7200 Processing Unit

Reference Manual

2903.002

MEMOREX

Computer System Products

Preliminary Information

This manual precedes initial release publications and therefore may undergo substantial revision.

Preliminary Edition October 1972

Requests for copies of Memorex publications should be made to your Memorex representative or to the Memorex branch office serving your locality.

A readers' comments form is provided at the back of this publication. If the form has been removed, comments may be addressed to the Memorex Corporation, Publications Dept., Santa Clara, California 95052.

7200 PROCESSING UNIT REFERENCE
(MRX/40 SYSTEM)

PREFACE

This publication covers the 7200 Processing Unit (for MRX/40 data-processing systems). It describes the functional characteristics, data formats, operating routines, and the System Control Panel. Each machine instruction is detailed, with a brief example.

The publication is intended to aid programmers, system engineers, and data-processing managers and operators in better understanding and using the system.

Appendices provide further details of machine formats and instructions.

September, 1972

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

1. GENERAL DESCRIPTION

INTRODUCTION

The MEMOREX 7200 Processing Unit is the major component in an I/O-oriented, business data-processing system. Its basic repertoire of 159 instructions provides a powerful facility for both business data-processing and scientific problem-solving.

The 7200 is a byte (8 bit)-oriented processing unit. In addition to single-precision operations in bytes and words (16 bits), the basic instruction set accommodates problems requiring double-precision (32-bit) solutions. An optional set of ten floating-point instructions is available to extend the scientific capabilities of the system.

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 is 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 computer systems:

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

PROCESSOR STATE CONCEPT

GENERAL

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 7200 Processing Unit divides its running time into segments called *major cycles*. These time segments are cyclically assigned to one of eight *processor states*. For each state, a group of hardware registers holds information relating to the operation currently being performed by that state. A processor state and its associated group of registers, then, constitute a resource that is "dedicated" to the solution of a particular problem. In order to effect this solution, each processor state shares with the other states the use of the common (or "shared") resources of the computer.

Because only one processor state is active for a given time segment, the shared resources (principally, the arithmetic/logical unit and main storage) need concentrate on but one task during each major cycle.

By assigning specific types of tasks to individual processor states, as well as by allocating major cycles sequentially to each state that is waiting to execute the next segment of its task, and finally by never granting a processor state two consecutive major cycles if another state is awaiting its turn, the processing unit ensures that the system will not be bound up by either computation or input-output activities.

Figure 1-1 shows the arrangement of the dedicated resources with respect to major-cycle assignments. If all processor states have tasks to do, the first major cycle is given to state 0, the second to state 1, and so on. When processor state 7 has had its turn (using major cycle 8), the scanning sequence begins again: cycle 9 is assigned to state 0, cycle 10 to state 1, and so on. If processor state 1 had no task to perform, state 2 would receive cycle 2, with state 7 receiving cycle 7. Then, during the second scan, state 2 would receive cycle 9.

In each scan, the I/O processor states are given precedence. This is because those states communicate with peripheral equipment, and normally handle that communication on a "can't wait" basis. Processor states that do not operate under such time constraints are assigned lower-order positions in the scanning sequence.

A software-controlled priority mode is provided for those instances when an I/O processor state can't wait for its normal turn in the sequence. If that situation occurs, the Resource Allocation Network — which assigns the major cycles — grants the I/O processor state an out-of-sequence major cycle. This decision is made at the end of each major cycle; the action is similar to that of a system operating under a priority-interrupt scheme. It is this tailoring of the Resource Allocation Network around the I/O requirements that makes the MRX/40 system so well suited for file management and data inquiry-retrieval applications.

CONSECUTIVE-CYCLE MODE

As implied earlier, the major cycles are equitably distributed among the processor states requesting access to shared resources. To prevent a monopoly of the shared resources, the Resource Allocation Network does not normally give two successive major cycles to the same processor states.

For those relatively infrequent cases where only one processor state is requesting access, the Resource Allocation Network may be directed to give that state consecutive cycles. This is accomplished through a bit in the Control register.

Specific information for setting up the consecutive-cycle mode is given under the discussion of the Control register in Section 4.

The Job Accounting option keeps track of the number of major cycles the Resource Allocation Network grants to each of the processor states.

MAJOR-CYCLE TIMING

Every major cycle consists of a number of 200-nanosecond "minor cycles", during which time individual micro instructions (μI's) are executed. References to main storage are also made during the major cycle.

The length of each major cycle — and consequently the number of minor cycles contained within it — depends upon whether or not a main storage reference is made. The resultant variations in major-cycle length are shown below.

Function	Minor Cycles	Length (Micro-seconds)
Execute instructions	8	1.6
Read or write Main Storage; execute instructions	9	1.8

PROCESSING-UNIT ORGANIZATION

A simplified block diagram of the 7200 Processing Unit is shown in Figure 1-2. The processor states gain access to the shared resources via the Resource Allocation Network, the function of which was implied earlier. Descriptions of other elements in the diagram follow. Information for the systems programmer about the inter-relationships of these components is contained in Section 4 of this manual.

PROCESSOR STATES

All eight processor states have equal capacity to perform logical, arithmetic, and bit-manipulative operations. Moreover, processor states 0 through 4 utilize special hardware that enable them to accomplish specific tasks. Because of this specificity, it is convenient to assign a name to each processor state according to its prime dedicated task:

- Processor State 0: Communications
- Processor State 1: Not assigned
- Processor State 2: Selector Channel

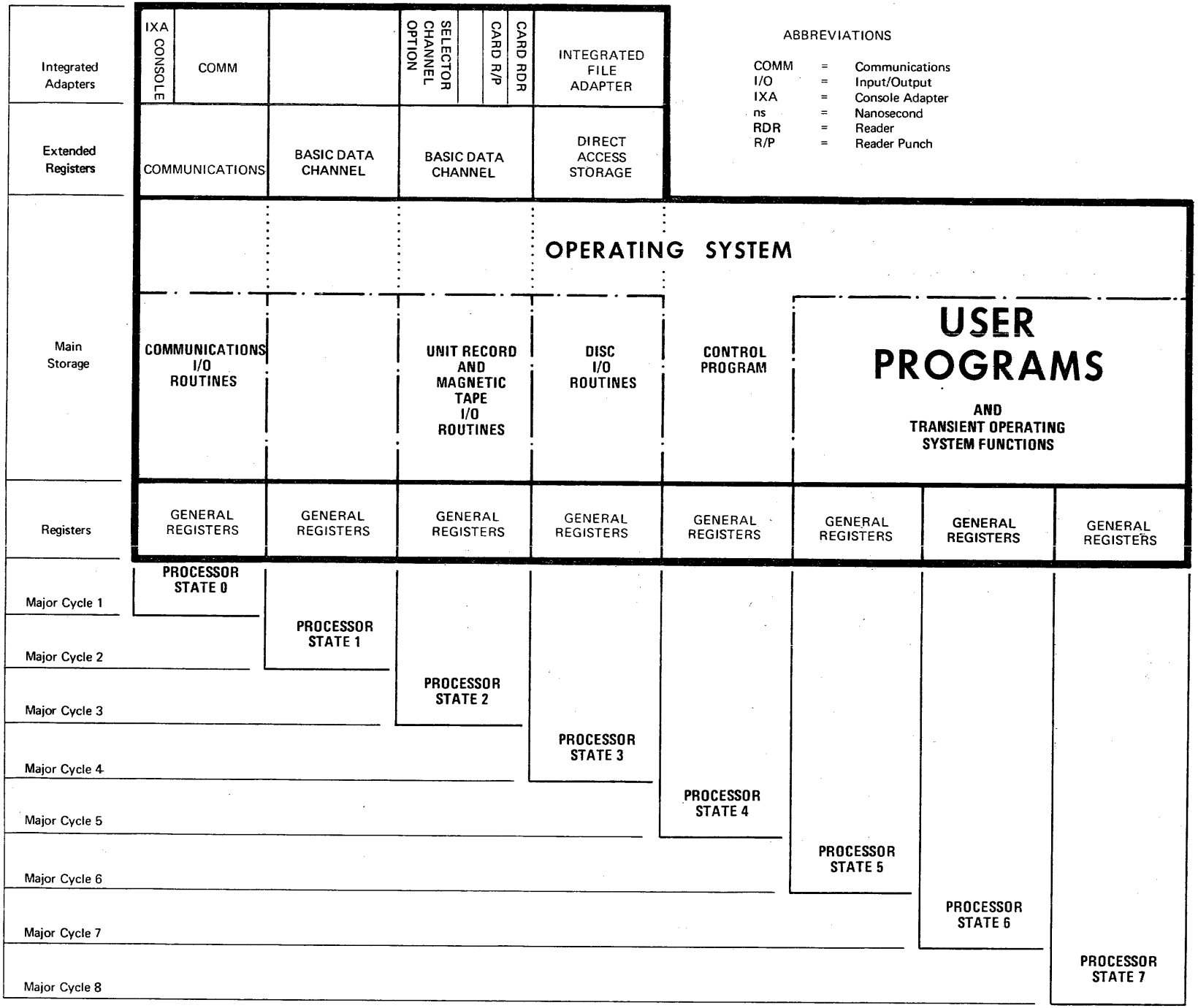
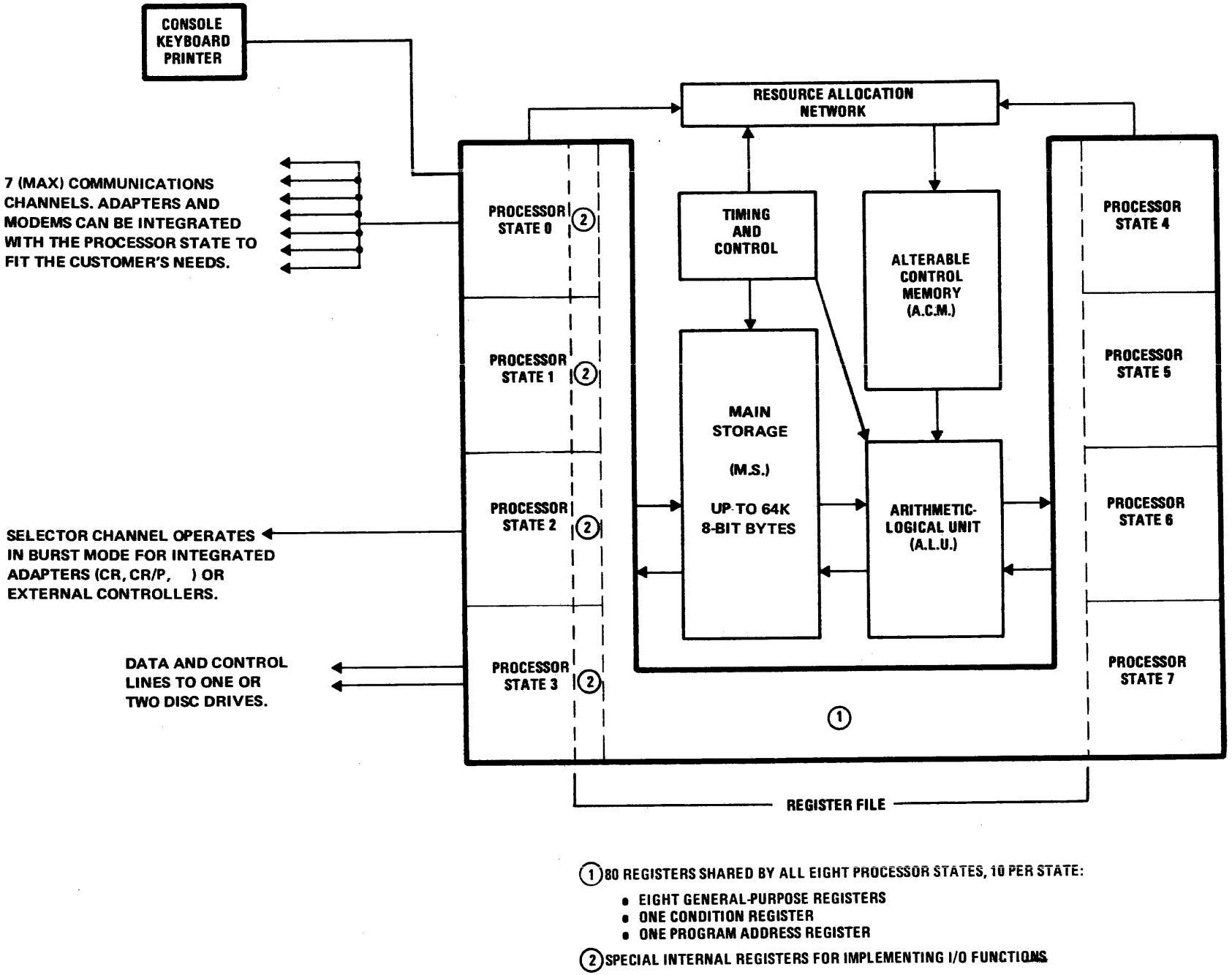


Figure 1-1. 7200 Processing Unit Architecture



(1) 80 REGISTERS SHARED BY ALL EIGHT PROCESSOR STATES, 10 PER STATE:

- EIGHT GENERAL-PURPOSE REGISTERS
- ONE CONDITION REGISTER
- ONE PROGRAM ADDRESS REGISTER

(2) SPECIAL INTERNAL REGISTERS FOR IMPLEMENTING I/O FUNCTIONS

Figure 1-2. 7200 Processing Unit Block Diagram

- Processor State 3: Disc
- Processor State 4: Executive
- Processor States 5, 6, 7: General Purpose

Processor State 0

This processor state contains an integrated communications adapter (ICA) to provide serial data communication over as many as 7 communications lines for both synchronous and asynchronous terminals utilizing switched networks and dedicated or local lines. An integrated console adapter, for service to the console keyboard/printer, is also attached to Processor State 0.

Each communications line is controlled by a line adapter that is field-modifiable through interchangeable printed-circuit boards to accommodate a wide variety of customer applications.

The ICA can communicate with local or remote devices operating with seven-level-plus-parity or eight-level codes in the following speeds.

Asynchronous		Synchronous	
Baud (bits/sec)	Characters/ sec	Baud (bits/sec)	Characters/ sec
110	10	600	75
150	15	1200	150
300	30	1800	225
600	60	2000	250
1200	120	2400	300
		3600	450
		4800	800
		9600	1200

Three methods of operation are provided:

1. Operating a local terminal over lines up to 50 feet in length (no modem required).
2. Communicating, via a common-carrier-provided data set or customer-provided modem, with a remote terminal using a compatible modem.
3. Communicating, via an internal modem and a common-carrier-provided data access arrangement, with a remote terminal using a compatible modem.

The asynchronous line adapters are full-duplex, permitting two-way simultaneous operation. In addition, when an adapter is supplemented by a suitable modem, a split-speed provision permits operating the primary channel at 1200 baud and the secondary channel at 150 baud. This mode allows either channel to be transmitting while the other channel receives. Echoplex is also supported; in this mode, the received message is transmitted back to the sending station, allowing the sending terminal to monitor the performance of the transmission line.

These line adapters feature the ability, under program control, to determine the speed of a remote terminal, to select a speed, to automatically answer, or to indicate a lost-data condition. Information may be coded in seven levels plus programmable parity, or in eight levels.

The asynchronous line adapters provide a subset of EIA standard RS-232-C interface (Data Transmission Configuration Interface Type L).

The synchronous line adapters are half duplex. However, when connected to a full-duplex facility, they permit two-way alternate message transfer without line turn around delay. Synchronous communications are provided for Basic or Code Transparent EBCDIC or ASCII.

The synchronous line adapters provide a subset of EIA standard RS-232-C interface (Data Transmission Configuration Interface Type D).

Processor State 2

This processor state controls data transfers to and from peripheral devices using integrated adapters, or to and from external controllers via the selector channel feature. An integrated card reader adapter (IRCA) provides control for a 1000-, 600-, or 300-card-per-minute reader; an integrated reader/punch adapter (IRPA) provides control for a 500/120 card-per-minute reader/punch.

In addition to the adapters, Processor State 2 can handle up to seven IBM* 360/370-compatible external controllers via the selector channel option. Parity is generated and checked for all transmissions to or from the external controllers.

Processor State 2 is capable of various modes of data transfer under control of different I/O Driver programs. It also can handle active devices in a controlled order of priority, as well as performing diagnostic testing operations.

Processor State 3

This processor state contains an integrated file adapter (IFA) with direct interface to the disc drives. It controls communication between main storage and one or two MEMOREX 3664 Disc Drives.

The IFA decodes commands to initiate these drive functions:

- Positioning read/write heads to a specific location on the disc surface (*seek*) and selecting a read/write head
- Locating a particular record or part of a record on the disc surface (*search*)
- Writing a record or part of a record
- Reading a record or part of a record

Seek operations may be performed concurrently. All other operations are non-concurrent and may be performed only on a selected drive.

Each record appears in fields of information separated by gaps. The disc processor can write count and data fields and read count, key and data fields. Command chaining is permitted; however, any attempt to read, write or search across a track boundary will result in an error condition.

*Tradename – International Business Machines

Processor State 4

This processor state is used by the operating system to monitor and control system operations. Unique functions performed by the operating system in this state are as follows:

1. Assigning the execution of programs to other processor states.
2. Assigning I/O operations within those programs to the cognizant I/O processor state.
3. Supplying time-of-day information when requested by the various programs.
4. Periodically reading the interval timer, and updating the time-of-day clock accordingly.
5. Initiating error-isolation and error-correction routines when needed.

Items 2 and 3 are effected in response to the *Service Request*, a processing-unit instruction issued by the general-purpose processor state requesting the information. (Certain instructions, including all of those directly relating to I/O, are "restricted", and cannot be executed by the general-purpose states.)

Processor States 5, 6, and 7

These are general-purpose processor states which may be assigned to execute user programs. Assigning one of these to a user program dedicates eight general-purpose registers, a condition register, and a program address register for exclusive use of that program. The assigning of general-purpose processor states to user programs is a function of the system's control program.

MAIN STORAGE

The 7200 Processing Unit provides byte addressability for up to 64K bytes of main storage (MS). Boundary protection during write references to storage is inherent for processor states 5, 6, and 7. Minimum storage size offered is 16K bytes.

ALTERABLE CONTROL MEMORY

The word-addressable ACM stores micro-instructions. There are 65 micro-instructions, used to form microprograms that implement execution of the processing unit instructions, control integrated I/O adapters, and control of certain special operations such as data entry or read-out from the System Control Panel.

ARITHMETIC-LOGICAL UNIT

Mathematical computations, Boolean functions (masking, for example), and data-shifting operations are handled by the ALU. Shifts of up to 15 binary positions may be made in a single pass through the 32-bit shift network. The existing ALU hardware readily accommodates double-precision and floating-point operations.

REGISTER FILE

All the registers associated with the various processor states are called, collectively, the register file.

The purpose of the register file is to hold information

pertinent to the operations currently being executed by the processor states. As mentioned earlier, the performance of a task by one processor state – executing a user application program, for example, or transferring 100 words to a disc – involves many brief periods of activity, each separated by wider spaces of dormancy during which other states are active. During each active period, the less time spent obtaining the operands, the better. The register file offers just such a means of rapid retrieval. Its access time is roughly one-seventh that of main storage.

The register file is divided into two functional areas, as shown in Figure 1-3. The first comprises the basic registers, of which each processor state has a set. The Basic register file is associated with the execution of processor state programs; its component parts are as follows:

- General Registers – used as the programmer desires: as index, base, transient, or working registers. (Exception: zero cannot be used as an index register; specifying it as an index modifier indicates no indexing.)

- Condition Register – records conditions resulting from instruction execution (results equal, for example). The specific conditions are detailed below.

- Program Address Register – contains the address of the instruction currently being executed by the associated processor state.

All registers in the Basic register file are addressable. A processor state may have access to any register within its own set (represented by the columns under the processor number in Figure 1-3) through use of the general-purpose machine-language instructions.

The Extended register file contains the I/O-related registers, the common block registers, and internal control registers which are used in conjunction with special-purpose functions, such as I/O data transfer and operating system control. The Extended registers have limited addressability. Some are addressable only by privileged instructions, and others only by restricted instructions.

A complete description of the Extended registers – their usage and functions – is in Section 4 under the heading *System Instructions.*

The Condition register records conditions resulting from the execution of both the basic instruction repertoire and the optional floating-point instructions. Twelve bits within the register record the resulting conditions. Figure 1-4 shows the conditions recorded for the basic instructions and their positions within the register.

- Bit 0 This bit position set indicates that arithmetic overflow occurred during an *add, subtract* or *divide* instruction. Being set during a *Zero and Add, Add Packed Decimal* or *Subtract Packed Decimal* instruction, it indicates that significant data within the field was lost. This bit position is always cleared following a *Compare* instruction.

- Bit 1 This bit position set following a *Compare* instruction indicates that the first operand R1 is greater than the second operand R2, as explained by the note in Figure 1-4. This position is also set for decimal arithmetic instructions when the result is positive.

- Bit 2 This bit position set following a *Compare* instruction indicates that the first operand R1 is less than the second operand R2 (note in Figure 1-4). This position is also set for decimal arithmetic instructions when the result is negative.

- Bit 3 This bit position set following a *Compare* instruction indicates that the two operands tested are identical. For decimal arithmetic instructions, it indicates that the result is zero. This position is also set for the non-decimal arithmetic instructions when a carry-out from adder bit position 0 occurs (link condition).

- Bit 4 This bit position is set during a packed decimal instruction if an EBCDIC character other than 0-9 is found in the unpacked field, or if a hex representation other than a letter A through F is found in the sign portion of the lowest-order byte. This position is always cleared following a *Compare* instruction.

- Bit 5 This bit position set following a *Compare* instruction indicates that the first operand R1 is greater than the second operand R2 (note on Figure 1-4).

- Bit 6 This bit position set following a *Compare* instruction indicates that the first operand R1 is less than the second operand R2 (note on Figure 1-4).

- Bit 7 This bit position set following a *Compare* instruction indicates that the two operands tested are identical.

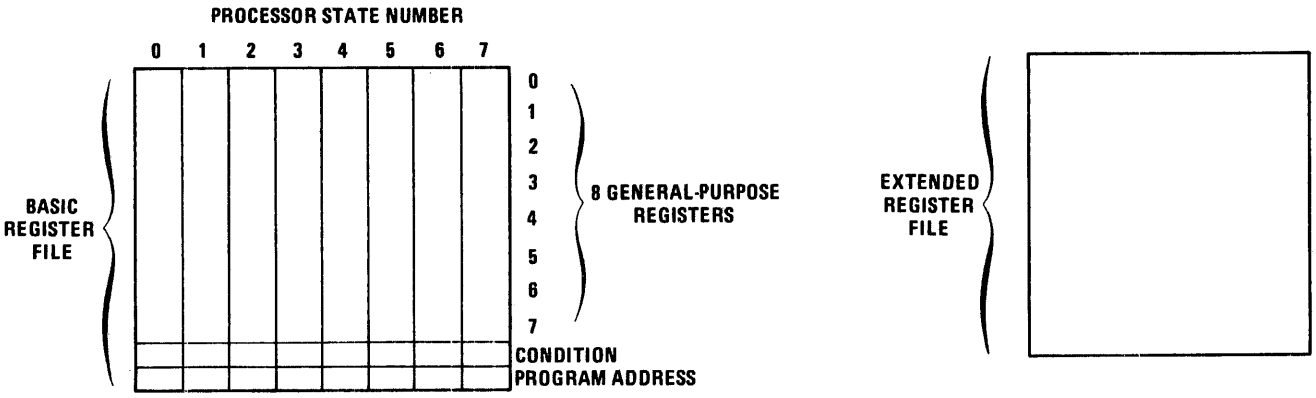
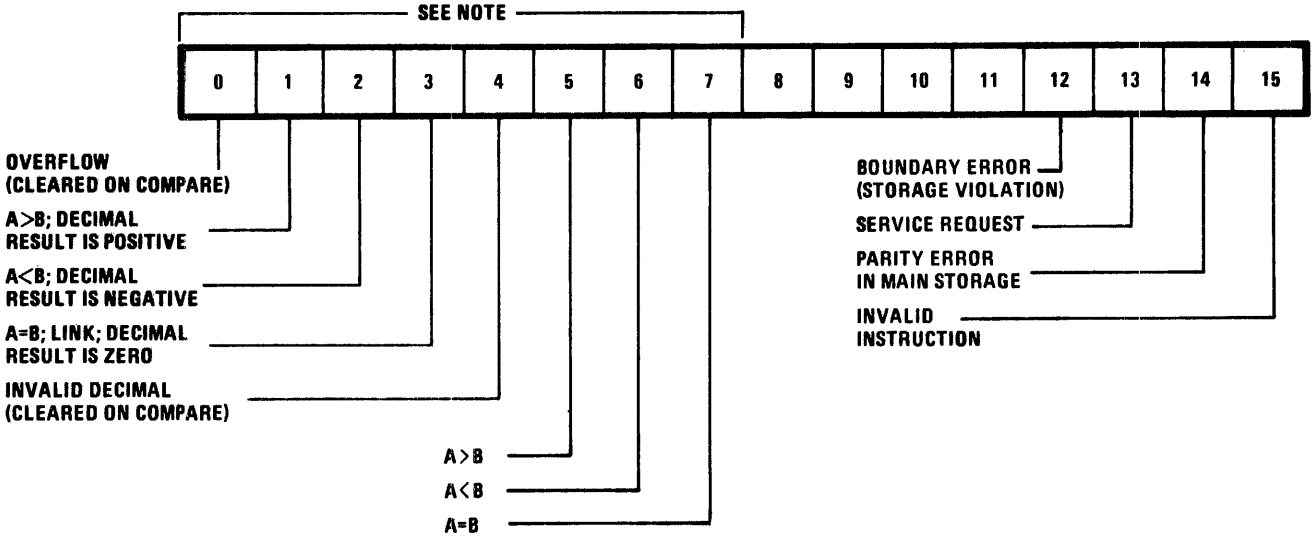


Figure 1-3. Register File Layout



NOTE

BIT GROUPS 0-3 AND 4-7 ARE BOTH SET AFTER ANY OF THE COMPARE INSTRUCTIONS. INTERPRETATION, WHETHER LOGICAL (MAGNITUDE ONLY) OR ARITHMETIC (SIGNED VALUES), DEPENDS UPON THE INSTRUCTION, AS FOLLOWS:

INSTRUCTION	PURPOSE	0-3	4-7
CMPX, CBY, CBYM	MAGNITUDE ONLY, BYTE-ORIENTED	LOGICAL	LOGICAL
CMPK	ARITHMETIC, PACKED DECIMAL	ARITHMETIC	ARITHMETIC
CMPF	ARITHMETIC, FLOATING POINT		
CMP, CMPD, CMPI, CMPM, CMPR, CMPT	ARITHMETIC, WORD-ORIENTED	ARITHMETIC	LOGICAL

Figure 1-4. Condition Register Assignments

- Bits 8-11 Reserved

- Bit 12 This bit position is set when a processor attempts to read from or write into main storage beyond its boundary limits invoked at the time.

- Bit 13 This bit position is set during the execution of a *Service Request* instruction.

- Bit 14 This bit position is set whenever a parity error is detected on a word (instruction or operand) read from main storage.

- Bit 15 This bit position is set when an undefined operation code is translated, or when an unprivileged processor attempts to execute a restricted or privileged instruction, or when a privileged processor attempts to execute an instruction that is restricted to another processor.*

Generally, the hardware writes into the Condition register in 4-bit hexadecimal groups. For example, if the result of a decimal arithmetic operation were positive, bit positions 1 and 5 would be set and positions 0, 2-4, 6 and 7 would be cleared. Bit positions 8-15 would be unchanged. On the other hand, for a *Compare* instruction that resulted in arithmetic and logical equality, 0-2 and 4-6 would be cleared, while 3 and 7 would be set. Bit positions 8-15 would be unchanged.

It is important to note that in the second instance, bit position 1 is cleared and no longer represents the result of the decimal arithmetic operation. This means that the Condition register must be examined before another instruction that can affect it is executed, or the prior condition indication will be lost.

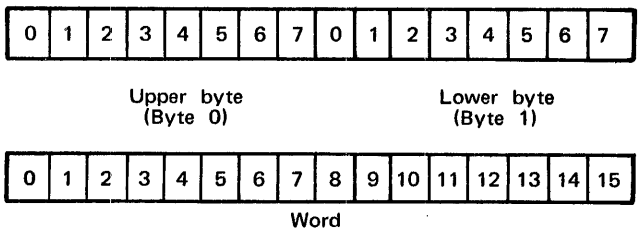
In the *Instruction Descriptions*, Section 4, any instruction that affects the Condition register will stipulate which bits are affected. It assumes, then, that other bits are unchanged.

*The terms "privileged" and "restricted" are defined in Section 4.

2. FUNCTIONAL CHARACTERISTICS

MAIN STORAGE ADDRESSING

The MEMOREX processing unit is a 16-bit, fixed-point, machine. The basic memory unit is an 8-bit byte; two bytes are contained in each 16-bit main storage word. The word and byte relationship, with their respective bit positions, is shown below.



For instructions, addressing must always be by words; however, the instructions themselves may deal with word-length (16-bit) or byte-length (8-bit) operands.

Though instructions vary in length, they must begin at a word address in main storage. The lowest-order bit of word addresses are not used; thus, all word addresses are even.

Byte-length operands may be in either byte of a storage word. Therefore, in byte-oriented instructions the lowest-order bit of the address is used to select the byte; the upper byte with an even address and the lower byte with an odd address.

BINARY REPRESENTATION

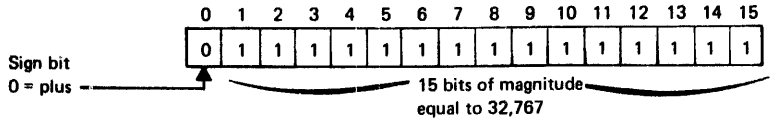
The arithmetic operands within the 7200 are either 16 or 32 bits represented in conventional signed binary notation. The highest-order bit specifies the sign, while the remaining bits specify the value of the operand. Positive operands are represented in true binary form with a 0 in the sign-bit position. Negative operands are represented in the two's complement form with a 1 in the sign bit.

ARITHMETIC

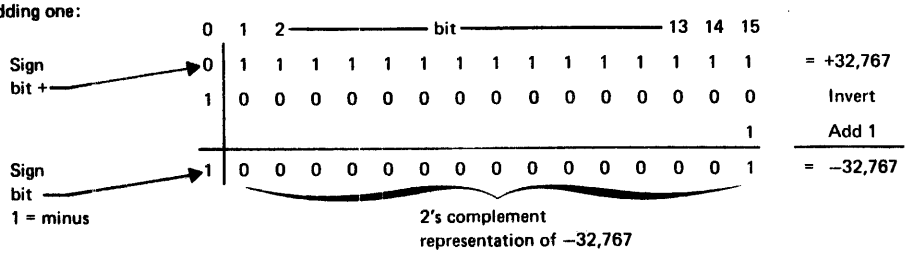
The 7200 performs both single-precision (16-bit) and double-precision (32-bit) addition and subtraction. The single-precision, fixed-point data format contains a 15-bit integer value and a sign bit (Figure 2-1).

POSITIVE AND NEGATIVE LIMITS

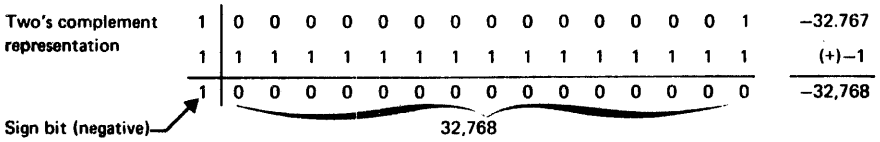
The largest positive number that can be expressed in a signed 16-bit word either in storage or in a register is +32,767:



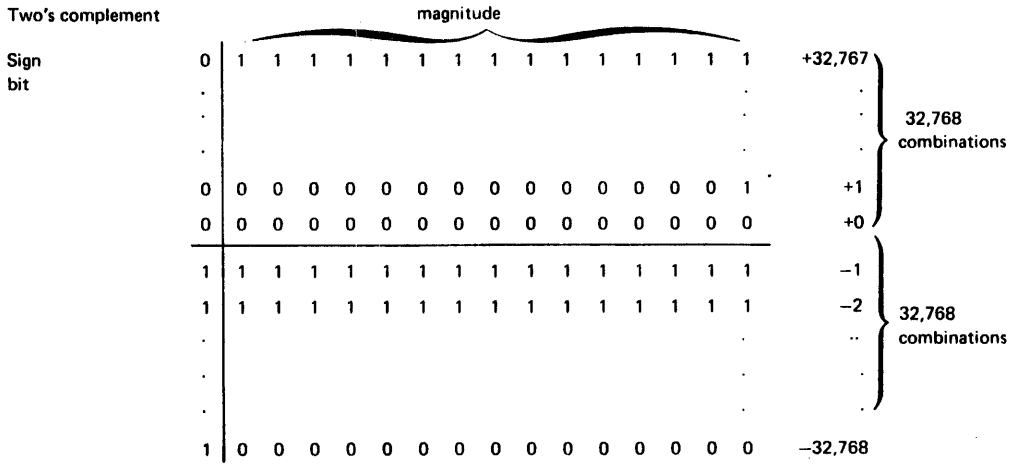
The two's complement representation for negative numbers is formed by inverting the positive binary representation and adding one:



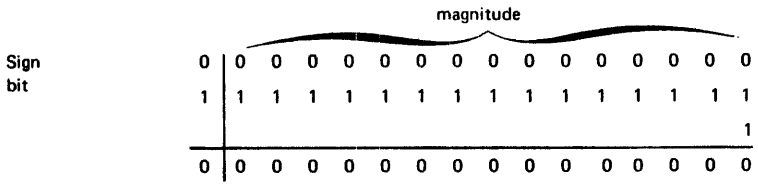
This -32,767 value, is however, one less in absolute value than the largest negative number that can be expressed in a signed 16-bit word. The largest negative number is formed by adding (-1) to -32,767.



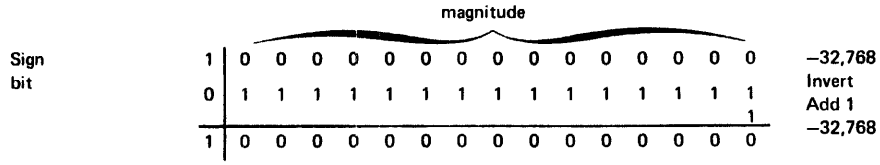
Although the same quantity of both positive and negative numbers can be represented in a signed 16-bit machine, the two's complement usage causes the positive and negative range of numbers to be offset by one number:



As a result of the offset there is one positive and one negative number which cannot be represented in its opposite form. The value *zero* can only be expressed as a positive number, which is apparent when we attempt to generate the two's complement representation for positive zero.



The same is true with negative 32,768, which can only be expressed as a negative value in single-precision notation.



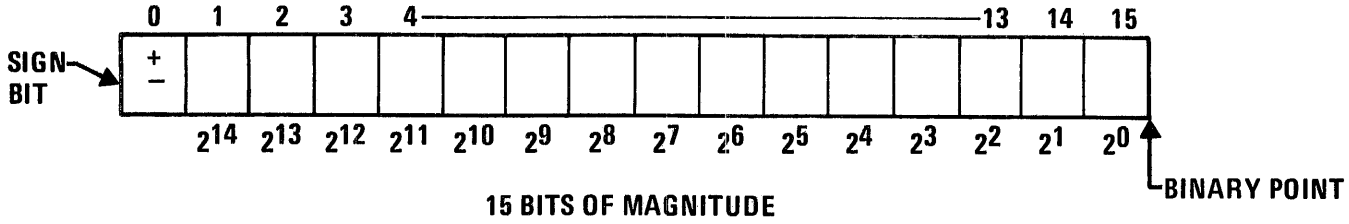
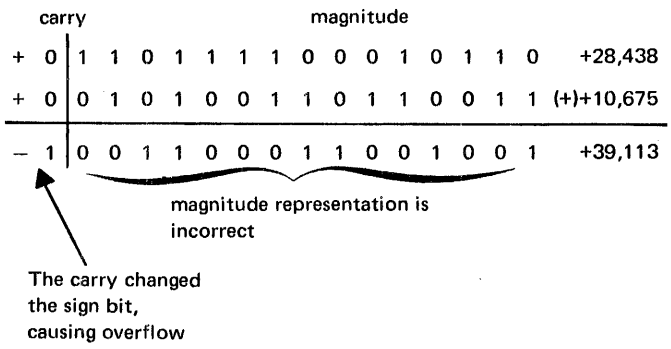


Figure 2-1. Single-Precision Fixed-Point Format

SINGLE-PRECISION ADDITION

When both operands have like signs, the resulting sum may be too large to express in fifteen bits of magnitude. In this case a carry will propagate from the magnitude field into the sign bit position, causing the sign to change — this is overflow, recorded in the Condition register. In this example two positive numbers are added, resulting in a sum too large for the magnitude field. The resultant overflow sum is an incorrect negative value.



In this example, two positive numbers are added, resulting in a sum too large for the magnitude field. The resultant overflow sum is an incorrect negative value.

The addition of two operands with unlike signs cannot generate overflow since the result is the difference between the two, and it will be less than or equal to the larger operand.

SINGLE-PRECISION SUBTRACTION

Single-precision subtraction of two operands with unlike signs can result in overflow. Of course, overflow will only occur when the result is too large to express in 15 bits of magnitude. Overflow cannot occur when like-signed values are subtracted, because the adder result is really the difference between the two operands.

DOUBLE-PRECISION

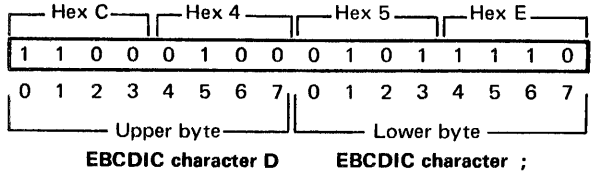
Double-precision, fixed-point operands contain a 31-bit integer value and one sign bit. A 32-bit operand resides in two adjacent storage locations or two adjacent file registers (Figure 2-2).

INFORMATION FORMATS

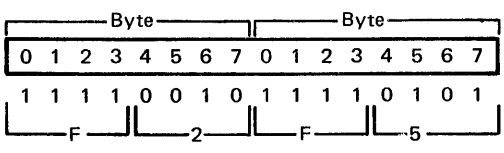
The 16-bit registers and main storage words within the 7200 facilitate the usage of hexadecimal notation. Hexadecimal numbering requires four binary places to represent each of the 16 different symbols within the system: decimal digits 0-9 and letters A-F. Thus, each 16-bit word or register can contain four hexadecimal symbols representing values from zero to $32{,}767_{10}$.

The hexadecimal symbols are also used to represent characters within the Extended Binary-Coded Decimal Interchange Code (EBCDIC). Each EBCDIC character, 8 bits long, comprises two hexadecimal symbols. Although 8 bits provide 256 possible variations, not all are used. The EBCDIC characters do represent the digits *0-9*, letters of the alphabet, all punctuation and arithmetic symbols (such as + - ;>=), and special graphic and control symbols as shown in Table 2-1.

Each EBCDIC character (letter, digit or symbol) is represented in one byte (two hex symbols) as shown below.



Since the decimal digits *0-9* can be represented in four bits and each EBCDIC character requires an entire byte, bits 0-3 are always 1111_2 (the hexadecimal character F) for those digits. As an example; the number *25* represented in EBCDIC code occupies two bytes.



The numeric fields in EBCDIC, unpacked decimal fields, require a byte for each decimal digit. In an unpacked decimal field, bits 0-3 are always F_{16} except in the byte containing the least significant decimal digit of the field.

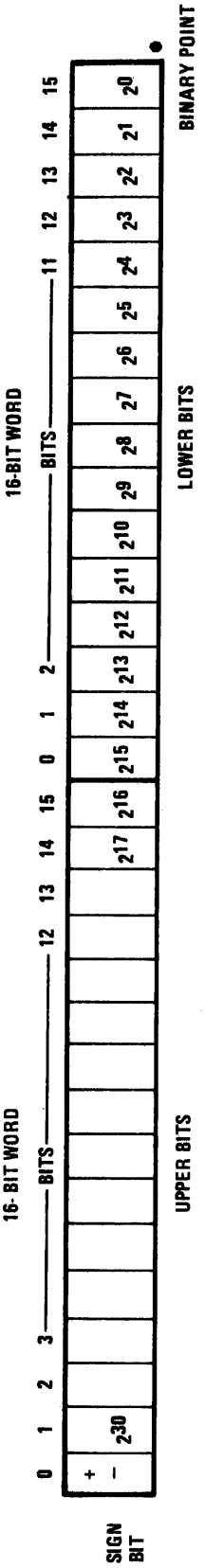


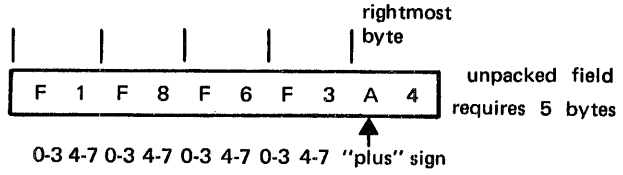
Figure 2-2. Double-Precision Fixed-Point Format

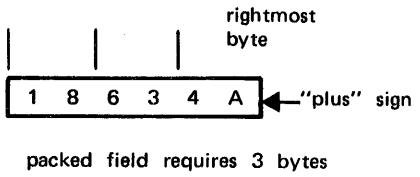
In that byte, bits 0-3 indicate the sign of the entire decimal field. The hexadecimal letters used to indicate the sign are these:

Plus: A,F, C, or E

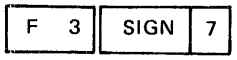
Minus: B or D

An unpacked decimal field can be packed to occupy a fewer number of bytes, by removing bits 0-3 of each EBCDIC digit except in the one containing the sign. The sign character is moved to the rightmost byte of the new packed field. The example below shows the number $+18{,}634_{10}$ in an unpacked decimal field and then in a packed decimal field.

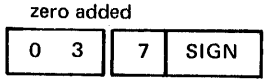




If an unpacked field with an odd number of digits is packed, no portion of the resulting packed field is unused. If a even-number unpacked field is packed, the leftmost four bits of the resulting packed field are zeros.



Unpacked field, even number of digits



packed field, zero added to fill out the byte

Table 2-1. EBCDIC Character Codes
(Bits 4-7)

(Bits 0-3)	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 0010	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 0100		/										,	%	—		?
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						

3. INSTRUCTION TYPES

Every instruction consists minimally of a two-byte (16-bit) base instruction word. This base instruction word comprises two parts: an 8-bit operation code which tells the computer what to do with data; and two fields that specify, in general, the sending and receiving locations of the operand or operands used in the instruction.

Often, the base instruction word is not sufficient to provide all the information needed for certain operations. In such instances, it is augmented by adding a second, or a third, and sometimes even a fourth word. The repertoire, then, comprises 2-byte, 4-byte, 6-byte, and 8-byte instructions (two bytes in a word). This is the first consideration in determining the instruction type.

An instruction operand may be located in one of three places: in the instruction itself ("immediate" operand), in a register ("register" operand), or in main storage ("memory" operand). Operand location provides the second factor to be considered when determining the type of an instruction. Considering byte length and operand location, the repertoire may be divided into six basic instruction types:

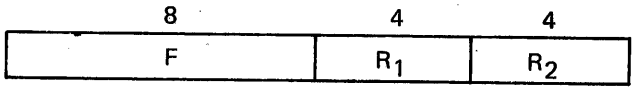
1. Register/Register (2-byte)
2. Immediate/Register (2-byte)
3. Memory/Register (4-byte)
4. Direct/Register (4-byte)
5. Memory/Memory (6-byte)
6. Memory/Memory, extended (8-byte)

These six types are explained below.

GENERALIZED INSTRUCTION FORMATS

Although a variety of formats are used to express the 159† instructions in the repertoire, better than 80% of them fall into one of the six types listed above. The following delineates the formats for these six basic types.

1. Register/Register (2-byte instruction)

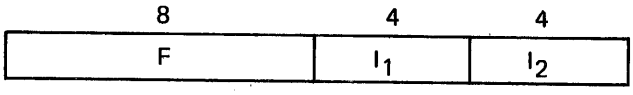


F: The basic operation code

R_1: A field specifying the location of the first operand. (A register number.)

R_2: A field usually specifying the location of the results of the operation. If the operation involves two operands, this also specifies the location of the second operand.

2. Immediate (2-byte instruction)

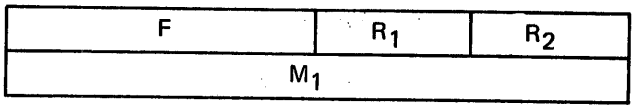


F: The basic operation code

I_1: A 4-bit quantity, the absolute value of which is used as the first operand.

R_2: A field usually specifying the location of the results of the operation. If the operation involves two operands, this also specifies the location of the second operand.

3. Memory/Register (4-byte instruction)



F: The basic operation code

M_1: An address specifying the location of the first operand.

†Does not include 10 for the floating-point option.

R_1: A register number, the contents of which may be used to modify the first operand address (see Indexing).

R_2: A field usually specifying the location of the results of the operation. If the operation involves two operands, this also specifies the location of the second operand.

4. Direct (4-byte instruction)

F	R_1	R_2
I_1		

F: The basic operation code

I_1: A 16-bit quantity, the absolute value of which is used as the first operand.

R_1: A register number, the contents of which may be used to modify the first operand (see Indexing).

R_2: A field usually specifying the location of the results of the operation. If the operation involves two operands, this also specifies the location of the second operand.

5. Memory/Memory (6-byte instruction)

F	R_1	R_2
M_1		
M_2		

F: The basic operation code

M_1: The first operand address

M_2: The second operand address; usually specifies the location of the results of the operation.

R_1, R_2: Two register numbers, the respective contents of which may be used to modify the first operand address and the second operand address (see Indexing).

6. Memory/Memory, extended (8-byte instruction)

F	R_1	R_2
M_1		
M_2		
L_1	L_2	

This instruction type deals with fields of operands, rather than single operands. F, R_1, R_2, M_1, and M_2 are as explained under 5, above. The field length, in bytes, of the first operand is given by L_1; that of the second operand is given by L_2.

As stated above, the foregoing are generalized formats only. Individual differences will be detailed in the instruction descriptions.

ADDRESSING MODES

Three addressing modes are available by which to obtain the operands used by an instruction: immediate, direct, and indirect.

IMMEDIATE ADDRESSING

Immediate addressing covers those cases where the operand is contained in the instruction word itself. Examples of "immediate" operands are:

- raw data
- shift count
- skip count
- a hexadecimal value indicating a bit position (in a register) that is to be tested, set, or cleared.
- an External Register number
- a processor number

DIRECT ADDRESSING

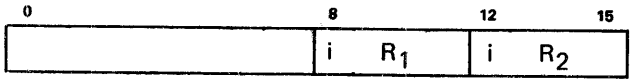
Direct addressing means that the related field of the instruction (R_1 or M_1; R_2 or M_2) specifies an address at which the operand may be found; the instruction specifies the location of the operand **directly.**

INDIRECT ADDRESSING

Indirect addressing means that the related field of the instruction (R_1 or M_1; R_2 or M_2) specifies a location that contains the **address** of the operand. That is, the instruction specifies the location of the operand **indirectly.**

IMPLEMENTATION

Direct and indirect addressing are indicated by the highest-order bits (designated "i") in the two R fields of the base instruction word.



For example, if bit 8 is 0, the register indicated by the lower three bits of the R_1 field (or the 16-bit address stored in the M_1 field if this is a multi-word instruction) contains the operand. On the other hand, if bit 8 is 1, the indicated register contains the operand address; if this were a multi-word instruction, M_1 would contain a 16-bit memory location the **contents** of which would stipulate the address of the operand. These rules apply equally to the bit-12 address-mode designator.

Direct addresses may be even or odd, depending upon the instruction. Indirect addresses must always be even; whether the final operand address, located at the "indirect" address, is even or odd depends upon the instruction.

Figures 3-1 through 3-5 show the addressing schemes available for each instruction type, using all possible combinations of the two 'i' designators. Note that as each format is titled, the basic instruction type name is manifested for the case where both bit 8 and bit 12 are 0. In each example, when bit 8 or bit 12 is 1, the additional manipulation of indirect addressing is required.

These figures do not give the complete addressing story, for there are still the indexing (or address modification) situations to consider. This is the subject of the next section, Indexing.

INDEXING

Indexing consists of adding to the value of an instruction's *M* field (or the contents of the address specified therein) the contents of the register indicated in the corresponding *R* field of the base instruction word. Indexing cannot apply to the 2-byte instructions because those instructions do not have an *M* field.

Indexing will occur in an instruction whenever the *R* field for the indexable *M* address contains a register number other than zero (that is, 1-7). If the *R* field (bits 9-11 for R_1; bits 13-15 for R_2) contains a zero, no indexing for the related *M* address field is performed.

The terms "pre-indexing" and "post-indexing" define at what point in the instruction execution the indexing occurs. If the register contents are added to the value of the *M* field, it is called pre-indexing. If the register contents are added to the **contents** of the address specified by the M field, it is called post-indexing.

In the instruction repertoire, only one instruction *(BCH)* uses pre-indexing. Figure 3-6 shows how pre-indexing applies to that instruction. Figure 3-7 shows a similar instruction, *B,* using post-indexing. Figure 3-8 uses one of the previously defined examples to show another example of post-indexing. In the *Machine Instructions* section, indexing is always indicated when it is valid. Except for the *BCH* instruction, which uses pre-indexing, post-indexing is assumed when indexing is indicated.

It should be pointed out that, as the examples show, the two indexing methods differ only for the indirect addressing mode. That is, post-indexing must operate on the contents of the *M* field for the direct addressing mode (inasmuch as the "indirect" address doesn't exist), and in this respect is identical to pre-indexing.

The foregoing discussion has dwelt on address modification. But indexing may also be used to modify an operand. This fact bears repeating, for although indexing most frequently refers to address modification, operand modification is possible — albeit only for the second operand of direct-type instructions as evidenced previously in Item 4, under *Generalized Instruction Formats.*

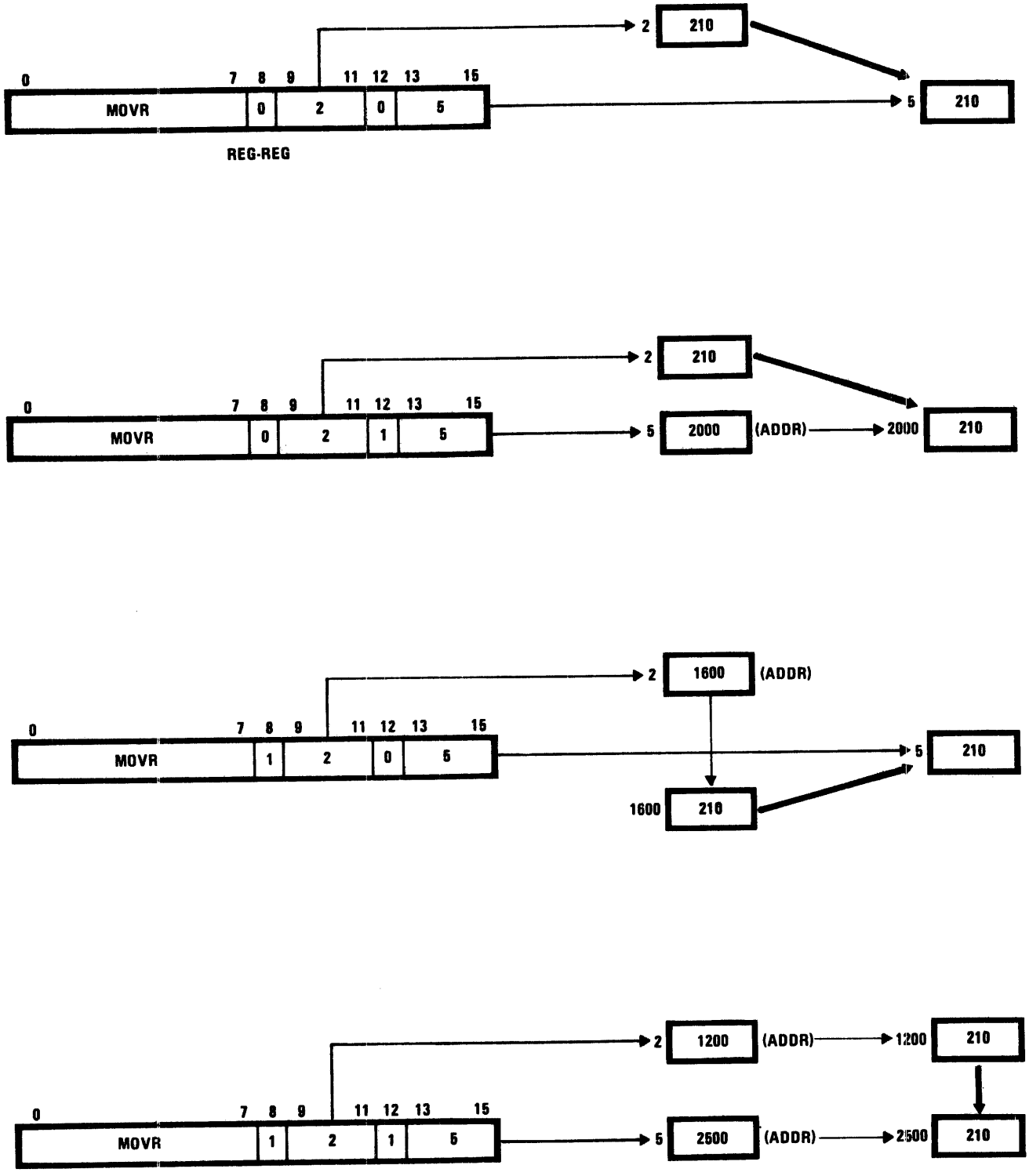


Figure 3-1. Register-to-Register Instruction Addressing

An important indexing consideration is this possibility: Indexing can attempt to produce an operand address greater than the 16-bit capacity of the adder. This condition occurs when the operand address, *M*, and the modifying value in the indexing register result in a overflow sum greater than $65{,}535_{10}$ or $FFFF_{16}$. If such happens, significance beyond 16 bits is lost; the remaining 16-bit sum is the operand address utilized by the instruction.

M Address	48,391 =		1011110100000111
Modifying Value	26,124 =		0110011000001100
Attempted Operand Address	74,515	1	0010001100010011
		Lost Bit	resulting operand address = 8979

Of course, this condition can also occur when indexing an operand itself (the *I* field) rather than the operand address. No error indication is recorded when the above condition occurs.

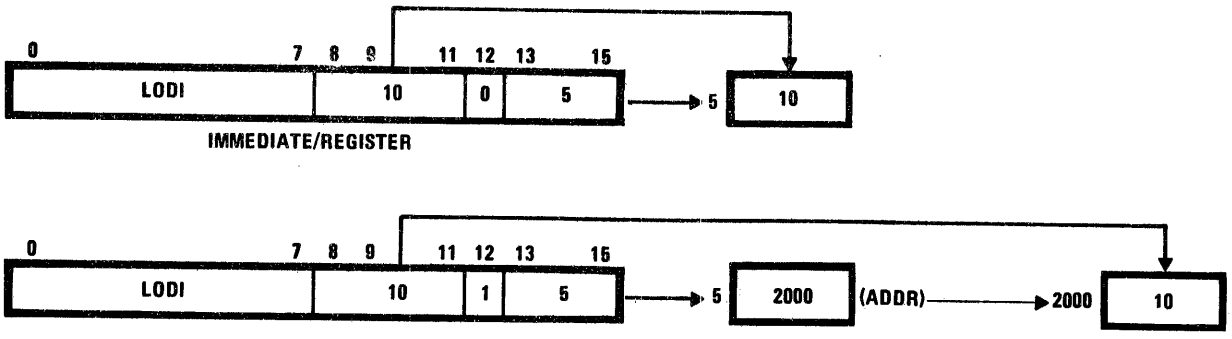
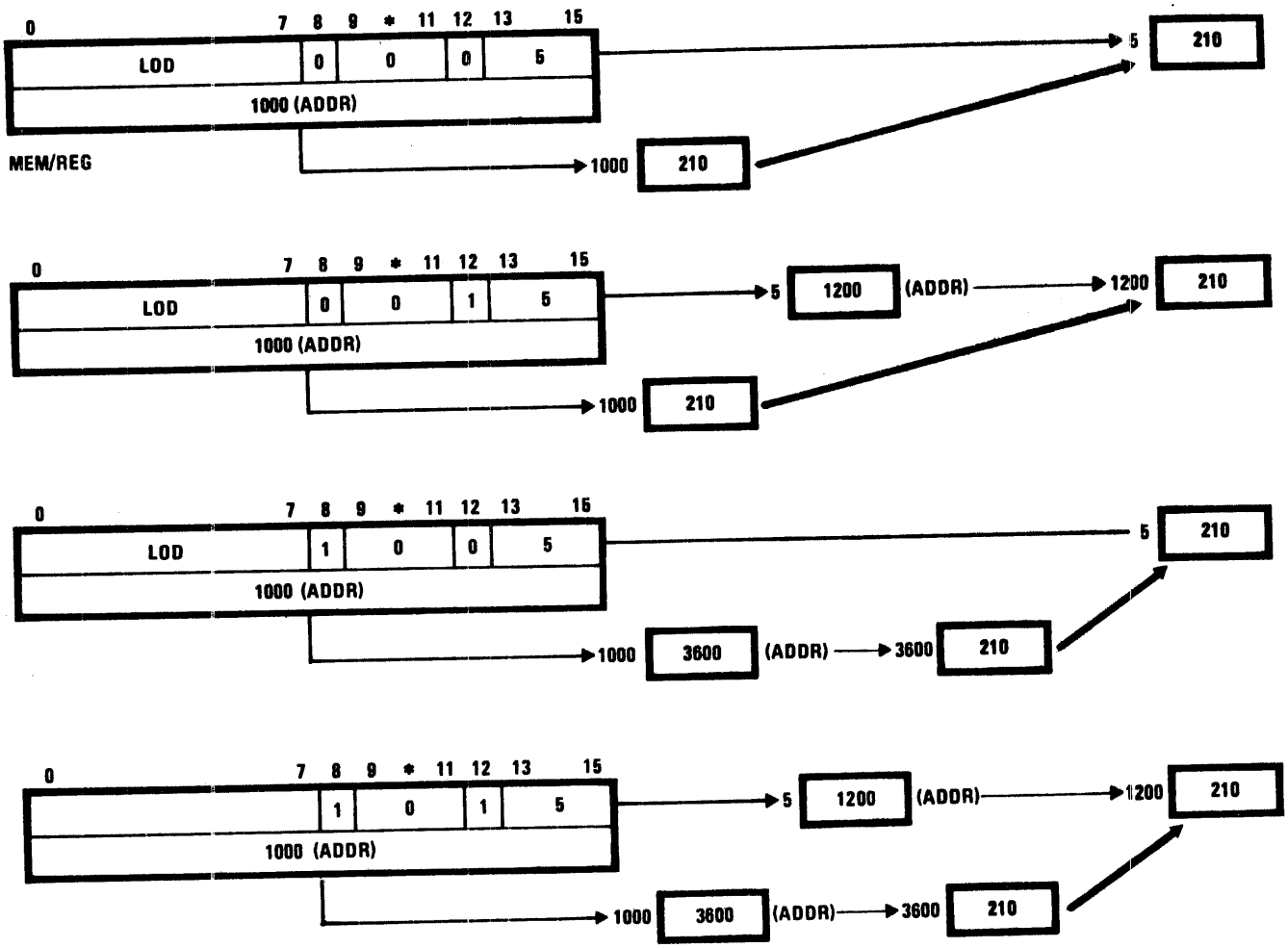
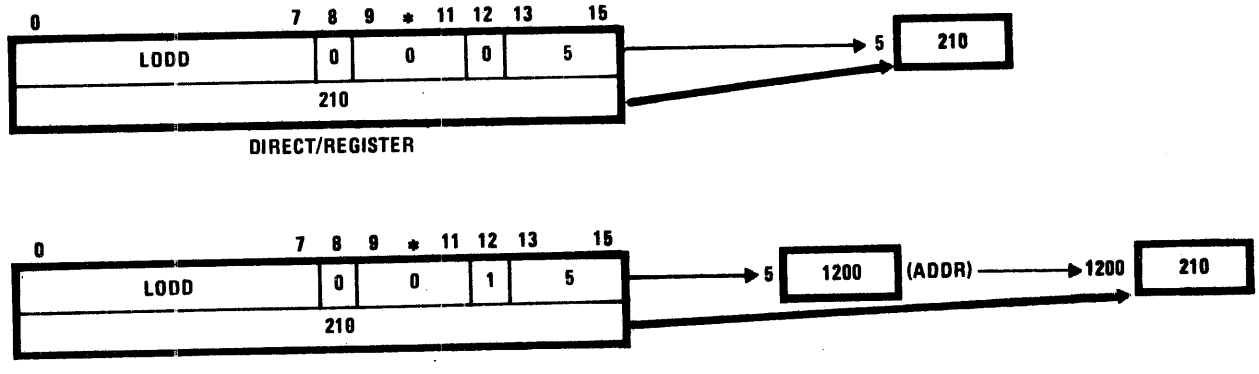


Figure 3-2. Immediate-Register Instruction Addressing



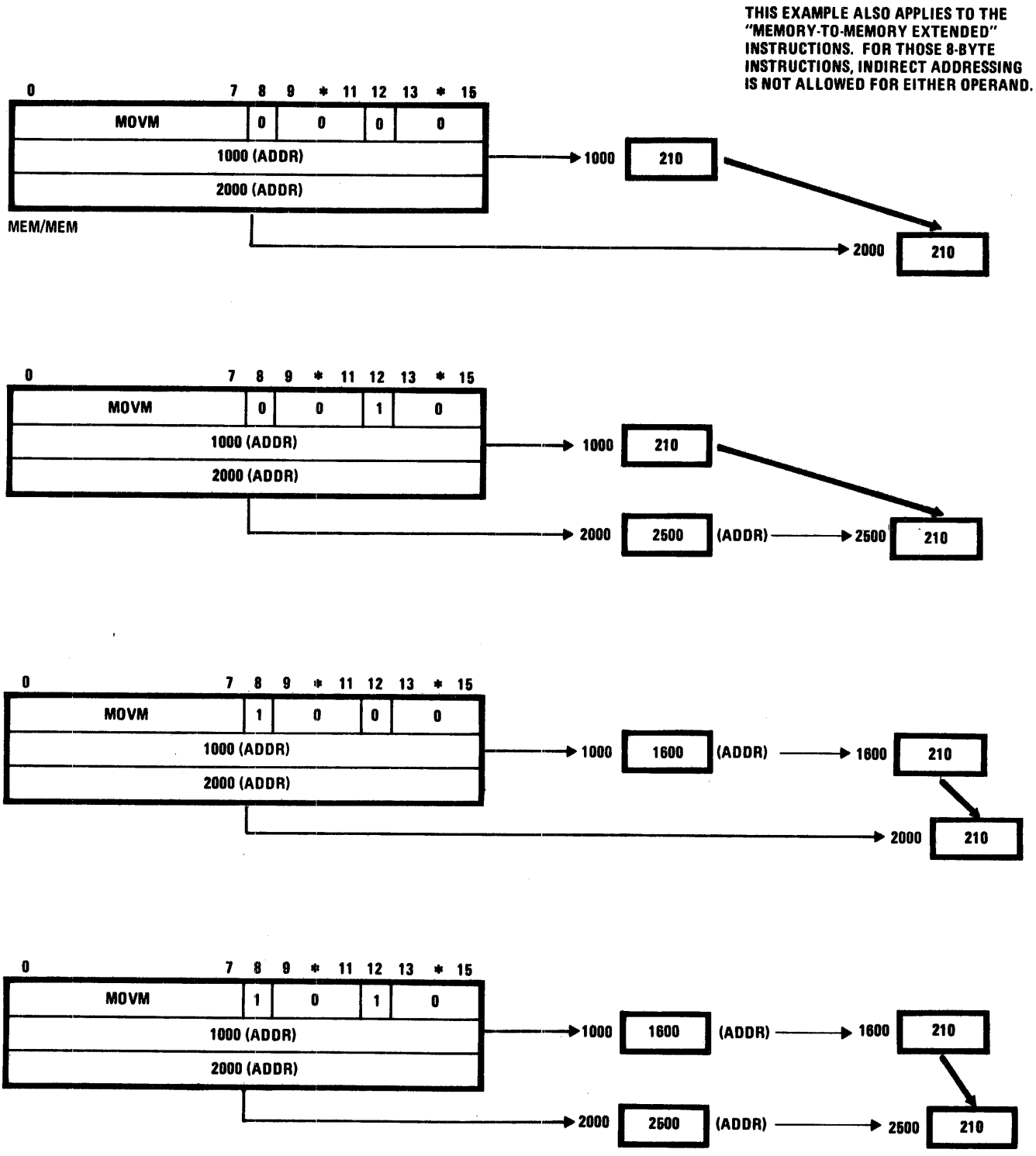
*SEE SECTION ON INDEXING FOR CASES WHERE $R_1 \neq 0$

Figure 3-3. Memory-to-Register Instruction Addressing



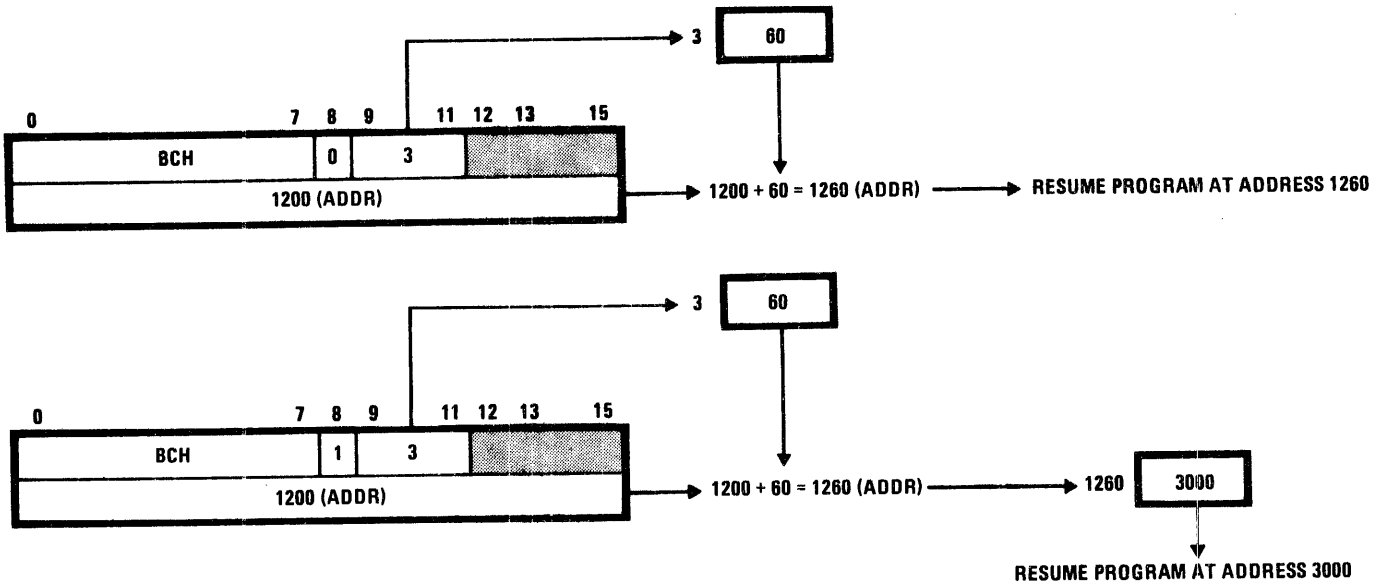
*SEE SECTION ON INDEXING FOR CASES WHERE R1 ≠ 0

Figure 3-4. Direct-Register Instruction Addressing



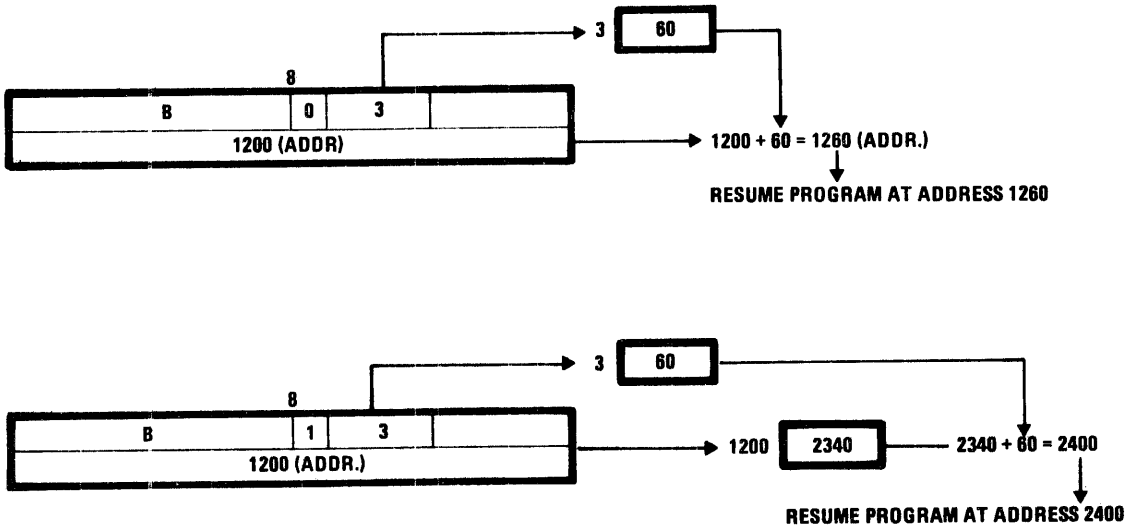
*SEE SECTION ON INDEXING FOR CASES WHERE $R_1, R_2 \neq 0$

Figure 3-5. Memory-to-Memory Instruction Addressing



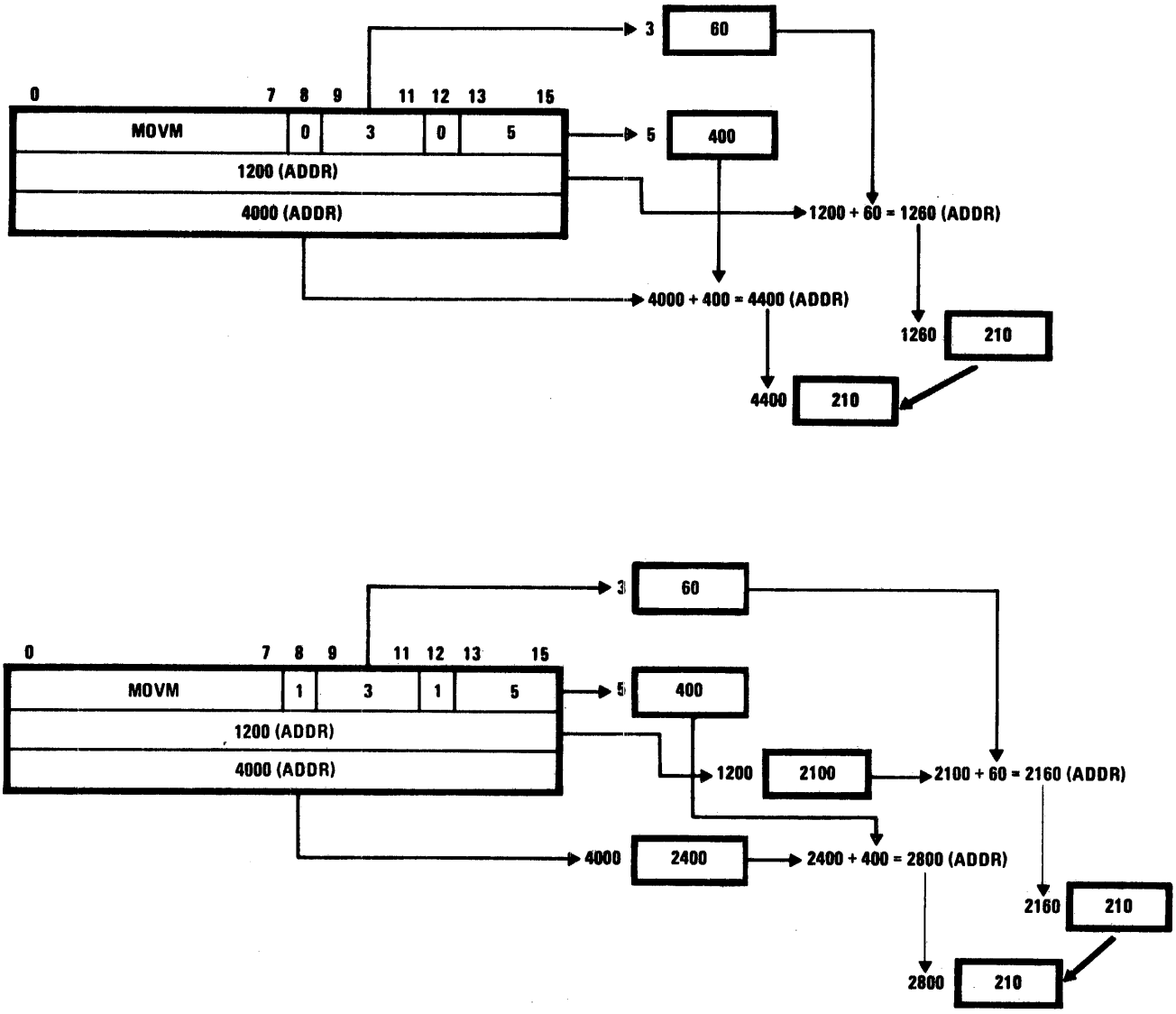
The BCH instruction causes an unconditional jump (or branch) to the address stipulated. If the R_1 field were 0, 1200 would be the new address (or, if Bit 8 were a 1, the new address would be stipulated by the contents of location 1200). Because R_1 is not 0, the contents of the indicated register are added to the value of the M-field. The state of Bit 8 then indicates whether or not an additional storage reference is needed to obtain the final "jump to" address.

Figure 3-6. Pre-Indexing With BCH Instruction



THE B INSTRUCTION ALSO PERFORMS AN UNCONDITIONAL JUMP, BUT USES POST-INDEXING. NOTE THAT B AND BCH (FIGURE 3-6) PERFORM IDENTICALLY WHEN BIT 8 IN BOTH INSTRUCTIONS IS A 0 (DIRECT ADDRESSING). THE INSTRUCTIONS DIFFER, HOWEVER, WHEN BIT 8 IN BOTH IS A 1 (INDIRECT ADDRESSING).

Figure 3-7. Post-Indexing With B Instruction



FOR OTHER VALUES OF THE ADDRESS-MODE DESIGNATORS (BITS 8 AND 12), COMBINE THE APPROPRIATE PARTS OF THESE EXAMPLES.

Figure 3-8. Memory-to-Memory Instruction With Post-Indexing

4. MACHINE INSTRUCTIONS

INTRODUCTION

The MRX/40/50 System machine instruction set is divided into two major categories: general-purpose instructions and system instructions. General-purpose instructions are the instructions needed to solve most data processing problems using a defined software system. System instructions are specialized instructions used to interpret and alter a software system.

Within these two major categories, the instructions are divided into functional groups, and these functional groups are listed in alphabetical order, as shown in the following table.

General-Purpose Instructions	System Instructions
Arithmetic	Control
Bit-Oriented	I/O
Boolean Logic	
Branching	
Compare	
Control	
Data Conversion	
Data Transfer	
Shift	
Optional: Floating Point	

The instructions in each functional group are listed alphabetically by mnemonic. This rule holds for all instructions except for logical pairs or groups of instructions — these instructions are listed alphabetically according to the first instruction of the pair. For instance, PAKX (Pack) will be followed by UNPX (Unpack), and SB (Skip Back Unconditional) will be followed by SF (Skip Forward Unconditonal).

Remember the following rules when reading the machine instruction descriptions.

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

2. A word is defined as two bytes; the bit positions in a word are numbered left to right, 0-15.

3. 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 0-65,535 bytes.

4. 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 e).

5. The effective address of a field in memory is defined as the final address of the field derived from all specified addressing techniques. If no optional addressing techniques are used, the effective address is in the M field, otherwise, the effective address is a result of indexing, indirect addressing, or both.

SOURCE AND OBJECT FORMAT INTERPRETATION

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 (0-7), or as an index register (1-7 only).
E	Extended register,0-15. (For RDX and WRX only.)
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 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. A source operand using both an indexing and a field length specification would be represented like this: $M_1(L_1,R_1)$. The comma in the parentheses must not only be coded when both the length and index register are used, but also if either one of them is used, as follows: $M_1(L_1,)$ or $M_1(,R_1)$. This enables the assembler to distinguish between the two specifications in parentheses.

Bits 8 and 12 of the object instructions are used in almost every instruction to convey information to the computer concerning that instruction. If these bits are not interpreted in any way, they are shaded; otherwise, the following symbols are used to define bits 8 and 12.

i — Indirect addressing indicator; for direct addressing i=0, for indirect addressing i=1. Indirect addressing is indicated by the programmer.

f — A sub-function indicator; indicates a function that the operation code alone cannot do. Function bits are set by the assembler.

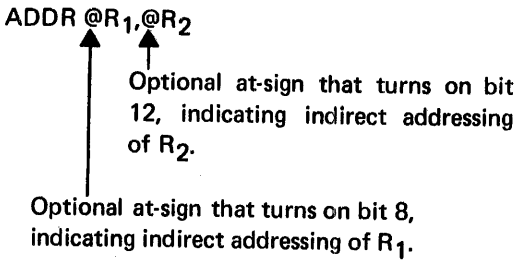
1,0 — If bit 8 or 12 must be a 1 or a 0 for a particular instruction, the bit will be shown as a 1 or 0. These bits are set by the assembler; if the wrong bit state appears in the object instruction, a no-operation occurs.

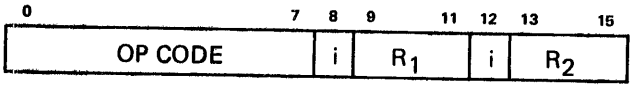
An R, M, I, or L in source operand 1 is identified as R_1, M_1, I_1, or L_1; in source operand 2 they are identified as R_2, M_2, I_2, or L_2. These symbols are numbered so that they can be referred to easily (distinguishing between R_1 and R_2 in the same instruction) and to make clear the location of these fields in the object format.

The two major operand fields must be separated by a comma; no blanks are allowed anywhere in the operand fields.

The following examples show how the source and object formats of the machine instructions are illustrated in this section. The at-sign and any designations in parentheses (field length and index registers) 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 stated otherwise.

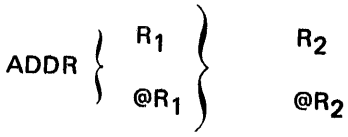
EXAMPLE 1: Add Register-Register





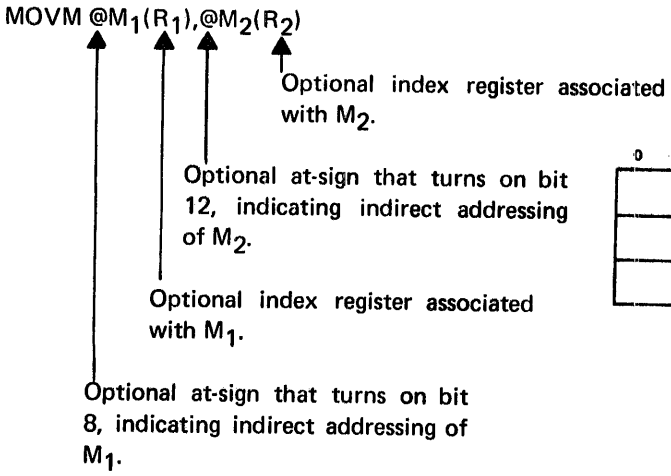
R_1 and R_2 are in the same relative position in the object format.

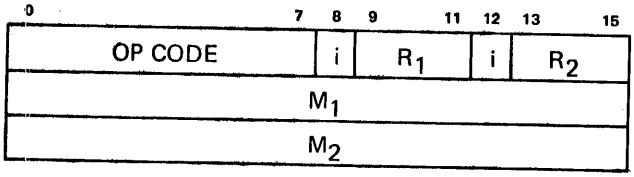
The variations of this instruction are shown in the following diagram; depending on the use of indirect addressing, data flow may be register to register, memory to register, register to memory, or memory to memory.



Since each operand can be expressed either of two ways, there are 2 x 2 = 4 possible variations of this instruction.

EXAMPLE 2: Move Memory-Memory





The variations of this instruction are shown in the following diagram. Data flow is always memory to memory, but there are many ways in which the addressing options can be used.

Since each operand can be expressed either of four ways, there are 4 x 4 = 16 variations of this instruction.

GENERAL-PURPOSE INSTRUCTIONS

ARITHMETIC INSTRUCTIONS

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

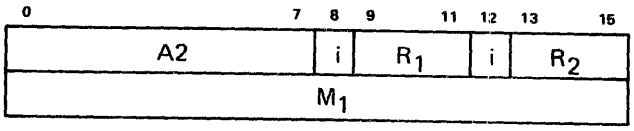
The following rules apply to binary addition and subtraction.

- 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 the results are greater than $2^{n-1}-1$ or if the results are less than -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 the results are greater than $2^{n}-1$ for an add (where n=16 or 32 bits).

Add Memory – Register

ADD @M_1(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.

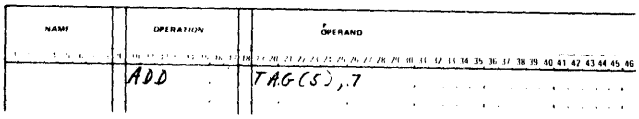
OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

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

RESULTS: The 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.

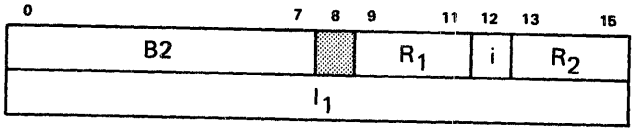
EXAMPLE



The field identified by TAG(5) is added to the contents of register 7; the sum will be in register 7.

Add Direct

ADDD I_1(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 the second word of the instruction; the value may range from -32,768 to +32,767.

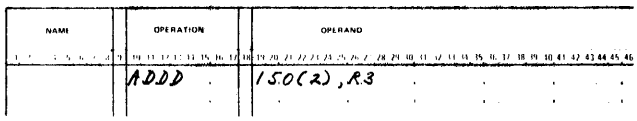
Indexing may be specified for operand 1. In this case, the value of operand 1 is derived by adding the I_1 value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

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

RESULTS: The 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

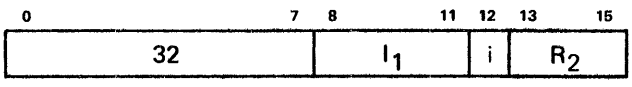


The immediate value of 150 is modified by the contents of register 2 and added to the contents of register 3; the sum will be in register 3.

If register 2 contains a value of 4, and register 3 contains a value of 10, the operation will add 150 + 4 + 10; the result of 164 will be in register 3.

Add Immediate

ADDI I_1,@R_2



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

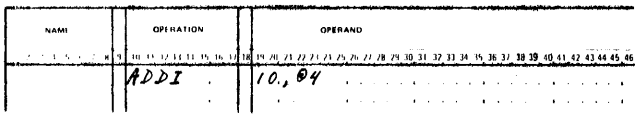
OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the I_1 value may range from 0-15. The I_1 value is added to bit positions 12-15 of operand 2; bits 0-11 are zeros.

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

RESULTS: The 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.

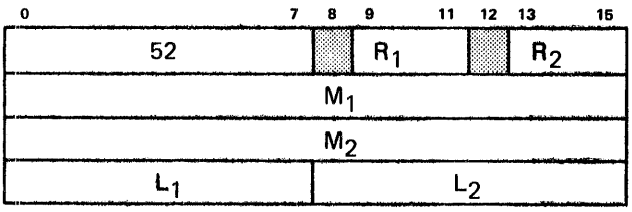
EXAMPLE



The immediate value of 10 is added to the field at the memory address specified in register 4; the sum will be at this address.

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, 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 but indirect addressing is not allowed. The effective operand address points to the most significant bytes of the decimal field.

RESULTS: The 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_2 and L_1 contains significant data, bit 0 of the Condition register is set.
- If L_1 = 0 and L_2 = 0, bits 3 and 7 of the Condition register are 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.

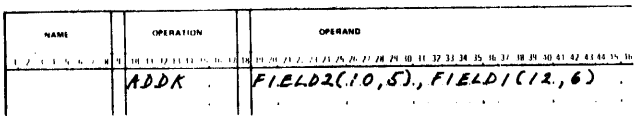
The Condition register is affected as follows:

- 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, 6 and 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.

SPECIAL FEATURES

1. Validity of source packed digits is not checked.
2. Invalid digits produce inconsistent results.
3. Negative zero cannot be produced unless overflow occurs (bit 0 of Condition register set).
4. Positive results receive a hexadecimal C sign, negative results receive a hexadecimal D sign.
5. The effective addresses must not be absolute address zero on machines with less than 65K, or they must not be the first location of the memory partition if the Relocation and Protection feature is installed and READ protection is invoked.

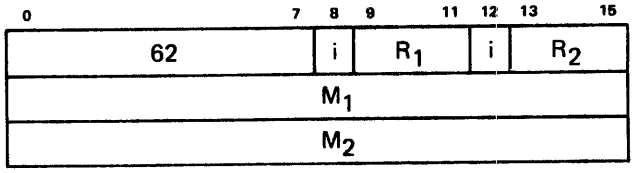
EXAMPLE



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.

Add Memory – Memory

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



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

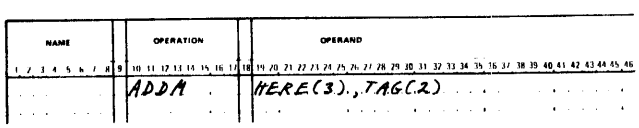
OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: Same length and addressing options (to base M_2) as operand 1.

RESULTS: The 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.

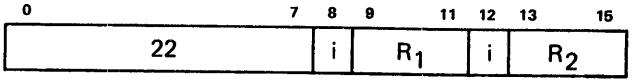
EXAMPLE



The field identified by HERE(3) is added to the field identified by TAG(2); the sum will be at the address represented by TAG(2).

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.

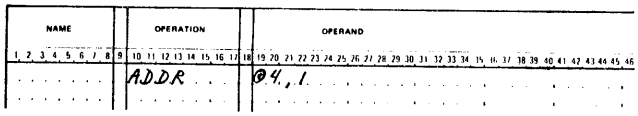
OPERAND 1: A one-word field in the general register specified by R_1, or in memory if indirect addressing is used.

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

RESULTS: The 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.

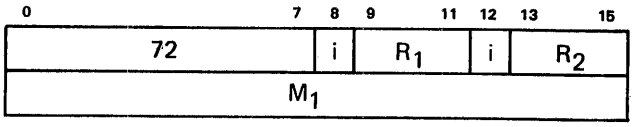
EXAMPLE



The operand at the memory address specified in register 4 is added to the contents of register 1; the sum will be in register 1.

Add Two-Word

ADDT @M_1(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 beginning at the specified effective address. The most significant bits are at this address.

Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A two-word field located in two general registers or in memory.

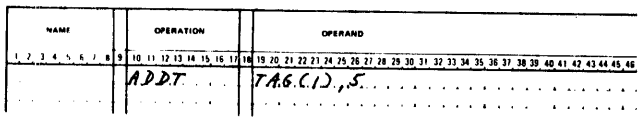
If direct addressing is used, the field is in the register specified by R_2 and the next highest register, R_2+1; the most significant bits are in the R_2 register. (Note: If register 7 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at the address in the R_2 register; the most significant bits are at this address.

RESULTS: The 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.

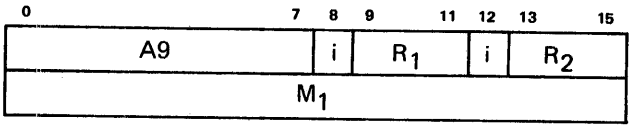
EXAMPLE:



The field identified by TAG(1) is added to the contents of registers 5 and 6; the results will be placed in registers 5 and 6.

Divide Memory – Register

DIV @M_1(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 memory beginning at the specified effective address. The most significant bits are at this address.

Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The dividend; a 32-bit signed value located in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

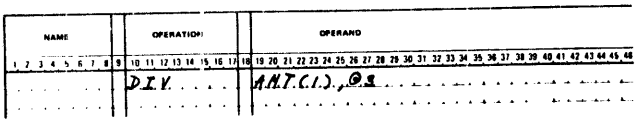
If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

RESULTS: The quotient, a 16-bit signed value, resides in the R_2 register (direct addressing), or at the address specified in the R_2 register (indirect addressing). The remainder, a 16-bit signed value resides in the R_2-1 register (direct addressing) or in memory beginning at an address 2 bytes less than the address specified in the R_2 register (indirect addressing); the remainder always has the same sign as the dividend.

The Condition register is affected as follows:

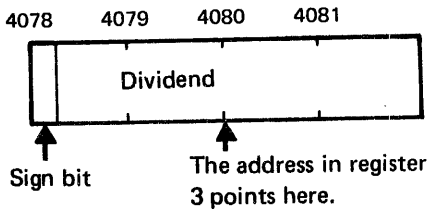
- Bit 0 (overflow) is set if the resulting quotient is greater than +32,767 or less than -32,768.
- Bit 0 (overflow) is set if the divisor is 0; the operands are unchanged.
- If neither of the above conditions occurs, bit 0 is cleared.

EXAMPLE

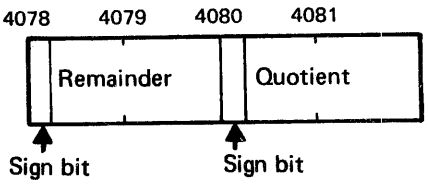


The instruction specifies a binary divide in which the divisor is a 16-bit signed value at the address specified by AMT(1); the dividend is located in memory beginning at an address two bytes less than the address in register 3.

As shown in the following illustration, the dividend is a 32-bit field located at an address two bytes less than the address specified in register 3. Assuming that the address in register 3 is 4080, the dividend actually begins at address 4078.

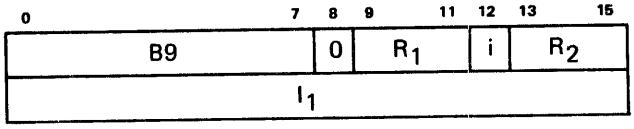


After the division, the dividend is overlayed with the remainder and quotient (each of these fields is signed). Thus, the quotient is located at the address specified in the R_2 register — in this case at address 4080, and the remainder at that address minus two bytes, 4078.



Divide Direct

DIVD $I_1(R_1),@R_2$



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 the second word of the instruction; the I_1 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_1 value and the contents of the general register specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: The dividend; a 32-bit signed value in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

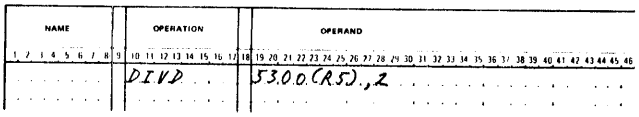
If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

RESULTS: The quotient, a 16-bit signed value, resides either in the R_2 register (direct addressing), or in memory at the address in the R_2 register (indirect addressing). The remainder, a 16-bit signed value, resides in the R_2-1 register (direct addressing), or in memory at an address two bytes less than the address in the R_2 register (indirect addressing); 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.
- If neither of the above conditions occurs, bit 0 is cleared.

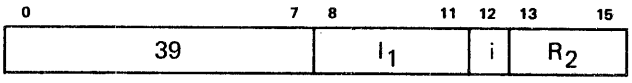
EXAMPLE



The divisor is formed by adding the immediate value of 5300 to the contents of register 5. The dividend is in registers 1 and 2. The quotient will be in register 2, the remainder in register 1.

Divide Immediate

DIVI I_1,@R_2



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 I_1 value is always positive and may range from 0-15.

OPERAND 2: The dividend; a 32-bit signed value located in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

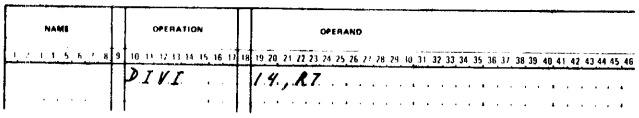
If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

RESULTS: The quotient, a 16-bit signed value, resides in the R_2 register (direct addressing), or in memory at the address specified in the R_2 register (indirect addressing). The remainder, a 16-bit signed value, resides in the R_2-1 register (direct addressing), or in memory at an address two bytes less than the address in the R_2 register (indirect addressing); 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.
- If neither of the above conditions occurs, bit 0 is cleared.

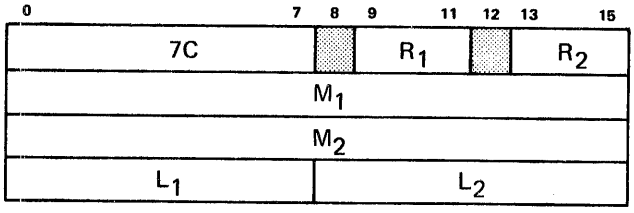
EXAMPLE



The divisor is the immediate value 14; the dividend is in registers 6 and 7. The quotient will be in register 7, and the remainder in register 6.

Divide Packed Decimal •

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



FUNCTION: Divides packed decimal fields in memory. The field lengths may vary, as defined in the following text.

OPERAND 1: The divisor; a packed decimal field in memory that can range from 1-127 bytes. The length of the divisor must always be less than that of the dividend. If the L_1 value is greater than 127, or greater than or equal to L_2, the operation will not be performed, and an overflow condition occurs. The results occur if the divisor is zero, or the L_1 value is zero.

OPERAND 2: The dividend; a packed decimal field in memory that can range from 2-255 bytes. The dividend field is overlaid by the quotient and remainder.

RESULTS: The signed quotient will be located at the operand 2 address; thus the address of the quotient will be the same as that of the dividend. The length of the quotient is L_2-L_1. The signed remainder is also placed in the operand 2 location, but it is right-aligned. The address of the remainder is: quotient address + (L_2-L_1). The length of the remainder is the same as the length of the divisor.

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 sign conventions apply (the numbers are hexadecimal).

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

Minus = 1,3,5,9,B,D

The preferred signs of X'C' for plus and X'D' for minus will be generated. A minus zero quotient or remainder is considered plus.

The operand fields remain unchanged if overflow occurs.

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, $L_1 \geq 127$; $L_1 > L_2$; $L_1 = 0$; or, divisor field contents are 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.

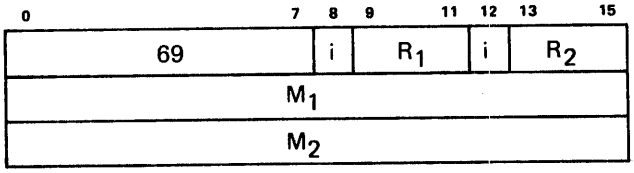
EXAMPLE

NAME	OPERATION	OPERAND
	DIVK	UNITS(4,1),TOTAL(8,1)

The TOTAL field is divided by the UNITS field. The quotient will be located at the address of TOTAL (the quotient length is 4 bytes). The remainder will be at the address TOTAL +4.

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, indirect addressing, or a combination of both.

OPERAND 2: The dividend; a 32-bit signed value in memory beginning at an address two bytes less than the specified effective address; the most significant bits are at this beginning address. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

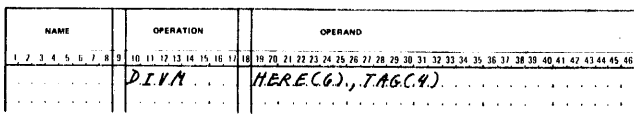
RESULTS: The quotient, a 16-bit signed value, resides at the effective address of operand 2. The remainder, a 16-bit signed value, resides at an address two bytes less than the effective address. The remainder always has the same sign as the dividend.

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 zero; the operands are unchanged.
- If neither of the above conditions exist, bit 0 is cleared.

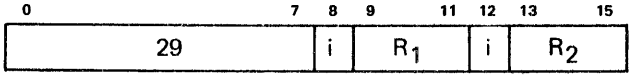
EXAMPLE



The dividend is identified by TAG(4) and the divisor is identified by HERE(6). The quotient will be at the address specified by TAG(4); the remainder will be at that address minus two bytes.

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.

OPERAND 2: The dividend; a 32-bit signed value located in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

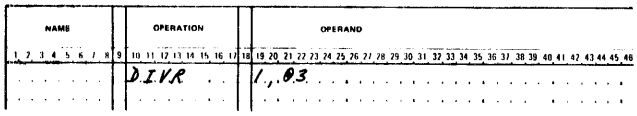
If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

RESULTS: The quotient, a 16-bit signed value, resides in the R_2 register (direct addressing), or at the address specified in the R_2 register (indirect addressing). The remainder, a 16-bit signed value, resides in the R_2-1 register (direct addressing) or in memory at an address two bytes less than the address specified in the R_2 register (indirect addressing); 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.
- If neither of the above conditions occurs, bit 0 is cleared.

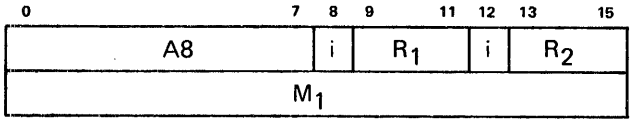
EXAMPLE



The least significant bits of the dividend are at the address specified in register 3, and the divisor is in register 1. The quotient will be stored at the address in register 3; the remainder is stored at that address minus two bytes.

Multiply Memory — Register

MPY @M_1(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_1 include indexing, indirect addressing, or a combination of both.

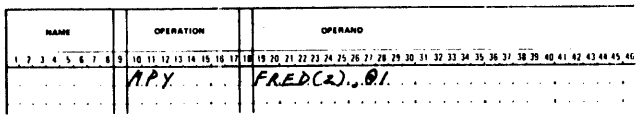
OPERAND 2: The multiplicand; a one-word field in the general register specified by R_2, or in memory if indirect addressing is used.

RESULTS: The product, a 32-bit signed value, resides in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

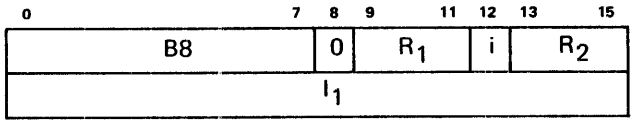
EXAMPLE



The field at the address specified in register 1 is multiplied by the field identified by @FRED(2). The 32-bit result field will be in memory at an address two bytes less than the address specified in register 1.

Multiply Direct

MPYD $I_1(R_1),@R_2$



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 the second word of the instruction; the I_1 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_1 value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

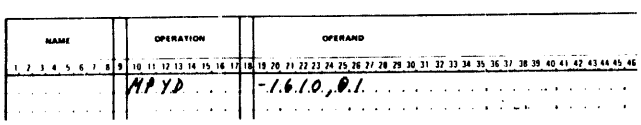
OPERAND 2: The multiplicand; a one-word field in the general register specified by R_2, or in memory if indirect addressing is used.

RESULTS: The product, a 32-bit signed value, resides in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2 the field is in registers 7 and 0, with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

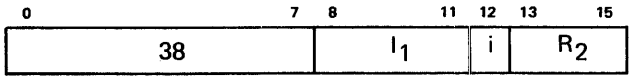
EXAMPLE:



The value at the address in register 1 is multiplied by the immediate value -16,101. The 32-bit product is stored at the address in register 1 minus two bytes.

Multiply Immediate

MPYI $I_1,@R_2$



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 I_1 value is always positive and may range from 0-15.

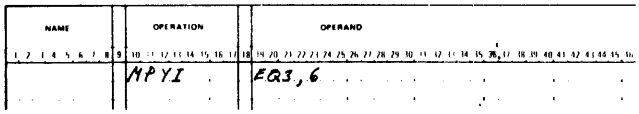
OPERAND 2: The multiplicand; a one-word field in the general register specified by R_2, or in memory if indirect addressing is used.

RESULTS: The product, a 32-bit signed value, resides in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

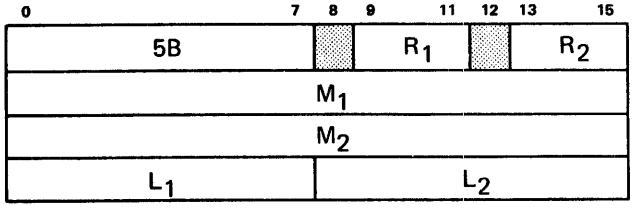
EXAMPLE



The value in register 6 is multiplied by the immediate value represented by EQ3. The product is stored in registers 5 and 6.

Multiply Packed Decimal •

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



FUNCTION: Multiplies two packed decimal fields in memory. The field lengths may vary, as defined in the following text.

OPERAND 1: The multiplier; a packed decimal field in memory that can range from 0-127 bytes. The length of the multiplier must *always* be less than that of the multiplicand. If the L_1 value is greater than 127, or greater than or equal to the L_2 value, the operation will not be performed and an overflow condition occurs. If L_1 is zero, and L_2 is not zero, a multiply by zero will be performed.

OPERAND 2: The multiplicand; a packed decimal field in memory that can range from 0-255 bytes. Since the number of digits in the product is the sum of the digits in the operands, the multiplicand must have at least enough high-order zero bytes to equal the multiplier field length; otherwise, an overflow condition occurs. This definition of the multiplicand field ensures that no product overflow can occur during the operation. At least one high-order digit of the product field is always zero.

RESULTS: The product will be placed at the operand 2 address.

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

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

Minus = 1,3,5,9,B,D

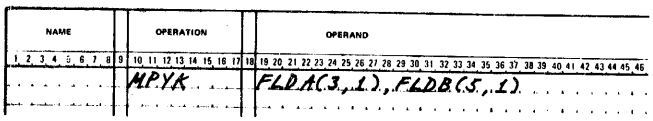
The preferred signs of X'C' for plus and X'D' for negative are generated for the product. A negative zero result is considered plus.

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. 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 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.

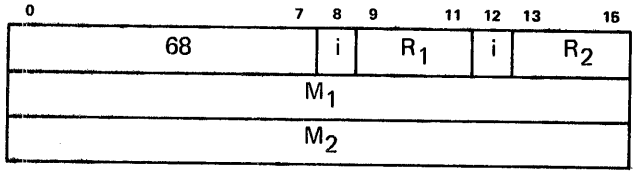
EXAMPLE



FLDA is multiplied by FLDB and the results are located at the FLDB address.

Multiply Memory – Memory

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



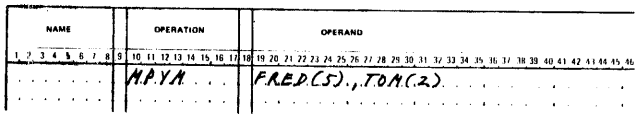
FUNCTION: Performs a binary multiplication of two one-word fields in memory.

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

RESULTS: The product; a 32-bit signed value. The result field begins at an address two bytes less than the specified effective address of operand 2.

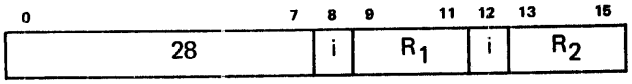
EXAMPLE



The field identified by TOM(2) is multiplied by the field identified by FRED(5). The 32-bit product is stored at an address two bytes less than the address specified by TOM(2).

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.

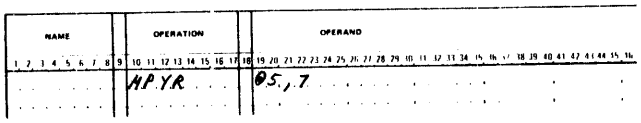
OPERAND 2: The multiplicand; a one-word field in the general register specified by R_1, or in memory if indirect addressing is used.

RESULTS: The product, a 32-bit signed value, resides in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_1 and the next lowest register, R_2-1; the most significant bits are in the R_2-1 register. (Note: If register 0 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at an address two bytes less than the address in the R_2 register; the most significant bits are at this address.

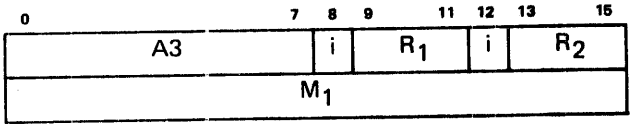
EXAMPLE:



The field in register 7 is multiplied by the field at the address specified in register 5. The 32-bit product is stored in registers 6 and 7.

Subtract Memory – Register

SUB @M_1(R_1),@R_2



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

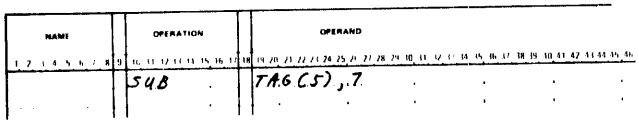
OPERAND 1: The subtrahend; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both. This operand is subtracted from operand 2.

OPERAND 2: The minuend; a one-word field in the general register specified by R_1, or in memory if indirect addressing is used.

RESULTS: The 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.

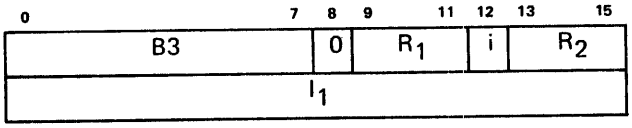
EXAMPLE



The field identified by TAG(5) is subtracted from the contents of register 7. The result will be in register 7.

Subtract Direct

SUBD I_1(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: The subtrahend; a 16-bit immediate signed value in bits 16-31 of the instruction. The I_1 value may range from -32,768 to +32,767. This operand is subtracted from operand 2.

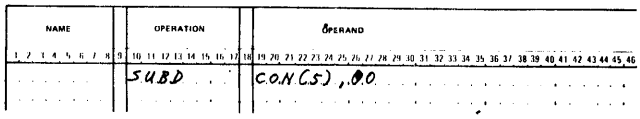
Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I_1 value and the general register contents specified by R_1; no check for overflow or link is made during the indexing.

OPERAND 2: The minuend; a one-word field in the general register specified by R_2 or in memory if indirect addressing is used.

RESULTS: The 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.

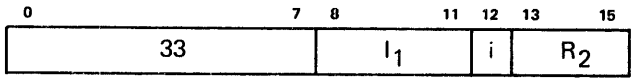
EXAMPLE



The immediate value identified by CON(5) is subtracted from the field in memory at the address specified in register 0. The result will be at the address in register 0.

Subtract Immediate

SUBI I_1,@R_2



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

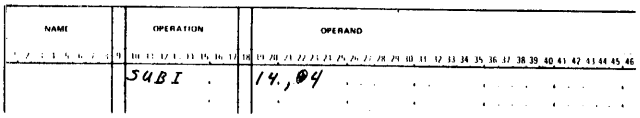
OPERAND 1: The subtrahend; a 4-bit unsigned value located in bits 8-11 of the instruction. This value may range from 0-15. The I_1 value is subtracted from bits 12-15 of operand 2; bits 0-11 of operand 1 are zeros.

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

RESULTS: The 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.

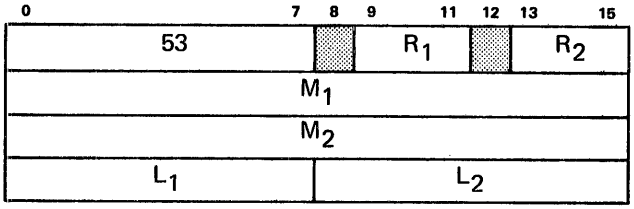
EXAMPLE



The immediate value 14 is subtracted from bits 12-15 of the field at the memory location specified in register 4. The result will be at the address in register 4.

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: The subtrahend; 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 address indicated by M_2 may be indexed, but indirect addressing is not allowed. The effective operand address points to the most significant bytes of the decimal field.

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

RESULTS: The 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, bits 3 and 7 of the Condition register are 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.

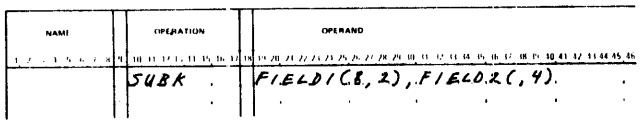
The Condition register is affected as follows:

- 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, 6, and 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.

SPECIAL FEATURES

1. Validity of source digits is not checked.
2. Invalid digits produce inconsistent results.
3. Negative zero cannot be produced unless overflow occurs (bit 0 of Condition register).
4. Positive results receive a hexadecimal C sign, negative results receive a hexadecimal D.
5. The effective addresses of the source fields must not be absolute address zero on machines with less than 65K main storage, and the addresses must not be the first location of the memory partition assigned to the processor executing the instruction if the Relocation and Protection feature is installed and READ protection is invoked.

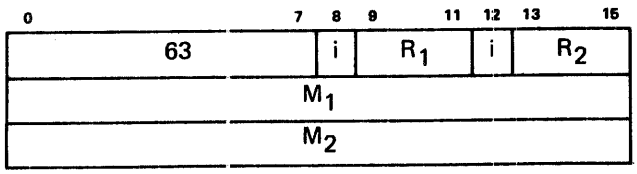
EXAMPLE



The field identified by FIELD1(8,2) is subtracted from the field identified by FIELD2(,4). The length of FIELD1 for this instruction is defined as 8 bytes; the length of FIELD2 is assumed to be the length as defined previously in the program. The result will be placed at the location identified by FIELD2(,4).

Subtract Memory – Memory

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



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

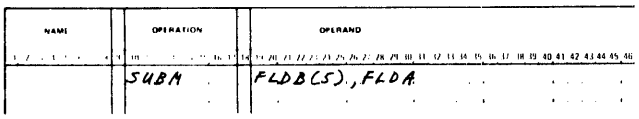
OPERAND 1: The subtrahend; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both. This operand is subtracted from operand 2.

OPERAND 2: The minuend; same length and addressing options (to base M_2) as operand 1.

RESULTS: The 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.

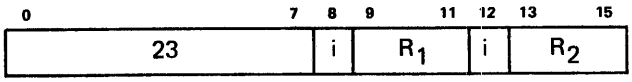
EXAMPLE



The field identified by FLDB(5) is subtracted from the field identified by FLDA. The result will be at the address represented by FLDA.

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.

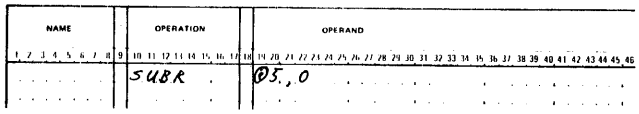
OPERAND 1: The subtrahend; a one-word field located in the general register specified by R_2, or in memory if indirect addressing is used. This operand is subtracted from operand 2.

OPERAND 2: The minuend; a one-word field located in the general register specified by R_2 or in memory if indirect addressing is used.

RESULTS: The 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.

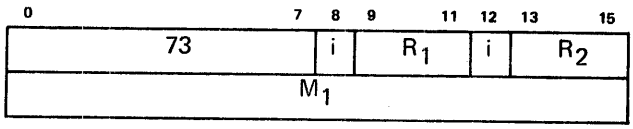
EXAMPLE



The field at the location specified in register 5 is subtracted from the field in register 0. The result will be placed in register 0.

Subtract Two-Word

SUBT @M_1(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: The subtrahend; a two-word field in memory beginning at the specified effective address. The most significant bits are at this address.

Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both. This operand is subtracted from operand 2.

OPERAND 2: The minuend; a two-word field located in two general registers or in memory.

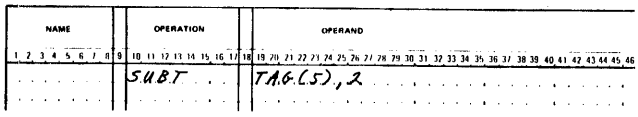
If direct addressing is used, the field is in the register specified by R_2 and the next highest register, R_2+1; the most significant bits are in the R_2 register. (Note: If register 7 is specified by R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at the address in the R_2 register; the most significant bits are at this address.

RESULTS: The 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 $+2^{32}$-1.
- Subtracting 80000000_{16} from 00000000_{16} results in 80000000_{16} and bit 0 is set.

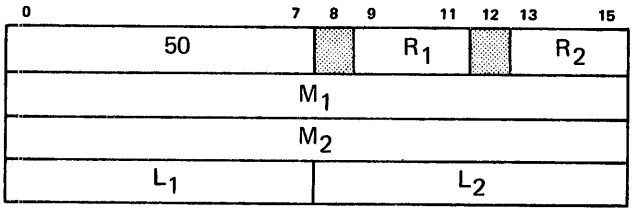
EXAMPLE



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

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 address indicated by M_1 may be indexed, but indirect addressing is not allowed. The effective 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 address indicated by M_2 may be indexed, but no indirect addressing is allowed. The effective address points to the most significant bytes of the field.

RESULTS: The result 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, bits 3 and 7 of the Condition register are 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.

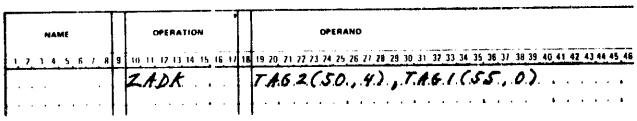
The Condition register is affected as follows:

- 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, 6, and 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.

SPECIAL FEATURES

1. Validity of source digits is not checked.
2. Invalid digits produce inconsistent results.
3. Negative zero cannot be produced unless overflow occurs (bit 0 of Condition register).
4. Positive results receive a hexadecimal C sign, negative results receive a hexadecimal D.
5. The effective addresses of the source fields must not be absolute address zero on machines with less than 65K bytes of main storage, and the addresses must not be the first location of the memory partition assigned to the executing processor if the Relocation and Protection feature is present and READ protection is invoked.

EXAMPLE



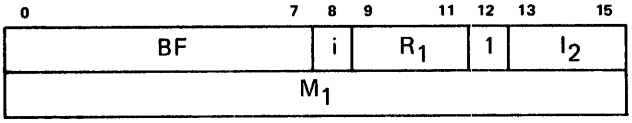
Initially, a 55-byte field identified by TAG1(55,0) is cleared to zero; then a 50-byte field identified by TAG2(50,4) is added to it. (See Results description for Condition register status.)

BIT-ORIENTED INSTRUCTIONS

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

Invert Bit •

IBIT @$M_1(R_1)$,I_2



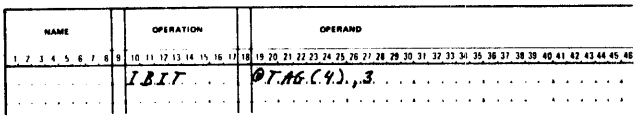
FUNCTION: Invert (toggle) a bit in a one-byte field in memory.

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

OPERAND 2: A 3-bit value in bits 13-15 of the instruction. The I_2 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 result field resides at the operand 1 location.

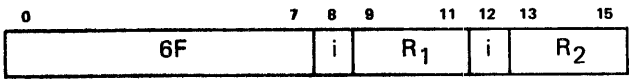
EXAMPLE



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).

Reverse Off-Bit

ROFR @R_1,@R_2



FUNCTION: Scans a one-word field, left to right, for the first off-bit (0-bit); turns that bit on; then increases another one-word field by an amount equal to the position (0-15) of the first 0-bit. If no 0-bit is found in the first field, the second 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. This field is scanned for the first 0-bit.

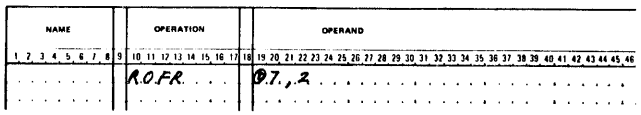
OPERAND 2: A one-word field in the general register specified by R_2, or in memory if indirect addressing is used. The field is increased by a value equal to the position of the first 0-bit in operand 1.

RESULTS: The results vary as described in the preceding text; the location of the operands is not changed.

If the same register is specified for both operands, the results are as shown, varying with the addressing mode of the operands:

Operand 1	Operand 2	Results
Both in same mode		Only the increased value is returned, with no alteration of the off-bit.
Indirect	Direct	Both operands are returned as normal.
Direct	Indirect	Operand 1 with the first off-bit set is returned in the register specified in Operand 1; then the increased value is written in memory at the location specified by this newly altered Operand 1.

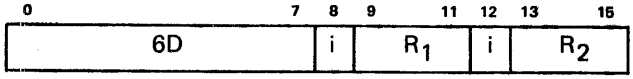
EXAMPLE



Register 7 contains the address of a 16-bit field which is scanned from left to right for a 0-bit. If a 0-bit is found, the relative position (0-15) determines the value to add to a one-word field in register 2. After this value is increased, the 0-bit is set to 1. If no 0-bit is found, register 2 is increased by a value of 16.

Reverse On-Bit

RONR @R_1,@R_2



FUNCTION: Scans a one-word field, left to right, for the first on-bit (1-bit); turns that bit off; then increases another one-word field by an amount equal to the position (0-15) of that first 1-bit. If no 1-bits are found in the first field, the second 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. This field is scanned for the first 1-bit.

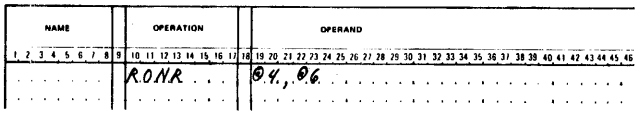
OPERAND 2: A one-word field located in the general register specified by R_1, or in memory if indirect addressing is used. This field is increased by a value equal to the position of the first 1-bit in operand 1.

RESULTS: The results vary as described in the preceding text.

If the same register is specified for both operands, the results are as shown, varying with the addressing mode of the operands.

Operand 1	Operand 2	Results
Both in same mode		Only the increased value is returned, with no alteration of the first on-bit.
Indirect	Direct	Both operands are returned as normal.
Direct	Indirect	Operand 1 with the first on-bit reset is returned in the register specified in Operand 1; then the increased value is written in memory at the location specified by the newly-altered Operand 1.

EXAMPLE

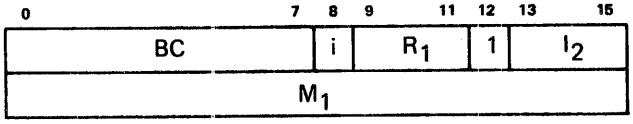


Register 4 contains the address of a 16-bit field which is scanned for a 1-bit. If a 1-bit is found, the relative

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 @$M_1(R_1),I_2$



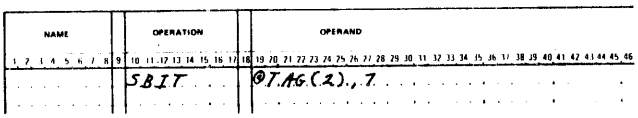
FUNCTION: Sets a bit (to 1) in a one-byte field in memory.

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

OPERAND 2: A 3-bit value in bits 13-15 of the instruction. The I_2 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 result field resides at the operand 1 location.

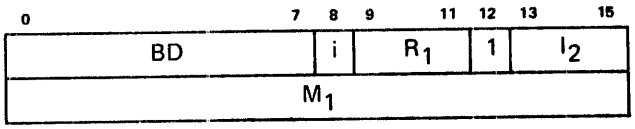
EXAMPLE



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

Reset Bit •

RBIT @$M_1(R_1),I_2$



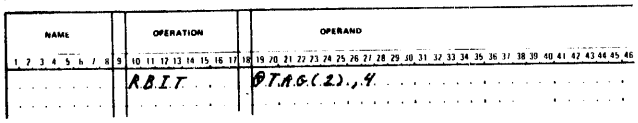
FUNCTION: Resets a bit (to 0) in a one-byte field in memory.

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

OPERAND 2: A 3-bit value in bits 13-15 of the instruction. The I_2 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 result field resides at the operand 1 location.

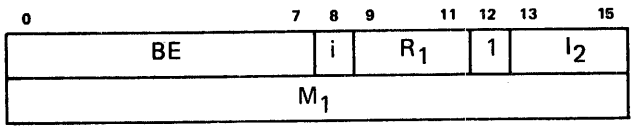
EXAMPLE



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

Test Bit •

TBIT @$M_1(R_1),I_2$



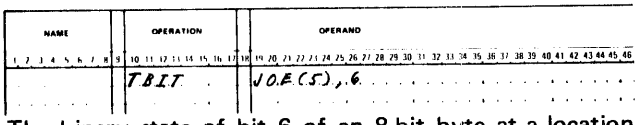
FUNCTION: Tests a bit in a one-byte field in memory, and transfers the bit state (1 or 0) to bit 0 of the Condition register.

OPERAND 1: The one-byte field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A 3-bit value in bits 13-15 of the instruction; the I_2 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 state of bit 0 in the Condition register.

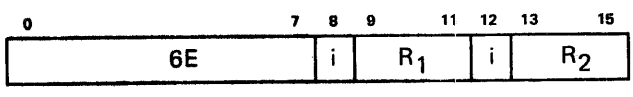
EXAMPLE



The binary state of bit 6 of an 8-bit byte at a location specified by JOE(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 (0-bit) and increases another one-word field by an amount equal to the position of that first 0-bit (0-15). The first 0-bit is not changed. If no 0-bits are

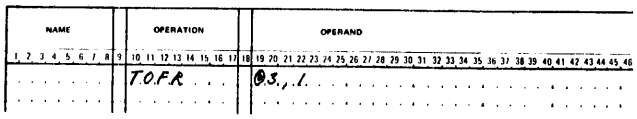
found in the first field, the second field is increased by a value of 16. Either field may be in a general register or in memory.

OPERAND 1: A one-word field in the general register specified by R_1, or in memory if indirect addressing is used. This field is scanned for the first 0-bit.

OPERAND 2: A one-word field in the general register specified by R_2, or in memory if indirect addressing is used. This field is increased by an amount equal to the position of the first 0-bit in operand 1.

RESULTS: The results vary as described in the preceding text; the location of the operands is not changed.

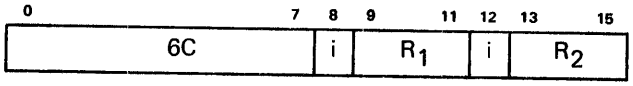
EXAMPLE



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 in register 1 is increased by 16. However, if a 0-bit is found, the relative position (0-15) of that bit specifies the value added to register 1.

Test for On-Bit

TONR @R_1,@R_2



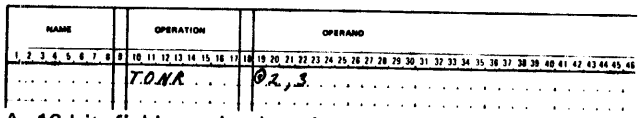
FUNCTION: Scans a one-word field, left to right, for the first on-bit (1-bit) and increases another one-word field by an amount equal to the bit position of the first 1-bit (0-15). The first 1-bit is not changed. If no 1-bits are found in the first field, the second field is increased by a value of 16. 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. This field is scanned for the first 1-bit.

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

RESULTS: The results vary as described in the preceding text; the location of the operands is not changed.

EXAMPLE



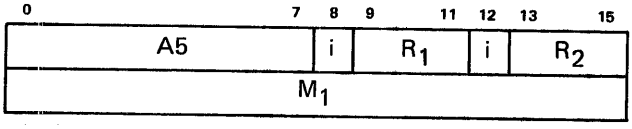
A 16-bit field at the location specified in register 2 is scanned left to right (0-15) for a 1-bit. If none is found, the value in register 3 is increased by 16. If a 1-bit is found, the relative position (0-15) of that bit corresponds to the value added to register 3.

BOOLEAN LOGIC INSTRUCTIONS

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

Logical Product Memory – Register

AND @M_1(R_1),@R_2



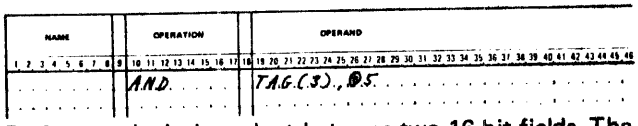
FUNCTION: Performs a logical product of a one-word field in memory and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

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

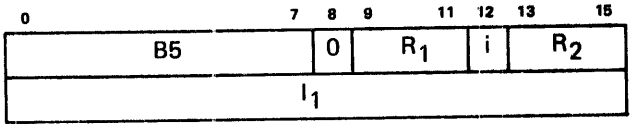
EXAMPLE



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

Logical Product Direct

ANDD $I_1(R_1),@R_2$



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.

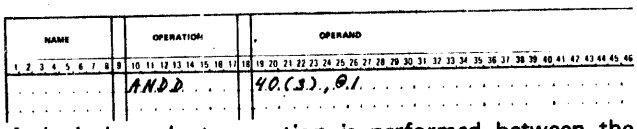
OPERAND 1: A 16-bit immediate value in the second word of the instruction; the value may range from 0-65,535.

Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I_1 value and the contents of the general register 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.

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

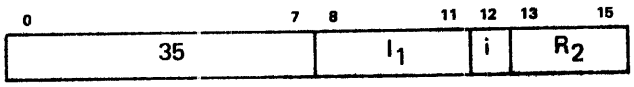
EXAMPLE



A logical product operation is performed between the immediate value 40, 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 $I_1,@R_2$



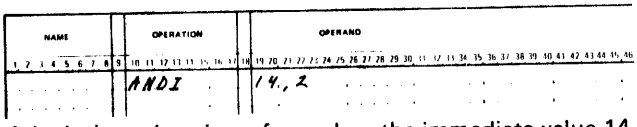
FUNCTION: Performs a logical product of a 4-bit immediate value and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction; the value may range from 0-15. The I_1 value is compared to operand 2 in bit positions 12-15; bits 0-11 are zeros.

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

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

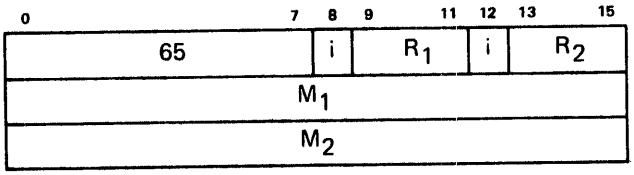
EXAMPLE



A logical product is performed on the immediate value 14 and bits 12-15 of the one-word field in register 2. This register will hold the results; bits 0-11 will always be 0's.

Logical Product Memory – Memory

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



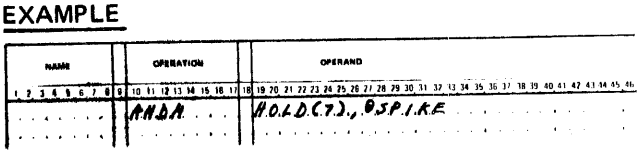
FUNCTION: Performs a logical product of two one-word fields in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

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

OPERAND 2: A one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

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

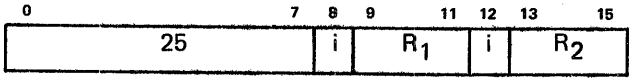
EXAMPLE



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. The result is stored at the @SPIKE address.

Logical Product Register — Register

ANDR @R_1,@R_2



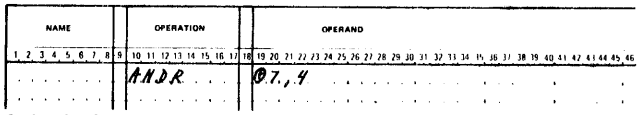
FUNCTION: Performs a logical product of two one-word fields; either field may be in a register or in memory. Corresponding bits in each operand are compared. If both bits are 1, the corresponding resultant bit is 1; in all other cases the resultant bit is 0.

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

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

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

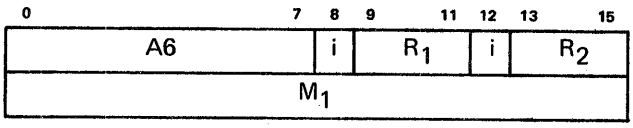
EXAMPLE



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

Exclusive OR Memory — Register

EOR @M_1(R_1),@R_2



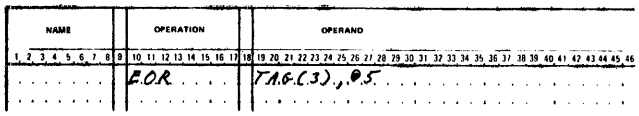
FUNCTION: Performs an exclusive OR of a one-word field in memory and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

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

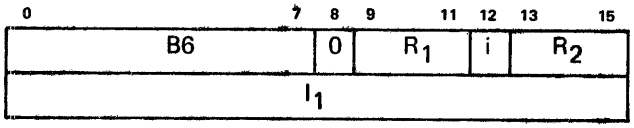
EXAMPLE



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 specified in register 5. The address specified in register 5 is also that of the result.

Exclusive OR Direct

EORD I_1(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.

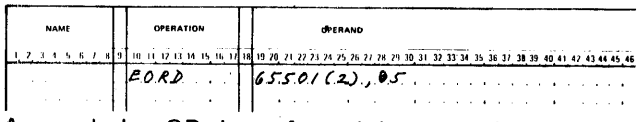
OPERAND 1: A 16-bit immediate value in the second word of the instruction; the value may range from 0 to 65,535.

Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I_1 value and the contents of the general register 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.

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

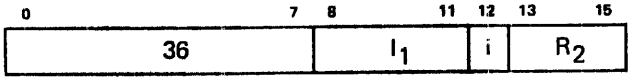
EXAMPLE



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

Exclusive OR Immediate

EORI I_1,@R_2



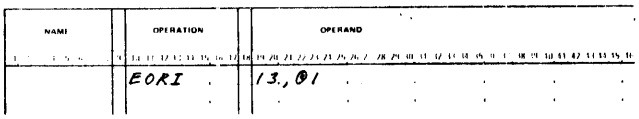
FUNCTION: Performs an exclusive OR between a 4-bit immediate value held in the instruction and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the resultant bit is 1; if the bits are the same, the resultant bit is 0.

OPERAND 1: A 4-bit unsigned value located in bits 8-11 of the instruction; the value may range from 0-15. The I_1 value is "OR"ed to operand 2 in bit positions 12-15; bits 0-11 are zeros.

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

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

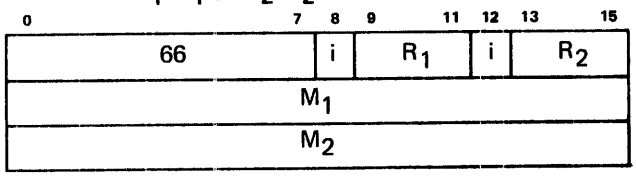
EXAMPLE



An exclusive OR is performed on the immediate value of 13 and the 16-bit field at the location specified 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)



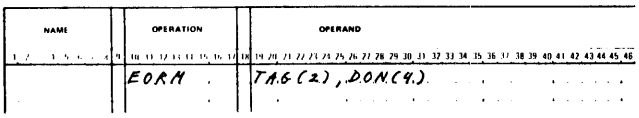
FUNCTION: Performs an exclusive OR of two one-word fields in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

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

OPERAND 2: A one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

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

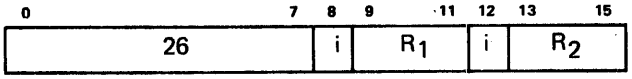
EXAMPLE



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 DON(4). The result is stored at the address specified by DON(4).

Exclusive OR Register-Register

EORR @R_1,@R_2



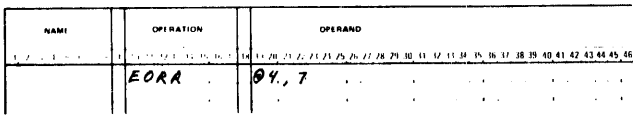
FUNCTION: Performs an exclusive OR of two one-word fields; either field may be in a register or in memory. Corresponding bits in each operand are compared. If the bits are unlike, the corresponding resultant bit is 1; if the bits are the same, the resultant bit is 0.

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

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

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

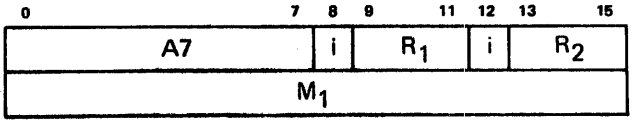
EXAMPLE



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

Inclusive OR Memory-Register

IOR @M_1(R_1),@R_2



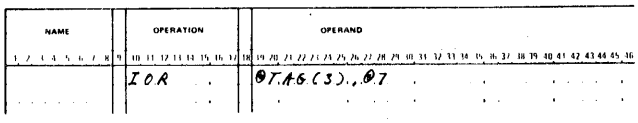
FUNCTION: Performs an inclusive OR of a one-word field in memory and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or if both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit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

OPERAND 1: A one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

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

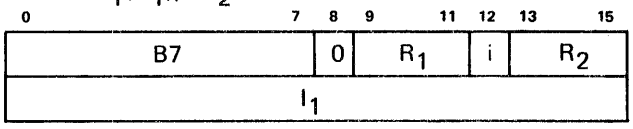
EXAMPLE



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 results will be at the address in register 7.

Inclusive OR Direct

IORD $I_1(R_1),@R_2$



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.

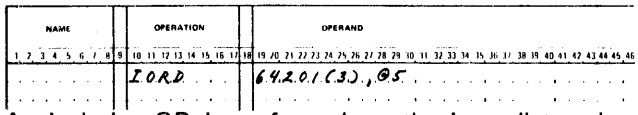
OPERAND 1: A 16-bit immediate value in the second word of the instruction; the value may range from 0-65,535.

Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I_1 value and the contents of the general register 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.

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

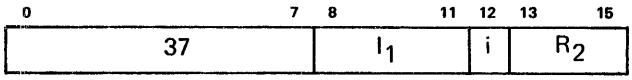
EXAMPLE



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 $I_1,@R_2$



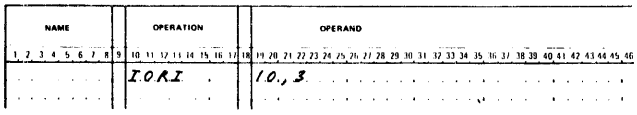
FUNCTION: performs an inclusive OR between a 4-bit immediate value held in the instruction and a one-word field in a general register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or both bits are 1, the corresponding resultant bit is 1. If both bits are 0, 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_1 value is "OR"ed to operand 2 in bit positions 12-15; bits 0-11 are zeros.

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

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

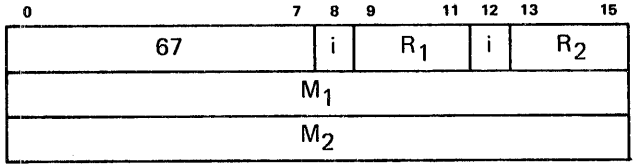
EXAMPLE



An inclusive OR is performed on the immediate value 10 and bits 12-15 of the one-word field in register 3. The result is stored in bits 12-15 in register 3; bits 0-11 are unaffected.

Inclusive OR Memory – Memory

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



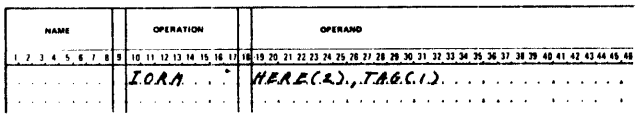
FUNCTION: Performs an inclusive OR of two one-word fields in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

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

OPERAND 2: A one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

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

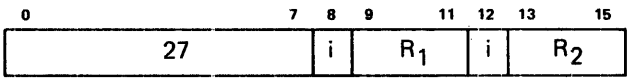
EXAMPLE



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

Inclusive OR Register – Register

IORR @R_1,@R_2



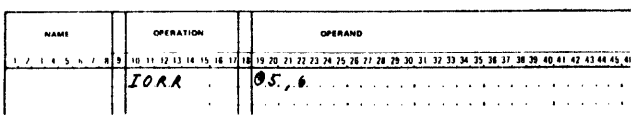
FUNCTION: Performs an inclusive OR of two one-word fields; either field may be in a register or in memory. Corresponding bits in each operand are compared. If either of the bits is 1 or if both of the bits are 1, the corresponding resultant bit is 1. If both bits are 0, the resultant bit is 0.

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

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

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

EXAMPLE



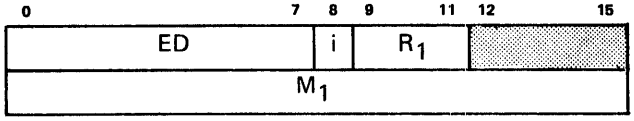
An inclusive OR is performed between a 16-bit field at the address specified in register 5 and the 16-bit field in register 6. Results are placed in register 6.

BRANCHING INSTRUCTIONS

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

Branch

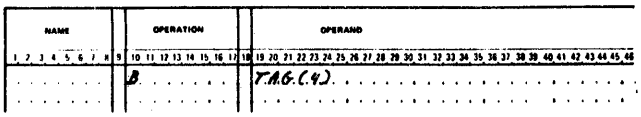
B @M_1(R_1)



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

OPERAND 1: The single operand identifies the memory location to which the program branches. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

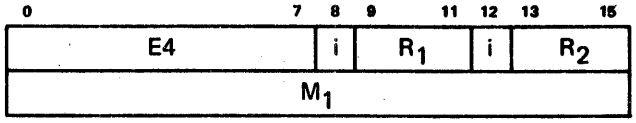
EXAMPLE



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

Branch Add One

BA1 @M_1(R_1),@R_2

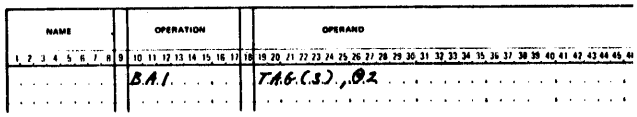


FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is increased by a value of 1, and the program branches to a specified memory location.

OPERAND 1: The memory location to which the program branches if the tested field is not zero. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

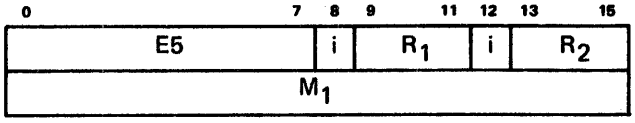
EXAMPLE



A 16-bit field at the address in register 2 is tested; 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_1(R_1),@R_2

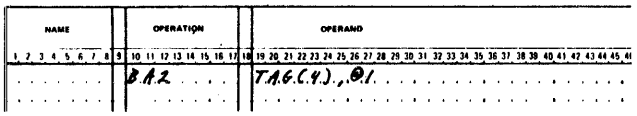


FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is increased by a value of 2, and the program branches to a specified memory location.

OPERAND 1: The memory location to which the program branches if the tested field is not zero. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

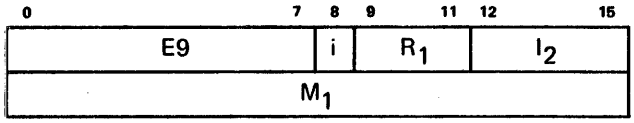
EXAMPLE



A 16-bit field at the address in register 1 is tested; 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 on Condition Register False

BCF @M_1(R_1),I_2

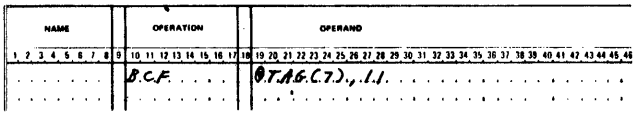


FUNCTION: Branches to a specified memory location if a designated Condition register bit is off. If the bit is on, the next instruction in the program is executed.

OPERAND 1: The memory location to which the program branches if the designated Condition register bit is on. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The I_2 value specifies the position of the bit to be tested in the Condition register and may range from 0-15.

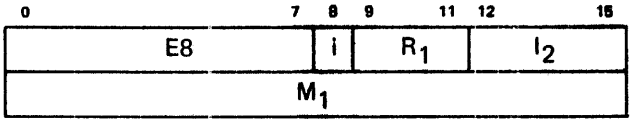
EXAMPLE



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

Branch on Condition Register True

BCT @$M_1(R_1),I_2$

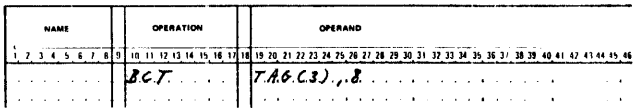


FUNCTION: Branches to a specified memory location if a designated Condition register bit is on. If the bit is off, the next instruction in the program is executed.

OPERAND 1: The memory location to which the program branches if the designated Condition register bit is on. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The I_2 value specifies the position of the bit to be tested in the Condition register and may range from 0-15.

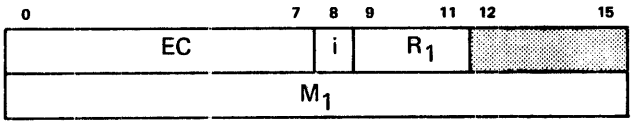
EXAMPLE



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 executed.

Branch Unconditional

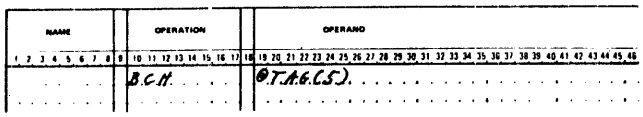
BCH @$M_1(R_1)$



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

OPERAND: The memory location to which the program jumps. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

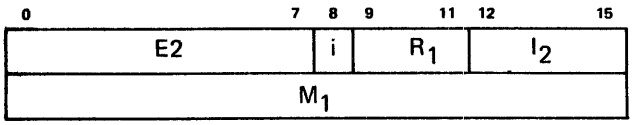
EXAMPLE



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).

Branch if Bit Off

BOF @$M_1(R_1),I_2$

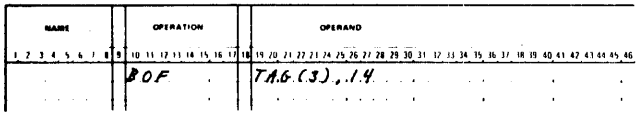


FUNCTION: Branches to a specified memory location if the bit tested in a general register is off. If the bit is on, the next instruction in the program is executed.

OPERAND 1: The operand is composed of two parts: the general register tested is specified by R_1, and the 16-bit value contained in M_1 is the memory address to which the program branches. Addressing options to the base address M_1 include indirect addressing, but not indexing.

OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The I_2 value specifies the position of the bit to be tested in the general register and may range from 0-15.

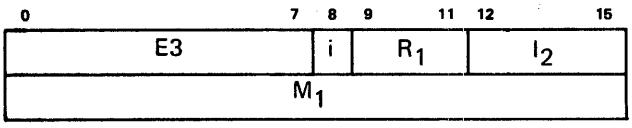
EXAMPLE



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

Branch if Bit On

BON @$M_1(R_1),I_2$

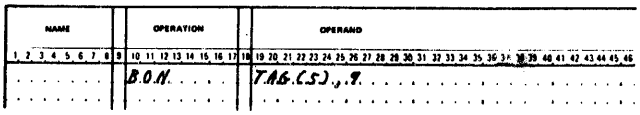


FUNCTION: Branches to a specified memory location if the bit tested in a general register is on. If the bit is off, the next instruction in the program is executed.

OPERAND1: The operand is composed of two parts: the general register tested is specified by R_1, and the 16-bit value contained in M_1 is the memory address to which the program branches. Addressing options to the base address M_1 include indirect addressing, but not indexing.

OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The I_1 value specifies the position of the bit to be tested in the general register and may range from 0-15.

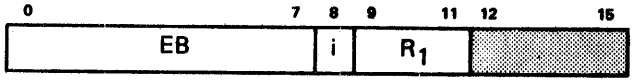
EXAMPLE



This instruction tests bit 9 in register 5. If bit 9 is on, the program branches to the address identified by TAG. Otherwise, no branch is made and the next instruction is executed.

Branch to Address in Register

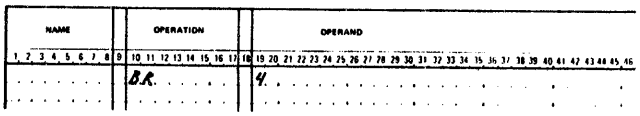
BR @R_1



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

OPERAND 1: The single operand for this instruction is a memory address. If direct addressing is used, the address is in the register specified by R_1. If indirect addressing is used, the address is located at the address specified in R_1.

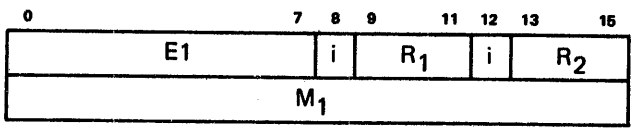
EXAMPLE



Branches to the address specified in register 4.

Branch if Register is Not Zero

BRN @M_1(R_1),@R_2

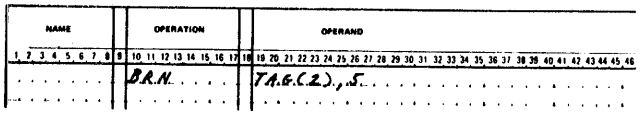


FUNCTION: Branches to a specified memory location if the general register tested does not contain all zeros. If the register contains all zeros, the next instruction in the program is executed.

OPERAND 1: The memory location to which the program branches if the general register does not contain all zeros. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

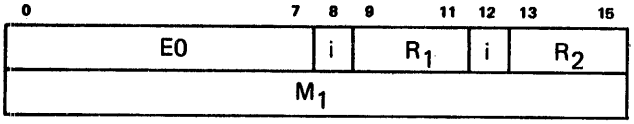
EXAMPLE



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 executed.

Branch if Register is Zero

BRZ @M_1(R_1),@R_2

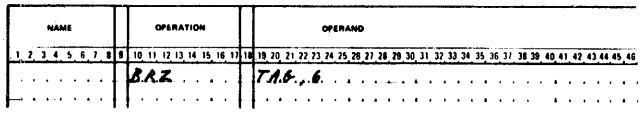


FUNCTION: Branches to a specified memory location if the general register tested contains all zeros. If the register does not contain all zeros, the next instruction in the program is executed.

OPERAND 1: The memory location to which the program branches if the general register contains all zeros. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

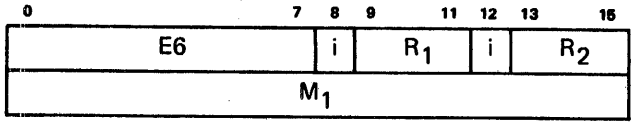
EXAMPLE



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 executed.

Branch Subtract One

BS1 @M_1(R_1),@R_2

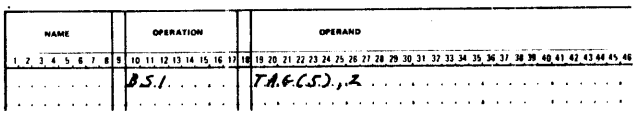


FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is decreased by a value of 1, and the program branches to a specified memory location.

OPERAND 1: The memory location to which the program branches if the tested field is not zero. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

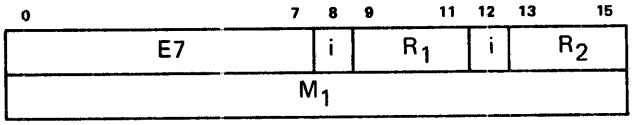
EXAMPLE



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

Branch Subtract Two

BS2 $@M_1(R_1),@R_2$

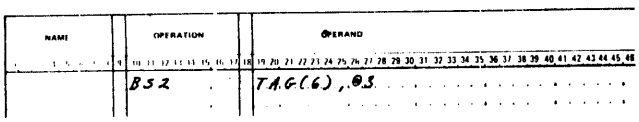


FUNCTION: Tests a one-word field in a general register or in memory for a zero value; if the field is zero, the next instruction in the program is executed. If the field tested is not zero, it is decreased by a value of 2 and the program jumps to a specified memory location.

OPERAND 1: The memory location to which the program jumps if the tested field is not zero. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

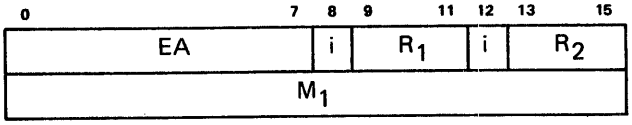
EXAMPLE



The field at the address in register 3 is tested. If the field contains a non-zero value, a branch is made to the field identified by TAG(6), and the value of the field identified by @3 is decreased by 2. If the tested field is zero, the next instruction is executed.

Branch and Save Return

BSR $@M_1(R_1),@R_2$

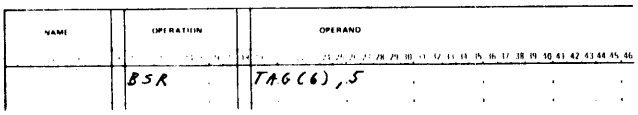


FUNCTION: Branches unconditionally to a specified memory address, storing the address of the next instruction in a general register or in memory. The address stored (return address) is the current program address plus four bytes.

OPERAND 1: The memory location to which the program branches. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The general register specified by R_2, or the memory field at the address in the R_2 register if indirect addressing is used, that contains the return address.

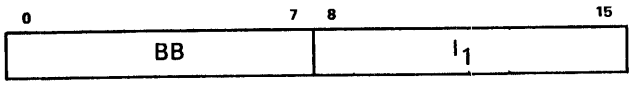
EXAMPLE



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.

Skip Back-Unconditional

SB I_1

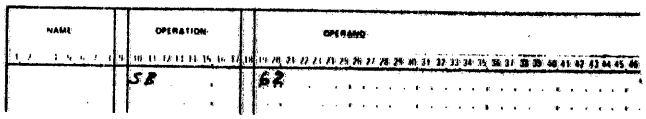


FUNCTION: Skips back in the program a specified number of words.

OPERAND 1: 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. When the instruction is

executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is subtracted from the current program address.

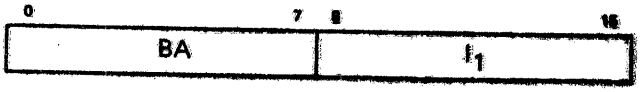
EXAMPLE



Skips back unconditionally in the program 62 words (124 bytes).

Skip Forward-Unconditional

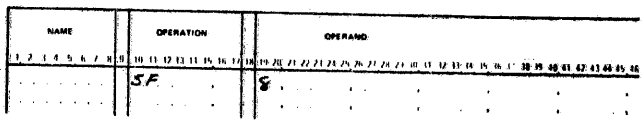
SF I_1



FUNCTION: Skips forward in the program a specified number of words.

OPERAND 1: 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. When the instruction is executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is added to the current program address.

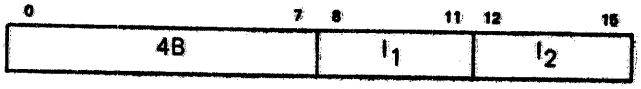
EXAMPLE



Skips forward unconditionally in the program 8 words (16 bytes).

Skip on Condition False-Back

SCFB I_1,I_2



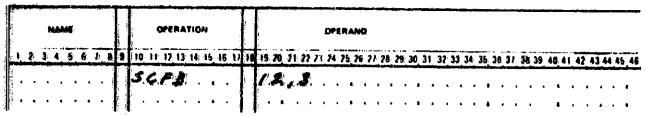
FUNCTION: Skips back in the program a specified number of words if the appropriate Condition register bit is off. If the bit is on, the next instruction in the program is executed.

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.

When the instruction is executed the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is 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.

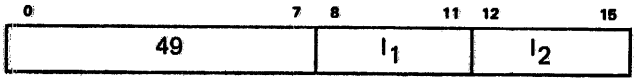
EXAMPLE



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 False-Forward

SCFF I_1,I_2

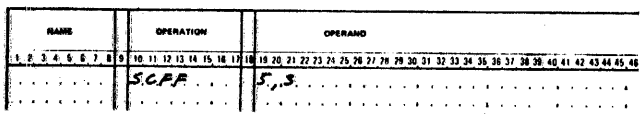


FUNCTION: Skips forward in the program a specified number of words if the appropriate Condition register bit is off. If the bit is on, the next instruction in the program is executed.

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. When the instruction is executed the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is added to the current program address.

OPERAND 2: A 4-bit value in bits 12-15 of the instruction. The I_1 value specifies the position of the bit tested for *off* in the Condition register and may range from 0-15.

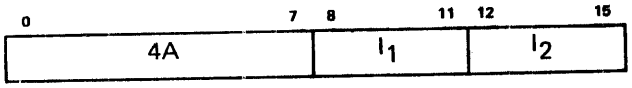
EXAMPLE



If bit 3 (equal bit) in the Condition register is off, the program will skip forward in the program five words (10 bytes). If bit 3 is on, the next instruction in the program is executed.

Skip on Condition True-Back

SCTB I_1,I_2

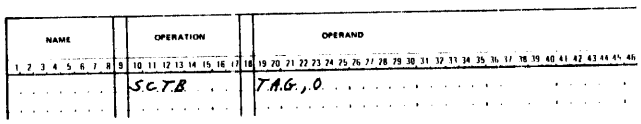


FUNCTION: Skips back in the program a specified number of words if the appropriate Condition register bit is on. If the bit is off, the next instruction in the program is executed.

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. When the instruction is executed the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is 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.

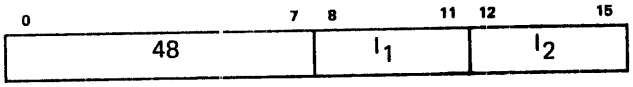
EXAMPLE



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

Skip on Condition True-Forward

SCTF I_1,I_2

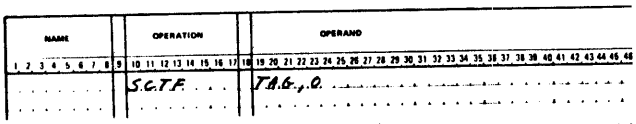


FUNCTION: Skips forward in the program a specified number of words if the appropriate Condition register bit is on. If the bit is off, the next instruction in the program is executed.

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. When the instruction is executed the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is 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.

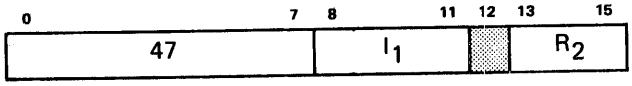
EXAMPLE



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 if Register Minus-Back

SRMB I_1,R_2

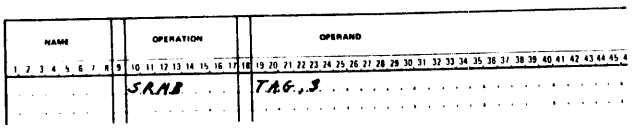


FUNCTION: Skips back in the program a specified number of words if the register contents tested are minus. If the register contents are plus, the next instruction in the program is executed.

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. When the instruction is executed the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is subtracted from the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

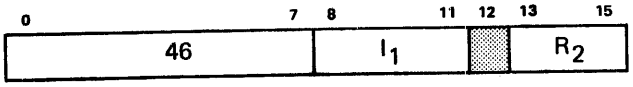
EXAMPLE



Tests the contents of register 3. If negative, skips back to the instruction located at TAG; if positive or zero, the next instruction is read. Note that the assembler determines the number of words (0-15) to skip by the location of TAG.

Skip if Register Minus-Forward

SRMF I_1,R_2

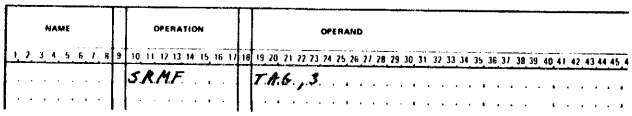


FUNCTION: Skips forward in the program a specified number of words if the register contents tested are minus. If the register contents are plus, the next instruction in the program is executed.

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. When the instruction is executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is added to the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

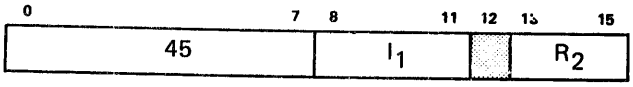
EXAMPLE



Tests the contents of register 3. If negative, skips forward to the instruction located at TAG; if positive or zero, the next instruction is read. (The assembler determines the distance of TAG from the SRMF instruction and uses this value for I_1.)

Skip if Register Plus-Back

SRPB I_1,R_2

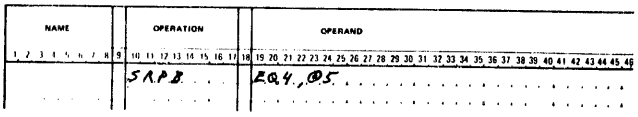


FUNCTION: Skips back in the program a specified number of words if the register contents tested are plus. If the register contents are minus, the next instruction in the program is executed.

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. When the instruction is executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is subtracted from the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

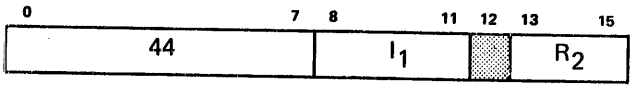
EXAMPLE



Assume that EQ4 is equated to -3. The program tests the field at the address in register 5. If positive, the program skips back three words (six bytes); if not positive, the next instruction is executed.

Skip if Register Plus-Forward

SRPF I_1,R_2

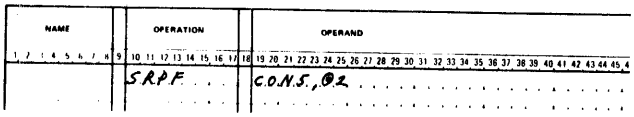


FUNCTION: Skips forward in the program a specified number of words if the register contents tested are plus. If the register contents are minus, the next instruction in the program is executed.

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. When the instruction is executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is added to the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

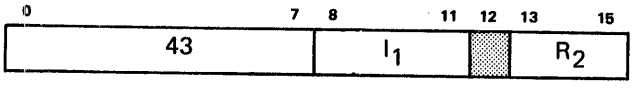
EXAMPLE



Assume that CON5 is equated to 11. The program tests the field at the address in register 2. If positive, the program skips forward 11 words (22 bytes); if not positive, the next instruction is executed.

Skip if Register Not Zero-Back

SRNB I_1,R_2



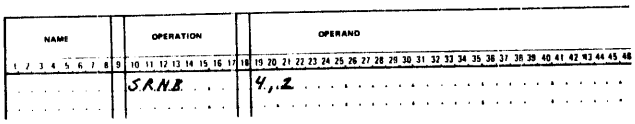
FUNCTION: Skips back in the program a specified number of words if the register contents tested are not zero. If the register does contain all zeros, the next instruction in the program is executed.

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. When the instruction is

executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is subtracted from the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

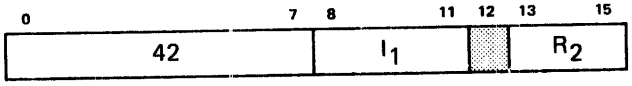
EXAMPLE



Tests the contents of register 2. If non-zero, the program skips four words (eight bytes) back in the program; if zero, executes next instruction.

Skip if Register Not Zero-Forward

SRNF I_1,R_2

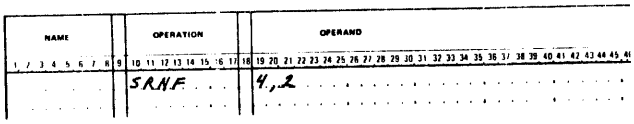


FUNCTION: Skips forward in the program a specified number of words if the register contents tested are not zero. If the register does contain all zeros, the next instruction in the program is executed.

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. When the instruction is executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is added to the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

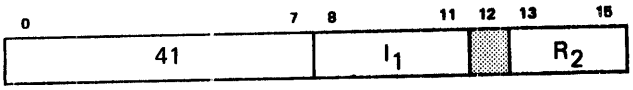
EXAMPLE



Tests the contents of register 2. If non-zero, the program skips four words (eight bytes) forward; if zero, executes next instruction.

Skip if Register Zero-Back

SRZB I_1,R_2

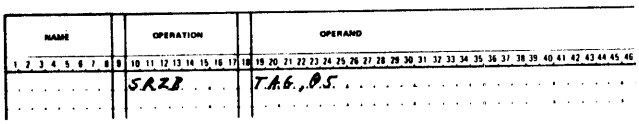


FUNCTION: Skips back in the program a specified number of words if the register tested contains all zeros. If the register does not contain all zeros, the next instruction in the program is executed.

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. When the instruction is executed, the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is subtracted from the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

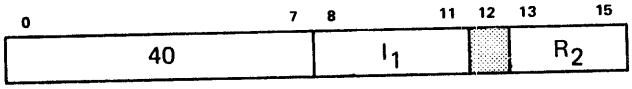
EXAMPLE



Tests the field at the address in register 5. If all zeros, reads the instruction at location TAG; if not all zeros, the next instruction is read. TAG must be within 15 words of SRZB. (The assembler determines the distance of TAG from the SRZB instruction and uses this value for I_1.)

Skip if Register Zero-Forward

SRZF I_1,R_2

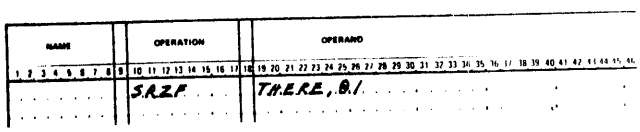


FUNCTION: Skips forward in the program a specified number of words if the register tested contains all zeros. If the register does not contain all zeros, the next instruction in the program is executed.

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. When the instruction is executed the number of bytes represented by I_1 is determined (I_1 is doubled). This byte value is added to the current program address.

OPERAND 2: The value tested; a one-word field in the general register specified by R_2.

EXAMPLE



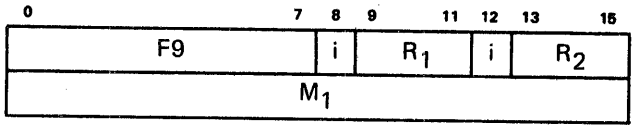
Tests the field at the address in register 1. If all zeros, reads the instruction at location THERE; if not all zeros, the next instruction is read. THERE must be within 15 words of SRZF. (The assembler determines the distance of THERE from the SRZF instruction and uses this value for I_1.)

COMPARE INSTRUCTIONS

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

Compare Byte Memory – Register ●

CBY @$M_1(R_1)$,@R_2



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

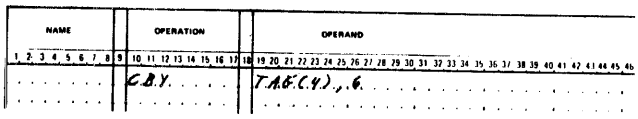
OPERAND 1: A one-byte field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

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.

RESULTS: Neither operand is changed. 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.

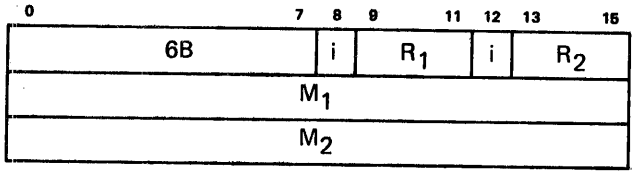
EXAMPLE



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

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.

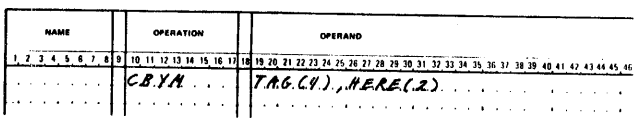
OPERAND 1: A one-byte field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A one-byte field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

RESULTS: Neither operand is changed. 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

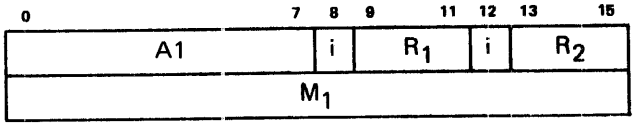


Compares a one-byte operand at the address specified by TAG(4) with another at the address specified by HERE(2). If the operand at TAG(4) is greater than the other operand, bit 1 of the Condition register is turned

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 Memory – Register

CMP @M_1(R_1),@R_2



FUNCTION: Performs a 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_1 include indexing, indirect addressing, 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.

RESULTS: Neither operand is changed. The Condition register is affected as follows.

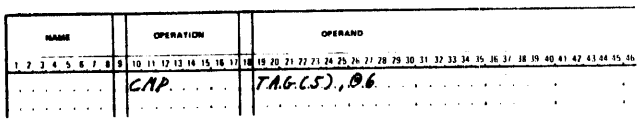
Bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare, as specified below:

- 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.

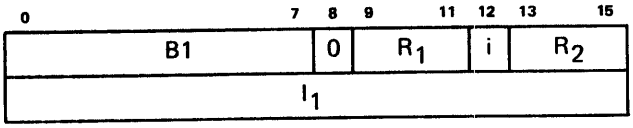
EXAMPLE



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 Direct

CMPD I_1(R_1),@R_2



FUNCTION: Performs a 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 the second word 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_1 value and the contents of the general register 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_1, or in memory if indirect addressing is used.

RESULTS: Neither operand is changed. 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, as specified below:

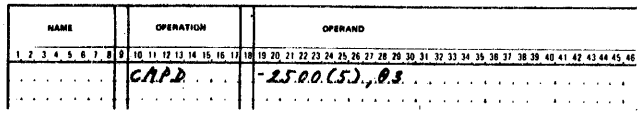
- Bits 0 and 4 are always cleared.
- If operand 1 is equal to operand 2, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.
- If operand 1 is arithmetically greater than operand 2, bit 1 is set and bits 2 and 3 are cleared.
- If operand 1 is arithmetically less than operand 2, bit 2 is set and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

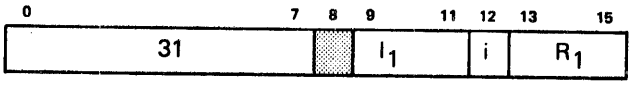
EXAMPLE



The value -2500 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_1,@R_2



FUNCTION: Performs a 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; bits 0-11 are zeros.

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

RESULTS: Operand 2 is not changed. The Condition register is affected as follows.

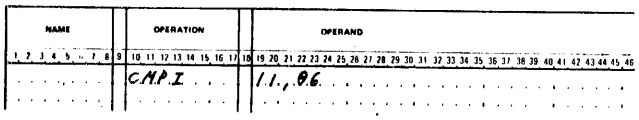
Bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare, as specified below:

- 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.

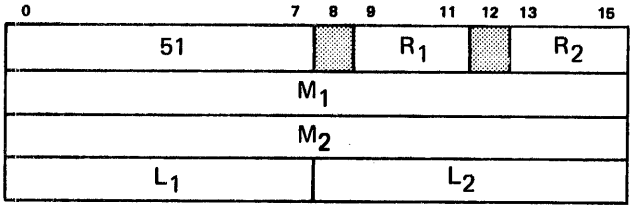
EXAMPLE



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

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.

RESULTS: Neither operand is changed. 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 $L_2 = 0$, bits 3 and 7 of the Condition register are set and bits 1, 2, 5, and 6 are cleared.

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.

CONSIDERATIONS:

1. Digit validity is not checked during the compare.
2. Invalid digits may produce inconsistent results (invalid digits are considered greater than valid digits).
3. A minus zero field is less than a positive zero field.
4. The compare is by sign first, and if it is like, then the compare is made bit by bit, left to right. The compare terminates as soon as an unequal condition occurs, therefore the timing is affected by data field contents as well as length. Timing supplied (Appendix E) is for equal compare and like signs.

EXAMPLE

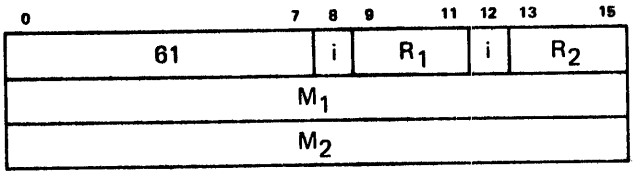
NAME	OPERATION	OPERAND
	CMPK	TAG(90,1),HERE(101,)

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), 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 would be made immediately if the signs are different.

Compare Memory – Memory

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



FUNCTION: Performs a 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, indirect addressing, or a combination of both.

OPERAND 2: A one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

RESULTS: Neither operand is changed. The Condition register is affected as follows.

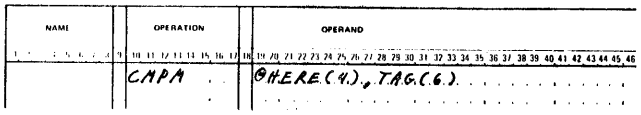
Bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare, as specified below:

- 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.

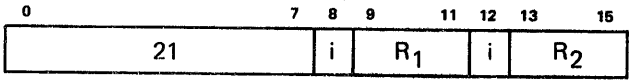
EXAMPLE



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 Register-Register

CMPR @R_1,@R_2



FUNCTION: Performs a 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.

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

RESULTS: Neither operand is changed. The Condition register is affected as follows.

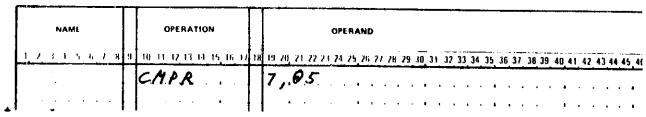
Bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare, as specified below:

- 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.

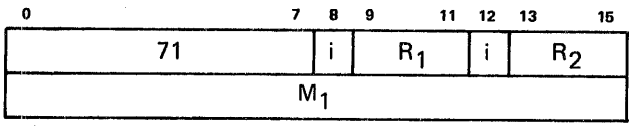
EXAMPLE



A 16-bit field 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_1(R_1),@R_2



FUNCTION: Performs a 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 beginning at the specified effective address. The most significant bits are at this address.

Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A two-word field located in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next highest register, R_2+1; the most significant bits are in the R_2 register. (Note: If register 7 is specified by R_2, the field is in registers 7 and 0 with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at the address in the R_2 register; the most significant bits are at this address.

RESULTS: Neither operand is changed. The Condition register is affected as follows.

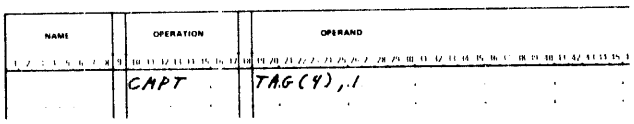
Bits 0-3 reflect the arithmetic results of the compare and bits 4-7 reflect the logical results of the compare, as specified below:

- 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.

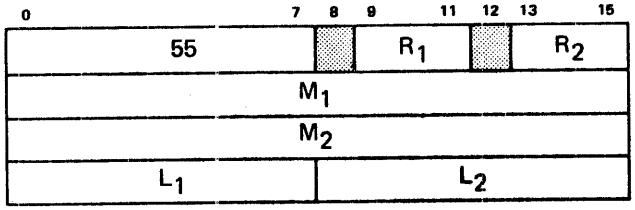
EXAMPLE



A 32-bit field at the address identified by TAG(4) is compared to a 32-bit field in registers 1 and 2; 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.

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.

RESULTS: Neither operand is changed. The following conditions may occur, depending on the values of L_1 and L_2:

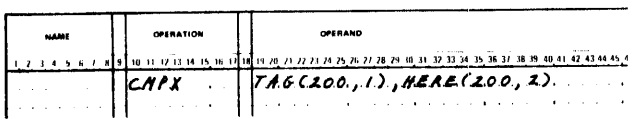
- If $L_1 = L_2$, the operands are compared byte-for-byte.
- If L_1 is less than L_2, the operands are compared until L_1 is exhausted, then blanks are compared to operand 2.
- If L_1 is greater than L_2, the operands are compared until L_2 is exhausted, then operand 2 is compared to blanks.
- If $L_1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

The Condition register is affected as follows:

- Bits 0 and 4 are always cleared.
- If operand 2 is greater than operand 1, bits 1 and 5 are set and bits 2, 3, 6 and 7 are cleared.
- If operand 2 is less than operand 1, bits 2 and 6 are set and bits 1, 3, 5 and 7 are cleared.
- If operand 2 is equal to operand 1, or if $L_1=L_2=0$, bits 3 and 7 are set and bits 1, 2, 5 and 6 are cleared.

CONSIDERATIONS: Word compare is performed if, and only if, the lengths L_1 and L_2 and the effective addresses are even.

EXAMPLE



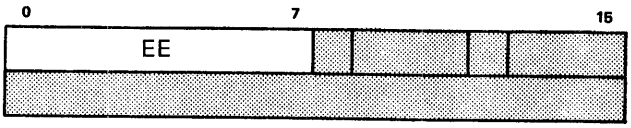
A 200-byte field identified by TAG(200,1) is compared to a 200-byte field identified by HERE(200,2). Comparison proceeds byte-by-byte until inequality is determined or all bytes have been compared and found equal. The Condition register is set accordingly.

CONTROL INSTRUCTIONS

Mnemonic	Name
NOP	No Operation
RDX	Read Extended Register
SR	Service Request

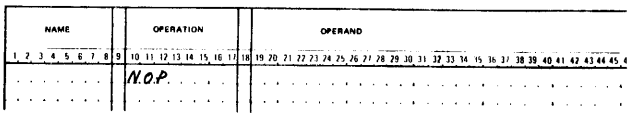
No Operation

NOP



FUNCTION: Performs no operation. This instruction has no operands.

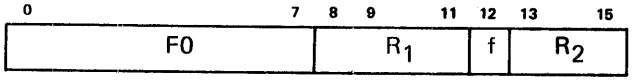
EXAMPLE



This instruction occupies four bytes in the program.

Read Extended Register

RDX E_1,R_2



FUNCTION: Reads a Group II extended register and stores the information in a general register.

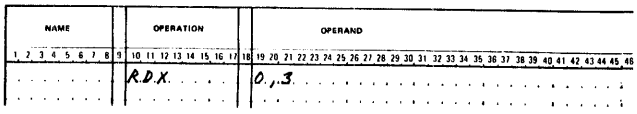
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction; this bit distinguishes between RDX and WRX.

OPERAND 1: The Group II extended register to be read.

OPERAND 2: The general register which is to receive the contents of the extended register.

CONSIDERATIONS: Any attempt to access a Group I register results in a trap to the Invalid Instruction routine.

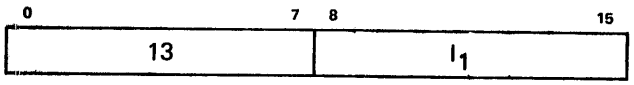
EXAMPLE



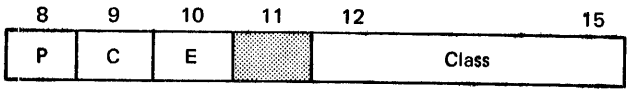
Reads the extended register 0 and stores the contents in general register 3.

Service Request

SR @I_1



FUNCTION: Provides an information byte called the request index (I_1) to be interpreted by the operating system (software). The request index byte has the following format:



P bit — Indicates location of parameter string (the @ sign in the source operand is used to turn this bit on).

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.

OPERAND 1: I_1 is the request index byte, which is defined by the operating system.

RESULTS: Execution of a Service Request instruction causes the following actions:

1. The Service Request bit (bit 13) of the executing processor's Condition register is set.
2. The Busy and Active bits of Processor 4 are both set.
3. The Busy and Active bits of the executing processor are both reset.

NOTE

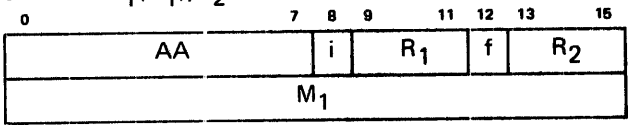
Any further effects of the SR instruction, beyond those described above, result from processing by the operating system software. For more information on the SR instruction, refer to the **Control Program and Data Management Services, Extended Reference** Manual.

DATA CONVERSION INSTRUCTIONS

Mnemonic	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 •
UNPX	Unpack •
TRNX	Translate •

Convert to Binary •

CVB @$M_1(R_1),R_2$



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; this function bit distinguishes between CVB and CVBT.

OPERAND 1: A 3-byte packed decimal field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both

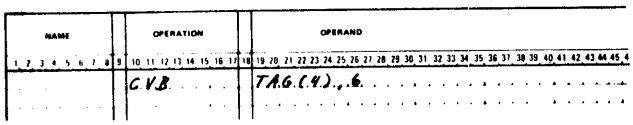
The effective address points to the most significant byte of the field. The packed decimal field may hold five digits and a sign.

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 result field resides at the operand 2 location. The Condition register is affected as follows.

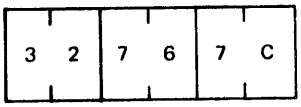
- Bit 0 (overflow) is set if results are greater than +32,767 or less than -32,767. (Note: -32,768 is converted correctly but the overflow bit is set.)
- Bits 1-7 unchanged.

EXAMPLE

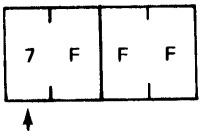


TAG(4) identifies a 3-byte packed decimal field (five digits plus sign) which is converted to a 16-bit (15 bits plus sign) binary value and loaded into register 6.

The 3-byte packed field at the effective address of TAG(4):

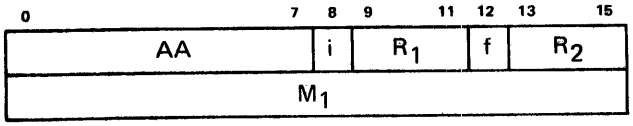


The resultant 2-byte binary field in register 6:



Convert to Binary Two-Word •

CVBT @$M_1(R_1),R_2$



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; this function bit distinguishes between CVB and CVBT.

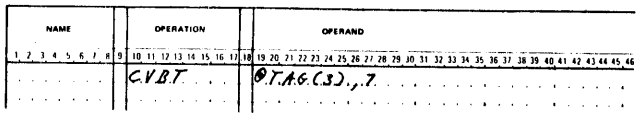
OPERAND 1: A 6-byte packed decimal field in memory. The packed decimal field can hold 11 digits and a sign. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both. The effective address points to the most significant byte of the field.

OPERAND 2: The resultant 4-byte field in two general registers that will hold a 32-bit (31 bits and a sign bit) binary field. The binary field will be in the register specified by R_2 and the next highest register R_2+1; the most significant bits are in the R_2 register. (Note: If register 7 is specified by R_2, the binary field is in registers 7 and 0, with the most significant bits in register 7.)

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

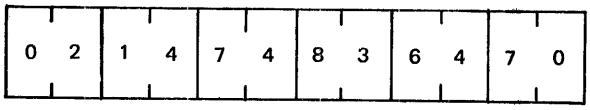
- Bit 0 (overflow) is set if results are greater than $+2^{31}-1$ or less than $-2^{31}-1$. (Note: -2^{31} is converted correctly but the overflow bit is set.)
- Bits 1-7 are unchanged.

EXAMPLE

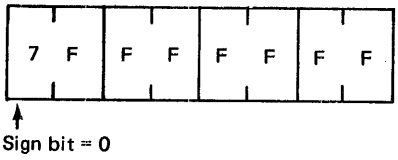


@TAG(3) identifies a 6-byte packed decimal field (11 digits plus sign) which is converted to a 32-bit (including sign) binary value and loaded into register 7.

The 6-byte packed decimal field at the effective address of @TAG(3):

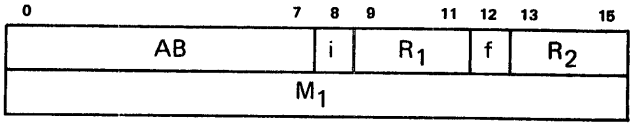


The resultant 4-byte binary field in register 7:



Convert to Decimal •

CVD @M_1(R_1),R_2



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

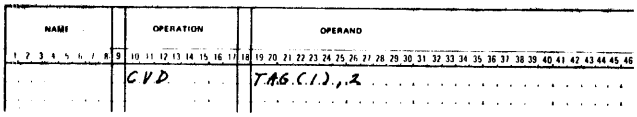
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction; this function bit distinguishes between CVD and CVDT.

OPERAND 1: The resultant 3-byte packed decimal field in memory which can hold five digits and a sign. Addressing options to the base address M_1 include indexing, indirect addressing 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.

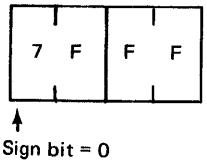
RESULTS: The result field resides at the operand 1 location.

EXAMPLE

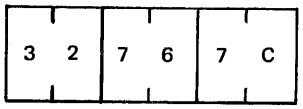


Register 2 contains a 16-bit binary value which is converted to a 3-byte packed decimal field and stored at the location specified by TAG(1).

The 2-byte binary field in register 2:

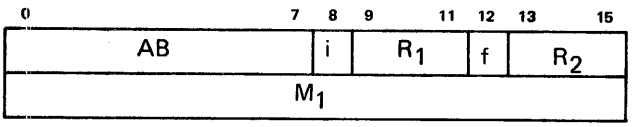


The resultant 3-byte packed field at the effective address of TAG(1):



Convert to Decimal Two-Word •

CVDT @M_1(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.

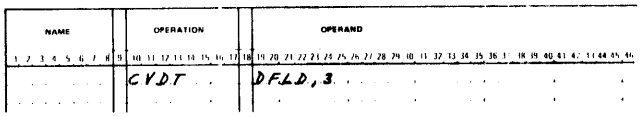
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for CVDT instruction; this function bit distinguishes between CVD and CVDT.

OPERAND 1: The resultant 6-byte packed decimal field located in memory. The packed decimal field may hold 11 digits and a sign. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both. The effective address points to the most significant byte of the field.

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

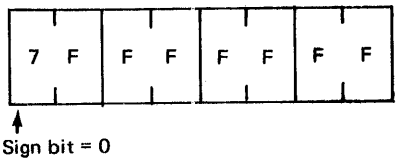
RESULTS: The result field resides at the operand 1 location.

EXAMPLE

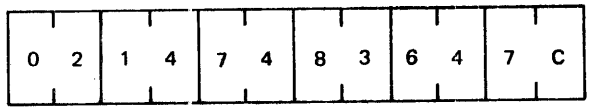


Registers 3 and 4 contain a 4-byte binary value which is converted to 6-byte packed decimal (11 digits plus sign) and stored at the location identified by DFLD.

The 4-byte binary field in registers 3 and 4:

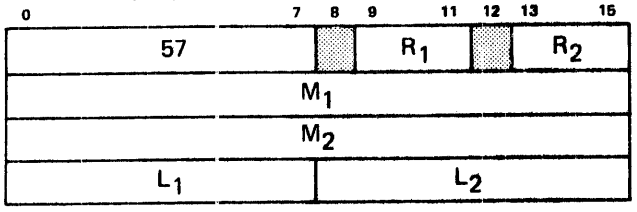


The resultant 6-byte packed decimal field at the address specified by DFLD:



Packed Decimal/Alpha Edit ●

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



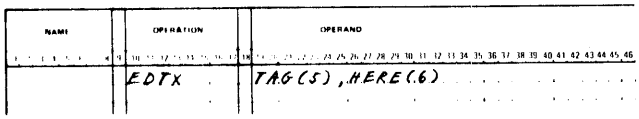
FUNCTION: This instruction moves the contents of a source field to a result field with editing symbols inserted according to an edit mask. The first-byte address of the mask field must be set in general register 1 prior to execution of the edit instruction. A more complete description of the functions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

OPERAND 1: The field in memory to be edited. It must be a packed decimal field for numeric editing; for alphanumeric 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 M_1 include only indexing.

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 alphanumeric editing, L_2 must be zero. Addressing options to the base address M_2 include only indexing.

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

EXAMPLE



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

The location TAG(5) contains the alphanumeric 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 alphanumeric 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=0). The source field must be an EBCDIC field when alphanumeric 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 1 is set if the source field is positive.
- Bit 2 is set if the source field is negative.
- Bit 3 is set if all source digits are zero.
- Bit 4 is always cleared.

The Condition register is not used or modified when alphanumeric 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 (double digit) 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 facilitiate 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).

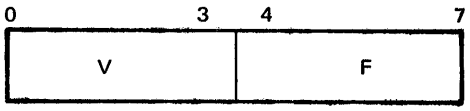
SG Sign Indicator

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

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 subcontrol information. Otherwise V is ignored.

The edit operators divide into three functional categories:

1. Data Transfer
2. Data Insert
3. 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 significant character within the source. It is set to zero if there is 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.

- V = 1, absolute set significance is performed:

  1. Set SD equal to one.
  2. Set current result field address -1 in general register 1 as float address.

- V = 0, conditional set significance is performed:

  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 but unpredictable results would be obtained.

Alphanumeric Editing

Alphanumeric editing is performed when $L_2 = 0$. The edit operators generally used are TE, IC, and MC.

The SSD, SFI, and ICS will function but are not generally of use in alphanumeric 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 general register 1.
- If V = 0 or SD = 1, no operation.

SFI – Set Fill (F = 6)

- Same as for numeric editing.

Unusual Conditions in Alphanumeric Editing – The following hexadecimal values are not legal in alphanumeric 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 alphanumeric editing and the SG does not contain a meaningful value.

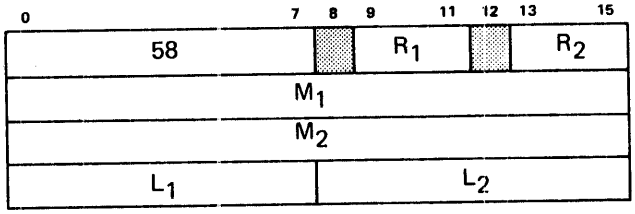
- F = 9, C, or D
  Treated as a normal Move Character (MC).

- 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 alphanumeric editing. General register 1 is set to zero upon return from alphanumeric 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 length of the result field (L_2) 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.

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.

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

- Bit 0 is always cleared.

- Bit 4 (invalid) 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 L_2 is exhausted.

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

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

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

CONSIDERATIONS

1. Zone of low order digit (sign) is the only one validated.

2. Packing continues until the length L_2 is exhausted.

3. If the number of packed digits in the receiving field ($2L_2$-1) is greater than the number of digits (L_1) of the sending field, zero fill is provided.

4. If the number of packed digits ($2L_2$-1) is less than the number of digits (L_1) of the sending field, truncation occurs but the overflow bit is not set.

5. The sign of the sending field (zone of the rightmost byte) becomes the sign of the receiving field.

6. A field may be packed to itself (if the receiving field length L_2 is at least as large as L_1).

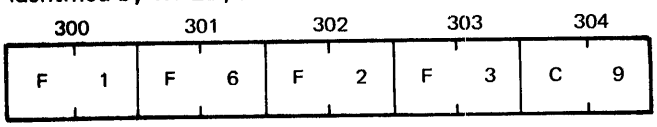
7. The packed result field always contains an odd number of digits (including significant and nonsignificant digits).

8. No significance in the result and a negative sending field sign generates a hex C sign and the setting of bit 3 of the Condition register.

9. If L_1 or L_2 is zero, bit 3 of the Condition register is set and bits 0-2 and 4 are cleared.

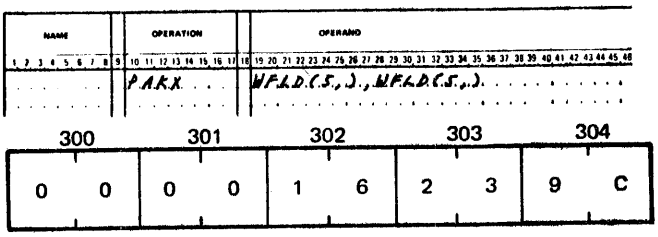
EXAMPLE

Assume that the following unpacked field is at an address identified by WFLD, and WFLD=300.



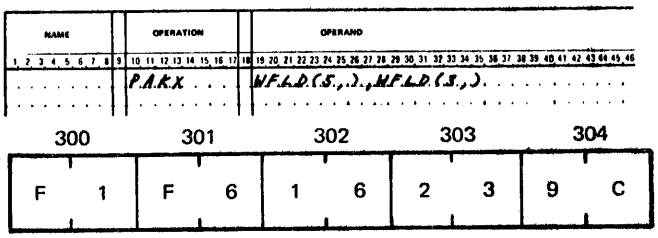
Unpacked
(L_1 = 5)

The following instruction shows how the field WFLD may be packed to itself; packing proceeds from right to left, and the unused bytes are filled with zeros.



Packed
(L_2 = 5)

The next instruction defines the result field as three bytes in length; the unused bytes (bytes 300 and 301) remain as they were in the original field.



Packed
(L_2 = 3)

SIGN RULES FOR PAKX AND UNPX

1. Valid signs for unpacked fields in hexadecimal, are: plus=F,A,C,E and minus=B,D.

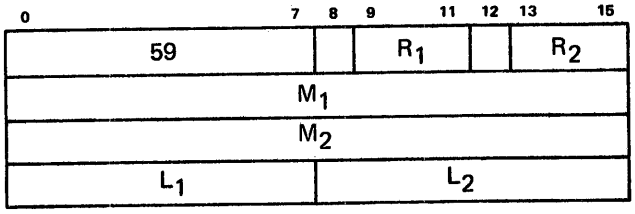
2. Valid signs for packed fields, in hexadecimal, are plus=F, A, C, E, 0, 2, 4, 6, 7, 8, and minus=B, D, 1, 3, 5, 9.

3. When packing is performed, the packed field will accept any valid sign in the unpacked field. Similarly, when unpacking, the unpacked field will accept any valid sign in the packed field.

4. The preferred signs are, in hexadecimal: plus=C, and minus=D. An arithmetic operation performed on a packed field will change any sign other than a preferred sign to the preferred C or D.

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.

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.

RESULTS: The zoned decimal field resides at the operand 2 location. The Condition register is affected as follows:

- Bits 0 and 4 (invalid) are always cleared.
- If no significance results bit 3 is set, bits 1 and 2 are cleared.
- 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.
- 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.

CONSIDERATIONS

1. No field validity checking is performed.

2. If the number of unpacked digits in the receiving field (L_2) is greater than the number of digits ($2L_1$-1) of the sending field, zero fill is provided.

3. If the number of unpacked digits (L_2) is less than the number of digits ($2L_1$-1) of the sending field, truncation occurs but the overflow bit is not set.

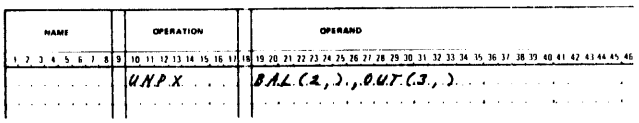
4. The sign of the sending field becomes the sign of the receiving field. No preferred sign is generated.

5. Unpacking a field to itself will result in a transposition of the sign and least significant digit, and an invalid result field if the field length L_2 is greater than 3.

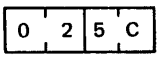
6. If L_1 or L_2 is zero, bit 3 of the Condition register is set and bits 0-2 and 4 are cleared.

EXAMPLE

The following instruction will unpack a field named BAL and place it in a field named OUT; the length of the packed field is 2 bytes, the unpacked field is 3 bytes.

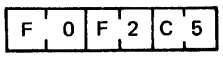


The packed field – BAL:



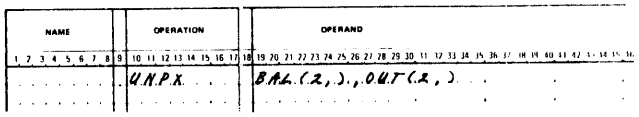
Packed
(L_1 = 2)

The resultant unpacked field – OUT:

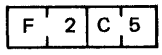


Unpacked
(L_2 = 3)

The unpacked field need only be 2 bytes in length; the next instruction defines OUT as 2 bytes long, while BAL is the same length as in the first example.



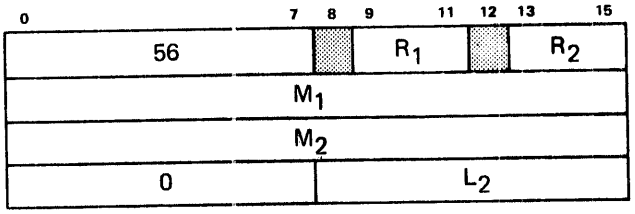
The resultant unpacked field – OUT:



Unpacked
(L_2 = 2)

Translate •

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



FUNCTION: Performs a byte-by-byte translation of the contents of a memory field, using a translate table. The first byte of the translate table is located at the address specified by the contents of general register 1. The table has an assumed length of 256 bytes.

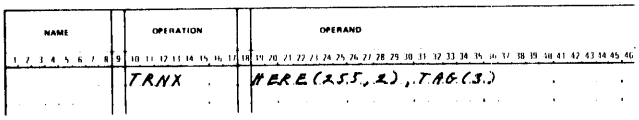
OPERAND 1: The field in memory that is to be translated. Addressing options include indexing, but not indirect addressing. The length of the M_1 field is always the same as the length of the M_2 field.

OPERAND 2: The field in memory that will hold the translated values at the conclusion of the operation. Addressing options include indexing, but not indirect addressing. The length (in bytes) of the M_2 field is always the same as the length of the M_1 field.

FIELD LENGTH: From 1 to 256 bytes may be translated. Because the 8-bit L_2 field cannot depict a value greater than 255, the number of bytes translated by this instruction is always one greater than the literal value of the L_2 field. If L_2=0, one byte will be translated; if L_2=255, 256 bytes will be translated.

RESULTS: The translated result field resides at the operand 2 location.

EXAMPLE



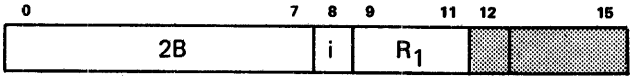
The 256 bytes identified by HERE(255,2) are translated byte-by-byte and placed in a field identified by TAG(3). Each byte value extracted from the HERE(255,2) field is entered into the table with the TAG(3) address added to it. Consequently, the table contains the addresses of the translated values, and therefore can be used as an index to them. (The address of the table is contained in register 1.)

DATA TRANSFER INSTRUCTIONS

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

Condition Register Load

CLDR @R_1



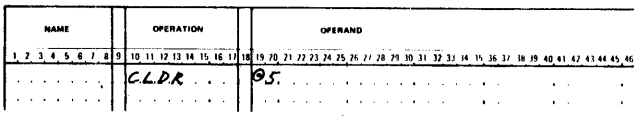
FUNCTION: Transfers the contents of a one-word field in a general register or in memory to the Condition register.

NOTE

Normally, bits in the Condition register are set or cleared by hardware to show the results following the execution of certain machine instructions (bits 0-7), or to identify a status condition that requires executive-program action (bits 12-15), such as a Bounds error. If any of the bits 12-15 in the field being transferred are true (on), false status conditions may be transmitted to the executive program when the Condition register is loaded. This caution does not apply if the field being transferred was originally generated by the CSTR instruction.

OPERAND: A one-word value in the general register specified by R_1, or in memory if indirect addressing is used.

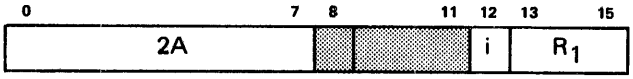
EXAMPLE



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

Condition Register Store

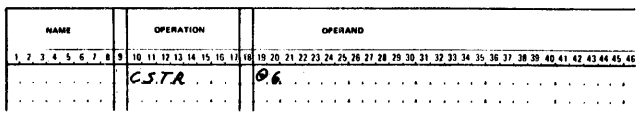
CSTR @R_1



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_1, or in memory if indirect addressing is used.

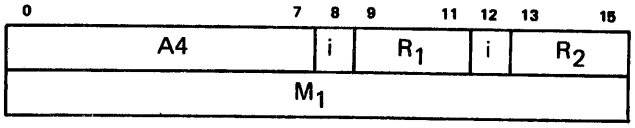
EXAMPLE



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

Inverse Move Memory – Register

INV @$M_1(R_1)$,@R_2



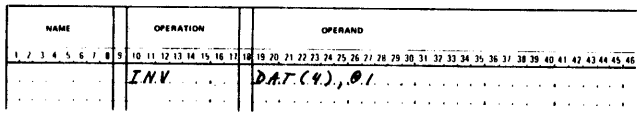
FUNCTION: Transfers the one's complement of a one-word field in memory to a one-word field in a general register or in memory.

OPERAND 1: The sending field; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

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

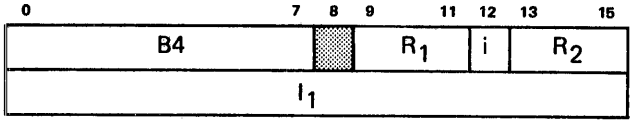
EXAMPLE



The 16-bit field identified by DAT(4) is transferred in one's complement format to the location specified in register 1.

Inverse Move Direct

INVD $I_1(R_1)$,@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.

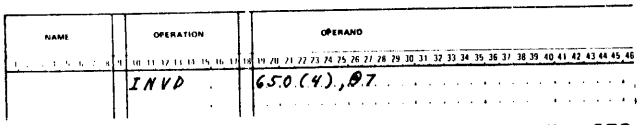
OPERAND 1: A 16-bit immediate value in the second word of the instruction; the value may range from 0-65,535.

Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I_1 value and the contents of the general register 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.

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

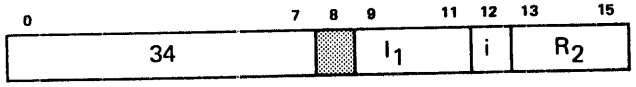
EXAMPLE



The one's complement of the value formed by adding 650 to the contents of register 4 is transferred to the location specified in register 7.

Inverse Move Immediate

INVI I_1,@R_2



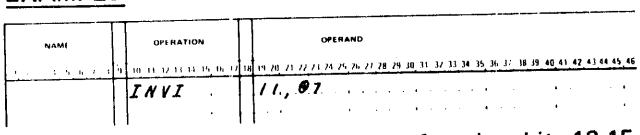
FUNCTION: Transfers the one's complement of a 4-bit immediate value to bits 12-15 of a one-word field in a general register or in memory. Bits 0-11 of the one-word field are always set to ones.

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_2, or in memory if indirect addressing is used.

RESULTS: The result field resides in bits 12-15 of operand 2.

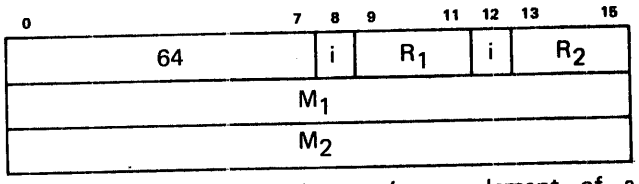
EXAMPLE



The one's complement of 11 is transferred to bits 12-15 of a 16-bit field located at the address specified in register 7. 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)



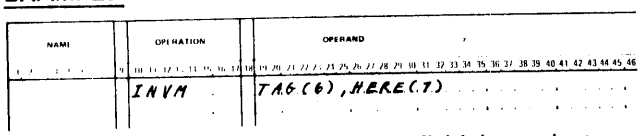
FUNCTION: Transfers the one's complement of a one-word field in memory to another one-word field in memory.

OPERAND 1: The sending field; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The receiving field; a one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

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

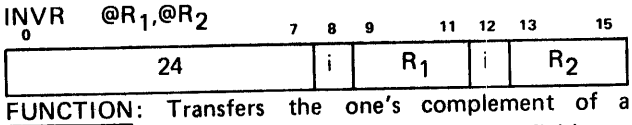
EXAMPLE



The one's complement of a 16-bit field located at an address identified by TAG(6) is transferred to the field at the address identified by HERE(7).

Inverse Move Register — Register

INVR @R_1,@R_2



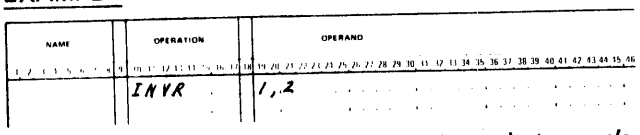
FUNCTION: Transfers the one's complement of a one-word field to another one-word field; either field may be in a register or in memory.

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

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

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

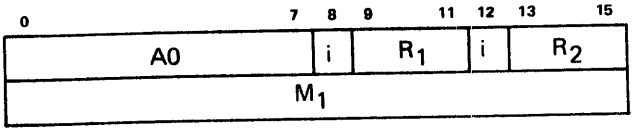
EXAMPLE



The contents of register 1 are converted to one's complement format and stored in register 2.

Load Memory — Register

LOD @M_1(R_1),@R_2



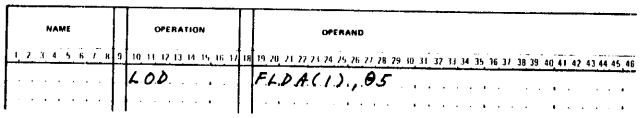
FUNCTION: Transfers the contents of a one-word field in memory to a one-word field in a general register or in memory.

OPERAND 1: The sending field; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, 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.

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

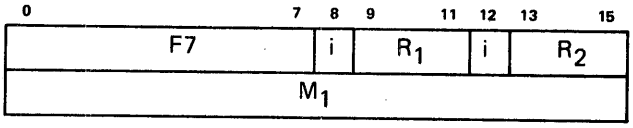
EXAMPLE



FLDA(1) is the address of a 16-bit value which is transferred to another location specified by the address in register 5.

Load Byte •

LODB @M_1(R_1),@R_2



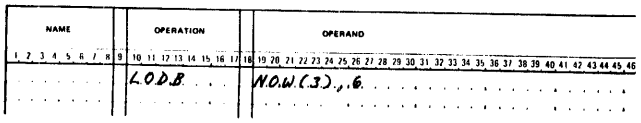
FUNCTION: Transfers a one-byte field in memory to bits 8-15 of a one-word field in a general register or to a one-byte field in memory.

OPERAND 1: The sending field; a one-byte field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The receiving field; a one-word field in the general register specified by R_2, or a one-byte field in memory if indirect addressing is used. 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 result field resides at the operand 2 location.

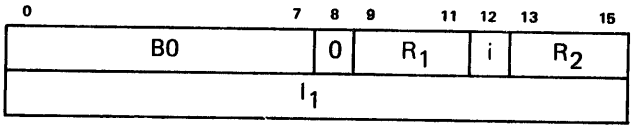
EXAMPLE



NOW(3) yields the address of a one-byte field in memory which is transferred to bits 8-15 of register 6.

Load Direct

LODD I_1(R_1),@R_2 or M_1(R_1),@R_2



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

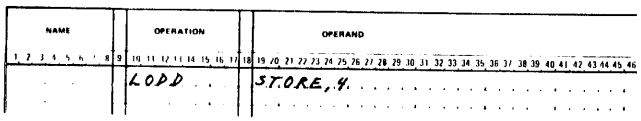
OPERAND 1: The sending field; a 16-bit immediate value in the second word of the instruction; the value may range from 0-65,535. If an address symbol is used in operand 1 (M_1), the address of that field is used in the load operation.

Indexing may be specified for operand 1. In this case, operand 1 is derived by adding the I_1 value and the contents of the index register specified by R_1; no check for overflow or link is made during the indexing.

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.

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

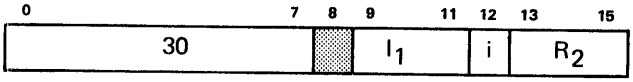
EXAMPLE



The address of the field named STORE (not the actual field) is loaded into register 4.

Load Immediate

LODI I_1,@R_2



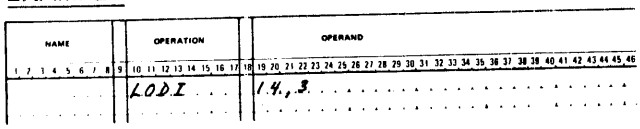
FUNCTION: Transfers a 4-bit immediate value to bits 12-15 of a one-word field in a general register or in memory.

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

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

RESULTS: The result field 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.

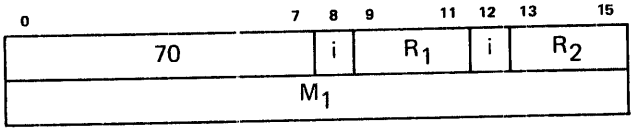
EXAMPLE



The immediate value 14 is loaded into register 3. The result in memory appears as follows: 0000000000001110.

Load Two-Word

LODT @M_1(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: The sending field; a two-word field in memory beginning at the specified effective address. The most significant bits are at this address.

Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

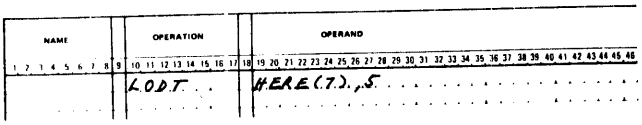
OPERAND 2: The receiving field; a two-word field located in two general registers or in memory.

If direct addressing is used, the field is in the register specified by R_2 and the next highest register, R_2+1; the most significant bits are in the R_2 register. (Note: If register 7 is specified by R_2, the field is in registers 7 and 0 with the most significant bits in register 7.)

If indirect addressing is used, the field is in memory beginning at the address in the R_2 register; the most significant bits are at this address.

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

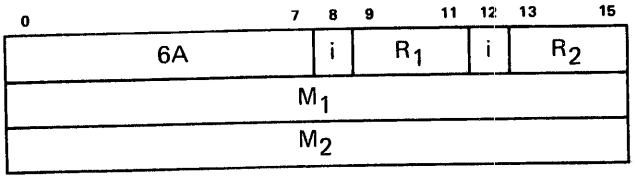
EXAMPLE



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)



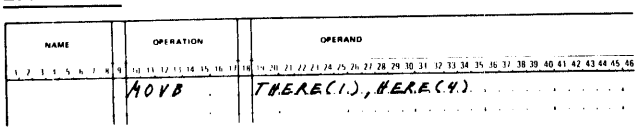
FUNCTION: Transfers a one-byte field in memory to another one-byte field in memory.

OPERAND 1: The sending field; a one-byte field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The receiving field; a one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

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

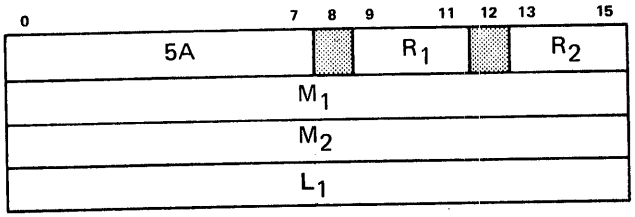
EXAMPLE



A byte at the address identified by THERE(1) is moved to the location identified by HERE(4).

Move Long •

MOVL M_1(L_1,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: The sending field, moved one byte at a time. The field length, 0-65,535 bytes, is specified by the L_1 value in the instruction. Addressing options to the base address M_1 include only indexing. The effective address points to the most significant byte of the field.

OPERAND 2: The receiving field. The field length, 0-65,535 bytes, is specified by the L_1 value in the

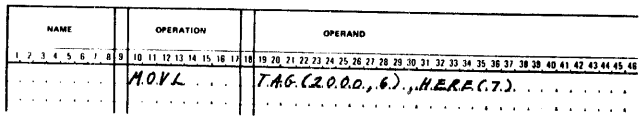
instruction. Addressing options to the base address M_2 include only indexing. The effective address points to the most significant byte of the field.

RESULTS: The result field resides at the operand 2 location. The sending field and receiving field may overlap. If $L_1 = 0$, no move of data is executed.

CONSIDERATIONS

A word move is performed if L_1 is even and the beginning addresses of both fields are even.

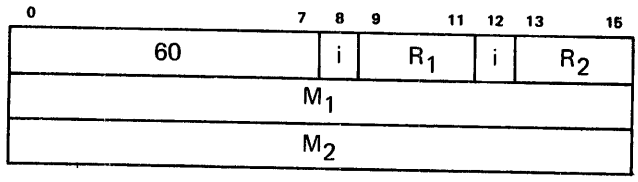
EXAMPLE



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)



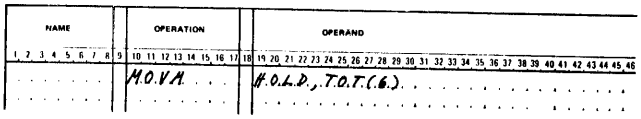
FUNCTION: Transfers the contents of a one-word field in memory to another one-word field in memory.

OPERAND 1: The sending field; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The receiving field; a one-word field in memory. Addressing options to the base address M_2 include indexing, indirect addressing, or a combination of both.

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

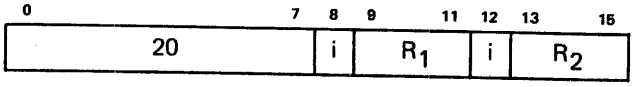
EXAMPLE



A 16-bit field identified by HOLD is moved to the field identified by TOT(6).

Move Register — Register

MOVR @R_1,@R_2



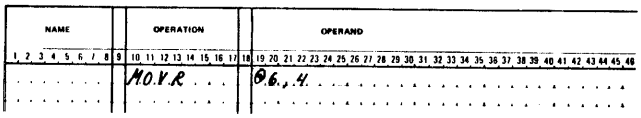
FUNCTION: Transfers the contents of a one-word field to another one-word field; either field may be in a general register or in memory.

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

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.

RESULTS: The moved value is at the operand 2 location.

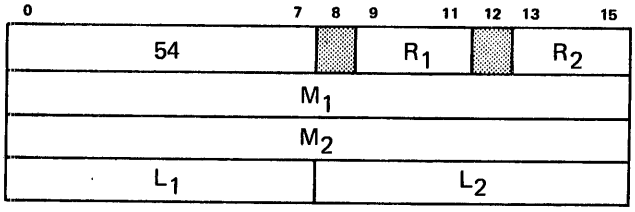
EXAMPLE



A 16-bit field at the address specified in register 6 is moved into register 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 to another field in memory; the field lengths may vary from 0-255 bytes.

OPERAND 1: The sending field. 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.

OPERAND 2: The receiving field. 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.

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

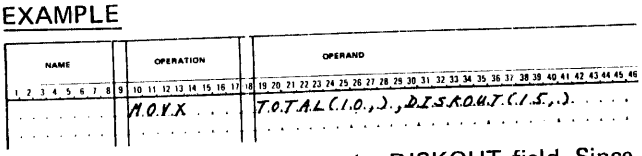
- If $L_1 = L_2$, the number of bytes specified by L_1 is transferred.
- If L_1 is less than L_2, the number of bytes specified by L_1 is transferred, then blanks are used to fill operand 2.

- 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.

CONSIDERATIONS

A word move is performed if, and only if, the lengths L_1 and L_2 and the effective addresses $M_1 + (R_1)$ and $M_2 + (R_2)$ are both even.

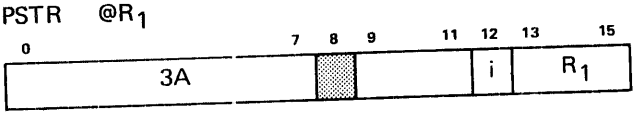
EXAMPLE



The TOTAL field is moved to the DISKOUT field. Since DISKOUT is larger than TOTAL, the rightmost 5 bytes of DISKOUT are filled with blanks.

Program Address Store

PSTR @R_1

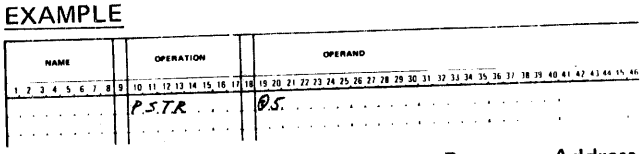


FUNCTION: Transfers the current program address to a one-word field in a register or in memory.

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

RESULTS: The current program address resides at the operand 1 location.

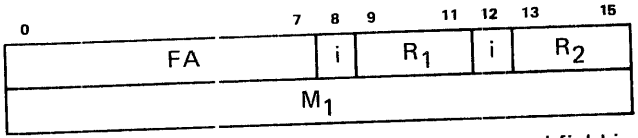
EXAMPLE



Transfers the current value of the Program Address register to the address specified in register 5.

Store Memory – Register

STO @M_1(R_1),@R_2



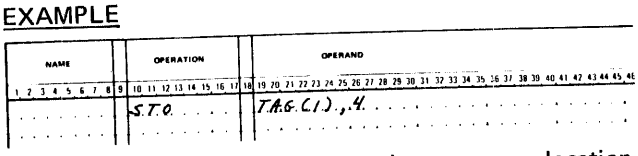
FUNCTION: Transfers the contents of a one-word field in a general register or in memory to another one-word field in memory.

OPERAND 1: The receiving field; a one-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

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

RESULTS: The field is stored in the operand 1 location.

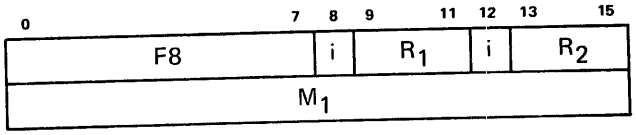
EXAMPLE



The contents of register 4 are stored at a memory location identified by TAG(1).

Store Byte •

STOB @M_1(R_1),@R_2



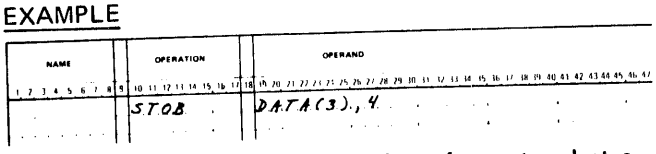
FUNCTION: Transfers the contents of bits 8-15 of a general register or a one-byte field in memory to a one-byte field in memory.

OPERAND 1: The receiving field, a one-byte field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The sending field; bits 8-15 of the general register specified by R_2, or a one-byte field in memory if indirect addressing is used.

RESULTS: The stored byte resides at the operand 1 location.

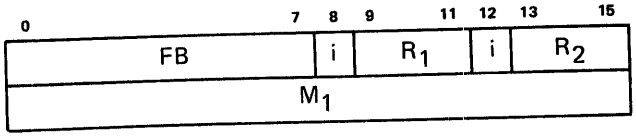
EXAMPLE



The contents of bits 8-15 of register 4 are stored at a memory location identified by DATA(3).

Store Two-Word

STOT @M_1(R_1),@R_2



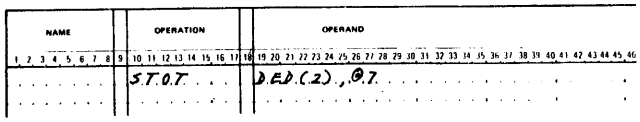
FUNCTION: Transfers the contents of a two-word field in two general registers or in memory to a two-word field in memory.

OPERAND 1: The receiving field; a two-word field in memory. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: The sending field; a two-word field in general registers R_2 (more-significant word) and R_2+1 (less significant). If indirect addressing is used, the registers contain the memory addresses of the words, in the same relationship as for direct addressing. (Note: If register 7 is specified as R_2, the field is in register 7 and 0, with 7 containing the more-significant word or address.)

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

EXAMPLE



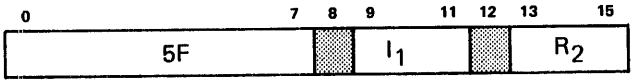
A 32-bit field, the address of which is specified in register 7, is stored in memory at a location identified by DED(2).

SHIFT INSTRUCTIONS

Mnemonic	Name
ARDI	Arithmetic Right Double Shift – Immediate
ARDR	Arithmetic Right Double Shift – Register
ARSI	Arithmetic Right Single Shift – Immediate
ARSR	Arithmetic Right Single Shift – Register
LLDI	Logical Left Double Shift – Immediate
LLDR	Logical Left Double Shift – Register
LLSI	Logical Left Single Shift – Immediate
LLSR	Logical Left Single Shift – Register
LRDI	Logical Right Double Shift – Immediate
LRDR	Logical Right Double Shift – Register
LRSI	Logical Right Single Shift – Immediate
LRSR	Logical Right Single Shift – Register
RLDI	Rotating Left Double Shift – Immediate
RLDR	Rotating Left Double Shift – Register
RLSI	Rotating Left Single Shift – Immediate
RLSR	Rotating Left Single Shift – Register
SHFK	Shift Packed Decimal •

Arithmetic Right Double Shift – Immediate

ARDI I_1,R_2

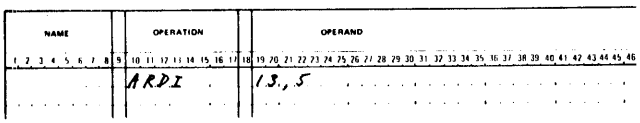


FUNCTION: Performs an arithmetic right shift of a two-word field in two general registers. The sign (bit 0) of the field is extended. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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 register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

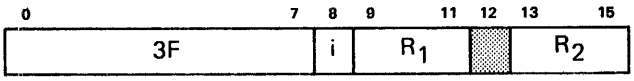
EXAMPLE



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 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.

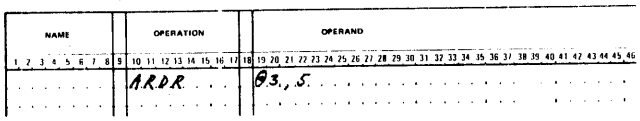
OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1, or in memory if indirect addressing is used. The shift count may range from 0-15.

OPERAND 2: A two-word field in two general registers: the register specified by R_2 and the next highest register,

R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

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 R_2+1 register is lost.

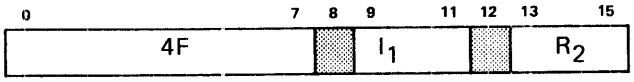
EXAMPLE



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 6 is lost.

Arithmetic Right Single Shift — Immediate

ARSI I_1,R_2

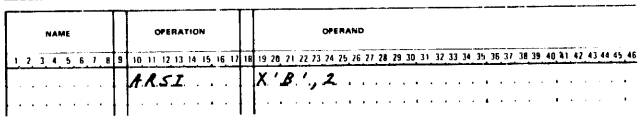


FUNCTION: Performs an arithmetic right shift of a field in a general register. The sign (bit 0) of the field is extended. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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_2. 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.

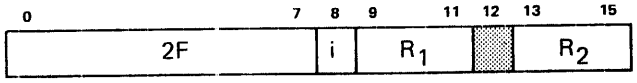
EXAMPLE



Shifts the data in register 2 to the right 11 bit positions. The sign is extended from bits 0-11; any data shifted out of register 2 is lost.

Arithmetic Right Single Shift — by Register

ARSR @R_1,R_2

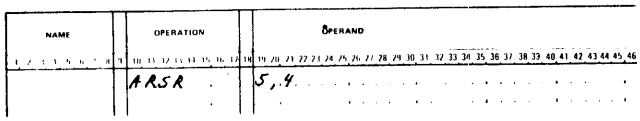


FUNCTION: Performs an arithmetic right shift of a one-word field in a general register. The sign (bit 0) of the field is extended. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1, or in memory if indirect addressing is used. 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.

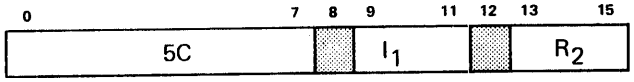
EXAMPLE



Shifts the data in register 4 to the right; register 5 contains the shift count. The sign is extended, and data shifted out of register 4 is lost.

Logical Left Double Shift — Immediate

LLDI I_1,R_2



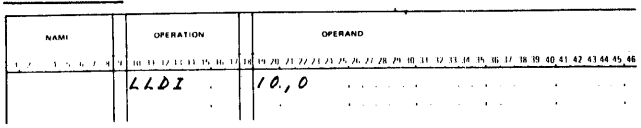
FUNCTION: Performs a left shift (zero fill from right) of a two-word field in two general registers. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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 register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

The data is shifted to the left the number of positions specified by operand 1. Any data shifted out of the R_2 register is lost.

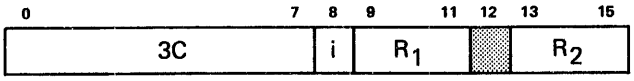
EXAMPLE



Shifts the data in registers 0 and 1 to the left 10 bit positions. Data shifted out of register 0 is lost.

Logical Left Double Shift – by Register

LLDR @R_1,R_2



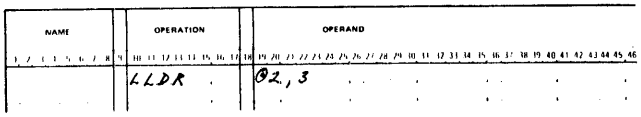
FUNCTION: Performs a left shift (zero fill from right) of a two-word field in two general registers. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1, or in memory if indirect addressing is used. The shift count may range from 0-15.

OPERAND 2: A two-word field in two general registers: the register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

The data is shifted to the left the number of positions specified by operand 1. Any data shifted out of the R_2 register is lost.

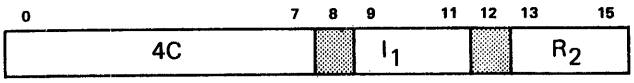
EXAMPLE



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 Single Shift – Immediate

LLSI I_1,R_2

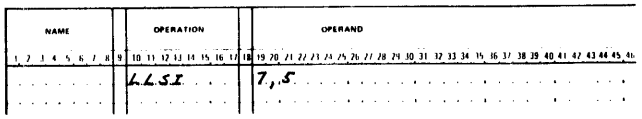


FUNCTION: Performs a left shift (zero fill from right) of a one-word field in a general register. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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_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.

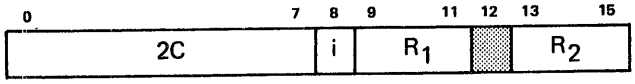
EXAMPLE



Shifts the data in register 5 to the left 7 bit positions. Data shifted out of register 5 is lost.

Logical Left Single Shift – by Register

LLSR @R_1,R_2

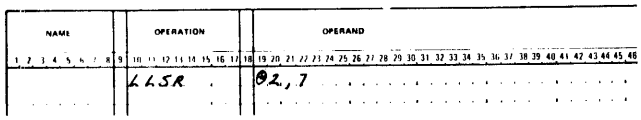


FUNCTION: Performs a left shift (zero fill from right) of a one-word field in a general register. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value in the general register specified by R_1, or in memory if indirect addressing is used. 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.

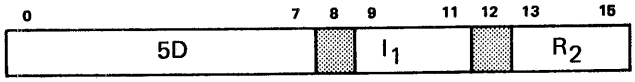
EXAMPLE



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 Right Double Shift – Immediate

LRDI I_1,R_2



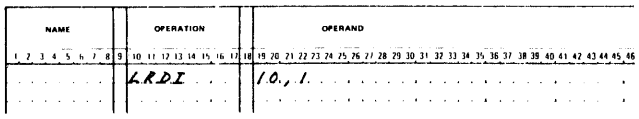
FUNCTION: Performs a right shift (zero fill from left) of a two-word field in two general registers. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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 register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

The data is shifted to the right the number of positions specified by operand 1. Any data shifted out of the R_2+1 register is lost.

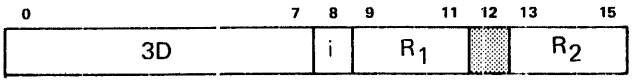
EXAMPLE



Shifts the data in registers 1 and 2 to the right 10 bit positions. Data shifted out of register 1 is lost.

Logical Right Double Shift — by Register

LRDR @R_1,R_2



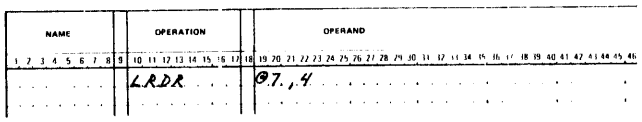
FUNCTION: Performs a right shift (zero fill from left) of a two-word field in two general registers. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1, or in memory if indirect addressing is used. The shift count may range from 0-15.

OPERAND 2: A two-word field in two general registers: the register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

The data is shifted to the right the number of positions specified by operand 1. Any data shifted out of the R_2+1 register is lost.

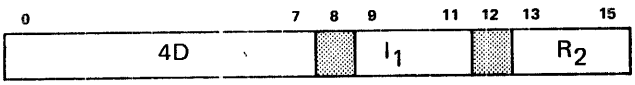
EXAMPLE



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 Single Shift — Immediate

LRSI I_1,R_2

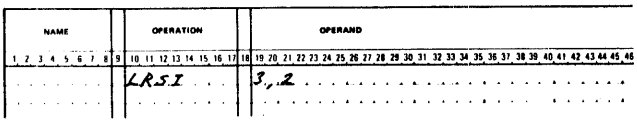


FUNCTION: Performs a right shift (zero fill from left) of a one-word field in a value.

OPERAND 1: The shift count; 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_2. The data is shifted to the right the number of positions specified by operand 1. Any data shifted out of the register is lost.

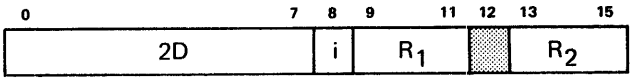
EXAMPLE



Shifts the data in register 2 to the right 3 bit positions. Data shifted out of the register is lost.

Logical Right Single Shift — by Register

LRSR @R_1,R_2

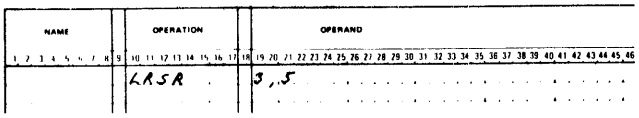


FUNCTION: Performs a right shift (zero fill from left) of a one-word field in a general register. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used. 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. Any data shifted out of the register is lost.

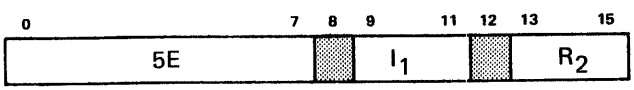
EXAMPLE



Shifts the data in register 5 to the right; the shift count is in register 3. Data shifted out of register 5 is lost.

Rotating Left Double Shift — Immediate

RLDI I_1,R_2



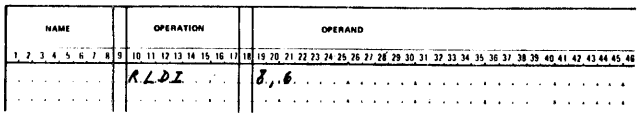
FUNCTION: Performs a rotating left shift of a two-word field in two general registers. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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 register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

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 R_2 register is brought back in the right end of the R_2+1 register. Bits shifted out of the lower register are not lost.

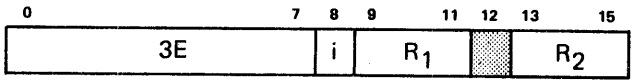
EXAMPLE



Shifts the data in registers 6 and 7 to the left 8 bit positions. Each bit shifted out of the left end of register 6 is brought back in the right end of register 7.

Rotating Left Double Shift — by Register

RLDR @R_1,R_2



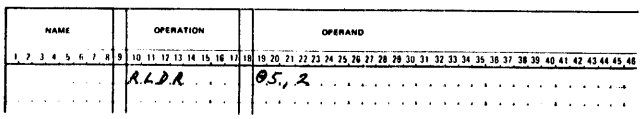
FUNCTION: Performs a rotating left shift of a two-word field in two general registers. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1, or in memory if indirect addressing is used. The shift count may range from 0-15.

OPERAND 2: A two-word field in two general registers: the register specified by R_2 and the next highest register, R_2+1. The most significant bits are in the R_2 register. (Note: If register 7 is specified as R_2, the field is in registers 7 and 0, with the most significant bits in register 7.)

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 R_2 register is brought back in the right end of the R_2+1 register. Bits shifted out of the register are not lost.

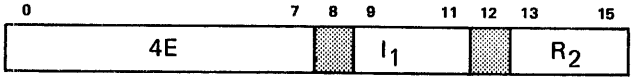
EXAMPLE



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 Single Shift — Immediate

RLSI I_1,R_2

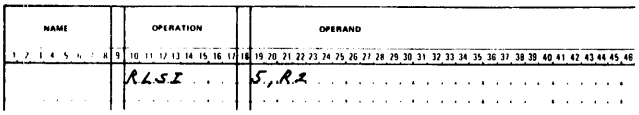


FUNCTION: Performs a rotating left shift of a one-word field in a general register. The shift count is a 4-bit immediate value.

OPERAND 1: The shift count; 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_2. 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.

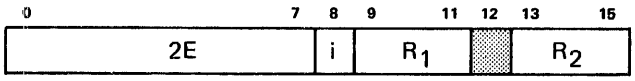
EXAMPLE



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.

Rotating Left Single Shift — by Register

RLSR @R_1,R_2

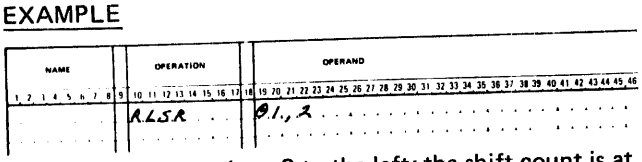


FUNCTION: Performs a rotating left shift of a one-word field in a general register. The shift count is a 4-bit field in a general register or in memory.

OPERAND 1: The shift count; a 4-bit unsigned value located in the general register specified by R_1 or in memory if indirect addressing is used. 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.

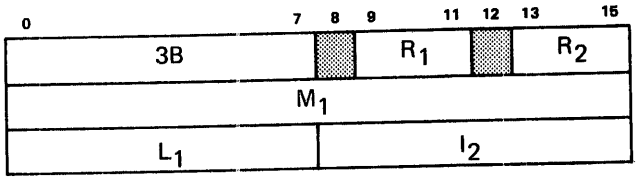
EXAMPLE



Shifts the data in register 2 to the left; the shift count is at the address specified in register 1. Data shifted out the left end of the register is brought back in the right end of the register.

Shift Packed Decimal

SHFK $M_1(L_1,R_1),I_2(R_2)$



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 (in bytes) is in L_1; this value may range from 0-255. Addressing options to the base address M_1 include only indexing.

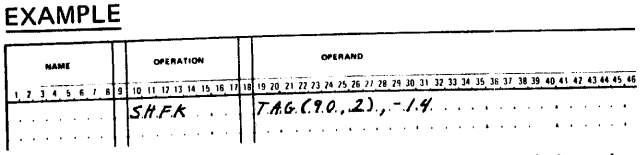
OPERAND 2: The shift count is the I_2 value in the instruction, a signed value from -128 to +127. If a register is specified in R_2, the contents of the register are added to the I_2 value to form the shift count.

A right shift is indicated in the source statement by a minus sign preceding the I_2 value. A left shift requires no sign preceding I_2, although a plus sign may be used. The shift count indicates the number of digits to be shifted. Note, however, that the value of R_2 could change the shift direction, because the *effective* shift count determines direction.

RESULTS: The Condition register is affected as follows:

- 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 $I_2 = 0$; otherwise bit 3 is cleared.

EXAMPLE



The contents of location TAG are modified by the contents of register 2 to identify a 90-byte packed decimal field, which is shifted 14 bit positions to the right.

CONSIDERATIONS

1. Validity of digits is not checked.
2. Invalid digits are propagated.
3. The sign is not changed unless all significance is right shifted out and the sign was originally minus. Under these circumstances the sign is changed to a hexadecimal C (positive).
4. Maximum shift count is left 127 to right 128 digits if $R_2 = 0$; if $R_2 \neq 0$, maximum shift count is left 32,767 to right 32,768 digits.

FLOATING-POINT INSTRUCTIONS

The optional floating-point instructions provide all the versatility required in manipulating floating-point values and expand the computational capacity of the processing unit with a minimum expansion of execution time. These ten instructions allow floating-point values to be converted, compared, transferred, and combined arithmetically.

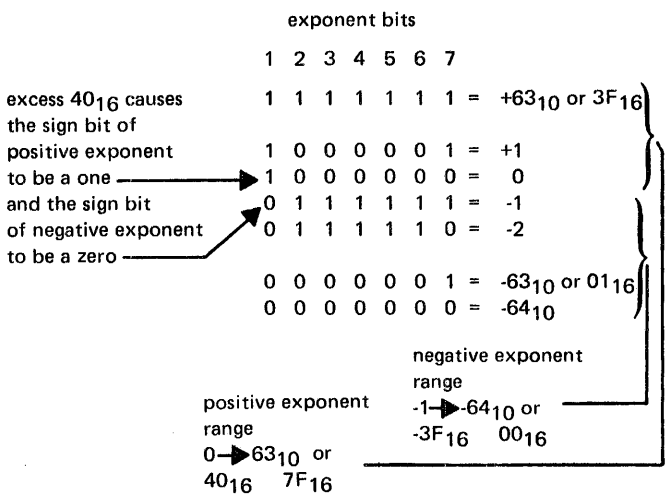
Mnemonic	Name
ADDF	Add Floating Point
CMPF	Compare Floating Point
DIVF	Divide Floating Point
FLTF	Convert Fixed to Float
INTF	Convert Float to Fixed
LODF	Load Floating Point Register
MPYF	Multiply Floating Point
NEGF	Negate Float Point Register
STOF	Store Floating
SUBF	Subtract Floating Point

The following discussion and figures explain particularities of floating-point usage.

DATA FORMAT

The floating-point number consists of a 64-bit fixed-length format as shown in Figure 4-1. Both the exponent and the fraction are signed values.

The floating-point value is the product of its fraction, and base 16 raised to the power of its exponent: fraction x 16^{exp}. The seven bits of the exponent allow a range of 0-127. The exponent is expressed in "excess 64 (40_{16})" notation, resulting in both a positive and negative range as shown below.



The fractional portion of the floating-point number is always represented in positive notation (a true fraction) but with its true sign indicated in bit zero. The base point is assumed to be at the left of the highest-order hex digit of the fraction.

NORMALIZATION

A floating-point number is normalized when the highest-order hex digit of the fraction is not zero. Floating-point instructions do not normalize their operands before execution; therefore, to avoid loss of significance, operands should be normalized prior to a floating-point instruction. However, all floating-point instructions except *Compare* and *Divide* will perform correctly with unnormalized operands and will produce normalized results.

ZERO REPRESENTATION

To express a true floating-point zero, all 64 bits are zero. Also, a zero fraction result, generated by a floating-point operation, forces the exponent and the sign (bit 0) to zero.

A zero participates normally in all arithmetic operations except a floating-point divide. Attempting to divide by a true zero or with a zero fraction in the divisor causes an arithmetic exception to be flagged in the Condition register, and no actual division occurs.

ARITHMETIC EXCEPTIONS

Conditions may occur during the execution of floating-point instructions which cause exceptions to normal completion. These arithmetic exceptions are flagged in the Condition register. The possible arithmetic exceptions and the instructions during which they may occur are as follows:

Exception	Floating Point Instruction
Exponent Overflow	*Add, Subtract, Multiply,* and *Divide*
Exponent Underflow	*Add, Subtract, Multiply, Divide,* and *Load*
Zero Divisor	*Divide*
Integer Overflow	*Convert Float to Fixed*

The user of floating-point instructions can detect arithmetic exceptions by checking the Condition register after execution, or by providing an arithmetic exception address pointer within the instruction itself. With the latter procedure, when an arithmetic exception occurs, control is passed to the exception address; the normal read-next-instruction address than is placed in the exception address general register.

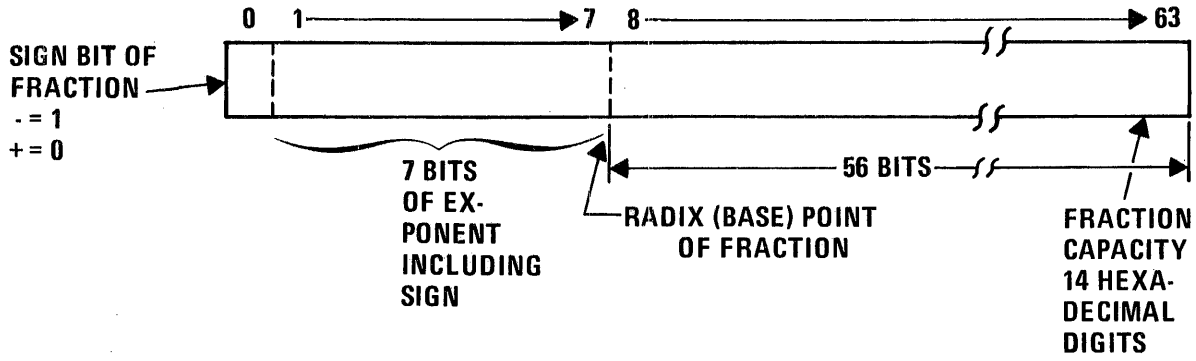


Figure 4-1. Floating-Point Data Format

The floating-point arithmetic exceptions are denoted by the setting of bit position 0 and the exception code, bit positions 1-3 of the Condition register. A floating-point *Compare* instruction sets bit groups 0-3 and 4-7 as shown in the note of Figure 1-4. The following table gives all the Condition register settings possible as a result of a floating-point instruction, except for *Compare* instructions.

Bit Positions Set	Condition Indicated
3	Result is zero
2	Result is less than zero
1	Result is greater than zero
0	Integer overflow on a *Convert Float to Fixed* instruction.
0 and 1	Exponent overflow
0 and 2	Exponent underflow
0 and 3	Zero divisor

FLOATING-POINT REGISTER

A floating-point register consists of four consecutive registers in the basic register file for each of the eight processor states. The register format is shown in Figure 4-2.

All floating-point instructions make reference to a floating-point register implicitly. The floating-point register is accessible only by micro-instructions.

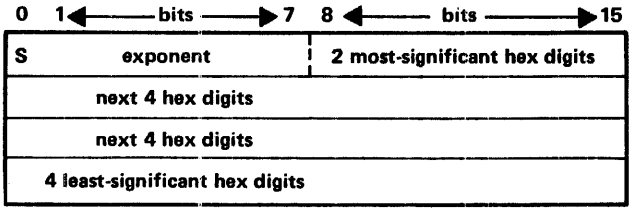
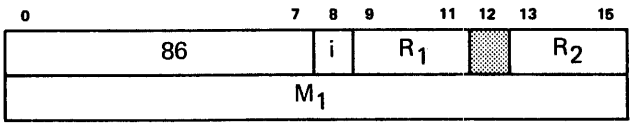


Figure 4-2. Floating-Point Register Format

Add Floating Point

ADDF @$M_1(R_1)$,R_2



FUNCTION: Performs an algebraic addition of a four-word floating point value in memory and the contents of the floating point register. Both operands may be normalized or unnormalized before the addition.

First, the exponents of the two operands are compared; if they do not agree, the smaller exponent is increased by one each time its fraction is right shifted one hexadecimal digit, until the two exponents agree. Next the fractions are algebraically added to form an intermediate 15 hexadecimal digit sum with a possible overflow carry.

If an overflow carry is used in generating the intermediate sum, the sum is right shifted one hexadecimal digit and the exponent increased by one. If the increased exponent is greater than +63, an exponent-overflow arithmetic exception occurs.

After the intermediate sum is formed, either including the carry maneuver or without, it is normalized. If the normalizing causes an exponent less than -64, an exponent-underflow arithmetic exception occurs. Finally, the 15 hexadecimal digit intermediate sum is right shifted to form the 14-digit resultant fraction, without rounding.

OPERAND 1: A four-word field in memory beginning at the effective word address. The effective address points to the most significant word of the four-word floating point number. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2. This field is the arithmetic exception address to which control is transferred if arithmetic exception occurs.

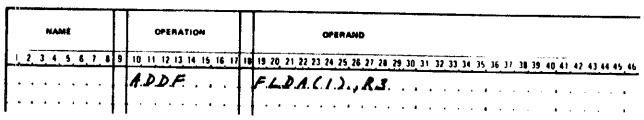
RESULTS: The sum resides in the floating point register as a normalized number. The Condition register is affected as follows:

- Bit 1 is set if the normalized number > 0.
- Bit 2 is set if the normalized number < 0.
- Bit 3 is set if the normalized number = 0.

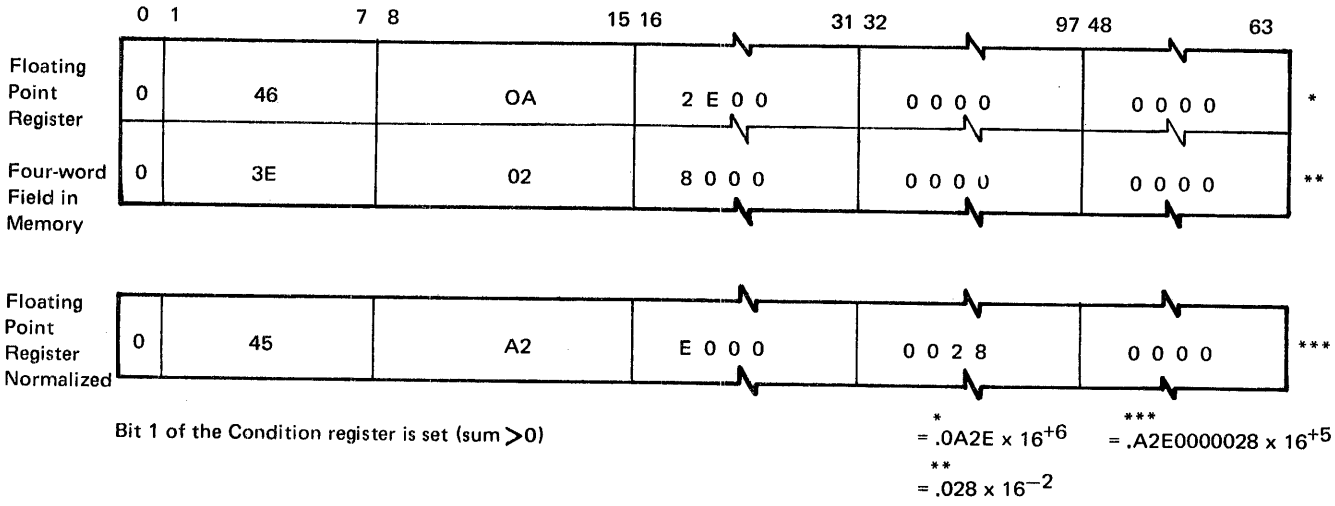
- Bits 0 and 2 are set if exponent underflow occurs during normalization. The sign and fraction of the resultant number are correct, but the exponent is too large by a value of 128.

- Bits 0 and 3 are set if exponent overflow occurs during the sum determination. The sign and fraction of the resultant number are correct, but the exponent is too small by a value of 128.

EXAMPLE

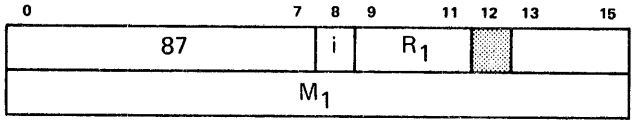


The field identified by FLDA(1) is added to the field in the floating point register; results are in the floating point register. General register 3 contains the arithmetic exception address.



Compare Floating Point

CMPF @$M_1(R_1)$



FUNCTION: Performs an algebraic comparison between the contents of the floating point register and a floating point number in a four-word field in memory. Both operands are assumed to be normalized. Invalid compare results may occur if they are not. Operands with zero fractions compare equal even though the signs and exponents may differ.

OPERAND 1: The four-word field in memory located at the effective address. The effective address points to the most significant word of the four-word memory field. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

RESULTS: Neither operand is changed. The Condition register is affected as follows:

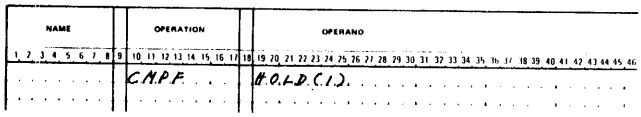
- Bits 0 and 4 are cleared.

- Bits 3 and 7 are set if the operands are equal. Bits 1, 2, 5, and 6 are cleared.

- Bits 2 and 6 are set if the operand in memory is less than the contents of the floating point register. Bits 1, 3, 5, and 7 are cleared.

- Bits 1 and 5 are set if the operand in memory is greater than the contents of the floating point register. Bits 2, 3, 6, and 7 are cleared.

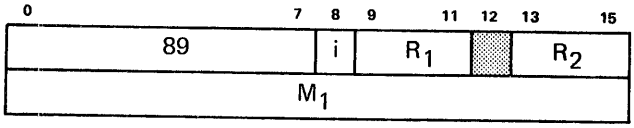
EXAMPLE



The four-word field identified by HOLD(1) is compared to the contents of the floating point register; the Condition register is set accordingly.

Divide Floating Point

DIVF @$M_1(R_1),R_2$



FUNCTION: Divides the contents of the floating point register (dividend) by a four-word floating point number in memory (divisor) and places the quotient in the floating point register. The quotient is a normalized value and the remainder is not saved. The divisor and the dividend must be normalized prior to the execution of the DIVF instruction.

The floating point division consists of three operations:

1. Subtracting the divisor exponent from the dividend exponent and adding 64 (40_{16}) because of the excess 64 exponent notation.

2. Division of the fractions with the resultant quotient sign, following the rules for signed numbers.

3. Quotient normalization.

An exponent-overflow arithmetic exception occurs if the final quotient exponent exceeds +63 and an exponent-underflow arithmetic exception occurs if the final exponent is less than -64. Also, a division-by-zero arithmetic exception occurs when a division with a zero divisor is attempted. In this case the dividend in the floating point register is unaltered.

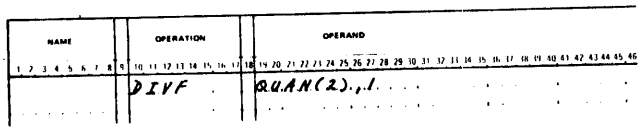
OPERAND 1: A four-word field in memory beginning at the effective word address. The effective address points to the most significant word of the four-word floating point number. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2. This field is the arithmetic exception address to which control is transferred if arithmetic exception occurs.

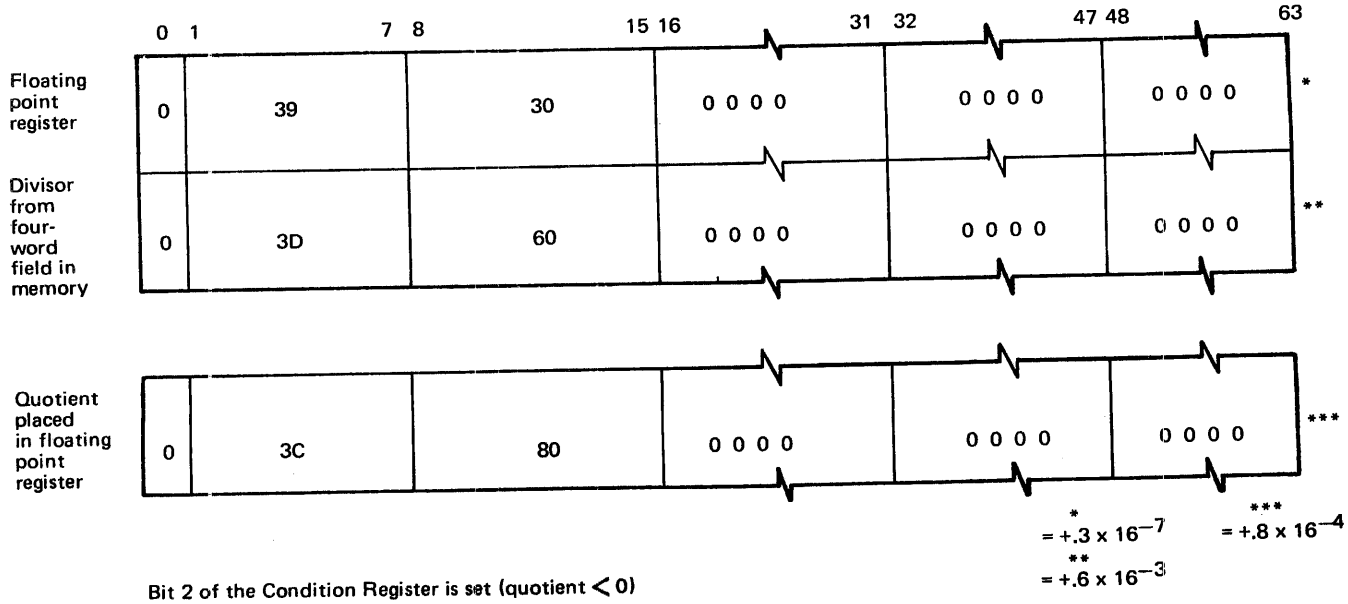
RESULTS: The normalized quotient resides in the floating point register. The Condition register is affected as follows:

- Bit 1 is set if the normalized quotient>0.
- Bit 2 is set if the normalized quotient<0.
- Bit 3 is set if the normalized quotient = 0.
- Bits 0 and 2 are set if exponent underflow occurs during normalization or exponent determination. The sign and the fraction of the dividend are correct, but the exponent is too large by a value of 127.
- Bits 0 and 3 are set if exponent overflow occurs during exponent determination. The sign and the fraction of the dividend are correct, but the exponent is too small by a value of 127.
- Bits 0 and 3 are set if a division by zero is attempted.

EXAMPLE



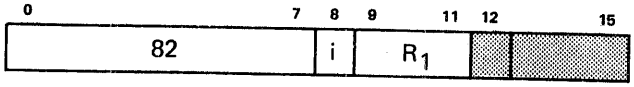
This instruction divides the contents of the floating point register by the value at the address identified by QUAN(2). Register 1 contains the arithmetic exception address.



Bit 2 of the Condition Register is set (quotient < 0)

Convert Fixed to Float

FLTF @R_1



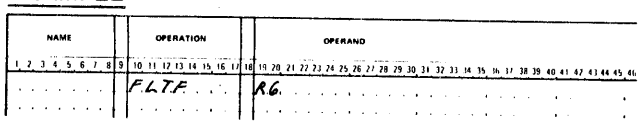
FUNCTION: Converts a 16-bit signed integer value to a normalized floating point number and places it in the floating point register.

OPERAND 1: A one-word sending field in the general register specified by R_1 or at the address specified by the contents of R_1 if indirect addressing is used. This field contains the signed integer value.

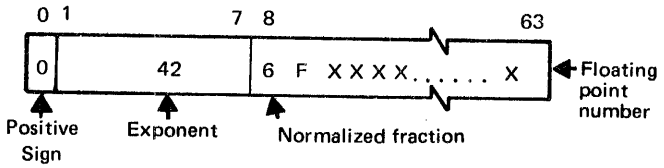
RESULTS: The Condition register is affected as follows:

- Bit 1 is set if the converted floating number > 0.
- Bit 2 is set if the converted floating number < 0.
- Bit 3 is set if the converted floating number = 0.

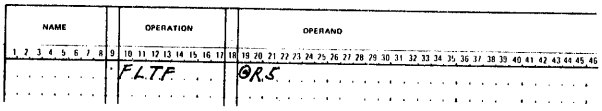
EXAMPLE



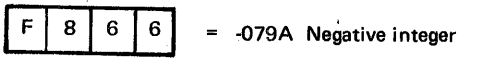
A positive fixed-point value in register 6 will be converted to floating point and placed in the floating point register.

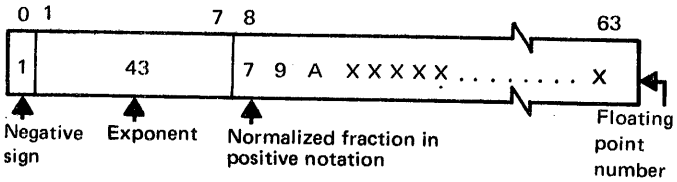


EXAMPLE



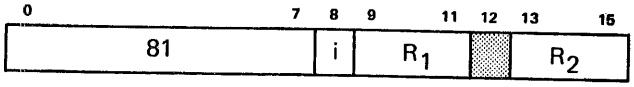
In this example, a negative number at the address in register 5 is converted from fixed to floating point.





Convert Float to Fixed

INTF @R_1,R_2



FUNCTION: Converts the floating-point number in the floating-point register to a 16-bit fixed point signed integer value without rounding. Arithmetic exception occurs if the floating point value is too large (greater than +32,767).

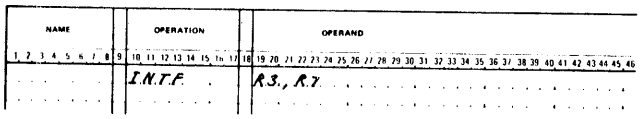
OPERAND 1: A one-word receiving field in the general register specified by R_1, or in memory if indirect addressing is used.

OPERAND 2: A one-word field in the general register specified by R_2. This field is the arithmetic exception address, to which control is transferred rather than to RNI if the floating point number is out of the expressable integer range.

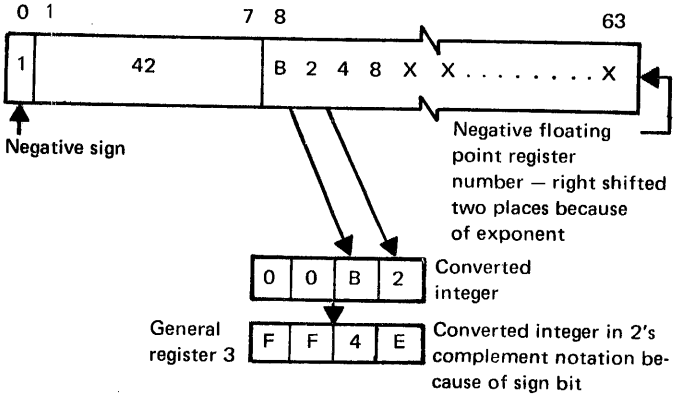
RESULTS: The fixed point integer resides at the operand 1 location. The floating point register is unaffected. The Condition register is affected as follows:

- Bit 0 is set if the floating point number is out of the expressable integer-range arithmetic exception.
- Bit 1 is set if the converted integer > 0.
- Bit 2 is set if the converted integer < 0.
- Bit 3 is set if the converted integer = 0.

EXAMPLE

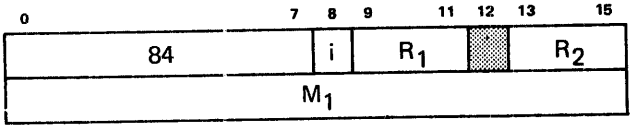


The value in the floating point register will be converted to fixed point and placed in register 3. Register 7 contains an arithmetic exception address.



Load Floating Point Register

LODF @$M_1(R_1),R_2$



FUNCTION: Transfers a floating point number contained in a four-word field in memory to the floating point register and normalizes it. Arithmetic exception occurs if exponent underflow is caused by the normalization.

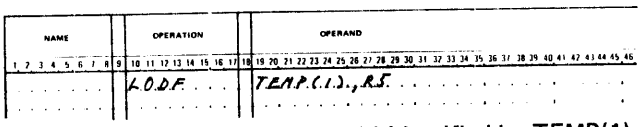
OPERAND 1: A four-word field in memory located at the effective word address. The effective address points to the most significant word of the four-word floating point number. The floating point number need not be normalized. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

OPERAND 2: A one-word field in the general register specified by R_2. This field is the arithmetic exception address, to which control is transferred rather than to RNI if exponent underflow occurs.

RESULTS: The Condition register is affected as follows:

- Bit 1 is set if the normalized number > 0.
- Bit 2 is set if the normalized number < 0.
- Bit 3 is set if the normalized number = 0.
- Bits 0 and 2 are set if exponent underflow occurs during the normalization. In this case the sign and fraction of the floating point number are correct but the exponent is too large by a value of 128.

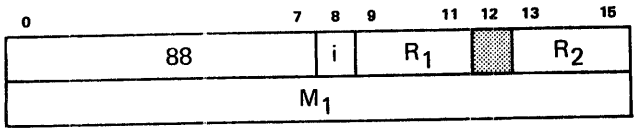
EXAMPLE



The four-word floating point field identified by TEMP(1) is loaded into the floating point register and normalized. Register 5 contains the arithmetic exception address.

Multiply Floating Point

MPYF @$M_1(R_1),R_2$



FUNCTION: Multiplies the contents of the floating point register by a four-word floating point number in memory (the multiplier). The fraction of the resulting product is normalized and is only 14 hexadecimal digits in length; therefore, maximum significance in the product can be presented by starting with two normalized operands, although it is not required.

The floating point multiplication consists of three operations:

1. Adding the two exponents and subtracting 64 (40_{16}) because of the excess 64 exponent notation.
2. Multiplication of the fractions with the resultant product sign following the rules of signed numbers.
3. Product normalization.

An exponent-overflow arithmetic exception occurs if the resultant product exponent is greater than +63 and an exponent-underflow arithmetic exception occurs if the resultant exponent is less than -64. Exponent overflow does not occur when the intermediate exponent exceeds +63 if the exponent is brought into range during normalization.

OPERAND 1: A four-word field in memory beginning at the effective word address. The effective address points to the most significant word of the four-word floating point number. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

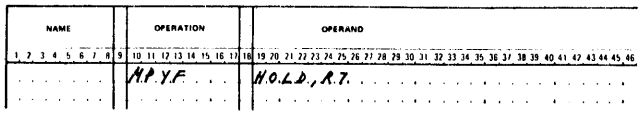
OPERAND 2: A one-word field in the general register specified by R_2. This field is the arithmetic exception address to which control is transferred if arithmetic exception occurs.

RESULTS: The normalized product resides in the floating point register. A final fraction of zero generates a true zero product. The Condition register is affected as follows:

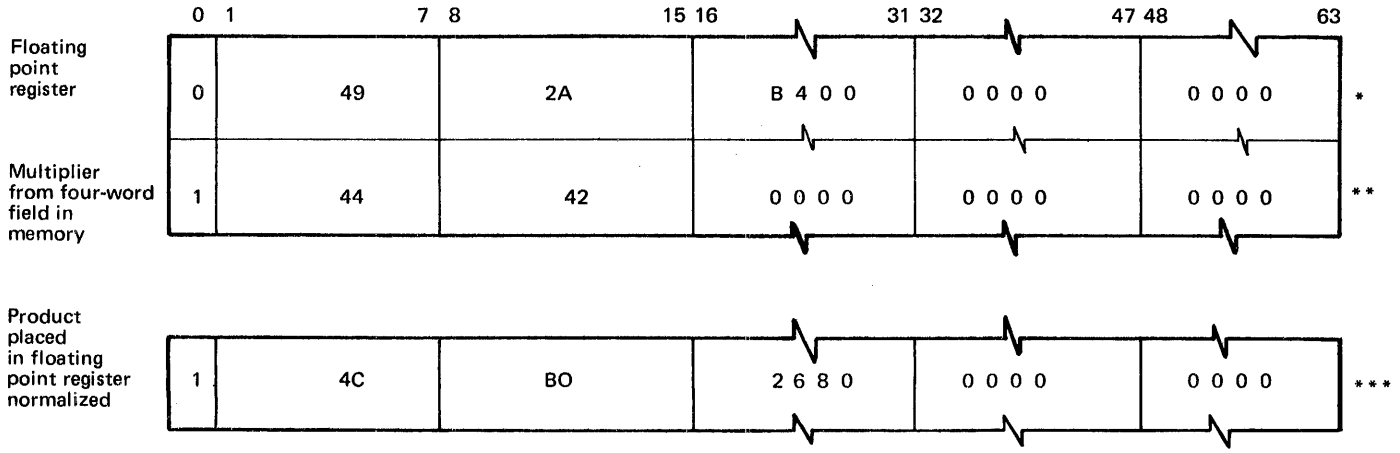
- Bit 1 is set if the normalized product > 0.
- Bit 2 is set if the normalized product < 0.
- Bit 3 is set if the normalized product = 0.
- Bits 0 and 2 are set if exponent underflow occurs during normalization or exponent determination. The sign and the fraction of the product are correct, but the exponent is too large by a value of 128.

- Bits 0 and 3 are set if exponent overflow occurs during exponent determination. The sign and the fraction of the product are correct, but the exponent is too small by a value of 128.

EXAMPLE



The contents of the floating point register are multiplied by the field identified by HOLD; the results will be in the floating point register. Register 7 contains an arithmetic exception address.

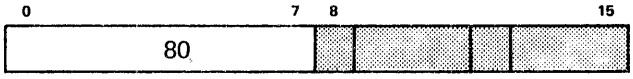


* = +.2AB4 x 16^{+9} *** = –.B0268 x 16^{+C}

Bit 2 of the Condition register is set (product < 0). ** = –.42 x 16^{+4}

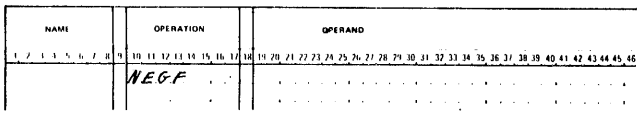
Negate Floating Point Register

NEGF



FUNCTION: Inverts the sign bit (bit 0) of the floating point register. The rest of the floating point register is unchanged. The Condition register is not affected. This instruction has no operands.

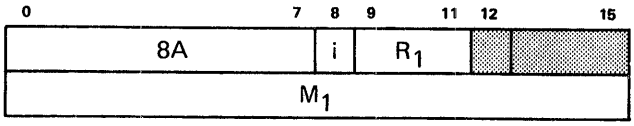
EXAMPLE



The sign bit in the Condition register is inverted; if the sign is minus it will become plus, and vice versa.

Store Floating Point Register

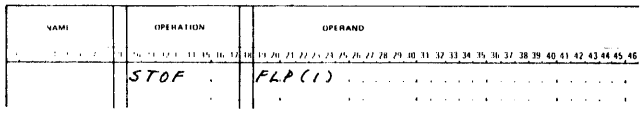
STOF @$M_1(R_1)$



FUNCTION: Transfers the contents of the floating point register to a four-word field in memory.

OPERAND 1: The four-word receiving field in memory located at the effective word address. The effective address points to the most significant word of the memory field. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

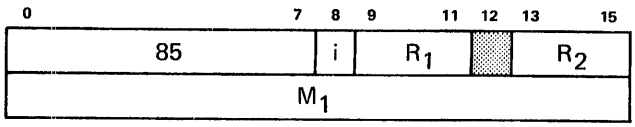
EXAMPLE



This instruction transfers the contents of the floating point register to the field identified by FLP(1).

Subtract Floating Point

SUBF @$M_1(R_1),R_2$



FUNCTION: Performs an algebraic subtraction of a four-word floating point value in memory from the contents of the floating point register. Both operands need not be normalized before the subtraction.

The subtract operation is performed in the same manner as the add except that the sign of the floating point number in memory (subtrahend) is inverted first.

OPERAND 1: A four-word field in memory beginning at the effective word address. The effective address points to the most significant word of the four-word floating point number. Addressing options to the base address M_1 include indexing, indirect addressing, or a combination of both.

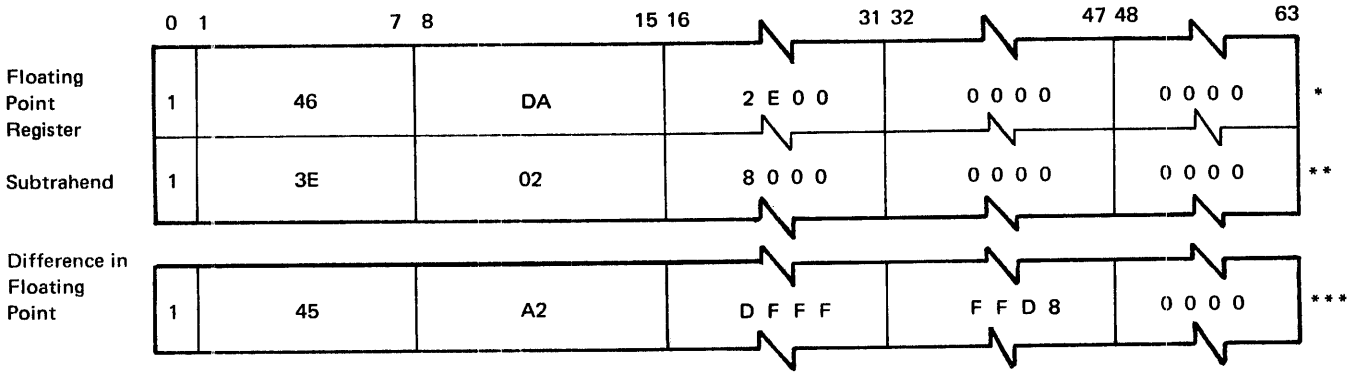
OPERAND 2: A one-word field in the general register specified by R_2. This field is the arithmetic exception address to which control is transferred if arithmetic exception occurs.

RESULTS: The difference resides in the floating point register as a normalized number. The Condition register is affected the same as for the ADDF and the same arithmetic exceptions are possible.

EXAMPLE

NAME	OPERATION	OPERAND
	SUBF	FLDA(2),,R6

The field identified by FLDA(2) is subtracted from the contents of the floating point register; the results will be in the floating point register. Register 6 contains an arithmetic exception address.



* = −.0A2E x 16^{+6} *** = −.A2DFFFFF x 16^{+5}

** = −.028 x 16^{-2}

Bit 1 of the Condition register is set (difference >0).

SYSTEM INSTRUCTIONS

System instructions are specialized instructions used to interpret and alter the defined operating system. These instructions are divided into two major groups, Control and I/O. For the system instructions, a distinction is also made between privileged and restricted instructions. *Privileged instructions* can be executed by a processor only if the bit in the Privileged Mode register associated with that processor is set. *Restricted instructions* can only be executed by one of the dedicated processors 0-4, that is, they are restricted to one of these processors. The Control instructions include privileged and restricted instructions; the I/O instructions are all restricted.

NOTE

In the discussion of System instructions, the term "hardware" implies a combination of the hardware logic components and the microprograms. It is the conjunctive action of both of these elements that determines the manner in which a given processing-unit instruction will be executed.

EXTENDED FILE REGISTER

The system instructions manipulate and use the Extended Register file. Figure 4-3 shows the breakdown of the Extended file. The Function (F) and Pμ registers are under internal control. Processor state 4 can have access to any register in both the Basic and the Extended Register files – with the exception of the I/O registers, which are reserved solely for the associated I/O processor states.

The I/O register exist only for processor states 0 through 3; and then, only those actually needed are present. A description of the I/O registers associated with each I/O processor state is found in the appropriate I/O instruction description in this section.

The Common Block registers may be addressed by any processor state, although some of them only if the

addressing state is in the "privileged" mode. Moreover, some registers are for the read-only mode: Real-Time Clock and Parity Error are examples.

Table 4-1 gives a brief description of the Extended file registers. It is followed by an in-depth discussion of the Common Block register functions and usages.

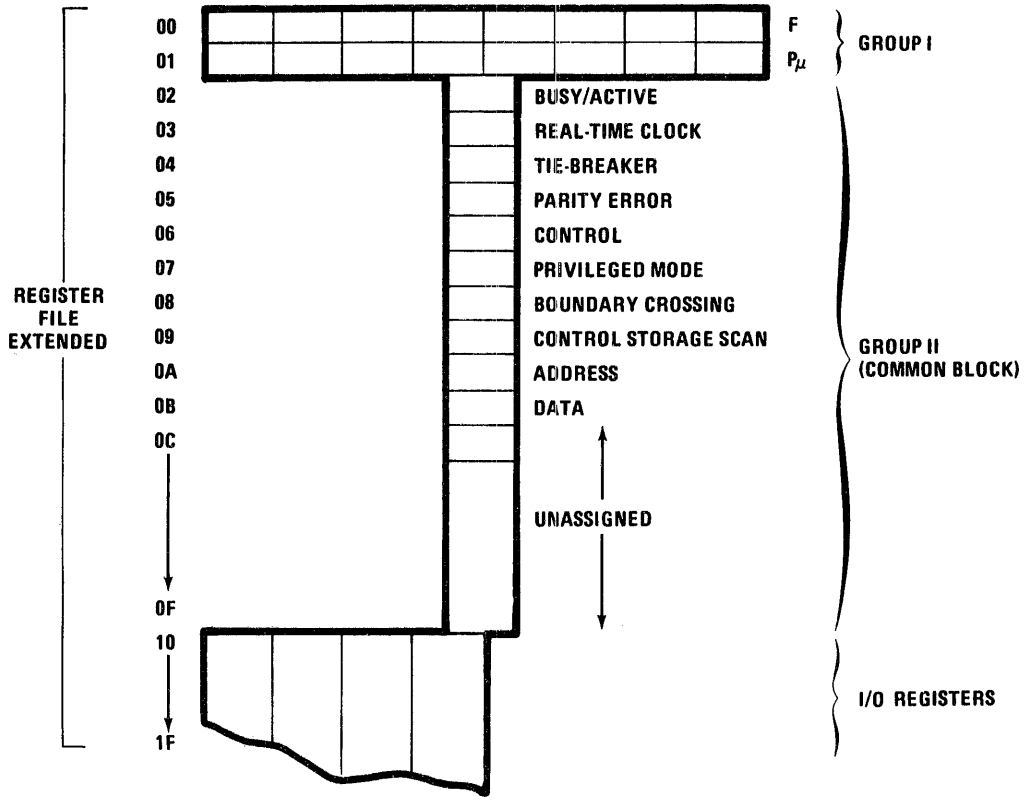


Figure 4-3. Extended Register File

Table 4-1. Extended Register File

Basic File

Register Number	Mnemonic Title	Name and Function
00	F	Function Register – contains the base instruction word (first two bytes) of the instruction being executed by the associated processor state.
01	P	Micro Address Register – contains the control storage address of the first micro-instruction (μI) to be executed during the next major cycle assigned to the associated processor state. It also records certain conditions resulting from the execution of μI's.
These "common block" registers are not specifically associated with a single processor state. As a result, they may be addressed by any state.		
02	B/A •	Busy/Active Register – indicates which processor states need a major cycle (Busy), and which are currently assigned a task to perform (Active).
03	RTC	Real-Time Clock – counts up to 107.4 seconds for interval timing. Its count is triggered every 1.6384 milliseconds by a 60-Hz (nominal) oscillator. Programs can periodically read (but not write into) this register to measure intervals of time. Certain operations, such as I/O transfers, require a definite amount of time to complete. If they are not complete within that time, chances are that a malfunction occurred. The RTC can be used to "time" such operations.

Table 4-1. (Continued)

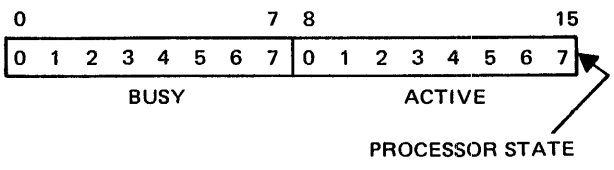
Register Number	Mnemonic Title	Name and Function
04	T	Tie Breaker Register — ensures that one program will not refer to a table that is simultaneously being updated by another program — provided that both programs utilize the Tie-Breaker register. It is set by software indicating when certain data tables, each represented by a bit position in the register, are being addressed.
05	PE	Parity-Error Register — holds the last main storage address in which a parity error occurred. It is updated by the hardware and is usually addressed by the Executive processor state for error logging.
06	C •	Control Register (dual function) — the lower eight bits indicate whether the respective processor state may be allowed to take consecutive major cycles, other conditions permitting. The upper eight bits indicate one of three levels of priority for each of the four I/O processor states (2 bits each). The register is set by software.
07	PM •	Privileged-Mode Register — indicates whether each processor state is empowered to operate in the "privileged mode". It contains a bit position representing each processor state. The privileged mode allows a state to execute certain instructions which, if executed indiscriminately, might cause hang-ups, invalid data, or other errors.
08	BC •	Boundary Crossing Register — records those instances in which, under certain conditions, the Executive processor state may read or write the registers of another state. The BC register holds the address of the register, along with the register set and the number of the processor state to be addressed.
09	CSS	Control Storage Scan — checks the longitudinal parity of 256-word "pages" in the CS. An exclusive OR performed on the contents of a CS page should yield all 1's in the register. Any attempt by software to write into this register will clear it.
0A	CONSOLE ADDRESS	Address-Related Console Register — provides entry and display of register, MS or CS addresses, in conjunction with the row of 16 illuminated Address pushbuttons on the System Control Panel.
0B	CONSOLE DATA	Data-Related Console Register — provides entry and display of data to or from the entity addressed by means of the Address pushbuttons, in conjunction with the row of 16 Data pushbuttons on the System Control Panel. In addition, a rotary switch on the panel allows the display (but not entry) of data stored in certain nonaddressable registers.
0C–0F		Unassigned

NOTE

The following four registers in the Common Block group of the Extended Regilster file are "limited access" registers. This means that although they may be read by any processor state, they can be written into only by a privileged processor state, or by the Executive processor state. Specific *write* instructions for each register are given in the separate discussions.

Busy-Active Register (B/A)

The bit assignments for the Busy/Active register are shown below.



When the Busy bit is 1, it indicates to the hardware (RAN) that a given processor state needs a major cycle. The "request" that sets Busy may come from the program or from one of the switches on the System Control Panel. (The use of these switches is discussed in Section 5.) Moreover, if one of the four I/O processor states is involved, its Busy bit may also be set by a request from an I/O device.

The following instructions are available to the programmer to set or clear bit positions in the B/A register via the ALU inputs. *Control Instructions* has complete descriptions of the instructions listed, and of the meanings of the terms, "privileged" and "restricted".

SR — *Service Request:* Turns on both the Busy and Active bits for the Exec; turns off both the Busy and Active bits for the requesting processor state.

SBA — *Set Busy/Active Register:* The processor state executing this instruction must be privileged. Turns on the B or A bit, or both, for the designated processor state.

RBA — *Reset* (clear) *Busy/Active Register:* The processor state executing this instruction must be privileged. Turns off the B or A bit, or both, for the designated processor state.

WRX — *Write Extended Register:* The processor state executing this instruction must be privileged. This instruction affects all 16 bits of the register. It must be used with an appropriate mask to set (or clear) the desired B/A bit positions without affecting the bits that are to remain unchanged.

WAR — *Write Any Register:* This instruction is restricted to the Executive (processor state 4). In other respects, its influence on the B/A register is identical to that of WRX.

The Active bit is used for several purposes; these can be segregated by processor states. The uses of Active for these four groups is discussed in order of increasing complexity.

States 5, 6, 7

When Busy and Active are set for these states, continuous operation ensues. The program associated with each state should be executed as rapidly as possible, within the constrainst of Resource Allocation Network (RAN) sequencing, so Busy will remain set for these states.

Active for these states is not set during the Trace mode, used for trouble-shooting or debugging a program. In this mode it is desired to have the processor state operate one major cycle or one instruction at a time, rather than operating continuously.

States 2, 3

The Active bit for these states serves as an "envelope of protection" against the setting of Busy when such setting (and clearing) must be under the exclusive control of the processor state itself. Such a situation exists when doing a data transfer, or when searching for a given record on a disc or magnetic tape. During a *Read* data transfer, for example, Busy is continually set and cleared. It is set when data from the connected I/O device is available; it is cleared when that data has been transferred, examined, and stored.

For non-buffered devices, this on-off sequencing of Busy is determined solely by the speed of the peripheral unit. For the relatively long periods when Busy is off during such a transfer, some outside influence such as the Exec or an *Attention* from another unit could – except for the Active envelope – cause the I/O processor state to react as if its Busy FF had been set because of the availability of data from the connected equipment when, in fact, it hadn't. The resulting "data" transferred would of course be invalid.

When the STOP-STEP switch on the System Control Panel is activated, no I/O transfers can be executed. This switch is primarily a diagnostic tool. Its ramifications with respect to I/O requests are considered in a separate manual.

States 0, 4

To the Communications (state 0) or Executive (state 4) processor, a set Active bit indicates that there is a job waiting to be executed that is different from the task in which it is currently engaged. When these states receive a major cycle, the first thing they do is clear the Active bit. At the conclusion of the current task, then, if Active is set it had to be because a new request was initiated while the processor was still performing the current assignment.

CONTROL REGISTER (C)

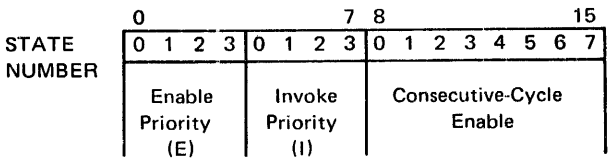
The Control register performs two functions:

1. It permits an I/O processor (states 0-3) to override the normal RAN sequencing – that is, to obtain priority for the next major cycle.

2. It permits any of the eight processors to obtain consecutive major cycles as long as no other processor state is also waiting for that "second" cycle (bits 8-15).

Bit positions in the Control register may be set or cleared individually by the privileged *SCN* (set control) or *RCN* (reset control) instructions. The *WAR* or *WRX* instructions may also be used, but as with the B/A register, these instructions require a mask to preserve the integrity of the unchanged bits.

The bit assignments for the C-register are shown below.



RAN Concept

The resource allocation network (RAN) ensures that each processor state receives its fair share of time with the shared resources. The scanning period of the network is based upon the cycle time of main storage (1.8 microseconds); under normal conditions, then, each state will use the shared resources once every 14.4 microseconds (1.8 microseconds x 8 states).

But normal conditions do not always prevail. A given high-performance disc, for example, may require exchanging a data word with its processor state every 2.5 microseconds. Under these requirements, an obvious need is for some scheme whereby the disc (or any other peripheral device, for that matter) can "override" the normal RAN sequence.

Any allocation scheme, to be the least bothersome to the user, should consider of highest priority those devices for which the recovery of lost data would demand some operator intervention — retyping the input data on a keyboard terminal, for example, or reloading a card deck. Of lower priority would be those devices from which the computer could recover the data automatically — rereading (or rewriting) a disc track or magnetic tape, etc. Finally, of lowest priority would be the non-I/O operations — arithmetic, comparing, shifting, etc. How these various considerations are effected is the subject of the following discussion.

The RAN essentially examines the Busy bits in the Busy/Active register to see which processor states require access to the shared resources, and then determines which state shall be given the next time slice.

RAN Normal Sequence Concept

Scanning is performed in numerical sequence, with highest preference given to processor state 0, lowest to state 7. In this manner, the processor states with the highest degree of "operator intervention", as described above, are serviced first. To prevent the lower-numbered processor states from monopolizing the shared resources, however, the RAN incorporates logic that prevents a lower-numbered state from receiving a second time slice if higher-numbered states are still waiting. Thus, if 0, 4, and 5 were waiting, state 0 would get the first slice, but could not receive another until both 4 and 5 had had their turn. In this manner, the shared resources are utilized most equitably.

RAN Override (Priority) Concept

Suppose, now, that while processor state 0 were waiting for 4 and 5 to be serviced so that it could receive another time slice, it received indications that it could not wait for normal sequencing. Under these conditions, an override situation is needed, to negate the normal RAN sequence, permitting state 0 to receive the third time slice at the expense of any higher-numbered states (in this case, 5, since 4 would have taken the second time slice).

Override, or priority, can be enabled by the control program at the start of each I/O data transfer. (There are no such priority provisions for the four non-I/O processor states.) The Control Word that precedes each I/O transfer can specify one of three priority levels:

- Enable Priority (processor's E-bit = 1) — This level allows an I/O processor state to secure an out-of-sequence time slice, provided that no lower-numbered processor state is also in a priority condition.*

---

* Priority Condition means that a state will receive an out-of-sequence time slice as a result of one of these conditions:

- E-bit position set AND being in danger of losing data,
- I bit position set.

This differs from Priority Mode, which simply refers to having either E or I set, without implying consequent override of the RAN. That is, *Condition* is a dynamic situation, while *Mode* is a statis situation. Moreover, a Priority Condition is not examined to see which of the bit positions is set. Therefore, if state 0 had *E* set and state 1 had *I* set, state 0 would receive the next available time slice.

- Invoke Priority (processor's I-bit = 1) — This level allows an I/O processor state to secure alternate time slices, provided that no lower-numbered processor state is also in a priority condition.
- Revoke Priority (processor's E- and I-bits both = 0) — This level negates the Enable or Invoke levels.

Consecutive Cycles

The action of bits 8-15 of the C-register in permitting consecutive cycles was mentioned in Section 1. The mode is useful when only one of the general-purpose processors is actively engaged in a user's problem program, and during periods when that program does not call for I/O activity. Then, in the relatively long gaps between the times when the real-time clock input from the 60-Hz oscillator turns on the Exec processor for system monitoring, the active G-P state may obtain consecutive major cycles.

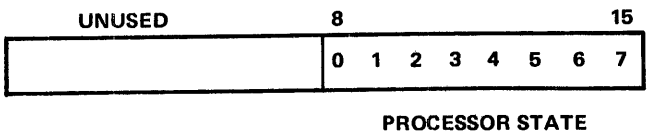
This illustration sets forth just one -- albeit perhaps the most frequently used -- application of the consecutive-cycle mode.

PRIVILEGED MODE REGISTER (PM)

The Privileged Mode register contains one bit for each of the eight processor states. When set, that bit position permits the associated processor to execute any of the privileged *Control* instructions described in Section 4.

One of these privileged instructions is *Set/Reset Privileged Mode Register (SPM/RPM);* this means that a processor state cannot use this instruction to set itself to the privileged mode. The Exec, however, may set itself privileged by using the restricted *Write Any Register (WAR)* instruction (with a suitable mask). It can then set any other processor to the privileged mode, as the need arises, by using *SPM.*

The bit assignments for the PM-register are shown below.

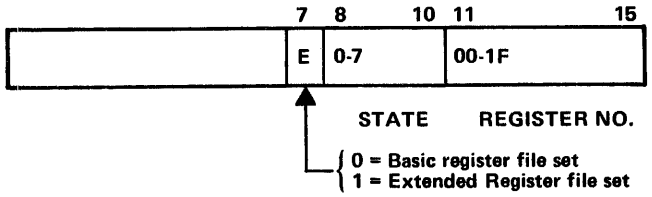


BOUNDARY CROSSING REGISTER (BC)

The BC-Register is used whenever software directs the Exec to write or read the register of another state.

Exception: Group III registers in the extended file cannot be reached via this register.

The format of the register is identical to that of the second word in the restricted *Write/Read Any Register WAR/RAR* instruction:



Execution of a properly formated *WAR/RAR* instruction automatically writes the specified state set, and register number into BC.

Because BC is an extended register, it may also be written into by a privileged processor state executing the *Write Extended Register WRX* instruction (with 08 as the destination register). However, the exercise would be futile if the intent was to read or write the register specified by the newly entered contents of BC, because only the Exec may cross processor-state boundaries.

REGISTER OPTION

General

The Register Option (RO) encompasses registers added to the processing unit hardware to implement the following features:

- Basic Storage Protection (Bounds)
- Job Accounting

NOTE

The reading and writing of RO registers is presently limited to the *RRO/WRO* machine-language instruction. This control instruction is privileged; in practice this means that its execution is limited to system programs. Later releases may include instructions that read or write certain of these registers in a non-privileged mode. Until such time, however, an attempt to execute *RRO/WRO* by a non-system program will result in an *Invalid Instruction* indication in the Condition register of the processor state attempting the execution.

The address structure of the register groups devoted to each of the RO features is shown in Figure 4-4. The method of coding hexadecimal address for RO registers uses the format prescribed by the second word of the *RRO/WRO* control instruction.

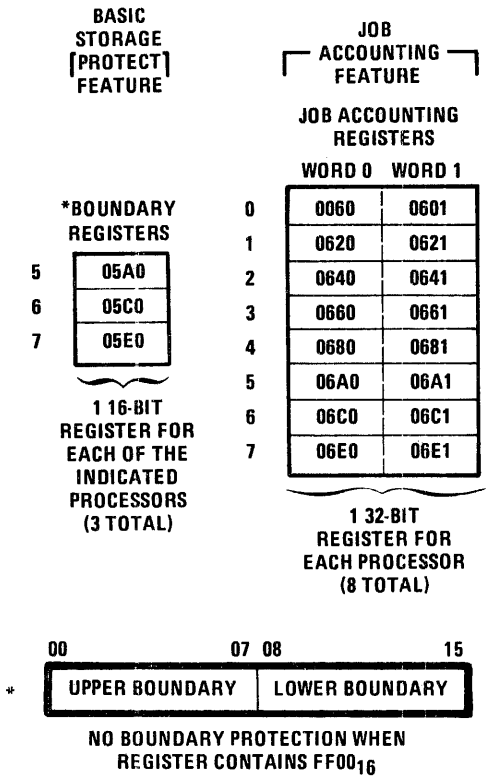


Figure 4-4. Register Option Address Structure

It should be pointed out that when a register set (or group) contains but one register, or when there is only one register within a group that relates to a given processor state (Bounds), that register number is always 0. The "register 0 only" selection applies to groups 3, 4, 5, and to groups 8 through F. An attempt to gain access to a non-existing RO feature will result in the following action:

> on read — all 0's will be entered into the "destination" (Operand 1) file register or memory location.
>
> on write — no operation

No flag will be set to indicate the addressing error.

Specific bit configurations for the registers are described under the feature headings that follow.

Basic Storage Protect

This feature checks storage bounds on write (but not on read) operations in main storage for general-purpose processors 5, 6, and 7. As shown in Figure 4-4, the 16-bit Bounds register for each of the three states provides an 8-bit "page" address (maximum page number, 255) for both upper and lower bounds. (Each page comprises 256 contiguous byte addresses.)

Protection is enforced except for the case where the Bounds Register contents are FF00, hexadecimal.

Job Accounting

The 32-bit Job Accounting Registers keep track of the number of major cycles (time slices) that the Resource Allocation Network has granted to each of the eight processor states. These registers are **read only**. Any attempt to write into one of these registers will clear it.

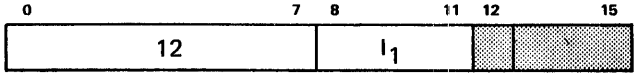
Two *Read Register Option (RRO)* instructions are required to read both addresses (word 0, word 1) of a given processor state's Job Accounting register.

CONTROL INSTRUCTIONS*

Mnemonic	Name
CTB	Clear Tie-Breaker Register (p)
TST	Test and Set Tie-Breaker Register (p)
BCM	Branch to Control Memory (p)
RAR	Read Any Register (r)
WAR	Write Any Register (r)
RRO	Read Register Option (p)
WRO	Write Register Option (p)
SAR	Save All Registers (r)
RSAR	Restore All Registers (r)
SBA	Set Busy-Active Register (p)
RBA	Reset Busy-Active Register (p)
SCN	Set Control Register (p)
RCN	Reset Control Register (p)
SPM	Set Privileged Mode Register (p)
RPM	Reset Privileged Mode Register (p)
WRX	Write Extended Register (p)**

Clear Tie-Breaker Register

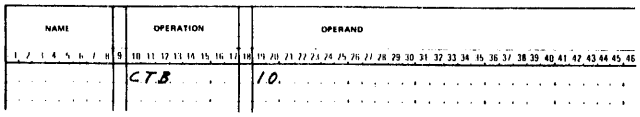
CTB I_1



FUNCTION: A privileged instruction that resets a single bit in the Tie-Breaker register.

OPERAND 1: A 4-bit unsigned value in bits 8-11 of the instruction. This value specifies the position of the bit to be reset in the Tie-Breaker register.

EXAMPLE



Turns off bit 10 of the Tie-Breaker register.

NOTE

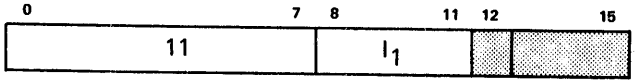
The CTB instruction is classified as a privileged instruction, but becomes a general-purpose instruction (privileged mode bit need not be set) when the bit named in the operand is bit 15 of the Tie-Breaker register.

*p = privileged, r = restricted

**Read Extended Register is a General-Purpose Control instruction.

Test and Set Tie-Breaker Register

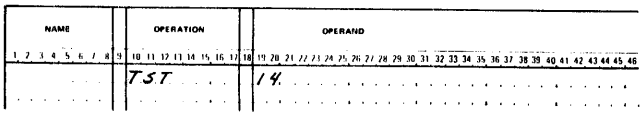
TST I_1



FUNCTION: A privileged instruction that tests and sets a single bit in the Tie-Breaker register in order to resolve synchronization problems resulting from parallel processor execution. If the designated bit is clear, it will be set and the Program Address register will be advanced six bytes. If the designated bit is set, it will remain unchanged and the program will fall through to the next instruction (the Program Address register will be advanced by two bytes). A TST instruction would generally be followed by a four-byte branch instruction.

OPERAND 1: A 4-bit value in bit positions 8-11 of the instruction. This value specifies the bit position in the Tie-Breaker register which is to be tested and set.

EXAMPLE



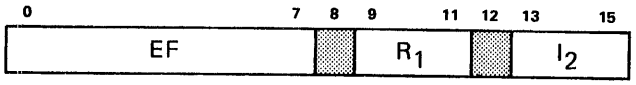
Tests bit number 14 in the Tie-Breaker register. If this bit is off, it is turned on and the address in the Program Address register is increased by six bytes. If the tested bit is on, the bit is not changed and the next instruction is read.

NOTE

The TST instruction is classified as a privileged instruction, but becomes a general-purpose instruction (privileged mode bit need not be set) when the bit named in the operand is bit 15 of the Tie-Breaker register.

Branch to Control Memory

BCM R_1,I_2

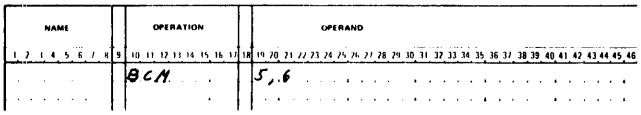


FUNCTION: A privileged instruction that transfers control to the control memory address contained in a general register.

OPERAND 1: The control memory address located in the general register specified by R_1.

OPERAND 2: A 3-bit value in bit positions 13-15 of the instruction. This operand is optional; when used it transfers information to the control memory routine to which control is transferred.

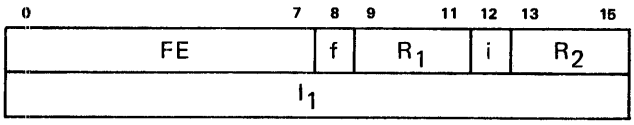
EXAMPLE



Jumps to the control memory address specified in register 5. Information contained in register 6 (parameters, etc.) is transferred to the control memory routine.

Read Any Register

RAR $I_1(R_1),@R_2$



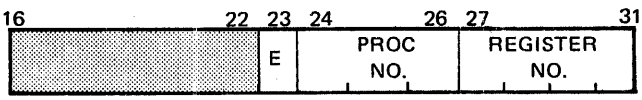
FUNCTION: A restricted instruction that allows the Executive processor (4) to read the contents of any file register for any of the eight processor states, with the exception of the Group III extended (I/O) registers for states 0 through 3.

Extended Function Code: Bit 8 serves as an extension to the basic function code and is a 0 for this instruction; this bit distinguishes between RAR and WAR.

OPERAND 1: The address of the file register to be read is specified by the I_1 value (bits 16-31 of the instruction). The I_1 value may optionally be modified by the contents of the R_1 register, thus the effective register address = I_1 + (R_1).

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

The format of the I_2 portion of the instruction is as follows:



Bit 23 indicates the register-file set:

0 = Basic Register

1 = Extended Register

Bits 24-26 indicate a processor state, 0-7.

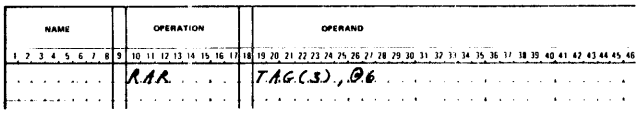
Bits 27-31 indicate a register number in the ranges shown below:

Basic Registers — 00-1F

Extended Registers — 00-0F

Bits 16-22 are ignored by the hardware, but should be zeros.

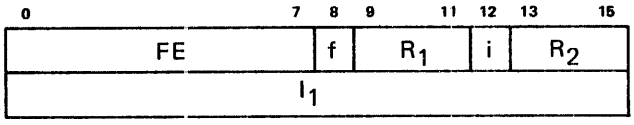
EXAMPLE



Reads a register in the register file addressed by TAG(3), and stores the contents of the register in the memory location specified in register 6.

Write Any Register

WAR $I_1(R_1),@R_2$



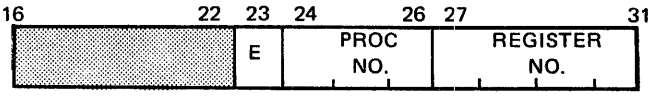
FUNCTION: A restricted instruction that allows the Executive processor (4) to write into any file register for any of the eight processor states, with the exception of the Group III extended (I/O) registers for states 0 through 3.

Extended Function Code: Bit 8 serves as an extension to the basic function code and is a 1 for this instruction; this bit distinguishes between RAR and WAR.

OPERAND 1: The address of the file register into which the data is to be written is specified by the I_2 value (bits 16-31 of the instruction). The I_1 value may optionally be modified by the contents of the R_1 register, thus the effective register address = $I_1 + (R_1)$.

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

The format of the I_2 portion of the instruction is as follows:



Bit 23 indicates the register-file set:

0 = Basic Register

1 = Extended Register

Bits 24-26 indicate a processor state, 0-7.

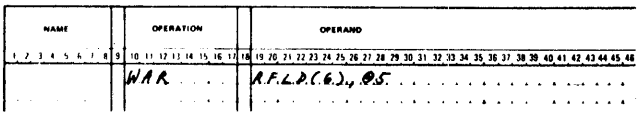
Bits 27-31 indicate a register number in the ranges shown below:

Basic Registers — 00-1F

Extended Registers — 00-0F

Bits 16-22 are ignored by the hardware, but should be zeros.

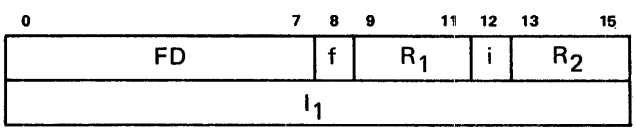
EXAMPLE



The contents of a memory location specified by the address in register 5 are written in the register with the register address specified by RFLD(6).

Read Register-Option Register

RRO $I_1(R_1),@R_2$



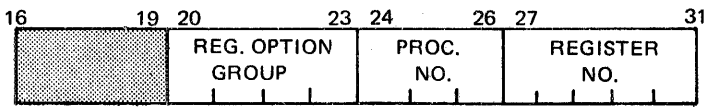
FUNCTION: A privileged instruction that reads the contents of a two-byte field from a register-option (RO) register and transfers the contents to a general register or a memory location.

Extended Function Code: Bit 8 serves as an extension to the basic function code and is 0 for this instruction; this bit distinguishes between RRO and WRO.

OPERAND 1: The address of the register-option register to be read is specified by I_1 (bits 16-31 of the instruction). The I_1 value may optionally be modified by the contents of the R_1 register, thus the effective register address = $I_1 + (R_1)$.

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

The format of the I_1 portion of the instruction is as follows:



Bits 20-23 specify the RO group, as follows:

Hex Value	Meaning	Structure
0	Segment Tag	32 4-bit registers for each proc.
1	Protection Matrix	2 16-bit registers for each proc.
2	Segment Relocation Table	16 24-bit registers
3	Address Mode	1 16-bit register
4	Parity Error Tag	1 16-bit register
5	Bounds	3 16-bit regs. (procs. 5, 6, 7)
6	Job Accounting	2 16-bit registers for each proc.
7	Not used	
8/9	Main Storage Data	1 16-bit register in each group
A/B	Error Log	
C/D	Generated Check Bits	
E/F	Read Check Bits	

Bits 24-26 specify a processor state, 0-7.

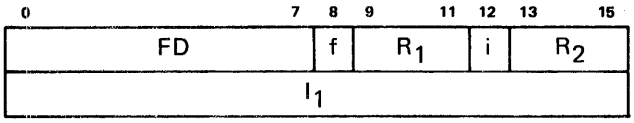
Bits 27-31 specify a register number as defined below:

Group	Reg. No. or Range	Remarks
0	0–1F	32 registers/processor
1, 2, 6	0–1	Reg. 0=word, Reg. 1=word 1
3, 4, 5, 8 – F	0	One register per group

See the *Register Option* discussion in Section 1, General Description, for additional information on the register-option registers.

Write Register-Option Register

WRO $I_1(R_1),@R_2$



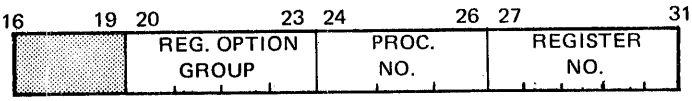
FUNCTION: A privileged instruction that writes the contents of a one-word field, located in a general register or in memory, into a specified register-option (RO) register.

Extended Function Code: Bit 8 serves as an extension to the basic function code and is a 1 for this instruction; this bit distinguishes between RRO and WRO.

OPERAND 1: The address of the register-option register into which the data is to be written is specified by the I_2 value (bits 16-31 of the instruction). The I_2 value may be optionally modified by the contents of the R_1 register, thus the effective register address = $I_2 + (R_2)$.

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

The format of the I_2 portion of the instruction is as follows:



Bits 20-23 specify the RO group:

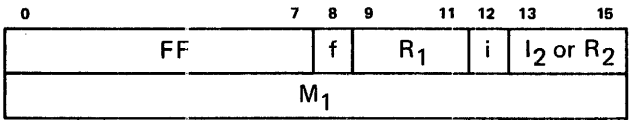
Bits 24-26 specify a processor state, 0-7.

Bits 27-31 specify a register number as defined:

See the *Register Option* discussion in Section 1, General Description, for additional information on register-option registers.

Save All Registers

SAR $M_1(R_1),I_2$ or $M_1(R_1),@R_2$



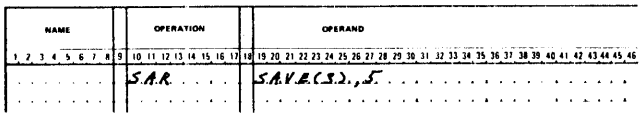
FUNCTION: A restricted instruction that stores in memory the contents of all general registers, the Condition register, and the Program Address register for the processor specified. This instruction may be executed only by the Executive processor state.

Extended Function Code: Bit 8 serves as an extension to the basic function code and is 0 for this instruction; this bit distinguishes between SAR and RSAR.

OPERAND 1: The starting address of the save area in memory where the register contents are to be stored. Addressing options to the base address M_1 include only indexing.

OPERAND 2: The processor number as specified by I_2, or contained in the general register specified by R_2 if bit 12 (i) is on. Bit 12 does not indicate indirect addressing for this instruction. If bit 12 is on, the value in bits 13-15 is considered a register number containing the processor number. If bit 12 is off, the value in bits 13-15 is considered to be an immediate value which is the processor number.

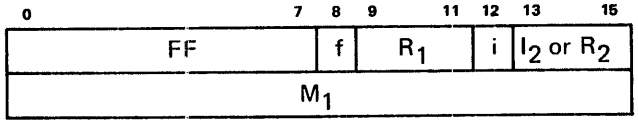
EXAMPLE



Loads a save area beginning at SAVE(3) into the general, Condition, and Program Address registers of Processor 5.

Restore All Registers

RSAR $M_1(R_1),I_2$ or $M_1(R_1),@R_2$



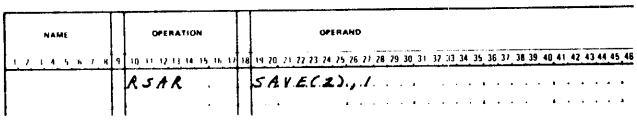
FUNCTION: A restricted instruction that loads the contents of a save area in memory into the general registers, Condition register, and the Program Address register of the processor specified. After the registers have been loaded, the Microprogram Address register of the specified processor will be cleared to facilitate resumption of execution with the next sequential instruction. This instruction may be executed only by the Executive processor state.

Extended Function Code: Bit 8 serves as an extension to the basic function code and is 1 for this instruction; this bit distinguishes between SAR and RSAR.

OPERAND 1: The starting address of the save area in memory from which the registers are loaded. Addressing options to the base address M_1 include only indexing.

OPERAND 2: The processor number as specified by I_2 or contained in the general register specified by R_2 if bit 12 (i) is on. Bit 12 does not indicate indirect addressing for this instruction. If bit 12 is on, the value in bits 13-15 is considered a register number containing the processor number. If bit 12 is off, the value in bits 13-15 is considered a register number containing the processor number.

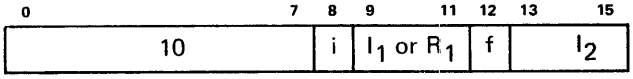
EXAMPLE



Loads Program Address register, Condition register, and general registers for Processor 1 with information taken from a save area identified by SAVE(2). The 16-bit field at SAVE(2) is loaded into register 0, the second into register 1, and so on until words 9 and 10 are loaded into the Condition and Program Address registers, respectively.

Set Busy/Active Register

SBA I_1,I_2 or $@R_1I_2$



FUNCTION: A privileged instruction that sets a busy and/or active bit in the Busy/Active register for the processor designated by I_1 or R_1.

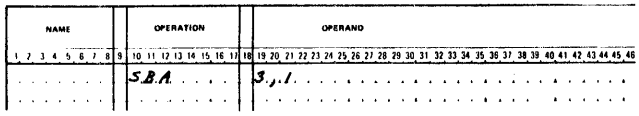
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction; this bit distinguishes between SBA and RBA.

OPERAND 1: The processor number, 0-7. Indirect addressing indicates that the processor number is in the register specified by R_1. Direct addressing indicates that the processor number is the I_1 value in the instruction.

OPERAND 2: A 2-bit designator in bits 14 and 15 used to specify the bit or bits to be set.

- Bit 15=1 means set Busy bit.
- Bit 14=1 means set Active bit.
- Bits 14 and 15=1 means set both bits.

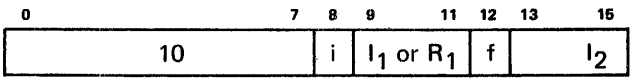
EXAMPLE



Turns on the Busy bit for Processor 3. (If a 2 had been specified as operand 2, the Active bit for Processor 3 would be turned on; if a 3 had been specified, both the Busy and Active bits would be turned on.)

Reset Busy/Active Register

RBA I_1,I_2 or $@R_1I_2$



FUNCTION: A privileged instruction that resets the busy and/or active bit in the Busy/Active register for the processor designated by I_1 or R_1.

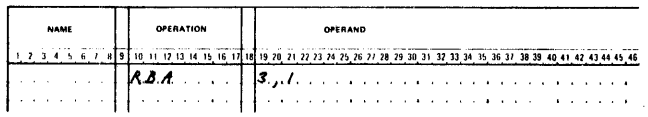
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for this instruction; this bit distinguishes between SBA and RBA.

OPERAND 1: The processor number, 0-7. Indirect addressing indicates that the processor number is in the register specified by R_1. Direct addressing indicates that the processor number is the I_1 value in the instruction.

OPERAND 2: A 2-bit designator in bits 14 and 15 used to specify the bit or bits to be reset.

- Bit 15=1 means reset Busy bit.
- Bit 14=1 means reset Active bit.
- Bits 14=1 and 15=1 means reset both bits.

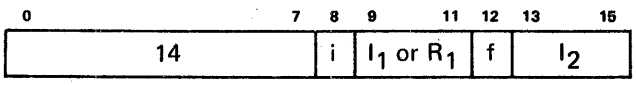
EXAMPLE



The Busy bit for Processor 3 is turned off. (If a 2 had been specified as operand 2, the Active bit for Processor 3 would be turned off; if a 3 had been specified, both the Active and Busy bits for Processor 3 would be cleared.)

Set Control Register

SCN I_1,I_2 or $@R_1,I_2$



FUNCTION: A privileged instruction that sets specified bits in the Control register for the processor designated by I_1 or R_1.

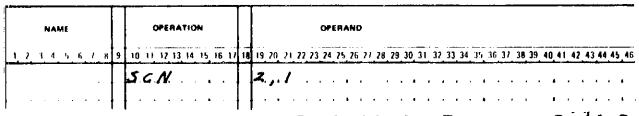
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction; this bit distinguishes between SCN and RCN.

OPERAND 1: The processor number, 0-7. Indirect addressing indicates that the processor number is in the register specified by R_1. Direct addressing indicates that the processor number is the I_1 value in the instruction.

OPERAND 2: A 3-bit designator in bits 13-15 used to specify the bits to set.

- Bit 13=1 means set Enable Priority bit.
- Bit 14=1 means set Invoke Priority bit.
- Bit 15=1 means set Consecutive Cycle bit.

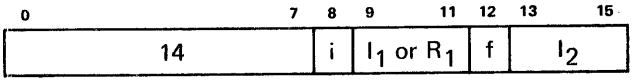
EXAMPLE



Turns on the Consecutive Cycle bit for Processor 2. (A 2 in operand 2 would turn on the Invoke Priority bit; a 4 would turn on the Enable Priority bit.)

Reset Control Register

RCN I_1,I_2 or $@R_1,I_2$



FUNCTION: A privileged instruction that resets specified bits in the Control register for the processor designated by I_1 or R_1.

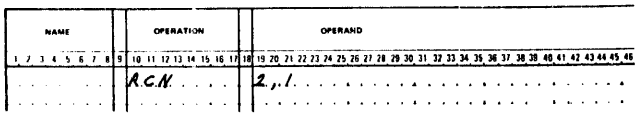
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for this instruction; this bit distinguishes between SCN and RCN.

OPERAND 1: The processor number, 0-7. Indirect addressing indicates that the processor number is in the register specified by R_1. Direct addressing indicates that the processor number is the I_1 value in the instruction.

OPERAND 2: A 3-bit designator in bits 13-15 used to specify the bits to reset.

- Bit 13=1 means reset Enable Priority bit.
- Bit 14=1 means reset Invoke Priority bit.
- Bit 15=1 means reset Consecutive Cycle bit.

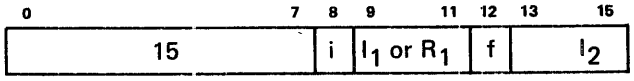
EXAMPLE



The Consecutive Cycle bit for Processor 2 is turned off. (A 2 in operand 2 turns off the Invoke Priority bit; a 4 turns off the Enable Priority bit.)

Set Privileged Mode Register

SPM I_1,I_2 or $@R_1,I_2$



FUNCTION: A privileged instruction that sets the bit in the Privileged Mode register associated with the processor specified by I_1 or R_1.

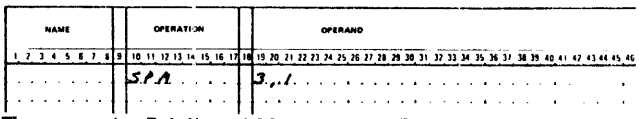
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 0 for this instruction; this bit distinguishes between SPM and RPM.

OPERAND 1: The processor number, 0-7. Indirect addressing indicates that the processor number is in the register specified by R_1. Direct addressing indicates that the processor number is the I_1 value in the instruction.

OPERAND 2: A 2-bit designator in bits 14 and 15 used to specify the bit to set.

- Bit 14 is reserved for future use.
- Bit 15=1 means set Privileged Mode bit.

EXAMPLE



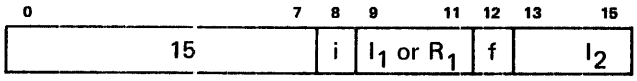
Turns on the Privileged Mode bit for Processor 3.

NOTE

A processor can only execute the SPM instruction if the privileged mode bit is set for that processor. Initial setting of the privileged mode bit can be accomplished with a WAR instruction.

Reset Privileged Mode Register

RPM I_1,I_2 or $@R_1,I_2$



FUNCTION: A privileged instruction that resets the bit in the Privileged Mode register associated with the processor specified by I_1 or R_1.

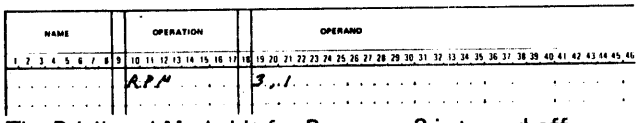
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for this instruction; this bit distinguishes between SPM and RPM.

OPERAND 1: The processor number, 0-7. Indirect addressing indicates that the processor number is in the register specified by R_1. Direct addressing indicates that the processor number is the I_1 value in the instruction.

OPERAND 2: A 2-bit designator in bits 14 and 15 used to specify the bit to reset.

- Bit 14 is reserved for future use
- Bit 15=1 means reset Privileged Mode bit

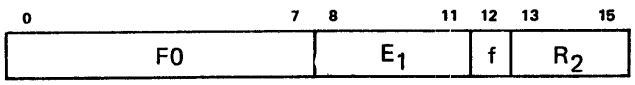
EXAMPLE



The Privileged Mode bit for Processor 3 is turned off.

Write Extended Register

WRX E_1,R_2



FUNCTION: A privileged instruction that writes the contents of the general register into a Group II extended register.

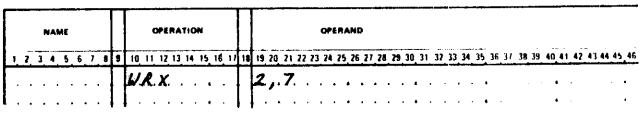
Extended Function Code: Bit 12 serves as an extension to the basic function code and is 1 for this instruction; this bit distinguishes between RDX and WRX.

OPERAND 1: The Group II extended register to be loaded is specified by R_1.

OPERAND 2: The general register which contains the data to be transferred is specified by R_2.

CONSIDERATIONS: Any attempt to access a Group I extended register will result in a trap to the Invalid Instruction routine.

EXAMPLE



Writes the contents of general register 7 into extended register 2.

Each of the three I/O processor states has an associated I/O driver to implement all I/O activity delegated by the Executive program. Conceptually, the I/O driver is the I/O processor. The driver is a series of machine instructions designated by the operating system for the associated processor state.

Implementation

All I/O control and data transfer is via the extended registers of the three I/O states. Machine language instructions are tailored to implement the requirements of three major classes of I/O operation:

- Communications Processor State 0
- Selector Channel Processor State 2
- Disc Processor State 3

Communications

Two I/O instructions provide for communications I/O. WRC is used for setting up the channel for a data transfer, and RDC implements the input or output transfer.

Selector Channels

Three I/O instructions provide for basic data channel operations: INP, OUT, and SIO. INP and OUT prepare the channel for a data transfer, and perform the transfer under software control. SIO can set up the channel, perform a hardware-controlled transfer and ending sequence, and process asynchronous status.

Disc

Three I/O instructions provide for disc operations: INP, OUT, and DIO. INP and OUT perform certain preparatory functions such as *Seek*, while DIO implements time-dependent operations - searching count fields and the actual input or output transfer.

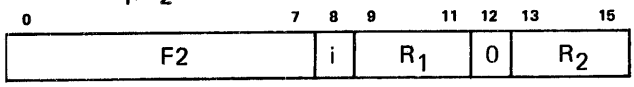
I/O Instruction List*

Mnemonic	Name
DIO	Disc I/O
INP	Input from I/O Register
OUT	Output from I/O Register
RDC	Communications I/O
WRC	Communications Output
SIO	System I/O

*All I/O instructions are restricted to one of the dedicated processors, 0-3.

Disc Input/Output

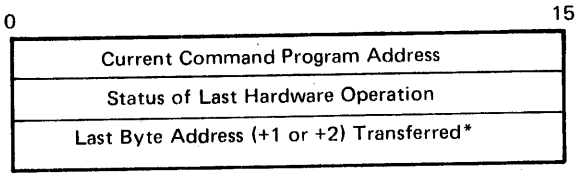
DIO @R_1,R_2



FUNCTION: Operates on a variable length command packet or several such packets located at the address specified by R_2 to effect a variety of disc channel operations including buffered data transfer (input or output). This is a restricted instruction limited to Processor 3.

OPERAND 1: A one-word field containing the first word address of the command packet. The command packet address is in the general register specified by R_1 for direct addressing, or in the memory location specified by the contents of R_1 for indirect addressing.

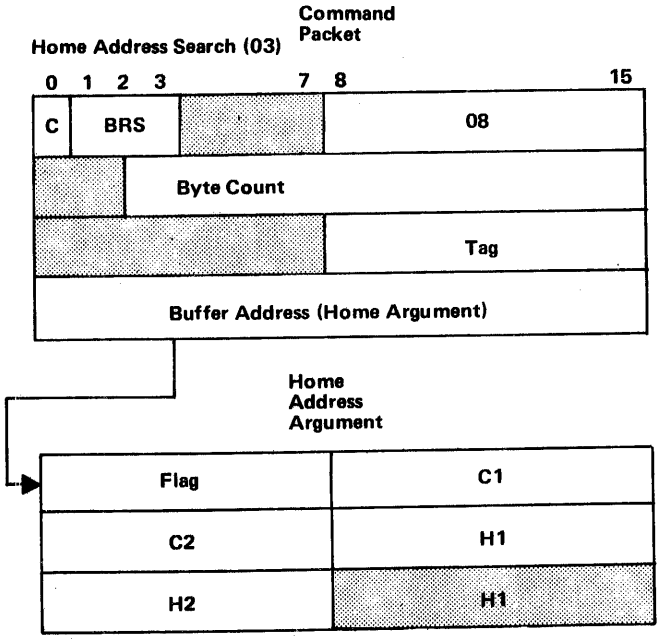
OPERAND 2: At the end of the operation, the status of the last operation to be performed is transferred to a 6-byte table. The table starting address is contained in the general register specified by R_2. The table has the following format:

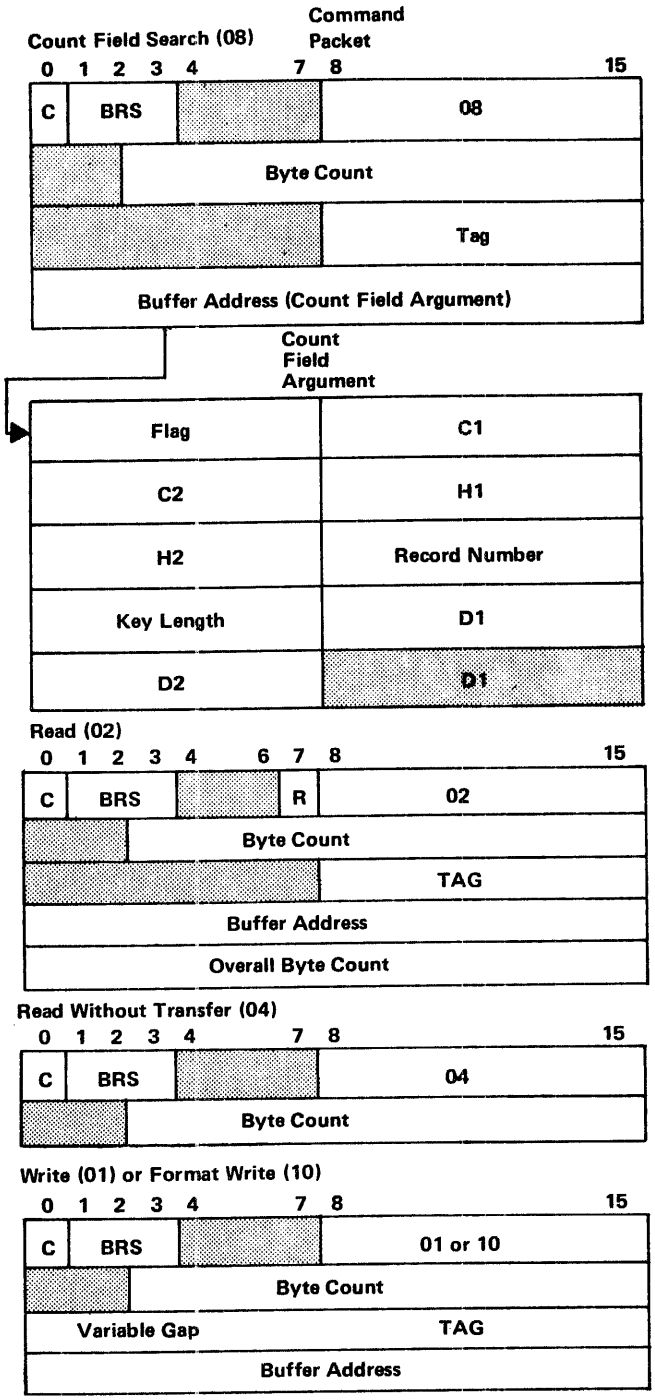


*+1 if byte count is even.
+2 if byte count is odd.

Command Packets

The DIO instruction operates on the following packets:





BRS (Bit Ring Sync) – determines the type of field (home address, count, key or data) to be read or written.

000 – Special Home Address
001 – Data Field
010 – Key Field
011 – R_0 Count Field
100 – Control Storage Data
101 – Home Address Record
110 – R_n Count Field
111 – Special Data

R_0 is the first record on a track, R_n is any record after the first record on the same track.

Byte Count – Specifies the number of bytes in the command argument (search packet), or the total number of bytes to be read or written.

Tag – Adds base register displacement to the buffer address.

Buffer Address – Specifies the starting address of the argument (search packet), or the starting address which will be used to store data from the disc, or where data to be recorded on the disc is located. Buffer address must be stored in an even memory location.

Flag – Indicates track usage and condition.

C1, C2 – Specifies the cylinder address. C1 is always set to zero; C2 can have a value between 000 and 202.

H1, H2 – Specifies the head address. H1 is always set to zero; H2 can have a value between 00 and 19.

Key Length – Specifies the number of bytes in the key field. If no key field is included, key length must be set to zero.

Overall Byte Count – Specifies total length of all data fields to be read during respective read (multiple of byte count when R=1). When R=0 this word must not be used and must not be included in the command packet.

Variable Gap – Specifies the one's complement of the hex number of supplementary gap bytes which must precede the next count field to be written. IFA logic automatically generates 41 gap bytes for every type of data field. The following table specifies the number of additional gap bytes which must be appended by the command after writing each data field.

Type of Field	Entry
R_0 Count, R_0 Data, R_n Data, R_n Key	FF_{16} (0 bytes)
Home Address*	DF_{16} (32_{10} bytes)
Special Home Address*, Special Data	Not Defined
R_n Count	0.043 x (Key Length + Data Length) not including Burst Check Bytes

*In the event of Home Address error, a gap length of 205 bytes will be inserted by the utility program.

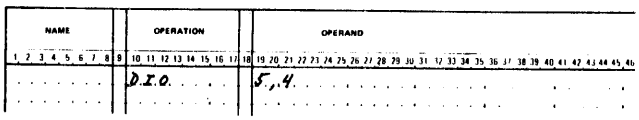
R (Repetitive Read Indicator) – If 0 = no multiple read; if 1 = multiple read requested.

C (Chain) – If this bit = 1, DIO will expect to find another command packet immediately following the first command packet. After executing the first command, the microprogram will proceed to execute the second command. Chaining will continue until DIO reaches a command packet with a chain bit = 0, which the microprogram will assume to be the last packet to be executed. Whenever a packet in the chain does not complete successfully, the chain is terminated and the status of the command packet being executed is returned to Shared Resources.

NOTE

A complete discussion of disc addressing, track formats and other pertinent information can be found in the publication Memorex 3664 Disc Storage Drive Reference.

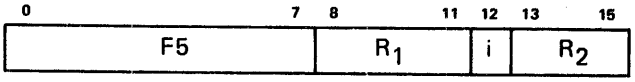
EXAMPLE



Register 5 contains a pointer to a set of I/O commands which are to be executed. After execution, the three-word end-operation status is stored at the location specified in register 4.

Input from I/O Register

INP R_1,@R_2



FUNCTION: Reads the contents of the Group III extended register specified by R_1 and transfers it to the general register specified by R_2 or to a memory location if indirect addressing is used. This is a restricted instruction limited to the I/O processors (0-3).

OPERAND 1: The data is read from the extended (Group III) register specified by the R_1 value.

OPERAND 2: For direct addressing, the receiving field is the general register specified by R_2. For indirect addressing, the receiving field is at the memory location specified by the contents of this register.

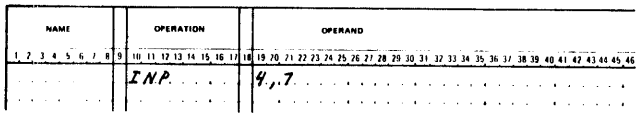
Application

The INP instruction is used by the disc and BDC (basic data channel) processors for setting up an operation in preparation for the actual data transfer.

Basic Data Channel Operations – For BDC operations in INP instruction is executed in Processor 2. The extended registers of Processor 2 have dedicated hardware functions such that the particular register number specified by field R_1 determines the hardware significance of the input word transferred. Figure 4-5 shows the relationship between the R_1 value, the extended register number, and the hardware significance of the input word.

Disc Channel Operations – For disc operations, the INP instruction is executed in Processor 3. Here the receiving field is determined by R_2; the value of the R_1 field is immaterial since the source of the word transferred is always the same. The nature of the word depends upon the value of the R_1 field in the previously executed OUT instruction.

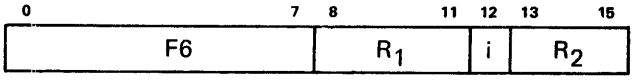
EXAMPLE



The contents of extended register 4 are transferred to general register 7.

Output to I/O Register

OUT R_1,@R_2



FUNCTION: Transfers the contents of the general register specified by R_2, or the contents of a memory location if indirect addressing is used, to the group III extended register specified by R_1. This is a restricted instruction limited to the I/O processors (0-3).

OPERAND 1: The data is transferred to the Group III extended register specified by the R_1 value.

OPERAND 2: For direct addressing, the sending field is the general register specified by R_2; for indirect addressing, the sending field is the memory location specified by the contents of this register.

Application

The OUT instruction is used by the disc and BDC (basic data channel) processors for setting up an operation in preparation for the actual data transfer.

R_1	Extended Register No.	Contents of Input Word
0, 4, 8, C	10, 14, 18, 1C	16-bit Mode: 0 7 8 15 — Bus In Register | Bus In Register; Byte Mode
1, 5, 9, D	11, 15, 19, 1D	0 7 8 15 — Tag In Lines | Tag Out Register
2, 6, A, E	12, 16, 1A, 1E	0 15 — Channel Control Register
3, 7, B, F	13, 17, 1B, 1F	0 15 — Byte Count Register

Figure 4-5. INP Instruction In Basic Data Channel Operation

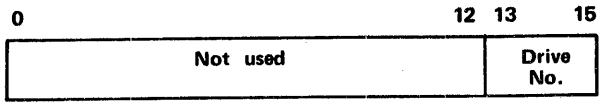
R_1	Extended Register Number (Hex)	Hardware Interpretation of Output Word
1	11	0 7 8 15 — | Tagout Register
2	12	0 15 — Channel Control Register
4	14	0 15 — Bus Out Register
8	18	0 15 — Byte Count Register
F	1F	0 15 — All Registers

Figure 4-6. OUT Instruction In Basic Data Channel Operation

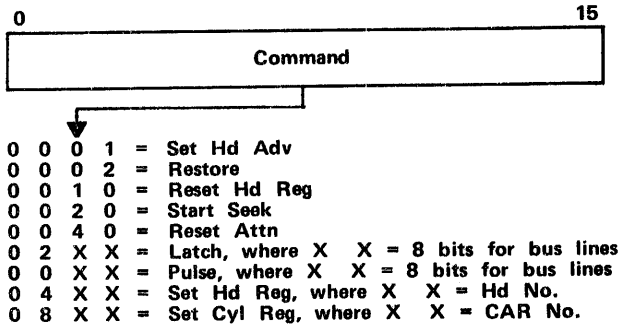
Basic Data Channel Operations – For BDC operations the OUT instruction is executed in Processor 2. The Group III extended registers of Processor 2 have dedicated hardware functions such that the particular register number specified by field R_1 determines the hardware significance of the output word. Figure 4-6 shows the relationship between the R_1 value, extended register number, and the hardware interpretation of the output word.

Disc Channel Operations – For disc operations, the OUT instruction is executed in Processor 3. The Group III extended registers of Processor 3 have dedicated functions such that the particular register number specified by field R_1 determines the hardware significance that will be assumed for the output word. The following material shows the relationship between the extended register number and the hardware interpretation of the output word.

Extended Register 10



Extended Register 11



Extended Register 12

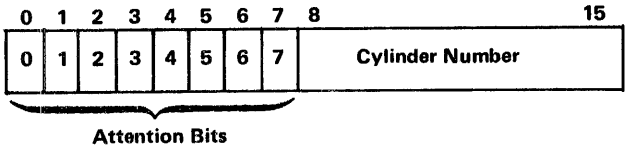
This register causes a request status indication from the IFA. The output word is immaterial in this case since the hardware will always return a status word to be picked up by an INP instruction. Significance of the status word is as shown in the following table.

8000	IFA Status Not Valid OR Command Early
4000	IFA Missed Window OR Command Early
2000	IFA Window
1000	IFA Track Boundary
0800	IFA Read/Write Termination
0400	IFA Burst Check Error
0206	IFA Lost Data
0207	IFA No Sync Compare
0080	IFA 3rd Rev Sync Find
0040	Disc (Not On Line) OR (Seek Incomplete and Not File Unsafe)
0020	Disc (File Unsafe) OR (Seek Incomplete and Not File Unsafe)
0010	Disc Read Only
0008	Disc Pack Change
0004	Disc End of Cylinder
0002	Disc Write Current Sense or No Search Find*
0001	Disc Busy

*No Search Find will exist only when the command word in error is a Search command word.

Extended Register 13

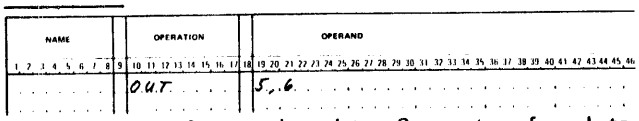
This register causes a request attention and physical address from the IFA. The output word is immaterial in this case since the hardware will always return the following word to be picked up by an INP instruction.



Extended Registers 14, 15

Not used for OUT instruction (see DIO instruction).

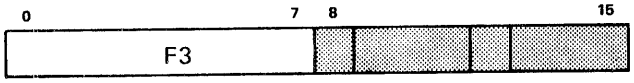
EXAMPLE



The contents of general register 6 are transferred to extended register 5.

Communications Input/Output

RDC



FUNCTION: Each time processor state 0 is turned on (Busy bit set), the Communications I/O driver executes the RDC instruction to determine whether the Busy bit was set by the Executive program (output) or by one of the communication lines (input). If there is no interface request (that is, no call-in on any of the communications lines), general register 5 is set to other than zero and execution falls through to the next instruction in the driver.

If there is an interface request, RDC places the line address in general register 2 and input information in general register 3, and then inspects the line address table (the starting address of which had previously been put in general register 4) to determine whether or not a line parameter table is assigned to that line. If there is no line parameter table, it indicates that activity on that line was neither expected nor should be acknowledged; general register 5 is reset, and execution then falls through to the next instruction in the driver.

If a line parameter table is assigned, the address of that table is placed in general register 5, and further analysis is performed to determine whether the request is to be handled by software or whether buffered data transfer (input or output) is indicated, in which case the RDC will implement the actual transfer by operating on the line

parameter table. If the interface request requires software processing, RDC will first determine the type of request and then transfer control to the appropriate software routine. The address of the routine is obtained from a jump table pointed to by one of the words in the parameter table.

Assuming that control is not transferred to a software routine, RDC will continue to service buffered data transfer requests in a multiplexed mode until control is diverted to a software routine.

The RDC instruction is a restricted instruction limited to Processor 0.

Line Address Table

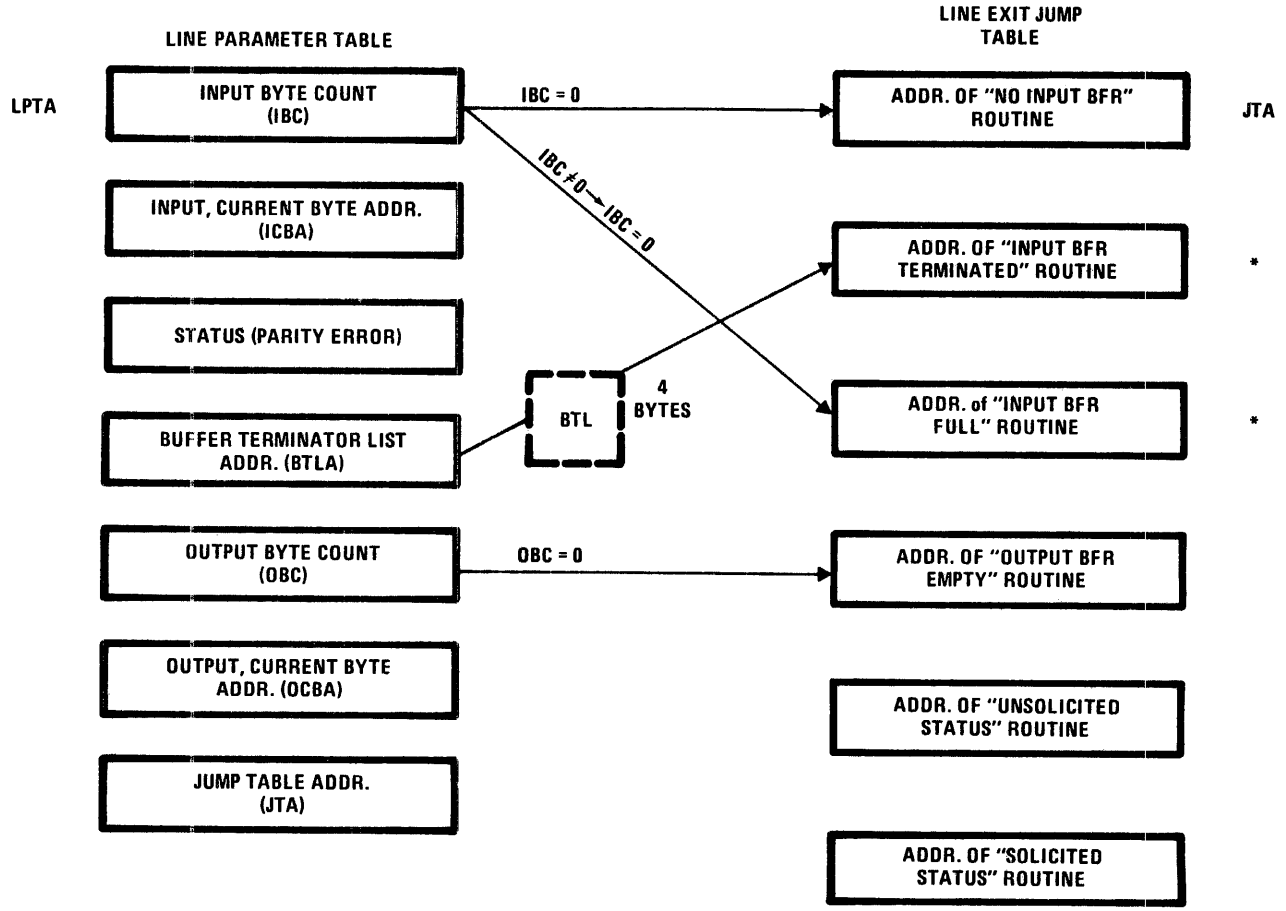
A 32-byte table containing a one-word address pointer to a line parameter table for each line address. If the pointer is zero, a line parameter table has not been provided for that line address.

Line Parameter Table (Unit Table Extension)

A block of information concerning a particular line address. Thus, it is possible for each line to be uniquely implemented. One of the words in each parameter table is a pointer to a line exit jump table (Figure 4-12).

A 12-byte table containing one-word address pointers to software routines for each line exit condition. The pointer to the line exit jump table contained in the line parameter table may be changed to a different line exit jump table, or any element of a line exit jump table may be changed to meet a line-modem situation.

The interrelationship of the Line Parameter table and the Line Exit Jump table is shown in Figure 4-7.



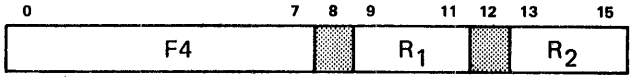
THE LINE ADDRESS TABLE (SEE TEXT) POINTS TO A DIFFERENT LINE PARAMETER TABLE ADDRESS (LPTA) FOR EACH COMMUNICATION LINE.

*THESE TWO ROUTINES CHECK WORD 3 OF THE PARAMETER TABLE TO SEE IF A PARITY ERROR OCCURRED.

Figure 4-7. Relationship, Line Parameter Table and Exit Jump Table

Communications Output

WRC R_1,R_2



FUNCTION: Transfers the contents (data or ICA command) of the general register specified by R_2 to the line address specified by R_1. This is a restricted instruction limited to Processor 0.

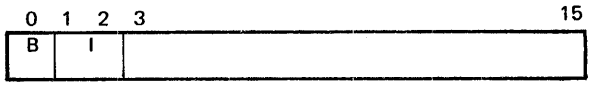
OPERAND 1: The destination of the ICA command format (or data) is the line address contained in the general register specified by R_1.

OPERAND 2: The command (or data) to be transferred is contained in the general register specified by R_2.

Application

The WRC instruction is designed for putting out a single ICA command (or data) format to a communications line in preparation for a data transmission to be implemented by the RDC instruction.

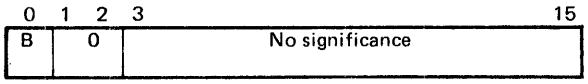
ICA Command Format: The general format of the output word transferred by the WRC instruction is shown in the first illustration. The specific variations of the word are shown in following illustrations.



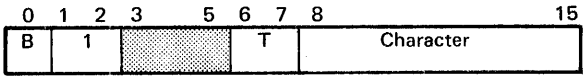
B is a broadcast designator; broadcast=1, not broadcast=0. When this bit is a 1, the command word is "broadcast" to all channels.

I is an identifier; its value, 0-3, determines the significance of bits 3-15.

If I = 0, (Input Request), the word has the following format.



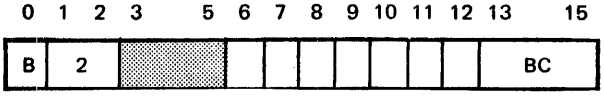
If I = 1, (Output Character), the word has the following format.



Bits 6-7 define the contents of bits 8-15 according to the following table.

T	Character
0	Data character
1	Control character
2	Dial digit
3	Reserved

If I = 2 (port command), the word has the following format.



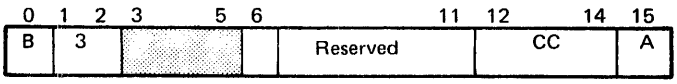
Each of the bits 6-12 has an assigned meaning, as listed.

Bit	Command Code
6	Return unsolicited status, bit = 1
7	Return solicited status, bit = 1
8	Clear link commands, bit = 1
9	8-level = 1; 7-level + parity = 0
10	Odd parity = 1; even parity = 0
11	Echoplex: invoke = 1; revoke = 0
12	Speed: split = 1; not split = 0

BC is a baud code; the values of BC (0-7) have the meanings listed.

BC	Baud Code
0	Not used
1	Not used
2	1200 baud, sync
3	1200 baud, 120 cps
4	600 baud, 60 cps
5	300 baud, 30 cps
6	150 baud, 15 cps
7	110 baud, 10 cps

If I = 3 (Link Command), the word has the following format.

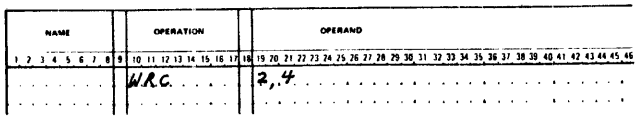


Bits 12-14 define a command code; the meanings assigned to the possible values (0-7) are listed.

CC	Command Code
0	Data terminal ready
1	Request to send
2	Transmit space clamp
3	Secondary request to send
4	Off hook
5	Dial request
6	Half duplex
7	Not loop test

A is an action bit: action if 1, no action if 0.

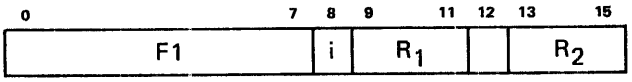
EXAMPLE



The command (or data) in register 4 is transferred to the line address in register 2.

System Input/Output

SIO @R_1,R_2



FUNCTION: Operates on a four-word command packet specified by R_1 and a one-word flag and address operand contained in the general register specified by R_2 in order to effect one of the following:

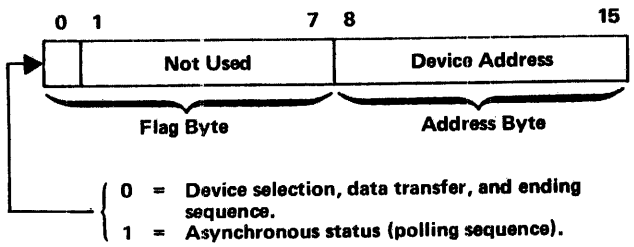
1. Basic Data channel set-up, selection sequence, hardware controlled data transfer (input or output), and an ending sequence.

2. Process Basic Data channel asynchronous status (polling sequence).

This is a restricted instruction limited to Processor 2.

OPERAND 1: A one-word field containing the first word address of a four-word command packet. The operand is in the general register specified by R_1 for direct addressing, or in the memory location specified by the contents of R_1 for indirect addressing. If in the flag byte in operand 2, bit 0=1 (polling sequence), operand 1 is ignored.

OPERAND 2: A one-word operand located in the general register specified by R_2. The operand has the following format.

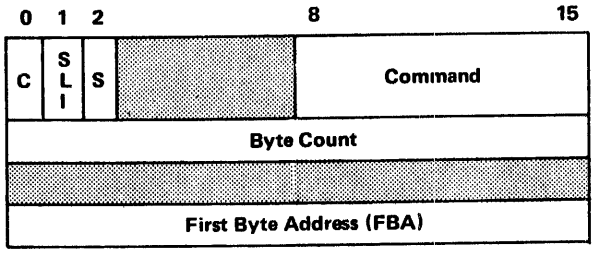


If bit 0 = 0, bits 8-15 contain the address of the byte to be selected. When bit 0 = 1, the device address is ignored.

Bits 8-15 contain the address of the device to be selected if bit 0=0. When bit 0=1, the device address is ignored.

Command Packet

The format of the command packet which the SIO instruction expects to find at the location specified by R_1 is:



Bit 0 is a chaining indicator used by software. Bit 1 is a suppress incorrect length indicator used by software. Bit 2 is a skip data-transfer bit used by the SIO to perform input operations without transferring data; if bit 2=0, normal input is assumed, if bit 2=1, input without transfer is assumed. Bits 8-15 contain the command byte issued to the device selected. The command byte codes are IBM-compatible, with the following exception: if a command code is non-zero (bits 8-12 non-zero) and bits 13-15 are zero, an invalid command status indication is returned.

If the command is not the exception above and a data transfer is initiated, bit 15 indicates whether the operation is read or a write: 0=read, 1=write.

The second word of the command packet holds a 16-bit byte count which must be non-zero, otherwise an invalid command status is returned. The non-zero count is required for all commands, including TEST I/O and control commands, which do not include a data transfer. For commands which require a data transfer, the byte count allows transfers of 1-65,535 bytes.

The last word of the command packet contains the first-byte address of the data field in main storage.

Application

The SIO instruction has two basic uses as follows:

1. Initiating and performing hardware controlled data transfer.

2. Processing asynchronous status.

The user defines the purpose of SIO via the flag byte of operand 2. The results of an SIO can be determined by examining the returned information as described below. If bit 0 of the flag byte is 0:

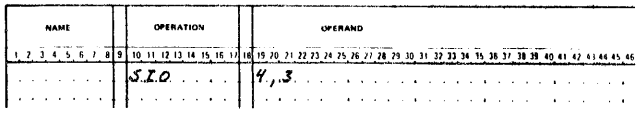
- General register 0 contains the command packet address.
- General register 1 contains the residual count of the data transfer.
- General register 2 contains 8 bits of device status and 7 bits of channel status.

If bit 0 of the flag byte is 1:

- General register 0 contains zero.
- General register 1 contains the address of the device which responded to the asynchronous status sequence, or zero if none of the devices responded.
- General register 2 contains 8 bits of device status and 7 bits of channel status.

Status information returned in general register 2 is explained in the table (right). The on condition (bit=1) for each bit indicates the status described.

EXAMPLE



The flag byte in register 3 is examined and if bit 0 is off (0):

1. The device address (bits 8-15 of register 3) is selected.
2. The six byte I/O command word pointed to by the address in register 4 is read and executed.
3. After execution (channel end or error condition) registers 0, 1, and 2 contain the result of the operation.

If bit 0 of the flag byte in register 3 is on (1):

1. The REQUEST IN tag line is examined.
2. If REQUEST IN is down, an immediate exit is made indicating this fact.
3. If REQUEST IN is up, the device address and status are returned in general registers 1 and 2, respectively.

General Register 2 Bit Number	Unit Status Information
0	Attention
1	Status Modifier
2	Control Unit End
3	Busy
4	Channel End
5	Device End
6	Unit Check
7	Unit Exception
	Channel Status Information
8	Initial Selection Sequence Error: • Device off-line • Bus out parity error • Program Address error
9	Invalid Command Word
10	Channel Address/Status Check • Wrong address-in on initial selection • Address or status parity invalid
11	No "Request In" (poll sequence)
12	Control Check
13	Transmission Check (invalid parity on sense, control, or data bytes)
14	Short Buffer
15	Unused

5. SYSTEM OPERATING PROCEDURES

INTRODUCTION

This section provides the operator and programmer with information for using the System Control Panel to operate the system. It describes the function of controls and indicators associated with the operator group, programmer group, system activity display group, and communication activity display group; plus step-by-step procedures for the most commonly used operator and programmer operations executed by the panel. 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.

The System Control Panel is illustrated in Figure 5-1.

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 leftward and upward.

NOTE

All pushbuttons are of the momentary-action type unless otherwise specified.

OPERATOR GROUP

EMERGENCY PULL Knob

When pulled, instantly removes all power from system except AC power to, and power from, the +24 VDC control supply (does not go through normal power-down sequence which is the case when using POWER OFF pushbutton).

MAINTENANCE GROUP

SYSTEM ACTIVITY DISPLAY GROUP

PROGRAMMER GROUP

OPERATOR GROUP

COMMUNICATIONS ACTIVITY GROUP

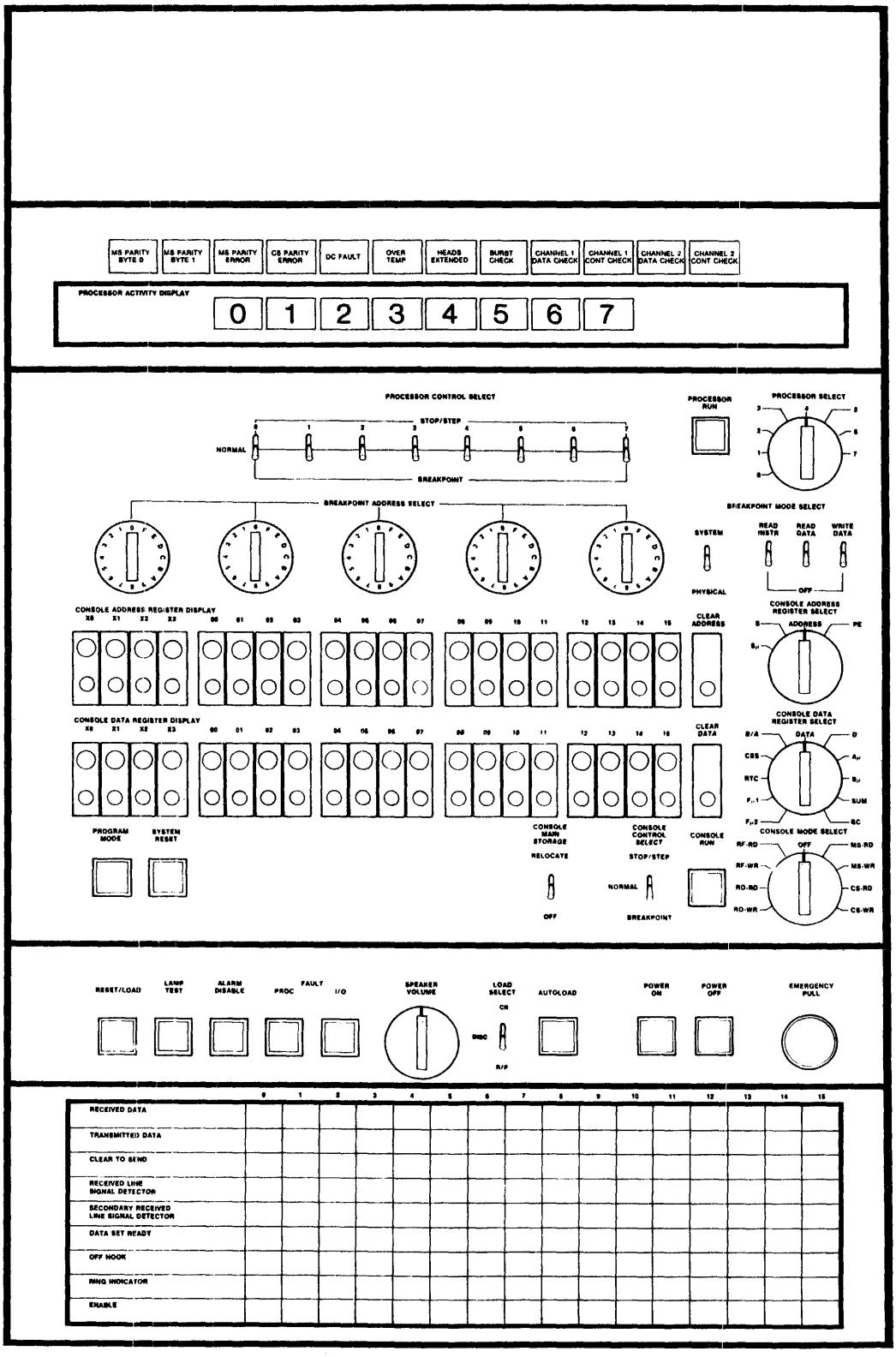


Figure 5-1. MRX/40 and 50 System Control Panel

CAUTION

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 interlock 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). Its use can cause equipment damage.

POWER OFF Pushbutton/Indicator

Turns system power off when POWER MODE switch is in LOCAL. This switch assures proper power-down sequencing. (POWER OFF has no effect while POWER MODE switch is in REMOTE.)

NOTE

POWER OFF will be lit unless one of the following conditions exist: 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 initilization 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 switch is in REMOTE.)

NOTE

Upon completion of the power-up sequence, a Reset/Load sequence is automatically initiated, provided the maintenance mode has not been selected. Moreover, at completion of the Reset/Load sequence, an Autoload sequence is automatically initiated. Further detail is provided in the procedures for loading CS and MS in either the operator or program mode.

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 LOAD SELECT switch is in PRIMARY) or from cards (when LOAD SELECT switch is in ALTERNATE.)

LOAD SELECT Switch

Determines input media used during a Reset/Load and/or Autoload sequence. Down position (PRIMARY) selects disc as input medium. Up position (ALTERNATE) selects cards as input media. In the case where a system is configured to have both a card reader and a reader/punch, the alternate source may be either and is field selectable.

NOTE

During an Autoload sequence, the PRIMARY position of this switch causes the first microprogram loader instruction to come from control storage address 0113_{16} and the ALTERNATE position causes the first microprogram loader instruction to come from control storage address 0112_{16}.*

SPEAKER Volume Control

Adjusts volume of the speaker contained in the 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 levels of these bits on the speaker are these: 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 light 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.)

*At present, there is no microprogram starting at address 0112_{16} to load CS from cards; consequently, CS can be loaded only via disc drives.

PROC FAULT Pushbutton/Indicator

PROC FAULT will light 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 except in the case of a DC Voltage Fault or an Over-temperature condition. In this case, the cause must first be corrected for the switch to have effect.

ALARM (located behind Panel)

Furnishes an audible signal when the LAMP TEST pushbutton is pressed or when any of the following conditions exist:

1. Blower failure within the computer
2. DC voltage fault
3. Failure of disc heads to retract during a power-down sequence

NOTE

When blower failure or DC fault conditions exist 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 light. When the alarm condition is corrected, ALARM DISABLE will extinguish.

LAMP TEST Pushbutton

Pressing this switch causes all indicators to light 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 to be transferred either to control storage and first-level decode address table and/or to main storage; the routing depends on the setting of the CONSOLE MODE SELECT selector (CS-WR or MS-WR) and the operating mode that has been selected. Selection of cards or disc as input medium is determined by position of the LOAD SELECT switch. (See the procedures for LOADING CONTROL STORAGE FROM CARD READER for explicit instructions regarding the use of this pushbutton.)

Upon completion of a Reset/Load operation from disc, an Autoload operation will automatically be initiated providing 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 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 switch.

CONSOLE CONTROL SELECT Switch

Three-position switch governing the way in which a 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.

This switch determines whether the contents of the S-Register are interpreted as a system or physical main storage address when an MS read or write operation is occurring in the Console Control Mode. When it is in the RELOCATE (up) position, the contents of the S-Register are interpreted as a system main storage address and are converted by the relocation mechanism into a physical main storage address. When it is in the OFF (down) position, the contents of the S-Register are directly interpreted as a physical main storage address and bypass the relocation mechanism.

SYSTEM RESET Pushbutton

The SYSTEM RESET pushbutton clears the following registers:

1. EXTENDED REGISTER FILE

   Group I: $P\mu$ 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

   $A\mu$, $B\mu$, D, $S\mu$, $F\mu$-1, $F\mu$-2, and Forced Carry Register.

PROGRAM MODE Pushbutton/Indicator

Pressing this switch enables those switches located in the programmer group area of the panel. The pushbutton is lit 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.

†$F\mu 2$	Micro-Command Function register, rank 2
†$F\mu 1$	Micro-Command Function register, rank 1
†RTC	Real-Time Clock register
†CSS	Control Storage Scan register
B/A	Busy/Active register
DATA	Systems Control Panel Data register
†D	Main Storage Data register
†$A\mu$	ALU Feeder register $A\mu$
†$B\mu$	ALU Feeder register $B\mu$
†SUM	Output of ALU (sum of A and B plus the forced-carry register)
†BC	Boundary-Crossing register

(those marked † enabled in Maintenance Mode only)

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 (not functional for 7200)

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, SUM 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 pushbutton function.)

NOTE

Only the S and ADDRESS positions are enabled when Program Mode is selected. The remaining two positions of this switch require Maintenance Mode to be effective.

S	Main Storage Address register
Sμ	Control Storage Address register (enabled in Maintenance mode only)
ADDRESS	Console Address register
PE	Main Storage Parity Error Address register (enabled in Maintenance mode only)

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 (not functional for 7200)

2. Pushbutton/indicators: 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 SEL ECT selector.

BREAKPOINT MODE SELECT Switches

1. WRITE DATA Switch

When up (on), causes breakpoint stop at end of each main storage reference cycle in which data was written at a breakpoint address.

2. READ DATA Switch

When up (on), causes breakpoint stop at the end of each main storage reference cycle in which data was read at the breakpoint address.

3. READ INSTR Switch

When up (on), causes a breakpoint stop immediately after the machine language instruction is read at the breakpoint address.

4. RELOCATE/PHYSICAL Switch (not functional for 7200)

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, CS-RD, 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 – Allow selected processor state to run continuously.
3. BREAKPOINT – Allow selected processor state to run until a breakpoint-comparison equality occurs.

MAINTENANCE GROUP

Except for the MAINTENANCE MODE pushbutton, controls and indicators in the maintenance group are not described in this manual because their use is restricted to maintenance personnel only. The MAINTENANCE MODE pushbutton, used to enable controls of the maintenance group, must be in the off state (Maintenance Mode not selected) to enable operation of the operator group controls.

SYSTEM ACTIVITY DISPLAY GROUP

PROCESSOR STATE Indicators

Dynamically indicate which processor states are executing major cycles.

Status Indicators

Twelve indicators that each light for a particular status.

MS PARITY BYTE 0 – Displays state of parity bit of upper byte (bits 0 through 7) of the last word 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 the last word 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 parity error in CS or first-level decode address table.

D.C. FAULT – Indicates that one or more DC power supplies 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.

COMMUNICATIONS ACTIVITY GROUP

These indicators show the adapter/modem status for the communications channels and the integrated communications adapter as follows.

RECEIVED DATA (BB) – ON condition indicates line is in the spacing condition (binary zero). OFF condition indicates line is in the marking condition (binary one).

TRANSMITTED DATA (BA) – ON condition indicates line is in the spacing condition (binary zero). OFF condition indicates line is in the marking condition (binary one).

CLEAR TO SEND (CB) – ON condition, together with ON condition on circuits CA, CC, and CD, indicates channel is in a transmit condition.

RECEIVED LINE SIGNAL DETECTOR (CF) – ON condition indicates that the modem is receiving a signal which meets its suitability criteria for demodulation.

SECONDARY RECEIVED LINE SIGNAL DETECTOR (SCF) – ON condition indicates the proper reception (where applicable) of the SECONDARY CHANNEL signal. Used to indicate the circuit assurance status and to signal a reverse channel interrupt condition.

DATA SET READY (CC) – ON condition indicates that the modem is connected to a communication channel and, for an auto-answer network, has completed the transmission of the answer tone. For a private line network, the ON condition indicates that the modem is ready.

OFF HOOK (OH) – For an outgoing call, ON condition indicates that a call is being placed. (Dial digits are generated by pulsing this signal.)

RING INDICATOR (CE) – ON condition indicates that a ringing signal is being received via the communication channel.

NOTE

Since under normal operation the communications handler will answer a call at the first generation of the ringing signal, ON condition implies either a malfunction or that the communications channel is not enabled.

ENABLE (EN) – ON condition indicates the line adapter is enabled and is therefore not in the system reset or loop test mode.

OPERATING PROCEDURES

The following paragraphs contain procedures which may be executed from the System Control Panel. These procedures enable 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 the program mode.

MODES OF OPERATION

The System Control Panel enables the system to operate in one of two fundamental control modes: *processor control* and *console control*. These two modes 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 is selected basically by the eight PROCESSOR CONTROL SELECT switches, the PROCESSOR SELECT selector, and the PROCESSOR RUN pushbutton. This mode enables the operator, in connection with programmed operations, 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 (STOP/STEP) mode), etc. Except for the internal effects of the programs themselves, the processor mode does not allow the contents of storage to be altered.

The *console control* mode is selected basically by the CONSOLE CONTROL SELECT switch, the CONSOLE

MODE SELECT selector, and the CONSOLE RUN pushbutton. This mode does not involve any actual execution of instructions by a processor state, but allows any individual cell of main or control storage or any of the hardware registers to be displayed or altered under either hardware or software control. The panel is allocated major cycles just as through it were a ninth processor.

Each of the fundamental control modes is influenced by the three operating modes: *operator mode, program mode,* and *maintenance mode*. These modes each determine a certain level of operating capability available to the operator. The *operator mode*, selected when neither the PROGRAM MODE or MAINTENANCE MODE pushbutton is activated, restricts the operator to use of the operator group controls only. This group allows the operator to turn on and turn off the system, perform reset/load and autoload operations, and detect fault and status conditions. These operations are always available to the operator regardless of whether the system is in the processor mode or console mode. The *program mode*, selected by the PROGRAM MODE pushbutton, enables an operator to use the controls of the programmer group as well as those of the operator group. This additional capability allows the operator to place the system in either the processor control or console control mode, thus enabling operations associated with these two control modes to be carried out. The *maintenance mode*, selected by the MAINTENANCE MODE pushbutton, allows still more capability than when operating in either the processor control or console control mode. Normally, this extended capability is required only by maintenance personnel when troubleshooting the system; therefore, procedures initiated by controls of the maintenance group are not included in this manual.

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 pushbutton) or by internal operations, or if the computer is already executing instructions. During processor mode operations, the breakpoint operation is initiated by setting one of the PROCESSOR CONTROL SELECT switches to BREAKPOINT. The processor then proceeds until the storage location selected on the BREAKPOINT MODE SELECT selectors is reached, at which point the processor stops. This breakpoint stop is interpreted in one of three ways, as selected by a corresponding BREAKPOINT MODE switch: READ INSTR, READ DATA, and WRITE DATA. Having activated the READ INSTR switch will stop the processor after it reads the instruction at the breakpoint address. Having activated the READ DATA switch will stop the processor after it reaches the operand at the breakpoint address. Having activated the WRITE DATA switch will stop the processor after it stores the operand at the breakpoint address.

NOTE

Word mode addressing will not result in a breakpoint stop where the rightmost byte (odd-numbered) 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 the rightmost byte address is designated by the BREAKPOINT ADDRESS SELECT switches.

A breakpoint stop activated by the READ INSTR switch will load only the first two bytes of an instruction. Thus, the computer treats the reading of the M_1, M_2, L_1, and L_2 portion of 4-, 6-, and 8-byte instructions as operands for breakpoint purposes.

For Console Mode operations (when the CONSOLE CONTROL SELECT switch is set to BREAKPOINT), the breakpoint 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 the operator or program mode. However, the facility is available for MS references if performed in the program mode.

SWITCHING POWER ON AND OFF

To turn the processing unit on, ensure that the LOCAL/REMOTE switch is in the LOCAL position, then simply press the POWER ON pushbutton. Upon completion of the power-up sequence, the POWER ON indicator will light.

The internal power-up switching sequence for the computer and disc drives is performed by the hardware.

To turn the processing unit off, press and hold the POWER OFF pushbutton for about 2 seconds. The POWER OFF indicator will light. (The delayed action of this switch is designed to prevent turning off power inadvertently.)

NOTE

The procedures which follow require that the processing unit not be in the Maintenance Mode — evidenced when the MAINTENANCE MODE indicator is not lit. Normally, access to controls of the maintenance group by operator and programmer personnel is not permitted. If the Maintenance Mode has been selected, however (MAINTENANCE MODE indicator lit), programmer access must be made for the sole purpose of negating this mode. Gain access by raising the front cover concealing maintenance group controls. Press the MAINTENANCE MODE pushbutton; when the associated indicator goes off, the mode has been changed.

LOADING CONTROL STORAGE FROM DISC

Loading control storage (CS) from the disc via the Panel formed in one of two ways: from a power on condition or by using the RESET/LOAD pushbutton. Essentailly, the power on condition loads CS when power is initially applied to the system (pressing the POWER ON push-button); using the RESET/LOAD pushbutton loads CS in the same manner as pressing the POWER ON pushbutton but it is after power has been applied. Each of the two ways depends in which operating mode the Panel has been placed: operator mode or program mode. Generally, the operator mode provides the maximum amount of internal hardware control with the least amount of operator intervention. In contrast, the *program* mode requires a greater amount of operator intervention but provides a greater amount of flexibility in using the panel controls.

Power On Condition

Operator Mode

STEP 1 — Set the LOAD SELECT switch to PRIMARY (for disc load).

STEP 2 — Select one and only one, of the disc drives as logical drive 0 by partially inserting plug 0 into the drive select slot. (Do not insert plug all the way in at this time.)

STEP 3 — Mount disc pack and enable power to disc drive 0 by pressing the START switch. (Ensure that the READ ONLY switch is set.)

STEP 4 — Press the POWER ON pushbutton. Upon completion of the power-up sequence and the *First Seek* operation (about 1 minute) drive number 0 in the select plug will light.

STEP 5 — Set the PROGRAM MODE pushbutton to the off position.

STEP 6 — Complete selection of disc drive 0 by fully inserting plug 0. When drive heads have been restored and the power-on sequence has been completed, CS load will begin. Disc data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table, automatically followed by a load of MS.

Program Mode

STEP 1 — Set the CONSOLE MODE SELECT selector to OFF and the CONSOLE DATA REGISTER SELECT selector to DATA.

STEP 2 — Set the LOAD SELECT switch to PRIMARY (for disc load).

STEP 3 — Select one, and only one, of the disc drives as logical drive 0 by partially inserting plug 0 into the drive select slot. (Do not insert plug all the way in at this time.)

STEP 4 — Mount disc pack and enable power to disc drive 0 by pressing the START switch. (Ensure that the READ ONLY switch is set.)

STEP 5 — Press the POWER ON pushbutton. Upon completion of the power-up sequence and the *First Seek* operation (about 1 minute) drive number 0 in the select plug will light.

STEP 6 — Set the PROGRAM MODE pushbutton to the on position.

STEP 7 — Complete selection of disc drive 0 by fully inserting plug 0. When drive heads have been restored and the power on sequence has been

completed, CS load will begin. Disc data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table, automatically followed by a load of MS.

Reset/Load Condition

Operator Mode

STEP 1 — Set the PROGRAM MODE pushbutton to the off position.

STEP 2 — Set the LOAD SELECT switch to PRIMARY (for disc load).

STEP 3 — Place disc pack on the disc drive and enable power to the drive selected as logical 0 by pressing the START switch. (Ensure that the READ ONLY switch is pressed.)

STEP 4 — Press the RESET/LOAD pushbutton. Disc data will be loaded in sequential locations starting at address 0000_{16} into both CS and FRJ Decode Address Table automatically followed by a load of MS.

Program Mode

STEP 1 — Set the PROGRAM MODE pushbutton to the on position.

STEP 2 — Set the CONSOLE MODE SELECT selector to OFF and the CONSOLE DATA REGISTER SELECT selector to DATA.

STEP 3 — Set the LOAD SELECT switch to PRIMARY (for disc load).

STEP 4 — Place disc pack on the disc drive and enable power to the drive selected as logical 0 by pressing the START switch. (Ensure that the READ ONLY switch is pressed.)

STEP 5 — Press the RESET/LOAD pushbutton. Disc data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table, automatically followed by a load of MS.

LOADING CONTROL STORAGE FROM CARD READER

Loading control storage (CS) from cards can be performed in one of two ways: from a power on condition or by using the RESET/LOAD pushbutton. Essentially, the power-on condition loads CS when power is initially applied to the system (pressing the POWER ON pushbutton); using the RESET/LOAD pushbutton loads CS in the same manner as pressing the POWER ON pushbutton but it is after power is already on. Each of the two ways depends in which operating mode the panel has been placed: operator mode or program mode. Generally, the operator mode provides the maximum amount of internal hardware control with the least amount of operator intervention. In contrast, the program mode requires greater amount of operator intervention but offers greater amount of flexibility in using the Panel controls.

Power On Condition

Operator Mode

STEP 1 — Set the LOAD SELECT switch to ALTERNATE (for card reader load).

STEP 2 — Press the POWER ON pushbutton.

STEP 3 — Set the PROGRAM MODE pushbutton to the off position.

STEP 4 — Place microprogram card deck in the card reader and press the START pushbutton on the card reader. Card data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table.

Program Mode

STEP 1 Set the LOAD SELECT switch to ALTERNATE (for card reader load).

STEP 2 Press the POWER ON pushbutton.

STEP 3 Set the PROGRAM MODE pushbutton to the ON position.

STEP 4 Set the CONSOLE MODE SELECT selector to OFF and the CONSOLE DATA REGISTER SELECT selector to DATA.

STEP 5 Place microprogram card deck on the card reader and press the START pushbutton on the card reader. Card data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table, automatically followed by a load of MS.

Reset/Load Condition

Operator Mode

STEP 1 Set the PROGRAM MODE pushbutton to the OFF position.

STEP 2 Set the LOAD SELECT switch to ALTERNATE (for card reader load).

STEP 3 Place microprogram card deck in the card reader and press the START pushbutton on the card reader.

STEP 4 Press the RESET/LOAD pushbutton. Card data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table.

Program Mode

STEP 1 Set the PROGRAM MODE pushbutton to the ON position.

STEP 2 Set the CONSOLE MODE SELECT selector to OFF and the CONSOLE DATA REGISTER SELECT selector to DATA.

STEP 3 Set the LOAD SELECT switch to ALTERNATE (for card reader load).

STEP 4 Place microprogram card deck in the card reader and press the START pushbutton on the card reader.

STEP 5 Press the RESET/LOAD pushbutton. Card data will be loaded in sequential locations starting at address 0000_{16} into both CS and the FRJ Decode Address Table.

LOADING MAIN STORAGE FROM DISC

Loading main storage (MS) from the disc can be accomplished in one of three ways: from a power on condition, by using the RESET/LOAD pushbutton, or by using the AUTOLOAD pushbutton. An MS load occurs automatically, if CS was loaded from a power on or reset/load condition, after the CS and FRJ decode address table is loaded. Therefore, only the procedure for performing an MS load using the AUTOLOAD pushbutton (in either the program mode or operator mode) is given here.

NOTE

Control storge must have been previously loaded to perform this operation.

STEP 1 Set LOAD SELECT switch to PRIMARY (for disc load).

STEP 2 Place disc pack on the disc drive selected as logical 0 and apply power by pressing the START switch. (Ensure that the READ ONLY switch is pressed.)

STEP 3 Press AUTOLOAD pushbutton. Disc data will be loaded into MS in sequential locations starting at address 0000_{16}.

LOADING MAIN STORAGE FROM CARD READER

At the present time, MS cannot be loaded from the card reader in either the operator or program mode.

READING MAIN STORAGE

Preconditions†

NOTE

Control storage must have been previously loaded to perform this operation.

1. PROGRAM MODE pushbutton/indicator switch on.
2. CONSOLE MODE SELECT selector at MS-RD.
3. CONSOLE CONTROL SELECT switch at STOP/STEP.
4. CONSOLE ADDRESS REGISTER SELECT selector at ADDRESS.
5. CONSOLE DATA REGISTER SELECT selector at DATA.

Procedure

STEP 1 Press CLEAR ADDRESS pushbutton.

STEP 2 Set the address of the main storage location to be read via the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons.

STEP 3 Press the CONSOLE RUN pushbutton. Contents of selected location will be displayed in the CONSOLE DATA REGISTER DISPLAY indicators.

STEP 4 To read up to an address, enter address into BREAKPOINT ADDRESS SELECT selectors. Position CONSOLE CONTROL SELECT switch at BREAKPOINT and press CONSOLE RUN pushbutton.

STEP 5 To step through individual storage locations, repeat Step 4, except position CONSOLE CONTROL SELECT switch at STOP/STEP. Contents of each storage location will be displayed in sequence each time CONSOLE RUN pushbutton is pressed.

STEP 6 To dynamically read a storage location in the normal (continuous) mode, enter the word address of the location into the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons with bit position 15 set. Set the CONSOLE CONTROL SELECT switch at the NORMAL Position and press the CONSOLE RUN pushbutton. The contents of the storage location entered in the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons will be continuously displayed in the CONSOLE DATA REGISTER DISPLAY indicators.

WRITING MAIN STORAGE

NOTE

Control storage must have been previously loaded to perform this operation.

Preconditions

1. PROGRAM MODE pushbutton/indicator on.
2. CONSOLE MODE SELECT selector at MS-WR.
3. CONSOLE CONTROL SELECT switch at STOP/STEP.
4. CONSOLE ADDRESS REGISTER SELECT selector at ADDRESS.
5. CONSOLE DATA REGISTER SELECT selector at DATA.

† In all procedures listed, preconditions must be satisfied before the procedure can be executed.

Procedure

STEP 1 Press CLEAR ADDRESS and CLEAR DATA pushbuttons.

STEP 2 Set the address of the main storage location to be written via the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons.

STEP 3 Set the data to be written on the CONSOLE DATA REGISTER DISPLAY pushbuttons.

STEP 4 Press the CONSOLE RUN pushbutton. Contents of the CONSOLE DATA REGISTER DISPLAY will be written at the address specified in the CONSOLE ADDRESS REGISTER DISPLAY.

STEP 5 To write the data register contents into all storage locations, set CONSOLE CONTROL SELECT switch to the NORMAL position and press the CONSOLE RUN pushbutton.

STEP 6 To write a block of data, enter starting address of block via the CONSOLE ADDRESS REGISTER SELECT pushbuttons and ending address via the BREAKPOINT ADDRESS SELECT selectors. Position CONSOLE CONTROL SELECT switch at BREAKPOINT and press the CONSOLE RUN pushbutton. Contents of the data register will be written in sequence in all locations within the block.

STEP 7 To write data into individual storage locations within the block, repeat Step 6, except set the CONSOLE CONTROL SELECT switch at STOP/STEP. Contents of the data register will be written in individual locations in the sequence, each time the CONSOLE RUN pushbutton is pressed.

READING REGISTERS OF REGISTER FILES

NOTE

Control storage must have been previously loaded to perform this operation.

Preconditions

1. PROGRAM MODE pushbutton/indicator on.
2. CONSOLE MODE SELECT selector at RF-RD.
3. CONSOLE CONTROL SELECT switch at STOP/STEP.
4. CONSOLE ADDRESS REGISTER SELECT selector at ADDRESS.
5. CONSOLE DATA REGISTER SELECT selector at DATA.

Procedure

STEP 1 Press CLEAR ADDRESS pushbutton.

STEP 2 Set processor state number and number of the register in basic file or extended file (Group I or II[†]) via the CONSOLE ADDRESS REGISTER SELECT pushbuttons as shown in Figure 5-2. The addresses of basic file and extended file, Groups I and II are listed in Figure 5-3.

STEP 3 Press CONSOLE RUN pushbutton. Contents of selected register will be displayed in the bits 00 through 15 indicators of the CONSOLE DATA REGISTER DISPLAY.

STEP 4 To dynamically read a register in the normal (continuous) mode, repeat Steps 1 through 3, except set the CONSOLE CONTROL SELECT switch at the NORMAL position. The CONSOLE DATA REGISTER DISPLAY indicators will continuously display the register contents as the running processor alters the contents.

[†] The Group III registers of the extended register file may not be addressed by this mechanism.

LOADING REGISTERS OF REGISTER FILES

Preconditions

NOTE

Control storage must have been previously loaded to perform this operation.

1. PROGRAM MODE pushbutton/indicator on.
2. CONSOLE MODE SELECT selector at RF-WR.
3. CONSOLE CONTROL SELECT switch at STOP/STEP.
4. CONSOLE ADDRESS REGISTER SELECT selector at ADDRESS.
5. CONSOLE DATA REGISTER SELECT selector at DATA.

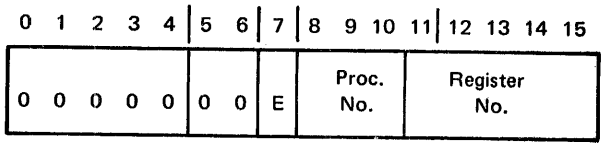
Procedure

STEP 1 Press CLEAR ADDRESS and CLEAR DATA pushbuttons.

STEP 2 Set processor state number and number of the register in basic file or extended file (Group I or II†) via the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons as shown in Figure 5-2. The addresses of the basic files and extended file registers are listed in Figure 5-3.

STEP 3 Set data to be loaded via the 00 through 15 pushbuttons of the CONSOLE DATA REGISTER DISPLAY.

STEP 4 Press CONSOLE RUN pushbutton. Contents of the CONSOLE DATA REGISTER DISPLAY will be loaded into the selected processor register.



E = extended register designator (signifies extended register when set, basic register when clear).

Figure 5-2. Register File Address Format

READING REGISTERS OF REGISTER OPTION

Preconditions

NOTE

Control storage must have been previously loaded to perform this operation.

1. PROGRAM MODE pushbutton/indicator on.
2. CONSOLE MODE SELECT selector at RO-RD.
3. CONSOLE CONTROL SELECT switch at STOP/STEP.
4. CONSOLE ADDRESS REGISTER SELECT selector at ADDRESS.
5. CONSOLE DATA REGISTER SELECT selector at DATA.

Procedure

STEP 1 Press CLEAR ADDRESS button.

STEP 2 Set feature number, processor state number, and register number via the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons as shown in Figure 5-4. There are two basic register address formats, depending on the feature selected.

†The I/O registers of the extended register file may not be addressed by this mechanism.

The first format is used to address feature registers associated with particular processor states. This format requires specifying the feature number only. Addresses of all registers in the register option are shown in Figure 5-5.

STEP 3 Press CONSOLE RUN pushbutton. Contents of selected register will be displayed in the CONSOLE DATA REGISTER DISPLAY indicators. (Contents of 4-bit registers will be right-justified.)

LOADING REGISTERS OF REGISTER OPTION

Preconditions

NOTE

Control storage must have been previously loaded to perfrom this operation.

1. PROGRAM MODE pushbutton/indicators on.
2. CONSOLE MODE SELECT selector at RO-WR.
3. CONSOLE CONTROL SELECT switch at STOP/STEP.
4. CONSOLE ADDRESS REGISTER SELECT selector at ADDRESS.
5. CONSOLE DATA REGISTER SELECT selector at DATA.

Procedure

STEP 1 Press CLEAR ADDRESS and CLEAR DATA pushbuttons.

STEP 2 Set feature number, processor state number, and register number via the CONSOLE ADDRESS REGISTER DISPLAY pushbuttons as shown on Figure 5-4. Addresses of all registers of the option are listed in Figure 5-5.

STEP 3 Set the data to be loaded via the CONSOLE DATA REGISTER DISPLAY pushbuttons. (Data to be set into 4-bit registers should be right-justified in the display.)

STEP 4 Press CONSOLE RUN pushbutton. Contents of the CONSOLE DATA REGISTER DISPLAY will be loaded into the selected register.

READING SHARED RESOURCES REGISTERS

Preconditions: PROGRAM MODE indicator on.

Procedure

STEP 1 Press CLEAR ADDRESS and CLEAR DATA pushbuttons.

NOTE

Only the S and ADDR positions of the CONSOLE ADDRESS REGISTER SELECT selector and the B/A and DATA positions of the CONSOLE DATA REGISTER SELECT selector may be enabled in the program mode.

STEP 2 Set either CONSOLE ADDRESS REGISTER SELECT or CONSOLE DATA REGISTER SELECT selector to the register to be read. The contents of the register selected will be dynamically displayed in either the CONSOLE ADDRESS or CONSOLE DATA REGISTER DISPLAY indicators.

EXECUTING PROGRAMS

Precondition: PROGRAM MODE indicator on.

Procedure

STEP 1 Use the previously described *Loading-Register-of-Register-File* procedure to load specified main storage starting address into the P-Register (R9) of the processor to be run.

CONSOLE ADRS REG

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

CONSOLE DATA REG

0	MS Starting Address	15

NOTE

If it is specified that the processor should start at a particular location other than micro-program position 0000_{16} in control storage, use the procedure in STEP 2 to load the address of this location into Pμ via the CONSOLE DATA REGISTER DISPLAY pushbuttons.

STEP 2 Use the *Loading-Register-of-Register-File* procedure to load 0000_{16} into the P -Register (extended R1) of the processor† to be run:

CONSOLE ADRS REG

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

CONSOLE DATA REG

0															15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

† If no other processor states are in use (executingμI subroutines), the SYSTEM RESET button may be used instead to clear Pμ (it clears the Pμ of all processor states).

STEP 3 Select processor state on the PROCESSOR SELECT selector.

STEP 4* Select mode of operation on appropriate PROCESSOR CONTROL SELECT SELECT switch.

STEP 5* If processor state is set to the breakpoint mode, select breakpoint address on BREAKPOINT ADDRESS SELECT selectors and select type of breakpoint on READ INSTR, READ DATA, or WRITE DATA switches.

STEP 6 Press PROCESSOR RUN pushbutton. Selected processor state will execute machine-language instructions commencing at location contained in its P-Register. (If the PROCESSOR CONTROL SELECT switch is set to the STOP/STEP position, only one machine language instruction is executed each time the PROCESSOR RUN pushbutton is pressed. The STOP/STEP position also disables I/O-originated start signals (REQUEST and ATTENTION) for processor states 0 through 3.)

NOTE

Selecting the breakpoint mode for processor state 4 will stop and lock out all processor 4 start signals except those originating from the panel.

STEP 7 Repeat steps 1 through 6 for other processors to be run.

*Steps 4 and 5 may be executed after Step 6 for Program stop/step or breakpoint operations.

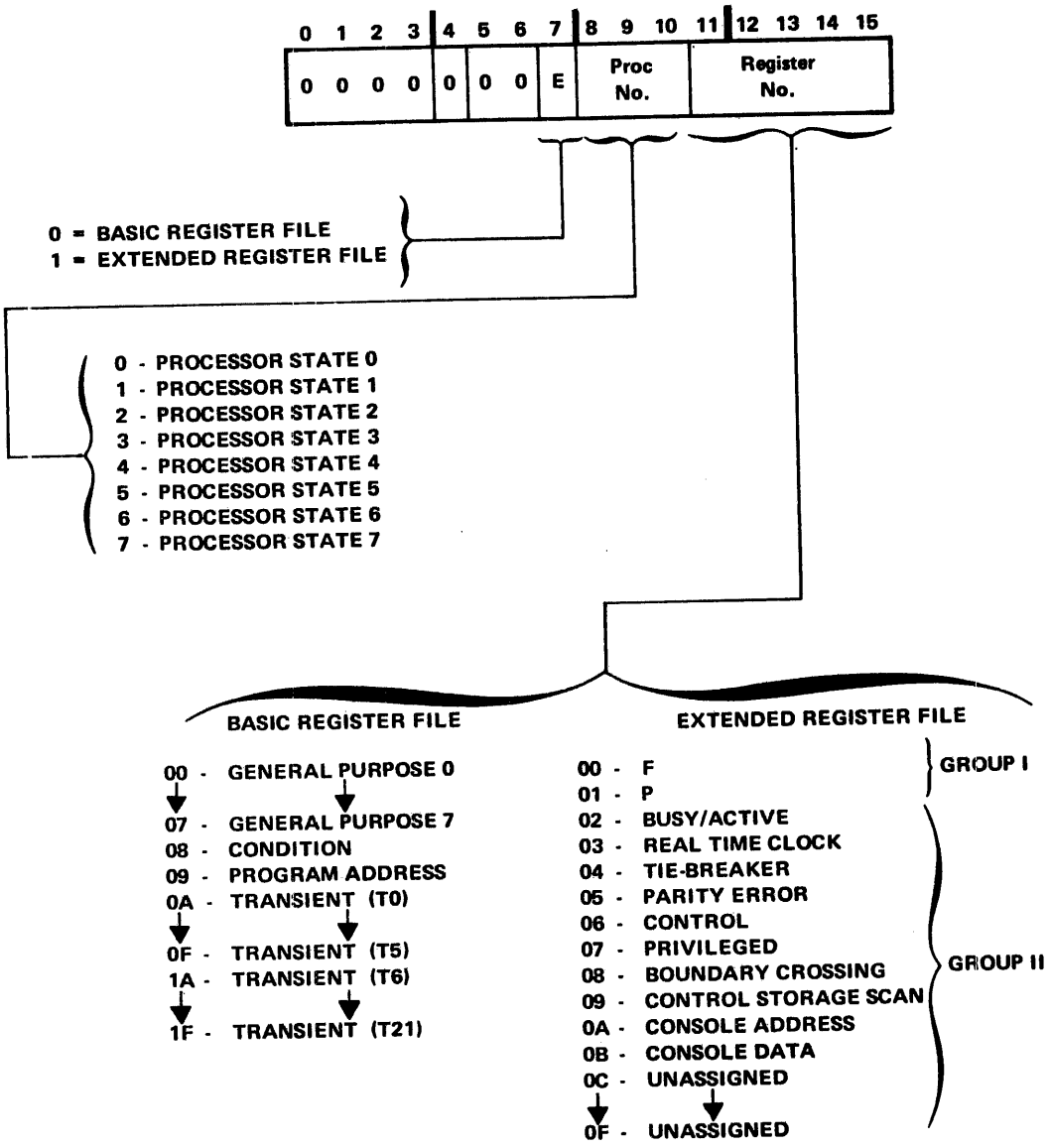
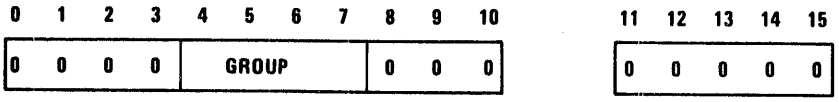
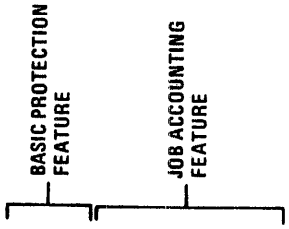


Figure 5-3. Register File and Associated Register Addresses



ADDRESS MODE, PE TAG, MS DATA, LOG, GENERATED CHECK BITS, AND READ CHECK BITS REGISTERS

Figure 5-4. Format: Registers of Register Option



ALL REGISTER ADDRESSES IN HEXADECIMAL FORM.

	JOB ACCOUNTING REG. FILE (GROUP 6) WD0	WD1
BOUNDS REG. (GROUP 5)	0600	0601
	0620	0621
	0640	0641
	0660	0661
	0680	0681
05A0	06A0	06A1
05C0	06C0	06C1
05E0	06E0	06E1

Figure 5-5. Addresses: Registers of Register Option

APPENDICES

A. INSTRUCTION SUMMARY and EXTENDED MNEMONIC CODES

The following symbols are used to define the source operands of the instruction set.

R A general register number, 0-7. The register may be used as a sending or receiving field (0-7), or as an index register (1-7 only).

E Extended register, 0-15. (For RDX and WRX only.)

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 Field length, 0-255 (for MOVL: 0-65,535), an optional feature. 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 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 by parentheses in a source operand. A source operand using both an indexing and a field length specification would be represented like this: $M_1(L_1,R_1)$. The comma in the parentheses must not only be coded when both the length and index register are used, but also if either one of them is used, as follows: $M_1(L_1,)$ or $M_1(,R_1)$. This enables the assembler to distinguish between the two specifications in parentheses.

• A bullet following an instruction name indicates the operands are byte-addressable; all other operands are word-addressable only.

GENERAL-PURPOSE INSTRUCTIONS

ARITHMETIC

Mnemonic	Name	Code	Length	Operands
ADD	Add Memory – Register	A2	4	$@M_1(R_1),@R_2$
ADDD	Add Direct	B2	4	$I_1(R_1),@R_2$
ADDI	Add Immediate	32	2	$I_1,@R_2$
ADDK	Add Packed Decimal •	52	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
ADDM	Add Memory – Memory	62	6	$@M_1(R_1),@M_2(R_2)$
ADDR	Add Register – Register	22	2	$@R_1,@R_2$
ADDT	Add Two-Word	72	4	$@M_1(R_1),@R_2$
DIV	Divide Memory – Register	A9	4	$@M_1(R_1),@R_2$
DIVD	Divide Direct	B9	4	$I_1(R_1),@R_2$
DIVI	Divide Immediate	39	2	$I_1,@R_2$
DIVK	Divide Packed Decimal •	7C	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
DIVM	Divide Memory – Memory	69	6	$@M_1(R_1),@M_2(R_2)$
DIVR	Divide Register – Register	29	2	$@R_1,@R_2$
MPY	Multiply Memory – Register	A8	4	$@M_1(R_1),@R_2$
MPYD	Multiply Direct	B8	4	$I_1(R_1),@R_2$
MPYI	Multiply Immediate	38	2	$I_1,@R_2$
MPYK	Multiply Packed Decimal •	5B	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
MPYM	Multiply Memory – Memory	68	6	$@M_1(R_1),@M_2(R_2)$
MPYR	Multiply Register – Register	28	2	$@R_1,@R_2$
SUB	Subtract Memory – Register	A3	4	$@M_1(R_1),@R_2$
SUBD	Subtract Direct	B3	4	$I_1(R_1),@R_2$
SUBI	Subtract Immediate	33	2	$I_1,@R_2$
SUBK	Subtract Packed Decimal •	53	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
SUBM	Subtract Memory – Memory	63	6	$@M_1(R_1),@M_2(R_2)$
SUBR	Subtract Register – Register	23	2	$@R_1,@R_2$
SUBT	Subtract Two-Word	73	4	$@M_1(R_1),@R_2$
ZADK	Zero and Add •	50	8	$M_1(L_1,R_1),M_2(L_2,R_2)$

BIT-ORIENTED INSTRUCTIONS

Mnemonic	Name	Code	Length	Operands
IBIT	Invert Bit •	BF	4	$@M_1(R_1),I_2$
ROFR	Reverse Off-Bit	6F	2	$@R_1,@R_2$
RONR	Reverse On-Bit	6D	2	$@R_1,@R_2$
SBIT	Set Bit •	BC	4	$@M_1(R_1),I_2$
RBIT	Reset Bit •	BD	4	$@M_1(R_1),I_2$
TBIT	Test Bit •	BE	4	$@M_1(R_1),I_2$
TOFR	Test for Off-Bit	6E	2	$@R_1,@R_2$
TONR	Test for On-Bit	6C	2	$@R_1,@R_2$

BOOLEAN LOGIC INSTRUCTIONS

Mnemonic	Name	Code	Length	Operands
AND	Logical Product Memory – Register	A5	4	$@M_1(R_1),@R_2$
ANDD	Logical Product Direct	B5	4	$I_1(R_1),@R_2$
ANDI	Logical Product Immediate	35	2	$I_1,@R_2$
ANDM	Logical Product Memory – Memory	65	6	$@M_1(R_1),@M_2(R_2)$
ANDR	Logical Product Register – Register	25	2	$@R_1,@R_2$

BOOLEAN LOGIC INSTRUCTIONS (Continued)

Mnemonic	Name	Code	Length	Operands
EOR	Exclusive OR Memory – Register	A6	4	$@M_1(R_1),@R_2$
EORD	Exclusive OR Direct	B6	4	$I_1(R_1),@R_2$
EORI	Exclusive OR Immediate	36	2	$I_1,@R_2$
EORM	Exclusive OR Memory – Memory	66	6	$@M_1(R_1),@M_2(R_2)$
EORR	Exclusive OR Register – Register	26	2	$@R_1,@R_2$
IOR	Inclusive OR Memory – Register	A7	4	$@M_1(R_1),@R_2$
IORD	Inclusive OR Direct	B7	4	$I_1(R_1),@R_2$
IORI	Inclusive OR Immediate	37	2	$I_1,@R_2$
IORM	Inclusive OR Memory – Memory	67	6	$@M_1(R_1),@M_2(R_2)$
IORR	Inclusive OR Register – Register	27	2	$@R_1,@R_2$

BRANCHING INSTRUCTIONS

B	Branch (post-indexing)	ED	4	$@M_1(R_1)$
BA1	Branch Add One	E4	4	$@M_1(R_1),@R_2$
BA2	Branch Add Two	E5	4	$@M_1(R_1),@R_2$
BCF	Branch on Condition Register False	E9	4	$@M_1(R_1),I_2$
BCT	Branch on Condition Register True	E8	4	$@M_1(R_1),I_2$
BCH	Branch Unconditional (pre-indexing)	EC	4	$@M_1(R_1)$
BOF	Branch if Bit Off	E2	4	$@M_1(R_1),I_2$
BON	Branch if Bit On	E3	4	$@M_1(R_1),I_2$
BR	Branch to Address in Register	EB	2	$@R_1$
BRN	Branch if Register is Not Zero	E1	4	$@M_1(R_1),@R_2$
BRZ	Branch if Register is Zero	E0	4	$@M_1(R_1),@R_2$
BS1	Branch Subtract One	E6	4	$@M_1(R_1),@R_2$
BS2	Branch Subtract Two	E7	4	$@M_1(R_1),@R_2$
BSR	Branch and Save Return	EA	4	$@M_1(R_1),@R_2$
SB	Skip Back – Unconditional	BB	2	I_1
SF	Skip Forward – Unconditional	BA	2	I_1
SCFB	Skip on Condition False – Back	4B	2	I_1, I_2
SCFF	Skip on Condition False – Forward	49	2	I_1,I_2
SCTB	Skip on Condition True – Back	4A	2	I_1,I_2
SCTF	Skip on Condition True – Forward	48	2	I_1,I_2
SRMB	Skip if Register Minus – Back	47	2	I_1,R_2
SRMF	Skip if Register Minus – Forward	46	2	I_1,R_2
SRPB	Skip if Register Plus – Back	45	2	I_1,R_2
SRPF	Skip if Register Plus – Forward	44	2	I_1,R_2
SRNB	Skip if Register Not Zero – Back	43	2	I_1,R_2
SRNF	Skip if Register Not Zero – Forward	42	2	I_1,R_2
SRZB	Skip if Register Zero – Back	41	2	I_1,R_2
SRZF	Skip if Register Zero – Forward	40	2	I_1,R_2

COMPARE INSTRUCTIONS

Mnemonic	Name	Code	Length	Operands
CBY	Compare Byte Memory – Register •	F9	4	$@M_1(R_1),@R_2$
CBYM	Compare Byte Memory – Memory •	6B	6	$@M_1(R_1),@M_2(R_2)$
CMP	Compare Memory – Register	A1	4	$@M_1(R_1),@R_2$
CMPD	Compare Direct	B1	4	$I_1(R_1),@R_2$
CMPI	Compare Immediate	31	2	$I_1,@R_2$
CMPK	Compare Packed Decimal •	51	8	$M_1(L_1,R_1),M_2(L_2,R_2)$

COMPARE INSTRUCTIONS (Continued)

Mnemonic	Name	Code	Length	Operands
CMPM	Compare Memory – Memory	61	6	$@M_1(R_1),@M_2(R_2)$
CMPR	Compare Register – Register	21	2	$@R_1,@R_2$
CMPT	Compare Two-Word	71	4	$@M_1(R_1),@R_2$
CMPX	Compare Characters •	55	8	$M_1(L_1,R_1),M_2(L_2,R_2)$

CONTROL INSTRUCTIONS (General Purpose Control instructions can be used at any time without preconditions; compare with System Control instructions.)

NOP	No Operation	EE	4	
RDX	Read Extended Register	F0	2	E_1,R_2
SR	Service Request	13	2	$@I_1$

DATA CONVERSION INSTRUCTIONS

CVB	Convert to Binary •	AA	4	$@M_1(R_1),R_2$
CVBT	Convert to Binary Two-Word •	AA	4	$@M_1(R_1),R_2$
CVD	Convert to Decimal •	AB	4	$@M_1(R_1),R_2$
CVDT	Convert to Decimal Two-Word •	AB	4	$@M_1(R_1),R_2$
EDTX	Packed Decimal/Alpha Edit •	57	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
PAKX	Pack •	58	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
UNPX	Unpack •	59	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
TRNX	Translate •	56	8	$M_1(R_1),M_2(L_2,R_2)$

DATA TRANSFER INSTRUCTIONS

CLDR	Condition Register Load	2B	2	$@R_1$
CSTR	Condition Register Store	2A	2	$@R_1$
INV	Inverse Move Memory – Register	A4	4	$@M_1(R_1),@R_2$
INVD	Inverse Move Direct	B4	4	$I_1(R_1),@R_2$
INVI	Inverse Move Immediate	34	2	$I_1,@R_2$
INVM	Inverse Move Memory – Memory	64	6	$@M_1(R_1),@M_2(R_2)$
INVR	Inverse Move Register – Register	24	2	$@R_1,@R_2$
LOD	Load Memory – Register	A0	4	$@M_1(R_1),@R_2$
LODB	Load Byte •	F7	4	$@M_1(R_1),@R_2$
LODD	Load Direct	B0	4	$I_1(R_1),@R_2$ or $M_1(R_1),@R_2$
LODI	Load Immediate	30	2	$I_1,@R_2$
LODT	Load Two-Word	70	4	$@M_1(R_1),@R_2$
MOVB	Move Byte •	6A	6	$@M_1(R_1),@M_2(R_2)$
MOVL	Move Long •	5A	8	$M_1(L_1,R_1),M_2(R_2)$
MOVM	Move Memory – Memory	60	6	$@M_1(R_1),@M_2(R_2)$
MOVR	Move Register – Register	20	2	$@R_1,@R_2$
MOVX	Move Characters •	54	8	$M_1(L_1,R_1),M_2(L_2,R_2)$
PSTR	Program Address Store	3A	2	$@R_1$
STO	Store Memory – Register	FA	4	$@M_1(R_1),@R_2$
STOB	Store Byte •	F8	4	$@M_1(R_1),@R_2$
STOT	Store Two-Word	FB	4	$@M_1(R_1),@R_2$

SHIFT INSTRUCTIONS

Mnemonic	Name	Code	Length	Operands
ARDI	Arithmetic Right Double Shift – Immediate	5F	2	I_1,R_2
ARDR	Arithmetic Right Double Shift – by Register	3F	2	$@R_1,R_2$
ARSI	Arithmetic Right Single Shift – Immediate	4F	2	I_1,R_2
ARSR	Arithmetic Right Single Shift – by Register	2F	2	$@R_1,R_2$
LLDI	Logical Left Double Shift – Immediate	5C	2	I_1,R_2
LLDR	Logical Left Double Shift – by Register	3C	2	$@R_1,R_2$
LLSI	Logical Left Single Shift – Immediate	4C	2	I_1,R_2
LLSR	Logical Left Single Shift – by Register	2C	2	$@R_1,R_2$
LRDI	Logical Right Double Shift – Immediate	5D	2	I_1,R_2
LRDR	Logical Right Double Shift – by Register	3D	2	$@R_1,R_2$
LRSI	Logical Right Single Shift – Immediate	4D	2	I_1,R_2
LRSR	Logical Right Single Shift – by Register	2D	2	$@R_1,R_2$
RLDI	Rotating Left Double Shift – Immediate	5E	2	I_1,R_2
RLDR	Rotating Left Double Shift – by Register	3E	2	$@R_1,R_2$
RLSI	Rotating Left Single Shift – Immediate	4E	2	I_1,R_2
RLSR	Rotating Left Single Shift – by Register	2E	2	$@R_1,R_2$
SHFK	Shift Packed Decimal •	3B	6	$M_1(L_1,R_1),I_2(R_2)$

FLOATING POINT INSTRUCTIONS (OPTIONAL)

ADDF	Add Floating Point	86	4	$@M_1(R_1),R_2$
CMPF	Compare Floating Point	87	4	$@M_1(R_1)$
DIVF	Divide Floating Point	89	4	$@M_1(R_1),R_2$
FLTF	Convert Fixed to Float	82	2	$@R_1$
INTF	Convert Float to Fixed	81	2	$@R_1,R_2$
LODF	Load Floating Point Register	84	4	$@M_1(R_1),R_2$
MPYF	Multiply Floating Point	88	4	$@M_1(R_1),R_2$
NEGF	Negate Floating Point Register	80	2	
STOF	Store Floating Point Register	8A	4	$@M_1(R_1)$
SUBF	Subtract Floating Point	85	4	$@M_1(R_1),R_2$

SYSTEM INSTRUCTIONS

CONTROL INSTRUCTIONS

CTB	Clear Tie-Breaker Register	12	2	I_1
TST	Test and Set Tie-Breaker Register	11	2	I_1
BCM	Branch to Control Memory	EF	2	R_1,I_2
RAR	Read Any Register	FE	4	$I_1(R_1),@R_2$
WAR	Write Any Register	FE	4	$I_1(R_1),@R_2$
RRO	Read Register – Option Register	FD	4	$I_1(R_1),@R_2$
WRO	Write Register – Option Register	FD	4	$I_1(R_1),@R_2$
SAR	Save All Registers	FF	4	$M_1(R_1),I_2$ or $M_1(R_1),@R_2$
RSAR	Restore All Registers	FF	4	$M_1(R_1),I_2$ or $M_1(R_1),@R_2$

SBA	Set Busy/Active Register	10	2	I_1,I_2 or $@R_1,I_2$
RBA	Reset Busy/Active Register	10	2	I_1,I_2 or $@R_1,I_2$
SCN	Set Control Register	14	2	I_1,I_2 or $@R_1,I_2$
RCN	Reset Control Register	14	2	I_1,I_2 or $@R_1,I_2$
SPM	Set Privileged Mode Register	15	2	I_1,I_2 or $@R_1,I_2$
RPM	Reset Privileged Mode Register	15	2	I_1,I_2 or $@R_1,I_2$
WRX	Write Extended Register	FO	2	E_1,R_2

I/O INSTRUCTIONS

DIO	Disc Input/Output	F2	2	$@R_1,R_2$
INP	Input from I/O Register	F5	2	$I_1,@R_2$
OUT	Output to I/O Register	F6	2	$I_1,@R_2$
RDC	Communications Input/Output	F3	2	
WRC	Communications Output	F4	2	R_1,R_2
SIO	System Input/Output	F1	2	$@R_1,R_2$

EXTENDED MNEMONIC CODES

The assembler provides extended mnemonic codes which allow unconditional skips, and conditional skips and branches to be written in a symbolic form that is easier to use than the machine-oriented forms specified for skips and branches. There are two reasons why these extended instructions are easier to use than the standard instructions:

1. Extended mnemonic codes for skip instructions do not have an F or B, as in SRPF, to denote the direction of the skip. Instead, the assembler determines the direction by the memory address or immediate value in the operand, for example: SRP THERE or S -8.

2. Extended mnemonic codes for branch and skip instructions that involve condition register testing specify the condition in the mnemonic, such as SOV for skip if overflow. The standard machine instruction names the direction and the bit status in the mnemonic, and the actual bit number tested in the operand. Thus, the extended instruction SOV 4 is the same as the standard instruction SCTF 4,0.

An extended mnemonic code does not correspond to one specific function code in the repertoire of machine instructions.

The extended instructions and the standard machine instruction(s) they replace are presented in the following tables:

- Address-Coded Skips
- After Arithmetic Instructions
- After Compare Instructions-Arithmetic Test
- After Compare Instructions-Logical Test
- After Decimal Instructions
- After PAKX Instruction
- After TBIT Instruction
- Condition Register Test

Just as for the standard instructions, indirect addressing and indexing are always optional for the extended instructions.

ADDRESS CODED SKIPS

Extended Code		Machine Instruction		Meaning
S	M_1 or I_1	SF	I_1	Skip forward or backward
		SB	I_1	
SRZ	M_1,R_2 or I_1,R_2	SRZF	I_1,R_2	Skip if reg. is zero, forward
		SRZB	I_1,R_2	or backward
SRN	M_1,R_2 or I_1,R_2	SRNF	I_1,R_2	Skip if reg. is non-zero for-
		SRNB	I_1,R_2	ward or backward
SRP	M_1,R_2 or I_1,R_2	SRPF	I_1,R_2	Skip if reg. is plus, forward
		SRPB	I_1,R_2	or backward
SRM	M_1,R_2 or I_1,R_2	SRMF	I_1,R_2	Skip if reg. is minus, forward
		SRMB	I_1,R_2	or backward

For S, the I_1 value = -255 through +255; for all other extended mnemonics in this category, I_1 = -15 through +15.

For SF and SB the I_1 value = 0-255; for all other regular instructions in this category I_1 = 0-15.

AFTER ARITHMETIC INSTRUCTIONS

BOV	@$M_1(R_1)$	BCT	@$M_1(R_1)$,0	Branch if overflow
BNV	@$M_1(R_1)$	BCF	@$M_1(R_1)$,0	Branch if no overflow
BCY	@$M_1(R_1)$	BCT	@$M_1(R_1)$,3	Branch if carry
BNC	@$M_1(R_1)$	BCF	@$M_1(R_1)$,3	Branch if no carry
SOV	M_1 or I_1	SCTF	I_1,0	Skip if overflow
		SCTB	I_1,0	
SNV	M_1 or I_1	SCFF	I_1,0	Skip if no overflow
		SCFB	I_1,0	
SCY	M_1 or I_1	SCTF	I_1,3	Skip if carry
		SCTB	I_1,3	
SNC	M_1 or I_1	SCFF	I_1,3	Skip if no carry
		SCFB	I_1,3	

I_1 = -15 through +15 for the extended instructions.

I_1 = 0-15 for the regular instructions.

AFTER COMPARE INSTRUCTIONS – ARITHMETIC TEST

The arithmetic test tests the result of a signed arithmetic compare between operand 1 and operand 2. In the following table, 1 and 2 under Meaning refer to the signed values of operands 1 and 2.

Extended Code		Machine Instruction		Meaning
BGT	@$M_1(R_1)$	BCT	@$M_1(R_1)$,1	Branch if 1<GT>2
BLT	@$M_1(R_1)$	BCT	@$M_1(R_1)$,2	Branch if 1<LT>2
BGE	@$M_1(R_1)$	BCF	@$M_1(R_1)$,2	Branch if 1<GE>2

Extended Code		Machine Instruction		Meaning
BLE	@$M_1(R_1)$	BCF	@$M_1(R_1)$,1	Branch if 1<LE>2
BEQ	@$M_1(R_1)$	BCT	@$M_1(R_1)$,3	Branch if 1<EQ>2
BNE	@$M_1(R_1)$	BCF	@$M_1(R_1)$,3	Branch if 1<NE>2
SGT	M_1 or I_1	SCTF SCTB	I_1,1 I_1,1	Skip if 1<GT>2
SLT	M_1 or I_1	SCTF SCTB	I_1,2 I_1,2	Skip if 1<LT>2
SGE	M_1 or I_1	SCFF SCFB	I_1,2 I_1,2	Skip if 1<GE>2
SLE	M_1 or I_1	SCFF SCFB	I_1,1 I_1,1	Skip if 1<LE>2
SEQ	M_1 or I_1	SCTF SCTB	I_1,3 I_1,3	Skip if 1<EQ>2
SNE	M_1 or I_1	SCFF SCFB	I_1,3 I_1,3	Skip if 1<NE>2

I_1 = -15 through +15 for extended instructions.

I_1 = 0-15 for regular instructions.

AFTER COMPARE INSTRUCTIONS – LOGICAL TEST

The logical test tests the results of an unsigned arithmetic (logical) compare between operand 1 and operand 2. In the following table, 1 and 2 under Meaning refer to the unsigned values of operands 1 and 2. COMPX and all variations of the CBY instruction always yield a logical result.

Extended Code		Machine Instruction		Meaning
BLGT	@$M_1(R_1)$	BCT	@$M_1(R_1)$,5	Branch if 1<GT>2
BLLT	@$M_1(R_1)$	BCT	@$M_1(R_1)$,6	Branch if 1<LT>2
BLGE	@$M_1(R_1)$	BCF	@$M_1(R_1)$,6	Branch if 1<GE>2
BLLE	@$M_1(R_1)$	BCF	@$M_1(R_1)$,5	Branch if 1<LE>2
BLEQ	@$M_1(R_1)$	BCT	@$M_1(R_1)$,7	Branch if 1<EQ>2
BLNE	@$M_1(R_1)$	BCF	@$M_1(R_1)$,7	Branch if 1<NE>2
SLGT	M_1 or I_1	SCTF SCTB	I_1,5 I_1,5	Skip if 1<GT>2
SLLT	M_1 or I_1	SCTF SCTB	I_1,6 I_1,6	Skip if 1<LT>2
SLGE	M_1 or I_1	SCFF SCFB	I_1,6 I_1,6	Skip if 1<GE>2
SLLE	M_1 or I_1	SCFF SCFB	I_1,5 I_1,5	Skip if 1<LE>2

AFTER COMPARE INSTRUCTIONS – LOGICAL TEST (Continued)

Extended Code		Machine Instruction		Meaning
SLEQ	M_1 or I_1	SCTF	I_1,7	Skip if 1<EQ>2
		SCTB	I_1,7	
SLNE	M_1 or I_1	SCFF	I_1,7	Skip if 1<NE>2
		SCFB	I_1,7	

I_1 = -15 through +15 for the extended instructions.

I_1 = 0-15 for the regular instructions.

AFTER DECIMAL INSTRUCTIONS

BKP	@M_1(R_1)	BCT	@M_1(R_1),1	Branch if plus
BKM	@M_1(R_1)	BCT	@M_1(R_1),2	Branch if minus
BKZ	@M_1(R_1)	BCT	@M_1(R_1),3	Branch if zero
SKP	M_1 or I_1	SCTF	I_1,1	Skip if plus
		SCTB	I_1,1	
SKM	M_1 or I_1	SCTF	I_1,2	Skip if minus
		SCTB	I_1,2	
SKZ	M_1 or I_1	SCTF	I_1,3	Skip if zero
		SCTB	I_1,3	

I_1 = -15 through +15 for the extended instructions.

I_1 = 0-15 for the regular instructions.

AFTER PAKX INSTRUCTION

BID	@M_1(R_1)	BCT	@M_1(R_1),4	Branch if invalid digit
BNI	@M_1(R_1)	BCF	@M_1(R_1),4	Branch if no invalid digit
SID	M_1 or I_1	SCTF	I_1,4	Skip if invalid digit
		SCTB	I_1,4	
SNI	M_1 or I_1	SCFF	I_1,4	Skip if no invalid digit
		SCFB	I_1,4	

I_1 = -15 through +15 for the extended instructions.

I_1 = 0-15 for the regular instructions.

AFTER TBIT INSTRUCTION

BBS	@M_1(R_1)	BCT	@M_1(R_1),0	Branch if bit is set
BBR	@M_1(R_1)	BCF	@M_1(R_1),0	Branch if bit is reset
SBS	M_1 or I_1	SCTF	I_1,0	Skip if bit is set
		SCTB	I_1,0	
SBR	M_1 or I_1	SCFF	I_1,0	Skip if bit is reset
		SCFB	I_1,0	

I_1 = -15 through +15 for the extended instructions.

I_1 = 0-15 for the regular instructions.

CONDITION REGISTER TEST

Extended Code	Machine Instruction		Meaning
SCF $\left\{\begin{matrix}M_1\\I_1\end{matrix}\right\},I_2$	SCFF	I_1,I_2	Skip if bit spec. by I_2 is off
	SCFB	I_1,I_2	
SCT $\left\{\begin{matrix}M_1\\I_1\end{matrix}\right\},I_2$	SCTF	I_1,I_2	Skip if bit spec. by I_2 is on
	SCTB	I_1,I_2	

I_1 = -15 through +15 and I_2 = 0-15 for the extended instructions.

I_1 and I_2 = 0-15 for the regular instructions.

B. EBCDIC and ASCII CODES

EBCDIC			ASCII*	EBCDIC			ASCII*
Hex Code	Graphic	Card Code	Hex Code	Hex Code	Graphic	Card Code	Hex Code
00	NUL	12-0-1-8-9	00	2F	BEL	0-7-8-9	07
01	SOH	12-1-9	01	30		12-11-0-1-8-9	
02	STX	12-2-9	02	31		1-9	
03	ETX	12-3-9	03	32	SYN	2-9	16
04	PF	12-4-9		33		3-9	
05	HT	12-5-9	09	34	PN	4-9	
06	LC	12-6-9		35	RS	5-9	1E
07	DEL	12-7-9	7F	36	UC	6-9	
08		12-8-9		37	EOT	7-9	04
09		12-1-8-9		38		8-9	
0A	SMM	12-2-8-9		39		1-8-9	
0B	VT	12-3-8-9	0B	3A		2-8-9	
0C	FF	12-4-8-9	0C	3B		3-8-9	
0D	CR	12-5-8-9	0D	3C	DC4	4-8-9	14
0E	S0	12-6-8-9	0E	3D	NAK	5-8-9	15
0F	SI	12-7-8-9	0F	3E		6-8-9	
10	DLE	12-11-1-8-9	10	3F	SUB	7-8-9	1A
11	DC1	11-1-9	11	40	SP	No punches	20
12	DC2	11-2-9	12	41		12-0-1-9	
13	DC3	11-3-9	13	42		12-0-2-9	
14	RES	11-4-9	14=DC4	43		12-0-3-9	
15	NL	11-5-9		44		12-0-4-9	
16	BS	11-6-9	08	45		12-0-5-9	
17	IL	11-7-9		46		12-0-6-9	
18	CAN	11-8-9	18	47		12-0-7-9	
19	EM	11-1-8-9	19	48		12-0-8-9	
1A	CC	11-2-8-9		49		12-1-8	
1B		11-3-8-9		4A	¢	12-2-8	
1C	IFS	11-4-8-9		4B	•	12-3-8	2E
1D	IGS	11-5-8-9	1D	4C	<	12-4-8	3C
1E	IRS	11-6-8-9		4D	(	12-5-8	28
1F	ITB(IUS)	11-7-8-9	1F	4E	+	12-6-8	2B
20	DS	11-0-1-8-9		4F	!	12-7-8	
21	SOS	0-1-9		50	&	12	26
22	FS	0-2-9	1C	51		12-11-1-9	
23		0-3-9		52		12-11-2-9	
24	BYP	0-4-9		53		12-11-3-9	
25	LF	0-5-9	0A	54		12-11-4-9	
26	EOB/ETB	0-6-9	17=ETB	55		12-11-5-9	
27	ESC/PRE	0-7-9	1B=ESC	56		12-11-6-9	
28		0-8-9		57		12-11-7-9	
29		0-1-8-9		58		12-11-8-9	
2A	SM	0-2-8-9		59		11-1-8	
2B		0-3-8-9		5A	!	11-2-8	21
2C		0-4-8-9		5B	$	11-3-8	24
2D	ENQ	0-5-8-9	05	5C	*	11-4-8	2A
2E	ACK	0-6-8-9	06	5D	)	11-5-8	29

*MRX/OS uses a seven-bit ASCII code.

EBCDIC			ASCII	EBCDIC			ASCII
Hex Code	Graphic	Card Code	Hex Code	Hex Code	Graphic	Card Code	Hex Code
5E	;	11-6-8	3B	95	n	12-11-5	6E
5F	¬	11-7-8	5E=¬	96	o	12-11-6	6F
60	-	11	2D	97	p	12-11-7	70
61	/	0-1	2F	98	q	12-11-8	71
62		11-0-2-9		99	r	12-11-9	72
63		11-0-3-9		9A		12-11-2-8	
64		11-0-4-9		9B		12-11-3-8	
65		11-0-5-9		9C		12-11-4-8	
66		11-0-6-9		9D		12-11-5-8	
67		11-0-7-9		9E		12-11-6-8	
68		11-0-8-9		9F		12-11-7-8	
69		0-1-8		A0		11-0-1-8	
6A		12-11		A1		11-0-1	
6B	,	0-3-8	2C	A2	s	11-0-2	73
6C	%	0-4-8	25	A3	t	11-0-3	74
6D	_	0-5-8	5F	A4	u	11-0-4	75
6E	>	0-6-8	3E	A5	v	11-0-5	76
6F	?	0-7-8	3F	A6	w	11-0-6	77
70		12-11-0		A7	x	11-0-7	78
71		12-11-0-1-9		A8	y	11-0-8	79
72		12-11-0-2-9		A9	z	11-0-9	7A
73		12-11-0-3-9		AA		11-0-2-8	
74		12-11-0-4-9		AB		11-0-3-8	
75		12-11-0-5-9		AC		11-0-4-8	
76		12-11-0-6-9		AD		11-0-5-8	
77		12-11-0-7-9		AE		11-0-6-8	
78		12-11-0-8-9		AF		11-0-7-8	
79		1-8		B0		12-11-0-1-8	
7A	:	2-8	3A	B1		12-11-0-1	
7B	#	3-8	23	B2		12-11-0-2	
7C	@	4-8	40	B3		12-11-0-3	
7D	'	5-8	27	B4		12-11-0-4	
7E	=	6-8	3D	B5		12-11-0-5	
7F	"	7-8	22	B6		12-11-0-6	
80		12-0-1-8		B7		12-11-0-7	
81	a	12-0-1	61	B8		12-11-0-8	
82	b	12-0-2	62	B9		12-11-0-9	
83	c	12-0-3	63	BA		12-11-0-2-8	
84	d	12-0-4	64	BB		12-11-0-3-8	
85	e	12-0-5	65	BC		12-11-0-4-8	
86	f	12-0-6	66	BD		12-11-0-5-8	
87	g	12-0-7	67	BE		12-11-0-6-8	
88	h	12-0-8	68	BF		12-11-0-7-8	
89	i	12-0-9	69	C0		12-0	
8A		12-0-2-8		C1	A	12-1	41
8B		12-0-3-8		C2	B	12-2	42
8C		12-0-4-8		C3	C	12-3	43
8D		12-0-5-8		C4	D	12-4	44
8E		12-0-6-8		C5	E	12-5	45
8F		12-0-7-8		C6	F	12-6	46
90		12-11-1-8		C7	G	12-7	47
91	j	12-11-1	6A	C8	H	12-8	48
92	k	12-11-2	6B	C9	I	12-9	49
93	l	12-11-3	6C	CA		12-0-2-8-9	
94	m	12-11-4	6D	CB		12-0-3-8-9	

EBCDIC			ASCII	EBCDIC			ASCII
Hex Code	Graphic	Card Code	Hex Code	Hex Code	Graphic	Card Code	Hex Code
CC		12-0-4-8-9		E6	W	0-6	57
CB		12-0-5-8-9		E7	X	0-7	58
CE		12-0-6-8-9		E8	Y	0-8	59
CF		12-0-7-8-9		E9	Z	0-9	5A
D0		11-0		EA		11-0-2-8-9	
D1	J	11-1	4A	EB		11-0-3-8-9	
D2	K	11-2	4B	EC		11-0-4-8-9	
D3	L	11-3	4C	ED		11-0-5-8-9	
D4	M	11-4	4D	EE		11-0-6-8-9	
D5	N	11-5	4E	EF		11-0-7-8-9	
D6	O	11-6	4F	F0	0	0	30
D7	P	11-7	50	F1	1	1	31
D8	Q	11-8	51	F2	2	2	32
D9	R	11-9	52	F3	3	3	33
DA		12-11-2-8-9		F4	4	4	34
DB		12-11-3-8-9		F5	5	5	35
DC		12-11-4-8-9		F6	6	6	36
DD		12-11-5-8-9		F7	7	7	37
DE		12-11-6-8-9		F8	8	8	38
DF		12-11-7-8-9		F9	9	9	39
E0		0-2-8		FA		12-11-0-2-8-9	
E1		11-0-1-9		FB		12-11-0-3-8-9	
E2	S	0-2	53	FC		12-11-0-4-8-9	
E3	T	0-3	54	FD		12-11-0-5-8-9	
E4	U	0-4	55	FE		12-11-0-6-8-9	
E5	V	0-5	56	FF		12-11-0-7-8-9	

SYMBOLS UNIQUE TO ASCII	
Graphic	Hex Code
DC4	14
¬	5E
[	5B
\	5C
]	5D
–	5F
`	60
{	7B
|	7C
}	7D
~	7E

C. HEXADECIMAL ARITHMETIC

HEXADECIMAL-DECIMAL CONVERSION

HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.
1	1	10	16	100	256	1000	4096	10000	65536
2	2	20	32	200	512	2000	8192	20000	131072
3	3	30	48	300	768	3000	12288	30000	196608
4	4	40	64	400	1024	4000	16384	40000	262144
5	5	50	80	500	1280	5000	20480	50000	327680
6	6	60	96	600	1536	6000	24576	60000	393216
7	7	70	112	700	1792	7000	28672	70000	458752
8	8	80	128	800	2048	8000	32768	80000	524288
9	9	90	144	900	2304	9000	36864	90000	589824
A	10	A0	160	A00	2560	A000	40960	A0000	655360
B	11	B0	176	B00	2816	B000	45056	B0000	720896
C	12	C0	192	C00	3072	C000	49152	C0000	786432
D	13	D0	208	D00	3328	D000	53248	D0000	851968
E	14	E0	224	E00	3584	E000	57344	E0000	917504
F	15	F0	240	F00	3840	F000	61440	F0000	983040

MULTIPLICATION

1	1													
2	2	4												
3	3	6	9											
4	4	8	C	10										
5	5	A	F	14	19									
6	6	C	12	18	1E	24								
7	7	E	15	1C	23	2A	31							
8	8	10	18	20	28	30	38	40						
9	9	12	1B	24	2D	36	3F	48	51					
A	A	14	1E	28	32	3C	46	50	5A	64				
B	B	16	21	2C	37	42	4D	58	63	6E	79			
C	C	18	24	30	3C	48	54	60	6C	78	84	90		
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

D. 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 in the headings: 1 refers to operand 1 with direct addressing, 2 to operand 2 with direct addressing, @1 to operand 1 with indirect addressing, and @2 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	1-2	@1-2	1-@2	@1-@2
10	SBA	$T_1 + 6T_2$	$T_1 + 6T_2$		
10	RBA			$T_1 + 6T_2$	$T_1 + 6T_2$
11	TST	$T_1 + 3M_2$ (Unprivileged Bit Set)	$T_1 + 4T_2$ (Unprivileged Bit Not Set)	$T_1 + 4T_2$ (Privileged Bit Set)	$T_1 + 5T_2$ (Privileged Bit Not Set)
12	CTB	$T_1 + 3T_2$ (Unprivileged)		$T_1 + 4T_2$ (Privileged)	
13	SR	$T_1 + 2T_2$			
14	SCN	$T_1 + 7T_2$ (processors 4-7)	$T_1 + 8T_2$ (processors 0-3)		
14	RCN			$T_1 + 7T_2$ (processors 4-7)	$T_1 + 8T_2$ (processors 0-3)
15	SPM	$T_1 + 3T_2$ (Unprivileged Mode)	$T_1 + 6T_2$ (Privileged Mode)		
15	RPM			$T_1 + 3T_2$ (Unprivileged Mode)	$T_1 + 6T_2$ (Privileged Mode)
20	MOVR	$T_1 + T_2$	$2T_1$	$2T_1 + T_2$	$3T_1$
21	CMPR	$T_1 + T_2$	$2T_1$	$2T_1 + 2T_2$	$3T_1 + T_2$
22	ADDR	$T_1 + T_2$	$2T_1$	$3T_1 + T_2$	$4T_1$
23	SUBR	$T_1 + T_2$	$2T_1$	$3T_1 + T_2$	$4T_1$
24	INVR	$T_1 + T_2$	$2T_1$	$2T_1 + T_2$	$3T_1$
25	ANDR	$T_1 + T_2$	$2T_1$	$3T_1 + T_2$	$4T_1$
26	EORR	$T_1 + T_2$	$2T_1$	$3T_1 + T_2$	$4T_1$
27	IORR	$T_1 + T_2$	$2T_1$	$3T_1 + T_2$	$4T_1$
28	MPYR	$2T_1+7T_2+K_1$	$3T_1+6T_2+K_1$	$5T_1+5T_2+K_1$	$6T_1+4T_2+K_1$
29	DIVR	$T_1+5T_2+K_2$	$2T_1+4T_2+K_2$	$5T_1+3T_2+K_2$	$6T_1+2T_2+K_2$
2A	CSTR	$T_1 + T_2$		$2T_1$	
2B	CLDR	$T_1 + T_2$	$2T_1$		
2C	LGSS	$T_1 + T_2$	$2T_1 + T_2$		
2D	LGSS–	$T_1 + 2T_2$	$2T_1 + 2T_2$		
2E	ROSS	$T_1 + T_2$	$2T_1 + T_2$		

Hex Code	Mnemonic	1-2	@1-2	1-@2	@1-@2
2F	ARSS–	$T_1 + 3T_2$	$2T_1 + 3T_2$		
30	LODI	$T_1 + 2T_2$		$2T_1 + T_2$	
31	CMPI	$T_1 + 2T_2$		$2T_1 + 2T_2$	
32	ADDI	$T_1 + 2T_2$		$3T_1 + T_2$	
33	SUBI	$T_1 + 2T_2$		$3T_1 + T_2$	
34	INVI	$T_1 + 2T_2$		$2T_1 + T_2$	
35	ANDI	$T_1 + 2T_2$		$3T_1 + T_2$	
36	EORI	$T_1 + 2T_2$		$3T_1 + T_2$	
37	IORI	$T_1 + 2T_2$		$3T_1 + T_2$	
38	MPYI	$2T_1+8T_2+K_1$		$5T_1+5T_2+K_1$	
39	DIVI	$T_1+6T_2+K_2$		$5T_1+3T_2+K_2$	
3A	PSTR		$T_1 + T_2$		$2T_1$
3B	SHFK	See special formula at end			
3C	LGDS	$T_1 + 5T_2$	$2T_1 + 5T_2$		
3D	LGDS–	$T_1 + 5T_2$	$2T_1 + 5T_2$		
3E	RODS	$T_1 + 5T_2$	$2T_1 + 5T_2$		
3F	ARDS–	$T_1 + 5T_2$	$2T_1 + 5T_2$		
40	SRZ+	$T_1 + 3T_2$			
41	SRZ–	$T_1 + 3T_2$			
42	SRN+	$T_1 + 3T_2$			
43	SRN–	$T_1 + 3T_2$			
44	SRP+	$T_1 + 3T_2$			
45	SRP–	$T_1 + 3T_2$			
46	SRM+	$T_1 + 3T_2$			
47	SRM–	$T_1 + 3T_2$			
48	SCR,T+	$T_1 + 3T_2$			
49	SCR,F+	$T_1 + 3T_2$			
4A	SCR,T–	$T_1 + 3T_2$			
4B	SCR,F–	$T_1 + 3T_2$			

See Note A

See Note B

$T_1 + T_2$ if skip not taken

Note A: T_1+T_2 if shift count = 0

Note B: $2T_1+T_2$ if shift count = 0

Hex Code	Mnemonic	1-2	@1-2	1-@2	@1-@2
4C	LGSS,I	$T_1 + T_2$			
4D	LGSS,I–	$T_1 + 2T_2$			
4E	ROSS,I	$T_1 + T_2$			
4F	ARSS,I–	$T_1 + 3T_2$			
50	ZADK	See special formulas at end			
51	CMPK				
52	ADDK				
53	SUBK				
54	MOVX				
55	CMPX				
56	TRNX				
57	EDTX				
58	PAKX				
59	UNPX				
5A	MOVL				
5C	LGDS,I	$T_1 + 5T_2$	$T_1 + T_2$ if shift count = 0		
5D	LGDS,I–	$T_1 + 5T_2$			
5E	RODS,I	$T_1 + 5T_2$			
5F	ARDS,I–	$T_1 + 5T_2$			
60	MOVM	$5T_1$	$6T_1$	$6T_1$	$7T_1$
61	CMPM	$5T_1 + T_2$	$6T_1 + T_2$	$6T_1 + T_2$	$7T_1 + T_2$
62	ADDM	$6T_1$	$7T_1$	$7T_1$	$8T_1$
63	SUBM	$6T_1$	$7T_1$	$7T_1$	$8T_1$
64	INVM	$5T_1$	$6T_1$	$6T_1$	$7T_1$
65	ANDM	$6T_1$	$7T_1$	$7T_1$	$8T_1$
66	EORM	$6T_1$	$7T_1$	$7T_1$	$8T_1$
67	IORM	$6T_1$	$7T_1$	$7T_1$	$8T_1$
68	MPYM	$8T_1+4T_2+K_1$	$9T_1+4T_2+K_1$	$9T_1+4T_2+K_1$	$10T_1+4T_2+K_1$
69	DIVM	$8T_1+2T_2+K_2$	$9T_1+2T_2+K_2$	$9T_1+2T_2+K_2$	$10T_1+2T_2+K_2$

Hex Code	Mnemonic	1-2	@1-2	1-@2	@1-@2
6A	MOVB	$5T_1$	$6T_1$	$6T_1$	$7T_1$
6B	CBYM	$5T_1 + T_2$	$6T_1 + T_2$	$6T_1 + T_2$	$7T_1 + T_2$
6C	TONR	$T_1 + 2T_2$	$2T_1 + 2T_2$	$3T_1 + 2T_2$	$4T_1 + 2T_2$
6D	RONR	$T_1 + 3T_2$	$3T_1 + 2T_2$	$3T_1 + 3T_2$	$5T_1 + 2T_2$
6E	TOFR	$T_1 + 2T_2$	$2T_1 + 2T_2$	$3T_1 + 2T_2$	$4T_1 + 2T_2$
6F	ROFR	$T_1 + 3T_2$	$3T_1 + 2T_2$	$3T_1 + 3T_2$	$5T_1 + 2T_2$
70	LODT	$4T_1$	$5T_1$	$6T_1$	$7T_1$
71	CMPT	$3.5T_1+T_2$ (average)	$4.5T_1+T_2$ (average)	$5T_1+T_2$ (average)	$6T_1+T_2$ (average)
72	ADDT	$4T_1 + T_2$	$5T_1 + T_2$	$8T_1$	$9T_1$
73	SUBT	$4T_1 + T_2$	$5T_1 + T_2$	$8T_1$	$9T_1$
A0	LOD	$3T_1$	$4T_1$	$4T_1$	$5T_1$
A1	CMP	$3T_1$	$4T_1$	$4T_1 + T_2$	$5T_1 + T_2$
A2	ADD	$3T_1$	$4T_1$	$5T_1$	$6T_1$
A3	SUB	$3T_1$	$4T_1$	$5T_1$	$6T_1$
A4	INV	$3T_1$	$4T_1$	$4T_1$	$5T_1$
A5	AND	$3T_1$	$4T_1$	$5T_1$	$6T_1$
A6	EOR	$3T_1$	$4T_1$	$5T_1$	$6T_1$
A7	IOR	$3T_1$	$4T_1$	$5T_1$	$6T_1$
A8	MPY	$4T_1+6T_2+K_1$	$5T_1+6T_2+K_1$	$7T_1+4T_2+K_1$	$8T_1+4T_2+K_1$
A9	DIV	$3T_1+4T_2+K_2$	$4T_1+4T_2+K_2$	$7T_1+2T_2+K_2$	$8T_1+2T_2+K_2$
AA	CVB	$4T_1+46T_2$	$5T_1+46T_2$		
AA	CVBT			$6T_1+90T_2$	$7T_1+90T_2$
AB	CVD	$6T_1 + K_3$	$7T_1 + K_3$		
AB	CVDT			$9T_1+38T_2+K_3$	$10T_1+38T_2+K_3$
B0	LODD	$2T_1 + T_2$		$3T_1$	
B1	CMPD	$2T_1 + T_2$		$3T_1 + T_2$	
B2	ADDD	$2T_1 + T_2$		$4T_1$	
B3	SUBD	$2T_1 + T_2$		$4T_1$	

Hex Code	Mnemonic	1-2	@1-2	1-@2	@1-@2	
B4	INVD	$2T_1 + T_2$		$3T_1$		
B5	ANDD	$2T_1 + T_2$		$4T_1$		
B6	EORD	$2T_1 + T_2$		$4T_1$		
B7	IORD	$2T_1 + T_2$		$4T_1$		
B8	MPYD	$3T_1+7T_2+K_1$		$6T_1+4T_2+K_1$		
B9	DIVD	$2T_1+5T_2+K_2$		$6T_1+2T_2+K_2$		
BA	SUN+	$T_1 + 2T_2$				
BB	SUN–	$T_1 + 2T_2$				
BC	SBIT	$4T_1$	$5T_1$			
BD	RBIT	$4T_1$	$5T_1$			
BE	TBIT	$3T_1 + T_2$	$4T_1 + T_2$			
BF	IBIT	$4T_1$	$5T_1$			
E0	JRZ	$2T_1$	$3T_1$	$3T_1$	$4T_1$	
E1	JRN	$2T_1$	$3T_1$	$3T_1$	$4T_1$	
E2	JOF	$2T_1$	$3T_1$			
E3	JON	$2T_1$	$3T_1$			
E4	JA1	$2T_1$	$3T_1$	$4T_1$	$5T_1$	$2T_1$ if jump not taken
E5	JA2	$2T_1$	$3T_1$	$4T_1$	$5T_1$	
E6	JS1	$2T_1$	$3T_1$	$4T_1$	$5T_1$	
E7	JS2	$2T_1$	$3T_1$	$4T_1$	$5T_1$	
E8	JCR,T	$2T_1$	$3T_1$			
E9	JCR,F	$2T_1$	$3T_1$			
EA	JSR	$2T_1 + T_2$	$3T_1 + T_2$	$3T_1$	$4T_1$	
EB	JUNR	$T_1 + T_2$	$2T_1$			
EC	JUN	$2T_1$	$3T_1$			
ED	JUMP	$2T_1$	$3T_1$			
EE	NOP	$T_1 + T_2$	$T_1 + T_2$	$T_1 + T_2$	$T_1 + T_2$	
EF	JCM	$T_1 + T_2$				
F0	RDX	$T_1 + 3T_2$				

Hex Code	Mnemonic	1-2	@1-2	1-@2	@1-@2
F0	WRX			$T_1 + 4T_2$	
F1	S10				
F2	D10				
F3	RDC				
F4	WRC	$T_1 + 2T_2$			
F5	INP	$T_1 + 5T_2$		$2T_1 + 5T_2$	
F6	OUT	$T_1 + 4T_2$		$2T_1 + 4T_2$	
F7	LODB	$3T_1$	$4T_1$	$4T_1$	$5T_1$
F8	STOB	$3T_1$	$4T_1$	$4T_1$	$5T_1$
F9	CBY	$3T_1 + T_2$	$4T_1 + T_2$	$4T_1 + T_2$	$5T_1 + T_2$
FA	ST0	$3T_1$	$4T_1$	$4T_1$	$5T_1$
FB	STOT	$4T_1$	$5T_1$	$6T_1$	$7T_1$
FD	RRO	$3T_1 + 3T_2$		$4T_1 + 3T_2$	
FD	WRO		$3T_1 + 3T_2$		$4T_1 + 2T_2$
FE	RAR	$2T_1 + 3T_2$		$3T_1 + 3T_2$	
FE	WAR		$2T_1 + 3T_2$		$3T_1 + 3T_2$
FF	SAR	$12T_1+22T_2$			
FF	RSAR		$12T_1+24T_2$		

SPECIAL FORMULAS

ZADK	L2≤L1	$(6+4L1*i_9)T_1+[11+(10+6L1-2L2)i_9]T_2$	
	L2>L1	$[6+(L2+3L1)i_9]T_1+[15+(10+L2+3L1)i_9]T_2$	
CMPK	L2≤L1	$(5+i_8+2i_9+L2+L1)T_1+(3+5i_9+3L1)T_2$	
	L2>L1	$(6+2i_9+L2+L1)T_1+(8+3L2)T_2$	
ADDK	L2≤L1	$(6+3L1*i_9)T_1+[10+(10+5L1-2L2)i_9]T_2$	
	L2>L1	$[6+(L2+2L1)i_9]T_1+[14+(10+2L1+L2)i_9]T_2$	
	if recomplement add:	$(3L1+1)T_1+(5L1+7)T_2$	
SUBK	same as ADDK		
MOVX	L2≥L1	$(5+L1)T_1+3T_2$	(word move)*
		$(6+2L1)T_1+(2+i_7)T_2$	(byte move)*
	L2<L1	$(5+L2+L1)T_1+3T_2$	
CMPX	L2≥L1	$(4+L2)T_1+(3+L2+.5L1)T_2$	(word compare)*
		$(4+2L2)T_1+(3+2L2+L1)T_2$	(byte compare)*
	L2<L1	$(4+L1)T_1+(3+L1+.5L2)T_2$	(word compare)*
		$(4+2L1)T_1+(3+2L1+L2)T_2$	(byte compare)*
TRNX		$(7+3L1)T_1+(4+L1)T_2$	
EDTX	Numeric Edit (L2 ≠ 0)	$(4+L_m+L2+L1)T_1+(13+3C_1+6C_2+C_3+2C_4+3C_5$	
		$+7.5C_6)T_2+[1.5L1+1)T_1+(5.5L1-1.5i_7+i_6+9)T_2]i_9$	
	Alpha Edit (L2 = 0)	$(4+L_m+2C_1+C_4)T_1+(5+2C_1+C_3+2C_4)T_2$	
PAKX		$4T_1+7T_2+[(3L1+i_8)T_1+(10L1+i_8-1)T_2]i_9$	
UNPX		$4T_1+7T_2+[(L1+L2*i_8)T_1+(5.5L1-1.5i_7+i_6+9)T_2]i_9$	
MOVL		$(5+L2)T_1+2T_2$	(word move)*
		$(5+2L2)T_1+2T_2$	(byte move)*
SHFK		$[3+(7L2+2i_{15}+2i_{16})i_8]T_1$	
		$+[7+(8+15L2-(5+i_{15})i_{12}+(3+2i_{15}+7i_{12})i_{14})i_8]T_2$	

LEGEND

	7300	7200
T_1 = memory reference cycle time	.9 μs 1.0 μs with ECC	1.8 μs ECC not available
T_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.

$K_1 = (i_1+i_2+i_3+i_4+2i_5)T_2$

$K_2 = [(51+i_1+i_2)T_1]\, i_0$

$K_3 = [44+2i_2+i_{17}+3i_{18}(15+16i_{19}-i_{20})]\, T_2$

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 1=0
1 if 1≠0

i_1 = 0 if 1≥0
1 if 1<0

i_2 = 0 if 2≥0
1 if 2<0

i_3 = 0 if 1 and 2 have like signs
1 if 1 and 2 have unlike signs

i_4 = number of one-bits in smaller of |1| or |2|

i_5 = 16 - bit # of MSB in smaller of |1| or |2|

i_6 = number of non-zero digits unpacked

i_7 = 0 if L1 is even
1 if L1 is odd

LEGEND (Continued)

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}+(2)

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 |2|

i_{18} = 0 if i_{17}=0
1 if i_{17}≠0

i_{19} = 0 if single precision (CVD)
1 if double precision (CVDT)

i_{20} = bit position of MSB in 2

INDEX

INSTRUCTION INDEX

BY MNEMONIC CODE

Mnemonic Code	Operation Code	Page Number
LLSR	2C	4-59
LOD	A0	4-52
LODB	F7	4-53
LODD	B0	4-53
LODF	84	4-68
LODI	30	4-53
LODT	70	4-54
LRDI	5D	4-59
LRDR	3D	4-60
LRSI	4D	4-60
LRSR	2D	4-60
MOVB	6A	4-54
MOVL	5A	4-54
MOVM	60	4-55
MOVR	20	4-55
MOVX	54	4-55
MPY	A8	4-11
MPYD	B8	4-12
MPYF	88	4-68
MPYK	5B	4-12
MPYI	38	4-12
MPYM	68	4-13
MPYR	28	4-14
NEGF	80	4-69
NOP	EE	4-41
OUT	F6	4-85
PAKX	58	4-48
PSTR	3A	4-56
RAR	FE	4-88
RBA	10	4-81
RBIT	BD	4-20
RCN	14	4-81
RDC	F3	4-87
RDX	F0	4-41
RLDI	5E	4-60
RLDR	3E	4-61
RLSI	4E	4-61
RLSR	2E	4-61
ROFR	6F	4-18
RONR	6D	4-19
RPM	15	4-82
RRO	FD	4-78
RSAR	FF	4-80
SAR	FF	4-80
SB	BB	4-30
SBA	10	4-80
SBIT	BC	4-20
SCN	14	4-81
SCFB	4B	4-31
SCFF	49	4-31
SCTB	4A	4-32
SCTF	48	4-32
SF	BA	4-31
SHFK	3B	4-62
SIO	F1	4-90
SPM	15	4-82
SR	13	4-41
SRMB	47	4-32
SRMF	46	4-32
SRNB	43	4-33
SRNF	42	4-34
SRPB	45	4-33
SRPF	44	4-33
SRZB	41	4-34
SRZF	40	4-34
STO	FA	4-56
STOB	F8	4-56
STOF	8A	4-69
STOT	FB	4-56
SUB	A3	4-14
SUBD	B3	4-14
SUBF	85	4-69
SUBI	33	4-15
SUBK	53	4-15
SUBM	63	4-16
SUBR	23	4-16
SUBT	73	4-17
TBIT	BE	4-20
TOFR	6E	4-20
TONR	6C	4-21
TRNX	56	4-50
TST	11	4-77
UNPX	59	4-49
WAR	FE	4-78
WRC	F4	4-89
WRO	FD	4-78
WRX	F0	4-82
ZADK	50	4-17

BY OPERATION CODE

Operation Code	Mnemonic Code	Page Number	Operation Code	Mnemonic Code	Page Number
10	SBA/RBA	4-79/4-81	4D	LRSI	4-60
11	TST	4-77	4E	RLSI	4-61
12	CTB	4-76	4F	ARSI	4-58
13	SR	4-41	50	ZADK	4-17
14	SCN/RCN	4-81	51	CMPK	4-37
15	SPM/RPM	4-82	52	ADDK	4-5
20	MOVR	4-55	53	SUBK	4-15
21	CMPR	4-39	54	MOVX	4-55
22	ADDR	4-57	55	CMPX	4-40
23	SUBR	4-16	56	TRMX	4-50
24	INVR	4-52	57	EDTX	4-44
25	ANDR	4-23	58	PAKX	4-48
26	EORR	4-24	59	UNPX	4-49
27	IORR	4-26	5A	MOVL	4-54
28	MPYR	4-14	5B	MPYK	4-12
29	DIVR	4-11	5C	LLDI	4-58
2A	CSTR	4-51	5D	LRDI	4-59
2B	CLDR	4-51	5E	RLDI	4-60
2C	LLSR	4-59	5F	ARDI	4-57
2D	LRSR	4-60	60	MOVM	4-55
2E	RLSR	4-61	61	CMPM	4-38
2F	ARSR	4-58	62	ADDM	4-6
30	LODI	4-53	63	SUBM	4-16
31	CMPI	4-37	64	INVM	4-52
32	ADDI	4-5	65	ANDM	4-22
33	SUBI	4-15	66	EORM	4-24
34	INVI	4-52	67	IORM	4-25
35	ANDI	4-22	68	MPYM	4-13
36	EORI	4-24	69	DIVM	4-10
37	IORI	4-25	6A	MOVB	4-54
38	MPYI	4-12	6B	CBYM	4-35
39	DIVI	4-9	6C	TONR	4-21
3A	PSTR	4-56	6D	RONR	4-19
3B	SHFK	4-62	6E	TOFR	4-20
3C	LLDR	4-59	6F	ROFR	4-18
3D	LRDR	4-60	70	LODT	4-54
3E	RLDR	4-61	71	CMPT	4-39
3F	ARDR	4-57	72	ADDT	4-7
40	SRZF	4-34	73	SUBT	4-17
41	SRZB	4-34	7C	DIVK	4-9
42	SRNF	4-34	80	NEGF	4-69
43	SRNB	4-33	81	INTF	4-67
44	SRPF	4-33	82	FLTF	4-67
45	SRPB	4-33	84	LODF	4-68
46	SRMF	4-32	85	SUBF	4-69
47	SRMB	4-32	86	ADDF	4-64
48	SCTF	4-32	87	CMPF	4-65
49	SCFF	4-31	88	MPYF	4-68
4A	SCTB	4-32	89	DIVF	4-65
4B	SCFB	4-31	8A	STOF	4-69
4C	LLSI	4-59	A0	LOD	4-52

Operation Code	Mnemonic Code	Page Number
A1	CMP	4-36
A2	ADD	4-4
A3	SUB	4-14
A4	INV	4-51
A5	AND	4-21
A6	EOR	4-23
A7	IOR	4-24
A8	MPY	4-11
A9	DIV	4-7
AA	CVB/CVBT	4-42
AB	CVD/CVDT	4-43
B0	LODD	4-53
B1	CMPD	4-36
B2	ADDD	4-4
B3	SUBD	4-14
B4	INVD	4-51
B5	ANDD	4-22
B6	EORD	4-23
B7	IORD	4-25
B8	MPYD	4-12
B9	DIVD	4-8
BA	SF	4-31
BB	SB	4-30
BC	SBIT	4-20
BD	RBIT	4-20
BE	TBIT	4-20
BF	IBIT	4-18
E0	BRZ	4-29
E1	BRN	4-29
E2	BOF	4-28
E3	BON	4-28
E4	BA1	4-27
E5	BA2	4-27
E6	BS1	4-29
E7	BS2	4-30
E8	BCT	4-28
E9	BCF	4-27
EA	BSR	4-30
EB	BR	4-29
EC	BCH	4-28
ED	B	4-26
EE	NOP	4-41
EF	BCM	4-77
F0	RDX/WRX	4-41/4-82
F1	SIO	4-90
F2	DIO	4-83
F3	RDC	4-87
F4	WRC	4-89
F5	INP	4-85
F6	OUT	4-85
F7	LODB	4-53
F8	STOB	4-56
F9	CBY	4-35
FA	STO	4-56
FB	STOT	4-56
FD	RRO/WRO	4-79
FE	RAR/WAR	4-77/4-78
FF	SAR/RSAR	4-80

COMMENTS FORM

7200 Processing Unit Reference Manual -- 2903.002

Please send us your comments, to help us produce better publications. Use the space below to qualify your responses to the following questions, if you wish, or to comment on other aspects of the publication. Please use specific page and paragraph/line references where appropriate. All comments become the property of the Memorex Corporation.

- Is the material:

	Yes	No
Easy to understand?	☐	☐
Conveniently organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐
Adequately indexed?	☐	☐

- For what purpose did you use this publication (reference, general interest, etc.)?

_____

- Please state your department's function: _____

_____

Please use the space below to describe any specific comments which you feel will help us to produce a better publication. _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

MEMOREX

First Class

Permit No. 250
Santa Clara
California 95050

Business Reply Mail

No Postage Necessary if Mailed in the United States

Postage Will Be Paid By

Memorex Corporation

Santa Clara Publications
Department 9722 – M/S 00-21
1200 Memorex Drive
Santa Clara, California 95052

Thank you for your information.....

Our goal is to provide better, more useful manuals, and your comments will help us to do so.

.....Memorex Publications