IBM Field Engineering

Manual of Instruction

226-2037-0

2065 Processing Unit, Volume 2

IBM®

Field Engineering

Manual of Instruction

2065 Processing Unit, Volume 2

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

SECTION 3. FLOATING-POINT INSTRUCTIONS

This section describes data and instruction formats, number representation, normalization, data flow paths, condition codes, and interruptions as they relate to floating-point instructions. The section then analyzes the floating-point instruction set.

3.5 INTRODUCTION

- Long and short operands for all instructions.
- Load, add, subtract, multiply, divide, compare, halve, store, and sign control instructions.
- Results always in true form.
- Signs determined algebraically.
- Four floating-point registers.
- RR and RX formats.

The floating-point instruction set performs calculations using operands with a wide range of magnitude and yielding results to preserve precision. Instructions provide for loading, adding, subtracting, comparing, halving, multiplying, dividing, and storing as well as sign control of long and short operands.

The operands and arithmetic results are always in true form. A plus sign indicates a positive number; a negative sign, a negative number. Intermediate results are changed to true form, if necessary, before the final result is stored in the first operand location. For the add, subtract, multiply, and divide instructions, the result signs are determined algebraically.

A floating-point number consists of a sign, a signed exponent (characteristic), and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess 64 binary notation. The fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit. For a description of numbering systems, refer to paragraph 1.4 of Chapter 1.

To preserve maximum precision, the results of addition, subtraction, multiplication, and division are normalized. Unnormalized addition and subtraction may also be performed. Normalized and unnormalized operands may be used in all floating-point operations.

The condition code (CC) is set as a result of all sign control, add, subtract, and compare instructions.

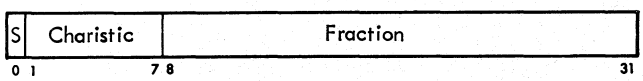
Four 64-bit floating-point registers in LS are reserved exclusively for floating-point instructions. Floating-point instructions occur in the RR and RX formats.

3.5.1 DATA FORMAT

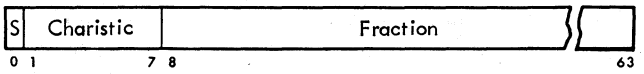
- Programmer must address LS floating-point register 0, 2, 4, or 6.
- Data format consists of 1-bit sign, 7-bit characteristic, and 24- or 56-bit fraction.
- Results are 32 bits (short operand) or 64 bits (long operand) in length.
- Multiply product is always 64 bits.
- Guard digit is retained in short operand instructions only.

Floating-point data occupies a fixed-length format which may be either a full-word short format or a double-word long format. Both formats may be used in main storage and in the LS floating-point registers. The four floating-point register addresses are 0, 2, 4, and 6. (See paragraph 3.5.5 for floating-point addressing.) The data formats for short and long operands are:

Short Operand



Long Operand



For both formats, the first bit position is the sign bit and the subsequent seven bit positions constitute the characteristic. The following 24 or 56 bits, the fraction, contain 6 or 14 hexadecimal digits for the short and long operands, respectively.

When short operands are specified, the results are 32-bit floating-point words; the rightmost 32 bits of the floating-point LS register do not participate in the operation and remain unchanged. An exception occurs in multiply instructions, where the product occupies a full floating-point register (64 bits).

When long operands are specified, all operands and results are 64-bit floating-point double words.

Although short operand results have six hexadecimal fractions digits, intermediate results in addition, subtraction, and compare operations may extend to seven fraction digits. The low-order digit of a 7-digit fraction, called the guard digit, increases the accuracy of the final result. Intermediate results in long operands do not exceed 14 hexadecimal fraction digits. No guard digit is used for long operand instructions.

3.5.2 NUMBER REPRESENTATION

- Radix point is to left of high-order hexadecimal digit.
- Number representation is ± fraction X $16^{n \text{ power}}$.
- Exponent range is -64 to +63.
- Excess 64 exponent range is 0 to 127 with 64 as midpoint.
- True zero result yields positive sign.

The fraction of a floating-point number is expressed in hexadecimal digits. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for a floating-point number, the fraction is considered to be multiplied by a power of 16 (fraction X $16^{n \text{ power}}$). The characteristic (bits 1-7) of both formats indicates the power (exponent).

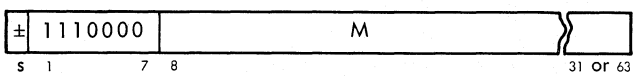
Since the fraction and the exponent are both signed numbers, some method must be employed to express the two signs in the data formats that provide for a single sign. This is accomplished by having the fraction sign use the sign (bit 0) associated with the word (or double word) and expressing the exponent in excess 64 arithmetic; that is, the original exponent value is added as a signed number to 64. The resulting number is called the characteristic. Since 64 is expressed by seven binary bits, the characteristic will vary from 0 to 127, permitting the exponent to vary from -64 through 0 to +63.

Because the CPU treats the characteristic in excess 64 notation, 64 must be algebraically added to the original hexadecimal exponent and expressed in true form. Adding 64 to the original exponent yields a range of 0 to 127 (decimal), which can be expressed by using 7 binary bits. This range (0-127) corresponds to the -64 to +63 range. To obtain the true characteristic value, 64 must be algebraically subtracted from the value in the characteristic field. The binary range and equivalent values are shown in Table 3-11.

TABLE 3-11. CHARACTERISTIC NOTATION

Excess 64 Notation		
Binary	Decimal	Exponent
0000000	0	-64
0000001	1	-63
↓	↓	↓
0111111	63	-1
1000000	64	0
1000001	65	+1
↓	↓	↓
1111110	126	62
1111111	127	63

For example, the value of ±M X 16^{48} must be stated in excess 64 notation. The characteristic of the fraction then becomes 48 + 64 = 112 (0110000 + 1000000 = 1110000). The floating-point number thus takes the form shown in the data format below:



Both positive and negative quantities are in true form, with the difference indicated by the sign (0 indicating plus, 1 indicating minus).

A number with a zero characteristic, a zero fraction, and a plus sign is called a true zero. A true zero may occur as the result of an arithmetic operation because of the magnitude of the operands. A true zero is forced when exponent underflow occurs during add, subtract, multiply, and divide instructions. A true zero is also forced when a result fraction is zero and no program interruption due to lost significance occurs during add, subtract, multiply, and divide instructions. A zero result fraction will not cause a true zero result to be forced for load, store, and halve instructions. When a lost-significance interruption is indicated, the true zero is not forced and the result sign is stored with the result characteristic and the zero fraction. Whenever a result has a zero fraction, an exponent overflow or exponent underflow interruption condition is ignored. If the divisor fraction equals zero, division is omitted and a floating-point-divide interruption condition exists; a program interruption therefore occurs. If the divisor fraction does not equal zero, zero fractions and zero characteristics participate as normal numbers in all arithmetic operations.

The sign of a sum, difference, product, or quotient with a zero fraction is positive. The sign of a zero fraction resulting from other operations is established algebraically from the operand signs.

3.5.3 NORMALIZATION

- Fraction is normalized when high-order hexadecimal digit is not zero.
- Characteristic is adjusted on normalization cycles.
- Postnormalization is normalization of final result.
- Prenormalization is normalization prior to result computation.
- Results are truncated when necessary.

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. The process of normalization consists of shifting the fraction left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called "postnormalization." In performing multiplication and division, the operands are normalized prior to the arithmetic process. This function is called "prenormalization."

Floating-point operations may be performed with or without normalization. Most operations are performed in only one of these two ways. Addition and subtraction may be specified either way.

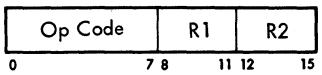
When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. The result may or may not be normalized, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. Also, intermediate fraction results are shifted right whenever a fraction carry from parallel adder bit position 8 occurs (fraction overflow), and the intermediate fraction result is truncated to the final result length after the shifting.

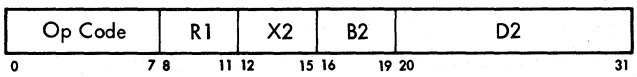
Since normalization applies to hexadecimal digits, the three high-order bits of a normalized number may be zero.

3.5.4 INSTRUCTION FORMATS

- RR format



- RX format



Floating-point instructions occur in the RR and RX formats. In these formats, R1 is the address of a floating-point LS register that contains the first operand. The second operand location is defined differently for the two formats.

In the RR format, R2 is the address of a floating-point LS register containing the second operand. The same register may be specified for the first and second operands.

In the RX format, the contents of the general-purpose LS registers specified by X2 and B2 are added to the contents of the D2 field to form an address designating the main storage location of the second operand. A zero in an X2 or B2 field indicates the absence of the corresponding address component.

The R1 and R2 fields should each specify 0, 2, 4, or 6 as the floating-point LS register address. Otherwise, a specification interruption occurs.

The main storage address of the second operand should designate word boundaries for short operands and double-word boundaries for long operands. Otherwise, a specification interruption occurs.

Results replace the first operand except for store operations, where the result replaces the second operand.

Except for the storing of the final result, the contents of all floating-point or general-purpose LS registers and main storage locations participating in operand addressing or operation execution remain unchanged.

The floating-point instructions are the only instructions that use the floating-point registers.

3.5.5 DATA FLOW

- Eight 32-bit LS registers are reserved for floating-point instructions.
- Micro-orders control low-order fraction fetch.
- LS floating-point register address specified must not be odd or greater than 7.
- Sign-handling is achieved via serial adder or STAT's.
- Characteristic-handling is performed via serial adder.
- Fraction-handling is performed via parallel adder.

Eight 32-bit LS registers (addresses 16-23) are reserved for floating-point instruction operands and results. These registers function as four double-length (64-bit) registers with assigned addresses of 0, 2, 4, and 6. Register 0 is contained in LS locations 16 and 17; register 2, in locations 18 and 19; register 4, in locations 20 and 21; register 6, in locations 22 and 23. For additional information on local storage, refer to Section 6 of Chapter 2.

Addressing of LS is limited to 16 general-purpose registers because the R1 and R2 fields contain four bits each. The LS address register (LAR) contains five bits. In floating-point instructions, an LS address of 0, 2, 4, or 6 must be specified in the R1 and R2 fields of the instruction word.

Because floating-point instructions use only the floating-point registers in LS, a 1 is forced into the zero position of LAR. If address 0 is specified in the R1 or R2 field, LS accesses address 16. Short operand instructions fetch only 32 bits (single word) from the specified floating-point LS register. Because ingating and outgating of LS are limited to 32 bits each, floating-point operands must be divided into two 32-bit words. A 1 is forced in the low-order bit position of LAR [LAR(4)] to fetch or store the low-order 32 bits of a long operand. Micro-orders are required to fetch the low-order fraction during instruction execution. The R1 + 1 and R2 + 1 registers are considered to be the odd-numbered addresses of floating-point LS registers.

At the beginning of the execution phase of floating-point instructions, a specification check establishes that:

1. An even address is specified in the R1 and R2 fields.
2. An address greater than 7 is not specified in the R1 and R2 fields.
3. The effective main storage address is on a full-word boundary for long operands and on a word boundary for short operands.

The specification check is made by testing E(11) and E(15) (RR format) to determine an even-odd address [E(11) and E(15) must equal zero]. If E(8) or E(12) is a 1, the specified LS address is greater than 7. In the RX format, the effective address is in D. D(21-23) must equal zero on long operand instructions, and D(22,23) must equal zero on short

operand instructions. Note that the effective address check is performed in the PAL's. If a specification check exists, operation is suppressed and a program interruption occurs.

At the beginning of the execution phase (RR format), the first operand is located in A, B, and D. The second operand is located in S and T. In the RX format, the first operand is located in S and T. The effective address of the second operand is in D. If a long operand instruction in the RR format is to be executed, the low-order fractions must be fetched from LS during the execute phase.

Data flow may be divided into two paths: (1) the fraction path and (2) the sign and characteristic path. See Figure 9054, FEDM. The fractions are transferred, added, or shifted via the parallel adder. The operands are located in DT, ST, and AB. The parallel adder is capable of shifting 0, R4, or L4 bit positions. Data paths exist between the PAL's and T, D, A, and B.

The sign and characteristic path is from ST or AB to F via the serial adder. The byte gated to the inputs of the serial adder depends upon the STC and the ABC values.

In many floating-point instructions, the signs are saved in STAT's. The sign of the first operand is stored in STAT F; the sign or the complement of the sign of the second operand is stored in STAT C.

The F-register is an 8-bit register. The data from F is gated to the serial adder. From the serial adder, the data is transferred to ST per the STC. The serial adder is capable of adding 1 to, and subtracting 1 or 64 from, the value at the inputs of the serial adder.

3.5.6 CONDITION CODES

- Result equals zero: CC = 0.
- Result less than zero: CC = 1.
- Result greater than zero: CC = 2.
- Exponent overflow: CC = 3.

The results of floating-point sign-control, add, subtract, and compare operations set the CC. Multiplication, division, loading, and storing leave the CC unchanged. The CC can be used for decision-making by subsequent branch-on-condition instructions.

The CC can be set to reflect two types of results for floating-point arithmetic. For most operations, CC's of 0, 1, and 2 respectively indicate that the contents of the result register are zero, less than zero, and greater than zero. A zero result is indicated whenever the result fraction is zero, including a forced zero. A CC of 3 is used when the exponent of the result overflows.

For compare instructions, CC's of 0, 1, and 2 respectively indicate that the first operand is equal, low, and high. The instructions and CC settings are shown in Table 3-12.

TABLE 3-12. CONDITION CODES FOR FLOATING-POINT INSTRUCTIONS

Instruction	Condition Code			
	0	1	2	3
Add Normalized S/L	Zero*	Less than zero	Greater than zero	Overflow
Add Unnormalized S/L	Zero	Less than zero	Greater than zero	Overflow
Compare S/L	Equal	Low	High	-
Load and Test S/L	Zero	Less than zero	Greater than zero	-
Load Complement S/L	Zero	Less than zero	Greater than zero	-
Load Negative S/L	Zero	Less than zero	-	-
Load Positive S/L	Zero	-	Greater than zero	-
Subtract Normalized S/L	Zero*	Less than zero	Greater than zero	Overflow
Subtract Unnormalized S/L	Zero	Less than zero	Greater than zero	Overflow

S = short operands

L = long operands

* An exponent underflow causes a true zero to be stored and the CC is set to zero. An interruption occurs if PSW(38) equals 1.

The CC is set at end-op time of the instruction by a micro-order. Setting the CC register depends upon hardware conditions at the time the set-condition-register micro-order is given. The hardware

conditions for all floating-point instructions that cause a CC to be set in the CC register are shown in Figure 9055, FEDM. For discussion purposes, assume that a Load and Test instruction is to be executed. The Load and Test instruction places the second operand in the first operand location and sets the CC. During instruction execution, the sign of the second operand is stored in STAT C. A zero test of the fraction is made, and STAT A is set if the fraction equals zero. If STAT C is set and the fraction is not zero, the result sign is minus. Therefore, the result is less than zero and a CC of 1 is set in the CC register. If the sign is plus, a CC of 2 is set in the CC register. If the fraction equals zero, STAT A is set and blocks setting the CC register, thus yielding a CC of 0. In add-type instructions, the CC is determined by STAT C, STAT F, STAT A, and A(7) values.

3.5.7 FLOATING-POINT INTERRUPTIONS

- Floating-point interruptions and action taken:
  1. Protection — Operation suppressed
  2. Addressing — Operation terminated
  3. Specification — Operation suppressed
  4. Exponent overflow
  5. Exponent underflow — True zero stored
  6. Significance — Sign and characteristic of result zero fraction stored
  7. Floating-point divide — Operation suppressed

Certain abnormal conditions may exist that require special attention by the program. When special processing is required, an interruption request is generated according to the conditions that exist. The interruption request is honored, if not masked off, and a program interruption occurs. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following program interruptions occur in floating-point instructions:

1. Protection: The storage key does not match the protection key in the PSW for all RX instructions. When an instruction causes a fetch-protection violation, execution of the instruction is terminated, the program execution is altered by a program interruption, and a protection interruption is indicated in the old PSW. When an instruction causes a store-protection violation, the operation is suppressed.

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

3. Specification: A short operand is not located on a 32-bit boundary, or a long operand is not located on a 64-bit boundary, or a floating-point register address other than 0, 2, 4, or 6 is specified. The instruction is suppressed. The address restrictions do not apply to the components (contents of the D2 field and the contents of the LS registers specified by X2 and B2) from which an address is generated.

4. Exponent Overflow: The result exponent of an addition, subtraction, multiplication, or division overflows, and the result fraction is not zero. The operation is terminated. The CC is set to 3 for addition and subtraction and remains unchanged for multiplication and division. An unconditional program interruption occurs.

5. Exponent Underflow: The result of an addition, subtraction, multiplication, or division underflows, and the result fraction is not zero. A program interruption occurs if the exponent-underflow mask bit is 1. The operation is completed by replacing the result with a true zero. The CC is set to 0 in addition and subtraction and remains unchanged for multiplication and division. The state of the mask bit does not affect the result.

6. Significance: The result fraction of an addition or subtraction is zero. A program interruption occurs if the significance mask bit is 1. The mask bit also affects the result of the operation. When the significance mask bit is a 0, the operation is completed by replacing the result with a true zero. When the significance mask bit is 1, the operation is completed without further change to the characteristic of the result. In either case, the CC is set to 0.

7. Floating-Point Divide: Division by a number with zero fraction is attempted. The division is suppressed, but the CC and the data in storage remain unchanged.

3.5.8 INITIAL CONDITIONS

- RR, short operands:
  1. 1st operand is in A, B, and D (24-bit fraction only).
  2. 2nd operand is in S and T.
  3. Instruction is in E.

- RX, short operands:
  1. 1st operand is in S and T.

2. Effective address of 2nd operand is in D.
3. Instruction is in E.

- RR, long operands:
  1. 32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  2. 32 bits of 2nd operand are in S and T.
  3. Instruction is in E.
  4. Low-order fractions of 1st and 2nd operands are in LS.

- RX, long operands:
  1. 32 bits of 1st operand are in S and T.
  2. Low-order fraction of 1st operand is in LS.
  3. Effective address of 2nd operand is in D.
  4. 2nd operand is in main storage.
  5. 1st 16 bits of instruction are in E.

During an RR I-Fetch (short operands), the sign, characteristic, and 24-bit fraction of the first operand specified by R1 are placed in A and B, and the 24-bit fraction is placed in D. The sign, characteristic, and 24-bit fraction of the second operand are placed in S and T. The address of the second operand is specified by R2. The instruction is contained in E.

The objectives of the RX I-Fetch (short operands) are to compute the effective address of the second operand and place this address in D. During I-Fetch, the sign, characteristic, and 24-bit fraction of the first operand are placed in S and T. The contents of A and B are unknown. The R1 and X2 fields of the instruction are in E, and the B2 and D2 fields are in Q. The contents of the LS register specified by X2 are added to the contents of the LS register specified by B2, and this total is added to the D2 field to make up the effective address in main storage of the second operand.

A specification check is made at the beginning of the execution phase. If an error exists, the operation is suppressed and an interruption occurs.

If a long operand in the RX format is to be executed, the low-order fraction of the first operand is fetched from LS and the entire second operand is fetched from main storage during the execution phase. For RR instructions, the low-order fractions are fetched from LS during the execution phase.

3.6 INSTRUCTION ANALYSIS

- Floating-point instruction set consists of 44 instructions.
- Table 3-13.

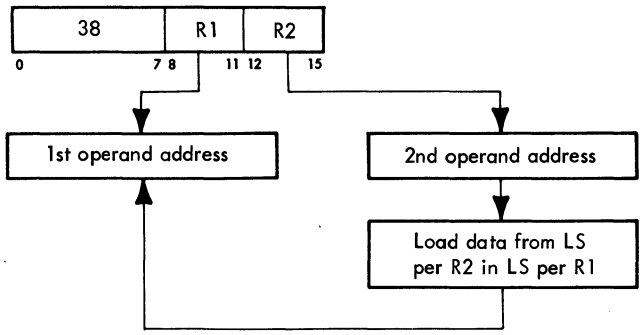
The floating-point instructions and formats, mnemonic codes, op codes, CC's, and interruptions are listed in Table 3-13. All operations can be specified in short and long operands. In the paragraphs following, the instructions are analyzed, illustrated by flow charts.

3.6.1 LOAD

- Loads 2nd operand in 1st operand location.
- 2nd operand location remains unchanged.

3.6.1.1 LER (38) - RR Short Operands

- Loads 2nd operand (per R2) in 1st operand location (per R1).
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- Figure 6035, FEDM.

3.6.1.2 LE (78) - RX Short Operands

- Loads 2nd operand (from main storage) in 1st operand location (per R1).
- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

TABLE 3-13. FLOATING-POINT INSTRUCTIONS

Instruction	Format	Mnemonic Code	Op Code	Condition Codes	Interruption
Load (Short	RR	LER	38	-	S
Load (Short)	RX	LE	78	-	P, A, S
Load (Long)	RR	LDR	28	-	S
Load (Long)	RX	LD	68	-	P, A, S
Load and Test (Short)	RR	LTER	32	0, 1, 2	S
Load and Test (Long)	RR	LTDR	22	0, 1, 2	S
Load Complement (Short)	RR	LCER	33	0, 1, 2	S
Load Complement (Long)	RR	LCDR	23	0, 1, 2	S
Load Positive (Short)	RR	LPER	30	0, 2	S
Load Positive (Long)	RR	LPDR	20	0, 2	S
Load Negative (Short)	RR	LNER	31	0, 1	S
Load Negative (Long)	RR	LNDR	21	0, 1	S
Add Normalized (Short)	RR	N AER	3A	0, 1, 2, 3	S, U, E, LS
Add Normalized (Short	RX	N AE	7A	0, 1, 2, 3	P, A, S, U, E, LS
Add Normalized (Long)	RR	N ADR	2A	0, 1, 2, 3	S, U, E, LS
Add Normalized (Long)	RX	N AD	6A	0, 1, 2, 3	P, A, S, U, E, LS
Add Unnormalized (Short)	RR	AUR	3E	0, 1, 2, 3	S, E, LS
Add Unnormalized (Short)	RX	AU	7E	0, 1, 2, 3	P, A, S, E, LS
Add Unnormalized (Long)	RR	AWR	2E	0, 1, 2, 3	S, E, LS
Add Unnormalized (Long)	RX	AW	6E	0, 1, 2, 3	P, A, S, E, LS
Subtract Normalized (Short)	RR	N SER	3B	0, 1, 2, 3	S, U, E, LS
Subtract Normalized (Short)	RX	N SE	7B	0, 1, 2, 3	P, A, S, U, E, LS
Subtract Normalized (Long)	RR	N SDR	2B	0, 1, 2, 3	S, U, E, LS
Subtract Normalized (Long)	RX	N SD	6B	0, 1, 2, 3	P, A, S, U, E, LS
Subtract Unnormalized (Short)	RR	SUR	3F	0, 1, 2, 3	S, E, LS
Subtract Unnormalized (Short)	RX	SU	7F	0, 1, 2, 3	P, A, S, E, LS
Subtract Unnormalized (Long)	RR	SWR	2F	0, 1, 2, 3	S, E, LS
Subtract Unnormalized (Long)	RX	SW	6F	0, 1, 2, 3	P, A, S, E, LS
Compare (Short)	RR	CER	39	0, 1, 2	S
Compare (Short)	RX	CE	79	0, 1, 2	P, A, S

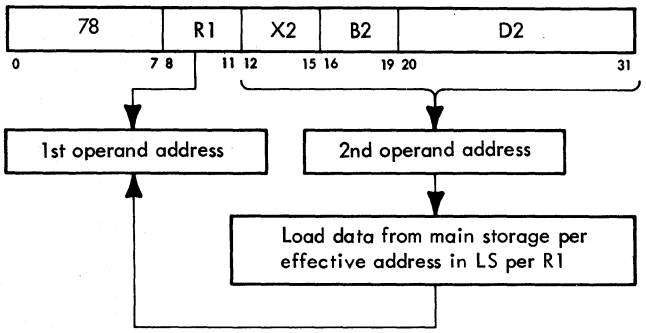
TABLE 3-13. FLOATING-POINT INSTRUCTIONS (Cont)

Instruction	Format	Mnemonic Code	Op Code	Condition Codes	Interruption
Compare (Long)	RR	CDR	29	0, 1, 2	S
Compare (Long)	RX	CD	69	0, 1, 2	P, A, S
Halve (Short)	RR	HER	34	-	S
Halve (Long)	RR	HDR	24	-	S
Multiply (Short)	RR	N MER	3C	-	S, U, E
Multiply (Short)	RX	N ME	7C	-	P, A, S, U, E
Multiply (Long)	RR	N MDR	2C	-	S, U, E
Multiply (Long)	RX	N MD	6C	-	P, A, S, U, E
Divide (Short)	RR	N DER	3D	-	S, U, E, FK
Divide (Short)	RX	N DE	7D	-	P, A, S, U, E, FK
Divide (Long)	RR	N DDR	2D	-	S, U, E, FK
Divide (Long)	RX	N DD	6D	-	P, A, S, U, E, FK
Store (Short)	RX	STE	70	-	P, A, S
Store (Long)	RX	STD	60	-	P, A, S

Notes:

A	Addressing	N	Normalized operation
E	Exponent overflow	P	Protection
FK	Floating-point divide	S	Specification
LS	Significance	U	Exponent underflow

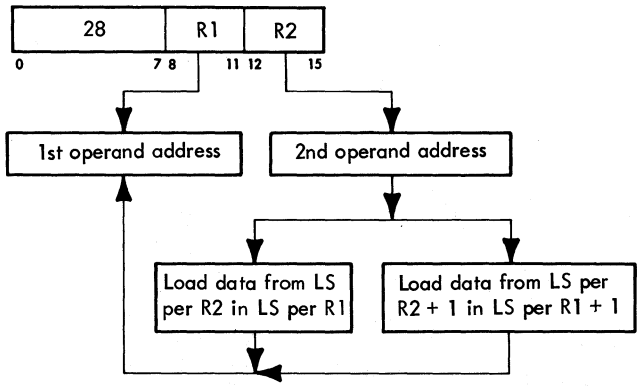
- RX format:



- Main storage address must be on word boundary.
- D(21) determines which word of main storage double word is used: if 1, right word; if 0, left word.
- Figure 6036, FEDM

3.6.1.3 LDR (28) - RR Long Operands

- Loads 2nd operand (per R2 and R2 + 1) in 1st operand location (per R1 and R1 + 1).
- RR format:



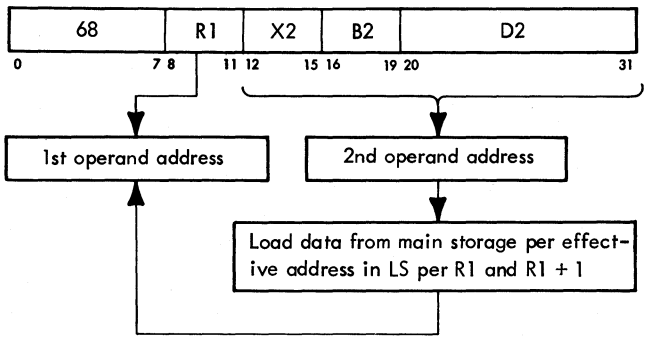
- Conditions at start of execution
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- Figure 6035, FEDM.

3.6.1.4 LD (68) - RX Long Operands

- Loads 2nd operand (double word from main storage) in 1st operand location (per R1 and R1 + 1).

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

- Main storage address must be on double-word boundary.

- Figure 6036, FEDM.

3.6.2 LOAD AND TEST

- Loads 2nd operand in 1st operand location; sign and magnitude of 2nd operand determine CC.

- 2nd operand location is unchanged.

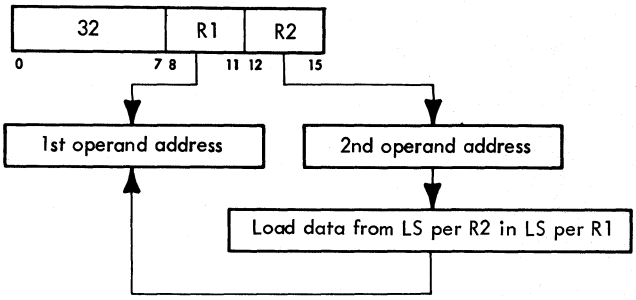
Except for the setting of the CC, the Load and Test instruction execution is similar to the Load RR format instructions.

The CC is determined during the normal end-op cycle. If the fraction in PAL equals zero, the CC is set to 0. The sign and the characteristic are not considered when the fraction equals zero. If the operand fraction is not equal to zero, the sign determines a greater-than- or less-than-zero condition. If the sign is minus (equals 1), the result is less than zero and a 1 is set in the CC. If the sign is plus (equals 0), the result is greater than zero and a 2 is set in the CC. A CC of 3 is invalid for this instruction. Setting the CC depends upon a micro-order, the instruction, and hardware conditions.

3.6.2.1 LTER (32) - RR Short Operands

- Loads 2nd operand (per R2) in 1st operand location (per R1); sign and magnitude of 2nd operand determine CC.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- Sign of 2nd operand saved in STAT C.

- STAT A set if fraction equals zero

- CC determined by STAT A, STAT C, and LTER instruction.

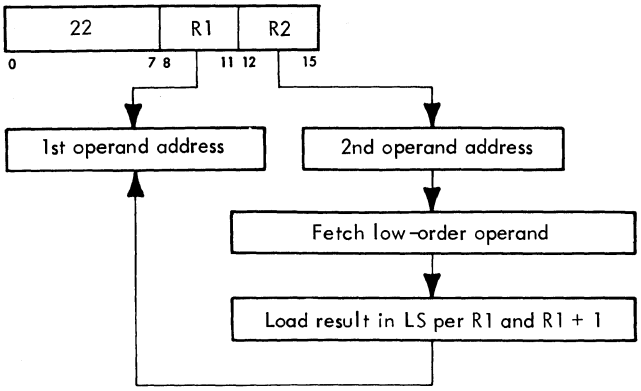
- CC setting:
  Fraction equals zero: CC = 0.
  Result less than zero: CC = 1.
  Result greater than zero: CC = 2.

- Figure 6037, FEDM.

3.6.2.2 LTDR (22) - RR Long Operands

- Loads 2nd operand (per R2 and R2 + 1) in 1st operand location (per R1 and R1 + 1); sign and magnitude of 2nd operand determine CC.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- Sign of 2nd operand saved in STAT C.

- STAT A set if fraction equals zero.

- CC determined by STAT A, STAT C, and LTDR instruction.

- CC setting:
  Fraction equals zero: CC = 0
  Result less than zero: CC = 1
  Result greater than zero: CC = 2.

- Figure 6038, FEDM

3.6.3 LOAD COMPLEMENT

- Loads 2nd operand in 1st operand location with sign changed to opposite value (complemented). Result sign and magnitude determine CC.

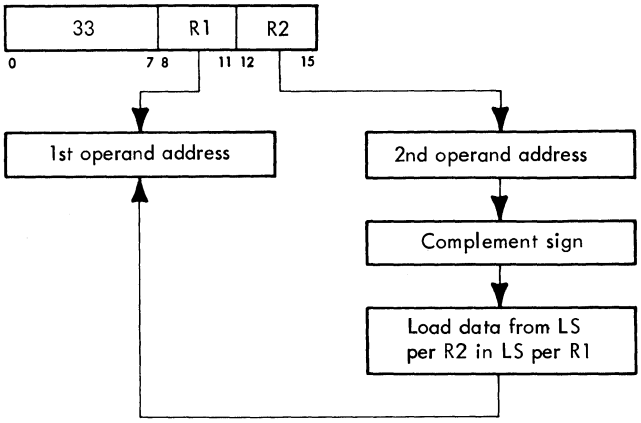
- 2nd operand location unchanged.

- Except for complementing sign of 2nd operand, Load Complement instructions are similar to Load and Test instructions (paragraph 3.6.2).

3.6.3.1 LCER (33) - RR Short Operands

- Loads 2nd operand (per R2) in 1st operand location (per R1) with sign complemented. Result sign and magnitude determine CC.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only)
  2nd operand is in S and T.
  Instruction is in E.

- Sign of 2nd operand saved in STAT C.

- STAT A set if fraction equals zero.

- CC determined by STAT A, STAT C, and LCER instruction.

- CC setting:
  Fraction equals zero: CC = 0.
  Result less than zero: CC = 1.
  Result greater than zero: CC = 2.

- Figure 6037, FEDM.

3.6.3.2 LCDR (23) - RR Long Operands

- Loads 2nd operand (per R2 and R2 + 1) in 1st operand location (per R1 and R1 + 1) with sign complemented. Result sign and magnitude determine CC.

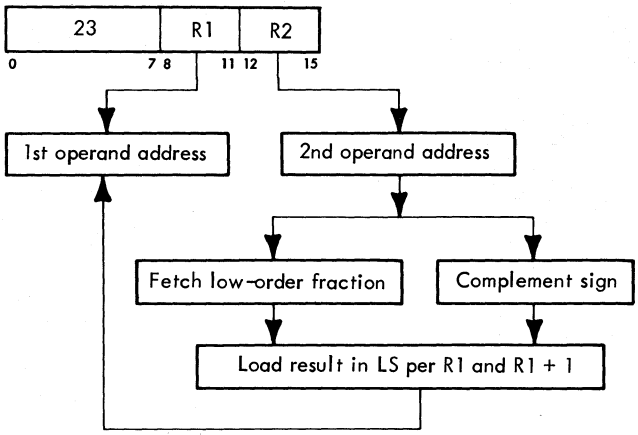
- Sign of 2nd operand saved in STAT C.

- STAT A set if fraction equals zero.

- CC determined by STAT A, STAT C, and LCDR instruction.

- CC setting:
  Fraction equals zero: CC = 0.
  Result less than zero: CC = 1.
  Result greater than zero: CC = 2.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- Figure 6038, FEDM.

3.6.4 LOAD POSITIVE

- Loads 2nd operand in 1st operand location with sign made plus

- 2nd operand location unchanged.

- Except for making sign of 2nd operand plus, Load Positive instructions are similar to Load and Test instructions (paragraph 3.6.2).

3.6.4.1 LPER (30) - RR Short Operands

- Loads 2nd operand (per R2) in 1st operand location (per R1) with sign made plus.

- Result stored must be zero or greater.

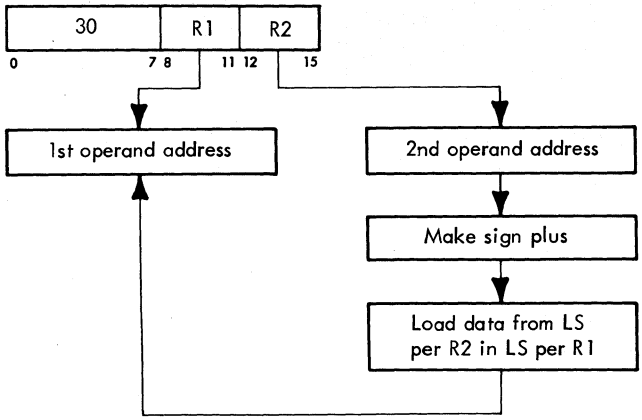
- Sign of 2nd operand saved in STAT C.

- STAT A set if fraction equals zero.

- CC determined by STAT A, STAT C, and LPER instruction.

- CC setting:
  Fraction equals zero: CC = 0.
  Result greater than zero: CC = 2.

- RR format:



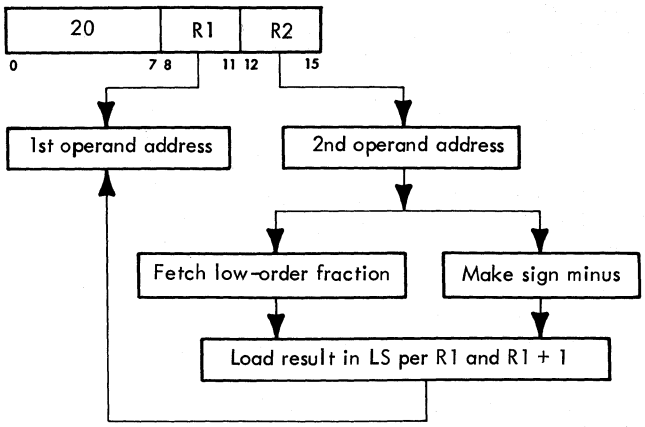
- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- Figure 6037, FEDM.

3.6.4.2 LPDR (20) - RR Long Operands

- Loads 2nd operand (per R2 and R2 + 1) in 1st operand location (per R1 and R1 + 1) with sign made plus.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

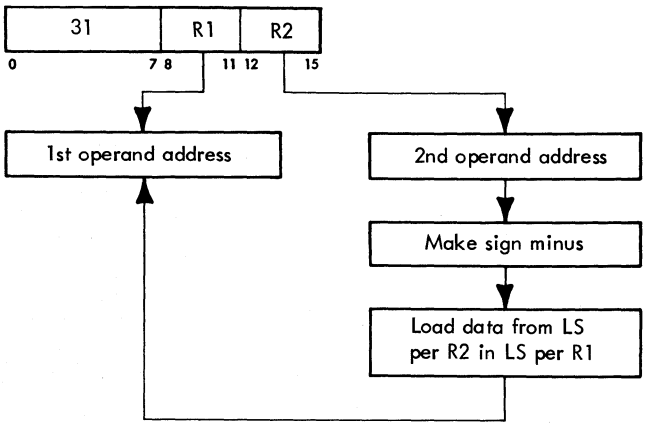
- Result stored must be zero or greater.
- Sign of 2nd operand saved in STAT C.
- STAT A set if fraction equals zero.
- CC determined by STAT A, STAT C, and LPDR instruction.
- CC setting:
  Fraction equals zero: CC = 0.
  Result greater than zero: CC = 2.
- Figure 6038, FEDM.

3.6.5 LOAD NEGATIVE

- Loads 2nd operand in 1st operand location with sign made minus.
- 2nd operand location unchanged.
- Except for making sign of 2nd operand minus, Load Negative instructions are similar to Load and Test Instructions (paragraph 3.6.2).

3.6.5.1 LNER (31) - RR Short Operands

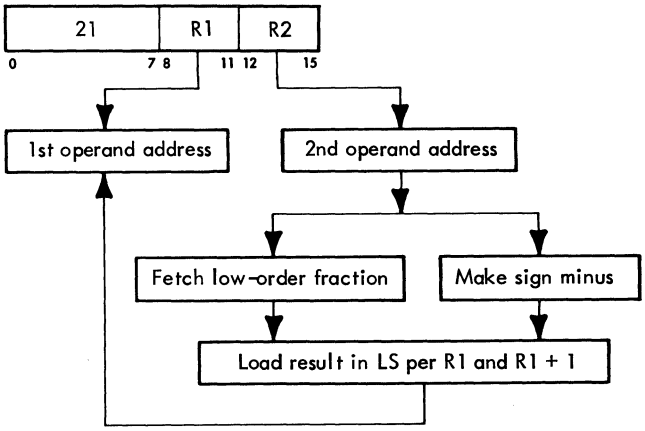
- Loads 2nd operand (per R2) in 1st operand location (per R1) with sign made minus.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- Result stored must be zero or less.
- Sign of 2nd operand saved in STAT C.
- STAT A set if fraction equals zero.
- CC determined by STAT A, STAT C, and LNER instruction.
- CC setting:
  Fraction equals zero: CC = 0.
  Result less than zero: CC = 1.
- Figure 6037, FEDM.

3.6.5.2 LNDR (21) - RR Long Operands

- Loads 2nd operand (per R2 and R2 + 1) in 1st operand location (per R1 and R1 + 1) with sign made minus.
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.
- Result stored must be zero or less.
- Sign of 2nd operand saved in STAT C.
- STAT A set if fraction equals zero.
- CC determined by STAT A, STAT C, and LNDR instruction.
- CC setting:
  Fraction equals zero: CC = 0.
  Fraction less than zero: CC = 1.

- Figure 6038, FEDM.

3.6.6 ADD-TYPE INSTRUCTIONS

- 20 add, subtract, and compare instructions.
- Short and long operands available in both formats.
- 2nd operand location is unchanged.
- Add and subtract instruction results may be normalized or unnormalized.
- Results in true form with plus or minus values.

There are 20 floating-point add-type instructions, divided into three major groups: add, subtract, and compare. The RR and RX formats using short and long operands are available in each group. The add and subtract instruction results may be normalized or unnormalized, depending upon the instruction being executed. A CC is set on all add-type instructions; the compare instructions cause a CC to be set with no result stored.

The CPU computes the sum of floating-point numbers as follows:

1. Equalizes the characteristics.
   a. If the characteristics are unequal, the operand with the smallest characteristic is shifted right the number of hexadecimal digits necessary to equalize the characteristics.
   b. If the number of shifts exceeds the number of hexadecimal digits available, the operand with the largest characteristic becomes the intermediate result.
2. When the characteristics are equal, algebraically adds the first and second operands.
   a. If the signs are alike, adds the first operand to the second operand.
   b. If the signs are unlike, subtracts the second operand from the first operand (adds 2's complement of second operand to first operand).
3. If the intermediate result fraction is in complement form, recomplements it (takes 2's complement) to obtain the true fraction value.
4. Determines the sign and characteristic value.
5. Stores the sign, characteristic, and fraction in LS as specified by the R1 field.
6. Sets the CC per hardware conditions.

The add instructions are discussed in paragraphs 3.6.6.1 and 3.6.6.2.

For subtraction of floating-point numbers, the algebraic rule applies: to subtract two numbers, change the sign of the subtrahend and proceed as in addition. When subtracting floating-point numbers, the sign of the second operand is complemented. The rules of addition apply as outlined in the previous paragraph. The subtract instructions are discussed in paragraphs 3.6.6.3 and 3.6.6.4.

The compare instructions are similar to the subtract instructions; the results, however, are not stored. The objectives of the compare instructions are to algebraically compare the first operand with the second operand and to set the CC accordingly. These objectives are accomplished by complementing the sign, algebraically adding the fraction, determining a high, low, or equal condition, and setting the CC. The compare instructions are discussed in paragraph 3.6.6.5.

As discussed in the previous paragraphs, the subtract and compare instructions may be treated as an add instruction after the sign is complemented. The basic objectives of the add-type instructions are shown in Figure 6039A, FEDM. The figure is divided into three phases. The objectives of the first phase are I-Fetch, operand fetch, and sign handling. After the RR or RX I-Fetch and the specification check, the remaining operand and/or low-order fraction(s) must be fetched or the low-order fractions reset to zeros. The signs are saved in STAT's. In short operand instructions, zeros are gated to the low-order fractions of the 64-bit operands.

In the second phase, the characteristics are compared. Preshifting occurs, if necessary, followed by the addition or subtraction of the fractions. Because the characteristics must be equal before algebraically adding the operands, the characteristics are subtracted to determine whether they are equal or whether preshifting is meaningful. For short operands, the characteristic difference must be 7 or less; for long operands, 15 or less. An exception

occurs if the first operand characteristic is greater than the second operand characteristic and the difference is 8 (short operands) or 16 (long operands). In this case, characteristic equalization results in a zero fraction.

If the characteristic difference is greater than 7 (short operands) or 15 (long operands), the fraction resulting after right-shifting equals zero. Therefore, preshifting is not performed and the operand with the largest characteristic becomes the result. If preshifting is meaningless, the value in AB or ST is the result. If the characteristics are within range, the smallest fraction is right-shifted until the characteristics are equal; the operands are then algebraically added.

Phase 3 tests for compare instructions, normalized instructions, or unnormalized instructions. Postnormalization, recomplementation, or fraction overflow correction is accomplished during this phase. The final result is stored (except on compare instructions), and the CC is set according to the computed results. An end-op completes instruction execution.

When short operand instructions are executed, the low-order halves of the floating-point registers are ignored and remain unchanged.

The addition of two floating-point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted until the two characteristics agree. The characteristic is increased by 1 for each hexadecimal digit shifted. The fractions are then added algebraically to form an intermediate result. If an overflow carry occurs, the intermediate result is right-shifted one hexadecimal digit and the characteristic is increased by 1. If this increase causes a characteristic overflow, an exponent-overflow interruption occurs.

The short intermediate result consists of seven hexadecimal digits and a possible carry. The low-order digit is a guard digit retained from the fraction which is shifted right. Only one guard digit participates in the fraction addition. The guard digit is zero if no shift occurs. For long operands, the intermediate result consists of 14 hexadecimal digits and a possible carry. No guard digit is retained.

After the addition, for normalized instructions the intermediate result fraction is left-shifted as necessary to form a normalized fraction. Vacated low-order digit positions are filled with zeros, and the characteristic is reduced by the amount of the shift.

If normalization causes the characteristic to underflow, an exponent-underflow interruption condition exists: the sign, characteristic, and fraction are made zero and, if the corresponding mask bit is a 1, a program interruption occurs. If no left shift takes place, the intermediate result is truncated to the proper fraction length (short operands).

When the intermediate result fraction is zero and the significance mask bit is a 1, a significance interruption takes place. No normalization occurs, and the intermediate result characteristic remains unchanged. When the intermediate result is zero and the significance mask bit is a 0, a significance interruption does not occur; rather, the characteristic and the sign are made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the result is derived algebraically. However, the sign of a result with a zero result fraction is always positive.

All instructions depend on ROS microprograms to perform the logic for instruction execution. Each instruction enters at a different ROS microprogram address. After microprogram entry, the address of the next micro-instruction is determined and the next micro-instruction is fetched and executed. In the add-type instructions, there are 20 different microprogram entries. The RR or RX I-Fetch routine is executed. After I-Fetch, one of the 20 ROS entries is selected and the execution phase begins. Since the add-type instructions may be considered as an algebraic add, the 20 microprograms eventually branch to perform the same logic functions. Instruction execution for all add-type instructions eventually continues in the same microprogram. The point of entry, for discussion purposes, is considered to be the branch on characteristic difference (10-way branch) shown in Figure 6039C, FEDM. The details of instruction execution are included in the discussion of the specific instruction.

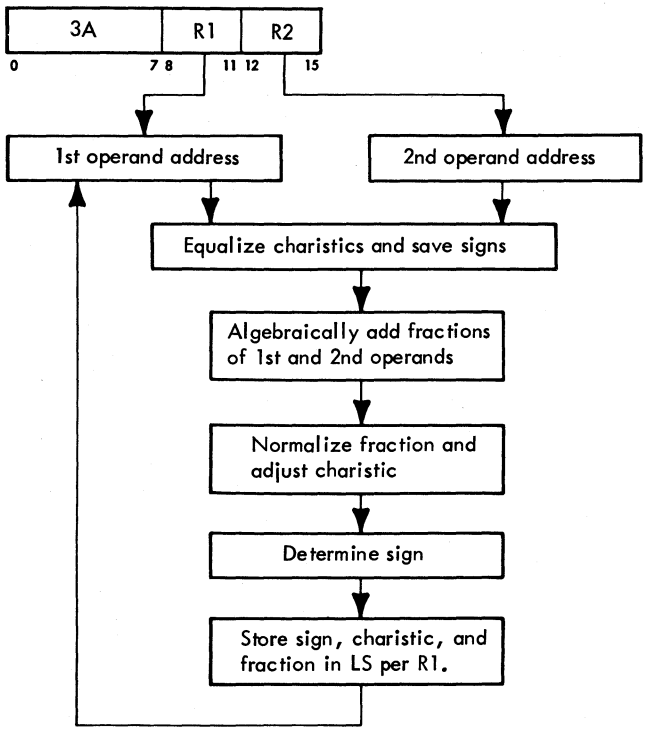
3.6.6.1 Add Normalized

- Algebraically adds 2nd operand to 1st operand; normalized sum is placed in 1st operand location.
- 2nd operand location is unchanged.
- Instructions:
  AER (3A) - RR Short Operands
  AE (7A) - RX Short Operands
  ADR (2A) - RR Long Operands
  AD (6A) - RX Long Operands

3.6.6.1.1 AER (3A) - RR Short Operands

- Algebraically adds 2nd operand (per R2) to 1st operand (per R1); normalized sum is placed in 1st operand location.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.

The Add Normalized, AER, instruction algebraically adds the second operand specified by R2 to the first operand specified by R1, and the normalized sum is placed in the first operand location. The CC is set according to hardware conditions.

The AER instruction is in the RR format with an op code of 3A. This instruction uses 32-bit operands. The contents of the low-order halves of the floating-point registers in LS remain unchanged.

The initial conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC was set to 4 during I-Fetch.
4. The AER instruction is in E.

A specification check is made at the beginning of the execution phase. If a specification check condition exists, instruction execution is suppressed and a specification interruption occurs. Assume that no specification check interruption condition exists (Figure 6039B, FEDM).

Because the AER instruction uses short operands (32 bits) in the RR format, no operand fetch during instruction execution is necessary. The sign of the first operand is saved in STAT F. Because the sign of the second operand depends upon the instruction being executed, STAT C contains the original sign or the complement of the original sign. For the AER instruction, STAT C contains the original sign. Because the AER instruction operates on short operands, B and T are reset, and the operand is treated as a 56-bit fraction. The first operand characteristic is subtracted from the second operand characteristic to determine the characteristic difference. The characteristic difference and the signs determine the next operation to be performed via a 10-way ROS branch (Figure 6039C, FEDM).

The 10-way ROS branch on characteristic difference and signs occurs for all add-type instructions. When this branch is encountered, the conditions are as follows:

1. The first operand is in AB (B equals zero for short operands).
2. The second operand is in S, T, and D (T equals zero for short operands).
3. The sign of the first operand is in STAT F.
4. The sign of the second operand is in STAT C.
5. The characteristic difference is in SAL and F.

The serial adder consists of eight binary bit positions. When subtracting the first operand characteristic from the second operand characteristic, the 2's complement of the first operand characteristic is

added to the second operand characteristic. A 1 is forced in bit position 0 on the A bus (first operand side) of the serial adder, and a 0 is forced in bit position 0 on the B bus (second operand side) of the serial adder. The characteristic difference is then routed to SAL and F.

The 10-way ROS branch is determined by the results of the characteristic subtraction and the signs in STAT F and STAT C. The 10-way ROS branch tests are defined as ABCD values in Figure 6039C, FEDM. The ABCD values are defined as follows:

1. A equals 1 when there is a serial adder carry. A carry indicates that the second operand is greater than or equal to the first operand.

2. B equals 1 when the signs are alike. If the signs are alike, the fractions are added; if unlike, the fractions are subtracted.

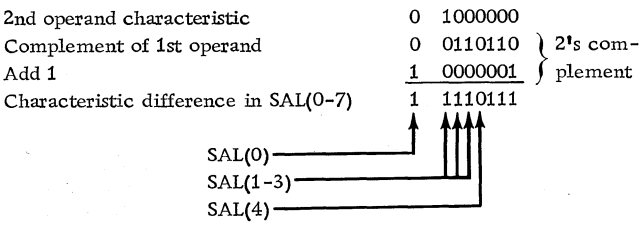
3. C equals 1 when the characteristic difference is within range. C equaling 1 implies that the characteristic difference is small enough, and equalizing the characteristics may be meaningful. (It is possible to have a zero fraction as a result of characteristic equalization.)

4. D equals 1 when the result in the serial adder latches is 0. This condition indicates that the characteristics are equal. A serial adder carry also occurs; thus A will also equal 1.

Assume that the following two characteristics are to be compared to determine the ABCD value:

2nd operand characteristic $64_{10} = 1000000_2$
1st operand characteristic $73_{10} = 1001001_2$

The first operand characteristic is subtracted from the second operand characteristic shown below:



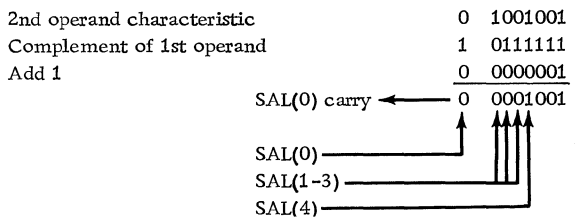
Because there was no SAL(0) carry, the value of A is 0. The value of B depends upon the signs assigned to the fractions of the two operands. The table in Figure 6039C, FEDM, shows the bit positions tested in SAL to determine the value of C and whether preshifting is meaningful. In this example, SAL(0-3) equals 1111 (binary). If a long operand instruction is being executed, C equals 1 and preshifting is meaningful. If a short operand instruction is being executed, C equals 0; therefore, the value is in AB because no carry from SAL(0) indicated that R1 is greater than R2. D equals 0 because SAL(0-7) is not all 0's.

If all positions of SAL are 0, D equals a 1, indicating that the characteristics of the two operands are equal. The value of B then determines the next operation (signs alike, add signs; unlike, subtract). In this case [SAL(0-7) equals 0], A also equals a 1 because there must be a SAL(0) carry.

Further, assume that the two characteristics in the previous example are interchanged. The characteristics are compared as follows:

2nd operand characteristic: 1001001
1st operand characteristic: 1000000

The first operand characteristic is subtracted from the second operand characteristic shown below:



In this example, A equals 1 because there is a SAL(0) carry. The value of B depends upon the signs of the two operands. The conditions listed in the table in Figure 6039C, FEDM, show that C equals 1 for long operand instructions and 0 for short operand instructions. If preshifting in this example is meaningless, the operand in ST is the result fraction because the SAL(0) carry indicated that the second operand is the largest.

The two examples discussed above illustrate the determination of the ABCD values. Additional examples are shown in Table 3-14. The value of ABCD determines the ROS branch that performs the next steps in executing the Add algorithm.

The examples of determining the ABCD values as shown in Table 3-14 indicate that the fraction of the operand with the smallest characteristic is shifted right when the characteristic difference is 7 or less for short operand instructions and 15 or less for long operand instructions. An exception occurs when the characteristic difference is 8 (short operands) or 16 (long operands) and the first operand characteristic

TABLE 3-14. EXAMPLES OF BRANCHING ON CHARACTERISTIC DIFFERENCE

Description	Example No. 1	Example No. 2	Example No. 3	Exampe No. 4	Example No. 5
2nd operand characteristic	1000000	1000000	1001000	1000111	1000000
1st operand characteristic	1001001	1001000	1000000	1000000	1000000
2nd operand characteristic	0 1000000	0 1000000	0 1001000	0 1000111	0 1000000
Complement of 1st operand characteristic	1 0110110	1 0110111	1 0111111	1 0111111	1 0111111
Add 1	0 0000001	0 0000001	0 0000001	0 0000001	0 0000001
Difference in SAL(0-7)	1 1110111	1 1111000	← 0 0001000	← 0 0000111	← 0 0000000
			SAL(0) carry	SAL(0) carry	SAL(0) carry
Within Range?* Short	No	Yes	No	Yes	Yes
Within Range?* Long	Yes	Yes	Yes	Yes	Yes
ABCD value	Sub 0000 Add 0100	Sub 0010 Add 0110	Sub 1000 Add 1100	Sub 1010 Add 1110	Sub 1011 Add 1111
Comments	Result in AB	Equalize fraction in ST	Result in ST	Equalize fraction in AB	Add or sub. No shift necessary.

Notes:

1. A equals 1 when there is a serial adder carry. A carry indicates that R2 $\geq$ R1.
2. B equals 1 when the signs are alike.
*3. C equals 1 when the characteristics are within range. C equals 1 on SAL results as follows:
   a. SAL carry and SAL(0-3) = 0 and long operands.
   b. SAL carry and SAL(0-4) = 0.
   c. No SAL carry and SAL(0-3) = 1's and long operands.
   d. No SAL carry and SAL(0-4) = 1's.
4. D equals 1 when the SAL outputs are equal to 0.

is the largest characteristic. In this case, 8 or 16 right hexadecimal shifts occur, resulting in a zero fraction.

Four possible ABCD values (0010, 0110, 1010, and 1110) cause characteristic equalization and then an algebraic addition of fractions (Figure 6039C, FEDM). For example, assume that an AER instruction requires characteristic equalization, fraction subtraction, recomplementation (second operand fraction is greater than first operand fraction), and normalization. Further, assume an ABCD value of 0010. The 0010 branch (Figure 6039C, FEDM) shows that one right shift of the second operand occurs and 1 is added to F. Note that one guard digit is retained. SAL(4-7) is checked for 1111 (binary) (Figure 6039D, FEDM). When SAL(4-7) equals 1111, the characteristics are equal. Since the test for a branch is made one machine cycle before the ROS branch occurs, the SAL value is one machine cycle behind the actual shift count. For this reason, a test is made for 1111 in SAL(4-7) instead of for 0000. Once the characteristics are equal, the second operand is subtracted from the first operand (signs unlike). To subtract fractions (signs unlike), the 2's complement of the second operand fraction in DT is added to the first operand fraction in AB with the intermediate fraction result placed in AB and DT. The intermediate fraction result may be in true form or in complement form, or it may be equal to zero. If a zero fraction results, STAT A is set. If the fraction of the second operand is greater than the first operand fraction, the result is in complement form. Conversely, the result is in true form if the first operand fraction is greater than the second operand fraction. If AB(7) equals 1, the intermediate result is in complement form. If AB(7) equals 0, the intermediate result is in true form. When the intermediate result is in complement form, the result must be recomplemented. The true form of the intermediate result fraction is accomplished by taking the 2's complement of the intermediate result after the algebraic addition (Figure 6039G, FEDM).

When the result is in true form, and if the fraction is not equal to zero, the fraction must be normalized and stored and the CC set. The microprogram assumes that the intermediate result is normalized. Therefore, the low-order result fraction

is stored in the first operand location in LS (long operands only). Since the AER instruction is a normalized instruction, the intermediate result is normalized, if necessary. The low-order fraction is stored after each left shift (long operands). After normalization is complete, the sign and the characteristic are inserted and stored with the high-order fraction in the first operand location (specified by R1). Assuming no error conditions or zero fraction, the CC is set (Figure 6039I, FEDM). An end-op cycle completes instruction execution.

When in the normalizing loop (Figure 6039G, FEDM), the intermediate fraction result can be left-shifted out of the high-order hexadecimal digit position if the intermediate fraction is 0001. This left shift results in a zero fraction. The zero fraction, in this case, is not a true zero result or a significance condition; therefore, the true value must be restored.

Since the test for ROS branches is made one machine cycle before the ROS branch occurs, a test for normalization is made before the recomplementation is performed. Therefore, the test for normalization is determined by the following conditions:

1. PAL(7-11) is 0's and PAL(7-63) not 0's.
2. PAL(6,8-11) is 1's and PAL(7-63) not 0's.

If one of these two conditions is met, the machine assumes that one normalization cycle is necessary after the recomplementation machine cycle. For example, if the following fractions are subtracted, the assumed normalization cycle is not necessary.

AB bit positions	6	7	8	9	10	11	12	13←→63
1st operand fraction	0	0	0	1	1	1	0	0←→0
2nd operand fraction	0	0	1	0	0	0	0	0←→0
1st operand fraction	0	0	0	1	1	1	0	0←→0
Subtract 2nd operand from 1st operand (2's complement of 2nd operand)	1	1	0	1	1	1	1	1←→1
Intermediate result fraction (Meets condition 2)	1	1	1	1	1	1	0	0←→0
Indicates recomplementation necessary								1
(2's complement of 11 1111 00←→0)	0	0	0	0	0	0	1	1←→1
Result before left shift	0	0	0	0	0	1	0	0←→0

The intermediate result fraction above shows that PAL(6,8-11) equals 1's and that PAL(7-63) does not equal 0's. This condition causes a ROS branch to the ROS normalization routine. Since A(7) equals a 1, the intermediate result fraction is in complement form, and the 2's complement of the intermediate fraction must be performed to obtain the true result fraction. As shown in the example, the true result fraction is .0001 0←→0. The one hexadecimal left shift that occurs yields a zero fraction result. Therefore, the result fraction located in DT is the true result fraction. The contents of D are transferred to T, and the sign, characteristic, and high-order fraction are stored in LS per the R1 field. The low-order fraction has previously been stored (long operands). An end op completes instruction execution.

If the signs were alike and characteristic equalization was necessary, the fractions are added (Figure 6039D or E, FEDM). When adding fractions, the possibility of a fraction overflow exists. The fraction overflow is indicated by a carry out of the high-order position [PA(8)]. After addition of the two fractions, the intermediate result is placed in DT and AB. If A(7) equals a 1, a fraction overflow occurred. Therefore, the fraction must be right-shifted one hexadecimal digit and 1 added to the intermediate result characteristic. If there is no fraction overflow, the result is normalized and stored, and the CC set (Figure 6039F, FEDM).

If a right shift of the intermediate result fraction is necessary, the possibility of an exponent overflow exists. During the normalization of the fraction, the possibility of an exponent underflow exists. When SAL(0) equals a 1 after a fraction shift (right or left), an exponent overflow or exponent underflow condition exists. A true zero result is stored in the first operand location when an exponent underflow occurs.

Whenever an exponent overflow or exponent underflow condition exists, F(1) and PSW(38) are examined to determine whether a program interruption is to be executed. If F(1) equals a 0, an exponent overflow exists and an interruption request is unconditionally generated. A program interruption occurs on all exponent overflows. If F(1) equals a 1, an exponent underflow exists; if PSW(38) equals a 1, an exponent underflow interruption request is generated. If PSW(38) equals a 0, exponent underflow is masked off, and a true zero is stored with no program interruption occurring.

Note that an interruption occurs on all exponent overflows. This overflow indicates that the value of the absolute result exceeds the limits of the machine;

therefore, the results are unpredictable and further action is necessary. In some scientific computations, very small numbers may be eliminated from an equation without serious error. In the case of exponent underflow, the computed result approaches zero. Therefore, the programmer may find that a program interruption is unnecessary, and a true zero result is desirable.

Significance and specification interruption conditions may also exist during execution of AER instruction. The action that occurs is shown in Figure 6039I, FEDM, and is discussed earlier in this section.

Tests for zero intermediate results are made at several points during instruction execution. If the result is zero, a program interruption occurs if PSW(39) is a 1. The positive sign, the result characteristic, and a zero fraction are stored in LS. A program interruption occurs, and the program interruption routine determines the action to be taken. If PSW(39) equals a 0, a true zero result is stored in LS.

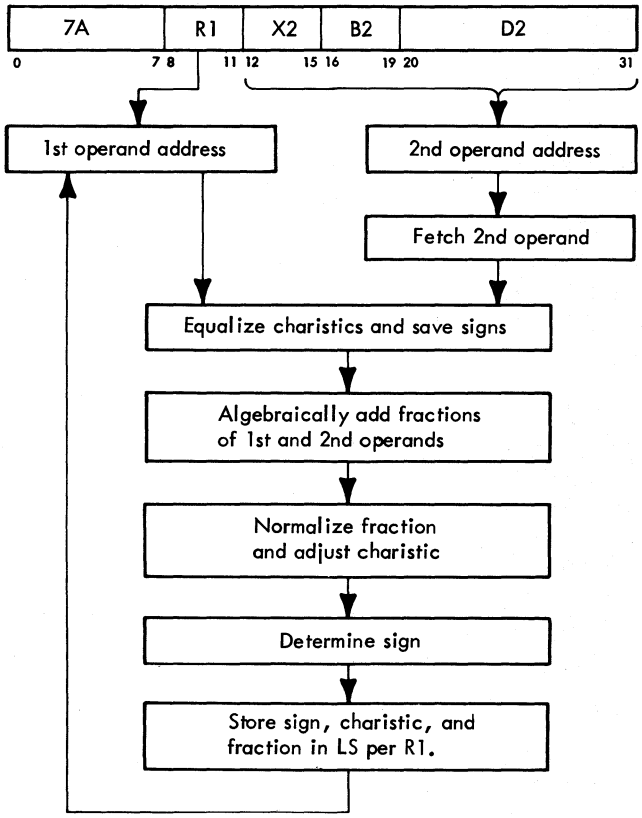
An interruption request is generated during instruction execution. At end-op time, the interruption request is honored, provided no interruption requests of higher priority are pending.

If the characteristics are not within limits when executing the AER instruction, the fraction with the largest characteristic is normalized and stored along with the sign in LS per the R1 field.

This discussion of the AER instruction is referred to in paragraphs 3.6.6.1.2 through 3.6.6.5.4. If the AER instruction is understood and the instruction differences noted, any add-type instruction execution path can be followed by referring to Figure 6039, FEDM.

3.6.6.1.2 AE (7A) - RX Short Operands

- Algebraically adds 2nd operand (from main storage) to 1st operand (per R1); normalized sum is placed in 1st operand location.
- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- RX format:



The Add Normalized (AE) instruction algebraically adds the second operand (from main storage) to the first operand (specified by R1), and the normalized sum is placed in the first operand location. The CC is set according to hardware conditions.

The AE instruction is in the RX format with an op code of 7A and uses 32-bit operands. The low-order halves of the floating-point registers in LS remain unchanged.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. The effective address of the second operand is in D.
3. The contents of A and B are unknown.
4. The AE instruction is in E.

A specification check is made at the beginning of the execution phase (Figure 6039B, FEDM). If a specification interruption condition exists, instruction execution is suppressed and a specification interruption occurs. Assume that no specification interruption condition exists.

Since the AE instruction uses short operands (32 bits) in the RX format, no low-order fractions need to be fetched. The first operand that is in S and T is moved from T to A. The second operand arrives from main storage and is placed in T. D(21) determines which word from the SDBO is gated to T. Note that main storage is addressed on full-word boundaries. If D(21) is equal to a 0, bit positions 0 through 31 of the SDBO are gated to T; if D(21) equals a 1, bits 32 through 63 of the SDBO are gated to T.

The sign of the first operand is saved in STAT F; the sign of the second operand is saved in STAT C. The sign, the characteristic, and the fraction of the second operand are placed in S. B and T are reset, and the operands are treated as 56-bit fractions. The characteristics are subtracted, and the characteristic difference and the signs determine the next operation to be performed via a 10-way branch (Figure 6039C, FEDM).

Operation from the 10-way microprogram branch is similar to the AER instruction (paragraph 3.6.6.1.1). From this point, the microprogram is not concerned with the instruction format.

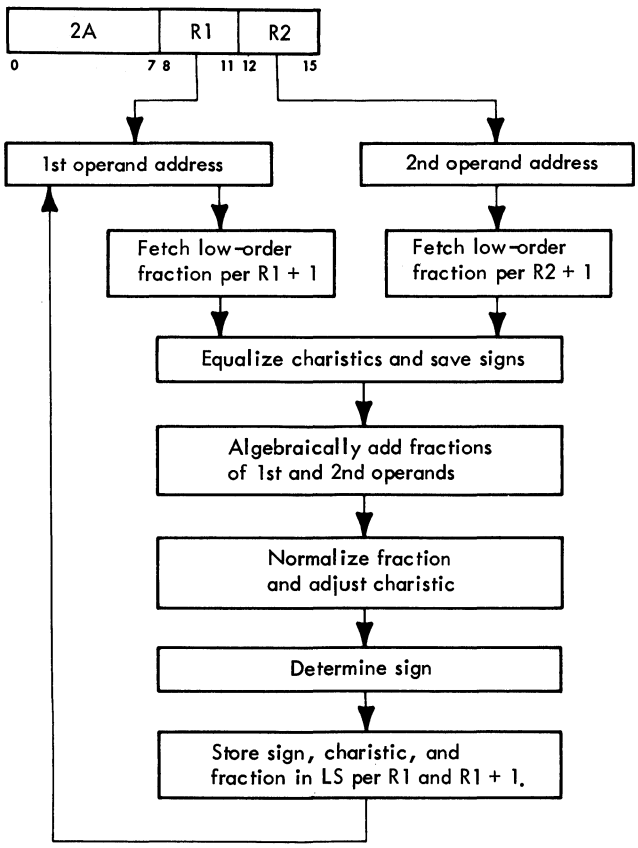
3.6.6.1.3 ADR (2A) - RR Long Operands

- Algebraically adds 2nd operand (per R2 and R2 + 1) to 1st operand (per R1 and R1 + 1); normalized sum is placed in 1st operand location.

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.

- RR format:



The Add Normalized (ADR) instructions algebraically adds the second operand (specified by R2 and R2 + 1) to the first operand (specified by R1 and R1 + 1), and the normalized sum is placed in the first operand location. The CC is set according to hardware conditions.

The ADR instruction is in the RR format with an op code of 2A and uses 64-bit operands.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand (sign, characteristic, and high-order fraction) are in A, B, and D (24 bits).

2. The second operand (sign, characteristic, and high-order fraction) is in S and T.

3. The STC contains a count of 4.

4. The ADR instruction is in E.

Since the ADR instruction uses long operands (64 bits) in the RR format, the low-order fractions of

the first and second operands must be fetched from LS. The low-order fraction of the first operand is fetched from LS per E(8-11) + 1 and placed in B via T and the parallel adder. The low-order fraction of the second operand is fetched from LS per E(8-11) + 1 and placed in T. The high-order fraction is placed in D. Addition is done by algebraically adding the contents of DT to the contents of AB with the intermediate fraction result placed in AB and DT.

The sign of the first operand is saved in STAT F; the sign of the second operand is saved in STAT C. The first operand characteristic is subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation via a 10-way branch (Figure 6039C, FEDM).

Operation from the 10-way microprogram branch is similar to that of the AER instruction (paragraph 3.6.6.1.1). From this point, the microprogram is not concerned with the instruction format.

The major differences between the ADR instruction and the AER instructions are as follows:

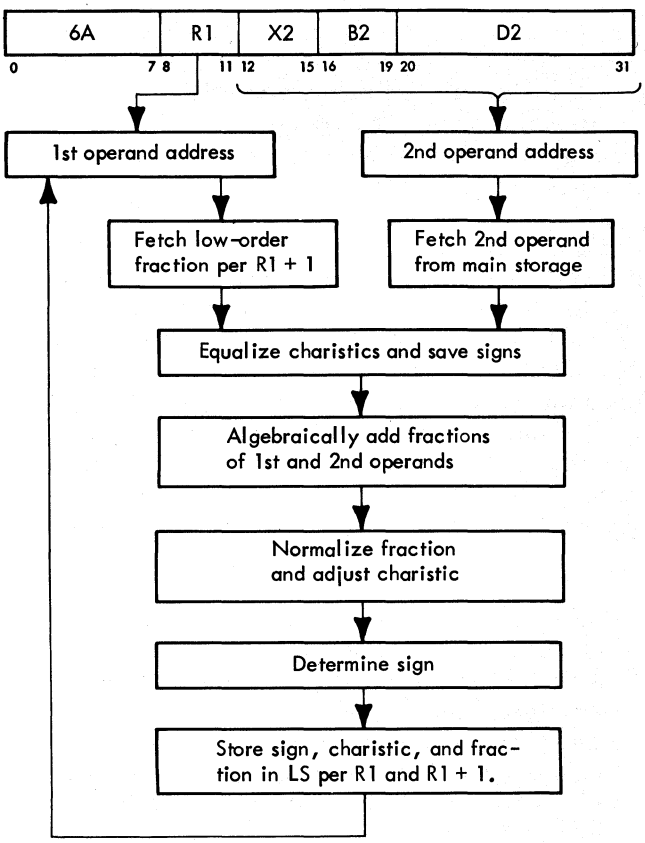
1. An additional operand fetch is needed.

2. The low-order portion of the floating-point registers is used.

3.6.6.1.4 AD (6A) - RX Long Operands

- Algebraically adds 2nd operand (from main storage) to 1st operand (per R1 and R1+1); normalized sum is placed in 1st operand location.
- RR format (see adjoining column).
- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.

The Add Normalized (AD) instruction algebraically adds the second operand (from main storage) to the first operand (specified by R1 and R1+1), and the normalized sum is placed in the first operand location. The CC is set according to hardware conditions.



The AD instruction is in the RX format with an op code of 6A and uses 64-bit operands.

The conditions at the beginning of the execution phase are:

1. The first operand (sign, characteristic, and high-order fraction) is in S and T.

2. The main storage effective address is in D.

3. The contents of A and B are unknown.

4. The AD instruction is in E.

Because the AD instruction uses long operands (64 bits) in the RX format, the low-order fraction of the first operand must be fetched from LS and the low-order fraction of the second operand must be fetched from main storage.

The sign, the characteristic, and the high-order fraction of the first operand are placed in A. The low-order fraction is fetched from LS per E(8-11) and routed to B via T and the parallel adder. The 64-bit second operand is fetched from main storage per D and placed in ST. The sign of the first operand is saved in STAT F; the sign of the second operand is saved in STAT C.

The high-order fraction is also placed in D, and addition is accomplished by algebraically adding the contents of DT to the contents of AB after characteristic equalization. The intermediate result is placed in AB and DT.

The first operand characteristic is subtracted from the second operand characteristic, and the characteristic difference and the signs determine the next operation via a 10-way branch (Figure 6039C, FEDM).

Except for long operands, operation of the AD instruction is similar to that of the AER instruction (paragraph 3.6.6.1.1). From this point, the microprogram is not concerned with instruction format.

3.6.6.2 Add Unnormalized

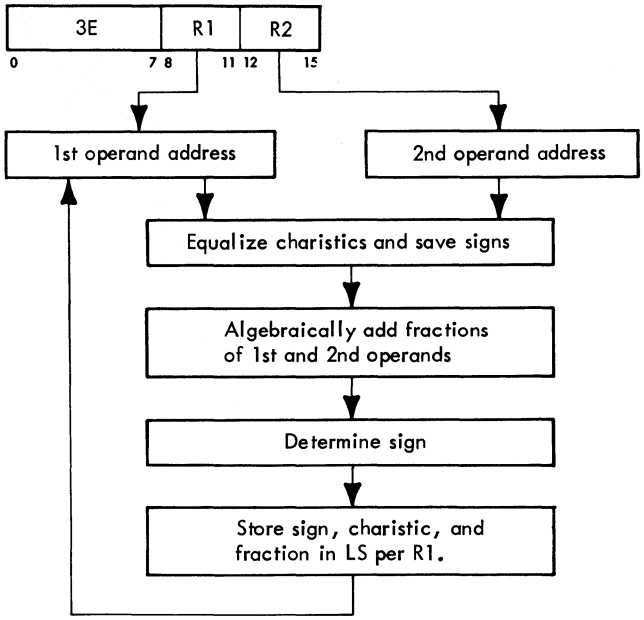
- Algebraically adds 2nd operand to 1st operand; unnormalized sum is placed in 1st operand location.
- 2nd operand location is unchanged.
- Instructions:
  AUR (3E) - RR Short Operands
  AU (7E) - RX Short Operands
  AWR (2E) - RR Long Operands
  AW (6E) - RX Long Operands

The Add Unnormalized instructions (AUR, AU, AWR, and AW) add the second operand to the first operand, and the unnormalized sum is placed in the first operand location. The CC is set according to hardware conditions at end-op time of the instruction being executed. Note that, when executing Add Unnormalized short operand instructions, the guard digit is not examined to determine the CC setting or checked for a significance condition.

With the exception that the intermediate result is not normalized, instruction execution is identical with that of Add Normalized instructions (AER, AE, ADR, and AD); therefore, references are made to the Add Normalized instructions when the Add Unnormalized instructions are discussed.

3.6.6.2.1 AUR (3E) - RR Short Operands

- Algebraically adds 2nd operand (per R2) to 1st operand (per R1); unnormalized sum is placed in 1st operand location.
- RR format:

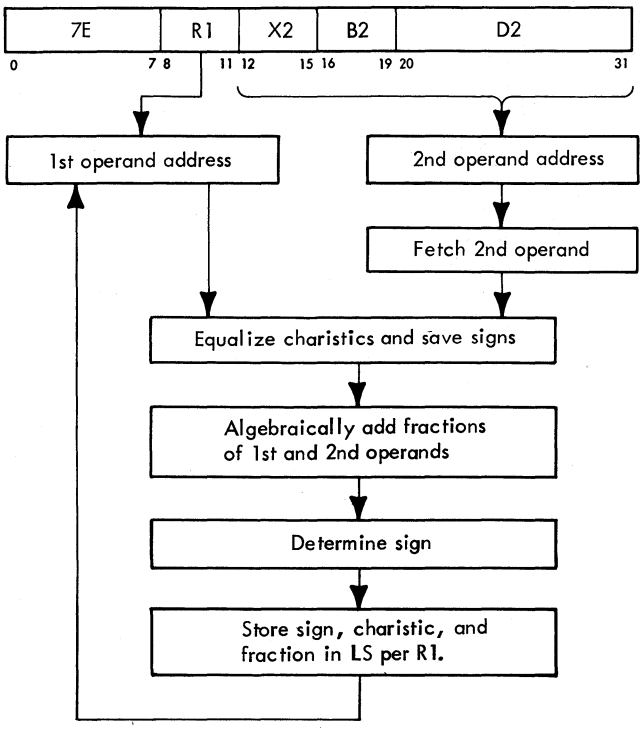


- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- Exponent underflow will not occur.
- For instruction execution, refer to AER description (paragraph 3.6.6.1.1).
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- Figure 6039, FEDM.

3.6.6.2.2 AU (7E) - RX Short Operands

- Algebraically adds 2nd operand (per R2) to 1st operand (per R1); unnormalized sum is placed in 1st operand location.
- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

- RX format:

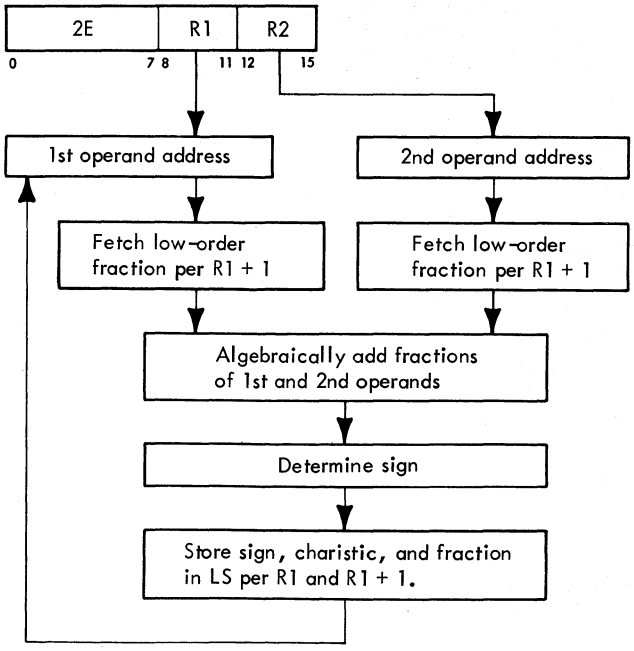


- Exponent underflow will not occur.
- For RX short operand fetch, refer to paragraph 3.6.6.1.2.
- For instruction execution, refer to paragraph 3.6.6.1.1.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- Figure 6039, FEDM.

3.6.6.2.3 AWR (2E) - RR Long Operands

- Algebraically adds 2nd operand (per R2 and R2 + 1) to 1st operand (per R1 and R1 + 1); unnormalized sum is placed in 1st operand location.
- Exponent underflow will not occur.
- For RR operand fetch, refer to paragraph 3.6.6.1.3.
- For instruction execution, refer to paragraph 3.6.6.1.1.

- RR format:

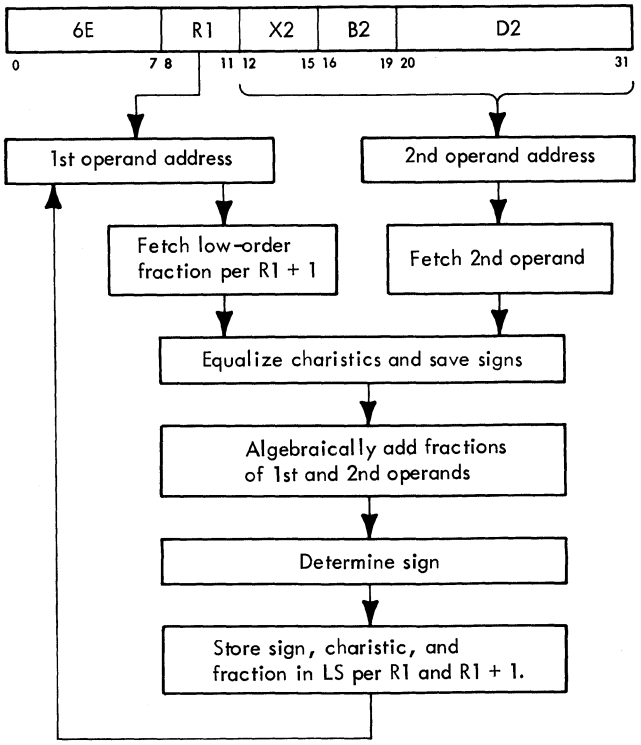


- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- Figure 6039, FEDM

3.6.6.2.4 AW (6E) - RX Long Operands

- Algebraically adds 2nd operand (from main storage) to 1st operand (per R1 and R1 + 1); unnormalized sum is placed in 1st operand location.
- Exponent underflow will not occur.
- For RX operand fetch, refer to paragraph 3.6.6.1.4.
- For instruction execution, refer to paragraph 3.6.6.1.1.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.

- Figure 6039, FEDM.

3.6.6.3 Subtract Normalized

- Algebraically subtracts 2nd operand from 1st operand; normalized difference is placed in 1st operand location.

- 2nd operand location is unchanged.

- Instructions:
  SER (3B) - RR Short Operands
  SE (7B) - RX Short Operands
  SDR (2B) - RR Long Operands
  SD (6B) - RX Long Operands

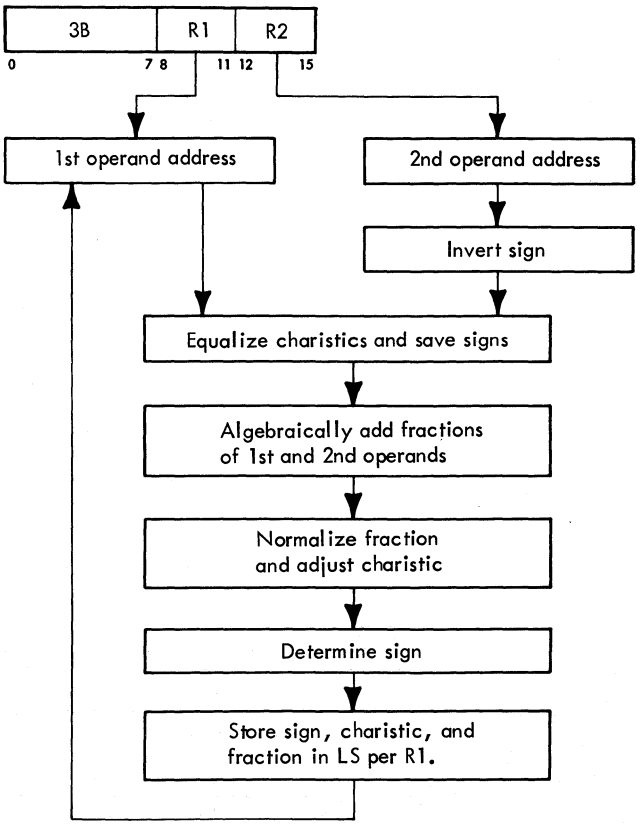
The Subtract Normalized instructions (SER, SE, SDR, and SD) subtract the second operand from the first operand, and the normalized difference is placed in the first operand location. The CC is set according to hardware conditions at end-op time.

When subtracting two numbers, the sign of the subtrahend (second operand) is inverted and the two numbers are algebraically added. In all subtract instructions (Subtract Normalized and Subtract Unnormalized), the sign of the second operand is complemented and saved in STAT C (Figure 6039B, FEDM), after which the algebraic subtraction is treated as an algebraic addition. Therefore, references are made to the add instructions to illustrate instruction execution.

3.6.6.3.1 SER (3B) - RR Short Operands

- Algebraically subtracts 2nd operand (per R2) from 1st operand (per R1); normalized difference is placed in 1st operand location.

- RR format:



- To subtract, change sign of 2nd operand and proceed as in addition.

- Conditions at start of execution:
  1st operand is in A, B, and D
  (24-bit fraction only)
  2nd operand is in S and T.
  Instruction is in E.

- For RR short operand fetch and instruction execution, refer to paragraph 3.6.6.1.1.

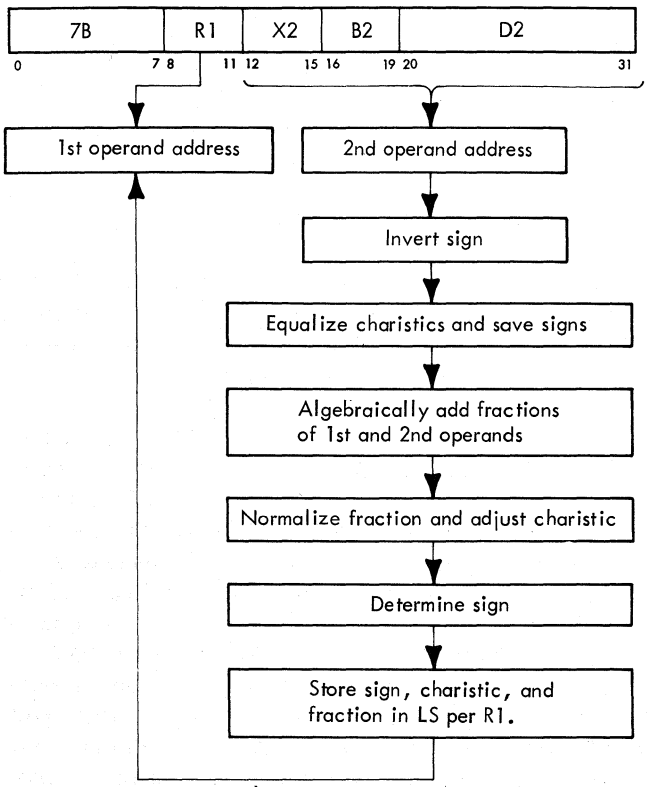
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero:
  CC = 1.
  Result fraction greater than zero:
  CC = 2.
  Result exponent overflows: CC = 3.

- Figure 6039, FEDM.

3.6.6.3.2 SE (7B) - RX Short Operands

- Algebraically subtracts 2nd operand (from main storage) from 1st operand (per R1); normalized difference is placed in 1st operand location.

- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

- To subtract, change sign of 2nd operand and proceed as in addition.

- For RX short operand fetch, refer to paragraph 3.6.6.1.2.

- For instruction execution, refer to paragraph 3.6.6.1.1.

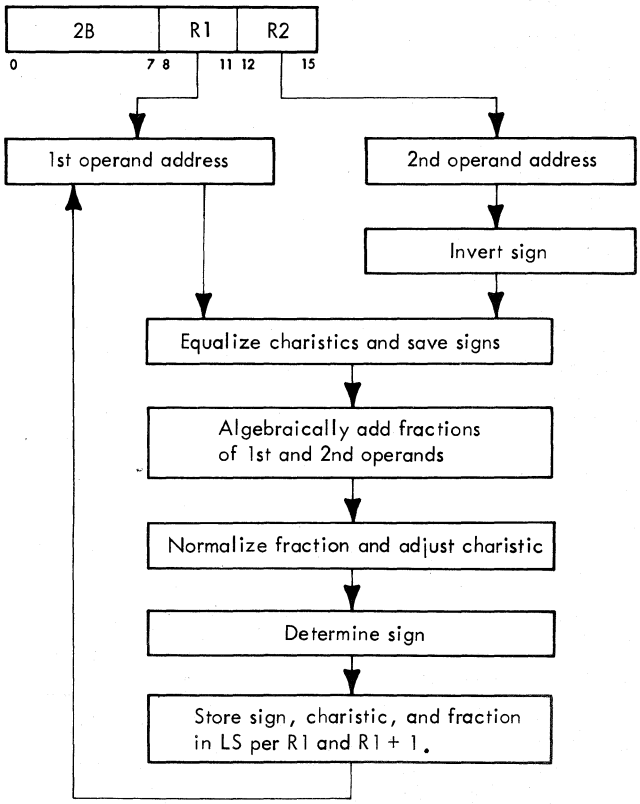
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero:
  CC = 1.
  Result fraction greater than zero:
  CC = 2.
  Result exponent overflows: CC = 3.

- Figure 6039, FEDM.

3.6.6.3.3 SDR (2B) - RR Long Operands

- Algebraically subtracts 2nd operand (per R2 and R2 + 1) from 1st operand (per R1 and R1 + 1); normalized difference is placed in 1st operand location.

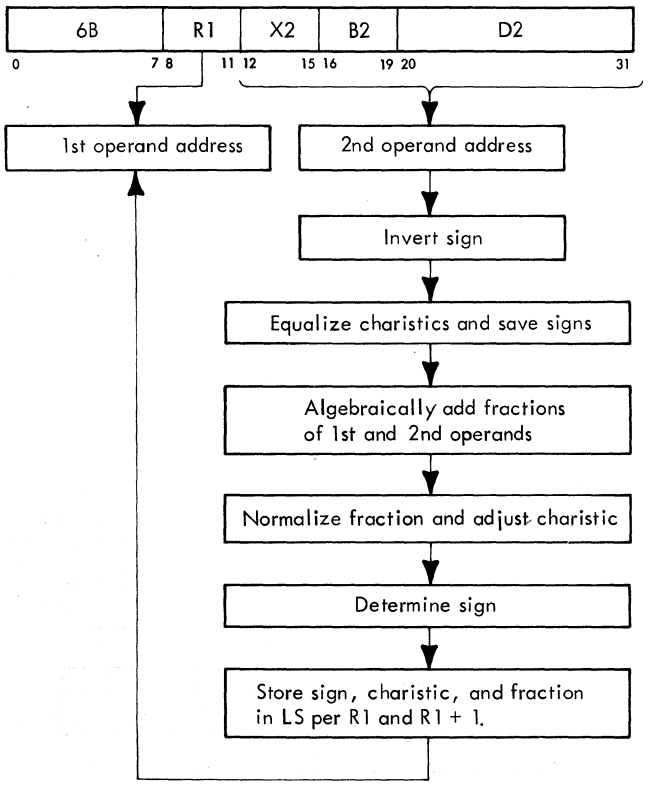
- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.
- To subtract, change sign of 2nd operand and proceed as in addition.
- For RR long operand fetch, refer to paragraph 3.6.6.1.3.
- For instruction execution, refer to paragraph 3.6.6.1.1.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- Figure 6039, FEDM.

3.6.6.3.4 SD (6B) - RX Long Operands

- Algebraically subtracts 2nd operand (from main storage) from 1st operand (per R1 and R1 + 1); normalized difference is placed in 1st operand location.
- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.
- To subtract, change sign of 2nd operand and proceed as in addition.
- For RX long operand fetch, refer to paragraph 3.6.6.1.4.
- For instruction execution, refer to paragraph 3.6.6.1.1.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- Figure 6039, FEDM.

3.6.6.4 Subtract Unnormalized

- Algebraically subtracts 2nd operand from 1st operand; unnormalized difference is placed in 1st operand location.
- 2nd operand location is unchanged.
- Instructions:
  SUR (3F) - RR Short Operands
  SU (7F) - RX Short Operands
  SWR (2F) - RR Long Operands
  SW (6F) - RX Long Operands

The Subtract Unnormalized instructions (SUR, SU, SWR, and SW) subtract the second operand from the first operand, and the unnormalized difference is placed in the first operand location.

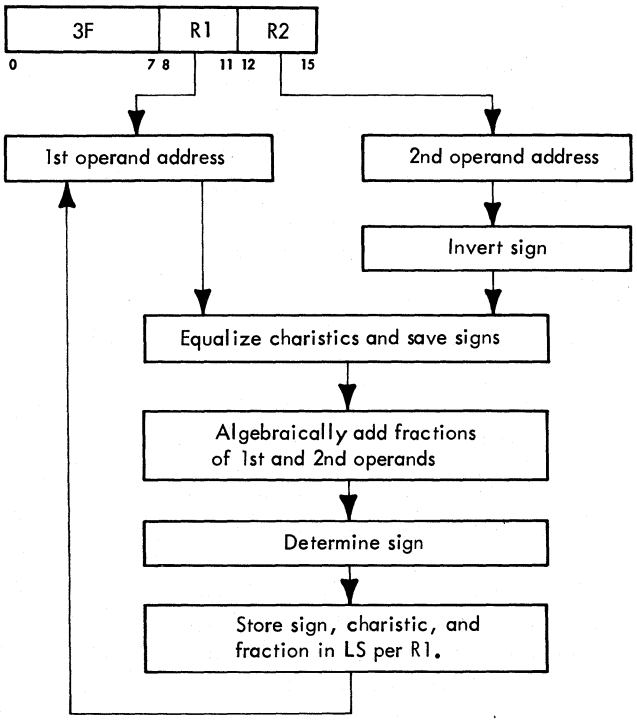
When subtracting two numbers, the sign of the subtrahend (second operand) is inverted and the two numbers are algebraically added. In all subtract instructions (Subtract Normalized and Subtract Unnormalized), the sign of the second operand is complemented and saved in STAT C. Algebraic addition is determined by STAT F and STAT C. Sign complementation is shown in Figure 6039B, FEDM. After the sign is complemented and saved, the algebraic subtraction is treated as an algebraic addition. The intermediate results of the Subtract Unnormalized instructions are not normalized. Other-

wise, the operation is the same as the Subtract Normalized instructions. Because the subtract instructions are similar to the add instructions, references are made to the add instructions to illustrate instruction execution.

Note that, when executing Subtract Unnormalized short operand instructions, the guard digit is not examined to determine the CC setting or checked for a significance condition.

3.6.6.4.1 SUR (3F) - RR Short Operands

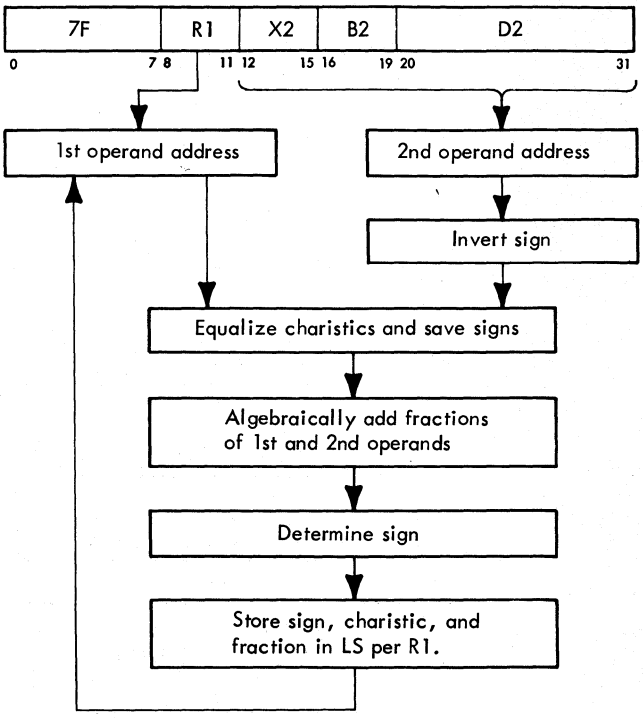
- Algebraically subtracts 2nd operand (per R2) from 1st operand (per R1); unnormalized difference is placed in 1st operand location.
- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.
- Exponent underflow will not occur.
- To subtract, change sign of 2nd operand and proceed as in addition.
- For RR short operand fetch and instruction execution, refer to paragraph 3.6.6.1.1.
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.
- Figure 6039, FEDM.

3.6.6.4.2 SU (7F) - RX Short Operands

- Algebraically subtracts 2nd operand (from main storage) from 1st operand (per R1); unnormalized difference is placed in 1st operand location.
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.
- To subtract, change sign of 2nd operand and proceed as in addition.

- For RX short operand fetch, refer to paragraph 3.6.6.1.2.

- For instruction execution, refer to paragraph 3.6.6.1.1.

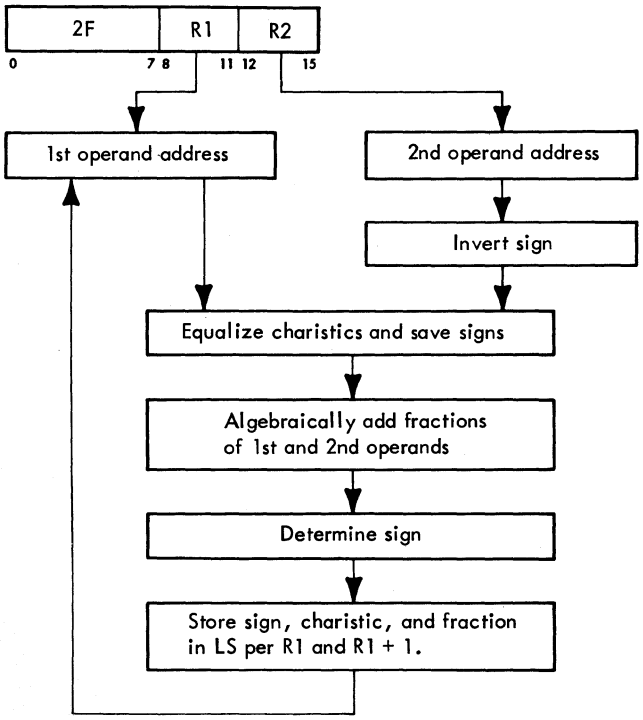
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.

- Figure 6039, FEDM.

3.6.6.4.3 SWR (2F) - RR Long Operands

- Algebraically subtracts 2nd operand (per R2 and R2 + 1) from 1st operand (per R1 and R1 + 1); unnormalized difference is placed in 1st operand location.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- To subtract, change sign of 2nd operand and proceed as in addition.

- For RR long operand fetch, refer to paragraph 3.6.6.1.3.

- For instruction execution, refer to paragraph 3.6.6.1.1.

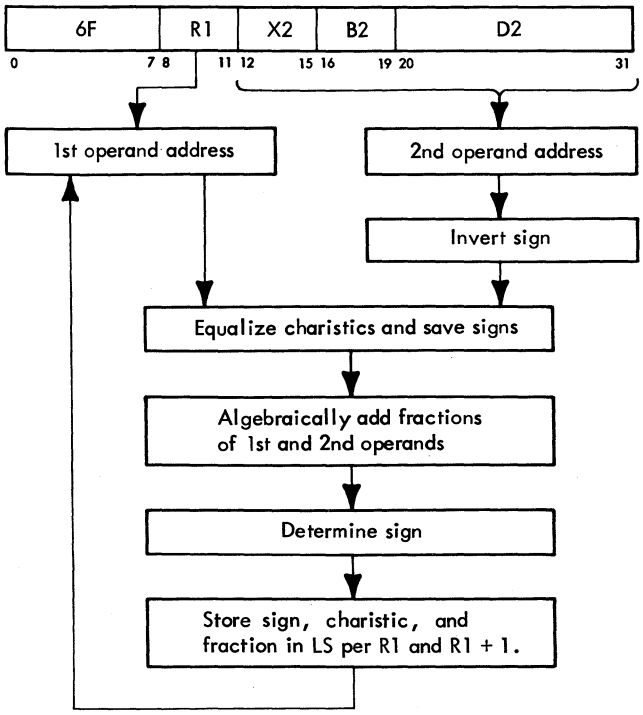
- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero: CC = 1.
  Result fraction greater than zero: CC = 2.
  Result exponent overflows: CC = 3.

- Figure 6039, FEDM.

3.6.6.4.4 SW (6F) - RX Long Operands

- Algebraically subtracts 2nd operand (from main storage) from 1st operand (per R1 and R1 + 1); unnormalized result is placed in 1st operand location.

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.

Effective address of 2nd operand
is in D.
Instruction is in E.

- To subtract, change sign of 2nd operand and proceed as in addition.

- For RX long operand fetch, refer to paragraph 3.6.6.1.4.

- For instruction execution, refer to paragraph 3.6.6.1.1.

- CC setting:
  Result fraction equals zero: CC = 0.
  Result fraction less than zero:
  CC = 1.
  Result fraction greater than zero:
  CC = 2.
  Result exponent overflows: CC = 3.

- Figure 6039, FEDM.

3.6.6.5 Compare

- Algebraically compares 1st operand with 2nd operand; CC indicates result.

- Operand locations are unchanged.

- Instructions:
  CER (39) - RR Short Operands
  CE (79) - RX Short Operands
  CDR (29) - RR Long Operands
  CD (69) - RX Long Operands

The Compare instructions (CER, CE, CDR, and CD) algebraically compare the first operand with the second operand; the CC indicates that the first operand is equal to, less than, or greater than the second operand.

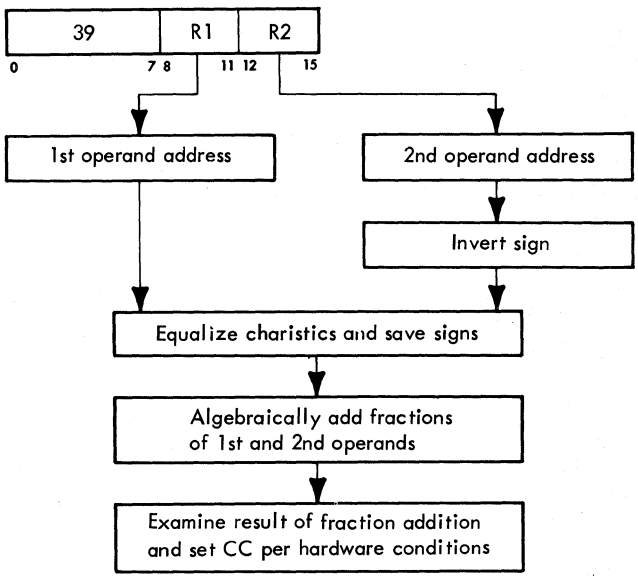
In short operand instructions, the low-order halves of the floating-point LS registers are ignored. Neither of the operand locations is changed as a result of the compare instructions.

Comparison is algebraic, taking into account the sign, fraction, and characteristic of each operand. An exponent inequality is not decisive for magnitude determination since the fractions may have different numbers of leading zeros. Equality is established by following the rules for floating-point subtraction. The intermediate result is not normalized or stored. The CC is set per hardware conditions at end-op time. When the intermediate result, including a possible guard digit, is 0, the operands are equal. Numbers with zero fractions compare equal even when they differ in sign or characteristic. Exponent overflow, exponent underflow, or lost significance cannot occur.

3.6.6.5.1 CER (39) - RR Short Operands

- Algebraically compares 1st operand (per R1) with 2nd operand (per R2); CC indicates result.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D
  (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- Exponent underflow, exponent overflow, or significance cannot occur.

- For RR short operand fetch and instruction execution, refer to paragraph 3.6.6.1.1.

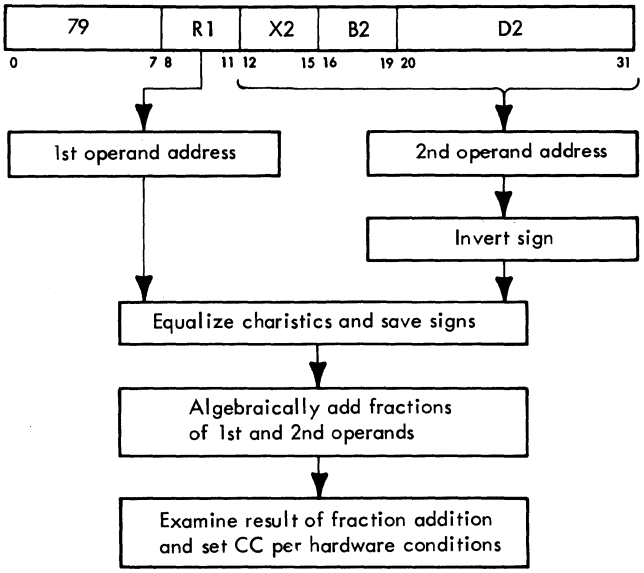
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

- Figure 6039, FEDM.

3.6.6.5.2 CE (79) - RX Short Operands

- Algebraically compares 1st operand (per R1) with 2nd operand (from main storage); CC indicates result.

- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

- Exponent underflow, exponent overflow, or significance cannot occur.

- For RX short operand fetch, refer to paragraph 3.6.6.1.2.

- For instruction execution, refer to paragraph 3.6.6.1.1.

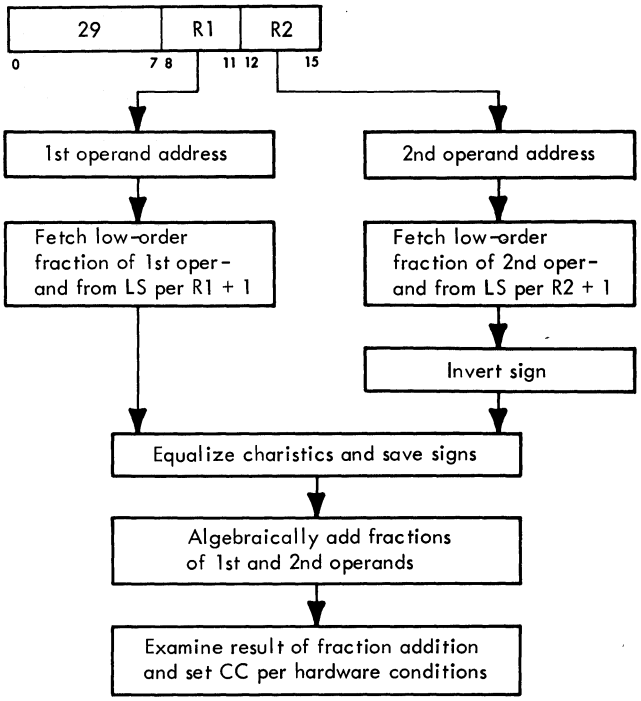
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

- Figure 6039, FEDM.

3.6.6.5.3 CDR (29) - RR Long Operands

- Algebraically compares 1st operand (per R1 and R1 + 1) with 2nd operand (per R2 and R2 + 1); CC indicates result.

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- Exponent underflow, exponent overflow, or significance cannot occur.

- For RR long operand fetch, refer to paragraph 3.6.6.1.3.

- For instruction execution, refer to paragraph 3.6.6.1.1.

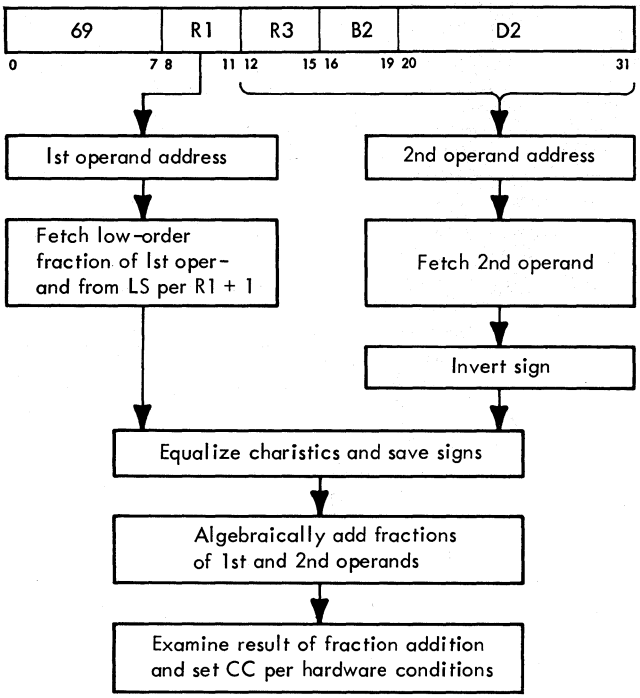
- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

- Figure 6039, FEDM.

3.6.6.5.4 CD (69) - RX Long Operands

- Algebraically compares 1st operand (per R1 and R1 + 1) with 2nd operand (from main storage); CC indicates result.

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

- Exponent underflow, exponent overflow, or significance cannot occur.

- For RX long operand fetch, refer to paragraph 3.6.6.1.4.

- For instruction execution, refer to paragraph 3.6.6.1.1.

- CC setting:
  Operands are equal: CC = 0.
  1st operand is less than 2nd operand: CC = 1.
  1st operand is greater than 2nd operand: CC = 2.

- Figure 6039, FEDM.

3.6.7 HALVE

- Divides 2nd operand by 2, and quotient is placed in 1st operand location.

- To halve, shift fraction R1 bit position.

- Instructions:
  HER (34) - RR Short Operands
  HDR (24) - RR Long Operands

The Halve instructions (HER and HDR) divide the second operand by 2, and the quotient is placed in the first operand location. The Halve instructions are in the RR format with short and long operand options available. In the HER instruction, the low-order half of the result register remains unchanged.

After the complete second operand is in ST, the sign and the characteristic are saved in F, and the high-order fraction (24 bits) is placed in D (long operands). Shifting the fraction R1 bit position divides the operand by 2. The sign and the characteristic are unchanged. After the R1 shift is completed, the sign, characteristic, and fraction are stored in LS per the R1 field.

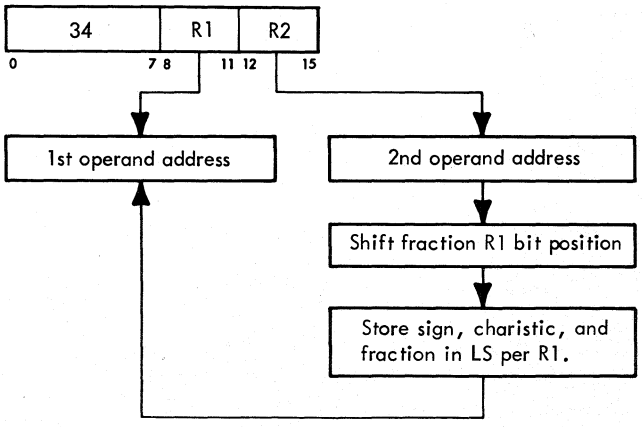
Because the R1 shift cannot be accomplished directly, two machine cycles are necessary. The R1 shift is accomplished by shifting the fraction L1 to the parallel adder and performing an R4 shift to PAL, thus yielding an effective R3 shift. The fraction, shifted R3, is placed in AB. An L2 shift occurs when the fraction is routed to DT, resulting in an R1 shift and thereby dividing the fraction by 2. The sign, characteristic, and fraction are stored in LS, completing instruction execution.

The halve operation differs from the divide operation in that 2 is the divisor, prenormalization or postnormalization does not occur, and a zero-fraction test does not occur.

3.6.7.1 HER (34) - RR Short Operands

- Divides 2nd operand (per R2) by 2; quotient is placed in 1st operand location (per R1).

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

- To divide by 2, shift fraction R1 bit position.

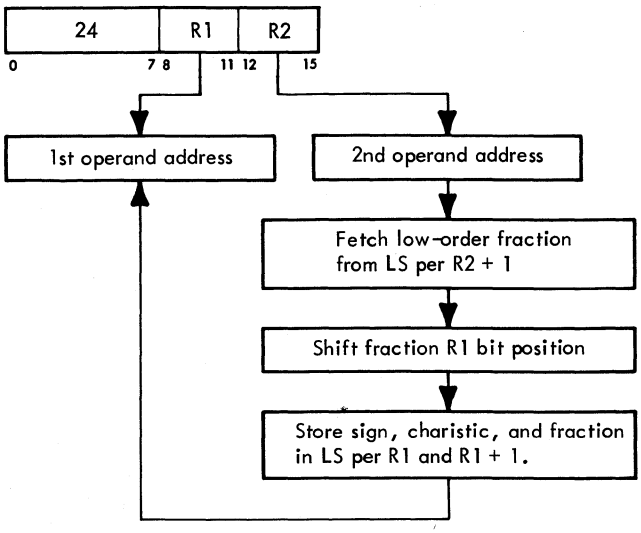
- Sign and characteristic remain unchanged.

- Figure 6040, FEDM.

3.6.7.2 HDR (24) - RR Long Operands

- Divides 2nd operand (per R2 and R2 + 1) by 2; quotient is placed in 1st operand location (per R1 and R1 + 1).

- RR format:



- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

- To divide by 2, shift fraction R1 bit position.

- Sign and characteristic remain unchanged.

- Figure 6041, FEDM.

3.6.8 MULTIPLY

- Multiplies 1st operand (multiplier) by 2nd operand (multiplicand); normalized product is placed in 1st operand location.

- Product is 64 bits for both short and long operand instructions.

- Characteristics added and 64 subtracted to obtain intermediate characteristics.

- Operands are always prenormalized before multiplying.

- Product is always normalized before storing.

- Sign of product determined algebraically.

- Instructions:
  MER (3C) - RR Short Operands
  ME (7C) - RX Short Operands
  MDR (2C) - RR Long Operands
  MD (6C) - RX Long Operands

Multiplication of two floating-point numbers consists of a characteristic addition and a fraction multiplication. The sum of the characteristics less 64 is used as the characteristic of an intermediate product. The sign of the product is determined algebraically.

If necessary, the product fraction is normalized by prenormalizing the operands and postnormalizing the intermediate product. The intermediate product characteristic is reduced by the number of left shifts. For long operands, the intermediate product fraction is truncated before left-shifting. For short operands (6-digit fractions), the product fraction has the full 14 hexadecimal digits of the long format, and the two low-order hexadecimal fraction digits are accordingly always zeros. The two low-order hexadecimal fraction digits are zeros in short operand instructions because it is possible to obtain a maximum of 12 nonzero hexadecimal digits when multiplying two 6-digit numbers. The maximum number of digits in the product will not exceed the sum of the operand digits available. Therefore, since 14 product digits are available, the two low-order hexadecimal digits of short operand products are always zeros.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is terminated, and a program interruption occurs. The overflow interruption condition does not occur for a

partial product characteristic exceeding 127 when the final characteristic is brought within range through normalization.

When exponent underflow occurs, the final product characteristic is less than zero. The sign, characteristic, and fraction are made zero, and a program interruption occurs if the corresponding mask bit is a 1. Underflow is not signalled when the characteristic of an operand becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 14 result fraction digits are zero, the product sign and characteristic are made zero, yielding a true zero result, exponent underflow is not signalled, and no interruption is taken. The program interruption for lost significance is never taken for multiplication.

When two floating-point numbers are multiplied, the characteristics must be added to yield the final characteristic value of the product. Since excess 64 notation is used, 64 must be subtracted from the characteristic sum because the characteristic value is in excess 128 [(C1 + 64) + (C2 + 64) = C1 + C2 + 128] after characteristic addition. When 64 is subtracted, the result is returned to excess 64 notation (C1 + C2 + 128 - 64 = C1 + C2 + 64).

Characteristic computations are accomplished in the serial adder. The sign and characteristic data paths are shown in B of Figure 6042A, FEDM. The signs are saved in STAT C and STAT F. To add characteristics, the first operand characteristic is gated to SAA(1-7) from AB per the ABC, and the second operand characteristic is gated to SAB(1-7) from ST per the STC. The characteristic sum is routed to the SAL's and to F. The characteristic carry is saved in STAT D and F(0). 64 is subtracted from the characteristic by adding the 2's complement of 64 to the sum in F.

In floating-point multiply, the operands are normalized before multiplication begins. Prenormalization is necessary to increase product precision. By prenormalizing the operands, a maximum of one postnormalization cycle is necessary. Prenormalization and postnormalization are accomplished by shifting the fraction left one hexadecimal digit and subtracting 1 from the characteristic value for each left shift. To subtract 1 from the characteristic, the 2's complement of 1 is added to the value in F (B of Figure 6042A, FEDM).

The product for both short and long operand multiply instructions is 64 bits in length. Note that, if the fraction is not prenormalized, the truncated product may result in loss of the low-order fraction bits, and in a false zero product. This result would be true in long operand instructions because 56 low-order bits of the product are lost when executing the multiply algorithm. To prevent a false zero, the product for long operand instructions would need to be 120 bits in length.

The basic multiply algorithm for the floating-point fraction multiply is similar in operation to fixed-point multiply. A of Figure 6042A, FEDM, is a basic flow chart of the floating-point multiply operation. The signs are saved, the characteristic is determined, and the operands are prenormalized before multiplying the fractions. A multiple of the multiplicand is then selected. This multiple is added to a value to form a partial product (PP). Multiples of the multiplicand are continually selected until E(12-15) indicates that all multiples have been selected. The intermediate product is contained in AB(4-67) and was derived by adding the selected multiple to the PP values. The intermediate product is postnormalized, its characteristic is adjusted, and the signs are determined. The final product is stored in LS per R1 and R1 + 1.

Selection of the multiplicand multiple is shown in A of Figure 6042B, FEDM. The contents of S (multiplier) are sent to the multiplier (MPR) bus, where the byte to be operated on is selected by means of E(12,13). Once the byte has been obtained, bits of that byte are selected to develop the multiple selection bits M1 and M2, which, with the TX trigger, gate the correct multiple value of the multiplicand to the PAA. E(14,15) selects the bits from the byte in S. During the operation, the contents in E are sequentially reduced by 1 to select the next multiple. Considering two multiplier bits at a time (M1 and M2) and the TX trigger, the multiply algorithm that follows is controlled by the select-multiple (SEL-MPL-*E3) micro-order and hardware conditions (the numbers in parentheses relate to the Multiple Selection Bits in Table 3-15):

1. Nothing is added to the PP, and the PP is shifted R2 bit positions.

2. Add in the multiplicand, and shift the PP R2 bit positions.

3. The multiplicand is shifted L1 (effective multiplication by 2), added to the PP located in AB, and shifted R2 bit positions.

4. Four times the multiplicand should be added to the PP in AB; however, minus DT is added

TABLE 3-15. VALUE OF MULTIPLE DETERMINED BY MULTIPLE SELECTION BITS (FLOATING-POINT)

	Multiple Selection Bits		TX Trigger	DT Register Times Value Indicated (Add to Partial Product in AB)	Set TX Trigger
	M1	M2			
(1)**	0	0	0	0 X DT	No
(2)	0	1	0	1 X DT	No
(3)	1	0	0	2 X DT	No
(4)	1	1	0	-1 x DT (2's Complement)	Yes
(5)	0	0	1	1 X DT	No*
(6)	0	1	1	2 X DT	No
(7)	1	0	1	-1 x DT (2's Complement)	Yes
(8)	1	1	1	0 X DT	Yes

* Used on last multiple select if TX trigger is set.

** Numbers in parentheses used for reference purposes in discussion.

TABLE 3-16. MULTIPLE SELECTION BITS, FLOATING-POINT MULTIPLY

S Register*	MPR Bus	E Register		Multiple Selection Bits	
		14	15	M1	M2
0, 8, 16, 24	0	0	0	1	0
1, 9, 17, 25	1	0	0	0	1
2, 10, 18, 26	2	0	1	1	0
3, 11, 19, 27	3	0	1	0	1
4, 12, 20, 28	4	1	0	1	0
5, 13, 21, 29	5	1	0	0	1
6, 14, 22, 30	6	1	1	1	0
7, 15, 23, 31	7	1	1	0	1

* If any one of these bits is active, the MPR bus is active. S bits are selected according to the value of E(12,13): if E(12,13) = 00, select S(0-7); if 01, select S(8-15); if 10, select S(16-23); if 11, select S(24-31).

and the TX trigger is set to remember this fact. The PP is shifted R2 bit positions.

5. Add in the multiplicand, and shift the PP R2 bit positions.

6. Add in twice the multiplicand, and shift the PP R2 bit positions.

7. Subtract the multiplicand from the PP, and shift the PP R2 bit positions. Set the TX trigger.

8. Add zero to the PP, and shift the PP R2 bit positions. Set the TX trigger.

The M1 and M2 bits are considered to be the two selected bits in S (multiplier). These multiplier bits are selected by E(12-15) as shown in A of Figure 6042B, FEDM. The byte in S is selected by E(12,13); the bits within the byte are selected by E(14,15). Initially, E(12-15) is set to 15 (1111). Therefore, the fourth byte (S register byte 3) and bits 30 and 31 are selected (A of Figure 6042B, FEDM). Table 3-16 shows which S bits are gated to the MPR bus per E(12-15). The value of M1 and M2 and the TX trigger determine the gating of DT to the PAA as previously defined (Table 3-15).

Once the multiplicand multiple has been selected, the PP is derived. The method of deriving the PP is shown in B of Figure 6042B, FEDM. The multiple is placed in PAA and added to the value generated from AB, thus forming a new PP. If this is the first multiple selected from DT, it is added to a value of zero. If the multiple is some multiple other than the first, the multiple selected from DT is added to the PP that was developed in previous cycles. Once the multiple and the PP have been added, the result is shifted R4 bit positions. AB(4-65) is then shifted L2 bit positions, thus placing the PP in PAB, where it is added to another multiple, forming a new PP. For each PP derived, an effective R2 shift occurs. Two low-order bits of the PP are shifted out of AB(66,67) and lost on each effective R2 shift. Operations continue in this manner until the intermediate product is obtained.

After decoding the last S bits, the TX trigger is checked. If the TX trigger is set, one additional termination cycle is necessary to obtain the final intermediate product. If the TX trigger is not set, no extra cycle is necessary. After the fraction intermediate product is obtained, the fraction is normalized (postnormalization), the characteristic is adjusted, the sign is determined, and the final 64-bit product is stored in LS as specified by R1 and R1 + 1 [located in E(8-11)].

To illustrate the multiply operation, assume that the following fractions are to be multiplied:

$0.24_{10} \times 0.15_{10} = 0.0360_{10}$ or $0.18_{16} \times .F_{16} = 0.168_{16}$

The operands in machine language become:

0 1000000.00011000 0↔0 X 0 1000000.11110000 0↔0

In hexadecimal notation, the example becomes:

+40.18X +40.F = +40.168

Further, assume that a short operand instruction in the RR format is to be executed. For this discussion, assume that 0.15 (decimal) is the multiplier (first operand) and 0.24 (decimal) is the multiplicand (second operand). At the start of execution, the instruction is contained in E; the first operand is in A, B, and D (this value is not used and is subsequently destroyed), and the second operand is in S and T.

As previously described, the signs are saved, the characteristics are determined, and the fractions are prenormalized before beginning the multiply algorithm. The value in E(12-15) selects the correct multiple of the multiplicand. Initially, E(12-15) is set to 15 and sequentially reduced by 1 during the operation. Before multiplication of the fraction begins, the first operand fraction is transferred to S, the second operand is transferred to D, and B and T are reset.

The first multiple of the multiplicand is determined by checking E(12-15), which presently contains 1111 (binary). Using Table 3-16 to determine the value of the multiple, it is found that the first byte selected in the multiplier is S(24-31). At this time, S(24-31) equals 0's. Since the multiple is determined by checking two bits of the multiplier at a time, it must now be determined by checking E(14, 15). At this point, all bits of the MPR bus are inactive; therefore, the first PP placed in AB equals zero. The sequential reduction of E(12-15) continues until the value equals 0101, at which time the PP in AB equals zero.

Referring to Table 3-16, when E(12-15) equals 0101, byte 1 in S [S(8-15)] is selected per E(12,13), and S(10,11) is selected per E(14,15). These selected bits determine the multiple (M1,M2) of the multiplicand to be added to the PP in AB. Because S(10,11) is equal to 11 (binary), the 2's complement of DT is gated to PAA. The contents of AB are shifted L2 at this time (AB equals zero) and gated to PAB. The output of the PA is shifted R4 to the PAL's. The contents of the PAL's are gated to AB, forming a new PP. The contents of AB(4-67) now contain 1111.1111111010000↔0. Note that PA(4) is propagated into PAL(4-7) by the R→micro-order. Because S(10,11) was equal to 11, the TX trigger is set (Table 3-15). Remember that E(12-15) is decremented after each multiple selection. Referring to Table 3-15, it can be seen that 0's are added to PAA on the next multiple selection (0 X DT). During this select multiple, the contents of AB are shifted L2 to PAB and the next PP is shifted R4 to the PAL's and AB(4-67), thus yielding an effective R2 shift. The new PP in AB(4-67) becomes 1111.111111111010000↔0.

At this point, all multiples of the multiplicand have been selected. If the TX trigger is not set, the PP in AB becomes the intermediate result. In this example, however, the TX trigger was set because the multiplier bits equalled 11 and the TX trigger was previously set (Table 3-15). Therefore, DT must be added to the PP in AB. The contents of AB are shifted L2 to PAB and added to DT. No R4 shift from the PA to the PAL's occurs at this time. The intermediate product is transferred from the PAL's to AB(4-67) and DT. The value of the intermediate product is .0001 0110 1000 0↔0 (0.168_{16}). In this example, normalization is not necessary. The sign, characteristic, and fraction are stored in LS per R1 and R1 + 1. An end-op cycle completes the operation.

If the integers were preceded by 0's in this example, prenormalization of the operands would occur before executing the multiple algorithm.

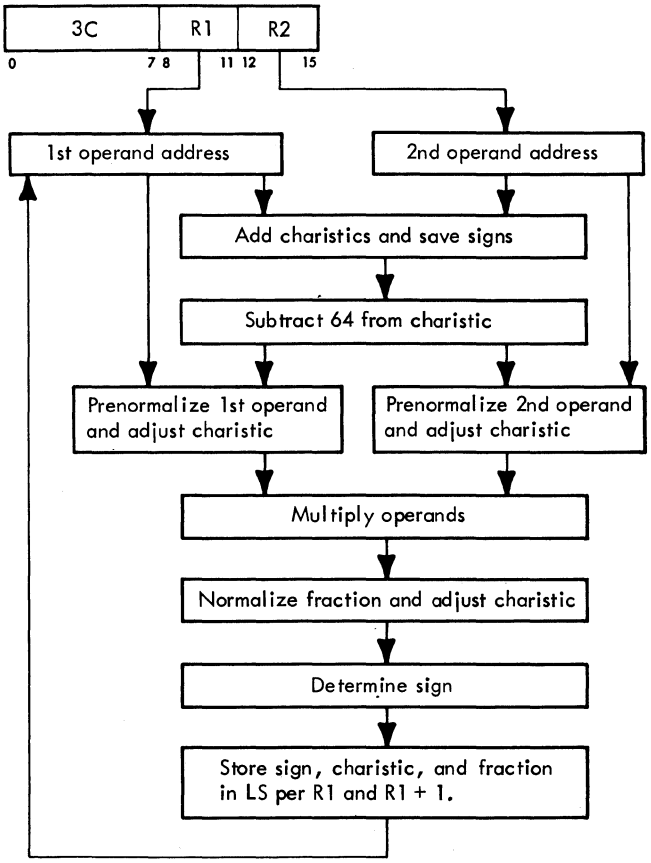
3.6.8.1 MER (3C) - RR Short Operands

- Multiplies 1st operand (multiplier per R1) by 2nd operand (multiplicand per R2); normalized product is placed in 1st operand location (per R1).

- RR format (see format on next page).

- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Multiply, MER, instruction multiplies the first operand (multiplier specified by R1) by the second operand (multiplicand specified by R2), and the normalized product is placed in the first operand location.



The MER instruction is in the RR format with an op code of 3C. This instruction uses 32-bit operands, and the final product is 64 bits in length. The entire 64-bit product is stored in LS as specified by R1 and R1 + 1.

The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only)

2. The second operand is in S and T.

3. The STC contains a value of 4.

4. The MER instruction is in E.

For the following MER instruction analysis, refer to Figure 6042, FEDM.

The second operand fraction (multiplicand) is transferred from T to D. The first operand sign is saved in STAT F; the second operand sign is saved in STAT C. The characteristics are added, yielding an excess 128 characteristic, and this sum is placed in F. SA(0) is saved in STAT D and placed in F(0). B and T are reset by transferring 0's from PAL(32-63) to B and T. The first operand (multiplier) is fetched from LS and placed in S for the select multiple function. The constant 15 is placed in E(12-15) for selecting the two multiple bits from S (Figure 6042E, FEDM). The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. However, the operands are tested to determine, via a 4-way branch, whether prenormalization is necessary. The 4-way branch tests for the following conditions:

1. First and second operands are normalized.

2. First operand is normalized, and the second operand is unnormalized.

3. First operand is unnormalized, and the second operand is normalized.

4. First and second operands are unnormalized.

Assume that both operands need normalizing. The second operand is normalized by shifting the fraction in DT L1 hexadecimal digit and subtracting 1 from the characteristic. Left-shifting continues until the second operand fraction is normalized.

After the fraction of the second operand is normalized, the first operand (multiplier) is transferred from S to B. The contents of T (0's for short operands) are saved in the LSWR (Figure 6042E, FEDM). Normalization of the first operand is shown in Figure 6042F, FEDM. Normalization is accomplished by shifting the contents of AB L1 hexadecimal digit and subtracting 1 from the characteristic. B is reset during the first shift. Left-shifting continues until the fraction of the first operand is normalized. On each left shift, the shifted low-order fraction (0's) is stored in LS per the R1 field [E(8-11)]. S is loaded with 0's for short operand instructions. The high-order fraction is transferrred from A and stored in LS per E(8-11). S is then loaded with the short operand multiplier. T is reset, and DT becomes a 56-bit multiplicand (second operand) (Figure 6042F, FEDM).

Since the characteristic is in excess 128, 64 is subtracted from F so that the excess 64 rule applies. AB is reset, and the multiply algorithm begins (Figure 6042E, FEDM). The multiply algorithm is discussed in paragraph 3.6.8. The multiply function enters at E in Figure 6042G, FEDM. A SEL-MPL*E3 micro-order is executed, and 1 is subtracted from E(12-15) with each machine cycle. When E(12-15) equals 0100, all 12 pairs of mul-

tiples have been selected. Since the TX trigger may have been set on the previous multiple selection, a select last multiple is necessary to add in the multiplicand to obtain the correct product.

Since the operands were normalized before multiplying, a maximum of one left shift is necessary to normalize the intermediate product fraction (Figure 6042G, FEDM). If A(8-11) equals zero, one left shift of the intermediate product fraction is necessary. When the left shift occurs, 1 is subtracted from the characteristic. The characteristic value of the final product is located in SAL(1-7) and F(1-7). The sign is determined algebraically; the sign, characteristic, and 56-bit fraction are stored in LS per the R1 field and R1 + 1.

If SAL(0) equals 1, an exponent overflow or exponent underflow condition exists and the product is incorrect. Zeros are stored in the first operand location if an exponent underflow has occurred. A program interruption occurs on all exponent overflows and on exponent underflows if masked on. If SAL(0) equals 0, the stored product is correct. An end-op cycle completes instruction execution.

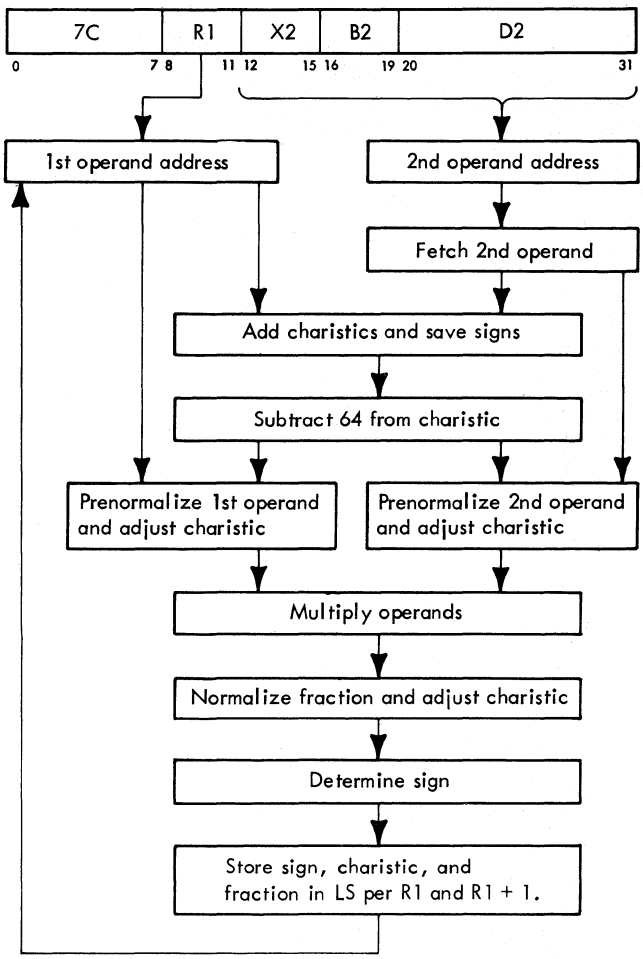
3.6.8.2 ME (7C) - RX Short Operands

- Multiplies 1st operand (multiplier per R1) by 2nd operand (multiplicand from main storage); normalized product is placed in 1st operand location (per R1).
- RX format (see adjoining column).
- Conditions at start of execution
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

The Multiply, ME, instruction multiplies the first operand (multiplier specified by R1) by the second operand (multiplicand from main storage), and the normalized product is placed in the first operand location.

The ME instruction is in the RX format with an op code of 7C. This instruction uses 32-bit operands, and the final product is 64 bits in length. The entire product is stored in LS as specified by R1 and R1 + 1.

The conditions at the beginning of the execution phase are:



1. The first operand is in S and T.
2. The effective address of the second operand is in D.
3. The ME instruction is in E.

For the following ME instruction analysis, refer to Figure 6042, FEDM.

The first operand (multiplier) is placed in A, the constant 15 is placed in E(12-15), and the STC is set to 4. Assume that the first operand is not normalized and that the second operand is normalized (Figure 6042D, FEDM). The second operand is fetched from main storage (per the effective address in D) and placed in ST. If D(21) equals 1, the second operand (multiplicand) is in T; conversely, if D(21) equals 0, the second operand is in S and must be placed in T. The sign of the first operand is saved in STAT F; the sign of the second operand, in STAT C. The characteristics are added, and the sum is placed in F. SAL(0) is saved in STAT D and F(0). The fraction of the second operand is placed in D. B and T are reset, the first operand is placed in S, and 15 is

loaded in E(12-15) (Figure 6042E, FEDM). A 4-way branch determines the next operation. From this point, operation is similar to that of the MER instruction (paragraph 3.6.8.1). Refer to paragraph 3.6.8 for a discussion of the multiply algorithm.

If the first operand was normalized, the second operand (multiplicand) from main storage is placed in AB (Figure 6042D, FEDM). T and the STC are reset. The transfer of the second operand fraction to D is determined by D(21). If D(21) equals a 1, the second operand from B is transferred to A and D. If D(21) equals a 0, the second operand in A is transferred to D. Note that the sign of the first operand is saved in STAT C, and the sign of the second operand is saved in STAT F. The characteristics are added, and the sum is saved in F. The characteristic carry is saved in STAT D and F(0).

Because the first operand was initially normalized, the ROS microprogram assumes that the second operand is also normalized. Therefore, the first PP is computed. If the second operand needs normalizing, however, the operands and the constant 15 in E(12-15) are restored and the second operand is normalized before multiplying (Figure 6042D and E, FEDM). Once both operands are normalized, the operands are multiplied and the results stored. Refer to paragraphs 3.6.8.1 and 3.6.8 for a discussion of multiply instruction execution.

3.6.8.3 MDR (2C) - RR Long Operands

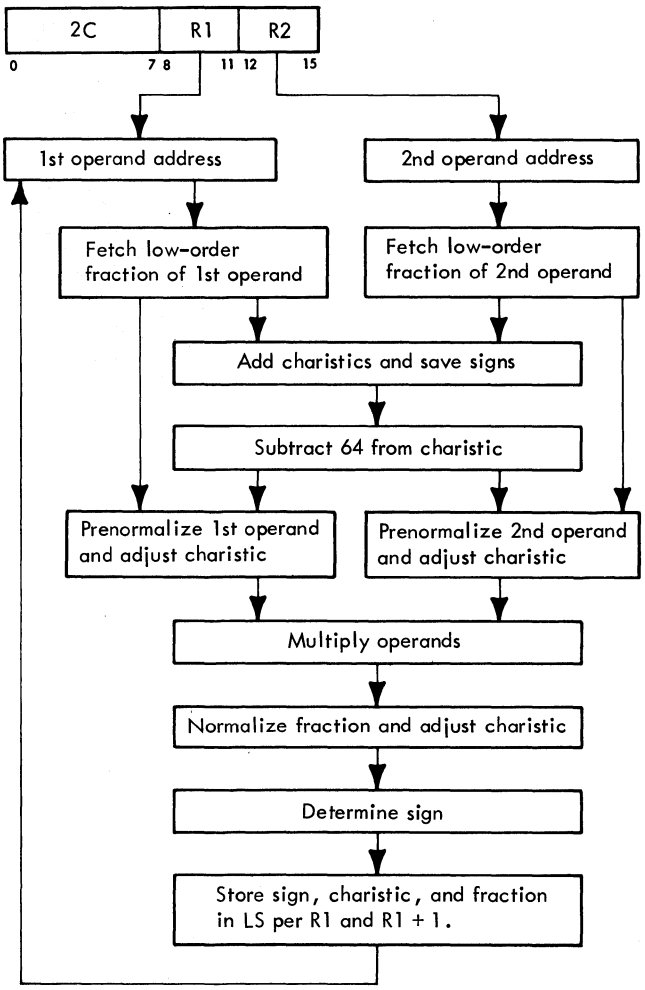
- Multiplies 1st operand (multiplier per R1 and R1 + 1) by 2nd operand (multiplicand per R2 and R2 + 1); normalized product is placed in 1st operand location (per R1 and R1 + 1).

- RR format (see adjoining column).

- Conditions at start of execution:
  32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  32 bits of 2nd operand are in S and T.
  Instruction is in E.

The Multiply, MDR, instruction multiplies the first operand (multiplier specified by R1 and R1 + 1) by the second operand (multiplicand specified by R2 and R2 + 1), and the normalized product is placed in the first operand location.

The MDR instruction is in the RR format with an op code of 2C. This instruction uses 64-bit



operands, and the final 64-bit product is stored in LS as specified by R1 and R1 + 1.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).

2. 32 bits of the second operand are in S and T.

3. The STC contains a value of 4.

4. The MDR instruction is in E.

For the following MDR instruction analysis, refer to Figure 6042, FEDM.

The second operand (multiplicand) high-order fraction is transferred from T to D. The low-order fraction of the first operand (multiplier) is placed in S, and the low-order fraction of the second operand is placed in T. DT contains the multiplicand fraction, and S contains the low-order fraction of the

multiplier. The signs are saved in STAT C and STAT F. The characteristics are added, and the sum is placed in F. The characteristic carry is saved in STAT D and also placed in F(0). The constant 15 is placed in E(12-15) for selecting the two multiple bits located in S (Figure 6042E, FEDM).

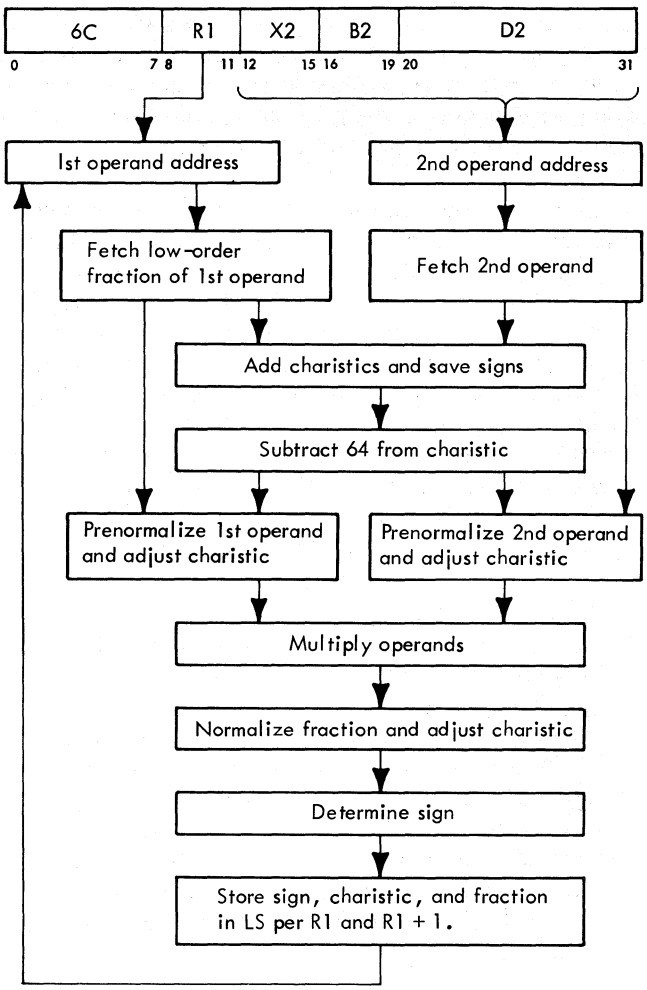
The operands are now in position so that multiplying may begin. The ROS microprogram assumes that both operands are normalized. The operands are tested, however, to determine, via a 4-way branch, whether prenormalization is necessary. Assume that the first operand is normalized and that the second operand needs normalizing. The second operand is normalized by shifting the contents of DT L1 hexadecimal digit and subtracting 1 from the characteristic on each shift. Left-shifting continues until the fraction is normalized. Since the characteristic sum is in excess 128, 64 is subtracted from the characteristic in F. AB is reset, and the first multiple is selected. The multiples are selected per E(12-15) until E(12-15) equals 0001. This value indicates that the multiples must be selected from the high-order fraction located in LS. This high-order fraction of the first operand (multiplier) is fetched from LS and placed in S as specified by the R1 field [E(8-11)]. From this point, multiply execution is the same as for short operand multiply instructions. See paragraphs 3.6.8.1 for completion of MDR instruction and 3.6.8 for description of multiply algorithm.

3.6.8.4 MD (6C) - RX Long Operands

- Multiplies 1st operand (multiplier per R1 and R1 + 1) by 2nd operand (multiplicand from main storage); normalized product is placed in 1st operand location.
- RX format (see adjoining column)
- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

The Multiply, MD, instruction multiplies the first operand (multiplier specified by R1 and R1 + 1) by the second operand (multiplicand from main storage), and the normalized product is placed in the first operand location.

The MD instruction is in the RX format with an op code of 6C. This instruction uses 64-bit operands, and the final 64-bit product is stored in LS as specified by R1 and R1 + 1.



The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The effective address of the second operand is in D.
3. The MD instruction is in E.

For the following MD instruction analysis, refer to Figure 6042, FEDM.

The first operand (multiplier) sign, characteristic, and high-order fraction are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed in S (Figure 6042C, FEDM). If the first operand is not normalized, the STC is reset, the low-order fraction is transferred from S to B, the second operand (multiplicand) is fetched from main storage and placed in ST and D (high-order fraction in D), the characteristics are added, and the sign is saved (first operand sign in STAT F, and second operand sign in STAT C).

The low-order fraction of the first operand is again placed in S, and the constant 15 is placed in E(12-15). The 4-way branch determines the next operation. The prenormalization operations, multiply operation, and multiply algorithm are discussed in paragraphs 3.6.8.1, 3.6.8.3, and 3.6.8, respectively.

If the first operand was normalized, the second operand is fetched from main storage and placed in AB (Figure 6042C, FEDM). The second operand fraction (multiplicand) is transferred from AB to DT. The sign of the first operand is saved in STAT C; the sign of the second operand is saved in STAT F. The characteristics of the first and second operands are added, with the results placed in F, and the characteristic carry is saved in STAT D. The carry is also transferred to F(0). Since the first operand is normalized, the ROS microprogram assumes that the second operand is also normalized; therefore, the first multiple is selected. If the second operand needs normalizing, the initial conditions are restored and the ROS microprogram proceeds with normalizing the second operand. Normalization of the second operand is shown in Figure 6042E, FEDM.

3.6.9 DIVIDE

- Divides 1st operand (dividend) by 2nd operand (divisor); normalized quotient is placed in 1st operand location.
- Operands are prenormalized before dividing.
- Quotient is 32 bits for short operands, 64 bits for long operands.
- No remainder is retained.
- Quotient is always normalized.
- Sign of quotient is determined algebraically.
- Characteristics are subtracted, and 64 is added to the characteristic difference.
- Instructions:
  DER (3D) - RR Short Operands
  DE (7D) - RX Short Operands
  DDR (2D) - RR Long Operands
  DD (6D) - RX Long Operands

The Divide instruction divides the first operand (dividend) by the second operand (divisor), and the normalized quotient is placed in the first operand location. In short operand instructions, the low-order halves of the floating-point registers are ignored and remain unchanged.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics, plus 64, is used as an intermediate quotient characteristic. The sign of the quotient is determined algebraically.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right shift of one hexadecimal digit may be necessary if the normalized dividend fraction is larger than the normalized divisor fraction. The intermediate quotient characteristic is adjusted for the shifts. The quotient fraction is truncated to the desired number of digits.

A program interruption for exponent overflow occurs when the final quotient characteristic exceeds 127. The operation is terminated when an exponent overflow occurs.

A program interruption for exponent underflow occurs when the final quotient characteristic is less than 0 and the corresponding mask bit is a 1. Underflow is not signalled for the intermediate quotient or for the operand characteristics during prenormalization.

If division by a divisor with a zero fraction is attempted, the divide operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient fraction will be zero. The quotient sign and the characteristic are made zero, yielding a true zero result without taking the program interruption for exponent underflow or exponent overflow. The program interruption for significance is never taken for division. The CC remains unchanged.

After the first and second operands are fetched and placed in the proper registers, the characteristics are subtracted. Since the complement gates to the SA are on the SAA bus, the first operand characteristic (C1) is subtracted from the second operand characteristic (C2). Therefore, the characterisitc computation procedure differs from what might be expected. (Normally, C1 - C2 would be expected.)

The CPU takes the following steps in computing the characteristic value:

1. Subtracts C1 from C2 (dividend characteristic from divisor characteristic).

2. Subtracts 64 from characteristic difference.

3. Normalizes the first operand and adds the number of shifts taken to the intermediate characteristic value.

4. Normalizes the second operand and subtracts (takes 2's complement) the number of shifts necessary from the intermediate characteristic value.

5. Takes 2's complement of the intermediate characteristic value.

6. Checks for divisor greater than dividend. If necessary, shifts the dividend R1 hexadecimal digit and adds 1 to the characteristic.

7. Saves the final characteristic value.

8. Checks final characteristic for exponent overflow or exponent underflow.

As an example of this computation, assume that two hexadecimal numbers are to be divided, .004 by .02:

$$\frac{\text{1st operand dividend}}{\text{2nd operand divisor}} \quad \frac{\pm .004 \times 16^{5}}{.02 \times 16^{2}} = \pm .2 \times 16^{3} = \pm\ 3.2$$

Fraction

Characteristic

Convert the above characteristics to excess 64 notation:

C1

69.004

66.02

C2

Convert the above characteristics to binary form:

$$\frac{01000101.004}{01000010.02} = \frac{69.004}{66.02} = \pm 67.2 \text{ or } .2 \times 16^{3}$$

after 64 is subtracted from the characteristic.

Step 1. The machine subtracts the characteristics (C2 - C1):

```
01000010      C2
10111010   }
       1   }  2's complement of C1
--------
11111101                          .004
                                  --------
                                  11111101.02
```

Step 2. 64 is subtracted from the characteristic value to maintain excess 64 notation:

```
11111101
10111111   }
       1   }  2's complement of 64          .004
--------                                    --------
10111101                                    10111101.02
```

Step 3. Note that the first operand hexadecimal fraction requires two left shifts to prenormalize. Shift L2 and add 2 to the characteristic:

```
10111101
00000010                                    .4
--------                                    --------
10111111                                    10111111.02
```

Step 4. The second operand hexadecimal fraction requires one left shift. Shift L1 and subtract 1 from the characteristic value:

```
10111111
11111110   }
       1   }  2's complement of 1           .4
--------                                    --------
10111110                                    10111110.2
```

Step 5. Take the 2's complement of the characteristic value:

```
01000001                                    01000010.4
       1                                    --------
--------    2's complement of 10111110      .2
01000010
```

Step 6. Since the dividend fraction is greater than the divisor fraction, the dividend is shifted R1 and 1 is added to the characteristic value before dividing fractions:

```
01000010
00000001                                    01000011.04
--------                                    --------
01000011    = 67 = final characteristic value   .2
```

Step 7. Save 67, which is the final characteristic value.

Step 8. Divide fractions and store quotient:

$$\frac{01000011.04}{.2} = \pm\ 67.2$$

Result fraction

Result characteristic

Subtracting the first operand characteristic from the second operand characteristic effectively makes the characteristic difference part of the divisor (dividend/divisor); to add to the characteristic, therefore, the value must be subtracted. For example, excess 64 notion is used in the CPU. Subtracting

(C1 + 64 from C2 + 64) equals C2 - C1 + 0; therefore, 64 must be added to the characteristic difference to maintain excess 64 notation. Since the C2 minus C1 difference is 2's complemented later in the operation, 64 must be subtracted (2's complement and add) from the characteristic that is part of the divisor. The characteristic must be part of the dividend to obtain the final quotient characteristic.

The 2's complement of the intermediate characteristic is necessary to obtain the correct characteristic value of the quotient because the initial characteristic subtraction places the intermediate characteristic in the divisor. Note that the intermediate characteristic is not considered to be in the 2's complement form.

In the divide operation, both fractions must be normalized before dividing the fractions. Also, the divisor must be larger than the dividend. If the divisor is less than the dividend, the dividend is divided by 16 by shifting the dividend right four binary bit positions. Prenormalizing and making the divisor larger than the dividend make postnormalizing unnecessary.

The basic divide algorithm for the floating-point fraction divide is similar in operation to the algorithm used in fixed-point divide. The basic floating-point divide algorithm may be stated as follows: the characteristics of the two operands are subtracted, and 64 is added to maintain excess 64 notation. The divisor fraction is subracted from the dividend fraction. A carry indicates that the dividend is greater than the divisor. The dividend must be less than the divisor; if not, an R4 shift of the dividend is required. Division is accomplished by successive subtractions and storing the quotient bits as determined by the carry. Successive subtractions are performed, and the dividend is effectively shifted L1 position for each subtraction.

The first operation that occurs in obtaining the final quotient is computation of the final characteristic. An example of characteristic computation is given earlier in this section. The data paths for the signs and characteristics are shown in A of Figure 6043A, FEDM. The signs are saved in STAT C and STAT F. Characteristic computation is accomplished in the SA. The first and second operand characteristics are gated to the SAA and SAB per the ABC and STC, respectively. To subtract characteristics, the 2's complement of the first operand characteristic is added to the second operand characteristic. The characteristic difference is stored in F(0-7), and the characteristic carry [SA(0)] is saved in STAT D. Other inputs to the SAB bus allow subtracting 64, subtracting 1, gating the 2's complement of F, or adding 1 to the value in F. After the final characteristic is computed, the result is stored in S(0-7) per the STC.

The data path for the derivation of the divide multiple is shown in B of Figure 6043A, FEDM. When the divide algorithm begins, the divisor (first operand) is in DT and the dividend is in AB.

The two micro-orders used when executing the divide algorithm are Divide Select Multiple L0 Insertion (DVDL0) and Divide Select Multiple L1 Insertion (DVDL1). These micro-orders have three functions: (1) to gate the true or 2's complement of DT to the PAA; (2) to determine the amount of shift (L0 = no shift, L1 = left one shift) of the divisor (contents of DT) to the PAA; and (3) to determine the partial quotient (PQ) bit and the PQ bit location after addition of the divide multiple and partial remainder has taken place.

The selection of the divide multiple is determined by the PA(4) carry from the previous algebraic addition of the dividend and partial remainder, and by the DVDL0 or the DVDL1 micro-order. If a PA(4) carry occurred, the 2's complement is gated (L0 or L1) to the PA. If not PA(4) carry occurred, DT is gated (L0 or L1) to the PA (C of Figure 6043A, FEDM). The data in AB is gated to PAB with no shift or an L2 shift. The gating from AB is under micro-order control.

As previously noted, the DVDL0 and the DVDL1 micro-orders determine the PQ bit and the location of the bit. The PQ bit is determined by testing AB(4) for a 0 or a 1. If AB(4) equals a 0, the partial remainder is in true form and a 1 is placed in the selected PQ bit location in SAL. If AB(4) equals a 1, the remainder is in 2's complement form and a 0 is placed in the selected PQ SAL location (C of Figure 6043A, FEDM).

As shown in C of Figure 6043A, FEDM, the PQ location in SAL is determined by E(14,15) and by the DVDL0 or DVDL1 micro-order. E(14,15) selects the pair of SAL bits in which the PQ bit is to be placed. The DVDL0 micro-order selects the odd bit of the selected pair; the DVDL1 micro-order selects the even bit. At the same time that the PQ bit is gated into SAL, the contents of F are added to the PQ bit and saved in F. After a PQ byte (8 bits) is available, the contents of F(0-7) are gated to S per the STC. After S is filled with the quotient (or PQ), the contents of S are stored in LS per E(8-11).

For a discussion of the divide algorithm, assume that the final characteristic is in S(0-7) and that the

normalized fractions are in DT (divisor) and AB (dividend). By definition, the CPU requires that floating-point numbers consist of a sign, a characteristic, and a fraction. Since no provisions are made in the CPU to handle integers in floating-point instructions, the divisor must be larger than the dividend to retain a fraction quotient. After both fractions are normalized, therefore, the contents of DT are subtracted from the contents of AB. A carry from PA(4) indicates that the dividend is larger than the divisor. Whenever the dividend is larger than the divisor, the contents of AB must be restored and shifted R4 (divided by 16) before proceeding with the divide algorithm, and 1 must be added to the characteristic. If there is no carry from PA(4), the dividend is less than the divisor and the CPU proceeds with the divide algorithm.

When the divisor (d) is subtracted from the dividend (D), the difference is placed in AB (D-d in AB). If an R4 shift was necessary, the divisor (d) is restored and divided by 16 (d in DT). AB now contains the dividend (D). At the beginning of the divide algorithm, the 2's complement of DT is shifted L1 and added to the contents of AB; the total is shifted L2 with the result placed in AB, thus yielding an effective L1 shift of AB (dividend). The contents of AB may be expressed by the equation 4D - 2d = contents of AB. The value 4D - 2d is in AB after the first machine cycle of the divide algorithm.

If the dividend was less than the divisor, D - d is in AB. The CPU proceeds to add the contents of DT shifted L1 to the contents of AB shifted L2, with the result placed in AB. This addition results in the equation 4(D - d) + 2d = contents of AB. Simplifying the equation yields 4D - 4d + 2d = 4D - 2d. At the end of the first cycle of the divide algorithm, the same result (4D - 2d) is obtained as when the dividend was larger than the divisor. The CPU continues with the divide algorithm.

During the first machine cycle of the divide algorithm, the DVDL0 micro-order also selects the DT gating to the PA per the PA(4) carry. The actual subtraction resulting from the DVDL0 micro-order is accomplished during the machine cycle following the divide multiple selection. The PQ bit is determined by the A(4) value that was computed during the previous machine cycle. On the first cycle of the divide algorithm, the contents of AB are shifted L2 by a micro-order and added to the contents of DT shifted L1 per a micro-order. The divisor is shifted R1 with respect to the dividend but displaced L2 in AB. On the next divide select multiple subtraction, the dividend and the divisor are subtracted, yielding the correct R1 shift. The following cycle causes AB and DT to shift again. Note that, as the remainder is shifted left, the low-order bit positions of AB are filled with 0's.

The divide algorithm may be divided into five parts:

1. PQ bit gating
2. Byte gating
3. Quotient storage
4. Instruction branch
5. End op

As previously noted, the PQ bit is gated to SAL per E(14,15) and the DVDL0 or DVDL1 micro-order (C of Figure 6043A, FEDM). A value of 1 is added to the ABC after each pair of PQ bits is gated to F via SAL. When the ABC equals 3, F contains eight PQ bits (one PQ byte). The PQ byte is gated to S per the STC.

After each byte is gated to S, 1 is added to the STC. When the STC equals 3, S contains the characteristic and fraction (or high-order fraction). The contents of S are stored in the LSWR.

Before initiating the divide algorithm, STAT D was reset to indicate the first pass at loading PQ bytes into S. After the sign, characteristic, and high-order fraction are stored in the LSWR, the instruction and STAT D determine the next operation. If a short operand instruction is being executed, the sign is inserted and stored with the characteristic and fraction in LS per E(8-11). An end-op cycle completes instruction execution.

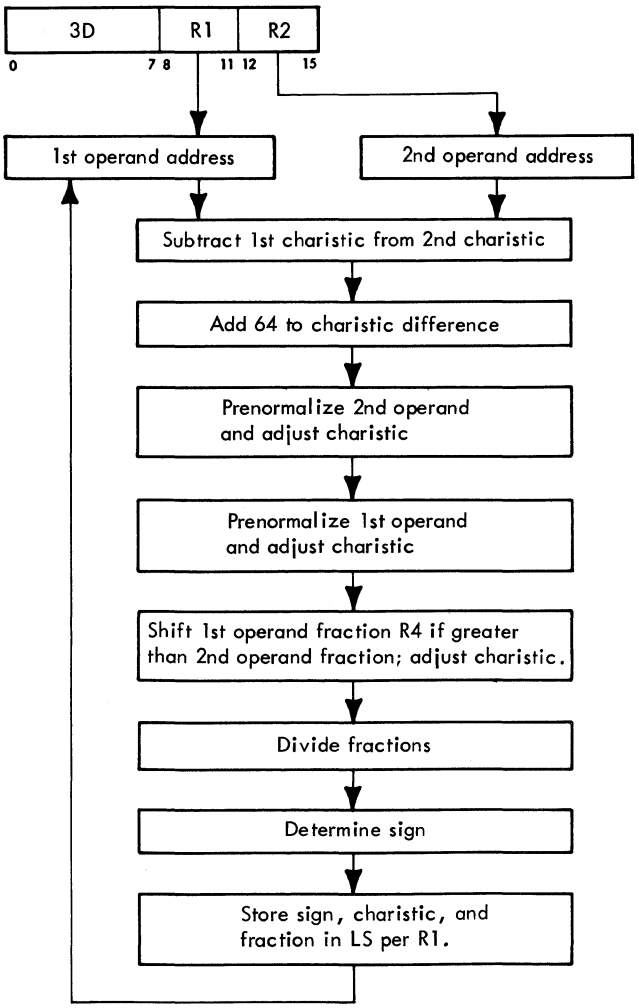
If the instruction was a long operand instruction, the sign, characteristic, and high-order fraction are stored in LS per E(8-11). STAT D is set, and the divide algorithm continues. The contents of the LSWR are returned to T. The same operations are performed as described above in obtaining the remaining low-order fraction part of the quotient, and the same 3-way branch is encountered. This time the divide algorithm is completed, and the low-order fraction is stored in LS per E(8-11) + 1. An end-op cycle completes instruction execution. The remainder in AB is not stored.

3.6.9.1 DER (3D) - RR Short Operands

- Divides 1st operand (dividend per R1) by 2nd operand (divisor per R2);

normalized quotient is placed in 1st operand location.

- RR format:



- Conditions at start of execution:
  1st operand is in A, B, and D (24-bit fraction only).
  2nd operand is in S and T.
  Instruction is in E.

The Divide, DER, instruction divides the first operand (dividend specified by R1) by the second operand (divisor specified by R2), and the normalized quotient is placed in the first operand location. No remainder is retained.

The DER instruction is in the RR format with an op code of 3D. This instruction uses 32-bit operands, and the final result is 32 bits in length.

The conditions at the beginning of the execution phase are:

1. The first operand is in A, B, and D (24-bit fraction only).
2. The second operand is in S and T.
3. The STC contains a value of 4.
4. The DER instruction is in E.

For the following DER instruction analysis, refer to Figure 6043, FEDM.

The fraction of the second operand is transferred from T to D. The characteristics are subtracted, and 64 is algebraically added to the characteristic difference to maintain excess 64 notation. The sign of the first operand is saved in STAT F; the sign of the second operand is saved in STAT C. B and T are reset, and the contents of AB and ST are treated as 56-bit fractions (Figure 6043B, FEDM).

In the divide instructions, both operands are prenormalized before the divide algorithm begins. A 4-way branch determines the prenormalizing path that is to be followed. The 4-way branch tests for the following by testing AB(8-11), the dividend, and PAL(40-43), the divisor, for the normalized conditions:

1. The first and second operands are normalized.
2. The first operand is normalized, and the second operand is unnormalized.
3. The first operand is unnormalized, and the second operand is normalized.
4. The first and second operands are unnormalized.

Assume that both operands are unnormalized. The second operand (dividend in DT) is shifted L4 until the operand is normalized. 1 is subtracted from the characteristic for each shift. For characteristic computation, refer to paragraph 3.6.9.

After the second operand is normalized, the first operand is normalized by left-shifting the first operand until the fraction contains a hexadecimal digit [A(8-11) not equal to zero]. On each left shift, 1 is added to the characteristic value in F (Figure 6043C, FEDM).

After the operands are normalized, the second operand fraction is subtracted (take 2's complement of second operand and add) from the first

operand fraction. Before branching on the PAL(4) carry, the 2's complement of the characteristic is computed and placed in F. Also, the constant 5 is placed in E(12-15) for controlling the divide algorithm (Figure 6043D, FEDM). A carry from PAL(4) indicates that the dividend is larger than the divisor. If the dividend is larger than the divisor, the dividend is restored and is divided by 16 by a right shift of one hexadecimal digit. 1 is added to the characteristic value, which is the final characteristic of the quotient. The final characteristic is placed in S(0-7).

No carry from PAL(4) indicates that the dividend is less than the divisor, at which time the first machine cycle of the divide algorithm is executed. A test is made to determine an overflow or underflow condition. Assume that no overflow or underflow condition exisits.

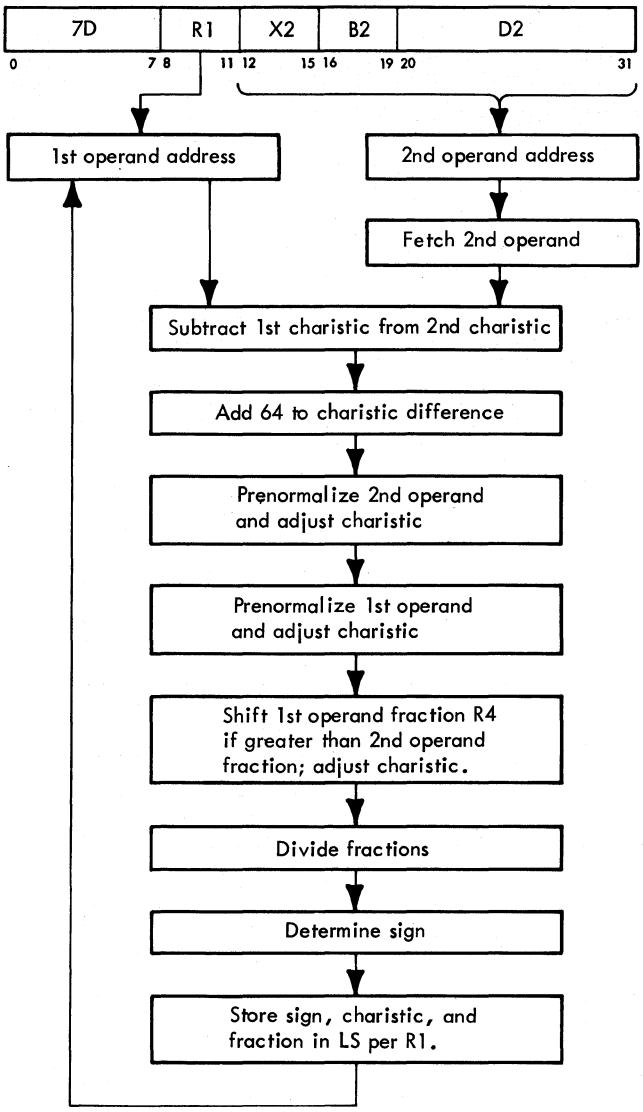
Fraction division begins as shown in Figure 6043D, FEDM. (Refer to paragraph 3.6.9 for a discussion of the divide algorithm.) Figure 3-6 is an example of the action that occurs in adder bits 4-11 (bits 12-31, or 12-63 for long operands, being considered to equal 0's).

During the normalization routine, tests for zero fractions are made. If the second operand fraction (divisor) equals zero, the divide operation is suppressed and a floating-point divide interruption occurs. If the first operand fraction (dividend) equals zero, a true zero quotient results (zero/divisor). A true zero is stored in the first operand location, and an end-op cycle completes instruction execution (Figure 6043C and E, FEDM). (Exponent overflow and underflow conditions are explained in paragraph 3.5.7.)

3.6.9.2 DE (7D) - RX Short Operands

- Divides 1st operand (dividend per R1) by 2nd operand (divisor from main storage); normalized quotient is placed in 1st operand location.
- RX format (see adjoining column)
- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

The Divide, DE, instruction divides the first operand (dividend specified by R1) by the second



operand (divisor from main storage), and the normalized quotient is placed in the first operand location. No remainder is retained.

The DE instruction is in the RX format with an op code of 7D. This instruction uses 32-bit operands, and the final result is 32 bits in length.

The conditions at the beginning of the execution phase are:

1. The first operand is in S and T.
2. The effective address of the second operand is in D.
3. The instruction is in E.

For the following DE analysis, refer to Figure 6043, FEDM.

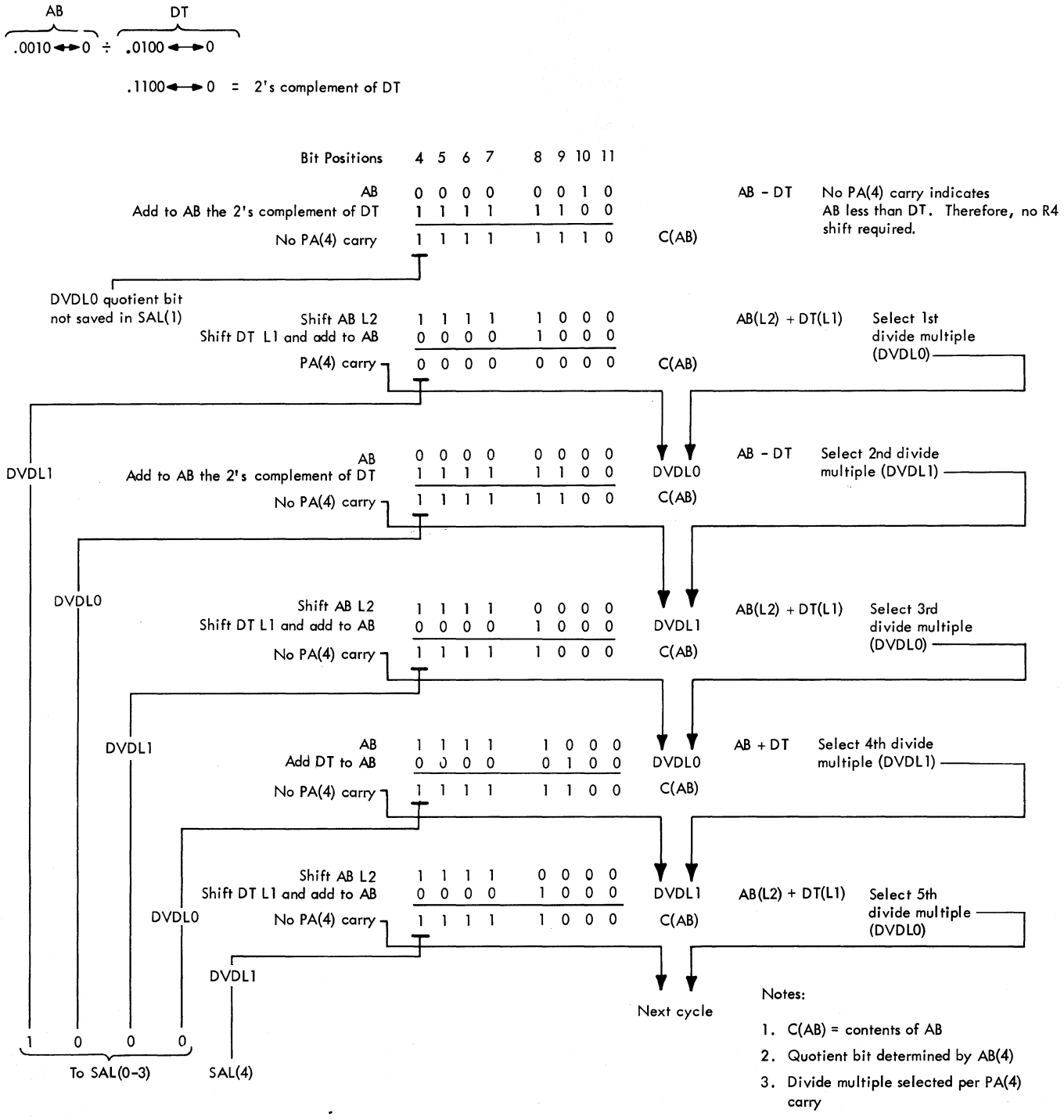


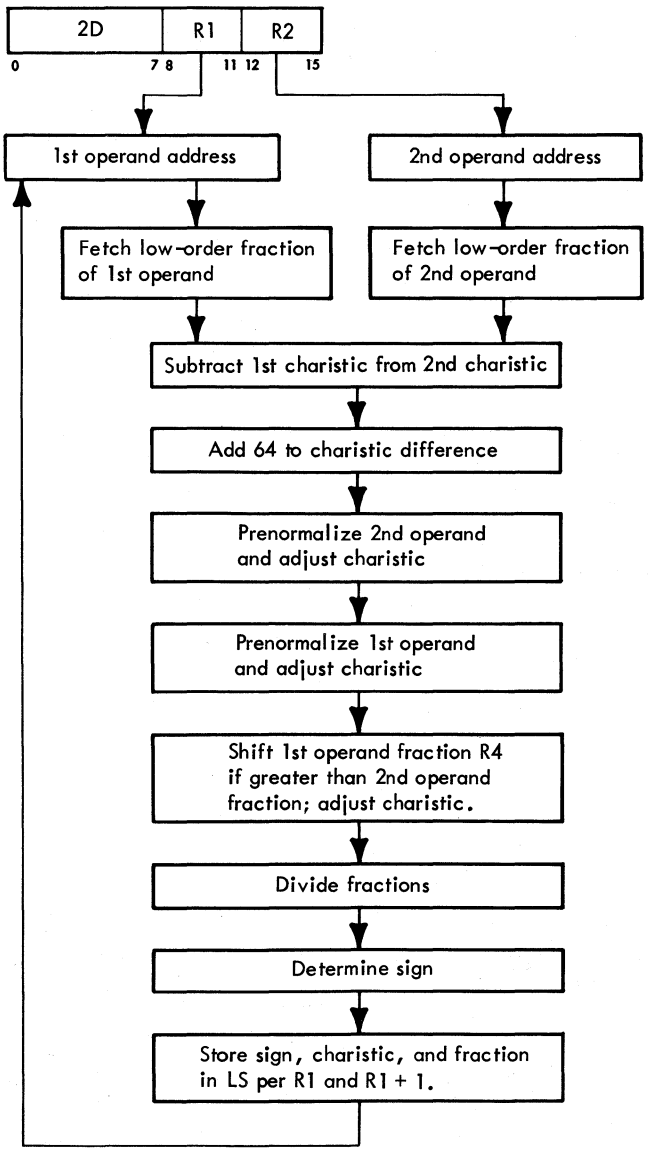
FIGURE 3-6. FLOATING-POINT DIVIDE EXAMPLE

The first operand is transferred from T to A, and the STC is set to 4. The second operand is fetched from main storage per D. D(21) determines which 32 bits of the 64-bit double word are gated to T (Figure 6043B, FEDM).

Instruction execution objectives from this point are the same as the DER instruction discussed in paragraph 3.6.9.1. For characteristic computation and the divide algorithm discussion, refer to paragraph 3.6.9.

3.6.9.3 DDR (2D) - RR Long Operands

- Divides 1st operand (dividend per R1 and R1 + 1) by 2nd operand (divisor per R2 and R2 + 1); normalized quotient is placed in 1st operand location.
- RR format:



- Conditions at start of execution:
  - 32 bits of 1st operand are in A, B, and D (24-bit fraction only).
  - 32 bits of 2nd operand are in S and T.
  - Instruction is in E.

The Divide, DDR, instruction divides the first operand (dividend specified by R1 and R1 + 1) by the second operand (divisor specified by R2 and R2 + 1), and the normalized quotient is placed in the first operand location. No remainder is retained.

The DDR instruction is in the RR format with an op code of 2D. This instruction uses 64-bit operands, and the final result is 64 bits in length.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in A, B, and D (24-bit fraction only).
2. 32 bits of the second operand are in S and T.
3. The STC contains a value of 4.
4. The DDR instruction is in E.

For DDR instruction analysis, refer to Figure 6043, FEDM.

The low-order fractions of the first and second operands are placed in B and T, respectively. Previous to the operand fetch, the high-order fraction is gated from T to D. After the operands are fetched and placed in the proper registers, the dividend fraction is in AB and the divisor fraction is in DT (Figure 6043B, FEDM).

The signs are saved in STAT C and STAT F. The characteristics are subtracted, and excess 64 notation is maintained. The ABC is reset.

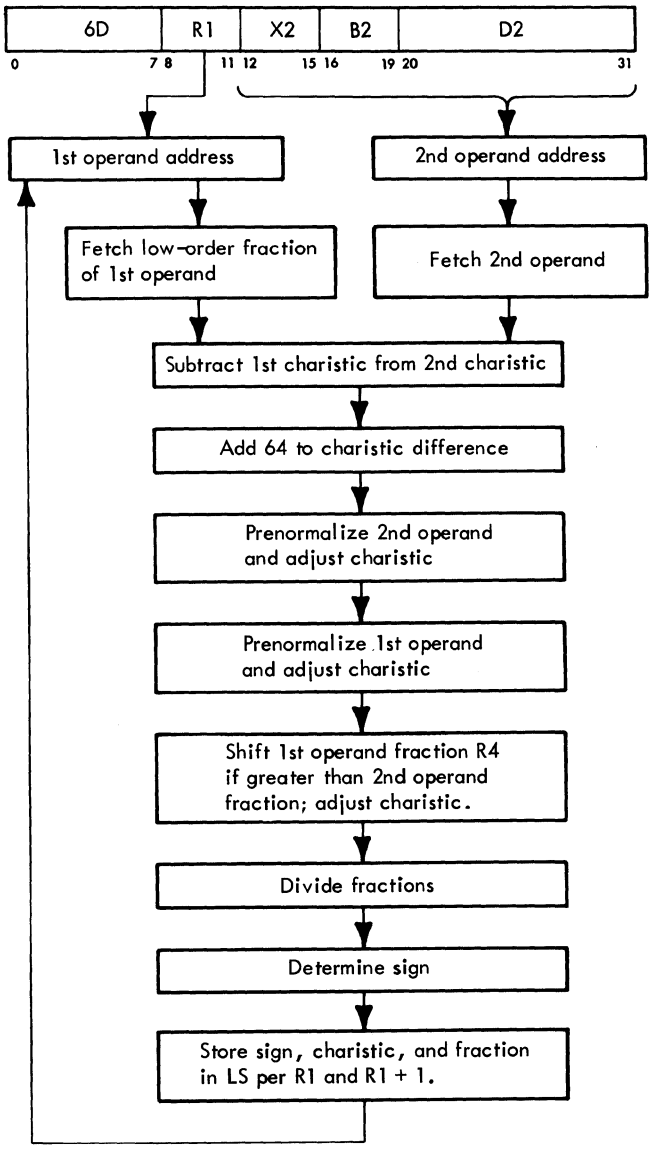
The 4-way branch (Figure 6043C, FEDM) determines the next operation. The normalization routine, divide operation, and end ops are explained in the DER instruction discussion (paragraph 3.6.9.1) and in the introduction to Divide (paragraph 3.6.9).

3.6.9.4 DD (6D) - RX Long Operands

- Divides 1st operand (dividend per R1 and R1 + 1) by 2nd operand (divisor from main storage); normalized

quotient is placed in 1st operand location.

- RX format:



- Conditions at start of execution:
  32 bits of 1st operand are in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.

The Divide, DD, instruction divides the first operand (dividend specified by R1 and R1 + 1) by the second operand (divisor from main storage), and the normalized quotient is placed in the first operand location. No remainder is retained.

The DD instruction is in the RX format with an op code of 6D. This instruction uses 64-bit operands, and the final result is 64 bits in length.

The conditions at the beginning of the execution phase are:

1. 32 bits of the first operand are in S and T.
2. The effective address of the second operand is in D.
3. The instruction is in E.

For DD instruction analysis, refer to Figure 6043, FEDM.

The sign, characteristic, and high-order fraction of the first operand are transferred from T to A. The low-order fraction of the first operand is fetched from LS and placed in B via T and the parallel adder. The second operand (64 bits) is fetched from main storage and placed in ST. The high-order fraction is transferred from S to D, and the contents of DT become the 56-bit fraction divisor. The signs are saved in STAT C and STAT F. The characteristics are subtracted, and 64 is added to the characteristic difference to maintain excess 64 notation (Figure 6043B, FEDM).

The first operand (dividend) is located in AB, and the second operand (divisor) in DT. The next step is to check for the prenormalization of the dividend and divisor fractions. Prenormalization, divide operation, and instruction terminations are discussed in paragraph 3.6.9.1. The divide algorithm is discussed in paragraph 3.6.9.

3.6.10 STORE

- Stores 1st operand in 2nd operand location.
- Instructions:
  STE (70) - RX Short Operands
  STD (60) - RX Long Operands
- Address store compare test is made on all store instructions.

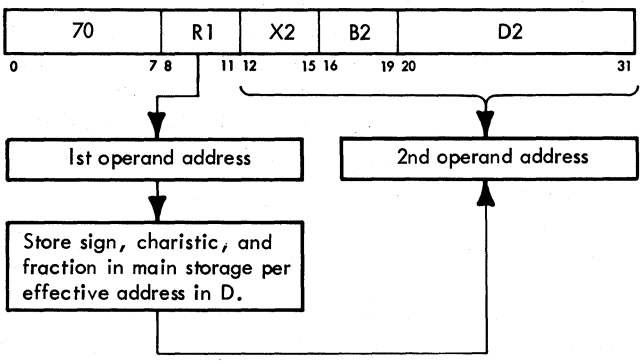
The Store instructions (STE and STD) store the first operand from LS in the second operand location in main storage. The Store instructions are in the RX format with short and long operand options available. In the STE instruction, the low-order half of the first operand register is ignored. The first operand location remains unchanged.

Storing must be on word boundaries for the STE instruction and on double-word boundaries for the STD instruction.

For all Store instructions, an address store compare test is made because the instructions that are in Q may be modified in main storage by the Store instruction. If an instruction is modified in main storage and is not corrected in Q, the program may not be properly executed; therefore, Q must be reloaded. The address store compare test is made by comparing the main storage address, where data is to be stored, with the effective address indicated by the Store instruction. The comparison is made by subtracting the contents of D (effective address) from the contents of the IC, shifting the difference R4, and testing for a zero result. If the difference equals zero, the difference is less than 16; therefore, Q must be reloaded. The address-store-compare trigger is set to indicate that the instructions in Q must be refetched. The address store compare test is discussed in detail in paragraph 3.2.5.1.

3.6.10.1 STE (70) - RX Short Operands

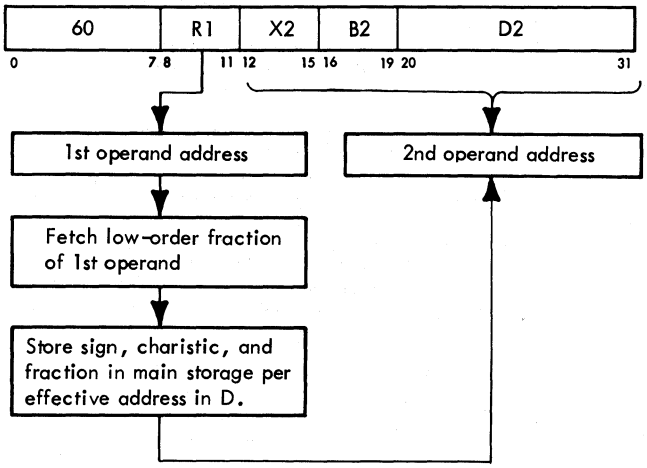
- Stores 1st operand (per R1) in 2nd operand location (in main storage).
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.
- Store 1st operand in main storage per effective address in D.
- Main storage address must be on word boundary.
- Figure 6044, FEDM.

3.6.10.2 STD (60) - RX Long Operands

- Stores 1st operand (per R1 and R1 + 1) in 2nd operand location (in main storage).
- RX format:



- Conditions at start of execution:
  1st operand is in S and T.
  Effective address of 2nd operand is in D.
  Instruction is in E.
- Store 1st operand in main storage per effective address in D.
- Main storage address must be on double-word boundary.
- Figure 6045, FEDM.

This section describes the general handling and specific execution sequences of the decimal instruction set. The decimal instructions provide for addition, subtraction, comparison, multiplication, and division, as well as format conversion of variable field length (VFL) operands.

3.7 INTRODUCTION

- Instructions operate on VFL data.
- VFL data is in main storage only.
- All instructions are in SS format.

All decimal instructions operate on VFL data which may range from 1 to 16 bytes in length. This data resides in main storage only. All decimal instructions are therefore in the SS format to provide for storage-to-storage operations. In general, most decimal instructions require fetching the operands from main storage, performing the operations specified by the instruction op code, and storing the results in main storage.

3.7.1 NUMBER REPRESENTATION

- Binary codes 0000-1001 are used for decimal digits 0-9.
- Binary codes 1010-111 are used for sign notation, and are invalid as digits.
- Sign notation is dependent on code: EBCDIC or ASCII-8.

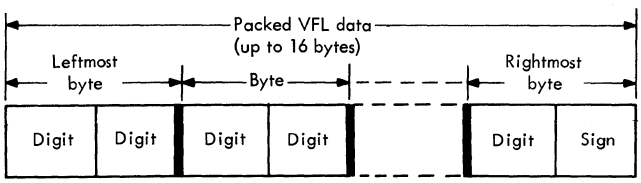
Representation of a binary digit requires a 4-bit field. Decimal digits 0-9 have the binary codes 0000-1001, respectively. Binary codes 1010-1111 (10 through 15, inclusive) are not valid as digits. This set of codes is used to represent the operand sign. Codes 1010, 1100, 1110, and 1111 are recognized as plus, and codes 1011 and 1101 as minus.

The specific sign codes, generated during decimal operations, differ for the extended binary-coded-decimal interchange code (EBCDIC) and the American Standard code for information interchange extended to 8 bits (ASCII-8). The choice between the two codes is established by bit 12 of the PSW. When bit 12 is 0, the preferred EBCDIC codes are generated: 1100 for plus, and 1101 for minus. When bit 12 is 1, the preferred ASCII-8 codes are generated: 1010 for plus, and 1011 for minus.

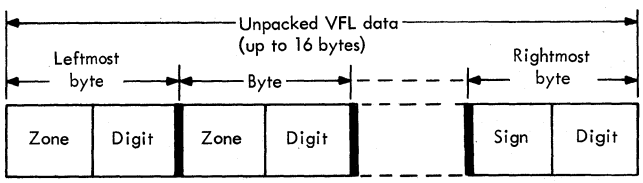
3.7.2 DATA FORMAT

- Packed: 2 digits per byte.
- Unpacked: zone and digit per byte.
- Operands are right-aligned in their field.
- Rightmost byte contains sign of operand.
- Operand processing is from right to left, or from low-order to high-order.

Data may be in the packed or unpacked (zoned) format. In the packed format, two decimal digits are normally placed adjacently in an 8-bit byte. An exception is the rightmost byte of the field, where a sign is placed to the right of the decimal digit.



In the unpacked or zoned format, a decimal digit normally occupies the four low-order bits of a byte, the "numeric." The four high-order bits of a byte are called the "zone."* An exception is the rightmost byte in the field, where the sign occupies the zone position.



* The zone codes generated in decimal operations differ for the EBCDIC and the ASCII-8 codes. The zone code used in the EBCDIC is 1111; the ASCII-8 uses a 0101 zone code.

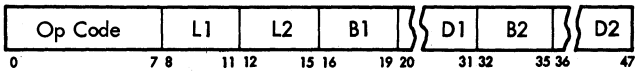
All arithmetic operations are performed on data in packed format. However, since data is often communicated to or from external devices in zoned format, the decimal instruction set provides the necessary format-conversion operations to convert from one format to another.

Decimal operands reside in main storage only. The operand field length may range from a minimum of 1 byte to a maximum of 16 bytes. The operands need not occupy the entire field length but are always right-aligned in the field; i.e., the sign of the operand is always in the rightmost byte of the specifi field field. This rightmost byte contains the lowest-order operand digit and the operand sign. All decimal instructions (except Divide) process the operands from low-order to high-order, or from right to left between main storage locations.

3.7.3 INSTRUCTION FORMAT

- Instructions specify two addresses.
- B1 (contents) + D1 + L1 specifies rightmost byte of 1st operand.
- B2 (contents) + D2 + L2 specifies rightmost byte of 2nd operand.
- Results are stored in true form at 1st operand location.

All decimal instructions use the SS format:



An SS instruction operates on two operands in main storage and stores the result in the same location from which the first operand was obtained. Therefore, the address of the first operand is also the destination address; the address of the second operand is commonly referred to as the "source address."

The contents of the general register specified by the B1 field are added to the D1 field to form an address. This address specifies the leftmost byte of the first operand. The number of operand bytes to the right of this byte is specified by the L1 field of the instruction. The L1 field may specify up to 16 bytes. Similarly, the address of the second operand is specified by the B2, D2, and L2 fields of the instruction. A zero in the instruction B1 or B2 fields indicates the absence of the corresponding address component.

Normally, decimal operands are processed from right to left. Thus the address for the initial operand fetch is:

LS register per B field + D field + L field.

Operands are fetched from main storage one double word, or 8 bytes, at a time. Since the L field may specify up to 16 bytes, several operand fetches may be required to completely access the operand. After each fetch, the operand address is decremented by 8 to access the next high-order 8 bytes of the operand.

The results of decimal operations are placed in the first operand field and must be in true form. The result is never stored outside the first operand field specified by the instruction. If the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand. Such extension does not modify the second operand in main storage, where it remains unchanged.

3.7.4 DATA HANDLING

- In excess-6 arithmetic, 6 is added to each digit input at A side of serial adder. Digits at B side are not affected.
- Decimal correction means subtraction of 6 from result upon detection of no-carry condition.
- During subtract operation, a no-carry condition indicates a complement result.

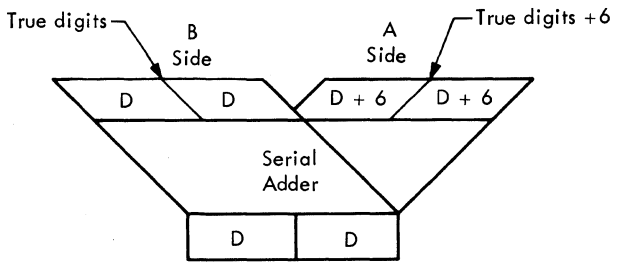
As stated previously, the decimal digits are represented by a straight binary code. Each digit consists of a 4-bit field, bit combinations 0000-1001 corresponding to decimal digits 0-9. This system of decimal notation allows relatively simple binary techniques to be applied when operating with decimal data, and also facilitates direct reading of decimal results. However, two problems are encountered. One problem is that the 4-bit field used to represent decimal digits has 16 possible codes, of which 6 (binary combinations for 10 through 15 inclusive) are invalid as decimal digits. Thus means must be provided to correct invalid results when they occur in an arithmetic operation. For example, the addition of decimal digits 0110 (six) and 0101 (five) must yield a decimal result of 0001 0001 (eleven).

If, however, a pure binary addition is carried out, it will yield an unacceptable result:

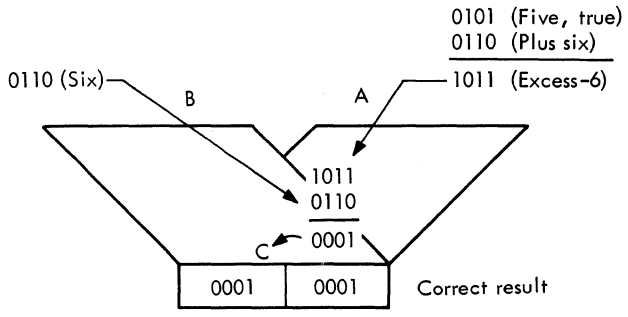
```
0110 (decimal or binary 6)
0101 (decimal or binary 5)
1011 (invalid as decimal, but 11 in binary)
```

The second problem is in the generation of a decimal carry. When the sum of two decimal digits exceeds 9, a carry must be sent to the next high-order digit. However, a pure binary addition will not yield a carry unless the sum of the digits exceeds 1111 (fifteen), which has the effect of a hexadecimal carry; i.e., carrying the order of sixteen rather than ten.

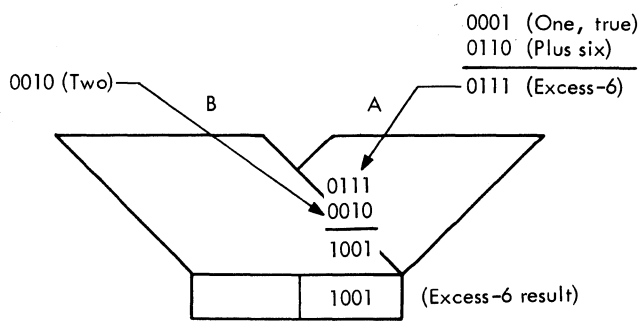
Both of the above problems are solved by the excess-6 arithmetic scheme and the decimal correction functions of the serial adder. In the excess-6 scheme, often referred to as true +6 arithmetic, a 6 is added to each digit as it is gated to the A side of the adder; the digits gated to the adder B side are not affected:



If the sum of the two digits to be added is greater than ten, the true +6 scheme will automatically eliminate the unwanted binary configuration and also supply a decimal carry in terms of hexadecimal carry. In true +6 arithmetic, the previous add example of digits 5 and 6 is executed as follows:

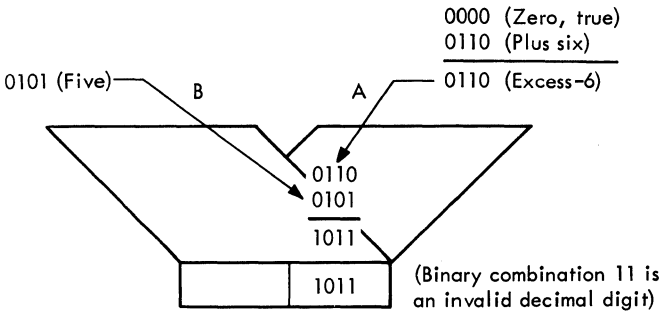


Addition of a 6 in all cases, however, may create an erroneous and sometimes invalid result. This occurs if the sum of the two digits to be added is less than 10. For example, consider the addition of decimal digits 1 and 2:



In the above case, the result (9) is clearly in excess-6 form; the digit 6 must be subtracted from the result to obtain the correct answer.

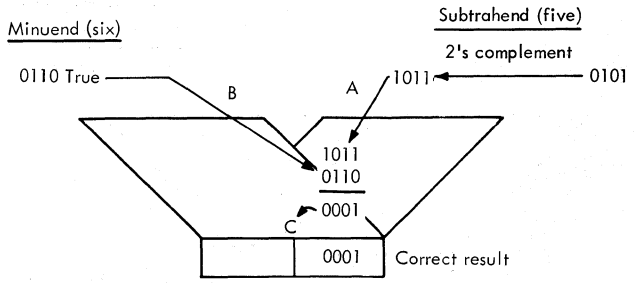
A further example illustrates how an excess-6 digit may generate an invalid result. Consider the addition of decimal digits 0 and 5:



Note that both the erroneous and the invalid results are characterized by a no-carry to the next high-order digit. This condition holds true in all cases when incorrect data is generated, and is utilized by the decimal correction logic of the adder. When a no-carry condition is detected, this logic automatically deducts 6 from the result, thus supplying the correct digit to the adder output.

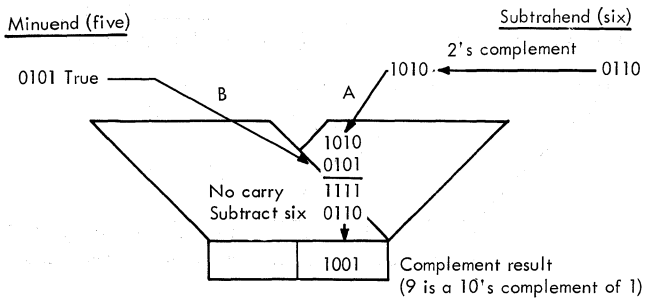
The decimal correct function of the adder is also used during subtract operations. Subtraction of one decimal digit from another is carried out by complement addition. The binary codes of the decimal digits at the adder A side are gated in 2's complement form; excess 6's are not supplied. The digits at the B side of the adder are gated in true form. The result of a subtract operation may be in true or complement form.

If the minuend is larger than the subtrahend, the result is in true form. Consider subtraction of decimal digit 5 from 6:

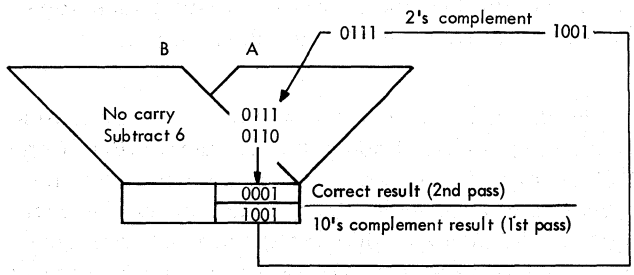


A true result is always characterized by a carry condition to the next digit. As in the case of the add operation, a carry to the next digit indicates that no decimal correction of the result is necessary.

If the minuend is smaller than the subtrahend, the result is in complement form. Since decimal data is always stored in true form, the result must be recomplemented. Consider subtraction of decimal digit 6 from 5:



Note that the decimal correction feature of the adder always subtracts 6 from the result upon detecting a no-carry condition. In subtract operation, however, a no-carry condition also indicates that the result is in complement form and must be recomplemented. This requires a second pass through the adder:



3.7.4.1 Data Flow

- All decimal instructions make use of serial adder.
- 1st operand is placed in ST, and 2nd operand in AB.
- STC specifies which ST byte is to be processed. ABC specifies which AB byte is to be processed.
- Destination bytes replace 1st operand bytes in ST.
- D contains 1st operand and destination address.
- IC contains 2nd operand address.
- L1 and L2 specify number of 1st and 2nd operand bytes, respectively, to be processed.

The data path used for decimal operations consists primarily of ST, AB, and the serial adder (Figure 3-7). ST contains the first operand, and AB the second. The input byte to the adder A side is selected from AB under control of ABC. The input to the B side of the adder is selected from ST under STC control. The selected bytes are gated to the adder simultaneously.

The serial adder handles the data at a rate of one byte per cycle; i.e., for each two input bytes, one output byte is generated at the SAL. The output byte is gated from SAL to ST under control of STC, after which ABC and STC are decremented and a new cycle is started. Thus, as the operation progresses, the first operand bytes in ST are replaced by the destination bytes.

The number of first and second operand bytes processed is dependent on length fields L1 and L2, respectively. The L1 count contained in E(8-11) is decremented once for each byte of first operand that is processed. Similarly, the L2 count, in E(12-15), is decremented once for each second operand byte processed.

D contains the address of the first operand, which is also the address of the destination. The initial address in D specifies the double word containing the rightmost byte of the operand. When STC is decremented to zero, indicating that all first operand bytes in ST have been processed, the contents of ST are stored in main storage. If additional first operand bytes remain in main storage (the L1 count has not stepped to zero), the D address is decremented by 8 and a fetch of the next operand double word is made to ST.

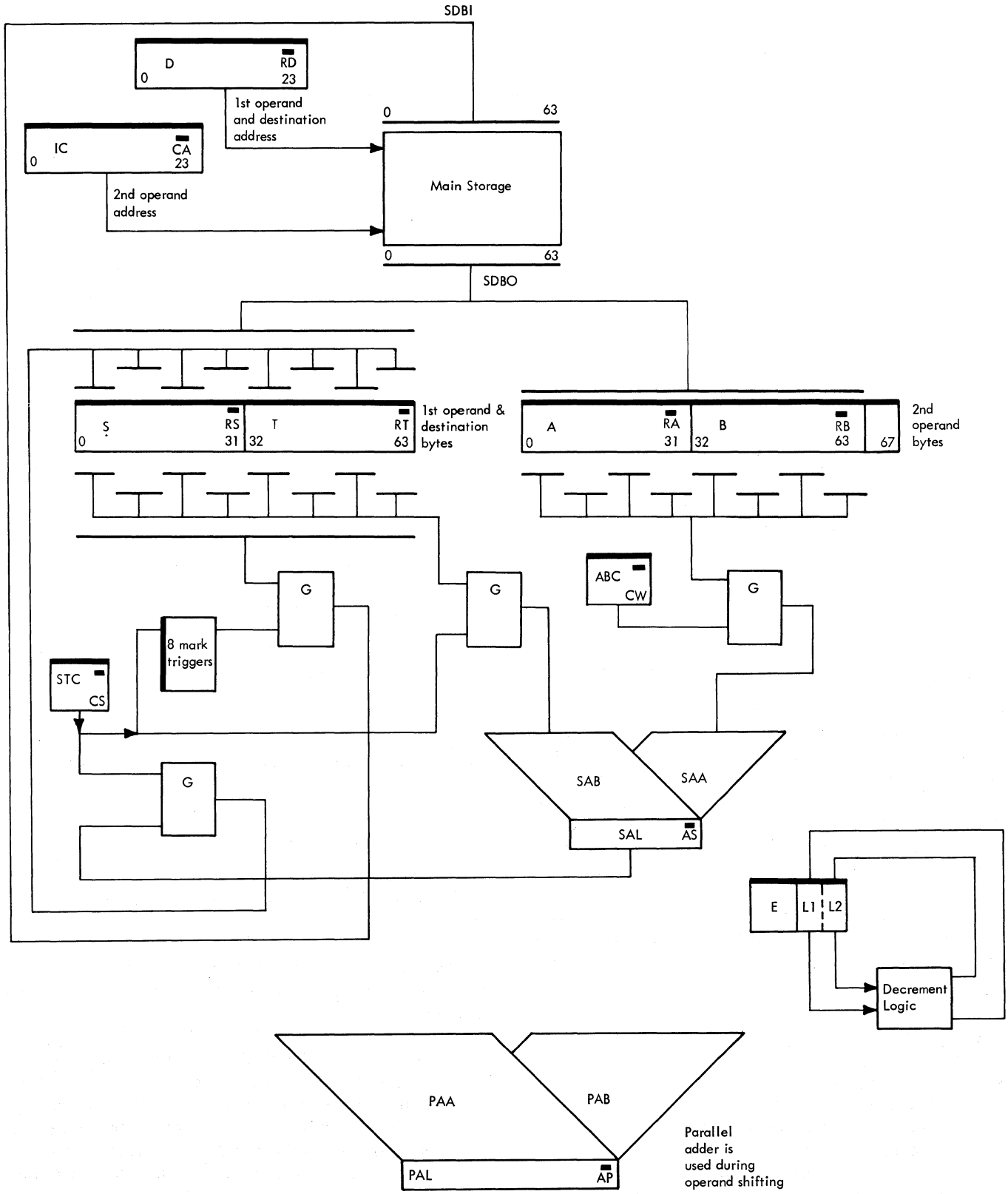


FIGURE 3-7. GENERAL DATA PATH FOR DECIMAL INSTRUCTIONS

Storage of the destination bytes in ST is controlled by the mark triggers. The mark triggers permit alteration of only those bytes in main storage that belong to the field being processed. There is one mark trigger for each of the eight bytes in ST. As a byte of processed data is gated to ST, the corresponding mark trigger is set, thus designating the byte for main storage.

The IC contains the address of the second operand (the instruction address is held in LSWR during execution of SS instructions). The initial IC address specifies the double word containing the rightmost byte of the second operand. When ABC is decremented to zero, all operand bytes in AB have been processed. If additional second operand bytes remain in main storage (L2 count has not stepped to zero), the IC address is decremented by 8 and a fetch is made of the next second operand double word to AB.

This pattern of fetching data, processing via the serial adder, assembling the results in ST, and storing the contents of ST in main storage is continued until either the first of the second operand length field (L1 or L2 count, respectively) has counted below zero. The operation at this point is dependent on the individual instruction. If L2 has been exhausted but not L1, some instructions may require extension of the remaining first operand bytes with high-order zeros. On the other hand, if L1 is exhausted before L2, the instruction may test the remaining second operand bytes for presence of significant digits. This test is performed to detect a possible overflow condition and is accomplished by running the excess second operand bytes through the serial adder and sensing nonzeros. In all cases, if both L1 and L2 counts are exhausted, the instruction execution ends after the last destination word is stored in main storage.

Some decimal operations require use of the parallel adder to perform a right-4 or left-4 shift of the entire operand. The "spilled" bits generated during the shift are held in B(64-67).

3.7.4.2 Initial Conditions

- Conditions prior to GIS:
  - 1st operand is in ST.
  - 2nd operand request is in progress.
  - 1st operand address is in D.
  - 2nd operand address is in IC.
  - Op code and L1, L2 counts are in E.

At the completion of the SS I-Fetch (paragraph 3.2.5), the CPU is in the following status:

1. ST contains the double word from main storage, which includes the low-order bytes of the first operand.

2. An IC request for the second operand, starting with the low-order address, has been issued.

3. The instruction address has been transferred from the IC to the LSWR.

4. D and the IC contain the low-order byte addresses for the first and second operands, respectively.

5. E(0-7) contains the instruction op code. E(8-11) contains the byte count (L1) for the first operand, and E(12-15) contains the byte count for the second operand.

Following the SS I-Fetch, a general initialization sequence (GIS) is performed for all decimal instructions.

3.7.4.3 General Initialization Sequence

- GIS performs following:
  - Setting of STC and ABC.
  - Ingating of SDBO to AB.
  - Sign handling.
  - Word overlap test.

At the completion of SS I-Fetch, a branch is made per the instruction op code to the appropriate GIS microprogram as shown in Figure 3-8. The general functions of the GIS for decimal instructions are described below. Functions peculiar to a specific instruction are covered in subsequent paragraphs dealing with the execution of that instruction. (The GIS for logical instructions is described in Section 5.)

The function of the GIS microprogram is to set up initial conditions for the execution phase. These include:

1. Setting of STC and ABC counters – STC is set to the rightmost first operand byte in ST, the byte to be processed first. Since the address of the rightmost byte is specified by D(21-23), STC is set per these bits. Similarly, the rightmost second operand byte is selected in AB by gating IC(21-23) to ABC.

2. Gating second operand to AB – An IC request for the second operand is issued on the last

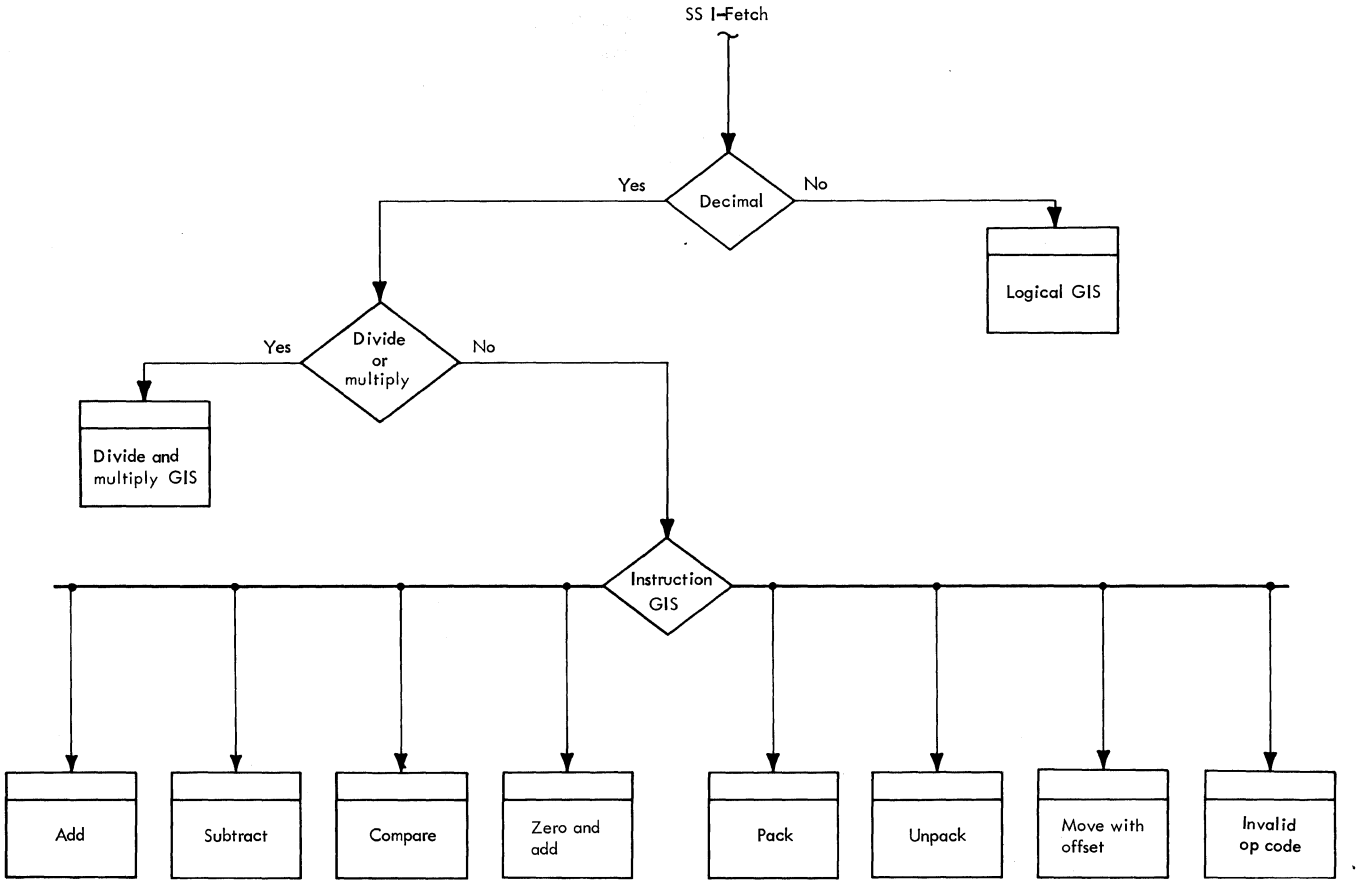


FIGURE 3-8. BRANCHING CONDITIONS AT START OF GIS

cycle of SS I-Fetch. The GIS gates the operand from SDBO to AB.

3. Sign handling – For add-type instructions, the sign of the result is determined prior to the execution phase. The GIS examines the signs in the rightmost bytes of both operands and establishes the sign for the result algebraically. For multiply and divide instructions, the operand signs are recorded by setting the appropriate STAT triggers, and the sign for the result is established during the execution phase.

4. Word overlap test – A word overlap test is performed during the GIS for Pack, Unpack, Move with Offset, and Zero and Add instructions. The purpose of this test is explained in paragraph 3.7.5.1.

3.7.5 INSTRUCTION HANDLING

- Depending on instruction, processing of 1 or 2 operands may be specified.
- Pack, Unpack, Move with Offset, and Zero and Add instructions operate on 1 operand.
- Add, Subtract, Compare, Multiply, and Divide instructions operate on 2 operands.
- All add-type instructions set CC.
- Major serial adder functions used by decimal instructions are:
  - Excess-6 translation.
  - Decimal correction.
  - Complement gating.
  - Cross-gating.
  - Zone or sign insertion.
  - Invalid digit and sign detection.
  - Zero detection.

Decimal instructions may be classified into the general categories of 1- and 2-operand instructions. The 1-operand instructions are Pack, Unpack, Move with Offset, and Zero and Add. The 2-operand

instructions are Add, Subtract, Compare, Multiply, and Divide.

In the 1-operand instructions, the first operand is not processed but is used only as the destination address; the second operand is processed, and the results are placed in the first operand location. The 1-operand instructions are handled by fetching the second operand to AB. Successive AB bytes are selected per ABC and processed in the serial adder, and the resultant bytes entered into ST per STC. After all second operand bytes have been processed, the contents of ST are stored in main storage at the first operand address.

The 2-operand instructions, such as Add, Subtract, and Compare, are executed by fetching the first operand to ST and the second operand to AB. An add or subtract operation is then performed in the serial adder one byte at a time, with the resultant bytes replacing the first operand bytes in ST as they are processed. For the Add and Subtract instructions, the results are stored in main storage at the first operand address. The Compare instruction does not store the result, but performs a test to determine the high, low, or equal relationship of first operand to the second and sets the CC accordingly.

For the 2-operand Multiply and Divide instructions, the operands must be properly aligned in the registers prior to entering execution. This function is performed by the appropriate right- and left-adjust sequences incorporated in the individual microprogram of the instruction.

Basically, the multiply operation is performed by over-and-over addition. The product bytes are developed one byte at a time, starting with the low-order byte. Each time that one byte of product is developed, it is stored in main storage under control of the corresponding mark trigger. The instruction then proceeds to develop the next higher-order product byte. Upon execution of the instruction, the first operand is completely replaced by the product.

The Divide instruction is performed by over-and-over subtraction. It is the only decimal instruction that processes the operands starting with the high-order bytes. The full divisor and a sufficient number of high-order dividend bytes are fetched to perform the first successful subtraction. Then by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient byte is developed. This byte is stored in main storage, and the instruction proceeds to develop the next lower-order quotient byte. Upon execution of the instruction, the first operand is completely replaced by the quotient and the remainder. The remainder occupies the low-order portion of the destination field.

The results of the Add, Subtract, and Compare instructions are used to set the CC. All other decimal instructions leave the code unchanged. The Add and Subtract instructions set the CC to 0, 1, or 2 to indicate a zero, less-than-zero, or greater-than-zero result; the CC is set to 3 if the result of the operation overflows. The Compare instruction sets the CC to 0, 1, or 2 to indicate that the first operand compared equal, low, or high.

The serial adder performs many functions on its input data. The functions of excess-6 translation, decimal correction, and complement gating have been discussed. Additional serial adder functions used by the decimal operations are:

1. Cross-gating — The 2-digit input to the A side of the adder is swapped upon gating to SAL; the digit at adder A side (0-3) is gated to SAL(4-7), and A side (4-7) is gated to SAL (0-3). This function is used mainly by the Pack and Unpack instructions to interchange the sign and digit positions.

2. Zone or sign insertion — The correct zone or sign code (ASCII-8 or EBCDIC) is applied from the ROSDR to the adder A side. The zone or sign may be merged with the digit in any combination.

3. Invalid digit and sign detection — The inputs to the A and B sides of the adder are checked for invalid digits or signs. An appropriate interrupt trigger is set upon detection of an invalid code.

4. Zero detection — This function is used to sense overflow conditions and also to detect all-zero results. An all-zero result placed in main storage must carry a positive sign. Consequently, arithmetic instructions such as Add and Subtract specify checking of each SAL byte for zeros. If upon execution of the instruction it is found that an all-zero result has been stored, the instruction will force storing of a plus sign at the low-order byte address.

3.7.5.1 Word Overlap Condition

- Word overlap condition exists when:
  IC(0-20) = D(0-20) and
  IC(21-23) > D(21-23).

- Test for word overlap is performed by GIS of all 1-operand instructions.

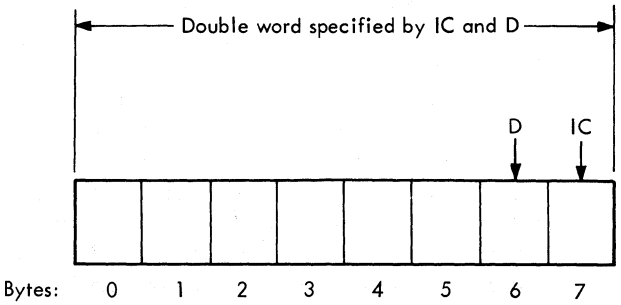
- Execution of 1-operand instructions provides separate microprogram to handle word overlap conditions.

- No special action is taken to detect word overlap in 2-operand instructions.

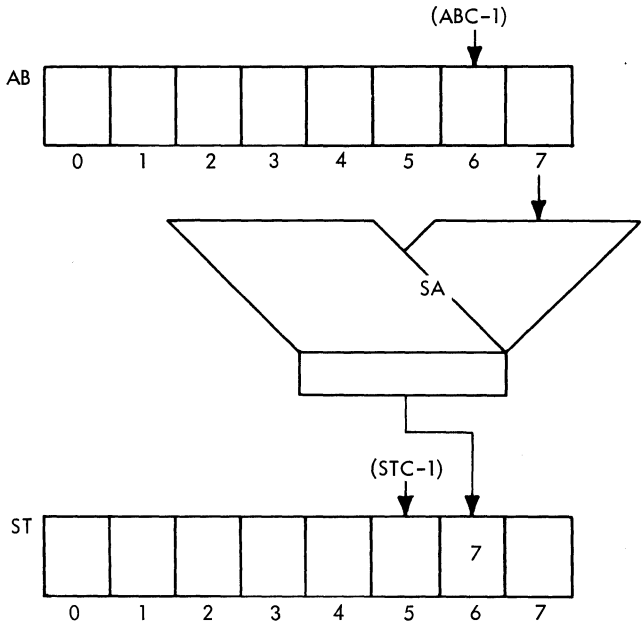
- Word overlap in 2-operand instructions will cause invalid data interruption.

Data is fetched from and placed in main storage one double word at a time. However, program compatibility of the 2065 CPU with smaller models in the System/360 requires that all results placed in main storage must be considered to be stored one byte at a time as they are processed. There are some cases where this compatibility would not be maintained unless special actions were taken. The condition that requires special handling is called "word overlap" and occurs when the fields of the first and second operands specified by the instruction overlap.

The operand addresses and field lengths may be such that one or more bytes in main storage are specified as part of both the first and the second operands. For example, consider the case in which the IC and D specify the same double word in storage; the IC specifies byte 7 as the starting second operand byte to be processed in this double word, and D specifies byte 6 as the starting first operand address. At least two operand bytes are to be processed.



This double word is fetched from main storage and placed in AB and ST. A single operand instruction, such as Move with Offset, will process the first AB byte (byte 7) in the serial adder and place the result in the designated first operand byte; i.e., byte 6 in ST. Then, ABC and STC are reduced 1 count to designate the next AB byte to be processed and the ST location in which the results must be placed.



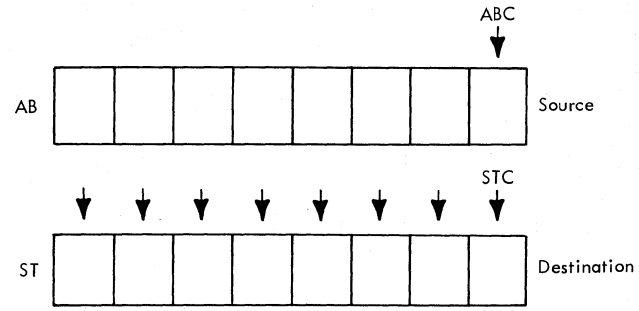
Note, however, that the preceding move operation has replaced the original contents of byte 6 in ST with the contents of byte 7. Thus, the next AB byte to be processed (original byte 6) is no longer valid and must be updated; i.e., the equivalent of storing ST byte 6 and then refetching this byte to AB must be performed.

As seen from the preceding example, the word-overlap condition may require special handling of data. Execution of all 1-operand decimal instructions (Pack, Unpack, Move with Offset, and Zero and Add) provides for two alternate microprograms. One microprogram is for the normal, or not-word-overlap, case; the other handles the word-overlap condition. Selection of the appropriate microprogram is dependent on the outcome of the word-overlap test, which is performed in the GIS of all 1-operand instructions.

A word-overlap condition exists when both operands have the same double-word address. The manner in which the first and second operand bytes are specified within this double word determines whether special data handling is required. When

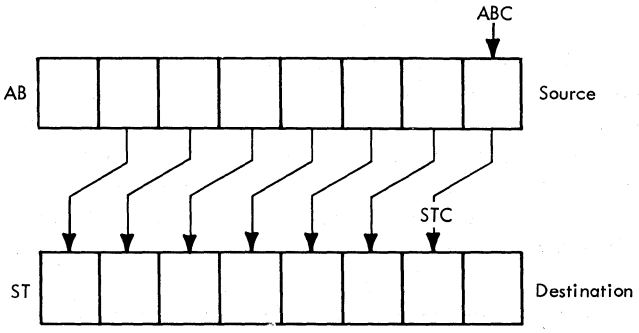
the word-overlap condition exists, three cases of byte specification may be distinguished:

1. The first and second operand bytes are the same - no special data handling required.*



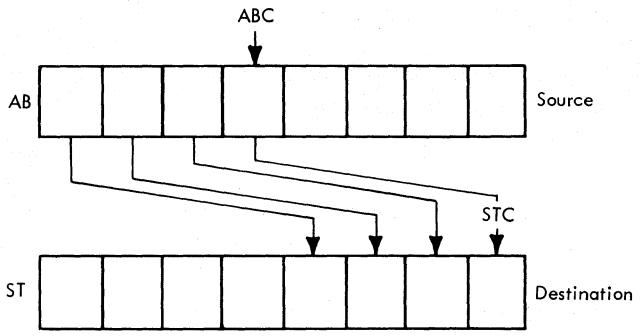
In this case, the destination bytes are placed in the same locations from which the source bytes are obtained. Since processing of any source byte does not affect the contents of the next source byte, no updating of source bytes is necessary.

2. The first operand bytes are specified "ahead" of the second operand bytes — special data handling required.



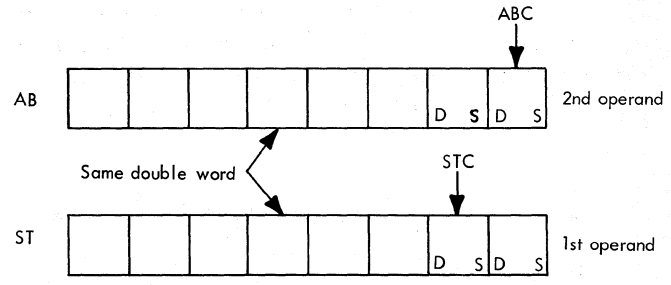
In this case, the destination bytes are placed in the locations from which the next source bytes will be processed. The data in AB becomes "obsolete" after processing of one or more source bytes. (The cross-over point at which data becomes obsolete depends on the amount of skew between ABC and STC.) This is a case of word overlap which requires special data-handling techniques.

3. The first operand bytes are specified "behind" the second operand bytes — no special data handling required.



In this case, the destination bytes are placed behind the source bytes as they are processed. Thus processing of any source byte cannot affect the contents of the next source byte, and no updating is necessary.

In 2-operand arithmetic instructions, no special action is taken to detect word overlap. Word overlap is completely ignored during execution of a Compare instruction, since this instruction does not store the results. The operand fields specified for Add, Subtract, Multiply, and Divide instructions either should not overlap at all or should have coincident rightmost bytes. The GIS for these instructions does not perform the word overlap test, because improper overlap of the operands will cause an invalid data condition to be detected in the execution phase. In 2-operand instructions, the operand fields are correctly specified when the rightmost byte of each operand contains the operand sign; all bytes to the left of the sign byte must contain only digits. This requirement cannot be fulfilled when both operands in the instruction specify the same double word with different byte addresses. From the following example, it can be seen that the sign byte of the first operand is also the "digit" byte of the second.



During execution of the instruction, all operand digits are checked for validity. Detection of a sign code in the digit position will force an invalid data interruption.

* Except for Unpack instruction. This instruction generates two bytes of destination for each byte of source and requires special data handling in all cases of word overlap.

3.7.5.2 Interruption Conditions

- Protection.
- Addressing.
- Specification.
- Data.
- Decimal divide check.
- Decimal overflow.

Exceptional conditions, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in decimal instructions:

1. Protection – The storage key does not match the protection key in the PSW.

2. Addressing – An address designates a location outside the available storage for the installed system.

(In the above two exceptions, the operation is terminated. The result data and the CC are unpredictable and should not be used for further computation.)

3. Specification – A multiplier or a divisor size exceeds 15 digits and sign or the multiplicand or dividend size. The instruction is suppressed.

4. Data – A sign or digit code of an operand specified in the Add, Subtract, Compare, Zero and Add, Multiply, or Divide instruction is incorrect, a multiplicand has insufficient high-order zeros, or the operand fields in these instructions overlap.

5. Decimal Divide Check – The quotient exceeds the specified data field, including division by zero. Division is suppressed. Therefore, the dividend and divisor remain unchanged in storage.

6. Decimal Overflow – Execution of the Add, Subtract, or Zero and Add instruction results in an overflow condition. The program interruption occurs only when the decimal-overflow mask bit is a 1. The operation is completed by placing the truncated low-order result in the result field and setting the CC to 3. The sign and low-order digits contained in the result field are the same as they would have been for an infinitely long result field.

3.8 INSTRUCTION EXECUTION

The following paragraphs describe the execution sequences for the instructions in the decimal set. Table 3-17 lists the instructions with their respective mnemonic and op codes, and indicates the exceptions that cause a program interruption for each instruction. Table 3-18 lists those instructions that affect the CC and indicates how the CC is set. All instructions use the SS format and assume packed operands and results. The only exceptions

TABLE 3-17. DECIMAL INSTRUCTION SET

Instruction	Mnemonic	Op Code	Interruptions*
Add	AP	FA	P, A, D, DF
Subtract	SP	FB	P, A, D, DF
Compare	CP	F9	D
Zero and Add	ZAP	F8	P, A, D, DF
Multiply	MP	FC	P, A, D, S
Divide	DP	FD	P, A, D, S, DK
Pack	PACK	F2	P, A
Unpack	UNPK	F3	P, A
Move with Offset	MVO	F1	P, A

A - Addressing
D - Invalid data
DF - Decimal overflow
DK - Divide check
P - Protection
S - Specification

TABLE 3-18. CONDITION CODES FOR DECIMAL INSTRUCTIONS

Instruction	Condition Code			
	0	1	2	3
Add	Result = 0	Result < 0	Result > 0	Overflow
Subtract	Result = 0	Result < 0	Result > 0	Overflow
Compare	Equal	Low	High	-
Zero and Add	Result = 0	Result < 0	Result > 0	Overflow

are Pack, which has a zoned operand, and Unpack, which has a zoned result.

3.8.1 ADD, SUBTRACT, AND COMPARE INSTRUCTIONS

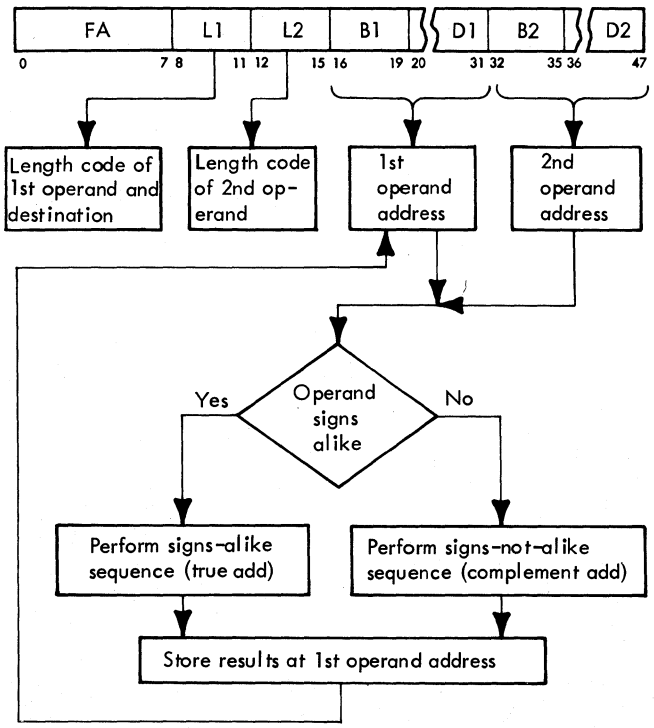
- All three instructions share common ROS microprogram.

During the GIS, separate sign handling is performed for each instruction; the sign of the second operand is effectively inverted for Subtract and Compare instructions. After exit from the GIS, all three instructions share a common addition or subtraction routine, depending on whether the signs of the operands are alike or unlike. Since the results of a Compare instruction are not stored in main storage, the setting of the mark triggers and overflow detection are inhibited in hardware during execution of this instruction.

3.8.1.1 Add, AP (FA)

- 2nd operand is added to 1st operand, and sum is placed in 1st operand location.

- SS format:



- CC setting:
  Result is zero: CC = 0
  Result is less than zero: CC = 1
  Result is greater than zero: CC = 2
  Overflow: CC = 3

- Interruptions:
  Protection.
  Addressing.
  Invalid data.
  Overflow.

The AP instruction specifies an algebraic addition. The operand signs are examined, and the sign of the sum is established algebraically. Then, if the operand signs are alike, an add sequence is performed; if they are not alike, the instruction performs a subtract sequence. All signs and digits are checked for validity. The first and the second operand fields may overlap when their low-order bytes coincide; therefore, it is possible to add a number to itself.

The resultant sum is stored at the first operand address. A zero sum is always stored with a positive sign. If the first operand field is too short to contain all significant digits of the sum, a decimal overflow occurs. Overflow has two possible causes. The first is the loss of a carry out of the high-order digit position of the result field. The second cause is an oversized result, which occurs when the second operand field is larger than that of the first operand and significant result digits are lost. The field sizes alone are not an indication of overflow.

3.8.1.1.1 GIS for Add Instruction

- Conditions at start of GIS:
  1st operand is in ST.
  STC is set at lowest-order operand byte.
  IC request is issued for 2nd operand.

- GIS:
  Transfers L1 to F(0-3).
  Loads 2nd operand in AB.
  Sets STAT F if 1st operand is minus.
  Sets STAT C if 2nd operand is minus.
  Branches per STAT's to add or subtract routine.

The GIS microprogram for the Add instruction is shown in Figure 6046, FEDM. This microprogram

transfers the L1 count to F(0-3), loads the second operand in AB, assigns the algebraic sign to the sum, and performs a branch on operand signs to enter either the add or subtract sequence.

The L1 count in E(8-11) is destroyed during subsequent execution and must be preserved in F(0-3). This action is necessary because, upon execution of the instruction, it may be found that results were placed in main storage in complement form. Since the final result must be true, the destination field is refetched and recomplemented. In such cases, the L1 count in F(0-3) is used to refetch the correct number of destination bytes.

The sum is arbitrarily assigned the sign of the first operand by performing a branch of STAT's F and C. STAT F is set if the first operand is minus. Note, however, that the sign of the second operand is not known at this time and STAT C will always be in the reset state. Thus, when STAT's F and C are alike, it indicates that STAT F is not set (first operand plus); when the STAT's are not alike, it indicates that STAT F has been set and the first operand is negative.

A second branch on STAT's F and C is performed after the sign of the second operand has been sensed, and STAT C set accordingly. If the STAT's are alike, an add sequence is entered. Upon entry into this sequence, the sum always carries the correct sign; the sum was assigned the sign of the first operand, and both operands have the same signs. If the STAT's are not alike, a subtract sequence is entered. In this case, the algebraic sign of the sum cannot be known at the start of the operation since it is dependent on the relative magnitude of the operands. If the sign has been assigned incorrectly, the result of the subtract operation will be complement form. This condition will be detected at the completion of the ADD instruction, in which case the result will be recomplemented and the correct sign inserted.

3.8.1.1.2 Signs-Alike Sequence, Add

- True +6 add operation will exit on one or more of following conditions:
  L1 or STC = 0
  L2 = 0
  ABC = 0
- STAT A set if result not zero.
- STAT B set if overflow.
- STAT E set if operand digit (or sign) invalid.
- STAT G set if Compare instruction.
- STAT H set if carry to next byte.

An overall flow chart of the signs-alike, or add, sequence and the data path used for its execution are shown in Figure 6047, FEDM. The flow chart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into the add sequence, the signs of both operands have been examined (by the GIS) and the correct sign has been entered in the low-order destination byte in ST. The first step in the microprogram is to add the digits contained in the sign bytes of the operands. The resultant sum is then placed in the digit portion of the destination byte. At this point, one complete byte of results has been developed. The operand length codes (L1 and L2) and the status of byte counters (STC and ABC) are examined for one or more of the following conditions:

1. The result byte is contained in the last byte of ST and must be stored (STC = 0).
2. Additional first operand bytes must be fetched from main storage (STC = 0 and L1 $\neq$ 0).
3. Additional second operand bytes must be fetched from main storage (ABC = 0 and L2 $\neq$ 0).
4. The second operand has run out; i.e., all second operand bytes have been processed, and zeros must be added to the first operand bytes (L2 = 0 but L1 $\neq$ 0).
5. The first operand has run out, i.e., the destination field has been completely processed (L1 = 0).

If none of the above exit conditions exist, the microprogram enters the add loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB and ST bytes are added in the serial adder, and the resultant sum replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit conditions, the operations dictated by the conditions are performed,

and if L1 is not zero, the add loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test and a test for an all-zero result.* If an all-zero result has been obtained, the address of the low-order destination byte is restored in D and a plus is stored at this address. Restoration is necessary, since D is decremented by 8 for each double word of first operand that is fetched. If, for example, two double words of first operand have been fetched, the address of the low-order destination byte is obtained by adding 16 to D.

A detailed flow chart of the signs-alike, or add, sequence is shown in Figure 6048, FEDM. It is an expanded version of the overall flow chart (Figure 6047, FEDM), showing the data handling used in the various subroutines of the add operation. This data handling is straightforward for the most part and requires no explanation. Those areas in need of clarification are discussed below:

1. Add Operation

   The selected AB byte is gated (true +6) to the serial adder and added as a binary number to the selected ST byte. The adder output is decimal-corrected at the input to SAL and gated back to the selected ST byte. For the first (or sign) byte, only bits 0-3 of the selected AB byte are gated to the adder. The decimal correction involves examining the carry from each digit and logically subtracting 6 from each result digit that did not have a carry. As each byte is processed, ABC, STC, L1, and L2 are decremented by 1 and the selected mark trigger is set (except for a Compare instruction). The carry from each byte is saved in STAT H and, if set, results in a carry to bit 7 of the next byte processed. STAT A is set if any non-zero result digit is detected. STAT E is set if any invalid digit is detected at the inputs to the serial adder.**

2. Exit Conditions

   An exit is made when one or more of the following conditions exist as determined by a functional branch micro-order (DECIMAL):

   L1 or STC equals zero.
   L2 equals zero.
   ABC equals zero.

   Since any combination of the above may exist, there are seven possible exits:

   a. L1 or STC = 0, L2 $\neq$ 0, ABC $\neq$ 0.

      If an Add or Subtract instruction is being executed, a storage request per D stores the results in ST in the destination field of main storage. For a Compare instruction, since no mark triggers are set, the D request fetches the next double word of first operand. A further test, L1 equal all 1's, is required to determine whether the L1 field is exhausted. If not, a first operand fetch sequence is started, after which the add loop is resumed.

   b. L1 or STC = 0, L2 $\neq$ 0, ABC = 0.

      Same as a, except that a second operand fetch is made after the first operand fetch prior to entering the add loop.

   c. L1 or STC = 0, L2 = 0, ABC $\neq$ 0.

      A storage request per D is issued to store ST in the detination field (unless a Compare instruction is being executed). AB and ABC are cleared to start the high-order source extend routine. A further test is required to determine whether L1 was zero.

      If L1 is now all 1's, all destination bytes have been processed. A carry from the last destination byte indicates an overflow condition, and STAT B is set.

      If L1 is not all 1's, a first operand fetch sequence is started, after which the high-order source extend routine is resumed.

   d. L1 or STC = 0, L2 = 0, ABC = 0.

      Same as c.

   e. L1 or STC $\neq$ 0, L2 = 0, ABC = 0.

      The second operand field has been completely processed. AB and ABC are

---

* The Add, Subtract, and Compare instructions have the same microprogram. The Compare instruction does not store results in main storage and, upon exit from the add loop, enters the end-op sequence.

** When an invalid digit is detected at the serial adder in-buses, the adder forces 1's into its sum and parity output latches. This action insures that valid parity is always gated to ST from the serial adder.

cleared, and the high-order source extend routine is started to process the remaining destination field.

f. L1 or STC $\neq 0$, L2 = 0, ABC $\neq 0$.

Same as e.

g. L1 or STC $\neq 0$, L2 $\neq 0$, ABC = 0.

A second operand fetch sequence is started to fetch the next double word of second operand to AB.

3. First Operand Fetch

A separate entry is made into this routine, per STAT G, for a Compare instruction. In a compare operation, a D request for the next double word of first operand has already been given. D is decremented by 8, and the double word arriving at the SDBO is gated to ST.

For Add or Subtract instructions, D is decremented by 8, and a D request is made for the next double word of first operand. F is incremented by 1 to record the number of fetches made. This information will be required to restore the low-order address in D in the event that an all-zero result is obtained. If ABC equals 111, a second operand fetch routine is started. If ABC does not equal 111, the appropriate addition or high-order source extend is started.

4. Second Operand Fetch

The IC is decremented by 8, and the next double word of second operand is fetched to AB. After this, the appropriate addition or overflow routine is started as determined by the L1 count.

5. Second Operand Runout

An all-zeros AB byte is gated true +6 to the serial adder and added to the selected ST byte. The result is decimal-corrected and gated back to the selected ST byte. STAT's A, E, and H and the mark triggers are set as previously explained.

L1 and STC are decremented by 1 as each byte is processed; ABC and L2 are not stepped. The sequence is repeated until L1 is stepped to zero, with an exit to the destination store, and first operand fetch sequences whenever STC equals 7. A carry from the last destination byte is an overflow condition and sets STAT B. The end-op sequence is started when L1 equals zero.

6. First Operand Runout and Overflow Test

An overflow condition exists whenever a carry results as the last destination byte is processed or whenever a nonzero digit is detected in the source field after the destination field has been processed. STAT B is set if STAT H is set when entering this routine.

Next, the remaining second operand bytes are gated true +6 to the serial adder with STAT B being set if any nonzero bytes are detected.

ABC and STC are decremented by 1 as each byte is processed. The next source double word is fetched to AB whenever ABC is stepped to zero unless L2 equals zero. When L2 is stepped to zero, the end-op sequence is started.

7. Zero Result

If at the completion of the add operation STAT A is not set, an all-zero result has been obtained. In this case, the Add and Subtract instructions will always force a positive sign in the low-order byte of the destination field. (If STAT G is set, an exit is made to the end-op sequence since no correction of the result is required for a Compare instruction.)

The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4-7). STC is set per D(21-23), and the selected ST byte is cleared.

STAT B is examined to determine overflow condition. For a zero result and no overflow, a plus sign is inserted via the serial adder in the low-order destination byte with the selected mark trigger being set. A storage request is given to store the sign in the destination field, and the end-op sequence is started. For a zero result and overflow, the destination sign is not corrected. The end-op sequence is started immediately.

8. End-Op Sequence

The instruction address (original content of the IC) is restored from the LSWR to the IC,

and STAT G is reset. An invalid data interruption occurs if STAT E is set. An overflow interruption occurs if STAT B is set and STAT E is not set. The CC is set per hardware conditions as shown in Table 3-19.

TABLE 3-19. CONDITION CODE SETTING PER HARDWARE CONDITIONS, DECIMAL INSTRUCTION SET

Hardware Conditions	Setting
STAT A • STAT F • (Add, or Subtract, or Zero and Add)	→ 01
STAT A • STAT F • STAT H • Compare	→ 01
STAT A • STAT C • not STAT H • Compare	→ 01
STAT F • STAT C • STAT H • Compare	→ 01
STAT A • not STAT F • (Add, or Subtract, or Zero and Add)	→ 10
STAT A • not STAT F • STAT H • Compare	→ 10
STAT A • not STAT C • not STAT H • Compare	→ 10
STAT H • not STAT F • not STAT C • Compare	→ 10
STAT B • (Add, or Subtract, or Zero and Add)	→ 11

Note: • Designates logical AND connective.

3.8.1.1.3 Signs-Not-Alike Sequence, Subtract

- Complement add operation with exit on one or more of following conditions:
  L1 or STC = 0
  L2 = 0
  ABC = 0
- STAT A set if result not zero.
- STAT B set if overflow.
- STAT D set if result must be re-complemented.
- STAT E set if operand digit or sign invalid.
- STAT G set if Compare instruction.
- STAT H set if carry to next byte.
- Carry out of last destination byte indicates that result is in true form; no carry condition indicates that result is in complement form.

An overall flow chart of the signs-not-alike, or subtract, sequence is shown in Figure 6049, FEDM. This flow chart outlines the major functional steps and sequences used in the Add, Subtract, and Compare microprogram.

Upon entry into this sequence, the signs of both operands have been examined (by the GIS) and the sign of the first operand has been inserted as the sign of the result. This sign may or may not turn out to be the correct sign: if the first operand is larger than the second, the result carries the correct sign; if the reverse is true, the sign of the result must be corrected.

Basically, the subtract microprogram is similar to the add sequence previously described. The first step in the microprogram is to subtract the digits contained in the sign bytes of the operands. The resultant difference is then placed in the digit portion of the destination sign byte. At this point, one complete byte of results has been developed. The operand length codes (L1 and L2) and the status of byte counters (STC and ABC) are examined for one or more of the following conditions:

1. The result byte is contained in the last byte of ST and must be stored (STC = 0).
2. Additional first operand bytes must be fetched from main storage (STC = 0 and L1 $\neq$ 0).
3. Additional second operand bytes must be fetched from main storage (ABC = 0 and L2 $\neq$ 0).
4. The second operand has run out; i.e., all second operand bytes have been processed, and zeros must be added to the first operand bytes (L2 = 0 but L1 $\neq$ 0).
5. The first operand has run out; i.e., the destination field has been completely processed (L1 = 0).

If none of the above conditions exist, the microprogram enters the subtract loop to generate the next destination byte. L1, L2, STC, and ABC are decremented one count, the selected AB byte is subtracted from the selected ST byte, and the resultant difference replaces the selected ST byte. After this, the status of all counters is again sensed for exit conditions.

Upon establishing one or more exit condtions, the operations dictated by the conditions are performed and, if L1 is not zero, the subtract loop is re-entered. When L1 is zero, all destination bytes have been processed. The microprogram then performs an overflow test, a zero-result test, and a complement result test.* If an all-zero or complement result has been generated, the address of the low-order destination byte is restored in D. Then, the result is either recomplemented or a plus is stored in the low-order destination byte.

A detailed flow chart of the signs-not-alike, or subtract, sequence is shown in Figure 6050, FEDM. It is an expanded version of the overall flow chart (Figure 6049, FEDM), showing the data handling in the various subroutines of the subtract operation. The subtract operation is similar to the add sequence described in paragraph 3.8.1.1.2. For this reason, only the differences are discussed below.

1. Subtract Operation

   The selected AB byte is converted to 2's complement form at the input to the serial adder. The 2's complement of the byte is then added to the selected ST byte. For the first (or sign) byte, bits 0-3 only of the selected AB byte are gated complement to the adder, with a hot carry supplied to bit 3 to convert to 2's complement.

2. Second Operand Runout

   An all-zeros AB byte is gated complement to the serial adder and added to the selected ST byte. Thus the second operand is extended with high-order 1's.

3. Overflow Test

   Generally, an overflow condition exists if, upon processing all destination bytes, a nonzero source byte is detected. One exception occurs when the first source byte sensed, after the first operand has run out, equals 1 and a "borrow" condition exists. A borrow condition is determined by STAT's A and H being set; i.e., a nonzero result and a carry from the previous byte.

   When L2 has been stepped to zero, a carry from the last destination byte is examined. A carry condition indicates a true result, and the end-op sequence is started. No carry indicates a complement result, and a recomplement sequence is started for Add and Subtract instructions. For a Compare instruction (STAT G set), the end-op sequence is started immediately.

4. Zero Result and Recomplement Setup

   STAT D is set when an entry is made to this routine because of the result's being in complement form. STAT D is not set when an entry is made because of zero result. If STAT G is set when entering this routine, an exit is made to the end-op sequence since no corrections of the result are required for the Compare instruction.

   The low-order destination address is regenerated by adding 8 to D the number of times indicated in F(4-7). STC is set per D(21-23), and the selected ST byte is cleared. If an overflow condition exists (STAT B set), the results need not be corrected and the end-op sequence is started immediately.

5. Recomplement Sequence

   The original L1 count was saved in F(0-3) by the GIS. This count is now placed in the L2 location in E; i.e., E(12-15). The L1 location in E(8-11) is set to zero and then decremented one count to provide an exit from the first operand fetch routine to the recomplement sequence. The complement result is gated from the SDBO to AB, and the recomplement sequence is started.

   The sign byte is processed by gating bits 0-3 of the selected AB byte complement to the serial adder, with a hot carry supplied to bit 3. A plus or minus sign is inserted in serial adder bits 4-7 as determined by the sign of the second operand (STAT C). Bits 0-3 are decimal-corrected, and the adder output is gated to ST.

   All bytes following the sign byte are processed by gating the selected AB byte complement to the serial adder, where it is added to an all-zero ST byte. The adder output is decimal-corrected and gated back to the selected ST byte.

---

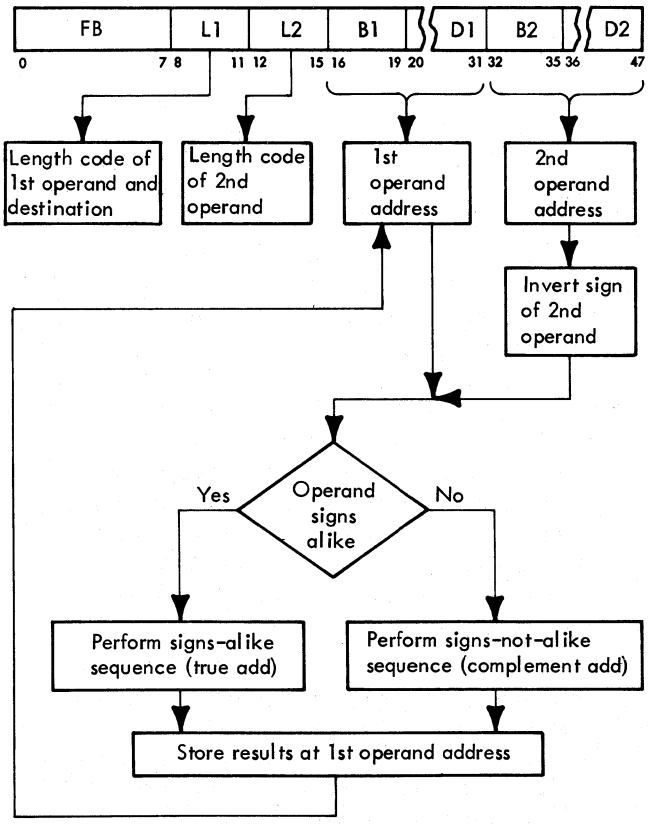
* The Compare instruction does not store results in main storage and, upon exit from the subtract loop, enters the end-op sequence.

As each byte (including the sign byte) is processed, the ABC, STC, and L2 counts are decremented. The mark trigger selected by the STC is set. The serial adder carry is saved in STAT H. STAT A is set on non-zero digits, and STAT E is set on invalid digits.

Recomplementation is continued until the L1 in E(12-15) is stepped to zero. If ABC steps to zero and L1 is not zero, ST is stored and the next double word of destination is fetched to AB. When L1 steps to zero, ST is stored in the destination field, AB is cleared, and STAT F is set or reset per STAT C. The CC is set per hardware conditions (see Table 3-19), and the instruction is ended.

3.8.1.2 Subtract, SP (FB)

- 2nd operand is subtracted from 1st operand, and difference is place in 1st operand location.
- SS format:

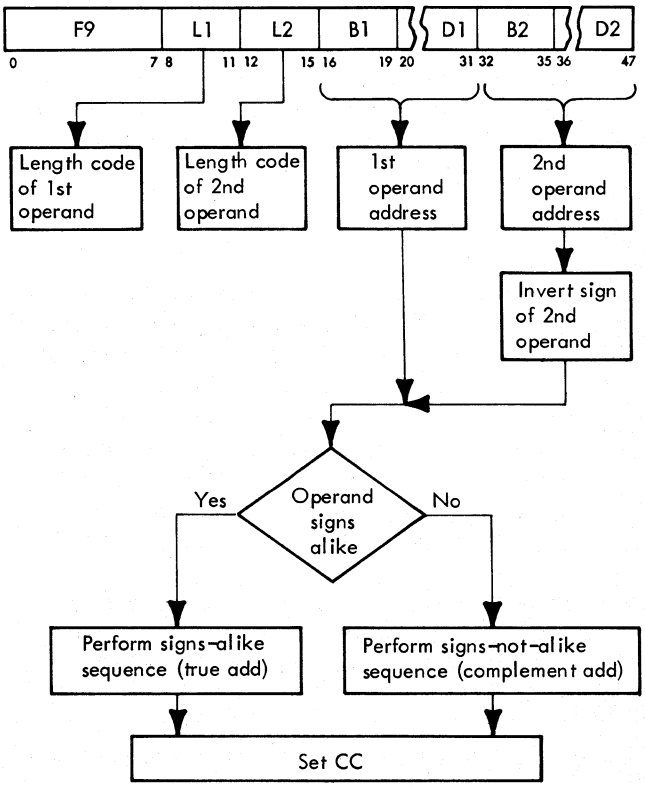


- CC setting:
  Result is zero: CC = 0.
  Result is less than zero: CC = 1.
  Result is greater than zero: CC = 2.
  Overflow: CC = 3.
- Interruptions:
  Protection.
  Addressing.
  Invalid data.
  Overflow.

The SP instruction specifies an algebraic subtraction of the second operand from the first operand. The instruction shares the same add or subtract routines used by the Add instruction. During the GIS of the Subtract instruction, however, the sign of the second operand is effectively inverted to perform subtraction. The GIS microprogram for the subtract instruction is shown in Figure 6051, FEDM. Upon exit from the GIS, the signs-alike sequence (paragraph 3.8.1.1.2) or the signs-not-alike sequence (paragraph 3.8.1.1.3) is entered, depending on the operand signs.

3.8.1.3 Compare, CP (F9)

- 1st operand is compared with 2nd operand, and CC is set to indicate results of comparison:
  Operands equal: CC = 0.
  1st operand low: CC = 1.
  1st operand high: CC = 2.
- SS format:



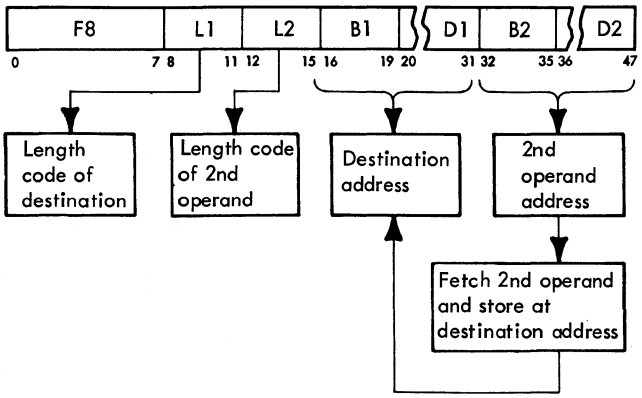
- Interruptions:
  Addressing.
  Invalid data.

The CP instruction shares the same add and subtract routines used by the Add and Subtract instructions and described in paragraphs 3.8.1.1.2 and 3.8.1.1.3. The GIS microprogram for the Compare instruction is shown in Figure 6052, FEDM. As in the Subtract instruction, this microprogram effectively inverts the sign of the second operand by setting STAT C on positive sign. The results of the compare operation are not placed in main storage. STAT G is set to provide a means of taking special action, where required for the Compare instruction, during execution of the common add or subtract sequences.

3.8.2 ZERO AND ADD, ZAP (F8)

- 2nd operand is placed in 1st operand location.

- SS format:



- CC setting:
  2nd operand is zero: CC = 0.
  2nd operand is less than zero:
  CC = 1.
  2nd operand is greater than zero:
  CC = 2.
  2nd operand cannot fit in destination field: CC = 3.

- Interruptions:
  Protection.
  Addressing.
  Invalid data.
  Overflow.

The operation specified by the ZAP instruction is equivalent to addition to zero. A zero result is always made positive. When high-order digits are lost because of overflow, a zero result has the sign of the second operand.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied if needed. When the first operand field is too short to contain all significant digits of the second operand, a decimal overflow occurs and results in a program interruption, provided that the decimal overflow mask bit is 1. The first and second operand fields may overlap when the rightmost byte of the first operand field is coincident with or to the right of the rightmost byte of the second operand.

A detailed flow chart of the GIS and execution of the Zero and Add instruction is shown in Figure 6053, FEDM.

At the start of the GIS, the following actions have been performed by SS I-Fetch: (1) the STC has been set at the low-order destination byte, and (2) an IC request has been issued for the second operand. During the GIS, a word overlap test is performed by comparing the source address, the destination address, and the source length code to determine whether a word-overlap condition exists, as defined in paragraph 3.7.5.1. If a word-overlap condition is predicted by this test, the instruction address is restored to the IC, the invalid-data-interrupt trigger is set, and the instruction is ended.

The sign byte is processed by gating bits 0-3 of the selected AB byte to the serial adder. Bits 4-7 of the AB byte are decoded for a positive, negative, or invalid sign. The approved plus or minus sign is inserted in SAL(4-7) and gated, with the digit, to the selected ST byte.

All bytes following the sign byte are processed by gating the selected AB byte true +6 to the serial adder. The selected ST byte is not gated to the adder, and the validity check at the adder B side is inhibited in hardware. The adder output is decimal-corrected and gated to the selected ST byte.

As each byte is processed, including the first byte, ABC, STC, L1, and L2 are decremented, the selected mark trigger is set, STAT E is set for invalid data, and STAT A is set for a nonzero digit.

The byte-by-byte transfer from AB to ST is continued until one or more of the following exit conditions are detected via an ROS branch (DECIMAL): L1 or STC equals zero, L2 equals

zero, and ABC equal zero. Depending on the exit conditions, the following actions may be performed:

1. L1 and L2 = 0

   a. The contents of ST are stored in the destination field. Note that F is incremented each time that a store operation is performed. This action is taken to enable regeneration of the low-order destination byte in case an all-zero result is detected in the end-op sequence.

   b. AB and ABC are cleared to zero.

   c. STAT F is set or reset per STAT C to enter the same end-op routine as that used by the signs-alike add sequence. (See Table 3-19 for CC setting.)

2. L1 = 0

   a. The contents of ST are stored in the destination field.

   b. STAT F is set or reset per STAT C.

   c. An overflow test is performed.

   d. If ABC is also zero, the next double word of 2nd operand is fetched prior to entering the overflow test.

3. L2 = 0

   a. AB and ABC are cleared to zero.

   b. The second operand is extended with high-order zeros.

   c. If STC is also zero, the contents of ST are stored in the destination field prior to entering the high-order zeros routine.

4. STC and ABC = 0

   a. The contents of ST are stored, and a fetch of the next double word of second operand is made to AB.

   b. The zero and add loop is resumed.

5. STC = 0

   The contents of ST are stored, and the zero and add loop is resumed.

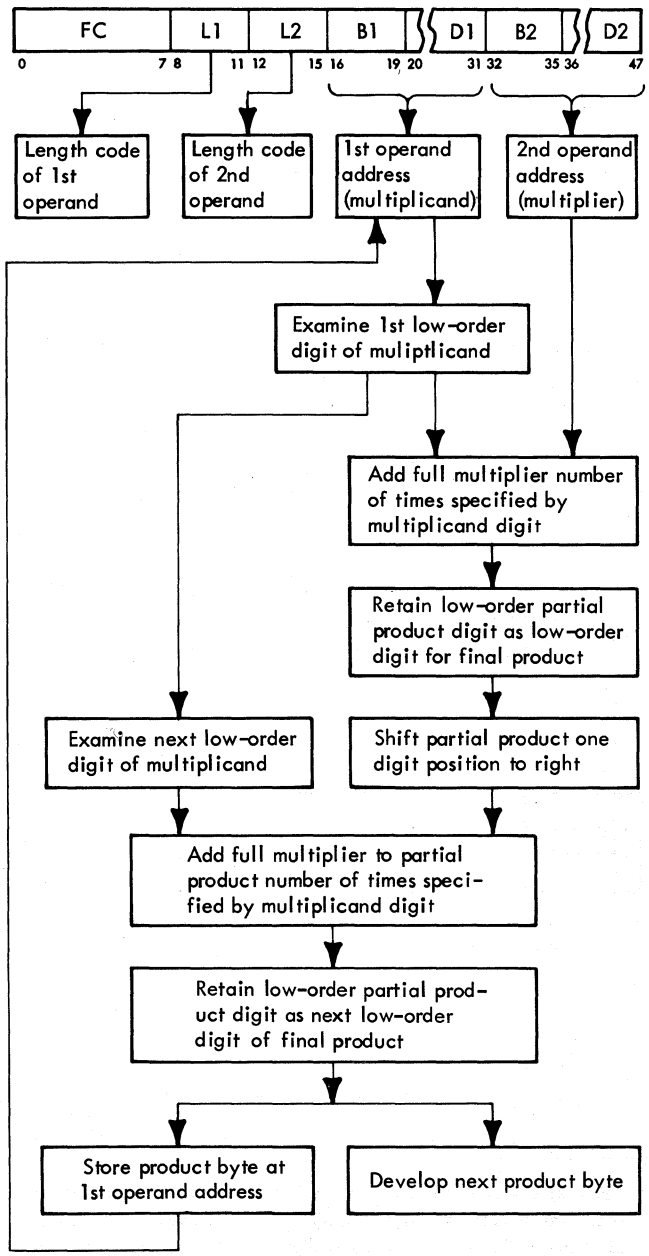
6. ABC = 0

   The next double word of second operand is fetched to AB, and the zero and add loop is resumed.

3.8.3 MULTIPLY, MP (FC)

- Product of multiplicand (1st operand) and multiplier (2nd operand) replaces multiplicand.

- SS format:



3.8.3.1 Introduction

- Maximum multiplicand field is 16 bytes.
- Maximum multiplier field is 8 bytes.
- Multiplicand field initially contains high-order zero field equal in length to multiplier field.
- L2 > 7 or L2 ≧ L1 will cause specification interruption.
- Multiplication accomplished by over-and-over addition or subtraction:

Multiplicand Digit	Sequence Selected
0	—
1 through 4	Addition
5 through 9	Subtraction

The MP instruction replaces the multiplicand (1st operand) with the product of the multiplicand and the multiplier (2nd operand). To be able to store the product in the multiplicand field at all times, several restrictions are imposed on both the multiplicand and the multiplier.

1. In any multiply operation, the maximum number of product digits that can be obtained is equal to the sum of the digits in the two operands. Since the product is stored in the multiplicand field, this field must initially contain high-order zero digits for at least a field size equal to that of the multiplier. Thus the multiplicand field is initially split into two parts; the high-order zero field of length equal to the multiplier and the low-order field containing the effective multiplicand digits. This arrangement of the multiplicand ensures that product overflow will not occur (Figure 3-9).

2. By definition, the multiplier field must be at least one digit less than the multiplicand. Since the multiplicand must initially contain a zero field equal in size to the multiplier digits, the multiplier size is limited to 8 bytes (15 digits and sign). A specification interruption occurs if the multiplier length code designates more than 8 bytes (L2 is greater than 7), or if L2 is greater than or equal to L1.

3. The maximum product size is 31 digits and sign (16 multiplicand digits plus 15 multiplier digits). The sign is determined algebraically from the multiplier and multiplicand signs, even if one or both operands are zero. Since during sign resolution two sign positions are merged into one, at least one high-order digit of the product field is zero.

The multiply operation is executed in much the same manner as in manual arithmetic.* The multiplicand is examined one digit at a time, starting with the low-order digit, and the entire multiplier is added the number of times specified by the multiplicand digit. After the first multiplicand digit has been processed, the low-order digit of the resulting partial product (PP) is saved as the low-order product digit. The PP is then shifted one digit position to the right and brought into computation of the next product digit (one order higher than before). This

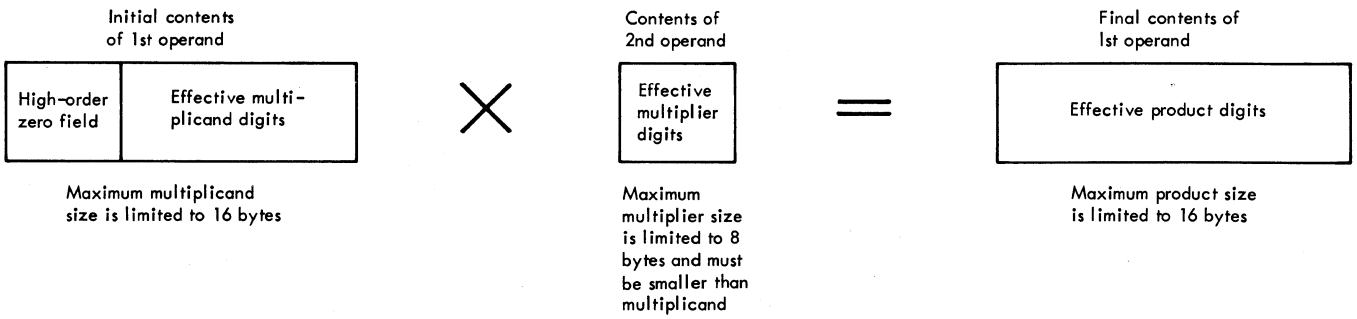


FIGURE 3-9. OPERAND SPECIFICATIONS FOR MP INSTRUCTION

* The major difference is that the roles of the multiplicand and the multiplier are reversed. Because of its size (up to 16 bytes), the entire multiplicand cannot be held in the machine at one time. For this reason, the full multiplier (up to 8 bytes) is fetched to the CPU and multiplied by the individual digits of the multiplicand, which is fetched from main storage 1 byte at a time.

time, the multiplier is added to the PP the number of times specified by the next digit of the multiplicand, and the low-order digit of the new PP thus formed becomes the next product digit. The PP is again shifted to the right, and the sequence is continued until all digits of the multiplicand have been processed. The PP resulting after the last multiplicand digit has been processed becomes the high-order product.

Figure 3-10 illustrates a typical over-and-over addition sequence used for mutiplication. As each multiplicand byte is processed, the multiplicand length code (L1) is reduced by one count and compared with the multiplier length code (L2). The condition when L1 = L2 indicates that all effective multiplicand digits have been processed and the operation is completed.

To reduce the number of computations in the multiply operation, either an over-and-over add or an over-and-over subtract sequence may be performed. Selection of the sequence is dependent on the magnitude of the multiplicand digit under consideration. An add sequence is selected if the magnitude of the digit is in the range of 1 through 4 (as was illustrated in Figure 6053, FEDM). For multiplicand digits of magnitude 5 or greater, a subtract sequence is selected. This sequence deducts the multiplier from the PP the number of times specified by the 10's complement of the multiplicand digit and then adds 1 to the next digit of the multiplicand; increasing the next high-order digit of the multiplicand has the effect of adding the multiplier 10 times. For example, the equivalent of a multiplication by 7 is subtracting the operand 3 times to obtain a negative PP and then effectively adding the operand to the PP 10 times.

An example of a typical subtract sequence used to accomplish multiplication is shown in Figure 3-11. Note that the PP resulting from a subtract operation is in 10's complement form. When the 10's complement PP is shifted right, its high-order digit position must be extended with a 9.

Following are general and detailed descriptions of the multiply microprogram. The general description (paragraph 3.8.3.2) outlines the overall structure of the microprogram, enumerates its major functional steps and sequences, and explains their relationship to the overall operation. The detailed description (paragraph 3.8.3.3.) analyzes each sequence individually, making specific references to the register-to-register data transfer in the machine.

3.8.3.2 General Description

- Initial Conditions:
  1. Low-order multiplicand bytes in ST.
  2. Low-order multiplier bytes requested from main storage.

- Interruptions:
  Protection.
  Addressing.
  Specification.
  Invalid data.

Upon entering the multiply microprogram, the following actions have been performed by SS I-Fetch:

1. The instruction address has been transferred to the LSWR, and the IC contains the address of the second operand.

2. A request from D has been issued for the multiplicand (first operand). This operand has been accessed (starting at the low-order address) and placed in ST. STC is set at the lowest-order multiplicand byte in ST.

3. A request from the IC has been issued for the multiplier (second operand), starting at the low-order address. This operand arrives at the SDBO during the GIS.

An overall flow chart of the multiply microprogram and the general data path used for its execution are shown in Figure 6054, FEDM. The major subroutines and functional steps, shown in the figure, are explained briefly below. Additional simplified diagrams are provided as an aid in visualizing the data handling performed. For the most part, these diagrams do not show the gates and data paths used in the machine, but are intended solely to convey how the multiply algorithm is implemented. For purposes of illustration, a 7-byte multiplicand and a 4-byte multiplier are assumed in these diagrams.

1. General Initialization Sequence (GIS)

   This sequence gates the multiplier from the SDBO to AB and sets the ABC at the lowest-order multiplier byte. It also performs several actions relating to the subsequent left-adjust sequence of the multiplier: (a) the lowest-order digit of the multiplicand (in ST) is transferred to F(0-3), (b) STAT F is set if the sign of the multiplicand is negative, and (c) the multiplier length code L2 is transferred

Objectives:

Multiply (+204) by (-32) to obtain a product of (-6,528)

Execution:

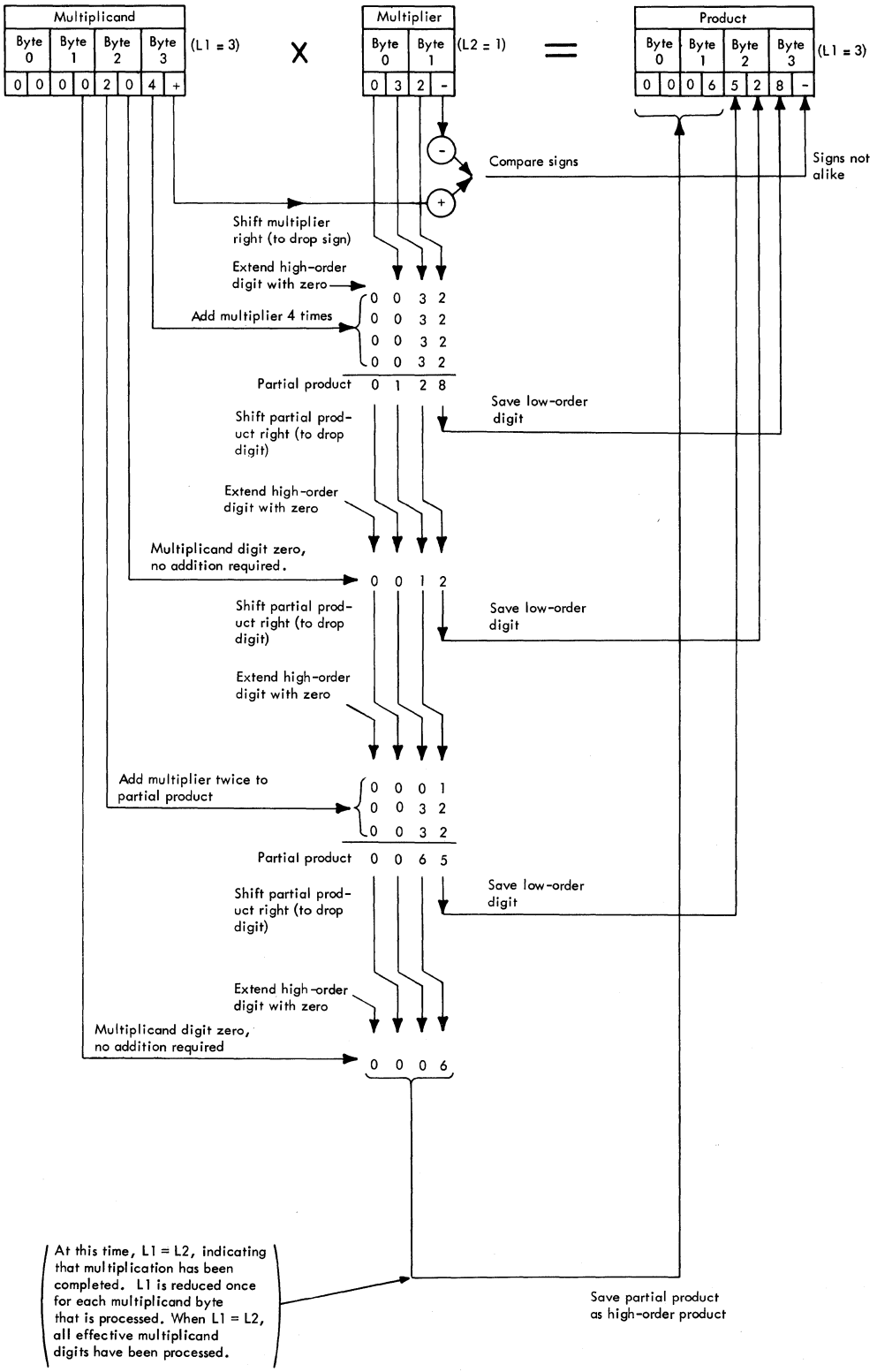


FIGURE 3-10. TYPICAL MULTIPLY ADD SEQUENCE, EXAMPLE 1

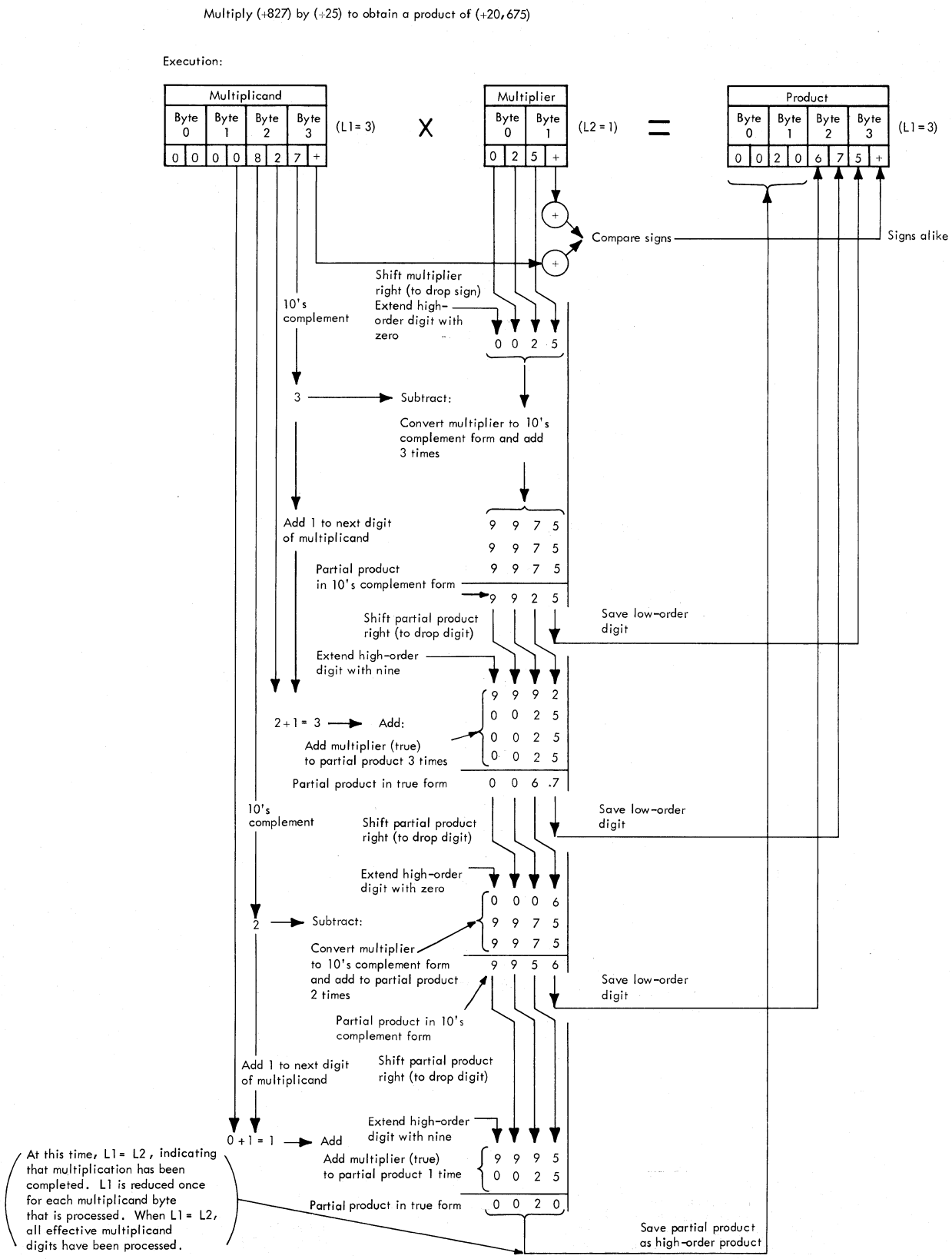


FIGURE 3-11. TYPICAL MULTIPLY SUBTRACT SEQUENCE, EXAMPLE 2

to F(4-7). The functions performed by the GIS are illustrated in Figure 3-12.

2. Specification Test

   This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies less than 8 bytes and is smaller than L1.

3. Incorrect Specification

   Detection of an invalid specification forces an interruption. The instruction address is restored from the IC to the LSWR, and the instruction is ended.

4. Multiplier Left-Adjust Sequence

   The multiplier bytes are transferred from AB to ST in such a manner that the highest-order multiplier byte occupies the leftmost byte in ST. STAT C is set if the multiplier sign is negative.

   The left-adjust transfer is initiated by setting STC per L2 (Figure 3-13). Since the maximum multiplier length is limited to 8 bytes, only 3 of the 4 bit positions in L2 are needed to effectively specify the length code; i.e., the count in L2 may range from a minimum of 0000 (for 1 byte) to a maximum of 0111 (for 8 bytes). Setting STC per L2 automatically selects, according to multiplier size, the correct ST position for the low-order byte of the multiplier; the number of bytes to the left of the selected ST position corresponds to the length field of the full multiplier.

   The actual transfer is performed one byte at a time, through the serial adder, starting with the low-order multiplier byte. ABC, STC, and L2 are decremented once for each byte transferred. The multiplier is completely transferred when the L2 count is decremented to zero. Since the first IC request (during I-Fetch) does not necessarily access the full multiplier to AB, it may be necessary to fetch the balance of the multiplier from main storage. (This fetch occurs if ABC steps to zero before L2 steps to zero.)

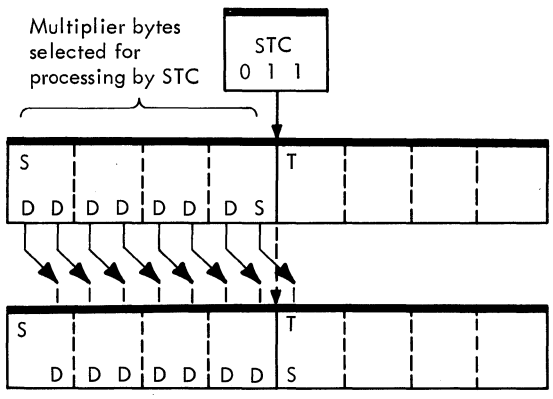
   After exit from the left-adjust sequence, the full multiplier has been fetched and left-adjusted to ST. Note that the original ST contents (the multiplicand) have been destroyed except for the lowest-order multiplicand byte, which is saved during the GIS; i.e., digit placed in F(0-3) and sign recorded by STAT F. The destroyed multiplicand bytes are later refetched from main storage, one byte at a time, as required by the multiply operation.

5. L2 Restoration

   During transfer of the multiplier bytes from AB to ST, the L2 count in E(12-15) is decremented once for each byte transferred. At the completion of the left-adjust sequence, L2 has been decremented to zero. The initial L2 count, saved in F(4-7) during the GIS, is now restored to E(12-15). The L2 count will be required by the subsequent multiply sequence.

6. Multiplier Right-4 Shift to Drop Sign

   In the multiply operation to follow, product bytes are developed by adding the entire multiplier the number of times specified by successive digits of the multiplicand. The sign of the multiplier does not enter into the over-and-over addition sequence and must be discarded. The sign is discarded by shifting the multiplier in ST 4 bit positions (1 digit) to the right as illustrated below. This action places the sign beyond the rightmost multiplier byte selected by STC for subsequent computation.



7. Sign Handling

   A test of STAT F and STAT C is made to establish the product sign algebraically:

   Signs alike (both STATS set or reset) — set sign plus.

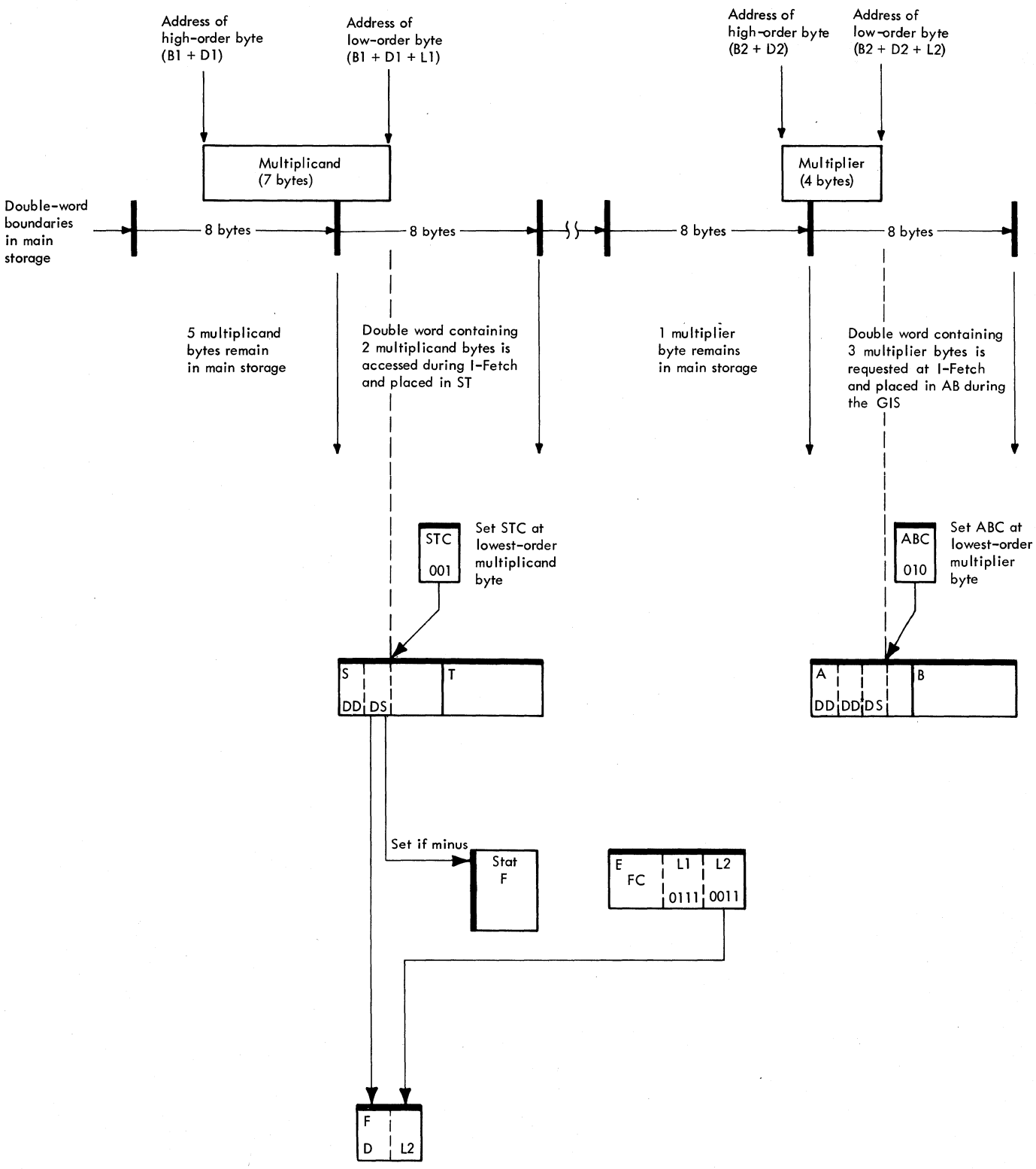


FIGURE 3-12. DATA HANDLING DURING GIS OF MULTIPLY INSTRUCTION

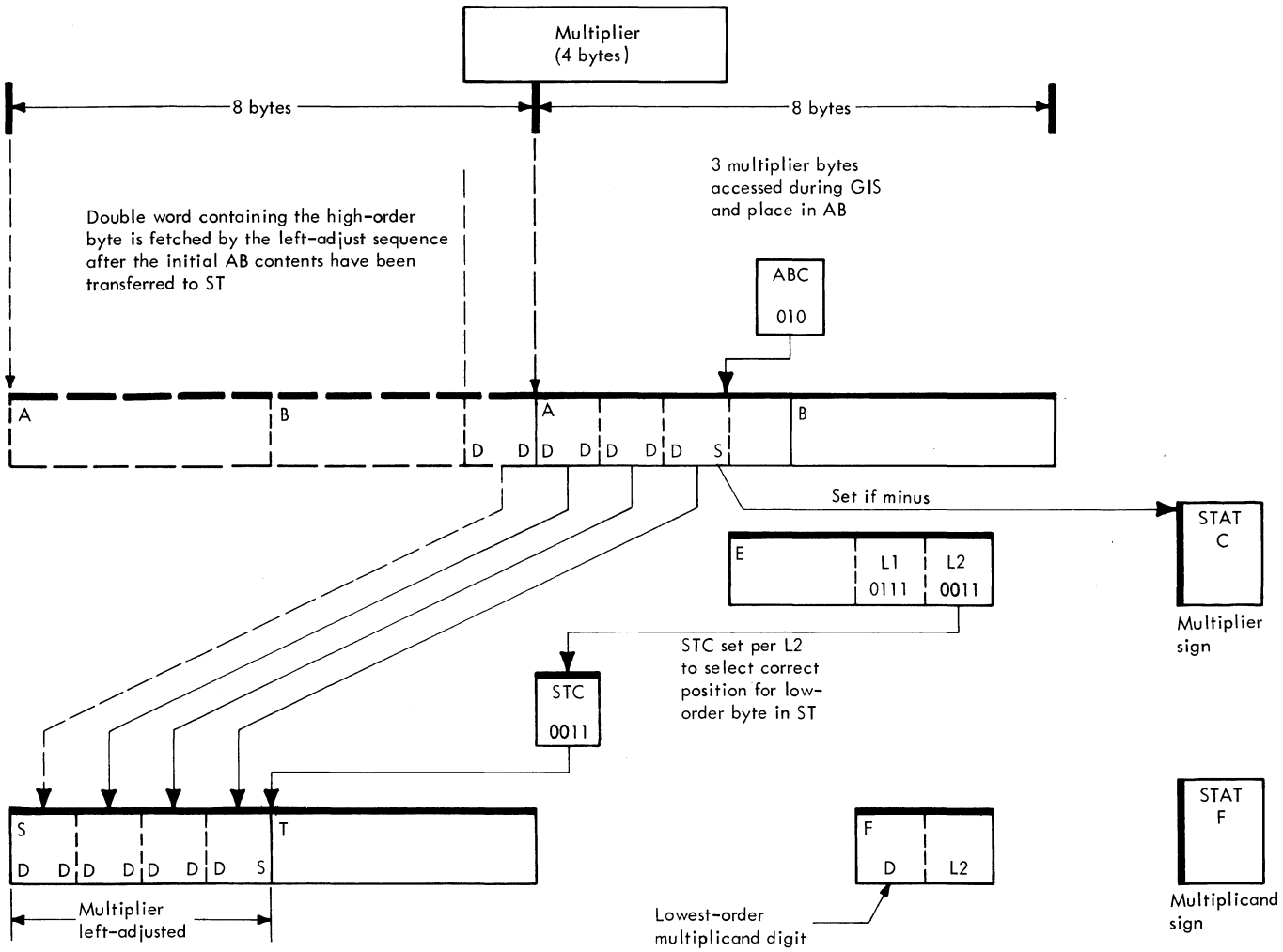


FIGURE 3-13. DATA HANDLING DURING MULTIPLIER LEFT-ADJUST SEQUENCE

Signs not alike (one STAT set and the other reset) — set sign minus.

Upon establishing the correct product sign, it is placed in F(4-7).

8. Basic Multiply Add or Subtract Sequence for Left Digit

This sequence processes the digit in the left portion of the multiplicand byte. (The lowest-order byte of the multiplicand always contains the digit in the left portion and the sign in the right portion.) The entire multiplier (in ST) is added or subtracted the number of times specified by the left digit of the multiplicand saved in F(0-3). An add sequence is performed if the digit in F(0-3) is 4 or less; a subtract sequence, if 5 or greater. A data interruption occurs if an invalid multiplicand or multiplier digit is detected. The PP resulting from the add or subtract sequence replaces the multiplicand in ST.

9. Product Byte Store

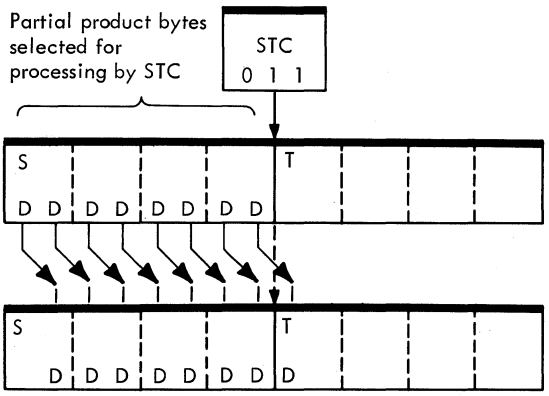
The product is stored in main storage one byte at a time. After exit from the left-digit sequence, one complete byte of product has been developed and must be stored. If the exit is made for the first time, this byte consists of the product sign (in F) and the lowest-order digit of PP (in ST). All product bytes generated thereafter consist of two digits: one digit (in F) has been saved from a previous PP developed in the right digit sequence, and the second digit is the low-order digit of a new PP (in ST) obtained in the left digit sequence.

10. Multiplicand Request

    A request from D is issued for the next byte of the multiplicand in main storage.

11. Partial Product Right-4 Shift to Drop Digit

    The low-order digit of PP has been stored as a product digit and must not enter into subsequent computation. The digit is discarded by shifting the PP is ST 4 bit positions to the right. This action places the digit beyond the rightmost PP byte selected by STC for computation of the next product digit.



12. L1 = L2

    This test establishes whether all digits of the multiplicand have been processed. At the start of the multiply operation, the total field length specified by L1 includes a zero field equal in size to the multiplier plus the effective field of the multiplicand:

    L1 = L2 + Number of effective multiplicand bytes.

    Since L1 is decremented once after each effective multiplicand byte is processed, all the effective multiplicand bytes have been processed when L1 equals L2.

13. Complete Multiplicand Byte Fetch

    If L1 is not equal to L2, the multiply sequence is continued. The next byte of the multiplicand (requested earlier) is selected from the SDBO and placed in F. Control is then transferred to the add or subtract sequence for the right digit of the multiplicand.

14. Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence.

    This sequence processes the digit in the right portion of the multiplicand byte. The entire multiplier is added to (or subtracted from) the PP in ST. The number of add or subtract operations is controlled by the right digit of the multiplicand contained in F(4-7). After a new PP has been developed in ST, its low-order digit replaces the right digit of the multiplicand in F(4-7). The PP is then shifted 4 bit positions to the right to drop the low-order digit, and an entry is made to the left-digit sequence to process the next multiplicand digit contained in F(0-3).

15. Multiplicand Zero Test and Partial Product Store

    When L1 equals L2, all the effective digits of the multiplicand have been processed. The remaining multiplicand bytes are fetched from main storage and tested for zero. Detection of a nonzero digit results in an interruption. After the zero test is completed, the PP is stored as the high-order product in main storage and the instruction is ended.

3.8.3.3 Detailed Description

- STAT E set if digit invalid.
- STAT A set if digit not zero.
- STAT G set if multiplier zero.
- STAT H set to generate hot carry.
- STAT D set to add 1 to next digit.

A detailed flow chart of the multiply microprogram is shown in Figure 6055, FEDM. This figure is an expanded version of the overall flow chart (Figure 6054, FEDM), showing the data handling used in the various subroutines of the Multiply instruction. For the most part, this data handling is straightforward and requires no explanation. Those areas in need of clarification are discussed below:

1. General Initialization Sequence

   This sequence shares a common microprogram with the Divide instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

2. Multiplier Left-Adjust Sequence

ABC has been set to select the low-order multiplier byte in AB. STC is now set per L2 count, E(12-15). The transfer is performed one byte at a time via the serial adder. As each byte is gated to the serial adder, it is tested for nonzero value and for invalid digits. STAT E is set upon detection of an invalid digit, and STAT A is set upon detection of a nonzero digit. If upon completion of the left-adjust transfer STAT A remains not set, the multiplier value is zero.

3. Multiplier Right-4 Shift and L2 Restoration

The multiplier is shifted 4 bit positions to the right and transferred from ST to AB. L2 is transferred from F(4-7) to E(12-15). Both actions are performed in parallel. As the high-order multiplier bytes are gated from S to PAA and the right-4 shift is initiated, the L2 count is transferred from F(4-7), via the serial adder, to S(28-31). After the right-4 shift has been performed through the parallel adder, the L2 count is gated from S to PAA and the zero count in E(12-15) is gated to PAB. The net result (original L2 count) is gated from PAL to E(12-15).

4. Sign Handling

STAT G is set if STAT A has not been set during preceding sequence. This step is taken to indicate a zero multiplier condition which requires special action.

5. Basic Multiply Add or Subtract Sequence

To perform a branch on the value of the multiplicand digit, the digit must be in SAL(4-7). This requirement is dictated by the W=(1-15) micro-order which samples SAL(4-7). For this reason, the contents of F are cross-gated through the serial adder and placed back in F. SAL(4-7) is then examined for the following values:

a. SAL(4-7) = 0

No addition cycles are required.

b. SAL(4-7) = 1 through 4

The multiplier in AB is added to the PP in ST the number of times specified by the digit value.

c. SAL(4-7) = 5 through 9

The multiplier in AB is subtracted from PP in ST the number of times specified by the 10's complement of the digit value (10 minus the digit value). STAT H is set to supply a hot carry for the subtract sequence. STAT D is set to add a 1 to the next digit of the multiplicand (equivalent to adding the multiplicand 10 times).

d. SAL(4-7) = invalid digit

The definition of an invalid digit is dependent on whether the digit to be processed is the first digit of the multiplicand; i.e., the digit immediately following the sign. If it is the first digit, then any value in the range of 10 through 15 is considered invalid and sets the interrupt trigger. After the first digit has been processed, a value of 10 is permissible in SAL(4-7), provided that it was formed by an original value of 9 to which a 1 has been added because STAT D was set. Under these conditions, the value of 10 does not set the interrupt trigger, no addition cycles are required, and a carry is propagated to the next digit by setting STAT D.

The multiplier to PP addition or subtraction is done one byte at a time in the serial adder, with the AB byte gated true +6 if adding and complement if subtracting. ABC and STC are both initially set to the L2 value and decremented by 1 each time a byte is processed. When the ABC count is stepped to 000, F(4-7) is examined to determine whether further additions or subtractions are necessary. If so, STC and ABC are again set to the L2 value, F(4-7) is incremented if subtracting or decremented if adding, and the multiplier is again added to or subtracted from the PP. The micro-order which steps the digit in F(4-7) is executed after the digit has been examined to determine whether further add or subtract cycles are required. For this reason, when a branch on F(4-7) is being made, a value of 1 when adding or of 9 when subtracting indicates that the multiplicand digit is completely processed. The low-order digit of the PP in ST is the product digit developed.

6. Product Byte Store — PP Right-4 Shift to Drop Digit — Multiplicand Request

   These three functions are accomplished in parallel fashion as illustrated in Figure 6055D, FEDM. After initiating the store operation, control is transferred to the shift-right-4 sequence. When the ST contents have been temporarily transferred, the store operation is resumed; the product byte is cross-gated, transferred to ST, and stored in main storage per D address. Thereafter, the microprogram requests the next multiplicand byte from main storage and simultaneoulsy completes the right-4 shift.

   As illustrated in Figure 3-14, the PP is shifted right-4 via the parallel adder. This shifting is done in several steps with the LSWR being used as temporary storage for the operand. Upon completion of the right-4 shift, B(64-67) is normally inserted as the high-order S digit. B(64-67) was previously set to 0 if the value in F(4-7) was less than 5, or to 9 if F(4-7) was 5 or greater, for then the PP was in 10's complement form. An exception is made when STAT G is set, indicating an all-zero multiplier. In this case, B(64-67) is not inserted in the high-order PP since it should always be zero.

7. Complete Multiplicand Byte Fetch

   If there are more multiplicand digits to be processed (L1 $\neq$ L2), the contents of T are temporarily transferred to the LSWR and either the left or the right half of the operand is gated from the SDBO to T. The next byte of the multiplicand is then selected per STC and transferred to F. (Note that if the left half word has been gated to T, the high-order STC bit is forced to 1, since, otherwise, STC would select a byte from S.)

8. Basic Multiply Add or Subtract Sequence for Right Digit, and Shift Right-4 Sequence.

   The next digit of the multiplicand in F(4-7) is examined, STAT's D and H are set or reset as required, a 0 or 9 is placed in B(64-67), and the appropriate add or subtract loop is entered. After exit from the add or subtract loop, the low-order digit of the resulting PP is saved in F(4-7). A right-4 shift is then performed on the PP in ST so that the low-order digit is dropped. At the completion of the right-4 shift, the left-digit sequence is resumed. As explained previously, the contents of F are cross-gated and the next digit of the multiplicand is sampled from SAL(4-7).

9. Multiplicand Zero Test and Partial Product Store

   An entry to this routine is made from the left-digit sequence. By operand definition, the remaining high-order multiplicand bytes should all be zeros. The PP in ST is the actual high-order product and must be stored in the high-order portion of the initial multiplicand field. However, if STAT D is set at this time, the multiplier must be added to the PP once more. After this has been done, the contents of ST are transferred to AB and the high-order multiplicand bytes are fetched to ST (per D address). STC is set per D(21-23) to designate the first high-order multiplicand byte to be tested for zero; ABC is set per L2 count to designate the first high-order PP byte in AB.

   The selected multiplicand byte in ST is tested for zero, then the selected PP byte in AB is transferred via the serial adder to ST, and the corresponding mark trigger is set. ABC, STC, and L1 are decremented once for each byte transferred. If a nonzero byte is detected in the high-order field of the multiplicand, the interrupt trigger is set and the instruction is ended.

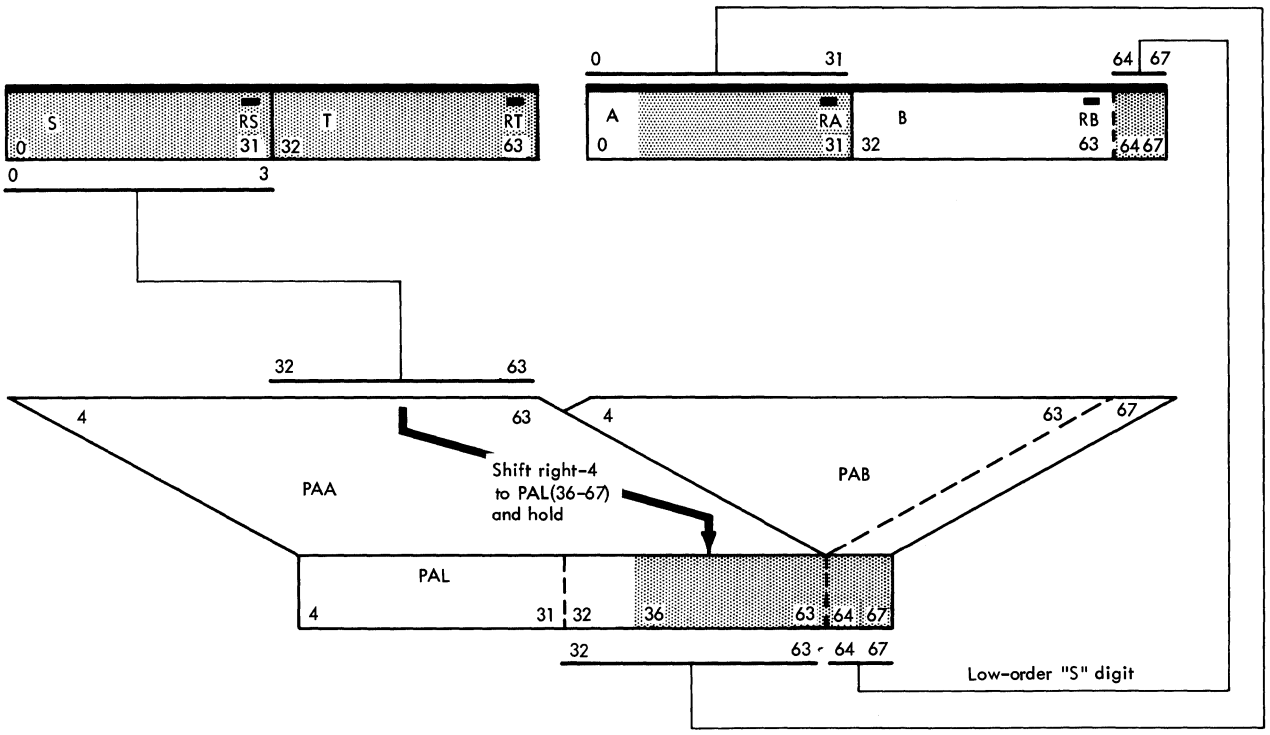
   The AB-to-ST byte transfer is continued until L1 or STC is stepped to zero, at which time the ST contents are stored in main storage. If STC has been stepped to zero (L1 $\neq$ 1111), the next high-order bytes of the multiplicand are fetched to ST and the sequence is resumed. If L1 has been stepped to zero (L1 = 1111), the instruction is ended.

3.8.4 DIVIDE, DP (FD)

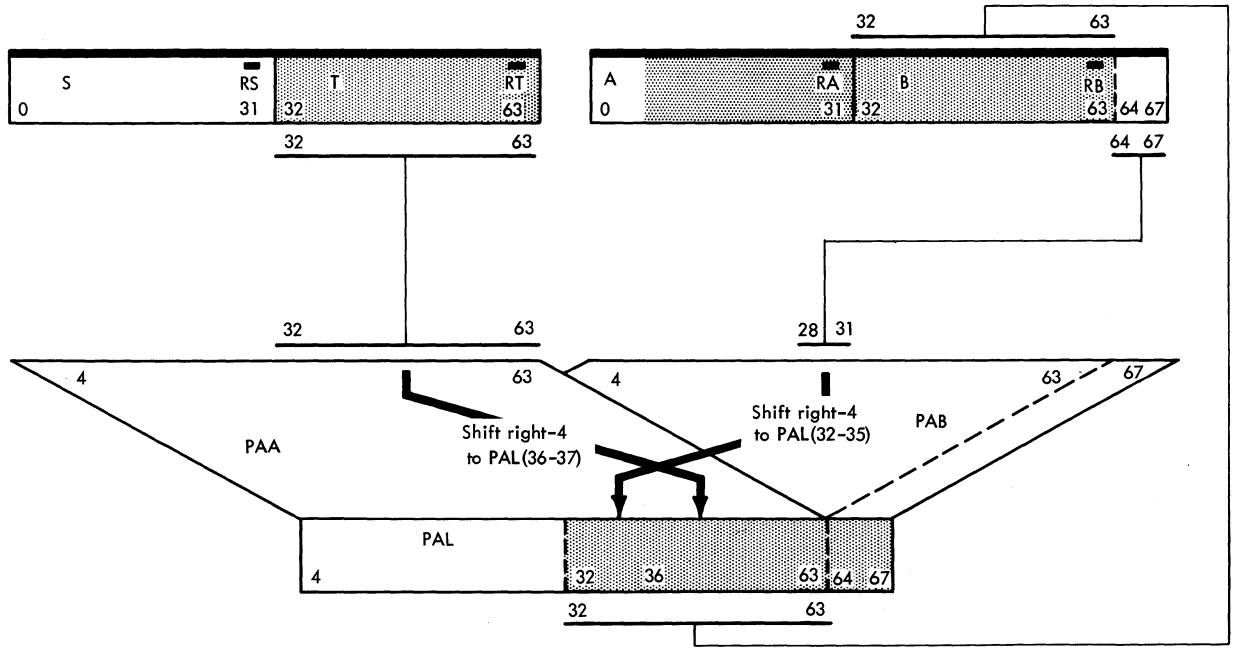
- Dividend (1st operand) is divided by divisor (2nd operand), and quotient and remainder replace dividend.
- SS format (see format on page 3-152).

3.8.4.1 Introduction

- Maximum dividend field is 16 bytes.
- Maximum divisor field is 8 bytes.

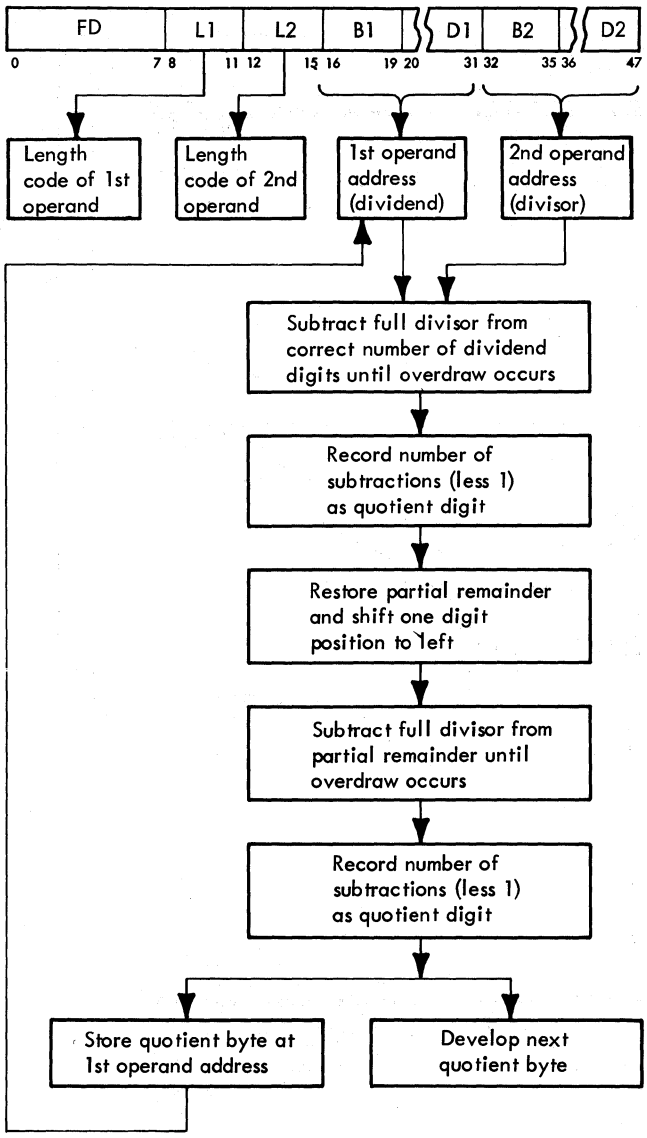


A. Step 1



B. Step 2

FIGURE 3-14. DATA FLOW FOR RIGHT-4 SHIFT OF ST TO AB, MULTIPLY INSTRUCTION



- L2 specifies byte length for divisor and remainder.
- L2 > 8 or L2 ≧ L1 will cause specification interruption.
- Division accomplished by over-and-over subtraction.
- Dividend field must initially contain sufficient number of high-order zeros to make possible storing of quotient and remainder.

The DP instruction replaces the dividend (1st operand) with the quotient and the remainder. To be able to store the quotient and the remainder in the dividend field at all times, several restrictions are imposed on the initial size of the dividend and the divisor (Figure 3-15).

The maximum dividend field is 16 bytes long. It is eventually replaced by the quotient, which is stored leftmost in the field, and by the remainder, which is stored rightmost. The size of the remainder is equal to the initial divisor size and is therefore predefined by length code L2. Since the minimum remainder size is 1 byte (L2 = 0), the maximum quotient size is limited to 15 bytes. By definition, the size of the divisor (and remainder) cannot exceed 8 bytes. A divisor greater than 8 bytes, or in excess of the dividend, is recognized as a specification error; the instruction is suppressed and a program interruption occurs.

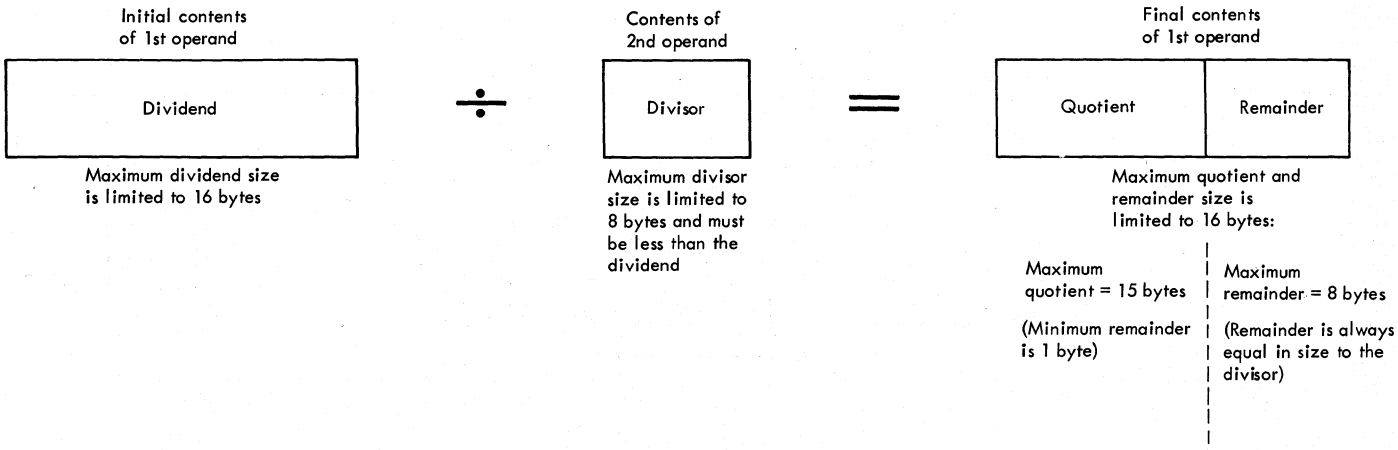


FIGURE 3-15. OPERAND SPECIFICATIONS FOR DP INSTRUCTION

To make sure that the quotient and remainder will fit in the destination field, the magnitudes of the dividend and the divisor are compared prior to entering the divide sequence. This comparison, called "divide check" or "trial subtraction," yields the number of quotient digits that will result if division is carried out. If the predicted quotient is larger than that allowed, the instruction is suppressed and a program interruption occurs. For this reason, and overflow condition cannot exist upon execution of a Divide instruction.

The dividend, divisor, quotient, and remainder are all signed integers, right-aligned in their fields. The sign of the quotient is determined algebraically from the dividend and divisor signs. The sign of the remainder is the same as that of the dividend. These rules hold true even when the quotient or the remainder is zero.

The divide operation is executed in much the same manner as in manual arithmetic. First, the divisor is properly aligned with the high-order dividend; then, by repeatedly subtracting the divisor from the dividend and counting the number of successful subtractions, the high-order quotient digit is determined. The partial remainder resulting from the last successful subtraction is shifted one digit position to the left, and the next lower-order dividend digit is inserted at the low-order end of the partial remainder. To obtain the next quotient digit, the divisor is again subtracted from the partial remainder. This sequence is repeated until all dividend digits have been processed. The remainder resulting from the final successful subtraction is given the sign of the dividend and stored in the low-order end of the dividend field.

Figure 3-16 illustrates a typical over-and-over subtract sequence used to accomplish division. Initially, a sufficient number of high-order dividend digits must be selected to perform the first successful subtraction. Successful subtractions of the divisor from the dividend occur until the partial remainder is overdrawn. The divisor is then added back once to restore the correct partial remainder. At the same time, the quotient digit is decremented once to compensate for the overdraw. As each dividend byte is processed, the length code of the dividend (L1) is reduced by 1 and compared with the length code of the divisor (L2). Since the size of the remainder is also defined by length code L2, the condition of L1 equal to L2 indicates that all the effective bytes of the dividend have been processed, and the remainder is to be stored in the rest of the destination field. Note that, to be able to fit the quotient and the remainder in the destination field, this field must initially contain high-order zeros. A program interruption will occur if the dividend does not have at least one leading zero.

Following are general and detailed descriptions of the divide microprogram. The general description (paragraph 3.8.4.2) outlines the overall structure of the microprogram, enumerates its major functional steps and subroutines, and explains their relationship to the overall operation. The detailed description (paragraph 3.8.4.3) analyzes each sequence individually, making specific references to the register-to-register data transfer in the machine.

3.8.4.2 General Description

- Initial conditions:
  1. High-order dividend bytes in ST.
  2. Low-order divisor bytes requested from main storage.

- CC remains unchanged.

- Interruptions:
  Protection.
  Addressing.
  Specification.
  Divide check.
  Invalid data.

Upon entering the divide microprogram, the following actions have been performed by SS I-Fetch:

1. The instruction address has been transferred to LSWR, and the IC contains the address of the divisor (second operand).

2. A request from D has been issued for the dividend (first operand). This operand has been accessed (starting at the high-order address) and placed in ST. STC is set at the lowest-order dividend byte in ST.

3. A request from the IC has been issued for the divisor (second operand), starting at the low-order address. This operand arrives at the SDBO during GIS.

An overall flow chart of the divide microprogram and the general data path used for its execution are shown in Figure 6056, FEDM. The major subroutines and functional steps, shown in the figure, are explained briefly below. Additional simplified diagrams are provided as an aid in visualizing the

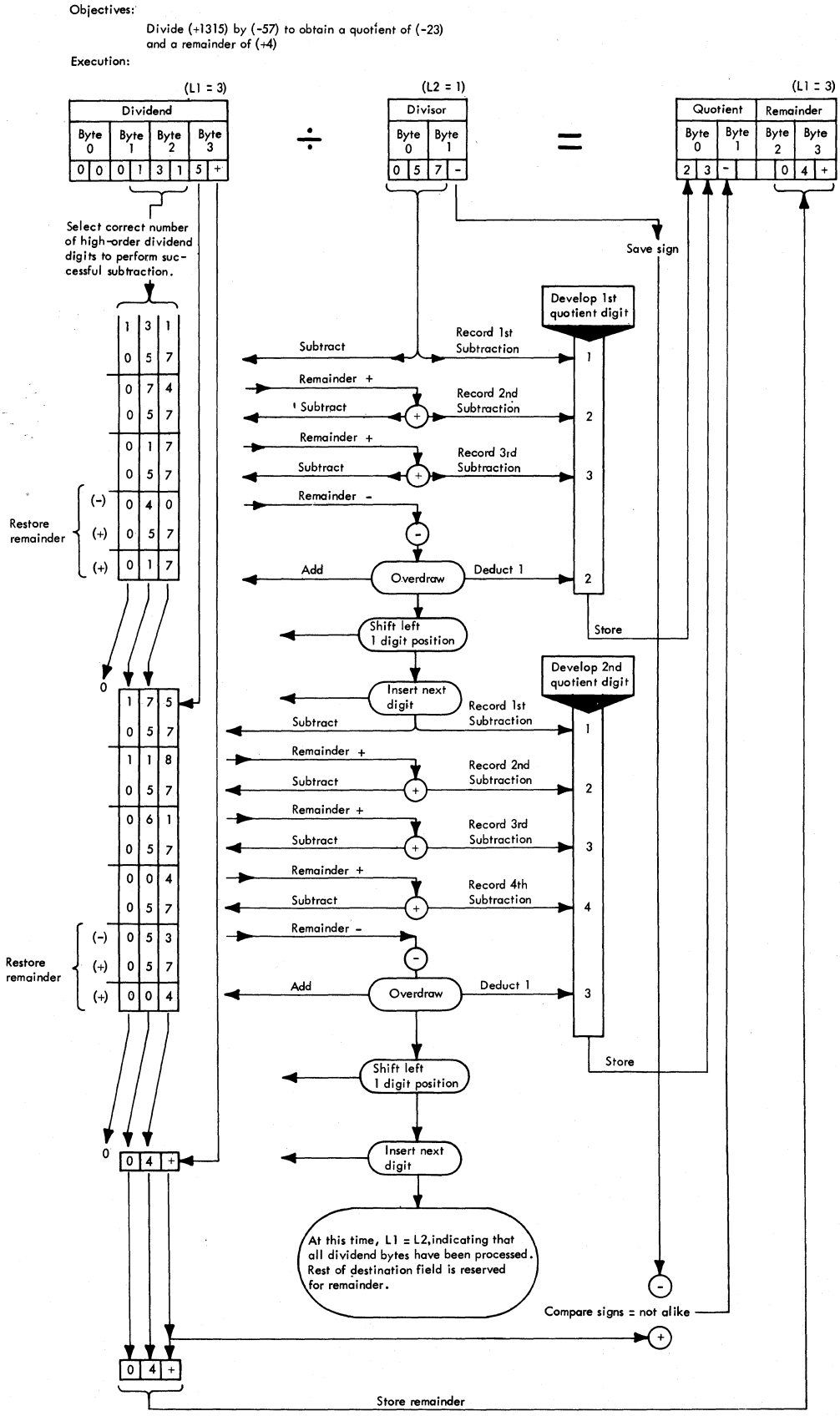


FIGURE 3-16. EXAMPLE OF A TYPICAL DIVIDE SEQUENCE

data handling performed. For the most part, these diagrams do not show the gates and data paths used in the machine, but are intended solely to convey how the divide algorithm is implemented. For purposes of illustration, a 9-byte dividend and a 3-byte divisor are assumed in these diagrams.

1. General Initialization Sequence (GIS)

   This sequence gates the divisor from SDBO to AB and sets ABC at the lowest-order divisor byte in AB (Figure 3-17). Two additional functions are performed to facilitate subsequent data handling: (a) STC is set per L2, and (b) length codes L1 and L2 are transferred to F.

2. Specification Test

   This test verifies that the length codes for both operands in the instruction are correctly specified; i.e., L2 specifies less than 8 bytes and is smaller than L1.

3. Incorrect Specification

   Detection of an invalid specification forces an interruption. The instruction address is restored to the LSWR, and the instruction is ended.

4. Divisor Left-Adjust Sequence

   The divisor bytes are transferred from AB to ST in such a manner that the highest-order divisor byte occupies the leftmost byte in ST. STAT C is set if the divisor sign is negative.

   The left-adjust transfer is initiated by setting STC per L2 (a function performed during the GIS). Since the maximum divisor length is limited to 8 bytes, only 3 of the 4 bit positions in L2 are needed to effectively specify the length code; i.e, the count in L2 may range from a minimum of 0000 (for 1 byte) to a maximum of 0111 (for 8 bytes). Setting STC per L2 automatically selects, according to the divisor size, the correct ST position for the low-order byte of the divisor; the number of bytes to the left of the selected ST position corresponds to the length field of the full divisor (Figure 3-18). The actual transfer

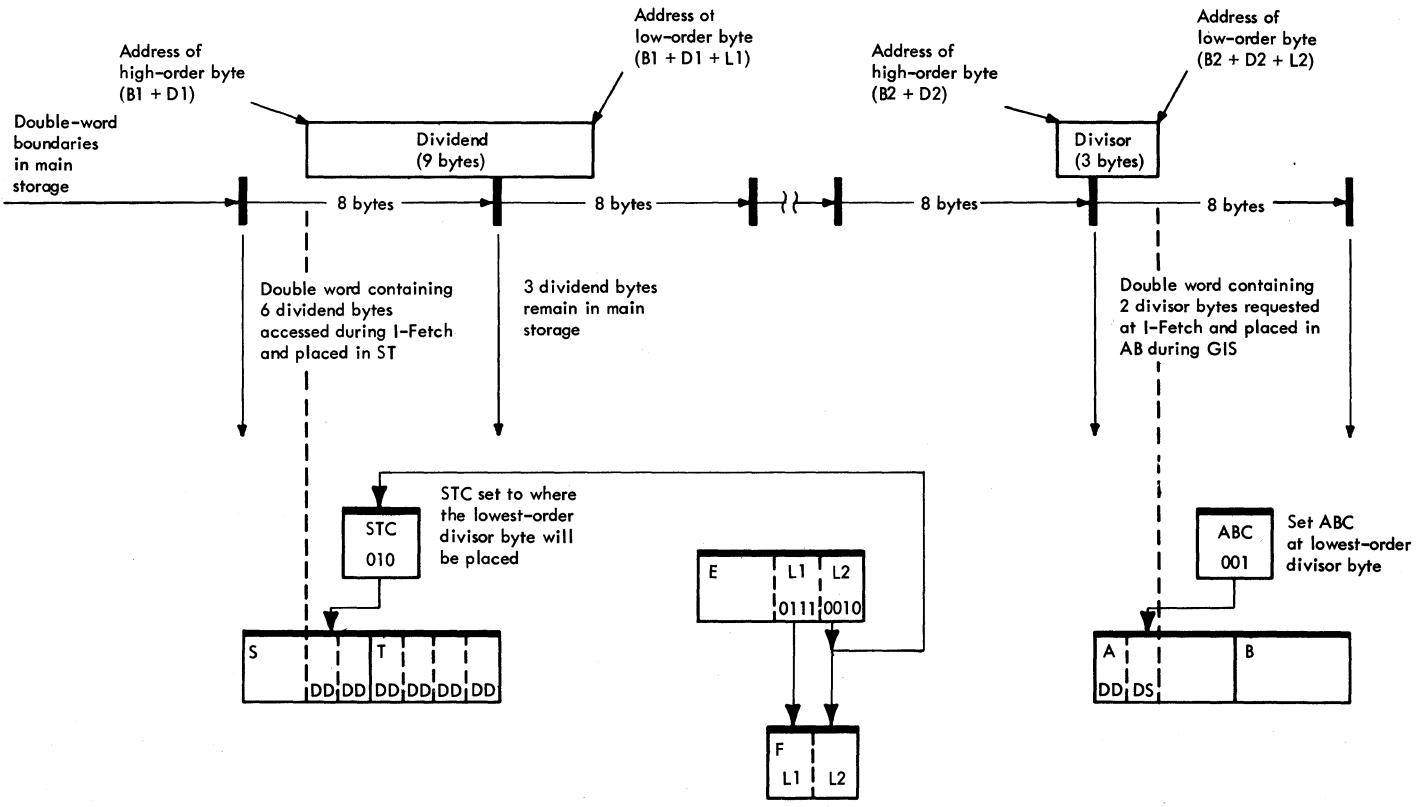


FIGURE 3-17. DATA HANDLING DURING GIS OF DIVIDE INSTRUCTION

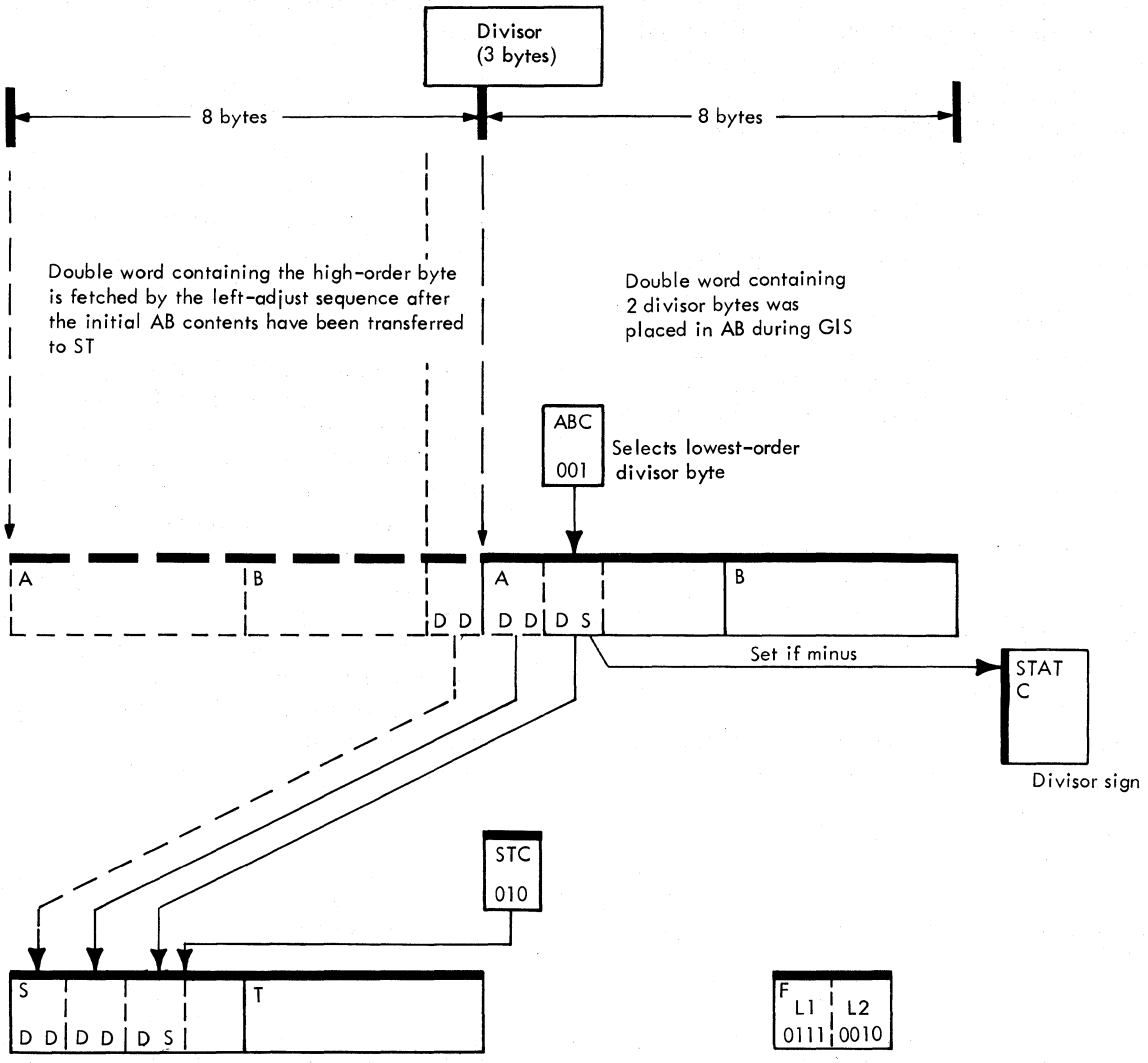


FIGURE 3-18. DATA HANDLING DURING DIVISOR LEFT-ADJUST SEQUENCE

is performed one byte at a time, through the serial adder, starting with the low-order divisor byte. ABC, STC, and L2 are decremented once for each byte transferred. The divisor is completely transferred when the L2 count is decremented to zero. Since the first IC request (during I-Fetch) does not necessarily access the full divisor to AB, it may be necessary to fetch the balance of the divisor from main storage. (This fetch occurs if ABC steps to zero before L2 steps to zero.)

After exit from the Divisor Left-Adjust sequence, the full divisor has been fetched and left-adjusted to ST. Note that the original ST contents (the dividend) have been destroyed. For this reason the dividend must be refetched from main storage.

5. Dividend Fetch and Left-Adjust Sequence

This sequence fetches a sufficient number of high-order dividend bytes to perform a trial subtraction of the divisor from the dividend. The full divisor is subtracted once from the high-order dividend. Since the maximum divisor size is 8 bytes, 8 high-order dividend bytes are required for trial subtraction. If the dividend is 8 bytes or less, it will be completely fetched during this sequence; if greater than 8 bytes, only the first 8 high-order bytes will be fetched. The dividend is fetched to AB and then transferred to ST in such a manner that the highest-order byte occupies the leftmost byte in ST.

Upon entry into this sequence, ST is assumed to be completely occupied by the divisor. (If

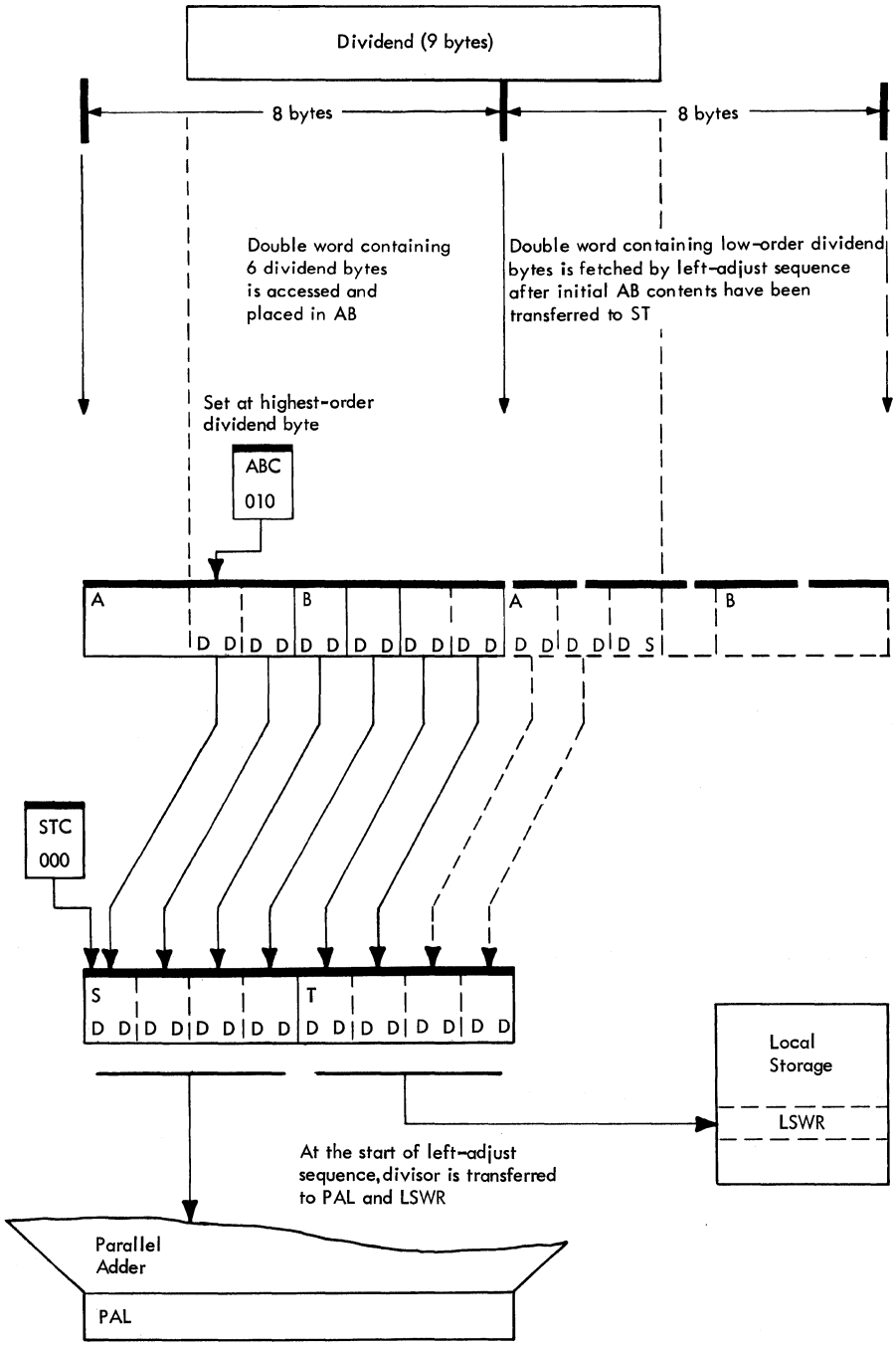


FIGURE 3-19. DATA HANDLING DURING DIVIDEND FETCH AND LEFT-ADJUST SEQUENCE

the divisor is 4 bytes or less, it is confined solely to S; if greater than 4 bytes, the divisor extends into T.) Since left-adjustment of the dividend requires the use of ST, the divisor must be transferred from ST: S is gated to the parallel adder and held in PAL, and T is stored in the LSWR (Figure 3-19).*

A request per the high-order dividend address is issued from D. Upon arrival of the dividend from main storage, the SDBO is gated to AB. ABC is set per D(21-23) to point at the highest-order dividend byte. The left-adjust transfer is initiated by setting STC to 000, thus selecting the leftmost byte in ST. The dividend bytes

* The ⟶HOLD micro-order is issued on each cycle of the left-adjust sequence to hold the S contents in PAL.

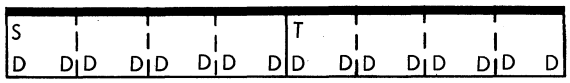
are then transferred to ST, starting with the high-order byte. (The actual transfer is performed one byte at a time through the serial adder.) ABC and STC are incremented, and L1 is decremented once for each byte transferred. If L1 steps to zero before ABC or STC steps to 7, the full dividend has been fetched and left-adjusted to ST. Since the first request does not necessarily access 8 bytes of dividend to AB, it may be necessary to fetch additional dividend bytes from main storage. This fetch occurs if ABC steps to 7 before STC steps to 7 or L1 steps to zero.

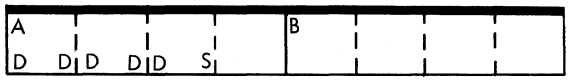
6. Restore L1 and L2 to E

   Left-adjustment of the divisor and dividend has decremented L2 and L1 to zero. The initial L2 and L1 counts, saved in F during GIS, are now restored to E(8-15). These counts will be required by the subsequent divide sequence.

7. Assemble Divisor in AB and Dividend in ST

   The divisor in PAL and LSWR is restored to AB. Upon completion of this function, both operands are left-aligned: the dividend is in ST, and the divisor is in AB.





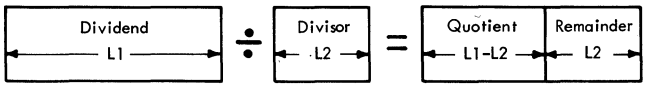
8. Trial Subtraction

   The divisor bytes in AB are subtracted from an equivalent number of high-order dividend bytes in ST. The remainder is then examined to establish whether the divide operation to follow till generate a result (quotient plus remainder) that will fit in the destination field. A negative remainder indicates that the destination field specified in the instruction is sufficiently large to accommodate the result. A positive remainder, however, indicates that the result cannot fit in the destination field, and an interruption occurs.

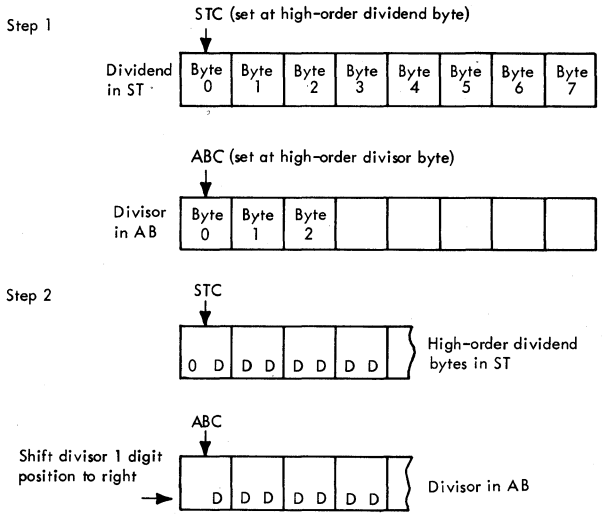
   How prediction by trial subtraction is possible may be understood from the following considerations:

   a. By definition, the dividend is at least one order higher than the divisor. The highest-order digit position in the dividend is always zero.

   b. The length code of the divisor (L2) is also the length code for the remainder. Consequently, the maximum number of quotient bytes that will fit in the destination field is equal to L1 - L2. By operand definition, the difference L1 - L2 may range from a minimum of 1 byte to a maximum of 8 bytes.



   c. To perform trial subtraction, the high-order divisor digit is aligned with the high-order digit of the dividend. This is performed in two steps: (1) the high-order divisor byte is aligned with the high-order byte of the dividend, and (2) since by definition the highest-order digit position in the dividend is always zero, the divisor is shifted right one digit position to align the significant digits in both operands.

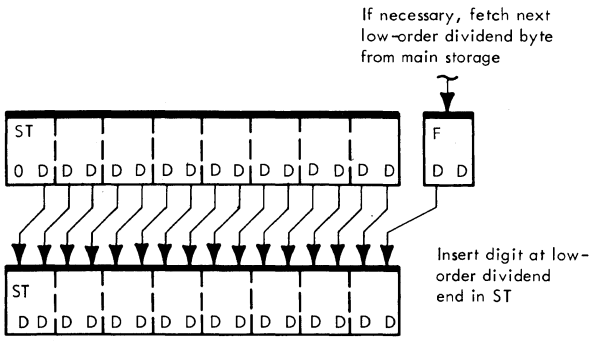


   d. Because the dividend is at least one order higher than the divisor, alignment of the high-order divisor digit with that of the dividend is equivalent to multiplying the divisor at least 10 times; if the dividend is one order higher, the divisor is multiplied 10 times; if two orders higher, 100 times; if three orders higher, 1000 times; and so on. Thus, during trial subtraction, a quantity at least 10 times that of the divisor is subtracted from the dividend.

e. Since the maximum number of quotient digits allowed (L1 - L2) corresponds to the difference between the orders of magnitude in the two operands, the result of the trial subtraction must always yield a negative remainder; otherwise, the number of quotient digits that would be generated would not fit in the destination field.

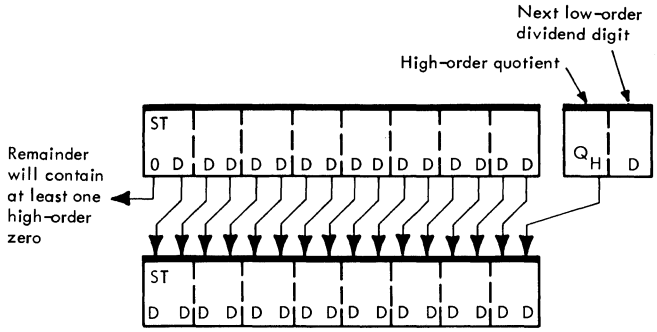
9. Shift Dividend One Digit to the Left

The dividend is shifted one digit to the left to allow successful subtraction of the divisor from the dividend. (To develop the quotient digit, the divisor must be repeatedly subtracted from the dividend until a negative remainder occurs.) Upon initiating the left-4 shift, a test is made to establish whether an additional low-order dividend digit is required for generation of the first quotient digit. If required, the next low-order dividend byte is fetched from main storage and placed in F. The digit is selected from F(0-3) and inserted at the low-order end of the dividend in ST.



10. Generate Quotient and Left-Digit Sequence

The divisor is repeatedly subtracted from the dividend until an overdraw occurs; i.e., a negative remainder is obtained. The number of successful subtractions is recorded and, after the last successful subtraction, becomes the high-order quotient digit. This digit is checked for validity and then inserted in F(0-3) by the left-digit sequence. This sequence also shifts the partial remainder (in ST) one digit to the left and inserts the next low-order dividend digit in the low-order end of ST.



11. Correct Low-Order Remainder Byte

In certain cases, the low-order remainder byte in ST must be corrected. The need for correction will become apparent when detailed analysis of the Divide instruction is undertaken (paragraph 3.8.4.3).

12. Generate Next Quotient Digit and Right Digit Sequence

After exit from the left digit sequence, the operand length codes (L1 and L2) are compared to establish whether the last byte of quotient is being processed. If L1 is equal to L2, the correct quotient sign is inserted in F(4-7). The last quotient byte (in F) is stored; then, the partial remainder (in ST) is stored in the low-order destination field as the final remainder.

If L1 is not equal to L2, the next quotient digit is generated and placed in F(4-7) by the right digit sequence. At the completion of this sequence, one complete byte of quotient is contained in F. This byte is stored in main storage, and the sequence for the left digit of the next quotient byte is started.

3.8.4.3 Detailed Description

- STAT A set to indicate nonzero divisor.
- STAT C set if divisor is negative.
- STAT D set if dividend less than 8 bytes.
- STAT E set if digit or sign invalid.
- STAT F set if dividend negative.

- STAT G is first set if divisor is 5 bytes or greater. STAT G is then set again to enter left-digit sequence.

- STAT H is set to generate hot carry for subtract sequence.

A detailed flow chart of the divide microprogram is shown in Figure 6057, FEDM. This figure is an expanded version of the overall flow chart (Figure 6056, FEDM), showing the data handling used in the various subroutines of the Divide instruction. The major subroutines are listed below, and those areas in need of clarification are explained:

1. General Initialization Sequence

   This sequence shares a common microprogram with the Multiply instruction. An appropriate branch is taken to enter either the divide or the multiply sequence.

   A test of L2 (contained in STC) is performed to establish the byte size of the divisor. STAT G is set if divisor is equal to or greater than 5 bytes. This function increases the execution speed when assembling the divisor in AB (see step 4); i.e., if the divisor is 4 bytes or smaller, the LSWR need not be restored to B.

2. Divisor Left Adjust Sequence

   a. The initial STC setting selects the rightmost ST byte that will contain the lowest-order divisor byte.

   b. STAT C is set if divisor sign is negative. Since the dividend is accessed starting with the high-order byte, the dividend sign is not available at this time.

   c. If ABC steps to zero before L2 steps to zero, the remaining low-order divisor bytes are fetched from main storage.

   d. The divisor digits are checked for validity, and STAT E is set if an invalid digit is detected. STAT A is set to indicate a nonzero divisor. Division by zero will result in a divide check interruption during trial subtraction.

   e. Upon fetching the full divisor, the divisor address is no longer needed, and the instruction address is restored to the IC.

3. Dividend Fetch and Left Adjust Sequence

   a. The divisor is shifted one digit position to the right so that its high-order digit will be aligned with that of the dividend. (The dividend is not yet available at this time.)

   b. The low-order divisor word is transferred from T to the LSWR. The high-order divisor word is gated to the parallel adder and held in PAL by the - → HOLD microorder.

   c. STC is set to zero to select the high-order ST byte (where the highest-order dividend byte will be placed).

   d. A test of the high-order L1 bit is performed to establish the byte size of the dividend. STAT D is set if the dividend is less than 8 bytes. This function increased the execution speed upon exit from the trial subtraction. As shown in Figure 6057F, FEDM, a branch per STAT D is made to determine whether the complete dividend has been fetched. (If STAT D is set, the full dividend has been fetched, since at least 8 dividend bytes are fetched to perform trial subtraction.)

   e. If ABC steps to zero before L1 or STC steps to zero, the D address is incremented by 8 and a fetch of the next double word of the dividend is made. The destination address for the subsequent quotient bytes is then restored by subtracting 8 from D.

4. Assemble Divisor in AB and Dividend in ST

   This function is performed in parallel with the L1 and L2 restoration sequence. The high-order divisor word is transferred from PAL to A, after which restoration of the L1 and L2 counts is started. During the restoration sequence, STAT G is tested to establish whether the full divisor has been assembled in AB. If STAT G is not set, the divisor is 4 bytes or less. Therefore, the full divisor was contained in PAL and has been placed in AB. In this case, the restoration sequence is completed and an exit is made to the trial subtraction routine.

   If STAT G is set, the low-order portion of the divisor is contained in the LSWR and must be transferred to B prior to entering trial subtraction. Transfer to the LSWR contents to

B must be performed via ST, which contains the left-aligned dividend. Several execution cycles are used to transfer the LSWR contents to B without destroying the ST contents.

5. Trial Subtraction

   The divisor is subtracted from the dividend one byte at a time. After the last subtract cycle, a branch is made on a carry condition from SAL(0). Presence of a carry indicates that the remainder is positive, and the instruction is ended. Absence of a carry indicates a negative remainder, and that the results of the divide operation will fit in the destination field.

6. Dividend (or Partial Remainder) Left-4 Shift

   The dividend is shifted left one digit position to perform the first successful subtraction of the divisor from the dividend. Upon initiating the left-4 shift, a test (per STAT D) is made to establish whether an additional dividend digit must be inserted in the low-order end of ST. If STAT D is set (see step 3, d), all dividend bytes have been fetched from main storage and the left-4 shift is completed. If STAT D is not set, the following actions take place:

   a. The D address is incremented by 8, and the next double word of the dividend is requested from main storage.

   b. The T contents are temporarily transferred to the LSWR. (Upon arrival of the dividend double word from main storage, T is loaded with the dividend word containing the next digit to be inserted.)

   c. STC is set per D(21-23) to select the correct dividend byte in the requested double word.

   d. The left-4 shift of the dividend is completed. The high-order dividend word is in S, and the low-order word is in the LSWR.

   e. A branch per D(21) is made to establish which word in the SDBO contains the next dividend byte. The correct word is then gated from SDBO to T. [Note that, if the left SDBO word is gated to T, STC(0) is forced to 1 to select the correct byte in T.]

   f. The selected dividend byte is transferred from T to F. The shifted low-order dividend word is then restored from the LSWR to T.

   g. The destination address is restored by subtracting 8 from D.

   h. The high-order L1 bit is tested to establish the byte size of the dividend, and STAT D is set if the dividend is less than 8 bytes. This function increases the execution speed upon exit from the right-digit sequence (Figure 6057H, FEDM).

7. Generate Quotient Sequence

   a. ABC and STC are set per L2 to select the low-order operand bytes. STAT H is set to provide a hot carry to the serial adder.

   b. The selected AB byte is subtracted from the selected ST byte via the serial adder. The result is gated back to the selected ST byte, with the carry being saved in STAT H. Any invalid digit detected in the serial adder will set STAT E.

   c. ABC and STC are decremented as each byte is processed. When ABC is stepped to zero, a 1 is added to F(4-7) and ABC and STC are again set per L2.

   d. If a serial carry results upon processing the high-order byte, the partial remainder in ST is positive and the divisor is again subtracted from the dividend. F(4-7) is incremented once each time a complete subtraction is made.

   e. If there is no carry upon processing the high-order byte, an exit is made to the appropriate left- or right-digit sequence, as determined per STAT G.

   f. Note that, before starting each subtract sequence, the partial remainder resulting from the previous subtraction is saved in the LSWR and PAL. This saving is done because, upon exit on a no-carry condition, an overdraw has occurred and the remainder in ST cannot be used for computation of the next quotient digit. Instead, the partial remainder resulting from the last successful subtraction is used for subsequent computation.

8. Left-Digit Sequence

   a. The quotient digit in F(4-7) is the left digit of a quotient byte. This digit is reduced one count, to compensate for the overdraw, and then cross-gated via the serial adder to F(0-3).

   b. The partial remainder resulting from the last successful subtraction and saved in the LSWR and PAL is shifted one digit to the left and restored to ST. The next low-order dividend digit, in B(64-67), is inserted in the low-order end of ST.

   c. A test on STC equal to or greater than 4 is made to establish whether the low-order byte of the partial remainder in the LSWR has been overdrawn. In the generate quotient sequence (Figure 6057F, FEDM), the contents of T are stored in the LSWR at the same time that the first subtract cycle is performed. Thus, if the partial remainder extends into T (which occurs if STC is 4 or greater), the low-order divisor byte is subtracted from the low-order partial remainder byte once to often. In such cases, the low-order byte of the partial remainder is corrected by adding it to the low-order divisor byte. After performing the correction, the left-digit sequence is re-entered.

   d. If L1 is equal to L2, the quotient sign byte is processed. Otherwise, the quotient digit generation routine is resumed to develop the next digit.

9. Correct Low-Order Remainder Byte

   This routine is entered from the left- or right-digit sequence if the low-order divisor byte has been subtracted once too often from the low-order byte of the partial remainder. Correction is performed as follows:

   a. STAT H is reset to initiate a true add cycle.

   b. The low-order partial remainder word is placed in T. STC is set per L2 to select the low-order byte in T.

   c. The low-order divisor byte (per ABC) is added once to the low-order partial remainder byte (per STC), and the result is gated to T per STC.

   d. The left- or right-digit sequence is re-entered, as applicable.

10. Right-Digit Sequence

    This sequence is entered when two quotient digits have been generated and placed in F. The following actions are performed:

    a. STC is set per D(21-23), and F is transferred to the selected ST byte. The corresponding mark trigger is set per STC.

    b. A storage request is issued to store the quotient byte per D address.

    c. A left-4 shift of the partial remainder (in PAL and LSWR) is initiated.

    d. If STAT D is set, indicating that dividend byte fetch is not required, the left-4 shift is completed and the partial remainder is restored to ST.

    e. If STAT D is not set, the dividend byte fetch sequence is entered as shown in Figure 6057F, FEDM.

    f. D is decremented by 1 to obtain the destination address for the next quotient byte.

    g. F is cleared and STAT G is set to enter the left-digit sequence, after the first quotient digit is generated.

    h. If STAT E is set, the invalid-data-interrupt trigger is set and the instruction is ended.

    i. If STAT E is not set, the generate-quotient sequence is entered.

11. Process Quotient Sign Byte

    This routine is entered from the left-digit sequence when L1 equals L2. At this time, all dividend digits have been processed: the low-order quotient digit is in F(0-3), the byte selected by STC is the dividend sign byte, and the remaining high-order contents of ST are the final remainder. The following actions take place:

    a. STC and ABC are set per the L2 count. STAT F is set if bits 4-7 of the selected ST byte indicate a negative sign. STAT E is set if the sign is invalid.

b. The correct negative or positive sign is put in F(4-7) as determined by a comparison of STAT's C and F.

c. The ST contents are transferred via the parallel adder to AB.

d. STC is set per D(21-23), and F is gated to the selected ST byte. The corresponding mark trigger is set, and the selected ST byte is stored in main storage.

12. Store Remainder Routine

a. The byte selected by ABC, which is the low-order remainder byte, is saved in F. If necessary, the remainder sign is corrected in the serial adder prior to gating to F.

b. The remainder is transferred from AB to ST one byte at a time. As each byte is transferred, the corresponding mark trigger is set, ABC and STC are incremented by 1, and L2 is decremented by 1.

c. When STC steps to 7, ST is stored per D address. D is then incremented by 8, and byte transfer is resumed.

d. When L2 steps to 0, STC is decremented by 1, and the remainder sign byte is gated from F to ST. The contents of ST are then stored in the low-order destination field, and the instruction is ended.

e. If STAT E is set, an exit is made to the interrupt microprogram.

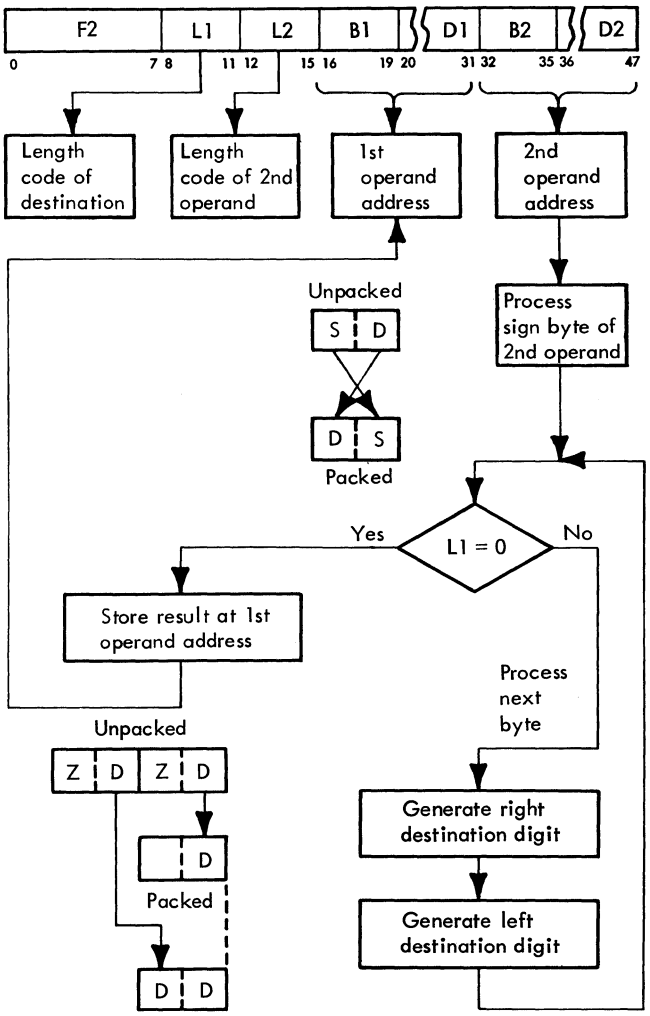
3.8.5 PACK, PACK (F2)

- Format of 2nd operand is changed from zoned to packed, and result is stored at 1st operand address.

- Separate microprogram used during word overlap.

- Interruptions:
  Protection.
  Addressing.

- SS format:



The Pack instruction assumes source data in the unpacked format. The low-order source byte consists of a sign (bits 0-3) and a digit (bits 4-7). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes consist of a zone (bits 0-3) and a digit (bits 4-7). Only the digits are gated to the destination field, with two bytes of source being processed for each byte of destination.

The sign and digits of the second operand are moved unchanged to the first operand field and are not checked for valid codes. A separate microprogram is provided for byte processing when a word-overlap condition exists as defined in paragraph 3.7.5.1. A test for word overlap is performed in the GIS of the instruction and also each time that a new double word of source is fetched from main storage.

The GIS microprogram for the Pack instruction is shown in Figure 6058, FEDM. This microprogram gates the second operand from the SDBO to AB and performs the word-overlap test.

The word-overlap test is performed in two steps. First, the double-word addresses for the destination and source are compared by subtracting D from the IC. The difference is then shifted 4 bit positions to the right and gated to PAL, and PAL(40-64) is sensed for an all-zero result to detect possible word overlap. [The right-4 shift is made to avoid comparison of byte addresses within the double word; i.e., the difference for the byte addresses is shifted to PAL(65-67), which is not sensed by the branch.] If the addresses for the double words of source and destination are different, no word-overlap condition exists. Thus, if PAL(40-64) is not zero, a branch is made to the appropriate not-word-overlap execution sequence of the instruction.

If PAL(40-63) equals zero, indicating that the same double-word address has been specified for source and destination, a second test must be made to verify whether special data handling is required. The contents of D are again subtracted from the IC, but this time a right-4 shift on the difference is not performed and the byte addresses within the same double word are compared. If PAL(30-63) equals zero, an identical address has been specified for both source and destination. Since this case of word overlap does not require special data handling (paragraph 3.7.5.1), a branch is made to the not-word-overlap microprogram. If, however, PAL (30-63) is not zero, the source and destination bytes are skewed; special data handling will be required in the execution phase and, accordingly, a branch is made to the appropriate microprogram.

3.8.5.1 Instruction Execution - Not Word Overlap

- Basic execution is as follows:
  Process sign byte, and test for exit conditions.
  If no exit conditions, process right destination digit.
  Process left destination digit, and test for exit conditions.

A detailed flow chart of execution of the Pack instruction without word overlap is shown in Figure 6059, FEDM. The major functional steps in the microprogram are as follows:

1. Process Sign Byte

   a. The selected AB byte is gated via the serial adder cross-gates to the selected ST byte. The mark trigger selected by STC is set.

   b. ABC, STC, L1, and L2 are decremented by 1.

   c. An exit is made to the appropriate routine if one or more of the counters (ABC, STC, L1, L2) was equal to zero prior to being stepped.

   d. If no exit is made, the next source byte is processed to obtain the right destination digit.

2. Generate Right Destination Digit

   a. Bits 4-7 of the selected AB byte are gated to SAA(4-7). No data is gated to SAA(0-3). The serial adder output is gated from SAL(0-7) to the selected ST byte.

   b. ABC and the L2 count are decremented by one count.

   c. If L2 equals zero prior to stepping, the remaining source bytes are extended with high-order zeros (see step 5).

   d. If ABC equals zero prior to stepping, an exit is made to the source fetch routine (see step 6). STAT G is set to cause a return to the left digit routine after source fetch.

3. Generate Left Destination Digit

   a. Bits 4-7 of the selected AB byte are gated to SAA(0-3). Bits 4-7 of the selected ST byte are gated to SAB(4-7). The serial adder output is gated back to the selected ST byte, and the mark trigger selected by STC is set.

   b. ABC, STC, L1, and L2 are decremented by one count. If none of these counters equalled zero prior to stepping, the right digit for the next destination byte is generated.

4. Exit Conditions

   An exit is made from the sign byte routine or from the left digit routine when one or more

of the following conditions are detected by the functional branch micro-order (DECIMAL):

a. L1 or STC equals zero.

b. L2 equals zero.

c. ABC equals zero.

When exit is on L1 or STC equals zero, a second test on L1-equal-all-1's is required to determine whether an end-op condition exists.

5. Extension of Source Bytes with High-Order Zeros

This routine is entered when the L2 count has stepped to zero before the L1 count has stepped to zero.

The serial adder output (zeros) is gated to the selected ST byte with the selected mark trigger being set. L1 and STC are decremented as each byte is processed. When L1 equals zero, the contents of ST are stored per D address and the common end-op routine is started. When STC equals zero, the contents of ST are stored, and D is decremented by 8. STAT H is set to cause a return to this routine after storing ST.

6. Source Fetch Routine

This routine is shared with the Move with Offset instruction. STAT D is set to cause a return to the pack microprogram.

The second operand is requested from main storage, and the IC is decremented by 8. A word-overlap test is performed as explained in the GIS. If no word-overlap condition exists, the next double word of second operand is gated from SDBO to AB. Processing of the left or right destination digit is resumed as determined by STAT G.

3.8.5.2 Instruction Execution - Word Overlap

- Basic execution is as follows:
  Process sign byte. Update AB, and test for exit conditions.
  If no exit conditions, process right destination digit.
  Process left destination digit, update AB, and test for exit conditions.

A detailed flow chart of Pack instruction execution under word-overlap conditions is shown in Figure 6060, FEDM. This microprogram is entered when a word-overlap condition is detected in the GIS or during source fetch. The major functional steps in the microprogram are as follows:

1. Process Sign Byte

The sign byte of the second operand in AB is processed in the same manner as in the not-word-overlap microprogram.

2. Update AB from ST

The data in AB is updated by transferring the contents of S to A or the contents of T to B, depending on the STC setting. ABC, STC, L1, and L2 counters are decremented by 1, and the mark trigger selected by STC is set.

If any counter value equalled zero prior to decrementing, an exit is made to the proper store, fetch, or extend-with-zeros routine as explained for the not-word-overlap sequence. If no exit conditions exist, processing of the right destination digit is started.

3. Generate Right Destination Digit

This routine is the same as in the not-word-overlap sequence.

4. Generate Left Destination Digit

This routine is the same as in the not-word-overlap sequence and is always followed by the update routine.

5. Source Fetch Routine

The next double word of source is requested from main storage, after which the IC is decremented by 8. Upon detection of a word-overlap condition, however, this double word is not used, since AB must be updated from ST. If, upon entering the source fetch routine, only the right destination digit has been placed in the selected ST byte, this byte is not transferred to AB. Instead, the following action takes place:

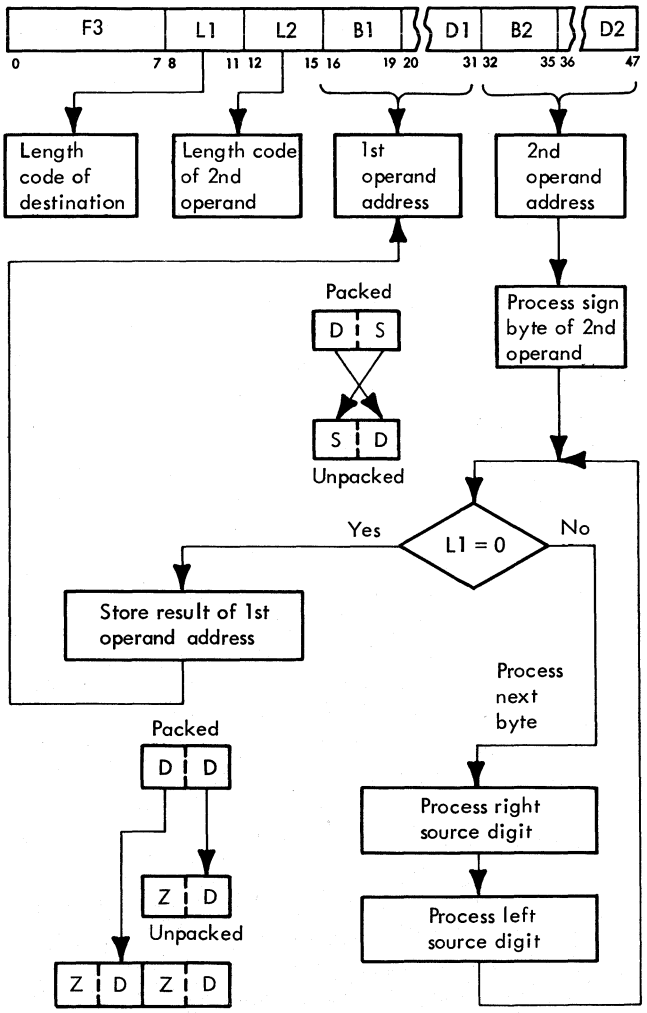
a. The portion of ST that has been processed (as determined by the mark triggers) is stored in the destination field, refetched from storage, and gated to both AB and ST.

b. If STAT G is set, indicating that only the right digit of the selected ST byte has been processed, the selected ST byte is transferred to F prior to ingating SDBO to ST. After ingating SDBO to ST, F is reinserted in the selected ST byte, and processing of the left digit is started.

c. If STAT G is not set, indicating that a complete ST byte has been processed, it is not necessary to save the selected ST byte. Processing of the right digit is started immediately.

3.8.6 UNPACK, UNPK (F3)

- Format of 2nd operand is changed from packed to zoned, and result is stored at 1st operand address.
- SS format:



- Separate microprogram used during word overlap.
- Word-overlap test is performed during GIS and in destination store and source fetch routines.
- Interruptions:
  Protection.
  Addressing.

The Unpack instruction assumes data in the packed format. The low-order source byte consists of a sign (bits 4-7) and a digit (bits 0-3). These two characters are swapped as they are gated to the low-order destination byte. All other source bytes contain a pair of decimal digits. Each digit is transferred to the low-order portion (bits 4-7) of the corresponding destination byte, and a zone character is inserted in the high-order portion byte (bits 0-3). During this transfer, the digits are not checked for validity.

A separate microprogram is provided for byte processing when a word-overlap condition exists. A test for a word-overlap condition is performed in the GIS of the instruction and also each time that a double word of data is fetched from or stored in main storage.

The Unpack instruction generates two bytes of destination for each byte of source. Therefore, the condition when the destination bytes are processed "ahead" of the source will always exist if the operand fields overlap. When the same double-word address is specified, special data handling is required regardless of how the operand bytes are arranged in this double word. Special handling is necessary each time that source data is fetched from main storage; also, upon storing unpacked data in the destination field, a word-overlap test must be made to determine whether the source data in the CPU must be updated from storage.

The GIS microprogram for the Unpack instruction is shown in Figure 6058, FEDM. When the first overlap indication occurs, the byte addresses are not checked. Instead, a branch is forced into the word-overlap sequence by supplying a hot carry to PAA(60), so that a test of PAL(30-63) will always yield a nonzero result.

3.8.6.1 Instruction Execution - Not Word Overlap

- Basic execution is as follows:
  Process sign byte and test for exit conditions.

If no exit conditions, process right source digit.
Process left source digit, and test exit conditions.

A detailed flow chart of Unpack instruction execution without word overlap is shown in Figure 6061, FEDM. The major functional steps in the microprogram are as follows:

1. Process Sign Byte

   The sign byte, selected by ABC, is gated via the serial adder cross-gates to the selected ST byte, and the corresponding mark trigger is set. ABC, STC, L1, and L2 are decremented by 1 count, and an exit is made if any counter equalled zero prior to stepping.

2. Process Right Source Digit

   a. Bits 4-7 of the selected AB byte are gated to SAA(4-7). The approved zone character is inserted in SAA(0-3). The serial adder output is gated to the selected ST byte, and the selected mark trigger is set.

   b. L1 and STC are decremented by 1.

   c. If L1 equalled zero prior to stepping, the contents of ST are stored and the common end-op sequence is started.

   d. If STC equalled zero prior to stepping (and L1 was not zero), the destination store routine is started. STAT G is set to record an exit from the right digit routine.

3. Process Left Source Digit

   a. Bits 0-3 of the selected AB byte are gated to SAA(4-7), and the zone character is inserted in SAA(0-3).

   b. The adder output is gated to the selected ST byte, and the selected mark trigger is set.

   c. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero prior to stepping. If no exit condition exists, the right source digit in the next source byte is processed.

4. Exit Conditions

   An exit is made from the byte processing routine whenever it is detected that L1, L2, ABC, or STC is equal to zero. Although a separate exit is provided for each possible combination of these conditions, they may be considered to be examined in the following order of priority:

   a. L1 = 0

      The contents of ST are stored per D address, and the common end-op routine is started.

   b. L2 = 0

      AB is cleared, ABC is set per L2 (which is 7), and STAT H is set to record the end of source field. If STC was also zero, the destination store routine is started. If STC was not zero, the high-order zeros routine is entered per STAT H.

   c. STC = 0

      The destination store routine is started. If ABC was also zero, STAT D is set to cause a source fetch after the destination store.

   d. ABC = 0.

      The source fetch routine is started.

5. Extension of Source Bytes with High-Order Zeros

   AB is cleared, and bits 4-7 of the selected AB byte (zeros) are gated to SAA(4-7); the approved zone character is inserted in SAA(0-3). The adder output is gated to the selected ST byte, and the corresponding mark trigger is set. L1 and STC are decremented once for each byte that is processed. An exit is made to the destination store routine when STC steps to zero, and to end-op when L1 steps to zero.

6. Source Fetch Routine

   A request is made per the IC address, after which the IC is decremented by 8. A word-overlap test is made. If there is no word-overlap condition, the next source word is

gated to AB, and the right digit of the next source byte is processed.

7. Destination Store Routine

   a. The contents of ST are stored in the destination field per D address, and D is decremented by 8.

   b. An exit is made to the source fetch routine if STAT D is set.

   c. An exit is made to the high-order zeros routine if STAT H is set.

   d. If neither STAT D nor STAT H is set, a word-overlap test is made by comparing the IC and D addresses. If no word overlap exists, the left or right digit is processed as determined by STAT G.

3.8.6.2 Instruction Execution - Word Overlap

- Basic execution is as follows:
  Process sign byte. Update AB, and test for exit conditions.
  If no exit conditions, process right source digit.
  Process left source digit, and test for exit conditions.

- Word-overlap test performed during source fetch and destination store routines.

A detailed flow chart of Unpack instruction execution under word-overlap conditions is shown in Figure 6062, FEDM. The steps in which this microprogram differs from that for not-word-overlap are explained below:

1. Process Sign Byte

   This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

2. Update AB from ST

   If STC is less than 4, the contents of S are transferred to A; the contents of T are always transferred to B. The mark trigger selected by STC is set. ABC, STC, L1, and L2 are decremented by 1. An exit is made to the appropriate routine if any of the above counters equalled zero prior to their being stepped. If no exit conditions exist, the right digit in the next source byte is processed.

3. Process Right Source Digit

   The right source digit is processed in the same manner as for not-word-overlap. If upon processing the right digit an exit is made on STC equal zero, and ABC is not zero, the contents of S are transferred to A. In this manner, the source is correctly updated prior to storing ST.

4. Process Left Source Digit

   This step is the same as in the not-word-overlap sequence except that it is always followed by the update routine.

5. Source Fetch Routine

   a. The source is requested per the IC address, after which the IC is decremented by 8.

   b. The contents of D are subtracted from the IC to prepare for the word-overlap test; also, a test on STC equals 7 is made to establish how the source is to be updated in case of an overlap condition.

   c. The condition when STC equals 7 indicates that STC was zero prior to entering the source fetch routine. In this case, the destination has been stored in main storage. Thus, to update the source, the double word at the SDBO is gated to AB and ST, and processing of the left source digit is started.

   d. If STC is not 7, AB must be updated from ST. After transfer of the contents of ST to AB, processing of the right source digit is started.

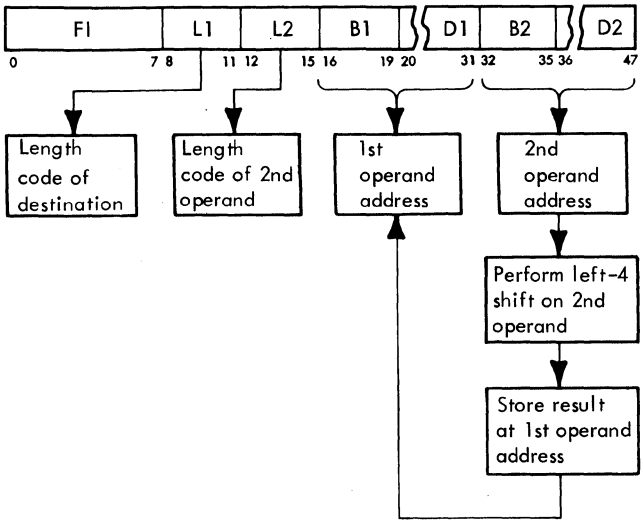
3.8.7 MOVE WITH OFFSET, MVO (F1)

- 2nd operand is stored to left and adjacent to low-order 4 bits of 1st operand.

- Separate microprogram used during word overlap.

- Interruptions.
  Protection.
  Addressing.

- SS format:



The MVO instruction performs a left-4 shift on the second operand and transfers the result to the first operand location. Thus, the four low-order bits of the first operand are preserved as the lowest-order character of the second operand. During execution of the instruction, the operand signs and digits are not checked for valid codes.

The decimal instruction set includes no shift instructions, since the equivalent of a shift can be obtained by programming. Programs for right or left shift, and for an even or an odd shift amount, are written with Move with Offset instruction and the logical move instructions described in Section 5.

A separate microprogram is provided for byte processing when a word-overlap condition exists as defined in paragrpah 3.7.5.1. A test for word overlap is performed in the GIS of the instruction, and also each time that a new double word of source is fetched from main storage.

The GIS for the Move with Offset instruction is shown in Figure 6058, FEDM. This microprogram is identical with that of the Pack instruction.

3.8.7.1 Instruction Execution - Not Word Overlap

- Basic execution is as follows:
  Transfer bits 4-7 of selected AB byte to bits 0-3 of selected ST byte. Decrement counters.
  Transfer bits 0-3 of selected AB byte to bits 4-7 of selected ST byte. Repeat first step.
  Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A detailed flow chart of the execution of the Move with Offset instruction when no word-overlap condition exists is shown in Figure 6063, FEDM. Basically, this microprogram specifies a 2-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

1. Cycle 1

   a. Bits 4-7 of the selected AB byte are gated to SAA(0-3).

   b. Bits 4-7 of the selected ST byte are gated to SAB(4-7).

   c. The serial adder output is gated back to the selected ST byte, and the corresponding mark trigger is set.

   d. L1 and STC counters are decremented 1 count. An exit is made to the destination store routine if L1 or STC equalled zero prior to stepping.

2. Cycle 2

   a. Bits 0-3 of the selected AB byte are gated to SAA(4-7). No data is gated to serial adder bits 0-3.

   b. The serial adder output is gated to the selected ST byte.

   c. L2 and ABC counters are decremented 1 count. If L2 was zero prior to stepping, an exit is made to the high-order zero extend routine. If L2 was not zero but ABC equalled zero, an exit is made to the source fetched routine.

   d. If L2 or ABC is not equal to zero, cycle 1 is repeated.

3. High-Order Zero Extend Routine

   An entry is made into this routine when the last source byte has been processed. The selected ST byte contains the high-order source digit in bits 4-7; bits 0-3 are zeros.

   The following actions are performed upon entry into the routine:

   a. STAT H is set.

   b. The selected mark trigger is set.

   c. L1 and STC are decremented 1 count.

d. If L1 or STC equals zero prior to stepping, an exit is made to the destination store routine.

If L1 or STC is not zero, a 1-cycle loop is started, which:

a. Gates the serial adder output (zeros) to the selected ST byte.

b. Sets the mark trigger selected by STC.

c. Decrements L1 and STC by 1 count.

d. Exits to the destination store routine when L1 or STC equals zero. (STAT H is set to cause re-entry into the high-order zeros routine after the destination is stored.)

4. Destination Store Routine

a. The contents of ST are stored in the destination field per D address.

b. A test is made for the end of the destination field. If the L1 count now equals all 1's, an exit is made to the common end-op sequence.

c. If L1 is not all 1's, D is decremented by 8.

d. If STAT H is set, the high-order zeros routine is resumed. If STAT H is not set, the byte processing loop is started at cycle 2.

5. Source Fetch Routine*

a. The source is requested from storage, and the IC is decremented by 8.

b. A word-overlap test is made by comparing the IC and D addresses.

c. If no word-overlap condition exists, the double word arriving from storage is gated to AB, and byte processing is resumed.

3.8.7.2 Instruction Execution - Word Overlap

- Basic execution is as follows:
  Transfer bits 4-7 of selected AB byte to bits 0-3 of selected ST byte.
  Transfer bits 0-3 of selected AB byte to bits 4-7 of selected ST byte.
  Update AB from ST, and repeat first step.
  Exit on L1 or STC = 0, L2 = 0, or ABC = 0.

A detailed flow chart of the execution of the Move with Offset instruction when a word-overlap condition exists is shown in Figure 6064, FEDM. Basically, this microprogram specifies a 3-cycle loop with appropriate exits to source fetch, destination store, high-order-zero extend, and end-op routines.

1. Cycle 1

This cycle is identical with that in the not-word-overlap microprogram.

2. Cycle 2

a. Bits 0-3 of the selected AB byte are gated to SAA(4-7).

b. The serial adder output is gated to the selected ST byte.

3. Cycle 3

a. If STC is less than 4, the contents of S are transferred to A.

b. The contents of T are transferred via the parallel adder to B.

c. The L2 and ABC count are decremented by 1.

d. An exit is made to the high-order zeros routines if L2 was equal to zero prior to stepping. An exit is made to the source fetch routine if ABC was equal to zero and L2 was not zero.

*This routine is shared with the Pack instruction. Return to the appropriate microprogram is effected per STAT D.

e. If no exit conditions exist, cycle 1 is repeated for the next byte.

The high-order zeros and destination store routines are the same as in the not-word-overlap sequence. The source fetch routine, however, is different.

4. Source Fetch Routine

Upon detecting a word-overlap condition, the source from main storage is not used. Instead, AB is updated from ST: if STC is equal to 7, the contents of T are transferred to B; if STC is not 7, the contents of S are transferred to A and the contents of T to B.

This section describes the general handling and specific execution sequences used by the logical instruction set. These instructions provide for logical manipulation of data: moving, comparing, bit testing, bit connecting, translating, editing, and shifting. The logical instructions use all five instruction formats and work with both fixed and variable field length (VFL) data.

3.9 INTRODUCTION

The logical instructions operate on data which may range from 1 to 256 bytes in length. The operands are obtained either from the main storage or from a general register in the CPU. Sometimes, the operand may be contained in the instruction itself.

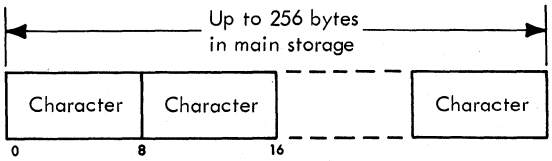
Processing of data in main storage proceeds from the high-order to the low-order address, or from left to right. The initial byte selected for processing may be at either an odd or even main storage address. As a rule, processing of data in a general register involves the complete register contents. Except for the editing instructions, data is not treated as numbers.

3.9.1 DATA FORMAT

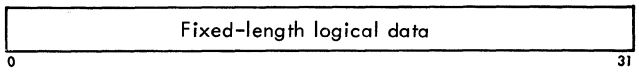
- Fixed or variable-field length.
- Operands in instruction itself are called "immediate operands."

The data size may be a single character, a single word, a double word, or variable in length. The data format is dependent on the type of operation performed:

1. In storage-to-storage operations, data has a VFL format, starting at any byte address and continuing for a maximum of 256 bytes.



2. In storage-to-register operations, the main storage data occupies either a word of 32 bits or a byte of eight bits. The word must be located on word boundaries; that is, the low-order two bits of its address must be zero. Data in general registers normally occupies all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of the register participate, leaving the remaining 24 bits unchanged. In some operations, 64 bits of an even/odd pair of registers participate.



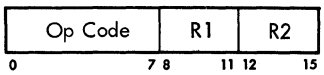
3. In operations introducing data directly from the instruction, as an immediate operand, data is restricted to an 8-bit byte. Only one byte may be introduced per instruction, and only one byte participated in main storage.

3.9.2 INSTRUCTION FORMAT

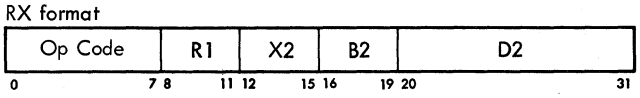
- RR, RX, RS, SI, and SS.

Logical instructions use the following five formats:

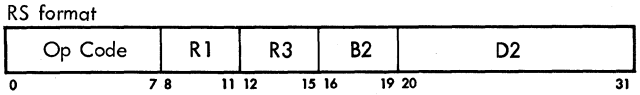
RR format



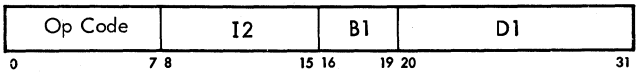
RX format



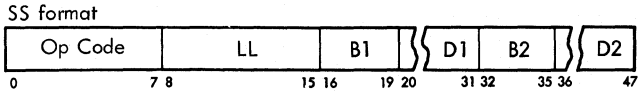
RS format



SI format



SS format



In the RR, RX, and RS formats, the contents of the register specified by R1 are called the "first operand." In the SI and SS formats, the contents of the general register specified by B1 are added to the contents of the D1 field to form an address. This address designates the leftmost byte of the first operand field. The number of bytes to the right of this first byte is specified by the LL field in the SS instruction. In the SI format, the operand size is one byte.

In the RR format, the R2 field specifies the register containing the second operand. The same register may be specified for the first and second operands.

In the RX format, the contents of the general registers specified by the X2 and B2 fields are added to the contents of the D2 field to form the address of the second operand.

In the RS format, used for shift operations, the contents of the general register specified by the B2 field are added to the contents of the D2 field. This sum is not used as an address but specifies the number of bits of the shift. The R3 field is ignored in the shift operations.

In the SI format, the second operand is the 8-bit immediate data field, I2, of the instruction.

In the SS format, the contents of the general register specified by B2 are added to the contents of the D2 field to form the address of the second operand. The second operand field has the same length as the first operand field.

A zero in the X2, B1, or B2 field indicates the absence of the corresponding address or shift-amount components. An instruction can specify the same general register both for address modification and for operand location. Address modification is always completed prior to operation execution.

3.9.3 DATA HANDLING

Generally, the operands are treated as 8-bit bytes. In a few cases, the left or right four bits of a byte are treated separately or operands are shifted a bit at a time. Except for editing instructions, data is not treated as numbers.

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

The contents of all general registers and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the result locations, general register 1 in the Edit and Mark instruction, and general registers 1 and 2 in the Translate and Test instruction.

Editing operations provide transformation from packed decimal digits to alphanumeric characters; i.e., editing requires a packed decimal field and generated zoned decimal digits. The digits, signs, and zones are recognized and generated as for decimal arithmetic; all bit configurations are considered valid.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the content of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function. The list is specified by an initial address — the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The address used to address the list is obtained by adding the argument to the low-order positions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. Where it is known that not all 8-bit argument values will occur, it may be possible to reduce the size of the list.

Use of general register 1 is implied in Edit and Mark and in Translate and Test instructions. A 24-bit address may be placed in this register during these operations. The Translate and Test instruction also implies general register 2. The low-order eight bits of register 2 may be replaced by a function byte during a translate-and-test operation.

3.9.4 CONDITION CODE SETTING

The results of most logical operations are used to set the CC in the PSW. The Load Address, Insert Character, Store Character, Translate, and the moving and shift instructions leave this code unchanged. (The CC can be used for decision-making by subsequent branch-on-condition instructions.)

The CC can be set to reflect five types of results for logical operations. For the Compare Logical instructions, the 0, 1, and 2 states indicate that the first operand is equal, low, or high.

For the logical connectives, the states 0 and 1 indicate a zero or nonzero result field.

For the Test under Mask instruction, the states 0, 1, and 3 indicate that the selected bits are all-zero, mixed zero and 1, or all-1.

For the Translate and Test instruction, the states 0, 1, and 2 indicate an all-zero function byte, a non-zero function byte with the operand incompletely tested, or a last function byte nonzero.

For editing, the states 0, 1, and 2 indicate a zero, less-than zero, or greater-than-zero content of the last result field. Table 3-20 lists those instructions that affect the CC and indicates how the CC is set.

TABLE 3-20. CONDITION CODES FOR LOGICAL INSTRUCTIONS

Instruction	Condition Code			
	0	1	2	3
Compare Logical	Equal	Low	High	-
AND	Result = 0	Result ≠ 0	-	-
OR	Result = 0	Result ≠ 0	-	-
Exclusive OR	Result = 0	Result ≠ 0	-	-
Test under Mask	Result = 0	Mixed result	-	Result = 1
Translate and Test	Result = 0	Incomplete result	Complete result	-
Edit	Result = 0	Result < 0	Result > 0	-
Edit and Mark	Result = 0	Result < 0	Result > 0	-

3.9.5 INTERRUPTION CONDITIONS

- Protection.
- Addressing.
- Specification.
- Data.

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

1. Protection — The storage key of a result location in main storage does not match the protection key in the PSW. The operation is suppressed. Therefore, the CC and data in registers and main storage remain unchanged. The only exceptions are the variable-length storage-to-storage operations, which are terminated. For terminated operations, the result data and CC, if affected, are unpredictable and should not be used for further computation.

2. Addressing — An address designates a location outside the available storage for the installed system. The operation is terminated. The result data and the CC, if affected, are unpredictable and should not be used for further computation.

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

4. Data — A digit code of the second operand in the Edit or Edit and Mark instruction is invalid. The operation is terminated. The result data and the CC are unpredictable and should not be used for further computation.

Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of the Load Address instruction is not tested. The address restrictions do not apply to the components from which an address is generated — the contents of the D1 and D2 fields, and the contents of the registers specified by X2, B1, and B2.

3.10 INSTRUCTION EXECUTION

The following paragraphs describe the execution sequences for instructions in the logical set. Table 3-21 lists the instructions with their respective formats, mnemonic and operation codes, and program interruptions.

3.10.1 MOVE

- 2nd operand is placed in 1st operand location.

TABLE 3-21. LOGICAL INSTRUCTION SET

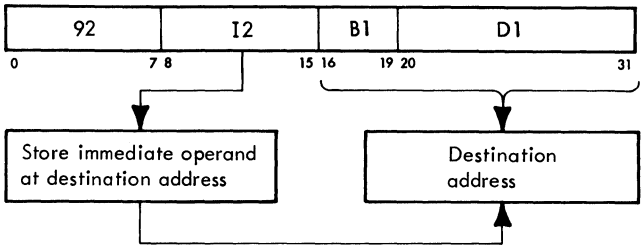
Instruction	Format	Mnemonic Code	Op Code	Interruptions*
Move	SI	MVI	92	P, A
Move	SS	MVC	D2	P, A
Move Numerics	SS	MVN	D1	P, A
Move Zones	SS	MVZ	D3	P, A
Compare Logical	RR	CLR	15	
Compare Logical	RX	CL	55	A, S
Compare Logical	SI	CLI	95	A
Compare Logical	SS	CLC	D5	A
AND	RR	NR	14	
AND	RX	N	54	A, S
AND	SI	NI	94	P, A
AND	SS	NC	D4	P, A
OR	RR	OR	16	
OR	RX	O	56	A, S
OR	SI	OI	96	P, A
OR	SS	OC	D6	P, A
Exclusive OR	RR	XR	17	
Exclusive OR	RX	X	57	A, S
Exclusive OR	SI	XI	97	P, A
Exclusive OR	SS	XC	D7	P, A
Test under Mask	SI	TM	91	A
Insert Character	RX	IC	43	A
Store Character	RX	STC	42	P, A
Load Address	RX	LA	41	
Translate	SS	TR	DC	P, A
Translate and Test	SS	TRT	DD	A
Edit	SS	ED	DE	P, A, D
Edit and Mark	SS	EDMK	DF	P, A, D
Shift Left Single Logical	RS	SLL	89	
Shift Right Single Logical	RS	SRL	88	
Shift Left Double Logical	RS	SLDL	8D	S
Shift Right Double Logical	RS	SRDL	8C	S

* A: Addressing
D: Invalid data
P: Protection
S: Specification

- Interruptions:
  Protection.
  Addressing.
- Instruction uses SI or SS format.

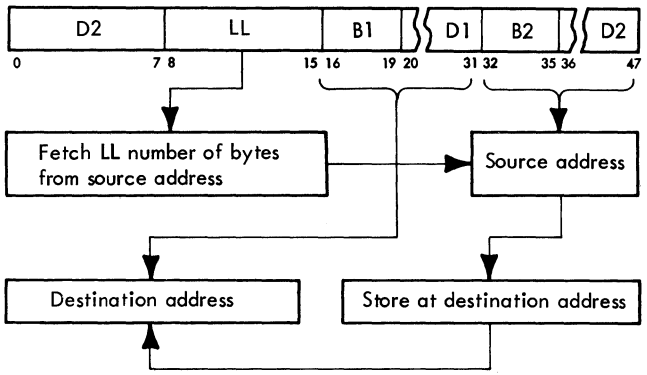
3.10.1.1 Move, MVI (92)

- SI format:



3.10.1.2 Move, MVC (D2)

- SS format:



- Move operation can be high or low speed.
- Three separate microprograms are provided:
  High-speed move.
  Word overlap.
  Low-speed move.

Three separate sequences are provided for the MVC instruction. The high-speed move sequence is used when it is possible to transfer a double word of data at a time. This condition exists when the high-order bytes of the source and destination are specified on double-word boundaries and a full double word of data remains to be processed; i.e., both ABC and

STC are equal to zero, and the LL count is greater than 6. The word-overlap sequence is used when a word-overlap condition exists (paragraph 3.7.5.1). The second operand in AB is updated after each AB byte is processed. The low-speed move sequence is used when the high-speed or word-overlap condition does not exist. (The high-speed and word-overlap conditions are detected in the GIS of the instruction.)

1. Low-Speed Move Sequence

   This sequence is basically a 1-cycle operation in which the AB byte selected by ABC is transferred through the serial adder to the ST byte selected by STC, and the mark trigger selected by STC is set.

   The STC and ABC are incremented, the LL count in E(8-15) is decremented, and the cycle is repeated for the next byte, unless an exit condition exists.

2. Word Overlap Move Sequence

   This sequence is a 2-cycle sequence in which the first cycle transfers the AB byte, selected by ABC, to the ST byte selected by STC. The second cycle updates the source operand in AB by transferring S to A, or T to B, as determined by the value of STC. The mark trigger selected by STC is set. The STC and ABC are incremented, the LL count is decremented, and the sequence is repeated for the next byte, unless an exit condition exists.

3. High-Speed Move Sequence

   This routine is entered from the GIS or from the low-speed move routine.

   a. When the entrance is made from the GIS, the source operand has been transferred to ST. The contents of ST are stored by setting mark triggers 0-7 and issuing a storage request per D.

   b. The LL count in E(8-15) is decremented by 8 via the parallel adder and is then tested for all 1's. If this condition exists, an end-op sequence is started. If no end-op condition exists, the IC is incremented by 8 via the parallel adder and a source fetch request is given.

   c. When the entrance is made from the low-speed routine, D is incremented by 8 and the source double word from main storage is gated to both AB and ST. If at least 8 bytes remain to be processed, as determined by an ROS branch on LL count being greater than 6, the high-speed move sequence is repeated (starting at step a). If fewer than 8 bytes remain to be processed, the low-speed move sequence is started to process the remaining data.

Exit is made from the low-speed or word-overlap move routines if one of the following conditions exists: (1) LL = 0, or STC = 7 and ABC $\neq$ 7, (2) LL = 0, or STC = 7 and ABC = 7, (3) only ABC = 7. A separate sequence is entered for each of these conditions as explained below:

1. LL = 0, or STC = 7 and ABC $\neq$ 7

   A destination store is initiated, and a test for an end-op condition is made. If LL count now equals all 1's, an entry is made into a common end-op sequence. If an end-op condition does not exist, D is incremented by 8 via the parallel adder and low-speed move sequence is continued.

2. LL = 0 or STC = 7 and ABC = 7

   A destination store is initiated, and a test for end-op is made (LL = all 1's). A further test for a high-speed move condition is made. If at this time the LL count is 7 or greater, the IC and D are incremented by 8, a source fetch is initiated, and an entry is made into the high-speed move sequence. If neither an end-op nor a high-speed move condition exists, D is incremented by 8 and a common source fetch routine is entered which will increment the IC by 8, fetch the next double word of source to AB, and test for a word-overlap condition. Since there is no word-overlap at this time (ABC = STC), the low-speed move sequence is continued.

3. ABC = 7

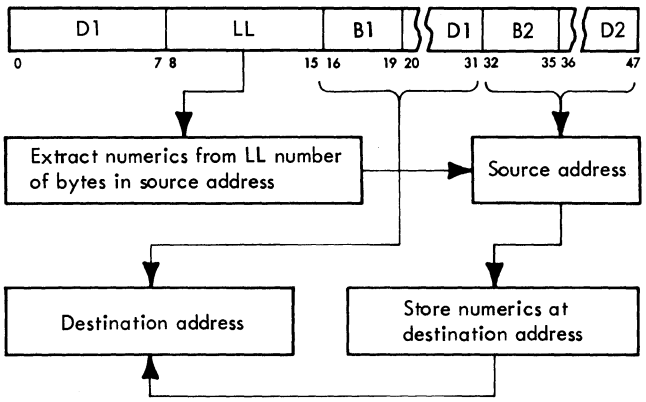
   The IC is incremented by 8 through the parallel adder, and a fetch request is given to fetch the next double word of source operand. The common source fetch sequence is entered, which will test for word overlap. In this case, word overlap may exist: if it is detected, the source operand from main storage is not gated to AB, but instead ST is gated to AB and a branch is made to the move-word-overlap sequence. If no word overlap exists, the low-speed move

sequence is continued after the source operand from main storage is gated to AB.

The common end-op routine is entered when the LL field has been decremented to zero. This routine restores the instruction address from the LSWR to the IC and resets STAT G (since it may have been used during the GIS).

3.10.2 MOVE NUMERICS, MVN (D1)

- Low-order four bits of each byte in 2nd operand field, numeric, are placed in low-order bit positions of corresponding bytes in 1st operand field.
- SS format:



- Separate sequence used for word overlap.
- Interruptions:
  Protection.
  Addressing.

The MVN instruction performs as follows:

1. Bits 4-7 of selected AB byte are gated to SAA(4-7).
2. Bits 0-3 of selected ST byte are gated to SAB (0-3).
3. Adder output is gated back to selected ST byte.

Data is processed one byte at a time, and the fields may overlap in any way. Separate sequences are used for the not-word-overlap and the word overlap conditions.

1. Not-Word-Overlap Sequence

   This sequence consists of a 1-cycle loop with an exit when LL = 0, STC = 7, or ABC = 7. As each byte is processed, the corresponding mark trigger is set per STC; ABC and STC are incremented by one count, and LL is decremented by one count.

2. Word Overlap Sequence

   This sequence consists of a 2-cycle loop with an exit when ABC or STC = 7, or when LL = 0.

   a. Cycle 1

      Numeric (bits 4-7) is moved from AB to ST.

   b. Cycle 2

      The contents of S are transferred to A, or the contents of T are transferred to B as determined by the STC value. The mark trigger selected by STC is set; STC and ABC are incremented one count, and LL is decremented one count.

An exit from the byte processing sequence is made when LL = 0, STC = 7, or ABC = 7. A separate sequence is entered for each of these conditions as explained below:

1. LL = 0

   The contents of ST are stored per D in the destination field. The common end-op sequence is started.

2. STC = 7

   The common destination store-fetch routine is started. If ABC also equals 7, STAT D is set to cause a source fetch prior to resuming the byte processing loop.

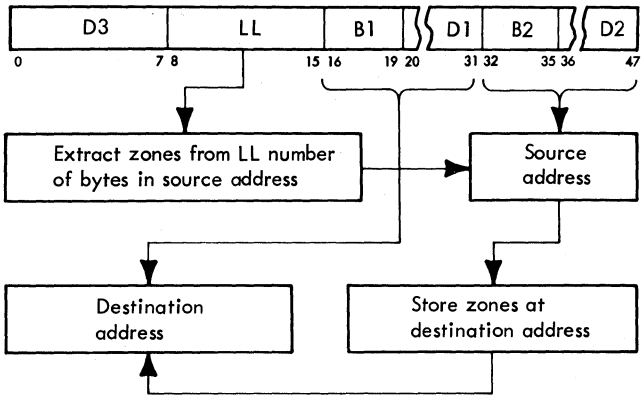
3. ABC = 7

   The common source fetch routine is started, which includes a word-overlap test, which will cause the appropriate instructions word-overlap or not-word-overlap loop to be continued.

3.10.3 MOVE ZONES, MVZ (D3)

- High-order four bits of each byte in 2nd operand field, zones, are placed in high-order four bit positions of corresponding bytes in 1st operand field.

- SS format:



- Separate microprogram used for word overlap.

- Interruptions:
  Protection.
  Addressing.

The MVZ instruction specifies the following actions:

1. Bits 0-3 of the selected AB byte are gated to SAA(0-3).

2. Bits 4-7 of the selected ST byte are gated to SAB(4-7).

3. The adder output is gated back to the selected ST byte.

Except for the above actions, the byte processing sequence is the same as that for the MVN instruction (paragraph 3.10.2).

3.10.4 COMPARE LOGICAL

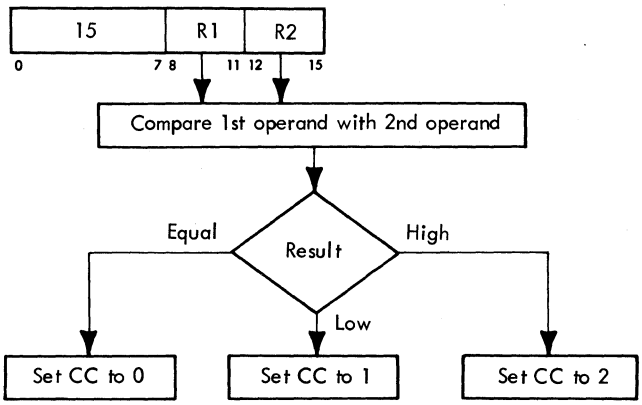
- 1st operand is compared with 2nd operand and CC is set as follows:
  Operands are equal, CC = 0
  1st operand is low, CC = 1
  1st operand is high, CC = 2

- Comparison is binary, and all codes are valid. Operation is terminated when an inequality is found.

- Instruction uses RR, RX, SI, or SS format.

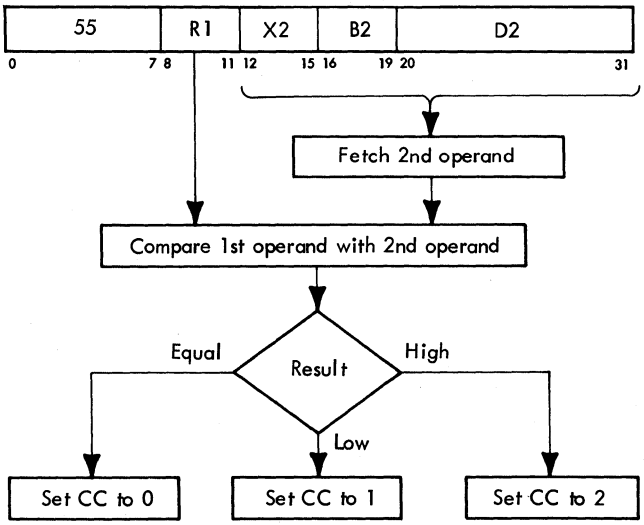
3.10.4.1 Compare, CLR (15)

- RR format:



3.10.4.2 Compare, CL (55)

- RX format:

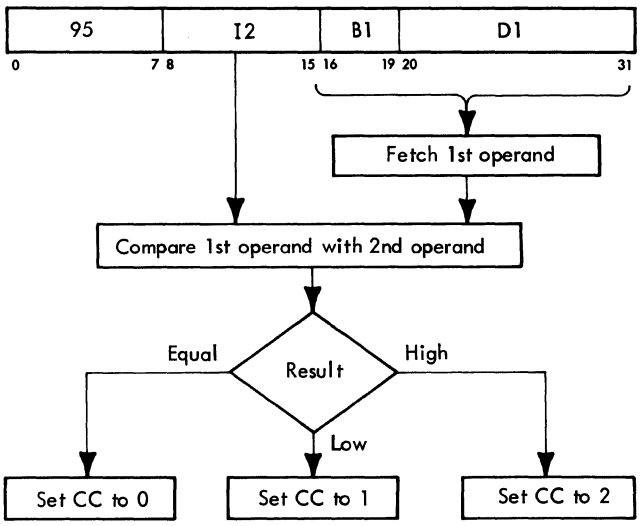


- Interruptions:
  Addressing.
  Protection.

3.10.4.3 Compare, CLI (95)

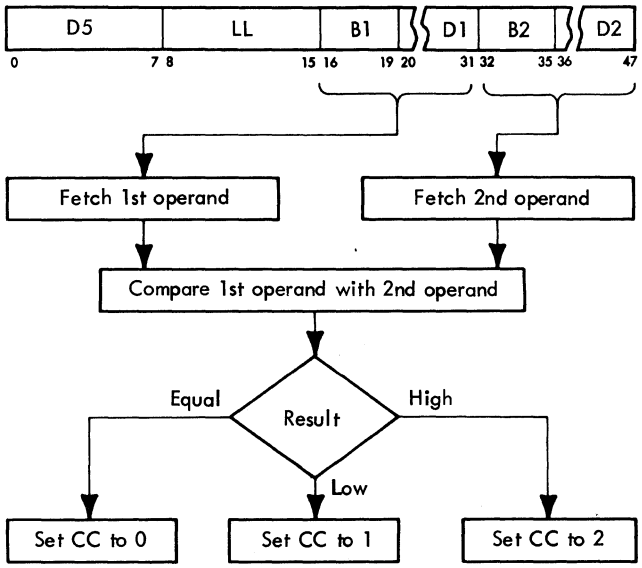
- SI format (see format on following page)

- Addressing interruption may result.



3.10.4.4 Compare, CLC (D5)

- SS format:



- Addressing interruption may result.
- Since results of operation are not stored in main storage, no special action is required during word overlap.

Sequencing of the CLC instruction is as follows:

1. The selected AB byte is gated complement to the serial adder with a hot carry to bit 7.
2. The selected ST byte is gated true to the serial adder.
3. The serial adder carry is saved in STAT H.
4. STAT A is set if a nonzero result byte is detected.
5. As each byte is processed, the LL count is decremented and the ABC and STC are incremented.
6. The above routine is continued until a nonzero result is detected in the serial adder, or until the LL count is stepped to zero, with exits for operand fetches when STC or ABC is stepped to 7.
7. If an exit is made because a nonzero byte is detected, one additional byte will have been gated to the serial adder before the exit is made via the ROS branch. Therefore, STAT H will reflect the carry of the nonzero result byte plus 1. Since STAT H is used to determine the setting of the CC, it is set or reset per the carry of the first nonzero byte encountered.
8. The common end-op routine is used, which will set the CC per following hardware conditions:

Hardware Conditions	CC Setting
STAT A is reset - equal compare	0
STAT A is set - STAT H is reset	1
STAT A is set - STAT H is set	2

3.10.5 AND

- Logical AND product of bits of 1st and 2nd operands is placed at 1st operand location. Operands are treated as unstructured logical quantities, and connective AND is applied bit by bit. All operands and results are valid.
- Instruction uses RR, RX, SI, or SS format.
- CC setting:
  Result is zero, CC = 0
  Result is not zero, CC = 1

The AND instruction mixes two operands on a logical AND basis. An AND operation is defined as follows: if both operand bits are 1, the resulting bit is 1; otherwise, the result is zero. The following example illustrates the AND'ing of two bytes:

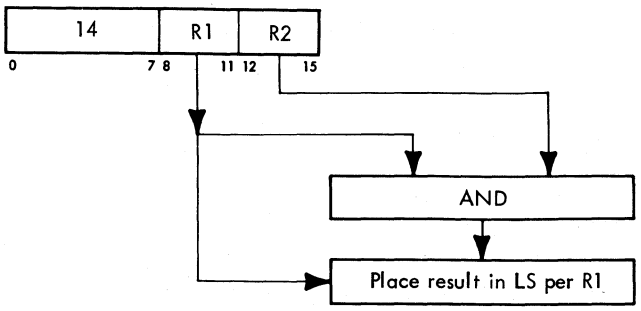
Bit positions	0 1 2 3 4 5 6 7
1st operand	1 0 1 0 1 0 1 0
2nd operand	1 0 0 1 1 1 0 0
Result	1 0 0 0 1 0 0 0

Note that only in bit positions 0 and 4 are both operand bits set to 1. Therefore, only bits 0 and 4 of the result are set to 1.

The AND operation may be executed by an instruction using the RR, RX, SI, or SS format. These instructions are described in the following paragraphs.

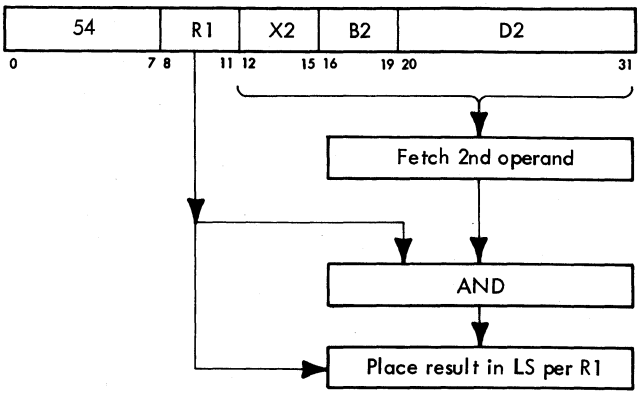
3.10.5.1 AND, NR (14)

- Contents of LS register specified by R1 are AND'ed with contents of LS register specified by R2. Result is placed in LS per R1 address.
- RR format:



3.10.5.2 AND, N (54)

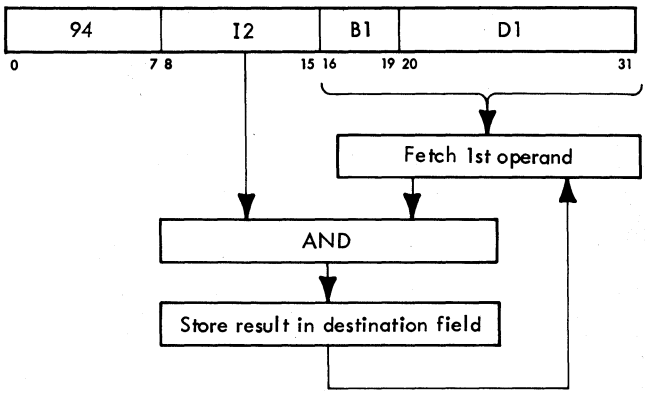
- Contents of LS register specified by R1 are AND'ed with 2nd operand (obtained from main storage). Result is placed in LS per R1 address.
- RX format:



- Interruptions:
  Addressing.
  Specification.

3.10.5.3 AND, NI (94)

- 1st operand is fetched from main storage and AND'ed with immediate operand in I2 field. Result is stored at 1st operand address.
- SI format:

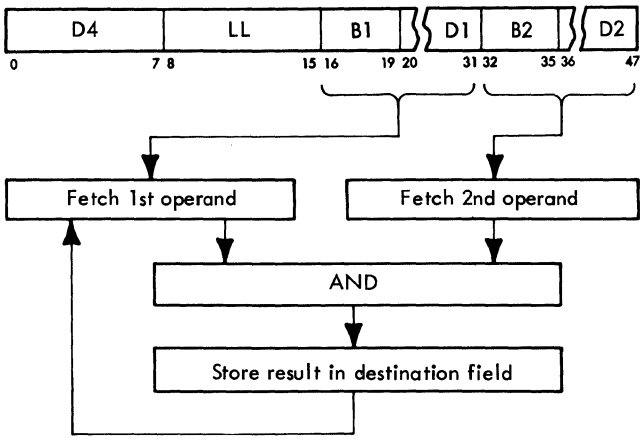


- Interruptions:
  Protection.
  Addressing.

3.10.5.4 AND, NC (D4)

- AND product of 1st and 2nd operands is stored at 1st operand address.

- SS format:



- Interruptions:
  Protection.
  Addressing.

The NC instruction specified the following actions:

1. The selected AB byte is gated to SAA(0-7).
2. The selected ST byte is gated to SAB(0-7).
3. Each AB and ST bit is combined using the serial adder AND function.
4. The adder output is gated back to ST. (STAT A is set if the result byte is not zero.)

Except for the above actions, the byte processing sequence is the same as that for the MVN instruction described in paragraph 3.10.2.

3.10.6 OR

- Logical OR sum of bits of 1st and 2nd operands is placed in 1st operand location. Operands are treated as unstructured logical quantities, and connective Inclusive OR is applied bit by bit. All operands and results are valid.
- Instruction has RR, RX, SI, or SS format.
- CC setting:
  Result is zero, CC = 0.
  Result is not zero, CC = 1.

The OR instruction mixes two operands on a logical OR basis. An OR operation is defined as follows: if either operand bit is a 1, the resulting bit is a 1: otherwise, the result is zero. The following example illustrates the OR'ing of two bytes.

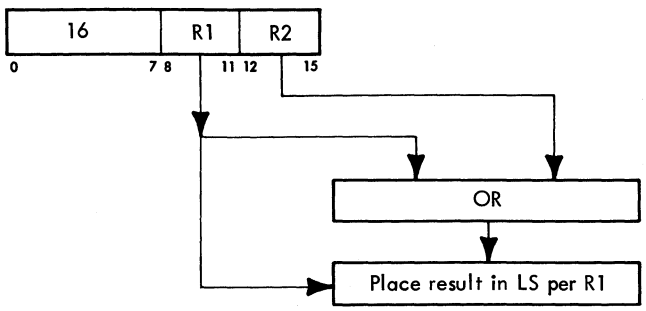
Bit positions	0 1 2 3 4 5 6 7
1st operand	1 0 1 0 1 0 1 0
2nd operand	1 0 0 1 1 1 0 0
Result	1 0 1 1 1 1 1 0

Note that only in bit positions 1 and 7 is neither bit set to 1. Thus, only bits 1 and 7 of the result are set to zero, and the remaining bits are set to 1.

The OR operation may be executed by an instruction in the RR, RX, SI, or SS format. The individual instructions are described in the following paragraphs.

3.10.6.1 OR, OR (16)

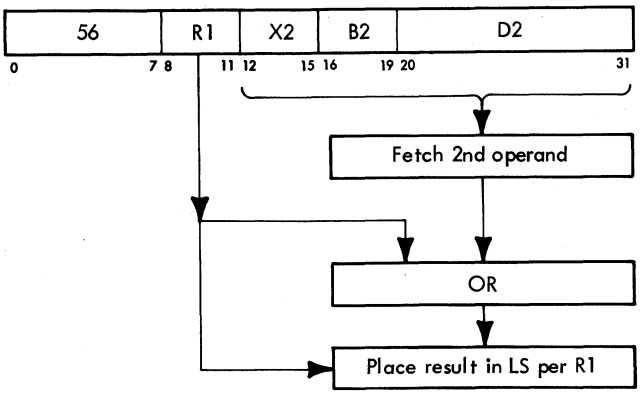
- Contents of LS register specified by R1 are OR'ed with contents of LS register specified by R2. Result is placed in LS per R1 address.
- RR format:



3.10.6.2 OR, O (56)

- Contents of LS register specified by R1 are OR'ed with 2nd operand (obtained from main storage). Result is placed in LS per R1 address.

- RX format:

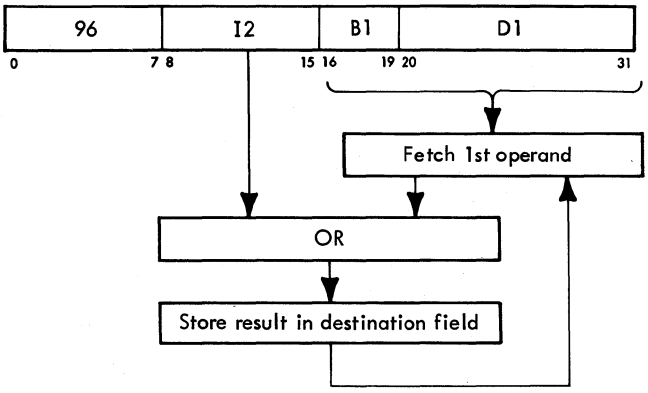


- Interruptions:
  Addressing.
  Specification.

3.10.6.3 OR, OI (96)

- 1st operand is fetched from main storage and OR'ed with immediate operand in I2 field. Result is stored at 1st operand address.

- SI format:

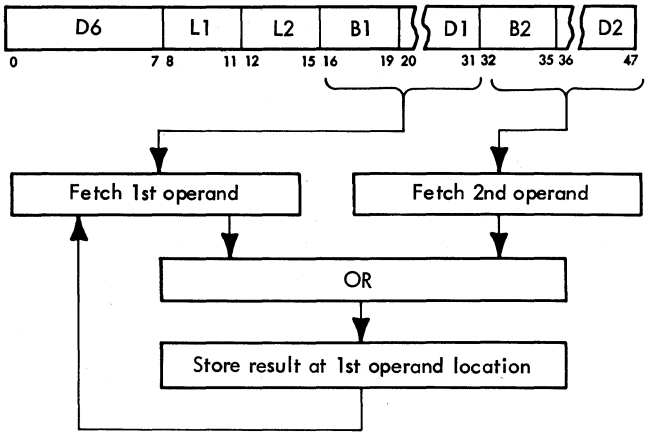


- Interruptions:
  Protection.
  Addressing.

3.10.6.4 OR, OC (D6)

- OR sum of 1st and 2nd operands is stored at 1st operand address.

- SS format:



- Interruptions:
  Protection.
  Addressing.

The OC instruction specifies the following actions:

1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder OR function.

2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per STC.

3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the MVN instruction described in paragraph 3.10.2.

3.10.7 EXCLUSIVE OR

- Module-2 sum (Exclusive OR) of bits of 1st and 2nd operands is placed at 1st operand location. Operands are treated as unstructured logical quantities, and connective Exclusive OR is applied bit by bit. All operands and results are valid.

- Instruction uses RR, RX, SI, or SS format.

- CC setting:
  Result is zero, CC = 0.
  Result is not zero, CC = 1.

The Exclusive OR instruction mixes two operands on a logical Exclusive OR basis. An Exclusive OR operation is defined as follows: if one and only one of the operand bits is a 1, the resulting bit is a 1; otherwise, the result is zero. The following example illustrates the Exclusive OR'ing of two bytes.

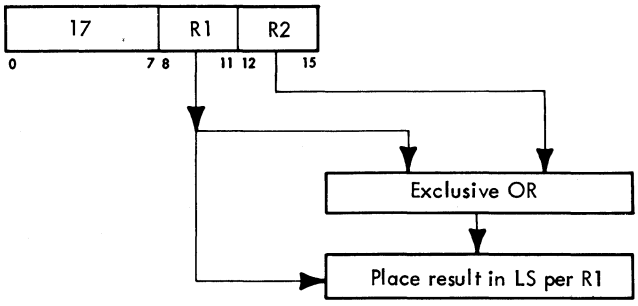
Bit position	0 1 2 3 4 5 6 7
1st operand	1 0 1 0 1 0 1 0
2nd operand	1 0 0 1 1 1 0 0
Result	0 0 1 1 0 1 1 0

Note that in bit positions 2, 3, 5, and 6 one and only one of the operand bits is a 1, and that the corresponding bit positions of the result are set to 1. In bit position 0, both operand bits are 1 and the corresponding result bit is 0. In bit position 1, both bits are 0 and the result is 0.

The Exclusive OR operation may be executed by an instruction in the RR, RX, SI, or SS format. The individual instructions are described in the following paragraphs.

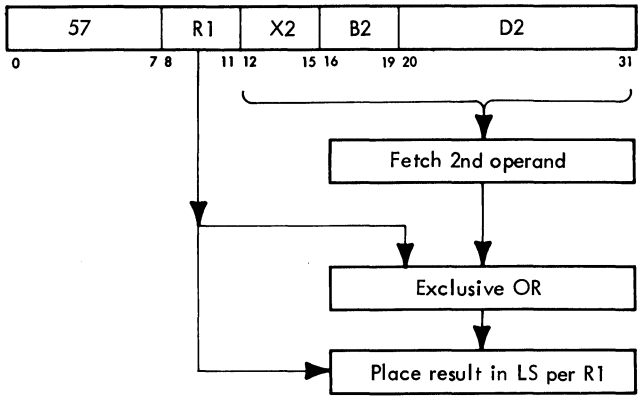
3.10.7.1 Exclusive OR, XR (17)

- Contents of LS register specified by R1 are Exclusive OR'ed with contents of LS register specified by R2. Result is placed in LS per R1 address.
- RR format:



3.10.7.2 Exclusive OR, X (57)

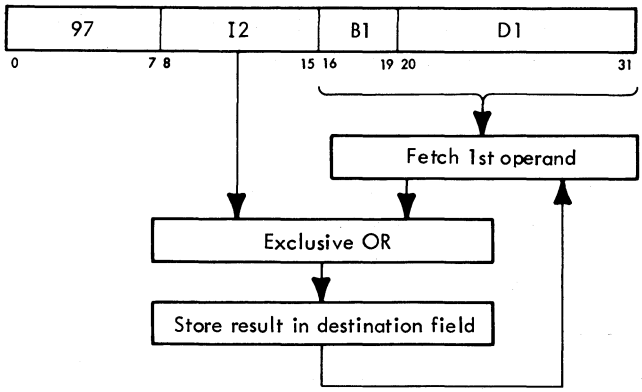
- Contents of LS register specified by R1 are Exclusive OR'ed with 2nd operand (obtained from main storage). Result is placed in LS per R1 address.
- RX format:



- Interruptions:
  Addressing.
  Specification.

3.10.7.3 Exclusive OR, XI (97)

- 1st operand is fetched from main storage and Exclusive OR'ed with immediate operand in I2 field. Result is stored at 1st operand address.
- SI format:

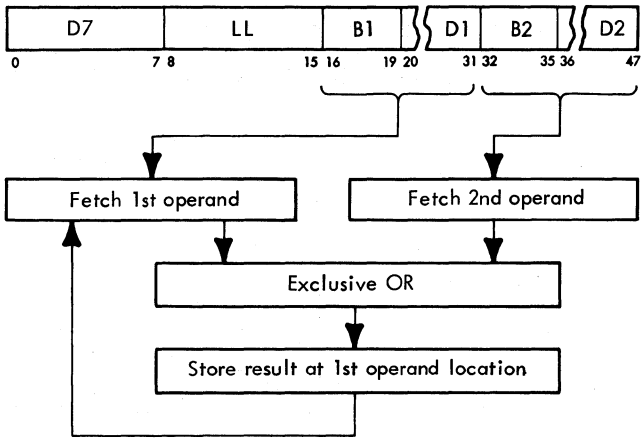


- Interruptions:
  Protection.
  Addressing.

3.10.7.4 Exclusive OR, XC (D7)

- Exclusive OR sum of 1st and 2nd operands is stored at 1st operand address.

- SS format:



- Interruptions:
  Protection.
  Addressing.

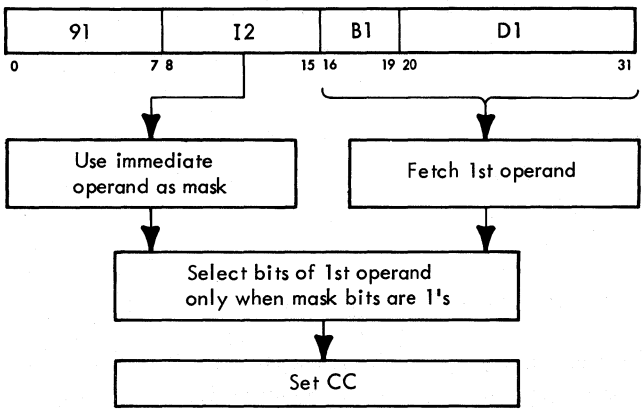
The XC instruction specifies the following actions:

1. The selected AB byte and the selected ST byte are gated to the serial adder, where they are combined per the serial adder Exclusive OR function.
2. The adder output is gated back to the selected ST byte, and the selected mark trigger is set per STC.
3. STAT A is set if the result is not zero.

Except for the above actions, the byte processing sequence is the same as that for the MVN instruction described in paragraph 3.10.2.

3.10.8 TEST UNDER MASK, TM (91)

- State of 1st operand bits selected by mask is used to set CC.
- SI format:



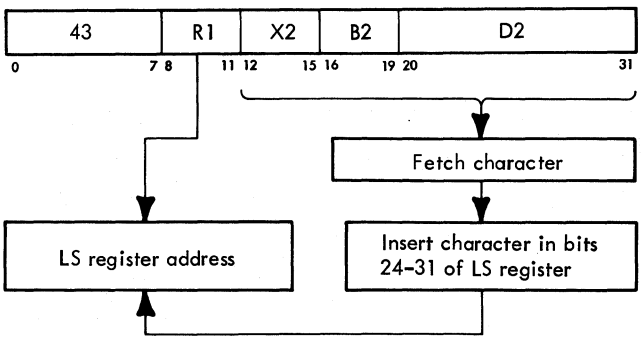
- CC setting:
  Selected bits all zero; mask is all zero – CC = 0.
  Selected bits mixed zero and 1 – CC = 1.
  Selected bits all 1 – CC = 3.
- Storage contents are not changed.
- Addressing interruption can occur.

The byte of immediate data, I2, is used as an 8-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in main storage specified by the first operand address.

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

3.10.9 INSERT CHARACTER, IC (43)

- 8-bit character at 2nd operand address is inserted into bit positions 24-31 of LS register specified by R1. Remaining bits of register are not changed.
- RX format:

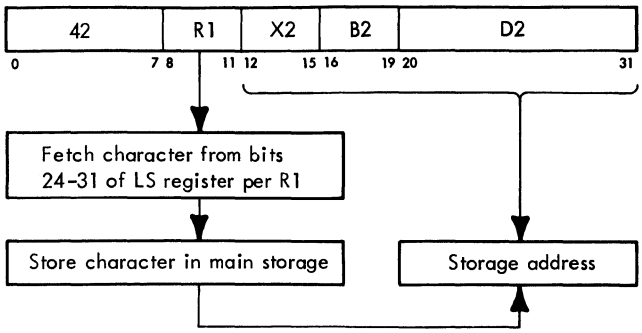


- Addressing interruption can occur.

3.10.10 STORE CHARACTER, STC (42)

- Bit positions 24-31 of LS register designated by R1 are stored at 2nd operand address.

- RX format:

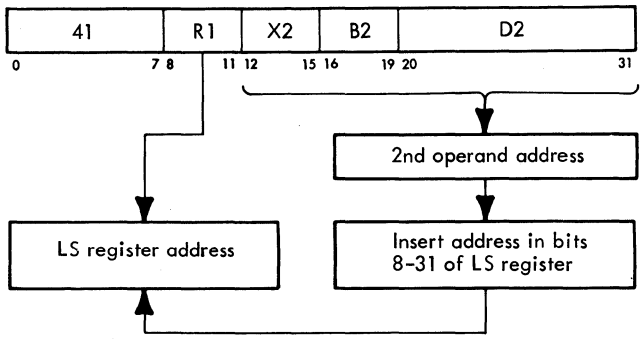


- Interruptions:
  Protecting.
  Addressing.

3.10.11 LOAD ADDRESS, LA (41)

- Address of 2nd operand is inserted in low-order 24 bits of LS register specified by R1. High-order 8 bits of LS register are made zero. 2nd operand is not fetched from main storage.

- RX format:



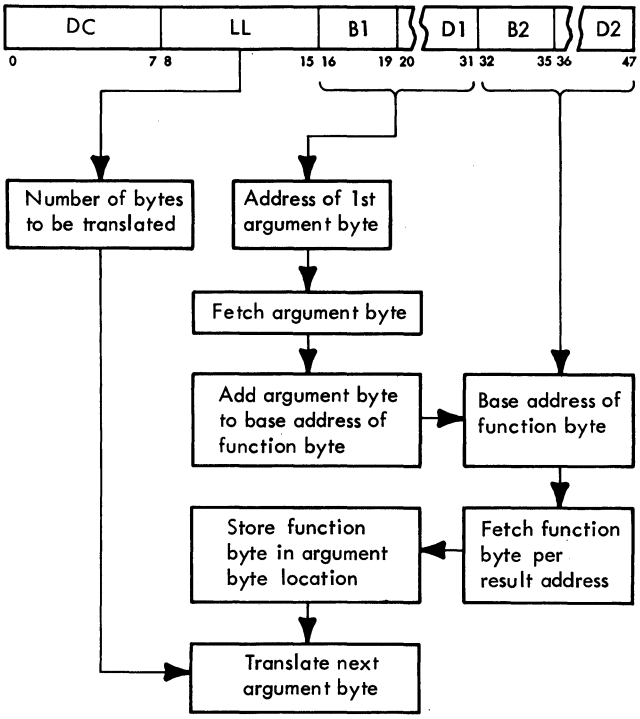
The address specified by the X2, B2, and D2 fields is inserted in bits 8-31 of the LS register. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 24th bit are ignored. The same LS register may be specified by the R1, X2, and B2 instruction field, except that LS register 0 can be specified only by the R1 field. In this manner, it is possible to increment the low-order 24 bits of an LS register, other than 0, by the contents of the D2 field of the instruction. The register to be incremented should be specified by R1 and by either X2 (with B2 set to zero) or B2 (with X2 set to zero).

3.10.12 TRANSLATE, TR (DC)

- 8-bit bytes of 1st operand are used as arguments to reference table of function bytes designated by 2nd operand address. Each function byte selected from table replaces corresponding argument byte in destination field.

- SS format:



- Interruptions:
  Protection.
  Addressing.

The TR instruction selects the first operand bytes for translation one byte at a time, proceeding from left to right. Each argument byte is added to the entire initial address, the second operand address, in the low-order bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte. All data is valid. The operation proceeds until the first operand field is exhausted. The table is not altered unless an overlap occurs.

At the start of the execution sequence, the first operand has been fetched to ST. A request per IC has been made for the second operand, but this double word from main storage is not used.

The execution sequence is as follows:

1. The selected ST byte is saved in F. The contents of T are saved in B. (The contents of the IC are saved in A.)

2. T is cleared, STC is set to 111, and the contents of F (selected destination byte) are placed in T(56-63) via the serial adder.

3. The contents of T are added to the contents of IC in the parallel adder, and the result is gated back to the IC.

4. A request for the second operand is issued per the IC.

5. ABC is set per IC(21-23), and STC is set per D(21-23).

6. The original source address is restored to the IC from A. The destination word is restored to T from B.

7. A word-overlap test was made prior to restoring the source address to the IC. If no word overlap exists, the table double word fetched from main storage is gated to AB. If word overlap is detected, the contents of S are transferred to A (B is already identical with T) and the double word from main storage is not used.

8. The selected AB byte is gated via the serial adder to the selected ST byte, and the selected mark trigger is set. The STC and D are incremented by 1. The LL count is decremented by 1.

9. Unless STC was 7 or LL was zero prior to stepping, the sequence is repeated for the next destination byte.

10. If LL was zero, the contents of ST are stored and the common end-op sequence is started.

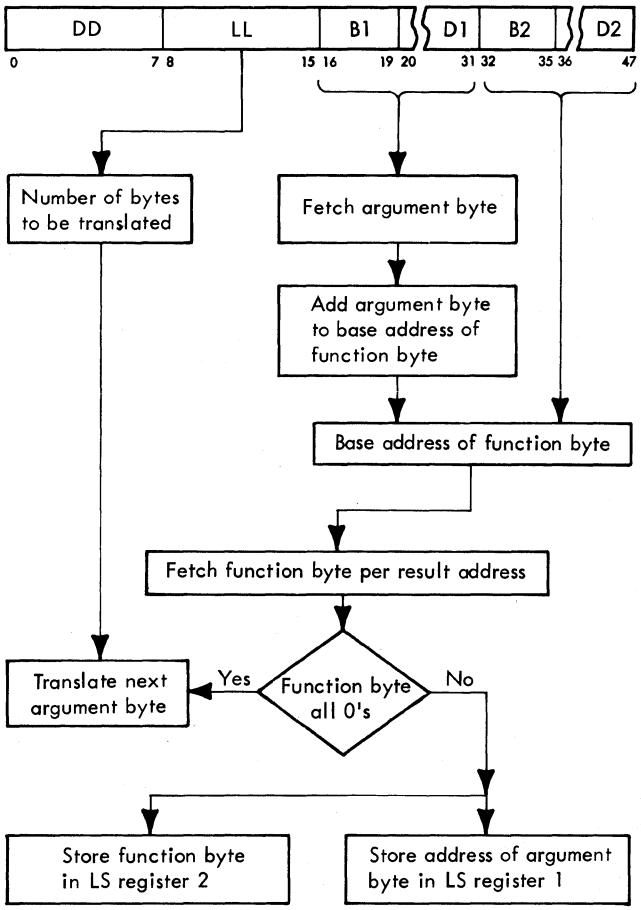
11. If STC was 7 and LL not equal to zero, the contents of ST are stored and the next destination word is fetched by the common destination fetch sequence, after which the translate sequence is resumed.

3.10.13 TRANSLATE AND TEST, TRT (DD)

- 8-bit bytes of 1st operand are used as arguments to reference table of function bytes at 2nd operand address.

  Each function byte thus selected from table determines continuation of operation. When function byte is a zero, operation proceeds by fetching and translating next argument byte. When function byte is nonzero, operation is completed by inserting related argument address in LS register 1 and function byte in LS register 2.

- SS format:



- CC setting:
  All function bytes are zero: CC = 0.
  Nonzero function byte before operand is exhausted: CC = 1.
  Last function byte is nonzero: CC = 2.

- Addressing interruption may result.

The TRT instruction fetches the function bytes in the same manner as the TR instruction (paragraph 3.10.12). Each function byte retrieved from the table is inspected for the all-zero combination.

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

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

The following abbreviations are used in this discussion of the TRT execution sequence:

DX: first byte in series of destination bytes

T(DX): table byte specified by DX

DX + 1: second byte in series of destination bytes

T(DX + 1): table byte specified by DX + 1

DX + 2: third byte in series of destination bytes

The TRT instruction uses the following execution sequence:

1. First Byte Sequence

   a. The selected ST byte is saved in F.

   b. The contents of ST are transferred to AB.

   c. STC is set to 3, and the contents of F (DX) are gated, via the serial adder, to byte 3 in S.

   d. S bytes 0, 1, and 2 are cleared by gating the contents of SAL to ST and successively decrementing STC by 1.

   e. ABC is set per D(21-23), and STC is set to 011.

   f. The DX in S is added to the contents of the IC, and an IC request is made for T(DX).

   g. A branch per STAT G is made to the T(DX + 1) address generation routine. (STAT G is used to indicate that a table byte has been fetched and is ready for test.)

2. T(DX + 1) Address Generation

   a. ABC is incremented by 1.

   b. DX is transferred from S to T.

   c. STAT G is set.

   d. The selected AB byte (DX + 1) is gated via the serial adder to byte 3 in S.

   e. STC is set per IC(21-23).

   f. The T(DX) ingate and T(DX + 1) fetch sequence is started.

3. T(DX) Ingate and T(DX + 1) Fetch Sequence

   a. The table word which contains byte T(DX) is available from main storage, and either the left- or right-half word is gated to T as determined by IC(21). STC(0) is set to 1 to select correct T byte. Simultaneously, the contents of T (DX) are subtracted from the contents of IC to restore the table base address.

   b. If LL equals zero, an exit is made to the T(DX) test sequence.

   c. If LL is not zero, DX + 1 (in S) is added to the contents of IC and a fetch request is made for T(DX + 1).

   d. A branch per STAT G starts the T(DX) test sequence.

4. T(DX) Test Sequence and T(DX + 2) Address Generation

   a. The selected T byte, T(DX), is gated to the serial adder for zero detection and is saved in F.

   b. STAT H is set if ABC equals zero.

   c. DX + 1 is transferred from S to T.

   d. STC is set to 011, and ABC is incremented by 1 (selecting byte to DX + 2).

   e. An exit is made to the LS mark sequence (see step 6) if a nonzero result is detected in the serial adder.

   f. An exit is made to the common end-op sequence if the serial adder result is zero and the LL count is zero.

g. The LL count is decremented and address in D is incremented by 1.

h. If STAT H is set, an exit is made to the destination fetch routine.

i. If no exit conditions are detected, the selected AB byte (DX + 2) is gated via the serial adder to S, and STC is set per IC(21-23).

j. The T(DX) ingating and T(DX + 1) fetch routine is started. The table byte previously referred to as T(DX) has been tested. The table byte previously referred to as T(DX + 1) is now considered T(DX), and processing loop is resumed.

5. Destination Fetch Routine

a. Prior to entering this routine, ABC has been stepped from 7 to 0 and a fetch request was made for a table byte using byte 0 of the present destination word to generate the table byte address. Since this was an erroneous address, the resulting word from main storage is not used.

b. STAT's G and H are reset, and the IC is restored to the table base address by subtracting DX + 1.

c. A fetch request is made per D address. The requested double word is gated to AB.

d. ABC was previously stepped from 0 to 1. It is now decremented to select byte zero of the new destination double word (considered byte DX).

e. The selected AB byte (DX) is gated via the serial adder to byte 3 in S.

f. The DX (in S) is added to the contents of IC, and a fetch request is made for T(DX).

g. Since STAT G is reset, the T(DX + 1) address generation routine is started.

6. LS Mark Routine

a. This routine is entered when a nonzero table byte is detected in the serial adder or when the LL count equals zero. (The last table byte tested is in F.)

b. If the table byte was nonzero, STAT G is reset.

c. E(8-15) is cleared and used for LAR addressing.

d. LS register 1 is accessed per E(8-15) + 1 and transferred to T.

e. STC is incremented to 100, and ST byte 4 is gated via the serial adder and back to ST. Simultaneously, the contents of D are gated to T via the parallel adder.

f. The contents of T are stored in LS register 1; E(8-11) is incremented twice, and STC is set to 7.

g. LS register 2 is read into T. The contents of F are gated via the serial adder to T(56-63), and the contents of T are stored in LS register 2.

h. STAT A is set if the byte in F was not zero.

i. The common end-op sequence is started, which sets CC per STAT's A and G.

3.10.14 EDIT AND EDIT AND MARK INSTRUCTIONS, ED AND EDMK (DE AND DF)

- Edit operation changes format of source (2nd operand) from packed to zoned. Source bytes are then edited under control of pattern (1st operand). Edited result is stored at 1st operand address.

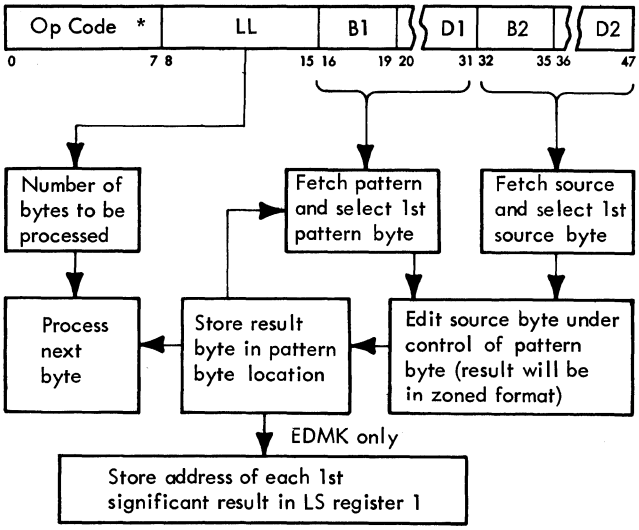
- Edit and mark operation is similar to edit except that address of 1st significant result digit is recorded in LS register 1.

- Edit and Edit and Mark instructions share common ROS microprogram with exit to separate mark routine for Edit and Mark instruction.

- SS format (see format on following page)

- Since results of word-overlap condition are unpredictable, no special action is taken when this condition occurs.

- CC setting:
  Result is zero: CC = 0
  Result is less than zero: CC = 1
  Result is greater than zero: CC = 2



* DE for Edit
DF for Edit and Mark

- Interruptions:
  Protection.
  Addressing.
  Invalid data.

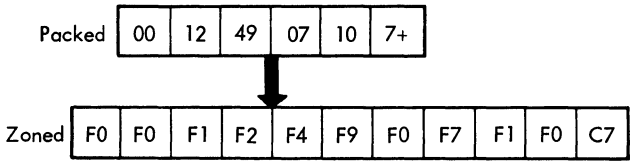
3.10.14.1 Introduction to Edit Operation

- Edit instruction is used to:
  Eliminate high-order zeros.
  Provide asterisk protection.
  Handle sign control (CR).
  Provide punctuation.
  Blank out an all-zero field.
  Protect decimal point by use of significance start character. (This character can also be used to retain high-order zeros when desired.)
  Edit multiple adjacent fields via field separator character.

The edit operation is used to produce easy-to-read documents by inserting proper punctuation into a data record. The data to be edited (2nd operand) is called the "source" and must be in the packed decimal format. Consider the following source field:

00	12	49	07	10	7+

For the above field to be printed in a document, it must first be converted into the zoned format (ASCII-8 or EBCDIC). One function of the edit operation is to change the source field from packed to zoned format.* If changing from packed to zoned format were all that was necessary to produce a legible report, the Edit instruction would not be necessary, since the Unpack instruction (paragraph 3.8.6) would be sufficient. For instance, if the above packed decimal operand were changed to the EBCDIC zoned format, it would look like this:



If the above zoned decimal field were printed, it would look like this:

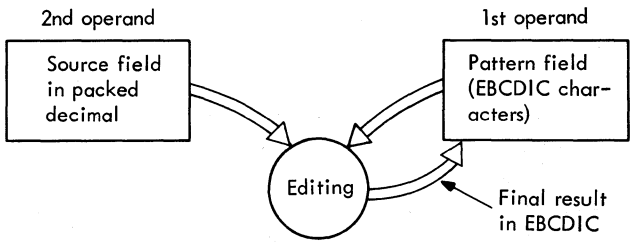
0 0 1 2 4 9 0 7 1 0 7 +

By examining the printed document, one could tell that it was a positive number with a low-order digit of 7. However, the printed document is still not too legible. If, for instance, the number represents money, it would be desirable to obtain the following printed result:

$1,249,071.07

This would require insertion of the commas and decimal points in the right place, as well as other editing. This is the main function of the edit operations.

The edit operation involves moving the source field (2nd operand) into the pattern field (1st operand). The pattern field is initially made up of EBCDIC characters that control the editing. The final edited result replaces the pattern field:



* Each time the digit from the source field replaces a digit select character, the 4-bit digit has the proper EBCDIC or ASCII-8 zone bits inserted. PSW(12) determines whether the EBCDIC or ASCII-8 zone is inserted. For the purposes of this discussion, it is assumed that the system is in EBCDIC mode.

As a rule, the second operand is shorter than the first because one source byte yields two result bytes.

The characters in the pattern field determine the editing that takes place. The high-order (leftmost) character in the pattern field is known as the "fill" character. Any of the 256 possible EBCDIC combinations can be used as the fill character. In many edit operations, however, the fill character consists of an EBCDIC blank (01000000). The blank character (represented by "b" in the discussion that follows) is not printed out and facilitates programmed blanking of high-order zero fields.

Besides the fill character, three more control characters in the pattern field have special meaning:

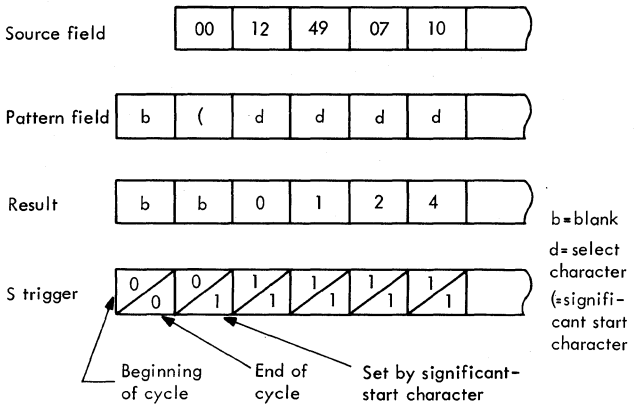
1. The digit select character.
2. The significant start character.
3. The field separator character.

The above characters can appear anywhere in the pattern field.

For purposes of discussion, the digit select character is represented by "d." (The binary code for the digit select character is 00100000, or a hexadecimal 20.) When a digit select character is encountered in a pattern field, it is usually replaced with a digit from the source field. If the digit in the source field is a high-order zero, however, the digit select character is replaced by the fill character. By using a blank as the fill character, high-order zeros can be blanked out. If an asterisk is used as the fill character, asterisk protection for paychecks can be achieved.

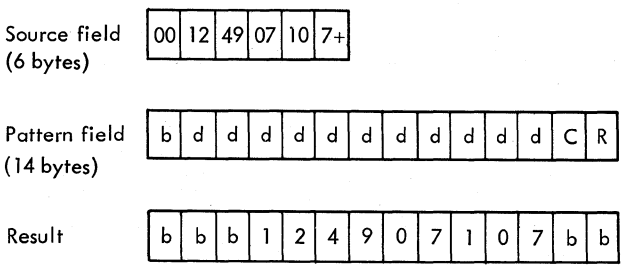
Since the digit select character may be replaced by either a source digit or the fill character, the system needs some way of knowing which of the two to choose. This function is provided by a special control trigger, known as the "S trigger". When the S trigger is set, it indicates that significant source digits are being processed. Consequently, the digit select characters in the pattern field are replaced with the digits from the source field. At the beginning of the edit operation, the S trigger is always reset. As long as the S trigger is reset, the digit select characters in the pattern field are replaced with the fill character.

As stated previously, the S trigger is set when a nonzero digit is detected in the source field. The S trigger is also set if a significant start character is detected in the pattern field. The significant start character has a bit code of 00100001 (hexadecimal 21). In this discussion, the symbol for the left parenthesis, (, is used to represent the significant start character. When a significant start character is detected in the pattern field, it is replaced by either a digit from the source field or the fill character. A typical edit operation using the b, d, and (characters is illustrated and explained below.



The edit operation begins by examining the fill character (which is b in the above case). If it is not a digit select or a significant start character, it is left in place in the pattern field. Then, the next pattern character is examined. Since this is a significant start character, the next high-order source digit is examined. Because this source digit is zero and the S trigger is reset (at this time), the significant start character is replaced with the fill character. However, the significant start character sets the S trigger so that all subsequent source digits are significant. The remaining pattern characters in the above example are digit select characters, which are replaced with source digits.

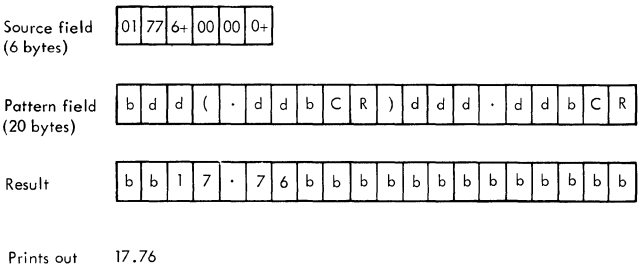
Once significance is started, the S trigger remains set until the sign of the source operand is examined. If a plus sign is detected, the S trigger is reset; if the source has a negative sign, the S trigger remains set because the usual method of indicating a negative quantity in a printed report is with the letters "CR". The following example illustrates how the state of the S trigger will identify the number as a positive or negative quantity:



When a pattern character is not one of the three special control characters and the S trigger is set, the character is not changed. If the S trigger is reset, the character is replaced by the fill character. Since detection of a positive sign resets the S trigger, the remaining pattern characters (CR) are replaced by the fill character. If the sign of the source field had been minus, the S trigger would have remained set and characters CR would have been left in the pattern field.

As stated previously, the S trigger is reset when a minus sign is detected in the source field. The S trigger is also reset if a field separator character is detected in the pattern field. The field separator character has a bit code of 00100010 (hexadecimal 22). In this discussion, the symbol for the right parenthesis,), represents the field separator character.

The field separator character is used when two or more packed decimal source fields are to be edited with one instruction into a single pattern field. The following edit example illustrates the use of the field separator character.



Note that after the field separator character resets the S trigger, the source field does not contain any significant digits. As a result, the pattern characters are replaced by the fill character (blank).

3.10.14.2 Introduction to Edit and Mark Operation

The operation is identical with the Edit instruction, except for the additional function of inserting a byte address in LS register 1. The byte address is inserted in bits 8-31 of this register. The byte address is inserted each time the S trigger is in the zero state and a nonzero digit is inserted in the result field. The address is not inserted when significance is forced by the significant start character of the pattern. Bits 0-7 are not changed. The Edit and Mark instruction facilitates the programming of floating currency-symbol insertion. The character address inserted in LS register 1 is 1 more than the address where a floating currency-sign would be inserted. (The Branch on Count instruction, with zero in the R2 field, may be used to reduce the inserted address by 1.)

The character address is not stored when significance is forced. Therefore, the address of the character following the significant start character should be placed in LS register 1 prior to the Edit and Mark instruction.

When a single instruction is used to edit several numbers, the address of the first significant digit of each number is inserted in LS register 1. Only the last address will be available after the instruction is completed.

3.10.14.3 General Data Handling

Special circuits are packaged in the serial adder for use in the Edit and Edit and Mark instructions. These circuits consist of:

1. A Decoder of Serial Adder Bus B (SBB) to detect a digit select, significant start, or field separator character in the selected ST byte.
2. Right-digit trigger for AB digit selection.
3. Controls for stepping of ABC.
4. Controls for determining which data (i.e., ST byte, F, or AB digit with zone) is to be used as the result byte, and controls for gating this data to the serial adder.
5. Zero detection of the selected AB digit.
6. Sign detection of the low-order digit of the selected AB byte.
7. Detection of a mark condition.
8. The S trigger with associated set-reset controls.
9. Controls for setting or resetting STAT's.

The destination field is considered a pattern field and is processed one byte at a time, from left to right, under control of STC. Each ST byte is gated to SBB for decoding and will be replaced by a byte of data which, depending on decoded conditions, may be:

1. Original data of ST byte.
2. A selected digit of AB with a zone inserted in the high-order four bits.

3. A fill character, which is contained in F.

The source field is processed, one digit at a time, from left to right, under control of ABC and S trigger, which selects which digit of a byte is to be used. The selected AB digit is examined only if a digit select or significant start character appears in the selected ST byte. The selected AB digit is not necessarily used as part of the result byte, but the next digit to be processed is selected after the digit has been examined.

3.10.14.4 Detailed Microprogram Description

The flow chart for the Edit and Edit and Mark microprogram is shown in Figure 6065, FEDM. At the start of the execution sequence, the fill character is gated from ST (per STC) through the serial adder to F. A 2-cycle data-processing sequence is then started and is repeated until all destination operand bytes have been processed. Exits from this sequence are made when required for operand fetching or marking, after which this sequence is continued. The microprogram is explained in three parts: (1) first cycle, (2) second cycle, and (3) exit conditions.

3.10.14.4.1 First Cycle

This is a decode cycle; no data is transferred. The selected ST byte is gated to SBB, and the selected AB byte is gated to serial adder bus A (SBA) with the digit to be examined determined by the right-digit trigger. The decode circuits are activated by ROS. Decoding of SBB, SBA, and the S trigger governs the selection of appropriate inputs to the serial adder, and also whether the S trigger is set or reset. STAT A is set if the selected source digit (in AB) is a nonzero digit. However, if a field separator character is decoded at SBB, STAT A is reset.

STAT E is set if an invalid digit is decoded in SBA(0-3). A 1 is added to D (except for the first entry from another sequence) to keep the byte address in D at the same value as STC for use in the marking sequence. A mark condition is detected and latched for a branch condition of the Edit and Mark instruction.

3.10.14.4.2 Second Cycle

At the start of this cycle, data is gated to the serial adder by hardware controls as explained in the first cycle. The second cycle performs the following control functions:

1. The serial adder output is gated back to the selected ST byte, and the appropriate mark trigger is set.

2. STC is incremented, and the LL count in E(8-15) is decremented by ROS control.

3. ABC is incremented by hardware controls.

4. If required, the digit-select trigger is complemented. This action is conditional on the following:

   a. The digit selection of AB is changed only if a significant start or a digit select was decoded during the first cycle.

   b. When a sign code is decoded in SBA(4-7) at the time bits 0-3 are selected for examination, the low-order digit (sign) will be skipped by stepping ABC and leaving the right-digit trigger reset.

5. If required, exit to a separate routine is made via an 8-way ROS branch, for end-op, operand fetching, or marking. If no exit conditions exist, the execution sequence is repeated.

3.10.14.4.3 Exit Conditions

Exits from the data processing sequence are made when one or more of the following conditions exist:

1. Edit and Mark instruction is being executed, and a mark condition is detected.

2. LL = 0 or STC = 7.

3. ABC = 7.

Where more than one of the above conditions exists, a branch is made to the proper sequence in the order they are listed above. An explanation of each sequence is given below.

1. Exit on Detection of Mark Condition

   Exit to the mark sequence is made regardless of other branch conditions. Special action is taken to return counter values to what they were prior to entering the mark sequence, so they can be retested. STAT H is set if ABC

has just stepped from 7 to 0, to record this condition. The contents of AB are destroyed by gating E(8-15) + 1 via the parallel adder to A, and the contents of T via the parallel adder to B. E(8-15) is cleared, and LS register 1 is read to T using E(12-15) + 1 as the LAR address. The contents of D, the byte address of the last byte processed, are placed in T(40-63). T(32-39) is retained by gating it through the serial adder and back to T at the same time the D-PAL-T transfer occurs. The contents of T are now written back to LS register 1. Registers and counters are restored to their original contents. The source operand is replaced in AB by refetching it from main storage, and a test is made, via an ROS branch, for any other exit condition which may have been present at the time mark sequence was started. If no other exit condition exists, the data-processing sequence is resumed.

2. Exit on LL = 0 or STC = 7 (End-Op or Destination Fetch)

   STAT D is set if ABC also equals 7, and a destination store is started and a check is made for invalid data. If STAT E has been set, a data-interruption trigger is set and an end-op sequence is started. If STAT E is not set, a test is made for end-op condition via an ROS branch. If the LL count has been stepped to all 1's, and end-op sequence is started which sets the CC, restores the instruction address to the IC, and resets STAT G.

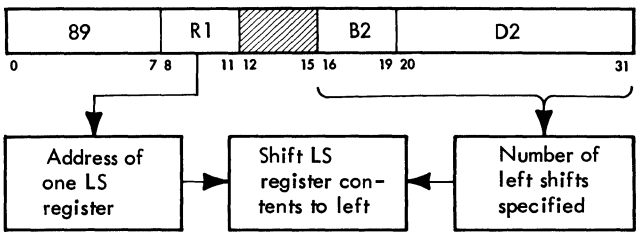
   If an end-op condition does not exist, D is incremented and a fetch request is initiated for the next double word of destination operand. A test is made to see whether a source fetch is also required (ABC = 0 and STAT D set). If not, the data-processing sequence is resumed.

3. Exit on ABC = 7 (Possible Source Fetch)

   A further test must be made to determine whether the last byte of AB has been completely processed. This is determined by testing ABC for an all-zero count (i.e., ABC was stepped from 7 to 0 in the previous cycle). If ABC is not zero, the data-processing routine is restarted; otherwise, the IC is incremented by 8 and a fetch is initiated for the next double word of source operand. This source fetch sequence is common to all VFL logical instructions and incorporates the word-overlap test. However, this test does not affect the edit operation. The source double word from main storage is gated from SDBO to AB, and the data-processing sequence is resumed.

3.10.15 SHIFT LEFT SINGLE, SLL (89)

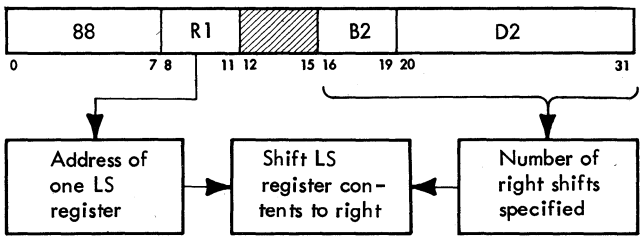
- 1st operand is shifted left number of bits specified by 2nd operand address.
- RS format:



- 2nd operand address is not used to address data; its low-order 6 bits indicate number of bit positions to be shifted. Remainder of address is ignored.
- All 32 bits of LS register specified by R1 participate in shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated low-order register positions.
- SLL instruction shares same microprogram as SLA instruction (see paragraph 3.4.9.1).

3.10.16 SHIFT RIGHT SINGLE, SRL (88)

- 1st operand is shifted right number of bits specified by 2nd operand address.
- RS format:



- 2nd operand address is not used to address data; its low-order 6 bits

indicate number of bit positions to be shifted. Remainder of address is ignored.

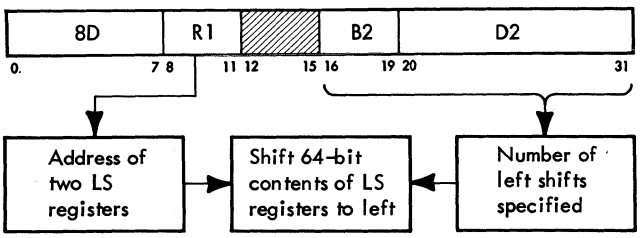
- All 32 bits of LS register specified by R1 participate in shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to vacated high-order register positions.

- SRL instruction shares same microprogram as SRA instruction (see paragraph 3.4.10.1).

3.10.17 SHIFT LEFT DOUBLE, SLDL (8D)

- Double-length 1st operand is shifted left number of bits specified by 2nd operand address.

- RS format:



- R1 field of instruction specifies an even/odd pair of registers and must contain an even register address. An odd value for R1 is a specification exception and causes a program interruption. 2nd operand address is not used to address data; its low-order 6 bits indicate number of bit positions to be shifted. Remainder of address is ignored.

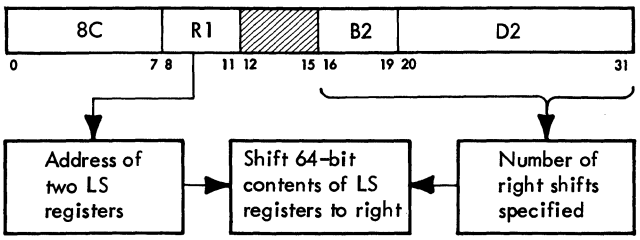
- All 64 bits of even/odd register pair specified by R1 participate in shift. High-order bits are shifted out of even-numbered register without inspection and are lost. Zeros are supplied to vacated low-order positions of odd-numbered registers.

- SLDL instruction shares same microprogram as SLDA instruction (see paragraph 3.4.9.2).

3.10.18 SHIFT RIGHT DOUBLE, SRDL (8C)

- Double-length 1st operand is shifted right number of bits specified by 2nd operand address.

- RS format:



- R1 field of instruction specifies an even/odd pair of registers and must contain an even register address. An odd value for R1 is a specification exception and causes a program interruption. 2nd operand address is not used to address data; its low-order 6 bits indicate number of bit positions to be shifted. Remainder of address is ignored.

- All 64 bits of even/odd register pair specified by R1 participate in shift. Low-order bits are shifted out of odd-numbered register without inspection and are lost. Zeros are supplied to vacated high-order positions of the registers.

- SRDL instruction shares same microprogram as SRDA instruction (see paragraph 3.4.10.2).

SECTION 6. BRANCHING INSTRUCTIONS

This section presents a functional analysis of the branching instructions, including a discussion of branching, instruction format, data flow, and program interruptions. Since the data used and generated during a branch instruction may depend upon or be used in the end-op cycle or the I-Fetch sequence, these operations should be thoroughly understood before studying the branch instructions. (Refer to Section 1 of this chapter.)

3.11 INTRODUCTION

- Branching causes departure from normal address sequencing.
- Branch address introduced as next sequential address.
- Branch address obtained from general-purpose LS register or specified as 2nd operand address.
- Branch may be conditional or unconditional.
- Conditional branches:
  - Branch on Condition
  - Branch on Count
  - Branch on Index
- Unconditional branches:
  - Branch and Link
  - Execute
- Conditional branches may or may not use branch address.
- Unconditional branches always use branch address.
- If branch is successful, storage request per IC issued during I-Fetch is blocked.
- If branch is unsuccessful, Q is refilled if required.

Normally, the CPU is controlled by instructions taken in sequential order. That is, an instruction is fetched from a main storage location specified by the instruction address in the IC. The address is then increased by the number of bytes needed to address the next instruction in sequence, and this updated address replaces the old address in the IC. The current instruction is executed, and the same steps are repeated using the updated instruction address to fetch the next instruction.

A departure from the normal instruction sequence occurs when branching is performed. A branch address is introduced as the next instruction address. This branch address may be obtained from one of the general-purpose registers in the LS, or it may be the second operand address specified by a particular instruction. Depending upon the format and the instruction, branching may be either conditional or unconditional. The conditional branches are (1) branch on condition, (2) branch on count, and (3) branch on index. The unconditional branches are (1) branch and link and (2) execute. Conditional branches may or may not use the branch address. If the branch is successful (that is, the branch is taken), the branch address is used and the storage request issued per the IC during I-Fetch is blocked. If the branch is unsuccessful, the instruction address in the IC is used to fill Q. Unconditional branches are always taken and use the branch address.

Branching is used to reference a subroutine, to resolve a 2-way choice, or to repeat a portion of a program. To save time and increase the speed of the operating program, branching is always considered to be successful unless proven otherwise. Therefore, whenever a branch instruction is decoded during I-Fetch, the next instruction address is the branch address located in D. If the branch is found to be unsuccessful (determined during execution of the branch instruction), the instruction address from D is ignored, and the correct instruction address is obtained from the IC.

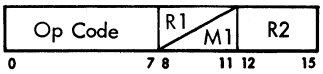
There are two methods of performing an end-op cycle in the branch operations: (1) normal end op and (2) branch end op. The normal end-op cycle allows decoding of the next instruction format from R and of the instruction address from the IC, and is normally used when ending an operation. Decoding off R is possible since the data placed in the register has become stable by the time the end-op cycle begins. The branch end-op cycle, on the other hand, allows decoding of the next instruction format from the SDBO and of the instruction address from D.

This end-op cycle is used when the data, which has been placed in R, is not yet stable and is some halfword other than the last halfword of Q. Decoding from the SDBO saves the time it takes for the data to stabilize in R and the instruction address to stabilize in the IC.

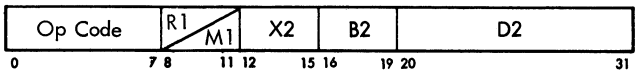
3.11.1 INSTRUCTION FORMAT

- Branch instructions use three formats:

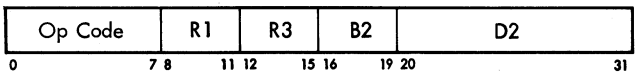
RR



RX



RS



- In RR format:
  R1 is LS address of 1st operand.
  M1 is mask field.
  R2 is LS register containing branch address.

- In RX format:
  R1 is LS address of 1st operand.
  M1 is mask field.
  Index + base + displacement = branch address.

- In RS format:
  R1 is LS address of 1st operand.
  R3 is LS address of increment value.
  Base + displacement = branch address.

In the formats shown above, bits 8-11 are normally the R1 field that specifies the address of an LS register containing the first operand. In the Branch on Condition instruction, however, bits 8-11 are designated as M1 and contain mask bits used in conjunction with PSW CC settings to determine whether the branch is successful.

In the RR format, the R2 field specifies the address of an LS register that contains the branch address except when R2 is zero, in which case no branching is to take place.

In the RX format, the contents of the LS registers specified by the X2 and B2 fields are added to the D2 field to form the branch address.

In the RS format, which is used in Branch on Index operations, the contents of the LS register specified by the B2 field are added to the D2 field to form the branch address. The R3 field specifies the address in LS of an increment value (third operand) which is added to the first operand to determine the index value.

3.11.2 DATA FLOW

- Main functional units used: Q, R, E, D, and IC.

- Secondary functional units used: T, AB, parallel adder, STC, and ABC.

Figure 9056, FEDM, is a diagram of the basic data flow for the branch instructions. The main functional units used to determine addresses and instructions in the branch operations are Q, R, E, D, and IC. The secondary functional units, T, AB, parallel adder, STC, and ABC, determine whether the branch is successful when the branch being executed is a conditional branch. The purpose of each functional unit is as follows:

1. Q: Holds the double word that contains the instruction addressed by the branch instruction if the branch is successful.

2. R: Contains the instruction to be performed after execution of the branch instruction.

3. E: Contains the branch instruction presently being executed.

4. D: Holds the address of the double word which, if the branch is successful, contains the next instruction to be executed.

5. IC: Holds the address of the double word which, if the branch is unsuccessful, contains the next instruction to be executed.

6. T: Buffers the operand being tested and operated on.

7. AB: Holds the first operand when added to some other value to determine whether the branch is successful.

8. Parallel adder: Determines whether conditions have been met when a conditional branch is being executed.

9. STC: Allows transfer of last byte of T during an Execute instruction when modifying the subject instruction of the Execute instruction.

10. ABC: Selects data being modified in the subject instruction during an Execute instruction.

3.11.3 INTERRUPTIONS

- Program interruptions:
  Execute
  Addressing
  Specification
  Protection

- Current PSW stored as old PSW and new PSW obtained.

- Interruption code in old PSW identifies cause of interruption.

Table 3-22 lists the interruptions that may occur in branching instructions.

TABLE 3-22. BRANCHING INSTRUCTION INTERRUPTIONS

Interruption	Interruption Code PSW Bits 16-31	ILC	How Instruction Execution is Finished
Execute	00000000 00000011	2	Suppressed
Addressing	00000000 00000101	0, 1, 2, 3	Suppressed
Specification	00000000 00000110	1, 2, 3	Suppressed
Protection	00000000 00000100	0, 2, 3	Suppressed

These interruptions cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following listing briefly describes the interruptions:

1. Execute: The subject instruction of an Execute instruction is another Execute instruction. The operation is suppressed.

2. Addressing: The branch address of an Execute instruction designates an instruction-halfword location outside the available storage area. The operation is suppressed.

3. Specification: The branch address of an Execute instruction is odd. The operation is suppressed.

4. Protection: The branch address of an Execute instruction is protected.

3.12 INSTRUCTION ANALYSIS

A list of the branching instructions with their format, mnemonic code, op-code, and interruptions is given in Table 3-23. A functional description of each instruction is contained in the paragraphs that follow.

TABLE 3-23. BRANCHING INSTRUCTIONS

Instruction	Format	Mnemonic Code	Op Code	Interruptions
Branch on Condition	RR	BCR	07	-
Branch on Condition	RX	BC	47	-
Branch and Link	RR	BALR	05	-
Branch and Link	RX	BAL	45	-
Branch on Count	RR	BCTR	06	-
Branch on Count	RX	BCT	46	-
Branch on Index High	RS	BXH	86	-
Branch on Index Low or Equal	RS	BXLE	87	-
Execute	RX	EX	44	Execute/ Addressing/ Specification/ Protection

3.12.1 BRANCH ON CONDITION

- Next sequential address replaced by branch address if state of CC is as specified by M1.

- Instructions:
  Branch on Condition, BCR (07)
  Branch on Condition, BC (47)

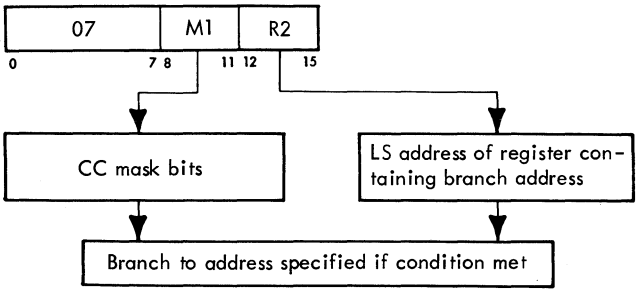
- M1 field used as 4-bit mask.

- Branch successful when mask bit for particular CC is 1.

- Branch unsuccessful if M1 field equals zero.
- Branch known to be successful or unsuccessful before execution of branch instruction.
- Storage request issued per IC or D, depending upon whether branch is successful and Q needs filling.

3.12.1.1 BCR (07)

- Replaces next sequential address with branch address if state of CC is as specified by M1.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  Branch address is in D.
- Branch is unsuccessful if R2 field equals zero.
- CC's and corresponding mask bits listed in Table 3-24.

The Branch on Condition (BCR) instruction, which has an RR format with an op code of 07, replaces the next sequential instruction address with the branch address located in the LS register specified by R2 if the CC agrees with the corresponding mask bit in the M1 field. The M1 field is used as a 4-bit mask. Table 3-24 lists the CC's and the corresponding mask bits in the M1 field. The branch instruction is successful whenever the CC has a corresponding mask bit of 1 and the R2 field is not zero. When all four mask bits are 0 or when the R2 field contains all 0's, the branch instruction is unsuccessful. The BCR is equivalent in this case to a no-operation instruction.

TABLE 3-24. CONDITION CODE MASK BITS

Condition Code	Mask Bit In (E(8-11)
0	8
1	9
2	10
3	11

At the start of execution, the instruction is contained in E, the branch address is contained in D, and a 3-cycle storage request for the branched-to instruction has been generated. Normally, a storage request is generated per the IC. For a BCR instruction, however, the storage request can be generated from two possible places, depending upon whether the branch is successful. If the branch is successful, the storage request is generated per D. If the branch is unsuccessful, the storage request is generated per the IC if Q needs to be refilled. Since the CC's, which have to be compared with the mask bits, are set in the execution phase of a previous instruction and are tested during I-Fetch of the branch instruction, success of the branch can be determined beforehand. Therefore, the BCR instruction knows whether it is successful or unsuccessful before actual execution of the instruction.

If the branch is successful, the branch address is placed in D by the normal I-Fetch sequence. A storage request is then issued to main storage per D. The correct halfword within the double word from main storage is then gated into Q and from Q to R per D(21, 22). The contents of D are updated by 8 and placed in the IC to address the next sequential instruction from main storage. If the branch is unsuccessful, the storage request is issued per the IC, if Q needs to be refilled, during I-Fetch, and the data from main storage is gated to Q during the execution of the branch.

A flow chart of the BCR operation is contained in Figure 6066, FEDM. Assume that the branch operation is a successful branch (Figure 6066, FEDM, point A). The execute and address-store-compare triggers are reset if they are set. The triggers are set if the branch instruction is the subject instruction of an Execute instruction.

The contents of D are now transferred to PAA(40-63). Then 8 is added to PAA, and the result (address of next double word to be operated on) is transferred

to the IC. D(21,22) is now tested. If D(21,22) equals 11, it signifies that the next instruction to be executed, when the data is gated into Q from the SDBO, occupies the last halfword of Q. If D(21,22) equals a value other than 11, then the next instruction to be executed is in some halfword other than the last halfword of Q.

Assume that D(21,22) equals 11. In this case, a storage request must be issued to obtain the next instruction to be executed. At this time, the data (branched-to instruction) that was requested during I-Fetch of the branch instruction is present at the SDBO and can be gated into Q. From Q, the data is gated to R per D(21,22), thus placing the last halfword of Q in R.

Normally, the instruction format is determined from R-register decoding. Since the branched-to instruction (located in the last halfword of Q) has just been placed in R, decoding cannot be performed on this cycle because the ingated data is not yet stable. Therefore, the branch instruction must perform a few operations to allow time for R to stabilize. The contents of the IC are then transferred to the parallel adder, where they are updated by 8 and replaced in the IC to address the next double word from main storage. After the IC has been updated, the next sequential double word (requested during execution of the branch instruction) is gated from the SDBO into Q. The data in R is stable at this time, and a normal end-op cycle can take place to complete the operation.

Now assume that D(21,22) equals a value other than 11 on a successful branch. This condition means that the next instruction to be executed is contained in either the first, second, or third halfword of Q when the data from storage is gated into Q. The data which was requested during I-Fetch of the branch instruction is now present at the SDBO and can be gated into Q. The halfword that contains the next instruction to be executed is then gated into R per D(21,22). As previously stated, format decoding is normally accomplished from R and instruction address decoding from the IC. Since the data to be decoded and the address of the next instruction have just been placed in R and the IC, it is not yet stable and therefore cannot be decoded in this cycle. Rather than delaying a few cycles until the information is stable, a branch end-op cycle is taken. This cycle allows decoding of the halfword (containing the next instruction) from the SDBO as the data is transferred from the SDBO to Q and decoding of the instruction address from D.

Assume, now, that the branch had been found to be unsuccessful (point B, Figure 6066, FEDM). Since the storage request was generated per the IC, IC(21,22) is now tested. At this point, IC(21,22) is either 11 or 00. First assume that IC(21,22) is equal to 00. This value means that the next instruction to be executed is to come from the first halfword of the double word which has been requested during I-Fetch. The contents of the IC are updated by 8 and placed in both the IC and D. The double word from main storage requested during I-Fetch is now gated from the SDBO to Q. The first halfword from Q is then transferred to R. Since the next instruction to be executed has just been placed in R and is not yet stable, the format cannot be decoded on this cycle. The instruction must therefore perform some operation to allow time for R to stabilize. The execute trigger is now tested. If reset, the branch instruction ends in a branch end op and format decoding of the next instruction takes place off the SDBO. If the execute trigger is set, the branch instruction was the subject instruction of an Execute instruction and the present contents of the IC are incorrect since the IC was increased by 8 during the Execute instruction. The contents of the IC are thus reduced by 8 and replaced in the IC. By this time, the data gated into R is stable and the format can be decoded from R. A normal end op, therefore, can take place to complete the operation.

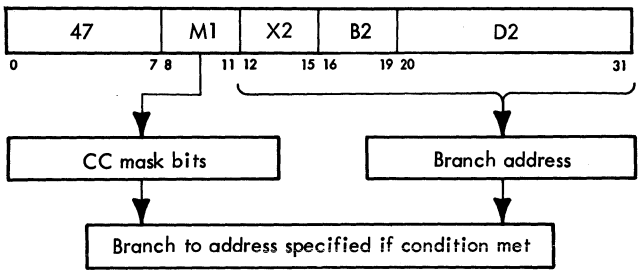
Now assume that IC(21,22) equals 11. This value means that the next instruction to be executed is located in the last halfword of the double word from main storage that contains the branch instruction unless the branch was the subject instruction of the Execute instruction. Then, the next instruction is located in the last halfword of the double word containing the Execute instruction. The contents of the IC are now transferred to PAB(40-63). Then 8 is added to PAB, and the result is transferred to the IC and D. At this time, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and can be gated into Q. The execute trigger is now tested. If the trigger is set, the IC is reduced by 8 and a normal end-op cycle is taken. If the execute trigger is reset, a branch end-op cycle is taken. The branch end-op cycle is taken because some of the data to be used in the next instruction may be in the double word requested during I-Fetch of the branch instruction. Since this data has just been placed in Q, it has not yet stabilized and cannot be decoded from Q. Therefore, the data is decoded from the SDBO during the branch end-op cycle.

Now assume that IC(21,22) contained either 01 or 10 and that the branch instruction was unsuccessful

(point C, Figure 6066, FEDM). In this case, the next instruction to be executed is in either the second or the third halfword of Q, and Q does not need to be refilled. Therefore, a normal end-op cycle is taken, the next instruction format is decoded from R, and the instruction address is decoded from the IC.

3.12.1.2 BC (47)

- Replaces next sequential address with branch address if state of CC is as specified by M1.
- RX format:



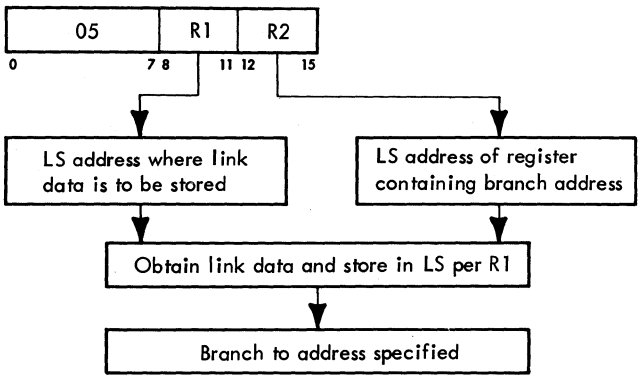
- Conditions at start of execution:
  First 16 bits of instruction are in E.
  Branch address is in D.
- Operation is identical with BCR. (Refer to paragraph 3.12.1.1.)
- Figure 6066, FEDM.

3.12.2 BRANCH AND LINK

- Stores link information in LS register specified by R1 and branches to address specified by 2nd operand.
- Instructions:
  Branch and Link, BALR (05)
  Branch and Link, BAL (45)
- Link information consists of:
  Instruction length code
  CC
  Program mask bits
  Address of next sequential instruction
- Link information stored whether branch is successful or unsuccessful.

3.12.2.1 BALR (05)

- Stores link information in LS register specified by R1 and branches to address specified by 2nd operand.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  2nd operand is in AB and D.
- If execute trigger is set and IC(21,22) equals 11, IC is reduced by 16 and stored as link address information.
- If execute trigger is set and IC(21,22) does not equal 11, IC is reduced by 8 and stored as link address information.
- Branch is unsuccessful if R2 equals zero.

The Branch and Link (BALR) instruction, which has an RR format with an op code of 05, stores the address of the instruction which, if the branch is unsuccessful, is the address of the next sequential instruction. Stored with the address is link information containing the instruction length code, the CC, and the program mask bits. The instruction length code stored will be either 1 or 2. If the instruction length code stored is 2, the BALR instruction is the subject of an Execute instruction. If during a BALR operation the R2 field is equal to zero, the branch is considered unsuccessful.

A flow chart of the BALR instruction is contained in Figure 6067, FEDM. At the start of execution, the instruction is contained in E, the second operand is in AB and D, and a storage request is issued per D for the branched-to instruction. The purpose of the BALR instruction is to branch to a subroutine and provide a means of returning from the subroutine

to the main flow of instructions in a program. How this is accomplished is shown in Figure 3-20. When processing the main instruction flow and a BALR instruction is encountered, the address of the double word containing the instruction which sequentially follows the BALR in the main instruction flow is stored in LS. For the example illustrated in Figure 3-20, the address of the double word containing the instruction is 8 and is stored in LS address 15. If the BALR instruction is in address 14 of the main instruction flow, then the address stored will be 16. Once the instruction address is stored, the branch to the subroutine occurs. The subroutine is performed and, when completed, a branch instruction is issued using the address that was stored during the BALR instruction as the branch address to return to the main flow of instructions. After returning to the main flow of instructions, the program will continue in its normal manner; that is, processing the remaining instructions.

In determining the address which is to be stored as the link address, IC(21,22) and the execute trigger must be tested (Figure 6067, FEDM). If IC(21, 22) equals 11 and the execute trigger is set (indicating the BALR is the subject of the Execute instruction and the Execute instruction is located in

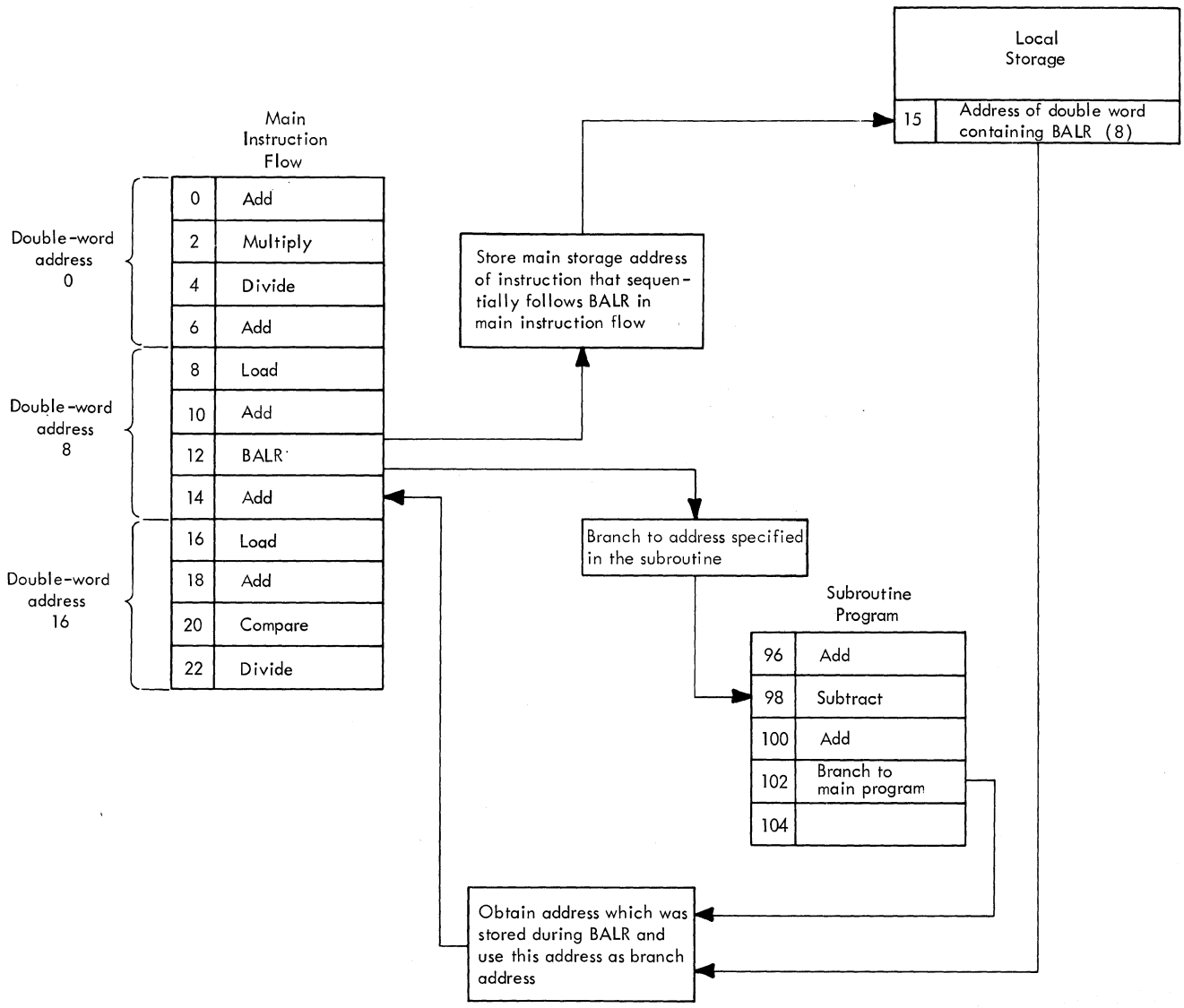


FIGURE 3-20. EXAMPLE OF USE OF BRANCH AND LINK INSTRUCTION

the second and third halfwords of its double word), 16 is subtracted from the IC and placed in T. Thus, if it is necessary to return to the main flow of instructions, the instruction which will be performed next is that instruction sequentially following the Execute instruction and is contained in the same double word as the Execute instruction. If IC(21,22) does not equal 11 or the execute trigger is reset, 8 is subtracted from the IC and the value is placed in T.

The contents of E(12-15), which contains the address of the LS register which has the branch address, are now examined. If E(12-15) equals zero, branching is not to take place and a No Operation occurs. If E(12-15) equals anything other than zero, branching occurs unconditionally. First assume that the branch is unsuccessful [E(12-15) equals zero]. The state of IC(21,22) is tested. If IC(21,22) equals 11, it indicates that the next instruction to be executed is in R (this instruction is the last halfword of the double word in Q) and a new double word must be placed in Q. If IC(21,22) equals any other value, the next sequential instruction is also located in R but Q contains data which is still good and may be operated on. In either case, the remainder of the link data [PSW(32-39)] is placed in T(32-39) and from there transferred to LS per E(8-11). If IC(21,22) equals a value other than 11, a normal end op is taken after storing the data and the next instruction is decoded from R. If IC(21,22) equals 11, after storing the link information, a 3-cycle storage request is issued per the IC to obtain the next sequential double word to refill Q. The execute trigger is again tested. If set, a normal end op is immediately taken and the next instruction to be executed is the instruction which sequentially follows the Execute instruction (of which the BALR instruction was the subject instruction). If the execute trigger is reset, the next instruction to be executed is in R. The IC is updated by 8 to select the next sequential double word after the one just requested. IC(21,22) is again tested; if it equals 11, the data on the SDBO is gated to Q and a branch end-op cycle is taken. If IC(21,22) equals some other value, the data on the SDBO is gated to Q and the first halfword of Q is gated to R. A branch end-op cycle is taken to allow decoding off the SDBO.

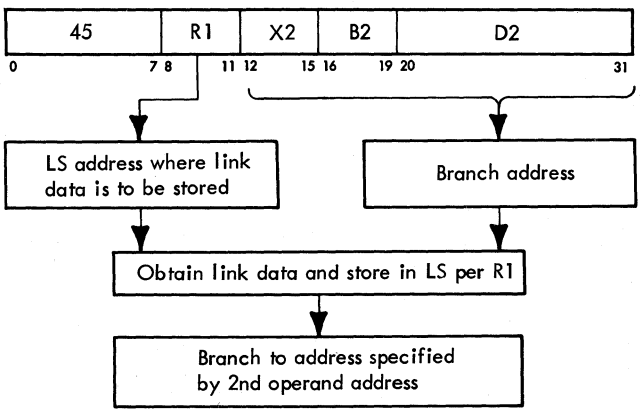
Now assume that E(12-15) does not equal zero (indicating a successful branch). D(21,22) is tested for a value of 11. If not equal to 11, PSW(32-39) is transferred to T(32-39). The double word containing the branched-to instruction (requested during I-Fetch of the branch instruction) is now present at the SDBO and is gated to Q.

From Q, the correct halfword is transferred to R per D(21,22). Then 8 is added to D, and the result placed in the IC to address the next sequential double word from main storage. The data in T is now transferred to LS per E(8-11). Since the data in R and the IC is not yet stable, thus preventing decoding of the next instruction from R or the instruction address from IC, a branch end-op cycle is taken. Decoding of the next instruction during a branch end op occurs off the SDBO which, at this time, is stable.

If D(21,22) is equal to 11, the next instruction is located in the last halfword of the double word requested during I-Fetch. The PSW data, in this case, is transferred to T(32-39). The double word requested during I-Fetch is present at the SDBO and is gated to Q. From Q, the last halfword is transferred to R per D(21,22). Since the halfword that contains the next instruction to be executed is the last halfword of the double word, Q must be filled with a new double word to allow continuous operation. Accordingly, a 3-cycle storage request is issued per the IC to obtain the next double word. The address of the double word is tested for validity; if the address is invalid, the I-Fetch-invalid-address trigger is set. Then 8 is added to D, and the result is transferred to the IC. This action allows selection of the next sequential double word when requested by the IC. The data in T is then transferred to LS per E(8-11), and the IC is updated by 8. By this time, the requested double word is present at the SDBO and is gated to Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded off R.

3.12.2.2 BAL (45)

- Stores link information in LS register specified by R1 and branches to address specified by 2nd operand.
- RX format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  Address of next instruction is in D.

- Unconditional branch.

- If execute trigger is reset and ABC equals 0, IC reflects correct address and is stored as link address information.

- If execute trigger is reset and ABC does not equal 0, IC is reduced by 8 and then stored as link address information.

- If execute trigger is set and IC(21, 22) equals 11, IC is reduced by 16 and stored as link address information.

- If execute trigger is set and IC(21, 22) does not equal 11, IC is reduced by 8 and stored as link address information.

The Branch and Link (BAL) instruction stores the address of the instruction which, if the branch were unsuccessful, would be the next sequential instruction address. Stored with the address is link information consisting of the instruction length code, the CC, and the program mask bits. The instruction length code stored is 2. The BAL instruction is an unconditional branch with an RX format and an op code of 45.

Figure 6068 FEDM, is a flow chart of the BAL instruction. At the start of execution, the first 16 bits of the instruction are in E, the address of the next instruction is in D, and a 3-cycle storage request has been generated per D for the instruction being branched to. At the beginning of the operation, the last three bits of the IC are transferred to the ABC. This value will be used to determine the correct value of the IC before it is stored in LS as link address information. The contents of the IC are transferred to the parallel adder and reduced by 8. This value is then transferred to T, from where it and the remainder of the link data [PSW(32-39)] will be transferred to LS. The state of the execute trigger is then tested; if it is set, the branch operation is the subject instruction of an Execute instruction.

First assume the execute trigger is reset. The ABC is now checked for all zeros. If equal to zero, it indicates that the branch instruction now being executed was located in the third and fourth halfwords of Q. Normally, when an instruction with an RX format occupies the last two halfwords of Q, a storage request is generated during I-Fetch per the IC and the IC is updated by 8. Since this is a branch instruction, however, the storage request from the IC is prevented and the IC is not increased by 8. Therefore, the address presently in T (after being reduced by 8) is incorrect and must be changed.

PSW(32-39), which contains the instruction length code, the CC, and the program mask bits, is transferred to T(32-39). The contents of D are then transferred to the parallel adder, increased by 8, and transferred to the IC. The IC now contains the address of the double word in main storage which follows the double word containing the branched-to instruction. At this time, a storage request for the next double word is issued if D(21, 22) equals 11.

The contents of T (link information) are now increased by 8 to give the correct double-word address in T(40-63). The link information can now be stored in LS. At this time, the data requested per D during I-Fetch is at the SDBO and is gated to Q. From Q, the correct halfword is transferred to R per D(21, 22). The contents of T are then transferred into LS per E(8-11).

D(21, 22) is now checked for a value of 11. If it is equal to 11, the next instruction to be executed is in the last halfword of Q. The IC is then updated by 8 to address the next double word in main storage. (This double word is two addresses higher than the branched-to instruction.) By this time, the double word that was requested during the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. If D(21, 22) did not equal 11, a branch end-op cycle is taken and the next instruction is decoded off the SDBO.

Assume that the ABC did not equal zero (Figure 6068, FEDM). PSW(32-39) is transferred to T(32-39). Since the ABC was not equal to zero, the address portion of the link information is correct and can be stored in LS per E(8-11). The contents of D are then increased by 8 and placed in the IC. This address is the address of the next sequential double word in main storage, following the double word containing the branched-to instruction. A storage request for this word is then issued if D(21, 22) is equal to 11. At this time, the branched-to instruction (that instruction requested during I-Fetch) is present at the SDBO and the double word from main storage is gated into Q. The correct halfword is transferred to R per D(21, 22). D(21, 22) is then tested. If D(21, 22) does not equal 11, a branch end-op cycle is taken, the instruction format to be

executed next is decoded off the SDBO, and the instruction address decoded from D. If D(21,22) equals 11, the operation is as previously described with a normal end-op cycle taking place.

Now assume that the execute trigger is set, indicating that the branch instruction is the subject instruction of an Execute instruction. IC(21,22) is tested for a value of 11. If IC(21,22) equals 11, the Execute instruction was located in the second halfword of its double word and, when in Q, automatically issued a storage request and increased the IC by 8. This increase results in an address in the IC that is 16 bytes higher than the double-word address containing the Execute instruction. Since the address that is stored as link information is the address of the double word containing the Execute instruction, the address has to be reduced by 16. The address in T, however, has already been reduced by 8; therefore, only 8 must be subtracted from it. The remainder of the link information is now transferred from PSW(32-39) to T(32-39).

The contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC to address the next double word. The execute trigger is now reset. A storage request for that double word whose address was just placed in the IC is issued if D(21,22) equals 11. At this time, the data requested during I-Fetch of the branch instruction is present on the SDBO and is gated to Q. The correct halfword in Q is then transferred to R per D(21, 22). The address portion of the link information located in T is transferred to the parallel adder, where it is decreased by 8. This value is now equal to the address of the double word that contains the Execute instruction and is transferred to T. From T, the link information is transferred to LS per E(8-11). D(21,22) is now tested for a value of 11. If D(21,22) equals 11, the data requested during the branch instruction is gated from the SDBO to Q, and a normal end-op cycle is taken. If D(21,22) equals some value other than 11, the operation proceeds directly to the normal end-op cycle to complete the operation.

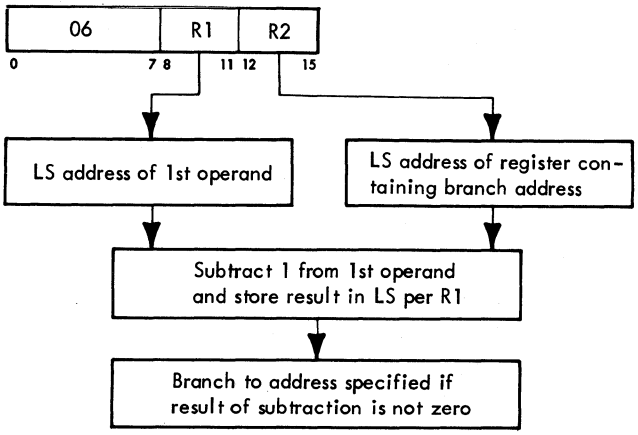
Now assume that IC(21,22) did not equal 11. This condition indicates that the Execute instruction was not in the second halfword and a storage request was not automatically generated. The link information presently in T is therefore correct and can be stored in LS per E(8-11). The contents of D are increased by 8 and placed in the IC. Again a storage request is generated if D(21,22) equals 11. At this time, the double word containing the branched-to instruction is located in the SDBO and is gated to Q. The correct halfword for Q is then transferred by R per D(21,22). The contents of D(21,22) are tested for a value of 11, and operations continue as previously described, ending with a branch end op or a normal end op.

3.12.3 BRANCH ON COUNT

- Subtracts 1 from 1st operand and, if result is not 0, branches to address specified.
- Instructions:
  Branch on Count, BCTR (06)
  Branch on Count, BCT (46)

3.12.3.1 BCTR (06)

- Subtracts 1 from 1st operand and, if result is not 0, branches to address specified.
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in ST.
  Branch address is in D.

The Branch on Count (BCTR) instruction subtracts 1 from the first operand (contents of the LS register specified by the R1 field) and, if the result does not equal zero or the R2 field does not equal zero, branches to the address specified by the contents of the LS register designated by the R2 field. The result of the subtraction is stored in the first operand location. If the result of the subtraction equals zero, the next sequential instruction is executed. If E(12-15) equals zero, the branch is automatically unsuccessful. The BCTR instruction has an RR format with an op code of 06.

A flow chart of the BCTR instruction is contained in Figure 6069, FEDM. At the start of execution, the instruction is in E, the first operand is in ST, the branch address is in D, and a 3-cycle storage request has been issued per D for the instruction being branched to. The first operand is transferred from T to B and from B to the parallel adder, where 1 is subtracted from the operand to determine whether the branch is successful. Before subtracting 1, E(12-15) is tested for zeros. As previously stated, if E(12-15) equals zero, the branch is unsuccessful; if other than zero, the branch is successful. Assume that E(12-15) does not equal zero. The contents of B are transferred to the parallel adder, where 1 is subtracted from the operand; the result is transferred via T into LS. The result of the subtraction is tested for all zeros; if zero, the branch is unsuccessful; if not zero, the branch is successful.

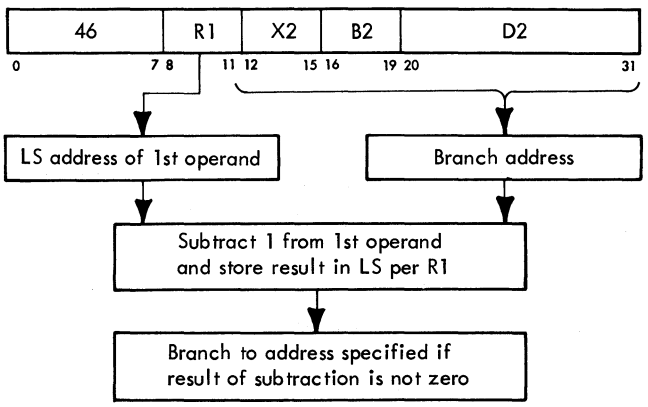
First assume that the branch is successful. Since D(21,22) indicates in which halfword the branched-to instruction is located, it is examined. If D(21,22) equals 11, the instruction is located in the last halfword of the double word requested during I-Fetch of the branch instruction. The contents of D, therefore, are updated by 8 and transferred to the IC. By this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q. The last halfword of Q is then transferred to R per D(21,22). A 3-cycle storage request for the next double word is now issued per the IC. The contents of the IC are then transferred to the parallel adder, updated by 8, and transferred back to the IC to select another double word from main storage. At this time, the double word which sequentially follows the double word containing the branched-to instruction is present at the SDBO and is gated into Q. A normal end-op cycle is taken, and the next instruction to be executed is decoded from R.

If D(21,22) did not equal 11, the branched-to instruction is located in some halfword other than the last. In this case, the contents of D are transferred to the parallel adder, updated by 8, and then transferred to the IC. At this time, the double word containing the branched-to instruction is present at the SDBO and can be gated into Q. From Q, the correct halfword is transferred to R per D(21,22). A branch end-op cycle is taken, and the next instruction is decoded off the SDBO.

Now assume that the branch is unsuccessful. If IC(21,22) equals 11, a storage request per the IC must be given during the branch operation execution phase to obtain the next sequential double word from main storage. Once the storage request is issued, the execute trigger is tested. If set, it indicates that the branch instruction is the subject instruction of an Execute instruction. Therefore, a normal end-op cycle is taken to complete the operation. The data requested in this case is not gated into Q. If the execute trigger is reset, IC(21,22) is tested to see whether it contains 11. If the value is 11, the IC is updated by 8 (to select another double word) and placed in the IC and D. By this time, the data requested by the storage request given during the execution phase of the branch instruction is present at the SDBO and can be gated into Q. A normal end-op cycle is then taken, and the next instruction to be executed is decoded off R. If IC(21,22) does not equal 11, then it equals 00, and the next instruction is located in the first halfword of the double word requested during the execution phase of the branch instruction. The IC is updated by 8, and the result placed in D and the IC. At this time, the data requested during the execution phase is present at the SDBO. This double word is gated to Q; the correct halfword from Q(0-15) is transferred to R. Since the format is normally decoded off R and the data just placed in R is not yet stable, a branch end-op cycle is taken. This cycle allows the next instruction to be decoded off the SDBO, which at this time is stable.

3.12.3.2 BCT (46)

- Subtracts 1 from 1st operand and, if result is not 0, branches to address specified.
- RX format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is in ST.
  Branch address is in D.

- This instruction is similar to BCTR but does not test E(12-15) for zero. (Refer to paragraph 3.12.3.1.)

- Figure 6069, FEDM.

3.12.4 BRANCH ON INDEX

- Adds 1st operand (R1) to 3rd operand (R3) and stores sum in LS register addressed by R1. Compares sum with comparand and if sum is greater than, less than, or equal to the comparand, depending on the instruction, branches to address specified.

- Instruction format: RS.

- Instructions:
  Branch on Index High, BXH (86)
  Branch on Index Low or Equal, BXLE (87)

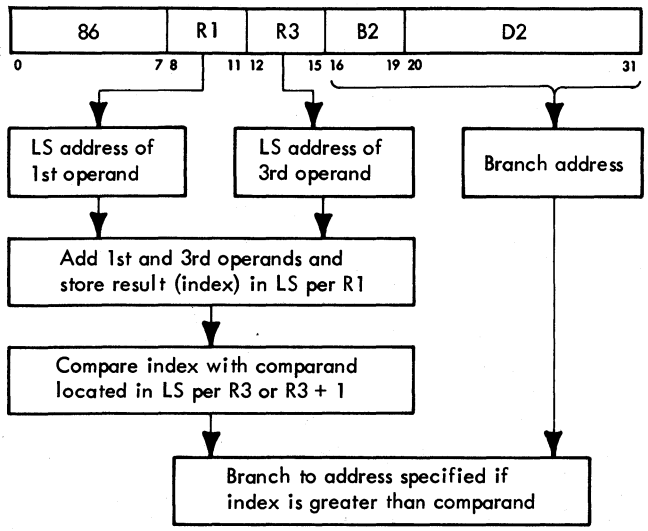
- Sum of 1st and 3rd operands always stored whether branch is successful or not.

- Comparand address (R3 or R3 + 1) must be odd.

3.12.4.1 Branch on Index High, BXH (86)

- Branches to address specified if result of addition is greater than comparand.

- RS format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is in ST.
  Branch address is in D.

The Branch on Index High (BXH) instruction, which has an RS format with an op code of 86:

1. Adds third operand to first operand.

2. Stores result (index) in LS register addressed by R1.

3. Compares index with a comparand obtained from an LS register addressed by R3 or R3+1.

4. Branches if sum is larger than comparand.

Figure 6070, FEDM, is a flow chart of the BXH instruction. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in ST, the branch address is in D, and a 4-cycle storage request is issued per D for the instruction being branched to. To allow the third operand to be placed in T without destroying the first operand, the first operand is transferred to B. The third operand is then transferred from LS per E(12-15) and placed in T. The two operands are added, and the result is transferred to B. The comparand, the value that the sum of the two operands (index) is compared with, is now transferred from LS and placed in T. The contents of B and the 2's-complement of T are transferred to the parallel adder and added to determine whether the branch is successful. IC(21,22) is tested for either 11 or 00. If the branch is unsuccessful, this test sets up conditions to issue a storage request per the IC to obtain the next sequential double word after the double word containing the branch instruction.

Assume that IC(21,22) equals 11 or 00. D(21,22) is now tested to determine where in the double word the next instruction is located. If D(21,22) is equal to 11, the next instruction to be executed is contained in the last halfword of the double word requested during I-Fetch of the branch instruction. The PAL's and E(7) are now tested to determine whether the branch is successful. First assume the branch is successful; that is, the PAL's are positive (index greater than the comparand) and E(7) equals 0. The contents of B (index) are transferred to T, and from there to LS per E(8-11). At this time, the double word requested during I-Fetch is present at the SDBO and is gated into Q. The halfword containing the next instruction to be executed is then transferred to R per D(21,22). The contents of D are now updated by 8 and placed in the IC. This

action allows the selection of the next double word in main storage. A 3-cycle storage request is then issued per the new value in the IC. The IC is then updated by 8 to allow selection of the next double word from main storage when needed. By this time, the data requested during the execution phase of the branch instruction is available at the SDBO and can be gated into Q. A normal end-op cycle is then taken to complete the operation. During the end-op cycle, the next instruction executed is decoded off R.

Now assume the branch is unsuccessful. That is, the PAL's are not positive and E(7) is 0. The contents of B (index) are transferred to T, and from there to LS per E(8-11). Also, the double word requested during I-Fetch is present at the SDBO and is gated into Q. A 3-cycle storage request is issued per the IC, which at this time contains the address of the double word that sequentially follows the double word containing the branch instruction. The state of the execute trigger is now tested. If set, it indicates that the branch instruction was the subject instruction of an Execute instruction and a normal end-op cycle is taken, completing the operation. If the execute trigger is reset, IC(21,22) is tested for a value of 11. Recall that, when IC(21,22) was previously tested, it was checked for a value of either 11 or 00. To proceed sequentially in the program without any delay, it is now necessary to determine which value the IC contains. First, assume that IC(21,22) equals 11. This value indicates that the next instruction occupies the last halfword of the double word in which the branch instruction is located and is presently in R. Recall that R is where format decoding of an instruction occurs. This situation being the case, the IC is updated by 8 to address the double word that is 16 bytes from the double-word address containing the branch instruction. The data that was requested when it was found that the branch was unsuccessful is now present at the SDBO (Figure 6070B, FEDM) and can be gated into Q. A branch end-op cycle is taken, completing the operation. The next instruction is decoded off the SDBO when the data is transferred to Q.

Now assume that IC(21,22) equals 00. In this case, the data to be executed is located in the first halfword of the double word requested during the execution of the branch when the branch instruction was found to be unsuccessful. The IC is updated by 8. At this time, the data is present at the SDBO and can be gated into Q. The first halfword in Q is transferred to R. Recall that the format for the next instruction is normally decoded off R. Since the next instruction to be executed has just been transferred into R, this data is not yet stable and cannot be decoded. A branch end-op cycle is therefore taken. This cycle allows the next instruction format to be decoded off the SDBO. The SDBO is stable at this time and therefore can be used.

Now assume that when D(21,22) was tested for 11 some other value was found. Again, conditions are tested to see whether the branch is successful (Figure 6070A, FEDM). If successful, the data in B is transferred to T and from there to LS per E(8-11). Also, the data that was requested during I-Fetch of the branch instruction is present at the SDBO and can now be gated into Q. From Q, the halfword containing the next instruction is transferred to R per D(21,22). The contents of D are then updated by 8 and transferred to the IC to address the next double word. Since the instruction to be executed next has just been placed in R and is not yet stable, a branch end-op cycle is taken and the instruction is decoded from the SDBO.

If D(21,22) equals 11 and the branch is unsuccessful, the contents of B are transferred to T, from where they are transferred to LS per E(8-11). A 3-cycle storage request is issued per the IC. This request is for the double word that sequentially follows the double word in main storage containing the branch instruction. The state of the execute trigger is tested next. From this point on, the operation is the same as that previously discussed for an unsuccessful branch.

Assume, now, that when IC(21,22) was initially tested for either 11 or 00 (Figure 6070A, FEDM), neither of these values was present. Again D(21, 22), the PAL's, and E(7) are tested to determine whether the operation is a successful branch and where the next instruction is located in the double-word address being branched to. Assume that D(21, 22) equals 11 and the branch is successful (Figure 6070B, FEDM). Operation from this point on is identical with a successful branch, as previously described, when D(21,22) equals 11.

Now assume that D(21,22) equals 11 and that the branch is unsuccessful. The contents of B are transferred to T, from where they are transferred to LS per E(8-11). Since the branch is unsuccessful and the contents of R have not been changed, R still contains the instruction that is located in the halfword following the last halfword of the branch instruction. A normal end-op cycle, therefore, can be taken and the instruction decoded off R.

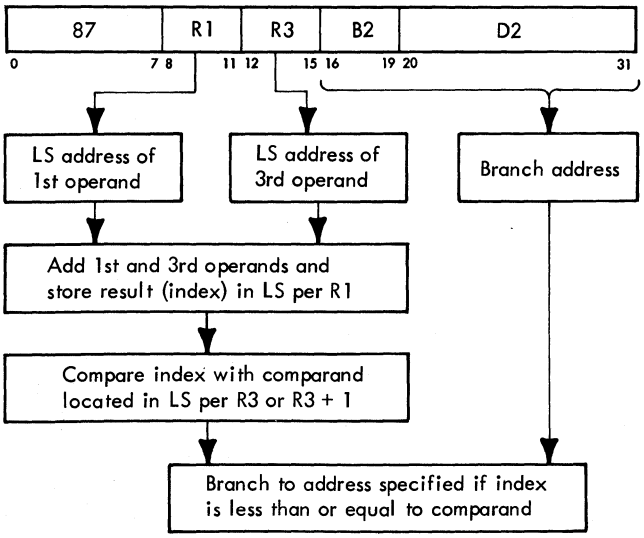
Now assume that D(21,22) did not equal 11 and the branch is successful (Figure 6070B, FEDM). The contents of B are transferred to T, from where they are transferred to LS per E(8-11). At this

time, the data requested during I-Fetch of the branch instruction is present at the SDBO and is gated to Q. From Q, the correct halfword containing the branched-to instruction is gated into R per D(21,22). The contents of D are then updated by 8 and transferred to the IC to select the next double word from main storage. Since the instruction to be executed has just been placed in R and is not yet stable, a branch end-op cycle is taken and the instruction format is decoded off the SDBO.

If the branch is unsuccessful and D(21,22) does not equal 11, the operation is identical with the case where D(21,22) equals 11 and the branch is unsuccessful. A normal end-op cycle is taken.

3.12.4.2 Branch on Index Low or Equal, BXLE (87)

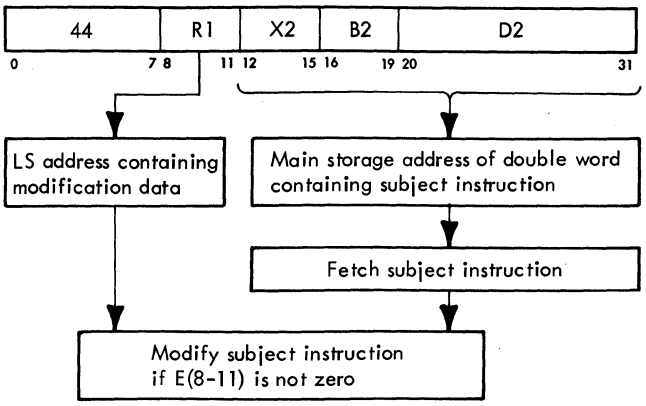
- Branches to address specified if result of addition is less than or equal to comparand.
- RS format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is in ST.
  Branch address is in D.
- Instruction is similar to BXH except branching occurs on low or equal result (paragraph 3.12.4.1).
- Figure 6070, FEDM.

3.12.5 EXECUTE, EX (44)

- Executes an instruction (subject) specified by 2nd operand address. Subject instruction can be modified by contents of LS register designated by R1.
- RX format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is in ST.
  Address of subject instruction is in D.
- Modification of subject instruction accomplished by logically OR'ing bits 8-15 of subject instruction and bits 24-31 of LS register specified by R1.
- If E(8-11) equals 0, no modification.
- If an Execute instruction is subject instruction, an execute interruption occurs.
- If effective address of Execute instruction is odd, a specification interruption occurs.
- Execute trigger set to indicate next instruction is subject of Execute instruction.
- Address-store-compare trigger set to indicate that Q data is not valid and needs to be refilled.
- If program interruptions are pending, normal end-op cycle is taken; if not, branch end-op cycle is taken.

The Execute (EX) instruction, which has an RX format with an op code of 44, executes a designated instruction whose address in main storage is the second operand address. This designated instruction is referred to as a subject instruction and can be modified by the contents of the first operand located in the LS register specified by R1. Modification of the subject instruction is accomplished by OR'ing bits 8-15 of the subject instruction with bits 24-31 of the LS register specified by R1. If R1 is equal to 0, no modification takes place. The subject instruction may be 16, 32, or 48 bits long. If the subject instruction is another EX instruction, an execute interruption occurs and operation is suppressed. If the effective address of the EX is odd, a specification interruption occurs.

A flow chart of the EX instruction is contained in Figure 6071, FEDM. At the start of execution, the first 16 bits of the instruction are in E, the first operand is in ST, the address of the subject instruction is contained in D, and a 3-cycle storage request for the subject instruction is generated per D. At the beginning of execution, a test for specification and execute interruptions is made. If the specification interruption is present (effective address of EX was odd), a program interruption occurs and the operation is suppressed. If an execute interruption is present (the EX instruction is the subject instruction of an Execute instruction), an execute interruption occurs and the operation is suppressed. If no execute or specification interruptions are present, the operation continues. The STC is loaded to 111, allowing the transfer of T(56-63) to the serial adder for modification of the subject instruction if modification is to be accomplished. The contents of D are now transferred to the parallel adder and updated by 8 to address the double word that follows the double word containing the subject instruction of the EX instruction. This value is then transferred to D. PAL(61-63) is now transferred to the ABC to select the correct byte for modification of the subject instruction.

D(21,22) is tested to determine whether the subject instruction is contained in the last halfword of the double word that was requested during I-Fetch, or in some halfword other than the last. If D(21,22) equals 11, the subject instruction is in the last halfword; if any other value, the subject instruction is in some other halfword. First assume that D(21,22) equals 11. Since the subject instruction is located in the last halfword of the double word addressed during I-Fetch, there is a possibility that part of the instruction is contained in the next double word to be addressed. This possibility exists if the subject instruction has a format other than RR. Therefore, the next few operations determine the format of the subject instruction. By doing these tests, an extra request can be prevented, one that can cause an invalid address or protection check if the instruction has an RR format.

At this time, the data requested during I-Fetch is present at the SDBO and can be gated into Q and AB. From Q, the last halfword is transferred to R. The sixth byte in AB is then transferred to the serial adder per the ABC. Minus 64 (11000000) is sent to the serial adder, where it is logically AND'ed with the op code of the subject instruction. If the op code denotes an RR format, SAL should now equal zero. 1 is then added to the ABC to transfer the last halfword of AB to the serial adder if the instruction is to be modified.

E(8-11) is now tested to see whether the subject instruction is to be modified. If E(8-11) equals 0000, the subject instruction is not to be modified; if any other value, the instruction is to be modified. First assume that E(8-11) equals 0000. SAL(0-7) is now tested. Recall that the value in SAL indicates whether the subject instruction has an RR format, and recall that it has already been determined that the subject instruction is contained in the last halfword of Q. Therefore, assume that SAL(0-7) equals zero. Since this value indicates that the subject instruction is an RR instruction, there is no need to issue a storage request for the next instruction because there is no information in that double word that will affect the operation of the RR instruction. The request for the next double word occurs during I-Fetch of the RR instruction.

The last byte in AB is then transferred to T via the serial adder per the ABC and STC. From T, the data is tranferred to R. The address-store-compare and execute triggers are set. A test is made for a pending program interruption. If one exists, a program interruption cycle is taken; if there is no interruption, the contents of R are transferred to E and STAT G is set. A branch end-op cycle completes the operation.

Now assume that SAL(0-7) does not equal zero, indicating that the subject instruction has some format other than RR. So that the complete word may be decoded prior to execution of the instruction, the double word that sequentially follows the subject instruction must be requested. The last byte in AB is transferred to T via the serial adder per the ABC and STC. The address-store-compare trigger is set, and a 3-cycle storage request is issued for the next sequential double word. The microprogram waits (two storage cycles) until the data requested

is present at the SDBO, at which time the data is gated to Q. The execute trigger is set, and a test is made for a pending interruption. If an interruption is present, a program interruption cycle is taken; if not, the data in R is transferred to E and STAT G is set. A branch end-op cycle completes the operation.

If when E(8-11) is tested some value other than 0000 is found, the subject instruction is to be modified. SAL(0-7) is tested to determine whether the subject instruction has an RR format. Assume an RR format. The last byte in AB is then transferred to the serial adder per the ABC. This byte is then logically OR'ed with the last byte of ST which was transferred to the serial adder per the STC. The results of this OR'ing are then transferred to ST per the STC. The data in T is transferred to R. The execute trigger is set, and the operation continues in the same manner described previously.

Now assume that SAL(0-7) contains some value other than zero. In this case, after the subject instruction is modified, a storage request for the next double word must be made. The microprogram waits (two storage cycles) until the data is present at the SDBO, after which the data is transferred to Q. From this point on, the operation is identical with that described for an RR instruction.

Now return to the point where D(21,22) is tested to see whether the subject instruction occupies the last halfword of the double word requested during I-Fetch. If D(21,22) equals some value other than 11, a storage request is not issued during the execution of the EX instruction. E(8-11) is now tested to determine whether the subject instruction is to be modified. If E(8-11) does not equal 0000, the instruction must be modified. After the data is placed in Q and AB, and the correct data is placed in the serial adder per the ABC, operation is identical with that of an RR instruction that is to be modified (Figure 6071, FEDM).

If E(8-11) equals 0000, however, the subject instruction is not to be modified. Therefore, the correct halfword in Q need only be transferred to R per D(21,22). The address-store-compare and execute triggers are set, and the test for a pending interruption is made. The operation continues in the same manner described previously.

SECTION 7. STATUS SWITCHING

This section discusses the program states and the instruction format, data flow, and interruptions for the status switching instructions. An analysis of these instructions follows.

3.13 INTRODUCTION

A set of instructions is provided to switch the status of the CPU and main storage, and for communication between systems. The overall CPU status is determined by eight program states: problem/supervisor, wait/running, masked/interruptable, and stopped/operating. Most of these states are indicated by a bit in the program status word (PSW). The CPU status is further determined by the instruction address, the CC, the instruction length code, the storage protection key, and the interruption code. These also occupy PSW bits.

3.13.1 PROGRAM STATES

The eight program states which determine the overall CPU status differ in the way they affect the CPU functions and in the way their status is indicated and switched. The following paragraphs contain a brief description of the program states.

3.13.1.1 Problem/Supervisor

- In problem state, all I/O, protection, and direct control instructions as well as Load PSW, Set System Mask, and Diagnose instructions are invalid.
- In supervisor state, all instructions are valid.
- PSW(15) determines whether problem or supervisor state:
  If a 1, CPU is in problem state.
  If a 0, CPU is in supervisor state.
- State of PSW(15) is changed by issuing Load PSW instruction or through IPL.

In the problem state, all I/O, protection, and direct control instructions are invalid as well as the Load PSW, Set System Mask, and Diagnose instructions. These instructions are called "privileged instructions". A privileged instruction encountered in the problem state constitutes a privileged-operation interruption and interrupts the operation. In the supervisor state, all instructions are valid.

The CPU is switched between the problem and supervisor states by changing PSW(15). When PSW(15) is a 1, the CPU is in the problem state; when a 0, the CPU is in the supervisor state. This bit can be changed only by introducing a new PSW. Thus, the status switching for problem/supervisor state may be performed by a Load PSW instruction containing a new PSW with the desired value in bit 15. Because the Load PSW instruction is a privileged instruction, the CPU must be in the supervisor state prior to the switch. The CPU status can also be changed between problem and supervisor states by issuing a Supervisor Call instruction or an initial program load (IPL). The Supervisor Call instruction causes an interruption which will load new PSW data. This new PSW data may change the state of the CPU. Similarly, the IPL introduces a new PSW. The new PSW may introduce the problem or supervisor state, regardless of the preceding CPU state.

3.13.1.2 Wait/Running

In the wait state, no instructions are processed and main storage is not addressed. In the running state, I-Fetch and execution proceed in the normal manner. The CPU status is switched between the wait and running states by PSW(14). When PSW(14) is a 1, the CPU is in the wait state; when a 0, in the running state. This bit can only be changed by introducing a new PSW. Thus, switching from the running to the wait state may be achieved by the privileged instruction Load PSW, by an interruption such as given by a Supervisor Call instruction, or by the IPL. Switching from the wait to the running state may be achieved by an I/O or external interruption or by IPL. The new PSW may introduce the wait or running state regardless of the preceding CPU state.

3.13.1.3 Masked/Interruptable

- Masked/interruptable state is determined by:
  System mask bits [PSW(0-7)].
  Machine check mask bit [PSW(13)].
  Program mask bits [PSW(36-39)].

- If mask bit = 1 (interruptable state), associated interruption is accepted.

- If mask bit = 0 (masked state), system and machine-check interruptions remain pending; program interruptions are ignored.

- Mask bit status is changed by issuing Load PSW or Supervisor Call instruction or by issuing IPL.

The masked/interruptable state of the CPU is determined by the system mask bits [PSW(0-7)], the machine-check mask bit [PSW(13)], and the program mask bits [PSW(36-39)]. If a mask bit = 1, the associated interruption is accepted; if it = 0, system and machine-check interruptions remain pending and program interruptions are ignored. The PSW bits and interruptions that will occur if the bit is active are listed in Table 3-25.

TABLE 3-25. PSW INTERRUPTION BIT DESIGNATION

PSW Bit	Interruption
System mask	
0	Multiplexor channel
1	Selector channel 1
2	Selector channel 2
3	Selector channel 3
4	Selector channel 4
5	Selector channel 5
6	Selector channel 6
7	Timer, INTERRUPT pushbutton, or external signals
Machine check mask	
13	Machine check
Program mask	
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow
39	Significance

The masked/interruptable state of the CPU is switched by changing the mask bits in the PSW. The program mask may be changed separately by the Set Program Mask instruction, and the system mask may be changed separately by the Set System Mask instruction. The machine-check mask bit can be changed only by introducing an entirely new PSW, as in the problem/supervisor and wait/running states. Thus, a change in the entire masked status may be achieved by the privileged instruction Load PSW, by an interruption such as for the Supervisor Call instruction, or by the IPL. Regardless of the preceding program state, the new PSW may introduce a new mask status.

3.13.1.4 Stopped/Operating

- In stopped state, instructions and interruptions are not executed.

- In operating state, instructions are executed as long as CPU is not in wait state. Interruptions are taken if not masked off.

- A change in the stopped/operating states can occur only by manual intervention or machine malfunction.

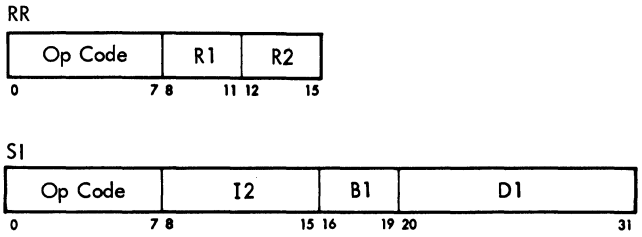
When the CPU is in the stopped state, instructions and interruptions are not executed. When the CPU is in the operating state, instructions are executed as long as the CPU is not also in the wait state. Interruptions are taken if they are not masked off. A change in the stopped/operating states can occur only by manual intervention or by machine malfunction. No instruction or interruption can start or stop the CPU. The CPU is placed in the stopped state when the STOP pushbutton on the system control panel is depressed, when an address comparison indicates equality, when the RATE switch is set to the INSN STEP position, and after power is turned on or following a system reset, except during IPL. The CPU is placed in the operating state when the START pushbutton on the system control panel is depressed and when an IPL is successfully completed.

Changing from the operating state to the stopped state occurs at the end of instruction execution and prior to the start of the next instruction execution. When the CPU is in the wait state, the change from operating to stopped occurs immediately. All interruptions pending and masked on are taken while the CPU is still in the operating state. The interruptions cause an old PSW to be stored and a new

PSW to be fetched before entering the stopped state. Once the CPU is in the stopped state, interruptions are no longer taken but remain pending.

3.13.2 INSTRUCTION FORMAT

Status switching instructions have two formats:



In the RR format, the R1 and R2 fields specify an LS general register except when used in the Supervisor Call instruction. The R1 and R2 fields in the Supervisor Call instruction contain interruption codes. In the Set Program Mask instruction, the R2 field is ignored.

In the SI format, the I2 field is ignored for the Load PSW and Set System Mask instructions. In the Write Direct and Read Direct instructions, the I2 field contains eight timing signals that are sent to an external unit. The contents of the LS general register specified by the B1 field are added to D1 to form a main storage address of an operand to be used in the instruction specified except for Read Direct. The Read Direct instruction uses the address derived as the address where the data from an external unit is to be stored.

3.13.3 DATA FLOW

- Functional units used are Q, R, E, ST, AB, F, G, D, IC, parallel adder, serial adder, STC, and ABC.

Figure 9057, FEDM, is a diagram of the data flow for the status switching instructions. Following is a list of the functional units and their purposes:

1. Q: Holds the double word containing the instruction being executed. It may also hold the next sequential double word after the double word containing the instruction being executed.

2. R: Contains the instruction to be performed after the instruction presently being executed.

3. E: Contains the instruction presently being executed. For Read Direct and Write Direct instructions, E contains the timing signals sent to the external unit.

4. ST: Buffers the operand being operated on.

5. AB: Buffers the operand being operated on.

6. F: Buffers the storage key both before it is placed in main storage during the Set Storage Key instruction and after it is taken from main storage during the Insert Storage Key instruction. This register also holds data coming from an internal unit during the Read Direct instruction.

7. G: Buffers a byte of data being sent to an external unit when performing a Write Direct instruction.

8. D: Contains the main storage address of the block containing the storage key requested during Set Storage Key and Insert Storage Key instructions. This register also selects the byte to be used in Set System Mask, Write Direct, and Read Direct instructions.

9. IC: Contains the address of the double word to be operated on next.

10. Parallel adder: Provides the data transfer path between AB, ST, D, and IC.

11. Serial adder: Provides the data transfer path between AB, F, ST, and PSW.

12. STC: Controls the selection of data from and placement of data into ST, primarily during the Set Storage Key, Insert Storage Key, Write Direct, Read Direct, and Test and Set instructions.

13. ABC: Controls the selection of data from and placement of data in AB, primarily during the Set System Mask, Set Storage Key, and Test and Set instructions.

3.13.4 INTERRUPTIONS

- Program interruptions:
  Operation
  Privileged operation
  Protection
  Addressing
  Specification

- Current PSW is stored as old PSW and new PSW is obtained.

- Interruption code in old PSW identifies cause of interruption.

Table 3-26 lists the interruptions that may occur in the status switching instructions.

TABLE 3-26. STATUS SWITCHING INSTRUCTION INTERRUPTIONS

Interruption	Interruption Code PSW Bits 16-31	ILC	How Instruction Execution Is Finished
Operation	00000000 00000001	1, 2, 3	Suppressed
Privileged operation	00000000 00000010	1, 2	Suppressed
Protection	00000000 00000011	0, 2, 3	Terminated
Addressing	00000000 00000101	1, 2, 3	Suppressed
Specification	00000000 00000110	1, 2, 3	Suppressed

These interruptions cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code inserted in the old PSW identifies the cause of the interruption. The following listing briefly describes the interruptions:

1. Operation: Occurs if the direct control feature is not installed and the instruction being executed is either Read Direct or Write Direct. The operation is suppressed.

2. Privileged Operation: Occurs if a Load PSW, Set System Mask, Set Storage Key, Insert Storage Key, Write Direct, Read Direct, or Diagnose instruction is encountered while the CPU is in the problem state. The operation is suppressed.

3. Protection: Occurs if the storage key of the location designated by the instruction does not match the protection key in the PSW. The operation is terminated.

4. Addressing: Occurs if an address designates a location outside the available main storage. The operation is suppressed.

5. Specification: Occurs if (1) the operand address of a Load PSW or Diagnose instruction does not have 0's in the three low-order bit positions, (2) the block address specified by the Set Storage Key or Insert Storage Key instruction does not have 0's in the four low-order bit positions, or (3) the protection feature is not installed and a PSW with two nonzero protection keys is introduced. The operation is suppressed.

3.14 INSTRUCTION ANALYSIS

Table 3-27 lists the 10 status switching instructions with their format, mnemonic code, op code, and interruptions. A functional description of each instruction is contained in the following paragraphs.

TABLE 3-27. STATUS SWITCHING INSTRUCTIONS

Instruction	Format	Mnemonic Code	Op Code	Interruptions
Load PSW	SI	LPSW	82	M, P, A, S
Set Program Mask	RR	SPM	04	-
Set System Mask	SI	SSM	80	M, P, A
Supervisor Call	RR	SVC	0A	-
Set Storage Key	RR	SSK	08	M, A, S, O
Insert Storage Key	RR	ISK	09	M, A, S, O
Write Direct	SI	WRD	84	M, P, A, O
Read Direct	SI	RDD	85	M, P, A, O
Diagnose	SI	-	83	M, P, A, S
Test and Set	SI	TS	93	P, A

Notes: M: Privileged operation
P: Protection
A: Addressing
S: Specification
O: Operation

3.14.1 LOAD PSW, LPSW (82)

- Loads 2nd operand into PSW register, thereby making operand new PSW.

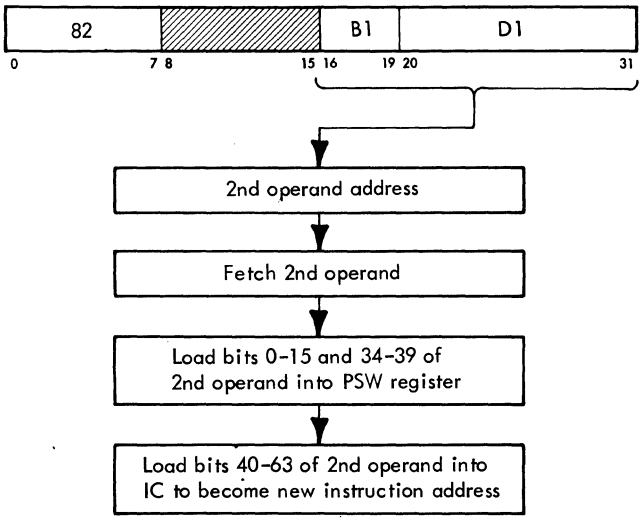
- Bits 16-33 of double word are ignored and not loaded into new PSW.

- Bits 0-15 become new system mask, protection key, and program state bits.

- Bits 34-39 become new CC and program mask.

- Bits 40-63 become new instruction address.

- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  2nd operand address is in D.

The Load PSW (LPSW) instruction, which has an SI format with an op code of 82, loads into the CPU the double word that is contained in the location designated by the second operand address. The double word becomes the new PSW for the next sequence of instructions. Bits 40 through 63 of the double word become the new instruction address. During the LPSW instruction, the address is not checked for storage availability or for an even byte address; these checks occur as part of the execution of the next instruction. Since bits 16 through 33 of the PSW (interruption code and instruction length code) are changed when the PSW is stored during an interruption, they are ignored when performing the LPSW instruction. When the double word being loaded as the new PSW contains a 1 in position 15 or 14, the CPU enters the problem state or wait state, respectively. This instruction is the only instruction available for entering the problem state or wait state.

Figure 6072, FEDM, is a flow chart of the LPSW instruction. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and a storage request for the second operand has been issued per D.

A test for a specification check is made at the beginning of the operation. The operand address for the LPSW must have 0's in its three low-order bit positions to designate a double word, and the CPU must be in the supervisor mode; otherwise, a specification interruption results. If there is no specification interruption, the I-Fetch-invalid-address trigger is now reset. This trigger is set if an invalid address was detected during I-Fetch. For an understanding of I-Fetch-invalid-address detection, refer to paragraph 3.2.6.5.2. At this time, a delay of one clock cycle occurs, allowing the data requested during I-Fetch (new PSW) to become present at the SDBO. When the data is at the SDBO, it is gated to ST. The contents of T are then transferred to PAL(32-63), from where the instruction address of the new PSW is transferred to the IC. A 3-cycle storage request is then issued per the IC. The instruction-memory-fetch trigger is set and will be used for invalid instruction address detection.

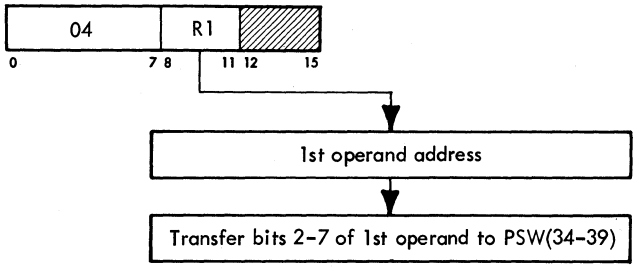
At this point, S(0-15) and T(34-39) are transferred to PSW(0-15) and PSW(34-39), respectively. S(0-15) comprises the new system mask [S(0-7)], key [S(8-11)], and program state [S(12-15)]. T(34-39) makes up the CC [T(34,35)] and the instruction address [T(36-39)]. The address-store-compare and execute triggers are reset. These triggers are set if the LPSW is the subject instruction of an Execute instruction. Since the next instruction to be performed is determined by the new PSW being loaded, the triggers must be reset to prevent the instruction normally following the Execute instruction from being performed after the LPSW. T(40-63) is then transferred to D, and the IC is updated by 8 to select the double word from main storage which follows the instruction addressed by the new PSW instruction address. The interrupt triggers are now reset. These triggers are set only if the IPL is in process. By this time, the data requested per the new PSW instruction address is available at the SDBO and can be gated to Q. From Q, the correct halfword is transferred to R per D(21,22). A 3-cycle storage request is then issued per the IC if D(21,22) equals 11. If D(21,22) equals 11, the halfword transferred to R from Q was located in the last halfword portion of Q. Therefore, another double word from main storage must be obtained to refill Q. The contents of D are updated by 8 and transferred to the IC to select the next double word.

D(21,22) is now tested for a value of 11. If it equals 11, the IC is updated by 8 and replaced in the IC to address the double word which sequentially follows the double word requested during the operation. The data requested during the operation should now be present at the SDBO and is gated to Q. A normal end-op cycle is taken to complete the operation.

If, when tested, D(21,22) did not equal 11, a branch end-op cycle takes place. This action allows the instruction format to be decoded off the SDBO since the data placed in R is not stable.

3.14.2 SET PROGRAM MASK, SPM (04)

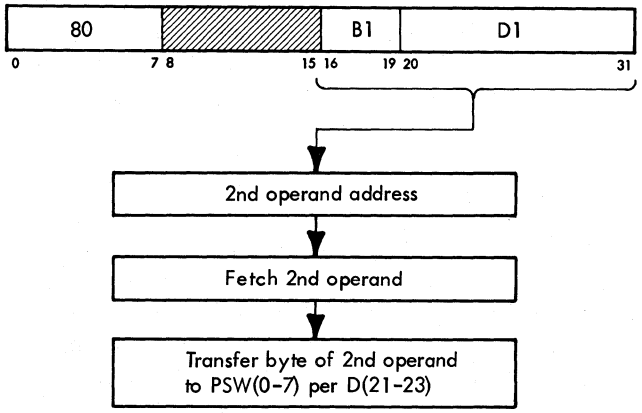
- Transfers bits 2-7 of 1st operand to PSW(34-39).
- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in AB and D.
  2nd operand is in ST (not used).
- Bits 2 and 3 of 1st operand are used as new CC.
- Bits 4-7 of 1st operand are used as new program mask.

3.14.3 SET SYSTEM MASK, SSM (80)

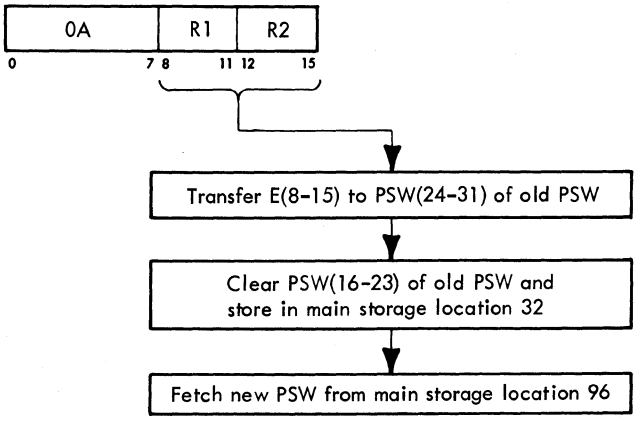
- Transfers byte of 2nd operand to PSW(0-7), replacing the system mask.
- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand (not used) is in ST.
  2nd operand address is in AB and D.
- Byte to be used as new system mask is selected per D(21-23).
- Byte is transferred to ST and PSW via SA.
- Delay 1 cycle to allow new system mask to become effective.
- Figure 6073, FEDM.

3.14.4 SUPERVISOR CALL, SVC (0A)

- Causes a supervisor call interruption; R1 and R2 fields of instruction provide interruption code to be stored in PSW.
- RR format:

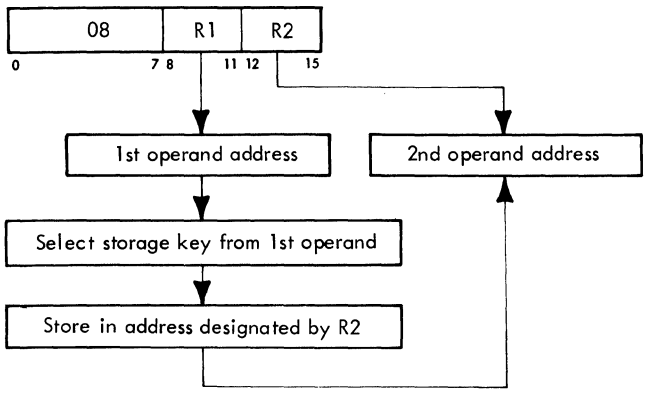


- Conditions at start of execution:
  Instruction is in E [E(8-15) is interruption code)].
  1st operand (not used) is in AB and D.
  2nd operand (not used) is in ST.
- Instruction sets supervisor-interrupt trigger and takes an end-op cycle.
- End-op cycle performs supervisor interruption.

3.14.5 SET STORAGE KEY, SSK (08)

- Sets storage key (designated by 1st operand) for block of storage bytes addressed by 2nd operand.

- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in AB and D.
  2nd operand is in ST.

- Bits 8-20 of 2nd operand designate block of storage bytes.

- Bits 21-27 of 2nd operand are ignored.

- Bits 28-31 of 2nd operand must be 0.

- Bits 24-28 of 1st operand are new storage key.

The Set Storage Key (SSK) instruction, which has an RR format with an op code of 08, sets the key of the storage block addressed by the second operand according to the key in the register designated by the first operand. Bits 8 through 20 of the second operand address a block of 2048 storage bytes. During the SSK instruction, bits 21 through 27, which address double words in the storage block, are ignored. Bits 28 through 31 of the second operand must be zero, or a specification interruption occurs. The new storage key is obtained from bits 24 through 28 of the first operand. The remaining bits of the operand are ignored.

Figure 6074, FEDM, is a flow chart of the SSK instruction. At the start of execution, the instruction is in E, the first operand is in AB and D, and the second operand is in ST. The ABC and STC are set to 111 at the beginning of the operation. This action allows selection of the correct byte containing the new storage key from AB. The contents of S are transferred to the parallel adder, where 8 is subtracted from it. The result is placed in D. A test for a specification check is then made. If PAL(28-31) contains a value other than zero, a specification interruption occurs.

Assuming no specification check, the byte containing the new storage key [B(56-63)] is transferred to F per the ABC. A 4-cycle storage request is then issued per D to set the storage key for the even or odd storage addresses. The set-key trigger for the even or odd storage (depending on the address) is then set, and F(0-4) is transferred to the storage-key bus. Mark triggers 0 through 7 are set, and the storage key is written into the address specified. The contents of D are then increased by 8 to address the other storage addresses. Assume that the last storage unit addressed was even. This time, the 4-cycle storage request is issued for the odd storage. The set-key trigger for the odd storage is set, and F(0-4) is transferred to the storage-key bus. Mark triggers 0 through 7 are set, and the storage key is written into the address specified. D is again updated by 8 to address the next storage unit. If a large-capacity storage is being used in the system, four storage units will have to be addressed. The SSK operation must then loop back and again send the storage key to these units.

STAT D is then examined to see whether it is set; at this time, STAT D is not set, indicating that only one odd and one even storage have had their storage keys changed. If the system has a large-capacity storage, another odd and even storage unit exists and their storage keys must be changed. Therefore, STAT D is set and a 4-cycle storage request is issued. From this point on, the operation is the same as previously discussed (Figure 6074, FEDM).

3.14.6 INSERT STORAGE KEY, ISK (09)

- Inserts into 1st operand and stores in LS a storage key (addressed by 2nd operand) of block of bytes in main storage.

- Bits 8-20 of 2nd operand address block of bytes in main storage.

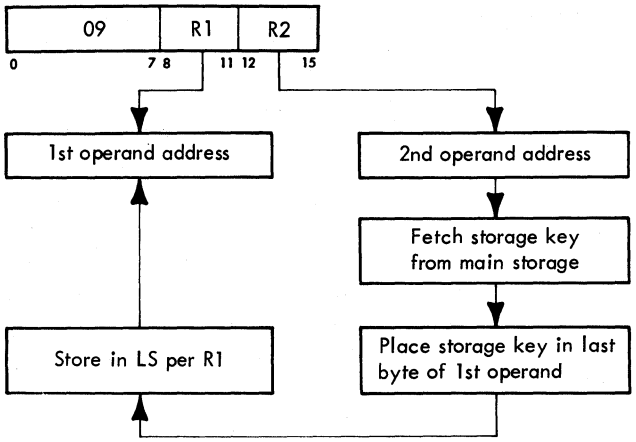
- Bits 0-7 and 21-27 of 2nd operand are ignored.

- Bits 28-31 of 2nd operand must be 0's or specification interruption occurs.

- Storage key is inserted in bits 24-28 of 1st operand.

- Bits 29-31 of 1st operand are cleared.

- RR format:



- Conditions at start of execution:
  Instruction is in E.
  1st operand is in AB and D.
  2nd operand is in ST.

The Insert Storage Key (ISK) instruction, which has an RR format with an op code of 09, inserts the storage key addressed by the second operand into bits 24 through 28 of the first operand. Bits 8 through 20 of the second operand address a block of 2048 bytes in main storage. Bits 0 through 7 and 21 through 27 of the second operand are ignored, whereas bits 28 through 31 must be 0's or a specification interruption occurs. The 5-bit storage key is set into bits 24 through 28 of the first operand; bits 29 through 31 are cleared.

Figure 6075, FEDM, is a flow chart of the ISK instruction. At the start of execution, the instruction is in E, the first operand is in AB and D, and the second operand is in ST. The STC is set to 011. The contents of S (second operand) are then transferred to A. (Later in the operation, the value in A will be reduced and placed in D to address storage.) A test for a specification check is now made. If PAL(28-31) contains a value other than zero, a specification interruption occurs.

Assuming no specification check, the contents of B are transferred to T. This action places the first operand, which is to hold the storage key when retrieved from storage, in T. The STC is set to 111 by placing a 1 in STC(0). This setting allows the last byte of ST to be gated to the serial adder, where it will be logically OR'ed with the storage key. The data in A is now transferred to PAB(32-63). Then 8 is subtracted from PAB, and the result is transferred to D. The contents of SAL are transferred to ST per the STC and to F(0-7). Since SAL is cleared at this time, the last bytes in ST and in F(0-7) are cleared.

The insert-key trigger is set to allow the storage key to be gated into F(0-4) when a storage request is issued. After the insert-key trigger is set, a 3-cycle storage request is issued per D for the storage key. The contents of D are transferred to the parallel adder, 8 is added, and the result is transferred back to D. This address selects the odd or even storage unit, depending on the address of the last storage unit. That is, if the last storage unit address was odd, the next storage address will be even; if it was even, the next storage address will be odd. The contents of F are then transferred to the serial adder and logically OR'ed with the contents of the last byte of ST (selected per the STC). The result is transferred to ST per the STC.

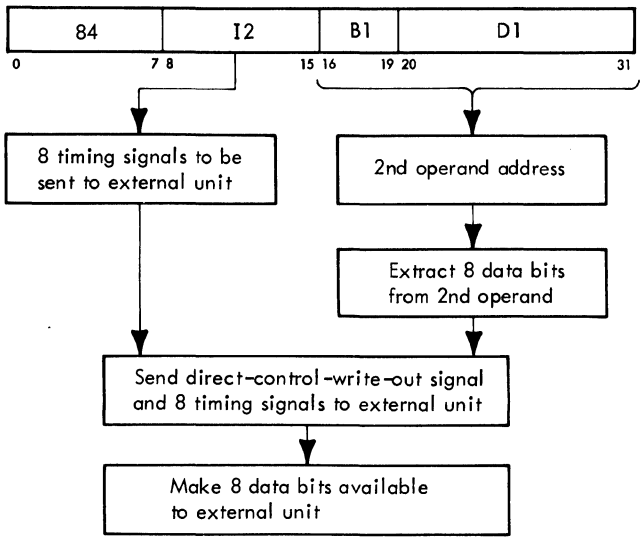
The insert-key trigger is again set. This time it is used with the storage address that was just placed in D. When the 3-cycle storage request is issued for the storage key, the insert-key trigger allows the key to be gated into F(0-4). The contents of D are updated by 8, and the result is placed in D. If the Large Capacity Storage feature is incorporated in the system, this new address allows selection of the correct storage unit to obtain the next storage key. If the Large Capacity Storage feature is not used, the address is ignored. At this time, the storage key just placed in F is transferred to the serial adder where it is logically OR'ed with the contents of the last byte in ST. Because there is a possibility that the storage key may have been changed by some other operation in process, the data in ST must be logically OR'ed with the last storage key to obtain the latest data available. This result is then placed in the last byte of ST per the STC. STAT D is tested. At this time, STAT D is not set and the insert-key trigger is set. After the trigger is set, STAT D is set. From this point on, the operation is identical with that previously described for the first two storage units (Figure 6075, FEDM).

When STAT D is again tested, it will be set. The data in T is therefore transferred to LS per E(8-11). This action places the storage key in LS. An end-op cycle completes the operation.

3.14.7 WRITE DIRECT, WRD (84)

- Sends 8 data bits and 9 signal-out timing signals to an external unit.

- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address is in D.

- Nine signal-out timing signals are available to external unit for 0.5 to 1.0 μs.

- Eight of the signal-out signals perform predetermined functions in external unit.

- Ninth signal-out timing signal (direct-control-write-out signal) alerts external unit that data is available.

- Eight data bits remain present until another WRD instruction is issued.

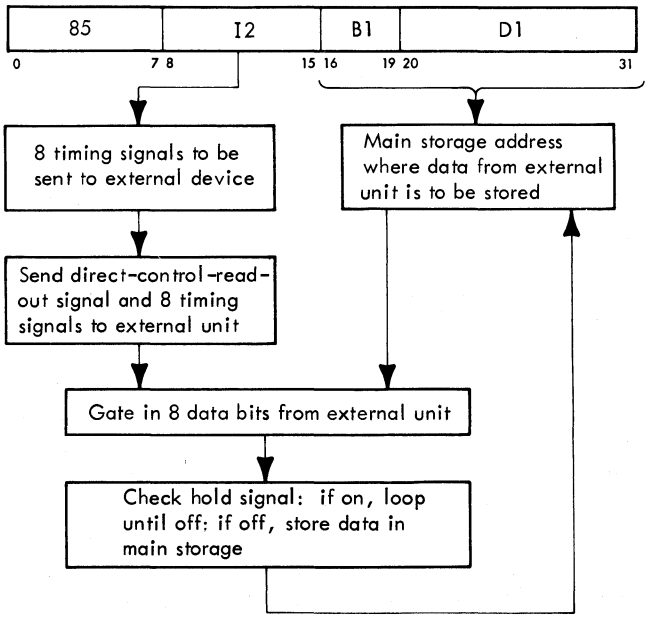
The Write Direct (WRD) instruction, which has an SI format with an op code of 84, sends eight data bits as static signals to an external device and makes nine signal-out timing signals available for a period of 0.5 to 1.0 μs. The eight data bits remain until the next WRD instruction is executed. No parity is presented with the eight data bits when sent to the external unit.

Figure 6076, FEDM, is a flow chart of the WRD instruction. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and a storage request for the second operand has been issued. (The first operand is not applicable in this instruction.) A privileged operation check is first made. If the privileged-operation condition exists, a privileged-operation interruption occurs; if not, D(21-23) is transferred to the STC. The STC is used later to select the byte of the second operand that contains the eight data bits to be sent to the external unit. At this point, the second operand is gated from the SDBO to ST. Next, the timing-gate trigger is set. This trigger allows sending of the nine signal-out timing signals to the external unit. Eight timing signals, E(8-15), are now sent to the timing bus-out, and the direct-control-write-out signal (ninth signal-out timing signal) is sent to the external unit. The eight timing signals (bits) from E perform certain predetermined functions in the external unit; the direct-control-write-out signal alerts the external unit that data is available. These nine signal-out timing signals are present for a period of 0.5 to 1.0 μs. A byte of ST (the eight data bits) is then selected per the STC and transferred to G. From G, the data is transferred to the external unit when needed. This data remains in G until another WRD instruction is issued and refills G. After the delay of 0.5 to 1.0 μs has elapsed, the timing-gate trigger is reset and an end-op cycle is taken to complete the operation.

3.14.8 READ DIRECT, RDD (85)

- Stores in main storage location designated by 2nd operand address eight data bits accepted from an external unit, if hold signal is not present.

- SI format:



- Nine signal-out timing signals are available to external unit for 0.5 to 1.0 μs.

- Eight data bits are accepted from external unit and stored in main storage if hold signal is off.
- If hold signal is on and not removed, CPU does not complete instruction.
- Excessive duration of instruction may result in incomplete updating of CPU timer.
- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address is in D.

The Read Direct (RDD) instruction, which has an SI format and an op code of 85, stores, in main storage, eight data bits that are accepted from an external unit if the hold signal is not present. Nine signal-out timing signals are made available to the external unit to perform predetermined functions in that unit. These signals are present for a period of 0.5 to 1.0μs. Eight data bits are accepted from a set of eight data-in lines when the hold signal is absent. The hold signal is sampled after the direct-control-read-out signal has been completed and should be absent for at least 0.5μs if data is to be accepted. No parity is accepted with the data signals, but a parity bit is generated as the data is placed in main storage. If the hold signal is not removed, the CPU does not complete the instruction. Excessive duration of this instruction may result in incomplete updating of the CPU timer.

Figure 6077, FEDM, is a flow chart of the RDD instruction. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and a storage request for the second operand is given. (The first operand is not applicable.) The second operand is never gated in and, therefore, is never used. A privileged-operation check is first made in the operation. If a privileged-operation error exists, a privileged-operation interruption is taken; if not, D(21-23) is transferred to the STC. The STC selects the byte in which the eight data bits are placed and also determines which mark triggers are to be set when storing the data in main storage. The timing-gate trigger is set. Eight timing signals, E(8-15), are then transferred to the timing bus-out and sent to an external unit along with the direct-control-read-out signal (ninth timing signal). An address-store-compare test is made on the second operand address to see whether the double word in which the data bits are to be stored is going to modify the double word presently being operated on. The eight data bits are then gated from the external unit to F. At this time, approximately 0.5 to 1.0μs has elapsed and the timing-gate trigger is reset, thus deactivating the nine signal-out timing signals. The mark triggers are then set per the STC.

The direct-control-hold latch is now tested. If it is set, the CPU goes into a loop, gating in the eight data bits from the external unit and setting the mark triggers according to the STC until the direct-control-hold latch is reset. If the hold signal is not removed, excessive duration of this instruction may result in incomplete updating of the CPU timer. If the direct-control-hold latch is not set, a 4-cycle storage request is issued per D. The contents of F are then transferred to ST per the STC, and from ST to the main storage per the mark triggers. An end-op cycle is taken to complete the operation.

Communication between CPU's can be provided by use of the RDD and WRD instructions. When CPU-to-CPU communication is desired, the data-out lines of one CPU are connected to the data-in lines of the other CPU. The signal-out timing signals of each CPU cause external interruptions in the other CPU. For example, assume there are two CPU's: CPU 1 and CPU 2. CPU 1 requests data by executing an RDD instruction. When CPU 1 determines it needs data, the signal-out timing signals are sent to CPU 2, causing an external interruption within CPU 2. After decoding the interruption code and determining the operation to be performed, CPU 2 starts a WRD operation, issues signal-out timing signals to CPU 1, and places data on the data-out lines. When CPU 1 receives the signal-out timing signals, an external interruption occurs within CPU 1, informing it that the data that was requested may now be read in. After the data has been read in, the operation is completed. The direct-control-write-out signal issued by CPU 2 during the operation may serve as the hold signal for the requesting CPU (CPU 1), temporarily inhibiting the RDD instruction when the timing signals are in transition.

3.14.9 DIAGNOSE (83)

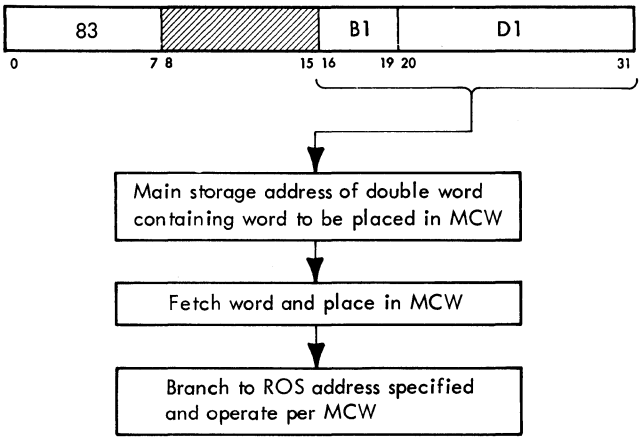
- Loads word designated by 2nd operand address into MCW and branches to address specified by MCW.
- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address is in D.

- If MCW(7) = 1 and MCW(6) = 0, channels are selected and MCW(0-5) applies to channel diagnostic functions.

- If MCW(7) = 1, CPU is selected and MCW(0-6) applies to CPU diagnostic functions.

- MCW(8-63) and the I2 field of the instruction are independent of MCW(7) and perform identically for either channel or CPU.

- SI format:



The Diagnose instruction, which is valid only in the supervisor mode, loads a word designated by the second operand address into the Maintenance Control Word (MCW) and branches to a location in ROS as designated by the MCW. The MCW causes certain predetermined functions to be performed according to its bit configuration. If MCW(7) is a 1 and MCW(6) is a 0, the MCW bits apply to the channels and MCW(0-5) assumes certain channel diagnostic functions. When a channel is selected by the MCW, the channel is disconnected from its control units and is connected to an internal interface simulator. The interface simulator consists of a 1-byte register and associated controls. This simulator allows the diagnostic program to test the channel regardless of the type of control units that may be attached and available. When used for channel selection, MCW(0-5) performs the following functions:

1. MCW(0-2): Selects the channel to be diagnosed according to the following bit configuration:

   000 - Select channel 0 (multiplexor channel).
   001 - Select channel 1.
   010 - Select channel 2.
   011 - Select channel 3.
   100 - Select channel 4.
   101 - Select channel 5.
   110 - Select channel 6.
   111 - No channel selected.

2. MCW(3) - Reverse Data Parity: Causes all bytes that are read from the interface simulator to have reversed parity. This action allows testing of the storage bus-in parity checker.

3. MCW(4) - Reverse Byte Counter Parity: Provides a means of testing the byte control check circuits.

4. MCW(5) - Suppress Storage Data Check: Prevents a storage data check from causing a channel data check and prevents a channel control check on a CCW fetch operation. Preventing the channel control check allows invalid CCW's to be brought into the channel to test sections of the channel check circuitry.

If MCW(7) is a 1, the MCW bits apply to the CPU and MCW(0-6) assumes certain CPU diagnostic functions. When used for CPU diagnose, MCW(0-6) performs the following functions:

1. MCW(0,1): These bits are spares and are not applicable when the CPU is selected.

2. MCW(2) - Reverse SA Full-Sum Parity: Reverses the parity bit in the full-sum latch of the serial adder, thus allowing testing of the parity-checking circuits.

3. MCW(3) - Reverse Mark Parity: Reverses the parity of the mark bits being sent from the BCU to main storage, thus allowing testing of the mark parity-checking circuits in main storage.

4. MCW(4,5) - Reverse SAR Parity 1 and 2: Causes the parity bits which are sent to the storage address register to be reversed as follows:

   00: No parity reversal
   01: Reverse low-order parity bit
   10: Reverse next-higher parity bit
   11: Reverse high-order parity bit

5. MCW(6) - Log on Count: Causes a log-out to main storage when the scan counter reaches a value of zero while performing fault-locating tests. At the conclusion of the logging operation, the CPU interrupts to the machine-check interruption location.

MCW(8-63) and the I2 field of the instruction are independent of MCW(7) and are used identically for both the channel and CPU Diagnose as follows:

1. MCW(8-19) - ROS Starting Address: When the Diagnose instruction has completed its execution phase, these address bits are placed in ROSAR and the operation branches to this location. The address placed in ROSAR can specify any location in ROS. The three locations most commonly used are (1) end-op, (2) enter FLT sequence, and (3) enter PADD full-sum sequence.

2. MCW(20) - This bit is a spare bit and is not used.

3. MCW(21-31) - Count Field: Specifies the number of cycles (200 ns) that are to occur before either a diagnostic-stop signal is sent to the channel selected or the CPU enters a log-out routine. The maximum value that can be set in this field is 2047. The value is placed in the scan counter, which is decremented by 1 after every cycle. The counter begins decrementing the first cycle after completion of the Diagnose instruction.

4. MCW(32-63) - These bits are spares and are not used.

5. I2 field of Diagnose instruction:

   a. Bit 8 - Disable Interleave: Disables interleaving and causes consecutive addresses to be in the same storage unit.

   b. Bit 9 - Disable Interleaving and Reverse Storage Address: Disables interleaving and reverses the high- and low-order halves of storage on the 2365 Storage Unit, the high- and low-order halves of storage in each unit are reversed.

   c. Bit 10 - Diagnose FLT: Allows FLT tests to be executed under control of the Diagnose instruction. During the time the FLT's are being executed, special CPU functions are generated, and storage requests and clock are inhibited.

   d. Bit 11 - Spare.

   e. Bit 12 - Permanent Spare.

   f. Bit 13 - Set Extended PSW Mode: Sets the extended-PSW-mode trigger when the channel-controller feature is present.

   g. Bit 14 - Set Emulator Mode: Allows the CPU to enter the emulator mode.

   h. Bit 15 - Spare.

Figure 6078, FEDM, is a flow chart of the Diagnose instruction. This instruction has an SI format with an op code of 83. At the start of execution, the first 16 bits of the instruction are in E, the second operand address is in D, and a storage request for the second operand has been given (the first operand is not applicable). The first operation to occur is to test for a specification check. If a specification check is present, a specification interruption is taken. If not, a test for an address-store-compare condition is made. If PAL(40-63) equals zero, then the address-store-compare trigger is set. The contents of D are transferred to the parallel adder and updated by 8. The result is then transferred back to D. This action places the next main storage double-word address in D. The scan-mode trigger is set, thus placing the D, F, and G fields of the ROS sense latches in the scan mode. The double word from main storage which was requested during I-Fetch is now present at the SDBO. SDBO(0-31) is transferred to T and SDBO(0-63) is transferred to AB. This data will be placed in the MCW.

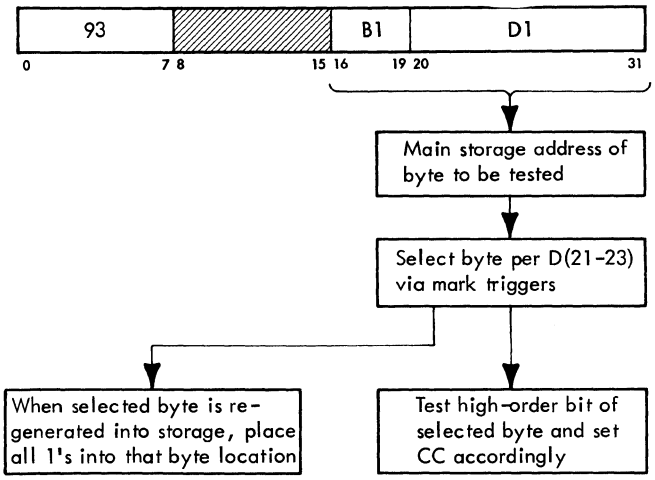
T(32-39) is now placed in MCW(0-7). These bits allow the appropriate diagnostic functions to be performed at the end of the execution phase of the Diagnose instruction. T(54-63) is transferred to the scan counter [MCW(21-31)]. As previously stated, the scan counter specifies the number of cycles that occur before a diagnostic-stop signal is sent to the channel selected or a log-out signal is sent to the CPU. The counter is decremented by 1 for every cycle following the completion of the Diagnose instruction. At this point, the scan-counter-control trigger is set. Then, T(40-51) [MCW(21-31)] is transferred to ROSAR to select the next ROS word to be executed. The scan-mode trigger is reset, and the operation is turned over to the next ROS word. Operations continue stepping through ROS until the scan counter reaches a value of zero. At this time, the operation issues either a diagnostic-stop signal if a channel is selected or a log-out signal if the CPU is selected to complete the Diagnose instruction.

3.14.10 TEST AND SET, TS (93)

- Selects a byte from double word addressed by B1 + D1 and tests high-order bit of byte setting CC according to value of that bit. When double word

is regenerated into storage, selected byte is changed to all 1's.

- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand (not used) is in ST.
  2nd operand address is in AB and D.

- Selects proper byte per D(21-23) via mark triggers.

- Tests high-order bit of selected byte; if equal to 1, sets CC to 1; if equal to 0, sets CC to 0.

- Causes 1's to be regenerated into storage in place of selected byte when selected double word is regenerated into storage.

- Figure 6079, FEDM.

This section describes an I/O system, its operation and system states, the condition codes derived from the system states, the I/O instruction format, and the interruptions that may occur during the execution of I/O instructions. Following this description is an analysis of the I/O instructions.

3.15 INTRODUCTION

Because the I/O instructions are dependent upon the I/O system attached to the CPU, a thorough understanding of the I/O system is required. The following paragraphs give a basic description of the I/O system attached to the 2065. For a detailed analysis of each I/O unit, refer to the applicable FEMI.

3.15.1 I/O SYSTEM

The transfer of information to and from main storage (other than to or from the CPU via the direct control path) is referred to as an "I/O operation." And I/O operation involves the use of an I/O device, a control unit to control the I/O device, and a channel which is used as a means of attaching the control unit to the CPU. Figure 3-21 illustrates the I/O system applicable to the System/360, Model 65. The CPU controls up to six selector channels and one multiplexor channel. These channels, in turn, control up to eight control units and up to 256 I/O devices.

3.15.1.1 Channels

- Direct flow of information between I/O devices and main storage.
- Provide standard interface for connecting different I/O devices to CPU and main storage.
- Contain all common facilities for control of I/O operations.
- Channel facilities required to sustain an I/O operation are called "subchannels."

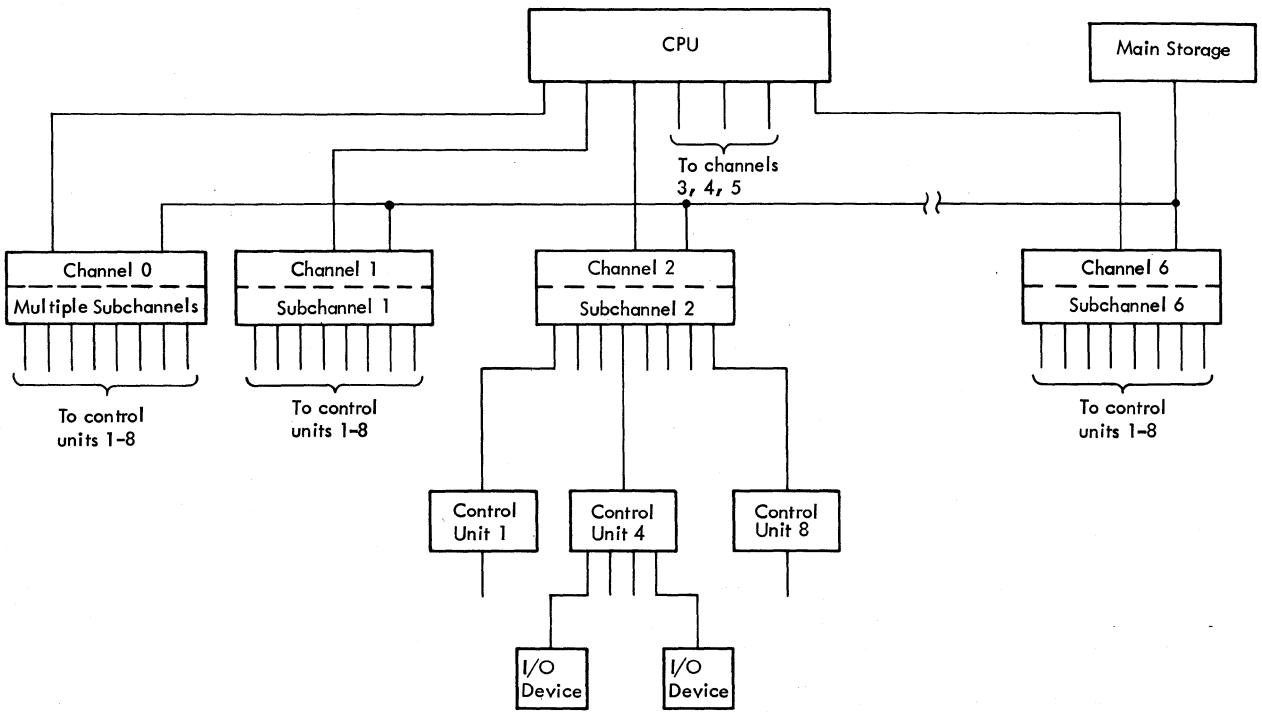


FIGURE 3-21. BASIC I/O SYSTEM

- Only 1 subchannel for selector channels.
- Multiple subchannels for multiplexor channels.

The channel directs the flow of information between the I/O devices and main storage. It relieves the CPU of the task of communicating directly with the I/O devices and permits data processing to proceed concurrently with I/O operations.

The channel provides a standard interface for connecting different types of I/O devices to the CPU and to main storage. It accepts control information from the CPU and changes it into a sequence of signals acceptable to a control unit. After the operation with the device has been initiated, the channel assembles or disassembles data and synchronizes the transfer of data bytes over the interface with main-storage cycles. When an I/O device provides signals that should be brought to the attention of the program, the channel again converts the signals to a format compatible with that used in the CPU.

The channel contains all the common facilities for the control of I/O operations. When these facilities are provided in the form of separate, autonomous equipment designed specifically to control I/O devices, I/O operations are completely overlapped with the activity in the CPU. The only main-storage cycles required during I/O operations in such channels are those needed to transfer data and control information to or from the final locations in main storage. These cycles do not interfere with the CPU program, except when the CPU and the channel simultaneously attempt to refer to the same main storage.

The channel facilities required for sustaining a single I/O operation are called "subchannels." The subchannel consists of a channel storage used for recording addresses, counts, and any status and control data associated with the I/O operation. If a channel contains more than one subchannel, it is referred to as a "multiplexor channel." If a channel contains only one subchannel and operates in the burst mode, it is referred to as a "selector channel." The 2065 uses both a multiplexor channel (channel address 0) and selector channels (channel addresses 1 through 6). When the selector channels are operating, a device monopolizes the channel and stays logically connected to the channel for the transmission of an information burst. The burst can consist of a number of bytes, a block of data or a sequence of blocks with associated control signals.

When the multiplexor channel is operating, the facilities of the channel may be shared by a number of concurrent I/O operations. The multiplex mode causes all I/O operations to be split into short intervals of time during which only a segment of information is transferred over the interface. The intervals associated with different operations are intermixed in response to demands from the I/O devices. The channel controls are occupied with any one operation only for the time required to transfer a segment of information. The segment of information can consist of a single byte of data, a few bytes of data, or a control sequence such as initiation of a new operation or a status report from the device.

3.15.1.2 Control Units

- Provide logical capability necessary to operate and control I/O devices.
- May be housed separately or may be physically and logically integral with I/O devices.
- Accept signals from channel and control timing of data transfer to channel.

The control unit provides the logical capability necessary to operate and control an I/O device and adapts the characteristics of each device to the standard form of control provided by the channel. A control unit may be housed separately or it may be physically and logically integral with the I/O device. The control unit accepts control signals from the channel, controls the timing of data transfer to the channel, and provides indications of device status.

Except for the signal used to establish priority among control units, all communications to and from the channel occur over a common bus, and any bus signals provided by the channel are available to all control units. At any instant, only one control unit is logically connected to the channel. Selection of a control unit for communication with the channel is controlled by a signal that passes serially through all control units and permits each control unit to respond sequentially to the signals provided by the channel. A control unit remains logically connected to the channel until it has transferred the information it needs or has, or until the channel signals it to disconnect.

The I/O device attached to the control unit may be designed to perform only certain limited operations, such as moving the recording medium and recording data. To accomplish these functions, the

device needs detailed signal sequences peculiar to the type of device. The control unit decodes the command received from the channel and then provides, for the particular I/O device, the signal sequence required for the operation.

3.15.1.3 I/O Devices

The I/O devices provide external storage and a means of communication between data-processing systems or between a system and the surrounding environment. I/O devices may be accessible from one or more channels. Devices accessible from one channel normally are attached to only one control unit. A device can be made accessible to two or more channels by switching it between two or more control units, each attached to a different channel, or by switching the control unit between two or more channels. The System/360, Model 65, has up to 256 directly addressable devices which can be attached to one channel.

3.15.2 I/O SYSTEM OPERATIONS

- I/O operations initiated and controlled by instructions, commands, and orders.

- Instructions are decoded by CPU and consist of:
  - Start I/O.
  - Test I/O.
  - Halt I/O.
  - Test Channel.

- Commands are decoded by channel and consist of:
  - Read.
  - Write.
  - Read backward.
  - Control.
  - Sense.
  - Transfer in channel.

- Orders are functions peculiar to I/O devices.

I/O operations are initiated and controlled by instructions, commands, and orders. Instructions are decoded by the CPU and are part of the CPU program. Commands are decoded and executed by the channels, and initiate I/O operations such as reading and writing. Both instructions and commands are fetched from main storage and are common to I/O devices of all types. The 2065 has four I/O instructions: Start I/O, Test I/O, Halt I/O, and Test Channel. The commands associated with the system are read, write, read backward, control, sense, and transfer in channel.

Functions peculiar to an I/O device, such as rewinding tape or spacing a line on the printer, are specified by orders. Orders are decoded and executed by I/O devices. The execution of orders is initiated by a control command, and the associated control information is transferred to the device, as data, during the control operation or is specified in the modifier bits of the command code.

The CPU program initiates I/O operations with the Start I/O instruction. This instruction identifies the I/O device and causes the channel to fetch a channel address word (CAW) from a fixed location in main storage. The CAW contains a protection key and designates the location in main storage from which the channel subsequently fetches the first channel command word (CCW). The CCW specifies the command to be executed and the storage area, if any, to be used.

If the channel is not operating and if the subchannel is not busy, the channel attempts to select the addressed I/O device by sending the address to all attached control units. A control unit that recognizes the address connects itself logically to the channel and responds to the selection by returning the address. The channel subsequently sends the command code over the interface, and the device responds with a status byte indicating whether it can execute the command.

At this time, the execution of the Start I/O instruction is terminated. The results of the attempt to initiate the execution of the command are indicated by setting the CC in the PSW and, under certain conditions, by storing a portion of the channel status word (CSW).

An I/O operation may involve the transfer of data to one main storage area or to a number of main storage areas. In the latter case, a chain of CCW's is used where each CCW designates an area in main storage for the original operation. The program can be notified of the chaining progress by specifying that the channel interrupt the program upon fetching a new CCW.

Termination of the I/O operation normally is indicated by two conditions: channel end and device end. The channel-end condition indicates that the control unit has received or provided all information associated with the operation and no longer

needs channel facilities. The device-end signal indicates that the I/O device has terminated the operation. The device-end condition can occur concurrently with the channel-end condition or later in the operation.

Operations that tie up the control unit after releasing the channel facilities may, under certain conditions, cause a third type of signal. This signal, called control-unit-end, may occur only after the channel-end signal and indicates that the control unit is available for initiation of another operation.

The conditions signalling the termination of an I/O operation can be brought to the attention of the CPU program by I/O interruptions or, when the channel is masked, by programmed interrogation of the I/O devices. In either case, these conditions cause storing of the CSW, which contains additional information concerning the execution of the operation. At the time the channel-end signal is generated, the channel provides an address and a count that indicates the extent of main storage used. Both the channel and the I/O device can provide indications of unusual conditions. The device-end and control-unit-end signals can be accompanied by error indications from the I/O device. For a more complete analysis of the I/O system operations, refer to the particular FEMI for the channel or I/O device concerned.

3.15.3 CONDITION CODES

The state of an I/O system depends upon the collective state of the channel, the control unit, and the I/O device. Each of these components of the I/O system can have up to four states in response to an I/O instruction from the CPU: (1) available, (2) interruption pending, (3) working, and (4) not operational. A channel, a control unit, or an I/O device that is available, that contains a pending interruption condition, or that is working is said to be "operational." A channel, a control unit, or an I/O device that is not in the system, that has power down, or that is in the test mode is said to be "not available."

The CC for an I/O instruction will vary according to the state of the I/O system and of the I/O instruction being performed. The CC is set in PSW(34, 35) to indicate whether the channel has performed the function specified by the I/O instruction. Paragraphs 3.15.3.1 through 3.15.3.6 discuss the state of the I/O system and the CC generated when each instruction is issued. Tables 3-28 through 3-32 list the results of attempting to execute an I/O instruction under working or busy, interruption, and not-available conditions in a channel, control unit, and device. An equipment configuration is assumed with more than one channel, more than one control unit per channel, and more than one device per control unit.

TABLE 3-28. CC FOR WORKING CHANNEL

Instruction	CC	CSW Stored	Comment
Start I/O to any control unit	2	No	No I/O operation started.
Test I/O to any control unit	2	No	Control unit not selected, status not transferred.
Halt I/O to any control unit	2	No	Halt I/O instruction causes a channel interrupt after halting current operation.
Test Channel	2	No	Indication given that channel is operating in burst mode.

TABLE 3-29. CC FOR INTERRUPTION PENDING IN CHANNEL

Instruction	CC	CSW Stored	Channel Interruption Cleared	Comment
Start I/O to any control unit	2	No	No	No I/O operation started.
Test I/O to control unit and device	1	Yes	Yes	Path cleared for new operation.
Test I/O to control unit and other devices	2	No	No	No address match, status not stored.
Test I/O to other control unit	2	No	No	No address match, status not stored.
Halt I/O to any control unit	0	No	No	Channel already not working.
Test Channel	1	No	No	Indication given that an interruption is available.

TABLE 3-30. CC FOR AVAILABLE CHANNEL, PENDING INTERRUPTION IN CONTROL UNIT

Instruction	CC	CSW Stored	Control Unit 1 Status Cleared	Comment
Start I/O to control unit and device	1	Status only (Notes 1, 7)	Yes	No I/O operation started, but path is now clear.
Start I/O to control unit and other devices (Note 2)	1	Status only (Notes 1, 7)	No	Status will probably be busy and status-modifier.
Start I/O to control unit and other device (Note 3)	0	No	No	I/O operation will be performed.
Start I/O to other control unit	0	No	No	I/O operation will be performed.
Test I/O to control unit and device	1	Yes	Yes	Path cleared for new operation.
Test I/O to control unit and other device (Note 2)	1	Yes (Note 7)	No	Status will probably be busy and status-modifier.
Test I/O to control unit and other device (Note 3)	0	No	No	Zero status on Test I/O instruction causes CC 0 and release signal to CPU.
Test I/O to other control unit	0	No	No	Same as above.
Halt I/O to control unit and device	1	All-zero status only (Note 1)	No	Halt I/O instruction does not request status from addressed unit.
Halt I/O to control unit and other device (Note 4)	1	All-zero status only (Note 1)	No	Same as above.
Halt I/O to control unit and other device (Note 5)	1	Busy, status modifier, and all-zero channel status only (Note 1)	No	Same as above except that a control-unit-busy selection sequence forces unit status to channel.
Halt I/O to other control unit	1	All-zero status only (Note 1)	No	Halt I/O instruction does not request status from addressed unit.
Test Channel	0	No	No	Note 6

Note 1 - "Status only" means that the CSW will store only the unit and channel status bytes (bits 32-47), leaving all other bits unchanged.

Note 2 - Assume that the outstanding status is the control unit type that makes the control unit appear busy. Examples are control unit end, channel end, device end with unit check, and device end with unit exception.

Note 3 - Assume that the outstanding status is the device type that does not make the control unit appear busy. Examples are device end only and attention only.

Note 4 - Assume that the control unit does not appear busy, or that the control unit is busy but must wait for a normal selection sequence to present nonzero status.

Note 5 - Assume that the control unit appears busy and responds with the control-unit-busy selection sequence.

Note 6 - This response is possible only for a short time, because the unit status immediately attempts to enter channel as a polling interruption, changing the test channel response to CC 1.

Note 7 - The busy bit will appear in the unit-status byte.

TABLE 3-31. CC FOR CHANNEL NOT AVAILABLE

Instruction	CC*	CSW Stored
Start I/O to any control unit	3	No
Test I/O to any control unit	3	No
Halt I/O to any control unit	3	No
Test Channel	3	No

* Channel does not generate the CC or release signal in any of these cases. The CPU generates the CC 3 and release signal internally as a result of the channel-available line to the CPU being functionally inactive.

TABLE 3-32. CC FOR AVAILABLE CHANNEL, CONTROL UNIT NOT AVAILABLE

Instruction	CC	CSW Stored	Comment
Start I/O to control unit	3	No	Channel fetches CAW and CCW before finding no-selection condition.
Start I/O to other control unit	0	No	Proceed with normal Start I/O operation.
Test I/O to control unit	3	No	Channel sends CC 3 and release signal immediately after select-in signal returns on the unit selection attempt.
Test I/O to other control unit	0	No	Zero status on Test I/O instruction causes CC 0, release signal, and no CSW store.
Halt I/O to control unit	3	No	Channel sends CC 3 and release signal immediately after select-in signal returns on the unit selection attempt.
Halt I/O to other control unit	1	All-zero status only (Note 1, Table 3-30)	Halt I/O instruction does not request status from addressed unit.
Test Channel	0	No	Indication is given that channel is available.

Note: Some control units, when the control unit meter is disabled, may provide a unit check status bit instead of appearing nonexistent to selection attempts. Responses to CPU instructions for these units would be as expected for a nonzero status condition.

3.15.3.1 Working Channel

Table 3-28 lists the CC placed in the PSW for each of the four I/O instructions when the channel being addressed is working. The channel appears busy to all CPU instructions except Halt I/O, which terminates the operation occurring in the channel. Conditions of the working channel do not affect the responses of other channels to new instructions.

3.15.3.2 Interruption Pending in Channel

Table 3-29 lists the CC's placed in the PSW for the four I/O instructions when an interruption is pending in the channel. The table is based on the assumption that a normal operation occurred for a Start I/O instruction in a channel, in the associated control unit, and in the I/O device, and that the data transfer, if any, has been completed with an interruption to store the CSW generated. The CPU in this example is masked for the channel and cannot respond to the channel interruption request. Any I/O instruction generated for other channels is not affected by the condition of the addressed channel. If a Test I/O instruction has not cleared the interruption, the CSW is not stored until the CPU unmasks the channel and accepts the interruption. When the CPU accepts the interruption, the interruption condition in the channel is cleared.

3.15.3.3 Available Channel, Pending Interruption in Control Unit

Table 3-30 lists the CC's placed in the PSW for the four I/O instructions when the channel is available and an interruption is pending in the control unit. The table is based on the assumption that the addressed channel has just become not-busy and has not yet begun polling. It is also assumed that associated control units have outstanding status from their devices, which has not yet been accepted by the channel. No other control units or devices are busy or working.

3.15.3.4 Channel Not Available

Table 3-31 lists the CC's placed in the PSW for the four I/O instructions when a channel is not available. One of the following conditions is assumed to be present in the addressed channel:

1. Channel disconnected from CPU interface.

2. Channel power down.

3. Channel in test mode.

4. Channel meter disabled.

3.15.3.5 Available Channel, Control Unit Not Available

Table 3-32 lists the CC's placed in the PSW for the four I/O instructions when a channel is available and the control unit is not available. It is assumed that the addressed channel is operative but not busy, and that one of the following conditions may be present in the control unit on the channel:

1. Control unit power down.

2. Control unit disconnected from the I/O interface and the interface cables jumpered.

3. Control unit under test.

4. Control unit meter disabled.

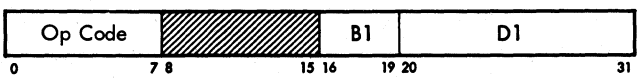
All other channels, control units, and devices in the system are operative but not busy.

3.15.3.6 Polling Interruption in Channel

The CC's placed in the PSW when a polling interruption in the channel occurs are identical with those listed in Table 3-30 for the Start I/O, Test I/O, and Halt I/O instructions. The assumption is that the control unit and device status has entered the channel and that the channel interruption request to the CPU is present but the CPU has not replied with a CPU interruption response. No other control units or devices have outstanding status. The Test Channel instruction CC changes from that listed in Table 3-30. This instruction sees the channel polling interruption as an interruption-available condition and provides a CC of 1 rather than of 0.

3.15.4 INSTRUCTION FORMAT

I/O instructions use the SI format:



Bits 8-15 are ignored. The base plus the displacement determines the channel and device address: bits 16-23 of the sum are the channel address, and bits 24-31 of the sum are the device address.

3.15.5 INTERRUPTIONS

The only interruption that may occur for an I/O instruction is the privileged-operation interruption. It occurs if the CPU is in any state other than the supervisor mode. The instruction is suppressed before the channel has been signalled to execute the instruction. The CSW, the CC in the PSW, and the state of the addressed channel and the I/O device remain unchanged. The interruption code in PSW(16-31) is 00000000 00000010. The instruction length code may be 1 or 2.

3.16 INSTRUCTION ANALYSIS

The four I/O instructions and their format, mnemonic code, op code, CC's, and interruptions are shown in Table 3-33. A functional analysis of each instruction is given in the following paragraphs.

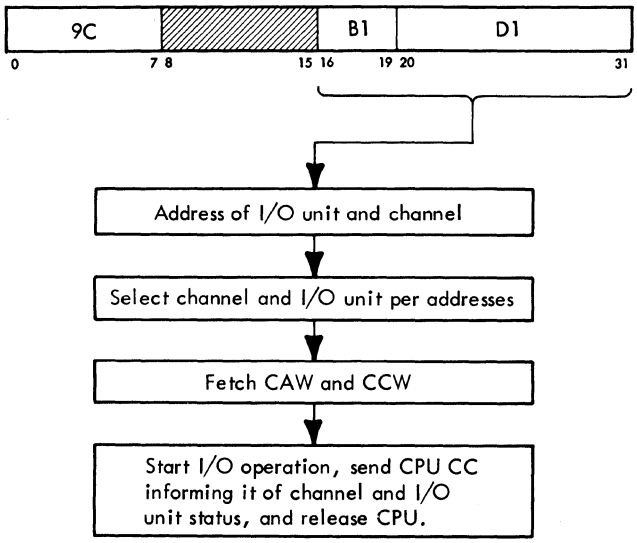
TABLE 3-33. I/O INSTRUCTIONS

Instruction	Format	Mnemonic Code	Op Code	Condition Codes	Interruption
Start I/O	SI	SIO	9C	0, 1, 2, 3	Privileged Operation
Test I/O	SI	TIO	9D	0, 1, 2, 3	Privileged Operation
Halt I/O	SI	HIO	9E	0, 1, 2, 3	Privileged Operation
Test Channel	SI	TCH	9F	0, 1, 2, 3	Privileged Operation

3.16.1 START I/O, SIO (9C)

- Selects specified I/O unit and initiates channel commands to that unit.
- D(8-15) is channel address.
- D(16-23) is I/O unit address.
- If available, channel selects CAW.
- CAW specifies CCW location.
- Channel stores status byte if errors in CAW or unit address.

- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address (address of channel and device) is in D.

- CC's specify status of channel and I/O unit.

- CC setting:
  I/O operation initiated and channel proceeding with its execution: CC = 0.
  CSW stored: CC = 1.
  Channel busy: CC = 2.
  Channel not operational: CC = 3.

The Start I/O (SIO) instruction, which has an SI format with an op code of 9C, selects a specified I/O unit and initiates a channel command to that unit. The following channel commands are associated with the SIO instruction: read, read backward, write, sense, and control. The effective operand address (base + displacement) determined during I-Fetch of the SIO instruction addresses the channel and I/O unit. Bits 21 through 23 of the effective operand address are decoded to select 1 of 7 channels; bits 24 through 31 are sent to the channel as an 8-bit I/O unit address, selecting the correct I/O unit.

Figure 6080, FEDM, is a flow chart of the SIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the second operand address is in D; the first operand is not applicable. Because this instruction is an I/O instruction, the address in D is the address of the channel and I/O unit and is not to be interpreted as a main storage address. Therefore, no data is requested from main storage. The SIO instruction can be executed only when the CPU is in the supervisor state. The first operation of the instruction, therefore, is to determine the state of the CPU. If the CPU is not in the supervisor state, a privileged-operation check occurs, causing a privileged-operation interruption. If the CPU is in the supervisor state, execution of SIO instruction begins by setting the timing-gate trigger. This trigger sends a select signal to the proper channel as determined by D(8-15). At this time, the unit address [D(16-23)] is also sent to the channel. If at this point the selected channel is busy or in the test mode, a CC of 2 or 3, respectively, is sent to the CPU. A release signal is also sent to the CPU, releasing it for execution of other instructions. If the channel is available, the unit address is gated to the unit address register in the channel. The channel then fetches the CAW from main storage address 72. The CAW specifies the address of the first CCW and the storage protection key for all the channel commands associated with the SIO instruction. If any errors are discovered in the CAW or the unit address, a status byte is stored in the channel and a CC of 1 is sent to the CPU. A release signal is also sent to the CPU, releasing it for execution of other instructions.

Two operations, fetching the CCW and selecting the I/O unit per the unit address, are now started simultaneously. Fetching of the CCW is initiated by issuing a storage request from the channel to main storage. When the BCU response is received by the channel, the channel places the command address on the storage address bus and waits for the advance pulse. The advance pulse is used to place the command information (command code, data address, flags, and counts) into the proper registers in the channel. The CCW-valid trigger is set, if there were no errors, to show that the CCW has been received. The CCW information is then checked for correct parity. During the storage operation, the command address was incremented by 1; the updated quantity is now gated back to the command address register in the channel.

The second operation, selection of the proper I/O unit, is started at the same time as fetching of the CCW. Selecting the proper I/O unit is accomplished by placing the unit address on the bus-out to the control unit and issuing an address-out signal followed approximately 400 ns later by a select-out signal. The control unit responds with an operation-in signal, which causes the channel to deactivate the address-out signal. When the control unit senses the deactivation of the address-out signal, it places

the address of the device selected on the bus-in and activates its address-in signal. The channel then compares the address it received from the control unit with the address it sent to the control unit to determine that proper selection has been made.

If the addresses are identical, the CCW-valid trigger is set, and no errors have been encountered, the operation continues. The CC is placed on the bus-out, and the command-out signal is sent to the control unit. The control unit responds with zero status if it can accept the command. The channel then sends a CC of 0 and a release signal to the CPU, releasing the CPU for further instruction execution. When the CPU receives the release signal, the timing-gate trigger is reset and an end-op cycle is taken, completing the operation.

If any errors occurred up to the point where the channel and control unit compare addresses or if the control unit responded with anything other than zero status, a hardware-generated test-I/O code would be placed on the bus-out to the control unit instead of the command code and the device status would be cleared. The channel would then disconnect from the device and request a storage cycle. When the BCU response is received, the status information, a CC of 1, is placed in the CSW, and a release signal is sent to the CPU. This action leaves the channel clear and ready to receive another instruction.

3.16.2 TEST I/O, TIO (9D)

- Clears interruption condition existing in addressed channel or associated I/O units and stores CSW in main storage location 64.

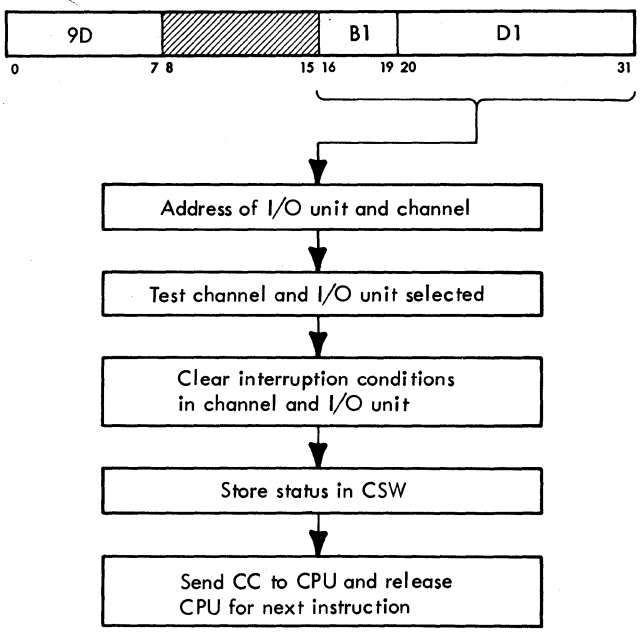
- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address (address of channel and device) is in D.

- D(8-15) is channel address.

- D(16-23) is I/O unit address.

- Sends CC to CPU, indicating status of channel or I/O unit, then releases CPU for next instruction.

- SI format:



- CC setting:
  Channel available: CC = 0.
  CSW stored: CC = 1.
  Channel busy: CC = 2.
  Channel unavailable: CC = 3.

The Test I/O (TIO) instruction, which has an SI format with an op code of 9D, clears interruption conditions existing in the addressed channel or its associated I/O units and stores a CSW in main storage location 64 whenever the I/O unit being tested has conditions for interruptions either within the channel or within the I/O unit itself. The CSW is also stored when the channel or I/O device detects an error during the execution of the TIO instruction. The status bits of the CSW identify the error conditions that occurred in the channel or I/O unit. The contents of the CSW pertain to the I/O device which is addressed by the effective operand address (base + displacement) determined during I-Fetch of the TIO instruction. Bits 21 through 23 of the effective operand address are decoded to select 1 of 6 channels; bits 24 through 31 are sent to the channel as an 8-bit I/O unit address to select the correct I/O unit.

Figure 6081, FEDM, is a flow chart of the TIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the second operand address is in D; the first operand is not applicable. Because this instruction is an I/O instruction, the address in D is the address of the channel and I/O unit and is not to be interpreted as a main storage address. Therefore, no data is requested from

main storage. The TIO instruction can be executed only when the CPU is in the supervisor state. Therefore, the first operation of the instruction is to determine the state of the CPU. If the CPU is not in the supervisor state, a privileged operation check occurs, causing a privileged operation interruption. If the CPU is in the supervisor state, execution of the TIO instruction begins by setting the timing-gate trigger. This trigger sends a select signal to the proper channel as determined by D(8-15). At this time, the unit address [D(16-23)] is sent to the channel. The channel, if not working, compares this unit address with the unit address it is holding in its unit address register. If they are equal, the interruption condition in the channel is stored in main storage location 64 (contains CSW) and a release signal is sent to the CPU, releasing if for execution of other instructions. If the addresses are not equal, a CC of 2 is sent to the CPU along with a release signal, allowing the CPU to start processing the next instruction. When the CPU receives the release signal, it resets the timing-gate trigger and takes an end-op cycle, completing the operation.

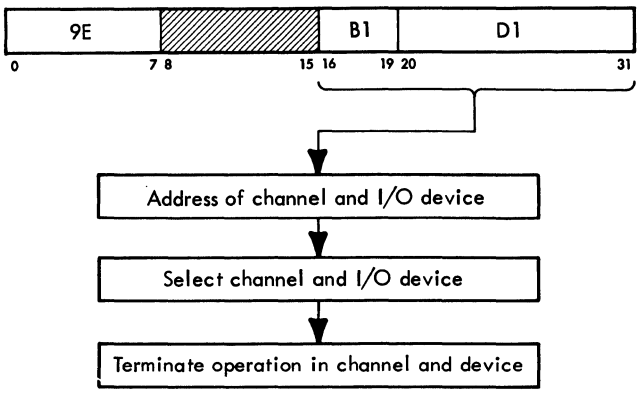
If the channel is available (working), the unit address sent from the CPU is placed in the unit address register in the channel to select the specified I/O unit in the same manner as the Start I/O instruction. (Refer to paragraph 3.16.1.) Because this is the TIO instruction, only status is required from the selected I/O unit. When the status of the I/O unit is received, a CSW is stored in main storage location 64 and a CC of 1 is sent to the CPU. A release signal is then sent to the CPU. When the CPU receives the release signal, the timing-gate trigger is reset and an end-op cycle is taken, completing the operation. If the status returned to the channel when the test I/O command is issued is all zeros, a CC of 0 is sent to the CPU. This value indicates that the I/O unit is available. The CPU is then released for further instruction execution.

3.16.3 HALT I/O, HIO (9E)

- Terminates current I/O operation at the selected channel. Status of channel and I/O device is sent to CPU via CC and status byte in CSW, respectively.

- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address (address of channel and device) is in D.

- SI format:



- Instruction is executed only in supervisor state.

- Channel to be halted is determined by D(8-15).

- I/O unit to be halted is determined by D(16-23).

- Channel sends status to CPU via CC and status byte.

- CC setting:
  Channel not working: CC = 0.
  CSW stored: CC = 1.
  Operation terminated: CC = 2.
  Not operational: CC = 3.

The Halt I/O (HIO) instruction, which has an SI format with an op code of 9E, terminates the current I/O operation at the selected channel. The status of the channel is sent, and the status of the I/O device may be sent, to the CPU through the CC and the CSW status byte, respectively. The effective operand address (base + displacement) determined during I-Fetch of the HIO instruction is used to address the channel and I/O unit. Bits 21 through 23 of the effective address select one of the channels available to the CPU, and bits 24 through 31 select the units available for that channel; the remaining bits are ignored.

Figure 6082, FEDM, is a flow chart of the HIO instruction. At the start of execution, the first 16 bits of the instruction are in E and the second operand address is in D; the first operand is not applicable. Because this instruction is an I/O instruction, the address in D is the address of the channel and I/O unit and is not to be interpreted as a main storage address. Therefore, no data is requested from main storage. The HIO instruction can be executed

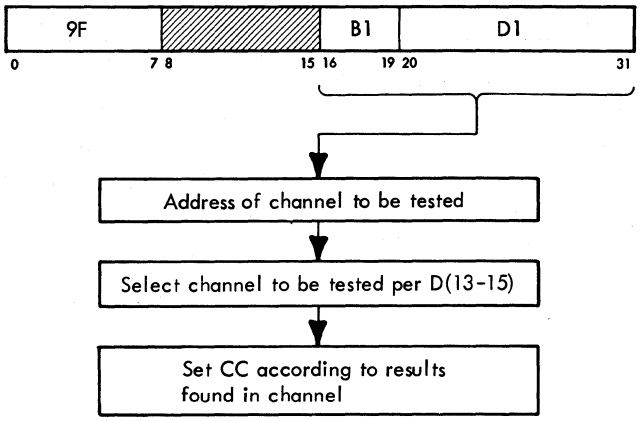
only when the CPU is in the supervisor state. Therefore, the first operation of the instruction is to determine the state of the CPU. If the CPU is not in the supervisor state, a privileged-operation check occurs, causing a privileged-operation interruption. If the CPU is in the supervisor state, execution of the HIO instruction begins by setting the timing-gate trigger. This trigger sends a select signal to the proper channel as determined by D(8-15). At this time, the unit address, D(16-23) is also gated to the channel.

When the channel receives the select signal, it is tested to see whether it is working. If not working, a CC of 0 is sent to the CPU. A release signal is then sent to the CPU, which resets the timing-gate trigger. An end-op cycle is taken, and the CPU is ready to process more instructions. If the channel is working, it is tested for possible interruptions pending. Assuming interruptions are pending, the I/O device addressed by the unit address is selected. The device response is determined. If a control-unit-busy response is found, the I/O device status is transferred to the CSW in main storage. The channel status bits of the CSW are set to 0's. If the I/O unit responds with an operational-in signal, an interface disconnect sequence is performed; when the sequence is completed, the CSW status bits are set to zero, and a CC of 1 is sent to the CPU. A release signal is also sent to the CPU to release the CPU for further instruction execution. If the I/O device responds with a select-in signal, a CC of 3 is sent to the CPU along with the release signal to release the CPU for further instruction execution.

Now assume no interruptions are pending. The address-out signal is activated, and the select-out signal is deactivated. The I/O unit operating on the I/O interface responds to this action by deactivating all in-tag signals and disconnecting itself from the channel. The channel then sets its interrupt-request trigger and issues a CC of 2 to the CPU. A release signal is sent to the CPU to reset the timing-gate trigger. An end-op cycle is taken to complete the operation.

3.16.4 TEST CHANNEL, TCH (9F)

- Tests state of channel addressed by bits 21-23 of effective operand address.
- SI format:



- Conditions at start of execution:
  1st 16 bits of instruction are in E.
  1st operand is not applicable.
  2nd operand address (address of channel and device) is in D.
- Instruction is executed only when CPU is in supervisor state.
- Channel to be tested is determined by D(8-15).
- D(16-23), which normally selects I/O unit, is ignored.
- State of channel is not affected by instruction.
- CC setting:
  Channel available: CC = 0.
  Interruption pending in channel: CC = 1.
  Channel operating in burst mode: CC = 2.
  Channel not operational: CC = 3.
- Figure 6083, FEDM.

The system control panel (Figure 3-22) contains the switches and indicators necessary to operate the system. The main functions of the panel are to reset the system, to store and display information in LS or main storage, in registers, and in the PSW, and to load program information.

3.17 MANUAL CONTROLS

- Hardware and ROS-controlled operations.
- CPU must be in stopped state before exiting to selected manual function.

Functionally, manual controls consist of special hardware and ROS-controlled microprograms. Special control logic and triggers control the selected manual operations. The CPU must be in the stopped state before most manual operations can be initiated. The force-address trigger (used during the pulse mode, system reset, and initial program-load operations) forces an overriding branch to a predetermined ROS microprogram. There are six normal methods of placing the CPU in the stopped state:

1. Perform power-on-reset operation.
2. Depress SYSTEM RESET pushbutton.
3. Depress STOP pushbutton.
4. Detect an address-compare condition when ADDRESS COMPARE STOP switch is in Stop position.
5. Initiate instruction-step-mode operation.
6. Initiate scan operation.

When the CPU is in the stopped state, it is in a ROS-controlled microprogram loop (stop loop) and the MANUAL indicator is on.

3.17.1 STOP LOOP

- ROS-controlled microprogram loop.
- Address of next instruction displayed in D.
- Stop and manual triggers control stop loop.
- Interruptions and I/O operations must be completed before entering stop loop.
- Six pushbuttons sampled by stop loop.

When the CPU is in the stopped state, a ROS-controlled microprogram, called the "stop loop," is continuously being executed. The stop loop is shown by the heavy lines in Figure 6084, FEDM.

When in the stop loop, no program instructions are executed, the interval timer (location 80 of main storage) is not stepped, and the time meters are stopped unless a channel is running. Also, the MANUAL indicator is on, and the stop loop determines the starting location of the next instruction to be executed and displays the results in D (contents of IC minus 8 or 16). To display the contents of D, set roller switch 1 to position 2; to display the contents of the IC, set roller switch 6 to position 3.

Two triggers control the stop loop: the stop trigger and the manual trigger. The stop trigger forces the CPU to the stop loop and is set by:

1. Depressing the STOP pushbutton.
2. Setting the instruction-step trigger in conjunction with an I-Fetch-reset micro-order or a reset-interrupt-triggers micro-order.
3. Detecting an address-compare condition when the ADDRESS COMPARE STOP switch is in the Stop position.
4. Being in scan operation.

At end op, the stop trigger's being set indicates an exceptional condition. A branch is forced to a count delay microprogram, provided all pending interruptions and I/O operations are completed. All interruptions (not masked off) and I/O operations are completed before entering the count delay and stop loop microprograms. The stop trigger forces in ROSAR the ROS address of a count delay microprogram. The count delay routine allows time to recognize that a pushbutton has been depressed before the stop loop is entered.

After the count delay microprogram is executed, the manual trigger is set by the set-stop-loop-trigger (1→ STOP LOOP) micro-order (Figure 6084, FEDM) and the stop loop is entered. The purpose of the manual trigger is to allow manual control operations only when the CPU is in the stopped state; however, SYSTEM RESET, CHECK RESET, and LOAD pushbuttons operate in all modes. The manual trigger also makes the RATE switch inoperative. Whenever the CPU is in the stop loop or in one of the microprogram routines entered from the stop loop, the CPU operates at normal machine speed regardless of the RATE switch position.

The stop trigger is not set during system reset or power-on reset; therefore, the stop loop is entered directly (Figure 6084, FEDM).

The stop loop continuously samples six pushbuttons:

1. STORE
2. DISPLAY
3. SET IC
4. START
5. ROS TRANSFER
6. PSW RESTART

When the stop loop senses that a pushbutton has been depressed, a branch is made to a new microprogram to perform the pushbutton function. The microprogram just executed either branches to the count delay microprogram and re-enters the stop loop or continues other preselected operations.

The microprograms executed for the six pushbuttons are shown in Figure 6084, FEDM, and discussed in paragraphs 3.17.9 through 3.17.14.

3.17.2 POWER ON RESET

Depressing the POWER ON pushbutton initiates the power-on sequence. After power is applied to the system, a system reset occurs. The power-on-system-reset signal forces an address in ROSAR, causes a system reset, and the power-on-reset microprogram is executed to clear all LS locations (Figure 6084, FEDM). Main storage locations remain unchanged. The stop loop is then entered.

3.17.3 SYSTEM RESET PUSHBUTTON

Activating the SYSTEM RESET pushbutton resets all channels, CPU controls, and check indicators to an initial state. Data flow registers remain unchanged. A system reset does not affect equipment in off-line channel operation. The SYSTEM RESET pushbutton is active in all modes of operation. After the system reset occurs, the stop loop is automatically entered.

Since a system reset may occur in the middle of an operation, the contents of the PSW and of result registers or storage locations are unpredictable.

A system reset micro-order is provided to reset the CPU in the scan mode. The micro-order does not reset the channels or storage units and does not force an ROS address. Depressing the LOAD pushbutton initiates a system reset before entering the initial program load (IPL) microprogram. When the LOAD pushbutton is depressed, the channels are not reset by the IPL line sent to the channels.

3.17.4 STOP PUSHBUTTON

The STOP pushbutton provides the ability to terminate machine operations while retaining the machine environment. The CPU proceeds to the end of the machine instruction being executed at the time the stop command is recognized. All I/O operations in process and all pending interruptions not masked off are completed before entering the stop loop. Depressing the STOP pushbutton sets the stop trigger. If there are no I/O operations in process or interruptions pending, the count delay microprogram is executed before entering the stop loop.

The operator may continue normal program operation by depressing the START pushbutton, or he may execute certain manual operations (e.g., instruction-step operation).

3.17.5 ADDRESS COMPARE STOP SWITCH

The ADDRESS COMPARE STOP switch provides the operator with a means of stopping at a predetermined address. To do so, he enters the address in the ADDRESS switches and places the ADDRESS COMPARE STOP switch in the Stop (down) position. When ADDRESS switches 2 through 20 match the address sent to the BCU, the address-compare-stop trigger is set. (Address comparison is performed on double-word boundaries.) The stop trigger is set, and the count delay microprogram is entered.

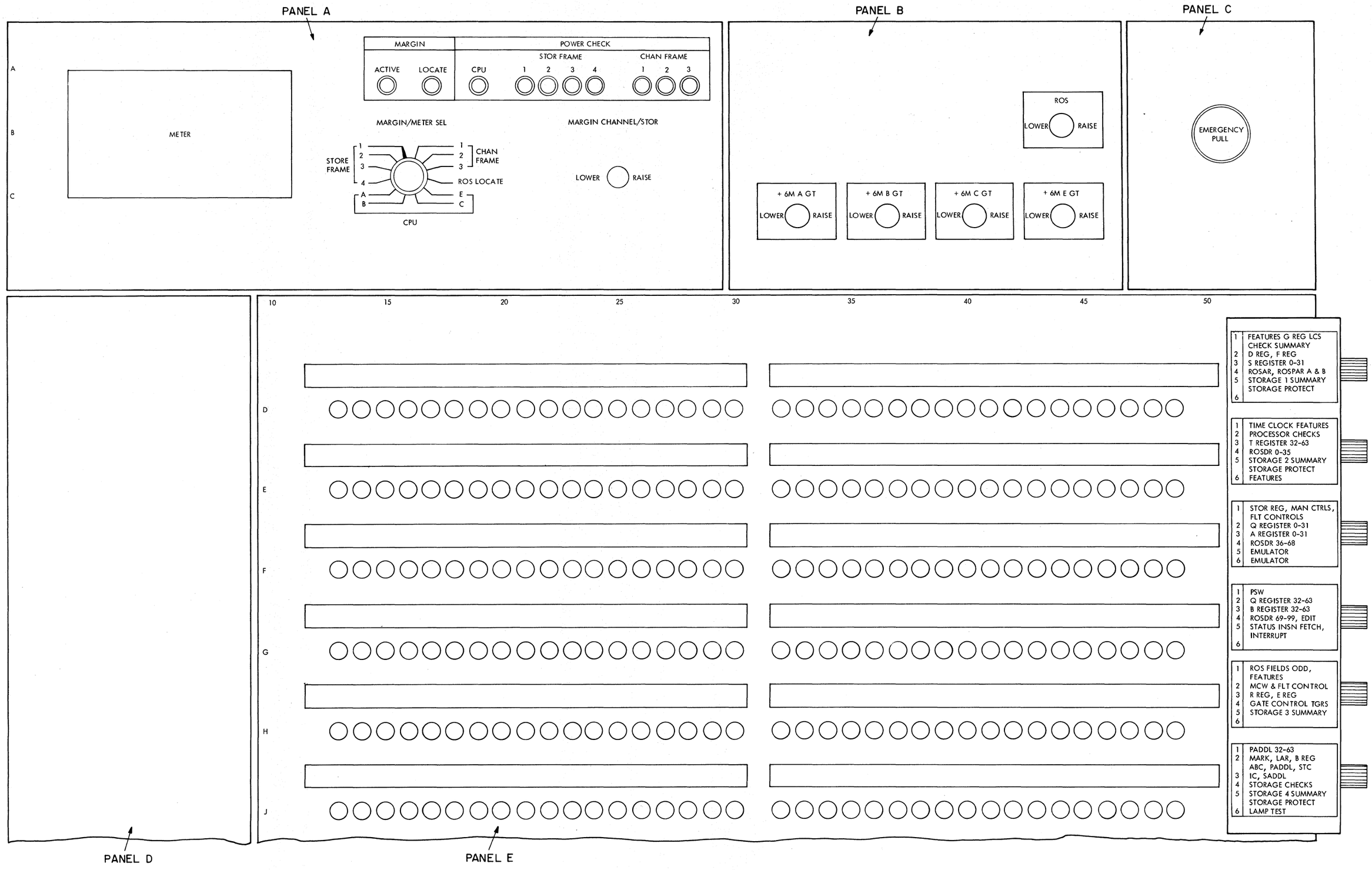


FIGURE 3-22. SYSTEM CONTROL PANEL
(SHEET 1 OF 2)

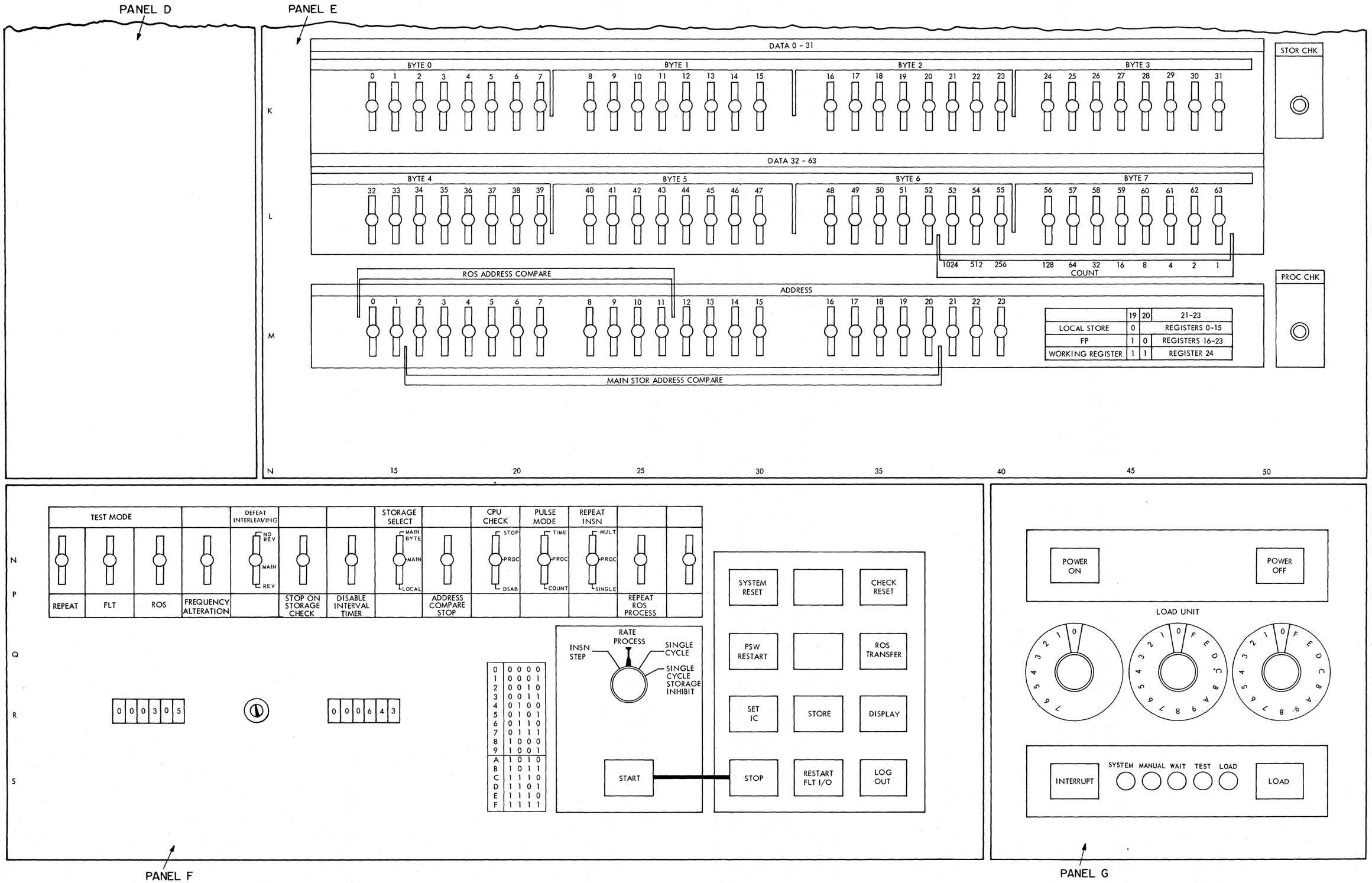


FIGURE 3-22. SYSTEM CONTROL PANEL (SHEET 2 OF 2)

After the stop trigger is set, entry to the stop loop is identical with depressing the STOP pushbutton (see paragraph 3.17.4).

3.17.6 DATA SWITCHES

The 64 DATA switches allow the operator to manually enter data into the system. Each switch is a 2-position toggle switch with the up position equalling a 0 and the down position equalling a 1. The 64 DATA switches are colored gray and white in hexadecimal groups to facilitate entering data in the CPU.

When data is entered in the CPU, correct parity is automatically generated. If the switches are altered during an operation, such as repeat instruction or storage ripple, an error will probably occur.

The DATA switches are used during the following manual operations:

1. Store
2. Storage ripple store
3. Repeat instruction
4. Pulse mode (count)

During manual operations, the selected manual microprogram places 1's in ST. If a DATA switch equals a 1, the corresponding bit in ST is unchanged (remains set). If a DATA switch equals a 0, however, the corresponding bit in ST is reset to 0.

3.17.7 ADDRESS SWITCHES

The 24 ADDRESS switches allow the operator to manually select any address in ROS. LS, or main storage.

Each switch is a 2-position toggle switch with the up position indicating a 0 and the down position indicating a 1. The 24 address switches are colored gray and white in hexadecimal groups. When an address is entered, correct parity is automatically generated when gated to the CPU.

To address main storage or LS, the ADDRESS switches are used with the STORAGE SELECT switch. ADDRESS switches 2 through 20 select a double word in main storage (may be used in conjunction with the ADDRESS COMPARE STOP switch when selecting an address for an address-compare stop or sync). ADDRESS switches 19 through 23 select an LS address.

ADDRESS switches 0 through 11 select an ROS address for comparison to obtain an ROS address sync or contain an ROS address for the ROS TRANSFER pushbutton.

The ADDRESS switches are used during the following functions:

1. Store
2. Display
3. Set IC
4. ROS transfer
5. ROS repeat
6. Main storage address-compare stop or sync
7. ROS address-compare sync

During manual operations, the selected manual microprogram places 1's in D. If an ADDRESS switch equals a 1, the corresponding bit in D is unchanged. If an ADDRESS switch equals a 0, however, the corresponding bit in D is reset.

3.17.8 STORAGE SELECT SWITCH

The STORAGE SELECT switch is a 3-position toggle switch that selects LS or main storage when storing or displaying data. The positions and corresponding functions are:

1. MAIN (center) position - Selects the double-word main storage address specified by ADDRESS switches 0 through 20 for both storing and displaying data.
2. MAIN BYTE position - For storing, selects the double-word location (per ADDRESS switches 0 through 20) and byte (per ADDRESS switches 21, 22, and 23) within the double word. For displaying, selects the double word in main storage to be displayed (same as MAIN position).
3. LOCAL position - Selects LS location (per ADDRESS switches 19 through 23) for both storing and displaying data.

The use of the STORAGE SELECT switch is discussed in the following paragraphs whenever it is involved in a specific manual operation.

3.17.9 STORE PUSHBUTTON

- Allows storing data in main storage or LS from DATA switches per STORAGE SELECT switch and ADDRESS switches.

The STORE pushbutton provides a means of storing information in any address of LS or main storage. The contents of the DATA switches are placed in the location specified by the ADDRESS switches and the STORAGE SELECT switch (Figure 6084, FEDM). If the STORAGE SELECT switch is in the LOCAL (down) position, the five low-order ADDRESS switches specify the LS location in which the contents (32 data bits plus 4 parity bits) of the right-half of the DATA switches will be stored.

If the STORAGE SELECT switch is in the MAIN BYTE position, one byte is stored in main storage per bits 12-23 of the ADDRESS switches. Bit positions 0 through 20 of the ADDRESS switches specify the double-word boundary in main storage. The value contained in the ADDRESS switches is placed in D for storing in main storage. The contents of the ADDRESS switches are placed in E for storing in LS.

If the STORAGE SELECT switch is in the MAIN position, the contents (64 data bits plus 8 parity bits) of the DATA switches are stored in main storage on a double-word boundary per the ADDRESS switches (0 through 20).

For all store operations, the original contents of D, S, and T are destroyed. Correct parity is automatically generated before storing in either main storage or LS.

After the data is stored, the microprogram enters the count delay routine and the CPU re-enters the stop loop (Figure 6084, FEDM).

3.17.10 DISPLAY PUSHBUTTON

- Allows displaying of data from main storage or LS in ST and AB per STORAGE SELECT switch and ADDRESS switches.

The DISPLAY pushbutton provides a means of displaying the contents of any location in LS or main storage. The address and the storage to be used are determined by the ADDRESS switches and the position of the STORAGE SELECT switch, respectively (Figure 6084, FEDM). Data from main storage (64 data bits plus 8 parity bits) is displayed in ST and AB. (Set roller switches 1 and 2 to position 3 for visual inspection of ST and roller switches 3 and 4 to position 3 for visual inspection of AB.) Data from LS (32 data bits plus 4 parity bits) is displayed in T. (Set roller switch 2 to position 3 for visual inspection of T.)

The original contents of S, T, and D are destroyed. After the selected data has been displayed, the count delay microprogram is executed and the stop loop is re-entered (Figure 6084, FEDM).

3.17.11 SET IC PUSHBUTTON

- Loads Q and R per ADDRESS switches.

The SET IC pushbutton sets the value contained in the ADDRESS switches into the instruction-address portion (bits 40 through 63) of the PSW [D(0-23)] and fills Q and R with the instructions beginning at the selected address. The value entered into the ADDRESS switches is the starting address of the next instruction to be executed. During the set-IC microprogram, four instruction halfwords are loaded into Q per the address in D (contents of the ADDRESS switches); 8 is then added to D, and the sum is placed in the IC.

The first addressed instruction halfword is loaded in R per D(21,22) (Figure 6084, FEDM). If D(21,22) equals 11, Q is loaded with the next group of four instruction halfwords per the IC, and 8 is added to the IC. If D(21,22) does not equal 11, Q is loaded only once.

Once the instruction halfwords are fetched and loaded into Q and R, the count delay microprogram is entered. After the count delay, the stop loop is entered (Figure 6084, FEDM). Further manual intervention is required to start program execution. (Normally, the START pushbutton is depressed.)

3.17.12 START PUSHBUTTON

- Starts CPU processing.
- Initiates selected manual functions.

The START pushbutton provides a means of starting the CPU in the process, instruction step, single-

cycle, or single-cycle storage inhibit mode. This pushbutton initiates manual operations depending upon the selected manual-control function. Figure 6084, FEDM, assumes that the CPU is in the process mode.

When the START pushbutton is depressed, a reset-stop-and-manual-triggers (0 → STP, STPLP) micro-order resets the stop and manual triggers. An end op occurs, and the I-Fetch of the next instruction begins program execution. If the START pushbutton is depressed after an abnormal stop loop entry or system reset, the results are unpredictable.

The block-interrupt trigger blocks interruptions at end op of the start microprogram. Therefore, interruptions are blocked until end op of the first instruction. The block-interrupt trigger is set by the 0 → STP, STPLP micro-order and reset by the I-Fetch-reset micro-order. This trigger does not block interruptions when the CPU is in the wait state.

3.17.13 ROS TRANSFER PUSHBUTTON AND STORAGE-RIPPLE MICROPROGRAM

- Allows ROS microprogram branch to any ROS location.

The ROS TRANSFER pushbutton allows entry into an ROS word. Depressing the ROS TRANSFER pushbutton places the contents of the 12 high-order ADDRESS switches in ROSAR. The next micro-instruction is taken from ROS and placed in the ROS sense latches. Further action now depends upon the RATE switch position.

If the RATE switch is in the PROCESS position, the CPU continues executing ROS commands from the entry point. In the instruction-step mode, the CPU continues until an end op is reached.

If the RATE switch is in the SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT position, the CPU stops with the micro-instruction displayed in the sense latches (indicators selected by switches). Depressing the START pushbutton advances ROS one cycle. RATE switch operation is discussed in paragraph 3.17.16.

Within ROS are microprograms that can be entered by placing the ROS address in the ADDRESS switches and depressing the ROS TRANSFER pushbutton, but only when the CPU is in the stop loop. To enter the ROS storage ripple microprogram, for example, with the CPU in the stop loop, a transfer to the correct ROS address (placed in the ADDRESS switches) is made by depressing the ROS TRANSFER pushbutton. The storage ripple routine is capable of (1) storing the data from the DATA switches in all addresses in LS or main storage and putting good parity in the storage-protect keys, or (2) reading all locations of LS or main storage and displaying the data. The choice of main storage or LS is made through the STORAGE SELECT switch.

If main storage is selected, the storage-ripple microprogram begins at address 0 and continues until an invalid address is detected. After the invalid address is detected, a restart beginning at address 0 occurs. Figure 6085, FEDM, shows the restart on an interruption request that resulted from detecting an invalid address. If LS is selected, the storage-ripple microprogram begins at address 0 and loops through all addresses in LS. Manual intervention (e.g., system reset or IPL) is required to exit from the storage-ripple microprogram.

The storage-ripple microprogram may also be used in troubleshooting by loading all storage locations with a predetermined value and then reading back the data. This microprogram may be used in conjunction with the STOP ON STORAGE CHECK switch. The storage-ripple microprogram is shown in Figure 6085, FEDM.

3.17.13.1 Storage-Ripple-Store Function

- Allows storing data in all locations in LS or main storage.

To store data in LS or main storage, the CPU must be in the stopped state, the data to be stored is entered in the DATA switches, and the STORAGE SELECT switch is positioned to select LS or main storage. Enter 800006 (hexadecimal) in ADDRESS switches 0 through 23, and depress the ROS TRANSFER pushbutton. The data previously entered in the DATA switches is stored in all locations in the selected storage. Also, good parity is placed in the storage protect keys on main storage ripple. Incorrect data may be stored if the DATA switches are changed when in the storage-ripple store routine. The ROS microprogram for the storage-ripple function is shown in Figure 6085, FEDM.

Since there is no automatic means of clearing main storage and LS, the storage-ripple store microprogram may be used to clear main storage and LS.

3.17.13.2 Storage-Ripple Display Function

- Allows reading and displaying data from all locations in LS or main storage.

The storage-ripple display microprogram reads all locations in LS or main storage as determined by the setting of the STORAGE SELECT switch. The CPU must be in the stopped state before the ROS TRANSFER pushbutton is depressed. To execute the storage-ripple display routine, enter 800000 (hexadecimal) in the ADDRESS switches, select main storage or LS, and depress the ROS TRANSFER pushbutton (Figure 6085, FEDM).

If LS is selected by means of the STORAGE SELECT switch, the data is displayed in S and PAL(32-63). If main storage is selected, the data is displayed in AB and ST. Data is checked in PAL for good parity.

3.17.14 PSW RESTART PUSHBUTTON AND WAIT STATE

- Allows loading new PSW.
- If PSW(14) = 1, enter wait state; if 0, enter running state.

The PSW RESTART pushbutton allows the operator to load a new PSW from main storage location 0 into the CPU. After the new PSW is fetched, the CPU will continue processing if the RATE switch is in the PROCESS position.

The stop and stop-loop (manual) triggers are reset at the beginning of the PSW-restart microprogram (Figure 6084, FEDM). The PSW RESTART pushbutton causes entry to the normal Load PSW instruction routine, which refills Q, R, and E.

At every end-op, PSW(14) is tested (Figure 6086, FEDM). If PSW(14) equals a 1, the wait state is entered. If the interval timer (location 80 of main storage) is to be stepped, the interval timer is stepped and the wait loop microprogram is re-entered. If the STOP pushbutton is depressed, the stop loop is entered. When a restart from the stop loop is executed, the wait state is re-entered if PSW(14) equals a 1.

An interruption causes a new PSW to be loaded in the CPU. PSW(14) is again tested, and the wait state is re-entered if PSW(14) equals a 1. If PSW(14) equals a 0, the CPU is placed in the running state. The time meter is stopped when in the wait state or stop loop unless a channel is running.

3.17.15 LOAD PUSHBUTTON (IPL)

- Allows loading a new program from preselected I/O device per LOAD UNIT switches.
- 24 bytes of data are read automatically into main storage locations 0-23. Remaining data is read in under CCW control.
- LOAD pushbutton is active in all modes of operation.

The IPL function provides a means of loading a new program into main storage from a preselected input device. Initial program loading is initiated manually by selecting the channel and I/O device by means of the three LOAD UNIT switches and depressing the LOAD pushbutton. The left LOAD UNIT switch, numbered from 0 through 7, selects 1 of 7 allowable channels. The remaining two switches are both numbered 0 through F (hexadecimal) and represent the unit address of the selected device.

Depressing the LOAD pushbutton causes a system reset, turns on the LOAD indicator, turns off the MANUAL indicator (if on), selects the I/O device, sends a start signal to the selected channel, and enters the IPL microprogram. The IPL operation is shown in Figure 6087, FEDM.

During the IPL, hardware-controlled logic generates an initial CCW. This initial CCW indicates that 24 bytes of data are to be read into locations 0 through 23 of main storage, and that chaining is required.

After the first 24 bytes are read into main storage, a new CCW is sent to the channel from main storage location 8. If chaining is indicated in the present CCW (from location 8), the next CCW is taken from location 16. The CCW also indicates the starting main storage address in which data is to be read. If more than two CCW's are required, the read-in program must control the additional CCW's.

After all program data is read into main storage, the channel sends a channel-release signal to the CPU. The signal is sensed by the microprogram (Figure 6087, FEDM). The I/O address is automatically stored in bits 21 through 31 of the first

word (PSW) in main storage. Bits 16 through 20 are made 0's, and bits 0 through 15 remain unchanged. The load-PSW microprogram loads the PSW from location 0 into the CPU. The PSW contains the starting address of the program which is loaded into the IC; assuming no errors or exceptional conditions exist, normal program execution begins.

The LOAD pushbutton is active in all CPU modes. The IPL system reset suspends all instruction processing, interruptions, and timer undating, and resets all channels, on-line nonshared control units, and I/O devices. The contents of general and floating-point LS registers remain unchanged.

If the selected I/O device is a disk file, the IPL information is read from track 0. If the I/O operations and PSW loading are not completed satisfactorily, the LOAD indicator remains on and the CPU waits for the release-CPU signal from the channel.

3.17.16 RATE SWITCH

- Controls rate of instruction execution.
- Must be in stop loop before activating switch.

The RATE switch selects the rate that instructions are executed. This rotary switch has four positions: PROCESS, INSN STEP, SINGLE CYCLE, and SINGLE CYCLE STORAGE INHIBIT.

Four triggers control the RATE switch operation: instruction-step, single-cycle, pass-pulse, and block.

The instruction-step trigger is set when the CPU is in the stopped state and the RATE switch is in the INSN STEP position, or when the pass-pulse trigger is reset and the RATE switch is in the INSN STEP position. The instruction-step trigger performs two functions: (1) it allows setting the stop trigger so that only one instruction is executed with each depression of the START pushbutton; (2) it disables the stepping of the interval timer.

The single-cycle trigger is set when the CPU is in the stopped state and the RATE switch is in the SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT position. This trigger allows single-cycle operation. One machine cycle is allowed with each depression of the START pushbutton unless the CPU requests additional machine cycles (single cycle). The interval timer is disabled by the single-cycle trigger.

When the pass-pulse trigger is set, CPU machine cycles are allowed. This trigger blocks the CPU machine cycles when in the single-cycle mode (RATE switch in SINGLE CYCLE or SINGLE CYCLE STORAGE INHIBIT position).

The block trigger is used in single-cycle operations to allow one clock pulse to be gated to the CPU each time the START pushbutton is depressed. Pulse blocking is accomplished by resetting the pass-pulse trigger.

3.17.16.1 PROCESS Position

When the RATE switch is in the PROCESS position, the system operates at the normal clock speed of 200 ns. This position is the position for normal program execution.

3.17.16.2 INSN STEP Position

The INSN STEP position allows the execution of one complete machine instruction for each depression of the START pushbutton. Any machine instruction may be executed in this mode. After completion of the instruction, the CPU enters the stop loop. The stop loop entry is identical with that achieved by the STOP pushbutton. All I/O operations and interruptions (if not masked off) are completed before the stop loop is entered.

When in the INSN STEP position, the stop trigger is set by an I-Fetch-reset micro-order or a reset-interrupt-triggers micro-order. The CPU remains in the stop loop until further action is taken by the operator.

Instruction-step operation is shown in Figure 6088, FEDM. The CPU must be in the stop loop before entering or leaving the instruction-step mode; the interval timer is disabled during the instruction-step mode.

3.17.16.3 SINGLE CYCLE Position

The SINGLE CYCLE position allows the CPU to advance one machine cycle (200 ns) each time the START pushbutton is depressed.

Single-cycle operation is shown in Figure 6089, FEDM. The CPU must be in the stop loop before entering or leaving the single-cycle mode and remains in the stop loop until the START pushbutton is depressed. The CPU begins executing instruc-

tions one machine cycle at a time for each depression of the START pushbutton. Figure 6089, FEDM, assumes that no CPU requests are generated; however, more than one machine cycle is required when an asynchronous device is used or a storage request is given. The single-cycle mode will continue through all CPU functions of the instruction to the point of initiation of the asynchronous operation. The asynchronous operation begins on the next depression of the START pushbutton and runs to the completion point in a normal manner.

If the asynchronous device initiates an interruption request during single-cycle operation, the interruption is broken into single machine cycles. More than one depression of the START pushbutton is therefore required. The CPU runs at normal machine speed in the stop loop.

3.17.16.4 SINGLE CYCLE STORAGE INHIBIT Position

The SINGLE CYCLE STORAGE INHIBIT position allows the CPU to advance one machine cycle (200 ns) each time the START pushbutton is depressed. All CPU requests are ignored, and asynchronous operations are suppressed.

3.17.17 REPEAT INSN SWITCH

The repeat-instruction functions provide a means of repeating a single instruction or repeating four instruction halfwords. The REPEAT INSN toggle switch has three positions:

1. PROC (center position) normal CPU operation.
2. SINGLE (down position).
3. MPLE (up position).

Two triggers control the repeat-instruction functions: (1) repeat-instruction-adjust trigger and (2) repeat-instruction-initialization trigger.

The repeat-instruction-adjust trigger sets the repeat-instruction-initialization trigger and forces a branch to the repeat-instruction microprogram at end op of the start microprogram. STAT G is set to block re-entering the repeat-instruction microprogram at end op. The repeat-instruction-adjust trigger is set when the CPU is in the stopped state and the REPEAT INSN switch is in the SINGLE or MPLE position. The trigger is reset when the CPU is in the stopped state and the REPEAT INSN switch is placed in the PROC position.

The pulse-mode-initialization trigger blocks ingating to Q and blocks stepping of the interval timer when in repeat-single-instruction mode. The trigger is reset when in the stop loop, and the RATE switch is placed in the PROCESS position. The CPU must be in the stopped state before entering or leaving the repeat-instruction mode. See Figure 6090, FEDM, for the repeat-instruction operations.

3.17.17.1 Repeat Single Instruction Function

When the REPEAT INSN switch is in the SINGLE position, one instruction is continuously executed. The instruction to be repeated must be entered in the DATA switches beginning with byte 0. If the CPU is in the stop loop, the repeat-instruction-adjust trigger is set when the REPEAT INSN switch is placed in either the MPLE or SINGLE position. To begin the instruction, the START pushbutton must be depressed.

In the repeat-single-instruction mode, a microprogram is executed to set up initial conditions before entering I-Fetch of the instruction to be executed (Figure 6090, FEDM).

The objectives of the repeat-instruction microprogram are to load the contents of the DATA switches in Q, set IC(21, 22) to 00, inhibit updating of IC(20) or above, gate the first instruction halfword from Q to R, set STAT G, and set the repeat-instruction-initialization trigger (Figure 6090, FEDM). A normal end op completes the repeat-instruction routine.

The instruction that was loaded in Q from the DATA switches is executed. Because the repeat-instruction-adjust trigger was not reset during the initial set-up routine, the ROS address to the repeat-instruction microprogram is forced at end op of the instruction. Re-entering the repeat-instruction microprogram on each instruction resets IC(21, 22) to 00, thus causing the first instruction to be repeated. Setting STAT G prevents returning to the repeat-instruction microprogram at end op.

3.17.17.2 Repeat-Multiple-Instructions Function

When the REPEAT INSN switch is in the MPLE position, the four instruction halfwords loaded in Q are continuously executed per IC(21, 22). The repeat-instruction-initialization trigger inhibits data from being gated from the SDBO to Q. Once instruction execution begins, the repeat-instruction microprogram is not entered because the repeat-instruction-adjust trigger is reset (Figure 6090, FEDM).

The repeat-multiple-instructions function is similar in operation to the repeat-single-instruction function except for the following:

1. The interval timer is allowed to step.
2. The repeat-instruction-adjust trigger is reset.
3. Interruptions are executed.

3.17.18 PULSE MODE SWITCH OPERATION

Pulse-mode operation provides a means of looping through a selected number of machine cycles, starting at a selected address, or of looping each time the interval timer is advanced.

The PULSE MODE toggle switch has three positions:

1. PROC (center position), normal CPU operation
2. TIME (up position)
3. COUNT (down position)

The PROC position of the PULSE MODE switch is used during normal program execution. The CPU must be in the stop loop before entering or leaving the pulse mode.

Two triggers control pulse mode operation: (1) pulse-mode-adjust trigger and (2) pulse-mode-initialization trigger. The pulse-mode-adjust trigger determines when to force an overriding branch to the pulse-mode-initialization microprogram and when to reset the system. This trigger can be set in one of three ways:

1. When the CPU is in the stop loop and the PULSE MODE switch is in either COUNT or TIME position.
2. When the CPU is not in the stop loop, the PULSE MODE switch is in the COUNT position, the scan-counter latch equals 0, and the pulse-mode-initialization trigger is set.
3. When the CPU is not in the stop loop, the PULSE MODE switch is in the TIME position, the time-clock-step single-shot is fired, and the pulse-mode-initialization trigger is set.

The pulse-mode-initialization trigger is set by depressing the START pushbutton with the pulse-mode-adjust trigger set. When the pulse-mode-initialization trigger is set, pulse mode operation begins.

3.17.18.1 TIME Position

- Load program in main storage.
- Place starting address of program in main storage byte locations 5-7.
- Enter stop loop.
- Place PULSE MODE switch in TIME position.
- Depress START pushbutton.

When the PULSE MODE switch is in the TIME position, instruction execution begins at the address specified in the address portion of the double word located in address 0 of main storage. Therefore, the starting address must be loaded in bits 40 through 63 of the double word located at address 0 prior to depressing the START pushbutton. Entering data manually into main storage (from the DATA switches) must be done with the CPU in the stop loop and the STORE pushbutton depressed. It is assumed that the program to be executed is contained in main storage.

The initial set-up conditions are:

1. The program is in main storage.
2. The starting address is in main storage byte locations 5 through 7.
3. The CPU is in the stop loop.
4. The PULSE MODE switch is in the TIME position.

Execution does not begin until the START pushbutton is depressed (Figure 6091, FEDM). After the START pushbutton is depressed, the pulse mode ROS address is forced in ROSAR. The objectives of the pulse-mode microprogram are to fetch the program starting address from main storage, load and update the IC, fetch the instruction halfwords, and branch to the first instruction. (These microprogram objectives are shown in Figure 6091, FEDM.) In TIME operation, the loading of the count in the maintenance-control-word (T → MCW microorder) is meaningless. Program execution continues until the interval timer is stepped. The pulse from

the time-clock-step single-shot causes a system reset. The pulse-mode microprogram is again entered and executed. This action results in executing the program from clock step to clock step. Looping through the pulse-mode microprogram and the main storage program continues until manually stopped.

3.17.18.2 COUNT Position

- Load program in main storage.
- Place starting address of program in main storage byte locations 5-7.
- Enter stop loop.
- Place PULSE MODE switch in COUNT position.
- Enter number of machine cycles (up to 2047) to be executed in DATA switches.
- Depress START pushbutton.

When the PULSE MODE switch is in the COUNT position, instruction execution begins at the address specified in the address portion of the double word located in address 0 of main storage. Each time the cycle counter is reduced to 0, a reset and a program restart occur.

Before executing the pulse-mode-count routine, the number of machine cycles must be determined and entered in DATA switches 53 through 63. The maximum count (number of machine cycles) is 2047.

Except for the PULSE MODE switch's being in the COUNT position and the count entered in DATA switches 53 through 63, the initial set-up conditions are similar to the Time mode. As shown in Figure 6091, FEDM, the same microprogram is executed for both COUNT and TIME positions. The program in main storage is entered at end-op of the microprogram. The cycle counter is reduced by 1 on each machine cycle. When the counter equals 0, a machine reset occurs and the pulse-mode microprogram is again executed. The looping through the microprogram and the storage program continues until manually stopped.

3.17.19 DISABLE INTERVAL TIMER SWITCH

The DISABLE INTERVAL TIMER switch prevents the interval timer from being advanced when placed in the down position. In the center position, the timer is stepped at regular predetermined intervals. The timer is contained in main storage location 80.

In addition to the switch, the timer is disabled when operating in the:

1. Single-cycle mode.
2. Instruction-step mode.
3. Repeat-instruction mode.
4. Stop-loop routine.

The DISABLE INTERVAL TIMER switch is inactive when the PULSE MODE switch is in the TIME position and the pulse-mode-initialization trigger is set.

3.17.20 DEFEAT INTERLEAVING SWITCH

The DEFEAT INTERLEAVING switch is a 3-position switch that performs the following functions:

1. NO REV (up) position - Interleaving of main storage addressing is disabled.
2. REV (down) position - Interleaving of main storage addressing is disabled, and the main storage addresses are reversed.
3. PROC (center) position - Normal position of the switch. Addressing is interleaved with no reversal of storage addresses.

This switch permits the operator to choose which halves of main storage are the high-order and low-order portions. When the switch is placed in the NO REV or REV position, the TEST indicator lights, indicating that the CPU is in the test mode.

3.17.21 CHECK RESET PUSHBUTTON

The CHECK RESET pushbutton provides a means of resetting all CPU and storage check triggers to the nonerror state. All logic check indicators are turned off. The CHECK RESET pushbutton is active in all modes of operation. Depressing the CHECK RESET pushbutton does not change the mode of operation. The operation continues as though no error conditions existed; the results, however, may be unpredictable.

3.17.22 INTERRUPT PUSHBUTTON

- If PSW(7) = 1, external interruptions occur; if 0, interruptions remain pending.

Depressing the INTERRUPT pushbutton initiates an external-interruption request by setting the console-signal trigger. If bit 7 of the current PSW is a 1, an interruption is taken after the current instruction and interruptions of higher priority are completed. If bit 7 of the current PSW is a 0, the manual interruption request remains pending.

During the interruption, bit 25 (interruption code portion) of the current PSW is made a 1, indicating that the INTERRUPT pushbutton is the source of the interruption. This pushbutton is effective while power is up for the system.

3.17.23 STOP ON STORAGE CHECK SWITCH

The STOP ON STORAGE CHECK switch provides a means of inhibiting storage accesses when a storage error occurs, so that the indicators will not be changed. The STOR CHK indicator lights to show that a storage error occurred and requires attention. The switchable indicators (rollers) are checked to determine the error and the address of the failing main storage word. The STOP ON STORAGE CHECK switch provides a storage stop upon encountering a storage check. The machine stop must not be confused with the stopped state or the stop loop. The STOP ON STORAGE CHECK switch may be used in conjunction with the CPU CHECK switch (see paragraph 3.17.24). The stop-on-storage-check function does not apply to LCS units.

3.17.24 CPU CHECK SWITCH

The CPU CHECK switch, a 3-position toggle switch, provides a means of controlling the system when a machine check is encountered.

3.17.24.1 PROC Position

Upon detection of a machine check and if PSW(13) (machine check mask bit) equals a 1, CPU clock pulses are blocked until the machine status is logged out to main storage. A machine check interruption is then executed. If PSW(13) equals a 0, the check triggers are set but no logout or interruption occurs until the PSW mask bit is set to 1.

3.17.24.2 STOP Position

Upon detection of a machine check, CPU clock pulses are blocked and no logout occurs. The check triggers are set, and the type of error is determined by examining the rollers. If the CHECK RESET pushbutton is depressed, operation is resumed, but the results may be unpredictable.

3.17.24.3 DSBL Position

Upon detection of a machine check, the check trigger are set. Logout or interruptions do not occur, and the operation is not terminated. Program execution continues, ignoring machine check errors. The check triggers may be reset by depressing the CHECK RESET pushbutton or the SYSTEM RESET pushbutton.

3.17.25 REPEAT ROS ADDRESS SWITCH

The REPEAT ROS ADDRESS switch provides a means for continuous readout of a specified ROS address. The address of the ROS microinstruction is entered in the ADDRESS switches, and the ROS TRANSFER pushbutton is depressed. This action initiates the reading of the specified ROS microinstruction. The same ROS address is accessed on each machine cycle. All storage requests are blocked. Changing the ADDRESS switches while in the repeat ROS loop may result in ROS parity checks.

3.17.26 CE KEY SWITCH

The CE key switch is a 2-position key-operated switch that selects one of two meters to indicate the CPU running time. In the Customer position, the customer meter indicates how long the customer used the CPU. In the CE position, the CE meter indicates the length of time that the CPU is used for noncustomer operation. The key cannot be removed when the switch is in the CE position.

3.17.27 FREQUENCY ALTERATION SWITCH

The FREQUENCY ALTERATION switch provides a means of increasing the CPU clock frequency. When the FREQUENCY ALTERATION switch is in the center position, each machine cycle is 200 ns (the normal machine speed).

The CE key switch must be in the CE position before the FREQUENCY ALTERATION switch functions

in the down position. When the FREQUENCY ALTERATION switch is in the down position and the CE key switch is in the CE position, the CPU clock cycle is decreased to 195 ns.

3.18 INDICATORS

The system control panel has 6 rows of indicators with 36 indicators in each row. Each row may be considered to represent one 32-bit word plus 4 parity bits (Figure 3-22). Associated with each row of indicators is a 6-position (position 6 not used) roller switch. The operator may display a register or the status of a trigger by placing the proper roller in the correct position. A roll chart is provided above each row of indicators to show the information being displayed for each indicator. As the roller position is changed, the roll chart rotates to correspond with the roller position. The rollers, the positions, and the information displayed for each row of indicators are shown in Figure 9058, FEDM.

Indicator lights are also located behind the POWER ON pushbutton on the system control panel. When power is on, the indicator glows white; when power is off or not completely cycled on for all units, the indicator glows red. Other indicators on the system control panel indicate machine status, check conditions, and power status. These indicators are defined below:

1. SYSTEM - Lights when the customer meter or CE meter is running.

2. MANUAL - Lights when the CPU is in the stopped state. The CPU is executing the stop loop ROS microprogram.

3. WAIT - Lights when the CPU is in the wait state. The CPU is executing the wait-loop ROS microprogram.

4. TEST - Lights when the following occur:

   a. The RATE switch is in a position other than PROCESS.

   b. The CPU CHECK switch is in a position other than PROC.

   c. The DISABLE INTERVAL TIMER switch is in the down position.

   d. The ADDRESS COMPARE STOP switch is in the stop position.

   e. The PULSE MODE switch is in a position other than PROC.

   f. The FLT switch is in the test mode position.

   g. The ROS switch is in the test mode position.

   h. The STOP ON STORAGE CHECK switch is in the down position.

   i. The REPEAT INSN switch is in a position other than PROC.

   j. The DEFEAT INTERLEAVING switch is in the NO REV or REV position.

   k. The channel is in test mode.

   l. The Diagnose instruction is active.

   m. The REPEAT ROS ADDRESS switch is in the repeat (down) position.

5. LOAD - Lights when the CPU is in a load state (IPL microprogram). The LOAD indicator is turned off after a successful load.

6. STOR CHK - Lights on all storage errors associated with the CPU. The rollers should be examined to determine the specific error.

7. PROC CHK - Lights on all CPU errors. A check of the rollers is necessary to determine the specific error.

CHAPTER 4

FEATURES

The features available for the 2065 are listed in the feature index below. Feature 7920 is described in a separate manual; features 3274, 8070, and 8080 are described elsewhere in this manual; features 7117, 7118, and 7119 will be issued as sections of this chapter by means of FE Supplements (FES).

FEATURE INDEX

Feature No.	Description	Refer To
3274	Direct Control:	
	Read Direct	Paragraphs 2.20 and 3.14.8
	Write Direct	Paragraphs 2.21 and 3.14.7
7117	7070/7074 Compatibility	Future FES
7118	7080 Compatibility	Future FES
7119	709/7040/7044/7090/7094/7094 II Compatibility	Future FES
7920	1052 Adapter	System/360 1052 Adapter, FEMI/FEMD, Form 223-2808
8070	2870 Attachment	Chapter 2, Section 3
8080	2361 Attachment	Chapter 2, Section 3

CHAPTER 5

POWER DISTRIBUTION AND CONTROL

This chapter describes the power distribution and control within the 2065 and the power control interface with the other units in the system. In this chapter, "system" refers to the 2065 CPU, the 2365 Storage unit(s), the 2860 and 2870 Channels, the 2361 Large Capacity Storage (LCS) unit(s), or any combination of these.

This chapter is divided into three major paragraphs. Paragraph 5.1 discusses the power for the 2060-2 or 2060-3 CPU converted to a 2065 CPU. This unit may be identified by the relay gate hinges on the right edge of the gate. Paragraph 5.2 discusses the power for the 2065 CPU as originally constructed. This unit may be identified by the relay gate hinge on the bottom edge of the gate. Paragraph 5.3 discusses those portions of power logic that are common to both versions.

Each unit (converted or originally constructed) has its own set of logic. Although the same logic page designations are used for both units, the part numbers are different, as shown in paragraphs 5.1 and 5.2.

5.1 2060 CONVERTED TO 2065

The descriptions and figures in this paragraph and in paragraph 5.3 are taken from the following logic:

Part Number	EC Level
5276565	705008C
5276578	705031
5276580 and 5276582	705008C
5276606	707482
5722219 through 5722221	707081
5753380 through 5753397	707481

A change is pending to modify 5753381 (logic YA021) from EC level 707481 to the level indicated in Figure 9059, FEDM.

5.1.1 POWER-ON SEQUENCE

- Close all circuit breakers.
- Close wall power switch.
- Move CPU READY switch to READY position.
- Depress CPU ON or POWER ON pushbutton.

Dc power in the CPU may be brought up in two ways. It may be initiated at the CE panel if only CPU power on is desired, or it may be initiated at the system control panel or the 2150 Console if full system power is desired. Figure 5-1 shows how the EPO loop is established. Figure 6092, FEDM, is a flow chart of the CPU power-on sequence, and Figure 9059, FEDM, is a simplified diagram of the circuits that are used.

The CPU power-on sequence is as follows:

1. Close all circuit breakers (CB1 through CB12).
2. Close wall power switch to apply main power at CB1.
3. Move CPU READY switch on CE panel to READY position.

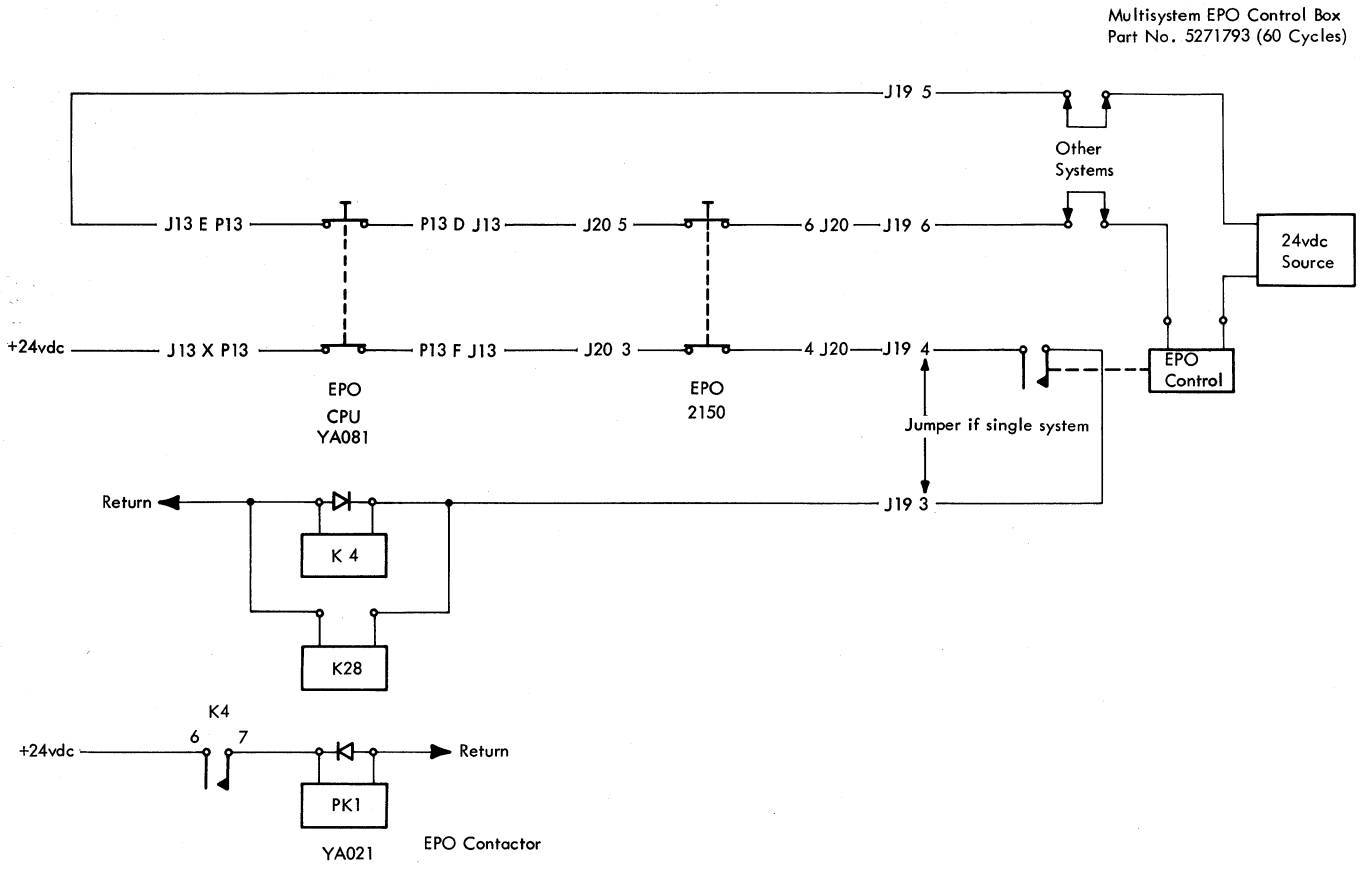
As a result of these three steps, the following occurs:

a. K4, the EPO-drive relay, and K28, the shared-EPO relay, are picked by 24vdc through the EPO switches and the multi-system EPO control box.

b. PK1, the EPO contactor, is picked by 24vdc through K4 contacts.

c. The 24vdc bus, source of all other 24vdc used in the CPU, is energized through PK1 contacts.

d. Main power is applied to PK2, the CPU-power contactor, through PK1 contacts and CB9.



Logic page YA051, except as noted

FIGURE 5-1. EPO LOOPS

e. K25 is picked by 24vdc through the power-off pushbuttons, K12 contacts, CPU READY switch, and protection-relay contacts. K25 has a 5-second resistive-capacitive (RC) time delay before it transfers.

The system waits at this point for the CPU ON pushbutton (step 4) or a POWER ON pushbutton (step 5) to be depressed. At least 5 seconds must elapse before the pushbutton is depressed.

4. Depress CPU ON pushbutton on CE panel.

a. K5 is picked by 24vdc through K12 contacts, CPU ON pushbutton, K25 contacts, and diode CR14.

b. K2 is picked through CR11.

c. K2 is held by 24vdc through K12 contacts, CPU READY switch, thermal and overcurrent sense relay contacts, CR7, and the K2 contacts.

d. K5 is momentarily held by 24vdc through the same protection-relay contacts, CR10, K5 and K26 contacts, and CR14.

e. K26 is picked by 24vdc through K5 contacts. K26 has a 5-second RC time delay before it transfers.

f. PK2 is picked by 24vdc through K5 contacts.

g. Main power is applied to the converter/inverter and the blowers through PK2 contacts.

h. The converter/inverter supplies 140v, 2500-cps ac to the regulators.

i. As the regulators develop the dc power for the CPU logic, the undervoltage sense relays, K10 and K11, are picked.

j. K5 is now held by 24vdc through the same protection-relay contacts as before, CR10, K5, K10, and K11 contacts, and CR14.

Note

If K10 and K11 had not picked before K26 transferred, K5 would have dropped, thus dropping PK2 and K26.

k. K25 drops as K10 or K11 picks.

l. If PS9 is in the CPU, it receives 140v, 2500-cps ac through relay II-K1 contacts. II-K1 is picked by 24vdc through K5, K10, and K11 contacts.

5. Depress POWER ON pushbutton on either system control panel or 2150 Console.

a. The stepper drive coil is pulsed by 24vdc through stepper switch contacts A-26 to A-COM and the interrupter contacts; the stepper switch advances to position 1.

b. K7 and K32 are picked through CR9; K6 is picked through CR9 and CR8.

c. K6, K7, and K32 are held by 24vdc through the power-off switches and K7 contacts. K6 may also be held by 24vdc from the storage units through K6 contacts. K7 and K32 are isolated from this 24vdc by CR8.

Note

If the stepper switch is not at the start position (position 26) when the 24vdc bus is energized, it is advanced to the start position by 24vdc through K32 and the start-interlock contacts. The start-interlock contacts remain closed until the stepper switch reaches the start position.

d. K12 is picked by 24vdc through K6 contacts.

e. K5 is picked by 24vdc through stepper switch contacts C-1 to B-1, K25 and K26 contacts, and CR14.

f. K2 is picked through CR11.

g. K2 is held by 24vdc through K6 contacts, CPU READY switch, thermal and over-current sense relay contacts, CR7, and K2 contacts.

h. K5 is momentarily held by 24vdc through the same protection-relay contacts, CR10, K5 and K26 contacts, and CR14.

i. The sequencing continues as described in steps 4e to 4l. The holding 24vdc, however, is through K6 contacts and not K12 contacts.

j. The stepper switch is advanced to position 2 by 24vdc through K10, K11, and stepper switch contacts A-1 to A-COM.

5.1.2 POWER-OFF SEQUENCE

- Move CPU READY switch to OFF position, or depress POWER OFF pushbutton.

Dc power in the CPU may be dropped in two ways. It may be initiated at the CE panel if only the CPU power is on. However, if any attached storage unit is on, data in storage may be lost as the CPU dc goes down. System dc power off may be initiated at the system control panel or at the 2150 Console. A holding line prevents the CPU and channel power from going off before the storage units to protect the data. Figure 6094, FEDM, is a flow chart of the power-off sequence.

The CPU power-off sequence is as follows:

1. Move CPU READY switch on CE panel to OFF position.

a. The 24vdc holding line to K5 and K2 is opened, causing them to drop.

b. The 24vdc holding line to PK2, through K5 contacts, is opened, causing PK2 to drop.

c. Main power to the converter/inverter, through PK2 contacts, is opened, causing CPU dc to go off.

d. If system power was up, it remains up since K6, K7, and K32 did not drop.

Note

Under these conditions, data may be lost in the storage units as the CPU dc goes down.

2. Depress POWER OFF pushbutton on either system control panel or 2150 Console. As a result:

a. The 24vdc holding line to K6, K7, and K32 is opened, causing them to drop.

Note

If any storage unit is on, K6 continues to hold with 24vdc (CPU voltage) returned from the unit, through K6 contacts.

b. The power-hold line to the storage units, through K7 and K32 contacts, is opened, causing them to cycle down.

c. As the last storage unit does down, the 24vdc holding K6 goes off, causing K6 to drop.

d. The 24vdc holding line to K12, K5, and K2, through K6 contacts, is opened, causing them to drop.

e. The 24vdc holding line to PK2, through K5 contacts, is opened, causing it to drop.

f. Main power to the converter/inverter, through PK2 contacts, is opened, causing CPU dc to go off.

g. The power-hold line to the channels, through K6 contacts, is opened, causing them to cycle down.

5.1.3 OVERCURRENT PROTECTION

The overcurrent sensing circuits are internal to each regulator. Refer to SLT Power Supply, FEMI, Form Z22-2799, for details of the circuits. Any fault that draws excessive current from any regulator, except PS9, causes the CPU to drop power. Figure 5-2 shows the overcurrent sense loop for converted units.

When an overcurrent condition exists, continuity between terminals 8 and 9 is broken. 24vdc, through auxiliary switch contacts on CB2 through CB8 and through terminals 8 and 9 of the regulators, except PS9, pick K8, the overcurrent-sense relay.

24vdc through terminals 8 and 9 of PS9 pick K18, the 48v power-check relay.

5.1.4 OVERVOLTAGE PROTECTION

The overvoltage sensing circuits provide fault protection and an indication that all CPU dc is below maximum voltage levels. Any fault that will raise the output voltage level of any regulator, except PS9, above the maximum causes CPU power to drop. (See logic YA091 for the sensing circuits.)

A silicon-controlled rectifier (SCR) in series with a low resistance is connected across the output terminals of each regulator, except PS9. When gated on, this SCR shorts the regulator output through the low resistance, causing an overcurrent condition within the regulator. The internal overcurrent protection circuit initiates CPU power off. The SCR gate is fed directly from the collector circuit of a sensing transistor (positive supplies) or an SCR gating transistor (negative supplies).

5.1.4.1 Positive Regulators

A line from each positive regulator, except PS9, is fed through a metering jack to the emitter of a sensing transistor. The base of this transistor is held at a preset upper limit voltage by Zener diode Z1 and potentiometer R4 (3v supplies) or R9 (6v supplies). The collector series load resistors are connected to ground or 0v. When its emitter is held negative with respect to its base by the regulator, the transistor is biased off, holding the collector circuit and the SCR gate at 0v. When the regulator output rises above the upper limit, causing the emitter to become positive with respect to the base, the transistor starts to conduct. The current drawn through the collector load resistors raises the collector circuit and the SCR gate to a positive level with respect to the regulator output level; the SCR is gated on.

5.1.4.2 Negative Regulators

A line from each negative regulator, except PS11, provides, through a metering jack, the return path for the collector circuit of an SCR gating transistor and the emitter of a sensing transistor. The base of the sensing transistor is held at a preset negative voltage limit by Z2 and R7. The collector load resistor is connected to ground or 0v. The base of the SCR gating transistor is fed by a resistor from the collector of the sensing transistor. The emitter of the SCR gating transistor is connected to ground or 0v. The sensing transistor, when its emitter is held positive with respect to its base by the regulator, is biased off, and its collector circuit and the base of the SCR gating transistor are held at 0v. As a result, the SCR gating transistor is biased off and its collector circuit and the SCR gate are held at the regulator output level. When the regulator output level increases beyond the negative voltage limit, causing the emitter of the sensing transistor to become negative with

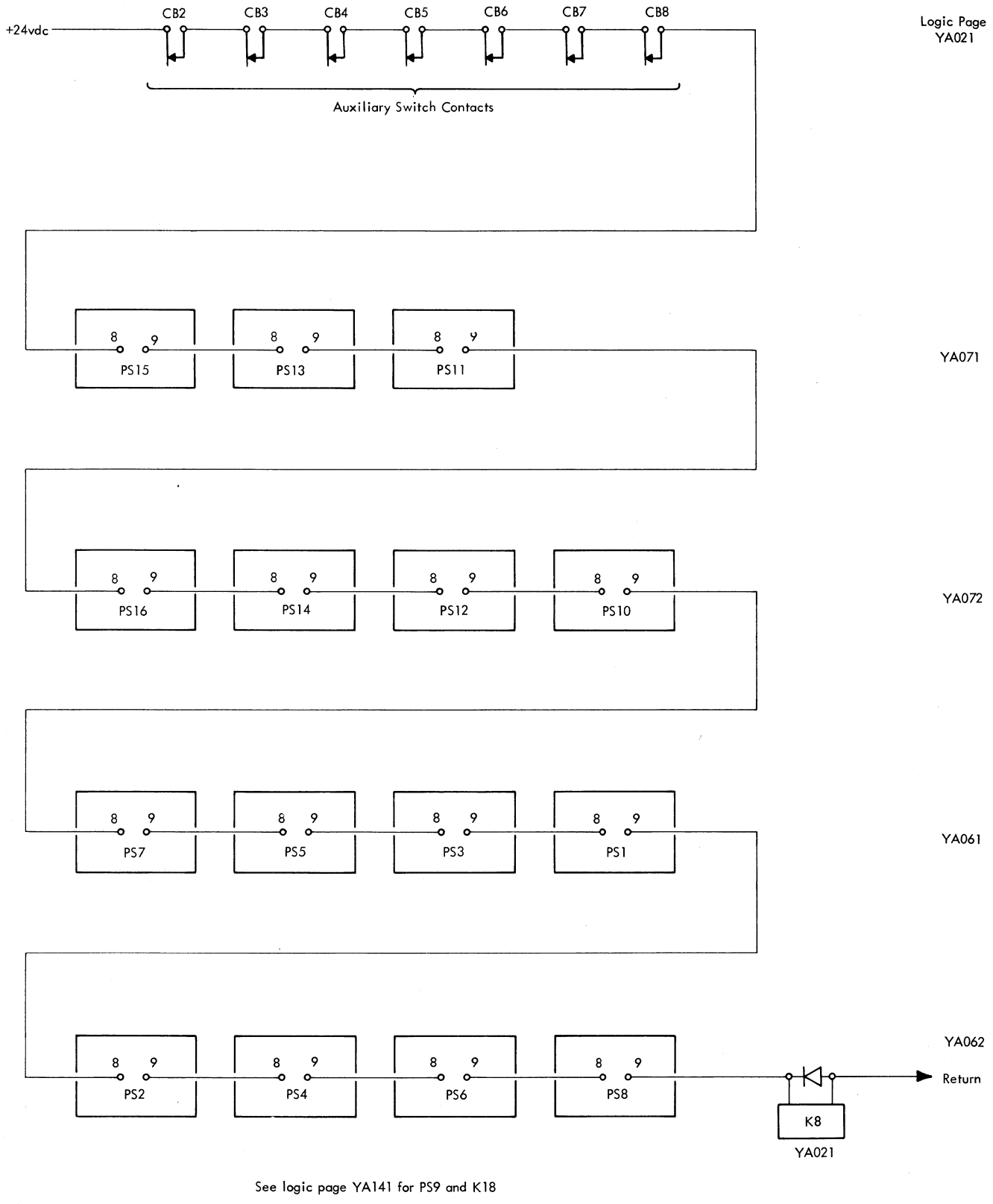


FIGURE 5-2. OVERCURRENT PROTECTION LOOP, CONVERTED UNITS

respect to its base, the transistor starts to conduct. The current drawn through the collector load resistor lowers its collector circuit and the base of the SCR gating transistor to a negative level. With its base made negative with respect to its emitter, the SCR gating transistor starts to conduct. The current drawn through the collector load resistor raises the collector circuit and the SCR gate to a positive level with respect to the regulator output level; the SCR is gated on.

Regulator PS11 is sensed in the same way as the other negative supplies, except that the emitter voltage of the sensing transistor is established through an adjustable voltage-dividing network in the line from the regulator. This dividing network permits the sensing transistor to use the same voltage reference (Z2 and R7) as the other negative supplies.

5.2 2065 ORIGINAL UNITS

The descriptions and figures in this paragraph and in paragraph 5.3 are taken from the following logic:

Part Number	EC Level
5276606	707482
5722202 and 5722203	707482
5722204 through 5722214	707081
5722215 through 5722218	707482
5722219 through 5722221	707081
5722222	707482
5722223 through 5722225	707081

A change is pending to modify 5722204 (logic YA021) from EC level 707081 to the level indicated in Figure 9060, FEDM, and Figure 5-3.

5.2.1 POWER-ON SEQUENCE

- Close all circuit breakers.
- Close wall power switch.
- Move CPU READY switch to READY position.
- Depress CPU ON or POWER ON pushbutton.

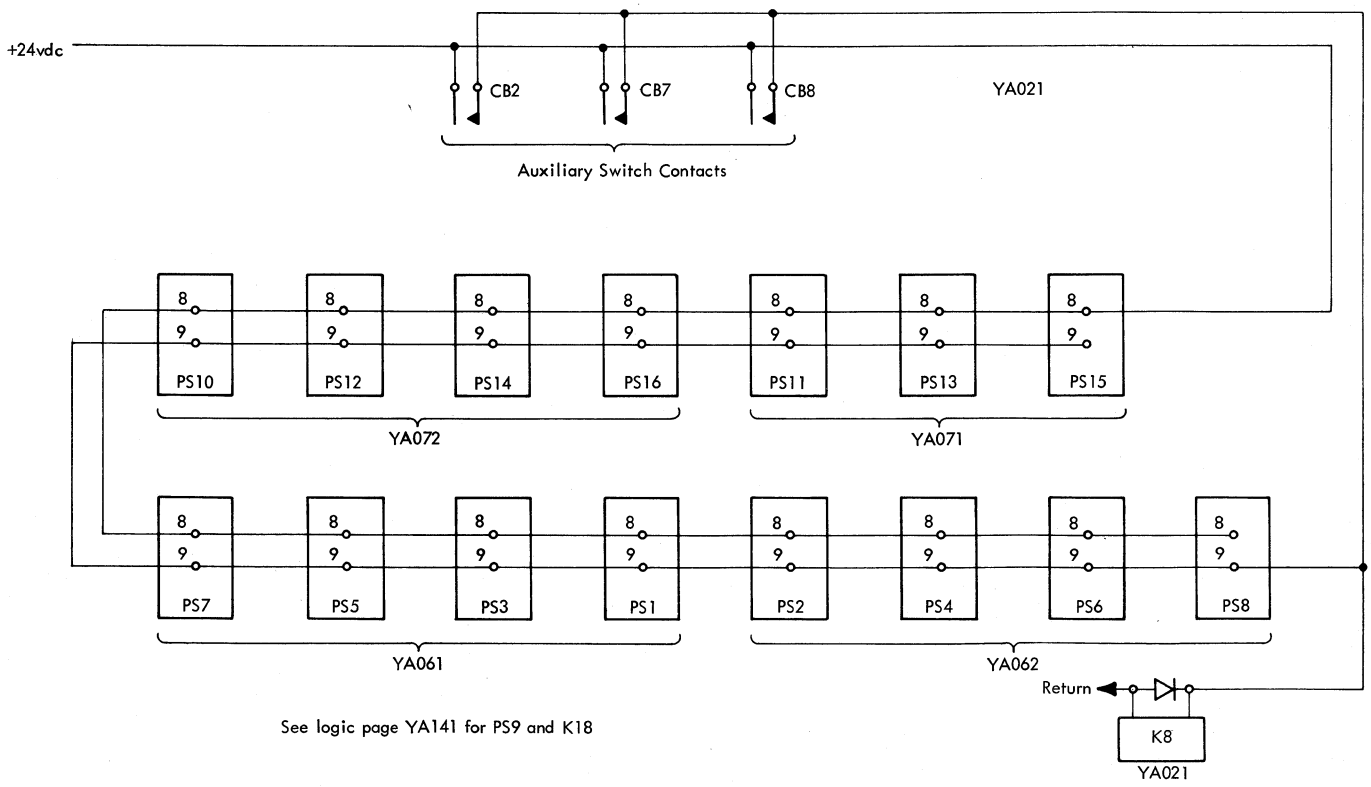


FIGURE 5-3. OVERCURRENT PROTECTION LOOP, ORIGINAL UNITS

Dc power in the CPU may be brought up in two ways. It may be initiated at the CE panel if only CPU power on is desired, or it may be initiated at the system control panel or the 2150 Console if full system power is desired. Figure 5-1 shows how the EPO loop is established. Figure 6093, FEDM, is a flow chart of the CPU power-on sequence, and Figure 9060, FEDM, is a simplified diagram of the circuits that are used.

CPU power-on sequence is as follows:

1. Close all circuit breakers (CB1, CB2, and CB7 through CB12).

2. Close wall power switch to apply main power at CB1.

3. Move CPU READY switch on CE panel to READY position.

   As a result of these three steps, the following occurs:

   a. K4, the EPO-drive relay, and K28, the shared-EPO relay, are picked by 24vdc through the EPO switches and the multi-system EPO control box.

   b. PK1, the EPO contactor, is picked by 24vdc through K4 contacts.

   c. The 24vdc bus, source of all other 24vdc used in the CPU, is energized through PK1 contacts.

   d. Main power is applied to PK2, the CPU-power contactor, through PK1 contacts and CB9.

   e. K47 is picked by 24vdc through the power-off pushbuttons, CPU READY switch, and protection-relay contacts. K47 has a 5-second RC time delay before it transfers.

   The system waits at this point for the CPU ON pushbutton (step 4) or a POWER ON pushbutton (step 5) to be depressed. At least 5 seconds must elapse before the pushbutton is depressed.

4. Depress CPU ON pushbutton on CE panel.

   a. K5 is picked by 24vdc through CPU ON pushbutton, K47 contacts, and diode CR14.

   b. K2 is picked through CR11.

   c. K2 is held by 24vdc through CPU READY switch, thermal and overcurrent sense relay contacts, CR7, and K2 contacts.

   d. K5 is momentarily held by 24vdc through the same protection-relay contacts, CR10, K5 and K46 contacts, and CR14.

   e. K46 is picked by 24vdc through K5 contacts. K46 has a 5-second RC time delay before it transfers.

   f. PK2 is picked by 24vdc through K5 contacts.

   g. Main power is applied to the converter/inverter and the blowers through PK2 contacts.

   h. The converter/inverter supplies 140v, 2500-cps ac to the regulators.

   i. As the regulators develop the dc power for the CPU logic, the undervoltage sense relays, K10 and K11, are picked.

   j. K5 is now held by 24vdc through the same protection-relay contacts as before, CR10, K5, K10, and K11 contacts, and CR14.

   Note

   If K10 and K11 had not picked before K46 transferred, K5 would have dropped, thereby dropping PK2 and K46.

   k. K47 drops as K10 or K11 picks.

   l. If PS9 is in the CPU, it receives 140v, 2500-cps ac through relay II-K1 contacts. II-K1 is picked by 24vdc through K5, K10, and K11 contacts.

5. Depress POWER ON pushbutton on either system control panel or 2150 Console.

   a. The stepper drive coil is pulsed by 24vdc through stepper switch contacts A-26 to A-COM and the interrupter contacts; the stepper advances to position 1.

   b. K7 and K32 are picked through CR9; K6 through CR9 and CR8.

   c. K6, K7, and K32 are held by 24vdc through the power-off switches and K7 contacts.

Note

If the stepper switch is not at the start position (position 26) when the 24vdc bus is energized, it is advanced to the start position by 24vdc through K6 and the start-interlock contacts. The start-interlock contacts remain closed until the stepper switch reaches the start position.

d. K5 is picked by 24vdc through stepper switch contacts C-1 to B-1, K47 and K46 contacts, and CR14.

e. K2 is picked through CR11.

f. K2 is held by 24vdc through the CPU READY switch, thermal and overcurrent sense relay contacts, CR7, and K2 contacts.

g. K5 is momentarily held by 24vdc through the same protection-relay contacts, CR10, K5 and K46 contacts, and CR14.

h. The sequencing continues as described in steps 4e to 4l.

i. The stepper switch is advanced to position 2 by 24vdc through K10, K11, and stepper switch contacts A-1 to A-COM.

5.2.2 POWER-OFF SEQUENCE

- Move CPU READY switch to OFF position, or depress POWER OFF pushbutton.

Dc power in the CPU may be dropped in two ways. It may be initiated at the CE panel if only the CPU power is on, or it may be initiated at the system control panel or 2150 Console if system power off is desired. If, however, any attached storage unit is on, data in storage may be lost as the CPU dc goes down. A change is pending to prevent this loss of data. Figure 6094, FEDM, is a flow chart of the power-off sequence.

The CPU power-off sequence is as follows:

1. Move CPU READY switch on CE panel to OFF position.

   a. The 24vdc holding line to K5 and K2 is opened, causing them to drop.

   b. The 24vdc holding line to PK2, through K5 contacts, is opened, causing PK2 to drop.

   c. Main power to the converter/inverter, through PK2 contacts, is opened, causing CPU dc to go off.

   d. If system power was up, it remains up since K6, K7, and K32 did not drop.

2. Depress POWER OFF pushbutton on either system control panel or 2150 Console. As a result:

   a. The 24vdc holding line to K2, K5, K6, K7, and K32 is opened, causing them to drop.

   b. The power-hold line to the storage units, through K7 and K32 contacts, is opened, causing them to cycle down.

   c. The 24vdc holding line to PK2, through K5 contacts, is opened, causing it to drop.

   d. Main power to the converter/inverter, through PK2 contacts, is opened, causing CPU dc to go off.

   e. The power-hold line to the channels, through K6 contacts, is opened, causing them to cycle down.

5.2.3 OVERCURRENT PROTECTION

The overcurrent sensing circuits are internal to each regulator. Refer to SLT Power Supply, FEMI, Form Z22-2799, for details of the circuits. Any fault that draws excessive current from any regulator, except PS9, causes the CPU to drop power. Figure 5-3 shows the overcurrent sense loop for original units.

When an overcurrent condition exists, continuity is provided between terminals 8 and 9. 24vdc, through auxiliary switch contacts on CB2, CB7, or CB8, or through terminals 8 and 9 of the regulators, except PS9, pick K8, the overcurrent-sense relay.

24vdc through terminals 8 and 9 of PS9 pick K18, the 48v power-check relay.

5.2.4 OVERVOLTAGE PROTECTION

The overvoltage sensing and protection circuits are internal to each regulator. Any fault that will

raise the output voltage level of any regulator, except PS9, above the maximum causes the CPU to drop power via the overcurrent sense loop.

5.3 COMMON PORTIONS

The descriptions and figures in this paragraph are taken from the prints referenced in paragraphs 5.1 and 5.2.

5.3.1 AC POWER DISTRIBUTION

The primary ac power distribution is shown in Figures 5-4 and 5-5, and the load on transformer T1 is shown in Figure 5-6.

Main power from CB1 is applied to T1 via fuses F1 and F2. T1 provides 28vac for the remote margin power, the alarm circuit, and T1-TB2. T1-TB2 supplies 24vdc to the EPO loop and, via PK1 contacts, to the 24vdc bus.

Main power is applied by PK1 contacts to:

1. T2, via CB12. T2 provides 115vac to (1) storage 1, via CB11, (2) wall convenience outlets, via CB10, and (3) 1052 Printer-Keyboard, via CB10 and PK2 contacts.

2. T3, via F3, F4, and CB2. T3 provides 40vac to the elapsed-time meters, 12.6vac to the logic clock, and low-voltage ac to T3-P-TB, which, in turn, provides 20vdc to the undervoltage and overvoltage protection circuits.

3. T4, via F3, F4, and CB7. T4 provides 28vac to the converter/inverter.

4. Three-phase line detector and PK2, via CB9. The 3-phase line detector is not used at this time and may not be installed.

5.3.2 DC POWER DISTRIBUTION

The various dc voltages required by the CPU logic are provided by 15 regulators. A 16th regulator, the 48v power supply (PS9), is not installed in the CPU when the 2150 Console option is used. The 2150 Console has a multivoltage power supply which provides the 48vdc. Table 5-1 lists the output voltage levels and the major load for each regulator. Tables 5-2 and 5-3 list the part numbers of the regulators for the converted units and the original units, respectively.

5.3.3 POWER CONTROL INTERFACE

Full system power is achieved in a sequential and interlocked manner by the power control interface circuitry. After initiating power on in the CPU, the CPU stepper switch waits for CPU power up to be confirmed, then advances to an attached stand-alone unit, directs it to bring its power up, waits for power up to be confirmed, and advances to the next unit, repeating this procedure until power is up in all attached units. Each stand-alone unit has its own internal power-sequencing control. Figure 5-7 shows a typical power control interface connection, using channel 1 as an example.

With the stepper switch at position 2 (see 5.1.1, step 5j, or 5.2.1, step 5i), the system power-on sequence continues automatically:

1. 24vdc (supplied by channel 1) through K4, K6, and stepper switch contacts C-2 to B-2 provide channel 1 with a power-pick level. This power-pick level initiates internal power-on sequencing of channel 1. When channel 1 has completed power-on sequencing, 24vdc (from the CPU) are returned as a power-complete signal through stepper switch contacts A-2 to A-COM to advance the stepper switch to position 3. K19 is picked and held at the same time.

2. Channel 2 is picked using position 3, K4, and K6 contacts. The power-complete signal advances the stepper switch to position 4 and picks and holds K21.

3. Channel 3 is picked using position 4, K4, and K6 contacts. The power-complete signal advances the stepper switch to position 5 and picks and holds K31.

4. The stepper switch is advanced to position 11 by 24vdc at stepper switch contacts A-5 through A-10 to A-COM.

5. Storage unit 1 is picked using position 11, K4, and K7 contacts. The power-complete signal advances the stepper switch to position 12 and picks and holds K29.

6. Storage unit 2 is picked using position 12, K4, and K7 contacts. The power-complete signal advances the stepper switch to position 13 and picks and holds K30.

7. Storage unit 3 is picked using position 13, K28, and K7 contacts. The power-complete signal advances the stepper switch to position 14 and picks and holds K20.

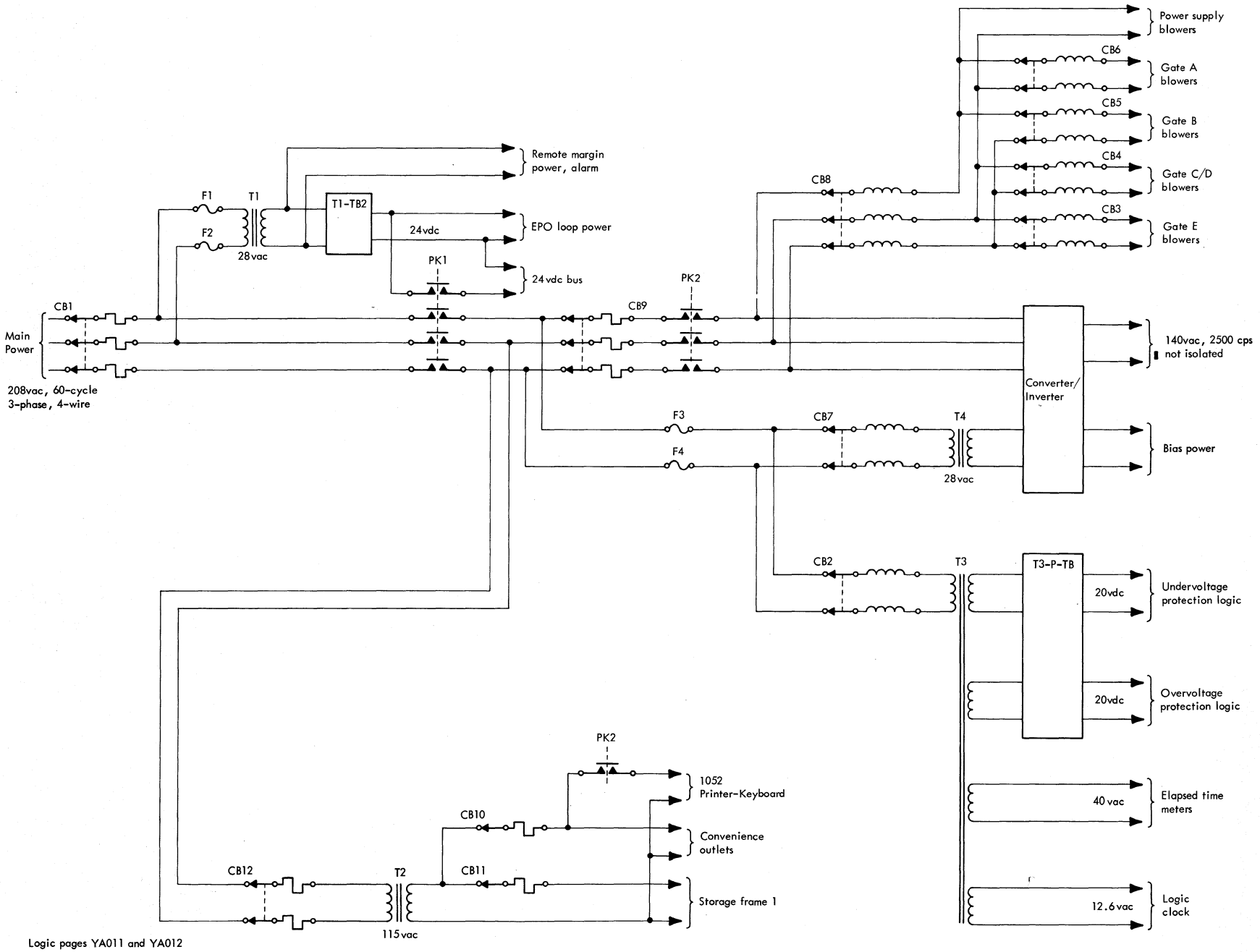


FIGURE 5-4. PRIMARY AC POWER DISTRIBUTION, CONVERTED UNITS

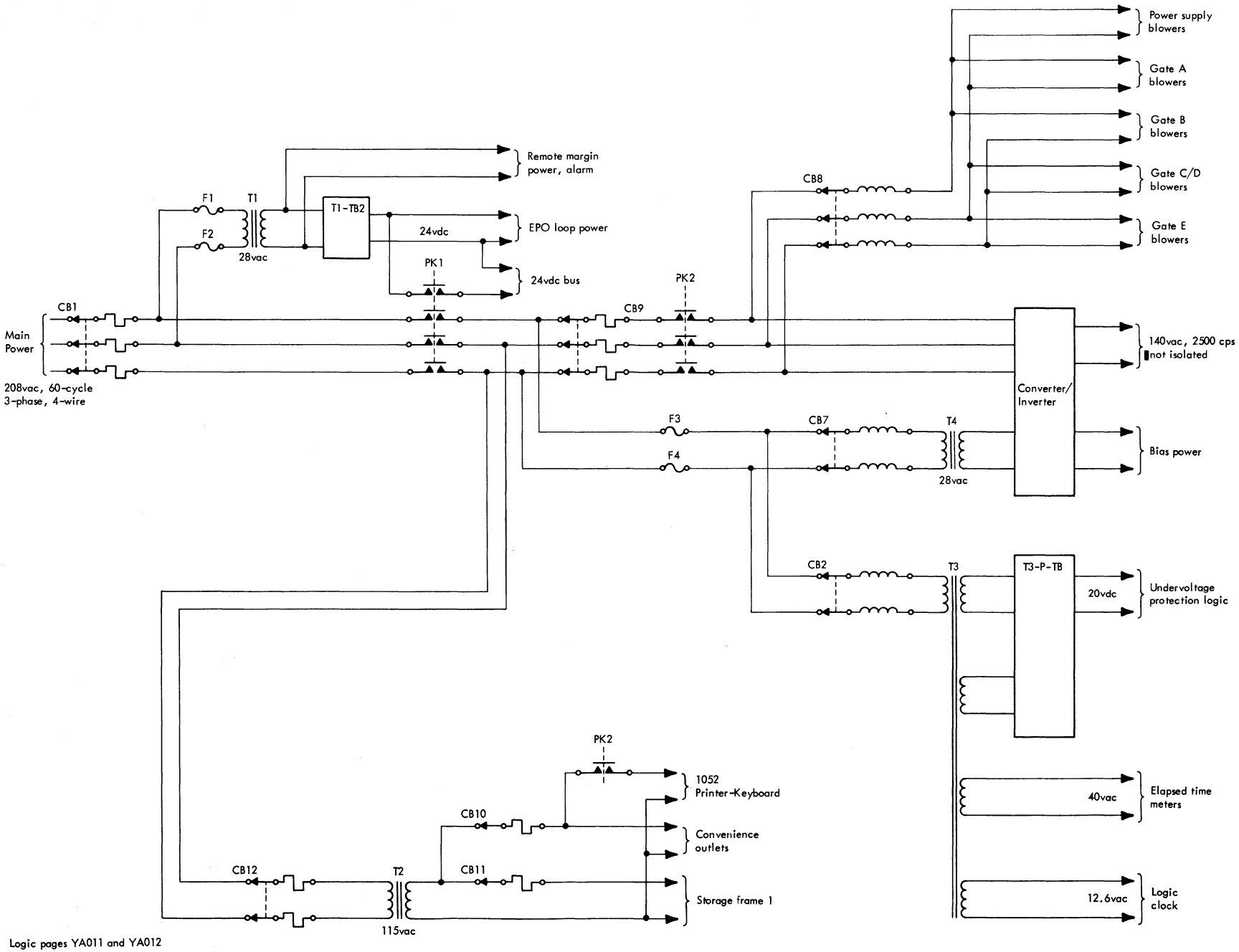


FIGURE 5-5. PRIMARY AC POWER DISTRIBUTION, ORIGINAL UNITS

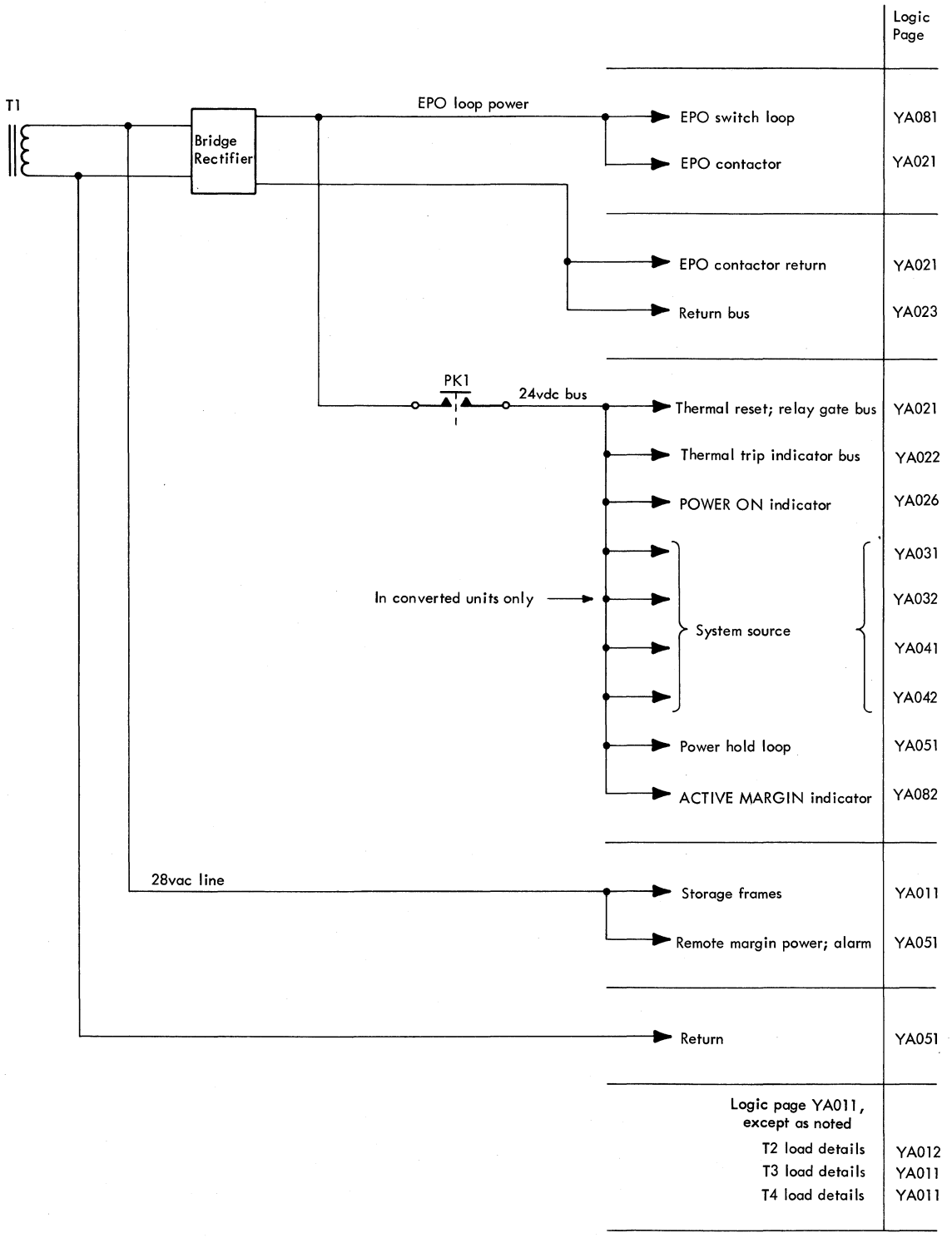


FIGURE 5-6. TRANSFORMER T1 LOAD DETAIL

TABLE 5-1. DC REGULATORS

Regulator	Output	Distribution		Return		Logic
		Bus	Pin	Pin	Common path	
PS1	+6v 25 amp	Gate C/D upper	12	11	Return bus: YA023	YA061
PS2	+3v 40 amp	Gate B lower	8	7	Jumper: YA131	YA062
PS3	+3v 40 amp	Gate E lower	8	7	Jumper: YA131	YA061
PS4	-3v 40 amp	Gate C/D upper	10	9	Jumper: YA131	YA062
		Gate C/D lower	10	9	Jumper: YA131	YA062
		Gate E upper	10	9	Jumper: YA131	YA062
		Gate E lower	10	9	Jumper: YA131	YA062
PS5	+3v 40 amp	Gate E upper	8	7	Jumper: YA131	YA061
PS6	+3v 40 amp	Gate C/D upper	4	3	- -	YA062
		Gate C/D lower	8	7	Jumper: YA131	YA062
PS7	+6v 40 amp	Gate E upper	12	11	Return bus: YA023	YA061
		Gate E lower	12	11	Return bus: YA023	YA061
PS8	+6v 40 amp	Gate C/D lower	12	11	Return bus: YA023	YA062
PS9*	+48v 2 amp	II BB9	1	-	Return bus: YA023	YA141
PS10	-3v 40 amp	Gate A upper	10	9	Jumper: YA131	YA072
		Gate A lower	10	9	Jumper: YA131	YA072
		Gate B upper	10	9	Jumper: YA131	YA072
		Gate B lower	10	9	Jumper: YA131	YA072
PS11	-18v 11 amp	Gate C/D upper	6	5	Jumper to 7: YA131	YA071
PS12	+3v 40 amp	Gate A lower	12	11	Return bus: YA023	YA072
		Indicators (J13)	-	-	- -	YA072
PS13	+3v 40 amp	Gate B upper	8	7	Jumper: YA131	YA071
PS14	+3v 40 amp	Gate A upper	12	11	Return bus: YA023	YA072
		Indicators, meter (J13)	-	-	- -	YA072
PS15	+6v 40 amp	Gate B upper	12	11	Return bus: YA023	YA071
		Gate B lower	12	11	Return bus: YA023	YA071
PS16	+6v 40 amp	Gate A upper	8	7	Jumper: YA131	YA072
		Gate A lower	8	7	Jumper: YA131	YA072

* PS9 may not be installed

TABLE 5-2. REGULATOR PART NUMBERS, CONVERTED UNITS

Regulator	Part Number	SCR Card
3v 40 amp	5261220	5276857
6v 25 amp	5261230	5276858
6v 40 amp	5261240	5276858
48v 2 amp	5261280	-
18v 11 amp	5244090	5351199

TABLE 5-3. REGULATOR PART NUMBERS, ORIGINAL UNITS

Regulator	Part Number
3v 40 amp	5712020
6v 25 amp	5712030
6v 40 amp	5712040
48v 2 amp	5712080
18v 11 amp	5709320

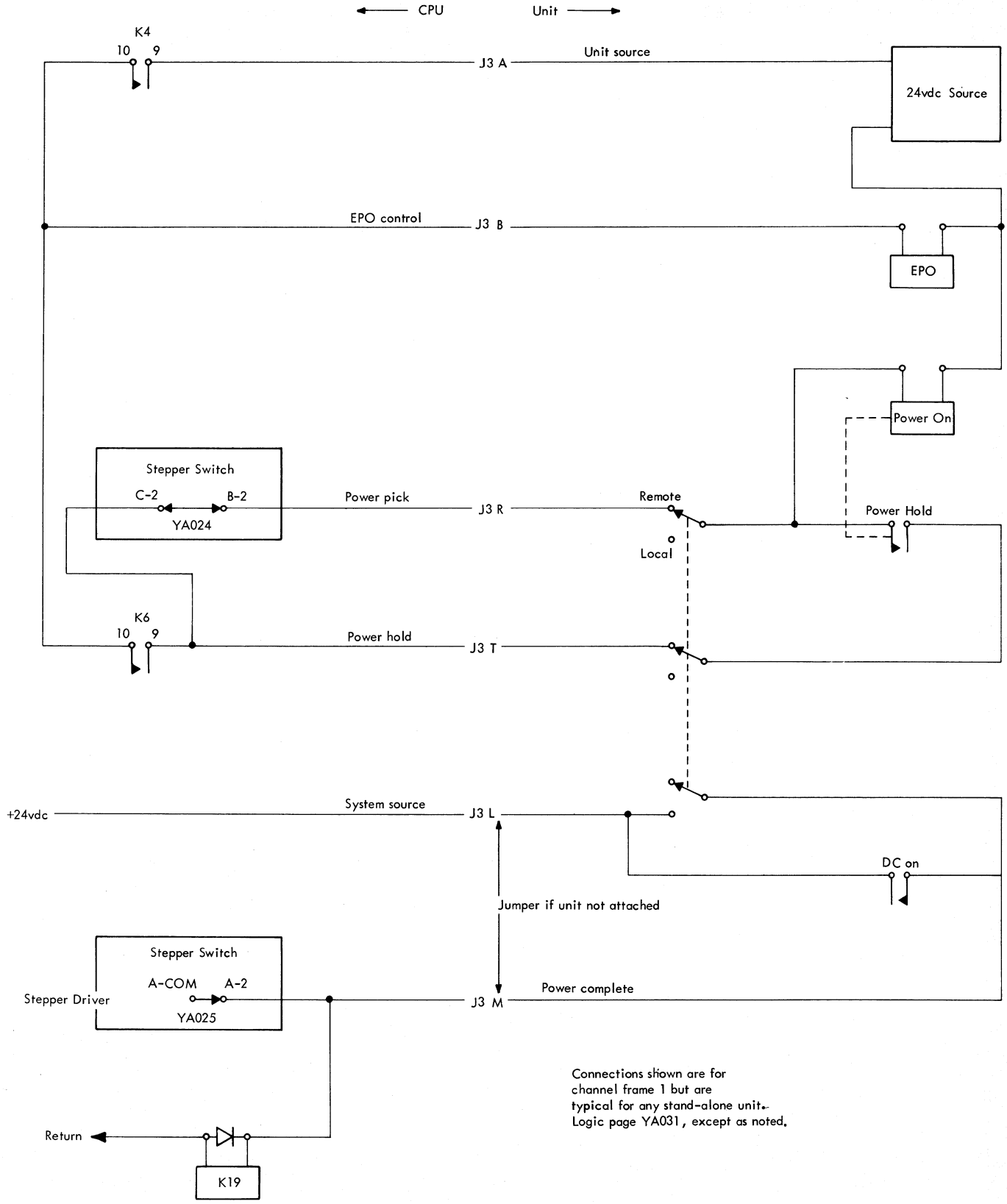


FIGURE 5-7. POWER CONTROL INTERFACE

8. Storage unit 4 is picked using position 14, K28, and K7 contacts. The power-complete signal advances the stepper switch to position 15 and picks and holds K22.

9. LCS unit 1 is picked using position 15, K28, and K32 contacts. The power-complete signal advances the stepper switch to position 16.

10. LCS unit 2 is picked using position 16, K28, and K32 contacts. The power-complete signal advances the stepper switch to position 17.

11. LCS unit 3 is picked using position 17, K28, and K32 contacts. The power-complete signal advances the stepper switch to position 18.

12. LCS unit 4 is picked using position 18, K28, and K32 contacts. The power-complete signal advances the stepper switch to position 19.

13. The stepper switch is advanced to position 26, the start position, by 24vdc at stepper switch contacts A-19 through A-25 to A-COM.

Note

If any of these units are not attached to the CPU, a jumper must be installed to provide a simulated power-complete level return to advance the stepper switch to the next position. If any unit is attached to the CPU, but is not being used in the system, a LOCAL-REMOTE switch in the unit allows it to be bypassed in the sequencing. In the LOCAL position, power control is at the unit, except for EPO. In the REMOTE position, the unit is sequenced on by the stepper switch. The partition switches for storage units 3 and 4 (located in the CPU) allow them to be bypassed from the CPU.

5.3.4 POWER-ON LOGIC RESET

Provision is made for resetting the CPU logic after power has been brought up. (See logic YA021.)

The momentary continuity required by the CPU logic for reset is provided by K35 contacts. K35 is in series with a 100-μf (converted units) or a 350-μf (original units) capacitor and a set of normally open contacts of K26 (converted units) or K46 (original units) to the 24vdc bus. The initial charging current of the capacitor is sufficient to pick K35. As the charge on the capacitor builds up, the current becomes insufficient to hold K35, causing it to drop. When K26 or K46 drops, the capacitor is discharged through a 12-ohm resistor. Relay K26 or K46 transfers 5 seconds after the regulators are energized and drops as the regulators are de-energized.

5.3.5 EMERGENCY POWER OFF

- Pull any EMERGENCY PULL switch.

Emergency power off in the system may be effected by pulling the EMERGENCY PULL switch at either the system control panel or the 2150 Console or by losing continuity through the multisystem EPO interface control box. If the switch at the system control panel or the 2150 Console is pulled, the multisystem EPO control box initiates emergency power off in the other systems connected to it. Figure 5-1 shows the circuits to establish the EPO loop. Figure 5-7 shows the circuit to drop power in the stand-alone units in this system, using channel 1 as an example.

Within the CPU, the following takes place:

1. The 24vdc holding line to K4, through the pull switches and the multisystem EPO control box, is opened, causing K4 to drop.

2. The 24vdc holding line to PK1, through K4 contacts, is opened, causing PK1 to drop.

3. Main power to the CPU, except to T1, through PK1 contacts, is opened, causing a complete power off. T1 provides the low-voltage ac to T1-TB2 for the dc to establish the EPO loop.

4. The EPO-hold line to the stand-alone units, through K4 or K28 contacts, is opened, causing them to drop complete power, even if they are in the LOCAL control mode.

Note

All EMERGENCY PULL switches latch mechanically and must be reset by maintenance personnel.

5.3.6 INDICATORS

5.3.6.1 Power Check Indicators

Indicators on the system control panel show an incomplete power-up status in the CPU or in the

attached storage units and channels. The CPU and the units are individually indicated, as is the system status. (See logic YA026 and YA083.) The system status is shown by the "system power check" indication consisting of red lamps backlighting the POWER ON pushbutton. This pushbutton is normally white. This condition is also indicated on the 2150 Console. There are nine power check indicators.

1. CPU POWER CHECK: 24vdc through K2 contacts are fed through parallel-connected contacts on the undervoltage-sense relays to the CPU POWER CHECK indicator. If either of these two relays is not picked after K2 is picked, this indicator lights. 24vdc are also fed through parallel-connected contacts on the overcurrent and thermal-sense relays to the CPU POWER CHECK indicator. 24vdc through 48v power-check relay contacts, or through the CPU READY switch in the OFF position, also light this indicator.

2. STOR FRAME 1 POWER CHECK: 24vdc, via K32 (converted units) or K6 (original units) contacts, are fed through K29 contacts to this indicator. If K29 is not picked after K32 or K6 is picked, this indicator lights.

3. STOR FRAME 2 POWER CHECK: Similarly, through K30 contacts, this indicator lights.

4. STOR FRAME 3 POWER CHECK: Similarly, through K20 contacts, this indicator lights.

5. STOR FRAME 4 POWER CHECK: Similarly, through K22 contacts, this indicator lights.

6. CHAN FRAME 1 POWER CHECK: Similarly, through K19 contacts, this indicator lights.

7. CHAN FRAME 2 POWER CHECK: Similarly, through K21 contacts, this indicator lights.

8. CHAN FRAME 3 POWER CHECK: Similarly, through K31 contacts, this indicator lights.

9. System Power Check

A line from the CPU POWER CHECK indicator and a line from each unit power check indicator are combined at an OR. The OR turns on the system power check indicator. If any one of the power check indicators is on, the system power check indicator also lights.

5.3.6.2 System Power-On Indicator

An indicator on the system control panel shows complete power-up status in the CPU and in the attached storage units and channels. (See logic YA026 and YA083.) This "system power on" indicator consists of clear or white lamps backlighting the POWER ON pushbutton. This condition is also indicated on the 2150 Console.

24vdc through the CPU undervoltage-sense relay contacts and the storage units and channels power-check relay contacts light the system power on indicator. If any of these relays is not picked, the indicator remains off.

5.3.7 THERMAL PROTECTION

Thermal protection is provided by placing sensing elements in the return path of each thermal relay. If an overtemperature condition exists, the return path is opened, thus dropping the associated relay and lighting an indicator. Logic YA022 shows the relay and indicator circuits.

The thermal relays are automatically reset when the 24vdc bus is energized by PK1 contacts, or when the THERMAL RESET pushbutton on the CE panel is depressed.

Relay K3, which provides the 24vdc pulse to reset the thermal relays, is in series with a 1500-μf capacitor and a set of normally closed contacts of K26 (converted units) or K46 (original units) and the 24vdc bus. The initial charging current of the capacitor is sufficient to pick K3. As the charge on the capacitor builds up, the current becomes insufficient to hold K3, causing it to drop. An alternate path to pick K3 is through the THERMAL RESET pushbutton to the 24vdc bus. When K26 or K46 transfers, the capacitor is discharged through a 12-ohm resistor. Relay K26 or K46 transfers 5 seconds after the regulators are energized.

24vdc through the transferred contacts of K3 and through diodes CR1 through CR6 cause the thermal relays to pick, provided a normal temperature condition exists (sensors closed). Each thermal relay holds through its own contacts as K3 drops. An open sensor causes the relay to drop, and its contacts light the associated indicator on the CE panel.

5.3.8 UNDERVOLTAGE PROTECTION

The undervoltage sensing circuits provide fault protection and an indication that all CPU dc is above

minimum voltage levels (2.4v). Any fault that will lower the output voltage level of any regulator, except PS9, below the minimum causes CPU power to drop. (See logic YA111 for the sensing circuits.)

A line from each positive regulator, except PS9, is fed through isolation switches to an input of an AND. The output level of the AND is determined by the lowest input level; that is, the one nearest zero or ground. Thus, if any supply, not isolated, is off, the output level of the AND is near zero. The output of the AND is transferred directly to the base of transistor Q1. Transistor Q1 shares an emitter load resistor with Q2. The base of Q2 is held at 2.4v by Zener diode CR59 and resistor R8. Transistor Q2 now carries the full emitter load current and holds the common emitter circuit to 2.4v. With its emitter held positive with respect to its base, Q1 is biased off. As the output level of the AND raises the base of Q1 to 2.4v, Q1 begins to conduct and starts to share with Q2 the current drawn by the emitter load resistor. As the output level of the AND raises the base of Q1 above 2.4v, Q1 raises the emitter circuit above 2.4v. With its emitter raised to a level positive with respect to its base, Q2 is biased off. Transistor Q1 now carries the full emitter load current; this current is sufficient to pick K10.

A similar circuit, of opposite polarity, checks the negative regulators and picks K11.

The isolation switches permit CPU dc power to be brought up; for servicing purposes, with a low output-voltage level from a regulator. At least one positive and one negative regulator isolation switch must be on to pick K10 and K11.

CAUTION

Under these conditions, CPU logic circuitry may be damaged by the nonstandard voltages.

5.3.9 MARGINAL ADJUSTMENTS

Several power supplies in the CPU and in the attached storage units and channels may have their output level varied from the nominal output. This feature allows testing critical circuits with nonstandard voltages as an aid in predicting failures.

When a supply or attached unit is margined, an "active margin" 24vdc level is generated. Each 24vdc level is fed to an input of an OR to light the ACTIVE MARGIN indicator, showing the system has an active marginal adjustment. The MARGIN/METER SEL rotary switch may be turned to successive positions to locate the margined supply or unit. The LOCATE MARGIN indicator lights when the switch is at the position of a margined supply or unit. The voltage level is shown on the meter. (See logic YA081, YA082, and YA121 for the circuits.)

Within the CPU, four of the 6v supplies (PS7, PS8, PS15, and PS16) and the 18 v ROS supply (PS11) have their output levels adjusted by individual panel controls. These marginal controls have a cam and switch for the 24vdc active margin level.

In the attached units, the margined supplies are adjusted by a motor drive which is controlled by lever switch MARGIN CHANNEL/STOR and selected by the MARGIN/METER SEL switch. 28vac through the raise and lower motor in each unit are returned through the rotary switch to select the unit and the lever switch to actuate the motor. The RAISE position causes the motor to adjust the marginal voltages to a higher level. The LOWER position adjusts the marginal voltages to a lower level. The active margin 24vdc level is sent to the CPU by each unit with a margined supply.

5.3.10 CONVERTER/INVERTER

The converter/inverter (PN 5703200) converts the 280v, 60-cps, 3-phase ac to dc and inverts the dc to 140v, 2500-cps, 1-phase ac for the regulators. A detailed description of the operation is given in the SLT Power Supply Manual of Instruction, Form Z22-2799. Figure 9061, FEDM, is a simplified schematic.

The 3-phase wall power is converted to dc by the 3-phase bridge rectifier. The output of the rectifier is filtered by capacitors C0, C1, and C2. In addition to filtering, C1 and C2 provide a split source for the inverter. Resistors R1 and R2 help balance this split source and provide bleeder loading to discharge the capacitors when power is turned off.

The inverter is basically two SCR's that alternately switch the load across Edc at a 2500-cycle rate. A 2500-cps square wave is formed at load points A and B. Switching is performed by the two load SCR's, SCR 1 and SCR 2, and the two commutating SCR's, SCR 3 and SCR 4. Assume SCR 1 and SCR 3 are gated on. Load current I1 enters the load at point A. Capacitor C3 charges to Edc. When C3 reaches full Edc, SCR 3 turns off. At this point, SCR 4 is gated on. Capacitor C3 discharges against SCR 1, turning it off. At this point, SCR 2 is gated on. Load current I2 enters the load at point B; C3 charges to Edc. When C3 reaches full Edc, SCR 4 turns off. At this

point, SCR 3 is gated on. Capacitor C3 discharges against SCR 2, turning it off. The cycle then repeats with SCR 1 gated on.

DANGER

The output to the regulators is not isolated from the wall power, presenting a lethal potential to ground.

5.3.11 REGULATORS

The regulators rectify, control, and filter the 2500-cps ac from the converter/inverter to the necessary dc voltage levels. Isolation from the wall power is provided. The operation of the regulators is described in detail in the SLT Power Supply Manual of Instruction, Form Z22-2799. Figure 9062, FEDM, is a simplified schematic.

DANGER

The input to the regulators from the converter/inverter is not isolated from the wall power, presenting a lethal potential to ground.

A bridge magnetic amplifier determines and controls the output dc voltage level. The magnetic amplifier is a "square loop" toroidal core with three windings. The gate winding passes power to the output relative to the degree of saturation in the core. The current in a bias winding sets the core at the approximate middle of the slope of the saturation curve. The current in a control winding adjusts this point according to the amount of error detected by the output voltage level check circuit.

At the start of the gate cycle (the half of the input ac cycle determined by the diodes to the particular gate winding), the inductance of the gate winding is high, causing a slow current rise. At some point in the gate cycle, the current in the gate winding rises to a point where the total of the currents in all three windings is sufficient to saturate the core. At saturation, the gate winding current jumps to full current. When the current cycle passes through zero, the saturation level returns to the point determined by the control and bias winding currents.

5.3.12 ELAPSED TIME METERS

Two meters on the system control panel show the CPU running time while it is processing customer data (process meter) and while it is being operated by the customer engineer (CE meter). A key-operated switch selects the meter to be driven. The normal position of this switch allows power to be applied to the process meter. The customer engineer, using a key, switches to the CE meter. (See logic YA082 for the circuit.)

Relay RR1 on the use meter card is picked by a signal from the CPU logic. 40vac through RR1 contacts drive the meter clock.

5.3.13 ALARM FEATURE

The alarm feature, a customer option, provides the operator with an audible signal when directed by the program. It may be used, for example, to signal a program hangup or the end of a program. (See logic YA141 for the circuit.)

The 48vdc through relay A-K2 have a return path through the CPU logic. When the CPU logic grounds the return path, A-K2 is picked. A-K2 is then held by A-K1 and A-K2 contacts. The 28vac through A-K2 contacts pick A-K1 and start the bell ringing. A-K1 has a short delay before it transfers; A-K2 drops as A-K1 transfers, and the bell stops ringing.

APPENDIX A

UNIT CHARACTERISTICS

Tables A-1 through A-39 give the characteristics of the units that may be part of a Model 65 system.

See IBM System/360 Installation Manual - Physical Planning SRL, Form C22-6820, for further details on the physical characteristics. For further details on the performance characteristics, see the Manual of Instruction and associated SRL for the particular unit.

TABLE A-1. CHARACTERISTICS OF 2065 PROCESSING UNIT

Characteristics	Description
Unit dimensions (including 2365 Processor Storage):	
Front width	19 feet, 8.5 inches
Depth	22 feet, 4.5 inches
Height	72.5 inches
Service clearances (including 2365 Processor Storage):	
Front	48 inches
Rear	30 inches
Left	53 inches
Right	53 inches
Weight	2400 lbs
Heat dissipation	12,000 BTU/hour 3100 cfm
Operating environment (including 2365 Processor Storage):	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 6.9-kva, 3-phase, 4-wire, 60-cycle

TABLE A-1. CHARACTERISTICS OF 2065 PROCESSING UNIT (Cont)

Characteristics	Description
Storage capacity	May use up to four 2365 Processor Storage units and up to four 2361 Core Storage Units
Access time (8-byte access), maximum	750 ns
Bytes per second	Up to 1.3 million (set by channels)

TABLE A-2. CHARACTERISTICS OF 1052-7 PRINTER-KEYBOARD*

Characteristics	Description
Unit dimensions:	
Front width	23 inches
Depth	19.75 inches
Height	9 inches
Weight	65 lbs
Heat dissipation	335 BTU/hour 0 cfm
Operating environment:	
Temperature	50°F - 110°F
Relative humidity	10-80%
Wet bulb temperature	80°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	10-80%
Wet bulb temperature	80°F
Power requirements (power from CPU)	0.1-kva
Printing rate	14.8 characters per second
Control unit required	None, direct to 2065

* Used as data entry and printout unit for 2065 CPU.

TABLE A-3. CHARACTERISTICS OF 1053-1 PRINTER

Characteristics	Description
Unit dimensions:	
Front width	33 inches
Depth	11.5 inches
Height	9 inches
Weight	35 lbs
Heat dissipation	335 BTU/hour
Operating environment:	
Temperature	50°F - 110°F
Relative humidity	10-80%
Wet bult temperature	80°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	10-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 0.1-kva, 1-phase, 3-wire, 60-cycle
Printing rate	14.8 characters per second
Typing line length	13 inches
Control unit required	1051 Control Unit

TABLE A-4. CHARACTERISTICS OF 1403-2, -3, -7, -N1 PRINTER

Characteristics	Description
Unit dimensions:	
Front width	57.125 inches
Depth	29 inches
Height	53.5 inches
Service clearances:	
Front	36 inches
Rear	36 inches
Left	42 inches
Right	42 inches
Weight	825 lbs

TABLE A-4. CHARACTERISTICS OF 1403-2, -3, -7, -N1 PRINTER (Cont)

Characteristics	Description
Heat dissipation	4600 BTU/hour 350 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Power requirements (from 2821 Control Unit)	1.4-kva
Printing rate (lines per minute)	Model 2: 600 Model 3: 1100 Model 7: 600 Model N1: 1100
Printing positions (60 characters)	Model 2: 132 Model 3: 132 Model 7: 120 Model N1: 132
Control unit required	2821 Control Unit

TABLE A-5. CHARACTERISTICS OF 1442-N2 CARD PUNCH

Characteristics	Description
Unit dimensions	Not available
Service clearances	Not available
Weight	Not available
Heat dissipation	Not available
Operating environment	Not available
Nonoperating environment	Not available
Power requirements	Not available
Storage capacity	1200 cards in hopper 1300 cards in stacker
Punch rate	Columns 1 to 10: 270 cards per minute Columns 1 to 80: 91 cards per minute
Channel required	Multiplexor or selector (control unit position)

TABLE A-6. CHARACTERISTICS OF 1443-N1 PRINTER

Characteristics	Description
Unit dimensions:	
Front width	58.875 inches
Depth	43 inches
Height	46 inches
Service clearances:	
Front	36 inches
Rear	36 inches
Left	30 inches
Right	48 inches
Weight	800 lbs
Heat dissipation	3200 BTU/hour 50 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	10-80%
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Power requirements	208v or 230v, 1.1-kva, 1-phase, 3-wire, 60-cycle
Printing rate (lines per minute)	240
Printing positions (52 characters)	120
Channel required	Multiplexor or selector

TABLE A-7. CHARACTERISTICS OF 2150 CONSOLE*

Characteristics	Description
Unit dimensions:	
Front width	64 inches
Depth	28.75 inches
Height	52.125 inches
Service clearances:	
Front	30 inches
Rear	48 inches
Left	30 inches
Right	30 inches

TABLE A-7. CHARACTERISTICS OF 2150 CONSOLE* (Cont)

Characteristics	Description
Weight	800 lbs
Heat dissipation	1740 BTU/hour 180 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	10-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 0.65-kva, 3-phase, 4-wire, 60-cycle
Control unit required	None, direct to 2065

* Remote operators position for 2065 CPU

TABLE A-8. CHARACTERISTICS OF 2250-1, -2 DISPLAY UNIT

Characteristics	Description
Unit dimensions:	
Front width	Model 1: 58 inches Model 2: 36 inches
Depth	Model 1: 72 inches Model 2: 44 inches
Height	Model 1: 50 inches Model 2: 50 inches
Service clearances:	
Front	Model 1: 27 inches Model 2: 30 inches
Rear	Model 1: 30 inches Model 2: 30 inches
Left	Model 1: 30 inches Model 2: 30 inches
Right	Model 1: 25 inches Model 2: 30 inches
Weight	Model 1: 894 lbs Model 2: 375 lbs
Heat dissipation	Model 1: 7200 BTU/hour 480 cfm Model 2: 6600 BTU/hour 320 cfm

TABLE A-8. CHARACTERISTICS OF 2250-1, -2 DISPLAY UNIT (Cont)

Characteristics	Description
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 150°F
Relative humidity	8-80%
Wet bulb temperature	85°F
Power requirements	208v or 230v, 2.8-kva for Model 1, 2.4-kva for Model 2, 1-phase, 3-wire, 60-cycle
Storage capacity	Model 1: up to 8192 bytes
Access time (1 byte)	Model 1: 4.2 ms
Channel or control unit required	Model 1: Multiplexor or selector (control unit position) Model 2: 2840-1 Display Control Unit

TABLE A-9. CHARACTERISTICS OF 2260-1 DISPLAY STATION

Characteristics	Description
Unit dimensions:	
Front width	13 inches
Depth	21 inches
Height	16 inches
Weight	40 lbs
Heat dissipation	408 BTU/hour
Operating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Nonoperating environment:	
Temperature	50°F - 150°F
Relative humidity	8-80%
Power requirements	115v, 1-phase, 3-wire, 60-cycle
Control unit required	2848-1, -2, -3 Display Control Unit

TABLE A-10. CHARACTERISTICS OF 2280-1 RECORDER, 2281-1 SCANNER, 2282-1 RECORDER SCANNER

Characteristics	Description
Unit dimensions:	
Front width	111 inches
Depth	42 inches
Height	70 inches
Service clearances:	
Front	69 inches
Rear	48 inches
Left	54 inches
Right	36 inches
Weight	1900 lbs
Heat dissipation	19,600 BTU/hour 1405 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-70%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	10-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 6.2-kva, 3-phase, 4-wire, 60-cycle
Control unit required	2840 Display Control Unit

TABLE A-11. CHARACTERISTICS OF 2301-1 DRUM STORAGE

Characteristics	Description
Unit dimensions:	
Front width	34.5 inches
Depth	29 inches
Height	64 inches
Service clearances:	
Front	48 inches
Rear	48 inches
Left	42 inches
Right	42 inches

TABLE A-11. CHARACTERISTICS OF 2301-1 DRUM STORAGE (Cont)

Characteristics	Description
Weight	850 lbs
Heat dissipation	3800 BTU/hour 320 cfm
Operating environment:*	
Temperature	60°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	0-80%
Power requirements	208v or 230v, 1.5-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	4.09 million bytes (200 tracks)
Access time:	
Average	8.6 ms
Maximum	17.5 ms
Bytes per second	1.2 million (from 2820 to CPU)
Bytes per track-density	20,486, maximum
Control unit required	2820 Storage Control Unit

* 2-hour temperature stabilization period required before power is applied.

TABLE A-12. CHARACTERISTICS OF 2302-3, -4 DISK STORAGE

Characteristics	Description
Unit dimensions:	
Front width	85.5 inches
Depth	33 inches
Height	68.75 inches
Service clearances:	
Front	60 inches
Rear	60 inches
Left	60 inches
Right	60 inches
Weight	Model 3: 4025 lbs Model 4: 4425 lbs

TABLE A-12. CHARACTERISTICS OF 2302-3, -4 DISK STORAGE (Cont)

Characteristics	Description
Heat dissipation	Model 3: 20,000 BTU/hour 2210 cfm Model 4: 28,000 BTU/hour 2210 cfm
Operating environment:	
Temperature	65°F - 90°F
Relative humidity	10-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Power requirements	208v or 230v, 9.0-kva for Model 3, 12.6-kva for Model 4, 3-phase, 4-wire, 60-cycle
Storage capacity	Model 3: 112.14 million bytes Model 4: 224.28 million bytes
Access time, maximum	180 ms
Bytes per second	156,000
Bytes per record-density	4984, maximum
Control unit required	2841 Storage Control Unit

TABLE A-13. CHARACTERISTICS OF 2311-1 DISK STORAGE DRIVE

Characteristics	Description
Unit dimensions:	
Front width	30 inches
Depth	24 inches
Height	38 inches
Service clearances:	
Front	36 inches
Rear	36 inches
Left	30 inches
Right	30 inches
Weight	390 lbs
Heat dissipation	2000 BTU/hour 100 cfm

TABLE A-13. CHARACTERISTICS OF 2311-1 DISK STORAGE DRIVE (Cont)

Characteristics	Description
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	8-80%
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	0-80%
Power requirements	208v or 230v, 0.75-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	7.25 million bytes
Bytes per second	156,000
Access time:	
Average	85 ms
Maximum	145 ms
Control unit required	2841 Storage Control Unit

TABLE A-14. CHARACTERISTICS OF 2314-1 DIRECT ACCESS STORAGE FACILITY (DISK)

Characteristics	Description
Unit dimensions	Not available
Service clearances	Not available
Weight	Not available
Heat dissipation	Not available
Operating environment	Not available
Nonoperating environment	Not available
Power requirements	Not available
Storage capacity	207 million bytes in eight modules at any one time
Access time:	
Average	75 ms
Maximum	140 ms
Bytes per second	312,000
Bytes per track-density	7188
Channel required	Selector (control unit position)

TABLE A-15. CHARACTERISTICS OF 2321-1 DATA CELL DRIVE

Characteristics	Description
Unit dimensions:	
Front width	68.5 inches
Depth	50.5 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	30 inches
Left	30 inches
Right	34 inches
Weight	1950 lbs
Heat dissipation	19,500 BTU/hour 850 cfm
Operating environment:	
Temperature	65°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 8.7-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	400 million bytes in 10 cells at any one time
Access time	175 to 600 ms
Bytes per record-density	2000 maximum
Control unit required	2841 Storage Control Unit

TABLE A-16. CHARACTERISTICS OF 2361-1, -2 CORE STORAGE

Characteristics	Description
Unit dimensions:	
Front width	64.25 inches
Depth	31.75 inches
Height	70.5 inches
Service clearances:	
Front	72 inches
Rear	42 inches
Left	36 inches
Right	30 inches

TABLE A-16. CHARACTERISTICS OF 2361-1, -2 CORE STORAGE (Cont)

Characteristics	Description
Weight	2125 lbs
Heat dissipation	24,600 BTU/hour 1095 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Power requirements	208v or 230v, 9.0-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	Model 1: 1,048,576 bytes Model 2: 2,097,152 bytes
Access time (8-byte access), maximum	8 us
Control unit required	None, direct to 2065

TABLE A-17. CHARACTERISTICS OF 2365-1, -2 PROCESSOR STORAGE

Characteristics	Description
Unit dimensions	See Table A-1
Service clearances	See Table A-1
Weight	Not available
Heat dissipation	Not available
Operating environment	Not available
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	Not available
Storage capacity	Model 1: 131,076 bytes Model 2: 262,144 bytes
Access time (8-byte access), maximum	750ns
Control unit required	None, direct to 2065

TABLE A-18. CHARACTERISTICS OF 2401-1, -2, -3 MAGNETIC TAPE UNIT

Characteristics	Description
Unit dimensions:	
Front width	30 inches
Depth	29 inches
Height	60 inches
Service clearances:	
Front	36 inches
Rear	36 inches
Left	30 inches (when not abutted to another tape or control unit)
Right	30 inches (when not abutted to another tape or control unit)
Weight	800 lbs
Heat dissipation	3500 BTU/hour 500 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements (from control unit)	208v or 230v, 1.6-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	0.5-inch magnetic tape
Bytes per second	Model 1: 30,000 Model 2: 60,000 Model 3: 90,000
Bytes per inch-density	Model 1: 800 Model 2: 800 Model 3: 800
Tape speed (inches per second)	Model 1: 37.5 Model 2: 75.0 Model 3: 112.5
Rewind and unload time (minutes)	Model 1: 2.2 Model 2: 1.5 Model 3: 1.1
Interrecord gap	Model 1: 0.6 inch 16.0 ms Model 2: 0.6 inch 8.0 ms Model 3: 0.6 inch 5.3 ms
Control unit required	2403, 2404, 2803, or 2804 Control Unit

TABLE A-19. CHARACTERISTICS OF 2402-1, -2, -3 MAGNETIC TAPE UNIT*

Characteristics	Description
Unit dimensions:	
Front width	60 inches
Depth	29 inches
Height	60 inches
Service clearances:	
Front	36 inches
Rear	36 inches
Left	30 inches (when not abutted to another tape or control unit)
Right	30 inches (when not abutted to another tape or control unit)
Weight	1600 lbs
Heat dissipation	7000 BTU/hour 1000 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements (from control unit)	208v or 230v, 3.2-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	0.5-inch magnetic tape
Bytes per second	Model 1: 30,000 Model 2: 60,000 Model 3: 90,000
Bytes per inch-density	Model 1: 800 Model 2: 800 Model 3: 800
Tape speed (inches per second)	Model 1: 37.5 Model 2: 75.0 Model 3: 112.5
Rewind and unload time (minutes)	Model 1: 2.2 Model 2: 1.5 Model 3: 1.1
Interrecord gap	Model 1: 0.6 inch 16.0 ms Model 2: 0.6 inch 8.0 ms Model 3: 0.6 inch 5.3 ms
Control unit required	2403, 2404, 2803, or 2804 Control Unit

*Consists of two independent drives in one frame.

TABLE A-20. CHARACTERISTICS OF 2403 AND 2404-1, -2, -3 MAGNETIC TAPE UNIT AND CONTROL

Characteristics	Description
Unit dimensions:	
Front width	60 inches
Depth	29 inches
Height	60 inches
Service clearances:	
Front	42 inches
Rear	42 inches
Left	30 inches (when not abutted to another tape or control unit)
Right	30 inches (when not abutted to another tape or control unit)
Weight	2000 lbs
Heat dissipation	For 2403: 5500 BTU/hour 1000 cfm For 2404: 6300 BTU/hour 1200 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 2.1-kva for 2403, 2.4-kva for 2402, 3-phase, 4-wire, 60-cycle
Storage capacity	0.5-inch magnetic tape
Bytes per second	Model 1: 30,000 Model 2: 60,000 Model 3: 90,000
Bytes per inch-density	Model 1: 800 Model 2: 800 Model 3: 800
Tape speed (inches per second)	Model 1: 37.5 Model 2: 75.0 Model 3: 112.5
Rewind and unload time (minutes)	Model 1: 2.2 Model 2: 1.5 Model 3: 1.1
Interrecord gap	Model 1: 0.6 inch 16.0 ms Model 2: 0.6 inch 8.0 ms Model 3: 0.6 inch 5.3 ms
Type of channel required	For 2403: Multiplexor or selector (control unit position) For 2404: Multiplexor or selector (control unit position)
Number of drives controlled	8 maximum

TABLE A-21. CHARACTERISTICS OF 2501-B1, -B2 CARD READER

Characteristics	Description
Unit dimensions:	
Front width	30 inches
Depth	24 inches
Height	44.5 inches
Service clearances:	
Front	36 inches
Rear	42 inches
Left	6 inches
Right	24 inches
Weight	290 lbs
Heat dissipation	2700 BTU/hour
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	10-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 0.5-kva, 1-phase, 3-wire, 60-cycle
Storage capacity	1200 cards in hopper 1300 cards in stacker
Read rate (cards per minute)	Model B1: 600 Model B2: 1000
Channel required	Multiplexor or selector (control unit position)

TABLE A-22. CHARACTERISTICS OF 2520-B1 CARD READ PUNCH; 2520-B2, -B3 CARD PUNCH

Characteristics	Description
Unit dimensions:	
Front width	43 inches
Depth	24 inches
Height	50 inches

TABLE A-22. CHARACTERISTICS OF 2520-B1 CARD READ PUNCH; 2520-B2, -B3 CARD PUNCH (Cont)

Characteristics	Description
Service clearances:	
Front	48 inches
Rear	36 inches
Left	36 inches
Right	18 inches
Weight	660 lbs
Heat dissipation	6350 BTU/hour 75 cfm
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	10-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.85-kva, 1-phase, 3-wire, 60-cycle
Storage capacity	1200 cards in hopper 1300 cards in stacker
Read rate (cards per minute)	Model B1: 500 Model B2: - Model B3: -
Punch rate (cards per minute)	Model B1: 500 Model B2: 500 Model B3: 300
Channel required	Multiplexor or selector (control unit position)

TABLE A-23. CHARACTERISTICS OF 2540-1 CARD READ PUNCH

Characteristics	Description
Unit dimensions:	
Front width	57.50 inches
Depth	29.25 inches
Height	45.25 inches (plus 20 inches for read file feed feature)

TABLE A-23. CHARACTERISTICS OF 2540-1 CARD READ PUNCH (Cont)

Characteristics	Description
Service clearances:	
Front	36 inches
Rear	36 inches
Left	36 inches
Right	36 inches
Weight	1050 lbs
Heat dissipation	3000 BTU/hour 50 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements (from 2821 Control Unit)	1.2-kva
Storage capacity	3100 cards in hopper 1350 cards in each of five stackers
Read rate (cards per minute)	1000
Punch rate (cards per minute)	300
Control unit required	2821-1, -5 Control Unit

TABLE A-24. CHARACTERISTICS OF 2701-1 DATA ADAPTER UNIT

Characteristics	Description
Unit dimensions:	
Front width	40 inches
Depth	25.5 inches
Height	40 inches
Service clearances:	
Front	42 inches
Rear	42 inches
Left	0 inch
Right	42 inches

TABLE A-24. CHARACTERISTICS OF 2701-1 DATA ADAPTER UNIT (Cont)

Characteristics	Description
Weight	320 lbs
Heat dissipation	1200 BTU/hour 120 cfm
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 0.3-kva, 1-phase, 3-wire, 60-cycle
Bauds per second	2 million, maximum*
Channel required	Multiplexor or selector (control unit position)

* Can handle 48 parallel data bits.

TABLE A-25. CHARACTERISTICS OF 2702-1 TRANSMISSION CONTROL

Characteristics	Description
Unit dimensions:	
Front width	28.75 inches
Depth	61.50 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	18 inches
Left	30 inches
Right	42 inches
Weight	900 lbs
Heat dissipation	1800 BTU/hour 800 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F

TABLE A-25. CHARACTERISTICS OF 2702-1 TRANSMISSION CONTROL (Cont)

Characteristics	Description
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 2.0-kva, 1-phase, 3-wire, 60-cycle
Bits per second:	180, maximum*
Channel required	Multiplexor

* Up to 31 lines may be connected.

TABLE A-26. CHARACTERISTICS OF 2802-1 HYPERTAPE CONTROL

Characteristics	Description
Unit dimensions:	
Front width	28.75 inches
Depth	61.5 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	30 inches
Left	42 inches
Right	42 inches
Weight	550 lbs
Heat dissipation	1360 BTU/hour 300 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	90°F
Power requirements	208v or 230v, 0.6-kva (plus 4.0 kva per 7340-3 attached), 3-phase, 4-wire, 60-cycle

TABLE A-26. CHARACTERISTICS OF 2802-1 HYPERTAPE CONTROL (Cont)

Characteristics	Description
Bytes per second:	
Low	170,000
High	340,000
Channel required:	
Low	Selector or multiplexor
High	Selector only
Number and type of units controlled	Maximum of eight 7340-3's *

* This unit may handle up to 16 tape drives with two 2816-2 units.

TABLE A-27. CHARACTERISTICS OF 2803-1 AND 2804-1 TAPE CONTROL*

Characteristics	Description
Unit dimensions:	
Front width	60 inches
Depth	29 inches
Height	60 inches
Service clearances:	
Front	42 inches
Rear	42 inches
Left	30 inches
Right	30 inches (when not abutted to another tape or control unit)
Weight	For 2803: 1400 lbs For 2804: 1600 lbs
Heat dissipation	For 2803: 2500 BTU/hour 500 cfm For 2804: 4000 BTU/hour 700 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.0-kva for 2803, 1.5-kva for 2804, 3-phase, 4-wire, 60-cycle
Bytes per second:	Determined by tape drive

TABLE A-27. CHARACTERISTICS OF 2803-1 AND 2804-1 TAPE CONTROL* (Cont)

Characteristics	Description
Channel required	For 2803: Multiplexor or selector (control unit position) For 2804: Multiplexor or selector (control unit position)
Number and type of units controlled	Up to 16 2401's or 2402's

* 2803 is a single-channel, read or write control
2804 is a two-channel, simultaneous-read-while-write control

TABLE A-28. CHARACTERISTICS OF 2816-1, -2 SWITCHING UNIT

Characteristics	Description
Unit dimensions:	
Front width	29 inches
Depth	42 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	18 inches
Left	42 inches
Right	42 inches
Weight	500 lbs
Heat dissipation	1500 BTU/hour 280 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.2-kva, 1-phase, 3-wire,
Control unit required	Tape Control Unit
Number of units controlled	Up to 16 drives (with two 2816's)

TABLE A-29. CHARACTERISTICS OF 2820-1 STORAGE CONTROL (DRUM)

Characteristics	Description
Unit dimensions:	
Front width	28.75 inches
Depth	61.50 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	30 inches
Left	42 inches
Right	42 inches
Weight	750 lbs
Heat dissipation	4000 BTU/hour 550 cfm
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.5-kva, 1-phase, 3-wire, 60-cycle
Bytes per second	1.2 million transfer rate
Channel required	Selector (control unit position)
Number and type of units controlled	Up to four 2301's

TABLE A-30. CHARACTERISTICS OF 2821-1, -2, -3, -5 CONTROL UNIT (CARD)

Characteristics	Description
Unit dimensions:	
Front width	32 inches
Depth	46 inches
Height	60 inches

TABLE A-30. CHARACTERISTICS OF 2821-1, -2, -3, -5 CONTROL UNIT (CARD) (Cont)

Characteristics	Description
Service clearances:	
Front	30 inches
Rear	18 inches
Left	48 inches
Right	48 inches
Weight	1000 lbs
Heat dissipation	7000 BTU/hour 300 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 2.4-kva, 3-phase, 4-wire, 60-cycle
Channel required	Multiplexor or selector (control unit position)
Type of units controlled	2540 and 1403

TABLE A-31. CHARACTERISTICS OF 2840-1 DISPLAY CONTROL

Characteristics	Description
Unit dimensions:	
Front width	29 inches
Depth	42 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	30 inches
Left	30 inches
Right	30 inches
Weight	550 lbs
Heat dissipation	4800 BTU/hour 300 cfm

TABLE A-31. CHARACTERISTICS OF 2840-1 DISPLAY CONTROL (Cont)

Characteristics	Description
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 150°F
Relative humidity	8-80%
Wet bulb temperature	85°F
Power requirements	208v or 230v, 1.4-kva, 1-phase, 3-wire, 60-cycle
Storage capacity	Up to 16,384 bytes
Access time (2 bytes)	4.2 ms
Channel required	Multiplexor or selector (control unit position)
Number and type of units controlled	Up to eight 2250-2's or up to four 2280's, 2281's or 2282's

TABLE A-32. CHARACTERISTICS OF 2841-1 STORAGE CONTROL

Characteristics	Description
Unit dimensions:	
Front width	32 inches
Depth	45.5 inches
Height	60 inches
Service clearances:	
Front	30 inches
Rear	30 inches
Left	48 inches
Right	30 inches
Weight	750 lbs
Heat dissipation	5500 BTU/hour 1000 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F

TABLE A-32. CHARACTERISTICS OF 2841-1 STORAGE CONTROL (Cont)

Characteristics	Description
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.9-kva, 3-phase, 4-wire, 60-cycle
Bytes per second	Up to 156,000 (depends on storage unit)
Channel required	Multiplexor or selector (control unit position)
Number and type of units controlled (8 access mechanisms maximum)	Two access mechanisms per 2302-3, four per 2302-4, eight per 2311-1, one per 2321-1, and one per 7320-4

TABLE A-33. CHARACTERISTICS OF 2848-1, -2, -3 DISPLAY CONTROL

Characteristics	Description
Unit dimensions:	
Front width	29 inches
Depth	60.75 inches
Height	72.5 inches
Service clearances:	
Front	30 inches
Rear	30 inches
Left	48 inches
Right	48 inches
Weight	1000 lbs
Heat dissipation	3542 BTU/hour
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.5-kva, 1-phase, 3-wire, 60-cycle
Characters per second	2560 maximum

TABLE A-33. CHARACTERISTICS OF 2848-1, -2, -3 DISPLAY CONTROL (Cont)

Characteristics	Description
Channel required	Multiplexor or selector (control unit position)
Number and type of units controlled	Model 1: up to 24 2260's Model 2: up to 16 2260's Model 3: up to 8 2260's

TABLE A-34. CHARACTERISTICS OF 2860-1, -2, -3 SELECTOR CHANNEL*

Characteristics	Description
Unit dimensions:	
Front width	32.25 inches
Depth	67.75 inches
Height	71 inches
Service clearances:	
Front	30 inches
Rear	51 inches
Left	66 inches
Right	66 inches
Weight	Model 1: 1150 lbs Model 2: 1450 lbs Model 3: 1750 lbs
Heat dissipation	Model 1: 8200 BTU/hour 420 cfm Model 2: 10,000 BTU/hour 740 cfm Model 3: 11,600 BTU/hour 1060 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	20-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 3.05-kva for Model 1, 3.65-kva for Model 2, 4.25-kva for Model 3, 3-phase, 4-wire, 60-cycle
Bytes per second	Up to 1.3 million
Control required	None, direct to 2065 CPU
Number and type of units controlled	Up to eight control units per channel Model 1: one channel = 8 control units Model 2: two channels = 16 control units Model 3: three channels = 24 control units

* Two of these units may be connected to 2065 CPU

TABLE A-35. CHARACTERISTICS OF 2870-1 MULTIPLEXOR CHANNEL

Characteristics	Description
Unit dimensions:	
Front width	32.75 inches
Depth	67.75 inches
Height	71 inches
Service clearances:	
Front	30 inches
Rear	51 inches
Left	66 inches
Right	66 inches
Weight	1450 lbs
Heat dissipation	11,600 BTU/hour 1060 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	20-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 4.25-kva, 3-phase, 4-wire, 60-cycle
Aggregate data rate:	Up to 450 kc
Control required	None, direct to 2065 CPU
Number and type of units controlled	Up to 190 I/O units

TABLE A-36. CHARACTERISTICS OF 7320-1 DRUM STORAGE

Characteristics	Description
Unit dimensions:	
Front width	30 inches
Depth	29 inches
Height	60 inches

TABLE A-36. CHARACTERISTICS OF 7320-1 DRUM STORAGE (Cont)

Characteristics	Description
Service clearances:	
Front	40 inches
Rear	40 inches
Left	42 inches
Right	42 inches
Weight	850 lbs
Heat dissipation	2800 BTU/hour 320 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	8-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	0-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.1-kva, 3-phase, 4-wire, 60-cycle
Storage capacity	830,000 bytes on 400 tracks
Access time:	
Average	8.6 ms
Maximum	17.5 ms
Bytes per second:	136,000
Bytes per track-density	2075
Control unit required	2841 Storage Control Unit

TABLE A-37. CHARACTERISTICS OF 7340-3 HYPERTAPE DRIVE

Characteristics	Description
Unit dimensions:	
Front width	29 inches
Depth	60 inches
Height	48 inches (plus 22 inches if autoloader is installed)

TABLE A-37. CHARACTERISTICS OF 7340-3 HYPERTAPE DRIVE (Cont)

Characteristics	Description
Service clearances:	
Front	46 inches
Rear	52 inches
Left	See Note
Right	See Note
Weight	1500 lbs (plus 250 lbs if autoloader is installed)
Heat dissipation	12,000 BTU/hour 700 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	8-80%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 4.0-kva (from control unit), 3-phase, 4-wire, 60-cycle
Bytes per second:	
Low	170,000
High	340,000
Bytes per inch-density:	
Low	1511
High	3022
Access time	3.5 ms average
Tape speed	112.5 inches per second
Rewind time	1.5 minutes
Interrecord gap	0.38 inch
Control unit required	2802 Hypertape Control Unit

Note: 7 inches minimum for two 7340's clearances should alternate: 7, 22, 7, 22 inches, etc. 30 inches minimum between any other units.

TABLE A-38. CHARACTERISTICS OF 7770-3 AUDIO RESPONSE UNIT

Characteristics	Description
Unit dimensions:	
Front width	37.50 inches (73.5 inches with expander)
Depth	31.50 inches
Height	70 inches
Service clearances:	
Front	42 inches
Rear	36 inches
Left	30 inches
Right	30 inches
Weight	For 16-line unit: 500 lbs For 48-line unit: 1000 lbs
Heat dissipation	For 16-line unit: 3000 BTU/hour 400 cfm For 48-line unit: 6000 BTU/hour 800 cfm
Operating environment:	
Temperature	60°F - 90°F
Relative humidity	20-80%
Wet bulb temperature	78°F
Power requirements	208v or 230v, 1.6-kva for 16 lines, 2.5-kva for 48 lines, 1-phase, 3-wire, 60-cycle
Storage capacity	128 words
Channel required	Multiplexor

TABLE A-39. CHARACTERISTICS OF 7772-3 AUDIO RESPONSE UNIT

Characteristics	Description
Unit dimensions:	
Front width	37.50 inches (73.5 inches with expander)
Depth	31.50 inches
Height	70 inches

TABLE A-39. CHARACTERISTICS OF 7772-3 AUDIO RESPONSE UNIT (Cont)

Characteristics	Description
Service clearances:	
Front	42 inches
Rear	36 inches
Left	30 inches
Right	30 inches
Weight	For 4-line unit: 600 lbs For 8-line unit: 1000 lbs
Heat dissipation	For 4-line unit: 5100 BTU/hour 900 cfm For 8-line unit: 7700 BTU/hour 1800 cfm
Operating environment:	
Temperature	50°F - 90°F
Relative humidity	10-90%
Wet bulb temperature	78°F
Nonoperating environment:	
Temperature	50°F - 110°F
Relative humidity	0-90%
Wet bulb temperature	80°F
Power requirements	208v or 230v, 1.0-kva for 4 lines, 1.5-kva for 8 lines, 1-phase, 3-wire, 60-cycle
Channel required	Multiplexor (control unit position)

APPENDIX B

CONTROLS AND INDICATORS

This appendix defines the functions of the controls and indicators on the system control panel and the CE panel. These two panels contain all the controls and indicators required for system operation.

B.1 SYSTEM CONTROL PANEL

The system control panel is divided into seven separate panels, as shown in Figure 3-22.

B.1.1 PANEL A

1. Meter. The meter indicates the voltage levels of the marginable supplies. The particular supply indicated is determined by the MARGIN/METER SEL switch.

2. MARGIN/METER SEL switch. This switch has 12 positions to select the power supply to be indicated by the meter and to determine which of the attached stand-alone units may be marginally checked:

   a. STORE FRAME 1: Indicates and selects storage unit 1.

   b. STORE FRAME 2: Indicates and selects storage unit 2.

   c. STORE FRAME 3: Indicates and selects storage unit 3.

   d. STORE FRAME 4: Indicates and selects storage unit 4.

   e. CHAN FRAME 1: Indicates and selects channel 1.

   f. CHAN FRAME 2: Indicates and selects channel 2.

   g. CHAN FRAME 3: Indicates and selects channel 3.

   h. ROS LOCATE: Indicates gate D in the CPU (ROS gate).

   i. CPU A: Indicates gate A in the CPU.

   j. CPU B: Indicates gate B in the CPU.

   k. CPU C: Indicates gate C in the CPU.

   l. CPU E: Indicates gate E in the CPU.

3. MARGIN indicators

   a. ACTIVE: Indicates that an internal power supply or an attached storage unit or channel is being marginally checked.

   b. LOCATE: Indicates when the MARGIN/METER SEL switch is at the position of a margined power supply or attached storage unit or channel.

4. POWER CHECK indicators: These eight indicators, CPU, STOR FRAME 1, 2, 3, and 4, and CHAN FRAME 1, 2, and 3, indicate an incomplete power-up status in the CPU, storage units 1, 2, 3, and 4, and channels 1, 2, and 3, respectively.

5. MARGIN CHANNEL/STOR switch: This switch applies power to a motor in the channel or storage unit selected by the MARGIN/METER SEL switch to lower or raise the output voltage levels from the marginable supplies in that unit or channel.

B.1.2 PANEL B

The five controls on this panel, ROS, +6M A GT, +6M B GT, +6M C GT, and +6M E GT, raise or lower the output voltage levels from the 18v ROS and the 6v gate A, B, C, and E supplies, respectively.

B.1.3 PANEL C

The pull switch on this panel, EMERGENCY PULL, initiates emergency power off in the system when it is pulled.

B.1.4 PANEL D

This panel is blank at present.

B.1.5 PANEL E

1. Roller switches and indicators. This section of the panel contains six roller switches and associated indicators. Figure 9058, FEDM identifies the indicators for the six positions of each roller switch. The roller switch indicators are tested between positions of the switch. Position 6 of roller 6 is used to test the remaining indicators on the system control panel and on the 2150 Console.

2. DATA 0-31 and DATA 32-63 data switches: These 64 switches, in hexadecimal groups, permit data to be entered manually. Correct parity is automatically generated.

3. ADDRESS switches: These 24 switches, in hexadecimal groups, select an addressable location in storage. Correct parity is automatically generated.

4. STOR CHK (storage check) indicator: Indicates an error in the storage units.

5. PROC CHK (processor check) indicator: Indicates an error in the CPU.

B.1.6 PANEL F

1. TEST MODE switches

   a. REPEAT switch: Repeats the ROS or FLT test in storage continuously.

   b. FLT switch: Places the CPU in the FLT mode and removes program control.

   c. ROS switch: Provides for checking each bit in ROS against a test tape.

2. FREQUENCY ALTERATION switch: Decreases the CPU clock cycle from 200 ns to 195 ns. Operates only with the CE key switch in the CE position.

3. DEFEAT INTERLEAVING switch. This switch has three positions:

   a. PROC (process) — normal position: Addressing is interleaved with no address reversal.

   b. REV (reverse): Interleaving is disabled with addresses reversed.

   c. NO REV (no reverse): Interleaving is disabled with no address reversal.

4. STOP ON STORAGE CHECK switch: Inhibits operation and maintains environment of storage upon detection of a storage error.

5. DISABLE INTERVAL TIMER switch: Prevents the interval timer from decrementing.

6. STORAGE SELECT switch. This switch has three positions:

   a. MAIN — normal position: Selects main storage for storing or displaying data.

   b. LOCAL: Selects local storage for storing or displaying data.

   c. MAIN BYTE: Same as the normal position except that the byte selected is the only byte affected by a manual store operation.

7. ADDRESS COMPARE STOP switch: Stops processing if the storage address agrees with bits 2 through 20 of the ADDRESS switches.

8. CPU CHECK switch. This switch has three positions:

   a. PROC (process) — normal position: If the PSW machine check mask is a 1, the CPU stops on detection of a CPU check and the status is logged into main storage. If the mask is a 0, the result is the same as if the switch is in the DSBL position.

   b. DSBL (disable): The CPU does not stop on detection of a machine check, but the check trigger is set.

   c. STOP: The CPU stops on detection of a machine check, but there is no log-in of data.

9. PULSE MODE switch. This switch has three positions:

   a. PROC (process) — normal position: Does not affect CPU operation.

   b. COUNT: Provides a means of looping through a selected number of machine cycles (maximum of 2047). The number of cycles is entered in DATA switches 53-63. Each loop starts at the address contained in main storage address zero.

c. TIME: Provides looping when the interval timer is decremented. Each loop starts at the address contained in main storage address zero.

10. REPEAT INSN (instruction) switch. This switch has three positions:

    a. PROC (process) — normal position: Does not affect CPU operation.

    b. SINGLE: Allows the first instruction in the DATA switch to be repeated continuously.

    c. MPLE (multiple): Allows looping through the four instruction halfwords in the DATA switches continuously.

11. REPEAT ROS ADDRESS switch: Continually reads out the ROS address specified in ADDRESS switches 0-11. The ROS TRANSFER pushbutton must be depressed to start this loop.

12. RATE switch. This switch has four positions:

    a. INSN STEP (instruction step): CPU executes one machine instruction for each depression of the START pushbutton.

    b. PROCESS: Does not affect CPU operation; CPU operates at normal clock speed.

    c. SINGLE CYCLE: CPU advances by its minimum clock amount for each depression of the START pushbutton; all CPU operations are as in PROC position.

    d. SINGLE CYCLE STORAGE INHIBIT: Same as SINGLE CYCLE position without storage references.

13. SYSTEM RESET pushbutton: Resets on-line channels, control units, and CPU controls, including machine checks, to their initial state.

14. CHECK RESET pushbutton. Resets all CPU and storage check triggers.

15. PSW RESTART pushbutton. Loads a PSW from main storage address zero and stars processing.

16. ROS TRANSFER pushbutton. Provides a means of visually interrogating the contents of an ROS location or of beginning processing from any ROS address.

17. SET IC (instruction counter) pushbutton. Enters an address from the ADDRESS switches into the active (current) PSW.

18. STORE pushbutton: Enters data into the storage location specified by the STORAGE SELECT and ADDRESS switches.

19. DISPLAY pushbutton: Displays data specified by the STORAGE SELECT and ADDRESS switches.

20. START pushbutton: Starts the CPU operating in the mode selected by the RATE switch.

21. STOP pushbutton: Terminates CPU operation without changing environment.

22. RESTART FLT I/O pushbutton: Backspaces one tape record and starts reading during FLT Test Mode operation.

23. LOG OUT pushbutton: Stores CPU status in fixed locations in main storage.

24. Elapsed-time meters and CE key switch: The elapsed-time meters indicate elapsed CPU running time: the process meter shows customer elapsed time; the CE meter shows customer engineering elapsed time. The key switch determines the meter used.

B.1.7 PANEL G

1. POWER ON pushbutton: Initiates power on in the CPU and the system units as defined in Chapter 5.

2. POWER OFF pushbutton: Initiates power off in the CPU and the system units as defined in Chapter 5.

3. LOAD UNIT switches: These three switches select the I/O unit used by a load operation.

4. INTERRUPT pushbutton: Causes an external interruption in the system and sets bit 25 of the interruption code to a 1.

5. LOAD pushbutton: Resets the system and starts a load operation.

6. SYSTEM indicator: Indicates a CPU elapsed-time meter is running.

7. MANUAL indicator: Indicates CPU is in the stopped state.

8. WAIT indicator: Indicates CPU is in the wait state.

9. TEST indicator: Indicates that a switch on panel F is not in the normal operating position or that a channel is in the test mode.

10. LOAD indicator: Indicates a CPU load operation. A successful load turns indicator off.

B.2 CE PANEL

The CE panel is shown in Figure B-1. The panel indicators and controls are as follows:

1. THERMAL RESET pushbutton: Resets the thermal sense relays in the CPU.

2. CPU READY switch. This switch has two positions:

   a. READY — normal position: Allows CPU power-up sequencing to continue if the thermal and overcurrent conditions are normal.

   b. OFF: Drops CPU power without affecting system power.

3. CPU ON pushbutton: Starts CPU power-on sequencing if the CPU READY switch is in the READY position. Does not affect system power.

4. THERMAL TRIP indicators: These six indicators show the location of the overtemperature condition that dropped CPU power. The temperature sensors are located in gates A, B, C/D, and E, the converter/inverter, and the power supplies tub.

5. UNDER VOLTAGE CHECK switches (located on the relay gate below the CE panel in the converted units): These 15 switches isolate the power supplies from the undervoltage sensing circuits. See paragraph 5.3.8 for details.

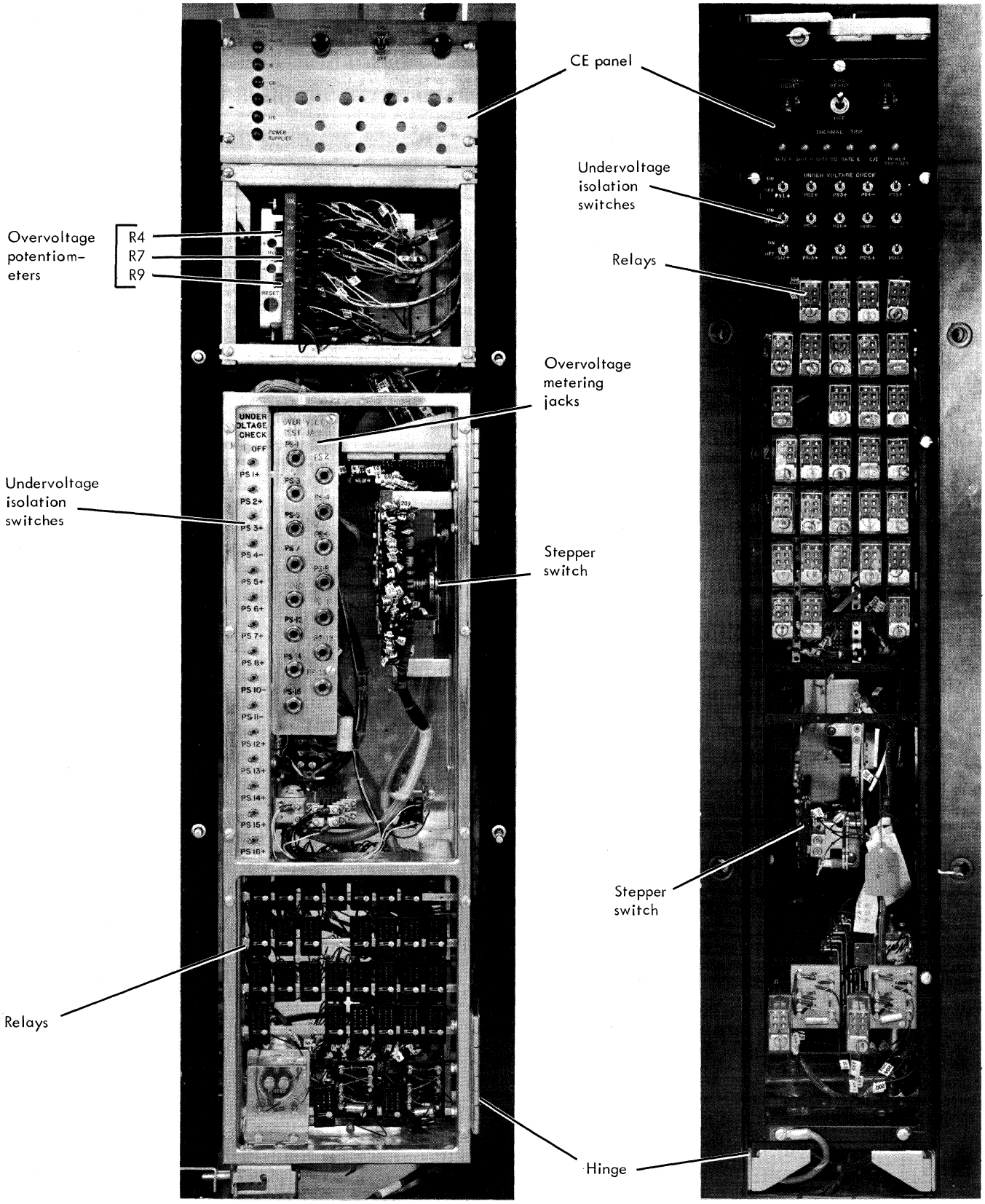


FIGURE B-1. CE PANEL

APPENDIX C

CE OPERATIONS

C.1 INTRODUCTION

This appendix outlines the steps to operate the system for normal powering, processing, and testing.

Normal processing operations described in this appendix are limited to the use of the switches and pushbuttons on the system control panel.

Testing operations described in this appendix are limited to the Fault Locating Tests (FLT). These tests isolate to an average of four small cards on solid failures and to an average of 100 logic blocks on intermittent failures, provided the frequency of the failure is equal to or less than the time it takes to run the applicable test. See Appendix B and the 2065 Processing Unit, FEMM, Form 226-2039, for details of the controls and indicators for program diagnostic aids.

At least one of the following input units is required with the associated test formats:

1. 2400 series tape unit and control unit
2. 7340 Hypertape Drive and control unit
3. 2311 Disk Storage Drive and control unit

At least one of the following output units is required:

1. 1052 Printer-Keyboard
2. 1403 Printer and control unit
3. 1443 Printer and control unit

C.2 POWER OPERATIONS

The procedures in paragraphs C.2.1 and C.2.2 are required only after initial installation of the system and of the primary ac power or in the event of a change in the system configuration.

C.2.1 ASSOCIATED STAND-ALONE UNITS INITIAL POWER

The following steps are required at each unit in the system, except the CPU:

1. Close all internal circuit breakers.
2. Close associated wall primary power switch.
3. Set LOCAL/REMOTE switch to REMOTE position.

C.2.2 CPU INITIAL POWER

The following steps are required at the CPU:

1. Close all internal circuit breakers.
2. Close associated wall primary power switch.
3. Set CPU READY switch (on CE panel) to READY position.

C.2.3 SYSTEM POWER ON

After the procedures in paragraphs C.2.1 and C.2.2 have been completed, the only step required to turn system power on is to depress the POWER ON pushbutton at either the system control panel or the 2150 Console.

C.2.4 SYSTEM POWER OFF

Depress the POWER OFF pushbutton at either the system control panel or the 2150 Console.

C.3 NORMAL PROCESSING OPERATIONS

Normal processing operations consist of the following procedures:

1. System Resetting
2. Check Logic Resetting
3. New Program Entry
4. Program Restart
5. Instruction Address Change
6. Data Entry or Modification

7. Data Display

8. External Interrupting

9. Terminating Machine Operations

C.3.1 SYSTEM RESETTING

The initial condition of the system is the stopped state with all channel, control unit, and CPU controls, including machine check logic, in the normal state. This condition is automatically set during normal power-on sequencing and may be manually set by depressing the SYSTEM RESET pushbutton.

C.3.2 CHECK LOGIC RESETTING

To restore the CPU error check logic to the normal or nonerror state, depress the CHECK RESET pushbutton. All CPU check indicators will clear, and processing will resume if it stopped because of a machine check; the results, however, may be inaccurate.

C.3.3 NEW PROGRAM ENTRY

To enter a new program, set the address of the input unit (e.g., tape or disk), in the LOAD UNIT rotary switches and depress the LOAD pushbutton. The left LOAD UNIT switch selects the channel (0 through 7), and the remaining two switches select the unit (00 through FF). The LOAD pushbutton causes a system reset and then loads the first 24 bytes from the input unit into main storage locations 0 through 23. These first 24 bytes contain the information required to store the remainder of the program. The CPU then starts to execute the program.

C.3.4 PROGRAM RESTART

To restart a program, depress the PSW RESTART pushbutton. The CPU takes the PSW in main storage location 0 for the one to continue processing with. This PSW contains the address of the next instruction to be executed. If processing was stopped because of an interruption, the processing will resume where it left off. If processing was stopped because the STOP pushbutton was depressed, the processing will return to the first instruction in the program, unless previous interruptions occurred.

C.3.5 INSTRUCTION ADDRESS CHANGE

To change the address of the instruction to be called by the current PSW, set the new address in ADDRESS switches 0-23 and depress the SET IC pushbutton. To commence processing with the new instruction, depress the START pushbutton.

C.3.6 DATA ENTRY OR MODIFICATION

Data may be entered in a specific location of either main or local storage from the DATA switches. The STORAGE SELECT and ADDRESS switches determine the location.

If the STORAGE SELECT switch is in the MAIN position, the contents of DATA switches 0-63 (plus automatically generated parity bits) will be entered in the main storage address selected by ADDRESS switches 0-20 when the STORE pushbutton is depressed.

If the STORAGE SELECT switch is in the LOCAL position, the contents of DATA switches 32-63 (plus parity) will be entered in the local storage address selected by ADDRESS switches 19-23 when the STORE pushbutton is depressed.

If the STORAGE SELECT switch is in the MAIN BYTE position, the contents of one byte group of the DATA switches (plus parity), selected by ADDRESS switches 21, 22, and 23 (hexadecimal), will be entered in the main storage address selected by ADDRESS switches 0-20 when the STORE pushbutton is depressed.

C.3.7 DATA DISPLAY

The contents of a specific location in either main or local storage may be displayed by the register indicators. Set roller switches 1, 2, 3, and 4 to position 3 to view the contents of ST and AB.

If the STORAGE SELECT switch is in the MAIN position, the contents of the main storage address selected by ADDRESS switches 0-20 are displayed in ST and AB when the DISPLAY pushbutton is depressed.

If the STORAGE SELECT switch is in the LOCAL position, the contents of the local storage address selected by ADDRESS switches 19-23 are displayed in T when the DISPLAY pushbutton is depressed.

If the STORAGE SELECT switch is in the MAIN BYTE position, the display is the same as for the MAIN position of the switch.

C.3.8 EXTERNAL INTERRUPTING

To stop the processing at a normal program interruption recognition point, depress the INTERRUPT pushbutton.

C.3.9 TERMINATING MACHINE OPERATIONS

To stop all machine operations without destroying the machine environment, depress the STOP pushbutton. The CPU will complete the instruction or I/O operation in process and enter the stopped state. Depressing the START pushbutton will cause the system to continue as if the STOP pushbutton had not been depressed.

C.4 TESTING OPERATIONS

The following tests locate hardware faults and are not of a program diagnostic nature. These tests should be run in the order shown unless the specific area of trouble is known. See the 2065 Processing Unit, FEMM, Form 226-2039, for test details and trouble-locating.

1. Storage Ripple Tests
2. ROS Repeat Tests
3. ROS Hardcore Tests
4. ROS Bit Tests
5. FLT Hardcore Tests
6. FLT Scan-In/Scan-Out Tests
7. FLT One-Cycle Tests

C.4.1 STORAGE RIPPLE TESTS

These tests first store data in all locations in main or local storage and then display the contents of all locations for checking.

C.4.1.1 Storage Ripple Store

1. Depress SYSTEM RESET pushbutton.
2. Set STORAGE SELECT switch to either MAIN or LOCAL position.
3. Enter desired data in DATA switches 0-63.
4. Enter 800006 (hexadecimal) in ADDRESS switches 0-23.
5. Depress ROS TRANSFER pushbutton.

Note

STOR CHK indicator should not come on.

6. Wait several seconds, then depress SYSTEM RESET pushbutton.

Note

STORAGE SELECT switch must be set to LOCAL position before SYSTEM RESET pushbutton is depressed or storage errors may occur.

The data entered in the DATA switches will have been stored in all storage locations several times; the system reset halted the loop.

C.4.1.2 Storage Ripple Display

1. Depress SYSTEM RESET pushbutton.
2. Set STORAGE SELECT switch to either MAIN or LOCAL position.
3. Enter 800000 (hexadecimal) in ADDRESS switches 0-23.
4. Depress ROS TRANSFER pushbutton.

Note

STOR CHK and PROC CHK indicators should not come on.

5. Set roller switches 3 and 4 to position 3 to view main storage in AB. Set roller switch 1 to position 3 and roller switch 6 to position 1 to view local storage in S and PAL.
6. If displayed data agrees with stored data, depress SYSTEM RESET pushbutton to halt display loop.

Note

STORAGE SELECT switch must be set to LOCAL position before SYSTEM RESET pushbutton is depressed, or storage errors may occur.

C.4.2 ROS REPEAT TESTS

1. Depress SYSTEM RESET pushbutton.
2. Enter 000 (hexadecimal) in ADDRESS switches 0-11.
3. Set REPEAT ROS ADDRESS switch.
4. Depress ROS TRANSFER pushbutton.
5. Set roller switches 1, 2, 3, and 4 to position 4.
   a. ROSAR should contain all 0's.
   b. ROSDR should contain all 0's.
   c. All ROS parity errors should be 1 (lit).
6. Depress SYSTEM RESET pushbutton.
7. Enter 801 (hexadecimal) in ADDRESS switches 0-11.
8. Depress ROS TRANSFER pushbutton.
9. View same registers as before.
   a. ROSAR should contain 801 801 801.
   b. ROSDR should contain all 1's.
   c. All ROS parity errors should be 0.
10. Depress SYSTEM RESET pushbutton.
11. Restore REPEAT ROS ADDRESS switch to normal position.

C.4.3 ROS HARDCORE TESTS

1. Load test tape or disk pack in appropriate input unit.
2. Set LOAD UNIT switches to address of input unit.
3. Set DATA switches 32-63 to 1's.
4. Depress STORE pushbutton.
5. Set TEST MODE ROS switch.
6. Set CPU CHECK switch to DSBL (disable) position.
7. Depress LOAD pushbutton. Machine should stop immediately.
8. Set roller switches 1 and 2 to position 3 and roller switch 5 to position 2:
   a. S should contain all 1's.
   b. T should contain all 0's.
   c. MCW register as follows:
      (1) UNCT should be 1.
      (2) CONDT should be 0.
      (3) ERSLT should be 1.
      (4) PASS should be 1.
      (5) BFR1 should be 1.
9. Depress RESTART FLT I/O pushbutton. Machine should stop immediately.
10. View roller switches for same conditions as in step 8.
11. Depress LOAD pushbutton. Machine should stop immediately
12. View roller switches for the following:
    a. S should contain all D's.
    b. T should contain FFFF0000.
    c. BFR1 should be 1.
13. Depress RESTART FLT I/O pushbutton. Machine should complete remaining hardcore tests without stopping.

C.4.4 ROS BIT TESTS

These tests follow the hardcore tests and should be called in automatically. There are no stops until termination of these tests, about 40 seconds.

1. Set roller switches as in step 8 of paragraph C.4.3.
   a. S should contain all 1's.
   b. T should contain all 1's.
   c. MCW(0-3) show the ROS plane under test.

2. To repeat a failing test after an error stop, set TEST MODE REPEAT switch and depress START pushbutton. To halt repeat loop, restore TEST MODE REPEAT switch.

3. To continue to next test after an error stop, depress RESTART FLT I/O pushbutton.

C.4.5 FLT HARDCORE TESTS

The procedures for these tests are identical with those of ROS Hardcore tests (paragraph C.4.3), except for step 12, b, where T should contain FEFE0101 (not FFFF0000).

C.4.6 FLT SCAN-IN/SCAN-OUT TESTS

These tests follow the hardcore tests and should be called in automatically. To repeat a failing test, perform steps 2 and 3 of paragraph C.4.4.

C.4.7 FLT ONE-CYCLE TESTS

These tests follow the scan-in/scan-out tests and should be called in automatically. To repeat a failing test, perform steps 2 and 3 of paragraph C.4.4.

APPENDIX D

SPECIAL CIRCUITS

This appendix is reserved for special circuits.

APPENDIX E

ABBREVIATIONS

ABC	AB byte counter
ALD	automated logic diagram
ASC	address store compare
ASCII-8	American Standard Code for Information Interchange, extended to 8 bits
BCD	binary-coded decimal
BCU	bus control unit
BTU	British thermal unit
CAS	control automation system
CAW	channel address word
CB	circuit breaker
CC	condition code
CCW	channel command word
cfm	cubic feet per minute
charistic	characteristic
CLD	CAS logic diagram
CPU	central processing unit
CROS	capacitive read-only storage
CSW	channel status word
DVD	dividend
DVR	divisor
DX	first byte in a series of destination bytes
DX + 1	second byte in a series of destination bytes
DX + 2	third byte in a series of destination bytes
EBCDIC	Extended Binary-Coded-Decimal Interchange Code
EC	engineering change
end op	end operation
FEDM	Field Engineering Diagram Manual
FEMI	Field Engineering Manual of Instruction
FEMM	Field Engineering Maintenance Manual
FLT	fault locating test
GIS	General Initialization Sequence
HO	high order
HSS	high-speed storage
IC	instruction counter
I-Fetch	instruction-fetching
ILC	instruction length code
I/O	input/output
IPL	initial program load
kc	kilocycle
kva	kilovolt ampere
LAR	local storage address register
LCS	large-capacity storage
LO	low order
LS	local storage
LSWR	local storage working register
mc	megacycle
MCW	maintenance control word
MPD	multiplicand

MPR	multiplier
ms	millisecond
ns	nanosecond
op code	operation code
PAA	parallel adder A side
PAB	parallel adder B side
PAL	parallel adder latch
pf	picofarad
PP	partial product
PQ	partial quotient
PSW	program status word
RC	resistive-capacitive
ROS	read-only storage
ROSAR	read-only storage address register
ROSDR	read-only storage data register
SAA	serial adder A side
SAB	serial adder B side
SAL	serial adder latch
SAR	storage address register
SBA	serial adder bus A
SBB	serial adder bus B
SCR	silicon-controlled rectifier
SDBI	storage data bus in
SDBO	storage data bus out
STAT	status trigger
STC	ST byte counter
T(DX)	table byte specified by DX
T(DX + 1)	table byte specified by DX + 1
μf	microfarad
μs	microsecond
VFL	variable field length

CUT ALONG LINE

COMMENT SHEET

FROM _____

OFFICE NO. _____ DATE _____

TYPE OF USING SYSTEM _____

MANUAL TITLE _____ FORM NO. _____

FOLD FOLD

CHECK ONE OF THE COMMENTS AND EXPLAIN IN THE SPACE PROVIDED

☐ SUGGESTED ADDITION (PAGE , TIMING CHART, DRAWING, PROCEDURE, ETC.)

☐ SUGGESTED DELETION (PAGE)

☐ ERROR (PAGE)

EXPLANATION

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES, STAPLE, AND MAIL

226-2037-0

FOLD FOLD

FIRST CLASS
PERMIT NO. 116
KINGSTON, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY
IBM CORPORATION
CPO BOX 120
KINGSTON, N. Y. 12401

ATTN: PRODUCT PUBLICATION MANUAL WRITING
DEPARTMENT 526

OLD FOLD

Printed in U.S.A. 226-2037-0

IBM
International Business Machines Corporation
Field Engineering Division
112 East Post Road, White Plains, N.Y. 10601