HEWLETT PACKARD

HP 1000 A900 Computer

Reference Manual

HP 1000 A-Series

HP 1000 A900 Computer

Reference Manual

FEDERAL COMMUNICATIONS COMMISSION
RADIO FREQUENCY INTERFERENCE
STATEMENT

The Federal Communications Commission (in Subpart J, of Part 15, Docket 20780) has specified that the following notice be brought to the attention of the users of this product.

Warning: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

MANUAL PART NO. 02139-90001
Printed in U.S.A. May 1984
E0584

PRINTING HISTORY

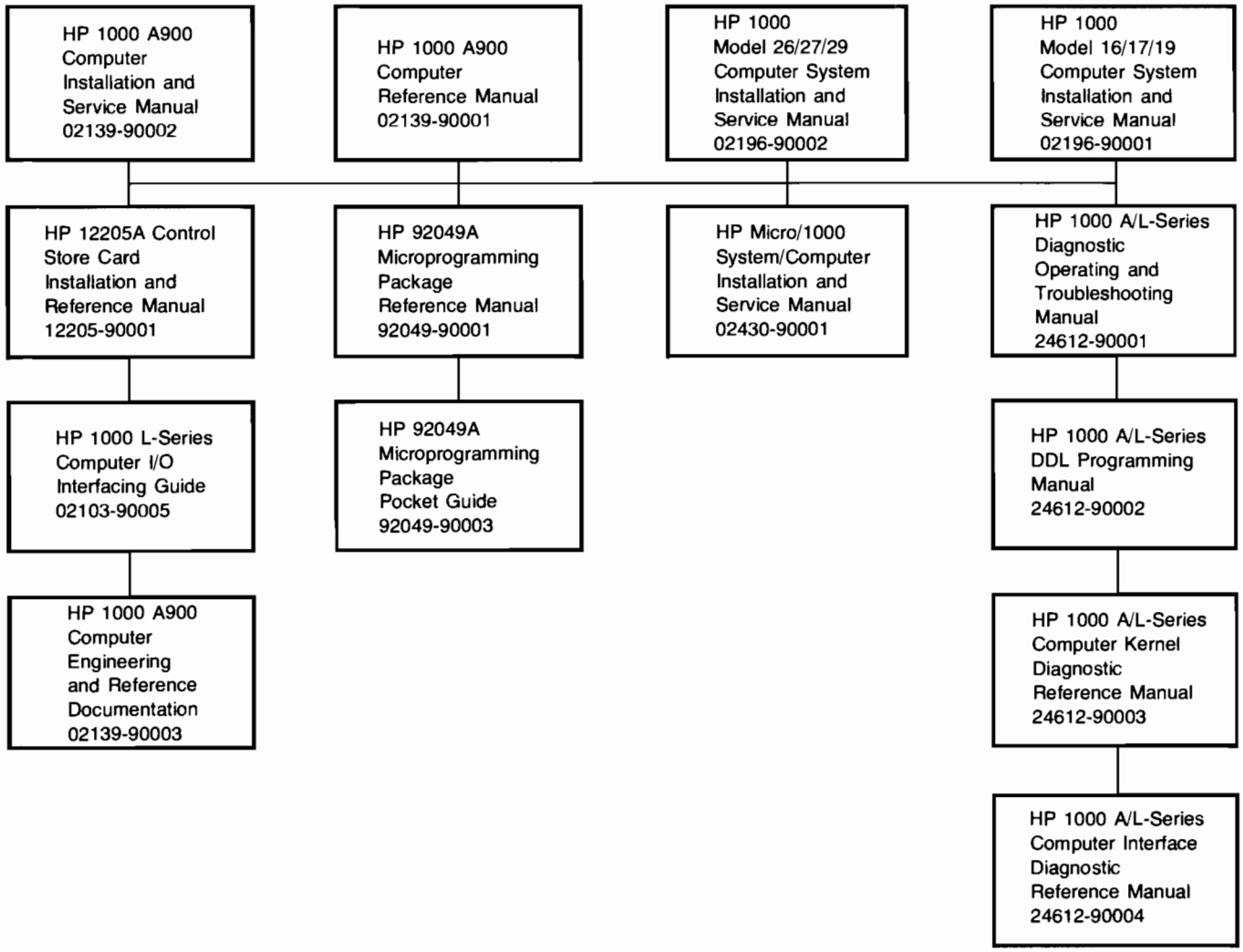
The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what software manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

First Edition	Dec 1982	
Update 1	Dec 1982	
Second Edition	Jun 1983	
Update 1	Dec 1983	
Update 2	May 1984	
Reprint	May 1984	Updates 1 and 2 have been incorporated.

DOCUMENTATION MAP



CONTENTS

Section I — Page

GENERAL FEATURES

Architecture ... 1-1
Floating Point Hardware ... 1-1
User Microprogramming ... 1-2
Virtual Control Panel ... 1-2
Bootstrap Loaders ... 1-2
Self-Test Routines ... 1-2
Time Base Generator ... 1-2
Power Supply ... 1-2
Input/Output ... 1-2
Memory ... 1-3
Cache Memory ... 1-3
Software ... 1-4
HP Interface Bus ... 1-4
Computer Network ... 1-4
Expansion and Enhancement ... 1-4
Specifications ... 1-4

Section II — Page

OPERATING FEATURES

Hardware Registers ... 2-1
- A-Register ... 2-1
- B-Register ... 2-1
- P-Register ... 2-1
- Extend (E) Register ... 2-1
- Overflow (O) Register ... 2-1
- Central Interrupt Register ... 2-1
- Violation Register ... 2-1
- Parity Violation Register ... 2-1
- Interrupt System Register ... 2-1
- X- and Y-Registers ... 2-1
- WMAP Register ... 2-1
- IMAP Register ... 2-2
- C- and Q-Registers ... 2-2
- Z-Register ... 2-2
- IQ-Register ... 2-2

Virtual Registers ... 2-2
- M-Register ... 2-2
- T-Register ... 2-2

Controls and Indicators ... 2-2
Self-Test ... 2-2
Bootstrap Loaders ... 2-3
- Loader Selection for Auto-Boot ... 2-3
- Program Starts ... 2-3
- VCP Re-entry for Extended Boot Loading ... 2-4
- Device Parameters and Media Formats ... 2-4

Virtual Control Panel ... 2-4
- VCP Program Operation ... 2-4
- Loader Commands ... 2-6
- VCP User Considerations ... 2-6
- VCP Slave Functions ... 2-6

Section III — Page

PROGRAMMING INFORMATION

Data Formats ... 3-1
Addressing ... 3-1
- Paging ... 3-1
- Direct and Indirect Addressing ... 3-1
- Memory Mapping ... 3-3
- Virtual Memory Area ... 3-3
- Code and Data Separation ... 3-3
- Base-Relative Addressing ... 3-3
- Reserved Memory Locations ... 3-3
- Nonexistent Memory ... 3-4

Base Set Instruction Formats ... 3-4
- Memory Reference Instructions ... 3-4
- Register Reference Instructions ... 3-5
- Input/Output Instructions ... 3-4
- Extended Arithmetic Memory Reference Instructions ... 3-5
- Extended Arithmetic Register Reference Instructions ... 3-5
- Extended Instructions ... 3-5
- Floating Point Instructions ... 3-5
- Language Instruction Set ... 3-5
- Double Integer Instructions ... 3-5
- Virtual Memory Instructions ... 3-6
- Operating System Instructions ... 3-6
- Scientific Instruction Set ... 3-6
- Vector Instruction Set ... 3-6
- CDS Instructions ... 3-6

Base Set Instruction Coding ... 3-6
- Memory Reference Instructions ... 3-6
- Register Reference Instructions ... 3-8
  - Shift/Rotate Group ... 3-8
  - Alter/Skip Group ... 3-11
- Input/Output Instructions ... 3-13
- Extended Arithmetic Memory Reference Instructions ... 3-15
- Extended Arithmetic Register Reference Instructions ... 3-16
- Extended Instruction Group ... 3-18
  - Index Register Instructions ... 3-18
  - Jump Instructions ... 3-21
  - Byte Manipulation Instructions ... 3-22
  - Bit Manipulation Instructions ... 3-23
  - Word Manipulation Instructions ... 3-24
- Floating Point Instructions ... 3-25
  - Single Precision Operations ... 3-25
  - Double Precision Operations ... 3-26
- Language Instruction Set ... 3-28
- Double Integer Instructions ... 3-31
- Virtual Memory Instructions ... 3-33
- Operating System Instruction Set ... 3-35
- Execution Times ... 3-35
- Scientific Instruction Set ... 3-35
- SIS Execution Times and Interrupts ... 3-40

Vector Instruction Set ... 3-40
- VIS Execution Times and Interrupts ... 3-47

Assembly Language ... 3-47
RTE Implementation ... 3-47

Section IV — Page

DYNAMIC MAPPING SYSTEM

Memory Addressing ... 4-1

CONTENTS (Continued)

General Descriptions 4-1
Page Mapping Register Instructions 4-1
Working Map Instructions 4-1
Cross-Map Instructions 4-2
Detailed Descriptions 4-3
DMS Instruction Execution Times 4-10
Assembly Language and RTE Implementation 4-10

Section V Page
CODE AND DATA SEPARATION
Code and Data Addressing 5-1
General Description 5-1
Procedure Call Instructions 5-1
Procedure Exit Instructions 5-2
C, Q, Z, and IQ Instructions 5-2
Stack Frame Description 5-2
Detailed Descriptions 5-4
Assembly Language and RTE Implementation 5-10
Execute Times 5-10

Section VI Page
INTERRUPT SYSTEM
Power Fail Interrupt 6-1
Multiple-Bit Error Interrupt 6-3
Memory Protect Interrupt 6-3
Unimplemented Instruction Interrupt 6-4
Time Base Generator Interrupt 6-4
Virtual Memory Area Interrupt 6-4
CDS Segment Interrupt 6-4
Input/Output Interrupt 6-5
Interrupt Priority 6-5
Central Interrupt Register 6-5
Processor Status Register 6-5
Interrupt Type Control 6-5
Instruction Summary 6-5

Section VII Page
INPUT/OUTPUT SYSTEM
Input/Output Addressing 7-1
Input/Output Priority 7-1
Interface Elements 7-4
Global Register 7-4
Control Bits 7-4
Flag Bits 7-4
Data Buffer Register 7-4
Control Register 7-4
Direct Memory Access 7-4
Control Word 1 7-5
Control Word 2 7-5
Control Word 3 7-5
DMA Transfer Initialization 7-5
Self-Configured DMA 7-5
DMA Data Transfer 7-7
Non-DMA Data Transfer 7-7
Input Data Transfer (Interrupt Method) 7-7
Output Data Transfer (Interrupt Method) 7-8
Non-Interrupt Data Transfer 7-9
Input 7-10
Output 7-10
Diagnose Modes 7-10
Diagnose Mode 1 7-10
Diagnose Mode 2 7-10
Diagnose Mode 3 7-10

Section VIII Page
MICROPROGRAMMING
The Microprogrammed Computer 8-1
The Microprogrammable Computer 8-1
Customized Instructions 8-1
System Speed 8-1
Memory Space and Security 8-1
Developing Microprograms 8-2
Support for the Microprogrammer 8-2
FPP Microprogramming 8-2
Conclusion 8-2

Appendix Page
Character Codes A-3
Octal Arithmetic A-4
Octal/Decimal Conversions A-5
Mathematical Equivalents A-6
Octal Combining Tables A-8
Instruction Codes in Octal A-9
Base Set Instruction Codes in Binary A-11
Extend and Overflow Examples A-16
Interrupt and Control Summary A-17

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

ILLUSTRATIONS

Title	Page
HP 1000 A900 Computers	1-0
A900 Computer Simplified Block Diagram	1-1
Loading Device Parameters and Media Formats	2-7
Loader Command Format	2-10
Data Formats and Octal Notation	3-2
Base Set Instruction Formats	3-4
Shift and Rotate Functions	3-9
Examples of Double-Word Shifts and Rotates	3-17
Basic Logical Memory Addressing Scheme	4-1
Expanded Memory Addressing Scheme	4-2
Stack Frame General Layout	5-3
Input/Output System	7-2
I/O Priority Assignments	7-2
Priority Linkage (Simplified)	7-3
Interrupt Sequence	7-3
General Bit Definitions for Control Word 1	7-6
DMA Input Data Transfer	7-7
Input Data Transfer (Interrupt Method)	7-8
Output Data Transfer (Interrupt Method)	7-9
Microprogramming Implementation Process	8-3

TABLES

Title	Page
Options and Accessories	1-5
Specifications	1-6
Start-Up Switch Settings	2-3
Sample VCP Loader Call Back Checkout Program	2-4A
VCP Characters and Associated Registers	2-5
VCP Commands	2-6
VCP Loader Command Errors	2-11
Memory Paging	3-3
Reserved Memory Locations	3-3
Shift/Rotate Group Combining Guide	3-9
Alter/Skip GroupCombining Guide	3-11
Typical Base Set Instruction Execution Times	3-36
SIS Instruction Error Codes	3-38
Instructions and Opcodes for RTE Implementation	3-48
Dynamic Mapping Instructions Execution Times	4-11
CDS Instruction Execution Times	5-10
A900 Interrupt Assignments	6-1
Sample Power Fail Subroutine	6-2
Instructions for Select Codes 00 through 07	6-6
Noninterrupt Transfer Routines	7-9
Diagnose Mode 1	7-10
Diagnose Mode 2	7-11

ALPHABETICAL INDEX OF STANDARD INSTRUCTIONS

INSTRUCTION		PAGE
ADA	Add to A	3-6
ADB	Add to B	3-6
ADQA	Add Q to A	5-9
ADQB	Add Q to B	5-9
ADX	Add Memory to X	3-18
ADY	Add Memory to Y	3-18
ALF	Rotate A Left Four	3-8
ALOG	Natural Logarithm	3-38
ALOGT	Common Loagrithm	3-39
ALR	A Left Shift, Clear Sign	3-9
ALS	A left Shift	3-9
AND	"And" to A	3-7
ARS	A Right Shift	3-9
ASL	Arithmetic Shift Left	3-16
ASR	Arithmetic Shift Right	3-16
ATAN	Arctangent	3-38
BLF	Rotate B Left Four	3-9
BLR	B Left Shift, Clear Sign	3-10
BLS	B Left Shift	3-10
BRS	B Right Shift	3-10
CACQ	Copy A to C and Q	5-8
CAX	Copy A to X	3-18
CAY	Copy A to Y	3-18
CAZ	Copy A to Z	5-8
CBCQ	Copy B to C and Q	5-8
CBS	Clear Bits	3-24
CBT	Compare Bytes	3-22
CBX	Copy B to X	3-18
CBY	Copy B to Y	3-18
CBZ	Copy B to Z	5-8
CCA	Clear and Complement A	3-11
CCB	Clear and Complement B	3-11
CCE	Clear and Complement E	3-12
CCQA	Copy C and Q to A	5-8
CCQB	Copy C and Q to B	5-8
CIQA	Copy Interrupted to A	5-9
CIQB	Copy Interrupted to B	5-9
CLA	Clear A	3-12
CLB	Clear B	3-12
CLC	Clear Control	3-13
CLE	Clear E	3-10, 3-12
CLF	Clear Flag	3-13
CLO	Clear Overflow	3-14
CMA	Complement A	3-12
CMB	Complement B	3-12
CME	Complement E	3-12
CMW	Compare Words	3-24
COS	Cosine	3-38
CPA	Compare to A	3-7
CPB	Compare to B	3-7
CXA	Copy X to A	3-18
CXB	Copy X to B	3-19
CYA	Copy Y to A	3-19
CYB	Copy Y to B	3-19
CZA	Copy Z to A	5-9
CZB	Copy Z to B	5-9
DIV	Divide	3-15
DLD	Double Load	3-15
DPOLY	Polynomial Evaluation	3-39
DST	Double Store	3-16
DSX	Decrement X and Skip if Zero	3-19
DSY	Decrement Y and Skip if Zero	3-19
DVABS	Vector Absolute Value	3-44
DVADD	Vector Add	3-41
DVDIV	Vector Divide	3-42
DVDOT	Vector Dot Product	3-45
DVMAB	Vector Maximum Absolute Value	3-45
DVMAX	Vector Maximum Value	3-45
DVMIB	Vector Minimum Absolute Value	3-46
DVMIN	Vector Minimum Value	3-46
DVMOV	Vector Move	3-46
DVMPY	Vector Multiply	3-42
DVNRM	Vector Norm	3-44
DVPIV	Vector Pivot	3-44
DVSAD	Scalar-Vector Add	3-42
DVSDV	Scalar-Vector Divide	3-43
DVSMY	Scalar-Vector Multiply	3-43
DVSSB	Scalar-Vector Subtract	3-43
DVSUB	Vector Subtract	3-41
DVSUM	Vector Sum	3-44
DVSWP	Vector Swap	3-47
ELA	Rotate E Left with A	3-10
ELB	Rotate E Left with B	3-10
ERA	Rotate E Right with A	3-10
ERB	Rotate E Right with B	3-10
EXIT	Procedure Exit	5-7
EXIT1	Procedure Exit With One Skip	5-7
EXIT2	Procedure Exit With Two Skips	5-7
EXP	E to the Power X	3-39
FAD	Floating Point Add	3-25
FDV	Floating Point Divide	3-26
FIX	Floating Point to Single Integer	3-26
FLT	Single Integer to Floating Point	3-26
FMP	Floating Point Multiply	3-26
FSB	Floating Point Subtract	3-26
HLT	Halt	3-14
INA	Increment A	3-12
INB	Increment B	3-12
IOR	"Inclusive Or" to A	3-7
ISX	Increment X and Skip if Zero	3-19
ISY	Increment Y and Skip if Zero	3-19
ISZ	Increment and Skip if Zero	3-7
JLA	Jump and Load A	3-22
JLB	Jump and Load B	3-22
JLY	Jump and Load Y	3-21
JMP	Jump	3-7
JPY	Jump Indexed by Y	3-21
JSB	Jump to Subroutine	3-7
LAX	Load A Indexed by X	3-19
LAY	Load A Indexed by Y	3-19
LBT	Load Byte	3-23
LBX	Load B Indexed by X	3-20
LBY	Load B Indexed by Y	3-20
LDA	Load A	3-8

ALPHABETICAL INDEX OF STANDARD INSTRUCTIONS (Continued)

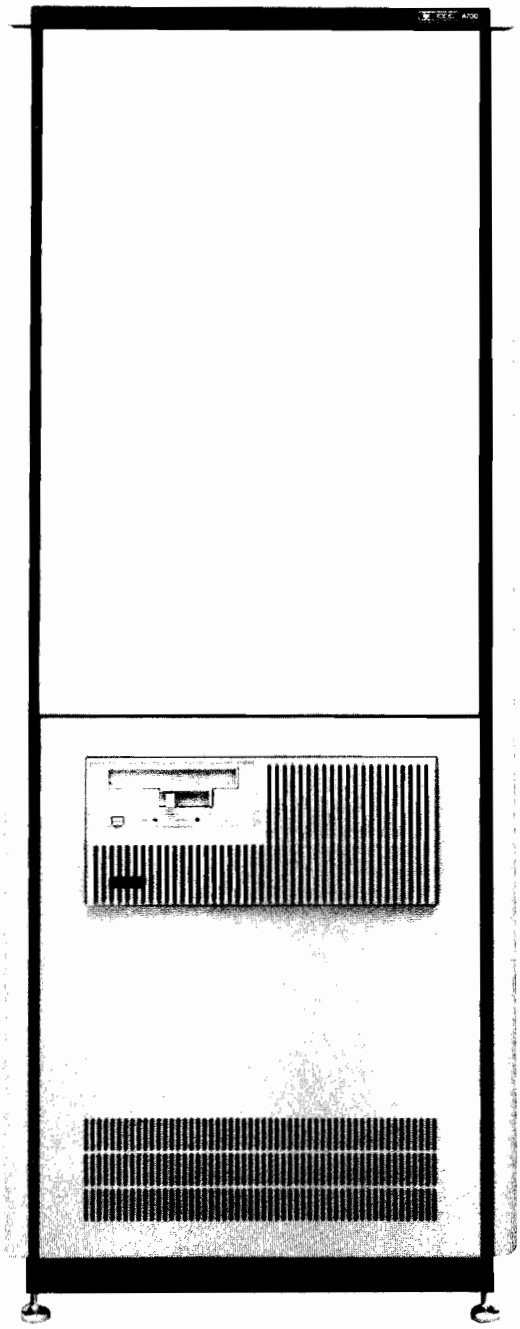
INSTRUCTION		PAGE
LDB	Load B	3-8
LDMP	Load A Map	4-3
LDX	Load X from Memory	3-20
LDY	Load Y from Memory	3-20
LIA	Load Input to A	3-14
LIB	Load Input to B	3-14
LPMR	Load Page Mapping Register	4-3
LSL	Logical Shift Left (32)	3-16
LSR	Logical Shift Right (32)	3-16
LWD1	Load DATA1 Map	4-5
LWD2	Load DATA2 Map	4-5
MB00	Cross Move Bytes, Execute to Execute	4-9
MB01	Cross Move Bytes, Execute to DATA1	4-9
MB02	Cross Move Bytes, Execute to DATA2	4-10
MB10	Cross Move Bytes, DATA1 to Execute	4-10
MB11	Cross Move Bytes, DATA1 to DATA1	4-10
MB12	Cross Move Bytes, DATA1 to DATA2	4-10
MB20	Cross Move Bytes, DATA2 to Execute	4-11
MB21	Cross Move Bytes, DATA2 to DATA1	4-11
MB22	Cross Move Bytes, DATA2 to DATA2	4-11
MBT	Move Bytes	3-23
MIA	Merge Into A	3-14
MIB	Merge Into B	3-14
MPY	Multiply	3-16
MVW	Move Words	3-25
MW00	Cross Move Words, Execute to Execute	4-7
MW01	Cross Move Words, Execute to DATA1	4-7
MW02	Cross Move Words, Execute to DATA2	4-8
MW10	Cross Move Words, DATA1 to Execute	4-8
MW11	Cross Move Words, DATA1 to DATA1	4-8
MW12	Cross Move Words, DATA1 to DATA2	4-8
MW20	Cross Move Words, DATA2 to Execute	4-9
MW21	Cross Move Words, DATA2 to DATA1	4-9
MW22	Cross Move Words, DATA2 to DATA2	4-9
NOP	No Operation	3-10
OTA	Output A	3-14
OTB	Output B	3-14
PCLAI	Internal Procedure Call	5-4
PCALN	Procedure Call, .ENTN Compatible	5-6
PCALR	Procedure Call, .ENTR Compatible	5-6
PCALV	Variable External Procedure Call	5-5
PCALX	External Procedure Call	5-4
RAL	Rotate A Left	3-10
RAR	Rotate A Right	3-11
RBL	Rotate B Left	3-11
RBR	Rotate B Right	3-11
RRL	Rotate Left (32)	3-18
RRR	Rotate Right (32)	3-18
RSS	Reverse Skip Sense	3-12
SAX	Store A Indexed by X	3-20
SAY	Store A Indexed by Y	3-20
SBS	Set Bits	3-24
SBT	Store Byte	3-23
SBX	Store B Indexed by X	3-20
SBY	Store B Indexed by Y	3-21
SDSP	Store Display	5-7
SEZ	Skip if E is Zero	3-12

INSTRUCTION		PAGE
SFB	Scan for Byte	3-23
SFC	Skip if Flag Clear	3-14
SFS	Skip if Flag Set	3-15
SIMP	Save Interrupted Map	4-4
SIN	Sine	3-39
SLA	Skip if LSB of A is Zero	3-11, 3-12
SLB	Skip if LSB of B is Zero	3-11, 3-13
SOC	Skip if Overflow Clear	3-15
SOS	Skip if Overflow Set	3-15
SPMR	Store Page Mapping Register	4-3
SQRT	Square Root	3-35
SSA	Skip if Sign of A is Zero	3-13
SSB	Skip if Sign of B is Zero	3-13
STA	Store A	3-8
STB	Store B	3-8
STC	Set Control	3-15
STF	Set Flag	3-15
STMP	Store A Map	4-4
STO	Set Overflow	3-15
STX	Store X to Memory	3-21
STY	Store Y to Memory	3-21
SWMP	Save Working Map	4-4
SZA	Skip if A is Zero	3-13
SZB	Slip if B is Zero	3-13
TAN	Tangent	3-35
TANH	Hyperbolic Tangent	3-39
TBS	Test Bits	3-24
VABS	Vector Absolute Value	3-44
VADD	Vector Add	3-41
VDIV	Vector Divide	3-42
VDOT	Vector Dot Product	3-45
VMAB	Vector Maximum Absolute Value	3-45
VMAX	Vector Maximum Value	3-45
VMIB	Vector Minimum Absolute Value	3-46
VMIN	Vector Minimum Value	3-46
VMOV	Vector Move	3-46
VMPY	Vector Multiply	3-42
VNRM	Vector Norm	3-44
VPIV	Vector Pivot	3-44
VSAD	Scalar-Vector Add	3-42
VSDV	Scalar-Vector Divide	3-43
VSMY	Scalar-Vector Multiply	3-43
VSSB	Scalar-Vector Subtract	3-43
VSUB	Scalar Subtract	3-41
VSUM	Vector Sum	3-44
VSWP	Vector Swap	3-47
XAX	Exchange A and X	3-21
XAY	Exchange A and Y	3-21
XBX	Exchange B and X	3-21
XBY	Exchange B and Y	3-21
XCA1	Cross Compare A through DATA1 Map	4-6
XCA2	Cross Compare A through DATA2 Map	4-7
XCB1	Cross Compare B through DATA1 Map	4-7
XCB2	Cross Compare B through DATA2 Map	4-7
XJCQ	Cross Map Jump (and Load C and Q)	4-4
XJMP	Cross Map Jump	4-4
XLA1	Cross Load A through DATA1 Map	4-5

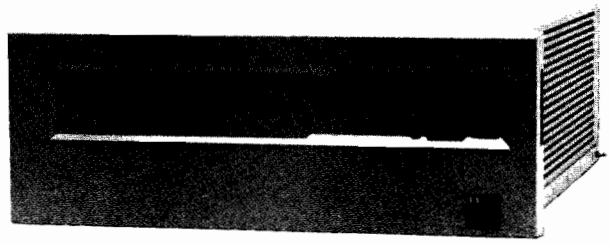
ALPHABETICAL INDEX OF STANDARD INSTRUCTIONS (Continued)

INSTRUCTION		PAGE
XLA2	Cross Load A through DATA2 Map	4-5
XLB1	Cross Load B through DATA1 Map	4-5
XLB2	Cross Load B through DATA2 Map	4-5
XOR	"Exclusive Or" to A	3-8
XSA1	Cross Store A through DATA1 Map	4-6
XSA2	Cross Store A through DATA2 Map	4-6
XSB1	Cross Store B through DATA1 Map	4-6
XSB2	Cross Store B through DATA2 Map	4-6
.BLE	Single Floating Point to Double Floating Point	3-28
.CFER	Transfer Complex or Double Floating Point	3-29
.CPM	Single Integer Arithmetic Compare	3-30
.CPUID	Processor Identification	3-35
.DAD	Double Integer Add	3-31
.DCO	Double Integer Compare	3-32
.DDE	Double Integer Increment	3-32
.DDI	Double Integer Divide	3-31
.DDIR	Double Integer Divide Reverse	3-32
.DDS	Double Integer Decrement and Skip if Zero	3-32
.DFER	Transfer Three Consecutive Words	3-29
.DIN	Double Integer Increment	3-32
.DIS	Double Integer Increment and Skip if Zero	3-32
.DMP	Double Integer Multiply	3-31
.DNG	Double Integer Negate	3-32
.DSB	Double Integer Subtract	3-31
.DSBR	Double Integer Subtract Reverse	3-31
.ENTC	Transfer Parameter Addresses	3-29
.ENTN	Transfer Parameter Addresses	3-29
.ENTP	Transfer Parameter Addresses	3-30
.ENTR	Transfer Parameter Addresses	3-30
.FIXD	FLoating Point to Double Integer	3-26
.FLTD	Double Integer to Floating Point	3-26

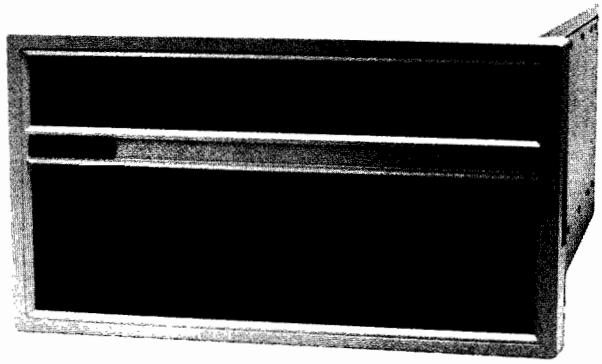
INSTRUCTION		PAGE
.FPWR	Exponentiation	3-40
.FWID	Firmware Identification	3-35
.IMAP	16-Bit Subscript Mapping	3-33
.IRES	16-Bit Subscript Resolution	3-33
.JMAP	32-Bit Subscript Mapping	3-34
.JRES	32-Bit Subscript Resolution	3-34
.LBP	Mapping with Registers	3-35
.LBPR	Mapping with DEF	3-34
.LPX	Indexed Mapping with Registers	3-34
.LPXR	Indexed Mapping with DEF	3-34
.NGL	Double Floating Point to Single Floating Point	3-28
.PMAP	Map Specified Page	3-33
.SETP	Sev A Table	3-30
.SIP	Skip if Interrupt Pending	3-35
.TADD	Double Floating Point Add	3-27
.TDIV	Double Floating Point Divide	3-27
.TFTD	Double Integer to Double Floating Point	3-28
.TFTS	Single Integer to Double Floating Point	3-27
.TFXD	Double Floating Point to Double Integer	3-28
.TFXS	Double Floating Point to Single Integer	3-27
.TMPY	Double Floating Point Multiply	3-27
.TPWR	Exponentiation	3-40
.TSUB	Double Floating Point Subtract	3-27
.WFI	Wait for Interrupt	3-35
.XFER	Transfer Three Consecutive Words	3-29
.ZFER	Transfer Eight Words	3-29
..FCM	Complement and Normalize Single Floating Point	3-30
..TCM	Complement and Normalize Double Floating Point	3-30
/ATLG	(1−X)/(1+X)	3-40



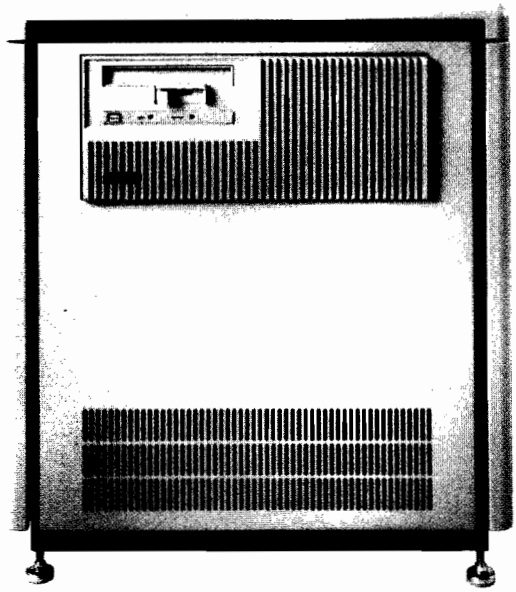
HP 2199A/C



HP 2439A/2489A–OPTION 110



HP 2139A



HP 2199B/D

Figure 1-1. HP 1000 A900 Computers

GENERAL FEATURES

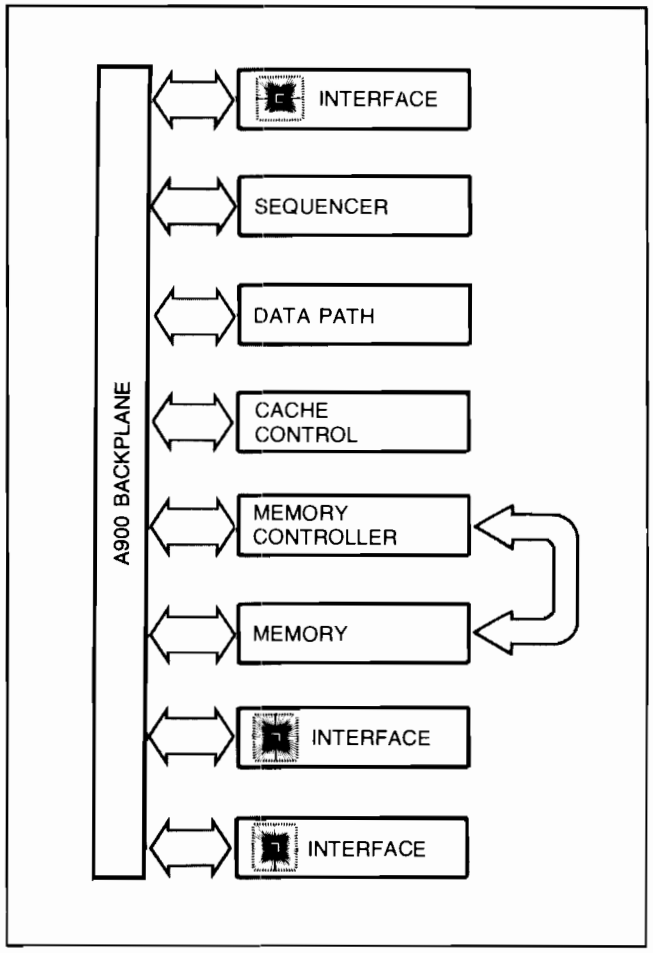
SECTION I

The HP 1000 A900 Computer and Computer System (hereafter referred to as A900 computers) are the most powerful members of the HP 1000 A-Series Computer family. The A900 computers deliver full minicomputer power to a wide variety of applications, and maintain software compatibility with previous HP 1000 Computers. The A900 hardware is available as HP 2139A/2439A Computers (boxes) and computer system processor units (HP 2199A/B/C/D and HP 2489A). (See Figure 1-1.)

1-1. ARCHITECTURE

The A900 computer architecture is based on a distributed intelligence concept that separates the processing of input/output (I/O) instructions from that of other instructions. The central processor unit (CPU) features a microprogrammed control processor and resides on four printed-circuit cards called the sequencer, data path, cache control, and memory controller cards. The computer has cache memory for fast access to instructions and data, and uses pipelining for fast processing. The CPU executes the powerful HP 1000 instruction set, including index instructions and a full complement of instructions for logical operations as well as bit and byte manipulation. The A900 computer base instruction set also includes floating-point, double-integer, and virtual memory addressing instructions, and a language instruction set that substantially increases program execution speed for such high-level languages as FORTRAN and Pascal. The CPU also performs several system level functions, including memory protect, power fail/auto restart, time base generation, memory error interrupt, and extensive self-tests. The A-Series architecture also includes a feature called Code and Data Separation (CDS) which accommodates programs that have up to 4 million words of code.

All input/output instructions are executed by custom silicon-on-sapphire (SOS) input/output processor (IOP) integrated circuit chips that reside on the individual I/O interface cards. The instructions are fetched from memory and decoded by the processor cards. When an instruction is decoded as being of the I/O type, it is broadcast on the backplane for execution by the appropriate I/O processor chip. Because each I/O card is capable of operating independently of the CPU, the A900 can perform direct memory access (DMA) I/O transfers very efficiently. An I/O card interacts with the CPU only on DMA initiation and completion; beyond that, the entire high-speed transfer is handled by the I/O card, leaving the CPU free to work on other tasks. This achieves high efficiency in CPU and I/O throughput. Figure 1-2 is a simplified block diagram of the A900 computer.



8200-137

Figure 1-2. A900 Computer Simplified Block Diagram

1-2. FLOATING-POINT HARDWARE

The A900 includes as a standard feature high-speed dedicated logic that performs floating-point arithmetic operations. The floating-point logic has three custom SOS chips and provides exceptionally fast execution of single precision (32-bit) and double precision (64-bit) floating point operations. Microcoded firmware implements a powerful Scientific Instruction Set (SIS) that performs trigonometric, logarithmic, and other transcendental functions. The A900 also includes a Vector Instruction Set (VIS) which uses the floating-point chips as a computing resource to perform vector and matrix arithmetic and to process large data arrays. The floating-point hardware achieves extremely high accuracy with computational speeds that are 6 to 30 times faster than comparable software routines.

1-3. USER MICROPROGRAMMING

The power and flexibility of microprogramming is made available to the A900 computer user through a microinstruction set of microorders. Microprogrammers have access to special scratch pad registers in addition to the other internal registers of the A900, and can address up to 32K 48-bit words of control store. A900 computers also support up to 14 levels of nested subroutines in microprograms. Microprogramming offers the advantages of speed and security as well as the ability to expand the instruction set to meet a variety of special computing needs.

Microprogramming is supported by Hewlett-Packard through a software package and customer training courses. A paraphraser microassembler allows the user to write microprograms in a versatile free-format style that greatly enhances program readability and documentation compared to traditional microprogramming techniques. User-developed microprograms can be dynamically loaded into optional Writable Control Store (WCS) for execution, and permanently fused into programmable read-only memory (PROM) chips for mounting on the optional Control Store card. (Refer to Section VIII for more information on microprogramming.)

1-4. VIRTUAL CONTROL PANEL

The Virtual Control Panel (VCP) program is an interactive program that enables an external device (such as a terminal) to control the CPU in a manner similar to a conventional computer control panel and also provides additional features. That is, it allows the operator to access the various registers (A, B, etc.), examine or change memory, and control execution of a program. The VCP program is stored in PROM on the cache control card. In a typical application, the VCP could be an HP 262x or HP 264x Terminal interfaced by an HP 12005 Asynchronous Serial Interface Card. When not being used as the VCP, the VCP-assigned terminal can be used in the same way as any other terminal connected to the system. When the A900 computer is operating as a node in a computer network via DS/1000-IV, the VCP can be an adjacent computer in the network.

1-5. BOOTSTRAP LOADERS

There are several bootstrap loaders stored in PROM on the cache control card. The loaders provide program loading from several sources including disc drives, PROM storage modules, a DS/1000-IV network link, HP mini-cartridge tapes, magnetic tape drive, and cartridge tapes of the HP 7908/11/12/14 Disc Drives. The first three loaders can be selected for auto-boot by switches on the data path card; any of the loaders can be selected by operator commands via the Virtual Control Panel.

1-6. SELF-TEST ROUTINES

Two self-test routines are standard in the A900 computer and are stored in PROM on the sequencer and cache control cards. These routines are executed whenever computer power is turned on, providing a convenient confidence-check of the processor cards and memory cards. Execution of both routines can also be initiated by a switch on the sequencer card, and execution of the second routine can be initiated by operator command via the Virtual Control Panel.

1-7. TIME BASE GENERATOR

The memory controller card includes a time base generator that can be used to time external events or to create a real-time clock in software. The time base generator (TBG) can generate an interrupt every 10 milliseconds. The TBG, which can be enabled and disabled by standard I/O instructions, is disabled at power up.

1-8. POWER SUPPLY

A900 computers have a power supply designed to continue normal operation in environments where ac input line voltages and frequencies may vary widely without affecting the operation of the computer. An optional battery backup card and battery pack can be installed area to sustain memory for up to 180 minutes in the event of a complete power failure, thus providing an automatic restart capability. Another power supply option provides two 25-kHz voltages that can be rectified at the load and used to power accessory plug-in cards used for measurement and control applications.

1-9. INPUT/OUTPUT

The input/output system for A900 computers features a custom SOS chip on each I/O card, enabling each card to process its own I/O instructions and handle direct memory access (DMA) data transfers. The I/O system has a multilevel vectored priority interrupt structure with 53 distinct interrupt levels, each of which has a unique priority assignment. Any I/O device can be selectively enabled or disabled, or all I/O devices can be enabled or disabled under program control.

Data transfer between the computer and I/O devices can take place under DMA control or program control. The DMA capability provides a direct link between memory and I/O devices. The total bandwidth through multiple DMA channels is 3.7 million bytes (1.85 million words) per second.

The A900 computer backplane provides the link between the processor, memory, interface cards, and the power supply. In the card cage of the HP 2439A/89A, the backplane has 16 plug-in card slots, of which two are reserved for the optional battery backup card and 25-kHz

power module. Three slots must be used for the processor cards, one for the memory controller card, and one for each memory array card. Thus, in the HP 2439A, there are nine slots available for I/O cards (if only one memory array card is installed) or eight slots available (if the maximum of two array cards is installed), provided that the battery backup card is not installed. In addition to the three processor cards and the two memory cards, the HP 2489A System Processor Unit (SPU) includes as standard a terminal interface card and an HP-IB interface card (for a disc drive), leaving either six or seven slots available for I/O cards.

In the HP 2139A and 2199A/B/C/D the backplane has 20 plug-in card slots. With the three processor cards and two memory cards installed, the HP 2139A has up to 15 slots available for I/O cards, depending upon the number of additional memory array cards installed. In addition to the five basic processor and memory cards, the HP 2199A/B/C/D SPU includes as standard a terminal interface card and an HP-IB interface card (for a disc drive), leaving up to 13 slots available for I/O cards. An SPU that has the full complement of six million bytes of memory still has six slots available for additional I/O cards.

The A900 computer uses standard HP A/L-Series I/O cards. An important feature of these cards is a common-content Global Register which can be loaded with the select code of a specific I/O card. When the Global Register is enabled all I/O instructions are executed only by the I/O card whose select code is in the Global Register. This not only facilitates setting up DMA transfers but also makes reconfiguration of an I/O driver a simple matter of changing the Global Register to the appropriate select code. Also, since the Global Register can direct I/O instructions to a specific I/O card, the I/O-instruction address bits can be used to access registers on an I/O card.

About one-third of the area on all A/L-Series I/O cards is occupied by identical logic called the I/O Master, consisting of an I/O processor chip and its associated logic. The I/O Master is also available in breadboard form for users who wish to design their own I/O cards. The I/O Master is described in detail in the *HP 1000 L-Series Computer I/O Interfacing Guide,* part no. 02103-90005.

1-10 MEMORY

A900 computers are available with semiconductor memory systems based on 64k-bit and/or 256k-bit dynamic RAM (random-access memory) chips. The memory system consists of a memory controller card with error-correction capability, up to eight HP 12220A/21A Memory Array Cards, and a memory frontplane. The 12220A card uses 64k-bit chips and provides 768 kilobytes of memory, while the 12221A card uses 256k-bit chips and provides three megabytes of memory; both cards may be used together in the same computer. The data is stored in memory as two 16-bit words with 7-bit error correction to detect errors. Data is transferred over the frontplane to the memory controller, and all data transfers occur 39 bits at a time (two 16-bit words and 7 check bits).

The error-correcting memory system provides fault-secure memory operation for the A900 computers. The system is capable of correcting all single-bit errors, and of detecting all double-bit errors and many multiple-bit errors. The memory controller card logs the physical address and error syndrome of single-bit errors, and an LED on each memory array card will indicate a multiple-bit error that occurs on that card after the last power-on. The error-correcting system is particularly valuable in computer systems with large amounts of memory, or where fault-secure operation is essential.

The maximum memory size available in A900 computers is 24 megabytes. Addressing physical memory configurations larger than 64k bytes is made possible by the use of the Dynamic Mapping System (DMS), which is standard in the A900 and is described in Section IV. The DMS is a powerful memory management scheme that allows A900 computer users to address up to 32 megabytes of memory and provides either write or read-and-write protection of each individual 2048-byte page.

1-11. CACHE MEMORY

For high performance, the A900 utilizes high-speed, 4k-byte, single-set cache memory with write-to and read-before-write features. This provides single-cycle access to memory for both reads and writes for the CPU when the requested data is in the cache. If the data is not in the cache, then the cache must wait for four CPU cycles before completing the request. Since the probability of the data being in cache (hit) is much more likely than its not being there (miss), the effective memory access time is close to one CPU cycle. The exact ratio of hits to total cycles varies according to what type of program is running and how localized its requests are. The average hit ratio is expected to be around 88 percent but is highly variable.

The main memory for the A900 is not connected to the computer backplane. DMA requests from I/O cards are buffered by the cache memory, and the CPU can still use memory (cache) during high DMA traffic. Likewise, CPU memory operations do not go across the I/O backplane; therefore, DMA does not get held back by a high rate of CPU memory requests. This parallelism makes the A900 ideal for applications where both DMA rate and processor throughput are very important.

For its address, the cache memory takes a 15-bit logical address plus a 5-bit map set select address and translates these via the Dynamic Mapping System into a 24-bit physical address. It also checks for violations of protected memory.

1-12. SOFTWARE

Software support for the A900 computer begins with RTE-A, a member of HP's family of Real-Time Executive (RTE) operating systems. RTE-A is a real-time multiprogramming, multi-user system designed to take full advantage of the A900 I/O structure to enhance overall CPU and I/O throughput. RTE-A offers a wide range of configurations, from a small, memory-based, execute-only system to a full disc-based system with on-line program development. Utilizing the A900 mapped memory system, RTE-A supports user partitions of up to 64k bytes and memory sizes from 128k bytes to 24 megabytes. Memory can be divided into fixed and dynamically allocated partitions at system generation time. Critical programs can be made resident in fixed partitions to ensure fastest possible response to requests for their execution. Other programs can be assigned partitions from the dynamic memory pool according to need, using the smallest available block of memory.

RTE-A also supports Virtual Memory Addressing (VMA) for access to data arrays much larger than main memory (up to 128 megabytes). The disc functions as an extension of main memory so far as data is concerned, in a manner that is transparent to the user and does not require any special programming. In addition, RTE-A supports a special case of VMA, called Extended Memory Area (EMA). With EMA, up to two megabytes of a program's data can be in main memory at once, which affords faster processing of data arrays small enough to use the EMA capability. The programmer chooses the data array handling mode at program load time.

Disc-based RTE-A systems support program development in FORTRAN 77, Pascal/1000, BASIC, and Macro/1000 Assembly Language. Program development for the A900 can also be performed on an HP 1000 System under RTE-6/VM or RTE-IVB.

The HP 92078A software accessory package provides software support, via the Code and Data Separation (CDS) feature, for programs that have up to 4M words of code. With CDS, a large application program is automatically segmented by the LINK loader program into one or more code segments, in addition to a data segment which may be up to 31k words in size; the program may also access a VMA area. The code segments may reside on disc or in memory, and the process of accessing code segments in physical memory, or loading a code segment from disc into physical memory, is automatically handled by a combination of microcode and software. CDS is described further in Section V.

The diagnostic packages listed in Table 1-1 may be used for testing and fault location.

1-13. HP INTERFACE BUS

Among the I/O interface cards available for the A900 computer is the HP 12009A HP-IB Interface Card which can interface the A900 computer to a variety of HP peripherals and other equipment compatible with the Hewlett-Packard Interface Bus (HP-IB). (HP-IB is the Hewlett-Packard implementation of IEEE standard 488-1978, "Digital Interface for Programmable Instrumentation".) A single HP 12009A can control up to 14 HP-IB instruments, and several can be used to achieve concurrent operation of multiple HP-IB instrumentation clusters under the RTE-A multiprogramming operating system.

1-14. COMPUTER NETWORK

The user can configure the A900 computer into an HP DS/1000-IV Distributed System by using either an HP 12007A or an HP 12044A HDLC Interface. Both of these interfaces support the high-level data link communications (HDLC) protocol, functioning as a preprocessor to handle low and medium levels of protocol processing. The A900 computers can be easily mixed with other members of the HP 1000 family in a single computer network. The HP 12042A Programmable Serial Interface allows the sophisticated OEM to design his own customized protocol for networks. Hewlett-Packard offers a customer training course on how to program the PSI card.

1-15. EXPANSION AND ENHANCEMENT

Table 1-1 lists accessory products available to expand or enhance the A900 computers.

1-16. SPECIFICATIONS

Complete specifications for the A900 computers and system processor units are given in a data handbook available from your nearest Hewlett-Packard Sales and Service Office. (These offices are listed at the rear of this manual.) Table 1-2 provides an abridged set of A900 specifications. Except where indicated, the specifications are applicable to both the computers and the system processor units. Both the computers and the SPUs meet the safety standards of the Underwriters' Laboratories (UL), the Canadian Standards Association (CSA), and the International Electrotechnical Commission (IEC). The A900 computers and SPUs also meet the Federal Communications Commission (FCC) Class A and Verband Deutscher Elektrotechniker (VDE) Level A standards for electromagnetic interference (EMI).

Table 1-1. Options and Accessories

DESCRIPTION	HP PRODUCT NO.	OPTION NO.
Removes standard memory array card	—	014
230 Vac Operation	—	015
768k Byte Memory Array Card	12220A	—
3M Byte Memory Array Card	12221A	—
Memory Frontplane for one memory array card	12222A	—
Memory Frontplane for two memory array cards	12222B	—
Memory Frontplane for three memory array cards	12222C	—
Memory Frontplane for four memory array cards	12222D	—
Memory Frontplane for five memory array cards	12222E	—
Memory Frontplane for six memory array cards	12222F	—
Memory Frontplane for seven memory array cards	12222G	—
Memory Frontplane for eight memory array cards	12222H	—
Asynchronous Serial Interface	12005B	—
Parallel Interface	12006A	—
HDLC Interface (modem operation)	12007A	—
PROM Storage Module	12008A	—
HP-IB Interface	12009A	—
Intelligent Breadboard	12010A	—
Extender Board (for memory and I/O cards)	12011A	—
Extender Board (for processor cards)	12240A	—
Priority Jumper Card	12012A	—
8-Channel Asynchronous Multiplexer	12040B	—
Programmable Serial Interface	12042A	—
HDLC Interface (hard-wired operation)	12044A	—
High-Level Analog Input Card	12060A	—
Expansion Multiplexer Card	12061A	—
Analog Output Card	12062A	—
16-In/16-Out Isolated Digital I/O Card	12063A	—
DS/1000-IV Data Link Slave Interface	12072A	—
DS/1000-IV Modem Interface to HP 3000	12073A	—
LAP-B Network Interface	12075A	—
DS/1000-IV Direct Connect Interface to HP 3000	12082A	—
Battery Backup Card	12154A**	—
Battery Backup Module	12157A	—
25 kHz Sine Wave Module	12158A	—
25 kHz Power Module	12159A**	—
Control Store Card	12205A	—
Diagnostic Package for A900 processor and interfaces	24612A*	—
Diagnostic Package for A900-compatible hard disc drives and magnetic tape units	24398B*	—

*Included with the HP 2199A/B/C/D and HP 2489A System Processor Units.
**For HP 2439A and 2489A only.

Table 1-2. Specifications

SPECIFICATIONS COMMON TO THE HP 2139A, 2199A/B/C/D, 2439A, and 2489A

CENTRAL PROCESSOR

Word Size:	16 bits
Instruction Set:	292 standard instructions
Memory Reference:	14
Register Reference:	43
Input/Output:	13
Extended Arithmetic:	10
Index:	34
Bit, Byte, Word Manipulation:	10
Floating Point:	16
Scientific:	14
Language:	14
Dynamic Mapping:	40
Vector Instructions:	38
Double Integer:	12
Virtual Memory:	9
Operating System:	4
Code and Data Separation:	21

Registers:

Accumulators:	Two (A and B), 16 bits each. Implicitly addressable, also explicitly addressable as memory locations.
Index:	Two (X and Y), 16 bits each
Memory Register:	One (P), 15 bits
Base:	One (Q), 15 bits; one (C), 1 bit.
Bounds:	One (Z), 16 bits
Supplementary:	Two (overflow and extend), one bit each
Power Fail Provisions:	When primary line power falls below a predetermined level while the computer is running, a power fail warning signal from the computer power supply causes an interrupt to memory location 00004. This location is intended to contain a jump-to-subroutine (JSB) instruction to a user-supplied power fail subroutine. A minimum of 5 milliseconds is available to execute the power fail subroutine.
Time Base Generator Interrupt:	A time base generator interrupt is provided for maintaining a real time clock. The interrupt request is made when the CPU signals, at 10-millisecond intervals, that its internal clock is ready to roll over. Timing accuracy of the time base generator is ±2.16 seconds per 24-hour day.

MEMORY

Implementation:	64k or 256k dynamic RAM
Cache Size:	4k bytes
Cache Cycle Time:	133 nanoseconds
Cache Fault Processing Time:	532 to 931 nanoseconds
Main Memory Cycle Time:	Read: 532 nanoseconds Write: 400 nanoseconds
Average Effective Memory Access Time:	Approximately 181 nanoseconds, assuming 88% cache hit rate.
Memory Structure:	32 pages of 2048 bytes per page, with direct access to current page or base page (page 0), and indirect or indexed access to all pages. With CDS enabled, a 15-bit base register is added to addresses on base page.
Memory Expansion:	Paged memory address space expandable to 16k pages of 2048 bytes with standard Dynamic Mapping System. Maximum physical memory capacity is currently 12k pages (24 megabytes).

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2139A, 2199A/B/C/D, 2439A, and 2489A (Continued)

INPUT/OUTPUT

Determination of I/O Address:	I/O address select code is set for each interface card by select code switches on the card and is therefore independent of interface card position in the card cage.
I/O Device Interrupt Priority:	Depends upon I/O interface card position in the card cage with respect to the processor cards.
Interrupt Masking:	The I/O Master Logic includes an interrupt mask register which provides for selective inhibition of interrupts from specific interfaces under program control. This capability can be programmed to temporarily cut off undesirable interrupts from any combination of interfaces when they could interfere with crucial transfers.
Interrupt Latency (without DMA interference):	3.7 to 19 microseconds. 4 microseconds typical. (Interrupts cannot be serviced until a DMA cycle or an instruction in progress has completed execution.) The worst-case latency of 19 microseconds is based upon the longest uninterruptible time.
Direct Memory Access (DMA):	The I/O processor chip supports DMA capability on each I/O interface, which reduces the number of interrupts from one per data item (byte or word) to one per complete DMA block.
Data Packing Under DMA:	When byte mode is specified in Control Word instructions, the I/O processor chip automatically packs or unpacks bytes.
Maximum Achievable DMA Rate:	1.85 million words (3.7 megabytes) per second for input transfers; 1.5 million words (3.0 megabytes) per second for output transfers.
SAFETY AND RFI QUALIFICATION:	HP 1000 A900 products meet the safety standards of the Underwriters' Laboratories (UL), the Canadian Standards Association (CSA), and the International Electrotechnical Commission (IEC). The A900 also complies with the RFI standards of the Federal Communications Commission (FCC) and Verband Deutscher Electrotechniker (VDE).
VIBRATION AND SHOCK:	HP 1000 A900 products are type tested for normal shipping and handling shock and vibration. (Contact factory for review of any application that requires operation under continuous vibration.)

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2139A, 2199A/B/C/D

POWER SUPPLY

Output:

DC voltages and tolerances

+5V	±2%
+12V	+6/−3%
−12V	±6%

Optional AC Voltages and Tolerances:

27V rms ±8%, 25 kHz nominal, split phase from three pins on backplane-mating connector. Total harmonic distortion: <10%.

Maximum Output Current Ratings:

+5V	+5M	+12V	−12V	25 kHz
70A	10.0A	5.6A	3.5A	1.5A

Short Circuit Protection:

All dc and ac power outputs are fault protected for short circuits. The power supply will shut down if any of the outputs are short circuited at turn on.

+5V Output Overvoltage Protection:

The +5V output is sensed for overvoltage and the +5V supply shuts down if its output voltage exceeds 5.5V. The ac power switch must be cycled to reset the +5V output.

DC Current Available for I/O Interfaces:

The power supply provides enough current for any combination of compatible HP interfaces and other HP plug-ins.

BATTERY BACKUP

12157-60001 Battery Backup Module

1420-0304 Battery Pack

The Battery Backup System provides from 15 to 75 minutes of sustaining power depending upon the number of memory array cards (5 maximum), state of charge, and temperature; additional hold-up time can be achieved by connecting an external battery.

Recharge Time:

24 hours for fully discharged battery pack

Battery Type:

Sealed lead acid

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2139A COMPUTER

ELECTRICAL SPECIFICATIONS

AC Power Required

Line Voltage:	86-138V (115V −25%/+20%) standard; 178-276V (230V −23%/+20%) option 015.
Line Frequency:	47.5 to 66 Hz
Maximum Power Required:	800 watts

PHYSICAL CHARACTERISTICS

Dimensions

Height:	266 mm (10.5 in)
Width:	483 mm (19 in)
Depth:	610 mm (24 in)
Weight:	29.1 kg (64 lb)
Ventilation:	Air intake is in through the front; exhaust is out through the rear.
Volume:	Approximately 10.7 cubic metres/min. (380 CFM).

ENVIRONMENTAL SPECIFICATIONS

Temperature

Operating:	0° to 55°C (32° to 131°F) to 3048 metres (10,000 ft); 0° to 45°C (32° to 113°F) to 4,572 metres (15,000 ft).
Non-operating:	−40° to 75°C (−40° to 167°F) −40° to 60°C (−40° to 140°F) with Battery Backup
Relative Humidity:	5% to 95% non-condensing

Altitude

Operating:	To 4.6 km (15,000 ft)
Non-operating:	15.3 km (50,000 ft)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2199A/B/C/D

ELECTRICAL SPECIFICATIONS

Standard Line Voltage and Line Frequency

Line Voltage (With HP 7908R):	88-127V (115V nominal)
Line Voltage (With HP 7911R, HP 7912R, or HP 7914R):	90-105V (100V nominal) or 108-126V (120V nominal)
Line Frequency:	With HP 7908R: 47.5 to 66 Hz With HP 7911/12/14R: 54 to 66 Hz.

Option 015 Line Voltage and Line Frequency

Line Voltage (With HP 7908R):	180-255V (230V nominal)
Line Voltage (With HP 7911R, 7912R, or 7914R):	198-231V (220V nominal) or 216-252V (240V nominal)
Line Frequency:	With HP 7908R: 47.5 to 66 Hz With HP 7911/12/14R: 47.5 to 55 Hz

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2199A/B/C/D (Continued)

Power Requirements:	Requires at least 20-ampere grounded power receptacle for 115 Vac operation, or at least 10-ampere grounded power receptacle for 230 Vac operation (option 015). The HP 2199A/C requires split-phase power; the HP 2199B/D requires single-phase power. An additional power receptacle is required for the system console.
Maximum Current Required:	HP 2199A/C: 16 amperes per phase HP 2199B/D: 16 amperes
PHYSICAL CHARACTERISTICS	
Dimensions	
Height:	HP 2199A/C: 1613 mm (63.4 in) HP 2199B/D: 720 mm (28.3 in)
Width:	635 mm (25 in)
Depth:	813 mm (32 in)
Weight	
Without Disc Drive:	HP 2199A/C: 139.7 kg (307.5 lb) HP 2199B/D: 94.3 kg (207.5 lb)
HP 7908R Disc Drive adds:	37.0 kg (81.6 lb)
HP 7911R/12/14R Drive adds:	67.3 kg (148 lb)
Ventilation:	Perforations in the HP 2199B/D cabinet and in the lower part of the HP 2199A/C cabinet facilitate front-torear ventilation driven by the fans in the computer and system disc. Four 120 CFM fans at the top rear of the HP 2199A/C cabinet draw in air through a filter at the bottom rear of the upper section, providing bottomto-top airflow of approximately 11.3 cubic metres per minute (400 CFM). The actual value of air flow depends upon the configuration of user equipment racked in the upper section of the cabinet.
Racking Limitations:	The additional spece in the top half of the HP 2199A/C cabinet is intended for user equipment installed on rails and not on slides.
ENVIRONMENTAL SPECIFICATIONS	
Temperature	
Operating (with HP 79xxR Disc):	0° to 40°C (32° to 104°F), rate of change is <10°C (18°F) per hour.
Non-operating:	−40° to 60°C (−40° to 140°F)
Relative Humidity:	
With HP 79xxR Disc:	20% to 80% non-condensing
Altitude	
Operating:	To 4.6 km (15,000 ft)
Non-operating:	To 15.3 km (50,000 ft)

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2439A and 2489A

ELECTRICAL

AC Power Required

Line Voltage:	86-138V (115V −25%/+20%) standard; 178-276V (230V −23%/+20%) option 015.
Line Frequency:	47.5 to 66 Hz
Operating Current:	6A, max. in 115V configuration; 3A, max. in 230V configuration.

PHYSICAL CHARACTERISTICS

Dimensions

Height:	178 mm (7 in)
Width:	483 mm (19 in)
Depth:	648 mm (25.5 in)

Weight

Without Integral Discs:	18.1 kg (40 lb)
Integral Discs Add:	2.27 kg (5 lb)
Ventilation:	Air intake is in through the left; exhaust is out through the right.

ENVIRONMENTAL SPECIFICATIONS

Temperature

Operating:	0° to 55°C (32° to 131°F) to 3048 metres (10,000 ft) without Option 110 internal discs. Maximum temperature is linearly derated 2°C (3.6°F) for each 304.8m (1000 ft) increase of altitude. Resulting temperature range is 0° to 45°C (32° to 113°F) at 4572 metres (15,000 ft). 5° to 45°C (40° to 113°F) with Option 110 internal discs; maximum rate of change <10°C (18°F) per hour.
Non-operating;	−40° to 75°C (−40° to 167°F) maximum temperature with Option 110 internal discs is 60°C (140°F).
Relative Humidity:	
Without Option 110 Discs:	Operating: 5% to 95%with maximum wet bulb temperature not to exceed 40°C (104°F), excluding all conditions which cause condensation.
With Option 110 Discs:	Operating: 20% to 80% with maximum wet bulb temperature not to exceed 29°C (85°F), excluding all conditions which cause condensation.
Non-operating:	5% to 95% non-condensing.
Altitude	
Operating:	To 4.6 km (15,000 ft)
Non-operating:	To 15.3 km (50,000 ft)

POWER SUPPLY

Output: DC voltages and tolerances

+5.1V	+/−2%
+12V	+6/−3%
−12V	+/−6%

Maximum Output Current Ratings:	+5.1V	+12V	−12V
	50A	7.0A	3.0A

Short Circuit Protection: All dc power outputs are fault protected for short circuits. The power supply will shut down if any of the outputs are short circuited at turn on.

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2439A and 2489A (Continued)	
25 KHZ AC VOLTAGE	HP 12159A 25 kHz Power Module The Power Module provides 27V p-p +/−8%, 25 kHz nominal, split phase from three pins on the backplane-mating connector. Maximum output is 30 watts.
BATTERY BACKUP	HP 12154A Battery Backup Card The Battery Backup Card provides from 45 to 180 minutes of memory sustaining power depending upon system configuration, state of charge, and temperature; additional hold-up time can be achieved by connecting an external battery.
Recharge Time:	14 hours for fully discharged battery pack
Battery Type:	Nickel cadmium

OPERATING FEATURES

SECTION II

This section describes the bootstrap loaders, the Virtual Control Panel (VCP) program, and the central processor registers accessible to the programmer.

2-1. [illegible]

The processor cards have several working registers that can be selected for display and modification via the Virtual Control Panel program. (Interface card registers are described in Section VI of this manual and in the interface card reference manuals.) The functions of these processor card registers are described in the following paragraphs.

2-2. A-REGISTER

The A-register is a 16-bit accumulator that holds the results of arithmetic and logical operations performed by programmed instructions. This register can be addressed directly by any memory reference instruction as location 000000 (octal), thus permitting interrelated operations with the B-register (e.g., "add B to A," "compare B with A," etc.) using a single-word instruction.

2-3. B-REGISTER

The B-register is a second 16-bit accumulator which can hold the results of arithmetic and logical operations completely independent of the A-register. The B-register can be addressed directly by any memory reference instruction as location 000001 (octal) for interrelated operations with the A-register.

2-4. P-REGISTER

The 15-bit P-register holds the address of the instruction being executed. When the machine is halted, the P-register contains the address of the next instruction to be fetched from memory.

2-5. EXTEND (E) REGISTER

The one-bit extend (E) register is used by rotate instructions to link the A- and B-registers or to indicate a carry from the most-significant bit (bit 15) of the A- or B-register by an add instruction or an increment instruction. This is of significance primarily for multiple-precision arithmetic operations. If already set (logic 1), the extend bit cannot be cleared by the absence of a carry. However, the extend bit can be selectively set, cleared, complemented, or tested by programmed instructions.

2-6. OVERFLOW (O) REGISTER

The one-bit overflow (O) register is used to indicate that an add instruction, divide instruction, floating-point multiply or subtract instruction, or an increment instruction referencing the A- or B-register has caused (or will cause) the accumulators to exceed the maximum positive or negative number that can be contained in these registers. The overflow bit can be selectively set, cleared, or tested by programmed instructions.

2-7. CENTRAL INTERRUPT REGISTER

The central interrupt register is a six-bit register that holds the select code of the last interface card or internal condition whose interrupt request was serviced.

2-8. VIOLATION REGISTER

The violation register is a 15-bit register that records the logical address of any fetched instruction that violates memory protection rules.

2-9. PARITY VIOLATION REGISTER

The 32-bit parity violation register (PVR) stores the physical address of the last memory location that caused a single-bit or multiple-bit error. (A single-bit error is automatically corrected by the memory controller, and the error syndrome is logged with the physical address.)

2-10. INTERRUPT SYSTEM REGISTER

The interrupt system register is a one-bit register that indicates the status of the interrupt system. When set (logic 1), the interrupt system is enabled; when cleared (logic 0), the interrupt system is disabled.

2-11. X- AND Y-REGISTERS

These two 16-bit registers, designated X and Y, are accessed through the use of the 32 index register instructions and two jump instructions described in Section III.

2-12. WMAP REGISTER

This 16-bit register holds the map numbers used for memory references by Dynamic Mapping System instructions. (The DMS is described in Section IV.)

2-13. IMAP REGISTER

The IMAP register is a 16-bit register that holds the value which WMAP had at the last interrupt. The IMAP register may be accessed only by the SIMP instruction.

2-14. C- AND Q-REGISTERS

The one-bit C-register determines whether the Code and Data Separation (CDS) feature is enabled (0=enabled; 1=disabled). The Q-register is a 15-bit base register whose value is added to memory addresses whenever CDS is enabled and a memory address is between 2 and 1023, inclusive.

2-15. Z-REGISTER

This 16-bit register is a bounds register used by Code and Data Separation instructions to protect user memory (refer to Section V).

2-16. IQ-REGISTER

This 16-bit register holds the value which the C- and Q-registers had at the last interrupt.

2-17. VIRTUAL REGISTERS

There are two virtual registers, M and T, that are created by the Virtual Control Panel program and which can be accessed, bia the VCP, to examine or change a program in memory or to manually create a program in memory.

2-18. M-REGISTER

The M-register holds the address of the memory cell currently being read from or written into by the Virtual Control Panel.

2-19. T-REGISTER

The T-register indicates the contents of the memory location currently pointed to by the M-register.

2-20. CONTROLS AND INDICATORS

Operator controls and indicators for an A900 computer system are described in the appropriate system installation and service manual.

On the A900 computer there is only one operator control: a line-power switch. This two-position switch controls the application of ac line power to the computer power supply and ventilating fans. Light-emitting diodes (LEDs) on the sequencer card provide indications for the computer self-test.

2-21. SELF-TEST

The self-test consists of two test programs (Test 1 and Test 2) that automatically execute each time the computer is powered up and which provide a quick, convenient check of basic computer operation. (Also, the self-test can be executed by pressing the Reset switch on the sequencer card.) Test 1 tests the processor at the level of individual circuits, and Test 2 tests the processor at a functional level. (For example, Test 1 tests the hardware associated with an LIA instruction to ensure that it works correctly, and Test 2 executes an LIA instruction and checks the result.) If either test program fails, the computer will not operate. Successful completion of the self-test is followed immediately by execution of either a bootstrap loader, the Virtual Control Panel program, or a program sustained in memory by an optional battery pack, as preselected by the user.

Test 1 is a microprogram stored in PROM on the sequencer card. It executes immediately upon power up and makes checks of all four processor cards, including a check of the microcode PROMs on the sequencer card and the VCP/Test 2 PROMs on the cache control card. On successful completion, Test 2 is started. If Test 1 detects a failure it stops executing and the LEDs on the sequencer card indicate an error code. (Refer to the computer installation and service manual for information on error codes.) Test 1 execution time is negligible.

Test 2 is an assembly language program stored in PROM on the cache control card and executes upon successful completion of Test 1. (Test 2 can also be initiated by the VCP command %T.) Test 2 checks the computer's basic instruction set, several internal flags, and all the memory. If memory was sustained by the optional battery pack, Test 2 checks it in a non-destructive manner by reading each memory location, thus making a parity check on the data. If a parity error does occur, the location is reported to the VCP (if present). If memory was not sustained, Test 2 writes all ones to each memory location, and reads back the data; and then writes all zeros and reads back. (The memory is cleared.) Test 2 also checks the I/O Master logic on each interface card to ensure that data transfer, flag, interrupt, and direct memory access (DMA) transfer and flag functions are processed correctly. If Test 2 detects a failure, it stops executing and the sequencer LEDs indicate an error code. (If a VCP is in the system and the failure does not hinder VCP operation, the VCP program is entered and the failure code is displayed on the VCP.) The LED indication on successful completion of Test 2 depends on the computer action selected by the Start-Up switches on the data path card. Test 2 has a maximum execution time of 10 seconds.

2-22. BOOTSTRAP LOADERS

Bootstrap loading of a program for the A900 computer is provided for by six loaders contained in PROMs on the cache control card. The loading devices are disc drive (via HP-IB or disc interface), PROM storage module, DS/1000-IV network link, HP 264x mini-cartridge tape, cartridge tape of the HP 7908/11/12/14 Disc Drive, and HP 7970E Magnetic Tape Drive. There are two ways to invoke a loader: auto-boot when power comes up; and by VCP command. Auto-boot can only invoke four of the loaders: two discs, PROM module, and DS/1000-IV; the VCP can invoke any of the loaders by a command from the operator. The VCP load commands are discussed later in this section.

2-23. LOADER SELECTION FOR AUTO-BOOT

The selection of an auto-boot is by means of four switches located on the data path card. These switches, the Start-Up switches, are set during installation and also provide options other than auto-boot selection. When a loader has been selected for auto-boot and the self-test completes, the boot loader executes if memory was lost; or the program in memory executes if memory was sustained by the optional battery backup pack. Refer to Table 2-1 for Start-Up switch settings.

2-24. PROGRAM STARTS

When an auto-boot completes without error, the loaded program starts execution at memory location 02. The loader sets the contents of the A- and B-registers as follows:

a. Cold start (memory not sustained):

   1. A = loader command parameters.
   2. B = pointer to a string area where:

      Word 1 = memory size.
      Word 2 = zero.

b. Auto-restart (memory sustained; execution starts at location 04):

   1. A = zero.
   2. B = zero.

c. %E command from VCP:

   1. A = −1.
   2. B = zero.

d. %B command from VCP:

   1. A = loader command parameters.

Table 2-1. Start-Up Switch Settings

BOOT SEL switches* S1	S2	S3	S4	S5	S6	COMPUTER ACTION
C	C	C	C	z	y	Loop on self-test Test 2 regardless of error.
C	C	O	C	z	y	Loop on self-test Test 2 and stop on error.
C	O	O	C	z	y	Run VCP** routine on completion of self-test.
O	C	C	C	z	y	If memory lost (not sustained), run VCP routine; otherwise, restart program (JMP 4B). (Note 2, Note 4.)
O	C	O	C	z	y	If memory lost, load and execute program from PROM card; otherwise, restart program (JMP 4B). (Note 2) (In order to auto-boot from PROM, the card must have select code 22. Equivalent to loader command %BRM.)
O	O	C	C	z	y	If memory lost, load and execute program via HDLC card; otherwise, restart program (JMP 4B). (Note 2) (In order to auto-boot via HDLC, the card must have select code 24. Equivalent to loader command %BDS.)
O	O	O	C	z	y	If memory lost, load and execute program from first file of disc (via HP-IB); otherwise, restart program (JMP 4B). (Note 2) (In order to auto-boot via HP-IB, the HP-IB interface card must have select code 27 and the disc drive must have HP-IB address 2. Equivalent to loader command %BDC.)
O	C	C	O	z	y	If memory lost, load and execute program from first file of disc (via HP 12022A interface); otherwise, restart program (JMP 4B). (Note 2) (In order to auto-boot via HP 12022A, the card must have select code 32 and operate with a hard disc drive having address 0. Equivalent to loader command %BDI.)

* O = open (up); C = closed (down)
y = C, system console uses ENQ-ACK handshake.
y = O, system console does not use ENQ-ACK handshake.
z = C, normal mode, break enabled.
z = O, break disabled (not halts).
** Virtual Control Panel.

Notes:
1. When a loader finishes an auto-boot, it starts execution of the loaded program at location 02.
2. If auto-restart feature is disabled (switch M closed), the program cannot restart and the boot loader (or VCP routine) will execute.
3. Do not use any switch combination that is not shown above.
4. Use this switch configuration for normal computer operation.

2. B = pointer to a string area where:

   Word 1 = memory size (64k bytes).
   Word 2 = string length. (in bytes)
   Word 3 = first word of string.
   Word n = n-2 word of string.

2-25. VCP RE-ENTRY FOR EXTENDED BOOT LOADING

The VCP PROM loader can be re-entered from a program to boot load. It executes a program from a loading device. The VCP code is re-entered as follows:

a. A VCP boot loader call allows the programmer to call any of the VCP loaders. This allows a complete call back sequence including a checkout routine. For a sample VCP loader call back checkout program, refer to Table 2-1A.

```
LDA COUNT          Negative number of characters in the boot string.
LDB POINTER        Starting address of the string.
HLT 0,C            Call VCP loader sequence.
<----->            VCP loader is started and the new program is loaded.
COUNT DEC -12      Negative number of characters (bytes) in the string.
POINTER DEF *+1    Starting address of the string.
                ASC 06,DC2027SYSTEM
```

The string can be any allowable string entered after the %B command (%Bxxffffbusctext). Note that %B is not actually entered but is assumed when using this call.

If the VCP loader encounters an error, the loader will report the error and return to the VCP) prompt.

b. With the disc loader, re-enter to boot load the specific program described by the "ABS" code in the following call back programming sequence.

```
CLA,CLE,INA    Indicate disc call back — do not suspend
HLT 3,C        Return to VCP loader
ABS...         HP-IB bus address
ABS...         Device unit number (head for 7906)
ABS...         Absolute starting sector (Vector 1 for 7908/11/12/14)
ABS...         Cylinder offset (Vector 2 for 7908/11/12/14)
ABS...         Vector 3 for 7908/11/12/14
```

This sequence assumes that the Global Register is set prior to entry to the loader and that the absolute starting sector is the combined cylinder/head/sector for that drive. When the load is completed, the loader will start execution in the standard JMP 2 manner. If a suspend after load was specified by the E-register being set when called, the program will return to VCP after the load. In the case of the suspend the operator can enter either a %E or a %R to continue. Any error will return to the VCP, if present, or start the original load over.

The HP 7906 Disc Drive will be accessed in the surface mode only, all other discs will be accessed in the cylinder mode.

2-26. DEVICE PARAMETERS AND MEDIA FORMATS

There is a specific data format for each combination of loader, interface card, loading device, and media. The data formats are described in Figure 2-1.

2-27. VIRTUAL CONTROL PANEL

The Virtual Control Panel (VCP) program is an interactive program that enables an external device (such as a terminal) to control the CPU in a manner similar to a conventional computer control panel. That is, it allows the operator to load programs using the loaders, access the various registers (A, B, etc. plus I/O card registers), examine or change memory, and control execution of a program. There are two VCP programs stored in PROM on the cache control card: one program is for use with an HP 12005 Interface Card, and the other is for either use with an HP 12007/12044 DS/1000-IV Card or the HP 12040B Multiplexer Card. Only one interface card in the computer can serve as a VCP interface; the card selection is established when the system is installed.

2-28. VCP PROGRAM OPERATION

The VCP program is executed from PROM as a software program and uses the various machine registers (A, B, etc.) during its execution. Therefore, these registers are automatically saved upon entry to the VCP code. (The save area is in boot RAM on the memory controller card.) Thus, the response to an inquiry is the data that was saved at the time of entry to the program. The exceptions to this are indicated by the absence of an asterisk in Table 2-2. When the operator enters the Run (%R) command, the VCP program restores the machine with the current data in the save area and starts execution as specified by the program execution address in the P-register.

The VCP program can be entered in three ways as follows:

a. After a power-up, PROM execution is directed to the VCP program instead of a boot load routine;

b. When the VCP interface card requests a slave cycle to enable the VCP program (e.g., BREAK key pressed on VCP); or

c. When a HLT (halt) instruction is fetched and one I/O card is enabled for break (otherwise the instruction has no effect).

Table 2-1A. Sample VCP Loader Call Back Checkout Program

LABEL	OPCODE	OPERAND	COMMENTS
ASMB,A,B,L,C			
	ORG	2B	
	JMP	START	Goto start of the program.
	NOP		
	NOP		No powerfail, auto restart.
	ORG	100B	
START	HLT	0	Test halt to compare string.
	LDA	COUNT	Negative number of characters in the boot string.
	LDB	PNTR	Starting address of the string.
	HLT	0, C	Call VCP loader sequence.
	NOP		
	NOP		
	NOP		
COUNT	DEC	−18	Negative numbers of characters (bytes) in the string.
PNTR	DEF	*+1	Starting address of the string.
	ASC 09,CT10020 Test String		
	END		

After a power-up, the total memory size is displayed on the VCP screen. The A-register is set to the number of I/O chips that were tested during the self-test. This enables the operator to verify that all installed memory and I/O cards were tested. (Also, except when the self-test detects an error and reports it in the B-register, the B-register contains the revision code of the VCP PROMs.) When entered, the VCP displays the basic set of registers (P, A, B, RW, M, and T) and issues the VCP prompt (VCP>) for an operator response. The operator can enter any of the characters or commands listed in Tables 2-2 and 2-3 and the VCP program will respond as indicated in the tables. A carriage return is entered to terminate a VCP entry.

After a response to an inquiry the operator can change the data contained in that register or memory location by entering new data; for example (operator inputs are underlined and <cr> indicates a carriage return):

A 001234 4321<cr>
A 004321

Table 2-2. VCP Characters and Associated Registers

CHARACTER ENTERED	RESPONSE†	MEANING
A*	xxxxxx	A-register contents
B*	xxxxxx	B-register contents
E*	x	E-register contents
G*	x000xx	Global Register (GR) contents and status (bit 15=1 if enabled, 0 if disabled)
I*	x	Interrupt system status (0=off, 1=on)
M*	0xxxxx	Memory address (pointer for T and Ln command)
O*	x	O-register contents
P*	0xxxxx	Program execution address
Q*	xxxxxx	C- and Q-register contents (C is bit 15)
RS	xxxxxx	Switch register contents
T	0xxxxx xxxxxx	Memory contents pointed to by M
V	xxxxxx	Violation register (memory protect)
X*	xxxxxx	X-register contents
Y*	xxxxxx	Y-register contents
Z*	xxxxxx	Z-register contents
RC	xxxxxx	Central Interrupt Register contents
RD**	xxxxxx xxxxxx	Data for I/O diagnose modes 1 and 2 (refer to paragraph 7-22)
RF**	xxxxxx	I/O flags: Flags 20 thru 24, and Flag 30 (1 = flag set; 0 = flag clear)
RI**	xxxxxx	Interrupt mask register
RP	xxxxx xxxxx xxx	Parity violation register contents
RS	xxxxxx	Switch Register
RW*	xxxxxx	Working map set (WMAP)
R20**	xxxxxx	DMA self-configuration register
R21**	xxxxxx	DMA control register
R22**	xxxxxx	DMA address register
R23**	xxxxxx	DMA count register
R24**	xxxxxx	I/O scratch register
R25**	xxxxxx	I/O scratch register
R26**	xxxxxx	I/O scratch register
R30**	xxxxxx	I/O card data register
R31**	xxxxxx	Optional I/O card register
R32**	xxxxxx	Optional I/O card register
?		Output Help file

† x = octal data.

* Registers that are maintained in the VCP save area of boot RAM.

** Applies only to the I/O card whose select code equals the contents of the Global Register.

NOTE: When a register's contents are changed by the user the new value is returned; if the VCP does not accept a change, the VCP prompt is returned.

Data input is terminated by the operator entering a carriage return. If during an input the program cannot interpret a character, the program will output the characters "|?" and then start a new line with the VCP prompt. Entry errors may be corrected by backspacing over them and entering the correct information. During any data input the operator can abort the input by entering a rub-out (DEL). The loader commands, %B, %L, and %W can also be aborted by a rub-out. When entering data into a register, leading zeros may be omitted. If the operator types a question mark, the VCP will output a "help" file that summarizes acceptable command entries.

2-29. LOADER COMMANDS

The loader commands can be entered via the VCP in either of two ways:

a. Allow the parameter default values (given in Figure 2-1) to be used; or

b. Specify all necessary parameters.

The VCP loader command format is shown in Figure 2-2. The loader command error codes and their meanings are listed in Table 2-4.

2-30. VCP USER CONSIDERATIONS

When using the VCP to debug a program the user should be aware of the following conditions:

a. The VCP program uses an interface card and modifies the characteristics of that card. When the VCP program exits, it sets Register 24 on the interface card to all ones to allow software detection of a VCP interaction and, thus, reinitialization of an operation. (This also causes an interrupt if the interrupt system is enabled.) Also, the VCP will leave the card in the output mode with both Flag 30 and Control 30 set.

b. The status of the interrupt system (STC 4 [on] or CLC 4 [off]) is not indicated and will remain unchanged unless %P is executed to preset the computer.

c. Memory protect is indicated by the sign bit of RW (WMAP register) and may be modified.

2-31. VCP SLAVE FUNCTIONS

The slave feature of an I/O processor chip is used in conjunction with the VCP program. The slave feature enable is read into the I/O chip of the VCP interface card on power-up and cannot be altered until the next power-up condition. After power-up a change in the state of the slave signal causes the I/O chip to generate a slave request. When the request is granted, the I/O chip requests the CPU's current P-register contents and saves these contents in a register in the I/O chip. The I/O chip then

Table 2-3. VCP Commands

COMMAND*	MEANING
%B	Load and go (boot). Execute a specified loader routine and start program execution at completion of load. See Figure 2-2 for format.
%C	Clear memory. Set all memory to zero and perform a preset.
%E	Execute. Start execution of program at location P=2 (A-register equals −1 (all ones) and B-register equals 0).
%L	Load. Similar to %B except do not start execution. See Figure 2-2 for format. (%L followed by %R is equivalent to %B.)
%P	Preset. Generate a control reset (CRS) signal on the backplane to initialize all cards.
%R	Run. Set all registers to the appropriate values in the save area and start execution at address specified by the P-register.
%T	Test. Initiate the self-test Test 2 and return to VCP (memory is sustained but the I/O system is reset).
%W	Write. Write to the selected device. (See Figure 2-2 for format.) When writing to a disc drive, the Count and Partial values defined in Figure 2-1 must be in memory locations 00000 and 00001.
D	Decrement. Decrement memory pointer and display the contents of the M- and T-registers. Valid only after T.
Ln	List. List n blocks of eight memory locations starting with location pointed to by the M-register.
N	Next. Same as D except increment the pointer. Valid only after T.
RMxx	List the 32 map registers in the DMS map set specified by xx.
RMxxPyy	Show the value of register yy in map set xx. If a number is input after this command, the register is changed to the new value.
?	Output Help file.

*Must be followed by a carriage return.

MINI-CARTRIDGE TAPE

Device:	HP 264x Terminal
Interface:	HP 12005B Asynchronous Serial Interface
Default Parameters*:	000020
Format:	Reads absolute binary file, writes 4k absolute binary block.
Loader:	Transmits special escape sequence to invoke a read of a record and does checksum of the data. When writing to tape, a block number is used to specify which 4k-word memory area is to be dumped to tape (0 = 0 to 4k). If a file number is specified then the program will issue a find file command; if not, the tape is read from where it stands. When writing to the tape, the program will not write a file mark; this allows sequential blocks to be written in a series. There are only two units (0 and 1) on the terminal; if a larger unit number is specified, the result will be unpredictable. More than 32k words may be loaded into a system from a single cartridge tape.

PROM MODULE

Device:	PROM (2k × 8 bits)
Interface:	HP 12008A PROM Storage Module
Default Parameters*:	000022
Format:	Count-Partial-Data Count = number of 64k byte blocks. Partial = number of words of partial 64k byte block. Data = 16-bit words, one word per location until Count and Partial are satisfied.
Loader:	Uses STC-LIA process to transfer data. The PROM cannot be written to nor does it use the block number or unit number.

*See Figure 2-2 for loader command formats.

Figure 2-1. Loading Device Parameters and Media Formats (Part 1 of 3)

stores the starting address of the VCP program into the CPU's P-register, instructs the CPU to enable the boot PROM, and allows execution to start. The VCP program can be started in several other ways, as follows:

a. On power-up and after the self-test the VCP program starts execution if it is selected in lieu of a boot loader. This selection may often be used because the loaders can be invoked individually from the VCP.

b. When a HLT* (halt) instruction is executed the I/O processor chip interprets it in the same manner as a change in the slave enable signal. This allows a program to have breakpoints for debugging purposes. (Note that a HLT instruction is not executed but causes a memory protect interrupt if memory protect is enabled.)

During execution of the VCP program, access to the P-save register in the I/O chip is accomplished with LIA/B 3 and OTA/B 3 (without the instruction's Flag bit set). It should also be noted that the I/O chip will not execute a slave request until an STC 2 (enable break feature) instruction has been executed. This prevents re-entry of the VCP program once it has been entered.

During the self-test, the starting address of the VCP program is assigned to the break-enabled I/O card by an OTA/B 3,C* instruction with the A- or B-register set to the address. This address can also be read back with an LIA/B 3,C* instruction.

*If break is not enabled on any I/O card, then the instruction has no effect.

DISC DRIVE

Device:	HP 9895, 7906, 7908, 7910, 7911, 7912, or 7914 Disc Drive, or cartridge tape drive of the 7908/11/12/14 Disc Drive.
Interface:	HP 12009A HP-IB Interface
Default Parameters*:	002027
Format:	Count-Partial-Data Count† = number of 64k byte blocks. Partial† = number of words of partial 64k byte block. Data = 16-bit words, one word per location until Count and Partial are satisfied.
Loader:	Uses HP-IB protocol to communicate with the disc. The load sequence is: 1. Device clear 2. Status check 3. Read/write 32k words via DMA 4. Status check

DISC DRIVE (VIA DISC INTERFACE)

Device:	HP 2439A/89A internal fixed/micro-floppy disc drive.
Interface:	HP 12022A Disc Interface.
Default Parameters*:	000032
Format:	Same as Disc Drive via HP-IB, above.
Loader:	Standard I/O for commands to interface, and DMA for data.

*See Figure 2-2 for loader command formats.
†The Count and Partial values are stored in memory locations 00000 and 00001, respectively.

Figure 2-1. Loading Device Parameters and Media Formats (Part 2 of 3)

MAGNETIC TAPE

Device: HP 7970E or 7974A Magnetic Tape Drive

Interface: HP 12009A HP-IB Interface.

Default Parameters*: 004027

Format: Memory image file
Count-Partial-Data
Count = number of 64k byte blocks.
Partial = number of words of partial 64k byte block.
Data = 256 byte records read until EOF or until Count and Partial are satisfied.

Loader: Uses HP-IB protocol to communicate with the magnetic tape.
The load sequence is:

1. Device ID
2. Device clear
3. Rewind/file forward (if file specified)
4. Read/write
5. Status check

COMPUTER NETWORK

Device: HP 1000 Computer.

Interface: HP 12007A/12044A HDLC Interface.

Default Parameters*: 000024

Format: Reads absolute binary or memory image files, writes a 32k memory image file.

Loader: Standard handshake using HP distributed system protocol. Block number and unit number are not used.

*See Figure 2-2 for loader command formats.

Figure 2-1. Loading Device Parameters and Media Formats (Part 3 of 3)

LOADER COMMAND FORMAT

%B/L/W *dv fffffbusc text*

where:

dv = device type as follows:

DC = disc (cartridge or flexible) via HP-IB
CT = cartridge tape (HP 264x)
RM = PROM card
DS = DS computer network Link
MT = magnetic tape via HP-IB
DI = disc via HP 12022A Card

fffff = file number (octal 0 to 77777 only)

b = 4k-word memory block number when writing to cartridge tape; HP-IB bus address of disc drive; or non-HP-IB drive address; otherwise, use 0. For the HP 2439A/89A internal disc drives, this is 0 for the first fixed drive, 1 for the second, and 3 for the micro-floppy drives.

u = unit number (0 to 7) only if used on device. For the HP 7906 Disc Drive, the unit number is the head number. For HP 7908, 7911, 7912, or 7914 Disc Drive that includes cartridge tape drive, unit 0 = disc drive and unit 1 = cartridge tape drive.

sc = select code of interface card to be used.

text = file name, or ASCII string to be passed to the program after it is loaded. This is only available with the %B and %L commands.

Note: See Figure 2-1 for default parameters for each loading device.

Note that spaces cannot be used in the command entry. The following formats are all acceptable:

%B*dvtext*	Device parameters are defaulted; text cannot start with a number.
%B*dvffbusc*	No text passed.
%B*dvffbusctext*	Text passed.

EXAMPLES:

%BDC	Load and start execution of the default program on disc. (Disc parameters defaulted to 002027; see Figure 2-1).
%BDC30	Load and start execution of the default program on the disc at select code 30 and default other parameters.
%LDC27025	Load (but don't execute) and override parameter default values: file number 2 (i.e., the third file) HP-IB bus address 7 unit 0 select code 25
%WDC27025	Same as above except write to file 2.

Figure 2-2. Loader Command Format

Table 2-4. VCP Loader Command Errors

ERROR CODE	MEANING
2	Select code less than 20 octal.
3	No card with the select code you specified.
Cartridge Tape Loader Errors	
110	File forward error. Status in B-register.
111	Checksum error.
112	No data before EOF (end of file).
120	Write error. Status in B-register.
PROM Module Loader Errors	
211	End of programs.
212	Bad format.
213	System larger than 32k must start on card boundary.
DS/1000 Loader Errors	
310	Time out after CLC 0. Check select code specified.
311	Checksum error. P file not absolute binary.
312	Time out after download request.
313	Time out after file number.
314	Bad transfer (Central generated). Status in B-register.
315	Time out after buffer request.
316	Time out after count echo.
317	Time out waiting for data.
320	Time out after VCP mode requests a DS write.
321	Central will not accept data. Status in B-register.
325	Data block out of sequence.
Disc Loader Errors (via HP-IB)	
411	Time out reading disc type. Check HP-IB address.
412	Time out UDC (Universal Device Code) or reading status. Check disc.
413	Status error. Status in B-register.
414	Time out during file mask.
415	Time out during seek.
416	Time out during read or write command.
417	Time out during DMA of data.
420	Parity error during DMA transfer.
421	Time out during FIFO flush.
422	Time out during DSJ (Device Specified Jump) command.
423	Bad DSJ return. Returned value in B-register.
460	Disc not identifiable. Disc ID in B-register.

ERROR CODE	MEANING
Magnetic Tape Loader Errors	
510	Time out during initialization/read ID.
511	Time out when issuing end/select unit.
512	Mag tape off line.
513	No write ring.
514	Time out during End command.
515	Time out waiting for rewind completion.
517	Time out waiting for DMA transfer.
520	Parity error during DMA transfer.
521	Time out doing a PHI flush.
522	Time out waiting for DSJ.
523	Bad DSJ response.
525	Time out waiting for Mag Tape Not Busy.
530	Time out after issuing a command.
531	Parallel Poll time out after issuing a command.
535	Bad status after read/write command.
550	No data transfer (read only).
560	Not mag tape ID.
HP 12022A Disc Interface Loader Errors	
610	Time out after SDH (Sector Drive Head) for read/write.
611	Time out after cylinder high.
612	Time out after cylinder low.
613	Time out after sector.
614	Time out after sector count.
615	Time out after read/write command.
616	Time out after DMA read/write transfer.
617	Parity error during transfer.
620	Fixed disc not ready.
630	Time out after request status register.
631	Time out after read status register.
632	Time out after waiting for not busy.
633	Time out after request error register.
634	Time out after read error register.
635	Status error: A-register = status register; B-register = error register.
650	Time out after SDH register for restore.
651	Time out after restore.
660	Disc not defined.

PROGRAMMING INFORMATION

SECTION III

This section describes the software data formats and the base set machine-language instruction coding required to operate the computer and its associated input/output system. Machine-language instruction coding for the Dynamic Mapping System is presented in Section IV, and coding for Code and Data Separation is presented in Section V.

3-2. DATA FORMATS

As shown in Figure 3-1, the basic data format is a 16-bit word in which bit positions are numbered from 0 through 15 in order of increasing significance. Bit position 15 of the data format is used for the sign bit; a logic 0 in this position indicates a positive number and a logic 1 in this position indicates a negative number. The data is assumed to be a two's complement whole number and the binary point is therefore assumed to be to the right of the number.

The basic word can also be divided into two 8-bit bytes or combined to form a 32-bit double integer. The byte format is used for character-oriented input/output devices; packing two bytes of data into one 16-bit word is accomplished by software drivers. In I/O operations, the higher-order byte (byte 0) is the first to be transferred.

The double integer format is used for extended arithmetic in conjunction with the extended arithmetic instructions described under paragraph 3-23 and 3-24. Bit position 15 of the most-significant word is the sign bit and the binary point is assumed to be to the right of the least-significant word. The integer value is expressed by the remaining 31 bits, using twos-complement form.

The two floating point formats in Figure 3-1 are used with floating point software. Bit position 15 of the most-significant word is the mantissa sign and bit position 0 of the least-significant word is the exponent sign. Bits 1 through 7 of the least-significant word express the exponent and the remaining bits express the mantissa. A single precision floating point number is made up of a 23-bit mantissa (fraction) and sign and a 7-bit exponent and sign, thus providing six significant decimal digits in the mantissa. A double precision floating point number is made up of a 55-bit mantissa and a 7-bit exponent and sign, thus providing 16 significant decimal digits in the mantissa. If either the mantissa or the exponent is negative, that part must be stored in two's complement form. The number must be in the approximate range of 10^{-38} 10^{+38}. When loaded into the accumulators, the A-register contains the most-significant word and the B-register contains the least-significant word.

Figure 3-1 also illustrates the octal notation for both single-length (16-bit) and double-length (32-bit) words. Each group of three bits, beginning at the right, is combined to form an octal digit. A single-length (16-bit) word can therefore be fully expressed by six octal digits and a double-length (32-bit) word can be fully expressed by 11 octal digits. Octal notation is not shown for byte or floating point formats, since bytes normally represent characters and floating point numbers are given in decimal form.

The range of representable numbers for single integer data is +32,767 to −32,768 (decimal) or +77,777 to −100,000 (octal). The range of representable numbers for double integer data is +2,147,483,647 to −2,147,483,648 (decimal) or +17,777,777,777 to −20,000,000,000 (octal).

3-3. MEMORY ADDRESSING

3-3. PAGING

The computer memory is logically divided into pages of 1,024 words each. A page is defined as the largest block of memory that can be directly addressed by the address bits of a single-length memory reference instruction. (Refer to paragraph 3-9.) These memory reference instructions use 10 bits (bits 0 through 9) to specify a memory address; thus, the page size is 1,024 locations (2000 octal). Octal addresses for each page, up to a maximum memory size of 32k words, are listed in Table 3-1.

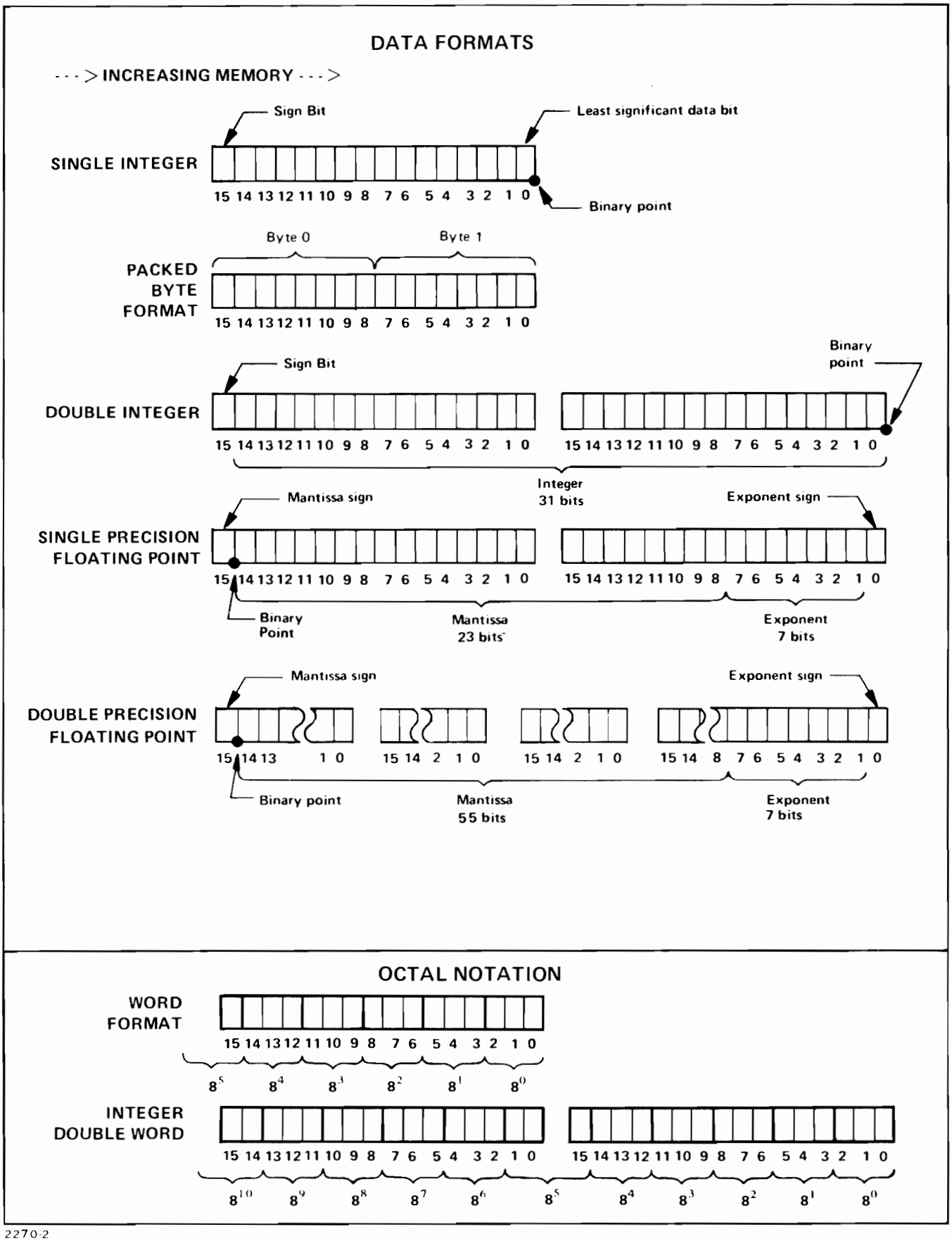
Provision is made to directly address one of two pages: page zero (the base page consisting of locations 00000 through 01777) and the current page (the page in which the instruction itself is located). Memory reference instructions reserve bit 10 to specify one or the other of these two pages. To address locations on any other page, indirect addressing is used as described in following paragraphs. Page references are specified by bit 10 as follows:

a. Logic 0 = Page Zero (Z).

b. Logic 1 = Current Page (C).

3-4. DIRECT AND INDIRECT ADDRESSING

All memory reference instructions reserve bit 15 to specify either direct or indirect addressing. For single-length memory reference instructions, bit 15 of the instruction word is used; for extended arithmetic memory reference instructions, bit 15 of the address word is used. Indirect addressing uses the address part of the instruction to ac-



2270-2

Figure 3-1. Data Formats and Octal Notation

Table 3-1. Memory Paging

MEMORY SIZE	PAGE	OCTAL ADDRESSES
	0	00000 to 01777
	1	02000 to 03777
	2	04000 to 05777
	3	06000 to 07777
	4	10000 to 11777
	5	12000 to 13777
	6	14000 to 15777
	7	16000 to 17777
	8	20000 to 21777
	9	22000 to 23777
	10	24000 to 25777
	11	26000 to 27777
	12	30000 to 31777
	13	32000 to 33777
	14	34000 to 35777
16K ↓	15	36000 to 37777
	16	40000 to 41777
	17	42000 to 43777
	18	44000 to 45777
	19	46000 to 47777
	20	50000 to 51777
	21	52000 to 53777
	22	54000 to 55777
	23	56000 to 57777
	24	60000 to 61777
	25	62000 to 63777
	26	64000 to 65777
	27	66000 to 67777
	28	70000 to 71777
	29	72000 to 73777
	30	74000 to 75777
	31	76000 to 77777

cess another word in memory, which is taken as the new memory reference for the same instruction. This new address word is a full 16 bits long: 15 address bits plus another direct/indirect bit. The 15-bit length of the address permits access to any location in logical memory. If bit 15 again specifies indirect addressing, still another address is obtained; thus, multistep indirect addressing may be done to any number of levels. The first address obtained that does not specify another indirect level becomes the effective address for the instruction. Direct or indirect addressing is specified by bit 15 as follows:

a. Logic 0 = Direct (D).

b. Logic 1 = Indirect (I).

3-5. MEMORY MAPPING

Memory mapping is a standard feature of the A900 computer and is used to access all locations of main memory. Memory mapping is provided by the Dynamic Mapping System described in Section IV.

3-5A. VIRTUAL MEMORY AREA

Under Virtual Memory Area (VMA) operation, a program may access two separate data areas, one being the 32k word logical address space, and the other being a virtual address space of up to 16M words. The virtual address space may be either memory-resident or disc-resident, and up to 1M words per program may reside in memory. This is accomplished through mapping pages of the logical address space to the virtual address space.

3-5B. CODE AND DATA SEPARATION

When Code and Data Separation (CDS) is enabled, a program's address space is partitioned into two separate address spaces: a code space and a data space of up to 31k words each. Opcodes and the operand pointers that follow the opcodes reside in code space, and variables and constants reside in data space. CDS instructions are provided that remap the code segment to other physical pages in memory, thus providing large program support. A program's code size may be up to 128 segments (each having 31k words of code), which may be either memory-resident or disc resident. The optional HP 92078A package for the RTE-A operating system provides software support for CDS.

3-5C. BASE-RELATIVE ADDRESSING

Under CDS, special hardware is used to access memory locations relative to a base register called the Q-register. When a memory address is in the range 2 through 1023, the Q-register value is added to produce an effective address in the data space. When CDS is enabled, code may not reside on the base page, which means that jump instructions may not jump to the base page.

3-6. RESERVED MEMORY LOCATIONS

The first 64 memory locations of these of the base page (octal address 00000 through 00077) are reserved as listed in Table 3-2. The first two locations are reserved as addresses for the two 16-bit accumulators (the A- and B-register). If options or input/output devices corresponding to locations 00020 through 00077 are not included in the system configuration, these locations can be used for programming purposes. The last 64 locations of the physical base page (octal addresses 1700 through 1777) are reserved for use by the Virtual Control Panel program for the string area.

Table 3-2. Reserved Memory Locations

MEMORY LOCATION	PURPOSE
00000	A-register address.
00001	B-register address.
00002-00003	Reserved.
00004	Power-fail interrupt.
00005	Memory multi-bit error interrupt.
00006	Time base generator interrupt.
00007	Memory protect interrupt.
00010	Unimplemented instruction interrupt.
00011	Reserved.
00012	Virtual Area Memory Interrupt.
00013	CDS Segment Interrupt.
00014-00017	Reserved.
00020-00077	Interrupt locations corresponding to interface card select codes.
01700-01777	VCP program string area.

3-7. NONEXISTENT MEMORY

Nonexistent memory is defined as those locations not physically implemented in the machine. For the A900 this definition refers to the memory array cards and not the cache memory. An attempted write to a nonexistent memory location only occurs in the cache; when the cache location is flushed, the data is not written to a memory array card. An attempted read from a nonexistent memory location that is **not** in the cache will return an all ones word (177777 octal); if the location was pulled into the cache on a previous access, the current cache value will be returned. If the nonexistent memory is protected, a memory protect interrupt will be generated.

The base set of instructions are classified according to format. The six formats used are illustrated in Figure 3-2 and described in the following paragraphs except for the DMS and CDS instructions, which are described in Sections IV and V. In all cases where a single bit is used to select one of two cases (e.g., D/I), the choice is made by coding a logic 0 or logic 1, respectively.

3-9. MEMORY REFERENCE INSTRUCTIONS

This class of instructions, which combines an instruction code and a memory address into one 16-bit word, is used to execute some function involving data in a specific memory location. Examples are storing, retrieving, and combining memory data to and from the accumulators (A- and B-registers) or causing the program to jump to a specified location in memory.

The memory cell referenced (i.e., the absolute address) is determined by a combination of 10 memory address bits (0 through 9) in the instruction word and 5 bits (10 through 14) from the P-register which holds the address of the instruction. This means that memory reference instructions can directly address any word in the current page; additionally, if the instruction is given in some location other than the base page (page zero), bit 10 (Z/C) of the instruction doubles the addressing range to 2,048 locations by allowing the selection of either page zero or the current page. (This causes bits 10 through 14 of the address to be set to zero instead of assuming the value

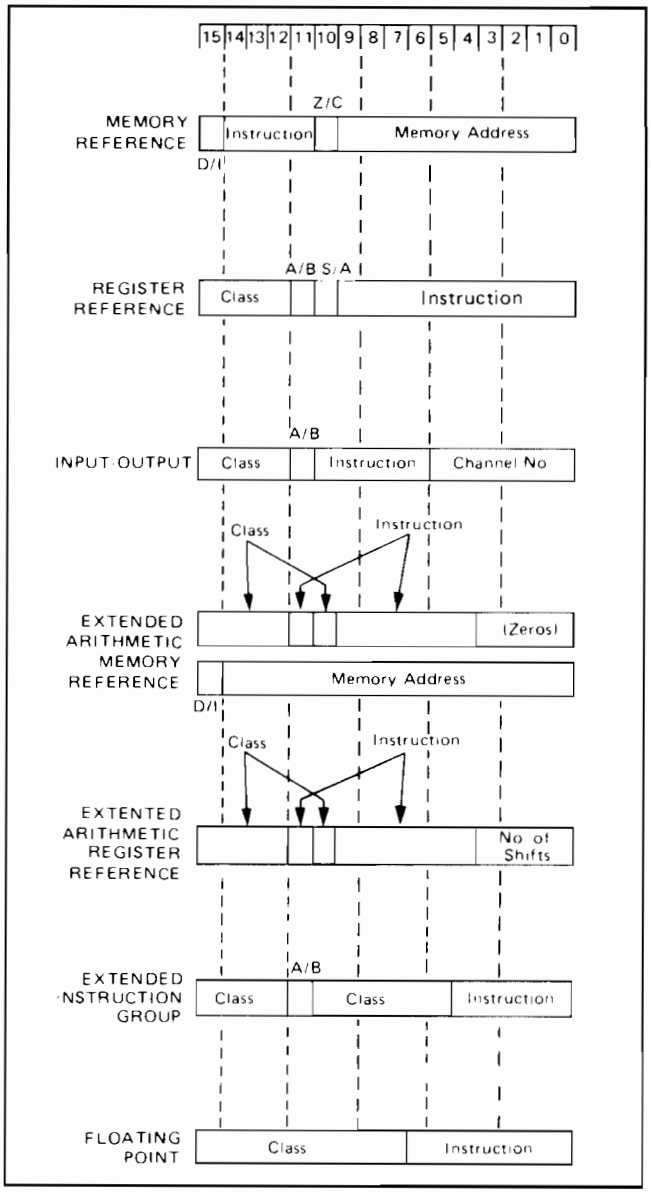


Figure 3-2. Base Set Instruction Formats

from the P-register.) This feature provides a convenient linkage between all pages of memory, since page zero can be reached directly from any other page. With CDS enabled, this feature becomes even more powerful as the base register is added to all base page references (addresses 2 through 1777 octal, or MRG instructions with Z/C = 0). This means that each single-word instruction has direct access to data on the current page or data up to 1k words relative to the base register.

As discussed under paragraph 3-4, bit 15 is used to specify direct or indirect memory addressing. Note also that since the A- and B-registers are addressable, any single-word memory reference instruction can apply to either of these registers as well as to memory cells. For example, an ADA 0001 instruction adds the contents of the B-register (address 0001) to the contents currently held in the A-register; specify page zero for these operations since the addresses of the A- and B-registers are on page zero.

3-10. REGISTER REFERENCE INSTRUCTIONS

In general, the register reference instructions manipulate bits in the A-register, B-register, and E-register; there is no reference to memory. This group includes 39 basic instructions which may be combined to form a one-word multiple instruction that can operate in various ways on the contents of the A-, B-, and E-registers. These 39 instructions are divided into two subgroups: the shift/rotate group (SRG) and the alter/skip group (ASG). The appropriate subgroup is specified by bit 10 (S/A). Typical operations are clear and/or complement a register, conditional skips, and register increment.

3-11. INPUT/OUTPUT INSTRUCTIONS

The input/output instructions use bits 6 through 11 for a variety of I/O instructions and bits 0 through 5 to apply the instructions either to a specific I/O channel (if the Global Register is disabled) or to an I/O card register. This provides the means of controlling all peripherals connected to the I/O channels and for transferring data to and from these peripherals. Included also in this group are instructions to control the interrupt system, overflow bit, and computer halt.

3-12. EXTENDED ARITHMETIC MEMORY REFERENCE INSTRUCTIONS

As the single-word memory reference instruction described previously, the extended arithmetic memory reference instructions include an instruction code and a memory address. In this case, however, two words are required. The first word specifies the extended arithmetic class (bits 12 through 15 and 10) and the instruction code (bits 4 through 9 and 11); bits 0 through 3 are not needed and are coded with zeros. The second word specifies the memory address of the operand. Since the full 15 bits are used for the address, this type of instruction may directly address any location in memory. If the CDS mode is enabled and the reference is to the base page, the base (Q) register will be added to the second word before referencing memory. As with all memory reference instructions, bit 15 is used to specify direct or indirect addressing. Operations performed by this class of instructions are integer multiply and divide (using double-length product and dividend) and double load and double store.

3-13. EXTENDED ARITHMETIC REGISTER REFERENCE INSTRUCTIONS

This class of instructions provides long shifts and rotates on the combined contents of the A- and B-registers. Bits 12 through 15 and 10 identify the instruction class; bits 4 through 9 and 11 specify the direction and type of shift; and bits 0 through 3 control the number of shifts, which can range from 1 to 16 places.

3-14. EXTENDED INSTRUCTIONS

The extended instructions include index register instructions, bit and byte manipulation instructions, and move and compare instructions. Instructions comprising the extended instruction group are one, two, or three words in length. The first word is always the instruction code; operand addresses are given in the words following the instruction code or in the A- and B-registers. The operand addresses are 15 bits long, with bit 15 (most-significant bit) generally indicating direct or indirect addressing.

3-15. FLOATING POINT INSTRUCTIONS

The floating point instructions allow addition, subtraction, multiplication, and division of floating point quantities. Conversion routines are provided for transforming numerical integer representations to/from floating point representations.

3-16. LANGUAGE INSTRUCTION SET

The language instruction set performs several frequently used high-level language operations, including parameter passing, array address calculations, and floating point conversion, packing, and rounding.

3-17. DOUBLE INTEGER INSTRUCTIONS

The double integer instructions allow arithmetic and test operations on 32-bit quantities. The data format for double integer values is shown in Figure 3-1.

3-18. VIRTUAL MEMORY INSTRUCTIONS

The virtual memory instructions perform accesses to Virtual Memory and the Extended Memory Area, which are extensions of logical memory.

3-19. OPERATING SYSTEM INSTRUCTIONS

The operating system instructions provide instructions for ascertaining the CPU and firmware identification, and instructions for interrupt conditions.

3-20. SCIENTIFIC INSTRUCTION SET

The Scientific Instruction Set performs nine single-precision and nine double-precision trigonometric and transcendental functions.

3-21. VECTOR INSTRUCTION SET

The Vector Instruction Set applies the floating point processing power of the A900 to highly efficient repetitive processing of vectors and matrices.

3-21A. CDS INSTRUCTIONS

The CDS instruction set includes instructions for examining and modifying the base (Q) register, bounds (Z) register, and CDS mode (C) register. This set also includes instructions for transferring control between subroutines (which may or may not be memory-resident).

All instructions that reference multi-word data (double integer, single and double precision floating point) as well as instructions using sequential addressing (DMA move instructions, .SETP and SFB) will have the base register added to the initial address if the instruction is base relative and CDS mode is enabled. Subsequent memory references are then executed sequentially.

Instructions that leave an address in a register upon completion (e.g., LBT, .ZFER, .SETP, MW00) will contain an address resolved for base relativity, incremented by the proper count.

3-22. BASE SET INSTRUCTION CODING

Machine language coding for the base set of instructions are provided in following paragraphs. Definitions for these instructions are grouped according to the instruction type.

Directly above each definition is a diagram showing the machine language coding for that instruction. The gray shaded bits code the instruction type and the gold shaded bits code the specific instruction. Unshaded bits are further defined in the introduction to each instruction type. The mnemonic code and instruction name are included above each diagram.

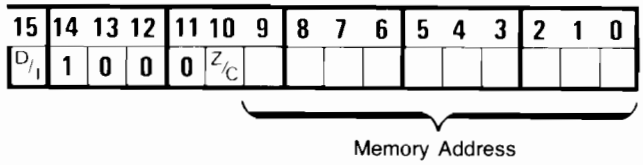
In all cases where an additional bit is used to specify a secondary function (D/I, Z/C, or H/C), the choice is made by coding a logic 0 or logic 1, respectively. In other words, a logic 0 codes D (direct addressing), Z (zero page), or H (hold flag); a logic 1 codes I (indirect addressing), C (current page), or C (clear flag).

3-23. MEMORY REFERENCE INSTRUCTIONS

The following 14 memory reference instructions execute a function involving data in memory. Bits 0 through 9 specify the affected memory location on a given memory page or, if indirect addressing is specified, the next address to be referenced. Indirect addressing may be continued to any number of levels; when bit 15 (D/I) is a logic 0 (specifying direct addressing), that location will be taken as the effective address. The A- and B-registers may be addressed as locations 00000 and 00001 (octal), respectively.

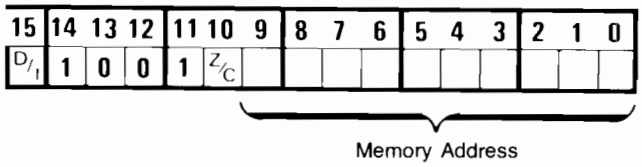
If bit 10 (Z/C) is a logic 1, the memory address is on the current page. If bit 10 is a logic 0, the memory address depends on whether CDS mode is enabled. If CDS mode is enabled, the base (Q) register will be added to bits 0 through 9 to provide the memory address. If CDS mode is not enabled, the memory address is on the base page (page 0). If the A- or B-register is addressed, bit 10 must be a logic 0 to specify page zero, unless the current page is page zero.

ADA ADD TO A



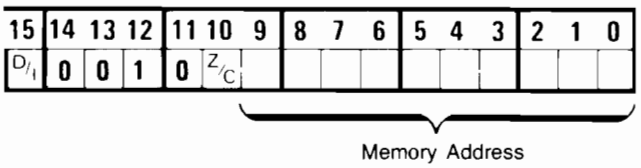
Adds the contents of the addressed memory location to the contents of the A-register. The sum remains in the A-register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit. (Extend and overflow examples are illustrated on page A-15.)

ADB ADD TO B



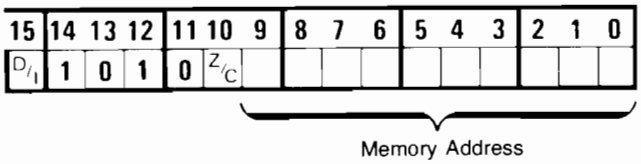
Adds the contents of the addressed memory location to the contents of the B-register. The sum remains in the B-register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit. (Extend and overflow examples are illustrated on page A-15.)

AND "AND" TO A



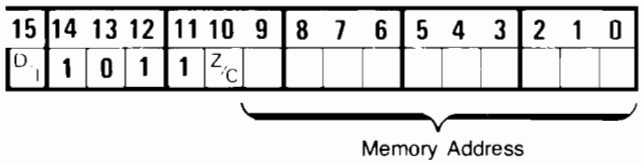
Combines the contents of the addressed memory location and the contents of the A-register by performing a logical "and" operation. The contents of the memory cell are unaltered.

CPA COMPARE TO A



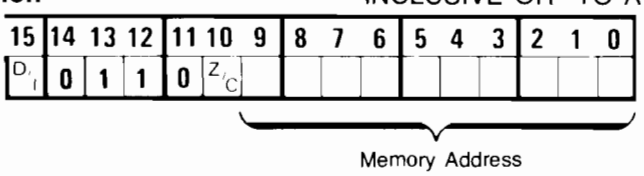
Compares the contents of the addressed memory location with the contents of the A-register. If the two 16-bit words are not identical, the next instruction is skipped; i.e., the P-register advances two counts instead of one count. If the two words are identical, the next sequential instruction is executed. Neither the A-register contents nor memory cell contents are altered.

CPB COMPARE TO B



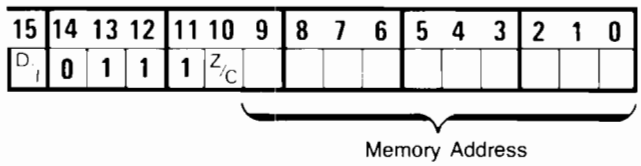
Compares the contents of the addressed memory location with the contents of the B-register. If the two 16-bit words are not identical, the next instruction is skipped; i.e., the P-register advances two counts instead of one count. If the two words are identical, the next sequential instruction is executed. Neither the B-register contents nor memory cell contents are altered.

IOR "INCLUSIVE OR" TO A



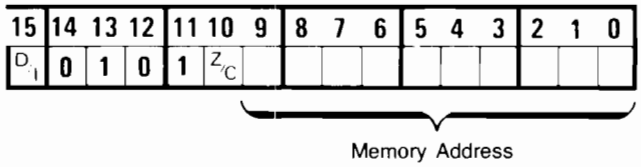
Combines the contents of the addressed memory location and the contents of the A-register by performing a logical "inclusive or" operation. The contents of the memory cell are unaltered.

ISZ INCREMENT AND SKIP IF ZERO



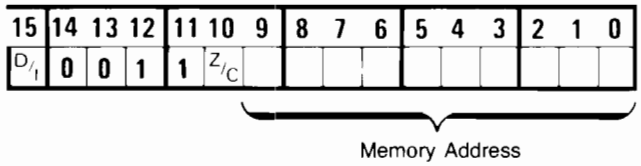
Adds one to the contents of the addressed memory location. If the result of this operation is zero (memory contents incremented from 177777 to 000000), the next instruction is skipped; i.e., the P-register is advanced two counts instead of one count. If the result of this operation is not zero, the next sequential instruction is executed. In either case, the incremented value is written back into the memory cell. Current page, direct addressing with this instruction produces undefined results when CDS is enabled.

JMP JUMP



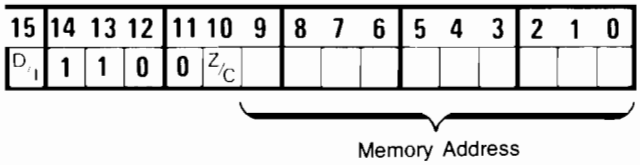
Transfers control to the addressed memory location. That is, a JMP causes the P-register count to set according to the memory address portion of the JMP instruction so that the next instruction will be read from that location.

JSB JUMP TO SUBROUTINE



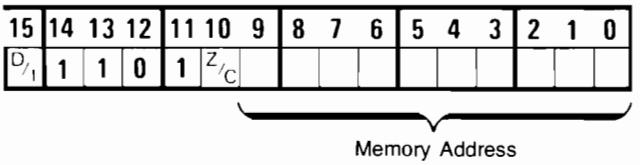
This instruction, executed in location P (P-register count), causes the computer control to jump unconditionally to the memory location (m) specified by the memory address portion of the JSB instruction. The contents of the P-register plus one (return address) is stored in memory location m, and the next instruction to be executed will be that contained in the next sequential memory location (m + 1). A return to the main program sequence at P + 1 will be effected by a JMP indirect through location m. This instruction produces undefined results when CDS is enabled.

LDA LOAD A



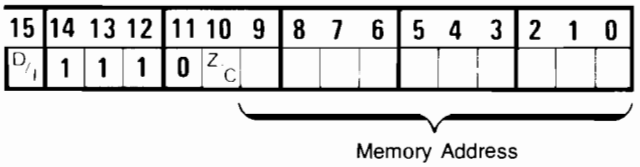
Loads the contents of the addressed memory location into the A-register. The contents of the memory cell are unaltered.

LDB LOAD B



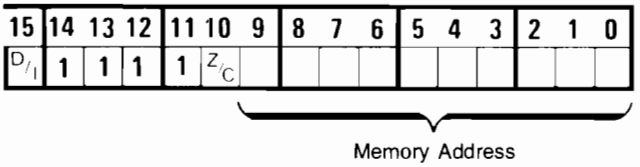
Loads the contents of the addressed memory location into the B-register. The contents of the memory cell are unaltered.

STA STORE A



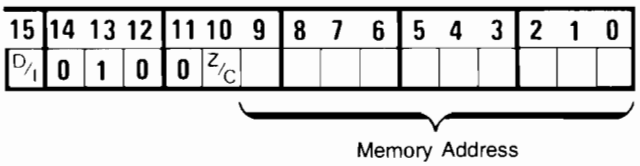
Stores the contents of the A-register in the addressed memory location. The previous contents of the memory cell are lost; the A-register contents are unaltered. Current page, direct addressing with this instruction produces undefined results when CDS is enabled.

STB STORE B



Stores the contents of the B-register in the addressed memory location. The previous contents of the memory cell are lost; the B-register contents are unaltered. Current page, direct addressing with this instruction produces undefined results when CDS is enabled.

XOR "EXCLUSIVE OR" TO A



Combines the contents of the addressed memory location and the contents of the A-register by performing a logical "exclusive or"operation. The contents of the memory cell are unaltered.

3-24. REGISTER REFERENCE INSTRUCTIONS

The 39 register reference instructions execute functions on data contained in the A-register, B-register, and E-register. These instructions are divided into two groups: the shift/rotate group (SRG) and the alter/skip group (ASG). In each group, several instructions may be combined into one word. Since the two groups perform separate and distinct functions, instructions from the two groups cannot be mixed. Unshaded bits in the coding diagrams are available for combining other instructions.

3-25. SHIFT/ROTATE GROUP. The 20 instructions in the shift/rotate group (SRG) are defined first; this group is specified by setting bit 10 to a logic 0. A comparison of the various shift/rotate functions are illustrated in Figure 3-3. Rules for combining instructions in this group are as follows (refer to Table 3-3):

a. Only one instruction can be chosen from each of the two multiple-choice columns.

b. References can be made to either the A-register or B-register, but not both.

c. Sequence of execution is from left to right.

d. In machine code, use zeros to exclude unwanted microinstructions.

e. Code a logic 1 in bit position 9 to enable shifts or rotates in the first position; code a logic 1 in bit position 4 to enable shifts or rotates in the second position.

f. The extend bit is not affected unless specifically stated. However, if a "rotate-with-E" instruction (ELA, ELB, ERA, or ERB) is coded but disabled by a logic 0 in bit position 9 and/or position 4, the E-register will be updated even though the A- or B-register contents are not affected; to avoid this situation, code a "no operation" (three zeros) in the first and/or second positions.

ALF ROTATE A LEFT FOUR

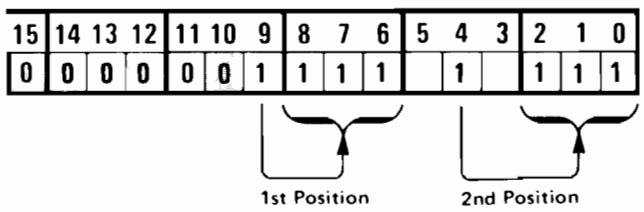
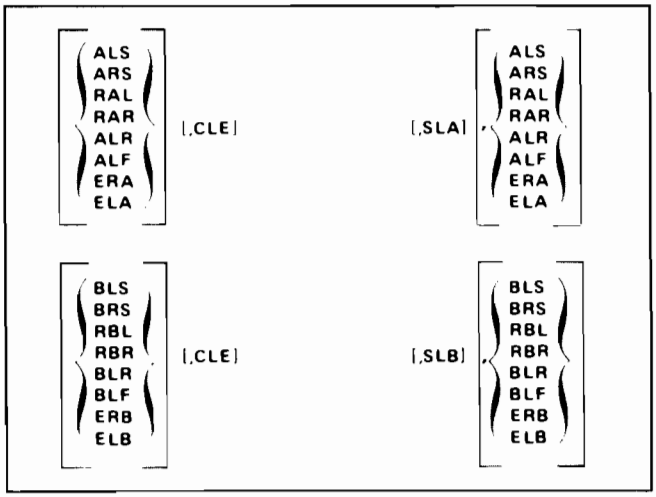


Table 3-3. Shift/Rotate Group Combining Guide



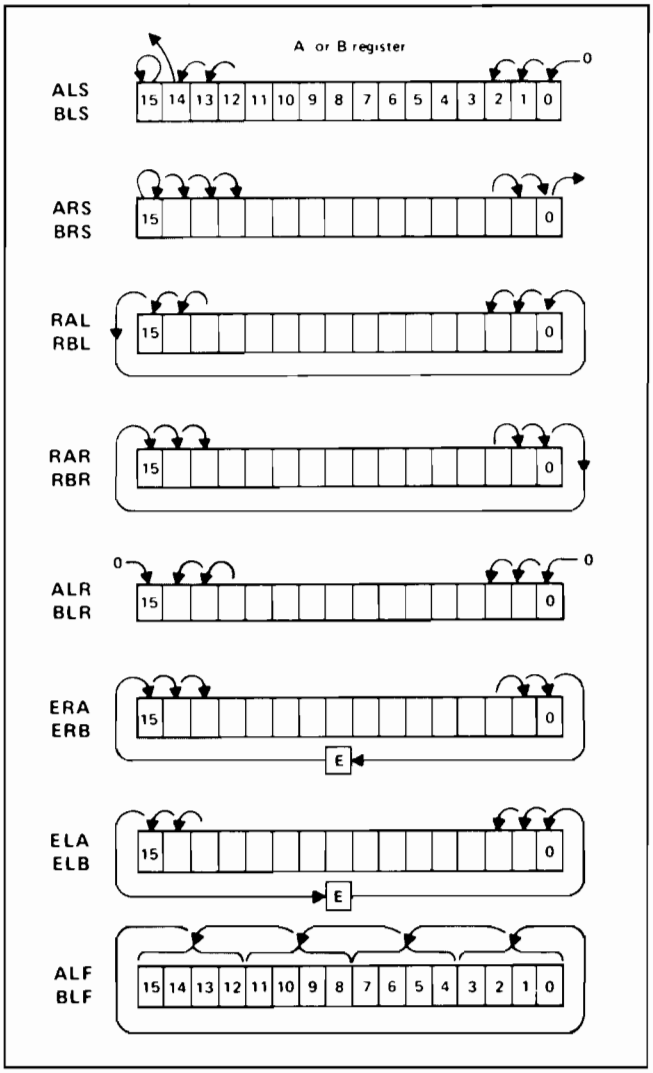
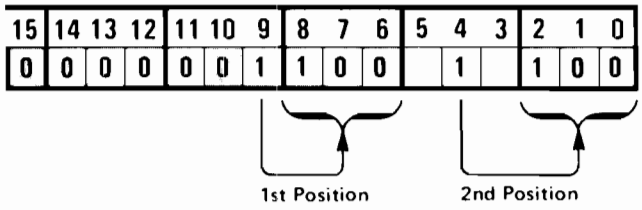


Figure 3-3. Shift and Rotate Functions

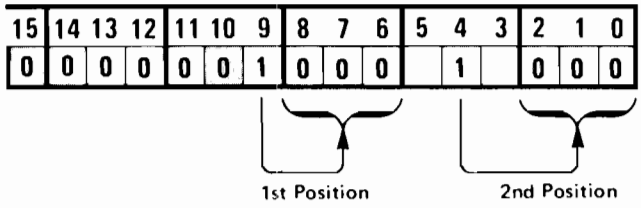
Rotates the A-register contents (all 16 bits) left four places. Bits 15, 14, 13, and 12 rotate around to bit positions 3, 2, 1, and 0, respectively. Equivalent to four successive RAL instructions.

ALR A LEFT SHIFT, CLEAR SIGN



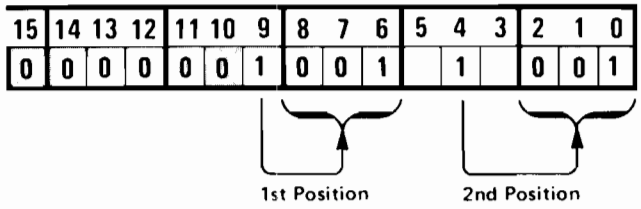
Shifts the A-register contents left one place and clears sign bit 15.

ALS A LEFT SHIFT



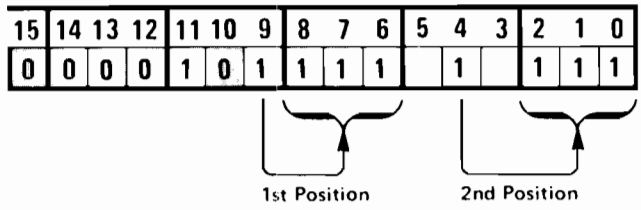
Arithmetically shifts the A-register contents left one place, 15 magnitude bits only; bit 15 (sign) is not affected. The bit shifted out of bit position 14 is lost; a logic 0 replaces vacated bit position 0.

ARS A RIGHT SHIFT



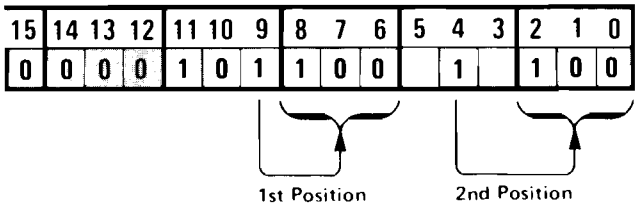
Arithmetically shifts the A-register contents right one place, 15 magnitude bits only; bit 15 (sign) is not affected. A copy of the sign bit is shifted into bit position 14; the bit shifted out of bit position 0 is lost.

BLF ROTATE B LEFT FOUR



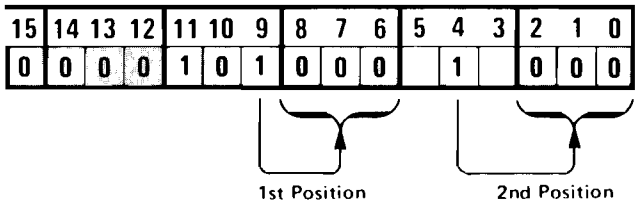
Rotates the B-register contents (all 16 bits) left four places. Bits 15, 14, 13, and 12 rotate around to bit positions 3, 2, 1, and 0, respectively. Equivalent to four successive RBL instructions.

BLR B LEFT SHIFT, CLEAR SIGN



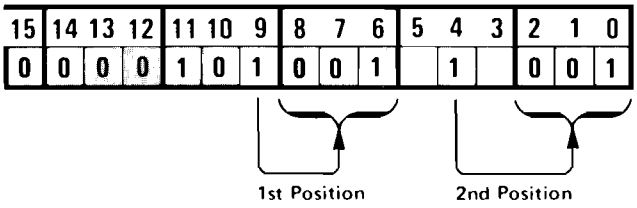
Shifts the B-register contents left one place and clears sign bit 15.

BLS B LEFT SHIFT



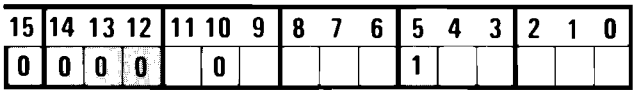
Arithmetically shifts the B-register contents left one place, 15 magnitude bits only; bit 15 (sign) is not affected. The bit shifted out of bit position 14 is lost; a logic 0 replaces vacated bit position 0.

BRS B RIGHT SHIFT



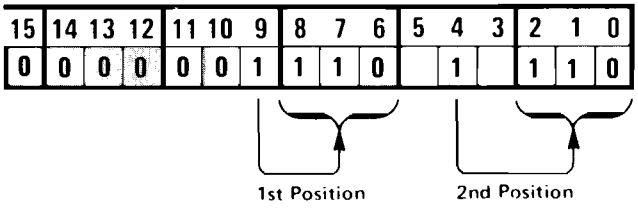
Arithmetically shifts the B-register contents right one place, 15 magnitude bits only; bit 15 (sign) is not affected. A copy of the sign bit is shifted into bit position 14; the bit shifted out of bit position 0 is lost.

CLE CLEAR E



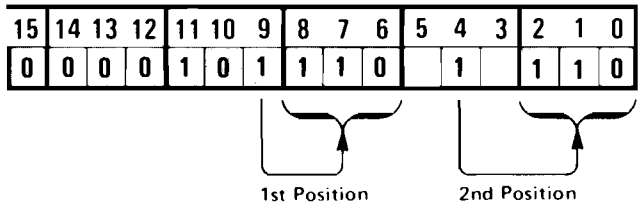
Clears the E-register; i.e., the extend bit becomes a logic 0.

ELA ROTATE E LEFT WITH A



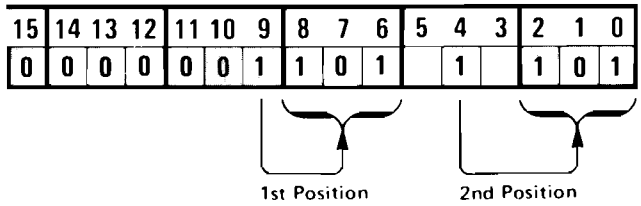
Rotates the E-register content left with the A-register contents (one place). The E-register content rotates into bit position 0; bit 15 rotates into the E-register.

ELB ROTATE E LEFT WITH B



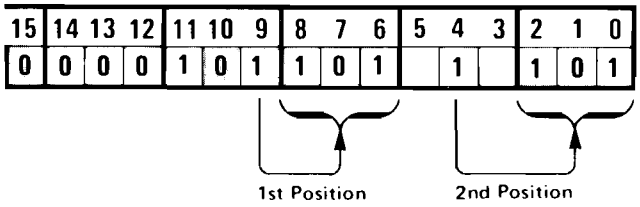
Rotates the E-register content left with the B-register contents (one place). The E-register content rotates into bit position 0; bit 15 rotates into the E-register.

ERA ROTATE E RIGHT WITH A



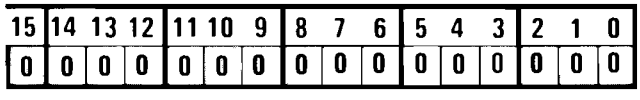
Rotates the E-register content right with the A-register contents (one place). The E-register content rotates into bit position 15; bit 0 rotates into the E-register.

ERB ROTATE E RIGHT WITH B



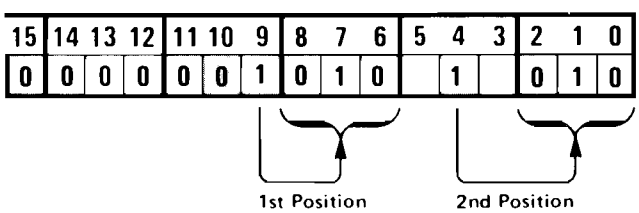
Rotates the E-register content right with the B-register contents (one place). The E-register content rotates into bit position 15; bit 0 rotates into the E-register.

NOP NO OPERATION



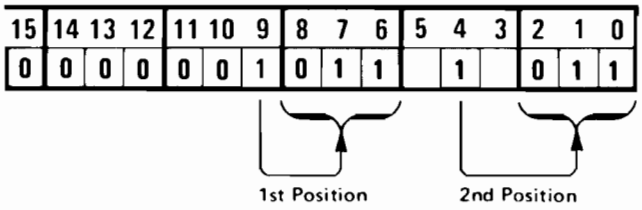
This all-zeros instruction causes a no-operation cycle.

RAL ROTATE A LEFT



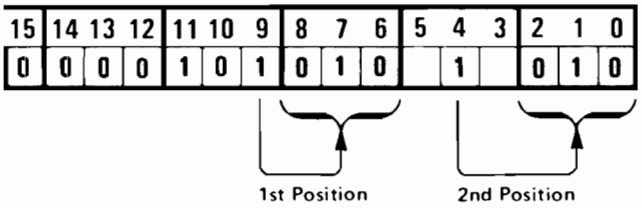
Rotates the A-register contents left one place (all 16 bits). Bit 15 rotates into bit position 0.

RAR ROTATE A RIGHT



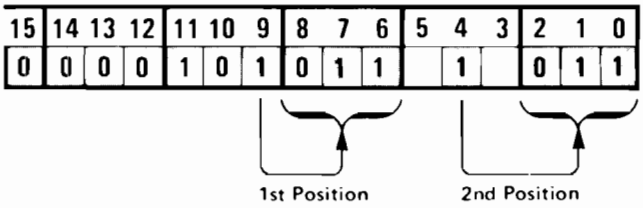
Rotates the A-register contents right one place (all 16 bits). Bit 0 rotates into bit position 15.

RBL ROTATE B LEFT



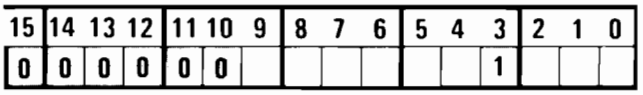
Rotates the B-register contents left one place (all 16 bits). Bit 15 rotates into bit position 0.

RBR ROTATE B RIGHT



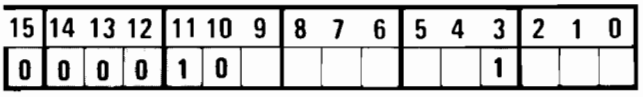
Rotates the B-register contents right one place (all 16 bits). Bit 0 rotates into bit position 15.

SLA SKIP IF LSB OF A IS ZERO



Skips the next instruction if the least-significant bit (bit 0) of the A-register is a logic 0.

SLB SKIP IF LSB OF B IS ZERO



Skips the next instruction if the least-significant bit (bit 0) of the B-register is a logic 0.

3-26. ALTER/SKIP GROUP. The 19 instructions comprising the alter/skip group (ASG) are defined next. This group is specified by setting bit 10 to a logic 1. Rules for combining instructions are as follows (refer to Table 3-4):

a. Only one instruction can be chosen from each of the two multiple-choice columns.

b. References can be made to either the A-register or B-register, but not both.

c. Sequence of execution is from left to right.

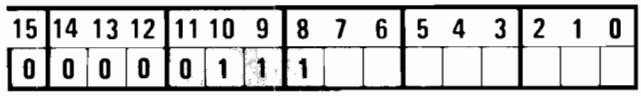
d. If two or more skip functions are combined, the skip function will occur if either or both conditions are met. One exception exists: refer to the RSS instruction.

e. In machine code, use zeros to exclude unwanted instructions.

Table 3-4. Alter/Skip Group Combining Guide

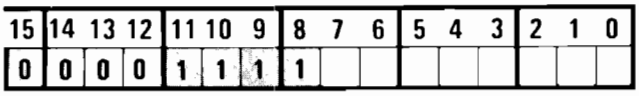
[CLA / CMA / CCA]	[,SEZ]	[CLE / CME / CCE]	[,SSA]	[,SLA]	[,INA]	[,SZA]	[,RSS]
[CLB / CMB / CCB]	[,SEZ]	[CLE / CME / CCE]	[,SSB]	[,SLB]	[,INB]	[,SZB]	[,RSS]

CCA CLEAR AND COMPLEMENT A



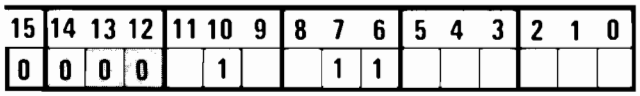
Clears and complements the A-register contents; i.e., the contents of the A-register become 177777 (octal). This is the two's complement form of −1.

CCB CLEAR AND COMPLEMENT B



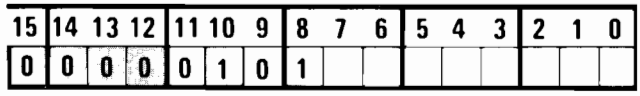
Clears and complements the B-register contents; i.e., the contents of the B-register become 177777 (octal). This is the two's complement form of −1.

CCE CLEAR AND COMPLEMENT E



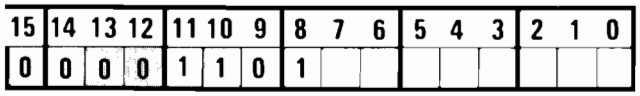
Clears and complements the E-register content (extend bit); i.e., the extend bit becomes a logic 1.

CLA CLEAR A



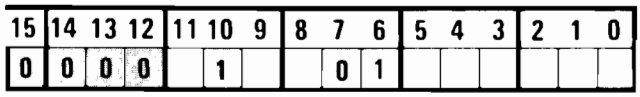
Clears the A-register; i.e., the contents of the A-register becomes 000000 (octal).

CLB CLEAR B



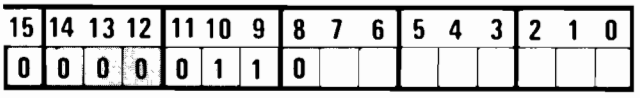
Clears the B-register; i.e., the contents of the B-register become 000000 (octal).

CLE CLEAR E



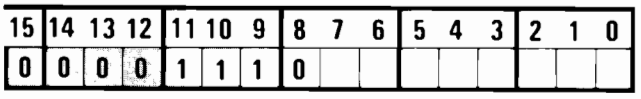
Clears the E-register; i.e., the extend bit becomes a logic 0.

CMA COMPLEMENT A



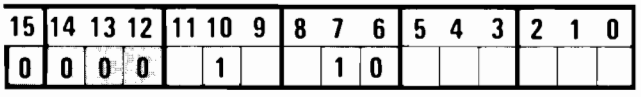
Complements the A-register contents (one's complement).

CMB COMPLEMENT B



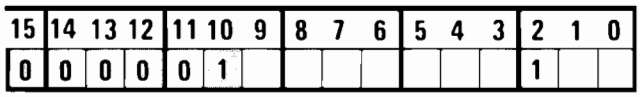
Complements the B-register contents (one's complement).

CME COMPLEMENT E



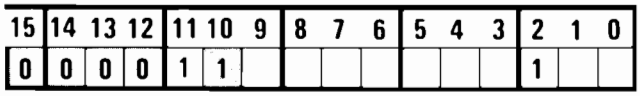
Complements the E-register content (extend bit).

INA INCREMENT A



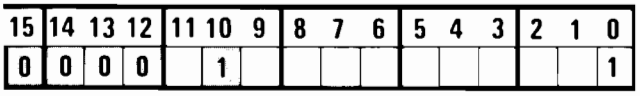
Increments the A-register by one. The overflow bit will be set if an increment of the largest positive number (077777 octal) is made. The extend bit will be set if an all-ones word (177777 octal) is incremented.

INB INCREMENT B



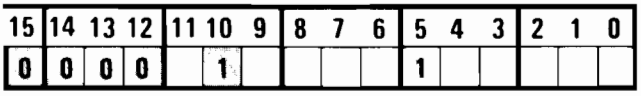
Increments the B-register by one. The overflow bit will be set if an increment of the largest positive number (077777 octal) is made. The extend bit will be set if an all-ones word (177777 octal) is incremented.

RSS REVERSE SKIP SENSE



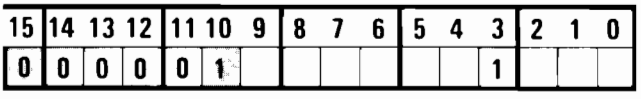
Skip occurs for any of the following skip instructions, if present, when the non-zero condition is met. An RSS without a skip instruction in the word causes an unconditional skip. If a word with RSS also includes both SSA and SLA (or SSB and SLB), bits 15 and 0 must both be logic 1's for a skip to occur; in all other cases, a skip occurs if one or more skip conditions are met.

SEZ SKIP IF E IS ZERO



Skips the next instruction if the E-register content (extend bit) is a logic 0.

SLA SKIP IF LSB OF A IS ZERO



Skips the next instruction if the least-significant bit (bit 0) of the A-register is a logic 0; i.e., skips if an even number is in the A-register.

SLB SKIP IF LSB OF B IS ZERO

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

Skips the next instruction if the least-significant bit (bit 0) of the B-register is a logic 0; i.e., skips if an even number is in the B-register.

SSA SKIP IF SIGN OF A IS ZERO

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

Skips the next instruction if the sign bit (bit 15) of the A-register is a logic 0; i.e., skips if a positive number is in the A-register.

SSB SKIP IF SIGN OF B IS ZERO

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

Skips the next instruction if the sign bit (bit 15) of the B-register is a logic 0; i.e., skips if a positive number is in the B-register.

SZA SKIP IF A IS ZERO

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

Skips the next instruction if the A-register contents are zero (16 zeros).

SZB SKIP IF B IS ZERO

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

Skips the next instruction if the B-register contents are zero (16 zeros).

3-27. INPUT/OUTPUT INSTRUCTIONS

The following input/output instructions provide the capability of setting, clearing or testing the flag and control bits associated with DMA, programmed I/O, interrupts, memory protect, time base generator, parity error, Global Register, and overflow. I/O instructions with select codes of seven or less have various functions. (Refer to Table 5-3 for further information regarding specific select-code functions.) I/O instructions permit data transfer between the A- and B-registers and either specific I/O devices or between registers associated with memory protect or interrupts. The various registers and I/O devices are addressed by means of their register numbers and select codes.

Bit 11, where relevant, specifies the A- or B-register or distinguishes between set control and clear control; otherwise, bit 11 may be a logic 0 or a logic 1 without affecting the instruction (although the assembler will assign zeros in this case). In those instructions where bit position 9 includes the letters H/C, the programmer has the choice of holding (logic 0) or clearing (logic 1) the device flag after executing the instruction. (Exception: the H/C bit associated with instructions SOC and SOS holds or clears the overflow bit instead of the device flag.) Note that this H/C option is not supported on many of the I/O instructions with select code less than 10 octal.

Bits 8, 7, and 6, specify the appropriate I/O instruction. When the Global Register is enabled, bits 5 through 0 apply the instruction to a register on the I/O card whose select code is in the Global Register. (The Global Register is discussed further in paragraph 7-4).

NOTE

Execution of I/O instructions is inhibited when the memory protect feature is enabled. Refer to paragraph 6-3.

The following instruction descriptions assume that the Global Register is disabled and, therefore, the instructions are addressed to a select code.

CLC CLEAR CONTROL

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

Select Code or Register Number

Clears the control bit (Control 30) of the selected I/O channel or function. This turns off the specific device channel and prevents it from interrupting. A CLC 00 instruction clears the control bits from select code 20 upward, effectively turning off all I/O devices.

CLF CLEAR FLAG

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

Select Code or Register Number

Clears the flag (Flag 30) of the selected I/O channel or function. A CLF 00 instruction disables the interrupt system for the time base generator and all interface cards; this does not affect the status of the individual channel flags.

CLO CLEAR OVERFLOW

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

Clears the overflow bit.

HLT HALT

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

Select Code or Register Number

Halts the computer, holds or clears the flag of the selected I/O channel, and invokes the virtual control panel program. The HLT instruction will be contained in the T-register, which is displayed on the VCP when the VCP program starts executing. The P-register (also displayed) will normally contain the HLT location plus one. Note that if break is not enabled on any I/O card, the HLT instruction has no effect.

LIA LOAD INTO A

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

Register Number

Loads the contents of the addressed I/O special function register into the A-register.

LIB LOAD INTO B

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

Register Number

Loads the contents of the addressed I/O special function register into the B-register.

MIA MERGE INTO A

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

Register Number

By executing a logical "inclusive or" function, merges the contents of the addressed I/O special function register into the A-register.

MIB MERGE INTO B

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

Register Number

By executing a logical "inclusive or" function, merges the contents of the addressed I/O special function register into the B-register.

OTA OUTPUT A

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

Register Number

Outputs the contents of the A-register to the addressed I/O special function register. The contents of the A-register are not altered.

OTB OUTPUT B

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

Register Number

Outputs the contents of the B-register to the addressed I/O special function register. The contents of the B-register are not altered.

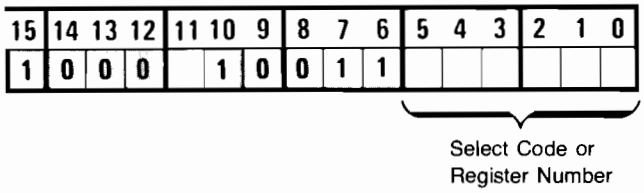
SFC SKIP IF FLAG CLEAR

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

Select Code or Register Number

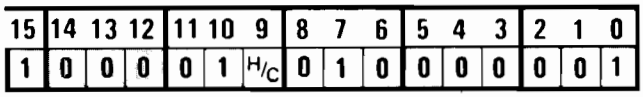
Skips the next programmed instruction if the flag (Flag 30) of the selected channel is clear (device busy).

SFS SKIP IF FLAG SET



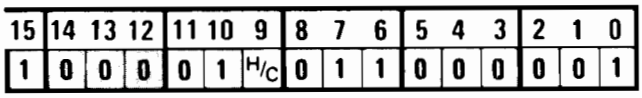
Skips the next programmed instruction if the flag (Flag 30) of the selected channel is set (device ready).

SOC SKIP IF OVERFLOW CLEAR



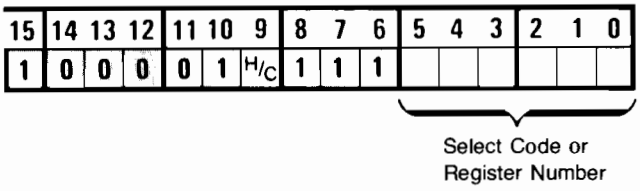
Skips the next programmed instruction if the overflow bit is clear. Use the H/C (bit 9) to either hold or clear the overflow bit following the completion of this instruction (whether the skip is taken or not).

SOS SKIP IF OVERFLOW SET



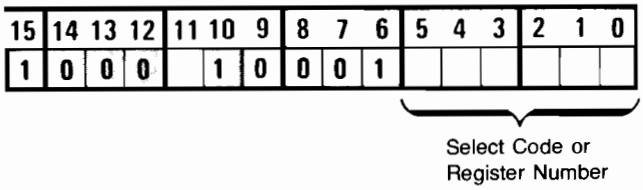
Skips the next programmed instruction if the overflow bit is set. Use the H/C bit (bit 9) to either hold or clear the overflow bit following the completion of this instruction (whether the skip is taken or not).

STC SET CONTROL



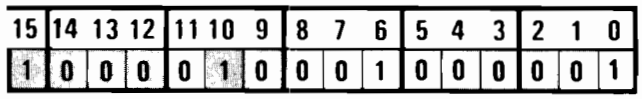
Sets the control bit (Control 30) of the selected I/O channel or function.

STF SET FLAG



Sets the flag (Flag 30) of the selected I/O channel or function. An STF 00 instruction enables the interrupt system for the time base generator and all interface cards.

STO SET OVERFLOW



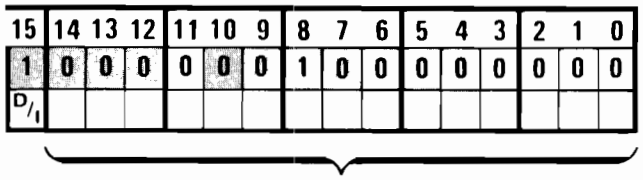
Sets the overflow bit.

3-28. EXTENDED ARITHMETIC MEMORY REFERENCE INSTRUCTIONS

The four extended arithmetic memory reference instructions provide for integer multiply and divide and for loading and storing double-length words to and from the A- and B-registers. The complete instruction requires two words: one for the instruction code and one for the address. When stored in memory, the instruction word is the first to be fetched; the address word is in the next sequential location.

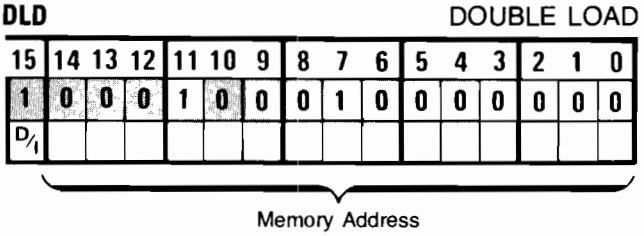
Since 15 bits are available for the address, these instructions can directly address any location in memory. As for all memory reference instructions, indirect addressing to any number of levels may also be used. A logic 0 in bit position 15 specifies direct addressing; a logic 1 specifies indirect addressing.

DIV DIVIDE



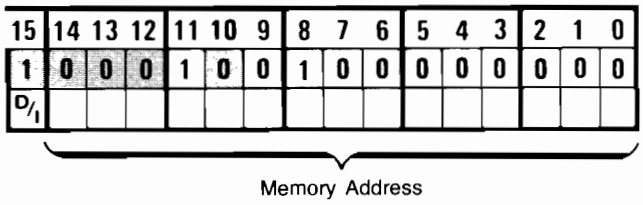
Divides a double-word integer in the combined B- and A-registers by a 16-bit integer in the addressed memory location. The result is a 16-bit integer quotient in the A-register and a 16-bit integer remainder in the B-register. Overflow can result from an attempt to divide by zero, or from an attempt to divide by a number too small for the dividend. In the former case (divide by zero), the division will not be attempted and the B- and A-register contents will be unchanged except that a negative quantity will be made positive. In the latter case (divisor too small), the execution will be attempted with unpredictable results left in the B- and A-registers. If there is no divide error, the overflow bit is cleared.

DLD DOUBLE LOAD



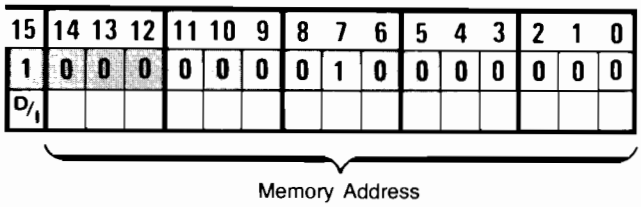
Loads the contents of addressed memory location m (and m + 1) into the A- and B-registers, respectively. If m is base relative and CDS mode is enabled, the base register will be added to m and the references will come from m+Q and m+Q+1 (even if m+1 is not base relative).

DST DOUBLE STORE



Stores the double-word quantity in the A- and B-registers into addressed memory locations m (and m + 1), respectively. If m is base relative and CDS mode is enabled, the base register will be added to m and the references will come from m+Q and m+Q+1 (even if m+1 is not base relative).

MPY MULTIPLY



Multiplies a 16-bit integer in the A-register by a 16-bit integer in the addressed memory location. The resulting double-length integer product resides in the B- and A-registers, with the B-register containing the sign bit and the most-significant 15 bits of the quantity. The A-register may be used as an operand (i.e., memory address 0), resulting in an arithmetic square. The instruction clears the overflow bit.

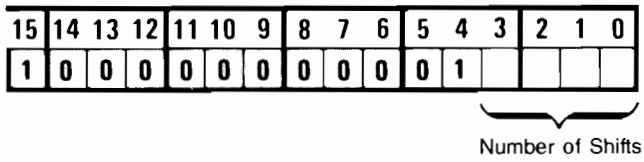
3-29 EXTENDED ARITHMETIC REGISTER REFERENCE INSTRUCTIONS

The six extended arithmetic register reference instructions provide various types of shifting operations on the combined contents of the B- and A-registers. The B-register is considered to be to the left (most-significant word) and the A-register is considered to be to the right (least-significant word). An example of each type of shift operation is illustrated in Figure 3-4.

The complete instruction is given in one word and includes four bits (unshaded) to specify the number of shifts (1 to 16). By viewing these four bits as a binary-coded number, the number of shifts is easily expressed; i.e., binary-coded 1 = 1 shift, binary-coded 2 = 2 shifts . . . binary-coded 15 = 15 shifts. The maximum number of 16 shifts is coded with four zeros, which essentially exchanges the contents of the B- and A-registers.

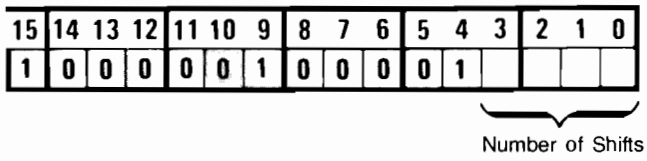
The extend bit is not affected by any of the following instructions. Except for the arithmetic shifts, overflow also is not affected.

ASL ARITHMETIC SHIFT LEFT



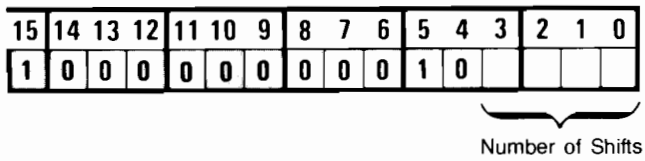
Arithmetically shifts the combined contents of the B- and A-registers left n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated low-order positions of the A-register. The sign bit is not affected, and data bits are lost out of bit position 14 of the B-register. If any one of the lost bits is a significant data bit ("1" for positive numbers, "0" for negative numbers), the overflow bit will be set; otherwise, overflow will be cleared during execution. See ASL example in Figure 3-4. Note that two additional shifts in this example would cause an error by losing a significant '1'.

ASR ARITHMETIC SHIFT RIGHT



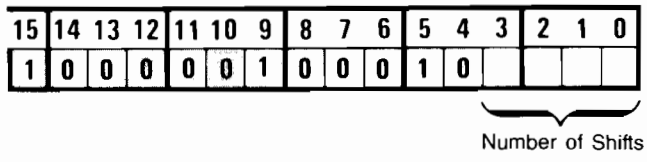
Arithmetically shifts the combined contents of the B- and A-registers right n places. The value of n may be any number from 1 through 16. The sign bit is unchanged and is extended into bit positions vacated by the right shift. Data bits shifted out of the least-significant end of the A-register are lost. Since overflow cannot occur, the instruction clears the overflow bit.

LSL LOGICAL SHIFT LEFT



Logically shifts the combined contents of the B- and A-registers left n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated low-order bit positions of the A-register; data bits are lost out of the high-order bit positions of the B-register.

LSR LOGICAL SHIFT RIGHT



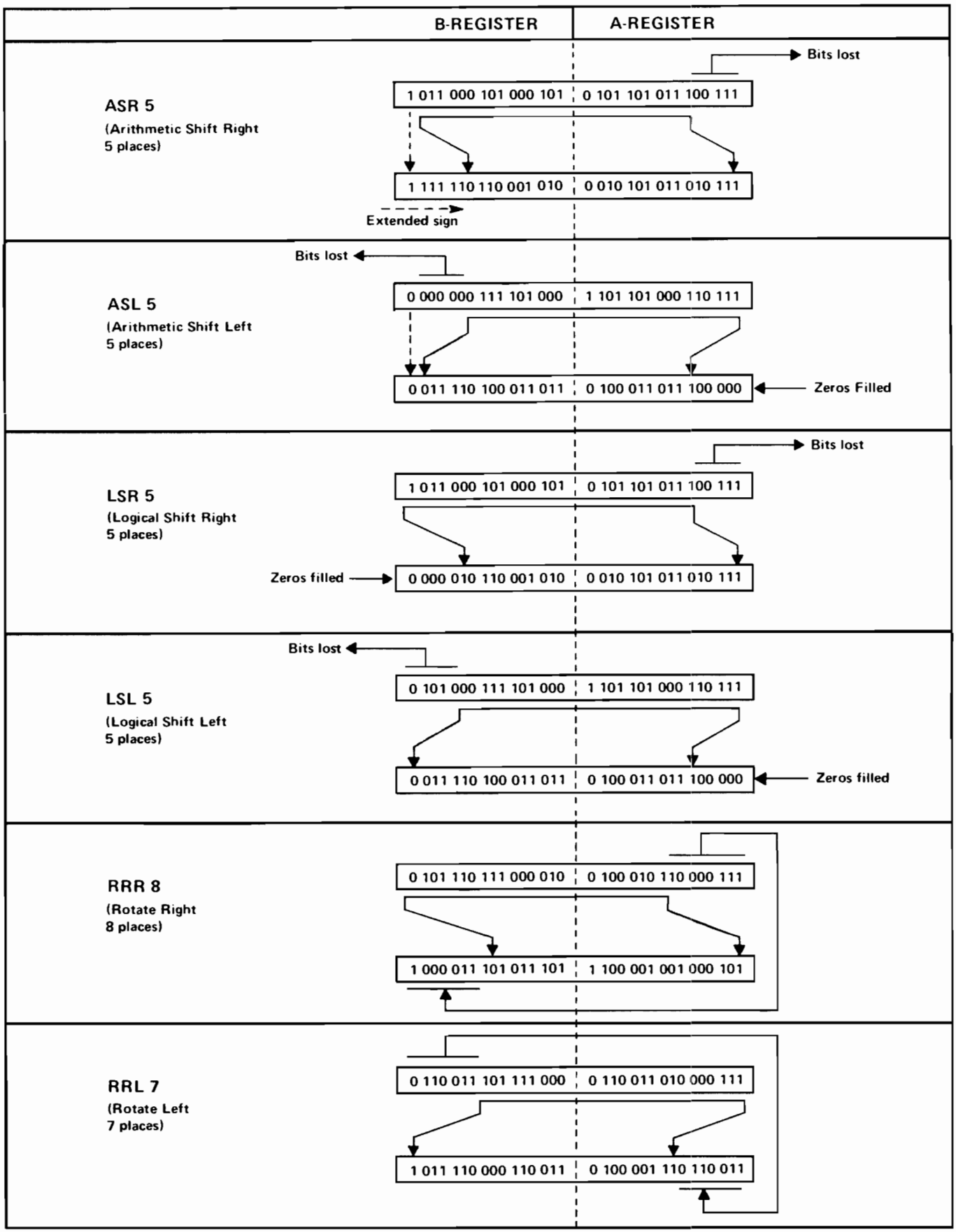
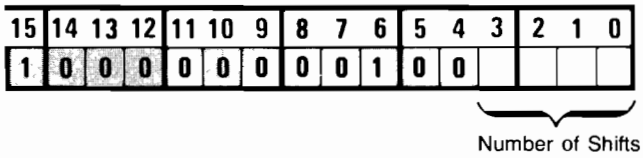


Figure 3-4. Example of Double-Word Shifts and Rotates

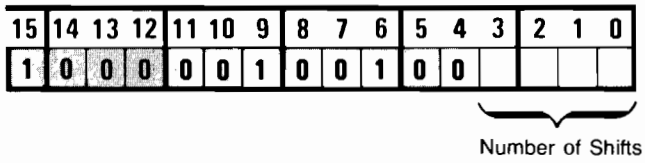
Logically shifts the combined contents of the B- and A-registers right n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated high-order bit positions of the B-register; data bits are lost out of the low-order bit positions of the A-register.

RRL ROTATE LEFT



Rotates the combined contents of the B- and A-registers left n places. The value of n may be any number from 1 through 16. No bits are lost or filled in. Data bits shifted out of the high-order end of the B-register are rotated around to enter the low-order end of the A-register.

RRR ROTATE RIGHT

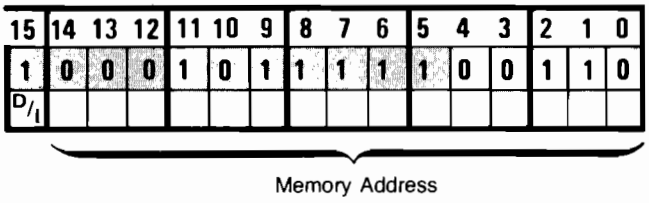


Rotates the combined contents of the B- and A-registers right n places. The value of n may be any number from 1 through 16. No bits are lost or filled in. Data bits shifted out of the low-order end of the A-register are rotated around to enter the high-order end of the B-register.

3-30. EXTENDED INSTRUCTION GROUP

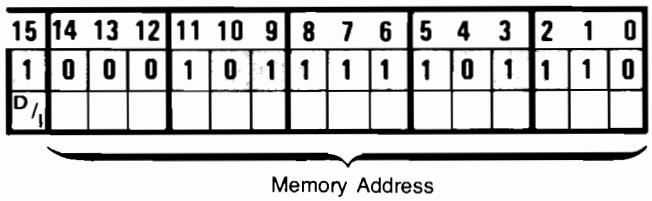
3-31. INDEX/REGISTER INSTRUCTIONS. The index registers (X and Y) are two 16-bit registers accessible by the following instructions.

ADX ADD MEMORY TO X



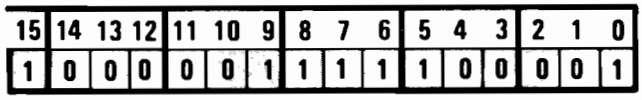
Adds the contents of the addressed memory location to the contents of the X-register. The sum remains in the X-register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit.

ADY ADD MEMORY TO Y



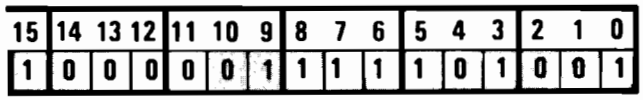
Adds the contents of the addressed memory location to the contents of the Y-register. The sum remains in the Y-register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit.

CAX COPY A TO X



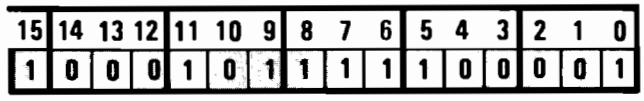
Copies the contents of the A-register into the X-register. The contents of the A-register are unaltered.

CAY COPY A TO Y



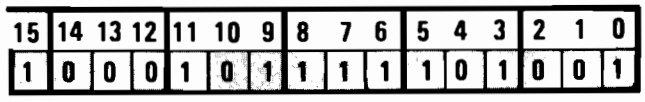
Copies the contents of the A-register into the Y-register. The contents of the A-register are unaltered.

CBX COPY B TO X



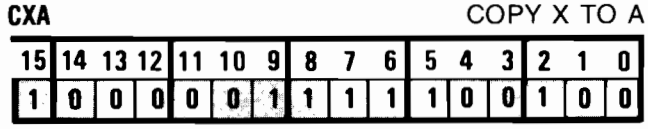
Copies the contents of the B-register into the X-register. The contents of the B-register are unaltered.

CBY COPY B TO Y



Copies the contents of the B-register into the Y-register. The contents of the B-register are unaltered.

CXA COPY X TO A



Copies the contents of the X-register into the A-register. The contents of the X-register are unaltered.

CXB COPY X TO B

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

Copies the contents of the X-register into the B-register. The contents of the X-register are unaltered.

CYA COPY Y TO A

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

Copies the contents of the Y-register into the A-register. The contents of the Y-register are unaltered.

CYB COPY Y TO B

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

Copies the contents of the Y-register into the B-register. The contents of the Y-register are unaltered.

DSX DECREMENT X AND SKIP IF ZERO

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

Subtracts one from the contents of the X-register. If the result of this operation is zero (X-register decremented from 000001 to 000000), the next instruction is skipped; i.e., the P-register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

DSY DECREMENT Y AND SKIP IF ZERO

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

Subtracts one from the contents of the Y-register. If the result of this operation is zero (Y-register decremented from 000001 to 000000), the next instruction is skipped; i.e., the P-register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

ISX INCREMENT X AND SKIP IF ZERO

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

Adds one to the contents of the X-register. If the result of this operation is zero (X-register rolls over to 000000 from 177777), the next instruction is skipped; i.e., the P-register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

ISY INCREMENT Y AND SKIP IF ZERO

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

Adds one to the contents of the Y-register. If the result of this operation is zero (Y-register rolls over to 000000 from 177777), the next instruction is skipped; i.e., the P-register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

LAX LOAD A INDEXED BY X

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

Operand Address

Loads the A-register with the contents of the memory location indicated by the effective address, which is computed by adding the contents of the X-register to the operand address. The X-register and memory contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

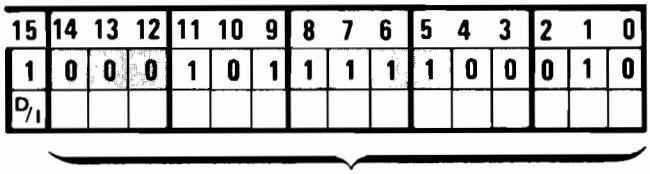
LAY LOAD A INDEXED BY Y

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

Operand Address

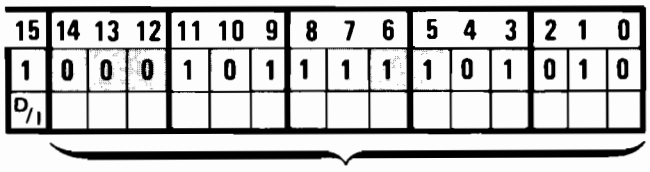
Loads the A-register with the contents of the memory location indicated by the effective address, which is computed by adding the contents of the Y-register to the operand address. The Y-register and memory contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

LBX LOAD B INDEXED BY X



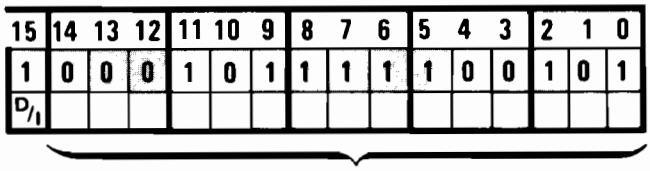
Loads the B-register with the contents of the memory location indicated by the effective address, which is computed by adding the contents of the X-register to the operand address. The X-register and memory contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

LBY LOAD B INDEXED BY Y



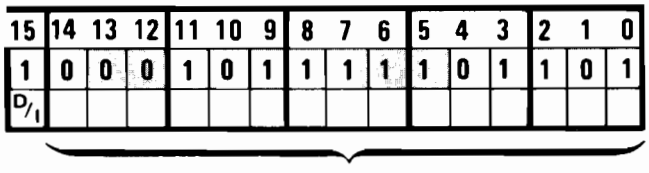
Loads the B-register with the contents of the memory location indicated by the effective address, which is computed by adding the contents of the Y-register to the operand address. The X-register and memory contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

LDX LOAD X FROM MEMORY



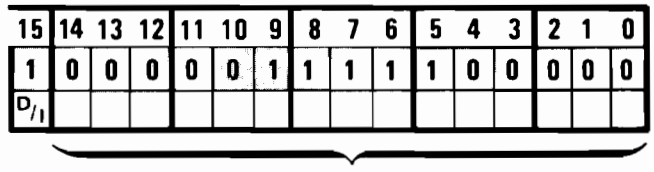
Loads the contents of the addressed memory location into the X-register. The A- and B-registers may be addressed as locations 00000 and 00001, respectively; however, if it is desired to load from the A- or B-register, copy instructions CAX or CBX should be used since they are more efficient.

LDY LOAD Y FROM MEMORY



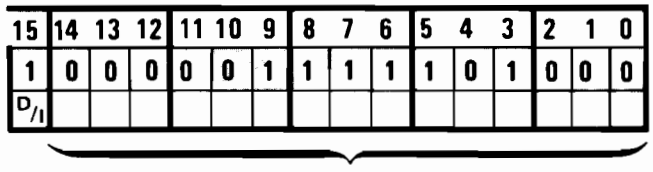
Loads the contents of the addressed memory location into the Y-register. The A- and B-registers may be addressed as locations 00000 and 00001, respectively; however, if it is desired to load from the A- or B-register, copy instructions CAY or CBY should be used since they are more efficient.

SAX STORE A INDEXED BY X



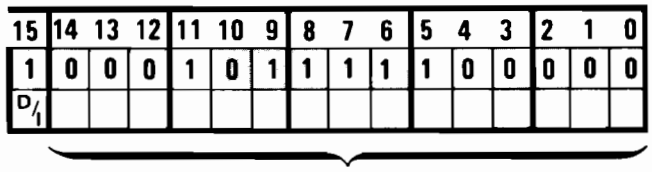
Stores the contents of the A-register into the memory location indicated by the effective address, which is computed by adding the contents of the X-register to the operand address. The A- and X-register contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

SAY STORE A INDEXED BY Y



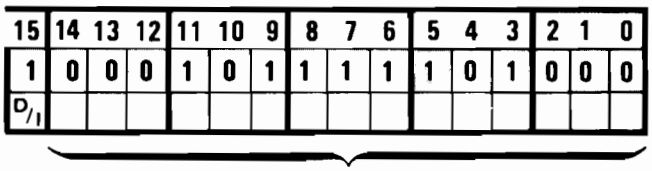
Stores the contents of the A-register into the memory location indicated by the effective address, which is computed by adding the contents of the Y-register to the operand address. The A- and Y-register contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

SBX STORE B INDEXED BY X



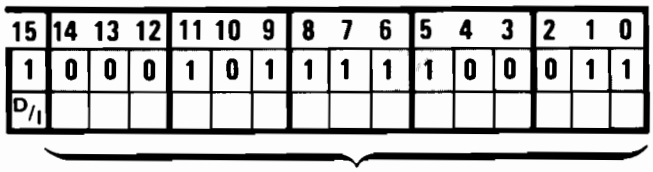
Stores the contents of the B-register into the memory location indicated by the effective address, which is computed by adding the contents of the X-register to the operand address. The B- and X-register contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

SBY STORE B INDEXED BY Y



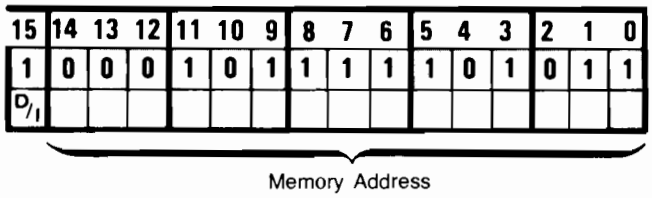
Stores the contents of the B-register into the memory location indicated by the effective address, which is computed by adding the contents of the Y-register to the operand address. The B- and Y-register contents are not altered. Indirect addressing and base relativity are resolved before indexing; bit 15 of the effective address is ignored.

STX STORE X TO MEMORY



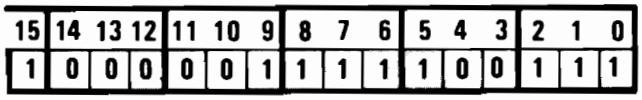
Stores the contents of the X-register into the addressed memory location. The A- and B-registers may be addressed as locations 00000 and 00001, respectively. The X-register contents are not altered.

STY STORE Y TO MEMORY



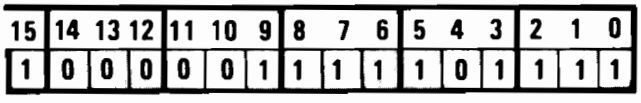
Stores the contents of the Y-register into the addressed memory location. The A- and B-registers may be addressed as locations 00000 and 00001, respectively. The Y-register contents are not altered.

XAX EXCHANGE A AND X



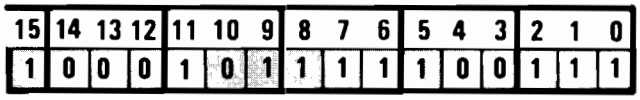
Exchanges the contents of the A- and X-registers.

XAY EXCHANGE A AND Y



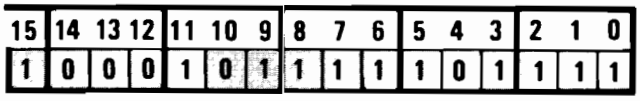
Exchanges the contents of the A- and Y-registers.

XBX EXCHANGE B AND X



Exchanges the contents of the B- and X-registers.

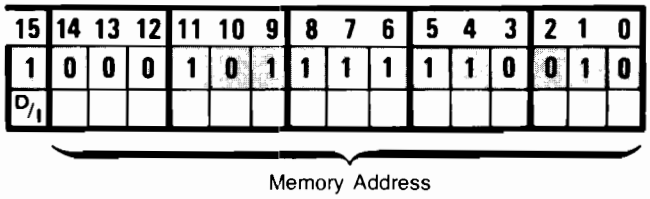
XBY EXCHANGE B AND Y



Exchanges the contents of the B- and Y-registers.

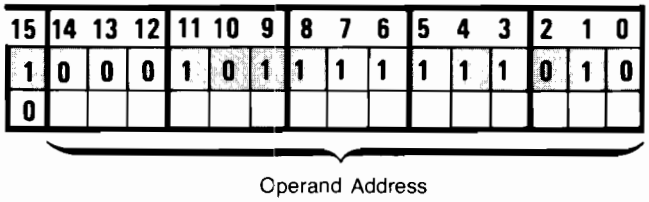
3-32. JUMP INSTRUCTIONS. The following four jump instructions allow a program to either jump to or exit from a subroutine.

JLY JUMP AND LOAD Y



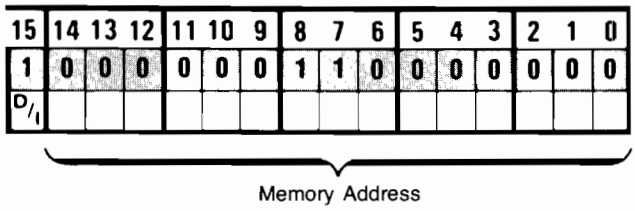
This instruction is designed for entering a subroutine. The instruction, executed in location P, causes computer control to jump unconditionally to the memory location specified in the memory address. Indirect addressing may be specified. The contents of the P-register plus two (return address) is loaded into the Y-register. A return to the main program sequence at P + 2 may be effected by a JPY instruction (described next).

JPY JUMP INDEXED BY Y



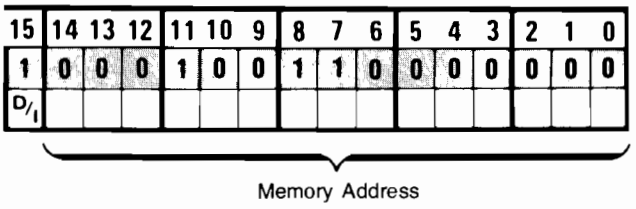
Transfers control to the effective address, which is computed by adding the contents of the Y-register to the operand address. Indirect addressing is not allowed. The effective address is loaded into the P-register; the Y-register contents are not altered.

JLA JUMP AND LOAD A



This instruction, executed in location P, causes computer control to jump unconditionally to the memory location specified by the second word of the instruction. The contents of the program counter plus two are stored in the A-register. A return to the main program will be effected by a JMP indirect through location 00000 (the A-register).

JLB JUMP AND LOAD B

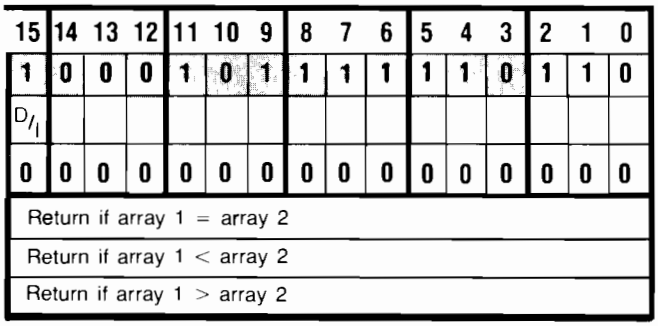


This instruction, executed in location P, causes computer control to jump unconditionally to the memory location specified by the second word of the instruction. The contents of the program counter plus two are stored in the B-register. A return to the main program will be effected by a JMP indirect through location 00001 (the B-register).

3-33. BYTE MANIPULATION INSTRUCTIONS. A byte address is defined as two times the word address plus zero or one, depending on whether the byte is in the high-order position (bits 8 through 15) or low-order position (bits 0 through 7) of the word containing it. If the byte of interest is in bit positions 8 through 15 of memory location 100, for example, then the address of that byte is 2* 100 + 0, or 200; the address of the low-order byte in the same location is 201 (2* 100 + 1). Because of the way byte addresses are defined, 16 bits are required to cover all possible byte addresses in the 32k-word logical address space. Hence, for byte addressing, bit 15 does not indicate indirect addressing. Memory references to byte addresses on the base page (4-3777) with CDS mode enabled will have 2*Q (byte base register) added to the base relative address.

Byte addresses 000 through 003 reference bytes in the A- and B-registers. These addresses will not cause memory violations. The user should, however, be careful in referencing these byte addresses; for example, storing into byte address 002 or 003 would destroy the byte address originally contained in the B-register.

CBT COMPARE BYTES



Compares the bytes in string 1 with those in string 2. This is a three-word instruction where

Word 1 = Instruction code,

Word 2 = Address of word containing the string count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-registers. The A-register contains the first byte address of string 1 and the B-register contains the first byte address of string 2.

The number of bytes to be compared is given in the memory location addressed by Word 2 of the instruction; the number of bytes to be compared is restricted to a positive integer greater than zero. The strings are compared one byte at a time; the ith byte in string 1 is compared with the ith byte in string 2. The comparison is performed arithmetically; i.e., each byte is treated as a positive number. If all bytes in string 1 are identical with all bytes in string 2, the "equal" exit is taken. As soon as two bytes are compared and found to be different, the "less than" or "greater than" exit is taken, depending on whether the byte in string 1 is less than or greater than the byte in string 2. The three ways this instruction exits are as follows:

a. No skip if string 1 is equal to string 2; the P-register advances one count from Word 3 of the instruction. The A-register contains its original value incremented by the count stored in the address specified in Word 2.

b. Skips one word if string 1 is less than string 2; the P-register advances two counts from Word 3 of the instruction. The A-register contains the address of the byte in string 1 where the comparison stopped.

c. Skips two words if string 1 is greater than string 2; the P-register advances three counts from Word 3 of the instruction. The A-register contains the address of the byte in string 1 where the comparison stopped.

For all three exits, the B-register will contain its original value incremented by the count stored in the address specified in Word 2. Wraparound of either byte address

produces undefined results. This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction produces undefined results when CDS is enabled.

LBT LOAD BYTE

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

This one word instruction loads into the A-register the byte whose address is contained in the B-register. The byte is right-justified with leading zeros in the left byte. The B-register is incremented by one.

MBT MOVE BYTES

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

Moves bytes in a left-to-right manner; i.e., the byte having the lowest address from the source is moved first. This is a three word instruction where

Word 1 = Instruction code,

Word 2 = Address of word containing the byte count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-registers. The A-register contains the first byte address source and the B-register contains the first byte address destination.

The number of bytes to be moved is given by a 16-bit positive integer greater than zero addressed by Word 2 of the instruction. The byte address in the A- and B-registers are incremented as each byte is being moved. Thus, at the end of the operation, the A- and B-registers are incremented by the number of bytes moved. Wraparound of either byte address produces undefined results. For each byte move, a memory protect check is performed.

If different logical pages are mapped to the same physical page and an attempt is made to move bytes between overlapping strings in physical memory via different logical pages, erroneous results may occur.

This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction produces undefined results when CDS is enabled.

SBT STORE BYTE

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

Stores the A-register low-order (right) byte in the byte address contained in the B-register. The B-register is incremented by one. A memory protect check is performed before the byte is stored. The left byte in the A-register does not have to be zeros. The other byte in the same word of the stored byte is not altered.

SFB SCAN FOR BYTE

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

This is a one word instruction with the operands in the A- and B-registers. The A-register contains a termination byte (high-order byte) and a test byte (low-order byte). The B-register contains the first byte address of the string to be scanned.

A string of bytes is scanned starting at the byte address given in the B-register. Scanning terminates when a byte in the string matches either the test byte or the termination byte in the A-register. The manner in which the instruction exits depends on which byte is matched first. If a byte in the string matches the test byte, the instruction will not skip upon exit; the B-register will contain the address of the byte matching the test byte. If a byte in the string matches the termination byte, the instruction will skip one word upon exit; the B-register will contain the address of the byte matching the termination byte *plus one*.

The scanning operation will not continue indefinitely even if neither the termination byte nor test byte exists in memory. These bytes are in the A-register with byte addresses 000 and 001, respectively. Thus, if no match is made by the time the B-register points to the last byte in memory, the B-register will roll over to zero and the next test will match the termination byte in the A-register with itself.

This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-registers.

3-34. BIT MANIPULATION INSTRUCTIONS. The following three instructions allow any number of bits in a specified memory location to be cleared, set, or tested.

CBS CLEAR BITS

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

Memory Address

Clears bits in the addressed location. This is a three-word instruction where

Word 1 = Instruction code,

Word 2 = Address of a 16-bit mask, and

Word 3 = Address of word where bits are to be cleared.

The bits to be cleared correspond to logic 1's in the mask. The bits corresponding to logic 0's in the mask are not affected. A memory protect check is performed prior to modifying the word in memory.

SBS SET BITS

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

Memory Address

Sets bits in the addressed location. This is a three-word instruction where

Word 1 = Instruction code,

Word 2 = Address of a 16-bit mask, and

Word 3 = Address of word where bits are to be set.

The bits to be set correspond to logic 1's in the mask. The bits corresponding to logic 0's in the mask are not affected. A memory protect check is performed prior to modifying the word in memory.

TBS TEST BITS

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

Memory Address

Tests (compares) bits in the addressed location. This is a three-word instruction where

Word 1 = Instruction code,

Word 2 = Address of a 16-bit mask, and

Word 3 = Address of word in which bits are to be tested.

The bits in the addressed memory word corresponding to logic 1's in the mask are tested. If all the bits tested are 1's, the instruction will not skip; otherwise the instruction will skip one word (i.e., the P-register will advance two counts from Word 3 of the instruction).

3-35. WORD MANIPULATION INSTRUCTIONS.

The following instructions facilitate the comparing and moving of word arrays.

CMW COMPARE WORDS

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

Return if array 1 = array 2

Return if array 1 < array 2

Return if array 1 > array 2

Compares the words in array 1 with those in array 2. This is a three-word instruction where

Word 1 = Instruction code,

Word 2 = Address of word containing the word count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-registers. The A-register contains the first word address of array 1 and the B-register contains the first word address of array 2. Bit 15 of the addresses in the A- and B-registers are ignored; i.e., no indirect addressing allowed.

The number of words to be compared is given in the memory location addressed by Word 2 of the instruction; a negative word count produces undefined results. The arrays are compared one word at a time; the ith word in array 1 is compared with the ith word in array 2. This comparison is performed arithmetically; i.e., each word is considered a two's complement number. If all words in array 1 are equal to all words in array 2, the "equal" exit is taken. As soon as two words are compared and found to be different, the "less than" or "greater than" exit is taken,

depending on whether the word in array 1 is less than or greater than the word in array 2. The three ways this instruction exits are as follows:

a. No skip if array 1 is equal to array 2; the P-register advances one count from Word 3 of the instruction. The A-register contains its original value incremented by the word count stored in the address specified in Word 2.

b. Skips one word if array 1 is less than array 2; the P-register advances two counts from Word 3 of the instruction. The A-register contains the address of the word in array 1 where the comparison stopped.

c. Skips two words if array 1 is greater than array 2; the P-register advances three counts from Word 3 of the instruction. The A-register contains the address of the word in array 1 where the comparison stopped.

For all three exits, the B-register will contain its original value incremented by the word count stored in the address specified in Word 2. This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. Wraparound of either word address produces undefined results with the following exception: if the last word accessed was at location 077777 octal, the A- or B-register will properly terminate with a value of 100000 octal. This instruction produces undefined results when CDS is enabled.

MVW MOVE WORDS

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

Moves words in a left-to-right manner; i.e, the word having the lowest address in the source is moved first. This is a three-word instruction where

Word 1 = Instruction code,

Word 2 = Address of word containing the count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-registers. The A-register contains the address of the first source word and the B-register contains the address of the first destination word. Bit 15 of the addresses in the A- and B-registers is ignored; i.e., no indirect addressing is allowed. The number of words to be moved is given in the memory location addressed by Word 2 of the instruction. A negative word count causes undefined results. For each word move, a memory protect check is performed.

At the end of the operation the A- and B-registers are incremented by the number of words moved. Wraparound of either word address causes undefined results with the following exception: if the last word accessed was at location 077777 octal, the A- or B-register will properly terminate with a value of 100000 octal.

This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction produces undefined results when CDS is enabled.

3-36. FLOATING POINT INSTRUCTIONS

The floating point instructions allow addition, subtraction, multiplication, and division of both single precision (32-bit) and double precision (64-bit) floating point quantities, and conversion of quantities from floating point format to integer format or vice versa. Data formats are shown in Figure 3-1. Except for zero, all floating point operands must be normalized (i.e., sign of mantissa differs from most significant bit of mantissa).

For multiple-word instructions, indirect addressing to any number of levels is permitted for the words indicated as memory address. A logic 0 in bit position 15 specifies direct addressing; a logic 1 specifies indirect addressing.

The execution times of the floating point instructions are specified in Table 3-5. These instructions are noninterruptible; any attempted interrupt is held off for the full execution time of the currently active floating point instruction. However, data transfer via direct memory access is not held off.

3-37. SINGLE PRECISION OPERATIONS. Overflow for single precision operations occurs if the result lies outside the range of representable single precision floating point numbers [-2^{127}, $(1-2^{-23})\ 2^{127}$]. In such a case, the overflow flag is set and the result $(1-2^{-23})\ 2^{127}$ is returned to the A- and B-registers. Underflow occurs if the result lies inside the range [-2^{-129} $(1+2^{-22})$, 2^{-129}]. In such a case, the overflow flag is set and the result 0 is returned to the A- and B-registers.

FAD FLOATING POINT ADD

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

Memory Address

Adds the floating point quantity in the A- and B-registers to the floating point quantity in the specified memory locations. The floating point result is returned to the A- and B-registers.

FSB FLOATING POINT SUBTRACT

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

Memory Address

Subtracts the floating point quantity in the specified memory locations from the floating point quantity in the A- and B-registers. The floating point result is returned to the A- and B-registers.

FMP FLOATING POINT MULTIPLY

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

Memory Address

Multiplies the floating point quantity in the A- and B-registers by the floating point quantity in the specified memory locations. The floating point result is returned to the A- and B-registers.

FDV FLOATING POINT DIVIDE

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

Memory Address

Divides the floating point quantity in the A- and B-registers by the floating point quantity in the specified memory locations. The floating point result is returned to the A- and B-registers.

FIX FLOATING POINT TO SINGLE INTEGER

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

Converts the floating point quantity in the A- and B-registers to single integer format. The integer result is returned to the A-register. If the magnitude of the floating point number is <1, regardless of sign, the integer 0 is returned. If the magnitude of the exponent of the floating point number is ≥ 16, regardless of sign, the integer 32767 (077777 octal) is returned as the result and the overflow flag is set.

FLT SINGLE INTEGER TO FLOATING POINT

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

Converts the single integer quantity in the A-register to single precision floating point format. The floating point result is returned to the A- and B-registers.

.FIXD* FLOATING POINT TO DOUBLE INTEGER

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

Converts the floating point quantity in the A- and B-registers to double integer format. The integer result is returned to the A- and B-registers. (The A-register contains the most-significant word and the B-register contains the least-significant word.) If the magnitude of the floating point number is <1, regardless of sign, the integer 0 is returned. If the magnitude of the exponent of the floating point number is ≥ 32, regardless of sign, the integer $2^{31}-1$ is returned as the result and the overflow flag is set.

.FLTD* DOUBLE INTEGER TO FLOATING POINT

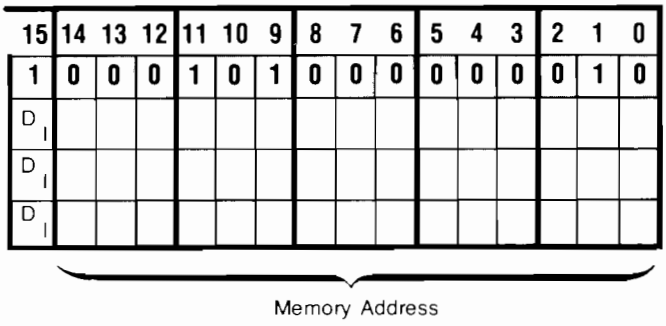
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0

Converts the double integer quantity in the A- and B-registers to single precision floating point format. The floating point result is returned to the A- and B-registers. Positive numbers truncate toward zero and negative numbers truncate away from zero when precision is lost in the conversion.

3-38. DOUBLE PRECISION OPERATIONS. Overflow for double precision operations occurs if the result lies outside the range of representable double precision floating point numbers $[-2^{127}, (1-2^{-55})\,2^{-127}]$. In such a case, the overflow flag is set and $(1-2^{-55})\,2^{127}$ is returned as the result. Underflow occurs if the result lies inside the range $[-2^{-129}(1+2^{-54}), 2^{-129}]$. In such a case, the overflow flag is set and 0 is returned as the result.

*For HP Assembly Language usage, refer to paragraph 3-48.

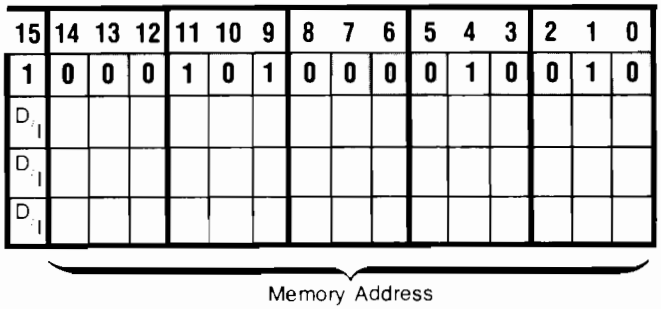
.TADD* DOUBLE FLOATING POINT ADD



Adds two double precision floating point quantities (augend plus addend). This is a four-word instruction where

Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of augend.
Word 4 = Address of addend.

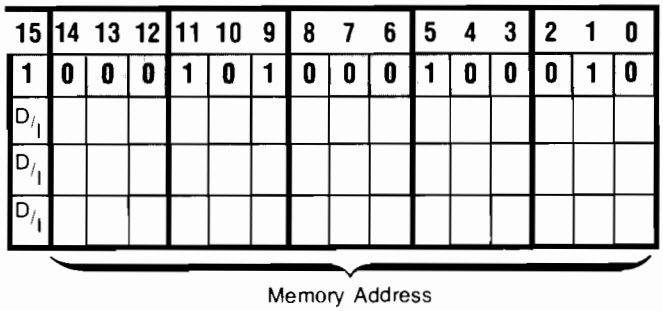
.TSUB* DOUBLE FLOATING POINT SUBTRACT



Subtracts one double precision floating point quantity from another (minued minus subtrahend). This is a four-word instruction where

Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of minuend.
Word 4 = Address of subtrahend.

.TMPY* DOUBLE FLOATING POINT MULTIPLY

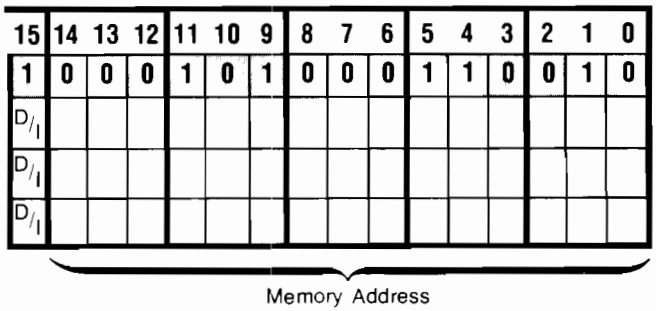


*For HP Assembly Language usage, refer to paragraph 3-48.

Multiplies one double precision floating point quantity by another (multiplicand by multiplier). This is a four-word instruction where

Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of multiplicand.
Word 4 = Address of multiplier.

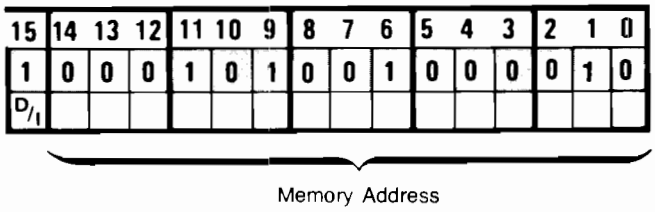
.TDIV* DOUBLE FLOATING POINT DIVIDE



Divides one double precision floating point quantity by another (dividend by divisor). This is a four-word instruction where

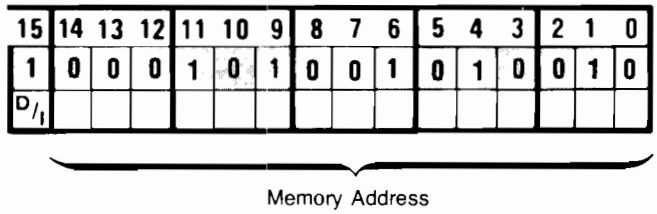
Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of dividend.
Word 4 = Address of divisor.

.TFXS* DOUBLE FLOATING POINT TO SINGLE INTEGER



Converts the double precision floating point quantity in the specified memory locations to single integer format. The integer result is returned to the A-register. If the magnitude of the floating point number is <1, regardless of sign, 0 is returned as the result. If the magnitude of the exponent of the floating point number is ≥ 16, regardless of sign, the integer $2^{15}-1$ is returned as the result and the overflow flag is set.

.TFTS* SINGLE INTEGER TO DOUBLE FLOATING POINT



Converts the single integer quantity in the A-register to double precision floating point format. The floating point result is returned to the specified memory locations.

.TFXD* DOUBLE FLOATING POINT TO DOUBLE INTEGER

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

Memory Address

Converts the double precision floating point quantity in the specified memory locations to double integer format. The integer result is returned to the A- and B-registers. (The A-register contains the most-significant word and the B-register contains the least-significant word.) If the magnitude of the floating point number is <1, regardless of sign, 0 is returned as the result. If the magnitude of the exponent of the floating point number is ≥32, regardless of sign, the integer $2^{31}-1$ is returned as the result and the overflow flag is set.

.TFTD* DOUBLE INTEGER TO DOUBLE FLOATING POINT

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

Memory Address

Converts the double integer quantity in the A- and B-registers to double precision floating point format. The floating point result is returned to the specified memory locations.

3-39. LANGUAGE INSTRUCTION SET

The Language Instruction Set (LIS) instructions perform several frequently-used FORTRAN operations including parameter passing, array address calculations, and floating point conversion, packing, and rounding operations.

For multiple-word instructions, indirect addressing to any number of levels is permitted for the words indicated as a memory address. A logic 0 in bit position 15 specifies direct addressing; a logic 1 specifies indirect addressing.

*For HP Assembly Language usage, refer to paragraph 3-48.

The following paragraphs provide machine language coding and definitions for the Language Instruction Set. Data formats are shown in Figure 3-1.

NOTE

For a more detailed description of the instructions in the Language Instruction Set, refer to the Relocatable Library Reference Manual, HP part no. 92077-90037.

.BLE* SINGLE FLOATING POINT TO DOUBLE FLOATING POINT

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

Memory Address

Converts the single precision floating point quantity in specified memory locations to a double precision floating point quantity. The result is returned to other specified memory locations. This is a four-word instruction where

Word 1 = Instruction code.
Word 2 = Return address
Word 3 = Address of result.
Word 4 = Address of operand.

.NGL* DOUBLE FLOATING POINT TO SINGLE FLOATING POINT

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

Memory Address

Converts the double precision floating point quantity in the specified memory locations to a single precision floating point quantity. The result is placed in the A- and B-registers. Overflow is cleared unless, during execution, rounding results in overflow or underflow of the exponent, in which case overflow is set and the result is truncated to the greatest positive number. This is a three word instruction where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of operand.

.XFER* TRANSFER THREE CONSECUTIVE WORDS

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

Transfers three consecutive words from one memory location to another. The A-register must contain the source address and the B-register must contain the destination address. The source address +3 is returned to the A-register; the destination address +3 is returned to the B-register. Wraparound of either address produces undefined results. Under CDS, the source and/or destination addresses may be adjusted for base relativity.

.DFER* TRANSFER THREE CONSECUTIVE WORDS

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

Memory Address

Transfers three consecutive words from one memory location to another. The source address +3 is returned to the A-register; the destination address +3 is returned to the B-register. This is a three word instruction where

Word 1 = Instruction code.
Word 2 = Destination address.
Word 3 = Source address.

Wraparound of either address produces undefined results. Under CDS, the source and/or destination addresses may be adjusted for base relativity.

.CFER* TRANSFER COMPLEX OR DOUBLE FLOATING POINT

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

Memory Address

Transfers a double floating point quantity (four consecutive words) from one memory location to another. The source address +4 is returned to the A-register; the destination address +4 is returned to the B-register. This is a three word instruction where

Word 1 = Instruction code.
Word 2 = Destination address.
Word 3 = Source address.

Wraparound of either address produces undefined results. Under CDS, the source and/or destination addresses may be adjusted for base relativity.

.ZFER* TRANSFER EIGHT WORDS

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

Memory Address

Transfers eight consecutive words from one memory location to another. The source address +8 is returned to the A-register; the destination address +8 is returned to the B-register. This is a three word instruction where

Word 1 = Instruction code.
Word 2 = Destination address.
Word 3 = Source address.

Wraparound of either address produces undefined results. Under CDS, the source and/or destination addresses may be adjusted for base relativity.

.ENTN* TRANSFER PARAMETER ADDRESSES

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

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. The return address stored in the SUB entry point references the word following the last parameter DEF in the calling routine. A true address is determined by eliminating all indirect references. This instruction may not be used when CDS is enabled.

.ENTC* TRANSFER PARAMETER ADDRESSES

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

*For HP Assembly Language usage, refer to paragraph 3-48.

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. The return address stored in the SUB entry point references the word following the last parameter DEF in the calling routine. There must be exactly two words between the subroutine entry point and the .ENTC instruction. A true address is determined by eliminating all indirect references. The true return address is returned to the A-register. Used for privileged or re-entrant subroutines. This instruction may not be used when CDS is enabled.

.ENTR* TRANSFER PARAMETER ADDRESSES

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

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. A true address is determined by eliminating all indirect references. (Refer to the Relocatable Library Reference Manual, part no. 92077-90037, for more information.) This instruction may not be used when CDS is enabled.

.ENTP* TRANSFER PARAMETER ADDRESSES

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

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. A true address is determined by eliminating all indirect references. The true return address is returned to the A-register. Used for privileged or re-entrant subroutines. (Refer to the Relocatable Library Reference Manual, part no. 92077-90037, for more information.) This instruction may not be used when CDS is enabled.

.CPM* SINGLE INTEGER ARITHMETIC COMPARE

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

Return if operand 1 = operand 2

Return if operand 1 < operand 2

Return if operand 1 > operand 2

Arithmetically compares operands addressed by second and third word. Does not skip if operands are equal; however, skips one instruction if the first operand is less than the second, or skips two instructions if the first operand is greater than the second.

.SETP* SET A TABLE

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

Sets a table of increasing numbers in consecutive memory locations. The A-register must contain the initial number and the B-register must contain the initial memory address (direct only); the contents of the succeeding memory location must define the number of memory locations (count ≥ 0). Entries in the table are established by incrementing the initial address and number by one (1) for each successive entry until the last number, initial number +COUNT−1, is reached and the A-register equals the initial value+COUNT. Wraparound will produce undefined results. This instruction is interruptible. On return, the B-register equals the initial address +COUNT. Under CDS, the memory addresses may be adjusted for base relativity.

NOTE

If the initial address +COUNT −1 results in an address which is beyond the end of logical memory, addresses within the base page may be destroyed.

..FCM* NEGATE SINGLE FLOATING POINT

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

Negates a packed single precision floating point quantity located in the A- and B-registers. The result is returned to the A- and B-registers.

..TCM* NEGATE DOUBLE FLOATING POINT

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

Memory Address

Negates a packed double precision floating point quantity located in the specified memory locations. The result is returned to the same specified memory locations.

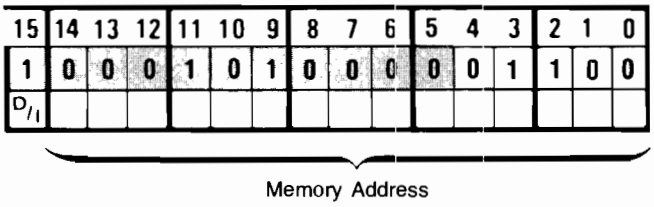
*For HP Assembly Language usage, refer to paragraph 3-48.

3-40. DOUBLE INTEGER INSTRUCTIONS

The double integer instructions allow arithmetic and test operations on 32-bit integer quantities. The data format for double integer values is shown in Figure 3-1. Double integer values contained in the (A,B) registers have the most significant bits in the A-register. Values stored in memory require two locations. The operand address in a double integer instruction points to the first memory location, which contains the most significant bits.

Instructions which do not return information in the extend or overflow bits will not alter the state of these flags. Operations which may return an overflow condition will clear overflow at entry.

.DAD* DOUBLE INTEGER ADD

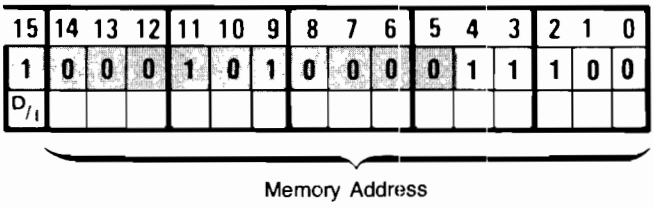


Perform the double integer operation:

(A,B) = (A,B) + <OPND>

The contents of <OPND> are unaltered. In the event of overflow, the overflow bit is set and the returned result contains the lower 32-bits of the actual sum, in unsigned form. The extend bit will be set if an unsigned carry out of the A-register occurs.

.DSB* DOUBLE INTEGER SUBTRACT

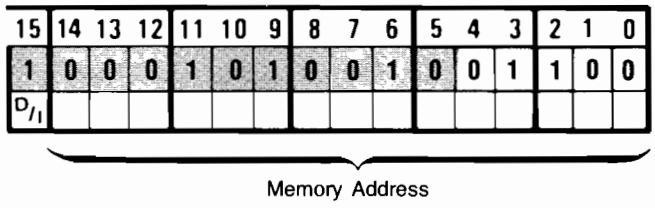


Performs the double integer operation:

(A,B) = (A,B) − <OPND>

The contents of <OPND> are unaltered. In the event of overflow, the overflow bit is set and the returned result contains the lower 32-bits of the actual difference, in unsigned form. The extend bit will be set if an unsigned borrow out of the A-register occurs.

.DSBR* DOUBLE INTEGER SUBTRACT REVERSE

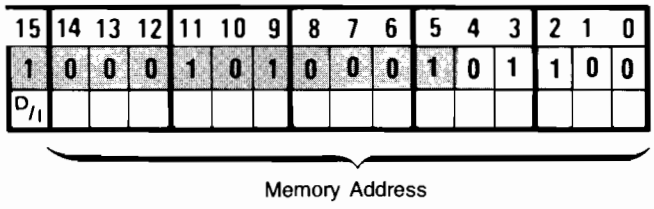


Performs the double integer operation:

(A,B) = <OPND> − (A,B)

The contents of <OPND> are unaltered. In the event of overflow, the overflow bit is set and the returned result contains the lower 32-bits of the actual difference, in unsigned form. The extend bit will be set if an unsigned borrow out of operand occurs.

.DMP* DOUBLE INTEGER MULTIPLY

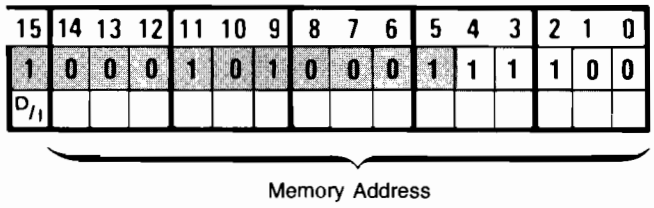


Performs the double integer operation:

(A,B) = (A,B) × <OPND>

The contents of <OPND> are unaltered. If overflow occurs, the result (077777, 17777) is returned and overflow is set.

.DDI* DOUBLE INTEGER DIVIDE



Performs the double integer operation:

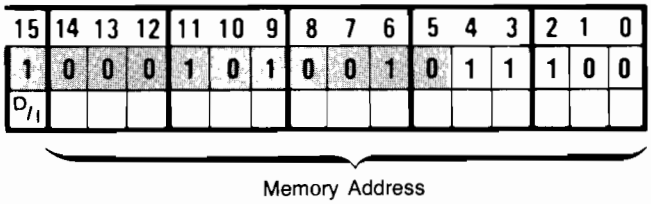
(A,B) = (A,B) ÷ <OPND>

The contents of <OPND> are unaltered. If overflow or divide by zero occurs, the result (077777, 177777) is returned and overflow is set.

---

*For HP Assembly Language usage, refer to paragraph 3-48.

.DDIR* DOUBLE INTEGER DIVIDE REVERSE

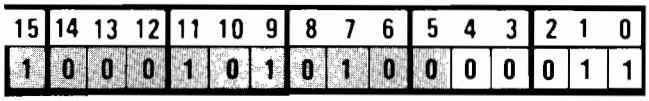


Performs the double integer operation:

(A,B) = <OPND> ÷ (A,B)

The contents of <OPND> are unaltered. If overflow or divide by zero occurs, the result (077777, 177777) is returned and overflow is set.

.DNG* DOUBLE INTEGER NEGATE

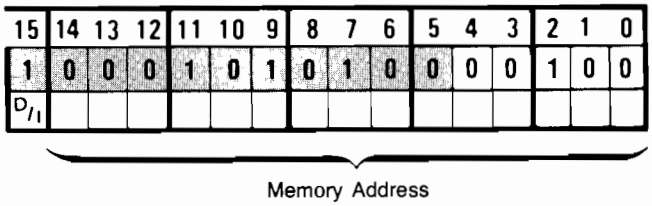


Performs the double integer operation:

(A,B) = − (A,B)

An input value of (10000,000000) is left unchanged and overflow is set. An input value of zero will cause the extend bit to be set.

.DCO* DOUBLE INTEGER COMPARE



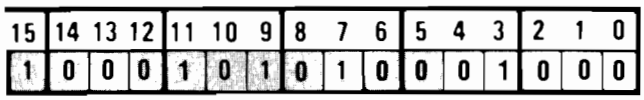
Compares the double integers (A,B) and <OPND>

If (A,B) = <OPND> Return to P+2
If (A,B) < <OPND> Return to P+3
If (A,B) > <OPND> Return to P+4

where P is the address of the .DCO instruction. The value of both double integers and the overflow bit are unaltered.

*For HP Assembly Language usage, refer to paragraph 3-48.

.DIN* DOUBLE INTEGER INCREMENT

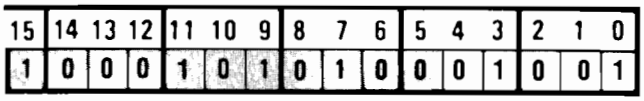


Performs the double integer operation:

(A,B) = (A,B) + 1

An input value of (077777, 177777) will return a result of (100000, 000000) and set overflow. An input value of (177777, 177777) will return a result of zero and cause the extend bit to be set.

.DDE* DOUBLE INTEGER DECREMENT

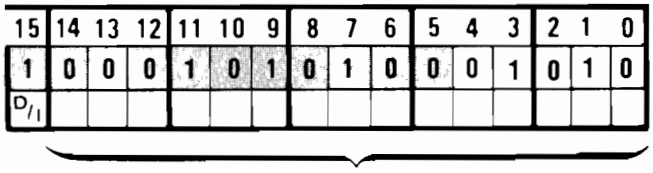


Performs the double integer operation:

(A,B) = (A,B) − 1

An input value of (100000, 000000) will return the result (077777, 177777) and set overflow. An input value of zero will return the result (177777, 177777) and cause the extend bit to be set.

.DIS* DOUBLE INTEGER INCREMENT AND SKIP IF ZERO

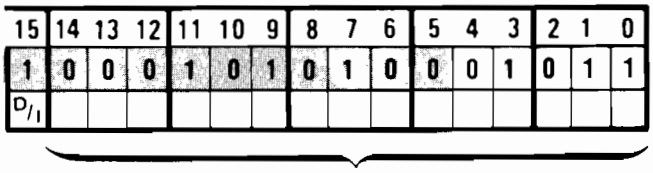


Performs the double integer operation:

<OPND> = <OPND> + 1

If the new value of <OPND> equals zero, the next instruction will be skipped. The value in <OPND> is treated as an unsigned number, and carry out of the <OPND> is ignored.

.DDS* DOUBLE INTEGER DECREMENT AND SKIP IF ZERO



Performs the double integer operation;

<OPND> = <OPND> – 1

If the new value of <OPND> equals zero, the next instruction will be skipped. The value in <OPND> is treated as an unsigned number, and a borrow out of the <OPND> is ignored.

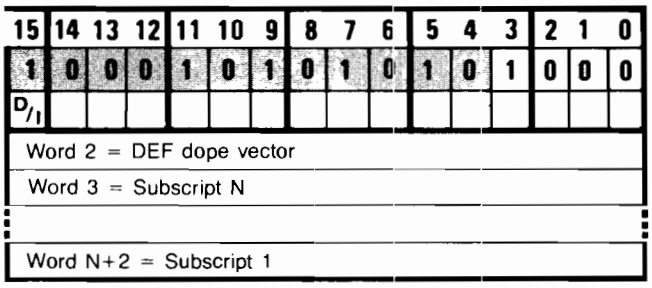
3-41. VIRTUAL MEMORY INSTRUCTIONS

The Virtual Memory Instructions perform accesses to Virtual Memory and Extended Memory Area, which are extensions of logical memory. If an addressed data item is in physical memory, the instructions perform the required mapping, including modification of map registers and entry of the appropriate page numbers into the user's logical address space. If an addressed data item is not in physical memory, a fault is generated to a macrocode routine which swaps the data from the disc into physical memory and then restarts the VMA instruction. The fault sequence generated depends on whether the CDS mode is enabled. If CDS mode is disabled, a JSB,I through memory location 04 in the user map is effected. Memory location 04 is expected to contain the address of the entry point of the VMA fault-handler in the user space (indirect addressing is not allowed). If CDS mode is enabled, an interrupt is generated to trap cell 12 octal in the system map. As the VMA fault interrupt is the lowest priority interrupt, any other pending interrupts will be serviced first.

With one exception, VMA always maps both the page that the requested VMA address is on and the next page, ensuring that entire data items up to 1k words in size are mapped in. The exception is .PMAP, which only maps in the requested page.

For more information on VMA and EMA, refer to the *RTE-A Programmer's Reference Manual,* HP part no. 92077-90007.

.IMAP* 16-BIT SUBSCRIPT MAPPING



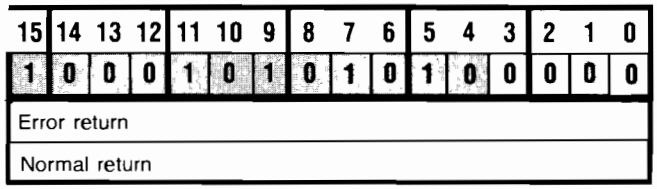
*For HP Assembly Language usage, refer to paragraph 3-48.

Performs a subscript calculation and maps the result into logical memory. Each of the subscripts and dimensions are 16-bit integers. However, the calculation uses 32-bit adds and multiplies. The subscript words cannot address the A- or B-register.

Word 2 points to a table that specifies the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

On a normal return, the A-register is undefined and the B-register contains the logical address.

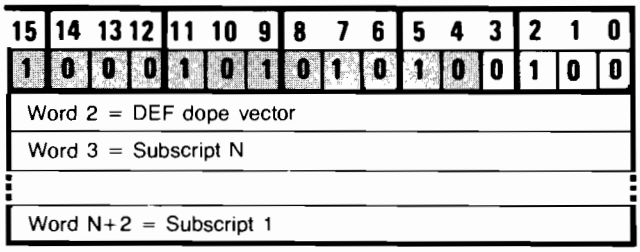
.PMAP* MAP SPECIFIED PAGE



On entry, the A-register is loaded with the number of the user-map register to be altered and the B-register is loaded with the page ID, which are the parameters passed to the routine. If an attempt is made to map in the last+1 page, that PMR is mapped read and write protected and the E-register is set. When no error occurs, a normal return occurs to the second word after the instruction; mapping is complete, and the contents of the A- and B-registers are incremented. If a fault occurs and the sign bit is set in the A-register, an error return to the word following the instruction occurs. If a fault occurs and the sign bit is not set in the A-register, a normal fault sequence is generated. The O-register is undefined. The E-register is set if an attempt was made to map the last+1 page; otherwise it is cleared.

The .PMAP instruction uses the last user page (31) of memory and then maps that logical page read and write protected. After a .PMAP call, memory references to address greater than 75777 octal will cause memory protect violations.

.IRES* 16-BIT SUBSCRIPT RESOLUTION

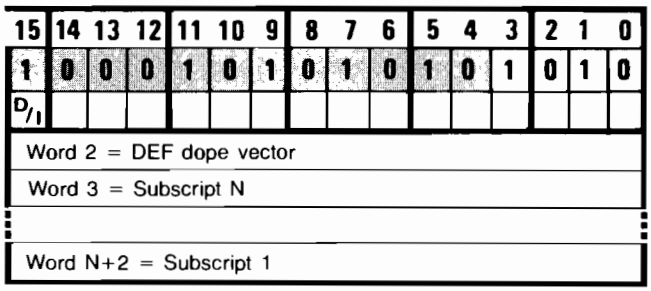


Performs a subscript calculation. Each of the subscripts and dimensions are 16-bit integers. However, the calculation uses 32-bit adds and multiplies. The subscript words cannot address the A- or B-register.

Word 2 points to a table that specifies the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

On a normal return, the A- and B-registers contain the address of the array element in double-integer format (most significant word in the A-register).

.JMAP* 32-BIT SUBSCRIPT MAPPING

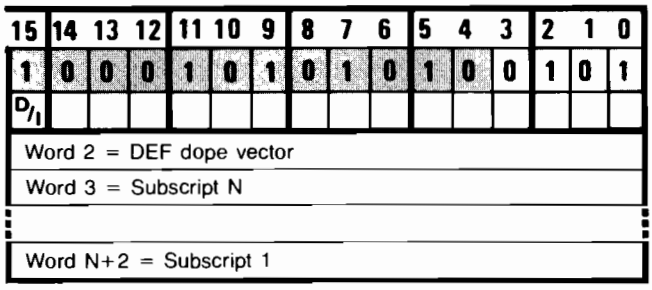


Performs a subscript calculation and maps the result into logical memory. Each of the subscripts and dimensions are 32-bit integers, and the calculation uses 32-bit adds and multiplies. The subscript words cannot address the A- or B-register.

Word 2 points to a table that specifies the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

On a normal return, the A-register is undefined and the B-register contains the logical address.

.JRES* 32-BIT SUBSCRIPT RESOLUTION

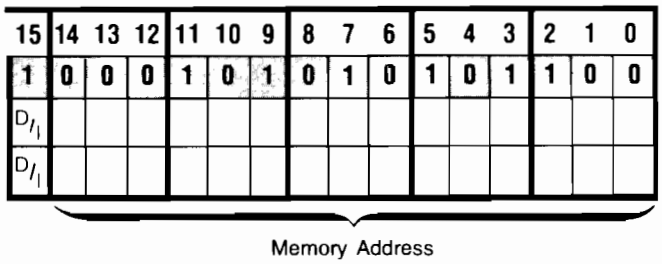


Performs a subscript calculation. Each of the subscripts and dimensions are 32-bit integers, and the calculation uses 32-bit adds and multiplies. The subscript words cannot address the A- or B-register.

Word 2 points to a table that specifies the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

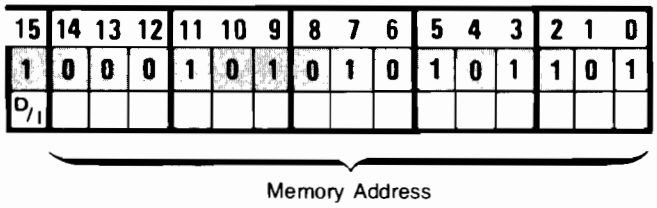
On a normal return, the A- and B-registers contain the address of the array element in double-integer format (most significant word in the A-register).

.LPXR* INDEXED MAPPING WITH DEF



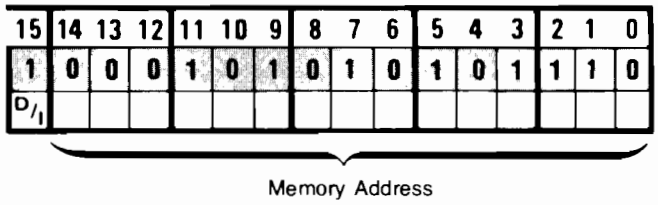
On entry, the pointer specified by the second instruction word is resolved, and the double word it points to is loaded into the A- and B-registers. The offset specified in the third instruction word is resolved, and the double word it points to is added to the contents of the A- and B-registers. The result is treated as a 26-bit VMA pointer and is mapped. On exit, the B-register contains the logical address of the data item, and the A-register is undefined. The offset word cannot refer to the A- or B-register.

.LPX* INDEXED MAPPING WITH REGISTERS



On entry, the second instruction word either directly or indirectly points to a double integer in memory, which is to be added to the double integer in the A- and B-registers to form a double-word VMA pointer. The result is treated as a 26-bit VMA pointer and is mapped. On exit, the B-register contains the logical address of the data item, and the A-register is undefined.

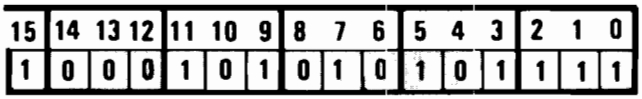
.LBPR* MAPPING WITH DEF



On entry, the pointer specified by the second instruction word is resolved and the double word it points to is loaded into the A- and B-registers. This value is treated as a 26-bit VMA pointer and is mapped. On exit, the B-register contains the logical address of the data item, and the A-register is undefined.

*For HP Assembly Language usage, refer to paragraph 3-48.

.LBP* MAPPING WITH REGISTERS

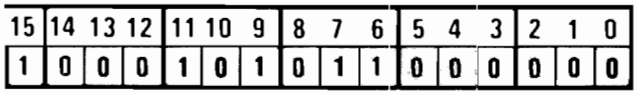


On entry, the 26-bit VMA pointer is contained in the A-register (most significant word) and B-register. The data item is mapped. On exit, the B-register contains the logical address of the data item, and the A-register is undefined.

3-42. OPERATING SYSTEM INSTRUCTION SET

The operating system instructions provide instructions for ascertaining the CPU and firmware identification, and instructions for interrupt conditions.

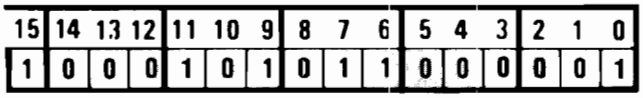
.CPUID* PROCESSOR IDENTIFICATION



The A-register is loaded with a number that identifies the processor installed in the computer system, where:

Octal 2 = A600 Computer.
Octal 3 = A700 Computer.
Octal 4 = A900 Computer.
Octal 5 = A600+ Computer.

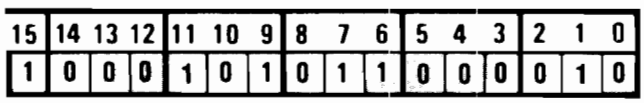
.FWID* FIRMWARE IDENTIFICATION



On entry, the B-register holds a number indicating which bank of 1k microwords is identified. On exit, the A-register contains a number that identifies the specific ROM package (lower byte) and revision date code (upper byte). If no microcode exists in the selected block, the A-register is set to 177777 octal.

ROM package = 0, 1, 2 = A900 base set without CDS support
3 = A900 base set with CDS support

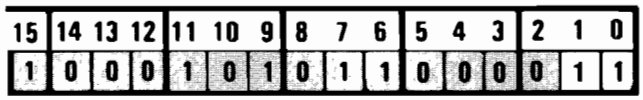
.WFI* WAIT FOR INTERRUPT



This instruction is equivalent to a JMP * except that the processor does not perform memory accesses, which would decrease the effective bandwidth of the memory backplane. This instruction is interruptible.

*For HP Assembly Language usage, refer to paragraph 3-48.

.SIP* SKIP IF INTERRUPT PENDING



The processor skips if an I/O interrupt is pending (INTRQ- is asserted on the A-Series backplane), which is independent of the Type 2 and Type 3 interrupt masks. (Refer to Table 6-1.)

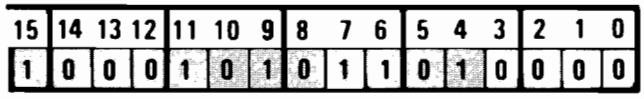
3-43. EXECUTION TIMES

Table 3-5 lists the execution times required for the various base set instructions.

3-44. SCIENTIFIC INSTRUCTION SET

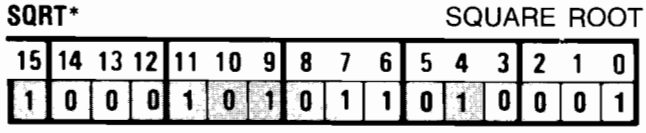
The Scientific Instruction Set (SIS) is included with the optional Floating Point Processor (FPP) card and performs nine trigonometric and logarithmic functions. The following paragraphs provide machine language coding and definitions for the SIS instructions. Error conditions and codes are given in Table 3-6. Note that except for zero, all floating point operands must be normalized (i.e., sign of mantissa differs from most significant bit of mantissa).

TAN* TANGENT



Calculates the tangent of the single precision floating point quantity (in radians) contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error return will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

SQRT* SQUARE ROOT



Calculates the square root of the single precision floating point quantity contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error return will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

Table 3-5. Typical Base Set Instruction Execution Times

INSTRUCTION	EXECUTION TIME (μsec)
Memory Reference Group	
LDA/B, ADA/B, IOR, XOR, AND	0.267
STA/B	0.40
CPA/B	0.533
ISZ	0.533
JSB	0.533
JMP	0.133
(Each Indirect Address Level)	0.133
Alter/Skip Group	
CLA/B or CMA/B	0.267
CCA/B or INA/B	0.267
SZA/B or SSA/B	0.533
All other combinations with skip	0.533
All other combinations w/out skip	0.267
Shift/Rotate Group	
With skip	0.533
Without skip	0.40
Extended Arithmetic Group	
DLD	0.533
DST	0.53
MPY	2.267
DIV	6.267
ASL	0.80
ASR, LSR, RRR	0.667
LSL, RRL	0.40
Input/Output Group	
HLT	3.067
By select code:	
SC0: CLF, STF	0.667
SFC, SFS	0.933
LIA/B	3.067
OTA/B	3.867
CLC	2.00
STC	0.40
SC1: CLF, STF, STC, CLC	0.267
SFC, SFS	0.40
LIA/B	1.867
OTA/B	0.40
SC2: STF, CLF, STC, CLC	0.80
SFC, SFS	0.80
LIA/B	3.067
OTA/B	3.60
SC3: STC, CLC	0.40
CLF	0.40
SFC, SFS	0.40
LIA/B	3.067
OTA/B	3.333
STF	1.067

INSTRUCTION	EXECUTION TIME (μsec)
SC4: CLF	0.40
SFC, SFS	0.80
LIA/B, OTA/B	0.533
STC, CLC	0.667
SFT (flushes cache)	2.00 msec.
SC5: STF, CLF	0.40
SFC, SFS	0.40
LIA/B	0.80
OTA/B	2.133
STC, CLC	0.80
SC6: STF, CLF, STC, CLC	0.80
SFC, SFS	0.80
LIA/B, OTA/B	0.40
SC7: CLF, CLC	0.40
STC	0.933
SFC, SFS	0.40
LIA/B	0.533
OTA/B	0.40
STF	0.80
SC20 and up:	
STC, CLC	2.667
CLF, STF	1.20
SFC, SFS without skip	1.333
Additional with skip	1.067
LIA/B, MIA/B	3.067
OTA/B	3.333
STC, CLC = 20	2.667
STC, CLC > 20	1.333
Extended Instruction Group	
(Index Register Instructions)	
ADX, ADY, LDX, LDY	0.40
CAX, CBX, CAY, CBY, CXA, CXB	0.40
CYA, CYB	0.75
DSX, DSY, ISX, ISY	0.667
LAX, LBX, LAY, LBY, STX, STY	0.533
SAX, SAY, SBX, SBY	0.533
XAX, XBX, XAY, XBY	0.533
JLY	0.533
Per each indirect address level	0.133
JPY	0.40
JLA, JLB	0.533
(Bit Manipulation Instructions)	
CBS, SBS, TBS	0.80
(Word Manipulation Instructions)	
MVW	0.933 plus 0.267/word
CMW	1.733 plus 1.20 for four words
(Byte Manipulation Instructions)	
CBT	1.467
Additional for two bytes	0.40
LBT	0.667
MBT	1.333
Additional per byte	0.133
SBT	0.933
SFB	1.733
Additional for two bytes	0.40

Table 3-5. Typical Base Set Instruction Execution Times (Continued)

INSTRUCTION	EXECUTION TIME (μsec)
Language Support Instructions	
.ENTR, .ENTP	1.60
.ENTN, .ENTC	1.20
Additional per word (no indirects)	0.267
Additional for each indirect level	0.133
.CPM	1.20
.SETP	1.067
Additional per word	0.267
..FCM	1.60
..TCM	2.267
.NGL	2.40
.BLE	1.60
.DFER	1.60
.CFER	1.867
.ZFER	2.933
.XFER	1.333
Double Integer Instructions	
.DAD, .DSB, .DSBR	0.80
.DIN, .DDE	0.667
.DIS, .DNG, .DDS	0.933
.DCO	1.333
.DMP	2.40
.DDI	5.20
.DDIR	5.333
Virtual Memory Instructions	
.LBP	1.733
.LBPR	2.00
.LPX	2.133
.LPXR	2.533
.IMAP (Basic)	3.60
Additional per parameter	2.00
.JMAP (Basic)	3.20
Additional per parameter	2.00
.IRES (Basic)	1.733
Additional per parameter	2.00
.JRES (Basic)	1.33
Additional per parameter	2.00
.PMAP	1.60
Operating System Instructions	
.WFI (Basic)	0.40
.WFI (Per loop)	0.133
.SIP	0.667
.CPUID	0.40
.FWID	1.733
Dynamic Mapping System Instruction Group	Refer to Section IV for detailed descriptions and execution times.

INSTRUCTION	EXECUTION TIME (μsec)
Floating Point Group	
(Single Precision)	
FAD, FSB	1.733
FMP	1.867
FDV	4.00
FIX	1.733
FLT	1.867
.FIXD	2.133
.FLTD	2.00
(Double Precision)	
.TADD, .TSUB	3.467
.TMPY	3.60
.TDIV	8.533
.TFXS	2.267
.TFTS	1.867
.TFXD	2.40
.TFTD	2.00

Scientific Instruction Set

Single-Precision Instructions	TIME (μs) Min.	Max.
Sine or Cosine	17.7	19.5
Tangent	20.3	24.0
Arc Tangent	13.5	20.5
Hyperbolic Tangent	9.7	25.0
Exponentiation	19.1	
Natural/Base 10 Logarithm	18.9	
Square Root	14.7	

Note: The maximum non-interruptible time is ≤ 15 microseconds.

Double-Precision Operations	TIME (μs) Min.	Max.
DPOLY	3.0	13.9
M	4.0	
N	3.7	
/ATLG	11.0	

SIS Accuracy:	RMS Relative Error	
	Single-Precision	Double-Precision
Sine	9.2E-8	1.2E-16
Cosine	7.7E-8	1.3E-16
Tangent	1.5E-7	1.9E-16
Arc Tangent	1.5E-7	2.3E-16
Hyperbolic Tangent	2.2E-7	5.5E-16
Square Root	6.7E-8	1.6E-17
Exponentiation	3.2E-7	8.8E-17
Natural Logarithm	1.2E-7	1.3E-16
Base 10 Logarithm	1.6E-7	1.3E-16

Table 3-5. Typical Base Set Instruction Execution Times (Continued)

INSTRUCTION	EXECUTION TIME (μsec)	
Vector Instruction Set		
INSTRUCTION	**EXECUTION TIME FIXED**	**LOOP***
Single-Precision Instructions		
VADD, VSUB	4.9	1.2
VMPY	5.1	1.2
VDIV	4.3	3.7
VSAD, VSSB	4.8	1.1
VSMY	5.1	1.2
VSDV	3.7	3.7
VPIV	6.8	1.6
VABS	4.8	1.1
VSUM	4.0	2.4
VNRM	4.4	2.4
VDOT	8.0	3.2
VMAX, VMIN	4.1	0.7 - 2.7
VMAB, VMIB	4.1	2.1 - 2.5
VMOV	2.8	0.7
VSWP	2.8	1.2
Double-Precision Instructions		
DVADD, DVSUB	6.1	2.0
DVMPY	6.1	2.0
DVDIV	4.7	7.2
DVSAD, DVSSB	5.7	1.6
DVSMY	5.9	1.6
DVSDV	4.0	7.2
DVPIV	8.4	2.7
DVABS	5.6	1.6
DVSUM	4.1	2.0
DVNRM	4.5	2.0
DVDOT	7.5	2.5
DVMAX, DVMIN	4.3	0.7 - 3.3
DVMAB, DVMIB	4.3	2.5 - 3.2
DVMOV	3.1	1.2
DVSWP	3.1	2.3

*Fixed time is instruction start-up time; loop time is the processing time per vector element. Total time equals fixed time plus the number of elements times loop time.

Maximum non-interruptible time is ≤ 15 microseconds for any VIS instruction.

NOTES: All times are in microseconds.

Memory refresh during a processor memory access, heavy DMA activity, or "misses" in the cache will degrade (lengthen) **all** instruction execution times.

Table 3-6. SIS Instruction Error Codes

INSTRUCTION	ERROR CODE
TAN	09OR if $\|x\| > 32768*\pi/4$
SQRT	03UN if $x < 0$
ALOG	02UN if $x \leq 0$
ATAN	None
COS	05OR if $\|x\| > 32768*\pi/4$
SIN	05OR if $\|x\| > 32768*\pi/4$
EXP	07OF if $x > 88.029678$
ALOGT	02UN if $x \leq 0$
TANH	None

Where:
OF = Integer or floating point overflow.
OR = Out of range.
UN = Floating point underflow.

ALOG* NATURAL LOGARITHM

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

Calculates the natural logarithm of the single precision floating point quantity contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error return will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

ATAN* ARCTANGENT

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

Calculates the arctangent of the single precision floating point quantity contained in the A- and B-registers. The result (in radians) is returned to the A- and B-registers. The overflow bit is cleared.

COS* COSINE

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

*For HP Assembly Language usage, refer to paragraph 3-48.

Calculates the cosine of the single precision floating point quantity (in radians) contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error return will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

SIN* SINE

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

Calculates the sine of the single precision floating point quantity (in radians) contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error return will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

EXP* E TO THE POWER X

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

Calculates e to the power x of the single precision floating point quantity contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error condition will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

ALOGT* COMMON LOGARITHM

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

Calculates the common logarithm of the single precision floating point quantity contained in the A- and B-registers. The result is returned to the A- and B-registers. A normal return will skip the next instruction. An error condition will execute the next instruction, set the overflow bit, and return an ASCII error code in the A- and B-registers.

TANH* HYPERBOLIC TANGENT

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

Calculates the hyperbolic tangent of the single precision floating point quantity contained in the A- and B-registers. The result is returned to the A- and B-registers. The overflow bit is cleared.

*For HP Assembly Language usage, refer to paragraph 3-48.

DPOLY* POLYNOMIAL EVALUATION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	1	0	1	1	0	0	1
F	S	X	X	X	X	X	X	X	X	X	X	X	X	X	T
D/I															
D/I															
D/I															
D/I															
D/I															

Memory Address

Evaluates a polynomial or quotient of polynomials using 64-bit floating-point. This is a seven-word instruction where:

Word 1 = Instruction code.
Word 2 = Sub-opcode.
Word 3 = Address of result Y (64-bit floating).
Word 4 = Address of argument X (64-bit floating).
Word 5 = Address of coefficient P_M (64-bit floating).
Word 6 = Address of numerator order M (integer).
Word 7 = Address of denominator order N (integer).

Define $P(Z) = P_M Z^M + P_{M-1} Z^{M-1} + \ldots + P_1 Z + P_0$
$Q(Z) = Z^N + Q_{N-1} + \ldots + Q_1 Z + Q_0$

The computation performed depends on the values of bits F, S, T of the sub-opcode:

F=0: Y = P(X)/Q(X)
F=1, S=0, T=1: Y = $P(X^2)/Q(X^2)$
F=1, S=0, T=0: Y = $X*P(X^2)/Q(X^2)$
F=1, S=1, T=1: Y = $P(X^2)/(Q(X^2)-P(X^2))$ (N>0)
F=1, S=1, T=0: Y = $X*P(X^2)/(Q(X^2)-X*P(X^2))$ (N>0)

Horner's Rule is used to evaluate the polynomial(s). The coefficients must be stored sequentially in memory, starting with P_M, in the order:

$P_M, P_{M-1}, \ldots, P_1, P_0, Q_{N-1}, \ldots, Q_1, Q_0$

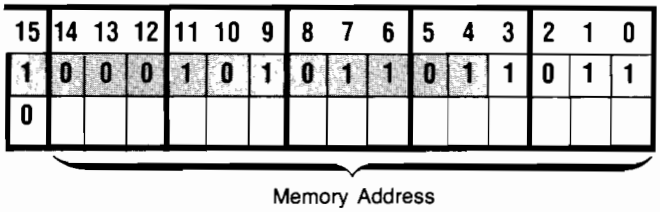
where $Q_N = 1.0$ is implied but not stored. If N=0, no coefficients are provided for Q and only P is evaluated. The case N=0 and S=1 is not allowed.

Any underflow or overflow which occurs invalidates the final result. M must be at least one. The A,B,X,Y and E registers are undefined after this instruction. The O register is undefined after the instruction.

This instruction is interruptible. Since it restarts after an interrupt, it is not recommended for very large values of (M+N).

Timing (in microseconds): Approximately
(3 to 14) + 4.0M + 3.7N

/ATLG* (1−X)/(1+X)



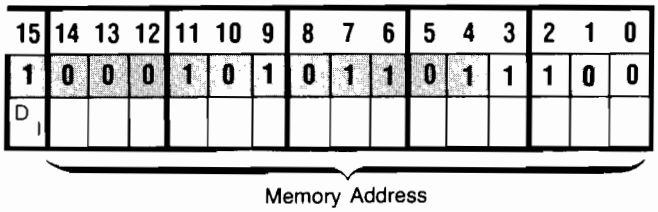
Performs the computation X =(1−X)/(1+X).

Word 1 = Instruction code.
Word 2 = Direct Address of X (64-bit floating).

The A,B,X,Y,E and O registers are undefined after this instruction.

Timing: 11.039 microseconds.

.FPWR* EXPONENTIATION



Raises a 32-bit floating-point number to an integer power. This is a two-word instruction, where:

Word 1 = Instruction code.
Word 2 = Address of base X (32-bit floating).

The power I is supplied in the A-register. It is unsigned and must be in the range [2,32768]. The left-to-right binary method is used to compute X^I, e.g. if $I=83_{10} = 123_8 = 1010011_2$ then

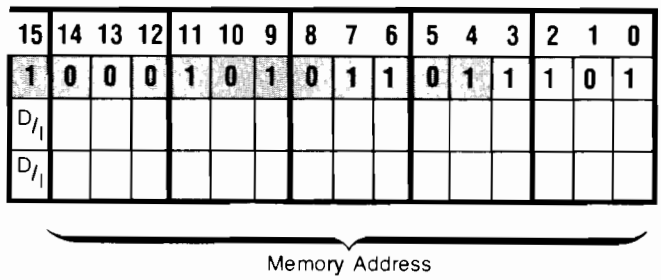
X^2	= $X*X$	10
X^4	= X^2*X^2	100
X^5	= X^4*X	101
X^{10}	= X^5*X^5	1010
X^{20}	= $X^{10}*X^{10}$	10100
X^{40}	= $X^{20}*X^{20}$	101000
X^{41}	= $X^{40}*X$	101001
X^{82}	= $X^{41}*X^{41}$	1010010
X^{83}	= $X^{82}*X$	1010011

The X, Y and E registers are undefined. The O register is set if underflow or overflow occur else cleared. The A- and B-registers contain the result.

Timing: Approximately 3.9 + 1.3M + 1.6N microseconds

where M = (# bits in I)
N = (# bits set in I)

.TPWR* EXPONENTIATION



Raises a 64-bit floating-point number to an integer power. This is a three-word instruction, where:

Word 1 = Instruction code.
Word 2 = Address of result (64-bit floating).
Word 3 = Address of base X (64-bit floating).

The power I is supplied in the A-register. It is unsigned and must be in the range [2,32768]. The left-to-right binary method is used.

The A,B,X,Y and E registers are undefined. The O register is set if underflow or overflow occurs, else cleared.

Timing: Approximately 4.7 + 1.9M + 2.1N microseconds

where M = (# bits in I)
N = (# bits set in I)

3-45. SIS EXECUTION TIMES AND INTERRUPTS

Table 3-5 lists the typical execution times required for the SIS instructions. Also listed is the maximum period of non-interruptible instruction execution. If an instruction is interrupted, its execution restarts from the beginning.

3-46. VECTOR INSTRUCTION SET

The Vector Instruction Set (VIS) performs arithmetic operations on arrays of floating point numbers. The VIS provides nineteen operations in both single and double precision formats, for a total of 38 instructions. For more information on the VIS instructions, refer to the Relocatable Library Reference Manual, part no. 92077-90037

WARNING

Only three levels of indirect instructions are allowed for any parameter. Failure to observe this constraint will produce unpredictable results.

*For HP Assembly Language usage, refer to paragraph 3-48.

Vector instructions require six to ten memory locations to specify parameters of the following type:

Opcode — Specifies the microcode entry point. Bit 4, or P-bit, indicates the precision of the operation. (P=0 for single precision, P=1 for double precision.)

Return address — Specifies the *direct* address of the next instruction.

The remaining parameters are addresses which may be direct or indirect, as indicated by bit 15. These include:

Vector — Specifies the address of the first vector element to be processed. Vector elements require two (single precision) or four (double precision) memory locations. All vectors in a given instruction must be of the same precision. Note that for instructions that contain two vector operands, these operands may both specify the same vector. Similarly, the result vector may replace one of the operands.

Scalar — Specifies the address of a single floating point quantity. Scalars are used for both operands or results. The precision of the scalar must match that of the associated vectors.

Integer — Specifies the address of an integer quantity in which a result is returned.

Increment — Specifies the address of an integer quantity associated with each vector. The increment indicates the spacing between vector elements to be processed. (An increment of 1 indicates that each element will be processed, an increment of 2 indicates every other element, etc.) An increment of zero will cause the first element of the vector to be used in all operations. Negative increments will step through the vector in reverse order, i.e., decreasing memory locations. Vector elements skipped over by the increment will not be modified.

#Elements — Specifies the address of an integer quantity indicating the number of vector elements to be processed. A value less than or equal to zero will result in a NOP operation.

The remaining parameters are addresses which may be direct or indirect, as indicated by bit 15. For further information on VIS instructions, refer to the Relocatable Library Reference Manual (HP part no. 92077-90037).

VADD/DVADD* VECTOR ADD

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

Memory Address

Performs the vector operation:

V3 = V1 + V2

This is a nine-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of vector V3.
Word 8 = Address of increment INCR3.
Word 9 = Address of # elements N.

VSUB/DVSUB* VECTOR SUBTRACT

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

Memory Address

Performs the vector operation:

V3 = V1 − V2

*For HP Assembly Language usage, refer to paragraph 3-48.

This is a nine-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of vector V3.
Word 8 = Address of increment INCR3.
Word 9 = Address of # elements N.

VMPY/DVMPY* VECTOR MULTIPLY

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

Memory Address

Performs the vector operation:

V3 = V1 * V2

This is a nine-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of vector V3.
Word 8 = Address of increment INCR3.
Word 9 = Address of # elements N.

*For HP Assembly Language usage, refer to paragraph 3-48.

VDIV/DVDIV* VECTOR DIVIDE

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

Memory Address

Performs the vector operation:

V3 = V1 / V2

This is a nine-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of vector V3.
Word 8 = Address of increment INCR3.
Word 9 = Address of # elements N.

VSAD/DVSAD* SCALAR-VECTOR ADD

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

Memory Address

Performs the vector operation:

V2 = S + V1

This is an eight-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar S.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of vector V2.
Word 7 = Address of increment INCR2.
Word 8 = Address of # elements N.

VSSB/DVSSB* SCALAR-VECTOR SUBTRACT

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

Memory Address

Performs the vector operation:

V2 = S − V1

This is an eight-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar S.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of vector V2.
Word 7 = Address of increment INCR2.
Word 8 = Address of # elements N.

VSMY/DVSMY* SCALAR-VECTOR MULTIPLY

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

Memory Address

Performs the vector operation:

V2 = S * V1

This is an eight-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar S.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of vector V2.
Word 7 = Address of increment INCR2.
Word 8 = Address of # elements N.

VSDV/DVSDV* SCALAR-VECTOR DIVIDE

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

Memory Address

Performs the vector operation:

V2 = S / V1

This is an eight-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar S.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of vector V2.
Word 7 = Address of increment INCR2.
Word 8 = Address of # elements N.

*For HP Assembly Language usage, refer to paragraph 3-48.

VPIV/DVPIV* VECTOR PIVOT

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

Memory Address

Performs the vector operation:

V3 = S * V1 + V2

This is a ten-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar S.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of vector V2.
Word 7 = Address of increment INCR2.
Word 8 = Address of vector V3.
Word 9 = Address of increment INCR3.
Word 10 = Address of # elements N.

VABS/DVABS* VECTOR ABSOLUTE VALUE

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

Memory Address

Performs the vector operation:

V2 = ABS (V1)

This is a seven-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of # elements N.

VSUM/DVSUM* VECTOR SUM

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

Memory Address

Performs the vector operation:

SUM = Σ V1

Note that for VSUM the sum is internally accumulated in double precision; the answer is then truncated to single precision.

This is a six-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar SUM.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of # elements N.

VNRM/DVNRM* VECTOR NORM

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

Memory Address

*For HP Assembly Language usage, refer to paragraph 3-48.

Performs the vector operation:

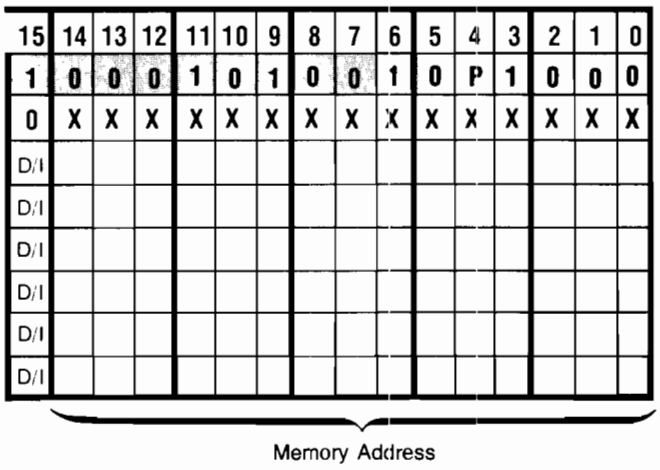
SUM = Σ ABS (V1)

Note that for VNRM the sum is internally accumulated in double precision; the answer is then truncated to single precision.

This is a six-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar SUM.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of # elements N.

VDOT/DVDOT* VECTOR DOT PRODUCT



Performs the vector operation:

DOT = Σ V1 * V2

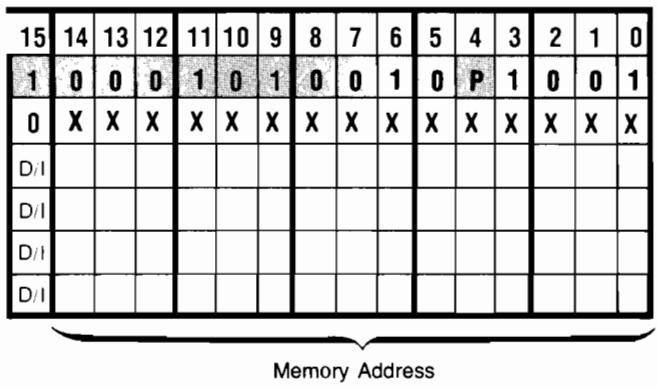
Note that for VDOT the product and sum is internally accumulated in double precision; the answer is then truncated to single precision.

This is an eight-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of scalar DOT.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of vector V2.
Word 7 = Address of increment INCR2.
Word 8 = Address of # elements N.

*For HP Assembly Language usage, refer to paragraph 3-48.

VMAX/DVMAX* VECTOR MAXIMUM VALUE



Performs the vector operation:

IMAX = Position (MAX(V1))

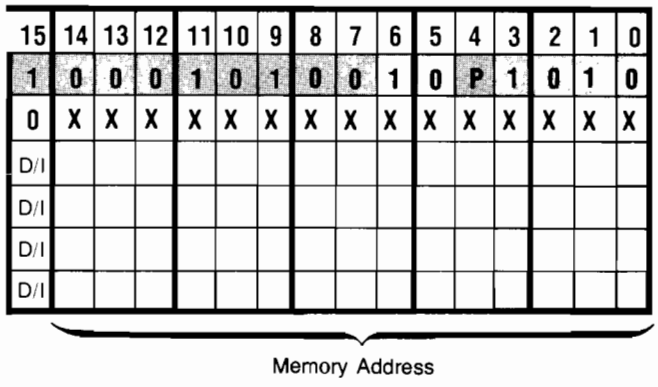
Note that IMAX is the position of the maximum of those elements that were tested, as requested by INCR1 and N. If INCR1 # 1, the position is given by:

IPOS = 1 + INCR1 * (IMAX − 1)

This is a six-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of integer IMAX.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of # elements N.

VMAB/DVMAB* VECTOR MAXIMUM ABSOLUTE VALUE



Performs the vector operation:

IMAB = Position (MAX(ABS(V1)))

Note that IMAB is the position of the maximum absolute value of those elements that were tested, as requested by INCR1 and N. If INCR1 # 1, the position is given by:

IPOS = 1 + INCR1 * (IMAB − 1)

This is a six-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of integer IMAB.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of # elements N.

VMIN/DVMIN* VECTOR MINIMUM VALUE

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

Memory Address

Performs the vector operation:

IMIN = Position (MIN(V1))

Note that IMIN is the position of the minimum absolute value of those elements that were tested, as requested by INCR1 and N. If INCR1 # 1, the position is given by:

IPOS = 1 + INCR1 * (IMIN − 1)

This is a six-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of integer IMIN.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of # elements N.

VMIB/DVMIB* VECTOR MINIMUM ABSOLUTE VALUE

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

Memory Address

Performs the vector operation:

IMIB = Position (MIN(ABS(V1)))

Note that IMIB is the position of the minimum absolute value of those elements that were tested, as requested by INCR1 and N. If INCR1 # 1, the position is given by:

IPOS = 1 + INCR1 * (IMIB − 1)

This is a six-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of integer IMIB.
Word 4 = Address of vector V1.
Word 5 = Address of increment INCR1.
Word 6 = Address of # elements N.

VMOV/DVMOV* VECTOR MOVE

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

Memory Address

Performs the vector operation:

V2 = V1

This is a seven-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of # elements N.

*For HP Assembly Language usage, refer to paragraph 3-48.

VSWP/DVSWP* VECTOR SWAP

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

Memory Address

Performs the vector operation:

V1 < = = > V2

This is a seven-word instruction, where

Word 1 = Instruction code.
Word 2 = Return address.
Word 3 = Address of vector V1.
Word 4 = Address of increment INCR1.
Word 5 = Address of vector V2.
Word 6 = Address of increment INCR2.
Word 7 = Address of # elements N.

3-47. VIS EXECUTION TIMES AND INTERRUPTS

Table 3-5 lists the typical execution times for the VIS instructions. VIS times are composed of two parts: a fixed time per instruction execution, and a per-element loop time. Thus the time to process 100 elements equals the fixed time plus 100 multiplied by loop time.

The maximum period of non-interruptible instruction execution for all VIS instructions is 15 microseconds.

When a VIS instruction is interrupted, the current state of execution is stored in the first word of the result memory location, as well as the A, B, X, and Y registers. When re-entered following the interrupt processing, the VIS instruction will resume execution from the point of suspension.

3-48. ASSEMBLY LANGUAGE

New instructions not recognized by the HP Macro-assembler require different handling in HP Assembly Language programming. These instructions are asterisked in the preceding paragraphs and must be used in the form: JSB x where x is the instruction. (The instruction, x, must be declared as an external at the beginning of the assembly language program.) Most of these instructions correspond to library subroutines* and must be implemented into HP RTE systems (as described in the following paragraph) to enable their execution in hardware-firmware instead of in software.

3-49. RTE IMPLEMENTATION

New instructions can be implemented in an HP RTE-A operating system simply by changing library entry points during the parameter input phase of system generation. (Refer to the appropriate RTE manual for the system generation procedure.) With the opcodes given in Table 3-7, the entry point changes would be as indicated below:

```
.JLA,RP,100600
.JLB,RP,104600
.
.
EXIT2,RP,105416
EXIT,RP,105417
```

Alternatively, entry points may be changed by loading (via LINK) a "replacement" module when user programs are loaded. Opcode replacement modules RPL90 and RPL91 are included in the RTE-A system software.

*For HP Assembly Language usage, refer to paragraph 3-48.

*Refer to the Relocatable Library Reference Manual, part no. 92077-90037.

Table 3-7. Instructions and Opcodes for RTE Implementation

INSTRUCTION MNEMONIC	OCTAL OPCODE	INSTRUCTION MNEMONIC	OCTAL OPCODE	INSTRUCTION MNEMONIC	OCTAL OPCODE
JLA	100600	.DIN	105210	STMP	105703
JLB	104600	.DDE	105211	LWD1	105704
.FAD	105000	.DIS	105212	LWD2	105705
.FSB	105020	.DDS	105213	SWMP	105706
.FMP	105040	.PMAP	105240	SIMP	105707
.FDV	105060	.IRES	105244	XJMP	105710
.FIX	105100	.JRES	105245	XJCQ	105711
.IFIX*	105100	.IMAP	105250	XLA2	101721
.FIXD	105104	.JMAP	105252	XSA2	101722
.FLT	105120	.LPXR	105254	XCA2	101723
FLOAT*	105120	.LPX	105255	XLA1	101724
.FLTD	105124	.LBPR	105256	XSA1	101725
.TADD	105002	.LBP	105257	XCA1	101726
.TSUB	105022	.CPUID	105300	XLB2	105721
.TMPY	105042	.FWID	105301	XSB2	105722
.TDIV	105062	.WFI	105302	XCB2	105723
.TFXS	105102	.SIP	105303	XLB1	105724
.TINT*	105102	VADD	105001	XSB1	105725
.TFXD	105106	VSUB	105003	XCB1	105726
.TFTS	105122	VMPY	105004	MB00	101727
.ITBL*	105122	VDIV	105005	MB01	101730
.TFTD	105126	VSAD	105006	MB02	101731
TAN	105320	VSSB	105007	MB10	101732
SQRT	105321	VSMY	105010	MB11	101733
ALOG	105322	VSDV	105011	MB12	101734
ATAN	105323	VPIV	105101	MB20	101735
COS	105324	VABS	105103	MB21	101736
SIN	105325	VSUM	105105	MB22	101737
EXP	105326	VNRM	105107	MW00	105727
ALOGT	105327	VDOT	105110	MW01	105730
TANH	105330	VMAX	105111	MW02	105731
DPOLY	105331	VMAB	105112	MW10	105732
/CMRT**	105332	VMIN	105113	MW11	105733
/ATLG	105333	VMIB	105115	MW12	105734
.FPWR	105334	VMOV	105116	MW20	105735
.TPWR	105335	VSWP	105117	MW21	105736
.DFER	105205	DVADD	105021	MW22	105737
.BLE	105207	DVSUB	105023	CCQA	101406
.NGL	105214	DVMPY	105024	CACQ	101407
.XFER	105220	DVDIV	105025	CZA	101410
.ENTR	105223	DVSAD	105026	CAZ	101411
.ENTP	105224	DVSSB	105027	CIQA	101412
.SETP	105227	DVSMY	105030	ADQA	101413
.CFER	105231	DVSDV	105031	PCALI	105400
..FCM	105232	DVPIV	105121	PCALX	105401
..TCM	105233	DVABS	105123	PCALV	105402
.ENTN	105234	DVSUM	105125	PCALR	105403
.ENTC	105235	DVNRM	105127	PCALN	105404
.CPM	105236	DVDOT	105130	SDSP	105405
.ZFER	105237	DVMAX	105131	CCQB	105406
.DAD	105014	DVMAB	105132	CBCQ	105407
.DSB	105034	DVMIN	105133	CZB	105410
.DMP	105054	DVMIB	105135	CBZ	105411
.DDI	105074	DVMOV	105136	CIQB	105412
.DSBR	105114	DVSWP	105137	ADQB	105413
.DDIR	105134	LPMR	105700	EXIT1	105415
.DNG	105203	SPMR	105701	EXIT2	105416
.DCO	105204	LDMP	105702	EXIT	105417

*Alternate mnemonic for the one preceding it.

**Not directly user callable. Used by HP 1000 software.

DYNAMIC MAPPING SYSTEM

SECTION IV

The basic addressing space of the HP 1000 A900 computer is 32768 words, which is referred to as logical memory. The amount of memory actually installed in the computer system is referred to as physical memory. The Dynamic Mapping System (DMS) is standard logic in the HP 1000 A900 computer and provides an addressing capability for up to 16 million words of physical memory. The DMS allows logical memory to be mapped into physical memory through the use of dynamically-alterable memory maps.

4-1. MEMORY ADDRESSING

The basic memory addressing scheme provides for addressing 32 pages of logical memory, each of which consists of 1024 words. This memory is addressed through a 15-bit logical address bus as shown in Figure 4-1. The upper 5 bits of this bus provide the logical page address and the lower 10 bits provide the relative word offset within the page.

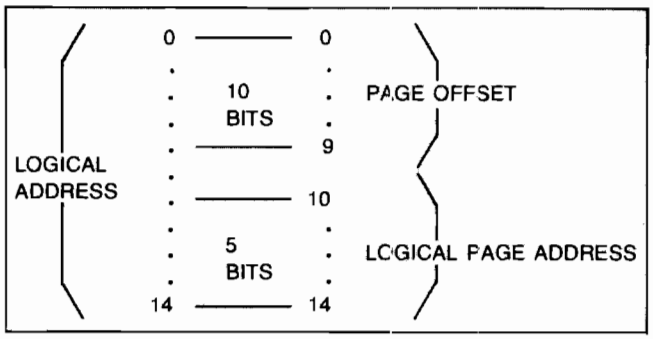


Figure 4-1. Basic Logical Memory Addressing Scheme

Also associated with any memory access is a 5-bit map number. The DMS converts the map number and the logical page address into a 14-bit physical page number, thereby allowing 16k (2^{14}) pages of physical memory to be addressed. This conversion is accomplished by having the 5-bit map number and the 5-bit logical page address access 1024 page mapping registers (PMRs), each of which is 16 bits wide. Each of these map registers contains the user-specified (privileged) 14-bit page address. This new page address is combined with the original 10-bit page offset to form a 24-bit memory address as shown in the Figure 4-2.

The PMRs also contain two bits of memory protection information. Bit 15 indicates that the page is read-protected when privileged mode is disabled. Bit 14 indicates that the page is write-protected when privileged mode is disabled. Any attempt to read from a read-protected page will result in a read violation and the memory read will return an undefined result. Any attempt to write into a write-protected page will result in a write violation and the memory will not be altered.

If a read or write violation occurs, the DMS signals the memory protect logic (located on the memory controller card) that a violation has occurred, which causes the memory protect logic to generate an interrupt. As discussed in Section VI, memory protect violations cause an interrupt to select code 07.

The width of the PMRs is limited to a 16-bit word, of which two bits specify read/write protection, so the maximum width of the physical page address is 14 bits.

4-2. GENERAL DESCRIPTIONS

4-3. PAGE MAPPING REGISTER INSTRUCTIONS

The page mapping register instructions allow the privileged user to alter the PMRs, each of which have the following format:

PAGE MAPPING REGISTER FORMAT

0 … 13	physical page number
14	— write protect this page
15	— read protect this page

The page mapping register instructions are:

LPMR - load a PMR indexed by register A from register B
SPMR - store a PMR indexed by register A to register B
LDMP - load a map from memory
STMP - store a map to memory

All of these instructions are privileged.

4-4. WORKING MAP INSTRUCTIONS

The computer will maintain three logical maps, cumulatively called the Working Map Set (WMAP). The working map instructions allow the system to alter the logical maps, and also to initiate a user program.

The Execute map is the map number used for instruction fetches and normal memory accesses. The data maps (DATA1 and DATA2) are the map numbers used in cross-map memory references. There are two data maps to allow the system to do cross-map moves from one area of memory to another without having to go through the system map. In addition, this feature allows the system to be able to quickly access one area of memory (such as a System Available Memory map) while being able to also access another (such as the user's map). A memory reference to locations 0 or 1 in the Execute map are defined to access the A- or B-registers, respectively. References to 0 or 1 in the data maps are defined to access physical memory locations.

The computer has an additional working map called the code map. The code map is defined as the Execute map that has been inclusively ORed with 1, following which the original Execute map is redefined as the data map. This use of separate maps for both code and data occurs only when CDS mode is enabled, and effectively doubles the logical address space for user programs.

The format of WMAP is as follows:

WMAP FORMAT:

0 . . 4	Execute map number
5 . . 9	DATA1 map number
10 . . 14	DATA2 map number
15	memory protection enable

Upon servicing interrupts, the computer saves the currently executing WMAP in a register called IMAP, and loads WMAP with the following values:

a. The DATA1 map is set to the old Execute map.

b. The new Execute map is set to zero.

c. The DATA2 map contains an undefined value.

d. Memory protection is disabled.

The working map instructions are:

XJMP - cross jump
XJCQ - cross map jump (and load C and Q)
SWMP - store current WMAP into memory
SIMP - store current IMAP into memory
LWD1 - load WMAP field DATA1 from memory
LWD2 - load WMAP field DATA2 from memory

All of these instructions are privileged.

4-5. CROSS-MAP INSTRUCTIONS

While the working map instructions provide a way to load the working map set, the cross-map instructions provide a means to use them.

These instructions are non-privileged. For all of these instructions, indirect DEF references are done through the Execute map, while the final reference is done through the specified map. When Code and Data Separation (CDS) is enabled, any memory accesses involving the Execute map number are considered to be data accesses, and the base register hardware will add the base (Q) register value to memory addresses from 2 through 1023. Memory ac-

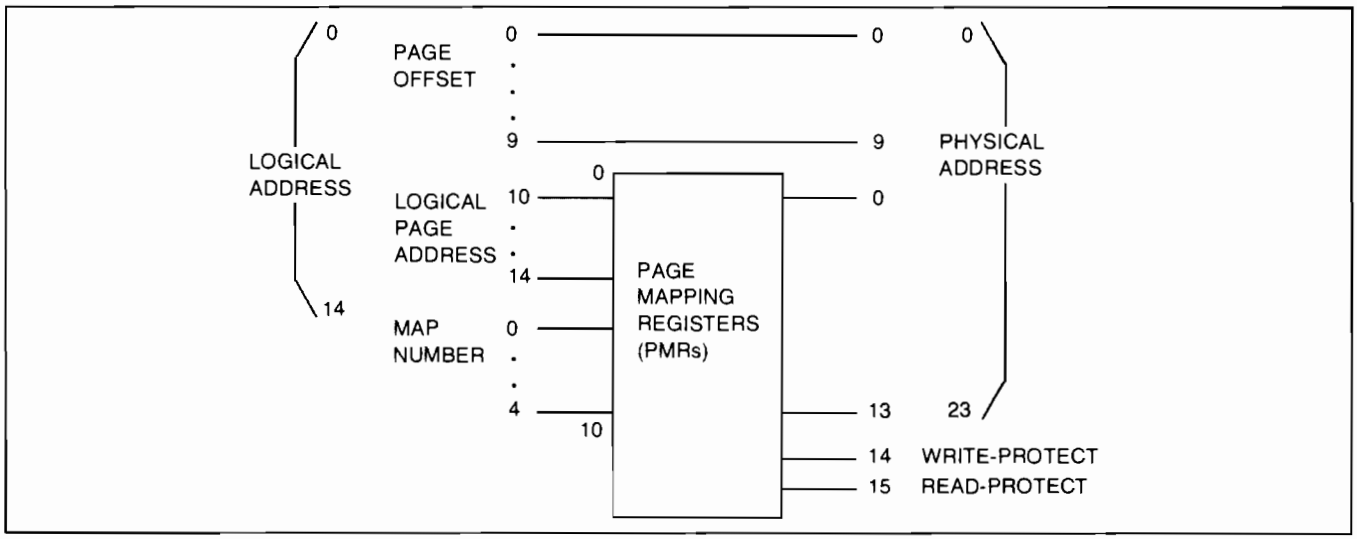


Figure 4-2. Expanded Memory Addressing Scheme

cesses involving the DATA1 or DATA2 map numbers are done with CDS disabled, so accesses to the base page will not have the base register added.

Abbreviations used are:

"0" - means logical Execute map
"1" - means logical DATA1 map
"2" - means logical DATA2 map

The cross map instructions are:

XLA1 - cross load A through the DATA1 map
XLB1 - cross load B through the DATA1 map
XLA2 - cross load A through the DATA2 map
XLB2 - cross load B through the DATA2 map
XSA1 - cross store A through the DATA1 map
XSB1 - cross store B through the DATA1 map
XSA2 - cross store A through the DATA2 map
XSB2 - cross store B through the DATA2 map
XCA1 - cross compare A through the DATA1 map
XCB1 - cross compare B through the DATA1 map
XCA2 - cross compare A through the DATA2 map
XCB2 - cross compare B through the DATA2 map
MW00 - cross move words from Execute to Execute
MW01 - cross move words from Execute to DATA1
MW02 - cross move words from Execute to DATA2
MW10 - cross move words from DATA1 to Execute
MW11 - cross move words from DATA1 to DATA1
MW12 - cross move words from DATA1 to DATA2
MW20 - cross move words from DATA2 to Execute
MW21 - cross move words from DATA2 to DATA1
MW22 - cross move words from DATA2 to DATA2
MB00 - cross move bytes from Execute to Execute
MB01 - cross move bytes from Execute to DATA1
MB02 - cross move bytes from Execute to DATA2
MB10 - cross move bytes from DATA1 to Execute
MB11 - cross move bytes from DATA1 to DATA1
MB12 - cross move bytes from DATA1 to DATA2
MB20 - cross move bytes from DATA2 to Execute
MB21 - cross move bytes from DATA2 to DATA1
MB22 - cross move bytes from DATA2 to DATA2

If CDS mode is enabled, the base (Q) register will be added to base relative addresses in the Execute map only. Cross map references to addresses in one of the alternate maps are not checked for base relativity.

4-6. DETAILED DESCRIPTIONS

The following paragraphs provide machine language coding and definitions for the DMS instructions. Note that all memory accesses are subject to the DMS memory protection rules.

LPMR LOAD PAGE MAPPING REGISTER

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

Loads the contents of the B-register into the page mapping register (PMR) addressed by the contents of the A-register. Any attempt to either address a PMR outside the range of 0 to 1023 or to modify a PMR that is currently being accessed produces undefined results. The format for the PMR contents is: bit 15 = read protect; bit 14 = write protect; and bits 13 to 0 = physical page number. This instruction is privileged. After the operation, the A-register is incremented.

SPMR STORE PAGE MAPPING REGISTER

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

Loads the contents of the page mapping register (PMR) addressed by the value in the A-register into the B-register. Any attempt to address a PMR outside the range of 0 to 1023 produces undefined results. The format for the PMR contents is: bit 15 = read protect; bit 14 = write protect; and bits 13 to 0 = physical page number. This instruction is privileged. After the operation, the A-register is incremented.

LDMP LOAD A MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	0
D_1															
D_0															

Loads the map number specified by Word 2 from the 32-word block of memory specified by Word 3, where:

Word 1 = Instruction code.
Word 2 = Pointer to Map number.
Word 3 = Pointer to Map image.

There are 32 maps of 32 PMRs each; the beginning PMR number of a map is related to the map number as follows:

PMR number = Map number × 32

Undefined results occur when a map number outside the range of 0 to 31 is addressed, when modification of a currently executing map is tried, or when the resolved address of the map image is outside the range of 2 to 77740 octal.

All memory references are done in the Execute map and may include the A- and B-registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

STMP STORE A MAP

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

Stores the map number specified by Word 2 to the 32-word block of memory specified by Word 3, where:

Word 1 = Instruction code.
Word 2 = Pointer to Map number.
Word 3 = Pointer to Map image.

There are 32 maps of 32 PMRs each; the beginning PMR number of a map is related to the map number as follows:

PMR number = Map number × 32

Undefined results occur when a map number outside the range of 0 to 31 is addressed or when the resolved address of the map image is outside the range of 2 to 77740 octal.

All memory references are done in the Execute map and may include the A- and B-registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XJMP CROSS MAP JUMP

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

Resolves indirect references, sets the program counter to the resolved address specified by Word 3, and loads WMAP with the value pointed to by the resolved address of Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to new WMAP number.
Word 3 = Pointer to next instruction (new PC value).

All memory references (direct and indirect) are done in the Execute map and may include the A- and B-registers. The next instruction will be fetched using the new WMAP. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XJCQ CROSS MAP JUMP (AND LOAD C AND Q)

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

Resolves indirect references, sets the program counter to the resolved address specified by Word 3, loads the WMAP specified by Word 2, and loads the C- and Q-registers with new values addressed by Word 4, where:

Word 1 = instruction opcode.
Word 2 = pointer to new WMAP number.
Word 3 = pointer to next instruction (new PC value).
Word 4 = pointer to new C- and Q-register values.

All memory references (direct and indirect) are done in the Execute map and may include the A- and B-registers. The next instruction will be fetched using the new WMAP, under a CDS mode specified by the new C-register value. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

SWMP SAVE WORKING MAP

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

Stores WMAP at the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to destination in memory.

All memory references are done in the Execute map and may include the A- and B-registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

SIMP SAVE INTERRUPTED MAP

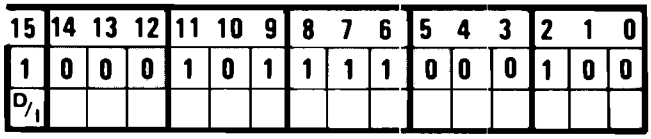
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	0	1	1	1
D/I															

Stores IMAP at the location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to destination in memory.

All memory references are done in the Execute map and may include the A- and B-registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

LWD1 LOAD DATA1 MAP

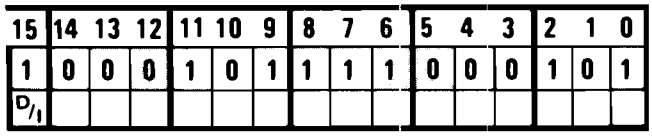


Loads the DATA1 field of the WMAP register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to new DATA1 map

All memory references are done in the Execute map and may include the A- and B-registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted. Map numbers outside the range of 0-31 produce undefined results.

LWD2 LOAD DATA2 MAP

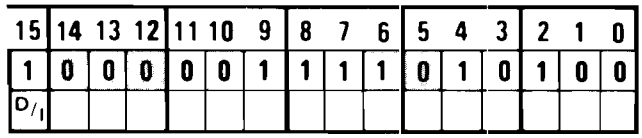


Loads the DATA2 field of the WMAP register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to new DATA2 map.

All memory references are done in the Execute map and may include the A- and B-registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted. Map numbers outside the range of 0-31 produce undefined results.

XLA1 CROSS LOAD A THROUGH DATA1 MAP

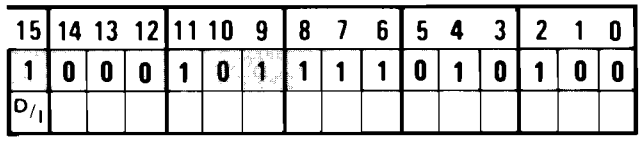


Loads the A-register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XLB1 CROSS LOAD B THROUGH DATA1 MAP

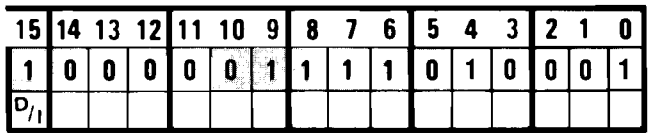


Loads the B-register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XLA2 CROSS LOAD A THROUGH DATA2 MAP

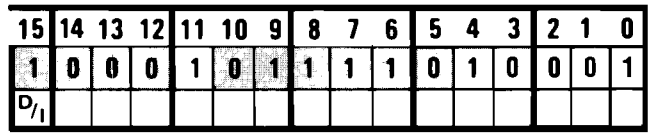


Loads the A-register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XLB2 CROSS LOAD B THROUGH DATA2 MAP



Loads the B-register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The

direct memory reference is done in the DATA2 map. Because A- and B-register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XSA1 CROSS STORE A THROUGH DATA1 MAP

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

Stores the A-register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XSB1 CROSS STORE B THROUGH DATA1 MAP

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

Stores the B-register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XSA2 CROSS STORE A THROUGH DATA2 MAP

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

Stores the A-register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XSB2 CROSS STORE B THROUGH DATA2 MAP

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

Stores the B-register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XCA1 CROSS COMPARE A THROUGH DATA1 MAP

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

Compares the A-register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-register addressing and base relative

checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XCB1 CROSS COMPARE B THROUGH DATA1 MAP

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

Compares the B-register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XCA2 CROSS COMPARE A THROUGH DATA2 MAP

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

Compares the A-register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XCB2 CROSS COMPARE B THROUGH DATA2 MAP

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

Compares the B-register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

MW00 CROSS MOVE WORDS, EXECUTE TO EXECUTE

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

Moves a block of words from the Execute map to the Execute map. The A-register specifies the source address, the B-register specifies the destination address, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the A- and B-registers will be checked for base relativity before execution. Upon exit these registers will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MW01 CROSS MOVE WORDS, EXECUTE TO DATA1

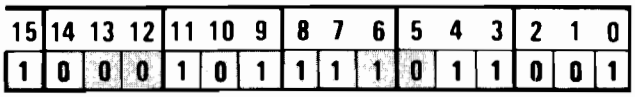
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	1	0	0	0

Moves a block of words from the Execute map to the DATA1 map. The A-register specifies the source address in the Execute map, the B-register specifies the destination address in the DATA1 map, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the A-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MW02 CROSS MOVE WORDS, EXECUTE TO DATA2

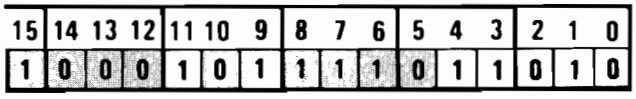


Moves a block of words from the Execute map to the DATA2 map. The A-register specifies the source address in the Execute map, the B-register specifies the destination address in the DATA2 map, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the A-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MW10 CROSS MOVE WORDS, DATA1 TO EXECUTE

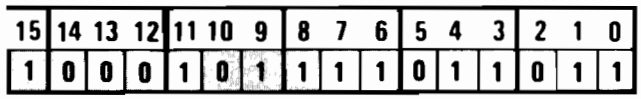


Moves a block of words from the DATA1 map to the Execute map. The A-register specifies the source address in the DATA1 map, the B-register specifies the destination address in the Execute map, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the B-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

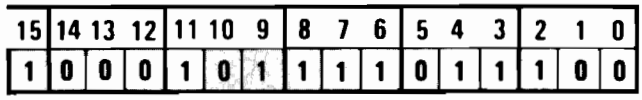
MW11 CROSS MOVE WORDS, DATA1 TO DATA1



Moves a block of words from one location in the DATA1 map to another in the DATA1 map. The A-register specifies the source address, the B-register specifies the destination address, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

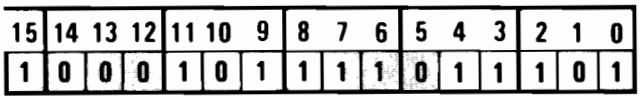
MW12 CROSS MOVE WORDS, DATA1 TO DATA2



Moves a block of words from the DATA1 map to the DATA2 map. The A-register specifies the source address in the DATA1 map, the B-register specifies the destination address in the DATA2 map, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MW20 CROSS MOVE WORDS, DATA2 TO EXECUTE

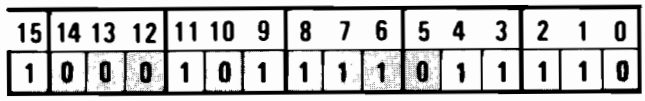


Moves a block of words from the DATA2 map to the Execute map. The A-register specifies the source address in the DATA2 map, the B-register specifies the destination address in the Execute map, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the B-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

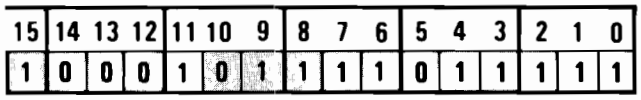
MW21 CROSS MOVE WORDS, DATA2 TO DATA1



Moves a block of words from the DATA2 map to the DATA1 map. The A-register specifies the source address in the DATA2 map, the B-register specifies the destination address in the DATA1 map, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

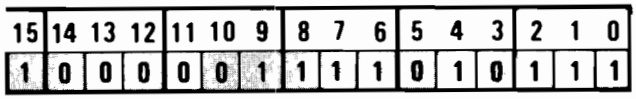
MW22 CROSS MOVE WORDS, DATA2 TO DATA2



Moves a block of words from the DATA2 map to the DATA2 map. The A-register specifies the source address, the B-register specifies the destination address, and the X-register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-register contains the last memory address in the source block moved plus one, the B-register contains the last memory address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the A-, B-, or X-register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB00 CROSS MOVE BYTES, EXECUTE TO EXECUTE

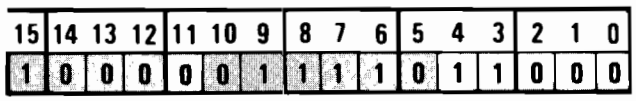


Moves a block of bytes from one location in the Execute map to another in the Execute map. The A-register specifies the source address and the B-register specifies the destination address. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the A- and B-registers will be checked for base relativity before execution. Upon exit these registers will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB01 CROSS MOVE BYTES, EXECUTE TO DATA1



Moves a block of bytes from a location in the Execute map to one in the DATA1 map. The A-register specifies the source address in the Execute map, and the B-register specifies the destination address in the DATA1 map. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the A-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB02 CROSS MOVE BYTES, EXECUTE TO DATA2

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

Moves a block of bytes from a location in the Execute map to one in the DATA2 map. The A-register specifies the source address in the Execute map, and the B-register specifies the destination address in the DATA2 map. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the A-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB10 CROSS MOVE BYTES, DATA1 TO EXECUTE

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

Moves a block of bytes from a location in the DATA1 map to one in the Execute map. The A-register specifies the source address in the DATA1 map, and the B-register specifies the destination address in the Execute map. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the B-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB11 CROSS MOVE BYTES, DATA1 TO DATA1

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

Moves a block of bytes from one location in the DATA1 map to another in the DATA1 map. The A-register specifies the source address and the B-register specifies the destination address. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB12 CROSS MOVE BYTES, DATA1 TO DATA2

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

Moves a block of bytes from a location in the DATA1 map to one in the DATA2 map. The A-register specifies the source address in the DATA1 map, and the B-register specifies the destination address in the DATA2 map. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address

uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB20 CROSS MOVE BYTES, DATA2 TO EXECUTE

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

Moves a block of bytes from a location in the DATA2 map to one in the Execute map. The A-register specifies the source address in the DATA2 map, and the B-register specifies the destination address in the Execute map. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

If CDS mode is enabled, the B-register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB21 CROSS MOVE BYTES, DATA2 TO DATA1

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

Moves a block of bytes from a location in the DATA2 map to one in the DATA1 map. The A-register specifies the source address in the DATA2 map, and the B-register specifies the destination address in the DATA1 map. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

MB22 CROSS MOVE BYTES, DATA2 TO DATA2

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

Moves a block of bytes from one location in the DATA2 map to another in the DATA2 map. The A-register specifies the source address and the B-register specifies the destination address. The X-register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-register contains the last memory byte address in the source block moved plus one, the B-register contains the last byte address in the destination block moved plus one, and the X-register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-registers.

4-7. DMS INSTRUCTION EXECUTION TIMES

Table 4-1 lists the execution times for the various DMS instructions.

4-8. ASSEMBLY LANGUAGE AND RTE IMPLEMENTATION

Refer to paragraphs 3-48 and 3-49 for information on implementing the DMS instructions in HP Assembly Language and in an HP RTE-A operating system.

Table 4-1. Dynamic Mapping Instructions Execution Times

INSTRUCTION	EXECUTION TIME (μs)
XLA1/XLB1, XLA2/XLB2	0.80
XSA1/XSB1, XSA2/XSB2	0.80
XCA1/XCB1, XCA2/XCB2	1.067
MW00, MW01, MW02, MW10, MW11, MW12, MW20, MW21, MW22	1.067 plus 0.267 per word moved
MB00, MB01, MB02, MB10, MB11, MB12, MB20, MB21, MB22	1.60 plus 0.133 per byte moved
LPMR	0.933
SPMR	0.933
LDMP	9.20
STMP	9.333
XJCQ	2.00
XJMP	1.867
LWD1	1.333
LWD2	1.20
SWMP	0.667
SIMP	0.667

Note: Memory refresh during a processor memory access, heavy DMA activity, or "misses" in the cache will degrade (lengthen) *all* instruction execution times.

CODE AND DATA SEPARATION

SECTION V

The basic logical address space of the HP 1000 A-Series architecture is 32768 words, in which both code and data reside. Code and Data Separation (CDS) is an enhancement to the A-Series architecture which separates code and data into separate logical address spaces. The main benefit of CDS is that it provides support of programs that may have up to 4M words of code, and this code may be either memory-resident or disc-resident. The optional HP 92078A package for RTE-A provides software support for CDS. Refer to the RTE-A Programmer's Reference Manual for a description of how to take advantage of CDS by using Macro/1000 and other HP languages.

5-1. CODE AND DATA ADDRESSING

CDS utilizes the Dynamic Mapping System environment of the A-Series architecture, and uses separate DMS maps to reference code and data. The term "code" refers to opcodes, DEFs to parameters, in-line constants, current-page links and constants for Memory Reference Group (MRG) instructions. The term "data" refers to variables and constants used by a program.

When CDS is disabled, both code and data are accessed through the logical address space of the computer, which is 32k words. The DMS maps this logical address space into the physical address space of up to 16M words. This is accomplished through the use of 32 memory maps of 32 pages each. A program executes in a single map, which is called the Execute map, although it may access memory through other maps using DMS instructions.

When CDS is enabled, code and data are accessed through separate maps. The Execute map number specifies which map is used to access data, and the Execute map number inclusive-ORed with '1' is used to access code. The Execute map number must be an even number between 0 and 30, inclusive. In all subsequent descriptions, DATA[n] and CODE[n] refers to memory locations in data space and code space, respectively. In addition, when CDS is enabled the base register (Q) is enabled, and all Execute map memory addresses that lie in the range 2 through 1023 have the Q-register added by the memory accessing hardware before the memory location is accessed. Locations 0 and 1 of data space are still defined to reference the A- and B-registers. Cross-map memory accesses, such as XLA1, are done with CDS disabled.

As an example, consider a DLD 500 instruction that is executed with CDS on, with an Execute map number of 2, and with the Q-register equal to 5000. The DLD opcode and the DEF 500 are read from memory using map number 3, because these words are considered to be code. The memory values loaded into A and B will be read through map number 2, because these words are considered to be data. The actual address of the memory locations to be loaded is 5500, because the hardware automatically adds the Q-register to memory addresses between 2 and 1023.

Most instructions separate code and data as was described for the previous example, but the Memory Reference Group has some exceptions. The JSB, STA current page direct, STB current page direct, and ISZ current page direct instructions may not be used when CDS is enabled because they attempt to write into code space. MRG references to base page always access memory in the data space, but MRG references to the current page always access code space for the first memory access and data space for all subsequent direct/indirect levels. That means that an LDA current page direct will load a constant from code space, that an LDA current page indirect will access a current page link in code space and then data in data space, and so on for the other MRG instructions. Note also that base page MRG references are useful for accessing variables that are Q-relative, such as the local variables or parameter pointers in a stack frame (to be described later).

The following restrictions must be met when CDS is enabled, otherwise undefined results may occur. The Q-register value must lie in the range of 1024 through 32767. The program counter must lie in the range 1024 to 32767, which means that jump instructions may not jump to the base page or to the A- or B-register.

Support for linking of relocatable code is provided by the RTE-A LINK program.

5-2. GENERAL DESCRIPTIONS

5-3. PROCEDURE CALL INSTRUCTIONS

The procedure call (PCAL) instructions are used to invoke a procedure, which may reside in code or data space. All of the PCAL instructions adjust the Q-register to allocate and set up a new stack marker (memory locations used to link procedure invocations and exits), and branch to the new procedure.

The PCAL instructions are:

- PCALI - procedure call to current segment
- PCALX - procedure call to any segment
- PCALV - procedure call to any segment (variable)
- PCALR - procedure call to .ENTR-compatible non-CDS code in data space
- PCALN - procedure call to .ENTN-compatible non-CDS code in data space

The PCALI instruction is the fastest PCAL instruction, and it is used to call a procedure that resides in the current code address space.

Two of the PCAL instructions (PCALX, PCALV) are capable of remapping the logical code space to another area of physical memory. Each logical code space is called a segment, and these PCALs are called cross-segment PCAL instructions.

The last two PCAL instructions (PCALR, PCALN) are used to call code that is not CDS-compatible. Such code resides in the data space, and must follow the .ENTR or .ENTN procedure call sequence.

The standard PCAL call sequence is:

```
    PCAL opcode (PCALI, PCALX, PCALV, PCALR, or PCALN)
    LABEL PE
    DEC AC [,I]
    DEF A_1 [,I]
      :
    DEF A_AC [,I]
    (return point from procedure PE)
      :
      :
PE  DEC FS
      (next instruction to be executed in procedure PE)
      :
    EXIT opcode (EXIT, EXIT1, or EXIT2)
```

The PCAL opcode is the appropriate opcode to be used to access the new procedure. If the new procedure is in the same segment, then PCALI should be used. If the new procedure is in another segment, then PCALX or PCALV should be used. If the new procedure is not CDS-compatible, then PCALR or PCALN should be used. Note that the selection of the PCAL opcode is done automatically by the RTE-A LINK program, which will also automatically segment your program for you.

The LABEL to the new procedure points to the location of the new procedure. In the case of PCALI, PCALR, PCALN, the LABEL is a DEF (a 15-bit logical address, possibly indirect) to the new procedure. In the case of PCALX, the LABEL consists of a word which contains information that determines how the logical code space must be remapped to get to the new procedure. In the case of PCALV, the DEF (which may be indirect) points to a word in data space which specifies how code space should be remapped.

AC is a word which specifies how many parameter pointers follow. Parameter pointers are 15-bit logical addresses (with the 16th bit specifying indirection) which point to variables that are being passed as parameters to the new procedure. From 0 to 255 parameter pointers may be passed in the PCAL call sequence.

5-4. PROCEDURE EXIT INSTRUCTIONS

There are three procedure exit instructions (EXIT, EXIT1, EXIT2). These instructions will remap the logical code space if necessary, adjust the Q-register value back to that of the calling procedure, and set the P-register to the return point in the calling procedure.

The EXIT instructions are:

EXIT - procedure exit with no skips
EXIT1 - procedure exit with one skip
EXIT2 - procedure exit with two skips

5-5. C, Q, Z, AND IQ INSTRUCTIONS

Other instructions are provided to access the C-, Q-, Z- and IQ-registers. These are:

CCQA (CCQB) - copy C and Q to A (or B)
CACQ (CBCQ) - copy A (or B) to C and Q
CZA (CZB) - copy Z to A (or B)
CAZ (CBZ) - copy A (or B) to Z
CIQA (CIQB) - copy IQ to A (or B)
ADQA (ADQB) - add Q to A (or B)
SDSP - store display

5-6. STACK FRAME DESCRIPTION

A stack frame is an area of memory in the logical data space that contains variables local to a procedure and pointers to variables of other procedures. The stack frame also contains six words of information called the stack marker, which links the procedure call chain from one procedure invocation to the next. The general layout of a stack frame is shown in Figure 5-1.

The Z-register, also called the bounds register, increases the reliability of CDS software. The bounds register detects the growth of a stack frame past the end of the allowed data space into areas used by VMA or memory used for other purposes. On every PCAL instruction, the microcode checks that the NEXT_Q value of a created stack marker is less than the Z-register. If this check fails then the program will interrupt to the memory protect handler (see PCALI description for more detail).

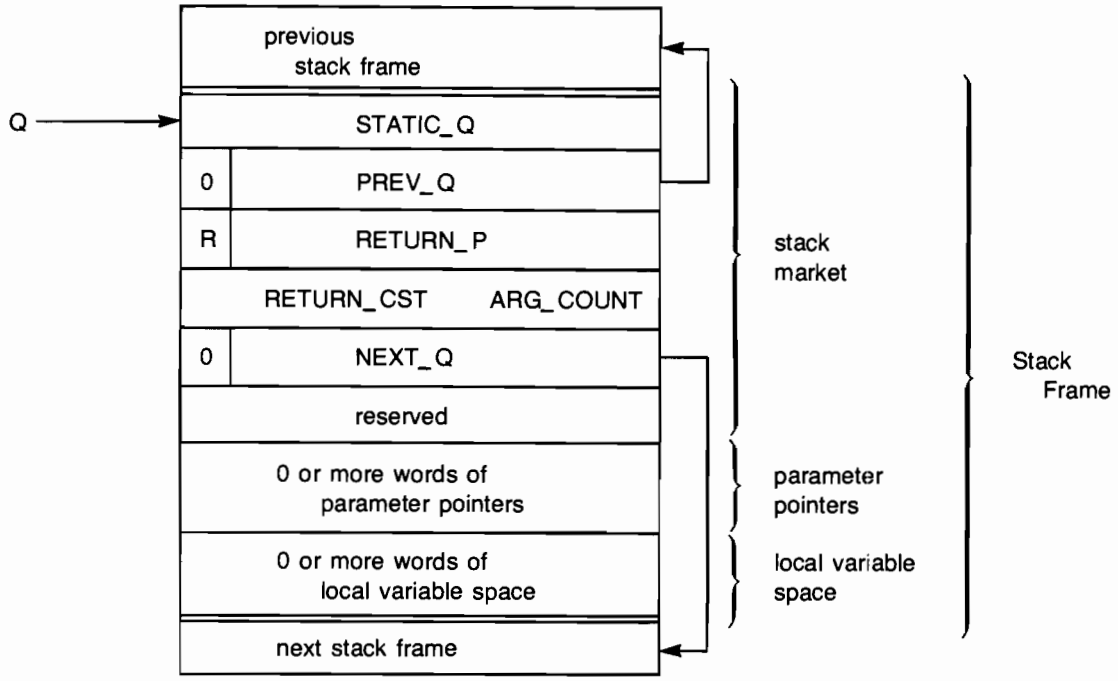
SOURCE : /RTEA/"CDSLB
LIBRARY : /RTEA/$CDSLB

```
macro,m,l,t
*
*
*    NAME:    "CDSLB
*    SOURCE: 92059-18027 REV.2326 820930
*    RELOC:   None
*    Pgmr:    R.S.N., M.O.S., G.W.K.
*
*    ****************************************************************
*    * (C) Copyright Hewlett-Packard Company 1983.  All rights    *
*    * reserved.  No part of this program  may be photocopied,    *
*    * reproduced  or translated  to another  program language    *
*    * without the  prior written  consent of  Hewlett-Packard    *
*    * Company.                                                   *
*    ****************************************************************
*
*
*        ENTRY macro comments
*        Brief Description:  This macro sets up a subroutine that will be
*          called using the standard calling format for CDS (PCALL SUBR
*          address, DEF P1, DEF P2, etc.).  It gives the subroutine formal
*          parameters names.  The CDS hardware automatically fills the
*          actual parameter value in at run time.
*        Registers Affected: all are clobbered
*        Unusual Side Effects/ Miscellaneous Notes: None so far
*        Parameters:
*          &NAME is the subroutine name
*          &FP1,&FP2,...,&FP10 are the formal parameters of the subroutine
*        Alternate Calling Formats/ Default Parameters:  may be called with
*          2 to 10 of the macro parameters &FP1 to &FP10; the ones not used
*          will default to ''.
*
         MACRO
&NAME    ENTRY &FP1,&FP2,&FP3,&FP4,&FP5,&FP6,&FP7,&FP8,&FP9,&FP10
         AIF :T:&.LocalMacroUsed <> 'U'
            AIF &.LocalMacroUsed <> 'Used last on END'
              MNOTE 'The LOCAL statement should not be used before ENTRY'
            AENDIF
         AENDIF
         RELOC CODE
         ENT &NAME
&NAME         DEF !#!LocalCount ;set up by END macro
         RELOC LOCAL
         AIF &FP1 <> ''
&FP1          NOP
         AENDIF
         AIF &FP2 <> ''
&FP2          NOP
         AENDIF
         AIF &FP3 <> ''
&FP3          NOP
         AENDIF
         AIF &FP4 <> ''
&FP4          NOP
         AENDIF
         AIF &FP5 <> ''
&FP5          NOP
         AENDIF
         AIF &FP6 <> ''
&FP6          NOP
```

```
        AENDIF
        AIF &FP7 <> ''
&FP7        NOP
        AENDIF
        AIF &FP8 <> ''
&FP8        NOP
        AENDIF
        AIF &FP9 <> ''
&FP9        NOP
        AENDIF
        AIF &FP10 <> ''
&FP10       NOP
        AENDIF

        RELOC CODE
        ENDMAC
*
*       EXIT macro comments
*       Brief description:  This macro exits a subroutine that has been
*         entered with the ENTRY macro.
*       Registers Affected: none

        MACRO
        EXIT
            OCT 105417
        ENDMAC

*
*       EXIT1 macro comments
*       Brief description:  This macro exits a subroutine that has been
*         entered with the ENTRY macro.
*       Registers Affected: none

        MACRO
        EXIT1
            OCT 105415
        ENDMAC

*
*       EXIT2 macro comments
*       Brief description:  This macro exits a subroutine that has been
*         entered with the ENTRY macro.
*       Registers Affected: none

        MACRO
        EXIT2
            OCT 105416
        ENDMAC

*       LOCAL macro comments
*       Brief description:  This macro is used to declare the names and
*         sizes of the local variables in a CDS user's program.
*         It is used both to assure that the users will not try to
*         to initialize their variables (either overtly like 'FOO DEC 12'
*         or accidently like 'FOO NOP') and to assure that no locals
*         precede the ENTRY macro.  This is done via an assembly time
*         variable.
*       Only data is generated.
*       The current relocation space counter is modified to be LOCAL.
*
```

```
         MACRO
&Lname   LOCAL &size
         AIF :T:&.LocalMacroUsed = 'U'
&.LocalMacroUsed CGLOBAL 'Used last on LOCAL'
         AELSE
&.LocalMacroUsed CSET    'Used last on LOCAL'
         AENDIF
         RELOC LOCAL
&Lname   bss &size
         ENDMAC

*        END macro comments
*        Brief description:  Used only to set up the variable !#!LocalCount
*          to contain a count of the number of local words used by the program
*          This is put into the first word of the code via the ENTRY macro.

         MACRO
          END &param
          RELOC LOCAL
!#!LocalCount equ *
          AIF :T:&.LocalMacroUsed <> 'U'
&.LocalMacroUsed CSET 'Used last on END'
          AENDIF
          :OP:END &param
         ENDMAC

*
*        CALL macro comments
*        Brief Description:  This macro generates the code to call a sub-
*          routine with 0 to 10 parameters.
*        Registers Affected: depends on subroutine called
*        Unusual Side Effects/ Miscellaneous Notes: none
*        Parameters:
*          &NAME is subroutine name
*          &P1,&P2,...,&P10 are the subroutine parameters
*        Alternate Calling Formats/ Default Parameters: may be called with
*          0 to 10 of the macro parameters &P1 to &P10; the ones not used
*          will default to ''
*        Example: To generate the code,        call CALL as follows:
*                      pcal exec,1,1,0            CALL EXEC,=D6
*
         MACRO
         CALL &NAME,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10
         EXT &NAME
&parms   ilocal &.pcount-1
         pcal &NAME,&parms,1,0
         .CALLPARS &P1,&P2,&P3,&P4,&P5
         AIF &.PCOUNT > 6
            .CALLPARS &P6,&P7,&P8,&P9,&P10
         AENDIF
         ENDMAC



*        PCALL macro comments
*        Brief Description:  This macro generates the code to call a CDS
*          subroutine with 0 to 10 parameters.
*        Registers Affected: depends on subroutine called
*        Unusual Side Effects/ Miscellaneous Notes: none
*        Parameters:
```

```
*         &NAME is subroutine name
*         &P1,&P2,...,&P10 are the subroutine parameters
*       Alternate Calling Formats/ Default Parameters: may be called with
*         0 to 10 of the macro parameters &P1 to &P10; the ones not used
*         will default to ''
*       Example: To generate the code,       call PCALL as follows:
*                      pcal sub,1,3,0          PCALL SUB, P1
*
        MACRO
        PCALL &NAME,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10
        EXT &NAME
&parms  ilocal &.pcount-1
        pcal &NAME,&parms,3,0
        .CALLPARS &P1,&P2,&P3,&P4,&P5
        AIF &.PCOUNT > 6
           .CALLPARS &P6,&P7,&P8,&P9,&P10
        AENDIF
        ENDMAC


*  Macro to do the parameters for the CALLx guys.
*  Does 5 at a time; this makes CALLs
*  with fewer than 6 parameters faster.
*
*  Macro seems to do something strange, making pcount NOT
*  be what you might think it would be.
*
        MACRO
        .CALLPARS &P1,&P2,&P3,&P4,&P5
        AIF &P1 <> ''
            DEF &P1
        AENDIF
        AIF &P2 <> ''
            DEF &P2
        AENDIF
        AIF &P3 <> ''
            DEF &P3
        AENDIF
        AIF &P4 <> ''
            DEF &P4
        AENDIF
        AIF &P5 <> ''
            DEF &P5
        AENDIF
        ENDMAC

        end
```



PREV_Q is the Q-register value for the calling procedure.

RETURN_P is the return address in the calling procedure.

R is the return segment indicator: R = 0 indicates the return address is in the same segment as the calling procedure (a segment reload is not required), R = 1 indicates the return address is in segment RETURN_CST (a segment reload is required).

ARG_COUNT is a number (0-255) that is the count of actual parameters passed to the called procedure. This field is maintained for all PCAL instructions.

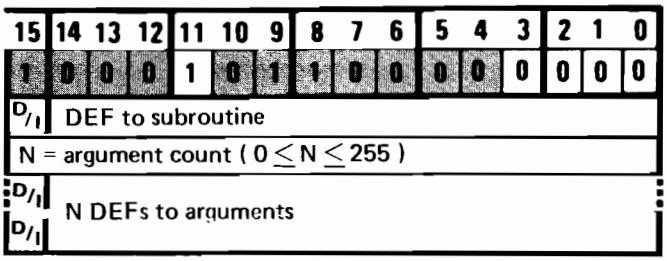
NEXT_Q is the Q-register value to use when building the next stack frame during a subsequent PCAL. NEXT_Q may be adjusted during the execution of a procedure to alter the size of the local variable space.

STATIC_Q is a word that is used by block-structured languages such as Pascal. This word and the RESERVED word are reserved for use by Hewlett-Packard software.

Figure 5-1. Stack Frame General Layout

5-7. DETAILED DESCRIPTIONS

PCALI INTERNAL PROCEDURE CALL



Function: Procedure call to current code segment

Use: *Current Code Segment*

```
PCALI
DEF pe [,I]
DEC ac
DEF a_1 [,I]
   :
DEF a_ac [,I]
```

Current Code Segment

```
pe  EQU *
    DEC fs
     :
     :
```

Operands:
- pe : Procedure entry point
- ac : Actual argument count
- a_i: Actual argument i (multiple indirects are supported)
- fs : Frame size in words

Interruptible: Yes

PCALI determines the new Q-register value for the called stack frame, which may be found at the current NEXT_Q value. The old Q value is written into the new stack frame at PREV_Q, which provides a link from the new stack frame to the old stack frame. The argument count (AC) of parameters to be passed is read from CODE[P+2], and the parameter pointers are copied from CODE[P+3] to DATA[new Q+6] after the parameter pointers have been resolved for indirection and base relativity. The value of AC is written into the ARG_COUNT location of the stack marker. Indirects are followed in memory until a direct address is found. If the (direct) address is between 2 and 1023, the current Q-register value is added before the parameter pointer is copied into data space. PCALI may be interrupted during parameter pointer resolution and copying, and the PCALI instruction may simply be restarted after the interrupt has been processed because the actual state of the calling procedure (specifically the P- and Q-registers) has not been altered.

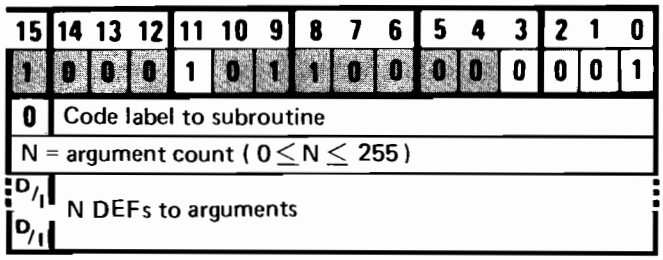
The actual parameter count (AC) is stored in the ARG_COUNT field of the new stack frame, and the upper byte of that word (RTN_CST) is undefined. The return point of the procedure (P+3+AC) is stored in the RETURN_P location of the new stack frame. The 'R' bit contains zero, which designates that a subsequent EXIT instruction should exit without loading a new segment.

The called procedure entry (PE) is found by resolving the address at CODE[P+1], and CODE[PE] contains the called frame size (FS). The NEXT_Q value of the new stack frame is set to the new Q value plus FS.

If the new NEXT_Q is greater than or equal to the bounds register (Z), stack overflow has occurred and a memory protect interrupt will be executed to memory location 07 of map zero. After the interrupt, the instruction violation register is equal to the fetch address of the PCAL instruction, and the program counter value at the time of the interrupt is undefined. The Q-register and IQ-register point to the offending stack marker. The new stack marker and formal arguments may have been written into memory locations at addresses greater than the Z-register value. To provide a safety zone, set the Z-register to 264 words below the area you want to protect.

If stack overflow did not occur, PCALI branches to the called procedure by setting the program counter, P, to PE+1 and the Q-register to the new Q value.

PCALX EXTERNAL PROCEDURE CALL



Function: Procedure call to procedure in external segment.

Use: *Current Code Segment*

```
PCALX
LABEL pe
DEC ac
DEF a_1 [,I]
   :
DEF a_ac [,I]
```

External Code Segment

```
pe  EQU *
    DEC fs
     :
     :
```

Operands: pe : Code label (Code Segment Table index and Segment Transfer Table index) to procedure.
ac : Actual argument count
a_i: Actual argument i
fs : Frame size in words

Interruptible: Yes

PCALX determines the new Q-register value for the called stack frame, which may be found at the current NEXT_Q value. The old Q value is written into the new stack frame at PREV_Q, which provides a link from the new stack frame to the old stack frame. The actual count (AC) of parameters to be passed is read from CODE[P+2], and the parameter pointers are copied from CODE[P+3] to DATA[new Q+6] after the parameter pointers have been resolved for indirection and base relativity. PCALX may be interrupted during the parameter pointer resolution and copying, and the PCALX instruction may simply be restarted after the interrupt has been processed because the actual state of the calling procedure (specifically the P- and Q-registers) has not been altered.

The return point of the procedure (P+3+AC) is stored in the RETURN_P location of the new stack frame. The 'R' bit contains one, which designates that a subsequent EXIT instruction should load the new segment indicated by RETURN_CST in the stack marker. The current segment number is read from CODE[2000B], ANDed with 177400B, inclusive ORed with AC, and stored in DATA[new Q+3].

PCALX now attempts to load the external segment. The upper byte of CODE[P+1] contains the CST (Code Segment Table) index. The PCALX instruction looks up the CST entry through the base page of the code map set. (The code map set number is the Execute map number inclusive ORed with one.) The memory address of the CST entry is the CST index shifted left two times. Restriction: the CST index must be in the range 0 through 127. Note that this process of looking up a CST entry is done with the base register hardware and A/B addressability off. If bit 15 of the CST entry is '1', then the called procedure is not in memory. PCALX will interrupt to memory location 13 octal of map zero and this location must contain a JSB to the segment interrupt handler. The program counter at the time of the interrupt points to the offending PCALX instruction, and the Q value is unchanged. After the segment is loaded, the PCALX instruction may be re-executed. The CDS segment interrupt is the lowest priority interrupt, and if other interrupts are present when a fault is detected, the instruction is simply restarted after the other interrupts are serviced. The following paragraphs describe what PCALX does if the segment is present in memory.

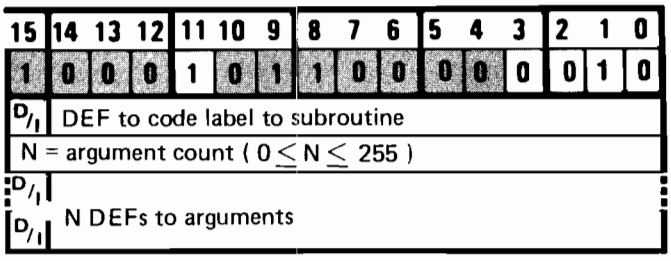
This paragraph describes how a code segment is 'mapped in'. The lower 14 bits of the CST entry contain the starting physical page of the new code segment, which the microcode maps in by setting the PMRs (page mapping registers) of code page 1 to the physical page number, code page 2 to the physical page number plus 1, code page 3 to the physical page number plus 2, and so on. These page mapping registers are write-protected to protect the code against alteration. The base page PMR of the code map is *not* altered.

After the new code segment has been mapped in, the entry point of the called procedure is determined. The low byte of the external label (in CODE[P+1] in the old segment) contains the STT (Segment Transfer Table) index. Beginning at location 2001B in code space is a table of address pointers (with bit 15 set to zero) that point to the externally accessible procedures in this segment. Location 2001B plus the STT index contains the 15 bit address of the called subroutine, and this value is the called procedure entry (PE).

CODE[PE] contains the called frame size (FS). The NEXT_Q value of the new stack frame is set to the new Q value plus FS. If the new NEXT_Q is greater than or equal to the bounds register (Z) then stack overflow has occurred, and a memory protect interrupt will be executed at memory location 07 of map zero. After the interrupt, the instruction violation register is equal to the fetch address of the PCALX instruction, and the program counter contains an undefined value. The Q-register and IQ-register point to the offending stack marker. The new stack marker and formal arguments may have written into memory locations at addresses greater than the Z-register value.

Now that the new stack marker is complete, PCALX branches to the called procedure by setting the program counter, P, to PE+1 and the Q-register to the new Q value.

PCALV VARIABLE EXTERNAL PROCEDURE CALL



Function: Procedure call, Code to Code, External procedure

Use: *Current Code Segment*

```
PCALV
DEF xl [,I]
DEC ac
DEF a_1 [,I]
   :
DEF a_ac [,I]
```

External Code Segment

```
pe  EQU *
    DEC fs
     :
     :
```

Data Segment

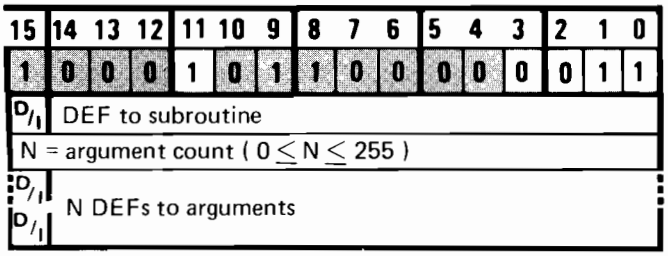
```
xl  LABEL pe
     :
```

Operands:
pe : Procedure entry point
xl : Procedure variable
ac ;: Actual argument count
a_i: Actual argument i
fs : Frame size in words

Interruptible: Yes

The difference between the PCALX and PCALV instructions is that the code label is in the call sequence in PCALX, while in PCALV it is in the data space. The pointer to the external label may be a multi-level indirect. See PCALX for a description of segment loading.

PCALR PROCEDURE CALL, .ENTR COMPATIBLE



Function: Procedure call, Code to Data, .ENTR compatible

Use: *Current Code Segment*

```
    PCALR
    DEF pe [,I]
    DEC ac
    DEF a_1 [,I]
      :
    DEF a_ac [,I]
```

Data Segment

```
    BSS fc
pe  NOP
    JSB .ENTR
    DEF pe-fc
      :
```

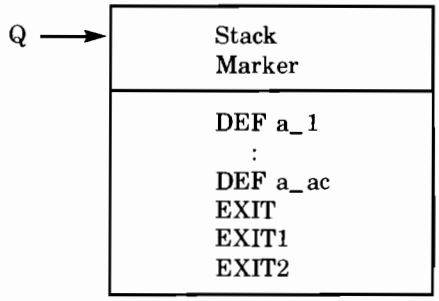
Operands:
pe : Procedure entry point
ac : Actual argument count
a_i: Actual argument i
fc : Formal argument count

Interruptible: Yes

PCALR is similar to PCALI except it is used for invoking procedures in the data segment that are .ENTR compatible. The mechanism for calling non-CDS-code involves copying a .ENTR call sequence (minus the JSB) into the stack frame. PCALR then turns off CDS, and executes the function of a JSB to the non-CDS-code procedure by writing a return address into the new procedure entry and branching to the procedure entry plus one. The procedure entry address must be between 1024 and 32766.

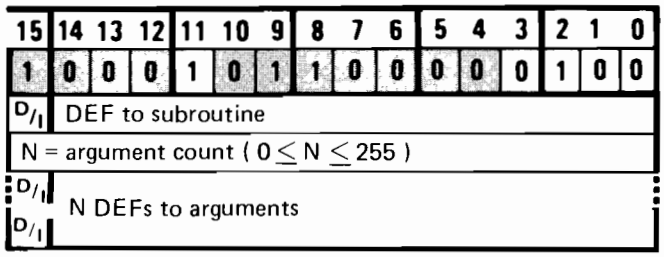
A "DEF *+AC+1" is written into the reserved word location (of the stack marker) for PCALR so as to follow the .ENTR calling convention.

The stack frame created by PCALR (and PCALN) is:



NEXT_Q in the stack marker is undefined.

PCALN PROCEDURE CALL, .ENTN COMPATIBLE



Function: Procedure call, Code to Data, Constant Internal procedure, .ENTN compatible

Use: *Code Segment*

```
    PCALN
    DEF pe [,I]
    DEC ac
    DEF a_1 [,I]
      :
    DEF a_ac [,I]
```

Data Segment

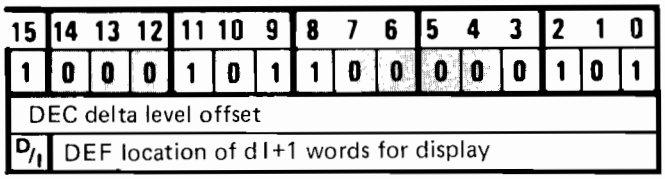
```
    BSS fc
pe  NOP
    JSB .ENTN
    DEF pe-fc
      :
```

Operands: pe : Procedure entry point
ac : Actual argument count
a_i: Actual argument i
fc : Formal argument count

Interruptible: Yes

The stack frame created by PCALN is similar to the stack frame created by PCALR. The difference between PCALR and PCALN is that the PCALR writes the return address at the non-CDS-code procedure entry, PE, with a return address of the new Q-register value plus 5, while PCALN writes a return address of the new Q value plus 6. Thus, the return address in PCALR points to a word that points around a parameter list (as in the .ENTR convention), while the return address in PCALN points to the parameter list (as in the .ENTN convention).

SDSP STORE DISPLAY



Function: Store display in memory.

Use: SDSP
DEC dl
DEF dsp [,I]

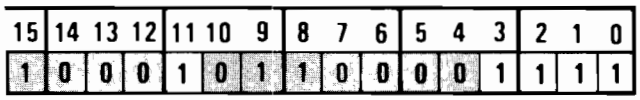
Operands: dl : delta level offset
dsp: location of dl+1 words for display

Interruptible: Yes

The store display instruction is used by block-structured languages such as PASCAL to store a number of STATIC_Q words into memory. SDSP begins by storing the current Q-register value into the DATA[disp]. The following is done dl times: the value just stored into memory is used as an address in memory, and this value, logically ANDed with 77777B, is stored in the word after the last word stored. The following table shows what is placed in the display by the SDSP instruction.

LOCATION	VALUE
disp	Q value for current procedue
disp+1	Q value for first lexically enclosing procedure
disp+2	Q value for second lexically enclosing procedure
...	
disp+dl	Q value for dl-th lexically enclosing procedure

EXIT PROCEDURE EXIT



Function: Exit from procedure.

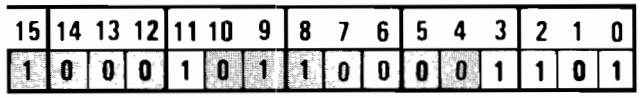
Use: EXIT

Interruptible: No

The EXIT instruction is used by any called procedure (in CDS mode or non-CDS mode) to return to the calling CDS procedure. The RETURN_P word in the stack marker holds the return address, and if bit 15 of that word is 1, then a new segment must be loaded first. The return segment is specified by the RETURN_CST field of the current stack marker. (See 'mapping in' in the PCALX description.) If the returning segment is not in memory, then an interrupt to memory location 13 octal of map zero will occur, with the P- and Q-registers unaltered by EXIT. The CDS segment interrupt is the lowest priority interrupt, and if other interrupts are present when a fault is detected then the instruction is simply restarted.

If EXIT was able to load the segment, or if the EXIT was to the current segment, then the C- and Q-registers are loaded from the PREV_Q word, and the P-register is set to RETURN_P.

EXIT1 PROCEDURE EXIT WITH ONE SKIP



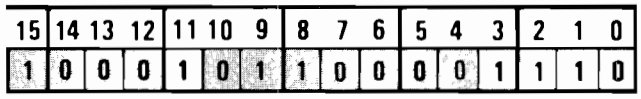
Function: Exit from procedure at normal exit + 1.

Use: EXIT1

Interruptible: No

EXIT1 is functionally identical to EXIT except that the program counter is set to RETURN_P plus one.

EXIT2 PROCEDURE EXIT WITH TWO SKIPS



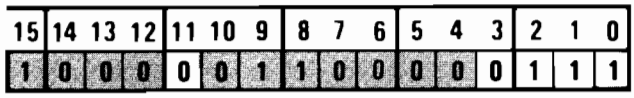
Function: Exit from procedure at normal exit + 2.

Use: EXIT2

Interruptible: No

EXIT2 is functionally identical to EXIT except that the program counter is set to RETURN_P plus two.

CACQ COPY A TO C AND Q



Function: Copy A- to C- and Q-registers

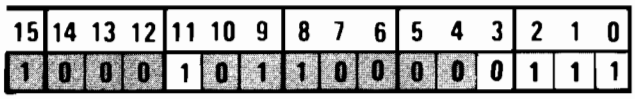
Use: CACQ

Operands: A : value to load into C and Q

Interruptible: No

The value contained in the A-register is copied to the C- and Q-registers. Bits 14 through 0 are copied into the Q-register. If bit 15 of the A-register is one, then CDS is turned off before the next instruction is fetched; otherwise, CDS is turned on.

CBCQ COPY B TO C AND Q



Function: Copy B- to C- and Q-registers

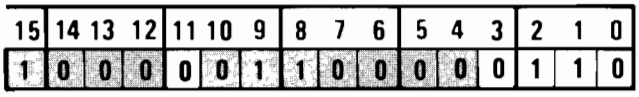
Use: CBCQ

Operands: B : value to load into C and Q

Interruptible: No

The value contained in the B-register is copied to the C- and Q-registers. Bits 14 through 0 are copied into the Q-register. If bit 15 of the B-register is one, then CDS is turned off before the next instruction is fetched; otherwise, CDS is turned on.

CCQA COPY C AND Q TO A



Function: Copy C- and Q-registers to A-register

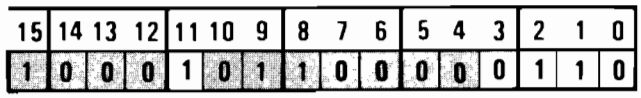
Use: CCQA

Operands: A gets values in C and Q

Interruptible: No

The C- and Q-registers are copied into the A-register. If CDS is enabled (C = 0), then bit 15 of the A-register is set to zero, otherwise, it is set to logic one.

CCQB COPY C AND Q TO B



Function: Copy C- and Q-registers to B-register

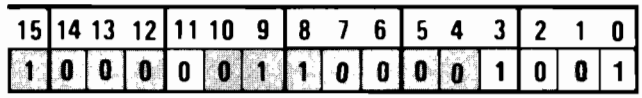
Use: CCQB

Operands: B gets values in C and Q

Interruptible: No

The C- and Q-registers are copied into the B-register. If CDS is enabled (C = 0), then bit 15 of the B-register is set to zero, otherwise, it is set to logic one.

CAZ COPY A TO Z



Function: Copy A-register to Z-register

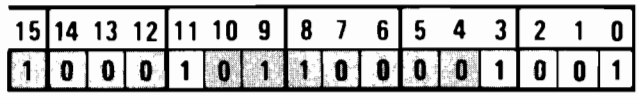
Use: CAZ

Operands: Z gets value in A

Interruptible: No

The contents of the A-register are copied into the Z-register. The results of setting bit-15 of the Z-register are undefined.

CBZ COPY B TO Z



Function: Copy B-register to Z-register

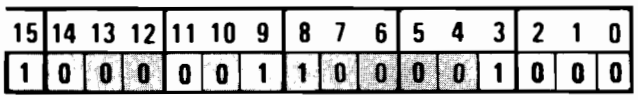
Use: CBZ

Operands: Z gets value in B

Interruptible: No

The contents of the B-register are copied into the Z-register. The results of setting bit-15 of the Z-register are undefined.

CZA COPY Z TO A



Function: Copy Z-register to A-register

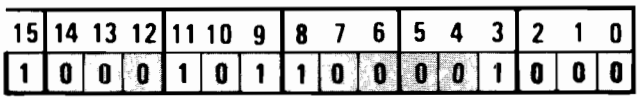
Use: CZA

Operands: A gets value in Z

Interruptible: No

The contents of the Z-register are copied into the A-register.

CZB COPY Z TO B



Function: Copy Z-register to B-register

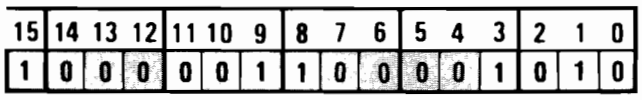
Use: CZB

Operands: B gets value in Z

Interruptible: No

The contents of the Z-register are copied into the B-register.

CIQA COPY INTERRUPTED Q TO A



Function: Copy interrupted Q-register to A-register

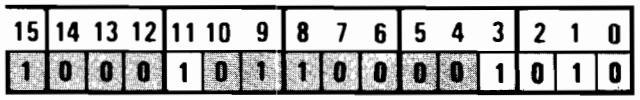
Use: CIQA

Operands: IQ : interrupted Q and C values

Interruptible: No

The A-register is loaded with the value of the IQ-register, which is the value of the C- and Q-registers at the time of the last interrupt or fault.

CIQB COPY INTERRUPTED Q TO B



Function: Copy interrupted Q-register to B-register

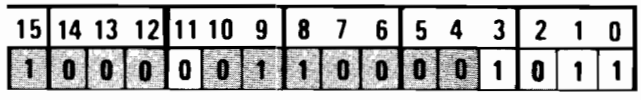
Use: CIQB

Operands: IQ : interrupted Q and C values

Interruptible: No

The B-register is loaded with the value of the IQ-register, which is the value of the C- and Q-registers at the time of the last interrupt or fault.

ADQA ADD Q TO A



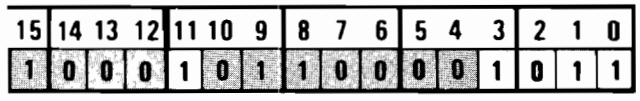
Function: Add Q-register to A-register

Use: ADQA

Interruptible: Yes

The Q-register is added to the A-register (A = A+Q). The ADQA instruction produces undefined results if executed while CDS is disabled.

ADQB ADD Q TO B



Function: Add Q-register to B-register

Use: ADQB

Interruptible: Yes

The Q-register is added to the B-register (B = B+Q). The ADQB instruction produces undefined results if executed while CDS is disabled.

5-8. ASSEMBLY LANGUAGE AND RTE IMPLEMENTATION

Refer to the Assembly Language and RTE Implementation paragraphs in Section III for information on implementing the CDS instructions in HP Assembly Language and in an HP RTE-A operating system.

5-9. EXECUTION TIMES

Table 5-1 shows the execution times for the CDS instructions.

Table 5-1. CDS Instruction Execution Times

INSTRUCTION	TIME (usec)
EXIT, EXIT1	
no segment mapping	0.931
with segment mapping	6.11
EXIT2	
no segment mapping	1.064
with segment mapping	6.11
PCALI (no parameters)	1.86
per parameter passed	0.4
per indirect	0.13
PCALX (includes segment mapping)	7.84
per parameter	0.4
per indirect	0.133
PCALV (includes segment mapping)	7.98
per parameter	0.4
per indirect	0.133
PCALR	2.66
per parameter	0.4
per indirect	0.133
PCALN	2.66
per parameter	0.4
per indirect	0.133
CACQ,CBCQ	0.53
CCQA,CCQB	0.53
CAZ,CBZ	0.53
CZA,CZB	0.53
CIQA,CIQB	0.66
ADQA,ADQB	0.53
SDSP	
display size = 0	0.93
per element of display	0.4

INTERRUPT SYSTEM

SECTION VI

The vectored priority interrupt system has up to 53 distinct interrupt levels, each of which has a unique priority assignment. In the A900 computer, the interrupt priority of an I/O card is based on the card's proximity to the processor card and is independent of the card's select code. The I/O card in the slot nearest to the processor card has the highest interrupt priority. Each I/O card has higher interrupt priority than I/O cards farther from the processor card and lower priority than cards closer to the processor card. As shown in Table 6-1, the select code of an interrupt level is associated with an interrupt location in memory.

Any device can be selectively enabled or disabled under program control, thus switching the device into or out of the interrupt structure. In addition, the interrupt system is divided into types of interrupts (Table 6-1). Interrupt Type 3 can be enabled or disabled under program control using a single instruction, and interrupt Types 2 and 3 combined can be enabled or disabled using a single instruction. Interrupt Type 4 cannot be disabled, but is lower priority than Types 1 through 3.

Table 6-1. A900 Interrupt Assignments

SELECT CODE (OCTAL)	INTERRUPT LOCATION	ASSIGNMENT	INTERRUPT TYPE
04	00004	Power Fail Interrupt	2
05	00005	Multi-bit Error Interrupt	1
06	00006	Time Base Generator Interrupt	3
07	00007	Memory Protect Interrupt	2
10	00010	Unimplemented Instruction Interrupt	1
11	00011	Reserved	
12	00012	Virtual Area Memory Interrupt	4
13	00013	CDS Segment Interrupt	4
14-17	00014-00017	Reserved	
20-77	00020-00077	I/O Card Interrupts	3

When a qualified interrupt is serviced, the state of the interrupted routine is saved in the IMAP and IQ registers, CDS mode and memory protect are turned off, the Execute map is set to 0 (System map), and the DATA1 map is set to the Execute map of the interrupted process. After this new state of the machine is set up, a fetch from the appropriate trap cell is performed. Trap cells are expected to contain a JSB to an appropriate routine that will service the interrupt and restore the state of the interrupted process before restarting it. If a JSB or JMP instruction is not contained in the trap cell, instruction execution will proceed at the address which was interrupted, but in the System map.

6-1. POWER FAIL INTERRUPT

The computer power supply is equipped with power-sensing circuits. When primary line power fails or drops below a predetermined level while the computer is running, an interrupt to memory location 00004 is automatically generated. Memory location 00004 is intended to contain a jump-to-subroutine (JSB) instruction referencing the entry point of a user-supplied power fail subroutine. The interrupt capability of lower-priority operations is automatically inhibited while a power fail subroutine is in process. (That is, the equivalent of a CLC 04 automatically occurs whenever a power fail interrupt is serviced.)

A minimum of five milliseconds is available between the detection of a power failure and the loss of usable power-supply power to execute a power fail subroutine; the purpose of such a routine is to transfer the current state of the computer system into memory and then wait for power to return. After the macro level state has been saved, an STF 4 instruction must be coded. This allows the internal state of the CPU to cleaned up (cache flushed, etc.) before power disappears. A sample power fail subroutine is given in Table 6-2. The optional battery backup module will supply enough power to preserve the contents of memory for a sustained line power outage of up to 180 minutes.

The user has a switch-selectable option of what action the computer will take upon restoration of primary power. When switch U0101S8 (labeled "M") on the data path card is closed, the computer will execute either a loader or the Virtual Control Panel routine, depending on the setting of the Start-Up switches. (The Start-Up switches are described in paragraph 2-23 and tabulated in Table 2-1.)

NOTE

Switch U0101S8 is mounted on the data path card and is not an operator control. The setting of this switch is normally determined by the System Manager prior to or during system installation.

When switch U0101S8 is open, the automatic restart feature is enabled. After the self-test is executed following the return to normal power levels, an interrupt to location 00004 occurs. This time the power-down portion of the

Table 6-2. Sample Power Fail Subroutine

LABEL	OPCODE	OPERAND	COMMENTS
PFAR	NOP		Power Fail/Auto Restart Subroutine.
	SFC	4B	Skip if interrupt was caused by power failure.
	JMP	UP	Power being restored; reset state of system.
DOWN	CLC	0B	Shut down any DMA or I/O.
	STA	SAVA	Save A-register contents.
	CCA		Set flag indicating that computer was running when power failed.
	STA	PFFLG	
	STB	SAVB	Save B-register contents.
	ERA,ALS		Transfer E-register content to A-register bit 15.
	SOC		Increment A-register if Overflow is set.
	INA		
	STA	SAVEO	Save E- and O-register contents.
	LDA	PFAR	Save contents of P-register at time of power failure.
	STA	SAVP	
	SIMP		
	DEF	SAVI	Save IMAP contents.
	.		
	.		Insert user-written routine to save I/O states.
	.		
	STF	4B	Clean up internal state (flush cache, etc.)
	SFS	4B	
	JMP	*−1	Wait in case power comes back up.
UP	LDA	PFFLG	Was computer running when power failed?
	SZA,RSS		
	HLT	4B	No, then halt.
	CLA		Yes, then reset computer Run flag to initial state.
	STA	PFFLG	
	.		
	.		Insert user-written routine to restore I/O devices.
	.		
	LDA	SAVEO	Restore the contents of the E-register and O-register.
	CLO		
	SLA,ELA		
	STF	1B	Set O-register.
	LDA	SAVA	Restore A-register contents.
	LDB	SAVB	Restore B-register contents.
	STC	4B	Reset power fail logic for next power failure.
	XJMP		Cross jump to program executing at power failure.
	DEF	SAVI	
	DEF	SAVP,I	
SAVEO	OCT	0	Storage for E and O.
SAVA	OCT	0	Storage for A.
SAVB	OCT	0	Storage for B.
SAVP	OCT	0	Storage for P.
PFFLG	OCT	0	Storage for Run flag.
SAVI	OCT	0	Storage for IMAP.

Note: The memory maps used must be saved and restored, as must (if used) the states of the interrupt mask register, memory protect (conditional restore), and Global Register.

subroutine is skipped and the power-up portion begins. (Refer to Table 6-2.) Those conditions existing at the time of the power fail interrupt are restored and the computer continues the program from the point of the interruption.

Note that an auto-restart interrupt to location 00004 occurs only if that location's contents are not zero; otherwise, the system is re-booted. This is done so that if power fails and is restored during a boot, an attempt to restart a partially loaded program can be avoided. To enable this to happen the program being loaded should initially load location 00004 with zero and load the power-fail JSB instruction only when the load is otherwise complete.

If the computer memory does not contain a subroutine to service the power fail interrupt, location 00004 should contain a NOP instruction (00 octal).

At the end of a restart routine, consideration should be given to re-initializing the power-fail logic and to restoring the interrupt capability of the lower priority functions.

6-2. MULTIPLE-BIT ERROR INTERRUPT

As a replacement for the parity checking of memory that exists in other HP 1000 computers, a standard feature of the A900 is error-correcting memory. Consequently, the 'parity error' interrupt has been replaced by a 'multiple-bit error' interrupt. The meaning is still the same: the data from the offending memory read cannot be trusted. (Unlike parity error interrupts, the error-correcting capability of the A900 cannot be disabled.)

The Error Detection and Correction (EDC) circuitry checks the memory array cards and the path between them and the cache. Whenever the cache writes to memory the EDC logic generates seven checkbits which accompany two 16-bit data words so that a memory array card stores a total of 39 bits per write. When the cache requests data from memory the EDC logic checks the 39 bits coming from the memory array card to determine whether an error of one or more bits has occurred. This determination is made by using the 32 bits of read data to recompute the seven checkbits, and comparing these checkbits with the old ones. The result of this comparison is called the error syndrome.

If a single-bit error is detected, it is automatically corrected before the data is sent to the cache. The even-numbered physical address of the two-word block containing the error and the error syndrome are latched into the parity violation register (PVR). If the error is determined to be a multiple-bit error, it is treated differently because it cannot be corrected. The data is sent to the cache without modification and a multiple-bit error interrupt to memory location 00005 is generated. The even physical address is still latched into the PVR, but then the PVR is frozen to prevent this address from being overwritten by another error. (Location 00005 may contain either a JSB instruction referencing the entry point of a user-supplied error subroutine (included with RTE) or a JMP instruction pointing to a HLT instruction.)

The PVR is accessible to the user through select code 05. An LIA 05 or LIB 05 provides bits 0 through 15 of the physical address. An LIA 5,C or LIB 5,C provides bits 16 through 23 of the physical address, the 7-bit syndrome, and a bit that indicates whether the error was a single-bit error or a multiple-bit error. An LIA 5,C or LIB 5,C also clears the PVR syndrome bits and the multiple-bit error interrupt. The multiple-bit error interrupt capability is enabled after power-up but may be disabled and enabled by CLC 05 and STC 05 instructions, respectively.

An OTA 05 or OTB 05 can be used to force memory errors to test the EDC logic; the A- or B-register specifies the address where the checkbit pattern held in the X-register will be written.

6-3. MEMORY PROTECT INTERRUPT

The memory protect feature provides the capability of protecting selected pages of memory against access or alteration by programmed instructions. References to the A- and B-registers as locations 00000 and 00001 are not affected.

The memory protect logic, when enabled by an STC 07 instruction, also prohibits the execution of all privileged instructions. This includes the I/O instructions except those referencing I/O select code 01 (the processor card status register and the overflow register). (Execution of all HLTs is prohibited.) This feature limits control of I/O operations to interrupt control only. Instructions which control the DMS are also privileged. Thus, an executive program residing in protected memory can have exclusive control of the DMS and the I/O system.

The memory protect logic is disabled automatically by any interrupt and must be re-enabled by an STC 07 or XJMP instruction at the end of each interrupt subroutine.

Programming rules pertaining to the use of memory protect are as follows (assuming that an STC 07 instruction has been given):

a. Locations 00000 and 00001 in the Execute map are the A- and B-registers and are not in protected memory. Locations 00000 and 00001 in the DATA1 and DATA2 maps are real memory locations (not the A- and B-registers) and may reside in protected memory.

b. A user-specified 1024-word page of memory is read and/or write protected by Page Mapping Register instructions described in Section IV.

c. Execution will be inhibited and an interrupt to location 07 will occur if any instruction attempts to read (or fetch) from read-protected memory, write to write-protected memory, or if any privileged instruction is attempted (this excludes those addressing select code 01 but not HLT 01). After three successive levels of indirect addressing, the logic will allow a pending I/O interrupt.

Following a memory protect interrupt, the address of the offending instruction will be present in the violation register. This address is made accessible to the programmer by an LIA 07 or LIB 07 instruction, which loads the address into the A- or B-register.

Note that DMA operation is not affected by memory protect.

6-4. UNIMPLEMENTED INSTRUCTION INTERRUPT

An unimplemented instruction interrupt (to memory location 00010 octal) is requested when the last instruction fetched was not recognized by the CPU. This interrupt provides a straightforward entry to software routines for the execution of instruction codes not recognized by the computer hardware. The unimplemented instruction interrupt must receive immediate service in order to recover the instruction code that caused it. For this reason, and because it is desirable to permit the use of unimplemented instructions anywhere, the unimplemented instruction interrupt is never inhibited.

6-5. TIME BASE GENERATOR INTERRUPT

A time base generator interrupt request is made when the CPU signals that its internal clock divider chain has rolled over. The clock divider is set to roll over at 10-millisecond intervals for maintaining a real-time clock. The interrupt occurs through location 00006 and can be masked (inhibited) by using bit 1 of the interrupt mask register. (The interrupt mask register allows interrupts from the TBG and the I/O cards to be selectively masked. For details on the interrupt mask register, refer to the *HP 1000 L-Series Computer I/O Interfacing Guide,* part no. 02103-90005.) The TBG can be turned on by an STC 06 instruction, and turned off by a CLC 06 or CLC 00 instruction. The first TBG interrupt will be requested 10 milliseconds after an STC 06.

6-6. VIRTUAL MEMORY AREA INTERRUPT

During the execution of a VMA instruction, the hardware may determine that the desired VMA address does not reside in physical memory and needs to be loaded from disc. This causes a VMA interrupt to memory location 000012 (octal). This interrupt can occur only when Code and Data Separation (CDS) is enabled.

6-7. CDS SEGMENT INTERRUPT

During the execution of a CDS instruction, the hardware may determine that a desired CDS segment does not reside in physical memory and needs to be loaded from disc. This causes a CDS segment interrupt to memory location 000013 (octal).

6-8. INPUT/OUTPUT INTERRUPT

Interrupt locations 20 through 77 (octal) are reserved for I/O devices. In a typical I/O operation, the computer issues a programmed command such as Set Control/Clear Flag (STC,C) to one or more external devices to initiate an input (read) or an output (write) operation, via either programmed I/O or DMA. While the I/O card is in the process of transferring data, the computer may be either running a program or looping, waiting for a flag to get set. Upon completion of the read or write operation, the interface flag is set. If the corresponding control bit is set, the interface will interrupt. Its request will be passed through a priority network so that only the highest priority interrupting device will receive service. The computer will acknowledge the interrupt and the highest priority device will receive service when the current instruction has finished executing, except under the following circumstances:

a. Interrupt system disabled or interface card interrupt disabled (or masked).

b. JMP indirect, JSB indirect, XJMP, or XJCQ instruction not sufficiently executed. These instructions inhibit all TBG, power fail, and I/O interrupts until the succeeding instruction is executed. After three successive levels of indirect addressing, the logic will allow a pending I/O interrupt.

c. A DMA (direct memory access) data transfer is in process.

d. Current instruction is any I/O group instruction (except those to select code 01). The interrupt in this case must wait until the succeeding instruction is executed.

After an interface card has been issued a Set Control (STC instruction) and its flag bit becomes set, all interrupt requests from lower-priority devices are inhibited until this flag bit is cleared by a Clear Flag (CLF) instruction, or until control is cleared by a Clear Control (CLC) instruction. A service subroutine in process for any device can be interrupted only by a higher-priority device; then, after the higher-priority device is serviced, the interrupted service subroutine can continue. In this way it is possible for several service subroutines to be in the

interrupt state at one time; each of these service subroutines will be allowed to continue after the higher-priority device is serviced. All such service subroutines normally end with a JMP indirect or XJMP instruction to return the computer to the point of interrupt.

Note that interrupt trap cells must contain a JMP or JSB instruction; any other trap-cell instruction produces undefined results.

6-9. INTERRUPT PRIORITY

The interrupt servicing priority, from highest priority to lowest, is as follows:

a. Unimplemented instruction (select code 10).

b. Time base generator (select code 6).

c. Multiple-bit memory error (select code 5).

d. Memory protect (select code 7).

e. Slave (break) interrupt.

f. Power fail (select code 4).

g. I/O interrupts (select codes 20 through 77).

h. Virtual Memory Area (select code 12) and CDS Segment (select code 13).

6-10. CENTRAL INTERRUPT REGISTER

Each time an interrupt occurs, the address of the inocation is stored in the central interrupt register. The contents of this register are accessible at any time by executing an LIA 04 or LIB 04 instruction. This loads the address of the most recent interrupt into the A- or B-register.

6-11. PROCESSOR STATUS REGISTER

The processor status register is two registers: one for input and one for output. The input register shows the status of the switches on the data path card and is read into the upper eight bits of the A- or B-register by an LIA/B 01 instruction. The switch, bit, and function relationships are as follows:

SWITCH U0101	BIT	MEANING
S1	8	Boot select 0 = closed
S2	9	Boot select 1 = open
S3	10	Boot select
S4	11	Break enabled (0)/disabled (1)
S5	12	Not used
S6	13	ENQ/ACK handshake enabled (0)/disabled (1)
S7	—	Not used
S8	14	Auto-restart enabled (1)/ disabled (0)

The output register drives the lower eight LEDs on the sequencer card. The output of the lower eight bits of the A- or B-register are sent to the LEDs by an OTA/B 01 instruction. A logic 1 in either register lights the corresponding LED.

6-12. INTERRUPT TYPE CONTROL

I/O address 00 is the master control address for Type 3 interrupts (TBG and I/O cards). An STF 00 instruction enables Type 3 interrupts and a CLF 00 disables Type 3 interrupts. (Type 3 interrupts are disabled when power is initially applied.) I/O address 04 is the master control address for Type 2 interrupts (power fail and memory protect) and Type 3 interrupts combined. An STC 04 instruction enables Type 2 and 3 interrupts and a CLC 04 disables Type 2 and 3 interrupts.

The Type 2 and 3 interrupt mask from I/O address 04 is a different Type-3 mask than the Type-3 mask at I/O address 00. If either of these two masks are set, Type 3 interrupts will be disabled. In addition to these two interrupt masks, the Time Base Generator flag interrupt can also be masked by bit 0 of the interrupt mask register. If any of these three masks are set then the TBG flag interrupt will be disabled

6-13. INSTRUCTION SUMMARY

Table 6-3 is a summary of instructions for select codes 00 through 07. For a summary of instructions used with the I/O cards, refer to an I/O card reference manual.

Table 6-3. Instructions for Select Codes 00 through 07

INSTRUCTION	FUNCTION	INSTRUCTION	FUNCTION
STC 0	NOP	STC 4	Enable Type 2 and 3 interrupts
CLC 0	System reset	CLC 4	Disable Type 2 and 3 interrupts
STF 0	Enable Type 3 interrupts	STF 4	Flush cache
CLF 0	Disable Type 3 interrupts	CLF 4	NOP
SFS 0	Skip if Type 3 interrupts enabled	SFS 4	Skip if power is stable
SFC 0	Skip if Type 3 interrupts disabled	SFC 4	Skip if power going down
LI* 0	Load from interrupt mask register	LI* 4	Load from central interrupt register
MI* 0	NOP	MI* 4	NOP
OT* 0	Output to interrupt mask register	OT* 4	Output to central interrupt register
STC 1	NOP	STC 5	Enable multiple-bit error interrupts
CLC 1	NOP	CLC 5	Disable multiple-bit error interrupts
STF 1	Same as Set Overflow (STO)	STF 5	NOP
CLF 1	Same as Clear Overflow (CLO)	CLF 5	NOP
SFS 1	Same as Skip if Overflow Set (SOS)	SFS 5	NOP
SFC 1	Same as Skip if Overflow Clear (SOC)	SFC 5	NOP
LI* 1	Load from processor status register	LI* 5	Load from parity register (bits 0-15)
MI* 1	Merge from processor status register	LI* 5,C	Load from parity register (bits 16-31)
OT* 1	Output to processor status register	MI* 5	NOP
		OT* 5	Force check bits in X-register to address in A- or B-register.
STC 2	Enable break feature	STC 6	Turn on time base generator
CLC 2	NOP	CLC 6	Turn off time base generator
STF 2	Disable Global Register	STF 6	Set time base generator flag
CLF 2	Enable Global Register	CLF 6	Clear time base generator flag
SFS 2	Skip if Global Register disabled	SFS 6	Skip if time base generator flag set
SFC 2	Skip if Global Register enabled	SFC 6	Skip if time base generator flag clear
LI* 2	Load from Global Register	LI* 6	NOP
MI* 2	NOP	MI* 6	NOP
OT* 2	Output to Global Register (Note 1)	OT* 6	NOP
STC 3	NOP	STC 7	Enable memory protect
CLC 3	NOP	CLC 7	NOP
STF 3	NOP	STF 7	NOP
CLF 3	NOP	CLF 7	NOP
SFS 3	NOP	SFS 7	NOP
SFC 3	NOP	SFC 7	NOP
LI* 3	Load from P SAVE	LI* 7	Load from violation register
MI* 3	NOP	MI* 7	NOP
OT* 3	Output to P SAVE	OT* 7	NOP
LI* 3,C	Load from ROM P		
OT* 3,C	Output to ROM P		

* = A or B.

Note 1. An OTA/B 2 with A/B equal to one through seven establishes a diagnose mode; refer to paragraph 7-22 for details.

INPUT/OUTPUT SYSTEM

SECTION VII

The purpose of the input/output system is to transfer data between the computer and external devices. As shown in Figure 7-1, data can be transferred either by a direct memory access (DMA) feature or through the A- or B-register in the CPU (non-DMA). Each A/L-Series I/O card has DMA logic and DMA is normally used for most I/O data transfers. Once the DMA logic has been initialized, no programming is involved and the transfer occurs in three distinct steps as follows:

a. Between the external device and its I/O interface card in the computer;

b. Between the I/O card and cache memory via the backplane data bus.

c. Between cache memory and main memory via the memory frontplane. This three-step process also applies to a DMA output transfer except in reverse order.

As mentioned above, data may be transferred under program control without using the DMA feature. This type of transfer allows the computer to manipulate the data during the transfer process. A non-DMA input transfer is a three-step process as follows:

a. Between the external device and its I/O card;

b. Between the I/O card and the Aor B-register via the d .ta bus and the processor card; and

c. Between the A- or B-register and memory via the processor, cache, and memory frontplane.

Note that in the DMA transfer the A- and B-registers are bypassed. Since a DMA transfer eliminates programmed loading and storing via the accumulators, the time involved is very short. Further information on the DMA feature is given in paragraph 7-9.

7-1. INPUT/OUTPUT ADDRESSING

As shown in Figure 7-2, an external device is connected by cable directly to an interface card located in the computer mainframe. The interface card, in turn, plugs into one of the input/output slots, each of which is assigned a fixed interrupt priority. Note, however, that the select code of the A/L-Series interface cards is independent of the priority. The computer communicates with a specific device on the basis of its select code which is set by switches on the interface card.

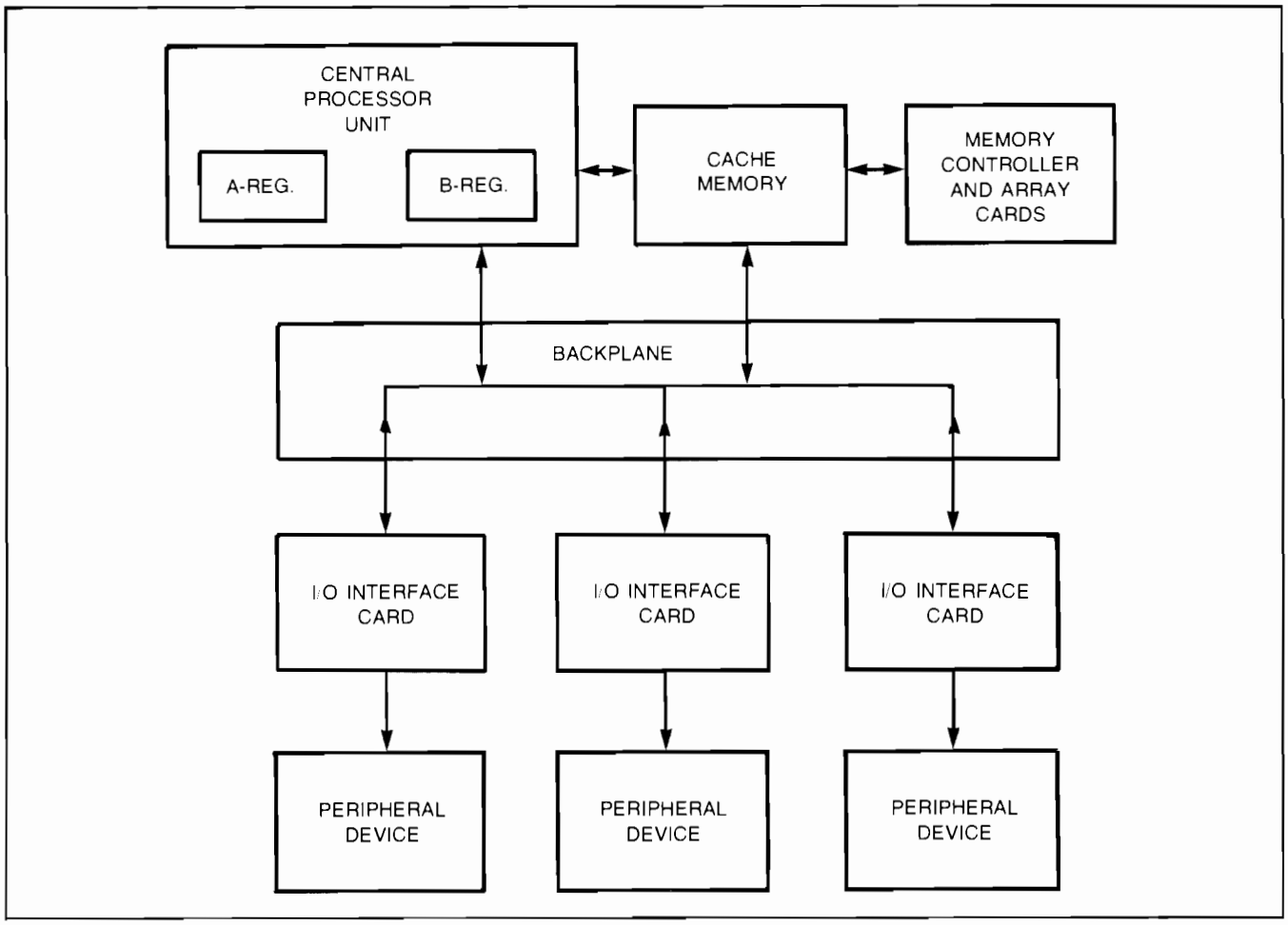
Figure 7-2 shows an interface card inserted in the I/O slot having the highest priority. If it is decided that the associated device should have lower priority, its interface card and cable may simply be exchanged with those occupying some other I/O slot. This will change the priority but not the I/O address (select code). Due to priority chaining, there can be no vacant slots from the highest priority slot to the lowest priority slot used. Only select codes 20 through 77 (octal) are available for input/output cards; the lower select codes (00 through 17) are reserved for other features.

7-2. INPUT/OUTPUT PRIORITY

The plug-in card slots of the A900 computers are numbered 1 through 16 or 1 through 20. Generally, slots 2 through 6 are used for the memory and processor cards and the remaining slots are available for I/O cards, with slot 1 having the highest I/O interrupt priority and slot 7 having the next highest I/O priority. (Note that slots 2 through 5 can only be used for the processor cards. Also, in the 16-slot computer slots 8 and 16 are reserved for the 25 kHz power module and the battery backup card, respectively.) An I/O channel consists of an I/O device (or devices) and its I/O card and is assigned the number of the card slot.

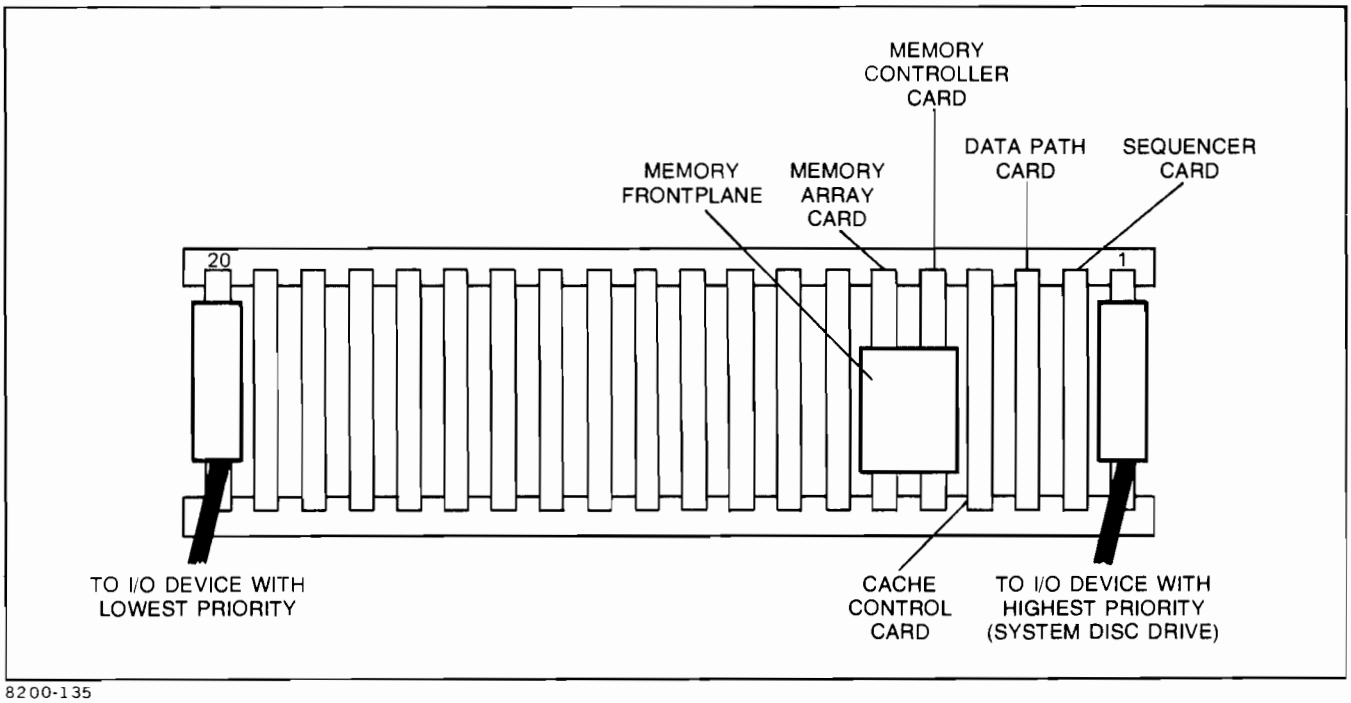
When an input/output device is ready to be serviced, it causes its interface card to request an interrupt so that the computer will interrupt the current program and service the device. Since many device interface cards will be requesting service at random times, it is necessary to establish an orderly sequence for granting interrupts. Also, it is desirable that high-speed devices should not have to wait for low-speed device transfers. Both of these requirements are met by a series-linked priority structure illustrated by Figure 7-3. The bold line, representing a priority enabling signal, is routed in series through each card capable of causing an interrupt. The card cannot interrupt unless this enabling signal is present at its input.

Each device (or other interrupt function) can break the enabling line when it requests an interrupt. If two devices simultaneously request an interrupt, the device with the highest priority will be the first one that can interrupt because it has broken the enable line for the lower-priority device. The other device cannot begin its service routine until the first device is finished. However, a still higher-priority device (one interfaced through a lower-numbered slot) may interrupt the service routine of the first device. Figure 7-4 illustrates a hypothetical case in which several devices request service by interrupting a CPU program. Both simultaneous and time-separated interrupt requests are considered.



8200-172

Figure 7-1. Input/Output System



8200-135

Figure 7-2. I/O Priority Assignments

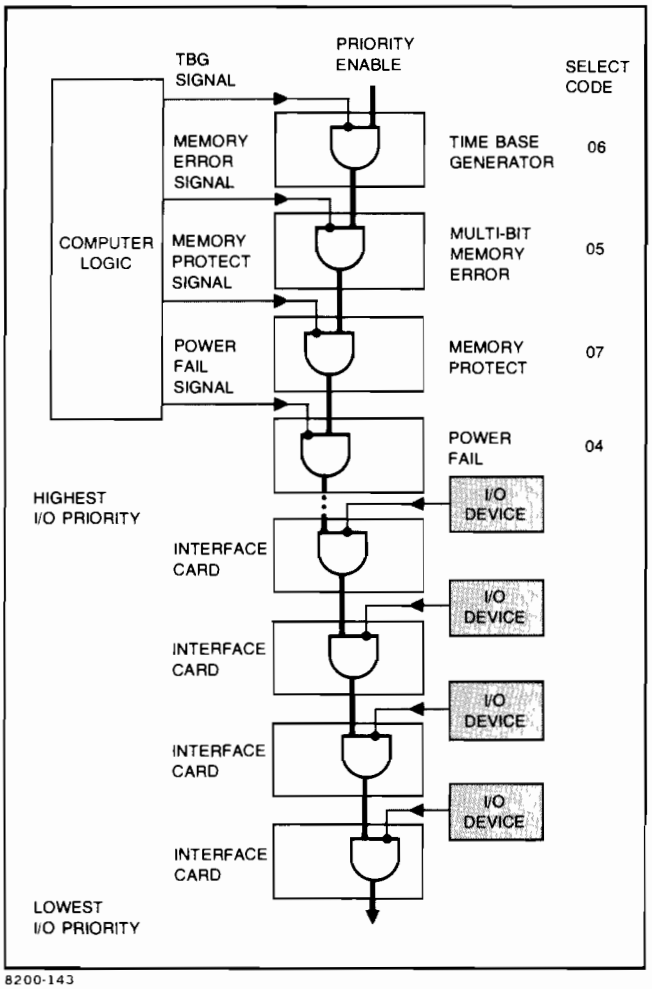
Assume that the computer is running a CPU program when an interrupt from I/O channel 8 occurs (at reference time t1), and that the card in slot 8 is assigned select code 22. When the interrupt is serviced, the I/O card supplies its select code (in this case 22) to the CPU. The CPU uses the select code as an address to initiate the interrupt service routine. A JSB instruction in the interrupt location for select code 22 causes a program jump to the service routine for the channel-8 device (select code 22). The JSB instruction automatically saves the return address (in a location which the programmer must reserve in his routine) for a later return to the CPU program.

The routine for channel 8 (select code 22) is still in progress when several other devices request service (set flag). First, channels 9 and 10 request simultaneously at time t2; however, since neither one has priority over channel 8, their flags are ignored and channel 8 continues transfer. But at t3, a higher priority device on channel 1 requests service. This request interrupts the channel 8 routine and causes the channel 1 routine to begin. The JSB instruction saves the return address for return to the channel 8 routine.

During the channel 1 routine, the channel 7 flag is set (t4). Since it has lower priority than channel 1, channel 7 must wait until the end of the channel 1 routine. And since the channel 1 routine, when it ends, contains a return address to the channel 8 routine, program control temporarily returns to channel 8 (even though the waiting channel 7 has higher priority). The JMP,I instruction used for the return inhibits all interrupts until fully executed. At the end of this short interval, the channel 7 interrupt request is granted.

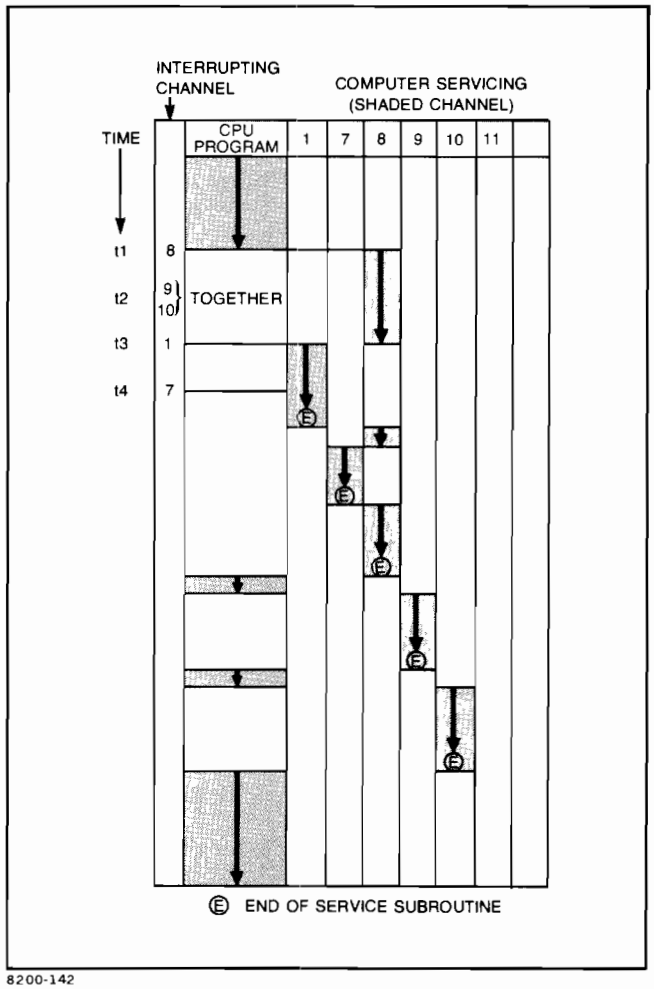
When channel 7 has finished its routine, control is returned to channel 8, which at last has sufficient priority to complete its routine. Since channel 8 has been saving a return address in the main CPU program, it returns control to this point.

The two waiting interrupt requests from channels 9 and 10 are now enabled. Channel 9 has the higher priority and goes first. At the end of the channel 9 routine control is temporarily returned to the CPU program. Then the lowest priority channel (channel 10) interrupts and completes its routine. Finally, control is returned to the CPU program, which resumes processing.



8200-143

Figure 7-3. Priority Linkage (Simplified)



8200-142

Figure 7-4. Interrupt Sequence

7-3. INTERFACE ELEMENTS

The interface card provides the communication link between the computer and one or more external devices. The interface card includes several basic elements which either the computer or the device can control in order to effect the necessary communication. These basic elements are the Global Register, control bits, flag bits, data buffer register, and control register. Other registers, associated only with DMA, are discussed in paragraph 7-9. The control and flag bits and the data buffer and control registers of an interface card can be addressed directly when the card's select code is in the Global Register (GR) and the GR is enabled. Refer to the interface card reference manuals for specific information on the data and control registers.

7-4. GLOBAL REGISTER

In the A-Series computers, the select code that is in the Global Register specifies which I/O card is enabled to execute I/O instructions. The Global Register (GR) is a register on each I/O card that can be loaded with the select code of any one of the I/O cards. (At any given time, the GR on all I/O cards is loaded with the same select code.) When the GR is enabled, an I/O instruction is executed only by the I/O card whose select code matches the select code in its GR. Also, the GR allows other registers on the selected I/O card to be accessed programmatically by I/O instructions. The Global Register on all I/O cards may be simultaneously loaded with an OTA/B 02 instruction, enabled with a CLF 02 instruction, and disabled with an STF 02 instruction.

7-5. CONTROL BITS

The control bits on an interface card are used to turn on a specific I/O function. In addition, a control bit must be set to allow the corresponding flag bit to interrupt. There are three control bits associated with each I/O select code: control 20, 21, and 30. Control 30 is the only control bit that can be accessed with or without the Global Register being enabled. When control 30 is set it generates an action command, allowing one word or character to be read or written. Control 20 and 21 can only be accessed when the Global Register is enabled. When control 20 is set it turns on DMA self-configuration. The setting of control 21 enables DMA transfers.

7-6. FLAG BITS

The flag bits (when set) are used primarily to interrupt or to signal completion of a task. Flag 30, the only flag bit accessible without using the Global Register, signals that either one data element has been transferred or that an interrupting condition has been detected. There are three other flags, all of which must be accessed with the Global Register enabled. Flag 20 signals DMA self-configuring transfer complete; flag 21 signals DMA transfer complete; and flag 22 signals a multiple-bit data error during DMA. The device cannot clear the flag bit. If the corresponding control bit is set, priority is high, and the interrupt system is enabled, then setting the flag bit will cause an interrupt to the location corresponding to the I/O card select code.

7-7. DATA BUFFER REGISTER

The data buffer register (designated Register 30) is used for the intermediate storage of data during an I/O transfer. Typically, the data capacity is 16 bits.

7-8. CONTROL REGISTER

The control register (designated Register 31) enables a general purpose interface card to be configured for compatibility with a specific I/O device or to be programmed for particular modes of operation. The control register must be programmatically set up for each particular application. Refer to the interface card manuals for specific information on the control register.

7-9. DIRECT MEMORY ACCESS

The direct memory access (DMA) capability of each A/L-Series interface card provides a direct data path between memory and a peripheral device, making it practical to use DMA for most data transfers. The use of DMA to perform I/O data transfers reduces the number of interrupts from one per byte or word to one per complete DMA block transfer. (Maximum DMA block size is 65,536 bytes.)

The maximum DMA transfer rate (inbound) is 3.7 megabytes per second; this is also the combined limit for DMA transfers by two or more I/O cards. DMA can interleave with CPU operations. Even at full DMA rate, the CPU should still be running at three-eighths full speed. The DMA feature is provided by the following elements:

a. The common cache memory that links the processor, main memory, and I/O cards;

b. The capability of the I/O cards to execute I/O instructions; and

c. The Global Register which:

   1. Enables only the I/O card whose select code is in the Global Register to execute I/O instructions, freeing the address bits of the I/O instruction; and

   2. Enables the I/O-instruction address bits to be used to access registers on the I/O card specified by the Global Register.

Each I/O card has four registers associated with DMA. Three of them must be loaded with control words that specify the DMA operation. The fourth register is used for a special type of DMA operation called self-configured

DMA which is discussed later. All of these registers can be accessed only when the select code of the desired I/O card is in the Global Register. The DMA registers and their functions are as follows:

a. Register 20, DMA Self-Configuration Address Register;

b. Register 21 (for Control Word 1), DMA Control Register;

c. Register 22 (for Control Word 2), DMA Address Register; and

d. Register 23 (for Control Word 3), Word/Byte Count Register.

7-10. CONTROL WORD 1

Control Word 1 (CW1) must be loaded into Register 21 of the desired I/O card as part of the DMA initialization process. The general definitions of the bits in Control Word 1 are given in Figure 7-5. Note that the requirements of individual I/O cards may vary slightly from the general definitions and that it is necessary to refer to the I/O card reference manuals for specific programming information.

7-11. CONTROL WORD 2

Control Word 2 (CW2) loads into Register 22 the address of the first memory location to be read from or stored into when the DMA operation is initiated. The most significant bit, bit 15, is not used by the DMA control logic; when CW2 is read for status, bit 15 is the complement of bit 7 in CW1 (Figure 7-5).

7-12. CONTROL WORD 3

Control Word 3 (CW3) loads into Register 23 the twos-complement of the number of data elements to be transferred by DMA. Data elements may be either words or bytes as specified by bit 13 of CW1 (Figure 7-5). The end of a DMA data transfer is indicated by the transition from -1 to 0 of the value in Register 23 (the Word/Byte Count Register); this causes the I/O card to generate a completion interrupt. (A DMA transfer can also be terminated in other ways as described in the interface card manuals.)

7-13. DMA TRANSFER INITIALIZATION

A DMA data transfer is started by:

a. Loading the Global Register with the select code of the desired I/O card;

b. Loading the three DMA registers: DMA control into Register 21, DMA address into Register 22, and word/byte count into Register 23;

c. Loading the control register (Register 31) of the I/O card (described in the individual interface card reference manuals); and

d. Issuing an STC instruction to Register 21 (DMA Control Register).

A typical programming sequence to configure the DMA logic for a DMA transfer is as follows:

```
LDA SC      Load select code
OTA 2,C     Set up Global Register
CLC 21B
LDA CW1
OTA 21B     Output DMA control word
LDA CW2
OTA 22B     Output DMA starting address
LDA CW3
OTA 23B     Output DMA word/byte count
LDA CNTL
OTA 31B     Output I/O card control word
STC 21B,C   Start DMA and device
<continue any other processing>
```

7-14. SELF-CONFIGURED DMA

Each I/O card also has logic that can automatically load the DMA registers discussed previously with the DMA control words from sequential locations in memory. This process is performed by using the I/O card's Register 20, the Self-Configuration Register. The DMA self-configuration feature is initialized by setting the value of Register 20 to the memory address of a list of DMA "triplets" or "quadruplets".

A triplet is of the form: DMA control word, DMA transfer address, and word/byte count. The triplet words are the words to be loaded into Registers 21, 22, and 23, respectively. A quadruplet is of the form: DMA control word, I/O-card control word, transfer address, and word/byte count. Bit 8 of the DMA control word (Control Word 1) determines whether a triplet or quadruplet is loaded. (A quadruplet is used only when the I/O-card control word must be changed; refer to the interface card manuals for detailed information.) As each register is loaded, the contents of Register 20 are incremented, leaving it pointing to the memory location to be loaded into the next register.

DMA self-configuration can be chained to enable consecutive DMA transfers via the same I/O card with a minimum of interrupts. If bit 15 of Control Word 1 in a triplet (or quadruplet) is a logic 1, the DMA registers will be loaded with the next triplet or quadruplet in memory (as pointed to by Register 20) upon completion of the current DMA block transfer. When bit 15 (and bit 11) is a logic 0, the current DMA block transfer is followed by a completion interrupt.

15	14	13	12	11	10	9	8	7	6 5	4 0
CONT	DVCMD	BYTE	RES	CINT	REM	FOUR	AUTO	IN	Various	ADDR EXT BUS

CONT (Continue), bit 15.

Bit 15 = 1: Enable a DMA re-configuration upon completion of a self-configured DMA transfer.

Bit 15 = 0: Stop DMA after current transfer.

DVCMD (Device Command), bit 14.

Bit 14 = 1: Issue a Device Command signal for each data element transferred.

Bit 14 = 0: No Device Command signal issued.

BYTE (Byte/word transfer), bit 13.

Bit 13 = 1: Conduct DMA transfer in byte mode.

Bit 13 = 0: Conduct DMA transfer in word mode.

RES (Residue), bit 12.

Bit 12 = 1: Write word/byte count back into memory.

Bit 12 = 0: Word/byte count is not written.

CINT (Completion Interrupt), bit 11.

Bit 11 = 1: Inhibit DMA completion interrupt.

Bit 11 = 0: Request completion interrupt when word/byte count goes from −1 to 0 and bit 15 equals 0.

REM (Remote), bit 10.

Bit 10 = 1: Enable remote (non-standard) memory for DMA transfer.

Bit 10 = 0: Remote memory not enabled.

FOUR (Fetch four control words), bit 9.

Bit 9 = 1: Causes DMA self-configuration to fetch four control words; i.e., three DMA control words and one I/O card control word.

Bit 9 = 0: Fetch three control words for DMA self-configuration.

AUTO (Automatic), bit 8. This bit is read only during self-configured DMA.

Bit 8 = 1: Initiate first data transfer once DMA is configured to output, without waiting for an SRQ. For input transfers, enable a Device Command signal after the last data element is transferred.

Bit 8 = 0: For output transfers, wait for a Service Request (SRQ) signal before performing the first transfer. For input transfers, the last data element is not followed by a Device Command.

IN (Transfer In), bit 7.

Bit 7 = 1: Perform DMA transfer from I/O device to memory.

Bit 7 = 0: Perform DMA transfer from memory to I/O device.

Various, bits 5 and 6, User definable.

ADDR EXT BUS, bits 4-0

These five bits allow DMA accesses to physical memory by referencing one map set of 32 registers each.

8200-53

Figure 7-5. General Bit Definitions for Control Word 1

7-15. DMA DATA TRANSFER

Figure 7-6 illustrates, in general, the sequence of operations for a DMA input data transfer (the minor differences for an output transfer are explained in text). Note that the Global Register has been enabled and loaded with the I/O card select code.

The initialization routine sets up the DMA control registers on the I/O card (1) and issues the start command (STC 21,C) to the DMA Control Bit (Control 21). (If the operation is an output, the I/O card buffer is also loaded at this time.) The DMA logic is now turned on and the computer program continues with other instructions.

Setting the DMA Control bit (2) causes the I/O card to send a Start signal (with a data word if it is an output transfer) to the external device (3). The device goes through a read or write cycle and returns a Done signal (with a data word if it is an input transfer). The Done signal (4) requests the DMA logic (5) to transfer a word into (or out of) cache memory (6). The process now loops back to step 3 to transfer the next word.

NOTE

Whenever a requested address is not already in cache memory, either for data input or output, it is transferred from main memory to cache.

After the specified number of words has been transferred, the DMA logic generates a completion interrupt (7). The program control is now forced to a completion routine (8), the content of which is the programmer's responsibility.

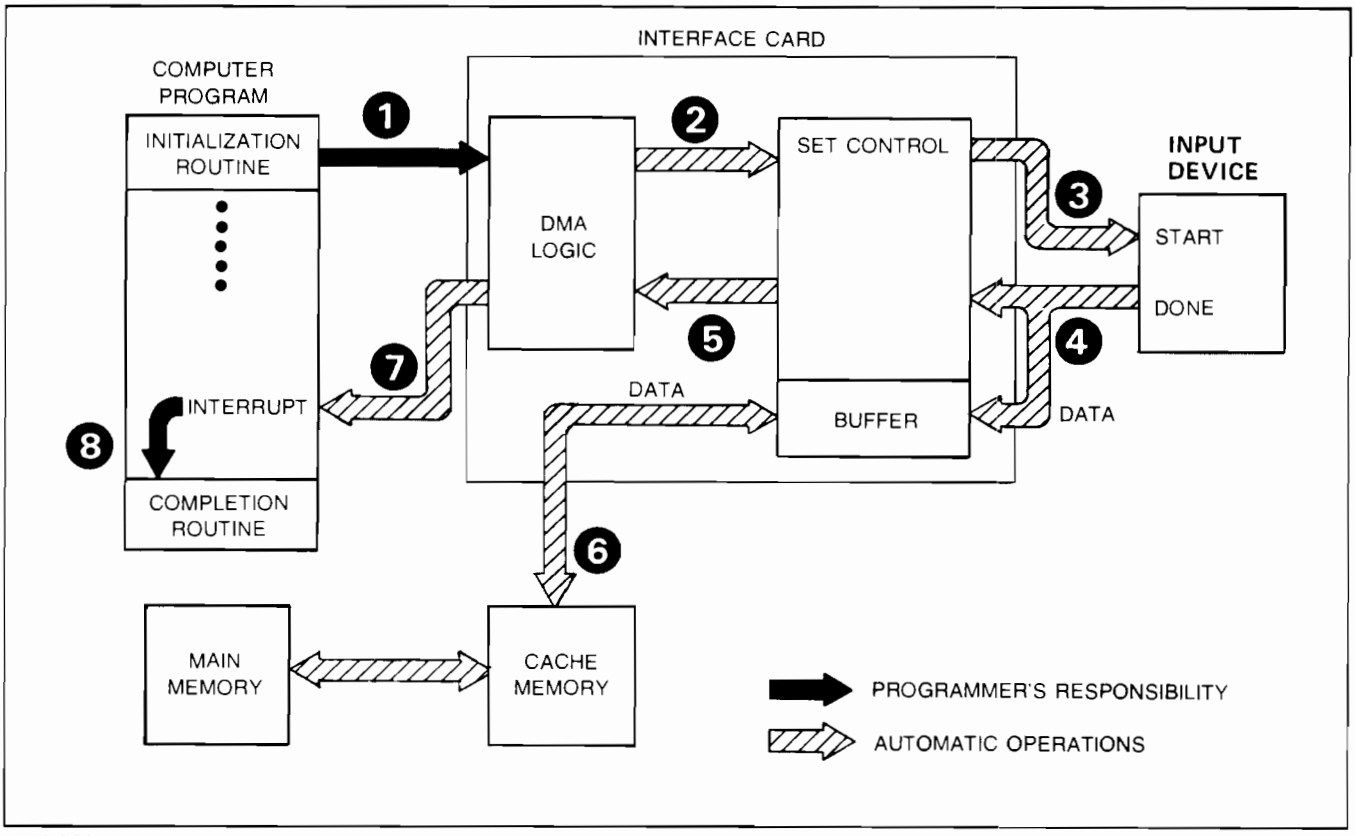
For more detailed information on DMA, refer to the I/O interfacing guide, part no. 02103-90005.

7-16. NON-DMA DATA TRANSFER

The following paragraphs describe how data is transferred between memory and input/output devices without using DMA. The sequences presented are simplified in order to present an overall view without the involvement of software operating systems or device drivers.

7-17. INPUT DATA TRANSFER (INTERRUPT METHOD)

Figure 7-7 illustrates the sequence of events required to input data using the interrupt method. Note that some operations are under control of the computer program (programmer responsibility) and some of the operations are automatic. Note also that the Global Register has been loaded and enabled and the I/O card control register has been loaded.



8200-144

Figure 7-6. DMA Input Data Transfer

The operations begin (1) with the programmed instruction STC 30,C which sets the Control bit (Control 30) and clears the Flag bit (Flag 30) on the I/O card. Since the next few operations are under control of the hardware, the computer program may continue the execution of other instructions. Setting the Control bit causes the card to output a Start signal (2) to the device, which reads out a data character and asserts the Done signal (3).

The device Done signal sets the Flag bit, which in turn generates an interrupt (4) provided that the interrupt conditions are met; i.e., the interrupt system must be on (STF 00 previously given), no higher priority interrupt is pending, and the Control bit is set (done in step 1).

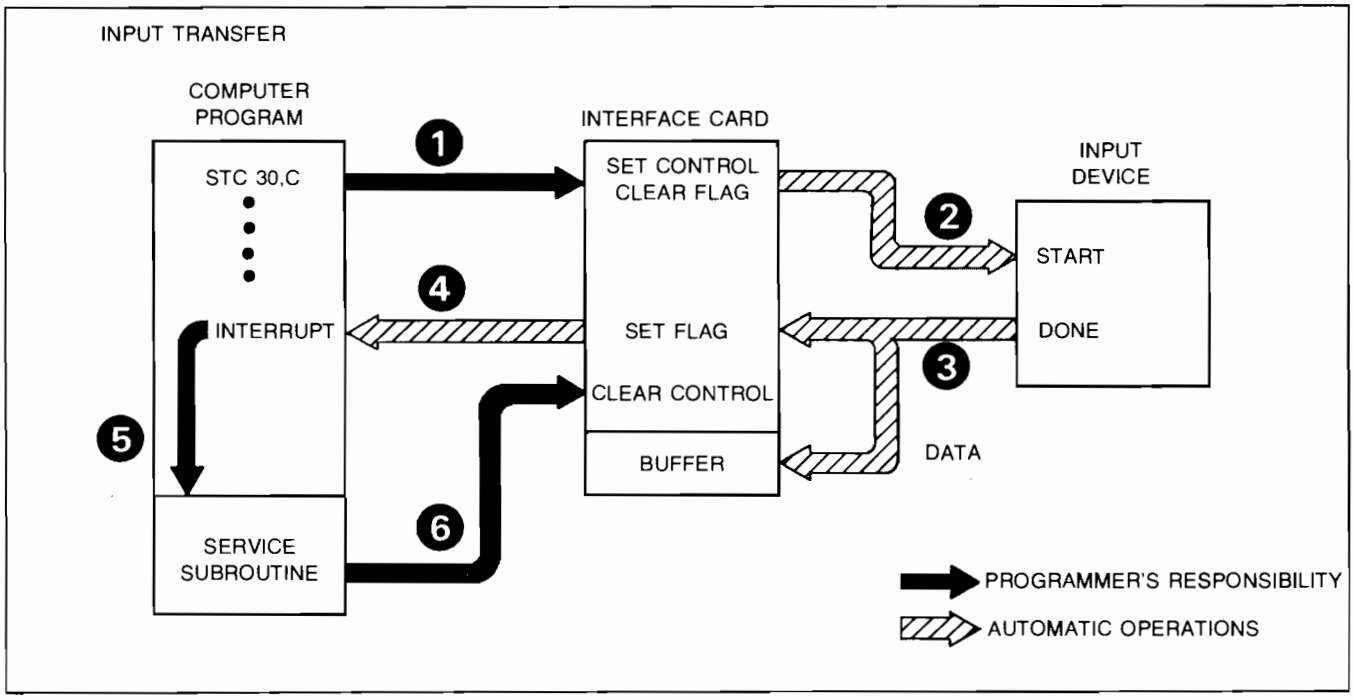
The interrupt causes the current computer program to be suspended and control is transferred to a service subroutine (5). It is the programmer's responsibility to provide the linkage between the interrupt location (which agrees with the select code) and the service subroutine. It is also the programmer's responsibility to include in his service subroutine the instructions for processing the data (loading into an accumulator, manipulating if necessary, and storing into memory).

The subroutine may then issue further STC 30,C instructions to transfer additional data characters. One of the final instructions in the service subroutine must be CLC 30,C. This step (6) restores the interrupt capability to lower priority devices and returns the I/O card to its static "reset" condition (Control clear and Flag clear). This condition is initially established by the computer at power turn-on and it is the programmer's responsibility to return the I/O card to the same condition on the completion of each data transfer operation. At the end of the subroutine, control is returned to the interrupted program via previously established linkages.

7-18. OUTPUT DATA TRANSFER (INTERRUPT METHOD)

Figure 7-8 illustrates the sequence of events required to output data using the interrupt method. Again note the distinction between programmed and automatic operations. Note also that the Global Register has been loaded and enabled and that the I/O card's control register has been loaded. It is assumed that the data to be transferred has been loaded into the A-register and is in a form suitable for output.

The output operation begins with a programmed instruction (OTA 30) to transfer the contents of the A-register to the I/O card buffer (1). This is followed (2) by the instruction STC 30,C which sets the Control bit (Control 30) and clears the Flag bit (Flag 30) on the I/O card. Since the next few operations are under control of the hardware, the computer program may continue the execution of other instructions. Setting the Control bit causes the card to output the buffered data and a Start signal (3) to the device, which writes (e.g., records, stores, etc.) the data character and asserts the Done signal (4).



8200-41

Figure 7-7. Input Data Transfer (Interrupt Method)

The device Done signal sets the card's Flag bit, which in turn generates an interrupt (5) provided that the interrupt system is on, priority is high, and the Control bit is set (done in step 2). The interrupt causes the current computer program to be suspended and control is transferred to a service subroutine (6). It is the programmer's responsibility to provide the linkage between the interrupt location (which agrees with the select code) and the service subroutine. The detailed contents of the subroutine are also the programmer's responsibility and the contents will vary with the type of device.

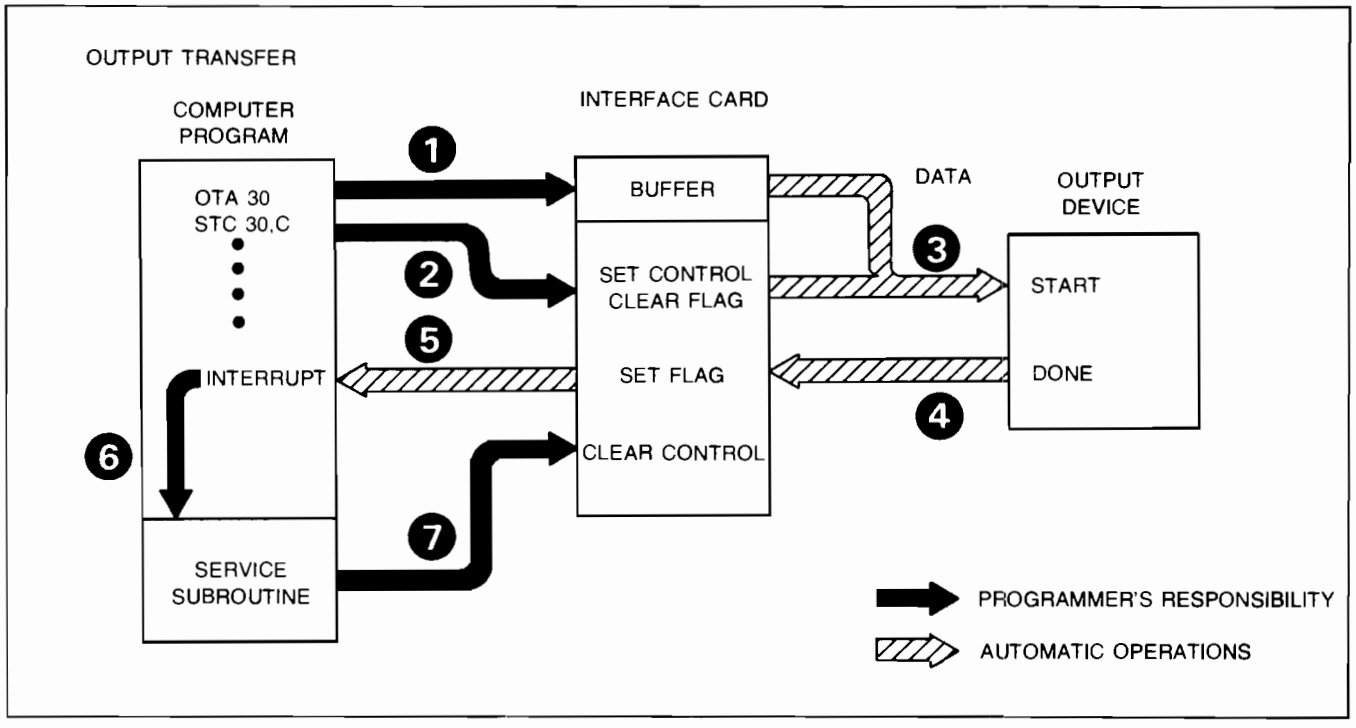
The subroutine may then output further data to the I/O card and reissue the STC 30,C command for additional data character transfers. One of the final instructions in the service subroutine must be a clear control (CLC 30,C). This step (7) allows lower priority devices to interrupt and restores the I/O card to its static "reset" condition (Control clear and Flag clear). At the end of the subroutine, control is returned to the interrupted program via the previously established linkages.

7-19. NON-INTERRUPT DATA TRANSFER

It is also possible to transfer data without using the interrupt system. This involves a "wait-for-flag" method in which the computer commands the device to operate and then waits for the completion response. In using this method to transfer data, computer time is relatively unimportant. It is assumed that the interrupt system is turned off (STF 00 not previously given). It is also assumed that the Global Register has been loaded and enabled and that the I/O card's control register has been loaded. As shown in Table 7-1, the programming is very simple; each of the routines will transfer one word or character of data.

Table 7-1. Noninterrupt Transfer Routines

INSTRUCTIONS	COMMENTS
INPUT ROUTINE	
STC 30,C	Start device
SFS 30	Is input ready?
JMP *-1	No, repeat previous instruction
LIA 30	Yes, load input into A-register
OUTPUT ROUTINE	
OTB 30	Output data to I/O card's data register
STC 30,C	Start device
SFS 30	Has device accepted the data?
JMP *-1	No, repeat previous instruction
NOP	Yes, proceed



8200-42

Figure 7-8. Output Data Transfer (Interrupt Method)

7-20. INPUT. As described in paragraph 7-17, an STC 30,C instruction begins the operation by commanding the device to read one word or character. The computer then goes into a waiting loop, repeatedly checking the status of the Flag bit (Flag 30). If the Flag bit is not set, the JMP *−1 instruction causes a jump back to the SFS instruction. (The *−1 operand is assembler notation for "this location minus one.") When the Flag bit is set, the skip condition for SFS is met and the JMP instruction is skipped. The computer thus exits from the waiting loop and the LIA 30 instruction loads the device input data into the A-register.

7-21. OUTPUT. The first step, which transfers the data to the I/O card buffer, is the OTA 30 instruction. Then STC 30,C commands the device to operate and accept the data. The computer then goes into a waiting loop as described in the preceding paragraph. When the Flag bit becomes set, indicating that the device has accepted the output data, the computer exits from the loop. (The final NOP is for illustration purposes only.)

7-22. DIAGNOSE MODES

A diagnose mode allows the I/O cards to be accessed for diagnostic or test purposes. A diagnose mode is established when an OTA/B 2 instruction (output to the Global Register) is executed with the A- or B-register value equal to one through seven. (The diagnose mode is terminated when an OTA/B 2 instruction is executed with the A- or B-register equal to zero.) When establishing a diagnose mode the current contents of the Global Register (GR) is not altered. The diagnose mode can be on an individual I/O card or on all I/O cards. If the GR is disabled then all I/O cards accept the diagnose mode. If the GR is enabled, only the I/O card whose select code is in the GR will accept the diagnose mode. Diagnose Mode 7 is used to disable any service request (SRQ) signal coming into the I/O chip which may cause DMA to cycle during a test. (Mode 7 can be disabled only by a CRS signal (CLC 0).) Diagnose Modes 4 through 6 are reserved for future definition. Diagnose Modes 1 through 3 are described in the following paragraphs.

7-23. DIAGNOSE MODE 1

When an OTA/B 2 instruction is executed with the A- or B-register equal to one each I/O card responds by turning off priority to the next I/O card. When the instruction is complete the only I/O card receiving priority will be the highest priority I/O card (i.e., the one directly next to the processor card. When a subsequent LIA/B 2 instruction is executed, the I/O card receiving priority sets the A- or B-register equal to its select code and identification data (ID) and passes priority to the next I/O card. Having responded once it will not respond again unless Mode 1 is established again. The next LIA/B 2 executed sets the A- or B-register equal to the second I/O card's select code and ID. The second I/O card at completion of the instruction passes priority to the next I/O card. This process continues until the last I/O card responds. After the last I/O card responds the next LIA/B 2 will not affect the A- or B-register and therefore can be detected as a no response. (An OTA/B 2 with the A- or B-register equal to 0 terminates this sequence.)

Mode 1 can also be used to retrieve the select code and ID of a desired I/O card without going through the priority process. This is accomplished by establishing Mode 1 and then executing an LIA/B xx, where xx is the I/O card select code. This procedure will not modify a priority sequence already in process. The Mode 1 select code and ID format is shown in Table 7-2.

Table 7-2. Diagnose Mode 1

A/B BITS	MEANING
15	Intelligent interface
14 13 12 11 10 9	Interface card type identification number
8 7 6	Interface card revision code
5 0	Interface card select code

7-24. DIAGNOSE MODE 2

Diagnose Mode 2 causes an I/O card to respond to an LIA/B 2 instruction in the same manner as in Mode 1 except that the data set into the A- or B-register is as shown in Table 7-3.

7-25. DIAGNOSE MODE 3

Diagnose Mode 3 allows an I/O chip to do a DMA transfer without affecting the I/O card. When Mode 3 is entered the I/O chip does a DMA input transfer of the data in the configuration address register to the location in memory pointed to by the DMA address register. The configuration address register is incremented after each transfer so that the data can be verified. The transfer continues until the DMA count is incremented to zero. Mode 3 also prevents any STC instructions from generating a device command to the I/O card.

Table 7-3. Diagnose Mode 2

A/B BITS	MEANING
15, 14, 13	Always zero
12	1 = Break feature is enabled
11	1 = Receiving interrupt priority
10	Always zero
9	Control bit
8	Flag bit
7	1 = Global register equals select code of interface card
6	Global register enabled/disabled
5, 4, 3, 2, 1, 0	Current global register value

MICROPROGRAMMING

SECTION VIII

This section contains an introductory discussion of the A900 computer microprogramming techniques and development. For additional information, refer to the *HP 92049A Microprogramming Package Reference Manual,* part no. 92049-90001.

8-1. THE MICROPROGRAMMED COMPUTER

The control section of a computer is the portion of the computer that directs and controls the other sections; i.e., the memory section, input-output section, and the arithmetic-logic section. In totally hardwired computers, the control section logic is normally "spread out" physically throughout the computer. This design approach makes it impossible to enhance the computer's instruction set without redesign. In contrast, A900 computers have a fully microprogrammed control section, which means that the sequence in which the control functions are performed are made programmable through the use of a technique called microprogramming.

The action taken when any one of the A900 base set of 292 assembly language instructions is executed is determined by a microprogram associated with the assembly language instruction (these microprograms reside in a special memory called control store); the control section oversees the translation and controls the execution of the microprogram. With this design approach, instruction set enhancements can be made by changing or adding to the set of microprograms that control the machine's execution. Many computers are microprogrammed; however, Hewlett-Packard has taken the concept one step further to offer the power of microprogramming to the user.

8-2. THE MICROPROGRAMMABLE COMPUTER

A900 computer users can more fully take advantage of the computer's power by utilizing microprogramming. The microprogrammer has more instructions, a more flexible word format, more registers, and faster execution times to work with than does the assembly language programmer. The microinstruction word length is 48 bits which enables concurrent operations to be performed in a single instruction. Microprogrammers can access 10 scratch pad registers in addition to those available to the assembly language programmer and have up to 32,768 48-bit words of memory (termed control store) in which to store microprograms. Up to 14 levels of nested subroutines are possible in A900 computers. The microprogrammer works in a much faster environment than does the assembly language programmer for two reasons. One, since microinstructions have access to most of the internal parts of the computer's architecture, fewer memory fetches are required to accomplish most tasks. Two, the microinstruction execution time of 250 nanoseconds is much faster than the typical assembly instruction execution time of 1 to 2 microseconds.

These capabilities are easily taken advantage of by A900 computer users through the extensive support provided by Hewlett-Packard. Some of the more important benefits of Hewlett-Packard's microprogramming are given in the following paragraphs.

8-3. CUSTOMIZED INSTRUCTIONS

Through the use of microprogramming, the computer's assembly language instruction set can be expanded with instructions tailored for specific applications. By adding special purpose instruction sets, the general purpose computer can be uniquely adapted for a certain job and thus become very efficient at that job. Applications that may be profitably microcoded include arithmetic calculations. I/O device driver programs, and sorts and table searches.

Microprogramming is very similar to assembly language programming, although it is more powerful in many ways. Some knowledge of the internal structure of the computer is required, but once this knowledge is attained, the increased power and flexibility of microprogramming can ease the solution of many programming tasks. Microprograms are easily callable by assembly or higher level language programs.

8-4. SYSTEM SPEED

Microprogramming often-used routines will typically decrease program execution time by factors of two to ten and sometimes by as much as twenty or more. Software routines can be made to execute at the hardware speeds of the microprogram environment and the additional registers available to the microprogrammer can serve to eliminate many time-consuming memory fetches.

8-5. MEMORY SPACE AND SECURITY

By converting software routines into microprograms, space in main memory that would normally be required for time-critical routines can be freed for other uses. The routines remain instantly callable, as opposed to routines

stored in a peripheral device. Microprograms are also less accessible than conventional software which affords a higher degree of security to microcoded routines.

8-6. DEVELOPING MICROPROGRAMS

Developing microprograms is similar to developing Pascal language programs and is done with the aid of the HP paraphraser (MPARA). Since the user will not normally want to microcode all of a certain program, some analysis is required to determine which segments of the assembly language program can be most profitably converted to microcode. By substituting this section of code with a microprogrammed subroutine that is callable by the assembly or higher level program, overall execution time is reduced.

Once the microprogrammer has determined what segment to implement in microcode, the microprogram is developed as shown in Figure 8-1. The paraphraser program (in main memory) is used to assemble the source microprogram into an object program. Then, the object microprogram is loaded into writable control store (WCS).

When the microprogram is fully checked out, that user may choose to have his program reside permanently in programmable read-only memory (PROM) or in WCS where it may be altered programmatically. Implementation in ROM is accomplished by programming the PROMs with a PROM writer and installing the programmed PROMs in the computer. The mask tapes shown in Figure 8-1 are required by the PROM writer and are generated by the software at the user's command. ROM-resident microprograms are permanent and do not have to be reloaded each time the computer is powered up; this implementation also prevents users from erroneously destroying the microprogram. The user who does not require such permanence for microprogram storage may execute his microcode from WCS. Microprograms used in this manner may be loaded with the WCS I/O utility routine and may be altered under program control to suit a variety of users. User-written microprograms are easily accessed by assembly or higher level programs. Once the microprogram is developed and loaded into control store, it may be called in a very similar manner to a software subroutine.

8-7. SUPPORT FOR THE MICROPROGRAMMER

Hewlett-Packard provides a comprehensive set of hardware manuals, software manuals, and training courses to make user microprogramming easy to learn and implement. For permanent implementation of microprograms, PROMs may be installed in the HP 12205A Control Store Card. Up to 2,048 48-bit words of control store in the form of 2K PROMs may be installed in the optional Control Store Card which occupies a slot in the card cage of the computer mainframe.

The 4K writable control store (WCS) of the HP 12205A Card provides read-write control store which can be used for the development and execution of user-supplied microprograms. Microprograms in WCS are executed at the same speed as those in the read-only control store. The WCS contains 4,096 48-bit locations of random-access-memory (RAM), including all necessary address and read/write circuits. WCS can be written into or read under computer control using standard input/output instructions. An I/O utility program makes it possible for FORTRAN and Pascal programs to write into or read from a WCS module using a conventional program call. The WCS is read at full speed by way of a frontplane connecting it to the control section of the processor.

Available microprogramming software includes the paraphraser as well as a diagnostic, driver program, and I/O utility program for use with the writable control store module. These software aids operate under the Hewlett-Packard Real Time Executive (RTE) operating systems.

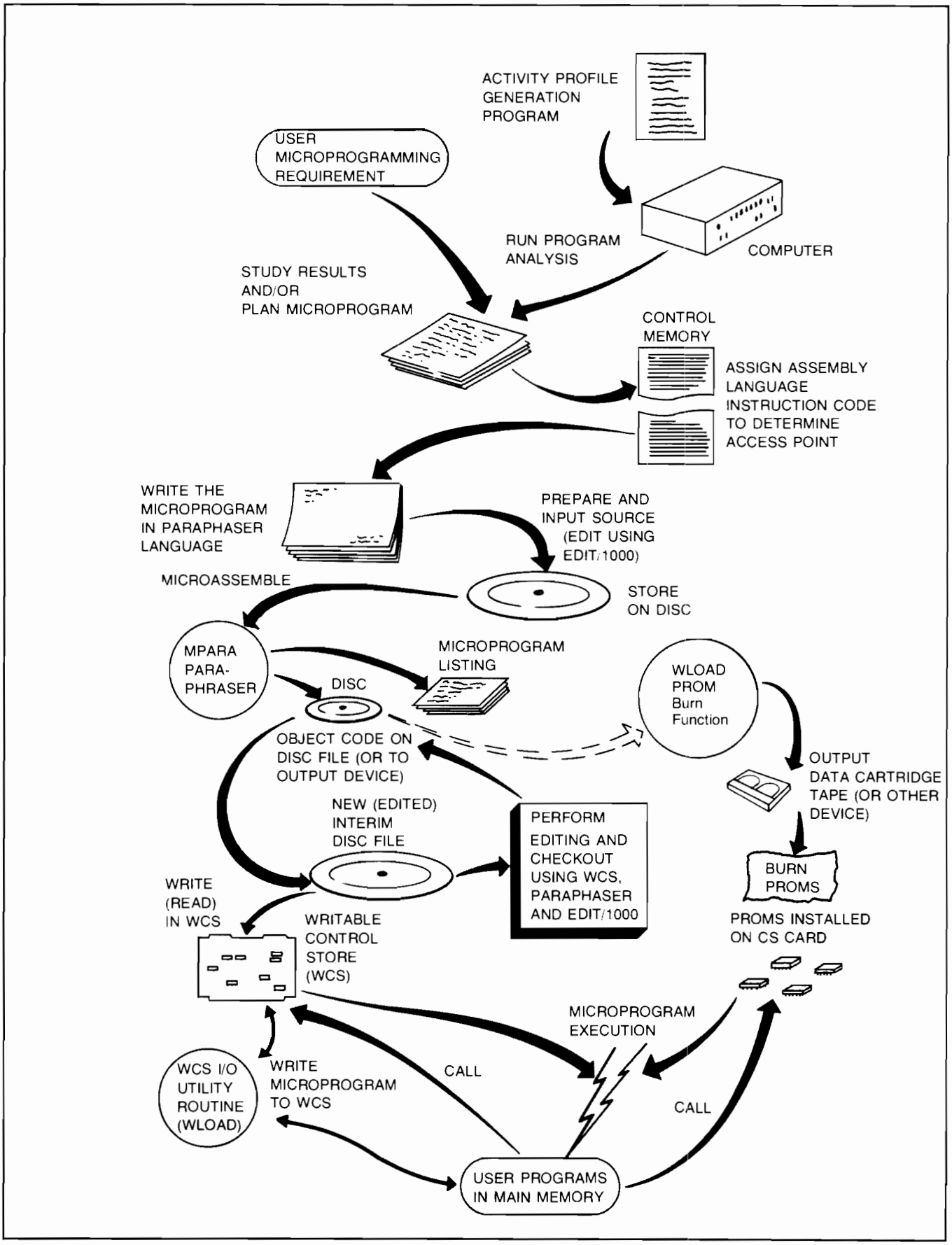
A course is offered at HP facilities in Cupertino, California for customer training. Requiring a knowledge of HP 1000 assembly language as a prerequisite, the course features in-depth coverage of microprogram development and implementation, and provides hands-on experience for the microprogrammer. The A900 microprogrammer may also take advantage of other user-written microprograms via the HP Contributed Library.

8-8 FPP MICROPROGRAMMING

Information on directly microprogramming the Floating Point Processor (FPP) is given in the *HP 92049A Microprogramming Package Reference Manual,* part no. 92049-90001.

8-9. CONCLUSION

Microprogramming is a very powerful tool that gives the user many advantages in terms of speed, flexibility, and program security. Microprogramming does have its limitations however, and the potential user should examine very closely the extent of support provided by the computer manufacturer. Hewlett-Packard has by far sold and supported the greatest number of microprogrammable computers in the world, and provides world-wide customer support. Customer training courses and documentation have been refined from years of customer-contributed feedback and actual implementation is made easy through extensive software support packages and inexpensive hardware tools.



8200-8A

Figure 8-1. Microprogramming Implementation Process

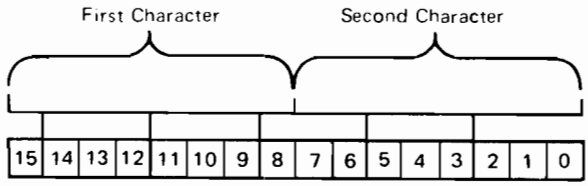
APPENDIX

A

CHARACTER CODES

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000110
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
a	060400	000141
b	061000	000142
c	061400	000143
d	062000	000144
e	062400	000145
f	063000	000146
g	063400	000147
h	064000	000150
i	064400	000151
j	065000	000152
k	065400	000153
l	066000	000154
m	066400	000155
n	067000	000156
o	067400	000157
p	070000	000160
q	070400	000161
r	071000	000162
s	071400	000163
t	072000	000164
u	072400	000165
v	073000	000166
w	073400	000167
x	074000	000170
y	074400	000171
z	075000	000172
0	030000	000060
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
NUL	000000	000000
SOH	000400	000001
STX	001000	000002
ETX	001400	000003
EOT	002000	000004
ENQ	002400	000005

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
ACK	003000	000006
BEL	003400	000007
BS	004000	000010
HT	004400	000011
LF	005000	000012
VT	005400	000013
FF	006000	000014
CR	006400	000015
SO	007000	000016
SI	007400	000017
DLE	010000	000020
DC1	010400	000021
DC2	011000	000022
DC3	011400	000023
DC4	012000	000024
NAK	012400	000025
SYN	013000	000026
ETB	013400	000027
CAN	014000	000030
EM	014400	000031
SUB	015000	000032
ESC	015400	000033
FS	016000	000034
GS	016400	000035
RS	017000	000036
US	017400	000037
SPACE	020000	000040
!	020400	000041
"	021000	000042
#	021400	000043
$	022000	000044
%	022400	000045
&	023000	000046
'	023400	000047
(	024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
,	026000	000054
-	026400	000055
.	027000	000056
/	027400	000057
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
[	055400	000133
\	056000	000134
]	056400	000135
^	057000	000136
_	057400	000137
`	060000	000140
{	075400	000173
|	076000	000174
}	076400	000175
~	077000	000176
DEL	077400	000177



OCTAL ARITHMETIC

ADDITION

TABLE

0	01	02	03	04	05	06	07
1	02	03	04	05	06	07	10
2	03	04	05	06	07	10	11
3	04	05	06	07	10	11	12
4	05	06	07	10	11	12	13
5	06	07	10	11	12	13	14
6	07	10	11	12	13	14	15
7	10	11	12	13	14	15	16

EXAMPLE

```
Add:    3677    OCTAL
      + 1331    OCTAL
      -------
       (111-)   CARRIES
      -------
        5230    OCTAL
```

MULTIPLICATION

TABLE

1	02	03	04	05	06	07
2	04	06	10	12	14	16
3	06	11	14	17	22	25
4	10	14	20	24	30	34
5	12	17	24	31	36	43
6	14	22	30	36	44	52
7	16	25	34	43	52	61

EXAMPLE

```
Multiply:  657    OCTAL
         x  54    OCTAL
         -------
          3274
          4153
         -------
         45024    OCTAL
```

(Reminder: add in octal)

COMPLEMENT

To find the two's complement form of an octal number. (Same procedure whether converting from positive to negative or negative to positive.)

RULE

1. Subtract from the maximum representable octal value.
2. Add one.

EXAMPLE

Two's complement of 556_8

```
  177777
- 000556
--------
  177221
+      1
--------
```

177222_8

8200-43

OCTAL/DECIMAL CONVERSIONS

OCTAL TO DECIMAL

TABLE

OCTAL	DECIMAL
0- 7	0- 7
10-17	8-15
20-27	16-23
30-37	24-31
40-47	32-39
50-57	40-47
60-67	48-55
70-77	56-63
100	64
200	128
400	256
1000	512
2000	1024
4000	2048
10000	4096
20000	8192
40000	16384
77777	32767

EXAMPLE

Convert 463_8 to a decimal integer.

$400_8 = 256_{10}$

$60_8 = 48_{10}$

$3_8 = 3_{10}$

307 decimal

DECIMAL TO OCTAL

TABLE

DECIMAL	OCTAL
1	1
10	12
20	24
40	50
100	144
200	310
500	764
1000	1750
2000	3720
5000	11610
10000	23420
20000	47040
32767	77777

EXAMPLE

Convert 5229_{10} to an octal integer.

$5000_{10} = 11610_8$

$200_{10} = 310_8$

$20_{10} = 24_8$

$9_{10} = 11_8$

12155_8

(Reminder: add in octal)

NEGATIVE DECIMAL TO TWO'S COMPLEMENT OCTAL

TABLE

DECIMAL	2's COMP
-1	177777
-10	177766
-20	177754
-40	177730
-100	177634
-200	177470
-500	177014
-1000	176030
-2000	174060
-5000	166170
-10000	154360
-20000	130740
-32768	100000

EXAMPLE

Convert -629_{10} to two's complement octal.

$-500_{10} = 177014_8$

$-100_{10} = 177634_8$

$-20_{10} = 177754_8$ (Add in octal)

$-9_{10} = 177767_8$

176613_8

For reverse conversion (two's complement octal to negative decimal):

1. Complement, using procedure on facing page.
2. Convert to decimal, using OCTAL TO DECIMAL table.

8200-44

MATHEMATICAL EQUIVALENTS

2 ± η IN DECIMAL

2^{η}	η	$2^{-\eta}$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1 024	10	0.00097 65625
2 048	11	0.00048 82812 5
4 096	12	0.00024 41406 25
8 192	13	0.00012 20703 125
16 384	14	0.00006 10351 5625
32 768	15	0.00003 05175 78125
65 536	16	0.00001 52587 89062 5
131 072	17	0.00000 76293 94531 25
262 144	18	0.00000 38146 97265 625
524 288	19	0.00000 19073 48632 8125
1 048 576	20	0.00000 09536 74316 40625
2 097 152	21	0.00000 04768 37158 20312 5
4 194 304	22	0.00000 02384 18579 10156 25
8 388 608	23	0.00000 01192 09289 55078 125
16 777 216	24	0.00000 00596 04644 77539 0625
33 554 432	25	0.00000 00298 02322 38769 53125
67 108 864	26	0.00000 00149 01161 19384 76562 5
134 217 728	27	0.00000 00074 50580 59692 38281 25
268 435 456	28	0.00000 00037 25290 29846 19140 625
536 870 912	29	0.00000 00018 62645 14923 09570 3125
1 073 741 824	30	0.00000 00009 31322 57461 54785 15625
2 147 483 648	31	0.00000 00004 65661 28730 77392 57812 5
4 294 967 296	32	0.00000 00002 32830 64365 38696 28906 25

10 ± η IN OCTAL

10^{η}	η	$10^{-\eta}$
1	0	1.000 000 000 000 000 000 00
12	1	0.063 146 314 631 463 146 31
144	2	0.005 075 341 217 270 243 66
1 750	3	0.000 406 111 564 570 651 77
23 420	4	0.000 032 155 613 530 704 15
303 240	5	0.000 002 476 132 610 706 64
3 641 100	6	0.000 000 206 157 364 055 37
46 113 200	7	0.000 000 015 327 745 152 75
575 360 400	8	0.000 000 001 257 143 561 06
7 346 545 000	9	0.000 000 000 104 560 276 41
112 402 762 000	10	0.000 000 000 006 676 337 66
1 351 035 564 000	11	0.000 000 000 000 537 657 77
16 432 451 210 000	12	0.000 000 000 000 043 136 32
221 411 634 520 000	13	0.000 000 000 000 003 411 35
2 657 142 036 440 000	14	0.000 000 000 000 000 264 11
34 327 724 461 500 000	15	0.000 000 000 000 000 022 01
434 157 115 760 200 000	16	0.000 000 000 000 000 001 63
5 432 127 413 542 400 000	17	0.000 000 000 000 000 000 14
67 405 553 164 731 000 000	18	0.000 000 000 000 000 000 01

8200-45

MATHEMATICAL EQUIVALENTS

2^x IN DECIMAL

x	2^x	x	2^x	x	2^x
0.001	1.00069 33874 62581	0.01	1.00695 55500 56719	0.1	1.07177 34625 36293
0.002	1.00138 72557 11335	0.02	1.01395 94797 90029	0.2	1.14869 83549 97035
0.003	1.00208 16050 79633	0.03	1.02101 21257 07193	0.3	1.23114 44133 44916
0.004	1.00277 64359 01078	0.04	1.02811 38266 56067	0.4	1.31950 79107 72894
0.005	1.00347 17485 09503	0.05	1.03526 49238 41377	0.5	1.41421 35623 73095
0.006	1.00416 75432 38973	0.06	1.04246 57608 41121	0.6	1.51571 65665 10398
0.007	1.00486 38204 23785	0.07	1.04971 66836 23067	0.7	1.62450 47927 12471
0.008	1.00556 05803 98468	0.08	1.05701 80405 61380	0.8	1.74110 11265 92248
0.009	1.00625 78234 97782	0.09	1.06437 01824 53360	0.9	1.86606 59830 73615

$\eta \log_{10} 2$, $\eta \log_2 10$ IN OCTAL

η	$\eta \log_{10} 2$	$\eta \log_2 10$	η	$\eta \log_{10} 2$	$\eta \log_2 10$
1	0.30102 99957	3.32192 80949	6	1.80617 99740	19.93156 85693
2	0.60205 99913	6.64385 61898	7	2.10720 99696	23.25349 66642
3	0.90308 99870	9.96578 42847	8	2.40823 99653	26.57542 47591
4	1.20411 99827	13.28771 23795	9	2.70926 99610	29.89735 28540
5	1.50514 99783	16.60964 04744	10	3.01029 99566	33.21928 09489

MATHEMATICAL CONSTANTS IN OCTAL SCALE

$\pi = (3.11037\ 552421)_{(8)}$

$\pi^{-1} = (0.24276\ 301556)_{(8)}$

$\sqrt{\pi} = (1.61337\ 611067)_{(8)}$

$\ln \pi = (1.11206\ 404435)_{(8)}$

$\log_2 \pi = (1.51544\ 163223)_{(8)}$

$\sqrt{10} = (3.12305\ 407267)_{(8)}$

$e = (2.55760\ 521305)_{(8)}$

$e^{-1} = (0.27426\ 530661)_{(8)}$

$\sqrt{e} = (1.51411\ 230704)_{(8)}$

$\log_{10} e = (0.33626\ 754251)_{(8)}$

$\log_2 e = (1.34252\ 166245)_{(8)}$

$\log_2 10 = (3.24464\ 741136)_{(8)}$

$\gamma = (0.44742\ 147707)_{(8)}$

$\ln \gamma = -(0.43127\ 233602)_{(8)}$

$\log_2 \gamma = -(0.62573\ 030645)_{(8)}$

$\sqrt{2} = (1.32404\ 746320)_{(8)}$

$\ln 2 = (0.54271\ 027760)_{(8)}$

$\ln 10 = (2.23273\ 067355)_{(8)}$

8200-46

OCTAL COMBINING TABLES

MEMORY REFERENCE INSTRUCTIONS

INDIRECT ADDRESSING

Refer to octal instruction codes given on the following page.
To combine code for indirect addressing, merge "100000" with octal instruction code.

REGISTER REFERENCE INSTRUCTIONS

SHIFT-ROTATE GROUP (SRG)

1. select to operate A or B.
2. select 1 to 4 instructions, not more than one from each column.
3. combine octal codes (leading zeros omitted) by inclusive or.
4. order of execution is from column 1 to column 4.

A OPERATIONS

1	2	3	4
ALS (1000)	CLE (40)	SLA (10)	ALS (20)
ARS (1100)			ARS (21)
RAL (1200)			RAL (22)
RAR (1300)			RAR (23)
ALR (1400)			ALR (24)
ERA (1500)			ERA (25)
ELA (1600)			ELA (26)
ALF (1700)			ALF (27)

B OPERATIONS

1	2	3	4
BLS (5000)	CLE (4040)	SLB (4010)	BLS (4020)
BRS (5100)			BRS (4021)
RBL (5200)			RBL (4022)
RBR (5300)			RBR (4023)
BLR (5400)			BLR (4024)
ERB (5500)			ERB (4025)
ELB (5600)			ELB (4026)
BLF (5700)			BLF (4027)

ALTER-SKIP GROUP (ASG)

1. select to operate on A or B.
2. select 1 to 8 instructions, not more than one from each column.
3. combine octal codes (leading zeros omitted) by inclusive or.
4. order of execution is from column 1 to column 8.

A OPERATIONS

1	2	3	4
CLA (2400)	SEZ (2040)	CLE (2100)	SSA (2020)
CMA (3000)		CME (2200)	
CCA (3400)		CCE (2300)	

5	6	7	8
SLA (2010)	INA (2004)	SZA (2002)	RSS (2001)

B OPERATIONS

1	2	3	4
CLB (6400)	SEZ (6040)	CLE (6100)	SSB (6020)
CMB (7000)		CME (6200)	
CCB (7400)		CCE (6300)	

5	6	7	8
SLB (6010)	INB (6004)	SZB (6002)	RSS (6001)

INPUT/OUTPUT INSTRUCTIONS

CLEAR FLAG

Refer to octal instruction codes given on the following page.
To clear flag after execution (instead of holding flag), merge "001000" with octal instruction code.

8200-47

INSTRUCTION CODES IN OCTAL

Memory Reference

ADA	04(0XX)—
ADB	04(1XX)—
AND	01(0XX)—
CPA	05(0XX)—
CPB	05(1XX)—
IOR	03(0XX)—
ISZ	03(1XX)—
JMP	02(1XX)—
JSB	01(1XX)—
LDA	06(0XX)—
LDB	06(1XX)—
STA	07(0XX)—
STB	07(1XX)—
XOR	02(0XX)—

Binary

Shift-Rotate

ALF	001700
ALR	001400
ALS	001000
ARS	001100
BLF	005700
BLR	005400
BLS	005000
BRS	005100
CLE	000040
ELA	001600
ELB	005600
ERA	001500
ERB	005500
NOP	000000
RAL	001200
RAR	001300
RBL	005200
RBR	005300
SLA	000010
SLB	004010

Alter-Skip

CCA	003400
CCB	007400
CCE	002300
CLA	002400
CLB	006400
CLE	002100
CMA	003000
CMB	007000
CME	002200
INA	002004
INB	006004
RSS	002001
SEZ	002040
SLA	002010
SLB	006010
SSA	002020
SSB	006020
SZA	002002
SZB	006002

Input/Output

CLC	1067–
CLF	1031–
CLO	103101
HLT	1020–
LIA	1025–
LIB	1065–
MIA	1024–
MIB	1064–
OTA	1026–
OTB	1066–
SFC	1022–
SFS	1023–
SOC	102201
SOS	102301
STC	1027–
STF	1021–
STO	102101

Extended Arithmetic

ASL	1000(01X)–
ASR	1010(01X)–
DIV	100400
JLA	100600
DLD	104200
DST	104400
JLB	104600
LSL	1000(10X)–
LSR	1010(10X)–
MPY	100200
RRL	1001(00X)–
RRR	1011(00X)–

Binary

Ext. Inst. Group

ADX	105746
ADY	105756
CAX	101741
CAY	101751
CBS	105774
CBT	105766
CBX	105741
CBY	105751
CMW	105776
CXA	101744
CXB	105744
CYA	101754
CYB	105754
DSX	105761
DSY	105771
ISX	105760
ISY	105770
JLY	105762
JPY	105772
LAX	101742
LAY	101752
LBT	105763
LBX	105742
LBY	105752
LDX	105745
LDY	105755
MBT	105765
MVW	105777
SAX	101740
SAY	101750
SBS	105773
SBT	105764
SBX	105740
SBY	105750
SFB	105767
STX	105743
STY	105753
TBS	105775
XAX	101747
XAY	101757
XBX	105747
XBY	105757

Floating Point

FAD	105000
FDV	105060
FIX	105100
FLT	105120
FMP	105040
FSB	105020
.FIXD	105104
.FLTD	105124
.TADD	105002
.TDIV	105062
.TFTD	105122
.TFXD	105106
.TFXS	105102
.TMPY	105042
.TSUB	105022

Language Inst. Set

.BLE	105207
.CFER	105231
.DFER	105205
.CPM	105236
.ENTC	105235
.ENTN	105234
.ENTP	105224
.ENTR	105223
..FCM	105232
.NGL	105214
.SETP	105227
..TCM	105233
.XFER	105220
.ZFER	105237

Double Integer

.DAD	105014
.DCO	105204
.DDE	105211
.DDI	105074
.DDIR	105134
.DDS	105213
.DIN	105210
.DIS	105212
.DNG	105203
.DMP	105054
.DSB	105034
.DSBR	105114

VMA/EMA

.IMAP	105250
.IRES	105244
.JMAP	105252
.JRES	105245
.LBP	105257
.LBPR	105256
.LPX	105255
.LPXR	105254
.PMAP	105240

Oper. Syst. Set

.CPUID	105300
.FWID	105301
.SIP	105303
.WFI	105302

Scientific Inst. Set

ALOG	105322
ALOGT	105327
ATAN	105323
/ATLG	105333
/CMRT*	105332

INSTRUCTION CODES IN OCTAL (Continued)

COS	105324	VMIB	105115	DVMAX	105131	MB20	101735	XSA2	101722
DPOLY	105331	VMIN	105113	DVMIB	105135	MB21	101736	XSB1	105725
EXP	105326	VMOV	105116	DVMIN	105133	MB22	101737	XSB2	105722
.FPWR	105334	VMPY	105004	DVMOV	105136	MW00	105727		
SIN	105325	VNRM	105107	DVMPY	105024	MW01	105730	**Code and Data Sep.**	
SQRT	105321	VPIV	105101	DVNRM	105127	MW02	105731		
TAN	105320	VSAD	105006	DVPIV	105121	MW10	105732	ADQA	101413
TANH	105330	VSDV	105011	DVSAD	105026	MW11	105733	ADQB	105413
.TPWR	105335	VSMY	105010	DVSDV	105031	MW12	105734	CACQ	101407
		VSSB	105007	DVSMY	105030	MW20	105735	CAZ	101411
Vector Inst. Set		VSUB	105003	DVSSB	105027	MW21	105736	CBCQ	105407
		VSUM	105105	DVSUB	105023	MW22	105737	CBZ	105411
VABS	105103	VSWP	105117	DVSUM	105125	SIMP	105707	CCQA	101406
VADD	105001	DVABS	105123	DVSWP	105137	STMP	105703	CCQB	105406
VDIV	105005	DVADD	105021			SPMR	105701	CIQA	101412
VDOT	105110	DVDIV	105025	**Dynamic Map Syst.**		SWMP	105706	CIQB	105412
VMAB	105112	DVDOT	105130			XCA1	101726	CZA	101410
VMAX	105111	DVMAB	105132	LDMP	105702	XCA2	101723	CZB	105410
				LPMR	105700	XCB1	105726	EXIT	105417
				LWD1	105704	XCB2	105723	EXIT1	105415
				LWD2	105705	XJMP	105710	EXIT2	105416
				MB00	101727	XJCQ	105711	PCALI	105400
				MB01	101730	XLA1	101724	PCALN	105404
				MB02	101731	XLA2	101721	PCALR	105403
				MB10	101732	XLB1	105724	PCALV	105402
				MB11	101733	XLB2	105721	PCALX	105401
				MB12	101734	XSA1	101725	SDSP	105405

Assuming: no indirect addressing.
no combined instructions.
shifts taken in first position only.
hold flag after I/O execution.

* Not directly user callable.
Used by HP software.

Refer to preceding page for octal combining tables.

BASE SET INSTRUCTION CODES IN BINARY

15	14 13 12	11	10	9	8 7 6	5 4 3	2 1 0
MEMORY REFERENCE INSTRUCTIONS							
D/I	AND 001	0	Z/C	← MEMORY ADDRESS →			
D/I	XOR 010	0	Z/C				
D/I	IOR 011	0	Z/C				
D/I	JSB 001	1	Z/C				
D/I	JMP 010	1	Z/C				
D/I	ISZ 011	1	Z/C				
D/I	AD* 100	A/B	Z/C				
D/I	CP* 101	A/B	Z/C				
D/I	LD* 110	A/B	Z/C				
D/I	ST* 111	A/B	Z/C				
SHIFT/ROTATE GROUP							
0	000	A/B	0	D/E	*LS 000	†CLE D/E ‡SL*	*LS 000
		A/B	0	D/E	*RS 001	D/E	*RS 001
		A/B	0	D/E	R*L 010	D/E	R*L 010
		A/B	0	D/E	R*R 011	D/E	R*R 011
		A/B	0	D/E	*LR 100	D/E	*LR 100
		A/B	0	D/E	ER* 101	D/E	ER* 101
		A/B	0	D/E	EL* 110	D/E	EL* 110
		A/B	0	D/E	*LF 111	D/E	*LF 111
		NOP	000		000	000	000
ALTER/SKIP GROUP							
0	000	A/B	1	CL* 01	CLE 01	SEZ SS* SL*	IN* SZ* RSS
		A/B		CM* 10	CME 10		
		A/B		CC* 11	CCE 11		
INPUT/OUTPUT GROUP							
1	000		1	H/C	HLT 000	← SELECT CODE →	
			1	0	STF 001		
			1	1	CLF 001		
			1	0	SFC 010		
			1	0	SFS 011		
		A/B	1	H/C	MI* 100		
		A/B	1	H/C	LI* 101		
		A/B	1	H/C	OT* 110		
		0	1	H/C	STC 111		
		1	1	H/C	CLC 111		
			1	0	STO 001	000	001
			1	1	CLO 001	000	001
			1	H/C	SOC 010	000	001
			1	H/C	SOS 011	000	001
EXTENDED ARITHMETIC GROUP							
1	000	MPY**	000		010	000	000
		DIV**	000		100	000	000
		JLA	000		110	000	000
		DLD**	100		010	000	000
		DST**	100		100	000	000
		JLB	100		110	000	000
		ASR	001		000	0 1	← NUMBER OF BITS →
		ASL	000		000	0 1	
		LSR	001		000	1 0	
		LSL	000		000	1 0	
		RRR	001		001	0 0	
		RRL	000		001	0 0	
FLOATING POINT INSTRUCTIONS							
1	000	101			00	FAD 000	0 000
						FSB 001	
						FMP 010	
						FDV 011	
						FIX 100	
						FLT 101	

Notes: * = A or B, according to bit 11.
D/I, A/B, Z/C, D/E, H/C coded 0/1.
**Second word is Memory Address.

†CLE: Only this bit is required.
‡SL*: Only this bit and bit 11 (A/B as applicable) are required.

8200-56

BASE SET INSTRUCTION CODES IN BINARY (Continued)

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
FLOATING POINT INSTRUCTION (Continued)					
1	000	101	00	.TADD 000 .TSUB 001 .TMPY 010 .TDIV 011 .TFXS 100 .TFTS 101 .FIXD 100 .FLTD 101 .TFXD 100 .TFTD 101	0 010 100 110
DOUBLE INTEGER INSTRUCTIONS					
1	000	101	000	001 011 101 111	.DAD 100 .DSB 100 .DMP 100 .DDI 100
			001 001 010	001 011 000 001	.DSBR 100 .DDIR 100 .DNG 011 .DCO 100 .DIN 000 .DDE 001 .DIS 010 .DDS 011
LANGUAGE INSTRUCTION SET					
1	000	101	010	0 00	.DFER 101 .BLE 111
				01	.NGL 100
				10	.XFER 000 .ENTR 011 .ENTP 100 .SETP 111
				11	.CFER 001 ..FCM 010 ..TCM 011 .ENTN 100 .ENTC 101 .CPM 110 .ZFER 111
VIRTUAL MEMORY INSTRUCTIONS					
1	000	101	010	100	.PMAP 000 .IRES 100 .JRES 101
				101	.IMAP 000 .JMAP 010 .LPXR 100 .LPX 101 .LBPR 110 .LBP 111
OPERATING SYSTEM INSTRUCTION SET					
1	000	101	011	000	.CPUID 000 .FWID 001 .WFI 010 .SIP 011

BASE SET INSTRUCTION CODES IN BINARY (Continued)

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
DMS INSTRUCTIONS					
1	000	1 01	111	000	LPMR 000
		1			SPMR 001
		1			LDMP 010
		1			STMP 011
		1			LWD1 100
		1			LWD2 101
		1			SWMP 110
		1			SIMP 111
		1		001	XJMP 000
		A/B		010	XL*1 100
		A/B			XS*1 101
		A/B			XC*1 110
		A/B			XL*2 001
		A/B			XS*2 010
		A/B			XC*2 011
		B/W			M°00 111
		B/W		011	M°01 000
		B/W			M°02 001
		B/W			M°10 010
		B/W			M°11 011
		B/W			M°12 100
		B/W			M°20 101
		B/W			M°21 110
		B/W			M°22 111
SCIENTIFIC INSTRUCTION SET					
1	000	101	011	010	TAN 000
					SQRT 001
					ALOG 010
					ATAN 011
					COS 100
					SIN 101
					EXP 110
					ALOGT 111
				011	TANH 000
					DPOLY 001
					/CMRT 010
					/ATLG 011
					.FPWR 100
					.TPWR 101
VECTOR INSTRUCTION SET					
1	000	101	000	000	VADD 001
					VSUB 011
					VMPY 100
					VDIV 101
					VSAD 110
					VSSB 111
				001	VSMY 000
					VSDV 001
				010	DVADD 001
					DVSUB 011
					DVMPY 100
					DVDIV 101
					DVSAD 110
					DVSSB 111
				011	DVSMY 000
					DVSDV 001
			001	000	VPIV 001
					VABS 011
					VSUM 101

Notes: * = A (0) or B (1), according to bit 11.
° = B (0) or W (1), according to bit 11.

BASE SET INSTRUCTION CODES IN BINARY (Continued)

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
VECTOR INSTRUCTION SET (Continued)					
1	000	101	001	001	VNRM 111 VDOT 000 VMAX 001 VMAB 010 VMIN 011 VMIB 101 VMOV 110 VSWP 111
				010	DPIV 001 DVABS 011 DVSUM 101 DVNRM 111
				011	DVDOT 000 DVMAX 001 DVMAB 010 DVMIN 011 DVMIB 101 DVMOV 110 DVSWP 111
CODE AND DATA SEPARATION					
1	000	001	100	000	CCQA 110 CACQ 111
				001	CZA 000 CAZ 001 CIQA 010 ADQA 011
		101	100	000	PCALI 000 PCALX 001 PCALV 010 PCALR 011 PCALN 100 SDSP 101 CCQB 110 CBCQ 111
				001	CZB 000 CBZ 001 CIQB 010 ADQB 011 EXIT1 101 EXIT2 110 EXIT 111

BASE SET INSTRUCTION CODES IN BINARY (Continued)

EXTENDED INSTRUCTION GROUP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAX/SAY/SBX/SBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	0
CAX/CAY/CBX/CBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	1
LAX/LAY/LBX/LBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	1	0
STX/STY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	0	1	1
CXA/CYA/CXB/CYB	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	0	0
LDX/LDY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	0	1
ADX/ADY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	1	0
XAX/XAY/XBX/XBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	1	1
ISX/ISY/DSX/DSY	1	0	0	0	1	0	1	1	1	1	1	1	X/Y	0	0	I/D
JUMP INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1		0	1	0

JLY = 0
JPY = 1

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

LBT	= 0	1	1
SBT	= 1	0	0
MBT	= 1	0	1
CBT	= 1	1	0
SFB	= 1	1	1

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

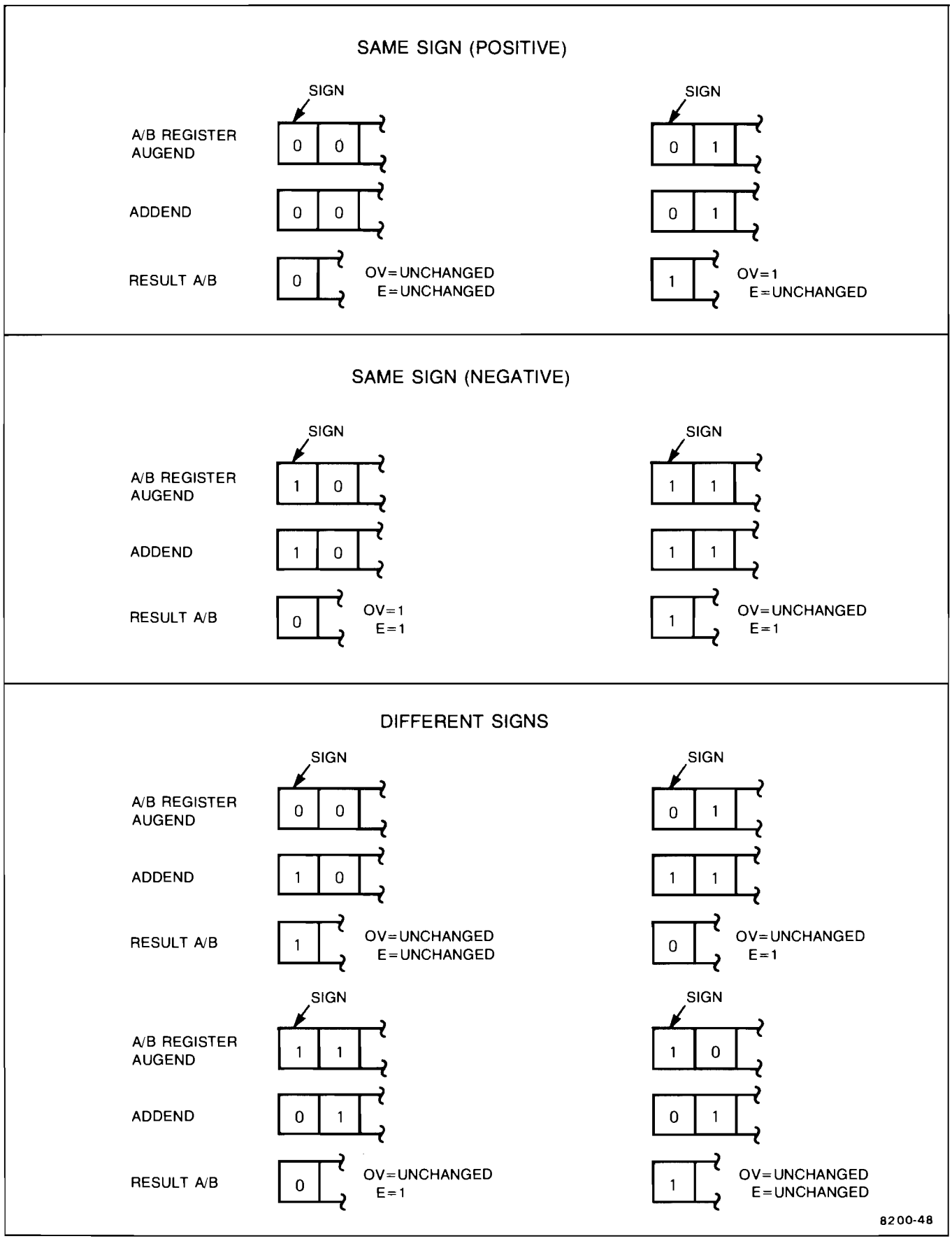
SBS	= 0	1	1
CBS	= 1	0	0
TBS	= 1	0	1

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

CMW = 0
MVW = 1

8200-52

EXTEND AND OVERFLOW EXAMPLES



INTERRUPT AND CONTROL SUMMARY

INST	S.C. 00	S.C. 01	S.C. 02	S.C. 03	S.C. 04	S.C. 05	S.C. 06	S.C. 07
STC	NOP	NOP	Enable break mode.	NOP	Enable Type 2 and 3 interrupts.	Enable multiple-bit error interrupts.	Turn on Time Base Generator.	Turn on memory protect.
CLC	System reset.	NOP	NOP	NOP	Disable Type 2 and 3 interrupts.	Disable multiple-bit error interrupts.	Turn off Time Base Generator.	NOP
STF	Enable Type 3 interrupts.	STO	Disable Global Register.	NOP	Flush cache	NOP	Set Time Base Generator flag.	NOP
CLF	Disable Type 3 interrupts.	CLO	Enable Global Register.	NOP	NOP	NOP	Clear Time Base Generator flag.	NOP
SFS	Skip if Type 3 interrupts are enabled.	SOS	Skip if Global Register is disabled.	NOP	Skip if power not going down	NOP	Skip if Time Base Generator flag is set.	NOP
SFC	Skip if Type 3 interrupts are disabled.	SOC	Skip if Global Register is enabled.	NOP	Skip if power is going down.	NOP	Skip if Time Base Generator flag is clear.	NOP
LI*	Load from interrupt mask register.	Load from processor status register.	Load from Global Register.	Load from PSAVE or (with ,C) ROMP.	Load from central interrupt register.	Load bits 0-15 from parity error register, or (with ,C) bits 16-31.	NOP	Load from violation register.
MI*	NOP	Merge from processor status register.	NOP	NOP	NOP	NOP	NOP	NOP
OT*	Output to interrupt mask register.	Output to processor status register.	Output to Global Register. (Note 1)	Output to PSAVE or (with ,C) ROMP.	Output to central interrupt register.	NOP	NOP	NOP

Note 1: An OTA/B 2 with A/B equal to one through seven establishes a diagnose mode; refer to paragraph 7-22 for details.

Note 2: An OTA 05 or OTB 05 can be used to force memory errors to test the Error Detection and Correction logic; the A- or B-register specifies the address where the checkbit pattern held in the X-register will be written.