IBM® Customer Engineering

Manual of Instruction

7030 Data Processing System

Introduction and Programming, Volume II

IBM Customer Engineering

Manual of Instruction

7030 Data Processing System

Introduction and Programming, Volume II

PREFACE

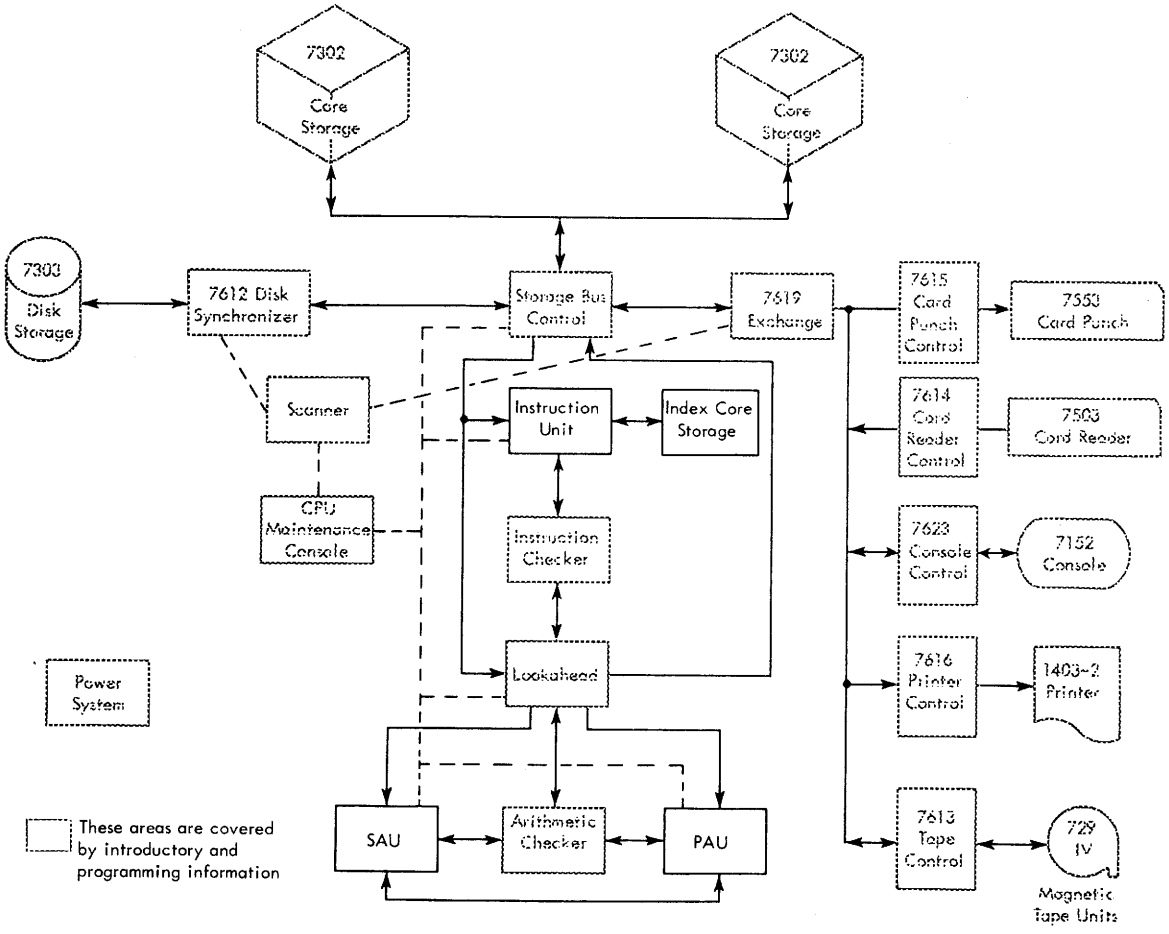
This is Volume II of the 7030 Introduction and Programming, Customer Engineering Manual of Instruction. The material presented is based on the information available on the commercial 7030 System as of October 17, 1960. Areas of the system are covered as shown in the frontispiece.

The manual contains:

1. General programming functions of the 7030 system.
2. Explanation of each instruction type.
3. Index Arithmetic and Instructions.
4. Floating Point Arithmetic and detailed descriptions of each FP Instruction.

This Volume II, Form R23-9694, obsoletes sections of the Preliminary Instruction Text on 7030 Introduction and Programming, Books A through G.

CONTENTS



IBM 7030 DATA PROCESSING SYSTEM

6 PROGRAMMING THE 7030

THIS SECTION of the manual will aid the Customer Engineer in programming the 7030, as required for operation and servicing. The 7030 instructions are introduced a few at a time, followed by programming examples.

It is expected that the Customer Engineer will use symbolic programming almost exclusively. For this reason, all programming examples are presented in STRAP (assembly program) notations. The rules for using STRAP are introduced as required.

Before discussing specific instructions, certain features of the 7030 which are pertinent to programming are covered.

6.1 PROGRAM INTERRUPTION SYSTEM

The sequence of instructions executed may be altered not only by the programmed branch type operations, but also by the computer itself upon occurrence of certain conditions. These conditions are recorded in a group of indicators that can be tested by a BRANCH ON INDICATOR. An actuated indicator can cause program interruption without an explicit programmed test.

For many indicators there are mask bits which the programmer can set. If a mask bit is set to zero, occurrence of the corresponding indication does not cause program interruption. If, however, the mask bit is set to one, occurrence of the corresponding indication can cause the instruction sequence to be immediately altered. When this occurs, a single instruction, located at a place unique to the causing condition, is executed. Since this instruction may be of the branch type, the sequence of instructions branched into could be a correction routine on the condition that caused the interrupt. Some indicators have mask bits that are permanently set to zero or one. Those with permanent mask bits of zero can never cause interruption, but they can be tested by branching operations. Indicators with mask bits permanently set to one cause interruption whenever both the indicator is actuated and the interruption system is enabled.

In order that interruption correction routines may have adequate time to correct conditions causing interruption, or at least to store the machine state at the time of interruption, it is necessary to be able to prevent immediately succeeding interruptions. This is accomplished by controlling an enabling mechanism. Upon execution of BRANCH DISABLED, the mechanism disables the computer's ability to interrupt a program. The system remains in the disabled state until execution of one of the instructions BRANCH ENABLED or BRANCH ENABLED AND WAIT. The system is then said to be "enabled," and interruption occurs whenever both an indicator and its corresponding mask bit are on.

Address monitoring is also controlled by the enabling mechanism. When the system is enabled, monitoring is effective; when disabled, monitoring is suspended. Most indicators may come on at any time whether the system is enabled or disabled.

6.1.1 Indicator Register

The indicator register contains 64 bits. Each bit is turned on (set to 1) or off (set to 0) when certain conditions occur in the computer system. Each bit is also turned off automatically as it causes a program interruption. An individual bit may also be turned off by a BRANCH ON INDICATOR that tests it.

The indicator register has address 11. Bit positions 0-19 may be read, but not loaded. Any attempt to store into these bit positions is suppressed. Bit positions 20-63 may either be read or loaded. Thus, the entire contents of the indicator register may be stored in any storage location by a transmission type of instruction. Similarly, the register may be loaded from any storage location by a transmission instruction, but only bit positions 20-63 of the indicator register will be affected by such operations.

For the purpose of program interruption, the bits in the indicator register have a built-in priority, which decreases from left to right. This affects interruption only when two conditions are present at the time interruption occurs.

On the list in Figure 6.1-1, some indicators are temporary and other indicators are permanent. The permanent indicators remain on, when once turned on, unless they cause program interruption. A temporary indicator remains on (off) only until the performance of the next result which could turn it off (on). It is then changed to correspond to the latest result. The comparison result indicators, for example, are temporary. At any time all three of those indicators are set to show the result of the most recent comparison performed unless they have been changed by programming.

The input-output status indicators EPGK, UK, EE, EOP, and CS are controlled as a group, so they collectively describe the status of a single unit, at any time. These indicators are set to describe a unit status and the channel address register is set to the address of the unit. As interruptions occur, the indicators are reset. When all five are zero and if the interruption mechanism is enabled, the exchange may set the whole group to describe another unit. This group of indicators is described in detail under "Input-Output Programming."

The 64 indicators are defined below in terms of the conditions which would actuate the indicator, and the peculiar conditions by which each indicator is turned off. Any indicator is turned off by interruption upon it, and can be turned off or left unchanged when it is tested with a BRANCH ON INDICATOR. Indicators 20 through 63 may also be turned on or off by a BRANCH ON BIT or by operating upon the indicator register contents as a field in storage. Except where noted, these are the only means of turning indicators off. The indicators are listed in the order of their priority. The indicators are also defined in the sections on instructions which affect the indicators.

Equipment Check

0 Machine Check (MK). An error has been detected by the machine checking circuits. There is no guarantee that any specific element in the machine is in either its original or correct state.

1 Instruction Check (IK). An error has been detected during the performance of the current instruction. Only elements usually affected by the operation can be in error.

The instruction counter contains the address of the next instruction in storage after the one during which the error was detected.

2 Instruction Reject (IJ). An error has been detected which was identified with the current instruction. The instruction has been executed as though it were a NO OPERATION. The instruction counter contains the address of the next instruction in storage.

3 Exchange Control Check (EK). The exchange has failed to function properly in a manner that is not identified with any particular unit.

Attention Request

4 Time Signal (TS). The interval timer has become zero.

5 CPU Signal (CPUS). Some other instruction execution system in another central processing unit desires the attention of this CPU.

Input-Output Reject

6 Exchange Check Reject (EKJ). An error was detected by the exchange in the course of testing and setting up the present instruction. This reject may indicate equipment malfunctions or attempted selection of a channel which is not available to the programmer. The instruction is not executed, and the indicator is actuated before the computer proceeds to the next instruction.

7 Unit Not Ready Reject (UNRJ). An instruction was given for a unit which was not ready to be operated. The unit ready status bit is zero. The instruction is not executed, and the indicator is actuated before the computer proceeds to the next instruction.

8 Channel Busy Reject (CBJ). An instruction was given for a unit which was still connected to the exchange or which is still waiting to give a program interruption as a result of a previous instruction. The instruction is not executed, and the indicator is actuated before the computer proceeds to the next instruction. Certain operations, such as rewinding tape, are completed by the unit after disconnecting from the exchange. New instructions for channels performing such operations can be accepted; they do not cause busy rejects.

Input-Output Status

9 Exchange Program Check (EPGK). The last operation initiated for the unit specified by the channel address register has been terminated because of a programming error. Such errors include giving a unit an instruction it cannot execute or specifying core storage locations to which the exchange does not have access.

10 Unit Check (UK). The last operation initiated for the unit specified by the channel address register encountered malfunctioning of the equipment or defects of the recording medium. These malfunctions include uncorrectable data errors, card jams, broken tapes, and so on.

11 End Exception (EE). The last operation initiated for the unit specified by the channel address register encountered an exceptional condition, usually associated with

the recording medium or some subdivision of the data to be transmitted, which is not expected to occur during every operation. Examples of such conditions are: out of material, sensing of a tape mark, or depression of the erase key on an operator's console.

12 End of Operation (EOP). The last operation initiated for the unit specified by the channel address has been completed as specified by the instruction and its control words, if any, unless the unit check indicator is also on. End of operation in conjunction with unit check indicates that the operation was terminated because an uncorrectable data error had been discovered.

13 Channel Signal (CS). A unit on the channel specified by the channel address register has transmitted a channel signal to the computer. If the signal originates while one of the units connected to this channel is in operation, it is transmitted to the computer at the end of the operation. The signal can originate upon depression of the signal key on the unit, by the readying of a unit, or by completion of some operation such as rewinding.

14 Reserved.

Instruction Exceptions

15 Operation Code Invalid (OP). The instruction just attempted used an operation code or combination of modifiers which is not defined. An instruction with an invalid operation code is executed as NO OPERATION.

16 Address Invalid (AD). The location of the instruction next due for execution is at an address not provided in the computer system, or the location specified by the effective operand address of the instruction just attempted is either not provided in the computer system or invalid for the specified operation. The affected instruction is executed as a NO OPERATION. Branching to an address below 32 also actuates the indicator.

17 Unended Sequence of Addresses (USA). A certain type of instruction has been in progress for longer than one millisecond. This indicator is further explained under the instruction by which it is affected.

18 Execute Exception (EXE). An instruction was executed under control of the EXECUTE or the EXECUTE INDIRECT AND COUNT operations that attempted to change the contents of the instruction counter. The subject operation was suppressed.

19 Data Store (DS). A store or other to-storage operation has attempted to change the contents of some location in the protected area of storage while the interruption system was enabled. The operation was suppressed.

20 Data Fetch (DF). A computer operation has attempted to fetch data from a location in the protected area of storage while the interruption system was enabled. If the corresponding mask bit was one, the fetch did not take place, and the entire operation was suppressed. If the mask bit was zero, the operation was completed normally.

21 Instruction Fetch (IF). An attempt has been made to fetch an instruction from, or branch to, a location in the protected area of storage while the interruption system was enabled. If the corresponding mask bit was one, the fetch or branch did not take place and the entire operation was suppressed. If the mask bit was zero, the instruction was executed normally.

Result Exceptions

22 Lost Carry (LC). Information was lost on the operation just executed because a carry was propagated beyond the end of the legitimate sum field, as defined for the operation. Lost carry is actuated in unnormalized floating point operation when a fraction overflow bit occurs in a sum or when a low order one is lost in a remainder. Lost carry is also actuated when a SHIFT FRACTION operation shifts non-zero bits left beyond the end of the fraction field.

23 Partial Field (PF). The operation just executed failed to use all the operand data provided.

24 Zero Divisor (ZD). The divisor or divisor fraction of the division just attempted consisted only of zero bits. The division was not attempted.

Result Exceptions, Floating Point Only

25 Imaginary Root (IR). A STORE ROOT was performed on the magnitude of a negative operand.

26 Lost Significance (LS). In a floating point add-type operation in which at least one operand was non-zero, both the final result placed in the accumulator and the overflow bit were zero. A more detailed description of the LS indicator is included under the instructions which affect the indicator.

27 Preparatory Shift Greater than 48 (PSH). A floating point add-type operation, using operands with exponent flags of zero, required a preparatory shift of the operands relative to one another by an amount greater than 48 places.

28 Exponent Flag Positive: Exponent$\geq 2^{10}$ (XPFP). The result of a floating point operation had a positive exponent with an exponent flag of 1 propagated from an operand with an exponent flag of 1.

29 Exponent Overflow: Exponent$\geq 2^{10}$ (XPO). The result of a floating point operation had a positive exponent with an exponent flag of 1 generated from operands with exponent flags of 0.

30 Exponent Range High: $2^{10} >$Exponent$\geq 2^9$ (XPH). The result of a floating point operation had a positive exponent with an exponent flag of 0, in the range 2^9 through $2^{10} - 1$, inclusive.

31 Exponent Range Low: $2^9 >$Exponent $\geq 2^6$ (XPL). The result of a floating point operation had a positive exponent, with an exponent flag of 0, in the range 2^6 through $2^9 - 1$ inclusive.

32 Exponent Underflow: Exponent $\leq -2^{10}$ (XPU). The result of a floating point operation had a negative exponent with an exponent flag of 1, generated from operands with exponent flags of 0.

33 ZERO MULTIPLY (ZM). The final result of a normalized floating point operation using multiplication is an order of magnitude zero with the exponent not in the XFN range.

34 Remainder Underflow (RU). The remainder developed during a floating point division had a negative exponent with an exponent flag of 1, generated from a dividend with exponent flag of 0.

Flagging

35 Data Flag T (TF). The data flag bit T (immediately to the right of the sign bit) of the operand of the instruction just executed was 1. The indicator is turned off by any arithmetic instruction whose operand from storage is not so flagged.

36 Data Flag U (UF). The second of the possible data flag bits of the operand just used was 1. The indicator is turned off by any arithmetic instruction whose operand from storage is not so flagged.

37 Data Flag V (VF). The third and rightmost of the possible data flag bits of the operand just used was 1. The indicator is turned off by any arithmetic instruction whose operand from storage is not so flagged.

38 Index Flag (XF). The index flag bit (bit 25) of the index word just modified was 1, before any refill operation was performed. The indicator is turned off by any index arithmetic or refull operation whose modificand does not have its flag on.

Transit Operations

39 Binary Transit (BTR). A binary LOAD TRANSIT AND SET was executed.

40 Decimal Transit (DTR). A decimal MULTIPLY, DIVIDE, MULTIPLY AND ADD, or LOAD TRANSIT AND SET was executed.

Program

41-47 Program Indicators Zero through Six (PG0-PG6). These indicators are set by programming only.

Index Result

48 Index Count Zero (XCZ). The result count field of the index word last modified by an index modification operation, including REFILL, was zero. This condition existed before a refill, if any, but after any other modifications. This indicator is turned off by an index modification operation whose result does not have a zero count field.

49 Index Value Less than Zero (XVLZ). The value field resulting from the index modification operation just executed was non-zero and negative. This indicator is

turned off by any index modification operation whose result has a value field which is zero or positive.

50 Index Value Zero (XVZ). The value field resulting from the index modification operation just executed was zero. This indicator is turned off by any index modification operation whose result has a value field which is non-zero.

51 Index Value Greater than Zero (XVGZ). The value field resulting from the index modification operation just executed was non-zero and positive. This indicator is turned off by any index modification operation whose result has a value field which is zero or negative. One and only one of indicators XVLZ, XVZ, XVGZ is on at any time, unless they have been changed by programming.

52 Index Low (XL). The result of the index comparison operation just executed was that the compared field of the index word was lower than that of the comparand specified by the effective address. This indicator is turned off by an index comparison whose result is not low.

53 Index Equal (XE). The result of the index comparison just executed was that the compared field of the index word was equal to that of the comparand. This indicator is turned off by an index comparison whose result is not equal.

54 Index High (XH). The result of the index comparison just executed was that the compared field of the index word was higher than that of the comparand. This indicator is turned off by an index comparison whose result is not high.

One and only one of the indicators XL, XE, and XH is on at any time, unless they have been changed by programming.

Arithmetic Result

55 To-Memory Operation (MOP). The result of the operation last executed was placed in storage.

56 Result Less than Zero (RLZ). The result of the integer or floating point arithmetic operation just executed was non-zero and negative. This indicator is turned off by any non-comparative arithmetic operation whose result is zero or positive, or by any connective operation.

57 Result Zero (RZ). The result of the non-comparative integer or floating point arithmetic operation is zero, or the result of a connective operation is zero.

58 Result Greater than Zero (RGZ). The result of the non-comparative arithmetic operation just executed was non-zero and positive or the result of the connective operation just executed was non-zero. This indicator is turned off by a non-comparative arithmetic operation whose result is zero or negative, or by a connective operation whose result is zero. One and only one of indicators RLZ, RZ, and RGZ is on at any one time, unless they have been changed by programming.

59 Result Negative (RN). The result of the non-comparative arithmetic operation just executed was negative whether zero or not. This indicator is turned off by a non-comparative arithmetic operation whose result is positive, whether zero or not, or by any connective operation.

60 Accumulator Low (AL). The result of the arithmetic comparison operation just executed was that the accumulator contents were less than the comparand specified by the effective address. The arithmetic comparison operations are COMPARE, COMPARE FIELD, COMPARE MAGNITUDE, COMPARE FOR RANGE, COMPARE FIELD FOR RANGE, COMPARE MAGNITUDE FOR RANGE, COMPARE IF EQUAL, and COMPARE FIELD IF EQUAL. This indicator is turned off by a comparison operation whose result is not low.

61 Accumulator Equal (AE). The result of the arithmetic comparison operation just executed was that the accumulator contents were equal to the comparand or within the compared range. This indicator is turned off by a comparison operation whose result is not equal or within the compared range.

62 Accumulator High (AH). The result of the arithmetic comparison operation just executed was that the accumulator contents were greater than the comparand. This indicator is turned off by a comparison operation whose result is not high. One and only one of indicators AL, AE, and AH is on at any time, unless they have been changed by programming.

Mode

63 Noisy Mode (NM). This indicator can only be turned on by program control, through the use of a connective operation, a loading of the indicator register, or BRANCH ON BIT. When it is on, all floating point operations are performed in the noisy mode, as defined in "Floating Point Instructions."

6.1.2 Mask Register

The mask word consists of 64 bits. Each bit corresponds to an indicator in the indicator register. If the bit of the mask is 1 and the interruption system enabled, occurrence of the corresponding condition will cause program interruption at the end of the operation during which the indicator is turned on. If the bit of the mask is 0, turning on the indicator does not cause program interruption. Thus, interruption requires the mask bit to be 1, the indicator bit to be 1, and the mechanism to be enabled.

Not all of the 64 mask bits can be set by program control. Those corresponding to indicators 0-19 are permanently set to 1. Those corresponding to indicators 20-47 can be set to 0 or 1 by program control. Those corresponding to indicators 48-63 are permanently set to 0. Thus, when the interruption mechanism is enabled, some indicators always cause program interruption, some may or may not, and others can never cause interruption. Figure 6.1-1 shows the masking property of each indicator.

The mask register has address 12 and can be loaded and stored by transmission instructions. When the mask contents are read, the permanently set bits read out as 1 or 0 according to their type, regardless of what may have been loaded into those positions earlier. Individual mask bits with addresses 12.20 through 12.47 can be set or changed by connective operations, arithmetic operations, or BRANCH ON BIT.

6.1.3 Interruption Address Register

Because of the varied conditions which turn on indicators, varied correction routines are required to meet the needs created by the conditions. There can be up to 48 such

No.	Mnemonic	Mask	Class	Name
Equipment Check				
0	MK	1	P,H	Machine Check
1	IK	1	P,H	Instruction Check
2	IJ	1	P,S	Instruction Reject
3	EK	1	P,C	Exchange Control Check
Attention Request				
4	TS	1	P,C	Time Signal
5	CPUS	1	P,C	CPU Signal
Input-Output Rejects				
6	EKJ	1	P,S	Exchange Check Reject
7	UNRJ	1	P,S	Unit Not Ready Reject
8	CBJ	1	P,S	Channel Busy Reject
Input-Output Status				
9	EPGK	1	P,C	Exchange Program Check
10	UK	1	P,C	Unit Check
11	EE	1	P,C	End Exception
12	EOP	1	P,C	End of Operation
13	CS	1	P,C	Channel Signal
14		1		Reserved
Instruction Exception				
15	OP	1	P,S	Operation Code Invalid
16	AD	1	P,S	Address Invalid
17	USA	1	P,S	Unended Sequence of Addresses
18	EXE	1	P,S	Execute Exception
19	DS	1	P,S	Data Store
20	DF	m	P,S*,C	Data Fetch
21	IF	m	P,S*,C	Instruction Fetch

*Class S when mask is one; otherwise Class C.

No.	Mnemonic	Mask	Class	Name
Result Exception				
22	LC	m	P,C	Lost Carry
23	PF	m	P,C	Partial Field
24	ZD	m	P,C	Zero Divisor
Result Exception - Floating Point				
25	IR	m	P,C	Imaginary Root
26	LS	m	P,C	Lost Significance
27	PSH	m	P,C	Preparatory Shift Greater than 48
28	XPFP	m	P,C	Exponent Flag Positive: Exp. $\geq 2^{10}$
29	XPO	m	P,C	Exponent Overflow: Exp. $\geq 2^{10}$
30	XPH	m	P,C	Exponent High: $2^{10} >$ Exp. $\geq 2^{9}$
31	XPL	m	P,C	Exponent Low: $2^{9} >$ Exp. $\geq 2^{6}$
32	XPU	m	P,C	Exponent Underflow: Exp. $\leq -2^{10}$
33	ZM	m	T,C	Zero Multiply
34	RU	m	P,C	Remainder Underflow

No.	Mnemonic	Mask	Class	Name
Flagging				
35	TF	m	T,C	Data Flag T — Reset by variable field length and floating point operations requiring data fetch.
36	UF	m	T,C	Data Flag U
37	VF	m	T,C	Data Flag V
38	XF	m	T,S*,C	Index Flag - Reset by index arithmetic operations
Transit Operations				
39	BTR	m	P,C	Binary Transit
40	DTR	m	P,C	Decimal Transit
Program				
41-47	PG0-PG6	m	P,C	Program Indicators Zero through Six
48	XCZ	0	T,C	Index Count Zero — Reset by index arithmetic ops. other than comparisons
49	XVLZ	0	T,C	Index Value Less than Zero
50	XVZ	0	T,C	Index Value Zero
51	XVGZ	0	T,C	Index Value Greater than Zero
52	XL	0	T,C	Index Low — Reset by index comparison operations.
53	XE	0	T,C	Index Equal
54	XH	0	T,C	Index High
Arithmetic Result				
55	MOP	0	T,C	To-Memory Op. — Reset by variable field length or floating point op.
56	RLZ	0	T,C	Result Less than Zero — Reset by variable field length or floating point ops. except comparisons
57	RZ	0	T,C	Result Zero
58	RGZ	0	T,C	Result Greater than Zero
59	RN	0	T,C	Result Negative
60	AL	0	T,C	Accum. Low — Reset by comparison operations
61	AE	0	T,C	Accum. Equal
62	AH	0	T,C	Accum. High
Mode				
63	NM	0	P,C	Noisy Mode

T Indicator temporary and is reset by later operations.

P Indicator permenent but may be turned off by interruption or during BI.

C The execution of the instruction during which the indicator is actuated is completed.

H The execution of the instruction during which the indicator is actuated is terminated.

S The execution of the instruction during which the indicator is actuated is suppressed, except during EX, EXIC, T, and SWAP.

FIGURE 6.1-1. INDICATOR LIST

routines tailored to the program in execution, and each corresponding to an indicator. The initial instruction for a routine occupies all or half of a location in a table of 48 successive full-word locations. The interruption address register contains 18 bits and has address 2.0, which places it in the protected area of storage.

When a program interruption occurs, a mechanism develops the number of the leftmost actuated indicator whose mask bit is one. After this leftmost indicator has been turned off, the developed number is added to the contents of the interruption address register to form the effective interruption address. This address does not replace the contents of the instruction counter but is used to fetch a single instruction which is executed before a second interruption can occur. The instruction fetch indicator (IF) cannot be turned on when this instruction is fetched even though it might lie in the protected area of core storage.

The instruction counter is stepped neither before nor during the execution of the instruction at the effective interruption address; however, the instruction counter can be replaced if a successful branch operation is caused by the instruction at the effective interruption address.

When interruption occurs and an instruction at the effective interruption address is executed (it may be of the branch type), the next instruction executed will be one of the following:

1. The instruction then addressed by the instruction counter if a second interruption has not occurred.
2. The instruction at the new effective interruption address if a second interruption has occurred.

If the interruption system is enabled, and an indicator and its corresponding mask bit are both on, interruption will follow as early as possible but never during the execution of an instruction. Interruption is permitted after the execution of an instruction is completed, terminated, or suppressed, and before execution of the next instruction.

6.2 SPECIAL LOCATIONS

The address field of all instructions provides for an 18-bit word address. This permits addressing of 262,144 (2^{18}) locations. Thirty-two of these locations, with addresses 0 through 31, are reserved for special purposes. The remaining addresses are available for normal core storage.

Instructions cannot be executed from storage locations 0-31. Any attempt to obtain an instruction from these locations will turn on indicator AD, address invalid. Similarly, any attempt to branch to these locations will turn on indicator AD, and the branch and any associated operations will be suppressed. A brief description of the special locations follow. See Figure 6.2-1.

6.2.1 Zeros (0) ($Z)

Location zero is used primarily as a convenient source of zeros for the programmer. Information may be stored in location zero, but any attempt to recover this information in its original form will result in getting an all-zero word. Note, however, that location

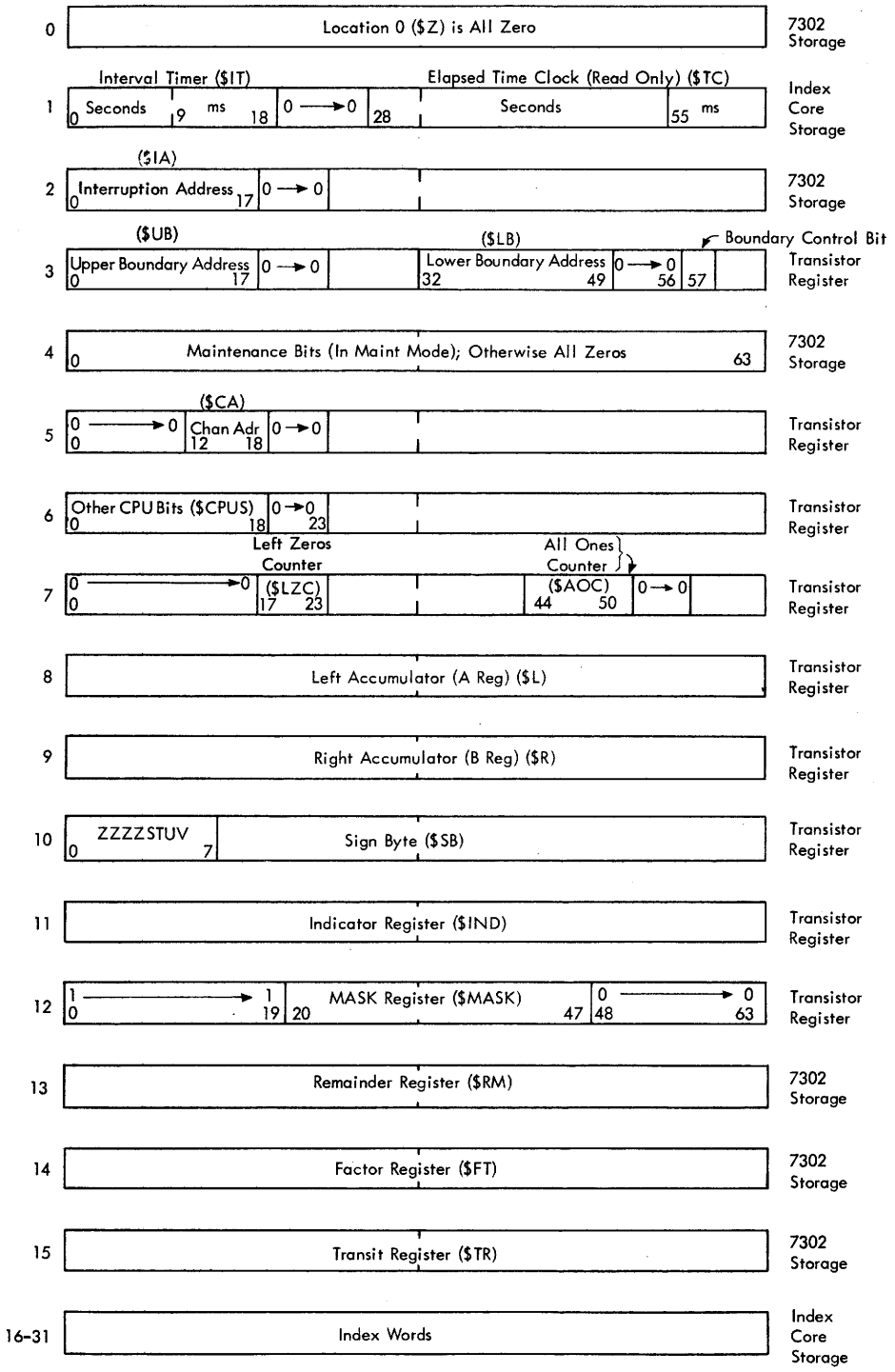


FIGURE 6.2-1. SPECIAL REGISTERS

zero could contain a random mixture of ones and zeros immediately after power has been applied to the system, and that any attempt to extract information from location zero at this time will bring out this meaningless word. Therefore, before using location zero as a source of zeros, the programmer should cause a store operation (of any information) into location zero.

Location 0 is a valid word address in all but the following cases, in which it will cause an address invalid or exchange program check indication:

1. Instructions cannot be executed from location 0.
2. Location 0 is not a valid refill address for control words used in input-output operations.
3. The control word for an input-output operation cannot be taken from location 0.
4. Data cannot be transmitted to or from location 0 in an input-output operation.

6.2.2 Locations 1-15

Addresses 1 through 15 are used for all addressable special registers used by the central processing unit. Except as noted in connection with the individual registers, addresses 1-15 are valid word addresses in all but the following cases, in which they cause an address invalid or exchange program check indication:

1. Instructions cannot be executed from the special registers.
2. The special registers are not valid refill addresses for index or control words.
3. The control word for an input-output operation cannot be located in a special register.
4. Data cannot be transmitted to or from the special registers in an input-output operation.
5. The instruction counter cannot be stored in a special register.

Address 1 is physically located in I unit high-speed core storage. Addresses 2, 4, 13, 14, and 15 are physically located in main core storage. The remaining locations, included in 0-15, are transistor registers.

Interval Timer (1) ($IT)

The interval timer is intended to measure elapsed time over relatively short intervals. It can be set to any value at any time, and indicator TS, time signal, is actuated when the time period has ended.

The value of the interval timer reading is stored in I unit core storage, bit positions 0 through 18 of location 1. It is continually stepped down by pulses originating from a stable oscillator. Each time the oscillator delivers a pulse, the value of the timer is read out, decremented by one, and returned to core storage. The oscillator operates at 1,024 (2^{10}) cycles per second, or a pulse about every millisecond. Bits 0-8 show time in seconds. A full cycle is about 8-1/2 minutes.

When the timer steps from one to zero, it turns on the time signal indicator in the indicator register. The timer does not stop after reaching zero. The next oscillator pulse sets the timer to all ones unless the program in the meantime has changed its contents.

Time Clock (1) ($TC)

The time clock measures time difference or duration over relatively long periods. This clock consists of a number 36 bits long which is continually stepped up by pulses originating from the same 1,024 cycle-per-second oscillator which controls the updating of the interval timer. The two clocks are updated consecutively in the same time interval. The leftmost 26 bits of the time clock measure time in seconds. A full cycle is about 777 days.

The value of the clock occupies bits 28-63 of location 1. Each time the oscillator delivers a pulse it is read out, incremented by one, using the index adder, and returned to I unit core storage.

Both the time clock and interval timer run continually while the computer is under program control. Unlike the interval timer, the time clock cannot be set to any predetermined value. The time clock can be read under program control.

Interruption Address (2) ($IA)

Bits 0-17 of location 2 are used in forming the interruption address and are covered in more detail in Section 6.1.3.

Address Boundaries (3) ($UB) ($LB)

A method of preventing the accidental altering of a certain storage area is provided by the address monitoring system.

The address monitoring system requires the definition of an area in core storage. The two limits of this area are defined by the upper and lower boundary registers. The area starts with the word at the address in the lower boundary register and includes all words in subsequent locations up to, but not including, the word at the address in the upper boundary register. If the contents of the upper boundary register are equal to or smaller than the contents of the lower boundary register, no words are included in the area.

The upper boundary register occupies bit positions 0-17 of location 3. The lower boundary register occupies bit positions 32-49 of location 3. The boundary control bit, which determines whether addresses inside or outside of the defined area are to be protected, is located in bit 57 of location 3.

The actual operation of the monitoring system is presented under "Address Monitoring."

Maintenance Bits (4) ($MB)

The 64 bits of storage location 4 are reserved for maintenance purposes. These bits are active only when the machine is in the maintenance mode. When it is not in this mode, they act like the bits of location 0.

Channel Address (5) ($CA)

The exchange channel responsible for the current settings of the input-output status indicators is identified by the channel address register, occupying bits 12-18 of location 5. The channel address register is a read-only register; any information stored there is lost. In this respect it resembles location 0. The channel address register can be set only by the exchange.

Other CPU Bits (6) ($CPU)

In some systems, several computers or other central processing units may be very closely interrelated, sharing common core storage units or a common exchange. In order to permit a program in one computer to signal others, the other-CPU register, bits 0-18 of location 6, and the CPU signal indicator, CPUS, have been provided. Each of the bits can actuate the CPU signal indicator of one and only one other computer. The system thus provides for up to nineteen other computers. When a bit is set to one by programming, the CPU signal indicator is actuated in the computer associated with that bit. In this manner any computer can actuate the indicator in any other computer, and thus interrupt the other computer's program. When a bit is zero, it has no effect on the associated computer.

The other CPU register is provided only in computers which are to be used in multiple-computer systems. When not present, these bit positions behave like the bit positions of address 0.

Left Zeros and All Ones Counters (7) ($LZC) ($AOC)

Location 7 contains two counters. These counters are set as the result of certain instructions and their operation is explained under those instructions.

Accumulator and Sign Byte Register (8-9-10) ($L) ($R) ($SB)

In many operations an implied operand is a field from the accumulator. The accumulator is a 128-bit register occupying locations 8 and 9. The left half of the accumulator is in location 8, and the right half is in location 9. The sign of the accumulator operand is contained in an associated register, the accumulator sign byte register, an 8-bit register occupying bits 0-7 of location 10.

Indicator Register (11) ($IND)

The indicator register location 11 contains 64 indicators. Each indicator is turned on (set to 1) or off (set to 0) when certain conditions unique to it occur in the computer system. When an indicator is on, the corresponding bit in the mask register is one, and the interruption system is enabled, an automatic interruption will occur. When an indicator causes an interruption, it is automatically turned off. An indicator may also be turned off by a BRANCH ON INDICATOR that tests it.

Bit positions 0-19 of the indicator register can be read, but not loaded. Any attempt to store into these positions has no effect. Bits positions 20-63 may either be read or loaded.

When the indicator register is addressed as the operand of a fetch type instruction, its contents at the start of execution of that instruction are obtained.

Mask Register (12) ($MASK)

Corresponding to each indicator in the indicator register is a bit position in the mask register, location 12. Those indicators for which the corresponding mask bit is zero cannot cause automatic program interruption. Those indicators for which the mask bit is one may cause interruption if the interruption system is enabled.

Bits 0-19 of the mask register are permanently set to one. Bits 20-47 can be set to either zero or one by storing the desired value in these positions. Bits 48-63 are permanently set to zero. Any information stored in bit positions 0-19 or 48-63 of location 12 is lost and is not recoverable.

Remainder Register (13)($RM)

The remainders developed during certain divide operations are placed in the remainder register in location 13. The function of this register is covered in more detail under the divide operations.

Factor Register (14) ($FT)

MULTIPLY AND ADD obtains one of the factors in the multiplication from the factor register as an implied operand. The factor register is a 64-bit register in location 14.

The factor register contents are loaded as an implied operand in LOAD FACTOR.

Transit Register (15) ($TR)

The transit register is used in the execution of specific instructions and is explained under those instructions.

6.2.3 Locations 16-31 ($X0-$X15)

As previously explained, addresses 16 through 31 are index word locations. These locations are contained in the I unit in index core storage.

6.3 ADDRESS MONITORING

In multiprogrammed operation, when several programs are in core storage at one time and are executed in intermixed fashion, it is important that errors in one program be prevented from causing changes in data or instructions belonging to another program. This protection may be provided by monitoring all addresses at which data are to be stored and suppressing the storage if an address lies within the protected area of core storage, as defined by the contents of two boundary registers.

By extending the address monitoring to include addresses from which instructions and data are fetched, it may also be used to verify the proper execution of a program and to detect incorrect instructions. For instance, in program check-out, incorrect instructions or data addresses, especially those which are the result of address computation, may be detected by this means.

The monitoring procedure can be refined by changing boundaries as the program proceeds. This refinement can be used to ascertain that the addresses used in each program section do not belong to those reserved for other program sections.

A protected storage area is defined by placing an address in each of two address boundary registers. This area includes the word specified by the address in the lower boundary register and all words in subsequent locations up to, but not including, the address in the upper boundary register. The address in the upper boundary register is normally larger than that in the lower boundary register. If it is equal or smaller, no words are contained in the protected storage area. The upper boundary and lower boundary registers are located in bits 0-17 and 32-49 of storage location 3.

The boundary control bit, bit 57 of storage location 3, determines which condition will cause an address monitoring signal. When the boundary control bit is zero, a signal is obtained for a comparand inside the specified area. When the boundary control bit is one, a signal is obtained for a comparand outside the specified area. An instruction or operand field that straddles a boundary always gives an address monitoring signal.

The monitoring of references to location 0-31 is independent of the contents of the boundary registers. Locations 1, 2, and 3 are referred to as the permanently protected area of storage. References to locations 0 and 4 through 31, inclusive, never cause an address monitoring signal.

The action of the address monitoring system depends upon the state of the interruption system and upon the masks of the indicators that can be activated by the address monitoring signal. The indicators that can be turned on by the presence of the address monitoring signal are DS (data store), DF (data fetch), and IF (instruction fetch).

When the interruption system is disabled, address monitoring is ineffective. In the disabled mode, core storage protection as specified by the address boundary registers in conjunction with the boundary control bit does not take place, and the address monitoring signal is ignored. As a result, the DS, DF, and IF indicators cannot be turned on when the system is disabled.

When the interruption system is enabled, any reference to the protected storage area causes one of these three indicators to be turned on, the choice depending upon the type of storage reference. The subsequent action depends upon the state of the mask bit. If the mask bit is one, the operation is terminated and an interruption occurs; if the mask bit is zero, the indicator is set, but the operation proceeds. The remaining description of the action of the address monitoring system applies only when the interruption system is in the enabled state.

When an address alarm is caused by a data address used in a store-type operation, indicator DS is turned on. Since this indicator is permanently masked on, storing in core storage is always suppressed, and the operation is always terminated. A store-type operation is one in which the contents of an addressed storage location are subject to change.

When an address alarm is caused by a data address used in a fetch-type operation, indicator DF is turned on. If the DF mask bit is one, the data fetch is suppressed and the operation is terminated. If the DF mask bit is zero, the operation proceeds normally.

A fetch-type operation is one in which the contents of an addressed storage location are read out for use but are not subject to change by the operation.

When an address alarm is caused by the location of the instruction which is to be executed next, indicator IF is turned on. If the indicator is masked for interruption, the interruption occurs before execution of the instruction causing the indication and after execution of all preceding instructions has been completed or terminated. If the branch address of a successful branch operation is such that control would pass into a protected area, the entire branch operation, including any counting, bit changing, or index alteration, is suppressed. The address boundaries are ignored if the mask bit is zero.

The three address alarm indicators, DS, DF, and IF, are permanent. Once turned on, they remain on until they cause an interruption or are turned off by program control.

When an interruption occurs, the resulting interruption address is not monitored for instruction fetch.

All addresses actually used by CPU in the course of an operation are subject to address monitoring. If an address which is associated with an operation is not actually used, it will not be monitored by the contents of the boundary registers. For example, the branch address of a BRANCH may set the IF indicator only if the branch is successful.

6.4 SYMBOLIC ADDRESSING

All programming examples thus far presented have incorporated STRAP symbols for stating the operation to be performed. The addresses, however, have been presented as real numbers. For example, a normalized ADD from location decimal 1000 is presented as +(N), 1000.

When using STRAP symbolic programming, the programmer need not assign specific storage locations to every word that is to be loaded into the computer. Rather, the programmer merely states a starting address. The program and the data are then read into the computer in a sequential manner beginning at that address. However, the programmer must be able to refer to specific instructions or data locations. Thus, if the programmer wishes to branch to a new location in storage, he must have a method of designating that location. This is accomplished by attaching "names" to the desired instructions or data at the time the program is written. For example, the first instruction in a group of instructions might be "named" JOE. The symbolic method of indicating a branch to this instruction is: B, JOE. Not every location need be named even though it is referred to by the program. For example, the statement B, JOE + 2.0 will result in a BRANCH to the location which is two full-words beyond JOE. The inserting of the exact address in the address field of the instruction is performed by the assembly program during the assembly of the subject program.

Remember that all information within the computer is actually in binary form. All symbolic presentations are merely read into the computer as data. Under control of the assembly program (STRAP), the symbolic information is converted to the appropriate binary code as the assembled program.

In addition to the locations which the programmer "names," several others may be referred to symbolically. These locations are fixed within the computer and each location has a specific symbol assigned to it. For example, the left half of the accumulator (location 8.0) is referred to symbolically as $L. These symbols which define specific locations within the system are referred to as system symbols. In previous examples system symbols were used for addressing index registers.

6.5 PROGRAM CONTROL INSTRUCTIONS

A large number of 7030 instructions are primarily used to control the sequence and execution of the program. These instructions do not normally contribute to the system's arithmetic ability, but they do give the computer greater flexibility. BRANCH, for example, does not increase the speed of any particular arithmetic operation. It does, however, allow the computer to use a given section of the program repeatedly, thus greatly reducing the number of storage locations required for a given program. When the more sophisticated CONDITIONAL BRANCH is used, the computer begins to exhibit logical ability. In such operations, the sequence of the program is no longer fixed. Rather, the sequence of the program is determined by the computer itself by investigating the results thus far obtained in executing the program.

Just as the sequence of the program itself is altered by certain instructions, the sequence of operands used in the program is altered by address modification, a function of the index registers. Section 6.5.1 of the manual deals with the instructions that alter the sequence of the program. Section 6.10 covers the instructions used to control the index registers, which in turn affect the sequence of operands.

6.5.1 Program Sequencing

Normally, the operation of the computer is controlled by instructions taken in sequential order. An instruction is fetched from a core storage location whose address is specified by the contents of the instruction counter system. Any required address modification on the instruction is then performed, and when execution of the previous instruction is sufficiently completed, execution of the modified instruction is begun. Instructions are taken in sequences other than normal as a result of branching operations, interruptions, and execution operations.

6.5.2 Branching

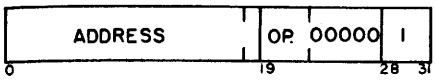
The normal sequence of instructions may be altered by the use of two types of branching instructions: unconditional and conditional. There are three types of conditional branches: indicator, index, and bit branching. Any of these instruction types may cause a new value to be transferred to the instruction counter, thereby causing the instruction sequence to proceed from the newly addressed location.

The addresses between 0.0 and 31.32 are invalid branching addresses. All other available 19-bit addresses are valid.

6.6 UNCONDITIONAL BRANCHING

Five unconditional branching operations provide for normal modification of the branch address by the contents of an index register. Unconditional branching instructions are specified in the half-word format shown below. The address field, bits 0-18, represents

the branch address. Unconditional branching is specified by bits 19-27, where bits 19-21, the operation field, select the particular operation. Bits 28-31 name the index register used for any address modification.



The general format for coding the unconditional branch operations in STRAP notation is: OP, A_{19}(I), where OP is the STRAP symbol of the specific branch to be performed, and A_{19} is a 19-bit branch address which is subject to modification as called for by the (I) field.

6.6.1 Branch (B)

BRANCH replaces the instruction counter contents with bits 0-18 of the effective address of the instruction.

STRAP Notation: B, JOE ($X7)
branch - address - modifying index register
Operation Code (positions 19-21): 010
Indicators Affected: If a branch to a location below location 32 is attempted, the address invalid (AD) indicator (16) is turned on. If the AD indicator is turned on, the BRANCH is performed as a NO OPERATION. A BRANCH with an effective address greater than the uppermost storage location available in the system also turns on indicator AD. A branch to a protected area of storage, including special locations 1, 2, and 3 turns on the instruction fetch (IF) indicator (21), if the interruption system is enabled. If the corresponding mask bit is one, the branch operation is suppressed and interruption occurs.

6.6.2 Branch Relative (BR)

This instruction causes the absolute value of the effective address to be added to the instruction counter contents. Bits 0-18 of the sum replace the instruction counter contents.

After an instruction is fetched and before its execution, the instruction counter contents are stepped to address the next instruction. With branching operations, this stepping is done prior to any attempt to replace the instruction counter contents. For example, if the instruction counter contents are 231.32 and a half-word branch instruction is fetched from that location, before the execution of the instruction, the instruction counter is stepped to 232.00.

STRAP Notation: BR, BUD,($X7)
Operation Code: 011
Indicators Affected: Same as BRANCH

6.6.3 Branch Enabled (BE)

This operation enables the interruption mechanism and then performs the same operations as BRANCH.

STRAP Notation: BE, JOB($X5)
Operation Code: 000
Indicators Affected: Same as BRANCH

6.6.4 Branch Disabled (BD)

This operation disables the interruption mechanism and then performs the same operation as BRANCH.

STRAP Notation: BD, FIX ($X2)
Operation Code: 001
Indicators Affected: Same as BRANCH

6.6.5 Branch Enabled and Wait (BEW)

This instruction provides a means of stopping program execution. The interruption mechanism is enabled, and the instruction counter contents are replaced with bits 0-18 of the effective address of the operation as in BRANCH ENABLED. No further instruction will be executed, however, until after an interruption occurs.

STRAP Notation: BEW, INFO ($X3)
Operation Code: 100
Indicators Affected: Same as BRANCH

6.6.6 No Operation (NOP)

The address and index fields of NO OPERATION are not interpreted, so they cannot cause any indicators to be actuated; no addressable bits or locations are altered. The computer proceeds to the next instruction in sequence.

STRAP Notation: NOP, --
Operation Code: 110
Indicators Affected: None

Notice that the operation code for a NOP differs from that of a BRANCH or BEW by only one bit. Thus, if NOP instructions are placed in the program they can easily be changed to one of the branch operations. This feature may be used to change the free instruction, executed as a result of an interrupt, from an inactive NOP to an active BRANCH. During portions of a program, it is possible that no action is desired when certain indicators are turned on. If any of these indicators are permanently masked 1(0-19) and the interruption mechanism is enabled (to provide address monitoring), interruption occurs even though no action is desired. To have no action performed, the effective interrupt addresses must contain NOP instructions. If later in the program corrective action is to be taken, as the result of turning on one of the indicators, the NOP can be changed to a BRANCH by removing the 1 in position 19 of the instruction. The instruction bits are easily changed by certain VFL instructions.

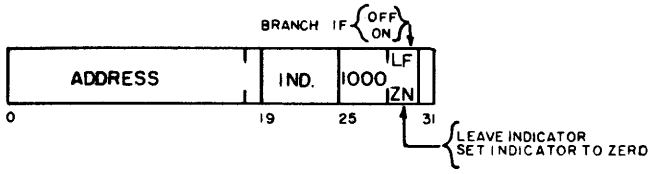
BE, BD, and BEW provide the sole means of controlling the status of the interrupt mechanism. Notice that BEW is especially useful in I-O applications. If the computer has completed all operations on the current data in the system and is reading in new

data, BEW allows the computer to wait until an interrupt indicates that the read operation is complete. The computer may then process the new data. BEW provides the only program method of holding the system in an inactive state.

When a BE or BEW is executed, the interrupt mechanism is turned on before the instruction operation is complete. Therefore, either of these two instructions may cause a program interruption regardless of the previous state of the interrupt mechanism. When BD is executed, the converse is true; that is, the interrupt mechanism is turned off before the instruction operation is completed. Therefore, the execution of the BD instruction cannot cause a program interruption.

6.7 CONDITIONAL BRANCHING

In conditional branching, the success of branching depends upon a test of one of the 64 indicators. Normal modification of the branch address is provided by index register X1 (address 17). The indicator branching instruction is specified in the half-word format shown below. Bits 0-18, the address, represent the branch address. Bits 19-24 specify the number of the tested indicator. Bits 25-30 specify the operation, where bits 29 and 30 serve as modifiers. Bit 31 determines whether or not there will branch address modification using index register X1.



STRAP Format: BIND, B_{19} (K). The symbol BIND represents BRANCH ON INDICATOR. The letters IND are replaced by the symbol representing the particular indicators to be tested. B_{19} is a 19-bit branch address. (K) is a one-bit I field.

6.7.1 Branch on Indicator (BIND)

This operation causes the specified indicator to be tested. If its value matches that of bit 30 of the instruction, bits 0-18 of the effective address replace the instruction counter contents. Whether or not a branching operation is successful, the tested indicator is reset to zero when instruction bit 29 is one; when bit 29 is zero, the tested indicator remains unchanged. Because there are several variations of BRANCH ON INDICATOR, more than one STRAP notation is used to designate the operations.

The STRAP notation and operation code for a BRANCH ON INDICATOR where the branch is to be performed if the indicator is on, and the indicator is to remain in its original state, is shown below. Assume the tested indicator to be result greater than zero (RGZ).

STRAP Notation: BRGZ, JOE
Operation Code (positions 29-30): 01
Indicators Affected: Same as BRANCH (except tested indicator).

To reset the indicator to zero, the suffix Z is added to the STRAP notation.

STRAP Notation: BRGZZ, JOE

Operation Code (29-30): 11

To cause branching if the indicator is off, a Z is prefixed to the indicator abbreviation.

STRAP Notation: BZRGZ, JOE
Operation Code (29-30): 00

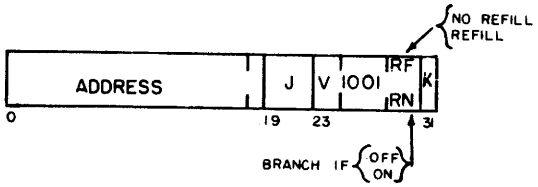
A branch operation which is to take place if the indicator is off and which is to reset the indicator, if it is on, is coded as shown below.

STRAP Notation: BZRGZZ, JOE
Operation Code: 10

Note that the symbols used to define the indicator to be tested actually constitute a type of symbolic addressing. The assembly program interprets BRGZ as a BI instruction with an indicator address of 11.58 (word 11, bit 58), the address of the RGZ indicator. The same instruction may be coded as: B58, JOE.

6.7.2 Index Branching

In index branching, the count field of an index register, J, is counted down and the index count zero indicator, XCZ, is set according to whether or not the count field became zero. This setting of XCZ determines the success of branching. Limited incrementing of the value field of the index register is possible. Normal branch address modification can be provided by index register X1. The index branching instructions are specified in the half-word format shown below. Bits 0-18, the address, represent the branch address. Bits 19-22 specify the index register, J, whose count field will be counted down. Bits 23-30 specify the operation, where bits 23, 24, 29, and 30 are modifiers. Bit 31 determines whether or not there will be branch address modification using index register X1. When bit 29 is zero, the instruction is COUNT AND BRANCH; when the bit is one, the instruction is COUNT, BRANCH, AND REFILL.



STRAP Format: OP, J, B_{19}(K). J specifies the index register to be operated on. (K) is a one-bit I field.

6.7.3 Count and Branch (CB)

The count field of index register J is counted down by subtracting one at bit position 45 of the index word and appropriately setting the index count zero indicator, XCZ. The value field of index register J is altered as indicated by the table below:

Instruction Bits		Code	Description of Index Value Incrementing
23	24		
0	0		No incrementing
0	1	H Add half to value	Add one in bit position 18.
1	0	+ Add one to value	Add one in bit position 17.
1	1	- Subtract one from value	Subtract one in bit position 17.

The incrementing takes into account the sign, bit 24, of the index word's value field. If the value field is minus, incrementing by plus one actually decreases the absolute value of the value field by one.

If, after counting down the count field, the status of the XCZ indicator matches that of instruction bit 30, bits 0-18 of the effective address replace the instruction counter contents.

6.7.4 Count, Branch, and Refill (CBR)

This operation is the same as COUNT AND BRANCH except that, after execution, index register J is refilled if the index count zero indicator is on (count reached zero). The index register is refilled by the contents of the location specified by the refill field of index register J. All available 18-bit addresses larger than and including 16.0 are valid for refilling.

Because there are several variations of COUNT AND BRANCH, more than one STRAP notation is used to represent them, as shown below.

Count and Branch (branch if zero count)
STRAP Notation: CBZ, $X7, JOE
Operation Code: 1001 01

Count, Branch and Refill (branch if non-zero)
STRAP Notation: CBR, $X7, JOE
Operation Code: 1001 10

Count, Branch and Refill (branch if zero)
STRAP Notation: CBRZ, $X7, JOE
Operation Code: 100111

To include modification of the value field with any of the above operations, the symbol +, -, or H is placed after the operation mnemonic.

Count and Branch - Leave Value
STRAP Notation: CB, $X7, JOE
V Field (positions 23-24): 00

Count and Branch - Add Half to Value
STRAP Notation: CBH, $X7, JOE
V Field: 01

Count and Branch - Add One to Value
STRAP Notation: CB+, $X7, JOE
V Field: 10

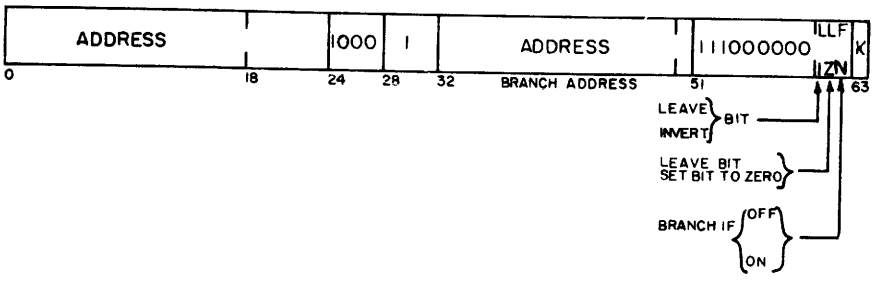
Count and Branch - Subtract one from Value
STRAP Notation: CB-, $X7, JOE
V Field: 11

The variations that modify the value field may be used with any of the CB and CBR instructions.

Indicators Affected: Several indicators may be affected by the CB and CBR instructions. As in the normal branch operations, an attempt to branch to a protected area of storage will turn on indicator IF, if the interrupt mechanism is enabled. An attempt to branch to a non-existent or invalid location will turn on indicator AD. If the interrupt mechanism is enabled, an attempt to refill from a protected area of storage will turn on indicator DF. If the count field of the specified index register reaches zero as a result of the CB or CBR operation, the XCZ indicator is turned on. The status of the value field is reflected by indicators XVLZ (value less than zero), XVZ (value zero) and XVGZ (value greater than zero).

6.7.5 Bit Branching

Any bit in storage can control branching with BRANCH ON BIT. The choice of the tested bit may be varied by address modification, and the bit may be set to zero, inverted, or set to one if desired. The bit branching instruction is specified in the full-word format shown below. Bits 0-23 constitute the bit address. Bits 24-27 and 51-62 designate the operation, where bits 60-62 are modifiers. Bits 28-31 name the index register to be used for bit address modification. Bits 32-50 represent the branch address. Bit 63 determines whether or not there will be branch address modification using index register X1.



STRAP Format: OP, A24(I) B_{19} (K)

6.7.6 Branch on Bit (BB)

The bit specified by the effective bit address is tested. If its status matches that of bit 62 of the instruction, bits 0-18 of the effective branch address replace the instruction counter contents. Following this operation, whether branching occurs or not, the tested bit is set according to the table shown.

Instruction Bits		Tested Bit Will Be
60	61	
0	0	Unchanged
0	1	Set to zero
1	0	Inverted
1	1	Set to one

Although any bit in storage can be tested by BRANCH ON BIT, those bits in read-only locations will remain unchanged when there is any attempt to change them. The variations of BRANCH ON BIT are listed below in STRAP format.

Branch on Bit (branch if bit is one - leave bit)
STRAP Notation: BB, 500.57, JOE
Operation Code (positions 60-62): 001

Branch on Bit (branch if bit is zero - leave bit)
STRAP Notation: BZB, 500.57, JOE
Operation Code: 000

Branch on Bit (branch if one - set bit to zero)
STRAP Notation: BBZ, 500.57, JOE
Operation Code: 011

Branch on Bit (branch if one - invert bit)
STRAP Notation: BBN, 500.57, JOE
Operation Code: 101

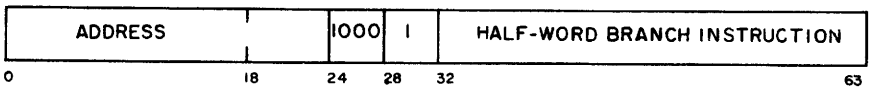
Branch on Bit (branch if one - set bit to one)
STRAP Notation: BB1, 500.57, JOE
Operation Code: 111

The above bit-modifying functions are also permissible with the BZB operations. The symbol, 1, Z, or N is merely added to the BZB symbol. For example, a branch on zero bit with the tested bit being set to one is coded as BZB1.

Indicators Affected: If a branch to a protected area of storage is attempted (successful), while the interrupt mechanism is enabled, indicator IF is turned on. Similarly, if the interrupt mechanism is enabled and the bit to be tested is located in a protected area of storage, indicator DF is turned on. If the address of the tested bit is invalid or if the successful branch is to an invalid address, indicator AD is turned on.

6.8 STORING OF THE INSTRUCTION COUNTER

Any half-length branching operation can be converted to a full-word instruction and also provide for storing the instruction counter in any specified storage location. This is done by prefixing any of the branch instructions except BRANCH ON BIT with the operation STORE INSTRUCTION COUNTER IF, as in the format shown below. Bits 0-23 constitute the storage address. Bits 24-27, together with the half-word branch instruction operation code, specify the full-word operation. Bits 28-31 name the index register used for storage address modification. Bits 32-63 specify the half-word branch instruction used.



STRAP Format: OP, A_{19}(I)

6.8.1 Store Instruction Counter If (SIC)

This instruction may always be prefixed to one of the half-word branching instructions. It is executed if and only if the associated branching is successful. When it is executed, it causes the instruction counter contents to be stored in bits 0-18 of the half-word specified by bits 0-18 of the effective storage address. Other bits of the half-word are not disturbed. The contents of the instruction counter before branching are stored. Thus, the stored instruction counter always contains an address one half-word greater than that of the BRANCH instruction.

STRAP Notation: SIC, MEM; B, JOE. Store the contents of the instruction counter into location MEM and branch to location JOE.

Indicators Affected: In addition to the indicator affected by the associated BRANCH, the SIC instruction may result in the turning on of indicator DS if the interrupt system is enabled and the storage address is in a protected area of storage.

6.9 EXECUTION INSTRUCTIONS

Three types of instruction sequencing have already been explained: normal sequencing by the stepping of the instruction counter, in which the program keeps control of sequencing; branching, in which the program gives control to a second program; and interruption, in which a new program takes control. Because it is often desirable to permit one program to execute the instructions of a second program without losing control to the second program, a fourth type of instruction sequencing is provided, which lends control. This is furnished by means of the two execution instructions.

EXECUTE directly specifies the location of a second instruction, termed the subject instruction, which is executed. EXECUTE INDIRECT AND COUNT specifies a location which imitates the instruction counter, a pseudo-instruction counter. The pseudo-instruction counter addresses the location of a single subject instruction, which is fetched and executed. After the subject instruction has been fetched, but prior to its execution, the pseudo-instruction counter is stepped.

In either of the execution instructions, the subject instruction may be of any type and is performed according to the normal rules for an operation of its type. In order that the main program may have full control at all times, however, the subject instruction is not allowed to change the state of the interruption system or replace the contents of the instruction counter. The execution exception indicator (EXE) is turned on by any branch operation of a subject instruction in which the condition for a successful branch is met. When indicator EXE is actuated, the branch and all associated operations are not performed. The suppressed actions include the disabling in BRANCH DISABLED, the enabling in BRANCH ENABLED or BRANCH ENABLED AND WAIT, the counting and refilling in COUNT, BRANCH and REFILL, and so on.

If an execution operation specifies an execution instruction as its subject instruction, the subject instruction of the second-level execution operation is obtained. If this, in turn, is still another execution instruction, the procedure is carried to succeeding levels, until an instruction that is not an execution instruction is obtained.

When an execution operation is initiated in the main program, the instruction counter is stepped in normal fashion as in all operations of the main program. However, it is never stepped during the execution of any subject instruction even if the subject instruction is again one of the execution instructions.

In general, when the interruption system is disabled, interruption cannot occur and address monitoring is suspended. In order that control may remain entirely with the main program during an execution operation, two special exceptions are made. First, address monitoring is always in force during EXECUTE and EXECUTE INDIRECT AND COUNT, except for the address of the location of the first-level pseudo-instruction counter. Second, at the completion, termination, or suppression of an execution operation, interruption is always permitted.

Addressing, such as provided by the execute operation, is called indirect addressing. The normal addressing scheme, where the address field of the instruction addresses an operand, is known as direct addressing. Still another method, where the address field of the instruction is used as data, is called immediate addressing. Immediate addressing is available with certain instructions that are described in other sections of the manual.

The execute operations are specified in the half-word format shown below. Bits 0-18 contain the indirect address. The operation code is contained in bits 19-27 with bits 22-27 being identical for both execute operations. The index register to be involved in any address modification is specified by the I field, position 28-31.

STRAP Format: OP, A_{19} (I)

6.9.1 Execute (EX)

Bits 0-18 of the effective address form the address of the location of a subject instruction that is fetched and executed. Subject instructions of EXECUTE may occupy any addressable location. An effective address of EXECUTE that is below 32.00 (including addresses between 1.00 and 3.32) is not monitored by the address monitoring mechanism; that is, indicators IF and AD cannot be actuated by these addresses.

STRAP Notation: EX, JOE
Operation Code (position 19-21): 010
Indicators Affected: As described above, indicator IF is not set as a result of fetching a subject instruction from addresses 1, 2, or 3. Similarly, indicator AD is not actuated as the result of executing a subject instruction from addresses 31 or lower. If at the end of one millisecond no subject instruction has been executed, the unended sequence of addresses (USA) indicator (17) is turned on. This condition may be caused by two EXECUTE instructions referring to each other as subject instructions.

If any attempt is made to change the instruction counter contents, indicator EXE (#18) is turned on.

6.9.2 Execute Indirect and Count (EXIC)

Bits 0-18 of the full word addressed by bits 0-17 of the effective address are used as a pseudo-instruction counter. The contents of the pseudo-instruction counter address the location of a subject instruction that is fetched and executed. After the subject instruction has been fetched but prior to its execution, the pseudo-instruction counter is stepped. If the subject instruction is half-length, the pseudo-instruction counter is stepped once; if full-length, twice.

When EXECUTE INDIRECT AND COUNT is a subject instruction in a multiple-level execution operation, each pseudo-instruction counter involved is appropriately stepped between the fetching and executing of its corresponding subject instruction.

Pseudo-instruction counters may occupy any of the index registers as well as main core storage locations, but cannot occupy locations addressed between 0 and 15. Subject instructions of EXECUTE INDIRECT AND COUNT may occupy any available locations with addresses 32.00 and greater.

STRAP Notation: EXIC, JOE
Operation Code: 011
Indicators Affected: Indicator AD is turned on if the pseudo-instruction counter is located in an address below 16.0. If the subject instruction is located at an address below 32.0, no indicators are set but the subject instruction is treated as a NOP. If, at the end of one millisecond, no subject instruction has been executed, indicator USA is actuated. If any attempt to change the instruction counter is made, indicator EXE is actuated.

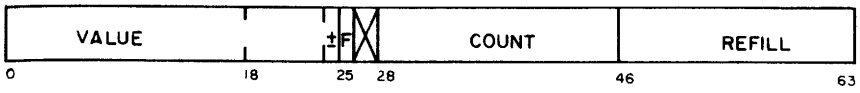
Programming Note: When interruption occurs following an execution operation with the interruption system disabled, it is caused only by the actuated indicator of highest priority whose mask bit is one. Further interruptions cannot occur until either the system is enabled or another execution operation is completed, terminated, or suppressed. Examination for other actuated indicators can be programmed into the correction routine.

Any actuated indicator whose mask bit is one may cause interruption at the completion, termination, or suppression of an execution operation.

6.10 INDEX ARITHMETIC

The three fields of an index word, as illustrated below, are intended for three distinct purposes. The value field contains the quantity which may be combined with the instruction address in order to obtain the effective operand address in address modification. The value can be changed by increment operations. The count field registers the number of times an increment is to be applied. The contents of the count field are reduced by one whenever a count is specified. The count field contains no sign. On reaching zero, the count proceeds to $262,143(2^{18}-1)$. The refill field contains the address of a new index word that can replace the original index word. A refill operation occurs only if it is specified by an instruction. The refill can be made conditional on the count's reaching zero or it can be specified to occur unconditionally.

A refill address of zero will replace the index word with all zeros. The instruction set takes full advantage of the use of the index fields for the above purposes.



The set of operations provided for index arithmetic can be divided into seven groups: direct index arithmetic, immediate index arithmetic, refill operations, count and branch operations, named index loading, multiple index loading, and indirect index loading.

Direct index arithmetic permits loading, storing, incrementing, and comparing of index quantities. The operand is specified in the instruction by the effective address. The effective address may be obtained by modifying the instruction address part with an index quantity. The operand is obtained from storage or placed in storage.

Immediate index arithmetic permits loading, incrementing, and comparing of index quantities. The operand is contained in the instruction itself and not subject to modification by indexing.

The count and branch instructions combine an index arithmetic operation with a conditional branch operation. The branch is conditioned by the result of an index count.

The remaining groups of index arithmetic permit loading of an index register only.

The index word to be modified by index arithmetic is specified in the field marked J, instruction bits 19-22. When one of the numbers 1-15 is specified in field J, the corresponding index word X1-X15 will be used in the operation. These words are in storage locations 17-31. When the field J is 0, storage location 16 is used in the operation. This location is named X0.

X0 cannot be used directly for address modification, since a zero I field in such an instruction specifies no indexing. However, X0 may take part in any index arithmetic instruction, because a zero J field always refers to location 16, X0.

6.10.1 Index Indicators

Eight indicators are used to describe the result of index arithmetic. These are the index flag indicator XF, the index result indicators XCZ, XVLZ, XVZ, and XVGZ, and the index comparison indicators XL, XE, and XH.

The index flag indicator is set in any index operation that specifies an index in the field J. These include index arithmetic and count and branch operation. This indicator is also set in the refill operations, according to the index specified in the address field of the instruction. The index result indicators are set in all cases in which the index flag indicator is set, except the index comparison operations. The indicators are not set when an index is used for address modification or to specify the count in a transmit operation.

The indicators describe the index after the index modification is completed, but before any refill operation conditioned by the modification takes place. In REFILL and REFILL ON COUNT ZERO, the indicators describe the index before the refill operation takes place. The particular conditions applying to each indicator are summarized below.

Index Flag (XF): This indicator is set to one when the index flag bit, bit 25, is one. It is set to zero when the index flag bit is zero.

Index Count Zero (XCZ): This indicator is set to one when the count field is zero. It is set to zero when the count field is not zero.

Index Value Less than Zero (XVLZ): This indicator is set to one when the value field is non-zero and negative. It is set to zero when the value field is zero or positive.

Index Value Zero (XVZ): This indicator is set to one when the value field is zero. It is set to zero when the value field is not zero.

Index Value Greater than Zero (XVGZ): This indicator is set to one when the value field is non-zero and positive. It is set to zero when the value field is zero or negative.

The index comparison operations compare the value or the count field with a quantity specified by the address. The index comparison indicators XL, XE, and XH described below are set for the comparison operations instead of the index result indicators. The names of the index comparison indicators refer to the index specified in the J field of the instruction as it is compared with the quantity of the operand.

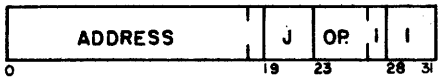
Index Low (XL): This indicator is set to one when the field in the index specified by J is lower than the comparand. It is set to zero when the index field is equal to or higher than the comparand.

Index Equal (XE): This indicator is set to one when the field in the index specified by J is equal to the comparand. It is set to zero when the index field is lower or higher than the comparand.

Index High (XH): This indicator is set to one when the field in the index specified by J is higher than the comparand. It is set to zero when the index field is equal to or lower than the comparand.

6.11 DIRECT INDEX ARITHMETIC

The direct index arithmetic operations make use of the format shown below. The format includes two index fields. The rightmost field, marked I, is used in standard fashion to produce an effective operand address. In most of these operations, 19 bits of the effective address are used in order to permit addressing of half words. Two exceptions use only 18 bits of the effective address because they require full-word operands. The index field J specifies the index register upon which the operation is performed.



STRAP Format: OP, J, A_{19} (I)

6.11.1 Load Index (LX)

The index word specified by J is replaced by the full word specified by bits 0-17 of the effective address.

STRAP Notation: LX, $X5, JOE
Operation Code (positions 23-27): 00001

6.11.2 Load Value (LV)

The value field, bits 0-24, of the index word specified by J is replaced by bits 0-24 of the half word specified by bits 0-18 of the effective address.

STRAP Notation: LV, $X5, JOE
Operation Code: 00 011

6.11.3 Load Count (LC)

The count field, bits 28-45, of the index word specified by J is replaced by bits 0-17 the half word specified by bits 0-18 of the effective address.

STRAP Notation: LC, $X5, JOE
Operation Code: 00101

6.11.4 Load Refill (LR)

The refill field, bits 46-63, of the index word specified by J is replaced by bits 0-17 of the half word specified by bits 0-18 of the effective address.

STRAP Notation: LR, $X5, JOE
Operation Code: 00111

6.11.5 Store Index (SX)

The index word specified by J is stored in the full-word memory location specified by bits 0-17 of the effective address.

STRAP Notation: SX, $X5, JOE
Operation Code: 1001

6.11.6 Store Value (SV)

The value field, bits 0-24, of the index word specified by J replace bits 0-24 of the half-word specified by bits 0-18 of the effective address.

STRAP Notation: SV, $X5, JOE
Operation Code: 10011

6.11.7 Store Count (SC)

The count field, bits 28-45, of the index word specified by J replace bits 0-17 of the half word specified by bits 0-18 of the effective address. Bits 18-24 of the half word are set to zero.

STRAP Notation: SC, $X5, JOE
Operation Code: 10101

6.11.8 Store Refill (SR)

The refill field, bits 46-63, of the index word specified by J replaces bits 0-17 of the half-word specified by bits 0-18 of the effective address. Bits 18-24 of the half-word are set to zero.

STRAP Notation: SR, $X5, JOE
Operation Code: 10111

6.11.9 Add to Value (V+)

The address part, bits 0-24, of the half-word addressed by bits 0-18 of the effective address is added to the value field, bits 0-24, of the index word specified by J. Addition is algebraic, taking into account the sign, bit 24, of both quantities. A one bit in position 24 of the addressed operand is interpreted as a minus sign.

STRAP Notation: V+, $X5, JOE
Operation Code: 01011

6.11.10 Add to Value and Count (V+C)

This operation is identical to ADD TO VALUE, except that in addition the count field of the index word specified by J is counted down by one.

STRAP Notation: V+C, $X5, JOE
Operation Code: 01101

6.11.11 Add to Value, Count and Refill (V + CR)

This operation is identical to ADD TO VALUE AND COUNT, except that in addition the index word specified by J is refilled if the count reaches zero as a result of this operation. The refill does not occur if the count was zero prior to the operation, because reducing a zero count by one sets the field to all ones.

STRAP Notation: V+CR, $X5, JOE
Operation Code: 01111

The index flag and index result indicators are set in all of the preceding direct index arithmetic operations. The index comparison indicators are unchanged.

6.11.12 Compare Value (KV)

The address part, bits 0-24, of the half-word addressed by bits 0-18 of the effective address is compared with the value field, bits 0-24, of the index word specified by J. Comparison is algebraic, taking into account the sign, bit 24, of both addresses. Positive and negative zero are considered equal.

STRAP Notation: KV, $X5, JOE
Operation Code: 01001

6.11.13 Compare Count (KC)

Bits 0-17 of the half-word specified by bits 0-18 of the effective address are compared with the count field, bits 28-45, of the index word specified by J. The comparison is unsigned and for magnitude only.

STRAP Notation: KC, $X5, JOE
Operation Code: 11001

The index flag and index comparison indicators are set in each of the last two operations; the index result indicators are not set in these operations.

LOAD INDEX and STORE INDEX involve transmission of full words. In all other operations the effective operand address refers to a half-word. Of this half-word, either the first 18 or the first 25 bits are used, depending on the operation to be performed. The remaining bits are ignored and unchanged. When instructions or data, rather than index words, are used as the operand in increment and compare operations, proper care should be taken that the left 25 bits are usable as a signed quantity. Note also that the first 25 bits of the half word have been numbered 0-24, but the actual field may be either 0-24 or 32-56.

The overflow which may occur in index arithmetic is ignored.

6.11.14 Programming Examples

Example A: Using only instructions which have been previously explained, consider the following programming problem. In 2,000 locations of storage, starting at location 500, the address of each location is to be placed in the address field. For example, positions 0-17 of location 500 are to contain the number 500. After these 2,000 locations have been "labeled," they are to be read out and checked to verify that they contain the proper value. The following program steps accomplish the described objective. Note that the same index register is not used for both the checking and generating of addresses.

Name	Instruction	Remarks
START	LX, $X5, FIRST	Load XR5 with index word from location FIRST.
PUT	SV, $X5, 0.0 ($X5)	Store address in address field of proper location.
MOD	V+C, $X5, ONE	Add the 1 from location ONE to the value field and reduce the count by one.

Name	Instruction	Remarks
RET	BZXCZ, $X5, PUT	If less than 5000 addresses have been produced, return to PUT.
PREP	LX,$X10, FIRST	Load XR10 for use in checking.
CHECK	KV, $X10, 0.0, ($X10)	Compare predicted address with generated address.
TEST	BZXE, OUT	ERROR
	CB+, $X10, CHECK	OK. Modify value by +1, reduce count by one. If less than 5000 locations have been tested, return to CHECK.
	B, NEXT	BRANCH to next section of program.
FIRST	XW, 500.0, 2000, FIRST	VALUE = 500 COUNT = 2000.
ONE	VF, 1.0	Value field with 1 in position 17.
OUT		Start of error print-out routine
OUT		
OUT		
OUT		
NEXT		Beginning of next section of program.

The STORE VALUE at location PUT stores the contents of the value field of XR5 in the storage location with an address equal to the value of XR5. Thus, on the first pass, the value 500 is stored in location 500. At location CHECK, the same procedure is used for comparing the contents of storage to the value of XR10.

The STRAP symbol, VF, at location ONE instructs the assembly program to compile the following field as a 24-bit signed value field. Therefore, the notation VF, 1.0 results in a word of all zeros with the exception of position 17 which contains a one.

With the exception of the special instruction STORE VALUE IN ADDRESS, the only means of setting the interval timer is by use of the instructions SV, SC, and SR.

When using instructions that alter the address field of other instructions, particular attention must be made to the length of fields. For example, if the SV instruction refers to the address field of a half-word instruction which contains only 18 or 19 bits, portions of the operation code as well as the address are altered. The operation code of a STORE INDEX contains a one bit in position 23. If a value field with zeros in position 23 and 24 is stored into this instruction, the STORE VALUE is changed to a LOAD VALUE. Although this feature may be conveniently used when programming, care must be taken to prevent the accidental changing of operation codes.

Example B. The programmer wants to find each positive value field included in index registers 2-15. He wants to place the sum of these value fields into location LOAD 2. Assume that the sum does not exceed the capacity of a value field (24 bits).

Name	Instruction	Remarks
	LX, $X1, LOAD 1	$X1 = 0.0, 14, 0
	LX, $X0, LOAD 2	Clears index word 0 to all zeros.
REPEAT	V+, $X0, 18.0 ($X1)	Add the value field of the next index word to the value field of index word 0.

Name	Instruction	Remarks
	KV, $X0, LOAD 2	Compare the sum of the value fields added up to this point with the previous sum of positive value fields.
	BXL, MINUS	Branch if present sum is less than previous sum.
	SV, $X0, LOAD 2	Present sum is of positive value fields. Store sum in location LOAD 2
CHECK	CB+, $X1, REPEAT	Add 1 to value field of index word 1. Branch to repeat addition of next index word.
	B, (Next Routine)	All index word value fields have been checked.
MINUS	LV, $X0, LOAD 2	Present sum is less than previous sum. Restore previous sum into the value field of index word 0.
	B, CHECK	Check if all value fields have been added.
LOAD 1	XW, 0.0, 14, 0	Index word to be used for counting and address modification.
LOAD 2	XW, 0.0, 0, 0	Index word used to clear index register 0.

6.12 IMMEDIATE INDEX ARITHMETIC

The immediate index arithmetic operations make use of the format shown below.

Index field J contains the address of the index to which the operation applies. The address part contains the operand. A 19-bit operand can be specified. In some operations, only 18 of these 19 bits are used. In this class of operations, the index field I is not available and no address modification by indexing takes place.

STRAP Format: OP, J, A_{19}

6.12.1 Load Value Immediate (LVI)

Bits 0-18 of the value field of the index word specified by J are replaced by the address part, bits 0-18, of the instruction. The remaining bits of the value field, bits 19-24, are set to zero. In so doing, the sign of the address contained in the value field is made positive.

STRAP Notation: LV1, $X10, BILL
Operation Code (positions 28-31): 0001

6.12.2 Load Count Immediate (LCI)

The count field, bits 28-45, of the index word specified by J is replaced by bits 0-17 of the address part of the instruction.

STRAP Notation: LC1, $X10, BILL
Operation Code: 0010

6.12.3 Load Refill Immediate (LRI)

The refill field, bits 46-63, of the index word specified by J is replaced by bits 0-17 of the address part of the instruction.

STRAP Notation: LRI, $X10, BILL
Operation Code: 0011

6.12.4 Load Value Negative Immediate (LVNI)

Bits 0-18 of the value field of the index word specified by J are replaced by the address part, bits 0-18, of the instruction. Bits 19-23 of the value field are set to zero. The sign bit, bit 24, of the value field is set to one. In so doing, the sign of the address contained in the value field is made negative.

STRAP Notation: LVNI, $X10, BILL
Operation Code: 1001

6.12.5 Add Immediate to Value (V+I)

The address part, bits 0-18, of the instruction is added to bits 0-18 of the value field of the index word specified by J. Zeros are added to bits 19-24 of the value field. Addition is algebraic, taking into account the sign, bit 24, of the value field.

STRAP Notation: V+I, $X10, BILL
Operation Code: 0101

6.12.6 Add Immediate to Value and Count (V+IC)

This operation is identical to ADD IMMEDIATE TO VALUE except that, in addition, the count field of the index word specified by J is counted down by one.

STRAP Notation: V+IC, $X10, BILL
Operation Code: 0110

6.12.7 Add Immediate to Value, Count and Refill (V+ICR)

This operation is identical to ADD IMMEDIATE TO VALUE AND COUNT except that, in addition, the index specified by J is refilled if the count reached zero.

STRAP Notation: V+ICR, $X10, BILL
Operation Code: 0111

6.12.8 Subtract Immediate from Value (V-I)

The address part, bits 0-18, of the instruction is subtracted from bits 0-18 of the value field of the index word specified by J. Zeros are subtracted from position 19-24 of the value field. Subtraction is algebraic, taking into account the sign, bit 24, of the value field.

STRAP Notation: V-I, $X15, AL
Operation Code: 1101

6.12.9 Subtract Immediate from Value and Count (V-IC)

This operation is identical to SUBTRACT IMMEDIATE FROM VALUE except that, in addition, the count field of the index word specified by J is counted down by one.

STRAP Notation: V-IC, $X15, AL
Operation Code: 1110

6.12.10 Subtract Immediate from Value, Count and Refill (V-ICR)

This operation is identical to SUBTRACT IMMEDIATE FROM VALUE AND COUNT except that, in addition, the index specified by J is refilled if the count reached zero.

STRAP Notation: V-ICR, $X15, AL
Operation Code: 1111

Note that the immediate indexing operations provide for value field subtraction as well as addition. Although no subtract operations are provided with direct indexing, bit 24 of the addressed operand is considered as a sign position. Thus, subtraction is possible with both types of operations.

6.12.11 Add Immediate to Count (C+I)

Bits 0-17 of the address part of the instruction are added to the count field, bits 28-45, of the index word specified by J. Both quantities are unsigned.

STRAP Notation: C+I, $X7, KEN
Operation Code: 0000

6.12.12 Subtract Immediate from Count (C-I)

Bits 0-17of the address part of the instruction are subtracted from the count field, bits 28-45, of the index word specified by J. Both quantities are unsigned.

STRAP Notation: C-I, $X7, KEN
Operation Code: 1000

The index flag and index result indicators are set in all of the preceding immediate index arithmetic operations. The index comparison indicators are not changed.

6.12.13 Compare Value Immediate (KVI)

The address part, bits 0-18, of the instruction is compared with bits 0-18 of the value field of the index word specified by J. Zeros are compared with bits 19-24 of the value field. Comparison is algebraic, taking into account the sign, bit 24, of the value field. Positive and negative zero are considered equal.

STRAP Notation: KVI, $X6, JIM
Operation Code: 0100

6.12.14 Compare Value Negative Immediate (KVNI)

The address part, bits 0-18, of the instruction is compared with bits 0-18 of the value field of the index word specified by J. Zeros are compared with bits 19-23 of the value field. A one, or negative sign, is compared with the sign, bit 24, of the value field.

STRAP Notation: KVNI, $X6, JIM
Operation Code: 1100

6.12.15 Compare Count Immediate (KCI)

Bits 0-17 of the address part of the instruction are compared with the count field, bits 28-45, of the index word specified by J. The comparison is unsigned and for magnitude only.

STRAP Notation: CKI, $X6, JIM
Operation Code: 1010

The index flag and index comparison indicators are set in each of the last three operations; the index result indicators are not set in these operations. To set the index flag and index result indicators according to the status of an index without modifying that index, an ADD IMMEDIATE TO VALUE operation with zero address part can be used as one of several alternatives.

6.12.16 Programming Note

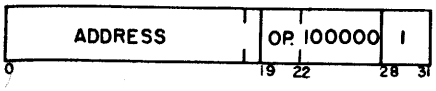
The immediate indexing instructions are used mostly for the initial setting up and updating of index quantities. These instructions are desirable in that a separate location of storage is not required for the operand.

No programming example for immediate indexing is presented here, but various immediate index instructions are used in other programming examples throughout the manual.

In the KVI operation, note that a 19-bit address is compared against a 25-bit value field. An equal result, therefore, can never be obtained unless positions 19-24 of the value field contain zeros.

6.13 REFILL OPERATIONS

The refill operations make use of the format shown below. These operations make possible the refilling of any index word in memory. The index word specified by the I field is used in standard fashion to form an effective address. Eighteen bits of the effective address are used to specify the full word in storage which is to be refilled. The refull operation can be made conditional on the count of the addressed index word being zero, or it can be specified to occur unconditionally. A refill operation replaces an index word with the word addressed by the refill field of the original index word.



STRAP Format: OP, A_{19} (I)

6.13.1 Refill (R)

The full word addressed by bits 0-17 of the effective address is replaced with the full word which is addressed by the refill field, bits 46-63, of the word at the effective address.

STRAP Notation: R, WORD
Operation Code (positions 19-21): 000

The index flag and index result indicators are set accor ding to the original contents of the word at the effective address.

6.13.2 Refill on Count Zero (RCZ)

This operation is the same as REFILL except that the refill will occur only if the count field, bits 28-45, of the addressed word is zero.

STRAP Notation: RCZ, WORD
Operation Code: 001

The index count zero indicator may be used to find out whether the refill operation took place.

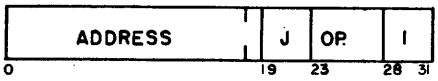
6.13.3 Programming Example

The count fields of index registers 3-6 are to be analyzed. If an index word count field of less than ten is encountered, that index word is to be refilled from the specified refill address. The program is:

Name	Instruction	Remarks
	LX, $X2, INDEX	$X2 = 0.0, 4, 0
START	LX, $X1, 19.0 ($X2)	Place index word to be checked next into index register 1.
	KCI, $X1, 10.0	Compare the count field contained in index register 1 with the quantity 10.
	BZXL, LOOP	Branch if count field is not less than 10.
	R, 19.0 ($X2)	Count field is less than 10. Refill index register.
LOOP	CB+ $X2, START	If the last index register has not been checked, repeat routine.
	B (Next Routine)	
INDEX	XW, 0.0, 4, 0	

6.14 NAMED INDEX LOADING

Fifteen index registers are directly available to the programmer for use in address modification, but some applications may require a larger number of index registers for frequent loading and storing of index words. To simplify the loading and storing of index words, named index loading has been provided. When this method of index loading is used, X0 contains the storage address, or name, of the index word last loaded into one of the registers X1-X15 by a named index load. When a named index load is specified, the contents of the index register are first stored in the memory location specified by X0. The name of the new index word is then placed in X0, and the word itself is placed in the specified index register. The named index load operation RENAME makes use of the format shown below. The field J specifies the index register which is to be stored and loaded. The field I is used in standard fashion to form an effective address.



STRAP Notation: OP, J A19 (I)

6.14.1 Rename (RNX)

The index word in the index register specified by the field J is stored in the memory location addressed by the refill field of X0. The refill field of X0 is then replaced by bits 0-17 of the effective address. The remaining bits of X0 remain unchanged. Next, the index word specified by J is replaced by the word in storage addressed by the new refill field of X0. When either the old or the new name of the index refers to one of the locations 1-31, the instruction is executed as NO OPERATION and indicator AD, address invalid, is turned on.

STRAP Notation: RNX, $X4, DOG
Operation Code (positions 23-27): 11111

The index flag and index result indicators are set according to the new contents of the index register specified by J. RENAME makes it possible to use any storage word conveniently as an index quantity.

6.14.2 Programming Example

Starting at location DATA are 100 data words. Each word is to be relocated in storage according to the high-order 18 bits of the word. For example, if the 18 high-order positions of a data word contain the value 5,000, the word is to be stored in location 5000. Assume positions 18-24 of all the words contain zeros. The program follows.

Name	Instruction	Remarks
PREP	LX, $X14, 0.0	Clear XR14
	LX, $X15, 0.0	Clear XR15
	LCI, $X14, 100.0	Place a count of 100 in XR14.
GO	L(BU,64,8),DATA ($X14),64	Load full word into accumulator.
	LV, $X15, 8.0	Place address part in value field of XR15.

Name	Instruction	Remarks
	ST(BU,64,8),0.0 ($X15),64	Store accumulator in location called for by XR15.
	CB +, $X14, GO	Count down count field; add 1 to value; if count is non-zero, branch to GO.
NEXT	First instruction of next routine	

In the above example index register 14 is used to locate the next data word. Index register 15 is used to determine where each data word is to be stored.

To illustrate the use of RENAME, assume the only index register available to the programmer is XR15, and the program is to be rewritten using only one index register.

Name	Instruction	Remarks
PREP	LRI, $X0, 0.0	Clear refill field of XR0.
	RNX, $X15, WORDS	Use XR15 for fetching operands.
GO	L(BU,64,8)DATA ($X15),64	Load full word into the accumulator.
STORE	RNX, $X15, ADDS	Use XR15 for storing.
	LV, $X15, 8.0	Place address part in value field of XR15.
	ST(BU,64,8),0.0($X15),64	Store accumulator in location called for by XR15.
FETCH	RNX, $X15, WORDS	Use XR15 for fetching operands.
	CB+, $X15, GO	Count down count field; add 1 to value; if count is non-zero, branch to GO.
WORDS	XW, 0.0, 100	Value = 0 count = 100
ADDS	XW, 0.0, 0	Value = 0 count = 0

RENAME at location STORE stores the contents of XR15 back into location WORDS. This instruction also loads XR15 from location ADDS. Index register 15 is then available for use in storing the accumulator contents. RENAME at location FETCH stores the contents of XR15 (value used for storing) in location ADDS; it also loads XR15 from location WORDS. The index register is then available for fetching the next operand. Note that the information loaded into XR15 as a result of the instruction at location STORE is never used. The same instruction, however, stores the old contents of XR15 into location WORDS for future use.

6.15 MULTIPLE INDEXING

In some applications index quantities are derived as the sum of a selected number of basic quantities. As the basic quantities are modified, the derived quantities should change accordingly. It may be convenient to update only the basic quantities and form a derived sum when that particular sum is required. To facilitate this mode of operation, it should be possible to specify in one instruction a group of index registers and indicate which registers of the group should be used in forming the derived sum. This mode of indexing is called multiple indexing. The index registers which participate in multiple indexing are X0-X15.

In multiple indexing, bits 0-15 of the address part of the instruction are used to indicate which index registers participate in the operation. When bit position n contains a one, the register Xn participates. When bit position n contains a zero, the register Xn does not participate. Bits 16-19 of the address part of the instruction are ignored.

The multiple indexing operation makes use of the instruction format shown below. The index field J contains the address of the index to which the LOAD VALUE WITH SUM applies. No address modification is available for this operation.

STRAP Format: OP, J, A1, A2, A3, -----An

6.15.1 Load Value with Sum (LVS)

The address parts, bits 0-24, of the index registers which participate are added and replace bits 0-24 of the index word specified by J. The addition is algebraic, taking into account the sign, bit 24, of all quantities. Bits 25-63 of the index word specified by J remain unchanged. If bits 0-15 of the address part of the instruction are all zero, the value field of the index register specified by J is cleared.

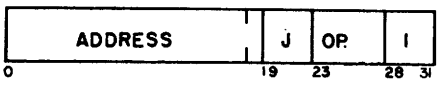
STRAP Format: LVS, $X10, $X1, $X2, $X3
OPERATION Code (positions 28-31): 1011

The index flag and index result indicators are set according to the new contents of the index register specified by J.

6.16 INDIRECT ADDRESSING

The effective address of an instruction is normally used to address the data upon which the instruction operation is performed. In another mode of addressing, the effective address can be used to address another instruction. This instruction, in turn, contains an operand address and index which can be used to from a second level effective address. In some applications, it is desirable to address the operand by the second level effective address rather than by the effective address of the original instruction. The process of using a first level effective address to obtain a second level effective address is called indirect addressing. The second level effective address, of course, can refer to another instruction of which, again, the effective address can be obtained. In this manner it is possible to extend indirect addressing through many levels.

Indirect addressing is made possible by LOAD VALUE EFFECTIVE whose format is shown below.



STRAP Format: OP, J, A_{19} (I)

6.16.1 Load Value Effective (LVE)

The value field, bits 0-24, of the index word specified by J is replaced by the effective address of the instruction found at the location specified by the effective address. When

the instruction found at the location specified by the effective address is another LOAD VALUE EFFECTIVE, the process is repeated with that instruction. The final effective address is placed in the index register specified by the field J of the original instruction.

STRAP Notation: LVE, $X9, FIX
Operation Code: 11011

All instructions encountered in a multiple level indirect addressing process, except for the last one, are LVE's. The instruction which produces the final effective address is any instruction other than LVE. It is assumed to be either a half-length instruction or the first half of a full-length instruction. The final effective address is produced according to the rules that apply to that instruction. In particular, the lengths of the address and index fields are determined from the instruction class to which the instruction belongs. No check is made for invalid operation codes.

Although the address part of the second half-word of TRANSMIT and SWAP is 19 bits, only the first 18 bits of the address field are used when the second half of those instructions is encountered in the process. If the second half-word of any other type of full-length instruction is encountered, the full 19 bits of the second address are used.

When the first half of any variable field length instruction terminates the operation, the address modification mode of indexing is assumed, even though the full instruction may use the progressive indexing mode.

Even though the instruction class of the final instruction is determined in order to obtain the proper effective address, the instruction operation is not executed. LOAD VALUE EFFECTIVE, which extends the indirect addressing to another level, is always executed. The index register in which the final effective address is to be stored is specified by the index field J of the original LVE. The fields J of all subsequent instructions encountered in the operation are ignored.

In indirect addressing, it is possible that addresses refer to each other in such a way that they form a loop. As a result, the computer cannot finish its operation and no interruption is accepted. This situation would be the result of a programmer's error. In order to prevent the machine from being locked up in this way, LOAD VALUE EFFECTIVE is terminated, if still active, after one millisecond and indicator USA, unended sequence of addresses, is turned on. Neither index register J nor the index flag and index result indicators are altered. The index flag and index result indicators are set according to the new contents of the index specified by J.

6.16.2 Programming Example

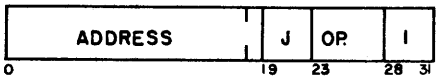
Suppose that a floating point instruction caused an interrupt. The instruction counter was stored in location IC. As part of the interruption correction routine, it is necessary to obtain the operand of the floating point operation that caused the interrupt, placing the operand in the accumulator. The desired action will be accomplished by the following program steps. Remember that if the instruction counter is stored as the result of an interrupt, the value stored is actually the address of the instruction following the "interrupting" instruction. Because the interrupt in this example was caused by a floating point instruction, the stored instruction counter contents are one half-word greater than the "interrupting" instruction.

Name	Instruction	Remarks
FIND	LX, $X1, IC	Load stored instruction counter into XR1.
	LVE, $X15, -0.32 ($X1)	Place effective address of "interrupting" instruction in XR15. XR1 value is one half-word too great. Thus, -0.32 corrects effective address.
	L(U), 0.0 ($X15)	Loads "offending" operand into the accumulator.

6.17 ADDRESS INSERTION

The length of the address field of an instruction may be 18, 19, or 24 bits, depending upon the operation class of the instruction. In order to facilitate the insertion of an address of proper length in an instruction, STORE VALUE IN ADDRESS is provided. This operation replaces the address field of an instruction with a field of proper length. The field is obtained from the value field of an index word. The instruction addresses a half word and assumes it to be either a half-length instruction or the first half of a full-length instruction. The length of the address field is determined from the instruction operation code.

STORE VALUE IN ADDRESS makes use of the format shown below. The index field J specifies the index register from which the address is to be obtained. The location of the instruction in which the address will be inserted is given by the effective address of the instruction.



STRAP Format: OP, J, A_{19} (I)

6.17.1 Store Value in Address (SVA)

The first 18, 19, or 24 bits of the value field of the index word specified by J replace the address field of the instruction specified by the effective address.

STRAP Notation: SVA, $X11, TEMP
Operation Code: 11101

The index flag and index result indicators are set according to the contents of the index register specified by J.

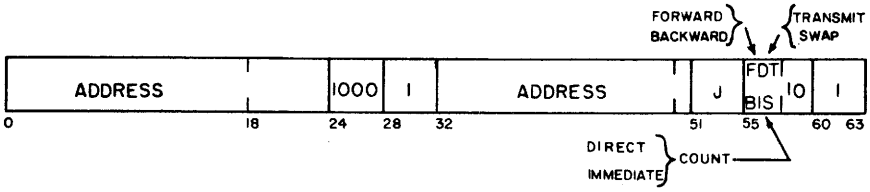
Although the address part of the second half-word of TRANSMIT and SWAP instructions is 19 bits, only the first 18 bits of the address field are replaced when the second half-word is referred to by STORE VALUE IN ADDRESS. If an address is stored in second half-word of any other type of full-length instruction, the full 19 bits of the second address are replaced.

6.18 INTERNAL DATA TRANSMISSION

Instructions are available to transmit data from one storage location to another. One word or a group of words may be transmitted in a single operation. The number of words to be transmitted is specified by a count. The count may be specified in the

instruction or contained in an index word. When the count is specified in the instruction it is called an immediate count. Up to 2^{18}-1 (262,143) full words may be transmitted in one operation.

Transmission may be unidirectional or bidirectional. In unidirectional transmission, data are moved from one storage location to another. After completion of the operation, the data remain unchanged in the original locations. The data which were originally in the receiving storage locations are destroyed. In bidirectional transmission, data are interchanged. Following this operation, both sets of data are preserved, but they have interchanged their storage locations. Unidirectional transmission of full words is specified by TRANSMIT. Bidirectional transmission of full words is specified by SWAP.



STRAP Format: OP, J, A18(I), A'18 (I')

A full-length instruction is used for data transmission operations. The format contains two addresses. Each address can be modified by the index word specified by the I field in its respective half-word. The two effective addresses obtained as a result of the modification specify the starting addresses of the two storage areas which participate in the word transmission. Only 18 bits of each effective address are used.

Bits 55-57 of the data transmission instructions are used as modifiers. The choice between unidirectional and bidirectional transmission is specified by modifier bit 57. A zero specifies unidirectional transmission; a one specifies bidirectional transmission. The choice between an immediate or a direct count is specified by modifier bit 56. A zero specifies a direct count; a one specifies an immediate count. The choice between forward and backward transmission is specified by modifier bit 55. A zero specifies forward transmission; a one specifies backward transmission.

In direct count operations, the field J, bits 51-54, specifies an index register. The count field, bits 28-45, of the index word in this register is used as a full word count. The contents of the index register are not altered during the data transmission. Rather, the J field is used only for the initial count. The updated count is maintained in a separate register in I unit. A count field of zero is interpreted as 2^{18}. Because this many words cannot be transmitted without encountering an invalid address, such a count is never valid. It is not detected, however, until the data transmission actually encounters an invalid or protected address. A zero J field refers to location 16, X0.

In immediate count operations, the word count is contained in the J field, bits 51-54. A zero count field indicates the maximum count of 16 full words.

In forward transmission, the sending and receiving addresses of each successive word transmitted are obtained by incrementing the previous addresses by one. The starting addresses specified in the instruction are, therefore, the lowest addresses of the two storage areas.

In backward transmission, the sending and receiving addresses of each successive word transmitted are obtained by decrementing the previous addresses by one. The starting addresses specified in the instruction are, therefore, the highest addresses of the two storage areas.

6.18.1 Transmit (T)

The number of full word specified by the count is transmitted from the area which starts at the location addressed by bits 0-17 of the left effective address to the area which starts at the location specified by bits 0-17 of the right effective address.

STRAP Notation: T, $X1, SEND.0($X2), REC.0($X3)

6.18.2 Swap (SWAP)

The number of full words specified by the count is transmitted from the area which starts at the location addressed by bits 0-17 of the left effective address to the area which starts at the location specified by bits 0-17 of the right effective address. At the same time, the full words of the area which starts at the right effective address are transmitted to the area which starts at the left effective address.

STRAP Notation: SWAP, $X1, MIX . 0($X2), CHANGE . 0($X3)

6.18.3 Programming Notes

The STRAP mnemonics used in writing the various TRANSMIT and SWAP instructions are presented below.

Operation	Mnemonic
Transmit Forward	T
Transmit Forward Immediate	TI
Transmit Backward	TB
Transmit Backward Immediate	TBI
Swap Forward	SWAP
Swap Forward Immediate	SWAPI
Swap Backward	SWAPB
Swap Backward Immediate	SWAPBI

The difference between forward and backward transmission is important when sending and receiving areas overlap. During transmit, the overlapped area must be used for sending before it is used for receiving. Otherwise, the original information in the overlapped area will be destroyed before it is transmitted.

All full-length instructions are assumed to have a 24-bit address part in the left half-word. Because data transmission instructions are full length, 24 bits of the left half-word of the instruction are used in forming the effective address. However, only 18 bits of the effective address are used by these operations.

Example 1: Storage locations 112, 113, and 114 contain the floating point constants a, b, and c. These constants are to be rotated so that their order is b, c, and a. The instruction is:

SWAPI, 2, 112.0, 113.0

Because an immediate count is specified, the instruction results in two SWAP's. At the completion of the first SWAP the operands appear in the order b, a, and c. At the completion of the second SWAP the operands appear in the desired order b, c, and a.

Example 2: Starting at location 100, the contents of 16 storage locations are to be transmitted to the next higher location. The contents of 100 are to be placed in 101, the contents of 101 are to be placed in 102, and so on. This action can be provided by the following instruction.

TBI, 0, 115.0, 116.0

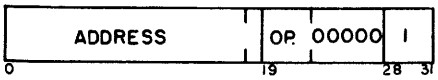
Notice that backward transmission must be used so that each location is used for sending before it is used for receiving. The all-zero J field in an immediate operation provides a count of 16.

Example 3: Assume that location 500 contains a particular configuration which, for test purposes, the customer engineer wishes to store in location 501 through 600. This result can be achieved by the following program steps:

Name	Instruction	Remarks
PREP	LCI, $X7, 100.0	Place count of 100 in XR7.
DO IT	T, $X7, 500.0, 501.0	500 to 501, 501 to 502, 502 to 503, etc.
NEXT	Continuation of program	

6.18.4 Store Zero (Z)

Several instructions may set the bits in a storage field to zero. The value, count, and refill fields of an index word can be set to zero with an IMMEDIATE LOAD. With CONNECT TO MEMORY, using connective 0000, a variable storage field may be set to zero. To facilitate setting all the bits in a full storage word to zero, STORE ZERO is provided. This instruction causes an all-zero word to be stored in the location specified by bits 0-17 of the effective address. The STORE ZERO is contained in the half-word format shown below.



STRAP Notation: Z, ERASE ($X5)
Operation Code (19-27): 100100000

6.19 FLOATING POINT ARITHMETIC

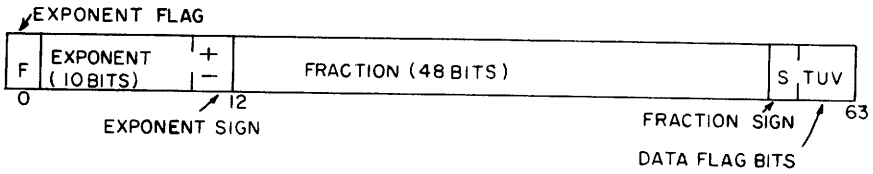
A floating point number consists of a signed exponent ±E, and a signed fraction ±F. The quantity expressed by this number is the product of the fraction and the number 2

raised to the power of the signed exponent, or: $\pm F \cdot 2^{\pm E}$. The exponent is expressed as a binary integer and the fraction is expressed as a binary number having a binary point to the left of the high-order digit.

The 7030 system provides maximum processing speed for floating point operations. Simplification of the floating point instruction set is achieved through full utilization of the uniform nature of floating point data. The necessary instruction information requires only a half-word, or two floating point instructions per storage word, thus increasing program storage efficiency and reducing the number of storage accesses. Uniform information such as bit address, field length, addressing mode, byte size, radix, and accumulator offset are implied by the operation and thus need not be specified as required in VFL arithmetic. Operations on the floating point fractions are performed at high speed in the parallel arithmetic unit.

6.19.1 Data Format

Floating point numbers are represented in storage in the following full-word format.



The exponent field uses the first 12 bits of the data word, bit positions 0-11. Bit position 0 is called the exponent flag, EF. Bit positions 1-10 are called the exponent magnitude, EM. Bit position 11 is called the exponent sign, ES. The exponent sign is set to 0 for positive values and set to 1 for negative values of the exponent. The exponent magnitude, EM, is a binary integer ranging from 0 to 1,023, inclusive. The exponent flag is set to 0 for the values of EM within the range. EF is set to 1 when EM exceeds this range. In reality, the exponent flag is the high-order position of the exponent field. It is called the flag and handled in a special manner merely to avoid exceeding the capacity of the arithmetic units. Generally speaking, the presence of a flagged exponent indicates that the exponent is either so large or so small that further use of this operand may exceed the capacity of the machine. If the original operand has a flagged exponent it is said to be a propagated flag. If the flag is produced by the current operation, it is said to be generated.

The standard operational range for exponents is from +1,023 through -1.023. This range is called the normal (N) range. N range values have magnitudes for normalized floating point numbers ranging between 2^{+1024} and 2^{-1024}, or approximately 10^{+308} and 10^{-308}.

The exponent flag gives two additional ranges of numbers: exponent flag positive, XFP, and range and exponent flag negative, XFN. The XFP range may be considered to have the properties of infinitely large numbers. This range corresponds to all values with exponents greater than +1.023. To visualize a number in this range, picture a 48-bit binary whole number followed by approximately 1,000 zeros. The XFN range may be considered to have the properties of the number zero. A number in this range corresponds to values with exponents less than -1,023. The relative size of a number in the XFN range may be realized by visualizing a 48-bit binary fraction which is at least 1,023 positions to the right of the binary point.

The fraction field uses the next 49 bits of the data word, bit positions 12-60. Bit positions 12-59 are the fraction magnitude, F. Bit position 60 is the fraction sign, S. The binary point associated with the fraction is defined to be to the left of the high-order position, bit position 12. This means that a 1, in bit position 12, has the numerical value of one-half.

The data flag bits, T, U, and V, occupy bit positions 61, 62, and 63. These bits can be set by the programmer and, when set to 1, turn on the associated indicators. The four bits, S, T, U, and V, are treated as the sign byte for floating point operations. In the accumulator sign byte register they occupy bit positions 4, 5, 6, and 7.

The floating point ranges specified above may be visualized through Figure 6.19-1, in which F represents the binary fraction and f represents the decimal equivalent fraction.

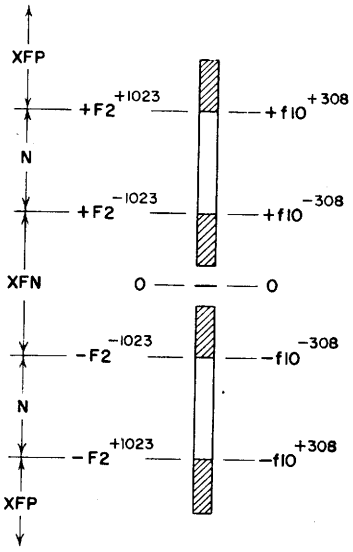
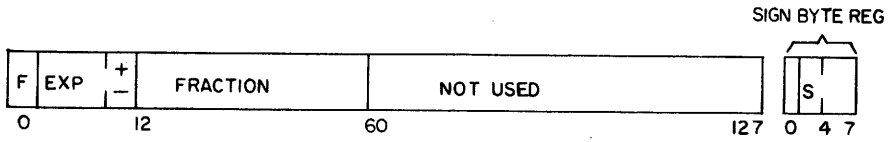


FIGURE 6.19-1. FLOATING POINT RANGES

The shaded areas represent the ranges of exceptional conditions that are handled by the system. These ranges may be considered as numerical buffers which assist in avoiding numerical difficulties encountered by exceeding the capacity of the system.

In the accumulator, floating point numbers are contained in the format shown below.



The exponent field is in bit positions 0-11. The fraction magnitude is in bit positions 12-59. The fraction sign bit does not appear in the accumulator register, but in bit position 4 of the accumulator sign byte register. Bit positions 5-7 of the accumulator sign byte register may contain the data flag bits associated with the information in the accumulator. Accumulator bit positions 60-63 are not used and remain unchanged during normal floating point operations. The contents of the right half of the accumulator

are neither used nor changed by a normal floating point operation unless address 9 is used as the addressed operand. Bit positions 0-3 of the accumulator sign byte register are not used by any floating point operation. They will not be changed by any floating point operation unless address 10 is used as the addressed operand.

The accumulator is used as an implied operand for most operations. It is possible, however, to use the accumulator as the addressed operand as well as the implied operand. The left half of the accumulator has address 8; the right half of the accumulator has address 9; the accumulator sign byte register has address 10.

In operations other than floating point, the contents of the left half of the accumulator, bit positions 0-63, participate when address 8 is used as the effective operand address. In floating point instructions, however, when address 8 is used as the effective operand address, accumulator bit positions 0-59 and sign byte register bit positions 4-7 are used as the operand. The four sign byte bits replace the four low-order bits of the accumulator operand . Bits 60-63 of the accumulator do not serve as operand bits.

6.19.2 Data Flag Bits

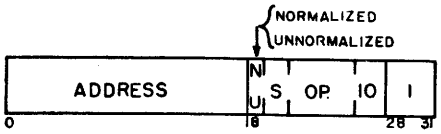
In some problems it may be desirable to mark some of the data to indicate the requirement for special handling. The flag bits of the sign bytes associated with VFL and floating point data permit such indicative marking. The indicators associated with data flags permit special action when these flag bits are encountered.

Bit positions 61-63 of a floating point word are used as data flag bits. These three bits are designated as the T, U, and V bits. When a floating point operand is entered into the accumulator by means of the operations LOAD WITH FLAG or LOAD DOUBLE WITH FLAG, the operand flag bits replace the accumulator flag bits in the sign byte register bit positions 5-7. The operations LOAD and LOAD DOUBLE set the data flag bit positions of the sign byte register to zero. All other floating point operations leave the accumulator flag bits unchanged unless address 10 is used in a to-storage operation. Flag bits of an addressed operand are changed only by operations of the store type, which replace the flag bits in storage by the accumulator flag bits.

The flag bits of a floating point data word set the data flag indicators, TF, UF, and VF for every operation involving a data fetch. This includes all floating point operations except stores, SHIFT FRACTION and ADD EXPONENT IMMEDIATE.

6.19.3 Instruction Format

Floating point instructions are contained in the half-word format shown below.



STRAP Format: OP (dds), A_{18} (I)

The data description code (dds) in the STRAP format is used to define the operation as normalized or unnormalized.

The address portion, bit positions 0-17, is the location of the full word floating point operand. Bit position 18 is used as the normalization modifier. Bit positions 19 and 20 are used as the sign modifiers. The operation code is specified in bit positions 21-27. Of these, bit positions 26 and 27 contain the fixed code (10) to indicate that the instruction belongs to the floating point class. The five remaining bits of the operation code, positions 21 through 25, allow 32 operations to be specified. Twenty-nine of these operations are used. The operand address can be modified by the value field of the index register specified in bit positions 28-31.

6.19.4 Normalization

In floating point instructions, normalized or unnormalized operation is specified through use of the normalization modifier, instruction bit position 18. Normalized operation is specified when this bit is zero; unnormalized operation is specified when this bit is one. In STRAP notation, normalized or unnormalized operation may be specified by an N or U in the data description field. In all instructions except ADD TO EXPONENT, if no modifier is specified, STRAP will interpret the instruction as normalized operation.

When normalized operation is specified, the initial operands used in the operation need not be in normalized form, but the result of the operation is usually normalized. The exceptions to providing a normalized result are discussed with each operation. When normalization occurs it may involve a left shift of the fraction to eliminate high-order zeros, or it may involve a right shift to insert a high-order one bit when the arithmetic operation produces an overflow bit in the fraction. Afer the shift the result fraction is truncated (adjusted) to 48 bits. When normalization of the result requires shifting, the exponent is changed by the amount of the shift. A right shift is added to the exponent; a left shift is subtracted from the exponent. An exception to the rules for normalization occurs when the result fraction is zero; then the normalization is suppressed and no change, due to normalization, is made in the result exponent.

When unnormalized operation is specified, the arithmetic operation result fraction is truncated to 48 bits without a normalization cycle. High-order zeros in the fraction are not eliminated. If a fraction overflow bit is produced, the extra bit is lost and the lost carry indicator, LC, is turned on. The details of unnormalized operation are discussed with each operation.

Normalization, when specified by the instruction, usually applies to the result of the operation. This is called post-normalization. In the arithmetic operation of division, another form of normalization may be used. This is called prenormalization and is applied to one or more of the operands specified in the operation. During division both the divisor and dividend may be prenormalized. Prenormalization of the divisor occurs as a part of the arithmetic procedure and is independent of the normalization modifiers. Prenormalization of the dividend is required when a normalized quotient is to be obtained. During addition the operand having the smaller exponent may be shifted right with its exponent increased by the amount of the shift, until the exponents of the two operands agree in magnitude and sign. This right shifting is called preshifting.

6.19.5 Sign Control

The effective sign of an operand used during instruction execution can be determined in three ways. Two operation modifiers, bit positions 19 and 20 of the instruction, are

associated with the floating point operations to provide flexible sign control. The third way is through the operation code, as in ADD TO MAGNITUDE, and this action is described under the various operations.

Instruction bit position 19 is called the absolute sign modifier. The absolute sign modifier applies to the storage operand when the data are fetched from storage. In store operations, the modifier applies to the number which is placed in storage. When the modifier bit is zero, the sign of the operand is used as it exists prior to further modification by the negative sign modifier. When the modifier bit is one, the sign of the operand is considered to be positive and the original sign is ignored. This assumed positive sign is subject to further modification by the negative sign modifier.

Instruction bit position 20 is called the negative sign modifier. The negative sign modifier applies to the operand which is not changed in the operation. In operations which change the accumulator contents, it applies to the operand from storage. In operations which change storage, it applies to the operand from the accumulator. In compare operations, this modifier applies to the operand from storage. When this modifier bit is zero, the sign of the operand, as modified by the operation code or absolute sign modifier, is used unchanged. When this modifier bit is one, the sign of the operand, as modified by the operation code or absolute sign modifier, is inverted. The negative sign modifier, when one, has the effect of changing algebraic additions to subtractions or changing the sign of the result of multiplications, divisions or stores.

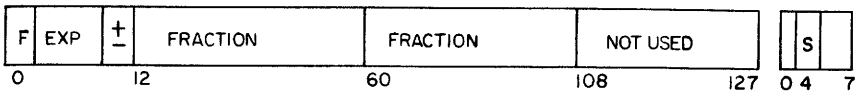
6.19.6 Single and Double Precision Arithmetic

In some applications, a fraction length greater than 48 positions may be required, to express a quantity with great precision. The exponent merely locates the binary point. The maximum number of significant digits in any operand is limited by the length of the fraction field. Arithmetic operations involving normal 7030 floating point operands are called "single precision" operations. When a single precision floating point word cannot express a quantity with sufficient precision, multiple precision operations are required. The floating point instruction set contains a number of operations providing double precision results.

Single precision and double precision operations differ in the number of bits used in the operands and developed in the final result fraction. The exponent field, in either case, has one EF bit, ten EM bits, and one ES bit.

In single precision arithmetic each operand has 48 fraction bits. The final result of a single precision operation also contains 48 fraction bits. During the process of obtaining the final result, however, an intermediate result is developed in the arithmetic units. This intermediate result may have more than 48 fraction bits. For example, during addition, a 96-bit sum fraction and a possible overflow bit is obtained as an intermediate result. During multiplication, a 96-bit product fraction is obtained. These intermediate result fractions are adjusted to form a final result fraction of 48 bits. In single precision operations, only accumulator bit positions 12-59 are used as an operand or to contain the final results.

In double precision arithmetic, the accumulator contains a 96-bit fraction as illustrated.



The storage operand has 48 bits in the fraction. The result, appearing in the accumulator, has a 96-bit fraction. As with some single-precision operations, the intermediate result of a double precision operation is a 96-bit fraction and a possible fraction overflow bit. DIVIDE DOUBLE does not produce a 96-bit intermediate result fraction; but another described in detail under that instruction.

6.19.7 Noisy Mode

During normalization, the fraction of a floating point number is shifted left. This shifting continues until the high-order bit position of the fraction contains the leading one bit of the fraction. As the fraction is shifted left, the vacated low-order positions are replaced with zeros. This process may result in a loss of significance throughout the course of a program. To determine whether the loss of significance is great enough to produce invalid results, the noisy mode of operation is provided.

Noisy mode operation provides for the introduction of ones, rather than zeros, in the low-order fraction positions during the left shift associated with normalization. By operating a program in standard mode (zeros) and then in noisy mode, the programmer can analyze the difference in the results. If the difference is great enough, the programmer may decide to solve the problem differently.

The choice of standard or noisy mode operation is left up to the programmer. A selection of either mode is made by setting or clearing the noisy mode (NM) indicator. When the indicator is zero, standard operation is used; when it is set to one, noisy mode operation is used. Noisy mode operation is used only when normalization is also specified. Unnormalized floating point operations are not affected by the noisy mode.

6.19.8 Zero Definition

A floating point number having a zero fraction may be considered in two ways. If the exponent of the number is in the XFN range (infinitely small), the entire number may be considered as zero and the exponent and sign are immaterial. Zeros of this type are called XFN zeros.

Floating point values with zero fractions and normal range exponents, called "order of magnitude zeros," may result from the cancellation of two non-zero numbers, each representing only a limited amount of accuracy. The result of this cancellation is not a valid zero and, therefore, the exponent and sign of the result fraction and fraction sign become the final result. If normalization is specified, it will be suppressed.

6.19.9 Range Definition

Three classes of exponent ranges, previously outlined, are classified as XFP, N, XFN. The following table defines these ranges.

Range	Exponent Flag	Exponent Sign	Definition
XFP	1	0	Exponent $\leq$ 1,024 Number considered = ∞
N	0	0 or 1	+1,024 > Exponent < -1,024
XFN	1	1	Exponent $\leq$ -1,024 Number considered = 0

The result of a floating point operation may have a flagged exponent from two distinct occurrences. First, the original operand exponents, when operated on, may produce a result exponent in either the XFP or XFN range. For example, consider a number with an exponent of +1,000 multiplied by another number with an exponent of +1,000. Assuming the fractions to have like signs, the result has an exponent of +2,000 and places the number in the XFP range. A flagged exponent so produced is termed "generated."

The second type of flagged exponent is "propagated," that is, it results when one, or both, of the original operand exponents are flagged before the operation is performed. This may force a flagged exponent to be placed in the result.

The handling of flagged exponents can vary with individual operations. For this reason, a more detailed description of flagged exponent handling is included with the particular operations.

6.19.10 Indicators

The indicators which may be turned on by floating point operations can be divided into two groups. The first group of indicators can be turned on by variable field length operations, as well as floating point operations, while the second group of indicators are specifically associated only with floating point operations.

Indicators are either permanent or temporary. The permanent indicators are turned on by certain specified operation conditions. Once on, they stay on until turned off by an interruption or by programming. The temporary indicators are made either zero or one, depending on the causing condition. These indicators are changed when a new operation is performed which sets the indicators. The temporary indicators remain unchanged for operations with which they are not associated.

A brief description of the indicators that may be affected by floating point operations follows.

Indicators Set by VFL or Floating Point Operations

Lost Carry (LC): This indicator is turned on when a fraction overflow occurs in the unnormalized mode of operation. The indicator may also be turned on as a result of an unnormalized DIVIDE DOUBLE if the low-order one bit of the remainder is lost,

as described in greater detail under "DIVIDE DOUBLE." Another method of setting the lost carry indicator is provided by SHIFT FRACTION. The LC indicator is turned on, during this operation, if a one bit is shifted past bit position 12 of the accumulator. This indicator is permanent.

Partial Field (PF): This indicator is turned on in unnormalized divide operations when the magnitude of the dividend is equal to, or greater than, the magnitude of the divisor. This indicator is permanent.

Zero Divisor (ZD): This indicator is turned on for a divide operation if the divisor is all zeros. If this condition exists, the division is suppressed.

Data Flags (TF, UF, VF): These indicators are set according to the data flag bits contained in the sign byte of the floating point storage operand. These indicators are temporary.

To-Memory Operation (MOP): This indicator is turned on if the operation is a "to-storage" type. This type includes stores and ADD TO MEMORY. For any other floating point operations, the indicator is turned off. The indicator is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are turned on according to the sign and magnitude of the fraction result, including the overflow bit, of any floating point operation except comparison operations. These indicators are temporary.

Indicators Set Only by Floating Point Operations

Imaginary Root (IR): This indicator is turned on if the operand of a STORE ROOT is negatively signed. If the absolute sign modifier in STORE ROOT is a one, the IR indicator cannot be turned on. This indicator is permanent.

Lost Significance (LS): This indicator is turned on if the intermediate result fraction, including overflow bit, of a floating point addition or shift operation is zero. Exceptions to this occurrence are:

1. If both operands in a floating point addition have zero fractions prior to preshifting.
2. If the zero fraction is a forced zero because of an ADD TO MAGNITUDE.
3. If the result exponent contains a propagated flag.

Preparatory Shift Greater than 48 (PSH): This indicator is turned on if the difference between operand exponents, for an addition operation, is greater than 48. Both exponents have to be in the "N" range, however, in order to set the PSH indicator. This indicator is also turned on if the product of a MULTIPLY AND ADD has a generated exponent flag and its use, during the addition portion of the operation, leads to an exponent difference of more than 48. The indicator is not turned on if either operand exponent is in the XFP range or if both operand exponents are in the XFN range. The indicator is permanent.

Exponent Flag Propagated (XPFP): This indicator is turned on if the result of a floating point operation has a propagated exponent flag ($-1,024 \leq$ Exponent $\geq +1,024$). This indicator is permanent.

Exponent Overflow (XPO): This indicator is turned on if the result exponent has a generated exponent flag and an exponent sign of "0" (exponent $\geq$ +1,024). The indicator is permanent.

Exponent Range High (XPH): This indicator is turned on if the result exponent is less than +1.024 but greater than, or equal to, +512 (+1,024 > Exponent $\geq$ +512). This indicator is permanent.

Exponent Range Low (XPL): This indicator is turned on if the result exponent is less than +512 but equal to, or greater than, +64 (+512 > exponent $\geq$ +64). The indicator is permanent.

Exponent Underflow (XPU): This indicator is turned on if the result has a generated exponent flag with an exponent sign of "1" (exponent $\leq$ -1,024). The indicator is permanent.

Zero Multiply (ZM): This indicator is turned on if the product of a floating point MULTIPLY is an order of magnitude zero. However, it is not turned on if the above conditions are met but an XFN range exponent is generated. The indicator is temporary.

Remainder Underflow (RU): This indicator is turned on if the remainder exponent of a DIVIDE DOUBLE has a generated exponent flag and an exponent sign of "1" (remainder exponent $\leq$ -1,024). If the remainder exponent has a propagated exponent flag, this indicator is not turned on. The indicator is permanent.

Noisy Mode (NM): This indicator is affected only by programming it on or off. When it is on, with normalization specified, floating point operations operate in the noisy mode.

6.20 FLOATING POINT INSTRUCTIONS

6.20.1 Add (+)

The operand specified by the effective address is algebraically added to the operand in the left half of the accumulator. The exponent and fraction of the sum replaces accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The fraction sign of the addressed operand is modified by the sign modifiers prior to the addition. The sign of the sum replaces the accumulator sign, bit position 4, of the accumulator sign byte register. Bits 0-3 and 5-7 of the accumulator sign byte register remain unchanged.

STRAP Notation: +(N), JOE = Normalized
+(U), JOE = Unnormalized
Operation Code (21-25): 00000

Before the actual addition is performed, the exponents of both operands must be made equal. To equalize the exponents, first, the exponent of the storage operand is subtracted from the exponent of the accumulator operand. The difference between the exponents is then algebraically added to the smaller exponent. Changing the smaller exponent modifies the value of the corresponding floating point number. Therefore, the fraction associated with the smaller exponent is preshifted right, by the amount of the exponent difference. The fractions are then added to form an intermediate result. The exponent of the intermediate result is the larger of the two original operand exponents.

The fraction of the intermediate result is 96 bits in length, plus a possible overflow bit. For the fraction addition, each operand fraction is extended to 96 bits. The low-order bits, of the fraction that was preshifted, take part in the addition, and appear in the intermediate sum. Consider the following example:

Two floating point numbers, A and B are to be added. (All values are represented as octal quantities.)

A = Exponent +50, Fraction + 0012345676543210
B = Exponent +42, Fraction + 2525252525252525

The exponent of A is larger than the exponent of B. Therefore, the difference in the exponents, 6, is added to the exponent of B. For B to retain its original value, the fraction is preshifted right six positions.

A = Exponent +50, Fraction +0012345676543210000000 ⟶ 00
B = Exponent +50, Fraction +0025252525252525000000 ⟶ 00
Sum = Exponent +50, Fraction +0037620424015735250000 ⟶ 00

This is the intermediate sum, and its fraction is 96 bits in length. To obtain this 96-bit fraction, the fraction of A is extended with 48 low-order zeros. The fraction of B contains that portion of the original fraction that was preshifted, plus enough low-order zeros to form a 96-bit fraction. In this example, 42 low-order zeros were added to the fraction of B.

The above example preshifted the fraction, associated with the smaller exponent, six positions right. The original operand fractions occupied only 48 bits. Therefore, the preshifting can be up to 48 positions right, and still retain all the original data. However, when the difference in exponents exceeds 48, the low-order bits of the preshifted fraction are shifted beyond the right end of the intermediate sum.

When the difference in exponents exceeds 95, preshifting destroys the entire fraction that is associated with the smaller exponent. For this reason, when the difference exceeds 95, no addition is performed and the intermediate result is the original operand that has the larger exponent.

When unnormalized operation is specified, the 48 high-order bits of the 96-bit intermediate sum replace accumulator bits 12-59. The overflow bit does not enter the accumulator, but turns on indicator lost carry, LC.

When normalized operation is specified, the entire 96-bit intermediate sum fraction and overflow bit are shifted to form a normalized fraction and the result exponent is adjusted accordingly. This normalization is not performed if the overflow bit and high-order 48 bits of the intermediate sum are zero. The 48 high-order bits of the normalized sum fraction replace accumulator bits 12-59.

When the noisy mode and normalized operation are specified, the 48-bit fraction of the number with the higher exponent is extended with 48 low-order one bits. If the operands have equal exponents, the accumulator operand is extended. The extension takes place before the intermediate sum is formed. Post-normalization occurs as described above.

Because numbers having flagged exponents are not considered normal, they are handled in other than normal fashion by PAU. If either or both operands are values in the XFP or XFN range, initially having an exponent flag of 1, the operation is performed as follows. Modification of the fraction sign of the addressed operand is made as specified by the sign modifiers. If the exponents of both operands are in the XFP range or both in the XFN range, the result is the operand having the algebraically larger exponent. When both operands have the same exponent the operand initially in the accumulator is the result. Normalization, if specified, is suppressed. If one operand has an exponent in the XFP range and the other has an exponent in the N or XFN ranges, the result is the operand in the XFP range. Normalization, if specified, is suppressed. If one operand is in the N range and the other is in the XFN range, the intermediate result is the operand in the N range. In this case, however, normalization, if specified, is performed and may lead to a generated EF of 1 through underflow.

FLOATING POINT ADDITION

Augend (Accumulator Operand)	Addend (Storage Operand) XFP	N	XFN
XFP	XFP (1)	XFP (1)	XFP (1)
N	XFP (1)	N (3)	N (2)
XFN	XFP (1)	N (2)	XFN (1)

Notes:

1. a. The sum exponent will be the larger of the two original exponents.
   b. The sum fraction will be the original fraction that has the larger exponent.
   c. No addition or normalization takes place.
   d. Indicator XPFP

2. a and b. The same as a and b of Note 1.
   c. No addition takes place but normalization can occur if specified.
   d. Indicator XPFP is not turned on.

3. Through normalization, the exponent could become flagged.

Indicators

When considering the indicators that are set as the result of an ADD operation it is helpful to remember that a number in the XFP range is extremely large (approaching infinity) and a number in the XFN range is extremely small (approaching zero).

Data Flags (TF, UF, VF): These indicators are set according to the flag bits of the addressed operand. They are temporary.

To-Memory Operation (MOP): This indicator is set to zero. It is temporary.

Result Less than Zero (RLZ); Result Zero (RZ); Result Greater than Zero (RGZ); Result Negative (RN): These indicators are set according to the final arithmetic result fraction which appears in the accumulator and any overflow produced by the operation. These indicators are temporary.

Lost Carry (LC): This indicator is turned on if a fraction overflow occurred in an unnormalized addition. The indicator is permanent.

Lost Significance (LS): This indicator is turned on, in addition, when at least one operand fraction is non-zero, if the high-order 48 bits of the intermediate sum and overflow bit are zero, and the intermediate result exponent does not have a propagated flag. In this case, the final sum fraction is zero, and a normal order of magnitude zero is produced with no further exponent change or normalization. This indicator will not be turned on in addition if both operands had zero fractions at the start of the operation or if the intermediate result exponent flag has a propagated EF of 1. This indicator is permanent.

Preparatory Shift Greater than 48 (PSH): When both operands are values in the N range, or if one operand is a value in the N range while the other operand is a value in the XFN range, and the algebraic difference of the exponents is greater than 48, this indicator is turned on. If both exponents are flagged or if either exponent is flagged positive, the result has a propagated flag, and indicator PSH is not turned on. This indicator is permanent.

Exponent Range (XPFP, XPO, XPH, XPL, XPU): These indicators are set according to the exponent range of the final result appearing in the accumulator. When the result has a propagated flag, the XPFP is set. If the result has a positive exponent, with a generated flag, XPO is turned on. If the result has a positive unflagged exponent, with a value of 2^9 or greater (512 to 1023) XPH is turned on. If the result has a positive exponent, with a value between 2^6 and 2^9 (64 to 511), indicator XPL is set. Indicator XPU is turned on if the result has a negative exponent, with a generated flag.

Zero Multiply (ZM): This indicator is turned off, if it is on as a result of a previous operation.

Programming Note

For most arithmetic operations, if one of the operands has EF of 1, the result has EF of 1, and the flag is said to be propagated. In addition operations, however, a value in the N range, added to a value in the XFN range, may produce an intermediate sum with exponent in the N range. If this result exponent is altered during normalization, the exponent flag may be set to 1 in the final result. When this occurs, the exponent flag so produced is said to be a generated exponent flag.

6.20.2 Subtract (-)

As mentioned previously, subtraction is accomplished by setting the negative sign modifier, position 20, of the instruction, to one. The negative sign modifier is automatically inserted, by the assembly program, when the symbol + is replaced by the symbol -. Thus, a subtract operation is coded: -(N), JOE. Similarly, the absolute sign modifier is set to one, during assembly, if the symbol A is included in the STRAP notation. Thus, if the storage operand is to be assumed positive, before adding to the accumulator, it is coded, +A(N), JOE.

6.20.3 Add to Magnitude (+MG)

This operation is performed in the same manner as ADD, with the following exceptions:

The fraction sign, of the operand in the accumulator, is assumed positive prior to the addition. When the fraction sign of the intermediate sum is positive, the sum is placed in the accumulator, bits 0-59, and the accumulator sign remains unchanged. When the fraction sign of the intermediate sum is negative, a zero fraction is placed in the accumulator, bits 12-59. The sign and exponent of the original accumulator operand become the sign and exponent of the forced order-of-magnitude zero result. For this forced zero, the indicator lost significance (LS), is not turned on. This distinguishes the forced zero from a zero sum. In an ADD TO MAGNITUDE, the sign of the accumulator, bit position 4, of the sign byte register, is not altered.

If either or both operands contain flagged exponents, the operation is performed as described in ADD. That is, the number with the larger exponent becomes the intermediate result. If the fraction sign of this intermediate result is positive, the intermediate sum, excluding the fraction sign, replaces the accumulator contents. If the fraction sign of the intermediate sum is negative, the accumulator fraction is set to zero and the accumulator exponent is unaltered.

STRAP Notation: +MG(N), JOE
Operation Code: 01000

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD.

Lost Significance (LS): This indicator is set as in ADD with the exception that if the intermediate sum is negative, giving a final result of a forced zero fraction it is not turned on.

Arithmetic Result Indicators (RLZ, RZ, RGZ, RN): Indicator RN reflects the status of the unaltered accumulator sign. Indicators RLZ, RZ, and RGZ are set in accordance with the result and overflow bit.

Programming Note

ADD TO MAGNITUDE allows arithmetic to be performed on the magnitude of the accumulator contents. Since the arithmetic action of the accumulator operand is restricted to positive magnitude, sign change is not allowed; instead, a forced zero replaces the result.

6.20.4 Load (L)

The operand specified by the effective address is placed in the left half of the accumulator. The exponent and fraction of the operand replace accumulator bits 0-59.

Accumulator bits 60-127 are not changed. The sign of the operand, as changed by the sign modifiers, replaces the accumulator sign, bit position 4 in the accumulator sign byte register. The accumulator data flag bits are set to zero. When normalized operation is specified, the result in the left half of the accumulator is normalized and zero bits enter the low-order positions. If the address operand has a zero fraction, normalization is suppressed, since the result is an order-of-magnitude zero. If noisy mode is also specified, ones enter the low-order fraction bits during normalization.

When the specified operand is a value in the XFP or XFN range, normalization, if specified, is suppressed. The propagated EF of 1 causes exponent range indicator XPFP to be set on. When the EF of 1 is generated by normalization the exponent range indicator XPU is turned on.

STRAP Notation: L(N), JOE
Operation Code: 00 001

Indicators

Data Flags (TS, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD.

6.20.5 Load with Flags (LWF)

This operation is the same as LOAD except that the data flag bits, T, U, and V, of the addressed operand sign bytes set the flag bits in positions 5-7 of the accumulator sign byte register. They also set the data flag indicators TF, UF, and VF.

STRAP Notation: LWF(N), JOE
Operation Code: 01001

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD.

Programming Note

LOAD, LOAD WITH FLAG, LOAD DOUBLE and LOAD DOUBLE WITH FLAG are the only floating point operations that affect the data flag positions in the accumulator sign byte register. No floating point operation changes the zone bits, bit positions 0-3, of the accumulator sign byte register, unless address 10 is used as the addressed operand in a to-storage operation.

6.20.6 Add to Memory (M+)

The operand in the left half of the accumulator is added to the operand specified by the effective address. The exponent and fraction of the sum replace bits 0-59 of the storage word. When normalized operation is specified, the sum is normalized before it is placed in storage. The sign of the addressed operand is used as modified by the absolute sign modifier only. The negative sign modifier is used to modify the sign of

the accumulator. The sign of the sum is placed in bit position 60 of the storage word. The flag bits, 61-63, of the storage word remain unchanged. The contents of the entire accumulator and its sign byte register remain unchanged throughout the operation, unless these registers are used as the addressed operand.

The addition of the two operands is identical to the addition described in ADD. The intermediate result is a 96-bit fraction and a possible overflow bit. When normalized operation is specified, the sum is shifted and the 48 high-order bits of the sum, including the overflow bit, are placed in storage bits 12-59. The exponent of the sum is adjusted by the amount of the shift.

When unnormalized operation is specified, the 48 high-order bits of the sum are placed in storage bits 12-59. The overflow bit does not enter the storage word, but turns on the indicator lost carry (LC).

When noisy mode and normalized operation are specified, the operand fraction associated with the larger exponents, or the accumulator fraction if the exponents are equal, is extended with 48 low-order ones before the sum is formed. The result fraction is truncated to 48 bits before it is stored.

If either or both of the operands contain flagged exponents, the sum is obtained as in ADD with flagged exponents.

STRAP Notation: M+(N), JOE
Operation Code: 00010

Indicators

Data Flags (TF, UF, VF): These indicators are set according to the flag bits of the addressed operand. They are temporary.

To-Memory Operation (MOP): This indicator is set to one. It is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the final arithmetic result fraction which appears in storage and any overflow produced by the operation. These indicators are temporary.

Lost Carry (LC): This indicator is turned on if a fraction overflow occurred in unnormalized addition. This indicator is permanent.

Lost Significance (LS): This indicator is set on when at least one operand fraction is non-zero but the high-order 48 bits of the intermediate sum and overflow bit are zero and the intermediate result exponent flag is zero. The final sum fraction is zero, and a normal order-of-magnitude zero is formed with no further normalization or exponent change. This indicator is not turned on when both original operand fractions are zero or when the intermediate result exponent flag has a propagated EF of 1. This indicator is permanent.

Preparatory Shift Greater than 48 (PSH): When both operands are values in the N range, or if one operand is in the N range, while the other operand is a value in the XFN range and the algebraic difference of exponents is greater than 48, this indicator is turned on. When both operands are values in the XFP or XFN ranges or when one

operand is a value in the XFP range while the other is a value in the N or XFN range, the result has a propagated EF of 1 and this indicator is not turned on. This indicator is permanent.

Exponent Range (XPFP, XPO, XPH, XPL, XPU): These indicators are set according to the exponent range of the final result appearing in storage. When this result has a propagated EF of 1, XPFP will be set. When the result has a generated EF of 1, XPO or XPU will be set.

Zero Multiply (ZM): If on, this indicator will be turned off.

6.20.7 Add Magnitude to Memory (M+MG)

This operation is similar to ADD TO MEMORY except that the sign of the accumulator operand is assumed positive before the negative sign modifier is applied. Also, if the sign of the sum differs from the effective sign of the addressed operand, zeros are placed in the fraction field of the addressed operand.

The sign of the sum is compared with the effective sign of the addressed operand. When the signs are the same, the sum replaces the addressed operand in storage. When the signs are opposite, the result fraction is made a forced zero in storage. The sign of the forced zero is the same as the original sign of the addressed operand. For a forced zero, the indicator lost significance (LS) is not turned on, thus distinguishing it from a zero sum. In an ADD MAGNITUDE TO MEMORY, the original sign of the addressed operand always remains unchanged.

STRAP Notation: M + MG(N), JOE
Operation Code: 01010

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD TO MEMORY.

Lost Significance (LS): This indicator is set as in ADD TO MEMORY except that a forced zero does not turn it on.

As a review of the various floating point add operations, several examples are given below. The letters A and N denote the sign modifiers. Remember that the negative sign modifer applies to the unchanged operand while the absolute modifier applies to the storage operand.

Storage Operand	3+	5+	3+	5+	3-	5-	3-	5-
Accumulator Operand	5+	3+	5-	3-	5+	3+	5-	3-
Operation and Modifiers				Results				
Add	8+	8+	2-	2+	2+	2-	8-	8-
Add, N	2+	2-	8-	8-	8+	8+	2-	2+
Add, A	8+	8+	2-	2+	8+	8+	2-	2+
Add, N, A	2+	2-	8-	8-	2+	2-	8-	8-

Operation and Modifiers	Results							
Add to Magnitude	8+	8+	8-	8-	2+	0+	2-	0-
Add to Magnitude, N	2+	0+	2-	0-	8+	8+	8-	8-
Add to Magnitude, A	8+	8+	8-	8-	8+	8+	8-	8-
Add to Magnitude, N, A	2+	0+	2-	0-	2+	0+	2-	0-
Add to Memory	8+	8+	2-	2+	2+	2-	8-	8-
Add to Memory, N	2-	2+	8+	8+	8-	8-	2+	2-
Add to Memory, A	8+	8+	2-	2+	8+	8+	2-	2+
Add to Memory, N,A	2-	2+	8+	8+	2-	2+	8+	8+
Add Magnitude to Memory	8+	8+	8+	8+	0-	2-	0-	2-
Add Magnitude to Memory, N	0+	2+	0+	2+	8-	8-	8-	8-
Add Magnitude to Memory, A	8+	8+	8+	8+	8-	8-	8-	8-
Add Magnitude to Memory, N,A	0+	2+	0+	2+	0-	2-	0-	2-

The STRAP notations used for the various floating point add operations are listed below.

Operation	STRAP Notation
Add	+(N or U)
Add, N	-(N or U)
Add, A	+A(N or U)
Add, N, A	-A(N or U)
Add to Magnitude	+MG (N or U)
Add to Magnitude, N	-MG (N or U)
Add to Magnitude, A	+MGA (N or U)
Add to Magnitude, N, A	-MGA (N or U)
Add to Memory	M + (N or U)
Add to Memory, N	M - (N or U)
Add to Memory, A	M + A(N or U)
Add to Memory, N,A	M - A(N or U)
Add Magnitude to Memory	M + MG (N or U)
Add Magnitude to Memory, N	M - MG (N or U)
Add Magnitude to Memory, A	M + MGA (N or U)
Add Magnitude to Memory, N, A	M - MGA (N or U)

6.20.8 Store (ST)

The operand in the left half of the accumulator is placed in the location specified by the effective address. The exponent and fraction of the operand are taken from accumulator bits 0-59 and replace bits 0-59 of the storage word. The accumulator sign, sign byte register bit position 4, as modified by the sign modifiers, and the accumulator flag bits, sign byte register bits 5-7, replace bits 60-63 of the storage word. The entire accumulator and its sign byte register remain unchanged throughout the operation.

STRAP Notation: ST(N), NEM
Operation Code: 00011

When normalized operation is specified, the 48-bit operand fraction in the accumulator is normalized before being placed in storage. If the noisy mode is also specified, ones enter the low-order fraction bit positions as they are shifted left on normalization. Normalization is suppressed when the 48 fraction bits are zero.

Note that the absolute sign modifier applies to the operand going to or coming from storage and the negative sign modifier applies to the unchanged operand. Therefore, both modifiers apply to the operand from the accumulator.

When the exponent of the accumulator operand is flagged, normalization, if specified, is suppressed.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD TO MEMORY.

Programming Examples

Example A: In storage are 20 locations containing floating point data words. These words may have flagged or unflagged exponents. The fractions are not necessarily normalized and their signs are unknown. The customer engineer wishes to add these floating point numbers. Not to be included in the addition are any numbers with propagated flagged exponents or that produce a generated flag when added to the partial sum. The block of words begins at location START. The following program performs these functions.

Name	Instruction	Remarks
	LX, $X2, INDEX	$X2=0.0, 20, INDEX. Store zeros in location
	Z, HOLD	HOLD.
NEW	L(N), START ($X2)	Load first number into accumulator.
	BXPFPX, STEP	Branch if exponent had a propagated flag.
STORE	St(N),HOLD + (N),	Store partial sum. Add next number to
	START +1.0($X2)	partial sum.
	BXPFPZ,RESTORE	Branch if exponent had a propagated flag.
	BXPOZ,	Branch if this addition caused an XFP
	RESTORE	condition.
	BXPUZ,	Branch if this addition caused an XFN
	RESTORE	condition.
NEXT	CB+, $X2, STORE	Set up for next addition.
STEP	B, (Next Routine)	
	CB+, $X2, NEW	First number had a propagated flag.
		Load next number into accumulator.
	B, (Next Routine)	
RESTORE	L(N), HOLD	Last partial sum had an exponent flag.
		Reload accumulator with previous partial sum.
	B, NEXT	
INDEX	XW, 0.0, 20,INDEX	

Example B: At location INFO is the first of 100 normalized, positive, floating point numbers. These numbers have been previously processed. During processing, flag

bits were attached to the data words to identify them as belonging to one of three categories. Now a total is to be obtained for each category. Because each data word is flagged, the words may be added to one of three storage locations as called for by the flag bits. Assume the first category is identified by a V flag, the second category is identified by a U flag, and the third category by a T flag. Words that are T flagged are to be added to storage location 4096, U flagged words are to be added to location 2048, and V flagged words are to be added to location 1024. The following program steps provide the desired action.

Name	Instruction	Remarks
PREP	LX, $X7,AID	Place 100 in count field of XR 7.
LOAD	L(N), INFO ($X7)	Load operand into accumulator.
PLACE	LV, $X8, 10.0	Place the flag in position 5,6,or 7 of XR 8 value field. This initially loads 1024, 2048, or 4096 into the value field.
ADD	M+(N), 0.0 ($X8)	Add accumulator to one of the three storage words.
RETURN	CB+, $X7, LOAD	Add one to value field. Step down count. If less than 100 words have been added, BRANCH TO LOAD.
AID	XW, 0.0, 100	VALUE = 0 COUNT = 100

6.20.9 Compare (K)

The operand in the left half of the accumulator is compared with the operand specified by the effective address. In this operation, both sign modifiers are applied to the storage operand.

The two numbers are compared algebraically by subtraction, but neither operand is altered. In the intermediate stage the subtraction operation is identical to ADD except for the inversion of the storage operand fraction sign. The result is discarded and the comparison result indicators rather than the arithmetic result indicators are set to indicate the comparison.

When the exponent difference between the operands exceeds 48, the algebraically larger of the two numbers, judged by exponent and fraction sign, is considered greater. This statement is true even when one or both of the numbers are zero. When the exponent difference is 48 or less, however, +0 is considered equal to -0.

In an ADD, the normalization modifier specifies post-normalization. Since there is no stored arithmetic result in a comparison operation, the normalization modifier affects the operation only during noisy mode. When noisy mode and normalized operation are specified, the operand having the higher exponent, or the accumulator operand if both operands have equal exponents, is extended with 48 low-order ones prior to forming the intermediate difference, as in ADD.

STRAP Notation: K(U), WORD
Operation Code: 00100

When one or more of the operands are values in the XFP or XFN ranges, comparisons are performed as follows. When the fractional signs are unlike, the value having the positive fraction sign is considered greater. When the fraction signs are both

positive, a value in the XFP range is considered greater than a value in the N or XFN range. Also, a value in the N range is considered greater than a value in the XFN range. If both operands have fraction signs which are negative, a value in the XFN range is considered greater than a value in the N or XFP range; and a value in the N range is considered greater than a value in the XFP range. If both operands are in the XFP range or both in the XFN range and the fraction signs are alike, the comparison is considered equal.

The arithmetic result and exponent range indicators are not altered.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Preparatory Shift Greater than 48 (PSH); These indicators are set as in ADD.

Accumulator Low (AL); Accumulator Equal (AE); Accumulator High (AH); These comparison result indicators reflect the results of the algebraic comparison.

6.20.10 Compare Magnitude (KMG)

The comparison is performed as described for COMPARE except that the comparison is performed as if the sign of the accumulator operand were positive.

STRAP Notation: KMG (U), WORD
Operation Code: 01100

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH); Preparatory Shift Greater than 48 (PSH); The indicators above are set as in COMPARE.

6.20.11 Compare for Range (KR)

When the indicator accumulator high (AH), is on, a comparison is made between the operand in the left half of the accumulator and the operand specified by the effective address. If the indicator accumulator high (AH), is off, no comparison will be performed, and the comparison result indicators are not changed.

STRAP Notation: KR(U), WORD
Operation Code: 00101

The comparison operation, if performed, is the same as COMPARE except for the setting of the comparison result indicators.

Indicators

Data Flags (TF, UF, VF): These indicators are set according to the data flag bits of the addressed operand, whether or not the comparison is performed.

To-Memory Operation (MOP): This indicator is set to zero whether or not the comparison is performed.

Accumulator Low (AL): This indicator is not changed.

Accumulator Equal (AE): This indicator is set to one if the accumulator operand is low when compared with the addressed operand.

Accumulator High (AH): This indicator remains one if the accumulator operand is equal or high when compared with the addressed operand; it is set to zero when the accumulator operand is low compared with the addressed operand.

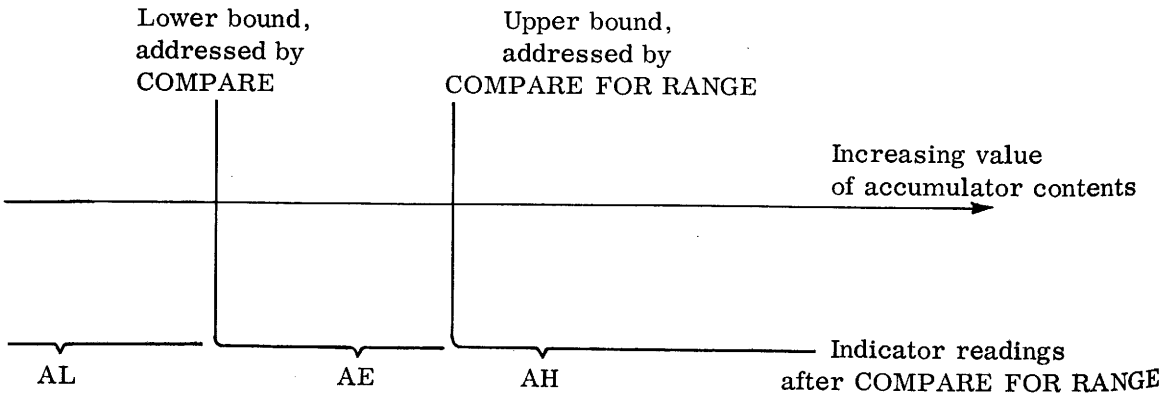
Programming Note

Following a COMPARE, the COMPARE FOR RANGE can be used to determine whether or not a quality falls within a given range. The addressed operand of the COMPARE may be considered as the lower bound of the range. The addressed operand of the COMPARE FOR RANGE may be considered as one greater than the upper bound of the range.

When both operations have been performed, the settings of the comparison result indicators are interpreted as follows: Accumulator low (AL) is one when the operand in the accumulator is below the range. Accumulator equal (AE) is one when the operand in the accumulator is within the range or equal to the lower boundary. Accumulator high (AH) is one when the operand in the accumulator is above the range or equal to the upper boundary.

The lower boundary is included in the range while the upper boundary is presumed to be outside the range.

The indicator settings resulting from a COMPARE followed by a COMPARE FOR RANGE are shown below.



6.20.12 Compare Magnitude for Range (KMGR)

This operation is the same as COMPARE FOR RANGE except that the comparison is performed as if the sign of the accumulator were positive.

STRAP Notation: KMGR(U), TOP
Operation Code: 01101

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH); Preparatory Shift Greater than 48 (PSH): These indicators are set as in COMPARE FOR RANGE.

The following figures summarize the settings of the comparison result indicators for comparison operations. Figure 6.20-1 describes the indicator settings when both operands have N range exponents. Figure 6.20-2 describes the indicator settings, if either or both operands have flagged exponents. The operand exponent is represented by XFP, N or XFN and the fraction sign is shown next, by a plus or minus. The indicators in small letters are those set by COMPARE FOR RANGE and COMPARE MAGNITUDE FOR RANGE.

Programming Example

In storage, starting at location VOLUME, are 300 floating point numbers in no particular order. Each number is to be placed in one of three groups depending on the size of the number. The numbers with values less than 5,000 are to be stored in as many locations as required starting at location FIRST. Numbers with values of 5,000 or greater but less than 10,000 are to be stored in a block starting at location SECOND. All values of 10,000 or greater are to be stored in a block starting at location THIRD. Each block contains at least 300 locations. Assume all the floating point numbers have exponents in the normal range. The program, using COMPARE and COMPARE FOR RANGE is:

Name	Instruction	Remarks
PREP	LX, $X10, NEED	Prepare XR 10 for fetching operands.
	LX, $X1, NEED	Prepare XR1 for storing operands.
	LX, $X2, NEED	Prepare XR2 for storing
	LX, $X3, NEED	Prepare XR3 for storing
GO	L(U), VOLUME ($X10)	Fetch operand
	K(U), AMOUNT 1	Determine if operand is greater than 5000; if so.....
	KR(U), AMOUNT 2	....determine is operand is equal to or greater than the range 5000-10000.
	BAL, PUT 1	Operand belongs in block FIRST.
	BAE, PUT 2	Operand belongs in block SECOND.
PUT 3	ST(U), THIRD ($X3)	Operand is 10,000 or greater.
	V+I, $X3, 1.0	Update XR3 for future storing.
UPDATE	CB+, $X10, GO	Update value of XR10 for fetching next operand. If less than 300 operands have been processed, return to GO.
	B, NEXT	Branch to next routine.
PUT 1	ST(U), FIRST ($X1)	Operand is less than 5000.
	CB+, $X1, UPDATE	Update XR1 for future storing. Branch for updating XR10.
PUT 2	ST(U), SECOND, ($X2)	Operand is at least 5,000 but less than 10,000.
	CB+, $X2, UPDATE	Update XR2 for future storing. Branch for updating XR10.
NEED	XW, 0.0, 300	VALUE = 0.0 COUNT = 300

Fm = Storage Operand Fraction
Fa = Accumulator Operand Fraction
Em = Exponent of Storage Operand
Ea = Exponent of Accumulator Operand

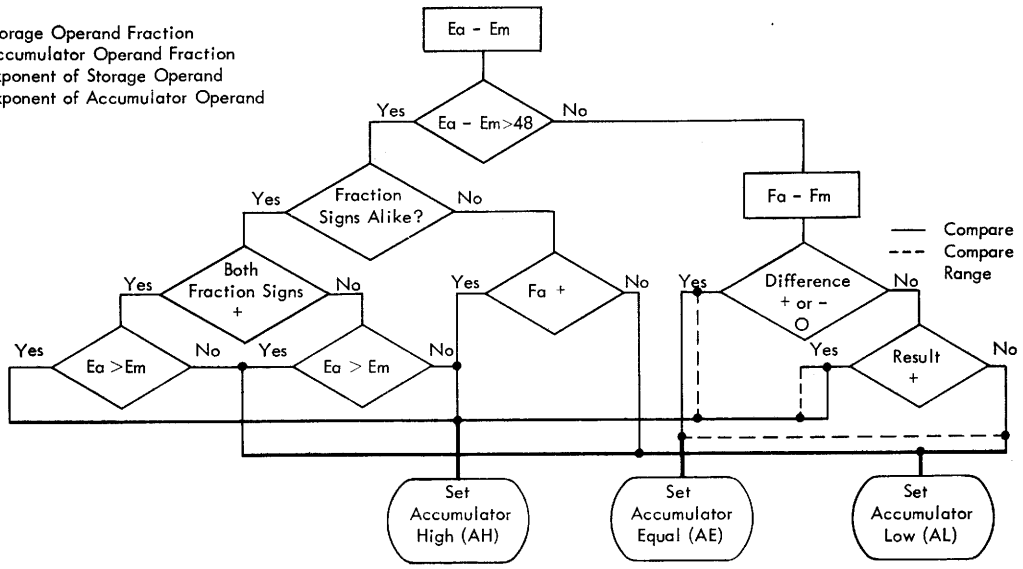


FIGURE 6.20-1. INDICATOR SETTINGS, OPERANDS IN NORMAL RANGE

ACCUMULATOR	STORAGE					
	XFP, +	N, +	XFN, +	XFN, -	N, -	XFP -
XFP, +	AE [ah]	AH	AH	AH	AH	AH
N, +	AL [ae]	*	AH	AH	AH	AH
XFN, +	AL [ae]	AL [ae]	AE [ah]	AH	AH	AH
XFN, -	AL [ae]	AL [ae]	AL [ae]	AE [ah]	AH	AH
N, -	AL [ae]	AL [ae]	AL [ae]	AL [ae]	*	AH
XFP, -	AL [ae]	AL [ae]	AL [ae]	AL [ae]	AL [ae]	AE [ah]

FIGURE 6.20-2. INDICATOR SETTINGS, FLAGGED OPERANDS

AMOUNT 1 Floating Point Equivalent of 5,000
AMOUNT 2 Floating Point Equivalent of 10,000
NEXT First Step of Next Routine.

6.20.13 Multiply (*)

The operand specified by the effective address, the multiplicand, is multiplied by the operand in the left half of the accumulator, the multiplier. The product exponent and fraction replace accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The sign of the product replaces the accumulator sign in bit position 4 of the sign byte register.

Multiplication of floating point numbers consists of an exponent addition and a fraction multiplication. The sum of the exponents is used as the exponent of the unnormalized product. The two 48-bit operand fractions are multiplied to form a 96-bit intermediate product. The product sign is determined by the rules of algebra, using the accumulator sign and the effective sign of the addressed operand. The sign of the addressed operand is modified according to the specifications of the sign modifiers.

If a normalized product is specified, the 96-bit intermediate product fraction is normalized. Following the normalization the intermediate product fraction is truncated to 48 bits to form the result product fraction that is placed in accumulator bits 12-59. Zeros are inserted in the low-order positions of the 96-bit intermediate product fraction as normalization occurs. If noisy mode is in effect, low-order ones instead of zeros will be inserted during normalization. The zeros or ones do not appear in the result product fraction unless the amount of post-normalization shift is greater than 48.

When unnormalized operation is specified, the 48 high-order bits of the intermediate product fraction are used as the result product fraction; this fraction is placed in accumulator bits 12-59.

For values of the operands in the XFP or XFN range, the following action is taken. When both operands are in the XFP range or both in the XFN range, the result exponent is the exponent of the accumulator operand. The result fraction is the unnormalized product of the operand fractions. If one operand is in the XFP range and the other is in the N or XFN range, the result exponent is the exponent of the operand in the XFP range and the result fraction is the unnormalized product of the operand fractions. If one of the operands is in the N range while the second is in the XFN range, the result exponent is the exponent of the operand in the XFN range and the result fraction is the unnormalized product of the operand fractions. For all these cases normalization, if specified, is suppressed. Thus, whenever either operand in a multiplication has an EF of 1, the result exponent always has a propagated EF of 1.

When both operands are values in the N range, a generated EF of 1 may be produced through overflow or underflow. When underflow exists and the result exponent is in the XFN range, normalization of the result fraction, if specified, is suppressed in order to prevent occurrence of a double carry. However, the indicator settings follow the rules for generated exponent flags.

STRAP Notation: *(N), RATE
Operation Code: 00110

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range(XPFP, XPO, XPH, XPL, XPU): These indicators are set for the final arithmetic result as in ADD.

Zero Multiply (ZM): If normalized operation is specified and an order of magnitude zero results with an exponent greater than -1024, this indicator is turned on. Otherwise it will be turned off, if already on.

Summary

The following multiplication table summarizes the operation.

FLOATING POINT MULTIPLICATION

Multiplier (Accumulator Operand)	Multiplicand (Storage Operand)		
	XFP	N	XFN
XFP	XFP (1)	XFP (1)	XFP (1)
N	XFP (2)	N (3)	XFN (2)
XFN	XFP (2)	XFN (1)	XFN (1)

NOTES: The exponent flags are arbitrarily checked in the following order: + Accumulator, + Storage, - Accumulator, - Storage. Therefore:

1. The product exponent will be that of the original multiplier. The product fraction will be the result of the normal multiplication. Normalization is suppressed.
2. The product exponent will be that of the original multiplicand. The product fraction will be the result of normal multiplication. Normalization is suppressed.
3. The product exponent and fraction will be a result of normal multiplication. However, the exponent could have a generated flag through the addition of the exponents, or through normalization.

If a generated flag places the exponent in the XFP range, normalization, if specified, takes place. If a generated flag places the exponent in the XFN range, normalization is suppressed. The reason for the suppression of normalization, in this last case, is to prevent a double underflow carry. The following example illustrates a double underflow carry. Consider the exponents only.

Ea + Em = E product; Ea and Em are "N" range exponents (all numbers are represented as octal quantities).

	Ea = -1776	
	Em = -1775	
Intermediate	Exp = -3773	This is an XFN range exponent.

Assume normalization is specified and the fraction was shifted six positions.

Algebraically subtracting the number of shifts, we obtain:

Intermediate Exp. = -3773
- 6
-0002 (Using end carry)

The exponent now appears to be in the N range, and is incorrect. This is a double underflow carry. The reason for suppressing normalization, if the intermediate exponent is in the XFN range, is to prevent this condition from existing.

6.20.14 Divide (/)

The operand in the left half of the accumulator, the dividend, is divided by the addressed operand, the divisor. The sign of the addressed operand is modified by the sign modifiers prior to the division. The resulting quotient exponent and fraction replace accumulator bits 0-59. Accumulator bits 60-127 remain unchanged. The quotient sign replaces the accumulator sign in bit position 4 of the sign byte register. No remainder is retained by this operation.

STRAP Notation: /(N), MEM
Operation Code: 00111

The actual division is accomplished by subtracting the divisor exponent from the dividend exponent and dividing the dividend fraction by the divisor fraction. The division of fractions takes place in all cases except when the divisor fraction is zero, when the zero divisor (ZD) indicator is set and the accumulator contents are unchanged.

The unusual division method used in PAU is presented as an aid to understanding the function of the divide operation.

The operation is partly determined by whether or not normalization is specified. When normalization is specified, both the divisor and the dividend are prenormalized. This will result in a quotient that is a normalized quantity. In prenormalizing the divisor, the number of positions the divisor fraction is shifted, is recorded in the left zeros counter (LZC). The dividend is then shifted and the number of shifts necessary to normalize the dividend is subtracted from the quantity in the LZC. If the dividend is shifted more positions than the divisor, the LZC is set to zero. A use for the contents of the LZC after a division operation is presented later in the manual.

If normalization is not specified, the divisor is still normalized. As previously described, the number of shifts required to normalize the divisor fraction is placed in the LZC. An attempt is then made to normalize the dividend. The dividend fraction is shifted until it is normalized, or until it is shifted the same number of positions as the divisor, whichever is less. In either case the number of positions shifted, by the dividend, is subtracted from the quantity in the LZC. At this time, the contents of the LZC indicate how many more positions the divisor was shifted than the dividend. It should be noted that the divisor is normalized, regardless of the setting of the normalization modifier, in the instruction. Whenever an operand fraction is shifted, the corresponding exponent is adjusted accordingly.

After the prenormalization is performed, a trial subtraction of the fractions takes place. If the dividend fraction is larger than the divisor fraction, the quotient is

expected to be greater than one. Therefore, a one is inserted in the high-order fraction position and the dividend exponent is increased by one. A one is also added to the contents of the LZC.

Both exponents have now been adjusted. The exponent of the quotient can be determined. The intermediate quotient exponent is found by subtracting the adjusted divisor exponent from the adjusted dividend exponent. The exponent sign is determined by rules of algebra.

The actual division of the fractions is performed next. If normalization is specified, the quotient will be a normalized quantity, because the divisor and dividend have been prenormalized.

If unnormalized operation is specified, the quotient may not be a normalized quantity. The quantity in the left zeros counter may now be used by the programmer. If the contents of the LZC are not zero at this time, it indicates that the divisor was shifted more positions than the dividend during prenormalization. To inform the programmer of this condition, the partial field (PF) indicator is set to one. The programmer can use the contents of the LZC to form a fixed point quotient. To do this, the quantity in the LZC indicates the number of left shifts required to make the quotient exponent zero. Keep in mind, however, that this is not a machine function, but would require additional programming.

Consider the following example: An unnormalized division of two fixed-point numbers is to be performed by the floating point circuitry. The fixed point numbers are represented in floating point formats with zero exponents. (All numbers are represented as octal quantities.)

Divisor = Exp. +0, Fraction +.005
Dividend = Exp. +0, Fraction +.043
The divisor is normalized.
Divisor = Exp. -6, Fraction +.5
The number of shifts required to normalize the divisor is 6. This quantity is placed in the LZC.
LZC = 6
The dividend fraction is shifted three positions left. For each position shifted, the LZC is reduced by one and the dividend exponent is adjusted accordingly.
Dividend = Exp. -3, Fraction +.43
LZC = 3
The division is performed.
Quotient = Exp. +3, Fraction +.7

This quotient is not the desired, fixed point quantity. Performing the same operation by long-hand division, the quotient is:

$$.005.\overline{)\,.043.}\quad 7.$$

Note that the radix point of this quotient is displaced three binary bit positions from the radix point of the quotient fraction obtained from the machine. The LZC indicates this condition (LZC = 3). It is up to the programmer to rescale the operation, if he wants to continue to work with fixed point quantities.

The operands in the division operations considered up to this point have been in the N range. However, operands with both generated and propagated exponents flags must be considered.

In MULTIPLY, a generated exponent flag is not detected until the intermediate product is developed. That is, a generated exponent flag could not be produced until the operand exponents were added, or until the intermediate product was normalized. In DIVIDE, however, a generated exponent flag could be produced prior to obtaining an intermediate result, because the operands may be normalized before the actual division is performed. Normalization can cause an exponent, originally in the N range, to fall into the XFN range. One situation will cause an exponent in the N range to go into the XFP range: a division when the dividend is larger than, or equal to, the divisor. In this case, a one is automatically placed in the high-order position of the quotient and one is added to the dividend exponent. If the original dividend exponent is equal to +1023 (N range), the addition of one makes the exponent equal to +1024, or an XFP range exponent.

The following table describes the results of a division operation if either operand has a generated exponent flag.

Divisor (Storage Operand)	Dividend (Accumulator Operand)			
	XFN	+N	-N	XFP
XFN	(1)	(3)	(1)	(3)
+N	(2)	N	(1)	(1)
-N	(1)	(1)	N	(1)

NOTES:

1. The quotient exponent will be the result of the normal subtraction of exponents. The quotient fraction will be the result of normal division.
2. The quotient exponent will be that of the dividend (XFN). The quotient fraction will be the result of normal division.
3. The quotient exponent will be that of the divisor, with the sign inverted (XFP). The quotient fraction will be the result of normal division.

Divide operations involving exponents with propagated flags are different from those presented up to this point. When either operand has a propagated exponent flag, the subtraction of exponents is suppressed. In such cases, the operands may be considered either extremely large or extremely small and the quotient exponents are set to reflect the general range to be expected from such a division. The magnitude of the quotient exponent is arbitrarily replaced by the exponent magnitude of one of the operands. The actual settings of the quotient exponents, when propagated flags are encountered, is presented in the following table.

FLOATING POINT DIVISION for Operand Exponents Containing Propagated Flags

Divisor (Storage Operand)	Dividend (Accumulator Operand) XFD	N	XFN
XFP	XPF (1)	XFN (2)	XFN (2)
N	XFP (1)	N	XFN (1)
XFN	XFP (2)	XFP (2)	XFP (2)

NOTES:

1. The quotient exponent will be the original dividend exponent. The quotient fraction will be the unnormalized result of regular division.
2. The quotient exponent will be the original divisor exponent, with the sign inverted. The quotient fraction will be the unnormalized result of regular division.

Programming Note

When the divisor is found to be zero, no division is performed. The contents of the accumulator and storage remain unchanged. When the dividend is found to be zero, division takes place but normalization is suppressed (quotient will be zero).

Indicators

Data Flags (TF, UF, VF): These indicators are set according to the flag bits of the addressed operand. They are temporary.

To-Memory Operation (MOP): This indicator is set to zero; it is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the quotient fraction as it appears in the accumulator. They are temporary.

Partial Field (PF): This indicator is turned on when unnormalized operation is specified, if the left zeros counter (LZC) is greater than zero.

Zero Divisor (ZD): This indicator is turned on when the divisor fraction is zero, and the division does not take place. This indicator is permanent.

Exponent Range (XPFP, XPO, XPH, XPL, XPU): These indicators are set according to the exponent range of the quotient. They are permanent.

Zero Multiply (ZM): If on, this indicator is turned off, except for the case of a zero divisor.

6.20.15 Reciprocal Divide (R/)

The operand specified by the effective address, the dividend, is divided by the operand in the left half of the accumulator, the divisor. The sign of the dividend is modified by the sign modifiers prior to the division. The quotient exponent and fraction replace accumulator bits 0-59; accumulator bits 60-127 remain unchanged. The sign of the quotient replaces the accumulator sign in bit position 4 of the sign byte register.

The division process is the same as the process described in DIVIDE with the exception of the interchange of the operands used for dividend and divisor.

STRAP Notation: R/(N), ABLE
Operation Code: 01111

Note that RECIPROCAL DIVIDE is appropriately named. Although the divisor and dividend are merely interchanged, the effect is the same as using the reciprocal of the original operands. This relationship is illustrated below.

	Hand Method	Machine Method
DIVIDE	$4 \div 2 = 2$	$2 \sqrt{4} = 2$
RECIPROCAL DIVIDE	$1/4 \div 1/2 = 2/4 = 1/2$	$4 \sqrt{2} = 1/2$

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Partial Field (PF); Zero Divisor (ZD); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in DIVIDE.

6.20.16 Store Root (SRT)

The square root of the operand in the left half of the accumulator is placed in the storage location specified by the effective address. The entire accumulator and its sign byte remain unchanged.

STRAP Notation: SRT (N), MASS
Operation Code: 11011

For values of the operand in the N range the square root operation consists of an exponent division and a fraction square root operation. The exponent is divided by two. When the exponent is odd, a one is added to the exponent before the division and the fraction is shifted right one position. If this addition produces an exponent in the XFP range the division of the exponent is still performed. Subsequently, the square root of the 48- or 49-bit fraction is obtained. During the process the fraction is extended with low-order zeros. The result is a 48-bit fraction.

When normalized operation is specified, the result is normalized and placed in storage. When unnormalized operation is specified, the result is placed unchanged in storage. In both cases the result is placed in bit positions 0-59 of the storage location specified by the effective address.

The absolute sign modifier is applied to the accumulator sign prior to the extraction of the square root. When this modifier is zero, the accumulator sign is used unchanged. If the accumulator sign is negative the imaginary root indicator (IR), is turned on. When the absolute sign modifier is one, the accumulator sign is assumed positive and IR cannot be turned on. In any case, a positive root is obtained.

The negative sign modifier is applied to the result of the square root process. When this modifier is zero, the sign of the stored result is positive. When this modifier is one, the sign of the stored result is negative.

As with other store operations, the accumulator flag bits, sign byte register bits 5-7, are placed in the flag bit positions, bits 61-63 of the addressed storage location.

When the operand is a value in the XFP or XFN range, the result exponent is the same as the operand exponent. The result fraction is the square root of the accumulator fraction. Normalization, if specified, is suppressed. Note that when the operand exponent is odd the fraction is shifted one position to the right before taking the square root and the exponent is increased by one. The exponent is not modified, however, as a result of the shift for operands in the XFP or XFN range.

Indicators

To-Memory Operation (MOP): This indicator is set to one; it is temporary.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the arithmetic result which appears in storage. These indicators are temporary.

Imaginary Root (IR): This (permanent) indicator is turned on when the accumulator sign, as modified by the absolute sign modifier, is negative.

Exponent Range (XPFP, XPH, XPL): These indicators (permanent) are set according to the exponent range of the result which appears in storage.

Zero Multiply (ZM): This indicator, if on, is turned off.

The negative and absolute sign modifiers are stated in STRAP format as follows:

Operation	Modifiers Applied
SRT	None
SNRT	Negative
SRTA	Absolute
SNRTA	Negative and Absolute

Programming Example

Assume a short routine is to check the operation of the instructions MULTIPLY, DIVIDE, and STORE ROOT. One method is to multiply a number by itself to form the square of the number, then divide the square of the number by the original number and compare the result with the result of a STORE ROOT instruction. Assume the original number, at location NUMBER, is positive and has an exponent well within the N range. The program is:

Name	Instruction	Remarks
PREP	LCI, $X1, 200.0	Routine to be executed 200 times.
START	L(N), NUMBER	Place number in accumulator.
	*(N), NUMBER	Square the number.
	SRT (N), TEMP	Store root of square in TEMP.
	/(N), NUMBER	Divide square by number.
CHECK	K(U), TEMP	Compare root with quotient; should be equal.
	BZAE, ERROR	If not equal, branch to error routine.
RETURN	CB, $X1, START	Repeat 199 times
NEXT	First Instruction of next routine	

6.20.17 Add Double (D+)

The operand specified by the effective address is added to the operand in the accumulator, bits 0-107. The addressed operand is extended with 48 zeros during the operation. The exponent and fraction of the sum replace the accumulator bits 0-107. Accumulator bits 108-127 remain unchanged. The sign of the addressed operand is modified by the sign modifiers prior to the addition. The sign of the sum replaces the accumulator sign.

STRAP Notation: D+ (N), SAM
Operation Code: 10000

The detailed operation of the addition is similar to that of ADD. When the difference between the exponent exceeds 48, the indicator preparatory shift greater than 48 (PSH), is turned on. When the difference between the exponents exceeds 95 the number with the lower exponent is not added, and the result is the number with the higher exponent. The fraction addition yields a 96-bit sum and possible fraction overflow bit.

When normalized operation is specified, the sum fraction is normalized and replaces accumulator bits 12-107. If noisy mode is also specified, one bits enter the low-order bit positions when the 96-bit intermediate result is shifted left during normalization. Noisy mode one bits are not added to either operand before the addition as is done in ADD.

When unnormalized operation is specified, the 96-bit intermediate sum fraction is placed in accumulator bits 12-107. The overflow bit does not enter the accumulator, but turns on the lost carry indicator.

If the 96-bit intermediate sum has a zero fraction, normalization, if specified, will be suppressed. The exponent replacing accumulator bits 0-11 will be the exponent of the intermediate result. The lost significance indicator (LS) is set to one.

When the operand values lie in the XFP or XFN range, the addition process is the same as for ADD except that the accumulator fraction is 96 bits and the operand specified by the effective address is extended with 48 zeros prior to the addition.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD.

Lost Significance (LS): This indicator is set to one when at least one operand is non-zero and the 96 bits of the result fraction are zero. This indicator is not set to one if the result exponent has a propagated flag or if both operands have 96-bit zero fractions initially.

6.20.18 Add Double to Magnitude (D+MG)

This operation is the same as ADD DOUBLE except that the sign of the accumulator operand is assumed to be positive and the result is made a forced zero when a negative

intermediate sum is obtained. When the sign of the intermediate sum is positive, the sum is placed in accumulator bits 0-107 and the sign remains unchanged. When the sign of the intermediate sum is negative, a zero fraction is placed into bits 12-107 of the accumulator and the sign and exponent of the accumulator operand remain equal to the original accumulator sign and exponent.

STRAP Notation: D+MG (N), BIG
Operation Code: 11000

Normalization, noisy mode, and values in the XFP or XFN range are handled as in ADD DOUBLE.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD TO MAGNITUDE.

Lost Significance (LS): This indicator is actuated as in ADD DOUBLE except that a forced zero does not turn it on.

6.20.19 Double Load (DL)

This operation is the same as LOAD except that the accumulator bits 60-107 are set to zero. The operand specified by the effective address is placed in the accumulator. The exponent and fraction replace bits 0-59. Accumulator bits 60-107 are set to zero. Accumulator bits 108-127 remain unchanged. The sign of the operand, modified by the sign modifiers, is placed in bit position 4 of the sign byte register. The data flag bits, bit positions 5-7 of the sign byte register, are set to zero.

STRAP Notation: DL(N), AMOUNT
Operation Code: 10001

When normalized operation is specified, the 96-bit fraction is shifted left and the exponent adjusted before being placed in the accumulator. If noisy mode is also specified, one bits are introduced into the low-order positions of the 96-bit fraction as it is shifted left during normalization.

Values in the XFP or XFN ranges are handled as in LOAD. Normalization of these operands, if specified, is suppressed.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM): The above indicators are set as in ADD.

6.20.20 Load Double with Flag (DLWF)

The operation is the same as LOAD DOUBLE except that the data flag bits, (T, U, and V) of the addressed operand set the accumulator flag bits in the sign byte register, bit positions 5-7, as well as the data flag indicators TF, UF, VF.

STRAP Notation: DLWF (N),
Operation Code: 11001

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD.

6.20.21 Store Rounded (SRD)

This instruction is normally used to store single-precision values which were generated as the result of double-precision operations. If the value in accumulator positions 60-107 is equal to or greater than one-half the value of a one in position 59, the single precision number fraction (12-59) is increased by a value of one before storing takes place. The contents of the accumulator, after rounding, are placed in the location specified by the effective address.

STRAP Notation: SRD (N), ANSWER
Operation Code: 01011

The rounding is performed by adding one to the fraction in the accumulator in bit position 60. This addition normally results in a 96-bit sum and a possible overflow bit. The 48 low-order bits are discarded. When normalized operation is specified, the 48-bit sum is then normalized and placed in bit positions 12-59. Low-order zeros are always supplied in the case of the left shift. Noisy mode does not affect this operation.

When unnormalized operation is specified, the 48 high-order bits of the sum are placed in storage bits 12-59. The overflow bit does not enter storage but sets the lost carry indicator (LC) on.

The sign of the accumulator, as changed by the sign modifiers, is placed in storage bit 60. The accumulator flag bits are placed in storage bits 61-63. The entire accumulator and its sign byte register remain unchanged throughout the operation.

If the value of the operand lies in the XFP or XFN range, the rounding process is performed but the exponent is not changed. Normalization, if specified, is suppressed. If, as a result of rounding, the fraction has an overflow bit set, the overflow bit is lost. However, the LC indicator is not set if overflow occurs when normalization is specified.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD TO MEMORY.

6.20.22 Store Low Order (SLO)

The low-order part of a double-length accumulator fraction with appropriate exponent is placed in the location specified by the effective address.

STRAP Notation: SLO (N), REST
Operation Code: 10011

The low-order bits of the accumulator fraction, bits 60-107, become the fraction of the value to be stored. The exponent is obtained by subtracting 48 from the exponent in bits 0-11 of the accumulator.

When normalized operation is specified, the low-order fraction is shifted left until a high-order one bit is leftmost. If noisy mode is also specified, ones are entered from the right during the normalization shifting. When unnormalized operation is specified, the low-order exponent and fraction remain unchanged. The exponent and fraction are subsequently placed in storage bits 0-59. The accumulator sign, as modified by the sign modifiers, and the accumulator flag bits, sign byte register bit positions 4-7, are placed in bits 60-63 of the storage word. The entire accumulator and the sign byte register remain unchanged throughout the operation.

When the exponent, accumulator bit positions 0-11, is flagged, the same exponent is set with the low-order fraction when stored. For this case normalization, if specified, is suppressed.

When the exponent with the low-order fraction has EF of 1 as a result of the operation, normalization, if specified, does occur.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in STORE.

6.20.23 Multiply Double (D*)

The operation is the same as MULTIPLY, except that the 96-bit product fraction is placed in accumulator bits 12-107.

The operand specified by the effective address, the multiplicand, is multiplied by the operand in the accumulator bits 0-59, the multiplier. The product exponent and fraction replace accumulator bits 0-107. The product sign replaces the accumulator sign, bit 4 of the sign byte register.

STRAP Notation: D*(N), AMOUNT
Operation Code: 10110

When normalized operation is specified, the 96-bit intermediate product fraction is normalized and the product exponent adjusted accordingly. Low-order fraction zeros are supplied. If the noisy mode is also specified, the final normalization introduces low-order ones instead of zeros. When unnormalized operation is specified, the intermediate product becomes the final product.

When either or both of the operands are values in the XFP or XFN range, the operation is performed as described in MULTIPLY except that a 96-bit fraction is obtained. Normalization, if specified, is suppressed for these cases. Also, if a flagged negative exponent is generated, normalization is suppressed as it may cause a double underflow carry condition.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Range (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in MULTIPLY.

6.20.24 Load Factor (LFT)

The operand specified by the effective address is placed in the factor register, FT. The accumulator does not participate in the operation and remains unchanged.

STRAP Notation: LFT(N), WORD
Operation Code: 10010

When normalized operation is specified, the exponent and fraction of the addressed operand are normalized and subsequently are placed in FT bits 0-59. If noisy mode is also specified, ones instead of zeros will enter the low-order bits of the fraction during normalization. If a shift of 48 or more is required, normalization is suppressed and the order-of-magnitude zero is stored in bits 0-59.

When unnormalized operation is specified, the exponent and fraction of the addressed operand are placed unaltered in FT bits 0-59. The sign of the addressed operand as modified by the sign modifiers is placed in FT bit 60. FT bits 61-63 are set to zero. When the operand is a value in the XFP or XFN ranges, normalization, if specified, is suppressed.

Indicators

Data Flags (TF, UF, VF): These indicators are set according to the flag bits of the addressed operand.

To-Memory Operation (MOP): This indicator is set to zero.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the arithmetic result which appears in the FT register.

Exponent Range (XPFP, XPH, XPL, XPU): These indicators are set according to the exponent range of the result which appears in the FT register.

Zero Multiply (ZM): If on, this indicator is turned off.

Programming Note

This operation is required for loading the multiplicand of a subsequent MULTIPLY AND ADD. It is not used to load either factor of a MULTIPLY operation.

The factor register, FT, is addressable and, therefore, can be addressed as the storage operand of any operation. FT is a 64-bit register with word address 14.

LOAD FACTOR is the only operation which changes the contents of FT as an integral part of its performance. FT is not changed by any other operation unless it is addressed explicitly.

6.20.25 Multiply and Add (* +)

The operand in the FT register, designated as the multiplicand, is multiplied by the operand specified by the effective address, designated as the multiplier. The product is added to the contents of the accumulator, bit positions 0-107. The sum exponent and fraction replace accumulator bits 0-107. The sign of the sum replaces the accumulator sign, sign byte register bit 4. Accumulator bits 108-127 remain unchanged.

STRAP Notation: *+(N), JOE
Operation Code: 01110

The intermediate product consists of an exponent formed from the sum of the operand exponents and a 96-bit fraction formed as the product of the two 48-bit operand fractions. The intermediate product fraction remains unnormalized throughout the addition. The sign of this product is developed by the rules of algebra, using the sign of the FT register and the effective sign of the addressed operand as developed through modification of this sign by the sign modifiers.

After the multiplication, an addition is performed similar to that described in ADD DOUBLE. When the difference in exponents exceeds 48, the indicator preparatory shift greater than 48 (PSH) is set to one. When the difference in exponents exceeds 95, the number with the smaller exponent is not added. In this case the result is the number with the high exponent. Addition of two 96-bit fractions yields a 96-bit sum fraction and possibly an overflow bit. When normalized operation is specified, the sum is shifted so that the high-order bits of the sum, including the overflow bit, are placed in accumulator bit positions 12-107. The sum exponent is adjusted by the amount of the shift. When noisy mode is specified with normalized operation, one bits are entered in the low-order positions of the 96-bit sum fraction during the left shifting associated with normalization.

This operation initially specifies three operands. They are the operand in the FT register, the operand specified by the effective address, and the operand in the accumulator. The description above considered the details of operation when all operands were in the N range and all intermediate results remained in the N range. When values of the operands or intermediate results are all in the XFP or XFN ranges, the following action is taken.

During multiplication, the operands used are the operand in the FT register and the operand specified by the effective address. If either operand exponent has EF of 1, the product exponent has a propagated EF of 1. When both operands are in the XFP range or both in the XFN range, the product exponent is the exponent of the operand specified by the effective address. When one operand is in the XFP range while the other is in the N or XFN range, the product exponent is the exponent of the operand in the XFP range. When one operand is in the N range while the other is in the XFN range, the product exponent is the exponent of the operand in the XFN range.

For all cases, the intermediate product fraction is the unnormalized fraction determined as the product of the two operand fractions. The normalization modifier applies only to the final result fraction developed during the addition portion of this operation.

During addition, the operands used are the intermediate product developed during multiplication and the operand in the accumulator. When both operands are in the XFP range or both in the XFN range, the result is the operand, exponent and fraction, having the algebraically larger exponent. When both operands have the same exponent value, the operand initially in the accumulator is the result. Normalization, if specified, is suppressed. If one operand is in the XFP range while the other is in the N or XFN range, the result is the operand, exponent and fraction, in the XFP range. Normalization, if specified, is suppressed. If one operand is in the N range and the other is in the XFN range, the intermediate result sum is the operand in the N range. In this case, normalization, if specified, occurs.

The exponent range indicators are set in accordance with the propagated or generated exponent flag when the sum exponent has EF of 1. The sum flag is considered generated if all operands are initally in the N range. It is also considered generated, if the intermediate product has a value in the XFN range with a propagated exponent flag while the accumulator operand is in the N range but the sum has a negative flag because of final normalization. All other cases lead to a propagated EF of 1, in the final result.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Preparatory Shift Greater than 48 (PSH); Exponent Range (XPFP, XPO, XPH, XPL, XPU): These indicators are set as in ADD DOUBLE.

Zero Multiply (ZM): If the final result, after addition, is an order-of-magnitude zero (exponent greater than -1024), this indicator is turned on. Otherwise, if on, it is turned off.

6.20.26 Divide Double (D/)

The operand in the accumulator bits 0-107, the dividend, is divided by the operand specified by the effective address, the divisor. The sign of the divisor is modified by the sign modifiers before the division. The quotient exponent and fraction replace accumulator bits 0-60. Accumulator bits 61-107 are set to zero. The sign of the quotient replaces the accumulator sign. Accumulator bits 108-127 remain unchanged. The remainder is placed in the remainder register, RM address 13. The original accumulator sign becomes the sign of the remainder, RM register bit 60. The flag bits 61-63 of the remainder register are set to zero.

STRAP Notation: D/(N), SQUARE
Operation Code: 10111

As noted in DIVIDE, division is always performed regardless of the original state of normalization or the setting of the normalization modifier, unless the divisor fraction is zero. The partial field indicator (PF) indicates the occurrence of an apparent quotient/

shift in the unnormalized operation while the left zeros counter (LZC) indicates the excess of divisor shift over dividend shift for prenormalization. In the case of a zero divisor, the division is not performed and the zero divisor indicator (ZD) is set to one.

Double precision division is designed to provide 48-bit quotient and remainder fractions from a 96-bit dividend fraction and a 48-bit divisor fraction. Depending upon the relative magnitudes of the dividends and divisor fractions, the remainder may be 48 or 49 bits after a 48-bit quotient is developed. In normalized operation, the entire remainder is normalized and in the case of 49 significant bits, the low-order bit is discarded. In unnormalized operation, the 48 high-order bits of the remainder are stored. In either case, the remainder exponent and fraction are stored in the remainder register, RM, bits 0-59. If the low order 49th bit dropped in storing is a one, the lost carry indicator (LC) is turned on.

All double precision results can be stored as rounded single precision values, by using STORE ROUNDED. In order to allow the quotient of a double precision division to be rounded, a 49th quotient fraction bit is developed after the remainder is preserved.

The details of prenormalization and the generation of the first 48 bits of the quotient fraction in normalized and unnormalized division follow the description given for DIVIDE. The introduction of noisy mode ones however, differs from that of DIVIDE. In DIVIDE DOUBLE, the dividend fraction is assumed to be 96 bits in length. Therefore, noisy mode ones are introduced only into the bit position vacated by left shifting.

In DIVIDE DOUBLE, the remainder is preserved. When the divisor fraction is equal to or smaller than the 48 high-order bits of the dividend fraction, a remainder of 49 bits is obtained. Otherwise a 48-bit remainder fraction is obtained. The intermediate remainder exponent is obtained by subtracting 48 from the adjusted exponent of the shifted dividend.

When normalized operation is specified, the remainder is normalized and placed in RM bit positions 0-59. In the case of a 49-bit remainder, the low-order remainder bit is lost. When unnormalized operation is specified, the remainder is not normalized, but the 48 high-order remainder fraction bits and the remainder exponent are placed unnormalized in RM bit positions 0-59. Again, if a 49-bit remainder was developed, the low-order bit is lost. Lost carry indicator (LC), is set to one if the lost bit is one.

After the remainder is preserved, it is compared with the divisor to obtain a 49th quotient fraction bit. The 49-bit fraction is placed in accumulator bits 12-60. The quotient exponent is placed in accumulator bits 0-11. The sign of the quotient, obtained by the rules of algebra, using the dividend sign and the effective divisor sign, replaced the accumulator sign, sign byte register bit 4. Accumulator bits 61-107 are set to zero. The accumulator data flag bits, sign byte register bits 5-7, remain unchanged.

When normalized operation is specified, the final quotient placed in the accumulator is normalized. When unnormalized operation is specified the quotient may or may not be normalized, depending on the extent of prenormalization shifting of the dividend. If the divisor shift for normalization exceeds the dividend shift for relative normalization or if the quotient had an overflow condition corrected, as in DIVIDE, the contents of the left zeros counter are greater than zero. In unnormalized operation the partial field indicator (PF), is set to one when these conditions exist.

When either or both of the operands are values in the XFP or XFN range, the quotient exponent and quotient fraction are developed as described under DIVIDE except that the dividend fraction is initially bits 12-107 of the accumulator.

When both operands are originally in the N range, either or both may acquire exponent flags, EF of 1, prior to the division. The divisor may become a value in the XFN range through prenormalization. The dividend may become a value in the XFP range through quotient overflow adjustment or a value in the XFN range through prenormalization. When a generated EF of 1 is developed, division proceeds normally except for the following cases. If the dividend has a generated negative flag while the divisor is in the N range, the quotient exponent is the adjusted dividend exponent. If the divisor has a generated plus flag and the dividend has a generated positive flag or is in the N range, the quotient exponent will be the divisor exponent.

When both operands, prior to prenormalization, are in the N range, the remainder exponent is the dividend exponent (after prenormalization and quotient overflow adjustment) minus 48. Normalization, if specified, occurs. If the final remainder exponent has a generated negative flag, the remainder underflow indicator (RU), is turned on.

When the quotient has a propagated EF of 1, the remainder exponent is the same as the original dividend exponent. Normalization of the remainder fraction, if specified, is suppressed. If the final remainder exponent has a propagated negative flag, the turning on of indicator RU is suppressed.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Partial Field (PF); Zero Divisor (ZD); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in DIVIDE.

Lost Carry (LC): The indicator is turned on if a 49 bit remainder is obtained, when unnormalized division is specified, and the low-order bit which was one was dropped.

Remainder Underflow (RU): When the remainder exponent has a generated negative flag this indicator is turned on. If the remainder exponent has a propagated negative flag, the setting of this indicator is suppressed.

Note: Because a 49 bit quotient is obtained, the Result Zero indicator (RZ) is turned on only if all 49 quotient bits are zero.

6.20.27 Add to Fraction (F+)

The operation is the same as ADD DOUBLE except that the exponent of the operand in the accumulator is used for both the addressed operand and the accumulator operand.

STRAP Notation: F+(N), JOE
Operation Code: 10100

The operand specified by the effective address is added to the operand in the accumulator. The exponent of the addressed operand is ignored. Instead, the exponent of the operand in the accumulator is also used as the exponent of the addressed operand. The sum fraction and exponent replace accumulator bits 0-107. Accumulator bits 108-127

remain unchanged. The sign of the addressed operand is modified by the sign modifiers prior to the addition. The sign of the sum replaces the accumulator sign.

When normalized operation is specified, the result fraction is normalized. If noisy mode is also specified, ones enter in the low-order position instead of zeros.

When the accumulator operand is a value in the XFP or XFN range the fraction addition is not performed because the operand specified by the effective address assumes the exponent of the accumulator operand.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Lost Carry (LC); Lost Significance (LS); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD DOUBLE.

6.20.28 Shift Fraction (SHF)

The fraction in the accumulator is shifted in the direction and amount specified by the effective address bits 4-11. A left shift is specified if bit 11 is zero; a right shift, if bit 11 is one. The amount of the shift is specified by bits 4-10. Zeros are inserted in the bit positions which are vacated. Only accumulator bits 12-107 participate in the shifting; all other accumulator bits remain unchanged. If a one bit is shifted left beyond accumulator bit position 12, it is lost and the LC indicator, is set to one. A one-bit shifted right beyond accumulator bit position 107 is lost, but no indicator is turned on. The sign of the shift (bit 11) of the effective address, or shift amount, is subject to modification by the sign modifiers. No change in the exponent field of the accumulator operand is made by this operation. The maximum shift is 127 positions.

STRAP Notation: SHF(U), AMOUNT
Operation Code: 11100

This operation does not address storage. The effective shift may be obtained by standard indexing procedures. The effective shift is not monitored for address boundaries. The operation is not affected by the normalization modifier. No noisy mode one bits are entered on left shift if noisy mode is specified.

When the accumulator operand is a value in the XFP or XFN range the operation is performed as described above. The result is the original accumulator exponent and the fraction shifted in the specified direction by the specified amount.

Indicators

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the arithmetic result which appears in the accumulator.

Lost Carry (LC): This indicator is set to one when a one-bit is shifted left beyond accumulator bit position 12.

Exponent Range (XPFP, XPH, XPL): These indicators are set according to the exponent range of the result as it appears in the accumulator.

Zero Multiply (ZM): If on, this indicator is turned off.

To-Memory Operation (MOP): This indicator is set to zero.

Programming Note

The sign modifiers which apply to bit 11 of the effective address may be stated in the normal STRAP format. For example, SHFN, SHFA, and SHFNA are legal operations. Also the appropriate modifiers can be called for by adding the letter L, for left shifts, or the letter R, for right shifts, to the mnemonic code. Therefore, right shifts may be coded SHFR, AMOUNT, and left shifts may be coded SHFL, AMOUNT.

6.20.29 Add to Exponent (E+)

The exponent of the operand specified by the effective address is added to the exponent of the operand in the accumulator. The addition is algebraic and takes into account the signs of both exponents. The sign of the exponent of the addressed operand is subject to modification by the sign modifiers. The fraction and fraction sign of the addressed operands are ignored. The result of the exponent addition is placed in accumulator bits 0-11. When normalized operation is specified, the accumulator fraction, bits 12-107, is normalized. If noisy mode is also specified low-order one bits instead of zeros are introduced during normalization.

STRAP Notation: E+(N), UPDATE
Operation Code: 10101

When the exponents of either or both operands lie in the XFP or XFN range, the rules for exponent handling follow the definitions given under MULTIPLY. When either operand exponent is flagged the result has a propagated flag and normalization, if specified, is suppressed. In this case the result fraction is the original accumulator fraction. When both operands have EF of 0, the result exponent may have a generated exponent flag. For this case normalization, if specified, occurs if the flag is positive and is suppressed if the flag is negative.

Indicators

Data Flags (TF, UF, VF): These indicators are set according to the flag bits of the address operand.

To-Memory Operation (MOP): This indicator is set to zero.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the entire floating point result which appears in the accumulator.

Exponent Range (XPFP, XPO, XPH, XPL, XPU): These indicators are set according to the exponent range of the results.

Zero Multiply (ZM): If on, this indicator is turned off.

6.20.30 Add Immediate to Exponent (E+I)

The operation is the same as ADD TO EXPONENT, except that the addend is bits 0-11 of the effective address. This operation does not address storage. The effective address field of the instruction is not monitored for address boundaries.

STRAP Notation: E+ I(N), AMOUNT
Operation Code: 11101

The addition is algebraic, taking into account the signs of exponent operands. The sign of the immediate operand, effective address bit 11, is subject to modification by the sign modifiers.

Indicators

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN); Exponent Range (XPFP, XPO, XPH, XPL, XPU); Zero Multiply (ZM): These indicators are set as in ADD TO EXPONENT.

6.20.31 Review of Floating Point Operations

It may appear that the various floating point operations are complex. Actually, the operations themselves are relatively straightforward and only the introduction of exceptional conditions such as flagged exponents make the operations appear complex. The following paragraphs, by reviewing the generalities of floating point operation, are intended to aid the customer engineer in developing the ability to recall the action performed by the various operations.

If the exponents of all operands involved in a particular operation are in the N range, the operation is performed in a normal manner regardless of fraction size. The exception to the statement is a DIVIDE operation involving a zero divisor.

Generally speaking, if N range operands yield generated exponent flags, the operation is performed as specified except that normalization may be suppressed.

When considering propagated flags, it is helpful to remember that a number with an exponent in the XFN range is approximately equal to zero while a number with an exponent in the XFP range is approaching infinity. Operations involving propagated flags are not performed in the normal manner. Rather, the result is "set" to reflect the approximate result to be expected from an operation involving such exponents. In some operations such as ADD, the entire result, exponent and fraction, is "set." In other operations such as MULTIPLY and in DIVIDE, the result fraction is developed and only the result exponent is "set."

The absolute sign modifier normally applies to the operand moving to or from storage. The negative sign modifier normally applies to the unchanged operand. When both modifiers apply to the same operand, the absolute modifier is applied first.

In the operation STORE ROOT, the absolute modifier controls the setting of indicator IR. The sign of the developed root is always positive and may be unconditionally set negative by the negative sign modifier.

In compare operations, both modifiers apply to the storage operand.

No programming examples of the double precision operations have been included in this section of the manual. It is felt that the use of these operations will be more readily understood after the customer engineer has studied PAU. Double precision programming examples are presented in the 7030 Data Processing System Reference Manual, Form A22-6530.

6.21 VARIABLE FIELD-LENGTH INSTRUCTIONS

The variable field length operations may be divided into three general classes: integer arithmetic operations, connective operations. and radix conversion operations. The arithmetic, connective and radix conversion operations are presented in this section of the manual.

6.21.1 Progressive Indexing

In each of the variable field length instructions, a special type of indexing may be specified, termed "progressive indexing." Bit positions 32-34 of the instruction format specify the type of progressive indexing required by the instruction. These bits are also used to specify direct or immediate addressing if progressive indexing is not to be used.

In progressive indexing, the value field of the index word specified by I, positions 28-31, of the instruction format, is the effective operand address of the instruction. There is no algebraic addition of the address portion of the instruction and the contents of the value field of an index word to obtain an effective address. Bits 0-23, of the value field contained in the specified index word, are used directly as the effective address. Bit 24, the sign of the value field, is not considered when determining the effective address.

After the value field is used as the effective address, the address portion of the instruction, bits 0-23, is added to the value field. The sum of this addition is placed back into the value field. The address portion of the instruction is considered as an unsigned, positive quantity. The addition is algebraic, taking into account the sign, bit 24, of the value field.

In addition to progressive indexing as described above, other variations may be specified. These variations are presented in Figure 6.20-1.

The STRAP notation for progressive indexing is inserted within parentheses between the VFL operation mnemonic and the (dds) field.

Progressive indexing combines immediate index incrementing with any one of the variable field length operations. It is possible only when the effective address can be specified in an index quantity and requires no further address modification.

The index flag and index result indicators are set following each progressive index operation. They are set according to the new contents of the index register specified by I. The setting occurs prior to the REFILL which may be specified.

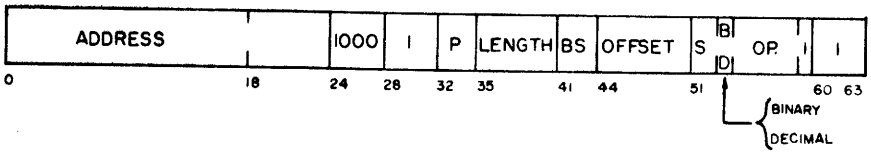
P-Field (Bits 32-34)	Type of Address Modification	Description
000	Direct Addressing	The value field of the specified index word is added to the address portion of the instruction to obtain the effective address of the data.
100	Immediate Addressing (I) --not available for VFL Store Operation	The effective address, as formed in direct addressing, is considered as the data and a data fetch is not required.
001	Progressive--Add to Value (V + I)	Progressive Indexing--after the value field has been used as the effective address, the address portion of the instruction as algebraically added to this value field. The sum is placed into the value field.
010	Progressive--Add to Value and Count (V +IC)	The operation is the same as Add to Value except that the count field of the specified index word is reduced by one as a part of the operation.
011	Progressive--Add to Value, Count and Refill. (V + ICR)	The operation is the same as Add to Value and Count with the exception that, if counting causes the count field to reach zero, a refill operation is performed.
101	Progressive--Subtract from Value (V - I)	After the value field of the specified index word is used as the effective address, the address portion of the instruction is algebraically subtracted from this value field. The difference is placed into the value field.
110	Progressive--Subtract from Value and Count (V - IC)	The operation is the same as V - I except that the count field of the index word specified by I is reduced by one.
111	Progressive--Subtract from Value Count and Refill (V - ICR)	The operation is the same as V - IC, except that, if the counting causes a zero count to be reached, the index word is refilled by the contents of the location specified by the refill address of the index word.

FIGURE 6.21-1. ADDRESS MODIFICATION SUMMARY

The index field I controlling progressive indexing is that located in the first half of the full length instruction. The index field I in the second half of the instruction may be used to modify bits 32-50 of the instruction. In so doing the progressive indexing code is not changed. If the same index register is specified in both halves of an instruction that specifies progressive indexing, the value of the index register before modification will be used in indexing the second half of the instruction.

6.21.2 Integer Arithmetic Instruction Format

The integer arithmetic operations use the full-word instruction format shown below:



STRAP Format: OP(dds), A24(I), OF7(I)

The (dds) field of the STRAP format is used to indicate binary or decimal, signed or unsigned, as well as field length and byte size.

Bits 0-23 may be used as a word and bit address. This field may be indexed by the contents of the index register specified in the I field, bits 28-31. The effective address is used to designate the leftmost bit of a storage field, or it may be used as data. The three high-order bits of the second half-word form the P field of the instruction and specify the choice between direct and immediate addressing and progressive indexing. Immediate addressing is specified by a one bit in position 32. The two low-order positions of the P field are used only to specify progressive indexing. Although the second half-word may be indexed, the P field is not modified. The index field I in positions 60 through 63 specifies the index word used to modify bits 35 through 50 of the instruction. On indexing the second half-word, position 3 of the index value is aligned with position 35 of the instruction, the three high-order bits of the index word being ignored.

To field length, bits 35-40, defines the length of the operand from storage, and the length of the result field when the result replaces the storage operand. The offset field, bits 44-50, specifies the number of bits by which the right end of the accumulator operand is offset from the right end of the accumulator. The byte size field, bits 41-43, specifies the byte size of the storage operand.

The operation code designates the operation to be performed. All integer arithmetic operations are subject to three modifiers specified in bits 51-53 of the instruction. One modifier, bit 51, designates whether the memory field is signed or not; another, bit 52, designates whether the sign of the unreplaced field is to be taken as-is or inverted; the third, bit 53, designates whether the operation is decimal or binary.

6.21.3 Field Definition

The operand specified explicitly by an instruction is defined by a word address, bit address and field length. If direct addressing is specified, the indexed word and bit address are used to define the leftmost bit of a storage field. The field length specifies

the total length of the storage operand, including the sign byte when present. A length field of zero in an instruction specifies a 64-bit storage field.

The storage word or word pair which contains the addressed storage field is brought from storage to the serial arithmetic unit. There, the desired field is extracted starting with the right end of the field. Processing is done from right to left, even though the bit address in the instruction specifies the leftmost bit of the field.

In integer arithmetic operations which affect storage, such as STORE and ADD TO MEMORY, the result replaces the storage operand in the bit positions specified by the address and field length. Data to the right or left of the specified field in storage are not affected.

If immediate addressing is specified, the effective address itself is used as the operand. The field length specifies the number of bits in the field starting with the leftmost bit of the effective address. The sign bit (position 24) of the effective address is not used. If the field length designated is greater than 24, zeros are added to the right to obtain a field of proper length. The resulting field is treated in every respect like a field from storage. Immediate addressing is not possible in operations where the result is placed in storage. If immediate addressing is specified in a to-memory operation, the operation is not performed and the operation code invalid (OP) indicator is turned on.

The implied second operand of most operations is a field from the accumulator. The right end of the accumulator operand is defined by the offset specified in the instruction. The left end of the field is at the left end of the accumulator. For example, any carries which occur in an accumulator-altering operation such as ADD may propagate to the left end of the accumulator. If a carry attempts to propagate beyond the left end of the accumulator, the carry is lost and the lost carry indicator is turned on.

The entire contents of the accumulator are treated basically as a unit. This means that all bits in the accumulator, even those to the right of the offset, can be changed as a result of an accumulator-altering operation. This occurs when such an operation causes the accumulator to be cleared or recomplemented. For example, LOAD clears the entire accumulator regardless of the offset specified. Recomplementation occurs when the sign of the accumulator changes as a result of an operation. In this event, the entire accumulator is recomplemented regardless of the offset.

In accumulator-altering operations, the offset can be conveniently thought of as equivalent to specifying the number of numeric zero bits to be added to the low-order end of the storage operand before it is combined with the accumulator contents. In storage-altering operations, the contents of the accumulator to the right of the offset are ignored.

In decimal operations, the accumulator is treated as 32 four-bit digit positions. Decimal fields may occupy any set of adjoining digit positions in the accumulator. In decimal operation, the offset is limited to decimal digit positions, and hence to a multiple of four, by ignoring the two low-order bits of the offset specified by the effective second half-word.

6.21.4 Sign Control

All the integer arithmetic operations include two modifiers, bits 51 and 52, which affect the use of the sign of one or both of the operands.

Numeric data fields in storage may be signed or unsigned. If the field is signed, the rightmost byte is the sign byte. The size of the sign byte is specified by the byte size. The sign of the accumulator operand is contained in position 4 of the accumulator sign byte register.

Instruction bit 51, the unsigned modifier, designates whether the addressed storage operand contains a sign byte or not. If the bit is zero, the operand is signed; if the bit is one, the operand is unsigned. A positive sign is assumed for an unsigned field and all bytes including the rightmost byte are processed as data.

Instruction bit 52, the negative sign modifier, designates whether the sign of the unreplaced operand is taken as-is or inverted before processing takes place. If the bit is zero, the sign is used as-is; if the bit is one, the sign is inverted. In general, if the result of an operation is placed in the accumulator, bit 52 determines the use of the sign of the operand from storage; and if the result is placed in storage bit 52 determines the use of the accumulator sign. The negative sign modifier, when one, has the effect of changing algebraic additions to subtractions or changing the sign of the result of multiplications, divisions, or stores.

Because the accumulator is addressable, it may also be used as the storage operand of any data handling operation. The left half of the accumulator has address 8; the right half has address 9; and the accumulator sign byte register has address 10. The accumulator contents can be used as both operands of an operation. For example, the amount in the accumulator can be added to or subtracted from itself.

When the accumulator is used as the addressed operand in a signed operation, the low-order bits of the addressed accumulator field are used as the sign byte as in any addressed storage operand. The contents of the sign byte register are not used as the sign and flag bits of the addressed accumulator operand unless the addressed field includes bit 0-7 of word 10 as the low-order bits of the field. The sign and flag bits in the sign byte register apply to the accumulator contents only when the accumulator is the implied operand of an operation.

6.21.5 Data Flag and Zone Bits

In some applications it may be desirable to mark certain data to indicate that special handling is required. The data flag bits in the sign byte of signed data fields allow such marking.

All integer arithmetic operations that obtain data flag bits from storage place these bits into the indicator register. LOAD WITH FLAG also places the flag bits into the three rightmost bits of the accumulator sign byte register. The only other integer arithmetic operation which affects the data flag positions of the sign byte register as an inherent part of its operation is LOAD, which sets these three bit positions to zero.

Operands in storage may have associated with them sign bytes ranging in size from one to eight bits. The eight possible sign bytes and their corresponding positions in the accumulator sign byte register are:

Byte Size	Sign Byte	Position in Sign Byte Register
		0 1 2 3 4 5 6 7
1	S	0 0 0 0 S 0 0 0
2	ST	0 0 0 0 S T 0 0
3	STU	0 0 0 0 S T U 0
4	STUV	0 0 0 0 S T U V
5	ZSTUV	0 0 0 Z S T U V
6	ZZSTUV	0 0 Z Z S T U V
7	ZZZSTUV	0 Z Z Z S T U V
8	ZZZZSTUV	Z Z Z Z S T U V

Here S is the sign bit, T, U, and V are data flag bits, and the bits marked Z are zone bits. The sign and flag bits may be placed in or obtained from the 8-bit accumulator sign byte register. The sign byte is positioned in such a way that the sign bit always occupies position 4 in the register. Zone bits in the sign byte of a signed storage operand do not replace the zone bit positions in the sign byte register as an inherent part of any operation. Both the zone and data flag bit positions of the sign byte register may be changed by addressing them as the storage operand of a to-memory operation.

Operations that place a signed result in storage, with the exception of STORE and STORE ROUNDED, do not alter the original flag bits or zone bits of the storage operand sign byte. STORE and STORE ROUNDED replace the data flag and zone bits in the sign byte with the contents of the corresponding bit positions in the accumulator sign byte register.

6.21.6 Radix and Byte Size

Bit 53 of the instruction, the radix modifier, specifies the type of arithmetic to be performed, the interpretation of the byte size field in the instruction, and the byte size of the accumulator operand. Binary arithmetic is specified if the bit is zero; decimal arithmetic is specified if the bit is one.

In a binary operation, the byte size field, bits 41-43, of the instruction defines only the length of the sign byte. The remainder of the field is processed in eight-bit bytes with the possible exception of the last byte whose size is the number of bits necessary to complete processing of the specified field length. In signed binary operations to storage, the stored result includes a sign byte of the size specified.

If decimal arithmetic is specified, the byte size of the instruction defines the length of each byte in the storage field including the sign byte if the data are signed. Decimal digits are represented in four-bit binary-coded decimal form and, therefore, can be stored in very compact form. The number of bits used to represent a digit in storage can be more than four bits, however; for example, two zone bits may be added to allow decimal information to be readily interspersed with alphabetic data coded in six-bit form. Decimal arithmetic can be performed on data of any byte size. If the byte size is greater than four, only the four low-order bits of each byte take part in the operation, and the remaining bits of each byte are ignored. If a byte size less than four is specified in a decimal operation, each byte of the storage operand is converted to a four-bit byte

for processing by the addition of high-order zeros. In all decimal operations the byte size of the accumulator is assumed to be four regardless of the byte size and field length of the storage operand.

Because the field length is not constrained to be multiples of the byte size, the high-order byte of the storage operand may have an effective byte size less than that specified in the operation. The high-order byte is processed according to its effective byte size rather than the byte size specified in the operation.

If the field length of a signed operation is less than the byte size, the byte size is used to determined the format of the sign byte so that the bits specified are properly used. For example, a LOAD with byte size four and field length two treats the two addressed bits as the U and V flags and sets the corresponding positions of the indicator register accordingly. The accumulator and the four low-order bits of sign byte register are cleared, since there is no sign or data in the addressed field.

In decimal operations which alter storage, the result byte generated by processing a byte from the storage field and a byte from the accumulator replaces the byte which was obtained from the storage field. The replacement is made in accordance with the byte size of the storage field. If the specified byte size is greater than four, the four bits in each result digit replace the four low-order bits of each byte of the storage field. In STORE and STORE ROUNDED, the zone bits of each byte in the storage field are replaced by the contents of corresponding zone positions in the accumulator sign byte register; they remain unchanged in all other arithmetic operations to storage. If the specified byte size is less than four, each byte of the storage field is replaced by the number of bits specified by the byte size. These bits are the low-order bits of each result byte. Higher order bits in each result byte are dropped, and no indication is given if these positions contain one bits. If the high-order byte of the addressed storage field is less than the byte size, it is replaced by the low-order bits of the corresponding result byte. Higher order bits are dropped, and no indication is given if they are non-zero.

6.21.7 Indicators

Several indicators in the indicator register are set as a result of the integer arithmetic operations. These include the general result exception, the data flag, the transit, and the arithmetic result indicators. The general result exception and the transit indicators are permanent. If they are turned on by any operation, they will remain on until they are turned off either by causing an interruption or by being explicitly made zero by an operation which addresses them as an operand. The data flag and arithmetic result indicators are temporary. At the completion of an integer arithmetic operation, all indicators of this type which apply to the particular operation will be on if the necessary condition arose during the operation; otherwise, they will be off. They then remain set until another operation which affects them is executed.

The setting of the indicators is explained under the various instructions which affect them.

6.21.8 Add (+)

The addressed operand is algebraically added to the contents of the accumulator with specified offset. The sign bit in the accumulator sign byte register is set in accordance with the sign of the algebraic sum. The remaining bits in the sign byte register remain unchanged.

STRAP Notation: +(BU, 64,8), JOE, 64
Operation Code: (54-58): 00000

In the ADD operation, the unsigned modifier applies to the addressed operand. The negative sign modifier designates whether the sign of the addressed operand is taken as-is or inverted before the operation takes place. Unsigned fields are considered positive before the negative sign modifier is applied.

The radix modifier specifies the accumulator byte size, determines the properties of the adder, and controls complementation and recomplementation when the effective operand signs are unlike. In decimal arithmetic four-bit bytes are used. Only the four low-order bits of each byte in the addressed operand participate in the operation and a carry occurs from one byte to the next when a sum byte exceeds 9. In decimal operations, the two low-order bits of the effective offset are ignored and considered zero. Thus, the offset is always a multiple of four. In binary arithmetic, an eight-bit byte is used whenever possible. Any offset may be specified in a binary operation.

The particular STRAP statements used to apply the various modifiers are illustrated below.

STRAP Notation	Modifiers Applied	Definition
+(B, 64, 8),	None	Add-Binary-Signed
+(BU, 64, 8),	Unsigned	Add-Binary-Unsigned
-(BU, 64, 8),	Unsigned-Negative	Subtract-Binary-Unsigned
-(B, 64, 8),	Negative	Subtract-Binary-Signed
+(D, 20, 4),	Radix	Add-Decimal-Signed
-(D, 20, 4),	Radix-Negative	Subtract-Decimal-Signed
+(DU, 24, 4),	Radix-Unsigned	Add-Decimal-Unsigned
-(DU, 30, 5),	Negative-Radix-Unsigned	Subtract-Decimal-Unsigned

Indicators

Lost Carry (LC): This indicator is actuated if information is lost because a carry attempts to propagate beyond the left end of the accumulator.

Partial Field (PF): This indicator is turned on if significant bits of the data field from storage overlap the left end of the accumulator. The accumulator operand is extended with high-order zeros to combine with the storage operand, but the high-order bits of the result are dropped. A partial field condition will not cause a lost carry indication. However, both conditions may arise during the execution of a single instruction.

Data Flags (TF, UF, VF): These indicators are set according to the flag bits of the storage operand.

To-Memory Operation (MOP): This indicator is turned off.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the contents of the entire accumulator when the operation is complete.

Programming Note

The sign modifiers apply to the storage operand in the ADD operation. Note that using the negative sign modifier causes an algebraic subtraction, whether the field addressed is signed or unsigned. There is, therefore, no need for a SUBTRACT, and no separate operation is provided. Furthermore, using the unsigned modifier is equivalent to absolute value addition or subtraction; there is no sign on the addressed operand so a plus sign is assumed. The accumulator sign is not ignored, but influences the operation in the usual manner. All add-type operations operate upon the contents of the entire accumulator. If an addition of operands with unlike signs is performed, there is a possibility of sign change. When this occurs, the whole contents of the accumulator may be changed, including the part below the offset. Consider the following decimal operation.

(1) ADD→ (376+) offset 8 bits (2 bytes) where acc. contains 75679+	75679+ 37600+ 113279+	Result Greater than Zero Indicator
(2) ADD→ (532-) offset 8 bits where acc. contains 45623+	45623+ 53200- 7577-	Result Less than Zero Indicator Result Negative Indicator
(3) ADD→ (532-) offset 8 bits where acc. contains 53200-	53200+ 53200- 00000+	Result Zero Indicator
(4) ADD→ (532+) offset 8 bits where acc. contains 53200-	53200- 53200+ 00000-	Result Zero Indicator Result Negative Indicator
(5) ADD→ (532-) offset 8 bits where acc. contains 53201+	53201+ 53200- 00001+	Result Greater than Zero Indicator

The result in every case is the same as one would get performing the same operations by hand, if one considers the offset as describing the number of low-order zeros which are to be attached to the addressed operand. On any operation in which the result in the accumulator is all zero, the sign remains as it was before the operation began.

6.21.9 Add to Magnitude (+MG)

The addressed operand is algebraically added to the magnitude of the contents of the accumulator with specified offset. The accumulator operand is assumed positive for purposes of the addition itself. If the result is positive, the sum is placed into the accumulator. If the result is negative, the entire accumulator is cleared to zero. In either case, the contents of the accumulator sign byte register are not changed.

STRAP Notation: +MG(B, 48, 6), NAME, 64
Operation Code: 10000

The negative, unsigned, and radix modifiers operate as in ADD. The indicators are also set as in ADD.

6.21.10 Load

The entire left and right halves of the accumulator (128 bits) and the four low-order bits of the accumulator sign byte register are cleared. The numeric part of the addressed operand is placed in the accumulator as specified by the offset. The sign bit in the sign byte register is set according to the sign of the data as affected by the sign modifiers. The zone bits of the accumulator sign byte are not altered.

STRAP Notation: L(B, 64, 8), NAME, 64
Operation Code: 00010

The negative, unsigned, and radix modifiers operate as in ADD.

Indicators

Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in ADD.

6.21.11 Load with Flag (LWF)

LOAD WITH FLAG is the same as LOAD except that if the addressed storage field includes data flag bits, these bits are placed in the accumulator sign byte register in addition to setting the data flag indicators.

STRAP Notation: LWF (D, 28, 4), NAME, 0
Operation Code: 10010

The negative, unsigned, and radix modifiers operate as in ADD.

Indicators

Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in ADD.

Programming Note

LOAD, LOAD WITH FLAG, and LOAD CONVERTED are the only variable field length data handling operations that affect the data flag bit positions in the accumulator sign byte register as an implied operand. All other operations of this type leave these positions unchanged. No operation changes the zone bit positions in the accumulator sign byte register as an implied operand.

Programming Example

The following problem illustrates a simple use of ADD TO MAGNITUDE. Given two 54-bit unsigned binary numbers x and y, put the larger of the two in the accumulator.

Name	Instruction	Remarks
START	L(BU,64,8),X,0	Zero offset - place X in B register.
TRY	-MG(BU,64,8),Y,0	Either eliminate X or reduce X by the amount of Y
FIX	+MG(BU,64,8),Y,0	Either replace X with Y or return X to original value.

If X is the larger number, the SUBTRACT MAGNITUDE at location TRY merely reduces X by the amount of Y. The ADD at location FIX then returns the accumulator to the original value of X. On the other hand, if Y is the larger number, the instruction at TRY clears the accumulator. The ADD at FIX then places Y in the accumulator. Thus, after executing the instruction at location FIX, the accumulator contains the larger number. Note that a zero offset is specified. This causes the result to occupy the right half (B) of the accumulator.

6.21.12 Add to Memory (M+)

The accumulator field specified by the offset is algebraically added to the addressed operand. The accumulator sign is used as modified by the negative sign modifier, and the sum replaces the addressed field in storage.

STRAP Notation: M+ (D, 36, 6), NAME, 0
Operation Code: 00100

Modifiers

Unsigned: This modifier describes the addressed field in storage. If the operation is signed, the sign bit of the storage operand is set to the sign of the algebraic sum.

Negative Sign: This modifier applies to the accumulator sign as used in the operation. The original accumulator sign remains unchanged in the sign byte register.

Radix: In a binary ADD TO MEMORY the byte size refers only to the length of the sign byte, thereby determining the position of the sign bit. The remainder of the field is processed in eight-bit bytes whenever possible.

In a decimal operation, the field is processed in four-bit bytes. If the byte size of a decimal storage field participating in an ADD TO MEMORY is greater than four, only the four low-order bits of each byte participate in the addition. The high-order bits of each byte are not changed by the operation. If a byte size less than four is specified in a decimal operation, each byte from the storage field is converted to a four-bit byte for processing by the addition of high-order zeros. Normal decimal results may not be produced in such cases. Only the low-order bits of each sum byte replace the byte from storage so that the stored result has the same byte size as the original storage field. Because normal decimal addition takes place, the high-order positions which are dropped from each sum byte may contain one bits. These are lost and no indication of the loss is given.

Indicators

Lost Carry (LC): This indicator is activated by a carry beyond the leftmost digit, either binary or decimal, of the addressed field. This indicator is also turned on if

the result of an unsigned operation is negative. In such a case, the magnitude of the result is stored.

Partial Field (PF): This indicator is turned on if there are non-zero digits, either binary or decimal, in the accumulator to the left of the highest order digit that is participating in the operation.

Data Flags (TF, UF, VF): These indicators are set according to the data flag bits of the storage operand. If the operation is unsigned, they are set to zero.

To-Memory Operation (MOP): This indicator is turned on.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the algebraic sum of the two operands, whether the sign is stored or not.

Programming Notes

The data flag bits of a signed operand in storage are not altered by ADD TO MEMORY.

The accumulator contents are not altered by ADD TO MEMORY.

If the negative sign modifier is one, ADD TO MEMORY will perform an algebraic subtraction of the accumulator contents from the addressed field. Although the negative sign-modifier applies to the accumulator sign in to-memory operations, the unsigned modifier always applies to the operand in storage. Unsigned storage fields are considered positive.

In operations to memory, the length of the accumulator field between the specified offset and the left end of the accumulator can be less than the number of bits from the storage operand which are to take part in the operation. In this event the accumulator operand is lengthened for the operation by the addition of high-order zeros.

6.21.13 Add Magnitude to Memory (M+MG)

The magnitude of the accumulator field as specified by the offset is algebraically added to the addressed operand. A positive sign is assumed for the accumulator field before the negative sign modifier is applied. The actual accumulator sign is not used or changed by the operation. If the sum has the same sign as the sign of the storage operand, it replaces that operand in storage. If the sum is of opposite sign, zero bytes replace the numeric part of the storage field. In either case the sign byte, if any, of the storage operand is not changed.

STRAP Notation: M+MG(D,60,4), NAME, 68
Operation Code: 10100

Modifiers

Unsigned: This modifier designates whether or not the storage operand contains a sign byte.

Negative Sign: The accumulator sign is considered positive when the negative sign modifier is zero and negative when the modifier is one.

Radix: This modifier operates as in ADD TO MEMORY.

Indicators

Lost Carry (LC); Partial Field (PF); Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD TO MEMORY.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the final result stored.

Programming Note

The four operations ADD, ADD TO MAGNITUDE, ADD TO MEMORY, and ADD MAGNITUDE TO MEMORY allow a flexible handling of the signs of the factors and the result. Not all possible combinations are provided, however, and in some cases the results may not be obvious. The following table shows the storage or accumulator field that is produced by all possible combinations of operations, sign modifiers, operand signs, and relative magnitudes of operands.

The storage field is assumed to be signed in all cases. When the unsigned modifier is specified, the field length is adjusted accordingly and the absolute value of the storage field is used in the operation. However, in an unsigned to-memory operation, the original sign of the field remains in storage and is therefore shown in the table as part of the result field.

Memory Operand	3+	5+	3+	5+	3-	5-	3-	5-
Accumulator Operand	5+	3+	5-	3-	5+	3+	5-	3-
Operation and Modifiers								
Add	8+	8+	2-	2+	2+	2-	8-	8-
Add, N	2+	2-	8-	8-	8+	8+	2-	2+
Add, U	8+	8+	2-	2+	8+	8+	2-	2+
Add, U,N	2+	2-	8-	8-	2+	2-	8-	8-
Add to Magnitude	8+	8+	8-	8-	2+	0+	2-	0-
Add to Magnitude, N	2+	0+	2-	0-	8+	8+	8-	8-
Add to Magnitude, U	8+	8+	8-	8-	8+	8+	8-	8-
Add to Magnitude, U,N	2+	0+	2-	0-	2+	0+	2-	0-
Add to Memory	8+	8+	2-	2+	2+	2-	8-	8-
Add to Memory, N	2-	2+	8+	8+	8-	8-	2+	2-
Add to Memory, U	8+	8+	2+*	2+	8-	8-	2-*	2-
Add to Memory, U,N	2+*	2+	8+	8+	2-*	2-	8-	8-
Add Magnitude to Memory	8+	8+	8+	8+	0-	2-	0-	2-
Add Magnitude to Memory, N	0+	2+	0+	2+	8-	8-	8-	8-
Add Magnitude to Memory, U	8+	8+	8+	8+	8-	8-	8-	8-
Add Magnitude to Memory, U,N	0+	2+	0+	2+	0-	2-	0-	2-

* The lost carry indicator is turned on.

Underlined results differ from those produced from a similar floating point operation.

6.21.14 Add One to Memory (M+ 1)

A plus or minus one is algebraically added to the addressed storage field.

STRAP Notation: M+1 (B, 64, 8), NAME.0,
Operation Code: 10101

The offset field in this instruction has no meaning and is ignored.

Modifiers

Unsigned: This modifier operates as in ADD TO MEMORY.

Negative Sign: This modifier specifies the sign of the bit to be added. If the modifier is zero, a plus one is algebraically added to the storage operand; if one, a minus one is added.

Radix: This modifier operates as in ADD TO MEMORY.

Indicators

Lost Carry (LC); Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators operate as in ADD TO MEMORY.

The partial field indicator is not affected by this operation.

Programming Note

ADD ONE TO MEMORY is identical to an ADD TO MEMORY operation in which the accumulator operand is +1. This operation can be used for ordinary adding or counting by ones in signed or unsigned fields. Counting by any power of the specified radix (2 or 10) can be performed in fields whose sign is known by an unsigned operation with proper bit address.

6.21.15 Store (ST)

The addressed field in storage is replaced by the accumulator field specified by the offset. If a signed operation is specified, a sign byte of proper size is obtained from the accumulator sign byte register. After modification by the negative sign modifier, this sign byte replaces the rightmost byte of the storage field. The accumulator and the sign byte register remain unchanged.

STRAP Notation: ST(B, 32, 4), NAME .32, 96
Operation Code: 00110

Modifiers

Unsigned: This modifier determines whether the stored field is signed or unsigned. When the field is signed, the contents of the accumulator sign byte register replace the rightmost byte of the addressed field. If the byte size is less than eight, only part of

the contents of the sign byte register is stored. The number of bits in the sign byte stored is determined by the specified byte size.

Negative Sign: If this modifier is one, the sign bit from the accumulator sign byte register is inverted before being placed in the storage field. The contents of the accumulator sign byte register remain unchanged. If the unsigned modifier is specified, the negative sign modifier has no effect on the field stored. However, the result indicators are set according to the modified accumulator sign, whether it is stored or not.

Radix: In a binary STORE, the byte size refers only to the length of the sign byte stored. The specified accumulator data field is stored in eight-bit bytes wherever possible. In a decimal STORE, the byte size refers to the size of each byte in the storage field, including the sign byte if the operation is signed. Each four-bit byte of the accumulator replaces the four low-order bits of each byte of the storage field except the sign byte. If the specified byte size is greater than four, the remaining high-order bits of each byte are replaced by the corresponding zone bits from the accumulator sign byte register. If the specified byte size is less than four, only the low-order bits from each accumulator byte replace each byte in the storage field.

Indicators

Partial Field (PF): The partial field indicator is turned on in a binary STORE if there is any one bit in the accumulator to the left of the last bit position stored. In a decimal STORE, this indicator is turned on if there is a non-zero byte to the left of the high-order byte stored. If the field length is not a multiple of the byte size in a decimal STORE, only part of the high-order digit is stored so that only the addressed field is replaced in storage. In order to set the PF indicator, there must be a non-zero byte to the left of the high-order accumulator byte. These non-zero bytes are to the left of the high-order accumulator byte, whether or not the entire high-order byte participates in the operation.

To-Memory Operation (MOP): This indicator is turned on.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the data field stored. The sign of the accumulator, as modified by the negative sign modifier, affects the setting whether it is stored or not.

Programming Notes

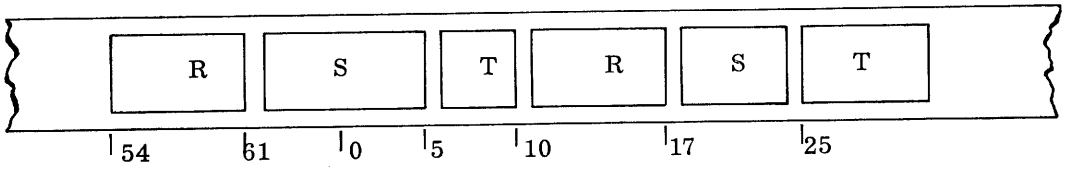
The field length and offset specified in a STORE instruction may be such that the field to be stored extends beyond the left end of the accumulator. In this event, the accumulator operand is lengthened for the operation by the addition of high-order zeros. In a binary operation, this results in replacing the left end of the storage field with zeros. In a decimal operation, the high-order part of the storage field is replaced by bytes whose numeric part is zero and whose zones, if any, are those provided from the sign byte register.

A signed STORE stores sign bytes with the data flag bits and zone bits which are in the accumulator sign byte register. To maintain the original flag bits on a data field which is to be processed and then stored, LOAD WITH FLAG should be used when the field is initially obtained. Remember that LOAD clears the data flag positions in the

sign byte register and does not place the flag bits of the addressed operand into these positions. Desired zone bits must be placed into the zone bit positions of the sign byte register by explicitly addressing these positions as the operand of a to-memory operation. These positions will then remain unchanged until they are once again explicitly addressed in a to-memory operation.

Programming Example

In storage starting at location LIST is a list of ten data records. Each record consists of three fields, R, S, and T, each an unsigned binary integer. Field R is 7 bits long, field S is 8 bits long, and field T is 5 bits long. For each record add field T to field R and place the result in field S. This example illustrates the use of progressive indexing.



Name	Instruction	Remarks
PREP	LX, $X12, CTLWD	Load first address into XR12.
LOOP	L(V+I) (BU,7,8),0.15 ($X12), 0	Load R into accumulator. Ready XR12 for T.
	+(V-I)(BU,5, 8), 0.08 ($X12),0	Add T to R. Ready XR12 for S.
	ST(V+IC)(BU,8,8),0.13 ($X12),0	Store in S. Ready XR12 for next R.
	BZXCZ, LOOP	Repeat nine times.
CTLWD	XW, LIST, 10, CTLWD	LIST = XX.54

6.21.16 Store Rounded (SRD)

The field from the accumulator is rounded by adding to its magnitude one-half of the proper radix in the digit position to the right of the offset. Carries are propagated as in any addition in that radix. This rounded field replaces the addressed storage field. If a signed operation is specified, the contents of the accumulator sign byte register replace the rightmost byte of the storage field after the sign bit has been modified by the negative sign modifier. The contents of the accumulator and the sign byte register are not changed by this operation. The accumulator sign is ignored in the actual rounding process.

STRAP Notation: SRD(DU,20,4), NAME .0, 108
Operation Code: 10110

Modifiers

Unsigned; Negative Sign: These modifiers operate as in STORE.

Radix: This modifier acts in the same manner as in STORE, except that in addition it governs the rounding process. If a binary operation is specified, a binary one is

added to the field from the accumulator in the bit position immediately to the right of the specified offset. Carries are propagated as in any binary addition. If a decimal operation is specified, a four-bit byte with value five is added to the four-bit byte immediately to the right of the offset in the field from the accumulator. Carries are propagated as in any decimal addition. If zero offset is specified, the operation is performed as STORE.

Indicators

Partial Field (PF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in STORE.

Lost Carry (LC): This indicator is turned on when the rounding process causes a carry to be propagated beyond the left end of the field stored.

6.21.17 Compare (K)

The contents of the accumulator left of the specified offset are algebraically compared with the addressed operand. The bits to the right of the offset do not participate in the comparison. The comparison result indicators are set to describe the accumulator field as compared with the storage field. Neither operand is changed by the operation. The arithmetic result indicators are not affected.

STRAP Notation: K(B, 64, 8), NAME .0, 64
Operation Code: 01000

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Accumulator Low (AL): This indicator is turned on if the accumulator operand is algebraically less than the storage operand.

Accumulator Equal (AE): This indicator is turned on if the accumulator operand is algebraically equal to the storage operand.

Accumulator High (AH): This indicator is turned on if the accumulator operand is algebraically greater than the storage operand.

Programming Note

Comparison operates numerically and algebraically. Therefore:

1. If the magnitude of the two comparands are represented by x and y, where $x > y > 0$, then the comparison sequence of the comparands with sign can be obtained from the statement

$$+x > +y > +0 = -0 > -y > -x$$

In other words, if the signs of both operands are positive, the operand with greater magnitude is considered high. If the signs of the operands are unlike, the operand with positive sign is considered high and the operand with negative sign is considered low. If the signs of both operands are negative, the more negative operand, that with the greater magnitude, is considered low. A negative zero is considered equal to a positive zero.

2. Alphabetical comparison, including zone bits, requires that the binary modifier be used. An alphameric code can readily be constructed to give any desired collating sequence.

3. The comparison result indicators correspond to and substitute for the arithmetic result indicators in comparison operations. The arithmetic result indicators are not changed by a comparison operation.

4. The comparison is performed by making a subtraction and testing the result. A decimal digit with byte size less than four is handled as in any other subtraction.

If the field length and offset are such that high-order positions of the storage field compare with positions to the left of the left end of the accumulator, the accumulator field is extended by adding zeros to its left before the comparison is made.

Neither the lost carry nor the partial field indicator can be turned on by a comparison operation.

6.21.18 Compare Field (KF)

A field in the accumulator is algebraically compared with the addressed operand. This accumulator field is the same length in bytes as the field from storage, exclusive of the sign byte, and its location is specified by the offset in the usual manner. The actual accumulator sign is not used by the operation. A positive sign is assumed for the accumulator field. The sign of the storage field is used as modified by the sign modifiers. The comparison result indicators are set to describe the accumulator field as compared with the storage field. Neither operand is changed by the operation. The arithmetic result indicators are not affected.

STRAP Notation: KF(BU, 64, 8), NAME .0, 64
Operation Code: 11000

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH): These indicators are set as in COMPARE.

Programming Notes

The field comparison operations are the only arithmetic operations that do not treat the entire accumulator contents as a single quantity. Comparisons are made between

fields of equal length in numeric bytes. For this reason, no indication is given in these operations if there is a one-bit in the accumulator to the left of the field which is being compared with the storage operand.

6.21.19 Compare if Equal (KE)

If the accumulator equal indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is executed exactly as in COMPARE.

STRAP Notation: KE(BU, 64, 8), NAME .0, 64
Operation Code: 01001

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in COMPARE.

Comparison Result (AL, AE, AH): These indicators are set as in COMPARE if the accumulator equal indicator is on when the operation is initiated. If the accumulator equal indicator is off, these indicators are not affected by the operation.

Programming Note

COMPARE IF EQUAL can be used with COMPARE to obtain a comparison of fields of more than 64 bits. A COMPARE operation is used to compare the high-order part of both fields. This is followed by one or more COMPARE IF EQUAL operations comparing the succeeding lower order parts of the field. At the completion of the series of comparisons, the indicators will describe the relation of the entire accumulator field to the entire memory field. The same technique may also be used to compare fields that are split into several parts.

Programming Example

Two programs for comparing two unsigned fields, M and N, each of 15 decimal digits in 6-bit bytes, are shown below. The first program, using COMPARE IF EQUAL, requires fewer instructions, but in some cases will require more execution time than the second program, which does not use this operation.

First Example, Using Compare if Equal

Name	Instruction	Remarks
START	L(DU, 60, 6), M, O	Load M into B register.
HIGH END	K(DU, 60,6), N, O	Compare M with N.
	L(DU, 30, 6), M .60,0	Load remainder of M into B.
LOW END	KE(DU, 30, 6), N .60,0	Compare M with N.
END	Continuation of Program	

At the completion of the program steps, the comparison result indicators reflect the result of the comparison between the two 15-digit comparands. If after execution of the COMPARE at location HIGH END the AE indicator is off, the COMPARE IF EQUAL at location LOW END is not performed. Notice, however, that the LOAD following the first comparison is executed regardless of the status of the comparison result indicators.

Second Example, without Compare if Equal

Name	Instruction	Remarks
BEGIN	L(DU, 60, 6), M, O	Load M into B register.
HIGH TRY	K(DU, 60, 6), N, O	Compare M with N.
TEST	BZAE, END	If not equal, skip.
	L(DU, 30, 6), M+ .60,0	Load remainder of M into B.
LOW TRY	K(DU, 30, 6), N+ .60,0	Compare M with N.
END	Continuation of Program	

In the above example, the address of the low-order five digits is expressed as M+ .60 rather than M.60. Either method is acceptable. The second method has the advantage of allowing the addressing of locations beyond M. For example, the symbolic address M+1.32 will result in the addressing of the half-word which begins at bit 32 of the location one greater than M. Notice in this example that if the first comparison does render an equal result, the following LOAD and COMPARE instructions are skipped. Thus, even though this example contains a greater number of programs steps it may be executed in less time than that required by the first program.

6.21.20 Compare Field if Equal (KFE)

If the accumulator equal indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is executed exactly as is COMPARE FIELD.

STRAP Notation: KFE (D, 44, 4), NAME .0, 64
Operation Code: 11001

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH): These indicators are set as in COMPARE IF EQUAL.

Programming Note

COMPARE FIELD IF EQUAL can be used with COMPARE FIELD to obtain comparison of fields of more than 64 bits in the same way that COMPARE IF EQUAL is used with COMPARE.

6.21.21 Compare for Range (KR)

If the accumulator high indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is identical to COMPARE with the exception of the setting of the comparison result indicators. If the accumulator operand is equal to or higher than the storage operand, the accumulator high indicator is turned on. If the accumulator operand is less than the storage operand, the accumulator equal indicator is turned on.

STRAP Notation: KR (BU, 64, 8), NAME .0, 64
Operation Code: 01010

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in COMPARE.

Accumulator Low (AL): This indicator cannot be affected by this operation.

Accumulator Equal (AE): This indicator is turned on if the accumulator operand is algebraically less than the memory operand.

Accumulator High (AH): This indicator is turned on if the accumulator operand is algebraically equal to or greater than the storage operand.

The VFL COMPARE FOR RANGE can be used in the same way as the floating point COMPARE FOR RANGE.

6.21.22 Compare Field for Range (KFR)

If the accumulator high indicator is off when this operation is interpreted, the comparison is not performed and the comparison result indicators are not changed. Otherwise, the operation is identical to COMPARE FIELD with the exception of the setting of the comparison result indicators. If the accumulator operand is equal to or higher than the storage operand, the accumulator high indicator is turned on. If the accumulator operand is less than the storage operand, the accumulator equal indicator is turned on.

STRAP Notation: KFR (DU, 24, 4), NAME .0, 0
Operation Code: 11010

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Comparison Result (AL, AE, AH): These indicators are set as in COMPARE FOR RANGE.

Programming Note

COMPARE FIELD FOR RANGE can be used with COMPARE FIELD to determine whether a quantity falls within a given range in the same manner that COMPARE FOR RANGE is used with COMPARE.

6.21.23 Load Transit and Set (LTRS)

The transit register (location 15), the left-zeros counter and the all-ones counter are cleared to zero. If the operation is signed, the sign bit of the storage operand replaces bit 60 of the transit register (TR). Bit positions 60-63 of TR form a sign byte, with positions 61-63 remaining cleared. The remaining 60 bits of the TR (bits 0-59) are loaded with the numeric portion of the storage operand with the low-order bit replacing bit 59 of the TR. The effective offset, specified in the instruction, is placed in the all-ones counter. The two high-order bits of the left-zeros counter, bits 17 and 18 of location 7, are set to ones. At the completion of the operation, the binary transit (BTR) or the decimal transit (DTR) indicator is turned on. This is determined by the status of the radix modifier specified in the instruction.

STRAP Notation: LTRS (D, 64, 4), NAME .4, 64
Operation Code: 11110

Modifiers

Unsigned; Negative: These modifiers operate as in ADD.

Radix: This modifier operates as in ADD. In addition, it determines whether the binary transit (BTR) or decimal transit (DTR) indicator is turned on.

Indicators

Partial Field (PF): Because the low-order four bits of the transit register are used as a sign byte, only 60 numeric bits can be loaded into it. This indicator is turned on if the numeric part of the field loaded contains more than 60 significant bits. If this occurs, the transit register is loaded up to its left end. Higher-order bits are lost. For example, this indicator may be turned on by loading a 64-bit unsigned number where a one bit is contained in any of the four high-order bit positions.

Binary Transit (BTR); Decimal Transit (DTR): One of these two indicators is turned on at the end of the operation, depending upon the value of the radix modifier bit.

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the contents of the transit register when the operation is complete.

Programming Note

The primary purpose of this operation is to serve as an entry to special subroutines. The actual entry may be caused by the interrupt which may result from setting either the BTR or DTR indicators. The effective offset, which has been placed in the all-ones counter, is available to the subroutine in the correct bit address to be directly loaded into an index word and used to provide the desired effective offset.

The two high-order bits of the left zeros count may be used after an interruption caused by the DTR indicator to distinguish this operation from decimal MULTIPLY, DIVIDE, or MULTIPLY AND ADD, which may also turn on this indicator. These two bits are set as follows by the four operations:

Multiply	00
Divide	01
Multiply and Add	10
Load Transit and Set	11

These bits are at the correct bit address to be used directly in indexing to a table of half-word BRANCH instructions.

The offset placed in the all-ones counter may also be used to load an index which in turn may be used for indexing to a table of half-word BRANCHES. Used in this manner, a LTRS may be used to cause the entry to one of 128 different subroutines. Thus, in a given installation, LTRS may be used as pseudo-instructions. For example, a binary LTRS with an offset of 200 may be defined, in a particular installation, as a cube root pseudo-instruction. The execution of this LTRS may then, via the interrupt, cause entry to a subroutine which extracts the cube root of the factor in the transit register. A LTRS with a different offset may cause entry to a different type of subroutine. Thus, once all the desired subroutines have been compiled in the determined format, all programs in a given installation may utilize them by using LTRS, with proper offsets, as pseudo-instructions.

6.21.24 Multiply (*)

If binary is specified, the product of the addressed operand and the accumulator field specified by the offset is placed into the cleared accumulator at offset 20. Neither operand may exceed 48 significant numeric bits. If decimal is specified, this operation has the same action as LOAD TRANSIT AND SET, except that the two high-order bits of the left zeros counter are set to zeros.

STRAP Notation: *(B, 48, 4), NAME .0, 64
Operation Code: 01100

The actual multiplication in this operation is performed by PAU. This restricts the operands to 48 significant bits each. Similarly, the result appears at offset 20 as this is the normal result field (12-107) for a double-precision PAU operation.

Modifiers

Unsigned; Negative Sign: These modifiers operate as in ADD.

Radix: This modifier operates as in ADD. It also determines whether a binary multiplication is performed or whether instead the action is like that of a decimal LOAD TRANSIT AND SET.

Indicators

Partial Field (PF): This indicator is turned on in a binary operation if either operand exceeds 48 significant bits in length. In such a case, only the 48 low-order bits of the oversized operand are used in the operation. If the accumulator operand was oversized, the higher order bits are lost. In a decimal operation, this indicator is set as in LOAD TRANSIT AND SET.

Decimal Transit (DTR): This indicator is turned on if a decimal operation is specified.

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN): In a binary operation, these indicators are set as in ADD. In a decimal operation, they are set as in LOAD TRANSIT AND SET.

Programming Note

A decimal MULTIPLY can be used to produce an automatic entry to a subroutine, where the desired decimal multiplication may be performed using the conversion operations and binary multiplication. Thus, if the proper subroutine is provided to handle interruptions by the decimal transit indicator, the decimal MULTIPLY may be used in a program as though this action were actually implemented in the machine. The subroutine has the two factors available to the accumulator and the transit register and the effective offset available in the all-ones counter.

6.21.25 Load Factor (LFT)

The factor register (FT) is cleared to zeros. For a signed operation, a 4-bit byte, in which the leftmost bit is the sign of the addressed storage operand as modified by the negative sign modifier, is placed in the four low-order bits of FT. The remaining three bits of this sign byte are zero. The numeric part of the addressed operand is positioned to the left of this sign byte in FT.

STRAP Notation: LFT (D, 64, 4), NAME .0, 0
Operation Code: 00101

The accumulator contents are not altered by this operation. The offset has no meaning and is ignored.

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in ADD.

Indicators

Partial Field (PF): Since the four low-order bits of FT are used as a sign byte, only 60 numeric bits can be loaded into FT. This indicator is turned on if the numeric part of the field loaded contains more than 60 significant bits. If this occurs, FT is loaded up to its left end. Higher order bits are lost.

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the contents of the factor register when the operation is complete.

Programming Notes

This operation is necessary to load the multiplicand of a subsequent MULTIPLY AND ADD. This instruction is not used to load either factor of a MULTIPLY.

The factor register is addressable and, therefore, can be addressed as the storage operand of any operation. FT is a 64-bit register with word address 14.

LOAD FACTOR is the only operation that changes the contents of FT as an integral part of its performance. FT is not changed by any other operation unless it is addressed explicitly.

6.21.26 Multiply and Add (*+)

If a binary operation is specified, the multiplicand field in the factor register (FT) is multiplied by the addressed storage operand. Neither multiplicand nor multiplier may exceed 48 significant numeric bits. The product is algebraically added to the accumulator contents as specified by the offset. If a decimal operation is specified, this operation has the same action as LOAD TRANSIT AND SET, except that the two high-order bits of the left zeros counter are set to 10.

STRAP Notation: *+(BU, 48, 8), NAME .0, 64
Operation Code: 11100

Modifiers

Unsigned; Negative Sign: These modifiers operate as in ADD.

Radix: This modifier operates as in ADD. It also determines whether a binary multiplication and addition are performed or whether, instead, the action is like that of a decimal LOAD TRANSIT AND SET.

Indicators

Lost Carry (LC): In a binary operation, this indicator is set as in ADD. Lost carry cannot be turned on if decimal is specified.

Partial Field (PF): This indicator is turned on in a binary operation if either the multiplicand or the multiplier exceeds 48 significant bits in length. In such a case, only the 48 low-order bits of the oversized operands are used in the operation. Partial field is also turned on in a binary operation if the offset is such that significant bits of

the product attempt to add with accumulator positions to the left of the left end of the accumulator. Such high-order bits of the product are lost. In a decimal operation, partial field is set as in LOAD TRANSIT AND SET.

Decimal Transit (DTR): This indicator is turned on if decimal is specified.

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN): In a binary operation the above indicators are set as in ADD. In a decimal operation, they are set as in LOAD TRANSIT AND SET.

Programming Notes

At the completion of the operation, FT still contains the multiplicand and its sign. It is unchanged by the operation. The product that was formed during the operation is not available anywhere. To retain both the product and the cumulative sum, the separate instructions MULTIPLY and ADD TO MEMORY can be used.

LOAD FACTOR should normally precede MULTIPLY AND ADD. However, the loading of FT need not be done in the operation immediately preceding the MULTIPLY AND ADD. If a partial field indication is to be avoided, the loaded factor must be less than 48 significant bits in length.

The offset specified has no effect upon the multiplication. The offset affects the addition of the product to the accumulator contents in the same manner that it affects ADD.

The arithmetic result indicators are set according to the entire contents of the accumulator when the operation is complete, and not according to the product.

6.21.27 Divide (/)

If a binary operation is specified, the entire accumulator contents to the left of the offset are divided by the addressed storage field. Accumulator positions to the right of the offset are ignored. At the completion of the operation, the quotient is placed in the cleared accumulator at the specified offset, and the remainder with sign is placed in the remainder register (RM). Both dividend and divisor are treated as integers (whole numbers) and only the integral part of the quotient is developed. The length of the dividend to the left of the offset cannot exceed 96 significant bits, and the divisor cannot exceed 48 significant numeric bits. The sign of the quotient replaces the sign bit in the accumulator sign byte register. Bits 60-63 of RM form a four-bit remainder sign byte with bit 60, the sign bit, set to the original accumulator sign. Bits 61-63 are set to zero. Bits 0-59 of RM are replaced by the integer remainder with the low-order remainder bit in bit position 59 of the remainder register.

It may be desirable for the programmer to predict the number of quotient bits produced by a DIVIDE. In working with binary quantities, the quotient length may be determined by using the following expression:

(Dividend length +1) - Divisor length = Maximum quotient length

This statement is true only if the bits to the right of the most significant divisor bit determine its length and if the most significant dividend bit is a one. For example:

```
                  1 1 0 0 1
0 0 1 0 1 √1 1 1 1 1 0 1     Dividend length + 1 = 8
          1 0 1              Divisor length      = 3
            1 0 1            Quotient length     = 5
          1 0 1
              0 1 0 1
                1 0 1
```

To assure that the high-order bit of the divisor is a one, the divisor is normalized as though it were a floating point fraction. The normalization is performed in PAU prior to the actual division.

If a decimal operation is specified, this operation has the same action as LOAD TRANSIT AND SET, except that the two high-order bits of the left zeros counter are set to 01.

STRAP Notation: /(B, 52, 4), NAME .12, 20
Operation Code: 01110

As in MULTIPLY, this operation is actually performed by PAU. For this reason the operands are restricted to the fraction length acceptable in floating point operations.

The rightmost bit of the quotient is always found at the offset specified in the instruction. The bits to the right of the offset are zero.

Modifiers

Unsigned; Negative Sign: These modifiers operate as in ADD.

Radix: This modifier operates as in ADD. It also determines whether a binary division is performed or whether, instead, the action is like that of a decimal LOAD TRANSIT AND SET.

Indicators

Partial Field (PF): In a binary DIVIDE, this indicator is turned on if there are more than 96 significant bits in the accumulator to the left of the offset or if the divisor exceeds 48 significant bits. In such a case, only the low-order 96 or 48 bits are used in the operation and higher-order bits are ignored. Such high-order dividend bits in the accumulator are lost in the operation. In a decimal operation this indicator is set as in LOAD TRANSIT AND SET.

Zero Divisor (ZD): This indicator is turned on if the binary divisor is zero. The division is not attempted, and both the accumulator and the remainder register remain unchanged. If the divisor contains over 48 bits, but the low-order 48 bits are zero, both the PF and ZD indicators are turned on.

Decimal Transit (DTR): This indicator is turned on if decimal is specified.

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN): In a binary operation these indicators are set as in ADD. In a decimal operation, they are set as in LOAD TRANSIT AND SET. Note that these indicators describe the condition of the quotient and not the remainder.

6.21.28 Review of VFL Integer Arithmetic Operations

The VFL arithmetic operations are relatively simple. The greatest variable that must be considered is the data involved.

The presence of a sign byte in the storage operand is defined by the unsigned modifier. If the data are unsigned, it is assumed positive, before applying the negative modifier, for the purpose of the arithmetic operation. In STRAP notation, the unsigned modifier is applied by adding the letter U in the data description field. Binary is stated as unsigned with the symbol BU. The negative modifier is stated in the mnemonic for the particular operation. In the add type operation the plus symbol is replaced by a minus symbol. In other operations the mnemonic is merely altered by the addition of the letter N. For example, the negative sign modifier is applied to a LOAD WITH FLAG by the mnemonic LWFN.

All VFL, other than store type, arithmetic operations may specify immediate addressing. When immediate is specified, the address field of the instruction is used as the data. In such cases, the specified field length defines the number of effective address bits to be used, starting at the high-order bit, as the operand. If a field length greater than 24 is specified, low-order zeros are added to the address field to form a field of proper length. Immediate addressing is specified by adding the letter I to the normal mnemonic. For example, an ADD with an immediate operand is stated +I. In MULTIPLY AND ADD, the negative sign modifier as well as the immediate modifier applies to the storage operand used in the multiplication. For this reason, the N and I symbols are inserted between the multiply symbol and the add symbol. Thus a MULTIPLY AND ADD (*+) with a negative immediate multiplier is encoded in STRAP notation as *NI+. In all other applicable operations the N and/or I symbols follow the basic mnemonic.

As compared with floating point operations, the VFL instructions encounter relatively few exceptions. Some of the main points to be remembered when programming the VFL operations are listed below.

1. The entire accumulator is treated as a unit. Thus, positions of the accumulator below the offset may be altered during an operation.

2. When decimal operation is specified, the accumulator is always processed in byte size four. If the specified byte size is other than 4, each byte of the storage operand is adjusted to four bit bytes for the operation. This may necessitate the addition of high-order zeros (byte size less than 4) or the dropping of extra high-order bits (byte size greater than 4). In either case, the specified byte size defines the number of positions between the low-order bits of adjacent storage bytes.

3. In a decimal store-type operation, the high-order bits (in excess of 4) of each storage byte are replaced by the contents of the equivalent zone positions of the sign byte register.

4. The byte size also defines the length of the sign byte. If binary is specified, byte size refers only to the sign byte.

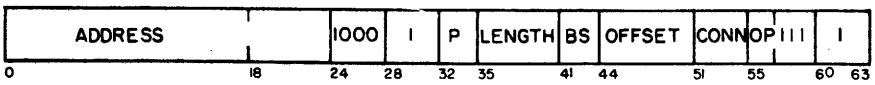
5. The flag bits of the sign byte register are set to zero during a LOAD. In LWF, the flag bits of the sign byte register are replaced by the appropriate flag bits of the storage word. In such cases the actual number of flag bits set is dependent on the specified byte size. The only other method of altering the flag positions of the sign byte register is by addressing them as a storage operand.

6. The only method of altering the zone bits of the sign byte register is by addressing them as a storage operand.

6.22 CONNECTIVE OPERATIONS

Instructions that logically compare bits in an AND, OR, or exclusive OR function are called connective instructions. The 7030 instruction set includes these as well as many other connectives. There are three connective operations, each of which can specify any one of the sixteen binary connectives. One operand of these operations is in the accumulator. The other is in storage or in the instruction itself. The accumulator operand extends from a low-order position specified by the offset and consists of the same number of bytes as are in the storage operand. The storage operand is defined by a bit address and field length as in integer arithmetic. In each of the connective operations, the two operands are combined according to the logical connective specified in the instruction. The result obtained is used to develop certain tests. In CONNECT, the result replaces the accumulator operand; in CONNECT TO MEMORY, the result replaces the storage operand; in CONNECT FOR TEST, the result is discarded after developing the tests.

The connective operations use basically the same full-word instruction format as the integer arithmetic operations. The word and bit address, the two I fields, the P field, the field length, byte size, and offset occupy the same bit positions in the instruction format and have the same function in the connective operations as they do in the integer arithmetic operations.



STRAP Format: OP (dds), A2a (I), OF7(I)

The operation code in bits 55-59 designates which of the three operations is to be performed. The connective which is to be applied is specified in bits 51-54. Notice that there are no radix or sign modifiers in the instruction format. All data used in connective operations are unsigned and binary.

6.22.1 Field Definition

The operand specified explicitly by a connective instruction is defined by the effective address in the same manner as in integer arithmetic. Immediate addressing is also handled in the same way in the connectives as in the arithmetic operations.

The implied second operand of the connective operations is a field from the accumulator. The right end of this operand is defined by the offset specified in the instruction. In contrast to the integer arithmetic operations, the accumulator operand has a definite length; it is composed of as many bytes as there are in the addressed storage field. In CONNECT, only those accumulator bytes which are combined with the storage operand can be affected. The remainder of the accumulator contents remain unchanged.

In CONNECT TO MEMORY, the number of accumulator bytes between the specified offset and the left end of the accumulator can be less than the number of bytes in the storage operand. In this event, the accumulator operand is lengthened for the operation by the addition of high-order zeros.

6.22.2 Byte Size

The connective operations process data in eight-bit bytes. In these operations, the byte size specified in the instruction designates the number of bits used to represent each character or symbol in the storage field. If the byte size of the storage operand is less than eight, each byte is converted into an eight-bit byte before processing by the addition of high-order zeros. The accumulator operand is assumed to have a byte size of eight. Any offset may be specified and the field is processed in eight-bit bytes starting with the offset. Each byte of the storage field, after conversion to eight-bit form is necessary, is combined with an eight-bit byte from the accumulator according to the specified connective. Consequently, the result bytes always have a byte size of eight. In CONNECT, these result bytes replace the accumulator field which participated in the operation. In CONNECT FOR TEST, the result field is discarded after developing certain tests. In CONNECT TO MEMORY, the result bytes replace the storage operand in accordance with the byte size of that operand. The storage field may have any byte size from one to eight. Since each result byte replaces the corresponding byte in the storage field, the result byte is truncated before being placed in storage if the byte size of the storage field is less than eight. Each byte in the storage field is replaced by the number of bits specified by the byte size. These bits are the low-order bits of each result byte; the remaining high-order bits of each result byte are dropped. No indication is given if the bits which are dropped are significant.

Because the field lengths of the storage operand need not be a multiple of the byte size, the last byte need not be of full size. When this is the case, only those bits of the high-order accumulator and storage bytes which are included in the field length take part in the processing. No high-order zeros are added to the short byte from storage. However, if the field length is a multiple of the byte size, and if the byte size is less than eight, high-order zeros are added to the high-order byte as well as to all other bytes so that all bytes of the field are treated in the same manner.

To combine a group of bits in storage with the same number of adjacent bits in the accumulator, a byte size of eight should be specified. This avoids the addition of high-order zeros to each byte of the storage operand.

6.22.3 Counts

Two counts describing the result field are available after a connective operation. The seven-bit all-ones counter, bits 44-50 of location 7, contains a count of all ones in the result field. The seven-bit left-zeros counter, bits 17-23 of location 7, contains a count of the number of zeros in the result field which are to the left of the most

significant one-bit. These counts are positioned so that they can readily be used for indexing. The position of the left zeros counter in the first half-word of location 7 is such that it corresponds to that part of an index value field which indexes the bit address of an instruction. The position of the all-ones counter in the second half-word of location 7 is such that it facilitates indexing of half-word addresses. Each count is on the final result field. In CONNECT and CONNECT FOR TEST, the counts are made on the result field with byte size eight. In CONNECT TO MEMORY, the counts are made on the result field after it is converted to the byte size of the storage operand. The high-order bits which are dropped from each result byte if the byte size is less than eight are ignored in developing the counts. Positions in the accumulator which do not take part in the operation have no effect on the counts. The left-zeros and all-ones counts are destroyed by all LOAD TRANSIT AND SET operations and by decimal MULTIPLY, MULTIPLY AND ADD, and DIVIDE operations. The left-zeros count is also destroyed by all floating point division operations. These counts are not changed as an inherent part of any other non-connective operations. The counts are set at the end of the connective operations, so their previous values may be used as operands in the operation.

6.22.4 Connectives

The sixteen connectives that can be specified are listed below. The storage operand is denoted by "m" and the accumulator operand by "a." The symbols used are defined as: . = AND, V = OR, $\underline{V}$ = exclusive OR, $\overline{a}$ = not a.

	Result Bit for Operand Bit Combinations				Symbolic Representation of Logical Function
	m a	m a	m a	m a	
	0 0	0 1	1 0	1 1	
0.	0	0	0	0	0
1.	0	0	0	1	$m . a$
2.	0	0	1	0	$m . \overline{a}$
3.	0	0	1	1	m
4.	0	1	0	0	$\overline{m} . a$
5.	0	1	0	1	a
6.	0	1	1	0	$m \underline{V} a$
7.	0	1	1	1	$m V a$
8.	1	0	0	0	$\overline{m} . \overline{a}$
9.	1	0	0	1	$m = a$
10.	1	0	1	0	$\overline{a}$
11.	1	0	1	1	$m V \overline{a}$
12.	1	1	0	0	$\overline{m}$
13.	1	1	0	1	$\overline{m} V a$
14.	1	1	1	0	$\overline{m} V \overline{a}$
15.	1	1	1	1	1

The four-bit code used for logical connectives is composed of the result bits obtained for each of the possible combinations of "m" and "a."

In this code, the first bit represents the result when "m" and "a" are both zero; the second bit represents the result when "m" is zero and "a" is one; the third bit

represents the result when "m" is one and "a" is zero; and the fourth bit represents the result when "m" and "a" are both one.

For example, if connective 0101 is specified, the result bit will be one when and only when the "a" operand is a one-bit. If connective 1011 is specified, the result bit is a one for all combinations of "m" and "a," except "m" equal to zero and "a" equal to one.

6.22.5 Connect (C)

The addressed storage operand is combined with the accumulator field specified by the offset in accordance with the logical connective specified. The result field replaces the accumulator operand. The remainder of the accumulator remains unchanged.

STRAP Notation: C0000 (BU, 24, 8), ANY .0, 104
Operation Code (55-56): 00

Indicators

Partial Field (PF): This indicator is turned on if the storage field attempts to combine with bits beyond the left end of the accumulator. If a partial field condition occurs, the storage operand is combined with the accumulator operand up to the left end of the accumulator; higher order bits in the storage operand are dropped. In such a case, the counts are developed and the result indicators are set according to that portion of the result actually developed.

To-Memory Operation (MOP): This indicator is turned off.

Result Zero (RZ): This indicator is turned on if the final result field is all binary zeros. This implies that the all-ones count is also zero and that the left-zeros count is equal to the number of bits in the final result field. This number is equal to the number of accumulator bits which take part in the operation.

Result Greater than Zero (RGZ): This indicator is turned on if the result field contains any one bit.

Result Less than Zero (RLZ); Result Negative (RN): These indicators are turned off by the connective operations.

6.22.6 Connect to Memory (CM)

The accumulator field specified by the offset is combined with the addressed storage operand in accordance with the logical connective specified. The result replaces the storage operand. The accumulator contents are unchanged by the operation.

STRAP Notation: CM 0000 (BU, 48, 6), 0.0,0
Operation Code: 01

Indicators

Partial Field (PF): This indicator is not affected by this operation.

To-Memory Operation (MOP): This indicator is set to one.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set to describe the final result as it appears in storage. Any bits which are dropped because of a byte size less than eight do not affect these indicators.

6.22.7 Connect for Test (CT)

The addressed storage operand is combined with the accumulator field specified by the offset in accordance with the logical connective specified as in CONNECT. The result is discarded after the left-zeros and all-ones counts are developed and the indicators set. Both the accumulator contents and the storage operand remain unchanged.

STRAP Notation: CT 0000 (BU, 48, 8), NAME .12, 64
Operation Code: 10

Indicators

Partial Field (PF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in CONNECT.

Programming Notes

CONNECT has many uses other than that of evaluating logical conditions. The connectives 0000 and 1111 can be used to set the specified accumulator or storage bits to zero or one. A section of the accumulator can be replaced by connective 0011; while connective 0101 has no effect except to test the accumulator field and develop the counts. Connective 1010 inverts the accumlator operand.

Testing of storage fields for all zeros is readily performed by CONNECT FOR TEST 0011, while testing for all ones requires connective 1100. In each case, the result zero indicator will be set to one if the test is satisfied.

To test certain combinations of bits in the accumulator for zero or one, a mask must be furnished in an immediate address or in storage. If the mask uses 1 for bits to be tested and 0 for bits to be ignored, the operation for testing the masked bits for all zeros is CONNECT FOR TEST 0001. If a test for ones is desired, the connective 0010 should be used. In either case, satisfaction of the test is indicated by the result zero indicator. If the field to be tested is in storage, the mask must be in the accumulator and connectives 0001 and 0100 must be used.

In to-accumulator integer arithmetic operations, the result indicators are set according to the entire contents of the accumulator when the operation is complete. This is not the case in the connective operations. In CONNECT and CONNECT FOR TEST, these indicators are not affected by accumulator bits which do not participate in the operation. For example, if a zero result field is obtained when the storage operand is combined with the accumulator operand, the result zero indicator is turned on even if there are non-zero bits in the accumulator to the right or left of the accumulator field which was combined with the storage operand.

It is possible for all 128 bits of the accumulator to participate in a single connective operation if the specified offset is zero and the byte size and field length are such that the storage operand equals or exceeds 128 bits when converted to byte size eight. In this event, a zero all-ones count can indicate either no ones or 128 ones in the result of a CONNECT or CONNECT FOR TEST because the counter, after stepping past 127, returns to zero. The two cases may be distinguished by interrogation of the result zero indicator. Similarly, a zero left-zeros count can indicate either no left zeros or 128 left zeros in the result. These two cases can also be distinguished by interrogation of the result zero indicator.

The data flag and lost carry indicators are not affected by the connective operations.

In STRAP notation, the immediate mode of addressing is specified by adding the letter I to the normal mnemonic: CI, CTI. Immediate addressing is not possible for CONNECT TO MEMORY.

Note that C0011 replaces the specified portion of the accumulator with the storage field. The remainder of the accumulator remains unchanged. Because no other basic instruction can accomplish this purpose it is expected that C0011 will be widely used. For this reason, a STRAP mnemonic has been provided which results in the assembling of CONNECT 3. The mnemonic is LF (load field). Also a mnemonic has been provided to facilitate the coding of CM0101 which places the accumulator field in the specified storage field. This mnemonic is SF (store field). Thus, LF = C0011, SF = CM0101.

Programming Example

Given a 128-bit quantity in the accumulator, shift the entire quantity left ten bits. Discard the high-order bits that are shifted out of the accumulator and shift zeros into the low-order positions. This example illustrates the use of the connective operations for purposes other than the evaluation of logical conditions.

Name	Instruction	Remarks
SHIFT 1	C0011 (BU, 54, 8), 8.10, 74	Replace accumulator 0-53 with contents of 10-63.
SHIFT 2	LF(BU, 64, 8), 9.0, 10	Replace accumulator 54-117 with contents of 64-127.
ZERO	CI0000 (BU, 10, 8), 0,0	Set 118-127 to zero.

6.23 RADIX CONVERSION OPERATIONS

There are four radix conversion operations. Any one of these may convert an integer either from binary to decimal or from decimal to binary. Two of these operations, LOAD CONVERTED and LOAD TRANSIT CONVERTED, obtain a field from storage, convert it, and place the result in the accumulator or the transit register. The other two operations, CONVERT and CONVERT DOUBLE, take a field from the accumulator, convert it, and return it to the accumulator. These latter two operations differ from each other in the length and position of the binary field involved.

The radix conversion operations used the same full-word instruction format as the integer arithmetic operations. In this case, bit 53, the radix modifier, describes the format of the original field and also determines the type of conversion, binary to decimal or decimal to binary.

The storage operand in LOAD CONVERTED and LOAD TRANSIT CONVERTED is defined by the effective address, field length, and byte size in the same manner as in integer arithmetic. Immediate addressing is also handled in the same way in these radix conversion operations as in the arithmetic operations.

In the accumulator conversion operations, CONVERT and CONVERT DOUBLE, no storage operand is required, and the word address, bit address, length, and byte size fields of the instruction are not normally used. If the conversion is from binary to decimal, the binary field is taken from a fixed position in the accumulator, and the decimal result is put back at a position specified by the offset. If the conversion is from decimal to binary, the decimal field is taken from the position in the accumulator specified by the offset, and the binary result is put back at a fixed position.

The radix conversion operations include two sign modifier bits which operate in the same manner as in the integer arithmetic operations. In all cases these modifiers apply to the original field before conversion and specify how its sign is to be modified during the conversion.

Bit 53 of the instruction, the radix modifier, specifies the radix of the operand to be converted, and whether binary to decimal or decimal to binary conversion is to be performed. If this bit is zero, the original field is binary; if this bit is one, the original field is decimal. This modifier also specifies the interpretation of the byte size field in the same manner as in integer arithmetic operations. Byte size four is assumed in the accumulator conversion operation.

A number of indicators in the indicator register are set as a result of radix conversion operations. These include the partial field indicator and the data flag and arithmetic result indicators. The partial field indicator is permanent. If it is turned on by any operation it will remain on until it either causes an interruption or is explicitly set to zero. The data flag and arithmetic result indicators are temporary. Those affected by a given operation are set at the end of the operation to reflect the result of that operation. They then remain as set until the execution of another operation that affects them. The indicators that are affected by the radix conversion operations are described with each operation.

The four radix conversion operations are shown in the table below, which indicates the source of the operand and the register in which the result is placed.

		Operand Taken from	Result Placed in
LCV	Load Converted	Storage	Accumulator
LTRCV	Load Transit Converted	Storage	Transit Register
CV	Convert	Accumulator	Accumulator
DCV	Convert Double	Accumulator	Accumulator

6.23.1 Load Converted (LCV)

The addressed storage field is converted as an integer from decimal to binary or from binary to decimal depending on the setting of the radix modifier. The accumulator is cleared and the converted field is placed into it at the specified offset. The four low-order bits of the sign byte register are cleared. The sign bit in the sign byte register is set according to the sign of the data as affected by the sign modifiers.

STRAP Notation: LCV (D, 64, 4), Decimal .0, 20
Operation Code: (54-58): 00001

Modifiers

Unsigned; Negative Sign: These modifiers operate as in ADD.

Radix: If decimal is specified, the storage operand is converted from decimal to binary after conversion of the decimal field to byte size four. If binary is specified, the storage operand is converted from binary to decimal with the decimal result in byte size four.

Indicators

Partial Field (PF): This indicator is actuated when significant bits of the converted field extend beyond the left end of the accumulator. In such a case, the accumulator is filled up to its left end, and higher order bits are lost. In all the conversion operations (except CONVERT DOUBLE) the operand (after being adjusted to byte size four) must contain no more than sixty significant bits excluding the sign byte. If more than sixty bits are contained in the operand, with byte size four, not all of the bits will participate in the conversion and meaningless results will be obtained. This condition also turns on indicator PF.

Data Flags (TF, UF, VF); To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in ADD.

6.23.2 Load Transit Converted (LTRCV)

The addressed storage field is converted as an integer from decimal to binary or from binary to decimal depending on the setting of the radix modifier. The transit register, storage location 15, is cleared. The four low-order bits of the transit register form a sign byte in which the leftmost bit is the sign of the addressed storage operand as modified by the negative sign modifier. The remaining three bits of the sign byte are zero. The converted field is placed in the transit register positioned to the left of the four-bit sign byte. The accumulator is not altered by this operation, and the offset field of the instruction is ignored.

STRAP Notation: LTRCV (D, 64, 4), Decimal .0,0
Operation Code: 10001

Modifiers

Unsigned; Negative Sign; Radix: These modifiers operate as in LOAD CONVERTED.

Indicators

Partial Field (PF): This indicator is actuated if the result in a binary-to-decimal conversion exceeds 15 significant decimal digits or if the result of a decimal-to-binary conversion exceeds 48 significant bits. In the former case the result is placed in the transit register up to its left end. In the latter case, only the 48 low-order bits of the result are placed in the transit register. This indicator is also actuated if the storage field specified in a decimal-to-binary conversion exceeds 60 bits when adjusted to byte size four. Meaningless results are then obtained.

Data Flags (TF, UF, VF); To-Memory Operation (MOP): These indicators are set as in ADD.

Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set according to the final result in the transit register.

6.23.3 Convert (CV)

A field from the accumulator is converted as an integer from binary to decimal or from decimal to binary depending on the setting of the radix modifier. If the original field is binary, it is obtained at an implied offset of 68 and its length is 48 numeric bits. The decimal result is loaded into the cleared accumulator at the specified offset. If the original field is decimal, it is obtained at the specified offset. The binary result, which can be a maximum of 48 significant bits, is loaded into the cleared accumulator at offset 68. The sign modifiers operate on the sign bit in the sign byte register; all other positions of the sign byte register remain unchanged.

STRAP Notation: CV(D, 0,0), 0.0, 64
Operation Code: 01101

Notice that a binary-to-decimal operation obtains the original field from that portion of the accumulator that would normally contain a single precision floating point fraction. Similarly, the result of a decimal-to-binary operation is placed at offset 68.

Modifiers

Unsigned: If the unsigned modifier is zero, the accumulator sign is taken as-is before the negative sign modifier is applied. If the modifier is one, the accumulator sign is taken as positive before the negative sign modifier is applied.

Negative Sign: If the negative sign modifier is zero, the accumulator sign is taken as-is after modification by the unsigned modifier. If the negative sign modifier is one, the accumulator sign after modification by the unsigned modifier is inverted.

Radix: If binary is specified, the 48-bit accumulator field at offset 68 is converted from binary to decimal. Higher-order bits (positions 0-11 of the accumulator) are ignored and lost. No indication of this loss is retained. The decimal result in byte size four is placed into the accumulator at the specified offset. If decimal is specified, the accumulator field at the specified offset is converted from decimal to binary. The binary result is placed into the accumulator at offset 68. The original decimal field is assumed to be byte size four.

Indicators

Partial Field (PF): This indicator is actuated in a decimal-to-binary conversion if the binary result exceeds 48 significant bits. In such a case, the low-order 48 bits of the result are placed in the accumulator. Any higher-order bits are lost. The indicator is actuated in a binary-to-decimal conversion if significant bits of the decimal result extend beyond the left end of the accumulator when this result is placed into the accumulator at the specified offset.

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in ADD.

6.23.4 Convert Double (DCV)

A field from the accumulator is converted as an integer from binary to decimal or from decimal to binary depending on the setting of the radix modifier. If the original field is binary, it is obtained at an implied offset of 20 and its length is 96 numeric bits. However, no more than 80 significant bits can be converted without causing a partial field condition. The decimal result, which can be a maximum of 24 significant digits (96 bits), is loaded into the cleared accumulator at the specified offset. If the original field is decimal, it is obtained at the specified offset. The binary result, which can be a maximum of 96 significant bits, is loaded into the cleared accumulator at offset 20. The sign modifiers operate on the sign bit in the sign byte register; all other positions of the sign byte register remain unchanged.

STRAP Notation: DCV (B, 0,0), 0.0, 32
Operation Code: 11101

Notice that CONVERT DOUBLE either obtains a binary operand from, or places a binary result in, the accumulator in the normal field of a double precision floating point fraction.

Modifiers

Unsigned; Negative Sign: These modifiers operate as in CONVERT.

Radix: If binary is specified, the 96-bit accumulator field at offset 20 is converted from binary to decimal. Higher-order bits (positions 0-11 of the accumulator) are ignored and lost. No indication of this loss is retained. The decimal result in byte size four is placed into the accumulator at the specified offset. If decimal is specified, the accumulator field at the specified offset is converted from decimal to binary. The binary result is placed into the cleared accumulator at offset 20. The original decimal field is assumed to be byte size four.

Indicators

Partial Field (PF): This indicator is actuated if the converted field exceeds 96 significant bits are in either type of conversion. The 96 low-order bits of the result are obtained, and higher-order bits are lost. The indicator is also actuated in a binary-to-decimal conversion if significant bits of the decimal result extend beyond the left end of the accumulator when this field is loaded into the accumulator at the specified offset. Because no more than 24 decimal digits (byte size 4) may be contained in 96 bits, the decimal result must be 24 digits or less. Eighty binary one bits when converted to decimal require 25 decimal digits. Therefore, the maximum length field that can be converted from binary to decimal is 80 bits.

To-Memory Operation (MOP); Arithmetic Result (RLZ, RZ, RGZ, RN): These indicators are set as in ADD.

Programming Notes

The effective address of the first half word, the byte size field, and the length field are not normally used by either CONVERT or CONVERT DOUBLE and are ignored. If progressive indexing is specified, the specified index register will be modified as specified by the progressing indexing code in the normal manner.

The definition of the sign modifiers permits complete control over the accumulator sign. It may be left as-is, inverted, made positive or made negative.

The fixed offsets used in the accumulator conversion operations are designed to facilitate conversion between binary floating point and decimal formats. Thus, the 48-bit binary field at offset 68 used in CONVERT corresponds to the fraction field of a single-precision floating point number, and the 96-bit binary field at offset 20 used in CONVERT DOUBLE corresponds to the fraction field of a double-precision floating point number. This 96-bit field also corresponds to the result field of a binary integer MULTIPLY.

Programming Example

The problem of multiplying two decimal integers illustrates the use of the conversion operations. Given two four-digit signed decimal integers, with byte size 4, in locations 1900 and 1900.20, form the eight-digit product and store it at location 2000.

Name	Instruction	Remarks
Radix 1	LCV (D, 20,4), 1900.0,0	Convert multiplicand
Radix 2	LTRCV (D, 20,4),1900.0,0	Convert multiplier
Rate	*(B,64,4), 15.0,0	Multiply accumulator by the transit register
Radix 3	DCV(B,0,0), 0.0,0	Convert product to decimal
Result	ST(D,36,4), 2000.0,0	Store product in 2000.0

At location rate, a byte size of 4 is used because the transit register contains a four-bit sign byte. It is safe to use a field length greater than 48 in the multiply instruction as there are only 16 significant bits in the transit register. At radix 3, DCV is used rather than CV because the product appears at offset 20.

IBM®
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York

Printed in U.S.A. R23-9694