LSP	32
165	LR1	32
171	LR2	32
175	LR3	32

Table E-1. Formulae for Execution Times of the 501A Processor (continued)

OP CODE		
OCT	MNE	EXECUTION TIMES (in microseconds)
		BINARY ARITHMETIC
041	AB	44 + 2*S(A,C) + 2*S(B,C) + 2*C
045	SB	44 + 2*S(A,C) + 2*S(B,C) + 2*C
051	ALB	36 + 2*A + 2*B
055	SLB	36 + 2*A + 2*B
		DECIMAL ARITHMETIC
043	A	If signs are alike, 44 + 2*S(A,C) + 2*S(B,C) + 2*C
		If signs are unlike, 46 + 2*S(A,C) + 2*S(B,C) + 6*C
047	S	If signs are unlike, 46 + 2*S(A,C) + 2*S(B,C) + 6*C
053	AL	If signs are alike, 36 + 2*A + 2*B
		If signs are unlike, 38 + 2*A + 6*B
057	SL	If signs are unlike, 38 + 2*A + 6*B
		SEQUENTIAL EDITING
014	CP	34 + 2*S(A,B) + 2*B
015	CPR	34 + 2*S(A,B) + 2*B
120	APR	If all source bytes are eliminated, 36 + 2*A
		If designators are set, 38 + 4*S(A,B) - 2*N
		If designators are not set, 44 + 4*S(A,B) - 2*N
121	APA	If designators are set, 34 + 4*S(A,B)
		If designators are not set, 40 + 4*S(A,B)
122	APE	If all source bytes are eliminated, 36 + 2*A
		If designators are set, 36 + 4*S(A,B) - 2*N
		If designators are not set, 42 + 4*S(A,B) - 2*N
130	EXV	If no designators are set, 46 + 2*S(A,B) + 2*B
		If 'EQUAL' is set, 40 + 2*B
		If 'ABN EDIT' is set, 40 + 4*B
		If 'EQUAL' and 'ABN EDIT' are set, 38 + 4*S(A,B)
131	EXP	42 + 4*S(A,B)
132	EX	34 + 4*S(A,B)
133	EXA	40 + 4*S(A,B)

Table E-2 gives the formulae for calculating the execution time of the instructions when a 502, 502A or 502B is operating in the processor. 1 - microsecond cycle time. When the 502, 502A or 502B processor is operating in the 2 - microsecond cycle time, the calculated time are doubled. The same symbology is used as in table E-1.

Table E-2. Formulae for Execution Times of 502, 502A & 502B Processors

	Op Code		EXECUTION TIMES (in microseconds)
	Octal	Mnem.	
DATA MOVE	000	M	7 + 2 S(A,B)
	001	MR	7 + 2 S(A,B)
	*003	MED	12 + N + 2(X-N) + M + C
	004	MF	8 + S(A,B) + B
	005	MRF	8 + S(A,B) + B
	006	MJ	8 + S(A,B) + B
	007	MRJ	8 + S(A,B) + B
	*050	MPK	7 + 2 S(A,B) + C
	*052	MUP	7 + S(A,B) + 2 C
	*140	TRL	10 + 4 S(A,B)
	141	ML	5 + A
BRANCHING	020	NOP	1
	021	GGT	5 if jump, 3 if no jump
	022	GLT	5 if jump, 3 if no jump
	023	GNE	5 if jump, 3 if no jump
	024	GE	5 if jump, 3 if no jump
	025	GNL	5 if jump, 3 if no jump
	026	GNG	5 if jump, 3 if no jump
	027	G	5 if jump, 3 if no jump
	030	GD	6 if jump, 4 if no jump
	031	GS	6 if jump, 4 if no jump
	*061	GBG	8 + A if jump, 6 + A if no jump
	*062	GBL	8 + A if jump, 6 + A if no jump
	*063	GNB	8 + A if jump, 6 + A if no jump
	*064	GBZ	8 + A if jump, 6 + A if no jump
	*065	GGBE	8 + A if jump, 6 + A if no jump
	*066	GLBE	8 + A if jump, 6 + A if no jump
	*071	GDG	8 + A if jump, 6 + A if no jump
	*072	GDL	8 + A if jump, 6 + A if no jump
	*073	GDN	8 + A if jump, 6 + A if no jump
	*074	GDZ	8 + A if jump 6 + A if no jump
	*075	GGDE	8 + A if jump, 6 + A if no jump
	*076	GLDE	8 + A if jump, 6 + A if no jump
	*170	GCT	10 if jump, 8 if no jump
	*172	GTB	14 if jump, 9 if no jump
	*173	GRT	12
	*176	GSB	16

* 502 Mode Only

Table E-2. Formulae for Execution Times of 502, 502A and 502B Processors (continued)

	Op Code		EXECUTION TIMES (in microseconds)
	Octal	Mnem.	
COMPARE	044	CB	7 + S(A,B) + 2 L(A,B)
	046	CD	If signs are alike, 7 + S(A,B) + 2 L(A,B) If signs are unlike, 10
	142	CAN	7 + 2 X
	144	CL	5 + X
TEST	040	TBS	5
	042	TDS	5
	150	TI	8 + X
	151	TL	5 + X
	152	TM	6
	*153	TIM	8 + X
INPUT/OUTPUT	100	INS	8 + B
	104	EF	11 + 4 B
	105	OTS	8 + B
	106	EFS	8 + B
	107	GA	7 if jump, 9 if no jump
	110	STC	10
	*111	STR	10
	*112	INR	10
	114	IN	10
	115	OUT	10
	*116	OTR	10
GENERAL PURPOSE	000	RN	15
	*124	STD	5
	*126	LD	5
	*134	STT	5
	*136	LT	5
	143	H	1
	146	SDI	5
	*147	GAP	1
	156	CDI	5
	*161	LSP	11
	*165	LR1	11
	*171	LR2	11
	*175	LR3	11

* 502 Mode Only

Table E-2. Formulae for Execution Times of 502, 502A and 502B Processors (continued)

	Op Code		EXECUTION TIMES (in microseconds)
	Octal	Mnem.	
LOGICAL	*160	X	10 + 3 S(A,B,C)
	*162	RCK	8 + S(A,B)
	*164	O	10 + 3 S(A,B,C)
	*166	N	10 + 3 S(A,B,C)
BINARY ARITHMETIC	041	AB	10 + S(A,B) + L(A,B) + C
	045	SB	10 + S(A,B) + L(A,B) + C
	051	ALB	8 + A + B
	055	SLB	8 + A + B
DECIMAL ARITHMETIC	043	A	If signs are alike, 10 + S(A,B) + L(A,B) + C If signs are unlike, 11 + S(A,B) + L(A,B) + 3 C
	047	S	If signs are unlike, 11 + S(A,B) + L(A,B) + 3 C
	053	AL	If signs are alike, 8 + A + B If signs are unlike, 9 + A + 3 B
	057	SL	If signs are unlike, 9 + A + 3 B
SEQUENTIAL EDITING	014	CP	9 + S(A,B) + B
	015	CPR	9 + S(A,B) + B
	120	APR	If all source bytes are eliminated, 10 + A If designators are set, 11 + 2 S(A,B) - N If designators are not set, 14 + 2 S(A,B) - N
	121	APA	If designators are set, 9 + 2 S(A,B) If designators are not set, 12 + 2 S(A,B)
	122	APE	If all source bytes are eliminated, 10 + A If designators are set, 10 + 2 S(A,B) - N If designators are not set, 13 + 2 S(A,B) - N
	130	EXV	If no designators are set, 15 + S(A,B) + B If 'EQUAL' is set, 12 + B If 'ABN EDIT' is set, 12 + B If 'EQUAL' and 'ABN EDIT' is set, 11 + 2 S(A,B)
	131	EXP	13 + 2 S(A,B)
	132	EX	9 + 2 S(A,B)
	133	EXA	12 + 2 S(A,B)

* 502 Mode Only

Table E-2. Formulae for Execution Times of 502, 502A & 502B Processors (continued

	Op Code		
	Octal	Mnem.	EXECUTION TIMES (in microseconds)
INTERRUPT	*113	GSI	7 if jump, 9 if no jump
	*117	GCI	7 if jump, 9 if no jump
	*154	SWS	34
	*155	SIL	2
	*157	CIL	2
	*174	IM	5
	*177	GIR	15
EXTERNAL EXECUTE INSTRUCTIONS OC - 145	*004	LC	12
	*014	SEE	24
	*015	SCE	11
	*020	MB	31 + 2 L(A,B) + 2 S(A,B) + 2 C
	*021	MLB	33 + 2 L(A,B) + 2 B
	*022	DB	30 + 2 [L(A,B) + S(A,B) + C]
	*023	DLB	32 + 2 [L(A,B) + B]
	*024	MD	79 + 2 [L(A,B) + S(A,B) + C
	*025	MLD	81 + 2 L(A,B) + B
	*026	DD	54 + 2 [L(A,B) + S(A,B) + C
	*027	DLD	56 + 2 L(A,B) + B
	*030	BTD	58 + 2 L(A,B) + B
	*031	DTB	10 + 2 [L(A,B) + B]
	*034	SDR	39 + A
	*035	SBR	15 + 2 A

* 501 Mode Only

Table E-3 indicates the processor that can execute each instruction. The following processors are listed.

* 501
* 502
* 502A
* 502B

Currently, the correlation of System to Processor is as follows:

Systems	Processor
2404	502
2405	501A or 502
2408	502A or 502B
2409-I	502B

An "X" in Table E-3 indicates that the processor can execute the instruction.

Table E-3. Instruction Set and Processor Model

	OP Code			Processor			
	OCT	MNE	Instruction	501A	502	502A	502B
DATA MOVE	000	M	Move, Left-Align, No Fill	X	X	X	X
	001	MR	Move, Right-Align, No Fill	X	X	X	X
	003	MED	Move, Edit	NO	X	X	X
	004	MF	Move, Lift-Align, Fill	X	X	X	X
	005	MRF	Move, Right-Align, Fill	X	X	X	X
	006	MJ	Move, Left-Justified, Fill	X	X	X	X
	007	MRJ	Move, Right-Justified, Fill	X	X	X	X
	050	MPK	Move, Pack	NO	X	X	X
	052	MUP	Move, Unpack	NO	X	X	X
	140	TRL	Translate Code	X	X	X	X
	141	ML	Move Literal	X	X	X	X

Table E-3. Instruction Set and Processor Model
(continued)

	OP Code			Processor			
	OCT	MNE	Instruction	501A	502	502A	502B
BRANCHING	020	NOP	No Operation	X	X	X	X
	021	GGT	GOTO Greater Than	X	X	X	X
	022	GLT	GOTO Less Than	X	X	X	X
	023	GNE	GOTO Not Equal	X	X	X	X
	024	GE	GOTO Equal	X	X	X	X
	025	GNL	GOTO Not Less Than	X	X	X	X
	026	GNG	GOTO Not Greater Than	X	X	X	X
	027	G	GOTO Unconditionally	X	X	X	X
	030	GD	GOTO on Designators	X	X	X	X
	031	GS	GOTO on Switches	X	X	X	X
	061	GBG	GOTO Binary Greater Than Zero	NO	X	X	X
	062	GBL	GOTO Binary Less Than Zero	NO	X	X	X
	063	GBN	GOTO Binary Non Zero	NO	X	X	X
	064	GBZ	GOTO Binary Zero	NO	X	X	X
	065	GGBE	GOTO Binary Equal/Greater Than Zero	NO	X	X	X
	066	GLBE	GOTO Binary Equal/Less Than Zero	NO	X	X	X
	071	GDG	GOTO Decimal Greater Than Zero	NO	X	X	X
	072	GDL	GOTO Decimal Less Than Zero	NO	X	X	X
	073	GDN	GOTO Decimal Non Zero	NO	X	X	X
	074	GDZ	GOTO Decimal Zero	NO	X	X	X
	075	GGDE	GOTO Decimal Equal/Greater Than Zero	NO	X	X	X
	170	GCT	GOTO on Count	NO	X	X	X
	172	GTB	GOTO Table	NO	X	X	X
	173	GRT	Return GOTO	NO	X	X	X
	176	GSB	GOTO Subroutine	NO	X	X	X

Table E-3. Instruction Set and Processor Model
(continued)

	OP Code			Processor			
	OCT	MNE	Instruction	501A	502	502A	502B
COMPARE	044	CB	Compare Binary	X	X	X	X
	046	CD	Compare Decimal	X	X	X	X
	142	CAN	Compare Alphanumerics	X	X	X	X
	144	CL	Compare Literal	X	X	X	X
TEST	040	TBS	Test Binary Sign	X	X	X	X
	042	TDS	Test Decimal Sign	X	X	X	X
	150	TI	Test Item	X	X	X	X
	151	TL	Test Literal	X	X	X	X
	152	TM	Test Mask	X	X	X	X
	153	TIM	Test Item Mask	NO	X	X	X
INPUT/OUTPUT	100	INS	Special In	X	X	X	X
	104	EF	External Function on Chan.	X	X	X	X
	105	OTS	Special Out	X	X	X	X
	106	EFS	External Function Special	NO	X	X	X
	107	GA	GOTO on Channel Active	X	X	X	X
	110	STC	Store Chan. Control Register	X	X	X	X
	111	STR	Store Channel Reverse	NO	X	X	X
	112	INR	Initiate Input Reverse	NO	X	X	X
	114	IN	Initiate Input on Chan.	X	X	X	X
	115	OUT	Initiate Output on Chan.	X	X	X	X
	116	OTR	Initiate Output Reverse	NO	X	X	X
	145	-	Execute External Instruction	NO	X	X	X
GENERAL PURPOSE	000	RN	Rename	X	X	X	X
	124	SID	Store Designators	NO	X	X	X
	126	LD	Load Designators	NO	X	X	X
	134	STT	Store Tally Counter	NO	X	X	X
	136	LT	Load Tally Counter	NO	X	X	X
	143	H	Halt	X	X	X	X
	146	SDI	Set Display Indicators	X	X	X	X
	147	GAP	No Operation (1 Byte)	NO	X	X	X
	156	CDI	Clear Display Indicators	X	X	X	X
	161	LSP	Load Storage Desc. Pointer	X	X	X	X
	165	LRI	Load Active Record 1	X	X	X	X
	171	LR2	Load Active Record 2	X	X	X	X
	175	LR3	Load Active Record 3	X	X	X	X

Table E-3. Instruction Set and Processor Model
(continued)

	OP Code			Processor			
	OCT	MNE	Instruction	501A	502	502A	502B
LOGICAL	*160	X	Exclusive OR	NO	X	X	X
	*162	RCK	Longitudinal Redundancy Check	NO	X	X	X
	*164	O	Logical OR	NO	X	X	X
	*166	N	Logical AND	NO	X	X	X
BINARY ARITHMETIC	041	AB	Add Binary	X	X	X	X
	045	SB	Subtract Binary	X	X	X	X
	055	SLB	Subtract Literal Binary	X	X	X	X
DECIMAL ARITHMETIC	043	A	Add Decimal	X	X	X	X
	047	S	Subtract Decimal	X	X	X	X
	053	AL	Add Literal Decimal	X	X	X	X
	057	SL	Subtract Literal Decimal	X	X	X	X
SEQUENTIAL EDITING	014	CP	Compress Item, Left Align, Fill	X	X	X	X
	015	CPR	Compress Item, Right Align, Fill	X	X	X	X
	120	APR	Append, Right Eliminate	X	X	X	X
	121	APA	Append, Advance	X	X	X	X
	122	APE	Append, Left Eliminate	X	X	X	X
	130	EXV	Extract Variable, Fill	X	X	X	X
	131	EXP	Extract Previous Item	X	X	X	X
	132	EX	Extract Item	X	X	X	X
	133	EXA	Extract Item, Advance	X	X	X	X

Table E-3. Instruction Set and Processor Model
(continued)

	OP Code			Processor			
	OCT	MNE	Instruction	501A	502	502A	502B
INTERRUPT	113	GSI	GOTO on Service Request	NO	X	X	X
	117	GCI	GOTO on Channel Interrupt	NO	X	X	X
	154	SWS	Swap States	NO	X	X	X
	155	SIL	Set Interrupt Lockout	NO	X	X	X
	157	CIL	Clear Interrupt Lockout	NO	X	X	X
	174	IM	Interrupt Mask	NO	X	X	X
	177	GIR	Interrupt Return GOTO	NO	X	X	X
EXTERNAL EXECUTE	004	LC	Load Delta Clock	NO	NO	NO	X
	014	SCE	Store External Instruction Error	NO	NO	X	X
	015	SLE	Store Channel Parity Error	NO	NO	X	X
	020	MB	Multiply Binary	NO	NO	NO	X
	021	MLB	Multiply Literal Binary	NO	NO	NO	X
	022	DB	Divide Binary	NO	NO	NO	X
	023	DLB	Divide Literal Binary	NO	NO	NO	X
	024	MD	Multiply Decimal	NO	NO	NO	X
	025	MLD	Multiply Literal Decimal	NO	NO	NO	X
	026	LD	Divide Decimal	NO	NO	NO	X
	027	DLD	Divide Literal Decimal	NO	NO	NO	X
	030	BTD	Binary to Decimal	NO	NO	NO	X
	031	DTB	Decimal to Binary	NO	NO	NO	X
	034	SDR	Store Decimal Remainder	NO	NO	NO	X
	035	SBR	Store Binary Remainder	NO	NO	NO	X

APPENDIX F

OCTAL NOTATION RULES

Octal notation is a convenient shorthand method of writing pure binary numbers. In programming it is used to represent such binary values as memory addresses, I/O control characters, constants, etc.

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 F-1.

Table F-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

OCTAL/DECIMAL CONVERSION PROCEDURE (See Table F-2)

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

Table F-2. Decimal/Octal Conversion Table

LOW-ORDER OCTAL DIGIT	DECIMAL INCREMENT																								LOW-ORDER OCTAL DIGIT
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
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
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
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
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
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
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
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

DECIMAL BASE NO.																										DECIMAL BASE NO.
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
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	61	0200
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	112	0400
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	143	0600
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	174	0800
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	225	1000
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	256	1200
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	307	1400
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	340	1600
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	371	1800
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	422	2000
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	453	2200
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	504	2400
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	535	2600
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	566	2800
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	617	3000
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	650	3200
3400	651	652	653	654	655	656	657	660	661	662	663	664	665	666	667	670	671	672	673	674	675	676	677	700	701	3400
3600	702	703	704	705	706	707	710	711	712	713	714	715	716	717	720	721	722	723	724	725	726	727	730	731	732	3600
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	763	3800
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	1014	4000
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	1045	4200
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	1076	4400
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	1127	4600
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	1160	4800
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	1211	5000
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	1242	5200
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	1273	5400
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	1324	5600
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	1355	5800
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	1406	6000
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	1437	6200
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	1470	6400
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	1521	6600
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	1552	6800
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	1603	7000
7200	1604	1605	1606	1607	1610	1611	1612	1613	1614	1615	1616	1617	1620	1621	1622	1623	1624	1625	1626	1627	1630	1631	1632	1633	1634	7200
7400	1635	1636	1637	1640	1641	1642	1643	1644	1645	1646	1647	1650	1651	1652	1653	1654	1655	1656	1657	1660	1661	1662	1663	1664	1665	7400
7600	1666	1667	1670	1671	1672	1673	1674	1675	1676	1677	1700	1701	1702	1703	1704	1705	1706	1707	1710	1711	1712	1713	1714	1715	1716	7600
7800	1717	1720	1721	1722	1723	1724	1725	1726	1727	1730	1731	1732	1733	1734	1735	1736	1737	1740	1741	1742	1743	1744	1745	1746	1747	7800
8000	1750	1751	1752	1753	1754	1755	1756	1757	1760	1761	1762	1763	1764	1765	1766	1767	1770	1771	1772	1773	1774	1775	1776	1777	2000	8000
8200	2001	2002	2003	2004	2005	2006	2007	2010	2011	2012	2013	2014	2015	2016	2017	2020	2021	2022	2023	2024	2025	2026	2027	2030	2031	8200
8400	2032	2033	2034	2035	2036	2037	2040	2041	2042	2043	2044	2045	2046	2047	2050	2051	2052	2053	2054	2055	2056	2057	2060	2061	2062	8400
8600	2063	2064	2065	2066	2067	2070	2071	2072	2073	2074	2075	2076	2077	2100	2101	2102	2103	2104	2105	2106	2107	2110	2111	2112	2113	8600
8800	2114	2115	2116	2117	2120	2121	2122	2123	2124	2125	2126	2127	2130	2131	2132	2133	2134	2135	2136	2137	2140	2141	2142	2143	2144	8800
9000	2145	2146	2147	2150	2151	2152	2153	2154	2155	2156	2157	2160	2161	2162	2163	2164	2165	2166	2167	2170	2171	2172	2173	2174	2175	9000
9200	2176	2177	2200	2201	2202	2203	2204	2205	2206	2207	2210	2211	2212	2213	2214	2215	2216	2217	2220	2221	2222	2223	2224	2225	2226	9200
9400	2227	2230	2231	2232	2233	2234	2235	2236	2237	2240	2241	2242	2243	2244	2245	2246	2247	2250	2251	2252	2253	2254	2255	2256	2257	9400
9600	2260	2261	2262	2263	2264	2265	2266	2267	2270	2271	2272	2273	2274	2275	2276	2277	2300	2301	2302	2303	2304	2305	2306	2307	2310	9600
9800	2311	2312	2313	2314	2315	2316	2317	2320	2321	2322	2323	2324	2325	2326	2327	2330	2331	2332	2333	2334	2335	2336	2337	2340	2341	9800
10,000	2342	2343	2344	2345	2346	2347	2350	2351	2352	2353	2354	2355	2356	2357	2360	2361	2362	2363	2364	2365	2366	2367	2370	2371	2372	10,000
10,200	2373	2374	2375	2376	2377	2400	2401	2402	2403	2404	2405	2406	2407	2410	2411	2412	2413	2414	2415	2416	2417	2420	2421	2422	2423	10,200
10,400	2424	2425	2426	2427	2430	2431	2432	2433	2434	2435	2436	2437	2440	2441	2442	2443	2444	2445	2446	2447	2450	2451	2452	2453	2454	10,400
10,600	2455	2456	2457	2460	2461	2462	2463	2464	2465	2466	2467	2470	2471	2472	2473	2474	2475	2476	2477	2500	2501	2502	2503	2504	2505	10,600
10,800	2506	2507	2510	2511	2512	2513	2514	2515	2516	2517	2520	2521	2522	2523	2524	2525	2526	2527	2530	2531	2532	2533	2534	2535	2536	10,800
11,000	2537	2540	2541	2542	2543	2544	2545	2546	2547	2550	2551	2552	2553	2554	2555	2556	2557	2560	2561	2562	2563	2564	2565	2566	2567	11,000
11,200	2570	2571	2572	2573	2574	2575	2576	2577	2600	2601	2602	2603	2604	2605	2606	2607	2610	2611	2612	2613	2614	2615	2616	2617	2620	11,200
11,400	2621	2622	2623	2624	2625	2626	2627	2630	2631	2632	2633	2634	2635	2636	2637	2640	2641	2642	2643	2644	2645	2646	2647	2650	2651	11,400
11,600	2652	2653	2654	2655	2656	2657	2660	2661	2662	2663	2664	2665	2666	2667	2670	2671	2672	2673	2674	2675	2676	2677	2700	2701	2702	11,600
11,800	2703	2704	2705	2706	2707	2710	2711	2712	2713	2714	2715	2716	2717	2720	2721	2722	2723	2724	2725	2726	2727	2730	2731	2732	2733	11,800
12,000	2734	2735	2736	2737	2740	2741	2742	2743	2744	2745	2746	2747	2750	2751	2752	2753	2754	2755	2756	2757	2760	2761	2762	2763	2764	12,000
12,200	2765	2766	2767	2770	2771	2772	2773	2774	2775	2776	2777	3000	3001	3002	3003	3004	3005	3006	3007	3010	3011	3012	3013	3014	3015	12,200
12,400	3016	3017	3020	3021	3022	3023	3024	3025	3026	3027	3030	3031	3032	3033	3034	3035	3036	3037	3040	3041	3042	3043	3044	3045	3046	12,400
12,600	3047	3050	3051	3052	3053	3054	3055	3056	3057	3060	3061	3062	3063	3064	3065	3066	3067	3070	3071	3072	3073	3074	3075	3076	3077	12,600
12,800	3100	3101	3102	3103	3104	3105	3106	3107	3110	3111	3112	3113	3114	3115	3116	3117	3120	3121	3122	3123	3124	3125	3126	3127	3130	12,800
13,000	3131	3132	3133	3134	3135	3136	3137	3140	3141	3142	3143	3144	3145	3146	3147	3150	3151	3152	3153	3154	3155	3156	3157	3160	3161	13,000
13,200	3162	3163	3164	3165	3166	3167	3170	3171	3172	3173	3174	3175	3176	3177	3200	3201	3202	3203	3204	3205	3206	3207	3210	3211	3212	13,200
13,400	3213	3214	3215	3216	3217	3220	3221	3222	3223	3224	3225	3226	3227	3230	3231	3232	3233	3234	3235	3236	3237	3240	3241	3242	3243	13,400
13,600	3244	3245	3246	3247	3250	3251	3252	3253	3254	3255	3256	3257	3260	3261	3262	3263	3264	3265	3266	3267	3270	3271	3272	3273	3274	13,600
13,800	3275	3276	3277	3300	3301	3302	3303	3304	3305	3306	3307	3310	3311	3312	3313	3314	3315	3316	3317	3320	3321	3322	3323	3324	3325	13,800
14,000	3326	3327	3330	3331	3332	3333	3334	3335	3336	3337	3340	3341	3342	3343	3344	3345	3346	3347	3350	3351	3352	3353	3354	3355	3356	14,000
14,200	3357	3360	3361	3362	3363	3364	3365	3366	3367	3370	3371	3372	3373	3374	3375	3376	3377	3400	3401	3402	3403	3404	3405	3406	3407	14,200
14,400	3410	3411	3412	3413	3414	3415	3416	3417	3420	3421	3422	3423	3424	3425	3426	3427	3430	3431	3432	3433	3434	3435	3436	3437	3440	14,400
14,600	3441	3442	3443	3444	3445	3446	3447	3450	3451	3452	3453	3454	3455	3456	3457	3460	3461	3462	3463	3464	3465	3466	3467	3470	3471	14,600
14,800	3472	3473	3474	3475	3476	3477	3500	3501	3502	3503	3504	3505	3506	3507	3510	3511	3512	3513	3514	3515	3516	3517	3520	3521	3522	14,800
15,000	3523	3524	3525	3526	3527	3530	3531	3532	3533	3534	3535	3536	3537	3540	3541	3542	3543	3544	3545	3546	3547	3550	3551	3552	3553	15,000
15,200	3554	3555	3556	3557	3560	3561	3562	3563	3564	3565	3566	3567	3570	3571	3572	3573	3574	3575	3576	3577	3600	3601	3602	3603	3604	15,200
15,400	3605	3606	3607	3610	3611	3612	3613	3614	3615	3616	3617	3620	3621	3622	3623	3624	3625	3626	3627	3630	3631	3632	3633	3634	3635	15,400
15,600	3636	3637	3640	3641	3642	3643	3644	3645	3646	3647	3650	3651	3652	3653	3654	3655	3656	3657	3660	3661	3662	3663	3664	3665	3666	15,600
15,800	3667	3670	3671	3672	3673	3674	3675	3676	3677	3700	3701	3702	3703	3704	3705	3706	3707	3710	3711	3712	3713	3714	3715	3716	3717	15,800
16,000	3720	3721	3722	3723	3724	3725	3726	3727	3730	3731	3732	3733	3734	3735	3736	3737	3740	3741	3742	3743	3744	3745	3746	3747	3750	16,000
16,200	3751	3752	3753	3754	3755	3756	3757	3760	3761	3762	3763	3764	3765	3766	3767	3770	3771	3772	3773	3774	3775	3776	3777	4000	4001	16,200
16,400	4002	4003	4004	4005	4006	4007	4010	4011	4012	4013	4014	4015	4016	4017	4020	4021	4022	4023	4024	4025	4026	4027	4030	4031	4032	16,400

HIGH-ORDER OCTAL DIGITS

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.

TRI-OCTAL NOTATION

In SYSTEM 2400 programming concepts, "tri-octal" notation is used to describe the contents of bytes. Tri-octal is simply a slight variation on octal notation (base eight).

1. To describe in tri-octal notation the value of an eight-bit byte: the first two bits are given an octal value between Ø and 3; the next three bits are given an octal value between Ø and 7; the last three bits are given an octal value between Ø and 7.

	Ø 1	2 3 4	5 6 7
BITS:	Ø 1	1 1 Ø	Ø 1 Ø
TRI-OCTAL:	1	6	2

2. To describe the value of more than one byte, the area is first divided into bytes; then each byte is divided as above. In tri-octal notation, each group of three digits will describe eight bits (one byte).

3. To convert decimal to tri-octal.

   a. Write down the decimal number to be converted.
   b. Write below it the value divided by two; ignore any remainder.
   c. Write below this second value its "half", as before, and continue until the final value is "1".
   d. In a column to the right of this column of numbers, write a "1" beside each *odd* result, and a "Ø" beside each *even* result.
   e. The binary representation of the decimal number now appears as this *second* column, with the *least* significant bit at the *top*.
   f. Group the binary representation into bytes, then translate into tri-octal as above.

			BINARY	TRI-OCTAL
DECIMAL:	423	ODD	1	
	211	ODD	1	7
	105	ODD	1	
	52	EVEN	0	
	26	EVEN	0	4
	13	ODD	1	
	6	EVEN	0	2
	3	ODD	1	
	1	ODD	1	1 (next byte)

FINAL RESULT: 423 decimal = 001-247 tri-octal

4. To convert tri-octal to decimal:
   Values of each tri-octal column of a 2-byte number is shows:

	BYTE Ø			BYTE 1		
DECIMAL VALUE	16384	2048	256	64	8	1
EXAMPLE:	1	0	3	2	7	6
	16384	0	768	128	56	6
	(1x16384)	(0x2048)	(3x256)	(2x64)	(7x8)	(6x1)

   16384 + 0 + 768 + 128 + 56 + 6 = 17342 decimal

5. Notes about tri-octal:
   a. Most operands in SYSTEM 2400 instructions used the first two bits of the byte to refer to an "active record". In tri-octal notation, the first digit (of the three) describing the byte will be "Ø", "1", "2", or "3" - this is the "active record" being used.
   b. Numbers with a leading tri-octal digit of "2" of "3" are negative binary values, and will be so treated by the binary arithmetic instructions (ALB, SLB, AB, SB) and binary compare instructions (CB, TBS).

APPENDIX G

SNAP P ADAPTER

The SNAP P adapter is connected to DMA channel 2 and comprises two functions:

Capture P

Interrupt

CAPTURE P

Upon a command (via a Special Out Instruction), the Adapter captures the address of the next instruction to be executed. The address is held by the Adapter until called for by a Special In instruction. Interrupts are locked out by the Capture P function and remain locked out until enabled by an Enable Interrupts command (via Special Out instruction).

A Special In instruction causes the Adapter to transmit the saved address to the specified item space.

Command Formats

OTS (Item 1), (Item 2)

where: Item 1 is a 1-byte field equal to $002_{(8)}$

Item 2 is a 1-byte field equal to $002_{(8)}$

INS (Item 1), (Item 2)

where: Item 1 ib a 1-byte field equal to $002_{(8)}$

Item 2 is a 2-byte input field reserved for storage of the saved address.

Programming Restrictions

Since the "Capture P" function records the absolute value of the instruction following the execution of the "Capture P" request, each subroutine must subtract the P-bias from the saved address and must add 3 to the result in order to return to the calling program.

Example:

```
    Jump  -  OTS      Capture P
             GOTO     Exit to subroutine

  Return from INS      Get saved address - Store at Exit +1
   Subroutine  SB      Subtract 20₈, 21₈ (P Bias)
               ALB     Plus 003

    Exit     GOTO     (Saved address)
```

INTERRUPT

The interrupt package on the SYSTEM 2400 Processor resides on DMA Channel # 2 and provides a facility for generating, sensing, and processing channel monitor and service request interrupts. In addition to the eight channel interrupts, the package is capable of accepting up to four auxiliary external interrupt request inputs. From a user's (software) point of view, the package provides a means of:

- Linking to and returning from the interrupt routine.
- Enabling and disabling all or individual interrupts.
- Preserving the integrity of the worker program state by providing an alternate set of Active Records (AR) locations and a means of saving the program designators.
- Capturing and identifying up to 16 major interrupting conditions.

Use of the Interrupt Feature

Before interrupts can be utilized, the program must link interrupt occurance to the interrupt processing routine and establish interrupt lockouts so that only desired interrupts are recognized.

Monitor interrupts are initialized when the channel active designator goes from the active to inactive state; therefore, caution must be exercised in initializing interrupts to clear any residual interrupts. This may be done by executing.

```
INS      ITEM 1,     ITEM 2

When ITEM 1 = DMA channel 2
and ITEM 2  = 5-byte status area
```

Linkage to the interrupt processing routine may be provided by a GOTO instruction at location 24_8.

Interrupt lockouts may be established by executing

OTS ITEM 1, ITEM 2

where ITEM 1 contains the Interrupt Adapter ID and by convention is equal to 2. ITEM2 is a 3-byte item with the byte meaning as depicted in Figure G-1.

BYTE 1	BYTE 2	BYTE 3
Snap P Functions	Chan. Interrupt Lockout	Interrupt Lockout

Figure G-1. OTS, ITEM2, Three Bytes

Byte 1 assignments are as follows:

001 - Enable Interrupts or Remove Interrupt Lockout (RIL) - Interrupts are held locked out for one instruction following the OTS and then disabled when in the EXEC (interrupt) state.

002 - Capture P and lockout interrupts - Since the Capture P does not cause a transfer of control by itself, it can be used as a programmable disable of interrupts.

Figure G-10 provides a quick-reference data sheet for those who have SYSTEM 2400 experience.

Figure G-2 depicts byte 2 with a bit position lockout of the interrupts in byte 3 of the INS. A one (1) in memory sets the lockout; a zero (Ø) clears the lockout.

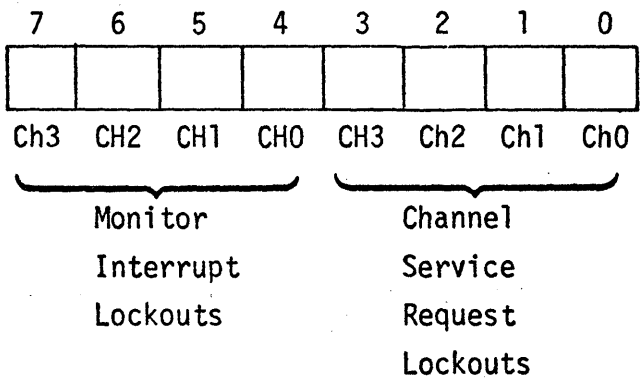


Figure G-2. OTS, ITEM2, Byte 2 Bit Assignments

Figure G-3 depicts byte 3 with a bit position lockout of the interrupts in byte 5 of the INS. A one (1) in memory sets the lockout; a zero (Ø) clears the lockout.

Figure G-3 depicts byte 3 with a bit position lockout of the interrupts in byte 5 of the INS. A one (1) in memory sets the lockout; a zero (Ø) clears the lockout.

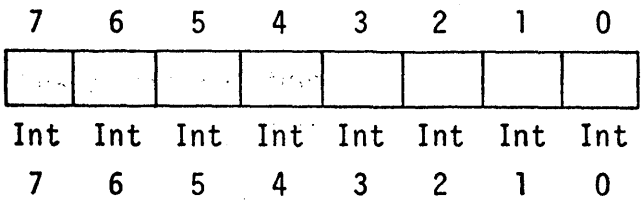


Figure G-3. OTS, ITEM2, Byte 3 Bit Assignments

Interrupt Execution

Upon interrupt, the instruction at location 24_8 is executed. Further interrupts are locked out until enabled with an OTS with RIL to the interrupt adapter. Furthermore, a different set of Active Records (AR's) are used while in the intertupt routine. They are assigned to storage locations 040_8 to 057_8. The worker state AR's ($000\text{-}017_8$) are used immediately following the execution of the enable interrupts.

To remove interrupt lockout, only byte 1 in Figure G-1 is required.

The event sequence for processing interrupts is shown in Figure G-8. The following paragraphs describe the event sequence depicted in Figure G-8 and in the order indexed (A through G) in the right margin.

A. Link to interrupt Program - This is generally a simple GOTO instruction.

B. Capture Return Address, Interrupt Status, and Program Designators - This is accomplished with one Special In instruction as follows:

```
INS     ITEM1,     ITEM2
```

where ITEM1 contains the Channel Number and by convention is equal to 2. ITEM2 is a 5-byte item with the byte meanings as depicted in Figure G-4.

BYTE 1	BYTE 2	BYTE 3	BYTE 4	BYTE 5
Return	Address	Channel Interrupt Status	Condition Designators	Interrupt Status

Figure G-4. INS, ITEM2 Bytes

Figure G-5 provides a snapshot of channel interrupt status of each bit in byte 3 at the time of interrupt. The monitor interrupt status must be serviced or saved following the snapshot. The channel service request interrupts will be cleared when the generating source has been service.

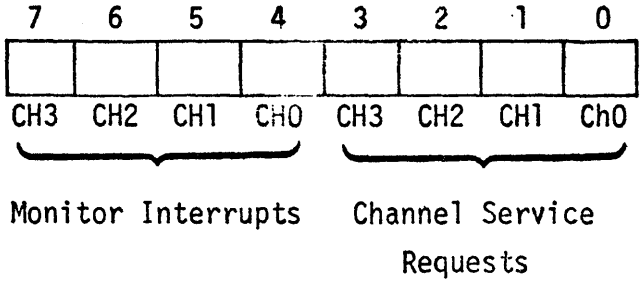


Figure G-5. INS, ITEM2, Byte 3 Bit Assignments

Figure G-6 bits show the status of the condition designators of byte 4 just prior to the interrupt. Only those conditions which are disturbed by the interrupt routine and are important to the proper operation of the worker program need to be restored.

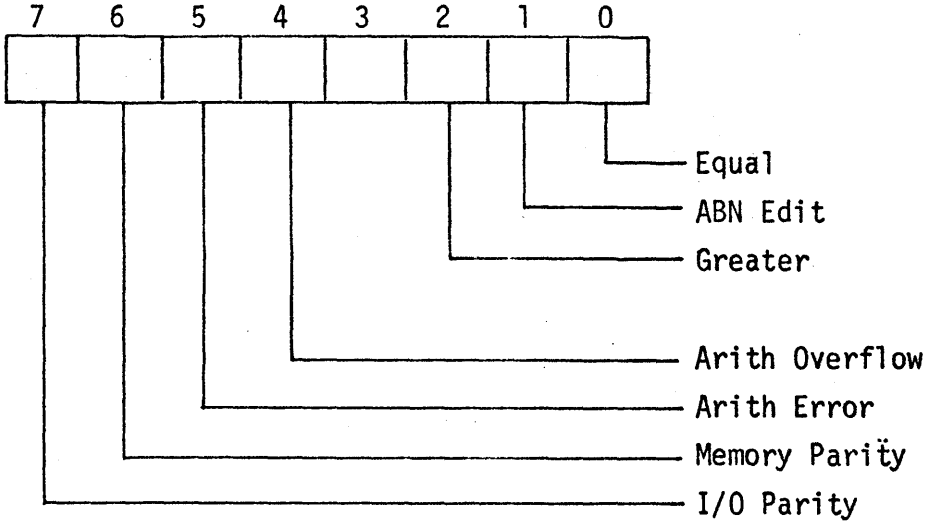


Figure G-6. INS, ITEM2, Byte 4 Bit Assignments

Figure G-7 depicts byte 5 of ITEM2 and the interrupt status bit assignment.

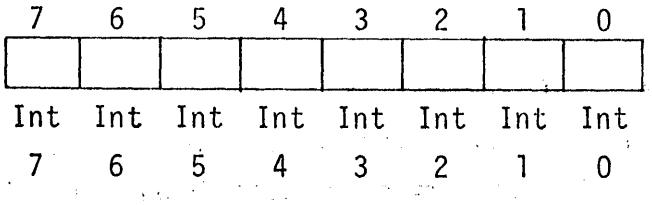


Figure G-7. INS, ITEM2, Byte 5 Bit Assignments

NOTE

This interrupt register is not assigned in the SYSTEM 2400 Processors. It could be used for Real Time Clock or other such interrupt conditions. These requests are cleared when the generating source has been serviced.

C. Determine Cause(s) of Interrupt - This is accomplished by a sequence of TEST MASK and GOTO instruction pairs.

D. Process Interrupts - This is unique to the individual programs. It may consist of swapping buffers and initiating an I/O or, on the other hand, it may simply involve setting a flag. If the interrupting condition requires a different set of interrupt lockouts, they must be established by the use of an OTS instruction to the interrupt package with ITEM2, byte 1 conditioned to capture P and disable interrupts, and bytes 2 and 3 selecting the desired lockouts. If no change to interrupt lockouts is desired, the OTS instruction is unnecessary.

E. Restore Condition Designators - This is accomplished by executing instructions which cause the Condition Designators to be set to the state existing prior to the interrupt. A sample restore designators routine is depicted in Figure G-9. This routine restores all designators. A typical path taken through this routine requires 232 usec. The worst case path requires 258 usec.

F. Enable Interrupts - This is accomplished by executing

```
OTS     ITEM1,     ITEM2
```

where ITEM1 contains the Interrupt Adapter ID and by convention is equal to 2. ITEM2 is a 1-byte item equal to 1.

G. Return to Interrupted Program - This GOTO instruction must immediately follow the Enable Interrupt to insure that interrupt return linkage is not lost.

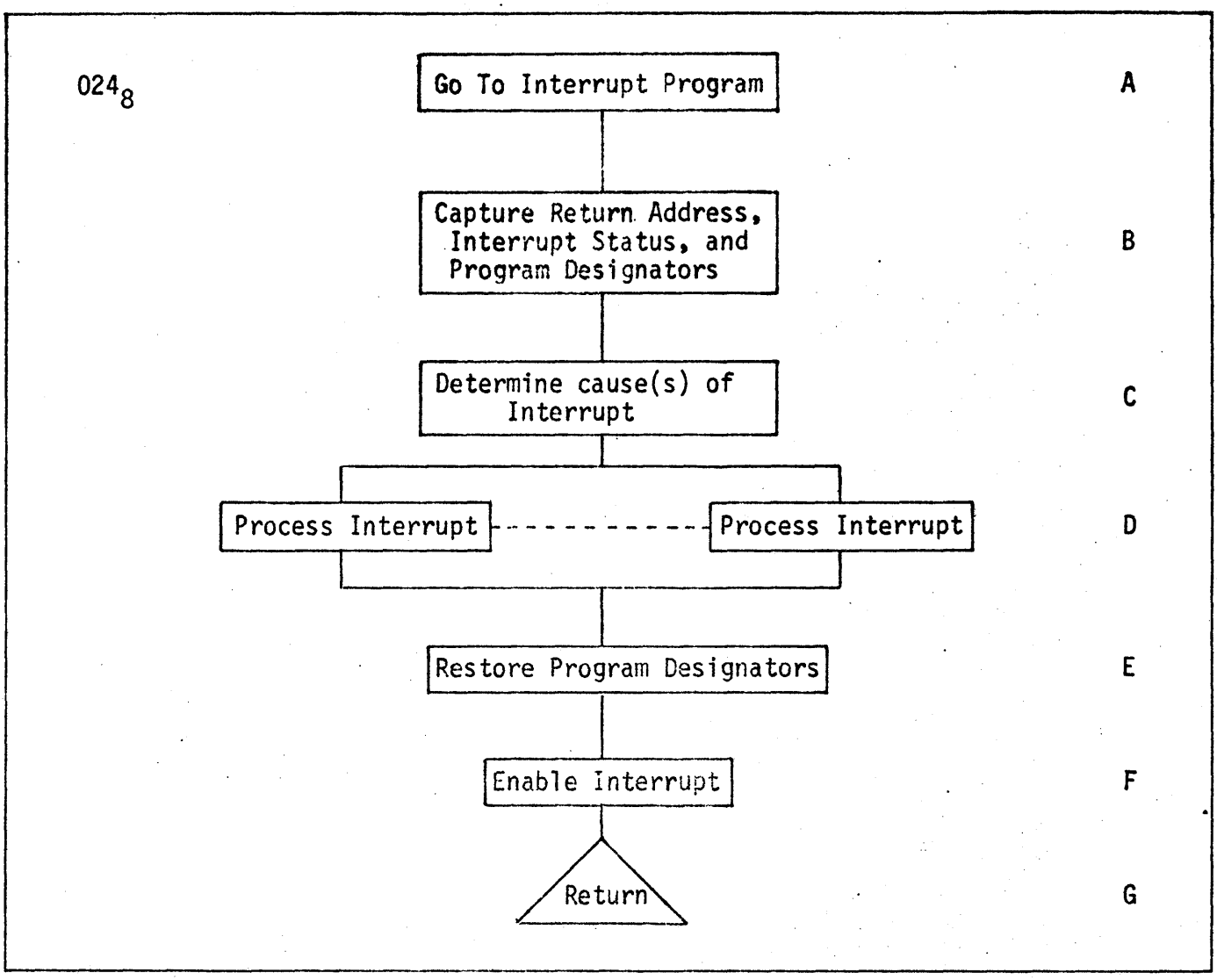
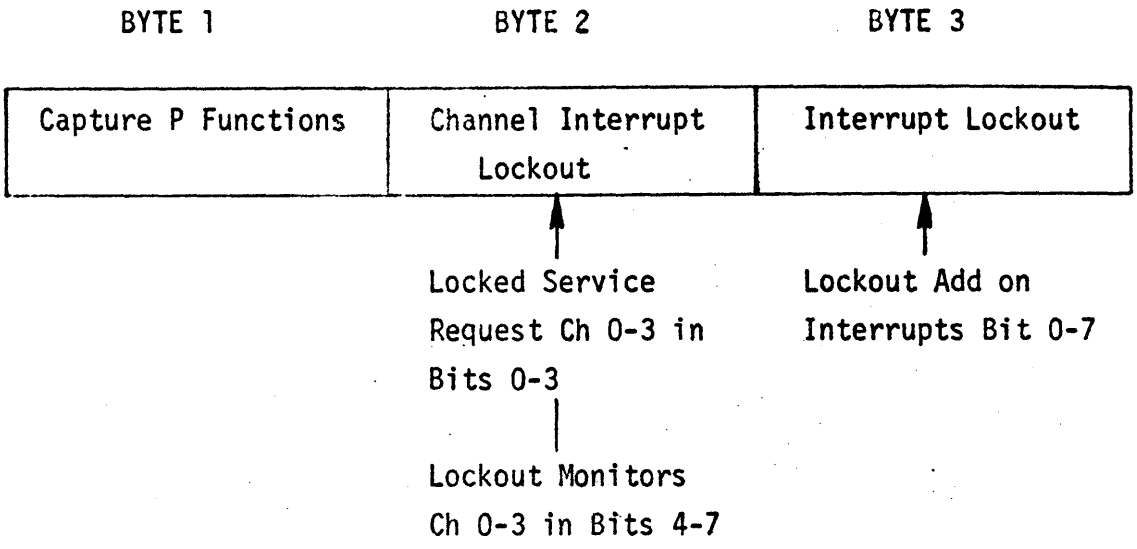


Figure G-8. Interrupt Processing Sequence

```
        M       SAVP, EXOUT             Set up Return
        SB      EXOUT, BIAS, EXOUT
        TM      DSG, 002                Check ABN Edit
        GE      ABED
        EX      ABUF, BYTE1             Clear ABN Edit
        G       ATEST
ABED    EX      ABUF, BYTE2             Set ABN Edit
ATEST   TM      DSG, 060                Test for Arith Conditions
        GNE     NOSET                   If None set, Jump, Else
        TM      DSG, 040                Test for Arith Error
        GE      ARER                    If Error, Jump, Else
        ALB     ARITH1, RESULT, 177     Set Overflow
        G       GTEST                   To Greater Test
ARER    ALB     ARITH2, RESULT, 000     Set Arith Error
        G       GTEST
NOSET   ALB     ARITH1, RESULT, 000     Clear Arith Conditions
GTEST   TM      DSG, 004                Test for Greater
        GE      GREAT                   Yes, Jump
        ML      RESULT, 200             Set Negative
        G       SETG
GREAT   ML      RESULT, 000             Set Positive
SETG    TBS     RESULT                  Set (or Clear) Greater
        TM      DSG, 001                Restore Equal
        OTS     ADP, ABLE               Enable Interrupts
        G       RETURN
```

Figure G-9. A Sample Restore Designators Routine

Interrupt Adapter - 002
OTS Functions - 3 Bytes



Capture P and Disable Interrupts - 002
Enable Interrupts or RIL - 001
INS Status - 5 Bytes

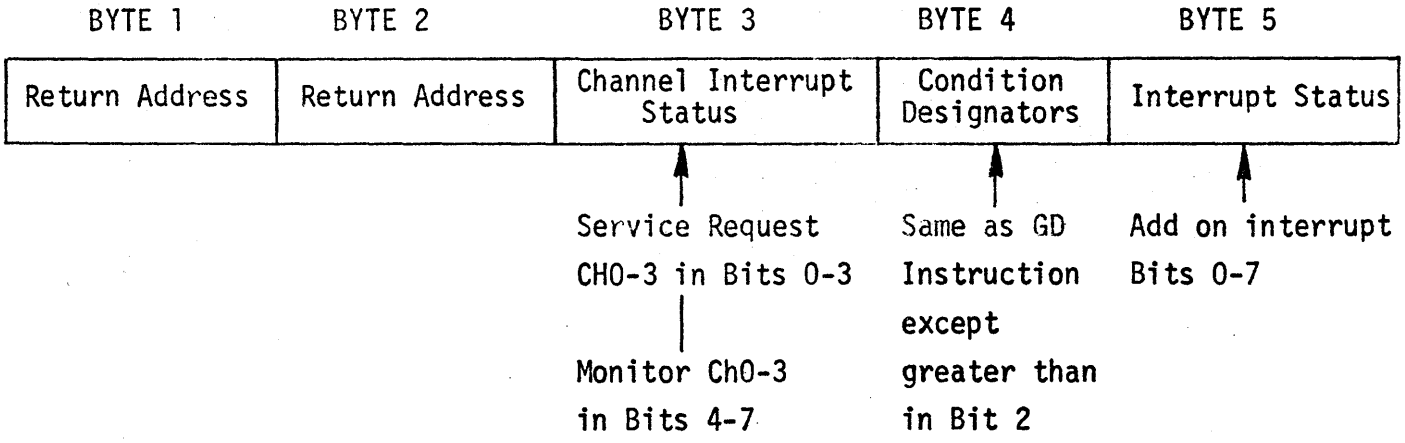


Figure G-10. Quick-Reference Data Sheet

Linkage

On interrupt, go to 024_8 and execute GOTO instruction. Interrupt set of AR's is located at 040_8 - 057_8. Hence, AR's need not be saved or restored. A common P-Bias is assumed. To return from the interrupt routine, simple Enable Interrupts to again utilize worker AR set.

APPENDIX H

UTILITY ADAPTER

GENERAL

The SYSTEM 2400 Utility Adapter accommodates special input and output functions which are not easily accomplished in the Control Unit and enhances the programmable capabilities of the 2400 Processors. It is configured to be on DMA Channel No. 1 (thereafter referred to as Adapter Channel No. 1 (101_8) and is accessed via Special In (INS) and Special Out (OTS) instructions.

The utility adapter comprises several of the Processor's complement of logic modules. Although the adapter is a standard hard-wired feature of the Processor, its operator/function is strictly under program control. Operationally, it is implemented to accommodate special programmable functions and parameters that will vary from system to system.

Due to the adapter's inherent variable programmables and requirements, the SYSTEM 2400 Mohawk Data Language does not include the software documentation normally afforede with the standard system software. However, related documentation is provided within this manual and SYSTEM 2400 *Processor Programming in Machine Code* (Form No. M-2269).

The Utility Adapter provides the following programmable functions:

- Logical instructions
- CRC Calculations
- Real Time Clock capability

These functions are controlled via the Special In and Special Out instructions. The formats utilized by the instructions are as follows:

Special Out

OTS (Item 1), (Item 2)

where: Item 1 is equal to (001) which is the DMA address of the Utility Adapter.

Item 2 is a multi-byte field comprising one command byte and X data bytes. The command byte is the first byte of the field and is further described under "Command Codes."

The data field can be from 1 to 255 bytes and is operated on by Utility Adapter byte-serial as defined by the command code.

If more than 255 bytes of data are to be transferred to the Adapter or if the data is to be transmitted with more than one Special Out instruction for a single operation, the Command code of all subsequent Special Out instructions must have the 2_6 bit set. This conditions the Adapter to save the result of the computation and proceed with the operation using the saved result.

Special In

Format:

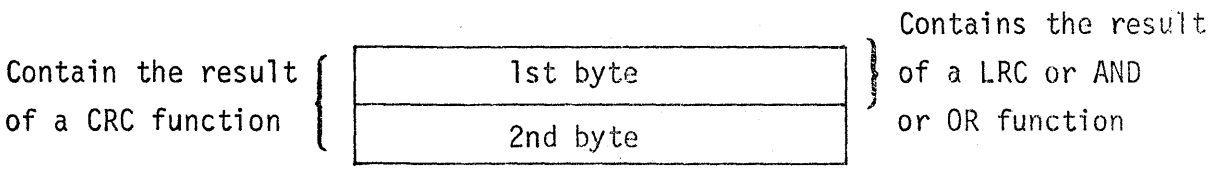
INS (Item 1), (Item 2)

where: Item 1 is equal to (001) which is the DMA address of the Utility Adapter. Item 2 describes a receiving field for information from the Utility Adapter.

The purpose of the input function is to input the result of the computation from the Logical and CRC feature.

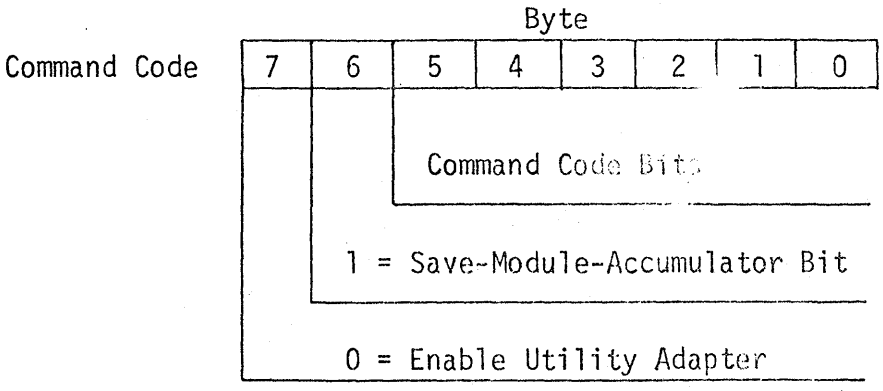
Upon being initiated by a Special In instruction, the Adapter will respond with up to two bytes.

If the program wishes only the results of the Logical Set, one byte is enough, while two bytes are required for the result of the CRC as shown below:



COMMAND CODES

The command codes used in the instructions to direct the Utility Adapter to perform a specific function contain two modifier bits, as shown below:



Utility adapter commands are:

$X01_8$ - Exclusive "OR"/LRC	Logical Set
$X02_8$ - Logical "AND"	
$X04_8$ - Inclusive "OR"	
$X10_8$ - 16-Bit CRC	CRC Set
040_8 - k2-Bit CRC	
050_8 - Load Utility Adapter	

Command byte modifiers are:

Bit 2^7 - Ø-Enable Utility Adapter
Bit 2^6 - k-Save Accumulator
060_8 - Clear Real Time Clock (RTC)

LOGICAL SET FEATURE

The Logical Set Feature consists of the OR (exclusive), AND, plus OR (inclusive) functions.

A. OR (exclusive)
The primary requirement for this function is to compute the Longitudinal Redundancy Check (LRC) character for a string of data characters.

The Adapter will accept a string of characters and compute the LRC character.

B. AND
The Adapter will logically AND two bytes of data.

C. OR (inclusive)
The Adapter will logically OR two bytes of data.

Logical Instructions

To execute Logical Instructions, perform the following:

```
OTS     Item 1, Item 2
INS     Item 1, Item 3
```

where, Item 1 is numerically equal to 1 (adapter channel number), Item 2 contains the logical command byte followed by the data to be operated upon, and Item 3 is a 1-or 2-byte item where the results will be placed.

The command byte must be the first byte of Item 2 and equal to one of the following:

001 or 101 - EXCLUSIVE OR
002 or 102 - AND
004 or 104 - INCLUSIVE OR
} ——— Generate 1-byte result

010 or 110 - CRC (16 bit)
040 or 140 - CRC (12 bit)
} ——— Generate 2-byte result

In each case, the first command type (e.g., 001) operates only on the bytes of Item 2; the second command type (e.g., 101) utilizes the prior result as well as all bytes of Item 2.

In each case, the logical operator applies sequentially to all bytes of Item 2 following the command byte.

Examples of each logical operation are given below for two data bytes. The process is accumulative for items with more than two data bytes.

Exclusive OR/LRC

Command Code: 001_8

Example:

01100101	First data byte
01001100	Second data byte
00101001	Results

Logical AND

Command Code: 002_8

Example:

01100101	First data byte
01001100	Second data byte
01000100	Results

Inclusive OR

Command Code: 004_8

Example:

01100101	First data byte
01001100	Second data byte
01101101	Results

CRC SET

The CRC Feature is a Cyclic Redundancy Check. There are two types, CRC-16 and CRC-12.

The adapter will accept a string of data characters and compute the CRC. The CRC result is two bytes in length.

Cyclic Redundancy Check (CRC)

The Utility Adapter performs cyclic redundancy checks.
The following two checking polynominals are implemented.

12-Bit CRC - $X^{12}+X^{11}+\ldots+X^3+X^2+X+1$
16-Bit CRC - $X^{16}+X^{15}+\ldots+X^2+X+1$

Results of the checking operation are contained in two bytes. Examples for both checking operations are given below:

16-Bit CRC

Command Code: 010_8

Example:

00000111	First data byte
00000011	Second data byte
	Results -
01000010	Least significant byte
00110001	Most significant byte

12-Bit CRC

Command Code: 040_8

Example:

These bits are ignored by the Tility Adapter

XX 000111	First data byte
XX 001111	Second data byte
	Results -
00 011010	Least significant byte
00 100001	Most significant byte

These bits are always zeros

LOAD UTILITY ADAPTER

A "preset" value may be sent to the adapter with a Load Adapter (050) command byte. Example:

```
OTS    ITEM 1,    ITEM 2
```

Where Item 1 is numerically equal to one and Item 2 is

```
050    XXX        YYY
```

where XXX YYY represents the 2-byte preset value to be loaded into the adapter. Two bytes should always be loaded. Byte XXX is loaded into the adapter register associated with one byte logical results (which is also used for the most significant byte for CRC). Byte YYY is loaded into the adapter register used for the least significant byte of CRC.

COMMAND BYTE MODIFIERS(X)

Save Module Accumulator Bit (2^6)

This bit directs the Utility Adapter to save the contents of the accumulator. This feature permits logical operations on strings of data characters in excess of 255 bytes (the maximum number of data bytes transferred with a single instruction is 256 with the first byte being the command code) or on a group of single bytes or strings of bytes located in different parts of memory. For example, an LRC operation on a string of data characters greater than 255 bytes would require that bit 2^6 be set to a "1" in all subsequent instructions conveying data during this operations. In addition, the contents of the module accumulator may be stored in main memory using the Store Module Accumulator instruction and may be returned to the accumulator in the Utility Adapter using the Enter Module Accumulator instruction, thus allowing more than one subroutine in the main program to utilize the features of the Utility Adapter.

Enable Utility Adapter Bit (2^7)

Bit 2^7 set to a "0" in a command code enables the Utility Adapter and informs the other connected peripheral to deselect.

REAL TIME CLOCK

The RTC consists of a 16-binary counter, free running oscillator, and adapter channel control logic. The 16-bit counter is incremented by the oscillator

frequency of 256HZ+ 2.5HZ. Each time the counter is incremented, the RTC makes a memory cycle request to the Processor. When the request is granted, the most significant eight bits are stored in memory location 002_8, and the least significant eight bits are stored in location 023_8.

Clear Real Time Clock Command (060_8)

This command presets the clock to a value of $000\ 000_8$. Sending this command to the Utility Adapter via the Special Out instruction forces the 16-bit counter in the Adapter to be cleared, which in turn clears memory locations 022_8 and 023_8.

Command Format

OTS (Item 1), (Item 2)

where: Item 1 is a 1-byte field with a value of 001_8.
Item 2 is a 1-byte field with a value of 060_8.
(Utility Adapter Command Code).

Programming Restrictions

The Real Time Clock loads addresses 22_8 and 23_8 by stealing memory cycles. It is possible for the RTC to increment the count in the middle of a program instruction which is manipulating addresses 22_8 and 23_8. When the program is utilizing both bytes of the RTC, it must take precautions to insure that this has not occurred (i.e., that the 16-bit count does not consist of one incremented byte and one non-incremented byte.)

The RTC counts from 000000_8 to 377377_8. When the count becomes all ones, the counter automatically "wraps back around" to 000000_8. No overflow indication is set.

The RTC oscillator runs asynchronous relative to the Processor. Execution of the clear RTC command does not cause the oscillator to be reset. It is possible, therefore, for the RTC to increment address 023 from 000_8 to 001_8 between a clear command and any instruction testing address 023 for zero.

READER'S COMMENT FORM

Machine Code and Assembly Language - Edition 2

Form No. PM-1948

Please restrict remarks to the publication itself, giving specific page and line references with your comments when appropriate. This form will be sent to the publication's author for appropriate action. All comments and suggestions become the property of MDS.

Requests for system assistance or publications should be directed to your MDS representative or to the MDS Branch Office serving your area.

ERRORS NOTED:

SUGGESTIONS FOR IMPROVEMENT:

How do you use this document?

- ☐ As an operator's Reference Manual
- ☐ As an introduction to the subject
- ☐ As an aid to instruction in a class
- ☐ As a student text book
- ☐ For advanced knowledge of subject

Do you wish a reply?

- ☐ Yes
- ☐ No

Your Name _____ Date _____

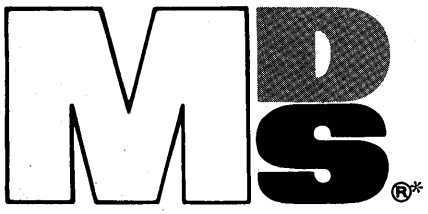
Occupation _____

Company _____

Address _____

Street City State Zip Code

THE THROUGHPUT SPECIALISTS



MOHAWK DATA
SCIENCES CORP.

PALISADE STREET • HERKIMER, NEW YORK 13350