

HP 3000 SUPPORT DATA



DATA SYSTEMS DIVISION
TECHNICAL SUPPORT

HP 3000 SUPPORT DATA

5951-5647

PRINTED IN U.S.A.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

TABLE OF CONTENTS

HP 3000 CENTRAL PROCESSOR UNIT

INSTRUCTION SET	
Index—Instruction Types	1
CPU Instructions by Name or Mnemonic	2
Definitions	6
WORD FORMATS	81
ADDRESSING MODES	85
INSTRUCTION CODING, STATUS, ETC.	
Instruction Coding (Numerical Index)	87
Interrupt Control Stack	88
ASCII Codes	88
Status Register	88
CPX Registers	88
Interrupts/Traps	88
HARDWARE AND AUXILIARY CONTROL PANELS	
Auxiliary Control Panel	89
Hardware Maintenance Panel	90
Registers	91
Panel Operation	93
System Dump	94
Cold Load	94
Maintenance Panel Block Diagram	97
CPU BLOCK DIAGRAM, ROM, ETC.	
CPU Backplane	98
Hardware Block Diagram	99
Firmware Block Diagram	100
Register Renaming	101
Stack Operations	102
ROM Formats	104
ROM Fields	105
ROM Cycles	121
CST/STT SEGMENT TRANSFER	125
FIXED CORE LOCATIONS	126
INTERRUPTS, DRT, ETC.	
Interrupts and Traps	127
Interrupt Control Stack	128
Device Reference Table	129

LIST OF EFFECTIVE PAGES

PAGES

1 thru 123
125 thru 129

DATE

June 1973
June 1973

INDEX

INSTRUCTION TYPES

INTEGER INSTRUCTIONS

DOUBLE INTEGER INSTRUCTIONS

LOGICAL INSTRUCTIONS

BIT, FIELD, AND BYTE INSTRUCTIONS

FLOATING POINT INSTRUCTIONS

**INDEX, EXCHANGE, LOOP CONTROL
AND BRANCH INSTRUCTIONS**

**REGISTER AND PROGRAM
CONTROL INSTRUCTIONS**

MOVE AND SPECIAL INSTRUCTIONS

I/O AND INTERRUPT INSTRUCTIONS

Other CPU information follows the instruction set, beginning at page 81.

CPU INSTRUCTIONS BY NAME OR MNEMONIC

Add A(TOS) to X	ADAX	10, 52
Add B(TOS-1) to X	ADBX	11, 53
Add (TOS to TOS-1)	ADD	10
Add immediate (to TOS)	ADDI	10
Add memory (to TOS)	ADDM	9
Add (N or TOS) to S	ADDS	63
Add X to A(TOS)	ADXA	11, 52
Add X to B(TOS-1)	ADXB	11, 53
Add immediate to X	ADXI	12
And, logical (TOS with TOS-1)	AND	35
And immediate, logical (immediate with TOS)	ANDI	35
Arithmetic shift left (TOS)	ASL	20
Arithmetic shift right (TOS)	ASR	20
Branch on Condition Code	BCC	59
Branch on Carry	BCY	60
Branch on no Carry	BNCY	60
Branch on no Overflow	BNOV	61
Branch on Overflow	BOV	60
Branch	BR	59
Branch if (TOS) even	BRE	61
Branch if (TOS) odd	BRO	61
Byte test (on TOS)	BTST	40
C(TOS-2) to A(TOS) to B(TOS-1) (to C(TOS-2)	CAB	56
Control I/O (Control word from TOS)	CIO	79
Command (central data bus)	CMD	74
Compare (TOS-1 to TOS)	CMP	8
Compare bytes (ARRY1 to ARRY2)	CMPB	40
Compare immediate (TOS to immediate)	CMPI	9
Compare memory (to TOS)	CMPM	8
Compare negative immediate (TOS to immediate)	CMPN	9
Compare range and branch (TOS>=X>=TOS-1)	CPRB	21, 55
Circular shift left (TOS)	CSL	32
Circular shift right (TOS)	CSR	33
Decrement A(TOS), branch if zero	DABZ	54
Double add (TOS, TOS-1 to TOS-2, TOS-3)	DADD	23
Double arithmetic shift left (TOS, TOS-1)	DASL	26
Double arithmetic shift right (TOS, TOS-1)	DASR	27
Double compare (TOS-2, TOS-3 to TOS, TOS-1)	DCMP	23
Double circular shift left (TOS, TOS-1)	DCSL	33
Double circular shift right (TOS, TOS-1)	DCSR	33
Double delete (TOS, TOS-1)	DDEL	26, 48
Double duplicate (TOS, TOS-1)	DDUP	25, 47
Decrement A(TOS)	DECA	17

CPU INSTRUCTIONS BY NAME OR MNEMONIC

Decrement B(TOS-1)	DECB	17
Decrement memory	DECM	17
Decrement X	DECX	18
Delete (TOS)	DEL	19,31
Delete B(TOS-1)	DELB	20
Double float (TOS, TOS-1)	DFLT	27, 44
Divide (TOS-1 by TOS)	DIV	14
Divide immediate (TOS by immediate)	DIVI	15
Divide long (TOS-1, TOS-2 by TOS)	DIVI	24
Double logical shift left (TOS, TOS-1)	DLSL	31
Double logical shift right (TOS, TOS-1)	DLSR	32
Double negate (TOS, TOS-1)	DNEG	25
Deposit field (TOS into TOS-1)	DPF	38
Double subtract (TOS, TOS-1 from TOS-2, TOS-3)	DSUB	24
Double test (TOS, TOS-1)	DTST	27, 48
Duplicate (TOS)	DUP	19, 30
Decrement X, branch if zero	DXBZ	54
Double exchange (TOS, TOS-1 with TOS-2, TOS-3)	DXCH	26, 56
Double (push zero)	DZRO	25
Exchange A(TOS) and X	XAX	55
Exchange B(TOS-1) and X	XBX	56
Exchange A(TOS) and B(TOS-1)	XCH	55
Exchange DB (with TOS)	XCHD	63
Execute (instruction in stack)	XEQ	67
Exclusive OR, logical (TOS with TOS-1)	XOR	34
Exclusive OR immediate, logical (. . . with TOS)	XORI	35
Extract (field from TOS)	EXF	38
Exit (from procedure)	EXIT	66
Floating add (TOS, TOS-1 to TOS-2, TOS-3)	FADD	46
Floating compare (TOS, TOS-1 to TOS-2, TOS-3)	FCMP	45
Floating divide (TOS-2, TOS-3 by TOS, TOS-1)	FDIV	47
Fix and round (TOS, TOS-1)	FIXR	45
Fix and truncate (TOS, TOS-1)	FIXT	45
Float (TOS)	FLT	21, 44
Floating multiply (TOS, TOS-1 by TOS-2, TOS-3)	FMPY	46
Floating negate (TOS, TOS-1)	FNEG	47
Floating subtract (TOS, TOS-1 from TOS-2, TOS-3)	FSUB	46
Halt	HALT	68
Increment A(TOS), branch if zero	IABZ	54
Increment A(TOS)	INCA	16
Increment B(TOS-1)	INCB	16
Increment memory	INCM	15
Increment X	INCX	16

CPU INSTRUCTIONS BY NAME OR MNEMONIC

Increment X, branch if zero	IXBZ	53
Logical add (TOS to TOS-1)	LADD	29
Logical compare (TOS-1 to TOS)	LCMP	28
Load byte (from memory)	LDB	38
Load double (from memory)	LDD	22, 43
Load immediate (immediate onto TOS)	LDI	6
Logical divide (TOS-1, TOS-2 by TOS)	LDIV	30
Load negative immediate (onto TOS)	LDNI	6
Load double from program, negative(P)	LDPN	22, 43
Load double from program, positive(P)	LDPP	22, 43
Load index (from memory)	LDX	7, 50
Load X into A(TOS)	LDXA	51
Load X into B(TOS-1)	LDXB	52
Load X (with) immediate	LDXI	7,50
Load X (with) negative immediate	LDXN	7, 50
Load label (onto TOS)	LLBL	64
Linked list search	LLSH	73
Logical multiply (TOS by TOS-1)	LMPY	29
Load (onto TOS)	LOAD	6, 28
Load relative address (onto TOS)	LRA	64
Logical shift left (TOS)	LSL	31
Logical shift right (TOS)	LSR	32
Logical subtract (TOS from TOS-1)	LSUB	29
Move (words) (ARRY1 to ARRY2)	MOVE	69
Multiply (TOS by TOS-1)	MPY	14
Multiply immediate (immediate by TOS)	MPYI	14
Multiply long (TOS by TOS-1)	MPYL	24
Multiply memory (by TOS)	MPYM	13
Modify, test and branch on A(TOS)	MTBA	58
Modify, test and branch on X	MTBX	58
Move bytes (ARRY1 to ARRY2)	MVB	69
Move (words) from DB+ to DL+	MVBL	70
Move bytes while (ARRY1 to ARRY2)	MVBW	71
Move (words) from DL+ to DB+	MVLB	70
Negate (TOS)	NEG	15
No operation	NOP	74
Not (function — one's complement)	NOT	30
OR, logical (TOS with TOS-1)	OR	34
OR immediate (logical or immediate with TOS)	ORI	34
Pause	PAUS	67
Procedure call	PCAL	65
Privileged load from absolute address	PLDA	72

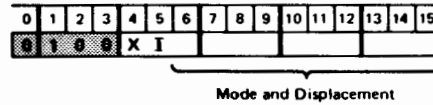
CPU INSTRUCTIONS BY NAME OR MNEMONIC

Push registers (onto stack)	PSHR	62
Privileged store into absolute address	PSTA	73
Read I/O (onto TOS)	RIO	77
Read mask (onto TOS)	RMSK	75
Read switch register (onto TOS)	RSW	72
Subtract immediate from X	SBXI	13
Subroutine call	SCAL	66
Scan bits (on TOS)	SCAN	36
Scan (bytes) until (ARRY1)	SCU	41
Scan (bytes) while (ARRY1)	SCW	41
Set enable/disable external interrupts	SED	75
Set registers (from stack)	SETR	62
Set interrupt (device)	SIN	76
Start I/O (@TOS)	SIO	77
Set external Interrupt Reference Flag	SIRF	76
Set mask (from TOS)	SMSK	75
Store A(TOS) into X	STAX	51
Store byte (into memory)	STB	39
Store B(TOS-1) into X	STBX	51
Store double (into memory)	STD	23, 44
Store (into memory)	STOR	8, 28
Subtract (TOS from TOS-1)	SUB	12
Subtract immediate (immediate from TOS)	SUBI	13
Subtract memory (from TOS)	SUBM	12
Subtract (N or TOS) from S	SUBS	64
Subroutine exit	SXIT	67
Triple arithmetic shift left (TOS, -1, -2)	TASL	49
Triple arithmetic shift right (TOS, -1, -2)	TASR	49
Test and branch on A(TOS)	TBA	57
Test bit and set Condition Code (TOS)	TBC	36
Test and branch on X	TBX	57
Test and complement bit, set Condition Code	TCBC	36
Test (word on TOS)	TEST	21
Test I/O (onto TOS)	TIO	79
Triple normalizing shift left (TOS, -1, -2)	TNSL	49
Test and reset bit, set Condition Code (TOS)	TRBC	37
Test and set bit, set Condition Code (TOS)	TSBC	37
Test and set bits in memory (from TOS)	TSBM	37
Write I/O (from TOS)	WIO	78
Zero (push zero)	ZERO	18
Zero B(TOS-1)	ZROB	18
Zero X	XROX	19

INTEGER INSTRUCTIONS

LOAD Load word onto stack

Indicators: CCA
P±, DB±, Q±, S- relative
Direct or indirect
Indexing available

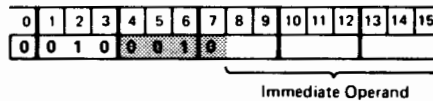


The content of the effective address location is pushed onto the stack. An attempt to load from beyond the limits defined by PB and PL or (in user mode only), DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	P REL.	023	AC1P	0027	1
(S) := (E);	DB REL.	020	AC1D	0013	1
	Q REL.	020	AC1D	0013	1
	S REL.	021	AC1S	0012	1
	JLUI	022	LOAD	0122	1

LDI Load immediate

Indicators: CCA on the
new TOS

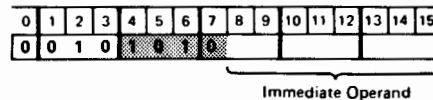


The immediate operand N is pushed onto the stack. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	UNC.	110	LPNI	1521	0
(S) := N;					

LDNI Load negative immediate

Indicators: CCA on the
new TOS

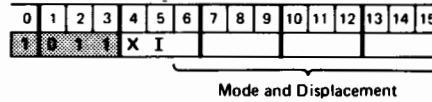


The immediate operand N is two's complemented and pushed onto the stack as a negative integer. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	UNC.	150	LPNI	1521	1
(S) := -N;					

LDX Load Index

Indicators: CCA
 P±, DB±, Q±, S relative
 Direct or indirect
 Indexing available



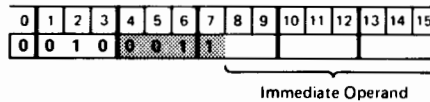
The content of the effective address memory location is loaded into the Index register. An attempt to load from beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

X := (E);

COND	LUTA	LABEL	RAR	W
P REL.	057	AC1P	0027	1
DB REL.	054	AC1D	0013	1
Q REL.	054	AC1D	0013	1
S REL.	055	AC1S	0012	1
JLUI	056	LDX	0121	1

LDXI Load X immediate

Indicators: unaffected



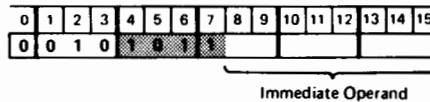
The Index register is loaded with the immediate operand N. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.

X := N;

COND	LUTA	LABEL	RAR	W
UNC.	114	XPNI	1527	0

LDXN Load X negative immediate

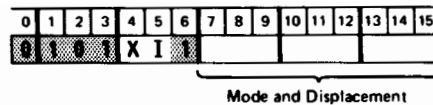
Indicators: unaffected



The Index register is loaded with the 16-bit two's complement of the immediate operand N. The value of N is expressed as a positive integer in the range 0 through 255.

X := -N;

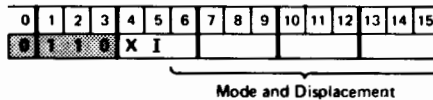
COND	LUTA	LABEL	RAR	W
UNC.	154	XPNI	1527	1

STOR Store TOS into memory

Indicators: unaffected
 DB+, Q±, S- relative
 Direct or indirect
 Indexing available

The content of the TOS is stored into the effective address memory location, and is then deleted from the stack. In user mode, an attempt to store beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

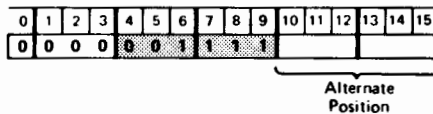
	COND	LUTA	LABEL	RAR	W
(E) := (S);	DB REL.	024	AC1D	0013	1
S := S-1;	Q REL.	024	AC1D	0013	1
	S REL.	025	AC1S	0012	1
	JLUI	026	STOR	0220	1

CMPM Compare TOS with memory

Indicators: CCC
 P±, DB+, Q±, S- relative
 Direct or indirect
 Indexing available

The Condition Code is set to pattern C as a result of the comparison of the TOS with the content of the effective address location. The TOS is then deleted. An attempt to reference a location beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
CC := (S) : (E);	P REL.	033	AC1P	0027	1
S := S-1;	DB REL.	030	AC1D	0013	1
	Q REL.	030	AC1D	0013	1
	S REL.	031	AC1S	0012	1
	JLUI	032	CMPM	0400	1

CMP Compare

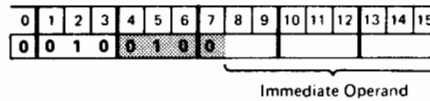
Indicators: CCC

The Condition Code is set to pattern C as a result of the integer comparison of the second word of the stack with the TOS. Both words are deleted.

	COND	LUTA	LABEL	RAR	W
CC := (S-1) : (S);	SR<4	317	SRP	0004	1
S := S - 2;	SR=4	357	CMP	0453	1

CMPI Compare immediate

Indicators: CCC

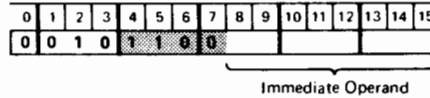


The Condition Code is set to pattern C as a result of the comparison of the TOS with the immediate operand N. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255. The TOS is deleted.

	COND	LUTA	LABEL	RAR	W
CC := (S) : N;	SR<2	---	SRP	0004	-
S := S - 1;	SR>=2	120	CPNI	0460	1

CMPN Compare negative immediate

Indicators: CCC

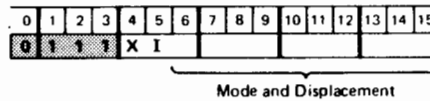


The Condition Code is set to pattern C as a result of the comparison of the TOS with the two's complement of the immediate operand N. The value of N is expressed as a positive integer in the range 0 through 255. The TOS is deleted.

	COND	LUTA	LABEL	RAR	W
CC := (S) : -N	SR<2	---	SRP	0004	-
S := S - 1;	SR>=2	160	CPNI	0460	0

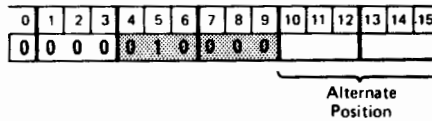
ADDM Add memory to TOS

Indicators: CCA, Carry,
Overflow
P±, DB±, Q±, S- relative
Direct or indirect
Indexing available



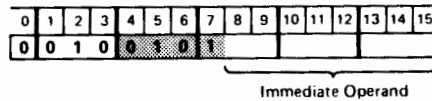
The content of the effective address memory location is added in integer form to the TOS. The result replaces the operand on the TOS. An attempt to reference a location beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
(S) := (S) + (E);	P REL.	037	AC1P	0027	1
	DB REL.	034	AC1D	0013	1
	Q REL.	034	AC1D	0013	1
	S REL.	035	AC1S	0012	1
	JLUI	036	ADDM	0254	1

ADD AddIndicators: CCA, Carry,
Overflow

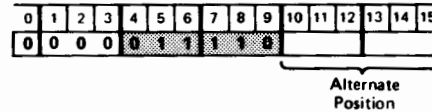
The top two words of the stack are added in integer form and are then deleted.
The resulting sum is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) + (S);	SR<2	---	SRP	0004	-
S := S - 1;	SR>=2	240	ADD	0463	1

ADDI Add immediateIndicators: CCA on the
new TOS,
Carry, Overflow

The immediate operand N is added to the TOS in integer form, and the sum replaces the TOS. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
(S) := (S) + N;	SR<2	---	SRP	0004	-
	SR>=2	124	ASI	1523	0

ADAX Add A to XIndicators: CCA on the
new X, Carry,
Overflow

The TOS is added in integer form to the content of the Index register. The sum replaces the content of the Index register, and the TOS is deleted.

	COND	LUTA	LABEL	RAR	W
X := X + (S);	SR<2	---	SRP	0004	-
S := S - 1;	SR>=2	256	ADAX	0720	1

ADXA Add X to A

Indicators: CCA on the new TOS, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	1						

Alternate Position

The content of the Index register is added to the TOS, and the sum replaces the TOS.

	COND	LUTA	LABEL	RAR	W
(S) := (S) + X;	SR<2	---	SRP	0004	-
	SR>=2	257	ADXA	0722	1

ADBX Add B to X

Indicators: CCA on the new X, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	1	0						

Alternate Position

The second word of the stack is added in integer form to the content of the Index register, and the result replaces the content of the Index register.

	COND	LUTA	LABEL	RAR	W
X := X + (S-1);	SR<2	---	SRP	0004	-
	SR>=2	276	ADBX	0721	1

ADXB Add X to B

Indicators: CCA on the new B, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	1	1						

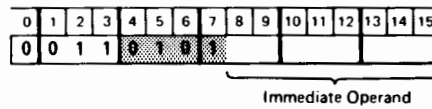
Alternate Position

The content of the Index register is added in integer form to the second word of the stack, and the sum replaces the second word of the stack.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) + X;	SR<2	---	SRP	0004	-
	SR>=2	277	ADXB	0723	1

ADXI Add immediate to X

Indicators: CCA on X

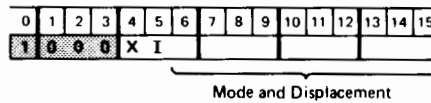


The immediate operand N is added to the content of the Index register in integer form. The sum replaces the Index register content. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
$X := X + N;$	SR<2	---	SRP	0004	-
	SR>=2	126	ASXI	1530	0

SUBM Subtract memory from TOS

Indicators: CCA, Carry,
Overflow
P±, DB±, Q±, S- relative
Direct or indirect
Indexing available

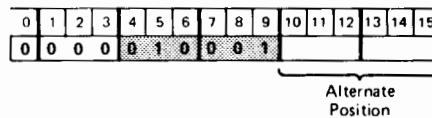


The content of the effective address memory location is subtracted in integer form from the TOS. The result replaces the operand on the TOS. An attempt to reference a location beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
(S) := (S) - (E);	P REL.	043	AC1P	0027	1
	DB REL.	040	AC1D	0013	1
	Q REL.	040	AC1D	0013	1
	S REL.	041	AC1S	0012	1
	JLUI	042	SUBM	0257	1

SUB Subtract

Indicators: CCA, Carry, Overflow



The TOS is subtracted in integer form from the second word of the stack and both words are then deleted. The resulting difference is then pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) - (S);	SR<2	---	SRP	0004	-
S := S - 1;	SR>=2	241	SUB	0464	1

SUBI Subtract immediate

Indicators: CCA on the new TOS, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	1	1	0								

Immediate Operand

The immediate operand N is subtracted from the TOS in integer form, and the result replaces the TOS. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.

(S) := (S) - N;

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	130	ASI	1523	1

SBXI Subtract immediate from X

Indicators: CCA on X

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	1	0								

Immediate Operand

The immediate operand N is subtracted from the content of the Index register in integer form. The result replaces the Index register content. The value of N is expressed as a positive integer in the range 0 through 255.

X := X - N;

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	130	ASI	1523	1

MPYM Multiply TOS by memory

Indicators: CCA, Overflow
P±, DB±, Q±, S- relative
Direct or indirect
Indexing available

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	1	X	1										

Mode and Displacement

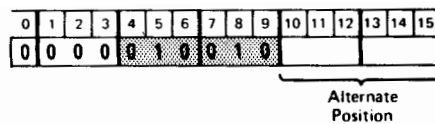
The TOS is multiplied in integer form by the content of the effective address memory location. The least significant word of the result replaces the operand on the TOS. An attempt to reference a location beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

(S) := (S) * (E);

COND	LUTA	LABEL	RAR	W
P REL.	047	AC1P	0027	1
DB REL.	044	AC1D	0013	1
Q REL.	044	AC1D	0013	1
S REL.	045	AC1S	0012	1
JLUI	046	MPYM	1700	1

MPY Multiply

Indicators: CCA, Overflow

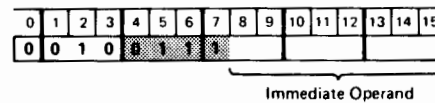


The top two words of the stack are multiplied in integer form, producing a single-length result. The two operand words are deleted and the least significant word of the double length product is pushed onto the stack. If the high order 17 bits of the double length product (including the sign bit of the second word) are not all zeros or all ones, Overflow is set.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) * (S);	SR<2	---	SRP	0004	-
S := S-1;	SR>=2	242	MPY	1705	1

MPYI Multiply immediate

Indicators: CCA on the new TOS, Overflow

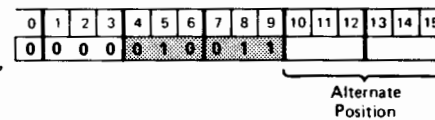


The immediate operand N is multiplied with the TOS in integer form; the 16-bit integer result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
(S) := (S) * N;	SR<2	---	SRP	0004	-
	SR>=2	134	MPYI	1703	0

DIV Divide

Indicators: CCA on quotient, Overflow

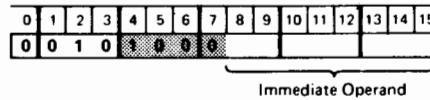


The integer in the second word of the stack is divided by the integer on the TOS. The top word is replaced by the remainder, and the second word is replaced by the quotient.

	COND	LUTA	LABEL	RAR	W
TEMP := (S-1)/(S);	SR<2	---	SRP	0004	-
(S) := (S-1) mod (S);	SR>=2	243	DIV	0577	1
(S-1) := TEMP;					

DIVI Divide immediate

Indicators: CCA on the new TOS

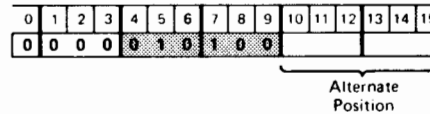


The immediate operand N is divided into the TOS in integer form; the 16-bit integer quotient replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255. Instruction traps if dividing by 0, causing Overflow indicator to set.

	COND	LUTA	LABEL	RAR	W
(S) := (S)/N;	UNC.	140	DIVI	0566	0

NEG Negate

Indicators: CCA, Overflow, Carry

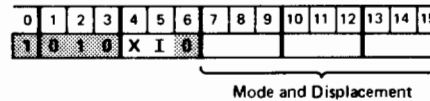


The integer in the TOS is replaced by its two's complement.

	COND	LUTA	LABEL	RAR	W
(S) := -(S);	SR<2	---	SRP	0004	-
	SR>=2	244	NEG	0465	1

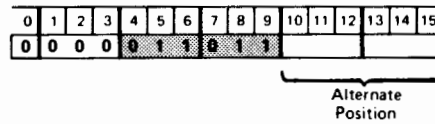
INCM Increment memory

Indicators: CCA, Carry, Overflow
DB+, Q±, S- relative
Direct or indirect
Indexing available



The content of the effective address memory location is incremented by one in integer form. In user mode, an attempt to reference a location beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

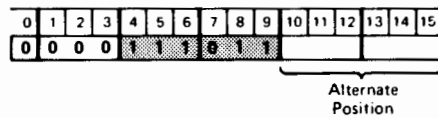
	COND	LUTA	LABEL	RAR	W
(E) := (E) + 1;	DB REL.	053	INCM	0035	1
	Q REL.	053	INCM	0035	1
	S REL.	053	INCM	0035	1
	JLUI	052	IDMY	0262	1

INCA Increment AIndicators: CCA, Carry,
Overflow

The TOS is incremented by one in integer form.

 $(S) := (S) + 1;$

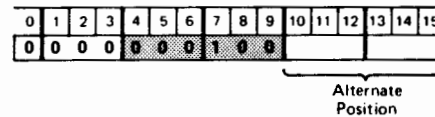
COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	253	INCA	0652	1

INCB Increment BIndicators: CCA, Carry,
Overflow

The second word of the stack is incremented by one in integer form. The TOS is unaffected.

 $(S-1) := (S-1) + 1;$

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	273	INCB	0654	1

INCX Increment XIndicators: CCA, Carry,
Overflow

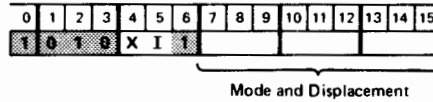
The content of the Index register is incremented by one in integer form.

 $X := X + 1;$

COND	LUTA	LABEL	RAR	W
SR<2	344	INCX	0656	1
SR>=2	304	INCX	0656	1

DECM Decrement memory

Indicators: CCA, Carry,
Overflow
DB+, Q±, S- relative
Direct or indirect
Indexing available



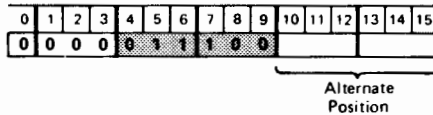
The content of the effective address memory location is decremented by one in integer form. In user mode, an attempt to reference a location beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

(E) := (E) - 1;

COND	LUTA	LABEL	RAR	W
DB REL.	050	AC1D	0013	1
Q REL.	050	AC1D	0013	1
S REL.	051	AC1S	0012	1
JLUI	052	IDMY	0262	1

DECA Decrement A

Indicators: CCA, Carry,
Overflow



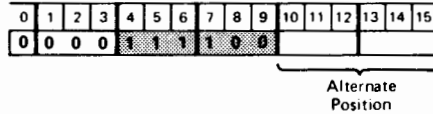
The TOS is decremented by one in integer form.

(S) := (S) - 1;

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	254	DECA	0651	1

DECB Decrement B

Indicators: CCA, Carry,
Overflow



The second word of the stack is decremented by one in integer form. The TOS is unaffected.

(S-1) := (S-1) - 1;

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	274	DECB	0653	1

DECX Decrement XIndicators: CCA, Carry,
Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	0	1						

Alternate
Position

The content of the Index register is decremented by one in integer form.

	COND	LUTA	LABEL	RAR	W
X := X - 1;	SR<2	345	DECX	0655	1
	SR>=2	305	DECX	0655	1

ZERO Push zero

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	1	0						

Alternate
Position

A zero word is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	SR<2	346	ZERO	0664	1
(S) := 0;	SR>=2	306	ZRO1	0663	1

ZROB Zero B

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	0	1						

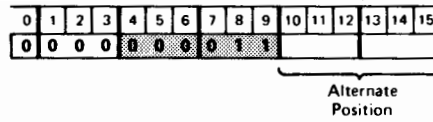
Alternate
Position

The second word of the stack is replaced by zero. The TOS is unaffected.

	COND	LUTA	LABEL	RAR	W
(S-1) := 0;	SR<2	361	SRP	0004	1
	SR>=2	321	ZROB	0666	1

ZROX Zero X

Indicators: unaffected



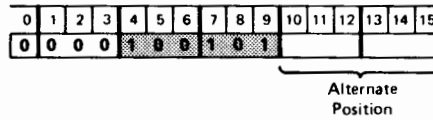
The content of the Index register is replaced by zero.

	COND	LUTA	LABEL	RAR	W
X := 0;	SR<2	343	ZROX	0665	1
	SR>=2	303	ZROX	0665	1



DUP Duplicate A

Indicators: CCA

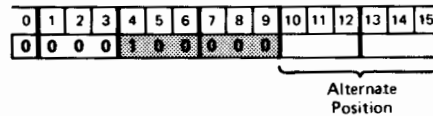


The top word of the stack is duplicated by pushing a copy of the TOS onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	SR<2	365	DUP	0671	1
(S) := (S-1);	SR>=2	325	DUP1	0670	1

DEL Delete A

Indicators: unaffected

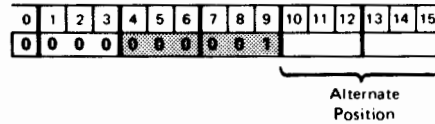


The top word of the stack is deleted.

	COND	LUTA	LABEL	RAR	W
S := S - 1;	SR<2	360	DEL1	0476	1
	SR>=2	320	DEL	0471	1

DELB Delete B

Indicators: unaffected

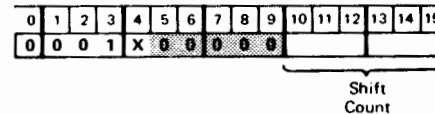


The second word of the stack is deleted and the stack is compressed. The content of the TOS is unchanged.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S);	SR<2	341	SRP	0004	1
S := S - 1;	SR>=2	301	DELB	0667	1

ASL Arithmetic shift left

Indicators: CCA

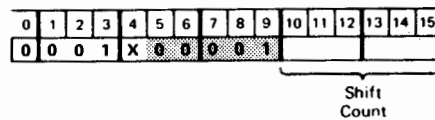


The TOS is shifted left n bits, preserving the sign bit. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

	COND	LUTA	LABEL	RAR	W
SR<2	---	---	SRP	0004	-
SR>=2	---	200	SHFL	0732	0

ASR Arithmetic shift right

Indicators: CCA



The TOS is shifted right n places, propagating the sign bit. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

	COND	LUTA	LABEL	RAR	W
SR<2	---	---	SRP	0004	-
SR>=2	---	201	SHFR	0724	0

TEST Test TOS

Indicators: CCA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	0	1						

Alternate Position

The condition code is set to pattern A according to the content of the TOS word.

	COND	LUTA	LABEL	RAR	W
CC := (S):0;	SR<2	---	SRP	0004	-
	SR>=2	245	TEST	0632	1

CPRB Compare range and branch

Indicators: CCE, CCL, CCG
P relative (+/-)
Direct or indirect

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	1	0	1	1	0	±						

Displacement

The integer in the Index register is tested to determine if it is within the interval defined by the upper bound integer on the TOS and the lower bound integer in the second word of the stack. The Condition Code is set by the comparison to a special pattern: CCE if within range, CCL if below range, CCG if above range. If the integer in the Index register is within the specified range, control is then transferred to P ± displacement L; otherwise to P+1. The top two elements of the stack are deleted in either case.

	COND	LUTA	LABEL	RAR	W
if (S-1) <= X <= (S) then	SR<2	---	SRP	0004	-
begin	SR>=2	226	CPRB	0405	1
CC := CCE;					
P := P + L;					
end else					
begin					
if X < (S-1) then CC := CCL;					
if X > (S) then CC := CCG;					
end;					
S := S - 2;					

FLT Float

Indicators: CCA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	1	1						

Alternate Position

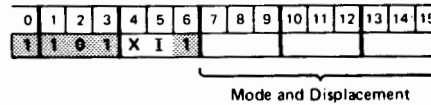
Converts the integer on the TOS to a 32-bit floating point number with rounding. The TOS is deleted and the doubleword floating point result is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	SR<2	367	FLT	1231	1
(S-1, S) := float (S-1);	SR>=2	327	FLT1	1226	1

DOUBLE INTEGER INSTRUCTIONS

LDD Load double

Indicators: CCA
 DB+, Q±, S- relative
 Direct or indirect
 (for final indirect:
 DB+ only)
 Doubleword indexing available

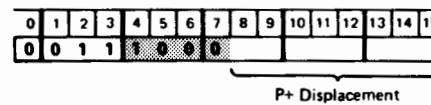


The contents of the effective address memory location (E) and the succeeding location (E + 1) are pushed onto the stack. The content of E, the most significant word, is loaded into B; the content of E + 1, the least significant word, is loaded into A. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative word address. If indexing is used, the effective address is obtained by adding the doubleword index in the Index register to the relative word address. In user mode, an attempt to load from beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 2;	DB REL.	064	AC3D	0062	1
(S-1, S) := (E, E+1);	Q REL.	064	AC3D	0062	1
	S REL.	065	AC3S	0061	1
	JLUI	066	LDD	0130	1

LDPP Load double from program, positive

Indicators: CCA
 P+ relative

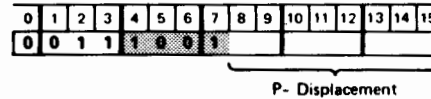


The double word contained at P+N is pushed onto the stack. An attempt to load from beyond the limit defined by PL will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	UNC.	142	LODP	0206	0
(S) := (P + N);					
S := S + 1;					
(S) := (P + N + 1);					

LDPN Load double from program, negative

Indicators: CCA
 P- relative

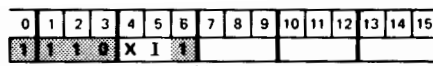


The double word contained at P-N is pushed onto the stack. An attempt to load from beyond the limit defined by PB will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	UNC.	146	LODP	0206	1
(S) := (P - N);					
S := S + 1;					
(S) := (P - N + 1);					

STD Store double

Indicators: unaffected
 DB+, Q+, S- relative
 Direct or indirect
 (for final indirect:
 DB+ only)
 Doubleword indexing available

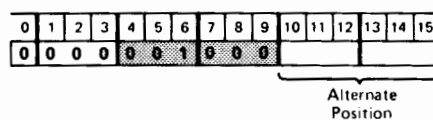


The top two words of the stack are stored into the effective address memory location (E) and the succeeding location (E + 1), and are then deleted from the Stack. The content of B, the most significant word, is stored into E; the content of A, the least significant word, is stored into E + 1. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative word address. If indexing is used, the effective address is obtained by adding the doubleword index in the Index register to the relative word address. In user mode, an attempt to store beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	CODE	LUTA	LABEL	RAR	W
(E, E+1) := (S-1, S);	DB REL.	070	AC4D	0102	1
S := S-2;	Q REL.	070	AC4D	0102	1
	S REL.	071	AC4S	0101	1
	JLUI	072	STD	0225	1

DCMP Double compare

Indicators: CCC

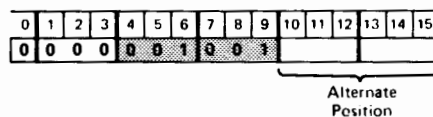


The Condition Code is set to pattern C as a result of the doubleword integer comparison of D,C and B,A. The two doublewords are deleted from the stack.

	CODE	LUTA	LABEL	RAR	W
CC := (S-3, S-2):(S-1, S);	SR<4	310	SRP	0004	1
S := S-4;	SR=4	350	DCMP	0554	1

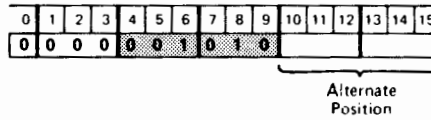
DADD Double add

Indicators: CCA, Carry, Overflow



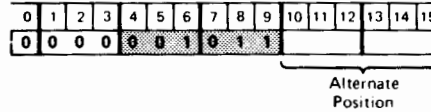
The two doubleword integers contained in the top four elements of the stack are added in double length integer form (D,C + B,A) and they are deleted. The doubleword integer sum is pushed onto the stack (B,A).

	CODE	LUTA	LABEL	RAR	W
(S-3, S-2) := (S-3, S-2) +	SR<4	311	SRP	0004	1
(S-1, S);	SR=4	351	DADD	0550	1
S := S-2;					

DSUB Double subtractIndicators: CCA, Carry,
Overflow

The doubleword integer contained in the top two words of the stack is subtracted from the doubleword integer contained in the third and fourth words of the stack (D,C - B,A). The top four words of the stack are deleted and the doubleword integer result is pushed onto the stack (B,A).

	COND	LUTA	LABEL	RAR	W
(S-3, S-2) := (S-3, S-2) -	SR<4	312	SRP	0004	1
(S-1, S);	SR=4	352	DSUB	0540	1
S := S - 2;					

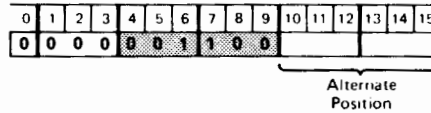
MPYL Multiply longIndicators: CCA, Carry,
Overflow

The top two words of the stack are multiplied in integer form. The words are replaced by the double length product, with the least significant half on the TOS. Overflow is cleared. Carry is cleared if the low order 16 bits represent the true result (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.

	COND	LUTA	LABEL	RAR	W
(S-1, S) := (S-1) * (S);	SR<4	313	SRP	0004	1
	SR=4	353	MPYL	1706	1

DIVL Divide long

Indicators: CCA, Overflow



The doubleword integer in the second and third elements of the stack is divided by the integer in the TOS (C,B ÷ A). The three words are deleted, and the quotient and remainder are pushed onto the stack (quotient in B, remainder in A).

	COND	LUTA	LABEL	RAR	W
TEMP1 := (S-2, S-1)/(S);	SR<4	314	SRP	0004	1
TEMP2 := (S-2, S-1) mod (S);	SR=4	354	DIVL	0604	1
S := S - 1;					
(S) := TEMP2;					
(S-1) := TEMP1;					

DNEG Double negateIndicators: CCA,
Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	0	1						

Alternate
Position

The doubleword integer contained in the top two words of the stack is negated (two's complemented) and replaces the original doubleword integer.

	COND	LUTA	LABEL	RAR	W
(S-1, S) := - (S-1, S);	SR<4	315	SRP	0004	1
	SR=4	355	DNEG	0544	1

DZRO Push double zero

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	1	1	1					

Alternate
Position

Two words containing all zeros are pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	SR<2	347	DZRO	0662	1
(S) := $\emptyset$;	SR>=2	307	ZRO2	0657	1
S := S + 1;					
(S) := $\emptyset$;					

DDUP Double duplicateIndicators: CCA on the
new TOS
double word

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	1	0						

Alternate
Position

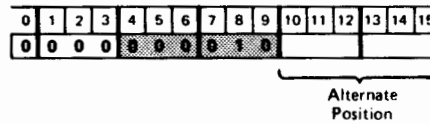
The doubleword in the top two words of the stack is duplicated by pushing a copy of it onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 2;	SR<2	366	DDUP	0676	1
(S-1, S) := (S-3, S-2);	SR>=2	326	DDP1	0673	1

DOUBLE INTEGER

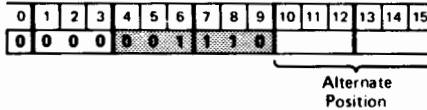
DDEL Double delete

Indicators: unaffected



The top two words of the stack are deleted.

	COND	LUTA	LABEL	RAR	W
S := S - 2;	SR<2	342	DDL1	0472	1
	SR>=2	302	DDEL	0770	1

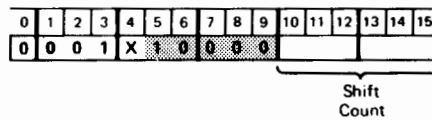
DXCH Double exchangeIndicators: CCA on the new
TOS double word

The top two doubleword pairs are interchanged on the stack.

	COND	LUTA	LABEL	RAR	W
TEMP := (S-1, S);	SR<4	316	SRP	0004	1
(S-1, S) := (S-3, S-2);	SR=4	356	DXCH	0633	1
(S-3, S-2) := TEMP;					

DASL Double arithmetic shift left

Indicators: CCA

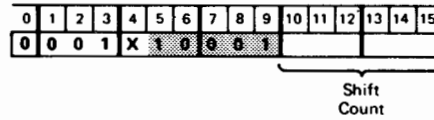


The double word contained in the top two words of the stack is shifted left n bits, preserving the sign bit (bit 0 of B). The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

	COND	LUTA	LABEL	RAR	W
SR<2	---		SRP	0004	-
SR>=2	220		SHDL	1400	0

DASR Double arithmetic shift right

Indicators: CCA

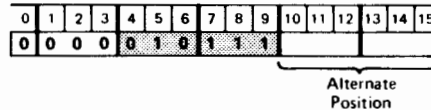


The double word contained in the top two words of the stack is shifted right n bits, propagating the sign bit (bit 0 of B). The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift ($n +$ the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	221	SHDR	1410	0

DTST Test double word on TOS

Indicators: CCA, Carry



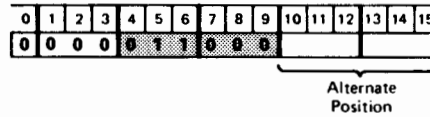
The condition code is set to pattern A according to the contents of the top two words of the stack. Also, Carry is cleared if the low order 16 bits of the doubleword result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	247	DTST	0636	1

CC := (S, S-1):0;

DFLT Double float

Indicators: CCA



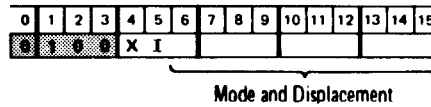
Converts the doubleword integer contained in the top two words of the stack to a floating point number with rounding.

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	250	DFLT	1234	1

(S-1, S) := dfloat((S-1, S));

LOGICAL INSTRUCTIONS

LOAD Load word onto stack

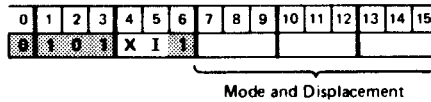


Indicators: CCA
 $P_{\pm}$, $DB_{\pm}$, $Q_{\pm}$, S_{-} relative
 Direct or indirect
 Indexing available

The content of the effective address location is pushed onto the stack. An attempt to load from beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
$S := S + 1;$	P REL.	023	AC1P	0027	1
$(S) := (E);$	DB REL.	020	AC1D	0013	1
	Q REL.	020	AC1D	0013	1
	S REL.	021	AC1S	0012	1
	JLUI	022	LOAD	0122	1

STOR Store TOS memory

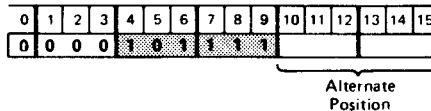


Indicators: unaffected
 $DB_{\pm}$, $Q_{\pm}$, S_{-} relative
 Direct or indirect
 Indexing available

The content of the TOS is stored into the effective address memory location, and is then deleted from the stack. In user mode, an attempt to store beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
$(E) := (S);$	DB REL.	024	AC1D	0013	1
	Q REL.	024	AC1D	0013	1
	S REL.	025	AC1S	0012	1
	JLUI	026	STOR	0220	1

LCMP Logical compare



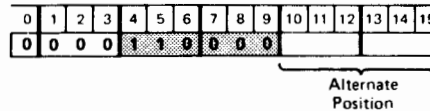
Indicators: CCC

The Condition Code is set to pattern C as a result of the comparison of the second word of the stack with the TOS. The two words are then deleted from the stack.

	COND	LUTA	LABEL	RAR	W
$CC := (S-1):(S);$	SR<4	337	SRP	0004	1
$S := S - 2;$	SR=4	377	LCMP	0466	1

LADD Logical add

Indicators: CCA (as a 2's complement result), Carry

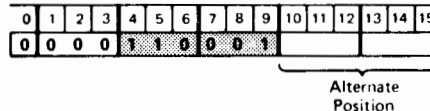


The top two words of the stack are added as 16-bit positive integers, and they are deleted from the stack. The resulting sum is pushed onto the stack. If the addition overflows bit 0, Carry is set.

	COND	LUTA	LABEL	RAR	W
(S-1) := ((S-1) + (S))	SR<2	---	SRP	0004	-
mod 2**16;	SR>=2	260	LADD	0504	1
S := S - 1;					

LSUB Logical subtract

Indicators: CCA (as a 2's complement result), Carry

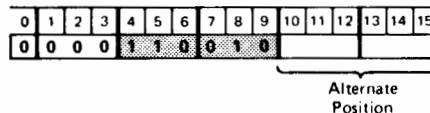


The top word of the stack is subtracted in logical form from the second word and they are deleted. The resulting difference is pushed onto the stack. If A<=B, Carry is set.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) - (S)	SR<2	---	SRP	0004	-
mod 2**16;	SR>=2	261	LSUB	0505	1
S := S - 1;					

LMPY Logical multiply

Indicators: CCA (as a 2's complement result), Carry



The top two words of the stack are multiplied as 16-bit positive integers. The words are replaced by the double length product with the least significant half on the TOS. Carry is cleared if the TOS word of the result represents the true logical value (i.e., if the high order 16 bits are all zeros); otherwise, Carry is set.

	COND	LUTA	LABEL	RAR	W
(S-1, S) := (S-1) * (S);	SR<2	---	SRP	0004	-
	SR>=2	262	LMPY	0511	1

LDIV Logical divide

Indicators: CCA on quotient
(as a 2's complement result),
Carry

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	0	1	1						

Alternate Position

The 32-bit positive integer in the second and third words of the stack is divided by the 16-bit positive integer on the TOS ($C, B \div A$). The top three words are deleted. The quotient is pushed onto the stack (B) and then the remainder (A). If overflow occurs, the quotient will be modulo $2^{*}16$.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
TEMP := logical ((S-2, S-1)/(S));	SR<2	---	SRP	0004	-
(S-1) := (S-2, S-1) mod (S);	SR>=2	263	LDIV	0517	1
(S-2) := TEMP;					
S := S - 1;					

NOT One's complement

Indicators: CCA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	1	0	0						

Alternate Position

The top word of the stack is converted to its one's complement.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
(S) := $\neg(S)$;	SR<2	---	SRP	0004	-
	SR>=2	264	NOT	0537	1

DUP Duplicate A

Indicators: CCA

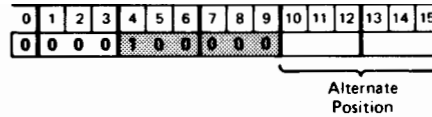
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	0	1						

Alternate Position

The top word of the stack is duplicated by pushing a copy of the TOS onto the stack.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
S := S + 1;	SR<2	365	DUP	0671	1
(S) := (S-1);	SR>=2	325	DUP1	0670	1

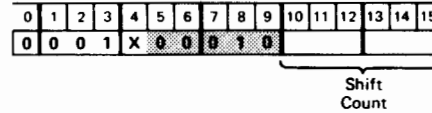
DEL Delete A.
Indicators: unaffected



The top word of the stack is deleted.

COND	LUTA	LABEL	RAR	W
SR<2	360	DEL1	0476	1
SR>=2	320	DEL	0471	1

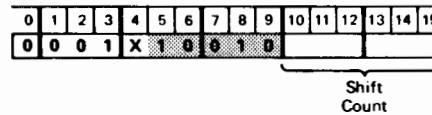
LSL Logical shift left
Indicators: CCA



The TOS is shifted left n bits logically. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	202	SHFL	0732	0

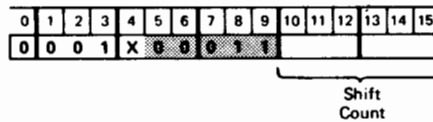
DLSL Double logical shift left
Indicators: CCA



The double word contained in the top two words of the stack is shifted left n bits logically. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	222	SHDL	1400	0

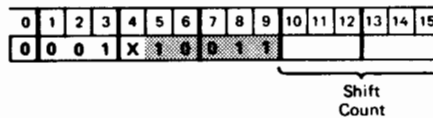
LSR Logical shift right
Indicators: CCA



The TOS is shifted right n bits logically. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	203	SHFR	0724	0

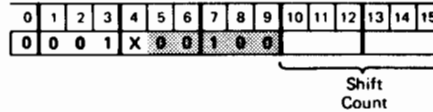
DLSR Double logical shift right
Indicators: CCA



The double word contained in the top two words of the stack is shifted right n bits logically. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	223	SHDR	1410	0

CSL Circular shift left
Indicators: CCA

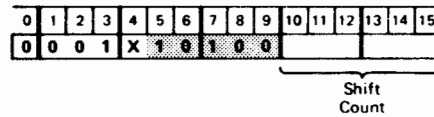


The TOS is shifted left n bits circularly. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	204	SHFL	0732	0

DCSL Double circular shift left

Indicators: CCA

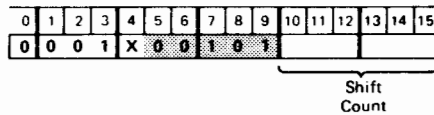


The double word contained in the top two words of the stack is shifted left n bits circularly. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift ($n +$ the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	224	SHDL	1400	0

CSR Circular shift right

Indicators: CCA

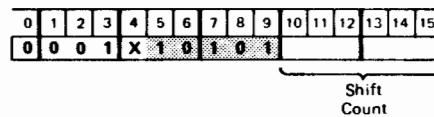


The TOS is shifted right n bits circularly. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift ($n +$ the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	205	SHFR	0724	0

DCSR Double circular shift right

Indicators: CCA



The double word contained in the top two words of the stack is shifted right n bits circularly. The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift ($n +$ the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	225	SHDR	1410	0

LOGICAL

OR	Logical OR	Indicators: CCA on the new TOS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			0	0	0	0	1	1	0	1	0	1						

Alternate Position

The top two words of the stack are merged by a logical inclusive-OR. The two words are deleted and the result is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) or (S);	SR<2	---	SRP	0004	-
S := S-1;	SR>=2	265	OR	0646	1

ORI	Logical OR immediate	Indicators: CCA	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			0	0	1	1	1	1	0	1								

Immediate Operand

The immediate operand N is expanded to 16 bits with high order zeros and merged (inclusive OR) with the TOS; the result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
(S) := (S) or N;	SR<2	---	SRP	0004	-
	SR>=2	166	ORI	1524	0

XOR	Logical exclusive-OR	Indicators: CCA on the new TOS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			0	0	0	0	1	1	0	1	1	0						

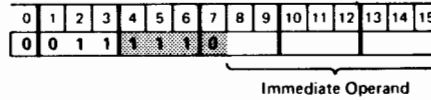
Alternate Position

The top two words of the stack are combined by a logical exclusive-OR. The two words are deleted and the result is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
(S-1) := $\neg$ (S-1) and (S)	SR<2	---	SRP	0004	-
or (S-1) and $\neg$ (S);	SR>=2	266	XOR	0647	1
S := S-1;					

XORI Logical exclusive OR immediate

Indicators: CCA

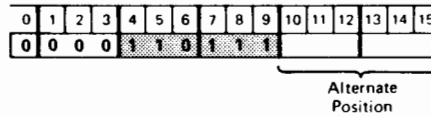


The immediate operand N is expanded to 16 bits with high order zeros and is combined by exclusive OR with the TOS; the result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
(S) := $\neg$ (S) and N;	SR<2	---	SRP	0004	-
or (S) and $\neg$ N;	SR>=2	172	XORI	1525	0

AND Logical AND

Indicators: CCA on the new TOS

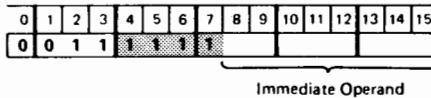


The top two words of the stack are combined by a logical AND. The two words are deleted and the result is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
(S-1) := (S-1) and (S);	SR<2	---	SRP	0004	-
S := S - 1;	SR>=2	267	AND	0650	1

ANDI Logical AND immediate

Indicators: CCA



The immediate operand N is expanded to 16 bits with high order zeros and is combined by logical AND with the TOS; the result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
(S) := (S) and N;	SR<2	---	SRP	0004	-
	SR>=2	176	ANDI	1526	0

BIT, FIELD, AND BYTE INSTRUCTIONS

SCAN Scan bits

Indicators: CCA on
final TOS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	1	1	0	0	0	0	0	0	0

Reserved

The TOS is shifted left until bit 0 contains a "1", then is shifted left one more bit. The shift count is left in the Index register, indicating the bit position which contained the "1". The instruction normally sets the Index register to -1 before beginning the shifts. However, if X is specified, the shift count adds on to the existing Index register content. If TOS is all zeros, the count will be 16 if unindexed, or X + 16 if indexed.

	COND	LUTA	LABEL	RAR	W
if CIR (4) = 0 then X := -1;	SR<2	---	SRP	0004	-
if (S) = 0 then	SR>=2	206	SCAN	0740	1

```

begin
  while (S(0)) = 0 then
    begin
      logical shift left A by 1;
      X := X + 1;
    end;
    logical shift left A by 1;
    X := X + 1;
  end if CIR(4) = 0 then X := 16
  else X := X + 16

```

TBC Test bit and set

Condition Code

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	0	1	0						

Bit Position

Indicators: CCE if the bit
was "0"
CCL or CCG if
the bit was "1"

One bit of the TOS word is tested and the Condition Code is set to a special pattern depending on the state of the bit. The bit position to be tested is specified by the argument field of the instruction plus, if X is specified, the content of the Index register. If the number specified exceeds 15, the bit position indicated is modulo 16; e.g., bit 0 is tested for counts of 0, 16, 32, 48, etc.

	COND	LUTA	LABEL	RAR	W
SR<2	---	---	SRP	0004	-
SR>=2		232	TBC	1465	0

TCBC Test and complement bit,

set Condition Code

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	1	0	1						

Bit Position

Indicators: CCE if the bit
was "0"
CCL or CCG if
the bit was "1"

One bit of the TOS word is tested and the Condition Code is set to a special pattern depending on the state of the bit. The bit is complemented after the test. The bit position to be tested is specified by the argument field of the instruction plus, if X is specified, the content of the Index register. If the number specified exceeds 15, the bit position indicated is modulo 16; e.g., bit 0 is tested for counts of 0, 16, 32, 48, etc.

	COND	LUTA	LABEL	RAR	W
SR<2	---	---	SRP	0004	-
SR>=2		235	TCBC	1473	0

TRBC Test and reset bit,
set Condition Code

Indicators: CCE if the bit
was "0"
CCL or CCG if
the bit was "1"

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	0	1	1						

Bit Position

One bit of the TOS word is tested and the Condition Code is set to a special pattern depending on the state of the bit. The bit is reset to "0" after the test. The bit position to be tested is specified by the argument field of the instruction plus, if X is specified, the content of the Index register. If the number specified exceeds 15, the bit position indicated is modulo 16; e.g., bit 0 is tested for counts of 0, 16, 32, 48, etc.

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	233	TRBC	1467	0

TSBC Test and set bit,
set Condition Code

Indicators: CCE if the bit
was "0"
CCL or CCG if
the bit was "1"

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	1	0	0						

Bit Position

One bit of the TOS word is tested and the Condition Code is set to a special pattern depending on the state of the bit. The bit is set to "1" after the test. The bit position to be tested is specified by the argument field of the instruction plus, if X is specified, the content of the Index register. If the number specified exceeds 15, the bit position indicated is modulo 16; e.g., bit 0 is tested for counts of 0, 16, 32, 48, etc.

COND	LUTA	LABEL	RAR	W
SR<2	---	SRP	0004	-
SR>=2	234	TSBC	1471	0

TSBM Test and set bits in memory,
set Condition Code

Indicators: CCA on the
new TOS
DB+ relative

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	1	0	0								

Displacement
DB+ Relative

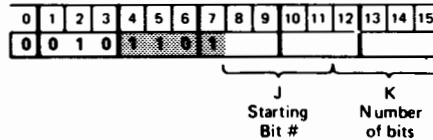
A mask word on the TOS is compared by logical AND with the contents of the memory location specified by DB + displacement; the result replaces the TOS and sets the Condition Code to pattern A. At the same time, the mask word is merged (logical OR) with the content of the specified location and this result is stored back in the memory cell. Displacement range is 0 through +255. (The memory location is set to all "1s" during execution until the merged result is stored back in the cell; solves the hardware "semaphore" problem with dual CPU configurations.)

	COND	LUTA	LABEL	RAR	W
TEMP := (DB + N); <<fetch memory cell>>	SR<2	---	SRP	0004	-
(DB + N) := 2**16-1; <<set memory cell to all ones>>	SR>=2	162	TSBM	1443	0
(DB + N) := TEMP or (S) <<set bits>>					
(S) := TEMP and (S) <<get bits>>					

BIT, FIELD, BYTE

EXF Extract field

Indicators: CCA on the new TOS

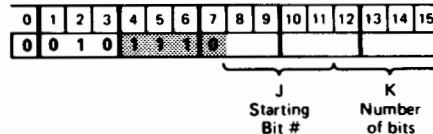


A specified set of bits in the TOS are extracted and right justified, and the result, with high order zeros, replaces the TOS. The J field specifies the starting (leftmost) bit number in the source field, and the K field specifies the number of bits to be extracted.

	COND	LUTA	LABEL	RAR	W
$(S(16-K:15)) := (S(J:J+K-1));$	SR<2	---	SRP	0004	-
$(S(0:15-K)) := 0;$	SR>=2	164	EXF	1475	0

DPF Deposit field

Indicators: CCA on the new TOS

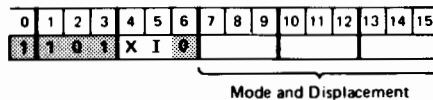


A specified number of the least significant bits of the TOS are deposited in the second word of the stack, beginning at the bit number specified by the J field; the remaining bits of the second word of the stack are unchanged. The K field specifies the number of bits to be deposited. The source operand is deleted from the stack.

	COND	LUTA	LABEL	RAR	W
$(S-1(J:J+K-1)) := (S(16-K:15));$	SR<2	---	SRP	0004	-
$(S-1(0:J)) := (S-1(0:J));$	SR>=2	170	DPF	1501	0
$(S-1(J+K:15)) := (S-1(J+K:15));$					
$S := S - 1;$					

LDB Load byte

Indicators: CCB
 Byte addressing
 DB+, Q+, S- relative
 Direct or indirect
 (for final indirect: DB+ only)
 Byte indexing available

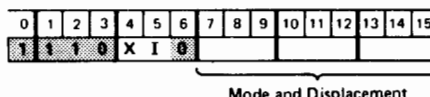


The content of the effective byte address memory location is loaded into the right half of the TOS. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative byte address. If indexing is used, the effective byte address is obtained by adding the byte index in the Index register to the relative byte address. In user mode, an attempt to load from beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
BASE := DB, Q, or S;	DB REL.	067	LDB	0143	0
E := BASE + D + if indexed	Q REL.	067	LDB	0143	0
and not INDIRECT then	S REL.	067	LDB	0143	0
X/2 else 0;					
FLAG := if indexed and (X(15) = 1) then TRUE else FALSE;					
TRY AGAIN:					
if not (DL <= E <= S) then					
if (DL <= DB <= Z) then <<not split stack>>					
begin					
E := E + %1000000;					
if not (DL <= E <= S) then if user mode then BOUNDS VIOLATION TRAP;					
end;					
if INDIRECT then					
begin					
E := ((E) + if INDEXED then X else 0)/2 + DB;					
FLAG := if ((E) + if INDEXED then X else 0) MOD2 = 1					
then TRUE else FALSE;					
INDIRECT := FALSE;					
go TRY AGAIN;					
end;					
S := S + 1;					
(S (0:7)) := 0;					
(S (8:15)) := if FLAG then (E (8:15)) else (E (0:7))					

STB Store byte

Indicators: unaffected
 Byte addressing
 DB+, Q+, S- relative
 Direct or indirect
 (for final indirect: DB+ only)
 Byte indexing available



The right byte (bits 8 through 15) of the TOS is stored into the effective byte address memory location and the TOS is deleted. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative byte address. If indexing is used, the effective byte address is obtained by adding the byte index in the Index register to the relative byte address. In user mode, an attempt to store beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
BASE := DB, Q, or S;	DB REL.	073	LDB	0143	0
E := BASE + D + if indexed	Q REL.	073	LDB	0143	0
and not INDIRECT then	S REL.	073	LDB	0143	0
X/2 else 0;					
FLAG := if indexed and (X(15) = 1) then TRUE else FALSE;					
TRY AGAIN:					
if not (DL <= E <= S) then					
if (DL <= DB <= Z) then <<split stack>>					
begin					
E := E + %1000000;					
if not (DL <= E <= S) then if user mode then BOUNDS VIOLATION TRAP;					
end;					
if INDIRECT then					
begin					
E := ((E) + if INDEXED then X else 0)/2 + DB;					
FLAG := if ((E) + if INDEXED then X else 0) MOD2 = 1					
then TRUE else FALSE;					
INDIRECT := FALSE;					
go TRY AGAIN;					
end;					
if FLAG then (E (8:15)) := (S (8:15)) else (E (0:7)) := (S (8:15));					
S := S - 1;					

CMPB Compare bytes
Byte addressing
DB+ or PB+ for source
DB+ for target

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	0	1					

PB/DB SDEC

Indicators: CCE if byte count = 0
CCG if target byte > source byte (final)
CCL if target byte < source byte (final)

This instruction scans two byte strings simultaneously until the compared bytes are unequal or until a specified number of comparisons have been made. CMPB expects a signed byte count in A, a DB or PB relative displacement for a source byte address in B, and a DB relative displacement for a target byte address in C. As long as the word count in A has not been counted to zero, the comparison proceeds as follows: The content of the byte address location specified by DB + B or PB + B is compared with the content of the byte address location specified by DB + C. If the byte count in A is positive, the source and target displacement values in B and C are incremented by one after each comparison, and the byte count is decremented by one. If the byte count in A is negative, the source and target displacement values in B and C are decremented by one after each comparison, and the byte count is incremented by one. Note that the byte count is always changed by one toward zero. The instruction terminates when either a comparison fails or the byte count in the TOS reaches zero. The Condition Code is set to a special pattern to indicate the terminating condition. On termination, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

	COND	LUTA	LABEL	RAR	W
While (S) $\neg$ = $\emptyset$ do	P REL.	151	CPBP	2023	0
Begin	DB REL.	155	CPBD	2024	0
BYTEMOVEADDRESS ((S - 1),					
(S - 2), S - 3);					
if (BTAD) $\neg$ = (BSAD) then GO TO N;					
(S - 2) := (S - 2) + Sign (S);					
(S - 1) := (S - 1) + Sign (S);					
(S) := (S) - Sign (S);					
end;					
N:CC := if (S) = $\emptyset$ then CCE else					
if (BTAD) > (BSAD) then CCG else CCL;					
S := S - SDEC;					

BTST Test byte on TOS

Indicators: CCB

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	0	1						

Alternate Position

The Condition Code is set to pattern B according to the contents of the byte contained in the eight least significant bits of the TOS word (bits 8-15).

COND	LUTA	LABEL	RAR	W
SR < 2	---	SRP	0004	-
SR >= 2	251	BTST	0644	1

SCW	Scan while memory bytes equal test byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0	0	1	0	0	0	0	0	0	1	0	1	1	1	1	
Indicators: Carry																	
CCB on the last character scanned																	
Byte addressing, DB+																	
		SDEC															

The SCW instruction expects the TOS to contain a test character in the right byte and a terminal character in the left byte. The second word of the stack contains a DB relative displacement for a source byte address. The source byte is tested against the test character. If they are equal the source byte address is incremented and the next byte is tested. This continues until a source byte is found that is not the same as the test character. If the last character scanned is the same as the terminal character, the Carry bit is set; if not, the Carry bit is cleared. On completion of the scan, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified in the SDEC field of the instruction.

	COND	LUT	LABEL	RAR	W
R:BSAD := (S - 1) + 2 * DB;	UNC.	125	SCAM	2126	0
<<17 bit source byte address>>					
SAD := BSAD/2;					
if not (DL <= SAD <= S - 2) then					
if (DL <= DB <= Z) then <<not a split stack>>					
Begin					
SAD := SAD + %1000000;					
if not (DL <= SAD <= S - 2) then INTERRUPT (seg %11);					
end;					
if (BSAD) = (S (8:15)) then					
Begin					
(S - 1) := (S - 1) + 1;					
GO TO R;					
end;					
CARRY := if (BSAD) = (S (0:7)) then 1 else 0;					
CC := CCB on (BSAD);					
S := S - SDEC;					

SCU	Scan until memory byte equals test byte or terminal byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	
	Indicators: Carry																
	Byte addressing DB+																
		SDEC															

The SCU instruction expects the TOS to contain a test character in the right byte and a terminal character in the left byte. The second word of the stack contains a DB relative displacement for a source byte address. The source byte is tested against the test and terminal characters. If the source byte differs from both of these characters, the byte address is incremented and the next byte is tested. This continues until either the test character or the terminal character is encountered. The address of the character remains in the second word of the stack. If the last character scanned was the same as the test character, the Carry bit is cleared; if it was the same as the terminal character, Carry is set. On completion of the scan, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified in the SDEC field of the instruction.

BIT, FIELD, BYTE

	COND	LUTA	LABEL	RAR	W
R:BSAD := (S - 1) + 2 * DB;	UNC.	135	SCAM	2126	0

```

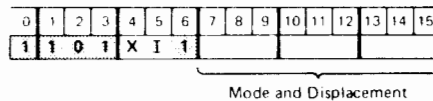
<<17 bit source
byte address>>
SAD := BSAD/2;
if not (DL <= SAD <= S - 2) then
  if (DL <= DB <= Z) then <<not a split stack>>
    Begin
      SAD := SAD + %1000000;
      if not (DL <= SAD <= S - 2) then INTERRUPT (seg %11);
    end;
  if (BSAD) ^ = (S (8:15)) and (BSAD) ^ = (S (0:7)) then
    Begin
      (S - 1) := (S - 1) + 1;
      GO TO R;
    end;
  CARRY := if (BSAD) = (S (0:7)) then 1 else 0;
  S := S - SDEC;

```

FLOATING POINT INSTRUCTIONS

LDD Load double

Indicators: CCA
DB+, Q3, Q-, S- relative
Direct or indirect (for final
indirect: DB+ only)
Doubleword indexing available

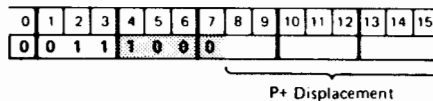


The contents of the effective address memory location (E) and the succeeding location (E + 1) are pushed onto the stack. The content of E, the most significant word, is loaded into B; the content of E + 1, the least significant word, is loaded into A. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative word address. If indexing is used, the effective address is obtained by adding the doubleword index in the Index register to the relative word address. In user mode, an attempt to load from beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 2;	DB REL.	064	AC3D	0062	1
(S-1,S) := (E, E+1);	Q REL.	064	AC3D	0062	1
	S REL.	065	AC3S	0061	1
	JLUI	066	LDD	0130	1

LDPP Load double from program, positive

Indicators: CCA
P+ relative

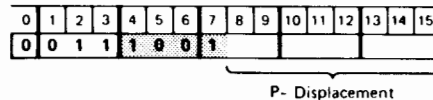


The doubleword contained at P+N is pushed onto the stack. An attempt to load from beyond the limit defined by PL will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	UNC.	142	LODP	0206	0
(S) := (P + N);					
S := S + 1;					
(S) := (P + N + 1);					

LDPN Load double from program, negative

Indicators: CCA
P- relative



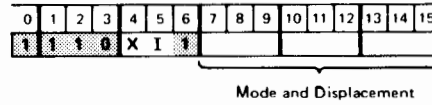
The doubleword contained at P-N is pushed onto the stack. An attempt to load from beyond the limit defined by PB will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	UNC.	146	LODP	0206	1
(S) := (P - N);					
S := S + 1;					
(S) := (P - N + 1);					

FLOATING POINT

STD Store double

Indicators: unaffected
 DB+, Q±, S- relative
 Direct or indirect (for final
 indirect: DB+ only)
 Doubleword indexing available

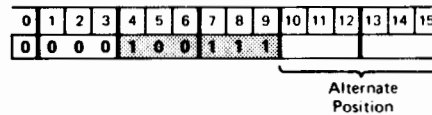


The top two words of the stack are stored into the effective address memory location (E) and the succeeding location (E + 1), and are then deleted from the stack. The content of B, the most significant word, is stored into E + 1. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative word address. If indexing is used, the effective address is obtained by adding the doubleword index in the Index register to the relative word address. In user mode, an attempt to store beyond the limits defined by DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
(E, E+1) := (S-1, S);	DB REL.	070	AC4D	0102	1
S := S-2;	Q REL.	070	AC4D	0102	1
	S REL.	071	AC4S	0101	1
	JLUI	072	STD	0225	1

FLT Float

Indicators: CCA

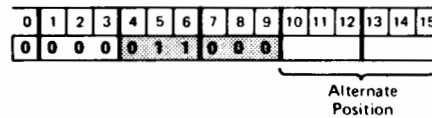


Converts the integer on the TOS to a 32-bit floating point number with rounding. The TOS is deleted and the doubleword floating point result is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	SR<2	367	FLT	1231	1
(S-1,S) := float (S-1);	SR>=2	327	FLT1	1226	1

DFLT Double float

Indicators: CCA

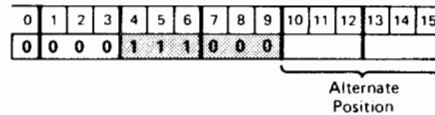


Converts the doubleword integer contained in the top two words of the stack to a floating point number with rounding.

	COND	LUTA	LABEL	RAR	W
(S-1,S) := dfloat(S-1,S);	SR<2	—	SRP	0004	—
	SR>=2	250	DFLT	1234	1

FIXR Fix and round

Indicators: CCA, Carry, Overflow

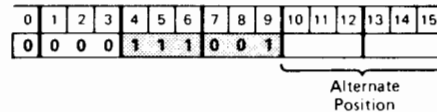


The floating point number contained in the top two words of the stack is converted to fixed point form and rounded. Carry is cleared if the low order 16 bits of the doubleword result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.

	COND	LUTA	LABEL	RAR	W
(S-1,S) := trunc((S-1,S) + .5*sign(S-1,S));	SR<2 SR>=2	-- 270	SRP FIXR	0004 1277	-- 1

FIXT Fix and truncate

Indicators: CCA, Carry, Overflow

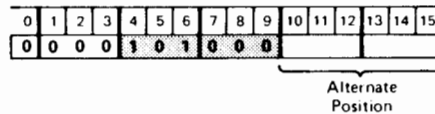


The floating point number contained in the top two words of the stack is converted to fixed point form and truncated. Carry is cleared if the low order 16 bits of the doubleword result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.

	COND	LUTA	LABEL	RAR	W
(S-1,S) := trunc((S-1,S));	SR<2 SR>=2	-- 271	SRP FIXT	0004 1276	-- 1

FCMP Floating compare

Indicator's: CCC

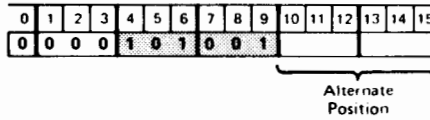


The Condition Code is set to pattern C as a result of the floating point comparison of D,C with B,A. The two floating point doublewords are deleted.

	COND	LUTA	LABEL	RAR	W
CC := (S-3, S-2):(S-1,S); S := S - 4;	SR<4 SR=4	330 370	SRP FCMP	0004 1255	1 1

FADD Floating add

Indicators: CCA, Overflow

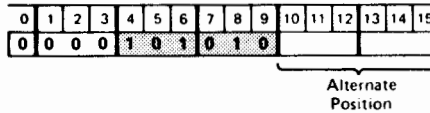


The two floating point numbers contained in the top four words of the stack are added in floating point form. The top four words of the stack are deleted and the two-word sum is pushed onto the stack.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
(S-3, S-2) := (S-3, S-2) +	SR<4	331	SRP	0004	1
(S-1, S);	SR=4	371	FADD	1001	1
S := S - 2;					

FSUB Floating subtract

Indicators: CCA, Overflow

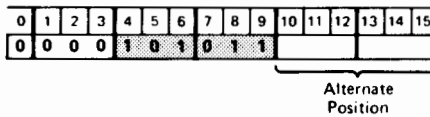


The floating point number contained in the top two words of the stack is subtracted in floating point form from the floating point number contained in the third and fourth words of the stack. The top four words of the stack are deleted and the two-word difference is pushed onto the stack. Overflow can indicate either underflow or overflow.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
(S-3, S-2) := (S-3, S-2) -	SR<4	332	SRP	0004	1
(S-1, S);	SR=4	372	FSUB	1000	1
S := S - 2;					

FMPY Floating multiply

Indicators: CCA, Overflow



The two floating point numbers contained in the top four words of the stack are multiplied in floating point form. The top four words of the stack are deleted and the two-word result is pushed onto the stack.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
(S-3, S-2) := (S-3, S-2) *	SR<4	333	SRP	0004	1
(S-1, S);	SR=4	373	FMPY	1110	1
S := S - 2;					

FDIV Floating divide

Indicators: CCA, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	0	0						

Alternate
Position

The floating point number contained in the third and fourth words of the stack is divided by the floating point number contained in the top two words of the stack. The top four words of the stack are deleted and the two-word quotient is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
(S-3, S-2) := (S-3, S-2) /	SR<4	334	SRP	0004	1
(S-1, S);	SR=4	374	FDIV	1141	1
S := S - 2;					

**FNEG** Floating negate

Indicators: CCA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	0	1						

Alternate
Position

The floating point number contained in the top two words of the stack is negated in floating point form.

	COND	LUTA	LABEL	RAR	W
(S-1, S) := -(S-1, S);	SR<4	335	SRP	0004	1
	SR=4	375	FNEG	1104	1

DDUP Double duplicateIndicators: CCA on new
TOS doubleword

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	1	0						

Alternate
Position

The doubleword in the top two words of the stack is duplicated by pushing a copy of it onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 2;	SR<2	366	DDUP	0676	1
(S-1, S) := (S-3, S-2);	SR>=2	326	DDP1	0673	1

DDEL Double delete

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	1	0						

Alternate
Position

The top two words of the stack are deleted.

	COND	LUTA	LABEL	RAR	W
S := S -2;	SR<2	342	DDL1	0472	1
	SR>=2	302	DDEL	0770	1

DTST Test doubleword on TOS

Indicators: CCA, Carry

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	1	1						

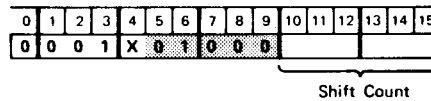
Alternate
Position

The condition code is set to pattern A according to the contents of the top two words of the stack. Also, Carry is cleared if the lower order 16-bits of the doubleword result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.

	COND	LUTA	LABEL	RAR	W
CC := (S, S-1):0;	SR<2	--	SRP	0004	--
	SR>=2	247	DTST	0636	1

THE FOLLOWING THREE INSTRUCTIONS ARE EXTENDED PRECISION INSTRUCTIONS.

TASL Triple arithmetic shift left

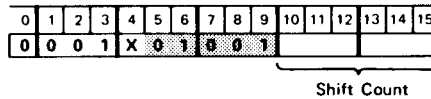


Indicators: CCA on the new TOS triple word

The triple word contained in the top three words of the stack is shifted left n bits, preserving the sign bit (bit 0 of C). The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	--	SRP	0004	--
SR>=2	210	TASL	1417	0

TASR Triple arithmetic shift right

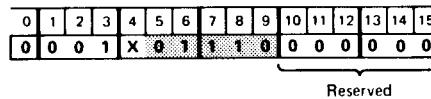


Indicators: CCA on the new TOS triple word

The triple word contained in the top three words of the stack is shifted right n bits, propagating the sign bit (bit 0 of C). The value of n (modulo 64) is the number specified in the argument field. If X is specified, then shift (n + the content of the Index register).

COND	LUTA	LABEL	RAR	W
SR<2	--	SRP	0004	--
SR>=2	211	TASR	1432	0

TNSL Triple normalizing shift left



Indicators: CCA on final value of top three words

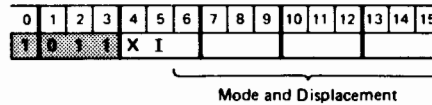
Bits 0 through 5 of C are cleared and the top three words of the stack are shifted left arithmetically until bit 6 of C is a "1". The shift count is stored in the Index register. The instruction initially clears the Index register unless X is specified ("1" in bit 4 of the instruction).

COND	LUTA	LABEL	RAR	W
if CIR(4) = 0 then X := 0;	SR<2	SRP	0004	--
(S-2(0:5)) := 0	SR>=2	216	TNSL	1342
if (S-2, S-1, S) = 0 then				1
begin				
while (S-2(6)) = 0 do				
begin				
X := X+1;				
arithmetic left shift 1 of (S-2, S-1, S);				
end;				
end else X := X + 42;				

FLOATING POINT

INDEX, EXCHANGE, LOOP CONTROL, AND BRANCH INSTRUCTIONS

LDX Load Index

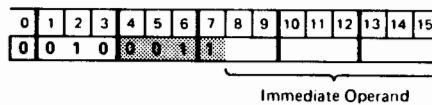


Indicators: CCA
P±, DB±, Q±, S- relative
Direct or indirect
Indexing available

The content of the effective address memory location is loaded into the Index register. An attempt to load from beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

	COND	LUTA	LABEL	RAR	W
X := (E);	P REL.	057	AC1P	0027	1
	DB REL.	054	AC1D	0013	1
	Q REL.	054	AC1D	0013	1
	S REL.	055	AC1S	0012	1
	JLUI	056	LDX	0121	1

LDXI Load X immediate

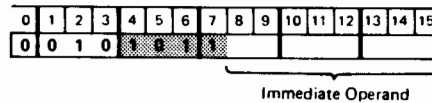


Indicators: unaffected

The Index register is loaded with the immediate operand N. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
X := N;	UNC.	114	XPNI	1527	0

LDXN Load X negative immediate



Indicators: unaffected

The Index register is loaded with the 16-bit two's complement of the immediate operand N. The value of N is expressed as a positive integer in the range 0 through 255.

	COND	LUTA	LABEL	RAR	W
X := -N;	UNC.	154	XPNI	1527	1

STAX Store A into X
Indicators: CCA on the new X

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	1	1						

Alternate Position

The TOS replaces the content of the Index register, and TOS is deleted from the stack.

	COND	LUTA	LABEL	RAR	W
X := (S);	SR<2	363	SRP	0004	1
S := S-1;	SR>=2	323	STAX	0716	1

STBX Store B into X
Indicators: CCA on the new X

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	1	0						

Alternate Position

The second word of the stack replaces the content of the Index register.

	COND	LUTA	LABEL	RAR	W
X := (S-1);	SR<2	—	SRP	0004	—
	SR>=2	246	STBX	0717	1

LDXA Load X onto stack
Indicators: CCA on the new TOS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	0	0						

Alternate Position

The content of the Index register is pushed onto the stack.

	COND	LUTA	LABEL	RAR	W
S := S + 1;	SR<2	364	LDXA	0714	1
(S) := X;	SR>=2	324	LDX1	0713	1

LDXB Load X into B
 Indicators: CCA on the new B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	1	0						

Alternate Position

The second word of the stack is replaced by the content of the Index register. The TOS is unaffected.

	COND	LUTA	LABEL	RAR	W
(S-1) := X;	SR<2	362	SRP	0004	1
	SR>=2	322	LDXB	0715	1

ADAX Add A to X
 Indicators: CCA on the new X, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	0						

Alternate Position

The TOS is added in integer form to the content of the Index register. The sum replaces the content of the Index register, and the TOS is deleted.

	COND	LUTA	LABEL	RAR	W
X := X + (S);	SR<2	--	SRP	0004	--
S := S - 1;	SR>=2	256	ADAX	0720	1

ADXA Add X to A
 Indicators: CCA on the new TOS, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	1						

Alternate Position

The content of the Index register is added to the TOS, and the sum replaces the TOS.

	COND	LUTA	LABEL	RAR	W
(S) := (S) + X;	SR<2	--	SRP	0004	--
	SR>=2	257	ADXA	0722	1

ADBX Add B to X

Indicators: CCA on the new X, Carry, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	1	0						

Alternate Position

The second word of the stack is added in integer form to the content of the Index register, and the result replaces the content of the Index register.

	COND	LUTA	LABEL	RAR	W
$X := X + (S-1);$	SR<2 SR>=2	-- 276	SRP ADBX	0004 0721	-- 1

ADXB Add X to B

Indicators: CCA on the new B, Carry Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	1	1						

Alternate Position

The content of the Index register is added in integer form to the second word of the stack, and the sum replaces the second word of the stack.

	COND	LUTA	LABEL	RAR	W
$(S-1) := (S-1) + X;$	SR<2 SR>=2	-- 277	SRP ADXB	0004 0723	-- 1

IXBZ Increment X, branch is zero

Indicators: CCA, Carry, Overflow
P relative (+/-)
Direct or indirect

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	1	0	1	0	±					

Displacement

The Index register is incremented. If the result is then zero, control is transferred to $P \pm$ displacement; otherwise to $P+1$.

	COND	LUTA	LABEL	RAR	W
$X := X + 1;$ if $X = 0$ then $P := P + L$ else $P := P + 1;$	SR<2 SR>=2	-- 212	SRP IXBZ	0004 3401	-- 1

DXBZ Decrement X, branch if zero

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	1	0	1	1	±					

Displacement

Indicators: CCA, Carry,
Overflow

P relative (+/-)

Direct or indirect

The Index register is decremented. If the result is then zero, control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
X := X - 1;	SR<2	--	SRP	0004	--
if X = 0 then P := P + L else P := P + 1;	SR>=2	213	DXBZ	3404	1

IABZ Increment A, branch if zero

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	0	1	1	1	±					

Displacement

Indicators: CCA, Carry,
Overflow

P relative (+/-)

Direct or indirect

The TOS is incremented. If the result is then zero, control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
(S) := (S) + 1;	SR<2	--	SRP	0004	--
if (S) = 0 then P := P + L else P := P + 1;	SR>=2	207	IABZ	2363	1

DABZ Decrement A, branch if zero

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	0	1	1	1	±					

Displacement

Indicators: CCA, Carry,
Overflow

P relative (+/-)

Direct or indirect

The TOS is decremented. If the result is then zero, control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
(S) := (S) - 1;	SR<2	--	SRP	0004	--
if (S) = 0 then P := P + L else P := P + 1;	SR>=2	227	DABZ	2366	1

CPRB Compare range and branch

Indicators: CCE, CCL, CCG
P relative (+/-)
Direct or indirect

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	0	1	1	0	±					

Displacement

The integer in the Index register is tested to determine if it is within the interval defined by the upper bound integer on the TOS and the lower bound integer in the second word of the stack. The Condition Code is set by the comparison to a special pattern: CCE if within range, CCL if below range, CCG if above range. If the integer in the Index register is within the specified range, control is then transferred to P ± displacement; otherwise to P+1. The top two elements of the stack are deleted in either case.

	COND	LUTA	LABEL	RAR	W
if (S-1) <= X <= (S) then	SR<2	--	SRP	0004	--
begin	SR>=2	226	CPRB	0405	1
CC := CCE;					
P := P + L;					
end else					
begin					
if X < (S-1) then CC := CCL;					
if X > (S) then CC := CCG;					
end;					
S := S - 2;					

XCH Exchange A and B

Indicators: CCA on the
new TOS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	1	0						

Alternate
Position

The top two words of the stack are interchanged.

	COND	LUTA	LABEL	RAR	W
TEMP := (S-1);	SR<2	--	SRP	0004	--
(S-1) := (S);	SR>=2	252	XCH	0702	1
(S) := TEMP;					

XAX Exchange A and X

Indicators: CCA on the
new TOS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	0	1						

Alternate
Position

The content of the TOS and the Index register are interchanged.

	COND	LUTA	LABEL	RAR	W
TEMP := X;	SR<2	--	SRP	0004	--
X := (S);	SR>=2	255	XAX	0704	1
(S) := TEMP;					

XBX Exchange B and X
Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	0	1						

Alternate
Position

The second word of the stack is interchanged with the content of the Index register.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
TEMP := X;	SR<2	--	SRP	0004	--
X := (S-1);	SR>=2	275	XBX	0706	1
(S-1) := TEMP;					

DXCH Double exchange
Indicators: CCA on the new
TOS doubleword

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	1	0						

Alternate
Position

The top two doubleword pairs are interchanged on the stack.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
TEMP := (S-1, S);	SR<4	316	SRP	0004	1
(S-1, S) := (S-3, S-2);	SR=4	356	DXCH	0633	1
(S-3, S-2) := TEMP;					

CAB Rotate A,B,C
Indicators: CCA on the
new TOS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	1	0						

Alternate
Position

The third word of the stack is removed from the stack, the two top words are compressed onto the rest of the stack, and the original third word is pushed onto the stack.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
TEMP := (S-2);	SR<4	336	SRP	0004	1
(S-2) := (S-1);	SR=4	376	CAB	0710	1
(S-1) := (S);					
(S) := TEMP;					

TBA Test and branch, limit in A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	0	0	±								

Indicators: unaffected
P± relative

Displacement

This instruction expects the top three elements of the stack to be initialized as follows: A contains a limit, B contains a step size, and C contains a DB+ relative displacement for the address of a variable. TBA tests the variable against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top three elements of the stack are deleted and execution continues at P + 1.

COND LUT LABEL RAR W

VAR := ((S-2) + DB);
STEP := (S-1);
FINAL := (S);
if STEP >= 0 then
 if VAR > FINAL then S := S - 3 else P := E;
if STEP < 0 then
 if VAR < FINAL then S := S - 3 else P := E;

P REL. 027 LCB1 0416 0

TBX Test and branch, variable in X

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	1	0	0	±								

Indicators: unaffected
P± relative

Displacement

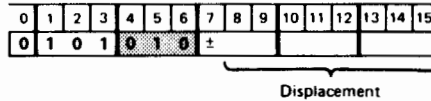
This instruction requires that the Index register contains the variable and that the top two elements of the stack are initialized as follows: A contains a limit and B contains a step size. TBX tests the variable in X against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top two elements of the stack are deleted and execution continues at P + 1.

COND LUT LABEL RAR W

VAR := X;
STEP := (S-1);
FINAL := (S);
if STEP >= 0 then
 if VAR > FINAL then S := S - 2 else P := E;
if STEP < 0 then
 if VAR < FINAL then S := S - 2 else P := E;

P REL. 027 LCB1 0416 0

MTBA Modify variable, test and branch, limit in A



Indicators: unaffected
P± relative

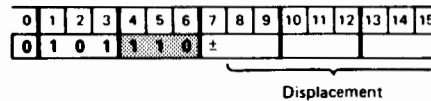
This instruction expects the top three elements of the stack to be initialized as follows: A contains a limit, B contains a modifying step size, and C contains a DB+ relative displacement for the address of a variable. MTBA adds the step size to the variable in integer form, replaces the old variable with this new sum, and tests the new sum against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top three elements of the stack are deleted and execution continues at P+1.

COND LUTA LABEL RAR W

```
VAR := ((S-2) + DB);
STEP := (S-1);
FINAL := (S);
VAR := VAR + STEP;
((S-2)) := VAR;
if STEP >= 0 then
  if VAR > FINAL then S := S - 3 else P := E;
if STEP < 0 then
  if VAR < FINAL then S := S - 3 else P := E;
```

P REL. 027 LCB1 0416 0

MTBX Modify variable in X, test and branch



Indicators: unaffected
P± relative

This instruction requires that the Index register contains the variable and that the top two elements of the stack are initialized as follows: A contains a limit and B contains a modifying step size. MTBX adds the step size to the variable in integer form, replaces the old Index register contents with this new sum, and tests the new sum against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top two elements of the stack are deleted and execution continues at P + 1.

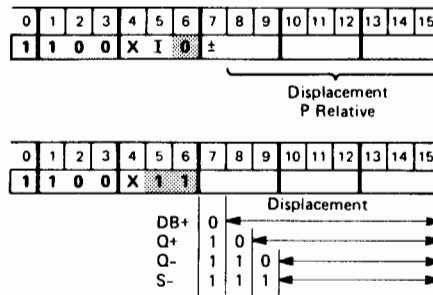
COND LUTA LABEL RAR W

```
VAR := X;
STEP := (S-1);
FINAL := (S);
VAR := VAR + STEP;
X := VAR;
if STEP >= 0 then
  if VAR > FINAL then S := S - 2 else P := E;
if STEP < 0 then
  if VAR < FINAL then S := S - 2 else P := E;
```

P REL. 027 LCB1 0416 0

BR Branch unconditionally

Indicators: unaffected
P relative (+/-), direct
or indirect
DB+ relative, indirect
Q+ relative, indirect
Q- relative, indirect
S- relative, indirect
Indexing available

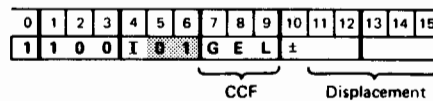


For P relative mode, control is transferred unconditionally to $P \pm$ displacement, plus (if specified) the value in X; may be indirect. For DB, Q, and S relative modes, control is transferred indirectly (only) via the location specified by DB, Q, or S $\pm$ the displacement; the content of the location so specified is added to PB (plus post-indexing if X is specified) to obtain the effective address for P.

	COND	LUTA	LABEL	RAR	W
if CIR(6) = 0 then P :=	P REL.	063	BRP	0302	0
E <<direct or indirect>>	DB REL.	060	BRD	0272	0
else P := PB + (DB or Q or	Q REL.	060	BRD	0272	0
S + - D) + X;	S REL.	061	BRS	0271	0
<<indirect only>>					

BCC Branch on Condition Code

Indicators: unaffected
P relative (+/-)
Direct or indirect



The Condition Code in the Status register is compared with conditions named in the CCF field of the instruction. If the named conditions are met, control is transferred to $P \pm$ displacement; otherwise to $P+1$. Control is transferred to the branch address under the following conditions:

- If CCF = 0, never branch
= 1, branch if CC = CCL
= 2, branch if CC = CCE
= 3, branch if CC = CCL or CCE
= 4, branch if CC = CCG
= 5, branch if CC = CCG or CCL
= 6, branch if CC = CCG or CCE
= 7, always branch

	COND	LUTA	LABEL	RAR	W
If CC and (CCF) $\neq$ 0 then	UNC.	060	BRD	0272	0
P := P + L;					

BCY Branch on carry

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	1	1	0	0	±					

Indicators: Carry cleared
P relative (+/-)
Direct or indirect

Displacement

If the Carry bit of the Status register is set ("1"), control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
if STATUS(5) = 1 then	SR<2	--	SRP	0004	—
P := P + L else P := P + 1;	SR>=2	214	BCY	0336	1

BNCY Branch on no carry

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	1	1	0	1	±					

Indicators: Carry cleared
P relative (+/-)
Direct or indirect

Displacement

If the Carry bit of the Status register is clear ("0"), control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
if STATUS(5) = 0 then	SR<2	--	SRP	0004	—
P := P + L else P := P + 1;	SR>=2	215	BNCY	0342	1

BOV Branch on overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	0	0	0	±					

Indicators: Overflow cleared
P relative (+/-)
Direct or indirect

Displacement

If the Overflow bit of the Status register is set ("1"), control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
if STATUS(4) = 1 then	SR<2	--	SRP	0004	—
P := P + L else P := P + 1;	SR>=2	230	BOV	0345	1

BNOV Branch on no overflow

Indicators: Overflow cleared
 P relative (+/-)
 Direct or indirect

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	0	0	1	±					

Displacement

If the Overflow bit of the Status register is clear ("0"), control is transferred to P ± displacement; otherwise to P+1.

	COND	LUTA	LABEL	RAR	W
if STATUS(4) = 0 then	SR<2	—	SRP	0004	—
P := P + L else P := P + 1;	SR>=2	231	BNOV	1770	1

BRO Branch on TOS odd

Indicators: unaffected
 P relative (+/-)
 Direct or indirect

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	1	1	0	±					

Displacement

If the TOS is odd (bit 15 = 1), control is transferred to P ± displacement; otherwise to P+1. The TOS is deleted.

	COND	LUTA	LABEL	RAR	W
if (S(15)) = 1 then P := P + L;	SR<2	—	SRP	0004	—
S := S-1;	SR>=2	236	BRO	2371	1

BRE Branch on TOS even

Indicators: unaffected
 P relative (+/-)
 Direct or indirect

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	1	1	1	±					

Displacement

If the TOS is even (bit 15 = 0), control is transferred to P ± displacement; otherwise to P+1. The TOS is deleted.

	COND	LUTA	LABEL	RAR	W
if (S(15)) = 0 then P := P + L;	SR<2	—	SRP	0004	—
S := S-1	SR>=2	237	BRE	1773	1

REGISTER AND PROGRAM CONTROL INSTRUCTIONS

PSHR Push registers

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	0	1	1	1	1	1	1	1	1	1

DB DL Z Sta X Q S

The content of a register (or the displacement it represents) specified by any bit 9 through 15 is pushed onto the stack. If more than one register (or displacement) is specified, the contents will be stacked in the order shown below, such that if all seven were specified, DB would be on the TOS after execution, DL - DB next, etc. Note that when S - DB is pushed, the value stacked will be as it existed before the execution of this instruction. Stack overflow occurs if S + 7 exceeds Z, regardless of the number of registers pushed.

COND _ _ LUTA _ LABEL _ RAR _ W

if CIR(15) = 1 then PUSH(S-DB); UNC. 144 PSHR 1557 0
 if CIR(14) = 1 then PUSH(Q-DB);
 if CIR(13) = 1 then PUSH(X);
 if CIR(12) = 1 then PUSH(STATUS);
 if CIR(11) = 1 then PUSH(Z-DB);
 if CIR(10) = 1 then PUSH(DL-DB);
 if CIR(9) = 1 then PUSH(DB);

where. . .

PUSH(REGISTER) is defined to be
 Procedure PUSH (Register);
 Value register; Integer register;
 Begin
 S := S + 1;
 (S) := REGISTER;
 End;

Note for pushing S-DB the following occurs:

S := S + 1;
 (S) := S - DB - 1;

SETR Set registers

Indicators: unaffected (may be changed if bit 12 = "1")

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

DB DL Z Sta X Q S

This is a semi-privileged instruction.

The registers specified by bits 9 through 15 of the instruction are filled by an absolute value from the TOS for the Index, Status, and DB registers, and an absolute value computed by adding DB to the TOS (displacement value) for the others. If more than one register (or displacement) is specified, the registers will be loaded in the order shown below, such that if all seven were specified, the DB-register would receive the first TOS and the value for S would be computed from the seventh TOS. The TOS is deleted after each register is set. If the Z-register is set to ZI, the Interrupt Stack flag is set to "1"; otherwise it is cleared. The Dispatcher flag is always cleared on setting Z. SETR is a privileged instruction except for setting the Index register, Q, S, and bits 2 and 4 through 7 of the Status register (user traps disable/enable, Overflow, Carry, and Condition Code).

	COND	LUTA	LABEL	RAR	W
if CIR(9) = 1 and PRIV. MODE then	SR<2	--	SRP	0004	--
POP(DB);	SR>=2	174	SETR	1605	0
if CIR(10) = 1 and PRIV. MODE then POP(DL + DB);					
if CIR(11) = 1 and PRIV. MODE then POP(Z + DB);					
if CIR(12) = 1 and if PRIV. MODE then POP(STATUS(0:15)) else POP(STATUS(2,4:7))					
if CIR(13) = 1 then POP(X);					
if CIR(14) = 1 then POP(Q + DB);					
if CIR(15) = 1 then POP(S + DB);					

where, . . .
 POP(REGISTER) is defined to be
 procedure POP(Register);
 Begin
 Register := (S);
 if Register = "S"
 then S := S - 1;
 End;

XCHD Exchange DB and TOS

Indicators: unaffected
 This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0

This instruction expects a new DB value on the TOS. The current DB replaces that value on the TOS while the new value is placed in the DB register. Bits 12 through 15 are ignored.

	COND	LUTA	LABEL	RAR	W
TEMP := DB;	UNC.	117	XCHD	1674	1
DB := (S);					
(S) := TEMP;					

ADDS Add to S

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	0	1	0								

Immediate Operand

The immediate operand N is added to S unless N is zero; if N is zero, the TOS content, minus one, is added to S instead.

	COND	LUTA	LABEL	RAR	W
if N > 0 then S := S + N;	UNC.	152	ADDS	1661	0
if N = 0 then					
Begin					
Temp := (S);					
S := S-1; <<This leaves original (S) undefined>>					
S := S + Temp;					
End;					

SUBS Subtract from S

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	0	1	1								

Immediate Operand

The immediate operand N is subtracted from S unless N is zero; if N is zero, the TOS content, minus one, is subtracted from S instead.

COND _ _ LUT _ LABEL _ RAR _ W

if N > 0 then S := S - N;

UNC.

156

SUBS

1660

1

if N = 0 then

Begin

Temp := (S);

S := S-1; <<This leaves original (S) undefined>>

S := S - Temp;

End;

LRA Load relative address

Indicators: unaffected

P±, DB+, Q±, S-

relative

Direct or indirect

Indexing available

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	X	I										

Mode and Displacement

The effective address is computed, then subtracted from the appropriate base register (PB for P± addressing or DB for DB+, Q±, and S- addressing). The resulting relative address is pushed onto the stack. An attempt to load from beyond the limits defined by PB and PL or (in user mode only) DL and S will cause a Bounds Violation interrupt to segment 11.

COND _ _ LUT _ LABEL _ RAR _ W

TEMP = if CIR(6) = 0 then

P REL.

077

AC2P

0055

1

E - PB else E - DB;

DB REL.

074

AC2D

0044

1

S := S + 1;

Q REL.

074

AC2D

0044

1

(S) := TEMP;

S REL.

075

AC2S

0043

1

JLUI

076

LRA

0124

1

LLBL Load label

Indicators: unaffected

PL- relative

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	1	1								

Displacement
PL-

The label in the Segment Transfer Table (STT) at PL-N is loaded onto the TOS. The value N is a displacement given in the argument field of the instruction. If the label is local, it is converted to external type when loaded. To be valid, the value N must point to a location which is actually in the STT (i.e., $N \leq \text{STTL}$) in all cases; additionally, in the case of local labels, N must not exceed octal 177 (decimal 127), since this is the maximum range for external labels. An invalid value of N will invoke a STT Violation trap.

	COND	LUTA	LABEL	RAR	W
if N > (PL(8:15)) then	SR<2	—	SRP	0004	—
STT violation trap else	SR>=2	136	LLBL	3407	1

```

Begin
  if ((PL - N) (0)) = 1 then
    Begin
      S := S + 1;
      (S) := (PL-N);
    End else
    if N > %177 then STT violation trap else
      Begin
        S := S + 1;
        (S(0)) := 1;
        (S(1:7)) := N;
        (S(8:15)) := STATUS (8:15);
      End;
    End;
  End;

```



PCAL Procedure call

Indicators: unaffected
 Indirect via:
 PL - N (if N ≠ 0)
 TOS (if N = 0)
 Local Label: PB+
 External Label: via CST
 to local label in target
 segment

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	1	0								

N

Control is transferred to the location pointed to by the evaluation of the program label at PL - N, unless N is zero; if N is zero, the program label is taken from the TOS and then deleted. Then a four word stack marker is placed on the stack, and Q and S are updated to point at this new marker. The program label may be local or external. If the Trace bit is on in the target CST entry, the PCAL will be made to the Trace segment. If a privileged user is calling a user segment, it will run in privileged mode.

	COND	LUTA	LABEL	RAR	W
if N = 0 then	UNC.	112	PCAL	2546	1

```

begin
  TEMP := (S);
  S := S - 1;
end else TEMP := (PL - N);
S := S + 1;
(S) := X;
S := S + 1;
(S) := P + 1 - PB
S := S + 1;
(S) := STATUS;
S := S + 1;
(S) := S - Q;
Q := S;
P := evaluation of label in TEMP;

```

Exit from routine

Indicators: unaffected

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	1	1								

N

This instruction is used to return from a routine called by the PCAL instruction or by an interrupt. A normal exit occurs by restoring the return address to P, restoring the previous contents of the Index and Status registers, and deleting all stack variables incurred by the called routine, plus its marker, plus N number of procedure parameters. The value of N may be any number from 0 through 255 for exits from PCAL routines; it must be 0 for exits from interrupt routines. An interrupt routine exits normally to the calling routine except when: a) exiting from the last routine to use the Interrupt Control Stack, or b) exiting from an external interrupt routine and there is another external interrupt pending. In case "a", the system automatically exits to the Dispatcher. In case "b", the new device number replaces the old device number on the stack (without changing the existing stack marker), and the new external interrupt is then processed. If the exit is from an external interrupt routine, EXIT clears the device's interrupt-active logic. If bit 0 of the return-P marker word is a "1", control is transferred to the Trace segment. If the return segment is absent, control is transferred to the Absence segment.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
TEMP := 4	UNC.	116	EXIT	2621	1
Begin << normal exit only >>					
TEMP := Q - TEMP;					
S := Q;					
Q := S - (S);					
S := S - 1;					
STATUS := (S); Note that user mode may not EXIT to privileged mode and may not change the interrupt disable bit, 1.					
If STATUS(8:15) = 0 then					
Begin PB := 0;					
PL := 2**16 - 1;					
End					
Else PB, PL := evaluation of segment descriptor specified by STATUS (8:15);					
S := S - 1;					
P := PB + (S);					
S := S - 1;					
X := (S);					
S := TEMP - N	Note that in exits from interrupt routine N must be 0.				
End;					

Indicators: unaffected
Indirect via:
 PL - N (if $N \neq 0$)
 TOS (if $N = 0$)
Local Label: PB+

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	1								

N

Control is transferred to the location pointed to by the evaluation of the local label at PL-N, unless N is zero; if N is zero the local label is taken from the TOS and then deleted. The return address is then pushed onto the stack. Only local labels are allowed; non-local label gives STT Violation trap.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
if N = 0 then begin TEMP := (S); S := S - 1; end else TEMP := (PL - N); S := S + 1; (S) := P + 1 - PB; P := evaluation of label in TEMP;	UNC.	106	SCAL	2545	1

SXIT	Exit from subroutine	<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	1	1	0	1	0	0								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																			
0	0	1	1	0	1	0	0																											
	Indicators: unaffected	N																																

This instruction is used to return from a subroutine called by the SCAL instruction. The SXIT instruction assumes that the return address is on the TOS, and returns program control to this address. The TOS is then deleted, plus N number of subroutine parameters. The value of N may be any number from 0 through 255.

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
P := PB + (S); S := S - N - 1;	SR<2 SR>=2	--- 122	SRP SXIT	0004 1531	-- 1

XEQ	Execute stack word	<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td></td><td></td><td></td><td></td></tr></table>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	1	1	0	0	0	0	0	1	1	0				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																			
0	0	1	1	0	0	0	0	0	1	1	0																							
	Indicators: set by the executed instruction	K																																

The content of the word in the stack at S-K is placed in the Current Instruction Register to be executed. After execution, control is returned to the instruction after the XEQ unless a transfer of control was executed (branch, PCAL, etc.). If the word to be executed is a Stack Op, only the first position (bits 4 through 9) may be used; bits 10 through 15 must be a NOP. The value of K is 0 through 15 (decimal).

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
CIR := (S - K);	UNC.	133	XEQ	1545	0

PAUS	Pause	<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td></td><td></td><td></td><td></td></tr></table>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	1	1	0	0	0	0	0	0	0	1				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																			
0	0	1	1	0	0	0	0	0	0	0	1																							
	Indicators: unaffected This is a privileged instruction.	Not Used																																

The computer hardware pauses; interrupts may occur. Bits 12 through 15 are ignored, and may be used to identify PAUS instructions (e.g., PAUS 2, PAUS 7).

	<u>COND</u>	<u>LUTA</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
	UNC.	107	PAUS	1541	1

HALT Machine halt

Indicators: unaffected
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	1	1				

Not Used

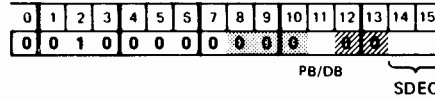
The computer hardware halts. Internal interrupts may be processed only in this state. Power Fail, Power On, Console Interrupt, or a halt-mode enabled external interrupt may force execution to begin; otherwise manual intervention is required to run. Bits 12 through 15 are ignored and may be used to identify HALT instructions (e.g., HALT 2, HALT 7).

<u>COND</u>	<u>LUT</u>	<u>A</u>	<u>LABEL</u>	<u>RAR</u>	<u>W</u>
UNC.	177		HALT	2733	1

MOVE AND SPECIAL INSTRUCTIONS

MOVE Move words

Indicators: unaffected
DB+ or PB+ for source
DB+ for target



This instruction transfers a specified number of words from one area of primary memory to another. The instruction expects a signed word count in A, a DB or PB relative displacement for a source address in B, and a DB relative displacement for a target address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The content of the memory location specified by DB + B or PB + B is transferred to the location specified by DB + C. If the word count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the word count is decremented by one. If the word count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the word count is incremented by one. Note that the word count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words specified by the SDEC (S decrement) field of the instruction; the range of this field is 0 through 3.

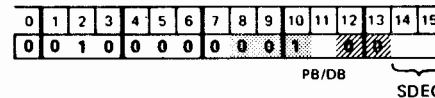
	<u>COND</u> _ _	<u>LUTA</u> _	<u>LABEL</u> _	<u>RAR</u> _	<u>W</u>
--	-----------------	---------------	----------------	--------------	----------

```

BASE := if CIR(11) = 0 then      P REL.    101    PROG    2027    0
      PB else DB;              DB REL.    105    DATA    2030    0
While (s) = 0 do
  if CIR(11) = 0 << PB relative >> then
    if ((PL >= (S - 1) + PB >= PB and S - 3 >= (S - 2) + DB >= DL)
       or PRIVMODE) then GO TO M
    else INTERRUPT (seg %11);
  else << DB relative >>
    if ((S - 3 >= (S - 1) + DB >= DL and S - 3 >= (S - 2) + DB >= DL)
       or PRIVMODE) then
      M: Begin
        ((S - 2) + DB) := ((S - 1) + BASE);
        (S - 2) := (S - 2) + Sign (S);
        (S - 1) := (S - 1) + Sign (S);
        (S) := (S) - Sign (S);
      end;
    else INTERRUPT (seg %11);
  S := S - SDEC;
  
```

MVB Move bytes

Indicators: unaffected
Byte addressing
DB+ or PB+ for source
DB+ for target



The MVB instruction transfers a specified number of bytes from one area of primary memory to another. The instruction expects a signed byte count in A, a DB or PB relative displacement for a source byte address in B, and a DB relative displacement for a target byte address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The content of the byte address location specified by DB + B or PB + B is transferred to the byte address location specified by DB + C. If the byte count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the byte count is decremented by one. If the byte count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the byte count is incremented by one. Note that the byte count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

MOVE, SPECIAL

	COND	LUTA	LABEL	RAR	W
While (S) $\bar{=}$ $\emptyset$ do	P REL.	111	PROG	2027	0
Begin	DB REL.	115	DATA	2030	0
BYTEMOVEADDRESS					
((S - 1), (S - 2), S - 3);					
(BTAD) := (BSAD); <<store source byte in target>>					
(S - 2) := (S - 2) + Sign(S);					
(S - 1) := (S - 1) + Sign(S);					
(S) := (S) - Sign(S);					
end;					
S := S - SDEC;					

MVLB Move words from
DL+ to DB+

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	1	0				

Indicators: unaffected
DL+ for source
DB+ for target
This is a privileged instruction.

SDEC

This instruction transfers a specified number of words from the DL+ area of the data segment to the DB+ area. The instruction expects a signed word count in A, a DL relative displacement for a source address in B, and a DB relative displacement for a target address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The contents of the memory location specified by DL + B is transferred to the location specified by DB + C. If the word count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the word count is decremented by one. If the word count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the word count is incremented by one. Note that the word count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

	COND	LUTA	LABEL	RAR	W
While (S) $\bar{=}$ $\emptyset$ do	UNC.	131	MVLB	2003	0
Begin					
((S-2) + DB) := ((S-1) + DL);					
(S-2) := (S-2) + Sign(S);					
(S-1) := (S-1) + Sign(S);					
(S) := (S) - Sign(S);					
End;					
S := S - SDEC;					

MVBL Move words from
DB+ to DL+

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	0	0	0			

Indicators: unaffected
DB+ for source
DL+ for target
This is a privileged instruction.

SDEC

This instruction transfers a specified number of words from the DB+ area of the data segment to the DL+ area. The instruction expects a signed word count in A, a DB relative displacement for a source address in B, and a DL relative displacement for a target address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The contents of the memory location specified by DB + B is transferred to the location specified by DL + C. If the word count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the word count is decremented by one. If the word count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the word count is incremented by one. Note that the word count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

	COND	LUTA	LABEL	RAR	W
While (S) $\neq$ 0 do	UNC.	121	MVBL	2000	0

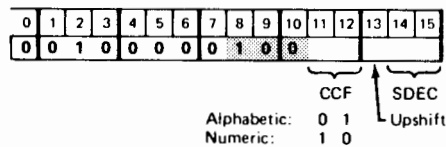
```

Begin
  ((S-2) + DL) := ((S-1) + DB);
  (S-2) := (S-2) + Sign(S);
  (S-1) := (S-1) + Sign(S);
  (S) := (S) - Sign(S);
End;
S := S - SDEC;

```

MVBW Move bytes while
of specified type

Indicators: CCB on the
last character scanned
Byte addressing, DB+



This instruction transfers an unspecified number of bytes from one area of primary memory to another. The instruction expects a source byte address in the TOS and a DB relative displacement for a target byte address in the second word of the stack. As long as the source byte is of the type specified in the CCF field, it is moved to the target area. The target displacement value in B is incremented by one on each transfer. If the byte to be moved is a lower case letter and the upshift bit is on, the target byte will be an upshifted copy of the source byte. Byte transfers continue until the source byte is not of the proper type. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

	COND	LUTA	LABEL	RAR	W
MOVING := True;	P REL.	141	SCAM	2126	0
While MOVING do	DB REL.	145	SCAM	2126	0

```

Begin
  BYTEMOVEADDRESS ((S), (S - 1), S - 2);
  if (CCB on (BSAD))  $\subset$  CCF then
    (BTAD) := if CIR(13) = 1 <<upshift indicator>> and
      (BSAD) is lower case letter then
        upshift (BSAD) else (BSAD);
    (S - 1) := (S - 1) + 1;
    (S) := (S) + 1;
  else MOVING := FALSE;
end;
S := S - SDEC;

```

MOVE, SPECIAL

RSW Read Switch register

Indicators: CCA

The content of the Switch register is pushed onto the stack.

COND _ _ LUTA _ LABEL _ RAR _ W

S := S + 1;
(S) := Switch register;

UNC. 161 RSW 0351 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0

LLBL Load label

Indicators: unaffected
PL- relative

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	1	1								

Displacement
PL-

The label in the Segment Transfer Table (STT) at PL-N is loaded onto the TOS. The value N is a displacement given in the argument field of the instruction. If the label is local, it is converted to external type when loaded. To be valid, the value N must point to a location which is actually in the STT (i.e., $N < \text{STTL}$) in all cases; additionally, in the case of local labels, N must not exceed octal 177 (decimal 127), since this is the maximum range for external labels. An invalid value of N will invoke a STT Violation trap.

COND _ _ LUTA _ LABEL _ RAR _ W

```

if N > (PL(8:15)) then
  STT violation trap else
  SR < 2
  SR >= 2
  136
  SRP
  LLBL
  0004
  3407
  1
Begin
  if ((PL - N) (0)) = 1 then
    Begin
      S := S + 1;
      (S) := (PL-N);
    End else
    if N > %177 then STT violation trap else
      Begin
        S := S + 1;
        (S(0)) := 1;
        (S(1:7)) := N;
        (S(8:15)) := STATUS (8:15);
      End;
    End;
  End;
End;

```

PLDA Privileged load from
absolute address

Indicators: CCA
Addressing mode: absolute
This is a privileged instruction.

Index register is a 16-bit absolute address; the content of this address is pushed onto the stack.

COND _ _ LUTA _ LABEL _ RAR _ W

S := S + 1;
(S) := (X);

UNC. 165 PLDA 0753 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0

PSTA Privileged store into
absolute address

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	1	0	1	0	1

Indicators: unaffected
Addressing mode: absolute
This is a privileged instruction.

The content of the Index register is a 16-bit absolute address; the top word of the stack is stored into memory at that address, and then deleted from the stack.

COND	LUTA	LABEL	RAR	W
UNC.	165	PLDA	0753	0

(X) := (S);
S := S - 1;

LLSH Linked List Search

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1

Indicators:
CCL if terminated by $X = 0$
CCE if terminated by
target $\geq B$
CCG if terminated by
target $= 2^{16} - 1$
Addressing mode:
absolute $\pm$ offset
This is a privileged instruction.

This instruction searches through a linked list in memory, comparing a test word with a target word, until either: a target word is found that is equal to or greater than the test word, or a target word is all ones, or the specified count has been counted to zero. The instruction expects a positive count value in the Index register. Also, A must contain an absolute pointer into the linked list, B must contain the test word, and C is an offset which indicates the position, relative to each link, of the target location. At each step, the test word is compared to the content of the target location. If the target content is logically greater than or equal to the test word, or if the target content is all ones, the instruction terminates. Otherwise, the next link replaces the current link in A, the count in the Index register is decremented, and the instruction repeats until the count becomes zero.

COND	LUTA	LABEL	RAR	W
UNC.	161	RSW	0351	0

while $X > 0$ and $((S) + (S-2))$
< $(S - 1)$ do
Begin
(S) := ((S));
X := X - 1;
End;

MOVE, SPECIAL

CMD	Command	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0	0	1	1	0	0	0	0	1	1	0	0				

K

Indicators: unaffected
This is a privileged instruction.

This instruction assumes that the TOS contains a 16-bit data word to be sent to a system hardware module and expects a command word in the stack at S-K. Bits 13 through 15 of the command word specify the module number, and bits 10 and 11 are used to specify a module command. (The four possible commands depend upon application and do not form a part of this instruction's definition.) CMD sends the 16-bit data word and 2-bit command over the central data bus to the specified module, and then deletes the TOS. (Note: if the destination module is not ready, the CPU will not proceed until that module becomes ready.)

<u>COND</u> <u>LUTA</u> <u>LABEL</u> <u>RAR</u> <u>W</u>
MCU Module # := (S-K(13:15)); UNC. 163 CMD 2356 0
Command := (S-K(10:11));
16 BIT DATA WORD := (S);
S := S - 1;

NOP	No operation	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0	0	0	0	0	0	0	0	0	0						

Alternate Position

Indicators: unaffected

The user's program space and data space remain unchanged.

<u>COND</u> <u>LUTA</u> <u>LABEL</u> <u>RAR</u> <u>W</u>
SR<2 340 NOP 1466 1
SR>=2 300 NOP 1466 1

I/O AND INTERRUPT INSTRUCTIONS

SED Set "enable/disable external interrupts" bit

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0

D/E

Indicators: unaffected
This is a privileged instruction.

The interrupt system is enabled or disabled according to the least significant bit (bit 15) of the instruction. If bit 15 is a "1", bit 1 of the Status register is set, thus enabling external interrupts. If bit 15 is a "0", bit 1 of the Status register is cleared, thus disabling external interrupts. Bits 12, 13, and 14 of the instruction are ignored.

COND LUTA LABEL RAR W

STATUS(1) := CIR(15); UNC. 113 SED 2315 0

SMSK Set mask

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	0	0				

Indicators:
CCE if no error
CCL if error
This is a privileged instruction.

Not Used

The SMSK instruction assumes that the TOS contains the mask word and transmits this word to all device controllers. Each "1" bit in the mask word sets each Mask flip-flop in the group of device controllers which are specifically wired to be controlled by that bit. Each "0" bit in the mask clears each Mask flip-flop in its group. If there is an I/O error (no acknowledgement), it means that the external interrupt system is in an unknown state. In this case, the SMSK instruction disables the external interrupt system (clears Status bit 1 to "0"), set CCL Condition Code, and leaves the mask on the TOS. If there is no I/O error, the SMSK instruction deletes the mask from the stack and sets the Condition Code to CCE.

COND LUTA LABEL RAR W

16 bit mask word to all device controllers := (S);
if I/O error then CC := CCL else
Begin
 MASKREG := (S)
 S := S - 1;
 CC := CCE;
End;

RMSK Read mask

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	0	1				

Indicators: unaffected

Not Used

This instruction transfers the 16-bit mask word from the Mask register to the TOS.

COND LUTA LABEL RAR W

S := S + 1;
(S) := 16 bit device mask UNC. 127 RMSK 2330 1

SIN Set interrupt

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	1	0				

K

Indicators:

CCE = responding
device controller
CCL = non-responding
device controller

This is a privileged instruction.

This instruction expects a device number to be given in the stack at S-K. SIN sets the Interrupt Request flip-flop in the specified device controller and sets the Condition Code to CCE. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.

COND	LUT	LABEL	RAR	W
UNC.	173	SIN	2323	0

Device = := (S-K(8:15));

If non-responding device
controller then

CC := CCL and terminate instruction else

CC := CCE and set device interrupts

SIRF Set external Interrupt
Reference Flag

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	0	1				

K

Indicators: unaffected

This is a privileged
instruction.

This instruction expects a device number to be given in the stack at S-K. SIRF sets the IRF bit for the specified device to "0" and increments the Interrupt Counter by one. (The IRF bit is bit 0 of the fourth word of the Device Reference Table entry for a device. The Interrupt Counter is fixed memory location 7_h for processor 1 and fixed memory location 13_h for processor 2.) If the IRF bit is already a "0", the Interrupt Counter is not incremented.

COND	LUT	LABEL	RAR	W
UNC.	167	SIRF	2343	0

Device = := (S-K(8:15));

E := Device = * 4 + 3;

If (E(0)) = 1 then

Begin

(E(0)) = 0;

Interrupt Counter := Interrupt Counter + 1

End

SIO Start I/O

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	1	1				

K

Indicators:

CCL = non responding

device controller

CCE = device ready

CCG = device not ready

This is a privileged instruction.

The SIO instruction expects the absolute starting address of an I/O program to be on the TOS, and a device number to be in the stack at S-K. The instruction first checks if the device is ready by checking bit 0 of the device controller's Status register. If it is ready (bit = "1"), the TOS is stored into the first word location of the DRT entry for the device specified at S-K; an SIO command is then issued to the device controller to begin execution of its I/O program, the TOS is deleted, and the Condition Code is set to CCE. If the device is not ready (bit 0 of device status = "0"), the content of the device controller's Status register is pushed onto the stack and the Condition Code is set to CCG. If the device controller does not respond to the readiness test, the Condition Code is set to CCL and the instruction is terminated.

COND	LUTA	LABEL	RAR	W
------	------	-------	-----	---

Device := (S-K(8:15));

UNC.

137

SIO

2233

0

If non-responding device controller then

CC = CCL and terminate instruction;

Begin

If SIO not ready then

Begin

S := S + 1;

(S) := Device Status;

CC := CCG;

End else

Begin

(4*Device) := (S);

S := S-1

Send SIO command to device;

CC = CCE;

End

End;

RIO Read I/O

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	0	0				

K

Indicators:

CCL = non-responding

device controller

CCE = device ready

CCG = device not ready

This is a privileged instruction.

This instruction expects a device number to be given in the stack at S-K. RIO first checks if the device is ready by checking bit 1 of the device controller's Status register. If it is ready (bit = "1"), the 16-bit direct data word from the device is pushed onto the stack and the Condition Code is set to CCE. If it is not ready (bit = "0"), the content of the device controller's Status register is pushed onto the stack and the Condition Code is set to CCG. If the device controller does not respond to the readiness test, the Condition Code is set to CCL and the instruction is terminated.

I/O, INTERRUPT

COND	LUT	LABEL	RAR	W
UNC.	143	RIO	2252	0

```

Device := (S-K(8:15));
If non-responding device
  controller then
  CC = CCL else
Begin
  If RIO not ready then
  Begin
    S := S + 1;
    (S) := Device Status;
    CC := CCG;
  End else;
  Begin
    S := S + 1;
    (S) := 16 bit word from device;
    CC := CCE;
  End;
End;

```

WIO Write I/O

Indicator:
 CCL = non-responding
 device controller
 CCE = device ready
 CCG = device not ready
 This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	0	1				

K

This instruction assumes that the TOS contains a direct data word and expects a device number to be given in the stack at S-K. WIO first checks if the device is ready by checking bit 1 of the device controller's Status register. If it is ready (bit = "1"), the word is transmitted to the specified device and then deleted from the stack; the Condition Code is set to CCE. If it is not ready (bit = "0"), the content of the device controller's Status register is pushed onto the stack and the Condition Code is set to CCG. If the device controller does not respond to the readiness test, the Condition Code is set to CCL and the instruction is terminated.

COND	LUT	LABEL	RAR	W
UNC.	147	WIO	2265	0

```

Device := (S-K(8:15));
If non-responding device
  controller then
  CC = CCL else
Begin
  If WIO not ready then
  Begin
    S := S + 1;
    (S) := Device Status;
    CC := CCG;
  End else
  Begin
    16 bit word to device := (S)
    S = S - 1;
    CC := CCE;
  End;
End;

```

CIO Control I/O

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	1	1				

K

Indicators:

CCE = responding
device controller
CCL = non-responding
device controller

This is a privileged instruction.

This instruction assumes that the TOS contains a control word and expects a device number to be given in the stack at S-K. CIO transmits the TOS to the specified device controller, along with a CIO signal. If the device controller acknowledges receiving the word, the TOS is deleted and the Condition Code is set to CCE. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.

COND	LUTA	LABEL	RAR	W
------	------	-------	-----	---

Device # := (S-K(8:15));	UNC.	157	CIO	2306	0
--------------------------	------	-----	-----	------	---

If non-responding device
controller then
CC := CCL else
Begin
16 bit control word to device := (S);
S := S - 1;
CC := CCE;
End;

TIO Test I/O

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	1	0				

K

Indicators:

CCE = responding
device controller
CCL = non-responding
device controller

This is a privileged instruction.

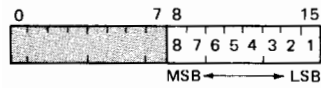
This instruction expects a device number to be given in the stack at S-K. TIO obtains a copy of the device status word from the device controller, pushes it onto the stack, and sets the Condition Code to CCE. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.

COND	LUTA	LABEL	RAR	W
------	------	-------	-----	---

Device # := (S-K(8:15));	UNC.	153	TIO	2300	0
--------------------------	------	-----	-----	------	---

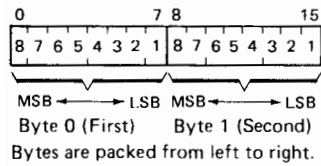
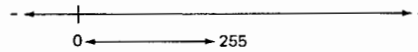
If non-responding device
controller then
CC := CCL else
Begin
S := S + 1;
(S) := Device Status;
CC := CCE;
End;

I/O INTERRUPT

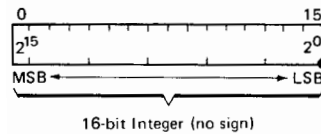


BYTE (BYTE)

Byte parameters have a range of 0-255 characters.

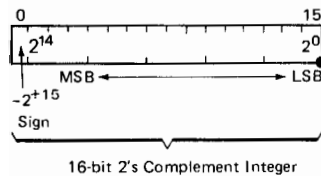
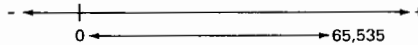


PACKED BYTES (BYTE)



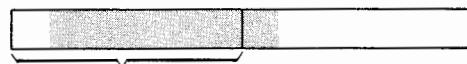
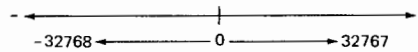
LOGICAL INTEGER (LOGICAL)

Logical integers have positive magnitudes only and have a range of 0 to $2^{16}-1$.

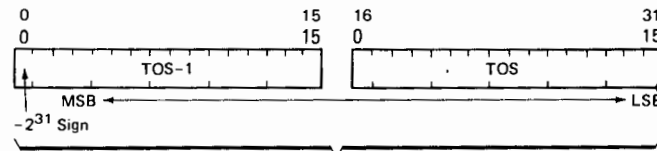


2'S COMPLEMENT INTEGER (INTEGER)

Bit 0 is a sign bit by defining its weight as -2^{15} . 2's complement integers have a range of -2^{15} to $2^{15}-1$.

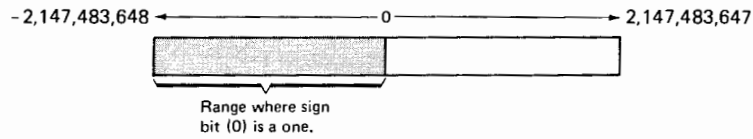
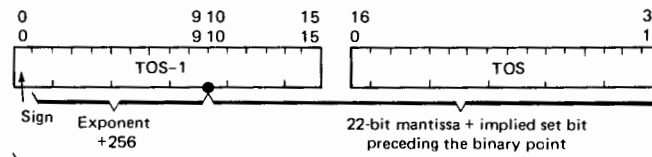


Range where sign bit (0)
is a one.

2'S COMPLEMENT DOUBLE INTEGER (DOUBLE)

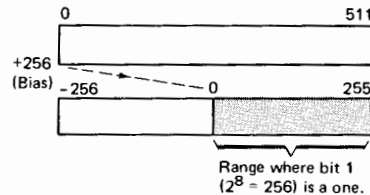
32-bit 2's Complement Double Integer

Bit 0 is a sign bit by defining its weight as -2^{31} . 2's complement double integers have a range of $-2,147,483,648$ to $2,147,483,647$.

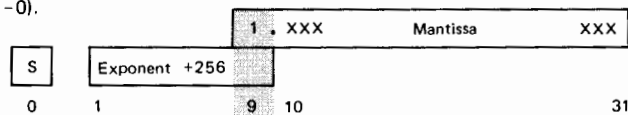
**FLOATING POINT (REAL)**

32-bit Floating Point

Bit 0 is a sign bit by simple definition. If it is clear, the magnitude (mantissa + exponent) is positive. If it is set, the magnitude is negative. Bits 1 to 9 represent the exponent plus (biased) 256; therefore the exponent equals the value expressed by bits 1 to 9 minus 256, and has an effective range of -2^7 to 2^7-1 .



Bits 10 to 31 represent the mantissa with an additional set bit implied in bit position 9 as shown below; therefore the mantissa equals bits 10 to 31 + bit 9 ($2^0 = 1$). The exceptions are 0, which is defined as an all-zero doubleword (would be 1.0×2^{-256}), and a 1 followed by 31 0's (illegal -0).

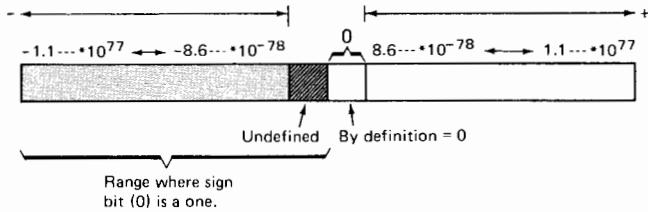


The approximate range of floating point numbers is:

Positive numbers 8.6362×10^{-78} to 1.1579×10^{77}

Negative numbers -8.6362×10^{-78} to -1.1579×10^{77}

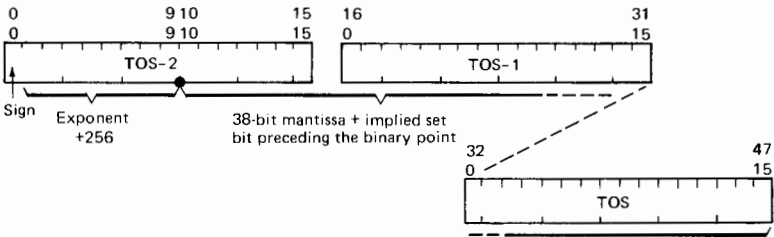
Accuracy is to 6 or 7 digits.



FLOATING POINT EXAMPLES

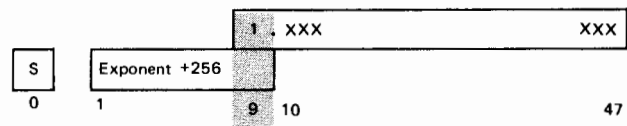
Exp-256	exponent		
2^{-256}	1 0 0 0 0 0	0 0 0 0 0 0	= Undefined
2^{-256}	0 0 0 0 0 0	0 0 0 0 0 0	= Zero (by definition)
2^{-256}	S 0 0 0 0 0	0 0 0 0 0 1	= 8.63616×10^{-78}
2^{-224}	S 0 4 0 0 0	0 0 0 0 0 0	= 3.70921×10^{-68}
2^{-192}	S 1 0 0 0 0	0 0 0 0 0 0	= 1.59312×10^{-58}
2^{-160}	S 1 4 0 0 0	0 0 0 0 0 0	= 6.84228×10^{-49}
2^{-128}	S 2 0 0 0 0	0 0 0 0 0 0	= 2.93823×10^{-39}
2^{-96}	S 2 4 0 0 0	0 0 0 0 0 0	= 1.26218×10^{-29}
2^{-64}	S 3 0 0 0 0	0 0 0 0 0 0	= 5.42101×10^{-20}
2^{-32}	S 3 4 0 0 0	0 0 0 0 0 0	= 2.32831×10^{-10}
2^0	S 4 0 0 0 0	0 0 0 0 0 0	= 1.00000
2^{32}	S 4 4 0 0 0	0 0 0 0 0 0	= 4.29497×10^9
2^{64}	S 5 0 0 0 0	0 0 0 0 0 0	= 1.84467×10^{19}
2^{96}	S 5 4 0 0 0	0 0 0 0 0 0	= 7.92282×10^{28}
2^{128}	S 6 0 0 0 0	0 0 0 0 0 0	= 3.40282×10^{38}
2^{160}	S 6 4 0 0 0	0 0 0 0 0 0	= 1.46150×10^{48}
2^{192}	S 7 0 0 0 0	0 0 0 0 0 0	= 6.27710×10^{57}
2^{224}	S 7 4 0 0 0	0 0 0 0 0 0	= 2.69600×10^{67}
2^{256}	S 7 7 7 7 7	1 7 7 7 7 7	= 1.15792×10^{77}

EXTENDED FLOATING POINT (LONG)



WORD FORMATS

The format of extended floating point numbers is the same as the floating point format given on the preceding page, except that an additional word of 16 bits (32 through 47) is added to the least significant end (right) of the format. This extends the significance of the mantissa to 38 bits + the implied set bit preceding the binary point. The approximate range of extended floating point numbers is the same as for floating point above, except that the accuracy is extended to 11 or 12 digits.



Memory Reference Instruction Format

OP CODE	X	I	Addr mode and Displ
---------	---	---	---------------------

OP CODE 04 to 17

X: 1 → post indexing

I: 1 → one level Indirect

PDQS Address Mode

	6	7	8	9	10	11	12	13	14	15
P +	0	0								
P -	0	1								
DB +	1	0								
Q +	1	1	0							
Q -	1	1	1	0						
S -	1	1	1	1						

P +	0 0	8 bits	0:255	
P -	0 1	8 bits	0:255	
DB +	1 0	8 bits	0:255	
Q +	1 1 0	7 bits	0:127	[DQS only doesn't require bit 6]
Q -	1 1 1 0	6 bits	0:63	
S -	1 1 1 1	6 bits	0:63	

E = Effective address

RA = Relative address = E - base

X = Contents of Index reg

S = SM + SR

D = Displacement field in instr word

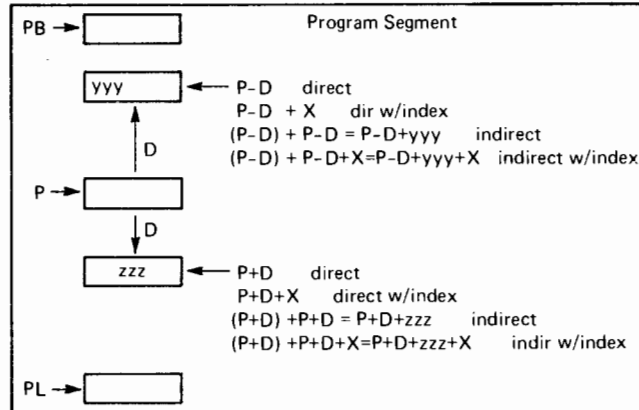
USES

P relative — branches, procedure calls, literals

DB relative — global variable addressing

S relative — temporary variables for subroutines

Q relative — parameter passing and local variables for procedures



Bounds checking $PB \leq E \leq PL$

RA = E - PB

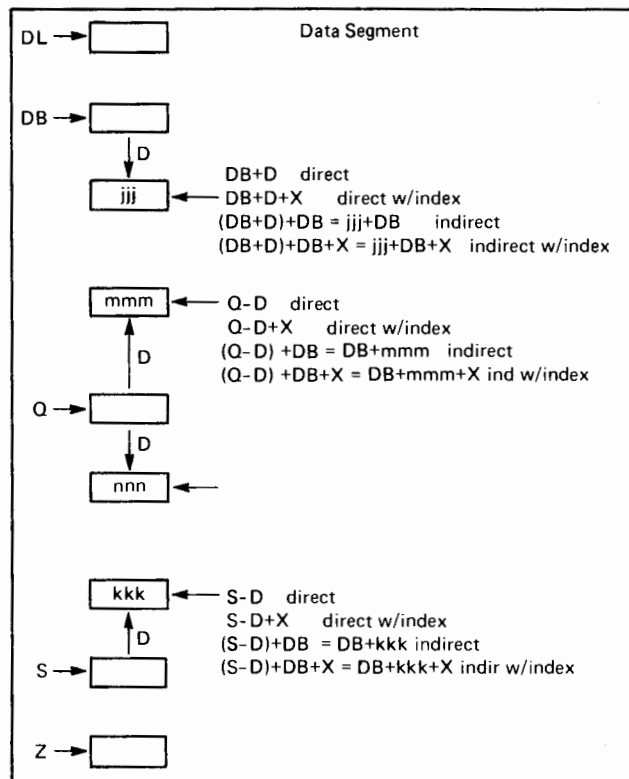
Indirect address in indirect cell is self relative

PCAL uses PL — to access Segment Transfer Table

PCAL return addresses are PB relative

Byte indirect addressing addr in indirect cell is byte address

ADDRESSING MODES



Bounds checking $DL \leq E \leq SM + SR$
 $RA = E - DB$
 Indirect address in indirect cell is DB relative

Addressing arithmetic is done MOD 65K. For array Byte and Double addressing Index register contains entry index. For Byte array X(15) indicates position, 0 = left E(0:7); 1 = right E(8:15). Shift X right to get word address. If out of bounds microcode adds 32K. For Double array, shift left to get word address. For byte indirect, word in indirect cell is byte address.

INTERRUPT CONTROL STACK

QI-5	ABSOLUTE S OF INTERRUPTED PROCESS
QI-4	
QI-3	X (DISPATCHER)
QI-2	ΔP (DISPATCHER)
QI-1	STATUS (DISPATCHER)
QI	ΔQ (=0 DISPATCHER)
QI+1	DB (DISPATCHER)
QI+2	DEV # OR PARAMETER OF INTRP 1

Q-3	X (INTRP 1)
Q-2	ΔP (INTRP 1)
Q-1	STATUS (INTRP 1)
Q	ΔQ (INTRP 1)
Q+1	DB (INTRP 1)
Q+2	DEV # OR PARAMETER INTRP 2

CHAR		BYTE POSITION		Dec.	
		Left	Right	Left	Right
A	A	04:0000	0001	66	66
B	B	04:0000	0002	67	67
C	C	04:0000	0003	68	68
D	D	04:0000	0004	69	69
E	E	04:0000	0005	70	70
F	F	04:0000	0006	71	71
G	G	04:0000	0007	72	72
H	H	04:0000	0010	73	73
I	I	04:0000	0011	74	74
J	J	04:0000	0012	75	75
K	K	04:0000	0013	76	76
L	L	04:0000	0014	77	77
M	M	04:0000	0015	78	78
N	N	04:0000	0016	79	79
O	O	05:0000	00:20	80	80
P	P	05:0000	00:21	81	81
Q	Q	05:0000	00:22	82	82
R	R	05:0000	00:23	83	83
S	S	05:0000	00:24	84	84
T	T	05:0000	00:25	85	85
U	U	05:0000	00:26	86	86
V	V	05:0000	00:27	87	87
W	W	05:0000	00:28	88	88
X	X	05:0000	00:29	89	89
Y	Y	05:0000	00:30	90	90
Z	Z	05:0000	00:32		
a	a	06:0000	00:01	91	91
b	b	06:0000	00:02	92	92
c	c	06:0000	00:03	93	93
d	d	06:0000	00:04	100	100
e	e	06:0000	00:05	101	101
f	f	06:0000	00:06	102	102
g	g	06:0000	00:07	103	103
h	h	06:0000	00:09	104	104
i	i	06:0000	00:10	105	105
j	j	06:0000	00:12	106	106
k	k	06:0000	00:13	107	107
l	l	06:0000	00:14	108	108
m	m	06:0000	00:15	109	109
n	n	06:0000	00:16	110	110
o	o	06:0000	00:17	111	111
p	p	07:0000	00:00	112	112
q	q	07:0000	00:01	113	113
r	r	07:0000	00:02	114	114
s	s	07:0000	00:03	115	115
t	t	07:0000	00:04	116	116
u	u	07:0000	00:05	117	117
v	v	07:0000	00:06	118	118
w	w	07:0000	00:07	119	119
x	x	07:0000	00:10	120	120
y	y	07:0000	00:11	121	121
z	z	07:0000	00:12	122	122
0	0	03:0000	00:00	48	48
1	1	03:0000	00:01	49	49
2	2	03:0000	00:02	50	50
3	3	03:0000	00:03	51	51
4	4	03:0000	00:04	52	52
5	5	03:0000	00:05	53	53
6	6	03:0000	00:06	54	54
7	7	03:0000	00:07	55	55
8	8	03:0000	00:10	56	56
9	9	03:0000	00:11	57	57

65 536	—	16
131 072	—	17
262 144	—	18
524 288	—	19
1 048 576	—	20
2 097 152	—	21
4 194 304	—	22
8 388 608	—	23
16 777 216	—	24
33 554 432	—	25
67 108 864	—	26
134 217 728	—	27
268 435 456	—	28
536 870 912	—	29
1 073 741 824	—	30
2 147 483 648	—	31

ASCII CODES

CHAR		Left		Right		Dec
A		0		1		
B		004000		000101		65
C		041000		000102		66
D		042000		000103		67
E		043000		000104		68
F		044000		000105		69
G		045000		000106		70
H		046000		000107		71
I		047000		000108		72
J		048000		000109		73
K		049000		000110		74
L		050000		000111		75
M		051000		000112		76
N		052000		000113		77
O		053000		000114		78
P		054000		000115		79
Q		055000		000116		80
R		056000		000117		81
S		057000		000118		82
T		058000		000119		83
U		059000		000120		84
V		060000		000121		85
W		061000		000122		86
X		062000		000123		87
Y		063000		000124		88
Z		064000		000125		89
[		065000		000126		90
\		066000		000127		91
]		067000		000128		92
{		068000		000129		93
		069000		000130		94
}		070000		000131		95
~		071000		000132		96
`		072000		000133		97
a		073000		000134		98
b		074000		000135		99
c		075000		000136		100
d		076000		000137		101
e		077000		000138		102
f		078000		000139		103
g		079000		000140		104
h		080000		000141		105
i		081000		000142		106
j		082000		000143		107
k		083000		000144		108
l		084000		000145		109
m		085000		000146		110
n		086000		000147		111
o		087000		000148		112
p		088000		000149		113
q		089000		000150		114
r		090000		000151		115
s		091000		000152		116
t		092000		000153		117
u		093000		000154		118
v		094000		000155		119
w		095000		000156		120
x		096000		000157		121
y		097000		000158		122
z		098000		000159		123
[		099000		000160		124
\		100000		000161		125
]		101000		000162		126
{		102000		000163		127
		103000		000164		128
}		104000		000165		129
~		105000		000166		130
`		106000		000167		131
a		107000		000168		132
b		108000		000169		133
c		109000		000170		134
d		110000		000171		135
e		111000		000172		136
f		112000		000173		137
g		113000		000174		138
h		114000		000175		139
i		115000		000176		140
j		116000		000177		141
k		117000		000178		142
l		118000		000179		143
m		119000		000180		144
n		120000		000181		145
o		121000		000182		146
p		122000		000183		147
q		123000		000184		148
r		124000		000185		149
s		125000		000186		150
t		126000		000187		151
u		127000		000188		152
v		128000		000189		153
w		129000		000190		154
x		130000		000191		155
y		131000		000192		156
z		132000		000193		157
[		133000		000194		158
\		134000		000195		159
]		135000		000196		160
{		136000		000197		161
		137000		000198		162
}		138000		000199		163
~		139000		000200		164
`		140000		000201		165
a		141000		000202		166
b		142000		000203		167
c		143000		000204		168
d		144000		000205		169
e		145000		000206		170
f		146000		000207		171
g		147000		000208		172
h		148000		000209		173
i		149000		000210		174
j		150000		000211		175
k		151000		000212		176
l		152000		000213		177
m		153000		000214		178
n		154000		000215		179
o		155000		000216		180
p		156000		000217		181
q		157000		000218		182
r		158000		000219		183
s		159000		000220		184
t		160000		000221		185
u		161000		000222		186
v		162000		000223		187
w		163000		000224		188
x		164000		000225		189
y		165000		000226		190
z		166000		000227		191
[		167000		000228		192
\		168000		000229		193
]		169000		000230		194
{		170000		000231		195
		171000		000232		196
}		172000		000233		197
~		173000		000234		198
`		174000		000235		199
a		175000		000236		200
b		176000		000237		201
c		177000		000238		202
d		178000		000239		203
e		179000		000240		204
f		180000		000241		205
g		181000		000242		206
h		182000		000243		207
i		183000		000244		208
j		184000		000245		209
k		185000		000246		210
l		186000		000247		211
m		187000		000248		212
n		188000		000249		213
o		189000		000250		214
p		190000		000251		215
q		191000		000252		216
r		192000		000253		217
s		193000		000254		218
t		194000		000255		219
u		195000		000256		220
v		196000		000257		221
w		197000		000258		222
x		198000		000259		223
y		199000		000260		224
z		200000		000261		225
[		201000		000262		226
\		202000		000263		227
]		203000		000264		228
{		204000		000265		229
		205000		000266		230
}		206000		000267		231
~		207000		000268		232
`		208000		000269		233
a		209000		000270		234
b		210000		000271		235
c		211000		000272		236
d		212000		000273		237
e		213000		000274		238
f		214000		000275		239
g		215000		000276		240
h		216000		000277		241
i		217000		000278		242
j		218000		000279		243
k		219000		000280		244
l		220000		000281		245
m		221000		000282		246
n		222000		000283		247
o		223000		000284		248
p		224000		000285		249
q		225000		000286		250
r		226000		000287		251
s		227000		000288		252
t		228000		000289		253
u		229000		000290		254
v		230000		000291		255
w		231000		000292		256
x		232000		000293		257
y		233000		000294		258
z		234000		000295		259
[		235000		000296		260
\		236000		000297		261
]		237000		000298		262
{		238000		000299		263
		239000		000300		264
}		240000		000301		265
~		241000		000302		266
`		242000		000303		267
a		243000		000304		268
b		244000		000305		269
c		245000		000306		270
d		246000		000307		271
e		247000		000308		272
f		248000		000309		273
g		249000		000310		274
h		250000		000311		275
i		251000		000312		276
j		252000		000313		277
k		253000		000314		278
l		254000		000315		279
m		255000		000316		280
n		256000		000317		281
o		257000		000318		282
p		258000		000319		283
q		259000		000320		284
r		260000		000321		285
s		261000		000322		286
t		262000		000323		287
u		263000		000324		288
v		264000		000325		289
w		265000		000326		290
x		266000		000327		291
y		267000		000328		292
z		268000		000329		293
[		269000		000330		294
\		270000		000331		295
]		271000		000332		296
{		272000		000333		297
		273000		000334		298
}		274000		000335		299
~		275000		000336		300
`		276000		000337		301
a		277000		000338		302
b		278000		000339		303
c		279000		000340		304
d		280000		000341		305
e		281000		000342		306
f		282000		000343		307
g		283000		000344		308
h		284000		000345		309
i		285000		000346		310
j		286000		000347		311
k		287000		000348		312
l		288000		000349		313
m		289000		000350		314
n		290000		000351		315
o		291000		000352		316
p		292000				

30350A AUXILIARY CONTROL PANEL
HEWLETT - PACKARD

CURRENT INSTRUCTION REGISTER

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

SELECTED REGISTER

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

HARDWARE REGISTERS

RA	RB	RC	RD	SP2	SP3	OPND	INTRPTG	DIR I/O	CNTR	CPX1	CPX2	MUX I/O	MOD NO.	PAOD	SR
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

SOFTWARE REGISTERS

MEM ADHS	MEM DATA	STATUS	X	PB	P	PL	DL	DB	Q	S	Z	MASK	TEST	R REG	S REG
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

REGISTER

REGISTER POINTER

REGISTER

LOAD
FROM A

LOAD
FROM B

MEMORY

MEMORY ADDRESS MODIFICATION

STORE
FROM A

STORE
FROM B

ENABLE INCR

INHIBIT DECR

A REGISTER

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

B REGISTER

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

MCU INTRPT FREEZE

ENABLE

INHIBIT

RUN MODE INTRPT

INHIBIT

ENABLE

HALT MODE INTRPT

ENABLE

INHIBIT

CONSOLE INTERRUPT

BREAKPOINT HALT

POWER FAILURE AUTO-RESTART

INHIBIT

ENABLE

BREAKPOINT HALT

ENABLE

INHIBIT

SYSTEM DUMP

ENABLE

INHIBIT

SINGLE B REG

RUN

☐

HALT

☐

BREAKPOINT HALT

☐

SYSTEM HALT

☐

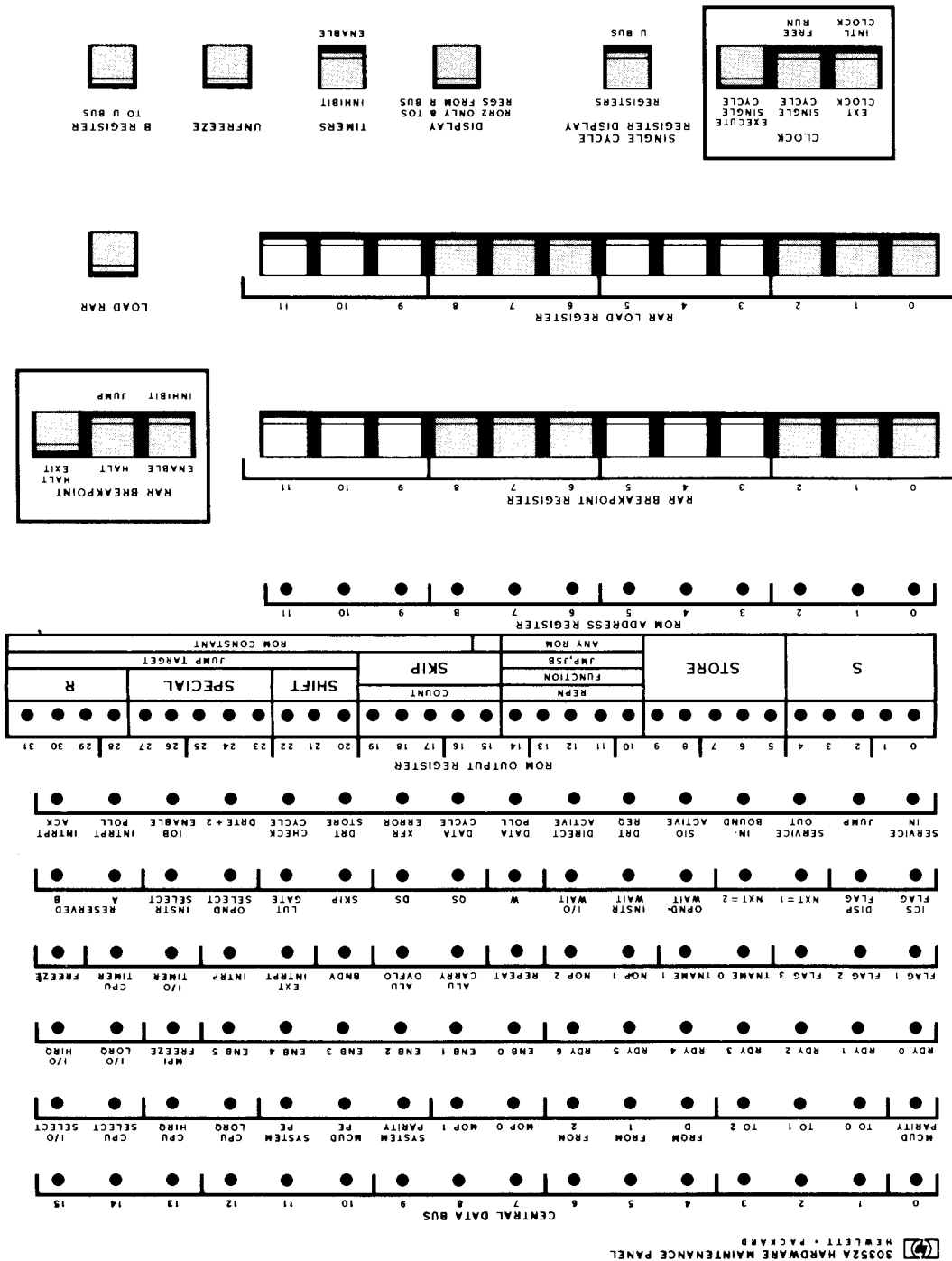
RUN-HALT

CPU RESET

I/O RESET

COLD LOAD

AUXILIARY CONTROL PANEL



CIR	Current Instruction Register — contains the current executing/ed, instruction.
CNTR	Counter register — used as a microcode counter. When in halt, contains the number of valid TOS hardware registers.
COR	CPU Output Register — serves as the data path to the central data bus for the CPU. Sourced either from the U-bus for Read/Write Address or Write Data or from the P-register for the read address of the next instruction to be fetched.
CPX1	CPU interrupt status (RUN) register — run mode status
CPX2	CPU interrupt status (HALT) register — halt mode status
DB	Data Base register — contains the absolute address of the first location of the stack global area.
DIR I/O DATA IN	Direct I/O Data In register — contains the data read by a "RIO" instruction.
DL	Data Limit register — contains the absolute address of the first location of the stack.
INTRPTG DVC NO	Interrupting Device Number register — contains the device number of the last device to have interrupted.
MUX I/O DATA	Multiplexor I/O Data register — contains the multiplexed data to be written to memory. Can only be read in single cycle with register display.
MASK	Mask register — contains the last interrupt mask word issued to all I/O devices. It does not necessarily reflect the state of a device's interrupt mask. This may have been set by a master reset to that device issued since a "MSK" instruction.
MEM ADRS	Memory Address — Scratch Pad 0 used in halt to contain the address of a "panel" memory display or store.
MEM DATA	Memory Data — Scratch Pad 1 used in halt to "panel" display memory data.
MOD NO.	Module Number register — contains the CPU ID number in bits 8-10 and (if) the interrupting module number in bits 0-2. The remaining bits are zero.
NIR	Next Instruction Register — receiver register for instructions returned to the CPU on the central data bus.
OPND	Operand register — receiver register for data returned to the CPU on the central data bus.
P	Program register — when in halt, contains the absolute address of the next instruction to be executed. When running, contains the absolute address of the next instruction to be fetched. States between the two are microcode dependent.

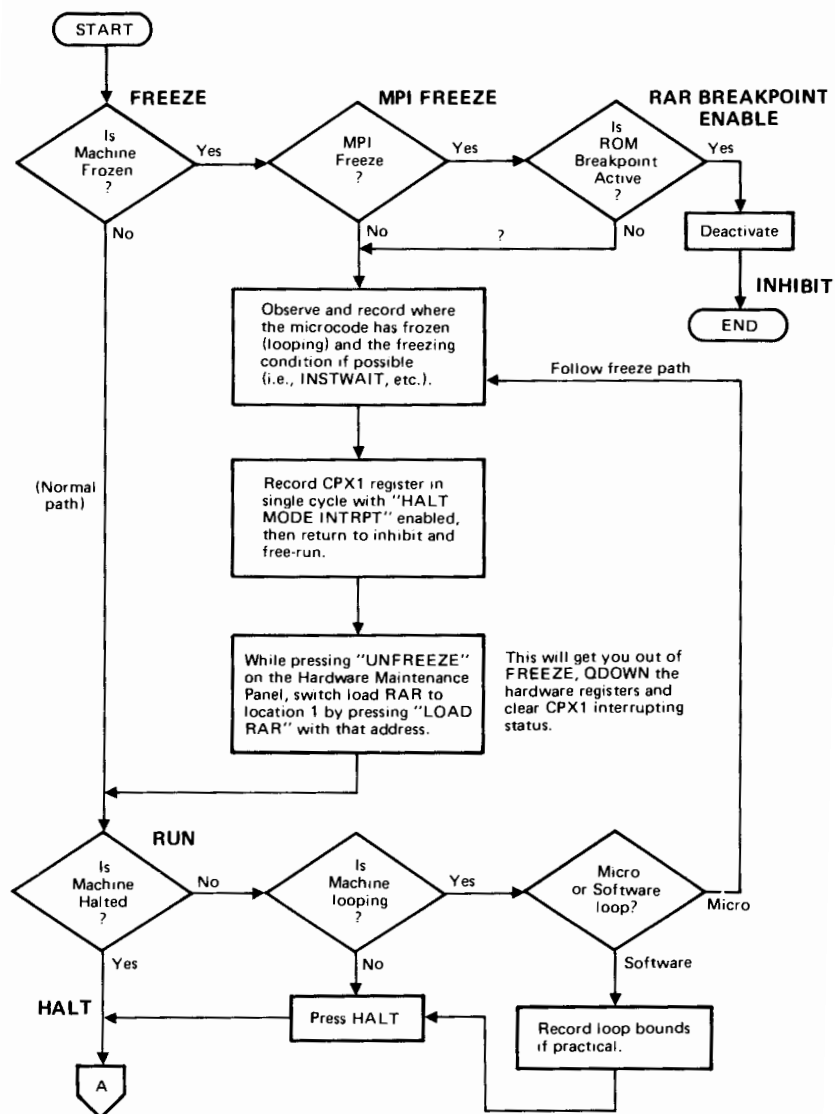
REGISTERS

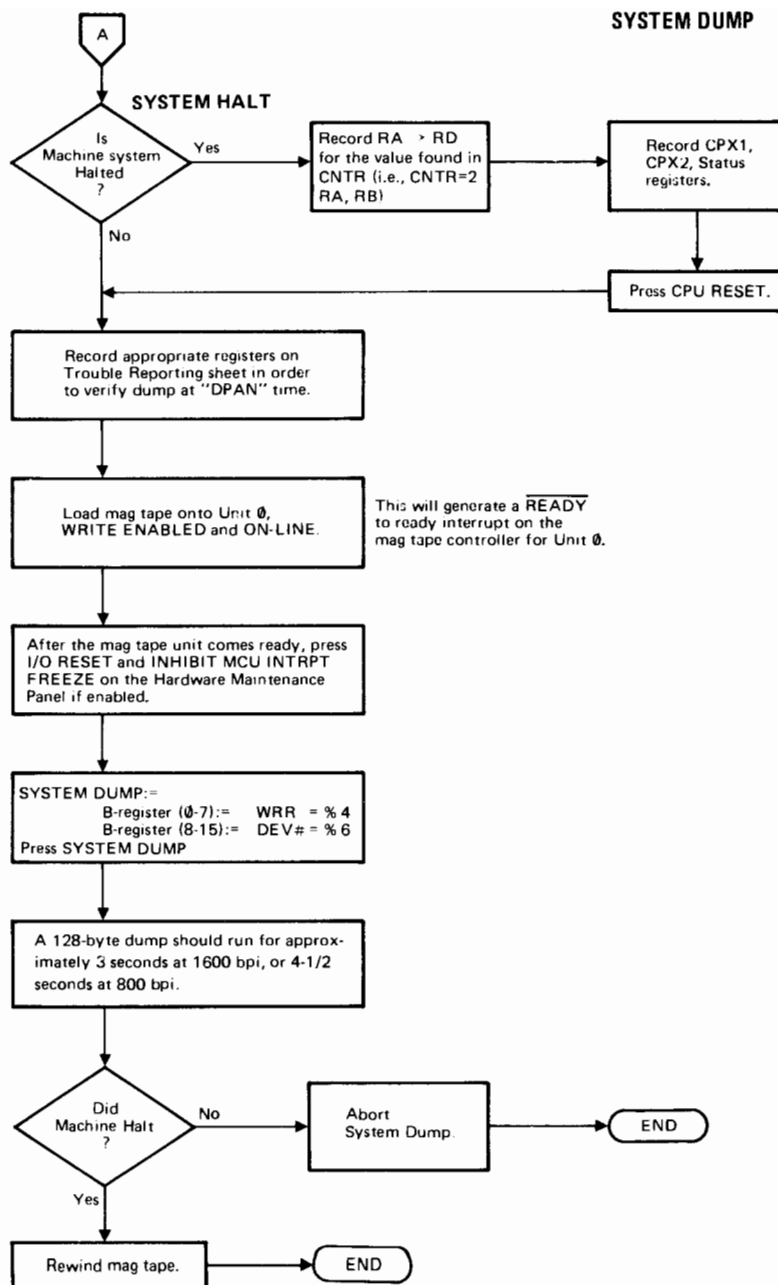
PADD	Padder — contains the hardware preadd result of the current executing instruction.
PB	Program Base register — contains the absolute address of the first location of the code segment.
PL	Program Limit register — contains the absolute address of the last location of the code segment, the start of the Segment Transfer Table.
Q	Q register — contains the absolute address of the first location of the current temporary stack.
RA,RB,RC,RD	Four TOS registers — they map thru the TNAME logic to the four hardware registers TR0, TR1, TR2, TR3.
RAR	ROM Address Register — contains the address of the current ROM instruction being fetched.
ROR1	ROM Output Register 1 — contains the contents of the addressed location previously specified by RAR (current RAR-1).
ROR2	ROM Output Register 2 — contains the previous contents of ROR1 or the contents of the addressed location specified two clock cycles prior by RAR (current RAR-2).
S	Stack (memory) register — contains the absolute address of the last word of the stack in memory. S as displayed by the "panel", is always the SM register. Pseudo register S is equal to SM+SR and when in halt, SR always equals zero; therefore SM=S. In other words, S is used rather than SM strictly for software purposes, although the display is always SM.
SAVE	Save register — contains the return address of a micro-code jump subroutine call.
SP2	Scratch Pad 2 — microcode scratch register.
SP3	Scratch Pad 3 — microcode scratch register.
SR	Stack Register — contains the number of valid hardware registers. SR is pushed into CNTR during halt, TOS registers "QDOWN"ed, and SR zeroed.
STATUS	Status register — contains the current CPU status.
TEST	Test register — an MPI register whose input is derived from an edge connector on the Maintenance Panel interface PCB.
X	X register — hardware index register and general purpose software register.
Z	Z register — contains the absolute address of the last available word of the data segment.

Information to be supplied later.

SYSTEM DUMP

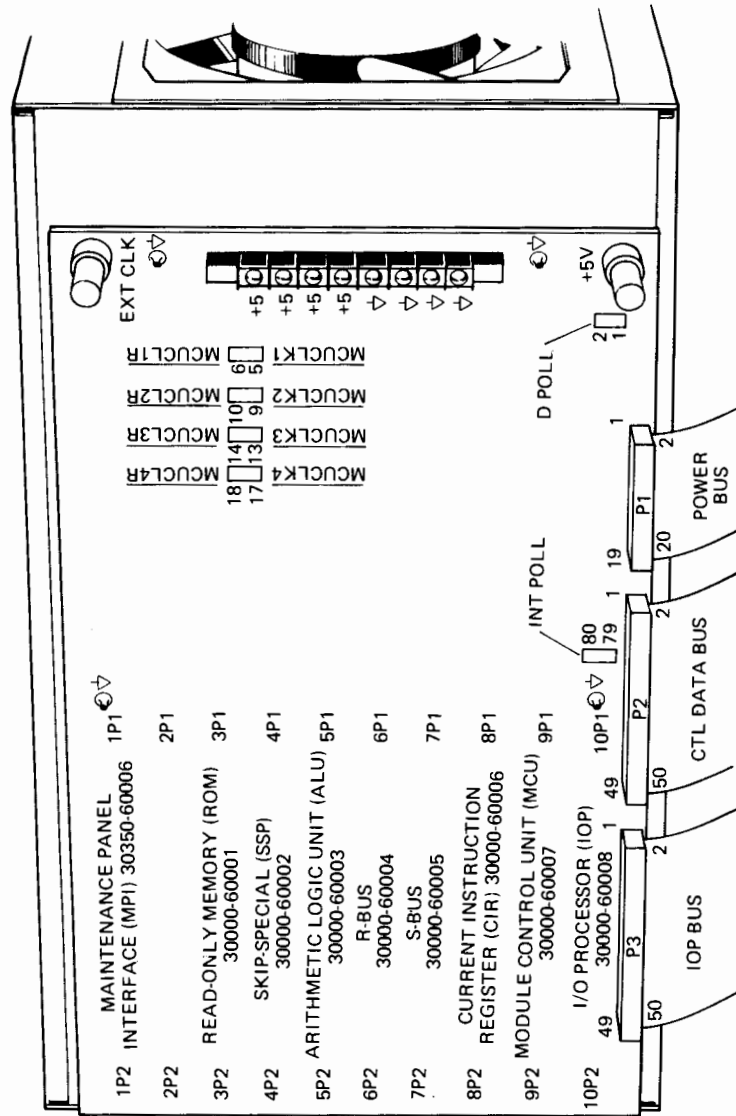
System Dump writes the hardware registers and all of memory to mag tape in one irrevocable pass. It can only be initiated from a HALT mode, although it executes in RUN. If the dump incurs a tape error during execution, it will never complete, while waiting for device status "SIO READY" to come true. For the most part, any error while attempting a dump will result in an invalid dump tape.

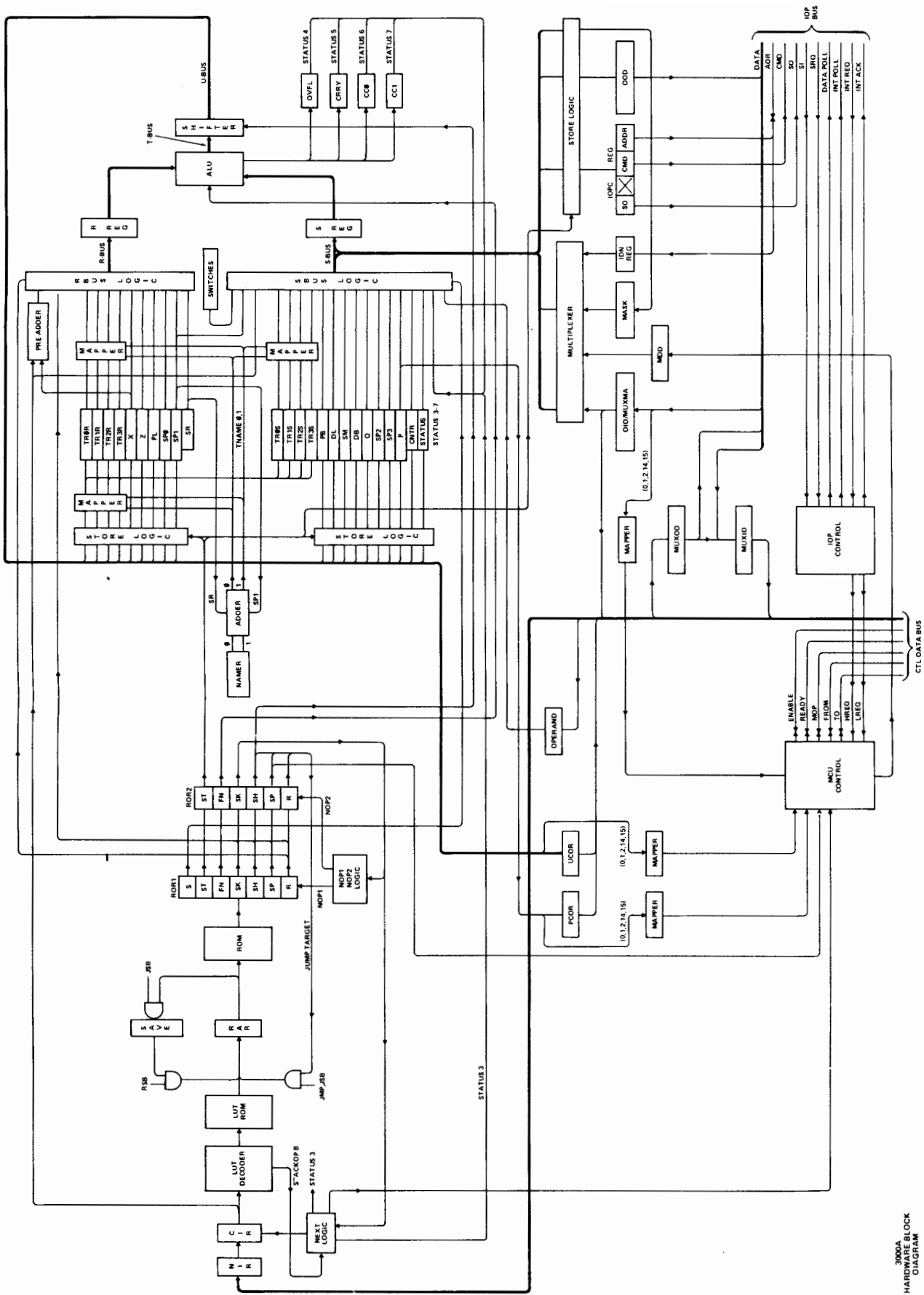




Information to be supplied later.

Diagram to be supplied later.

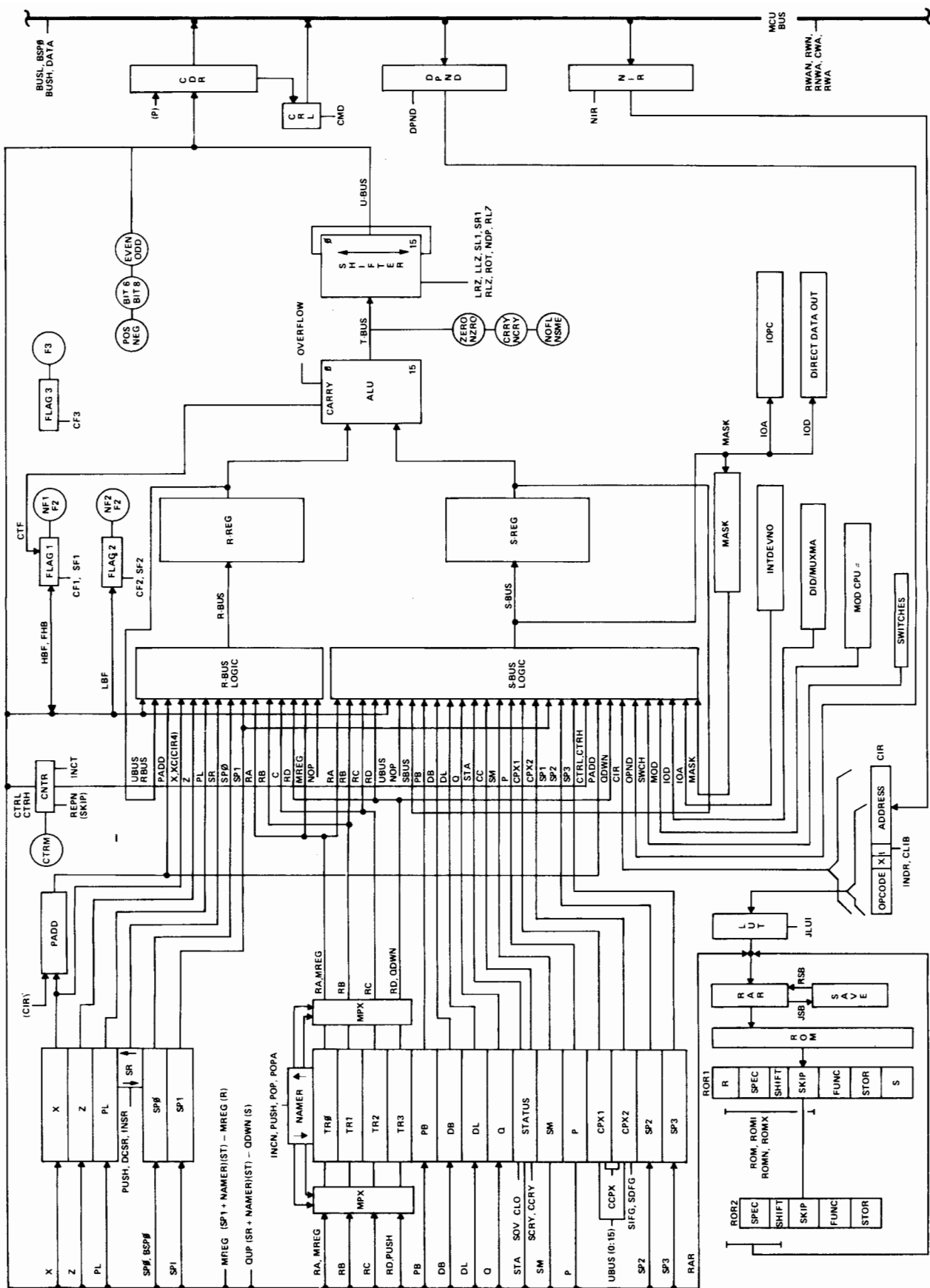


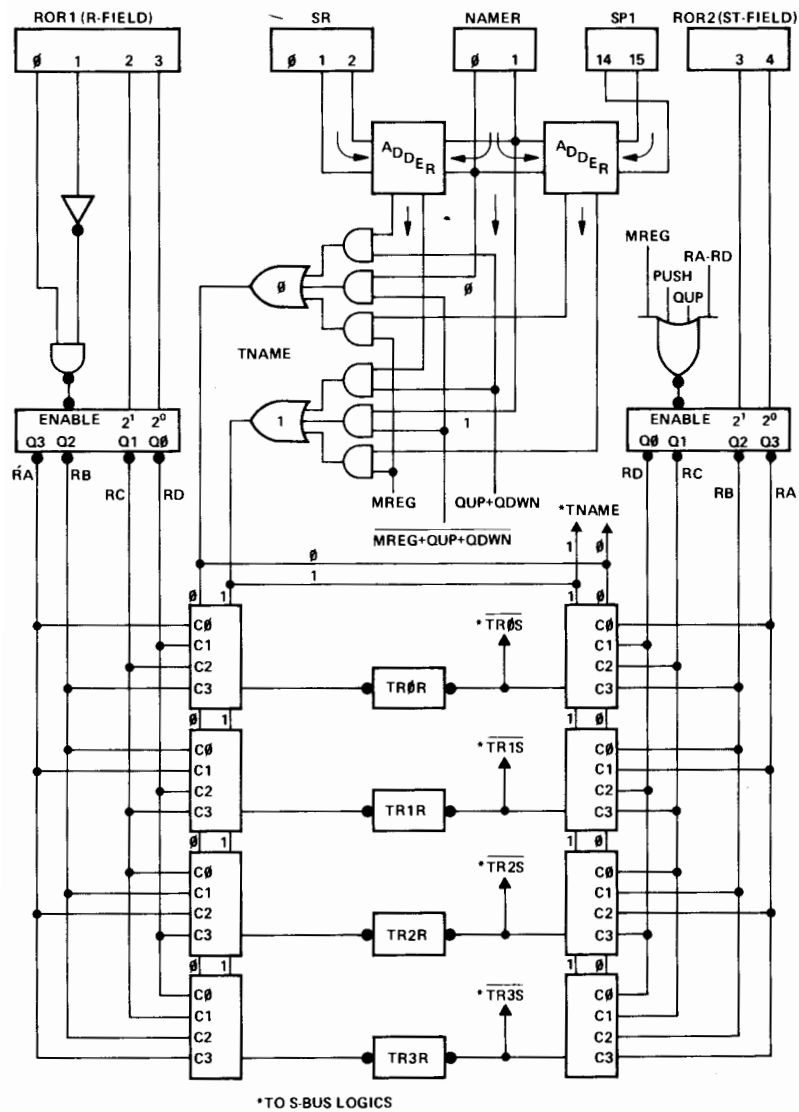


3000A
HARDWARE BLOCK
DIAGRAM

JUNE 1973

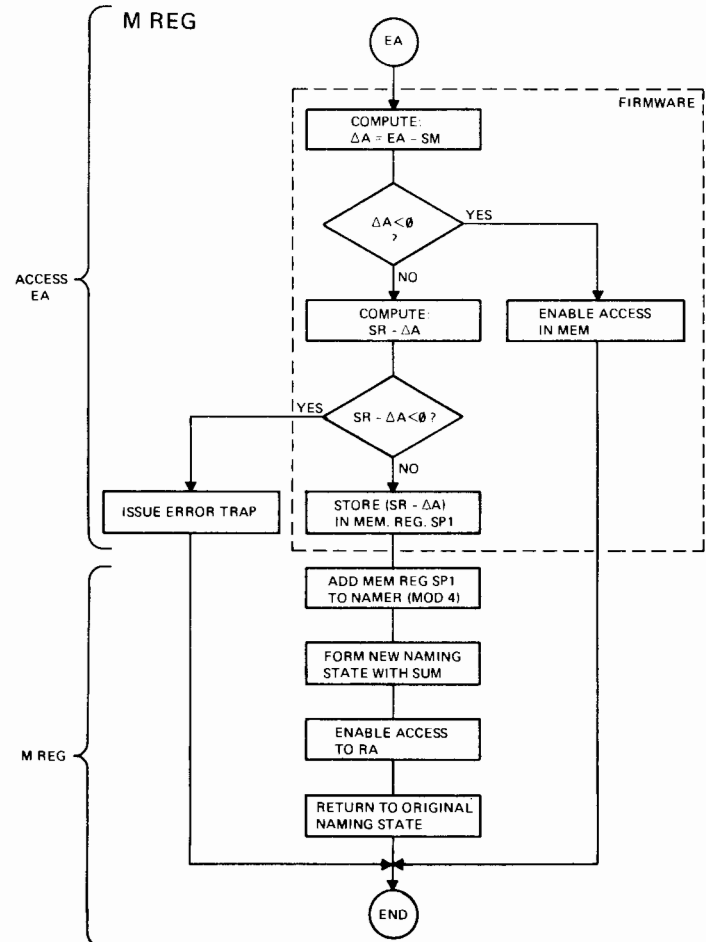
FIRMWARE BLOCK DIAGRAM



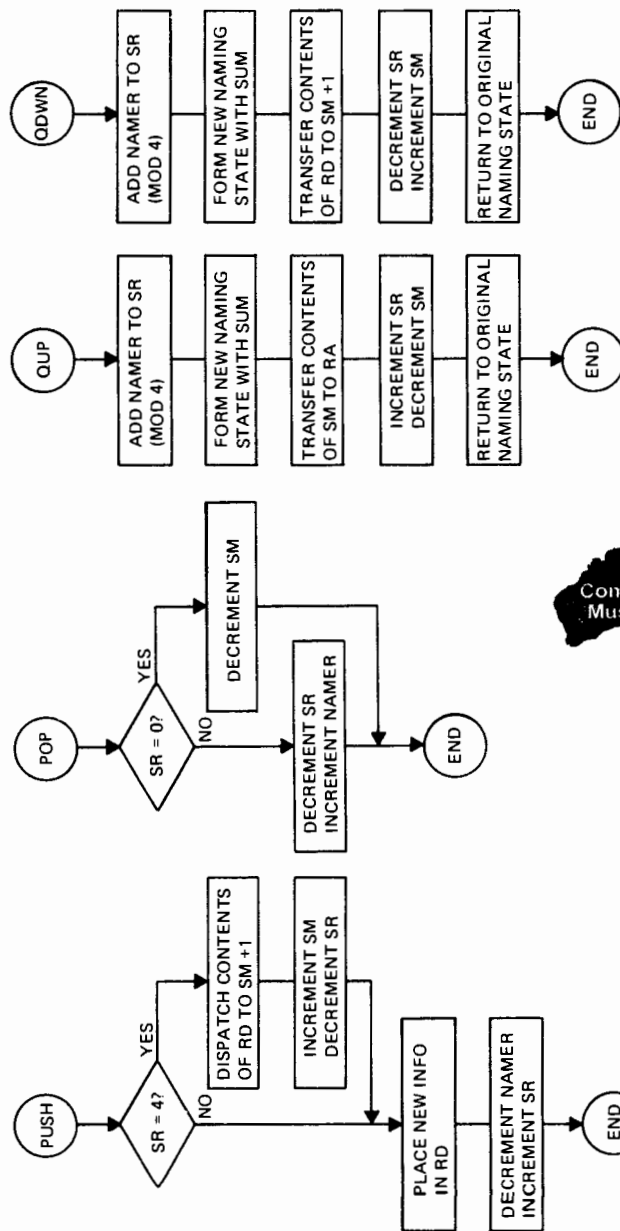


STACK OPERATIONS

NAMER		LOGICAL REGISTERS			
		RA	RB	RC	RD
NAMER VALUE	00	TR0	TR1	TR2	TR3
	01	TR1	TR2	TR3	TR0
	10	TR2	TR3	TR0	TR1
	11	TR3	TR0	TR1	TR2



EA = effective address
 ΔA = delta address from SM



indicates execution by firmware.

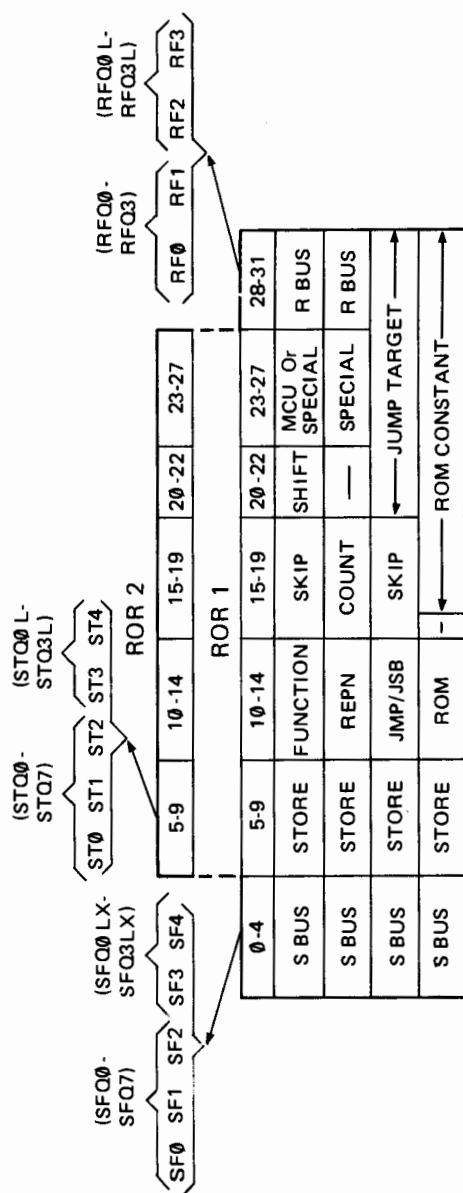
ROM FIELDS

	RBUS	SBUS	FCN	SHIFT	STORE	SPEC*	SKIP	MCU*	
00	PL	CIR	TASL	LRZ	MASK	CCB	ZERO		00
01	SR	SP1	TASR	LLZ	IOA	CCPX	NZRO		01
02	Z	PADD	ROMX	SL1	IOD	SR0	EVEN		02
03	MREG		ROMN	SR1	MREG	HALT	ODD		03
04	PADD	CPX1	JSB	RRZ	BUSL*	SIFG	NSME		04
05	RBUS	MOD	CAND	RLZ	BSP0*	SDFG	BIT6		05
06	X	CPX2	XOR	ROT	BUSH*	CTF	BIT8		06
07	XC	SWCH	AND	NOP	DATA	CF3	NOFL		07
10	RD	QDWN	DVSB		PUSH	INSR	CRRY		10
11	RC	IOA	UBNT		PL	DCSR	NCRY		11
12	RB	IOD	CADO		Z	INCN	POS		12
13	RA	MASK	SUBO		QUP	INCT	NEG		13
14	SP1	CTRL	JMP		SP1	HBF	F1		14
15	SP0	CTRH	BNDT		SP0	FHB	NF1		15
16	UBUS	UBUS	CAD		CTRL	CLIB	F2		16
17	NOP	SBUS	SUB		CTRH	LBF	NF2		17
20		P	PNLR		P	SF2	SRZ	RWAN	20
21		Q	PNLS+		Q	CF2	SRNZ	RWAN	21
22		DB	ROMI		SM	CF1	SR4	RWAN	22
23		SM	ROM+		DB	SF1	SRN4	RWAN	23
24		STA	REPC+		STA	SCRY	INDR	RWAN	24
25		SP3	REPN+		SP3	CCRY	SRL2	RWAN	25
26		OPND	IOR		X	POPA	NPRV	RWAN	26
27		CC	CTSD+		RAR	POP	SRL3	RWAN	27
30		RD	MPAD+		RD	SOV	RSB	CMD	30
31		RC	INCO+		RC	CLO	JLUI	CRL	31
32		RB	CRS+		RB	CCZ	TEST	NIR	32
33		RA	ADDO+		RA	CCL	CTRM	RWN	33
34		DL	CTSS+		DL	CCG	F3	OPND	34
35		SP2	INC+		SP2	CCE	NEXT	RNWA	35
36		PB	RPTY+		PB	CCA	UNC	CWA	36
37		NOP	ADD+		NOP	NOP	NOP	RWA	37

*These STORE options "NOP" the special field functions and cause the field to be used for MCU options.

+These functions cause an "ADD".

Will cause "JMP" freeze.



The "Rom Constant" is gated onto the R-Bus

ROM FIELDS

R-BUS FIELD ROM (28:31)	
MREG	SP1(14:15) are added to the contents of the namer register to get a temporary name — which is used to get RA to the R-bus register. A TOS used as a store option on the line immediately preceding this instruction will assume this temporary name. Note: the two bits of SP1 that are used for MREG are a separate physical register that does not shift with SP1 when it is used as a shift register.
NOP	(blank field). Zero is placed in the R-bus register.
PADD	PRE-ADDER contents . . .
PL	The Program Limit register (PL) . . .
RA	The TOS register as determined by the NAMER . . .
RB	The TOS-1 register as determined by the NAMER . . .
RBUS	The R-Bus register remains unchanged from the previous cycle.
RC	The TOS-2 register as determined by the NAMER . . .
RD	The TOS-3 register as determined by the NAMER . . .
SPØ	Scratch Pad register Ø (SPØ) . . .
SP1	Scratch Pad register 1 (SP1) . . .
SR	The Stack Register counter (SR) . . .
UBUS	The output of the Shifter (U-Bus) . . .
X	The Index Register (X) . . .
XC	X Conditional. (CIR(4) = indexing where allowed.) if CIR(4) = 0, then 0 . . . if CIR(4) = 1, then X . . .
Z	The Stack Limit Register (Z) . . .

. . . "is placed in the R-Bus register"

S-BUS FIELD		ROM (0:4)
CC	Condition code. S-BUS (8:9) := STATUS (6:7); and, if STATUS (6:7) = 00 then S-Bus (7) :=1, else S-Bus (7):=0.	27
CIR	The contents of Current Instruction Reg . . .	00
CPX1	Interrupt status register. Run mode status.	04
CPX2	Interrupt status register. Halt mode status.	06
CTRH	Counter High. S-BUS REG (4:9) :=CNTR(0:5) (used mostly in floating point exponent manipulations).	15
CTRL	Counter Low. S-BUS REG(10:15) :=CNTR(0:5).	14
DB	The Data Base register (DB) . . .	22
DL	The Data Limit register (DL) . . .	34
IOA	I/O Address register . . . Contains last interrupting device #.	11
IOD	The I/O Data register . . .	12
MASK	The I/O Mask register . . .	13
MOD	Module number. A constant is brought to the S-Bus register in the following way: S-Bus (0:4) :=0; S-Bus (5:7) :=interrupting module number; S-Bus (8:15) :=4(if CPU1), 8 (if CPU2).	05
NOP	Zero . . .	37
OPND	The Operand register (data from memory) . . .	26
P	The Program counter (P) . . .	20
PADD	The PRE-ADDER . . .	02
PB	The Program Base register (PB) . . .	36
Q	The Stack Marker Pointer register (Q) . . .	21
QDWN	It takes the TOS register at logical address SM+1 and puts it in the S-Bus register in the following way: The TOS registers are renamed by Namer + SR. RD is dispatched to the S-Bus and TOS registers are returned to their former names. A TOS register used in the STORE field of the previously executed instruction will assume a temporary name.	10

. . . "is placed in the S-Bus register".

ROM FIELDS

RA	The TOS register (RA) . . .	33
RB	The TOS-1 register (RB) . . .	32
RC	The TOS-2 register (RC) . . .	31
RD	The TOS-3 register (RD) . . .	30
SBUS	The S-Bus register remains unchanged from the previous cycle.	17
SM	The Top of Stack memory pointer (SM) . . .	23
SP1	Scratch Pad register 1 (SP1) . . .	01
SP2	Scratch Pad register 2 (SP2) . . .	35
SP3	Scratch Pad register 3 (SP3) . . .	25
STA	The Status register (STA) . . .	24
SWCH	The Switch Register . . .	07
UBUS	The output of the Shifter (U-Bus) . . .	16

. . . "is placed in the S-Bus register".

FUNCTION FIELD ROM(10:14)

ADD	The contents of the R-Bus and the S-Bus registers are added and the result is left on the T-Bus. (NOP condition.)	37
ADD0	The contents of the R-Bus and the S-Bus are added and the result is left on the T-Bus. The overflow and carry bits are enabled. CCA is set from the U-Bus.	33
AND	The logical AND of the R-Bus and the S-Bus is left on the T-Bus.	07
BNDT	Hardware bounds test. T-Bus:=R-Bus - S-Bus. If Carry = 1 continue. If Carry = 0, and in user mode, jump to boundary violation routine at RAR = 2 (takes precedence over Skip field).	15
CAD	Complement and add. T-Bus := R-Bus + 1's complement of S-bus.	16
CADO	Same as CAD except that Carry and Overflow bits are enabled. CCA on U-Bus.	12
CAND	T-Bus := R-Bus AND 1's complement of S-Bus.	05

FUNCTION FIELD (Continued)

CRS	Circular shift one bit, implied T-Bus := R-Bus + S-Bus. If SR1: right, and U(0) := T(15). If SL1: left, and U(15) := T(0).	32
CTSD	This function performs a double register shift of the T-Bus and a scratch pad register. A left shift, indicated by an SL1 in the shift field, expects the least significant word in SP1. A right shift (SR1) expects the least significant word in SP3. The type of shift is determined from the contents of the CIR as follows: T-Bus := R-Bus + S-Bus implied. CIR(7) = 1 Circular shift CIR(7:8) = 0,1 Logical shift CIR(7:8) = 0,0 Arithmetic shift Note: both SP1 and SP3 are shifted regardless of the direction of the shift.	27
CTSS	The T-Bus is shifted in a manner determined by CIR(7:8:9) as follows. Implied T-Bus := R-Bus + S-Bus. CIR(7) = 1 Circular shift CIR(7:8) = 0,1 Logical shift CIR(7:8) = 0,0 Arithmetic shift CIR(9) = 1 Right shift direction CIR(9) = 0 Left shift direction } interpreted by firmware	34
DVSB	This function performs the subtract, shift, and test necessary to implement a divide algorithm. To start, F2 = 0, the divisor is in the S-Reg, and the double word dividend is in the R-Reg (MSW) and SP1. SL1 must be in the shift field. One bit quotient comes in SP1 (15). T-Bus := R-Bus - S-Bus; U-Bus (0:14) := T-Bus (1:15); R-Reg (0:14) := if (ALU carry OR F2), then U-Bus (0:14) else R-Reg (1:15); R-Reg (15) := SP1(0); SP1(0:14) := SP1(1:15); SP1(15) := if (ALU carry OR F2) then 1 else 0. F2 := if (ALU carry OR F2) then T-Bus (0) else R-REG (0).	10
INC	T-Bus := R-Bus + S-Bus + 1.	35
INCO	Same as INC with the addition that the carry and overflow bits are enabled. CCA is set from the U-Bus.	31
IOR	The R-Bus and S-Bus are logically ORed together and the result placed on the T-Bus.	26
JMP	This function performs a micro-jump to the ROM address specified in bits 20 to 31 if the condition contained in the skip field is met. If the skip condition is not met, the next ROM instruction in sequence is fetched. Also, implied U-Bus := T-Bus := S-Bus. Execution requires two clock cycles.	14
JUNE 1973		109

FUNCTION FIELD (Continued)

JSB	This function causes a subroutine jump, and is executed like the JMP function with the following addition. The RAR register is stored into the SAVE register before the target address is transferred from the ROR. The return address is saved in the SAVE register. Since there is 1 SAVE, subroutines may be nested only 1 deep.	04
MPAD	This function performs the shift, test, and add functions necessary to implement a multiply algorithm. To start, the multiplier is in SP3, the multiplicand is in the R-Bus register, and the S-Bus register = 0. An SR1 is required in the shift field. One bit result comes in SP3(0). T-Bus := R-Bus + S-Bus; U-Bus(1:15) := T-Bus(0:14); U-Bus(0) := ALU carry; if SP3(15) = 1, then [S-Reg := U-Bus, SP3(1:15) := SP3(0:14), SP3(0) := T(15)], else [S-Reg(1:15) := S-Reg(0:14), SP3(1:15) := SP3(0:14), SP3(0) := S-Reg(15)].	30
PNLR	A control function. The appropriate register selected from the front panel is brought to the T-Bus through ALU.	20
PNLS	A control function. The U-Bus is stored in the appropriate register selected from the front panel.	21
REPC	"Repeat until condition". The next microinstruction is repeated until the skip condition is met. Implied T-Bus := R-Bus + S-Bus.	24
REPN	"Repeat number of times". The skip field constant is sent to the CNTR. CNTR(1:5) := ROR2(15:14); CNTR(0) := 1. The value is stored in 2's complement form. The next microinstruction is repeated until skip condition is met. Typically the special field is INCT and the skip condition is CTRM (counter rolls over by end of cycle). Implied T-Bus := R-Bus + S-Bus.	25
ROM	ROM constant. R-Bus := ROM(16:31). Implied T-Bus := R-Bus + S-Bus. Overrides shift, skip, special and R fields.	23
ROMI	Same as ROM except implied inclusive — OR of R-Bus and S-Bus.	22
ROMN	This function is like ROM except implied T-Bus := R-Bus AND S-Bus.	03
ROMX	This function is like ROM except implied T-Bus := R-Bus XOR S-Bus.	02
RPTY	T-Bus := R-Bus + S-Bus. This function reverses parity generation for diagnostic use only.	36

FUNCTION FIELD (Continued)

SUB	T-Bus := R-Bus - S-Bus.	17
SUBO	Like SUB, except carry and overflow bits enabled. CCA is set from the U-Bus.	13
TASL	"Triple Arithmetic Shift Left". SL1 register in shift field. Shifter ← SP1 ← R-Bus ← T-Bus := S-Reg U(0):=T(0), T(1) is lost, U(1:14):=T(2:15), U(15):=SP1(0), SP1(0:14):=SP1(1:15), SP1(15):=R(0), R(0:14):=R(1:15), R(15):=0	00
TASR	"Triple Arithmetic Shift Right". SR1 register in shift field. Shifter → SP3 → S-Bus → T-Bus := R-Reg U(0:1):=T(0), U(2:15):=T(1:14), SP3(0):=T(15), SP3(1:15):=SP3(0:14), S(0):=SP3(15), S(1:15):=S(0:14), S(15) is lost.	01
UBNT	Unconditional bounds test. T-Bus := R-Bus - S-Bus . If no carry then a hardware jump to the bounds violation routine is invoked. (RAR := 2).	11
XOR	T-Bus := R-Bus EXCLUSIVE OR S-Bus.	06

ALU FUNCTION ENCODING FOR 1820-0606

OP	FNCT.	S ₃	S ₂	S ₁	S ₀	ALUMODE	ALUFC	ADD	SUB	IOR	AND	XCR	CAND	A	B
00	TASL	1	0	1	0	1	1								X
01	TASR	1	1	1	1	1	1							X	
02	ROMX	0	1	1	0	1	1					X			
03	ROMN	1	0	1	1	1	1				X				
04	JSB	1	0	1	0	1	1								X
05	CAND	0	1	1	1	1	1						X		
06	XOR	0	1	1	0	1	1					X			
07	AND	1	0	1	1	1	1				X				
10	DVSB	0	1	1	0	0	0		X						
11	UBNT	0	1	1	0	0	0		X						
12	CADO	0	1	1	0	0	1		X						
13	SUBO	0	1	1	0	0	0		X						
14	JMP	1	0	1	0	1	1								X
15	BNDT	0	1	1	0	0	0		X						
16	CAD	0	1	1	0	0	1		X						
17	SUB	0	1	1	0	0	0		X						
20	PNLR +	1	0	0	1	0	1	X							
21	PNLS +	1	0	0	1	0	1	X							
22	ROMI	0	0	0	1	0	1			X					
23	ROM +	1	0	0	1	0	1	X							
24	REPC +	1	0	0	1	0	1	X							
25	REPN +	1	0	0	1	0	1	X							
26	IOR	0	0	0	1	0	1			X					
27	CTSD +	1	0	0	1	0	1	X							
30	MPAD +	1	0	0	1	0	1	X							
31	INCO +	1	0	0	1	0	0	X							
32	CRS +	1	0	0	1	0	1	X							
33	ADDO +	1	0	0	1	0	1	X							
34	CTSS +	1	0	0	1	0	1	X							
35	INC +	1	0	0	1	0	0	X							
36	RPTY +	1	0	0	1	0	1	X							
37	ADD +	1	0	0	1	0	1	X							
—	NOP2 +	1	0	0	1	0	1	X							

NOTES:

+ Denotes functions causing an ADD

A = T Bus := R Reg

B = T Bus := S Reg

	SHIFT FIELD	ROM(20:22)	
LLZ	“Left to left and zero” U-Bus(0:7) := T-Bus(0:7) U-Bus(8:15) := 0		01
LRZ	“Left to right and zero” U-Bus(0:7) := 0 U-Bus(8:15) := T-Bus(0:7)		00
RLZ	“Right to left and zero” U-Bus(0:7) := T-Bus(8:15) U-Bus(8:15) := 0		05
ROT	“Rotate” U-Bus(0:7) := T-Bus(8:15) U-Bus(8:15) := T-Bus(0:7)		06
RRZ	“Right to right and zero” U-Bus(0:7) := 0 U-Bus(8:15) := T-Bus(8:15)		04
SL1	“Shift left one” U-Bus(J) := T-Bus(J+1) U-Bus(15) := 0 T-Bus(0) is lost [When used with CRS, CTSD, DVSB, and TASL implies left shift.]		02
SR1	“Shift right one” U-Bus(J) := T-Bus(J-1) U-Bus(0) := 0 T-Bus(15) is lost [When used with CRS, CTSD, MPAD, and TASR implies right shift.]		03
NOP	(blank) Straight through. U-Bus (0:15):=T-Bus (0:15).		07

ROM FIELDS

STORE FIELD		ROM(5:9)
BSP0	Stores U-Bus into the COR and into SPO and initiates a Bus low request. It disables the special field and enables the MCU options, one of which must be used.	05
BUSH	Stores U-Bus in COR (CPU Output Register) and makes an MCU BUS-Hi request.	06
BUSL	COR := U-Bus. Initiates a low request for the bus. It disables the special field and enables the MCU options, one of which must be used.	04
CTRH	Counter High. CNTR := U-Bus(4:9).	17
CTRL	Counter Low. CNTR :=U-Bus(10:15)	16
DATA	Stores the U-Bus into the COR and issues a store command to memory which stores the contents of COR in the address held by the memory address register in the memory module previously accessed by a BUSL or a BSP0.	07
DB	Stores the U-Bus into the Data Base Register (DB).	23
DL	... Data Limit Register (DL).	34
IOA	... I/O Address Register. Contains command and device #.	01
IOD	... I/O Data Register. Contains output data to IOP data bus.	02
MASK	... Mask Register.	00
MREG	The contents of Namer is added to two bits SP1(14:15) to obtain temporary name. With this name, RA is picked and U-Bus is stored in it. TOS registers used in the R and S fields following this instruction will assume the temporary name.	03
NOP	(blank). No store.	37
P	... Program Counter (P).	20
PB	... Program Base Register (PB).	36
PL	... Program Limit Register (PL).	11
PUSH	Stores the U-Bus into the RD register, increments the SR register by one and at the end of the microinstruction cycle renames the TOS registers such that: N(RA) := RB, N(RB) :=RC, N(RC) := RD, N(RD) := RA.	10
Q	... Stack Marker Pointer (Q).	21
... "Stores the U-Bus into the"		

STORE FIELD (Continued)

QUP	The TOS registers are renamed by NAMER + SR. Temporarily named RA := U-Bus. The TOS register names are returned to NAMER. Incrementing of SR is not implicit. INSR (inc. SR) must appear in the special field in order to increment SR. TOS registers used in the R and S fields following this instruction will assume the temporary name.	13
RA	... TOS register (RA)*.	33
RAR	ROM Address Register. RAR(0:11) := U-Bus(4:15).	27
RB	... TOS-1 register (RB)*.	32
RC	... TOS-2 register (RC)*.	31
RD	... TOS-3 register (RD)*.	30
SM	... Memory Stack Pointer (SM).	22
SPØ	... Scratch Pad Register 0 (SPO).	15
SP1	... Scratch Pad Register 1 (SP1).	14
SP2	... Scratch Pad Register 2 (SP2).	35
SP3	... Scratch Pad Register 3 (SP3).	25
STA	... Status Register.	24
X	... Index Register (X).	26
Z	... Stack Limit Pointer (Z).	12

... "Stores the U-Bus into the"

*TOS Register designed by the NAMER at the beginning of the cycle.

ROM FIELDS

SPECIAL FIELD ROM(23:27)

NOTE: If store field is "BUSH", "BSPO" or "BUSL" then *special field* is disabled and *MCU option* field is enabled.

CCA	Sets Condition Code A in STATUS word: CCL if T-Bus < 0. STA(6:7) := 01 CCE if T-Bus = 0. STA(6:7) := 10 CCG if T-Bus > 0. STA(6:7) := 00	36
CCB	Sets Condition Code B in STATUS word: CCG if $60 \leq \text{T-Bus} \leq 71$ STA(6:7) := 00 CCE if $101 \leq \text{T-Bus} \leq 132$ or 141 $\leq \text{T-Bus} \leq 172$ STA(6:7) := 10 otherwise CCL STA(6:7) := 01	10
CCE	Sets Condition Code equal: STA(6:7) := 10.	35
CCG	Sets Condition Code greater: STA(6:7) := 00.	34
CCL	Sets Condition Code less than: STA(6:7) := 01	33
CCPX	Clears the Interrupt Status Register (CPX1/CPX2) bits as specified by the true bits on the U-Bus. Refer to Page B-3.	01
CCRY	Clear the Carry bit in the STATUS word. STA(5) := 0.	25
CCZ	Set Condition Code Zero: CCE if U-Bus = 0. CCG if U-Bus $\neq$ 0.	32
CF1	At the end of the cycle CF1 clears Flag 1.	22
CF2	... CF2 clears Flag 2.	21
CF3,	... CF3 clears Flag 3.	07
CLIB	... CLIB clears the indirect line. Equivalent to having CIR(5) := 0.	16
CLO	... CLO clears the overflow bit in the STATUS word STA(4) := 0.	31
CTF	Sets the Carry to Flag 1 at the end of the cycle.	06
DCSR	Decrements the SR counter by 1.	11
INCN	Increments the Namer by 1.	12
FHB	Flag to high bit. U-Bus (0) := FLAG1.	15
HALT	Enables the front panel by setting CPX1(0) := 0.	03

[ASCII
ranges
in octal]

... "At the end of the cycle"

SPECIAL FIELD (Continued)

HBF	High bit to Flag. FLAG 1 := U-Bus(0).	14
INCT	Increments the counter by 1 (modulo 64).	13
INSR	Increment SR by 1.	10
LBF	Low bit to Flag 2. F2 := U-Bus(15).	17
NOP	(blank) No Special Option.	37
POP	This option decrements the SR by 1 and then renames the TOS registers such that: N(RA) := RD, N(RB) := RA, N(RC) := RB, N(RD) := RC.	27
POPA	Exactly like POP, except CCA is set on the contents of the U-Bus.	26
SCRY	Set the carry bit in the STATUS word. STA(5) := 1.	24
SDFG	Sets the dispatcher flag CPX2(12) := 1.	05
SF1	Sets FLAG1 at the end of the cycle.	23
SF2	Sets FLAG 2 . . .	20
SIFG	Set the Interrupt Stack flag CPX2(11) := 1.	04
SOV	Sets the Overflow bit in the STATUS word . . . STA(4) := 1.	30
SRO	Sets SR to zero during the cycle (not a clocked function). Cannot be used with any other SR dependent option.	02

. . . "at the end of the cycle."

SKIP FIELD ROM(15:19)

The skip field does one of two things:

1. Sets the Condition Met Flag or
2. Initiates a hardware micro-jump. A hardware micro-jump needs no jump target in the micro-instruction.

The condition met flag after a REPN or REPC function option indicates the condition on which to terminate the repeated micro-instruction. Otherwise it indicates that the next micro-instruction is to be skipped.

BIT6	Condition met if U-Bus (6) = 1.	05
BIT8	... U-Bus (8) = 1.	06
CRRY	... the carry out of the ALU is a one. (Note this is <i>not</i> the carry bit in the status word.)	10
CTRM	... the counter is all ones at the beginning of the current cycle (ends up at CNTR = 0).	33
EVEN	... the U-Bus (15) = 0.	02
F1	... *** flag 1 is set.	14
F2	... *** flag 2 is set.	16
F3	... *** flag 3 is set.	34
INDR	... the Indirect bit of the current instruction register is set CIR(5) = 1.	24
JLUI	Conditional hardware microjump to the address that the look-up table is displaying if the indirect signal is not set (CLIB must be used to guarantee a jump on all instructions). Normally LUT bit W will have changed for this execute phase. Indirect signal = $(\text{CIR}(4) \cdot \text{memory ref group} + \text{CIR}(5) \cdot \text{memory ref group}) \text{CLIBFF}$.	31
NCRY	... the carryout of the ALU is zero. (Note this is <i>not</i> the carry bit in the status word.)	11
NEG	... U-Bus(0) = 1.	13
NEXT	This terminates the microcode sequence and initiates the fetch of the next instruction. If current instruction is STACK OP A, and STACK OP B is not NOP, then it executes STACK OP B. In all other cases, CIR := NIR. CIR is decoded, PREADDER is sent to R-Bus (and in Mem Ref Inst the Base register is sent to the S-Bus) thus priming the Pipe Line Registers. COR := P and a "Read/Write to Next" bus request is made. P := P+1, and NIR awaits the new instruction for prefetch.	35
	... "Condition met if" ***"at the beginning of the cycle"	

SKIP FIELDS (Continued)

NF1	... *** flag 1 is cleared.	15
NF2	... *** flag 2 is cleared.	17
NOFL	... *** no overflow out of the ALU occurs. (Note this is <i>not</i> the overflow bit in the status word.)	07
NOP	(blank). No skip option — never skip.	37
NPRV	... *** the privileged mode bit is not set.	26
NSME	... all the bits of the T-Bus are not the same.	04
NZRO	... the T-Bus is non-zero.	01
ODD	... the U-Bus (15) = 1.	03
POS	... the U-Bus (0) = 0.	12
RSB	Hardware micro-jump to the address held in the SAVE register. The SAVE register contents are transferred to the RAR.	30
SR4	... the SR register is 4.	22
SRL2	... the SR register is less than 2.	25
SRL3	... the SR register is less than 3.	27
SRN4	... the SR register is not 4.	23
SRNZ	... the SR register is non-zero.	21
SRZ	... the SR register is zero.	20
TEST	... the external or internal interrupt line is high.	32
UNC	Condition met unconditionally.	36
ZERO	... the T-Bus is zero.	00

... "condition met if"

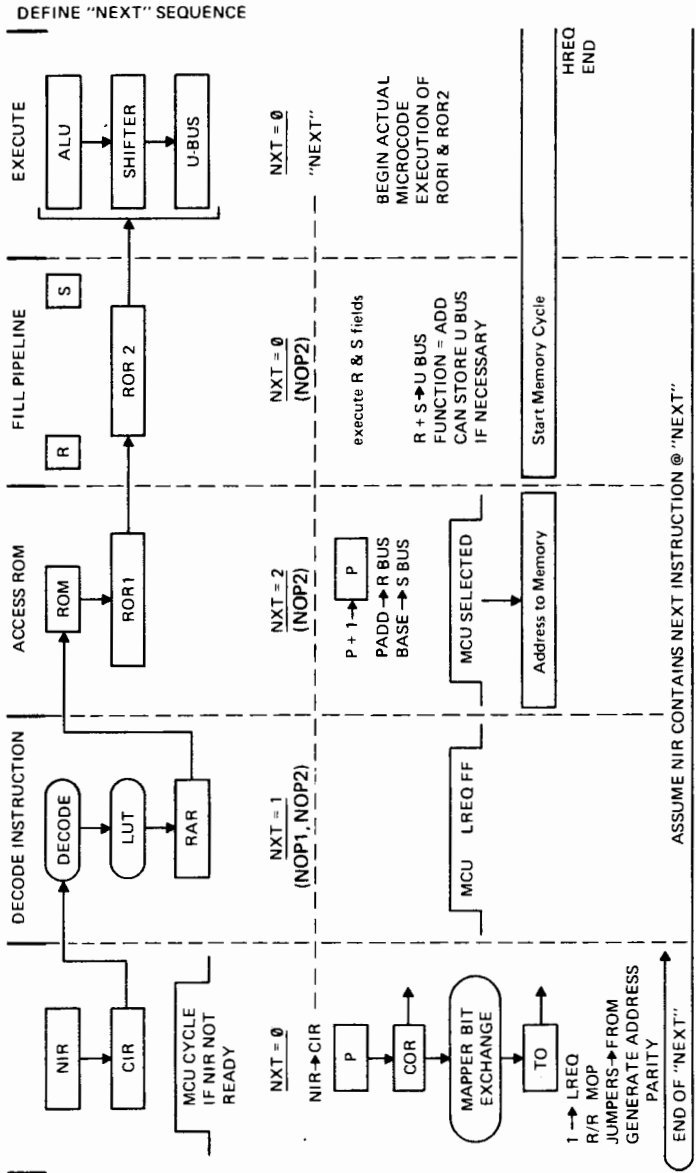
*** "at the beginning of the cycle"

ROM FIELDS

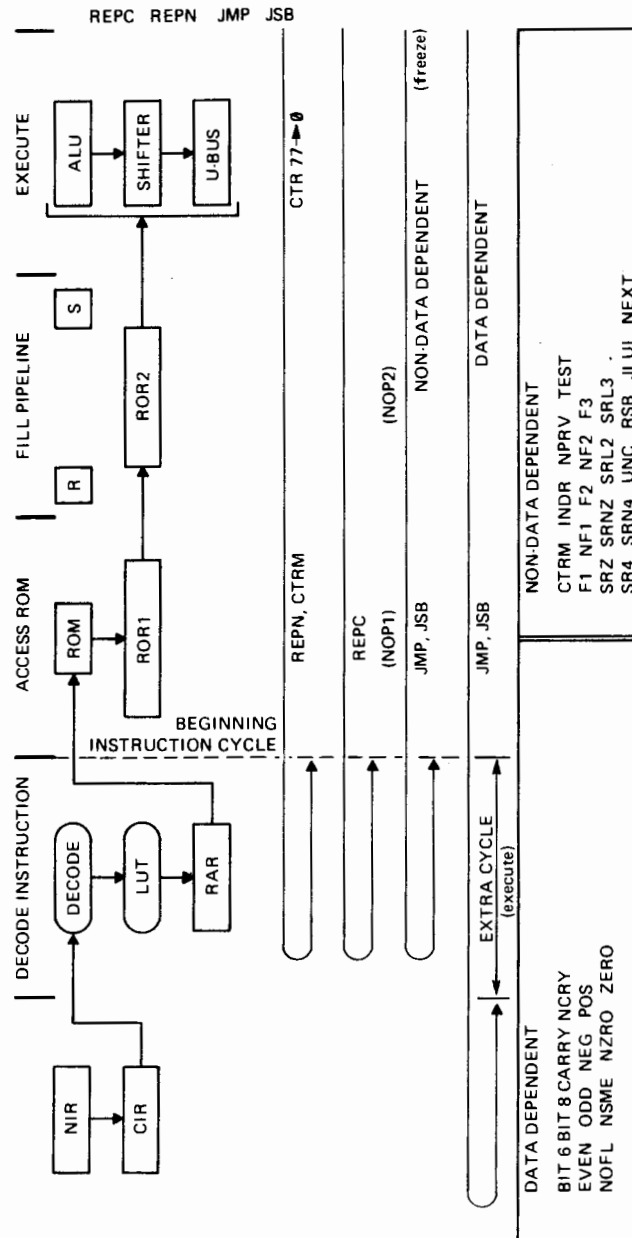
MCU OPTIONS ROM(23:27)

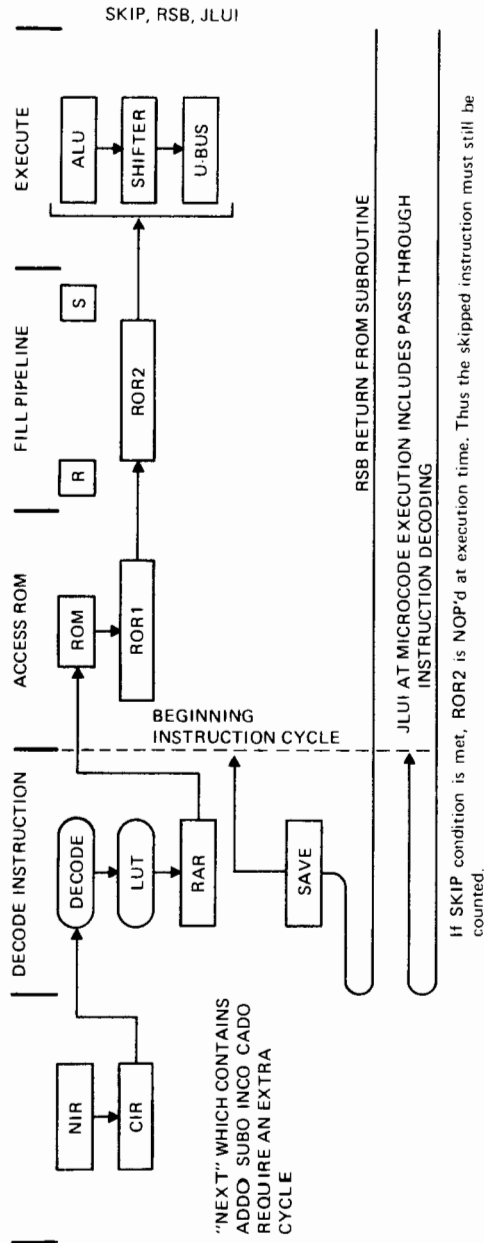
(Override special field if there is "BUSH", "BSPO" or "BUSL" in store field.)

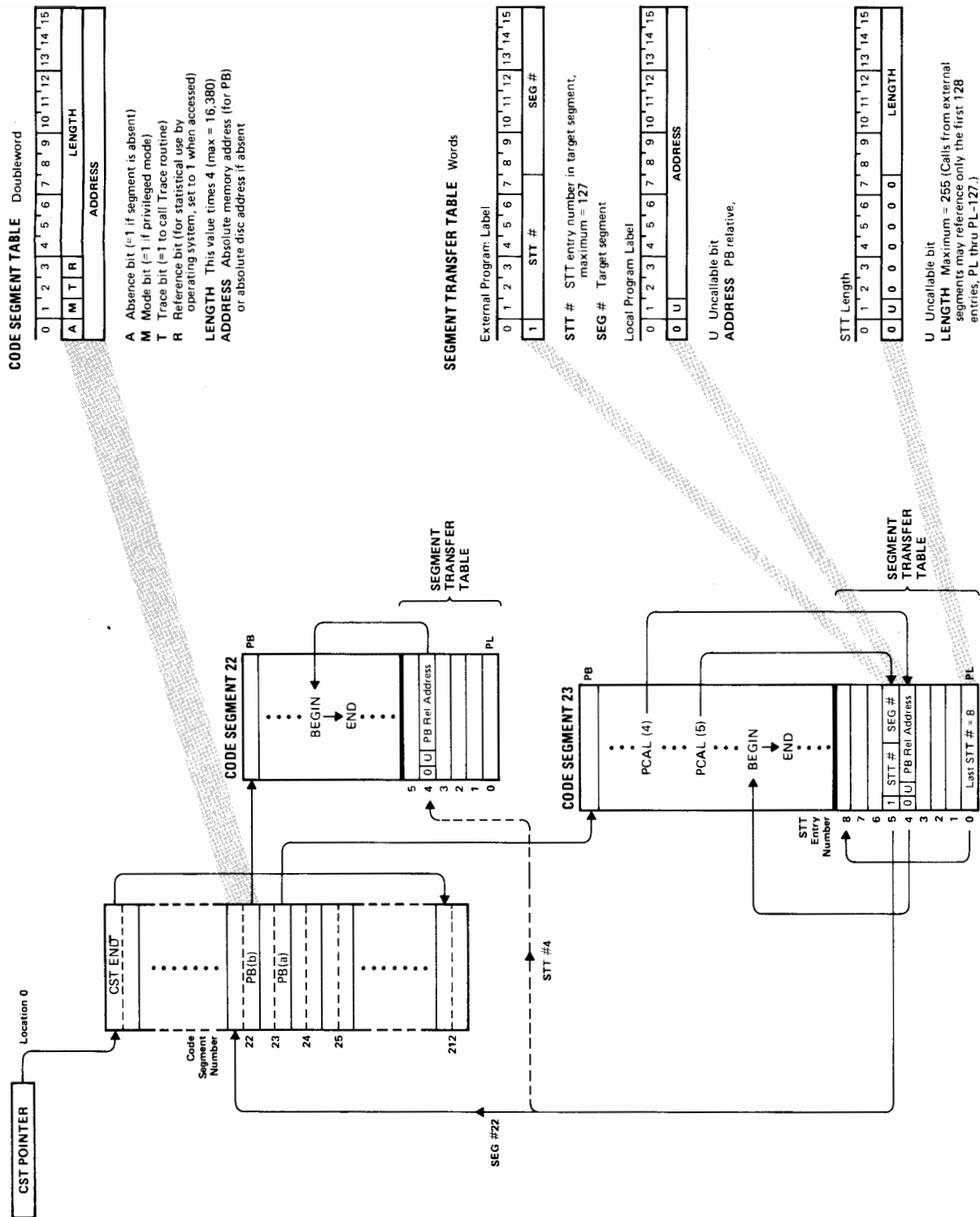
CMD	MCU "to" lines and command Bus are taken from the CRL Register on the Bus transfer.	30
CRL	Takes the MCU bus and stores it in the CRL register. It is used as BUSL CRL.	31
CWA	"Clear Write address" initiates a bus request for a memory and tells the memory that the data on the bus is an address, and that the data will be sent on the next transmission. Memory will hold the address in the Memory Address Register. The memory will be busy for all modules until the data is received via the DATA using a high bus request.	36
NIR	"Next Instruction Register." NEXT in the skip field brings the new instruction in NIR.	32
OPND	Operand Register. Takes information from MCU bus and puts it in OPND register (used as BUSH OPND).	34
RNWA	"Read No Write Address" initiates a bus request for a memory and tells the memory that the data on the bus is an address the contents of which are to be sent back to the OPND register, and not to restore the data, thus leaving all ones in the word read.	35
RWA	"Read Write address" is a bus request for memory. The original contents of the cell are restored, and the data is returned to OPND.	37
RWAN	"Read/Write Address and Next" is a bus request for memory. The original contents of the cell are restored, and the data is returned to both OPND and NIR.	20-27
RWN	"Read/Write Next" is a bus request for memory. The original contents of the cell are restored, and the data is returned to NIR.	33



ROM CYCLES







FIXED CORE

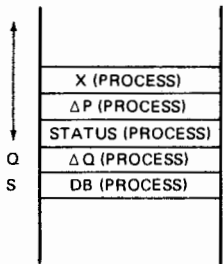
0	CSTP	Pointer to the "Code Segment Table".
1	P/DSTP	Cold Load program pointer/DSTP pointer to the "Data Segment Table"
2	PCBP	Pointer to the "Process Control Block"
3	DB	System (DB) global area
4	CPCB1	Pointer to the current PCB entry
5	QI1	Interrupt Control Stack
6	ZI1	
7	I/C1	
10	CPCB2	Interrupt Counter
11	QI2	Interrupt Control Stack
12	ZI2	
13	I/C2	
14		Pointer to the current PCB entry
15		Interrupt Counter
16		Interrupt Control Stack
17		Interrupt Counter
20-23		DRT entry for device #3
1774-1777		DRT for device #4, etc. Last possible DRT entry, for device #255

SIO	Program Pointer
PI	Interrupt Program Pointer
DBI	Interrupt Data Base
IRF	Interrupt Reference Flag

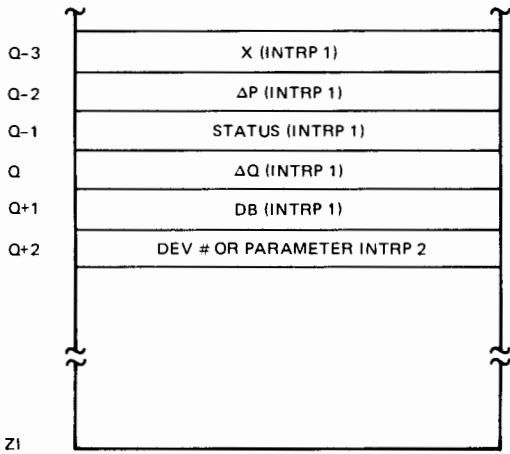
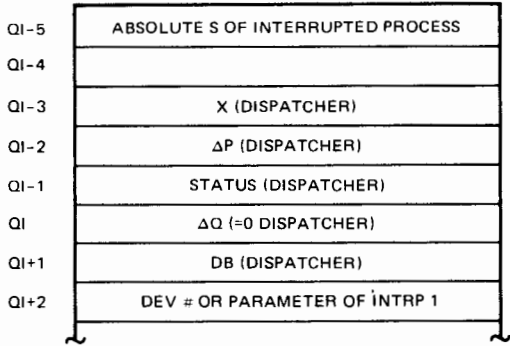
INTERRUPTS/TRAPS

Interrupt	Parameter	
0	EXTERNAL INTERRUPT	DEV #
1	POWER FAIL	-
2	POWER ON	-
3	STACK OVERFLOW	-
4	MODULE INTERRUPT	MODL #
5	CONSOLE INTERRUPT	CPU ID
6	COLD LOAD	-
7	-	-
}		
ON THE ICS @ Q+2		
10	-	-
11	MODULE ERROR	-
	Illegal Address	1000
	Bounds Violation	2000
	Non-responding Module	4000
12	PARITY ERROR	Type
	System Parity Error	40000
	Memory Address Parity Error	20000
	Data Parity Error	10000
13	MISCELLANEOUS	-
	Stack Underflow	1
	CST Bounds Violation	2
	STT Bounds Violation	3
14	CODE SEGMENT ABSENCE	-
	PCAL	P-label
	EXIT	N
15	STT ENTRY UNCALLABLE	P-label
16	TRACE (PCAL ONLY)	P-label
17	USER TRAPS	-
	-	0
	Integer Overflow	1
	Floating Point Overflow	2
	Floating Point Underflow	3
	Integer ÷ by 0	4
	Floating Point ÷ by 0	5
	SYSTEM (ALWAYS ENABLED)	-
	Privileged Instruction	6
	Unimplemented Instruction	7
}		
ON THE CURRENT STACK @ Q+1		

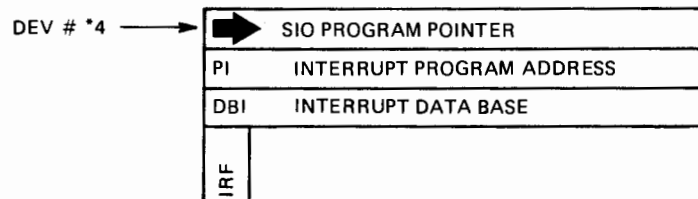
INTERRUPT CONTROL STACK



INTERRUPT CONTROL STACK



DEVICE REFERENCE TABLE



Up to 253 entries (DEV# 3 to 256)
Each entry has 4 words

- SIO Program pointer points to the absolute address of the next SIO order pair to be executed.
- PI Interrupt program address is the absolute address of this device's interrupt process program pointer.
- DBI Interrupt data base is the absolute address (DB) of this device's interrupt process data area (device information table).
- IRF Interrupt reference flag used by dispatcher

TABLE OF CONTENTS

I/O, MEMORY, AND POWER



CENTRAL DATA BUS

Modular Configuration	1
Signals	2

MCU-IOP-MEMORY CONFIGURATION

Configuration Dependency	3
Jumpers	4
S-Bus/IOP Interleavers	5
Memory Module Configuration	5, 6
PCA Card Configuration	7
4-32K Interleaved Memory Addressing	10
2-64K Interleaved Memory Addressing	12

I/O CONFIGURATION

Data and Interrupt Polling	13
Power Bus Diagram	14
Direct Only Bus Logic	15
HP 3000 System Backplane	17

MULTIPLEXER CHANNEL

Multiplexer Channel Block Diagram	18
I/O Orders	20
Device Reference Table	21
SIO Bus Logic	23
IOP-SIO Block Diagram	24
DRT Fetch Timing	25
Device Bound Transfer Timing	26
Memory Board Transfer Timing	27
Jump Timing	28
Multiplexer Channel Diagnostic Words	29

CONTROLLER PROCESSOR BLOCK DIAGRAM 31

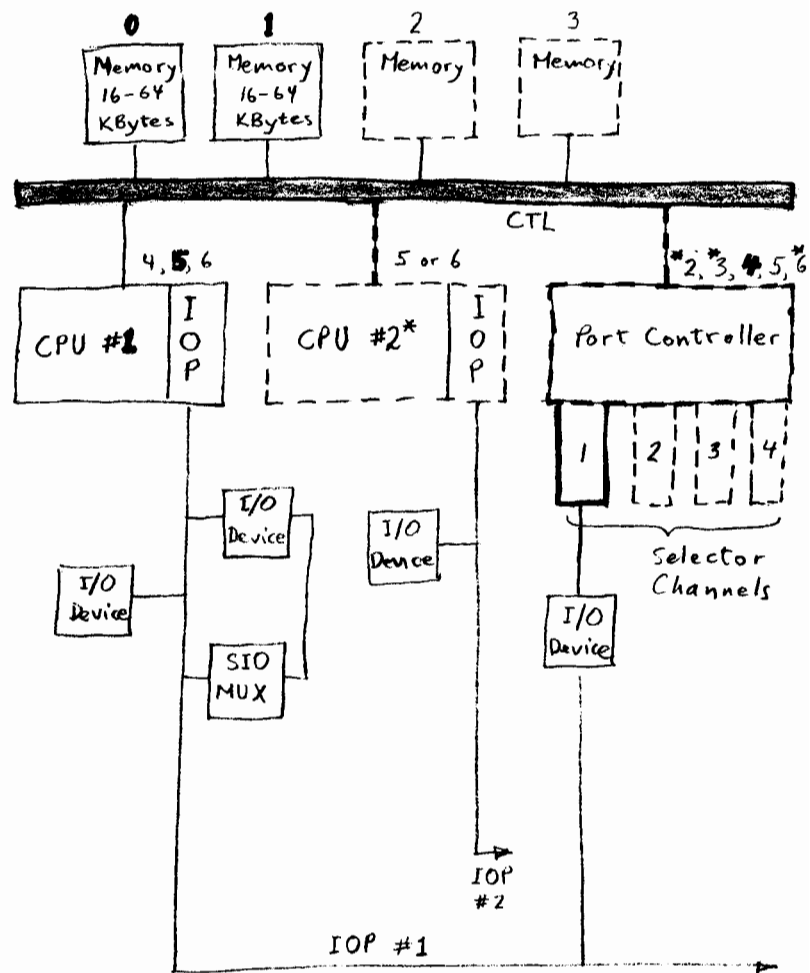
POWER DISTRIBUTION

30390-60002 Power Control Module	33
30330A Power Control Unit	34
Power Distribution Unit	35
DC Power Control Signals	36
30310A Power Supply Connectors	36

LIST OF EFFECTIVE PAGES

PAGES	DATE
1 thru 15	June 1973
17 thru 21	June 1973
23 thru 29	June 1973
31	June 1973
33 thru 36	June 1973

CENTRAL DATA BUS (CTL)



MODULAR CONFIGURATION

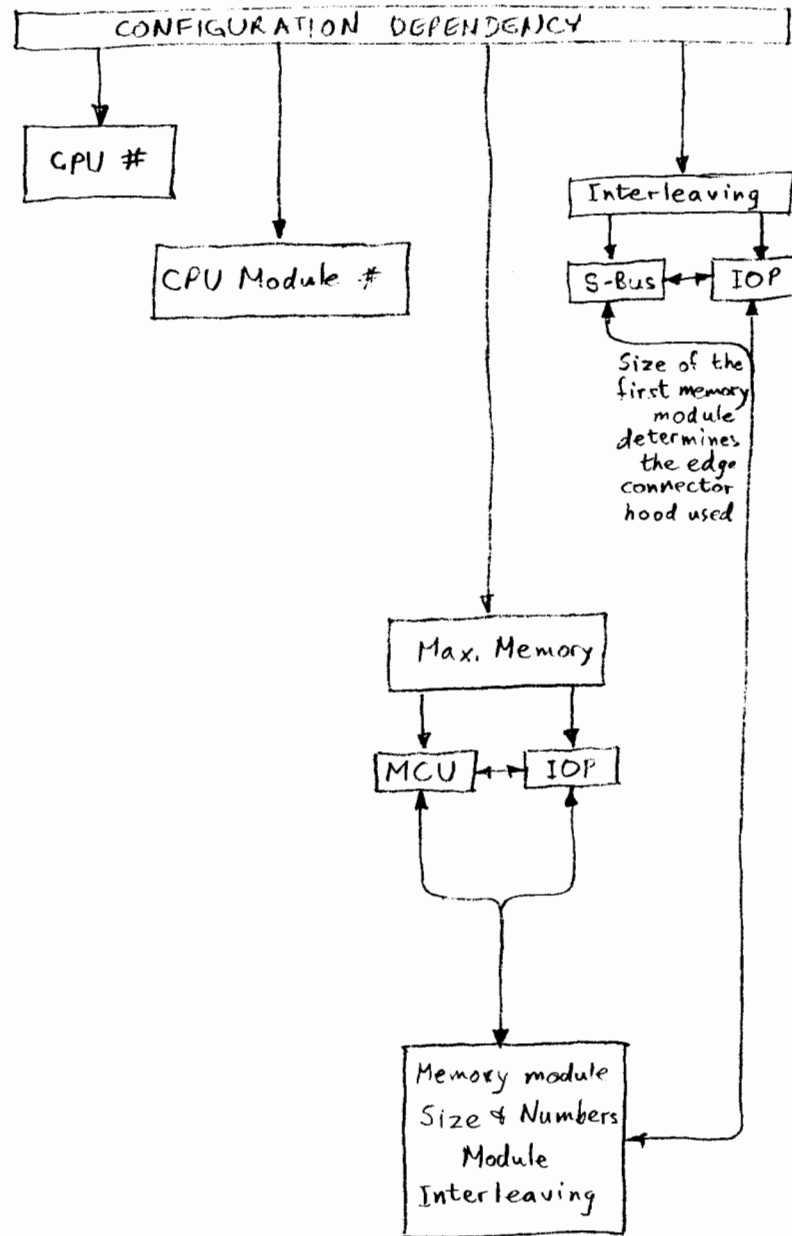
* For servicing only. Configurations shown are not necessarily those supported by Data Systems division.

JUNE 1973

CENTRAL DATA BUS (CTL)

CONNECTOR	SIGNAL	INTERFACE LOGIC
P2		
1	↓	
2	MCUD 0	} TRI-STATE
3	MCUD 1	
4	MCUD 2	
5	MCUD 3	
6	MCUD 4	
7	MCUD 5	
8	↓	
9	MCUD 6	} TRI-STATE
10	MCUD 7	
11	MCUD 8	
12	MCUD 9	
13	MCUD 10	
14		
15	MCUD 11	} TRI-STATE
16	MCUD 12	
17	MCUD 13	
18	MCUD 14	
19	MCUD 15	
20	MCUDPRTY	
21	↓	
22	TO 0	} TRI-STATE
23	TO 1	
24	TO 2	
25	FROM 0	
26	FROM 1	
27	FROM 2	
28	MOP 0	}
29	MOP 1	
30	↓	
31	READY 6	OPEN COLLECTOR
32	SYSPTY	TRI-STATE
33	READY 0	} OPEN COLLECTOR
34	READY 1	
35	READY 2	
36	READY 3	
37	READY 4	
38	READY 5	
39	↓	
40	ENABLE 0	
41	ENABLE 1	
42	ENABLE 2	
43	ENABLE 3	
44	ENABLE 4	
45	ENABLE 5	
46	↓	
47	MCUD PE	} OPEN COLLECTOR
48	SYS PE	
49	MCU RST	
50	↓	

MCU-IOP-MEMORY CONFIGURATION



MCU-IOP-MEMORY CONFIGURATION

CPU # 1 2** (MCU)

W1	I	0
W2	0	I

I = Jumper In
0 = Jumper Out

CPU Module # 4 5 6 (MCU)

FROM	W6	I	0	I
	W7	I	I	0
	W8	0	0	0
READY	W9	0	0	I
	W10	0	I	0
	W11	I	0	0
ENABLE	W12	I	0	0
	W13	0	I	0
PRIORITY	W14	0	I	I
	W15	0	0	I

Maximum Memory jumpers (MCU, IOP)
(Total Number of KBytes)

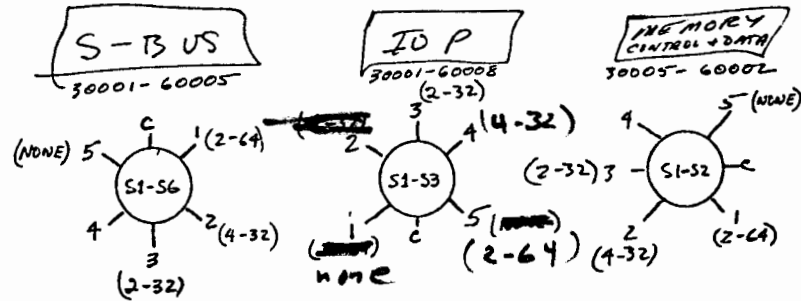
	16*	32*	48*	64*	80	96	112	128
IOP/MCU								
W1/W3	I	I	I	I	0	0	0	0
W2/W4	I	I	0	0	I	I	0	0
W3/W5	I	0	I	0	I	0	I	0

* For Service Personnel only
** Not yet available

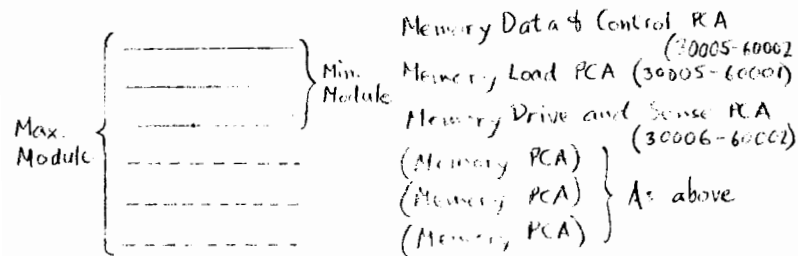
MCU-IOP-MEMORY CONFIGURATION

S-BUS/IOP INTERLEAVERS

INTERLEAVING	NONE	2-32K	4-32K	2-64K
IOP	1	3	4	5
ALL SWITCHES MUST BE IN THE SAME POSITION	5	3	2	1



MEMORY MODULE CONFIGURATION



MCU-IOP-MEMORY CONFIGURATION

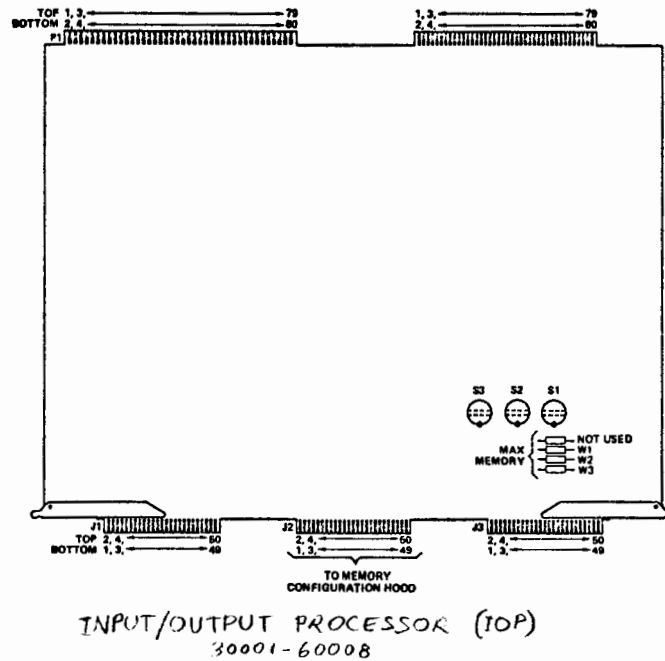
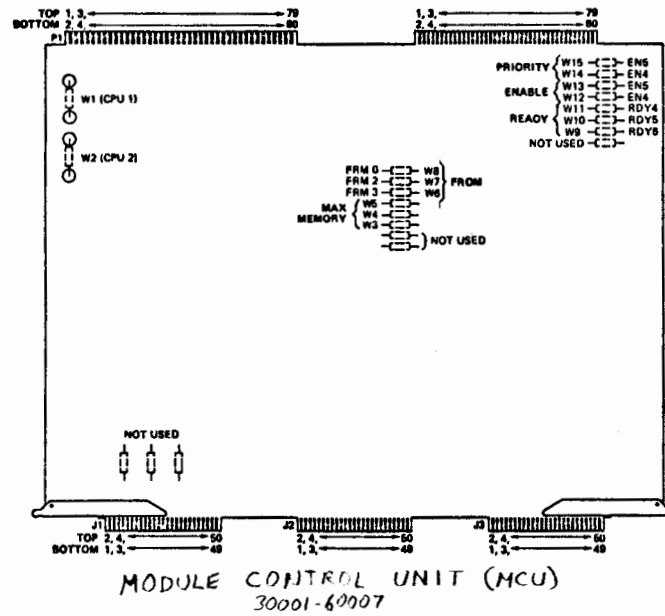
MEMORY DRIVE AND SENSE PCA (30006-60002)

CORE STACK #	JUMPER-IN POSITION
$\emptyset$	$\emptyset-1, \emptyset, 1$
1	2-3, 2, 3
2	4-5, 4, 5
3	6-7, 6, 7

MEMORY DATA AND CONTROL PCA (30005-60002)

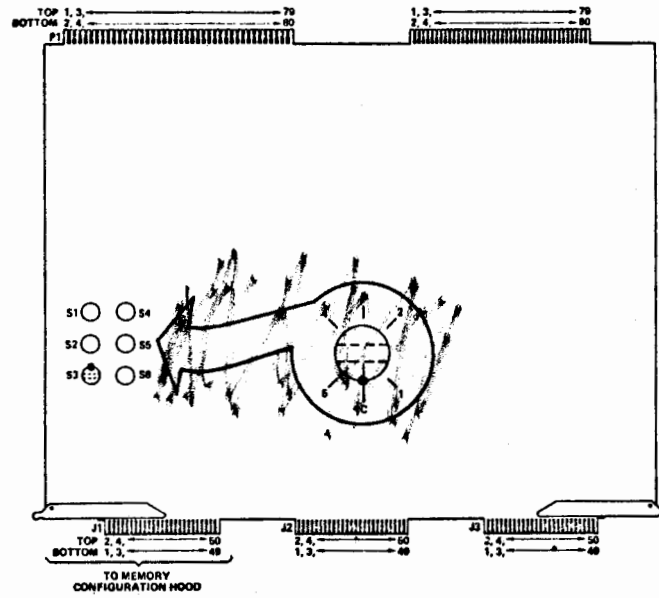
CTL MODULE #	$\emptyset$	1	2	3	MODULE SIZE
ENABLE $\emptyset-3$	C $\emptyset$	I	O	O	O
	C1	O	I	O	O
	C2	O	O	I	O
	C3	O	O	O	I
READY $\emptyset-3$	D $\emptyset$	I	O	O	O
	D1	O	I	O	O
	D2	O	O	I	O
	D3	O	O	O	I
TO-FROM $\emptyset-1$	E $\emptyset$	I	O	I	O
	E1	I	I	O	O
ENABLE $\emptyset-2$	F $\emptyset$	O	I	I	I
	F1	O	O	I	I
	F2	O	O	O	I

PCA CARD CONFIGURATION

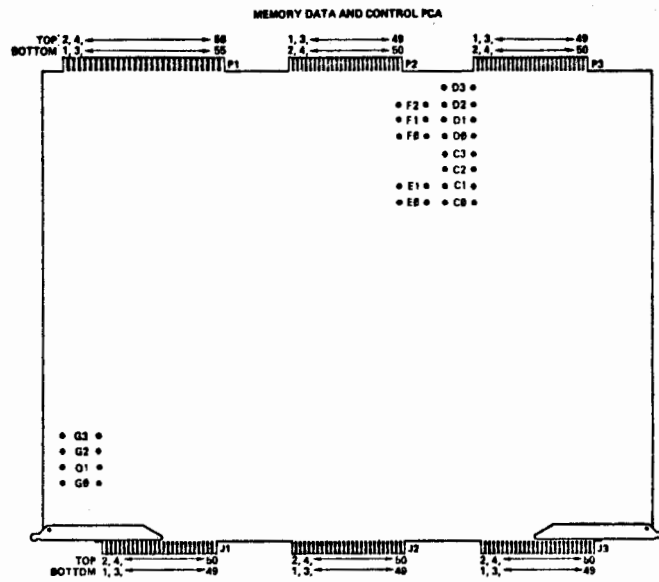


JUNE 1973

PCA CARD CONFIGURATION

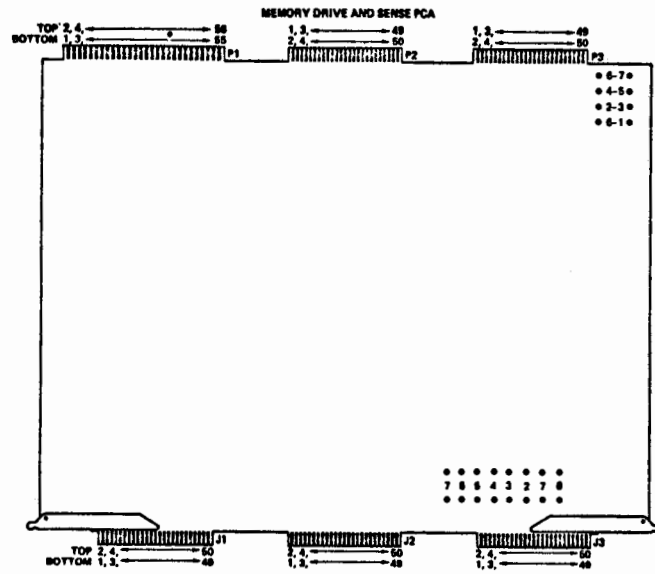


S-BUS PCA (30001-60005)



(30006-60002)

PCA CARD CONFIGURATION



JUNE 1973

4-32K INTERLEAVED MEMORY ADDRESSING

MEMORY MODULES

CTL
Module 0

40000
100000
140000
4
40004
100004
140004
10
...
...
...
37774
77774
137774
177774

STACK MODULES

0 { 40000 - 17774
100000 - 57774
140000 - 157774 } Every 4th Address

1 { 20000 - 37774
60000 - 77774
120000 - 137774
160000 - 177774 } Every 4th Address

CTL
Module 1

40001
100001
140001
5
40005
100005
140005
11
...
...
...
37775
77775
137775
177775

0 { 40001 - 17775
100001 - 57775
140001 - 157775 } Every 4th Address

1 { 20001 - 37775
60001 - 77775
120001 - 137775
160001 - 177775 } Every 4th Address

4-32K INTERLEAVED MEMORY ADDRESSING

MEMORY MODULES

CTL
Module 2

40002	2
100002	
140002	
6	
40006	
100006	
140006	
12	
...	
...	
...	
37776	
77776	
137776	
177776	

STACK MODULES

0 { 2 - 17776
40002 - 57776
100002 - 117776
140002 - 157776 } Every 4th
Address

1 { 20002 - 37776
60002 - 77776
120002 - 137776
160002 - 177776 } Every 4th
Address

CTL
Module 3

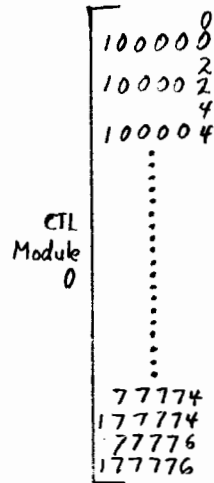
40003	3
100003	
140003	
7	
40007	
100007	
140007	
13	
...	
...	
...	
37777	
77777	
137777	
177777	

0 { 3 - 17777
40003 - 57777
100003 - 117777
140003 - 157777 } Every 4th
Address

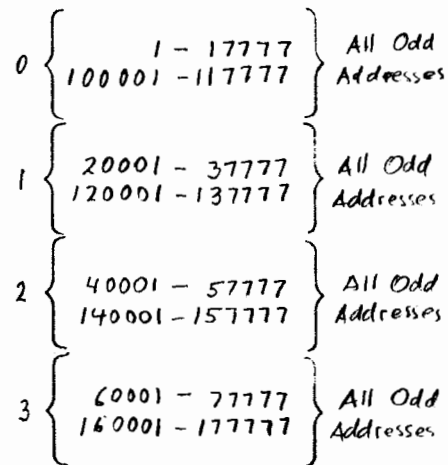
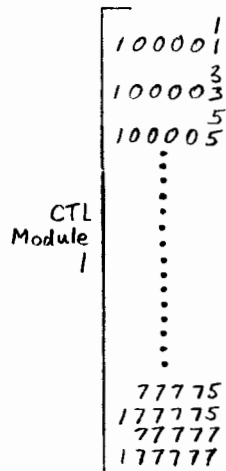
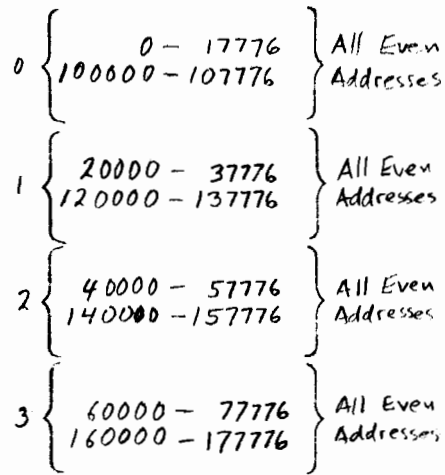
1 { 20003 - 37777
60003 - 77777
120003 - 137777
160003 - 177777 } Every 4th
Address

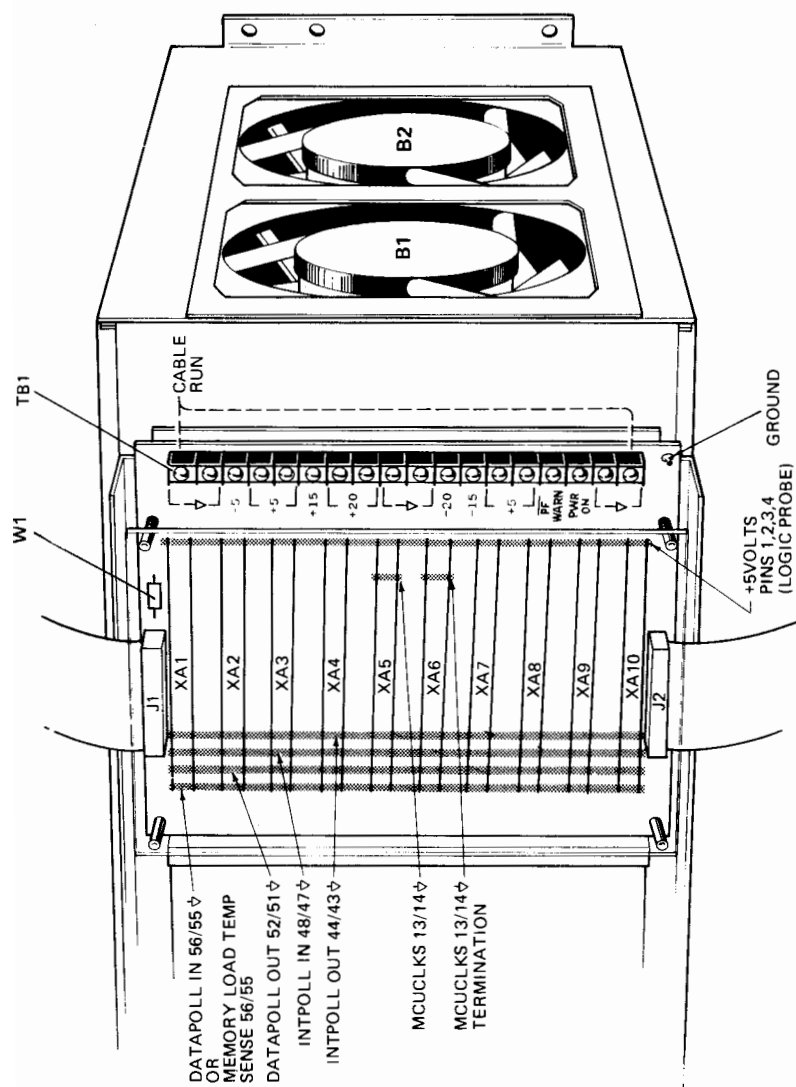
2-64K INTERLEAVED MEMORY ADDRESSING

MEMORY MODULES



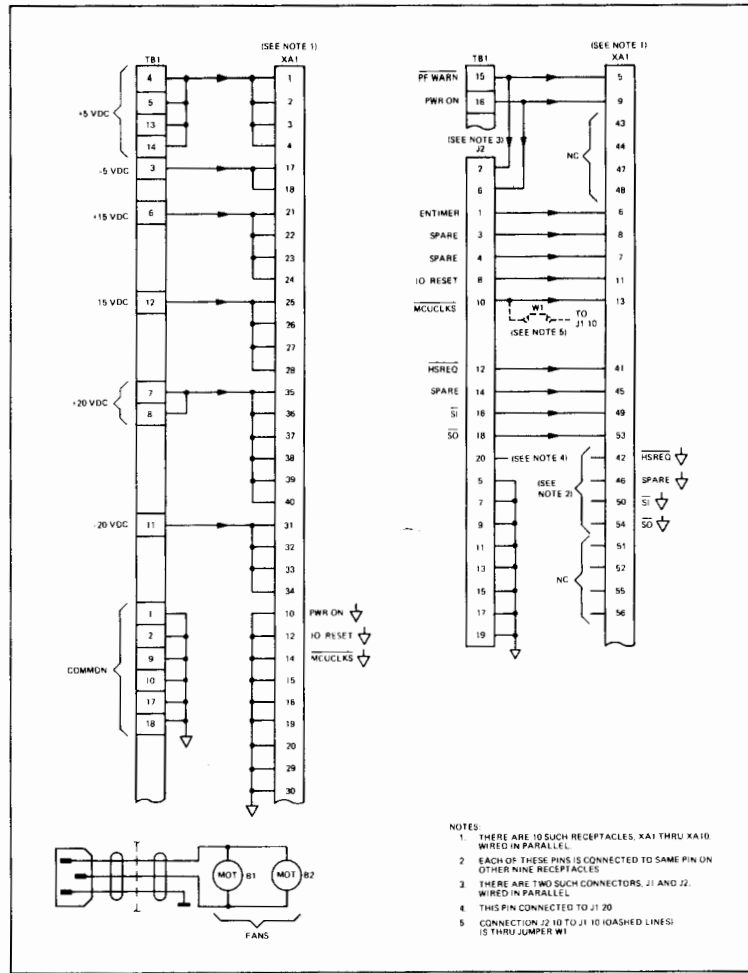
STACK MODULES



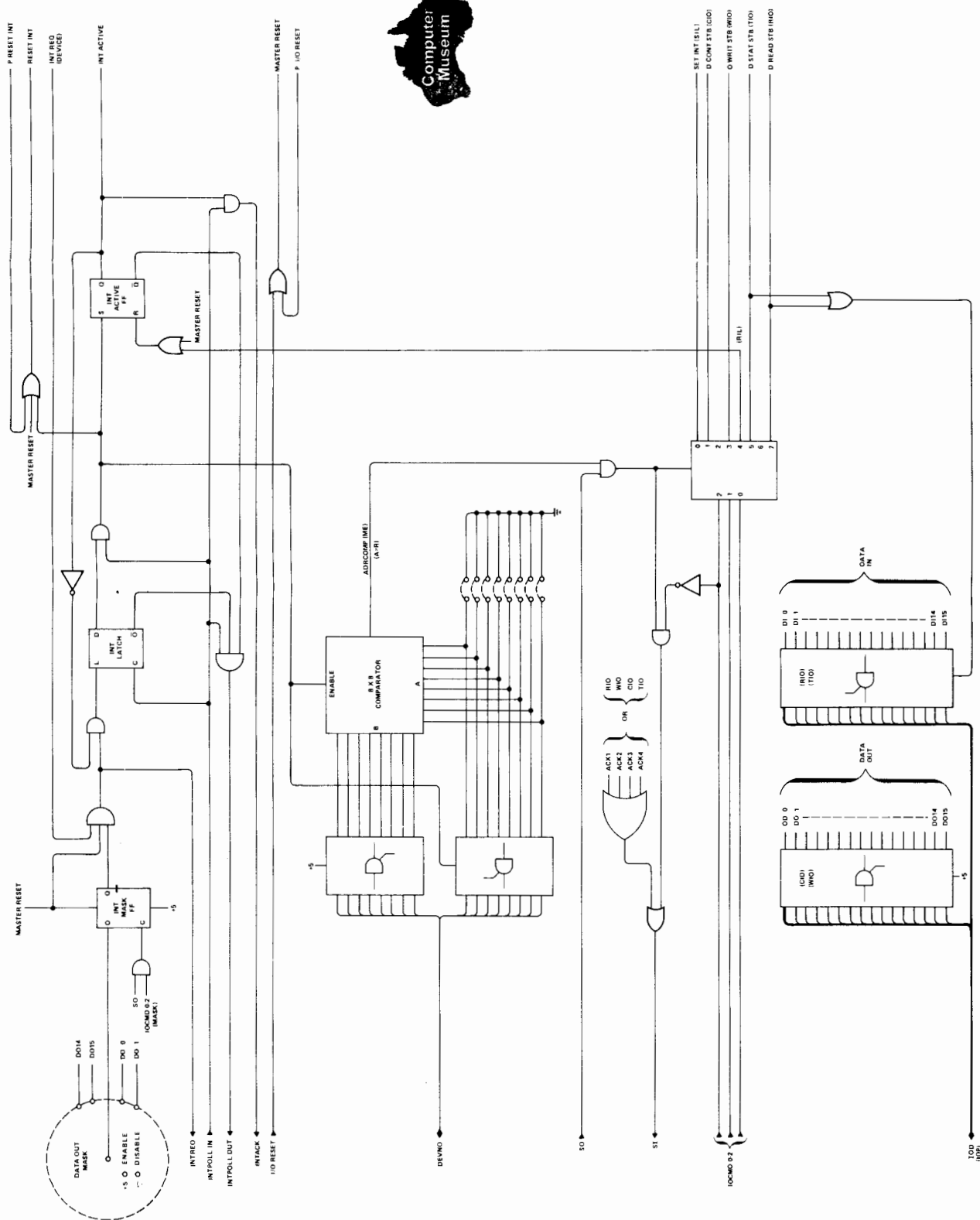


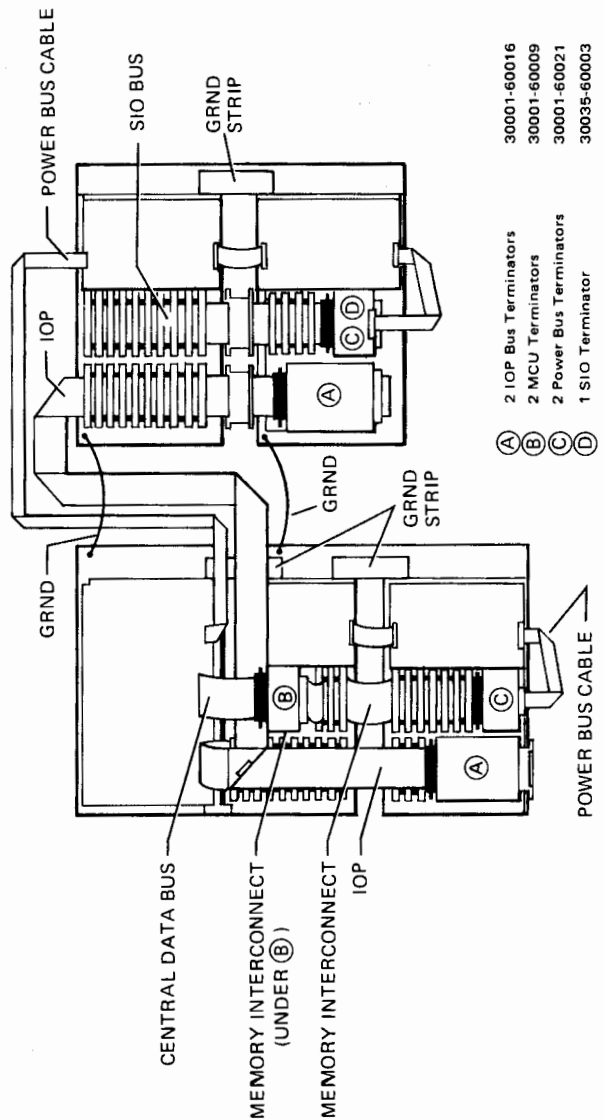
30002A UNIVERSAL PCA MODULE

POWER BUS DIAGRAM

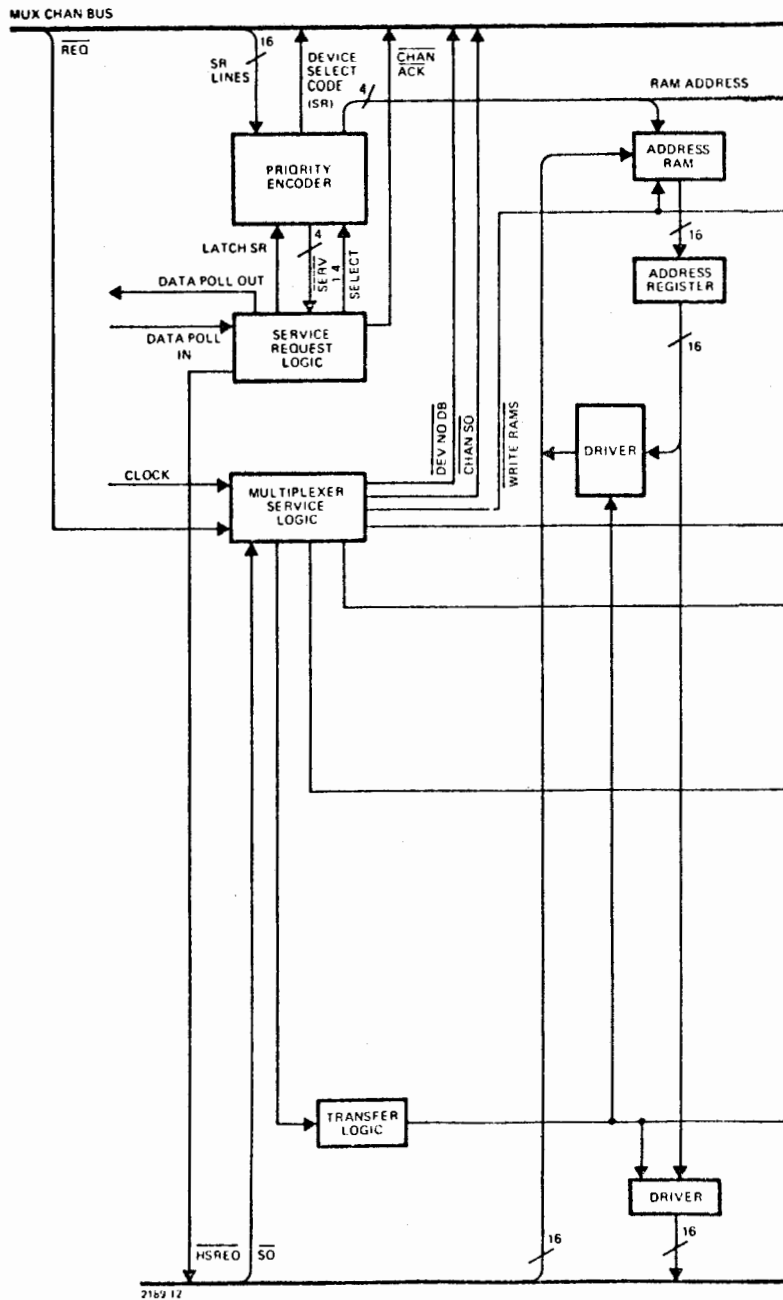


30002A UNIVERSAL PCA MODULE

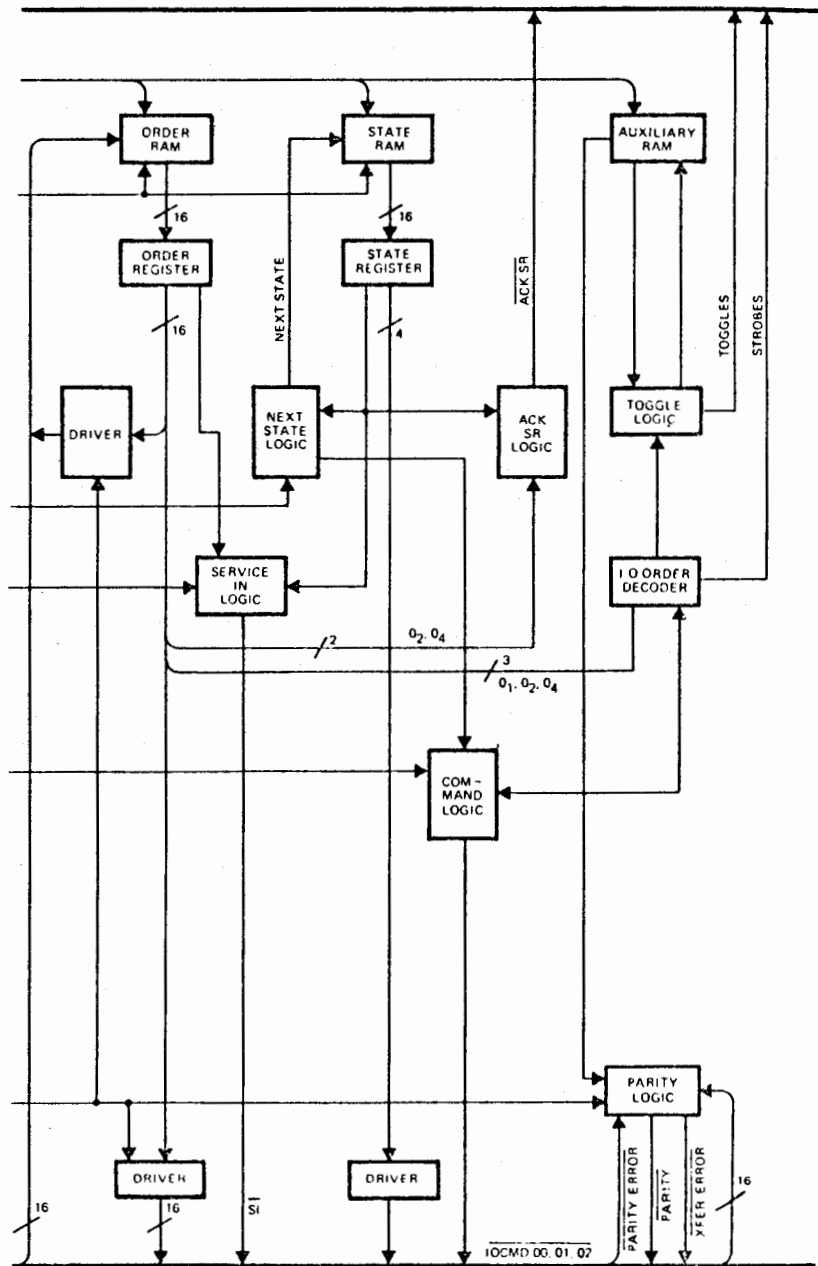




MULTIPLEXER CHANNEL BLOCK DIAGRAM



MULTIPLEXER CHANNEL BLOCK DIAGRAM



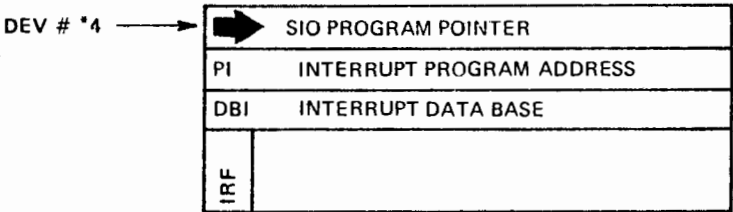
JUNE 1973

I/O ORDERS

"SIO Commands"				15
0	1 - 3	4		
IOCW	DC	ORDER	#OF WORDS (2's COMP) */CONTROL INFO	
IOAW			DATA ADDRESS/CONTROL INFO/SENSE STORAGE	
ORDER	0	1 - 3	4 5	15
000		JUMP	C	
			JUMP TARGET ADDRESS	
			IOCW (4) = 1 → CONDITIONAL	
	0	1 - 3	4	15
001		RETURN		
		RESIDUE		
			RESIDUE OF WORD COUNT	
	0	1 - 3	4	15
010		INTERRUPT		
	0	1 - 3	4 5	15
011		END	INT	
			STATUS (will be returned)	
			IOCW (4) = 1 → INTERRUPT	
	0	1 - 3	4	15
100		CONTROL	CONTROL WORD #1	12 BITS
			CONTROL WORD #2	
	0	1 - 3	4	15
101		SENSE		
			STATUS (will be returned)	
	0	1 - 3	4	15
110	DC	WRITE	#OF WORDS (2's COMP)*	
			ABSOLUTE ADDRESS	
			IOCW (0) = 1 → DATA CHAIN	
	0	1 - 3	4	15
111	DC	READ	#OF WORDS (2's COMP)*	
			ABSOLUTE ADDRESS	

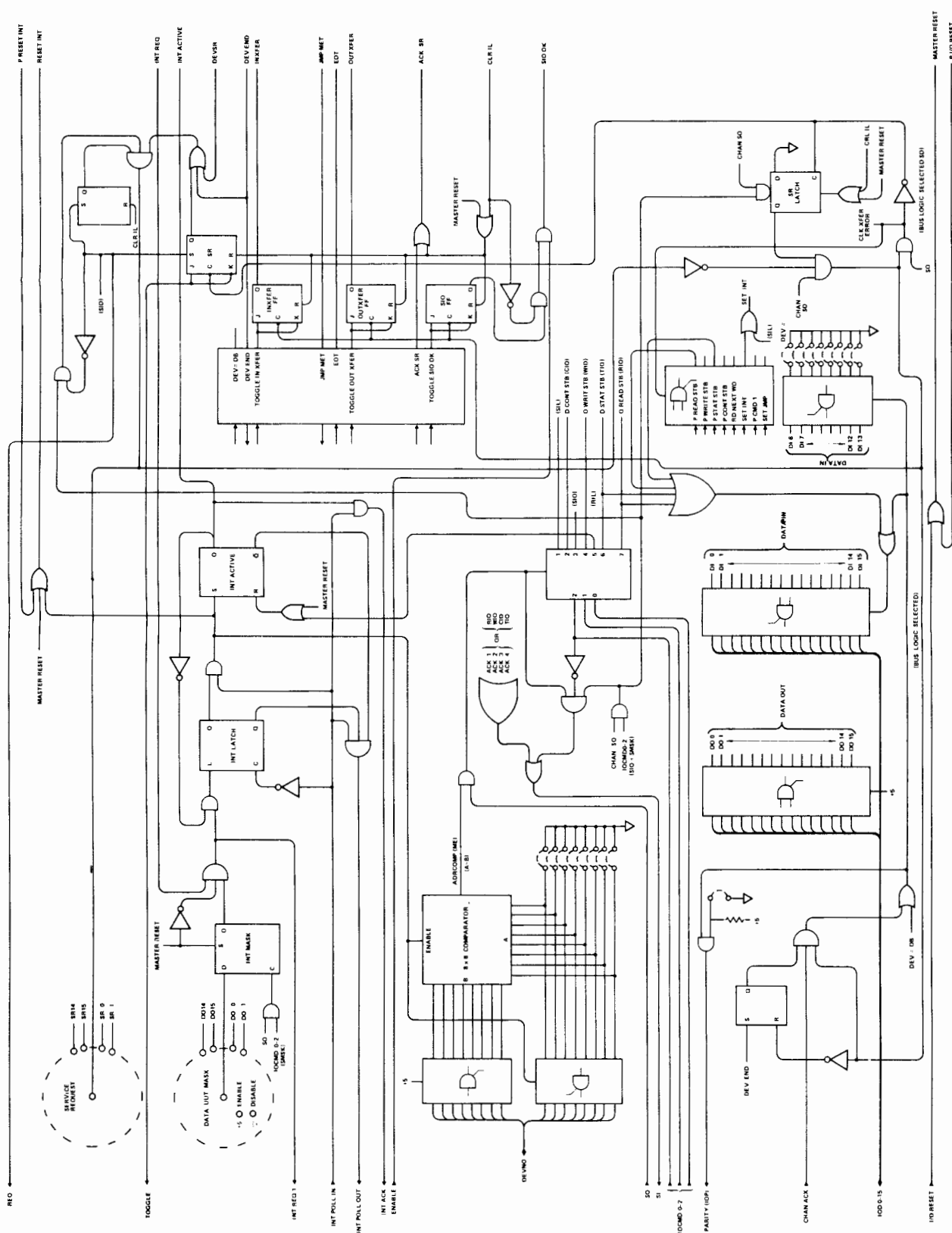
*#OF WORDS for I/O order pair 1-4096
DC=DATA CHAINING
□ - UNUSED

DEVICE REFERENCE TABLE

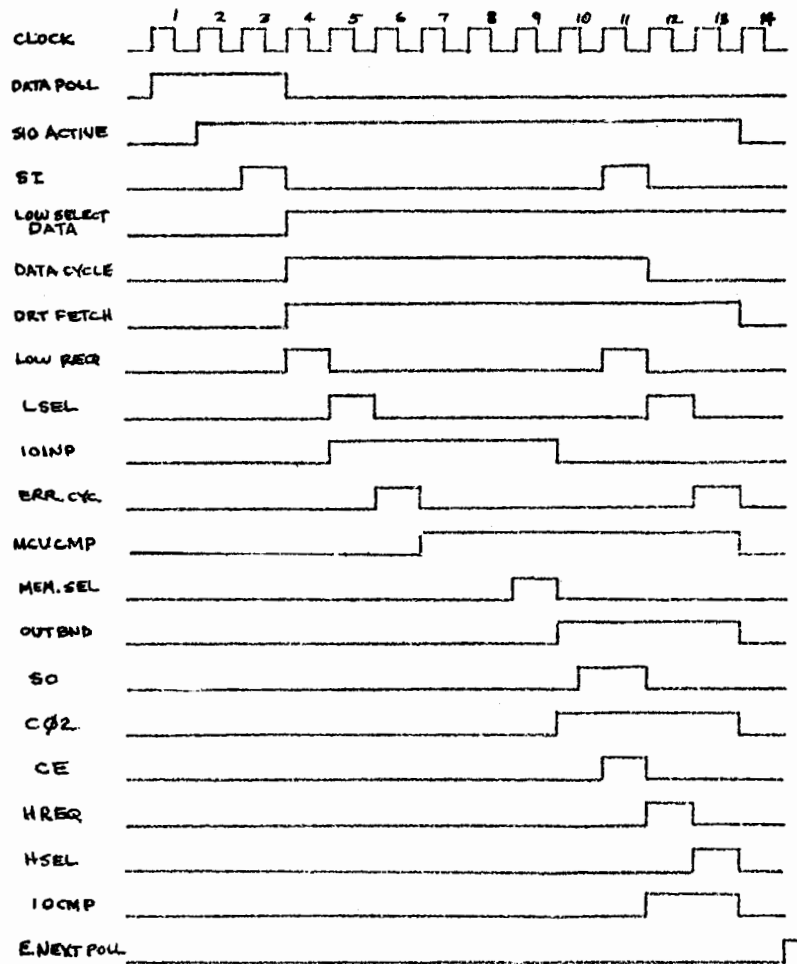


Up to 253 entires (DEV# 3 to 256)
Each entry has 4 words

- SIO Program pointer points to the absolute address of the next SIO order pair to be executed.
- PI Interrupt program address is the absolute address of this device's interrupt process program pointer.
- DBI Interrupt data base is the absolute address (DB) of this device's interrupt process data area (device information table).
- IRF Interrupt reference flag used by dispatcher

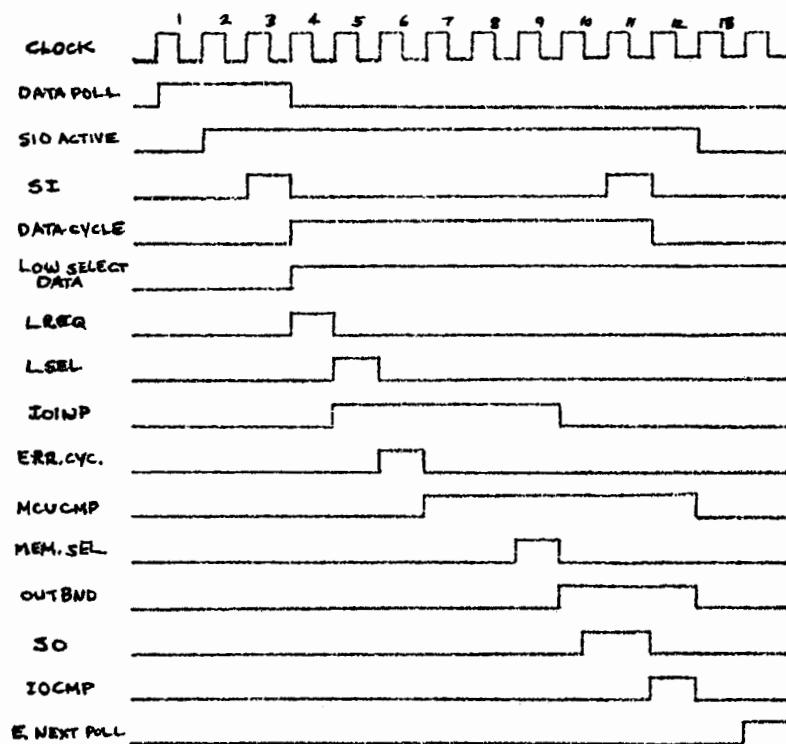


DRT FETCH TIMING

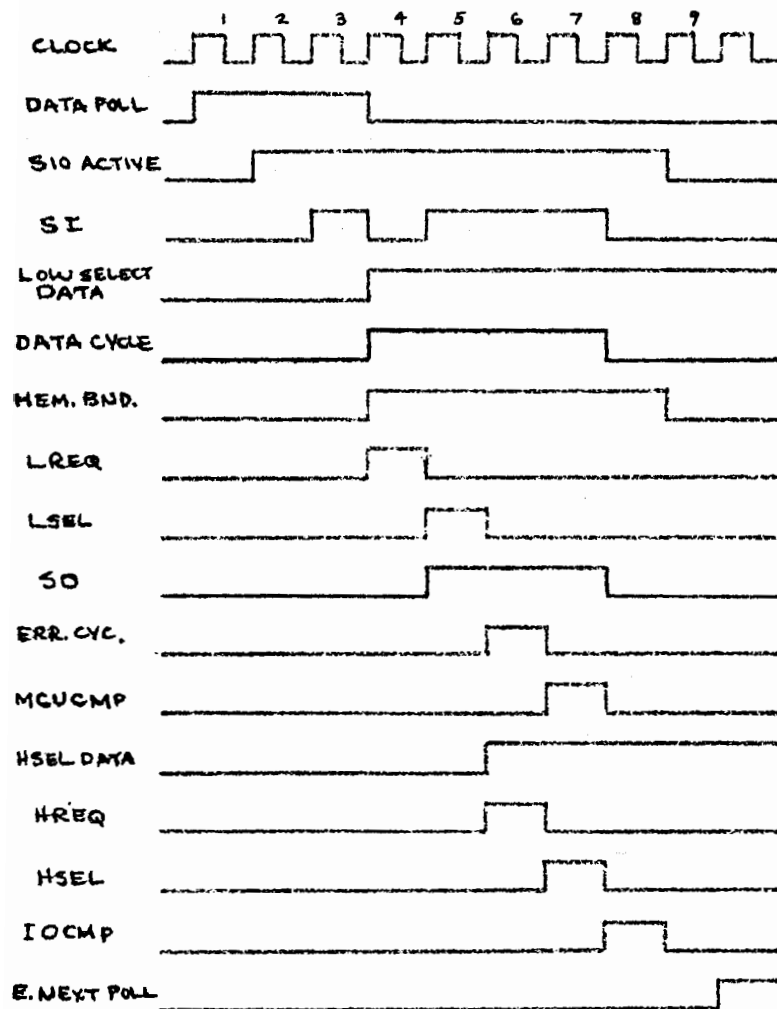


JUNE 1973

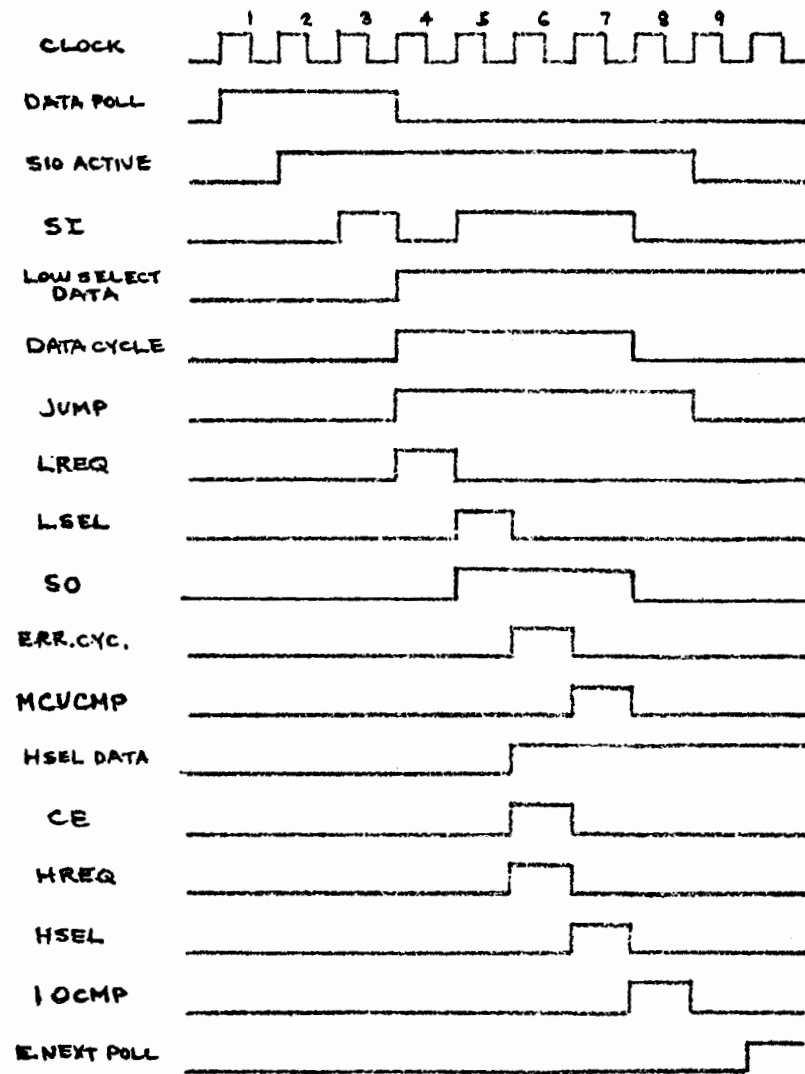
DEVICE BOUND TRANSFER TIMING



MEMORY BOUND TRANSFER TIMING



JUMP TIMING



MULTIPLEXER CHANNEL DIAGNOSTIC WORDS

MUX CHAN STARTING STATE WORD (WIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

STATE A
STATE B
STATE C
STATE D

PREVIOUS CONTROL WORD
MUST SELECT STATE RAM

MUX CHAN DIAGNOSTIC CONTROL WORD (CIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

I/O RESET

RAM
ADDRESS
0-17₈

ADDRESS

ORDER

STATE

LOAD REGISTERS
FROM RAMS

INCREMENT
(ADDRESS OR WORD
COUNT)

SELECTS

MUX CHAN STATUS WORD (TIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

STO READY

RIO-WIO
READY

STATE RAM
PARITY ERROR

LAST RAM
ADDRESS
SERVICED

MUX CHAN DIAGNOSTIC STATE WORD (RIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

PREVIOUS CONTROL WORD
MUST SELECT STATE RAM

STATE A

STATE B

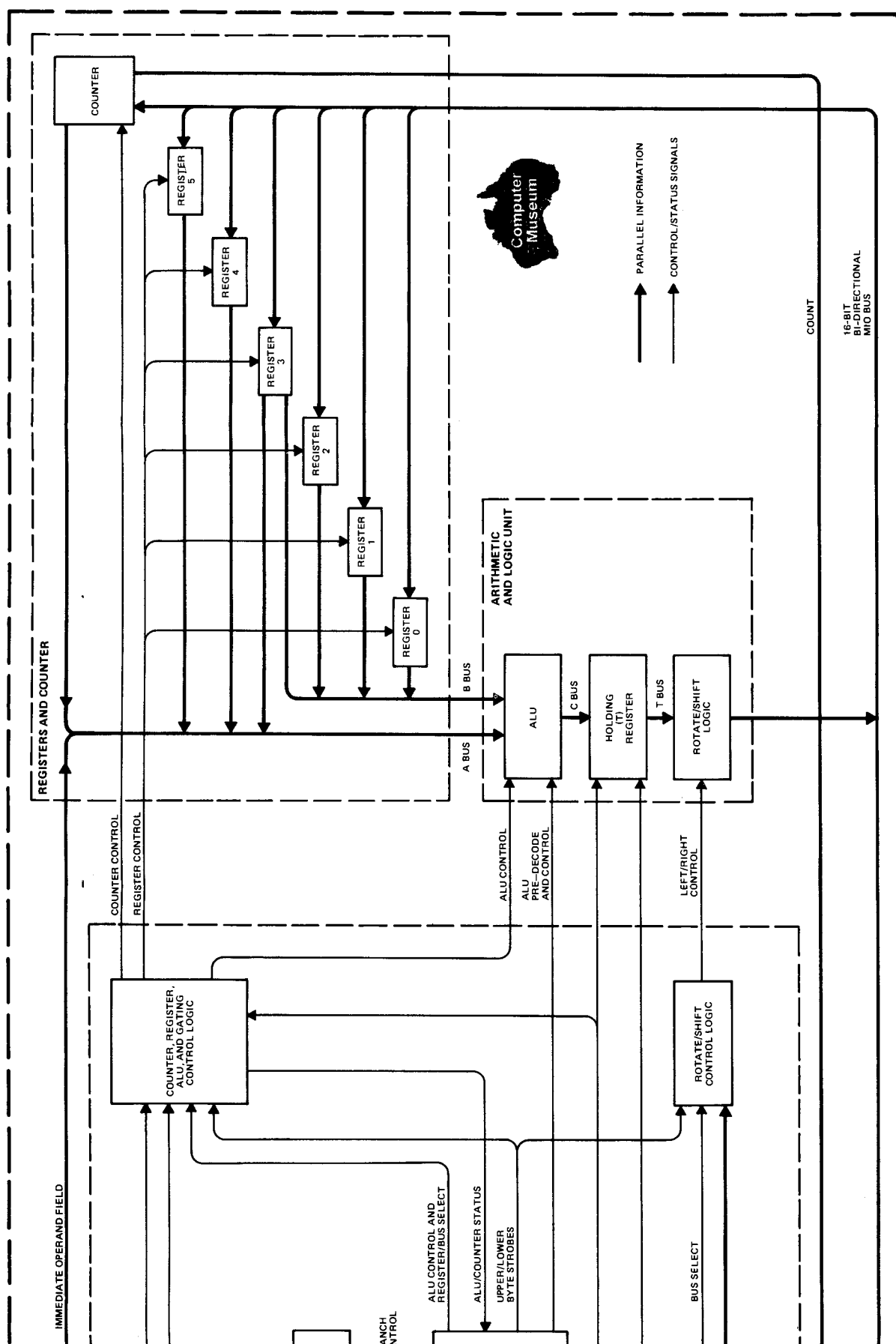
STATE C

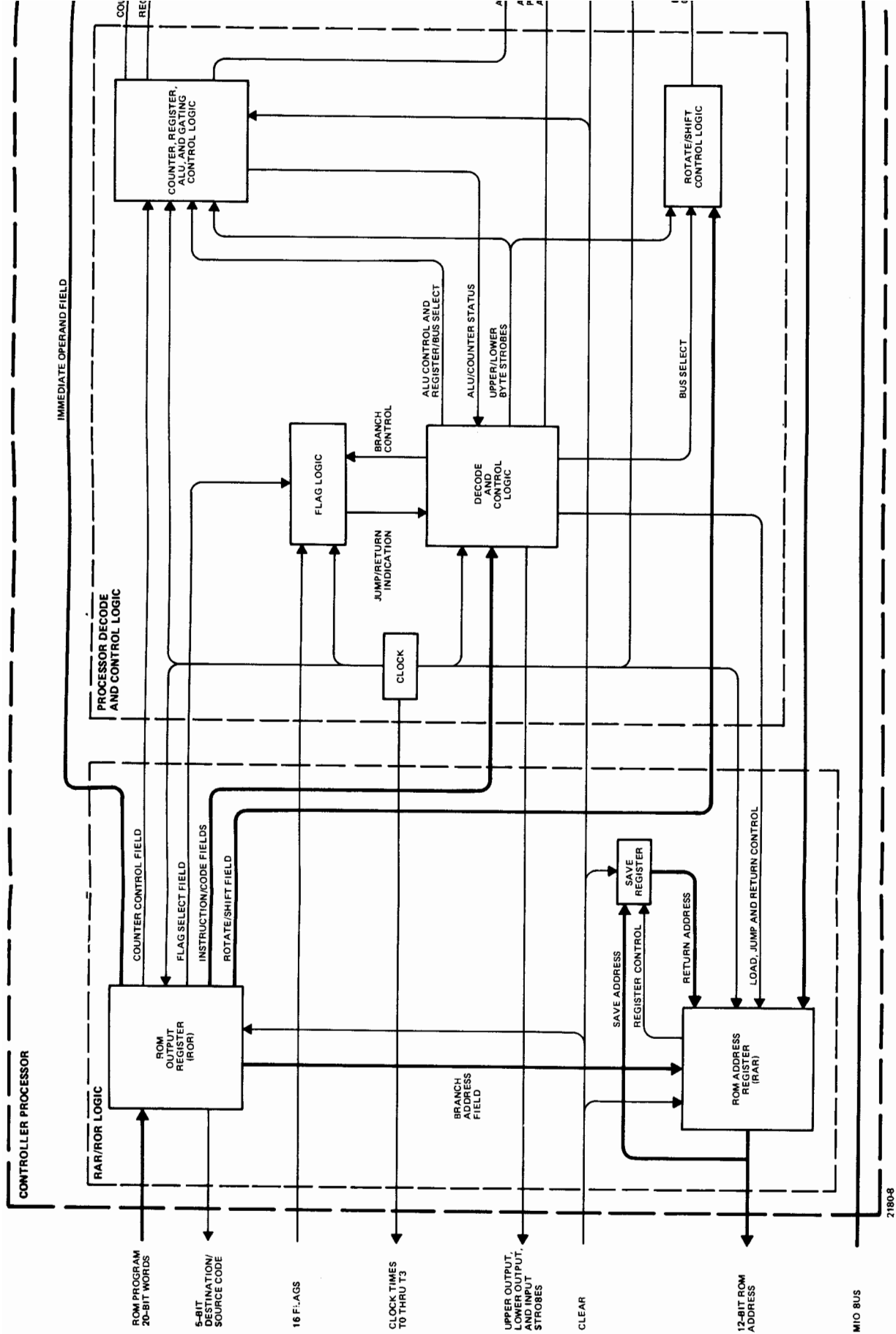
STATE D

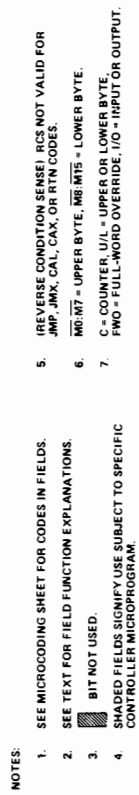
EDT

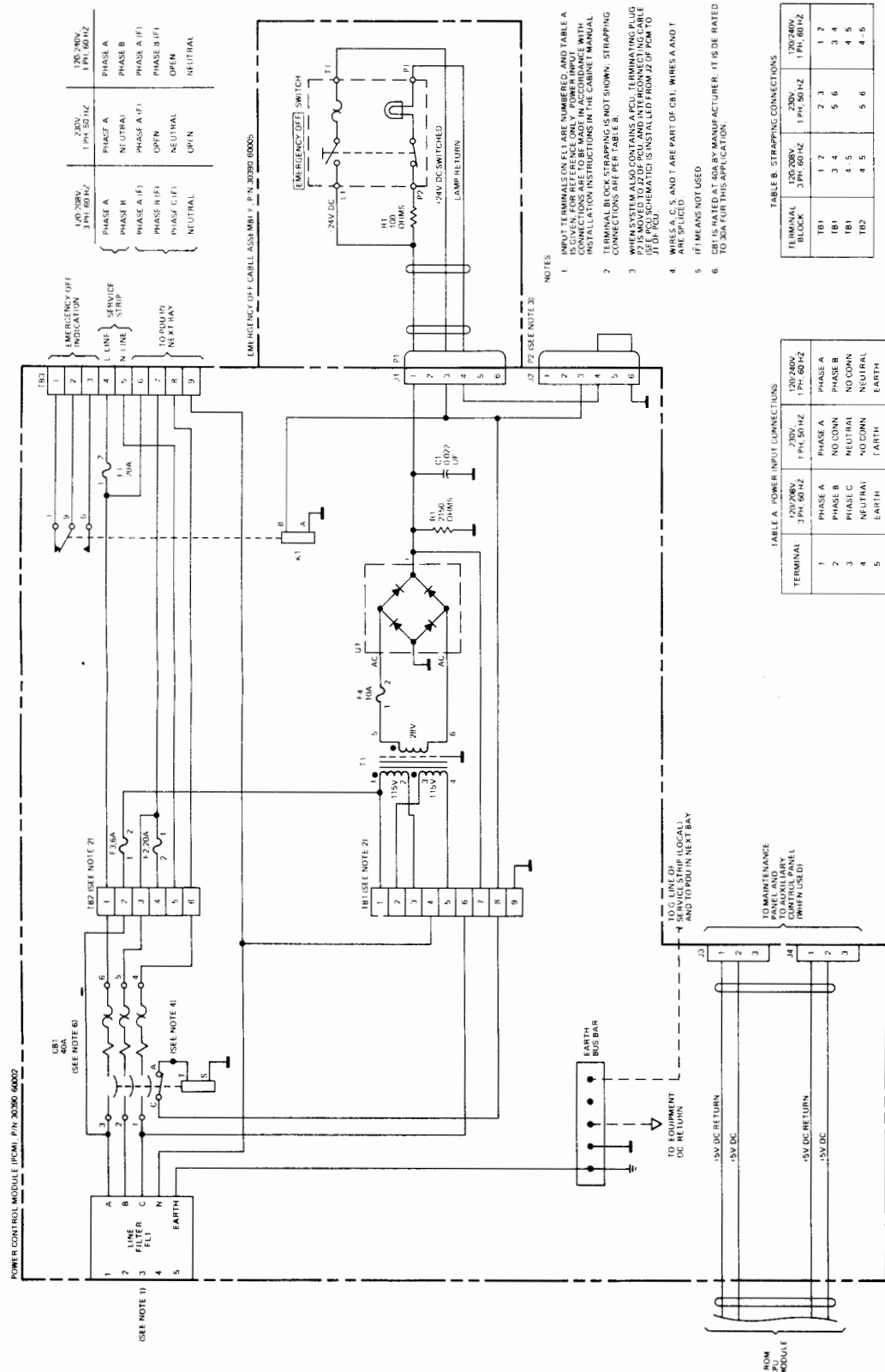
ADDRESS PARITY

STATE PARITY

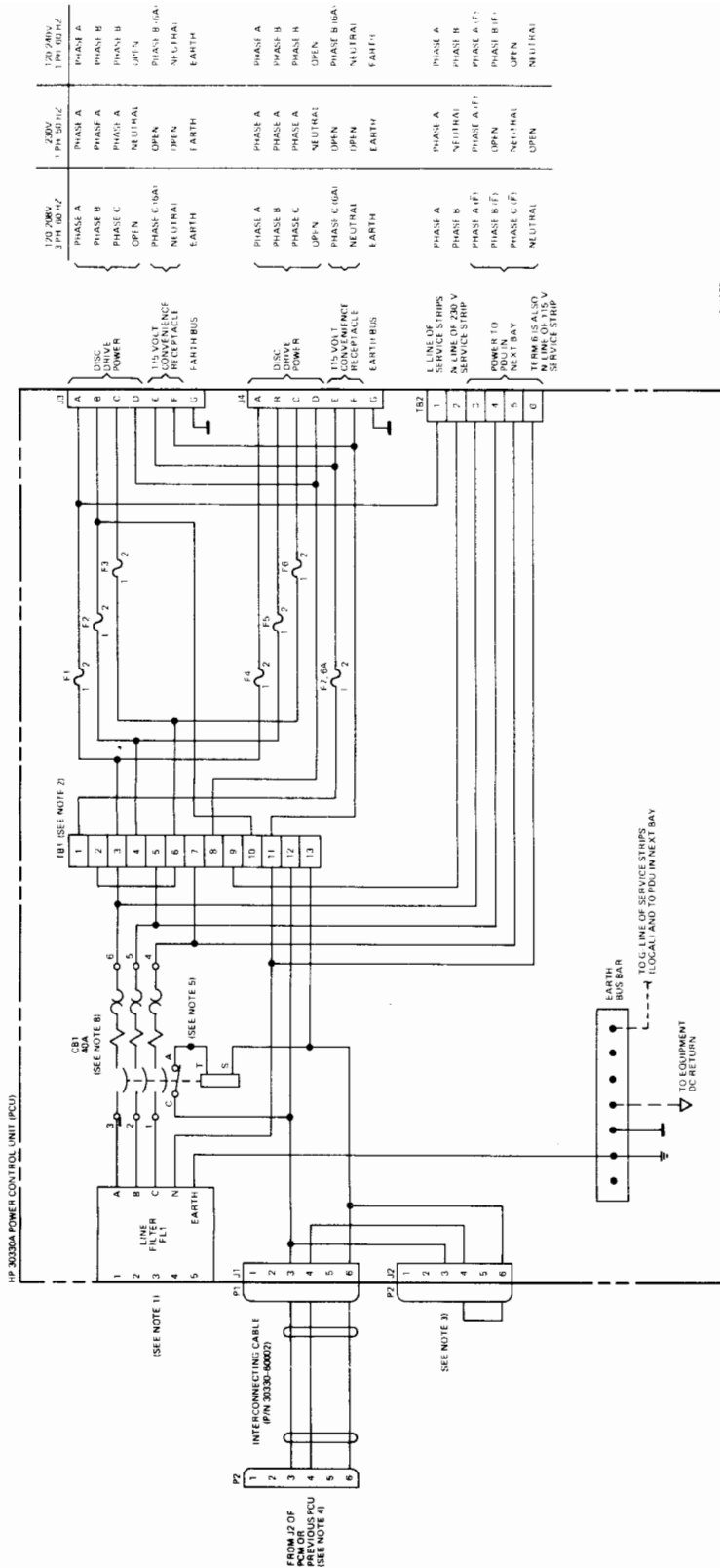








30390-60002 POWER CONTROL MODULE
30330A POWER CONTROL UNIT



NOTES:

1. INPUT TERMINALS ON P1 ARE NUMBERED AND Labeled AS GIVEN FOR REFERENCE ONLY. POWER INPUT TERMINALS ARE NOT NUMBERED FOR REFERENCE WITH INSTRUCTIONS IN THE CABINET MANUAL.
2. TERMINAL BLOCK STRAPPING IS NOT SHOWN. STRAPPING CONNECTIONS ARE PER TABLE B.
3. TERMINATING PLUG P2 OBTAINED FROM J2 OF PCM OR NEIGHBORING PCU.
4. P2 OF P41 RECONNECTING CABINET WATERS WITH J2 OF PCM OR NEIGHBORING PCU.
5. WIRES A, C, S, AND T ARE PART OF FBI WHEN A AND T ARE NOT USED.
6. FUSES F1 THROUGH F6 ARE RATED 20A.
7. F1 MEANS NOT FUSED.
8. FBI IS RATED 60A BY MANUFACTURER. IT IS IN RATED TO 30A FOR THIS APPLICATION.
9. NOT USED WHEN PCU IS BEING USED TO POWER DISC DRIVES.

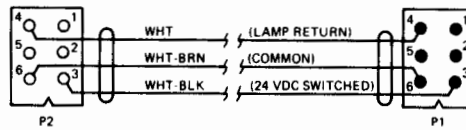
TABLE A. POWER INPUT CONNECTIONS

TERMINAL	120/208V, 3 PH, 60 HZ	220V, 1 PH, 50 HZ	120/240V, 1 PH, 60 HZ
1	PHASE A	PHASE A	PHASE A
2	PHASE B	NO CONN.	PHASE B
3	PHASE C	NO CONN.	NO CONN.
4	NEUTRAL	NO CONN.	NEUTRAL
5	EARTH	EARTH	EARTH

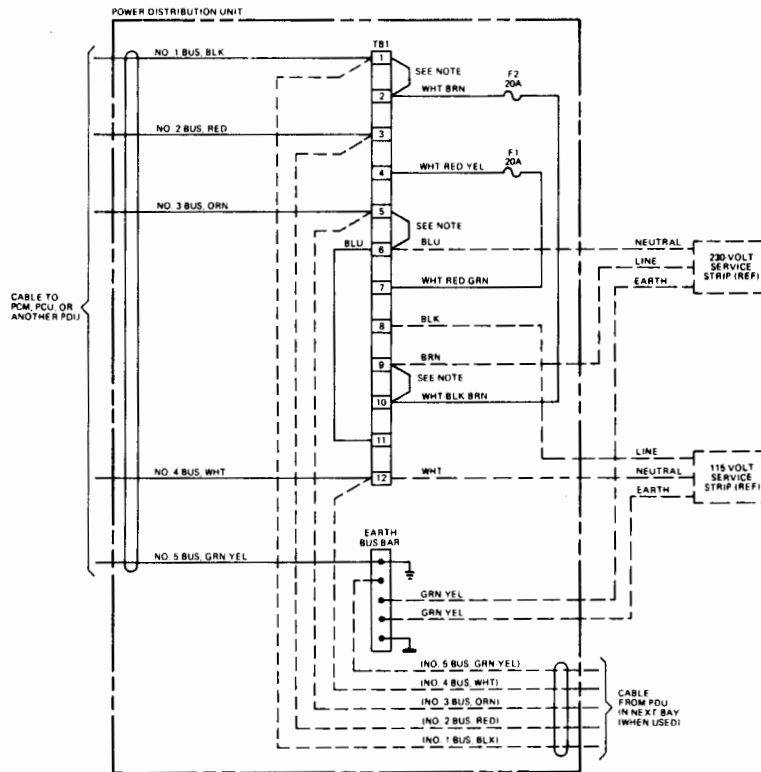
TABLE B. STRAPPING CONNECTIONS

TERMINAL BLOCK	120/208V, 3 PH, 60 HZ	220V, 1 PH, 50 HZ	120/240V, 1 PH, 60 HZ
TB1	1 2	2 3	1 2
TB2	4 5	3 4	4 5
TB3	6 7	7 8	6 7
TB4	9 10	8 9	9 10

POWER DISTRIBUTION UNIT



PCU-to-PCU/PCM Interconnecting Cable



NOTE
THESE THREE LINES REPRESENT OPTION 015
STRAPPING CONNECTIONS

DC POWER CONTROL SIGNALS

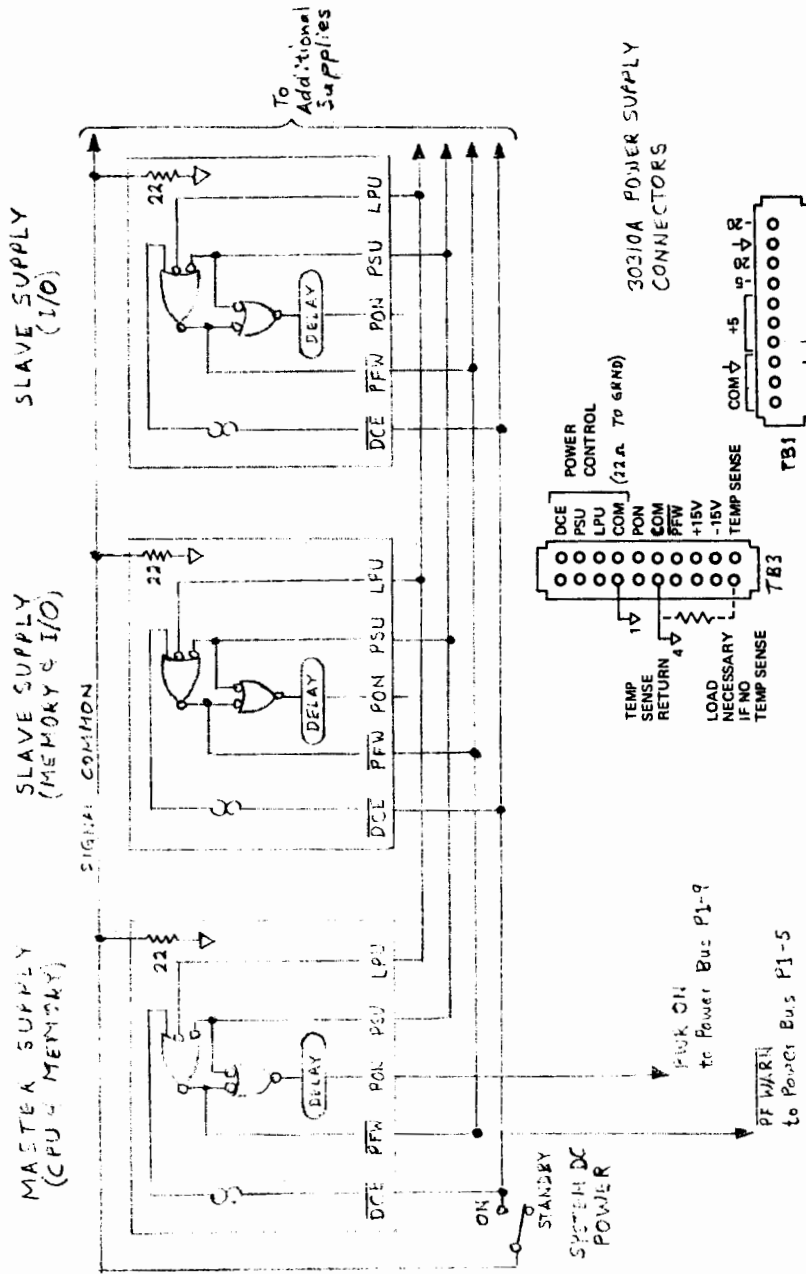


TABLE OF CONTENTS

I/O SUBSYSTEMS AND DEVICES

ASYNCHRONOUS TERMINAL CONTROLLER

Terminal Data Status Word	30032-2
Terminal Data Control Word	30032-2
Terminal Data Output Data Word	30032-3
Terminal Data Input Data Word	30032-3
Terminal Data Parameter Word	30032-4
Terminal Control #1 Status Word	30032-5
Terminal Control #1 Control Word	30032-5
Signal Descriptions	30032-6

DISC FILE

Disc File Status Word	30102-2
Disc File Control Word (IOCW)	30102-4
Disc File Control Word (IOAW)	30102-4
Cabling Arrangements	30102-6
PCA Position and Connection Details	30102-7
AC/DC Cable Diagrams	30102-8
Disc Pack Recording Format	30102-9
PWA Assignments in Electronic Gate	30102-10
Arrangement of Read/Write Heads	30102-10

FIXED HEAD DISC

Fixed Head Disc Status Word	30103-2
Fixed Head Disc Control Word	30103-2
Fixed Head Disc Error Status Word 1	30103-3
Fixed Head Disc Error Status Word 2	30103-3
Fixed Head Disc Control Word (IOCW)	30103-3
Fixed Head Disc Control Word (IOAW)	30103-3

CARD READER

Card Reader Status Word	30106-2
Card Reader Control Word	30106-3
Interface Cable Connector Pin Index	30106-4

LINE PRINTER

Line Printer Interrupt Status Word	30108-2
Line Printer Device Status Word	30108-2
Line Printer Control Word	30108-3
Line Printer Byte Mode Data Word	30108-4
Line Printer Word Mode Data Word	30108-4
Line Printer Format Word	30108-4

TABLE OF CONTENTS (Continued)

CARTRIDGE DISC

Cartridge Disc Status Word	30110-2
Cartridge Disc Control Word (IOCW)	30110-3
Cartridge Disc Control Word (IOAW)	30110-3
Cartridge Disc Control Word (CC)	30110-4
Sector Format	30110-5

MAGNETIC TAPE

Mag Tape Status Word	30115-2
Mag Tape Control Word	30115-3

CONSOLE/TIMER

Console/Timer Status Word	30124-2
Console/Timer Control Word	30124-3

LIST OF EFFECTIVE PAGES

PAGES	DATE
30032-1 thru 30032-7	June 1973
30102-1 thru 30102-10	June 1973
30103-1 thru 30103-3	June 1973
30106-1 thru 30106-4	June 1973
30108-1 thru 30108-4	June 1973
30110-1 thru 30110-5	June 1973
30115-1 thru 30115-3	June 1973
30124-1 thru 30124-3	June 1973



ASYNCHRONOUS TERMINAL CONTROLLER

JUNE 1973

30032-1

TERMINAL DATA STATUS WORD (TIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

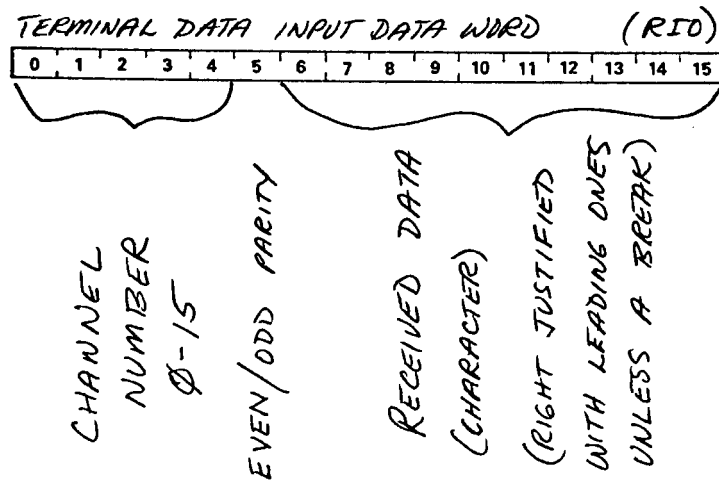
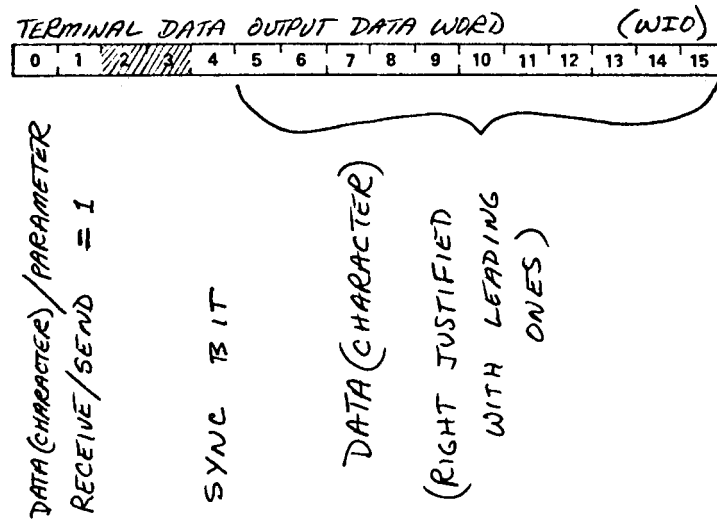
SIO READY $\downarrow$
 RIO-WIO READY
 INTERRUPT REQUEST
 READ-WRITE COMPLETION FLAG
 RECEIVE/SEND CHARACTER
 CHARACTER LOST
 BREAK STATUS

TERMINAL DATA CONTROL WORD (CIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

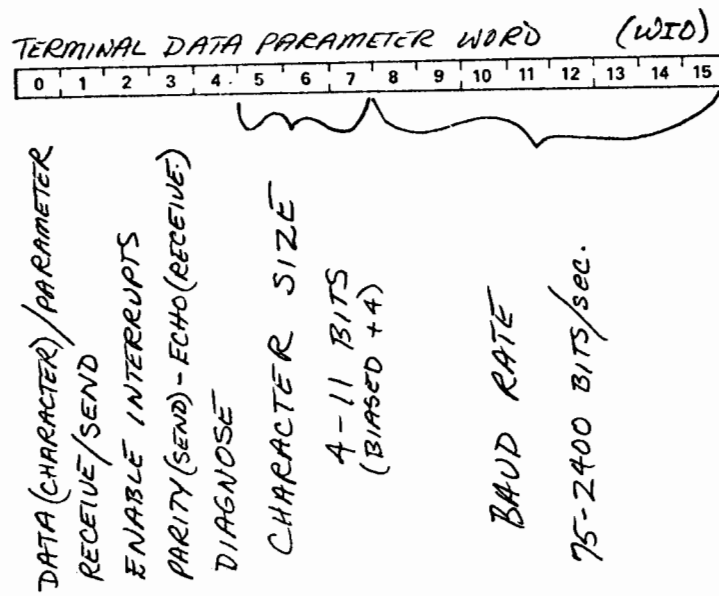
MASTER RESET
 RESET INTERRUPTS
 CHANNEL
 NUMBER
 0-15, 16-20

OUTPUT DATA
 ACKNOWLEDGE INTERRUPT

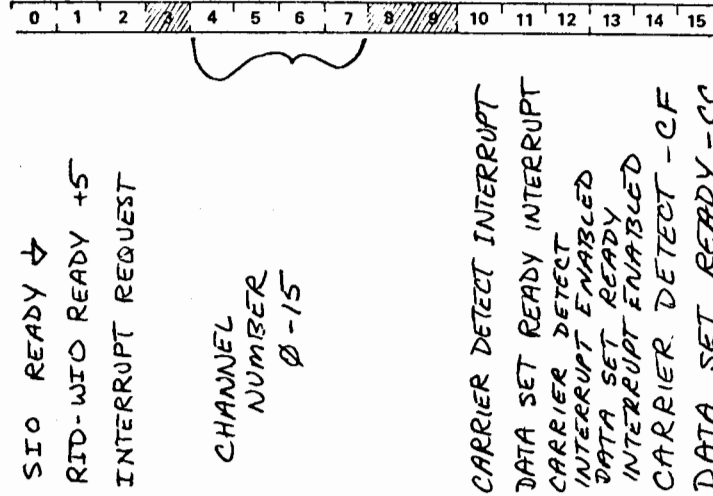


JUNE 1973

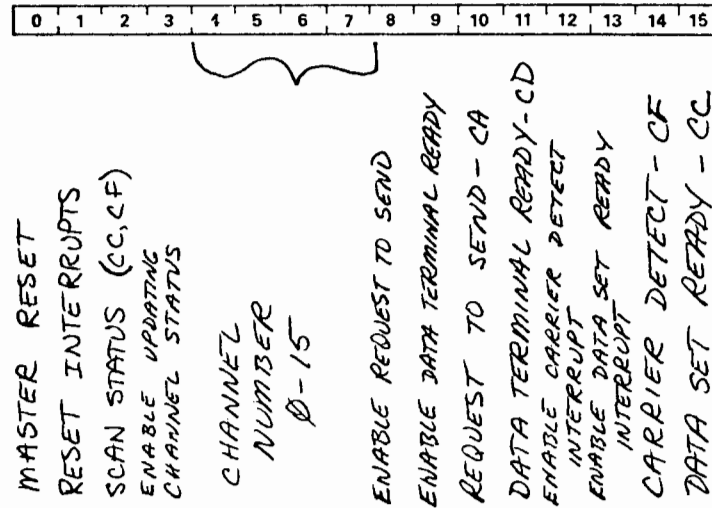
30032-3



TERMINAL CONTROL #1 STATUS WORD (TIO)



TERMINAL CONTROL #1 CONTROL WORD (CIO)



JUNE 1973

30032-5

Pin	Circuit	Signal Description
1	AA	Protective Ground
2	BA	Transmitted Data
3	BB	Received Data
4	CA	Request to Send
5	CB	Clear to Send
6	CC	Data Set Ready
7	AB	Signal Ground (Common Return)
8	CF	(Carrier Detect)
9	--	(Reserved for Data Set Testing)
10	--	(Reserved for Data Set Testing)
11		Unassigned
12	SCF	Secondary (Carrier Detect)
13	SCB	Secondary Clear to Send
14	SBA	Secondary Transmitted Data
15	DB	Transmission Signal Element Timing (DCE Source)
16	SBB	Secondary Received Data
17	DD	Receiver Signal Element Timing (DCE Source)
18		Unassigned
19	SCA	Secondary Request to Send
20	CD	Data Terminal Ready
21	CG	Signal Quality Detector
22	CE	Ring Indicator
23	CH/CI	Data Signal Rate Selector (DTE/DCE Source)
24	DA	Transmit Signal Element Timing (DTE Source)
25		Unassigned

DTE (Data Terminal Equipment)
DCE (Data Communications Equipment)

Circuit	Signal Description	Gnd	Data		Control		Timing	
			From DCE	To DCE	From DCE	To DCE	From DCE	To DCE
AA	Protective Ground	X						
AB	Signal Ground/Common Return	X						
BA	Transmitted Data		X	X				
BB	Received Data							
CA	Request to Send				X	X		
CB	Clear to Send				X			
CC	Data Set Ready				X			
CD	Data Terminal Ready					X		
CE	Ring Indicator				X			
CF	Carrier Detect				X			
CG	Signal Quality Detector				X			
CH	Data Signal Rate Selector (DTE)					X		
CI	Data Signal Rate Selector (DCE)				X			
DA	Transmitter Signal Element Timing (DTE)							X
DB	Transmitter Signal Element Timing (DCE)						X	
DD	Receiver Signal Element Timing (DCE)						X	
SBA	Secondary Transmitted Data		X	X				
SBB	Secondary Received Data							
SCA	Secondary Request to Send				X	X		
SCB	Secondary Clear to Send				X			
SCF	Secondary Carrier Detect				X			

JUNE 1973

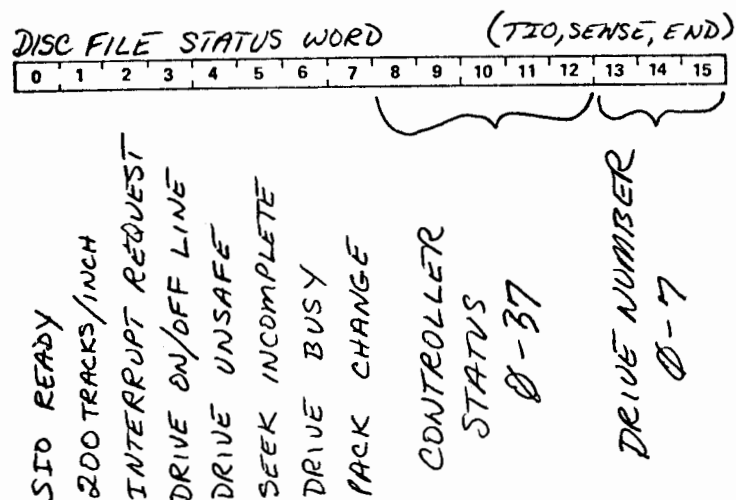
30032-7

30102A/30202A/2888A

DISC FILE

JUNE 1973

30102-1



CONTROLLER STATUS

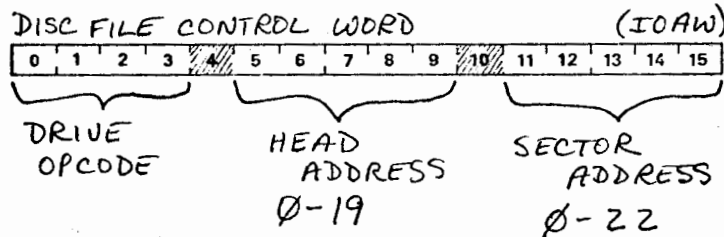
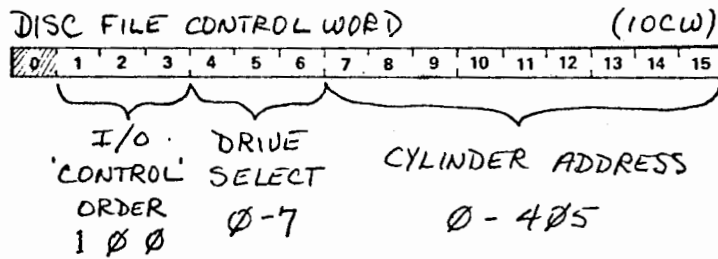
8	9	10	11	12	
0	0	0	0	0	ERROR FREE
0	0	0	0	1	ILLEGAL OP CODE
0	0	0	1	1	CYLINDER NUMBER > 405
0	0	1	0	0	HEAD NUMBER > 19
0	0	1	0	1	TIME-OUT, SEARCH FOR ADDRESS (HEAD, SECTOR) > 100MS
0	0	1	1	0	DEFECTIVE TRACK ADDRESSED
0	0	1	1	1	HEADS MISPOSITIONED
0	1	0	0	0	CYCLIC ERROR IN ADDRESS FIELD
0	1	0	0	1	CYCLIC ERROR IN DATA FIELD
0	1	0	1	0	I/O PROGRAM ERROR
0	1	0	1	1	SEQUENCE ERROR - DATA FIELD DOES NOT FOLLOW AN ADDRESS FIELD DURING SEARCH

(CONT.)

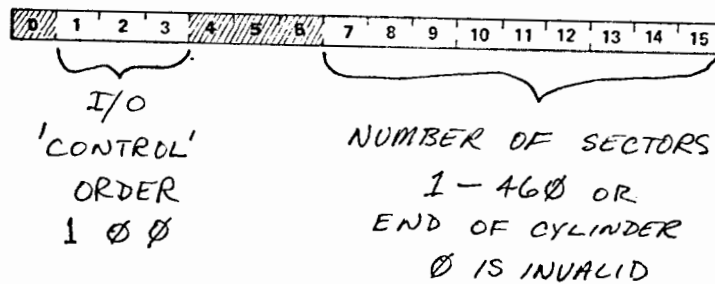
CONTROLLER STATUS (CONT.)

8	9	10	11	12
---	---	----	----	----

0	1	1	0	0	CYLINDER OVERRUN
0	1	1	0	1	ZERO SECTOR COUNT IN CYCLIC CHECK OP CODE
0	1	1	1	0	DATA OVERRUN
1	0	0	0	0	ILLEGAL TERMINATION - I/O TERMINATED BEFORE COMPLET- ING A READ OR WRITE ADDRESS
1	0	0	1	1	DRIVE ERROR
1	0	1	0	0	TRANSFER ERROR
1	1	1	1	1	DRIVE ATTENTION



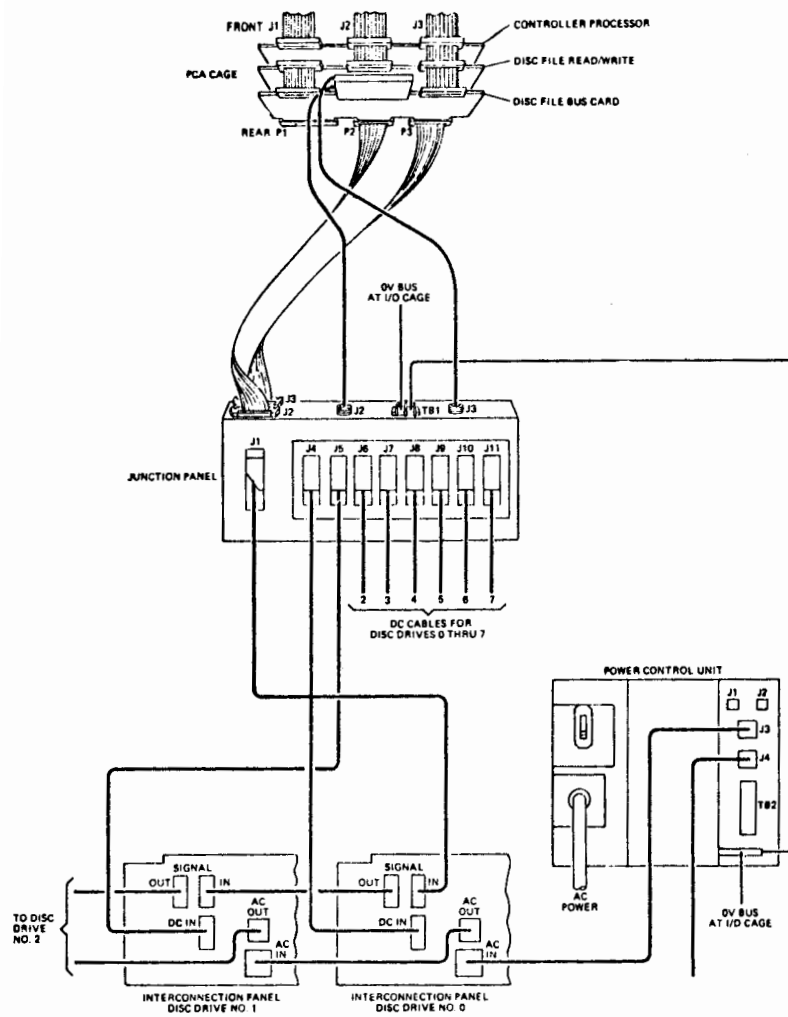
WHEN THE DRIVE OPCODE IS A 'CYCLIC CHECK',
A SECOND IOW IS REQUIRED CONTAINING
THE NUMBER OF SECTORS TO BE CHECKED.
THE SECOND IOAW IN THIS CASE IS IGNORED.



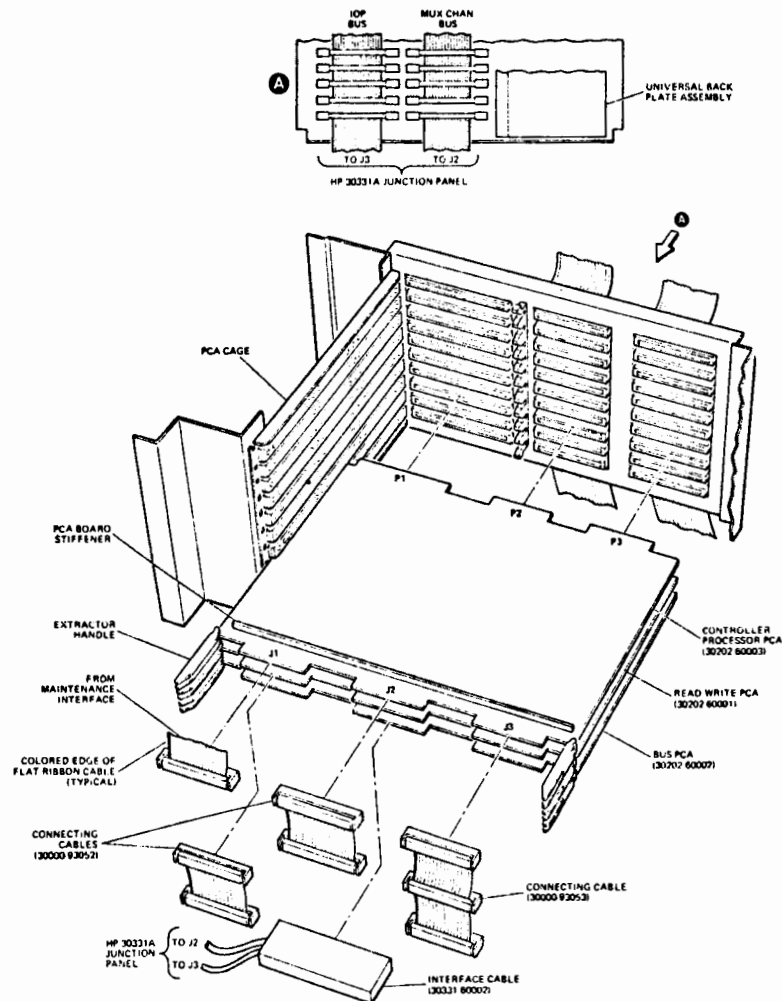
DRIVE OP CODE

	0	1	2	3	
CLR	0	0	0	0	COLD LOAD READ
RC	0	0	0	1	RECALIBRATE
SEEK	0	0	1	0	SEEK
SC	0	0	1	1	STATUS CHECK
RA	0	1	0	0	READ ADDRESS
RD	0	1	0	1	READ DATA
*RFS	0	1	1	0	READ FULL SECTOR
CC	0	1	1	1	CYCLIC CHECK
WD	1	0	0	0	WRITE DATA
*WFS	1	0	0	1	WRITE FULL SECTOR
*SA	1	0	1	0	SKIP ADDRESS-READ DATA
WA	1	0	1	1	WRITE ADDRESS
*PCT	1	1	0	0	PACK CERTIFICATION

* FOR DIAGNOSING PURPOSES.



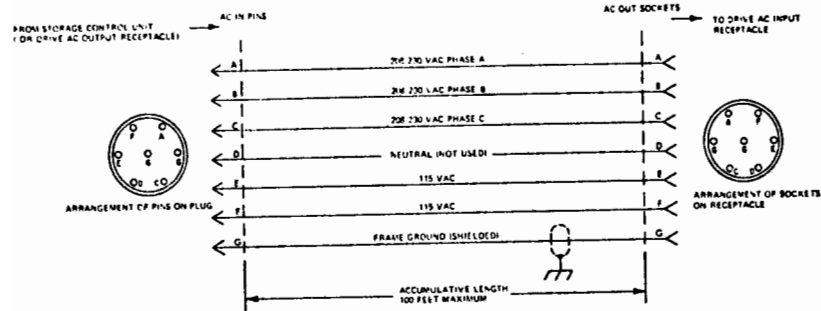
30102A Disc File Subsystem, Cabling Arrangements



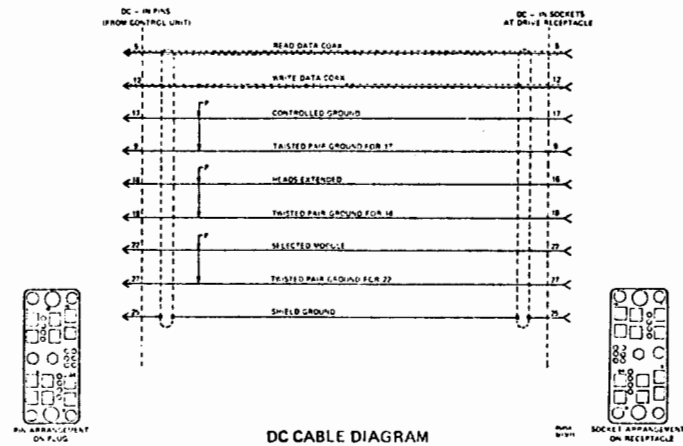
30202A Disc File Interface
PCA Position and Connection Details

JUNE 1973

30102-7



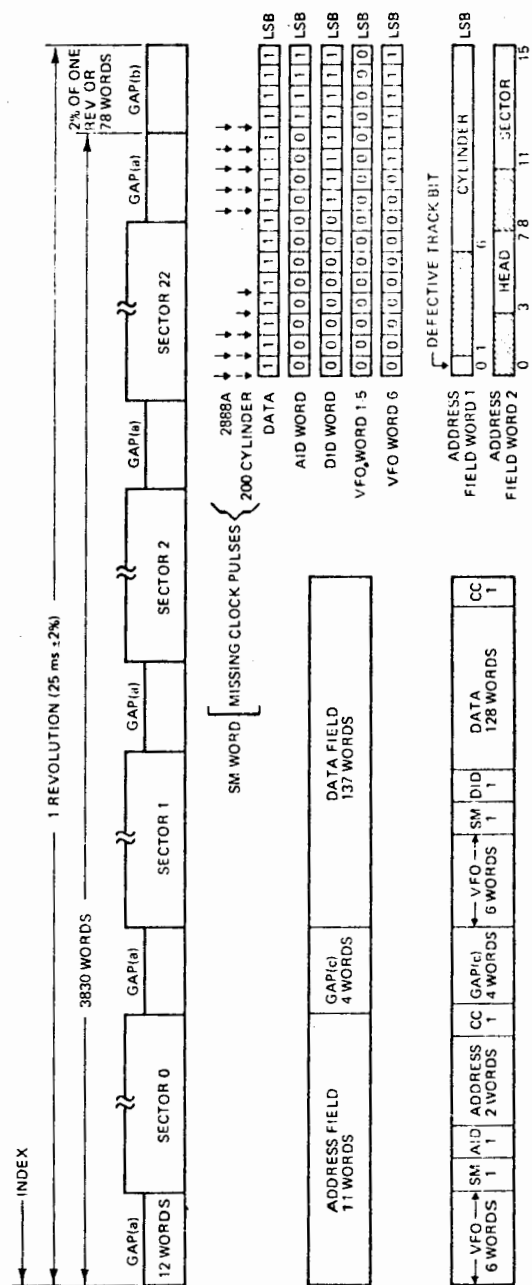
AC CABLE DIAGRAM



DC CABLE DIAGRAM

GAP(a). (b). (c). (d) FILLED WITH ONES
GAP(a) > (5 WORDS + 6% OF PREVIOUS SECTOR)

30102-4

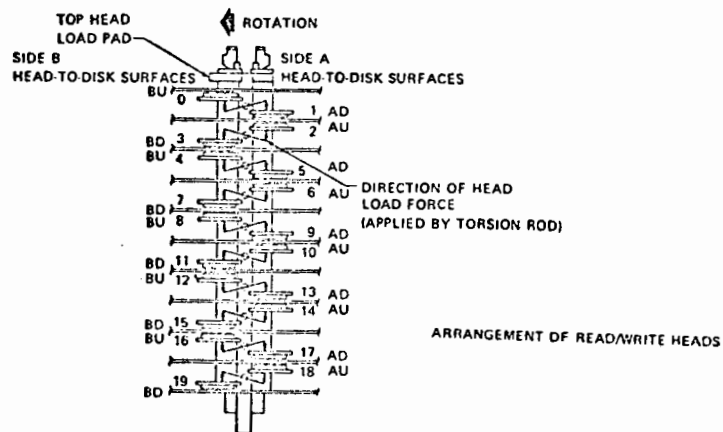


JUN 1973

PWA Assignments in Electronic Gate

GATE POS.	PART NUMRER	PWA TITLE
1	INTERFACE	READ/WRITE PADDLE
2	75000003-X	SELECT 0-9
3	75000068-X	SELECT 10-19
4	75000004-X	WRITE
5	75003548-X	READ
6	75003978-X	DEFECT DETECTOR
7	INTERFACE	SCU SIGNAL PADDLE
8	75004483-X	HEAD ADDRESS REGISTER (HAR)
9	75003727-X	CYLINDER ADDRESS REGISTER (CAR)
10	INTERFACE	DIAG UNIT DISPLAY PADDLE
11	NOT USED	
12	75000011-X	FILE STATUS
13	75003728-X	DIFFERENCE COUNTER
14	75003370-X	ACCESS CONTROL
15	75002523-X	CURVE GENERATOR
16	75000070-X	LOGICAL ADDRESSING
17	75000015-X	INDEX TRANSDUCER
18	75004139-X	TEMPERATURE COMP & SAFETY
19	INTERFACE	TRANSDUCER PADDLE
20	75003713-X	CYLINDER DETECTOR
21	75002521-X	AUTOMATIC GAIN CONTROL (AGC)
22	NOT USED	
23	75003719-X	TACHOMETER
24	75003720-X	PULSER AND SUMMING AMP
25	75004138-X	CYLINDER SELECTOR
26	INTERFACE	PWR SEQ AND PULSER PADDLE
27	75000019-X	POWER CONTROL
*28	75000065-X	MATRIX A
*29	75000065-X	MATRIX B

*These PWA's located on CAM Tower Assembly.



30103A/30203A/2660A

FIXED HEAD DISC

JUNE 1973

30103-1

FIXED HEAD DISC STATUS WORD (TIO, SENSE, END)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

SIO READY
 RFO-WIO READY &
 INTERRUPT REQUEST
 DISC BUSY
 CRC ERROR
 SIO ABORT (ERROR STATUS)
 TRANSFER ERROR
 TRACK ADDRESS TOO LARGE
 WRITE REJECT
 SECTOR ADDRESS TOO LARGE

CURRENT
 SECTOR (ARC)
 ADDRESS
 0-31

FIXED HEAD DISC CONTROL WORD (CIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

MASTER RESET
 RESET INTERRUPTS

STATUS
 WORD
 SELECTION

14	15
----	----

0 0 NORMAL STATUS WORD
 - 1 ERROR STATUS WORD 1
 1 0 ERROR STATUS WORD 2

FIXED HEAD DISC ERROR STATUS WORD 1 (TIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

SAME AS NORMAL
STATUS WORD

SECTOR ADDRESS
WHERE ERROR
OCCURED

FIXED HEAD DISC ERROR STATUS WORD 2 (TIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

SAME AS NORMAL
STATUS WORD

TRACK ADDRESS WHERE
ERROR OCCURED

FIXED HEAD DISC CONTROL WORD (IOW)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

I/O
'CONTROL'
ORDER
1 0 0

SECTOR ADDRESS
0-31

FIXED HEAD DISC CONTROL WORD (IOAW)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

FULL TRANSFER

TRACK ADDRESS

0-255 OPT. 1

0-511 OPT. 2

JUNE 1973

30103-3

30106/7A/30206A/2893A/2950A

CARD READER

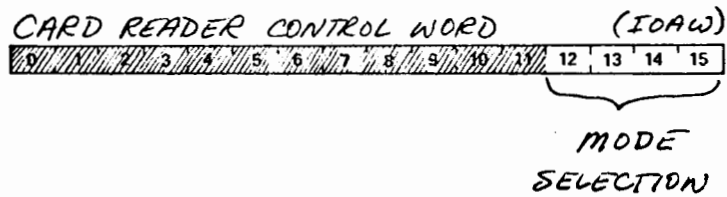
JUNE 1973

30106-1

CARD READER STATUS WORD (TIO, SENSE, END)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SIO READY	RIO-WIO READY ↓	INTERRUPT REQUEST	TIMING OR TRANSFER ERROR	LIGHT-DARK CHECK ERROR	PACKED BINARY	ASCII	*COMPARE ERROR	END OF FILE	STACKER FULL	HOPPER EMPTY OR STACKER FULL	INVALID HOLLERITH	PICK FAIL OR MOTION CHECK	TEST MODE (OFFLINE)	ANY TROUBLE (3, 4, 7, 10, 11 OR 12)	BUSY OR OFFLINE

*COMPARE ERROR LOGIC EXISTS ON THE
INTERFACE FOR USE BY THE 2950A
READER HAVING A DUAL READ STATION
(OPTION 001)

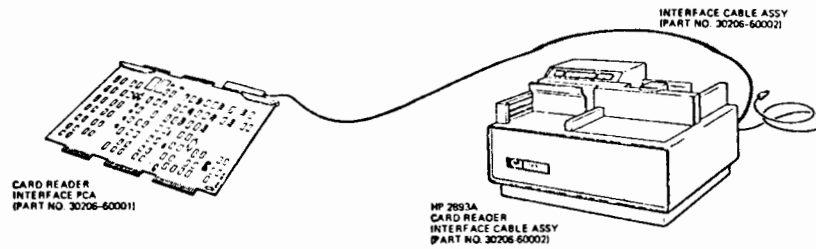


MODE SELECTION

12	13	14	15
----	----	----	----

0	1	0	1	COLUMN BINARY
1	0	0	1	PACKED BINARY
1	0	1	0	ASCII CONVERTED HOLLERITH
0	0	0	0	NO MODE CHANGE

30106/7A/30206A/2893A/2950A



CARD READER INTERFACE PCA		CARD READER	
J1	P1	P2	J2
CARD ROW 12	45	23A	WHT-BRN BLU
CARD ROW 12	46	23B	WHT
CARD ROW 11	21	11B	WHT
CARD ROW 11	22	11A	WHT-BLK BRN
CARD ROW 0	29	15B	WHT
CARD ROW 0	30	15A	WHT-BLK GRN
CARD ROW 1	39	20B	WHT
CARD ROW 1	40	20A	WHT-BRN GRN
CARD ROW 2	35	18B	WHT
CARD ROW 2	36	18A	WHT-BLK GRY
CARD ROW 3	25	13A	WHT-BLK GRN
CARD ROW 3	26	13B	WHT
CARD ROW 4	43	22B	WHT
CARD ROW 4	44	22A	WHT-BRN GRN
CARD ROW 5	47	24B	WHT
CARD ROW 5	48	24A	WHT-BRN VIO
CARD ROW 6	41	21A	WHT-BRN YEL
CARD ROW 6	42	21B	WHT
CARD ROW 7	33	17A	WHT-BLK VIO
CARD ROW 7	34	17B	WHT
CARD ROW 8	37	19A	WHT-BRN RED
CARD ROW 8	38	19B	WHT
CARD ROW 9	49	25A	WHT-BRN GRY
CARD ROW 9	50	25B	WHT
ERROR(LIGHT/DARK)	31	16A	WHT-BLK BLU
ERROR(LIGHT/DARK)	32	16B	WHT
HOPPER EMPTY	17	9B	WHT
HOPPER EMPTY	18	9A	WHT-VIO
MOTION/PICK CHECK	7	4A	WHT-RED
MOTION/PICK CHECK	8	4B	WHT
PICK COMMAND	3	2A	WHT-BLK
PICK COMMAND	4	2B	WHT
NC	NC		
INDEX MARK	19	10B	WHT
INDEX MARK	20	10A	WHT GRY
STACKER FULL	9	5B	WHT
STACKER FULL	10	5A	WHT-GRN
END OF FILE CLEAR	5	3B	WHT
END OF FILE CLEAR	6	3A	WHT-BRN
END OF FILE	27	14B	WHT
END OF FILE	28	14A	WHT-BLK YEL
OFF LINE	13	7B	WHT
OFF LINE	14	7A	WHT-GRN
+5 VOLTS	NC		
COMMON	11	6A	WHT YEL
COMPARE	16	8B	WHT
COMPARE	15	8A	WHT-BLU
READY	23	12A	WHT-BLK RED
READY	24	12B	WHT

PINS NOT LISTED ARE NOT CONNECTED

Interface Cable Connector Pin Index

30106-4

JUNE 1973

30108/9A/30057A/2610/14A

LINE PRINTER

JUNE 1973

30108-1

LINE PRINTER INTERRUPT STATUS WORD (TIO, SENSE, END)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

SIO READY
 RIO-WIO READY
 INTERRUPT REQUEST
 SEQUENCE FF
 DEVICE FLAG
 INTERRUPT/DEVICE STATUS = 0
 LINE READY INTERRUPT
 READY INTERRUPT
 NOT READY INTERRUPT
 DATA TRANSFER
 COMPLETION INTERRUPT
 CLEAR INTERFACE INTERRUPT
 I/O SYSTEM INTERRUPT
 TRANSFER ERROR
 WATCHDOG TIMER INTERRUPT

LINE PRINTER DEVICE STATUS WORD (TIO, SENSE, END)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

SIO READY
 RIO-WIO READY
 INTERRUPT REQUEST
 SEQUENCE FF
 DEVICE FLAG
 INTERRUPT/DEVICE STATUS = 1
 LINE READY
 READY
 PAPER OUT

LINE PRINTER CONTROL WORD (CIO, CONTROL)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

MASTER CLEAR
RESET INTERRUPTS
INTERRUPT RESET
SELECTION



SET PRINT COMMAND
ENABLE DATA TRANSFER
INTERRUPT
INTERRUPT/DEVICE STATUS
WORD/BYTE TRANSFER
ENABLE INTERRUPTS
ENABLE WATCHDOG TIMER

INTERRUPT RESET SELECTION

2	3	4
---	---	---

0	0	0	NO RESET
0	0	1	WATCHDOG TIMER AND TRANSFER ERROR
0	1	0	I/O SYSTEM
0	1	1	CLEAR INTERFACE
1	0	0	DATA TRANSFER COMPLETION
1	0	1	LINE READY
1	1	0	READY
1	1	1	NOT READY

SEQUENCE FLIP FLOPS

2	3
---	---

0	0	NO TRANSFER IN PROGRESS
1	0	DEVICE COMMAND ISSUED
1	1	DEVICE FLAG RECEIVED
0	1	SECOND DEVICE COMMAND ISSUED (BYTE TRANSFER)

JUNE 1973

30108-3

LINE PRINTER BYTE MODE DATA WORD (W10, WRITE)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

A A A A A A A B B B B B B B

1st ASCII CHARACTER 2nd ASCII CHARACTER

LINE PRINTER WORD MODE DATA WORD (W10, WRITE)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

A A A A A A A

ASCII CHARACTER

LINE PRINTER FORMAT WORD (W10, WRITE)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

F F F F F F F

A LINE PRINTER FORMAT WORD FOLLOWS
A CONTROL WORD HAVING BITS 10 AND
13 SET AND CAUSES THE PRINTER TO
PRINT AND FORMS CONTROL

30110A/30210A/7900A

CARTRIDGE DISC

JUN 1973

30110-1

CARTRIDGE DISC STATUS WORD (*TIO, SENSE, END)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

STO READY
 SEEK CHECK
 INTERRUPT REQUEST
 PROTECT OVERRIDE
 DRIVE UNSAFE
 DRIVE BUSY
 ACCESS NOT READY

CONTROLLER

STATUS

Ø-37

DRIVE
SELECTED

Ø-3

* DURING DATA TRANSFER ONLY STATUS
 BITS Ø, 2, 14 AND 15 ARE VALID USING
 A 'TIO' INSTRUCTION.

CONTROLLER STATUS

8	9	10	11	12
---	---	----	----	----

Ø Ø Ø Ø Ø ERROR FREE
 Ø Ø Ø Ø 1 ILLEGAL OPCODE
 Ø Ø 1 Ø Ø SECTOR TOO LARGE
 Ø Ø 1 1 Ø DP ATTEMPTED ON DEFECTIVE TRACK
 Ø Ø 1 1 1 HEADS MISPOSITIONED
 Ø 1 Ø Ø 1 CYCLIC ERROR
 Ø 1 Ø 1 Ø I/O PROGRAM ERROR
 Ø 1 1 Ø Ø CYLINDER OVERRUN
 Ø 1 1 Ø 1 ZERO SECTOR COUNT IN CYCLIC
 CHECK OPCODE
 Ø 1 1 1 Ø DATA OVERRUN

CONTROLLER STATUS (CONT.)

8	9	10	11	12
---	---	----	----	----

1	0	0	1	0	HEAD-SECTOR COMPARE ERROR
1	0	0	1	1	ACCESS NOT READY
1	0	1	0	0	TRANSFER ERROR
1	0	1	1	0	PROTECT NOT OVERRIDEN
1	1	1	1	1	DRIVE ATTENTION

CARTRIDGE DISC CONTROL WORD (IOCW)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

I/O 'CONTROL' ORDER 1 0 0 UNIT 0-3 CYLINDER ADDRESS 0-202

CARTRIDGE DISC CONTROL WORD (IOAW)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

DRIVE OPCODE HEAD 0-3 SECTOR 0-23

WHEN THE DRIVE OPCODE IS A 'CYCLIC CHECK',
 A SECOND IOCW IS REQUIRED CONTAINING
 THE NUMBER OF SECTORS TO BE CHECKED.
 THE SECOND IOAW IN THIS CASE IS IGNORED.

CARTRIDGE DISC CONTROL WORD (IDCW)*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

I/O
'CONTROL'
ORDER
1 0 0

NUMBER OF SECTORS
1-48 OR
END OF CYLINDER
0 IS INVALID

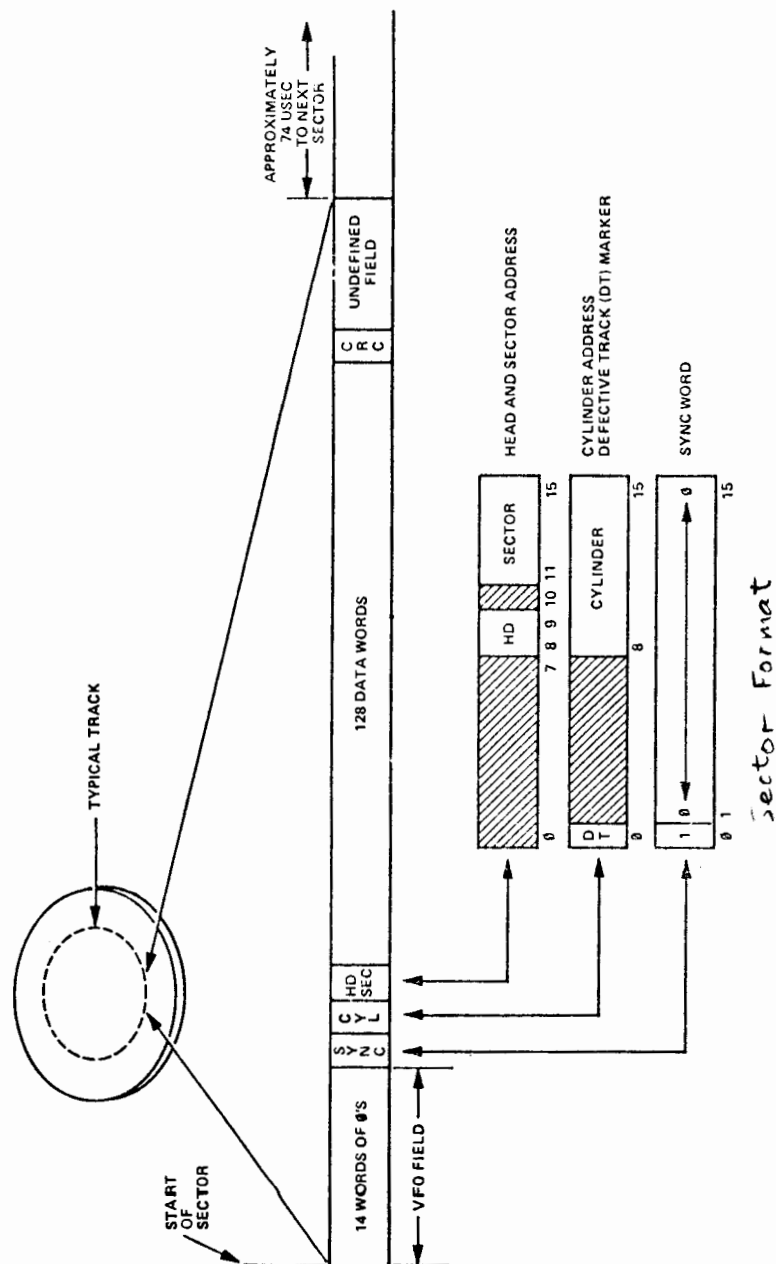
* AS A RESULT OF A CYCLIC CHECK OP CODE

DRIVE OP CODE

0	1	2	3
---	---	---	---

RD	0 0 0 0	READ DATA (COLD LOAD)
RC	0 0 0 1	RECALIBRATE (RESET SEEK CHECK ERROR)
SEEK	0 0 1 0	SEEK
SC	0 0 1 1	STATUS CHECK
FTD	0 1 0 0	FLAG TRACK DEFECTIVE
RD	0 1 0 1	READ DATA (PROGRAMMATIC)
*RFS	0 1 1 0	READ FULL SECTOR
CC	0 1 1 1	CYCLIC CHECK
WD	1 0 0 0	WRITE DATA
*WFS	1 0 0 1	WRITE FULL SECTOR
RNFS	1 0 1 0	READ NEXT FULL SECTOR (I/O SCHEDULING)
ID	1 0 1 1	INITIALIZE DATA

* FOR DIAGNOSING PURPOSES.



JUNE 1973

30110-5

30115A/30215A/7970B/E

MAGNETIC TAPE

JUNE 1973

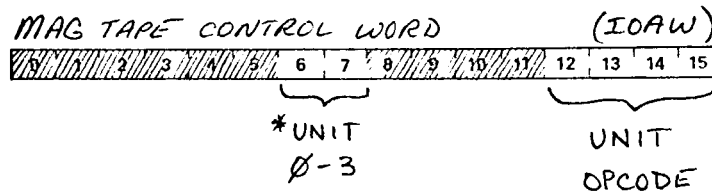
30115-1

MAG TAPE STATUS: WORD															(TIO, SENSE, END)		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
SIO READY			UNIT			END OF TAPE (EOT)			WRITE FILE PROTECT			UNIT READY (ON LINE)			BEGINNING OF TAPE (BOT)		
ODD BYTES READ			INTERRUPT REQUEST			800/1600 CPI			WRITE STATUS			FILE MARK DETECTED			CONTROLLER STATUS		

CONTROLLER STATUS

12	13	14	
0	0	0	UNIT READY TO READY INTERRUPT
0	0	1	TRANSFER ERROR
0	1	0	COMMAND REJECTED
0	1	1	TAPE RUNAWAY (25 FEET)
1	0	0	TIMING ERROR
1	0	1	TAPE ERROR (PARITY, CRC, etc.)
1	1	1	ERROR FREE

110 RECEIVED



* USED FOR A 'UNIT SELECT' OPCODE ONLY

UNIT OPCODE

	12	13	14	15	
SEL	0	0	0	0	UNIT SELECT
WRR	0	1	0	0	WRITE RECORD
GAP	0	1	0	1	WRITE GAP
RDR	0	1	1	0	READ RECORD
FSR	0	1	1	1	FORWARD SPACE RECORD
REW	1	0	0	0	REWIND
RST	1	0	0	1	REWIND AND RESET(OFFLINE)
BSR	1	0	1	0	BACKSPACE RECORD
BSF	1	0	1	1	BACKSPACE FILE
Δ WRZ	1	1	0	0	WRITE RECORD WITH ZERO PARITY
WFM	1	1	0	1	WRITE FILE MARK
Δ RDC	1	1	1	0	READ RECORD WITH CRCC
FSF	1	1	1	1	FORWARD SPACE FILE

Δ USED FOR DIAGNOSING PURPOSES.

30124A/30031A/2749B

CONSOLE/TIMER

JUNE 1973

30124-1

CONSOLE/TIMER STATUS WORD (TID)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

SIO READY $\downarrow$
 RIO-WIO READY $\downarrow$
 COUNT RATE
 SELECTED

INPUT DATA OVERRUN
 INPUT DATA READY
 READY FOR OUTPUT DATA
 DATA TERMINAL READY

COUNT REGISTER READ ERROR
 LIMIT REG. = COUNT REG.
 LIM. REG. = COUNT REG. OVERFLOW
 BREAK STATUS

I/O SYSTEM INTERRUPT
 LIM. REG./COUNT REG. SELECTED
 RESET COUNT REG. AFTER INTERRUPT

COUNT RATE

2	3	4	
0	0	0	1 usec
0	0	1	10 usec
0	1	0	100 usec
0	1	1	1 msec
1	0	0	10 msec
1	0	1	100 msec
1	1	0	1 sec
1	1	1	10 sec

CONSOLE/TIMER CONTROL WORD (CIO)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

MASTER RESET (BIT 3 = 0)
COUNT RATE

LOAD COUNT RATE

ENABLE CONSOLE INTERRUPTS

INTERRUPT RESET
SELECTION

RESET COUNT REGISTER
AFTER INTERRUPT

RIO-WIO TO LIM. REG./COUNT REG.

RESET INTERRUPTS

ECHO (RECEIVED DATA)

RIO-WIO TO CONSOLE/TIMER

RESET CONSOLE

ENABLE TIMER INTERRUPTS

INTERRUPT RESET SELECTION

5	6	7
---	---	---

0	0	0	NO RESET
0	0	1	LIMIT REG. = COUNT REG.
0	1	0	LIMIT REG. = COUNT REG. OVERFLOW
0	1	1	I/O SYSTEM (SIN)
1	0	0	INPUT DATA OVERRUN
1	0	1	INPUT DATA READY
1	1	0	READY FOR OUTPUT DATA
1	1	1	COUNT REGISTER READ ERROR

JUNE 1973

30124-3

TABLE OF CONTENTS

GENERAL INFORMATION

OCTAL: DECIMAL CONVERSION	1
HEXADECIMAL TO DECIMAL CONVERSION	5
INTEGER BASE CONVERSION	6
USASCII CODE	7
EBCDIC CODE	8
CODE CONVERSION	9
Hollerith to ASCII	9
IBM 029 Key Punch	9
ASCII to Hollerith	10
ASCII/BYTES	12



LIST OF EFFECTIVE PAGES

PAGES

1 thru 12

DATE

June 1973

OCTAL: DECIMAL CONVERSION

(ASSUME 16 BIT POSITIVE INTEGER)

OCTAL DECIMAL		OCTAL DECIMAL		OCTAL DECIMAL	
0	0	60000	24576	140000	49152
1000	512	61000	25088	141000	49664
2000	1024	62000	25600	142000	50176
3000	1536	63000	26112	143000	50688
4000	2048	64000	26624	144000	51200
5000	2560	65000	27136	145000	51712
6000	3072	66000	27648	146000	52224
7000	3584	67000	28160	147000	52736
10000	4096	70000	28672	150000	53248
11000	4608	71000	29184	151000	53760
12000	5120	72000	29696	152000	54272
13000	5632	73000	30208	153000	54784
14000	6144	74000	30720	154000	55296
15000	6656	75000	31232	155000	55808
16000	7168	76000	31744	156000	56320
17000	7680	77000	32256	157000	56832
20000	8192	100000	32768	160000	57344
21000	8704	101000	33280	161000	57856
22000	9216	102000	33792	162000	58368
23000	9728	103000	34304	163000	58880
24000	10240	104000	34816	164000	59392
25000	10752	105000	35328	165000	59904
26000	11264	106000	35840	166000	60416
27000	11776	107000	36352	167000	60928
30000	12288	110000	36864	170000	61440
31000	12800	111000	37376	171000	61952
32000	13312	112000	37888	172000	62464
33000	13824	113000	38400	173000	62976
34000	14336	114000	38912	174000	63488
35000	14848	115000	39424	175000	64000
36000	15360	116000	39936	176000	64512
37000	15872	117000	40448	177000	65024
40000	16384	120000	40960	177777	65535
41000	16896	121000	41472		
42000	17408	122000	41984		
43000	17920	123000	42496		
44000	18432	124000	43008		
45000	18944	125000	43520		
46000	19456	126000	44032		
47000	19968	127000	44544		
50000	20480	130000	45056		
51000	20992	131000	45568		
52000	21504	132000	46080		
53000	22016	133000	46592		
54000	22528	134000	47104		
55000	23040	135000	47616		
56000	23552	136000	48128		
57000	24064	137000	48640		

OCTAL: DECIMAL CONVERSION

2'S COMP		0	1	2	3	4	5	6	7
0	0	0	1	2	3	4	5	6	7
177770	10	8	9	10	11	12	13	14	15
177760	20	16	17	18	19	20	21	22	23
177750	30	24	25	26	27	28	29	30	31
177740	40	32	33	34	35	36	37	38	39
177730	50	40	41	42	43	44	45	46	47
177720	60	48	49	50	51	52	53	54	55
177710	70	56	57	58	59	60	61	62	63
177700	100	64	65	66	67	68	69	70	71
177670	110	72	73	74	75	76	77	78	79
177660	120	80	81	82	83	84	85	86	87
177650	130	88	89	90	91	92	93	94	95
177640	140	96	97	98	99	100	101	102	103
177630	150	104	105	106	107	108	109	110	111
177620	160	112	113	114	115	116	117	118	119
177610	170	120	121	122	123	124	125	126	127
177600	200	128	129	130	131	132	133	134	135
177570	210	136	137	138	139	140	141	142	143
177560	220	144	145	146	147	148	149	150	151
177550	230	152	153	154	155	156	157	158	159
177540	240	160	161	162	163	164	165	166	167
177530	250	168	169	170	171	172	173	174	175
177520	260	176	177	178	179	180	181	182	183
177510	270	184	185	186	187	188	189	190	191
177500	300	192	193	194	195	196	197	198	199
177470	310	200	201	202	203	204	205	206	207
177460	320	208	209	210	211	212	213	214	215
177450	330	216	217	218	219	220	221	222	223
177440	340	224	225	226	227	228	229	230	231
177430	350	232	233	234	235	236	237	238	239
177420	360	240	241	242	243	244	245	246	247
177410	370	248	249	250	251	252	253	254	255
177400	400	256	257	258	259	260	261	262	263
177370	410	264	265	266	267	268	269	270	271
177360	420	272	273	274	275	276	277	278	279
177350	430	280	281	282	283	284	285	286	287
177340	440	288	289	290	291	292	293	294	295
177330	450	296	297	298	299	300	301	302	303
177320	460	304	305	306	307	308	309	310	311
177310	470	312	313	314	315	316	317	318	319
177300	500	320	321	322	323	324	325	326	327
177270	510	328	329	330	331	332	333	334	335
177260	520	336	337	338	339	340	341	342	343
177250	530	344	345	346	347	348	349	350	351
177240	540	352	353	354	355	356	357	358	359
177230	550	360	361	362	363	364	365	366	367
177220	560	368	369	370	371	372	373	374	375
177210	570	376	377	378	379	380	381	382	383

OCTAL: DECIMAL CONVERSION

2'S COMP		0	1	2	3	4	5	6	7
177200	600	384	385	386	387	388	389	390	391
177170	610	392	393	394	395	396	397	398	399
177160	620	400	401	402	403	404	405	406	407
177150	630	408	409	410	411	412	413	414	415
177140	640	416	417	418	419	420	421	422	423
177130	650	424	425	426	427	428	429	430	431
177120	660	432	433	434	435	436	437	438	439
177110	670	440	441	442	443	444	445	446	447
177100	700	448	449	450	451	452	453	454	455
177070	710	456	457	458	459	460	461	462	463
177060	720	464	465	466	467	468	469	470	471
177050	730	472	473	474	475	476	477	478	479
177040	740	480	481	482	483	484	485	486	487
177030	750	488	489	490	491	492	493	494	495
177020	760	496	497	498	499	500	501	502	503
177010	770	504	505	506	507	508	509	510	511
177000	1000	512	513	514	515	516	517	518	519
176770	1010	520	521	522	523	524	525	526	527
176760	1020	528	529	530	531	532	533	534	535
176750	1030	536	537	538	539	540	541	542	543
176740	1040	544	545	546	547	548	549	550	551
176730	1050	552	553	554	555	556	557	558	559
176720	1060	560	561	562	563	564	565	566	567
176710	1070	568	569	570	571	572	573	574	575
176700	1100	576	577	578	579	580	581	582	583
176670	1110	584	585	586	587	588	589	590	591
176660	1120	592	593	594	595	596	597	598	599
176650	1130	600	601	602	603	604	605	606	607
176640	1140	608	609	610	611	612	613	614	615
176630	1150	616	617	618	619	620	621	622	623
176620	1160	624	625	626	627	628	629	630	631
176610	1170	632	633	634	635	636	637	638	639
176600	1200	640	641	642	643	644	645	646	647
176570	1210	648	649	650	651	652	653	654	655
176560	1220	656	657	658	659	660	661	662	663
176550	1230	664	665	666	667	668	669	670	671
176540	1240	672	673	674	675	676	677	678	679
176530	1250	680	681	682	683	684	685	686	687
176520	1260	688	689	690	691	692	693	694	695
176510	1270	696	697	698	699	700	701	702	703
176500	1300	704	705	706	707	708	709	710	711
176470	1310	712	713	714	715	716	717	718	719
176460	1320	720	721	722	723	724	725	726	727
176450	1330	728	729	730	731	732	733	734	735
176440	1340	736	737	738	739	740	741	742	743
176430	1350	744	745	746	747	748	749	750	751
176420	1360	752	753	754	755	756	757	758	759
176410	1370	760	761	762	763	764	765	766	767

JUNE 1973

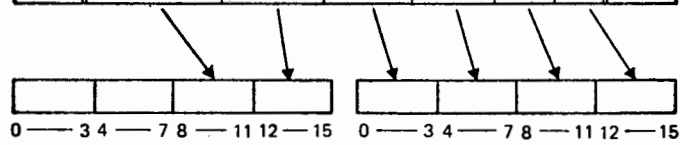
3

OCTAL: DECIMAL CONVERSION

2'S COMP		0	1	2	3	4	5	6	7
176400	1400	768	769	770	771	772	773	774	775
176370	1410	776	777	778	779	780	781	782	783
176360	1420	784	785	786	787	788	789	790	791
176350	1430	792	793	794	795	796	797	798	799
176340	1440	800	801	802	803	804	805	806	807
176330	1450	808	809	810	811	812	813	814	815
176320	1460	816	817	818	819	820	821	822	823
176310	1470	824	825	826	827	828	829	830	831
176300	1500	832	833	834	835	836	837	838	839
176270	1510	840	841	842	843	844	845	846	847
176260	1520	848	849	850	851	852	853	854	855
176250	1530	856	857	858	859	860	861	862	863
176240	1540	864	865	866	867	868	869	870	871
176230	1550	872	873	874	875	876	877	878	879
176220	1560	880	881	882	883	884	885	886	887
176210	1570	888	889	890	891	892	893	894	895
176200	1600	896	897	898	899	900	901	902	903
176170	1610	904	905	906	907	908	909	910	911
176160	1620	912	913	914	915	916	917	918	919
176150	1630	920	921	922	923	924	925	926	927
176140	1640	928	929	930	931	932	933	934	935
176130	1650	936	937	938	939	940	941	942	943
176120	1660	944	945	946	947	948	949	950	951
176110	1670	952	953	954	955	956	957	958	959
176100	1700	960	961	962	963	964	965	966	967
176070	1710	968	969	970	971	972	973	974	975
176060	1720	976	977	978	979	980	981	982	983
176050	1730	984	985	986	987	988	989	990	991
176040	1740	992	993	994	995	996	997	998	999
176030	1750	1000	1001	1002	1003	1004	1005	1006	1007
176020	1760	1008	1009	1010	1011	1012	1013	1014	1015
176010	1770	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL TO DECIMAL CONVERSION

HEX	DECIMAL						HEX
0	0	0	0	0	0	0	0
1	1 048 576	65 536	4 096	256	16	1	1
2	2 097 152	131 072	8 192	512	32	2	2
3	3 145 728	196 608	12 288	768	48	3	3
4	4 194 304	262 144	16 384	1 024	64	4	4
5	5 242 880	327 680	20 480	1 280	80	5	5
6	6 291 456	393 216	24 576	1 536	96	6	6
7	7 340 032	458 752	28 672	1 792	112	7	7
8	8 388 608	524 288	32 768	2 048	128	8	8
9	9 437 184	589 824	36 864	2 304	144	9	9
A	10 485 760	655 360	40 960	2 560	160	10	A
B	11 534 336	720 896	45 056	2 816	176	11	B
C	12 582 912	786 432	49 152	3 072	192	12	C
D	13 631 488	851 968	53 248	3 328	208	13	D
E	14 680 064	917 504	57 344	3 584	224	14	E
F	15 728 640	983 040	61 440	3 840	240	15	F



JUNE 1973

5

INTEGER BASE CONVERSION

DECIMAL TO BINARY division by 2

Decimal 53. = Binary 110 101.

2 53	remainder =	1
2 26	remainder =	0
2 13	remainder =	1
2 6	remainder =	0
2 3	remainder =	1
2 1	remainder =	1
0		

DECIMAL TO OCTAL division by 8

Decimal 1375 = Octal 2537.

8 1375	remainder =	7
8 171	remainder =	3
8 21	remainder =	5
8 2	remainder =	2

OCTAL TO DECIMAL nested multiplication by 8

Octal 2567. = Decimal 1399.

2							
x8							
16	+	5	= 21				
		x8					
		168	+	6	= 174		
				x8			
				1392	+	7	= 1399

FRACTIONAL BASE CONVERSION

DECIMAL TO OCTAL multiplication by 8

Decimal .2734375 = Octal .214

.2734375	x8		
2.1875000		.1875	
	x8		
	1.5000	.5	
		x8	
		4.0	
			.214

OCTAL TO DECIMAL division by 8

Octal .524 = Decimal .6640625

.5			
8 4.0			
	.3125		
8 2.5000			
	.6640625		
8 5.3125000			

USASCH* CODE

*U.S.A. STANDARD CODE FOR INFORMATION INTERCHANGE,
USAS X3.4-1968

USAS X3.4-1

0-1 / 8-9 2-4 / 10-12	5-7 / 13-15	0	1	2	3	4	5	6	7
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	
01	BS	HT	LF	VT	FF	CR	SO	SI	
02	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	
03	CAN	EM	SUB	ESC	FS	GS	RS	US	
04	SP	!	"	#	\$	%	&	'	
05	(	)	*	+	,	-		/	
06	0	1	2	3	4	5	6	7	
07	8	9	:	;	<	=	>	?	
10	@	A	B	C	D	E	F	G	
11	H	I	J	K	L	M	N	O	
12	P	Q	R	S	T	U	V	W	
13	X	Y	Z	[	\	]	^	_	
14	`	a	b	c	d	e	f	g	
15	h	i	j	k	l	m	n	o	
16	p	q	r	s	t	u	v	w	
17	x	y	z	{		}	~	DEL	

ACK	Acknowledge	FS	File Separator
BEL	Bell	GS	Group Separator
BS	Back Space	HT	Horizontal Tab
CAN	Cancel	LF	Line Feed
CR	Carriage Return	NAK	Negative Acknowledge
DC1-DC4	Device Controls	NUL	Null
DEL	Delete	RS	Record Separator
DLE	Data Link Escape	SI	Shift In
EM	End of Media	SO	Shift Out
ENQ	Enquiry	SOH	Start of Heading
EOT	End of Transmission	STX	Start of Text
ESC	Escape	SUB	Substitute
ETB	End of Xmission Block	SYN	Synchronize
ETX	End of Text	US	Unit Separator
FF	Form Feed	VT	Vertical Tab

(X-ON = DC1; X-OFF = DC3)

2749 Teleprinter	
CNTL	Inhibits bit 1
SHIFT	Complements bit 2
Keyboard	Print requires bit 1 OR bit 2
Parity bit	Bit 0, even parity (ignored by HP Software)

JUNE 1973

*EBCDIC CODE

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																
	1111	SI	IUS	BEL	SUB															

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																
	1111	SI	IUS	BEL	SUB															

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																
	1111	SI	IUS	BEL	SUB															

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																
	1111	SI	IUS	BEL	SUB															

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																
	1111	SI	IUS	BEL	SUB															

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																
	1111	SI	IUS	BEL	SUB															

Bit Positions				00				01				10				11				
				23	00	01	10	11												
4567	0000	NUL	DLE	DS					SP	&	-						{			
	0001	SOH	DC1	SOS							/						A	J		
	0010	STX	DC2	FS													B	K	S	2
	0011	ETX	DC3														C	L	T	3
	0100	PF	RES	BYP	PN												D	M	U	4
	0101	HT	NL	LF	RS												E	N	V	5
	0110	LC	BS	EOB	UC												F	O	W	6
	0111	DEL	IDL	PRE	EOT												G	P	X	7
	1000		CAN														H	Q	Y	8
	1001	RLF	EM														I	R	Z	9
	1010	SMM	CC	SM																LVM
	1011	VT	CUI	CU2	CU3															
	1100	FF	IFS		DC4															
	1101	CR	IGS	ENO	NAK															
	1110	SO	IRS	ACK																

*Extended Binary-Coded-Decimal Interchange Code

HOLLERITH TO ASCII
 64 CHARACTER

CHAR	HOLLERITH	ASCII	CHAR	HOLLERITH	ASCII
0	0	60	Space	No Punch	40
1	1	61	(†)	12,8,7	41
2	2	62	"	8,7	42
3	3	63	#	8,3	43
4	4	64	\$	11,8,3	44
5	5	65	%	0,8,4	45
6	6	66	&	12	46
7	7	67	'	8,5	47
8	8	70	(	12,8,5	50
9	9	71	)	11,8,5	51
A	12,1	101	*	11,8,4	52
B	12,2	102	+	12,8,6	53
C	12,3	103	,	0,8,3	54
D	12,4	104	-	11	55
E	12,5	105	.	12,8,3	56
F	12,6	106	/	0,1	57
G	12,7	107	¢ (l)	12,8,2	133
H	12,8	110	\	0,8,2	134
I	12,9	111	!	11,8,2	135
J	11,1	112	¬ (j)	11,8,7	136
K	11,2	113	:	8,2	72
L	11,3	114	;	11,8,6	73
M	11,4	115	<	12,8,4	74
N	11,5	116	=	8,6	75
O	11,6	117	>	0,8,6	76
P	11,7	120	?	0,8,7	77
Q	11,8	121	@	8,4	100
R	11,9	122	- (→)	0,8,5	137
S	0,2	123			
T	0,3	124			
U	0,4	125			
V	0,5	126			
W	0,6	127			
X	0,7	130			
Y	0,8	131			
Z	0,9	132			

IBM 029 KEY PUNCH

ASCII CHAR	029 KEY	
[	¢	NUMERIC R
\	0.8.2	NUMERIC T
]	!	NUMERIC B
^ or ↑	-	NUMERIC G
!		NUMERIC Y

JUNE 1973

ASCII → HOLLERITH

OCTAL	HOL.	CHAR	%	%	%	%
000	5403	NUL	072	0202	:	162 6001 r
001	4401	SOH	073	2012	:	163 3200 s
002	4201	STX	074	4042	^	164 3100 t
003	4101	EXT	075	0012	=	165 3040 u
004	0005	EOT	076	1012	>	166 3020 v
005	1023	ENQ	077	1006	?	167 3010 w
006	1013	ACK	100	0042	@	170 3004 x
007	1007	BEL	101	4400	A	171 3002 y
010	2011	BS	102	4200	B	172 3001 z
011	4021	HT	103	4100	C	173 5000 {
012	1021	LF	104	4040	D	174 6000
013	4103	VT	105	4020	E	175 3000 }
014	4043	FF	106	4010	F	176 3400 ~
015	4023	CR	107	4004	G	177 4005 DEL
016	4013	SO	110	4002	H	200 3403 K0
017	4007	SI	111	4001	I	201 1401 K1
020	6403	DLE	112	2400	J	202 1201 K2
021	2401	DC1	113	2200	K	203 1101 K3
022	2201	DC2	114	2100	L	204 1041 K4
023	2101	DC3	115	2040	M	205 2021 K5
024	0043	DC4	116	2020	N	206 4011 K6
025	0023	NAC	117	2010	O	207 2005 K7
026	0201	SYN	120	2004	P	210 1003 K8
027	1011	ETB	121	2002	Q	211 1403 K9
030	2003	CAN	122	2001	R	212 1203 K10
031	2403	EM	123	1200	S	213 1103 K11
032	0007	SUB	124	1100	T	214 1043 K12
033	1005	ESC	125	1040	U	215 4403 K13
034	2043	FS	126	1020	V	216 4203 K14
035	2023	GS	127	1010	W	217 2103 K15
036	2013	RS	130	1000	X	220 7403 K16
037	2007	US	131	1002	Y	221 0401 K17
040	0000	SP	132	1001	Z	222 2203 K18
041	4006	!	133	4202	[	223 0101 K19
042	0006	"	134	1202	\	224 0041 K20
043	0102	#	135	2202	]	225 0021 K21
044	2102	\$	136	2006	^	226 0011 K22
045	1042	%	137	1022	_	227 4003 K23
046	4000	&	140	0402	`	230 0003 K24
047	0022	'	141	5400	a	231 0403 K25
050	4022	(	142	5200	b	232 0203 K26
051	2022	)	143	5100	c	233 0103 K27
052	2042	*	144	5040	d	234 4041 K28
053	4012	+	145	5020	e	235 2041 K29
054	1102	,	146	5010	f	236 0013 K30
055	2000	-	147	5004	g	237 3401 K31
056	4102	.	150	5002	h	240 5401 N0
057	1400	/	151	5001	i	241 5201 N1
060	1000	0	152	6400	j	242 5101 N2
061	0400	1	153	6200	k	243 5041 N3
062	0200	2	154	6100	l	244 5021 N4
063	0100	3	155	6040	m	245 5011 N5
064	0040	4	156	6020	n	246 5005 N6
065	0020	5	157	6010	o	247 5003 N7
066	0010	6	160	6004	p	250 4402 N8
067	0004	7	161	6002	q	251 6401 N9
070	0002	8				252 6201 N10
071	0001	9				253 6101 N11
						254 6041 N12

11

BYTE POSITION				BYTE POSITION			
CHAR	Left	Right	Dec.	CHAR	Left	Right	Dec.
A	040400	000101	65	NUL	000000	000000	0
B	041000	000102	66	SOH	000400	000001	1
C	041400	000103	67	STX	001000	000002	2
D	042000	000104	68	ETX	001400	000003	3
E	042400	000105	69	EOT	002000	000004	4
F	043000	000106	70	ENO	002400	000005	5
G	043400	000107	71	ACK	003000	000006	6
H	044000	000110	72	BEL	003400	000007	7
I	044400	000111	73	BS	004000	000010	8
J	045000	000112	74	HT	004400	000011	9
K	045400	000113	75	LF	005000	000012	10
L	046000	000114	76	VT	005400	000013	11
M	046400	000115	77	FF	006000	000014	12
N	047000	000116	78	CR	006400	000015	13
O	047400	000117	79	SO	007000	000016	14
P	050000	000120	80	SI	007400	000017	15
Q	050400	000121	81	DLE	010000	000020	16
R	051000	000122	82	DC1	010400	000021	17
S	051400	000123	83	DC2	011000	000022	18
T	052000	000124	84	DC3	011400	000023	19
U	052400	000125	85	DC4	012000	000024	20
V	053000	000126	86	NAK	012400	000025	21
W	053400	000127	87	SYN	013000	000026	22
X	054000	000130	88	ETB	013400	000027	23
Y	054400	000131	89	CAN	014000	000030	24
Z	055000	000132	90	EM	014400	000031	25
				SUB	015000	000032	26
a	060400	000141	97	ESC	015400	000033	27
b	061000	000142	98	FS	016000	000034	28
c	061400	000143	99	GS	016400	000035	29
d	062000	000144	100	RS	017000	000036	30
e	062400	000145	101	US	017400	000037	31
f	063000	000146	102	SPACE	020000	000040	32
g	063400	000147	103	!	020400	000041	33
h	064000	000150	104	"	021000	000042	34
i	064400	000151	105	#	021400	000043	35
j	065000	000152	106	\$	022000	000044	36
k	065400	000153	107	%	022400	000045	37
l	066000	000154	108	&	023000	000046	38
m	066400	000155	109	'	023400	000047	39
n	067000	000156	110	(	024000	000050	40
o	067400	000157	111	)	024400	000051	41
p	070000	000160	112	*	025000	000052	42
q	070400	000161	113	+	025400	000053	43
r	071000	000162	114	,	026000	000054	44
s	071400	000163	115	-	026400	000055	45
t	072000	000164	116	.	027000	000056	46
u	072400	000165	117	/	027400	000057	47
v	073000	000166	118	:	035000	000072	58
w	073400	000167	119	;	035400	000073	59
x	074000	000170	120	<	036000	000074	60
y	074400	000171	121	=	036400	000075	61
z	075000	000172	122	>	037000	000076	62
				?	037400	000077	63
0	030000	000060	48	@	040000	000100	64
1	030400	000061	49	[	055400	000133	91
2	031000	000062	50	\	056000	000134	92
3	031400	000063	51	]	056400	000135	93
4	032000	000064	52	^	057000	000136	94
5	032400	000065	53	_	057400	000137	95
6	033000	000066	54	`	060000	000140	96
7	033400	000067	55	{	075400	000173	123
8	034000	000070	56	}	076000	000174	124
9	034400	000071	57	~	076400	000175	125
				DEL	077000	000176	126
					077400	000177	127

MPE USER OPERATION

JUNE 1973

U-1

STANDARD CAPABILITY COMMANDS

Command	Parameters	Function	When Issued						
:ABORT		Aborts the current program.	B						
:BUILD	<i>[reference]</i> <i>[,DEV=device]</i> <i>[,DISC=[filesize] [, [numextents] [, initialloc]]]</i> <i>[,REC=[recsize] [, [blockfactor] [<table style="display: inline-table; vertical-align: middle;"><tr><td>F</td><td>[.BINARY]</td></tr><tr><td>U</td><td>[.ASCII]</td></tr><tr><td>V</td><td></td></tr></table>]]]</i> <i>[,NOLABEL]</i> <i>[,CCTL]</i> <i>[,TEMP]</i> <i>[,CODE=filcode]</i>	F	[.BINARY]	U	[.ASCII]	V		Creates a new file.	J,S,B
F	[.BINARY]								
U	[.ASCII]								
V									
:BASIC	<i>[command:file]</i> <i>[, [inputfile]]</i> <i>[, [listfile]]</i>	Calls BASIC/3000 interpreter.	J,S						
:BYE		Terminates a session.	S						
:CONTINUE		Disregards job-error condition.	J						
:DATA	<i>[jobname], username[/upass] acctname[/apass] [, [filename]]</i>	Defines data from outside standard input stream.	J,S,B						
:EDITOR	<i>[listfile]</i>	Calls the EDITOR.	J,S						
:EOD		Denotes end of data.	J,S						
:EOJ		Denotes end of batch job.	J						
:FILE	(This command has many parameters that can be combined in four different formats; the user is referred to Section V.)	Defines or redefines a file's characteristics.	J,S,B						
:FORTRAN	<i>[sourcefile] [, [ustfile] [, [listfile] [, [masterfile] [, .ewfile]]]]</i>	Compiles a FORTRAN program.	J,S						
:FREERIN	<i>rin</i>	Deallocates a global RIN, and returns it to RIN pool.	J,S						
:GETRIN	<i>[rinpassword]</i>	Acquires a global RIN.	J,S,B						
:HELLO	<i>[sessionname], username[/upass] acctname[/apass] [, group[/gpass]] [, TERM=termtype] [, Pkt=executionpriority]</i>	Initiates a session.	S						
:JOB	<i>[jobname], username[/upass] acctname[/apass] [, groupname [/gpass]] [, TERM=termtype] [, PRI=executionpriority] [, OUTCLASS=outputclass]</i>	Initiates a batch job.	J						

:LISTF	<i>[fileset] [,detail] [,listfile]</i>	Lists descriptions of filesets.	J,S,B
:PREP	<i>uslfile,progfile</i> <i>[;ZERODD]</i> <i>[;PMAP]</i> <i>[;MAXDATA=segsize]</i> <i>[;STACK=stacksize]</i> <i>[;DL=dlsize]</i> <i>[;CAP=caplist]</i> <i>[;RL=filename]</i>	Prepares a compiled program into segmented form.	J,S
:PTAPE	<i>[filename]</i>	Reads a paper tape without X-OFF control.	S,B
:PURGE	<i>[filereference] [,TEMP]</i>	Deletes a file from the system.	J,S,B
:RENAME	<i>oldfilereference,newfilereference</i> <i>[,TEMP]</i>	Renames a file.	J,S,B
:RESTORE	<i>storedset[,filesetlist]</i> <i>[,dec] [,KEEP]]]</i>	Restores a complete fileset, stored off-line.	J,S,B
:RESUME		Resumes an interrupted program.	B
:RESET	<i>formaldesignator</i>	Resets a formal file designator.	J,S,B
:RUN	<i>progfile[,entrypoint]</i> <i>[;NOPRIV]</i> <i>[;LMAP]</i> <i>[;MAXDATA=segsize]</i> <i>[;PARAM=parameterum]</i> <i>[;STACK=stacksize]</i> <i>[;DL=dlsize]</i> <i>[;LIB=library]</i>	Loads and executes a program.	J,S
:SAVE	<i>{ [\$OLDPASS,newfilereference]</i> <i>[tempfilereference]</i>	Changes a file to permanent status.	J,S,B
:SEGMENTER	<i>[listfile]</i>	Calls MPE/3000 Segmenter	J,S
:SHOWTIME		Displays current date and time-of-day.	J,S,B
:SPEED	<i>[inspeed] [,outspeed]</i>	Changes input speed or output speed of terminal.	S
:SPL	<i>[sourcefile] [,uslfile]</i> <i>[,listfile] [,masterfile]</i> <i>[,newfile]]]</i>	Compiles an SPL/3000 program.	J,S
:STAR	<i>[listfile] [,NOLIST]</i>	Calls the Statistical Analysis Routines.	J,S
:STORE	<i>[filesetlist] ;destfile</i>	Stores a set of files off line.	J,S,B
:TTLLOP	<i>message</i>	Transmits a message from the user to the computer operator.	J,S,B

For old files:

```

:FILE formaldesignator      { - $OLDPASS
                             [ - filereference ] [ .OLD
                             [ .OLDTEMP ] }

    { :DEV = device }

    { :NOBUF }
    { :CTL }
    { :MR }

    [ :EXC ]
    [ :EAR ]
    [ :SHR ]

    { :ACC = accesstype }
    { :BUF = numbuffers }

    [ :DEL ]
    [ :SAVE ]
    [ :TEMP ]

    { :CODE = filecode }

```

For user pre-defined files:

```

:FILE formaldesignator = *formaldesignatori

```

For system-defined files:

```

:FILE formaldesignator      = $NULL

:FILE formaldesignator      = { $STDIN
                             $STDINX
                             $STDLIST }

    { :BUF = numbuffers }

    { :NOBUF }
    { :CTL }
    { :MR }

    [ :EXC ]
    [ :EAR ]
    [ :SHR ]

```

Command	Syntax	Use
ADD	ADD[Q] [<i>line number</i>]. [,HOLD[Q] [,NOW]]	To enter lines of text into the Text file from the keyboard or from the Hold file.
BEGIN	BEGIN[Q]	Used as the first expression in a matching BEGIN-END pair.
CHANGE	CHANGE[Q] [<i>abs. col. pos./abs. col. pos.</i>] [<i>char. string</i> TO <i>char. string</i> [IN <i>range list</i>]	To replace old text with new text.
DELETE	DELETE[Q] [<i>range list</i>]	To delete characters and lines from the Text file.
END	END	To terminate execution of EDIT/3000 or, when used with a matching BEGIN command, to terminate an iterative operation.
FIND	FIND[Q] <i>range</i>	To locate a point in the Text file.
GATHER	GATHER[Q] <i>range</i> TO <i>line number</i> [BY <i>increment</i>]	To move portions of text from one location to another in the Text file and renumber the lines.
HOLD	HOLD[Q] [<i>range</i>] [,APPEND]	To copy text from the Text file into the Hold file.
INSERT	INSERT[Q] <i>position</i> [BY <i>increment</i>] [,HOLD[Q] [,NOW]]	To insert text into the Text file from the terminal and the Hold file.
JOIN	JOIN[Q] <i>file name</i> [<i>line number/line number</i>] [(# <i>integer</i> /n <i>integer</i>).] [TO <i>line number</i>] [BY <i>increment</i>]	To add all or part of a file to the Text file.

JUNE 1973

U-5

Command	Syntax	Use
KEEP	KEEP[Q] <i>file name</i> {(<i>range</i>)} [,UNNUMBERED] [,OFFLINE] [,TRANSLATE] [,NOTEXT]	To save all or part of the Text file in a user file
LIST	LIST[Q] { <i>range</i> } [,UNNUMBERED] [,OFFLINE] [,TRANSLATE] [,NOTEXT]	To print out any portion or all of the Text file.
MODIFY	MODIFY[Q] <i>range list</i>	To modify text in the Text file using three operations: delete (D), insert (I), and replace (R).
NOT	NOT	Causes setting of FLAG to be reversed after the next following expression is interpreted.
OR	OR	Resets FLAG to true if it is false, or to false if FLAG is true.
PROCEDURE	PROCEDURE $\left. \begin{matrix} G \\ P \\ S \end{matrix} \right\}$ <i>pname</i> , $\left. \begin{matrix} G \\ P \\ S \end{matrix} \right\}$, <i>range list</i>	Calls logical procedure stored in a library outside the subsystem.
Q	Q <i>character string</i>	To print a message at the terminal while in Use mode.
REPLACE	REPLACE[Q] <i>range list</i> [,HOLD[Q] [,NOW]]	To replace lines in the Text file.

Command	Syntax	Use
SET	SET[FROM= <i>line number</i>] [DELTA= <i>increment</i>] [LEFT= <i>column position</i>] [RIGHT= <i>column position</i>] [LENGTH= <i>line length</i>] { QUIET } { DISPLAY } { SHORT } { LONG } { BATCH } { POLL } [DEPTH= <i>unsigned integer</i>] [TIME= <i>unsigned integer</i>] [SIZE= <i>unsigned integer</i>] { FRONT } { REAR } { FIXED } { VARIABLE }	To alter options that are normally set by the subsystem and that govern editing operations.
TEXT	TEXT <i>file name</i> { (<i>line number/line number</i>) } { (# <i>integer</i> / # <i>integer</i>) } [UNNUMBERED]	To copy contents of a user file into the Text file to be edited.
USE	USE <i>file name</i>	To instruct the subsystem to read commands from a user file but to send messages to and expect input from the terminal.
VERIFY	VERIFY { ALL } { <i>option list</i> }	To obtain a reminder of the values of options declared in the SET command.
WHILE	WHILE[FLAG]	Causes EDIT/3000 to iterate the next two edit expressions (or one, if FLAG is declared) until the first one fails.
XPLAIN	XPLAIN { <i>com. name list</i> } { ALL } [OFFLINE]	To obtain an explanation of the commands.
YES	YES	Sets the FLAG to true
Z :=	Z := <i>object record</i>	To assign the value of the object record to Z :=.

JUNE 1973

U-7

range list ::= range, range, ...

range ::= $\left[\begin{array}{l} \text{position} \\ \text{position/position} \\ \text{ALL} \\ \text{null} \end{array} \right]$ $\left\{ \begin{array}{l} \text{one particular position} \\ \text{a group of positions} \\ \text{all positions} \\ \text{no positions} \end{array} \right.$

position ::= line position (column position)

line position ::= $\left[\begin{array}{l} \text{line number} \\ \text{"string"} \\ * \\ \text{FIRST} \\ \text{LAST} \end{array} \right] (\pm \text{ number of lines})$ $\left. \begin{array}{l} \text{current} \\ \text{pointer} \\ \text{position} \end{array} \right\}$

column position ::= $\left\{ \begin{array}{l} \text{column number} \\ \text{LEFT} \\ \text{RIGHT} \\ \text{LAST} \\ * \end{array} \right\} \pm \text{ number of columns}$

string ::= Delimit character {character string} delimit character
 'decimal number'

delimit character ::= " ' ! \$ % & : < > = ? @ [] ^ \ _ ' "

Creating New USL's

-BUILDUSL *fname,records,extents*

Designating USL's For Management By The User

-USL *fname*

Activating Entry Points

-USE [*pointspec,*] *name* [(*index*)]

pointspec	Is ENTRY, to activate the entry point indicated by <i>name</i> (<i>index</i>); or UNIT, to activate all entry points in the RBM indicated by (the primary entry point) <i>name</i> (<i>index</i>); or SEGMENT, to activate all entry points in all RBM's associated with the segment <i>name</i> . The default value is ENTRY. (Optional parameter.)
name	The name of the entry point, RBM (referred to by its primary entry-point name), or segment referenced. (Required parameter.)
index	The index integer further specifying the entry point <i>name</i> . The index may be used when the USL contains more than one entry point of this name. If <i>index</i> is omitted, a default value of 0 is assigned. If <i>pointspec</i> is SEGMENT, the <i>index</i> parameter does not apply. (Optional parameter.)

Deactivating Entry Points

-CEASE [*pointspec,*] *name* [(*index*)]

Deleting RBM's

-PURGERBM [*rbmspec,*] *name* [(*index*)]

rbmspec	Is UNIT, to delete the RBM identified by <i>name</i> (<i>index</i>); or SEGMENT, to delete all RBM's associated with the segment <i>name</i> . The default parameter is UNIT. (Optional parameter.)
----------------	---

JUNE 1973

U-9

Assigning New Segment Names to RBM's

-NEWSEG *newsegname ,rbmname*{(*index*)}

Transferring RBM's

-AUXUSL *fname*



The name of the USL file from which the RBM's are to be transferred. (Required parameter.)

Next, the user transfers (copies) the RBM's to the destination USL by entering the **-COPY** command.

-COPY [*rbmspec,* *name*]{(*index*)}

Listing RBM's

-LISTUSL

- 1 The name of the USL file (filename, groupname, and accountname).
- 2 The segment name.
- 3 The RBM (entry-point) name.
- 4 The number of words in the RBM code module.
- 5 The type of entry-point named, where:
 - P = Procedure entry point.
 - SP = Secondary entry point for procedure.
 - OB = Outer block entry point.
 - SO = Secondary entry point for outer block.
 - BD = Block data entry point.
 - I_n = Interrupt procedure entry point (n is the interrupt procedure type number, ranging from 0 through 2).
- 6 The active/inactive indicator, where:
 - A = Active
 - I = Inactive
- 7 The callability indicator, where:
 - C = The entry point is directly callable by the user.
 - U = The entry point is not directly callable by the user.
- 8 The privileged/not-privileged indicator:
 - P = Privileged.
 - N = Not privileged.
- 9 The internal flag status:
 - H = Internal flag on
 - R = Internal flag off.
- 10 The total size of the USL file, in words (octal).
- 11 The number of words (in octal) used in the directory portion of the USL file.
- 12 The number of words (in octal) in the directory part of the USL file, filled with meaningless information, as a result of various RBM manipulations.
- 13 The number of unused words (in octal) in the directory portion of the USL file.
- 14 The number of words (in octal) used in the RBM-information portion of the USL file.
- 15 The number of words (in octal) in the RBM-information portion of the USL file, filled with meaningless information, as a result of various RBM manipulations.
- 16 The number of unused words (in octal) in the RBM-information portion of the USL file.

JUNE 1973

U-11

USL FILE SCR4.PUB.SYS

SEG40 → ②
 LISTRL → ③
 CLEANUPRLXTHUF 351 P A C N R → ④
 CLEANUPRLXTHUF 13 F A C N R → ⑤
 GETNEXTREXTN 34 F A C N R → ⑥
 SETUPRLXTHUF 36 F A C N R → ⑦
 ADDPROC 27 F A C N R → ⑧
 DELETEPROC SF A C N R → ⑨
 FIXUPRL 502 F A C N R
 REMOVERL 72 F A C N R
 INSERTRL 434 F A C N R
 CLEANUPRLHUF 11 F A C N R
 DELETEENTRY 46 F A C N R
 FINDWLSpace 72 F A C N R
 GETNEXTREXTN 44 F A C N R
 SETUPRLHUF 22 F A C N R
 RLNTYPARMS 23 F A C N R
 GETNEXTWELD 20 F A C N R
 SEARCHPL 76 F A C N R
 SAVEDMAP 11 F A C N R
 GETPLMAP 14 F A C N R
 RETURNWLSpace 27 F A C N R
 FINDWLSpace 126 F A C N R
 SEG33
 REMOVESE 106 F A C N R
 BINDSEGS 415 F A C N R
 FIXUPSE 125 F A C N R
 RETURNWLSpace 14 F A C N R
 SEG32
 INSERTNEWSEG 207 F A C N R
 MAKEENTRIES 163 F A C N R
 INSERTSEG 203 F A C N R
 FINDWLSpace 220 F A C N R
 FINDSLRECORD 35 F A C N R
 SEG31
 CLEANUPRLHUF 11 F A C N R
 GETREFTAHENTRY 33 F A C N R

SEG1
 CB 606 OB A C N
 CORRECTCLASS 32 F A C N P

SETACTIVITY 43 P A C N R
 FILE SIZE 400000 → ⑩
 DIR. USED 10001 → ⑪
 DIR. GARB. 0 → ⑫
 DIR. AVAIL. 67577 → ⑬
 INFO USED 40737 → ⑭
 INFO GARB. 0 → ⑮
 INFO AVAIL. 237041 → ⑯

-LISTUSL Command Output

Preparing Program Files

-PREPARE *progfile*

```
{:ZERODB}
{:PMAP}
{:MAXDATA=segsz}
{:STACK=stacksz}
{:DL=dlsz}
{:CAP=caplist}
{:RL=filename}
```

ZERODB	An indication that the initially-defined DL-DB area, and uninitialized portions of the DB-Q (initial) area will be initialized to zero. If this parameter is omitted, these areas are not affected. (Optional parameter.)
PMAP	An indication that a listing describing the prepared program will be produced on the device specified by the :SEGMENTER command parameter <i>listfile</i> . If this parameter is omitted, the prepared program is not listed. (Optional parameter.)
segsz	Maximum stack area (Z-DL) size permitted, in <i>words</i> . This parameter is included if the user expects to change the size of the DL-DB or Z-DB areas during process execution. If omitted, MPE/3000 assumes that he will not change these areas. (Optional parameter.)
stacksz	The size of the user's initial local data area (Z-Q (initial)) in the stack, in <i>words</i> . This overrides the <i>stacksz</i> estimated by the segmenter, which applies if the <i>stacksz</i> parameter is omitted. (The default is a function of estimated stack requirements for each program unit in the program. Since it is difficult for the system to predict the behavior of the stack at run time, the user may want to override the default by supplying his own estimate with <i>stacksz</i> .) (Optional parameter.)
dlsz	The DL-DB area to be initially assigned to the stack. This area is mainly of interest only in programmatic applications, and is discussed in detail in Section 11. If the <i>dlsz</i> parameter is omitted, a value of zero is used. (Optional parameter.)
caplist	The <i>capability-class attributes</i> associated with the user's program; specified as two-character mnemonics. If more than one mnemonic is specified, each must be separated from its neighbor by a comma.

The mnemonics are

```
IA  = Interactive access
BA  = Local batch access
PH  = Process handling
DS  = Data segment management
MR  = Multiple resource management
RT  = High-priority (BS) subqueue access
CR  = Very high-priority (AS) subqueue access
PM  = Privileged-mode operation
NS  = Non-standard subqueue access
```

filename	The name of a relocatable procedure library (RL) to be searched to satisfy external references during preparation. This can be any permanent file of type RL. It need not belong to the log-on group, nor does it have a reserved, local name. This file yields a single segment that is incorporated into the segments of the program file prepared. If <i>filename</i> is omitted, no library is searched. (Optional parameter.)
-----------------	--

JUNE 1973

U-13

Creating a Relocatable Procedure Library (RL) File

-BUILDRL *fname, records, extents*

Designating RL's for Management by the User

-RL *fname*

Adding a Procedure to an RL

-ADDRL *name[(index)]*

Deleting an Entry-Point or Procedure from an RL

-PURGERL [*rlspec,*] *name*

rlspec Is UNIT, to delete the procedure identified by *name*; or

ENTRY, to delete the entry point identified by *name*.

The default parameter is ENTRY. (Optional parameter.)

name If *rlspec* is UNIT, this is the name of the procedure to be deleted. If
rlspec is ENTRY, this is the name of the entry point to be deleted.
(Required parameter.)

Listing Procedures in an RL

-LISTRL

Creating a Segmented Procedure Library (SL) File

-BUILDSL *fname, records, extents*

fname	Any file whose local name is SL, in the <i>filereference</i> format. (Required parameter.)
records	The total maximum file capacity, specified in terms of 128-word binary logical records. (Required parameter.)
extents	The total number of disc extents that can be dynamically-allocated to the file as logical records are written to it. The size of each extent is determined by the <i>records</i> parameter value divided by the <i>extents</i> parameter value. The <i>extents</i> value must be an integer from 1 to 16. (Required parameter.)

Designating SL's for Management by the User

-SL *fname*

Adding a Procedure Segment to an SL

-ADDSL *name[.PMAP]*

Deleting an Entry-Point or Procedure From a Segment in an SL

-PURGFSL '[*unitspec*,] *name*

unitspec	Is ENTRY, to delete the entry-point identified by <i>name</i> ; or SEGMENT, to delete the segment identified by <i>name</i> . (Optional parameter.) The default parameter is ENTRY.
name	The name of the entry-point or segment to be deleted. (Required parameter.)

Listing Procedures in an SL

-LISTSL

JUNE 1973

U-15

SETTING RBM INTERNAL FLAGS

In addition to the activity bit associated with every entry-point on an RBM, there is also an *internal flag* that determines whether or not that entry-point is placed in the library directory — and thus made available to users and segments within that SL — when the RBM is segmented and added to an SL. (The internal flag is *not* interrogated until the segment is actually added.) As an example of how this facility can be used, suppose that a programmer is writing a complicated utility routine that he wants to add to an SL. He writes this routine as a set of procedures to be placed in the same code segment, but wants only the entry-points of certain procedures made available to other users; he does not want them to use the entry-points of the support procedures for the main routine. He can effectively hide these private entry-points from users by using the `OPTION INTERNAL` statement in the declarations of the applicable procedures, or by setting the internal flag *on* with the `-HIDE` command described below. When these procedures are prepared onto a segment and added to an SL, the entry-points designated as internal are not entered in the directory of the library.

To set an internal flag *on*, enter:

`-HIDE name[(index)]`

name The name of the entry-point in the currently-managed USL whose internal flag is to be set *on*. (Required parameter.)

index The index integer further specifying the entry-point *name*; the index may be used when the RBM contains more than one entry-point of this name. If *index* is omitted, 0 is assigned by default. (Optional parameter.)

To set an internal flag to *off*, enter the following command:

`-REVEAL name[(index)]`

name The name of the entry-point in the currently-managed USL whose internal flag is to be set *off*. (Required parameter.)

index The index integer further specifying the entry-point *name*; the index may be used when the RBM contains more than one entry-point of this name. If *index* is omitted, a value of 0 is assigned. (Optional parameter.)

USING EXTERNAL PROCEDURE LIBRARIES

As part of its file domain, each group within an account may optionally include files comprising *procedure libraries*. These libraries contain external procedures frequently called by programmers accessing this group. They permit several users to concurrently access common routines with efficiency.

Under MPE/3000, there are two types of libraries: Relocatable Libraries and Segmented Libraries. Each group optionally may have one or more Relocatable Libraries and one Segmented Library.

Relocatable Libraries

Relocatable libraries (RL's), are specially-formatted files that are searched at *program preparation* time to satisfy references to external procedures required by the user's program. Within such libraries, these procedures exist in REM form (as they would on a USL). When a program is prepared, these procedures are placed in a single segment and linked to the user's program in the resulting program file. Since a segment containing procedures from an RL is specifically constructed for a given program, any such segment cannot be shared concurrently by different programs. Instead, individual copies of the required procedures must be made and segmented for each program requesting them.

An RL is always a permanent file, but it need not belong to the user's log-on group nor have a reserved local file name.

The RL to be searched during preparation of the user's program is specified by the *filename* parameter of the :PREP or -PREPARE command. For this parameter, the *filereference* format introduced in Section V is used.

Segmented Libraries

Segmented Libraries (SL's) are specially-formatted files that are searched at *program-allocation* time to satisfy references to external procedures. These libraries, like program files, contain procedures in segmented (prepared) form. An individual procedure may be the only procedure in its segment, or may exist in a segment containing many other procedures. When a procedure is referenced, the segment containing it is loaded along with other segments referenced by that procedure. Since the segmentation is not altered when different programs reference procedures in an SL, these procedures may be shared concurrently by many programs.

From one to three SL's may be searched during allocation of the user's program. These are

1. The Group Library (GSL), which is actually the library of the group under which the program file is stored. This library is readable by any user who can access this group.
2. The account's Public Library (PSL), which is the library of the public group of the account under which the program file is stored. This library is readable by any user who can access this account.
3. The System Library (SSL), which is the library of the public group of the system account. This library can be accessed by all users of the system.

The order in which the libraries are searched is specified by the LIB parameter in the :RUN command that requests program allocation/execution, as follows:

LIB Parameter	Specification
LIB = G	The libraries are searched in this order: • Group Library • Public Library • System Library
LIB = P	The libraries are searched in this order: • Public Library • System Library
LIB = S	Only the System Library is searched.

When no LIB parameter is specified, the default is LIB = S.

JUNE 1973

U-17

As an example of how SL's are used, procedures called frequently by many users throughout the system are stored in an SSL. Those used frequently throughout an account can be stored in an PSL. Those used by only a few users in a single group, less frequently, might be stored in a GSL.

Because the global area of the user's stack is already formed at the time a user's program references an external procedure in an SL, the library procedures cannot contain global variables. From the user's standpoint, this means that SPL/3000 procedures in a library cannot contain EXTERNAL or OWN variables. Also, FORTRAN procedures in a library cannot contain: DATA, COMMON, LABELED COMMON, or TRACE variables; FORMAT statements; or references to LOGICAL UNITS.

Within an account, the name of any public group is always PUB; the name of the segmented library is always SL. Thus, when referencing a library file in an MPE/3000 command or intrinsic, the following rules apply.

1. If the user's program file is a permanent file, with the name X.Y.Z (in the filereference format), then:

GSL is referenced as SL.Y.Z

PSL is referenced as SL.PUB.Z

SSL is referenced as SL.PUB.SYS

If Y.Z = PUB.SYS, then only LIB = S is a legal entry in the :RUN command.

If Y.Z = PUB.Z, then LIB = P or LIB = S are legal entries in the :RUN command.

During allocation, the proper libraries are determined by the position of the program file in the file system directory.

2. If the user's program file is not a permanent file, but has the name X.Y.Z, where Y is the log-on group and Z is the log-on account, or is nameless (such as the file SOLDPASS) then:

GSL = SL.Y.Z

PSL = SL.PUB.Z

SSL = SL.PUB.SYS.

In the above cases, LIB = G, LIB = P, or LIB = S are all legal entries in the :RUN command.

During allocation, the proper libraries are determined by the log-on group and account.

For new files:

```

;FILE formaldesignator      {  = $NEWPASS
                             [ = filereference][.NEW]  }

[.DEV = device]

[.DISC = [filesize] [, [numextents] [, initialloc]]]

[.REC = [resize] [[. [blockfactor] [[ [F
                                     U
                                     V] [.BINARY
                                     .ASCII] ]]]]]

[.NOBUF]
[.CCTL]
[.MR]

[.EXC]
[.EAR]
[.SHR]

[.ACC = accesstype]
[.BUF = numbuffers]

[.DEL]
[.SAVE]
[.TEMP]

[.CODE = filecode]
```

JUNE 1973

U-19

CUSTOMER	SERIAL NUMBER
LOCATION	SITE TELEPHONE
CONTACTS	SYSTEM TELEPHONE

SITE TELEPHONE

Dev. No.	MUX- Chan	SR	Intrp Mask	Intrp Priority	Date
-------------	--------------	----	---------------	-------------------	------

[illegible]

[illegible]