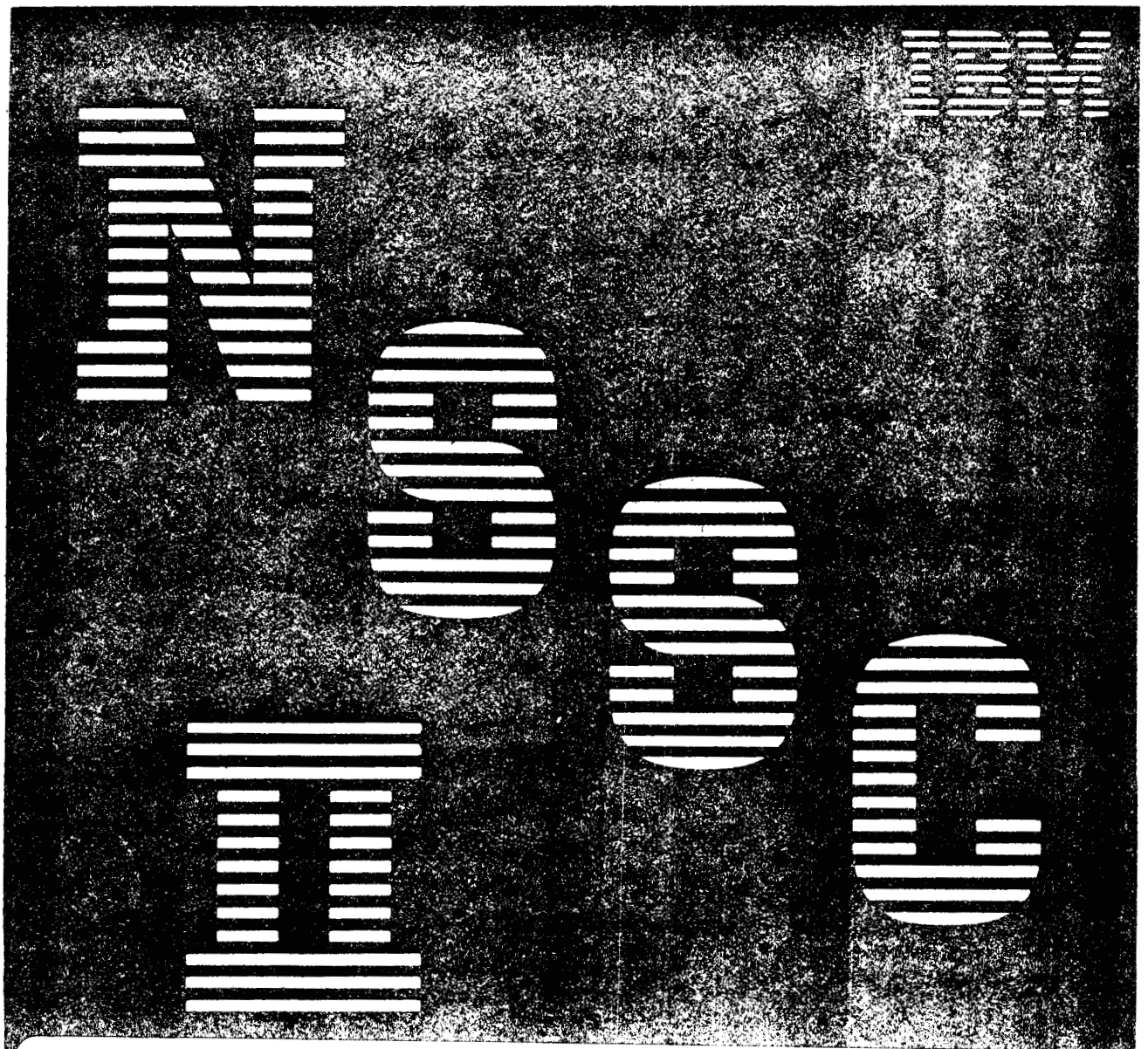
NASA STANDARD SPACECRAFT COMPUTER (NSSC-II)

PRINCIPLES OF OPERATION



	*CHK	ENGRG NOTICE	LTR	DESCRIPTION	DATE	APPROVED
				Initial Release	12/15/77	
				Errata Sheets	5/15/79	

The attached pages are replacements or additions to the NSSC-II Principles of Operation.

CONTR NO. NAS8-32808	INTERNATIONAL BUSINESS MACHINES CORP. FEDERAL SYSTEMS DIVISION GAITHERSBURG, MARYLAND
PREPARATION	TITLE NASA Standard Spacecraft Computer II (NSSC-II) Principles of Operation
DSGN CHK	
DWG CHK	
DSGN APPROVAL	SIZE / CODE IDENT NO. / DWG NO. 7935402
	SCALE / WT / SHEET

1685-0

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

PREFACE

This document is the Machine Reference Manual for the NSSC-II. It provides a description of the system structure, the arithmetic, logical, branching, status switching, I/O operations, and the interrupt and timer systems.

The NSSC-II is a 16-bit, fixed point, microprogram controlled, general purpose computer.

The NSSC-II architecture is the same as the IBM System/360 architecture. The basic NSSC-II supports 83 of the 87 instructions in the IBM System/360 Standard Instruction Set; the basic NSSC-II also supports three unique instructions which control the timers, I/O, and storage protection. The first nine sections of this document describe the basic NSSC-II.

A short precision option is available for the NSSC-II. This option consists of 53 additional instructions which deal primarily with 16-bit operands. These instructions generally execute faster than their counterparts in the basic NSSC-II instruction set, which operate on 32-bit operands. An additional instruction format is included in this option which increases execution speed and reduces main storage requirements. The short precision option is described in Section X.

A double precision fixed point option is also available for the NSSC-II. This option consists of 10 additional instructions which operate with 64-bit fixed point operands. This option is described in Section XI.

A floating point option is also available for the NSSC-II. This option consists of 22 additional instructions which are used to perform calculations on operands with a wide range of magnitude and yield results scaled to preserve precision. This option is described in Section XII.

The following NSSC-II documents contain essentially the same information as provided in the corresponding System/360 documentation referenced herein:

NSSC-II Assembler Language, IBM Number 7935401
NSSC-II Linkage Editor, IBM Number 7935413

SECTION I

NSSC-II ARCHITECTURE

1.1 NSSC-II INSTRUCTION SET

The NSSC-II is compatible with the IBM System/360 Problem State Standard Instruction Set. Problem programs written for the S/360 Standard Instruction Set will execute properly without change on the NSSC-II.

There are 171 valid NSSC-II instructions. Eighty-three of them are from the 87-member S/360 Standard Instruction Set. Omitted from the NSSC-II set are HIO, SIO, TCH, and TIO.

Three additional instructions, also described below, are:

	Mnemonic	OP Code	Format
Timer Read and Set	TMRS	A4	RS
Start I/O	SIO	A5	RS
Set Storage Key	SSK	08	RR

Note that although mnemonic SIO is used for Start I/O, and is the only NSSC-II I/O instruction, it is not the same instruction (and does not have the same op code) as the 360 SIO. Op codes A4 and A5 are unused in 360. SSK does have the same op code as 360 SSK, but performs a different function, as described below.

1.2 EXCEPTIONS

The NSSC-II is a Supervisor State compatible with the IBM System/360 with the following exceptions:

1.2.1 INPUT/OUTPUT

The I/O portion of the NSSC-II provides the means of communication between the system I/O and test support equipment (TSE) with the CPU and the main store (MS). In the 16 bit NSSC-II the I/O is implemented as a 16 bit parallel channel providing direct I/O, buffered I/O, external interrupt, and direct memory access (DMA). The 16 bit channel is SP-1 hardware compatible. There is only one I/O instruction - the SIO (Start I/O) instruction which controls direct I/O. All other I/O is device controlled.

1.2.2 TIMER

The NSSC-II has a real time clock and an interval timer, each containing both hardware and microprogrammed elements. Both are accessed by using the TMRS instruction. The S/360 interval timer in memory location 80 is not supported.

The interval timer (INTIMER) is 16 bits and is decremented every 112.64 microseconds. It has a maximum of 7.38 seconds. Underflow of the interval timer causes a timer external interrupt (which can be masked; see paragraph 3.6.5.1, External Interrupt.)

The real time clock (RTC) is 32 bits and is incremented every 112.64 microseconds. It has a maximum of 5 days, 14 hours, 23 minutes, and 5.116 seconds. It causes no interrupt on overflow.

1.2.3 STORAGE PROTECT

The size of the storage protect blocks in the NSSC-II is 1024 bytes (512 halfwords) and the operand of the SSK (Set Storage Key) instruction supports one bit for CPU and Buffered I/O protection and a second bit for DMA protection. The 4 or 5 bit protection key of S/360 is not supported. The instruction ISK (Insert Storage Key) does not exist on the NSSC-II.

1.2.4 EXECUTION TIMES

The instruction execution time is not the same for the NSCC-II and any IBM 360.

1.2.5 UNPREDICTABLE RESULTS

These occur due to addressing errors, etc., on the IBM 360 series and will not necessarily be the same unpredictable results on the NSSC-II.

1.2.6 ADDRESSING EXCEPTION

Execution of most instructions residing in the last fullword of memory will yield unpredictable results, unless memory size is 64/K bytes.

1.2.7 ADDRESSING

All effective address computation is limited to 20 bits except for the LA (Load Address) instruction, which is 24 bits. Effective addresses larger than 65,535 will be truncated to 20 bits (modulo 1,048,575) and will not cause an addressing exception unless the modolo 1,048,575 address exceeds the available main memory. If the NSSC-II has 1,048,575 bytes of main memory, an addressing exception cannot occur.

SECTION II

SYSTEM STRUCTURE

2.1 MAIN STORAGE

The NSSC-II has a maximum capacity of one mega-byte; however, the current capacity is 112K-bytes of Simplex memory or 80K-bytes of Fault Tolerant memory. The programmer should be aware of the size of the NSSC-II being programmed. The system transmits information between main storage and the CPU in units of eight bits, or a multiple of eight bits at a time. Each eight bit unit of information is called a byte, the basic building block of all formats.

Bytes may be handled separately or grouped together in fields. A halfword is a group of two consecutive bytes and is the basic building block of instructions. A word is a group of four consecutive bytes; a double word is a field consisting of two words (Figure 1). The location of any field or group of bytes is specified by the address of its leftmost byte.

The length of fields is either implied by the operation to be performed or stated explicitly as part of the instruction. When the length is implied, the information is said to have a fixed length, which can be either one, two, four, or eight bytes.

When the length of a field is not implied by the operation code, but is stated explicitly, the information is said to have variable field length. This length can be varied in one-byte increments.

Within any program format or any fixed length operand format, the bits making up the format are consecutively numbered from left to right starting with the number 0.

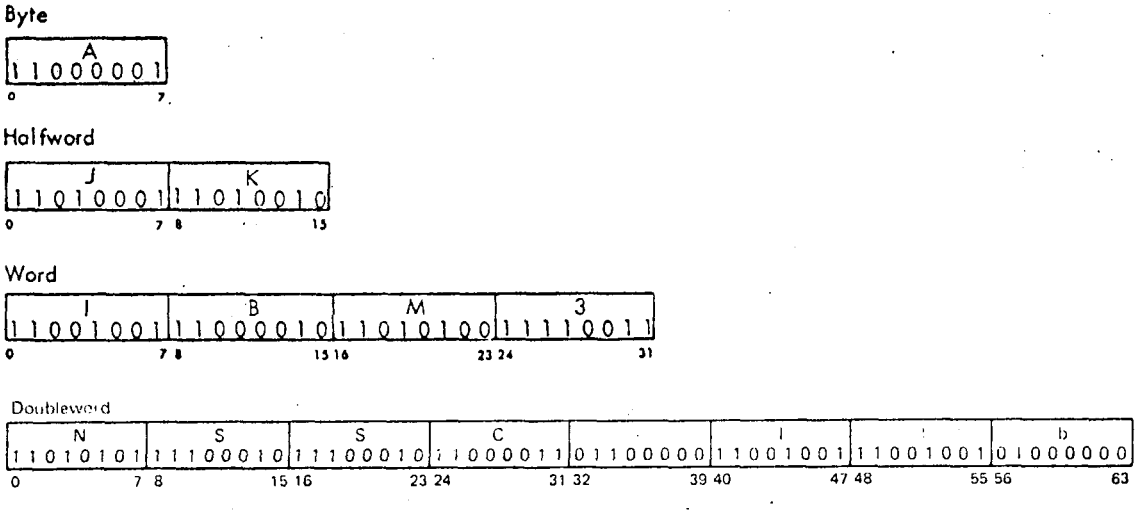


Figure 1. Sample Information Formats

2.2 ADDRESSING

Byte locations in storage are consecutively numbered starting with 0; each number is considered the address of the corresponding byte. A group of bytes in storage is addressed by the leftmost byte of the group. The number of bytes in the group is either implied or explicitly defined by the operation. The addressing arrangement uses a 20 bit binary address. This set of main storage addresses includes some locations reserved for special purposes.

Storage addressing wraps around from the maximum byte address to address 0. Variable length operands may be located partially in the last and partially in the first location of storage, and are processed without any special indication of crossing the maximum address boundary, except, perhaps, storage protection.

When only a part of the maximum storage capacity is available in a given installation, the available storage is normally contiguously addressable, starting at address 0. An addressing exception is recognized when any part of an operand is located beyond the maximum available capacity of an installation. Except for a few instructions, the addressing exception is recognized only when the data are actually used and not when the operation is completed before using the data. The addressing exception causes a program interruption.

2.3 INFORMATION PROCESSING

Fixed length fields, such as halfwords and double words, must be located in main storage on an integral boundary for that unit of information. A boundary is called integral for a unit of information when its storage address is a multiple of the length of the unit in bytes. For example, words (four bytes) must be located in storage so that their address is a multiple of the number 4. A halfword (two bytes) must have an address that is a multiple of the number 2, and double word (eight bytes) must have an address that is a multiple of the number 8.

Storage addresses are expressed in binary form. In binary, integral boundaries for halfwords, words, and double words can be specified only by the binary addresses in which one, two, or three of the low order bits, respectively, are zero (Figure 2). For example, the integral boundary for a word is a binary address in which the two low order positions are zero.

Variable length fields are not limited to integral boundaries, and may start on any byte location.

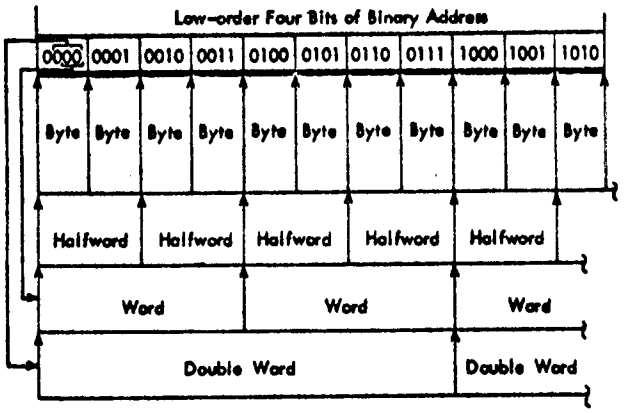


Figure 2. Integral Boundaries for Halfwords, Words, and Double Words

2.4 STORAGE PROTECTION

Memory is protected (for storing only) in blocks of 1K = 1024 bytes. There is no fetch protection. A two bit protect key is associated with each block. The first bit on protects the block against stores by the CPU; the second bit on protects the block against stores by Direct Memory Access (DMA). The key is set by the SSK instruction but cannot be read (refer to paragraph 8.5.5, SSK, and paragraph 3.7.3, DMA).

An interrupt will set the storage protect key for the first block to 01. This will allow the CPU to store in the first block and prevent DMA from storing in the first block. All other storage protect keys are unaltered by interrupts.

SECTION III

CPU

3.1 CENTRAL PROCESSING UNIT FUNCTIONS

The Central Processing Unit (CPU) (Figure 3) contains the facilities for addressing main storage, for fetching or storing information, for arithmetic and logical processing of data, for sequencing instructions in the desired order, and for initiating the communication between storage and external devices.

The system control section provides the normal CPU control that guides the CPU through the functions necessary to execute the instructions.

The CPU provides 16 general registers for fixed point operands and 4 floating point registers for floating point operands.

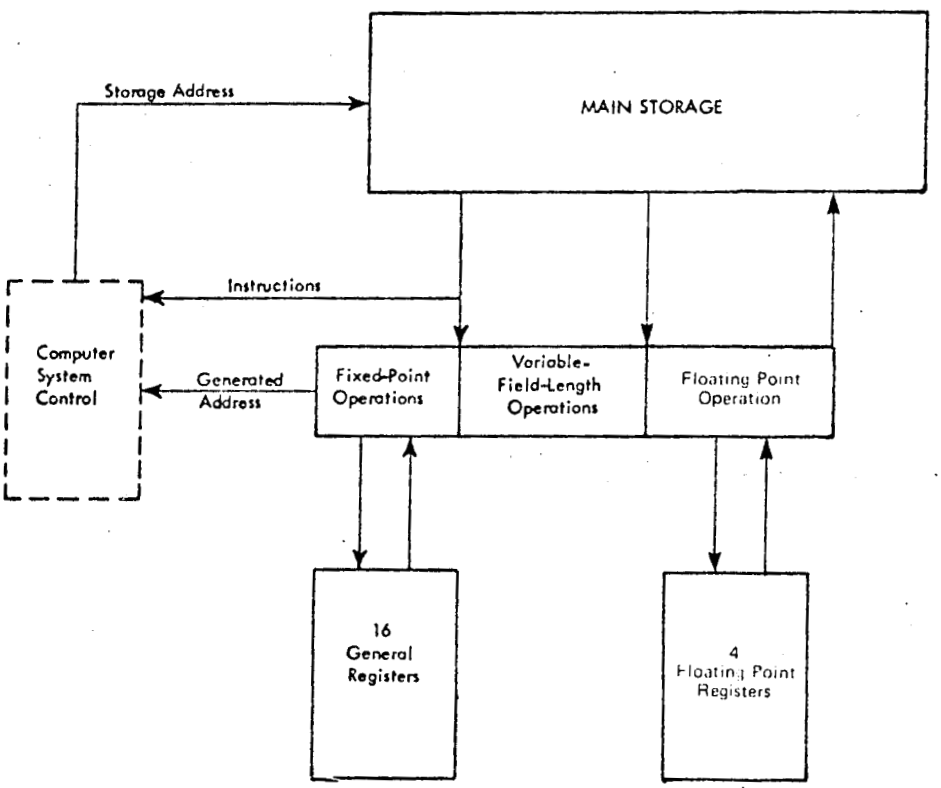


Figure 3. Basic Concept of Central Processing Unit Functions

3.2 REGISTERS

The CPU can address information in 16 general registers. The general registers can be used as index registers, in address arithmetic and indexing, and as accumulators in fixed point arithmetic and logical operations. The registers have a capacity of one word (32 bits). The general registers are identified by numbers 0-15 and are specified by a four bit R field in an instruction. Some instructions provide for addressing multiple general registers by having several R fields.

For some operations, two adjacent general registers are coupled together, providing a two word capacity. In these operations, the addressed register contains the high order operand bits and must have an even address; and the implied register, containing the low order operand bits, has the next higher address.

The CPU can address information in 4 floating point registers. The registers have a capacity of one word (32 bits). The floating point registers are identified by the numbers 0-2-4-6 and are specified by the four bit R field in an instruction. The floating point registers cannot be used as index registers.

3.3 ARITHMETIC AND LOGICAL UNIT

The arithmetic and logical unit can process binary integers of fixed length and logical information of either fixed or variable length.

3.3.1 FIXED POINT ARITHMETIC

The basic arithmetic operand is the 32 bit fixed point binary word. Sixteen bit halfword operands may be specified in most operations for improved performance or storage utilization (see Figure 4). To preserve precision, some products and all dividends are 64 bits long.

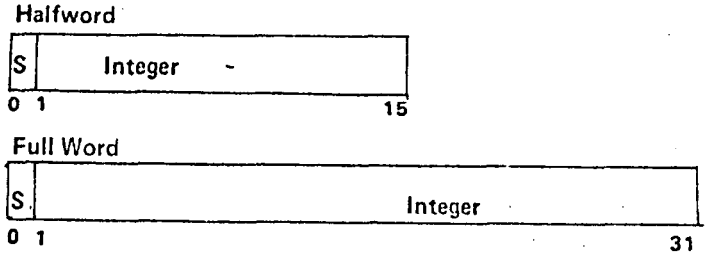


Figure 4. Fixed-Point Number Formats

Because the 32 bit word size readily accommodates a 16-bit address, fixed point arithmetic can be used both for integer operand arithmetic and for address arithmetic. This combined usage provides economy and permits the entire fixed point instruction set and several logical operations to be used in address computation. Thus, multiplication, shifting, and logical manipulation of address components are possible.

Additions, subtractions, multiplications, divisions, and comparisons are performed upon one operand in a register and another operand either in a register or from storage. Multiple precision operation is made convenient by the twos-complement notation and by recognition of the carry from one word to another. A word in one register or a double word in a pair of adjacent registers may be shifted left or right. A pair of conversion instructions -- CONVERT TO BINARY and CONVERT TO DECIMAL -- provides transition between decimal and binary radix (number base) without the use of tables. Multiple register loading and storing instructions facilitate subroutine switching.

3.4 DECIMAL NUMBERS

Decimal numbers are represented by four bit binary coded decimal digits packed two to a byte (see Figure 5). They appear in fields of variable length and are accompanied by a sign in the right-most four bits of the low order byte. Operand fields may be located on any byte boundary,

Digit	Code	Sign	Code
0	0000	+	1010
1	0001	–	1011
2	0010	+	1100
3	0011	–	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

Figure 5. Bit Codes for Digits and Signs

and may have a length up to 31 digits and sign. Operands participating in an operation may have different lengths. Packing of digits within a byte (Figure 6) and of variable length fields within storage results in efficient use of storage, in increased arithmetic performance, and in an improved rate of data transmission between storage and files.

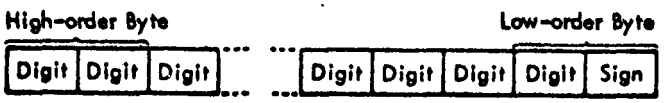


Figure 6. Packed Decimal Number Format

Decimal numbers may also appear in a zoned format as a subset of the eight bit alphanumeric character set (Figure 7). This representation is required for character set sensitive I/O devices. A zoned format number carries its sign in the left-most four bits of the low order byte .

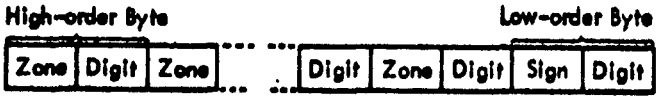


Figure 7. Zoned Decimal Number Format

Instructions are provided for packing and unpacking decimal numbers so that they may be changed from the zoned to the packed format and vice versa.

3.5 LOGICAL OPERATIONS

Logical information is handled as fixed or variable length data. It is subject to such operations as comparison, translation, bit testing, and bit setting.

When used as a fixed length operand, logical information can consist of either one, four, or eight bytes and is processed in the general registers (Figure 8).

A large portion of logical information consists of alphabetic or numeric character codes, called alphameric data, and is used for communication with character set sensitive I/O devices. This information has the variable-field-length format and can consist of up to 256 bytes (Figure 9). It is processed storage to storage, left to right, an eight bit byte at a time.

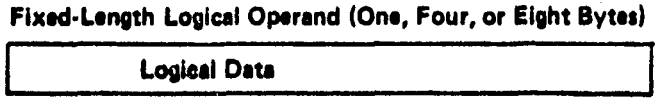


Figure 8. Fixed-Length Logical Information

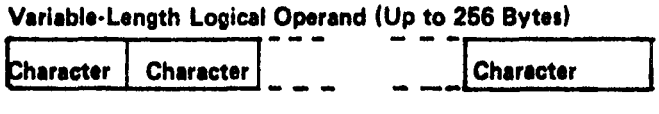


Figure 9. Variable-Length Logical Information

3.6 PROGRAM EXECUTION

The CPU program consists of instructions, index words, and control words specifying the operations to be performed. This information resides in main storage and general registers, and may be operated upon as data.

3.6.1 INSTRUCTION FORMAT

The length of an instruction format can be one, two, or three halfwords. It is related to the number of storage addresses necessary for the operation. An instruction consisting of only one halfword causes no reference to main storage. A two halfword instruction provides one storage address specification; a three halfword instruction provides two storage address specifications. All instructions must be located in storage on integral boundaries for halfwords. Figure 10 shows five basic instruction formats.

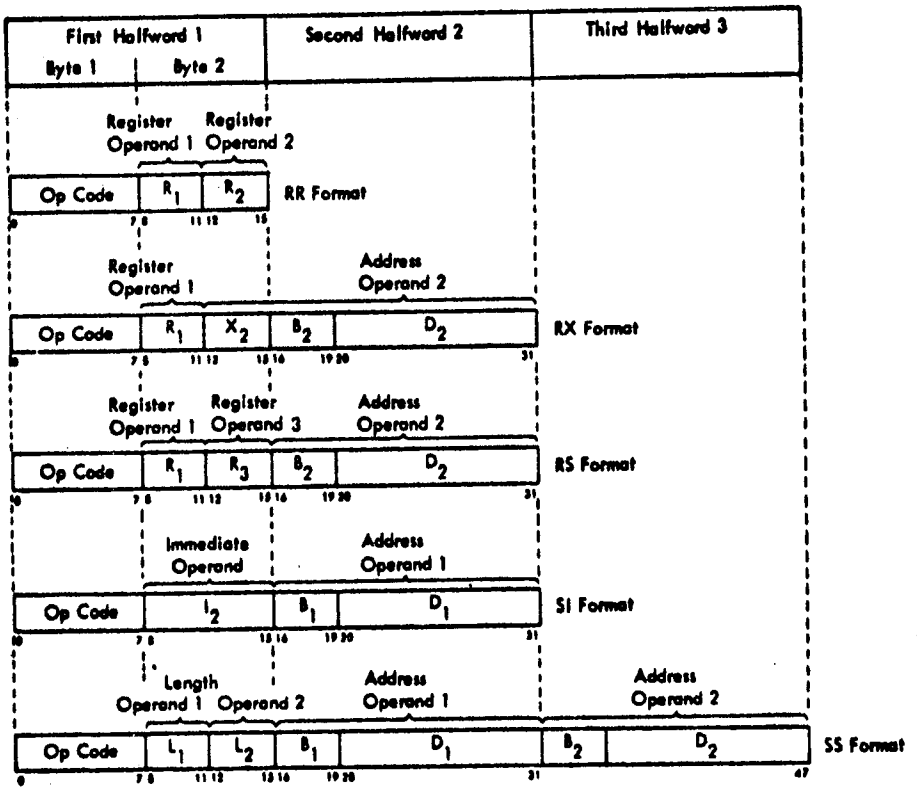
The five basic instruction formats are denoted by the format codes RR, RX, RS, SI, and SS. The format codes express, in general terms, the operation to be performed. RR denotes a register-to-register operation; RX, a register-and-indexed storage operation; RS, a register-and-storage operation; SI, a storage and immediate-operand operation; and SS, a storage-to-storage operation. An immediate operand is one contained within the instruction.

For purposes of describing the execution of instructions, operands are designated as first and second operands and, in the case of branch-on-index instructions, third operands. These names refer to the manner in which the operands participate. The operand to which a field in an instruction format applies is generally denoted by the number following the code name of the field, for example, R_1, B_1, L_2, D_2.

In each format, the first instruction halfword consists of two parts. The first byte contains the operation code (op code). The length and format of an instruction are specified by the first two bits of the operation code.

3.6.2 ADDRESS GENERATION

For addressing purposes, operands can be grouped in three classes: explicitly addressed operands in main storage; immediate operands placed as part of the instruction stream in main storage; and operands located in the general registers.



INSTRUCTION LENGTH RECORDING

BIT POSITIONS (0-1)	INSTRUCTION LENGTH	INSTRUCTION FORMAT
00	One halfword	RR
01	Two halfwords	RX
10	Two halfwords	RS or SI
11	Three halfwords	SS

NOTE: NSSC-II instructions above the standard System/360 set may not adhere to this instruction length format convention.

Figure 10. Five Basic Instruction Formats

To permit the ready relocation of program segments and to provide for the flexible specifications of input, output, and working areas, all instructions referring to main storage have been given the capacity of employing a full address.

The address used to refer to main storage is generated from the following three binary numbers.

3.6.2.1 Base Address (B)

Base Address (B) is a 20-bit number contained in a general register specified by the program in the B field of the instruction. The B field is included in every address specification. The base address can be used as a means of static relocation of programs and data. In array-type calculations, it can specify the location of an array and, in record-type processing, it can identify the record. The base address provides for addressing the entire main storage. The base address may also be used for indexing purposes.

3.6.2.2 Index (X)

Index (X) is a 20-bit number contained in a general register specified by the program in the X field of the instruction. It is included only in the address specified by the RX instruction format. The RX format instructions permit double indexing; i.e., the index can be used to provide the address of an element within an array.

3.6.2.3 Displacement (D)

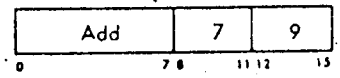
Displacement (D) is a 12-bit number contained in the instruction format and is included in every address computation. The displacement provides for relative addressing up to 4095 bytes beyond the element or base address. In array type calculations the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

In forming the address, the base address and index are treated as unsigned 20-bit positive binary integers. The displacement is similarly treated as a 12-bit positive binary integer. The three are added as 20 bit binary numbers, ignoring overflow. Since every address includes a base, the sum is always 20 bits long. The address bits are numbered 12-31 corresponding to the numbering of the base address and index bits in the general register.

The program may have zeros in the base address, index, or displacement fields. A zero is used to indicate the absence of the corresponding address component. A base or index of zero implies that a zero quantity is to be used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance. Initialization, modification, and testing of base addresses and indexes can be carried out by fixed point instructions, or by BRANCH AND LINK, BRANCH ON COUNT, or BRANCH-ON-INDEX instructions.

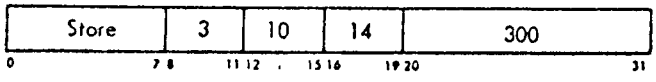
As an aid in describing the logic of the instruction format, examples of two instructions and their related instruction formats follow.

RR Format



Execution of the ADD instruction adds the contents of general register 9 to the contents of general register 7 and the sum of the addition is placed in general register 7.

RX Format



Execution of the store instruction stores the contents of general register 3 at a main storage location addressed by the sum of 300 and the low order 20 bits of general registers 14 and 10.

3.6.3 SEQUENTIAL INSTRUCTION EXECUTION

Normally, the operation of the CPU is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction address in the current PSW. The instruction address is increased by the number of bytes in the instruction fetched to address the next instruction in sequence. The instruction is then executed and the same steps are repeated using the new value of the instruction address.

Conceptually, all halfwords of an instruction are fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage word size and overlap of instruction execution with storage access may cause actual instruction fetching to be different. Thus, it is possible to modify an instruction in storage by the immediately preceding instruction. A change from sequential operation may be caused by branching, status switching, interruptions, or manual intervention.

3.6.3.1 Branching

The normal sequential execution of instructions is changed when reference is made to a subroutine, when a two-way choice is encountered, or when a segment of coding, such as a loop, is to be repeated. All these tasks can be accomplished with branching instructions. Provision is made for subroutine linkage, permitting not only the introduction of a new instruction address but also the preservation of the return address and associated information.

Decision making is generally and symmetrically provided by the BRANCH ON CONDITION instruction. This instruction inspects a two bit condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. Each of these operations can set the code in any one of four states, and the conditional branch can specify any selection of these four states as the criterion for branching. For example, the condition code reflects such conditions as nonzero, first operand high, equal, overflow, channel busy, zero, etc. Once set, the condition code remains unchanged until modified by an instruction that reflects a different condition code.

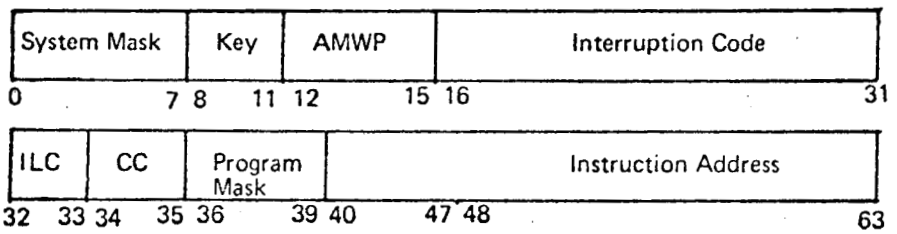
The two bits of the condition code provide for four possible condition code settings: 0, 1, 2, and 3. The specific meaning of any setting is significant only to the operation setting the condition code.

Loop control can be performed by the conditional branch when it tests the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, the instructions BRANCH ON COUNT and BRANCH ON INDEX are provided. These specialized branches provide increased performance for these tasks.

3.6.4 PROGRAM STATUS WORD

A double word, the program status word (PSW), contains the information required for proper program execution. The PSW includes the instruction address, condition code, and other fields to be discussed. In general, the PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently

being executed. The active or controlling PSW is called the "current PSW". By storing the current PSW during an interruption, the status of the CPU can be preserved for subsequent inspection. By loading a new PSW or part of a PSW, the state of the CPU can be initialized or changed. Figure 11 shows the PSW format.



0-7	System Mask
0	I/O Mask
1	Unused
2	Unused
3	Unused
4	Don't Care
5	Don't Care
6	Don't Care
7	Timer Mask
8-11	Must be 0
12	ASCII(A)
13	Machine-Check Mask (M)
14	Wait State (W)
15	Problem State (P)
16-31	Interruption Code
32-33	Instruction Length Code (ILC)
34-35	Condition Code (CC)
36-39	Program Mask
36	Fixed Point Overflow Mask
37	Unused
38	Exponent Overflow Mask
39	Significance Mask
40	If on in machine check OLD PSW indicates a parity error
41-43	Unused
44-63	Instruction Address

Figure 11. Program Status Word Format

3.6.5 INTERRUPTION

The interruption system permits the CPU to change state as a result of conditions external to the system, in input/output (I/O) units, or in the CPU itself. Five classes of interruption conditions are possible: I/O, program, supervisor call, external, and machine check.

Each class has two related PSWs called "old" and "new" in unique main storage locations (Figure 12). In all classes, an interruption involves merely storing the current PSW in its "old" position and making the PSW at the "new" position the current PSW. The "old" PSW holds all necessary status information of the system existing at the time of the interruption. If, at the conclusion of the interruption routine, there is an instruction to make the old PSW the current PSW, the system is restored to the state prior to the interruption and the interrupted routine continues.

	ADDRESS	LENGTH	PURPOSE
0	0000 0000	Double Word	Initial Program Loading PSW
8	0000 1000	Double Word	Unused
16	0001 0000	Double Word	Unused
24	0001 1000	Double Word	External old PSW
32	0010 0000	Double Word	Supervisor call old PSW
40	0010 1000	Double Word	Program old PSW
48	0011 0000	Double Word	Machine check old PSW
56	0011 1000	Double Word	Input/Output old PSW
66-67	0100 0000	Double Word	Buffered I/O Status Word
72	0100 1000	Word	Channel Address Word
76	0100 1100	Word	Unused
80	0101 0000	Word	Unused
84	0101 0100	Word	Unused
88	0101 1000	Double Word	External new PSW
96	0110 0000	Double Word	Supervisor call new PSW
104	0110 1000	Double Word	Program new PSW
112	0111 0000	Double Word	Machine check new PSW
120	0111 1000	Double Word	Input/Output new PSW

Figure 12. Permanent Storage Assignments

Interruptions are taken only when the CPU is interruptible for the interruption source. The system mask, program mask, and machine check mask bits in the PSW may be used to mask certain interruptions. When masked off, an interruption either remains pending or is ignored. The system mask may cause I/O and timer interruptions to be ignored, and the machine-check mask may cause machine hard stops. Other interruptions cannot be masked off.

An interruption always takes place after one instruction execution is finished and before a new instruction execution is started. However, the occurrence of an interruption may affect the execution of the current instruction. To permit proper programmed action following an interruption, the cause of the interruption is identified and provision is made to locate the last executed instruction.

3.6.5.1 External Interrupts

External interrupts from two sources can occur: timer interrupts (when the interval timer underflows) and interrupts from the interrupt key. These interrupts are serviced between instructions.

These two types of external interrupts may be masked off. Timer interrupts are masked by bit 7 of the system mask (PSW bit 7), as usual. A 1 enables timer interrupts; a 0 masks them and the interrupt remains pending. External interrupt key interrupts are masked by system mask bit 0 (PSW bit 0) which is also used to mask I/O interrupts (refer to paragraph 3.6.5.3). If a key interrupt is disabled, it remains pending and the channel is hung. Other bits of the system mask are ignored and need not be zero upon PSW load or in the SSM instruction.

In NSSC-II, the two types of external interrupts are not presented simultaneously if they occur simultaneously or if they are enabled simultaneously. In either case, the timer interrupt is taken and the key

interrupt remains pending. The timer interrupt has interrupt code X'0080' and the external interrupt key interrupt has interrupt code X'0040'.

3.6.5.2 Program Interrupts

The following program exceptions are monitored in NSSC-II:

Interruption Code		Program Interruption Cause
1	00000001	Operation
2	00000010	Privileged operation
3	00000011	Execute
4	00000100	Protection
5	00000101	Addressing
6	00000110	Specification
7	00000111	Data
8	00001000	Fixed-point overflow
9	00001001	Fixed-point divide
10	00001010	Unused
11	00001011	Unused
12	00001100	Exponent overflow
13	00001101	Exponent underflow
14	00001110	Significance
15	00001111	Floating-point divide

More than one cause of a program interruption may occur at once, but only one program interrupt is taken. In NSSC-II, the following priorities apply when this occurs:

Instruction Fetch:

Addressing and specification exceptions may co-occur. If the instruction address (address of the first halfword of the instruction) is out of the bounds of implemented memory, an addressing interrupt will occur. If, however, specification is bad (not on halfword boundary) and the second, third, or fourth halfword of the instruction has a bad address, a specification interrupt occurs.

Instruction Execution:

Occurrence of an operation exception (invalid op code) rules out other interruptions. However, privileged operation, protection, addressing, and specification may co-occur. Privileged instructions are dealt with below. Barring other factors, also discussed below, the priority of these interrupts is:

Addressing
Specification
Storage Protection

a) Privileged operations

There are six privileged instructions in NSSC-II. If a privileged operation exception occurs together with a memory reference exception (one of the three above) the following exception has priority and causes the interrupt:

i) SSK - privileged operation
ii) SSM - memory reference, in the above order
iii) Diagnose - privileged operation
iv) SIO - memory reference, in the above order
v) TMRS - For the interval timer, the memory reference takes priority. This is also the case for a bad RTC address. If only the second halfword of the RTC address is bad, the privileged operation exception will take precedence over the addressing exception (the RTC address need not be fullword aligned). A memory reference exception can take place even in cases of timer read only, but only for the first halfword address. (The second halfword is not read when the RTC is to be read only.)
vi) LPSW - If the new PSW address is not on a halfword boundary or is an invalid address, the addressing or specification exception will have priority over privileged operation. If the new PSW address is not on a double word boundary, the privileged operation interrupt will occur if in problem state.

b) In the instructions D and M, a memory reference exception (for the second operand) takes precedence over a specification exception caused by improper (odd) register specification for the first operand.

c) In SS instructions, memory reference exceptions for the second operand take precedence over those for the first operand.

Program interruptions can be masked off by the program mask in the PSW (bits 36-39). Each bit is associated with a program exception, as specified in the following table. When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The significance mask bit also determines the manner in which floating-point addition and subtraction are completed.

PROGRAM MASK BIT	PROGRAM EXCEPTION
36	Fixed-point overflow
37	Unused
38	Exponent underflow
39	Significance

3.6.5.3 Input/Output Interruption

An I/O interruption provides a means by which the CPU responds to conditions in the I/O units.

An I/O interruption can occur only when the mask bit associated with I/O is set to one. The status and address of the I/O unit involved are recorded in bits 16-31 of the old PSW.

3.6.6 MACHINE STATES

3.6.6.1 Running or Waiting State

In the running state, instruction fetching and execution proceed in the normal manner. The wait state is normally entered by the program to await an interruption, for example, an I/O interruption. In the wait state, no instructions are processed; the timer is updated, and the I/O and external interruptions are accepted, unless masked. Running or waiting state is determined by the setting of bit 14 in the PSW.

3.6.6.2 Masked or Interruptible State

The CPU may be interruptible or masked for I/O, timer, machine-check, and some program interruptions. When the CPU is interruptible for a class of interruptions, these interruptions are accepted. When the CPU is masked, the I/O and timer interruptions remain pending, whereas program interruptions are ignored. The interruptible states of the CPU are changed by changing the mask bits of the PSW.

3.6.6.3 Supervisor or Problem State

In the problem state, I/O and a group of control instructions are invalid. In the supervisor state, all instructions are valid. The choice of problem or supervisor state is determined by bit 15 of the PSW.

3.7 SYSTEM I/O

This section describes the interface of the NSSC-II with other system equipment.

NSSC-II has a single 16-bit I/O channel providing communication between the CPU and main memory, and the I/O devices (of which there may be 16) and the test support equipment. Direct, or program-initiated, I/O is provided only through SIO instruction. All other I/O is device or TSE initiated.

The NSSC-II channel provides three types of device initiated information transfer: (1) Buffered I/O, (2) Direct Memory Access (DMA), and (3) External Interrupts. Program initiated I/O is provided by Direct I/O. A four bit device identification code permits up to 16 system devices to be attached directly to the HTC channel.

3.7.1 DIRECT I/O

This allows, using the SIO instruction, the transfer of a 16-bit control word to a device and the transfer of a 16-bit data (half) word to or from a device. Though this is the only program-controlled I/O, and it only allows transfer of one halfword at a time to or from memory, the programmer can, using SIO, send a control word to a device telling it to initiate I/O (see SIO).

3.7.2 BUFFERED I/O

This permits devices to transfer one or more 16-bit halfwords to or from a table in main memory without knowing the location of the table. Buffered I/O occurs between instructions but does not cause an interrupt (i.e., PSWs are not swapped, etc.). A device can cause an I/O interrupt to signal that buffered I/O has occurred.

3.7.3 DIRECT MEMORY ACCESS (DMA)

DMA allows devices to send data to or from main memory without going through the CPU. It is invisible to the programmer except when an error occurs; a device may signal DMA by causing an interrupt

3.7.4 INPUT/OUTPUT OPERATIONS

The NSSC-II interface provides a 16-bit parallel channel for support of two classes of I/O equipment. These are:

1. System I/O devices and
2. Test support equipment (TSE) I/O devices.

A particular I/O device is classified based on whether it is attached directly to the NSSC-II or indirectly via the TSE. Further provision has been made to allow both program and device initiated information transfer which includes I/O commands, data words, and external I/O interrupts.

This portion of the manual describes the programmed control of I/O devices by the channel and central processing unit (CPU) including formats for the various types of I/O control information. Although

certain information, formats, etc., may be applicable to both system and TSE I/O, each type is described individually for simplicity.

Buffered I/O allows a device to transfer single or multiple words of data to/from a table in main memory without knowing the location of the table. The CPU keeps track of table word count and address incrementing. When the table is full/empty the device is notified by a signal on the ZERO COUNT line. Separate input and output tables are maintained for each buffered device code (16 codes).

The Channel Address Word (CAW) at memory location 72, points to the first location of a table that consists of sixteen (16) eight (8) BYTE entries that contain the input storage address and count, and output storage address and count of each of the (possible) 16 Buffered I/O devices (see Figure 13).

The programmer controls Buffered I/O by initialization of the I/O address and word count in the Buffered I/O Control Table.

To initialize a buffer I/O sequence the programmer must:

1. Set the CAW (loc 72) to the address of the start of the Buffered I/O table.

2. Set the device I/O word count (in the buffered I/O table) to the number of 16 bit data words to be transferred.

3. Set the device I/O word address (in the Buffered I/O table) to the memory address of the beginning of the data to be written out (or to a location for the data to be written in).

4. Start the I/O device so it will request a buffer I/O interrupt. This is usually done by giving a direct out command to the device via a SIO instruction.

DEVICE NO.	INPUT		OUTPUT	
	2 BYTES	2 BYTES	2 BYTES	2 BYTES
*0	I/O WORD COUNT	I/O WORD ADDRESS	I/O WORD COUNT	I/O WORD ADDRESS
1	"	"	"	"
2	"	"	"	"
3	"	"	"	"
4	"	"	"	"
5	"	"	"	"
6	"	"	"	"
7	"	"	"	"
8	"	"	"	"
9	"	"	"	"
15	"	"	"	"

*CAW = ADDRESS OF THIS LOCATION

Figure 13. Buffered I/O Device Table

The I/O word count is updated by one and the I/O address is updated by two in the I/O Control Table for each sixteen (16) bit [two byte] word that is transferred to or from memory by the CPU, unless the I/O word count is in TWOs complement form. If the I/O word count is in twos complement form, the I/O word count and I/O word address are not updated at the end of a Buffered I/O transfer. Therefore, the I/O address and word count start from the initial value each time the I/O device initiates a data transfer. This method of data transfer is useful for devices that send a burst of data periodically. Once a device initiates a transfer, the I/O channel is tied up until the device releases it.

If an I/O device requests a data transfer and the I/O Word Count is zero, an I/O error interrupt will be generated. The I/O Channel Code word is furnished as the interruption code in the I/O old PSW upon most I/O interrupts including error interrupt. The NSSC-II channel code word is shown in Figure 14.

It should be noted that even though the CPU hardware is interrupted to handle the buffered I/O transfers, the program is not interrupted and the time consuming save operations associated with program interrupt are not required. Buffered I/O operations are handled between instructions and do not use any register visible to the programmer.

3.7.5 BUFFERED I/O STATUS WORD

The Buffered I/O status word (loc 66-67) is set to the current Buffered I/O address during Buffered I/O operations and is cleared to zero when a Buffered I/O operation is completed successfully.

If an addressing exception, memory protect exception, or parity error occurs during Buffered I/O, an exception program interruption will be generated with the Buffered I/O status word set non zero; the contents will indicate the address of the Buffered I/O word in use at the time the error occurred.

3.7.6 SERVICE INTERRUPT

Interrupts permit a device to interrupt the normal program sequence. A single level of interrupt is provided. Programmed priorities may be implemented. In the interrupt sequence an I/O Channel Code word is sent from the device and stored as the Interruption Code in the old I/O PSW (see Figure 14).

The new I/O PSW is used as the current PSW on all I/O interrupts except Buffered I/O. A Buffered I/O interrupt is not visible to the programmer. He will never see bit 1 set in the I/O interrupt code.

A Direct Memory Access (DMA) error will cause a normal I/O interrupt, except the only bits set in the old I/O PSW interrupt code will be bits 3 or 4 indicating DMA error 1 or 2 (see Figure 14).

IOTSE BITS	USE	
0	LOGIC 0 - I/O SERVICE LOGIC 1 - TSE INTERRUPT	
1	LOGIC 1 - BUFFERED INPUT LOGIC 2 - BUFFERED OUTPUT	USED ONLY FOR BIT 0 = 0
2	LOGIC 1 - EXTERNAL INTERRUPT LOGIC 0 - BUFFERED I/O	
3	DMA ERROR NO. 1	
4	DMA ERROR NO. 2	
5	UNUSED	
8-11	TAG (DEVICE ADDRESS) OR INTERRUPT CODE	
12-15	DEVICE FUNCTION CODE (MUST NOT BE ZERO)	

Figure 14. I/O Interrupt Word

3.7.7 TSE I/O DEVICES

The TSE has a Typewriter/Paper Tape Reader or Typewriter/Magnetic Tape Reader, which are both direct I/O. All commands are sent to the TSE equipment and all data received by using the SIO instruction. The command word (see Figure 15 for TSE commands) is placed at the effective address (EA); the output data word is placed in the register designated by R1; and the data word read will be in the register designated by R3 after instruction completion. To write data to the typewriter:

1. Send a command to put the typewriter in the output mode.

2. Send each character (byte) to the typewriter by placing it right justified in R1 and sending a write typewriter command.

3. Check the condition code after each SIO instruction to ensure the I/O interface was not busy and the instruction was completed successfully.

4. The Typewriter will give a typer cycle complete interrupt (see Figure 16) after each character is complete and it is ready to receive another command.

NOTE:

It is possible to preclude the typer cycle complete interrupt by immediately generating another Direct Out command to send the next character to the typewriter. The SIO instructions may be strung together in this manner and all I/O interrupts will be locked out as the typewriter will have control of the I/O channel for the whole period. Note that this type of operation will prevent the Clock and the Timer from being updated while the channel is tied up.

The typewriter input cycle is exactly the same as the output cycle except the typewriter must be placed in the input mode and each character of input is the register designated by R3 at the completion of each SIO instruction. To read from the paper tape:

1. Send a Start Tape command.

2. When a tape character has been read and is ready to be transmitted to the CPU a Tape Data Ready I/O-TSE interrupt will be generated.

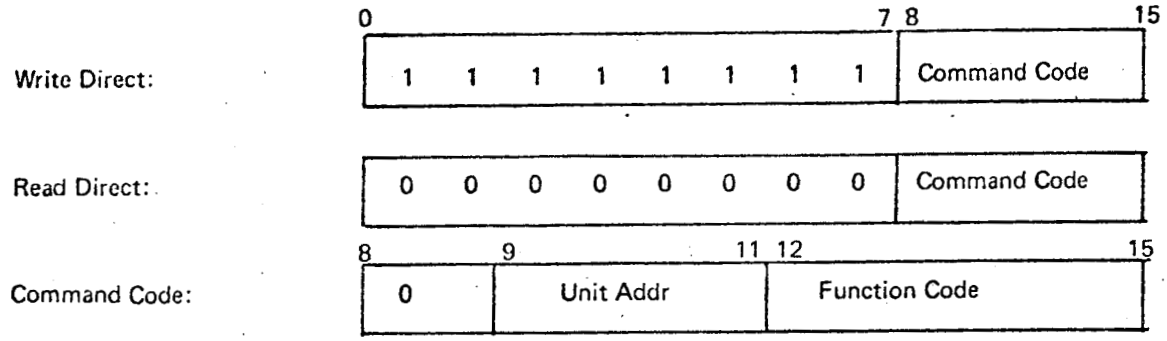
3. Read each character by sending a read tape command. Each character will be right justified in the register designated by R3 after the SIO instruction.

4. When the last desired character is read in, a stop tape command is sent to the tape reader.

NOTE:

It is possible to preclude the Tape Data Ready Interrupt by immediately generating another Direct In command to fetch the next character from the tape. The SIO instructions may be strung together in this manner and I/O interrupts will be locked out as the Tape Reader will have control of the I/O channel for the whole period. Note that this type of operation will prevent the Timer and Clock from being updated while the channel is tied up.

Direct I/O provides a means for the programmer to send a command or data word to an I/O device or request a data word from a device. Each Direct I/O instruction sends a 16-bit command word out on the System I/O channel and may send or request a data word to/from the addressed I/O device. Figure 17 shows the format of the command word. The channel is attached and relinquished for each Direct I/O instruction.



Command	Code
Typewriter Output Mode	0000 0111 0000 0000
Typewriter Input Mode	1111 1111 0110 0010
Read Typewriter	0000 0000 0110 0011
Write Typewriter	1111 1111 0110 0100
Start Tape	1111 1111 0100 0001
Stop Tape Advance	1111 1111 0100 0010
Read Tape	0000 0000 0100 0011
Read 16 right-most bits of panel address register	0000 0000 0000 0100
Read 8 left-most bits of panel address register	0000 0000 0000 0101
Read 16 left-most bits of panel data register	0000 0000 0000 0111
Read 16 right-most bits of panel data register	0000 0000 0000 0110
Display Registers	1111 1111 0000 1000

Figure 15. TSE Command Words

INTERRUPT	CODE	CAUSE
Enter Soft Stop	1000 0000 0000 0001	Depression of STOP switch
Read SPM	1000 0000 0000 0010	Depression of READ SPM switch
Write SPM	1000 0000 0000 0011	Depression of Write SPM switch
Read Main Memory	1000 0000 0000 0100	Depression of READ Memory switch
Write Main Memory	1000 0000 0000 0101	Depression of WRITE Memory switch
Exit Soft Stop	1000 0000 0000 0110	Depression of START switch
External Interrupt	1000 0000 0000 0111	Depression of external interrupt switch
Tape Load	1000 0000 0000 1000	Depression of IPL Program Load switch
PSW Restart	1000 0000 0000 1001	Depression of PSW restart switch
Attention *	1000 1000 0110 1011	Depression of attention key on typewriter
Clear Memory	1000 0000 0000 1010	Depression of clear memory switch
Tape Data Ready *	1000 0000 0100 1011	Reading a character on paper tape
Typer Cycle Complete *	1000 0000 0110 1011	Completion of typing an input or output character on the typewriter.

*These interrupts are visible to the NSSC-II program. The other interrupts in this table are intercepted and acted upon by the microprogram.

Figure 16. Tester Interrupts

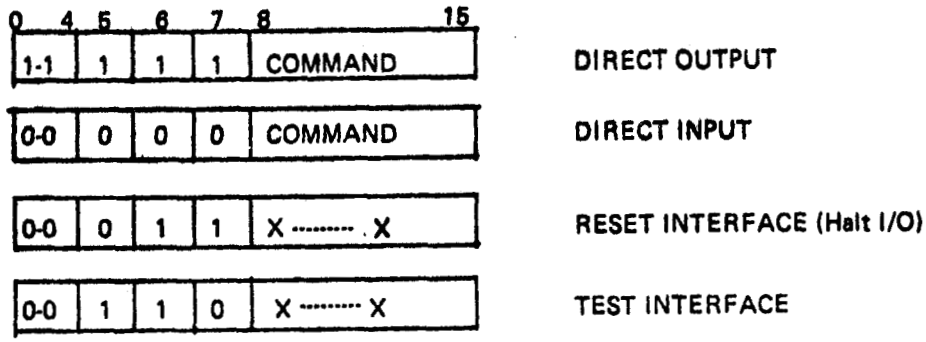


FIGURE 17. DIRECT I/O COMMAND WORD AND CPU TO I/O COMMAND WORD

The start I/O Instruction (SIO) is used to generate all Direct I/O commands.

If the I/O Interface is busy, the condition code is set to 1 without performing the I/O operation. A condition code of 0 indicates successful completion of the SIO Instruction.

The Direct I/O command word is also used for CPU to I/O commands. Figure 17 shows those commands. Reset Interface immediately halts any I/O operation and clears the I/O channel by sending the Service Acknowledge signal and holding it on for 10 microseconds minimum.

Test Interface tests for channel busy and sets the condition code (1 if busy, 0 if not busy).

Electromechanical devices such as typewriters, perforated tape readers, and punches will have a special operation under Direct I/O. Direct Out (DO) will be as follows:

The NSSC-II I/O places the command word and data word on the line normally.

The addressed device takes the command and data word and starts to perform the indicated operation (type a character, etc.).

The DO sequence is terminated and the channel freed up. (All standard so far).

Programmer option: Normally during system operation the program would perform useful work while the device is executing the command.

When the device has completed its task and is ready for the next task (such as type another character), it will generate a standard I/O interrupt to indicate device ready.

If the program had more tasks another DO would be generated and the sequence repeated.

3.8 SOFT STOP

NSSC-II normally operates in the wait and running states, handling interrupts, executing instruction, etc.; or it can operate in "soft stop" mode. In soft stop, instructions are not executed and interrupts are ignored. NSSC-II just waits for requests from the test support equipment (TSE). When the system reset button or the stop button is pressed, NSSC-II is put in the soft stop mode.

In soft stop:

a) TSE requests are enabled.

b) The real time clock is incremented, but the interval timer is not.

c) All interrupts are ignored except parity, which hangs up.

d) Buffered I/O requests are ignored.

3.9 TEST SUPPORT EQUIPMENT

The TSE allows various functions to be performed. Some of these functions can be performed only in soft stop (where all such functions are enabled) and some can be performed only outside soft stop (if the system mask bit 0 is one). External interrupt key interrupts are implemented in this way. A TSE request is signaled by a channel code word (see paragraph 3.7.4) with bit 0 on and bits 12-15 containing a function code from 1 to 11. The function codes cause the following actions. They are actually caused by console switches.

3.9.1 FUNCTION CODE

1. STOP Enter soft stop mode. Not allowed in soft stop.

2. These initiate non-architectured functions which do not change anything visible to the programmer. Allowed only in soft stop.

3. READMAIN Architecturally, a 16 bit word addressed by the 20 address switches of the test support equipment (TSE) is read from main storage. It is displayed by the TSE. Exits to soft stop. Allowed only in soft stop.

4. WRITEMAIN The 16 bit word designated by the TSE data switches is written to memory at the address given by the TSE address switches, and displayed. Exits to soft stop. Allowed only in soft stop.

5. START Soft stop is exited and control is determined by the current PSW. Allowed only in soft stop.

6. EXTINT This code is used to implement the external interrupt key. Not allowed in soft stop.

7. PROGRAM LOAD Main memory is located from paper tape and the IPL PSW (at location zero) becomes the current PSW. Soft stop is exited. Allowed only in soft stop.

   The paper tape frames are 8 bits. These are 3 frames for each 16 bit data word or address.

   Frame 1:

   0-parity
   1-sync bit: on
   2-on indicates stop tape
   3-on indicates address
   4, 5, 6, 7 } data

   Frame 2 and 3:

   0-parity
   1-sync bit: off
   2, 3, 4, 5, 6, 7 } data

   Each 16 bit word is read from the tape. If it is an address, succeeding data (non-address) words are written at that and succeeding addresses until another address is encountered or the stop tape bit is on. Upon stop tape, soft stop is exited and the PSW at location zero becomes the PSW. Paper tape read errors and any I/O interrupts cause a hang up. (Addressing, specification, and protection exceptions can occur during this operation and will cause IPL to hang up).

8. PSW RESTART Soft stop is exited and the IPL PSW is used to start. Allowed only in soft stop.

9. MEMORY CLEAR Beginning at location zero, each halfword of memory is written with all ones, read back, and compared with the word written. If they differ a hangup occurs. Then a word of all zeros is written and also compared. All storage protect keys are zeroed. Following this operation, all memory is zeroed. Only a parity exception can occur. This exits to soft stop. Allowed only in soft stop.

10. INTIO Causes an I/O interrupt with the channel code word as the interrupt code. Not allowed in soft stop.

3.9.2 SYSTEM RESET

When the system reset button is activated, the soft stop mode is entered. The interval timer is set to its maximum value (but is not decremented while in soft stop).

SECTION IV

FIXED-POINT ARITHMETIC

The fixed-point instruction set performs binary arithmetic on operands serving as addresses, index quantities, and counts, as well as fixed-point data. In general, both operands are signed and 32 bits long. Negative quantities are held in twos-complement form. One operand is always in one of the 16 general registers; the other operand may be in main storage or in a general register.

The instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, and storing, as well as for the sign control, radix conversion, and shifting of fixed-point operands. The entire instruction set is included in the standard instruction set.

The condition code is set as a result of all sign-control, add, subtract, compare, and shift operations.

4.1 DATA FORMAT

Fixed-point numbers occupy a fixed-length format consisting of a one-bit sign followed by the integer field. When held in one of the general registers, a fixed-point quantity has a 31-bit integer field and occupies all 32 bits of the register. Some multiply, divide, and shift operations use an operand consisting of 64 bits with a 63-bit integer field. These operands are located in a pair of adjacent general registers and are addressed by an even address referring to the left-most register of the pair. The sign-bit position of the rightmost register contains part of the integer. In register-to-register operations, the same register may be specified for both operand locations.

Full Word Fixed-Point Number

Halfword Fixed-Point Number

Fixed-point data in main storage occupy a 32-bit word or a 16-bit halfword, with a binary integer field of 31 or 15 bits, respectively. The conversion instructions use a 64-bit decimal field. These data must be located on integral storage boundaries for these units of information, that is, double word, fullword, or halfword operands must be addressed with three, two, or one low-order address bit(s) set to zero.

A halfword operand in main storage is extended to a fullword as the operand is fetched from storage. Subsequently, the operand participates as a fullword operand.

In all discussions of fixed-point numbers in this publication, the expression "32-bit signed integer" denotes a 31-bit integer with a sign bit, and the expression "64-bit signed integer" denotes a 63-bit integer with a sign bit.

4.2 NUMBER REPRESENTATION

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in twos-complement notation with a one in the sign bit. The twos-complement representation of a negative number may be considered the sum of the integer part of the field, taken as a positive number, and the maximum negative number. The twos-complement of a number is obtained by inverting each bit of the number and adding a one in the low-order bit position.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number, including the sign bit, are zeros. When the number is negative, all these bits, including the sign bit, are ones. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by prefixing a field in which each bit is set equal to the high-order bit of the operand.

Twos-complement notation does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit for sign.

The CPU cannot represent the complement of the maximum negative number. When an operation, such as a subtraction from zero, produces the complement of the maximum negative number, the number remains unchanged, and a fixed-point overflow exception is recognized. An overflow does not result, however, when the number is complemented and the final result is within the representable range. An example of this case is a subtraction from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

The sign bit is leftmost in a number. In an arithmetic operation, a carry out of the integer field changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order bits are shifted out of the integer field.

4.3 CONDITION CODE

The results of fixed-point sign-control, add, subtract, compare, and shift operations are used to set the condition code in the program status word (PSW). All other fixed-point operations leave this code undisturbed. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect three types of results for fixed-point arithmetic. For most operations, the states 0, 1, or 2 indicate a zero, less than zero, or greater than zero content of the result register, while the state 3 is used when the result overflows.

For a comparison, the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

For ADD LOGICAL and SUBTRACT LOGICAL, the codes 0 and 1 indicate a zero or nonzero result register content in the absence of a logical carry out of the sign position; the codes 2 and 3 indicate a zero or nonzero result register content with a logical carry out of the sign position.

CONDITION CODE SETTINGS FOR FIXED-POINT ARITHMETIC

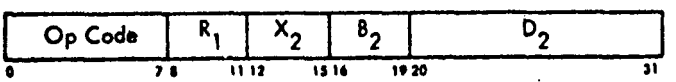
	0	1	2	3
Add H/F	zero	<zero	>zero	overflow
Add Logical	zero, no carry	not zero, no carry	zero, carry	not zero, carry
Compare H/F	equal	low	high	--
Load and Test	zero	<zero	>zero	--
Load Complement	zero	<zero	>zero	overflow
Load Negative	zero	<zero	--	--
Load Positive	zero	--	>zero	overflow
Shift Left Double	zero	<zero	>zero	overflow
Shift Left Single	zero	<zero	>zero	overflow
Shift Right Double	zero	<zero	>zero	--
Shift Right Single	zero	<zero	>zero	--
Subtract H/F	zero	<zero	>zero	overflow
Subtract Logical	--	not zero, not carry	zero, carry	not zero, carry

4.4 INSTRUCTION FORMAT

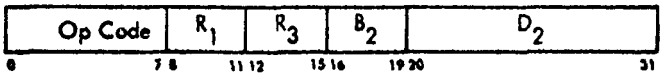
Fixed-point instructions use the following three formats:

RR Format

RX Format



RS Format



In these formats, R_1 specifies the general register containing the first operand. The second operand location, if any, is defined differently for each format.

In the RR format, the R_2 field specifies the general register containing the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general registers specified by the X_2 and B_2 fields are added to the content of the D_2 field to form an address designating the storage location of the second operand.

In the RS format, the content of the general register specified by the B_2 field is added to the content of the D_2 field. This sum designates the storage location of the second operand in LOAD MULTIPLE and STORE MULTIPLE. In the shift operations, the sum specifies the number of bits of the shift. The R_3 field specifies the address of a general register in LOAD MULTIPLE and STORE MULTIPLE and is ignored in the shift operations.

A zero in an X_2 or B_2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register both for address modification and for operand location. Address modification is always completed before operation execution.

Results replace the first operand, except for STORE and CONVERT TO DECIMAL, where the result replaces the second operand.

The contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged, except for the storing of the final result.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the NSSC-II assembly language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and R_1, R_2 the operand designation.

4.5 INSTRUCTIONS

The fixed-point arithmetic instructions and their mnemonics, formats, and operation codes are listed in the following table. The table also indicates when the condition code is set and the exceptional conditions operand designations, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE		EXCEPTIONS		CODE
Load	LR	RR				18
Load	L	RX		A,S		58
Load Halfword	LH	RX		A,S		48
Load and Test	LTR	RR	C			12
Load Complement	LCR	RR	C		IF	13
Load Positive	LPR	RR	C		IF	10
Load Negative	LNR	RR	C			11
Load Multiple	LM	RS		A,S		98
Add	AR	RR	C		IF	1A
Add	A	RX	C	A,S,	IF	5A
Add Halfword	AH	RX	C	A,S,	IF	4A
Add Logical	ALR	RR	C			1E
Add Logical	AL	RX	C	A,S		5E
Subtract	SR	RR	C		IF	1B
Subtract	S	RX	C	A,S,	IF	5B
Subtract Halfword	SH	RX	C	A,S,	IF	4B
Subtract Logical	SLR	RR	C			1F
Subtract Logical	SL	RX	C	A,S		5F
Compare	CR	RR	C			19
Compare	C	RX	C	A,S		59
Compare Halfword	CH	RX	C	A,S		49
Multiply	MR	RR		S		1C
Multiply	M	RX		A,S		5C
Multiply Halfword	MH	RX		A,S		4C
Divide	DR	RR		S,	IK	1D
Divide	D	RX		A,S,	IK	5D
Convert to Binary	CVB	RX		A,S,D,	IK	4F
Convert to Decimal	CVD	RX		P,A,S		4E
Store	ST	RX		P,A,S		50
Store Halfword	STH	RX		P,A,S		40
Store Multiple	STM	RS		P,A,S		90
Shift Left Single	SLA	RS	C		IF	8B
Shift Right Single	SRA	RS	C			8A
Shift Left Double	SLDA	RS	C	S,	IF	8F
Shift Right Double	SRDA	RS	C	S		8E

NOTES

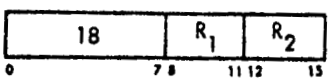
A	Addressing exception
C	Condition code is set
D	Data exception
IF	Fixed-point overflow exception
IK	Fixed-point divide exception
P	Protection exception
S	Specification exception

Programming Note

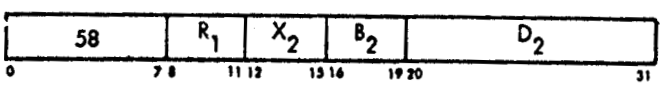
The logical comparisons, shifts, and connectives, as well as LOAD ADDRESS, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL, also may be used in fixed-point calculations.

4.5.1 LOAD

LR R_1, R_2 [RR]



L $R_1, D_2(X_2, B_2)$ [RX]



The second operand is placed in the first operand location. The second operand is not changed.

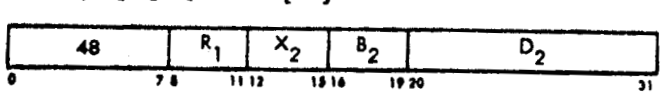
Condition Code: The code remains unchanged.

Program Interruptions:

Addressing (L only)
Specification (L only)

4.5.2 LOAD HALFWORD

LH $R_1, D_2(X_2, B_2)$ [RX]



The halfword second operand is placed in the first operand location.

The halfword second operand is expanded to a fullword by propagating the sign-bit value through the 16 high-order bit positions. Expansion occurs after the operand is obtained from storage and before insertion in the register.

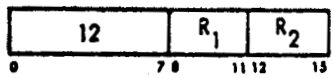
Condition Code: The code remains unchanged.

Program Interruptions:

Addressing
Specification

4.5.3 LOAD AND TEST

LTR R_1, R_2 [RR]



The second operand is placed in the first operand location, and the sign and magnitude of the second operand determine the condition code. The second operand is not changed.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

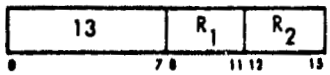
Program Interruptions: None

Programming Note

When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

4.5.4 LOAD COMPLEMENT

LCR R_1, R_2 [RR]



The twos complement of the second operand is placed in the first operand location.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

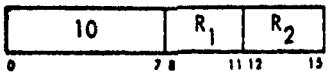
Fixed-point overflow

Programming Note

Zero remains invariant under complementation.

4.5.5 LOAD POSITIVE

LPR R_1, R_2 [RR]



The absolute value of the second operand is placed in the first operand location.

The operation includes complementation of negative numbers; positive numbers remain unchanged.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

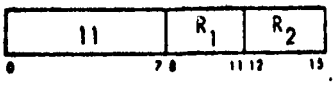
0 Result is zero
1 --
2 Result is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow

4.5.6 LOAD NEGATIVE

LNR R_1, R_2 [RR]



The twos complement of the absolute value of the second operand is placed in the first operand location. The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged with positive sign.

Resulting Condition Code:

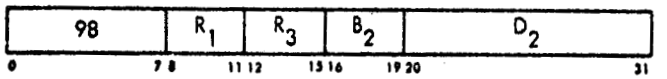
0 Result is zero
1 Result is less than zero
2 --

3 --

Program Interruptions: None.

4.5.7 LOAD MULTIPLE

LM R_1, R_3, $D_2(B_2)$ [RS]



The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is loaded from the locations designated by the second operand address.

The storage area from which the contents of the general registers are obtained starts at the location designated by the second operand address and continues through as many words as needed. The general registers are loaded in the ascending order of their addresses, starting with the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following register 15.

The second operand remains unchanged.

Condition Code: The code remains unchanged.

Program Interruptions:

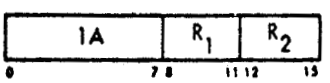
Addressing
Specification

Programming Note

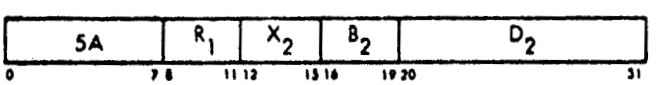
All combinations of register addresses specified by R_1 and R_3 are valid. When the register addresses are equal, only one word is transmitted. When the address specified by R_3 is less than the address specified by R_1, the register addresses wrap around from 15 to 0.

4.5.8 ADD

AR R_1, R_2 [RR]



A R_1, $D_2(X_2, B_2)$ [RX]



The second operand is added to the first operand, and the sum is placed in the first operand location.

Addition is performed by adding all 32 bits of both operands. If the carries out of the sign-bit position and the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Interruptions:

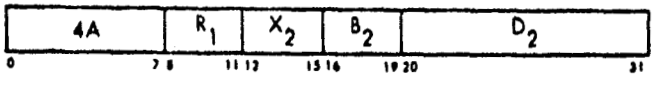
Addressing (A only)
Specification (A only)
Fixed-point overflow

Programming Note

In twos complement notation, a zero result is always positive.

4.5.9 ADD HALFWORD

AH $R_1, D_2(X_2, B_2)$ [RX]



The halfword second operand is added to the first operand and the sum is placed in the first operand location.

The halfword second operand is expanded to a fullword before the addition by propagating the sign-bit value through the 16 high-order bit positions.

Addition is performed by adding all 32 bits of both operands. If the carries out of the sign-bit position and the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

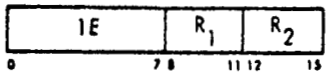
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Interruptions:

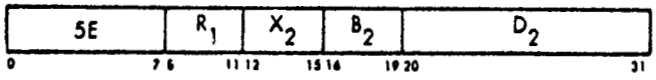
Addressing
Specification
Fixed-point overflow

4.5.10 ADD LOGICAL

ALR R_1, R_2 [RR]



AL $R_1, D_2(X_2, B_2)$ [RX]



The second operand is added to the first operand, and the sum is placed in the first operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical addition is performed by adding all 32 bits of both operands without further change to the resulting sign bit. The instruction differs from ADD in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code (PSW bit 34) is made one. In the absence of a carry, bit 34 is made zero. When the sum is zero, the rightmost bit of the condition code (PSW bit 35) is made zero. A nonzero sum is indicated by a one in bit 35.

Resulting Condition Code:

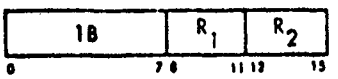
0 Sum is zero (no carry)
1 Sum is not zero (no carry)
2 Sum is zero (carry)
3 Sum is not zero (carry)

Program Interruptions:

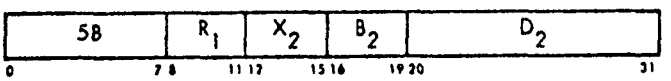
Addressing (AL only)
Specification (AL only)

4.5.11 SUBTRACT

SR R_1, R_2 [RR]



S $R_1, D_2(X_2, B_2)$ [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is performed by adding the twos complement of the second operand to the first operand. All 32 bits of both operands participate, as in ADD. If the carries out of the sign-bit position and the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Interruptions:

Addressing (S only)
Specifications (S only)
Fixed-point overflow

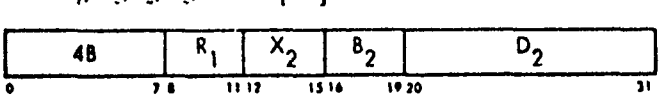
Programming Note

When the same register is speicifed as the first and second operand location, subtracting is equivalent to clearing the register.

Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

4.5.12 SUBTRACT HALFWORD

SH $R_1, D_2(X_2, B_2)$ [RX]



The halfword second operand is subtracted from the first operand, and the difference is placed in the first operand location.

The halfword second operand is expanded to a fullword before the subtraction by propagating the sign-bit value through 16 high-order bit positions.

Subtraction is performed by adding the twos complement of the expanded second operand to the first operand. All 32 bits of both operands participate, as in ADD. If the carries out of the sign-bit position and the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

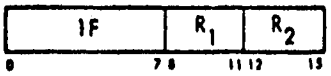
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Interruptions:

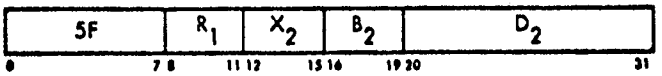
Addressing
Specification
Fixed-point overflow

4.5.13 SUBTRACT LOGICAL

SLR R_1, R_2 [RR]



SL $R_1, D_2(X_2, B_2)$ [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical subtraction is performed by adding the twos complement of the second operand to the first operand. All 32 bits of both operands participate, without further change to the resulting sign bit. The instruction differs from SUBTRACT in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code (PSW bit 34) is made one. In the absence of a carry, bit 34 is made zero. When the sum is zero, the rightmost bit of the condition code (PSW bit 35) is made zero. A nonzero sum is indicated by a one in bit 35.

Resulting Condition Code:

0 --
1 Difference is not zero (no carry)
2 Difference is zero (carry)
3 Difference is not zero (carry)

Program Interruptions:

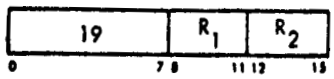
Addressing (SL only)
Specification (SL only)

Programming Note

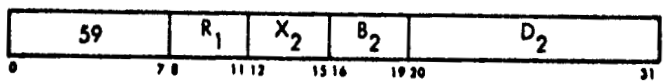
A zero difference cannot be obtained without a carry out of the sign position.

4.5.14 COMPARE

CR R_1, R_2 [RR]



C $R_1, D_2(X_2, B_2)$ [RX]



The first operand is compared with the second operand, and the result determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

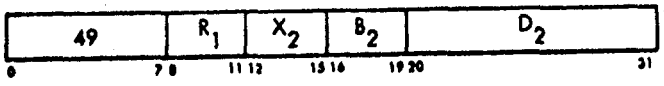
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

Addressing (C only)
Specification (C only)

4.5.15 COMPARE HALFWORD

CH $R_1, D_2(X_2, B_2)$ [RX]



The first operand is compared with the halfword second operand, and the result determines the setting of the condition code.

The halfword second operand is expanded to a fullword before the comparison by propagating the sign-bit value through the 16 high-order bit positions.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

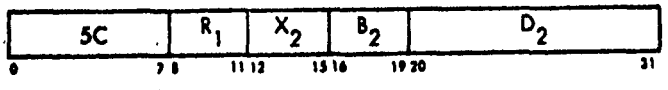
Program Interruptions:

Addressing
Specification

4.5.16 MULTIPLY

MR R_1, R_2 [RR]

M $R_1, D_2(X_2, B_2)$ [RX]



The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

Both multiplier and multiplicand are 32-bit signed integers. The product is always a 64-bit signed integer and occupies an even/odd register pair. Because the multiplicand is replaced by the product, the R_1 field of the instruction must refer to an even-numbered register. A specification exception occurs when R_1 is odd. The multiplicand is taken from the odd register of the pair. The content of an even-numbered register replaced by the product is ignored, unless the register contains the multiplier. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Interruptions:

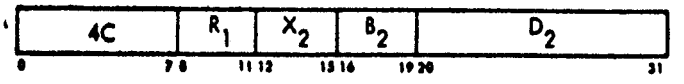
Addressing (M only)
Specification

Programming Note

The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed. Since twos-complement notation is used, the sign bit is extended right until the first significant product digit is encountered.

4.5.17 MULTIPLY HALFWORD

MH $R_1, D_2(X_2, B_2)$ [RX]



The product of the halfword multiplier (second operand) and multiplicand (first operand) replaces the multiplicand.

Both multiplicand and product are 32-bit signed integers and may be located in any general register. The halfword multiplier is expanded to a fullword before multiplication by propagating the sign-bit value through the 16 high-order bit positions. The multiplicand is replaced by the low-order part of the product. The bits to the left of the 32 low-order bits are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Interruptions:

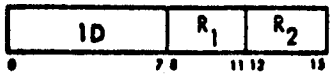
Addressing
Specification

Programming Note

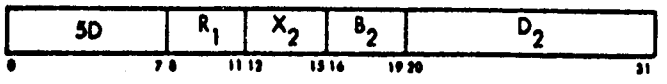
The significant part of the product usually occupies 46 bits or fewer, the exception being 47 bits when both operands are maximum negative. Since the low-order 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow.

4.5.18 DIVIDE

DR R_1, R_2 [RR]



D $R_1, D_2(X_2, B_2)$ [RX]



The dividend (first operand) is divided by the divisor (second operand) and replaced by the quotient and remainder.

The dividend is a 64-bit signed integer and occupies the even/odd pair of registers specified by the R_1 field of the instruction. A specification exception occurs when R_1 is odd. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered registers, respectively. The devisor is a 32-bit signed integer.

The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. All operands and results are treated as signed integers. When the relative magnitude of dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a fixed-point divide exception is recognized (a program interruption occurs, no division takes place, and the dividend remains unchanged in the general registers).

Condition Code: The code remains unchanged.

Program Interruptions:

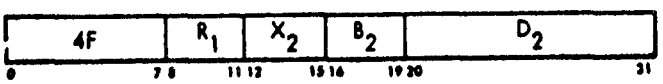
Addressing (D only)
Specification
Fixed-point divide

Programming Note

Division applies to fullword operands in storage only.

4.5.19 CONVERT TO BINARY

CVB $R_1, D_2(X_2, B_2)$ [RX]



The radix of the second operand is changed from decimal to binary, and the result is placed in the first operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The second operand has the packed decimal data format and is checked for valid sign and digit codes. Improper codes are a data exception and cause a program interruption. The decimal operand occupies a double-word storage field, which must be located on an integral boundary. The low-order four bits of the field represent the sign. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation. The packed decimal data format is described in Section V, "Decimal Arithmetic."

The result of the conversion is placed in the general register specified by R_1. The maximum number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the minimum number is -2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 low-order binary bits in the register; a fixed-point divide exception exists, and a program interruption follows. In the case of a negative second operand, the low-order part is in twos-complement notation.

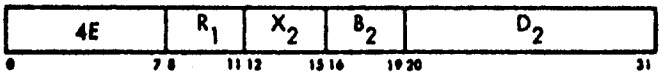
Condition Code: The code remains unchanged.

Program Interruptions:

Addressing
Specification
Data
Fixed-point divide

4.5.20 CONVERT TO DECIMAL

CVD $R_1, D_2(X_2, B_2)$ [RX]



The radix of the first operand is changed from binary to decimal, and the result is stored in the second operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The result is placed in the storage location designated by the second operand and has the packed decimal format, as described in Section V, "Decimal Arithmetic." The result occupies a double-word in storage and must be located on an integral boundary. The low-order four bits of the field represent the sign. A positive sign is encoded as 1100 or 1010; a negative sign is encoded as 1101 or 1011. The choice betweeen the two sign representations is determined by the state of PSW bit 12. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation.

The number to be converted is obtained as a 32-bit signed integer from a general register. Since 15 decimal digits are available for the decimal equivalent of 31 bits, an overflow cannot occur.

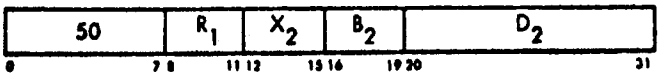
Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing
Specification

4.5.21 STORE

ST $R_1, D_2(X_2, B_2)$ [RX]



The first operand is stored at the second operand location.

The 32 bits in the general register are placed unchanged at the second operand location.

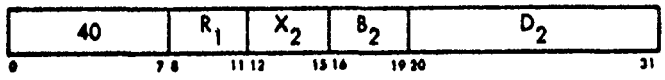
Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing
Specification

4.5.22 STORE HALFWORD

STH $R_1, D_2(X_2, B_2)$ [RX]



The first operand is stored at the halfword second operand location.

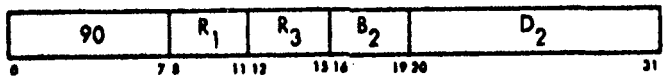
The 16 low-order bits in the general register are placed unchanged at the second operand location. The 16 high-order bits of the first operand do not participate and are not tested.

Condition Code: The code remains unchanged.
Program Interruptions:

Protection
Addressing
Specification

4.5.23 STORE MULTIPLE

STM $R_1, R_3, D_2(B_2)$ [RS]



The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is stored at the locations designated by the second operand address.

The storage area where the contents of the general registers are placed starts at the location designated by the second operand address and continues through as many words as needed. The general registers are stored in the ascending order of their addresses, starting with the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following register 15. The contents of the general registers remain unchanged.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection

Addressing
Specification

4.5.24 SHIFT LEFT SINGLE

SLA $R_1, D_2(B_2)$ [RS]

The integer part of the first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the left shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit is shifted out of position 1, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow

Programming Note

For numbers with an absolute value of less than 2^{30}, a left shift of one bit position is equivalent to multiplying the number by 2.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -2^{31}.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

4.5.25 SHIFT RIGHT SINGLE

SRA $R_1, D_2(B_2)$ [RS]

The integer part of the first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the right shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted out without inspection and are lost.

Resulting Condition Code:

0	Result is zero
1	Result is less than zero
2	Result is greater than zero
3	--

Program Interruptions:

None

Programming Note

A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position, the value of the field is that obtained by dividing the next lower number by 2. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field of a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -1.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

4.5.26 SHIFT LEFT DOUBLE

SLDA $R_1, D_2(B_2)$ [RS]

The double-length integer part of the first operand is shifted left the number of bits specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R_1 is odd.

The second operand address is not used to address data; its low-order 6-bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The operand is treated as a number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the content of the odd register participates in the shift in the same manner as the other integer bits. Zeros are supplied to the vacated positions of the registers.

If a bit unlike the sign bit is shifted out of bit position 1 of the even register, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

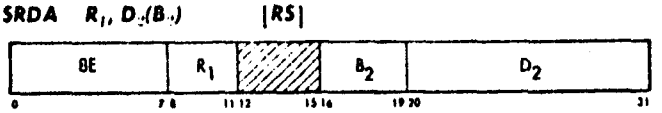
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Specification
Fixed-point overflow

4.5.27 SHIFT RIGHT DOUBLE

SRDA $R_1, D_2(B_2)$ [RS]



The double-length integer part of the first operand is shifted right the number of places specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R_1 is odd.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The operand is treated as a number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the content of the odd register participates in the shift in the same manner as the other integer bits. The low-order bits are shifted out without inspection and are lost. Bits equal to the sign are supplied to the vacated positions of the registers.

Resulting Condition Code:

0	Result is zero
1	Result is less than zero
2	Result is greater than zero
3	--

Program Interruptions:

Specification

Programming Note

A zero shift amount in the double-shift operations provides a double-length sign and magnitude test.

4.6 FIXED-POINT ARITHMETIC EXCEPTIONS

Exceptional operand designations, data, or results cause a program interruption. When a program interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in fixed-point arithmetic.

Protection

The CPU storage protection bit is set to a one (1). The operation is suppressed for a store violation. Therefore, the condition code and data in registers and storage remain unchanged.

The only exception is STORE MULTIPLE, which is terminated; the amount of data stored is unpredictable and should not be used for further computation. The operation is terminated on any fetch violation.

Addressing

An address designates an operand location outside the available storage for a particular installation. In most cases, the operation is terminated. Therefore, the result data are unpredictable and should not be used for further computation. The exceptions are STORE, STORE HALFWORD, and CONVERT TO DECIMAL, which are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the components from which an address is generated--the content of the D_2 field and the contents of the registers specified by X_2 and B_2.

Specification

A double-word operand is not located on a 64-bit boundary, a fullword operand is not located on a 32-bit boundary, a halfword operand is not located on a 16-bit boundary, or an instruction specifies an odd register address for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Data

A sign or a digit code of the decimal operand in CONVERT TO BINARY is incorrect. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Fixed-Point Overflow

The result of a sign-control, add, subtract, or shift operation overflows. The interruption occurs only when the fixed-point overflow mask bit is one. The operation is completed by placing the truncated low-order result in the register and setting the condition code to 3. The overflow bits are lost. In add-type operations the sign stored in the register is the opposite of the sign of the sum or difference. In shift operations the sign of the shifted number remains unchanged. The state of the mask bit does not affect the result.

Fixed-Point Divide

The quotient of a division exceeds the register size, including division by zero, or the result in CONVERT TO BINARY exceeds 31 bits. Division is suppressed. Therefore, data in the registers remain unchanged. The conversion is completed by recording the truncated low-order result in the register.

SECTION V

DECIMAL ARITHMETIC

5.1 DATA FORMAT

Decimal operands reside in main storage only. They occupy fields that may start at any byte address and are composed of one to 16 eight-bit bytes.

Lengths of the two operands specified in an instruction need not be the same. If necessary, they are considered to be extended with zeros to the left of the high-order digits. Results never exceed the limits set by address and length specification.

Decimal operands may be either in the packed or zoned format.

5.1.1 PACKED DECIMAL NUMBER

In the packed format, two decimal digits normally are placed adjacent in a byte, except for the rightmost byte of the field. In the rightmost byte a sign is placed to the right of decimal digit. Both digits and a sign are encoded and occupy four bits each.

5.1.2 ZONED DECIMAL NUMBER

In the zoned format the low-order four bits of a byte, the numeric, are normally occupied by a decimal digit. The four high-order bits of a byte are called the zone, except for the rightmost byte of the field, where normally the sign occupies the zone position.

In the zoned format, the digits are represented as part of an alphanumeric character set. A PACK instruction is provided to transform zoned data into packed data, and an UNPACK instruction performs the reverse transformation.

5.2 NUMBER REPRESENTATION

Numbers are represented as right-aligned true integers with a plus or minus sign.

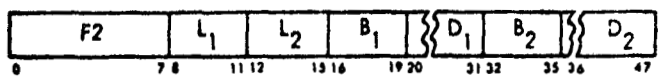
The digits 0-9 have the binary encoding 0000-1001. The codes 1010-1111 are invalid as digits. This set of codes is interpreted as sign codes, with 1010, 1100, 1110, and 1111 recognized as plus and with 1011 and 1101 recognized as minus. The codes 0000-1001 are invalid as sign codes. The zones are not tested for valid codes inasmuch as they are eliminated in changing data from the zoned to the packed format.

The sign and zone codes generated for all decimal arithmetic results differ for the extended binary-coded-decimal interchange code (EBCDIC) and the USA Standard Code for Information Interchange (USASCII-8). The choice between the two codes is determined by bit 12 of the PSW. When bit 12 is zero, the preferred EBCDIC codes are generated; these are plus, 1100; minus, 1101; and zone, 1111. When bit 12 is one, the preferred USASCII-8 codes are generated; these are plus, 1010; minus, 1011; and zone, 0101.

5.3 INSTRUCTIONS

5.3.1 PACK

PACK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The format of the second operand is changed from zoned to packed, and the result is placed in the first operand location.

The second operand is assumed to have the zoned format. All zones are ignored, except the zone over the low-order digit, which is assumed to represent a sign. The sign is placed in the right four bits of the low-order byte, and the digits are placed adjacent to the sign and to each other in the remainder of the result field. The sign and digits are moved unchanged to the first operand field and are not checked for valid codes.

The fields are processed right to left. If necessary, the second operand is extended with high-order zeros. If the first operand field is too short to contain all significant digits of the second operand field, the remaining high-order digits are ignored. Overlapping fields may occur and are processed by storing one result byte immediately after the necessary operand bytes are fetched. Except for the right-most byte of the result field, which is stored immediately upon fetching the first operand byte, two operand bytes are needed for each result byte.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing

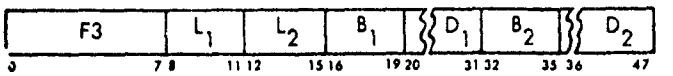
Programming Notes

The PACK instruction may be used to switch the two digits in one byte by specifying a zero in the L_1 and L_2 fields and the same address for both operands.

To remove the zones of all bytes of a field, including the low-order byte, both operands must be extended with a dummy byte in the low-order position, which subsequently is ignored in the result field.

5.3.2 UNPACK

UNPK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The format of the second operand is changed from packed to zoned, and the result is placed in the first operand location.

The digits and sign of the packed operand are placed unchanged in the first operand location, using the zoned format. Zones with coding 1111 in EBCDIC and coding 0101 in USASCII-8 are supplied for all bytes, except the low-order byte, which receives the sign of the packed operand. The operand sign and digits are not checked for valid codes.

The fields are processed right to left. The second operand is extended with high-order zero digits before unpacking, if necessary. If the first operand field is too short to contain all significant digits of the second operand, the remaining high-order digits are ignored. The first and second operand fields may overlap and are processed by storing the first result byte immediately after the rightmost operand byte is fetched; for the remaining operand bytes, two result bytes are stored immediately after one byte is fetched.

Condition Code: The code remains unchanged.

Program Interruptions:

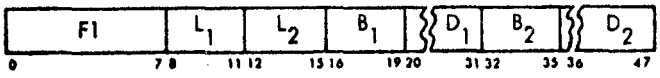
Addressing
Protection

Programming Note

A field that is to be unpacked can be destroyed by improper overlapping. If it is desired to save storage space for unpacking by overlapping the operand fields, the low-order position of the first operand must be to the right of the low-order position of the second operand by the number of bytes in the second operand minus two. If only one or two bytes are to be unpacked, the low-order positions of the two operands may coincide.

5.3.3 MOVE WITH OFFSET

MVO $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The second operand is placed to the left of and adjacent to the low-order four bits of the first operand.

The low-order four bits of the first operand are attached as low-order bits to the second operand; the second operand bits are offset by four bit positions, and the result is placed in the first operand location. The first and second operand bytes are not checked for valid codes.

The fields are processed right to left. If necessary, the second operand is extended with high-order zeros. If the first operand field is too short to contain all bytes of the second operand, the remaining information is ignored. Overlapping fields may occur and are processed by storing a result byte as soon as the necessary operand bytes are fetched.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing

Programming Note

The instruction set for decimal arithmetic includes no shift instructions since the equivalent of a shift can be obtained by programming. Programs for right or left shift and for an even or odd shift amount may be written with MOVE WITH OFFSET and the logical move instructions.

SECTION VI

LOGICAL OPERATION

A set of instructions is provided for the logical manipulation of data. Generally, the operands are treated as eight-bit bytes. In a few cases the left or right four bits of a byte are treated separately or operands are shifted a bit at a time. The operands are either in storage or in general registers. Some operands are introduced from the instruction stream.

Processing of data in storage proceeds left to right through fields which may start at any byte position. In the general registers, the processing, as a rule, involves the entire register contents.

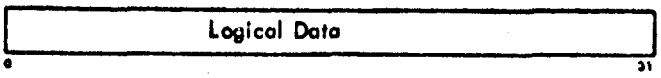
The set of logical operations includes moving, comparing, bit connecting, bit testing, translating, and shift operations. All logical operations are part of the standard instruction set.

The condition code is set as a result of all logical comparing, connecting, and testing.

6.1 DATA FORMAT

Data reside in general registers or in storage or are introduced from the instruction stream. The data size may be a single or double word, a single character, or variable in length. When two operands participate they have equal length.

6.1.1 FIXED-LENGTH LOGICAL INFORMATION

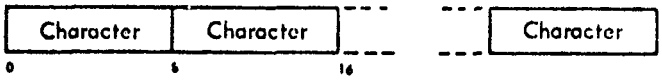


Data in general registers normally occupy all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of a register participate, leaving the remaining 24 bits unchanged. In some shift operations, 64 bits of an even/odd pair or registers participate.

The LOAD ADDRESS introduces a 24-bit address into a general register. The high-order eight bits of the register are made zero.

In storage-to-register operations, the storage data occupy either a word of 32 bits or a byte of eight bits. The word must be located on word boundaries, that is, its address must have the two low-order bits zero.

6.1.2 VARIABLE-LENGTH LOGICAL INFORMATION



In storage-to-storage operations, data have a variable field-length format, starting at any byte address and continuing for up to a total of 256 bytes. Processing is left to right.

Operations introducing data from the instruction stream into storage, as immediate data, are restricted to an eight-bit byte. Only one byte is introduced from the instruction stream, and only one byte in storage participates.

Use of general register 1 is implied in TRANSLATE AND TEST. A 24-bit address may be placed in this register during this operation. The TRANSLATE AND TEST also implies general register 2. The low-order eight bits of register 2 may be replaced by a function byte during a translate-and-test operation.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the content of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function.

A list is specified by an initial address - the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The actual address used to address the list is obtained by adding the argument to the low-order positions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. In cases where it is known that not all eight-bit argument values will occur, it may be possible to reduce the size of the list.

In a storage-to-storage operation, the operand fields may be defined in such a way that they overlap. The effect of this overlap depends upon the operation. When the operands remain unchanged, as in COMPARE or TRANSLATE AND TEST, overlapping does not affect the execution of the operation. In the case of MOVE and TRANSLATE, one operand is replaced by new data, and the execution of the operation may be affected by the amount of overlap and the manner in which data are fetched or stored. For purposes of evaluating the effect of overlapped operands, consider that data are handled one eight-bit byte at a time. All overlapping fields are considered valid.

6.2 CONDITION CODE

The results of most logical operations are used to set the condition code in the PSW. The LOAD ADDRESS, INSERT CHARACTERS, STORE CHARACTER, TRANSLATE, and the moving and shift operations leave this code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect five types of results for logical operations: FOR COMPARE LOGICAL the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

For the logical-connectives, the states 0 or 1 indicate a zero or non-zero result field.

For TEST UNDER MASK, the states 0, 1, or 3 indicate that the selected bits are all-zero, mixed zero and one, or all-one.

For TRANSLATE AND TEST, the states 0, 1, or 2 indicate an all-zero function byte, a non-zero function byte with the operand imcompletely tested, or a last function byte non-zero.

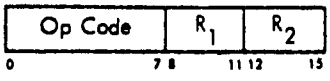
CONDITION CODE SETTING FOR LOGICAL OPERATIONS

	0	1	2	3
And	zero	not zero	--	--
Compare Logical	equal	low	high	--
Exclusive Or	zero	not zero	--	--
Or	zero	not zero	--	--
Test Under Mask	zero	mixed	--	one
Translate and Test	zero	incomplete	complete	--

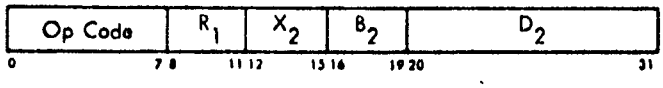
6.3 INSTRUCTION FORMAT

Logical instructions use the following five formats:

RR Format



RX Format



Results replace the first operand, except in STORE CHARACTER, where the result replaces the second operand. A variable-length result is never stored outside the field specified by the address and length.

The contents of all general registers and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the result locations and general registers 1 and 2 in TRANSLATE AND TEST.

NOTE:

In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the NSSC-II assembly language are shown with each instruction: for MOVE NUMERICS, for example, MVN is the mnemonic and D_1 (L_1B_1), D_2 (B_2) the operand designation.

6.4 INSTRUCTIONS

The logical instructions, their mnemonics, formats, and operation codes follow. The table also indicates when the condition code is set and the exceptions in operand designations, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Move	MVI	SI	P,A	92
Move	MVC	SS	P,A	D2
Move Numerics	MVN	SS	P,A	D1
Move Zones	MVZ	SS	P,A	D3
Compare Logical	CLR	RR C		15
Compare Logical	CL	RX C	A,S	55
Compare Logical	CLI	SI C	A	95
Compare Logical	CLC	SS C	A	D5
AND	NR	RR C		14
AND	N	RX C	A,S	54
AND	NI	SI C	P,A	94
AND	NC	SS C	P,A	D4
OR	OR	RR C		16
OR	O	RX C	A,S	56
OR	OI	SI C	P,A	96
OR	OC	SS C	P,A	D6
Exclusive OR	XR	RR C		17
Exclusive OR	X	RX C	A,S	57
Exclusive OR	XI	SI C	P,A	97
Exclusive OR	XC	SS C	P,A	D7
Test Under Mask	TM	SI C	A	91
Insert Character	IC	RX	A	43
Store Character	STC	RX	P,A	42
Load Address	LA	RX		41
Translate	TR	SS	P,A	DC

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Translate and Test	TRT	SS C	A	DD
Shift Left Single Logical	SLL	RS		89
Shift Right Single Logical	SRL	RS		88
Shift Left Double Logical	SLDL	RS	S	8D
Shift Right Double Logical	SRDL	RS	S	8C

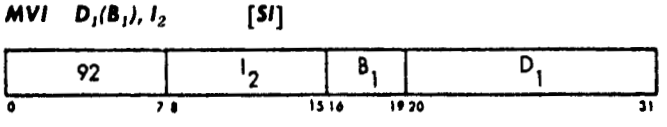
Notes:

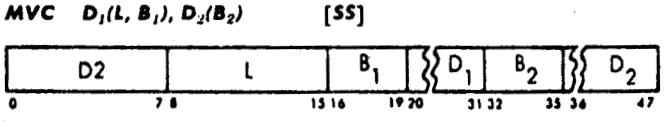
A	Addressing exception
C	Condition code is set
D	Data exception
P	Protection exception
S	Specification exception

Programming Note

The fixed-point loading and storing instructions also may be used for logical operations.

6.4.1 MOVE





The second operand is placed in the first operand location.

The SS format is used for a storage-to-storage move. The SI format introduces one 8-bit byte from the instruction stream.

In storage-to-storage movement the fields may overlap in any desired way. Movement is left to right through each field a byte at a time.

The bytes to be moved are not changed or inspected.

Condition Code: The code remains unchanged.

Program Interruptions:

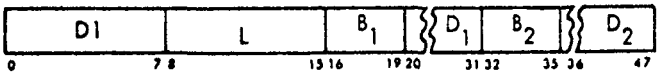
Protection
Addressing

Programming Note

It is possible to propagate one character through an entire field by having the first operand field start one character to the right of the second operand field.

6.4.2 MOVE NUMERICS

MVN $D_1(L, B_1), D_2(B_2)$ [SS]



The low-order four bits of each byte in the second operand field, the numerics, are placed in the low-order bit positions of the corresponding bytes in the first operand fields.

The instruction is storage to storage. Movement is left to right through each field one byte at a time, and the fields may overlap in any desired way.

The numerics are not changed or checked for validity. The high-order four bits of each byte, the zones, remain unchanged in both operand fields.

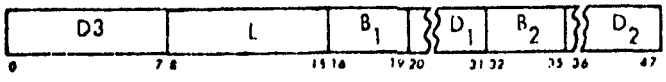
Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing

6.4.3 MOVE ZONES

MVZ $D_1(L, B_1), D_2(B_2)$ [SS]



The high-order four bits of each byte in the second operand field, the zones, are placed in the high-order four bit positions of the corresponding bytes in the first operand field.

The instruction is storage-to-storage. Movement is left to right through each field one byte at a time, and the fields may overlap in any desired way.

The zones are not changed or checked for validity. The low-order four bits of each byte, the numerics, remain unchanged in both operand fields.

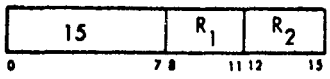
Condition Code: The code remains unchanged.

Program Interruptions:

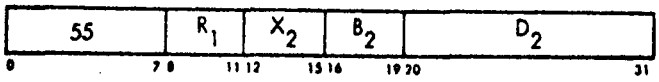
Protection
Addressing

6.4.4 COMPARE LOGICAL

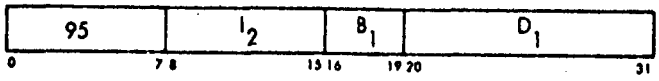
CLR R_1, R_2 [RR]



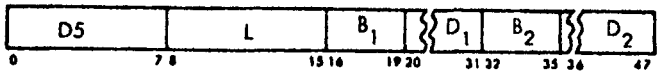
CL $R_1, D_2(X_2, B_2)$ [RX]



CLI $D_1(B_1), I_2$ [SI]



CLC $D_1(L, B_1), D_2(B_2)$ [SS]



The first operand is compared with the second operand, and the result is indicated in the condition code.

The instructions allow comparisons that are register-to-register, storage-to-register, instruction-to-storage, and storage-to-storage.

Comparison is binary, and all codes are valid. The operation proceeds left to right and ends as soon as an inequality is found or the end of the fields is reached. However, when part of an operand in CLC is specified in an unavailable location, the operation may be terminated by the addressing exception, even though an inequality could have been found in a comparison of the available operand parts.

Resulting Condition Code:

0	Operands are equal
1	First operand is low
2	First operand is high
3	--

Program Interruptions:

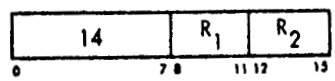
Addressing (CL, CLI, CLC only)
Specification (CL only)

Programming Note

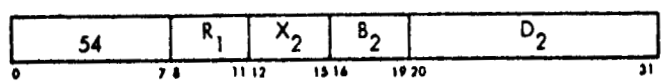
The COMPARE LOGICAL is unique in treating all bits alike as part of an unsigned binary quantity. In variable-length operation, comparison is left to right and may extend to field lengths of 256 bytes. The operation may be used to compare unsigned packed decimal fields or alphameric information in any code that has a collating sequence based on ascending or descending binary values. For example, EBCDIC has a collating sequence based on ascending binary values.

6.4.5 AND

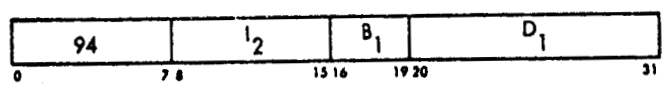
NR R_1, R_2 [*RR*]



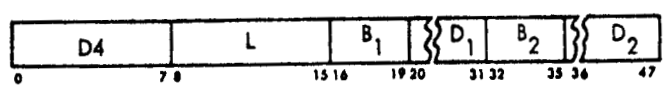
N $R_1, D_2(X_2, B_2)$ [*RX*]



NI $D_1(B_1), I_2$ [*SI*]



NC $D_1(L, B_1), D_2(B_2)$ [*SS*]



The logical product (AND) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in both operands contain a one; otherwise, the result bit is set to zero. All operands and results are valid.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

Program Interruptions:

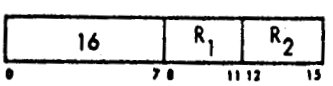
Protection (NI or NC)
Addressing (N, NI, NC only)
Specification (N only)

Programming Note

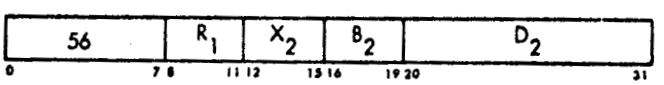
The AND may be used to set a bit to zero.

6.4.6 OR

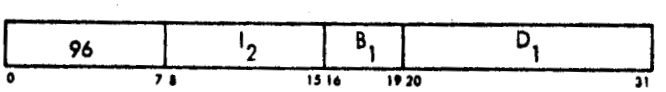
OR R_1, R_2 [RR]



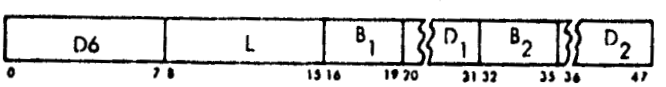
O $R_1, D_2(X_2, B_2)$ [RX]



OI $D_1(B_1), I_2$ [SI]



OC $D_1(L, B_1), D_2(B_2)$ [SS]



The logical sum (OR) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit position in one or both operands contain a one; otherwise, the result bit is set to zero. All operands and results are valid.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

Program Interruptions:

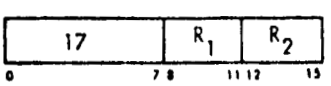
Protection (OI and OC)
Addressing (O, OI, OC only)
Specification (O only)

Programming Note

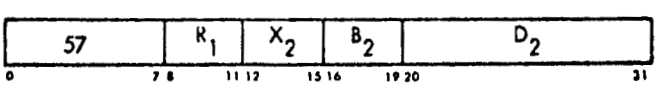
The OR may be used to set a bit to one.

6.4.7 EXCLUSIVE OR

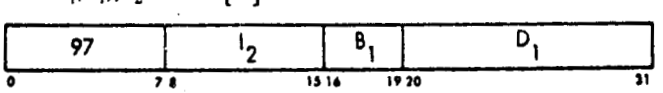
XR R_1, R_2 [*RR*]



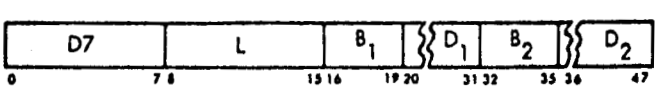
X $R_1, D_2(X_2, B_2)$ [*RX*]



XI $D_1(B_1), I_2$ [*SI*]



XC $D_1(L, B_1), D_2(B_2)$ [*SS*]



The modulo-two sum (exclusive OR) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective exclusive OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

The instruction differs from AND and OR only in the connective applied.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

Program Interruptions:

Protection (XI and XC)
Addressing (X, XI, XC only)
Specification (X only)

Programming Notes

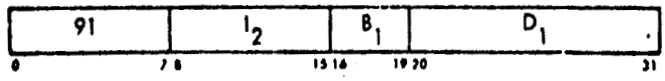
The exclusive OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

Any field exclusive OR'ed with itself becomes all zeros.

The sequence A exclusive OR'ed B, B exclusive OR'ed A, A exclusive OR'ed B results in the exchange of the contents of A and B without the use of an auxiliary buffer area.

6.4.8 TEST UNDER MASK

TM $D_1(B_1), I_2$ [SI]



The state of the first operand bits selected by a mask is used to set the condition code.

The byte of immediate data, I_2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in storage specified by the first operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, the condition code is made 0. The code is also made 0 when the mask is all-zero. When the selected bits are all-one, the code is 3; otherwise, the code is made 1. The character in storage is not changed.

Resulting Condition Code:

0 Selected bits all-zero; mask is all-zero
1 Selected bits mixed zero and one

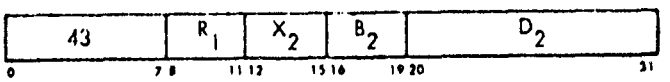
2 --
3 Selected bits all-one

Program Interruptions:

Addressing

6.4.9 INSERT CHARACTER

IC $R_1, D_2(X_2, B_2)$ |RX|



The eight-bit character at the second operand address is inserted into bit positions 24-31 of the register specified as the first operand location. The remaining bits of the register remain unchanged.

The instruction is storage to general register. The byte to be inserted is not changed or inspected.

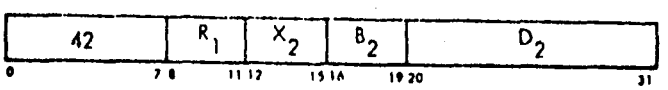
Condition Code: The code remains unchanged.

Program Interruptions:

Addressing

6.4.10 STORE CHARACTER

STC $R_1, D_2(X_2, B_2)$ |RX|



Bit positions 24-31 of the register designated as the first operand are placed at the second operand address.

The instruction is general register to storage. The byte to be stored is not changed or inspected.

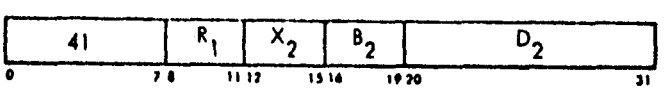
Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing

6.4.11 LOAD ADDRESS

LA $R_1, D_2(X_2, B_2)$ |RX|



The address of the second operand is inserted in the low-order 24 bits of the general register specified by R_1 . The remaining bits of the general register are made zero. No storage references for operands take place.

The address specified by the X_2, B_2, and D_2 fields is inserted in bits 8-31 of the general register specified by R_1. Bits 0-7 are set to zero. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 24th bit are ignored.

Condition Code: The code remains unchanged.

Program Interruptions:

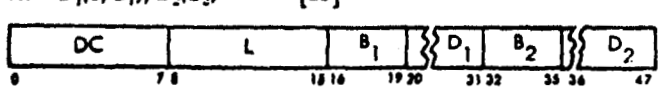
None.

Programming Note

The same general register may be specified by the R_1, X_2, and B_2 instruction field, except that general register 0 can be specified only by the R_1 field. In this manner, it is possible to increment the low-order 24 bits of a general register, other than 0, by the contents of the D_2 field of the instruction. The register to be incremented should be specified by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero).

6.4.12 TRANSLATE

TR $D_1(L, B_1), D_2(B_2)$ [SS]



The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each eight-bit function byte selected from the list replaces the corresponding argument in the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is added to the entire initial address, the second operand address, in the low-order bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte.

All data are valid. The operation proceeds until the first operand field is exhausted. The list is not altered unless an overlap occurs.

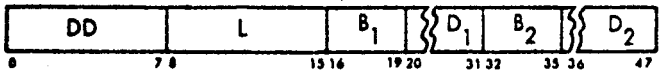
Condition Code; The code remains unchanged.

Program Interruptions:

Protection
Addressing

6.4.13 TRANSLATE AND TEST

TRT $D_1(L, B_1), D_2(B_2)$ [SS]



The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each eight-bit function byte thus selected from the list is used to determine the continuation of the operation. When the function byte is a zero, the operation proceeds by fetching and translating the next argument byte. When the function byte is non-zero, the operation is completed by inserting the related argument address in general register 1, and by inserting the function byte in general register 2.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in storage. Fetching of the function byte from the list is performed as in TRANSLATE. The function byte retrieved from the list is inspected for the all-zero combination.

When the function byte is zero, the operation proceeds with the next operand byte. When the first operand field is exhausted before a non-zero function byte is encountered, the operation is completed by setting the condition code to 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is non-zero, the related argument address is inserted in the low-order 24 bits of general register 1. This address points to the argument last translated. The high-order eight bits of register 1 remain unchanged. The function byte is inserted in the low-order eight bits of general register 2. Bits 0-23 of register 2 remain unchanged. The condition code is set to 1 when the one or more argument bytes have not been translated. The condition code is set to 2 if the last function byte is non-zero.

Resulting Condition Code:

0	All function bytes are zero
1	Non-zero function byte before the first operand field is exhausted
2	Last function byte is non-zero
3	--

Program Interruptions:

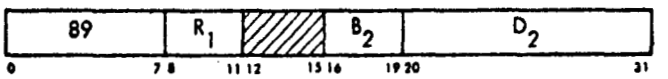
Addressing

Programming Note

The TRANSLATE AND TEST is useful for scanning an input stream and locating delimiters. The stream can thus be rapidly broken into statements or data fields for further processing.

6.4.14 SHIFT LEFT SINGLE

SLL $R_1, D_2(B_2)$ [RS]



The first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the general register specified by R_1 participate in the shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated low-order register positions.

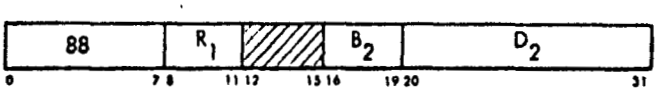
Condition Code: The code remains unchanged.

Program Interruptions:

None

6.4.15 SHIFT RIGHT SINGLE

SRL $R_1, D_2(B_2)$ [RS]



The first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

Supplement to Paragraph 6.4.13 Programming Notes - Page 75

The address of operand 1 in this instruction is limited to 64K (K = 1024). The operand 1 address at which the non-zero byte is found is stored in general register 1 at the conclusion of instruction execution. Only the low 16 bits of operand 1 address will be stored. If the operand address is greater than 64K, the high bits will be lost.

All 32 bits of the general register specified by R_1 participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated high-order register positions.

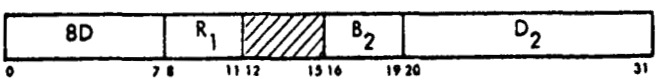
Condition Code: The code remains unchanged.

Program Interruptions:

None

6.4.16 SHIFT LEFT DOUBLE

SLDL $R_1, D_2(B_2)$ [RS]



The double-length first operand is shifted left the number of bits specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. An odd value for R_1 is a specification exception and causes a program interruption. The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the even/odd register pair specified by R_1 participate in the shift. High-order bits are shifted out of the even-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

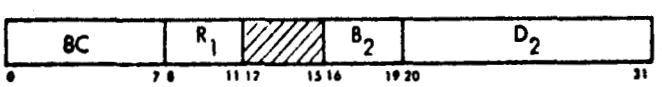
Condition Code: The code remains unchanged.

Program Interruptions:

Specification

6.4.17 SHIFT RIGHT DOUBLE

SRDL $R_1, D_2(B_2)$ [RS]



The double-length first operand is shifted right the number of bits specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. An odd value for R_1 is

a specification exception and causes a program interruption. The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the even/odd register pair specified by R_1 participate in the shift. Low-order bits are shifted out of the odd-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

Condition Code: The code remains unchanged.

Program Interruptions:

Specification

Programming Note

The logical shifts differ from the arithmetic shifts in that the high-order bit participates in the shift and is not propagated, the condition code is not changed, and no overflow occurs.

6.5 LOGICAL OPERATION EXCEPTIONS

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in logical operations.

Protection

The CPU storage protection bit is set to a one (1). The operation is suppressed on a store violation. Therefore, the condition code and data in registers and storage remain unchanged. The only exceptions are the variable-length, storage-to-storage operations (those containing a length specification), which are terminated. For terminated operations, the result data and condition code, if affected, are unpredictable and should not be used for further computation.

Addressing

An address designated an operand location outside the available storage for the installation: In most cases, the operation is terminated. The result data and the condition code, if affected, are unpredictable and should not be used for further computation. The exceptions are the immediate operations AND (NI), EXCLUSIVE OR (XI), OR (OI), MOVE (MVI), and STORE CHARACTER, which are suppressed.

Specification

A fullword operand in a storage-to-register operation is not located on a 32-bit boundary or an odd register address is specified for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of LOAD ADDRESS is not tested. The address restrictions do not apply to the components from which an address is generated - the contents of the D_1 and D_2 fields, and the contents of the registers specified by X_2, B_1, and B_2.

SECTION VII

BRANCHING

Instructions are performed by the central processing unit primarily in the sequential order of their locations. A departure from this normal sequential operation may occur when branching is performed. The branching instructions provide a means for making a two-way choice, to reference a subroutine, or to repeat a segment of coding, such as a loop.

Branching is performed by introducing a branch address as a new instruction address.

The branch address may be obtained from one of the general registers or it may be the address specified by the instruction. The branch address is independent of the updated instruction address.

The detailed operation of branching is determined by the condition code which is part of the program status word (PSW) or by the results in the general registers which are specified in the loop-closing operations.

During a branching operation, the rightmost half of the PSW, including the updated instruction address, may be stored before the instruction address is replaced by the branch address. The stored information may be used to link the new instruction sequence with the preceding sequence.

The instruction EXECUTE is grouped with the branching instructions. The branch address of EXECUTE designates a single instruction to be inserted in the instruction sequence. The updated instruction address normally is not changed in this operation, and only the instruction located at the branch address is executed.

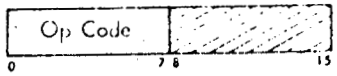
All branching operations are provided in the standard instruction set.

7.1 NORMAL SEQUENTIAL OPERATION

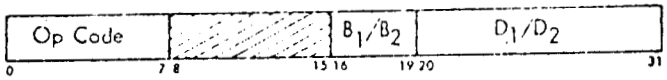
Normally, operation of the CPU is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction-address field of the PSW. The instruction address is increased by the number of bytes of the instruction to address the next instruction in sequence. This new instruction-address value, called the updated instruction address, replaces the previous contents of the instruction-address field in the PSW. The current instruction is executed, and the same steps are repeated, using the updated instruction address to fetch the next instruction.

Instructions occupy a halfword or a multiple thereof. An instruction may have up to three halfwords. The number of halfwords in an instruction is specified by the first two instruction bits. A 00 code indicates a halfword instruction, codes 01 and 10 indicate a two-halfword instruction, and code 11 indicates a three-halfword instruction.

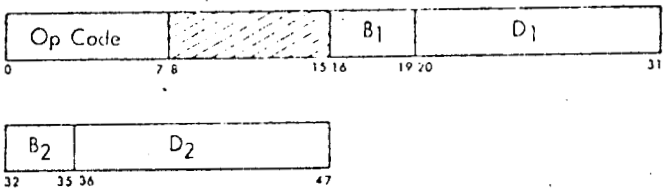
Halfword Format



Two-Halfword Format



Three-Halfword Format



Storage wraps around from the maximum addressable storage location, byte location 1,048,575, to byte location 0. An instruction having its last halfword at the maximum storage location is followed by the instruction at address 0. Also, a multiple-halfword instruction may straddle the upper storage boundary; no special indication is given in these cases.

Conceptually, an instruction is fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage width and overlap of instruction execution with storage access may cause actual instruction fetching to be different.

A change in the sequential operation may be caused by branching, status switching, interruption, or manual intervention. Sequential operation is initiated and terminated from the system control panel.

Programming Note

It is possible to modify an instruction in storage by means of the immediately preceding instruction.

7.1.1 SEQUENTIAL OPERATION EXCEPTIONS

Exceptional instruction addresses or operation codes cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. (In this manual, part of the description of each class of instructions is a list of the program interruptions that may occur for these instructions.) The new PSW is not checked for exceptions when it becomes current. These checks occur when the next instruction is executed. The following program interruptions may occur in normal instruction sequencing, independently of the instruction performed.

Operation

An operation exception occurs when the CPU attempts to decode an operation code that is not assigned. The operation exception can be accompanied by an addressing or specification exception if the instruction class associated with the undefined operation has uniform requirements for operand designation. An instruction class is a group of instructions whose four leftmost bits are identical.

Addressing

An addressing exception occurs when an instruction halfword is located outside the available storage for the particular installation. This situation can occur when normal instruction sequencing goes from a valid storage region into an unavailable region, or following a branching or load-PSW operation or an interruption. When the last locations in available storage contain an instruction that again introduces a valid instruction address (i.e., a branch), a program interruption is caused because the updated instruction address designated an unavailable location which will be referenced by instruction prefetching.

Specification

A specification exception occurs when the instruction address in the PSW is odd. This odd-address error can occur when after a branching or load-PSW operation or after an interruption.

A specification exception will occur when the PSW protection key is non-zero. This error can occur after a PSW is loaded or after an interruption.

In each case, the instruction is suppressed; therefore, the condition code and data in storage and registers remain unchanged. The instruction address stored as part of the old PSW has been updated by the number of halfwords indicated by the instruction length code in the old PSW.

Programming Notes

When a program interruption occurs, the current PSW is stored in the old PSW location. The instruction address stored as part of this old PSW is thus the updated instruction address, having been updated by the number of halfwords indicated in the instruction-length code of the same PSW. The interruption code in this old PSW identifies the cause of the interruption and aids in the programmed interpretation of the old PSW.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

7.2 DECISION-MAKING

Branching may be conditional or unconditional. Unconditional branches replace the updated instruction address with the branch address. Conditional branches may use the branch address or may leave the updated instruction address unchanged. When branching takes place, the instruction is called successful; otherwise, it is called unsuccessful.

Whether a conditional branch is successful depends on the result of operations concurrent with the branch or preceding the branch. The former case is represented by BRANCH ON COUNT and the branch-on-index instructions. The latter case is represented by BRANCH ON CONDITION, which inspects the condition code that reflects the result of a previous arithmetic, logical, or I/O operation.

The condition code provides a means for data-dependent decision-making. The code is inspected to qualify the execution of the conditional-branch instructions. The code is set by some operations to reflect the result of the operation, independently of the previous setting of the code. The code remains unchanged for all other operations.

The condition code occupies bit positions 34 and 35 of the PSW. When the PSW is stored during status switching, the condition code is preserved as part of the PSW. Similarly, the condition code is stored as part of the rightmost half of the PSW in a branch-and-link operation. A new condition code is obtained by a LOAD PSW or SET PROGRAM MASK or by the new PSW loaded as a result of an interruption.

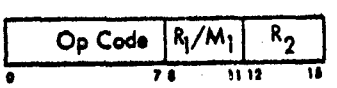
The condition code indicates the outcome of some of the arithmetic, logical, or I/O operations. It is not changed for any branching operation, except for EXECUTE. In the case of EXECUTE, the condition code is set or left unchanged by the subject instruction, as would have been the case had the subject instruction been in the normal instruction stream.

The table at the end of this section lists all instructions capable of altering the condition code and the meaning of the codes for these instructions.

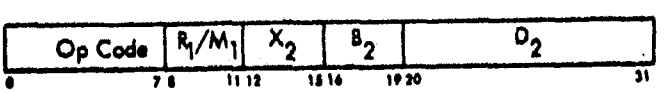
7.3 INSTRUCTION FORMATS

Branching instructions use the following formats:

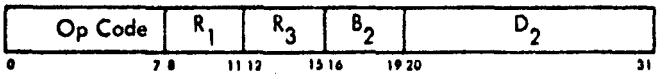
RR Format



RX Format



RS Format



In these formats R_1 specifies the address of a general register. In BRANCH ON CONDITION a mask field (M_1) identifies the bit values of the condition code. The branch address is defined differently for the three formats.

In the RR format, the R_2 field specifies the address of a general register containing the branch address, except when R_2 is zero, which indicates no branching. The same register may be specified by R_1 and R_2.

In the RX format, the contents of the general registers specified by the X_2 and B_2 fields are added to the content of the D_2 field to form the branch address.

In the RS format, the content of the general register specified by the B_2 field is added to the content of the D_2 field to form the branch address. The R_3 field in this format specifies the location of the second operand and implies the location of the third operand. The first operand is specified by the R_1 field. The third operand location is always odd. If the R_3 field specifies an even register, the third operand is obtained from the next higher addressed register. If the R_3 field specifies an odd register, the third operand location coincides with the second operand location.

A zero in a B_2 or X_2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register for both address modification and operand location. The order in which the contents of the general registers are used for the different parts of an operation is:

1. Address computation.
2. Arithmetic or link information storage.
3. Replacement of the instruction address by the branch address obtained under step 1.

Results are placed in the general register specified by R_1. Except for the storing of the final results, the contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

NOTE:

In the detailed description of the individual instructions, the mnemonic and the symbolic operand designation for the NSSC-II assembly language are shown with each instruction. For BRANCH ON INDEX HIGH, for example, BXH is the mnemonic and R_1, R_3, $D_2(B_2)$ the operand designation.

Programming Note

In several instructions the branch address may be specified in two ways: in the RX format, the branch address is the address specified by X_2, B_2, and D_2; in the RR format, the branch address is in the low-order 20 bits of the register specified by R_2. Note that the relation of the two formats in branch-address specification is not the same as in operand-address specification. For operands, the address specified by X_2, B_2, and D_2 is the operand address, but the register specified by R_2 contains the operand itself.

7.4 BRANCHING INSTRUCTIONS

The branching instructions and their mnemonics, formats, and operation codes follow. The table also shows the exceptions that cause a program interruption during execution of EXECUTE. The subject instruction of EXECUTE follows its own rules for interruptions. The condition code is never changed for branching instructions.

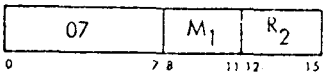
NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Branch on Condition	BCR	RR		07
Branch on Condition	BC	RX		47
Branch and Link	BALR	RR		05
Branch and Link	BAL	RX		45
Branch on Count	BCTR	RR		06
Branch on Count	BCT	RX		46
Branch on Index High	BXH	RS		86
Branch on Index Low or Equal	BXLE	RS		87
Execute	EX	RX	A,S,EX	44

NOTES

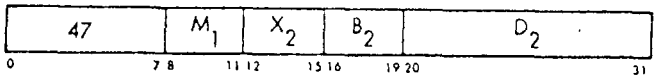
A	Addressing exception
EX	Execute exception
S	Specification exception

7.4.1 BRANCH ON CONDITION

BCR M_1, R_2 [RR]



BC $M_1, D_2(X_2, B_2)$ [RX]



The updated instruction address is replaced by the branch address if the state of the condition code is as specified by M_1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The M_1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four condition codes (0, 1, 2, and 3) as follows:

INSTRUCTION BIT	MASK POSITION VALUE	CONDITION CODE
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of one.

Condition Code: The code remains unchanged.

Program Interruptions:

None

Programming Note

When a branch is to be made on more than one condition code, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made on condition codes 0 and 1.

When all four mask bits are ones, that is, the mask position value is 15, the branch is unconditional. When all four mask bits are zero or when the R_2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation.

Condition-Code Settings

	CODE STATE 0	1	2	3
Fixed-Point Arithmetic				
Add H/F	zero	<zero	>zero	overflow
Add Logical	zero, no carry	not zero, no carry	zero, carry	not zero, carry
Compare H/F	equal	low	high	--

	CODE STATE 0	1	2	3
Fixed-Point Arithmetic (Continued)				
Load and Test	zero	<zero	>zero	carry
Load Complement	zero	<zero	>zero	overflow
Load Negative	zero	<zero	--	--
Load Positive	zero	--	>zero	overflow
Shift Left Double	zero	<zero	>zero	overflow
Shift Left Single	zero	<zero	>zero	overflow
Shift Right Double	zero	<zero	>zero	--
Shift Right Single	zero	<zero	>zero	--
Subtract H/F	zero	<zero	>zero	overflow
Subtract Logical	--	not zero, no carry	zero, carry	not zero, carry
Logical Operations				
And	zero	not zero	--	--
Compare Logical	equal	low	high	--
Exclusive Or	zero	not zero	--	--
Or	zero	not zero	--	--
Test Under Mask	zero	mixed	--	one
Translate and Test	zero	incomplete	complete	--
Status Switching				
Test and Set	zero	one	--	--
Input/Output Operations				
Start I/O	successful	busy		

Notes

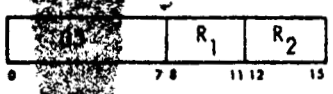
busy	Unit or channel busy
carry	A carryout of the sign position occurs
complete	Last result byte nonzero
equal	Operands compare equal
F	Fullword
>zero	Result is greater than zero
H	Halfword
high	First operand compares high
incomplete	Nonzero result byte; not last
<zero	Result is less than zero
low	First operand compares low
mixed	Selected bits are both zero and one
not zero	Result is not all zero
one	Selected bits are one
overflow	Result overflows
zero	Result or selected bits are zero

NOTE:

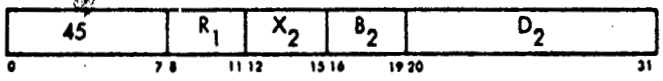
The condition code also may be changed by LOAD PSW, SET PROGRAM MASK, DIAGNOSE and by an interruption.

7.4.2 BRANCH AND LINK

BALR R_1, R_2 [RR]



BAL $R_1, D_2(X_2, B_2)$ [RX]



The rightmost 32 bits of the PSW, including the updated instruction address, are stored as link information in the general register specified by R_1. Subsequently, the instruction address is replaced by the branch address.

The branch address is determined before the link information is stored. The link information contains the instruction length code, the condition code, and the program mask bits, as well as the updated instruction address. The instruction-length code is 1 or 2, depending on the format of the BRANCH AND LINK.

Condition Code: The code remains unchanged.

Program Interruptions:

None

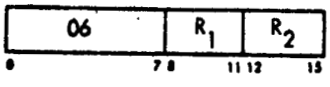
Programming Note

The link information is stored without branching when the R_2 field contains zero.

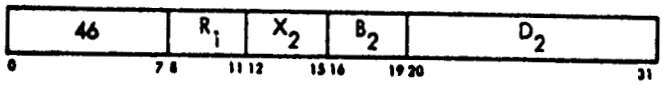
When BRANCH AND LINK is the subject instruction of EXECUTE, the instruction-length code is 2.

7.4.3 BRANCH ON COUNT

BCTR R_1, R_2 [RR]



BCT $R_1, D_2(X_2, B_2)$ [RX]



The content of the general register specified by R_1 is algebraically reduced by one. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address is replaced by the branch address.

The branch address is determined prior to the counting operation. Counting does not change the condition code. The overflow occurring on transition from the maximum negative number to the maximum positive number is ignored. Otherwise, the subtraction proceeds as in fixed-point arithmetic, and all 32 bits of the general register participate in the operation.

Condition Code: The code remains unchanged.

Program Interruptions:

None

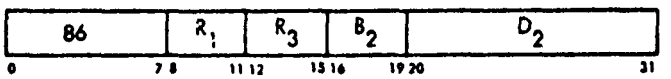
Programming Notes

An initial count of one results in zero, and no branching takes place. An initial count of zero results in minus one and causes branching to be executed.

Counting is performed without branching when the R_2 field in the RR format contains zero.

7.4.4 BRANCH ON INDEX HIGH

BXH $R_1, R_3, D_2(B_2)$ [RS]



An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is high, the instruction address is replaced by the branch address. When the sum is low or equal, instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is determined prior to the addition and comparison.

Overflow caused by the addition is ignored and does not affect the comparison. Otherwise, the addition and comparison proceed as in fixed-point arithmetic. All 32 bits of the general registers participate in the

operations, and negative quantities are expressed in two's-complement notation. When the first operand and comparand locations coincide, the original register contents are used as the comparand.

Condition Code: The code remains unchanged.

Program Interruptions:

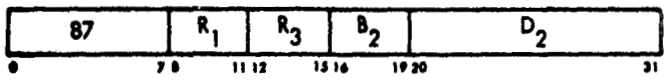
None

Programming Note

The name "branch on index high" indicates that one of the major purposes of this instruction is the incrementing and testing of an index value. The increment may be algebraic and of any magnitude.

7.4.5 BRANCH ON INDEX LOW OR EQUAL

BXLE $R_1, R_3, D_2(B_2)$ [RS]



An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is low or equal, the instruction address is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is determined prior to the addition and comparison.

This instruction is similar to BRANCH ON INDEX HIGH, except that the branch is successful when the sum is low or equal compared to the comparand.

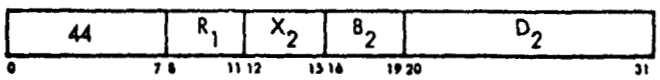
Condition Code: The code remains unchanged.

Program Interruptions:

None

7.4.6 EXECUTE

EX $R_1, D_2(X_2, B_2)$ [RX]



The single instruction at the branch address is modified by the content of the general register specified by R_1, and the resulting subject instruction is executed.

Bits 8-15 of the instruction designated by the branch address are OR'ed with bits 24-31 of the register specified by R_1, except when register 0 is specified, which indicates that no modification takes place. The subject instruction may be 16, 32, or 48 bits in length. The OR'ing does not change either the content of the register specified by R_1 or the instruction in storage and is effective only for the interpretation of the instruction to be executed.

The execution and exception handling of the subject instruction are exactly as if the subject instruction were obtained in normal sequential operation, except for instruction address and instruction-length recording.

The instruction address of the PSW is increased by the length of EXECUTE. This updated address and the length code (2) of EXECUTE are stored in the PSW in the event of a branch-and-link subject instruction or in the event of an interruption.

When the subject instruction is a successful branching instruction, the updated instruction address of the PSW is replaced by the branch address of the subject instruction. When the subject instruction in turn is an EXECUTE, an execute exception occurs and results in a program interruption. The effective address of EXECUTE must be even; if not, a specification exception will cause a program interruption.

Condition Code: The code may be set by the subject instruction.

Program Interruptions:

Execute
Addressing
Specification

Programming Notes

The OR'ing of eight bits from the general register with the designated instruction permits indirect length, index, mask, immediate data, and arithmetic-register specification.

If the subject instruction is a successful branch, the length code still stands at 2.

An addressing or specification exception may be caused by EXECUTE or by the subject instruction.

Supplement to Paragraph 7.4.6 Programming Notes - Page 90

The target instruction of execute cannot specify general register Ø as a base register unless the register is first initialized to a desired value. (Normally, when general register Ø is specified as a base register, the hardware substitutes all zeros rather than use the actual contents of the register.)

7.4.6.1 Execute Exceptions

Exceptional operand designations and subject-instruction operation code specifying EXECUTE cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause. Exceptions that cause a program interruption in the use of EXECUTE are:

Execute

An EXECUTE instruction has as its subject instruction another EXECUTE.

Addressing

The branch address of EXECUTE designates an instruction-halfword location outside the available storage for the particular installation.

Specification

The branch address of EXECUTE is odd.

These four exceptions occur only for EXECUTE. The instruction is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Exceptions arising for the subject instruction of EXECUTE are the same as would have arisen had the subject instruction been in the normal instruction stream. However, the instruction address stored in the old PSW is the address of the instruction following EXECUTE. Similarly, the instruction-length code in the old PSW is the instruction-length code (2) of EXECUTE.

The address restrictions do not apply to the components from which an address is generated - the content of the D_1 field and the content of the register specified by B_1.

Programming Note

An unavailable or odd branch address of a successful branch is detected during the execution of the next instruction and not as part of the branch.

SECTION VIII

STATUS SWITCHING

A set of operations is provided to switch the status of the CPU, of storage, and of communication between systems.

The over-all CPU status is determined by several program-state alternatives, each of which can be changed independently to its opposite and most of which are indicated by a bit in the program status word (PSW). The CPU status is further defined by the instruction address, the condition code, the instruction-length code, the storage-protection key, and the interruption code. These all occupy fields in the PSW.

Protection of main storage is achieved by matching a key in storage with a protection key in the PSW or in a channel. The protection status of storage may be changed by introducing new storage keys, using SET STORAGE KEY. The storage keys may be inspected by using INSERT STORAGE KEY.

Facilities are provided whereby a system formed by CPU, storage, and I/O can communicate with other systems. The instruction READ DIRECT make signals available to the CPU; WRITE DIRECT provides signals to other systems.

All status-switching instructions, other than those of the protection feature or direct control feature, are provided in the standard instruction set.

8.1 PROGRAM STATES

The four types of program-state alternatives, which determine the overall CPU status, are named Problem/Supervisor, Wait/Running, Masked/Interruptible, and Stopped/Operating. These states differ in the way they affect the CPU functions and in the way their status is indicated and switched. The masked states have several alternatives; all other states have only one alternative.

All program states are independent of each other in their function, indication, and status switching. Status switching does not affect the contents of the arithmetic registers or the execution of I/O operations but may affect the timer operation.

8.1.1 PROBLEM STATE

The choice between supervisor and problem state determines whether the full set of instructions is valid. The names of these states reflect their normal use.

In the problem state all I/O, protection, and clock set instructions are invalid, as well as LOAD PSW, SET SYSTEM MASK, and DIAGNOSE. These are called privileged instructions. A privileged instruction encountered in the problem state constitutes a privileged-operation exception and causes a program interruption. In the supervisor state all instructions are valid.

When bit 15 of the PSW is zero, the CPU is in the supervisor state. When bit 15 is one, the CPU is in the problem state. The supervisor state is not indicated on the operator section of the system control panel.

The CPU is switched between problem and supervisor state by changing bit 15 of the PSW. This bit can be changed only by introducing a new PSW. Thus status switching may be performed by LOAD PSW, using a new PSW with the desired value for bit 15. Since LOAD PSW is a privileged instruction, the CPU must be in the supervisor state prior to the switch. A new PSW is also introduced when the CPU is interrupted. The SUPERVISOR CALL causes an interruption and thus may change the CPU state. Similarly, initial program loading introduces a new PSW and with it a new CPU state. The new PSW may introduce the problem or supervisor state regardless of the preceding state. No explicit operator control is provided for changing the supervisor state.

Timer incrementing/decrementing is not affected by the choice between supervisor and problem state.

Programming Note

To allow return from an interruption-handling routine to a preceding program by a LOAD PSW, the PSW for the interruption routine should specify the supervisor state.

8.1.2 WAIT STATE

In the wait state no instructions are processed, and storage is not addressed repeatedly for this purpose, whereas in the running state, instruction fetching and execution proceed in the normal manner.

When bit 14 of the PSW is one, the CPU is waiting. When bit 14 is zero, the CPU is in the running state.

The CPU is switched between wait and running state by changing bit 14 of the PSW. This bit can be changed only by introducing an entire new PSW, as is the case with the problem-state bit. Thus, switching from the running state may be achieved by the privileged instruction LOAD PSW, by an interruption such as for SUPERVISOR CALL, or by initial program loading. Switching from the wait state may be achieved by an I/O or external interruption or, again by initial program loading. The new PSW may introduce the wait or running state regardless of the preceding state. No explicit operator control is provided for changing the wait state.

Timer incrementing/decrementing is not affected by the choice between running and wait state.

Programming Note

To leave the wait state without manual intervention, the CPU should remain interruptible for some active I/O or external interruption source.

When because of machine malfunction the CPU is unable to end an instruction or I/O interrupts are disabled, the stop key is not effective, and initial program loading or system reset should be used.

Input/output operations continue to completion while the CPU is in the problem, wait, or masked state. However, no new I/O operations can be initiated while the CPU is stopped, waiting, or in the problem state. Also, the interruption caused by I/O completion is lost when the CPU is in the stopped state.

8.2 PROTECTION

Protection is provided to protect the contents of certain areas of main storage from destruction (or misuse) caused by erroneous storing of information during the execution of a program. Locations may be protected against store violations.

8.2.1 AREA IDENTIFICATION

For protection purposes, main storage is divided into blocks of 1024 bytes, each block having an address that is a multiple of 1024.

8.2.2 PROTECTION ACTION

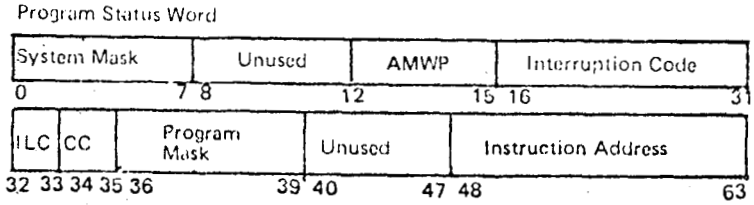
A 2 bit key is associated with each block of storage for protection from the CPU and DMA.

8.3 PROGRAM STATUS WORD

The PSW contains all information not contained in storage or registers but required for proper program execution. By storing the PSW, the program can preserve the detailed status of the CPU for subsequent insepction. By loading a new PSW or part of a PSW, the state of the CPU may be changed.

In certain circumstances all of the PSW is stored or loaded; in others, only part of it. The entire PSW is stored, and a new PSW is introduced when the CPU is interrupted. The rightmost 32 bits are stored in BRANCH AND LINK. The LOAD PSW introduces a new PSW; SET PROGRAM mask introduces a new condition code and program-mask field in the PSW; SET SYSTEM MASK introduces a new system-mask field.

The PSW has the following format:



The following is a summary of the purposes of the PSW fields:

System Mask

Bits 0-7 of the PSW are associated with I/O channels and external signals as specified in the following table. When a mask bit is one, the source can interrupt the CPU. When a mask bit is zero, the corresponding source can not interrupt the CPU, and interruptions remain pending.

SYSTEM MASK BIT	INTERRUPTION SOURCE
0	I/O
1	Unused
2	Unused
3	Unused
4	Unused
5	Unused
6	Unused
7	Timer

Bits 8-11 of the PSW must be zero when loaded; otherwise, a specification exception is recognized when an attempt is made to execute the instruction designated by the PSW. The protection key is stored unchanged.

ASCII(A)

When bit 12 of the PSW is one, the codes preferred for the USASCII-8 code are generated for decimal results. When PSW bit 12 is zero, the codes preferred for the extended binary-coded-decimal interchange code are generated.

The following instructions cause either the sign or zone code to be generated in accordance with the setting of PSW bit 12: CVD, UNPK.

Machine-Check Mask (M)

When bit 13 of the PSW is one, detection of a machine-check condition causes a machine-check interruption. When bit 13 of the PSW is zero, the CPU is disabled for machine-check interruptions; when a machine check occurs, the machine enters a hard stop.

Wait State (W)

When bit 14 of the PSW is one, the CPU is in the wait state. When PSW bit 14 is zero, the CPU is in the running state.

Problem State (P)

When bit 15 of the PSW is one, the CPU is in the problem state. When PSW bit 15 is zero, the CPU is in the supervisor state.

Interruption Code

Bits 16-31 of the PSW identify the cause of an interruption. Use of the code for all five interruption types is shown in a table appearing in Section IX, "Interruptions".

Instruction Length Code (ILC)

The code in PSW bits 32 and 33 indicates the length, in halfwords, of the last-interpreted instruction when a program or supervisor-call interruption occurs. The code is unpredictable for I/O, external, or machine-check interruptions. Encoding of these bits is summarized in a table appearing in Section IX, "Interruptions".

Condition Code (CC)

Bits 34 and 35 of the PSW are the two bits of the condition code. The condition codes for all instructions are summarized in a table appearing in Section VII, "Branching".

Program Mask

Bits 36-39 of the PSW are the four program mask bits. Each bit is associated with a program exception, as specified in the following table. When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs.

PROGRAM MASK BIT	PROGRRAM EXCEPTION
36	Fixed-point overflow
37	Unused
38	Unused
39	Unused

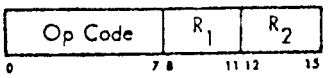
Instruction Address

Bits 44-63 of the PSW are the instruction address. This address specifies the leftmost eight-bit byte position of the next instruction. Bit 63 must be zero when loaded; otherwise, a specification exception is recognized when an attempt is made to execute the instruction designated by the PSW.

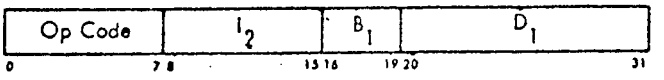
8.4 INSTRUCTION FORMAT

Status-switching instructions use the following two formats:

RR Format



SI Format



In the RR format, the R_1 field specifies a general register, except for SUPERVISOR CALL. The R_2 field specifies a general register in SET STORAGE KEY and INSERT STORAGE KEY. The R_1 and R_2 fields in SUPERVISOR CALL contain an identification code. In SET PROGRAM MASK THE R_2 field is ignored.

In the SI format the eight-bit immediate field (I_2) of the instruction contains an identification code. The I_2 field is ignored in LOAD PSW, SET SYSTEM MASK, and TEST AND SET. The content of the general register specified by B_1 is added to the content of D_1 field to form an address designating the location of an operand in storage. Only one operand location is required in status-switching operations.

A zero in the B_1 field indicates the absence of the corresponding address component.

NOTE:

In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the NSSC-II assembly language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and $D_1(B_1)$ the operand designation.

8.5 INSTRUCTIONS

The status-switching instructions and their mnemonics, formats, and operation codes follow. The table also indicates the feature which an instruction belongs and the exceptions in instruction and operand designation that cause a program interruption.

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Load PSW	LPSW	SI L	M,A,S	82
Set Program Mask	SPM	RR L		04
Set System Mask	SSM	SI	M,A	80
Supervisor Call	SVC	RR		0A
Set Storage Key	SSK	RR Z	M,A,S	08
Diagnose	-	SI	M,P,A,S	83
Test and Set	TS	SI C	P,A	93

NOTES

A	Addressing exception
C	Condition code is set
L	New condition code loaded
M	Privileged-operation exception
P	Protection exception
S	Specification exception

Programming Note

The program status is also switched by interruptions, initial program loading, and manual control.

8.5.1 LOAD PSW

LPSW $D_1(B_1)$ [SI]

The double word at the location designated by the operand address replaces the PSW.

The operand address must have its three low-order bits zero to designate a double word; otherwise, a specification exception results in a program interruption.

The double word which is loaded becomes the PSW for the next sequence of instructions. Bits 8-11 become the new protection key. Bits 48-63 of the double word become the new instruction address. Only bits 8-11 of the PSW are checked for program interruptions during the load-PSW operation. Other checks occur as part of the execution of the next instructions.

The interruption code in bit positions 16-31 of the new PSW is not retained as the PSW is loaded. When the PSW is subsequently stored because of an interruption, these bit positions contain a new code. Similarly, bits 32 and 33 of the PSW are not retained upon loading. They will contain the instruction-length code for the last-interpreted instruction when the PSW is stored during a branch-and-link operation or during a program or supervisor-call interruption.

Condition Code: The code is set according to bits 34 and 35 of the new PSW loaded.

Program Interruptions:

Privileged operation
Addressing
Specification

Programming Note

The CPU enters the problem state when LOAD PSW loads a double word with a one in bit position 15 and similarly enters the wait state if bit position 14 is one. The LOAD PSW is the only instruction available for entering the problem state or the wait state.

8.5.2 SET PROGRAM MASK

SPM R_1 [RR]

Bits 2-7 of the general register specified by the R_1 field replace the condition code and the program mask bits of the current PSW.

Bits 0, 1, and 8-31 of the register specified by the R_1 field are ignored. The contents of the register specified by the R_1 field remain unchanged.

The instruction permits setting of the condition code and the mask bits in either the problem or supervisor state.

Condition Code: The code is set according to bits 2 and 3 of the register specified by R_1.

Program Interruptions:

None

Programming Note

Bits 2-7 of the general register may have been loaded from the PSW by BRANCH AND LINK.

8.5.3 SET SYSTEM MASK

SSM $D_1(B_1)$ [SI]

The byte at the location designated by the operand address replaces the system mask bits of the current PSW.

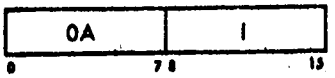
Condition Code: The code remains unchanged.

Program Interruptions:

Privileged operation
Addressing

8.5.4 SUPERVISOR CALL

SVC I [RR]



The instruction causes a supervisor-call interruption, with the I field of the instruction providing the interruption code.

The contents of bit positions 8-15 of the instruction are placed in bit positions 24-31 of the old PSW which is stored in the course of the interruption. Bit positions 16-23 of the old PSW are made zero. The old PSW is stored at location 32, and a new PSW is obtained from location 96. The instruction is valid in both problem and supervisor state.

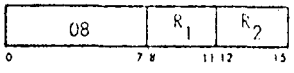
Condition Code: The code remains unchanged in the old PSW.

Program Interruptions:

None

8.5.5 SET STORAGE KEY

SSK R_1, R_2 [RR]



The key of the storage block addressed by the register designated by R_2 is set according to the key in the register designated by R_1.

The storage block of 1024 bytes, located on a multiple of the block length, is addressed by bits 12-21 of the register designated by the R_2 field. Bits 0-15 and 22-31 of this register are ignored.

The two bit key is obtained from bits 30 and 31 of the register designated by the R_1 field. Bits 0 through 29 of this register are ignored. Bit 30 = 1 protects against CPU and buffered I/O storing, and bit 31 = 1 protects against DMA storing.

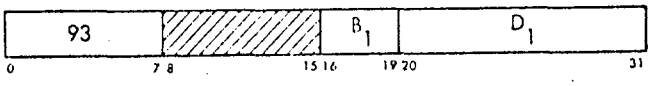
Condition Code: The code remains unchanged.

Program Interruptions:

Privileged operation
Addressing
Specification

8.5.6 TEST AND SET

TS $D_1(B_1)$ [SI]



The leftmost bit (bit position 0) of the byte located at the first operand address is used to set the condition code, and the entire addressed byte is set to all ones.

The byte in storage is set to all ones as it is fetched for the testing of bit position 0. No other access to this location is permitted between the moment of fetching and the moment of storing all ones.

The operation is terminated on any protection violation. The condition-code setting is unpredictable when a protection violation occurs.

Resulting Condition Code:

0 Leftmost bit of byte specified is zero
1 Leftmost bit of byte specified is one
2 --
3 --

Program Interruptions:

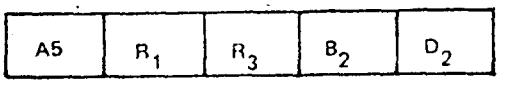
Protection
Addressing

Programming Note

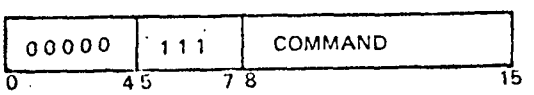
TEST AND SET can be used for controlled sharing of a common storage area by more than one program. To accomplish this, bit position 0 of a byte must be designated as the control bit. The desired interlock can be achieved by establishing a program convention in which a zero in the bit position indicates that the common area is available but a one means that the area is being used. Each using program then must examine this byte by means of TEST AND SET before making access to the common area. If the test sets the condition code to zero, the area is available for use; if it sets the condition code to one, the area cannot be used. Because TEST AND SET permits no access to the test byte between the moment of fetching (for testing) and the moment of storing all ones (setting), the possibility is eliminated of a second program's testing the byte before the first program is able to reset it.

8.5.7 START INPUT OUTPUT

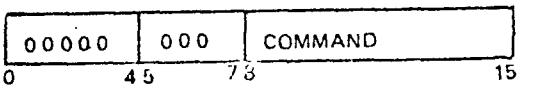
SIO $R_1, R_3, D_2(B_2)$ [RS]



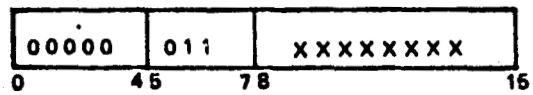
The SIO instruction is used to issue one of the four (4) 16-bit command words as shown below.



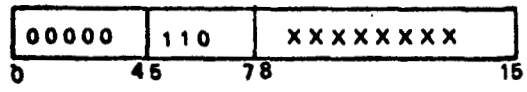
DIRECT OUTPUT (WRITE DIRECT)



DIRECT INPUT (READ DIRECT)



RESET INTERFACE (HALT I/O)



TEST INTERFACE

The least significant bits of the command (9-15) are the device address and function. The most significant bit of the command (bit 8) shall be a one (1) for a non-tester I/O function and zero (0) for a Tester function.

The SIO Instruction is also used to send or receive a 16-bit data word to/from an I/O device in conjunction with the command word (such as a write character to the typewriter).

The command word is placed at the Effective Address (EA); R_1 must contain any output word and any input word will be placed in R_3 during the instruction operation.

The least significant 16 bits of R_1 will be made available to the I/O interface after a direct INPUT/OUTPUT command is transmitted.

Similarly, the least significant bits of the R_3 will be replaced by the I/O input word after a direct INPUT/OUTPUT command is transmitted.

If the I/O interface is busy the condition code is set to one and the instruction terminates without performing the I/O operation.

On a test interface, the command is transmitted and the condition code is set without any attempt to exchange data; R_1 and R_3 are ignored.

Condition Codes:

0 Interface Not Busy
1 Interface Busy

Program Interruptions:

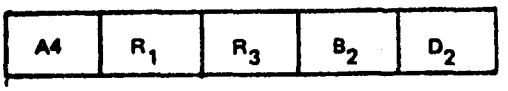
Addressing
Specification
Privileged operation

Programming Note

Both R_1 and R_3 must be designated on all Direct Input or Output functions.

8.5.8 TIMER READ AND SET

TMRS $R_1, R_3, D_2(B_2)$ (RS)



The R_1 field is used as an extension of the opcode to allow multiple functions of this instruction as shown below:

1. R_1 = 0000 Read the Real Time Clock
2. R_1 = 0010 Read and Set the Real Time Clock
3. R_1 = 0001 Read the Interval Timer
4. R_1 = 0011 Read and Set the Interval Clock

The Real Time Clock (or the Interval Timer) is read into the register designated by R_3.

The Real Time Clock is comprised of 32 bits and increments by one every 112.64 μsec. The period of the Real Time Clock is approximately 5.6 days. No interrupt is taken when the clock overflows.

The Interval Timer is comprised of 16 bits and decrements by one every 112.64 μsec. The Interval Timer has a period of approximately 7.38 seconds. An interrupt is taken when the clock decrements beyond zero if PSW bit 7 is set to one. If PSW bit 7 is set to zero, the timer interrupt will remain pending. When the Interval Timer is read, it will be placed into bits 16-31 of the register designated by R_3 and bits 0-15 will be set to zero.

When the Real Time Clock is to be set, the effective address (B_2+D_2) must designate a fullword (32 bit) memory location, which contains the number to be set into the clock. When the Interval Timer is to be set, the effective address (B_2+D_2) must designate a halfword (16 bit) memory location which contains the number to be set into the Interval Timer.

Condition Code: The code remains unchanged.

Program Interruptions:

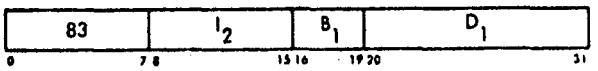
Addressing
Specification
Privileged operation (only 2 and 4 above)

Programming Note

The contents of the register designated by R3 will always be replaced by the current contents of either the clock or the timer even if R_3 is zero.

8.5.9 DIAGNOSE

SI



The purpose of the DIAGNOSE instruction is to execute self test and/or special purpose microprograms that may be developed from the NSSC-II.

The B_1 and D_1 fields are added together to compute an effective address. The NSSC-II microprogram branches to that address in the NSSC-II microprogram control storage and executes the microprogram routine that begins at that address.

The purpose of the I field may be defined in any way by the microprogram. The microprogram may affect any part of or all of the PSW, the program counter; the problem status, supervisor status, and interruptible status of the CPU, the condition code, and the contents of storage, registers, and timers, as well as the progress of I/O operations.

Since the instruction is not intended for problem-program or supervisor-program use, DIAGNOSE has no mnemonic.

Condition Code: The code is unpredictable.

Program Interruptions:

Privileged operation
Protection
Specification
Addressing

8.6 STATUS-SWITCHING EXCEPTIONS

Exceptional instructions, operand designations, or data cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code inserted in the old PSW identifies the cause of the interruption. The following exception conditions cause a program interruption in status-switching operations.

Privileged Operation

A LOAD PSW, SET SYSTEM MASK, SET STORAGE KEY, TMRS (Set only), or DIAGNOSE is encountered while the CPU is in the problem state.

Addressing

An address designates a location outside the available storage for the installation. The operation is terminated.

Specification

The operand address of a LOAD PSW does not have all three low-order bits zero; a PSW with a nonzero bits 8-11 is introduced.

When an instruction is suppressed, storage and external signals remain unchanged, and the PSW is not changed by information from storage.

When an interruption is taken, the instruction address stored as part of the old PSW has been updated by the number of halfwords indicated by the instruction-length code in the old PSW.

Operand addresses are tested only when used to address storage. The address restrictions do not apply to the components from which an address is generated: the content of the D_1 field and the content of the register specified by B_1.

Programming Notes

When a program interruption occurs, the current PSW is stored in the old PSW location. The instruction address stored as part of this old PSW is thus the updated instruction address, having been updated by the number of halfwords indicated in the instruction-length code of the same PSW.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

SECTION IX

INTERRUPTIONS

The interruption system permits the CPU to change its state as a result of conditions external to the system, in I/O units, or in the CPU itself. The five classes of these conditions are input/output, program, supervisor-call, external, and machine-check interruptions.

9.1 INTERRUPTION ACTION

An interruption consists of storing the current PSW as an old PSW and fetching a new PSW.

Processing resumes in the state indicated by the new PSW. The new PSW is not checked for programming errors when it becomes the current PSW. These checks are made when the next instruction is executed. The old PSW contains the address of the instruction that would have been executed next if an interruption had not occurred and the instruction-length code of the last-interpreted instruction.

Interruptions are taken only when the CPU is interruptible for the interruption source. Input/output and external interruptions may be masked by the system mask; one of the 9 program interruptions may be masked by the program mask; and the machine-check interruptions may be masked by the machine-check mask.

An interruption always takes place after one instruction interpretation is finished and before a new instruction interpretation is started. However, the occurrence of an interruption may affect the execution of the current instruction. To permit proper programmed action following an interruption, the cause of the interruption is identified and provision is made to locate the last-interpreted instruction.

The details of instruction execution, source identification, and location determination are explained in later sections.

Programming Note

A pending interruption will be taken even if the CPU becomes interruptible during only one instruction.

SOURCE IDENTIFICATION	INTERRUPTION CODE PSW BITS 16-31	MASK BITS	ILC SET	OPERATION EXECUTION
Input/Output (old PSW 56, new PSW 120, priority 4)				
Channel 0	aaaaaaaa aaaaaaaa	0	x	completed
Program Call (old PSW 40, new PSW 104, priority 2)				
Operation	00000000 00000001		1,2,3	suppressed
Privileged operation	00000000 00000010		1,2,3	suppressed
Execute	00000000 00000011		2	suppressed
Protection	00000000 00000100		0,2,3	suppressed or terminated
Addressing	00000000 00000101		0,1,2,3	suppressed or terminated
Specification	00000000 00000110		1,2,3	suppressed
Data	00000000 00000111		2,3	terminated
Fixed-point overflow	00000000 00001000	36	2,3	completed
Fixed-point divide	00000000 00001001		1,2	suppressed or completed
Exponent overflow	00000000 00001100		1,2	terminated
Exponent underflow	00000000 00001101	38	1,2	completed
Significance	00000000 00001110	39	1,2	completed
Floating-point divide	00000000 00001111		1,2	suppressed
Supervisor Call (old PSW 32, new PSW 96, priority 2)				
Instruction bits	00000000 rrrrrrrr		1,2,3	completed
External (old PSW 24, new PSW 88, priority 2)				
Timer	00000000 10000000	7	x	completed
Interrupt key	00000000 01000000	0	x	completed
Machine Check (old PSW 48, new PSW 112, priority 1)				
Machine malfunction	cccccccc cccccccc	13	x	terminated

NOTES
- a Device-dependent code bits
- c Bits of memory-dependent code, bit 40 of old PSW, if on, specifies parity error
- r Bits of R1 and R2 field of SUPERVISOR CALL
- x Unpredictable

9.1.1 INSTRUCTION EXECUTION

An interruption occurs when the preceding instruction is finished and the next instruction is not yet started. The manner in which the preceding instruction is finished may be influenced by the cause of the interruption. The instruction is said to have been completed, terminated, or suppressed.

In the case of instruction completion, results are stored and the condition code is set as for normal instruction operation, although the result may be influenced by the exception which has occurred.

In the case of instruction termination, all, part, or none of the result may be stored. Therefore, the result data are unpredictable. The setting of the condition code, if called for, may also be unpredictable. In general, the results should not be used for further computation.

In the case of instruction suppression, the execution proceeds as if no operation were specified. Results are not stored, and the condition code is not changed.

9.1.2 SOURCE IDENTIFICATION

The five classes of interruptions are distinguished by the storage locations in which the old PSW is stored and from which the new PSW is fetched. The detailed causes are further distinguished by the interruption code of the old PSW, except for the machine-check interruption. The bits of the interruption code are numbered 16-31, according to their position in the PSW.

For machine-check interruptions, additional information is provided by the diagnostic procedure, which is part of the interruption.

The following table lists the permanently allocated main-storage locations.

ADDRESS		LENGTH	PURPOSE
0	0000 0000	Double Word	Initial program loading PSW
8	0000 1000	Double Word	Unused
16	0001 0000	Double Word	Unused
24	0001 1000	Double Word	External old PSW
32	0010 0000	Double Word	Supervisor call old PSW
40	0010 1000	Double Word	Program old PSW
48	0011 0000	Double Word	Machine old PSW
56	0011 1000	Double Word	Input/output old PSW
64	0100 0000	Double Word	Buffered I/O status word
72	0100 1000	Word	Channel address word
76	0100 1100	Word	Unused
80	0101 0000	Word	Unused
84	0101 0100	Word	Unused
88	0101 1000	Double Word	External new PSW
96	0110 0000	Double Word	Supervisor call new PSW
104	0110 1000	Double Word	Program new PSW
112	0111 0000	Double Word	Machine-check new PSW
120	0111 1000	Double Word	Input/output new PSW
128	1000 0000	Word	Unused

9.1.3 LOCATION DETERMINATION

For some interruptions, it is desirable to locate the instruction being interpreted when the interruption occurred. Since the instruction address in the old PSW designates the instruction to be executed next, it is necessary to know the length of the preceding instruction. This length is recorded in bit positions 32 and 33 of the PSW as the instruction-length code.

The instruction-length code is predictable only for program and supervisor-call interruptions. For I/O and external interruptions, the interruption is not caused by the last-interpreted instruction, and the code is not predictable for these instructions. For machine-check interruptions, the setting of the code may be affected by the malfunction and, therefore, is unpredictable.

For the supervisor-call interruption, the instruction-length code is 1, indicating the halfword length of SUPERVISOR CALL. For program interruptions, the codes 1, 2, and 3 indicate the instruction length in halfwords. The following table shows the states of the instruction-length code.

ILC	PSW BITS 32-33	INSTRUCTION BITS 0-1	INSTRUCTION LENGTH	FORMAT
1	01	00	One halfword	RR
2	10	01	Two halfwords	RX
2	10	10	Two halfwords	RS or SI
3	11	11	Three halfwords	SS

Programming Notes

When a program interruption is due to incorrect branch address, the location determined from the instruction address and instruction-length code is the branch address and not the location of the branch instruction.

When an interruption occurs while the CPU is in the wait state, the instruction-length code is always unpredictable.

The instruction EXECUTE represents upon interruption an instruction-length code which does not reflect the length of the instruction executed, but is 2, the length of EXECUTE.

9.2 INPUT/OUTPUT INTERRUPTION

The I/O interruption provides a means by which the CPU responds to signals from I/O devices.

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved in the I/O section until accepted by the CPU. Priority is established among requests so that only one interruption request is processed at a time.

An I/O interruption can occur only after execution of the current instruction is completed and while the CPU is interruptible for I/O. I/O interrupts are masked by system mask bit 0. Interruptions masked off remain pending.

The I/O interruption causes the old PSW to be stored at location 56. Subsequently, a new PSW is loaded from location 120.

The interruption code in the old PSW identifies the device and status causing the interruption. The old PSW will contain the device dependent data. The instruction-length code is unpredictable.

9.3 PROGRAM INTERRUPTION

Exceptions resulting from improper specification or use of instructions and data cause a program interruption.

The current instruction is completed, terminated, or suppressed. Only one program interruption occurs for a given instruction and is identified in the old PSW. The occurrence of a program interruption does not preclude the simultaneous occurrence of other program-interruption causes. Which of several causes is identified may vary from one occasion to the next.

A program interruption can occur only when the corresponding mask bit, if any, is one. When the mask bit is zero, the interruption is ignored. Program interruptions do not remain pending. Program mask bit 36 permits masking of one of the 9 interruption causes.

The program interruption causes the old PSW to be stored at location 40 and a new PSW to be fetched from location 104.

The cause of the interruption is identified by the four low-order bit positions in the interruption code, PSW bits 28-31. The remainder of the interruption code, bits 16-27 of the PSW, are made zero. The instruction-length code indicates the length of the preceding instruction in halfwords. For a few cases, the instruction length is not available. These cases are indicated by code 0.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

A description of the individual program exceptions follows. The application of these rules to each class of instructions is further described in the applicable sections. Some of the exceptions listed may also occur in operations executed by I/O channels. In that event, the exception is indicated in the channel status word stored with the I/O interruption (as explained in paragraph 3.7.4, "Input/Output Operations").

9.3.1 OPERATION EXCEPTION

When an operation code is not assigned or the assigned operation is not available on the particular model, an operation exception is recognized. The operation is suppressed.

The instruction-length code is 1, 2, or 3.

9.3.2 PRIVILEGED-OPERATION EXCEPTION

When a privileged instruction is encountered in the problem state, a privileged-operation exception is recognized. The operation is suppressed.

The instruction-length code is 1 or 2.

9.3.3 EXECUTE EXCEPTION

When the subject instruction of EXECUTE is another EXECUTE, an execute exception is recognized. The operation is suppressed.

The instruction-length code is 2.

9.3.4 PROTECTION EXCEPTION

When an instruction tries to store into a protected location a protection exception is recognized.

The operation is suppressed on a store violation, except in the case of STORE MULTIPLE, TEST AND SET, and variable-length operations, which are terminated.

The instruction-length code is 0, 2, or 3.

9.3.5 ADDRESSING EXCEPTION

When an address specifies any part of data, an instruction, or a control word outside the available storage for the particular installation, an addressing exception is recognized.

In most cases, the operation is terminated for an invalid data address. Data in storage remain unchanged, except when designated by valid addresses. In a few cases, an involved data address causes the instruction to be suppressed - AND (NI), EXCLUSIVE OR (XI), OR (OI), MOVE (MVI), CONVERT

TO DECIMAL, DIAGNOSE, EXECUTE, and certain store operations (ST, STC, and STH). The operation is suppressed for an invalid instruction address.

The instruction-length code normally is 1, 2, or 3, but may be 0 in the case of a data address.

9.3.6 SPECIFICATION EXCEPTION

A specification exception is recognized when:

1. A data, instruction, or control-word address does not specify an integral boundary for the unit of information.

2. The R_1 field of an instruction specifies an odd register address for a pair of general registers that contains a 64-bit operand.

3. A PSW with nonzero bits 8-11 is encountered.

The operation is suppressed. The instruction-length code is 1, 2, or 3.

9.3.7 DATA EXCEPTION

A data exception is recognized when:

1. The sign or digit code of operands editing operations or in CONVERT TO BINARY are incorrect.

The operation is terminated. The instruction-length code is 2 or 3.

9.3.8 FIXED-POINT-OVERFLOW EXCEPTION

When a high-order carry occurs or high-order significant bits are lost in fixed-point add, subtract, shift, or sign-control operations, a fixed-point-overflow exception is recognized.

The operation is completed by ignoring the information placed outside the register. The interruption may be masked by PSW bit 36.

The instruction-length code is 1 or 2.

9.3.9 FIXED-POINT-DIVIDE EXCEPTION

A fixed-point-divide exception is recognized when a quotient exceeds the register size in fixed-point division, including division by zero, or the result of CONVERT TO BINARY exceeds 31 bits.

Division is suppressed. Conversion is completed by ignoring the information placed outside the register.

The instruction-length code is 1 or 2.

9.3.10 EXPONENT-OVERFLOW EXCEPTION

When the result characteristic in floating-point addition, subtraction, multiplication, or division exceeds 127 and the result fraction is not zero, an exponent-overflow exception is recognized. The operation is completed. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic.

The instruction-length code is 1 or 2.

9.3.11 EXPONENT-UNDERFLOW EXCEPTION

When the result characteristic in floating-point addition, subtraction, multiplication, halving, or division is less than zero and the result fraction is not zero, an exponent-underflow exception is recognized. The operation is completed.

The setting of the exponent-underflow mask (PSW bit 38) affects the results of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

The instruction-length code is 1 or 2.

9.3.12 SIGNIFICANCE EXCEPTION

When the result of a floating-point addition or subtraction has an all-zero fraction, a significance exception is recognized.

The operation is completed. The interruption may be masked by PSW bit 39. The manner in which the operation is completed is determined by the mask bit.

The instruction-length code is 1 or 2.

9.3.13 FLOATING-POINT-DIVIDE EXCEPTION

When division by a floating-point number with zero fraction is attempted, a floating-point divide exception is recognized. The operation is suppressed.

The instruction-length code is 1 or 2.

9.3.14 BUFFERED I/O EXCEPTION

If any of the addressing exceptions or parity errors occur during a Buffered I/O operation, an Exception Program Interruption will be generated in the same manner as any exception. However, the Buffered I/O status word (Loc 66-67) will be set non-zero and will contain the address of the Buffered I/O word in use at the time the error occurred.

9.3.15 SUPERVISOR-CALL INTERRUPTION

The supervisor-call interruption occurs as a result of the execution of SUPERVISOR CALL.

The supervisor-call interruption causes the old PSW to be stored at location 32 and a new PSW to be fetched from location 96.

The contents of bit positions 8-15 of the SUPERVISOR CALL become bits 24-31 in the interruption code of the old PSW. PSW bit positions 16-23 in the old PSW are made zero. The instruction-length code is 1, indicating the halfword length of SUPERVISOR CALL.

Programming Notes

The name "supervisor call" indicates that one of the major purposes of the interruption is the switching from problem to supervisor state. This major purpose does not preclude the use of this interruption for other types of status switching.

The interruption code may be used to convey a message from the calling program to the supervisor.

When SUPERVISOR CALL is performed as the subject instruction of EXECUTE, the instruction-length code is 2.

9.4 EXTERNAL INTERRUPTION

The external interruption provides a means by which the CPU responds to signals from the timer, from the interrupt key, and from external units.

A request for an external interruption may occur at any time, and requests from different sources may occur at the same time. Requests are preserved until honored by the CPU. Each request is presented only once. When several requests from one source are made before the interruption is taken, only one interruption occurs.

An external interruption from the interval timer can occur only when system mask bit 7 is one and after execution of the current instruction is completed. An external interruption from the TSE external interrupt key can occur only when system mask bit 0 is one. The interruption causes the old PSW to be stored at location 24 and a new PSW to be fetched from location 88.

The source of the interruption is identified in bit positions 24-31 of the old PSW. The remainder of the interruption code, PSW bits 16-23, is made zero. The instruction-length code is unpredictable for external interruptions.

9.4.1 TIMER

A timer value changing from positive to negative causes an external interruption with PSW bit 24 set to one.

9.4.2 INTERRUPT KEY

Pressing the interrupt key on the operator control section of the TSE control panel causes an external interruption with PSW bit 25 set to one.

The key is active while power is on.

9.4.3 INTERVAL TIMER

The interval timer is a 16 bit decrementing counter that contains both hardware and microprogrammed elements.

The interval timer may be read by using the TMRS (opcode A4) instruction with the R_1 field set to 01. It may be set by using the TMRS instruction with the R_1 field set to 03. The interval timer is treated as a 16 bit absolute value (unsigned integer), that is counted down one bit for each decrementing pulse.

The interval timer may be set to any value between Hex 1 to FFFF.

The Real Time Clock is a 32 bit incrementing counter that contains both microprogrammed and hardware elements.

The Real Time Clock may be read by using the TMRS instruction (opcode A4) with the R_1 field set to zero. It may be set by using the TMRS instruction with the R_1 field set to 02.

Timing Pulse	Occurs Every 110 µsec
Clock Low Order Bit	Incremented by Timing Pulse ÷ 10 = 112.64 µsec
Interval Timer Low Order Bit	Decremented by Timing Pulse ÷ 10 = 112.64 µsec
Max Value of Interval Timer (16 bits)	7.3819 Seconds
Max Value of Real Time Clock (32 bits)	5 days 14 hours 23 min. 5.1162 Seconds

9.5 MACHINE-CHECK INTERRUPTION

The machine-check interruption provides a means for recovery from and fault location of machine malfunction.

The old PSW is stored at location 48 and the new PSW is fetched from location 112. Bit 40 of the old PSW will be on and will indicate that a parity error has occurred. Any recovery procedure should be aware that bit 40 should not be included as part of the instruction address.

SECTION X

SHORT PRECISION OPTION

The short precision instruction set performs binary arithmetic on operands serving as addresses, index quantities, and counts, as well as fixed-point data. In general, both operands are signed and 16 bits long. Negative quantities are held in two's-complement form. One operand is always in one of the 16 general registers; the other operand may be in main storage or in a general register.

The instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, sign control, and shifting of fixed-point operands.

The condition code is set as a result of all sign-control, add, subtract, compare, and arithmetic shift operations.

10.1 DATA FORMAT

Short precision fixed-point numbers occupy a fixed-length format consisting of a one-bit sign followed by the integer field consisting of 15 bits. When held in a general register, a short precision quantity occupies the rightmost 16 bits (16-31). Unless otherwise stated, the leftmost 16 bits (0-15) are neither tested nor altered. In register-to-register operations the same register may be specified for both operand locations.

SHORT PRECISION FIXED-POINT NUMBER IN MAIN STORAGE

SHORT PRECISION FIXED-POINT NUMBER IN A GENERAL REGISTER

Short precision data in main storage occupy a 16-bit halfword, with a binary integer field of 15 bits. These data must be located on integral storage boundaries for these units of information, that is, halfword operands must be addressed with the last low-order address bit set to zero.

In all discussions of fixed-point numbers in this section, the expression "16-bit signed integer" denotes a 15-bit integer with a sign bit.

10.2 NUMBER REPRESENTATION

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in two's-complement notation with a one in the sign bit. The two's-complement representation of a negative number may be considered the sum of the integer part of the field, taken as a positive number, and the maximum negative number. The two's-complement of a number is obtained by inverting each bit of the number and adding a one in the low-order bit position.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number, including the sign bit, are zeros. When the number is negative, all these bits, including the sign bit, are ones. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by prefixing a field in which each bit is set equal to the high-order bit of the operand.

Two's-complement notation does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit sign.

The CPU cannot represent the complement of the maximum negative number. When an operation, such as a subtraction from zero, produces the complement of the maximum negative number, the number remains unchanged, and a fixed-point overflow exception is recognized. An overflow does not result, however, when the number is complemented and the final result is within the representable range. An example of this case is a subtraction from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

The sign bit is leftmost in a number. In an arithmetic operation, a carry out of the integer field changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order bits are shifted out of the integer field.

10.3 CONDITION CODE

The results of fixed-point sign-control, add, subtract, compare, and shift operations are used to set the condition code in the program status word

(PSW). All other short precision operations leave this code undisturbed. The condition code can be used for decision-making by subseqeunt branch-on-condition instructions.

The condition code can be set to reflect three types of results for short precision fixed-point arithmetic. For most operations, the states 0, 1, or 2 indicate a zero, less than zero, or greater than zero content of the result register, while the state 3 is used when the result overflows.

For a comparison, the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

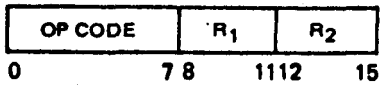
CONDITION CODE SETTINGS FOR FIXED-POINT ARITHMETIC

	0	1	2	3
Add Half/Short	zero	<zero	>zero	overflow
Compare Short	equal	low	high	--
Load and Test	zero	<zero	>zero	--
Load and Test Short	zero	<zero	>zero	--
Load Complement Short	zero	<zero	>zero	overflow
Load Negative Short	zero	<zero	--	--
Load Positive Short	zero	--	>zero	overflow
Normalize	zero	<zero	>zero	--
Shift Left Short	zero	<zero	>zero	overflow
Shift Right Short	zero	<zero	>zero	--
Subtract Half/Short	zero	<zero	>zero	overflow
Test Bits	zero	mixed	--	ones

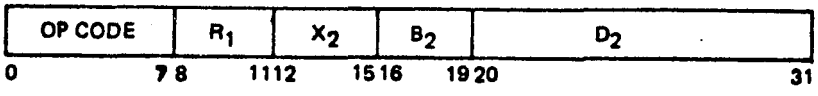
10.4 INSTRUCTION FORMAT

Fixed-point instructions use the following four formats:

RR FORMAT



RX FORMAT



NOTE:

In this document, instructions with halfword second operands (16 bits) propagate the sign bit to form a 32-bit number before combining it with a 32-bit first operand. Short operand instructions differ from halfword operand instructions in that both short operands are 16 bits in length and only the rightmost 16 bits of register operands are altered and/or tested. Short operands are never expanded to 32 bits. (The short operand multiply and divide instructions work with a 32-bit product, dividend, and quotient.)

10.5 INSTRUCTIONS

The short precision instructions and their mnemonics, formats, and operation codes are listed in the following table. The table also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

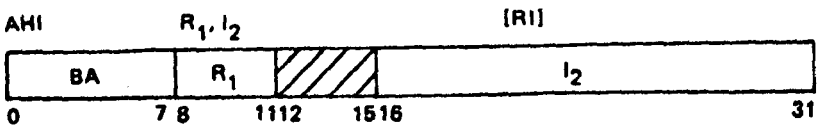
NAME	MNEMONIC	TYPE		EXCEPTIONS		CODE
Add Halfword Immediate	AHI	RI	C		IF	BA
Add Short	AS	RX	C	A,S	IF	53
Add Short Immediate	ASI	RI	C		IF	AA
Add Short Register	ASR	RR	C		IF	CA
Branch Unconditional	BU	RX				73
Branch Unconditional Register	BUR	RR				CE
Compare Halfword Immediate	CHI	RI	C			B9
Compare Logical Short	CLS	RX	C	A,S		65
Compare Logical Short Immediate	CLSI	RI	C			B5
Compare Logical Short Register	CLSR	RR	C			C5
Compare Short	CS	RX	C	A,S		61
Compare Short Immediate	CSI	RI	C			A9
Compare Short Register	CSR	RR	C			C9
Divide Short	DS	RX		A,S	IK	4D
Divide Short Immediate	DSI	RI			IK	B0
Divide Short Register	DSR	RR			IK	CD
Load Address Short	LAS	RX				51
Load Complement Short Register	LCSR	RR	C		IF	C3
Load Full to Short Register	LFSR	RR	C		IF	0B
Load Halfword Immediate	LHI	RI				B8

NAME	MNEMONIC	TYPE		EXCEPTIONS		CODE
Load Halfword Register	LHR	RR				D0
Load Negative Short Register	LNSR	RR	C			C1
Load Positive Short Register	LPSR	RR	C		IF	C0
Load Short	LS	RX		A,S		74
Load Short Immediate	LSI	RI				A8
Load Short Register	LSR	RR				C8
Load and Test	LT	RX	C	A,S		62
Load and Test Short	LTS	RX	C	A,S		52
Load and Test Short Register	LTSR	RR	C			C2
Multiply Halfword Immediate	MHI	RI				BC
Multiply Short	MS	RX		A,S		71
Multiply Short Immediate	MSI	RI				B3
Multiply Short Register	MSR	RR				CC
Normalize	NRM	RR	C			CF
AND Short	NS	RX	C	A,S		64
AND Short Immediate	NSI	RI	C			B4
AND Short Register	NSR	RR	C			C4
OR Short	OS	RX	C	A,S		66
OR Short Immediate	OSI	RI	C			A6
OR Short Register	OSR	RR	C			C6
Shift Left Arithmetic Short	SLAS	RS	C		IF	A3
Shift Left Logical Short	SLLS	RS				A1
Shift Right Arithmetic Short	SRAS	RS	C			A2
Shift Right Logical Short	SRLS	RS				A0
Subtract Halfword Immediate	SHI	RI	C		IF	BB
Subtract Short	SS	RX	C	A,S	IF	72
Subtract Short Immediate	SSI	RI	C		IF	AB
Subtract Short Register	SSR	RR	C		IF	CB
Test Bits	TB	RX	C	A,S		75
Test Bits Immediate	TBI	RI	C			AE
Exclusive OR Short	XS	RX	C	A,S		63
Exclusive OR Short Immediate	XSI	RI	C			A7
Exclusive OR Short Register	XSR	RR	C			C7

NOTES:

A	Addressing exception
B	Condition code is set
IF	Fixed-Point overflow exception
IK	Fixed-Point divide exception
S	Specification exception

10.5.1 ADD HALFWORD



The halfword second operand is added to the first operand and the sum is placed in the first operand location.

The halfword second operand is expanded to a fullword before the addition by propagating the sign-bit value through the 16 high-order bit positions.

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

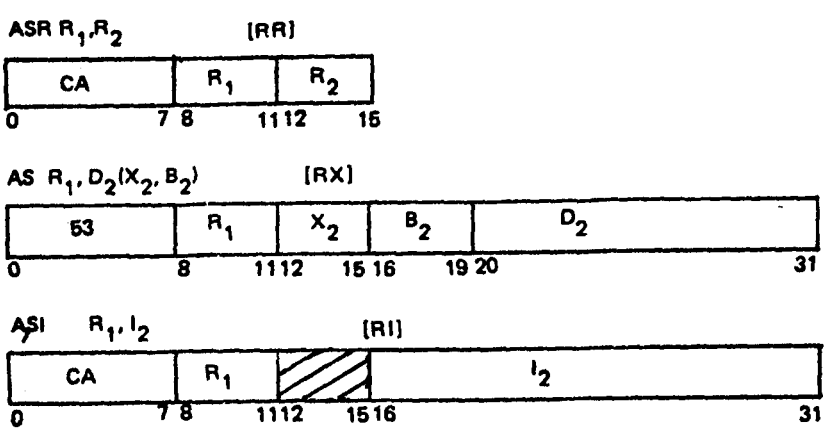
Resulting Condition Code:

0	Sum is zero
1	Sum is less than zero
2	Sum is greater than zero
3	Overflow

Program Interruptions:

Fixed-point overflow

10.5.2 ADD SHORT



The second operand is added to the first operand, and the sum is placed in the first operand location.

Addition is performed by adding 16 bits of both operands. If the carry out of the sign-bit position 16 and the carry out of the high-order numeric bit position 17 agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

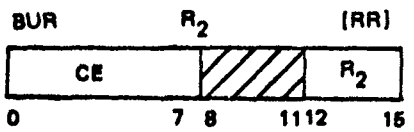
Program Interruptions:

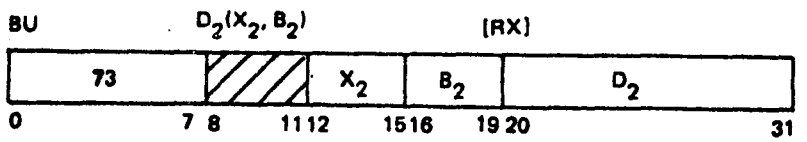
Addressing (AS only)
Specification (AS only)
Fixed-point overflow

Programming Note

In two's-complement notation, a zero result is always positive.

10.5.3 BRANCH UNCONDITIONAL





The updated instruction address is unconditionally replaced by the branch address. This instruction is almost functionally equivalent to a Branch On Condition instruction with a mask of all ones, but this instruction

executes about 40% faster than the Branch On Condition. Unlike the Branch On Condition, this instruction will branch when the R_2 field in the RR format contains zero.

Condition Code: The code remains unchanged.

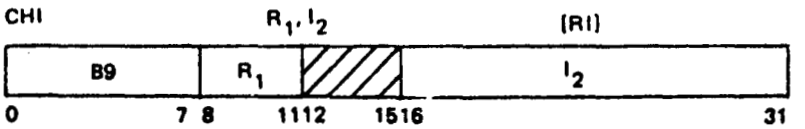
Program Interruptions:

None

Programming Note

The NSSC-II assembler OPSYN statement may be used to force the extended mnemonics B and BR to generate the Branch Unconditional opcodes.

10.5.4 COMPARE HALFWORD



The first operand is compared with the halfword second operand, and the result determines the setting of the condition code.

The halfword second operand is expanded to a fullword before the comparison by propagating the sign-bit value through the 16 high-order bit positions.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

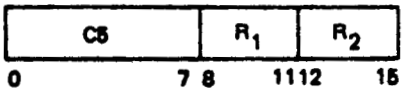
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

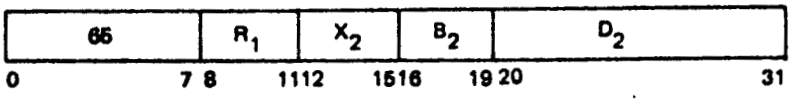
None

10.5.5 COMPARE LOGICAL SHORT

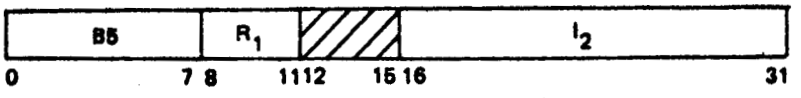
CLSR R_1,R_2 [RR]



CLS R_1,D_2(X_2,B_2) [RX]



CLSI R_1,I_2 [RI]



The first operand is compared with the second operand, and the result is indicated in the condition code.

The instructions allow comparisons that are register to register, storage to register, and register to instruction.

Comparison is binary, and all codes are valid.

Resulting Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

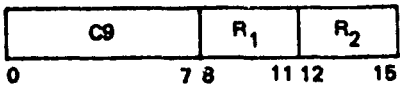
Addressing (CLS only)
Specification (CLS only)

Programming Note

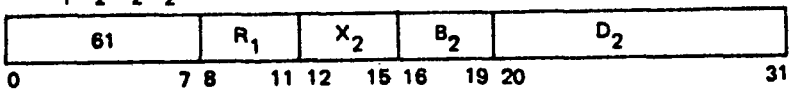
The COMPARE LOGICAL is unique in treating all bits alike as part of an unsigned binary quantity.

10.5.6 COMPARE SHORT

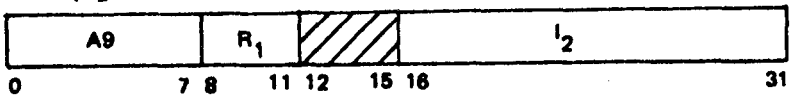
CSR R_1,R_2 [RR]



CS $R_1,D_2(X_2,B_2)$ [RX]



CSI R_1,I_2 [RI]



The first operand is compared with the second operand, and the result determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 16-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

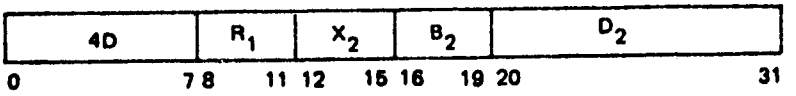
Program Interruptions:

Addressing (CS only)
Specification (CS only)

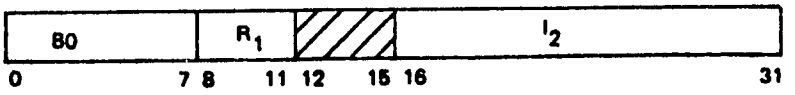
10.5.7 DIVIDE SHORT

DSR R_1,R_2 [RR]

DS R_1,D_2(X_2,B_2) [RX]



DSI R_1,I_2 [RI]



The dividend (first operand) is divided by the divisor (second operand) and replaced by the quotient. A remainder is not developed.

The dividend is a 32-bit signed integer and occupies the register specified by the R_1 field of the instruction. A 32-bit signed quotient replaces the dividend. The divisor is a 16-bit signed integer.

The sign of the quotient is determined by the rules of algebra. All operands and results are treated as signed integers. When the relative magnitude of dividend and divisor is such that the quotient cannot be expressed by a 16-bit signed integer, a fixed-point divide exception is recognized (a program interruption occurs, no division takes place, and the dividend remains unchanged in the general registers).

Condition Code: The code remains unchanged.

Program Interruptions:

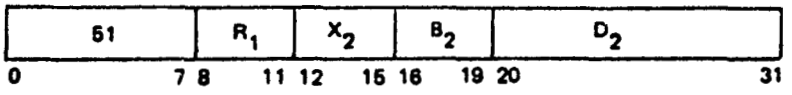
Addressing (DS only)
Specification (DS only)
Fixed-point divide

Programming Note

Divide short develops a 16-bit signed quotient. The sign bit is then propagated to create the 32-bit signed quotient.

10.5.8 LOAD ADDRESS SHORT

LAS $R_1, D_2(X_2, B_2)$ [RX]



The address of the second operand is inserted in the low-order 16 bits of the general register specified by R_1. The remaining bits of the general register are not altered. No storage references for operands take place.

The address specified by the X_2, B_2, and D_2 fields is inserted in bits 16-31 of the general register specified by R_1. Bits 0-15 are not changed. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 16th bit are ignored.

Condition Code: The code remains unchanged.

Program Interruptions:

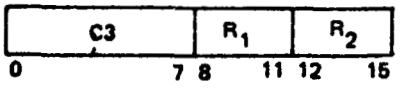
None

Programming Note

The same general register may be specified by R_1, X_2, and B_2 instruction field, except that general register 0 can be specified only by the R_1 field. In this manner, it is possible to increment the low-order 16 bits of a general register, other than 0, by the contents of the D_2 field of the instruction. The register to be incremented should be specified by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero).

10.5.9 LOAD COMPLEMENT SHORT

LCSR R_1, R_2 [RR]



The two's-complement of the second operand is placed in the first operand location.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one. Only bits 16-31 of both registers participate.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow

Programming Note

Zero remains invariant under complementation.

10.5.10 LOAD FULL TO SHORT REGISTER

LFSR R_1,R_2 [RR]

The second operand is placed in the first operand location; and the sign and magnitude of the second operand determines the condition code. The second operand is not changed.

The second operand is a 32-bit signed integer and the first operand is a 16-bit signed integer. An overflow condition occurs when the second operand is too large to be contained by the first operand; the left truncated second operand is placed in the first operand location regardless of the overflow condition. The overflow causes a program interruption when the fixed-point overflow mask is one.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero

2 Result is greater than zero
3 Overflow

Program Interruptions:

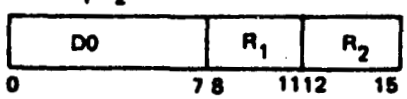
Fixed-point overflow

Programming Note

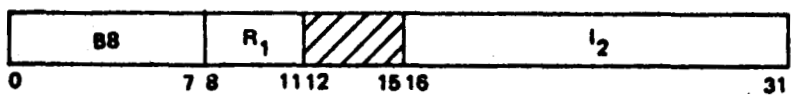
When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

10.5.11 LOAD HALFWORD

LHR R_1,R_2 [RR]



LHI R_1,I_2 [RI]



The halfword second operand is placed in the first operand location.

The halfword second operand is expanded to a fullword by propagating the sign-bit value through the 16 high-order bit positions. Expansion occurs after the operand is obtained and before insertion in the register.

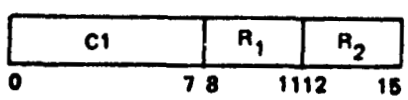
Condition Code: The code remains unchanged.

Program Interruptions:

None

10.5.12 LOAD NEGATIVE SHORT

LNSR R_1,R_2 [RR]



The two's-complement of the absolute value of the second operand is placed in the first operand location. The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged with positive sign. Only bits 16-31 of both registers participate.

Resulting Condition Code:

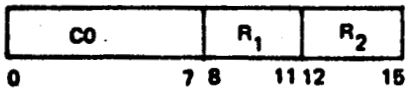
0 Result is zero
1 Result is less than zero
2 --
3 --

Program Interruptions:

None

10.5.13 LOAD POSITIVE SHORT

LPSR R_1, R_2 (RR)



The absolute value of the second operand is placed in the first operand location.

The operation includes complementation of negative numbers; positive numbers remain unchanged.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one. Only bits 16-31 of both registers participate.

Resulting Condition Code:

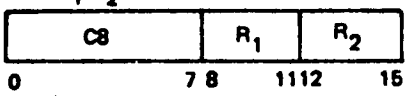
0 Result is zero
1 --
2 Result if greater than zero
3 Overflow

Program Interruptions:

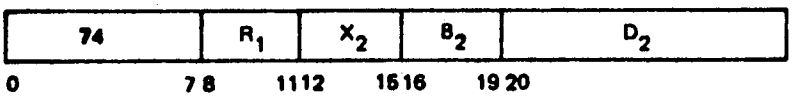
Fixed-point overflow

10.5.14 LOAD SHORT

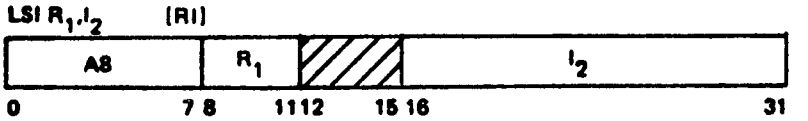
LSR R_1,R_2 [RR]



LS R_1, $D_2(X_2, B_2)$ [RX]



LSI R_1,I_2 [RI]



The second operand is placed in the first operand location. The second operand is not changed. Only bits 16-31 of the first operand register are loaded. Bits 0-15 remain unchanged.

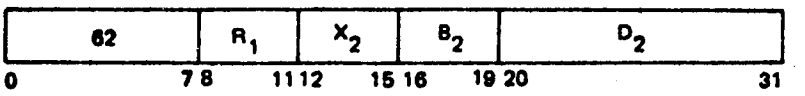
Condition Code: The code remains unchanged.

Program Interruptions:

Addressing (LS only)
Specification (LS only)

10.5.15 LOAD AND TEST

LT R_1, $D_2(X_2, B_2)$ [RX]



The second operand is placed in the first operand location, and the sign and magnitude of the second operand determine the condition code. The second operand is not changed. Both operands are 32 bits in length.

Resulting Condition Code:

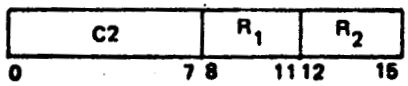
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

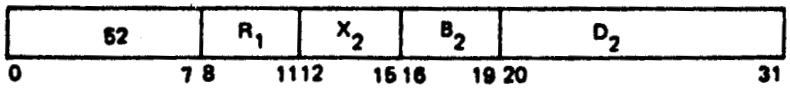
Addressing
Specification

10.5.16 LOAD AND TEST SHORT

LTSR R_1, R_2 [RR]



LTS $R_1, D_2(X_2, B_2)$



The second operand is placed in the first operand location, and the sign and magnitude of the second operand determine the condition code. The second operand is not changed. Only bits 16-31 of R_1 are loaded and tested.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

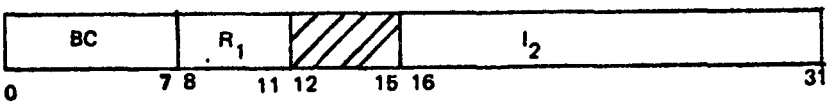
Addressing (LTS only)
Specification (LTS only)

Programming Note

When LTS is used with the same register specified as first and second operand location, the operation is equivalent to a test without data movement.

10.5.17 MULTIPLY HALFWORD

MHI R_1, I_2 [RI]



The product of the halfword multiplier (second operand) and multiplicand (first operand) replaces the multiplicand.

Both multiplicand and product are 32-bit signed integers and may be located in any general register. The halfword multiplier is expanded to a full-word before multiplication by propagating the sign-bit value through the 16 high-order bit positions. The multiplicand is replaced by the low-order part of the product. The bits to the left of the 32 low-order bits are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Interruptions:

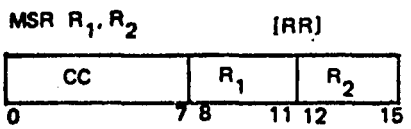
None

Programming Note

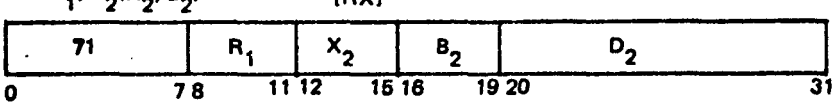
The significant part of the product usually occupies 46 bits or fewer, the exception being 47 bits when both operands are maximum nagative. Since the low-order 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow.

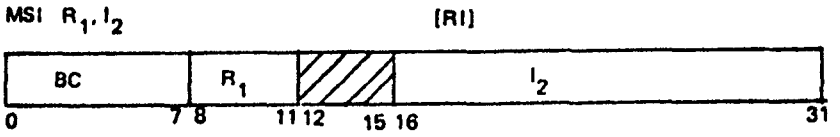
10.5.18 MULTIPLY SHORT

MSR R_1, R_2 [RR]



MS $R_1, D_2(X_2, B_2)$ [RX]





The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

Both multiplier and multiplicand are 16-bit signed integers. The product is always a 32-bit signed integer. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

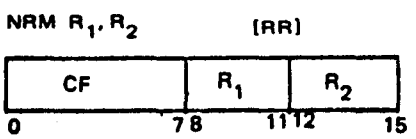
Program Interruptions:

Addressing (MS only)
Specification (MS only)

Programming Note

The significant part of the product usually occupies 30 bits or fewer. Only when two maximum negative numbers are multiplied are 31 significant product bits formed. Since two's-complement notation is used, the sign bit is extended right until the first significant product digit is encountered.

10.5.19 NORMALIZE



The 32 bits in the register specified by R_1 are shifted arithmetically left until bit 0 is not equal to bit 1. The number of shifted bit positions is then placed into the 32-bit register specified by R_2.

If the first operand is already normalized, no shifting takes place and a zero is placed into the second operand. If the first operand is all zero, it is considered to be normalized.

If the first and second operand are in the same register, the first operand will be normalized and the shift count will be lost.

Resulting Condition Code:

0	Result is zero
1	Result is negative
2	Result is positive
3	-

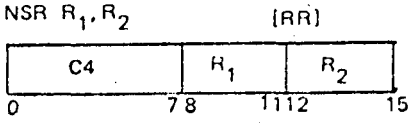
Program Interruptions:

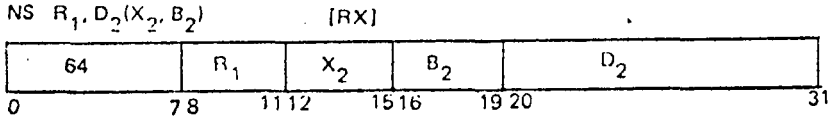
None

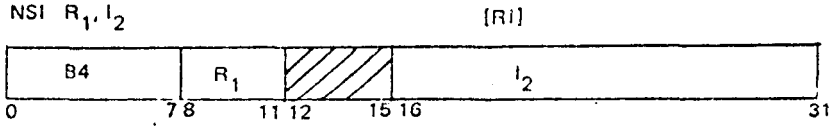
Programming Note

The maximum shift count for a positive number is 30 and for a negative number is 31.

10.5.20 AND SHORT







The logical product (AND) of the bits of the first and second operand is placed in the first operand location. Both operands are 16 bits in length.

Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set

to one if the corresponding bit positions in both operands contain a one; otherwise, the result bit is set to zero. All operands and results are valid.

Resulting Condition Code:

0	Result is zero
1	Result not zero
2	--
3	--

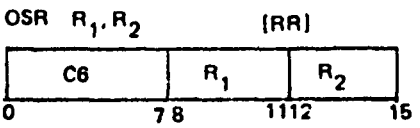
Program Interruptions:

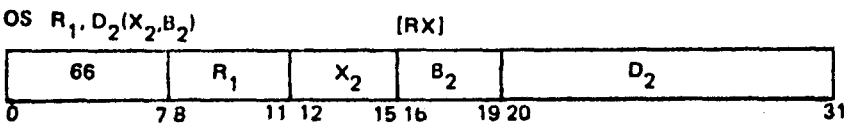
Addressing (NS only)
Specification (NS only)

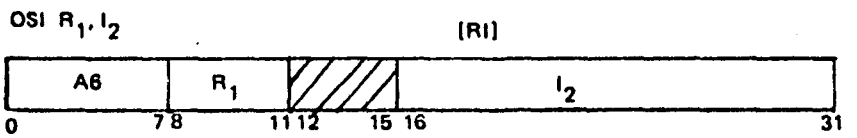
Programming Note

The AND may be used to set a bit to zero.

10.5.21 OR SHORT







The logical sum (OR) of the bits of the first and second operand is placed in the first operand location. Both operands are 16 bits in length.

Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero. All operands and results are valid.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

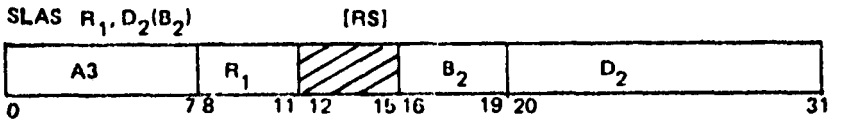
Program Interruptions:

Addressing (OS only)
Specification (OS only)

Programming Note

The OR may be used to set a bit to one.

10.5.22 SHIFT LEFT ARITHMETIC SHORT



The integer part of the first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 15 integer bits of the operand participate in the left shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit position 16 is shifted out of position 17, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow

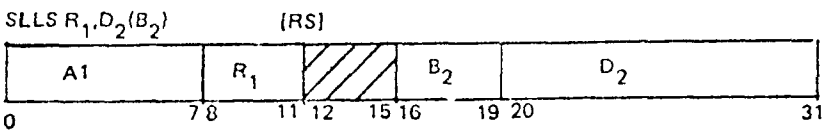
Programming Note

For numbers with an absolute value of less than 2^{14}, a left shift of one bit position is equivalent to multiplying the number by 2.

Shift amounts from 15-63 cause the entire integer to be shifted out of the right half of the register. When the entire integer field for a positive number has been shifted out, the half register contains a value of zero. For a negative number, the half register contains a value of -2^{15}.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

10.5.23 SHIFT LEFT LOGICAL SHORT



The first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

Only bits 16-31 of the general register specified by R_1 participate in the shift. High-order bits are shifted out from bit position 16 without inspection and are lost. Zeros are supplied to the vacated low-order register positions. As in all short operand instructions, bit positions 0-15 of the first operand are neither inspected nor changed.

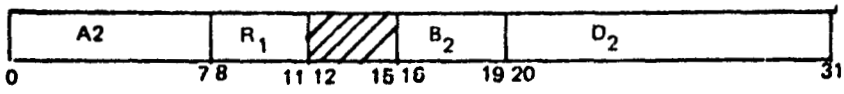
Condition Code: The code remains unchanged.

Program Interruptions:

None

10.5.24 SHIFT RIGHT ARITHMETIC SHORT

SRAS $R_1, D_2(B_2)$ (RS)



The integer part of the first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 15 integer bits of the operand participate in the right shift. Bits equal to the sign bit position 16 are supplied to the vacated high-order bit position 17. Low-order bits are shifted out without inspection and are lost.

Resulting Condition Code:

0	Result is zero
1	Result is less than zero
2	Result is greater than zero
3	--

Program Interruptions:

None

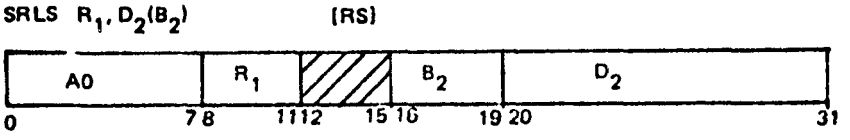
Programming Note

A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position, the value of the field is that obtained by dividing the next lower number by 2. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.

Shift amounts from 15-63 cause the entire integer to be shifted out of the right half of the register. When the entire integer field of a positive number has been shifted out, the half register contains a value of zero. For a negative number, the half register contains a value of -1.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

10.5.25 SHIFT RIGHT LOGICAL SHORT



The first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

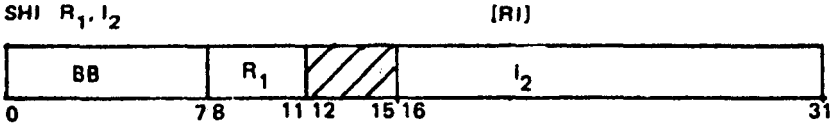
Only bits 16-31 of the general register specified by R_1 participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated high-order register bit position 16. As in all short operand instructions, bits 0-15 of the first operand are neither inspected nor altered.

Condition Code: The code remains unchanged.

Program Interruptions:

None

10.5.26 SUBTRACT HALFWORD



The halfword second operand is subtracted from the first operand, and the difference is placed in the first operand location.

The halfword second operand is expanded to a fullword before the subtraction by propagating the sign-bit value through 16 high-order bit positions.

Subtraction is considered to be performed by adding the one's complement of the expanded second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow is recognized. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

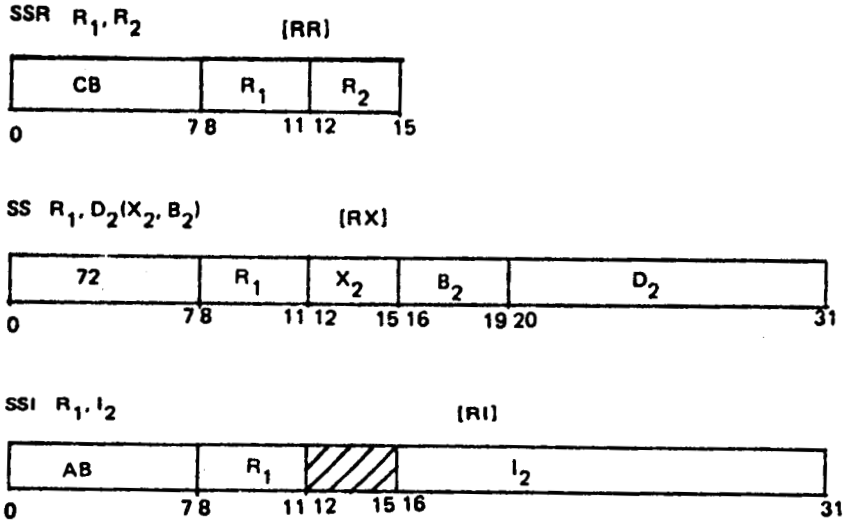
Resulting Condition Code:

0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow

10.5.27 SUBTRACT SHORT



The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is considered to be performed by adding the one's-complement of the second operand and a low-order one to the first operand. All 16 bits of both operands participate, as in ADD. If the carry out of the

sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow is recognized. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Interruptions:

Addressing (SS only)
Specifications (SS only)
Fixed-point overflow

Programming Note

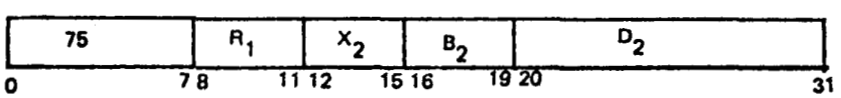
The use of the one's-complement and the low-order one instead of the two's complement of the second operand is necessary for proper recognition of overflow.

When the same register is specified as first and second operand location, subtracting is equivalent to clearing the register.

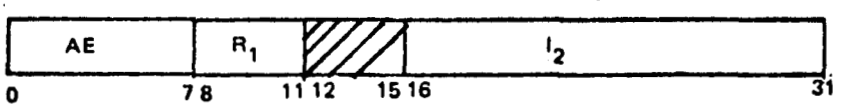
Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

10.5.28 TEST BITS

TB $R_1, D_2(X_2, B_2)$ (RX)



TBI R_1, I_2 (RI)



The state of the first operand bits selected by a mask is used to set the condition code.

The second operand is used as a 16-bit mask. The bits of the mask are made to correspond one for one with bits 16-31 of the register specified by the first operand.

A mask bit of one indicates that the first operand bit is to be tested. When the mask bit is zero, the first operand bit is ignored. When all first operand bits thus selected are zero, the condition code is made 0. The code is also made 0 when the mask is all-zero. When the selected bits are all-one, the code is made 3; otherwise, the code is made 1. The first operand is not changed.

Resulting Condition Code:

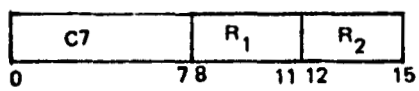
0	Selected bits all-zero; mask if all-zero
1	Selected bits mixed zero and one
2	--
3	Selected bits all-one

Program Interruptions:

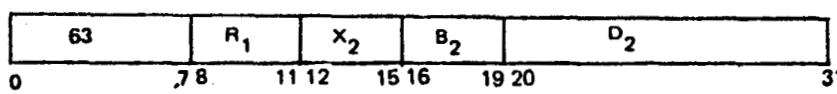
Addressing (TB only)
Specification (TB only)

10.5.29 EXCLUSIVE OR SHORT

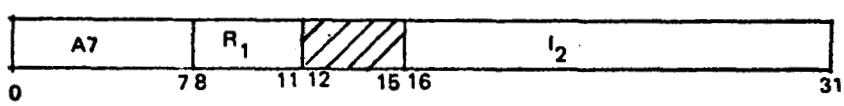
XSR R_1, R_2 [RR]



XS $R_1, D_2(X_2, B_2)$ [RX]



XSI R_1, I_2 [RI]



The modulo-two sum (exclusive OR) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective exclusive OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

The instruction differs from AND and OR only in the connective applied.

Resulting Condition Code:

0	Result is zero
1	Result not zero
2	--
3	--

Program Interruptions:

Addressing (XS only)
Specification (XS only)

Programming Note

The exclusive OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

Any field exclusive ORed with itself becomes all zeros.

The sequence A exclusive ORed B, B exclusive ORed A, A exclusive ORed B results in the exchange of the contents of A and B without the use of an auxiliary buffer area.

10.6 SHORT PRECISION EXCEPTIONS

Exceptional operand designations, data, or results cause a program interruption. When a program interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in fixed-point arithmetic.

Addressing

An address designates an operand location outside the available storage for a particular installation. In most cases, the operation is terminated. Therefore, the result data are unpredictable and should not be used for further computation. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the components from which an address is generated - the content of the D_2 field and the contents of the registers specified by X_2 and B_2.

Specification

A halfword operand is not located on a 16-bit boundary.

The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Fixed-Point Overflow

The result of a sign-control, add, subtract, or shift operation overflows. The interruption occurs only when the fixed-point overflow mask bit is one. The operation is completed by placing the truncated low-order result in the register and setting the condition code to 3. The overflow bits are lost. In add-type operations the sign stored in the register is the opposite of the sign of the sum or difference. In shift operations the sign of the shifted number remains unchanged. The state of the mask bit does not affect the result.

Fixed-Point Divide

The quotient of a division exceeds 16 bits including division by zero. Division is suppressed. Therefore, data in the register remain unchanged.

SECTION XI

DOUBLE PRECISION FIXED-POINT ARITHMETIC OPTION

The double precision fixed-point instruction set performs binary arithmetic on fixed-point data where both operands are signed and 64 bits long. Negative quantities are held in two's-complement form. One operand is always in a pair of the 16 general registers; the other operand may be in main storage or in a general register pair.

The instruction set provides for loading, adding, subtracting, comparing, complementing, and storing.

The condition code is set as a result of all add, subtract, complement, and compare operations.

11.1 DATA FORMAT

Fixed-point numbers occupy a fixed-length format consisting of a one-bit sign followed by the integer field. When held in a pair of general registers, a fixed-point quantity has a 63-bit integer field and occupies all 64 bits of the register pair. These operands are located in a pair of adjacent general registers and are addressed by an even address referring to the left-most register of the pair. The sign-bit position of the rightmost register contains part of the integer. In register-to-register operations the same register may be specified for both operand locations.

DOUBLE PRECISION FIXED-POINT NUMBER

Double precision fixed-point data in main storage occupy a 64-bit word with a binary integer field of 63 bits. These data must be located on integral storage boundaries for fullword units of information, that is, double word operands must be addressed with two low-order address bits set to zero.

In all discussions of fixed-point numbers in this publication, the expression "64-bit signed integer" denotes a 63-bit integer with a sign bit.

11.2 NUMBER REPRESENTATION

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in two's-complement notation with a one in the sign bit. The two's-complement representation of a negative number may be considered the sum of the integer part of the field, taken as a

positive number, and the maximum negative number. The two's complement of a number is obtained by inverting each bit of the number and adding a one in the low-order bit position.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number, including the sign bit, are zeros. When the number is negative, all these bits, including the sign bit, are ones. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by prefixing a field in which each bit is set equal to the high-order bit of the operand.

Two's-complement notation does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit for sign.

The CPU cannot represent the complement of the maximum negative number. When an operation, such as a subtraction from zero, produces the complement of the maximum negative number, the number remains unchanged, and a fixed-point overflow exception is recognized. An overflow does not result, however, when the number is complemented and the final result is within the representable range. An example of this case is a subtraction from minus one.

The sign bit is leftmost in a number. In an arithmetic operation, a carry out of the integer field changes the sign.

11.3 CONDITION CODE

The results of fixed-point sign-control, add, subtract, and compare operations are used to set the condition code in the program status word (PSW). All other double precision fixed-point operations leave this code undisturbed. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect three types of results for fixed-point arithmetic. For most operations, the states 0, 1, or 2 indicate a zero, less than zero, or greater than zero content of the result register, while the state 3 is used when the result overflows.

For a comparison, the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

Condition Code Settings for Double Precision Fixed-Point Arithmetic

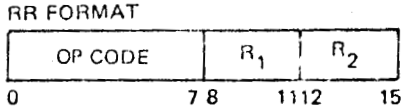
	0	1	2	3
Add Double	zero	<zero	>zero	overflow
Compare Double	equal	low	high	--
Load Complement Double	zero	<zero	>zero	overflow
Subtract Double	zero	<zero	>zero	overflow

Programming Note

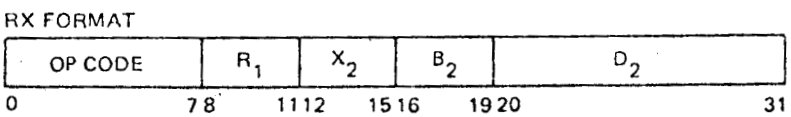
The S/360 instruction SRDA with a zero shift count can be used to test a general register pair.

11.4 INSTRUCTION FORMAT

RR FORMAT



RX FORMAT



In these formats, R_1 specifies the general register pair containing the first operand. The second operand location, if any, is defined differently for each format.

In the RR format, the R_2 field specifies the general register pair containing the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general registers specified by the X_2 and B_2 fields are added to the content of the D_2 field to form an address designating the storage location of the second operand.

A zero in an X_2 or B_2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register both for address modification and for operand location. Address modification is always completed before operation execution.

The contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged, except for the storing of the final result.

NOTE:

In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the NSSC-II assembly language are shown with each instruction. For ADD DOUBLE, for example, ADR is the mnemonic and R_1, R_2 the operand designation.

11.5 INSTRUCTIONS

The double precision fixed-point arithmetic instructions and their mnemonics, formats, and operation codes are listed in the following table. The table also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

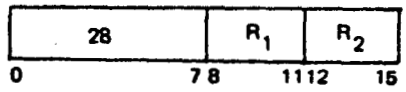
NAME	MNEMONIC	TYPE		EXCEPTIONS		CODE
Load Double	LDR	RR		S		28
Load Double	LD	RX		A,S		68
Load Complement Double	LCDR	RR	C	S	IF	23
Add Double	ADR	RR	C	S	IF	2A
Add Double	AD	RX	C	A,S,	IF	6A
Subtract Double	SDR	RR	C	S	IF	2B
Subtract Double	SD	RX	C	P,A,S,	IF	6B
Compare Double	CDR	RR	C	S		29
Compare Double	CD	RX	C	A,S		69
Store Double	STD	RX		P,A,S		60

NOTES

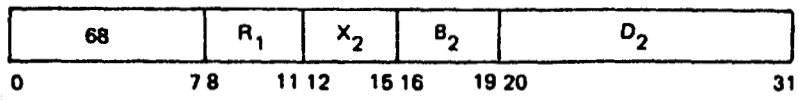
A	Addressing exception
C	Condition code is set
IF	Fixed-Point overflow exception
P	Protection exception
S	Specification exception

11.5.1 LOAD DOUBLE

LDR R_1, R_2 (RR)



LD $R_1, D_2(X_2, B_2)$ (RX)



The second operand is placed in the first operand location. The second operand is not changed. Both operands are 64 bits in length.

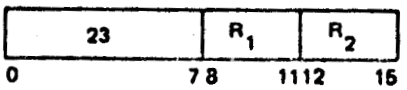
Condition Code: The code remains unchanged.

Program Interruptions:

Addressing (LD only)
Specification

11.5.2 LOAD COMPLEMENT DOUBLE

LCDR R_1, R_2 [RR]



The two's complement of the second operand is placed in the first operand location. Both operands are 64 bits in length.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

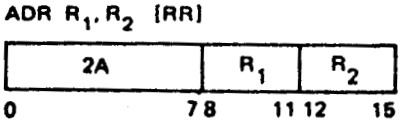
Fixed-point overflow
Specification

Programming Note

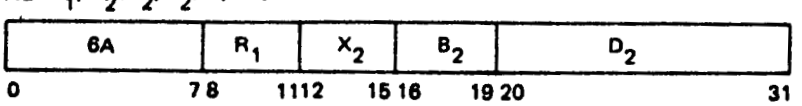
Zero remains invariant under complementation.

11.5.3 ADD DOUBLE

ADR R_1, R_2 [RR]



AD $R_1, D_2(X_2, B_2)$ [RX]



The second operand is added to the first operand, and the sum is placed in the first operand location.

Addition is performed by adding all 64 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow.

A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Interruptions:

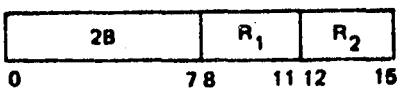
Addressing (AD only)
Specification
Fixed-point overflow

Programming Note

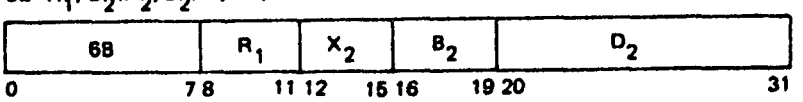
In two's-complement notation, a zero result is always positive.

11.5.4 SUBTRACT DOUBLE

SDR R_1, R_2 [RR]



SD $R_1, D_2(X_2, B_2)$ [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 64 bits of both operands participate, as in AD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow is recognized. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

0 Difference is zero
1 Difference is less than zero

2 Difference is greater than zero
3 Overflow

Program Interruptions:

Addressing (SD only)
Specifications
Fixed-point overflow

Programming Note

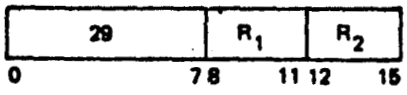
The use of the one's complement and the low-order one instead of the two's complement of the second operand is necessary for proper recognition of overflow.

When the same register is specified as first and second operand location, subtracting is equivalent to clearing the register pair.

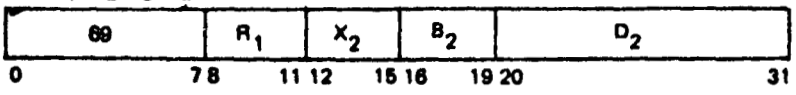
Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

11.5.5 COMPARE DOUBLE

CDR R_1, R_2 [RR]



CD $R_1, D_2(X_2, B_2)$ [RX]



The first operand is compared with the second operand, and the result determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 64-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

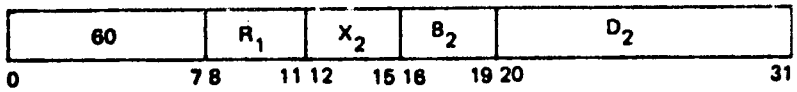
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

Addressing (CD only)
Specification

11.5.6 STORE DOUBLE

STD $R_1,D_2(X_2,B_2)$ (RX)



The first operand is stored at the second operand location.

The 64 bits in the general register pair are placed unchanged at the second operand location.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection
Addressing
Specification

11.6 DOUBLE PRECISION FIXED-POINT ARITHMETIC EXCEPTIONS

Exceptional operand designations or results cause a program interruption. When a program interuption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in fixed-point arithmetic.

Protection

The second operand in storage is storage protected. The operation is suppressed for a storage violation. Therefore, the condition code and data in registers and storage remain unchanged.

Addressing

An address designates an operand location outside the available storage for a particular installation. In most cases, the operation is terminated. Therefore, the result data are unpredictable and should not be used for further computation. The exception is STORE which is suppressed. Operand addresses are tested only when used to address storage. The address restrictions do not apply to the components from which an address is generated - the content of the D_2 field and the contents of the registers specified by X_2 and B_2.

Specification

A double-word operand is not located on a 32-bit boundary, or an instruction specifies an odd register address for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Fixed-Point Overflow

The result of a sign-control, add, or subtract, operation overflows. The interruption occurs only when the fixed-point overflow mask bit is one. The operation is completed by placing the truncated low-order results in the register and setting the condition code to three. The overflow bits are lost. In add-type operations the sign stored in the register is the opposite of the sign of the sum or difference. The state of the mask bit does not affect the result.

SECTION XII

FLOATING-POINT ARITHMETIC OPTION

The floating-point instruction set is used to perform calculations on operands with a wide range of magnitude and yielding results scaled to preserve precision.

A floating-point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess 64 binary notation; the fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit.

To avoid unnecessary storing and loading operations for results and operands, four floating-point registers are provided. The floating-point instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, and storing, as well as the sign control. Operations may be either register to register or storage to register. All floating-point instructions and registers are part of the floating-point feature.

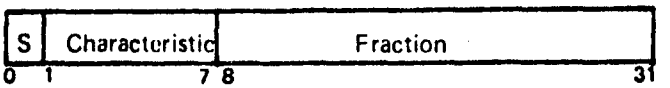
Maximum precision is preserved in addition, subtraction, multiplication, and division by producing normalized results. For addition and subtraction, instructions are also provided that generate unnormalized results. Normalized and unnormalized operands may be used in any floating-point operation.

The condition code is set as a result of all sign control, add, subtract and compare operations.

12.1 DATA FORMAT

Floating-point data occupy a fixed-length format, which is a fullword short format. This format may be used in main storage and in the floating-point registers. The floating-point registers are numbered 0, 2, 4, and 6.

Short Floating-Point Number



The first bit is the sign bit (S). The subsequent seven bit positions are occupied by the characteristic. The fraction field may have six hexadecimal digits.

All operands and results are 32-bit floating-point words.

Although final results have six fraction digits, intermediate results in ADD NORMALIZED, SUBTRACT NORMALIZED, ADD UNNORMALIZED, SUBTRACT UNNORMALIZED, COMPARE, HALVE, and MULTIPLY may have one additional low-order digit. This low-order digit, the guard digit, increases the precision of the final result.

12.2 NUMBER REPRESENTATION

The fraction of a floating-point number is expressed in hexadecimal digits. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1-7, indicates this power. The bits within the characteristic field can represent numbers from 0 through 127. To accommodate large and small magnitudes, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through +63. This technique produces a characteristic in excess 64 notation.

Both positive and negative quantities have a true fraction, the difference in sign being indicated by the sign bit. The number is positive or negative accordingly as the sign bit is zero or one.

The range covered by the magnitude (M) of a normalized floating-point number is

$$16^{-65} \leq M \leq (1 - 16^{-6}) \cdot 16^{63} \text{ or approximately } 5.4 \cdot 10^{-79} \leq M \leq 7.2 \cdot 10^{75}.$$

A number of zero characteristic, zero fraction, and plus sign is called a true zero. A true zero may arise as the result of an arithmetic operation because of the particular magnitude of the operands. A result is forced to be true zero when (1) an exponent underflow occurs and the exponent-underflow mask (PSW bit 38) is zero, (2) a result fraction of an addition or subtraction operation is zero and the significance mask (PSW bit 39) is zero, or (3) the operand of HALVE, one or both operands of MULTIPLY, or the dividend in DIVIDE has a zero fraction. When a program interruption due to exponent underflow occurs, a true zero fraction is not forced; instead, the fraction and sign remain correct, and the characteristic is 128 too large. When a program interruption due to lost significance occurs, the fraction remains zero, and the fraction sign and characteristic remain correct. Whenever a result has a zero fraction, the exponent overflow and underflow exceptions do not cause a program interruption. When a divisor has a zero fraction, division is omitted, a floating-point divide exception exists, and a program interruption occurs. In addition and subtraction, an operand with a zero fraction or characteristic participates as a normal number.

The sign of a sum, difference, product, or quotient with zero fraction is positive. The sign of a zero fraction resulting from other operations is established by the rules of algebra from the operand signs.

12.3 NORMALIZATION

A quantity can be represented with the greatest precision by a floating-point number of given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. The process of normalization consists of shifting the fraction left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called *postnormalization*. In performing multiplication and division, the operands are normalized prior to the arithmetic process. This function is called *prenormalization*.

Floating-point operations may be performed with or without normalization. Most operations are performed in only one of these two ways. Addition and subtraction may be specified either way.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. The result may or may not be normalized, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. Also, intermediate fraction results are shifted right when an overflow occurs, and the intermediate fraction result is truncated to the final result length after the shifting, if any.

Programming Note

Since normalization applies to hexadecimal digits, the three high-order bits of a normalized number may be zero.

12.4 CONDITION CODE

The results of floating-point sign-control, add, subtract, and compare operations are used to set the condition code. Multiplication, division, loading, and storing leave the code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect two types of results for floating-point arithmetic. For most operations, the states 0, 1, or 2 indicate that the result is zero, less than zero, or greater than zero. A zero result is indicated whenever the result fraction is zero, including a forced zero. State 3 is never set by floating-point operations.

For comparison, the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

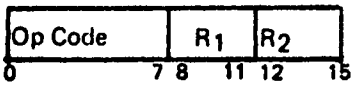
CONDITION CODE SETTING FOR FLOATING-POINT ARITHMETIC

	0	1	2	3
Add Normalized	zero	<zero	>zero	---
Add Unnormalized	zero	<zero	>zero	---
Compare	equal	low	high	---
Load and Test	zero	<zero	>zero	---
Load Complement	zero	<zero	>zero	---
Load Negative	zero	<zero	---	---
Load Positive	zero	---	>zero	---
Subtract Normalized	zero	<zero	>zero	---
Subtract Unnormalized	zero	<zero	>zero	---

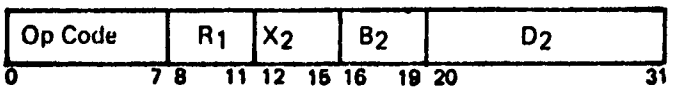
12.5 INSTRUCTION FORMAT

Floating-point instructions use the following two formats:

RR Format



RX Format



In these formats, R_1 designates the address of a floating-point register. The contents of this register will be called the first operand. The second operand location is defined differently for two formats.

In the RR format, the R_2 field specifies the address of a floating-point register containing the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general register specified by X_2 and B_2 are added to the content of the D_2 field to form an address designating the location of the second operand.

A zero in an X_2 or B_2 field indicates the absence of the corresponding address component.

The register address specified by the R_1 and R_2 fields should be 0, 2, 4, or 6. Otherwise, a specification exception is recognized, and a program interruption is caused.

The storage address of the second operand should designate word boundaries for short operands and double-word boundaries for long operands. Otherwise, a specification exception is recognized, and a program interruption is caused.

Results replace the first operand, except for the storing operations, where the second operand is replaced.

Except for the storing of the final result, the contents of all floating-point or general registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

The floating-point instructions are the only instructions using the floating-point registers.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the NSSC-II assembly language are shown with each instruction. For a register-to-register operation using LOAD, for example, LER is the mnemonic and R_1, R_2 the operand designation.

12.6 INSTRUCTIONS

The floating-point arithmetic instructions and their mnemonics, formats, and operation codes follow. All operations are part of the floating-point feature. The following table indicates when the condition code is set and the exceptions in operand designations, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Load	LER	RR F	S	38
Load	LE	RX F	A,S	78
Load and Test	LTER	RR F,C	S	32
Load Complement	LCER	RR F,C	S	33
Load Positive	LPER	RR F,C	S	30
Load Negative	LNER	RR F,C	S	31
Add Normalized	AER	RR F,C	S,U,E,LS	3A
Add Normalized	AE	RX F,C	A,S,U,E,LS	7A
Add Unnormalized	AUR	RR F,C	S,E,LS	3E
Add Unnormalized	AU	RX F,C	A,S,E,LS	7E
Subtract Normalized	SER	RR F,C	S,U,E,LS	3B
Subtract Normalized	SE	RX F,C	A,S,U,E,LS	7B
Subtract Unnormalized	SUR	RR F,C	S,E,LS	3F
Subtract Unnormalized	SU	RX F,C	A,S,E,LS	7F
Compare	CER	RR F,C	S	39
Compare	CE	RX F,C	A,S	79

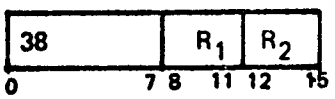
Halve	HER	RR F	S,U	34
Multiply	MER	RR F	S,U,E	3C
Multiply	ME	RX F	A,S,U,E	7C
Divide	DER	RR F	S,U,E,FK	3D
Divide	DE	RX F	A,S,U,E,FK	7D
Store	STE	RX F	P,A,S	70

NOTES

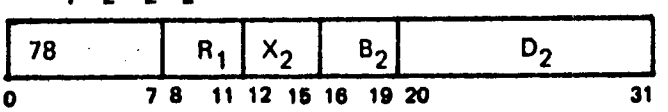
A	Addressing exception
C	Condition code is set
E	Exponent-overflow exception
F	Floating-point feature
FK	Floating-point divide exception
LS	Significance exception
P	Protection exception
S	Specification exception
U	Exponent-underflow exception

12.6.1 LOAD

LER R_1,R_2 [RR]



LE $R_1,D_2(X_2,B_2)$ [RX]



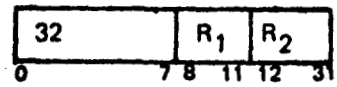
The second operand is placed in the first operand location.

The second operand is not changed. Exponent overflow, exponent underflow, or lost significance cannot occur.

Condition Code: The code remains unchanged.
Program Interruptions:
Addressing (LE, only)
Specification

12.6.2 LOAD AND TEST

LTER R_1,R_2 [RR]



The second operand is placed in the first operand location, and its sign and magnitude determine the condition code.

The second operand is not changed.

Resulting Condition Code:
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:
Specification

Programming Note

When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

12.6.3 LOAD COMPLEMENT

LCER R_1,R_2 [RR]

The second operand is placed in the first operand location with the sign changed to the opposite value.

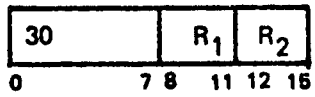
The sign bit of the second operand is inverted, while characteristic and fraction are not changed.

Resulting Condition Code:
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:
Specification

12.6.4 LOAD POSITIVE

LPER R_1,R_2 [RR]



The second operand is placed in the first operand location with the sign made plus.

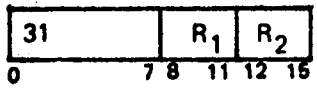
The sign bit of the second operand is made zero, while characteristic and fraction are not changed.

Resulting Condition Code:
0 Result fraction is zero
1 --
2 Result is greater than zero
3 --

Program Interruptions:
Specification

12.6.5 LOAD NEGATIVE

LNER R_1,R_2 [RR]



The second operand is placed in the first operand location with the sign made minus.

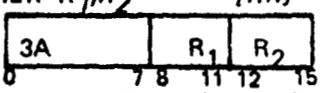
The sign bit of the second operand is made one, even if the fraction is zero. Characteristic and fraction are not changed.

Resulting Condition Code:
0 Result fraction is zero
1 Result is less than zero
2 -
3 -

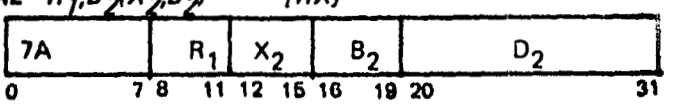
Program Interruptions:
Specification

12.6.6 ADD NORMALIZED

AER R_1,R_2 [RR]



AE $R_1,D_2(X_2,B_2)$ [RX]



The second operand is added to the first operand, and the normalized sum is placed in the first operand location.

The low-order halves of the floating point registers are ignored and remain unchanged.

Addition of two floating-point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each hexadecimal digit of shift, until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit, and the characteristic is increased by one. If this increase causes a characteristic overflow, an exponent-overflow exception is signaled, and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 smaller than the correct characteristic.

The intermediate sum consists of 7 hexadecimal digits and a possible carry. The low-order digit is a guard digit obtained from the fraction which is shifted right. Only one guard digit position participates in the fraction addition. The guard digit is zero if no shift occurs.

After the addition, the intermediate sum is left-shifted as necessary to form a normalized fraction; vacated low-order digit positions are filled with zeros; the characteristic is reduced by the amount of shift.

If normalization causes the characteristic to underflow and if the corresponding mask bit is one, a program interruption occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 larger than the correct one. If the corresponding mask bit is zero, the result is made a true zero. If no left shift takes place, the intermediate sum is truncated to the proper fraction length.

When the intermediate sum is zero and the significance mask bit is one, a significance exception exists, and a program interruption takes place. No normalization occurs; the intermediate sum characteristic remains unchanged. When the intermediate sum is zero and the significance mask bit is zero, the program interruption for the significance exception does not occur; rather, the characteristic is made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the sum is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

Resulting Condition Code:

0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

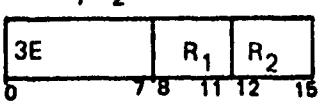
Addressing (AE only)
Specification
Significance
Exponent overflow
Exponent underflow

Programming Note

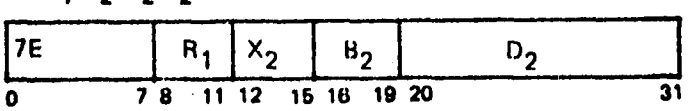
Interchanging the two operands in a floating-point addition does not affect the value of the sum.

12.6.7 ADD UNNORMALIZED

AUR R_1,R_2 (RR)



AU $R_1,D_2(X_2,B_2)$ (RX)



The second operand is added to the first operand, and the unnormalized sum is placed in the first operand location.

After the addition the intermediate sum is truncated to the proper fraction length.

When the resulting fraction is zero and the significance mask bit is one, a significance exception exists and a program interruption takes place. When the resulting fraction is zero and the significance mask bit is zero, the program interruption for the significance exception does not occur; rather, the characteristic is made zero, yielding a true zero result.

Leading zeros in the result are not eliminated by normalization, and an exponent underflow cannot occur.

The sign of the sum is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

Resulting Condition Code:

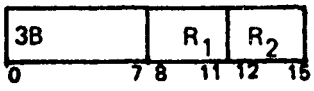
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

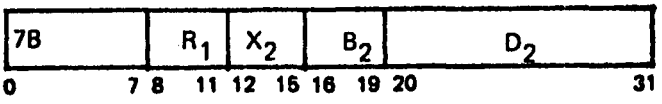
Addressing (AU only)
Specification
Significance
Exponent overflow

12.6.8 SUBTRACT NORMALIZED

SER R_1,R_2 [RR]



SE $R_1,D_2(X_2,B_2)$ [RX]



The second operand is subtracted from the first operand, and the normalized difference is placed in the first operand location.

The SUBTRACT NORMALIZED is similar to ADD NORMALIZED, except that the sign of the second operand is inverted before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

Resulting Condition Code:

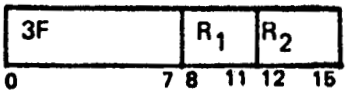
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

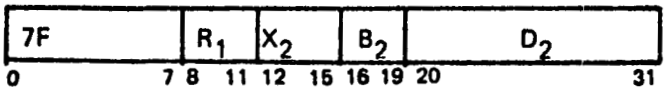
Addressing (SE only)
Specification
Significance
Exponent overflow
Exponent underflow

12.6.9 SUBTRACT UNNORMALIZED

SUR R_1,R_2 [RR]



SU $R_1,D_2(X_2,B_2)$ [RX]



The second operand is subtracted from the first operand, and the unnormalized difference is placed in the first operand location.

The SUBTRACT UNNORMALIZED is similar to ADD UNNORMALIZED, except for the inversion of the sign of the second operand before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

Resulting Condition Code:

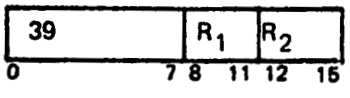
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

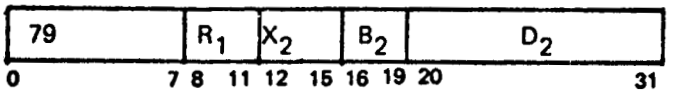
Addressing (SU only)
Specification
Significance
Exponent overflow

12.6.10 COMPARE

CER R_1,R_2 [RR]



CE $R_1,D_2(X_2,B_2)$ [RX]



The first operand is compared with the second operand, and the condition code indicates the result.

Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. An exponent inequality is not decisive for magnitude determination since the fractions may have different numbers of leading zeros. An equality is established by following the rules for normalized floating-point subtraction. When the intermediate sum, including the guard digit, is zero, the operands are equal. Neither operand is changed as a result of the operation.

An exponent-overflow, exponent-underflow, or lost-significance exception cannot occur.

Resulting Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

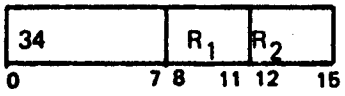
Addressing (CE only)
Specification

Programming Note

Numbers with zero fraction compare equal even when they differ in sign or characteristic.

12.6.11 HALVE

HER R_1,R_2 (RR)



The second operand is divided by two, and the normalized quotient is placed in the first-operand location.

The second operand remains unchanged.

The fraction of the second operand is shifted right one bit position, placing the contents of the low-order bit position into the high-order bit position of the guard digit and introducing a zero into the high-order bit position of the fraction. The intermediate result is subsequently normalized, and the normalized quotient is placed in the first-operand location. The guard digit participates in the normalization.

When normalization causes the characteristic to become less than zero, exponent underflow occurs. If the exponent-underflow mask is zero, the sign, characteristic, and fraction are set to zero, thus making the result a true zero. If the exponent-underflow mask is one, a program interruption occurs. The result is normalized, its sign and fraction remain correct, and the characteristic is made 128 larger than the correct characteristic.

When the fraction of the second operand is zero, the sign, characteristic, and fraction of the result are made zero. No normalization is attempted, and a significance exception is not recognized.

Condition Code: The code remains unchanged.
Program Interruptions:
Specification
Exponent underflow

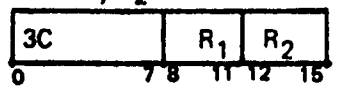
Programming Notes

The halve operation is identical to a divide operation with the number two as divisor. The halve operation is identical to a multiply operation with one-half as a multiplier.

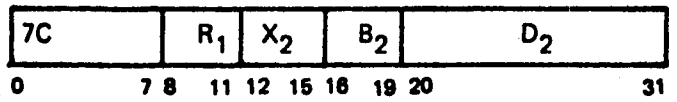
The result of HALVE is replaced by a true zero only when the second-operand fraction is zero, or when exponent underflow occurs with the exponent-underflow mask set to zero. When the fraction of the second operand is zero, except for the low-order bit position, the low-order one is shifted into the guard digit position and participates in the postnormalization.

12.6.12 MULTIPLY

MER R_1,R_2 *[RR]*



ME $R_1,D_2(X_2,B_2)$ *[RX]*



The normalized product of multiplier (the second operand) and multiplicand (the first operand) replaces the multiplicand.

The multiplication of two floating-point numbers consists of a characteristic addition and a fraction multiplication. The sum of the characteristics less 64 is used as the characteristic of an intermediate product. The sign of the product is determined by the rules of algebra.

The product fraction is normalized by prenormalizing the operands and postnormalizing the intermediate product, if necessary. The intermediate product characteristic is reduced by the number of left-shifts. The intermediate product fraction has 12 digits which is normalized and then truncated to 6 digits.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is completed, and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 smaller than the correct characteristic. The overflow exception does not occur for an intermediate product characteristic exceeding 127 when the final characteristic is brought within range because of normalization.

Exponent underflow occurs if the final product characteristic is less than zero. If the corresponding mask bit is one, a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 larger than the correct characteristic. If the corresponding mask bit is not one, the result is made a true zero. Underflow is not signaled when an operand's characteristic becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 6 digits of the intermediate product fraction are zero, the product sign and characteristic are made zero, yielding a true zero result. No interruption for exponent underflow or exponent overflow can occur when the result fraction is zero. The program interruption for lost significance is never taken for multiplication.

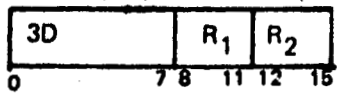
Condition Code: The code remains unchanged.
Program Interruptions:
Addressing (ME only)
Specification
Exponent overflow
Exponent underflow

Programming Note

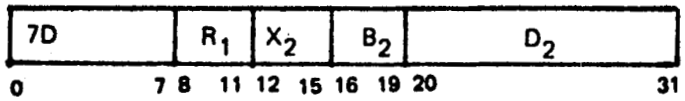
Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

12.6.13 DIVIDE

DER R_1,R_2 [RR]



DE $R_1,D_2(X_2,B_2)$ [RX]



The dividend (the first operand) is divided by the divisor (the second operand) and replaced by the quotient. No remainder is preserved.

The low-order halves of the floating-point register are ignored and remain unchanged.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics plus 64 is used as an intermediate quotient characteristic. The sign of the quotient is determined by the rules of algebra.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift may be called for. The intermediate-quotient characteristic is adjusted for the shifts. All dividend fraction digits participate in forming the quotient, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to the desired number of digits.

A program interruption for exponent overflow occurs when the final-quotient characteristic exceeds 127. The operation is completed. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 smaller than the correct characteristic.

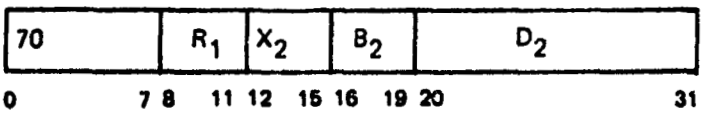
If the final quotient characteristic is less than zero and the mask bit is one, a program interruption for exponent underflow occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 larger than the correct characteristic. If the corresponding mask bit is not one, the result is made a true zero. Underflow is not signaled for the intermediate quotient or for the operand characteristics during prenormalization.

When division by a divisor with zero fraction is attempted, the operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient fraction will be zero, yielding a true zero result without taking the program interruption for exponent underflow and exponent overflow. The program interruption for significance is never taken for division.

Condition Code: The code remains unchanged.
Program Interruptions:
Addressing (DD only)
Specification
Exponent overflow
Exponent underflow
Floating-point divide

12.6.14 STORE

STE $R_1,D_2(X_2,B_2)$ [RX]



The first operand is stored at the second operand location.

The first operand remains unchanged.

Condition Code: The code remains unchanged.
Program Interruptions:

Addressing
Protection (store violation)
Specification

12.7 FLOATING-POINT ARITHMETIC EXCEPTIONS

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in floating-point arithmetic.

Operation: The floating-point feature is not installed, and an attempt is made to execute a floating-point instruction. The instruction is suppressed. The condition code and data in registers and storage remain unchanged.

Protection: The storage protection bit for CPU stores is set to a 1 in the addressed block of main storage when a STE instruction is encountered.

Addressing: An address designates an operand location outside the available storage for the installed system. In most cases, the operation is terminated. The result data and the condition code, if affected, are unpredictable and should not be used for further computation. The exception is STORE (STE), which is suppressed.

Specification: A storage operand is not located on a 32-bit boundary or a floating-point register address other than 0, 2, 4, or 6 is specified. The instruction is suppressed. Therefore, the condition code and data in registers and storage remain unchanged. The address restriction does not apply to the components from which an address is generated--the content of the D_2 field and the contents of the registers specified by X_2 and B_2.

Exponent Overflow: The result characteristic in addition, subtraction, multiplication, or division exceeds 127, and the result fraction is not zero. The operation is completed, and a program interruption occurs. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic. For addition and subtraction, the condition code is set to 1 when the result is less than zero, and the condition code is set to 2 when the result is greater than zero. For multiplication and division, the condition code remains unchanged.

Exponent Underflow: The result characteristic in addition, subtraction, multiplication, halving, or division is less than zero, and the result fraction is not zero. The operation is completed, and a program interruption occurs if the exponent-underflow mask bit (PSW bit 38) is one.

The setting of the exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, thus making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

For addition and subtraction, the condition code is set to 0 when the exponent-underflow mask bit is zero. With the mask bit one, the condition code for addition and subtraction is set to 1 when the result is less than zero, and the condition code is set to 2 when the result is greater than zero. For multiplication, halving, and division, the condition code is left unchanged.

Significance: The result fraction of an addition or subtraction is zero. A program interruption occurs if the significance mask bit (PSW bit 39) is one. The mask bit affects also the result of the operation. When the significance mask bit is a zero, the operation is completed by replacing the result with a true zero. When the significance mask bit is one, the operation is completed without further change to the characteristic of the result. In either case, the condition code is set to 0.

Floating-Point Divide: Division by a number with zero fraction is attempted. The division is suppressed; therefore, the condition code and data in registers and storage remain unchanged.