SPERRY UNIVAC

V70 Series Architecture

Reference Manual

Mini-Computer Operations

2722 Michelson Drive
P.O. Box C-19504
Irvine, California 92713

UP-8634 Rev. 1

SPERRY UNIVAC®

V70 SERIES ARCHITECTURE
REFERENCE MANUAL

UP—8634 Rev. 1
98A 9906 002
OCTOBER 1979

The statements in this publication are not intended to create any warranty, express or implied. Equipment specifications and performance characteristics stated herein may be changed at any time without notice. Address comments regarding this document to Sperry Univac, Mini-Computer Operations, Publications Department, 2722 Michelson Drive, P.O. Box C-19504, Irvine, California, 92713.

PAGE STATUS SUMMARY

ISSUE: UP-8634 Rev. 1

Part/Section	Page Number	Update Level
Cover		
Title Page		
PSS	1	
CR	1	
Contents	1 thru 3	
1	1 thru 6	
2	1 thru 14	
3	1 thru 2	
4	1 thru 11	
5	1 thru 114	
A	1 thru 10	
B	1 thru 9	
C	1	
D	1 thru 6	

Part/Section	Page Number	Update Level

Part/Section	Page Number	Update Level

*New pages

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

CHANGE RECORD

Change Designation	Issue Date	Change Description
All	10-76	Original issue.
V75	5-77	Revised format of manual and added V75 and ECS instructions. Revised BCS and IDE instructions.
All	11-77	Deleted all references to Varian.
Update A	6-79	Added V77-800 standard extension instructions and formats 26 and 27. Deleted V77-600 floating point processor instructions. Added V77-800 information to existing BCS instruction. Revised appendix A.
Revision 1	10-79	Incorporated information from update A. Deleted octal/decimal conversions charts.

Change Procedure:

When changes are made to this manual, updated pages are issued. These updated pages are either added to this manual or used to replace obsolete pages. The specific pages affected by each change are identified on the PAGE STATUS SUMMARY page.

Printed in U.S.A.

TABLE OF CONTENTS

SECTION 1
INTRODUCTION

SECTION 2
INSTRUCTION FORMATS

SECTION 3
DATA FORMATS

SECTION 3 *(continued)*

SECTION 4
ADDRESSING MODES

SECTION 5
INSTRUCTION SET

SECTION 5 *(continued)*

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1
INTRODUCTION

This manual is the basic reference manual for the SPERRY UNIVAC V70 series computers. The manual describes the machine functions to the level of detail required for preparing an assembly language program. It does not, however, describe the notation and conventions used in writing such a program. For this information, the user should refer to the appropriate software manuals, such as those described in section 1.3.

All machine functions described in this manual are not necessarily available with every V70 series computer. For information on the characteristics and features of a specific computer model, the user should refer to the appropriate hardware reference manual, such as those described in section 1.2.

This manual consists of five sections and several appendixes:

Section 2 describes the formats for all the instructions in the instruction set.

Section 3 describes the various formats used for data and addresses within the system.

Section 4 describes the addressing modes used by the computers.

Section 5 contains the instruction set with a description of each instruction.

1.1 V70 SERIES COMPUTERS

The V70 series computers have been designed with flexibility as the keystone. They offer the capability to configure systems with a wide range of application requirements, modular expansion and open-ended system growth, microprogramming for control, adaptability to changing technology, reliability, and easy maintenance. V70 series computers are designed for maximum performance in instrumentation, data acquisition, and communications systems, making them ideal for a variety of scientific, commercial, and industrial applications.

The instruction set of a V70 comprises over 180 instructions, many of which can be microcoded to extend the effective repertoire to several hundred instructions.

1.1.1 Hardware General Description

The central processing unit features a set of general purpose registers, 16-bit wide data paths, arithmetic and logical function generators, and data-path selection logic under control of microprogramming firmware stored in a read-only memory or writable control store. The processor, while completely general purpose, is offered in a variety of configurations for the widest possible range of applications. Also available is a convenient full programmer's console.

The V70 series maintains software compatibility with the 620 series computers through microprogramming. Increased performance is obtained through a faster processing system. This compatibility includes direct, multilevel indirect, immediate, preindexing and postindexing, relative, and extended addressing modes.

1.1.2 Software General Description

Standard software for the V70 series includes the V70 Omnitask Real-time Executive (VORTEX or VORTEX II), which is a modular operating system for controlling, scheduling, and monitoring tasks in a real-time multiprogramming environment. Major subsystems offered by Sperry Univac includes TOTAL for data base management, VTAM for data communications, PRONTO for transaction processing and network control, HASP for remote job entry, and TSS for multiuser editing and time-shared BASIC. Other software features are FORTRAIN IV, COBOL, RPG II, and VIDEO, an on-line data entry program.

1.1.3 User Services

User services such as field service and customer education are offered by Sperry Univac to assist the user in operating and maintaining his system.

1.1.3.1 Field Service

The Sperry Univac field service organization provides a comprehensive service program to assist the user in system planning, installation, and maintenance. Service contracts may be for full-service maintenance, per-call maintenance, or on-site maintenance.

With the full-service maintenance contract, Sperry Univac assumes the responsibility for all maintenance and performs all the corrective and preventive maintenance necessary to keep the user's system up and running. The user receives guaranteed on-site response, scheduled preventive maintenance, and all enhancements to keep the system up to date. In addition, the maintenance contract also places at the user's disposal the resources of Sperry Univac's nationwide network of fully qualified service representatives and technical liaison engineers.

The per-call maintenance contract provides corrective and preventive maintenance on a per-call basis. This arrangement is for users who have their own service capability and from time-to-time need specialized service. Charges for per-call maintenance are made on a time and material basis.

On-site service is available for customers with unique applications or where a heavy workload demands almost continuous use of equipment. With this contract, the user receives the services of a Sperry Univac service representative who is dedicated exclusively to keeping the user's system up and running.

1.1.3.2 Customer Education

The Sperry Univac Minicomputer Operation's department of customer education offers regularly scheduled training classes covering the complete spectrum of Sperry Univac's

growing mini-computer family. Both programming and maintenance courses are offered as well as a complete course of the dynamic software VORTEX systems. All classes are a combination of lecture and applications with special emphasis given to hands-on training.

For further details, a training course brochure is available through any local Sperry Univac office.

1.2 V70 SERIES HARDWARE MANUALS

In addition to this manual, other publications are available which describe individual computers in the series, system components, and peripheral devices.

1.2.1 System Reference Manual

A System Reference Manual is provided with each V70 system. These manuals contain system hardware information that is unique to the particular model. Contents of these manuals include:

- Features
- Options
- Physical characteristic
- Specifications
- Memory
- System configurations
- Installation
- Operation
- Input/output

1.2.2 System Documentation

The system documentation is assembled for each system prior to its shipment. The contents include:

- System memoranda
- System arrangement drawing
- Hardware performance standards

- Test data
- Logic diagrams and schematics
- Option and controller documentation
- All engineering notices affecting any supplied documentation

1.2.3 V70 Series Technical Manuals

A technical manual is provided for each major hardware component of a V70 series system. Major components are the processor, memory modules, mainframe options, and power supplies. The manuals contain the following information:

- Installation
- Operation
- Theory of operation
- Maintenance
- Mnemonic definitions

1.2.4 Peripheral Equipment Manuals

A peripheral equipment manual (or manuals) is provided for each peripheral device in the system. These manuals are supplied by the peripheral equipment manufacturer and shipped as part of the system documentation.

1.3 V70 SERIES SOFTWARE MANUALS

The V70 series software manuals describe the various software languages and operating systems.

1.3.1 VORTEX Manuals

The VORTEX Reference Manual describes the V70 Omnitask Real-Time Executive (VORTEX) operating systems. It provides the user with the information needed to operate and program an installation using the system. The VORTEX Installation Manual explains in detail the procedures for determining system requirements and capabilities. It also describes the procedures for system generation: e.g., loading program modules.

1.3.2 Assembly Language Reference Manual

The Assembly Language Reference Manual describes the symbolically coded instructions, directives, and data used by the assembler. It explains their use so that the programmer may specify instructions, addresses, address modifications, and constants in a straightforward manner meaningful to the computer.

1.3.3 Test Programs Manual

All processor, memory, and mainframe-option test programs are described in the MAINTAIN III Reference Manual. The manual describes the purpose and operation of the tests and explains error message printouts or other fault indications.

1.3.4 Microprogramming Guide

The Microprogramming Guide is provided for systems with writable control store. It describes the fields of the control store word and the use of the microprogramming assembler.

1.3.5 Software Package

A software package is assembled for each system prior to its shipment. Included in this package are:

- Letter to the customer
- Listings
- Write-ups*
- Paper tapes
- Card decks
- Disc pack
- Magnetic tapes

* Write-ups have document numbers starting with 32W and contain operating information not covered in manuals or software performance specifications.

1.3.6 Other Software Manuals

Separate manuals are offered for other software facilities, such as:

- V70 FORTRAN IV

- V70 BASIC
- V70 RPG II
- V70 COBOL
- V70 TOTAL (data base management)
- V70 HASP/RJE (remote job entry)
- V70 VIDEO (on-line data entry)
- VTAM (VORTEX telecommunications access method)
- V70 Message Switching System

SECTION 2
INSTRUCTION FORMATS

All instructions contain a code field directing the type of operation to be performed. They may have one or more additional fields indicating the addressing mode to be used, memory location to be accessed, register or registers to be used, or conditions to be tested.

Data may also be contained in the instruction. This data may be an operand, a direct/indirect address, an external device address, or an external device function.

Instructions fall into four format groups and are either addressing or non-addressing, single or double word:

- Single-word addressing
- Single-word non-addressing
- Double-word addressing
- Double-word non-addressing

Each of the four format groups contains one or more formats. These formats are designated 1 through 27

Instructions are classified as "addressing" when, as a result of instruction decoding, they require memory to be accessed.

Some instructions (see section 5) make eight registers available to the programmer. Table 2-1 identifies the registers and their corresponding registers in other instructions.

Table 2-1. Multiple Registers

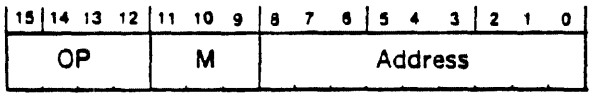
Nomenclature	Function	Corresponding Nomenclature
R0	Byte or word accumulator, or most-significant half of double-precision register R0-R1.	A
R1	Word accumulator, index register, or least-significant half of double-precision register R0-R1	B

R2	General purpose register	X
R3	General purpose register	
R4	General purpose register or most-significant half of double-precision register R4-R5	
R5	General purpose register or least-significant half of double-precision register R4-R5	
R6	General purpose register	
R7	General purpose register	

In the formats which follow, the term "OP" identifies the field containing the instruction operation code.

2.1 SINGLE WORD ADDRESSING

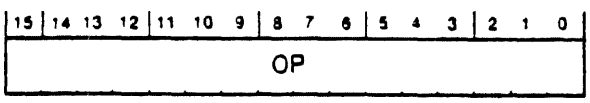
FORMAT 1: Load, Store, Arithmetic, Logic



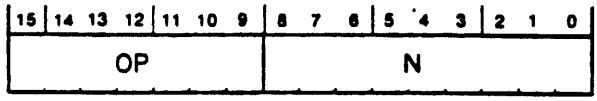
M	Addressing Mode
0xx	Direct
100	Relative (P + 1)
101	Indexed by X
110	Indexed by B
111	Indirect

2.2 SINGLE WORD NON-ADDRESSING

FORMAT 2: Set Overflow, Reset Overflow, Transfer Switches to A Register, Unconditional Skip

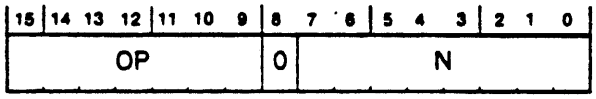


FORMAT 3: Halt, Branch to Processor's Extended Control Store



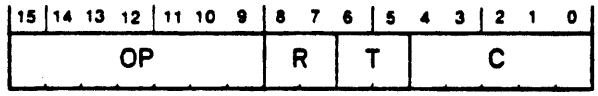
N = Any number, or special use

FORMAT 4: Branch to Control Store, Interpreter Decoder



N = Any number, or special use

FORMAT 5: Shift and Rotate



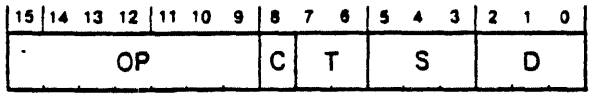
C	Count
00000	0
00001	1
00010	2
•	
•	
11111	31

T	Type
00	Arithmetic shift left
01	Rotate left
10	Arithmetic shift right
11	Logical shift right

R	Register(s) Used
00	B register
01	A register
10	A and B registers
11	Not used*

* Not used with shift and rotate instructions. Reference the double precision format (FORMAT 20) and the double precision instructions in section 5.

FORMAT 6: Register Transfer and Modification



D	Destination Register
000	Undefined (except NOP)
001	A register
010	B register
011	A and B registers
100	X register
101	X and A registers
110	X and B registers
111	X, A, and B registers

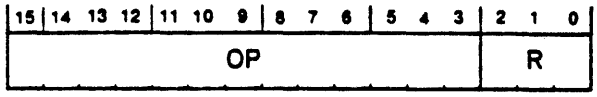
S	Source Register(s)
000	None*
001	A register
010	B register
011	A and B registers
100	X register
101	X and A registers
110	X and B registers
111	X, A, and B registers

T	Type
00	Unmodified transfer
01	Increment and transfer
10	Complement and transfer
11	Decrement and transfer

C	Conditional Execution
0	Transfer unconditionally
1	Transfer only if overflow indicator set

* Used to transfer 0, +1, or -1, as specified by the T field, to the destination register(s).

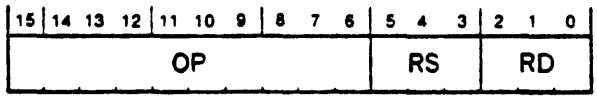
FORMAT 7: Single Register



R Register

000 R0 (A)
001 R1 (B)
010 R2 (X)
011 R3
100 R4
101 R5
110 R6
111 R7

FORMAT 8: Register to Register



RD Destination Register

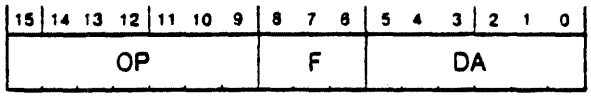
000 R0 (A)
001 R1 (B)
010 R2 (X)
011 R3
100 R4
101 R5
110 R6
111 R7

RS Source Register

000 R0 (A)
001 R1 (B)
010 R2 (X)
011 R3
100 R4
101 R5
110 R6
111 R7

INSTRUCTION FORMATS

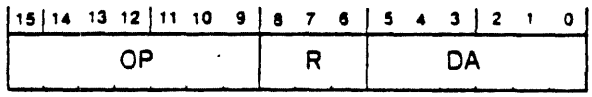
FORMAT 9: External Control



DA	Device Address
000000	0
000001	1
000010	2
.	
.	
111111	63

F	Function Code
000	0
001	1
.	
.	
111	7

FORMAT 10: Input to Register, Output from Register

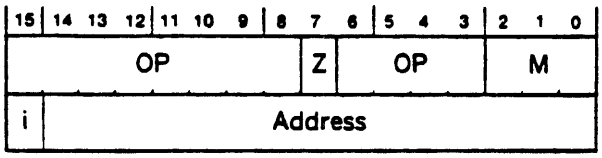


DA	Device Address
000000	0
000001	1
000010	2
.	
.	
111111	63

R	Source or Destination Register
000	Not used
001	A register: input or output
010	B register: input or output
011	A and B registers: input or output
100	Undefined
101	A register: clear and input
110	B register: clear and input
111	A and B registers: clear and input

2.3 DOUBLE WORD ADDRESSING

FORMAT 11: Extended

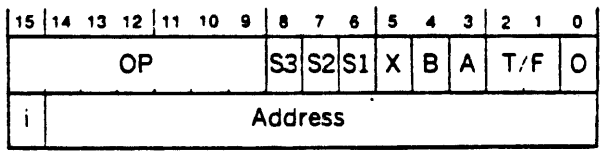


Z	Type of Indexing When i = 1
0	Pre
1	Post

M	Addressing Mode
000	Not used*
001	Undefined
010	Undefined
011	Undefined
100	Relative
101	Indexed by X
110	Indexed by B
111	Direct/indirect

* Code used by immediate instructions.

FORMAT 12: Jump, Jump-and-Mark, Execute, Unconditional Skip

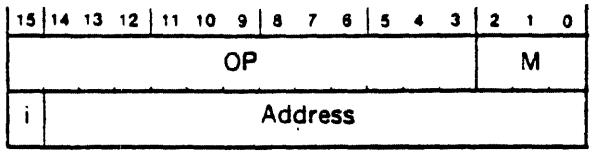


Condition Tested*	
S3	SENSE switch 3 set
S2	SENSE switch 2 set
S1	SENSE switch 1 set
X	X register = 0
B	B register = 0
A	A register = 0
O	Overflow set

T/F	True/False
00	Test for all specified conditions true
01	Test for A register positive and all specified conditions true
10	Test for A register negative and all specified conditions true
11	Test for all specified conditions false

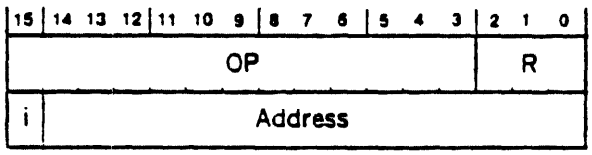
* Multiple conditions may be specified.

FORMAT 13: Indexed Jump



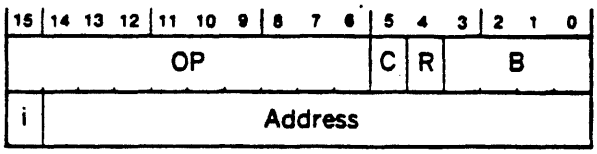
M	Addressing Mode
000	Undefined
001	Undefined
010	Undefined
011	Undefined
100	Undefined
101	Indexed by X
110	Indexed by B
111	Undefined

FORMAT 14: Jump and Set Return



R	Return Register
000	Undefined
001	Undefined
010	Undefined
011	Undefined
100	Undefined
101	X register
110	B register
111	Undefined

FORMAT 15: Bit Test

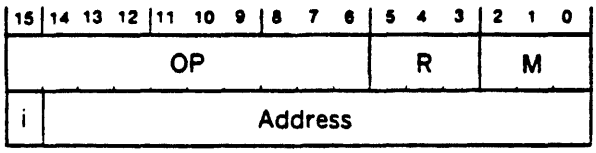


C	Condition Tested
0	Selected bit = 1
1	Selected bit = 0

B	Bit Tested
0000	0
0001	1
0010	2
•	
•	
1111	15

R	Register Selection
0	A register
1	B register

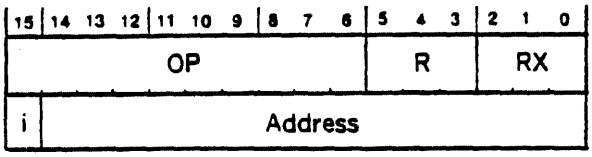
FORMAT 16: Skip if Register Equal



R	Register Selection
000	Undefined
001	A register
010	B register
011	Undefined
100	X register
101	Undefined
110	Undefined
111	Undefined

M	Addressing Mode
000	Undefined
001	Undefined
010	Undefined
011	Undefined
100	Relative
101	Indexed by X
110	Indexed by B
111	Direct/Indirect

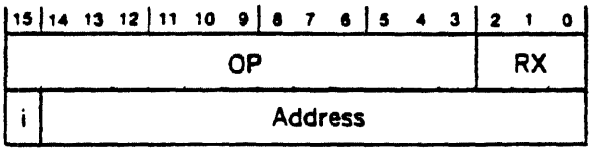
FORMAT 17: Register to Memory



R	Source or Destination Register	
000	R0	(A)
001	R1	(B)
010	R2	(X)
011	R3	
100	R4	
101	R5	
110	R6	
111	R7	

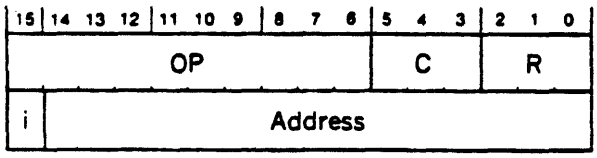
RX	Index Register	
000	No indexing	
001	R1	(B)
010	R2	(X)
011	R3	
100	R4	
101	R5	
110	R6	
111	R7	

FORMAT 18: Byte



RX	Index Register	
000	R0	(A)
001	R1	(B)
010	R2	(X)
011	R3	
100	R4	
101	R5	
110	R6	
111	R7	

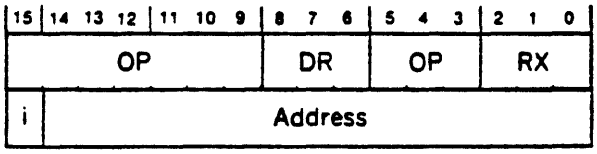
FORMAT 19: Jump If



C	Condition Tested
000	Undefined
001	DJP(V77-800 only)
010	Register = 0
011	Register ≠ 0
100	Register negative
101	Register positive
110	Double precision register = 0
111	Double precision register ≠ 0

R	Register Tested
000	R0 (A)
001	R1 (B)
010	R2
011	R3
100	R4
101	R5
110	R6
111	R7

FORMAT 20: Double Precision

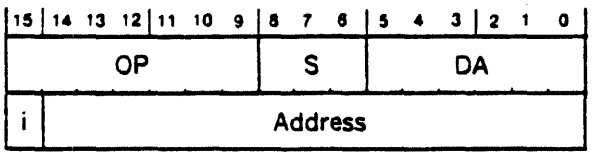


DR	Operand Register
000	Undefined
001	Undefined
010	Undefined
011	Undefined
100	Undefined
101	Undefined
110	Double precision register R0-R1
111	Double precision register R4-R5

RX	Index Register
000	No indexing
001	R1 (B)
010	R2 (X)
011	R3
100	R4
101	R5
110	R6
111	R7

FORMAT 21: Not used

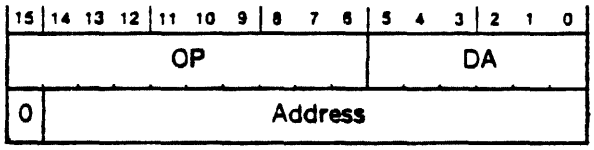
FORMAT 22: Sense



S	Status Line Sensed
000	0
001	1
010	2
.	
.	
111	7

DA	Device Address
000000	0
000001	1
000010	2
.	
.	
111111	63

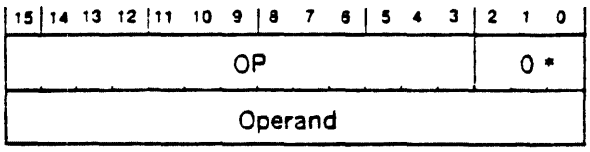
FORMAT 23: Input to Memory, Output from Memory



DA	Device Address
000000	0
000001	1
000010	2
.	
.	
111111	63

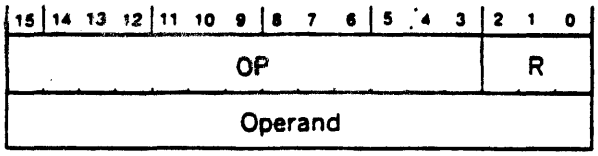
2.4 DOUBLE WORD NON-ADDRESSING

FORMAT 24: Immediate



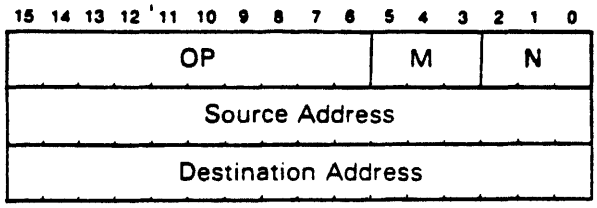
* Codes 001, 010, and 011 are undefined. Codes 100 through 111 are used by extended instructions. Reference FORMAT 11 and section 5.

FORMAT 25: Register Immediate



R	Source or Destination Register
000	R0 (A)
001	R1 (B)
010	R2 (X)
011	R3
100	R4
101	R5
110	R6
111	R7

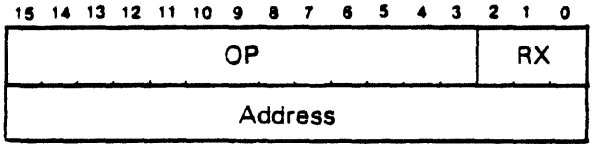
FORMAT 26: Double Word Move



M	Addressing Mode
000	Not used.
001	Not used.
010	Not used.
011	Not used.
100	Both source and destination addresses are direct.
101	Source address is indexed by register R2 (X) and destination address is direct.
110	Source address is direct and destination address is indexed by register R2 (X).
111	Both source and destination addresses are indexed by register R2 (X).

N	Number of Double Words Moved
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

FORMAT 27: Registers Load, Registers Store



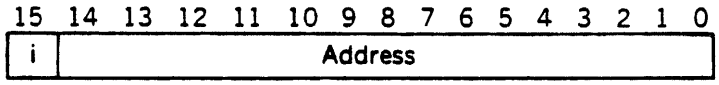
RX	Index Register
000	No indexing
001	R1 (B)
010	R2 (X)
011	R3
100	R4
101	R5
110	R6
111	R7

SECTION 3
DATA FORMATS

Computer words other than instructions may contain either operands or direct/indirect addresses, depending on the instruction or addressing mode in process.

3.1 DIRECT/INDIRECT ADDRESS

When the data word is a direct/indirect address rather than an operand, the format is:

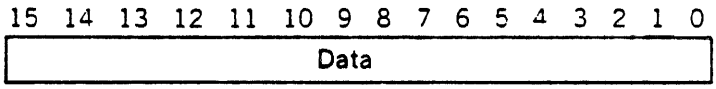


i = 0 word contains operand address
i = 1 word contains indirect address

The i-bit (bit 15) indicates whether the address contained in the word is a direct address (i = 0) or an indirect address (i = 1). Indirect addressing may be extended to several levels before the i-bit is 0, indicating the effective address of the operand (see section 4).

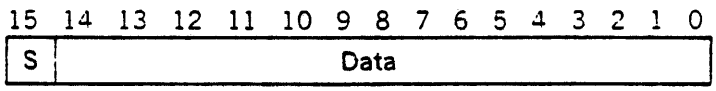
3.2 SINGLE-PRECISION NON-ARITHMETIC DATA

The single-precision non-arithmetic data format consists of one unsigned 16-bit word:



3.3 SINGLE-PRECISION ARITHMETIC DATA

The single-precision arithmetic data format consist of a sign bit and 15 bits of data:



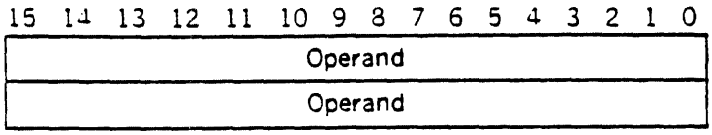
The most significant bit (bit 15) is the sign bit. It is 0 for positive numbers and 1 for negative numbers. The other 15 bits (0-14) contain the data itself.

Negative numbers are represented in twos-complement form. Zero is considered a positive number.

Number values range from a positive of 32.767_{10} (077777_8), to the maximum negative of $32,768_{10}$ (100000_8).

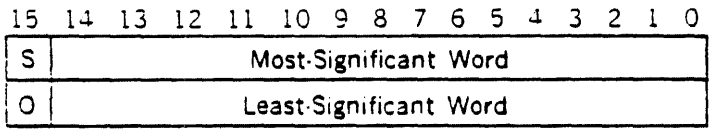
3.4 DOUBLE-PRECISION NON-ARITHMETIC DATA

Double-precision non-arithmetic data consists of two 16-bit unsigned words stored in two consective registers or memory locations:



3.5 DOUBLE-PRECISION ARITHMETIC DATA

Double-precision arithmetic data consists of two 16-bit twos complement words stored in two consective registers or memory locations:



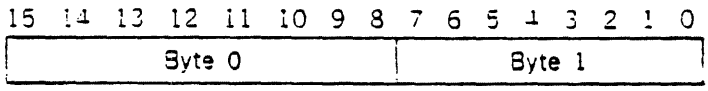
The most significant bit (bit 15) of the first word is the sign bit. Positive numbers are represented in straight binary form with the sign bit a 0. Negative numbers are represented in twos-complement form with the sign bit a 1.

At the beginning of every double-precision arithmetic instruction, bit 15 of the second word must be 0 for all double-precision arithmetic data. At the completion of every double-precision arithmetic instruction, the resulting double-precision arithmetic data will have bit 15 of the second word set to 0.

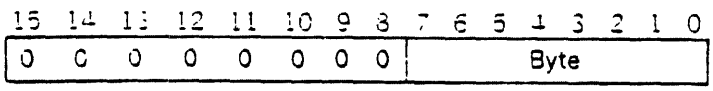
3.6 BYTE DATA

Byte data consists of 8-bit non-arithmetic data stored as two bytes in a memory location or as one byte in the right half of register R0 (A).

The memory location format is:



The register R0 format is:



SECTION 4
ADDRESSING MODES

The V70 series computers feature a number of addressing modes. These modes provide different ways to access a memory location through address modification. The addressing mode is a function of both the instruction type (section 2) and the coding within the instruction (section 5). Only single-word addressing and double-word addressing instructions have addressing modes.

There are four basic addressing modes: direct, indirect, indexed, and relative. The basic modes may be modified and combined; for example, a pre-indexed indirect addressing mode may be specified.

Byte addressing is a special form of indexed and indirect addressing and is discussed separately.

A non-addressing mode, immediate, is included in this section because the operand is accessed from a memory location and not from a register or external source.

Table 4-1 summarizes the addressing modes available with each type of addressing instruction. For the complete instruction formats, refer to section 2.

4.1 IMMEDIATE ADDRESSING

Immediate instructions are classified as double-word non-addressing instructions (section 3) because the second word of the instruction is the operand itself, not an address. No further addressing of memory is required.

Since there is no separate addressing phase, no modification of the address is possible. Direct, indirect, indexed, and relative addressing do not apply to immediate instructions.

"Immediate addressing" is included here because it is one of the options available to the programmer for accessing operands stored in memory. The address of the operand, in this case, is the memory location containing the second word of the instruction. The processor addresses this location when it fetches the immediate instruction for execution.

4.2 DIRECT ADDRESSING

In direct addressing, the address of the operand is contained within the instruction itself.

4.2.1 Direct Addressing with Single-Word Instructions

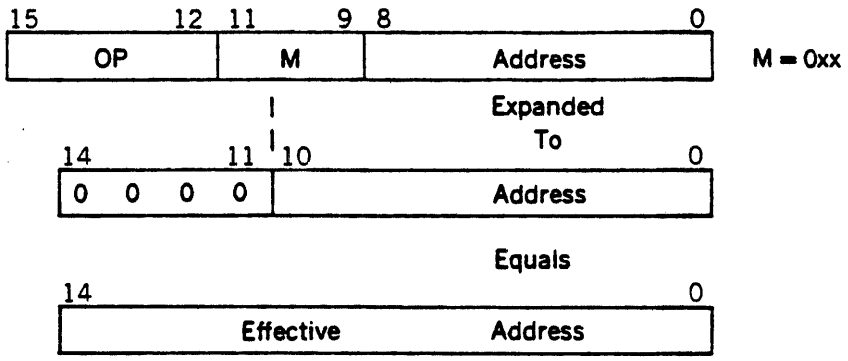
Single-word addressing instructions operate in the direct mode whenever the most significant bit in the M field (bit 11) is 0. (See figures 4-1 and 4-2.)

Table 4-1. Addressing Modes Available With Each Instruction Format

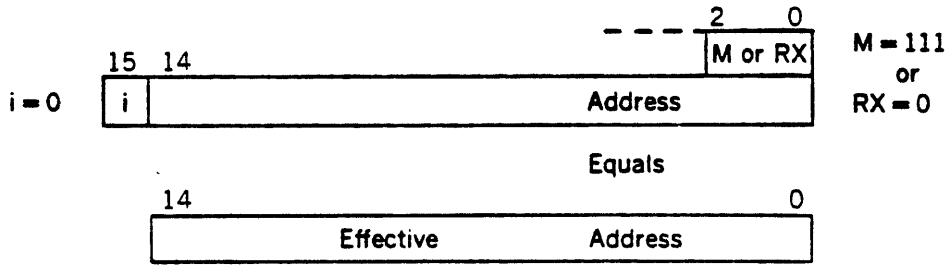
Instruction Class	Format	Instruction	Addressing Mode								
			None	Direct	Indirect	Relative	Pre-Relative Indirect	Post-Relative Indirect	Indexed	Pre-Indexed Indirect	Post-Indexed Indirect
Single Word Addressing	1	Load, Store, Arithmetic, and Logical		X	X	X			X		
Single Word Non-Addressing	2	No-op, Set and Reset Overflow	X								
		Transfer Switches to A Reg	X								
	3	Halt	X								
	4	Branch to Control Store	X								
	5	Shift and Rotate	X								
	6	Register Transfer & Modification	X								
	7	Single Register	X								
	8	Register to Register	X								
	9	I/O Single Word	X								
	10	Register I/O	X								
Double Word Addressing	11	Extended		X	X	X	X	X	X	X	X
	12	Jump, Jump & Mark, Execute		X	X						
	13	Indexed Jump		X	X	X		X	X		X
	14	Jump and Set Return		X	X						
	15	Bit Test		X	X						
	16	Skip If Register Equal		X	X	X		X	X		X
	17	Register to Memory		X	X				X	X	
	18	Byte							X		X
	19	Jump If		X	X						
	20	Double Precision		X	X				X	X	
	21	Floating Point		X	X						
	22	Sense		X	X						
	23	Input/Output from Memory		X							
Double Word Non-Addressing	24	Immediate	X								
	25	Register Immediate	X								

VTII-3597

The remaining two bits of the M field (bits 9 and 10) are combined with the nine bits in the A field to form an 11-bit effective address. The address directs the processor to any location in the first 2,048 words of memory.



a. Single Word Instruction



b. Double Word Instruction

Figure 4-1. Direct Addressing Mode

4.2.2 Direct Addressing with Double-Word Instructions

Double-word addressing instruction with an M field operate in the direct mode whenever an octal 7 is placed in the M field of the instruction and the indirect bit (bit 15) in the second word of the instruction is a 0. Double-word addressing instructions without an M field operate in the direct mode when the indirect bit (bit 15) in the second word is a 0. (See figure 4-1, b.)

The remaining 15 bits of the second word form the effective address. Any location in a full 32K of memory can be addressed directly.

4.3 INDIRECT ADDRESSING

In indirect addressing, the effective address is stored in memory at a location pointed to by the instruction.

4.3.1 Indirect Addressing with Single-Word Instructions

Single-word addressing instructions operate in the indirect mode whenever an octal 7 is placed in the M field. (See figure 4-2, a.)

The nine bits of the address field direct the processor to an address location in the first 512 words of memory. The word stored at that location is either the operand address or another indirect address, depending on the indirect bit of that word. Any location in 32K of memory can be addressed.

4.3.2 Indirect Addressing with Double-Word Instructions

Double-word addressing instructions with an M field operate in the indirect mode whenever an octal 7 is placed in the M field and the indirect bit (bit 15) in the second word is a 1. Double-word addressing instructions without an M field operate in the indirect mode when the indirect bit (bit 15) in the second word is a 1.

The remaining 15 bits of the second word direct the processor to any location in 32K of memory. The word stored at that location is either the effective address or another indirect address, depending on the indirect bit (bit 15) of that word.

4.3.3 Multi-Level Indirect Addressing

The word stored in the memory location specified by an indirect-addressing instruction may itself be an indirect address. When the most-significant bit of the word (bit 15) is a 1, the processor is directed to another memory location specified by the remaining 15 bits of the word.

Multi-level indirect addressing is limited to five levels with single-word instructions, to four levels with double-word instructions (except byte instructions), and to one level with byte instructions. (The V77-200 processor is not limited in number of indirect addressing levels.)

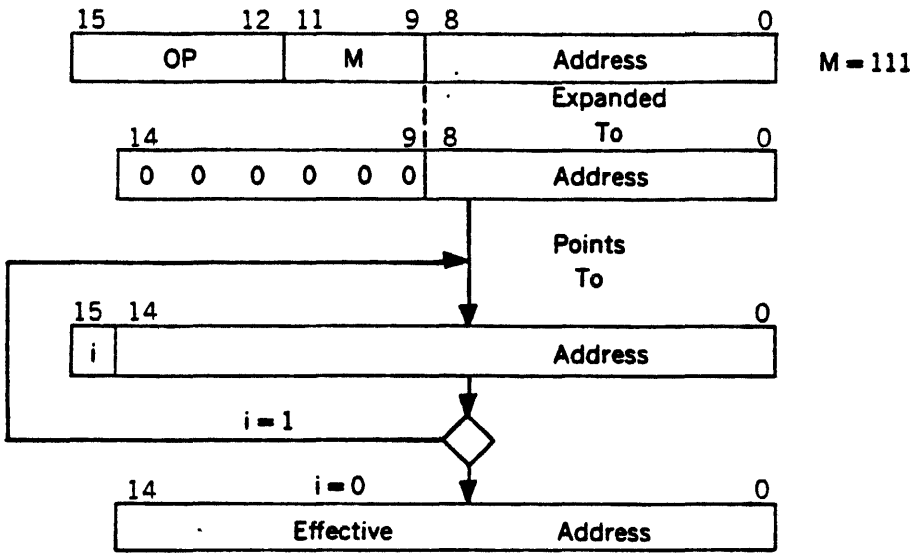
4.3.4 Indirect Combined Modes

The indirect addressing mode can be combined with either the relative or indexed mode in double-word addressing instructions. Indirect addressing is specified when the indirect-bit (bit 15) in the second word of the instruction is a 1.

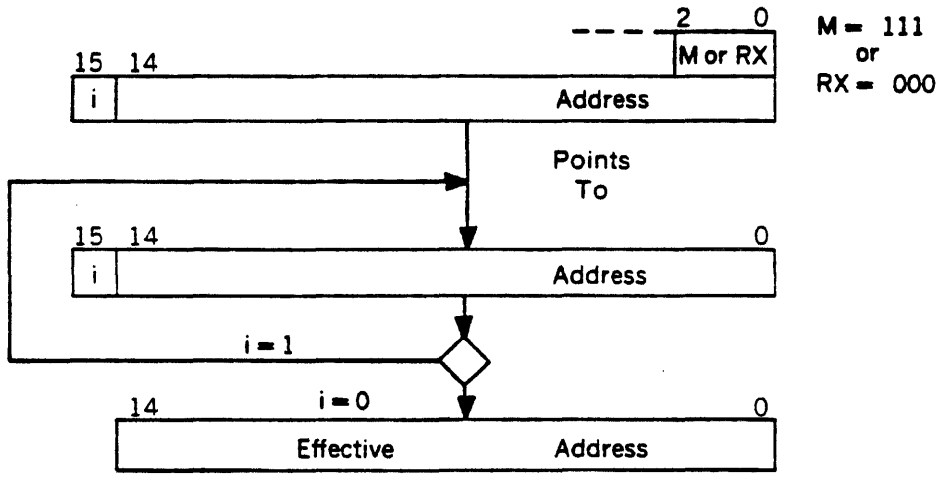
Combined modes may include pre-relative indirect, post-relative indirect, pre-indexed indirect, and post-indexed indirect.

4.4 INDEXED ADDRESSING

In the indexed addressing mode, the address contained within the instruction is modified by adding to it the contents of one of the following registers: R1 through R7, B, or X.



a. Single Word Instruction



b. Double Word Instruction

Figure 4-2. Indirect Addressing Mode

4.4.1 Indexing with Single-Word Instructions

Single-word addressing instructions may be indexed with either the X register or B register by inserting an octal 5 or 6 respectively in the M field. (See figure 4-3. a.)

The contents of the address field are added to the contents of the specified register (X or B) to form the address of the operand. Any location in 32K of memory may be addressed.

4.4.2 Indexing with Double-Word Instructions

Double-word addressing instructions may be indexed by placing the appropriate code in the M field or RX field:

M = 5, X register M = 6, B register

RX = 1 through 7, registers R1 through R7

4.4.2.1 Indexed (i = 0)

When the indirect bit (i-bit) in the second word is 0, the contents of the specified indexing register are added to the base address in the second word to form the effective address. (See figure 4-3, b.)

Any location in 32K of memory may be addressed.

4.4.2.2 Pre-Indexed Indirect

When the indirect bit (i-bit) is 1 and the Z-bit (bit 7) within the operation code is 0, pre-indexing is specified.

The contents of the specified indexing register are added to the base address in the second word to form a new base address pointing to the effective address. Multi-level indirect addressing may occur. (See figure 4-4 a.)

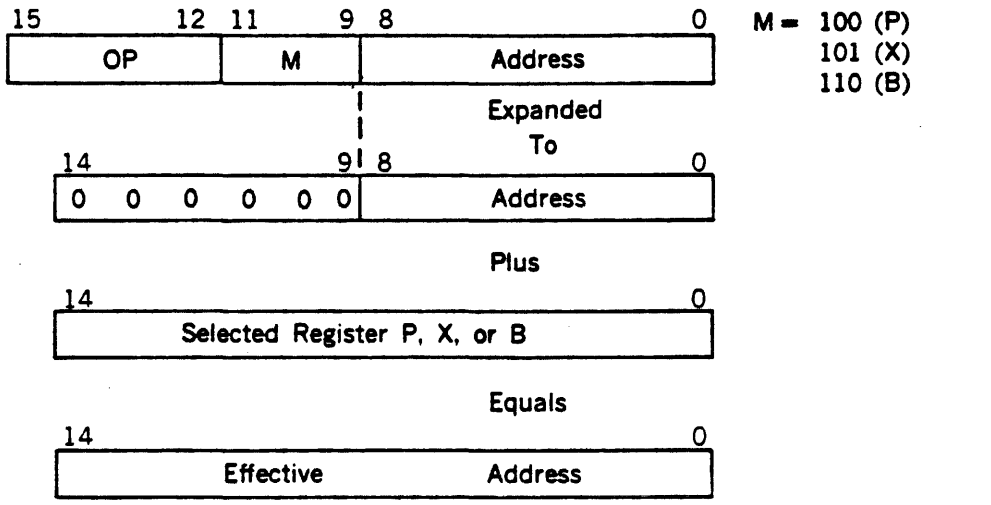
Any location in 32K of memory may be addressed.

4.4.2.3 Post-Indexed Indirect

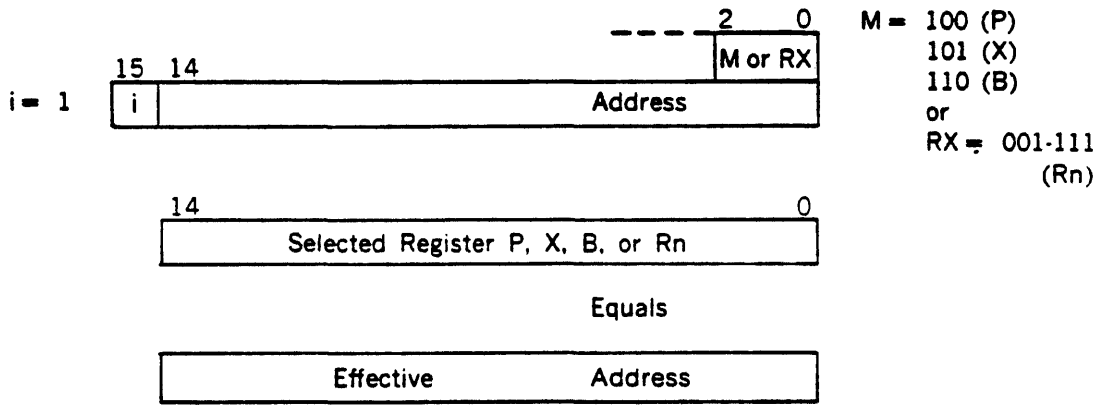
When the indirect bit (i-bit) is 1 and the Z-bit (bit 7) within the operation code is 1, post-indexing is specified.

The base address in the second word points to a new base address in memory. After all multi-level indirect addressing steps are complete, the final base address is added to the contents of the specified register to form the effective address. (See figure 4-4, b.)

Any location in 32K of memory may be addressed.



a. Single Word Instruction



b. Double Word Instruction, i = 0

Figure 4-3. Indexed/Relative Addressing Mode

4.5 RELATIVE ADDRESSING

In relative addressing, the address contained within the instruction is modified by adding to it the contents of the program counter (P register).

4.5.1 Relative Addressing with Single-Word Instructions

Single-word addressing instructions operate in the relative mode whenever an octal 4 is placed in the M field (See figure 4-3, a.)

The contents of the A field are added to the contents of the program counter (P register) to form the effective address. Any of the first 512 locations following the current instruction may be addressed.

4.5.2 Relative Addressing with Double-Word Instructions

Double-word addressing instructions operate in the relative mode whenever an octal 4 is placed in the M field of the instruciton. Note that the P register contains the address of the second word of the instruction (first word address plus 1).

4.5.2.1 Relative (i = 0)

When the indirect bit (i-bit) in the second word is 0, the contents of the program counter (P register) are added to the base address in the second word to form the effective address. (See figure 4-3, b.)

Any location in 32K of memory may be addressed.

4.5.2.2 Pre-Relative Indirect

When the indirect bit (i-bit) is 1 and the P-bit (bit 7) within the operation code is 0, pre-relative addressing is specified.

The contents of the program counter (P register) are added to the base address in the second word to form a new base address pointing to the effective address. Multi-level indirect addressing may occur. (See figure 4-4, a.)

Any location in 32K of memory may be addressed.

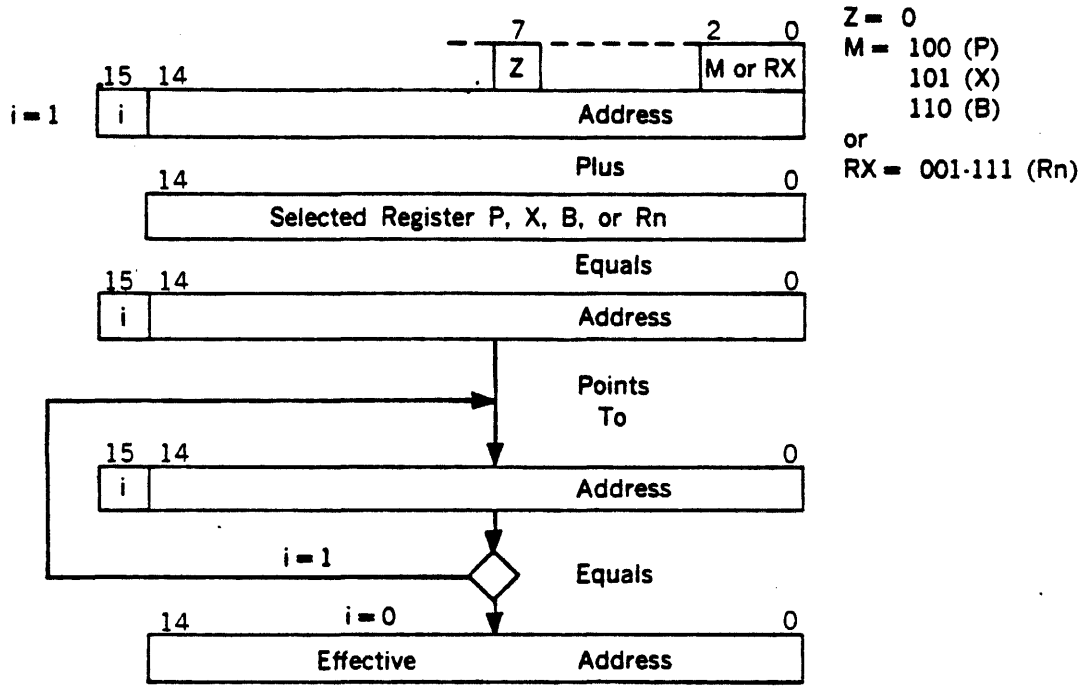
4.5.2.3 Post Relative Indirect

When the indirect bit (i-bit) is 1 and the Z-bit (bit 7) within the operation code is 1, post-relative addressing is specified. The base address in the second word points to a new base address in memory. After all multi-level indirect addressing steps are complete, the final base address is added to the contents of the P register to form the effective address. (See figure 4-4, b.)

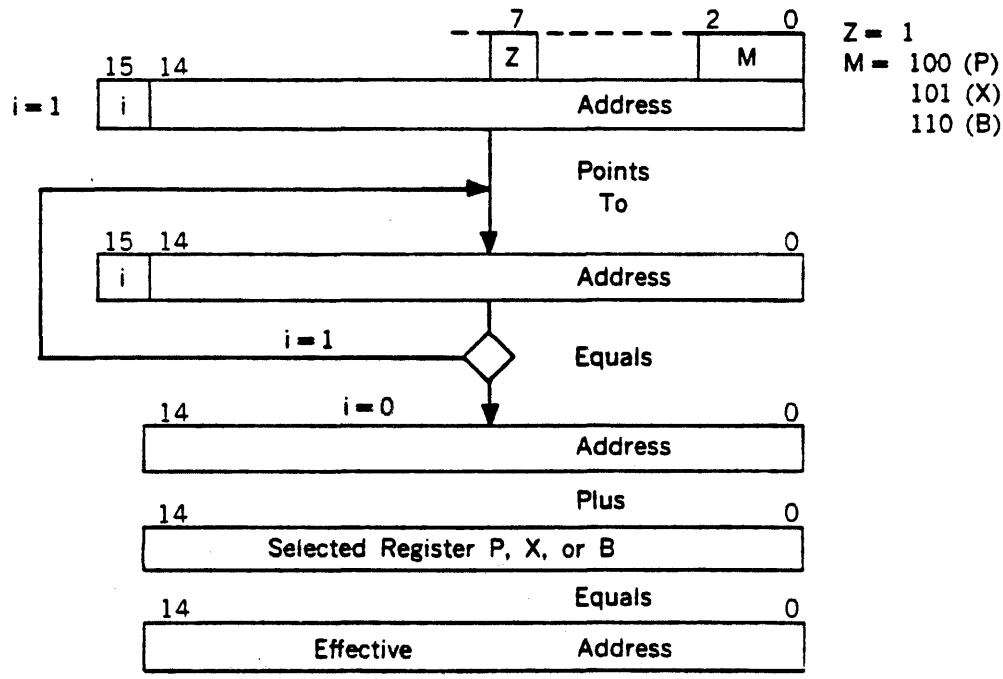
Any location in 32K of memory may be addressed.

4.6 BYTE ADDRESSING

Byte addressing is used in conjunction with the two byte instructions. Use of the byte instructions permit a maximum of 64K byte to be addressed. Any location in 32K of memory may be addressed.



a. Pre-Indexed/Pre-Relative Indirect



b. Post-Indexed/Post-Relative Indirect

Figure 4-4. Indexed/Relative Indirect Addressing Mode

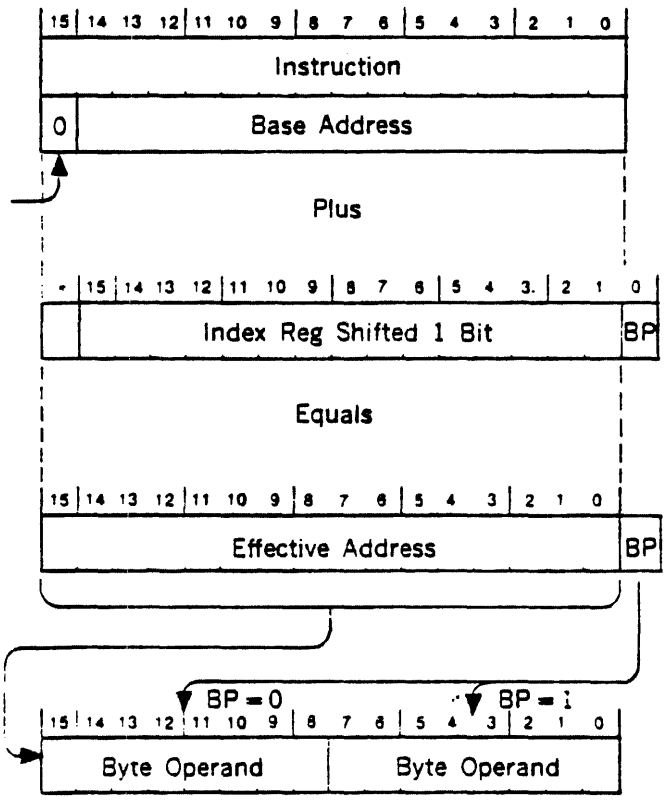
4.6.1 Byte Indexed Mode

When the indirect bit (i-bit) is 0, byte indexed mode is specified. The base address is contained in the least-significant 15 bits of the second word. The base address is added to the contents of the index register shifted arithmetically (sign extended) one bit to the right to form the effective address of the word containing the byte operand. (See figure 4-5.)

The least-significant bit (bit 0) of the index register becomes, when shifted, the byte pointer (BP). The byte pointer selects the byte operand within the addressed word. When BP = 0, the left byte (bits 8 through 15) is selected. When BP = 1, the right byte (bits 0 through 7) is selected.

4.6.2 Byte Indexed Indirect Mode

When the indirect bit (i-bit) is 1, the byte indexed indirect mode is specified. The address contained in the least-significant 15 bits of the second word is an indirect address. The contents of that location is the base address. As in the indexed mode above, the base address is added to the shifted index register to form the effective address of the word containing the byte operand. (See figure 4-6.)

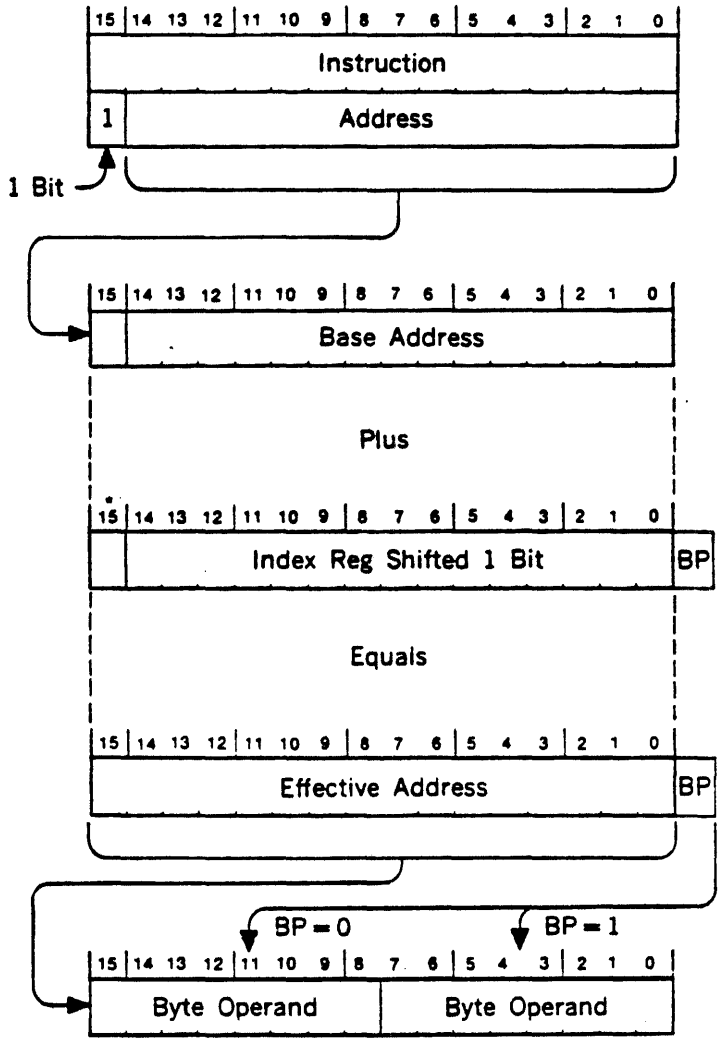


* After the shift to the right, the content of original bit position 15 remains unchanged.

Figure 4-5. Byte Addressing, Indexed Mode

Again, as in the indexed mode, the byte pointer (BP) selects the byte operand within the addressed word.

Only one level of indirect addressing is permitted in byte instructions.



* After the shift to the right, the content of original bit position 15 remains unchanged.

Figure 4-6. Byte Addressing, Indexed-Indirect Mode

SECTION 5
INSTRUCTION SET

Detailed descriptions of the V77 series instructions are contained in this section. V77-800 standard extensions, which are instructions available only with V77-800 computers, are also included. Not included, however, are the floating point processor instructions, since they are not part of the basic V77 series architecture. Descriptions of these instructions are provided in the appropriate floating point processor functional analysis and servicing manuals (V77-600 and V77-800 computers only).

The V77 series instructions are divided into the following functional instruction groups:

- Load/Store
- Arithmetic
- Logic
- Shift/Rotation
- Register Transfer/Modification
- Jump
- Jump-and-Mark
- Special Jump and Skip
- Execute
- Control
- I/O
- Register to Memory*
- Byte*
- Jump If*
- Register Immediate*
- Register to Register*
- Single Register*
- Double Precision*
- V77-800 Standard Extensions*

*The instruction groups marked by an asterisk (*) use the eight registers R0 through R7 in data handling and addressing operations. The remaining groups use the A, B, and X registers.

This section provides a functional description and identifies the format, addressing modes, and registers altered for each instruction in the instruction set. Used in conjunction with

section 2 (instruction formats) and section 4 (addressing modes), a complete description may be formed.

Not all of the instructions defined here are available on all V70 series computer models. The system reference manual for each model identifies the instructions available to that model.

A number of binary instruction codes are not used. They are undefined as to the operation which is performed if they are executed. Those codes in the instruction set which are identified as "undefined" should not be used in programming.

Appendix A contains a list of the instructions arranged alphabetically by mnemonic and indexed to the page where the instruction is defined. Following the alphabetical list is a numerical list by octal code.

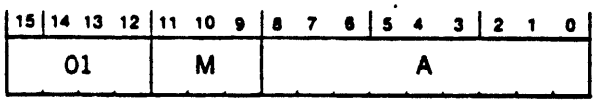
5.1 LOAD/STORE INSTRUCTIONS

This group consists of the instructions for loading registers from memory or for storing the contents of registers in memory. Subgroups of these instructions permit such loading or storing in normal, extended, or immediate formats.

Each of the extended instructions has two numbers in the operation code field. The first number is used when pre-indexed indirect or pre-relative indirect addressing is specified. The second number is used when post-indexed indirect or post-relative indirect addressing is specified. Either number may be used in any other addressing mode.

Mnemonic	Instruction
LDA	Load A register
LDAE	Load A register extended
LDAI	Load A register immediate
LDB	Load B register
LDBE	Load B register extended
LDBI	Load B register immediate
LDX	Load X register
LDXE	Load X register extended
LDXI	Load X register immediate
STA	Store A register
STAE	Store A register extended
STAI	Store A register immediate
STB	Store B register
STBE	Store B register extended
STBI	Store B register immediate
STX	Store X register
STXE	Store X register extended
STXI	Store X register immediate

LDA — Load A Register

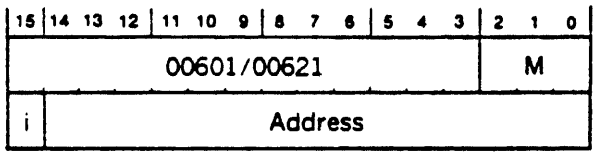


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Loads the contents of the effective address into the A register.

LDAE — Load A Register Extended

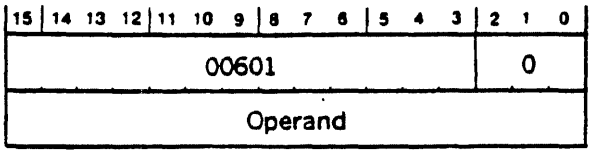


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, port-relative indirect.

Description: Loads the contents of the effective address into the A register.

LDAI — Load A Register Immediate

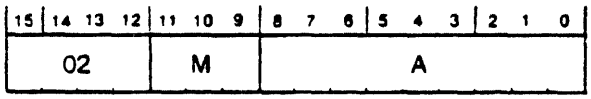


Format: 24

Addressing: None

Description: Loads the contents of the operand field into the A register.

LDB — Load B Register

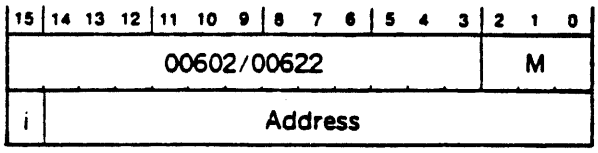


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Loads the contents of the effective address into the B register.

LDBE — Load B Register Extended

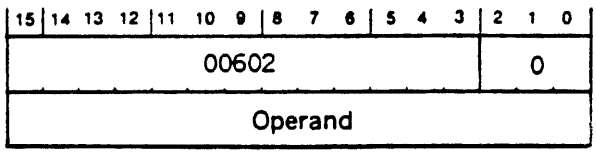


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Loads the contents of the effective memory address into the B register.

LDBI — Load B Register Immediate

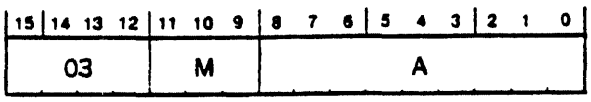


Format: 24

Addressing: None

Description: Loads the contents of the operand field into the B register.

LDX — Load X Register

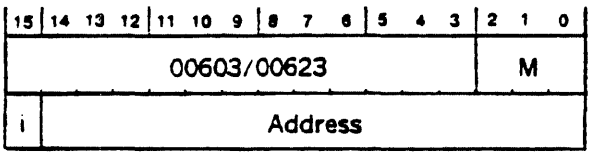


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Loads the contents of the effective memory address into the X register.

LDXE — Load X Register Extended

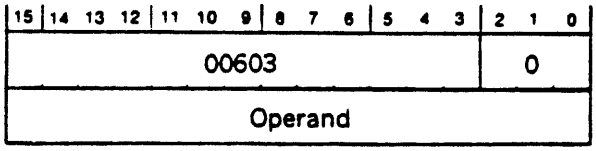


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Loads the contents of the effective memory address into the X register.

LDXI **Load X Register Immediate**

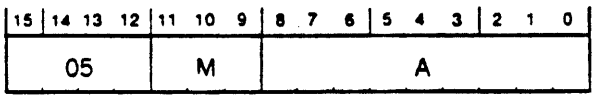


Format: 24

Addressing: None

Description: Loads the contents of the operand field into the X register.

STA **Store A Register**

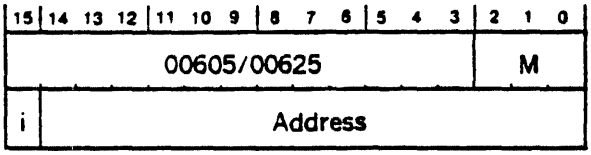


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Stores the contents of the A register into the effective memory location.

STAE — Store A Register Extended

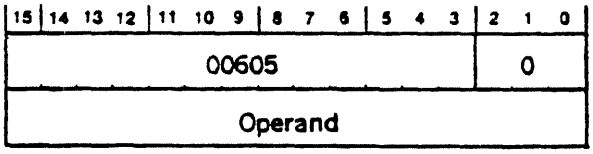


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Stores the contents of the A register into the effective memory location.

STAI — Store A Register Immediate

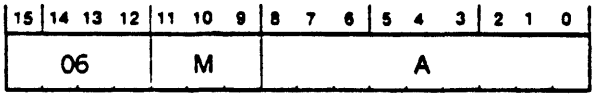


Format: 24

Addressing: None

Description: Stores the contents of the A register into the operand field.

STB **Store B Register**

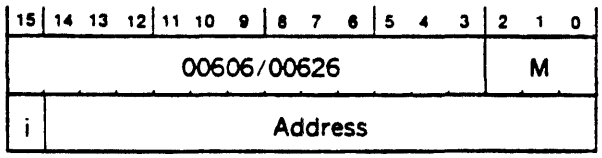


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Stores the contents of the B register into the effective memory location.

STBE **Store B Register Extended**

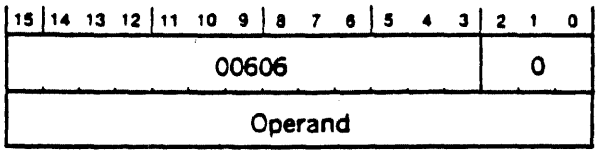


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect.

Description: Stores the contents of the B register into the effective memory location.

STBI **Store B Register Immediate**

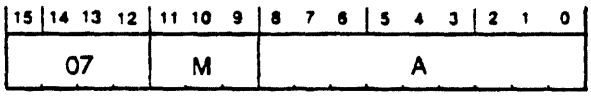


Format: 24

Addressing: None

Description: Stores the contents of the B register into the operand field.

STX **Store X Register**

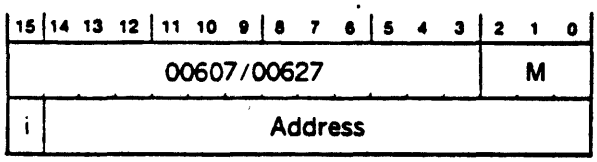


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Stores the contents of the X register into the effective memory location.

STXE — Store X Register Extended

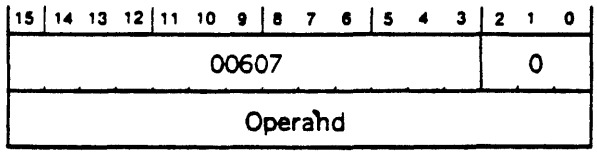


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Stores the contents of the X register into the effective memory location.

STXI — Store X Register Immediate



Format: 24

Addressing: None

Description: Stores the contents of the X register into the operand field.

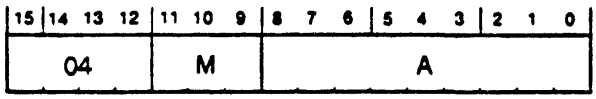
5.2 ARITHMETIC INSTRUCTIONS

This group consists of instructions for incrementing the contents of a memory location and for performing the arithmetic functions of addition, subtraction, multiplication, and division. Subgroups of these instructions permit such operations in normal, extended, or immediate formats.

Each of the extended instructions has two numbers in the operation code field. The first number is used when pre-indexed indirect or pre-relative indirect addressing is specified. The second number is used when post-indexed indirect or post-relative indirect addressing is specified. Either number may be used in any other addressing mode.

Mnemonic	Instruction
INR	Increment memory and replace
INRE	Increment memory and replace extended
INRI	Increment and replace immediate
ADD	Add memory to A register
ADDE	Add to A register extended
ADDI	Add to A register immediate
SUB	Subtract memory from A register
SUBE	Subtract from A register extended
SUBI	Subtract from A register immediate
MUL	Multiply
MULE	Multiply extended
MULI	Multiply immediate
DIV	Divide
DIVE	Divide extended
DIVI	Divide immediate

INR — Increment Memory and Replace

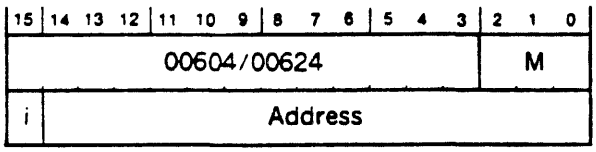


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Increments by one the contents of the effective memory address. Sets the overflow indicator (OF) if the maximum positive number (077777_8) is exceeded. The value in the memory address is then negative (100000_8).

INRE — Increment Memory and Replace Extended

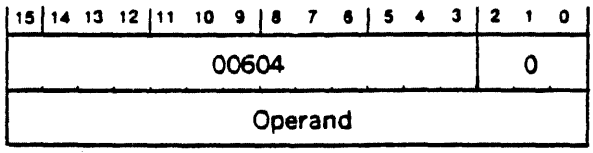


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect.

Description: Increments by one the contents of the effective memory address. Sets the overflow indicator (OF) if the maximum postive number (077777_8) is exceeded. The value in the memory address is then negative (100000_8).

INRI — Increment and Replace Immediate

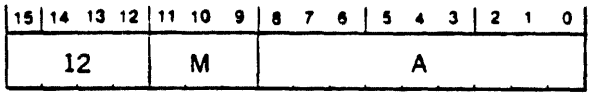


Format: 24

Addressing: None

Description: Increments by one the operand in the second word. Sets the overflow indicator (OF) is the maximum positive number (077777_8) is exceeded. The value of the operand in the second word is then negative (100000_8).

ADD — Add Memory to A Register

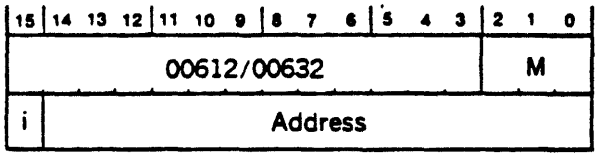


Format: 1

Addressing: Direct, indirect, indexed. relative

Description: Adds the contents of the effective memory address to the contents of the A register and places the sum into the A register. The overflow indicator (OF) is set if the sign bits of the two operands are equal and the sum has the opposite sign.

ADDE **Add to A Register Extended**

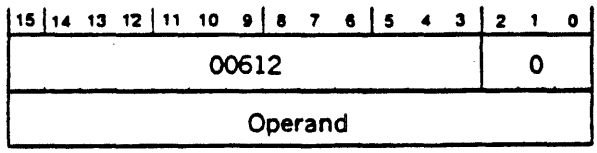


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect.

Description: Adds the contents of the effective memory address to the contents of the A register and places the sum into the A register. The overflow indicator (OF) is set if the sign bits of the two operands are equal and the sum has the opposite sign.

ADDI **Add to A Register Immediate**

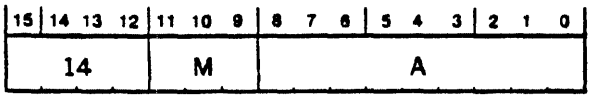


Format: 24

Addressing: None

Description: Adds the operand field to the contents of the A register and places the sum into the A register. The overflow indicator (OF) is set if the sign bits of the two operands are equal and the sum has the opposite sign.

SUB — Subtract Memory from A Register

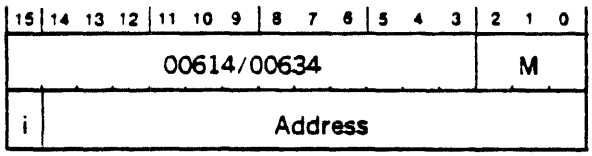


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Subtracts the contents of the effective memory address from the contents of the A register and places the difference into the A register. The overflow indicator (OF) is set if the sign bits of the two operands are not equal and the difference has the sign of the contents of the effective memory address.

SUBE — Subtract from A Register Extended

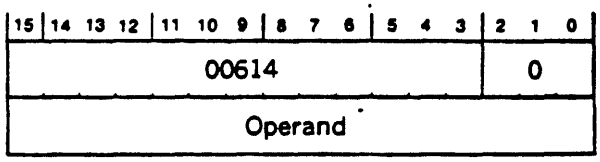


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Subtracts the contents of the effective memory address from the contents of the A register and places the difference into the A register. The overflow indicator (OF) is set if the sign bits of the two operands are not equal and the difference has the sign of the contents of the effective memory address.

SUBI **Subtract from A Register Immediate**

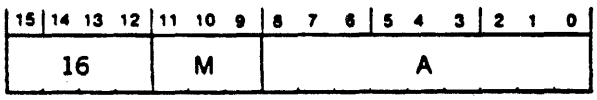


Format: 24

Addressing: None

Description: Subtracts the contents of the operand field from the contents of the A register and places the difference into the A register. The overflow indicator (OF) is set if the sign bits of the two operands were not equal and the difference has the sign of contents of the operand field.

MUL **Multiply**

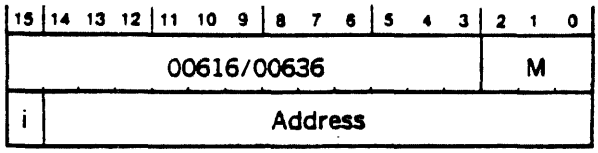


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Multiplies the contents of the effective memory address by the contents of the B register. The contents of the A register are sign extended and added to the double precision product. The double precision result is placed into the A and B registers with the most significant portion in the A register. The sign bit of the A register gives the sign of the result. The sign bit of the B register is set to zero. The overflow indicator (OF) is set if the original contents of the B register and the effective memory address were the greatest possible negative number and the original contents of the A register were positive.

MULE **Multiply Extended**

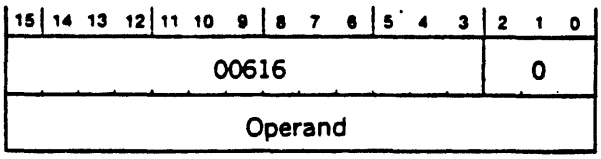


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect.

Description: Multiplies the contents of the effective memory address by the contents of the B register. The contents of the A register are sign extended and added to the double precision product. The double precision result is placed into the A and B registers with the most significant portion in the A register. The sign bit of the A register gives the sign of the result. The sign bit of the B register is set to zero. The overflow indicator (OF) is set if the original contents of the B register and effective memory address were the greatest possible negative number and the original contents of the A register were positive.

MULI **Multiply Immediate**

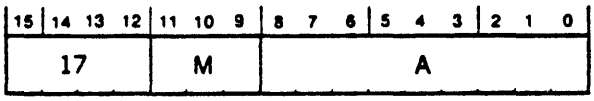


Format: 24

Addressing: None

Description: Multiplies the contents of the operand field by the contents of the B register. The contents of the A register are sign extended and added to the double precision product. The double precision result is placed into the A and B registers. The sign bit of the A register gives the sign of the result. The sign bit of the B register is set to zero. The overflow indicator (OF) is set if the original contents of the B register and the operand field were the greatest possible negative number and the original contents of the A register were positive.

DIV **Divide**

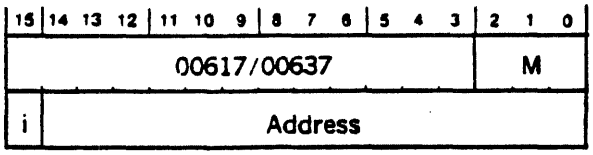


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Divides the double precision contents of the A and B registers by the contents of the effective memory address, and places the signed quotient into the B register and the signed remainder into the A register. The sign of the remainder is equal to the sign of the original contents of the A register or is zero if the remainder is zero. The overflow indicator (OF) is set if the quotient is less than $-2^{15}+1$ or greater than $2^{15}-1$. If OF is set, the result in the A and B registers is undefined.

DIVE **Divide Extended**

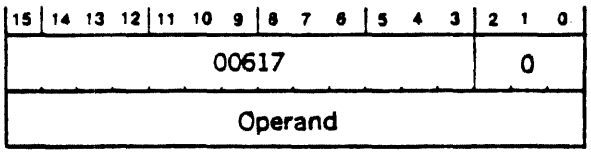


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect.

Description: Divides the double precision contents of the A and B registers by the contents of the effective memory address, and places the signed quotient into the B register and the signed remainder into the A register. The sign of the remainder is equal to the sign of the original contents of the A register or is zero if the remainder is zero. The overflow indicator (OF) is set if the quotient is less than $-2^{15}+1$ or greater than $2^{15}-1$. If OF is set, the result in the A and B register is undefined.

DIVI **Divide Immediate**



Format: 24

Addressing: None

Description: Divides the double precision contents of the A and B registers by the operand field, and places the signed quotient into the B register and the signed remainder into the A register. The sign of the remainder is equal to the sign of the original contents of the A register or is zero if the remainder is zero. The overflow indicator (OF) is set if the quotient is less than $-2^{15}+1$ or greater than $2^{15}-1$. If OF is set the result in the A and B registers is undefined.

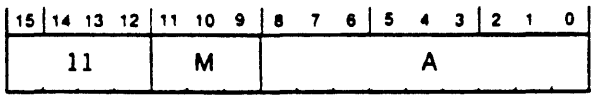
5.3 LOGIC INSTRUCTIONS

This group consists of inclusive-OR, exclusive-OR, and AND instructions and their expanded and immediate-addressing counterparts.

Each of the extended instructions has two numbers in the operation code field. The first number is used when pre-indexed indirect or pre-relative indirect addressing is specified. The second number is used when post-indexed indirect or post-relative indirect addressing is specified. Either number may be used in any other addressing mode.

Mnemonic	Instruction
ORA	Inclusive-OR memory and A register
ORAE	Inclusive-OR extended
ORAI	Inclusive-OR immediate
ERA	Exclusive-OR memory and A register
ERAE	Exclusive-OR extended
ERAI	Exclusive-OR immediate
ANA	AND memory and A register
ANAE	AND extended
ANAI	AND immediate

ORA **Inclusive-OR Memory and A Register**

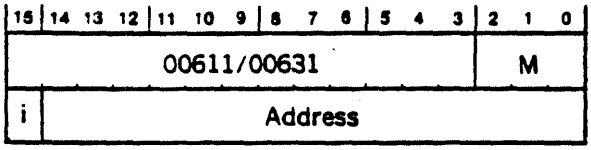


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Performs an inclusive-OR of each bit of the A register with the corresponding bit of the operand located at the effective memory address, and places the result into the A register.

ORAE **Inclusive-OR Extended**

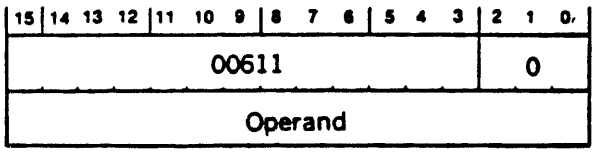


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Performs an inclusive-OR of each bit of the A register with the corresponding bit of the operand located at the effective memory address, and places the result into the A register.

ORAI **Inclusive-OR Immediate**

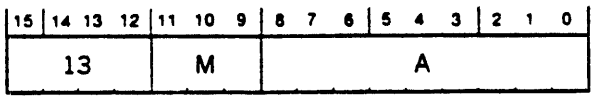


Format: 24

Addressing: None

Description: Performs an inclusive-OR of each bit of the A register with the corresponding bit of the operand field and places the result into the A register.

ERA **Exclusive-OR Memory and A Register**

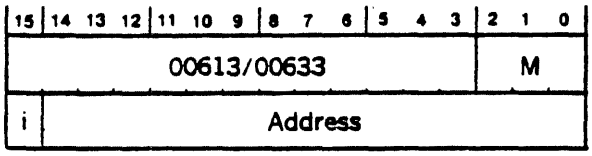


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Performs an exclusive-OR of each bit of the A register with the corresponding bit of the operand located at the effective memory address and places the result into the A register

ERAE Exclusive-OR Extended

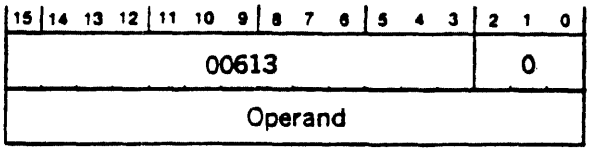


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect.

Description: Performs an exclusive-OR of each bit of the A register with the corresponding bit of the operand located at the effective memory address, and places the result into the A register.

ERAI Exclusive-OR Immediate

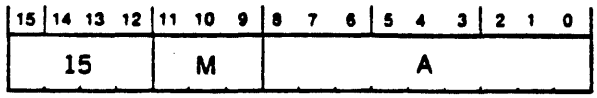


Format: 24

Addressing: None

Description: Performs an exclusive-OR of each bit of the A register with the corresponding bit of the operand field, and places the result into the A register.

ANA **AND Memory and A Register**

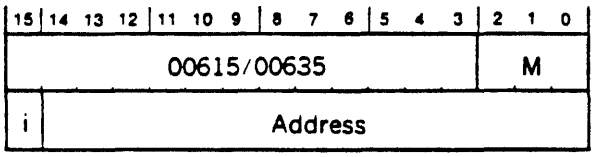


Format: 1

Addressing: Direct, indirect, indexed, relative

Description: Performs an AND of each bit of the A register with the corresponding bit of the operand located at the effective memory address, and places the result into the A register.

ANAE **AND Extended**

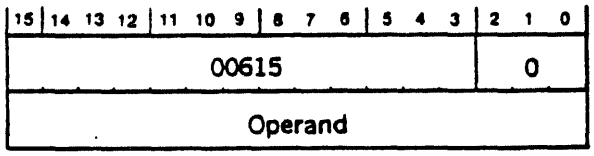


Format: 11

Addressing: Direct, indirect, indexed, relative, pre-indexed indirect, post-indexed indirect, pre-relative indirect, post-relative indirect

Description: Performs an AND of each bit of the A register with the corresponding bit of the operand located at the effective memory address, and places the result into the A register.

ANAI **AND Immediate**



Format: 24

Addressing: None

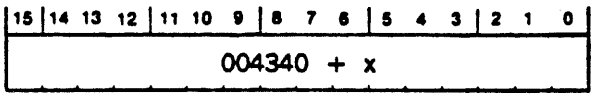
Description: Performs an AND of each bit of the A register with the corresponding bit of the operand field, and places the result into the A register.

5.4 SHIFT/ROTATION INSTRUCTIONS

This group consists of instructions which shift or rotate the contents of registers. In shift instructions, the bits shifted out of the register are lost. In rotation instructions, the bits shifted out one end of a register are loaded one at a time into the opposite end of the register. Shift and rotation instructions are non-addressing.

Mnemonic	Instruction
LSRA	Logical shift right A register
LSRB	Logical shift right B register
LRLA	Logical rotate left A register
LRLB	Logical rotate left B register
LLSR	Long logical shift right
LLRL	Long logical rotate left
ASRA	Arithmetic shift right A register
ASRB	Arithmetic shift right B register
ASLA	Arithmetic shift left A register
ASLB	Arithmetic shift left B register
LASR	Long arithmetic shift right
LASL	Long arithmetic shift left

LSRA Logical Shift Right A Register

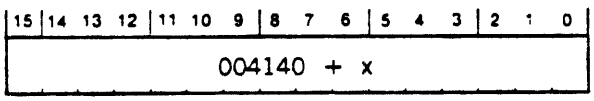


Format: 5

Addressing: None

Description: Shifts the contents of the A register x places (x = 0 to 037_8) to the right and loads the vacated high-order bit(s) with zeros. Information shifted out of the low-order bit(s) is lost.

LSRB Logical Shift Right B Register

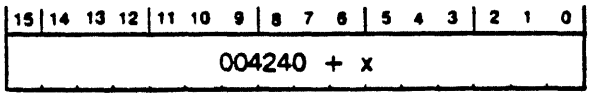


Format: 5

Addressing: None

Description: Shifts the contents of the B register x places (x = 0 to 037_8) to the right and loads the vacated high-order bits(s) with zeros. Information shifted out of the low-order bit(s) is lost.

LRLA **Logical Rotate Left A Register**

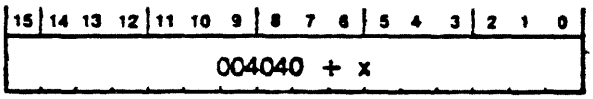


Format: 5

Addressing: None

Description: Rotates the contents of the A register x places (x = 0 to 037_8) to the left. Each bit shift out of bit 15 is shifted into bit 0 during the execution.

LRLB **Logical Rotate Left B Register**

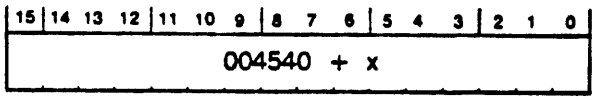


Format: 5

Addressing: None

Description: Rotates the contents of the B register x places (x = 0 to 037_8) to the left. Each bit shifted out of bit 15 is shifted into bit 0 during the execution.

LLSR Long Logical Shift Right

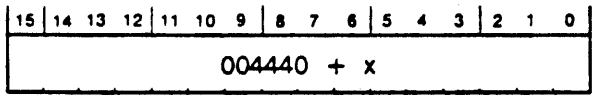


Format: 5

Addressing: None

Description: Shifts the double precision contents of the A and B registers x places ($x = 0$ to 037_8) to the right. Loads the vacated high-order bit(s) of the A register with zeros. Each bit shifted out of bit 0 of the A register is shifted into bit 15 of the B register. Information shifted out of the low-order bit(s) of the B register is lost.

LLRL Long Logical Rotation Left

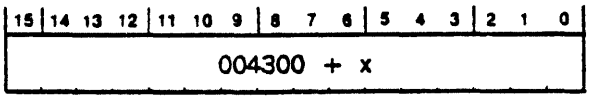


Format: 5

Addressing: None

Description: Rotates the double precision contents of the A and B registers x places ($x = 0$ to 037_8) to the left. Each bit shifted out of bit 15 of the B register is shifted into bit 0 of the A register. Each bit shifted out of bit 15 of the A register is shifted into bit 0 of the B register.

ASRA — Arithmetic Shift Right A Register

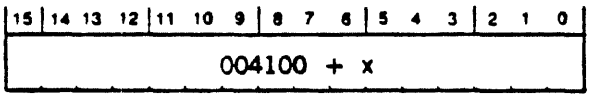


Format: 5

Addressing: None

Description: Shifts the contents of the A register, including the sign bit, x places ($x = 0$ to 037_8) to the right. Loads the vacated high-order bit(s) with the value of the sign bit. Information shifted out of the low-order bit(s) is lost.

ASRB — Arithmetic Shift Right B Register

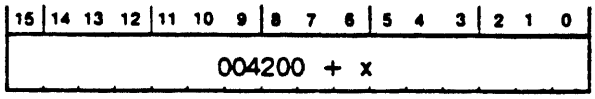


Format: 5

Addressing: None

Description: Shifts the contents of the B register, including the sign bit, x places ($x = 0$ to 037_8) to the right. Loads the vacated high-order bit(s) with the value of the sign bit. Information shifted out of the low-order bit(s) is lost.

ASLA — Arithmetic Shift Left A Register

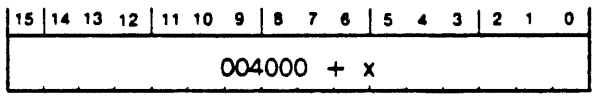


Format: 5

Addressing: None

Description: Shifts the contents of the A register x places (x = 0 to 037_8) to the left. Loads the vacated low-order bit(s) with zeros. The sign bit is not affected. Information shifted out of bit 14 is lost.

ASLB — Arithmetic Shift Left B Register

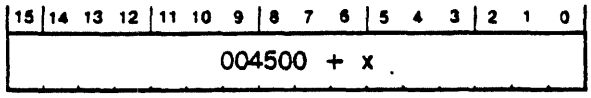


Format: 5

Addressing: None

Description: Shifts the contents of the B register x places (x = to 037_8) to the left. Loads the vacated low-order bit(s) with zeros. The sign bit is not affected. Information shifted out of bit 14 is lost.

LASR Long Arithmetic Shift Right

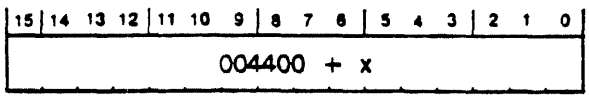


Format: 5

Addressing: None

Description: The double precision contents of the A and B registers are shifted x places ($x = 0$ to 037_8) to the right. Each bit shifted out of bit 0 of the A register is shifted into bit 14 of the B register. The sign bit of the A register (bit 15) is extended x places to the right. The sign bit of the B register remains unchanged. Information shifted out of bit 0 of the B register is lost.

LASL Long Arithmetic Shift Left



Format: 5

Addressing: None

Description: Shifts the double precision contents of the A and B registers x places ($x = 0$ to 037_8) to the left. Each bit shifted out of bit 14 of the B register is shifted into bit 0 of the A register. The sign bit (bit 15) of the B register is unchanged. The sign bit (bit 15) of the A register is unchanged. Information shifted out of bit 14 of the A register is lost.

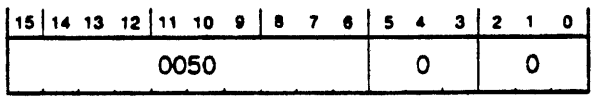
5.5 REGISTER TRANSFER/MODIFICATION INSTRUCTIONS

This group consists of instructions for: unmodified register transfers; incrementing, decrementing, and complementing registers; adjusting the contents of registers with the overflow indicator (OF); and combinations of these operations.

5.5.1 Unmodified Register Transfer Instructions

Mnemonic	Instruction
NOP	No operations
TAB	Transfer A register to B register
TAX	Transfer A register to X register
TBA	Transfer B register to A register
TBX	Transfer B register to X register
TXA	Transfer B register to A register
TXB	Transfer X register to B register
TZA	Transfer zeros to A register
TZB	Transfer zeros to B register
TZX	Transfer zeros to X register

NOP **No Operation**

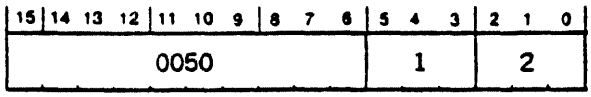


Format: 6

Addressing: None

Description: Waits one cycle. The P register is incremented by one. The A, B, and X registers and memory remain unchanged.

TAB **Transfer A Register to B Register**

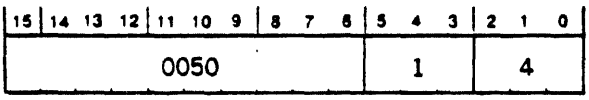


Format: 6

Addressing: None

Description: Transfers the contents of the A register to the B register. The A register is unchanged.

TAX **Transfer A Register to X Register**

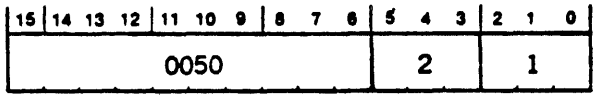


Format: 6

Addressing: None

Description: Transfers the contents of the A register to the X register. The A register is unchanged.

TBA **Transfer B Register to A Register**

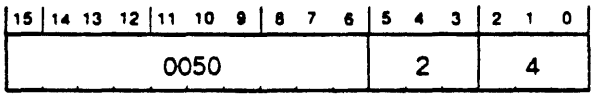


Format: 6

Addressing: None

Description: Transfers the contents of the B register to the A register. The B register is unchanged.

TBX **Transfer B Register to X Register**

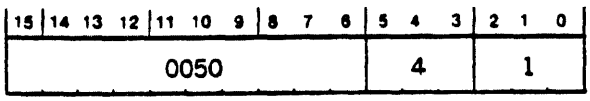


Format: 6

Addressing: None

Description: Transfers the contents of the B register to the X register. The B register is unchanged.

TXA **Transfer X Register to A Register**

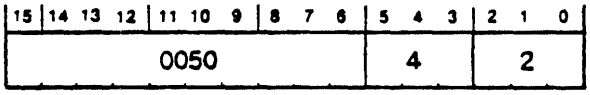


Format: 6

Addressing: None

Description: Transfers the contents of the X register to the A register. The X register is unchanged.

TXB **Transfer X Register to B Register**

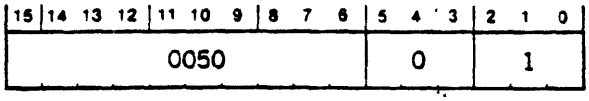


Format: 6

Addressing: None

Description: Transfers the contents of the X register to the B register. The X register is unchanged.

TZA **Transfer Zeros to A Register (Clear A)**

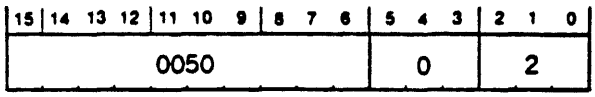


Format: 6

Addressing: None

Description: Transfers zeros to the A register. Clears the A register.

TZB **Transfer Zeros to the B Register (Clear B)**

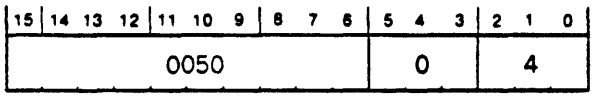


Format: 6

Addressing: None

Description: Transfers zeros to the B register. Clears the B register.

TZX **Transfer Zeros to the X Register (Clear X)**



Format: 6

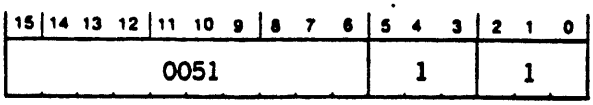
Addressing: None

Description: Transfers zeros to the X register. Clears the X register.

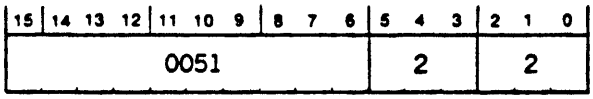
5.5.2 Register Modification Instructions

Mnemonic	Instruction
IAR	Increment A register
IBR	Increment B register
IXR	Increment X register
DAR	Decrement A register
DBR	Decrement B register
DXR	Decrement X register
CPA	Complement A register
CPB	Complement B register
CPX	Complement X register
AOFA	Increment A register if overflow indicator set
AOFB	Increment B register if overflow indicator set
AOFX	Increment X register if overflow indicator set
SOFA	Decrement A register if overflow indicator set
SOFB	Decrement B register if overflow indicator set
SOFX	Decrement X register if overflow indicator set
ICA	Increment cleared A register
ICB	Increment cleared B register
ICX	Increment cleared X register
DCA	Decrement cleared A register
DCB	Decrement cleared B register
DCX	Decrement cleared X register

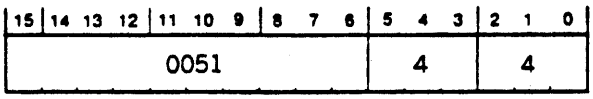
IAR **Increment A Register**



IBR **Increment B Register**



IXR **Increment X Register**

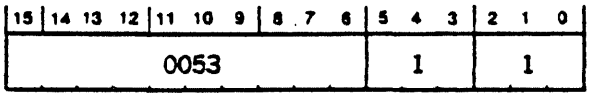


Format: 6

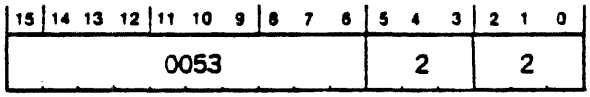
Addressing: None

Description: Increments (by one) the contents of the specified register. Sets the overflow indicator (OF) if the register initially contains the maximum positive number (077777_8), and changes the contents to the maximum negative number (100000_8).

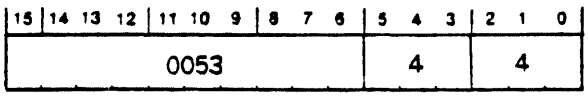
DAR **Decrement A Register**



DBR **Decrement B Register**



DXR **Decrement X Register**

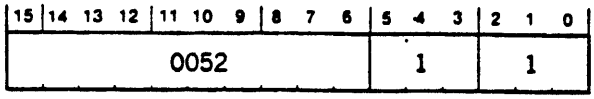


Format: 6

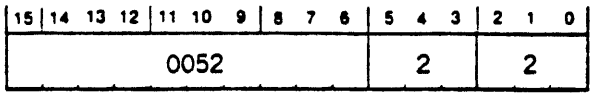
Addressing: None

Description: Decrements (by one) the contents of the specified register. Sets the overflow indicator (OF) if the register initially contains the maximum negative number (100000_8), and changes the contents to the maximum positive number (077777_8).

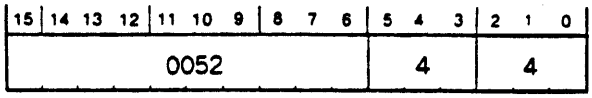
CPA **Complement A Register**



CPB **Complement B Register**



CPX **Complement X Register**

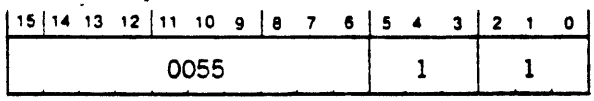


Format: 6

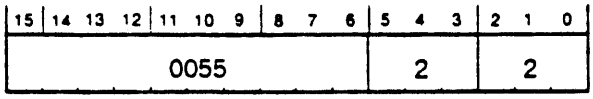
Addressing: None

Description: Ones-complements the contents of the specified register.

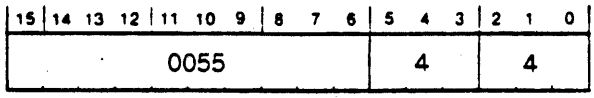
AOFA **Increment A Register if Overflow Indicator Set**



AOFB **Increment B Register if Overflow Indicator Set**



AOFX **Increment X Register if Overflow Indicator Set**



Format: 6

Addressing: None

Description: Adds one to the contents of the specified register only if the overflow indicator (OF) is set. Does not change the setting of OF.

SOFA **Decrement A Register if Overflow Indicator Set**

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0057			1	1

SOFB **Decrement B Register if Overflow Indicator Set**

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0057			2	2

SOFX **Decrement X Register if Overflow Indicator Set**

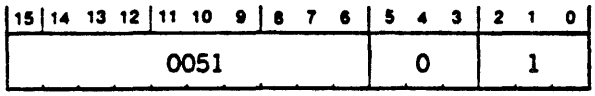
15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0057			4	4

Format: 6

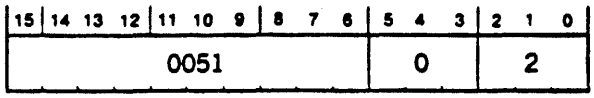
Addressing: None

Description: Subtracts one from the contents of the specified register only if the overflow indicator (OF) is set. Does not change the setting of OF.

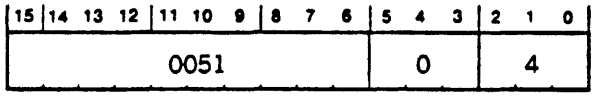
ICA **Increment Cleared A Register**



ICB **Increment Cleared B Register**



ICX **Increment Cleared X Register**

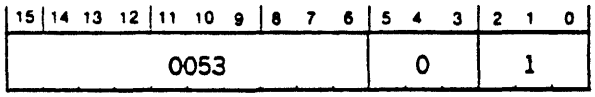


Format: 6

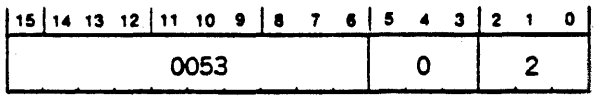
Addressing: None

Description: Replaces the contents of the specified register with +1 (000001_8).

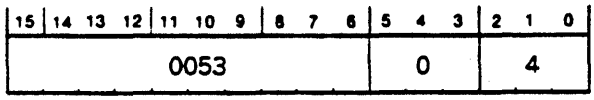
DCA **Decrement Cleared A Register**



DCB **Decrement Cleared B Register**



DCX **Decrement Cleared X Register**



Format: 6

Addressing: None

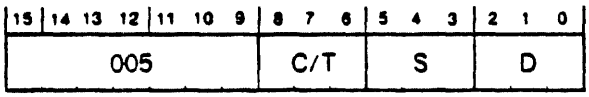
Description: Replaces the contents of the specified register with -1 (177777).

5.5.3 Combined Register Transfer/Modification Instructions

These instructions are used to specify combinations of the register transfer and modification instructions to perform simultaneous multiple operations. [The combined conditions are established in the variable field of the DAS assembler statement when the program is written.].

Mnemonic	Instruction
MERG	Merge source to destination registers
INCR	Increment source to destination registers
DECR	Decrement source to destination registers
COMP	Complement source to destination registers
ZERO	Zero (clear) registers

MERG **Merge source to Destination Registers**



Format: 6

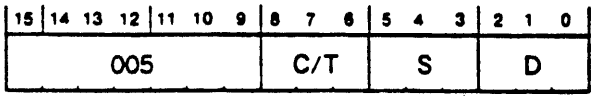
Addressing: None

Description: Transfers the inclusive-OR of the contents of the source register(s) to the destination register(s).

If the C/T field contains 0, the instruction is performed unconditionally. If the field contains 4, the instruction is performed only if the overflow indicator (OF) is set.

The no-operation and transfer instructions of section 5.5.1 are a sub-set of the instructions which can be generated using MERG. MERG instructions for which no destination register is specified (D = 0) are undefined, except for the no-operation instruction. If no source register is specified (S = 0), the contents of each specified register are replaced by zero.

INCR **Increment Source to Desination Registers**



Format: 6

Addressing: None

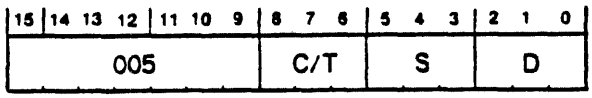
Description: Adds one to the inclusive-OR of the contents of the source register(s) and places the result into the destination register(s).

If the C/T field contains 1, the instruction is performed unconditionally. The overflow indicator is set if the inclusive-OR of the source register(s) is the maximum positive number (077777_8). The maximum negative number (100000_8) is then stored in the destination register(s).

If the C/T field contains 5, the instruction is performed only if the overflow indicator (OF) is set. OF remains unchanged.

The increment instructions of section 5.5.2 are a sub-set of the instruction which can be generated by using INCR. INCR instructions for which no destination register is specified (D = 0) are undefined. If no source register is specified (S = 0), the contents of each destination register are replaced by +1 (000001_8).

DECR **Decrement Source to Destination Registers**



Format: 6

Addressing: None

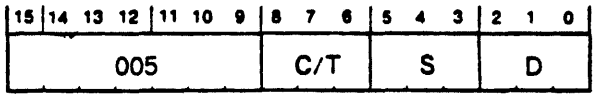
Description: Subtracts one from the inclusive-OR of the contents of the source register(s) and places the results in the destination register(s).

If the C/T field contains 3, the instruction is performed unconditionally. The overflow indicator (OF) is set if the inclusive-OR of the contents of the source registers equals the maximum negative number (100000_8). The maximum positive number (077777_8) is then stored in the destination register(s).

If the C/T field contains 7, the instruction is performed only if the overflow indicator (OF) is set. OF remains unchanged.

The decrement instructions of section 5.5.2 are a sub-set of the instructions which can be generated by using DECR. DECR instructions for which no destination register is specified (D = 0) are undefined. If no source register is specified, the contents of each destination register are replaced by -1 (177777_8).

COMP **Complement Source to Destination Registers**



Format: 6

Addressing: None

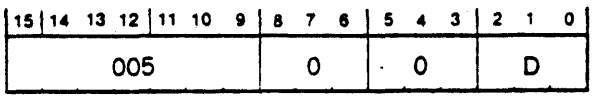
Description: Ones-complements the inclusive-OR of the contents of the source register(s) and places the result in the destination register(s).

If the C/T field contains 2, the instruction is performed unconditionally.

If the C/T field contains 6, the instruction is performed only if the overflow indicator (OF) is set.

The complement instructions of section 5.5.2 are a sub-set of the instructions which can be generated using COMP. COMP instructions for which no source register (S = 0) or no destination register (D = 0) is specified are undefined.

ZERO **Zero (Clear) Registers**



Format: 6

Addressing: None

Description: Clears the destination registers.

The transfer zero instructions of section 5.5.1 are a sub-set of the instructions which can be generated using ZERO. If no destination register is specified (D = 0); the resulting instructions is a no-operation instruction.

5.6 JUMP INSTRUCTIONS

This group consists of instructions that direct the program to a nonsequential address for execution of the instruction located there. They neither mark the location (as do the jump-and-mark instructions) nor do they bring the program back to the main program (as do the execution instructions).

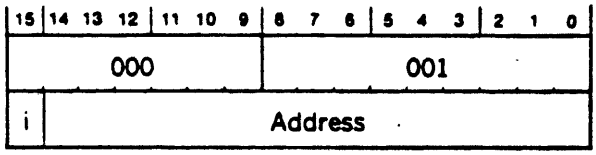
In these instructions, the effective jump address is the address of the next instruction to be executed if the jump condition is met. The program then continues to execute instructions following the jump address.

If the jump condition is not met, the program executes the instruction immediately following the second word of the jump instruction.

The jump instructions in this section are functionally similar to the "jump-if" instructions of section 5.14 but differ in both format and type of jump condition that can be specified. The "jump-if" instructions must not be confused with the instructions of this sections, the JIF instruction in particular.

Mnemonic	Instruction
JMP	Jump unconditionally
JOF	Jump if overflow indicator set
JOFN	Jump if overflow indicator not set
JAP	Jump if A register positive
JAN	Jump if A register negative
JAZ	Jump if A register zero
JBZ	Jump if B register zero
JXZ	Jump if X register zero
JANZ	Jump if A register not zero
JBNZ	Jump if B register not zero
JXNZ	Jump if X register not zero
JSS1	Jump if SENSE switch 1 set
JSS2	Jump if SENSE switch 2 set
JSS3	Jump if SENSE switch 3 set
JS1N	Jump if SENSE switch 1 not set
JS2N	Jump if SENSE switch 2 not set
JS3N	Jump if SENSE switch 3 not set
JIF	Jump if condition(s) met

JMP — Jump Unconditionally

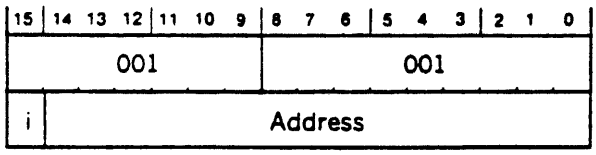


Format: 12

Addressing: Direct, indirect

Description: Jumps unconditionally to the instruction at the effective jump address and executes it next.

JOF — Jump If Overflow Indicator Set



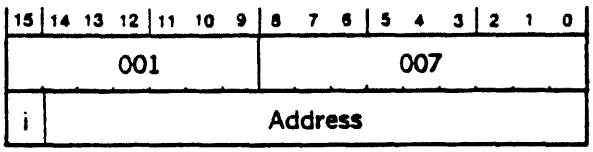
Format: 12

Addressing: Direct, indirect

Description: If the overflow indicator (OF) is set, jumps to the instruction at the effective jump address and executes it next. Resets the overflow indicator.

If the overflow indicator is not set, executes the next instruction in sequence.

JOFN — Jump If Overflow Indicator Not Set

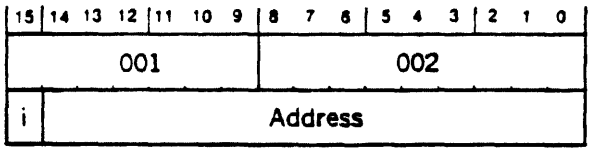


Format: 12

Addressing: Direct, indirect

Description: If the overflow indicator (OF) is not set, jumps to the instruction at the effective jump address and executes it next. If the overflow indicator is set, executes the next instruction in sequence.

JAP — Jump If A Register Positive

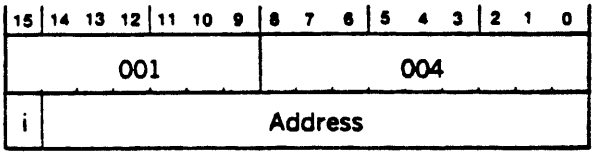


Format: 12

Addressing: Direct, indirect

Description: If the A register contains a positive value (including zero), jumps to the instruction at the effective jump address and executes it next. If the A register contains a negative value, executes the next instruction in sequence.

JAN — Jump If A Register Negative

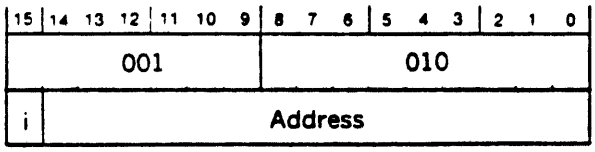


Format: 12

Addressing: Direct, indirect

Description: If the A register contains a negative value, jumps to the instruction at the effective jump address and executes it next. If the A register contains a positive value (including zero), executes the next instruction in sequence.

JAZ — Jump If A Register Zero

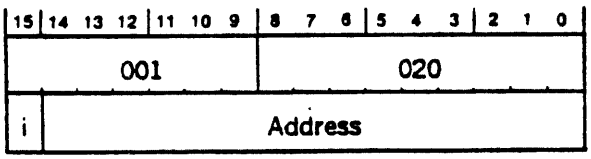


Format: 12

Addressing: Direct, indirect

Description: If the A register contains zero, jumps to the instruction at the effective jump address and executes it next. If the A register does not contain zero, executes the next instruction in sequence.

JBZ Jump If B Register Zero

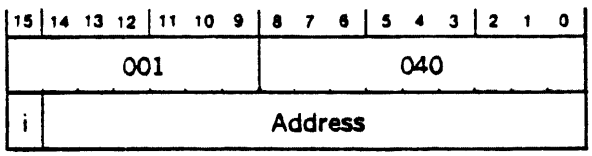


Format: 12

Addressing: Direct, indirect

Description: If the B register contains zero, jumps to the instruction at the effective jump address and executes it next. If the B register does not contain zero, executes the next instruction in sequence.

JXZ Jump If X Register Zero

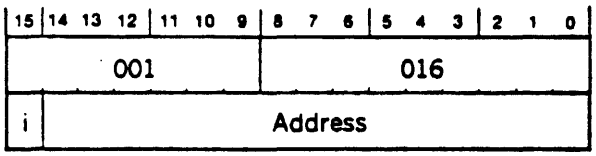


Format: 12

Addressing: Direct, indirect

Description: If the X register contains zero, jumps to the instruction at the effective jump address and executes it next. If the X register does not contain zero, executes the next instruction in sequence.

JANZ — Jump If A Register Not Zero

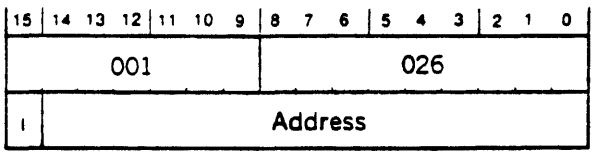


Format: 12

Addressing: Direct, indirect

Description: If the A register is not zero, jumps to the instruction at the effective jump address and executes it next. If the A register is zero, executes the next instruction in sequence.

JBNZ — Jump If B Register Not Zero

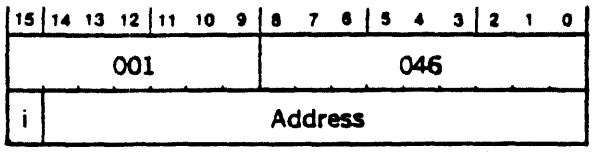


Format: 12

Addressing: Direct, indirect

Description: If the B register is not zero, jumps to the instruction at the efffective jump address and executes it next. If the B register is zero, executes the next instruction in sequence.

JXNZ — Jump If X Register Not Zero

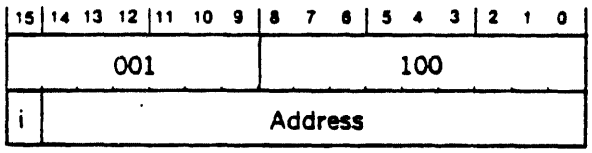


Format: 12

Addressing: Direct, indirect

Description: If the X register is not zero, jumps to the instruction at the effective jump address and executes it next. If the X register is zero, executes the next instruction in sequence.

JSS1 — Jump If SENSE Switch 1 Set

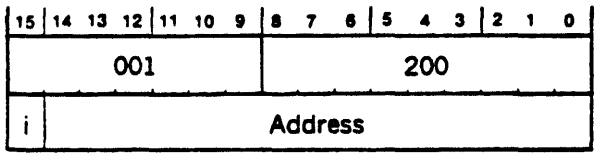


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 1 is set, jumps to the instruction at the effective jump address and executes it next. If SENSE switch 1 is not set, executes the next instruction in sequence.

JSS2 Jump If SENSE Switch 2 Set

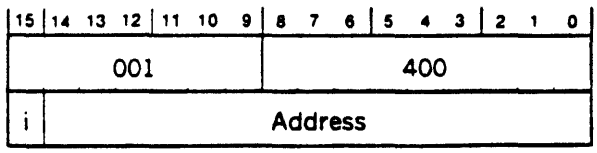


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 2 is set, jumps to the instruction at the effective jump address and executes it next. If SENSE switch 2 is not set, executes the next instruction in sequence.

JSS3 Jump If SENSE Switch 3 Set

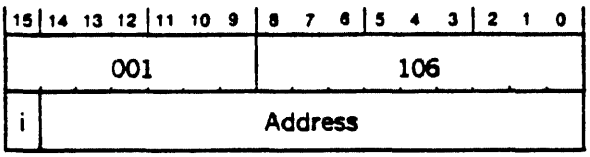


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 3 is set, jumps to the instruction at the effective jump address and executes it next. If SENSE switch 3 is not set, executes the next instruction in sequence.

JS1N Jump If SENSE Switch 1 Not Set

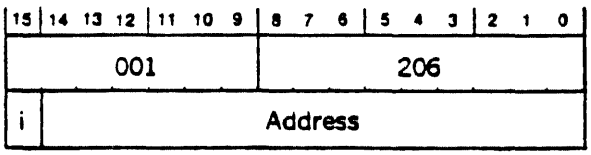


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 1 is not set, jumps to the instruction at the effective jump address and executes it next. If SENSE switch 1 is set, executes the next instruction in sequence.

JS2N Jump If SENSE Switch 2 Not Set

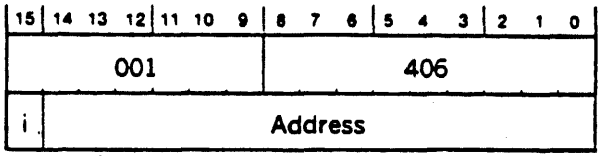


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 2 is not set, jumps to the instruction at the effective jump address and executes it next. If SENSE switch 2 is set, executes the next instruction in sequence.

JS3N — Jump If SENSE Switch 3 Not Set

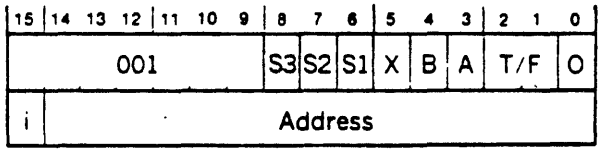


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 3 is not set, jumps to the instruction at the effective jump address and executes it next. If SENSE switch 3 is set, executes the next instruction in sequence.

JIF — Jump If Condition(s) Met



Format: 12

Addressing: Direct, indirect

Description: If all the conditions specified by bits 0 through 8 are met, jumps to the instruction at the effective jump address and executes it next. The condition specified by setting combinations of bits 0 through 8 is the AND of each bit specification. JIF instructions in which bits 2 and 3 are set and bit 1 is reset are undefined. If bits 0 through 8 contain 006_8, the instruction is an unconditional skip instruction (see section 5.10). If any specified jump condition is not met, JIF executes the next instruction in sequence.

5.7 JUMP-AND-MARK INSTRUCTIONS

This group consists of instructions that direct the program to a nonsequential mark address, store the contents of the P register at that address, and execute the instruction folloiwng that address.

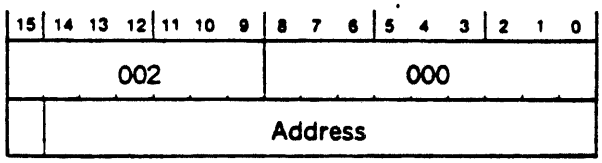
The effective mark address is the address where the contents of the P register are stored if the jump-and-mark condition is met. The instruction next to be executed is located in the effective mark address plus one. The program then continues to execute instructions following the one in the mark address plus one.

If the jump-and-mark condition is not met, the program executes the instruction following the jump-and-mark instruction.

Note: The contents of the P register, when stored, equal the address of the first word of the jump-and-mark instruction plus 2.

Mnemonic	Instruction
JMPM	Jump and mark unconditionally
JOFM	Jump and mark if overflow indicator set
JOFNM	Jump and mark if overflow indicator not set
JAPM	Jump and mark if A register positive
JANM	Jump and mark if A register negative
JAZM	Jump and mark if A register zero
JBZM	Jump and mark if B register zero
JXZM	Jump and mark if X register zero
JANZM	Jump and mark if A register not zero
JBNZM	Jump and mark if B register not zero
JXNZM	Jump and mark if X register not zero
JS1M	Jump and mark if SENSE switch 1 set
JS2M	Jump and mark if SENSE switch 2 set
JS3M	Jump and mark if SENSE switch 3 set
JS1NM	Jump and mark if SENSE switch 1 not set
JS2NM	Jump and mark if SENSE switch 2 not set
JS3NM	Jump and mark if SENSE switch 3 not set
JIFM	Jump and mark if condition(s) met

JMPM **Jump and Mark Unconditionally**

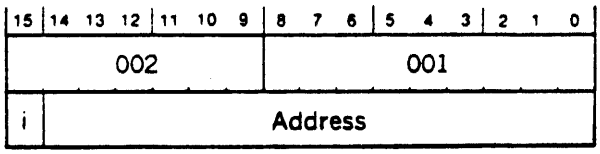


Format: 12

Addressing: Direct, indirect

Description: Stores the contents of the P register at the effective mark address and jumps unconditionally to the instruction at the mark address plus one and executes it next.

JOFM **Jump and Mark If Overflow Indicator Set**

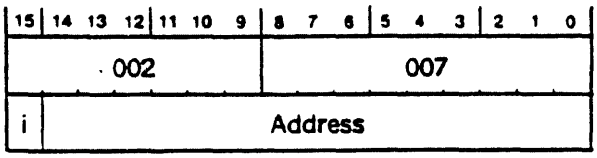


Format: 12

Addressing: Direct, indirect

Description: If the overflow indicator (OF) is set, stores the contents of the P register at the effective mark address and resets the overflow indicator. Jumps to the instruction at the effective mark address plus one and executes it next. If the overflow indicator is not set, executes the next instruction in sequence.

JOFNM **Jump and Mark If Overflow Indicator Not Set**

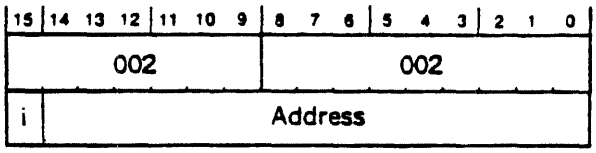


Format: 12

Addressing: Direct, indirect

Description: If the overflow indicator (OF) is not set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the overflow indicator is set, executes the next instruction in sequence. Does not reset the overflow indicator.

JAPM **Jump and Mark If A Register Positive**

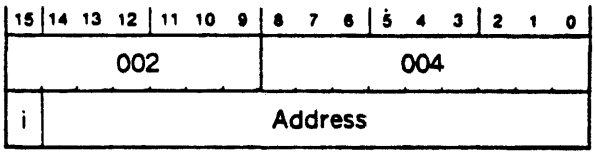


Format: 12

Addressing: Direct, indirect

Description: If the A register contains a positive value (including zero), stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the A register contains a negative value, executes the next instruction in sequence.

JANM Jump and Mark If A Register Negative

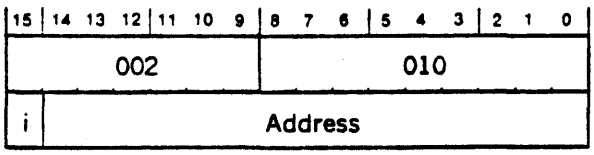


Format: 12

Addressing: Direct, indirect

Description: If the A register contains a negative value, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the P register contains a positive value (including zero), executes the next instruction in sequence.

JAZM Jump and Mark If A Register Zero

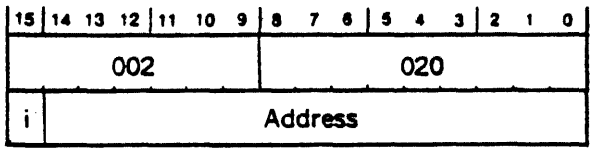


Format: 12

Addressing: Direct, indirect

Description: If the A register contains zero, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the A register does not contain zero, executes the next instruction in sequence.

JBZM Jump and Mark If B Register Zero

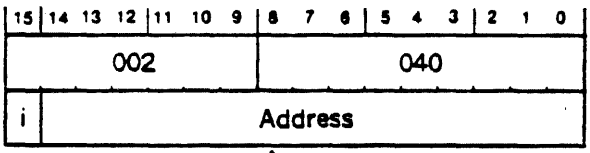


Format: 12

Addressing: Direct, indirect

Description: If the B register contains zero, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the B register does not contain zero, executes the next instruction in sequence.

JXZM Jump and Mark If X Register Zero

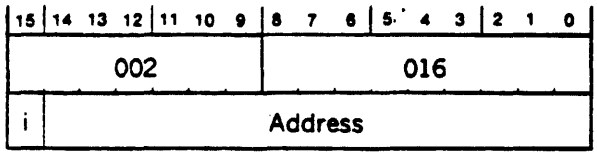


Format: 12

Addressing: Direct, indirect

Description: If the X register contains zero, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the X register does not contain zero, executes the next instruction in sequence.

JANZM Jump and Mark If A Register Not Zero

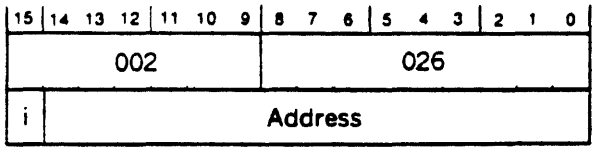


Format: 12

Addressing: Direct, indirect

Description: If the A register is not zero, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the A register is zero, executes the next instruction in sequence.

JBNZM Jump and Mark If B Register Not Zero

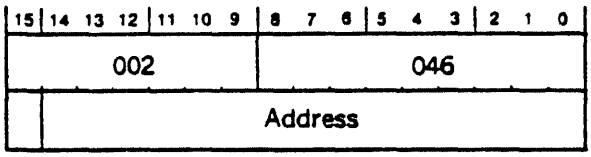


Format: 12

Addressing: Direct, indirect

Description: If the B register is not zero, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the B register is zero, executes the next instruction in sequence.

JXNZM Jump and Mark If X Register Not Zero

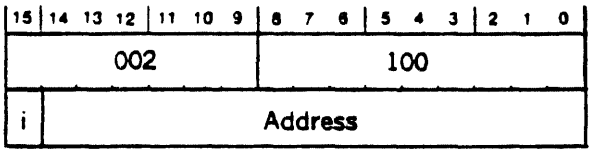


Format: 12

Addressing: Direct, indirect

Description: If the X register is not zero, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If the X register is zero, executes the next instruction in sequence.

JS1M Jump and Mark If SENSE Switch 1 Set

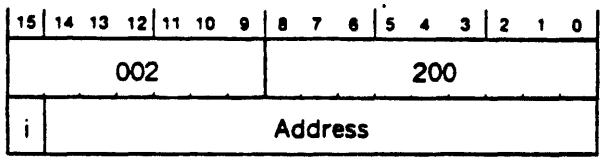


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 1 is set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If SENSE switch 1 is not set, executes the next instruction in sequence.

JS2M Jump and Mark If SENSE Switch 2 Set

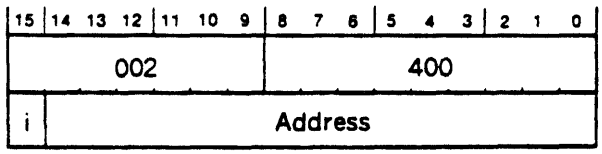


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 2 is set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If SENSE switch 2 is not set, executes the next instruction in sequence.

JS3M Jump and Mark If SENSE Switch 3 Set

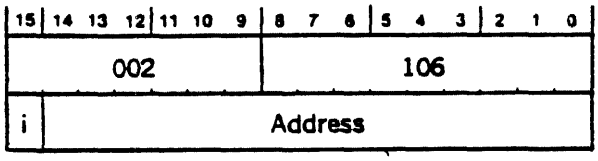


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 3 is set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If SENSE switch 3 is not set, executes the next instruction in sequence.

JS1NM Jump and Mark If SENSE Switch 1 Not Set

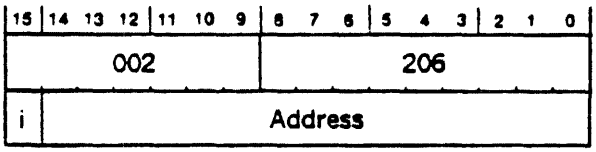


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 1 is not set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If SENSE switch 1 is set, executes the next instruction in sequence.

JS2NM Jump and Mark If SENSE Switch 2 Not Set

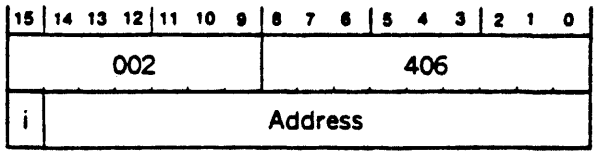


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 2 is not set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one end executes it next. If SENSE switch 2 is set, executes the next instruction in sequence.

JS3NM Jump and Mark If SENSE Switch 3 Not Set

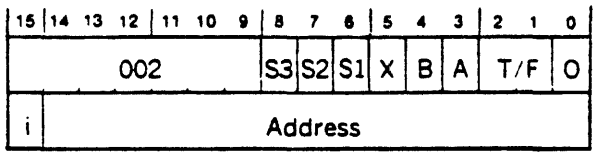


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 3 is not set, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. If SENSE switch 3 is set, executes the next instruction in sequence.

JIFM Jump and Mark If Condition(s) Met



Format: 12

Addressing: Direct, indirect

Description: Analogous to JIF. If all the conditions specified are met, stores the contents of the P register at the effective mark address, jumps to the instruction at the effective mark address plus one and executes it next. The condition specified by setting combinations of bits 0 through 8 is the AND of each bit specification. JIFM instructions in which bits 2 and 3 are set and bit 1 is reset or bits 0 through 8 contain 006_8 are undefined. If any of the jump conditions is not met, executes the next instruction in sequence.

5.8 SPECIAL JUMP AND SKIP INSTRUCTIONS

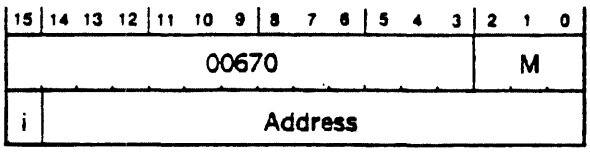
This group consists of four special instructions that direct the program to a non-sequential address for the execution of the instruction there; but they neither mark the location (as do the jump-and-mark instructions) nor do they return to the main program sequence (as do the execute instructions).

In these instructions, the effective jump or skip address is the address of the next instruction to be executed if the instruction is unconditional or if the jump condition is met. The program then continues to execute instructions following the jump or skip address.

For the two conditional instructions, if the condition is not met, the program executes the instruction immediately following the conditional instruction.

Mnemonic	Instruction
IJMP	Indexed jump
JSR	Jump and set return register
BT	Bit test
SRE	Skip if register equal

IJMP **Indexed Jump**

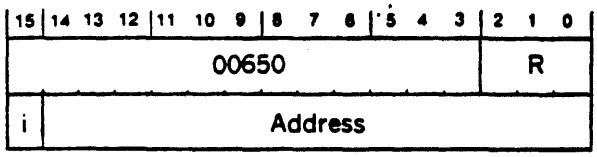


Format: 13

Addressing: Indexed, post-indexed indirect

Description: Jumps unconditionally to the instruction at the effective jump address and executes it next.

JSR — Jump and Set Return in Indexing Register

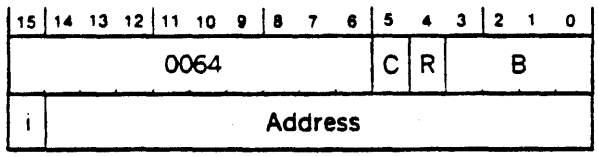


Format: 14

Addressing: Direct, indirect

Description: Stores the address of the first word of the instruction plus two (return address) in the indexing register specified by bits 0 through 2, jumps unconditionally to the instruction at the effective jump address and executes it next.

BT — Bit Test



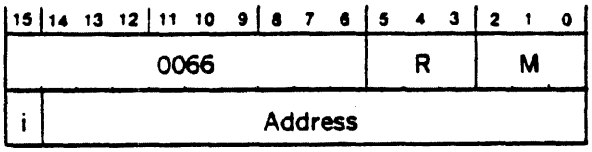
Format: 15

Addressing: Direct, indirect

Description: Tests the condition of a selected bit in either the A or B register. If the condition is met, jumps to the instruction at the effective jump address and executes it next. If the condition is not met, executes the next instruction in sequence.

The B field specifies the bit to be tested. The R field specifies the register. The C field specifies the test condition.

SRE **Skip If Register Equal**



Format: 16

Addressing: Direct, indirect, indexed, relative, post-indexed indirect, post-relative indirect.

Description: Makes a logical comparison between the register specified by bits 3 through 5 and the word at the effective address. If the compared quantities are equal, the program skips the next two locations and executes the instruction in the third location. If the compared quantities are not equal, the program executes the instruction immediately following the SRE instruction.

5.9 EXECUTION INSTRUCTIONS

This group consist of instructions that direct the program to a non-sequential address for execution of the instruction located there, and then direct the program back to the main sequence to execute the instruction following the execution instruction.

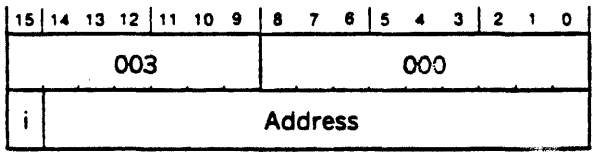
The effective address (derived from the address field of the second word) is the address of the next instruction to be executed if the execution condition is met. After executing that instruction, the program returns to the main sequence and executes the next instruction in sequence.

Note that only single-word instructions that do not specify relative addressing may be contained in the effective execution address.

Mnemonic	Instruction
XEC	Execute unconditionally
XOF	Execute if overflow indicator set

XOFN	Execute if overflow indicator not set
XAP	Execute if A register positive
XAN	Execute if A register negative
XAZ	Execute if A register zero
XBZ	Execute if B register zero
XXZ	Execute if X register zero
XANZ	Execute if A register not zero
XBNZ	Execute if B register not zero
XXNZ	Execute if X register not zero
XS1	Execute if SENSE switch 1 set
XS2	Execute if SENSE switch 2 set
XS3	Execute if SENSE switch 3 set
XS1N	Execute if SENSE switch 1 not set
XS2N	Execute if SENSE switch 2 not set
XS3N	Execute if SENSE switch 3 not set
XIF	Execute if condition(s) met

XEC **Execute Unconditionally**

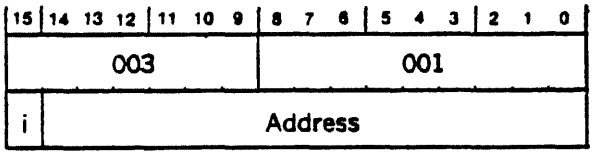


Format: 12

Addressing: Direct, indirect

Description: Executes the instruction at the effective address and then returns to execute the instruction following the XEC.

XOF — Execute if Overflow Indicator Set

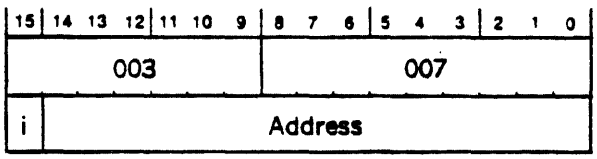


Format: 12

Addressing: Direct, indirect

Description: If the overflow indicator (OF) is set, executes the instruction at the effective address and then returns to execute the instruction following the XOF. Resets the overflow indicator. If the overflow indicator is not set, executes the next instruction in sequence.

XOFN — Execute if Overflow Indicator Not Set

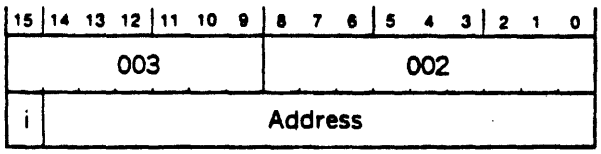


Format: 12

Addressing: Direct, indirect

Description: If the overflow indicator (OF) is not set, executes the instruction at the effective address and then returns to execute the instruction following XOFN. If the overflow indicator is set, executes the next instruction in sequence. Does not reset OF.

XAP — Execute if A Register Positive

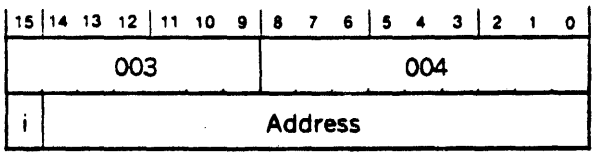


Format: 12

Addressing: Direct, indirect

Description: If the A register contains a positive value (including zero), executes the instruction at the effective address and then returns to execute the instruction following the XAP. If the A register contains a negative value, executes the next instruction in sequence.

XAN — Execute if A Register Negative

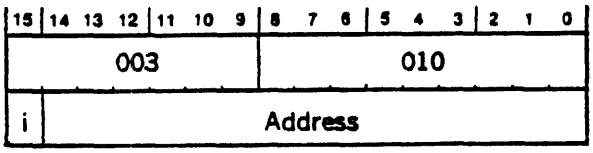


Format: 12

Addressing: Direct, indirect

Description: If the A register contains a negative value, executes the instruction at the effective address and then returns to execute the instruction following the XAN. If the A register contains a positive value (including zero), executes the next instruction in sequence.

XAZ — Execute if A Register Zero

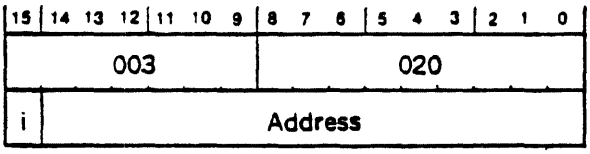


Format: 12

Addressing: Direct, indirect

Description: If the A register contains zero, executes the instruction at the effective address and then returns to execute the instruction following the XAZ. If the A register does not contain zero, executes the next instruction in sequence.

XBZ — Execute if B Register Zero

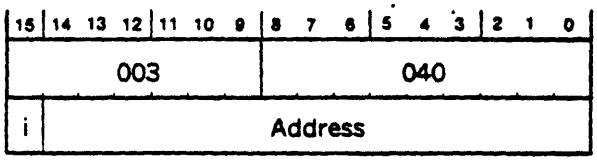


Format: 12

Addressing: Direct, indirect

Description: If the B register contains zero, executes the instruction at the effective address and then returns to execute the instruction following the XBZ. If the B register does not contain zero, executes the next instruction in sequence.

XXZ Execute if X Register Zero

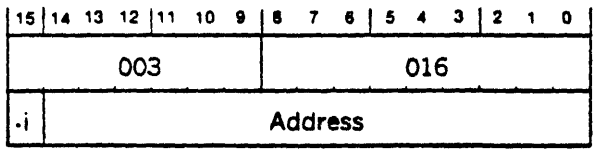


Format: 12

Addressing: Direct, indirect

Description: If the X register contains zero, executes the instruction at the effective address and then returns to execute the instruction following the XXZ. If the X register does not contain zero, executes the next instruction in sequence.

XANZ Execute if A Register Not Zero

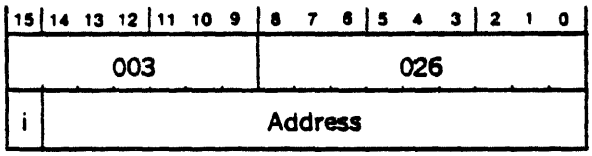


Format: 12

Addressing: Direct, indirect

Description: If the A register does not contain zero, executes the instruction at the effective address and then returns to execute the instruction following the XANZ. If the A register contains zero, executes the next instruction in sequence.

XBNZ — Execute if B Register Not Zero

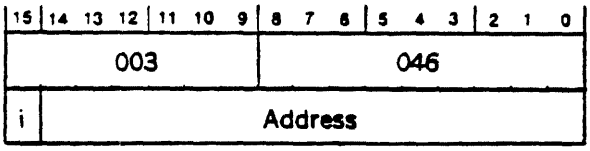


Format: 12

Addressing: Direct, indirect

Description: If the B register does not contain zero, executes the instruction at the effective address and then returns to execute the instruction following the XBNZ. If the B register contains zero, executes the next instruction in sequence.

XXNZ — Execute if X Register Not Zero

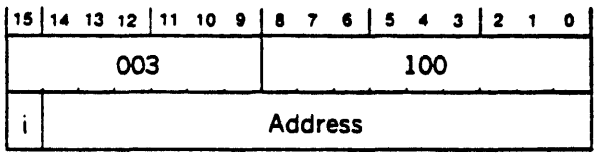


Format: 12

Addressing: Direct, indirect

Description: If the X register does not contain zero, executes the instruction at the effective address and then returns to execute the instruction following the XXNZ. If the X register contains zero, executes the next instruction in sequence.

XS1 Execute if SENSE Switch 1 Set

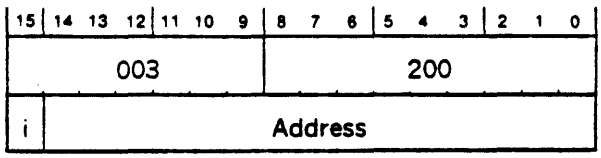


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 1 is set, executes the instruction at the effective address and then returns to execute the instruction following the XS1. If SENSE switch 1 is not set, executes the next instruction in sequence.

XS2 Execute if SENSE Switch 2 Set

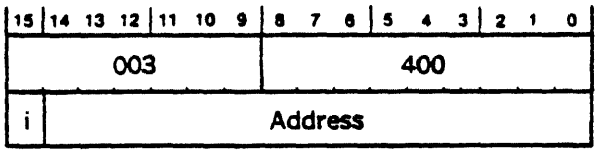


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 2 is set, executes the instruction at the effective address and then returns to execute the instruction following the XS2. If SENSE switch 2 is not set, executes the next instruction in sequence.

XS3 — Execute if SENSE Switch 3 Set

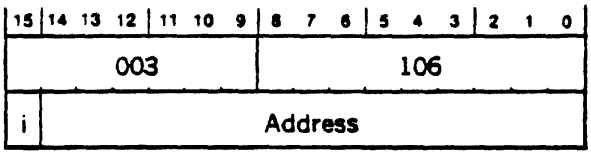


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 3 is set, executes the instruction at the effective address and then returns to execute the instruction following the XS3. If SENSE switch 3 is not set, executes the next instruction in sequence.

XS1N — Execute if SENSE Switch 1 Not Set

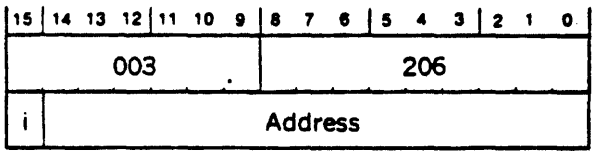


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 1 is not set, executes the instruction at the effective address and then returns to execute the instruction following the XS1N. If SENSE switch 1 is set, execute the next instruction in sequence.

XS2N **Execute if SENSE Switch 2 Not Set**

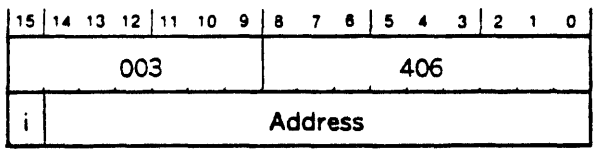


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 2 is not set, executes the instruction at the effective address and then returns to execute the instruction following the XS2N. If SENSE switch 2 is set, executes the next instruction in sequence.

XS3N **Execute if SENSE Switch 3 Not Set**

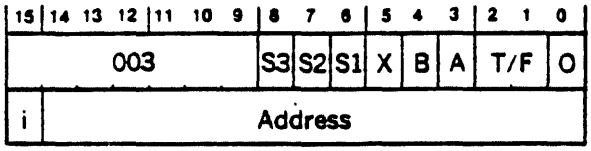


Format: 12

Addressing: Direct, indirect

Description: If SENSE switch 3 is not set, executes the instruction at the effective address and then returns to execute the instruction following the XS3N. If SENSE switch 3 is set, executes the next instruction in sequence.

XIF **Execute if Condition(s) Met**



Format: 12

Addressing: Direct, indirect

Description: Analagous to JIF. If all execution conditions are met, executes the instruction at the effective address and then returns to execute the instruction following the XIF. If all execution conditions are not met, executes the next instruction in sequence.

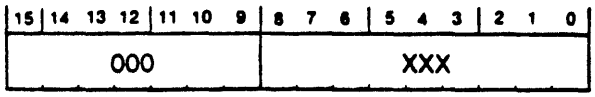
The conditions specified by setting combinations of bits 0 through 8 is the AND of each bit specification. XIF instructions in which bits 2 and 3 are both set and bit 1 is reset, or in which bits 0 through 8 contain 006_8, are undefined.

5.10 Control Instructions

This group consist of general control instructions.

Mnemonic	Instruction
HLT	Halt
ROF	Reset overflow indicator
SOF	Set overflow indicator
TSA	Transfer switches to A register
USKP	Unconditional Skip
BCS	Branch to Control Store
IDE	Interpreter Decoder
ECS	Branch to processor's extended control store

HLT **Halt**



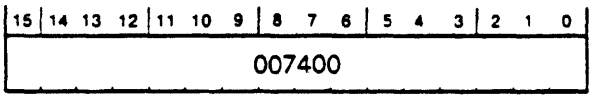
Format: 3

Addressing: None

Description: Stops computation and places the computer in step mode. To restart computation with the next instruction in sequence, press START on the control panel.

Bits 0 through 8 can contain any value assigned by the programmer to identify the halt.

ROF **Reset Overflow Indicator**

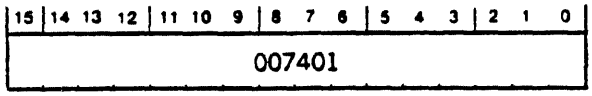


Format: 2

Addressing: None

Description: Resets the overflow indicator (OF).

SOF **Set Overflow Indicator**

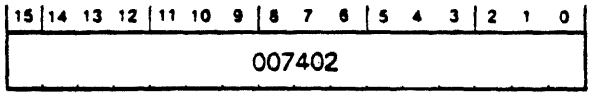


Format: 2

Addressing: None

Description: Sets the overflow indicator (OF).

TSA **Transfer Switches to A Register**

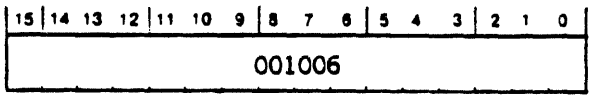


Format: 2

Addressing: None

Description: Transfers the contents of the register entry switches to the A register

USKP **Unconditional Skip**

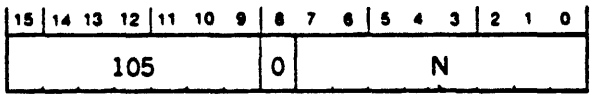


Format: 2

Addressing: None

Description: Unconditionally skips the next location in sequence.

BCS **Branch to Writable Control Store**



Format: 4

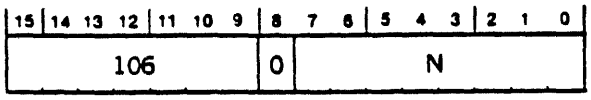
Addressing: None

Description: Branches to one of the first 32 addresses of page 0 of the writable control store (WCS). The branch address is specified by bits 0 through 4. Bits 5 through 8 are reserved for use with the micro-routine stored at the specified address. Bit 8 must be zero. An optional configuration is available to permit branching to one of the first 256 addresses of page 0 of the WCS. The address is specified by the 8-bit N field.

For V77-800 computers, three BCS instructions exist with operation codes for bits 9 through 15 specified as either 105, 106, or 107. Bit 0 through 8 have no effect. Each BCS instruction causes a branching operation to page 0 of the WCS. Functions of these instructions, along with the WCS entry address, are listed as follows:

BCS 105XXX	Branches to address 13 (octal 15). Reserved for COBOL, Pascal, and VORTEX microprogrammed packages (if installed).
BCS 106XXX	Branches to address 14 (octal 16). Reserved for FORTRAN 77 microprogrammed package (if installed).
BCS 107XXX	Branches to address 15 (octal 17). Reserved for user microprogramming.

IDE **Interpreter Decoder**

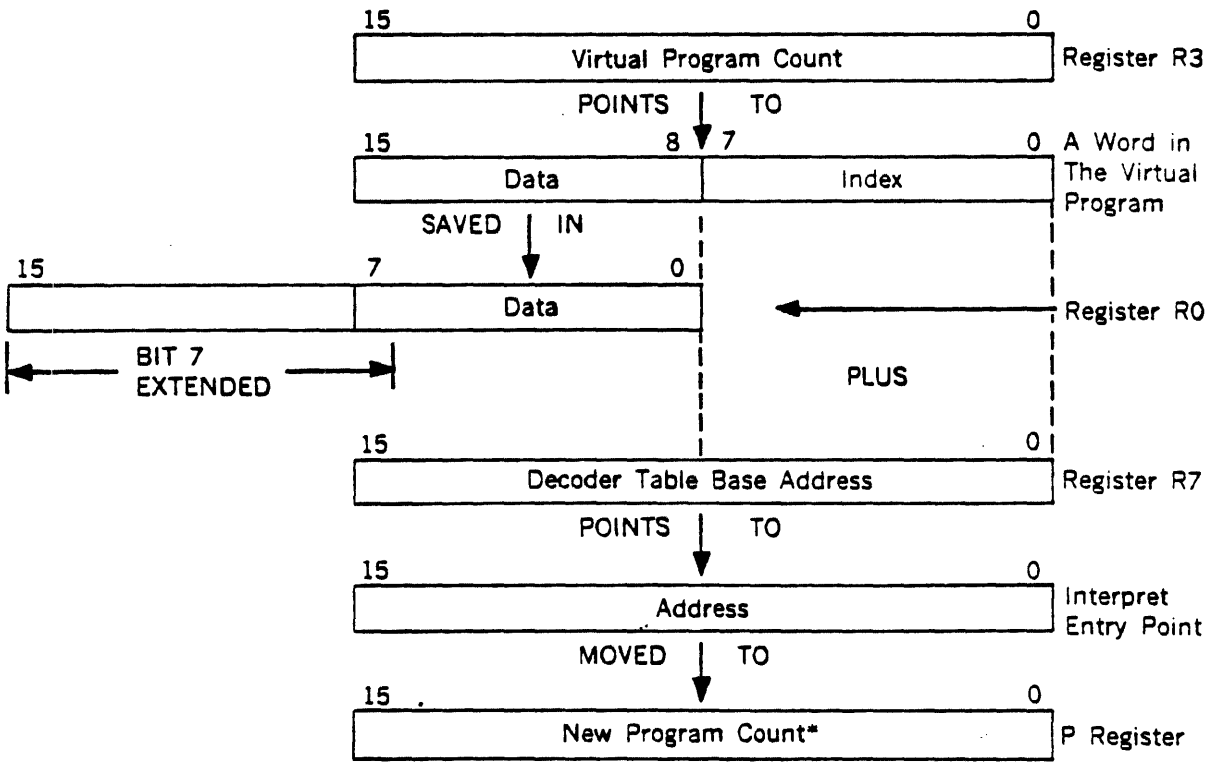


Format: 4

Addressing: None

Description: Fetches a word from a virtual program and uses the right byte as an index to a decoder table. An entry in the decoder table points to an interpreter program. Register R3 contains the virtual program count. Register R0 saves the left byte of the first word fetched. Register R7 contains the decoder table base address. The N field may contain any binary number. The decoder sequence is diagrammed in figure 5-1.

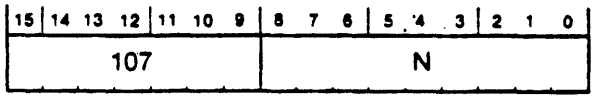
Bit 8 must be zero.



*Address of the next instruction to be executed.

Figure 5-1. Interpreter Decoder Instruction

ECS **Branch to Processor's Extended Control Store**



Format: 3

Addressing: None

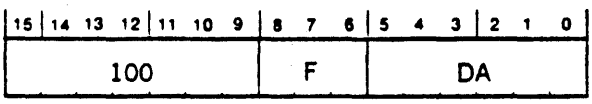
Description: Branches to one of the first 32 addresses of the processor's extended control store. The branch address is specified by bits 0 through 4. Bits 5 through 8 are reserved for use with the micro-routine stored at the specified address

5.11 I/O Instructions

This group consists of instructions for implementing communication between the computer and the peripheral devices on the I/O bus.

Mnemonic	Instruction		
EXC	External control	INB	Input to B register
EXC2	Auxiliary external control	INAB	Input to A and B registers
SEN	Program sense	OAR	Output from A register
CIA	Clear and input to A register	OBR	Output from B register
CIB	Clear and input to B register	OAB	Output from A and B registers
CIAB	Clear and input to A and B registers	IME	Input to memory
INA	Input to A register	OME	Output from memory

EXC **External Control**

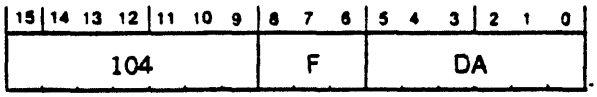


Format: 9

Addressing: None

Description: Orders the peripheral device, specified by the device address DA, to perform function F.

EXC2 Auxiliary External Control

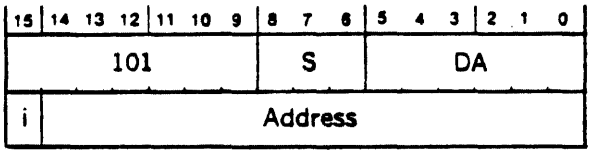


Format: 9

Addressing: None

Description: Orders the peripheral device, specified by the device address DA, to perform function F.

SEN Program sense

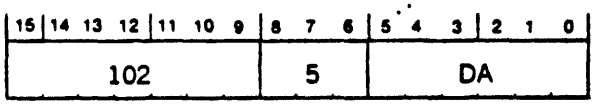


Format: 22

Addressing: Direct, indirect

Description: Senses status line S in the peripheral device specified by the device address DA. If the computer receives a true response signal, jumps to the instruction at the effective address and executes it next. If the response signal is false, executes the next instruction in sequence.

CIA — Clear and Input to A Register

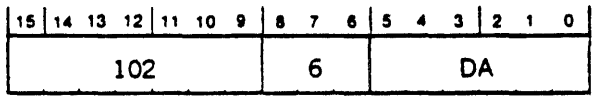


Format: 10

Addressing: None

Description: Clears the A register and inputs to the A register a data word from the peripheral device specified by the device address DA.

CIB — Clear and Input to B Register

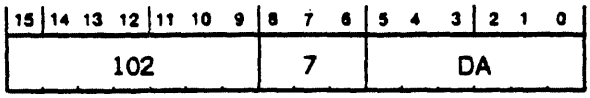


Format: 10

Addressing: None

Description: Clears the B register and inputs to the B register a data word from the peripheral device specified by the device address DA.

CIAB **Clear and Input to A and B Registers**

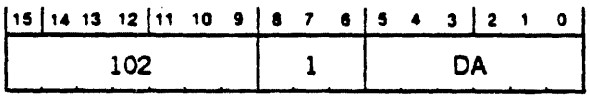


Format: 10

Addressing: None

Description: Clears the A and B registers and inputs to both registers a data word from the peripheral device specified by the device address DA.

INA **Input to A Register**

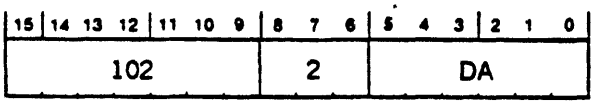


Format: 10

Addressing: None

Description: Inclusively-ORs a data word from the specified peripheral device with the contents of the A register and places the result in the A register. The DA field contains the device address.

INB — Input to B Register

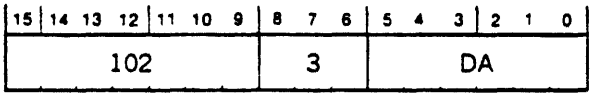


Format: 10

Addressing: None

Description: Inclusively-ORs a data word from the specified peripheral device with the contents of the B register and places the result in the B register. The DA field contains the device address.

INAB — Input to A and B Registers

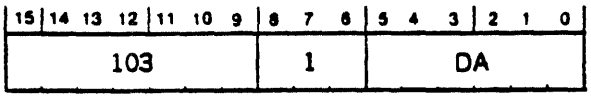


Format: 10

Addressing: None

Description: Inclusively-ORs a data word from the specified peripheral device with the inclusive-OR of the contents of the A and B registers and places the result into both the A and B registers. The DA field contains the device address.

OAR — Output from A Register

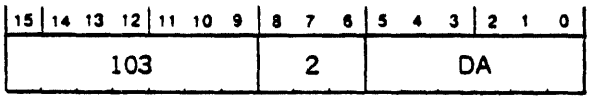


Format: 10

Addressing: None

Description: Outputs the contents of the A register to the peripheral device specified by the device address DA.

OBR — Output from B Register

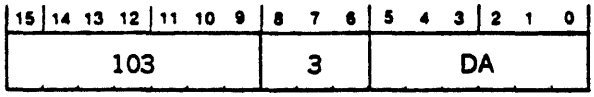


Format: 10

Addressing: None

Description: Outputs the contents of the B register to the peripheral device specified by the device address DA.

OAB **Output Inclusive-OR of A and B Registers**

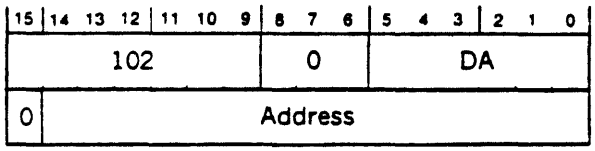


Format: 10

Addressing: None

Description: Outputs the inclusive-ORed contents of the A and B registers to the peripheral device specified by the device address DA.

IME **Input to Memory**

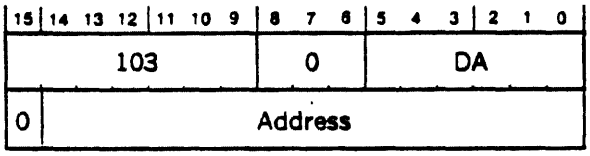


Format: 23

Addressing: Direct

Description: Inputs a data word from the specified peripheral device to the cleared memory address. The DA field contains the device address.

OME **Output from Memory**



Format: 23

Addressing: Direct

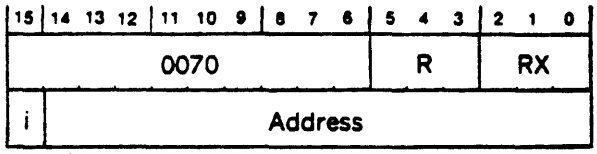
Description: Outputs the contents of the memory location addressed by the second word to the specified peripheral device. The DA field contains the device address.

5.12 Register-to-Memory Instructions

This group consists of instructions which perform load, store, add, or subtract operations between the contents of a memory location and one of eight registers, R0 through R7. The RX field specifies either no indexing or indexing by one of seven registers (R1 through R7).

Mnemonic	Instruction
LD	Load
ST	Store
AD	Add
SB	Subtract

LD Load

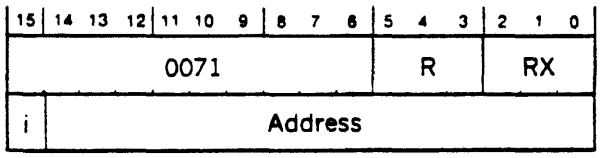


Format: 17

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: The contents of the effective memory location replace the contents of the register specified by the R field.

ST Store

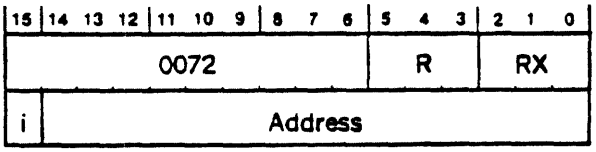


Format: 17

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: The contents of the register specified by the R field replace the contents of the effective memory address.

AD Add

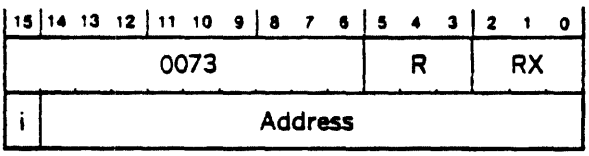


Format: 17

Addressing: Direct, indirect, indexed, pre-indexed indirect.

Description: The contents of the register specified by the R field are added to the contents of the effective memory address. The sum replaces the contents of the register specified by the R field. If both operand have the same sign and the result has the opposite sign, the overflow indicator (OF) is set.

SB Subtract



Format: *17*

Addressing: Direct, indirect, indexed, pre-indexed indirect

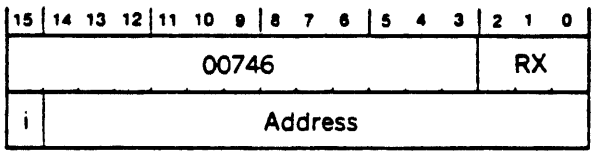
Description: The contents of the effective memory address are subtracted from the contents of the register specified by the R field. The difference replaces the contents of the register specified by the R field. If the operands have different signs and the sign of the result equals the sign of the contents of the effective memory address, the overflow indicator (OF) is set.

5.13 Byte Instructions

This group consists of instructions for loading a byte of data from memory into register R0 or for storing a byte of data into memory from register R0. Indexing by any one of eight registers (R0 through R7) is specified by the RX field. For these instructions, a byte is defined as either the right (lower) or left (upper) eight bits of a 16-bit word.

Mnemonic	Instruction
LBT	Load byte
SBT	Store Byte

LBT Load Byte

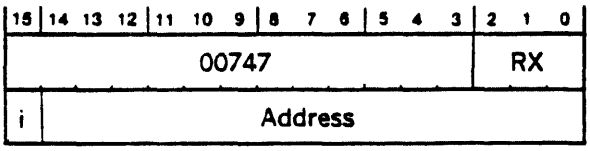


Format: 18

Addressing: Indexed, post-indexed indirect

Description: The contents of the effective byte address replace the contents of the right byte of register R0. The contents of the left byte of register R0 are replaced by zeros.

SBT **Store Byte**



Format: 18

Addressing: Indexed, post-indexed indirect

Description: The contents of the right byte of register R0 replace the contents of the effective byte address.

5.14 Jump-If Instructions

This group consists of instructions that direct the program to a non-sequential address for execution of the instruction there, but they neither mark the location (as do jump-and-mark instruc- tions) nor do they bring the program back to the main program sequence (as do the execution instructions).

The effective jump address is the address of the next instruction to be executed if the jump condition is met. The program then continues to execute instructions following the jump address.

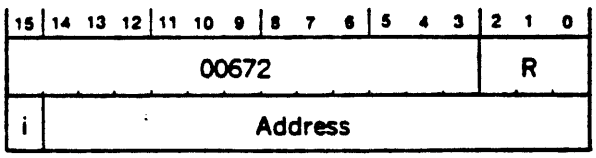
If the jump condition is not met, the program executes the instruction immediately following the second word of the jump-if instruction.
The jump-if instructions are functionally similar to the jump instructions of section 5.6 but differ in format and in the type of jump condition that can be specified. These instructions must not be confused with the jumps instructions of section 5.6, particularly the JIF instruction.

Mnemonic	Instruction
JZ	Jump If register zero
JNZ	Jump If register not zero
JN	Jump If register negative
JP	Jump If register positive

JDZ Jump If double precision register zero

JDNZ Jump If double precision register not zero

JZ Jump If Register Zero

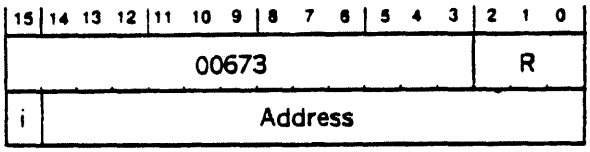


Format: 19

Addressing: Direct, indirect

Description: If the register specified by the R field contains zero, the instruction at the effective jump address is executed. If the register (R) does not contain zero, the next instruction in sequence is executed. Contents of the register (R) are unaltered.

JNZ — Jump If Register Not Zero

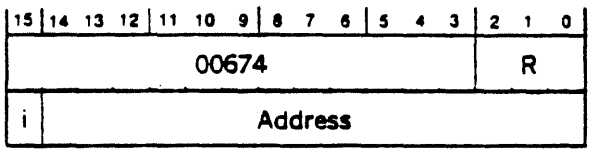


Format: 19

Addressing: Direct, indirect

Description: If the register specified by the R field contains a value that is not zero, the instruction at the effective jump address is executed. If the register (R) contains zero, the next instruction in sequence is executed. The contents of the register (R) are unaltered.

JN — Jump If Register Negative

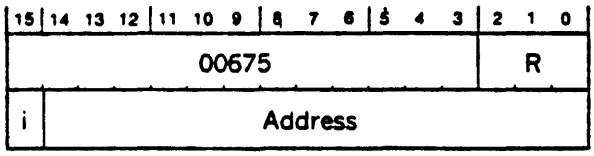


Format: 19

Addressing: Direct, indirect

Description: If the register specified by the R field contains a negative value, the instruction at the effective jump address is executed. If the register (R) contains a positive value (including zero), the next instruction in sequence is executed. Contents of the register (R) are unaltered.

JP — Jump If Register Positive

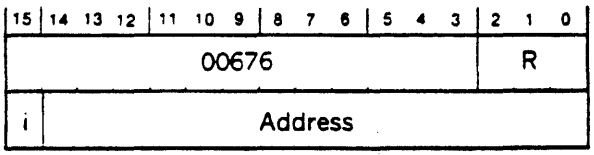


Format: 19

Addressing: Direct, indirect

Description: If the register specified by the R field contains a positive value (including zero), the instruction at the effective jump address is executed next. If the register (R) contains a negative value, the next instruction in sequence is executed. The contents of the register (R) are unaltered.

JDZ — Jump If Double-Precision Register Zero

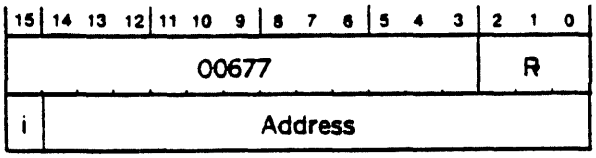


Format: 19

Addressing: Direct, indirect

Description: If the double-precision register specified by the R field contains zero, the instruction at the effective jump address is executed. If the value of the R field is 0, double-precision register R0-R1 is specified; if the value is 4, double-precision register R4-R5 is specified. If the double-precision register (R) does not contain zero, the next instruction in sequence is executed. The contents of the double-precision register (R) are unaltered.

JDNZ **Jump If Double-Precision Register Not Zero**



Format: 19

Addressing: Direct, indirect

Description: If the double-precision register specified by the R field does not contain zero, the instruction at the effective jump address is executed next. If the value of the R field is 0, double-precision register R0-R1 is specified; if the value is 4, the double-precision register R4-R5 is specified. If the double-precision register (R) contains zero, the next instruction in sequence is executed. The contents of double-precision register (R) are unaltered.

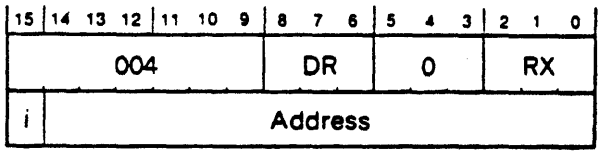
5.15 Double-Precision Instructions

This group consists of instructions which perform a load, store, add, subtract, AND, OR, or exclusive-OR between the contents of a double-precision memory location and either of two double-precision registers. Either no indexing or indexing by any one of seven index registers (R1 through R7) can be specified by the RX field. The double-precision registers are R0-R1 and R4-R5; R0 and R4 are the most significant halves of their respective double-precision registers.

Mnemonic	Instruction
DLD	Double load
DST	Double store
DADD	Double add
DSUB	Double subtract
DAN	Double AND

DOR Double OR
DER Double exclusive-OR

DLD **Double Load**

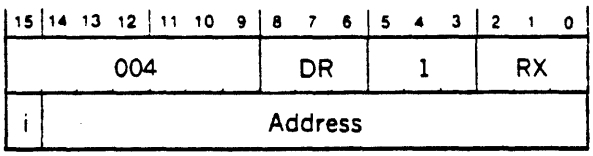


Format: 20

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: Double-precision contents of the effective memory address replace the contents of the double-precision register specified by the DR field. If the value of the DR field is 6, double-precision register R0-R1 is specified; if the value is 7, double-precision register R4-R5 is specified.

DST **Double Store**

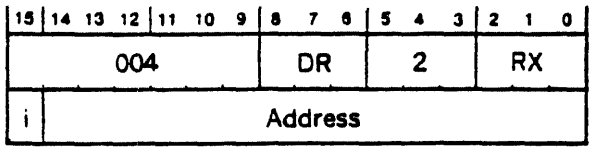


Format: 20

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: Contents of the double-precision register specified by the DR field replace the double-precision contents of the effective memory location. If the value of the DR field is 6, double-precision register R0-R1 is specified; if the value is 7, double-precision register R4-R5 is specified.

DADD **Double Add**

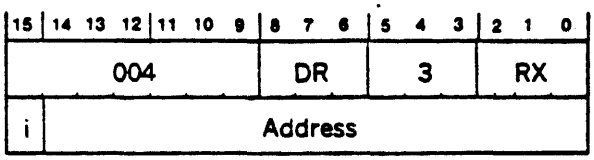


Format: 20

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: Contents of the double-precision register specified by the DR field are added to the double-precision contents of the effective memory address. The sum replaces the contents of the double-precision register specified by the DR field. If the value of the DR field is 6, double-precision register R0-R1 is specified; if the value is 7, double-precision register R4-R5 is specified. If both double-precision operands have the same sign and the result has the opposite sign, the overflow indicator (OF) is set.

DSUB **Double Subtract**

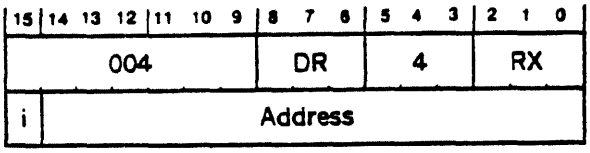


Format: 20

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: Double-precision contents of the effective memory address are subtracted from the contents of the double-precision register specified by the DR field. The difference replaces the contents of the double-precision register specified by the DR field. If the value of the DR field is 6, double-precision register R0-R1 is specified; if the value is 7, double-precision register R4-R5 is specified. If the double-precision operand have opposite signs and the sign of the result does not equal the sign of the original contents of the specified double-precision register, the overflow indicator (OF) is set.

DAN **Double AND**

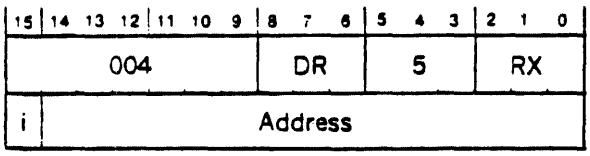


Format: 20

Addressing: Direct, indirect, indexed, pre-indexed indirect

Description: A bit by bit logical AND function is formed between corresponding bits of the double-precision register specified by the DR field and the double-precision contents of the effective memory address. The logical results replace the contents of the speci- field double-precision register. If the value of the DR field is 6, double-precision register R0-R1 is specified; if the value is 7, double-precision register R4-R5 is specified.

DOR **Double OR**

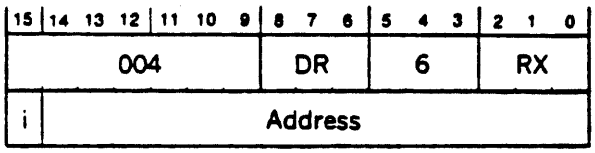


Format: 20

Addressing: Direct. indirect, indexed. pre-indexed indirect

Description: A bit by bit logical OR function is formed between corresponding bits of the double-precision register specified by the DR field and the double-precision contents of the effective memory address. The logical results replace the contents of the spec- fied double-precision register. If the value of the DR field is 6. double-precision register R0-R1 is specified: if the value is 7. double-precision register R4-R5 is specified.

DER **Double Exclusive-OR**



Format: 20

Addressing: Direct, indirect, indexed, pre-indexed indirect

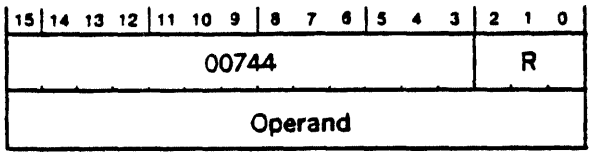
Description: A bit by bit logical exclusive-OR function is formed between corresponding bits of the double-precision register specified by the DR field and the double-precision contents of the effective memory address. The logical results replace the contents of the specified double-precision register. If the value of the DR field is 6, double-precision register R0-R1 is specified; if the value is 7, double-precision register R4-R5 is specified.

5.16 Register Immediate Instructions

This group consist of instructions which perform load or add operations between the immediate operand in the second word of the instruction and any one of eight register (R0 through R7).

Mnemonic	Instruction
LDI	Load immediate
ADI	Add immediate

LDI — Load Immediate

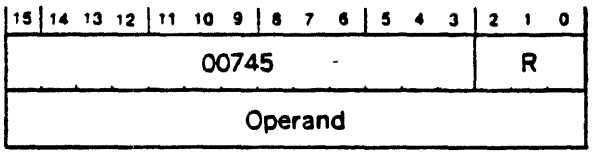


Format: 25

Addressing: None

Description: The immediate operand replaces the contents of the register specified by the R field.

ADI — Add Immediate



Format: 25

Addressing: None

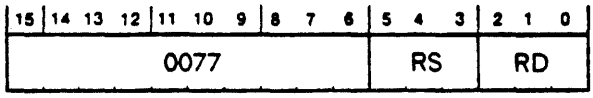
Description: Contents of the register specified by the R field are added to the immediate operand. The sum replaces the contents of the register specified by the R field. If the operands have the same sign and the result has an opposite sign, the overflow indicator (OF) is set.

5.17 Register-to-Register Instructions

This group consists of instructions which perform transfer, add, or subtract operations between source and destination registers. Any one of eight registers (R0 through R7) may be specified as a source or as a destination register.

Mnemonic	Instruction
T	Transfer
ADR	Add register
SBR	Subtract register
INC	Increment
DEC	Decrement
COM	Complement

T — Transfer

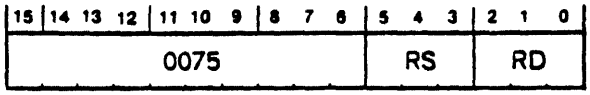


Format: 8

Addressing: None

Description: The contents of the register specified by the RS field replace the contents of the register specified by the RD field.

ADR — Add Register

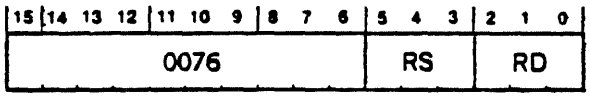


Format: 8

Addressing: None

Description: Contents of the source register specified by the RS field are added to the contents of the destination register specified by the RD field. The sum replaces the contents of the specified destination register. If both operands have the same sign and the result has the opposite sign, the overflow indicator (OF) is set.

SBR — Subtract Register

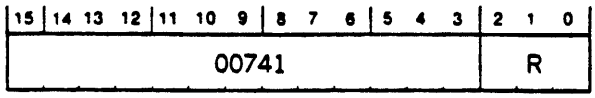


Format: 8

Addressing: None

Description: Contents of the source register specified by the RS field are subtracted from the contents of the destination register specified by the RD field. The difference replaces the contents of the specified destination register. If the operands have opposite signs and the sign of the result equals the sign of the specified source register, the overflow indicator (OF) is set.

INC Increment

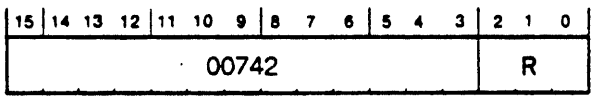


Format: 7

Addressing: None

Description: Contents of the register specified by the R field are incremented by one. The incremented value replaces the contents of the specified register (R). If the specified register (R) contains an original value of 077777_8, the resulting value of the register becomes 100000_8 and the overflow indicator (OF) is set.

DEC Decrement

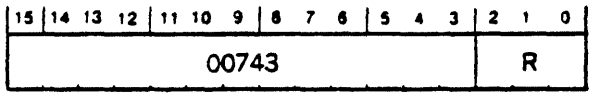


Format: 7

Addressing: None

Description: Contents of the register specified by the R field are decremented by one. The decremented value replaces the contents of the specified register (R). If the specified register contains an original value of 100000_8, the resulting value of the register is 077777_8 and the overflow indicator (OF) is set.

COM **Complement**



Format: 7

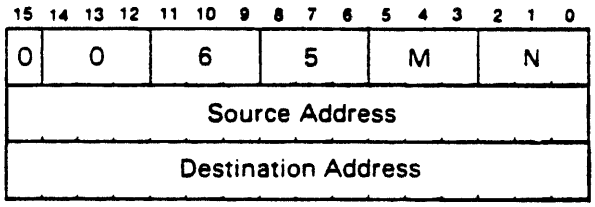
Addressing: None

Description: The ones complement (logical inversion) of the contents of the register specified by the R field replaces the original contents of the specified register.

5.18 V77-800 STANDARD EXTENSIONS

These instructions are available only with V77-800 computers.

DMOVSD Double Word Move

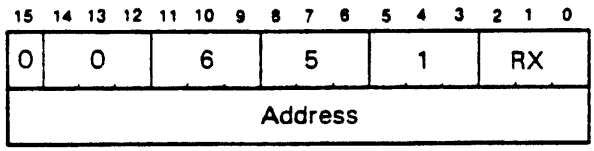


Format: 26.

Addressing: Direct, indexed

Description: Moves up to seven double words from one part of main memory to another. Source and destination address fields specify starting addresses for the source and destination of the transfer. Number of words transferred is specified by the N field. Addressing is either direct or indexed as specified by the M field codes (see section 2).

RGLD Registers Load

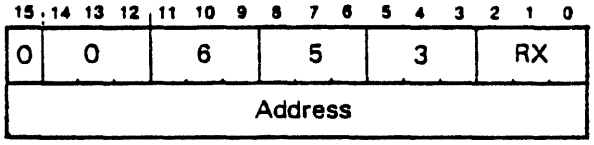


Format: 27

Addressing: Direct, indexed

Description: Registers R0 through R7 are loaded respectively from a block of eight consecutive memory locations. The starting address of the memory locations is specified by the address field. Indexing by any one of seven registers (R1 through R7) is specified by the RX field. With the RX field equal to zero, no indexing is specified.

RGST **Registers Store**

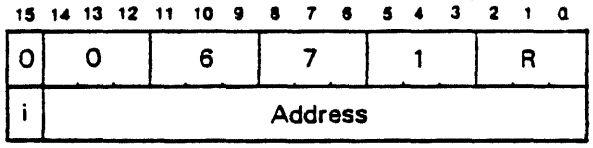


Format: 27

Addressing: Direct, indexed

Description: Contents of registers R0 through R7 are stored respectively into a block of eight consecutive memory locations. The starting address of the memory locations is specified by the address field. Indexing by any one of seven registers (R1 through R7) is specified by the RX field. With the RX field equal to zero, no indexing is specified.

DJP **Decrement Register and Jump**

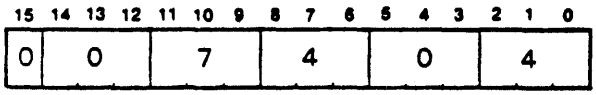


Format: 19

Addressing: Direct, indirect

Description: Subtracts one from contents of the register specified by the R field and, if the initial register value was not negative, jumps to the effective address.

BMOVW **Block Move**

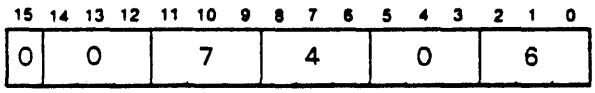


Format: 2

Addressing: None

Description: Moves a block of up to 32K words from one part of main memory to another. Starting address of the source block is specified by register R0. Starting address of the destination block is specified by register R1. Block length is specified by register R6.

STWRDS **Store Words**

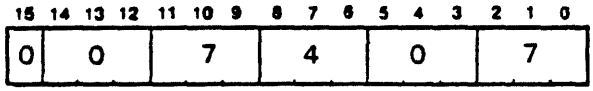


Format: 2

Addressing: None

Description: Stores contents of register R0 into a block of up to 32K words of main memory. Starting address of the memory block is specified by register R1. Block length is specified by register R6.

STBYTS **Store Bytes**



Format: 2

Addressing: None

Description: Stores the right-byte contents of register R0 into a block of up to 32K words of main memory. Starting address of the memory block is specified by register R1. Block length is specified by register R6.

APPENDIX A
INDEX OF INSTRUCTIONS

Table A-1 is a listing by mnemonic. Table A-2 is a listing by octal code.

Table A-1

Mnemonic	Octal Code	Description	Page
AD	0072xx	Add	5-94
ADD	12xxxx	Add memory to A register	5-14
ADDE	00612x 00632x	Add extended	5-15
ADDI	006120	Add immediate	5-15
ADI	00745x	Add Immediate	5-106
ADR	0075xx	Add Register	5-108
ANA	15xxxx	AND memory and A register	5-25
ANAE	00615x 00635X	AND extended	5-25
ANAI	006150	AND immediate	5-26
AOFA	005511	Add overflow to A register	5-41
AOFB	005522	Add overflow to B register	5-41
AOFX	005544	Add overflow to X register	5-41
ASLA	004200 + n	Arithmetic shift left A register	5-31
ASLB	004000 + n	Arithmetic shift left B register	5-31
ASRA	004300 + n	Arithmetic shift right A register	5-30
ASRB	004100 + n	Arithmetic shift right B register	5-30
BCS	105xxx	Branch to writable control store	5-83
BMOVW	007404	Block Move	5-113
BT	0064xx	Bit test	5-69
CIA	1025xx	Clear and input of A register	5-87
CIAB	1027xx	Clear and input to A and B registers	5-88
CIB	1026xx	Clear and input to B register	5-87
COM	00743x	Complement Register	5-110
COMP	005xxx	Complement source to destination registers	5-47
CPA	005211	Complement A register	5-41
CPB	005222	Complement B register	5-41
CPX	005244	Complement X register	5-41
DADD	004x2x	Double Add	5-102
DAN	004x4x	Double And	5-104
DAR	005311	Decrement A register	5-40
DBR	005322	Decrement B register	5-40

Table A-1 *(continued)*

Mnemonic	Octal Code	Description	Page
DCA	005301	Decrement cleared A register	5-43
DCB	005302	Decrement cleared B register	5-43
DCX	005304	Decrement cleared X register	5-43
DEC	00742x	Decrement Register	5-109
DECR	0053xx	Decrement source to destination registers	5-46
DER	004x6x	Double Exclusive OR	5-105
DIV	17xxxx	Divide	5-19
DIVE	00617x 00637x	Divide extended	5-20
DIVI	006170	Divide immediate	5-20
DJP	00671xx	Decrement Register and Jump	5-112
DLD	004x0x	Double Load	5-101
DMOVSD	0065xx	Double Word Move	5-111
DOR	004x5x	Double OR	5-104
DST	004x1x	Double Store	5-101
DSUB	004x3x	Double Subtract	5-103
DXR	005344	Decrement X register	5-40
ECS	107xxx	Branch to Processor's Extended Control Store	5-85
ERA	13xxxx	Exclusive-OR memory and A register	5-23
ERAE	00613x 00633x	Exclusive-OR extended	5-24
ERAI	006130	Exclusive-OR immediate	5-24
EXC	100xxx	External control	5-85
EXC2	104xxx	Auxiliary external control	5-86
HLT	000000	Halt	5-81
IAR	005111	Increment A register	5-39
IBR	005122	Increment B register	5-39
ICA	005101	Increment cleared A register	5-43
ICB	005102	Increment cleared B register	5-43
ICX	005104	Increment cleared X register	5-43
IDE	106000	Interpreter decoder	5-84
IJMP	0067xx	Indexed jump	5-68
IME	1020xx	Input to memory	5-91
INA	1021xx	Input to A register	5-88
INAB	1023xx	Input to A and B registers	5-89
INB	1022xx	Input to B register	5-89
INC	04741x	Increment Register	5-109
INCR	0051xx	Increment source to destination registers	5-45
INR	04xxxx	Increment memory and replace	5-13
INRE	00604x 00624x	Increment memory and replace extended	5-13

Table A-1 *(continued)*

Mnemonic	Octal Code	Description	Page
INRI	006040	Increment memory and replace immediate	5-14
IXR	005144	Increment X register	5-39
JAN	001004	Jump if A register negative	5-51
JANM	002004	Jump and mark if A register negative	5-61
JANZ	001016	Jump if A register not zero	5-53
JANZM	002016	Jump and mark if A register not zero	5-63
JAP	001002	Jump if A register positive	5-50
JAPM	002002	Jump and mark if A register positive	5-60
JAZ	001010	Jump if A register zero	5-51
JAZM	002010	Jump and mark if A register zero	5-61
JBNZ	001026	Jump if B register not zero	5-53
JBNZM	002026	Jump and mark if B register not zero	5-63
JBZ	001020	Jump if B register zero	5-52
JBZM	002020	Jump and mark if B register zero	5-62
JDNZ	00677x	Jump if Double-Precision Register Not Zero	5-100
JDZ	00676x	Jump if Double-Precision Register Zero	5-99
JIF	001xxx	Jump if conditions met	5-57
JIFM	002xxx	Jump and mark if conditions met	5-67
JMP	001000	Jump unconditionally	5-49
JMPM	002000	Jump and mark unconditionally	5-59
JN	00674x	Jump If Register Negative	5-98
JNZ	00673x	Jump If Register Not Zero	5-98
JOF	001001	Jump if overflow indicator set	5-49
JOFN	001007	Jump if overflow indicator not set	5-50
JOFM	002001	Jump and mark if overflow indicator set	5-59
JOFNM	002007	Jump and mark if overflow indicator not set	5-60
JP	00675x	Jump If Register Positive	5-99
JSR	0065xx	Jump unconditionally and set return in X register	5-69
JS1M	002100	Jump and mark if SENSE switch 1 set	5-64

Table A-1 *(continued)*

Mnemonic	Octal Code	Description	Page
JS2M	002200	Jump and mark if SENSE switch 2 set	5-65
JS3M	002400	Jump and mark if SENSE switch 3 set	5-65
JS1N	001106	Jump if SENSE switch 1 not set	5-56
JS2N	001206	Jump if SENSE switch 2 not set	5-56
JS3N	001406	Jump if SENSE switch 3 not set	5-57
JS1NM	002106	Jump and mark if SENSE switch 1 not set	5-66
JS2NM	002206	Jump and mark if SENSE switch 2 not set	5-66
JS3NM	002406	Jump and mark if SENSE switch 3 not set	5-67
JSS1	001100	Jump if SENSE switch 1 set	5-54
JSS2	001200	Jump if SENSE switch 2 set	5-55
JSS3	001400	Jump if SENSE switch 3 set	5-55
JXNZ	001046	Jump if X register not zero	5-54
JXNZM	002046	Jump and mark if X register not zero	5-64
JXZ	001040	Jump if X register zero	5-52
JXZM	002040	Jump and mark if X register zero	5-62
JZ	00672x	Jump If Register Zero	5-97
LASL	004400 + n	Long arithmetic shift left	5-32
LASR	004500 + n	Long arithmetic shift right	5-32
LBT	00746x	Load Byte	5-95
LD	0070xx	Load	5-93
LDA	01xxxx	Load A register	5-3
LDAE	00601x 00621x	Load A register extended	5-3
LDAI	006010	Load A register immediate	5-4
LDB	02xxxx	Load B register	5-4
LDBE	00602x 00622x	Load B register extended	5-5
LDBI	006020	Load B register immediate	5-5
LDI	00744x	Load Immediate	5-106
LDX	03xxxx	Load X register	5-6
LDXE	00603x 00623x	Load X register extended	5-6
LDXI	006030	Load X register immediate	5-7
LLRL	004440 + n	Long logical rotation left	5-29
LLSR	004540 + n	Long logical rotation right	5-29
LRLA	004240 + n	Logical rotation left A register	5-28

Table A-1 *(continued)*

Mnemonic	Octal Code	Description	Page
LRLB	004040 + n	Logical rotation left B register	5-28
LSRA	004340 + n	Logical shift right A register	5-27
LSRB	004140 + n	Logical shift right B register	5-27
MERG	0050xx	Merge source to destination registers	5-44
MUL	16xxxx	Multiply	5-17
MULE	00616x 00636x	Multiply extended	5-18
MULI	006160	Multiply immediate	5-19
NOP	005000	No operation	5-33
OAB	1033xx	Output inclusive-OR of A and B registers	5-91
OAR	1031xx	Output from A register	5-90
OBR	1032xx	Output from B register	5-90
OME	1030xx	Output from memory	5-92
ORA	11xxxx	Inclusive-OR memory and A register	5-21
ORAE	00611x 00631x	Inclusive-OR extended	5-22
ORAI	006110	Inclusive-OR immediate	5-23
RGLD	00651x	Register Load	5-111
RGST	00653x	Registers Store	5-112
ROF	007400	Reset overflow indicator	5-81
SB	0073xx	Subtract	5-94
SBR	0076xx	Subtract Register	5-108
SBT	00747x	Store Byte	5-96
SEN	101xxx	Program sense	5-86
SOF	007401	Set overflow indicator	5-82
SOFA	005711	Subtract overflow from A register	5-42
SOFB	005722	Subtract overflow from B register	5-42
SOFX	005744	Subtract overflow from X register	5-42
SRE	0066xx	Skip if register equal	5-70
ST	0071xx	Store	5-93
STA	05xxxx	Store A register	5-7
STAE	00605x 00625x	Store A register extended	5-8
STAI	006050	Store A register immediate	5-8
STB	06xxxx	Store B register	5-9
STBE	00606x 00626x	Store B register extended	5-9

Table A-1 *(continued)*

Mnemonic	Octal Code	Description	Page
STBI	006060	Store B register immediate	5-10
STBYTS	007407	Store Bytes	5-114
STX	07xxxx	Store X register	5-10
STXE	00607x 00627x	Store X register extended	5-11
STXI	006070	Store X register immediate	5-11
SUB*	14xxxx	Subtract memory from A register	5-16
SUBE	00614x 00634x	Subtract extended	5-16
SUBI	006140	Subract immediate	5-17
STWRDS	007406	Store Words	5-113
T	0077xx	Transfer	5-107
TAB	005012	Transfer A register to B register	5-34
TAX	005014	Transfer A register to X register	5-34
TBA	005021	Transfer B register to A register	5-35
TBX	005024	Transfer B register to X register	5-35
TSA	007402	Transfer switches to A register	5-82
TXA	005041	Transfer X register to A register	5-36
TXB	005042	Transfer X register to B register	5-36
TZA	005001	Transfer zero to A register	5-36
TZB	005002	Transfer zero to B register	5-37
TZX	005004	Transfer zero to X register	5-37
USKP	001006	Unconditional Skip	5-82
XAN	003004	Execute if A register negative	5-73
XANZ	003016	Execute if A register not zero	5-75
XAP	003002	Execute if A register positive	5-73
XAZ	003010	Execute if A register zero	5-74
XBNZ	003026	Execute if B register not zero	5-76
XBZ	003020	Execute if B register zero	5-74
XEC	003000	Execute unconditionally	5-71
XIF	003xxx	Execute if conditions met	5-80
XOF	003001	Execute if overflow indicator set	5-72
XOFN	003007	Execute if overflow indicator not set	5-72

Table A-1 *(continued)*

Mnemonic	Octal Code	Description	Page
XS1	003100	Execute if SENSE switch 1 set	5-77
XS2	003200	Execute if SENSE switch 2 set	5-77
XS3	003400	Execute if SENSE switch 3 set	5-78
XS1N	003106	Execute if SENSE switch 1 not set	5-78
XS2N	003206	Execute if SENSE switch 2 not set	5-79
XS3N	003406	Execute if SENSE switch 3 not set	5-79
XXNZ	003046	Execute if X register not	5-76
XXZ	003040	Execute if X register zero	5-75
ZERO	005007	Zero (clear) registers	5-47

Table A-2

OCTAL	MNEMONIC	OCTAL	MNEMONIC
000 XXX	HLT	003 016	XAZN
		003 020	XBZ
001 000	JMP	003 026	XBNZ
001 001	JOF	003 040	XXZ
001 002	JAP	003 046	XXNZ
001 004	JAN	003 100	XS1
001 006	USKP	003 106	XS1N
001 007	JOFN	003 200	XS2
001 010	JAZ	003 206	XS2N
001 016	JANZ	003 400	XS3
001 020	JBZ	003 406	XS3N
001 026	JBNZ	003 XXX	XIF
001 040	JXZ		
001 046	JXNZ	004 000	ASLB
001 100	JSS1	004 040	LRLB
001 106	JS1N	004 100	ASRB
001 200	JS2	004 140	LSRB
001 206	JS2N	004 200	ASLA
001 400	JS3	004 240	LRLA
001 406	JS3N	004 300	ASRA
001 XXX	JIF	004 340	LSRA
		004 400	LASL
002 000	JMPM	004 440	LLRL
002 001	JOFM	004 500	LASR
002 002	JAPM	004 540	LLSR
002 004	JANM	004 XOX	DLD
002 007	JOFNM	004 X1X	DST
002 010	JAZM	004 X2X	DADD
002 016	JANZM	004 X3X	DSUB
002 020	JBZM	004 X4X	DAN
002 026	JBNZM	004 X5X	DOR
002 040	JXZM	004 X6X	DER
002 046	JXNZM		
002 100	JS1M	005 000	NOP
002 106	JS1NM	005 001	TZA
002 200	JS2M	005 002	TZB
002 206	JS2NM	005 004	TZX
002 400	JS3M	005 007	ZERO
002 406	JS2NM	005 012	TAB
002 XXX	JIFM	005 014	TAX
		005 021	TBA
003 000	XEC	005 024	TBX
003 001	XOF	005 041	TXA
003 002	XAP	005 042	TXB
003 004	XAN	005 0XX	MERG
003 007	XOFN	005 111	IAR
003 010	XAZ	005 122	IBR

Table A-2

OCTAL	MNEMONIC	OCTAL	MNEMONIC
005 144	IXR	006 24X	INRE
005 1XX	INCR	006 25X	STAE
005 211	CPA	006 26X	STBE
005 222	CPB	006 27X	STXE
005 244	CPX	006 31X	ORAE
005 311	DAR	006 32X	ADDE
005 322	DBR	006 33X	ERAE
005 344	DXR	006 34X	SUBE
005 3XX	DECR	006 35X	ANAE
005 511	AOFA	006 36X	MULE
005 522	AOFB	006 37X	DIVE
005 544	AOFX	006 4XX	BT
005 711	SOFA	006 51x	RGLD
005 722	SOFB	006 53x	RGST
005 744	SOFX	006 5XX	JSR
005 XXX	COMP	006 5xx	DMOVSD
		006 6XX	SRE
006 010	LDAI	006 71x	DJPADD
006 01X	LDAE	006 72X	JZ
006 020	LDBI	006 73X	JNZ
006 02X	LDBE	006 74X	JN
006 030	LDXI	006 75X	JP
006 03X	LDXE	006 76X	JDZ
006 040	INRI	006 77X	JDNZ
006 04X	INRE	006 7XX	IJMP
006 050	STAI		
006 05X	STAE	007 0XX	LD
006 060	STBI	007 1XX	ST
006 06X	STBE	007 2XX	AD
006 070	STXI	007 3XX	SB
006 07X	STXE	007 404	BMOVW
006 110	ORAI	007 406	STWRDS
006 11X	ORAE	007 407	STBYTS
006 120	ADDI	007 5XX	ADR
006 12X	ADDE	007 6XX	SBR
006 130	ERAI	007 7XX	T
006 13X	ERAE	007 400	ROF
006 140	SUBI	007 401	SOF
006 14X	SUBE	007 402	TSA
006 150	ANAI	007 41X	INC
006 15X	ANAE	007 42X	DEC
006 160	MULI	007 43X	COM
006 16X	MULE	007 44X	LDI
006 170	DIVI	007 45X	ADI
006 17X	DIVE	007 46X	LBT
006 21X	LDAE	007 47X	STB
006 22X	LDBE		
006 23X	LDXE		

INDEX OF INSTRUCTIONS

Table A-2

OCTAL	MNEMONIC	OCTAL	MNEMONIC
01X XXX	LDA	103 0XX	OME
02X XXX	LDB	103 1XX	OAR
03X XXX	LDX		
04X XXX	INR	103 2XX	OBR
05X XXX	STA	103 3XX	OAB
06X XXX	STB		
07X XXX	STX	104 XXX	EXC2
100 XXX	EXC		
101 XXX	SEN	106 XXX	IDE
		107 XXX	ECS
		11X XXX	ORA
102 0XX	IME	12X XXX	ADD
102 1XX	INA	13X XXX	ERA
102 2XX	INCR	14X XXX	SUB
102 3XX	INAB		
102 5XX	CIA	15X XXX	ANA
102 6XX	CIB	16X XXX	MUL
102 7XX	CIAB	17X XXX	DIV

APPENDIX B - NUMBER SYSTEMS

Digital computers use a binary number system based on a count (radix) of two. The binary number system has simpler rules than the familiar decimal (radix of 10) number system, making it ideal for computers. The electronic components that make up a digital computer are inherently binary. A relay is either opened or closed; magnetic materials (tape or core) are magnetized in one direction or another; a vacuum tube or transistor is either fully conducting or nonconducting; an electrical pulse can be transmitted at a given time or it cannot be transmitted.

Binary System

In the decimal system, we think in "tens". For example, the number 35 means: $10 + 10 + 10 + 5 = 35$. Or 35 can be written as: $3(10) + 5(1) = 35$. Or 35 can be written in positional notation as: $3(10^{1}) + 5(10^{0}) = 35$. In the pure binary system, we deal with powers of two rather than powers of 10. The positions of the digits do not have the meaning of units, tens, hundreds, thousands, etc.; instead these positions signify units, twos, fours, eights, sixteens, etc. The sum of these binary positions gives the same decimal sum.

Decimal values

32	16	8	4	2	1

Binary positional notational weight

2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Remembering that in the binary system we have only two marks, 0 and 1, we then convert the decimal number 35 to a binary number; reading from right to left, we place a one in the first, second, and sixth positions and zeros in the other three positions.

$$1(2^{0}) + 0(2^{1}) + 0(2^{2}) + (2^{3}) + 1(2^{4}) + 1(2^{5})$$

The resulting binary number is 100011, which is binary for decimal 35 ($32 + 0 + 0 + 0 + 2 + 1 = 35$).

Decimal-to-Binary Conversion

Method 1. The positional notation chart used in the example above for converting decimal 35 to a binary number suggests a method for decimal-to-binary conversions. We ask the question, what is the largest number to the power of two that can be contained in the decimal number 35. The answer is 32, or 2^{5}; we place a one in that position. We next ask which power of two can be contained in the remainder $35 - 32$, or 3). Since 2^{1} equals 2 and 2^{0} equals 1 and both are contained in the remainder 3, we place ones in those positions. Hence, binary 100011 = decimal 35.

Figure B-1 is an example of decimal-to-binary conversion using method 1.

Example

Convert decimal 943 to binary:

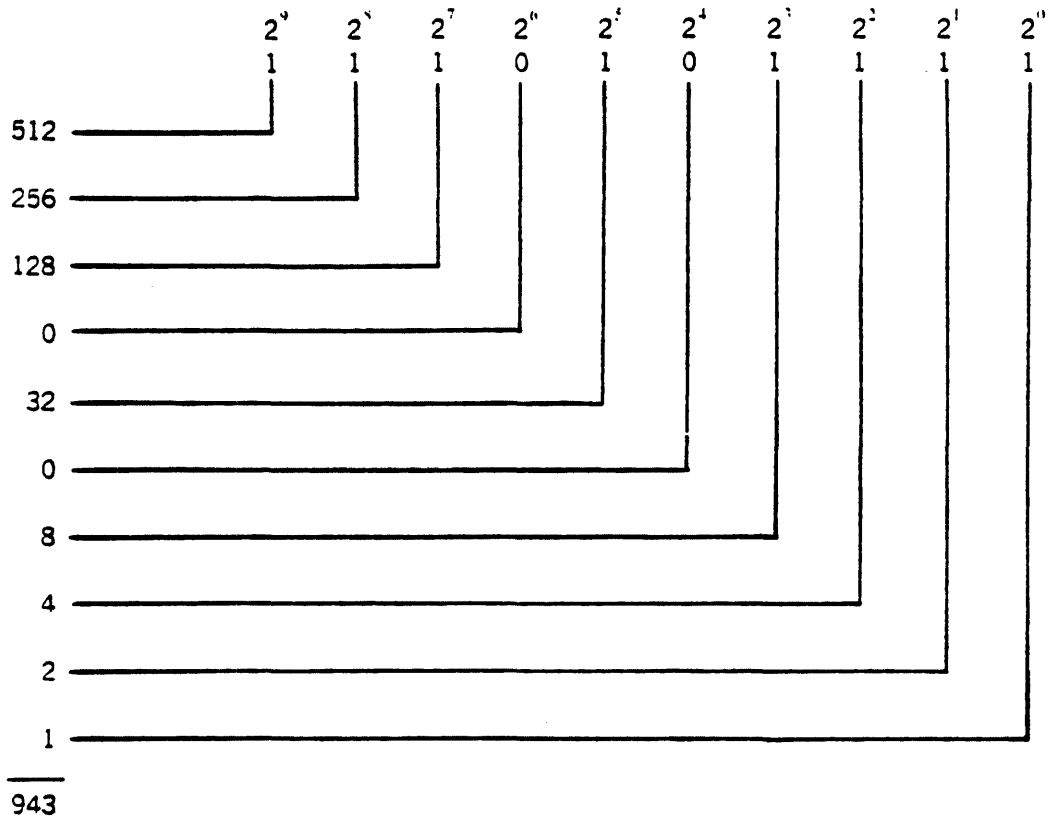


Figure B-1. Converting Decimal to Binary (Method 1)

Method 2. A decimal number can be converted to its binary equivalent by successive division of the number by two. If there is a remainder after the first division, a binary one is placed in the least significant (right-most) binary position. The occurrence or lack of a remainder after each division determines the binary state of each position.

Figure B-2 illustrates decimal-to-binary conversion using method 2.

Example

Convert decimal 135 to binary:

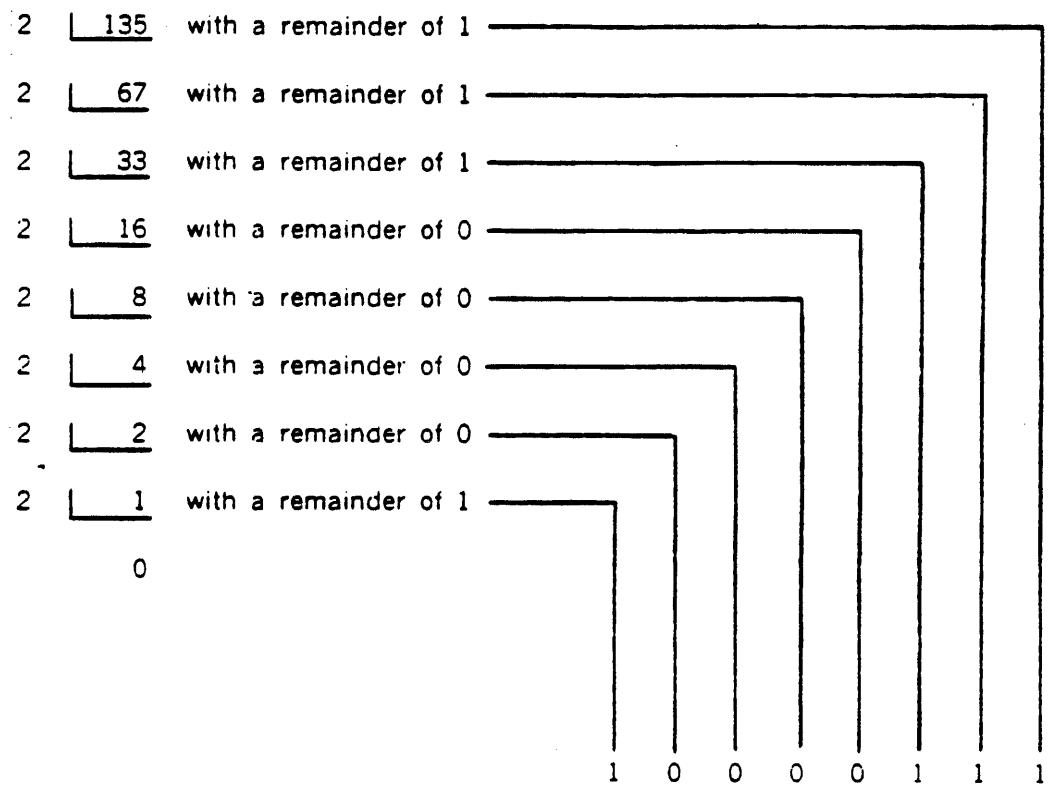


Figure B-2. Converting Decimal to Binary (Method 2)

Binary-to-Decimal Conversion

Binary numbers are converted to their decimal equivalents by multiplying each digit by two (starting with the most significant -- left-most -- digit) and adding the decimal value of the next digit to the right as illustrated in figure B-3.

Convert binary 100001 to decimal:

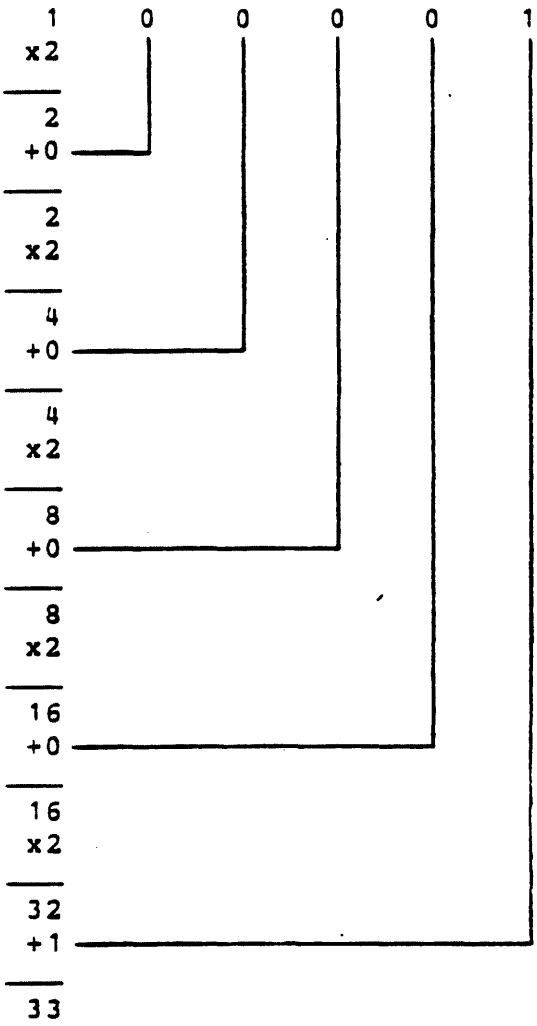


Figure B-3. Binary to Decimal Conversion

Binary Addition

Only four rules apply in binary addition:

0 + 0 = 0	1 + 0 = 1
0 + 1 = 1	1 + 1 = 10

Here 1 plus 1 is 10 (pronounced "one - oh") because binary 10 is decimal 2. This is the same as saying that decimal 1 plus 1 is 2. Following the above rules, it is possible to add any two binary numbers directly. For example, add decimal 12 and 5:

Decimal	Binary
12	01100
5	101
17	10001

To add 01011 and 00110:

Binary	Decimal
01011	11
00110	6
10001	17

In binary addition, there is the problem of the carry, as when 1 + 1 = 10 -- that is, 0 plus carry 1. This is shown by the following example, where binary 111101 is added to 10110:

(A)	111101
(B)	10110
(C)	101011
(D)	1 1
(E)	1010011

The first step is to add A and B to get the partial sum C. Line D shows the two carries resulting from the 1 + 1 sums. Adding partial sum C and the carries D produces the final sum E, or 1010011.

Binary Subtraction

Four rules apply in binary subtraction:

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

To subtract 1011 from 101101:

```
 101101
-  1011
--------
 100010
```

Note that to subtract 1 from 0, it is necessary to borrow 1, making 1 from 10, or 1.

Complements also provide a means of subtraction. In the decimal system, the ten's complement is the difference between 10 and a given number -- hence, the complement of 7 is 3. The nine's complement is the difference between 9 and a given number, the complement of 7 being 2.

By adding complements, it is possible to subtract. To subtract, using the ten's complement system:

```
  7
 +7     (ten's complement of 3:
----              10-3 = 7)
 14
```

Delete the extra digit (which occurs because of the complement), giving the remainder 4 -- just as in the decimal system 7 - 3 = 4.

Using the nine's complement system:

```
  7
 +6     (nine's complement of 3:
----      9 - 3 = 6)
 13
```

Here the extra 1 is not deleted, but is added to the 3, giving the same answer, 4. This adding of the extra digit, known as end-around carry, is a vital step in computer subtraction.

Since in the binary system there are only two digits, there can be only two complements. To find the one's complement of binary 1, subtract 1 - 1 = 0. To find the one's complement of binary 0, subtract 1 - 0 = 1. Thus, to find the complement of a binary number, change all ones to zeros and all zeros to ones; e.g., the complement of 1011 is 0100.

Thus, binary numbers can be subtracted directly:

```
 1101
-1011
------
 0010
```

And, since the complement of 1011 is 0100, subtraction is also possible by adding complements:

```
 1101
+0100
------
 10001
     1      (end-around carry)
------
 0010
```

Binary Multiplication

Four rules apply in binary multiplication:

0 x 0 = 0	0 x 1 = 0
1 x 0 = 0	1 x 1 = 1

No carries are considered in multiplication. Each digit of the multiplier is examined; when a one is found, the multiplicand is added to the result. When a zero is found in the multiplier, zeros are added to the result. The multiplicand is shifted left one digit for each multiplier digit.

Binary multiplication is thus a series of shifts and additions, as in the decimal system. For example, to multiply 100101 by 101:

```
   100101
      101
-----------
   100101
   00000     (shift left, no add)
100101       (shift left and add)
-----------
10111001     (sum)
```

For every 1 in the multiplier (101), the multiplicand (100101) is moved one place to the left and added. For every 0 in 101, there is one shift but no addition.

Binary Division

By applying the concepts of binary addition, subtraction, and multiplication, we can divide binary numbers. The divisor is subtracted from the dividend, and a 1 is placed in the quotient. If the divisor cannot be subtracted, a 0 is placed in the quotient.

To divide 101 into 1101010:

```
          10101
      ------------
101 | 1101010
      101
      ---
        110
        101
        ---
          110
          101
          ---
            1   (remainder)
```

Binary-Coded Decimal (BCD) System

This system for representing decimal numbers expresses each decimal digit by a four-digit code called a word) written in binary notation:

BCD		Decimal
0000	=	0
0001	=	1
0010	=	2
0011	=	3
0100	=	4
0101	=	5
0110	=	6
0111	=	7
1000	=	8
1001	=	9
0001 0000	=	10

Thus, decimal 1971 would be expressed in BCD as:

0001	1001	0111	0001
1	9	7	1

Octal System

The octal system of assigning numerical values to binary forms is useful as a shorthand method of writing pure binary numbers. The octal system deals with groups of three binary positions; each group is considered a single digit. This means that, in any octal digit, there is a possibility of eight different binary configurations:

Binary		Octal
000	=	0
001	=	1
010	=	2
011	=	3
100	=	4
101	=	5
110	=	6
111	=	7

Given a series of binary digits, the first three to the left of the binary point are represented by the decimal notation 1, 2, 3.....7 x 8^0, the next three digits in order are represented decimally by 1, 2, 3.....7 x 8^1. As can be seen, each group of three binary bits represents some number (from 0 to 7) multiplied by a positional power of eight.

A binary number can be converted without using octal notation; however, the process requires the addition of seven quantities, instead of the three quantities in octal notation. To avoid confusion, octal numbers are designated with a leading zero, e.g., 0173.

Octal-to-Decimal Conversion

Octal representation can be converted to its decimal equivalent by multiplying each digit by eight (starting with the most significant -- left-most -- digit) and adding the decimal value of the next digit to the right as illustrated below.

Convert 0207 to decimal:

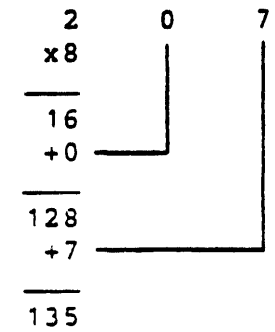


Figure B-4 shows the relationship of a binary number to its octal and decimal equivalents.

Binary Groups	001		111		011	
Octal Notation	1		7		3	
Octal Equivalents	($1x8^2$)		($7x8^1$)		($3x8^0$)	
Decimal Equivalents	64	+	56	+	3	= 123

Figure B-4. Relationship of Binary, Octal, and Decimal

Decimal-to-Octal Conversion

A decimal number can be converted to its octal equivalent by dividing the decimal number by eight and developing the octal number from the remainder as illustrated in figure B-5.

Convert decimal 135 to octal:

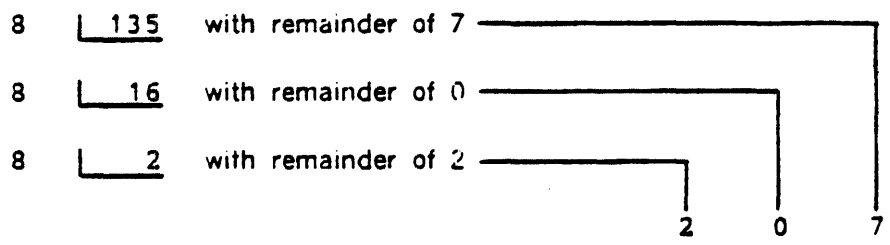


Figure B-5. Decimal to Octal Conversion

Hexadecimal System

Hexadecimal data are expressed to the radix (base) 16 and are related to the decimal numbers as follows:

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Thus, hexadecimal numbers proceed from 0 through F (0 through 15 decimal), 10 through 1F (16 through 31 decimal), 20 through 2F (32 through 47 decimal), etc. To avoid confustion, hexadecimal numbers are designated with a leading dollar sign, e.g., $0F3C.

Hexadecimal-to-Decimal Conversion

A hexadecimal number can be converted to its decimal equivalent by expanding each position. Figure B-6 illustrates hexadecimal-to-decimal conversion.

$$
\begin{aligned}
\$55F &= (5 \times 16^2) + (5 \times 16^1) + (15 \times 16^0) \\
&= (5 \times 256) + (5 \times 16) + (15 \times 1) \\
&= 1280 + 80 + 15 \\
&= 1375
\end{aligned}
$$

Figure B-6. Hexadecimal-to-Decimal Conversion

APPENDIX C - POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

APPENDIX D
V70 SERIES ASCII CHARACTER CODES

Octal	Decimal	Character	029	026	Description
200	128	NUL			Null
201	129	SOH			Start of Heading
202	130	STX			Start of Text
203	131	ETX			End of Text
204	132	EOT			End of Transmission
205	133	ENQ			Enquiry
206	134	ACK			Acknowledge
207	135	BEL			Bell
210	136	BS			Backspace
211	137	HT			Horizontal Tab
212	138	LF			Line Feed
213	139	VT			Vertical Tab
214	140	FF			Form Feed
215	141	CR			Carriage Return
216	142	SO			Shift Out
217	143	SI			Shift In
220	144	DLE			Data Link Escape
221	145	DC1			Device Control 1
222	146	DC2			Device Control 2
223	147	DC3			Device Control 3
224	148	DC4			Device Control 4

V70 SERIES ASCII CHARACTER CODES

Octal	Decimal	Character	029	026	Description
225	149	NAK			Negative Acknowledge
226	150	SYN			Synchronous File
227	151	ETB			End of Transmission Block
230	152	CAN			Cancel
231	153	EM			End of Medium
232	154	SUB			Substitute
233	155	ESC			Escape
234	156	FS			File Separator
235	157	GS			Group Separator
236	158	RS			Record Separator
237	159	US			Unit Separator
240	160	SP	(blank)	(blank)	Space
241	161	!	11/2/8	11/2/8	Exclamation Point
242	162	"	7/8	0/5/8	Quotation Mark
243	163	#	3/8	0/7/8	Pound Sign
244	164	$	11/3/8	11/3/8	Dollar Sign
245	165	%	0/4/8	11/7/8	Percent Sign
246	166	&	12	12/7/8	Ampersand
247	167	'	5/8	4/8	Apostrophe
250	168	(	12/5/8	0/4/8	Left Paren
251	169	)	11/5/8	12/4/8	Right Paren
252	170	*	11/4/8	11/4/8	Asterisk
253	171	+	12/6/8	12	Plus Sign
254	172	,	0/3/8	0/3/8	Comma

Octal	Decimal	Character	029	026	Description
255	173	-	11	11	Minus Sign
256	174	.	12/3/8	12/3/8	Period
257	175	/	0/1	0/1	Slash
260	176	0	0	0	
261	177	1	1	1	
262	178	2	2	2	
263	179	3	3	3	
264	180	4	4	4	
265	181	5	5	5	
266	182	6	6	6	
267	183	7	7	7	
270	184	8	8	8	
271	185	9	9	9	
272	186	:	2/8	5/8	Colon
273	187	;	11/6/8	11/66/8	Semi-Colon
274	188	<	12/4/8	12/6/8	Less Than
275	189	=	6/8	3/8	Equal Sign
276	190	>	0/6/8	6/8	Greater Than
277	191	?	0/7/8	12/2/8	Question Mark
300	192	@	4/8	0/2/8	At
301	193	A	12/1	12/1	
302	194	B	12/2	12/2	
303	195	C	12/3	12/3	
304	196	D	12/4	12/4	

V70 SERIES ASCII CHARACTER CODES

Octal	Decimal	Character	029	026	Description
305	197	E	12/5	12/5	
306	198	F	12/6	12/6	
307	199	G	12/7	12/7	
310	200	H	12/8	12/8	
311	201	I	12/9	12/9	
312	202	J	11/1	11/1	
313	203	K	11/2	11/2	
314	204	L	11/3	11/3	
315	205	M	11/4	11/4	
316	206	N	11/5	11/5	
317	207	O	11/6	11/6	
320	208	P	11/7	11/7	
321	209	Q	11/8	11/8	
322	210	R	11/9	11/9	
323	211	S	0/2	0/2	
324	212	T	0/3	0/3	
325	213	U	0/4	0/4	
326	214	V	0/5	0/5	
327	215	W	0/6	0/6	
330	216	X	0/7	0/7	
331	217	Y	0/8	0/8	
332	218	Z	0/9	0/9	
333	219	[	12/2/8	12/5/8	Left Bracket
334	220	\	11/7/8	0/6/8	Backslash

Octal	Decimal	Character	029	026	Description
335	221	]	0/2/8	11/5/8	Right Bracket
336	222	↑ or ^	12/7/8	7/8	Vertical Arrow
337	223	← or –	0/5/8	2/8	Horizontal Arrow
340	224				Accent Grave
341	225	a			
342	226	b			
343	227	c			
344	228	d			
345	229	e			
346	230	f			
347	231	g			
350	232	h			
351	233	i			
352	234	j			
353	235	k			
354	236	l			
355	237	m			
356	238	n			
357	239	o			
360	240	p			
361	241	q			
362	242	r			
363	243	s			
364	244	t			

V70 SERIES ASCII CHARACTER CODES

Octal	Decimal	Character	029	026	Description
365	245	u			
366	246	v			
367	247	w			
370	248	x			
371	249	y			
372	250	z			
373	251	{			Left Brace
374	252	|			Vertical Line
375	253	}			Right Brace
376	254	~			Sine Curve
377	255	DEL			Delete, Rub Out