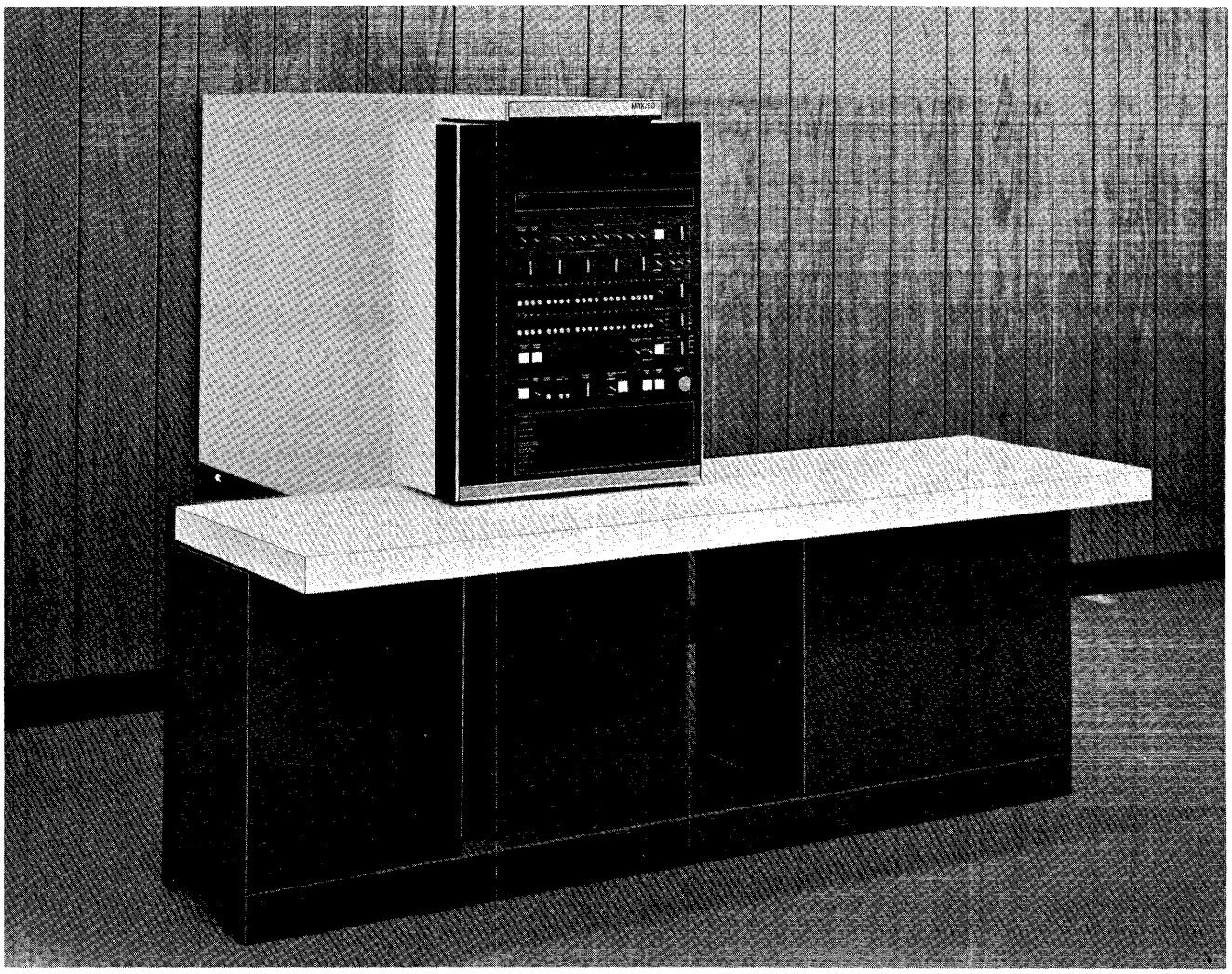
MEMOREX

MRX/40/50 Systems
7200/7300 Computers Product Description



COMPANY CONFIDENTIAL

Document 2999.001
March 1972

FOR INTERNAL USE ONLY

PREFACE

This document describes hardware aspects of the MEMOREX 7200/7300 Computers at the design level of March, 1972. It completely replaces the previous versions.

This product description contains information for programmers, engineers, field support and marketing personnel, and other Memorex personnel concerned with a general overview of the processor's capabilities of the computer; discuss its functional characteristics and principles of operation, including instruction descriptions, and describe the System Control Panel and its use.

It is recognized that changes are occurring, and potential changes are being evaluated; present and future inaccuracies will be treated in later publications.

The Product Description is intended to serve as a source of official information for Memorex personnel until it is supplanted by the series of 7200 and 7300 Computer manuals now being generated. Comments may be sent to the Publications Department, Midwest Operations, Memorex Corporation, 9200 Science Center Drive, Minneapolis, Minnesota, 55428, for use in preparing these future publications.

SPECIAL MRX MARKETING EDITION

This abridged edition of the 7200/7300 computers Product Description has been prepared from more extensive Memorex internal development documentations.

SECTIONAL TABLE OF CONTENTS

SECTION I
GENERAL DESCRIPTION

TABLE OF CONTENTS

GENERAL DESCRIPTION

SECTION I
GENERAL DESCRIPTION

INTRODUCTION

An overview of the MRX/40/50 data-processing systems is provided in Section 1 of this product description. The remainder of this manual is devoted to the detailed description of the functional and logical characteristics of the MEMOREX 7200 and 7300 computers. Appendix A tabulates the performance differences between the 7200 and 7300. While peripheral devices are referred to throughout, they are not discussed in detail.

The MRX/40 and 50 are low-cost communications-oriented systems with large data base capacity. A wide range of storage sizes, peripheral devices, and integrated adapters affords maximum flexibility in tailoring a system to meet a user's specific need. MRX/40/50 are supported by an unusually comprehensive operating system developed by Memorex. This extensive programming systems support permits the user to concentrate on his application, rather than on the functions of the system. The combination of hardware and software capability provides a price/performance level normally associated with more costly data processing systems. The result is a more efficient and economical data-processing system for the user.

Several characteristics distinguish the MRX/40/50 computer systems:

- Advanced architecture
- Wide range of peripheral devices
- Extensive communications support
- Comprehensive programming systems support

ARCHITECTURE

The MEMOREX 7200 and 7300 offer advanced hardware design at all levels, from monolithic circuit components to architecture. The use of monolithic components produces a high-speed machine cycle. Miniaturized circuits permit the concentration of great computational power in a physically small cabinet. They also reduce the number of required components, thus decreasing the potential sources of hardware failure; the result is a high degree of reliability. The 7200/7300 printed-circuit boards are packaged and arranged by function, which makes fault isolation and correction faster.

Main storage is implemented through the use of metal-oxide semiconductors (MOS) which allow for greater packing density of components and lower memory costs. Storage sizes of the 7200/7300 range from 16K to 128K bytes (each byte contains 8 bits plus a parity bit).

The 7200 and 7300 are microprogrammed computers. All machine language instructions are implemented through the use of microprogramming instructions residing in Control Storage. These micro-instructions perform within the much shorter cycle time of Control Storage. Microprograms, automatically loaded at Initial Program Load (IPL) time, are provided by Memorex; they reside as a small reserved portion of a disc pack and require no user intervention or maintenance.

The design of the 7200/7300 computers is such that certain portions of the hardware are dedicated to specific functions such as communications input-output, direct-access input-output, and arithmetic functions. A full set of eight general registers is dedicated for each such use by the operating system. Three additional sets of eight general registers are dedicated for user programs. The use of these various hardware functions is controlled by the operating system which interfaces to a hardware-controlled priority network.

The 7200/7300 incorporate integrated adapters for control of disc storage, communications, card readers, and card punches. Their use eliminates the need for large external control units, and thus provides for a lower-cost system.

7200/7300 computer features include these:

- Alterable Control Storage (4K words)
- Up to 128K bytes of Main Storage
- Error Correction Code (ECC) option for Main Storage
- Job Accounting Aids
- Interval Timer

- Storage Protection
- Relocation and Protection
- Integrated Console Adapter (standard)
- Integrated Communications Adapter (ICA) servicing up to 15 communications channels
- Integrated File Adapter (IFA) for up to eight MEMOREX 3664 disc drives.
- Integrated adapters for the following:

  - up to 1000 cpm card readers (80-column)
  - up to 500/120 cpm reader/punch (80-column)

- Printer attachment for up to 1200 lpm printer
- Up to two selector channels for supporting external control units.

PERIPHERAL DEVICES

An unusually wide range of peripheral devices to solve a variety of application needs can be accommodated by the MRX/40/50 systems:

- Card equipment (80-column cards).
- Printers may range in speed up to 1200 lines per minute.
- From one to eight disc drives may be used, each with a capacity of 14 or 28 megabytes. This adds up to a potential of 224 million bytes of on-line storage.
- Magnetic tape transfer rates may be up to 60 kilobytes per second.

COMMUNICATIONS SUPPORT

The MRX/40/50 systems have low-cost communications hardware, combined with simplified communications software. Up to 15 communications lines are provided for operations in asynchronous or synchronous modes. Other features include autocall, autoanswer, speed selection, autopoll, station select, and synchronous transparency.

For terminals, the user may choose from the MEMOREX 1200 series or certain IBM* and Teletype* terminals. Networks ranging from small to relatively large may be configured for the MRX/40/50.

A logical-level macro language is provided to handle both asynchronous and synchronous transmissions. Included are the standard functions such as READ, WRITE, GET, PUT, and code translate. Messages are queued in memory. A special communications monitor controls multiple terminals and multiple transactions for the user-written application programs.

PROGRAMMING SYSTEMS SUPPORT

The MRX/40/50 systems are supported by a level of programming systems normally only available with large-scale systems. The operating system provided for the MRX/40/50 is a comprehensive multi-programming system which includes the following features:

- Cataloged job control statements
- Input spooling
- Output spooling
- Complete data management
- Extensive utility programs
- High-performance disc sort
- Advanced library facilities
- Modular systems generation
- IBM* System/360 Model 20 compatibility

Four major programming languages are available for MRX/40/50: Assembler, ANSI COBOL, RPG II and FORTRAN IV. The Assembler language has complete macro capability. COBOL includes many significant extensions. The RPG II language is designed so that most IBM 360/20 and System/3 programs will recompile and execute without modification. All languages include the ability to exit to assembler language subroutines, and programs can be segmented.

---

*Tradenames, International Business Machines Corporation and the Teletype Corporation.

SECTION II
FUNCTIONAL DESCRIPTION

TABLE OF CONTENTS

FUNCTIONAL DESCRIPTION

FIGURE TITLES

SECTION II
FUNCTIONAL DESCRIPTION

INTRODUCTION

This section contains information intended to provide program implementation understanding for the MRX/40/50 system (MEMOREX 7200 and 7300 computers respectively). It does so by further explaining the system architecture (hardware and software concept and relationship) and describing data and instruction formats, storage addressing, and principles pertinent to program implementation.

SYSTEM ARCHITECTURE

Data processing systems divide their time between input/output operations and arithmetic/logical functions. This conflict of interest usually causes large periods of system time to be dominated by input/output functions. Obviously, while this domination exists, hardware such as that dedicated to arithmetic/logical functions stands idle. This results in uneconomical time-versus-hardware usage.

To avoid this uneconomical usage, the 7300 computer employs a multiplex technique which divides one complete *machine* cycle into eight segments called "major cycles". Along with this time division, a design concept is implemented dedicating those resources (hardware) which are function oriented to one of these time segments. Figure 2-1 illustrates this combination of one *major cycle* and a particular set of dedicated resources as a "processor state". These *processor states* are

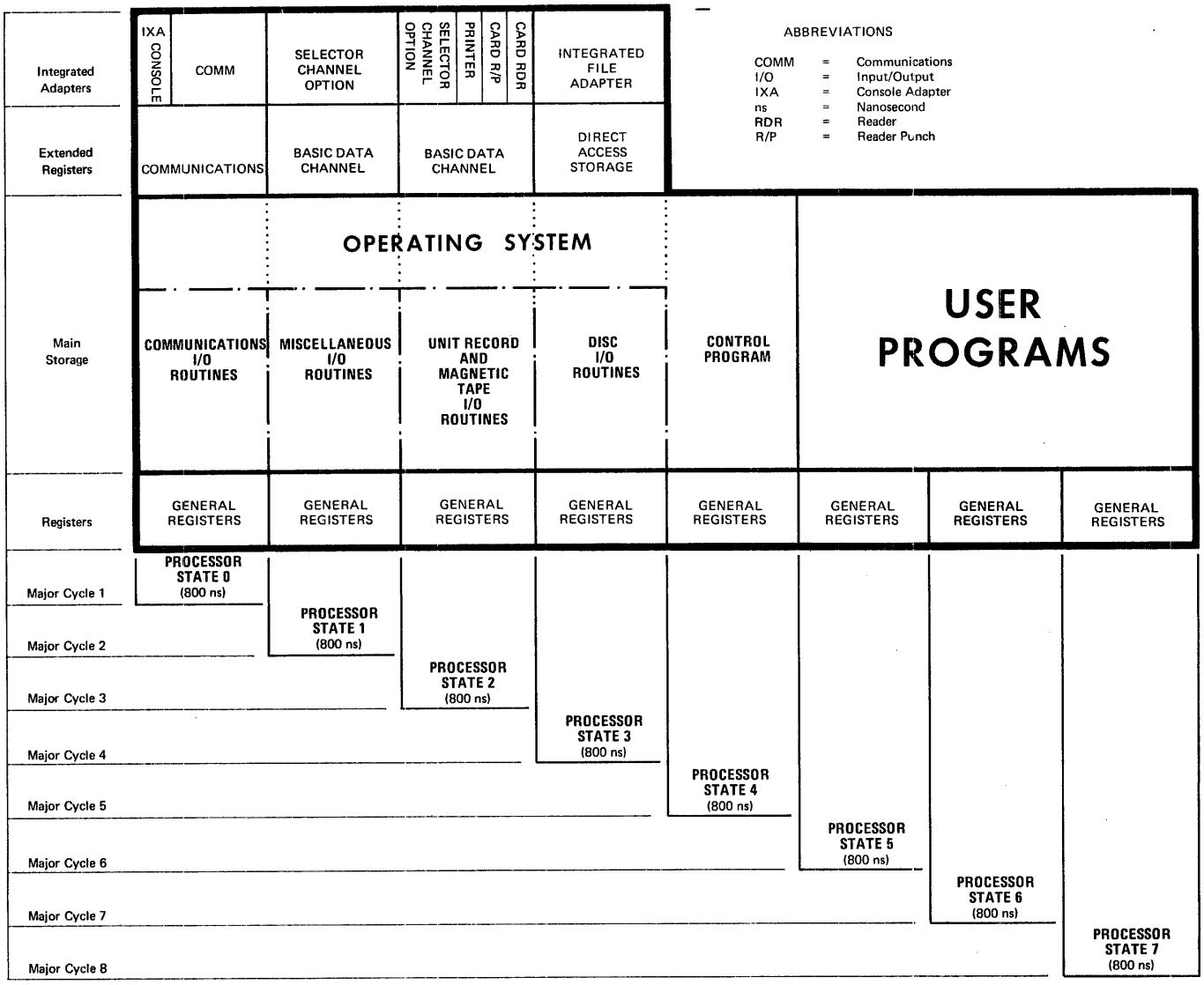


Figure 2-1. System Architecture

functionally oriented and during the time they exist have exclusive access to what is termed "shared resources". *Shared resources* are those elements of hardware, such as an arithmetic section, which are common to all processors.

Input/Output

The first consideration in any data-processing system is how well it can transmit or receive data. As shown in Figure 2-1, the 7300 computer has functionally oriented integrated adapters and hardware as described by the following paragraphs.

- There is an integrated communications adapter capable of controlling up to 15 independent communications lines. These communications lines are connected through the adapter to the communications processor.

  The integrated adapter (IXA) for the system console also uses the communications processor.

- There are two basic data channel processors.
  1. The card reader, card reader/punch, and printer connect to the basic data channel via their respective adapters. Up to seven external control units connect to the basic data channel via the first selector channel option.

  2. Up to eight additional external control units may be attached to the second selector channel.

- To accommodate disc storage devices, an integrated file adapter is provided connecting up to eight disc drives to the disc processor.

Computer Hardware

Each of the four input/output processors has a set of general registers. These registers are used by the input/output routines for their respective function.

Four additional sets of general registers are illustrated in Figure 2-1. One of these register sets is dedicated to the control programs of the operating system. The remaining three sets of general registers are dedicated to executing user programs.

Hardware timing for the 7300 computer divides the total computer cycle into eight segments (major cycles). During each major cycle, specific functionally dedicated resources (hardware) perform in conjunction with shared resources to become a processing entity. Figure 2-1 illustrates this period of time as a "processor state". One of these major cycles is allotted to each processor state.

Priority

As previously mentioned, the hardware machine timing is divided into major cycles, allocated one cycle per processor state. Figure 2-1 illustrates the first major cycle being allocated to processor state 0 (communications processor). In other words, for that cycle the computer is operating the communications hardware. During the next major cycle, processor state 1 (a multi-device processor) hardware is in operation. This continues through processor states 2, 3, 4, 5, 6, and 7.

If, as the diagram shows, all processor states have work to perform of equal priority, these cycles occur sequentially. Processor states performing high-speed input-output operations need priority over those performing lower-speed input-output or general computational functions. For this purpose, a priority scheme is provided to take care of any I/O processor state that needs major cycles sooner than its normal turn in the sequence. This priority scheme can be adjusted at the end of each major cycle; that is, every major cycle a test is made to see if the normal priority (sequential processor-state operation) is to be followed, or if an I/O processor should be given an out-of-sequence cycle. This action is similar to that of a system that operates on a priority-interrupt system.

Storage Protection

This feature provides main storage protection for each of the processor states 5, 6, and 7. A register designated Upper Bounds (UB) and Lower Bounds (LB) is provided for each of these processor states. These registers are accessible only by processor state 4.

Upper and Lower Bounds values are set up by the control program (processor state 4). Each writing access attempted by an instruction executing under processor states 5, 6, or 7 is then checked by the Storage Protection hardware against the UB and LB values for that processor state. If the address falls outside the bounds values, writing is prevented and the control program detects a problem program addressing error.

Computer Options

The following paragraphs describe optional features of the 7300 computer.

JOB ACCOUNTING

This option comprises eight 32-bit registers, one for each processor state, and an incremental adder. Each time a processor state gets a major cycle, the hardware automatically increases the contents of the associated Job Accounting register by 1. A job accounting register may be read at any time, but any attempt to write into it will reset it to zero.

RELOCATION AND PROTECTION

The Relocation and Protection feature provides three functions:

1. Allows for relocation of programs without instruction modification
2. Extends the addressing capability of the system from 65K to up to one-million bytes
3. Provides a comprehensive storage protection scheme for all processor states.

This feature is implemented through the use of a Segment Relocation Table, Protection Matrix, Segment Tag File, and Address Mode Register.

Relocation. Each processor state has a hardware "segment tag" associated with its program address (P) register and with each of its general registers. This segment tag identifies an entry in the Segment Relocation Table, from which a relocation constant or lower boundary page number of the storage segment (or block) is obtained. Each entry in the Segment Relocation Table also contains an upper boundary number for that segment.

The effective address presented by each processor to the relocation and protection hardware is called the system address. The 20-bit system address consists of two parts: a 4-bit Segment Number and a 16-bit displacement within the segment. The displacement consists of an 8-bit Page Number (identifying blocks of 256 bytes each) and an 8-bit byte displacement.

The relocation and protection hardware uses the segment tag for access to the appropriate entry in the Segment Relocation Table. The 16-bit displacement is then added to the relocation constant to form the physical memory address used in the instruction. (The 8-bit displacement is never altered. Figure 2-2.)

The general rule for obtaining a Segment Number is this: if there is a general register involved in the formation of the address, its associated segment tag is used; if there is no general register so involved, the segment tag associated with P-Register is supplied.

Protection. Storage may be divided into a maximum of 16 segments; access to each is controlled through the Protection Matrix. This matrix is organized as shown in Figure 2-3. Each processor has two rows assigned to it; one for READ and one for WRITE. Access by a processor state for READ or WRITE to a specific segment is allowed only if the bit is 0 in the column for the correct segment corresponding to the row for that processor state.

If access is to be permitted, a check is also performed by the relocation and protection hardware to assure that the page number of the system address displacement does not exceed the "upper bounds" entry in the Segment Relocation Table for that segment. "Upper bounds" is the maximum page number which may be addressed within that segment. If the bounds is exceeded, the instruction will not be executed and a condition code will be set for the processor state.

Control. Control of the Relocation and Protection feature is supported through special machine language instructions and the Address-Mode Register. Access to these is controlled by the control program (processor state 4).

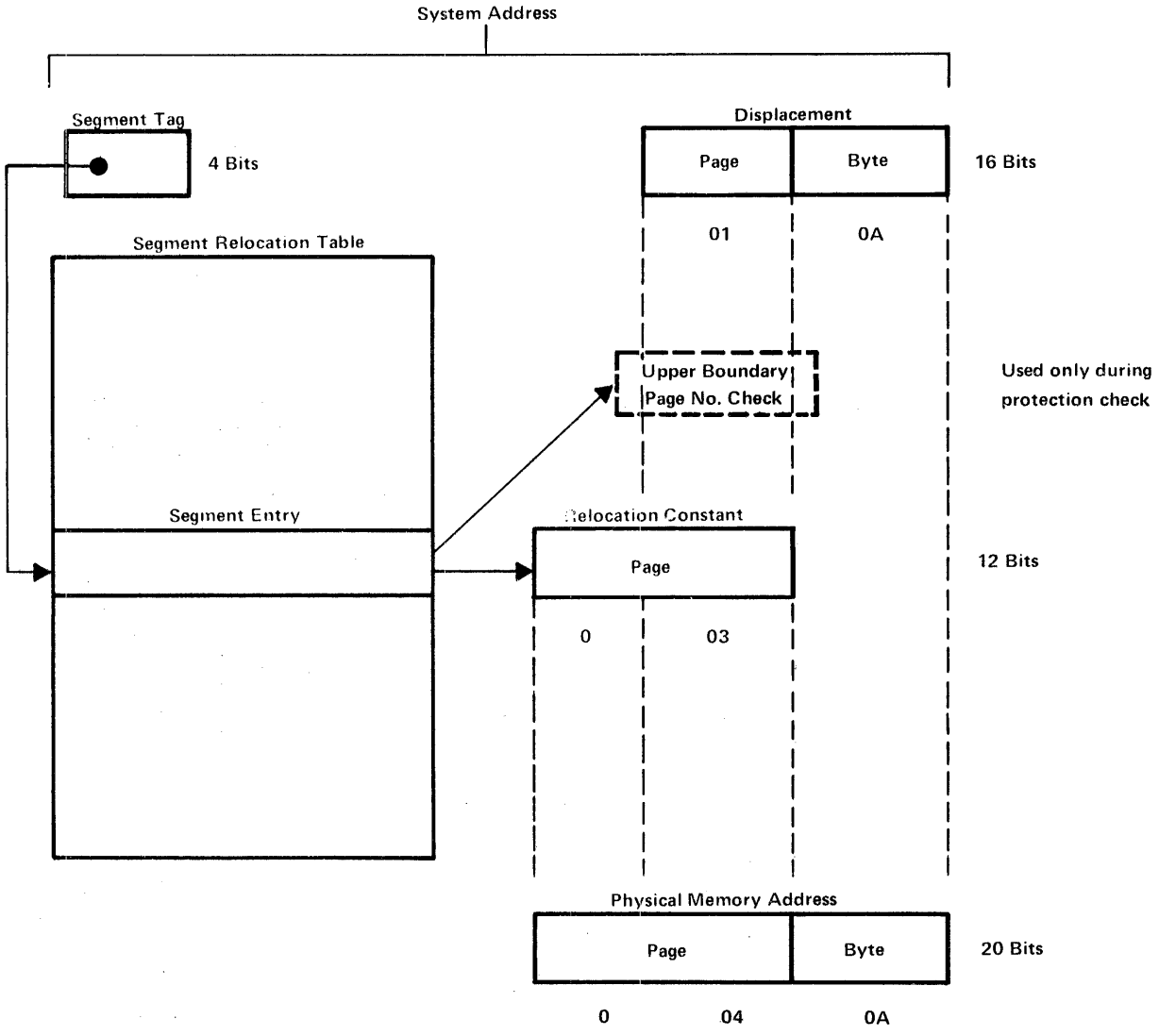


Figure 2-2. Generating Relocation Address

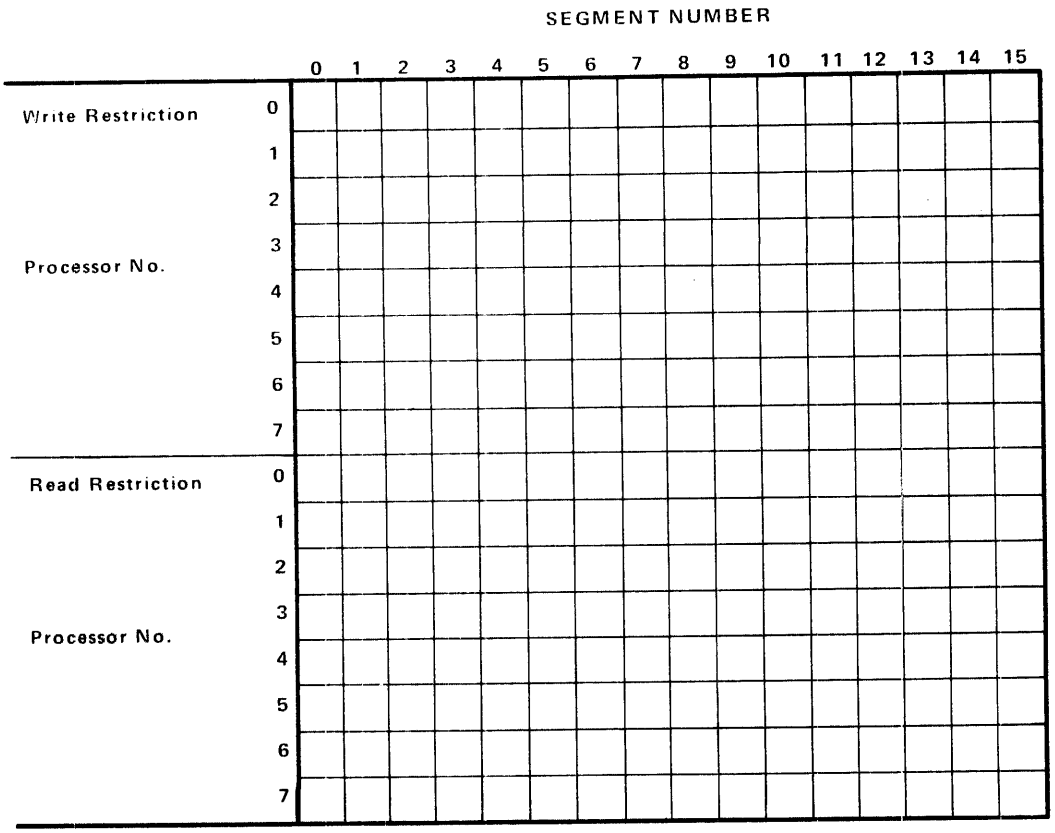


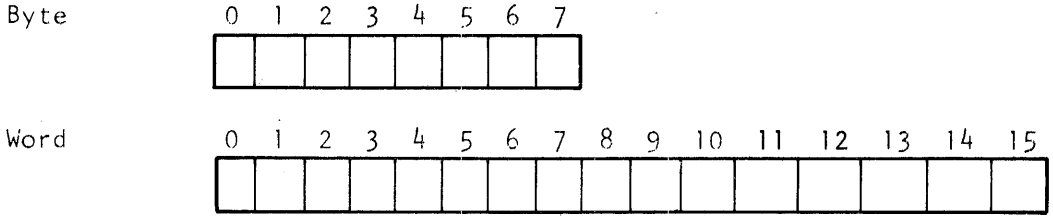
Figure 2-3. Storage Access Protection Matrix

ERROR CORRECTION CODE OPTION

The Error Correction Code (ECC) option corrects single-bit errors, and detects but does not correct double-bit errors made during a storage reference. In either case, an entry is made in the ECC Error Log register.

DATA REPRESENTATION

The basic unit of memory for the MEMOREX 7300 processor is an 8-bit byte. The bytes in memory are numbered from 0 to a maximum of 131,072 (128K). The computer transfers two bytes at a time into or out of memory; these two bytes are treated as a unit, called a word. The bits in a byte or word are numbered left to right from 0 through 7 or 0 through 15, respectively.



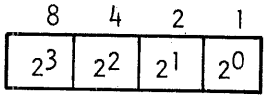
Data can be represented in any of the following formats: hexadecimal, EBCDIC, binary, and zoned or packed decimal.

Hexadecimal Representation

Hexadecimal notation is a system in which four bits are used to represent the decimal values, 0-15. Since the hexadecimal system implies 16 different numbering symbols, this system uses digits 0-9 and letters A, B, C, D, E, and F for decimals 10, 11, 12, 13, 14, and 15, respectively.

Each hexadecimal symbol has a corresponding 4-bit pattern. This system uses every possible combination of the four bits (16 combinations).

For example, the value of each of four bits is shown, representing the equivalents of 2^3, 2^2, 2^1, and 2^0.



Binary

0	1	0	1	4 + 1 = 5
1	0	1	0	8 + 2 = A
1	1	1	1	8+4+2+1 = F

The following table shows the correlation between decimal, hexadecimal, and binary notation.

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

Using hexadecimal notation, a direct conversion from binary to hexadecimal, or vice-versa, is very easily made.

Hexadecimal	9		8		E		B	
Binary	1 0 0 1		1 0 0 0		1 1 1 0		1 0 1 1	

EBCDIC Representation

Extended Binary-Coded-Decimal Interchange Code is a system that uses an 8-bit code to represent 256 possible characters (256 possible variations of eight bits). The EBCDIC system not only represents digits 0-9, but also the letters of the alphabet (upper and lower case), all punctuation and arithmetic symbols (such as + - ; > =), and some special graphic and control symbols (such as SMM - start of manual message, and EOT - end of transmission).

Each character (letter, digit, or other symbol) is represented in one byte (eight bits). The byte is divided into two portions of four bits each:

Bits 0-3 Bits 4-7

Some examples of EBCDIC characters and their corresponding 8-bit codes are:

1	1	0	0	0	0	0	1	= A
1	1	0	1	0	0	0	1	= J
1	1	1	1	0	0	0	1	= 1
1	1	1	1	1	0	0	1	= 9
0	1	1	0	1	1	0	0	= %
0	0	1	1	0	1	1	1	= EOT (end of transmission)

Numeric fields in EBCDIC are in a zoned decimal format.

The following chart shows the entire set of EBCDIC symbols with the hexadecimal equivalents of the zone and digit portions.

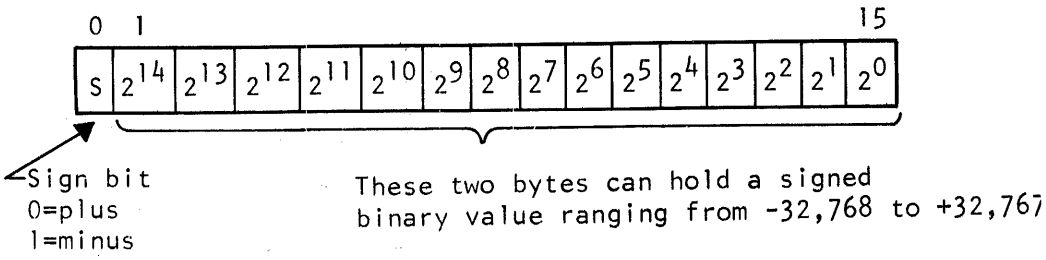
Bits 4-7
Hexadecimal
and Binary
Representation

Bits 0-3
Hexadecimal
and Binary
Representation

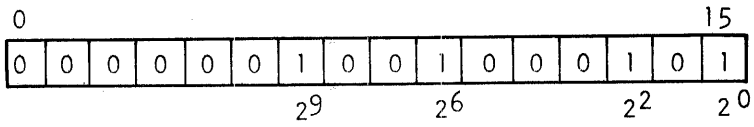
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000	NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
1 0001	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
2 0010	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
3 0011			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
4 0100	SP										¢		<	(	+	|
5 0101	&										!	$	*	)	;	¬
6 0110	-	/										´	%	_	>	?
7 0111											:	#	@	'	=	"
8 1000		a	b	c	d	e	f	g	h	i						
9 1001		j	k	l	m	n	o	p	q	r						
A 1010			s	t	u	v	w	x	y	z						
B 1011																
C 1100		A	B	C	D	E	F	G	H	I						
D 1101		J	K	L	M	N	O	P	Q	R						
E 1110			S	T	U	V	W	X	Y	Z						
F 1111	0	1	2	3	4	5	6	7	8	9						

Binary Representation

A word (two bytes) in a signed binary format has a sign bit (bit 0) and 15 bits in which to hold a value (bits 1-15). The value of each bit increases from right to left by a power of two.

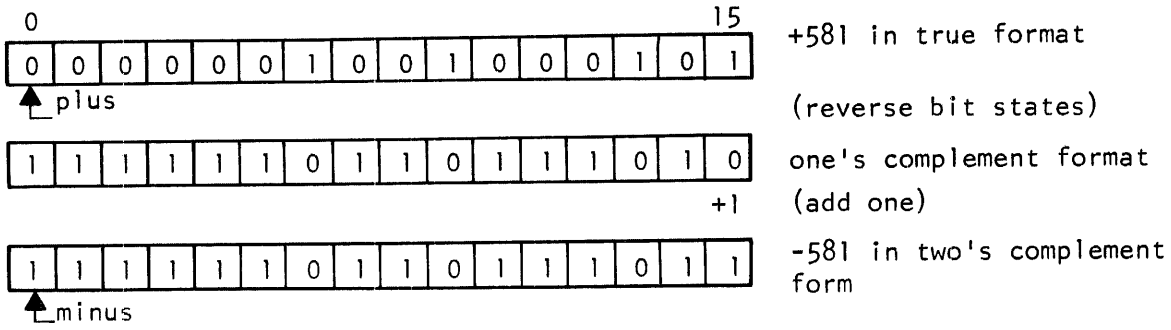


To determine the decimal value of a binary field, the decimal equivalents are added for each bit position that is on (1).



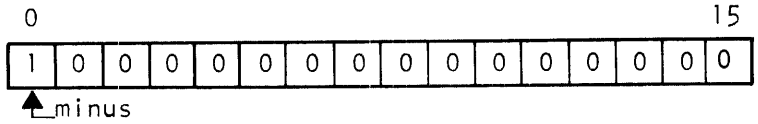
2^{9} = 512	512
2^{6} = 64	64
2^{2} = 4	4
2^{0} = 1	1
	581 = decimal value of field

Positive numbers are in the format described in the preceding text; each bit position stands for a specific value. Negative numbers are in a two's complement form. The two's complement of a value is formed by reversing the state of each bit (this is the one's complement form) and then adding 1 to the entire field(this is the two's complement form).



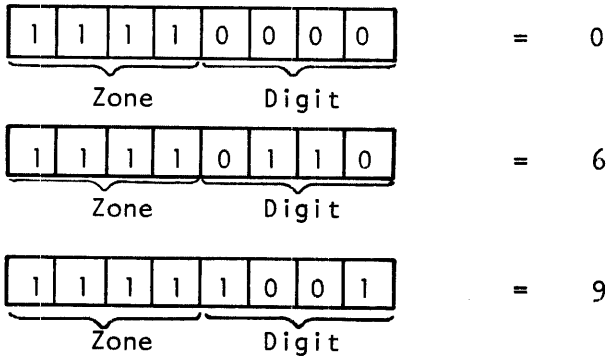
The highest possible number in two's complement form is +32,767.

The lowest possible number in two's complement form is -32,768.

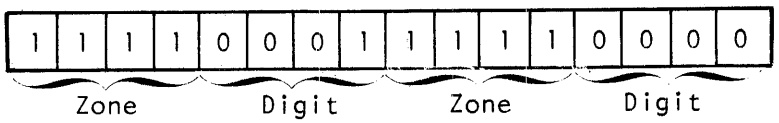


Zoned Decimal Format

In the zoned decimal format, each digit 0-9 is held in one byte. Bits 4-7 of a byte hold the digit 0-9 in the standard binary format; this is called the digit portion of a byte. Bits 0-3 of a byte are always 1111_{16}; this is called the zone portion of the byte.

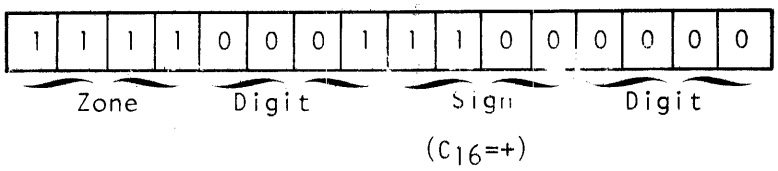


Since each digit requires a byte, a three-digit field requires three bytes, a seven-digit field requires seven bytes, and so forth. The number 10 would occupy two bytes.



The digit portions equal 1 and 0, or 10.

All numeric fields must have a sign, and in the zoned decimal format the first four bits (0-3) of the rightmost byte are used to hold a sign. A hexadecimal C is a plus sign, and a hexadecimal D is a minus sign. The number +10 would appear as follows:



Packed Decimal Representation

In the zoned decimal format, a byte is required to hold each digit, 0-9. The zone portion of each byte is unused (except for the sign in the rightmost byte). These zoned decimal fields can be packed into a fewer number of bytes if these zones are removed; thus a packed or unzoned decimal format. In the packed format, the 4-bit sign is moved to the last four bits of the rightmost byte.

The following illustration shows the number 18,634 in a zoned format and a packed (unzoned) format. Note the difference in placement of the sign. For easier interpretation, the actual bit patterns are not shown.

Zoned

Z	1	Z	8	Z	6	Z	3	+	4
Zone	Digit	Zone	Digit	Zone	Digit	Zone	Digit	Sign	Digit

Packed

1	8	6	3	4	+
Digit	Digit	Digit	Digit	Digit	Sign

If a field with an odd number of digits is packed, no portion of the packed field is unused. If a field with an even number of digits is packed, however, the first four bits of the packed field are unused, and are filled with zeros.

In the last example, 18,634 was converted from a five-byte zoned field to a three-byte packed field. For the number 618,634, the zoned field requires six bytes and the packed field four bytes. The zoned and packed fields would appear as follows. The unused portion of the leftmost byte of the packed field is considered zero.

Zoned

Z	6	Z	1	Z	8	Z	6	Z	3	+	4
Zone	Digit	Zone	Digit	Zone	Digit	Zone	Digit	Zone	Digit	Sign	Digit

Packed

0	6	1	8	6	3	4	+
Zero	Digit	Digit	Digit	Digit	Digit	Digit	Sign

In packed decimal format, there are several 4-bit patterns for plus and minus. The following 4-bit patterns, identified in hexadecimal notation, indicate plus or minus.

Plus: F, A, C, E, 0, 2, 4, 6, 7, or 8

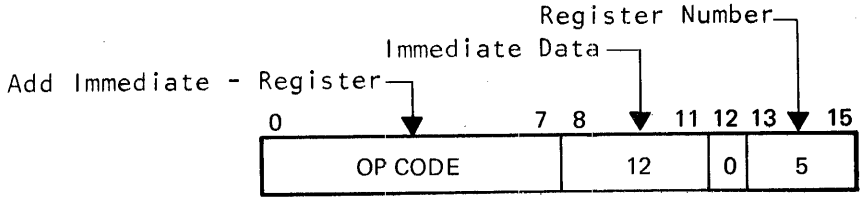
Minus: B, D, 1, 3, 5, or 9

Addressing

The data involved in an instruction can be in one of three places: in the instruction itself, in a general register, or in main storage. If the data is in the instruction, there really is no addressing; if it is in a general register, the programmer simply provides the register number. But if the data is in main storage, there are several techniques for defining the location of that data to the computer. The following material discusses the basic addressing techniques and how these techniques are combined. For ease of presentation, all numbers in the object formats in this section are decimal numbers.

IMMEDIATE DATA

Data held in the instruction is called immediate data, since it is immediately available to the computer; there is no addressing involved.



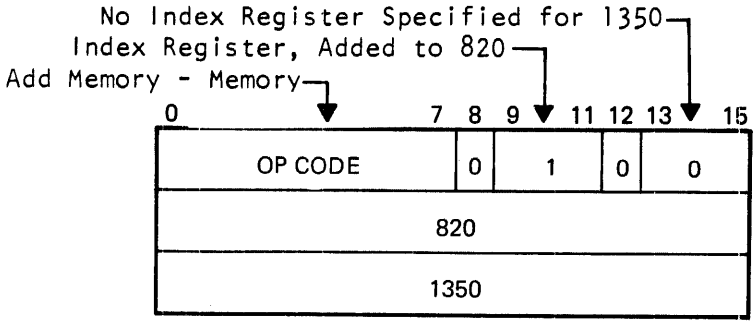
The immediate value 12 (decimal) will be added to the contents of general register 5. (Bit 12 is 0, indicating direct addressing of register 5.)

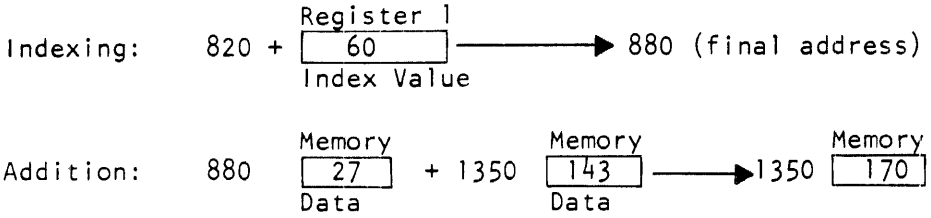
Note: Bits 8 and 12 have a special use for many instructions, this is described in the discussions of direct and indirect addressing in this section.

INDEXING

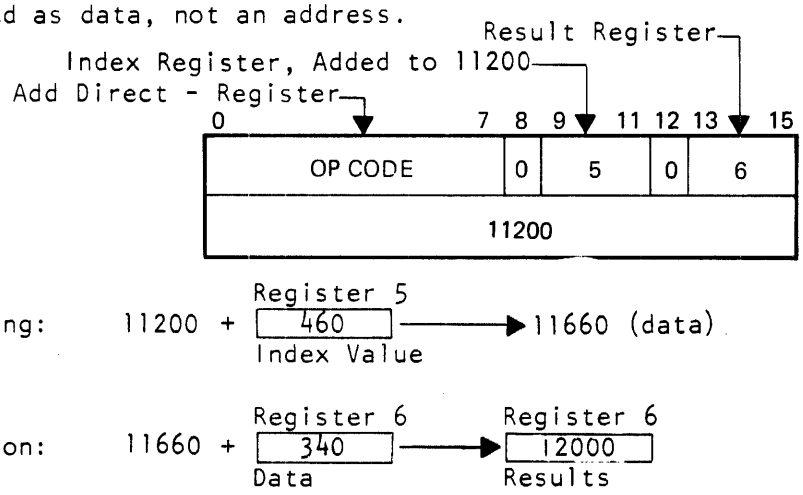
Indexing is an addressing technique in which a memory address is added to the contents of a general register (1-7) to form the address of data used in the instruction. A general register used for indexing is called an index register.

Indexing is optional for most of the 4-, 6-, and 8-byte instructions. For instructions that allow indexing the first register indicated (R_1) in an instruction is associated with the first memory address (M_1), and the second register (R_2) is associated with the second memory address (M_2). The following example, shows how indexing applies to a 6-byte instruction. Instruction contents in this and subsequent examples are expressed in decimal.





Several instructions use indexing to form a value used in the instruction rather than to form an address. These instructions add an immediate value to the contents of the index register; the result is treated as data, not an address.

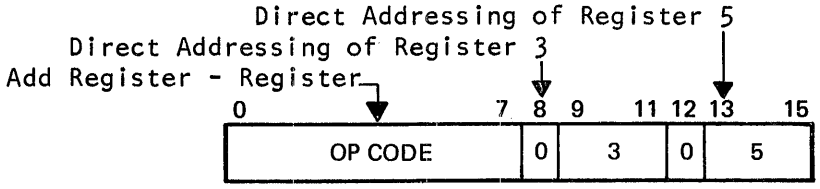


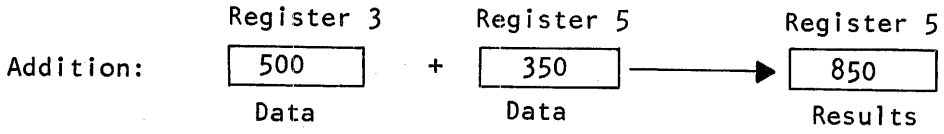
DIRECT ADDRESSING

A direct address is one (a memory address or a register number) that is either the final address of the data used in the instruction, or an address used for indexing. A direct address is not the address of a second address.

Direct addressing is indicated by bits 8 and 12 for instructions in which this technique applies. If the bit is 0 (off), direct addressing is indicated.

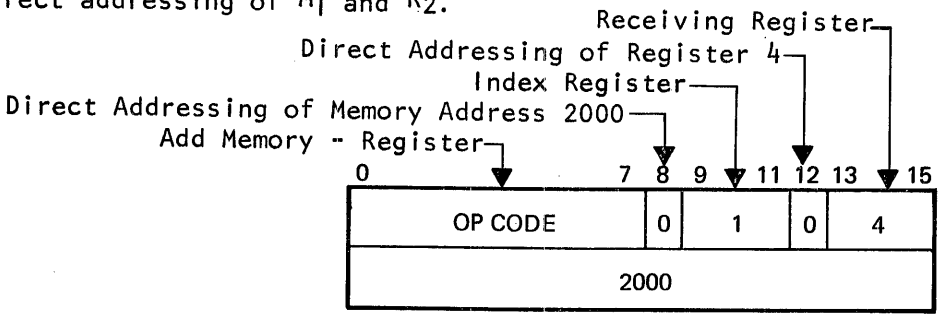
In this example, bits 8 and 12 are off. This indicates that the registers identified in the instruction contain the data used in the operation.

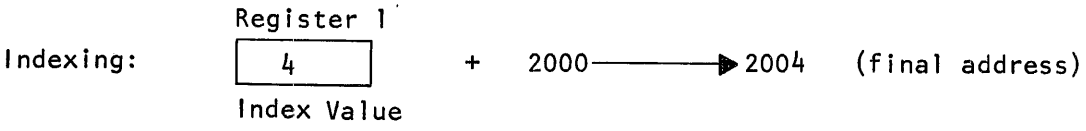


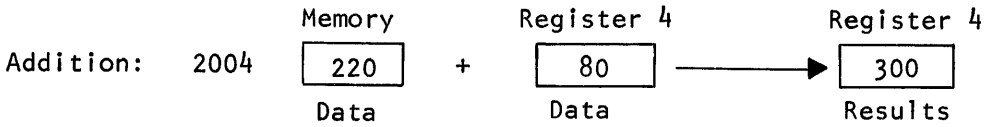


Direct addressing applies to register numbers and memory addresses. For 2-byte instructions, bit 8 is associated with R_1 (identified in bits 9-11) and bit 12 is associated with R_2 (identified in bits 13-15). For 4-byte instructions, bit 8 is associated with M_1 (identified in bits 16-31) and bit 12 is associated with R_2. R_1 in these instructions is used as an index register, and direct or indirect addressing does not apply to index registers. For 6-byte instructions, bit 8 is associated with M_1 and bit 12 is associated with M_2 (identified in bits 32-47). The 8-byte instructions all use direct addressing - it is a requirement.

The following 4-byte instruction shows an example of indexing and direct addressing of M_1 and R_2.



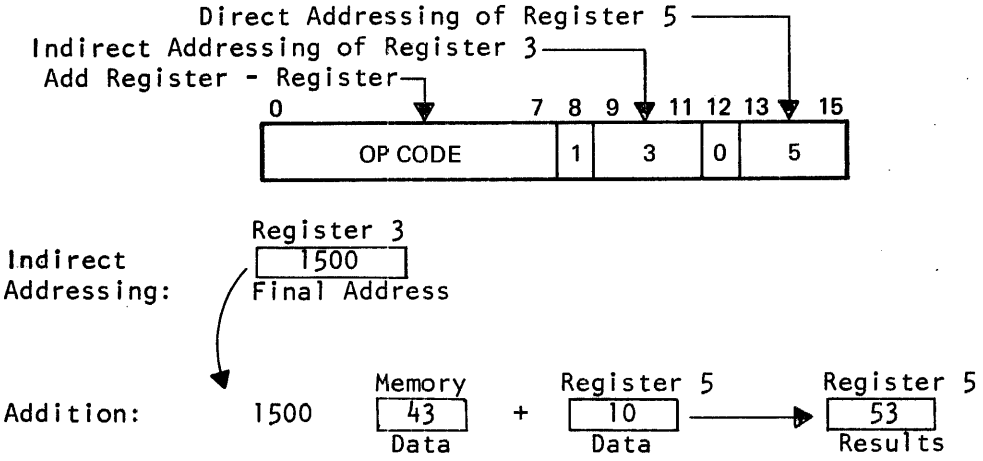




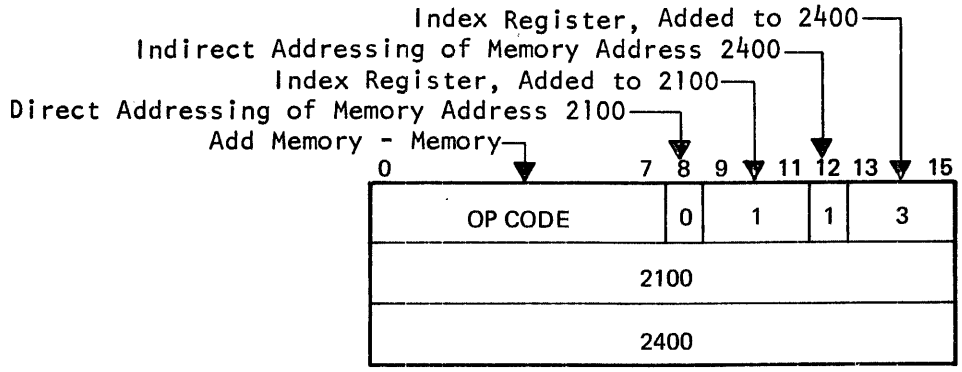
INDIRECT ADDRESSING

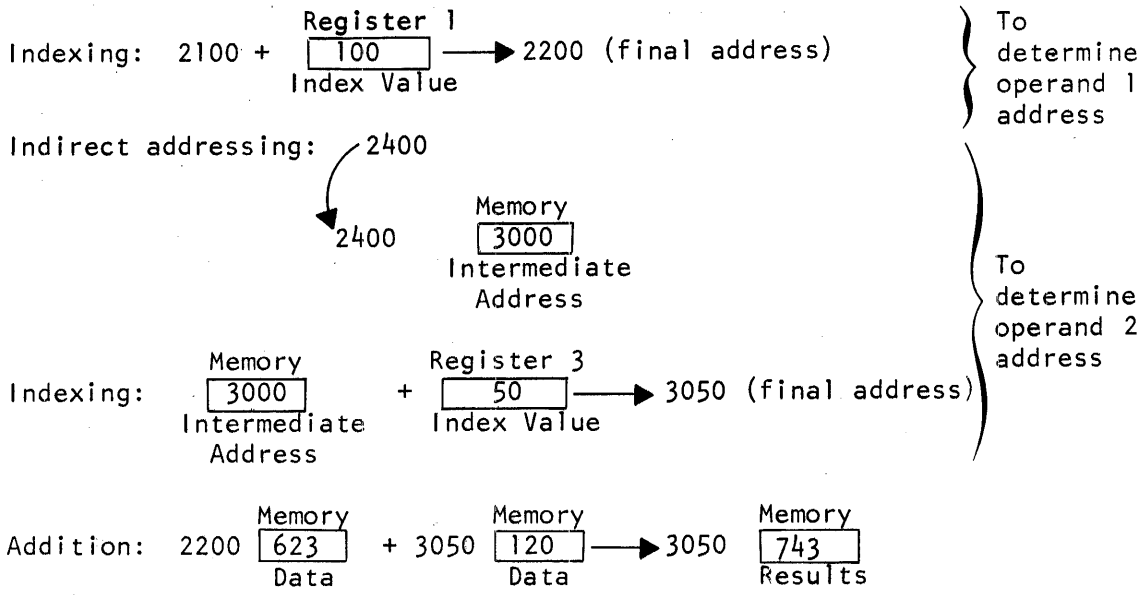
An indirect address (memory address or a register number) is the address of a second address. This second address may be the final address of the data used in the instruction, or it may be an address used for indexing. The second address must always be a memory address; it cannot be a register number. An indirect memory address must be an even-numbered address; the final address may be even or odd-numbered, depending on the instruction used.

Indirect addressing is indicated by bits 8 and 12 for instructions in which this technique applies. If the bit is 1 (on), indirect addressing is indicated. In the following example, bit 8 is on (1) indicating that the register identified in bits 9-11 contains a memory address, and at that memory address is the data used in the instruction. Bit 12 is off (0) indicating that the register identified in bits 13-15 contains data, not an address of data. The data flow in the example is memory to register.



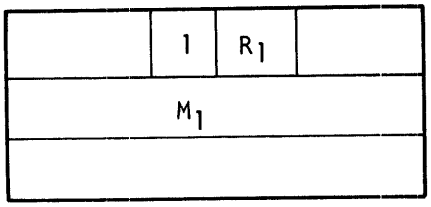
Indirect addressing applies to register numbers and memory addresses in the same way that direct addressing does. Bits 8 and 12 are associated with R_1 and R_2 for 2-byte instructions, with M_1 and R_2 for 4-byte instructions, and with M_1 and M_2 for 6-byte instructions. Indirect addressing is not allowed for 8-byte instructions. The following example of a 6-byte instruction shows how indirect addressing works in combination with indexing.





In the preceding example, to determine the operand 2 address, the indirect addressing was resolved, then the indexing was done. This technique is called post-indexing, because the indexing is done after the indirect addressing is resolved. Almost all instructions in which indirect addressing and indexing are allowed use the post-indexing technique. Very few instructions use a pre-indexing technique, in which the indexing is done before the indirect addressing is resolved. The following example shows how post-indexing and pre-indexing differ.

POST-INDEXING



The address of the operand is determined by these steps:

(1) Indirect addressing

Memory

M_1 → Intermediate Address

(2) Indexing

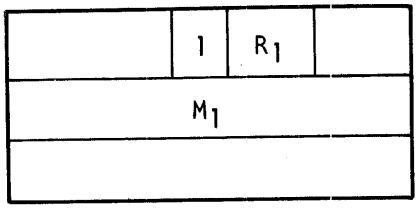
R_1

Intermediate Address = Index Value → Final Address

(3) Final address

Final address → Data

PRE-INDEXING



The address of the operand is determined by these steps:

(1) Indirect addressing

R_1

M_1 = Index Value → Intermediate Address

(2) Indexing

Memory

Intermediate Address → Final Address

(3) Final address

Final address → Data

Note: Bits 8 and 12 in the object instructions are used as indicator bits; they indicate something to the computer that the operation code alone cannot do. These bits are not always used to indicate direct or indirect addressing; sometimes they tell the computer more about the operands or the operation itself. When used in such a manner, these bits are called function indicators; they may be turned on by the assembler in some cases and by the programmer in other cases. Function bits are clearly defined for each instruction that uses them in the section Machine Instructions.

MACHINE INSTRUCTION FORMATS

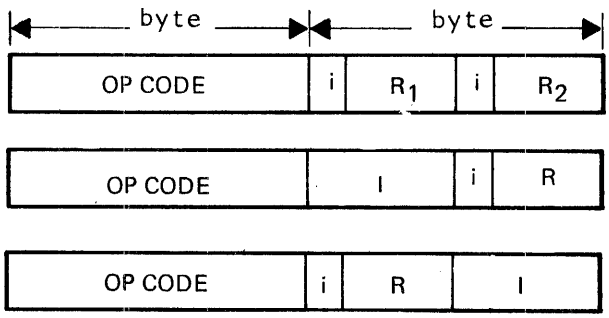
All programs used in the computer, whether system, utility, or application programs, have one thing in common: when they are executed they reside in main storage as a series of machine instructions. Machine instructions can be directly executed by the computer; they are the result of a translation of source instructions. The source instructions are in a format that can be easily interpreted by a programmer; the object (machine) instructions are in a format that is easily interpreted by the machine. The following material describes the machine instructions in their object format, as they would appear in main storage.

There are four basic machine instruction formats.

RR	Register-to-Register
RX	Register-to-Indexed Storage
XX	Indexed Storage-to-Indexed Storage
SS	Storage-to-Storage

The machine instructions are divided into these four categories on the basis of the instruction length (2, 4, 6, or 8 bytes of 8 bits each) and the type of operations they perform. The most common variations of each type of instruction are illustrated in the following diagrams.

RR Instructions (2-Byte)



R = general register

R_1 = general register

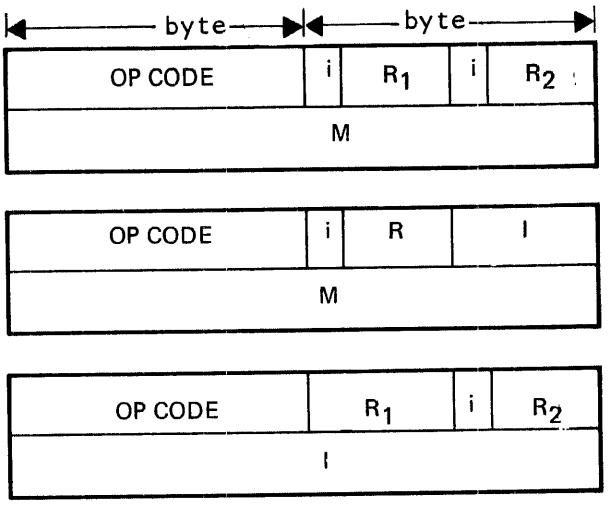
R_2 = general register

I = immediate data

i = indicator bit for direct or indirect addressing of R_1 and/or R_2

Data flow is 1 ⟶ 2

RX Instructions (4-Byte)



R_1 = general register for indexing of M or I

R_2 = general register

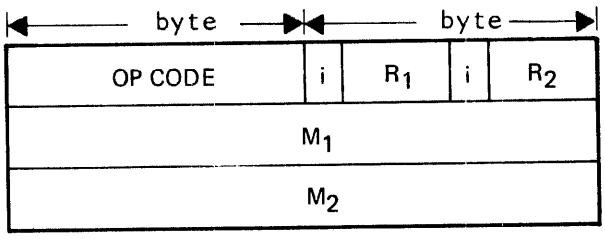
M = memory address

I = immediate data

i = indicator bit, for direct or indirect addressing of M or R_2

Data flow is usually 1 ⟶ 2

XX Instructions (6-Byte)



R_1 = general register for indexing of M_1

R_2 = general register for indexing of M_2

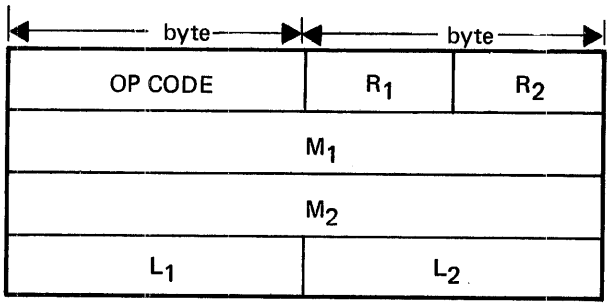
M_1 = memory address

M_2 = memory address

i = indicator bit, for direct or indirect addressing of M_1 or M_2

Data flow is 1 ⟶ 2

SS Instructions (8-Byte)



R_1 = general register for indexing of M_1

R_2 = general register for indexing of M_2

M_1 = memory address

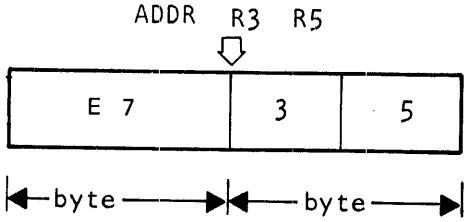
M_2 = memory address

L_1 = field length

L_2 = field length

Data flow is 1 ⟶ 2

An operation code in the first byte of each instruction tells the computer what type of operation to perform. This code is most easily represented as a two-digit hexadecimal number, each digit representing four bits. For instance, an *Add Register - Register* operation would have the following source and object representation. (The object format uses hexadecimal numbers in this example.)



The operation code tells the computer what to do with the data, while the rest of the instruction is concerned with identifying the location or address of the data to be used in the instruction.

SECTION III
MACHINE INSTRUCTIONS

TABLE OF CONTENTS

MACHINE INSTRUCTIONS

SECTION III
MACHINE INSTRUCTIONS

INTRODUCTION

This section describes the regular set of machine instructions. The restricted (Control and I/O) instructions are described in Section IV, Theory of Operations (not included in this version).

The instructions are grouped by function in the following order:

1. Arithmetic
2. Bit-Oriented
3. Boolean Logic
4. Branching
5. Compare
6. Data Conversion
7. Data Transfer
8. No Operation
9. Service Request
10. Shift

Within each group, the individual instructions are arranged alphabetically by name. Each instruction description includes the source format, the object format, a discussion of the operation involved, and an example. The following rules apply to the instruction descriptions.

- Most instructions must address even bytes in memory; the rest can address even or odd bytes. The instructions which can address even or odd bytes are identified by a bullet following the instruction name (such as, Compare Packed Decimal ●).

- The address of a memory field refers to the leftmost byte of that field.

- A word is defined as two bytes (8 bits in a byte); the bit positions in a word are numbered left to right (0-15).

- The operand fields of the instructions may be fixed or variable in length. Fixed-length operand fields may be one byte, one word (2 bytes), or two words (4 bytes) in length. Variable-length fields may range from 0-255 bytes.

- The operand fields for some instructions are contained in two registers or two words; however, only one register or word is addressed by the instruction. If the address of the operand is R, the operand may be at R and R+1 (plus one register) or R and R-1 (minus one register). For example: if R=5, the operand field may be in 5 and 6 or 4 and 5.

- If the address of the operand is M, the operand may be at M and M+1 (plus one word) or M and M-1 (minus one word). For example: if M=1324, the operand field is a four-byte field occupying locations 1324, 1325, 1326, and 1327 (for M and M+1) or 1322, 1323, 1324, and 1325 (for M and M-1).

- The Overflow bit (bit 0) in the Condition register is set if the results of a binary add or subtract exceed the limits of a signed one-word or two-word result field. Specifically, overflow is indicated if results > $2^{n-1}-1$ or results < -2^{n-1} (where n=16 or 32 bits).

- The Link bit (bit 3) in the Condition register is set if the results of a binary add or subtract exceed the limits of an unsigned one-word or two-word result field. Specifically, link is indicated if results > $2^{n}-1$ for an add (where n=16 or 32 bits), or if operand 1 $\geq$ operand 2 for a subtract.

The source and object formats of the operands are defined using the following symbols.

OP Code	The operation codes are presented in hexadecimal (00 through FF).
R	A general register number, 0-7. The register may be used as a sending or receiving field or as an index register.
M	A memory address, 0-65,535.
I	An immediate value; the value varies depending on the instruction. The value may represent an amount used in an arithmetic operation, a shift count, a skip count, or a bit number.
L	A field length, usually 0-255, but longer for some instructions. For certain instructions the length of an operand field may be defined in the instruction. The length specified in the instruction overrides any previous field length definition, but is only in effect for that instruction.

An R, M, I, or L in source operand 1 is identified as R_1, M_1, I_1, or L_1; in source operand 2 they are identified as R_2, M_2, I_2, or L_2. If there is only one operand, no number is used (R, M, I, or L). These symbols are numbered so that they can be referred to easily (distinguishing between R_1 and R_2 in the same instruction) and to make clear the location of these fields in the object format.

An at-sign (@) in a source operand indicates indirect addressing, an optional feature. For the instructions in which a register is a sending or receiving field, the at-sign indicates indirect addressing for R_1 or R_2. If a field in memory is the sending or receiving field, the at-sign indicates indirect addressing of M_1 or M_2.

Index registers and field lengths are optional; they are enclosed in parentheses in a source operand.

The two major operand fields are separated by a comma.

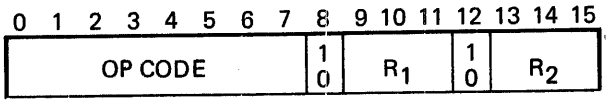
The following examples show how the source and object formats are illustrated. The at-sign and any designations in parentheses are almost always optional; if any of these designations are not optional, this fact will be noted. Data flow is usually operand 1 to operand 2, unless otherwise stated.

EXAMPLE 1

ADDR @R_1,@R_2

An optional at-sign which turns on bit 12

An optional at-sign which turns on bit 8.



Data flow is 1 to 2.

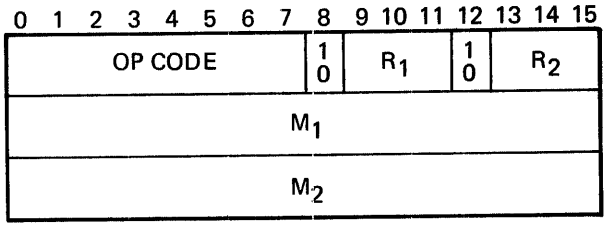
The variations of this instruction are as follows: data flow may be register to register, register to memory, memory to register, or memory to memory, depending on the use of indirect addressing.

ADDR { R_1 / @R_1 } { R_2 / @R_2 }

There are 2 x 2 = 4 possible variations of this instruction.

EXAMPLE 2

ADDM @M_1(R_1),@M_2(R_2)



An optional index register added to M_2

An optional at-sign which turns on bit 12, specifying indirect addressing of M_2

An optional index register added to M_1

An optional at-sign which turns on bit 8, specifying indirect addressing of M_1

The data flow of this instruction is always memory to memory. Any variation of operand 1 can be used with any variation of operand 2.

$$\text{ADDM} \begin{Bmatrix} M_1 \\ @M_1 \\ M_1(R_1) \\ @M_1(R_1) \end{Bmatrix} \quad \begin{Bmatrix} M_2 \\ @M_2 \\ M_2(R_2) \\ @M_2(R_2) \end{Bmatrix}$$

There are $4 \times 4 = 16$ possible variations of this instruction.

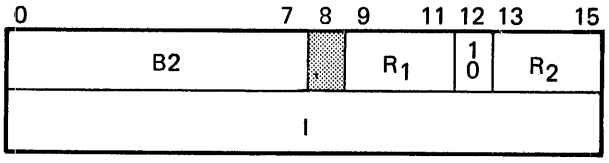
Timing formulas for all instructions are included as a part of Appendix C to this document.

ARITHMETIC INSTRUCTIONS

Mnemonic Code	Name
ADDD	Add Direct
ADDI	Add Immediate
ADDM	Add Memory - Memory
ADD	Add Memory - Register
ADDK	Add Packed Decimal ●
ADDR	Add Register - Register
ADDT	Add Two-Word
DIVD	Divide Direct
DIVI	Divide Immediate
DIVM	Divide Memory - Memory
DIV	Divide Memory - Register
DIVK	Divide Packed Decimal
DIVR	Divide Register - Register
MPYD	Multiply Direct
MPYI	Multiply Immediate
MPYM	Multiply Memory - Memory
MPY	Multiply Memory - Register
MPYK	Multiply Packed Decimal
MPYR	Multiply Register - Register
SUBD	Subtract Direct
SUBI	Subtract Immediate
SUBM	Subtract Memory - Memory
SUB	Subtract Memory - Register
SUBK	Subtract Packed Decimal ●
SUBR	Subtract Register - Register
SUBT	Subtract Two-Word
ZADK	Zero and Add

Add Direct

ADDD $I(R_1),@R_2$



FUNCTION: Performs a binary addition of a one-word immediate value* and a one-word field in a general register or in memory.

OPERAND 1: A 16-bit immediate signed value in bits 16-31 of the instruction; the value may range from -32,768 to +32,767.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting sum resides at the operand 2 location. The Condition register is affected as follows:

- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.

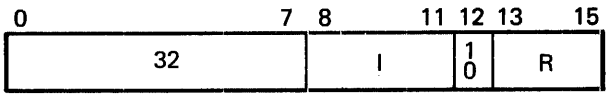
EXAMPLE: ADDD 15000(2),3

The immediate value of 15,000 is modified by the contents of register 2 and added to the value in register 3. The sum remains in register 3.

If @ (indirect addressing) is specified, register 3 contains a memory address.

Add Immediate

ADDI I,@R



FUNCTION: Performs a binary addition of a 4-bit immediate value and a one-word field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the value may range from 0-15. The I value is added to operand 2 in bit positions 12-15 with bits 0-11 zeros.

OPERAND 2: A one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting sum resides at the operand 2 location. The Condition register is affected as follows:

- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.

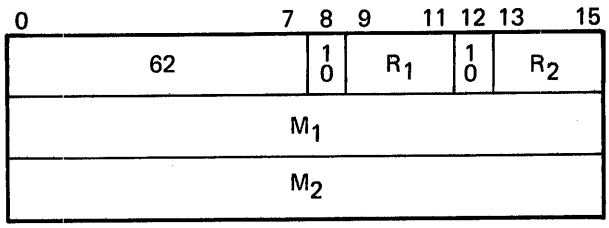
EXAMPLE: ADDI 10,@4

The immediate value of 10 is added to the operand at the memory address specified in register 4, at which address the sum is also placed.

If @ (indirect addressing) is not specified, the register contains an operand and subsequent sum, rather than an operand address.

Add Memory — Memory

ADDM @$M_1(R_1)$,@$M_2(R_2)$



FUNCTION: Performs a binary addition of two one-word fields in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

RESULTS: The resulting sum resides at the operand 2 location. The Condition register is affected as follows:

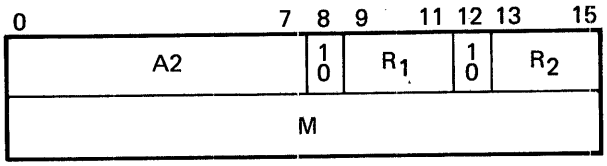
- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,335.

EXAMPLE: ADDM @HERE(3),@TAG(2)

@HERE(3) identifies a 16-bit operand which is added to another operand identified by @TAG(2). The subsequent sum is located at the address specified at @TAG(2).

Add Memory — Register

ADD @M(R_1),@R_2



FUNCTION: Performs a binary addition of a one-word field in memory and a one-word field in a general register or in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting sum resides at the operand 2 location. The Condition register is affected as follows:

- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.

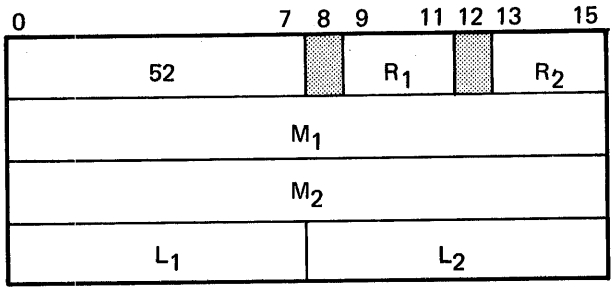
EXAMPLE: ADD TAG(5),7

A 16-bit operand identified by TAG(5) is added to the contents of register 7.

Although not specified in the example, @ (indirect addressing) can be applied to both operators.

Add Packed Decimal •

ADDK $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Performs a signed decimal addition, proceeding from right to left, of the two packed decimal fields in memory. The field lengths L_1 and L_2 may vary from 0-255 bytes.

OPERAND 1: A packed decimal field in memory. The field length, 0-255 bytes, is specified in the L_1 field of the instruction. The operand address indicated by M_1 may be indexed (R_1), but indirect addressing is not allowed. The effective address points to the most significant bytes of the decimal field.

OPERAND 2: A packed decimal field in memory. The field length, 0-255 bytes, is specified by the L_2 value in the instruction. The operand address indicated by M_2 may be indexed (R_2), but indirect addressing is not allowed. The effective operand address points to the most significant bytes of the decimal field.

RESULTS: The resulting decimal sum resides at the operand 2 location. The following conditions can occur, depending on the values of L_1 and L_2.

- If L_1 is greater than L_2 and the difference between L_1 and L_2 contains significant data, bit 0 of the Condition register is set.
- If $L_1 = 0$ and $L_2 = 0$, bit 3 of the Condition register is set.
- If $L_1 = 0$, an add of zero is assumed.
- If L_2 is greater than L_1, zeros are used to make up the difference in field lengths.

The Condition register is also affected as follows:

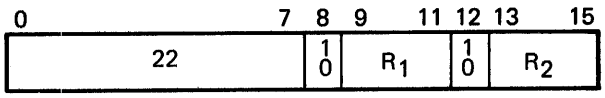
- Bit 0 is set if significant data is lost; bits 1-7 are cleared.
- Bits 1 and 5 are set if results are plus; bits 0, 2-4, and 6-7 are cleared.
- Bits 2 and 6 are set if results are minus; bits 0-1, 3-5, and 7 are cleared.
- Bits 3 and 7 are set if results are zero; bits 0-2, and 4-6 are cleared.

EXAMPLE: ADDK FIELD2(10,5),FIELD1(12,6)

A 10-byte packed field identified by FIELD2(10,5) is added to a 12-byte packed field identified by FIELD1(12,6). Had the field lengths been reversed (that is, trying to add the larger field to the smaller) and the overlap contains significant (non-zero) bytes, bit 0 of the Condition register is set to indicate lost data (an incorrect add).

Add Register — Register

ADDR @R_1,@R_2



FUNCTION: Performs a binary addition of two one-word fields; either field may be in a general register or in memory.

OPERAND 1: A one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting sum resides at the operand 2 location. The Condition register is affected as follows:

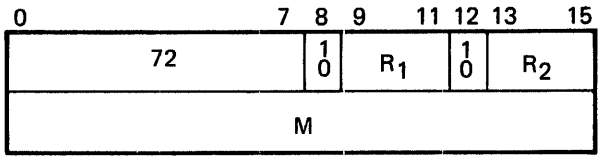
- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is more than 65,535.

EXAMPLE: ADDR @4,@1

In this example, both source operands specify indirect addressing; therefore, register 4 and register 1 contain memory addresses of operands. The operation adds the operand found through register 4 to the operand found through register 1 (sum remains at the latter location).

Add Two-Word

ADDT @M(R_1),@R_2



FUNCTION: Performs a binary addition of a two-word field in memory and a two-word field in two general registers or in memory.

OPERAND 1: A two-word field in memory at the effective address. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A two-word field located in two general registers (the most significant bits in the register specified by R_2 and the least significant bits in the register R_2+1) or in memory at the address if indirect addressing is used, bit 12=1. (Note: If register 7 is specified by R_2, the most significant bits are in register 7 and the least significant bits are in register 0.)

The effective address points to the most significant word of the operand.

RESULTS: The resulting sum resides at the operand 2 location. The Condition register is affected as follows:

- Bit 0 (overflow) is set if the result is greater than $+2^{31}-1$ or less than -2^{31}.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than $+2^{32}-1$.

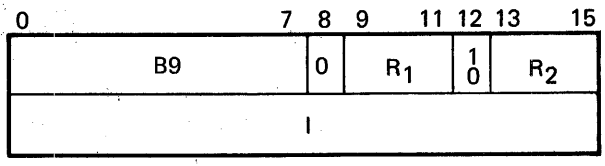
EXAMPLE: ADDT TAG(1),5

TAG(1) identifies a two-word field which is added to the contents of registers 5 and 6. The two word sum remains in registers 5 and 6.

Although not used in this example, @ can precede each operator to specify indirect addressing.

Divide Direct

DIVD I(R_1),@R_2



FUNCTION: Performs a binary division; the divisor is a one-word immediate value*, and the dividend is a two-word field in two general registers or in memory.

OPERAND 1: The divisor; a 16-bit immediate signed value in bits 16-31 of the instruction; the value may range from -32,768 to +32,767.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: The dividend; a 32-bit signed value contained in a two-word field.

If direct addressing is used (bit 12=0), the least significant bits are in the register specified by R_2 and the most significant bits are in the register R_2-1. (Note: If register 0 is specified by R_2, the least significant bits of the dividend are in register 0 and the most significant bits are in register 7.)

If indirect addressing is used (bit 12=1), the least significant bits are in memory at the effective address and the most significant bits at the effective address -2.

RESULTS: The quotient, a 16-bit signed value, resides at the R_2 register or the effective address of the operand 2 location. The remainder, a 16-bit signed value, resides at the R_2-1 register or effective address -2 of the operand 2 location; the remainder always has the same sign as the dividend.

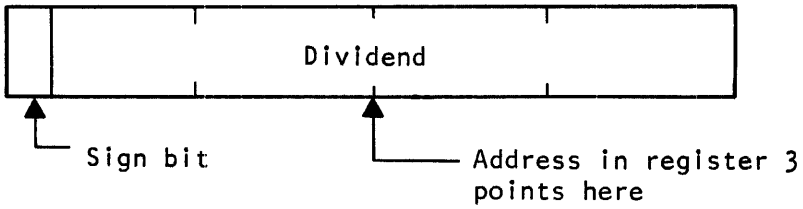
The Condition register is affected as follows:

- Bit 0 (overflow) is set if the resulting quotient is greater than +32,767 or less than -32,768.
- Bit 0 (overflow) is set if the divisor is 0; the operands are unchanged.

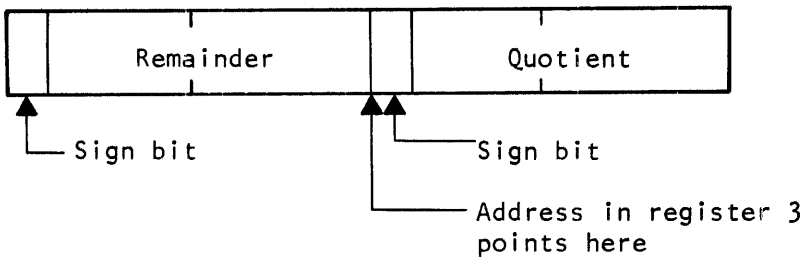
EXAMPLE: DIVD -32255(4),@3

The instruction specifies a binary divide in which the divisor is formed by adding -32,255 to the contents of register 4, and the dividend is found at the address specified in register 3.

As shown in the following illustration, the address specified in register 3 points to the third byte of the 32-bit dividend. This is the 4-byte field in memory that holds the dividend.

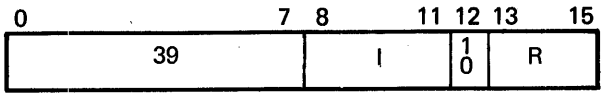


After the division, the quotient (including sign bit) is placed in the third and fourth bytes and the remainder (also with sign bit) is placed in bytes one and two. Thus, for subsequent operations, the quotient can be found at the address as specified in register 3. This is the same field after the divide operation.



Divide Immediate

DIVI I,@R



FUNCTION: Performs a binary divide; the divisor is a 4-bit immediate value, and the dividend is a two-word field in two general registers or in memory.

OPERAND 1: The divisor is a 4-bit unsigned value in bits 8-11 of the instruction; the value is always positive and may range from 0-15.

OPERAND 2: The dividend; a 32-bit signed value contained in a two-word field.

If direct addressing is used (bit 12=0), the least significant bits are in the register specified by R_1 and the most significant bits are in the register R. (Note: If register 0 is specified by R, the least significant bits of the dividend are in register 0 and the most significant bits are in register 7.)

If indirect addressing is used (bit 12=1), the least significant bits are in memory at the effective address and the most significant bits at the effective address -2.

RESULTS: The quotient, a 16-bit signed value, resides at the R register or the effective address of the operand 2 location. The remainder, a 16-bit signed value, resides at the R-1 register or effective address -2 of the operand 2 location; the remainder always has the same sign as the dividend.

The Condition register is affected as follows:

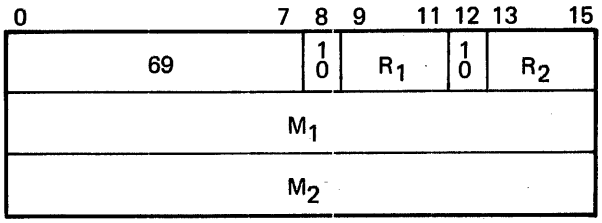
- Bit 0 (overflow) is set if the resulting quotient is greater than +32,767 or less than -32,768.
- Bit 0 (overflow) is set if the divisor is 0; the operands are unchanged.

EXAMPLE: DIVI 14,@1

Divides a 32-bit field (31 bits plus sign) by 14. The divisor is an absolute value in the instruction, and the dividend is located in memory according to the method described under DIVD. The quotient is stored at the address specified in register 1.

Divide Memory — Memory

DIVM @$M_1(R_1)$,@$M_2(R_2)$



FUNCTION: Performs a binary division; the divisor is a one-word field in memory, and the dividend is a two-word field in memory.

OPERAND 1: The divisor; a 16-bit signed value in a one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The dividend; a 32-bit signed value in memory at the effective address and the effective address -1. Addressing options to the base address M_2 include indexing (R_2), indirect addressing (bit 12=1), or a combination of both. The effective operand 2 address points to the least significant bits of the dividend.

RESULTS: The quotient, a 16-bit signed value, resides at the effective address of the operand 2 location. The remainder, 16-bit signed value, resides at the effective address -2 of the operand 2 location. The remainder always has the sign of the dividend.

The Condition register is affected as follows:

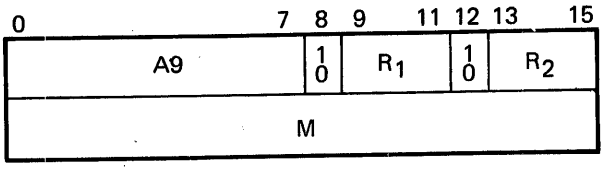
- Bit 0 (overflow) is set if the resulting quotient is greater than +32,767 or less than -32,768.
- Bit 0 (overflow) is set if the divisor is zero; the operands are unchanged.

EXAMPLE: DIVM @HERE(6),@TAG(4)

TAG(4) and HERE(6) are locations each containing an address of an operand. The operand identified by @TAG(4) is divided by the operand identified by @HERE(6). The dividend is a four-byte field; the address of the dividend, @TAG(4), points to the third byte of the four-byte field. After the division, this address points to the quotient; this address minus two bytes points to the remainder.

Divide Memory — Register

DIV @M(R_1),@R_2



FUNCTION: Performs a binary division; the divisor is a one-word field in memory, and the dividend is a two-word field in two general registers or in memory.

OPERAND 1: The divisor; a 16-bit signed value in a one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The dividend; a 32-bit signed value contained in a two-word field.

If direct addressing is used (bit 12=0), the least significant bits are in the register specified by R_2 and the most significant bits are in the register R_2-1. (Note: If register 0 is specified by R_2, the least significant bits of the dividend are in register 0 and the most significant bits are in register 7.)

If indirect addressing is used (bit 12=1), the least significant bits are in memory at the effective address and the most significant bits are at the effective address - 2.

RESULTS: The quotient, a 16-bit signed value, resides at the R_2 register or the effective address of the operand 2 location. The remainder, a 16-bit signed value, resides at the R_2-1 register or effective address - 2 of the operand 2 location; the remainder always has the same sign as the dividend.

The Condition register is affected as follows:

- Bit 0 (overflow) is set if the resulting quotient is greater than +32,767 or less than -32,768.
- Bit 0 (overflow) is set if the divisor is 0; the operands are unchanged.

E_4 = 0 if divisor = 0
E_4 = 1 if divisor ≠ 0

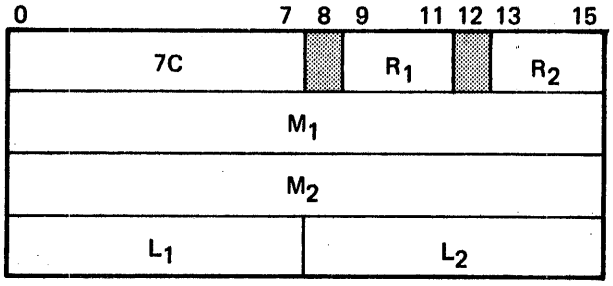
EXAMPLE: DIV TAG(4),@2

Register 2 contains the address (see DIVD) of the dividend (32-bit operand), and TAG(4) identifies the divisor (16-bit operand). The quotient is at the address in register 2, and the remainder is at that address minus two bytes.

Although not shown in the example, @ (indirect addressing) can be specified for the divisor also.

Divide Packed Decimal

DIVK $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Operand two, the dividend, is divided by operand one, the divisor, and replaced by the quotient and remainder.

OPERANDS: The quotient field is placed leftmost in the operand two, dividend field. The remainder field is placed rightmost in the same field and has a size equal to the length code, L_1, of the divisor. The quotient and remainder occupy the entire dividend field, thus the address of the dividend, operand two, is the address of the quotient. The size of the quotient field is L_2-L_1.

The length code, L_1, of the divisor must not exceed 127 (253 digits and sign) or be greater than or equal to the length code, L_2, of the dividend. Otherwise, overflow occurs and the operation is not performed. Also, a length code of zero for the divisor or zero divisor field contents causes overflow and operation suppression. The range of L_1 is 1-127, the range of L_2 is 2-255. The divisor, dividend, quotient and remainder are all signed packed decimal fields right-aligned. The sign of the quotient is determined by the rules of algebra. The sign of the remainder has the same value as the dividend sign. Sign validity checking is not performed. The following rules apply:

plus: hex 0,2,4,6,7,8,A,C,E,F

minus: hex 1,3,5,9,B,D

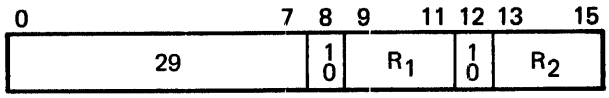
The preferred sign; plus, X'C', and minus, X'D', are generated. A minus zero quotient or remainder is forced to plus.

The operand fields remain unchanged if overflow occurs.

RESULTS: The operand fields may not overlap. Invalid digits cause undefined results. The following condition register settings can occur:

Bits 0-7	Condition
1000 0000	Overflow, $L_1 > 127$; $L_1 \geq L_2$, L_1=0; or; divisor field contents of zero
0100 0100	The quotient is greater than zero
0010 0010	The quotient is less than zero
0001 0001	The quotient is equal to zero

Divide Register — Register



DIVR $@R_1,@R_2$

FUNCTION: Performs a binary division; the divisor is a one-word field in a register or in memory, and the dividend is a two-word field in two adjacent registers or in memory.

OPERAND 1: The divisor; a 16-bit signed value in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: The dividend; a 32-bit signed value contained in a two-word field.

If direct addressing is used (bit 12=0), the least significant bits are in the register specified by R_2 and the most significant bits are in the register R_2-1. (Note: If register 0 is specified by R_2, the least significant bits of the dividend are in register 0 and the most significant bits are in register 7.)

If indirect addressing is used (bit 12=1), the least significant bits are in memory at the effective address and the most significant bits are in the effective address - 2.

RESULTS: The quotient, a 16-bit signed value, resides at the R_2 register or the effective address of the operand 1 location. The remainder, a 16-bit signed value, resides at the R_2-1 register or effective address - 1 of the operand 2 location; the remainder always has the same sign as the dividend.

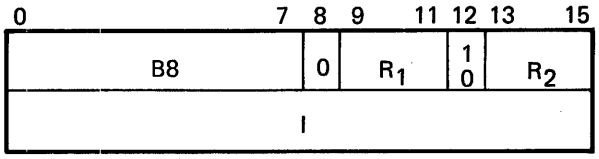
The Condition register is affected as follows:

- Bit 0 (overflow) is set if the resulting quotient is greater than +32,767 or less than -32,768.
- Bit 0 (overflow) is set if the divisor is 0; the operands are unchanged.

EXAMPLE: DIVR @1,@3

A binary divide is performed on a 32-bit (31 plus sign bit) dividend residing at the address specified in register 3. The 16-bit divisor resides at a location specified in register 1; the quotient is stored at the address in register 3; the remainder is stored at that address minus two bytes.

MPYD I(R_1),@R_2



FUNCTION: Performs a binary multiplication of a one-word immediate value* and a one-word field in a general register or in memory.

OPERAND 1: The multiplier; a 16-bit immediate signed value in bits 16-31 of the instruction; the value may range from -32,768 to +32,767.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: The multiplicand; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

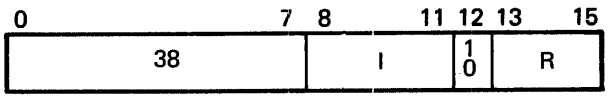
RESULTS: The product, a 32-bit signed value, resides at the operand 2 location. If operand 2 is in a register, the least significant bits of the product are in the register specified by R_2 and the most significant bits are in register R_2-1. (Note: If register 0 is specified by R_2, the least significant bits are in register 0 and the most significant bits are in register 7.) If operand 2 is in memory, the least significant bits of the product are at the effective address and the most significant bits are at the effective address - 2.

EXAMPLE: MPYD 16101(3),@1

A 16-bit (includes sign) multiplicand at the address in register 1 is multiplied by the sum of 16,101 and the contents of register 3. The 32-bit (includes sign bit) product is stored at the address in register 1 minus two bytes.

Multiply Immediate

MPYI I,@R



FUNCTION: Performs a binary multiplication of a 4-bit immediate value and a one-word field in a general register or in memory.

OPERAND 1: The multiplier; a 4-bit unsigned field located in bits 8-11 of the instruction; the value is always positive and may range from 0-15.

OPERAND 2: The multiplicand; a one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

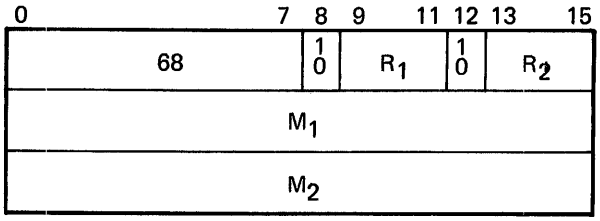
RESULTS: The product, a 32-bit signed value, resides at the operand 2 location. If operand 2 is in a register, the least significant bits of the product are in the register specified by R and the most significant bits are in register R-1. (Note: If register 0 is specified by R, the least significant bits are in register 0 and the most significant bits are in register 7.) If operand 2 is in memory, the least significant bits of the product are at the effective address and the most significant bits are at the effective address - 2.

EXAMPLE: MPYI 15,@6

The multiplicand located at the address specified in register 6 is multiplied by 15. The subsequent product is stored at the address in register 6 minus two bytes.

Multiply Memory — Memory

MPYM @M_1(R_1),@M_2(R_2)



FUNCTION: Performs a binary multiplication of two one-word fields in memory.

OPERAND 1: The multiplier; a one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The multiplicand; a one-word field in memory. Addressing options to the base address M_2 include indexing (R_2), indirect addressing (bit 12=1), or a combination of both.

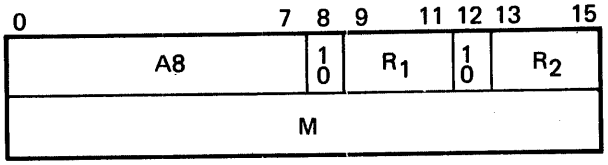
RESULTS: The product; a 32-bit signed value. The least significant bits reside at the operand 2 effective address, and the most significant bits reside at the operand 2 effective address - 2.

EXAMPLE: MPYM @FRRD(5),@TTM(2)

The location of the multiplicand is defined by @TTM(2), and the location of the multiplier by @FRRD(5). The 32-bit product is stored at the operand 2 address minus two bytes.

Multiply Memory — Register

MPY @M(R_1),@R_2



FUNCTION: Performs a binary multiplication of a one-word field in memory and a one-word field in a general register or in memory.

OPERAND 1: The multiplier; a one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The multiplicand; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

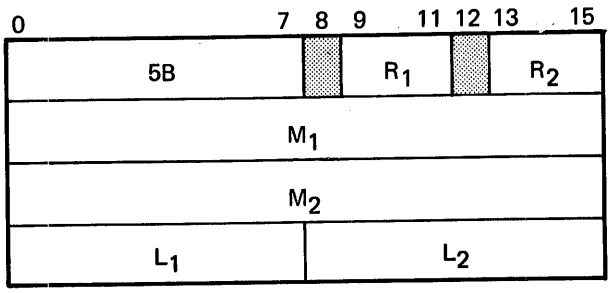
RESULTS: The product, a 32-bit signed value, resides at the operand 2 location. If operand 2 is in a register, the least significant bits of the product are in the register specified by R_2 and the most significant bits are in register R_2-1. (Note: If register 0 is specified by R_2, the least significant bits are in register 0 and the most significant bits are in register 7.) If operand 2 is in memory, the least significant bits of the product are at the effective address and the most significant bits are at the effective address - 2.

EXAMPLE: MPY @FRRD(2),@1

The multiplicand located at the address specified in register 1 is multiplied by a value at a location determined by @FRRD(2).

Multiply Packed Decimal

MPYK $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: The product of operand 1 (the multiplier) and operand 2 (the multiplicand) replaces operand 2. The multiplier size is limited to 253 digits and sign and also must be less than the multiplicand size. The length code, L_1, of the multiplier, if larger than 127, or greater than or equal to the length code, L_2, of the multiplicand is recognized as an overflow and no product is formed.

OPERANDS: Since the number of digits in the product is the sum of the digits in the operands, the multiplicand must have high-order zero bytes for at least a field size that equals the multiplier field size; otherwise, an overflow condition occurs. This definition of the multiplicand field insures that no product overflow can occur during the operation. At least one high-order digit of the product field is zero.

A field size specification of zero, length code L_1, for the multiplier causes an effective multiply by zero if the multiplicand length code, L_2, is non-zero. If either operand (or both) contains all zeros, the product field is set to zeros and a sign of plus (X'C') is forced. The range of L_1 is 0-127, the range of L_2 is 1-255.

The sign of the product is determined by the rules of algebra. Sign validity is not checked, therefore, the following interpretations are made:

plus hex 0,2,4,6,7,8,A,C,E,F

minus hex 1,3,5,9,B,D

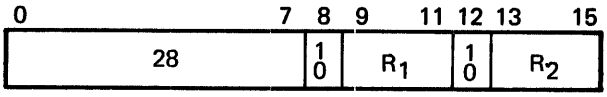
The fields may not overlap. Digit validity is not checked, undefined results will occur if non-decimal digits occur. If an overflow condition occurs, the operands remain unchanged. The preferred signs of X'C' for plus and X'D' for negative are generated for the product. A negative zero result is forced to plus.

RESULTS: The following condition register settings can occur:

Bits 0-7	Condition
1000 0000	Overflow: $L_1 > 127$; $L_1 \geq L_2$; or; less than L_1 bytes of high-order zeros in the multiplicand
0100 0100	Product is greater than zero.
0010 0010	Product is less than zero.
0001 0001	Product is equal to zero.

Multiply Register – Register

MPYR @R_1,@R_2



FUNCTION: Performs a binary multiplication of two one-word fields; either field may be in a register or in memory.

OPERAND 1: The multiplier; a one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: The multiplicand; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

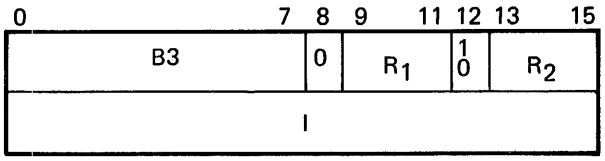
RESULTS: The product, a 32-bit signed value, resides at the operand 2 location. If operand 2 is in a register, the least significant bits of the product are in the register specified by R_2 and the most significant bits are in register R_2-1. (Note: If register 0 is specified by R_2, the least significant bits are in register 0 and the most significant bits are in register 7.) If operand 2 is in memory, the least significant bits of the product are at the effective address and the most significant bits are at the effective address -2.

EXAMPLE: MPYR @6,@7

The multiplicand at the memory location specified in register 7 is multiplied by the multiplier residing at the memory location specified in register 6. The multiplicand and multiplier are 16-bit operands while the subsequent product is 32 bits. The product is stored at the address in register 7 minus two bytes.

Subtract Direct

SUBD I(R_1),@R_2



FUNCTION: Performs a binary subtraction of a one-word immediate value* from a one-word field in a general register or in memory.

OPERAND 1: A 16-bit immediate signed value in bits 16-31 of the instruction; the value may range from -32,768 to +32,767. Operand 1 is subtracted from operand 2.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting difference resides at the operand 2 location. The Condition register is affected as follows:

- Bit 0 (overflow) is set if the result is less than -32,768 or grea than +32,767.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.
- Subtracting 8000_{16} from 0000_{16} results in 8000_{16} and bit 0 is set.

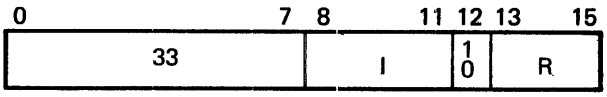
EXAMPLE: SUBD TAG(5),@0

A two's complement binary subtraction is performed. The subtrahend address is contained at the location specified by @TAG(5).

The minuend, from which the subtrahend is subtracted, resides at a memory location identified by the address specified in register 0. (Without indirect addressing, the minuend is contained in the named register.) The subsequent difference is stored at the location of the minuend.

Subtract Immediate

SUBI I,@R



FUNCTION: Performs a binary subtraction of a 4-bit field from a one-word field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the value may range from 0-15. The I value is subtracted from bits 12-15 of operand 2; bits 0-11 of operand 1 are zeroed out.

OPERAND 2: A one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting difference resides at the operand 2 location. The Condition register is affected as follows:

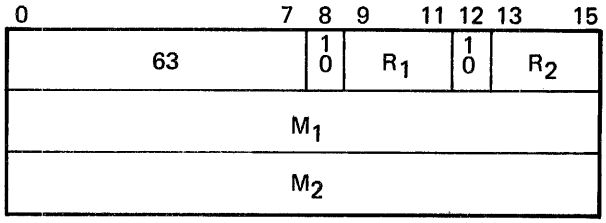
- Bit 0 (overflow) is set if the result is less than -32,768. or greater than +32,767.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.

EXAMPLE: SUBI 14,@4

The immediate value, 14, is subtracted from the 16-bit minuend residing at the memory location specified in register 4, which also is the location of the difference.

Subtract Memory — Memory

SUBM @$M_1(R_1)$,@$M_2(R_2)$



FUNCTION: Performs a binary subtraction of two one-word fields in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. This operand is subtracted from operand 2.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

RESULTS: The resulting difference resides at the operand 2 location. The Condition register is affected as follows:

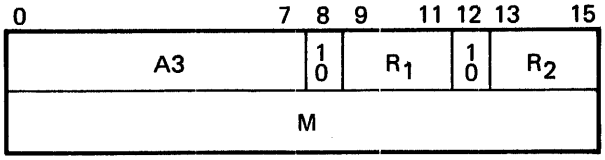
- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.
- Subtracting 8000_{16} from 0000_{16} results in 8000_{16} and bit 0 is set.

EXAMPLE: SUBM @BITL(6),@JLE(5)

@BITL(6) defines the memory location of the subtrahend which is subtracted from the minuend pointed to by @JLE(5). Both operands and the difference are 16-bit fields; the difference is at the location of the minuend.

Subtract Memory — Register

SUB @M(R_1),@R_2



FUNCTION: Performs a binary subtraction of a one-word field in memory and a one-word field in a general register or in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. This operand is subyracted from operand 2.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting difference resides at the operand 2 location. The Condition register is affected as follows:

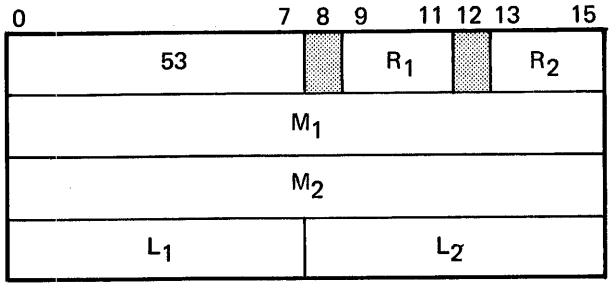
- Bit 0 (overflow) is set if the result is greater than +32,767 or less than -32,768.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.
- Subtracting 8000_{16} from 0000_{16} results in 8000_{16} and bit 0 is set.

EXAMPLE: SUB @TAG(5),@7

The location of the subtrahend is defined by @TAG(5), and that of the minuend by @7. Register 7 contains a minuend address which is also the address of the difference. @TAG(5) is the location in memory of the subtrahend.

Subtract Packed Decimal •

SUBK $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Performs a signed decimal subtraction of the two packed decimal fields in memory. The field lengths L_1 and L_2 may vary from 0-255 bytes.

OPERAND 1: A packed decimal field in memory which is subtracted from operand 2. The field length, 0-255 bytes, is specified by the L_1 value in the instruction. The operand address indicated by M_1 may be indexed (R_1), but indirect addressing is not allowed. The effective operand address points to the most significant bytes of the decimal field.

OPERAND 2: A packed decimal field in memory. The field length, 0-255 bytes, is specified by the L_2 value in the instruction. The operand address indicated by M_2 may be indexed (R_2), but indirect addressing is not allowed. The effective operand address points to the most significant bytes of the decimal field.

RESULTS: The resulting decimal difference resides at the operand 2 location. The following conditions can occur, depending on the values of L_1 and L_2.

- If L_1 is greater than L_2 and the difference between L_1 and L_2 contains significant data, bit 0 of the Condition register is set.
- If $L_1 = 0$ and $L_2 = 0$, bit 3 of the Condition register is set.
- If $L_1 = 0$, a subtract of zero is assumed.
- If L_2 is greater than L_1, zeros are used to make up the difference in field lengths.

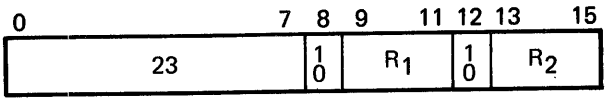
The Condition register is also affected as follows:

- Bit 0 is set if significant data is lost; bits 1-7 are cleared.
- Bits 1 and 5 are set if results are plus; bits 0, 2-4, and 6-7 are cleared.
- Bits 2 and 6 are set if results are minus; bits 0-1, 3-5, and 7 are cleared.
- Bits 3 and 7 are set if results are zero; bits 0-2, and 4-6 are cleared.

EXAMPLE: SUBK FIELD1(105,2),FIELD2(201,4)

In this example, a 105-byte field is subtracted from a 201-byte field. The location of the subtrahend is FIELD1(105,2). The location of the minuend and subsequent difference is identified by FIELD2(201,4).

Subtract Register — Register



SUBR $@R_1,@R_2$

FUNCTION: Performs a binary subtraction of two one-word fields; either field may be in a general register or in memory.

OPERAND 1: A one-word field located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. This operand is subtracted from operand 2.

OPERAND 2: A one-word field located in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting difference resides at the operand 2 location. The Condition register is affected as follows:

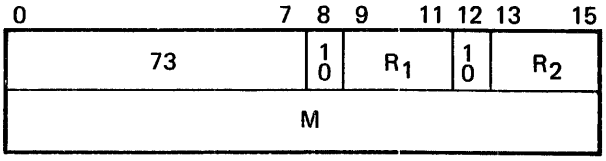
- Bit 0 (overflow) is set if the result is less than -32,768 or greater than +32,767.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than 65,535.
- Subtracting 8000_{16} from 0000_{16} results in 8000_{16} and bit 0 is set.

EXAMPLE: SUBR @5,@0

The value at the location specified in register 5 is subtracted from the value at the location specified in register 0. The difference replaces the minuend.

Subtract Two-Word

SUBT @M(R_1),@R_2



FUNCTION: Performs a binary subtraction of a two-word field in memory from a two-word field in two general registers or in memory.

OPERAND 1: A two-word field in memory at the effective address. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. This operand is subtracted from operand 2. The effective address points to the most significant bits of the two-word field.

OPERAND 2: A two-word field located in two general registers (the most significant bits in the register specified by R_2 and the least significant bits in the register R_2+1) or in memory at the address if indirect addressing is used, bit 12=1. (Note: If register 7 is specified by R_2, the most significant bits are in register 7 and the least significant bits are in register 0.)

RESULTS: The resulting difference resides at the operand 2 location. The Condition register is affected as follows:

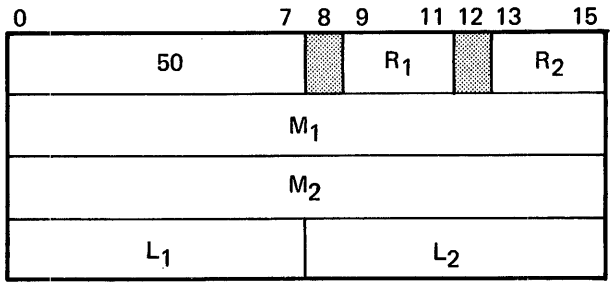
- Bit 0 (overflow) is set if the result is greater than $+2^{31}-1$ or less than -2^{31}.
- Bits 1-2 and 4-7 are cleared.
- Bit 3 (link) is set if the result is greater than $+2^{31}-1$.
- Subtracting 80000000_{16} from 00000000_{16} results in 80000000_{16} and bit 0 is set.

EXAMPLE: SUBT TAG(5),2

A two-word field identified by TAG(5) is subtracted from the contents of registers 2 and 3. The difference is held in the registers.

Zero and Add •

ZADK $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Zeros out a field in memory, then performs an addition of a packed decimal field in memory and the zero field. The field lengths L_1 and L_2 may vary from 0-255 bytes.

OPERAND 1: A packed decimal field in memory; the field length, 0-255 bytes, is specified by the value in the instruction. The operand address indicated by M_1 may be indexed (R_1), but indirect addressing is not allowed. The effective operand address points to the most significant bytes of the decimal field.

OPERAND 2: A field in memory that is zeroed out before the addition. The field length, 0-255 bytes, is specified by the L_2 value of the instruction. The field address indicated by M_2 may be indexed by (R_2), but no indirect addressing is allowed. The effective field address points to the most significant address of the field.

RESULTS: The resulting field resides at the operand 2 location. The following conditions can occur, depending on the values of L_2 and L_1.

- If L_1 is greater than L_2 and the difference between L_1 and L_2 contains significant data, bit 0 of the Condition register is set.
- If L_1 = 0 and L_2 = 0, bit 3 of the Condition register is set.
- If L_1 = 0, an add of zero is assumed.
- If L_2 is greater than L_1, zeros are used to make up the difference in field lengths.

The Condition register is also affected as follows:

- Bit 0 is set if significant data is lost; bits 1-7 are cleared.
- Bits 1 and 5 are set if results are plus; bits 0, 2-4, and 6-7 are cleared.
- Bits 2 and 6 are set if results are minus; bits 0-1, 3-5, and 7 are cleared.
- Bits 3 and 7 are set if results are zero; bits 0-2, and 4-6 are cleared.

EXAMPLE: ZADK TAG2(50,4),TAG1(55,0)

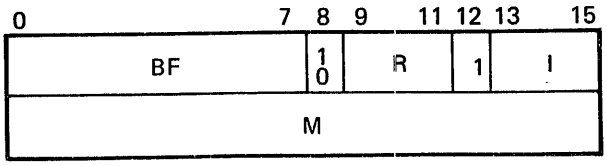
Initially, a 55-byte field identified by TAG1(55,0) is cleared to zero; then a 50-byte field identified by TAG2(50,4) is added to it. (See Results description for Condition register status.)

BIT-ORIENTED INSTRUCTIONS

Mnemonic Code	Name
IBIT	Invert Bit •
RBIT	Reset Bit •
ROFR	Reverse Off-Bit
RONR	Reverse On-Bit
SBIT	Set Bit •
TBIT	Test Bit •
TOFR	Test for Off-Bit
TONR	Test for On-Bit

Invert Bit •

IBIT @M(R),I



FUNCTION: Invert (toggle) a bit in a one-byte field in memory.

OPERAND 1: A one-byte field in memory. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A 3-bit value in bits 13-15 of the instruction. This value specifies the position of the bit to be toggled and may range from 0-7; 0 specifies the leftmost position and 7 the rightmost position.

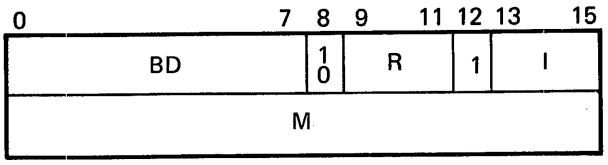
RESULTS: The resultant field resides at the operand 1 location.

EXAMPLE: IBIT @TAG(4),3

This instruction reverses the existing binary state of a specific bit in memory. @TAG(4) points to an 8-bit byte and 3 is the number of the bit (bits are numbered left to right, 0-7).

Reset Bit •

RBIT @M(R),I



FUNCTION: Resets a bit (to 0) in a one-byte field in memory.

OPERAND 1: The one-byte field in memory. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

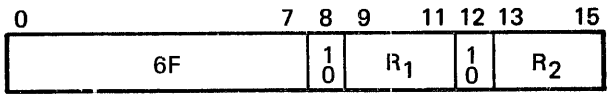
OPERAND 2: A 3-bit value in bits 13-15 of the instruction. This value specifies the position of the bit to be reset and may range from 0-7; 0 specifies the leftmost position and 7 specifies the rightmost position.

RESULTS: The resultant field resides at the operand 1 location.

EXAMPLE: IBIT @TAG(2),4

Bit number 4 of an 8-bit byte located at @TAG(2) in memory is given the binary state of 0.

Reverse Off-Bit



ROFR @R_1,@R_2

FUNCTION: Scans a one-word field, left to right, for the first off-bit, and increases another one-word field by an amount equal to the position of the first off-bit (0-15). The first off-bit is then turned on. If no off-bit is found, the field is increased by a value of 16. Either field may be in a general register or in memory.

OPERAND 1: A one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. This field is scanned for the first off-bit.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1. The field is increased by a value equal to the position of the first off-bit in operand 1.

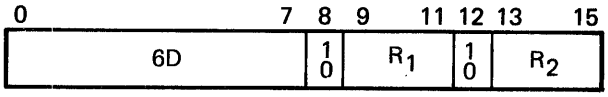
RESULTS: The resultant operands reside at their original locations.

EXAMPLE: ROFR @7,@2

Register 7 contains an address of a 16-bit field which is scanned from left to right for a 0-bit. If a 0-bit is found, the bit position (0-15) determines the value to add to a one-word field located at an address specified in register 2. After this value is increased, the 0-bit is set to 1.

Reverse On-Bit

RONR @R_1,@R_2



FUNCTION: Scans a one-word field, left to right, for the first on-bit, and increases another one-word field by an amount equal to the position of that first on-bit (0-15). The first on-bit is then turned off. If no on-bits are found, the field is increased by a value of 16. Either field may be in a general register or in memory.

OPERAND 1: A one-word field located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. This field is scanned for the first on-bit.

OPERAND 2: A one-word field located in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1. This field is increased by a value equal to the position of the first on-bit in operand 2.

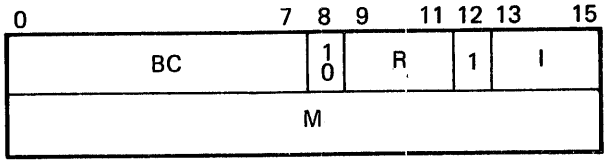
RESULTS: The resultant operands reside at their original locations.

EXAMPLE: RONR @4,@6

Register 4 contains the address of a 16-bit field which is scanned for an on-bit. If a 1-bit is found, the binary position (0-15) of that bit determines the value (0-15) added to another one-word field at the location specified in register 6. After this field is increased in value, the state of the original bit is changed to a 0-bit. If no 1-bit is found during the scan, the other field is increased by a value of 16.

Set Bit ●

SBIT @M(R),I



FUNCTION: Sets a bit (to 1) in a one-byte field in memory.

OPERAND 1: A one-byte field in memory. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A 3-bit value in bits 13-15 of the instruction. This value specifies the position of the bit to be set and may range from 0-7; 0 specifies the leftmost position and 7 the rightmost position.

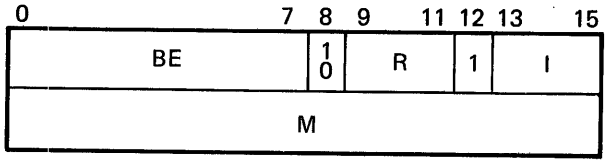
RESULTS: The resultant field resides at the operand 1 location.

EXAMPLE: SBIT @TAG(2),7

Turns on bit number 7 of an 8-bit byte at the location specified by @TAG(2).

Test Bit •

TBIT @M(R),I



FUNCTION: Tests a bit in a one-byte field in memory, and transfers the bit state (1 or 0) to bit 0 of the Condition register.

OPERAND 1: The one-byte field in memory. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A 3-bit value in bits 13-15 of the instruction; this value specifies the position of the bit in operand 2 and may range from 0-7; 0 specifies the leftmost position and 7 the rightmost position.

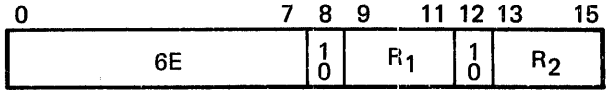
RESULTS: The result is reflected in the bit state of bit 0 in the Condition register.

EXAMPLE: TBIT @JTE(5),6

The binary state of bit 6 of an 8-bit byte at a location specified by @JTE(5) is reproduced in bit 0 of the Condition register.

Test for Off-Bit

TOFR @R_1,@R_2



FUNCTION: Scans a one-word field, left to right, for the first off-bit, and increases another one-word field by an amount equal to the position of that first off-bit (0-15). The first off-bit is not changed. If no off-bits are found, the field is increased by a value of 16. Either field may be in a general register or in memory.

OPERAND 1: A one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. This field is scanned for the first off-bit.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1. This field is increased by an amount equal to the position of the first off-bit in operand 1.

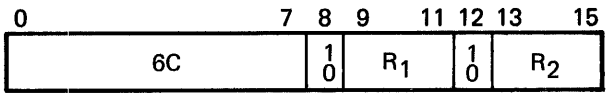
RESULTS: The resultant operands reside at their original locations.

EXAMPLE: TOFR @3,@1

A 16-bit field at a location specified in register 3 is scanned left to right (0-15) for a 0-bit. If none is found, the value at the location specified in register 1 is increased by 16. However, if a 0-bit is found, the bit position (0-15) of that bit specifies the value added to the other field.

Test for On-Bit

TONR @R_1,@R_2



FUNCTION: Scans a one-word field, left to right, for the first on-bit, and increases another one-word field by an amount equal to the bit position of the first on-bit (0-15). The first on-bit is not changed. If no on-bits are found, the field is increased by a value of 16. Either field may be in a general register or in memory.

OPERAND 1: A one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. This field is scanned for the first on-bit.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1. This field is increased by an amount equal to the position of the first on-bit in operand 1.

RESULTS: The resultant operands reside at their original locations.

EXAMPLE: TONR @2,@3

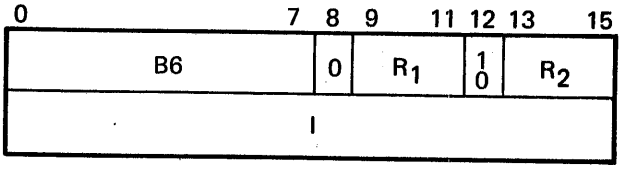
A 16-bit field at a location specified in register 2 is scanned left to right (0-15) for a 1-bit. If none is found, the value at the location specified in register 3 is increased by 16. If a 1-bit is found, the binary position (0-15) of that bit corresponds to the value added to the other field.

BOOLEAN LOGIC INSTRUCTIONS

Mnemonic Code	Name
EORD	Exclusive OR Direct
EORI	Exclusive OR Immediate
EORM	Exclusive OR Memory – Memory
EOR	Exclusive OR Memory – Register
EORR	Exclusive OR Register – Register
IORD	Inclusive OR Direct
IORI	Inclusive OR Immediate
IORM	Inclusive OR Memory – Memory
IOR	Inclusive OR Memory – Register
IORR	Inclusive OR Register – Register
ANDD	Logical Product Direct
ANDI	Logical Product Immediate
ANDM	Logical Product Memory – Memory
AND	Logical Product Memory – Register
ANDR	Logical Product Register – Register

Exclusive OR Direct

EORD I(R_1),@R_2



FUNCTION: Performs an exclusive OR of a one-word immediate value* and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A 16-bit immediate value in bits 16-31 of the instruction; the value may range from 0 to 65,535.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

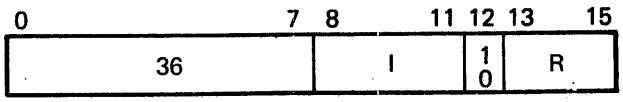
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: EORD 65501(2),@5

An exclusive OR is performed between the value of 65,501(2) and the 16-bit field at a location specified by the address in register 5; this address is also the address of the result.

Exclusive OR Immediate

EORI I,@R



FUNCTION: Performs an exclusive OR between a 4-bit immediate value held in the instruction and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the value may range from 0-15. The I value is ORed to operand 2 in bit positions 12-15 with bits 0-11 zeros.

OPERAND 2: A one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

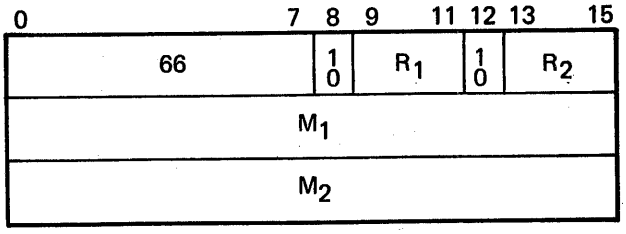
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: EORI 13,@1

An exclusive OR is performed on the immediate value of 13 and the 16-bit field at a location specified by the address in register 1; this address is also the address of the result.

Exclusive OR Memory — Memory

EORM @$M_1(R_1)$,@$M_2(R_2)$



FUNCTION: Performs an exclusive OR of two one-word fields in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

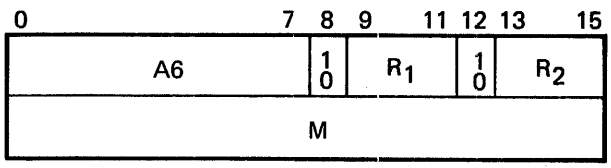
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: EORM @TAG(2),@DTN(4)

An exclusive OR is performed between a 16-bit field at the address identified by @TAG(2) and the 16-bit field at the address identified by @DTN(4). The result is stored at the address specified by @DTN(4).

Exclusive OR Memory — Register

EOR @M(R_1),@R_2



FUNCTION: Performs an exclusive OR of a one-word field in memory and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

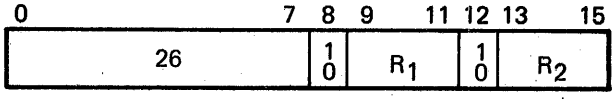
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: EOR @TAG(3),@5

An exclusive OR is performed between a 16-bit field at the address identified by @TAG(3) and the 16-bit field at the address identified in register 5. The address specified in register 5 is also that of the result.

Exclusive OR Register — Register

EORR $@R_1,@R_2$



FUNCTION: Performs an exclusive OR of two one-word fields; either field may be in a register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: Same as operand 1 except use R_2 and bit 12=1.

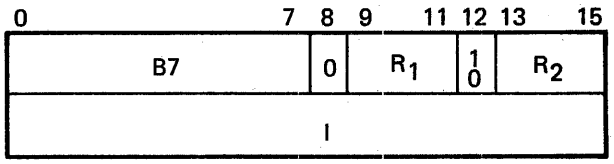
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: EORR @4,@7

An exclusive OR is performed between two 16-bit fields, one of which is at the address specified in register 4 and the other at the address specified in register 7. The result is placed at the address specified in register 7.

Inclusive OR Direct

IORD I(R_1),@R_2



FUNCTION: Performs an inclusive OR of a one-word immediate value* and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or if both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

OPERAND 1: A 16-bit immediate value in bits 16-31 of the instruction; the value may range from 0-65,535.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

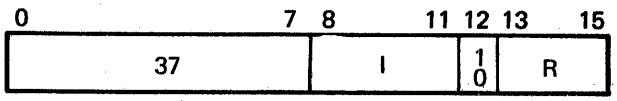
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: IORD 64201(3),@5

An inclusive OR is performed on the immediate value 64,201, as modified by the contents of register 3, and the 16-bit field at the address specified in register 5. This address is also the address of the result.

Inclusive OR Immediate

IORI I,@R



FUNCTION: Performs an inclusive OR between a 4-bit immediate value held in the instruction and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or both bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the value may range from 0-15. The I value is ORed to operand 2 in bit positions 12-15 with bits 0-11 zeros.

OPERAND 2: A one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

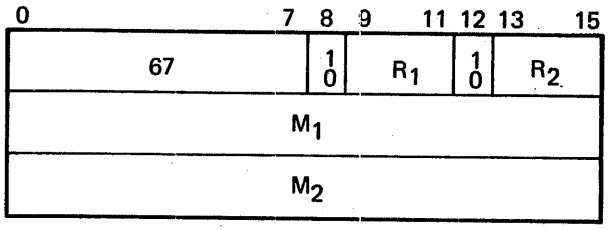
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: IORI 10,@3

An inclusive OR is performed on the immediate value 10 and the 16-bit field at the address in register 3. The result is stored at the address in register 3.

Inclusive OR Memory — Memory

IORM @M_1(R_1),@M_2(R_2)



FUNCTION: Performs an inclusive OR of two one-word fields in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

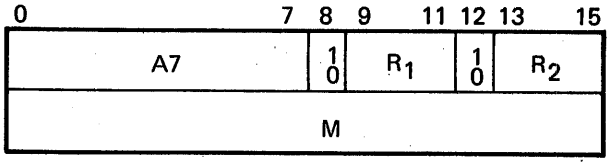
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: IORM @HERE(2),@TAG(1)

Performs an inclusive OR between two 16-bit fields. @TAG(1) represents the address of the results.

Inclusive OR Memory — Register

IOR @M(R_1),@R_2



FUNCTION: Performs an inclusive OR of a one-word field in memory and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or if both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

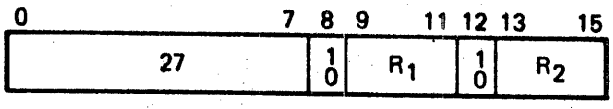
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: IOR @TAG(3),@7

An inclusive OR is performed on two 16-bit fields, one at the address identified by @TAG(3) and the other at the address specified in register 7. The address in register 7 is the address of the result.

Inclusive OR Register -- Register

IORR @R_1,@R_2



FUNCTION: Performs an inclusive OR on two one-word fields; either field may be in a register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or if both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

OPERAND 1: A one-word field located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: Same as operand 1 except use R_2 and bit 12=1.

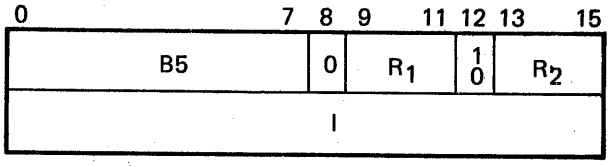
RESULTS: The resulting OR operand resides at the operand 2 location.

EXAMPLE: IORR @5,@6

An inclusive OR is performed between the two 16-bit fields at the addresses specified in registers 5 and 6. Results are placed at the address in register 6.

Logical Product Direct

ANDD I(R_1),@R_2



FUNCTION: Performs a logical product of a one-word immediate value* and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases, the resultant bit is 0.

OPERAND 1: A 16-bit immediate value in bits 16-31 of the instruction; the value may range from 0-65,535.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

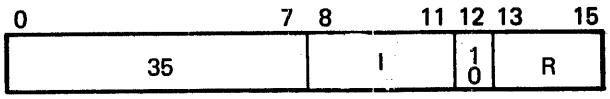
RESULTS: The resulting field resides at the operand 2 location.

EXAMPLE: ANDD 40000(3),@1

A logical product operation is performed between 40,000, as modified by the contents of register 3, and the 16-bit field at the address in register 1. This address is also the address of the results.

Logical Product Immediate

ANDI I,@R



FUNCTION: Performs a logical product of a 4-bit immediate value and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction; the value may range from 0-15. The I value is compared against operand 2 in bit positions 12-15 and bits 0-11 are zeros.

OPERAND 2: A one-word field located in the general register specified by R or in memory if indirect addressing is specified, bit 12=1.

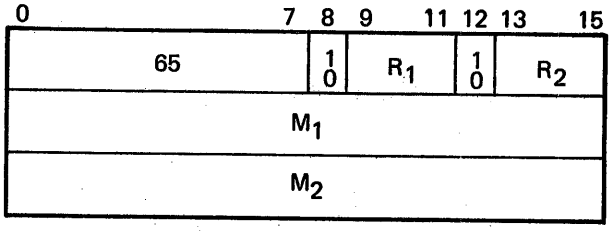
RESULTS: The resulting field resides at the operand 2 location.

EXAMPLE: ANDI 14,@2

A logical product is performed on the immediate value 14 and the 16-bit field at the address specified in register 2. This address is also the address of the result.

Logical Product Memory -- Memory

ANDM @$M_1(R_1)$,@$M_2(R_2)$



FUNCTION: Performs a logical product of two one-word fields in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

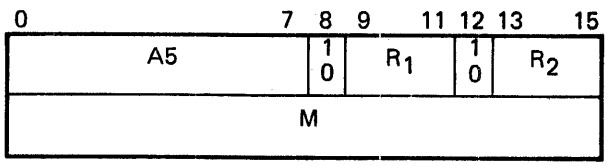
RESULTS: The resulting field resides at the operand 2 location.

EXAMPLE: ANDM @HOLD(7),@SPIKE(2)

A logical product is performed between a 16-bit field at the address identified by @HOLD(7) and the 16-bit field at the address specified by @SPIKE(2). The result is stored at the @SPIKE(2) address.

Logical Product Memory — Register

AND @M(R_1),@R_2



FUNCTION: Performs a logical product of a one-word field in memory and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

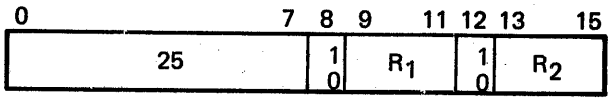
RESULTS: The resulting field resides at the operand 2 location.

EXAMPLE: AND @TAG(3),@5

Performs a logical product between two 16-bit fields. The results are stored at the address specified in register 5.

Logical Product Register – Register

ANDR @R_1,@R_2



FUNCTION: Performs a logical product of two one-word fields; either field may be in a register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

OPERAND 1: A one-word field located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: A one-word field located in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: The resulting field resides at the operand 2 location.

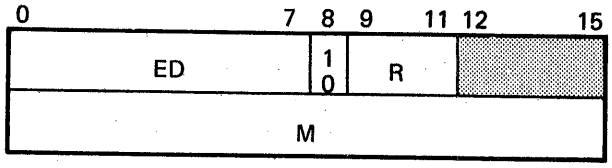
EXAMPLE: ANDR @7,@4

A logical product is performed between the 16-bit field at the address specified in register 7 and the 16-bit field at the address specified in register 4. The address in register 4 is the address of the results.

BRANCHING INSTRUCTIONS

Mnemonic Code	Name
B	Branch
BA1	Branch Add One
BA2	Branch Add Two
BSR	Branch and Save Return
BOF	Branch if Bit Off
BON	Branch if Bit On
BRN	Branch if Register Not Zero
BRZ	Branch if Register Zero
BCF	Branch on Condition Register False
BCT	Branch on Condition Register True
BS1	Branch Subtract One
BS2	Branch Subtract Two
BR	Branch to Address in Register
BCH	Branch Unconditional
SRMF	Skip if Register Minus - Forward
SRMB	Skip if Register Minus - Backward
SRNF	Skip if Register Not Zero - Forward
SRNB	Skip if Register Not Zero - Backward
SRPF	Skip if Register Plus - Forward
SRPB	Skip if Register Plus - Backward
SRZF	Skip if Register Zero - Forward
SRZB	Skip if Register Zero - Backward
SCFF	Skip on Condition Register False - Forward
SCFB	Skip on Condition Register False - Backward
SCTF	Skip on Condition Register True - Forward
SCTB	Skip on Condition Register True - Backward
SF	Skip Unconditional - Forward
SB	Skip Unconditional - Backward

B @M(R)



FUNCTION: Branches unconditionally to a specified memory location. This instruction differs from BCH which uses pre-indexing; B uses post-indexing.

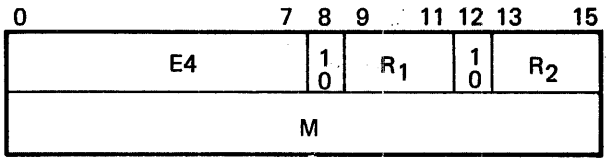
OPERAND: A 16-bit value in bits 16-31 of the instruction that may range from 0-65,535. It is the memory location to which the program branches. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

EXAMPLE: B @TAG(4)

The program branches unconditionally to the address identified by @TAG(4).

Branch Add One

BAI @M(R_1),@R_2



FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is increased by a value of 1, and the program branches to a specified memory location.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the tested field is not zero. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

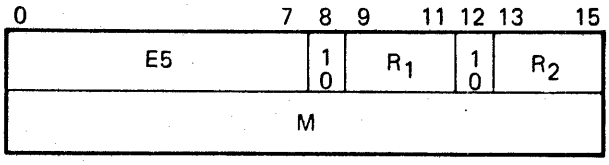
OPERAND 2: The value tested; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: BAI TAG(3),@2

The 16-bit field is tested at the address in register 2; if the field is non-zero, a value of 1 is added to the field and the program branches to the address identified by TAG(3). If the field tested is zero, the program continues with the next instruction.

Branch Add Two

BA2 @M(R_1),@R_2



FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is increased by a value of 2, and the program branches to a specified memory location.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the tested field is not zero. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

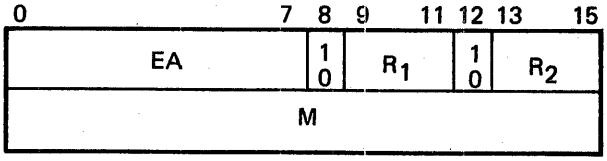
OPERAND 2: The value tested; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: BA2 TAG(4),@1

The 16-bit field is tested at the address in register 1; if the field is non-zero, a value of 2 is added to this field and the program branches to the address identified by TAG(4). If the field is zero when the test is made, no branch is performed and the program continues with the next instruction.

Branch and Save Return

BSR @M(R_1),@R_2



FUNCTION: Branches unconditionally to a specified memory address, storing the address of the next instruction in a general register. The address stored (return address) is the current program address plus four bytes.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

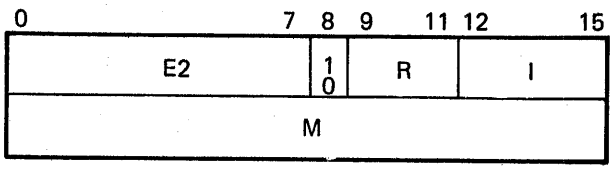
OPERAND 2: The general register specified by R_2, or the memory field if indirect addressing is used (bit 12=1), that upon completion contains the return address.

EXAMPLE: BSR TAG(6),5

Branches unconditionally to the address identified by TAG(6) and stores the next instruction address (current program address plus four bytes) into register 5. After the instructions beginning at TAG(6) are executed, the program continues with the instruction at the address in register 5.

Branch if Bit Off

BOF @M(R),I



FUNCTION: Branches to a specified memory location if the bit tested in a general register is off. If the bit is on, the next instruction in the program is executed.

OPERAND 1: The operand is composed of two parts: the general register tested is specified by R, and the 16-bit value contained in M is the memory address to which the program branches. Addressing options to the base address M include indirect addressing (bit 8=1), but not indexing.

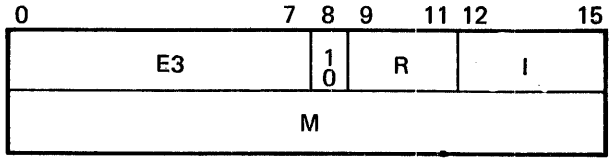
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. This value specifies the position of the bit to be tested in the general register and may range from 1-15.

EXAMPLE: BOF @TAG(3),14

Branches to the address identified by @TAG(3) if bit 14 of register 3 is off (0). If bit 14 is on (1), the next instruction is read.

Branch if Bit On

BON @M(R),I



FUNCTION: Branches to a specified memory location if the bit tested in a general register is on. If the bit is off, the next instruction in the program is executed.

OPERAND 1: The operand is composed of two parts: the general register tested is specified by R, and the 16-bit value contained in M is the memory address to which the program branches. Addressing options to the base address M include indirect addressing (bit 8=1), but not indexing.

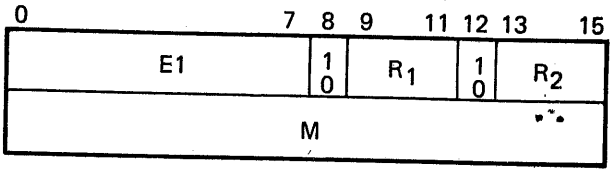
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. This value specifies the position of the bit to be tested in the general register and may range from 1-15.

EXAMPLE: BON TAG(5),9

This instruction tests bit 9 in register 5. If bit 9 is on (1), the program branches to the address identified by TAG. Otherwise, no branch is made and the next instruction pointed to by the program counter is read.

Branch if Register is Not Zero

BRN @M(R_1),@R_2



FUNCTION: Branches to a specified memory location if the general register tested does not contain all zeros. If the register contains all zeros, the next instruction in the program is executed.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the general register does not contain all zeros. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

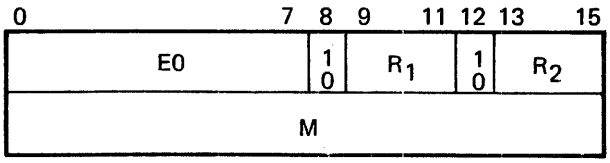
OPERAND 2: The value tested; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: BRN TAG(2),5

The program branches to the address specified by TAG(2) if the contents of register 5 are not zeros; if the contents are zeros, the next instruction is read.

Branch if Register is Zero

BRZ @M(R_1),@R_2



FUNCTION: Branches to a specified memory location if the general register tested contains all zeros. If the register does not contain all zeros, the next instruction in the program is executed.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the general register contains all zeros. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

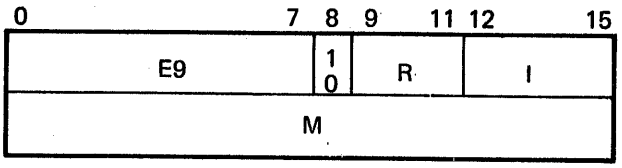
OPERAND 2: The value tested; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: BRZ TAG,6

The program branches to the address of TAG if the contents of register 6 are all zeros; if the contents are not all zeros, the next instruction is read.

Branch on Condition Register False

BCF @M(R),I



FUNCTION: Branches to a specified memory location if a designated Condition register bit is off. If the bit is on, the next instruction in the program is executed.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the designated Condition register bit is on. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

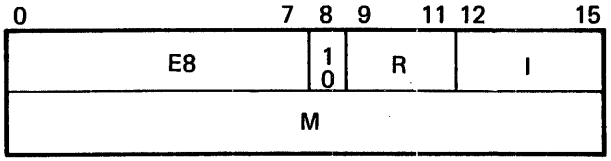
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The value specifies the position of the bit to be tested in the Condition register and may range from 0-15.

EXAMPLE: BCF @TAG(7),11

Branches to the location specified by @TAG(7) if bit 11 of the Condition register is off (0). If bit 11 is on (1), the next instruction is read.

Branch on Condition Register True

BCT @M(R),I



FUNCTION: Branches to a specified memory location if a designated Condition register bit is on. If the bit is off the next instruction in the program is executed.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the designated Condition register bit is on. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

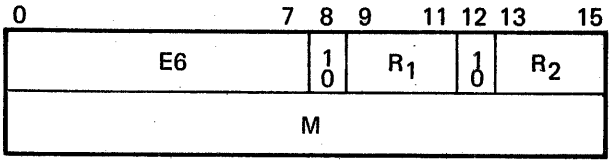
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The value specifies the position of the bit to be tested in the Condition register and may range from 0-15.

EXAMPLE: BCT @TAG(3),8

Branches to the address identified by @TAG(3) if bit 8 of the Condition register is on (1). If the bit is off (0), no branch is made and the next instruction is read.

Branch Subtract One

BS1 @M(R_1),@R_2



FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is decreased by a value of 1, and the program branches to a specified memory location.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program branches if the tested field is not zero. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

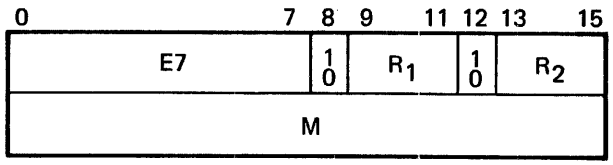
OPERAND 2: The value tested; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: BS1 @TAG(5),@2

Tests 16-bit field at the address in register 2. If the field contains a non-zero value, a branch is made to the address identified by @TAG(5), and the field value is decreased by 1. If the tested field is zero, the next instruction is read.

Branch Subtract Two

BS2 @M(R_1),@R_2



FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is decreased by a value of 2 and the program jumps to a specified memory location.

OPERAND 1: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program jumps if the tested field is not zero. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

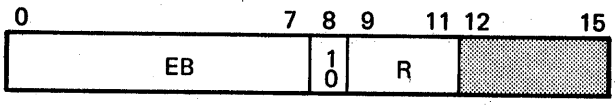
OPERAND 2: The value tested; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: BS2 @TAG(5),@2

Same as BS1, but decreases value in tested field by 2 after branch is made.

Branch to Address in Register

BR @R



FUNCTION: Causes an unconditional branch to a specified memory location.

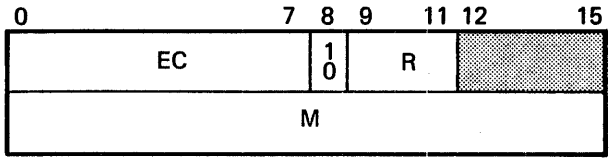
OPERAND: A 3-bit unsigned value in bits 9-11 of the instruction. It points to the address to which the program jumps. The address may be in the general register specified by R or in memory if indirect addressing is used, bit 8=1.

EXAMPLE: BR 4

Branches to the location specified in register 4.

Branch Unconditional

BCH @M(R)



FUNCTION: Branches unconditionally to a specified memory location. This instruction differs from B which uses post-indexing; BCH uses pre-indexing.

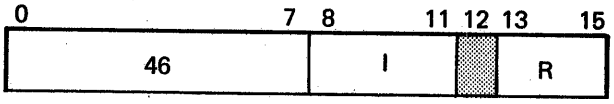
OPERAND: A 16-bit value in bits 16-31 of the instruction. It is the memory location to which the program jumps. Addressing options to the base address M include indexing (R), indirect addressing (bit 8=1), or a combination of both.

EXAMPLE: BCH @TAG(5)

Branches to the address identified by @TAG(5). The value of TAG is added to the contents of register 5; the address formed is the address of an address to which the program will branch (this is the pre-indexing technique).

Skip if Register Minus - Forward

SRMF I,R



FUNCTION: Skips forward a specified number of words if the register contents tested are minus. If the register contents are plus, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and added to the current program address.

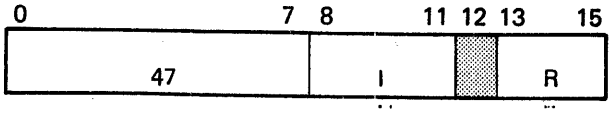
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRMF TAG,3

Tests the contents of register 3. If negative, skip to instruction located at TAG; if positive or zero, the next instruction is read.

Skip if Register Minus - Backward

SRMB I,R



FUNCTION: Skips back a specified number of words if the register contents tested are minus. If the register contents are plus, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and subtracted from the current program address.

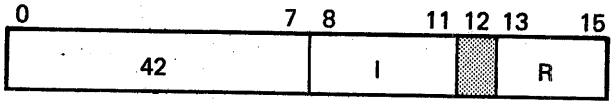
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRMB TAG,3

Tests the contents of register 3. If negative, skip to instruction located at TAG; if positive or zero, the next instruction is read.

Skip if Register Not Zero - Forward

SRNF I,R



FUNCTION: Skips forward a specified number of words if the register contents tested are not zero. If the register does contain all zeros, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and added to the current program address.

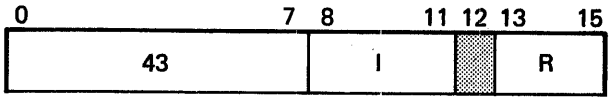
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRNF TAG,2

Tests the contents of register 2. If non-zero, skip to instruction located at TAG; if zero, read next instruction.

Skip if Register Not Zero - Backward

SRNB I,R



FUNCTION: Skips back a specified number of words if the register contents tested are not zero. If the register does contain all zeros, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and added to the current program address.

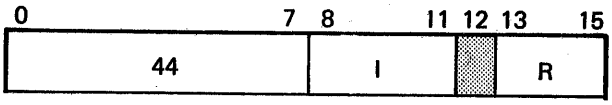
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRNB TAG,2

Tests the contents of register 2. If non-zero, skip to instruction located at TAG; if zero, read next instruction.

Skip if Register Plus - Forward

SRPF I,R



FUNCTION: Skips forward a specified number of words if the register contents tested are plus. If the register contents are minus, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and added to the current program address.

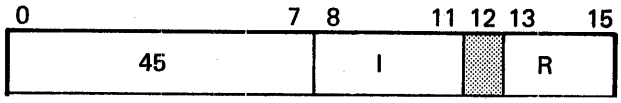
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRPF TAG,4

Tests the value in register 4. If positive, skips to the instruction at location TAG; if not positive, the next instruction is read. TAG must be within 15 words of SRP.

Skip if Register Plus - Backward

SRPB I,R



FUNCTION: Skips back a specified number of words if the register contents tested are plus. If the register contents are minus, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and subtracted from the current program address.

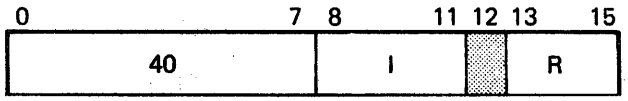
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRPB TAG,4

Tests the value in register 4. If positive, skips to the instruction at location TAG; if not positive, the next instruction is read. TAG must be within 15 words of SRP.

Skip if Register Zero - Forward

SRZF I,R



FUNCTION: Skips forward a specified number of words if the register tested contains all zeros. If the register does not contain all zeros, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and added to the current program address.

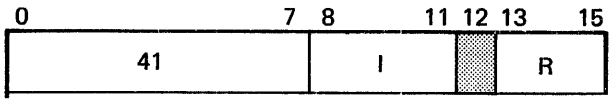
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRZF TAG,5

Tests the value in register 5. If all zeros, read the next instruction at location TAG; if not all zeros, the next instruction is read. TAG must be within 15 words of SRZ.

Skip if Register Zero - Backward

SRZB I,R



FUNCTION: Skips back a specified number of words if the register tested contains all zeros. If the register does not contain all zeros, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. The value is multiplied by 2 and subtracted from the current program address.

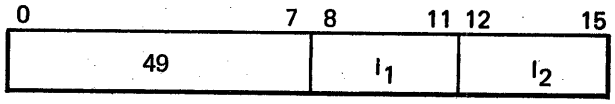
OPERAND 2: The value tested; a one-word field in the general register specified by R.

EXAMPLE: SRZB TAG,5

Tests the value in register 5. If all zeros, read the next instruction at location TAG; if not all zeros, the next instruction is read. TAG must be within 15 words of SRZ.

Skip on Condition Register False - Forward

SCFF I_1, I_2



FUNCTION: Skips forward a specified number of words if the appropriate Condition register bit is off. If the bit is on, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. It is multiplied by 2 and added to the current program address.

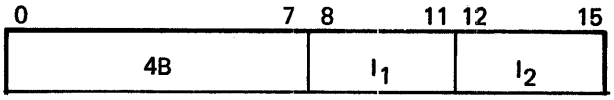
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. This value specifies the position of the bit tested for off in the Condition register and may range from 0-15.

EXAMPLE: SCFF 12,3

If bit 3 (equal bit) in the Condition register is off, the program will skip forward in the program 12 words (24 bytes). If bit 3 is on, the next instruction in the program is executed.

Skip on Condition Register False - Backward

SCFB I_1, I_2



FUNCTION: Skips back a specified number of words if the appropriate Condition register bit is off. If the bit is on, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. Is multiplied by 2 and subtracted from the current program address.

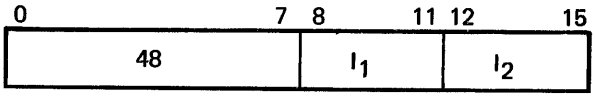
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. This value specifies the position of the bit tested for off in the Condition register and may range from 0-15.

EXAMPLE: SCFB 12,3

If bit 3 (equal bit) in the Condition register is off, the program will skip back in the program 12 words (24 bytes). If bit 3 is on, the next instruction in the program is executed.

Skip on Condition Register True - Forward

SCTF I_1, I_2



FUNCTION: Skips forward a specified number of words if the appropriate Condition register bit is on. If the bit is off, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. It is multiplied by 2 and added to the current program address.

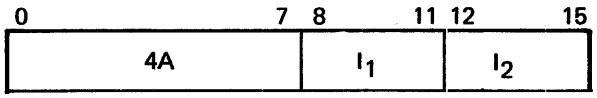
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. This value specifies the position of the bit tested for on in the Condition register and may range from 0-15.

EXAMPLE: SCTF TAG,0

Assume that TAG is six words ahead of this instruction. If bit 0 (overflow bit) in the Condition register is on, the program will skip six words (12 bytes) forward to TAG. If bit 0 is off, the next instruction in the program is executed.

Skip on Condition Register True - Backward

SCTB I_1, I_2



FUNCTION: Skips back a specified number of words if the appropriate Condition register bit is on. If the bit is off, the next instruction in the program is executed.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the number of words to skip and may range from 0-15. It is multiplied by 2 and subtracted from the current program address.

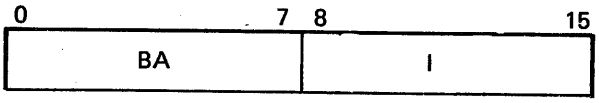
OPERAND 2: A 4-bit value in bits 12-15 of the instruction. This value specifies the position of the bit tested for on in the Condition register and may range from 0-15.

EXAMPLE: SCTB TAG,0

Assume that TAG is six words behind this instruction. If bit 0 (overflow bit) in the Condition register is on, the program will skip back six words (12 bytes) to TAG. If bit 0 is off, the next instruction in the program is executed.

Skip Unconditional - Forward

SF I or M



FUNCTION: Skips forward a specified number of words.

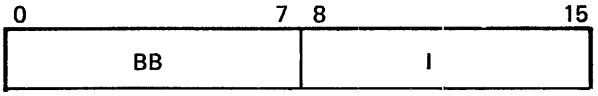
OPERAND: An 8-bit unsigned value in bits 8-15 of the instruction. This value specifies the number of words to skip and may range from 0-255. It is multiplied by 2 and added to the current program address.

EXAMPLE: SF TAG

Skips unconditionally to TAG.

Skip Unconditional - Backward

SB I or M



FUNCTION: Skips back a specified number of words.

OPERAND: An 8-bit unsigned value in bits 8-15 of the instruction. This value specifies the number of words to skip and may range from 0-255. It is multiplied by 2 and subtracted from the current program address.

EXAMPLE: SB TAG

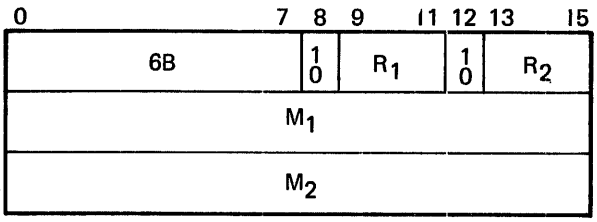
Skips unconditionally to TAG.

COMPARE INSTRUCTIONS

Mnemonic Code	Name
CBYM	Compare Byte Memory - Memory ●
CBY	Compare Byte Memory - Register ●
CMPX	Compare Characters ●
CMPD	Compare Direct
CMPI	Compare Immediate
CMPM	Compare Memory - Memory
CMP	Compare Memory - Register
CMPK	Compare Packed Decimal ●
CMPR	Compare Register - Register
CMPT	Compare Two-Word

Compare Byte Memory – Memory ●

CBYM @M_1(R_1),@M_2(R_2)



FUNCTION: Performs a magnitude-only comparison of one-byte fields in memory.

OPERAND 1: A one-byte field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows:

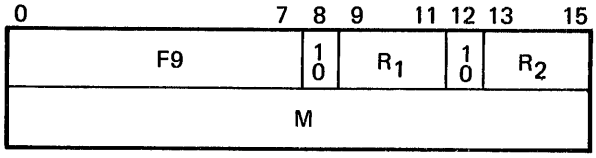
- Bits 0 and 4 are always cleared.
- If operand 1 is greater than operand 2, bits 1 and 5 are set and bits 2, 3, 6 and 7 are cleared.
- If operand 1 is less than operand 2, bits 2 and 6 are set and bits 1, 3, 5 and 7 are cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.

EXAMPLE: CBYM @TAG(4),@HERE(2)

Compares a one-byte operand at the address specified by @TAG(4) with another at the address specified by @HERE(2). If the operand at @TAG(4) is greater than the other operand, bit 1 of the Condition register is turned on; if less than the other, bit 2 of the Condition register is turned on; if they are equal bit 3 of the Condition register is turned on. (Only one of these bits in the Condition register will be turned on; the others remain off.)

Compare Byte Memory — Register ●

CBY @M(R_1),@R_2



FUNCTION: Performs a magnitude-only comparison of a one-byte field in memory and the low-order byte of a general register or a one-byte field in memory.

OPERAND 1: A one-byte field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-byte field (bits 8-15) in a general register specified by R_2 or a one-byte field in memory if indirect addressing is used, bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows:

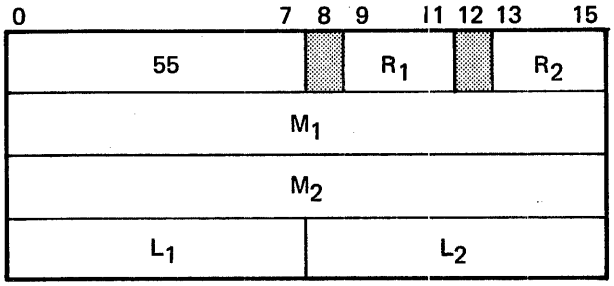
- Bits 0 and 4 are always cleared.
- If operand 1 is greater than operand 2, bits 1 and 5 are set and bits 2, 3, 6 and 7 are cleared.
- If operand 1 is less than operand 2, bits 2 and 6 are set and bits 1, 3, 5 and 7 are cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.

EXAMPLE: CBY @TAG(4),6

Compares a one-byte operand identified by @TAG(4) with the rightmost byte of register 6, and the Condition register is set accordingly.

Compare Characters •

CMPX $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Performs a magnitude-only comparison of two fields in memory. The field lengths may vary from 0-255 bytes. The comparison is byte-by-byte and proceeds from left to right. The operation continues until either of the following occurs: the operands are found unequal or the greater of L_1 or L_2 is exhausted.

OPERAND 1: A field in memory. The field length, 0-255 bytes, is specified by the L_1 value in the instruction. Addressing options to the base address M_1 include only indexing (R_1).

OPERAND 2: A field in memory. The field length, 0-255 bytes, is specified by the L_2 value in the instruction. Addressing options to the base address M_2 include only indexing (R_2).

RESULTS: The following conditions may occur, depending on the values of L_1 and L_2.

- If $L_1 = L_2$, the operands are compared byte-for-byte.
- If L_1 is less than L_2, the operands are compared until L_1 is exhausted, then blanks are compared to operand 2.
- If L_1 is greater than L_2, the operands are compared until L_2 is exhausted, then operand 1 is compared to blanks.
- If $L_1 = 0$ and $L_2 \neq 0$, blanks are compared to operand 2.
- If $L_2 = 0$ and $L_1 = 0$, no compare is performed.

Neither operand is disturbed, but the Condition register is affected as follows:

- Bit 0 (overflow) is always cleared.
- If operand 1 is greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 2 is equal to operand 1, bit 3 is set and bits 1 and 2 are cleared.
- If L1=0 and L2=0, bits 1 and 2 are cleared and bit 3 is set.

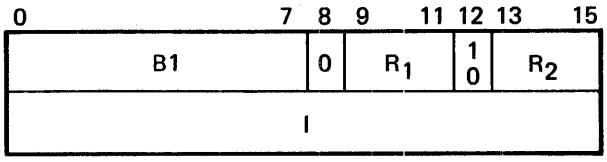
Note: CMPX is a word compare if L_1 and L_2 are both even and the beginning addresses are even.

EXAMPLE: CMPX TAG(200,1),HERE(200,2)

A 200-byte field identified by TAG(200,1) is compared to a 200-byte field identified by HERE(200,2). Comparison proceeds byte-by-byte until inequality is determined or all bytes have been compared and found equal. The Condition register is set accordingly.

Compare Direct

CMPD I(R_1),@R_2



FUNCTION: Performs a signed arithmetic comparison of a one-word immediate value* and a one-word field in a general register or in memory.

OPERAND 1: A 16-bit immediate signed value in bits 16-31 of the instruction; the value may range from -32,768 to +32,767.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows (bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare):

- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 1 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is logically greater than operand 2, bit 5 is set and bits 6 and 7 are cleared.
- If operand 1 is logically less than operand 2, bit 6 is set and bits 5 and 7 are cleared.

For arithmetic results, $7FFF_{16}$ is the largest number and 8000_{16} is the smallest number.

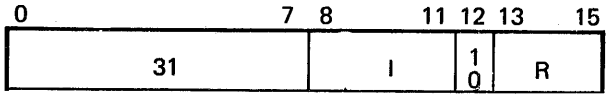
For logical results, $FFFF_{16}$ is the largest number and 0000_{16} is the smallest number.

EXAMPLE: CMPD -25000(5),@3

The value -25,000 modified by the contents of register 5 is compared with the value at the location specified in register 3; the Condition register is set accordingly.

Compare Immediate

CMPI I,@R



FUNCTION: Performs a signed arithmetic comparison of a 4-bit immediate value and a one-word field in a general register or in memory.

OPERAND 1: A 4-bit signed value in bits 8-11 of the instruction; the value may range from 0-15. The 4-bit value is compared with operand 2 in bit positions 12-15 with bits 0-11 zeros.

OPERAND 2: A one-word field located in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

RESULTS: Operand 1 is not disturbed, but the Condition register is affected as follows (bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results):

- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 1 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is logically greater than operand 2, bit 5 is set and bits 6 and 7 are cleared.
- If operand 1 is logically less than operand 2, bit 6 is set and bits 5 and 7 are cleared.

For arithmetic results, $7FFF_{16}$ is the largest number and 8000_{16} is the smallest number.

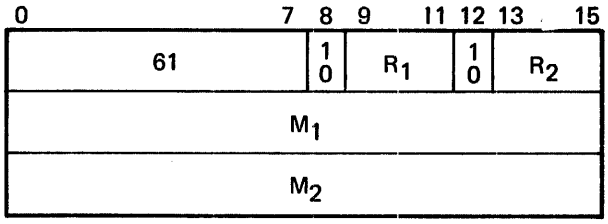
For logical results, $FFFF_{16}$ is the largest number and 0000_{16} is the smallest number.

EXAMPLE: CMPI 11,@6

Compares the immediate value 11 to the value at the location specified in register 6 and sets the Condition register accordingly.

Compare Memory — Memory

CMPM @M_1(R_1),@M_2(R_2)



FUNCTION: Performs a signed arithmetic comparison of two one-word fields in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows (bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results):

- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 1 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is logically greater than operand 2, bit 5 is set and bits 6 and 7 are cleared.
- If operand 1 is logically less than operand 2, bit 6 is set and bits 5 and 7 are cleared.

For arithmetic results, $7FFF_{16}$ is the largest number and 8000_{16} is the smallest number.

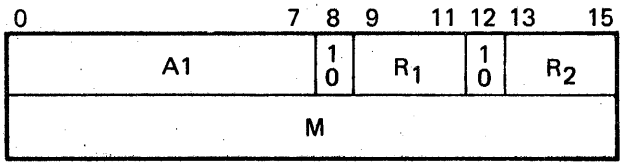
For logical results, $FFFF_{16}$ is the largest number and 0000_{16} is the smallest number.

EXAMPLE: CMPM @HERE(4),@TAG(6)

A 16-bit value at the address specified by @HERE(4) is compared to a 16-bit value at the address identified by @TAG(6); the Condition register is set accordingly.

Compare Memory — Register

CMP @M(R_1),@R_2



FUNCTION: Performs a signed arithmetic comparison of a one-word field in memory and a one-word field in a general register or in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows (bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare):

- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 1 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is logically greater than operand 2, bit 5 is set and bits 6 and 7 are cleared.
- If operand 1 is logically less than operand 2, bit 6 is set and bits 5 and 7 are cleared.

For arithmetic results, $7FFF_{16}$ is the largest number and 8000_{16} is the smallest number.

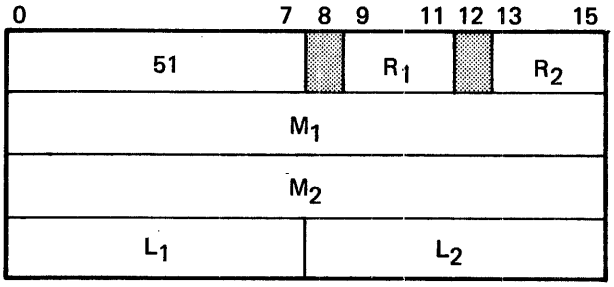
For logical results, $FFFF_{16}$ is the largest number and 0000_{16} is the smallest number.

EXAMPLE: CMP @TAG(5),@6

A 16-bit field at the address identified by @TAG(5) is compared to a 16-bit field at the address specified in register 6; the Condition register is set accordingly.

Compare Packed Decimal •

CMPK $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Performs a comparison of packed decimal fields in memory; the signs are compared first, then the comparison proceeds digit-by-digit, left to right. The field lengths may vary from 0-255 bytes. The operation continues until either of the following occurs: the operands are found unequal or the greater of L_1 or L_2 is exhausted.

OPERAND 1: A packed decimal field in memory. The field length, 0-255 bytes, is specified by the L_1 value in the instruction. Addressing options to the base address M_1 include only indexing (R_1).

OPERAND 2: A packed decimal field in memory. The field length, 0-255 bytes, is specified by the L_2 value in the instruction. Addressing options to the base address M_2 include only indexing (R_2).

RESULTS: The following conditions may occur, depending on the values of L_1 and L_2:

- If $L_1 = L_2$, the operands are compared digit-by-digit.
- If L_1 is less than L_2, the operands are compared until L_1 is exhausted, then zeros are compared to operand 2.
- If L_1 is greater than L_2, the operands are compared until L_2 is exhausted, then zeros are compared to operand 1.
- If $L_1 = 0$ and $L_2 = 0$, bit 3 of the Condition register is set and bits 1 and 2 are cleared.

Neither operand is disturbed, but the Condition register is affected as follows:

- Bit 0 is always cleared.
- If operand 1 is greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is equal to operand 2, bit 3 is set and bits 1 and 2 are cleared.

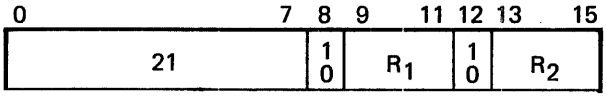
EXAMPLE: CMPK TAG(90,1),HERE(101,3)

Two packed decimal fields are compared; the Condition register is set accordingly. In this example, the field represented by TAG(90,1) is shorter than the other; therefore, bytes 91 through 101 of the larger field, HERE(101,3), will be compared to zeros if an inequality determination cannot be made before exhaustion of the smaller field.

Since the signs are checked first, an inequality decision could be made immediately if the signs are different.

Compare Register – Register

CMPR @R_1,@R_2



FUNCTION: Performs a signed arithmetic comparison of two one-word fields; either field may be in a register or in memory.

OPERAND 1: A one-word field located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: Same as operand 1 except use R_2 and bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows (bits 0-3 reflect the arithmetic results of the compare, and bits 4-7 reflect the logical results):

- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 1 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is logically greater than operand 2, bit 5 is set and bits 6 and 7 are cleared.
- If operand 1 is logically less than operand 2, bit 6 is set and bits 5 and 7 are cleared.

For arithmetic results, $7FFF_{16}$ is the largest number and 8000_{16} is the smallest number.

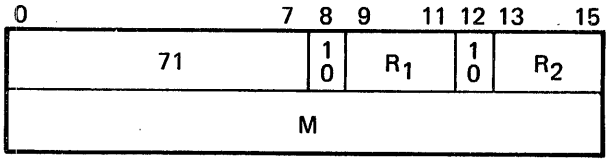
For logical results, $FFFF_{16}$ is the largest number and 0000_{16} is the smallest number.

EXAMPLE: CMPR @7,@5

A 16-bit field at a location specified in register 7 is compared to a 16-bit field at a location specified in register 5; the Condition register is set accordingly.

Compare Two-Word

CMPT @M(R_1),@R_2



FUNCTION: Performs a signed arithmetic comparison of a two-word field in memory and a two-word field in two general registers or in memory.

OPERAND 1: A two-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. The effective address points to the most significant bits of the two-word field.

OPERAND 2: A two-word field in two general registers (the most significant bits in the register specified in R_2 and the least significant bits in the register specified by R_2 + 1) or in memory if indirect addressing is used, bit 12=1.

RESULTS: Neither operand is disturbed, but the Condition register is affected as follows (bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results):

- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand B, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 2 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and bits 1 and 3 are cleared.
- If operand 1 is logically greater than operand 2, bit 5 is set and bits 6 and 7 are cleared.
- If operand 1 is logically less than operand 2, bit 6 is set and bits 5 and 7 are cleared.

For arithmetic results, $7FFF_{16}$ is the largest number and 8000_{16} is the smallest number.

For logical results, $FFFF_{16}$ is the largest number and 0000_{16} is the smallest number.

EXAMPLE: CMPT @TAG(4),@1

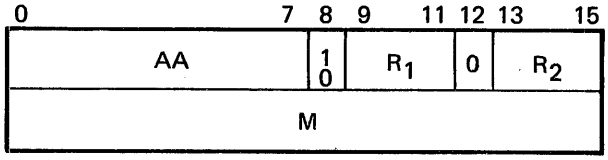
A 32-bit field at the address identified by @TAG(4) is compared to a 32-bit field at the address specified in register 1; the Condition register is set accordingly.

DATA CONVERSION INSTRUCTIONS

Mnemonic Code	Name
CVB	Convert to Binary ●
CVBT	Convert to Binary Two-Word ●
CVD	Convert to Decimal ●
CVDT	Convert to Decimal Two-Word ●
EDTX	Packed Decimal/Alpha Edit ●
PAKX	Pack ●
TRNX	Translate ●
UNPX	Unpack ●

Convert to Binary •

CVB @M(R_1),R_2



FUNCTION: Converts a 3-byte packed decimal field in memory to a 2-byte binary field in a general register.

Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction.

OPERAND 1: A 3-byte packed decimal field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. The packed decimal field may hold five digits and a sign. The effective address points to the most significant byte of the field

OPERAND 2: The resultant 2-byte signed binary value in the general register specified by R_2. The binary value has 15 bits and a sign bit.

RESULTS: The resulting value resides at the operand 2 location.

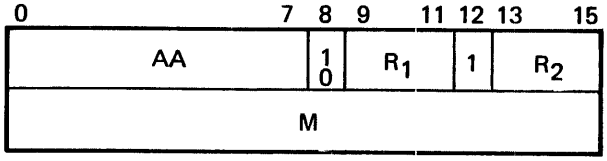
- Bit 0 (overflow) is set if results are greater than +32,767 or less than -32,767. (Note: -32,768 is converted correctly but the overflow bit is set.)
- Bits 1-3 unchanged.

EXAMPLE: CVB @TAG(4),6

@TAG(4) yields the address of a 3-byte packed decimal field (five digits plus sign) which is converted to a 16-bit (15 bits plus sign) binary value and loaded into register 6.

Convert to Binary Two-Word •

CVBT @M(R_1),R_2



FUNCTION: Converts a 6-byte packed decimal field in memory to a 4-byte binary field in two general registers.

Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for this instruction.

OPERAND 1: A 6-byte packed decimal field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. The packed decimal field may hold 11 digits and a sign. The effective address points to the most significant byte of the field.

OPERAND 2: The resultant 4-byte signed binary value located in two general registers (the most significant bits in the register specified by R_2 and the least significant bits in the register R_2+1). (Note: If register 7 is specified by R_2, the most significant bits are placed in register 7 and the least significant bits in register 0.) The binary value has 31 bits of magnitude and a sign bit.

RESULTS: The resulting value resides at the operand 2 location.

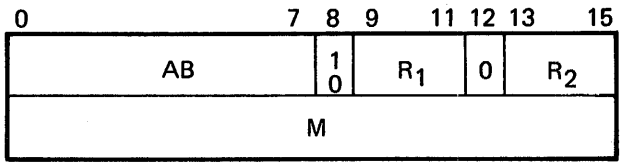
- Bit 0 (overflow) is set if results are greater than $+2^{31}-1$ or less than $-2^{31}-1$. (Note: -2^{31} is converted correctly but the overflow bit is set.)
- Bits 1-3 are unchanged.

EXAMPLE: CVBT @TAG(3),7

@TAG(3) yields the address of a 6-byte packed decimal field (11 digits plus sign) which is converted to a 32-bit (include sign) binary value and loaded into register 7.

Convert to Decimal ●

CVD @M(R_1),R_2



FUNCTION: Converts a 2-byte binary field in a general register to a 3-byte packed decimal field in memory.

Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction.

OPERAND 1: The resultant 3-byte packed decimal field in memory which can hold as many as five digits and a sign. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. The effective address points to the most significant byte of the field.

OPERAND 2: A 2-byte signed binary value in the general register specified by R_2. The binary value has 15 bits and a sign bit.

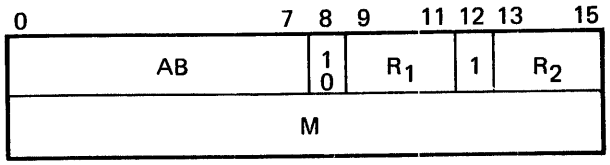
RESULTS: The resulting value resides at the operand 1 location.

EXAMPLE: CVD @TAG(1),2

Register 2 contains a 16-bit binary value which is converted to a 3-byte packed decimal field and stored at the location specified by @TAG(1).

Convert to Decimal Two-Word •

CVDT @M(R_1),R_2



FUNCTION: Converts a 4-byte signed binary field located in two general registers to a 6-byte packed decimal field located in memory.

Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for the CVDT instruction.

OPERAND 1: The resultant 6-byte packed decimal field located in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. The packed decimal field may hold 11 digits and a sign. The effective address points to the most significant byte of the field.

OPERAND 2: A 4-byte signed binary value located in two general registers. The most significant bits are in the register specified by R_2 and the least significant bits in register R_2+1. The binary value has 31 bits and a sign bit. (Note: If register 7 is specified by R_2, the most significant bits are in register 7 and the least significant bits are in register 0.)

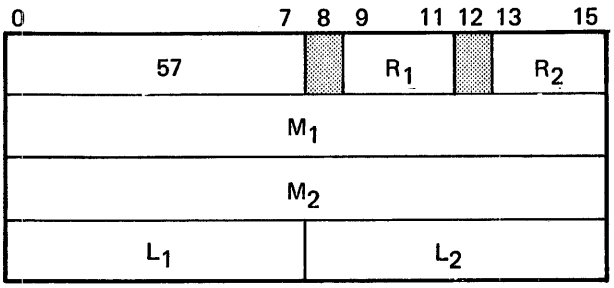
RESULTS: The resulting value resides at the operand 1 location.

EXAMPLE: CVDT @TAG(4),3

Registers 3 and 4 contain a 32-bit binary value which is converted to 6-byte packed decimal (11 digits plus sign) and stored at the location identified by @TAG(4).

Packed Decimal/Alpha Edit •

EDTX $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: This instruction moves the contents of a source field to a result field with editing symbols inserted according to an edit mask. The first-byte address of the mask field must be set in general register 1 prior to execution of the edit instruction. A more complete description of the functions performed and details of the edit mask

OPERAND 1: The field in memory to be edited. It must be a packed decimal field for numeric editing; for alpha editing it must be an EBCDIC field. For numeric editing, the number of digits in the source field is specified in L_1; for alpha editing, no length is specified. Addressing options to the base address include only indexing (R_1).

OPERAND 2: The field in memory that will hold the edited results. It will always be an EBCDIC field. For numeric editing, the length in bytes of the result field is specified in L_2. For alpha editing, L_2 must be zero. Addressing options to the base address M_2 include only indexing (R_2).

RESULTS: An EBCDIC field at the operand 2 location. Results depend on the type of editing and on the contents of the edit mask.

EXAMPLE: EDTX TAG(5),HERE(6)

No length is specified because, in the example, the field to be edited is assumed to be an alpha field; as such, both fields are EBCDIC.

The location TAG(5) contains the alpha field to be edited, and the field at the address HERE(6) receives the editing symbols generated during the edit.

DETAILED DESCRIPTION OF EDIT

The EDTX instruction performs both numeric editing and alpha editing.

The source field is moved to the result field with editing symbols inserted according to the edit mask. The result field is always an EBCDIC field. The source field must be a packed decimal field when numeric editing is requested ($L_2 \neq 0$). The source field must be an EBCDIC field when alpha editing is requested ($L_2 = 0$).

The editing function is terminated as dictated by the edit mask. The length specifications (L_1 and L_2) are used when numeric editing is requested to unpack the source (using UNPX) before actual editing begins.

Upon return from a numeric edit, general register 1 contains the byte address of the last nonsignificant (F0 value) character. This address is used to store the float character if desired. If there is no significant character (source field has zero value), general register 1 will be set to zero. This register will always be set to zero following an alpha edit.

The Condition register is set as follows when numeric editing is requested.

- Bit 3 set — All source digits were zero
- Bit 2 set — The source field was negative
- Bit 1 set — The source field was positive

The Condition register is not used or modified when alpha editing is requested.

Edit Mask: Editing is accomplished by means of an edit control technique. The mask field, which is referenced (but not changed) by the EDTX instruction, is used to control data movement from the source field to the result field. The mask field is made up of a string of one character (two hexadecimal digits) edit operators and EBCDIC insert characters. The edit operators are basically control functions directing the edit microcode rather than the traditional mask used by the edit microcode to drive the editing function.

To facilitate a clear understanding of the editing process, the following microcode indicators are defined. These indicators are internal to the microcode and not directly accessible by the user. They are initialized by the microcode as defined below. The edit mask operators direct the resetting and use of these indicators.

There are two microcode indicators and one 8-bit value field which the edit microcode requires to effect the execution of the EDTX instruction.

SD - Significance Digit Indicator

Initially SD is set to zero and is set to one when significance is detected (by edit operator or by occurrence of a non-zero digit in the source).

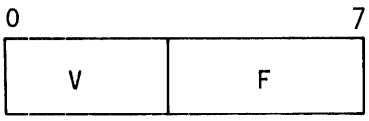
SG - Sign Indicator

The SG indicator is set according to the Condition register after the source has been unpacked.
EQ = Bit 3 set - source is equal to zero
LT = Bit 2 set - source is negative
GT = Bit 1 set - source is positive

FI - Fill Character - 8-bit EBCDIC

Initially set with EBCDIC space (40 hex). The fill character may be specified via the Set Fill edit operator to any EBCDIC value.

The edit operators within the mask field have the following format.



The first (zone) digit of each edit operator character is the variant specifier, V. The second (numeric) digit is the function specifier, F. When used, V specifies either repeat count of the function F or sub-control information. Otherwise V is ignored.

The edit operators divide into three functional categories.

- Data Transfer
- Data Insert
- Control

Data Transfer operators specify conditional and unconditional transfer of data from the source field to the result field.

Data Insert operators specify conditional and unconditional insertion of characters into the result field where there is no dependency or reference to the source field.

Control operators function as explicit edit performance controls where there is no reference directly to the source or result fields.

The function code F specifies the operation to be performed. These codes have a numerical 4-bit hexadecimal assignment. Following is a list of the edit operators with the F code divided into categories.

CATEGORY	OPERATOR	F-HEX	F-BINARY	OPERATOR DESCRIPTION
Data Transfer	MC	8	1000	Move character
	MCS	9	1001	Move character suppress
Data Insert	IC	4	0100	Insert character
	ICS	5	0101	Insert character suppress
	ISG	7	0111	Insert sign
Control	TE	0	0000	Terminate edit
	SSD	2	0010	Set significance (SD)
	SFI	6	0110	Set fill (FI)

The IC, ICS, and SFI operators require one insert character following the edit operator. The ISG operator requires either one or two insert characters following the ISG edit operator. In all cases the insert characters are bypassed automatically by the microcode to obtain the next edit operator.

Numeric Editing: The edit operators function as follows during a numeric edit.

MC - Move Character F = 8

- If SD equals zero, perform the SSD operation (absolute).
- Move a character from the unpacked source field to the result field.
- V specifies a repeat count (0-15).

MCS - Move Character Suppress F = 9

- If SD equals one, perform the MC operation.
- If SD equals zero and the next source character equals zero, move the fill character from FI to the result field.

- If SD equals zero and the next source character is non-zero, perform the MC operation (SD gets set equal to one by MC).

- V specifies a repeat count (0-15).

IC - Insert Character F = 4

- Move the character following this edit operator to the result field.

- V specifies a repeat count (0-15). The same character will be inserted V+1 number of times.

ICS - Insert Character Suppress F = 5

- If SD equals one, perform the IC operation.

- If SD equals zero, move the fill character from F1 to the result field.

- V specifies a repeat count (0-15). The same character (fill character or insert character) will be inserted V+1 number of times.

ISG - Insert Sign F = 7

- If SG = LT (negative source) and
  V=0, move the character following this edit operator to the result field;
  V=1, move the character following the edit operator to the result field;
  V=2, move the two characters following this edit operator to the result field.

- If SG = EQ or GT (positive source) and
  V=0, move a + (4E hex) to the result field;
  V=1, move a space (40 hex) to the result field;
  V=2, move two spaces to the result field.

- V is a sub-control function specifying the type of sign inserted.

TE - Terminate Edit F = 0

- Immediately terminates the EDTX instruction.

- The Condition register has been set to EQ, GT, or LT.

- General register 1 is set to the address - 1 of the first significant character within the source. It is set to zero if there was no significance found.

- V is not used.

SSD - Set Significance F = 2

V is a sub-control function specifying whether absolute or conditional set significance is requested.

If V = 1 - Absolute Set Significance
1) Set SD equal to one.
2) Set current result field address - 1 in general register 1 as float address.

If V = 0 - Conditional Set Significance
1) If SD equals one, this is a no operation.
2) If SD equals zero and SG=EQ, this is a no operation.
3) If SD equals zero and SG=GT or LT (source non-zero), perform the absolute set significance.

SFI - Set Fill Character F = 6

- Set FI with the character following this edit operator in the mask field.
- V is not used.

Unusual Conditions in Numeric Editing: The following hexadecimal values are not legal numeric editing functions. If encountered, the following results will be obtained.

- F = 1 - A normal Terminate Edit (TE) will be executed.
- F = 3 - A normal Set Significance (SSD) will be executed.
- F = C - A normal Move Character (MC) will be executed.
- F = D - A normal Move Character Suppress (MCS) will be executed.
- F = A or E
  1) If SD equals one, move source character to result field.
  2) If SD equals zero, perform the absolute Set Significance operation, skip the next source character.
  3) V is ignored.
- F = B or F
  1) If SD equals one, move the source character to the result field.
  2) If SD equals zero and the source character is non-zero, perform the absolute Set Significance; skip the next source character.
  3) If SD equals zero and the source character is zero, move the FI value to the result field.
  4) V is ignored.

Since the source field is unpacked into the result field, right justified, the length specification L_2 must be greater than L_1.

If L_2 is equal to or less than L_1, the source character could possibly be replaced by editing insert characters and unpredictable results may be obtained.

Alpha Editing: Alpha editing is performed when $L_2 = 0$. The edit operators generally used to effect alpha editing are TE, IC, and MC.

The SSD, SFI, and ICS will function but are not generally of use in the alpha editing. Following is a description of the editing operations.

MC - Move Character F = 8

- Move a character from the source field to the result field.
- V is a repeat count (0-15).
- SD is not set.

IC - Insert Character F = 4

- Same as for numeric editing.

ICS - Insert Character Suppress F = 5

- Same as for numeric editing.

TE - Terminate Edit F = 0

- Immediately terminate the edit and return control to the caller.

SSD - Set Significance F = 2

- If V = 1 and SD = 0, set SD = 1.
- The address of the last byte moved or inserted into the result is placed in R_1.
- If V = 0 or SD = 1, no operation.

SFI - Set Fill F = 6

- Same as for numeric editing.

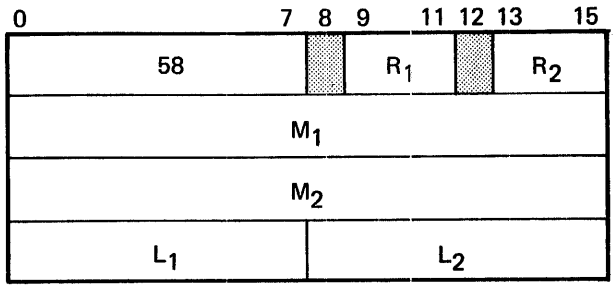
Unusual Conditions in Alpha Editing: The following hexadecimal values are not legal alpha editing functions. If encountered, the following results will be obtained.

- F = 1 - A normal Terminate Edit (TE) will be executed.
- F = 3 - A normal Set Significance (SSD) will be executed.
- F = 7 - The Insert Sign has no meaning in that the Condition register is not set in alpha editing and the SG does not contain a meaningful value.
- F = 9, C, or D - Treated as a normal Move Character (MC).
- F = A, B, E, or F - One character is moved from the source field to the result field. V is ignored.

The Condition register is not used or modified by alpha editing.
General register 1 is set to zero upon return from alpha editing.

Pack •

PAKX $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Converts a zoned decimal field to a packed decimal field. Both fields must be in memory; the field lengths may vary from 0-255. Packing proceeds from right to left until the greater of the field lengths is exhausted.

OPERAND 1: The zoned decimal field; the length of the field, in bytes, is specified by the L_1 value in the instruction. Addressing options to the base address M_1 include only indexing (R_1).

OPERAND 2: The resultant packed decimal field; the length of the field, in bytes, is specified by the L_2 value in the instruction. Addressing options to the base address M_2 include only indexing (R_2).

RESULTS: The resulting field resides at the operand 2 location. Condition register settings are:

1. Overflow (bit 0) is always cleared.

2. Invalid (bit 4) is set if an invalid decimal digit (not 0-9) occurs in operand 1 or if the sign field is not A-F; bits 1-3 are cleared. However, packing continues until the length L_2 is exhausted.

3. No significance in the result (packed field) sets bit 3, clears 1, 2 and 4.

4. Significance and a sign of F, A, C or E sets bit 1 and clears 2-4.

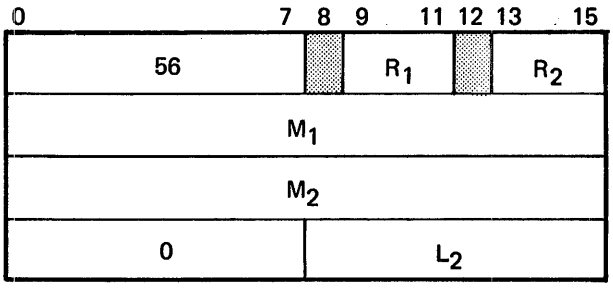
5. Significance and a sign of B or D sets bit 2 and clears 1, 3 and 4.

EXAMPLE: PAKX TAG(239,3),HERE(120,4)

The unpacked (zoned) 239-byte field identified by TAG(239,3) is packed (4-bit zone codes removed) and then stored at the address identified by HERE(120,4).

Translate •

TRNX $M_1(R_1),M_2(L_2,R_2)$



FUNCTION: Translates, byte-by-byte, the contents of a memory field by a table, replacing the source values with the translated table values. The table is in memory at the address specified in general register 1. The table has an assumed length of 256 bytes.

OPERAND 1: The field in memory to be translated. Addressing options to the base address M_1 include only indexing (R_2).

OPERAND 2: The field in memory which holds the translated values. Addressing options to the base address M_2 include only indexing (R_2). The field length, 1-256 bytes, is specified by the L_2 value.

RESULTS: The translated results reside at the operand 2 location.

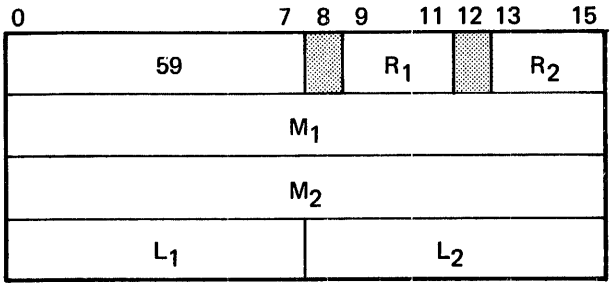
EXAMPLE: TRNX TAG(3),HERE(255,2)

The 256 bytes identified by TAG(3) are moved byte-by-byte to a field identified by HERE(256,2). Each byte value extracted from the TAG(3) field is used as an index to the translate table (address specified in register 1). The byte stored in operand 2 is the translate value at the indexed byte.

Note: The length specified in assembler language is the actual field length, but the length in the machine instruction is one less than the actual length.

Unpack •

UNPX $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Converts a packed decimal field to a zoned decimal field. Both fields must be in memory; the field lengths may vary from 0-255. Unpacking proceeds from right to left. The zoned decimal field must contain at least as many bytes as there are significant digits in the packed decimal field.

OPERAND 1: The packed decimal field; the length of the field, 0-255 bytes, is specified by the L_1 value in the instruction. Addressing options to the base address M_1 include only indexing (R_1).

OPERAND 2: The resultant zoned decimal field; the length of the field, 0-255 bytes, is specified in the L_2 field of the instruction. Addressing options to the base address M_2 include only indexing (R_2).

RESULTS: The resulting field resides at the operand 2 location. Condition register settings are:

1. Bit 0 and bit 4 (invalid) are always cleared.
2. If no significance results bit 3 is set, bit 1 and 2 are cleared.
3. If significance results and the sign is F, A, C, E, 0, 2, 4, 6, 7, or 8, bit 1 is set and bits 2 and 3 are cleared.
4. If significance results and the sign is B, D, 1, 3, 5, or 9, bit 2 is set and bits 1 and 3 are cleared.

EXAMPLE: UNPX TAG(120,4),HERE(239,2)

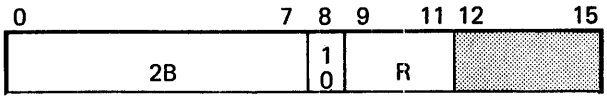
Converts the 120-byte packed decimal field identified by TAG(120,4) to unpacked decimal, and stores them at the field identified by HERE(239,2).

DATA TRANSFER INSTRUCTIONS

Mnemonic Code	Name
CLDR	Condition Register Load
CSTR	Condition Register Store
INVD	Inverse Move Direct
INVI	Inverse Move Immediate
INVM	Inverse Move Memory - Memory
INV	Inverse Move Memory - Register
INVR	Inverse Move Register - Register
LODB	Load Byte ●
LODD	Load Direct
LODI	Load Immediate
LOD	Load Memory - Register
LODT	Load Two-Word
MOVB	Move Byte ●
MOVX	Move Characters ●
MOVL	Move Long ●
MOVM	Move Memory - Memory
MOVR	Move Register - Register
PSTR	Program Address Store
STOB	Store Byte ●
STO	Store Memory - Register
STOT	Store Two-Word

Condition Register Load

CLDR @R



FUNCTION: Transfers the contents of a one-word field in a register or in memory to the Condition register. (Note: If any of the bits 12-15 in the field being transferred are on, false status conditions (such as a bounds error) may be transmitted to the executive program when the Condition register is loaded. This does not apply if the field being transferred was originally generated by a CSTR instruction.)

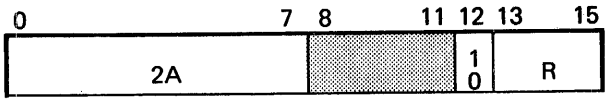
OPERAND: A one-word value in the general register specified by R or in memory if indirect addressing is used, bit 8=1.

EXAMPLE: CLDR @5

A 16-bit field located at the address specified in register 5 is transferred to the Condition register.

Condition Register Store

CSTR @R



FUNCTION: Transfers the contents of the Condition register to a one-word field in a register or in memory.

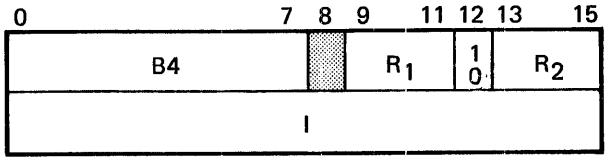
OPERAND: A one-word value in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

EXAMPLE: CSTR @6

The contents of the Condition register are transferred to the location specified in register 6.

Inverse Move Direct

INVD I(R_1),@R_2



FUNCTION: Transfers the one's complement of a one-word immediate value* to a one-word field in a general register or in memory.

OPERAND 1: A 16-bit immediate value in bits 16-31 of the instruction; the value may range from 0-65,535.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

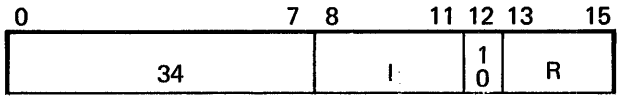
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: INVD 60150(4),@7

The one's complement of the value formed by adding 60,150 to the contents of register 4 is transferred to the location specified in register 7.

Inverse Move Immediate

INVI I,@R



FUNCTION: Transfers the one's complement of a 4-bit immediate value to bits 12-15 of a one-word field in a general register or in memory. Bits 0-11 of the one-word field are always set to ones.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction; the value may range from 0-15.

OPERAND 2: A one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

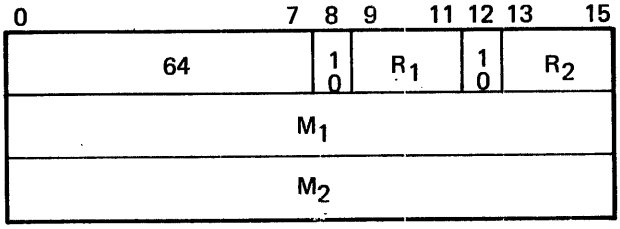
RESULTS: The resulting value resides in bits 12-15 of operand 2.

EXAMPLE: INVI 11,@3

The one's complement of 11 is transferred to bits 12-15 of a 16-bit field located at the address specified in register 3. Bits 0-11 in the 16-bit field are turned on; the result field appears as follows: 1111111111110100.

Inverse Move Memory — Memory

INVM @$M_1(R_1)$,@$M_2(R_2)$



FUNCTION: Transfers the one's complement of a one-word field in memory to another one-word field in memory.

OPERAND 1: The field to be transferred; a one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

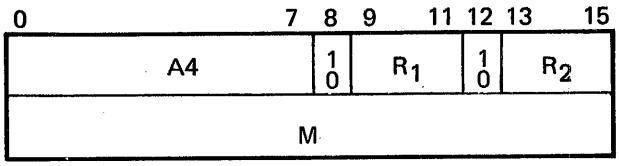
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: INVM @TAG(3),@HERE(2)

The one's complement of a 16-bit field located at an address identified by @TAG(3) is transferred to the field at the address identified by @HERE(2).

Inverse Move Memory — Register

INV @M(R_1),@R_2



FUNCTION: Transfers the one's complement of a one-word field in memory to a one-word field in a general register or in memory.

OPERAND 1: The field to be transferred; a one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The receiving field; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

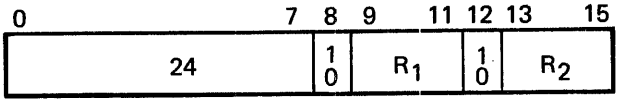
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: INV @TAG(4),@1

The 16-bit field identified by @TAG(4) is transferred in one's complement format to the location specified in register 1.

Inverse Move Register — Register

INVR @R_1,@R_2



FUNCTION: Transfers the one's complement of a one-word field to another one-word field; either field may be in a register or in memory.

OPERAND 1: The value to be transferred; a one-word field located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: A one-word field located in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

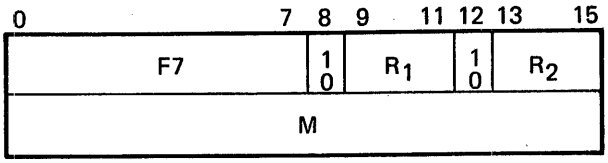
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: INVR 1,2

The contents of register 1 are converted to one's complement format and stored in register 2.

Load Byte ●

LODB @M(R_1),@R_2



FUNCTION: Transfers a one-byte field in memory to bits 8-15 of a one-word field in a general register or to one-byte in memory.

OPERAND 1: The value to be transferred; a one-byte field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1. If the field is in a general register, the byte is placed in bits 8-15 and bits 0-7 are zeroed out.

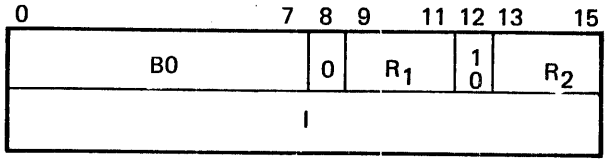
RESULTS: The resulting byte resides at the operand 2 location.

EXAMPLE: LODB @TAG(3),6

@TAG(3) yields the address of a one-byte field in memory which is transferred to bits 8-15 of register 6. (If @6 had been specified, bits 0-7 of a memory word would receive the byte.)

Load Direct

LODD $I(R_1),@R_2$



FUNCTION: Transfers contents of a one-word immediate value* to a one-word field in a general register or in memory.

OPERAND 1: A 16-bit immediate value in bits 16-31 of the instruction; the value may range from 0-65,535.

*Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I value and the index register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

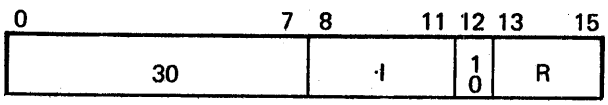
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: LODD 40000(4),@5

The immediate value 40,000 is added to the contents of register 4, and the resulting value is transferred to the address specified in register 5.

Load Immediate

LODI I,@R



FUNCTION: Transfers a 4-bit immediate value to bits 12-15 of a one-word field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the value may range from 0-15.

OPERAND 2: A one-word field located in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

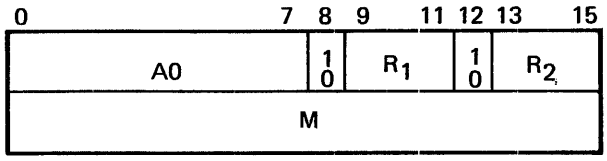
RESULTS: The resulting value resides at the operand 2 location. The value from operand 1 is placed in bits 12-15 of operand 2; bits 0-11 of operand 2 are always zeroed out.

EXAMPLE: LODI 14,@3

The immediate value 14 is written in memory at the location specified in register 3. The result in memory appears as follows: 0000000000001110.

Load Memory — Register

LOD @M(R_1),@R_2



FUNCTION: Transfers the contents of a one-word field in memory to a one-word field in a general register or in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

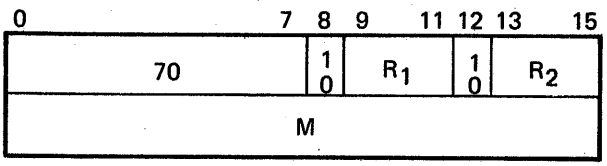
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: LOD @TAG(4),@5

@TAG(4) yields the address in memory of a 16-bit value which is transferred to another location specified by the address in register 5.

Load Two-Word

LODT @M(R_1),@R_2



FUNCTION: Transfers the contents of a two-word field in memory to a two-word field in a general register or in memory.

OPERAND 1: A two-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both. The effective address points to the most significant byte of the field.

OPERAND 2: A two-word field in two general registers (the most significant bit in the register specified by R_2 and the least significant bit in register R_2+1) or in memory if indirect addressing is specified, bit 12=1. The effective address points to the most significant bits of the two-word field.

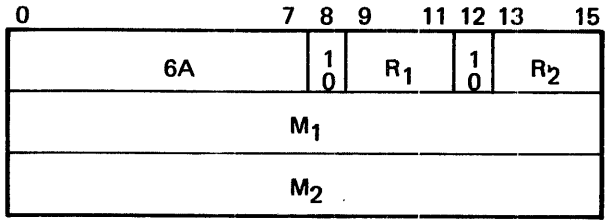
RESULTS: The resulting values reside at the operand 2 location.

EXAMPLE: LODT @HERE(7),5

The 32-bit field identified by @HERE(7) is loaded into registers 5 and 6.

Move Byte ●

MOVB @M_1(R_1),@M_2(R_2)



FUNCTION: Transfers a one-byte field in memory to another one-byte field in memory.

OPERAND 1: The byte transferred; a one-byte field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The receiving field; a one-byte field in memory. Addressing is the same as for operand 1 except use M_2, R_2, and bit 12=1.

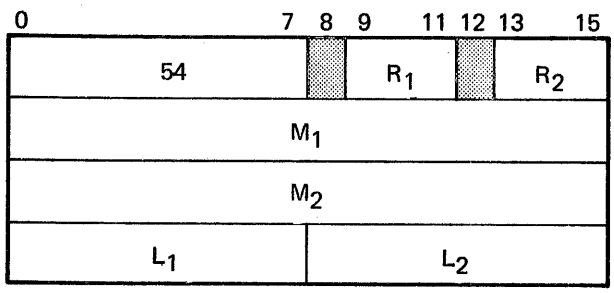
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: MOVB @TAG(1),@HERE(4)

A byte at the address identified by @TAG(1) is moved to the location identified by @HERE(4).

Move Characters •

MOVX $M_1(L_1,R_1),M_2(L_2,R_2)$



FUNCTION: Transfers the contents of a field in memory, one byte at a time, to another field in memory; the field lengths may vary from 0-255 bytes.

OPERAND 1: A field in memory moved one byte at a time. The field length 0-255 bytes, is specified by the L_1 value in the instruction. Addressing options to the base address M_1 include only indexing (R_1).

OPERAND 2: The receiving field in memory. The field length, 0-255 bytes, is specified in the L_2 field of the instruction. Addressing options to the base address M_2 include only indexing (R_2).

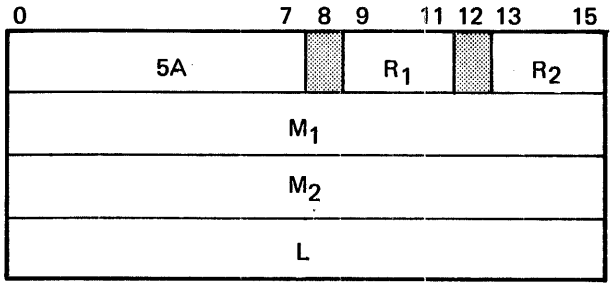
RESULTS: The resulting value resides at the operand 2 location. The following conditions may occur, depending on the values of L_1 and L_2.

- If $L_1 = L_2$, the number of bytes specified by L_1 is transferred.
- If L_1 is less than L_2, the number of bytes specified by L_1 is transferred, then blanks are used to fill operand 2.
- If L_1 is greater than L_2, the number of bytes specified by L_2 is transferred.
- If $L_1 = 0$ and $L_2 \neq 0$, the number of bytes specified by L_2 is filled with blanks.
- If $L_1 = 0$ and $L_2 = 0$, no transfer is executed.

Note: MOVX is a word move if L_1 and L_2 are both even and the beginning addresses are both even.

Move Long ●

MOVL $M_1(L,R_1),M_2(R_2)$



FUNCTION: Moves a field in memory to another location in memory. The length of both fields must be the same; this length can vary from 0-65,535.

OPERAND 1: A field in memory moved one byte at a time. The field length, 0-65,535 bytes, is specified by the L value in the instruction. Addressing options to the base address M_1 include only indexing (R_1). The effective address points to the most significant word of the field.

OPERAND 2: The receiving field in memory. The field length, 0-65,535 bytes, is specified by the L value in the instruction. Addressing options to the base address M_2 include only indexing (R_2). The effective address points to the most significant word of the field.

RESULTS: The resulting field resides at the operand 2 location. The sending field and receiving field may overlap. If L = 0, no move of data is executed.

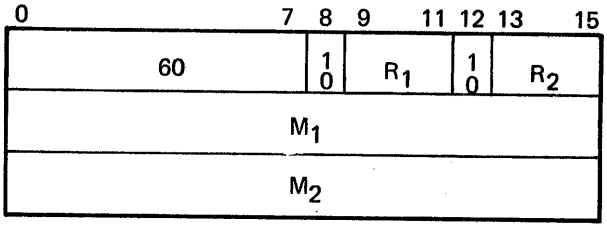
Note: MOVL is a word move if the length, L, is even and the beginning addresses are even.

EXAMPLE: MOVL TAG(2000,6),HERE(7)

A 2,000-byte field identified by TAG(2000,6) is moved to another memory location beginning at the address represented by HERE(7).

Move Memory - Memory

MOVM @M_1(R_1),@M_2(R_2)



FUNCTION: Transfers the contents of a one-word field in memory to another one-word field in memory.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: Same as operand 1 except use M_2, R_2, and bit 12=1.

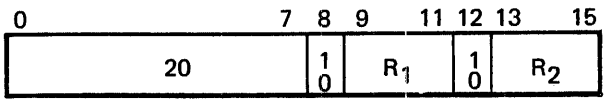
RESULTS: The resulting value resides at the operand 2 location.

EXAMPLE: MOVM @TAG(3),@HERE(6)

A 16-bit field identified by @TAG(3) is moved to the field identified by @HERE(6).

Move Register - Register

MOVR @R_1,@R_2



FUNCTION: Transfers the contents of a one-word field to another one-word field; either field may be in a general register or in memory.

OPERAND 1: A one-word field in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1.

OPERAND 2: Same as operand 1 except use R_2 and bit 12=1.

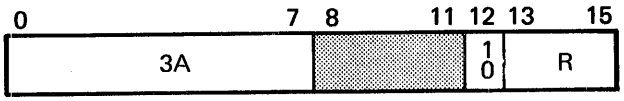
RESULTS: The moved value resides at the operand 2 location.

EXAMPLE: MOVR @6,@4

A 16-bit field at the address specified in register 6 is moved to the address specified in register 4.

Program Address Store

PSTR @R



FUNCTION: Transfers the current program address to a one-word field in a register or in memory.

OPERAND: A one-word field in the general register specified by R or in memory if indirect addressing is used, bit 12=1.

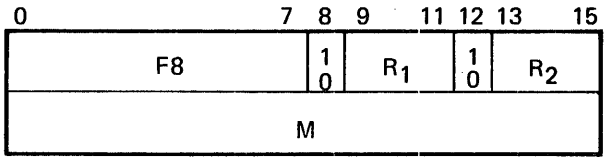
RESULTS: The current program address resides at the operand 1 location.

EXAMPLE: PSTR @5

Transfers the current value of the Program Address register to the address specified in register 5.

Store Byte •

STOB @M(R_1),@R_2



FUNCTION: Transfers the contents of bits 8-15 of a general register or a one-byte field in memory to a one-byte field in memory.

OPERAND 1: The receiving field; it is a one-byte field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The field to be stored; bits 8-15 of the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

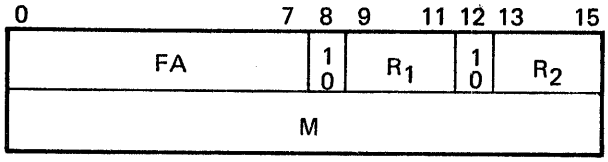
RESULTS: The stored byte resides at the operand 1 location.

EXAMPLE: STOB @TAG(3),4

The contents of bits 8-15 of register 4 are stored at a memory location identified by @TAG(3).

Store Memory - Register

STO @M(R_1),@R_2



FUNCTION: Transfers the contents of a one-word field in a general register or in memory to another one-word field in memory.

OPERAND 1: The receiving field; it is a one-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The field to be stored; it is a one-word field in the general register specified by R_2 or in memory if indirect addressing is used, bit 12=1.

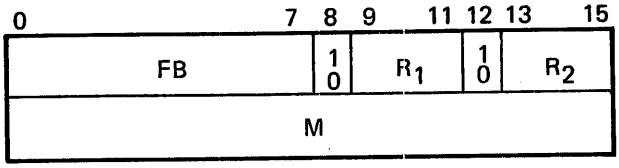
RESULTS: The field is stored in the operand 1 location.

EXAMPLE: STO @TAG(3),4

The contents of register 4 are stored at a memory location identified by @TAG(3).

Store Two-Word

STOT @M(R_1),@R_2



FUNCTION: Transfers the contents of a two-word field in two general registers or in memory to a two-word field in memory.

OPERAND 1: The receiving field; a two-word field in memory. Addressing options to the base address M include indexing (R_1), indirect addressing (bit 8=1), or a combination of both.

OPERAND 2: The field to be stored; a two-word field in two general registers (the most significant word in the register specified by R_2 and the least significant word in the register specified by R_2+1) or in memory if indirect addressing is used, bit 12=1.

RESULTS: The stored values reside at the operand 1 location.

EXAMPLE: STOT @TAG(2),@7

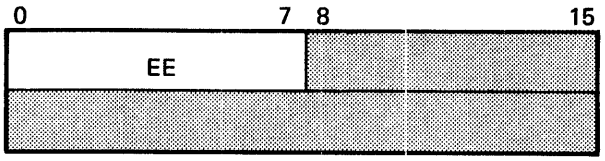
A 32-bit field, the address of which is specified in register 7, is stored in memory at a location identified by @TAG(2).

NO OPERATION

Mnemonic Code	Name
NOP	No Operation

No Operation

NOP



FUNCTION: Performs no operation. This instruction has no operands.

EXAMPLE: NOP

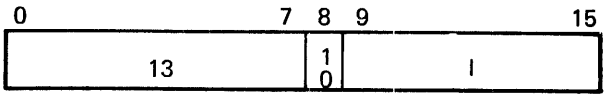
This instruction occupies four bytes in the program.

SERVICE REQUEST

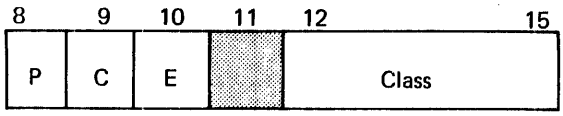
Mnemonic Code	Name
SR	Service Request

Service Request

SR @I



FUNCTION: Provides an information byte called the request index (I) to be interpreted by the operating system (software). The request index byte has the following format:



P bit Indicates location of parameter string.

0 means immediately following service request.
1 means at address contained in register 6.

C bit Indicates when control is to be returned to requesting program.

0 means after service request is completed.
1 means after service request is recognized by the control program.

E bit Indicates if the requesting program will process exception completion of the request.

0 means requesting program will not process exception completion.
1 means requesting program will process exception completion.

CLASS Denotes major class in which the service request falls. Service requests fall into the following seven major classes.

CLASS 0 Debugging service request.
CLASS 1 Restricted service request.
CLASS 2 Control program service request.
CLASS 3 Block I/O service request.
CLASS 4 Physical I/O service request.
CLASS 5 Supervisor service request.
CLASS 6 Telecommunications service request.

NOTE: This material on the Service Request instruction does not cover all the necessary details; parameter strings, for instance, must be preceded by a four-byte field that further defines the operation. For more information on service requests, see the Operating System specifications.

OPERAND: I is the request index byte, which is defined by the operating system.

RESULTS: Execution of a service request instruction causes the following:

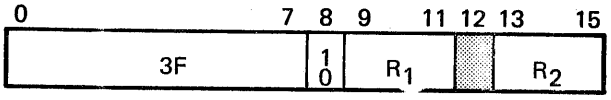
- The Service Request bit (bit 13) of the executing processor's Condition register is set.
- The Busy and Active bits of Processor 4 are both set.
- The Busy and Active bits of the executing processor are both reset.

SHIFT INSTRUCTIONS

Mnemonic Code	Name
ARDR	Arithmetic Right Double Shift - by Register
ARDI	Arithmetic Right Double Shift - Immediate
ARSR	Arithmetic Right Single Shift - by Register
ARSI	Arithmetic Right Single Shift - Immediate
LLDR	Logical Left Double Shift - by Register
LLDI	Logical Left Double Shift - Immediate
LLSR	Logical Left Single Shift - by Register
LLSI	Logical Left Single Shift - Immediate
LRDR	Logical Right Double Shift - by Register
LRDI	Logical Right Double Shift - Immediate
LRSR	Logical Right Single Shift - by Register
LRSI	Logical Right Single Shift - Immediate
RLDR	Rotating Left Double Shift - by Register
RLDI	Rotating Left Double Shift - Immediate
RLSR	Rotating Left Single Shift - by Register
RLSI	Rotating Left Single Shift - Immediate
SHFK	Shift Packed Decimal ●

Arithmetic Right Double Shift — by Register

ARDR @R_1,R_2



FUNCTION: Performs an arithmetic right shift of a two-word field in two general registers. The sign (bit 0) of the field is extended. The shift count is a 4-bit field in a general register or in memory.

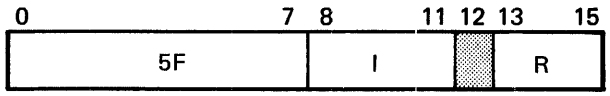
OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R_2 and the least significant bits in the register specified by R_2+1. The data is shifted to the right the number of positions specified by operand 1. The sign of the data is extended. Any data shifted out of the higher register is lost. (Note: If register 7 is specified as operand 2, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: ARDR @3,5

Shifts the data in registers 5 and 6 to the right; register 3 contains the address of a memory field that holds the shift count. The sign is extended, and data shifted out of register 5 is lost.

Arithmetic Right Double Shift — Immediate



ARDI I,R

FUNCTION: Performs an arithmetic right shift of a two-word field in two general registers. The sign (bit 0) of the field is extended. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

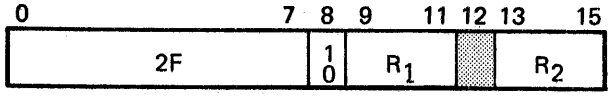
OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R and the least significant bits in the register specified by R+1. The data is shifted to the right the number of positions specified by operand 1. The sign of the data is extended. Any data shifted out of the higher register is lost. (Note: If register 7 is specified for operand 2, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: ARDI 13,5

Shifts the data in registers 5 and 6 to the right 13 bit positions. The sign is extended from bits 0-13 in register 5; data shifted out of register 6 is lost.

Arithmetic Right Single Shift — by Register

ARSR @R_1,R_2



FUNCTION: Performs an arithmetic right shift of a one-word field in a general register. The sign (bit 0) of the field is extended. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

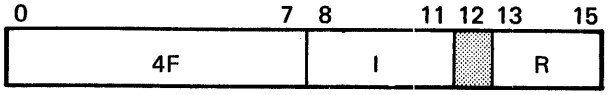
OPERAND 2: A one-word field in the general register specified by R_2. The data is shifted to the right the number of positions specified by operand 1. The sign of the data is extended. Any bits shifted out of the register are lost.

EXAMPLE: ARSR @5,4

Shifts the data in register 4 to the right; register 5 contains the address of the shift count. The sign is extended, and data shifted out of register 4 is lost.

Arithmetic Right Single Shift — Immediate

ARSI I,R



FUNCTION: Performs an arithmetic right shift of a field in a general register. The sign (bit 0) of the field is extended. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

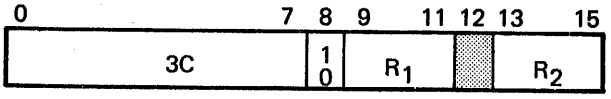
OPERAND 2: A one-word field in the general register specified by R. The data is shifted to the right the number of positions specified by operand 1. The sign of the data is extended. Any data shifted out of the register is lost.

EXAMPLE: ARSI 11,2

Shifts the data in register 2 to the right 11 bit positions. The sign is extended from bit 0-11; any data shifted out of register 2 is lost.

Logical Left Double Shift — by Register

LLDR @R_1,R_2



FUNCTION: Performs a left shift (zero fill from right) of a two-word field in two general registers. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

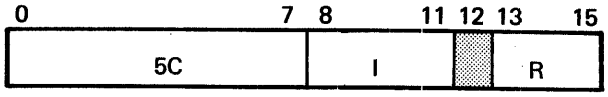
OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R_2 and the least significant bits in the register specified by R_2+1. The data is shifted to the left the number of positions specified by operand 1. Any data shifted out of the lower register is lost. (Note: If register 7 is specified for operand 2, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: LLDR @2,3

Shifts the data in registers 3 and 4 to the left; the address of the shift count is in register 2. Data shifted out of register 3 is lost.

Logical Left Double Shift — Immediate

LLDI I,R



FUNCTION: Performs a left shift (zero fill from right) of a two-word field in two general registers. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

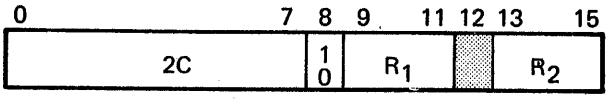
OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R and the least significant bits in the register specified by R+1. The data is shifted to the left the number of positions specified by operand 1. Any data shifted out of the lower register is lost. (Note: If register 7 is specified for operand 2, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: LLDI 10,0

Shifts the data in registers 0 and 1 to the left 10 bit positions. Data shifted out of register 0 is lost.

Logical Left Single Shift — by Register

LLSR @R_1,R_2



FUNCTION: Performs a left shift (zero fill from right) of a one-word field in a general register. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

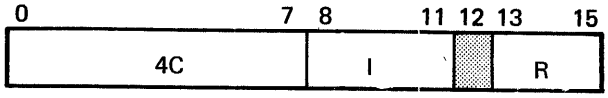
OPERAND 2: A one-word field in the general register specified by R_2. The data is shifted to the left the number of positions specified by operand 1. Any data shifted out of the register is lost.

EXAMPLE: LLSR @2,7

Shifts the data in register 7 to the left; the address of the shift count is in register 2. Data shifted out of register 7 is lost.

Logical Left Single Shift — Immediate

LLSI I,R



FUNCTION: Performs a left shift (zero fill from right) of a one-word field in a general register. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

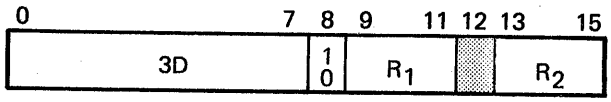
OPERAND 2: A one-word field in the general register specified by R. The data is shifted to the left the number of positions specified by operand 1. Any data shifted out of the register is lost.

EXAMPLE: LLSI 7,5

Shifts the data in register 5 to the left 7 bit positions. Data shifted out of register 5 is lost.

Logical Right Double Shift — by Register

LRDR @R_1,R_2



FUNCTION: Performs a right shift (zero fill from left) of a two-word field in two general registers. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

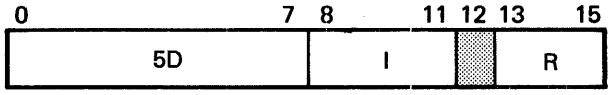
OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R_2 and the least significant bits in the register specified by R_2+1. The data is shifted to the right the number of positions specified by operand 1. The sign of the data is not extended. Any data shifted out of the higher register is lost. (Note: If register 7 is specified as operand 1, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: LRDR @7,4

Shifts the data in registers 4 and 5 to the right. The shift count is at the memory address specified in register 7. Data shifted out of register 5 is lost.

Logical Right Double Shift — Immediate

LRDI I,R



FUNCTION: Performs a right shift (zero fill from left) of a two-word field in two general registers. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

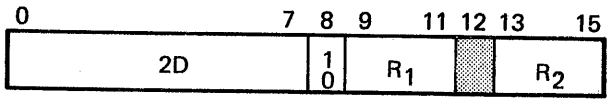
OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R and the least significant bits in the register specified by R+1. The data is shifted to the right the number of positions specified by operand 1. Any data shifted out of the higher register is lost. (Note: If register 7 is specified for operand 2, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: LRDI 10,0

Shifts the data in registers 0 and 1 to the right 10 bit positions. Data shifted out of register 1 is lost.

Logical Right Single Shift — by Register

LRSR @R_1,R_2



FUNCTION: Performs a right shift (zero fill from left) of a one-word field in a general register. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

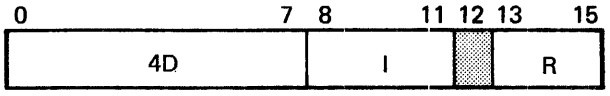
OPERAND 2: A one-word field in the general register specified by R_2. The data is shifted to the right the number of positions specified by operand 1. The sign is not extended. Any data shifted out of the register is lost.

EXAMPLE: LRSR @3,5

Shifts the data in register 5 to the right; the address of the shift count is in register 3. Data shifted out of register 5 is lost.

Logical Right Single Shift — Immediate

LRSI I,R



FUNCTION: Performs a right shift (zero fill from left) of a one-word field in a value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

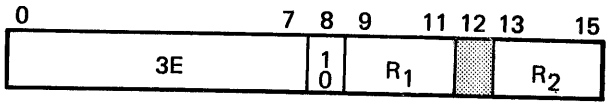
OPERAND 2: A one-word field in the general register specified by R. The data is shifted to the right the number of positions specified by operand 1. The sign of the data is not extended. Any data shifted out of the register is lost.

EXAMPLE: LRSI 7,5

Shifts the data in register 5 to the right 7 bit positions. Data shifted out of the register is lost.

Rotating Left Double Shift — by Register

RLDR @R_1,R_2



FUNCTION: Performs a rotating left shift of a two-word field in two general registers. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

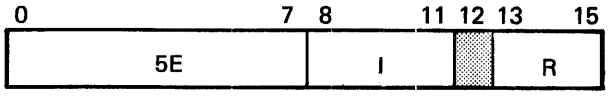
OPERAND 2: A two-word field in two general registers: the most significant bits in the register specified by R_2 and the least significant bits in the register specified by R_2+1. The data is shifted to the left the number of positions specified by operand 1. Each bit shifted out of the left end of the lower register is brought back in the right end of the higher register. Bits shifted out of the register are not lost. (Note: If register 7 is specified as operand 1, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: RLDR @5,2

Shifts the data in registers 2 and 3 to the left; the shift count is at the memory location specified in register 5. Data shifted out the left end of register 2 is brought back in the right end of register 3.

Rotating Left Double Shift — Immediate

RLDI I,R



FUNCTION: Performs a rotating left shift of a two-word field in two general registers. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

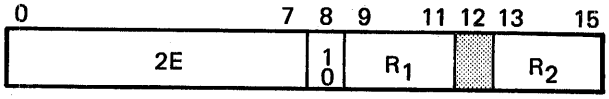
OPERAND 2: A two-word field located in two general registers: the most significant bits in the register specified by R and the least significant bits in the register specified by R+1. The data is shifted to the left the number of positions specified by operand 1. Each bit shifted out of the left end of the lower register is brought back in the right end of the higher register. Bits shifted out of the lower register are not lost. (Note: If register 7 is specified as operand 1, the operand is assumed to be in registers 7 and 0 with the most significant bits in 7.)

EXAMPLE: RLDI 6,1

Shifts the data in registers 1 and 2 to the left six bit positions. Each bit shifted out of the left end of register 1 is brought back in the right end of register 2.

Rotating Left Single Shift — by Register

RLSR @R_1,R_2



FUNCTION: Performs a rotating left shift of a one-word field in a general register. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: A 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used, bit 8=1. The shift count may range from 0-15.

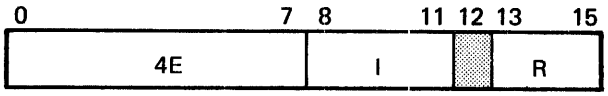
OPERAND 2: A one-word field in the general register specified by R_2. The data is shifted to the left the number of positions specified by operand 1. Each bit shifted out the left end of the register is brought back in the right end. Bits shifted out of the register are not lost.

EXAMPLE: RLSR @1,2

Shifts the data in register 2 to the left. Data shifted out the left end of the register is brought back in the right end of the register. The shift count is at the memory address in register 1.

Rotating Left Single Shift — Immediate

RLSI I,R



FUNCTION: Performs a rotating left shift of a one-word field in a general register. The shift count is a 4-bit immediate value.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the shift count may range from 0-15.

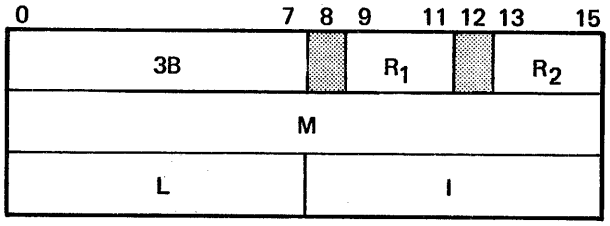
OPERAND 2: A one-word field located in the general register specified by R. The data is shifted to the left the number of bit positions specified by operand 1. Bits shifted out of the register are not lost; each bit shifted out of the left end of the register is brought back in the right end of the register.

EXAMPLE: RLSI 5,2

Shifts the data in register 2 to the left 5 bit positions. Data shifted out the left end of the register is brought back in the right end; no bits are lost.

Shift Packed Decimal

SHFK M(L,R_1),I(R_2)



FUNCTION: Shifts a packed decimal field in memory a specified number of digits. The length of the packed decimal field can be defined for each SHFK instruction.

OPERAND 1: The packed decimal field in memory. The length of this field (number of decimal digits) is in L; this value may range from 0-255. Addressing options to the base address M include only indexing (R_1).

OPERAND 2: The shift count is the I value in the instruction, this is a signed value from -128 to +127. If a register is specified in R_2, the contents of the register are added to the I value to form the shift count.

CONDITION REGISTER:

- Bit 0 is set if any significance is shifted out in a left shift; otherwise bit 0 is cleared.
- Bit 1 is set if the result is positive; otherwise bit 1 is cleared.
- Bit 2 is set if the result is negative; otherwise bit 2 is cleared.
- Bit 3 is set if all significance is shifted out, resulting in zero, or if L=0; otherwise bit 3 is cleared.

EXAMPLE: SHFK TAG(90,2),14(5)

The contents of location TAG are modified by the contents of register 2 to find a 90-byte packed decimal field, which is shifted 14 bit positions (modified by register 5) to the right.

SECTION V

SYSTEM CONTROL PANEL

TABLE OF CONTENTS

SYSTEM CONTROL PANEL

SECTION V
SYSTEM CONTROL PANEL

INTRODUCTION

This section describes the function of controls and indicators associated with the operator group, programmer group, system activity display group, and communication activity display group. Description of controls, indicators, and procedures associated with the maintenance group portion of the panel are specifically excluded from this section because their use is restricted to properly-qualified maintenance personnel only.

CONTROLS AND INDICATORS

Controls and indicators on the System Control Panel are divided into five groups:

- Operator Group
- Programmer Group
- Maintenance Group
- System Activity Display Group
- Communications Activity Display Group

The following paragraphs provide a functional description for each control and indicator on the panel. The descriptions are arranged by panel group, starting with the bottom right control in each group and proceeding in a leftward and upward manner.

Operator Group

- EMERGENCY PULL Knob

  When pulled, instantly removes all power from system (does not go through normal power-down sequence which is the case when using POWER OFF pushbutton).

  NOTE: The EMERGENCY PULL knob is not meant for normal "power-off" sequencing. Whenever this switch is used to remove power from the system, power cannot be reapplied until a mechanical inter-lock within the cabinet is released. (This is a maintenance activity.) Thus the EMERGENCY PULL knob is intended to be used only in emergency situations (circumstances involving a safety hazard).

- POWER OFF Pushbutton/Indicator

  Turns system power off when POWER MODE switch in the Maintenance Group is in LOCAL. This switch assures proper power-down sequencing. (POWER OFF has no effect while POWER MODE is in REMOTE.)

  NOTE: POWER OFF will be illuminated unless no primary power is available, the Main Disconnect switch is off, the EMERGENCY PULL knob has been pulled, or power is on (that period between the completion of a power-up sequence and the initialization of a power-down sequence).

- POWER ON Pushbutton/Indicator

  Turns system power on when the POWER MODE switch in the Maintenance Group is in LOCAL. This switch assures proper power-up sequencing. (POWER ON has no effect while POWER MODE is in REMOTE.)

  NOTE: Upon completion of the power-up sequence, a Reset/Load sequence is automatically initiated. Moreover, at completion of the Reset/Load sequence, an Autoload sequence is automatically initiated provided the Maintenance Mode has not been selected. Further detail is provided under Reset/Load, Autoload, and Maintenance Mode in this section.

- AUTOLOAD Pushbutton

  Causes Main Storage to be loaded, starting at a location determined by the microprogram subroutine with data obtained either from disc drive zero (when AUTOLOAD SELECT switch is in PRIMARY) or from a card reader (when AUTOLOAD SELECT switch is in ALTERNATE).

- AUTOLOAD SELECT Switch

  Down position (PRIMARY) selects disc as input device. Up position (ALTERNATE) selects card reader as input device.

NOTE: The PRIMARY position of this switch causes the first Autoload microprogram loader instruction to come from Control Storage address 011_{16}. The ALTERNATE position of this switch causes the first autoload microprogram loader instruction to come from Control Storage address 011_{16}.

This switch also determines the device, as defined above, that Control Storage is loaded from during a Reset/Load operation.

- SPEAKER VOLUME Control

  Adjusts volume of the speaker contained in the System Control Panel enclosure. This speaker is driven by the circuits associated with bit positions 13, 14, and 15 of the Console Data Register Display indicators.

  NOTE: The relative loudness of these bits on the speaker are: bit 14 will be twice as loud as bit 13 and bit 15 twice as loud as bit 14.

- I/O FAULT Pushbutton/Indicator

  I/O FAULT will illuminate if any of the following conditions occur.

  1. Channel 1 Transmission Parity Error
  2. Channel 2 Transmission Parity Error
  3. Channel 1 Control Check Error
  4. Channel 2 Control Check Error
  5. Burst Check Error (during a Reset/Load operation from disc).
  6. Failure of disc heads to retract during power-down sequence

  Pressing I/O FAULT extinguishes the indicator. (Refer to individual I/O fault indications in the System Activity Display Group, further in this section.)

- PROC FAULT Pushbutton/Indicator

  PROC FAULT will illuminate if any of the following conditions occur.

  1. Control Storage Parity Error
  2. Main Storage Parity Error
  3. DC Voltage Fault
  4. Over-Temperature condition

Pressing PROC FAULT extinguishes the indicator.

- ALARM (located behind System Control Panel)

  Furnishes an audible signal when the LAMP TEST pushbutton is depressed or when any of the following conditions exist.

  1. Blower failure within the computer
  2. D.C. voltage fault
  3. Failure of disc heads to retract during a power-down sequence

  NOTE: When blower failure or D.C. fault condition exists for approximately 60 seconds, the power-down sequence is automatically initiated.

  If the heads fail to retract from a disc during the power-down sequence, DC voltages will be removed within the computer. The power-down sequence will stop at that point until the problem is corrected.

- ALARM DISABLE Pushbutton/Indicator

  Pressing this pushbutton, if the audible alarm is on, causes the ALARM to stop and the ALARM DISABLE to illuminate. When the alarm condition is corrected, the ALARM DISABLE will extinguish.

- LAMP TEST Pushbutton

  Pressing this switch causes all indicators to illuminate and the alarm to sound. Releasing the switch returns them to their prior state.

- RESET/LOAD Pushbutton

  Causes data to be read from either cards or disc and transferred to either Control Storage and first-level decode address table or Main Storage, depending on whether the CONSOLE MODE SELECT selector is set to CS-WR or MS-WR and if the Maintenance Mode has been selected. Selection of cards or disc as input medium is determined by position of AUTOLOAD SELECT switch.

  Upon completion of a Reset/Load operation from disc, an Autoload operation will automatically be initiated providing the Maintenance Mode has not been selected.

Programmer Group

NOTE: Controls within this group are conditioned by the PROGRAM MODE pushbutton/indicator except where otherwise designated.

- CONSOLE MODE SELECT Selector

  Selects basic mode of operation for System Control Panel:

  RO-RD - Register Option read

  RO-WR - Register Option write

  RF-WR - Register File write

  RF-RD - Register File read

  OFF - select switch disabled

  MS-RD - Main Storage read

  MS-WR - Main Storage write

  CS-RD - Control Storage and first-level decode address table read or scan (enabled in Maintenance Mode only)

  CS-WR - Control Storage write (enabled in Maintenance Mode only except during a Reset/Load operation)

- CONSOLE RUN Pushbutton

  Initiates the function selected on the CONSOLE MODE SELECT selector in a manner determined by the CONSOLE CONTROL SELECT and Console MS switches.

- CONSOLE CONTROL SELECT Switch

  Three-position switch governing way in which selected console control operation is executed:

  STOP/STEP - stop and step

  NORMAL - run continuously

  BREAKPOINT - run as far as breakpoint (applies to CS-RD, CS-WR, MS-RD, and MS-WR only)

- Console Main Storage Switch

  NOTE: This switch has no effect unless the Relocation and Protection feature is installed.

  When Console Mode Selector is set to MS-RD or MS-WR this switch determines whether the contents of the S-Register is interpreted as a System or Physical Main Storage Address. When in the RELOCATE (up) position, the contents of the S-Register are interpreted as a System Main Storage address and is converted by the relocation mechanism into a Physical Main Storage Address. When in the OFF (down) position, the contents of the S-Register is directly interpreted as a Physical Main Storage Address and will bypass the relocation mechanism.

- SYSTEM RESET Pushbutton

  The SYSTEM RESET pushbutton clears the following registers:

  1. EXTENDED REGISTER FILE

     Group I: Pμ of all processor states

     Group II: Busy/Active, Tie-Breaker, Control, Privileged, Boundary-Crossing, Control Storage Scan, Console Address, and Console Data.

  2. SHARED RESOURCE REGISTERS

     Aμ, Bμ, D, Sμ, Fμ-1, Fμ-2, and Forced Carry Register

- PROGRAM MODE Pushbutton/Indicator

  Pressing this switch enables those switches located in the programmer group area of the panel unless otherwise indicated. The pushbutton is illuminated when Program Mode is selected.

- CONSOLE DATA REGISTER SELECT Selector

  Selects one of eleven registers to be displayed by the CONSOLE DATA REGISTER DISPLAY indicators. (Does not affect the pushbutton function.)

  NOTE: Only the DATA and B/A positions are enabled when Program Mode is selected. All other positions of this switch require Maintenance Mode to be effective.

  Fμ2 - Micro-command Function register, rank 2

  Fμ1 - Micro-command Function register, rank 1

  RTC - Real-Time Clock register

  CSS - Control Storage Scan register

  B/A - Busy/Active register

DATA - Console data register

D - Main Storage Data register

Aμ - ALU feeder register Aμ

Bμ - ALU feeder register Bμ

SUM - output of ALU (sum of Aμ and Bμ)

BC - Boundary-Crossing register

- CLEAR DATA Pushbutton

  Clears contents of Console Data Register.

- CONSOLE DATA REGISTER DISPLAY Pushbutton/Indicators

  Twenty pushbutton/indicators horizontally located as 5 groups of 4 bits each. These groups function as follows:

  1. Pushbutton/indicators: X0 - X3

     These pushbutton/indicators are not functional unless the Relocation and Protection Feature is present in the 7200/7300 computer.

     In the presence of the Relocation and Protection feature, pressing these pushbuttons will cause corresponding bits to be set in the Segment Tag portion of the Console Data Register. The indicators will be on for corresponding bit positions that are set and off for corresponding bit positions that are clear in the Segment Tag portions of the Console Data Register.

  2. Pushbutton/indicators: 00 - 15

     Pressing these pushbuttons will cause corresponding bits to be set in the Console Data Register only. However, the indicators will be on for corresponding bit positions that are set and off for corresponding bit positions that are clear at Fμ2, Fμ1, RTC, CSS, B/A, DATA, D, Aμ, Bμ, SUN or BC outputs as determined by the CONSOLE DATA REGISTER SELECT Selector.

     The digital inputs to the Console Data Register Display lamp drivers in bit positions 13, 14, and 15 are also used as inputs to the panel speaker drivers.

- CONSOLE ADDRESS REGISTER SELECT Selector

  Selects one of four registers to be displayed by the CONSOLE ADDRESS REGISTER DISPLAY indicators. (Does not affect the Console Address Register pushbutton function.)

NOTE: Only the S and ADDRESS positions are enabled when Program Mode is selected. The remaining positions of this switch require Maintenance Mode to be effective.

S - Main Storage address register

Sμ - Control Storage address register

ADDRESS - Console Address register

PE - Main Storage Parity Error address register

- CLEAR ADDRESS Pushbutton

  Clears contents of Console Address Register.

- CONSOLE ADDRESS REGISTER DISPLAY Pushbutton/Indicators

  Twenty pushbutton/indicators horizontally located as 5 groups of 4 bits each. These groups function as follows:

  1. Pushbutton/indicators: X0 - X3

     These pushbutton/indicators are not functional unless the Relocation and Protection Feature is present in the 7200/7300 computer.

     In the presence of the Relocation and Protection Feature, pressing these pushbuttons will cause corresponding bits to be set in the Segment Tag Portion of the Console Address Register. The indicators will be on for corresponding bit positions that are set and off for corresponding bit positions that are clear in the Segment Tag portions of the S, Console Address, and PE Registers as determined by the CONSOLE ADDRESS REGISTER SELECT Selector.

  2. Pushbutton/indicator: 00 - 15

     Pressing these pushbuttons will cause corresponding bits to be set in the Console Address Register only. However, the indicators will be on for corresponding bit positions that are set and off for corresponding bit positions that are clear in the Sμ, S, Console Address and PE Registers as determined by the CONSOLE ADDRESS REGISTER SELECT Selector.

- BREAKPOINT MODE SELECT Switches:

  1. WRITE DATA Switch

     When up (on), causes breakpoint stop at end of each storage reference cycle in which data was written at breakpoint address.

  2. READ DATA Switch

     When up (on), causes breakpoint stop at end of each storage reference cycle in which data was read at breakpoint address.

3. READ INSTR Switch

   When up (on), causes breakpoint stop immediately after the machine language instruction is read at breakpoint address.

4. SYSTEM/PHYSICAL Switch

   NOTE: This switch has no effect unless the Relocation and Protection Feature is installed.

   This switch determines whether the System or Physical Main Storage Address is compared with the Breakpoint Address SELECT. Additionally, this switch determines whether the System or Physical Main Storage Addresses are sent to the Console Address Register Display indicators when the Console Address Register Select switch is set to the S position. When in the SYSTEM (up) position, System Main Storage Addresses are selected as described above. When in the PHYSICAL (down) position, Physical Main Storage Addresses are selected as described above.

- BREAKPOINT ADDRESS SELECT Selectors

  Five selectors which provide a hexadecimal stop address for processor state(s) operating in the breakpoint mode. Also applies to console mode, MS-RD, MS-WR, and CS-WR selections.

- PROCESSOR SELECT Selector

  Selects one of the eight processor states to execute in the mode selected by the corresponding Processor Control Select switches.

- PROCESSOR RUN Pushbutton

  Starts the processor state selected by the PROCESSOR SELECT selector.

- PROCESSOR CONTROL SELECT Switches

  Eight three-position switches which place individual processor states in one of three modes:

  1. STOP/STEP - Stop and step selected processor state.

  2. NORMAL - Allows selected processor state to run continuously.

  3. BREAKPOINT - Allows selected processor state to run until a breakpoint-comparison equality occurs.

System Activity Display Group

- PROCESSOR STATE Indicators

  Dynamically indicate which processor states are executing major cycles.

- Status Indicators

  Twelve indicators that illuminate particular status conditions:

  MS PARITY BYTE 0 - Displays state of parity bit of upper byte (bits 0 through 7) of word currently being read out of MS. (Not enabled if ECC is present.)

  MS PARITY BYTE 1 - Displays state of parity bit of lower byte (bits 8 through 15) of word currently being read out of MS. (Not enabled if ECC is present.)

  MS PARITY ERROR - Displays state of MS Parity Error flip-flop. Indicator is on if flip-flop is set and off if flip-flop is cleared.

  CS PARITY ERROR - Indicates a parity error in CS or first-level decode address table.

  D.C. FAULT - Indicates that one or more DC power supply in system is not within allowable output range. Remains on until condition is corrected.

  OVER TEMP. - Indicates a blower failure condition within cabinet.

  HEADS EXTENDED - Indicates that heads in one or more disc files fail to retract during the power-down sequence.

  BURST CHECK - Indicates detection of a burst check error during a Reset/Load sequence from the disc file.

  CHANNEL 1 DATA CHECK - Indicates state of Channel 1 Transmission flip-flop.

  CHANNEL 1 CNTRL. CHECK - Indicates state of Channel 1 Control Check flip-flop.

  CHANNEL 2 DATA CHECK - Indicates state of Channel 2 Transmission flip-flop.

  CHANNEL 2 CNTRL. CHECK - Indicates state of Channel 2 Control Check flip-flop.

Maintenance Group

Controls and indicators of the maintenance group are not described in this manual because their use is restricted to maintenance personnel only.

Communications Line Display Group

These indicators show the adapter/modem status for the 15 communications channels and the integrated communications adapter as follows.

- RECEIVED DATA (BB) - The ON condition indicates that the line is in the spacing condition (i.e., a binary zero). The OFF condition indicates that the line is in the marking condition (i.e., a binary one).
- TRANSMITTED DATA (BA) - The ON condition indicates that the line is in the spacing condition (i.e., a binary zero). The OFF condition indicates that the line is in the marking condition (i.e., a binary one).
- CLEAR TO SEND (CB) - The ON condition together with the ON condition on circuits CA, CC, and CD, indicates that the channel is in a transmit condition.
- RECEIVED LINE SIGNAL DETECTOR (CF) - The ON condition indicates that the modem is receiving a signal which meets its suitability criteria for demodulation, and in the case of half-duplex channels, that the line adapter is in the receive mode.
- SECONDARY RECEIVED LINE SIGNAL DETECTOR (SCF) - The ON condition indicates the proper reception (where applicable) of the secondary channel line signal. It is used to indicate the circuit assurance status and to signal the interrupt condition.
- DATA SET READY (CC) - The ON condition on this circuit is presented to indicate that the modem is connected to a communication channel and has completed the transmission of the answer tone.
- OFF HOOK (OH) - When answering a call, this signal ultimately completes a d.c. path to the serving Central Office, tripping the incoming ringing signal. When originating a call, this lead is operated to obtain dial tone and then pulsed to generate the desired dial digits.
- RING INDICATOR (CE) - the ON condition indicates that a ringing signal is being received on the communication channel.

NOTE: Since under normal operation the communications handler will answer a call on the leading edge of the ring signal, the ON condition on this indicator implies either a malfunction or that the channel is not enabled.

- ENABLE (EN) - The ON condition indicates that the line adapter is enabled and is therefore not in the master clear or loop test mode.

OPERATING PROCEDURES

The following paragraphs contain procedures which may be executed from the System Control Panel. These procedures facilitate loading Control or Main Storage from either a disc or card reader, reading from or writing into Main Storage or registers within Register Files or Register Options and executing programs in Maintenance Mode.

- *SWITCHING POWER ON*

  To turn the computer on, ensure that the LOCAL/REMOTE switch is in the LOCAL position, then simply depress the POWER ON pushbutton/indicator. Upon completion of the power-up sequence, the associated indicator will light.

  The internal power-up switching sequence for the computer and disc drives is performed by the hardware.

  To turn the computer off, depress the POWER OFF pushbutton/ indicator. The associated indicator will light.

- *FUNDAMENTAL MODES*

  Apart from the Reset/Load and Autoload facilities, which are always available to the operator, the System Control Panel proper (activated by the PROGRAM MODE and MAINTENANCE MODE) has two fundamental modes of operation. These two modes, Processor Control and Console Control, are not mutually exclusive from the hardware point of view, but should be clearly distinguished and kept separate in operating practice. This separation is necessary since the Console Mode can directly alter the contents of storage and registers and in this way could completely disrupt Processor Mode operations.

  The Processor Control Mode enables the operator, in connection with programmed operation, to directly control execution of instructions by all eight processor states. Thus, individual processors states may be switched on and off or may be made to run one instruction at a time (STEP mode) etc. Except for the internal effects of the programs themselves, the Processor Mode does not enable the contents of storage to be altered.

The Console Control Mode, on the other hand, does not involve any actual execution or instructions, but allows any individual cell of Main or Control Storage to be either displayed or altered. The console mode also allows any of the hardware registers to be displayed or altered. The console (System Control Panel) is allocated major cycles just as though it were a ninth processor, except that it cannot execute machine instructions.

- *BREAKPOINT FACILITY*

The Breakpoint facility provides a way of terminating Processor Mode or Console Mode operations at a specific point in either Main Storage or Control Storage (including the first-level decode address table). This facility may be invoked if the selected processor is started either from the panel (Processor RUN) or by internal operations, or if the computer is already executing instructions. The Breakpoint operation is initiated by setting the Processor Control Select switch to BREAKPOINT. The computer then proceeds until the storage location on the BREAKPOINT ADDRESS SELECT is used for one of three purposes as selected on the three Breakpoint Mode switches. Read Instr will stop the selected processor when it reads an instruction from the Breakpoint address; Read Data will stop it when it reads an operand from the Breakpoint address and Write Data will stop it when it stores an operand at the Breakpoint address.

With respect to the Breakpoint facility, a Read Data reference is only the loading of the first two bytes of an instruction. Thus, the reading of the V_1, V_2, L_1, and L_2 portions of 4-, 6-, and 8-byte instructions are treated as operand references for Breakpoint purposes.

NOTE: Word addressing will not result in a Breakpoint stop where the right-most byte address of the referenced word is designated in the Breakpoint address switches. An example is the case of MS-RD or MS-WR operations which will not perform a breakpoint stop if any odd-numbered (right-most byte) address is designated by the BREAKPOINT ADDRESS SELECT.

For console operations (when the CONSOLE CONTROL SELECT switch is set to BREAKPOINT), the stop will always occur at the end of the storage reference cycle in which data is read or written at the Breakpoint address.

NOTE: The Breakpoint facility is not available for CS references in either Operator or Program Mode.

APPENDICES

APPENDIX A
COMPARISON OF 7200/7300 PERFORMANCE CHARACTERISTICS

PERFORMANCE CHARACTERISTICS	7300	7200
M_1 — Memory Reference Cycle Time	.9 μs 1.0 μs with ECC	1.8 μs ECC not available
M_2 — Non-memory Reference Cycle Time	.8 μs	1.6 μs

NOTE: The above times will be increased by .2 μs if the computer is used for special purposes which require the machine to be run in a single processor state only.

APPENDIX B
MNEMONIC CODE TO HEX CODE

ADD	A2	CMPM	61	LLSR	2C
ADDD	B2	CMPR	21	LOD	A0
ADDI	32	CMPT	71	LODB	F7
ADDK	52	CMPX	55	LODD	B0
ADDM	62	CSTR	2A	LODI	30
ADDR	22	CTB	12	LODT	70
ADDT	72	CVB	AA	LRDI	5D
AND	A5	CVBT	AA	LRDR	3D
ANDD	B5	CVD	AB	LRSI	4D
ANDI	35	CVDT	AB	LRSR	2D
ANDM	65	DIO	F2	MOVB	6A
ANDR	25	DIV	A9	MOVL	5A
ARDI	5F	DIVD	B9	MOVM	60
ARDR	3F	DIVK	7C	MOVR	20
ARSI	4F	DIVI	39	MOVX	54
ARSR	2F	DIVM	69	MPY	A8
B	ED	DIVR	29	MPYD	B8
BA1	E4	EDTX	57	MPYK	5B
BA2	E5	EOR	A6	MPYI	38
BCF	E9	EORD	B6	MPYM	68
BCH	EC	EORI	36	MPYR	28
BCM	EF	EORM	66	NOP	EE
BCT	E8	EORR	26	OUT	F6
BOF	E2	IBIT	BF	PAKX	58
BON	E3	INP	F5	PSTR	3A
BR	EB	INV	A4	RAR	FE
BRN	E1	INVD	B4	RBA	10
BRZ	E0	INVI	34	RBIT	BD
BS1	E6	INVM	64	RCN	14
BS2	E7	INVR	24	RDC	F3
BSR	EA	IOR	A7	RDX	F0
CBY	F9	IORD	B7	RLDI	5E
CBYM	6B	IORI	37	RLDR	3E
CLDR	2B	IORM	67	RLSI	4E
CMP	A1	IORR	27	RLSR	2E
CMPD	B1	LLDI	5C	ROFR	6F
CMPI	31	LLDR	3C	RONR	6D
CMPK	51	LLSI	4C	RPM	15

RSAR	FF	SRMB	47	SUBK	53
SB	BB	SRMF	46	SUBM	63
SF	BA	SRNB	43	SUBR	23
SAR	FF	SRNF	42	SUBT	73
SBA	10	SRPB	45	TBIT	BE
SBIT	BC	SRPF	44	TOFR	6E
SCFB	4B	SRZB	41	TONR	6C
SCFF	49	SRZF	40	TRNX	56
SCN	14	STO	FA	TST	11
SCTB	4A	STOB	F8	UNPX	59
SCTF	48	STOT	FB	WAR	FE
SHFK	3B	SUB	A3	WRC	F4
SIO	F1	SUBD	B3	WRX	F0
SPM	15	SUBI	33	ZADK	50
SR	13				

APPENDIX C
MACHINE LANGUAGE INSTRUCTION TIMING FORMULAS

This appendix lists formulas for calculating execution times of machine language instructions. Instructions are listed by hexadecimal operation code and assembler mnemonic. In most cases, times are dependent on the type of addressing (direct/indirect) used for instruction operands. Formulas for each of the addressing combinations possible are listed, where: A refers to operand 1 with direct addressing, B to operand 2 with direct addressing, (A) to operand 1 with indirect addressing, and (B) to operand 2 with indirect addressing. A legend of meanings of symbols used in the formulas is at the end of this appendix.

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
10	SBA	$M_1 + 6M_2$	$M_1 + 6M_2$		
10	RBA			$M_1 + 6M_2$	$M_1 + 6M_2$
11	TST	M_1+3M_2 (Unprivileged Bit Set)	$M_1 + 4M_2$ (Unprivileged Bit Not Set)	$M_1 + 4M_2$ (Privileged Bit Set)	$M_1 + 5M_2$ (Privileged Bit Not Set)
12	CTB	$M_1 + 3M_2$ (Unprivileged)		$M_1 + 4M_2$ (Privileged)	
13	SR	$M_1 + 2M_2$			
14	SCN	$M_1 + 7M_2$ (Processors 4-7)	$M_1 + 8M_2$ (Processors 0-3)		
14	RCN			$M_1 + 7M_2$ (Processors 4-7)	$M_1 + 8M_2$ (Processors 0-3)
15	SPM	$M_1 + 3M_2$ (Unprivileged Mode)	$M_1 + 6M_2$ (Privileged Mode)		
15	RPM			$M_1 + 3M_2$ (Unprivileged Mode)	$M_1 + 6M_2$ (Privileged Mode)
20	MOVR	$M_1 + M_2$	$2M_1$	$2M_1 + M_2$	$3M_1$
21	CMPR	$M_1 + M_2$	$2M_1$	$2M_1 + 2M_2$	$3M_1 + M_2$
22	ADDR	$M_1 + M_2$	$2M_1$	$3M_1 + M_2$	$4M_1$
23	SUBR	$M_1 + M_2$	$2M_1$	$3M_1 + M_2$	$4M_1$
24	INVR	$M_1 + M_2$	$2M_1$	$2M_1 + M_2$	$3M_1$
25	ANDR	$M_1 + M_2$	$2M_1$	$3M_1 + M_2$	$4M_1$
26	EORR	$M_1 + M_2$	$2M_1$	$3M_1 + M_2$	$4M_1$
27	IORR	$M_1 + M_2$	$2M_1$	$3M_1 + M_2$	$4M_1$
28	MPYR	$2M_1+7M_2+K_1$	$3M_1+6M_2+K_1$	$5M_1+5M_2+K_1$	$6M_1+4M_2+K_1$
29	DIVR	$M_1+5M_2+K_2$	$2M_1+4M_2+K_2$	$5M_1+3M_2+K_2$	$6M_1+2M_2+K_2$
2A	CSTR	$M_1 + M_2$		$2M_1$	
2B	CLDR	$M_1 + M_2$	$2M_1$		
2C	LLSR	$M_1 + M_2$	$2M_1 + M_2$		

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
2D	LRSR	$M_1 + 2M_2$	$2M_1 + 2M_2$		
2E	RLSR	$M_1 + M_2$	$2M_1 + M_2$		
2F	ARSR	$M_1 + 3M_2$	$2M_1 + 3M_2$		
30	LODI	$M_1 + 2M_2$		$2M_1 + M_2$	
31	CMPI	$M_1 + 2M_2$		$2M_1 + 2M_2$	
32	ADDI	$M_1 + 2M_2$		$3M_1 + M_2$	
33	SUBI	$M_1 + 2M_2$		$3M_1 + M_2$	
34	INVI	$M_1 + 2M_2$		$2M_1 + M_2$	
35	ANDI	$M_1 + 2M_2$		$3M_1 + M_2$	
36	EORI	$M_1 + 2M_2$		$3M_1 + M_2$	
37	IORI	$M_1 + 2M_2$		$3M_1 + M_2$	
38	MPYI	$2M_1+8M_2K_1$		$5M_1+5M_2+K_1$	
39	DIVI	$M_1+6M_2+K_2$		$5M_1+3M_2+K_2$	
3A	PSTR		$M_1 + M_2$		$2M_1$
3B	SHFK	See special formula at end			
3C	LLDR	$M_1 + 5M_2$	$2M_1 + 5M_2$		
3D	LRDR	$M_1 + 5M_2$	$2M_1 + 5M_2$		
3E	RLDR	$M_1 + 5M_2$	$2M_1 + 5M_2$		
3F	ARDR	$M_1 + 5M_2$	$2M_1 + 5M_2$		
40	SRZF	$M_1 + 3M_2$			
41	SRZB	$M_1 + 3M_2$			
42	SRNF	$M_1 + 3M_2$			
43	SRNB	$M_1 + 3M_2$			
44	SRPF	$M_1 + 3M_2$			
45	SRPB	$M_1 + 3M_2$			
46	SRMF	$M_1 + 3M_2$			

See Note A

See Note B

Note A: M_1+M_2 if shift count=0

Note B: $2M_1+M_2$ if shift count=0

$M_1 + M_2$ if skip not taken

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
47	SRMB	$M_1 + 3M_2$			
48	SCTF	$M_1 + 3M_2$			
49	SCFF	$M_1 + 3M_2$	$M_1 + M_2$ if skip not taken		
4A	SCTB	$M_1 + 3M_2$			
4B	SCFB	$M_1 + 3M_2$			
4C	LLSI	$M_1 + M_2$			
4D	LRSI	$M_1 + 2M_2$			
4E	RLSI	$M_1 + M_2$			
4F	ARSI	$M_1 + 3M_2$			
50	ZADK				
51	CMPK				
52	ADDK				
53	SUBK				
54	MOVX				
55	CMPX	See special formulas at end			
56	TRNX				
57	EDTX				
58	PAKX				
59	UNPX				
5A	MOVL				
5B	MPYK				
5C	LLDI	$M_1 + 5M_2$			
5D	LRDI	$M_1 + 5M_2$			
5E	RLDI	$M_1 + 5M_2$	$M_1 + M_2$ if shift count=0		
5F	ARDI	$M_1 + 5M_2$			
60	MOVM	$5M_1$	$6M_1$	$6M_1$	$7M_1$
61	CMPM	$5M_1 + M_2$	$6M_1 + M_2$	$6M_1 + M_2$	$7M_1 + M_2$

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
62	ADDM	$6M_1$	$7M_1$	$7M_1$	$8M_1$
63	SUBM	$6M_1$	$7M_1$	$7M_1$	$8M_1$
64	INVM	$5M_1$	$6M_1$	$6M_1$	$7M_1$
65	ANDM	$6M_1$	$7M_1$	$7M_1$	$8M_1$
66	EORM	$6M_1$	$7M_1$	$7M_1$	$8M_1$
67	IORM	$6M_1$	$7M_1$	$7M_1$	$8M_1$
68	MPYM	$8M_1+4M_2+K_1$	$9M_1+4M_2+K_1$	$9M_1+4M_2+K_1$	$10M_1+4M_2+K_1$
69	DIVM	$8M_1+2M_2+K_2$	$9M_1+2M_2+K_2$	$9M_1+2M_2+K_2$	$10M_1+2M_2+K_2$
6A	MOVB	$5M_1$	$6M_1$	$6M_1$	$7M_1$
6B	CBYM	$5M_1 + M_2$	$6M_1 + M_2$	$6M_1 + M_2$	$7M_1 + M_2$
6C	TONR	$M_1 + 2M_2$	$2M_1 + 2M_2$	$3M_1 + 2M_2$	$4M_1 + 2M_2$
6D	RONR	$M_1 + 3M_2$	$3M_1 + 2M_2$	$3M_1 + 3M_2$	$5M_1 + 2M_2$
6E	TOFR	$M_1 + 2M_2$	$2M_1 + 2M_2$	$3M_1 + 2M_2$	$4M_1 + 2M_2$
6F	ROFR	$M_1 + 3M_2$	$3M_1 + 2M_2$	$3M_1 + 3M_2$	$5M_1 + 2M_2$
70	LODT	$4M_1$	$5M_1$	$6M_1$	$7M_1$
71	CMPT	$3.5M_1+M_2$(avg.)	$4.5M_1+M_2$(avg.)	$5M_1+M_2$(avg.)	$6M_1+M_2$(avg.)
72	ADDT	$4M_1 + M_2$	$5M_1 + M_2$	$8M_1$	$9M_1$
73	SUBT	$4M_1 + M_2$	$5M_1 + M_2$	$8M_1$	$9M_1$
7C	DIVK	See special formula at end.			
A0	LOD	$3M_1$	$4M_1$	$4M_1$	$5M_1$
A1	CMP	$3M_1$	$4M_1$	$4M_1 + M_2$	$5M_1 + M_2$
A2	ADD	$3M_1$	$4M_1$	$5M_1$	$6M_1$
A3	SUB	$3M_1$	$4M_1$	$5M_1$	$6M_1$
A4	INV	$3M_1$	$4M_1$	$4M_1$	$5M_1$
A5	AND	$3M_1$	$4M_1$	$5M_1$	$6M_1$
A6	EOR	$3M_1$	$4M_1$	$5M_1$	$6M_1$
A7	IOR	$3M_1$	$4M_1$	$5M_1$	$6M_1$
A8	MPY	$4M_1+6M_2+K_1$	$5M_1+6M_2+K_1$	$7M_1+4M_2+K_1$	$8M_1+4M_2+K_1$

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
A9	DIV	$3M_1+4M_2+K_2$	$4M_1+4M_2+K_2$	$7M_1+2M_2+K_2$	$8M_1+2M_2+K_2$
AA	CVB	$4M_1+46M_2$	$5M_1+46M_2$		
AA	CVBT			$6M_1+90M_2$	$7M_1+90M_2$
AB	CVD	$6M_1 + K_3$	$7M_1 + K_3$		
AB	CVDT			$9M_1+38M_2+K_3$	$10M_1+38M_2+K_3$
B0	LODD	$2M_1 + M_2$		$3M_1$	
B1	CMPD	$2M_1 + M_2$		$3M_1 + M_2$	
B2	ADDD	$2M_1 + M_2$		$4M_1$	
B3	SUBD	$2M_1 + M_2$		$4M_1$	
B4	INVD	$2M_1 + M_2$		$3M_1$	
B5	ANDD	$2M_1 + M_2$		$4M_1$	
B6	EORD	$2M_1 + M_2$		$4M_1$	
B7	IORD	$2M_1 + M_2$		$4M_1$	
B8	MPYD	$3M_1+7M_2+K_1$		$6M_1+4M_2+K_1$	
B9	DIVD	$2M_1+5M_2+K_2$		$6M_1+2M_2+K_2$	
BA	SF	$M_1 + 2M_2$			
BB	SB	$M_1 + 2M_2$			
BC	SBIT	$4M_1$	$5M_1$		
BD	RBIT	$4M_1$	$5M_1$		
BE	TBIT	$3M_1 + M_2$	$4M_1 + M_2$		
BF	IBIT	$4M_1$	$5M_1$		

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
E0	BRZ	$2M_1$	$3M_1$	$3M_1$	$4M_1$
E1	BRN	$2M_1$	$3M_1$	$3M_1$	$4M_1$
E2	BOF	$2M_1$	$3M_1$		
E3	BON	$2M_1$	$3M_1$		
E4	BA1	$2M_1$	$3M_1$	$4M_1$	$5M_1$
E5	BA2	$2M_1$	$3M_1$	$4M_1$	$5M_1$
E6	BS1	$2M_1$	$3M_1$	$4M_1$	$5M_1$
E7	BS2	$2M_1$	$3M_1$	$4M_1$	$5M_1$
E8	BCT	$2M_1$	$3M_1$		
E9	BCF	$2M_1$	$3M_1$		
EA	BSR	$2M_1 + M_2$	$3M_1 + M_2$	$3M_1$	$4M_1$
EB	BR	$M_1 + M_2$	$2M_1$		
EC	BCH	$2M_1$	$3M_1$		
ED	B	$2M_1$	$3M_1$		
EE	NOP	$M_1 + M_2$	$M_1 + M_2$	$M_1 + M_2$	$M_1 + M_2$
EF	BCM	$M_1 + M_2$			
F0	RDX	$M_1 + 3M_2$			
F0	WRX			$M_1 + 4M_2$	
F1	SIO				
F2	DIO				
F3	RDC				
F4	WRC	$M_1 + 2M_2$			
F5	INP	$M_1 + 5M_2$		$2M_1 + 5M_2$	
F6	OUT	$M_1 + 4M_2$		$2M_1 + 4M_2$	
F7	LODB	$3M_1$	$4M_1$	$4M_1$	$5M_1$
F8	STOB	$3M_1$	$4M_1$	$4M_1$	$5M_1$
F9	CBY	$3M_1 + M_2$	$4M_1 + M_2$	$4M_1 + M_2$	$5M_1 + M_2$

2M if jump not taken

Hex Code	Mnemonic	A-B	(A)-B	A-(B)	(A)-(B)
FA	STO	$3M_1$	$4M_1$	$4M_1$	$5M_1$
FB	STOT	$4M_1$	$5M_1$	$6M_1$	$7M_1$
FD	RRO	$3M_1 + 3M_2$		$4M_1 + 3M_2$	
FD	WRO		$3M_1 + 3M_2$		$4M_1 + 2M_2$
FE	RAR	$2M_1 + 3M_2$		$3M_1 + 3M_2$	
FE	WAR		$2M_1 + 3M_2$		$3M_1 + 3M_2$
FF	SAR	$12M_1+22M_2$			
FF	RSAR		$12M_1+24M_2$		

SPECIAL FORMULAS

ZADK	L2≤L1	$(6+4L1*i_9)M_1+[11+(10+6L1-2L2)i_9]M_2$	
	L2>L1	$[+(L2+3L1)i_9]M_1+[15+(10+L2+3L1)i_9]M_2$	
CMPK	L2≤L1	$(5+i_8+2i_9+L2+L1)M_1+(3+5+i_9+3L1)M_2$	
	L2>L1	$(6+2i_9+L2+L1)M_1+(8+3L2)M_2$	
ADDK	L2≤L1	$(6+3L1*i_9)M_1+[10+(10+4L1-2L2)i_9]M_2$	
	L2>L1	$[6+(L2+2L1)i_9]M_1+[14+(10+2L1+L2)i_9]M_2$	
	if recomplement add:	$(3L1+1)M_1+(5L1+7)M_2$	
SUBK	same as ADDK		
MOVX	L2≥L1	$(5+L1)M_1+3M_2$	(word move)*
		$(6+2L1)M_1+(2+i_7)M_2$	(byte move)*
	L2<L1	$(5+L2+L1)M_1+3M_2$	
CMPX	L2≥L1	$(4+L2)M_1+(3+L2+.5L1)M_2$	(word compare)*
		$(4+2L2)M_1+(3+2L2+L1)M_2$	(byte compare)*
	L2<L1	$(4+L1)M_1+(3+L1+.5L2)M_2$	(word compare)*
		$(4+2L1)M_1+(3+2L1+L2)M_2$	(byte compare)*
TRNX		$(7+3L1)M_1+(4+L1)M_2$	
EDTX	Numeric Edit (L2≠0)		
		$(4+L_m+L2+L1)M_1+(13+3C_1+6C_2+C_3+2C_4+3C_5$	
		$+7.5C_6)M_2+[(1.5L1+1)M_1+(5.5L1-1.5i_7$	
		$+i_6+9)M_2]i_9$	
	Alpha Edit (12=0)		
		$(4+L_m+2C_1+C_4)M_1 + (5+2C_1+C_3+2C_4)M_2$	

PAKX $4M_1+7M_2+[(3L1+i_8)M_1+(10L1+i_8-1)M_2]i_9$

UNPX $4M_1+7M_2+[(L1+L2*i_8)M_1+(5.5L1-1.5i_7+i_6+9)M_2]i_9$

MOVL $(5+L2)M_1+2M_2$ (word move)*

$(5+2L2)M_1+2M_2$ (byte move)*

SHFK $[3+(7L2+2i_{15}+2i_{16})i_8]M_1$

$+[7+(8+15L2-(5+i_{15})i_{12}+(3+2i_{15}+7i_{12})i_{14})i_8]M_2$

MPYK $[7+2N_Z+4L_2+((20L_1+2L_2-8)N_S+10L_1-5)N_0]M_1$

$+ [16+2M_Z+((27.5L_1-7.75)N_S+13.75L_1-6.87)N_0]M_2$

DIVK $[15+24L_1(L_2-L_1)+2L_2(L_2-L_1-5)]M_1$

$+ [35+52L_1(L_2-L_1)-6L_1-20L_2]M_2$

LEGEND

M_1= Memory reference cycle time

M_2= Non-memory reference cycle time

$K_1 = (i_1+i_2+i_3+i_4+2i_5)M_2$

$K_2 = ((51+i_1+i_2)M_2)i_0$

$K_3 = [44+2i_2+i_{17}+3i_{18}(15+16i_{19}-i_{20})]M_2$

L1, L2 } lengths as specified in the machine language

L_m=length of edit mask including new fill characters, characters to be inserted and all the edit operators.

C_1=count of source characters (digits) moved to result via the MC or MCS operators.

C_2=count of source digits suppressed by fill.

C_3=count of IC and ICS operators in edit mask.

C_4=count of mask characters inserted or suppressed in result via IO and ICS operators.

C_5=count of SSD and SFI operators in edit mask.

C_6=count of ISG operators in edit mask.

i_0 = 0 if A=0
1 if A≠0

i_1 = 0 if A ≥ 0
1 if A < 0

i_2 = 0 if B ≥ 0
1 if B < 0

i_3 = 0 if A and B have like signs
1 if A and B have unlike signs

i_4 = number of one-bits in smaller of |A| or |B|

i_5 = 16 - bit# of MSB in smaller of |A| or |B|

i_6 = number of non-zero digits unpacked

i_7 = 0 if L1 is even
1 if L1 is odd

i_8 = 0 if L2=0
1 if L2≠0

i_9 = 0 if L1=0
1 if L1≠0

i_{10} = sign extended shift count from 6th byte of instruction

$i_{11} = i_{10}+(B)$

i_{12} = smaller of $|i_{11}|$ and (2L-1)

i_{14} = 0 if i_{12}=0
1 if i_{12}≠0

i_{15} = 0 if i_{11}> 0 left shift
1 if i_{11}< 0 right shift

i_{16} = 0 if result of shift≠0
1 if result of shift=0

i_{17} = number of one-bits in |B|

i_{18} = 0 if $i_{17} = 0$
1 if $i_{17} \neq 0$

i_{19} = 0 if single precision (CVD)
1 if double precision (CVDT)

i_{20} = bit position of MSB in B

N_0 = 1 if $N_Z < L_2$

0 if $N_Z = L_2$

N_Z = number of bytes of leading zeros in operand B field (multiplier field). ($N_Z \geq L_1$)

$N_S = L_2-N_Z-1$