IBM®

# 709 Data Processing System

# Reference Manual

## Contents

followed by

## 709 FORTRAN Automatic Coding System

# Computer Instructions

This section defines all computer instructions and describes their execution, indicators that may be affected, and timing.

A diagram representing the format of the instruction is given for each instruction. Preceding this diagram is the alphabetic code which identifies the instruction. The official name of the instruction is also given (Figure 22).

**CLA — Clear and Add**

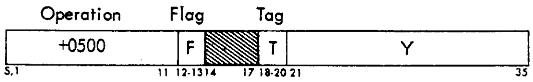


Figure 22. Sample Format of Instructions

The numerical operation code is given in the octal number system. This can be easily converted to the binary system for reference to the bit pattern interpreted by the computer. The numbers appearing beneath the diagram indicate the bit positions of the computer-word that are concerned with this particular instruction.

The symbol "Y" appearing in the diagram denotes the address part of the instruction. Y may stand for the address of a word in core storage, the length of a shift, or the address of an input-output unit. For some index transmission instructions, Y may also represent a number which is to be loaded either in true or complement form into an index register.

For some instructions, positions 21-35 are used to contain part of the operation code. The appearance of octal numbers instead of Y in the address field will distinguish this type of instruction from others. In all cases, the full operation code is shown by its octal representation.

Those instructions for which indirect addressing may be specified will have the symbol F (flag) appearing in positions 12 and 13 of the instruction diagram (Figure 22). This symbol represents 1 bits in both positions 12 and 13 of the instruction. The description of those operations which can have indirect addressing will be defined in terms of direct addressing.

Similarly, for instructions that are subject to effective address modification by an index register, the diagram has the symbol T in the tag field of the instruction. This T is also used to specify any index register to be changed, stored, or tested. The description accompanying an instruction defines the manner in which it is executed when its tag is zero.

The shaded area in the instruction diagrams represent fields that are not used in that instruction.

The symbols D, C, and R are used to denote the decrement, count, and right-half word fields of instructions which use these fields. Each of these fields is interpreted only by certain classes of instructions. If such a field is interpreted by an instruction, the bit positions used by the field will be shown in the instruction diagram. If an instruction has a D or R part, neither indirect addressing nor effective address modification is ever possible.

Descriptions of the instructions use the following special terms and definitions:

1. C (Y) denotes the contents of location Y, where Y refers to some location in storage. Similarly, C (AC), C (MQ), C (SR) and C (SI) denote the contents of the accumulator, multiplier-quotient, storage and sense indicator registers, respectively. In addition, subscripts refer to individual bit positions of a register. For example, C (MQ)$_{S, 1\text{-}17}$ is read "the contents of positions S, 1 through 17 of the MQ." When subscripts are not used with this notation, the entire register is implied. For example, C (AC) denotes the contents of positions S,Q,P, 1-35, inclusive.

2. With input-output operations, DC denotes data channel, LR denotes a DC location register, AR denotes a DC address register, and WR denotes a DC word count register.

3. When a register or part of a register is *cleared,* the cleared part is reset to zeros.

4. The *negative* of a number is the number with its sign reversed.

5. The magnitude of a number is the number with its sign made positive. (A zero in position S corresponds to a positive sign.)

6. When the word "store" is used in the title of an instruction, the transmission of a word or part of a word *from* some special register (e.g., the AC, MQ, SI or an index register) *to* some location in core storage is always implied.

7. When the word "load" is used in the title of an instruction, the transmission of a word or part of a word *from* some location in core storage *to* some special register (e.g., tne MQ, SI, or DC registers, but not the AC) is always implied.

8. When the word "place" is used in the title of an instruction, the AC is always one of the agents.

9. All logical operations interpret the sign position (S) of Y as a numerical binary bit corresponding to position P of the AC or position 0 of the SI. The S position of the AC is either ignored or cleared by logical operations.

10. In the three-letter alphabetic code:
    a. The letter Q designates the MQ register.
    b. The letter X in the second or third position designates an index register.
    c. The first letter of all transfer instructions is a T.

In the following instruction descriptions, an instruction format is shown for each instruction. Under the "Indicator" heading, only those indicators that may alter the course of a program through test instructions or by trapping are noted. Under the "Execution" section, when instructions are similar, only the differences are noted and a statement (e.g., "Same as ADD procedure") will mean that the operations are alike except for the differences noted. Instruction flow charts are used with many instructions to aid in presenting the data flow.

Note again that all addresses and numbers, unless otherwise specified, are given in the octal number system.

### Instruction Timing

All instructions are listed in the appendix in alphabetic and numerical sequence. Timing is noted in cycles with modification type, if any. The 709 cycle is 12 microseconds. If an instruction is subject to address modification through indexing and/or indirect addressing the facts will be noted by a T or F, respectively. With indirect addressing, the execution time is increased one cycle. The modification types are:

*Type 1 Instructions.* Multiply instructions are executed in two cycles if the number brought from storage contains zeros in positions 1 to 35. If the number brought from storage is not all zeros, execution time is a function of the number of sequential zero bits.

*Type 2 Instructions.* The execution time of these instructions is determined by the count field (C) specified in positions 10 through 17. The maximum number of cycles for a given value is $C/2 + 3$. Any remainder should be discarded.

*Type 3 Instructions.* FAD, FAM, FSB, and FSM will be executed in 6 cycles if the difference in characteristics is greater than 63 or if the extent of shift is less than 11 places during the adjustment of characteristics (step 5); also if the extent of shift is less than four places when normalizing (step 9b).

*Type 4 Instructions.* UFA, UAM, UFS and USM will be executed in five cycles if the difference in characteristics is greater than 63 or if the extent of shift is less than 11 places in step 5.

*Type 5 Instructions.* The execution of a FDH or FDP instruction requires only three cycles if the fraction of the dividend is zero.

*Type 6 Instructions.* The execution of a convert instruction is increased by one cycle for each storage reference specified by the count field in positions 10 through 17 of the instruction.

*Type 7 Instructions.* The instruction will be executed in two cycles if the extent of shift is 9 places or less. Each additional 12 shifts, or portion thereof, require another cycle.

*Type 8 Instructions.* The execution of these instructions may be delayed an indefinite length of time after interpretation, depending on the status of the I-O unit. For example, if multiple select instructions are given for the same data channel, the second select will be delayed if both selects are of the data-select type of operation.

All variable cycle instructions that have a precise minimum, average and maximum number of machine cycles are shown in Table I.

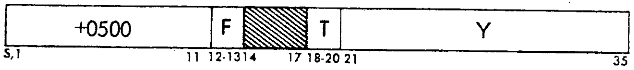
Table I. Variable Cycle Instructions

| | MACHINE CYCLES | | |
|---|---|---|---|
| INSTRUCTIONS | AVERAGE | MIN. | MAX. |
| MPY, MPR | 15.8 | 2 | 20 |
| DVH, DVP | 20 | 3 | 20 |
| FMP, UFM | 14.2 | 2 | 17 |
| FDH, FDP | 18 | 3 | 18 |
| FAD, FAM, FSB, FSM | 6.4 | 6 | 15 |
| UAM, USM | — | 5 | 11 |
| UFA, UFS | — | 5 | 11 |
| ALS, ARS, RQL | — | 2 | 5 |
| LLS, LRS, LGL, LGR | — | 2 | 8 |
| CAD, CAQ, CVR | — | 2 | 8 |
| VDH, VDP, VMP | — | 2 | 20 |

Instructions with a count whose value is larger than that implied by the size of the arithmetic registers may exceed the times shown. Average multiply times are derived assuming a random distribution of ones and zeros. In floating-point, a normalized operand is assumed. In determining the average floating-point add speed, a number of representative programs were traced. The time shown is based on an analysis of several million operands.

## Fixed Point Operation

### CLA — Clear and Add



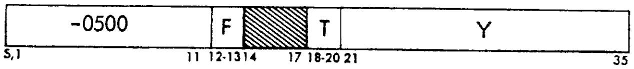
*Description.* The $\text{C(AC)}_{S,\ 1\text{-}35}$ are replaced with the C(Y). Positions P and Q of the AC are set to zero. The C(Y) remain unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* The C(Y) are brought to the SR. $\text{C(SR)}_{1\text{-}35}$ is taken to the adders, the adders to $\text{AC}_{(1\text{-}35)}$ and the SR(S) to AC(S).

### CAL — Clear and Add Logical Word



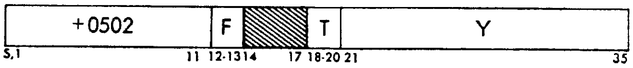
*Description.* The C(Y) replace the $\text{C(AC)}_{P,1\text{-}35}$. The sign of Y appears in position P of the AC. Positions S and Q of the AC are set to zero. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* The SR (S) goes to adder position P. The rest of the operation is the same as for CLA.

### CLS — Clear and Subtract



*Description.* The negative of C(Y) replaces the $\text{C(AC)}_{S,1\text{-}35}$. Positions P and Q of the AC are set to zero. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* (1) Invert sign of Y as it is entered into the SR. (2) Same as CLA.

The logic flow diagram for both the CLA and CLS instructions is shown in Figure 23.

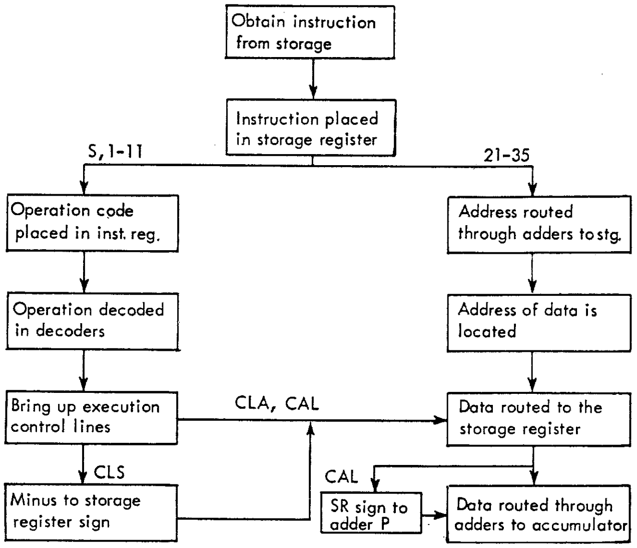
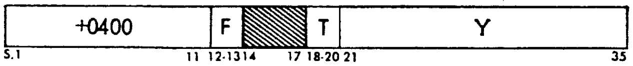

Obtain instruction from storage
Instruction placed in storage register
S,1-11
21-35
Operation code placed in inst. reg.
Address routed through adders to stg.
Operation decoded in decoders
Address of data is located
Bring up execution control lines
CLA, CAL
Data routed to the storage register
CLS
CAL
Minus to storage register sign
SR sign to adder P
Data routed through adders to accumulator

Figure 23. CLA and CLS Flow Chart

### ADD — Add



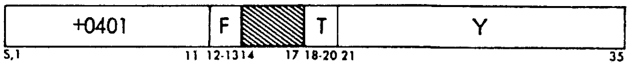
*Description.* The C(Y) are algebraically added to the C(AC). The resulting sum is placed in the AC. The C(Y) are unchanged. Numbers of the same magnitude but different signs give a resultant sign the same as the sign of the original AC.

*Indicators.* AC overflow.

*Timing:* 2 cycles

*Execution.* The C(Y) are taken to the SR and then to the adders. With signs alike, the true AC (Q-35) is also taken to the adders, and the sum returned to the AC. With signs unlike, the complement of the AC (Q-35) is taken to the adders; any Q carry is taken to adder 35 and is remembered. The resultant sum in the adders is then taken back to the AC. If the signs were unlike and there was no Q carry, the complement of the AC (Q-35) is again taken to the adders and then back to the AC. With a Q carry, reverse the AC sign (Figure 24).

### ADM — Add Magnitude



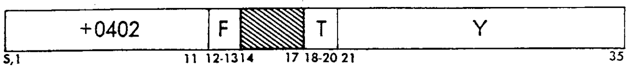
*Description.* The magnitude of the C(Y) is added to the C(AC). The resulting sum is placed in the AC. The C(Y) are unchanged. The sign of Y is ignored and Y is treated as a positive number. With a minus AC sign, a subtractive process will occur.

*Indicators.* AC overflow.

*Timing:* 2 cycles

*Execution.* (1) SR (S) is forced plus. (2) Procedure is the same as for ADD.

### SUB — Subtract



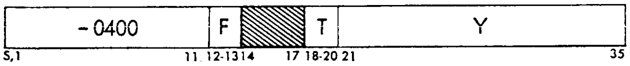
*Description.* The C(Y) are algebraically subtracted from the C(AC). The difference replaces the C(AC). The C(Y) are unchanged.

*Indicators.* AC overflow.

*Timing:* 2 cycles

*Execution.* (1) Sign of SR is reversed. (2) Same as ADD procedure.

### SBM — Subtract Magnitude



*Description.* The magnitude of the C(Y) is subtracted from the C(AC). The difference is placed in the C(AC). The sign of Y is ignored and the C(Y) are treated as a negative number. The C(Y) are unchanged. If the sign of the AC is minus, an ADD will occur.

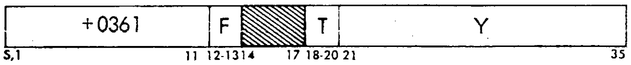
*Indicators.* AC overflow.

*Timing:* 2 cycles

*Execution.* (1) SR(S) is forced minus. (2) Same as add procedure.

The logic flow diagram for the ADD, SUB, ADM, and SBM instructions is shown in Figure 24.

### ACL — Add and Carry Logical Word



*Description.* The C(Y) are added to the C(AC)$_{P,1-35}$. The resultant sum replaces the C(AC)$_{P,1-35}$. The sign

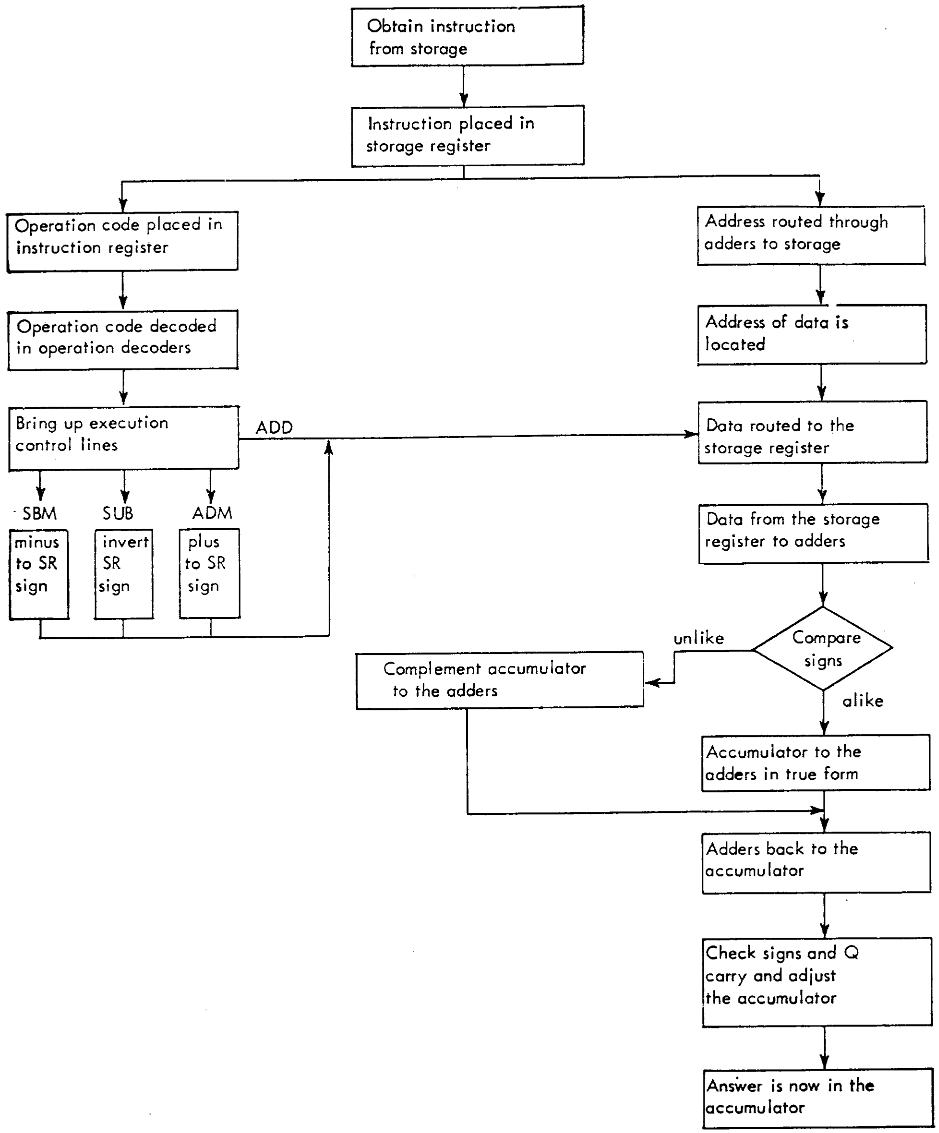

Obtain instruction from storage
Instruction placed in storage register
Operation code placed in instruction register
Operation code decoded in operation decoders
Bring up execution control lines
ADD
SBM
SUB
ADM
minus to SR sign
invert SR sign
plus to SR sign
Address routed through adders to storage
Address of data is located
Data routed to the storage register
Data from the storage register to adders
Compare signs
unlike
alike
Complement accumulator to the adders
Accumulator to the adders in true form
Adders back to the accumulator
Check signs and Q carry and adjust the accumulator
Answer is now in the accumulator

Figure 24. ADD, ADM, SUB, and SBM Flow Chart

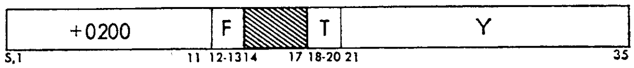
of Y is added to position P of the AC. A carry from AC (P) is added to AC (35). Positions S and Q of the AC are not affected.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* The C (Y) are taken to the SR. The SR (S, 1-35) are then taken to the adders (P, 1-35). An adder P carry goes to adder 35. Adders (P, 1-35) are then returned to AC (P, 1-35).

## MPY — Multiply



*Description.* The C(Y) are multiplied by the C(MQ). The 35 most significant bits of the 70-bit product replace $C(AC)_{1\text{-}35}$ and the 35 least significant bits replace the $C(MQ)_{1\text{-}35}$. AC (P and Q) are cleared. The signs of the AC and MQ are set to the algebraic sign of the product. The number of bits to the right of the binary point of the first factor added to the number of bits to the right of the binary point of the second factor give the total number of bits to the right of the binary point in the product.

*Indicators.* None

*Timing:* 2-20 cycles, modification 1.

*Execution.* (1) The C (Y) are tested, and if the magnitude of the C (Y) is zero, the C (AC) and C (MQ) are cleared. Step 2 is skipped and step 3 occurs. (2) If the magnitude of the C (Y) is not zero, the C (AC) $_{Q,P,1\text{-}35}$ are cleared and multiplication proceeds:

a. If $MQ_{35}$ contains a 1, the $C(Y)_{1\text{-}35}$ are added to the AC. The $C(AC)_{Q,P,1\text{-}35}$ and the $C(MQ)_{1\text{-}35}$ are then shifted right one position.

b. If $MQ_{35}$ contains a 0, the C (AC) $_{Q,P,1\text{-}35}$ and C (MQ) $_{1\text{-}35}$ are shifted right one position. Step 2 occurs 3 times per cycle on the 709. With sequential zeros, up to 12 shifts may occur per cycle.

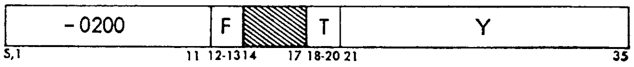
(3) If the signs of the MQ and location Y are the same, the signs of the AC and MQ are made positive. If the signs differ, the signs of the AC and MQ are made negative.

As an example, assume that the AC, MQ, and location Y are four bits in length instead of 35. The following sequence of steps would occur during a multiply. The number 13 is in the MQ and the C (Y) are 6. The actual bit-configuration appears in each register (after the step is complete).

The flow chart is shown in Figure 25.

| AC | MQ | Y | COMMENTS |
|---|---|---|---|
| 0000 | 1101 | 0110 | Initial contents of the registers. MQ 35 ready to be tested. |
| 0110 | 1101 | | C (Y) added to AC since MQ 35 is a 1. |
| 0011 | 0110 | | C (AC, MQ) shifted right one place. Test MQ 35. |
| 0001 | 1011 | | No addition, since MQ 35 contained a 0. C (AC, MQ) again shifted right and MQ 35 is tested. |
| 0111 | 1011 | | C (Y) added since MQ 35 is a 1. |
| 0011 | 1101 | | C (AC, MQ) shifted right and MQ 35 tested. |
| 1001 | 1101 | | C (Y) added, since MQ35 is a 1. |
| 0100 | 1110 | | C (AC, MQ) shifted right. At this point the shift counter has been reduced to zero and the process stops with the eight-bit product in the AC and MQ registers. |

## MPR — Multiply and Round



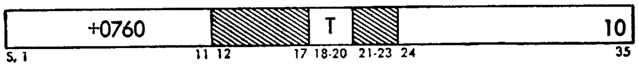
*Description.* This operation is the same as multiply except that the C (AC) are increased by 1 if MQ (1) contains a one after multiplication is complete.

*Indicators.* None.

*Timing:* 2-20 cycles, modification 1.

*Execution.* (1) Develop the product as in multiply. (2) If MQ (1) contains a 1, add a 1 to AC (35).

## RND — Round



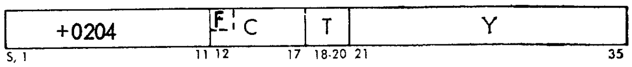
*Description.* If position 1 of the MQ contains a 1, the C (AC) are increased by one. If MQ (1) contains a 0, the C (AC) are unchanged. In either case the C (MQ) are unchanged. Note that positions 24-35 of this instruction represent part of the operation code. Modification by indexing may change the operation code itself.

*Indicators.* AC overflow.

*Timing:* 2 cycles

*Execution.* If MQ (1) contains a 1, the C (AC) $_{Q\text{-}35}$ is sent to the adders with a carry to adder 35. The adder (Q-35) is then taken to AC (Q-35). If MQ (1) contains a 0, no rounding occurs.

## VLM — Variable Length Multiply



*Description.* This instruction multiplies the C(Y) by the C low-order bits of the C (MQ), to produce a 35 + C

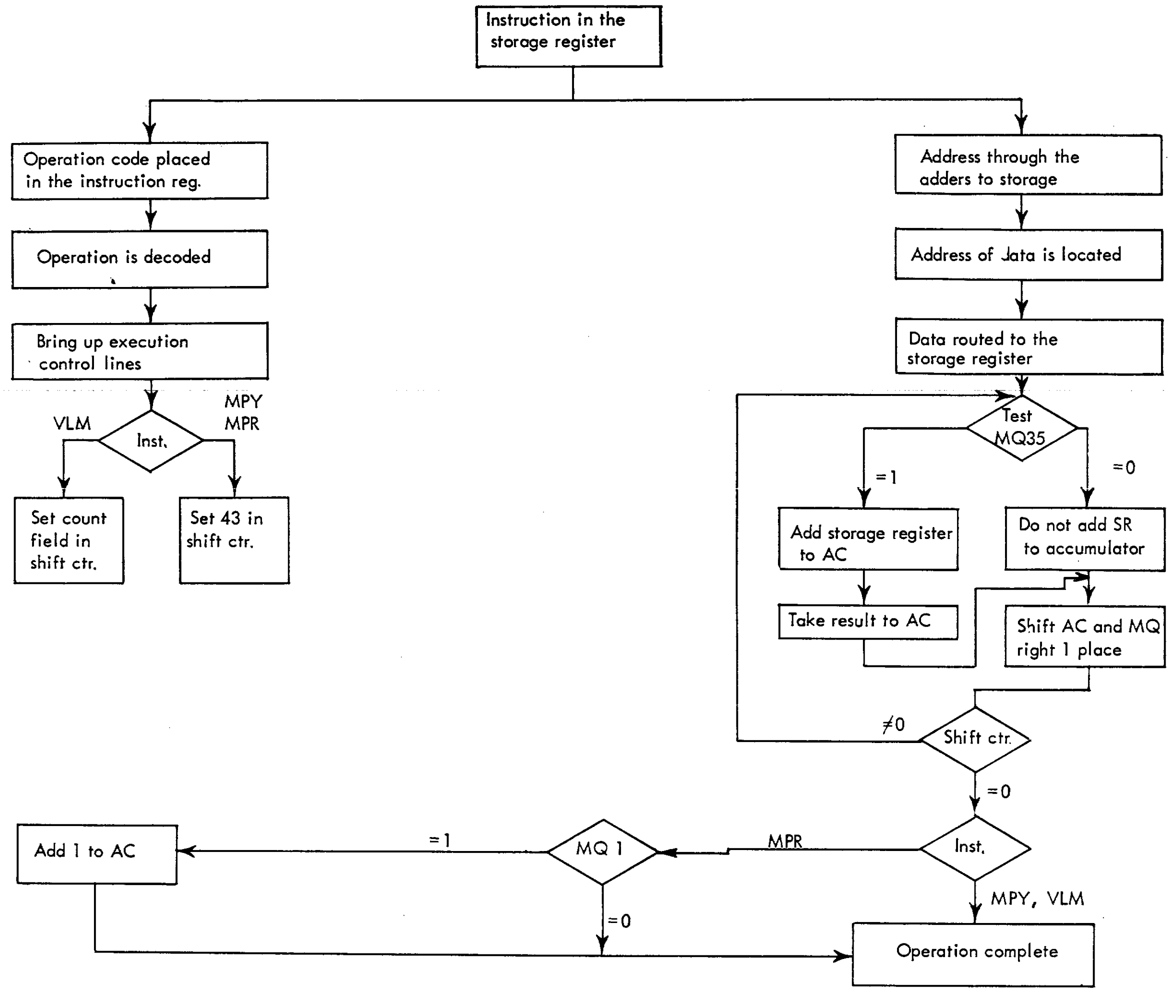

Instruction in the storage register
Operation code placed in the instruction reg.
Operation is decoded
Bring up execution control lines
VLM
Inst.
MPY
MPR
Set count field in shift ctr.
Set 43 in shift ctr.
Address through the adders to storage
Address of data is located
Data routed to the storage register
Test MQ35
=1
=0
Add storage register to AC
Do not add SR to accumulator
Take result to AC
Shift AC and MQ right 1 place
≠0
Shift ctr.
=0
Add 1 to AC
=1
MQ 1
MPR
Inst.
MPY, VLM
=0
Operation complete

Figure 25. MPY, MPR, VLM Instruction Flow Chart

bit product. The 35 most significant bits of the product replace the C(AC)$_{1-35}$ and the C least significant bits replace the C(MQ) 1 through C. Positions Q and P of the AC are cleared. The remaining 35—C positions of the MQ will contain the original 35—C high-order positions of the MQ. The sign of the AC and MQ is the algebraic sign of the product. An example is shown in Figure 26.

If C is zero, the instruction is interpreted as a no-operation and the computer proceeds directly to the next instruction in sequence, leaving the AC unchanged.

If C is not zero but the C(Y) are zero, the C(AC) and C(MQ) are cleared. If the signs of the MQ and location Y are the same, the signs of the AC and MQ are made positive. If the original signs of the SR and MQ differ, the signs of the AC and MQ are made negative. NOTE: A count field which places a 1 bit in both positions 12 and 13 (60 or larger) will cause indirect addressing. In general, counts larger than 35 are meaningless.

*Indicators.* None.

*Timing:* 2-20 cycles, modifications 1 and 2

*Execution.* The instruction is the same as multiply except that the contents of the count field, instead of 43, are placed in the shift counter.

Figure 25 shows the flow chart for MPY, MPR, and VLM instructions.

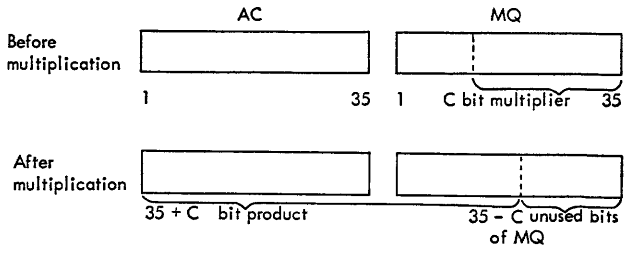


Figure 26. Variable Length Multiply

### DVH — Divide or Halt

| | | | | |
|---|---|---|---|---|
| +0220 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C(AC)$_{Q,P,1\text{-}35}$ and the C(MQ)$_{1\text{-}35}$ are treated as a 70-bit dividend plus sign, and the C(Y) as a 35-bit divisor. If the magnitude of C(Y) is greater than the magnitude of C(AC), division takes place. A 35-bit quotient replaces the C(MQ)$_{1\text{-}35}$ and the remainder replaces the C(AC)$_{1\text{-}35}$. The MQ sign is the algebraic sign of the quotient and the AC sign is the sign of the dividend.

If the magnitude of the C (Y) is less than or equal to the magnitude of the C (AC), division does not occur and the computer stops with the divide-check indicator on. For example, if Q or P of the AC contains a 1, the magnitude of the C (Y) is less than the C (AC). If division does not occur, the dividend remains unchanged in the AC and MQ.

*Indicators.* Divide check

*Timing:* 3-20 cycles.

*Execution.* (1) The C (AC and MQ)$_{1\text{-}35}$ are shifted left one position, creating a zero in position 35 of the MQ. (2) If the magnitude of the C (Y) is less than or equal to the magnitude of C (AC), the magnitude of C (Y) is subtracted from the magnitude of C (AC) and a one replaces the zero in MQ$_{35}$. Step 1 is then repeated (Figure 27). (3) If the magnitude of the C (Y) is greater than the magnitude of the C(AC), the computer returns to step 1.

The above process occurs 35 times for each division, two times per machine cycle.

The following example is a division problem. Again assume a four-bit machine. The problem is 66 divided by 5, and the binary numbers represent the result of the described step.

| AC | MQ | Y | COMMENTS |
|---|---|---|---|
| 0100 | 0010 | 0101 | Initial contents. C (AC) less than C (Y); division will take place. |
| 1000 | 0100 | | C (AC and MQ) shifted left one place; C (AC) greater than C (Y). |
| 0011 | 0101 | | C (Y) subtracted from C (AC) and a 1 replaces MQ 35. |
| 0110 | 1010 | | C (AC and MQ) shifted left one place; C (AC) greater than C (Y). |
| 0001 | 1011 | | C (Y) subtracted from C (AC) and a 1 replaces MQ 35. |
| 0011 | 0110 | | C (AC and MQ) shifted left one place; C (AC) less than C (Y). |
| 0110 | 1100 | | C (AC and MQ) shifted left one place; C (AC) greater than C (Y). |
| 0001 | 1101 | | C (Y) subtracted from C (AC) and a 1 replaces MQ 35. |
| | | | The quotient is now complete in the MQ with the remainder in the AC. |

### DVP — Divide or Proceed

| | | | | |
|---|---|---|---|---|
| +0221 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* If the magnitude of the C (Y) is greater than the magnitude of the C (AC), division occurs as with the DVH instruction. If the magnitude of the C (Y) is less than or equal to the magnitude of the C (AC), the divide-check indicator is turned on and the computer proceeds to the next instruction.

*Indicators.* Divide check

*Timing:* 3-20 cycles.

*Execution.* Exactly the same as DVH except that instead of halting, when a divide-check occurs, the computer executes the next sequential instruction (Figure 27).

### VDH — Variable Length Divide or Halt

| | | | |
|---|---|---|---|
| +0224 | F C | T | Y |
| S, 1 11 | 12 17 | 18-20 | 21 35 |

*Description.* This instruction is the same as a DVH except that a C-bit quotient plus sign replaces the C low-order positions of the MQ. The remainder replaces the C (AC)$_{1\text{-}35}$ and the 35—C high-order positions of the MQ. Instead of 43 being placed in the shift counter initially, C is placed there. If C is zero the instruction is interpreted as a no-operation and the computer proceeds directly to the next instruction in sequence.

*Indicators.* Divide check.

*Timing:* 2-20 cycles, modification 2

*Execution.* The same operation as DVH except as noted above.

NOTE: Indirect addressing may occur if the count field places 1-bits in positions 12 and 13 of the instruction (Figure 27).

### VDP — Variable Length Divide or Proceed

| | | | |
|---|---|---|---|
| +0225 | F C | T | Y |
| S, 1 11 | 12 17 | 18-20 | 21 35 |

*Description.* This instruction is the same as DVP except that a C bit quotient with a sign replaces the C low-order positions of the MQ. The remainder replaces the C (AC)$_{1\text{-}35}$ and the 35—C high-order positions of the MQ. C rather than 43 is placed in the shift counter initially. If C is zero the instruction is interpreted as a no-operation and the computer proceeds directly to the next instruction in sequence.

*Indicators.* Divide check.

*Timing:* 2-20 cycles, modification 2

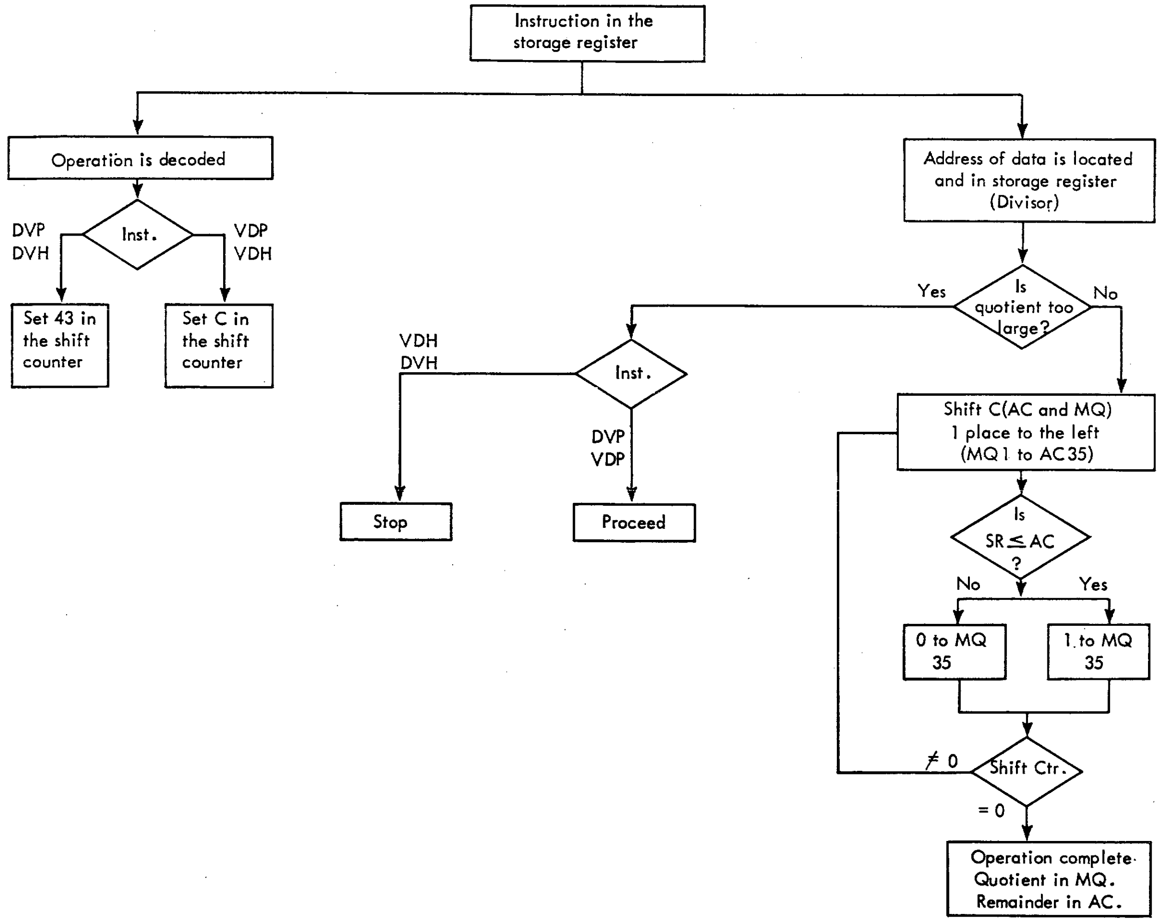

Instruction in the storage register
Operation is decoded
Inst.
DVP
DVH
VDP
VDH
Set 43 in the shift counter
Set C in the shift counter
Address of data is located and in storage register (Divisor)
Yes
Is quotient too large?
No
VDH
DVH
Inst.
DVP
VDP
Stop
Proceed
Shift C(AC and MQ) 1 place to the left (MQ 1 to AC35)
Is SR≤AC ?
No
Yes
0 to MQ 35
1 to MQ 35
≠ 0
Shift Ctr.
= 0
Operation complete. Quotient in MQ. Remainder in AC.

Figure 27. DVH, DVP, VDH, and VDP Flow Chart

*Execution.* The same procedure as DVP except as noted above.

NOTE: Indirect addressing may occur if the count field places 1 bits in positions 12 and 13 of the instruction.

Figure 27 shows the logic flow chart for DVH, DVP, VDH, and VDP instructions.

## Floating Point Operations

The following operations are divided into two groups to describe the processing of floating-point numbers in either normalized or unnormalized form. The possibility of floating-point overflow or underflow during the execution of a floating-point instruction is indicated by an asterisk (*). All conditions of underflow and overflow are discussed following the last floating-point instruction.

### Floating Point Arithmetic

The algebraic addition of two floating-point numbers in the computer is analogous to the ordinary algebraic addition of two signed numbers with decimal points. An example is the algebraic addition of the two numbers 100 and −0.1009:

$$\begin{array}{r} 100.0000 \\ -\ \ 0.1009 \\ \hline 99.8991 \end{array}$$

Note that the second number must be shifted to the right to line up the decimal points, and that the first number must be supplied with additional zeros. The same addition performed with numbers expressed in floating-point decimal form, would be:

$$\begin{array}{r} .1000 \times 10^3 \\ -.1009 \times 10^0 \end{array}$$

Again, before the addition, the lower number is shifted to the right with a compensating change in the exponent and corresponding zeros are added to the number on the upper line:

$$\begin{array}{r} .1000000 \times 10^3 \\ -.0001009 \times 10^3 \\ \hline .0998991 \times 10^3 = .998991 \times 10^2 \end{array}$$

Note also that the digits of the answer must be moved to the left to be in normalized form and that the final fraction contains more digits than either of the two numbers involved in the addition.

In the computer the two numbers are expressed as binary fractions, each having an 8-bit binary characteristic to represent the exponent of 2. The "lining up" is done by shifting from the AC into the MQ. The result of an addition or multiplication is normalized by shifting the fractions in the AC and MQ left while making compensating changes in the characteristic of the sum or product.

## FAD — Floating Add

| +0300 | F | ▨ | T | Y |
|---|---|---|---|---|
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The floating-point numbers located in Y and the AC are added together. The most significant portion of the result appears as a normal floating-point number in the AC. The least significant portion of the result appears in the MQ as a floating-point number with a characteristic 33 (octal) less than the AC characteristic. The signs of the AC and MQ are set to the sign of the larger factor. The sum in the AC and MQ is always normalized whether the original factors were normal or not. If C (AC)$_{1\text{-}35}$ contain zeros, the FAD may be used to normalize an unnormal floating-point number.

*Indicators.* Floating-point underflow, overflow, and floating-point trap.

*Timing:* 6-15 cycles, modification 3

*Execution*

1. The MQ register is cleared to zeros.

2. The C (Y) are placed in the SR.

3. If the characteristic in the SR is less than the characteristic in the AC, the C (SR) and C (AC)$_{S,1\text{-}35}$ are interchanged, as the number with the smaller characteristic must appear in the AC before addition can take place.

4. The MQ is given the same sign as the AC.

5. If the difference in the characteristics is greater than 63, the C (AC) are cleared. If the difference in the characteristics is a number $N$ less than or equal to 63, the C (AC)$_{9\text{-}35}$ are shifted right $N$ places. Bits shifted out of position 35 of the AC enter position 9 of the MQ. Bits shifted out of position 35 of the MQ are lost.

6. The characteristic in the SR replaces the C (AC)$_{1\text{-}8}$.

7. The C (SR)$_{9\text{-}35}$ are added to the C (AC)$_{9\text{-}35}$ and this sum replaces the C (AC)$_{9\text{-}35}$. If the signs of the AC and SR are unlike, the C (SR)$_{9\text{-}35}$ are added to the 1's complement of the C (AC)$_{9\text{-}35}$. Since the C (AC)$_{9\text{-}35}$ represent a pure fraction, the magnitude of their 1's complement is equal to $(1 - 2^{-27})$ $-$ C (AC)$_{9\text{-}35}$.

8. Regardless of the sign or relative magnitudes of the SR and AC, the result appears in double-precision form with signs alike in both the AC and MQ. If the signs of the AC and SR are the same and the magnitude of the sums of the fractions is greater than or equal to one, there is a carry from position 9 into position 8 of the AC. Thus, the characteristic of the AC is increased by one.* In this event, the fractions of the AC and MQ are shifted right one position and a 1 is inserted into position 9 of the AC. If the signs of the AC and SR are different, there are two cases, both depending on the difference between the SR and AC fractions.

CASE 1. If the magnitude of the SR fraction is greater than the fraction in the AC, the AC and MQ signs are both changed to the sign of the SR. If the fraction of the MQ is zero, the difference between the fractions of the SR and AC is placed in the AC. If the fraction of the MQ is not zero, the difference between the fractions of the SR and AC, minus one, is placed in the AC; the 2's complement of the MQ fraction replaces the fraction in the MQ.

CASE 2. If the magnitude of the SR fraction is less than the fraction in the AC, the difference of the two fractions replaces the fraction of the AC. The sign of the AC and the entire MQ remain unchanged.

9a. If the resulting fractions in both the AC and MQ are zero, the AC is cleared, yielding a normal zero. If the fractions are in normalized form before the FAD is given, this result can only occur if the signs are different and the C (Y)$_{1\text{-}35}$ are equal to the C (AC)$_{1\text{-}35}$. The signs of the AC and MQ will be equal to the sign of the number originally in the AC. If the resulting fraction in the AC is zero and the two numbers were not in normalized form before addition, the signs of the AC and MQ are equal to the sign of the original number having the smaller characteristic.

9b. If the resulting fractions in the AC and MQ are not zero, the fractions of the AC and MQ are shifted left until a 1 appears in position 9 of the AC. Bits enter position 35 of the AC from position 9 of the MQ. The characteristic in the AC is reduced by one for each position shifted.* No shifting is necessary if the fraction of the AC is in normal form at the beginning of this step.

10. The MQ is given a characteristic which is 27 less than the characteristic in the AC,* unless the AC contains a normal zero, in which case zeros are left in positions 1-8 of the MQ.

If the P and/or Q positions of the AC are not zero before the execution of the FAD, the result will usually be incorrect. Non-zero bits in P and/or Q which are initially interpreted as part of the AC characteristic make it larger than the characteristic in the SR so that the interchange in step 3 will always take place. During the interchange a 1 will be placed in position S of the SR if there is a 1 in either S or P positions of the AC, so that the sign of the number may be changed. Any bit in Q is lost during the interchange and both P and Q are cleared when the C (SR) replace the C (AC). The difference between the two characteristics is computed after the interchange occurs, so that in step 5, N will not be equal to the difference between the original characteristics. In step 6 the characteristic in the SR, with its Q and P bits missing, replaces the characteristic in the AC. Consider as a sample problem the addition of:

$$2^2 \times .1001 = \quad \text{(SR)} +10000010.1001$$
$$2^5 \times .1001 = \quad \text{(AC)} +10000101.1001$$

First, the exponents must be equalized and then the addition may proceed. The characteristics are checked and found unequal, with the largest in the AC. The numbers in the AC and SR are then exchanged, giving:

| | |
|---|---|
| SR | +10000101.1001 |
| AC | +10000010.1001 |

The MQ content is zeros at this time. The $\text{C (AC)}_{9\text{-}35}$ are then shifted right the number of places needed to equalize the exponents. (Remember that the binary point is located between positions 8 and 9 of all registers.) The registers then appear as:

| | |
|---|---|
| SR | +10000101.1001 |
| AC | +10000101.0001 |
| MQ | +00000000.0010 |

The fractions (positions 9-35) may now be added.

| | |
|---|---|
| SR | +10000101.1001 |
| AC | +10000101.1010 |
| MQ | +00000000.0010 |

AC position 9 is checked for a 1 and no normalizing occurs. The MQ characteristic is now set. It is equal to the AC characteristic minus the number of places in the AC fraction (27 in the computer, 4 in this example):

| | |
|---|---|
| SR | +10000101.1001 |
| AC | +10000101.1010 |
| MQ | +10000001.0010 |

Decoding the results into the original format, we find:

$$\begin{array}{l l}
2^5 \times .1001 & \text{MQ} = 2^1 \times .0010 = 2^5 \times .00000010 \\
\underline{2^5 \times .0001001} & \underline{\text{AC} = \qquad\qquad 2^5 \times .1010} \\
2^5 \times .1010001 & \text{resultant sum } = 2^5 \times .10100010
\end{array}$$

### FAM — Floating Add Magnitude

| +0304 | F | ▨ | T | Y |
|---|---|---|---|---|
| S,1   11 | 12-13 | 14   17 | 18-20 | 21   35 |

*Description.* This instruction algebraically adds the positive magnitude of the floating-point numbers contained in Y to the signed floating-point number in the AC. The sum is normalized.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 6-15 cycles, modification 3

*Execution.* The same procedure as FAD except that the magnitude of the number in the SR is used (SR sign is forced plus).

### UFA — Unnormalized Floating Add

| -0300 | F | ▨ | T | Y |
|---|---|---|---|---|
| S,1   11 | 12-13 | 14   17 | 18-20 | 21   35 |

*Description.* This instruction algebraically adds two floating-point numbers contained in the AC and Y. The sum is not normalized.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 5-11 cycles, modification 4

*Execution.* The same procedure as FAD except that no normalizing will occur (step 9).

### FSB — Floating Subtract

| +0302 | F | ▨ | T | Y |
|---|---|---|---|---|
| S,1   11 | 12-13 | 14   17 | 18-20 | 21   35 |

*Description.* This instruction algebraically subtracts the floating-point number located in Y from the floating-point number in the AC, and normalizes the result.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 6-15 cycles, modification 3

*Execution.* The same procedure as FAD except that the negative of the C(Y) are placed in the SR (SR sign is reversed).

### UAM — Unnormalized Add Magnitude

| | | | | |
|---|---|---|---|---|
| -0304 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction algebraically adds the magnitude of the floating-point number contained in Y to the signed floating-point number in the AC. The sum is not normalized.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 5-11 cycles, modification 4

*Execution.* The same procedure as FAD except that the sign of the number in the SR is made positive and the result is not normalized.

### FSM — Floating Subtract Magnitude

| | | | | |
|---|---|---|---|---|
| +0306 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction algebraically subtracts the magnitude of a floating-point number stored at Y from the signed floating-point number in the AC. The result is normalized.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 6-15 cycles, modification 3

*Execution.* The same procedure as FAD except that the negative magnitude of the contents of Y are used (SR sign is forced minus).

### UFS — Unnormalized Floating Subtract

| | | | | |
|---|---|---|---|---|
| -0302 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction algebraically subtracts the floating-point number located in Y from the floating-point number in the AC. The result is not normalized.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 5-11 cycles, modification 4

*Execution.* The same procedure as FAD except that the negative of the contents of Y are placed in the SR and normalizing does not occur.

### USM — Unnormalized Subtract Magnitude

| | | | | |
|---|---|---|---|---|
| -0306 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction algebraically subtracts the magnitude of a floating-point number stored at Y from the signed floating-point number in the AC. The result is not normalized.

*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 5-11 cycles, modification 4

*Execution.* The same procedure as FAD except that the negative magnitude of the contents of Y are used and the result is not normalized.

The differences between answers, received after execution of a floating-point add or subtract operation, when using a 704 or 709 system is a matter of increased precision and is summarized as:

| | 704 | 709 |
|---|---|---|
| 1. | Accumulator sign and MQ sign not necessarily the same. | Accumulator sign and MQ sign are guaranteed to be equal. |
| 2. | Characteristic difference between accumulator and MQ is usually $27_{10}$, but it can be $28_{10}$ when adding numbers of unlike signs. | Characteristic difference between accumulator and MQ is always $27_{10}$. |
| 3. | If the accumulator is zero and the MQ is not, the sum will not be shifted and the accumulator will be made equal to a normal zero. | If the accumulator is zero and the MQ is not, the MQ factor will be shifted in order to normalize the sum. |

### FRN — Floating Round

| | | | | |
|---|---|---|---|---|
| +0760 | ▨ | T | ▨ | 11 |
| S, 1 11 | 12 17 | 18-20 | 21-23 | 24 35 |

*Description.* Floating-point add, subtract, and multiply produce a double-word result. The instruction

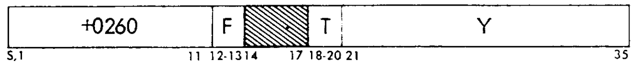
FRN will add 1 to position 35 of the AC if the MQ fraction is equal to or exceeds half the magnitude of a 1-bit in AC 35. The AC is corrected if rounding results in a carry from AC 9.

*Indicators.* Floating-point overflow, floating-point trap.

*Timing:* 2 cycles

*Execution.* If MQ 9 contains a 1, a carry will be added to AC 35. A carry out of AC 9 increases the characteristic of the AC by 1, causes the fraction of the AC to be shifted right and a 1 to be placed in AC 9. Since the address part of this instruction represents part of the operation code, any modification by an index register may result in changing the operation itself.

## FMP — Floating Multiply



*Description.* The C(Y) are multiplied by the C(MQ). The most significant part of the product appears in the AC and the least significant`part appears in the MQ. The product of two normalized numbers is in normalized form. If either of the numbers is not normalized, the product may or may not be in normalized form.

*Indicators.* Floating-point underflow, floating-point overflow, and floating-point trap.

*Timing:* 2-17 cycles, modification 1

*Execution*

1. The C(Y) are placed in the SR and the AC is cleared.

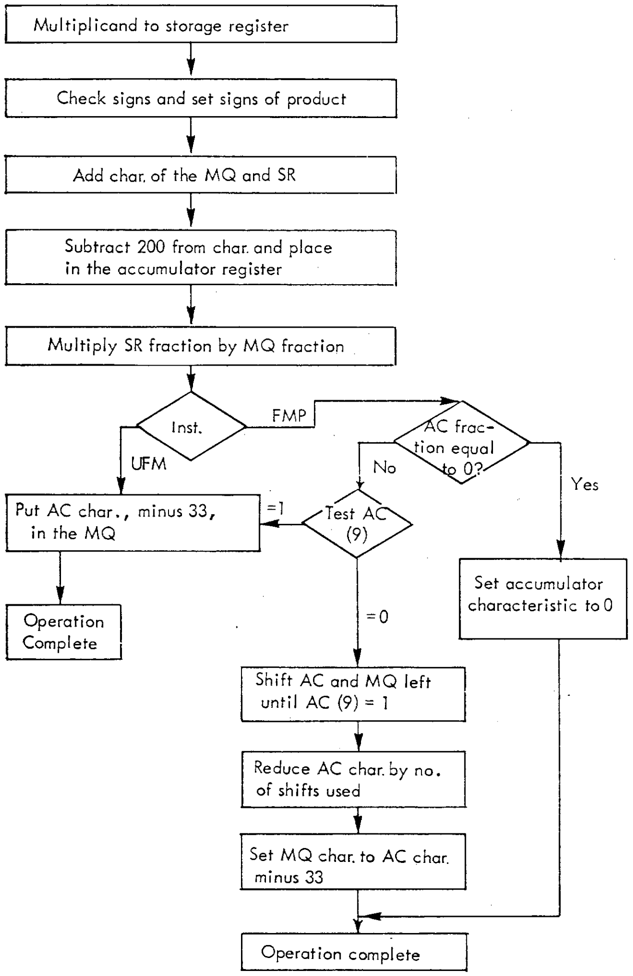
2a. If the multiplicand is a normal zero so that C(SR)$_{1-35}$ are equal to zero, the MQ$_{1-35}$ is cleared and the calculator proceeds directly to the next instruction in sequence.

2b. If the C(SR)$_{9-35}$ are not equal to zero, the sum of the characteristics in the SR and MQ minus 128 is placed in positions 1-8 of the AC* (Figure 28).

3. The C(SR)$_{9-35}$ are multiplied by the C(MQ)$_{9-35}$. The 27 most significant digits of the 54-digit product replace the C(AC)$_{9-35}$ and the 27 least significant digits replace the C(MQ)$_{9-35}$. The sign of the AC is the algebraic sign of the product.

4a. If the fraction in the AC is zero, the C(AC)$_{Q,\ P,\ 1-35}$ are cleared, yielding a signed normal zero.

4b. If the position 9 of the AC contains a zero but the fraction in the AC is not zero, the C(AC)$_{10-35}$ and the C(MQ)$_{9-35}$ are shifted left one position and the characteristic in the AC is reduced by 1.



Multiplicand to storage register
Check signs and set signs of product
Add char. of the MQ and SR
Subtract 200 from char. and place in the accumulator register
Multiply SR fraction by MQ fraction
Inst.
FMP
UFM
AC fraction equal to 0?
No
Yes
Put AC char., minus 33, in the MQ
=1
Test AC (9)
=0
Set accumulator characteristic to 0
Operation Complete
Shift AC and MQ left until AC (9) = 1
Reduce AC char. by no. of shifts used
Set MQ char. to AC char. minus 33
Operation complete

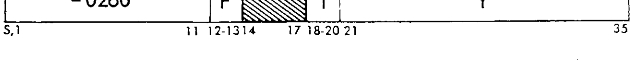
Figure 28. FMP and UFM Flow Chart

5a. If the AC contains a normal zero, positions 1-8 of the MQ are cleared.

5b. If the AC does not contain a normal zero, the C(MQ)$_{1-8}$ are replaced by a characteristic which is 27 less than the characteristic in the AC (*).

6. The sign of the MQ is replaced by the sign of the AC.

## UFM — Unnormalized Floating Multiply



*Description.* This instruction multiplies the floating-point number at Y by the floating-point number in the MQ. The result is not normalized.

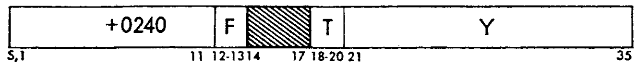
*Indicators.* Floating-point underflow and floating-point overflow; floating-point trap.

*Timing:* 2-17 cycles, modification 1

*Execution.* The same procedure as FMP except that the product is not normalized or zero tested.

Figure 28 shows a flow chart of the FMP and UFM instructions.

### FDH — Floating Divide or Halt



*Description.* The C(AC) are divided by the C(Y). The quotient appears in the MQ and the remainder appears in the AC. If the magnitude of the AC fraction is greater than or equal to twice that of the C(Y)$_{9-35}$, or if the magnitude of the C(Y)$_{9-35}$ is zero, the divide check indicator is turned on and the computer stops, leaving the dividend in the AC unchanged and a normal zero in the C(MQ). The quotient is in normal form if both the dividend and divisor are in that form. If they are, the magnitude of the ratio of the fraction in the AC to the fractional part of C(Y) is less than two but greater than one-half.

*Indicators.* Floating-point underflow, floating-point overflow, divide check, and floating-point trap.

*Timing:* 3-18 cycles, modification 5

*Execution*

1. The C(Y) are placed in the storage register.

2. The MQ is cleared.

3. The sign of the MQ is made equal to the algebraic sign of the quotient. The sign of the AC remains unchanged throughout so that the signs of the remainder and dividend always agree.

4. If the magnitude of the fraction in the AC is greater than or equal to twice the magnitude of the fraction in the SR, or if the fraction in the SR is zero, the divide-check indicator and panel light are turned on, the calculator stops and the dividend is left unchanged in the AC.

5. If the fraction in the AC is zero, the C(AC)$_{Q,P,1-35}$ are cleared and the remaining steps are skipped. If S(AC) is minus, the sign is forced plus.

6. If the magnitude of the fraction in the AC is greater than or equal to the magnitude of the fraction in the SR, the AC is shifted right one position, and the characteristic in the AC is increased by one.* The bit in position 35 of the AC enters position 9 of the MQ.

7. The characteristic of the AC minus the characteristic of the SR plus 128 is placed in positions 1-8 of the MQ.*

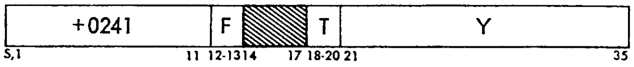
8. The fractional part of the dividend, which consists of the C(AC)$_{9-35}$ (and the C(MQ) if the condition of step 6 is met), is divided by the fraction in the SR and the quotient replaces the C(MQ)$_{9-35}$.

9. The 27-bit remainder resulting from the division in step 8 replaces the C(AC)$_{9-35}$.

10. The characteristic in the AC is reduced by 27*.

NOTE: Even though the numbers are not in normalized form, the quotient will be normalized if the ratio above holds. If the fraction in the AC is zero, a normalized zero will result in the MQ.

### FDP — Floating Divide or Proceed



*Description.* This instruction divides the floating-point number stored in the AC by the floating-point number located at Y.

*Indicators.* Floating-point underflow, floating-point overflow, divide check, and floating-point trap.

*Timing:* 3-18 cycles, modification 5

*Execution.* The same procedure as FDH except that if the computer cannot handle the problem, it does not halt but proceeds to the next sequential instruction.

### Floating-Point Trap

During the execution of floating-point instructions the resultant characteristic in the AC and MQ may exceed eight bit positions (result is too large for storage). The capacity is exceeded if the exponent goes beyond +177 or below −200. Beyond +177 is termed *overflow* while below −200 is termed *underflow*. Overflow and underflow may occur in either the AC or the MQ registers.

To aid the programmer in checking for these conditions, a unique check called *floating-point trap* is used. The computer will, upon sensing an underflow or overflow, put the address plus one of the instruction that caused the condition into the address portion of location 0000.

An identifying code, telling whether an underflow or an overflow occurred and whether the most significant result is in the AC or MQ, is placed in the decrement portion of location 0000. The computer then executes the instruction at location 0010 and proceeds from there. These underflows and overflows are termed *spills*. The spill code is produced as follows:

| | | | DEC. PORTION | | | | OCTAL |
|---|---|---|---|---|---|---|---|
| OPERATION | ACCUMULATOR | MQ | 14 | 15 | 16 | 17 | CODE |
| Add, Subtract | | underflow | 0 | 0 | 0 | 1 | 01 |
| Multiply | underflow | underflow | 0 | 0 | 1 | 1 | 03 |
| Round | overflow | | 0 | 1 | 1 | 0 | 06 |
| | overflow | overflow | 0 | 1 | 1 | 1 | 07 |
| Divide | | underflow | 1 | 0 | 0 | 1 | 11 |
| | underflow | | 1 | 0 | 1 | 0 | 12 |
| | underflow | underflow | 1 | 0 | 1 | 1 | 13 |
| | | overflow | 1 | 1 | 0 | 1 | 15 |

## Shifting Operations

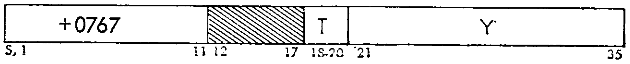
Shift instructions are used to move the contents of the AC and/or the MQ either to the right or the left of their original positions. With the exception of the ROTATE MQ LEFT instruction, zeros are automatically introduced in the vacated positions of a register. Thus, a shift larger than the bit capacity of the register will cause the contents of the register to be replaced by zeros.

When a shift instruction is interpreted, the amount of the shift is determined by bit positions 28-35 of the instruction. This provides a maximum shift of 377 places. Any number larger than 377 is interpreted as modulo 400. By modulo 400 is meant that, given any shift count, the actual number of positions shifted will be the remainder after dividing the shift count by 400.

All shift instructions are subject to address modification through indexing. Shifting a number in a register is equivalent to multiplying or dividing it by a power of 2 (as long as none of the significant bits is lost).

In the following description of the shift instructions, the number of positions to be shifted is specified by "positions 28-35." With indexing, this shift is modified by positions 10-17 of the specified index register or registers.

### ALS — Accumulator Left Shift



*Description.* This instruction causes the $C(AC)_{Q,\ P,\ 1\text{-}35}$ to be shifted left the number of places specified in positions 28-35 of the address portion of the instruction. The sign position is unchanged.

*Indicators.* AC overflow.

*Timing:* 2-5 cycles, modification 7

*Execution.* If a non-zero bit is shifted into position P from position 1, the AC overflow indicator is turned on. Bits shifted past position Q are lost. Vacated positions are filled with zeros (Figure 29).

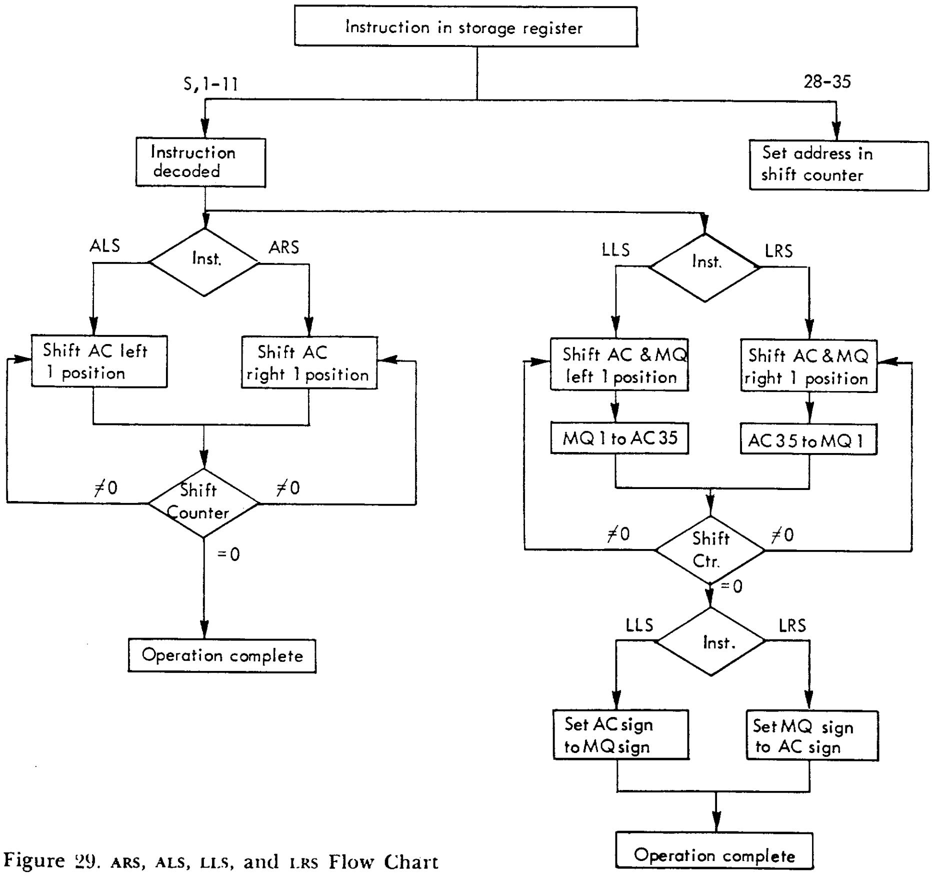


Figure 29. ARS, ALS, LLS, and LRS Flow Chart

### ARS — Accumulator Right Shift

| | | | |
|---|---|---|---|
| +0071 | ▨ | T | Y |
| S, 1 11 | 12 17 | 18 20 | 21 35 |

*Description.* The C (AC)$_{Q,\,P,\,1\text{-}35}$ are shifted right the number of places specified in positions 28-35 of the address portion of the instruction. The sign position is unchanged.

*Indicators.* None.

*Timing:* 2-5 cycles, modification 7

*Execution.* Bits shifted past position 35 of the accumulator are lost. Bits shifted from Q enter P and bits from P enter position 1. Vacated positions are filled with zeros (Figure 29).

### LLS — Long Left Shift

| | | | |
|---|---|---|---|
| +0763 | ▨ | T | Y |
| S, 1 11 | 12 17 | 18 20 | 21 35 |

*Description.* The C (AC)$_{Q,\,P,\,1\text{-}35}$ and the C (MQ)$_{1\text{-}35}$ are treated as one register. The contents of these registers are shifted left the number of places specified in positions 28-35 of the address portion of the instruction. The MQ sign position is unchanged and the sign of the AC is made to agree with it.

*Indicators.* AC overflow.

*Timing:* 2-8 cycles, modification 7

*Execution.* Bits enter position 35 of the AC from position 1 of the MQ. If a non-zero bit is shifted into or through position P, the AC overflow indicator is turned on. Bits shifted past position Q are lost. Positions vacated are filled with zeros (Figure 29).

### LRS — Long Right Shift

| | | | |
|---|---|---|---|
| +0765 | ▨ | T | Y |
| S, 1 11 | 12 17 | 18 20 | 21 35 |

*Description.* The C (AC)$_{Q,\,P,\,1\text{-}35}$ and the C (MQ)$_{1\text{-}35}$ are treated as one register. The contents of these registers are shifted right the number of places specified in positions 28-35 of the address portion of the instruction. The AC sign is unchanged and the sign of the MQ is made to agree with it.

*Indicators.* None.

*Timing:* 2-8 cycles, modification 7

*Execution.* Bits enter position 1 of the MQ from position 35 of the AC. Bits shifted past position 35 of the MQ are lost. Vacated positions are filled with zeros (Figure 29).

Figure 29 shows the flow chart for the ARS, ALS, LLS, and LRS instructions.

### LGL — Logical Left Shift

| | | | |
|---|---|---|---|
| -0763 | ▨ | T | Y |
| S, 1 11 | 12 17 | 18 20 | 21 35 |

*Description.* The C (AC)$_{Q,\,P,\,1\text{-}35}$ and the C (MQ)$_{S,\,1\text{-}35}$ are treated as one register. Their contents are shifted left the number of places specified in positions 28-35 of the address portion of the instruction. The sign of the AC is unchanged.

*Indicators.* AC overflow.

*Timing:* 2-8 cycles, modification 7

*Execution.* Bits enter position S of the MQ from position 1 of the MQ. Bits from MQ (S) then enter position 35 of the accumulator. If a non-zero bit is shifted into or through position P of the AC, the AC overflow indicator is turned on. Bits are shifted from P to Q and any bits shifted from Q are lost. Vacated positions are filled with zeros.

### LGR — Logical Right Shift

| | | | |
|---|---|---|---|
| -0765 | ▨ | T | Y |
| S, 1 11 | 12 17 | 18 20 | 21 35 |

*Description.* The C (AC)$_{Q,\,P,\,1\text{-}35}$ and the C (MQ)$_{S,\,1\text{-}35}$ are treated as one register. Their contents are shifted right the number of places specified in positions 28-35 of the address portion of the instruction. The sign of the AC is unchanged.

*Indicators.* None.

*Timing:* 2-8 cycles, modification 7

*Execution.* Bits enter position S of the MQ from position 35 of the AC. Bits enter MQ 1 from MQ (S). Bits shifted past position 35 of the MQ are lost. Vacated positions are filled with zeros.

### RQL — Rotate MQ Left

| | | | |
|---|---|---|---|
| -0773 | ▨ | T | Y |
| S, 1 11 | 12 17 | 18 20 | 21 35 |

*Description.* The C (MQ) are shifted left the number of places specified by positions 28-35 of the address portion of the instruction. The instruction shifts position S into position 35, and thus the register becomes a circular one.

*Indicators.* None.

*Timing:* 2-5 cycles, modification 7

*Execution.* Bits are rotated from position 1 of the MQ to position S, and from position S to position 35. No bits are lost.

## Word Transmission Operations

The operations described in this section are concerned with the movement of words or parts of words from one core location or register to another.

### LDQ — Load MQ

|  |  |  |  |  |
|---|---|---|---|---|
| +0560 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction places the contents of Y into the MQ. The C (Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

### STQ — Store MQ

|  |  |  |  |  |
|---|---|---|---|---|
| -0600 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction places the contents of the MQ into the specified Y location. The C (MQ) remain unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### SLQ — Store Left Half MQ

|  |  |  |  |  |
|---|---|---|---|---|
| -0620 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (MQ)$_{S,\,1\text{-}17}$ replace the C (Y)$_{S,\,1\text{-}17}$. The C (MQ) and the C (Y)$_{18\text{-}35}$ are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### STO — Store

|  |  |  |  |  |
|---|---|---|---|---|
| +0601 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (AC)$_{S,\,1\text{-}35}$ replace the C (Y). The C (AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### SLW — Store Logical Word

|  |  |  |  |  |
|---|---|---|---|---|
| +0602 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C(AC)$_{P,1\text{-}35}$ replace the C(Y). The C (AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### STP — Store Prefix

|  |  |  |  |  |
|---|---|---|---|---|
| +0630 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (AC)$_{P,\,1,\,2}$ replace the C (Y)$_{S,\,1,\,2}$. The C (Y)$_{3\text{-}35}$ and the C (AC) are unchanged.

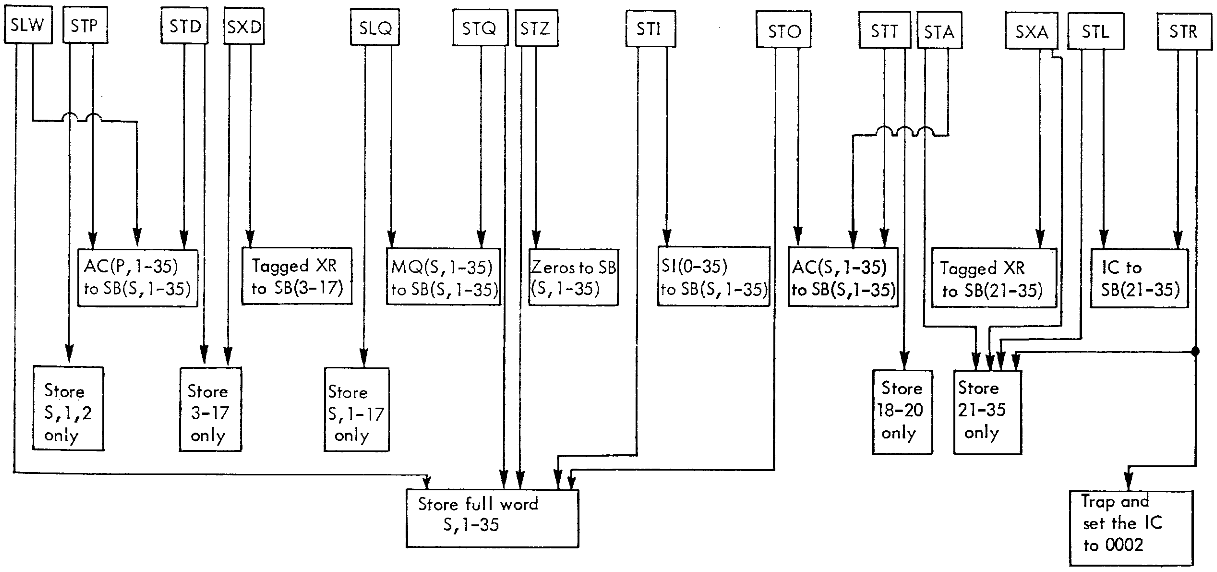


Figure 30. Data Flow Chart for Store Instructions

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### STD — Store Decrement

| | | | | |
|---|---|---|---|---|
| +0622 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (AC)$_{3-17}$ replace the C (Y)$_{3-17}$. The C (Y)$_{S, 1, 2, 18-35}$ and the C (AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### STT — Store Tag

| | | | | |
|---|---|---|---|---|
| +0625 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (AC)$_{18-20}$ replace the C (Y)$_{18-20}$. The C (Y)$_{S, 1-17, 21-35}$ and the C (AC) remain unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### STA — Store Address

| | | | | |
|---|---|---|---|---|
| +0621 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (AC)$_{21-35}$ replace the C (Y)$_{21-35}$. The C (Y)$_{S, 1-20}$ and the C (AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### STL — Store Instruction Location Counter

| | | | | |
|---|---|---|---|---|
| -0625 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The location of the STL instruction plus 1 replaces the C (Y)$_{21-35}$. The C (Y)$_{S, 1-20}$ are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* Instruction counter contents to address switches. Address switches to the storage bus. SB (21-35) to storage. See Figure 30.

### STR — Store Location and Trap

| | |
|---|---|
| -1 | ▨ |
| S,1-2 | 3 35 |

*Description.* The location of the STR instruction, plus one, replaces positions 21-35 of location 0000. The computer then takes its next instruction from location 0002. The contents of positions 3-35 of this instruction are not interpreted by the computer.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.*

NOTE: Conflicts may arise when the computer is operated in the trapping mode. This instruction, TRANSFER TRAP MODE, and floating-point trap, all use location 0000. See Figure 30.

### STZ — Store Zero

| | | | | |
|---|---|---|---|---|
| +0600 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* The C (Y)$_{1-35}$ are replaced by zeros and the C (Y)$_{S}$ are made plus.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 30.

### XCA — Exchange AC and MQ

| | |
|---|---|
| +0131 | ▨ |
| S, 1 11 | 12 35 |

*Description.* The C (AC)$_{S, 1-35}$ are exchanged with the C(MQ)$_{S,1-35}$. Positions P and Q of the AC are cleared.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 31.

### XCL — Exchange Logical AC and MQ

| | |
|---|---|
| -0130 | ▨ |
| S 1 11 | 12 35 |

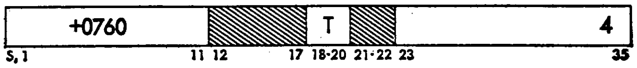
*Description.* The C (AC)$_{P, 1-35}$ are exchanged with the C (MQ)$_{S, 1-35}$. Positions S and Q of the AC are cleared.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 31.

### ENK — Enter Keys



*Description.* This instruction places the contents of 36 panel input switches into the C(MQ). When a panel input switch is down it represents a 1; when it is up, it represents a zero.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* Since the address part of this instruction contains part of the operation code for this instruction, any address modification by an index register may result in the changing of the operation itself.

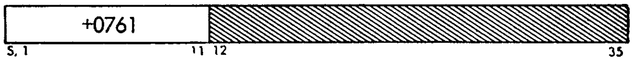
## Control Instructions

Instructions which govern the flow of a program, and in particular those which cause an alteration in the computer's normal process of taking its instructions from sequential locations, are called *control instructions.*

*Unconditional transfer instructions* specify the location "Y" from which the computer is to take the next instruction. *Conditional transfer instructions* also specify a location Y. However, whether the computer takes its next instruction from Y or the next sequential location depends upon the outcome of a test. This test is specified by the operation code of the instruction.

*Test instructions* are similar to conditional control instructions in that they cause some test to be performed. Unlike conditional instructions, however, test instructions do not specify a location Y to which control may be transferred. Instead, the alternative location to which control may be transferred is fixed relative to the location of the test instruction.

### NOP — No Operation

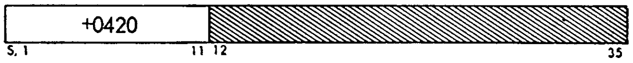


*Description.* This instruction causes the computer to take the next instruction in sequence.

*Indicators.* None.

*Timing:* 2 cycles

### HPR — Halt and Proceed



*Description.* This instruction causes the computer to halt. The IC contains the location of the next se-

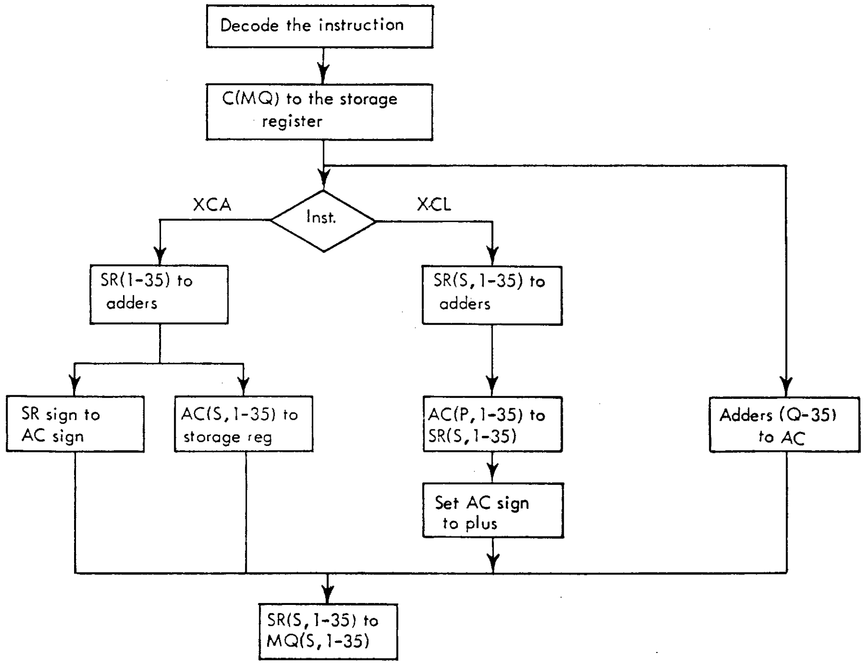


Figure 31. XCA and XCL Flow Chart

quential instruction. When the start key on the operator's console is depressed, the computer proceeds and executes the next sequential instruction.

*Indicators.* None.

*Timing:* 2 cycles

### HTR — Halt and Transfer

| | | | | |
|---|---|---|---|---|
| +0000 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction causes the computer to halt. The IC contains the location of the HTR instruction. Depression of the start key, on the operator's console, causes the computer to transfer to location Y and execute that instruction.

*Indicators.* Trap Mode.

*Timing:* 2 cycles

### XEC — Execute

| | | | | |
|---|---|---|---|---|
| + 0522 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction causes the computer to perform or "execute" the instruction at location Y.

*Indicators.* None.

*Timing:* 1 cycle

*Execution.* Since the location counter is not altered (when Y contains any instruction other than a successful transfer or test instruction), the program advances to the next sequential instruction following the execute instruction after performing the instruction at location Y. If location Y contains a transfer instruction, it will be executed and program control will be altered from the sequential process. If location Y contains a test instruction, the instruction following EXECUTE will be located relative to the EXECUTE rather than the TEST instruction. Thus, any instruction which changes the instruction counter (STR, TRA, DCT, etc.) will alter program control when that instruction is executed by XEC.

### TRA — Transfer

| | | | | |
|---|---|---|---|---|
| +0020 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* This instruction causes the computer to take its next instruction from location Y and proceed from there.

*Indicators.* Trap Mode.

*Timing:* 2 cycles

*Execution.* See Figure 32.

### ETM — Enter Trapping Mode

| | | | | |
|---|---|---|---|---|
| +0760 | ▨ | T | ▨ | 7 |
| S,1 11 | 12 17 | 18-20 | 21-22 | 23 35 |

*Description.* This instruction causes the computer to enter the transfer trapping mode. The transfer trapping indicator on the operator's console is turned on.

*Indicators.* Trap Mode.

*Timing:* 2 cycles

*Execution.* The computer operates in the trapping mode until either a LEAVE TRAPPING MODE OPERATION is executed or the clear or reset key on the operator's console is depressed. Since positions 23-35 represent part of the operation code of this instruction, any modification by an index register may result in the changing of the operation itself. See Figure 32.

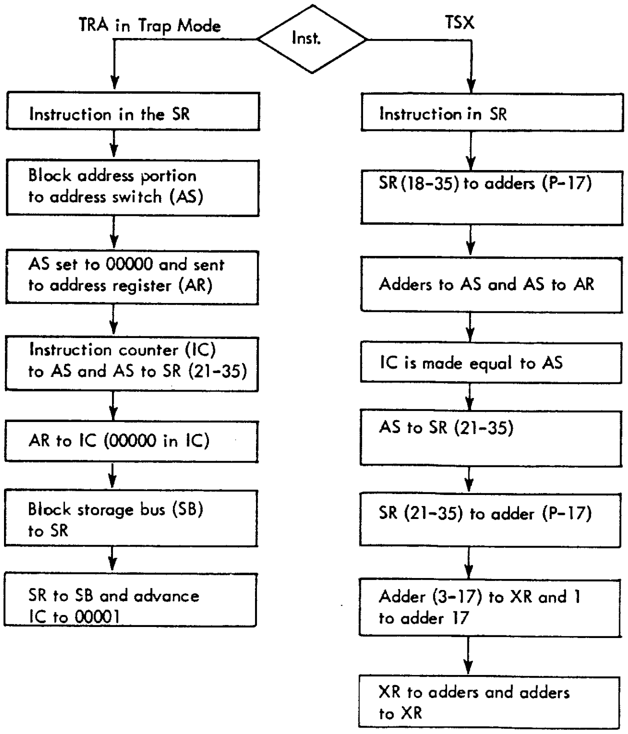
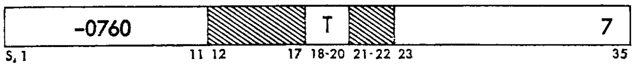

TRA in Trap Mode
Inst.
TSX
Instruction in the SR
Block address portion to address switch (AS)
AS set to 00000 and sent to address register (AR)
Instruction counter (IC) to AS and AS to SR (21-35)
AR to IC (00000 in IC)
Block storage bus (SB) to SR
SR to SB and advance IC to 00001
Instruction in SR
SR(18-35) to adders (P-17)
Adders to AS and AS to AR
IC is made equal to AS
AS to SR (21-35)
SR (21-35) to adder (P-17)
Adder (3-17) to XR and 1 to adder 17
XR to adders and adders to XR

Figure 32. TRA, TSX, and Trap Mode Flow Chart

### LTM — Leave Trapping Mode



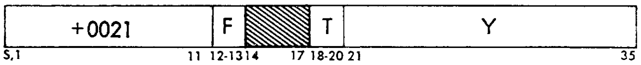
*Description.* This instruction turns off the trap mode indicator and causes the computer to leave the transfer trapping mode. Transfer instructions, therefore, will not be trapped again until an ETM operation is executed.

*Indicators.* Trap Mode.

*Timing:* 2 cycles

*Execution.* When the computer is in the trapping mode and any transfer instruction except a TTR is executed, the location of the transfer instruction replaces the address part of location 0000 whether the condition for transferring is met or not. If the transfer condition is met, the computer takes its next instruction from location 0001 and proceeds from there. Only instructions which have "transfer" in their title are affected by the transfer trapping mode. Address modification may change the operation, since positions 23-35 of the instruction are a part of the operation code.

### TTR — Trap Transfer



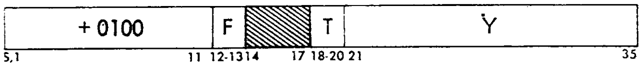
*Description.* This instruction causes the computer to take its next instruction from location Y and to proceed from there whether in the transfer trap mode or not. This makes it possible to have an unconditional transfer in the transfer trapping mode.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 32.

### TZE — Transfer on Zero



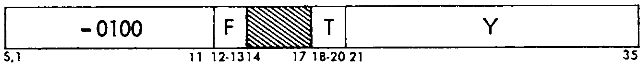
*Description.* If the $c(AC)_{Q,P,1\text{-}35}$ are zero, the computer takes its next instruction from location Y and proceeds from there. If they are not zero, the next sequential instruction is taken.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 33.

### TNZ — Transfer on No Zero



*Description.* If the $c(AC)_{Q,P,1\text{-}35}$ are not zero, the computer takes its next instruction from location Y and proceeds from there. If they are zero, the next sequential instruction is taken.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 33.

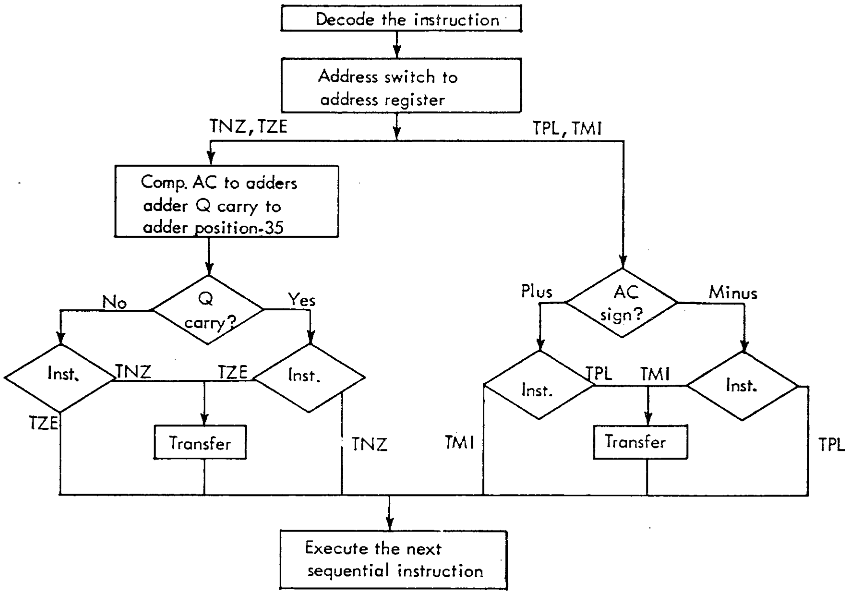


Figure 33. TNZ, TZE, TPL, and TMI Flow Chart

### TPL — Transfer on Plus

| | | | | |
|---|---|---|---|---|
| +0120 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* If the sign position of the AC is positive, the computer takes its next instruction from location Y and proceeds from there. If the sign position is negative, the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 33.

### TMI — Transfer on Minus

| | | | | |
|---|---|---|---|---|
| -0120 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* If the sign position of the AC is negative, the computer takes its next instruction from location Y and proceeds from there. If the sign position is positive, the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 33.

### TOV — Transfer on Overflow

| | | | | |
|---|---|---|---|---|
| +0140 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* If the AC overflow indicator is on, it is turned off and the computer takes its next instruction from location Y. If the indicator is off, the computer takes the next sequential instruction.

*Indicators.* AC overflow, trap mode.

*Timing*: 2 cycles

*Execution.* See Figure 34.

### TNO — Transfer on No Overflow

| | | | | |
|---|---|---|---|---|
| -0140 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* If the AC overflow indicator is off, the computer takes its next instruction from location Y. If the indicator is on, it is turned off and the computer takes the next sequential instruction.

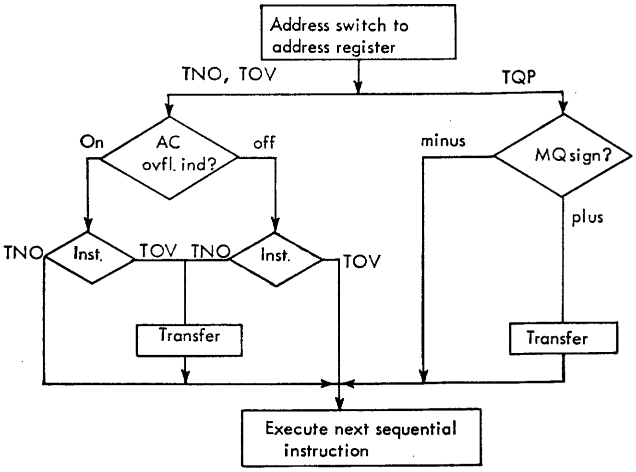


Figure 34. TOV, TNO, and TQO Flow Chart

*Indicators.* AC overflow, trap mode.

*Timing*: 2 cycles

*Execution.* See Figure 34.

### TQP — Transfer on MQ Plus

| | | | | |
|---|---|---|---|---|
| +0162 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

*Description.* If the sign position of the MQ is plus, the computer takes its next instruction from location Y. If the sign position is negative, the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing*: 2 cycles

*Execution.* See Figure 34.

### TQO — Transfer on MQ Overflow

| | | | | |
|---|---|---|---|---|
| +0161 | F | ▨ | T | Y |
| S,1 11 | 12-13 | 14 17 | 18-20 | 21 35 |

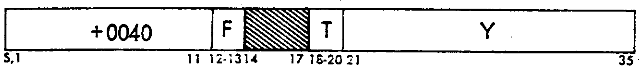
*Description.* This instruction is a conditional transfer when the computer is operating in the 704 floating-point mode. If the MQ overflow indicator is on, the computer takes its next instruction from location Y and turns the indicator off.

*Indicators.* MQ overflow.

*Timing*: 2 cycles

*Execution.* If this instruction is executed while the computer is in the normal mode, it is treated as a no-operation whether the MQ overflow indicator is on or not.

### TLQ — Transfer on Low MQ



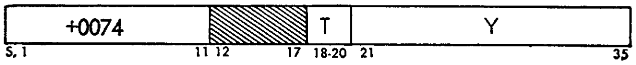
*Description.* If the C(MQ) are algebraically less than the C(AC), the computer takes its next instruction from location Y. If the C(MQ) are algebraically greater than or equal to the C(AC), the computer takes the next sequential instruction. NOTE: a plus zero is algebraically greater than a minus zero.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 35.

### TSX — Transfer and Set Index



*Description.* This instruction places the 2's complement of the instruction counter contents in the specified index register (T). The computer takes its next instruction from location Y. NOTE: Subtracting the 2's complement of a number is equivalent to adding the number.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 32.

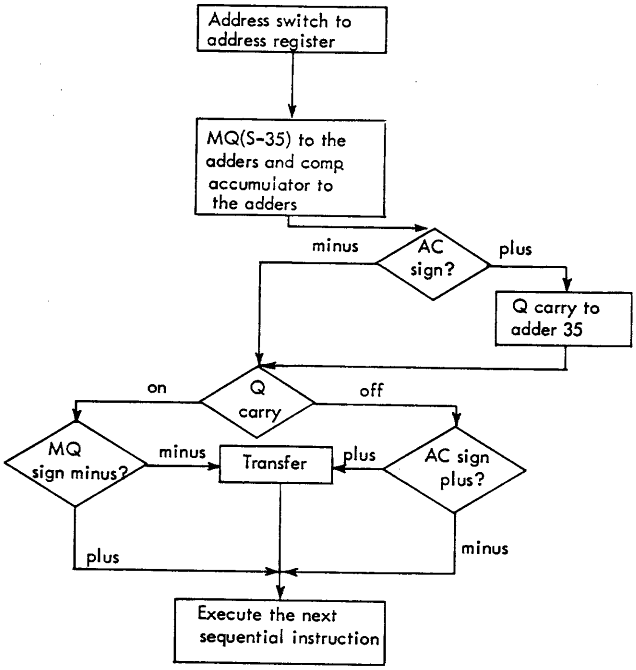
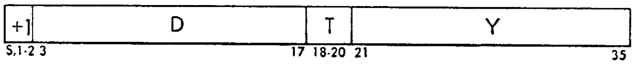

Address switch to address register
MQ(S-35) to the adders and comp accumulator to the adders
minus
AC sign?
plus
Q carry to adder 35
on
Q carry
off
MQ sign minus?
minus
Transfer
plus
AC sign plus?
plus
minus
Execute the next sequential instruction

Figure 35. TLQ Flow Chart

### TXI — Transfer with Index Incremented



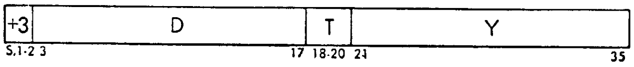
*Description.* This instruction adds the decrement (D) to the contents of the specified index register (T) and replaces the contents of the index register with the resulting sum. The computer then takes its next instruction from location Y.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 36.

### TXH — Transfer on Index High



*Description.* If the number in the specified index register (T) is greater than the decrement (D), the computer takes its next instruction from location Y. If the number in the specified index register is less than or equal to D, the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 37.

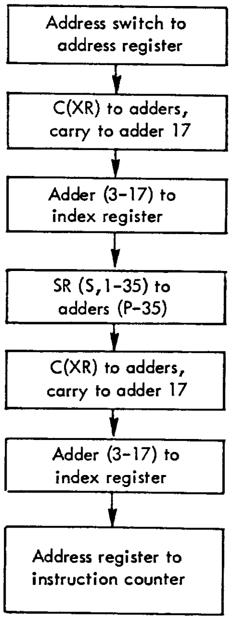

Address switch to address register
C(XR) to adders, carry to adder 17
Adder (3-17) to index register
SR (S,1-35) to adders (P-35)
C(XR) to adders, carry to adder 17
Adder (3-17) to index register
Address register to instruction counter

Figure 36. TXI Flow Chart

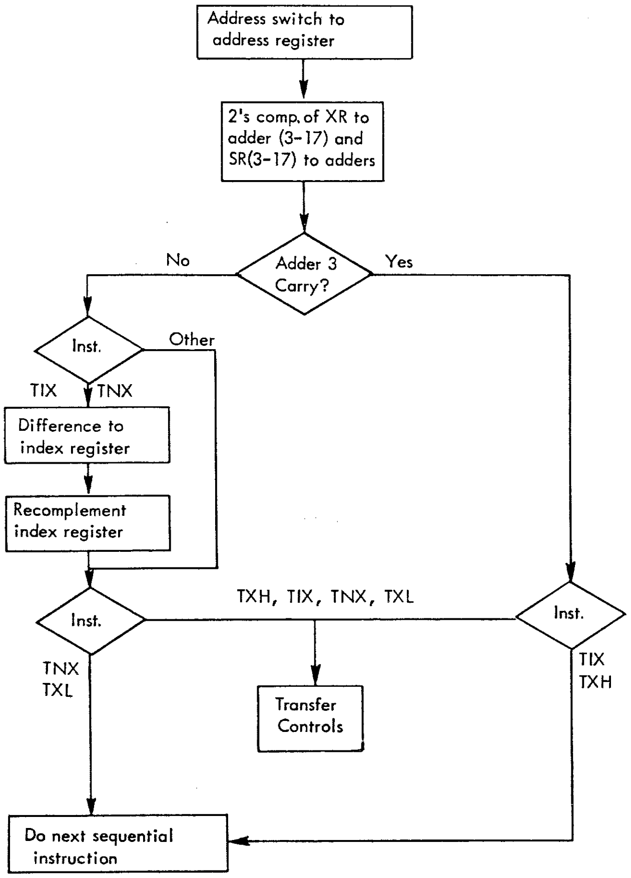
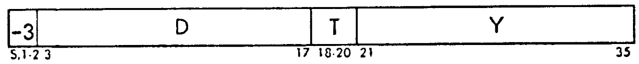

Address switch to address register
2's comp. of XR to adder (3-17) and SR(3-17) to adders
Adder 3 Carry?
No
Yes
Inst.
Other
TIX
TNX
Difference to index register
Recomplement index register
TXH, TIX, TNX, TXL
Inst.
Inst.
TNX
TXL
TIX
TXH
Transfer Controls
Do next sequential instruction

Figure 37. TIX, TXH, TNX and TXL Flow Chart

### TXL — Transfer on Index Low or Equal



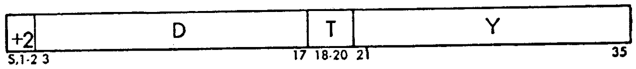
*Description.* If the contents of the index register specified by T are less than or equal to the D portion, the computer takes its next instruction from location Y. If the contents of T are greater than D, the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 37.

### TIX — Transfer on Index



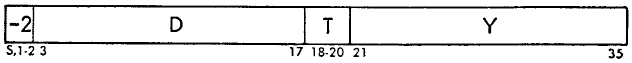
*Description.* If the C(XR) specified by T are greater than the C(D), the number in the index register is reduced by D and the computer takes its next instruction from Y. If C(T) is less than or equal to D, the C(T) are unchanged and the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 37.

### TNX — Transfer on No Index



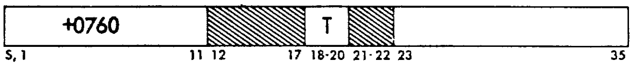
*Description.* If the C(XR) specified by T are equal to or less than D, the C(T) are unchanged and the computer takes its next instruction from Y. If C(T) are greater than D, the C(T) are reduced by D and the computer takes the next sequential instruction.

*Indicators.* Trap mode.

*Timing:* 2 cycles

*Execution.* See Figure 37.

### PSE — Plus Sense



*Description.* This instruction provides a means of testing the status of the sense switches and of turning on or off the sense lights on the operator's console. The instruction also permits the transmission of an impulse to or from the exit or entry hubs on the printer or card punch control panels. Address modification may cause the operation code to be changed.

*Indicators.* Sense indicators and switches.

*Timing:* 2 cycles

*Execution.* The address part (23-35) of this instruction determines whether a light, switch, printer, or card punch is being sensed. Further, it determines which light, switch, or hub is sensed. The octal addresses for sense instructions are:

| | |
|---|---|
| 0030 | Advances film of CRT Recorder. |
| 0140 | Turns off all sense lights on the operator's console. |
| 0141 - 0144 | Turn on sense lights 1, 2, 3 or 4, respectively, on the console. |
| 0161 - 0166 | Test sense switches on the console. If the corresponding switch is down (on), the computer skips the next instruction and proceeds from there. If the sense switch is up (off), the computer takes the next sequential instruction. |

1341 - 1342 (A)
3341 - 3342 (C)
5341 - 5342 (E)

An impulse will appear at the specified exit hub of the card punch control panel attached to appropriate data channel. Hubs are numbered 1 and 2.

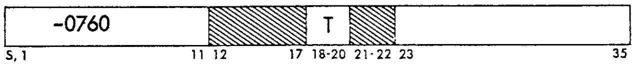
1360 (A)
3360 (C)
5360 (E)

If an impulse is present at the sense entry hub of the printer control panel, the computer skips the next instruction and proceeds from there. If no impulse is present, the computer takes the next sequential instruction.

1361 - 1372 (A)
3361 - 3372 (C)
5361 - 5372 (E)

The computer causes an impulse to appear at the specified hub on the control panel of the printer attached to that particular data channel.

## MSE — Minus Sense



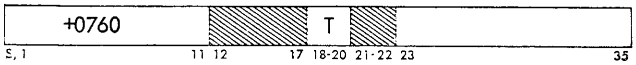
*Description.* This instruction provides a means of testing the status of the sense lights on the operator's console. The lights may be turned on by a PSE instruction with an address of 0141 to 0144. Address modification may cause the operation code to be changed.

*Indicators.* Sense lights.

*Timing:* 2 cycles

*Execution.* The addresses of the four sense lights are 0141 to 0144. If the corresponding sense light is on, the light is turned off and the computer skips the next instruction and proceeds from there. If the light is off, the computer executes the next sequential instruction.

## BTT — Beginning of Tape Test



*Description.* This instruction is used to test the status of data channel beginning-of-tape indicators. The channel whose indicator is to be tested is specified by the address portion (Y) of the BTT instruction. Address modification may cause the operation code to be changed. The addresses for the channels are:

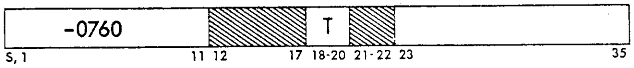
| | |
|---|---|
| Data channel A | 1000 |
| Data channel B | 2000 |
| Data channel C | 3000 |
| Data channel D | 4000 |
| Data channel E | 5000 |
| Data channel F | 6000 |

*Indicators.* All beginning-of-tape indicators.

*Timing:* 2 cycles

*Execution.* If the beginning-of-tape indicator for data channel Y is on, the computer takes the next sequential instruction, and the indicator is turned off. If the beginning-of-tape indicator is off, the computer skips the next instruction and proceeds from there. The beginning-of-tape indicator is turned on by a backspace record or backspace file instruction given to a tape unit that is positioned at its load point. See Figure 38.

## ETT — End of Tape Test



*Description.* This instruction is used to test the status of data channel end-of-tape indicators. The channel whose indicator is to be tested is specified by the address portion of the ETT instruction. The addressing system is the same as specified for the BTT instruction. Address modification may cause the operation code to be changed.

*Indicators.* End-of-tape indicators.

*Timing:* 2 cycles

*Execution.* If the end-of-tape indicator for data channel Y is on, the computer takes the next sequential instruction and turns the indicator off. If the

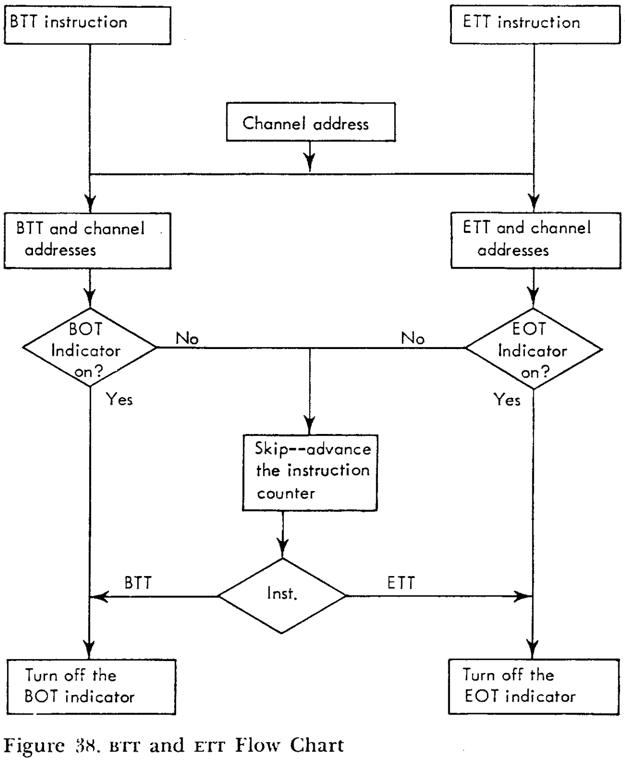
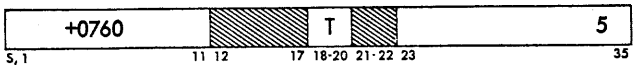

BTT instruction
ETT instruction
Channel address
BTT and channel addresses
ETT and channel addresses
BOT Indicator on?
No
No
EOT Indicator on?
Yes
Yes
Skip--advance the instruction counter
BTT
Inst.
ETT
Turn off the BOT indicator
Turn off the EOT indicator

Figure 38. BTT and ETT Flow Chart

indicator is off, the computer skips the next instruction and proceeds from there. The end-of-tape indicator is turned on when either a write select or a write end of file causes the end-of-tape marker to be passed over. See Figure 38.

### IOT — Input-Output Check Test



*Description.* If the I-O check indicator is on, the indicator is turned off and the computer takes the next sequential instruction. If the indicator is off, the computer skips the next instruction and proceeds from there. Any address modification may result in the changing of the operation itself.

*Indicators.* I-O check.

*Timing:* 2 cycles

*Execution.* The I-O check indicator will be turned on by any of the following conditions:

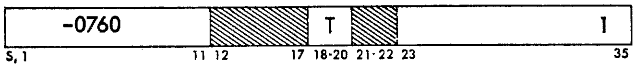
1. If a drum select instruction without a copy or locate-drum-address instruction is executed.

2. If a RESET AND LOAD CHANNEL or a LOAD CHANNEL is executed and the specified channel is not selected.

3. If, when writing, a channel data register has not been loaded with a word from storage by the time its contents are to be sent to the output unit.

4. If, when reading, a channel data register has not stored its contents by the time new data are to be loaded from an input unit.

### PBT — P-Bit Test



*Description.* If the $\mathrm{C(AC)}_{P}$ are a 1, the computer skips the next instruction and proceeds from there. If P contains a 0, the computer takes the next sequential instruction. Address modification may result in the changing of the instruction itself.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 39.

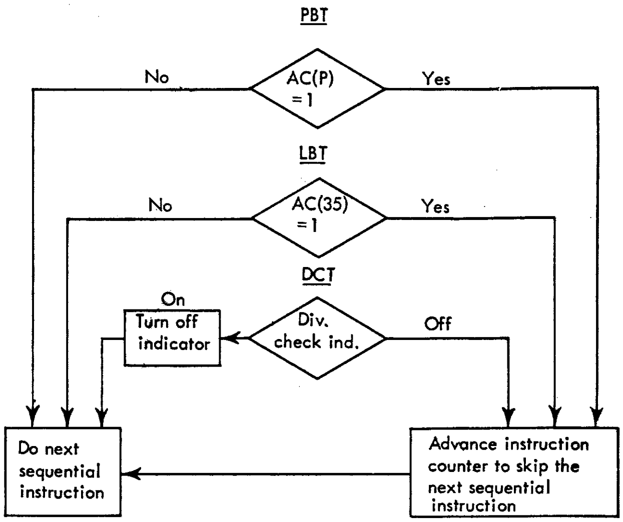
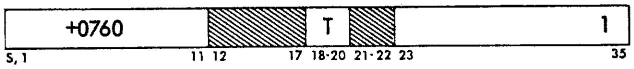

PBT
No
AC(P) = 1
Yes
LBT
No
AC(35) = 1
Yes
DCT
On
Turn off indicator
Div. check ind.
Off
Do next sequential instruction
Advance instruction counter to skip the next sequential instruction

Figure 39. PBT, LBT, and DCT Flow Chart

### LBT — Low-Order Bit Test



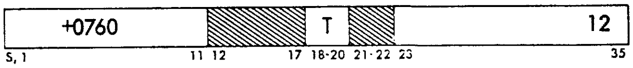
*Description.* If the $\mathrm{C(AC)}_{35}$ is a 1, the computer skips the next instruction and proceeds from there. If 35 is a 0, the computer takes the next sequential instruction. Address modification may result in the changing of the instruction.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 39.

### DCT — Divide Check Test



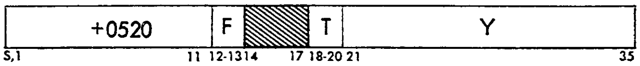
*Description.* If the indicator is on, it is turned off and the computer takes the next sequential instruction. If the indicator is off, the computer skips the next instruction and proceeds from there. Address modification may result in the changing of the instruction itself.

*Indicators.* Divide check.

*Timing:* 2 cycles

*Execution.* See Figure 39.

### ZET — Storage Zero Test



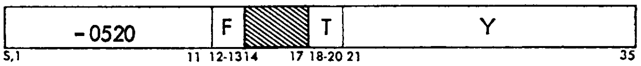
*Description.* If the C(Y)$_{1\text{-}35}$ are 0, the computer skips the next instruction and proceeds from there. If the C(Y)$_{1\text{-}35}$ are not zero, the computer takes the next sequential instruction. The C(Y) are not changed.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 40.

### NZT — Storage not Zero Test



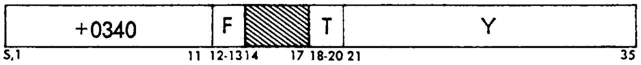
*Description.* If the C(Y)$_{1\text{-}35}$ are not 0, the computer skips the next instruction and proceeds from there. If the C(Y)$_{1\text{-}35}$ are 0, the computer takes the next sequential instruction. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 40.

### CAS — Compare Accumulator with Storage



*Description.* If the C(AC) are algebraically greater than the C(Y), the computer takes the next sequential instruction. If the C(AC) are algebraically equal to the C(Y), the computer skips the next instruction and proceeds from there. If the C(AC) are algebraically less than the C(Y) the computer skips the next two instructions and proceeds from there.

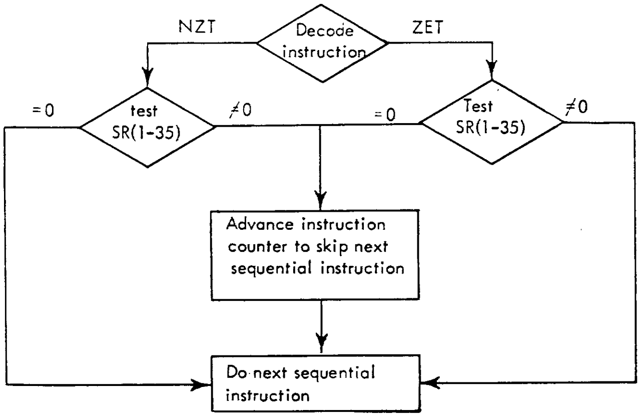

NZT
Decode instruction
ZET
=0
test SR(1-35)
≠0
=0
Test SR(1-35)
≠0
Advance instruction counter to skip next sequential instruction
Do next sequential instruction

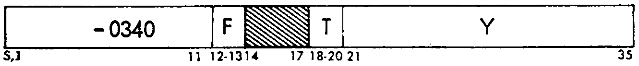
Figure 40. NZT and ZET Flow Chart

*Indicators.* None.

*Timing:* 3 cycles

*Execution.* NOTE: Two numbers are considered algebraically equal if the magnitudes and signs of both are equal. A plus zero is algebraically greater than a minus zero.

### LAS — Logical Compare Accumulator with Storage



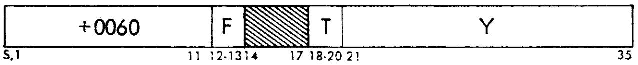
*Description.* The C(AC)$_{Q,P,1\text{-}35}$ are treated as an unsigned 37-bit number and are compared with the C(Y)$_{S,1\text{-}35}$ which are treated as an unsigned 36-bit quantity. If the C(AC)$_{Q,P,1\text{-}35}$ are greater than the C(Y), the computer takes the next sequential instruction. If the C(AC)$_{Q,P,1\text{-}35}$ are equal to the C(Y), the computer skips the next instruction and proceeds from there. If the C(AC)$_{Q,P,1\text{-}35}$ are less than the C(Y), the computer skips the next two instructions and proceeds from there.

*Indicators.* None.

*Timing:* 3 cycles

FOR THE FOLLOWING control operations that refer to data channels, the description of the operation is given for channel A. For the other channels, the operation code and the title are given.

### TCOA — Transfer on Channel A in Operation



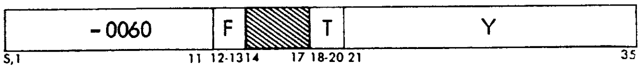
*Description.* If channel A is in operation, the computer takes its next instruction from location Y. If the channel is not in operation, the computer takes the next sequential instruction, and the operation of the channel is not affected. The channel is in operation as long as a select register contains information.

*Indicators.* Trap.

*Timing:* 2 cycles

| INSTRUCTION | CODE | NAME |
|---|---|---|
| TCOB | +0061 | Transfer on Channel B in Operation |
| TCOC | +0062 | Transfer on Channel C in Operation |
| TCOD | +0063 | Transfer on Channel D in Operation |
| TCOE | +0064 | Transfer on Channel E in Operation |
| TCOF | +0065 | Transfer on Channel F in Operation |

### TCNA — Transfer on Channel A not in Operation



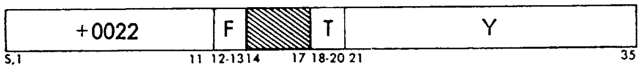
*Description.* If channel A is not in operation, the computer takes its next instruction from location Y. If the channel is in operation, the computer takes the next sequential instruction, and the operation of the channel is not affected.

*Indicators.* Trap.

*Timing:* 2 cycles.

| INSTRUCTION | CODE | NAME |
|---|---|---|
| TCNB | −0061 | Transfer on Channel B not in Operation |
| TCNC | −0062 | Transfer on Channel C not in Operation |
| TCND | −0063 | Transfer on Channel D not in Operation |
| TCNE | −0064 | Transfer on Channel E not in Operation |
| TCNF | −0065 | Transfer on Channel F not in Operation |

### TRCA — Transfer on Channel A Redundancy Check



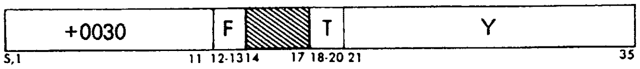
*Description.* If the tape check indicator for channel A is on, it is turned off and the computer takes its next instruction from location Y. If the indicator is off, the next sequential instruction is taken.

*Indicators.* Tape Check, Trap.

*Timing:* 2 cycles.

| INSTRUCTION | CODE | NAME |
|---|---|---|
| TRCB | −0022 | Transfer on Channel B Redundancy Check |
| TRCC | +0024 | Transfer on Channel C Redundancy Check |
| TRCD | −0024 | Transfer on Channel D Redundancy Check |
| TRCE | +0026 | Transfer on Channel E Redundancy Check |
| TRCF | −0026 | Transfer on Channel F Redundancy Check |

### TEFA — Transfer on Channel A End of File



*Description.* If the end-of-file indicator for channel A is on, it is turned off and the computer takes its next instruction from location Y. If the indicator is off, the computer takes the next sequential instruction.

*Indicators.* End-of-file, Trap.

*Timing:* 2 cycles.

| INSTRUCTION | CODE | NAME |
|---|---|---|
| TEFB | −0030 | Transfer on Channel B End of File |
| TEFC | +0031 | Transfer on Channel C End of File |
| TEFD | −0031 | Transfer on Channel D End of File |
| TEFE | +0032 | Transfer on Channel E End of File |
| TEFF | −0032 | Transfer on Channel F End of File |

## Index Transmission Operations

This section of operations deals with the loading and storing of the contents of index registers.

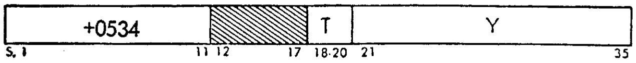
The operations always involve one or more index registers and either the address or decrement field of some location in storage or the accumulator register. The following 15-bit fields may serve as one of the agents in an index transmission operation: the address or decrement of the accumulator, the address or decrement of any location in storage, or the address part of the index transmission instruction itself. In addition, the number to be loaded may be placed in the specified index register in either true or complement form.

Single registers or any combination of index registers may be specified. If more than one register is specified in an unloading operation, their contents are "OR'ed" together to produce the effective number. OR'ing matches the registers position-for-position. If there is a bit in either or both of the registers, the result is a bit. For example:

| | |
|---|---|
| XRA | 101100 |
| XRB | 011000 |
| Result | 111100 |

If more than one index register is specified in a loading operation, the data are loaded into all registers specified.

### LXA — Load Index from Address



*Description.* The $C(Y)_{21-35}$ replace the contents of the specified index register. The C(Y) are unchanged

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* See Figure 41.

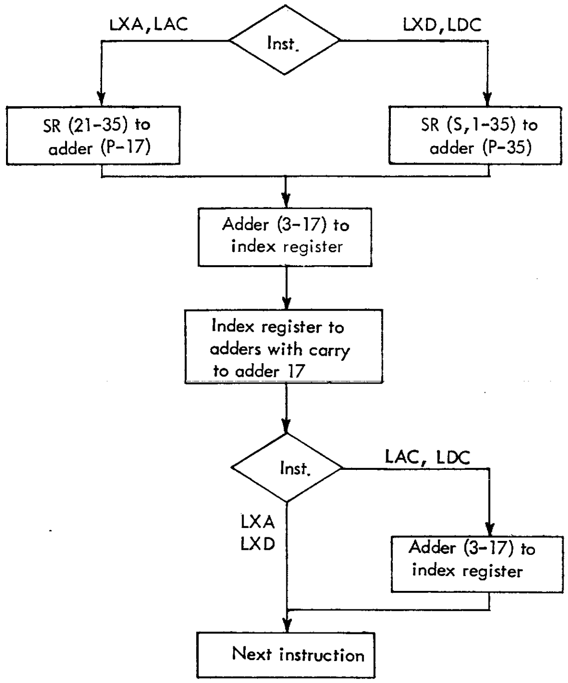
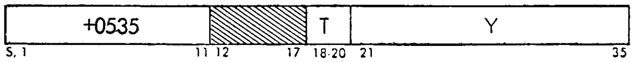

LXA, LAC
Inst.
LXD, LDC
SR (21-35) to adder (P-17)
SR (S, 1-35) to adder (P-35)
Adder (3-17) to index register
Index register to adders with carry to adder 17
Inst.
LAC, LDC
LXA
LXD
Adder (3-17) to index register
Next instruction

Figure 41. LXA, LAC, LXD and LDC Flow Chart

## LAC — Load Complement of Address in Index



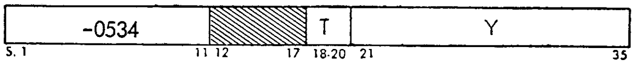
*Description.* The 2's complement of the $C(Y)_{21-35}$ replaces the contents of the specified index register. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 41.

## LXD — Load Index from Decrement



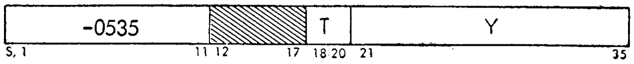
*Description.* The $C(Y)_{3-17}$ replace the contents of the specified index register. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 41.

## LDC — Load Complement of Decrement in Index



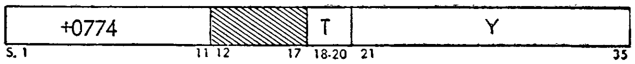
*Description.* The 2's complement of the $C(Y)_{3-17}$ replaces the contents of the specified index register. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 41.

## AXT — Address to Index True



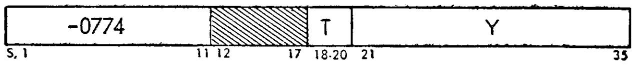
*Description.* Positions 21-35 of this instruction replace the contents of the specified index register. The instruction is unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 42.

## AXC — Address to Index Complemented



*Description.* The 2's complement of positions 21-35 of this instruction replaces the contents of the specified index register. The instruction is unchanged.

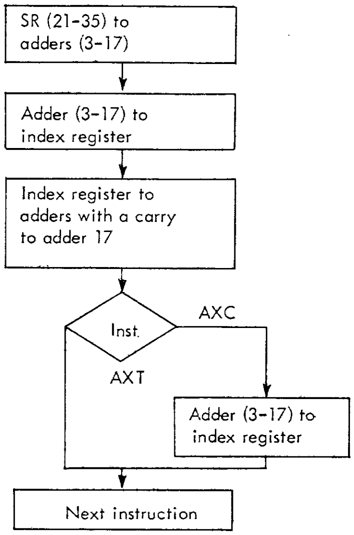

SR (21-35) to adders (3-17)
Adder (3-17) to index register
Index register to adders with a carry to adder 17
Inst.
AXC
AXT
Adder (3-17) to index register
Next instruction

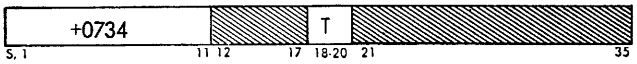
Figure 42. AXT and AXC Flow Chart

*Indicators.* None.

*Timing*: 2 cycles

*Execution.* See Figure 42.

## PAX — Place Address in Index



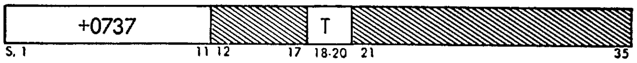
*Description.* The C(AC)$_{21-35}$ replace the contents of the specified index register. The C(AC) are unchanged.

*Indicators.* None.

*Timing*: 2 cycles

*Execution.* See Figure 43.

## PAC — Place Complement of Address in Index



*Description.* The 2's complement of the C(AC)$_{21-35}$ replaces the contents of the specified index register. The C(AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 43.

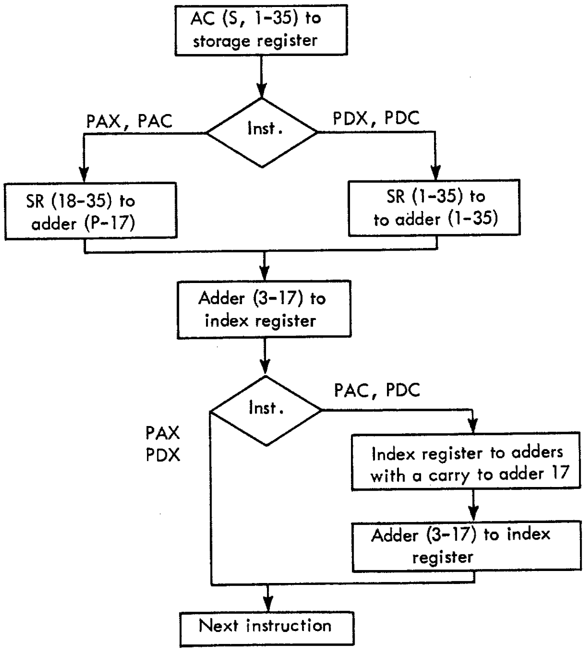
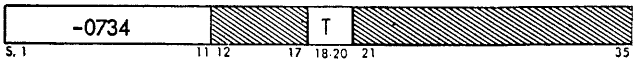

AC (S, 1-35) to storage register
PAX, PAC
Inst.
PDX, PDC
SR (18-35) to adder (P-17)
SR (1-35) to to adder (1-35)
Adder (3-17) to index register
Inst.
PAC, PDC
PAX
PDX
Index register to adders with a carry to adder 17
Adder (3-17) to index register
Next instruction

Figure 43. PAX, PAC, PDX, and PDC Flow Chart

## PDX — Place Decrement in Index



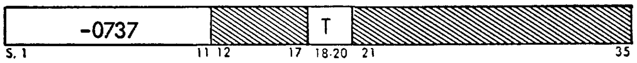
*Description.* The C(AC)$_{3-17}$ replace the contents of the specified index register. The C(AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 43.

## PDC — Place Complement of Decrement in Index



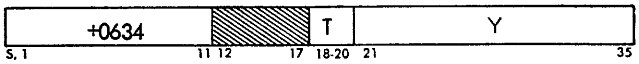
*Description.* The 2's complement of the C(AC)$_{3-17}$ replaces the contents of the specified index register. The C(AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 43.

## SXA — Store Index in Address



*Description.* The C(Y)$_{21-35}$ are replaced by the contents of the specified index register. The C(Y)$_{S,1-20}$ are unchanged.

*Indicators.* None.

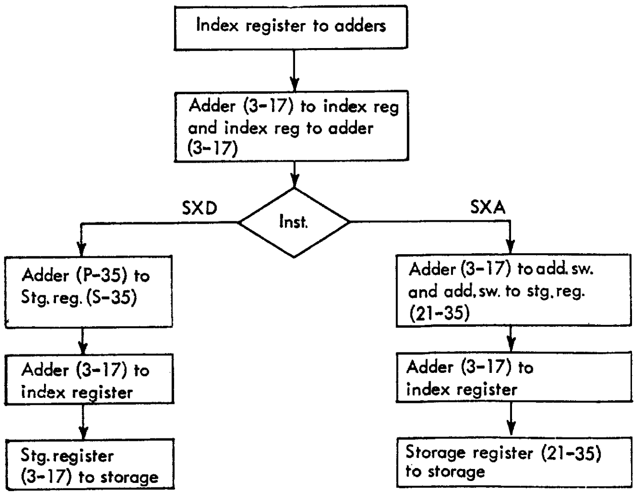
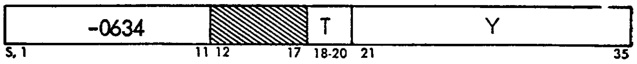

Index register to adders
Adder (3-17) to index reg and index reg to adder (3-17)
SXD
Inst.
SXA
Adder (P-35) to Stg. reg. (S-35)
Adder (3-17) to add. sw. and add. sw. to stg. reg. (21-35)
Adder (3-17) to index register
Adder (3-17) to index register
Stg. register (3-17) to storage
Storage register (21-35) to storage

Figure 44. SXA and SXD Flow Chart

*Timing:* 2 cycles.

*Execution.* See Figure 44.

### SXD — Store Index in Decrement



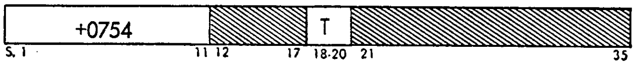
*Description.* The C(Y)$_{3-17}$ are replaced by the contents of the specified index register. The C(Y)$_{S,1,2,18-35}$ are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 44.

### PXA — Place Index in Address



*Description.* The entire accumulator is cleared and the contents of the specified index register are placed in the address part of the AC$_{21-35}$. With a tag of 0 the C(AC) are set to zeros.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 45.

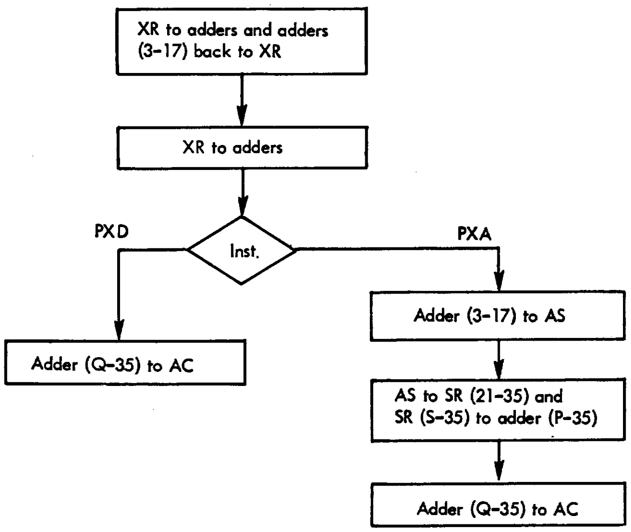
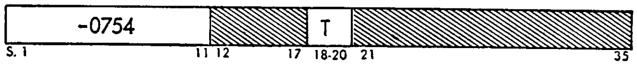

XR to adders and adders (3-17) back to XR
XR to adders
PXD
Inst.
PXA
Adder (3-17) to AS
Adder (Q-35) to AC
AS to SR (21-35) and SR (S-35) to adder (P-35)
Adder (Q-35) to AC

Figure 45. PXD and PXA Flow Chart

### PXD — Place Index in Decrement



*Description.* The entire accumulator is cleared and the contents of the specified index register are placed in the decrement part of the AC$_{3-17}$. With a tag of 0, the C(AC) are set to zeros.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 45.

## Logical Operations

Logical instructions operate on a 36-bit word. The sign position is simply another bit position. The exception to this is when the P position is used instead of the sign position. Logical instructions are frequently used in a process called *masking.* This is the process of extracting one or more small parts of a word from the whole word.

The AND and OR concept 's used with logical operations. When two numbers are combined by an AND, they are matched bit-for-bit. If the same position in each word contains a 1, the result is a 1. If in one word the position is 0 and in the other word it is a 1, the result is a 0. If the same position in both words is a zero, the result is a 0. The following is an example of a logical AND operation:

```
101101011011
101001001101
____________
101001001001 Resulting AND
```

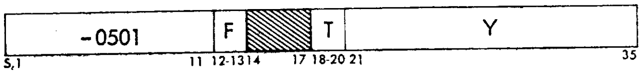
An OR function (sometimes called "inclusive OR") also matches two numbers bit-for-bit. The difference, however, when compared with an AND, is: (1) if the same position in either word contains a 1, the result is a 1; (2) if the same position in both words is a 1, the result is again a 1; (3) only if the same position in both words is a 0, is the resulting position a 0. For example:

```
011010110101
001100100100
____________
011110110101 Resulting inclusive OR
```

One other function of the logical operations is an *exclusive* OR. In this operation, only those positions which do not match result in a 1. If the same position in each word is a zero or if the same position in each word is a 1, the result is a zero. If the same position in one word is a 1 and in the other a 0, then the result is a 1. For example:

```
 101101100101
 001011001101
 100110101000 Resulting exclusive OR
```

## ORA — OR to Accumulator



*Description.* Each bit of the C(Y)$_{S,1-35}$ is matched with the corresponding bit of the C(AC)$_{P,1-35}$. C(Y)$_S$ is matched with C(AC)$_P$.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* When the corresponding bit of either location Y or the AC (or both) is a 1, a 1 replaces the contents of that position in the AC. When the corresponding bits of both location Y and the AC are 0's, a 0 replaces the contents of that position of the AC. The C(Y) and the S and Q positions of the AC are unchanged. See Figure 46.

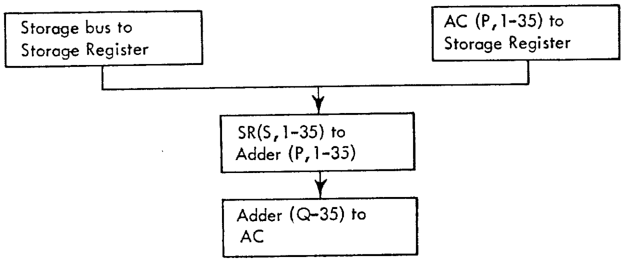
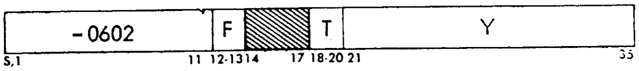

Storage bus to Storage Register
AC (P,1-35) to Storage Register
SR(S,1-35) to Adder (P,1-35)
Adder (Q-35) to AC

Figure 46. ORA Flow Chart

## ORS — OR to Storage



*Description.* Each bit of the C(AC)$_{P,1-35}$ is matched with the corresponding bit of the C(Y)$_{S,1-35}$, C(AC)$_P$ being matched with C(Y)$_S$.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* When the corresponding bit of either the AC or location Y (or both) is a 1, a 1 replaces the contents of that position in location Y. When the corresponding bits of both the AC and location Y are 0's, a 0 replaces the contents of that position in location Y. The C(AC) are unchanged. See Figure 47.

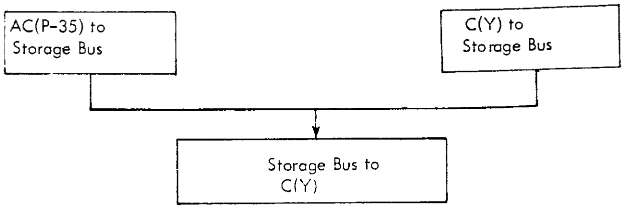
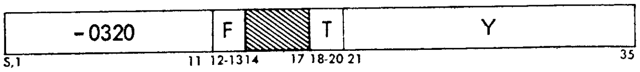


Figure 47. ORS Flow Chart

## ANA — AND to Accumulator



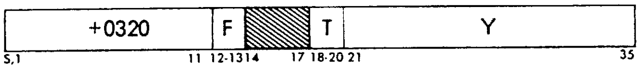
*Description.* Each bit of the C(Y)$_{S,1-35}$ is matched with the corresponding bit of the C(AC)$_{P,1-35}$. C(Y)$_S$ is matched with C(AC)$_P$.

*Indicators.* None.

*Timing:* 3 cycles

*Execution.* When the corresponding bits of both location Y and the AC are 1's, a 1 replaces the contents of that position in the AC. When the corresponding bit of either location Y or the AC, or both, is a 0, a 0 replaces the contents of that position in the AC. The S and Q positions of the AC are cleared. The C(Y) are unchanged. See Figure 48.

## ANS — AND to Storage



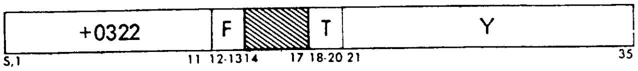
*Description.* Each bit of the C(AC)$_{P,1-35}$ is matched with the corresponding bit of the C(Y)$_{S,1-35}$, C(AC)$_P$ being matched with C(Y)$_S$.

*Indicators.* None.

*Timing:* 4 cycles

*Execution.* When the corresponding bits of both the AC and location Y are 1's, a 1 replaces the contents of that position in location Y. When the corresponding bit of either the AC or location Y, or both, is a 0, a 0 replaces the contents of that position in location Y. The C(AC) are unchanged. See Figure 48.

## ERA — Exclusive OR to Accumulator



*Description.* Each bit of the C(Y)$_{S,1-35}$ is matched with the corresponding bit of the C(AC)$_{P,1-35}$, C(Y)$_S$ being matched with C(AC)$_P$.

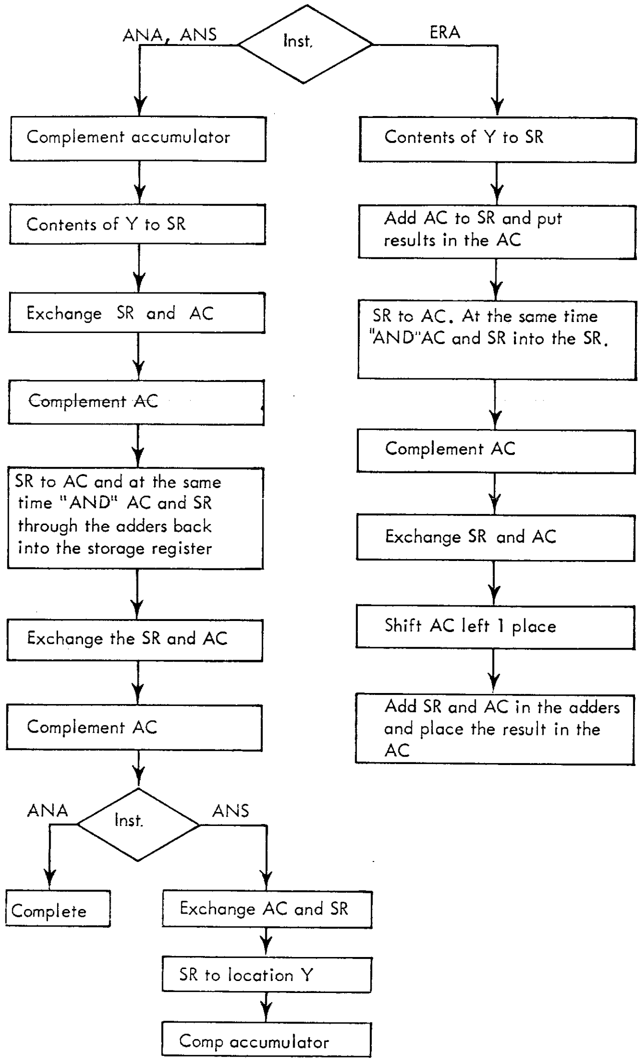


Figure 48. ANA, ANS, and ERA Flow Chart

*Indicators.* None.

*Timing:* 3 cycles

*Execution.* When the corresponding position of the AC matches the position in location Y, a 0 replaces the contents of that position in the AC. When the corresponding position of the AC does not match the position in location Y, a 1 replaces the contents of that position in the AC. Positions S and Q of the AC are cleared. The C(Y) are unchanged. See Figure 48.

THE FOLLOWING logical operations affect the contents of the accumulator only.

### COM — Complement Magnitude

| +0760 | ▨ | T | ▨ | 6 |
|---|---|---|---|---|
| S, 1 | 11 12 | 17 18-20 | 21-23 24 | 35 |

*Description.* All 1's are replaced by 0's and all 0's are replaced by 1's in the $C(AC)_{Q,P,1\text{-}35}$. The $C(AC)_S$ is unchanged. Address modification may change the instruction itself.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 49.

### CLM — Clear Magnitude

| +0760 | ▨ | T | ▨ | 0 |
|---|---|---|---|---|
| S, 1 | 11 12 | 17 18-20 | 21-23 24 | 35 |

*Description* The $C(AC)_{Q,P,1\text{-}35}$ are cleared. The $C(AC)_S$ are unchanged. Address modification by an index register may change the operation itself.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 49.

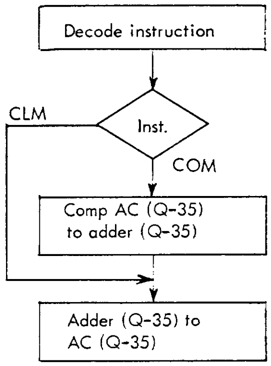


Figure 49. CLM and COM Flow Chart

### CHS — Change Sign

| +0760 | ▨ | T | ▨ | 2 |
|---|---|---|---|---|
| S, 1 | 11 12 | 17 18-20 | 21-23 24 | 35 |

*Description.* If AC sign is plus, it is made negative. If it is negative, it is made plus. Address modification by an index register may change the operation itself. $C(AC)_{Q,P,1\text{-}35}$ are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 50.

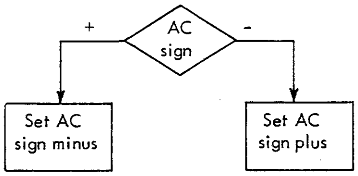
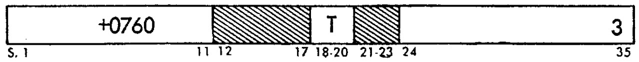


Figure 50. CHS Flow Chart

### SSP — Set Sign Plus



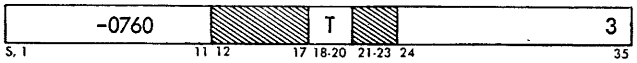
*Description.* The sign of the AC is set to plus (0). Address modification by an index register may result in changing the operation itself.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 51.

### SSM — Set Sign Minus



*Description.* The sign of the AC is set to minus (1). Address modification may result in changing the operation itself.

*Indicators.* None

*Timing:* 2 cycles

*Execution.* See Figure 51.

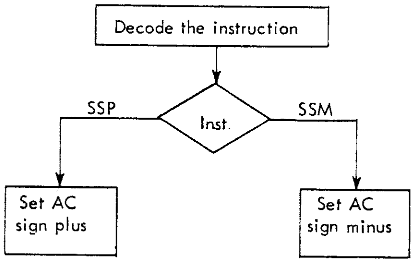


Figure 51. SSM and SSP Flow Chart

## Sense Indicator Operations

The following 24 instructions make reference to the 36-bit sense indicator (SI) register. The 36 bits of the SI may be thought of as switches which may be turned on or off and tested either singly or in groups by the program.

The contents of the SI register are manipulated through the use of a *mask*. The mask is a bit pattern comprised of 1's and 0's which may appear in an instruction, the AC, or any storage location. All masks for SI operations are used in the same way; that is, each position in the mask is compared with the corresponding position in the SI register. For the positions in the mask which contain a 1, the corresponding position in the SI is either modified or tested depending upon the SI operation used. For the zero bits in the mask, the corresponding positions of the SI are not affected.

Four of the sense indicator operations are concerned with the transmission of full 36-bit words between the SI and either the AC or core storage. The remaining 20 operations are used to test or modify the C(SI). These 20 operations may be classified by the following five functions:

1. *Set or Logical OR.* These operations replace with a 1, the contents of each SI position selected by the mask.

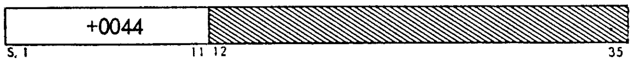
2. *Reset.* These operations replace with a 0 the contents of each SI position selected by the mask.

3. *Invert.* These operations replace the contents of each SI position selected by the mask with its complement; i.e., 1's are replaced by 0's and 0's are replaced by 1's.

4. *On Test.* These operations examine the contents of each SI position selected by the mask. If all examined positions contain a 1, the calculator will either transfer to a location Y or skip the next instruction, depending upon the testing operation used.

5. *Off Test.* These operations examine the contents of each SI position selected by the mask. If all examined positions contain a 0, the calculator either transfers to location Y or skips the next instruction, depending upon the testing operation used.

### PAI — Place Accumulator in Indicators



*Description.* The C(AC)$_{P,1\text{-}35}$ replace the C(SI)$_{0\text{-}35}$. The C(AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* See Figure 52.

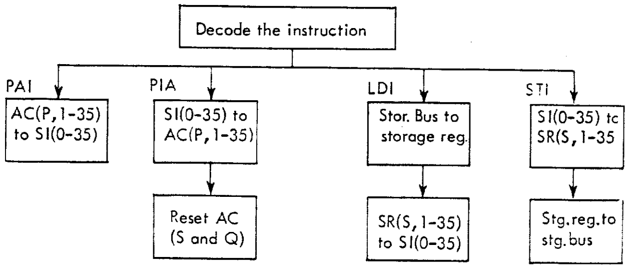
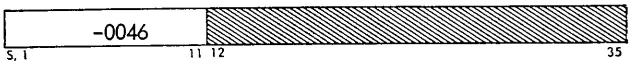


Figure 52. PAI, PIA, LDI, and STI Flow Chart

### PIA — Place Indicators in Accumulator



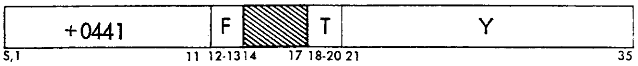
*Description.* The c(SI)$_{0-35}$ replace the c(AC)$_{P,1-35}$. Positions S and Q of the AC are cleared. The c(SI) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* See Figure 52.

### LDI — Load Indicators



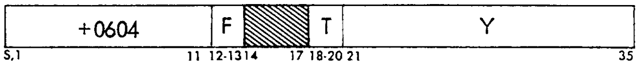
*Description.* The c(Y)$_{S,1-35}$ replace the c(SI)$_{0-35}$. The c(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 52.

### STI — Store Indicators



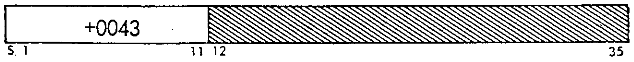
*Description.* The c(SI)$_{0-35}$ replace the c(Y)$_{S,1-35}$. The c(SI) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 52.

### OAI — OR Accumulator to Indicators



*Description.* Each bit of the c(AC)$_{P,1-35}$ is matched with the corresponding bit of the c(SI)$_{0-35}$. The c(AC) are unchanged. When the corresponding bit of either (or both) the AC or SI is a 1, a 1 replaces the contents of that position in the SI. When the corresponding bit of both the AC and SI is a 0, a 0 replaces the contents of that position of the SI.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* See Figure 53.

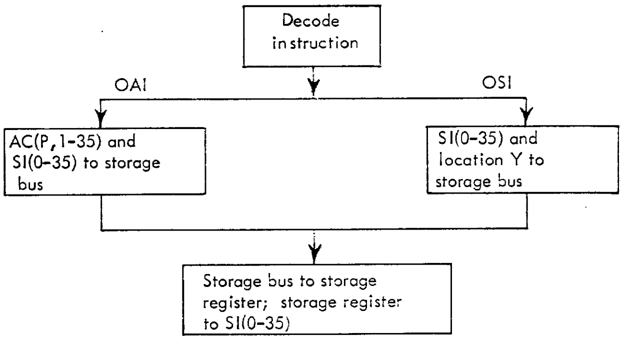
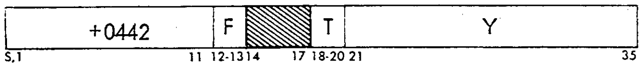

Decode instruction
OAI
OSI
AC(P,1-35) and SI(0-35) to storage bus
SI(0-35) and location Y to storage bus
Storage bus to storage register; storage register to SI(0-35)

Figure 53. OAI and OSI Flow Chart

### OSI — OR Storage to Indicators



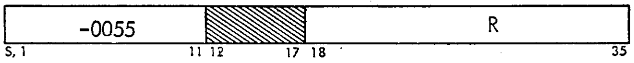
*Description.* Each bit of the c(Y)$_{S,1-35}$ is matched with the corresponding bit of the c(SI)$_{0-35}$. The c(Y) are unchanged. When the corresponding bit of either location Y or SI (or both) is a 1, a 1 replaces the contents of that position of the SI. When the corresponding bit of both Y and SI is a 0, a 0 replaces the contents of that position in the SI.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* See Figure 53.

### SIL — Set Indicators of Left Half



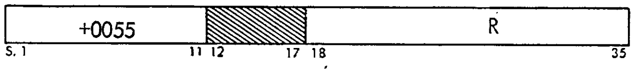
*Description.* Each bit in positions 18-35 (R) of this instruction is matched with the corresponding bit of the c(SI)$_{0-17}$. The c(SI)$_{18-35}$ and R are unchanged. When the corresponding bit of either (or both) R or the SI is a 1, a 1 replaces the contents of that position in the SI. When the corresponding bit of both the R and the SI is a 0, a 0 replaces the contents of that position in the SI.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* See Figure 54.

### SIR — Set Indicators of Right Half



*Description.* Each bit in positions 18-35 (R) of this instruction is matched with the corresponding bit of

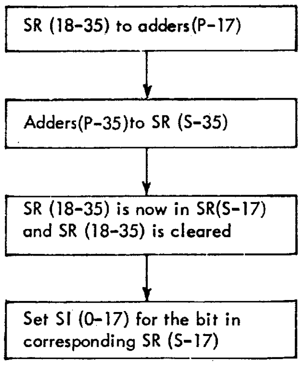

SR (18-35) to adders(P-17)
Adders(P-35)to SR (S-35)
SR (18-35) is now in SR(S-17) and SR (18-35) is cleared
Set SI (0-17) for the bit in corresponding SR (S-17)

Figure 54. SIL Flow Chart

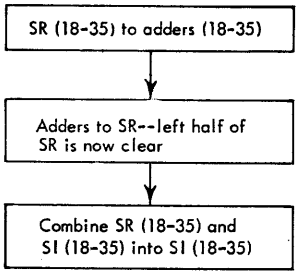

SR (18-35) to adders (18-35)
Adders to SR--left half of SR is now clear
Combine SR (18-35) and SI (18-35) into SI (18-35)

Figure 55. SIR Flow Chart

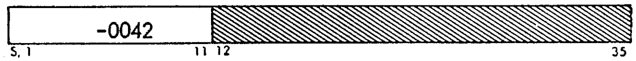
the C(SI)$_{18-35}$. The C(SI)$_{0-17}$ and R are unchanged. When the corresponding bit of either (or both) R or the SI is a 1, a 1 replaces the contents of that position in the SI. When the corresponding bit of both the R and SI is a 0, a 0 replaces the contents of that position in the SI.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* See Figure 55.

## RIA — Reset Indicators from Accumulator



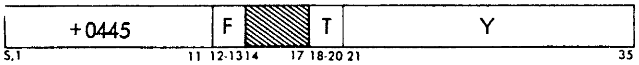
*Description.* Each bit of the C(AC)$_{P,1-35}$ resets to 0 the corresponding bit of the C(SI)$_{0-35}$. The C(AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* When the bit in the AC is a 1, a 0 replaces the contents of that position in the SI. When the bit in the AC is a 0, the contents of that position in the SI are unchanged. This is accomplished by taking the contents of the accumulator, bit for bit, and feeding it into a reset input of the sense register. This input will accept only a "1" pulse which turns that position off (0 condition).

## RIS — Reset Indicators from Storage



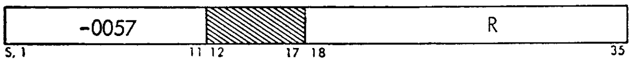
*Description.* Each bit of the C(Y)$_{S,1-35}$ resets to 0 the corresponding bit of the C(SI)$_{0-35}$. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* When the bit in location Y is a 1, a 0 replaces the contents of that position in the SI. When the bit in location Y is a 0, the contents of that position in the SI are unchanged. The operation is identical to that of RIA except that the contents of a storage location, instead of the accumulator, are used to reset the indicators.

## RIL — Reset Indicators of Left Half



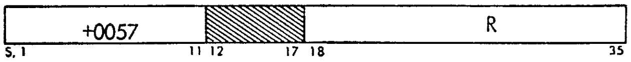
*Description.* Each bit in positions 18-35 (R) of this instruction resets to 0 the corresponding bit of the C(SI)$_{0-17}$. The C(SI)$_{18-35}$ and R are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* When the bit in R is a 1, a 0 replaces the contents of that position in the SI. When the bit in R is a 0, the contents of that position in the SI are unchanged. The operation is identical to that of RIA except that positions 18-35 of the RIL instruction are used to reset positions 0-17 of the sense indicators.

## RIR — Reset Indicators of Right Half



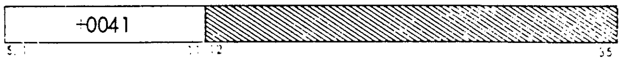
*Description.* Each bit in positions 18-35 (R) of this instruction resets to 0 the corresponding bit of the C(SI)$_{18-35}$. The C(SI)$_{0-17}$ and R are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* When the bit in R is a 1, a 0 replaces the contents of that position in the SI..When the bit in R is a 0, the contents of that position in the SI are unchanged. The operation is identical to that of RIA except that positions 18-35 of the RIR instruction are used to reset positions 18-35 of the sense indicators.

## IIA — Invert Indicators from Accumulator



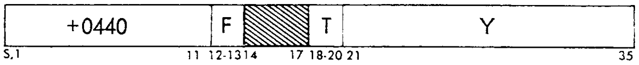
*Description.* Each bit of the C(AC)$_{P,1-35}$ is matched with the corresponding bit of the C(SI)$_{0-35}$. When the bit in the AC is a 1, the contents of that position in the SI are complemented. When the bit in the AC is a 0, the contents of that position in the SI are unchanged. The C(AC) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* Sense indicator positions may have "binary input." When this input is used, a "1" pulse fed to the position will reverse its status. For example, if the position holds a 0 and a 1 is fed to it, the position will reverse to a 1 status. Likewise, if the position holds a 1 and a 1 is fed to it, it will flip to a zero status. Zeros fed to the binary input do not affect the position. For the IIA instruction, the C(AC)$_{P,1-35}$ are fed to the binary inputs of the SI(0-35).

### IIS — Invert Indicators from Storage



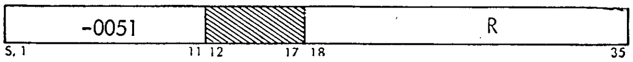
*Description.* Each bit of the C(Y)$_{S,1-35}$ is matched with the corresponding bit in the C(SI)$_{0-35}$. When the bit in the location Y is a 1, the contents of that position in the SI are complemented. When the bit in location Y is a zero, the contents of that position in the SI are unchanged. The C(Y) are unchanged.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* The same procedure as IIA except that the contents of storage location Y, instead of the contents of the accumulator, are used to reset the sense indicators.

### IIL — Invert Indicators of Left Half



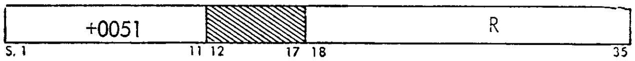
*Description.* Each bit of positions 18-35 (R) of this instruction is matched with the corresponding bit of the C(SI)$_{0-17}$. The C(SI)$_{18-35}$ and R are unchanged. When the bit in R is a 1, the contents of that position in the SI are complemented. If the bit in R is a zero, the contents of that position in the SI are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* The same as IIA except that 18-35 of this instruction is used in resetting positions 0-17 of the sense indicators.

### IIR — Invert Indicators of Right Half



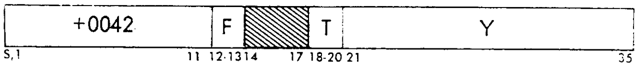
*Description.* Each bit of positions 18-35 (R) of this instruction is matched with the corresponding bit of the C(SI)$_{18-35}$. The C(SI)$_{0-17}$ and R are unchanged. When the bit in R is a 1, the contents of that position in the SI are complemented. When the bit in R is a zero, the contents of that position in the SI are unchanged.

*Indicators.* None.

*Timing:* 2 cycles.

*Execution.* The same as IIA except that 18-35 of the IIR are used to reset positions 18-35 of the sense indicators.

### TIO — Transfer when Indicators On



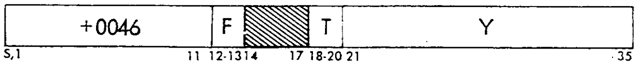
*Description.* For each bit in the C(AC)$_{P,1-35}$ that is a 1, the corresponding position of the C(SI)$_{0-35}$ is examined. If all the examined positions in the SI contain a 1, the computer takes its next instruction from location Y. If any of the examined positions is not a 1, the computer takes the next sequential instruction. The C(AC) and C(SI) are unchanged.

*Indicators.* Trap.

*Timing:* 2 cycles

*Execution.* See Figure 56.

### TIF — Transfer when Indicators Off



*Description.* For each bit of the C(AC)$_{P,1-35}$ that is a 1, the corresponding position of the C(SI)$_{0-35}$ is examined. If all examined positions contain 0's, the computer takes its next instruction from location Y. If any of the examined positions does not contain a 0, the computer takes the next sequential instruction. The C(AC) and C(SI) are unchanged.

*Indicators.* Trap.

*Timing:* 2 cycles

*Execution.* See Figure 56.

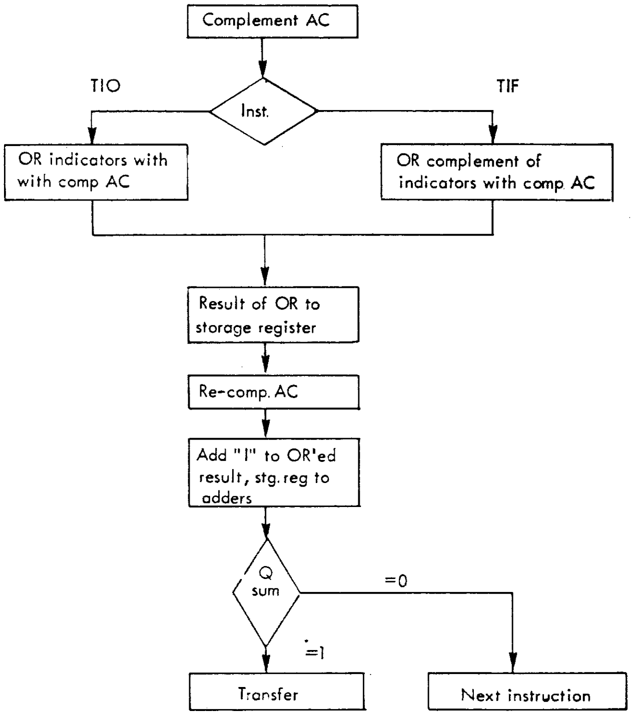
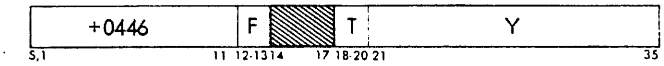

Complement AC
TIO
TIF
Inst.
OR indicators with with comp AC
OR complement of indicators with comp AC
Result of OR to storage register
Re-comp. AC
Add "I" to OR'ed result, stg. reg to adders
Q sum
=0
=1
Transfer
Next instruction

Figure 56. TIO and TIF Flow Chart

## ONT — On Test for Indicators



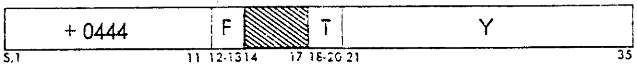
*Description.* For each bit in the C(Y)$_{S,1-35}$ that is a 1, the corresponding position of the C(SI)$_{0-35}$ is examined. If all the examined positions in the SI contain a 1, the computer skips the next instruction and proceeds from there. If any of the examined positions do not contain 1's, the computer takes the next sequential instruction. The C(Y) and C(SI) are unchanged.

*Indicators.* None.

*Timing:* 4 cycles

*Execution.* See Figure 57.

## OFT — Off Test for Indicators



*Description.* For each bit of the C(Y)$_{S,1-35}$ that is a 1, the corresponding position of the C(SI)$_{0-35}$ is examined. If all the examined positions contain 0's, the computer skips the next instruction and proceeds from there. If any of the examined positions does not contain a 0, the computer takes the next sequential instruction. The C(Y) and C(SI) are unchanged.

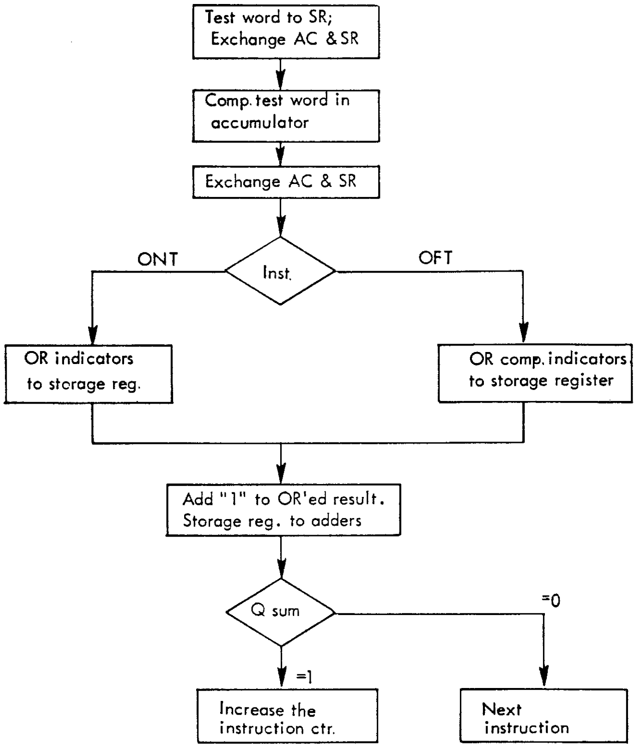

Test word to SR; Exchange AC & SR
Comp. test word in accumulator
Exchange AC & SR
ONT
Inst.
OFT
OR indicators to storage reg.
OR comp. indicators to storage register
Add "1" to OR'ed result. Storage reg. to adders
Q sum
=0
=1
Increase the instruction ctr.
Next instruction

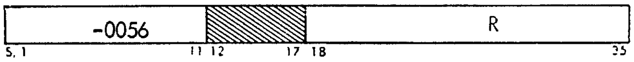
Figure 57. ONT and OFT Flow Chart

*Indicators.* None.

*Timing:* 4 cycles

*Execution.* See Figure 57.

## LNT — Left Half Indicators on Test

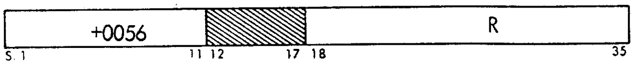


*Description.* For each bit in positions 18-35 (R) of this instruction that is a one, the corresponding position of the C(SI)$_{0-17}$ is examined. The C(SI) and R are unchanged. If all the examined positions contain a 1, the computer skips the next instruction and proceeds from there. If any of the examined positions does not contain a 1, the computer takes the next sequential instruction.

*Indicators.* None.

*Timing:* 4 cycles

## RNT — Right Half Indicators on Test

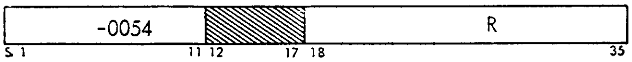


*Description.* For each bit in positions 18-35 (R) of this instruction that is a 1, the corresponding position of the C(SI)$_{18\text{-}35}$ is examined. The C(SI) and R are unchanged. If all the examined positions contain a 1, the computer skips the next instruction and proceeds from there. If any of the examined positions does not contain a 1, the computer takes the next sequential instruction.

*Indicators.* None.

*Timing:* 4 cycles

### LFT — Left Half Indicators Off Test

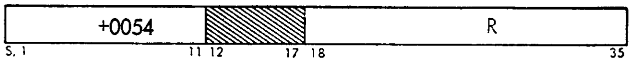


*Description.* For each bit in positions 18-35 (R) of this instruction that is a 1, the corresponding position of the C(SI)$_{0\text{-}17}$ is examined. The C(SI) and R are unchanged. If all the examined positions contain a 0, the computer skips the next instruction and proceeds from there. If any of the examined positions does not contain a zero, the computer takes the next sequential instruction.

*Indicators.* None.

*Timing:* 4 cycles

### RFT — Right Half Indicators Off Test



*Description.* For each bit in positions 18-35 (R) of this instruction that is a one, the corresponding position of C(SI)$_{18\text{-}35}$ is examined. If all the examined positions contain a zero, the computer skips the next instruction and proceeds from there. If any of the examined positions does not contain a zero, the computer takes the next sequential instruction. The C(SI) and R are unchanged.

*Indicators.* None.

*Timing:* 4 cycles

## Convert Instructions

The convert operations enable a program to have very rapid access to information stored in tables in core storage. A single convert instruction can perform a series of table look-up operations by making multiple references to core storage. Three such convert operations are available in the computer.

To illustrate the method of execution of these convert instructions, the following section describes the CONVERT BY REPLACEMENT FROM THE MQ (CRQ) instruction. The following two definitions will apply throughout this description:

*Argument*—The known reference factor necessary to find a desired item in a table.

*Function*—The unknown factor in a table associated with a known reference factor (argument).

The contents of the MQ are interpreted as six 6-bit quantities. Each of these quantities may be considered as a 6-bit binary integer and will be designated as L1, L2, ........, L6 (Figure 58).

MQ Register

| L1 | L2 | L3 | L4 | L5 | L6 |
|---|---|---|---|---|---|
| S 5 | 6 11 | 12 17 | 18 23 | 24 29 | 30 35 |

Figure 58. MQ Register

The number, Y1, contained in the address part of the CRQ instruction is interpreted by the instruction as the *address* of the first location (origin) of a table in core storage. The format of a typical table is shown in Figure 59. The 6-bit binary integer, L1, is taken as the first argument and the address Y1 + L1 is formed. The instruction then looks up the word located at Y1 + L1. The left-most six bits of this word, positions S, 1-5, are taken as the desired function V1. This 6-bit number, V1, then replaces the number L1 in the MQ. The right-most 15 bits of this word, positions 21-35, are interpreted as an address, Y2, specifying the origin of a second table in core storage. The number L2 is then taken as the second argument and a reference to the second table is made at location Y2 + L2. The contents of Y2 + L2, positions S, 1-5, make up the second function, V2, and replace L2 in the MQ. Positions 21-35 of this word are interpreted as a third address, Y3, the origin of a third table in core storage, and the process continues.

The function V1 replaces L1 in the MQ through a shifting operation. The MQ is shifted left six positions and the bits of L1 are shifted out of position S of the

| Location | Contents | | |
|---|---|---|---|
| Y1 | V0 | | Y2 |
| .<br>. | .<br>. | | .<br>. |
| Y1 + L1 | V1 | | Y2 |
| .<br>. | .<br>. | | .<br>. |
| Y1 + N | VM | | Y2 |
| | S 5 | 6 20 | 21 35 |

Figure 59. Convert Table

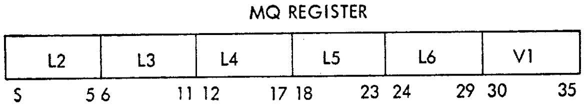


Figure 60. MQ Register

MQ and are lost. The number V1 then replaces the contents of the MQ, positions 30-35 (Figure 60).

Thus, the CRQ instruction provides for replacement of the contents of the MQ from left to right.

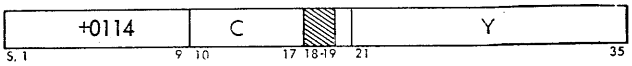
The number of such table references made by a single convert instruction is specified by the count field, positions 10-17, of the instruction. If a count of six is given, six numbers V1, V2, ........, V6 will occupy the exact MQ positions originally containing the six corresponding arguments L1, L2, ........, L6. If a count of one was specified for a CRQ instruction, the final contents of the MQ would be as shown in Figure 60. When a count of more than six is specified, the values taken from the tables during the first six references will be used for additional table references. For example, V1 = L7, V2 = L8, and so on.

After the last function, Vn, has been placed in the MQ, the location in core storage from which Vn has been taken contains as its address part a number, Yn. If the tag field of the convert instruction contains a one, the number Yn replaces the contents of index register 1. This provides a convenient method for the program to determine where the last storage table reference has been made or where the next table reference is to be made. (Only index register 1 can be used.)

The CONVERT BY REPLACEMENT FROM THE AC (CVR) instruction is analogous to the CRQ instruction. For this instruction the $C(AC)_{P,1\text{-}35}$ rather than the C(MQ) are interpreted as the 6-bit numbers L1, L2, ......, L6. Also, the six-place shifts which occur during the execution of the instruction are right shifts rather than left shifts. Thus, for the CVR instruction, the replacement takes place right to left, whereas, for the CRQ, the replacement takes place from left to right.

The CAQ instruction interprets the C(MQ) as six 6-bit quantities L1, L2, ......, L6 and uses these numbers as arguments in the same manner as the CRQ instruction. However, instead of replacing the numbers L1, L2, ......, L6, the contents of the looked-up words are added into the AC. The address parts of these words are then used as origins for additional look-ups in the usual manner. Addition into the AC is logical, with the S position of the word being added into the P position of the AC. After an argument has been used for a table reference, the C(MQ) are rotated left so that bits leaving position S enter position 35 of the MQ. Thus, if a count of six is specified for a CAQ instruction, the final and original C(MQ) are identical.

## CVR — Convert by Replacement from the AC



*Description.* This instruction treats the $C(AC)_{P,1\text{-}35}$ as six 6-bit quantities and replaces the first C of these quantities by values from tables in core storage. Position S of the AC is unchanged. NOTE: Bit position Q is not cleared and will OR with the first function, if present initially.

*Indicators.* None.

*Timing:* 2-8 cycles, modification 6

*Execution.* The instruction is executed in the following steps:

1. The address part (Y) replaces the $C(SR)_{21\text{-}35}$.

2. The count field (C) is placed in the shift register.

3. The contents of the shift register are tested. If the register contains zero, step 4a follows. If the register is non-zero, step 4b follows.

4a. If position 20 of this instruction contains a 1, the $C(SR)_{21\text{-}35}$ replace the contents of index register 1 (XRA), and the computer takes the next sequential instruction. If position 20 contains a 0, the computer proceeds directly to the next sequential instruction.

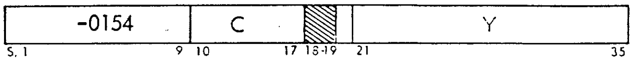
4b. The $C(SR)_{21\text{-}35}$ are added to the $C(AC)_{30\text{-}35}$ to form an address (X). The C(X) replace the C(SR).

5. The $C(AC)_{Q,P,1\text{-}35}$ are shifted right six places. Positions vacated are filled with zeros.

6. The $C(SR)_{S,1\text{-}5}$ replace the $C(AC)_{P,1\text{-}5}$. However, if position Q of the AC initially contains a 1, it will be shifted to position 5 of the AC during step 5. This 1 in position 5 of the AC will remain regardless of the contents of position 5 of the SR.

7. The contents of the shift register are decreased by one and the computer returns to step 3.

## CRQ — Convert by Replacement from the MQ



*Description.* This instruction treats the C(MQ) as six 6-bit quantities and replaces the first C of these quantities by values from tables in core storage.

*Indicators.* None.

*Timing:* 2-8 cycles, modification 6

*Execution.* The instruction is executed in the following steps:

1. The address part (Y) replaces the c(SR)$_{21-35}$.

2. The count field (C) is placed in the shift register.

3. The contents of the shift register are tested. If the register contains 0, step 4a follows. If the register is not 0, step 4b follows.

4a. If position 20 of this instruction contains a 1, the c(SR)$_{21-35}$ replace the contents of XRA and the computer proceeds to the next sequential instruction. If position 20 contains a 0, the computer proceeds directly to the next sequential instruction.

~~4b. The c(SR)$_{21-35}$ are added to the c(MQ)$_{S,1-5}$ to form~~ an address (X). The c(x) then replace the c(SR).

5. The c(MQ) are shifted left six places. Bits shifted out of position S of the MQ are lost. Positions vacated are filled with zeros.

6. The c(SR)$_{S,1-5}$ replace the c(MQ)$_{30-35}$.

7. The contents of the shift register are decreased by one, and the computer returns to step 3.

### CAQ — Convert by Addition from the MQ

| -0114 | C | ▨ | | Y |
|---|---|---|---|---|
| S, 1 9 | 10 17 | 18-19 | | 21 35 |

*Description.* This instruction treats the c(MQ) as six 6-bit quantities. The first C of these quantities are used in making references to tables in core storage. Words selected by the references are added to the c(AC)$_{Q,P,1-35}$. Position S of the AC is unchanged. NOTE: Care should be taken that the binary sum of the quantities added from 21-35 does not carry into position 19 of the AC.

*Indicators.* None.

*Timing:* 2-8 cycles, modification 6

*Execution.* The instruction is executed in the following steps:

1. The address part (Y) replaces the c(SR)$_{21-35}$.

2. The count field (C) is placed in the shift register.

3. The contents of the shift register are tested. If the register contains 0, step 4a follows. If the register is not 0, step 4b follows.

4a. If position 20 of this instruction contains a 1, the c(SR)$_{21-35}$ replace the contents of XRA, and the computer proceeds to the next sequential instruction. If position 20 contains a 0, the computer proceeds directly to the next sequential instruction.

4b. The c(SR)$_{21-35}$ are added to the c(MQ)$_{S,1-5}$ to form an address (X). The c(x) then replace the c(SR).

5. The c(MQ) are rotated six positions to the left. Bits leaving position S of the MQ enter position 35 of the MQ.

6. The c(SR)$_{S,1-35}$ are added to the c(AC)$_{Q,P,1-35}$ and the sum replaces the c(AC)$_{Q,P,1-35}$. The sign position of the SR is added into position P of the AC, and the sign of the AC is disregarded. NOTE: Even though AC overflow is possible, the overflow indicator is not affected by this instruction.

7. The contents of the shift register are reduced by one and the computer returns to step 3.

The reader is referred to the CONVERT programming examples contained in the programming section of this manual for possible uses of these instructions.

## Input-Output Operations

As has been previously stated, the address part of an instruction may refer either to a location in core storage, the length of shift, or may be interpreted as a part of the operation code itself.

The identifying number for the various input-output units appears in the address part of the instruction. For tapes, card machines and printers the address part specifies both the particular I-O unit and the data channel to which it is attached. Channels A through H are specified by the numbers 1 through 10, respectively, appearing as the first two digits of an octal five-digit address. The last three digits specify the I-O unit. If the I-O unit is a tape, the mode of operation, either binary or BCD, is also specified by the address.

The addresses of the input-output devices are shown below:

| DEVICE | CHANNEL | OCTAL ADDRESS |
|---|---|---|
| Tape Units (BCD) | A | 1201-1212 |
| | B | 2201-2212 |
| | C | 3201-3212 |
| | D | 4201-4212 |
| | E | 5201-5212 |
| | F | 6201-6212 |
| Tape Units (binary) | A | 1221-1232 |
| | B | 2221-2232 |
| | C | 3221-3232 |
| | D | 4221-4232 |
| | E | 5221-5232 |
| | F | 6221-6232 |
| Card Reader | A | 1321 |
| | C | 3321 |
| | E | 5321 |

| DEVICE | CHANNEL | OCTAL ADDRESS |
|---|---|---|
| Card Punch | A | 1341 |
| | C | 3341 |
| | E | 5341 |
| Printer (normal) | A | 1361 |
| | C | 3361 |
| | E | 5361 |
| Printer (binary) | A | 1362 |
| | C | 3362 |
| | E | 5362 |
| Drum | None | 0301-0310 |
| CRT | None | 0030 |

In the following instruction description, only the operation code will be shown. The address part will appear in the normal code of Y. All input-output instructions may be trapped using the compatibility feature.

Since it is possible to create magnetic tapes of mixed density, precautions should be taken so that all read tape instructions are executed when the tape unit is set to the density in which the tape was recorded.

### RDS — Read Select

| +0762 | ▨ | T | Y |
|---|---|---|---|
| S 1 11 | 12 17 | 18-20 | 21 35 |

*Description.* This instruction causes the computer to prepare to read information, from the I-O device specified by Y, into core storage. If Y specifies a tape, printer, or card reader, Y also specifies the channel to which the device is attached.

*Indicators.* Simulate, end of file. (See device being used.)

*Timing:* 2 cycles, modification 8

*Execution.* When a channel is designated, a RESET AND LOAD CHANNEL instruction must be given within the specified time following the RDS, or the I-O device will be logically disconnected from the computer and one record will be passed. (See each device for timings.) When an RDS specifies channel operation, only positions 27-35 of the address part of the instruction are subject to effective address modification.

### WRS — Write Select

| +0766 | ▨ | T | Y |
|---|---|---|---|
| S, 1 11 | 12 17 | 18-20 | 21 35 |

*Description.* This instruction causes the computer to prepare to write information from storage to the I-O device specified by Y. If Y specifies a tape, printer or card punch, Y also specifies the channel to which the device is attached.

*Indicators.* Simulate, end of tape. (See device being used.)

*Timing:* 2 cycles, modification 8

*Execution.* When a channel is designated, a RESET AND LOAD CHANNEL instruction must be given within specified time following the WRS, or the I-O device will be logically disconnected from the computer and, if the WRS specified a tape, a blank section of tape will be written. If the end of tape reflective spot is encountered during the execution of a WRS, the end-of-tape indicator in the proper channel will be turned on. When a WRS specifies an I-O device attached to a channel, only positions 27-35 of the address part of the instruction are subject to effective address modification.

### BSR — Backspace Record

| +0764 | ▨ | T | Y |
|---|---|---|---|
| S, 1 11 | 12 17 | 18-20 | 21 35 |

*Description.* This instruction causes the tape designated by Y to move backward until recorded information is reached and then to continue this backward motion over information until an end-of-record gap or load point is encountered.

*Indicators.* Beginning of tape and simulate.

*Timing:* 2 cycles, modification 8

*Execution.* If the tape designated by Y is positioned at its load point, a BSR is interpreted as a no-operation and the beginning-of-tape indicator in the proper channel is turned on. If load point is encountered before information is encountered, the BOT indicator is turned on. Only positions 27-35 of the address part of this instruction are subject to effective address modification.

### BSF — Backspace File

| -0764 | ▨ | T | Y |
|---|---|---|---|
| S, 1 11 | 12 17 | 18-20 | 21 35 |

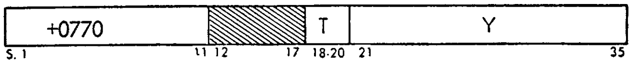
*Description.* This instruction causes the tape designated by Y to move backward until recorded information is reached and then to continue this backward motion over information and end-of-record gaps until an end-of-file record or the load point is encountered.

*Indicators.* Beginning of tape, simulate.

*Timing:* 2 cycles, modification 8.

*Execution.* If a BSF is given to a tape positioned at its load point, the BSF is interpreted as a no-operation and the beginning-of-tape indicator in the proper channel is turned on. If load point is encountered before an end-of-file record, the beginning-of-tape indicator is turned on. Only positions 27-35 of the address part of this instruction are subject to effective address modification.

### WEF — Write End-of-File



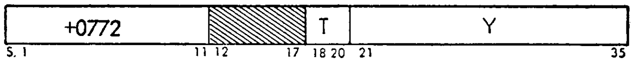
*Description.* This instruction causes the tape designated by Y to write an end-of-file gap followed by a tape mark (and its check character) on the tape.

*Indicators.* End of tape, simulate.

*Timing:* 2 cycles, modification 8.

*Execution.* If an end of tape reflective spot is passed over during the execution of a WEF, the end-of-tape indicator in the proper channel is turned on. Only positions 27-35 of the address part of this instruction are subject to effective address modification.

### REW — Rewind



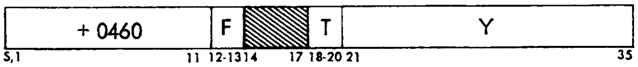
*Description.* This instruction causes the tape designated by Y to rewind its tape to the load point position.

*Indicators.* Simulate.

*Timing:* 2 cycles, modification 8.

*Execution.* If the tape is positioned at its load point at the time the REW is interpreted, the instruction is treated as a no-operation. Only positions 27-35 of the address part of this instruction are subject to effective address modification.

### LDA — Locate Drum Address



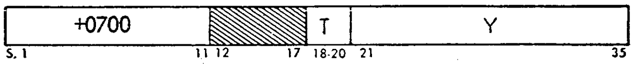
*Description.* This instruction follows an RDS or WRS instruction which refers to a magnetic drum, but before the first CPY instruction is given. The address Y specifies a core storage location from which positions 21-35 will be used to denote the first drum location to be involved in the transmission.

*Indicators.* I-O check, simulate.

*Timing:* 3 cycles, modification 8.

*Execution.* The first CPY following an LDA will read from or write into the drum location found by the LDA. Subsequent CPY's will denote the storage locations to be used by succeeding drum locations. If no LDA is given following an RDS or WRS, it is equivalent to giving an LDA with an effective address of zero. This operation clears the MQ. If an LDA is given without a prior RDS or WRS, the LDA is treated as a no-operation and the I-O check indicator is turned on.

### CPY — Copy



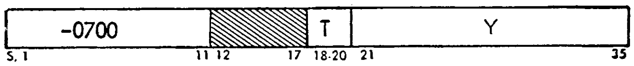
*Description.* This instruction follows an RDS or WRS referring to either a drum or CRT device, or following an LDA or another CPY. If the initial select was a WRS, a word will be transmitted from storage location Y to the selected device. If the initial select was an RDS, a word will be read from the device to be stored in location Y.

*Indicators.* I-O check, simulate.

*Timing:* 3 cycles, modification 8.

*Execution.* Whether reading or writing, the MQ is used as a buffer between storage and the I-O device. After a drum has been selected and the first CPY or an LDA has been issued, succeeding CPY's must be given within 36 microseconds. A CPY given after this time interval will cause the I-O check indicator to be turned on and the CPY will be executed as a no-operation. If a CPY is given without a prior RDS or WRS, the CPY is treated as a no-operation and the I-O check indicator is turned on.

### CAD — Copy and Add Logical



*Description.* This instruction executes a CPY followed by an ACL.

*Indicators.* I-O check, simulate.

*Timing:* 2 cycles.

*Execution.* See CPY and ACL descriptions. Refer to "Compatibility" section.

## Data Channel Commands

The eight types of data channel commands are described in much the same manner as other computer instructions. The following steps list command conditions:

1. The letter Y in the address part (21-35) of a command is used to denote a core storage location.

2. The letter C in the decrement part (3-17) of a command is used to denote a word count amount.

3. The numerical operation code is shown by an octal digit in the prefix part (S, 1 and 2). The digit may be visually converted to its binary equivalent for reference to the bit pattern actually used.

4. Indirect addressing of data channel commands is optional on the 709 system. Position 18 of the command contains the flag bit. Thus, with a command having an address part (Y) and a one in position 18, the address part of location Y replaces the address part of the command before it is executed. With a zero word count, indirect addressing does not occur on IOCP or IOSP commands. Indirect addressing will occur only on read or write operations.

5. Separate commands have not been formulated to handle bit position 19. Instead, a fifth character (N—denoting non-transmit) is appended to the mnemonic codes used for positions S, 1, and 2. This type of command is used for read operations only.

6. Seven of the eight commands deal with data transmission. The codes for these commands all contain the letters "I-O".

7. The eighth command is a transfer in channel command.

8. The word disconnected means that the units involved are separated logically rather than physically. When a disconnect is signalled, it may be delayed depending on the operation being performed. For example:

*a. Write Tape.* Disconnect will not occur until the end of record gap is written. This permits a programmed ETT or TRC test to be valid if given after the channel leaves operation (disconnects).

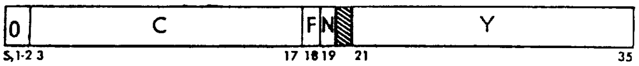
*b. Read Tape.* Disconnect will not occur until tape control circuits and tape units have accepted the read instruction. This assures that any read select instruction will move tape before the disconnect occurs even if an immediate disconnect is programmed. Disconnect will not occur between the last word of the record and the interrogation of the longitudinal redundancy check character (LRC). This assures that with a channel programmed to read an entire record, a TRC test, given after the channel leaves operation, will have tested the LRC character.

Recognition of an end of file, while reading tape or cards, causes a disconnect regardless of the command being used. The tape unit does not leave operation until the LRC character has been checked and the end of record reached. An IORP, IORT, IOSP, or IOST command will not recognize a logical end of record on an end of file.

*c. Card Machines.* Unless a disconnect is programmed immediately following a transmission or end of record time, the disconnect will generally be delayed until the next transmission or end of record time. For example, if the channel is loaded with an IOCD command (with zero word count) an extra machine cycle will occur ten milliseconds after end of record time. The disconnect then occurs at the 9-left transmission time of this extra cycle.

IN EXECUTION descriptions of the I-O commands, tape is assumed to be the I-O device. However, mechanical motion on the tape (from record to record) may be compared to card equipment (from card to card or line to line) on the printer. The descriptions may then be applied to I-O devices other than tape.

### IOCD — Input-Output under Count Control and Disconnect



*Description.* C words are transmitted between an I-O device and core storage beginning with location Y. The data transmission is under control of the count (C) field only.

*Indicators.* See I-O device being used.

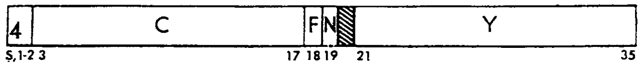
*Timing:* See I-O device being used.

*Execution*

*Read Operation.* If the word count has been reduced to zero before a record gap is reached, the channel leaves operation and tape motion will continue until a record gap is reached. No transmission will occur during this time. If a record gap is encountered when the word count is not zero, the gap is ignored and reading continues from the next record. An IOCD command with a zero word count loaded at the end of record will disconnect the channel.

*Write Operation.* When C words have been written on tape, a record gap is written. If this command is given at the beginning of a record and C is initially zero, approximately eight inches of blank tape will be written before the record gap is written. The channel is then disconnected unless restricted by the operation. (See part 8 of command conditions.)

### IOCP — Input-Output under Count Control and Proceed



*Description.* C words are transmitted between an I-O device and core storage location Y. The data transmission is under control of the count field only. When C is reduced to zero, or is initially zero, the next sequential command is brought into the data channel and executed.

*Indicators.* See I-O device being used.

*Timing.* See I-O device being used.

*Execution*

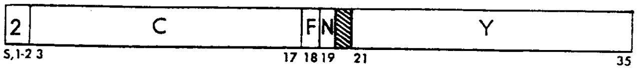
*Read Operation.* C words are read from tape and stored in consecutive storage locations beginning with location Y. When the word count has been reduced to zero, the channel takes its next command in sequence and executes it.

If the word count is reduced to zero by the last word of a record and the next command is an IORP, IOSP, IORT, or IOST, the present end of record will be recognized. Unlike the IOSP, the IOCP command need not proceed within 11 cycles in order to recognize the present end of record.

*Write Operation.* C words from storage beginning with location Y are written on tape. When the specified words have been written, the channel proceeds to the next sequential command. An end of record is not written on tape when the word count is reduced to zero.

In printing or the punching of cards, if the word count is reduced to zero on the 12-row right transmission point, and the next command is an IORP or IORT, the end of the present machine cycle (record) will be recognized.

### IORP — Input-Output of a Record and Proceed



*Description.* See "Execution."

*Indicators.* See I-O device being used.

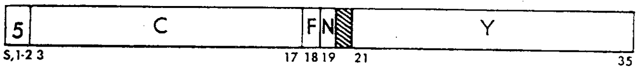
*Timing:* See I-O device being used.

*Execution*

*Read Operation.* Words are transmitted from tape and stored in consecutive storage locations until either an end of record is encountered or the word count has been reduced to zero. If the word count is reduced to zero (or is initially zero) before an end of record is reached, the rest of the words in that record are skipped without transmission to storage. When the record gap is reached, the channel takes the next sequential command.

*Write Operation.* When C words have been written, a record gap is written and the channel proceeds to the next sequential command.

### IOCT — Input-Output under Count Control and Transfer



*Description.* C words are transmitted between an I-O device and storage beginning with location Y. The data transmission is under control of the count field only. When C is reduced to zero, the next command is taken from a core location specified by the load channel instruction in the main program or disconnects (and traps if the channel is enabled) if no load channel instruction is waiting. If this command is loaded by an RCH or LCH instruction and C is initially zero, the channel is immediately disconnected unless restricted by the operation. If command trap is enabled, the trap occurs when recognized even if the disconnect is restricted by the operation.

*Indicators.* Command Trap.

*Timing:* See I-O device being used.

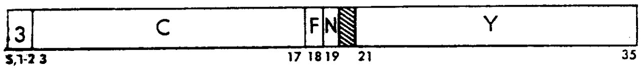
*Execution*

*Read Operation.* Execution of the command is under control of the count field only and end of record gaps are ignored. If the word count is reduced to zero by the last word of a record and the next command is an IORP, IOSP, IORT, or IOST, the present end of record will be recognized. Unlike the IOST command, the LCH instruction need not load the channel within 11 cycles in order to recognize the present end of record.

*Write Operation.* An end of record will not be written after C words have been written if the LCH instruction results in bringing into the channel a new word count.

In printing or the punching of cards, if the word count is reduced to zero on the 12-row right transmission point and the next command is an IORP or IORT, the end of the present machine cycle (record) will be recognized.

### IORT — Input-Output of a Record and Transfer



*Description.* See "Execution."

*Indicators.* Command trap.

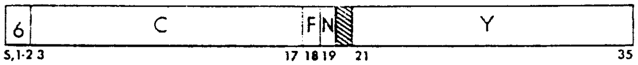
*Timing:* See I-O device being used.

*Execution*

*Read Operation.* Words are transmitted from tape and stored in consecutive storage locations until either an end of record is encountered or the word count has been reduced to zero. If the word count is reduced to zero (or is initially zero) before an end of record is reached, the rest of the words in that record are skipped without transmission to storage. When the record gap is reached, the next command is taken from a core location specified by the LCH instruction in the main program or disconnects (and traps if the channel is enabled) if no LCH is waiting.

*Write Operation.* When C words have been written, a record gap is written and the channel takes its next command from a core location specified by the LCH instruction in the main program or disconnects (and traps if the channel is enabled) if no LCH instruction is waiting.

### IOSP — Input-Output until Signal, then Proceed



*Description.* See "Execution."

*Indicators.* See I-O device being used.

*Timing:* See I-O device being used.

*Execution*

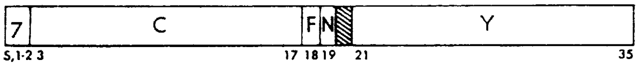
*Read Operation.* Words are read into consecutive storage locations beginning with location Y until either the contents of the word counter are reduced to zero or an end of record is reached. When either event occurs, the channel proceeds immediately to the next sequential command and executes it.

With a tape read operation and an IOSP or IOST command whose word count is reduced to zero by the last word in the record and whose next command is an IORP, IORT, IOSP, or IOST, this next command will normally enter the channel in time to recognize the present end of record. This next command transmits no data and is effectively skipped. If this next command cannot enter the channel within 11 machine cycles, the IORP, IORT, IOSP, or IOST command will not recognize the present end of record gap, but will process the following record. To determine if this sequence can be safely programmed, three cycles should be allowed if the CPU is processing a TCO instruction (five cycles if other instructions are being processed) plus one cycle for each channel in operation, plus one cycle for each channel programmed with proceed commands, plus one cycle for each channel which may process a TCH at this time, plus one cycle for each channel which may have indirect addressing at this time. If the total exceeds 11 cycles, the sequence described must not be used.

*Write Operation.* C words are written on tape beginning with storage location Y. When the specified words have been written, the channel proceeds to the next sequential command. An end of record is not written on tape when the word count is reduced to zero. Note that this is identical to the operation of the IOCP.

In printing or the punching of cards, if the word count is reduced to zero on the 12-row right transmission point and the next command is an IORP or IORT, the end of the present machine cycle (record) will be recognized.

### IOST — Input-Output until Signal then Transfer



*Description.* See "Execution."

*Indicators.* See I-O device being used.

*Timing:* See I-O device being used.

*Execution*

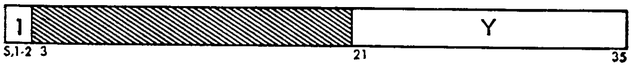
*Read Operation.* Words are read into consecutive storage locations beginning with location Y until either the contents of the word counter are reduced to zero or an end of record is reached. When either event occurs, the channel takes its next command from a core location specified by the LCH instruction in the main program or disconnects (and traps if the channel is enabled) if no LCH instruction is waiting.

With a tape read operation and an IOST command whose word count is reduced to zero by the last word in the record and whose next command is an IORP, IORT, IOSP, or IOST, this next command will normally enter the channel in time to recognize the present end of record. This next command transmits no data and is effectively skipped. If this next command cannot enter the channel within 11 cycles, the IORP, IORT, IOSP, or IOST command will not recognize the present end of record gap but will process the following record. To determine if this sequence can be safely programmed, two cycles should be allowed for the LCH instruction plus one cycle for each channel in operation, plus one cycle for each channel programmed with proceed commands, plus one cycle for each channel using indirect addressing. If the total exceeds 11 cycles, this sequence must not be used.

*Write Operation.* C words are written on tape from storage beginning with location Y. When C words have been written, the channel takes its next command from a core location specified by the LCH instruction in the main program or disconnects (and traps if the channel is enabled) if no LCH instruction is waiting to be executed.

In printing or the punching of cards, if the word count is reduced to zero on the 12-row right transmission point and the next command is an IORP or IORT, the end of the present machine cycle (record) will be recognized.

### TCH — Transfer in Channel



*Description.* This command is the transfer command for all data channels.

*Indicators.* None.

*Timing:* 2 cycles

*Execution.* When a TCH command is executed, the data channel proceeds immediately to its next command which is taken from location Y. The location register is set to Y + 1. The command located at Y is then loaded into the data channel.

### Program Examples

The following examples illustrate the use of data channel commands to read and write tape. The programs, aside from the instructions and command codes, are shown in the octal number system.

Figure 61 shows a program which may be used to read and skip tape records. The first instruction (500) selects the proper channel and tape. The second one loads the first command (1000) into the channel. The command execution proceeds as follows:

*1000 — IOCP 00006 0 03000.* This command reads the first six words from tape and places them into core locations starting with 3000.

*1001 — IOCPN 00005 2 00000.* This command reads, but does not transmit, the next five words, in effect skipping them.

*1002 — IORP 77777 0 03006.* The next command reads the remaining words in the record into locations starting with 3006. When the record gap is sensed, the next command (IOCD) will disconnect the channel and the tape unit, thus ending the channel program.

| Location | Instruction | | Comments |
|---|---|---|---|
| 00500 | RTDA | 01201 | Select tape 1, channel A |
| 00501 | RCHA | 01000 | Load first command |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| 01000 | IOCP | 00006 0 03000 | Read first six words |
| 01001 | IOCPN | 00005 2 00000 | Skip next five words |
| 01002 | IORP | 77777 0 03006 | Read remaining words in record |
| 01003 | IOCD | 00000 0 00000 | Disconnect |

Figure 61. Read and Skip Tape Program

Figure 62 shows a program that could be used to write all of core storage on a tape and delay at location 502 until all but the last word has been written.

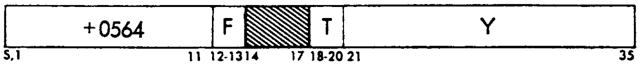
The first two instructions select the channel and tape unit to be used and load the first channel command. The central processing unit then senses the load channel instruction (LCHF) and the main program will wait until the next-to-last word has been written before loading the second channel command which will write location 77777 and disconnect both the channel and the tape unit.

| Location | Instruction | | Comments |
|---|---|---|---|
| 00500 | WTDF | 06202 | Select tape 2, channel F |
| 00501 | RCHF | 01000 | Load first command |
| 00502 | LCHF | 01001 | Wait until last word, then load channel F |
| . | . | . | |
| . | . | . | |
| . | . | . | |
| 01000 | IOCT | 77777 0 00000 | Write locations 00000-77776 |
| 01001 | IOCD | 00001 0 77777 | Write location 77777 and then disconnect |

Figure 62. Write Tape Program

## Channel Trap Instructions

### ENB — Enable from Y



*Description.* When this instruction is executed, the contents of location Y determine which signals may cause a trapping operation. Execution of each enable instruction cancels the effect of previous enable instructions. All channels may be disabled (traps will not occur) by executing an enable instruction whose operand contains all zeros.

*Indicators.* None.

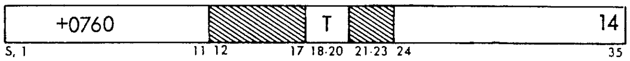
*Timing:* 2 cycles

*Execution.* Trapping signals are controlled as follows:

| SIGNAL DUE TO | CHANNEL | EFFECTIVE IF A "1" IN POSITION |
|---|---|---|
| Channel command or EOF | A | 0035 |
| Channel command or EOF | B | 0034 |
| Channel command or EOF | C | 0033 |
| Channel command or EOF | D | 0032 |
| Channel command or EOF | E | 0031 |
| Channel command or EOF | F | 0030 |
| Tape check | A | 0017 |
| Tape check | B | 0016 |
| Tape check | C | 0015 |
| Tape check | D | 0014 |
| Tape check | E | 0013 |
| Tape check | F | 0012 |

Execution of a trap will inhibit all further traps until a new enable instruction is executed or a restore-channel-trap instruction is executed. Depression of the reset or clear key will also disable all channels.

### RCT — Restore Channel Traps



*Description.* This instruction will allow traps to occur as specified by the previous enable instruction. It cancels the inhibiting effect of an executed trap.

*Indicators.* Trap Control.

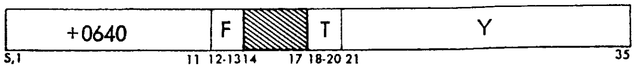
*Timing:* 2 cycles

*Execution.* Since the address part of this instruction is a part of the operation code, modification by an index register may change the operation itself.

## Input-Output Transmission Operations

Instructions that either send commands to a data channel or store information from data channel registers are classified as input-output transmission operations. These instructions are described in groups of eight. All eight instructions within a group are identical in function and differ only in that each refers specifically to one of the eight possible data channels. The instructions will be described for data channel A.

### SCHA — Store Channel A



*Description.* This instruction replaces the C(Y) with the contents of the channel A address, location, and operation registers. If channel A is not attached to the computer when the SCHA is given, the C(Y) are cleared by the SCHA.

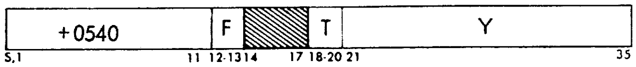
*Indicators.* None.

*Timing:* 2 cycles

*Execution.* The C(Y)$_{21-35}$ are replaced by the C(AR), the C(Y)$_{3-17}$ are replaced by the C(LR), and the C(Y)$_{S,1,2,19}$ are replaced by the contents of the operation register. An SCHA instruction may be executed at any time, regardless of whether or not the specified channel is in operation. If the channel is in operation and the channel registers are in the process of being changed, the execution of the SCHA will be delayed until the change has been completed. Note that the C(AR) will be one greater than the storage location of the last word involved in data transmission and that the C(LR) are one greater than the storage location from which the current command was taken.

| INSTRUCTION | CODE | NAME |
|---|---|---|
| SCHB | −0640 | Store Channel B |
| SCHC | +0641 | Store Channel C |
| SCHD | −0641 | Store Channel D |
| SCHE | +0642 | Store Channel E |
| SCHF | −0642 | Store Channel F |

### RCHA — Reset and Load Channel A



*Description.* If channel A has been selected by either an RDS or WRS, the C(Y)$_{S,1,2,19}$ replace the channel operation register, C(Y)$_{3-17}$ replace the C(WR) and the C(Y)$_{21-35}$ replace the C(AR). In addition, the number Y plus one replaces the C(LR).

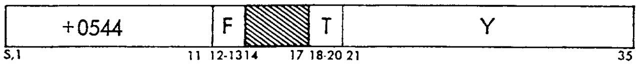
*Indicators.* I-O Check.

*Timing:* 3 cycles

*Execution.* If channel A is not selected when the RCHA is given, the RCHA executes normally but the I-O indicator is turned on. If the command loaded by the RCHA specifies indirect addressing, it will not occur. For each RDS or WRS, the corresponding RCHA must be given if any transmission between storage and the selected I-O device is to take place. If a second RCHA is given at a later time, the order is executed immediately. See "Programmed Interruption of a Data Channel" for further details regarding this type of operation.

| INSTRUCTION | CODE | NAME |
|---|---|---|
| RCHB | −0540 | Reset and Load Channel B |
| RCHC | +0541 | Reset and Load Channel C |
| RCHD | −0541 | Reset and Load Channel D |
| RCHE | +0542 | Reset and Load Channel E |
| RCHF | −0542 | Reset and Load Channel F |

### LCHA — Load Channel A



*Description.* If the data channel has been selected, the computer delays until an IOCT, IORT, or IOST command is processed for channel A or the channel leaves operation. After an IOCT, IORT, or IOST command has been executed by the channel A, the LCHA is executed as shown below.

*Indicators.* I-O check.

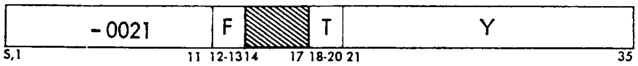
*Timing:* 3 cycles, modification 8

*Execution.* The $C(Y)_{S,1,2,19}$ replace the contents of channel A operation register. $C(Y)_{3-17}$ replace the C(WR), the $C(Y)_{21-35}$ replace the C(AR), and the number Y plus one replaces the C(LR). If an LCHA is issued and either (1) the channel is not selected, or (2) channel A is selected but an IOCT, IORT, or IOST command is not executed before the channel disconnects, the I-O check indicator is turned on and the LCHA is treated as a no-operation.

| INSTRUCTION | CODE | NAME |
|---|---|---|
| LCHB | −0544 | Load Channel B |
| LCHC | +0545 | Load Channel C |
| LCHD | −0545 | Load Channel D |
| LCHE | +0546 | Load Channel E |
| LCHF | −0546 | Load Channel F |

## System Compatibility Operations

### ESNT — Enter Storage Nullification and Transfer



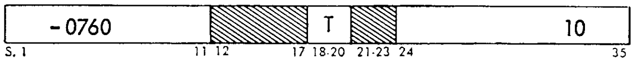
*Description.* This instruction turns on a half-storage mode indicator, which serves as a protective device for a program being run while using the compatibility feature of the computer

*Indicators.* Simulate.

*Timing:* 2 cycles

*Execution.* With the half-storage mode indicator on, the following events occur: (1) the upper half of storage is made unavailable for reference by a 704 program, (2) index register capacity is halved, and (3) program control is transferred to location Y. The indicator may be reset by depressing the reset, clear, or any of the load keys, execution of any I-O trap (except data channel trap), or execution of an LSNM instruction.

### LSNM — Leave Storage Nullification Mode



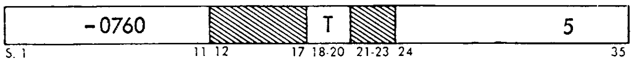
*Description.* Execution of this instruction returns the computer system to its normal operating capacity by turning the half-storage mode indicator off. If the computer is in its normal operating mode and an LSNM instruction is executed, the instruction will be treated as a no-operation.

*Indicators.* Simulate.

*Timing:* 2 cycles

*Execution.* Since the address part of this instruction is a part of the operation part, modification by an index register may change the operation itself.

### ESTM — Enter Select Trap Mode



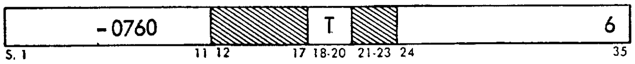
*Description.* This instruction turns on the select trap mode indicator, and, while in this mode, I-O select and sense instructions are not executed but are trapped. It should be used before entry into a 704 program, so that a 704 instruction will be trapped rather than result in indefinite delays.

*Indicators.* Simulate.

*Timing:* 2 cycles

*Execution.* Instructions that will be trapped include: WEF, BSF, BSR, REW, RDS, WRS, I-O sense, RTT, EOT, redundancy and BOT. The location plus one of the trapped instruction is stored in core storage location 10,000 or 40,000 (depending on which core storage is being used). Program control is transferred to location 10,001 or 40,001. Trapping also turns off the half-storage, select trap, and copy trap indicators. The indicators may also be turned off by: depression of reset, clear, or load keys, or execution of any I-O trap (except a data channel trap). Since the address part of this instruction is a part of the operation code, modification by an index register may change the operation itself.

### ECTM — Enter Copy Trap Mode



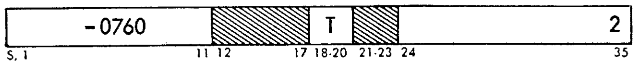
*Description.* This instruction turns on the copy trap mode indicator. With the indicator on, CPY, CAD, and LDA instructions are trapped instead of being executed.

*Indicators.* Simulate.

*Timing:* 2 cycles

*Execution.* The location plus one of the trapped instruction is stored in core location 10,000 or 40,000 and program control is transferred to location 10,002 or 40,002. The execution of this instruction will also turn off the half-storage, select trap, and the copy trap mode indicators. The copy trap mode indicator may also be turned off by: depression of the clear, reset, or load keys, or the execution of any I-O trap (except a data channel trap). Since the address of this instruction is a part of the operation code, modification by an index register may change the operation itself.

### EFTM — Enter Floating Trap Mode



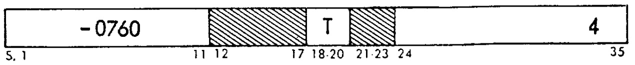
*Description.* This instruction turns on the indicator for floating-point trap mode. When in this mode, floating-point overflow and/or underflow will cause a trapping operation.

*Indicators.* MQ overflow.

*Timing:* 2 cycles

*Execution.* This mode is the normal operating mode. Floating-point overflow-underflow have the operating characteristics of the standard computer (store location plus one in address 0000 and then transfer to 0010). Since the address part of this instruction is a part of the operation code, modification by an index register may change the operation itself.

### LFTM — Leave Floating Trap Mode



*Description.* This instruction turns off the floating trap mode indicator, giving the computer floating-point overflow characteristics of the standard 704 (turns on the AC or MQ overflow indicators only).

*Indicators.* MQ overflow.

*Timing:* 2 cycles

*Execution.* Depression of the reset, clear, or load keys will return the computer to its normal operating mode (floating trap mode) by turning the floating trap mode indicator on. Since the address of this instruction is a part of the operation code, modification by an index register may change the operation itself.

## Systems Program Compatibility

### 704 Programs on 709 System

The compatibility II program makes possible the execution of programs written for a 704 system on the 709 system. The compatibility program simulates 704 input-output operations through use of the storage-nullification, input-output select trap, and copy trap modes. The program requires no modification of the 709 on which it is used; however, it cannot be used on a 709 system with less than 8192 words of core storage.

Compatibility II executes a leave floating-point trap mode (LFTM) instruction before entering a 704 program, in order that overflow will function as on a standard 704. Computations are not affected in any other way. The program is designed to use all of the upper half of storage. Some of these locations are used for storing the program itself; the rest are used as an input-output buffer between the 704 program and tape or card units, and, if required, for simulating magnetic drums. All locations of the upper half of storage not used for the compatibility program instructions or for drum simulation are used for the input-output buffer.

Before a 704 program can be processed, a control card must be read. This control card indicates the 709 equivalents of the 704 tapes used in the processing of the 704 program. After the control card is read, the compatibility program enters select trap, copy trap, and storage nullification modes and simulates a load card, load tape, or load drum operation depending on the setting of the console entry keys. Until the completion of the 704 program, input-output operations are simulated through use of the select trap and copy trap modes of operation. (All other instructions are compatibile with the, 709.)

A complete explanation of the compatibility II program is found in the *IBM 709 Data Processing System Bulletin,* J28-6039.

### 709 Programs on 7090 System

Programs written for the 709 may be run on the 7090 without modification or sacrifice in efficiency and still take advantage of the increased system speed. There are, however, differences which are potential areas of incompatibility. These are:

1. Read or write drum instructions will be trapped if an ESTM instruction has been executed. CPY, CAD, and LDA instructions will be trapped if an ECTM is executed. When the instruction is not trapped, the I-O indicator will be turned on and the instruction will be treated as a no-operation. CRT instructions may be trapped but they will always turn on the I-O check indicator.

2. It is usually possible to simulate the drum on the 7090 system by using the 704 compatibility feature. If, however, the 709 program uses data channel traps, difficulty may be encountered if traps occur while the compatibility program is being executed. This will result in returning control to the 709 program without allowing the 7090 to set proper operating modes.

3. The change in the ratio of compute speed to input-output speed will affect programs that depend upon computed delays for satisfactory operation. Since the 7090 is faster than the 709, difficulty will occur only in cases where "shrinkage" of a delay loop can cause trouble. For instance, a program may assume that certain storage locations may be changed *n* machine cycles after a write tape instruction. Thus, if that area is used without making a test, the 709 and 7090 may not write the same data.

4. To achieve compatibility, the 7090 system must have the same complement of data channels, tape units, and card equipment. Further, these units must be arranged in the same way with regard to addressing.

5. Since magnetic tape may be recorded at two densities on the 7090, provision must be made to set up the tape units for the particular density required. This may be done by use of the change density switch located on each tape unit.

### 7090 Programs on 709 System

To run a 7090 program on a 709, the same general precautions observed with a 709-to-7090 program must be used. Instructions pertinent to the 7090 only (instructions referring to channels G and H) will cause the 709 to hang up.

Inasmuch as the computed delays will be much longer when 7090 programs are run on a 709, trouble will be encountered whenever some critical timing may not be exceeded. For example, if the maximum allowable number of 7090 instructions are executed between select and reset and load instructions, an I-O check will result when this program is run on the 709. This is essentially the inverse of the problem encountered when 709 programs are run on a 7090. Again, the size and configuration of the systems must be the same. It is possible, however, to write programs for 7090 systems which are not attainable with the 709. This will occur because a 7090 data channel has more than eight tape units, the 7090 system uses channels G and H, and card equipment on the 7090 may appear on channels B, D, or F.

Since the 709 does not have dual-density tape units, all programs will produce low density tape as output and must have low density tape as input.

The keys and lights found on the 709 operator's and data channel consoles and on the tape control are described in this section of the manual. The operation of these keys, when the central processing unit and/or a channel is in manual status, is also given.

## IBM 709 Operator's Console

The 709 console panel lights are divided into two sections, for discussion. The first section concerns neon lights and the second section incandescent lamps, including the color of the lamp cover. The items are numbered in the discussion for easy reference to Figure 121.

### Neon Lights

*1. Instruction Counter.* The instruction counter is 15 positions long to accommodate the largest core storage address (with an IBM 738 Core Storage). If the 709 system has one 737 unit, only 12 of the 15 positions are used. If two 737 units are used, only 13 positions of the instruction counter are active.

The instruction counter (IC) is used to tell the computer the location of the next instruction to be performed. It may be reset to zero at the start of the program or may be set to a predetermined address. Once the program is started, the IC counts sequentially unless a transfer instruction is executed. In this case,

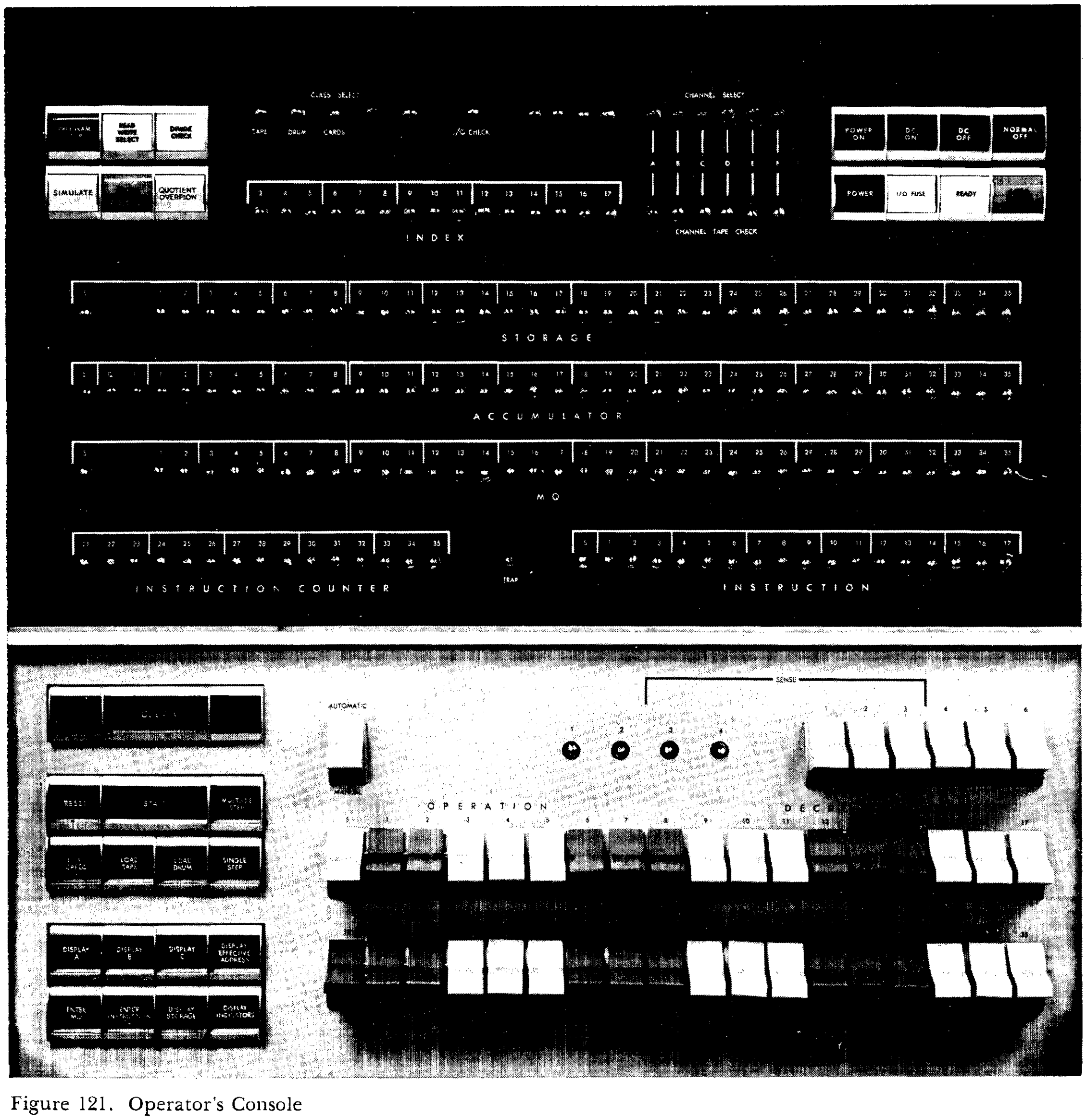


Figure 121. Operator's Console

the IC is set to the address specified by the transfer instruction and is again stepped sequentially starting with this new address. The highest location in core storage and location zero are treated as consecutive addresses.

The IC is normally advanced at the end of the I cycle. However, some instructions cause the computer to skip one or two instructions and, therefore, the IC may be advanced as many as three times while the instruction is being completed. If a halt occurs during the execution of a divide or halt instruction or a halt and proceed instruction, the IC has the address of the instruction being executed *plus one.* Upon execution of a halt and transfer instruction, the IC contains the address of the instruction.

If the auto-manual switch is depressed during the execution of a CPU program, the IC contains the address of the instruction to be executed, *plus one.*

2. *Instruction Register.* Instructions and data are both in the form of 36-bit words, and the computer finds the difference between the two in the following manner:

1. Any word brought into the computer during an I cycle is treated as an instruction.
2. A word brought in at any other time is treated as data.

The instruction register (IR) is divided into two parts: positions S, 1-9 contain the operation part of the instruction, while positions 10-17 form a counter known as the "shift counter." In shifting, multiplication, and division instructions, the number of shifts to be made is placed in this counter.

With an I-O instruction, the address part of the instruction is placed into the shift counter. The counter in turn sets the class and unit selectors for the type of instruction (such as read, write, and so forth) and the exact unit involved.

3. *Internal Registers.* The contents of the internal registers (storage, accumulator, and MQ) are displayed on the console. The display is marked off in groups of three, making the direct conversion from binary notation to octal a matter of sight.

4. *Index Registers.* A row displays the contents of any index register, depending on which one of the display (A, B, or C) keys is depressed with the computer in manual status.

5. *Trap Light.* The trap light goes on whenever the CPU is operating in the transfer trap mode.

6. *Sense Lights.* There are four sense lights which may be turned on or off by the main program. They are explained under plus and minus sense instructions in the instruction section.

7. *I-O Check Light.* This light goes on as stated in the description of the I-O check test instruction. The light may be turned off by the execution of an IOT.

8. *Tape Check Lights.* The tape check lights (six lights, one for each channel) are turned on if any error is detected while writing or if both the HI and LO registers are in error on reading. The lights may be turned off by the execution of a transfer on DSC redundancy check.

9. *Channel Select Lights.* These lights (six, one for each channel) will be turned on according to the channel that is in operation. They will be turned off if the channel is not in operation.

NOTE: Both the tape check and the channel select lights are duplicated on the data channel panel.

10. *Class Select Lights.* There are three class select lights (tape, drum, and card machines). One of these lights will be on according to the I-O class being selected or waiting to be selected.

### Incandescent lights

11. *Simulate (Yellow).* The simulate light will be turned on when the 709 is operating in the following modes associated with the 704-709 compatibility feature.

Input-output select and sense trap mode
Copy and load drum address trap mode
Storage nullify mode

12. *MQ Overflow Light (White).* This light will be on whenever an MQ overflow occurs and the calculator is using the compatibility program.

13. *Accumlator Overflow Light (Green).* This light will be turned on at any time (during fixed point operation or shifting operations) when a carry out of position 1 of the accumulator occurs. It may be turned off by the TNO or TOV instructions.

14. *Divide Check Light (Yellow).* The divide check light will be turned on (fixed point division) if the dividend (accumulator) is greater than or equal to the divisor (storage). On floating point division a divide check occurs if the divisor is zero or if the magnitude of the fraction of the dividend is greater than or equal to twice the magnitude of the fraction of the divisor. The divide check indicator is tested by the DCT instruction.

15. *Read-Write Select Light (Yellow).* The read-write select light will be turned on when an input-output unit has been selected for reading or writing. The light goes off when the input-output unit is disconnected.

*16. Program Stop Light (Red).* This light will be turned on when the calculator executes a HALT instruction and stops.

*17. Automatic Light (Green).* The automatic light is on if the calculator is executing instructions in the automatic mode.

NOTE: Any of the above lights may also be turned off by using the clear or the reset keys on the 709 console.

*18. Ready Light (White).* The ready light indicates that the electronic circuits have reached operating level. The light remains on except when the system is operating in automatic mode or when processing stops owing to a halt and no channel is in operation. It is important to make sure that this light is on before using the computer for any operation.

*19. Power-On Light (Red).* This light is on whenever power is applied to the 709 system (except the tape and data channels).

*20. I-O Fuse Light (Yellow).* The I-O fuse light will be turned on if an I-O indicating fuse is blown. The fuse light will be turned off when the fuse has been replaced by a customer engineer.

### Panel Keys and Switches

*21. Auto-manual Switch.* Pressing this switch down stops the computer after it has completed the execution of the instruction then being processed, unless an I-O unit is connected to the logical unit. In this case, the computer stops after the I-O unit in use has been disconnected. The automatic light goes out, and all of the switches and the following keys become effective: display sense indicators, enter MQ, enter instruction, display effective address, display A, display B, display C, multiple step, and single step. The clear key and load key become ineffective.

*22. Single Step and Multiple Step Keys.* These keys enable the operator, when the 709 is in manual status, to proceed with his program either step-by-step (one step at a time) or at a slow automatic rate of speed. If an instruction is executed which causes an input-output device to be connected to the computer, the computer operates in the automatic mode until the I-O unit is disconnected. When this occurs, the computer returns to the manual mode.

*23. Sense Switches.* Six sense switches give the operator manual control over the program while it is being executed by the computer at high speed. At various points in the program, giving sense instructions with the addresses of the sense switches causes the computer to follow one of two courses, depending on whether or not the sense switch tested is depressed. The sense switches are also effective while the computer is in manual status.

*24. Panel Input Switches.* These 36 panel input switches enable the operator to insert a word of information into the MQ or the instruction registers of the computer if it is in manual status and the enter-MQ or enter-instruction key is depressed. When a panel input switch is down, it represents "1"; when up it represents a "0". (A bit configuration may be set in these keys, and with the calculator in automatic mode, an ENK instruction will set the input switches' contents into the MQ.)

*25. Index Display Keys.* The three index display keys let the operator display the contents of any of the index registers, while the computer is in manual, by pressing the key marked with the letter corresponding to the index register in question. For example, to display the contents of index register A, the operator presses the display A key; the contents of index register A then appear in the index register neons. To display index register B, the operator presses the display B key. The contents of index register B would then replace the information from index register A. The index register remains displayed until the computer is returned to automatic status.

*26. Load Keys.* The load keys let the operator initiate the loading of a self-loading program stored on binary cards, a drum, or a tape. If a self-loading program is stored on the tape whose logical identification is 221, and is attached to channel A, pressing the load-tape key causes the computer to perform the following sequence of instructions:

Read select channel A for tape unit 221.

Reset load channel A with a bit in position S, a word count of 3, and an address of 0000. (The first three words are sent to core storage.)

The contents of location zero are sent to channel A as a command.

Transfer to core storage location 0001.

This sequence of instructions starts the loading of a program stored on tape 221.

Pressing the load card key causes the same sequence of instructions to be executed, except that the address in the first instruction is 321, selecting the reader instead of the tape unit.

A somewhat similar situation holds for the load drum key except that the instructions are as follows:

Read select drum 301.
Copy 0000.
Copy 0001.
Transfer 0000.

When loading is started, it is essential that the particular input unit from which information is to be loaded into storage be in ready status. Depressing the load keys resets both channels A and B. Note that the MQ register will not be reset. The keys are operative only when the auto-manual switch is in automatic and the 709 is in a ready status.

*27. Reset Key.* Pressing the reset key resets all registers and indicators in the logical section of the machine, except the SI. That is, the SR, AC, MQ, instruction location counter, instruction register, and index registers are set to zero and all indicators are turned off. The panel lights are all turned off with the exception of power and ready. Core storage is not affected by the reset key. Any channels in automatic status, and their associated registers, are also reset.

*28. Clear Key.* With the computer in automatic status, pressing the clear key sets all magnetic cores to zero. In addition, all registers and indicators are reset as with the reset key depression. The clear key is inoperative when the computer is in true manual status.

*29. Start Key.* Pressing the start key continues calculation at high speed if the computer has halted at a program stop, or if it has been returned to automatic operation after having been in manual status. Pressing the start key will reset the program stop light, and calculation starts with the operation specified in the instruction counter. Pressing the start key (CPU in manual) resets the program stop or read-write check light. All register contents are not destroyed. An index register may then be displayed or the program may be stepped at slow speed.

*30. Enter MQ Key.* If the operator manually keys a given word of information into the panel input keys and if the enter-MQ key is pressed while the calculator is in manual, then the keyed-in word replaces the contents of the MQ. The contents of the SR are destroyed by this operation.

*31. Enter Instruction Key.* If the operator presses the enter instruction key while the computer is in manual status, the word in the keys is executed.

*32. Display Storage Key.* If, while the computer is in manual status, the operator keys a storage location into the address part of the panel entry keys and presses the display storage key, the contents of the address appearing in the keys are displayed in the SR where they may be read from the SR lights.

*33. Display Effective Address Key.* Assume that the computer is in manual status, an instruction is in the SR, and the display effective address key is pressed. The difference between the contents of the address field in the SR and those of the index register tagged in that instruction (if one is tagged) will appear in the address field of the SR, where it may be read from the SR lights.

*34. Display Sense Indicators.* If the display-sense-indicators key is depressed while the computer is in manual, the contents of the 36-position sense-indicator register will be displayed in the storage register neons. The contents of the storage register are destroyed by this sequence but the contents of the sense-indicator register remain unchanged.

*35. Power-On, Normal-Off, DC-On, and DC-Off Keys.* These keys are for servicing the system and have no programming significance.

### Operation

The sequence of operations necessary to enter a constant or new instruction into a core storage location is as follows: Assume that the instruction CLA 0100 is to be inserted in storage.

1. With the auto-manual switch in the manual position, set the information to be stored (CLA . . . 0100) into the panel entry keys.

2. Depression of the enter MQ key stores the contents of the entry keys into the MQ register.

3. Set –0600 (store MQ), with an address of 0200 into the keys and then push the enter instruction key. This action stores the contents of the MQ register (CLA . . . 0100) into core storage location 0200.

## Data Synchronizer Console

### Indicator Lights

The contents of all registers and counters in the 709 system data synchronizer (DS) are displayed on the neon indicator panel (Figure 122). In addition, the input-output selection and operation indicators (S, 1, 2 and 19) are displayed. There are also nine special indicators for each channel on the neon panel. For reference assume that vertical rows of neons are numbered 1 through 18 and the horizontal rows are lettered A through O.

*1. Data Register (A-B, I-J, 1-18).* These neons reflect the contents of the data register.

*2. Word Counter (C, K, 4-18).* The word count minus one is indicated with these neons.

*3. Address Register (D, L, 3-18).* The core storage address is indicated with these neons.

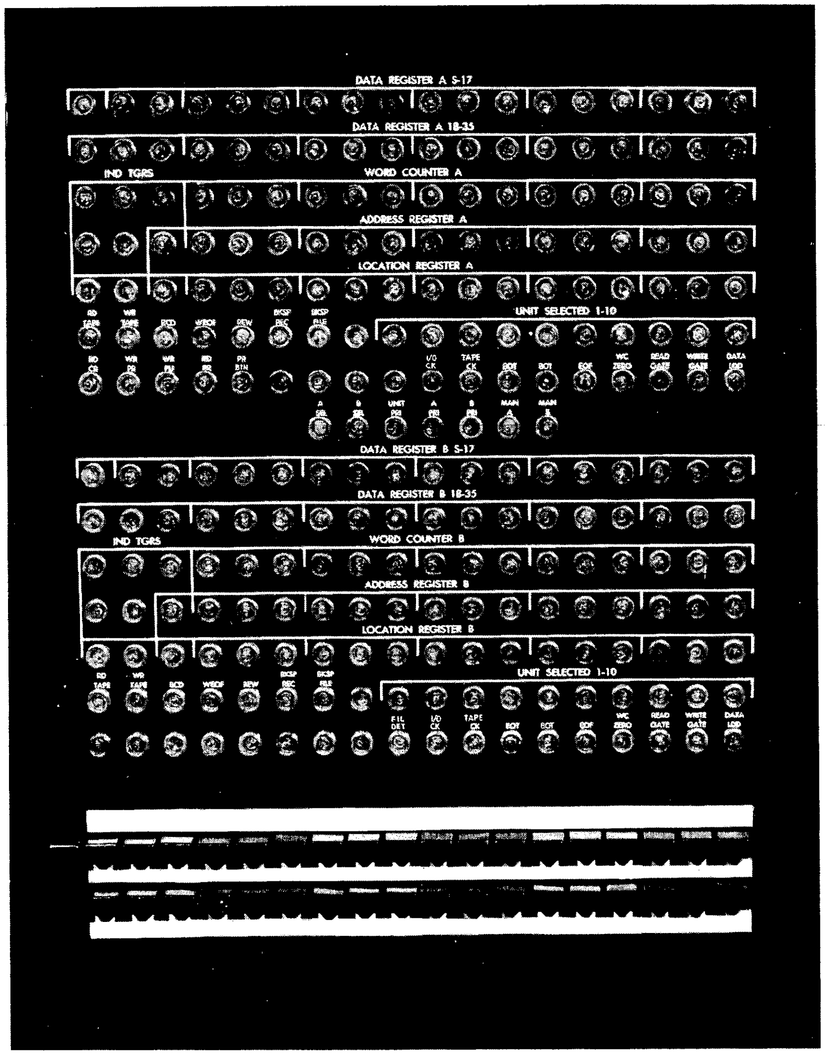


Figure 122. DSC Console, Top

4. *Location Register (E, M, 3-18).* The location register neons contain the address of the core storage location of the next command to be executed.

5. *I-O Check (G, O, 10).* This indicator turns on for lack of a storage reference cycle on one channel operation.

6. *Tape Check (G, O, 11).* This indicator is turned on if any error is detected while reading or writing on tape. The channel tape check indicator parallels the tape check indicator on the 709 console.

7. *End of Tape (G, O, 12).* The end-of-tape indicator will be turned on, during a write operation, whenever the end-of-tape reflective spot of the selected tape passes the read-write head.

8. *Beginning of Tape (BOT) (G, O, 13).* This indicator is set whenever a backspace record or backspace file instruction moves a tape to its load point or attempts to backspace it beyond its load point.

9. *End of File (G, O, 14).* When a disconnect occurs because an end of file is sensed while reading cards or tape, this indicator is turned on.

10. *Word Count Equal to Zero (G, O, 15).* This indicator will be turned on whenever the word count is equal to zero.

11. *Read Gate (G, O, 16); Write Gate (G, O, 17); Data Register Loaded (G, O, 18).* These three neon indicators are customer engineering aids.

12. *Input-Output Indicators: Read Tape (F, N, 1); Write Tape (F, N, 2); BCD (F, N, 3); WEOF (F, N, 4); Rewind (F, N, 5); Backspace Record (F, N, 6); and Backspace File (F, N, 7).* These indicators are turned on whenever the individual operations are being executed or stacked in the channel.

13. *Unit-Selected Indicators (F, N, 9-18).* There are ten unit-selected indicators for each channel. During

an automatic operation they are turned on from the 709 unit register or PSE instruction to designate the unit which is to be selected. Under manual conditions, the indicators will be turned on according to the setting of the unit selection switches.

*14. Card Machine Indicators: Read Card Reader (G, 1); Write Printer (G, 2); Write Punch (G, 3); Read Printer (G, 4); and Print Binary (G, 5).* These indicators are concerned with channel A, C or E. They will be turned on whenever the particular card machine is selected.

*15. Indicator Triggers (C, K, 1-3; D, L, 1-2).* These indicators will be on or off according to the bit configuration of the command in the channel. They are explained in detail under "Data Channel Registers."

*16. A Select (H, 7); B Select (H, 8).* These indicators reflect which channel is in operation.

*17. Unit Priority (H, 9).* This indicator, when on, shows which of the three DS's has priority at any instant.

*18. A Priority (H, 10); B Priority (H, 11).* One of these indicators will be on, showing which channel has priority, assuming this DS has priority.

*19. Manual A, B (H, 12, 13).* These indicators reflect the setting of the auto-manual switches.

*20. Fil. Det. (O, 9).* This indicator is for customer engineering use.

## Keys and Switches

The operator's panel on the data synchronizer console is shown in Figure 123. It contains all switches, keys, and lights necessary for DS operation. Some of the switches are a locking type and others are spring-return switches.

### Locking Keys

*1. Entry Keys.* There are 36 entry keys, one for each position of the 709 word. When a key is depressed, it remains in the position until reset by the reset key for the entry keys. These keys are operative only when the channel is in manual status.

*2. Auto-Manual Channel A.* In automatic position, this switch permits 709 operation of channel A. It permits resets initiated by the 709 to reset channel A indicators. In manual position, this switch activates the other manual controls and entry keys for channel A. It also permits the reset key on the DS to reset channel A indicators while blocking any resets originating in the 709.

With this switch in the manual position, the following instructions will be affected as indicated:

Transfer on channel A in-use will not transfer.
Transfer on channel A not-in-use will transfer.
Transfer on channel A EOF will not transfer.
Transfer on channel A redundancy-check will not transfer.
Store channel A probably will store zeros, but may store any combination of zeros and ones.

The execution of any of the following instructions will result in "hanging up" the CPU:

Load channel A or reset and load channel A.
Read channel A or write channel A.
Rewind channel A or write end of file channel A.
Backspace record channel A or backspace file channel A.

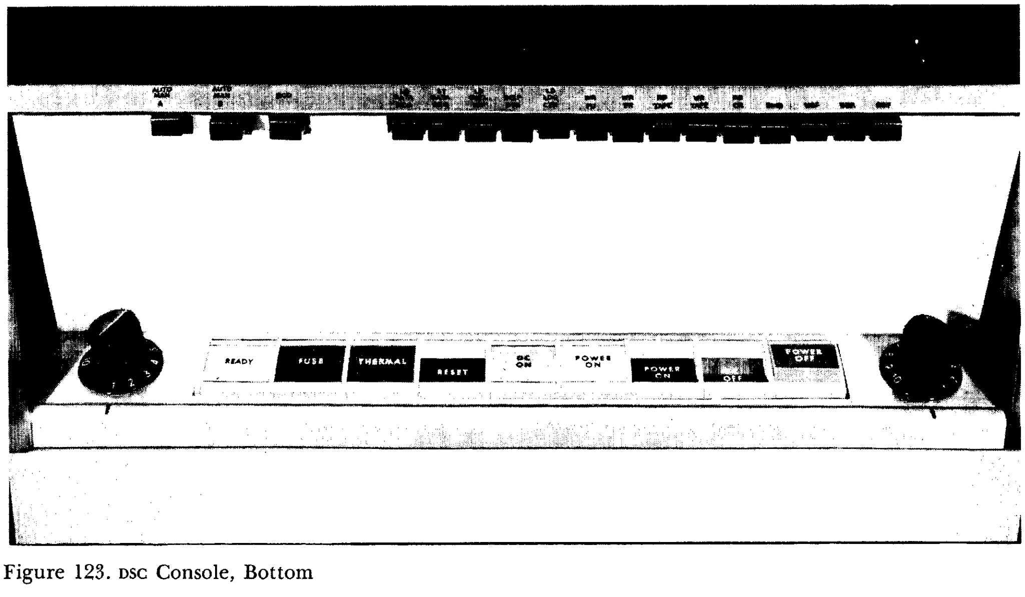


Figure 123. DSC Console, Bottom

If either sense-printer-channel-A or sense-punch-channel-A instruction is executed, it will be treated as no-operation. With the execution of either beginning-of-tape-test-channel-A or end-of-tape-test-channel-A, the program will skip the next instruction and then proceed.

*3. The automatic, read-write select, channel select, tape check, or the I-O check lights* on the CPU console will not be turned on by operations of a channel that is in manual status.

*4. Auto-Manual Channel B.* This switch operates the same as the channel A switch except that it pertains to channel B.

*5. BCD.* The BCD switch is operative only in manual status and, when depressed, forces a BCD mode of operation.

### Spring-Return Keys

*6. Load Data Register.* Depression of this key sets the condition of the entry keys into the data register of whichever channel is in manual status. The key has no effect if both channels are in automatic.

*7. Store Data Register.* This key, when depressed, will cause the contents of the data register, for the channel in manual status, to be stored in core storage at the address in the address register of the channel. After the word is stored, the address register is increased by 1 and the word count register is decreased by 1. When both channels are in automatic, this key is inoperative.

*8. Load Control Word.* This key, when depressed, causes information set up on the DS entry keys to be entered into the indicators, word counter, and address register of whichever channel is in manual status. Entry keys S, 1, 2 and 19 are entered into the indicators, keys 3-17 are sent to the word counter, and keys 21-35 are sent to the address register.

*9. Display Storage.* Depression of this key will cause the contents of the storage location, whose address appears in the channel's address register, to be displayed in the data register of whichever channel is in manual status. The key is inoperative if both channels are in automatic status.

NOTE: If either the store data register or the display storage key is depressed more than one time, for each depression the next sequential storage location will either receive the contents of the data register or be displayed in the data register.

*10. Load Location Counter.* Depression of this key causes the information set up in the DS entry keys (21-35) to be entered into the location counter of whichever channel is in manual status.

*11. Write Punch, Write Printer, Read Tape, Write Tape, Read Reader.* These keys control some phase of input-output operation when a channel is in manual status. They are inoperative if the channel is in automatic status. They are explained as a group because their operation is identical except for class of operation. One example of their use is in punching a card:

1. The auto-manual switch for channel A is in manual.
2. A command with a count of 24 (keys 13 and 14) and an address of 100 (keys 11, 12 and 15) is set up in the DS entry keys.
3. The load control key of the DS is depressed, entering the command set up in step 2.
4. The write-punch key is depressed.

These steps would result in writing 24 words from core storage, starting at core location 100. Upon completion of the punching operation, the last word transmitted to the punch would be retained in the data register, the word count would be reduced to zero, and the address register would have advanced once for each word that was punched.

In another example, reading a tape, the sequence of operations would be the same as those in the first example for steps 1, 2, and 3. Step 4 would change in three ways:

a. The tape selector switch on the DS, for the appropriate channel, would be set to the tape unit desired.

b. The mode, BCD or binary, would be set by use of the BCD switch.

c. The rest of the operation would be the same as example 1 except that the first 24 words on the selected tape unit would be stored in locations 100 through 123.

*12. Rewind, Write End-of-File, Backspace Record, Backspace File.* These keys are also grouped because their operation is the same except that they refer to the non-data type of operation and are concerned only with tape. Depending on the setting of the selector switch on the DS, the specified tape unit would, according to which key was depressed, rewind, write an end-of-file, backspace one record, or backspace one file.

*13. Unit Selection Switches.* Two selection switches, one for each channel and each numbered 1 through 10, control unit selection under manual status. For example, assume that the selection switch for channel A is turned to position 3 with channel A in manual status. Any tape unit attached to channel A whose individual selector switch is set at 3 will be selected for manual operation under the control of the channel tape control keys.

In addition to its tape function, the channel A switch is active when a printer or punch select key is depressed to activate the corresponding sense exit on the printer or punch control panel. If no sense exit is to be set, the switch must be set to an unlabeled position.

*14. Fuse Light.* When this light is on, it indicates a blown fuse in the DS. A DC-off or power-off condition will also have occurred.

*15. Thermal Light.* This light indicates, when lit, that the temperature inside the DS has gone beyond limits. A power-off condition occurs at the time the light is turned on.

*16. Power-On Light.* This light indicates that AC power is being supplied to the DS for filaments, blowers and power supplies.

*17. DC-On Light.* This light indicates that all voltages in the DS have reached operating levels.

*18. Ready Light.* This light indicates that both channels in the DS are available for automatic operation.

*19. Power-On Key.* This key initiates a sequence of operations in which the various voltages are supplied to the DS in the proper order.

*20. DC-Off Key.* Depression of this key causes the removal of all DC voltages from the tube panels. Power is still supplied to the filaments, blowers and power supplies.

*21. Power-Off Key.* Depression of this key causes all power to be removed except the 40-volt supply and 110 volts AC to a customer engineering outlet.

*22. Reset Key.* This key is operative only if one or both of the DS's auto-manual switches are in manual. If either or both are in manual, depression of the reset key will reset all indicators, registers and counters which are under manual control. With the DS in automatic, all resets originate in the 709.

# Programming Examples

A computer program is similar to the program received at baseball games, concerts, and many other presentations in that it is a plan of operations or events that will occur. The process of getting to work each morning may be compared to a program concerned with the following problem: Compute A + B − C and store the result (D), if it is a plus number; if minus, halt the computer (Figure 124).

Block diagrams, also called flow charts, are a schematic diagram of the logic of the computer and methods it uses in solving a problem. The main reason for a flow chart is that it is easier to write and understand than a written paragraph about the problem. The flow chart is a map of all logic paths and decisions used by the computer, and simplifies the writing of a coded computer program.

The same program used in Figure 124 may be expressed in program terminology as shown in Figure 125. Given: Factor *A* stored in location 100, factor *B* in 200, factor *C* in 300.

The instruction location designates the place, in core storage, where the instruction is stored. The instruction abbreviations are such that they represent the actual operation involved. For example, SUB means subtract and STO means store, while CLA means clear the register to zero and add. The address part designates a location in core storage where a number is located or where a number may be stored. Thus, the operation of the program would proceed as follows.

The program is started with the first instruction (CLA 100) which is contained in location 0000. This instruction will clear the accumulator register to zero and then bring the contents of core location 100 into the accumulator (factor *A*). The next instruction (ADD 200) will bring factor *B* from storage and combine it with factor *A*. The third instruction (SUB 300) brings factor *C* from storage and subtracts it from the combined factors *A* and *B*. The fourth instruction then takes the result in the accumulator and stores it in storage location 400. Thus *D* has been formed and stored.

| Instruction Location | Instruction | Address |
|---|---|---|
| Move A ............ 0000 | CLA | 100 |
| Form A + B .......... 0001 | ADD | 200 |
| Form A + B − C ...... 0002 | SUB | 300 |
| Form Answer (D) ..... 0003 | STO | 400 |

Figure 125. Simple Program

A possible use of two of the shifting instructions is shown in Figure 126 with the following facts known. Two numbers are contained in the same storage location. One number is located in positions 6 through 20, and the other is in positions 21 through 35. (Assume that this word is already located in the accumulator.) The problem is to multiply the number in positions 6-20 by the number in positions 21-35.

| Location | Instruction | Address | Comments |
|---|---|---|---|
| 0000 | LRS | 0015 | Move positions 21-35 into the MQ. |
| 0001 | RQL | 0016 | Align this number in proper place to be used as the multiplier. |
| 0002 | STO | 0100 | Store the multiplicand so that it may be used in the multiplication. |
| 0003 | MPY | 0100 | Multiply the two numbers. |
| 0004 | STQ | 0200 | Store the result. (STQ is used because the result is small enough to be completely contained in MQ.) |

Figure 126. Multiply and Shifting Problem

Conditional transfers may be used to solve the following type of problem. Assume that *A* and *B* are two positive numbers located in storage at locations 100

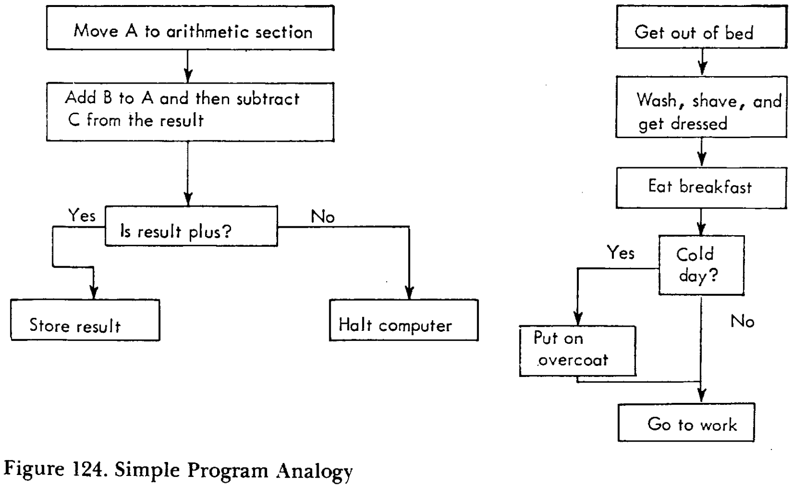


Figure 124. Simple Program Analogy

| Location | Instruction | Address | Remarks |
|---|---|---|---|
| 0000 | CLA | 0100 | Factor A |
| 0001 | SUB | 0101 | Subtract factor B from factor A |
| 0002 | TZE | 0017 | Factors are equal |
| 0003 | TPL | 0005 | A is larger than B |
| 0004 | TMI | 0012 | A is smaller than B |
| 0005 | CLA | 0100 | Factor A |
| 0006 | STO | 0201 | Store A |
| 0007 | CLA | 0101 | Factor B |
| 0010 | STO | 0200 | Store B |
| 0011 | HTR | 0022 | Stop. A was larger than B |
| 0012 | CLA | 0100 | Factor A |
| 0013 | STO | 0200 | |
| 0014 | CLA | 0101 | Factor B |
| 0015 | STO | 0201 | |
| 0016 | HTR | 0022 | Stop. A was smaller than B |
| 0017 | CLA | 0100 | Factor A |
| 0020 | STO | 0200 | |
| 0021 | HTR | 0022 | Stop. A was equal to B |
| 0022 | Proceed with program | | |

Figure 127. Flow Chart and Program for Sorting

and 101. The problem is to find the smaller number and put it in location 200; also, to place the larger number in 201. If they are equal, put one number in 200 and nothing in 201. The computer program is shown in Figure 127.

The use of index registers can be pointed up by showing the number of program steps saved, and thus also computer time saved. Given numerical constants in locations 1 through 50, with a 1 in location 100, the numerical value 50 in location 200 and the value 50 stored in location 300. The problem is to add 1 to each of the 50 constants. Figure 128 shows the problem solved without using index registers. Figure 129 shows the same problem solved with index registers being used. The advantages and flexibility of indexing are readily evident.

| | Location | Instruction | Address |
|---|---|---|---|
| Form constant | 1000 | CLA | 0001 |
| Plus one and | 1001 | ADD | 0100 |
| store | 1002 | STO | 0001 |
| Increase constant | 1003 | CLA | 1000 |
| address and store | 1004 | ADD | 0100 |
| | 1005 | STO | 1000 |
| Reduce the counter | 1006 | CLA | 0300 |
| by one | 1007 | SUB | 0100 |
| Test | 1010 | TNZ | 1000 |
| Stop | 1011 | HLT | |

Figure 128. Address Modification without Indexing

| | Location | Instruction | Address |
|---|---|---|---|
| Set 50 in XRA | 1000 | LXA A | 0200 |
| Constant modification loop and | 1001 | CLA | 0100 |
| | 1002 | ADD A | 0051 |
| store | 1003 | STO A | 0051 |
| Test for equal XRA | 1004 | TIX (1) A | 1001 |
| Stop | 1005 | HLT | |

Figure 129. Address Modification with Indexing

Another programming aid which permits the changing of an instruction's address is indirect addressing. Bits in positions 12 and 13 denote indirect addressing. They are signified in the instruction format by an "F" and in programs and text by an asterisk following the instruction code (CLA*). One additional computer cycle will be taken whenever indirect addressing occurs. During this cycle the word located at the instruction's address is brought out of storage and its address is used to locate the word upon which the instruction operates. This is sometimes called "the second effective address."

As an example of the feature's use, assume that a word has been read into storage by an input-output device. The programmer knows that the command which read in the data is in location 0100. To bring the data back into the accumulator, a portion of the program could be as shown in Figure 130.

The indirect addressing feature may also be combined with indexing, as mentioned above, to obtain a second effective address.

| Location | Instruction | Address | Remarks |
|---|---|---|---|
| 0077 | XXX | XXXX | Previous command |
| 0100 | I-O | ----- | Input-output command |
| | . | . | |
| 0200 | CLA* | 0100 | The CLA* would bring in the data serviced by the I-O instruction even though the address portion is not known. |

Figure 130. Indirect Addressing Example

## Definition of an Assembly Program

An example of an assembly program is one that defines the symbols and their use as follows:

1. The general format of each instruction is: LOCATION, OPERATION, ADDRESS, TAG, DECREMENT.

   Only those instructions that are referred to by other instructions in the program need be given a symbolic location. All other instructions may be written leaving the location field blank. If the tag and decrement fields are not used, they are left blank. In the case of instructions such as CAQ and VLM, the count is placed in the decrement field. Also, for these instructions, if a tag is not required the instruction would be written in the form OPERATION, ADDRESS, O, COUNT.

2. A symbolic address or location can be composed of one to six alphamerical characters, one of which must be non-numerical. For example: TEMP1, GO, HALT, X1 are all allowable symbols. Thus, each symbol can have an important

mnemonic value. Six special characters may not be used in a symbol. They are + – * / , and $. These characters are used for special operations. For example, the plus sign is used for the addition of two or more symbols and/or numbers. Such an operation might be A1 + A2 or HALT + 3.

3. When dealing with a block of data words, only one location in the block need be assigned a symbol. For example, if a block of data words consisted of A1, A2, . . . A75, the location of the first word of the block could be given the symbol AONE. All other words in the block would be related to this point. If A23 were to be referred to it, would be by the symbol AONE + 22.
4. When the actual value of an address, decrement, or count is known, it should be written in absolute form.

When the program has been written it is prepared for assembly by punching each instruction and piece of data into a separate IBM card. These cards are then referred to as symbolic cards.

This symbolic deck is converted to magnetic tape through the card-to-tape equipment and entered, along with the assembly program, into the computer. If desired, the symbolic program may be entered directly into the computer through the on-line card reader.

## Assembly

In the assembly process, the symbolic instructions are processed as follows:

1. The symbolic operation codes are replaced with the actual patterns used by the computer. For example, CLA is replaced by the combination of bits 000 101 000 000 which occupy positions S, 1-11 of the 36-bit instruction word in storage.
2. The absolute location for the first instruction of the program is determined by the programmer and given to the assembly program. Each succeeding instruction and data word is given an absolute location stepped up by one. It is therefore important that the symbolic deck be in the proper order. Each symbolic location detected by the assembly program is entered into a table (called the symbolic table) along with its assigned absolute location. The assembly program then replaces the symbolic address with the absolute locations from the table.

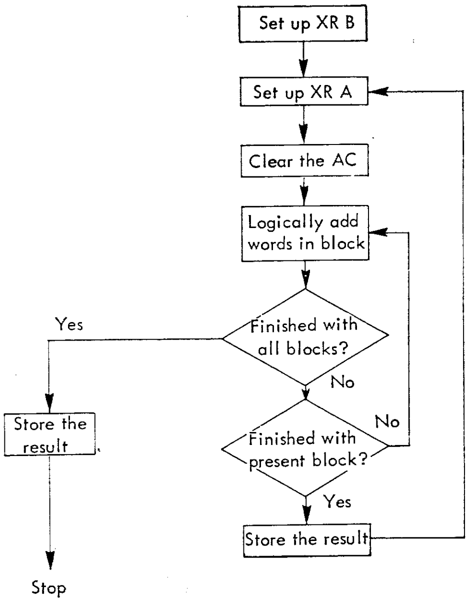
Normally, as a product of the assembly program, a listing of the program in the symbolic format and the actual machine language program is made. In addition, the assembly program furnishes the programmer with a deck of cards containing the machine language program.

## Logical Check Sums

One of the principal methods of keeping a check on a block of information in storage is to attach to this block a sum value of all the words in the block. This sum is called the check sum. The best possible check sum that can be formed is one that is developed using the logical operations of the computer. This check sum is known as a logical check sum. It is normally not equal to the algebraic sum of the block. When using a logical check sum there is no possibility of overflow as in the case of algebraic sums. Furthermore, it does not matter in what direction the words of the block are added. This is not true in algebraic summation where overflow possibilities are affected by the direction of summing. An example of the computing of check sums is shown in Figure 131. The programmer knows that there are five blocks with nine words in each block. The first block starts at location 0601, the second at 0611, the third at 0621, and so on. Location 0500 contains a 9 and location 0501 contains a 49. The problem is to find the logical sum of each block and place it in the first location preceding that block.

The coding of a program instruction normally follows this sequence: (1) the location of the instruction, (2) the instruction mnemonic, (3) the address, if any, (4) the index register, if any (5) the decrement. Thus a TIX, 1, A, 1000 would mean that the TIX has a decrement of 1, index register A is to be used, and the address for the transfer is 1000. The location of the instruction would precede the TIX.

The program shown in Figure 132 will compute the logical check sum for a block of 300 words in core storage. Assume that the 300 words are located in storage in locations 700 through 999. The resulting check sum is to be stored in location 1000. The program uses an index-register-controlled loop to form the logical check sum. The contents of index register 1 are used to effectively modify the address of the instruction in location 102. The index register initially contains the number 300. The final value in the index register will be 1 since the decrement of the TIX instruction is 1. The manner in which the two-instruction loop is performed is as follows:


Set up XR B
Set up XR A
Clear the AC
Logically add words in block
Finished with all blocks?
Yes
No
Store the result
Finished with present block?
No
Yes
Store the result
Stop

| LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | |
|---|---|---|---|
| 0100 | LXA | 501,B | 49 to XRB |
| 0101 | LXA | 500,A | 9 to XRA |
| 0102 | CLM | | Clear the accumulator |
| 0103 | ACL | 650,B | Add the block |
| 0104 | TNX | 110,B,1 | Test all blocks for end |
| 0105 | TIX | 103,A,1 | Reduce count |
| 0106 | SLW | 640,B | Store the check sum for block |
| 0107 | TIX | 101,B,1 | Test for end of block |
| 0110 | SLW | 640 | Store check sum (last one) |
| 0111 | HPR | | Stop |

Figure 131. Computing Check Sum Program and Flow Chart

| LOOP CYCLE | I.R. 1 | EFFECTIVE ADDRESS OF ACL INSTRUCTION |
|---|---|---|
| End of 1st cycle (Before TIX executed) | 300 | ACL 700 |
| End of 2nd cycle (Before TIX executed) | 299 | ACL 701 |
| End of 3rd cycle (Before TIX executed) | 298 | ACL 702 |
| End of 299th cycle (Before TIX executed) | 2 | ACL 998 |
| End of 300th cycle (Before TIX executed) | 1 | ACL 999 |
| End of 300th cycle (After TIX executed) | 1 | Not executed |

Normally a symbolic location is assigned to the block of words. For example, the symbol FIRST could be used to designate the location of the first word of the block. The symbol CKSUM could be used to specify the location where the computed logical check sum is to be stored. The program would then be written as shown in Figure 133.

The number of times the loop is executed is dependent upon the value placed into the index register and the value of the decrement of the TIX instruction. In the preceding example, since XRA contained 300 and the decrement of the TIX instruction is 1, the loop is executed 300 times. If the decrement had been 2, the loop would have been executed 150 times. In this case the logical check sum would have been com-

| LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|
| 100 | AXT | 300, 1 | LOAD 300 INTO INDEX REGISTER 1 | |
| 101 | CLM | | CLEAR ACCUMULATOR (EXCEPT FOR SIGN) | |
| 102 | ACL | 1000, 1 | TWO INSTRUCTION LOOP TO COMPUTE LOGICAL | |
| 103 | TIX | 102, 1, 1 | CHECK SUM. TIX USED TO TEST END OF LOOP | |
| 104 | SLW | 1000 | STORE LOGICAL CHECK SUM IN LOCATION 1000 | |

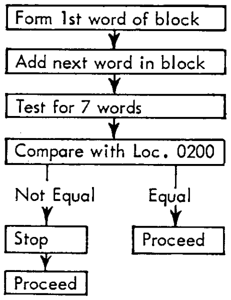
Figure 132. Logical Check Sum Program, Actual

| LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|
| | AXT | 300, 1 | LOAD 300 INTO INDEX REGISTER 1 | |
| | CLM | | CLEAR AC (EXCEPT FOR SIGN) | |
| ADDER | ACL | FIRST + 300, 1 | TWO INSTRUCTION LOOP TO COMPUTE LOGICAL | |
| | TIX | ADDER, 1, 1 | CHECK SUM. TIX USED TO TEST END OF LOOP | |
| | SLW | CKSUM | STORE LOGICAL CHECK SUM IN LOCATION CKSUM | |

Figure 133. Logical Check Sum Program, Symbolic

puted for every other word in the block. Note that at the end of the 300th cycle the index register contained 1. The contents of an index register are never reduced to zero as the result of using a TIX or TNX instruction. The final value found in an index register is dependent on the decrement of the TIX or TNX instruction. If the decrement is the integer *K*, then, depending upon the initial value of the contents of the index register, the final value of the index register can vary in the range *K*, *K*-1, *K*-2, . . . . , 3, 2, 1.

One of the ways check sums could be used is shown in Figure 134. The problem is to find the logical sum of a block of seven numbers starting in location 0100. If the sum does not equal the predetermined amount in location 0200, transfer to an error stop. If it does equal the amount in 0200, proceed with the program. The first check sum (original) is in location 0200, and a 6 is in the address part of location 0006.



```
0000    CAL        0100
0001    LXA,   A   0006
0002    ACL,   A   0107
0003    TIX,1, A   0002
0004    ERA        0200
0005    TZE        0007
0006    HPR        0006
0007    ADD        0300
```

Figure 134. Use Check Sums and Tests

## Drum Copy Loop

Whenever information is transmitted between core storage and magnetic drums, either a CPY or a CAD instruction must be executed for each word transmitted. These instructions specify the sending or receiving location in core storage. The starting location on the drum is specified by a LDA.

The use of these instructions is illustrated by the program in Figure 135. Five hundred consecutive words in core storage are to be transferred to magnetic drum 1. The block of words is in storage starting with location FIRST and is to be copied onto the drum starting at location INITL. The logical check sum for the block of words is stored in location FIRST + 500.

This same program can be used to copy 500 words from the drum into magnetic core storage. The only change necessary is to substitute an RDS for the WRS. The CAD instruction will simultaneously write (or read) each word onto (or from) the drum and logically add it into the accumulator. Thus it is necessary to copy the block check sum by use of the CPY instruction, which only copies the word onto (or from) the drum. The block check sum is then compared with the check sum developed during the transmission of the data between core storage and the drum. This is done through the use of the ERA instruction. This instruction will provide a positive test since any position in the 36-bit words having the same bit (either 1's or 0's) will be set to zero. Thus, two check sums equal to one another will produce a zero word in the AC.

| H | LOCATION 2–6 | 7 | OPERATION 8 | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS 72 | IDENTIFICATION 73–80 |
|---|---|---|---|---|---|---|
| | WRITE | | WRS | 301 | WRITE SELECT DRUM 1 | |
| | | | AXT | 500, 2 | LOAD 500 INTO INDEX REGISTER 2 | |
| | | | LDA | DRADD | LOCATION DRADD CONTAINS FIRST DRUM ADDRESS | |
| | | | CLM | | CLEAR ACCUMULATOR | |
| | CPYLP | | CAD | FIRST + 500, 2 | COPY LOOP. 500 WORDS WRITTEN FROM STORAGE | |
| | | | TIX | CPYLP, 2, 1 | DRUM 1. LOGICAL CHECK SUM COMPUTED | |
| | | | CPY | FIRST + 500 | PREVIOUS CHECK SUM COPIED TO DRUM | |
| | | | ERA | FIRST + 500 | COMPARE PREVIOUS AND NEW CHECK SUMS AND | |
| | | | TZE | OUT | TRANSFER TO LOCATION OUT IF EQUAL | |
| | | | HTR | WRITE | STOP AND THEN REPEAT IF ERROR | |
| | DRADD | | | INITL | STARTING DRUM ADDRESS | |
| | OUT | | | | MAIN PROGRAM CONTINUES HERE | |

Figure 135. Drum Copy Loop

## Packing and Unpacking

There are many cases where the information to be handled by the computer is made up of individual items, each of which is less than the size of a computer word. For example, it may be necessary to work with numbers no larger than three decimal digits. To conserve storage space, three such numbers can be stored in the same word as illustrated in Figure 136, where positions S, 14 and 25 are the sign positions of the numbers $N_1$, $N_2$, and $N_3$, *respectively.* Handling of information in this manner is called "packing." In addition to conserving storage space, packing also increases the entry and exit speed of information by reducing, for instance, the amount of magnetic tape which must be read or written.

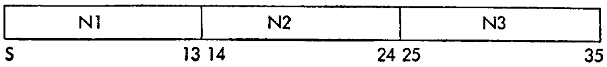


Figure 136. Diagram of Packed Word

Assume that a word in core storage has the form shown in Figure 136, and the number $N2$ is to be operated upon. Before arithmetic operations can be performed with this item, it must be separated from the other data in the word. The method of doing this is called unpacking. The logical operations are employed in this type of operation, as they provide a powerful and flexible tool for carrying out the method. The number $N2$ is to be unpacked from the word without destroying the numbers $N1$ and $N3$. Therefore, the unpacking will be done in the accumulator, saving the packed word in core storage.

The program shown in Figure 137 will accomplish this. The mask used in the program contains 1's in positions 14-24 and 0's elsewhere. The result of using this mask with the ANA instruction will place the number $N2$ in positions 14-24 of the accumulator. By varying the format of the mask, any of the three numbers could have been unpacked (extracted) from the packed word.

| H<br>1 | LOCATION<br>2 6 | 7 | OPERATION<br>8 | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS<br>72 | IDENTIFICATION<br>73 80 |
|---|---|---|---|---|---|---|
| | | | CAL | PAKWD | PLACE PACKED WORD INTO AC POSITIONS P, 1-35 | |
| | | | ANA | MASK | N2 LEFT IN AC AS RESULT OF ANA OPERATION | |
| | | | ALS | 14 | SHIFT N2 UNTIL SIGN OCCUPIES POSITION P | |
| | | | SLW | LOCN2 | STORE N2 IN LOCATION LOCN2 | |
| | | | ⋮ | | | |
| | MASK | | OCT | 000017774000 | MASK CONFIGURATION TO OBTAIN N2 ONLY | |

Figure 137. Unpacking Program

| H | LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|---|
| 1 | 2 6 | 7 8 | | 72 | 73 80 |
| | | CAL | MASK | PLACE MASK IN AC POSITIONS P, 1-35 | |
| | | ANS | PAKWD | ERASE N2 FROM PACKED WORD | |
| | | CAL | LOCN4 | PLACE N4 INTO POSITIONS P, 1-10 OF AC | |
| | | ARS | 14 | SHIFT N4 INTO POSITIONS 14-24 OF AC | |
| | | ORS | PAKWD | INSERT N4 INTO POSITIONS 14-24 OF LOCATION | |
| | | : | | PAKWD. POSITIONS S, 1-13 AND 25-35 UNCHANGED | |
| | MASK | OCT | 777760003777 | MASK TO REMOVE N2 FROM LOCATION PAKWD | |

Figure 138. Packing Program

After performing the desired arithmetic operations on the number *N*2, a new number, *N*4, is the result. This number is the same size as *N*2. Now this new number is to be packed (inserted) in location PADWD replacing *N*2. *N*1 and *N*3 are to remain unchanged. The program in Figure 138 will accomplish this. The program assumes that the number *N*4 occupies positions S, 1-10 of location LOCN4. The mask used with the ANS preserves the numbers *N*1 and *N*3 while replacing N2 with 0's.

Masking may be used to extract a full number or portion of a word from a given location instead of shifting and adjusting the result. Another example is shown in Figure 139 where a number located in positions 12-35 of location 0100 is to be extracted and stored in location 0200. The mask used is located in 0050 and consists of zeros in positions S-11 and ones in positions 12-35.

The program example in Figure 140 shows a number of test instructions, the compare instruction, and some arithmetic operations. The instruction PRINT means that a print routine is being used and the data being printed are denoted by its address.

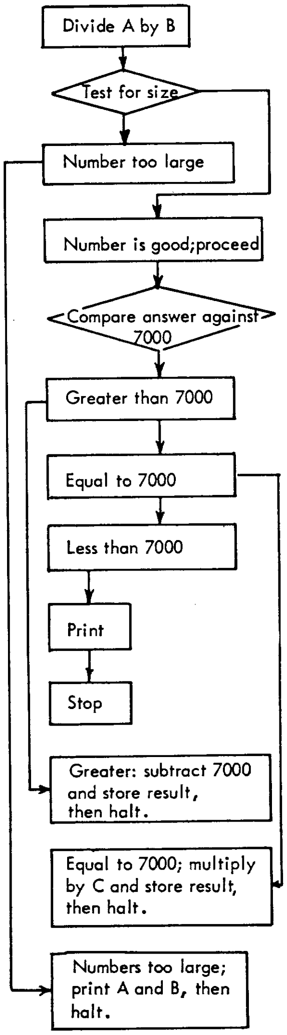
The problem is to divide *A* by *B*. If the computer cannot handle the problem, print both *A* and *B*. If the answer equals 7000, multiply it by *C* and save the answer in location 0400. If it is less than 7000, print the answer. If it is more than 7000 put the difference in location 0500.

The programmer is given *A* in location 0100, *B* in location 0101, *C* in location 0102, and 7000 in location 0103.

Again the flow chart should serve as an aid in the program steps and is a reference when reading the program.

| Location | Instruction | Address | Remarks |
|---|---|---|---|
| 0000 | CLA | 0100 | Put the number in accumulator |
| 0001 | ANA | 0050 | Extract positions 12-35 |
| 0002 | STO | 0200 | Store the result, properly aligned. |
| 0003 | HTR | | |

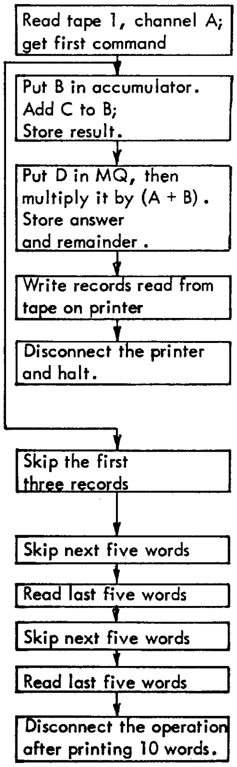
Figure 139. Masking Program


Divide A by B
Test for size
Number too large
Number is good;proceed
Compare answer against 7000
Greater than 7000
Equal to 7000
Less than 7000
Print
Stop
Greater: subtract 7000 and store result, then halt.
Equal to 7000; multiply by C and store result, then halt.
Numbers too large; print A and B, then halt.

```
0000   LDQ    0100
0001   DVP    0101
0002   DCT

0003   TRA    0022

0004   STQ    0105

0005   CLA    0105
0006   CAS    0103

0007   TRA    0014

0010   TRA    0017

0011   STO    0600

0012   Print  0600

0013   HTR

0014   SUB    0103
0015   STO    0500
0016   HTR

0017   MPY    0102
0020   STO    0400
0021   HTR

0022   Print  0100
0023   Print  0101
0024   HTR
```

Figure 140. Program Example

An example of input-output and computing is shown in Figure 141. There are eight binary records on tape unit 1 attached to channel A. Each record contains ten words. The program should: (1) skip the first three records, (2) skip the first five words of the fourth record, (3) read the last five words of that record, (4) skip the first five words of the fifth record, and (5) read the last five words of the fifth record. Put the ten words read into binary print using the printer attached to channel C. Simultaneously with the reading, solve $(B + C) \times D$ and store the result in 0110 and 0111. $B$ is in location 0200, $C$ is in location 0201, and $D$ in location 0202.


Read tape 1, channel A; get first command
Put B in accumulator. Add C to B; Store result.
Put D in MQ, then multiply it by (A + B). Store answer and remainder.
Write records read from tape on printer
Disconnect the printer and halt.
Skip the first three records
Skip next five words
Read last five words
Skip next five words
Read last five words
Disconnect the operation after printing 10 words.

```
0000    RTBA        1221
0001    RCHA        0300

0002    CLA         0200
0003    ADD         0201
0004    STO         0207

0005    LDQ         0202
0006    MPY         0207
0007    STO         0110
0010    STQ         0111
0011    TCOA        0011

0012    WPBC        3362

0013    RCHC        0307
0014    HTR

      Input-Output Program

0300    IORPN       1000
0301    IORPN       1000
0302    IORPN       1000

0303    IOCPN (5)   1000

0304    IOCP (5)    0100

0305    IOCPN (5)   1000

0306    IOCT (5)    0105

0307    IOCD (10)   0100
```

Figure 141. Simultaneous Read, Write and Compute, then Print

## Subroutines

It is very often necessary to repeat the same group of instructions many times during the execution of a program. Examples are the series of instructions necessary for decimal-to-binary conversion, square root, or computing a logical check sum. It is not desirable to write out the necessary instructions each time a function is needed. Instead, the instructions needed are written only once and the main program is then arranged to transfer to this block of instructions each time they are required. Such a block of instructions is called a "subroutine."

These subroutines normally perform such basic functions that they may be used in the solution of many types of problems. For instance, a subroutine which computes a square root can be used in a wide variety of problems. Another example of such a subroutine would be one which computes the logical check sum for a block of words in storage.

Subroutines may be used in two ways with respect to the main program. One method is to insert the subroutine into the main program at the point where it is to be used. Subroutines designed for this type of usage are called "open-subroutines." The open subroutine is "sandwiched" into a program as though it were part of the original coding of the program. This type of subroutine usage is normally restricted to the cases where the main program uses the subroutine only once.

When the main program uses a subroutine several times, which is the common situation, it is apparent that the open subroutine is not desirable. Here, the second method of employing subroutines is used. The subroutine used in these situations is called a "closed subroutine." A closed subroutine may occur several times within one main program, but the set of instructions comprising the subroutine need appear only once. The transfer of control from the main program to the subroutine takes place from a set of instructions known as the calling sequence or basic linkage. The calling sequence transfers control to the subroutine, tells the subroutine where to return to the main program, and gives the subroutine any other information required (Figures 142 and 143).

The subroutine illustrated computes the logical check sum for a block of words in core storage. Three parameters are needed by this subroutine. There are the initial location of the block, the number of words in the block, and the location for storing the resulting check sum. The subroutine then returns control to the main program at the instruction following the last parameter of the calling sequence.

The calling sequence is of the form shown in Figure 142. The subroutine is of the form shown in Figure 143.

The complete transfer of control between the main program and the subroutine is based upon the TSX instruction. As the result of the execution of this instruction, the twos complement of the location LINK is placed in index register 4. From the standpoint of algebraic operation the twos complement of a number is equivalent to the negative of the number. For example, 1 minus (twos complement of LINK) is equivalent to 1 minus (minus LINK) = 1 + LINK.

| H | LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|---|
| 1 | 2 6 | 7 8 |  | 72 | 73 80 |
|  | LINK | TSX | BLKSM, 4 | AUTOMATIC LINKING INSTRUCTION |  |
|  |  |  | FIRST | LOCATION OF FIRST WORD IN BLOCK |  |
|  |  |  | N | NUMBER OF WORDS IN BLOCK |  |
|  |  |  | CKSUM | LOCATION FOR STORING CHECK SUM |  |
|  | --- | --- | ------- | LOCATION TO WHICH CONTROL WILL BE RETURNED |  |

Figure 142. Calling Sequence

| H | LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|---|
| 1 | 2 6 | 7 8 |  | 72 | 73 80 |
|  | BLKSM | CLA | 2, 4 | GET NUMBER OF WORDS IN BLOCK |  |
|  |  | PAX | 0, 1 | PLACE N IN INDEX REGISTER 1 |  |
|  |  | ADD | 1, 4 | ADD LOCATION FIRST TO FORM FIRST +N |  |
|  |  | STA | ADDER | INITIALIZE LOGICAL ADD INSTRUCTION |  |
|  |  | CLA | 3, 4 | GET LOCATION TO STORE CHECK SUM |  |
|  |  | STA | STSUM | PLACE ADDRESS IN STORE INSTRUCTION |  |
|  |  | CLM |  | CLEAR AC |  |
|  | ADDER | ACL | 0, 1 | TWO INSTRUCTION LOOP FOR COMPUTING LOGICAL |  |
|  |  | TIX | ADDER, 1, 1 | CHECK SUM FOR BLOCK OF N WORDS |  |
|  | STSUM | SLW | 0 | STORE CHECK SUM IN LOCATION CKSUM |  |
|  |  | TRA | 4, 4 | RETURN CONTROL TO MAIN PROGRAM |  |

Figure 143. Subroutine to Compute Logical Check Sum

In the subroutine the instructions which make use of this property are:

| INSTRUCTION | EFFECTIVE EXECUTION | EQUIVALENT EXECUTION |
|---|---|---|
| CLA 2,4 | CLA 2 — (2's comp. LINK) | CLA LINK + 2 |
| ADD 1,4 | ADD 1 — (2's comp. LINK) | ADD LINK + 1 |
| CLA 3,4 | CLA 3 — (2's comp. LINK) | CLA LINK + 3 |
| TRA 4,4 | TRA 4 — (2's comp. LINK) | TRA LINK + 4 |

From the above table it can be seen that the subroutine will be able to make use of the information found in the parameter locations of the calling sequence without knowledge of their exact location in storage. Since LINK is a symbol representing any location in core storage, the subroutine can thus communicate with the main program at any location in the main program. By means of the TRA 4,4 instruction, the subroutine has the ability to transfer control back to the proper location in the main program.

One of the main responsibilities of a subroutine is to insure that when control is transferred back to the main program the status of all the registers is the same as when control was transferred to the subroutine. This does not apply, of course, to a subroutine designed specifically to change a machine condition. For example, in the previous illustration the contents of index register 1 are destroyed by the subroutine. Thus, when control is transferred back to the main program, index register 1 may not be the same as when control was transferred to the subroutine. The contents of index register 1 may be preserved by adding the instruction SXA SAVE, 1 just after the instruction CLA 2, 4. The contents of XR 1 will be stored in the address part of location SAVE. Now, if location SAVE is inserted just before the TRA 4,4 instruction and contains the instruction AXT 0,1, the original contents of XR 1 will be replaced just before control is transferred back to the main program.

## Convert Instructions

Three convert instructions are available in the computer. During their execution these instructions use, in addition to core storage, the accumulator, multiplier-quotient, and storage registers. Index register 1 may also be used, if desired, to receive information at the conclusion of a convert instruction execution. These instructions normally work with tables stored in core storage.

These convert instructions provide the programmer with a rapid means of performing such operations as

BCD-to-binary and binary-to-BCD conversion, BCD arithmetic, editing of records, and modification of collating sequences.

When the convert instructions are used, either the AC or the MQ contains a 36-bit word that is divided into six 6-bit binary numbers. Each 6-bit number (sometimes referred to as a character) is treated separately in consecutive order by the convert instructions. For the instructions CRQ and CVR, this 36-bit word represents the actual word operated upon. The CVR examines the word six bits at a time from right to left, while CRQ examines the word six bits at a time from left to right. For the CAQ, the contents of the MQ are examined six bits at a time from left to right while addition of quantities found in core storage, as determined by this word in the MQ, takes place in the AC.

The problem of replacing the leading zeros of a BCD number with blanks is reduced to a short rapid program through the use of the CRQ instruction. This particular convert instruction is used because, to remove leading zeros, the BCD number must be tested from left to right.

To carry out the editing (modification) of the BCD number, it is necessary to set up a table in core storage. This table has the following format:

| LOCATION | CONTENTS S,1 – 5 | 21 – 35 |
|---|---|---|
| A | BCD blank (b) | A |
| A + 1 | BCD one | A + 10 |
| A + 2 | BCD two | A + 10 |
| . | . | . |
| . | . | . |
| . | . | . |
| A + K | BCD (K) | A + 10 |
| . | . | . |
| . | . | . |
| . | . | . |
| A + 9 | BCD nine | A + 10 |
| A + 10 | BCD zero | A + 10 |
| A + 11 | BCD one | A + 10 |
| A + 12 | BCD two | A + 10 |
| . | . | . |
| . | . | . |
| . | . | . |
| A + 19 | BCD nine | A + 10 |

The program shown in Figure 144 will perform the required editing operation on a BCD number of 12 digits occupying two consecutive core storage locations. The program must consider the following cases:

1. All leading zeros must be sensed and replaced by blank characters.
2. All non-zero digits must be preserved.
3. Once a non-zero digit is found, all succeeding zeros must be preserved.
4. If a non-zero digit is found in the high-order six digits, the second half of the number need not be processed.

The program is executed as follows: The high-order six digits of the BCD number are placed in the MQ by the LDQ BCD1 instruction. The first table reference made by the CRQ A,1,6 instruction will be at location A + *N*, where *N* is the high-order digit in the MQ, that is, $c(MQ)_{S,1\text{-}5}$. If *N* is zero, the CRQ instruction will go to table location A and from this location will replace the zero with the BCD blank character. Since the address part of location A contains A, the second table reference will begin at location A of the table. Once a non-zero digit occurs, the CRQ instruction will make a table reference at location A + *K*, where *K* is the non-zero digit. Location A + *K* contains *K* in positions S,1-5 and A + 10 in the address part. Thus, the value *K* will replace the number *K* and this insures that the non-zero digits will be preserved. Once the first non-zero digit is found, only the second part of the table, location A + 10 through A + 19, is used. This insures that zeros following the first non-zero will be replaced with zeros instead of blanks. Figure 145 illustrates the execution of the convert instruction using the BCD number 000307.

When all six of the BCD digits have been tested (the count reduced to zero), the execution of the CRQ instruction is terminated. Since the instruction contains a tag of one, the address part of the last table reference location will be placed in index register 1. After

| H 1 | LOCATION 2–6 | 7 | OPERATION 8 | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS 72 | IDENTIFICATION 73–80 |
|---|---|---|---|---|---|---|
| | START | | LDQ | BCD1 | LOAD HIGH ORDER SIX DIGITS INTO MQ | |
| | | | CRQ | A,1,6 | EDIT HIGH ORDER SIX DIGITS | |
| | | | STQ | BCD1 | STORE EDITED DIGITS | |
| | | | TXH | OUT,1,A | TEST FOR NON-ZERO THIS HALF | |
| | | | LDQ | BCD2 | LOAD LOW ORDER SIX DIGITS INTO MQ | |
| | | | CRQ | A,0,6 | EDIT LOW ORDER SIX DIGITS | |
| | | | STQ | BCD2 | STORE EDITED LOW ORDER DIGITS | |
| | OUT | | --- | -------- | PROGRAM CONTINUES HERE | |

Figure 144. Edit Program

storing the edited BCD number (STQ BCD1) the contents of index register 1 are then compared with the number A by the TXH OUT,1,A instruction. If the index register contains A, then all six of the high-order BCD digits were zero and the program will continue to examine the remaining digits in the number. If the index register contains A + 10, this indicater that a non-zero digit was found in the high-order six digits. Thus, the low-order digits need not be processed and control is transferred to location OUT.

The convert instruction CVR can be used to perform BCD addition without having to rely on a complex logical routine. The CVR instruction is used in this application since it is necessary to process the decimal sum from right to left to provide for carries from one position to the next. The program which will add two 6-digit unsigned BCD numbers is shown in Figure 146. This program insures that the following conditions are satisfied.

1. Any position of the sum that does not produce a carry must be preserved.
2. Any position of the sum that does produce a carry must be modified with the carry propagated to the next position.
3. A test must be made to determine whether or not a carry occurs out of the high-order position. If such a carry does occur, a one must be placed in the next highest word location.

| C(MQ) | | | | | | Count | C(SR) | | C(MQ) + C(SR) = X | | | C(X) | | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S,1-5 | 6-11 | 12-16 | 17-23 | 24-29 | 30-35 | | S,1-5 | 21-35 | S,1-5 | 21-35 | | S,1-5 | 21-35 | |
| 0 | 0 | 0 | 3 | 0 | 7 | 6 | — | A | 0 | A | A+0 | b | A | Start cycle 1 |
| 0 | 0 | 3 | 0 | 7 | b | 5 | b | A | | | | | | End cycle 1 |
| 0 | 0 | 3 | 0 | 7 | b | 5 | b | A | 0 | A | A+0 | b | A | Start cycle 2 |
| 0 | 3 | 0 | 7 | b | b | 4 | b | A | | | | | | End cycle 2 |
| 0 | 3 | 0 | 7 | b | b | 4 | b | A | 0 | A | A+0 | b | A | Start cycle 3 |
| 3 | 0 | 7 | b | b | b | 3 | b | A | | | | | | End cycle 3 |
| 3 | 0 | 7 | b | b | b | 3 | b | A | 3 | A | A+3 | 3 | A+10 | Start cycle 4 |
| 0 | 7 | b | b | b | 3 | 2 | 3 | A+10 | | | | | | End cycle 4 |
| 0 | 7 | b | b | b | 3 | 2 | 3 | A+10 | 0 | A+10 | A+10 | 0 | A+10 | Start cycle 5 |
| 7 | b | b | b | 3 | 0 | 1 | 0 | A+10 | | | | | | End cycle 5 |
| 7 | b | b | b | 3 | 0 | 1 | 0 | A+10 | 7 | A+10 | A+17 | 7 | A+10 | Start cycle 6 |
| b | b | b | 3 | 0 | 7 | 0 | 7 | A+10 | | | | | | End cycle 6 |

Figure 145. Execution of CRQ

```
H  LOCATION     OPERATION  ADDRESS, TAG, DECREMENT/COUNT        COMMENTS                IDENTI-
                                                                                       FICATION
1 2        6 7 8                                                                   72 73      80
                CAL        DEC1          FIRST UNSIGNED BCD NUMBER TO AC
                ADD        DEC2          ADD SECOND BCD NUMBER, SUM IN AC
                CVR        A,1,6         REPLACE VALUES FOR WHICH CARRIES OCCURED
                SLW        SUM           STORE SUM
                TXL        OUT,1,A       TEST FOR HIGH ORDER CARRY
                CLA        LOCONE        CARRY OCCUPIED, PLACE BCD ONE IN AC
                STO        SUM + 1       PLACE HIGH ORDER CARRY IN NEXT LOCATION
   OUT          ----       --------      PROGRAM CONTINUES HERE
                 .
                 .
   LOCONE       HTR        1
```

Figure 146. BCD Addition Program

| LOCATION | BCD CHARACTER POSITIONS S-5 | NEXT TABLE REFERENCE POSITIONS 21-35 |
|---|---|---|
| A | 0 | A |
| A + 1 | 1 | A |
| A + 2 | 2 | A |
| . | . | . |
| . | . | . |
| A + 9 | 9 | A |
| A + 10 | 0 | A + 1 |
| A + 11 | 1 | A + 1 |
| A + 12 | 2 | A + 1 |
| . | . | . |
| . | . | . |
| A + 19 | 9 | A + 1 |

Figure 147. Table for BCD Addition

The convert instruction CVR used in the program uses the table found in Figure 147.

The execution of this program can best be illustrated by the following example. The two BCD unsigned numbers 434589 and 691593 are to be added. The resulting sum is considered as six 6-bit numbers (Figure 148). The low-order 6-bit number has the value 12. Thus, the first table reference made by the CVR A,1,6 instruction is at location A + 12. Positions S,1-5 of this location contain a BCD 2 which replaces the number 12 in the AC. Positions 21-35 of this location contain the number A + 1. The address A + 1 causes the next table reference to be made at location A + 18 rather than A + 17. Thus, a carry of one is propagated from the units to the tens position of the sum. The execution of the CVR will end when the count has been reduced to zero. Since this instruction has a tag of one, the contents of the storage register

| Location | Binary equivalent of BCD Digit. Positions S,1-19 | Next table location Positions 21-35 |
|---|---|---|
| A | 0 | B |
| A + 1 | 1 x $10^5$ | B |
| A + 2 | 2 x $10^5$ | B |
| . | . | . |
| . | . | . |
| A + 9 | 9 x $10^5$ | B |
| B | 0 | C |
| B + 1 | 1 x $10^4$ | C |
| B + 2 | 2 x $10^4$ | C |
| . | . | . |
| . | . | . |
| B + 9 | 9 x $10^4$ | C |
| C | 0 | D |
| C + 1 | 1 x $10^3$ | D |
| C + 2 | 2 x $10^3$ | D |
| . | . | . |
| . | . | . |
| C + 9 | 9 x $10^3$ | D |
| D | 0 | E |
| D + 1 | 1 x $10^2$ | E |
| D + 2 | 2 x $10^2$ | E |
| . | . | . |
| . | . | . |
| D + 9 | 9 x $10^2$ | E |
| E | 0 | F |
| E + 1 | 1 x 10 | F |
| E + 2 | 2 x 10 | F |
| . | . | . |
| . | . | . |
| E + 9 | 9 x 10 | F |
| F | 0 | 0 |
| F + 1 | 1 | 0 |
| F + 2 | 2 | 0 |
| . | . | . |
| . | . | . |
| F + 9 | 9 | 0 |

Figure 149. Table for BCD-to-Binary Conversion

| | | ACCUMULATOR CONTENTS | | | | | | STORAGE REGISTER | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | START OF CYCLE | | END OF CYCLE | |
| INSTRUCTION | COUNT | P - 5 | 6-11 | 12-17 | 18-23 | 24-29 | 30-35 | S - 5 | 21 - 35 | S - 5 | 21 - 35 |
| CAL DEC1 | | 4 | 3 | 4 | 5 | 8 | 9 | | | | |
| ADD DEC2 | | 10 | 12 | 5 | 10 | 17 | 12 | | | | |
| CVR A,1,6 | 6 | 10 | 12 | 5 | 10 | 17 | 12 | — | A+12 | 2 | A+1 |
| | 5 | 2 | 10 | 12 | 5 | 10 | 17 | 2 | A+1+17 | 8 | A+1 |
| | 4 | 8 | 2 | 10 | 12 | 5 | 10 | 8 | A+1+10 | 1 | A+1 |
| | 3 | 1 | 8 | 2 | 10 | 12 | 5 | 1 | A+1+5 | 6 | A |
| | 2 | 6 | 1 | 8 | 2 | 10 | 12 | 6 | A+12 | 2 | A+1 |
| | 1 | 2 | 6 | 1 | 8 | 2 | 10 | 2 | A+1+10 | 1 | A+1 |
| | 0 | 1 | 2 | 6 | 1 | 8 | 2 | 1 | A+1 | | |

Figure 148. Execution of CVR

positions 21-35 (A + 1) will be placed in index register 1. The TXL OUT,1,A instruction then tests the contents of this index register. Since A + 1 is greater than A, the program continues and a BCD 1 is placed in positions 30-35 of location SUM + 1. Had index register 1 contained A, the program would have transferred control to location OUT.

Conversion from one number system to another may be performed by the CAQ convert instruction. An example of BCD-to-binary conversion is illustrated here. The program which performs this conversion is based upon the fact that a BCD number (e.g., 803157) is really a sum of terms of the form:

$$8 \times 10^5 + 0 \times 10^4 + 3 \times 10^3$$
$$+ 1 \times 10^2 + 5 \times 10 + 7.$$

The binary number equivalent to a BCD number is obtained simply by finding the sum of the binary equivalents of each term. For this example, the binary equivalent of $8 \times 10^5$ plus the binary equivalent of $0 \times 10^4$ plus . . . plus the binary equivalent of 7. The table used with this program is thus divided into six parts (Figure 149). Each section consists of ten words, one word for each of the digits 0-9 multiplied by a power of ten. The binary equivalent of each BCD digit is contained in the twenty positions S-19 of each word in the table. This is based on the fact that only 20 binary positions are necessary to represent a 6-digit BCD number.

The CAQ instruction uses a straightforward table look-up operation to build up the binary equivalent of the BCD number, digit by digit. Each digit of the BCD number is employed to look up its binary equivalent from one of the six parts of the table. The first BCD digit will choose its binary equivalent from the first section ($10^5$) of the table; the second BCD digit will choose its equivalent from the second section of the table; and so forth. This operation is continued until the complete binary equivalent is formed in positions P-19 of the AC. Caution must be taken to choose table locations so that the sum of the locations (as illustrated in Figure 151) will not overflow into the resulting binary number portion of the AC.

The program in Figure 150 will perform the conversion operation.

The execution of the CAQ A,0,6 instruction is illustrated in Figure 151. As can be seen, the sum of the table locations B,C, . . .,F must not form a sum that will overflow into the binary portion of the AC.

| H | LOCATION 2-6 | 7 | OPERATION 8 | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS 72 | IDENTIFICATION 73-80 |
|---|---|---|---|---|---|---|
| | | | LDQ | BCDWD | LOAD BCD WORD INTO MQ | |
| | | | CLM | | CLEAR AC (EXCEPT FOR SIGN) | |
| | | | CAQ | A, 0, 6 | CONVERT BCD TO BINARY | |
| | | | ARS | 16 | SHIFT BINARY RESULT TO PROPER POSITION | |
| | | | SLW | BINWD | STORE BINARY RESULT | |
| | | | | | | |

Figure 150. BCD-to-Binary Conversion Program

| INSTRUCTION | COUNT | ACCUMULATOR CONTENTS (Binary equivalent) P,1 - 19 | 21 - 35 | MQ CONTENTS | STORAGE REGISTER START OF CYCLE S - 19 | 21-35 | END OF CYCLE S - 19 | 21 - 35 |
|---|---|---|---|---|---|---|---|---|
| LDQ BCDWD | | | | 803157 | | | | |
| CLM | | 0000.....................000 | 00......000 | 803157 | | | | |
| CAQ A,0,6 | 6 | | | | — | A + 8 | $8\times10^5$ | B |
| | 5 | $8\times10^5$ | B | 031578 | $8\times10^5$ | B + 0 | 0 | C |
| | 4 | $8\times10^5+0$ | B+C | 315780 | 0 | C + 3 | $3\times10^3$ | D |
| | 3 | $8\times10^5+0+3\times10^3$ | B+C+D | 157803 | $3\times10^3$ | D + 1 | $1\times10^2$ | E |
| | 2 | $8\times10^5+0+3\times10^3+1\times10^2$ | B+C+D+E | 578031 | $1\times10^2$ | E + 5 | 5x10 | F |
| | 1 | $8\times10^5+0+3\times10^3+1\times10^2+5\times10$ | B+C+D+E+F | 780315 | 5x10 | F + 7 | 7 | 0 |
| | 0 | $8\times10^5+0+3\times10^3+1\times10^2+5\times10+7$ | B+C+D+E+F | 803157 | 7 | 0 | | |

Figure 151. Execution of CAQ

## Sense Indicators

In many applications of data-processing machines it is desirable to have switches which can be set and tested by the program. A special register, the sense indicator register, provides 36 such devices. Each of these can be turned on or off (zero or one). The individual positions are called sense indicators.

Either singly or in groups these indicators can be turned on or set (set to ones) or turned off or reset (set to zeros). These indicators can be used as program-controlled sense switches, sense lights, or selectors. In addition, they are also useful in extracting or inserting 'parts of words and for testing fields within a word.

A program is often written that deals with a problem having several variations. In cases of this type, one way of developing the general program is to construct the program so that it is composed of self-contained sections. Control in the general program is then transferred from section to section in the sequence determined by the particular variation being solved. The sense indicators may be used to direct and monitor the desired sequence of control.

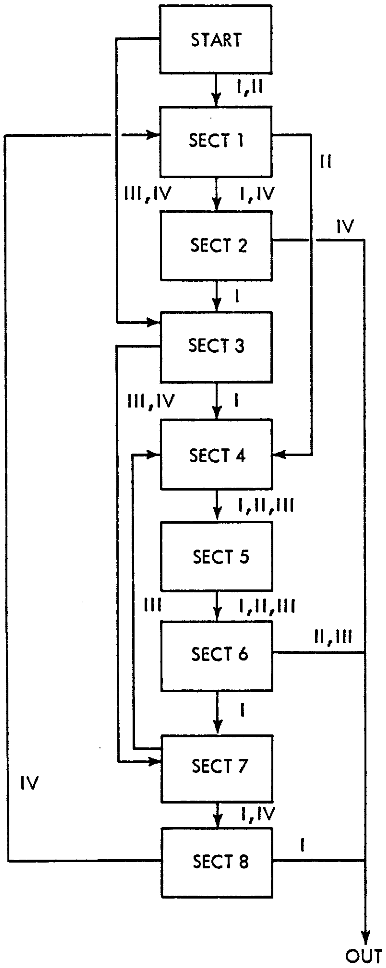


Figure 152. Block Diagram of General Program

For example, a general program is composed of eight sections and is to deal with a problem having four variations. The eight sections and the four variations are illustrated in Figure 152. The sequence of control necessary for each variation and the representation of the sense indicator bits needed for the direction of this sequence are shown in Figure 153.

| VARIATION | SEQUENCE OF CONTROL BY PROGRAM SECTION | SENSE INDICATOR REGISTER POSITIONS 32 | 33 | 34 | 35 |
|---|---|---|---|---|---|
| I | 1-2-3-4-5-6-7-8 | 0 | 0 | 0 | 1 |
| II | 1-4-5-6 | 0 | 0 | 1 | 0 |
| III | 3-7-4-5-6 | 0 | 1 | 0 | 0 |
| IV | 3-7-8-1-2 | 1 | 0 | 0 | 0 |

Figure 153. Sense Indicator Pattern

Four control words are used to contain the different bit patterns to be used in the sense indicator register for the four different problem variations (Figure 154). The general program is started by loading the desired control word into the sense indicator register.

| LOCATIONS | CONTROL WORD | REMARKS |
|---|---|---|
| VARI | Oct 1 | Variation 1 bit pattern for the SI register |
| VARII | Oct 2 | Variation 2 bit pattern for the SI register |
| VARIII | Oct 4 | Variation 3 bit pattern for the SI register |
| VARIV | Oct 10 | Variation 4 bit pattern for the SI register |

Figure 154. Problem Variations

The instructions in the general program that are necessary for the direction of control are shown in Figure 155.

Both the RFT and RNT instructions contain masks (in octal) in positions 18-35. These 18 bits are compared with the right-most (positions 18-35) 18 bits of the sense indicator register. The instruction LFT and LNT with the same masks could have been used in place of the RFT and RNT instructions. However, since these instructions compare with the left-most

| H | LOCATION |  | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTI-FICATION |
|---|---|---|---|---|---|---|
| 1 | 2 6 | 7 | 8 |  | 72 | 73 80 |
|  | START |  | LDI | CONWD | LOAD CONTROL WORD FOR DESIRED SEQUENCE |  |
|  |  |  |  |  |  |  |
|  |  |  | RFT | 14 | INITIAL SEQUENCE TEST. SI POSITIONS 32 AND 33 |  |
|  |  |  | TRA | SECT3 | TESTED FOR ZEROS. IF ZEROS CONTROL TO SECT1. IF |  |
|  | SECT1 |  |  |  | NOT CONTROL TO SECT 3 |  |
|  |  |  |  |  |  |  |
|  |  |  | RFT | 2 | SECTION 1 END-SEQUENCE TEST. TEST POSITION 34. |  |
|  |  |  | TRA | SECT4 | IF ZERO CONTROL GOES TO SECTION 2. IF ONE, |  |
|  | SECT2 |  |  |  | CONTROL TO SECTION 4. |  |
|  |  |  |  |  |  |  |
|  |  |  | RFT | 10 | SECTION 2 END-SEQUENCE TEST. TEST POSITION 32. |  |
|  |  |  | TRA | OUT | IF ZERO CONTROL GOES TO SECTION 3. IF ONE, END- |  |
|  | SECT3 |  |  |  | OF-PROGRAM. |  |
|  |  |  |  |  |  |  |
|  |  |  | RFT | 14 | SECTION 3 END-SEQUENCE TEST. TEST POSITIONS 32 |  |
|  |  |  | TRA | SECT7 | AND 33. IF ZERO, CONTROL GOES TO SECTION 4. IF |  |
|  | SECT4 |  |  |  | ONES, CONTROL TO SECTION 7. |  |
|  |  |  |  |  |  |  |
|  | SECT5 |  |  |  | CONTROL ALWAYS GOES FROM SECTION 4 TO SECTION |  |
|  |  |  |  |  | 5. |  |
|  |  |  |  |  |  |  |
|  | SECT6 |  |  |  | CONTROL ALWAYS GOES FROM SECTION 5 TO SECTION |  |
|  |  |  |  |  | 6. |  |
|  |  |  |  |  |  |  |
|  |  |  | RNT | 1 | SECTION 6 END-SEQUENCE TEST. TEST POSITION 35. |  |
|  |  |  | TRA | OUT | IF ONE, CONTROL GOES TO SECTION 7. IF ZERO, END- |  |
|  | SECT7 |  |  |  | OF-PROGRAM. |  |
|  |  |  |  |  |  |  |
|  |  |  | RFT | 4 | SECTION 7 END-SEQUENCE TEST. TEST POSITION 33. |  |
|  |  |  | TRA | SECT4 | IF ZERO, CONTROL GOES TO SECTION 8. IF ONE, |  |
|  | SECT8 |  |  |  | CONTROL TO SECTION 4. |  |
|  |  |  |  |  |  |  |
|  |  |  | RFT | 10 | SECTION 8 END-SEQUENCE TEST. TEST POSITION 32. |  |
|  |  |  | TRA | SECT1 | IF ONE, TRANSFER CONTROL TO SECTION 1. IF ZERO, |  |
|  | OUT |  |  |  | END-OF PROGRAM. |  |

Figure 155. Control Instructions in Program

18 positions (positions 0-17) of the sense indicator register, the four commands must be changed so that the four SI positions concerned are positions 14-17.

It may be necessary to unpack a word without affecting the accumulator or the multiplier-quotient register. In this case, the sense indicator register may be used for the unpacking operation. For example, to unpack a number occupying positions 14-24 of a packed word, a mask containing ones in positions S,1-13,25-35 is used to perform the extraction.

Figure 156 is the program which is used to execute the extracting. The packed word is loaded into the SI register by the LDI instruction. With the execution of the RIS instruction all the positions in the SI corresponding to the ones in the mask are set to zero. Thus, as a result of the mask used, the desired number in positions 14-24 is left intact and the remaining positions are set to zero. The number extracted from the packed word is simply dependent on the mask used.

A companion operation to the extracting described above is that of insertion. This may also be accomplished by using the SI register. An example is inserting a new number in positions 14-24 of the packed word of the last example. The new number to be inserted occupies positions 14-24 of the AC. The program is shown in Figure 157. The packed word is loaded into the SI by the LDI instruction. The mask used by the RIS instruction sets positions 14-24 of the SI to zero. The remaining positions are unchanged. The new number is then "OR'ed" into the packed word by the OAI instruction.

## Floating Point Overflow and Underflow

During many scientific and engineering problems, the programmer is faced with the difficulty of keeping track of the decimal point. To aid in this respect, the computer is equipped with a complete set of floating point instructions. Briefly, a floating point number is treated as a 27-bit signed proper fraction and an 8-bit characteristic which represents a signed exponent.

By the nature of floating point operation, the fraction may never overflow the registers, and an underflow of the fraction produces a normal zero which is a proper result. The characteristic enjoys no such freedom. A floating point operation resulting in a characteristic, either too large or too small for that portion of the word set aside for it, produces a condition known as *floating point overflow or underflow,* respectively. These conditions are referred to collectively as *floating point spill.*

When floating point spill occurs, the factors must be scaled to fall within the range of the computer registers if calculation is to continue. The exact details of this scaling usually depend upon the conditions of the problem. However, the computer provides adequate facilities to assist the programmer in deciding what these conditions are and for controlling the corrective process.

The computer may be operated in two modes with regard to floating point operation. Normally, the computer operates in *floating point trap mode;* that is, the floating point trap device is normally on. This device is referred to as "FPT." If the instruction leave floating trap mode (LFTM) is executed, the computer operates in what is called the 704 floating point trap mode.

### IBM 704 Floating Point Trap Mode

To enter the 704 floating point trap mode, the instruction LFTM must be executed. It is necessary be-

| H 1 | LOCATION 2 6 | 7 | OPERATION 8 | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS 72 | IDENTIFICATION 73 80 |
|---|---|---|---|---|---|---|
| | | | LDI | PAKWD | LOAD PACKED WORD INTO SI | |
| | | | RIS | MASK | MASK TO PRESERVE POSITIONS 14-24 OF SI | |
| | | | STI | UNPAK | STORE UNPACKED NUMBER INTO STORAGE | |
| | | | : | | | |
| | MASK | | OCT | 777760003777 | MASK TO OBTAIN POSITIONS 14-24 | |

Figure 156. Extraction Program Using Sense Indicator

| H 1 | LOCATION 2 6 | 7 | OPERATION 8 | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS 72 | IDENTIFICATION 73 80 |
|---|---|---|---|---|---|---|
| | | | LDI | PAKWD | LOAD PACKED WORD INTO SI | |
| | | | RIS | MASK | MASK TO CLEAR POSITIONS 14-24 | |
| | | | OAI | | OR NEW NUMBER INTO POSITIONS 14-24 | |
| | | | STI | PAKWD | STORE NEW PACKED WORD | |
| | | | : | | | |
| | MASK | | OCT | 000017774000 | MASK TO CLEAR POSITIONS 14-24 | |

Figure 157. Insertion Program Using Sense Indicator

cause FPT is normal in the computer. If a floating point spill occurs while in the 704 mode, the accumulator and/or the MQ overflow indicators are turned on. Which indicator is set is determined by the register producing the spill. Each indicator may be tested by the program, subsequent to the spill, by executing the proper overflow test instruction. The mathematics involved in the following examples are specialized to point out how the logical facilities of the computer might be employed to detect and to control the correction of floating point spill.

### Floating Point Spill in the 704 Mode

This problem is defined below.

A body of surface area (SURA) is being bombarded by particles of some nature. The researcher has measured the number of impacts (IMP) on the surface at N discreet time intervals. At a certain point in the calculations, the programmer wishes to know if the following conditions exist: $Q < MAX$, where MAX is a constant and Q is the average number of impacts per unit area. If the inequality holds, the program is to continue at location OK. Otherwise, program control is to be transferred to symbolic location TOBIG.

All the $IMP_n$ (total impacts in the N time intervals) are written within the range $0 \leq IMP_n < 10^{45}$. The $IMP_n$ are stored at symbolic locations IMP to IMP + N − 1. The values N, MAX, and SURA are known. The original values of $IMP_n$ must remain intact. The first step is to calculate the total impacts. If a spill occurs, the $IMP_n$ are scaled by $2^{-100}$, and the summation repeats. If another spill occurs, it is treated as an irretrievable error. Note that MQ spill has no effect on this summation. The program is shown in Figure 158 and executed as described below:

*Line 1.* LFTM must be given to enter the 704 mode. The accumulator overflow indicator is then turned off by the TOV instruction; this is necessary because its condition cannot be assumed.

*Line 6.* The $IMP_n$ is brought to the accumulator. Because of the method of scaling used, zero words are skipped over.

*Line 8.* Because spill detection is most useful if the first spill is detected, scaling is accomplished without using floating point. If XRA is zero, scaling is not required and the program proceeds to line 11. If XRA is not zero, scaling is required. A word containing an octal 100 in the characteristic field is fixed-point subtracted from the $IMP_n$. This is equivalent to floating point division by $2^{100}$. If the accumulator goes minus, the $IMP_n$ is not within the given range. If $IMP_n$ is not in this range, control is transferred to 1ERR.

*Line 11.* The partial sum is added to the $IMP_n$ in the accumulator. If spill does not occur, the program proceeds with the summation; if spill does occur, XRA is set to 1 and the summation is restarted. If scaling has already occurred, an error is indicated and control transfers to 2ERR.

*Line 18.* The MQ overflow and the divide check indicators are reset to the off position, because their condition cannot be assumed. The FDP is then executed and both the divide check and MQ overflow indicators are tested (MQ overflow or underflow is defined as a TOBIG condition).

*Line 27.* With the quotient in the accumulator and the condition of the accumulator overflow indicator established (lines 25 and 26), a test to find if scaling has taken place is executed. If so, the word subtracted in line 9 is added back to increase the characteristic by 100, which is equivalent to floating point multiplication by $2^{100}$. An overflow at this point (fixed point overflow) indicates that Q is too big. If scaling has not occurred, control transfers.

*Line 30.* MAX is then subtracted from the quotient. A spill at this point is not significant; the sign of the result establishes the range of the number. If the accumulator is minus, the program proceeds to symbolic location OK, where FPT is reset to the on position and the main program may resume.

### Floating Point Trap

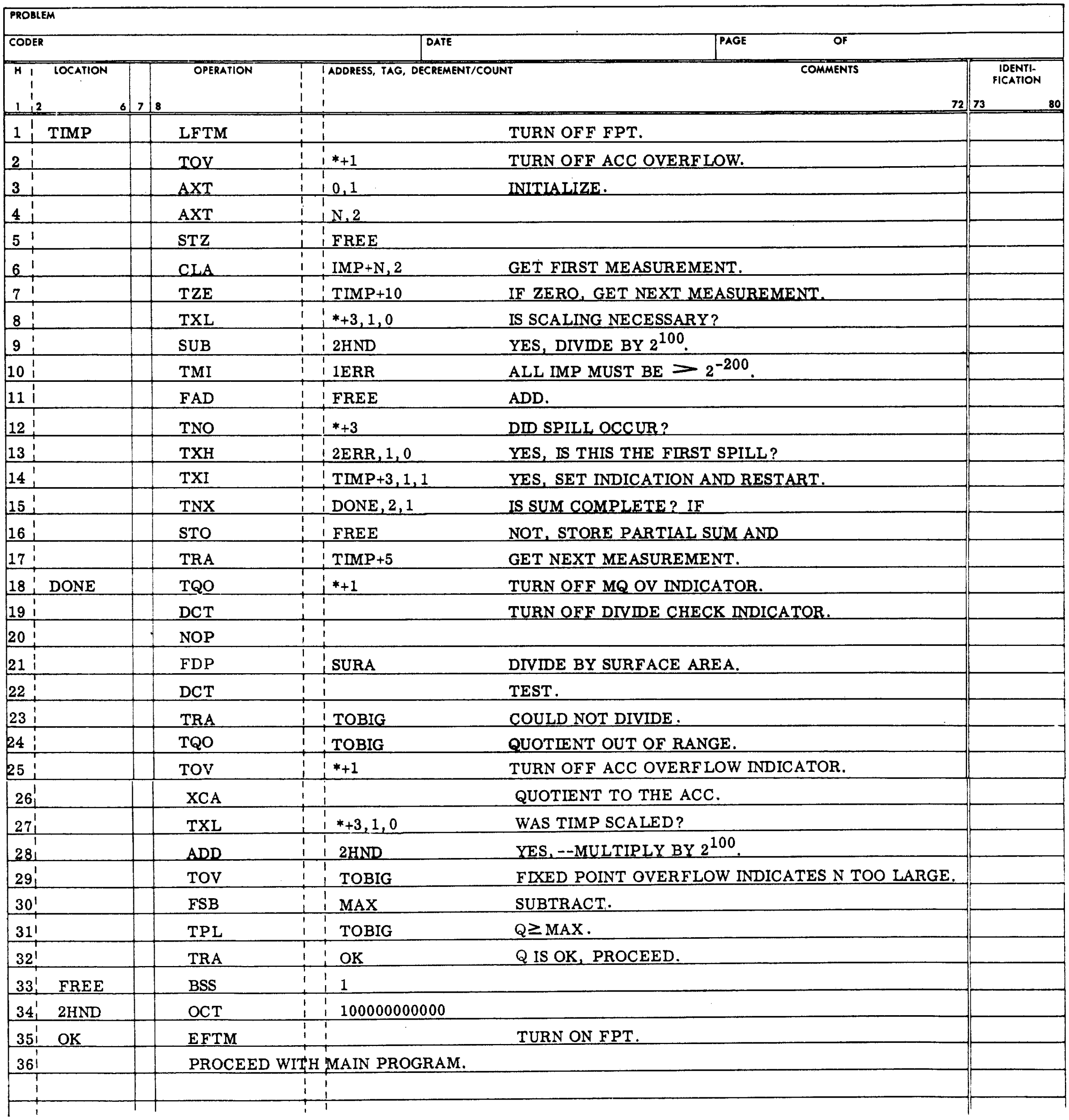
Figure 158 illustrates how the interpretation of spill depends upon the conditions prevailing when the spill is detected. In general, the program should: (1) detect the spill as soon as it occurs, (2) know in which register the spill occurred, and (3) know what instruction was being executed when the spill occurred. This information can be provided automatically by the floating point trap.

When spill occurs with the FPT on, the computer automatically performs the following steps:

1. The address plus 1 of the instruction causing spill is placed in the address field of core location 0000.
2. A four-bit code which identifies the nature of the spill is placed in positions 14 through 17 of core location 0000.
3. The computer takes its next instruction from location 0010 and proceeds from there.

To illustrate a use of FPT, the problem used in Figure 158 is repeated using the floating-point-trap feature. Note that, although the program is larger, the possibilities for control are increased. Figure 159 shows that the trap routine starting at location 0010 can conditionally return to the main program or initiate a general operation (in this case, a print program and then halt).

The practice of continually storing certain addresses from the main program in a standard routine is sometimes referred to as "breakpoint" programming; as Figure 159 shows, it is an extremely powerful and



PROBLEM

CODER | DATE | PAGE OF

| H | LOCATION | | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|---|---|
| 1 | TIMP | | LFTM | | TURN OFF FPT. | |
| 2 | | | TOV | *+1 | TURN OFF ACC OVERFLOW. | |
| 3 | | | AXT | 0,1 | INITIALIZE. | |
| 4 | | | AXT | N,2 | | |
| 5 | | | STZ | FREE | | |
| 6 | | | CLA | IMP+N,2 | GET FIRST MEASUREMENT. | |
| 7 | | | TZE | TIMP+10 | IF ZERO, GET NEXT MEASUREMENT. | |
| 8 | | | TXL | *+3,1,0 | IS SCALING NECESSARY? | |
| 9 | | | SUB | 2HND | YES, DIVIDE BY $2^{100}$. | |
| 10 | | | TMI | 1ERR | ALL IMP MUST BE $> 2^{-200}$. | |
| 11 | | | FAD | FREE | ADD. | |
| 12 | | | TNO | *+3 | DID SPILL OCCUR? | |
| 13 | | | TXH | 2ERR,1,0 | YES, IS THIS THE FIRST SPILL? | |
| 14 | | | TXI | TIMP+3,1,1 | YES, SET INDICATION AND RESTART. | |
| 15 | | | TNX | DONE,2,1 | IS SUM COMPLETE? IF | |
| 16 | | | STO | FREE | NOT, STORE PARTIAL SUM AND | |
| 17 | | | TRA | TIMP+5 | GET NEXT MEASUREMENT. | |
| 18 | DONE | | TQO | *+1 | TURN OFF MQ OV INDICATOR. | |
| 19 | | | DCT | | TURN OFF DIVIDE CHECK INDICATOR. | |
| 20 | | | NOP | | | |
| 21 | | | FDP | SURA | DIVIDE BY SURFACE AREA. | |
| 22 | | | DCT | | TEST. | |
| 23 | | | TRA | TOBIG | COULD NOT DIVIDE. | |
| 24 | | | TQO | TOBIG | QUOTIENT OUT OF RANGE. | |
| 25 | | | TOV | *+1 | TURN OFF ACC OVERFLOW INDICATOR. | |
| 26 | | | XCA | | QUOTIENT TO THE ACC. | |
| 27 | | | TXL | *+3,1,0 | WAS TIMP SCALED? | |
| 28 | | | ADD | 2HND | YES,--MULTIPLY BY $2^{100}$. | |
| 29 | | | TOV | TOBIG | FIXED POINT OVERFLOW INDICATES N TOO LARGE. | |
| 30 | | | FSB | MAX | SUBTRACT. | |
| 31 | | | TPL | TOBIG | Q≥MAX. | |
| 32 | | | TRA | OK | Q IS OK, PROCEED. | |
| 33 | FREE | | BSS | 1 | | |
| 34 | 2HND | | OCT | 100000000000 | | |
| 35 | OK | | EFTM | | TURN ON FPT. | |
| 36 | | | PROCEED WITH MAIN PROGRAM. | | | |
| | | | | | | |
| | | | | | | |

Figure 158. Floating-point Spill, 704 Mode

PROBLEM

CODER | DATE | PAGE OF

| LOCATION | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS | IDENTIFICATION |
|---|---|---|---|---|
| TIMP | STZ | | CLEAR. | |
| | AXT | SPILL, 1 | SET CONTROL ADDRESS IN THE | |
| | SXA | 8, 1 | FPT CONTROL ROUTINE. | |
| | AXT | 0, 2 | INITIALIZE | |
| ADD | AXT | N, 1 | | |
| | STZ | FREE | | |
| | CLA | IMP+N, 1 | FIRST MEASUREMENT. | |
| | TXL | *+4, 2, 0 | IS SCALING NEEDED? | |
| | TZE | *+3 | YES, BUT SKIP ZEROS. | |
| | SUB | 2HND | DIVIDE BY $2^{100}$ | |
| | TMI | 1ERR | SHOULD NOT BE MINUS. | |
| | FAD | FREE | ADD | |
| | STO | FREE | STORE PARTIAL SUM. | |
| | TIX | *-7, 1, 1 | CONTINUE | |
| | TRA | PROC | SUM COMPLETE, PROCEED. | |
| SPILL | TXI | *+1, 2, 1 | GET CONTROL, SET SCALE SIGNAL. | |
| | STO | FREE | SAVE PARTIAL SUM | |
| | CAL | 0 | GET WORD AT LOCATION 0000. | |
| | ARS | 19 | MQ UNDERFLOW IS NOT SIGNIFICANT. | |
| | TNZ | *+2 | ACCUMULATOR SPILL IF NOT ZERO. | |
| | TXI | SPILL-2, 2, 32767 | MAKE XRB = 0 AND PROCEED. | |
| | TXH | 2ERR, 2, 1 | ERROR IF A SECOND SPILL. | |
| | TRA | ADD | RESTART SUM WITH SCALING | |
| PROC | AXT | SPIL2, 1 | NEW CONTROL ADDRESS FOR | |
| | SXA | 8, 1 | FPT CONTROL ROUTINE. | |
| | DCT | | TURN INDICATOR OFF. | |
| | NOP | | | |
| | FDP | SURA | DIVIDE BY SURFACE AREA. | |
| | DCT | | TEST DIVIDE CHECK INDICATOR. | |
| | TRA | TOBIG | NUMBER IS OUT OF RANGE. | |
| | TOV | *+1 | TURN INDICATOR OFF. | |
| | XCA | | MQ TO ACCUMULATOR. | |
| | TXL | *+3, 2, 0 | WAS MQ SCALED? | |
| | ADD | 2HND | YES, MULTIPLY BY $2^{100}$. | |
| | TOV | TOBIG | NUMBER IS OUT OF RANGE. | |
| | FSB | MAX | CHECK MAX. | |
| | TPL | TOBIG | NUMBER IS OUT OF RANGE IF PLUS. | |
| | TRA | OK | CONTINUE. | |
| FREE | BSS | | | |
| 2HND | OCT | 100000000000 | | |
| OK | AXT | OK, 1 | NEW CONTROL ADDRESS. | |
| | SXA | 8, 1 | | |
| | PROCEED WITH MAIN PROGRAM | | | |
| SPIL2 | TRA | *+1 | TAKE CONTROL. | |
| | LXD | 0, 1 | GET SPILL INDICATOR CODE. | |
| | TXL | *+3, 1, 8 | IF NOT FDP, RETURN TO ROUTINE. | |
| | TXH | TOBIG, 1, 10 | SPILL IN ACC ALONE IS NOT SIGNIFICANT. | |
| | TXL | TOBIG, 1, 9 | MQ SPILL MEANS QUOTIENT IS OUT OF RANGE. | |
| | TRA* | 0 | CONTINUE ROUTINE IF OK. | |
| 0010 | XEC | | ADDRESS SUPPLIED BY THE PROGRAM. | |
| | STQ | FPO+2 | IF CONTROL REMAINS HERE. | |
| | STO | FPO+1 | THEN PROGRAM IS FINISHED. | |
| | LRS | 33 | GO INTO PRINT | |
| | STA | FPO | ROUTINE AND | |
| | TSX | PRINT, 4 | PRINT, THEN | |
| | HTR | | STOP. | |

Figure 159. Floating-Point Trap

flexible technique. Note the use made of the decrement bits at location 0000 and a use of the address field of location 0000 in the routine SPIL2.

*Line 1.* Location 0000 is cleared to erase any data that may have been placed in it by a previous routine. The symbolic address SPILL is placed in the XEC instruction located at 0010. This causes the trap routine to return control to this program. NOTE: a post-mortem print that prints location 0010 informs the programmer that his program has passed this point.

*Line 16.* The TXI instruction regains control of the program from the trap routine, and sets the scaling signal by placing a 1 in XRB. Recall that an MQ spill was not significant at this point; therefore, with the partial sum stored, the bit code in the decrement portion of location 0000 is checked. If position 17 has a 1 while positions 14, 15, and 16 have 0's, the summation is continued and XRB is set to zero (line 21).

*Line 22.* If scaling has already been in progress, a second spill is defined as an error.

*Line 24.* A new control address is placed at location 0010, because a spill in the following routines is to be treated in a different fashion. (Again, the address field of location 0010 can inform the programmer that this point of the program has been processed).

*Line 35.* Note that the accumulator overflow indicator is related to fixed-point operations only, when the FPT is on.

*Line 44.* The transfer instruction seizes control from the trap routine. The decision is made as follows:

1. If the spill occurred in the FSB instruction, the octal code in the decrement of location zero is less than 0010. Such a spill is not significant and a return to the routine is made by an indirectly addressed TRA instruction.

2. If the spill occurred in the accumulator alone (on FDP), the octal code in the decrement of location 0000 is not greater than 0012 and not less than or equal to 0011. Accumulator spill, alone, at the FDP instruction is not significant. With an MQ spill, the conditions just stated are not met; the program proceeds to TOBIG location (MQ overflow or underflow is defined as a TOBIG condition).

## Card Reader Wiring for Columnar Binary

The modification (special feature) required in order to use the IBM 714 Card Reader for auxiliary columnar binary card-to-tape operations includes the addition of nine hubs (labeled A, B, C, D, E, F, G, H and J in Figure 160) for use in control panel wiring. The functions of these hubs are:

*F and G* are common hubs and are connected on an electronic sensing circuit which recognizes the presence of the columnar binary indication (9 or 9-7). The detection of a 9-punch (or 9-7 punches) when a card is in the first read position causes that card to be treated as a columnar binary card when at the second read.

*H and J* are connected only during the portion of read time of a columnar binary card at second read. These hubs are connected through a relay contact which is transferred during 9-time only, if the columnar binary identification is a 9-punch, or during 9-, 8- and 7-time if the identification is a 9-7 combination. These hubs can be wired to prevent a 9-punch (or 9, 8, 7 punches) from being read into the record storage unit if the columnar binary identification is not to be placed on tape. (Extra bits are injected into the checking circuits when a columnar binary card is at second read, thus correcting the horizontal row count at second read.)

*A, B, C, D and E* are connected in the following manner: A and C, B and D, are connected when a BCD card is at second read. When a columnar binary card is at second read, hubs B and C, D and E, are connected.

Figures 160 and 161 show SHARE standard wiring for columnar binary card-to-tape operations. With this wiring, columnar binary is converted to standard binary tape format with the proper check bits, and the IBM card codes are converted to standard BCD tape characters with the proper check bits. The wiring in Figure 161 is straightforward and requires no explanation. However, Figure 160 wiring does require some explanation.

Hub F is wired from hub 1 of first read because column 1 will contain the columnar binary card identification. The wires numbered 4 through 10 create a "look-ahead" indication in every record to indicate the type of tape record which follows it. The wiring is such that when a columnar binary card is at first read, 9-impulses enter hubs 81 and 82, and 7-impulses enter hubs 83 and 84 of record storage entry, regardless of the type of card at second read. If the card at second read is a columnar binary card, a 7 impulse

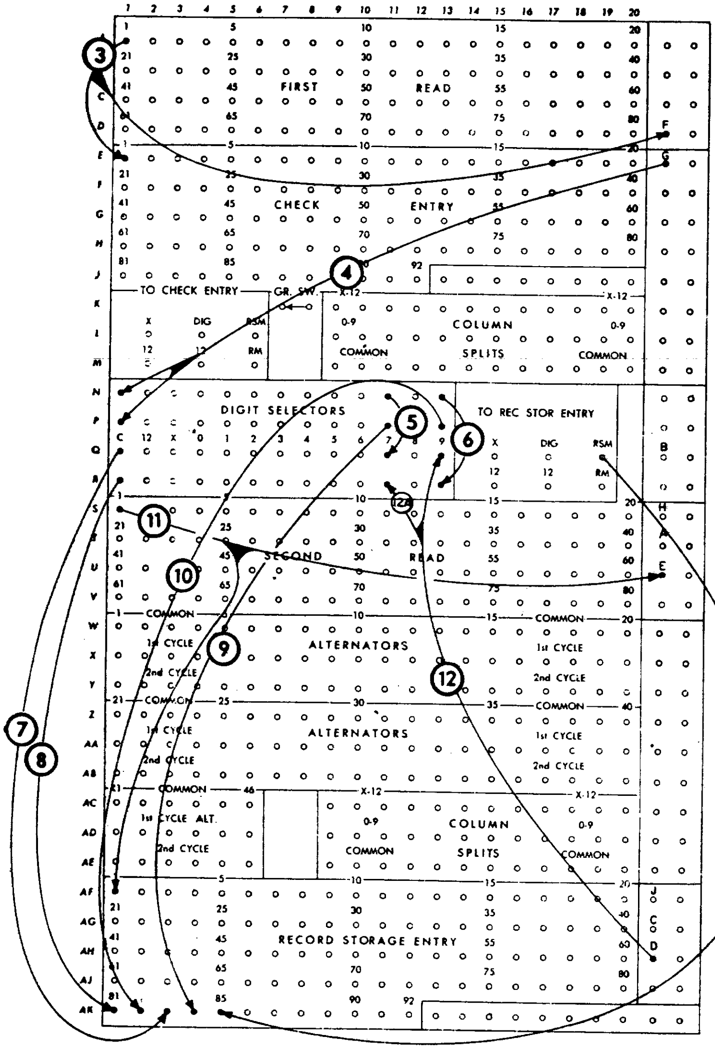


Figure 160. Columnar Binary Wiring

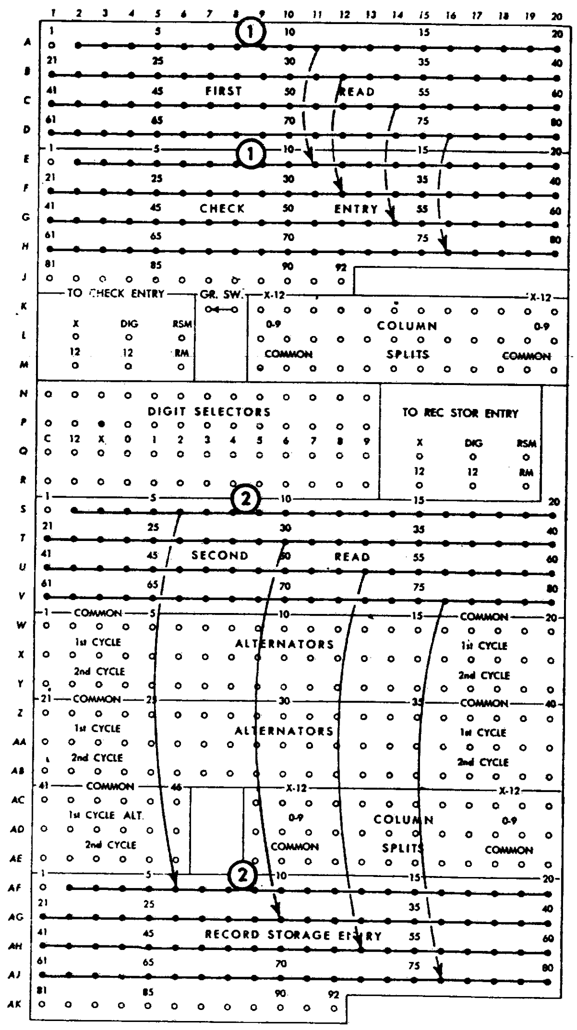


Figure 161. Columnar Binary Wiring

enters hub 81, and a 9 impulse enters hub 83 of record storage entry. This arrangement creates the following look-ahead words in tape records.

A. BCD Record

1. If followed by a binary tape record, the 14th word of the record will be (*xx xx* 11 11 07 07)$_8$, where *X* indicates a digit which varies from record to record.
2. If followed by another BCD record, the 14th word will be (*xx xx* 00 00 00 00)$_8$.

B. Binary Record

1. If followed by another binary record, the 27th and 28th words of the record will be: (*xxxx xxxx* 0005)$_8$ and (0001 0005 0004)$_8$.
2. If followed by a BCD record, the 27th and 28th words of the record will be: (*xxxx xxxx* 0004)$_8$ and (0000 0001 0000)$_8$.

Note that wire 12 crates redundant bits when a columnar binary card is at second read. These bits are necessary because, as previously explained, a 9 impulse or 9 and 7 impulses are injected into the checking circuit when a columnar binary card is at second read. Wire 12 compensates for these injected bits so that the horizontal row count at second read will be correct. When the columnar binary identification is a 9 punch only, wire 12a should be removed in order to prevent a read check from occurring.

# Appendix A

## Number Systems and Conversion

The common decimal notation of the commercial and scientific world is familiar to all of us. This notation is so familiar that you probably have never before questioned its use. Could it be possible that, for some purposes, another system is more convenient? The decision is entirely a matter of convenience. Decimal notation is used because it is most familiar and is understood by most people. However, had our primeval ancestors developed eight fingers instead of ten we would probably be more familiar with the octal system and would be questioning the decimal system.

The decimal system, with its ten digits, is learned by most people early in their training. This system serves very well for counting purposes. Why then, should computers which are designed to assist mathematicians, or engineers and businessmen, be designed to use the binary system of numbers?

Current digital computers use binary circuits and the mathematics of the computers is therefore binary in nature. The only convenient way to learn the operation of a computer is to learn the binary system. The octonary or octal system is a shorthand method of writing long binary numbers. Octal notation is used when discussing the computer but has no relation to the internal computer circuits.

Perhaps, as a first step, it would be well to see what is meant by the binary system of numbers. The binary, or base-two system, uses two symbols, 0 and 1, to represent all quantities. Counting is started in the binary system in the same manner as in the decimal system with 0 for zero and 1 for one. At two in the binary system it is found that there are no more symbols to be used. It is therefore necessary to take the same move at two in the binary system that is taken at ten in the decimal system. This move is to place a 1 in the next position to the left and start again with a 0 in the original position. A binary 10 is equivalent in this respect to a 2 in the decimal system. Counting is continued in an analogous manner with a carry to the next higher order every time a two is reached instead of every time a ten is reached. Counting in the binary system is as follows:

| BINARY | DECIMAL | BINARY | DECIMAL |
|---|---|---|---|
| 0 | 0 | 101 | 5 |
| 1 | 1 | 110 | 6 |
| 10 | 2 | 111 | 7 |
| 11 | 3 | 1000 | 8 |
| 100 | 4 | 1001 | 9 |

The binary system is used in computers because all present components are inherently binary. That is, a relay maintains its contacts either closed or open, magnetic materials are utilized by magnetizing them in one direction or the other, a vacuum tube is conveniently maintained either fully conducting or nonconducting, or the transmission of information along a wire may be accomplished by transmitting or not transmitting an electrical pulse at a certain time.

Although binary numbers in general have more terms than their decimal counterparts (about 3.3 times as many), computation in the binary system is quite simple.

For *addition,* it is only necessary to remember the following three rules:

1. Zero plus zero equals zero.
2. Zero plus one equals one.
3. One plus one equals zero with a carry of one to the next position on the left. To see how the rules work, consider the addition of 15 plus 7 with these numbers expressed in binary notation:

| | SIXTEENS | EIGHTS | FOURS | TWOS | ONES |
|---|---|---|---|---|---|
| (carries) | (1) | (1) | (1) | (1) | |
| | 0 | 1 | 1 | 1 | 1 = 15 |
| | + 0 | 0 | 1 | 1 | 1 = 7 |
| | 1 | 0 | 1 | 1 | 0 = 22 |

In the ones column we have 1 plus 1 for a sum of 0 and a 1 carried to the two column. In the twos column we have 1 plus 1 for a sum of 0 but we must also add the carry from the ones column, making a final sum of 1 with a carry to the fours column. The same procedure occurs in the fours column. In the eights column we have a 1 plus a 0 giving a sum of 1, but adding in the carry from the fours column makes the final sum 0 with a carry to the sixteens column. In this column we have 0 plus 0 giving a sum of 0 and to this we add the carry from the eights column, making a final sum of 1.

The resultant sum of the addition contains 1's in the sixteens, fours, and twos columns, which is the binary representation of 22, the correct sum of 15 plus 7 (16 plus 4 plus 2 equals 22).

The rules for *subtraction* of binary digits are equally simple:

1. Zero minus zero equals zero.
2. One minus one equals zero.

3. One minus zero equals one.
4. Zero minus one equals one, with one borrowed from the left.

Using the same numbers as we did in the addition, the subtraction works as follows:

| | SIXTEENS | EIGHTS | FOURS | TWOS | ONES | |
|---|---|---|---|---|---|---|
| (borrows) | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 1 | 1 | 1 | 1 | = 15 |
| | − 0 | 0 | 1 | 1 | 1 | = 7 |
| | 0 | 1 | 0 | 0 | 0 | = 8 |

In the ones column we have 1 minus 1 for a sum of 0 with no borrows. The same procedure occurs in the twos and fours columns. In the eights column we have 1 minus 0 for a sum of 1. In the sixteens column we have 0 minus 0 for a sum of 0. With the subtraction finished we have 1's in the eights column only, signifying the answer to be 8.

For *multiplication* only three rules need to be remembered:

1. Zero times zero equals zero.
2. Zero times one equals zero; no carries are considered.
3. One times one equals one.

The binary multiplication table is such that all that is necessary when multiplying one number (multiplicand) by another (multiplier) is to examine the multiplier digits one at a time and, each time a 1 is found, add the multiplicand into the result, and each time a 0 is found add nothing. Of course, the multiplicand must be shifted for each multiplier digit, but this is not different from the shifting that is done in the decimal system.

An example of binary multiplication is 26 multiplied by 19:

| DECIMAL | BINARY |
|---|---|
| 26 = 16 + 8 + 0 + 2 + 0 = | 11010 |
| × 19 = 16 + 0 + 0 + 2 + 1 = | 10011 |
| Using the above rules, the product | 11010 |
| will be arrived at by a series | 11010 |
| of adding the multiplicand | 00000 |
| and shifting whenever | 00000 |
| a 1 is found in the | 11010 |
| multiplier. | 111101110 |

Interpreting the binary result of the multiplication by using the ones, twos, fours, . . . etc., system we find that we have,

256 + 128 + 64 + 32 + 0 + 8 + 4 + 2 + 0

which equals 494, thus proving the problem.

Binary division is accomplished by applying similar concepts. From the examples of addition, subtraction, and multiplication, it may be seen that whatever operation the computer is working on will be accomplished by repetitive addition.

The computer operates internally using the binary system. However, it is able to convert from one system to another by use of a stored program. Thus, input-output data may be expressed in decimal (or any other) form when the operator finds it more convenient to do so.

## Octal Number System

It has already been pointed out that binary numbers require about three times as many positions as decimal numbers to express the equivalent number. This is not much of a problem to the computer itself. However, in talking and writing, these binary numbers are bulky. A long string of ones and zeros cannot be effectively transmitted from one individual to another. Some shorthand method is necessary. The octal number system fills this need. Because of its simple relationship to binary, numbers can be converted from one system to another by inspection. The base or radix of the octal system is 8. This means there are eight symbols: 0, 1, 2, 3, 4, 5, 6, and 7. There are no 8's or 9's in this number system. The important relationship to remember is that three binary positions are equivalent to one octal position. The following table is used constantly when working on or about the computer.

| BINARY | OCTAL |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

At this point a carry to the next higher position of the number is necessary, since all eight symbols have been used.

| BINARY | OCTAL |
|---|---|
| 001 000 | 10 |
| 001 001 | 11 |
| 001 010 | 12 |
| 001 011 | 13 |
| 001 100 | 14 |

and so on.

Remember that as far as the internal circuitry of the computer is concerned it only understands binary ones and zeros. The octal system is used to provide a shorthand method of reading and writing binary numbers.

## Number Conversions

Before an attempt is made to convert numbers from one system to another, it is best to review what a number represents. In the demical system a number is represented or expressed by a sum of terms. Each individual term consists of a product of a power of ten and some integer from 0 to 9. For example, the number 123 means 100 plus 20 plus 3. This may also be expressed as:

$$(1 \times 10^2) + (2 \times 10^1) + (3 \times 10^0)$$

Ten is said to be the base or radix of this system because of the role that the powers of 10 and the integers up to 10 play in the above expansion. If two is chosen as the base, numbers are said to be represented in the binary system. Consider the binary number 1 111 011. What do these zeros and ones represent? They represent the coefficients of the ascending powers of 2. Expressed in another way the number is:

$$(1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) +$$
$$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

The various orders do not have the meaning of units, tens, hundreds, thousands, etc., as in the decimal system; instead they signify units, twos, fours, eights, sixteens, etc. In applying the above information it is found that the number 123 breaks down in both systems as follows:

```
BINARY                                DECIMAL
1  111  011                           1 2 3
           1  units                        3  units
           2  twos                        20  tens
           0  fours                      100  hundreds
           8  eights                     ---
          16  sixteens                   123
          32  thirty-twos
          64  sixty-fours
         ---
         123
```

In the octal system, a number is represented in the same manner except that the base is 8. The digits of the number represent the coefficients of the ascending powers of 8. Consider the octal number:

$$\begin{aligned} 173 &= (1 \times 8^2) + (7 \times 8^1) + (3 \times 8^0) \\ &= \ \ 64 \quad + \quad 56 \quad + \quad 3 \\ &= 123 \text{ (decimal)} \end{aligned}$$

Similarly:

```
Octal 173
        3  units
       56  eights
       64  sixty-fours
```

By remembering what a number represents in the binary or octal system, the number can be converted to its decimal equivalent by the method shown above. As the numbers get bigger, this method becomes quite impossible to use. The following section provides detailed methods for converting from one system to another.

### Integers

DECIMAL TO OCTAL

Convert the decimal number 149 to its octal equivalent. RULE: Divide the decimal number by 8 and develop the octal number as per example.

```
8 | 149  Remainder  5   ↑
8 |  18     "       2   |   = 225
8 |   2     "       2   |
      0               read
```

We first divided the original number to be converted by 8. The remainder of this first division becomes the low-order digit of the conversion (5). We then divide the quotient (received from the first division) by 8. Again the remainder becomes a part of the answer (next higher order, 2). This is continued until the quotient is smaller than the divisor. At this time the final quotient is considered the high order of the conversion (2).

OCTAL TO DECIMAL

Convert the octal number 225 to its decimal equivalent. RULE: Multiply by 8 and add, as per example.

```
  2 2 5
× 8
  16
+  2
  18
×  8
 144
+  5
 149
```

The high-order digit is multiplied by 8 and the next lower-order digit is added to the result. The resultant answer is then multiplied by 8 and the next lower-order digit is added to the result. When the low-order digit has been added to the answer, the process ends. In the following examples, where multiplication or division is used, detailed explanations will not be used because the operations are similar.

### Octal to Binary and Binary to Octal

Rule: Express the number in binary groups of three.

OCTAL TO BINARY

$$\underbrace{2}_{010}\ \underbrace{2}_{010}\ \underbrace{5}_{101} = 010\ 010\ 101$$

BINARY TO OCTAL

$$\underbrace{010}_{2}\ \underbrace{010}_{2}\ \underbrace{101}_{5} = 225$$

### Decimal to Binary

Rule: Divide the decimal number by 2 and develop as per example; convert 149 to its binary equivalent.

$$\begin{array}{rrcc}
2 & \underline{149} & \text{Remainder} & 1\uparrow \\
2 & \underline{74} & \text{"} & 0 \\
2 & \underline{37} & \text{"} & 1 \\
2 & \underline{18} & \text{"} & 0 \\
2 & \underline{9} & \text{"} & 1 \\
2 & \underline{4} & \text{"} & 0 \\
2 & \underline{2} & \text{"} & 0 \\
2 & \underline{1} & \text{"} & 1 \\
 & 0 & \text{"} & \text{read}
\end{array} = 010\quad 010\quad 101$$

### Binary to Decimal

Rule: Multiply by 2 and add as per example; convert 010 010 101 to its decimal equivalent.

$$\begin{array}{r}
10\quad 010\quad 101 \\
\times\ 2 \\
\hline
2 \\
+\ 0 \\
\hline
2 \\
\times\ 2 \\
\hline
4 \\
+\ 0 \leftarrow \\
\hline
4 \\
\times\ 2 \\
\hline
8 \\
+\ 1 \leftarrow \\
\hline
9 \\
\times\ 2 \\
\hline
18 \\
+\ 0 \leftarrow \\
\hline
18 \\
\times\ 2 \\
\hline
36 \\
+\ 1 \leftarrow \\
\hline
37 \\
\times\ 2 \\
\hline
74 \\
+\ 0 \leftarrow \\
\hline
74 \\
\times\ 2 \\
\hline
148 \\
+\ 1 \leftarrow \\
\hline
149
\end{array}$$

OR 10 010 101

$$= 1\,(2^7) + 0\,(2^6) + 0\,(2^5) + 1\,(2^4) + 0\,(2^3) + 1\,(2^2) + 0\,(2^1) + 1\,(2^0)$$

$$= 128 + 16 + 4 + 1$$

$$= 149$$

## Fractions

### Decimal to Octal

Rule: Multiply by 8 and develop the octal number as per example:

$$\begin{array}{c|r}
\text{Read} & .149 \\
\bullet & \times\ 8 \\
\hline
1 & .192 \\
 & \times\ 8 \\
\hline
1 & .536 \\
 & \times\ 8 \\
\hline
4 & .288 \\
 & \times\ 8 \\
\hline
2\ \downarrow & .304
\end{array}$$

$$= .1142 +$$

### Octal to Decimal

Rule: Express as powers of 8, add and divide as per example:

$$\begin{aligned}
.1142 &= 1\,(8^{-1}) + 1\,(8^{-2}) + 4\,(8^{-3}) + 2\,(8^{-4}) \\
&= 1/8 + 1/64 + 4/512 + 2/4096 \\
&= 610/4096 \\
&= .1489 \text{ plus} \\
&\text{or } .149
\end{aligned}$$

### Octal to Binary and Binary to Octal

Rule: The same rule applies for fractions as for whole numbers.

Example:

$$\underbrace{.1}_{.001}\ \underbrace{1}_{001}\ \underbrace{4}_{100}\ \underbrace{2}_{010} \qquad \underbrace{.001}_{.1}\ \underbrace{001}_{1}\ \underbrace{100}_{4}\ \underbrace{010}_{2}$$

### Binary to Decimal

The same rule applies as for whole numbers; for example:

.001 001 100 010

$$\begin{aligned}
&= 1\,(2^{-3}) + 1\,(2^{-6}) + 1\,(2^{-7}) + 1\,(2^{-11}) \\
&= 1/8 + 1/64 + 1/128 + 1/2048 \\
&= 305/2048 \\
&= .1489 \text{ plus} \\
&\text{or } .149
\end{aligned}$$

### Decimal to Binary

The same rule applies as for whole numbers. For example:

$$\begin{array}{c|l}
\text{Read} & .149\\
\bullet & \times 2\\
0 & .298\\
 & \times 2\\
0 & .596\\
 & \times 2\\
1 & .192\\
 & \times 2\\
0 & .384\\
 & \times 2\\
0 & .768\\
 & \times 2\\
1 & .536\\
 & \times 2\\
1 & .072\\
 & \times 2\\
0 & .144\\
 & \times 2\\
0 & .288\\
 & \times 2\\
0 & .576\\
 & \times 2\\
1 & .152\\
 & \times 2\\
0 & .304\\
\downarrow\ = & .001\ 001\ 100\ 010\ +
\end{array}$$

## Improper Fractions

### Decimal to Binary

This requires conversion from decimal to octal and then to binary. For example, convert 149.149 to its binary equivalent.

$$\begin{array}{rlll}
8\,\underline{|\ 149.} & \text{remainder} & 5 & \uparrow\\
8\,\underline{|\ 18.} & \text{"} & 2 & |\\
8\,\underline{|\ 2.} & \text{"} & 2 & |\\
0 & & & \text{read.}
\end{array}
\qquad
\begin{array}{c|l}
 & .149\\
 & \times 8\\
1 & .192\\
 & \times 8\\
1 & .536\\
 & \times 8\\
4 & .288\\
 & \times 8\\
\downarrow\ \text{read. } 2 & .304
\end{array}$$

$$= \overbrace{010}^{2}\ \overbrace{010}^{2}\ \overbrace{101}^{5}\bullet\overbrace{001}^{1}\ \overbrace{001}^{1}\ \overbrace{100}^{4}\ \overbrace{010}^{2}$$

$$149.149_{10} = 225.1142_8 = 010\ 010\ 101.001\ 001\ 100\ 010_2$$

### Binary to Decimal

This requires conversion from binary to octal and then to decimal.

Convert to decimal:

$$\underbrace{010}_{2}\ \underbrace{010}_{2}\ \underbrace{101}_{5}\bullet\underbrace{001}_{1}\ \underbrace{001}_{1}\ \underbrace{100}_{4}\ \underbrace{010}_{2}$$

$$\begin{array}{r}
= 2\\
\times 8\\
\hline 16\\
+ 2\\
\hline 18\\
\times 8\\
\hline 144\\
+ 5\\
\hline 149.
\end{array}
\qquad
\begin{array}{l}
\frac{1}{8} + \frac{1}{64} + \frac{4}{512} + \frac{2}{4096} =\\
\frac{610}{4096} =\\
.149
\end{array}$$

As with decimal-to-binary, conversion of the integer and fraction parts is performed independently.

## Floating-Point Word

### Decimal to Floating Point

Convert decimal 149.149 to normal floating-point word.

Decimal to octal:

$149.149_{10} = 225.1142_8$

Octal to binary:

$225.1142_8 = 010\ 010\ \ 101.001\ 001\ 100\ 010_2$

Binary to floating point word:

$10\ 010\ 101.001\ 001\ 100\ 010 \times 2^0 =$

$.10\ 010\ 101\ 001\ 001\ 100\ 010 \times 2^8$

$8 + 128 = 136$ (Characteristic)

$$\underbrace{10\ 001\ 000}_{\text{Characteristic}}.\underbrace{100\ 101\ 010\ 010\ 011\ 000\ 1}_{\text{Fraction}}\ \text{FP}$$

$2\ \ 1\ \ 0\ .\ 4\ \ 5\ \ 2\ \ 2\ \ 3\ \ 0\ \ 4_8$

NOTE: Word is normal if the fraction is less than 1, but greater than or equal to one-half.

# Appendix B

## Table of Powers of Two

| $2^n$ | $n$ | $2^{-n}$ |
|---:|---:|:---|
| 1 | 0 | 1.0 |
| 2 | 1 | 0.5 |
| 4 | 2 | 0.25 |
| 8 | 3 | 0.125 |
| 16 | 4 | 0.062 5 |
| 32 | 5 | 0.031 25 |
| 64 | 6 | 0.015 625 |
| 128 | 7 | 0.007 812 5 |
| 256 | 8 | 0.003 906 25 |
| 512 | 9 | 0.001 953 125 |
| 1 024 | 10 | 0.000 976 562 5 |
| 2 048 | 11 | 0.000 488 281 25 |
| 4 096 | 12 | 0.000 244 140 625 |
| 8 192 | 13 | 0.000 122 070 312 5 |
| 16 384 | 14 | 0.000 061 035 156 25 |
| 32 768 | 15 | 0.000 030 517 578 125 |
| 65 536 | 16 | 0.000 015 258 789 062 5 |
| 131 072 | 17 | 0.000 007 629 394 531 25 |
| 262 144 | 18 | 0.000 003 814 697 265 625 |
| 524 288 | 19 | 0.000 001 907 348 632 812 5 |
| 1 048 576 | 20 | 0.000 000 953 674 316 406 25 |
| 2 097 152 | 21 | 0.000 000 476 837 158 203 125 |
| 4 194 304 | 22 | 0.000 000 238 418 579 101 562 5 |
| 8 388 608 | 23 | 0.000 000 119 209 289 550 781 25 |
| 16 777 216 | 24 | 0.000 000 059 604 644 775 390 625 |
| 33 554 432 | 25 | 0.000 000 029 802 322 387 695 312 5 |
| 67 108 864 | 26 | 0.000 000 014 901 161 193 847 656 25 |
| 134 217 728 | 27 | 0.000 000 007 450 580 596 923 828 125 |
| 268 435 456 | 28 | 0.000 000 003 725 290 298 461 914 062 5 |
| 536 870 912 | 29 | 0.000 000 001 862 645 149 230 957 031 25 |
| 1 073 741 824 | 30 | 0.000 000 000 931 322 574 615 478 515 625 |
| 2 147 483 648 | 31 | 0.000 000 000 465 661 287 307 739 257 812 5 |
| 4 294 967 296 | 32 | 0.000 000 000 232 830 643 653 869 628 906 25 |
| 8 589 934 592 | 33 | 0.000 000 000 116 415 321 826 934 814 453 125 |
| 17 179 869 184 | 34 | 0.000 000 000 058 207 660 913 467 407 226 562 5 |
| 34 359 738 368 | 35 | 0.000 000 000 029 103 830 456 733 703 613 281 25 |
| 68 719 476 736 | 36 | 0.000 000 000 014 551 915 228 366 851 806 640 625 |
| 137 438 953 472 | 37 | 0.000 000 000 007 275 957 614 183 425 903 320 312 5 |
| 274 877 906 944 | 38 | 0.000 000 000 003 637 978 807 091 712 951 660 156 25 |
| 549 755 813 888 | 39 | 0.000 000 000 001 818 989 403 545 856 475 830 078 125 |

# Appendix C Octal-Decimal Integer Conversion Table

| 0000<br>to<br>0777<br>(Octal) | 0000<br>to<br>0511<br>(Decimal) |
|---|---|

| Octal | Decimal |
|---|---|
| 10000 - | 4096 |
| 20000 - | 8192 |
| 30000 - | 12288 |
| 40000 - | 16384 |
| 50000 - | 20480 |
| 60000 - | 24576 |
| 70000 - | 28672 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0000 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 |
| 0010 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 0020 | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 |
| 0030 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 0040 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 |
| 0050 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 0060 | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 |
| 0070 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 0100 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 |
| 0110 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 0120 | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 |
| 0130 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 0140 | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 |
| 0150 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 0160 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 |
| 0170 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 0200 | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 |
| 0210 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 0220 | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 |
| 0230 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0240 | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 |
| 0250 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0260 | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 |
| 0270 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0300 | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 |
| 0310 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0320 | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 |
| 0330 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0340 | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 |
| 0350 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0360 | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 |
| 0370 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0400 | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 |
| 0410 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 0420 | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 |
| 0430 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 0440 | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 |
| 0450 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 0460 | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 |
| 0470 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 0500 | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 |
| 0510 | 0328 | 0329 | 0330 | 0331 | 0332 | 0333 | 0334 | 0335 |
| 0520 | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 |
| 0530 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 0540 | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 |
| 0550 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 0560 | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 |
| 0570 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 0600 | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 |
| 0610 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 0620 | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 |
| 0630 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 0640 | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 |
| 0650 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 0660 | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 |
| 0670 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 0700 | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 |
| 0710 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 0720 | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 |
| 0730 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 0740 | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 |
| 0750 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 0760 | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 |
| 0770 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |

| 1000<br>to<br>1777<br>(Octal) | 0512<br>to<br>1023<br>(Decimal) |
|---|---|

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1000 | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 |
| 1010 | 0520 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 1020 | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 |
| 1030 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 1040 | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 |
| 1050 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 1060 | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 |
| 1070 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 1100 | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 |
| 1110 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 1120 | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 |
| 1130 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 1140 | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 |
| 1150 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 1160 | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 |
| 1170 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 1200 | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 |
| 1210 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 1220 | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 |
| 1230 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 1240 | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 |
| 1250 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 1260 | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 |
| 1270 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 1300 | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 |
| 1310 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 1320 | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 |
| 1330 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 1340 | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 |
| 1350 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 1360 | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 |
| 1370 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1400 | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 |
| 1410 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 1420 | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 |
| 1430 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 1440 | 0800 | 0801 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 |
| 1450 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 1460 | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 |
| 1470 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 1500 | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0839 |
| 1510 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 1520 | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 |
| 1530 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 1540 | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 |
| 1550 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 1560 | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 |
| 1570 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 1600 | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 |
| 1610 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 1620 | 0912 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 |
| 1630 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 1640 | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 |
| 1650 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 1660 | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 |
| 1670 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 1700 | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 |
| 1710 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 1720 | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 |
| 1730 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 1740 | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 |
| 1750 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 1760 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 |
| 1770 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |

# Octal-Decimal Integer Conversion Table

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 2000 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 |
| 2010 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 2020 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 |
| 2030 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 2040 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 |
| 2050 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 2060 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 |
| 2070 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 2100 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 |
| 2110 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 2120 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 |
| 2130 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 2140 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 |
| 2150 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 2160 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 |
| 2170 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 2200 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 |
| 2210 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 2220 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 |
| 2230 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 2240 | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 |
| 2250 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 2260 | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 |
| 2270 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 2300 | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 |
| 2310 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 2320 | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 |
| 2330 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 2340 | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 |
| 2350 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 2360 | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 |
| 2370 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 2400 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 |
| 2410 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 |
| 2420 | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 |
| 2430 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 2440 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 |
| 2450 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 2460 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 |
| 2470 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 2500 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 |
| 2510 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 |
| 2520 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 |
| 2530 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 2540 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 |
| 2550 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 2560 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 |
| 2570 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 2600 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 |
| 2610 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 |
| 2620 | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 |
| 2630 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 2640 | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 |
| 2650 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 2660 | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 |
| 2670 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 2700 | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 |
| 2710 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 2720 | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 |
| 2730 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 2740 | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 |
| 2750 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 2760 | 1520 | 1521 | 1522 | 1523 | 1524 | 1525 | 1526 | 1527 |
| 2770 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |

| | |
|---|---|
| 2000<br>to<br>2777<br>(Octal) | 1024<br>to<br>1535<br>(Decimal) |

| Octal | Decimal |
|---|---|
| 10000 - | 4096 |
| 20000 - | 8192 |
| 30000 - | 12288 |
| 40000 - | 16384 |
| 50000 - | 20480 |
| 60000 - | 24576 |
| 70000 - | 28672 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 3000 | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 |
| 3010 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 3020 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 |
| 3030 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 3040 | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 |
| 3050 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 3060 | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 |
| 3070 | 1592 | 1593 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 3100 | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 |
| 3110 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 3120 | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 |
| 3130 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 3140 | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 |
| 3150 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 3160 | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 |
| 3170 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 3200 | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 |
| 3210 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 3220 | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 |
| 3230 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 3240 | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 |
| 3250 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 3260 | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 |
| 3270 | 1720 | 1721 | 1722 | 1723 | 1724 | 1725 | 1726 | 1727 |
| 3300 | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 |
| 3310 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 3320 | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 |
| 3330 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 3340 | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 |
| 3350 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 3360 | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 |
| 3370 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 3400 | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 |
| 3410 | 1800 | 1801 | 1802 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 3420 | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 |
| 3430 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 3440 | 1824 | 1825 | 1826 | 1827 | 1828 | 1829 | 1830 | 1831 |
| 3450 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 3460 | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 |
| 3470 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 3500 | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 |
| 3510 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 3520 | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 |
| 3530 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 3540 | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 |
| 3550 | 1896 | 1897 | 1898 | 1899 | 1900 | 1901 | 1902 | 1903 |
| 3560 | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 |
| 3570 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 3600 | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 |
| 3610 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 3620 | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 |
| 3630 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 3640 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 |
| 3650 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 3660 | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 |
| 3670 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 3700 | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 |
| 3710 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 3720 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 |
| 3730 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 3740 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 |
| 3750 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 3760 | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 |
| 3770 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |

| | |
|---|---|
| 3000<br>to<br>3777<br>(Octal) | 1536<br>to<br>2047<br>(Decimal) |

# Octal-Decimal Integer Conversion Table

| 4000<br>to<br>4777<br>(Octal) | 2048<br>to<br>2559<br>(Decimal) |
|---|---|

| Octal | Decimal |
|---|---|
| 10000 - | 4096 |
| 20000 - | 8192 |
| 30000 - | 12288 |
| 40000 - | 16384 |
| 50000 - | 20480 |
| 60000 - | 24576 |
| 70000 - | 28672 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 4000 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 |
| 4010 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 4020 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 |
| 4030 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 4040 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 |
| 4050 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 4060 | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 |
| 4070 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 4100 | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 |
| 4110 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 4120 | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 |
| 4130 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 4140 | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 |
| 4150 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 4160 | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 |
| 4170 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 4200 | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 |
| 4210 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 4220 | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 |
| 4230 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 4240 | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 |
| 4250 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 4260 | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 |
| 4270 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 4300 | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 |
| 4310 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 4320 | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 |
| 4330 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 4340 | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 |
| 4350 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 4360 | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 |
| 4370 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 4400 | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 |
| 4410 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 4420 | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 |
| 4430 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 4440 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 |
| 4450 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 4460 | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 |
| 4470 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 4500 | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 |
| 4510 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 4520 | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 |
| 4530 | 2392 | 2393 | 2394 | 2395 | 2396 | 2397 | 2398 | 2399 |
| 4540 | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 |
| 4550 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 4560 | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 |
| 4570 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 4600 | 2432 | 2433 | 2434 | 2435 | 2436 | 2437 | 2438 | 2439 |
| 4610 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 4620 | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 |
| 4630 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 4640 | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2470 | 2471 |
| 4650 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 4660 | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 |
| 4670 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| 4700 | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 |
| 4710 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 4720 | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 |
| 4730 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 4740 | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 |
| 4750 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 4760 | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 |
| 4770 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |

| 5000<br>to<br>5777<br>(Octal) | 2560<br>to<br>3071<br>(Decimal) |
|---|---|

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 5000 | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 |
| 5010 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| 5020 | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 |
| 5030 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| 5040 | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 |
| 5050 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| 5060 | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 |
| 5070 | 2616 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| 5100 | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 |
| 5110 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| 5120 | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 |
| 5130 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| 5140 | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 |
| 5150 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| 5160 | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 |
| 5170 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| 5200 | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 |
| 5210 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| 5220 | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 |
| 5230 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| 5240 | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 |
| 5250 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| 5260 | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 |
| 5270 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| 5300 | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 |
| 5310 | 2760 | 2761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| 5320 | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 |
| 5330 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| 5340 | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 |
| 5350 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| 5360 | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 |
| 5370 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 5400 | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 |
| 5410 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| 5420 | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 |
| 5430 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| 5440 | 2848 | 2849 | 2850 | 2851 | 2852 | 2853 | 2854 | 2855 |
| 5450 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| 5460 | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 |
| 5470 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| 5500 | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2886 | 2887 |
| 5510 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| 5520 | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 |
| 5530 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| 5540 | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 |
| 5550 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| 5560 | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 |
| 5570 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| 5600 | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 |
| 5610 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| 5620 | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 |
| 5630 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| 5640 | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 |
| 5650 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| 5660 | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 |
| 5670 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| 5700 | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 |
| 5710 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| 5720 | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 |
| 5730 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| 5740 | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 |
| 5750 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| 5760 | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 |
| 5770 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |

# Octal-Decimal Integer Conversion Table

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 6000 | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 |
| 6010 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| 6020 | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 |
| 6030 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| 6040 | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 |
| 6050 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| 6060 | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 |
| 6070 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| 6100 | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 |
| 6110 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| 6120 | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 |
| 6130 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| 6140 | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 |
| 6150 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| 6160 | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 |
| 6170 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| 6200 | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 |
| 6210 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| 6220 | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 |
| 6230 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| 6240 | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 |
| 6250 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| 6260 | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 |
| 6270 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| 6300 | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 |
| 6310 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| 6320 | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 |
| 6330 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| 6340 | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 |
| 6350 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| 6360 | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 |
| 6370 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 6400 | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 |
| 6410 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| 6420 | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 |
| 6430 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| 6440 | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 |
| 6450 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| 6460 | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 |
| 6470 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| 6500 | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 |
| 6510 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| 6520 | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 |
| 6530 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| 6540 | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 |
| 6550 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| 6560 | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 |
| 6570 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| 6600 | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 |
| 6610 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| 6620 | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 |
| 6630 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| 6640 | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 |
| 6650 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| 6660 | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 |
| 6670 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| 6700 | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 |
| 6710 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| 6720 | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 |
| 6730 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| 6740 | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 |
| 6750 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| 6760 | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 |
| 6770 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |

| 6000 | 3072 |
|---|---|
| to | to |
| 6777 | 3583 |
| (Octal) | (Decimal) |

| Octal | Decimal |
|---|---|
| 10000 | 4096 |
| 20000 | 8192 |
| 30000 | 12288 |
| 40000 | 16384 |
| 50000 | 20480 |
| 60000 | 24576 |
| 70000 | 28672 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 7000 | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 |
| 7010 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| 7020 | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 |
| 7030 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| 7040 | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 |
| 7050 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| 7060 | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 |
| 7070 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| 7100 | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 |
| 7110 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| 7120 | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 |
| 7130 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| 7140 | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 |
| 7150 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| 7160 | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 |
| 7170 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| 7200 | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 |
| 7210 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| 7220 | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 |
| 7230 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| 7240 | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 |
| 7250 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| 7260 | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 |
| 7270 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |
| 7300 | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 |
| 7310 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| 7320 | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 |
| 7330 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| 7340 | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 |
| 7350 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| 7360 | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 |
| 7370 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 7400 | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 |
| 7410 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| 7420 | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 |
| 7430 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| 7440 | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 |
| 7450 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| 7460 | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 |
| 7470 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| 7500 | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 |
| 7510 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| 7520 | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 |
| 7530 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| 7540 | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 |
| 7550 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| 7560 | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 |
| 7570 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| 7600 | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 |
| 7610 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| 7620 | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 |
| 7630 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| 7640 | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 |
| 7650 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| 7660 | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 |
| 7670 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| 7700 | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 |
| 7710 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| 7720 | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 |
| 7730 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| 7740 | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 |
| 7750 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| 7760 | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 |
| 7770 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |

| 7000 | 3584 |
|---|---|
| to | to |
| 7777 | 4095 |
| (Octal) | (Decimal) |

# Appendix D  Octal-Decimal Fraction Conversion Table

| OCTAL | DEC. | OCTAL | DEC. | OCTAL | DEC. | OCTAL | DEC. |
|---|---|---|---|---|---|---|---|
| .000 | .000000 | .100 | .125000 | .200 | .250000 | .300 | .375000 |
| .001 | .001953 | .101 | .126953 | .201 | .251953 | .301 | .376953 |
| .002 | .003906 | .102 | .128906 | .202 | .253906 | .302 | .378906 |
| .003 | .005859 | .103 | .130859 | .203 | .255859 | .303 | .380859 |
| .004 | .007812 | .104 | .132812 | .204 | .257812 | .304 | .382812 |
| .005 | .009765 | .105 | .134765 | .205 | .259765 | .305 | .384765 |
| .006 | .011718 | .106 | .136718 | .206 | .261718 | .306 | .386718 |
| .007 | .013671 | .107 | .138671 | .207 | .263671 | .307 | .388671 |
| .010 | .015625 | .110 | .140625 | .210 | .265625 | .310 | .390625 |
| .011 | .017578 | .111 | .142578 | .211 | .267578 | .311 | .392578 |
| .012 | .019531 | .112 | .144531 | .212 | .269531 | .312 | .394531 |
| .013 | .021484 | .113 | .146484 | .213 | .271484 | .313 | .396484 |
| .014 | .023437 | .114 | .148437 | .214 | .273437 | .314 | .398437 |
| .015 | .025390 | .115 | .150390 | .215 | .275390 | .315 | .400390 |
| .016 | .027343 | .116 | .152343 | .216 | .277343 | .316 | .402343 |
| .017 | .029296 | .117 | .154296 | .217 | .279296 | .317 | .404296 |
| .020 | .031250 | .120 | .156250 | .220 | .281250 | .320 | .406250 |
| .021 | .033203 | .121 | .158203 | .221 | .283203 | .321 | .408203 |
| .022 | .035156 | .122 | .160156 | .222 | .285156 | .322 | .410156 |
| .023 | .037109 | .123 | .162109 | .223 | .287109 | .323 | .412109 |
| .024 | .039062 | .124 | .164062 | .224 | .289062 | .324 | .414062 |
| .025 | .041015 | .125 | .166015 | .225 | .291015 | .325 | .416015 |
| .026 | .042968 | .126 | .167968 | .226 | .292968 | .326 | .417968 |
| .027 | .044921 | .127 | .169921 | .227 | .294921 | .327 | .419921 |
| .030 | .046875 | .130 | .171875 | .230 | .296875 | .330 | .421875 |
| .031 | .048828 | .131 | .173828 | .231 | .298828 | .331 | .423828 |
| .032 | .050781 | .132 | .175781 | .232 | .300781 | .332 | .425781 |
| .033 | .052734 | .133 | .177734 | .233 | .302734 | .333 | .427734 |
| .034 | .054687 | .134 | .179687 | .234 | .304687 | .334 | .429687 |
| .035 | .056640 | .135 | .181640 | .235 | .306640 | .335 | .431640 |
| .036 | .058593 | .136 | .183593 | .236 | .308593 | .336 | .433593 |
| .037 | .060546 | .137 | .185546 | .237 | .310546 | .337 | .435546 |
| .040 | .062500 | .140 | .187500 | .240 | .312500 | .340 | .437500 |
| .041 | .064453 | .141 | .189453 | .241 | .314453 | .341 | .439453 |
| .042 | .066406 | .142 | .191406 | .242 | .316406 | .342 | .441406 |
| .043 | .068359 | .143 | .193359 | .243 | .318359 | .343 | .443359 |
| .044 | .070312 | .144 | .195312 | .244 | .320312 | .344 | .445312 |
| .045 | .072265 | .145 | .197265 | .245 | .322265 | .345 | .447265 |
| .046 | .074218 | .146 | .199218 | .246 | .324218 | .346 | .449218 |
| .047 | .076171 | .147 | .201171 | .247 | .326171 | .347 | .451171 |
| .050 | .078125 | .150 | .203125 | .250 | .328125 | .350 | .453125 |
| .051 | .080078 | .151 | .205078 | .251 | .330078 | .351 | .455078 |
| .052 | .082031 | .152 | .207031 | .252 | .332031 | .352 | .457031 |
| .053 | .083984 | .153 | .208984 | .253 | .333984 | .353 | .458984 |
| .054 | .085937 | .154 | .210937 | .254 | .335937 | .354 | .460937 |
| .055 | .087890 | .155 | .212890 | .255 | .337890 | .355 | .462890 |
| .056 | .089843 | .156 | .214843 | .256 | .339843 | .356 | .464843 |
| .057 | .091796 | .157 | .216796 | .257 | .341796 | .357 | .466796 |
| .060 | .093750 | .160 | .218750 | .260 | .343750 | .360 | .468750 |
| .061 | .095703 | .161 | .220703 | .261 | .345703 | .361 | .470703 |
| .062 | .097656 | .162 | .222656 | .262 | .347656 | .362 | .472656 |
| .063 | .099609 | .163 | .224609 | .263 | .349609 | .363 | .474609 |
| .064 | .101562 | .164 | .226562 | .264 | .351562 | .364 | .476562 |
| .065 | .103515 | .165 | .228515 | .265 | .353515 | .365 | .478515 |
| .066 | .105468 | .166 | .230468 | .266 | .355468 | .366 | .480468 |
| .067 | .107421 | .167 | .232421 | .267 | .357421 | .367 | .482421 |
| .070 | .109375 | .170 | .234375 | .270 | .359375 | .370 | .484375 |
| .071 | .111328 | .171 | .236328 | .271 | .361328 | .371 | .486328 |
| .072 | .113281 | .172 | .238281 | .272 | .363281 | .372 | .488281 |
| .073 | .115234 | .173 | .240234 | .273 | .365234 | .373 | .490234 |
| .074 | .117187 | .174 | .242187 | .274 | .367187 | .374 | .492187 |
| .075 | .119140 | .175 | .244140 | .275 | .369140 | .375 | .494140 |
| .076 | .121093 | .176 | .246093 | .276 | .371093 | .376 | .496093 |
| .077 | .123046 | .177 | .248046 | .277 | .373046 | .377 | .498046 |

# Octal-Decimal Fraction Conversion Table

| OCTAL | DEC. | OCTAL | DEC. | OCTAL | DEC. | OCTAL | DEC. |
|---|---|---|---|---|---|---|---|
| .000000 | .000000 | .000100 | .000244 | .000200 | .000488 | .000300 | .000732 |
| .000001 | .000003 | .000101 | .000247 | .000201 | .000492 | .000301 | .000736 |
| .000002 | .000007 | .000102 | .000251 | .000202 | .000495 | .000302 | .000740 |
| .000003 | .000011 | .000103 | .000255 | .000203 | .000499 | .000303 | .000743 |
| .000004 | .000015 | .000104 | .000259 | .000204 | .000503 | .000304 | .000747 |
| .000005 | .000019 | .000105 | .000263 | .000205 | .000507 | .000305 | .000751 |
| .000006 | .000022 | .000106 | .000267 | .000206 | .000511 | .000306 | .000755 |
| .000007 | .000026 | .000107 | .000270 | .000207 | .000514 | .000307 | .000759 |
| .000010 | .000030 | .000110 | .000274 | .000210 | .000518 | .000310 | .000762 |
| .000011 | .000034 | .000111 | .000278 | .000211 | .000522 | .000311 | .000766 |
| .000012 | .000038 | .000112 | .000282 | .000212 | .000526 | .000312 | .000770 |
| .000013 | .000041 | .000113 | .000286 | .000213 | .000530 | .000313 | .000774 |
| .000014 | .000045 | .000114 | .000289 | .000214 | .000534 | .000314 | .000778 |
| .000015 | .000049 | .000115 | .000293 | .000215 | .000537 | .000315 | .000782 |
| .000016 | .000053 | .000116 | .000297 | .000216 | .000541 | .000316 | .000785 |
| .000017 | .000057 | .000117 | .000301 | .000217 | .000545 | .000317 | .000789 |
| .000020 | .000061 | .000120 | .000305 | .000220 | .000549 | .000320 | .000793 |
| .000021 | .000064 | .000121 | .000308 | .000221 | .000553 | .000321 | .000797 |
| .000022 | .000068 | .000122 | .000312 | .000222 | .000556 | .000322 | .000801 |
| .000023 | .000072 | .000123 | .000316 | .000223 | .000560 | .000323 | .000805 |
| .000024 | .000076 | .000124 | .000320 | .000224 | .000564 | .000324 | .000808 |
| .000025 | .000080 | .000125 | .000324 | .000225 | .000568 | .000325 | .000812 |
| .000026 | .000083 | .000126 | .000328 | .000226 | .000572 | .000326 | .000816 |
| .000027 | .000087 | .000127 | .000331 | .000227 | .000576 | .000327 | .000820 |
| .000030 | .000091 | .000130 | .000335 | .000230 | .000579 | .000330 | .000823 |
| .000031 | .000095 | .000131 | .000339 | .000231 | .000583 | .000331 | .000827 |
| .000032 | .000099 | .000132 | .000343 | .000232 | .000587 | .000332 | .000831 |
| .000033 | .000102 | .000133 | .000347 | .000233 | .000591 | .000333 | .000835 |
| .000034 | .000106 | .000134 | .000350 | .000234 | .000595 | .000334 | .000839 |
| .000035 | .000110 | .000135 | .000354 | .000235 | .000598 | .000335 | .000843 |
| .000036 | .000114 | .000136 | .000358 | .000236 | .000602 | .000336 | .000846 |
| .000037 | .000118 | .000137 | .000362 | .000237 | .000606 | .000337 | .000850 |
| .000040 | .000122 | .000140 | .000366 | .000240 | .000610 | .000340 | .000854 |
| .000041 | .000125 | .000141 | .000370 | .000241 | .000614 | .000341 | .000858 |
| .000042 | .000129 | .000142 | .000373 | .000242 | .000617 | .000342 | .000862 |
| .000043 | .000133 | .000143 | .000377 | .000243 | .000621 | .000343 | .000865 |
| .000044 | .000137 | .000144 | .000381 | .000244 | .000625 | .000344 | .000869 |
| .000045 | .000141 | .000145 | .000385 | .000245 | .000629 | .000345 | .000873 |
| .000046 | .000144 | .000146 | .000389 | .000246 | .000633 | .000346 | .000877 |
| .000047 | .000148 | .000147 | .000392 | .000247 | .000637 | .000347 | .000881 |
| .000050 | .000152 | .000150 | .000396 | .000250 | .000640 | .000350 | .000885 |
| .000051 | .000156 | .000151 | .000400 | .000251 | .000644 | .000351 | .000888 |
| .000052 | .000160 | .000152 | .000404 | .000252 | .000648 | .000352 | .000892 |
| .000053 | .000164 | .000153 | .000408 | .000253 | .000652 | .000353 | .000896 |
| .000054 | .000167 | .000154 | .000411 | .000254 | .000656 | .000354 | .000900 |
| .000055 | .000171 | .000155 | .000415 | .000255 | .000659 | .000355 | .000904 |
| .000056 | .000175 | .000156 | .000419 | .000256 | .000663 | .000356 | .000907 |
| .000057 | .000179 | .000157 | .000423 | .000257 | .000667 | .000357 | .000911 |
| .000060 | .000183 | .000160 | .000427 | .000260 | .000671 | .000360 | .000915 |
| .000061 | .000186 | .000161 | .000431 | .000261 | .000675 | .000361 | .000919 |
| .000062 | .000190 | .000162 | .000434 | .000262 | .000679 | .000362 | .000923 |
| .000063 | .000194 | .000163 | .000438 | .000263 | .000682 | .000363 | .000926 |
| .000064 | .000198 | .000164 | .000442 | .000264 | .000686 | .000364 | .000930 |
| .000065 | .000202 | .000165 | .000446 | .000265 | .000690 | .000365 | .000934 |
| .000066 | .000205 | .000166 | .000450 | .000266 | .000694 | .000366 | .000938 |
| .000067 | .000209 | .000167 | .000453 | .000267 | .000698 | .000367 | .000942 |
| .000070 | .000213 | .000170 | .000457 | .000270 | .000701 | .000370 | .000946 |
| .000071 | .000217 | .000171 | .000461 | .000271 | .000705 | .000371 | .000949 |
| .000072 | .000221 | .000172 | .000465 | .000272 | .000709 | .000372 | .000953 |
| .000073 | .000225 | .000173 | .000469 | .000273 | .000713 | .000373 | .000957 |
| .000074 | .000228 | .000174 | .000473 | .000274 | .000717 | .000374 | .000961 |
| .000075 | .000232 | .000175 | .000476 | .000275 | .000720 | .000375 | .000965 |
| .000076 | .000236 | .000176 | .000480 | .000276 | .000724 | .000376 | .000968 |
| .000077 | .000240 | .000177 | .000484 | .000277 | .000728 | .000377 | .000972 |

# Octal-Decimal Fraction Conversion Table

| OCTAL | DEC. | OCTAL | DEC. | OCTAL | DEC. | OCTAL | DEC. |
|---|---|---|---|---|---|---|---|
| .000400 | .000976 | .000500 | .001220 | .000600 | .001464 | .000700 | .001708 |
| .000401 | .000980 | .000501 | .001224 | .000601 | .001468 | .000701 | .001712 |
| .000402 | .000984 | .000502 | .001228 | .000602 | .001472 | .000702 | .001716 |
| .000403 | .000988 | .000503 | .001232 | .000603 | .001476 | .000703 | .001720 |
| .000404 | .000991 | .000504 | .001235 | .000604 | .001480 | .000704 | .001724 |
| .000405 | .000995 | .000505 | .001239 | .000605 | .001483 | .000705 | .001728 |
| .000406 | .000999 | .000506 | .001243 | .000606 | .001487 | .000706 | .001731 |
| .000407 | .001003 | .000507 | .001247 | .000607 | .001491 | .000707 | .001735 |
| .000410 | .001007 | .000510 | .001251 | .000610 | .001495 | .000710 | .001739 |
| .000411 | .001010 | .000511 | .001255 | .000611 | .001499 | .000711 | .001743 |
| .000412 | .001014 | .000512 | .001258 | .000612 | .001502 | .000712 | .001747 |
| .000413 | .001018 | .000513 | .001262 | .000613 | .001506 | .000713 | .001750 |
| .000414 | .001022 | .000514 | .001266 | .000614 | .001510 | .000714 | .001754 |
| .000415 | .001026 | .000515 | .001270 | .000615 | .001514 | .000715 | .001758 |
| .000416 | .001029 | .000516 | .001274 | .000616 | .001518 | .000716 | .001762 |
| .000417 | .001033 | .000517 | .001277 | .000617 | .001522 | .000717 | .001766 |
| .000420 | .001037 | .000520 | .001281 | .000620 | .001525 | .000720 | .001770 |
| .000421 | .001041 | .000521 | .001285 | .000621 | .001529 | .000721 | .001773 |
| .000422 | .001045 | .000522 | .001289 | .000622 | .001533 | .000722 | .001777 |
| .000423 | .001049 | .000523 | .001293 | .000623 | .001537 | .000723 | .001781 |
| .000424 | .001052 | .000524 | .001296 | .000624 | .001541 | .000724 | .001785 |
| .000425 | .001056 | .000525 | .001300 | .000625 | .001544 | .000725 | .001789 |
| .000426 | .001060 | .000526 | .001304 | .000626 | .001548 | .000726 | .001792 |
| .000427 | .001064 | .000527 | .001308 | .000627 | .001552 | .000727 | .001796 |
| .000430 | .001068 | .000530 | .001312 | .000630 | .001556 | .000730 | .001800 |
| .000431 | .001071 | .000531 | .001316 | .000631 | .001560 | .000731 | .001804 |
| .000432 | .001075 | .000532 | .001319 | .000632 | .001564 | .000732 | .001808 |
| .000433 | .001079 | .000533 | .001323 | .000633 | .001567 | .000733 | .001811 |
| .000434 | .001083 | .000534 | .001327 | .000634 | .001571 | .000734 | .001815 |
| .000435 | .001087 | .000535 | .001331 | .000635 | .001575 | .000735 | .001819 |
| .000436 | .001091 | .000536 | .001335 | .000636 | .001579 | .000736 | .001823 |
| .000437 | .001094 | .000537 | .001338 | .000637 | .001583 | .000737 | .001827 |
| .000440 | .001098 | .000540 | .001342 | .000640 | .001586 | .000740 | .001831 |
| .000441 | .001102 | .000541 | .001346 | .000641 | .001590 | .000741 | .001834 |
| .000442 | .001106 | .000542 | .001350 | .000642 | .001594 | .000742 | .001838 |
| .000443 | .001110 | .000543 | .001354 | .000643 | .001598 | .000743 | .001842 |
| .000444 | .001113 | .000544 | .001358 | .000644 | .001602 | .000744 | .001846 |
| .000445 | .001117 | .000545 | .001361 | .000645 | .001605 | .000745 | .001850 |
| .000446 | .001121 | .000546 | .001365 | .000646 | .001609 | .000746 | .001853 |
| .000447 | .001125 | .000547 | .001369 | .000647 | .001613 | .000747 | .001857 |
| .000450 | .001129 | .000550 | .001373 | .000650 | .001617 | .000750 | .001861 |
| .000451 | .001132 | .000551 | .001377 | .000651 | .001621 | .000751 | .001865 |
| .000452 | .001136 | .000552 | .001380 | .000652 | .001625 | .000752 | .001869 |
| .000453 | .001140 | .000553 | .001384 | .000653 | .001628 | .000753 | .001873 |
| .000454 | .001144 | .000554 | .001388 | .000654 | .001632 | .000754 | .001876 |
| .000455 | .001148 | .000555 | .001392 | .000655 | .001636 | .000755 | .001880 |
| .000456 | .001152 | .000556 | .001396 | .000656 | .001640 | .000756 | .001884 |
| .000457 | .001155 | .000557 | .001399 | .000657 | .001644 | .000757 | .001888 |
| .000460 | .001159 | .000560 | .001403 | .000660 | .001647 | .000760 | .001892 |
| .000461 | .001163 | .000561 | .001407 | .000661 | .001651 | .000761 | .001895 |
| .000462 | .001167 | .000562 | .001411 | .000662 | .001655 | .000762 | .001899 |
| .000463 | .001171 | .000563 | .001415 | .000663 | .001659 | .000763 | .001903 |
| .000464 | .001174 | .000564 | .001419 | .000664 | .001663 | .000764 | .001907 |
| .000465 | .001178 | .000565 | .001422 | .000665 | .001667 | .000765 | .001911 |
| .000466 | .001182 | .000566 | .001426 | .000666 | .001670 | .000766 | .001914 |
| .000467 | .001186 | .000567 | .001430 | .000667 | .001674 | .000767 | .001918 |
| .000470 | .001190 | .000570 | .001434 | .000670 | .001678 | .000770 | .001922 |
| .000471 | .001194 | .000571 | .001438 | .000671 | .001682 | .000771 | .001926 |
| .000472 | .001197 | .000572 | .001441 | .000672 | .001686 | .000772 | .001930 |
| .000473 | .001201 | .000573 | .001445 | .000673 | .001689 | .000773 | .001934 |
| .000474 | .001205 | .000574 | .001449 | .000674 | .001693 | .000774 | .001937 |
| .000475 | .001209 | .000575 | .001453 | .000675 | .001697 | .000775 | .001941 |
| .000476 | .001213 | .000576 | .001457 | .000676 | .001701 | .000776 | .001945 |
| .000477 | .001216 | .000577 | .001461 | .000677 | .001705 | .000777 | .001949 |

# Appendix E • SCAT Mnemonic Operation Codes

| CODE | COMMENT | INDEXABLE | IND. ADDRESS | PAGE |
|---|---|---|---|---|
| ACL | Add and Carry Logical Word | X | X | 27 |
| ADD | Add | X | X | 26 |
| ADM | Add Magnitude | X | X | 26 |
| ALS | Accumulator Left Shift | X | | 37 |
| ANA | AND to Accumulator | X | X | 54 |
| ANS | AND to Storage | X | X | 54 |
| ARS | Accumulator Right Shift | X | | 38 |
| AXC | Address to Index, Complemented | | | 51 |
| AXT | Address to Index, True | | | 51 |
| BSFA | Backspace File, Ch. A | X | | 64 |
| BSFB | Backspace File, Ch. B | X | | 64 |
| BSFC | Backspace File, Ch. C | X | | 64 |
| BSFD | Backspace File, Ch. D | X | | 64 |
| BSFE | Backspace File, Ch. E | X | | 64 |
| BSFF | Backspace File, Ch. F | X | | 64 |
| BSFG* | Backspace File, Ch. G | X | | 64 |
| BSFH* | Backspace File, Ch. H | X | | 64 |
| BSR | Backspace Record | X | | 64 |
| BSRA | Backspace Record, Ch. A | X | | 64 |
| BSRB | Backspace Record, Ch. B | X | | 64 |
| BSRC | Backspace Record, Ch. C | X | | 64 |
| BSRD | Backspace Record, Ch. D | X | | 64 |
| BSRE | Backspace Record, Ch. E | X | | 64 |
| BSRF | Backspace Record, Ch. F | X | | 64 |
| BSRG* | Backspace Record, Ch. G | X | | 64 |
| BSRH* | Backspace Record, Ch. H | X | | 64 |
| BTTA | Beginning of Tape Test, Ch. A | X | | 47 |
| BTTB | Beginning of Tape Test, Ch. B | X | | 47 |
| BTTC | Beginning of Tape Test, Ch. C | X | | 47 |
| BTTD | Beginning of Tape Test, Ch. D | X | | 47 |
| BTTE | Beginning of Tape Test, Ch. E | X | | 47 |
| BTTF | Beginning of Tape Test, Ch. F | X | | 47 |
| BTTG* | Beginning of Tape Test, Ch. G | X | | 47 |
| BTTH* | Beginning of Tape Test, Ch. H | X | | 47 |
| CAD** | Copy and Add Logical Word | X | | 65 |
| CAL | Clear and Add Logical Word | X | X | 26 |
| CAQ | Convert by Addition from MQ | | | 63 |
| CAS | Compare Accumulator with Storage | X | X | 49 |
| CFF | Change Film Frame | X | | 46 |
| CHS | Change Sign | X | | 55 |
| CLA | Clear and Add | X | X | 26 |
| CLM | Clear Magnitude | X | | 55 |
| CLS | Clear and Subtract | X | X | 26 |
| COM | Complement Magnitude | X | | 55 |
| CPY** | Copy | X | | 65 |
| CRQ | Convert by Replacement from MQ | | | 62 |
| CVR | Convert by Replacement from AC | | | 62 |
| DCT | Divide Check Test | X | | 48 |
| DVH | Divide or Halt | X | X | 30 |
| DVP | Divide or Proceed | X | X | 30 |
| ECTM | Enter Copy Trap Mode | X | | 72 |
| EFTM | Enter Floating Trap Mode | X | | 72 |
| ENB | Enable | X | X | 69 |
| ENK | Enter Keys | X | | 41 |
| ERA | Exclusive OR to Accumulator | X | X | 54 |
| ESNT | Enter Storage Nullification, and Transfer | X | X | 71 |
| ESTM | Enter Select Trap Mode | X | | 71 |
| ETM | Enter Trapping Mode | X | | 42 |
| ETTA | End of Tape Test, Ch. A | X | | 47 |
| ETTB | End of Tape Test, Ch. B | X | | 47 |
| ETTC | End of Tape Test, Ch. C | X | | 47 |
| ETTD | End of Tape Test, Ch. D | X | | 47 |
| ETTE | End of Tape Test, Ch. E | X | | 47 |
| ETTF | End of Tape Test, Ch. F | X | | 47 |
| ETTG* | End of Tape Test, Ch. G | X | | 47 |
| ETTH* | End of Tape Test, Ch. H | X | | 47 |
| FAD | Floating Add | X | X | 32 |
| FAM | Floating Add Magnitude | X | X | 33 |
| FDH | Floating Divide or Halt | X | X | 36 |
| FDP | Floating Divide or Proceed | X | X | 36 |
| FMP | Floating Multiply | X | X | 35 |
| FOR | Four | | | [1] |
| FRN | Floating Round | X | | 34 |
| FSB | Floating Subtract | X | X | 33 |
| FSM | Floating Subtract Magnitude | X | X | 34 |
| FVE | Five | | | [1] |
| HPR | Halt and Proceed | | | 41 |
| HTR | Halt and Transfer | X | X | 42 |
| IIA | Invert Indicators from Accumulator | | | 58 |
| IIL | Invert Indicators of the Left Half | | | 59 |
| IIR | Invert Indicators of the Right Half | | | 59 |
| IIS | Invert Indicators from Storage | X | X | 59 |
| IOCD | Input-Output under Count Control and Disconnect | | | 66 |
| IOCDN | (IOCD with No Transmission) | | | 66 |
| IOCP | Input-Output under Count Control and Proceed | | | 67 |
| IOCPN | (IOCP with No Transmission) | | | 67 |
| IOCT | Input-Output under Count Control and Transfer | | | 67 |
| IOCTN | (IOCT with No Transmission) | | | 67 |
| IORP | Input-Output of a Record and Proceed | | | 67 |
| IORPN | (IORP with No Transmission) | | | 67 |
| IORT | Input-Output of a Record and Transfer | | | 68 |
| IORTN | (IORT with No Transmission) | | | 68 |
| IOSP | Input-Output until Signal then Proceed | | | 68 |
| IOSPN | (IOSP with No Transmission) | | | 68 |
| IOST | Input-Output until Signal then Transfer | | | 68 |
| IOSTN | (IOST with No Transmission) | | | 68 |
| IOT | Input-Output Check Test | X | | 48 |
| LAC | Load Complement of Address in Index | | | 51 |
| LAS | Logical Compare Accumulator with Storage | X | X | 49 |
| LBT | Low-Order Bit Test | X | | 48 |
| LCHA | Load Channel A | X | X | 71 |
| LCHB | Load Channel B | X | X | 71 |
| LCHC | Load Channel C | X | X | 71 |
| LCHD | Load Channel D | X | X | 71 |
| LCHE | Load Channel E | X | X | 71 |
| LCHF | Load Channel F | X | X | 71 |
| LCHG* | Load Channel G | X | X | 71 |
| LCHH* | Load Channel H | X | X | 71 |
| LDA** | Locate Drum Address | X | X | 65 |
| LDC | Load Complement of Decrement in XR | | | 51 |
| LDI | Load Indicators | X | X | 57 |
| LDQ | Load the MQ | X | X | 39 |
| LFT | Left Half Indicators, Off Test | | | 61 |
| LFTM | Leave Floating Trap Mode | X | | 72 |
| LGL | Logical Left Shift | X | | 38 |
| LGR | Logical Right Shift | X | | 38 |
| LLS | Long Left Shift | X | | 38 |
| LNT | Left Half Indicators, On Test | | | 60 |
| LRS | Long Right Shift | X | | 38 |
| LSNM | Leave Storage Nullification Mode | X | | 71 |
| LTM | Leave Trapping Mode | X | | 43 |
| LXA | Load Index from Address | | | 50 |
| LXD | Load Index from Decrement | | | 51 |
| MON | Minus One | | | [1] |
| MPR | Multiply and Round | X | X | 28 |
| MPY | Multiply | X | X | 28 |
| MSE | Minus Sense | X | | 47 |
| MTH | Minus Three | | | [1] |
| MTW | Minus Two | | | [1] |
| MZE | Minus Zero | | | [1] |
| NOP | No Operation | | | 41 |
| NZT | Storage Not-Zero Test | X | X | 49 |

[1] The mnemonic code is an extended code; no particular machine code is concerned with it.

* 7090 Instruction only, not included in this manual.

** 709 Instruction only.

## SCAT Mnemonic Operation Codes *(Cont'd)*

| CODE | COMMENT | INDEXABLE | IND. ADDRESS | PAGE |
|---|---|---|---|---|
| OAI | OR Accumulator to Indicators | | | 57 |
| OFT | Off Test for Indicators | X | X | 60 |
| ONT | On Test for Indicators | X | X | 60 |
| ORA | OR to Accumulator | X | X | 54 |
| ORS | OR to Storage | X | X | 54 |
| OSI | OR Storage to Indicators | X | X | 57 |
| PAC | Place Complement of Address in XR | | | 52 |
| PAI | Place Accumulator in Indicators | | | 56 |
| PAX | Place Address in Index | | | 52 |
| PBT | P-Bit Test | X | | 48 |
| PDC | Place Complement of Decrement in XR | | | 52 |
| PDX | Place Decrement in Index | | | 52 |
| PIA | Place Indicators in Accumulator | | | 57 |
| PON | Plus One | | | (1) |
| PSE | Plus Sense | X | | 46 |
| PTH | Plus Three | | | (1) |
| PTW | Plus Two | | | (1) |
| PXA | Place Index in Address | | | 53 |
| PXD | Place Index in Decrement | | | 53 |
| PZE | Plus Zero | | | (1) |
| RCDA | Read Card Reader, Ch. A | X | | 64 |
| RCDB* | Read Card Reader, Ch. B | X | | 64 |
| RCDC | Read Card Reader, Ch. C | X | | 64 |
| RCDD* | Read Card Reader, Ch. D | X | | 64 |
| RCDE | Read Card Reader, Ch. E | X | | 64 |
| RCDF* | Read Card Reader, Ch. F | X | | 64 |
| RCDG* | Read Card Reader, Ch. G | X | | 64 |
| RCDH* | Read Card Reader, Ch. H | X | | 64 |
| RCHA | Reset and Load, Ch. A | X | X | 70 |
| RCHB | Reset and Load, Ch. B | X | X | 70 |
| RCHC | Reset and Load, Ch. C | X | X | 70 |
| RCHD | Reset and Load, Ch. D | X | X | 70 |
| RCHE | Reset and Load, Ch. E | X | X | 70 |
| RCHF | Reset and Load, Ch. F | X | X | 70 |
| RCHG* | Reset and Load, Ch. G | X | X | 70 |
| RCHH* | Reset and Load, Ch. H | X | X | 70 |
| RCT | Restore Channel Traps | X | | 70 |
| RDR** | Read Drum | X | | 64 |
| RDS | Read Select | X | | 65 |
| REWA | Rewind, Ch. A | X | | 65 |
| REWB | Rewind, Ch. B | X | | 65 |
| REWC | Rewind, Ch. C | X | | 65 |
| REWD | Rewind, Ch. D | X | | 65 |
| REWE | Rewind, Ch. E | X | | 65 |
| REWF | Rewind, Ch. F | X | | 65 |
| REWG* | Rewind, Ch. G | X | | 65 |
| REWH* | Rewind, Ch. H | X | | 65 |
| RFT | Right Half Indicators, Off Test | | | 61 |
| RIA | Reset Indicators from Accumulator | | | 58 |
| RIL | Reset Indicators of Left Half | | | 58 |
| RIR | Reset Indicators of Right Half | | | 58 |
| RIS | Reset Indicators from Storage | X | X | 58 |
| RND | Round | X | | 28 |
| RNT | Right Half Indicators, On Test | | | 60 |
| RPRA | Read Printer, Ch. A | X | | 64 |
| RPRB* | Read Printer, Ch. B | X | | 64 |
| RPRC | Read Printer, Ch. C | X | | 64 |
| RPRD* | Read Printer, Ch. D | X | | 64 |
| RPRE | Read Printer, Ch. E | X | | 64 |
| RPRF* | Read Printer, Ch. F | X | | 64 |
| RPRG* | Read Printer, Ch. G | X | | 64 |
| RPRH* | Read Printer, Ch. H | X | | 64 |
| RQL | Rotate MQ Left | X | | 38 |
| RTBA | Read Tape Binary, Ch. A | X | | 64 |
| RTBB | Read Tape Binary, Ch. B | X | | 64 |
| RTBC | Read Tape Binary, Ch. C | X | | 64 |
| RTBD | Read Tape Binary, Ch. D | X | | 64 |
| RTBE | Read Tape Binary, Ch. E | X | | 64 |
| RTBF | Read Tape Binary, Ch. F | X | | 64 |
| RTBG* | Read Tape Binary, Ch. G | X | | 64 |
| RTBH* | Read Tape Binary, Ch. H | X | | 64 |
| RTDA | Read Tape Decimal, Ch. A | X | | 64 |
| RTDB | Read Tape Decimal, Ch. B | X | | 64 |
| RTDC | Read Tape Decimal, Ch. C | X | | 64 |
| RTDD | Read Tape Decimal, Ch. D | X | | 64 |
| RTDE | Read Tape Decimal, Ch. E | X | | 64 |
| RTDF | Read Tape Decimal, Ch. F | X | | 64 |
| RTDG* | Read Tape Decimal, Ch. G | X | | 64 |
| RTDH* | Read Tape Decimal, Ch. H | X | | 64 |
| RUNA* | Rewind and Unload Channel A | X | | |
| RUNB* | Rewind and Unload Channel B | X | | |
| RUNC* | Rewind and Unload Channel C | X | | |
| RUND* | Rewind and Unload Channel D | X | | |
| RUNE* | Rewind and Unload Channel E | X | | |
| RUNF* | Rewind and Unload Channel F | X | | |
| RUNG* | Rewind and Unload Channel G | X | | |
| RUNH* | Rewind and Unload Channel H | X | | |
| SBM | Subtract Magnitude | X | X | 27 |
| SCHA | Store, Ch. A | X | X | 70 |
| SCHB | Store, Ch. B | X | X | 70 |
| SCHC | Store, Ch. C | X | X | 70 |
| SCHD | Store, Ch. D | X | X | 70 |
| SCHE | Store, Ch. E | X | X | 70 |
| SCHF | Store, Ch. F | X | X | 70 |
| SCHG* | Store, Ch. G | X | X | 70 |
| SCHH* | Store, Ch. H | X | X | 70 |
| SDLA* | Set Density Low, Channel A | X | | |
| SDLB* | Set Density Low, Channel B | X | | |
| SDLC* | Set Density Low Channel C | X | | |
| SDLD* | Set Density Low, Channel D | X | | |
| SDLE* | Set Density Low, Channel E | X | | |
| SDLF* | Set Density Low, Channel F | X | | |
| SDLG* | Set Density Low, Channel G | X | | |
| SDLH* | Set Density Low, Channel H | X | | |
| SDHA* | Set Density High, Channel A | X | | |
| SDHB* | Set Density High, Channel B | X | | |
| SDHC* | Set Density High, Channel C | X | | |
| SDHD* | Set Density High, Channel D | X | | |
| SDHE* | Set Density High, Channel E | X | | |
| SDHF* | Set Density High, Channel F | X | | |
| SDHG* | Set Density High, Channel G | X | | |
| SDHH* | Set Density High, Channel H | X | | |
| SIL | Set Indicators of Left Half | | | 57 |
| SIR | Set Indicators of Right Half | | | 57 |
| SIX | Six | | | (1) |
| SLF | Sense Lights Off | X | | 46 |
| SLN | Sense Lights On | X | | 46 |
| SLQ | Store Left Half MQ | X | X | 39 |
| SLT | Sense Light Test | X | | 46 |
| SLW | Store Logical Word | X | X | 39 |
| SPRA | Sense Printer, Ch. A | X | | 46 |
| SPRB* | Sense Printer, Ch. B | X | | 46 |
| SPRC | Sense Printer, Ch. C | X | | 46 |
| SPRD* | Sense Printer, Ch. D | X | | 46 |
| SPRE | Sense Printer, Ch. E | X | | 46 |
| SPRF* | Sense Printer, Ch. F | X | | 46 |
| SPRG* | Sense Printer, Ch. G | X | | 46 |
| SPRH* | Sense Printer, Ch. H | X | | 46 |
| SPTA | Sense Printer Test, Ch. A | X | | 46 |
| SPTB* | Sense Printer Test, Ch. B | X | | 46 |
| SPTC | Sense Printer Test, Ch. C | X | | 46 |
| SPTD* | Sense Printer Test, Ch. D | X | | 46 |
| SPTE | Sense Printer Test, Ch. E | X | | 46 |
| SPTF* | Sense Printer Test, Ch. F | X | | 46 |
| SPTG* | Sense Printer Test, Ch. G | X | | 46 |
| SPTH* | Sense Printer Test, Ch. H | X | | 46 |
| SPUA | Sense Punch, Ch. A | X | | 46 |

(1) The mnemonic code is an extended code; no particular machine code is concerned with it.

* 7090 Instruction only, not included in this manual.

** 709 Instruction only.

## SCAT Mnemonic Operation Codes *(Cont'd)*

| CODE | COMMENT | INDEXABLE | IND. ADDRESS | PAGE |
|---|---|---|---|---|
| SPUB* | Sense Punch, Ch. B | X | | 46 |
| SPUC | Sense Punch, Ch. C | X | | 46 |
| SPUD* | Sense Punch, Ch. D | X | | 46 |
| SPUE | Sense Punch, Ch. E | X | | 46 |
| SPUF* | Sense Punch, Ch. F | X | | 46 |
| SPUG* | Sense Punch, Ch. G | X | | 46 |
| SPUH* | Sense Punch, Ch. H | X | | 46 |
| SSM | Set Sign Minus | X | | 56 |
| SSP | Set Sign Plus | X | | 56 |
| STA | Store Address | X | X | 40 |
| STD | Store Decrement | X | X | 40 |
| STI | Store Indicators | X | X | 57 |
| STL | Store Instruction Location Counter | X | X | 40 |
| STO | Store | X | X | 39 |
| STP | Store Prefix | X | X | 39 |
| STQ | Store MQ | X | X | 39 |
| STR | Store Location and Trap | | | 40 |
| STT | Store Tag | X | X | 40 |
| STZ | Store Zero | X | X | 40 |
| SUB | Subtract | X | X | 27 |
| SVN | Seven | | | (1) |
| SWT | Sense Switch Test | X | | 46 |
| SXA | Store Index in Address | | | 52 |
| SXD | Store Index in Decrement | | | 53 |
| TCH | Transfer in Channel | | | 69 |
| TCNA | Transfer on Ch. A Not in Operation | X | X | 50 |
| TCNB | Transfer on Ch. B Not in Operation | X | X | 50 |
| TCNC | Transfer on Ch. C Not in Operation | X | X | 50 |
| TCND | Transfer on Ch. D Not in Operation | X | X | 50 |
| TCNE | Transfer on Ch. E Not in Operation | X | X | 50 |
| TCNF | Transfer on Ch. F Not in Operation | X | X | 50 |
| TCNG* | Transfer on Ch. G Not in Operation | X | X | 50 |
| TCNH* | Transfer on Ch. H Not in Operation | X | X | 50 |
| TCOA | Transfer on Ch. A in Operation | X | X | 49 |
| TCOB | Transfer on Ch. B in Operation | X | X | 49 |
| TCOC | Transfer on Ch. C in Operation | X | X | 49 |
| TCOD | Transfer on Ch. D in Operation | X | X | 49 |
| TCOE | Transfer on Ch. E in Operation | X | X | 49 |
| TCOF | Transfer on Ch. F in Operation | X | X | 49 |
| TCOG* | Transfer on Ch. G in Operation | X | X | 49 |
| TCOH* | Transfer on Ch. H in Operation | X | X | 49 |
| TEFA | Transfer on End of File, Ch. A | X | X | 50 |
| TEFB | Transfer on End of File, Ch. B | X | X | 50 |
| TEFC | Transfer on End of File, Ch. C | X | X | 50 |
| TEFD | Transfer on End of File, Ch. D | X | X | 50 |
| TEFE | Transfer on End of File, Ch. E | X | X | 50 |
| TEFF | Transfer on End of File, Ch. F | X | X | 50 |
| TEFG* | Transfer on End of File, Ch. G | X | X | 50 |
| TEFH* | Transfer on End of File, Ch. H | X | X | 50 |
| TIF | Transfer if Indicators Off | X | X | 59 |
| TIO | Transfer if Indicators On | X | X | 59 |
| TIX | Transfer on Index | | | 46 |
| TLQ | Transfer on Low MQ | X | X | 45 |
| TMI | Transfer on Minus | X | X | 44 |
| TNO | Transfer on No Overflow | X | X | 44 |
| TNX | Transfer on No Index | | | 46 |
| TNZ | Transfer on No Zero | X | X | 43 |
| TOV | Transfer on Overflow | X | X | 44 |
| TPL | Transfer on Plus | X | X | 44 |
| TQO | Transfer on Quotient Overflow | X | X | 44 |
| TQP | Transfer on MQ Plus | X | X | 44 |
| TRA | Transfer | X | X | 42 |
| TRCA | Transfer on Redun. Check, Ch. A | X | X | 50 |
| TRCB | Transfer on Redun. Check, Ch. B | X | X | 50 |
| TRCC | Transfer on Redun. Check, Ch. C | X | X | 50 |
| TRCD | Transfer on Redun. Check, Ch. D | X | X | 50 |
| TRCE | Transfer on Redun. Check, Ch. E | X | X | 50 |
| TRCF | Transfer on Redun. Check, Ch. F | X | X | 50 |
| TRCG* | Transfer on Redun. Check, Ch. G | X | X | 50 |
| TRCH* | Transfer on Redun. Check, Ch. H | X | X | 50 |
| TSX | Transfer and Set Index | | | 45 |
| TTR | Trap Transfer | X | X | 43 |
| TXH | Transfer on Index High | | | 45 |
| TXI | Transfer with Index Incremented | | | 45 |
| TXL | Transfer on Index Low or Equal | | | 46 |
| TZE | Transfer on Zero | X | X | 43 |
| UAM | Unnormalized Add Magnitude | X | X | 34 |
| UFA | Unnormalized Floating Add | X | X | 33 |
| UFM | Unnormalized Floating Multiply | X | X | 35 |
| UFS | Unnormalized Floating Subtract | X | X | 34 |
| USM | Unnormalized Subtract Magnitude | X | X | 34 |
| VDH | Variable Length Divide or Halt | X | | 30 |
| VDP | Variable Length Divide or Proceed | X | | 30 |
| VLM | Variable Length Multiply | X | | 28 |
| WDR** | Write Drum | X | | 64 |
| WEF | Write End of File | X | | 65 |
| WEFA | Write End of File, Ch. A | X | | 65 |
| WEFB | Write End of File, Ch. B | X | | 65 |
| WEFC | Write End of File, Ch. C | X | | 65 |
| WEFD | Write End of File, Ch. D | X | | 65 |
| WEFE | Write End of File, Ch. E | X | | 65 |
| WEFF | Write End of File, Ch. F | X | | 65 |
| WEFG* | Write End of File, Ch. G | X | | 65 |
| WEFH* | Write End of File, Ch. H | X | | 65 |
| WPBA | Write Printer Binary, Ch. A | X | | 64 |
| WPBB* | Write Printer Binary, Ch. B | X | | 64 |
| WPBC | Write Printer Binary, Ch. C | X | | 64 |
| WPBD* | Write Printer Binary, Ch. D | X | | 64 |
| WPBE | Write Printer Binary, Ch. E | X | | 64 |
| WPBF* | Write Printer Binary, Ch. F | X | | 64 |
| WPBG* | Write Printer Binary, Ch. G | X | | 64 |
| WPBH* | Write Printer Binary, Ch. H | X | | 64 |
| WPDA | Write Printer Decimal, Ch. A | X | | 64 |
| WPDB* | Write Printer Decimal, Ch. B | X | | 64 |
| WPDC | Write Printer Decimal, Ch. C | X | | 64 |
| WPDD* | Write Printer Decimal, Ch. D | X | | 64 |
| WPDE | Write Printer Decimal, Ch. E | X | | 64 |
| WPDF* | Write Printer Decimal, Ch. F | X | | 64 |
| WPDG* | Write Printer Decimal, Ch. G | X | | 64 |
| WPDH* | Write Printer Decimal, Ch. H | X | | 64 |
| WPUA | Write Punch, Ch. A | X | | 64 |
| WPUB* | Write Punch, Ch. B | X | | 64 |
| WPUC | Write Punch, Ch. C | X | | 64 |
| WPUD* | Write Punch, Ch. D | X | | 64 |
| WPUE | Write Punch, Ch. E | X | | 64 |
| WPUF* | Write Punch, Ch. F | X | | 64 |
| WPUG* | Write Punch, Ch. G | X | | 64 |
| WPUH* | Write Punch, Ch. H | X | | 64 |
| WRS | Write Select | X | | 64 |
| WTBA | Write Tape Binary, Ch. A | X | | 64 |
| WTBB | Write Tape Binary, Ch. B | X | | 64 |
| WTBC | Write Tape Binary, Ch. C | X | | 64 |
| WTBD | Write Tape Binary, Ch. D | X | | 64 |
| WTBE | Write Tape Binary, Ch. E | X | | 64 |
| WTBF | Write Tape Binary, Ch. F | X | | 64 |
| WTBG* | Write Tape Binary, Ch. G | X | | 64 |
| WTBH* | Write Tape Binary, Ch. H | X | | 64 |
| WTDA | Write Tape Decimal, Ch. A | X | | 64 |
| WTDB | Write Tape Decimal, Ch. B | X | | 64 |
| WTDC | Write Tape Decimal, Ch. C | X | | 64 |
| WTDD | Write Tape Decimal, Ch. D | X | | 64 |
| WTDE | Write Tape Decimal, Ch. E | X | | 64 |
| WTDF | Write Tape Decimal, Ch. F | X | | 64 |
| WTDG* | Write Tape Decimal, Ch. G | X | | 64 |
| WTDH* | Write Tape Decimal, Ch. H | X | | 64 |
| WTV** | Write Cathode Ray Tube | X | | 64 |
| XCA | Exchange Accumulator and MQ | | | 40 |
| XCL | Exchange Logical Accumulator and MQ | | | 40 |
| XEC | Execute | X | X | 42 |
| ZET | Storage Zero Test | X | X | 49 |

(1) The mnemonic code is an extended code; no particular machine code is concerned with it.

* 7090 Instruction only, not included in this manual.

** 709 Instruction only.

# Appendix F • Listing of Instructions

## Alphabetic Listing

| OPERATION CODE ALPHA | OCTAL | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| ACL | 0361 | Add and Carry Logical Word | | X | X | 27 |
| ADD | 0400 | Add | | X | X | 26 |
| ADM | 0401 | Add Magnitude | | X | X | 26 |
| ALS | 0767 | Accumulator Left Shift | 7 | X | | 37 |
| ANA | —0320 | AND to Accumulator | | X | X | 54 |
| ANS | 0320 | AND to Storage | | X | X | 54 |
| ARS | 0771 | Accumulator Right Shift | 7 | X | | 38 |
| AXC | —0774 | Address to Index Complemented | | | | 51 |
| AXT | 0774 | Address to Index True | | | | 51 |
| BSF | —0764 | Backspace File | 8 | X | | 64 |
| BSR | 0764 | Backspace Record | 8 | X | | 64 |
| BTT | 0760..xxxx | Beginning of Tape Test | | X | | 47 |
| CAD | —0700 | Copy and Add Logical Word | 8 | X | | 65 |
| CAL | —0500 | Clear and Add Logical Word | | X | X | 26 |
| CAQ | —0114 | Convert by Addition from MQ | 6 | | | 63 |
| CAS | 0340 | Compare AC with Storage | | X | X | 49 |
| CHS | 0760..0002 | Change Sign | | X | | 55 |
| CLA | 0500 | Clear and Add | | X | X | 26 |
| CLM | 0760..0000 | Clear Magnitude | | X | | 55 |
| CLS | 0502 | Clear and Subtract | | X | X | 26 |
| COM | 0760..0006 | Complement Magnitude | | X | | 55 |
| CPY | 0700 | Copy | 8 | X | | 65 |
| CRQ | —0154 | Convert by Replacement from MQ | 6 | | | 62 |
| CVR | 0114 | Convert by Replacement from AC | 6 | | | 62 |
| DCT | 0760..0012 | Divide Check Test | | X | | 48 |
| DVH | 0220 | Divide or Halt | | X | X | 30 |
| DVP | 0221 | Divide or Proceed | | X | X | 30 |
| ECTM | —0760..0006 | Enter Copy Trap Mode | | X | | 72 |
| EFTM | —0760..0002 | Enter Floating Trap Mode | | X | | 72 |
| ENB | 0564 | Enable | | X | X | 69 |
| ENK | 0760..0004 | Enter Keys | | X | | 41 |
| ERA | 0322 | Exclusive OR to Accumulator | | X | X | 54 |
| ESNT | —0021 | Enter Storage Null. and Transfer | | X | X | 71 |
| ESTM | —0760..0005 | Enter Select Trap Mode | | X | | 71 |
| ETM | 0760..0007 | Enter Trapping Mode | | X | | 42 |
| ETT | —0760..xxxx | End of Tape Test | | X | | 47 |
| FAD | 0300 | Floating Add | 3 | X | X | 32 |
| FAM | 0304 | Floating Add Magnitude | 3 | X | X | 33 |
| FDH | 0240 | Floating Divide or Halt | 5 | X | X | 36 |
| FDP | 0241 | Floating Divide or Proceed | 5 | X | X | 36 |
| FMP | 0260 | Floating Multiply | 1 | X | X | 35 |
| FRN | 0760..0011 | Floating Round | | X | | 34 |
| FSB | 0302 | Floating Subtract | 3 | X | X | 33 |
| FSM | 0306 | Floating Subtract Magnitude | 3 | X | X | 34 |
| HPR | 0420 | Halt and Proceed | | | | 41 |
| HTR | 0000 | Halt and Transfer | | X | X | 42 |
| IIA | 0041 | Invert Indicators from AC | | | | 58 |
| IIL | —0051 | Invert Indicators of Left Half | | | | 59 |
| IIR | 0051 | Invert Indicators of Right Half | | | | 59 |
| IIS | 0440 | Invert Indicators from Storage | | X | X | 59 |
| IOT | 0760..0005 | Input-Output Check Test | | X | | 48 |
| LAC | 0535 | Load Complement of Address in Index | | | | 51 |
| LAS | —0340 | Logical Compare Accumulator with Storage | | X | X | 49 |
| LBT | 0760..0001 | Low-Order Bit Test | | X | | 48 |
| LCHA | 0544 | Load Channel A | 8 | X | X | 71 |
| LCHB | —0544 | Load Channel B | 8 | X | X | 71 |
| LCHC | 0545 | Load Channel C | 8 | X | X | 71 |

## Alphabetic *(Continued)*

| OPERATION CODE ALPHA | OCTAL | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| LCHD | —0545 | Load Channel D | 8 | X | X | 71 |
| LCHE | 0546 | Load Channel E | 8 | X | X | 71 |
| LCHF | —0546 | Load Channel F | 8 | X | X | 71 |
| LDA | 0460 | Locate Drum Address | 8 | X | X | 65 |
| LDC | —0535 | Load Complement of Decrement in XR | | | | 51 |
| LDI | 0441 | Load Indicators | | X | X | 57 |
| LDQ | 0560 | Load MQ | | X | X | 39 |
| LFT | —0054 | Left Half Indicators, Off Test | | | | 61 |
| LFTM | —0760..0004 | Leave Floating Trap Mode | | X | | 72 |
| LGL | —0763 | Logical Left Shift | 7 | X | | 38 |
| LGR | —0765 | Logical Right Shift | 7 | X | | 38 |
| LLS | 0763 | Long Left Shift | 7 | X | | 38 |
| LNT | —0056 | Left Half Indicators, On Test | | | | 60 |
| LRS | 0765 | Long Right Shift | 7 | X | | 38 |
| LSNM | —0760..0010 | Leave Storage Nullification Mode | | X | | 71 |
| LTM | —0760..0007 | Leave Trapping Mode | | X | | 43 |
| LXA | 0534 | Load Index from Address | | | | 50 |
| LXD | —0534 | Load Index from Decrement | | | | 51 |
| MPR | —0200 | Multiply and Round | 1 | X | X | 28 |
| MPY | 0200 | Multiply | 1 | X | X | 28 |
| MSE | —0760 | Minus Sense | | X | | 47 |
| NOP | 0761 | No Operation | | | | 41 |
| NZT | —0520 | Storage Non-zero Test | | X | X | 49 |
| OAI | 0043 | OR Accumulator to Indicators | | | | 57 |
| OFT | 0444 | Off Test for Indicators | | X | X | 60 |
| ONT | 0446 | On Test for Indicators | | X | X | 60 |
| ORA | —0501 | OR to Accumulator | | X | X | 54 |
| ORS | —0602 | OR to Storage | | X | X | 54 |
| OSI | 0442 | OR Storage to Indicators | | X | X | 57 |
| PAC | 0737 | Place Complement of Address in XR | | | | 52 |
| PAI | 0044 | Place Accumulator in Indicators | | | | 56 |
| PAX | 0734 | Place Address in XR | | | | 52 |
| PBT | —0760..0001 | P-bit Test | | X | | 48 |
| PDC | —0737 | Place Complement of Decrement in XR | | | | 52 |
| PDX | —0734 | Place Decrement in Index | | | | 52 |
| PIA | —0046 | Place Indicator in Accumulator | | | | 57 |
| PSE | 0760 | Plus Sense | | X | | 46 |
| PXA | 0754 | Place Index in Address | | | | 53 |
| PXD | —0754 | Place Index in Decrement | | | | 53 |
| RCHA | 0540 | Reset and Load Channel A | | X | X | 70 |
| RCHB | —0540 | Reset and Load Channel B | | X | X | 70 |
| RCHC | 0541 | Reset and Load Channel C | | X | X | 70 |
| RCHD | —0541 | Reset and Load Channel D | | X | X | 70 |
| RCHE | 0542 | Reset and Load Channel E | | X | X | 70 |
| RCHF | —0542 | Reset and Load Channel F | | X | X | 70 |
| RCT | 0760..0014 | Restore Channel Traps | | X | | 70 |
| RDS | 0762 | Read Select | 8 | X | | 64 |
| REW | 0772 | Rewind | 8 | X | | 65 |
| RFT | 0054 | Right Half Indicators, Off Test | | | | 61 |
| RIA | —0042 | Reset Indicators from Accumulator | | | | 58 |
| RIL | —0057 | Reset Indicators of Left Half | | | | 58 |
| RIR | 0057 | Reset Indicators of Right Half | | | | 58 |
| RIS | 0445 | Reset Indicators from Storage | | X | X | 58 |
| RND | 0760..0010 | Round | | X | X | 28 |
| RNT | 0056 | Right Half Indicators, On Test | | | | 60 |

## Alphabetic *(Continued)*

| OPERATION CODE ALPHA | OCTAL | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| RQL | —0773 | Rotate MQ Left | 7 | X | | 38 |
| SBM | —0400 | Subtract Magnitude | | X | X | 27 |
| SCHA | 0640 | Store Channel A | | X | X | 70 |
| SCHB | —0640 | Store Channel B | | X | X | 70 |
| SCHC | 0641 | Store Channel C | | X | X | 70 |
| SCHD | —0641 | Store Channel D | | X | X | 70 |
| SCHE | 0642 | Store Channel E | | X | X | 70 |
| SCHF | —0642 | Store Channel F | | X | X | 70 |
| SIL | —0055 | Set Indicator of Left Half | | | | 57 |
| SIR | 0055 | Set Indicator of Right Half | | | | 57 |
| SLQ | —0620 | Store Left Half MQ | | X | X | 39 |
| SLW | 0602 | Store Logical Word | | X | X | 39 |
| SSM | —0760..0003 | Set Sign Minus | | X | | 56 |
| SSP | 0760..0003 | Set Sign Plus | | X | | 56 |
| STA | 0621 | Store Address | | X | X | 40 |
| STD | 0622 | Store Decrement | | X | X | 40 |
| STI | 0604 | Store Indicators | | X | X | 57 |
| STL | —0625 | Store Instruction Location Counter | | X | X | 40 |
| STO | 0601 | Store | | X | X | 39 |
| STP | 0630 | Store Prefix | | X | X | 39 |
| STQ | —0600 | Store MQ | | X | X | 39 |
| STR | —1000 | Store Location and Trap | | | | 40 |
| STT | 0625 | Store Tag | | X | X | 40 |
| STZ | 0600 | Store Zero | | X | X | 40 |
| SUB | 0402 | Subtract | | X | X | 27 |
| SXA | 0634 | Store Index in Address | | | | 52 |
| SXD | —0634 | Store Index in Decrement | | | | 53 |
| TCNA | —0060 | Transfer on DSC A Not in Operation | | X | X | 48 |
| TCNB | —0061 | Transfer on DSC B Not in Operation | | X | X | 48 |
| TCNC | —0062 | Transfer on DSC C Not in Operation | | X | X | 48 |
| TCND | —0063 | Transfer on DSC D Not in Operation | | X | X | 48 |
| TCNE | —0064 | Transfer on DSC E Not in Operation | | X | X | 48 |
| TCNF | —0065 | Transfer on DSC F Not in Operation | | X | X | 48 |
| TCOA | 0060 | Transfer on DSC A in Operation | | X | X | 49 |
| TCOB | 0061 | Transfer on DSC B in Operation | | X | X | 49 |
| TCOC | 0062 | Transfer on DSC C in Operation | | X | X | 49 |
| TCOD | 0063 | Transfer on DSC D in Operation | | X | X | 49 |
| TCOE | 0064 | Transfer on DSC E in Operation | | X | X | 49 |
| TCOF | 0065 | Transfer on DSC F in Operation | | X | X | 49 |
| TEFA | 0030 | Transfer on DSC A End of File | | X | X | 50 |
| TEFB | —0030 | Transfer on DSC B End of File | | X | X | 50 |
| TEFC | 0031 | Transfer on DSC C End of File | | X | X | 50 |
| TEFD | —0031 | Transfer on DSC D End of File | | X | X | 50 |

## Alphabetic *(Continued)*

| OPERATION CODE ALPHA | OCTAL | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| TEFE | 0032 | Transfer on DSC E End of File | | X | X | 50 |
| TEFF | —0032 | Transfer on DSC F End of File | | X | X | 50 |
| TIF | 0046 | Transfer if Indicators Off | | X | X | 59 |
| TIO | 0042 | Transfer if Indicators On | | X | X | 59 |
| TIX | 2000 | Transfer on Index | | | | 46 |
| TLQ | 0040 | Transfer on Low MQ | | X | X | 45 |
| TMI | —0120 | Transfer on Minus | | X | X | 44 |
| TNO | —0140 | Transfer on No Overflow | | X | X | 44 |
| TNX | —2000 | Transfer on No Index | | | | 46 |
| TNZ | —0100 | Transfer on No Zero | | X | X | 43 |
| TOV | 0140 | Transfer on Overflow | | X | X | 44 |
| TPL | 0120 | Transfer on Plus | | X | X | 44 |
| TQO | 0161 | Transfer on Quotient Overflow | | X | X | 44 |
| TQP | 0162 | Transfer on MQ Plus | | X | X | 44 |
| TRA | 0020 | Transfer | | X | X | 42 |
| TRCA | 0022 | Transfer on DSC A Redundancy Check | | X | X | 50 |
| TRCB | —0022 | Transfer on DSC B Redundancy Check | | X | X | 50 |
| TRCC | 0024 | Transfer on DSC C Redundancy Check | | X | X | 50 |
| TRCD | —0024 | Transfer on DSC D Redundancy Check | | X | X | 50 |
| TRCE | 0026 | Transfer on DSC E Redundancy Check | | X | X | 50 |
| TRCF | —0026 | Transfer on DSC F Redundancy Check | | X | X | 50 |
| TSX | 0074 | Transfer and Set Index | | | | 45 |
| TTR | 0021 | Trap Transfer | | X | X | 43 |
| TXH | 3000 | Transfer on Index High | | | | 45 |
| TXI | 1000 | Transfer with XR Increments | | | | 45 |
| TXL | —3000 | Transfer on XR Low or Equal | | | | 46 |
| TZE | 0100 | Transfer on Zero | | X | X | 43 |
| UAM | —0304 | Unnormalized Add Magnitude | 4 | X | X | 34 |
| UFA | —0300 | Unnormalized Floating Add | 4 | X | X | 33 |
| UFM | —0260 | Unnormalized Floating Multiply | 1 | X | X | 35 |
| UFS | —0302 | Unnormalized Floating Subtract | 4 | X | X | 34 |
| USM | —0306 | Unnormalized Subtract Magnitude | 4 | X | X | 34 |
| VDH | 0224 | Variable Length Divide or Halt | 2 | X | | 30 |
| VDP | 0225 | Variable Length Divide or Proceed | 2 | X | | 30 |
| VLM | 0204 | Variable Length Multiply | 1, 2 | X | | 28 |
| WEF | 0770 | Write End of File | 8 | X | | 65 |
| WRS | 0766 | Write Select | 8 | X | | 64 |
| XCA | 0131 | Exchange AC and MQ | | | | 40 |
| XCL | —0130 | Exchange Logical AC and MQ | | | | 40 |
| XEC | 0522 | Execute | | X | X | 42 |
| ZET | 0520 | Storage Zero Test | | X | X | 49 |

## Numerical Listing

| OPERATION CODE ALPHA | OCTAL | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| HTR | 0000 | Halt and Transfer | | X | X | 41 |
| TRA | 0020 | Transfer | | X | X | 42 |
| TTR | 0021 | Trap Transfer | | X | X | 43 |
| ESNT | —0021 | Enter Storage Null. and Transfer | | X | X | 71 |
| TRCA | 0022 | Transfer on DSC A Redundancy Check | | X | X | 50 |
| TRCB | —0022 | Transfer on DSC B Redundancy Check | | X | X | 50 |
| TRCC | 0024 | Transfer on DSC C Redundancy Check | | X | X | 50 |
| TRCD | —0024 | Transfer on DSC D Redundancy Check | | X | X | 50 |
| TRCE | 0026 | Transfer on DSC E Redundancy Check | | X | X | 50 |
| TRCF | —0026 | Transfer on DSC F Redundancy Check | | X | X | 50 |
| TEFA | 0030 | Transfer on DSC A End of File | | X | X | 50 |
| TEFB | —0030 | Transfer on DSC B End of File | | X | X | 50 |
| TEFC | 0031 | Transfer on DSC C End of File | | X | X | 50 |
| TEFD | —0031 | Transfer on DSC D End of File | | X | X | 50 |
| TEFE | 0032 | Transfer on DSC E End of File | | X | X | 50 |
| TEFF | —0032 | Transfer on DSC F End of File | | X | X | 50 |
| TLQ | 0040 | Transfer on Low MQ | | X | X | 45 |
| IIA | 0041 | Invert Indicators from AC | | | | 58 |
| TIO | 0042 | Transfer if Indicators On | | X | X | 59 |
| RIA | —0042 | Reset Indicators from AC | | | | 58 |
| OAI | 0043 | OR Accumulator to Indicators | | | | 57 |
| PAI | 0044 | Place Accumulator in Indicators | | | | 56 |
| TIF | 0046 | Transfer if Indicators Off | | X | X | 59 |
| PIA | —0046 | Place Indicators in Accumulator | | | | 57 |
| IIR | 0051 | Invert Indicators of Right Half | | | | 59 |
| IIL | —0051 | Invert Indicators of Left Half | | | | 59 |
| RFT | 0054 | Right Half Indicators, Off Test | | | | 61 |
| LFT | —0054 | Left Half Indicators, Off Test | | | | 61 |
| SIR | 0055 | Set Indicator of Right Half | | | | 57 |
| SIL | —0055 | Set Indicator of Left Half | | | | 57 |
| RNT | 0056 | Right Half Indicators, On Test | | | | 60 |
| LNT | —0056 | Left Half Indicators, On Test | | | | 60 |
| RIR | 0057 | Reset Indicators of Right Half | | | | 58 |
| RIL | —0057 | Reset Indicators on Left Half | | | | 58 |
| TCNA | —0060 | Transfer on DSC A Not in Operation | | X | X | 50 |
| TCNB | —0061 | Transfer on DSC B Not in Operation | | X | X | 50 |
| TCNC | —0062 | Transfer on DSC C Not in Operation | | X | X | 50 |
| TCND | —0063 | Transfer on DSC D Not in Operation | | X | X | 50 |
| TCNE | —0064 | Transfer on DSC E Not in Operation | | X | X | 50 |
| TCNF | —0065 | Transfer on DSC F Not in Operation | | X | X | 50 |
| TCOA | 0060 | Transfer on DSC A in Operation | | X | X | 49 |
| TCOB | 0061 | Transfer on DSC B in Operation | | X | X | 49 |
| TCOC | 0062 | Transfer on DSC C in Operation | | X | X | 49 |

## Numerical *(Continued)*

| OPERATION CODE ALPHA | OCTAL | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| TCOD | 0063 | Transfer on DSC D in Operation | | X | X | 49 |
| TCOE | 0064 | Transfer on DSC E in Operation | | X | X | 49 |
| TCOF | 0065 | Transfer on DSC F in Operation | | X | X | 49 |
| TSX | 0074 | Transfer and Set Index | | | | 45 |
| TZE | 0100 | Transfer on Zero | | X | X | 43 |
| TNZ | —0100 | Transfer on No Zero | | X | X | 43 |
| CVR | 0114 | Convert by Replacement from AC | 6 | | | 62 |
| CAQ | —0114 | Convert by Addition from MQ | 6 | | | 63 |
| TPL | 0120 | Transfer on Plus | | X | X | 44 |
| TMI | —0120 | Transfer on Minus | | X | X | 44 |
| XCL | —0130 | Exchange Logical Accumulator and MQ | | | | 40 |
| XCA | 0131 | Exchange Accumulator and MQ | | | | 40 |
| TOV | 0140 | Transfer on Overflow | | X | X | 44 |
| TNO | —0140 | Transfer on No Overflow | | X | X | 44 |
| CRQ | —0154 | Convert by Replacement from MQ | 6 | | | 62 |
| TQO | 0161 | Transfer on Quotient Overflow | | X | X | 44 |
| TQP | 0162 | Transfer on MQ Plus | | X | X | 44 |
| MPY | 0200 | Multiply | 1 | X | X | 28 |
| MPR | —0200 | Multiply and Round | 1 | X | X | 28 |
| VLM | 0204 | Variable Length Multiply | 1,2 | X | | 28 |
| DVH | 0220 | Divide or Halt | | X | X | 30 |
| DVP | 0221 | Divide or Proceed | | X | X | 30 |
| VDH | 0224 | Variable Length Divide or Halt | 2 | X | | 30 |
| VDP | 0225 | Variable Length Divide or Proceed | 2 | X | | 30 |
| FDH | 0240 | Floating Divide or Halt | 5 | X | X | 36 |
| FDP | 0241 | Floating Divide or Proceed | 5 | X | X | 36 |
| FMP | 0260 | Floating Multiply | 1 | X | X | 35 |
| UFM | —0260 | Unnormalized Floating Multiply | 1 | X | X | 35 |
| FAD | 0300 | Floating Add | 3 | X | X | 32 |
| UFA | —0300 | Unnormalized Floating Add | 4 | X | X | 33 |
| FSB | 0302 | Floating Subtract | 3 | X | X | 33 |
| UFS | —0302 | Unnormalized Floating Subtract | 4 | X | X | 34 |
| FAM | 0304 | Floating Add Magnitude | 3 | X | X | 33 |
| UAM | —0304 | Unnormalized Add Magnitude | 4 | X | X | 34 |
| FSM | 0306 | Floating Subtract Magnitude | 3 | X | X | 34 |
| USM | —0306 | Unnormalized Subtract Magnitude | 4 | X | X | 34 |
| ANS | 0320 | AND to Storage | | X | X | 54 |
| ANA | —0320 | AND to Accumulator | | X | X | 54 |
| ERA | 0322 | Exclusive OR to Accumulator | | X | X | 54 |
| CAS | 0340 | Compare Accumulator with Storage | | X | X | 49 |
| LAS | —0340 | Logical Compare AC with Storage | | X | X | 49 |
| ACL | 0361 | Add and Carry Logical Word | | X | X | 27 |
| ADD | 0400 | Add | | X | X | 26 |
| SBM | —0400 | Subtract Magnitude | | X | X | 27 |
| ADM | 0401 | Add Magnitude | | X | X | 26 |
| SUB | 0402 | Subtract | | X | X | 27 |
| HPR | 0420 | Halt and Proceed | | | | 41 |
| IIS | 0440 | Invert Indicators from Storage | | X | X | 59 |
| LDI | 0441 | Load Indicators | | X | X | 57 |
| OSI | 0442 | OR Storage to Indicators | | X | X | 57 |
| OFT | 0444 | Off Test for Indicators | | X | X | 60 |
| RIS | 0445 | Reset Indicators from Storage | | X | X | 58 |
| ONT | 0446 | On Test for Indicators | | X | X | 60 |
| LDA | 0460 | Locate Drum Address | 8 | X | X | 65 |

## Numerical *(Continued)*

| OPERATION CODE | | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| ALPHA | OCTAL | | | | | |
| CLA | 0500 | Clear and Add | | X | X | 26 |
| CAL | —0500 | Clear and Add Logical Word | | X | X | 26 |
| ORA | —0501 | OR to Accumulator | | X | X | 54 |
| CLS | 0502 | Clear and Subtract | | X | X | 26 |
| ZET | 0520 | Storage Zero Test | | X | X | 49 |
| NZT | —0520 | Storage Non-Zero Test | | X | X | 49 |
| XEC | 0522 | Execute | | X | X | 42 |
| LXA | 0534 | Load Index from Address | | | | 50 |
| LXD | —0534 | Load Index from Decrement | | | | 51 |
| LAC | 0535 | Load Complement of Address in XR | | | | 51 |
| LDC | —0535 | Load Complement of Decrement in XR | | | | 51 |
| RCHA | 0540 | Reset and Load Channel A | | X | X | 70 |
| RCHB | —0540 | Reset and Load Channel B | | X | X | 70 |
| RCHC | 0541 | Reset and Load Channel C | | X | X | 70 |
| RCHD | —0541 | Reset and Load Channel D | | X | X | 70 |
| RCHE | 0542 | Reset and Load Channel E | | X | X | 70 |
| RCHF | —0542 | Reset and Load Channel F | | X | X | 70 |
| LCHA | 0544 | Load Channel A | 8 | X | X | 71 |
| LCHB | —0544 | Load Channel B | 8 | X | X | 71 |
| LCHC | 0545 | Load Channel C | 8 | X | X | 71 |
| LCHD | —0545 | Load Channel D | 8 | X | X | 71 |
| LCHE | 0546 | Load Channel E | 8 | X | X | 71 |
| LCHF | —0546 | Load Channel F | 8 | X | X | 71 |
| LDQ | 0560 | Load MQ | | X | X | 39 |
| ENB | 0564 | Enable | | X | X | 69 |
| STZ | 0600 | Store Zero | | X | X | 40 |
| STQ | —0600 | Store MQ | | X | X | 39 |
| STO | 0601 | Store | | X | X | 39 |
| SLW | 0602 | Store Logical Word | | X | X | 39 |
| ORS | —0602 | OR to Storage | | X | X | 54 |
| STI | 0604 | Store Indicators | | X | X | 57 |
| SLQ | —0620 | Store Left Half MQ | | X | X | 39 |
| STA | 0621 | Store Address | | X | X | 40 |
| STD | 0622 | Store Decrement | | X | X | 40 |
| STT | 0625 | Store Tag | | X | X | 40 |
| STL | —0625 | Store Instruction Location Counter | | X | X | 40 |
| STP | 0630 | Store Prefix | | X | X | 39 |
| SXA | 0634 | Store Index in Address | | | | 52 |
| SXD | —0634 | Store Index in Decrement | | | | 53 |
| SCHA | 0640 | Store Channel A | | X | X | 70 |
| SCHB | —0640 | Store Channel B | | X | X | 70 |
| SCHC | 0641 | Store Channel C | | X | X | 70 |
| SCHD | —0641 | Store Channel D | | X | X | 70 |
| SCHE | 0642 | Store Channel E | | X | X | 70 |
| SCHF | —0642 | Store Channel F | | X | X | 70 |
| CPY | 0700 | Copy | 8 | X | | 65 |
| CAD | —0700 | Copy and Add Logical | 8 | X | | 65 |
| PAX | 0734 | Place Address in Index | | | | 52 |
| PDX | —0734 | Place Decrement in Index | | | | 52 |
| PAC | 0737 | Place Complement of Address in XR | | | | 52 |

## Numerical *(Continued)*

| OPERATION CODE | | INSTRUCTION | MODIF'N | INDEXABLE | INDIRECTLY ADDRESSABLE | PAGE |
|---|---|---|---|---|---|---|
| ALPHA | OCTAL | | | | | |
| PDC | —0737 | Place Complement of Decrement in XR | | | | 52 |
| PXA | 0754 | Place Index in Address | | | | 53 |
| PXD | —0754 | Place Index in Decrement | | | | 53 |
| PSE | 0760 | Plus Sense | | X | | 46 |
| MSE | —0760 | Minus Sense | | X | | 47 |
| CLM | 0760..0000 | Clear Magnitude | | X | | 55 |
| LBT | 0760..0001 | Low-Order Bit Test | | X | | 48 |
| PBT | —0760..0001 | P-bit Test | | X | | 48 |
| CHS | 0760..0002 | Change Sign | | X | | 55 |
| EFTM | —0760..0002 | Enter Floating Trap Mode | | X | | 72 |
| SSP | 0760..0003 | Set Sign Plus | | X | | 56 |
| SSM | —0760..0003 | Set Sign Minus | | X | | 56 |
| ENK | 0760..0004 | Enter Keys | | X | | 41 |
| LFTM | —0760..0004 | Leave Floating Trap Mode | | X | | 72 |
| IOT | 0760..0005 | Input-Output Check Test | | X | | 48 |
| ESTM | —0760..0005 | Enter Select Trap Mode | | X | | 71 |
| COM | 0760..0006 | Complement Magnitude | | X | | 55 |
| ECTM | —0760..0006 | Enter Copy Trap Mode | | X | | 72 |
| ETM | 0760..0007 | Enter Trapping Mode | | X | | 42 |
| LTM | —0760..0007 | Leave Trapping Mode | | X | | 43 |
| RND | 0760..0010 | Round | | X | | 28 |
| LSNM | —0760..0010 | Leave Storage Nullification Mode | | X | | 71 |
| FRN | 0760..0011 | Floating Round | | X | | 34 |
| DCT | 0760..0012 | Divide Check Test | | X | | 48 |
| RCT | 0760..0014 | Restore Channel Traps | | X | | 70 |
| NOP | 0761 | No Operation | | | | 41 |
| RDS | 0762 | Read Select | 8 | X | | 64 |
| LLS | 0763 | Long Left Shift | 7 | X | | 38 |
| LGL | —0763 | Logical Left Shift | 7 | X | | 38 |
| BSR | 0764 | Backspace Record | 8 | X | | 64 |
| BSF | —0764 | Backspace File | 8 | X | | 64 |
| LRS | 0765 | Long Right Shift | 7 | X | | 38 |
| LGR | —0765 | Logical Right Shift | 7 | X | | 38 |
| WRS | 0766 | Write Select | 8 | X | | 64 |
| ALS | 0767 | Accumulator Left Shift | 7 | X | | 37 |
| WEF | 0770 | Write End of File | 8 | X | | 65 |
| ARS | 0771 | Accumulator Right Shift | 7 | X | | 38 |
| REW | 0772 | Rewind | 8 | X | | 65 |
| RQL | —0773 | Rotate MQ Left | 7 | X | | 38 |
| AXT | 0774 | Address to Index True | | | | 51 |
| AXC | —0774 | Address to Index Complemented | | | | 51 |
| TXI | 1000 | Transfer with XR Incremented | | | | 45 |
| STR | —1000 | Store Location and Trap | | | | 40 |
| TIX | 2000 | Transfer on Index | | | | 46 |
| TNX | —2000 | Transfer on No Index | | | | 46 |
| TXH | 3000 | Transfer on Index High | | | | 45 |
| TXL | —3000 | Transfer on XR Low or Equal | | | | 46 |

709 FORTRAN Automatic Coding System for the IBM 709

## TABLE OF CONTENTS

APPENDICES

# CHAPTER 1 — GENERAL PROPERTIES OF A FORTRAN SOURCE PROGRAM

A FORTRAN source program consists of a sequence of source statements, of which there are 38 different types. These statement types are described in detail in the chapters which follow.

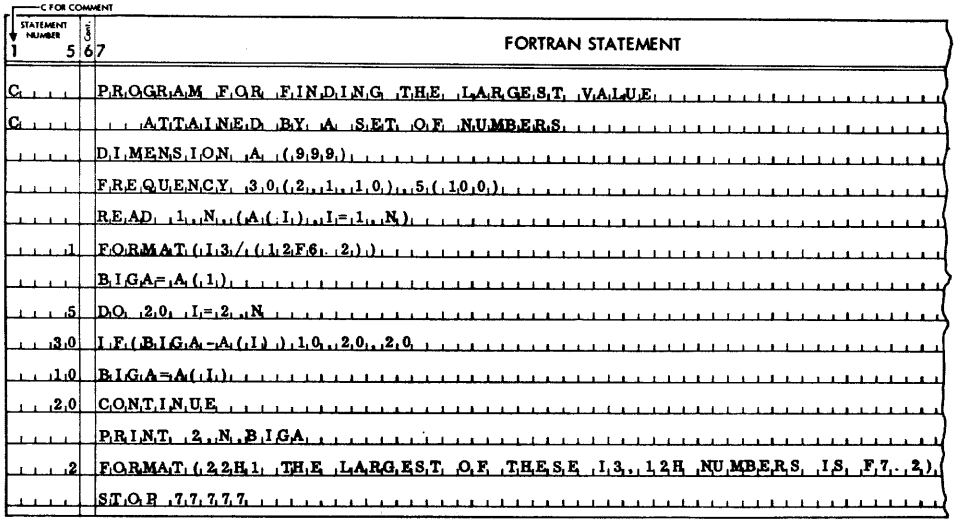
## Example of a FORTRAN Program

The brief program shown in Figure 1 will serve to illustrate the general appearance and some of the properties of a FORTRAN program. It is shown as coded on a standard FORTRAN coding sheet.

The purpose of the program is to determine the largest value attained by a set of numbers, A (I), and to print the number on the attached printer. The numbers exist on punched cards, 12 to a card, each number occupying a field of 6 columns. There are no more than 999 numbers; the actual number is punched on the leading card and is the only number on that card.

## Punching a Source Program

Each statement of a FORTRAN source program is punched into a separate card (the standard FORTRAN card form is shown in Figure 2); however, if a statement is too long to fit on one card, it can be continued on as many as nine "continuation cards." The order of the source statements is governed solely by the order of the statement cards.


C FOR COMMENT
STATEMENT NUMBER
Cont.
1 5 6 7
FORTRAN STATEMENT

```
C     PROGRAM FOR FINDING THE LARGEST VALUE
C        ATTAINED BY A SET OF NUMBERS
      DIMENSION A (999)
      FREQUENCY 30(2,1,10), 5(100)
      READ 1,N,(A(I),I=1,N)
    1 FORMAT(I3/(12F6.2))
      BIGA= A(1)
    5 DO 20 I=2,N
   30 IF(BIGA-A(I))10,20,20
   10 BIGA=A(I)
   20 CONTINUE
      PRINT 2,N,BIGA
    2 FORMAT(22H1 THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)
      STOP 77777
```

Figure 1

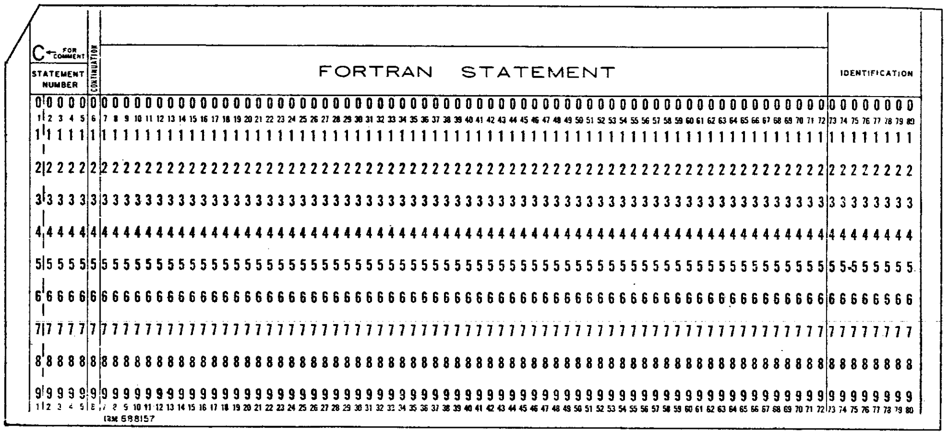


Figure 2

Cards which contain a "C" in column 1 are not processed by the FORTRAN program. Such cards may, therefore, be used to carry comments which will appear when the source program deck is listed.

Numbers less than 32,768 may be punched in columns 1-5 of the initial card of a statement. When such a number appears in these columns, it becomes the statement number of the statement. These statement numbers permit cross references within a source program, and when necessary, facilitate the correlation of source and object programs.

Column 6 of the initial card of a statement must be left blank or punched with a zero. Continuation cards (other than for comments), on the other hand, must have column 6 punched with some character other than zero, and may be punched with numbers from 1 through 9. Continuation cards for comments need not be punched in column 6; only the "C" in column 1 is necessary.

The statements themselves are punched in columns 7-72, both on initial and continuation cards. Thus, a statement may consist of not more than 660 characters (i.e., 10 cards). A table of the admissible characters for FORTRAN is given in Appendix B. Blank characters, except in column 6, are simply ignored by FORTRAN, and may be used freely to improve the readability of the source program listing.

Columns 73-80 are not processed by FORTRAN, and may, therefore, be punched with any desired identifying information.

The input to FORTRAN may be either the deck of source statement cards, or a BCD tape prepared on peripheral card-to-tape equipment using the standard SHARE 80 x 84 board. On such a tape, an end-of-file mark is required after the last card.

Types of FORTRAN Statements

The 38 types of statements which can be used in a FORTRAN program may be classified as follows:

1. The arithmetic formula which specifies a numerical computation. Part I, Chapter 2 discusses the symbols available for referring to constants, variables and functions; and Part II, Chapter 1 the combining of these into arithmetic formulas.

2. The fifteen control statements which govern the flow of control in the program. These, plus the END statement, are discussed in Part II, Chapter 2.

3. The four subprogram statements which enable the programmer to define and use subprograms. The method for utilizing subprograms is discussed in Part II, Chapter 3.

4. The thirteen input/output statements which provide the necessary input and output routines. These statements are discussed in Part II, Chapter 4.

5. The four specification statements which provide information required, or desirable to make the object program efficient. These are discussed in Part II, Chapter 5.

# CHAPTER 2 — CONSTANTS, VARIABLES, SUBSCRIPTS, AND EXPRESSIONS

As required of any programming language, FORTRAN provides a means of expressing numerical constants and variable quantities. In addition, a subscript notation is provided for expressing one-, two-, or three-dimensional arrays of variables.

## CONSTANTS

Two types of constants are permissible in the FORTRAN source program language: Fixed point (restricted to integers), and floating point (characterized by being written with a decimal point).

### Fixed Point Constants

| GENERAL FORM | EXAMPLES |
|---|---|
| 1 to 5 decimal digits. A preceding + or - sign is optional. The magnitude or absolute value of the constant must be less than $2^{17}$. | 3<br>+1<br>-28987 |

Where a fixed point constant is used for the value of a subscript, it is treated modulo (size of core storage).

### Floating Point Constants

| GENERAL FORM | EXAMPLES |
|---|---|
| Any number of decimal digits, with a decimal point at the beginning, at the end, or between two digits. A preceding + or - sign is optional. | 17.<br>5.0<br>-.0003<br>5.0E3 (5.0 x $10^3$) |
| A decimal exponent preceded by an E may follow a floating point constant. | 5.0E+3 (5.0 x $10^3$) |
| The magnitude of a number thus expressed must lie between the approximate limits of $10^{-38}$ and $10^{38}$, or be zero. | 5.0E-7 (5.0 x $10^{-7}$) |

## VARIABLES

Two types of variables are permissible: fixed point (restricted to integral values) and floating point. References to variables are made in the FORTRAN source language by symbolic names consisting of alphabetic and, if desired, numerical characters.

### Fixed Point Variables

| GENERAL FORM | EXAMPLES |
|---|---|
| 1 to 6 alphabetic or numerical characters (not special characters), of which the first is I, J, K, L, M, or N. | I<br>M2<br>JOBNO |

A fixed point variable can assume any integral value provided the magnitude is less than $2^{17}$. Values used for subscripts, however, are treated modulo (size of core storage).

To avoid the possibility that a variable may be considered by FORTRAN to be a function (see page 13), the following two warnings should be observed with respect to the naming of variables:

Warning: A variable cannot be given a name which coincides with the name of a function without its terminal F. Thus, if a function is named TIMEF, no variable should be named TIME.

Unless their names are less than four characters in length, subscripted variables (see below) must not be given names ending with F, because FORTRAN will consider variables so named to be functions.

### Floating Point Variables

| GENERAL FORM | EXAMPLES |
|---|---|
| 1 to 6 alphabetic or numerical characters (not special characters), of which the first is alphabetic but not I, J, K, L, M, or N. | A<br>B7<br>DELTA |

A floating point variable can assume any value expressible as a normalized floating point number, i. e., zero or with magnitude between approximately $10^{38}$ and $10^{-38}$.

Note: The restrictions on naming fixed point variables also apply to floating point variables.

### SUBSCRIPTS

A variable can be made to represent any element of a one-, two-, or three-dimensional array of quantities by appending 1, 2, or 3 subscripts to it, respectively. The variable is then a subscripted variable. These subscripts are fixed point quantities whose values determine the member of the array to which reference is made.

| GENERAL FORM | EXAMPLES |
|---|---|
| Let v represent any fixed point variable and c (or c') any unsigned fixed point constant. Then a subscript is an expression in one of the forms:<br>v<br>c<br>v+c or v-c<br>c*v<br>c*v+c' or c*v-c'<br>(The symbol * denotes multiplication.) | I<br>3<br>MU+2<br>MU-2<br>5*J<br>5*J+2<br>5*J-2 |

The variable in a subscript must not itself be subscripted.

## Subscripted Variables

| GENERAL FORM | EXAMPLES |
|---|---|
| A fixed or floating point variable, followed by parentheses enclosing 1, 2, or 3 subscripts which are separated by commas. | A (I)<br>K (3)<br>BETA (5*J-2, K+2, L) |

Each variable which appears in subscripted form must have the size of its array (i.e., the maximum values which its subscripts can attain) specified in a DIMENSION statement preceding the first appearance of the variable in the source program.

The value of a subscript exclusive of its addend, if any, must be greater than zero and not greater than the corresponding array dimension.

## Arrangement of Arrays in Storage

If an array, A, is 2-dimensional, it will be stored sequentially in the order $A_{1,1}; A_{2,1}; \ldots ; A_{m,1}; A_{1,2}; A_{2,2}; \ldots ; A_{m,2}; \ldots ; A_{m,n}$. Arrays are thus stored "columnwise," with the first of their subscripts varying most rapidly, and the last varying least rapidly. The same is true of 3-dimensional arrays. Arrays which are 1-dimensional are of course simply stored sequentially.

All arrays are stored backwards; i.e., in the order of decreasing absolute storage locations.

## EXPRESSIONS

A FORTRAN expression is any sequence of constants, variables (subscripted or not subscripted), and functions (see page 13 for naming of functions), separated by operation symbols, commas, and parentheses so as to form a meaningful mathematical expression.

### Rules for Constructing Expressions

By repeated use of the following rules, all permissible expressions may be derived.

1. Although a FORTRAN expression may be either fixed point or floating point, it must not be a mixed expression. This does not mean that a floating point quantity cannot appear in a fixed point expression, or vice versa, but rather that a quantity of one mode can appear in an expression of the other mode only in certain ways.

   These are:

   A. A floating point quantity can appear in a fixed point expression only as an argument of a function.

   B. A fixed point quantity can appear in a floating point expression only as an argument of a function, or as a subscript, or as an exponent.

2. Any fixed point (or floating point) constant, variable, or subscripted variable is also an expression of the same mode. Since variables with names beginning with I, J, K, L, M, or N, and intergers are fixed point, 3 and I are fixed point expressions. However, ALPHA and A (I, J, K) are floating point expressions.

3. If SOMEF is some function of n variables, and if E, F, . . . . , H are a set of n expressions of the correct modes for SOMEF, then SOMEF (E, F, . . . . . , H) is an expression of the same mode as SOMEF.

4. If E is an expression, and if its first character is not + or -, then +E and -E are expressions of the same mode as E. Thus -A is an expression, but + -A is not.

5. If E is an expression, then (E) is an expression of the same mode as E. Thus (A), ((A)), (((A))), etc., are expressions.

6. If E and F are expressions of the same mode, and if the first character of F is not + or -, then

   E + F<br>
   E - F<br>
   E * F<br>
   E / F

are all expressions of the same mode, but A-+B and A/+B are not expressions. (The characters +, -, *, and / denote addition, subtraction, multiplication, and division, respectively.)

7. If E and F are expressions, and F is not a floating point expression unless E is too, and the first character of F is not + or -, and neither E nor F is of the form A**B; then

   E**F

   is an expression of the same mode as E. Thus A**(B**C) is an expression, but I**(B**C) and A**B**C are not. (In FORTRAN, the symbol ** denotes exponentiation; hence A**B means $A^{B}$.)

### Hierarchy of Operations

When the hierarchy of operations in an expression is not explicity specified by the use of parantheses, it is understood by FORTRAN to be in the following order (from innermost operations to outermost):

Exponentiation
Multiplication and Division
Addition and Subtraction

For example, the expression

A+B/C+D**E*F-G

will be taken to mean

$A+(B/C)+(D^{E}*F)-G$

### Ordering within a Hierarchy

Parentheses which have been omitted from a sequence of consecutive multiplications and divisions (or consecutive additions and subtractions) will be understood to be grouped from the left. Thus, if • represents either * or / (or either + or -), then

A • B • C • D • E

will be taken by FORTRAN to mean

((((A•B)•C)•D)•E)

### Verification of Correct Use of Parentheses

The following procedure can be used for checking that the parentheses in a complicated expression correctly express the desired operations:

> Label the first open parenthesis "1"; thereafter, working from left to right, increase the label by 1 for each open parenthesis and decrease it by 1 for each closed parenthesis.

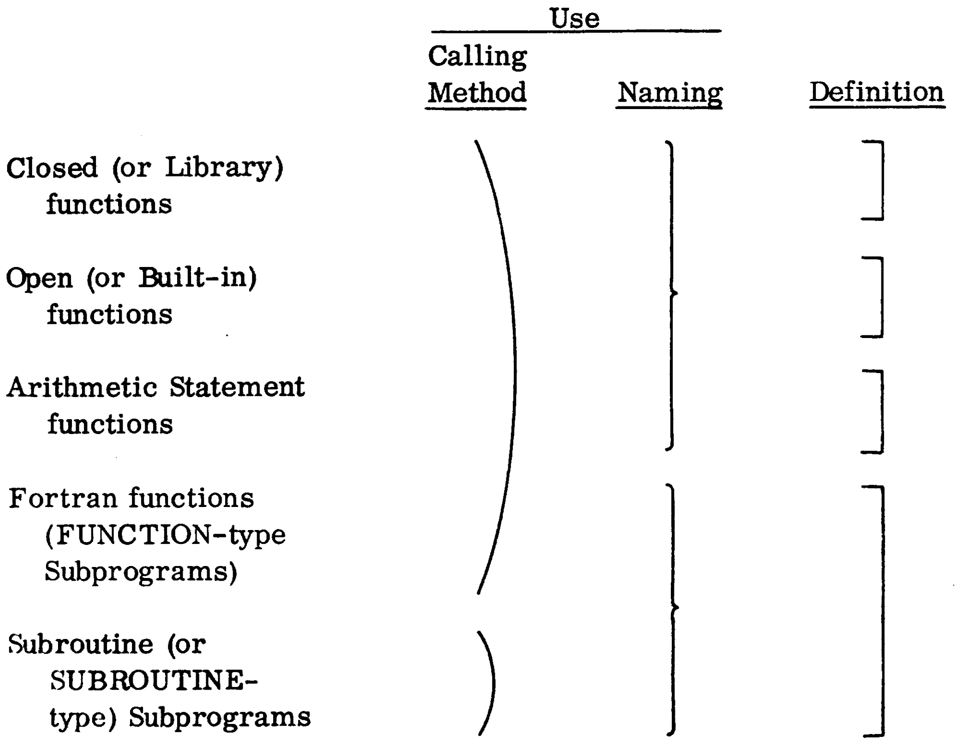
Then the label of the last parenthesis should be 0; the mate of an open parenthesis labeled n will be the next parenthesis labeled n-1.

### Optimization of Arithmetic Expressions

The efficiency of instructions compiled from arithmetic expressions may also be influenced by the way in which the expressions are written. The section on Optimization of Arithmetic Expressions in Part III, Chapter 2 mentions some of the considerations which affect object program efficiency.

# CHAPTER 3 — FUNCTIONS

This chapter is intended to discuss the four function-types which may be utilized in FORTRAN. However, in order to better clarify the meaning and uses of functions, they will first be shown in their relation to subroutine-types as a whole. A subroutine is considered as any sequence of instructions which performs some desired operation. In addition to functions, subroutines consist of subprograms. Their interrelationship can be represented schematically as follows:



Use
Calling Method
Naming
Definition
Closed (or Library) functions
Open (or Built-in) functions
Arithmetic Statement functions
Fortran functions (FUNCTION-type Subprograms)
Subroutine (or SUBROUTINE-type) Subprograms

It is thus clear that from the standpoint of use there are four subroutine-types (i.e., the functions) which are alike. They differ only in the way that one of them is named. When defining (writing) these subroutines for incorporation in a FORTRAN program, there are four different possibilities. (These are discussed on page 14.)

## CALLING

As indicated in the schematic there are two distinct ways by which subroutines may be referred to. One of these is by means of an arithmetic expression. (This applies to the four functions: Closed, Open, Arithmetic Statement, and Fortran functions.) The other, which applies to Subroutine subprograms, is by means of a CALL statement. (See page 33.)

Following are examples of arithmetic expressions including function names.

```
Y = A - SINF(B-C)

C = MIN0F (M,L)+ABC (B*FORTF(Z), E)
```

The names of Open, Closed, Arithmetic Statement, and Fortran functions are all used in this way. The appearance in the arithmetic expression serves to call the function; the value of the function is then computed, using the arguments which are supplied in the parentheses following the function name. Only one value is produced by these four functions, whereas the Subroutine subprogram may produce many values. (A value is here defined to be a single numerical quantity.)

## NAMING

The following paragraphs describe the rules for naming Open, Closed, and Arithmetic Statement functions.

### Naming of Open, Closed, and Arithmetic Statement Functions

| GENERAL FORM | EXAMPLES |
|---|---|
| The name of the function consists of 4 to 7 alphabetic or numerical characters (not special characters), of which the last must be F and the first must be alphabetic. Further, the first must be X if and only if the value of the function is to be fixed point. The name of the function is followed by parentheses enclosing the arguments separated by commas. | ABSF (B)<br>XMODF (M/N,K)<br>COSF (A)<br>FIRSTF (Z + B, Y) |

Mode of a Function and its Arguments. Consider a function of a single argument. It may be desired to state the argument either in fixed or floating point; similarly the function itself may be in either of these modes. Thus a function of a single argument has 4 possible mode configurations; in general a function of n arguments will have $2^{n+1}$ mode configurations.

A separate name must be given, and a separate routine must be available, for each of the mode configurations which is used. Thus, a complete set of names for a given function might be:

| | |
|---|---|
| SOMEF | Fixed argument, floating function |
| SOME0F | Floating argument, floating function |
| XSOMEF | Fixed argument, fixed function |
| XSOME0F | Floating argument, fixed function |

The X's and F's are mandatory, but the rest of the naming is arbitrary.

### Naming of Fortran Functions

Although these functions are referred to by arithmetic expressions in the same manner as the previous three types, the rules for naming them are different. These functions are named in exactly the same way as ordinary variables of the program, except that no name of a Fortran function which is 4 to 6 characters long may end in F. This means that the name of a fixed point Fortran function must have I, J, K, L, M or N for its first character.

Further details on naming Fortran functions are given on page 30.

## DEFINITION

Each of the four types of functions is defined (or generated) in a different way.

### Open (or Built-in) Functions

The 709 FORTRAN system, as distributed, contains 20 built-in subroutines. It, further, has the capacity for 7 more built-in subroutines. The additional subroutines may be inserted into the system by the particular installation. The detailed rules for doing this are given in the 709 FORTRAN Operations Manual.

Following are the 20 functions that are compiled as open subroutines into the arithmetic statement which calls them.

| Type of Function | Definition | No. of Args. | Name | Mode of Argument | Mode of Function |
|---|---|---|---|---|---|
| Absolute value | $\lvert Arg \rvert$ | 1 | ABSF | Floating | Floating |
| | | | XABSF | Fixed | Fixed |
| Truncation | Sign of Arg times largest integer $\leq \lvert Arg \rvert$ | 1 | INTF | Floating | Floating |
| | | | XINTF | Floating | Fixed |
| Remaindering (see note below) | $Arg_1$ (mod $Arg_2$) | 2 | MODF | Floating | Floating |
| | | | XMODF | Fixed | Fixed |
| Choosing largest value | Max ($Arg_1$, $Arg_2$, . . . ) | ≥ 2 | MAX0F | Fixed | Floating |
| | | | MAX1F | Floating | Floating |
| | | | XMAX0F | Fixed | Fixed |
| | | | XMAX1F | Floating | Fixed |
| Choosing smallest value | Min ($Arg_1$, $Arg_2$, . . . ) | ≥ 2 | MIN0F | Fixed | Floating |
| | | | MIN1F | Floating | Floating |
| | | | XMIN0F | Fixed | Fixed |
| | | | XMIN1F | Floating | Fixed |
| Float | Floating a fixed number | 1 | FLOATF | Fixed | Floating |
| Fix | Same as XINTF | 1 | XFIXF | Floating | Fixed |
| Transfer of sign | Sign of $Arg_2$ times ($Arg_1$) | 2 | SIGNF | Floating | Floating |
| | | | XSIGNF | Fixed | Fixed |
| Positive difference | $\lvert Arg_1 - Arg_2 \rvert$ | 2 | DIMF | Floating | Floating |
| | | | XDIMF | Fixed | Fixed |

NOTE: The function MODF ($Arg_1$, $Arg_2$)is defined as $Arg_1 - [Arg_1 / Arg_2] Arg_2$, where [x]=integral part of x.

### Closed (or Library) Functions

These are functions which are pre-written and may exist on the library tape or in prepared card decks. These functions constitute "closed" subroutines, i.e., instead of appearing in the object program for every reference that has been made to them in the source program, they appear only once regardless of the number of references.

Hand-coded closed functions may be added to the library. Rules for coding these subroutines are given in Appendix E; those for adding them to the library are included in the FORTRAN Operations Manual.

Seven library functions are included in the 709 FORTRAN system, as distributed. These are listed in Appendix C.

### Arithmetic Statement Functions

These are functions which are defined by a single FORTRAN arithmetic statement and apply only to the particular program or subprogram in which their definition appears.

| GENERAL FORM | EXAMPLES |
|---|---|
| "a = b" where a is a function name followed by parentheses enclosing its arguments (which must be distinct non-subscripted variables) separated by commas, and b is an expression which does not involve subscripted variables. Any functions appearing in b must be built-in, or available on the master tape, or already defined by preceding function statements. | FIRSTF(X) = A*X + B<br>SECONDF (X, B) = A*X + B<br>THIRDF(D) = FIRSTF(E)/D<br>FOURTHF (F, G) = SECONDF (F, THIRDF (G))<br>FIFTHF(I, A) = 3.0*A**I<br>SIXTHF(J) = J + K<br>XSIXTHF(J) = J + K |

Just as with the other functions, the answer will be expressed in fixed or floating point according as the name does or does not begin with X.

The right-hand side of a function statement may be any expression, not involving subscripted variables, that meets the requirements specified for expressions. In particular, it may involve functions freely, provided that any such function, if it is not built-in or available on the master tape, has been defined in a preceding function statement.

As many as desired of the variables appearing in the expression on the right-hand side may be stated on the left-hand side to be the arguments of the function. Since the arguments are really only dummy variables, their names are unimportant (except as indicating fixed or floating point mode) and may even be the same as names appearing elsewhere in the program.

Those variables on the right-hand side which are not stated as arguments are treated as parameters. Thus if FIRSTF is defined in a function statement as FIRSTF(X) = A*X + B then a later reference to FIRSTF (Y) will cause ay+b, based on the current values of a, b, and y, to be computed. The naming of parameters, therefore, must follow the normal rules of uniqueness.

A function defined by a function statement may be used just as any other function. In particular, its arguments may be expressions and may involve subscripted variables; thus a reference to FIRSTF(Z + Y(I)), with the above definition of FIRSTF, will cause $a(z+y_i) + b$ to be computed on the basis of the current values of a, b, $y_i$, and z.

Functions defined by arithmetic statements are always compiled as closed subroutines.

Note: All the arithmetic statements defining functions to be used in a program must precede the first executable statement of the program.

### Fortran Functions

This class of functions covers those subroutines which on the one hand cannot be defined by only one arithmetic statement, and on the other, are not utilized frequently enough to warrant a place on the library tape.

They are called Fortran functions because they may conveniently be defined by a conventional FORTRAN program. In this instance compiling a FORTRAN program produces a Function subroutine in exactly the form required for object program execution.

Since Fortran functions and Subroutine subprograms are defined in the same way, a discussion of the definition of Fortran functions is included in Part II, Chapter 3.

# CHAPTER 1 — THE ARITHMETIC FORMULA

## Arithmetic Formula

| GENERAL FORM | EXAMPLES |
|---|---|
| "a = b" where a is a variable (subscripted or not subscripted) and b is an expression. | Q1 = K<br>A(I)=B(I)+SINF(C(I)) |

The arithmetic formula defines a numerical calculation. A FORTRAN arithmetic formula resembles very closely a conventional arithmetic formula. However, in a FORTRAN arithmetic formula the = sign means "is to be replaced by," not "is equivalent to." Thus, the arithmetic formula

Y = N-LIMIT (J-2)

means that the value of N-LIMIT (J-2) is to replace the value of Y. The result is stored in fixed point or in floating point form if the variable to the left of the = sign is a fixed point or a floating point variable, respectively.

If the variable on the left is fixed point and the expression on the right is floating point, the result will first be computed in floating point and then truncated and converted to a fixed point integer. Thus, if the result is +3. 872 the fixed point number stored will be +3, not +4. If the variable on the left is floating point and the expression on the right fixed point, the latter will be computed in fixed point, and then converted to floating point.

Examples of Arithmetic Formulas

| | |
|---|---|
| A = B | Store the value of B in A. |
| I = B | Truncate B to an integer, convert to fixed point, and store in I. |
| A = I | Convert I to floating point, and store in A. |
| I = I+1 | Add 1 to I and store in I. This example illustrates the fact that an arithmetic formula is not an equation, but is a command to replace a value. |
| A = 3. 0*B | Replace A by 3B. |
| A = 3*B | Not permitted. The expression is mixed, i. e., contains both fixed point and floating point variables. |
| A = I*B | Not permitted. The expression is mixed. |

# CHAPTER 2 — CONTROL STATEMENTS AND END STATEMENT

The second class of FORTRAN statements is the set of sixteen control statements which enable the programmer to state the flow of his program.

## Unconditional GO TO

| GENERAL FORM | EXAMPLES |
|---|---|
| "GO TO n" where n is a statement number. | GO TO 3 |

This statement causes transfer of control to the statement with statement number n.

## Computed GO TO

| GENERAL FORM | EXAMPLES |
|---|---|
| "GO TO ($n_1$, $n_2$, . . . , $n_m$), i" where $n_1$, $n_2$, . . . . . , $n_m$ are statement numbers and i is a non-subscripted fixed point variable. | GO TO (30, 42, 50, 9), I |

Control is transferred to the statement with statement number $n_1$, $n_2$, $n_3$, . . . , $n_m$, depending on whether the value of i at time of execution is 1, 2, 3, . . . , m, respectively. Thus, in the example, if i is 3 at the time of execution, a transfer to the 3rd statement of the list, namely statement 50, will occur.

This statement is used to obtain a computed many-way fork.

## Assigned GO TO

| GENERAL FORM | EXAMPLES |
|---|---|
| "GO TO n, ($n_1$, $n_2$, . . . , $n_m$)" where n is a non-subscripted fixed point variable appearing in a previously executed ASSIGN statement, and $n_1$, $n_2$, . . . , $n_m$ are statement numbers. | GO TO K, (17, 12, 19) |

This statement causes transfer of control to the statement with statement number equal to that value of n which was last assigned by an ASSIGN statement; $n_1$, $n_2$, . . . , $n_m$ are a list of the values which n may have assigned.

The assigned GO TO is used to obtain a pre-set many-way fork.

When an assigned GO TO exists in the range of a DO, there is a restriction on the values of $n_1$, $n_2, \ldots,$ $n_m$. (See page 25.)

## ASSIGN

| GENERAL FORM | EXAMPLES |
|---|---|
| "ASSIGN i TO n" where i is a statement number and n is a non-subscripted fixed point variable which appears in an assigned GO TO statement. | ASSIGN 12 TO K |

This statement causes a subsequent GO TO n, ($n_1, \ldots,$ $n_i, \ldots,$ $n_m$) to transfer control to the statement with the statement number i.

## IF

| GENERAL FORM | EXAMPLES |
|---|---|
| "IF (a) $n_1$, $n_2$, $n_3$" where a is an expression and $n_1$, $n_2$, $n_3$ are statement numbers. | IF(A(J,K)-B)10, 4, 30 |

Control is transferred to the statement with the statement number $n_1$, $n_2$, or $n_3$ if the value of a is less than, equal to, or greater than zero, respectively.

## SENSE LIGHT

| GENERAL FORM | EXAMPLES |
|---|---|
| "SENSE LIGHT i" where i is 0, 1, 2, 3, or 4. | SENSE LIGHT 3 |

If i is 0, all Sense Lights will be turned Off; otherwise Sense Light i only will be turned On.

## IF (SENSE LIGHT)

| GENERAL FORM | EXAMPLES |
|---|---|
| "IF (SENSE LIGHT i) $n_1$, $n_2$" where $n_1$, and $n_2$ are statement numbers and i is 1, 2, 3, or 4. | IF (SENSE LIGHT 3) 30,40 |

Control is transferred to the statement with statement number $n_1$ or $n_2$ if Sense Light i is On or Off, respectively. If the light is On, it will be turned Off.

### IF (SENSE SWITCH)

| GENERAL FORM | EXAMPLES |
|---|---|
| "IF (SENSE SWITCH i) $n_1$, $n_2$" where $n_1$ and $n_2$ are statement numbers and i is 1, 2, 3, 4, 5, or 6. | IF (SENSE SWITCH 3)<br>30, 108 |

Control is transferred to the statement with statement number $n_1$ or $n_2$ if Sense Switch i is Down or Up, respectively.

### IF ACCUMULATOR OVERFLOW

| GENERAL FORM | EXAMPLES |
|---|---|
| "IF ACCUMULATOR OVERFLOW $n_1$, $n_2$" where $n_1$ and $n_2$ are statement numbers. | IF ACCUMULATOR OVERFLOW 30, 49 |

### IF QUOTIENT OVERFLOW

| GENERAL FORM | EXAMPLES |
|---|---|
| "IF QUOTIENT OVERFLOW $n_1$, $n_2$" where $n_1$ and $n_2$ are statement numbers. | IF QUOTIENT OVERFLOW 30, 49 |

Control is transferred to the statement with statement number $n_1$ if an overflow condition is present in either the Accumulator or the Multiplier-Quotient Register, and to $n_2$ is no overflow is present at all. That is, in 709 FORTRAN, programming either of these statements is equivalent to programming a non-FORTRAN statement, IF OVERFLOW $n_1$, $n_2$. In 709 FORTRAN, an internal indicator is used to denote the overflow condition; it is reset to the no-overflow condition after execution of either of these two statements.

When either the Accumulator or Multiplier-Quotient Register overflows, the register is set to contain the highest possible quantity, i.e., $377777777777_8$. The sign is unchanged.

If an underflow occurs in either register, that register is set to zero, the sign remains unchanged. There is no test for the underflow condition.

## IF DIVIDE CHECK

| GENERAL FORM | EXAMPLES |
|---|---|
| "IF DIVIDE CHECK $n_1$, $n_2$" where $n_1$ and $n_2$ are statement numbers. | IF DIVIDE CHECK<br>84, 40 |

Control is transferred to the statement with statement number $n_1$ or $n_2$, if the Divide Check trigger is On or Off, respectively. If it is On, it will be turned Off.

## DO

| GENERAL FORM | EXAMPLES |
|---|---|
| "DO n i = $m_1$, $m_2$" or "DO n i = $m_1$, $m_2$, $m_3$" where n is a statement number, i is a non-subscripted fixed point variable, and $m_1$, $m_2$, $m_3$ are each either an unsigned fixed point constant or non-subscripted fixed point variable. If $m_3$ is not stated, it is taken to be 1. | DO 30 I = 1, 10<br>DO 30 I = 1, M,3 |

The DO statement is a command to execute repeatedly the statements which follow, up to and including the statement with statement number n. The first time the statements are executed with i = $m_1$. For each succeeding execution i is increased by $m_3$. After they have been executed with i equal to the highest of this sequence of values which does not exceed $m_2$, control passes to the statement following the last statement in the range of the DO.

The range of a DO is that set of statements which will be executed repeatedly; i.e., it is the sequence of consecutive statements immediately following the DO, up to and including the statement numbered n.

The index of a DO is the fixed point variable i, which is controlled by the DO in such a way that its value begins at $m_1$ and is increased each time by $m_3$ until it is about to exceed $m_2$. Throughout the range it is available for computation, either as an ordinary fixed point variable or as the variable of a subscript. After the last execution of the range, the DO is said to be satisfied.

Suppose, for example, that control has reached statement 10 of the program

```
10  DO    11 I = 1, 10
11  A(I) = I * N(I)
12
```

The range of the DO is statement 11, and the index is I. The DO sets I to 1 and control passes into the range. The value of 1· N(1) is computed, converted to floating point, and stored in location A (1). Since statement 11 is the last statement in the range of the DO and the DO is unsatisfied, I is increased to 2 and control returns to the beginning of the range, statement 11. The value of 2· N(2) is then computed and stored in location A(2). The process continues until statement 11 has been executed with I = 10. Since the DO is satisfied, control then passes to statement 12.

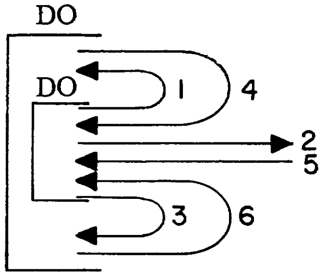
<u>DOs within DOs</u>. Among the statements in the range of a DO may be other DO statements. When this is so, the following rule must be observed:

Rule 1: If the range of a DO includes another DO, then all of the statements in the range of the latter must also be in the range of the former.

A set of DOs satisfying this rule is called a <u>nest of DOs</u>.

<u>Transfer of Control and DOs</u>. Transfers of control from and into the range of a DO are subject to the following rule:

Rule 2: No transfer is permitted into the range of any DO from outside its range. Thus, in the configuration below, 1, 2 and 3 are permitted transfers, but 4, 5 and 6 are not.



Exception. There is one situation in which control can be transferred into the range of a DO from outside its range. Suppose control is in the range of the innermost DO of a nest of DOs which are completely nested (i.e., every pair of DOs in the nest is such that one contains the other). Suppose also that control is transferred to a section of the program, completely outside the nest to which these DOs belong, which makes no change in any of the indexes or indexing parameters (m's) in the nest. Then after the execution of this latter section of the program, control can be transferred back to the range of the same innermost DO from which it originally came. This provision makes it possible to exit temporarily from the range of some DOs to execute a subroutine.

Restriction on Assigned GO TOs in the Range of a DO. When an assigned GO TO is in the range of a DO, the statements to which it may transfer must all be in the exclusive range of a single DO (i.e., among those statements in the range of a DO which are not in the range of any DO in its range), or all outside the DO nest.

Preservation of Index Values. When control leaves the range of a DO in the ordinary way (i.e., when the DO becomes satisfied and control passes on to the next statement after the range) the exit is said to be a normal exit. After a normal exit from a DO occurs, the value of the index controlled by that DO is not defined, and the index cannot be used again until it is redefined. (In this connection, see "Further Details about DO Statements," page 63.)

However, if exit occurs by a transfer out of the range, the current value of the index remains available for any subsequent use. If exit occurs by a transfer which is in the ranges of several DOs, the current values of all the indexes controlled by those DOs are preserved for any subsequent use.

Restrictions on Statements in the Range of a DO. Only one type of statement is not permitted in the range of a DO, namely any statement which redefines the value of the index or of any of the indexing parameters (m's). In other words, the indexing of a DO loop must be completely set before the range is entered.

The first statement in the range of a DO must not be one of the non-executable FORTRAN statements. The range of a DO cannot end with a transfer.

Exits. When a CALL statement is executed in the range of a DO, care must be taken that the called subprogram does not alter the DO index or indexing parameters. This applies as well when a Fortran Function is called for in the range of a DO.

## CONTINUE

| GENERAL FORM | EXAMPLES |
|---|---|
| "CONTINUE" | CONTINUE |

CONTINUE is a dummy statement which gives rise to no instructions in the object program. It is most frequently used as the last statement in the range of a DO to provide a transfer address for IF and GO TO statements which are intended to begin another repetition of the DO range.

As an example of a program which requires a CONTINUE, consider the table search:

```
     .
     .
     .
10   DO  12  I = 1,  100
     IF (ARG - VALUE (I))12, 20, 12
12   CONTINUE
     .
     .
     .
```

This program will scan the 100-entry VALUE table until it finds an entry which equals the value of the variable ARG, whereupon it exits to statement 20 with the value of I available for fixed point use; if no entry in the table equals the value of ARG, a normal exit to the statement following the CONTINUE will occur.

## PAUSE

| GENERAL FORM | EXAMPLES |
|---|---|
| "PAUSE" or "PAUSE n" where n is an unsigned octal fixed point constant. | PAUSE<br>PAUSE 77777 |

The machine will halt with the octal number n in the address field of the Storage Register. If n is not specified, it is understood to be 0. Depressing the Start key causes the program to resume execution of the object program with the next FORTRAN statement.

STOP

| GENERAL FORM | EXAMPLES |
|---|---|
| "STOP" or "STOP n" where n is an unsigned octal fixed point constant. | STOP<br>STOP 77777 |

This statement causes a halt in such a way that depressing the Start key has no effect. Therefore, in contrast to PAUSE, this statement is used where a terminal, rather than a temporary stop, is desired. The octal number n is positioned in the address field of the Storage Register. If n is not specified, it is understood to be 0.

END

| GENERAL FORM | EXAMPLES |
|---|---|
| END ($I_1$, $I_2$, $I_3$, $I_4$, $I_5$) where I is 0, 1, or 2. | END (2, 2, 2, 2, 2)<br>END (1, 2, 0, 1, 1) |

This statement differs from the previous statements discussed in this chapter in that it does not affect the flow of control in the object program being compiled. Its application is to the FORTRAN executive program during compilation. It serves two purposes:

1. FORTRAN provides the option of running under monitor control, which allows the compilation of a number of separate FORTRAN source programs in succession. The END statement, then, marks the end of any given FORTRAN source program, separating it from the program that follows.

2. The END statement specifies the treatment of the setting of Sense Switches 1 through 5. (The sense switch options are given in Appendix D.)

For each I of the statement's list,

| | |
|---|---|
| I = 0 | Ignore actual sense switch setting. Assume it to be Up. |
| I = 1 | Ignore actual sense switch setting. Assume it to be Down. |
| I = 2 | Note actual setting and act accordingly. (See Appendix D.) |

The END statement does not,of course, physically change the setting of a sense switch.

# CHAPTER 3 — SUBPROGRAM STATEMENTS

It is possible to program, in the FORTRAN language, subroutines which are referred to by other programs. These subroutines may, in turn, refer to still other lower level subroutines which may also be coded in FORTRAN language. It is therefore possible, by means of FORTRAN, to code problems using several levels of subroutines. This configuration may be thought of as a total problem consisting of one main program and any number of subprograms.

Because of the interrelationship among several different programs, it is possible to include a block of hand-coded instructions in a sequence including instructions compiled from FORTRAN source programs. It is only necessary that hand-coded instructions conform to rules for subprogram formation, since they will comprise a distinct subprogram.

This chapter presents a discussion of the two types of FORTRAN coded subprograms possible. These are the FUNCTION subprogram and the SUBROUTINE subprogram. Four statements, described subsequently, are necessary for their definition and use. Two of these, SUBROUTINE and FUNCTION, are dealt with in Section A; the other two, CALL and RETURN, are discussed in Section B.

Illustrations of, and the rules for hand-coding subprograms are given on page 79.

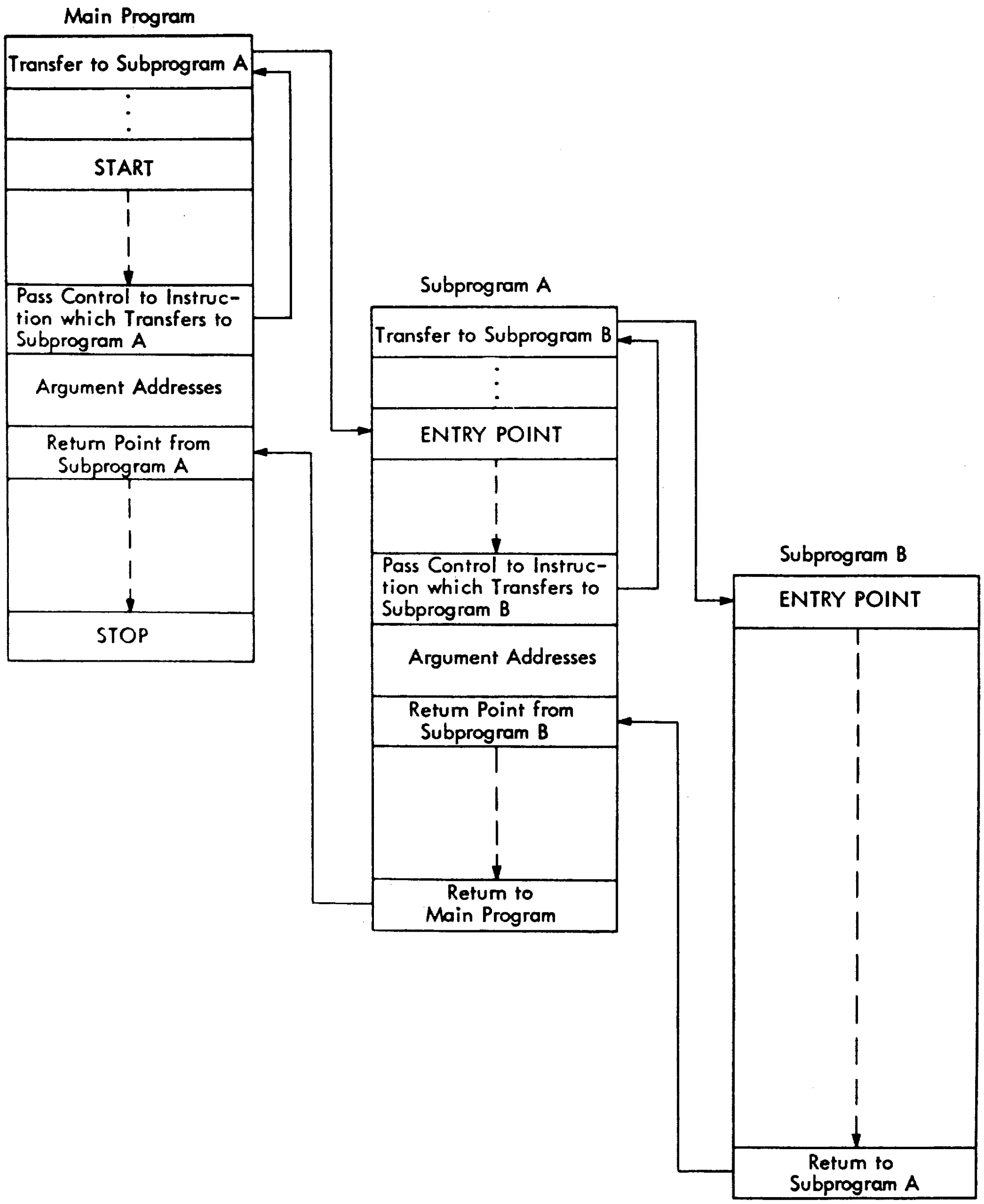
Although FUNCTION subprograms and SUBROUTINE subprograms are treated together and may be viewed as similar, it must be remembered that they differ in two fundamental respects.

1. The FUNCTION subprogram, which results in a Fortran function as defined on page 16, is always single-valued, whereas the SUBROUTINE subprogram may be multi-valued.

2. The FUNCTION subprogram is called or referred to by the arithmetic expression containing its name; the SUBROUTINE subprogram can only be referred to by a CALL statement.

Each of these two types of subprogram, when coded in FORTRAN language must be regarded as independent FORTRAN programs. In all respects, they conform to rules for FORTRAN programming. However, they may be compiled with the main program of which they are parts by means of multiple program compilation. In this way the results of a multiple program compilation will be a complete main program–subprogram sequence ready to be executed.

Schematically, the relationship among nested main and subprograms can be shown as follows. This diagram, further, indicates the main division of the internal structure of each program.



## SECTION A: FUNCTION and SUBROUTINE

### FUNCTION

| GENERAL FORM | EXAMPLES |
|---|---|
| "FUNCTION Name ($a_1$, $a_2$, . . . , $a_n$)" where Name is the symbolic name of a single-valued function, and the arguments $a_1$, $a_2$, . . . $a_n$, of which there must be at least one, are non-subscripted variable names.<br>The function name consists of 1 to 6 alphanumerical characters, the first of which must be alphabetic; the first character must be I, J, K, L, M, or N if and only if the value of the function is to be fixed point, and the final character must not be F if there are more than three characters in the name. The function name must not occur in a DIMENSION statement in the FUNCTION subprogram, or in a DIMENSION statement in any program which uses the function.<br>The arguments may be any variable names occurring in executable statements of the subprogram. | FUNCTION ARCSIN (RADIAN)<br>FUNCTION ROOT (B, A, C)<br>FUNCTION INTRST (RATE, YEARS) |

The FUNCTION statement must be the first statement of a Fortran function subprogram and defines it to be such.

In a FUNCTION subprogram, the name of the function must appear at least once as the variable on the left-hand side of an arithmetic statement, or alternately in an input statement list, e.g.,

```
FUNCTION NAME (A, B)
.
.
.
NAME = Z + B
.
.
.
RETURN
```

By this means, the output value of the function is returned to the calling program.

This type of program may either be compiled independently, or multiple-compiled with others. A FUNCTION subprogram must never be inserted between two statements of any other single program.

The arguments following the name in the FUNCTION statement, may be considered as "dummy" variable names. That is, during object program execution, other actual arguments are substituted for them. Therefore, the arguments which follow the function reference in the calling program must agree with those in the FUNCTION statement in the subprogram in number, order, and mode. Furthermore, when a dummy argument is an array name, the corresponding actual argument must also be an array name. Each of these array names must appear in DIMENSION statements of their respective programs with the same dimensions.

None of the dummy variables may appear in EQUIVALENCE or COMMON statements in the FUNCTION subprogram.

SUBROUTINE

| GENERAL FORM | EXAMPLES |
|---|---|
| "SUBROUTINE Name ($a_1$, $a_2$,..., $a_n$)" where Name is the symbolic name of a subprogram, and the arguments $a_1$, $a_2$,..., $a_n$, if any, are non-subscripted variable names.<br>The name of the subprogram may consist of 1 to 6 alphanumerical characters, the first of which is alphabetic; its final character must not be F if there are more than three characters in the name. Also, the name of the subprogram must not be listed in a DIMENSION statement of any program which calls the subprogram, or in a DIMENSION statement of the subprogram itself.<br>The arguments may be any variable names occurring in executable statements in the subprogram. | SUBROUTINE MATMPY (A,N,M,B,L,C)<br>SUBROUTINE QDRTIC (B,A,C, ROOT1, ROOT2) |

This statement is used as the first statement of a Subroutine function and defines it to be such. A subprogram introduced by the SUBROUTINE statement must be a FORTRAN program and may contain any FORTRAN statements except FUNCTION or another SUBROUTINE statement.

A Subroutine subprogram must be referred to be a CALL statement (see page 33) in the calling program. The CALL statement specifies the name of the subprogram and its arguments.

Unlike the FUNCTION-type subprogram which returns only a single numerical value, the Subroutine subprogram uses one or more of its arguments to return output. The arguments so used, must, therefore, appear on the left side of an arithmetic statement someplace in the program (or alternately, in an input statement list within the program).

The arguments of the SUBROUTINE statement are dummy variables which are replaced, at execution, by the actual arguments which are supplied by the CALL statement. There must, therefore, be correspondence in number, order, and mode, between the two sets of arguments. Furthermore, when a dummy argument is an array name, the corresponding actual argument must also be an array name. Each of these array names must appear in DIMENSION statements of their respective programs with the same dimensions.

For example, the subprogram headed by

```
        SUBROUTINE MATMPY (A, N, M, B, L, C)
```

could be called by the main program through the statement

```
        CALL MATMPY (X,5,10, Y, 7, Z)
```

where the dummy variables, A, B, C, are the names of matrices. A, B, and C must appear in a DIMENSION statement in subprogram MATMPY and X, Y, and Z must appear in a DIMENSION statement in the calling program. The dimensions assigned must be the same in both statements.

None of the dummy variables may appear in EQUIVALENCE or COMMON statements in the Subroutine subprogram. These subprograms may be compiled independently or multiple-compiled with others.

## SECTION B: CALL AND RETURN

The CALL statement has reference only to the Subroutine subprogram, whereas the RETURN statement is used by both the Function and Subroutine subprograms.

### CALL

| GENERAL FORM | EXAMPLES |
|---|---|
| "CALL Name ($a_1$, $a_2$,.., $a_n$)" where Name is the name of a Subroutine subprogram, and $a_1$, $a_2$,.., $a_n$ are arguments which take one of the forms described below. | CALL MATMPY<br>(X, 5, 10, Y, 7, Z)<br>CALL QDRTIC<br>(P*9. 732, Q/4. 536,<br>R - S**2. 0, X1, X2) |

This statement is used to call Subroutine subprograms; the CALL transfers control to the subprogram and presents it with the parenthesized arguments. Each argument may be one of the following.

1. Fixed point constant.

2. Floating point constant.

3. Fixed point variable, with or without subscripts.

4. Floating point variable, with or without subscripts.

5. Arithmetic expression.

6. Alphanumerical characters. Such arguments must be preceded be nH where n is the count of characters included in the argument, e. g. , 9HEND POINT. Note that blank spaces and special characters are considered characters when used in alphanumerical fields.

   Alphanumerical arguments can, of course, only be used as input to hand-coded programs. (See Appendix E.)

The arguments presented by the CALL statement must agree in number, order, mode and array size with the corresponding arguments in the SUBROUTINE statement of the called subprogram.

RETURN

| GENERAL FORM | EXAMPLES |
|---|---|
| "RETURN" | RETURN |

This statement terminates any subprogram, whether of the type headed by a SUBROUTINE or a FUNCTION statement, and returns control to the calling program. A RETURN statement must, therefore, be the last executed statement of the subprogram. It need not be physically the last statement of the subprogram; it can be any point reached by a path of control and any number of RETURN statements may be used.

# CHAPTER 4 — INPUT/OUTPUT STATEMENTS

There are thirteen FORTRAN statements available for specifying the transmission of information during execution of the object program, between storage on the one hand, and magnetic tapes, drums, card reader, card punch, and printer on the other hand. These input/output statements can be grouped as follows:

1. Five statements (READ, READ INPUT TAPE, PUNCH, PRINT, and WRITE OUTPUT TAPE) which cause transmission of a specified list of quantities between storage and an external input/output medium: cards, printed sheet, or magnetic tape, for which information is expressed in Hollerith punching, alphanumerical print, or binary-coded-decimal (BCD) tape code, respectively.

2. One statement (FORMAT), which is a non-executable statement, that specifies the arrangement of the information in the external input/output medium with respect to the five source statements of group 1 above.

3. Four statements (READ TAPE, READ DRUM, WRITE TAPE, and WRITE DRUM) which cause information to be transmitted in binary machine-language.

4. Three statements (END FILE, BACKSPACE, and REWIND) that manipulate magnetic tapes.

## Specifying Lists of Quantities

Of the thirteen input/output statements, nine call for the transmission of information and must, therefore, include a list of the quantities to be transmitted. This list is ordered, and its order must be the same as the order in which the words of information exist (for input), or will exist (for output) in the input/output medium.

The formation and meaning of a list is best described by an example.

```
A, B(3), (C(I), D(I,K), I = 1, 10), ((E(I,J),

I = 1, 10, 2), F(J,3), J = 1,K)
```

Suppose that this list is used with an output statement. Then the information will be written on the input/output medium in this order:

```
A, B(3), C(1), D(1, K), C(2), D(2, K),....., C(10), D(10, K),

E(1, 1), E(3, 1),....., E(9, 1), F(1, 3),

E(1, 2), E(3, 2),....., E(9, 2), F(2, 3),......, F(K, 3).
```

Similarly, if this list were used with an input statement, the successive words, as they were read from the external medium, would be placed into the sequence of storage locations just given.

Thus, the list reads from left to right with repetition for variables enclosed within parentheses. Only variables, and not constants, may be listed. The execution is exactly that of a DO-loop, as though each opening parenthesis (except subscripting parentheses) were a DO, with indexing given immediately before the matching closing parenthesis, and with the DO range extending up to that indexing information. The order of the above list can thus be considered the equivalent of the "program:"

```
1. A
2. B(3)
3. DO 5 I = 1, 10
4. C(I)
5. D(I,K)
6. DO 9 J = 1, K
7. DO 8 I = 1, 10, 2
8. E(I,J)
9. F(J,3)
```

Note that indexing information, as in DOs, consists of three constants or fixed point variables, and that the last of these may be omitted, in which case it is taken to be 1.

For a list of the form K, (A(K)) or K, (A(I), I = 1, K) where an index or indexing parameter itself appears earlier in the list of an input statement, the indexing will be carried out with the newly read-in value.

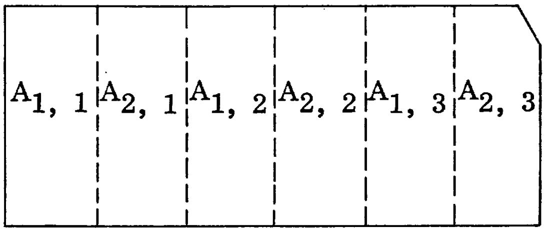
### Input/Output in Matrix Form

As outlined on page 8, FORTRAN treats variables according to conventional matrix practice. Thus, the input/output statement

```
READ 1, ((A(I,J), I = 1, 2), J = 1, 3)
```

causes the reading of I x J (in this case 2 x 3) items of information. The data items will be read into storage in the same order as they are found on the input medium.

For example, if punched on a data card in the form:



| | | | | | |
|---|---|---|---|---|---|
| $A_{1, 1}$ | $A_{2, 1}$ | $A_{1, 2}$ | $A_{2, 2}$ | $A_{1, 3}$ | $A_{2, 3}$ |

the items will be stored in locations N, N-1, N-2, . . . , N-5, respectively, where N is the highest absolute location used for the array of information to be read in.

### Input/Output of Entire Matrices

When input/output of an entire matrix is desired, an abbreviated notation may be used for the list of the input/output statement; only the name of the array need be given and the indexing information may be omitted.

Thus, if A has previously been listed in a DIMENSION statement, the statement,

```
READ 1, A
```

is sufficient to read in all of the elements of the array A. In 709 FORTRAN, the elements, read in by this notation, are stored in their natural order, i. e., in order of decreasing storage locations. If A has not previously appeared in a DIMENSION statement, only the first element will be read in.

Note: Certain restrictions to these rules exist with respect to lists for the statements READ DRUM and WRITE DRUM, for which the abbreviated notation mentioned immediately above is the only one permitted.

### FORMAT

| GENERAL FORM | EXAMPLES |
|---|---|
| "FORMAT (Specification)" where Specification is as described below. | FORMAT<br>(I2/(E12. 4, F10. 4) ) |

The five input/output statements of group 1 (listed on page 35) contain, in addition to the list of quantities to be transmitted, the statement number of a FORMAT statement describing the information format to be used. It also specifies the type of conversion to be performed between the internal machine-language and external notation. FORMAT statements are not executed, their function is merely to supply information to the object program. Therefore, they may be placed anywhere in the source program, except as the first statement in the range of a DO.

For the sake of clarity, the details of writing a FORMAT specification are given below for use with PRINT statements. However, the description is valid for any case simply by generalizing the concept of "printed line" to that of unit record in the input/output medium. A unit record may be:

1. A printed line with a maximum of 120 characters.
2. A punched card with a maximum of 72 characters.
3. A BCD tape record with a maximum of 120 characters.

Three basic types of decimal-to-binary or binary-to-decimal conversion are available:

| INTERNAL | →<br>Type<br>← | EXTERNAL |
|---|---|---|
| Floating point variable | E | Floating point, decimal |
| Floating point variable | F | Fixed point, decimal |
| Fixed point variable | I | Decimal integer |

The FORMAT specification describes the line to be printed by giving, for each field in the line (from left to right, beginning with the first type wheel):

1. The type of conversion (E, F, or I) to be used.

2. The width (w) of the field.

3. For E- and F-type conversion, the number of places (d) after the decimal point that are to be printed (d is treated modulo 10).

### Basic Field Specifications

These basic field specifications are given in the forms

Iw, Ew.d, and Fw.d

with the specification for successive fields separated by commas. Thus the statement FORMAT (I2, E12.4, F10.4) might give the line:

```
27 -0.9321E 02   -0.0076
```

As in this example, the field widths may be made greater than necessary so as to provide spacing blanks between numbers. In this case, there is 1 blank following the 27, 1 blank after the E (automatically supplied except in cases of a negative exponent, when a minus sign will appear), and 3 blanks after the 02. Within each field the printed output will always appear in the right-most positions.

It may be desired to print n successive fields within one record, in the same fashion. This may be specified by giving n before E, F, or I. Thus, the statement FORMAT (I2, 3E12.4) might give:

```
27 -0.9321E 02  -0.7580E-02 0.5536E 00
```

### Repetition of Groups

A limited parenthetical expression is permitted in order to enable repetition of data fields according to certain format specifications within a longer FORMAT statement specification. Thus, FORMAT (2(F10.6, E10.2), I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4).

To permit more general use of F-type conversion, a scale factor followed by the letter P may precede the specification. The scale factor is defined such that:

$$\text{Printer number} = \text{Internal number} \times 10^{\text{scale factor}}$$

Thus, the statement FORMAT (I2, 1P3F11.3) used with the data of the preceding example, would give

```
27    -932.096      -0.076        5.536
```

whereas FORMAT (I2, - 1P3F11.3) would give

```
27      -9.321      -0.001        0.055
```

A positive scale factor may also be used with E-type conversion to increase the number and decrease the exponent. Thus, FORMAT (I2, 1P3E12.4) would produce with the same data

```
27  -9.3210E 01 -7.5804E-03  5.5361E-01
```

The scale factor is assumed to be zero if no other value has been given. However, once a value has been given, it will hold for all E- and F-type conversions following the scale factor within the same FORMAT statement. This applies to both single-record and multiple-record formats (see page 40). Once a scale factor has been given, a subsequent scale factor of zero in the same FORMAT statement must be specified by 0P. Scale factors have no effect on I-conversion.

### Hollerith Fields

A field may be designed as Hollerith, in which case alphanumerical information will be printed in it. The field width, followed by the desired characters, should appear in the appropriate place in the specification. For example, the statement FORMAT (3HXY= F8.3, 4H Z = F6.2, 7H W/AF= F7.3) would give

```
XY = -93.210  Z =  -0.01  W/AF =    0.554
```

Note that any Hollerith characters, including blanks may be printed. This is the only instance in which FORTRAN does not ignore blanks. It is possible to print Hollerith information only, by giving no list with the input/output statement and specifying no I, E, or F fields in the FORMAT statement.

**Originating Hollerith Text**

Consider a Hollerith field in a FORMAT statement at the time of execution of the object program. If the FORMAT statement is used with an input statement, the Hollerith text listed in the FORMAT statement will be replaced by whatever text is read in from the corresponding field in the input/output medium. When that same FORMAT statement is used for output, whatever information is then in the FORMAT statement will appear in the output data. Thus, text can be originated in the source program, or as input to the object program.

**Multiple-Record Formats**

To deal with a block of more than one line of print, a FORMAT specification may have several different one-line formats, separated by a slash (/) to indicate the beginning of a new line. Thus, FORMAT (3F9.2, 2F10.4/8E14.5) would specify a multi-line block of print in which lines 1, 3, 5,.... have format (3F9.2, 2F10.4), and lines 2, 4, 6,.... have format 8E14.5.

If a multiple-line format is desired such that the first two lines will be printed according to a special format and all remaining lines according to another format, the last line-specification should be enclosed in a second pair of parentheses; e.g., FORMAT (I2, 3E12.4/2F10.3, 3F9.4/ (10F12.4)). If data items remain to be transmitted after the format specification has been completely "used," the format repeats from the last open parenthesis.

As these examples show, both the slash and the closing parenthesis of the FORMAT statement indicate a termination of a record.

Blank lines may be introduced into a multi-line FORMAT statement, by listing consecutive slashes. N + 1 consecutive slashes produce N blank lines.

**FORMAT and Input/Output Statement Lists**

The FORMAT statement indicates, among other things, the maximum size of each record to be transmitted. In this connection, it must be remembered that the FORMAT statement is used in conjunction with the list of some particular input/output statement, except when a FORMAT statement consists entirely of alphanumerical fields. In all other cases, control in the object program switches back and forth between the list (which specifies whether data remains to be transmitted) and the FORMAT statement (which gives the specifications for transmission of that data).

**Ending a FORMAT Statement**

During input/output of data, the object program scans the FORMAT statement to which the relevant input/output statement refers. When a specification for a numerical field is found and list items remain to be transmitted, input/output takes place according to the specification and scanning of the FORMAT statement resumes. If no items remain, transmission ceases and execution of that particular input/output statement is terminated. Thus, a decimal input/output operation will be brought to an end when a specification for a numerical field or the end of the FORMAT statement is encountered, and there are no items remaining in the list.

### Carriage Control

The WRITE OUTPUT TAPE statement prepares a decimal tape which can later be used to obtain off-line printed output. The off-line printer is manually set to operate in one of three modes: single space, double space, and Program Control. Under Program Control, which gives the greatest flexibility, the first character of each BCD record controls spacing of the off-line printer and that character is not printed. The control characters and their effects are:

| | |
|---|---|
| Blank | Single space before printing |
| 0 | Double space before printing |
| + | No space before printing |
| 1 - 9 | Skip to printer control channels 1-9* |
| J - R | Short skip to printer control channels 1-9* |

Thus, a FORMAT specification for WRITE OUTPUT TAPE for off-line printing with Program Control, will usually begin with 1H followed by the appropriate control character. This is required for the PRINT statement since on-line printing simulates off-line printing under Program Control.

### Data Input to the Object Program

Decimal input data to be read by means of a READ or READ INPUT TAPE when the object program is executed, must be in essentially the same format as given in the previous examples. Thus, a card to be read according to FORMAT (I2, E12.4, F10.4) might be punched

```
27 -0.9321E 02    -0.0076
```

Within each field, all information must appear at the extreme right. Plus signs may be omitted or indicated by a blank or +. Minus signs may be punched with an 11-punch or an 8-4 punch. Blanks in numerical fields are regarded as zeros. Numbers for E- and F-type conversion may contain any number of digits, but only the high-order 8 digits will be retained (no rounding will be performed). Numbers for I-type conversion will be treated modulo $2^{17}$.

* See the section on Carriage Control in the Reference Manual for the IBM 709 Data Processing System (Form A22-6501).

To permit economy in punching, certain relaxations in input data format are permitted.

1. Numbers of E-type conversion need not have 4 columns devoted to the exponent field. The start of the exponent field must be marked by an E, or if that is omitted, by a + or - (not a blank). Thus E2, E02, +2, +02, E 02, and E+02 are all permissible exponent fields.

2. Numbers for E- or F-type conversion need not have their decimal point punched. If it is not punched, the FORMAT specification will supply it; for example, the number -09321+2 with the specification E12.4 will be treated as though the decimal point had been punched between the 0 and the 9. If the decimal point is punched in the card, its position over-rides the indicated position in the FORMAT specification.

## READ

| GENERAL FORM | EXAMPLES |
|---|---|
| "READ n, List" where n is the statement number of a FORMAT statement, and List is as described on page 35. | READ 1, ((ARRAY (I,J), I = 1, 3), J = 1, 5) |

The READ statement causes the reading of cards from the card reader. For 709 FORTRAN, the Data Synchronizer Channel to which the card reader is attached, must be specified by the installation (see "Symbolic Input/Output Unit Designation" page 43). Record after record (i.e., card after card) is read until the complete list has been "satisfied," i.e., brought in, converted, and stored in the locations specified by the list of the READ statement. The FORMAT statement to which the READ refers, describes the arrangement of information on the cards and the type of conversion to be made.

## READ INPUT TAPE

| GENERAL FORM | EXAMPLES |
|---|---|
| "READ INPUT TAPE i, n, List" where i is an unsigned fixed point constant or a fixed point variable; n is the statement number of a FORMAT statement; and List is as described on page 35. | READ INPUT TAPE 24, 30, K, A(J)<br>READ INPUT TAPE N, 30, K, A(J) |

The READ INPUT TAPE statement causes the object program to read BCD information from symbolic tape unit i (in 709 FORTRAN, $0<i<49$). Record after record is brought in, in accordance with the FORMAT statement, until the complete list has been satisfied.

The object program tests for the proper functioning of the tape reading process. In the event that the tape cannot be read properly, the object program halts.

## Symbolic Input/Output Unit Designation

Tape units. In order to enable 709 FORTRAN to accept source programs written in connection with other programming systems, a distinction is made between the logical tape unit numbers specified in the source program, and the actual tape units which will be affected by the resulting object program. Logical/actual equivalences for the 709 FORTRAN system are specified in the system as distributed, but these may be changed by the installation in accordance with its own needs. The equivalences are established by the insertion of a control card into the edit deck of the 709 FORTRAN system. (See "The 709 FORTRAN Editing Program, " in the 709 FORTRAN Operations Manual. )

Card reader, on-line printer, and card punch. One each of these input/output units can be attached to Data Synchronizer Channels A, C, or E of the 709. The card reader, on-line printer, or card punch which will actually be involved in the execution of READ, PRINT, or PUNCH, respectively, is specified by the system as distributed and may be changed by the installation.

At the time the 709 FORTRAN object program is executed, the equivalence between the logical and actual input/output units must be known.

## PUNCH

| GENERAL FORM | EXAMPLES |
|---|---|
| "PUNCH n, List" where n is the statement number of a FORMAT statement, and List is as described on page 35. | PUNCH 30, (A(J), J = 1, 10) |

The PUNCH statement causes the object program to punch Hollerith cards. Cards are punched in accordance with the FORMAT statement until the complete list has been satisfied.

## PRINT

| GENERAL FORM | EXAMPLES |
|---|---|
| "PRINT n, List" where n is the statement number of a FORMAT statement and List is as described on page 35. | PRINT 2, (A(J), J = 1, 10) |

The PRINT statement causes the object program to print output data on an on-line printer. Successive lines are printed in accordance with the FORMAT statement, until the complete list has been satisfied.

## WRITE OUTPUT TAPE

| GENERAL FORM | EXAMPLES |
|---|---|
| "WRITE OUTPUT TAPE i, n, List" where i is an unsigned fixed point constant or a fixed point variable, n is the statement number of a FORMAT statement, and List is as described on page 35. | WRITE OUTPUT TAPE 42, 30, (A(J), J = 1, 10)<br>WRITE OUTPUT TAPE L, 30, (A(J), J = 1, 10) |

The WRITE OUTPUT TAPE statement causes the object program to write BCD information on symbolic tape unit i (in 709 FORTRAN, $0<i<49$).

Successive records are written in accordance with the FORMAT statement until the complete list has been satisfied. An end-of-file is not written after the last record.

## READ TAPE

| GENERAL FORM | EXAMPLES |
|---|---|
| "READ TAPE i, List" where i is an unsigned fixed point constant or a fixed point variable, and List is as described on page 35. | READ TAPE 24, (A(J), J = 1, 10)<br>READ TAPE K, (A(J), J = 1, 10) |

The READ TAPE statement causes the object program to read binary information from symbolic tape unit i (in 709 FORTRAN, $0<i<49$), into locations specified in the list. A record is read completely only if the list specifies as many words as the tape record contains; no more than one record will be read. The tape, however, always moves to the beginning of the next record.

Binary tapes read by a 709 FORTRAN compiled program should have been written by a 709 FORTRAN object program. It is, however, possible to use a non-FORTRAN written binary tape provided the tape records are in the proper format. The following is a description of this record format.

Consider a logical record as being any sequence of binary words to be read by any one input statement. This logical record must be broken into physical records, each of which is a maximum of $128_{10}$ words long. Of course, if a logical record consists of fewer than $128_{10}$ words, it will comprise only 1 physical record. The first word of each physical record is a "signal" word that is not part of the list. This word contains zero for all but the last physical record of a logical record. The first word of the last physical record contains a number designating the number of physical records in this logical record.

The object program checks tape reading. In the event that a record cannot be read properly, the object program halts.

## READ DRUM

| GENERAL FORM | EXAMPLES |
|---|---|
| "READ DRUM i, j, List" where i and j are each either an unsigned fixed point constant or a fixed point variable, with the value of i between 1 and 8 inclusive; and List is as described below. | READ DRUM 2, 1000, A, B, C, D (3)<br>READ DRUM K, J, A, B, C, D (3) |

The READ DRUM statement causes the object program to read words of binary information from consecutive locations on drum i, beginning with the word in drum location j, where $0<j<2048$. (If $j>2047$, it is interpreted modulo 2048.) Reading continues until all words specified by the list have been read in. If the list specifies an array, the array is stored in inverse order.

The list for the READ DRUM and WRITE DRUM statements can consist only of variables without subscripts or with only constant subscripts, such as A, B(5), C, D. Variables consisting of only one element of data will be read into storage in the ordinary way; those which are arrays will be read with indexing obtained from their DIMENSION statements. Thus, the statement READ DRUM i, j, A, where A is an array, causes the complete array to be read. The array, A, is stored in inverse order.

## WRITE TAPE

| GENERAL FORM | EXAMPLES |
|---|---|
| "WRITE TAPE i, List" where i is an unsigned fixed point constant or a fixed point variable, and List is as described on page 35. | WRITE TAPE 24, (A(J), J = 1, 10)<br>WRITE TAPE K, (A(J), J = 1, 10) |

The WRITE TAPE statement causes the object program to write binary information on the tape unit with symbolic tape number i (in 709 FORTRAN, $0<i<49$). One record is written consisting of all the words specified in the list.

The object program checks tape writing. In the event that a record cannot be written properly, the object program halts.

## WRITE DRUM

| GENERAL FORM | EXAMPLES |
|---|---|
| "WRITE DRUM i, j, List" where i and j are each either an unsigned fixed point constant or a fixed point variable, with the value of i between 1 and 8, inclusive, and List is as described for READ DRUM. | WRITE DRUM 2, 1000, A, B, C, D(6)<br>WRITE DRUM K, J, A, B, C, D(6) |

The WRITE DRUM statement causes the object program to write words of binary information onto consecutive locations on drum i, beginning with drum location j. (If $j>2047$, it is interpreted modulo 2048.) Writing continues until all the words specified by the list have been written.

The list of the WRITE DRUM statement is subject to the same restrictions that apply to READ DRUM.

## END FILE

| GENERAL FORM | EXAMPLES |
|---|---|
| "END FILE i" where i is an unsigned fixed point constant, or a fixed point variable. | END FILE 29<br>END FILE K |

The END FILE statement causes the object program to write an end-of-file mark on symbolic tape unit i (0<i<49).

## REWIND

| GENERAL FORM | EXAMPLES |
|---|---|
| "REWIND i" where i is an unsigned fixed point constant, or a fixed point variable. | REWIND 3<br>REWIND K. |

The REWIND statement causes the object program to rewind symbolic tape unit i (0<i<49).

## BACKSPACE

| GENERAL FORM | EXAMPLES |
|---|---|
| "BACKSPACE i" where i is an unsigned fixed point constant, or a fixed point variable. | BACKSPACE 18<br>BACKSPACE K |

The BACKSPACE statement causes the object program to backspace symbolic tape unit i (0<i<49) by one record.

# CHAPTER 5 — SPECIFICATION STATEMENTS

The final type of FORTRAN statement consists of the four specification statements: DIMENSION, FREQUENCY, EQUIVALENCE, and COMMON. These are non-executable statements which supply necessary information, or information to increase object program efficiency.

## DIMENSION

| GENERAL FORM | EXAMPLES |
|---|---|
| "DIMENSION v, v, v, . . . " where each v is the name of a variable, subscripted with 1, 2, or 3 unsigned fixed point constants. Any number of v's may be given. | DIMENSION A(10), B(5, 15), CVAL(3,4,5,) |

The DIMENSION statement provides the information necessary to allocate storage in the object program for arrays.

Each variable which appears in subscripted form in a program or subprogram must appear in a DIMENSION statement of that program or subprogram; the DIMENSION statement must precede the first appearance of that variable. The DIMENSION statement lists the maximum dimensions of arrays; in the object program references to these arrays must never exceed the specified dimensions.

The above example indicates that B is a 2-dimensional array for which the subscripts never exceed 5 and 15. The DIMENSION statement therefore, causes 75 (i. e., 5 x 15) storage locations to be set aside for the array B.

A single DIMENSION statement may specify the dimensions of any number of arrays. A program must not contain a DIMENSION statement which includes the name of the program itself, or any program which it calls.

## FREQUENCY

| GENERAL FORM | EXAMPLES |
|---|---|
| "FREQUENCY n (i, j,...), m(k, l,...),..." where n, m,... are statement numbers and i, j, k, l, ... are unsigned fixed point constants. | FREQUENCY 30(1,2,1), 40 (11), 50(1, 7, 1, 1) 10 (1, 7, 1, 1) |

The FREQUENCY statement has no direct effect upon the execution of the object program. Its sole purpose is to inform FORTRAN about the number of times which the programmer believes that each branch of one or more specified control branchings will be executed.

The purpose of the statement is to make the object program as efficient as possible in terms of execution time and storage locations required. In no case will the logical flow of an object program be altered by a FREQUENCY statement.

A FREQUENCY statement can be placed anywhere in the FORTRAN source program except as the first statement in the range of a DO, any may be used to give frequency estimates for any number of branch-points. For each branch-point, the information consists of the statement number of the statement causing the branch, followed by parentheses enclosing the estimated frequencies separated by commas.

In a program including the above example, statement 30 might be an IF, and statement 50, a computed GO TO. In these cases, the probability of going to each of the 3 or 4 branch points, respectively, is given by the corresponding entry of the FREQUENCY statement. Statement 40 must be a DO, in which at least one of the parameters is variable and the value of which is not known in advance. An estimate is made that the DO range will be executed 11 times before the DO is satisfied.

All frequency estimates, except those about DOs are relative. Thus, the example given above could have been FREQUENCY 30(2,4,2), 40(11), 50(3,21,3,3), with equivalent results. A frequency can be estimated as 0; this will be taken to mean that the expected frequency is very small.

## Statements to Which Applicable

The following table lists the seven FORTRAN statements about which frequency information may be given.

| STATEMENT | No. of Branches | REMARKS |
|---|---|---|
| (Computed) GO TO | ≥2 | Frequencies must appear in the same order as the branches. If no frequencies are given they are assumed to be equal for all branches. |
| IF | 3 | |
| IF (SENSE SWITCH) | 2 | |
| IF ACCUMULATOR OVERFLOW | 2 | |
| IF QUOTIENT OVERFLOW | 2 | |
| IF DIVIDE CHECK | 2 | |
| DO | 1 | Frequency need be given only when $m_1$, $m_2$, or $m_3$ is variable. |

A frequency estimate concerning a DO is ignored unless at least one of the indexing parameters of that DO is variable. Moreover, such frequency estimates should be based only on the expected values of those variable parameters; in other words, even if the range of a DO were to contain transfer exits (see page 24), the frequency estimate should specify the number of times the range must be executed to cause a normal exit. A DO with variable indexing parameters and for which no FREQUENCY statement is given will be treated by FORTRAN as though a frequency of 5 has been estimated.

## EQUIVALENCE

| GENERAL FORM | EXAMPLES |
|---|---|
| "EQUIVALENCE (a,b,c,...), (d,e,f,...),..." where a, b, c, d, e, f,... are variables optionally followed by a single unsigned fixed point constant in parentheses. | EQUIVALENCE (A,B(1), C(5)), (D(17), E(3)) |

The EQUIVALENCE statement provides the option of controlling the allocation of data storage in the object program. In particular, when the logic of the program permits it, the number of storage locations used can be reduced by causing locations to be shared by two or more variables.

An EQUIVALENCE statement may be placed anywhere in the source program, except as the first statement of the range of a DO. Each pair of parentheses of the statement list encloses the names of two or more quantities which are to be stored in the same locations during execution of the object program; any number of equivalences (i.e., sets of parentheses) may be given.

In an EQUIVALENCE statement, the meaning of C(5) would be "the 4th storage location following the one which contains C, or (if C is an array) contains $C_1$, $C_{1,1}$, or $C_{1,1,1}$." In general C(p) is defined for p>0 to mean the (p-1) th location after C or after the beginning of the C-array; i.e., the p th location in the array. If p is not specified, it is taken to be 1.

Thus, the above sample statement indicates that the A, B, and C arrays are to be assigned storage locations such that the elements A, B, and C(5) are to occupy the same location. In addition, it specifies that D(17) and E(3) are to share the same location.

Quantities or arrays which are not mentioned in an EQUIVALENCE statement will be assigned unique locations.

Locations can be shared only among variables, not among constants.

The sharing of storage locations cannot be planned safely without a knowledge of which FORTRAN statements, when executed in the object program, will cause a new value to be stored in a location. There are seven such statements:

A. Execution of an arithmetic formula stores a new value of the variable for the left-hand side of the formula.

B. Execution of an ASSIGN i TO n stores a new value in n.

C. Execution of a DO will in general store a new indexing value. (It will not always do so, however; see the section, "Further Details about DO statements," page 63 .)

D. Execution of a READ, READ INPUT TAPE, READ TAPE, or READ DRUM each stores new values for the variables mentioned in the statement list.

## COMMON

| GENERAL FORM | EXAMPLES |
|---|---|
| "COMMON A, B,..." where A, B,... are the names of variables and non-subscripted array names. | COMMON X, ANGLE, MATA, MATB |

Variables, including array names, appearing in COMMON statements are assigned to upper storage. They are stored in locations completely separate from the block of program instructions, constants, and data (see page 57). This area is assigned separately for each program compiled. For 709 FORTRAN, the area is assigned beginning at location $77461_8$ and continuing downwards. This separate (COMMON) area may be shared by a program and its subprograms. In this way, COMMON enables data storage area to be shared between programs in a way analogous to that by which EQUIVALENCE permits data storage sharing within a single program. Where the logic of the programs permits, this can result in a large saving of storage space.

Array names appearing in COMMON must also appear in a DIMENSION statement in the same program.

The programmer has complete control over the locations assigned to the variables appearing in COMMON. The locations are assigned in the sequence in which the variables appear in the COMMON statements, beginning with the first COMMON statement of the problem.

### Arguments in Common Storage

Because of the above, COMMON statements may be used to serve another important function. They may be used as a medium by which to transmit arguments from the calling program to the called Fortran function or Subroutine subprogram. In this way, they are transmitted implicitly rather than explicitly by being listed in the parentheses following the subroutine name.

To obtain implicit arguments, it is necessary only to have the corresponding variables in the two programs occupy the same location. This can be obtained by having them occupy corresponding positions in COMMON statements of the two programs.

Notes:

1. In order to force correspondence in storage locations between two variables which otherwise will occupy different relative positions in COMMON storage, it is valid to place dummy variable names in

a COMMON statement. These dummy names, which may be dimensioned, will cause reservation of the space necessary to cause correspondence.

2. While implicit arguments can take the place of all arguments in CALL-type subroutines, there must be at least one explicit argument in a Fortran function. Here, too, a dummy variable may be used for convenience.

The entire COMMON area may be relocated downward for any one problem by means of a Control Card. ( See FORTRAN Operations Manual. )

When a variable is made equivalent to a variable which appears in a COMMON statement, the first variable will also be located in COMMON storage. When COMMON variables also appear in EQUIVALENCE statements, the ordinary sequence of COMMON variables is changed and priority is given to those variables in EQUIVALENCE statements, in the order in which they appear in EQUIVALENCE statements. For example,

```
COMMON A, B, C, D

EQUIVALENCE (C, G), (E, B)
```

will cause storage to be assigned in the following way.

| | |
|---|---|
| $77461_8$ | C and G |
| $77460_8$ | B and E |
| $77457_8$ | A |
| $77456_8$ | D |

# CHAPTER 1 — MISCELLANEOUS DETAILS ABOUT FORTRAN

## SOURCE AND OBJECT MACHINES

The source machine is that which is used to translate a FORTRAN source program into the object program. The object machine is that on which the object program is executed.

For 709 FORTRAN, the source machine must be an IBM 709 Data Processing System which includes at least 8, 192 storage locations, 5 tape units, 1 on-line card punch, 1 on-line card reader, and 1 on-line printer. When multiple-program compiling, 3 additional tape units are required.

The object machine may be of any size. The information produced at compiling time by FORTRAN includes a count of the storage locations required by the object program. From this information it can be determined whether an object program, together with its subprograms, is too large for a given object machine.

## ARRANGEMENT OF THE OBJECT PROGRAM

A main object program and its associated subprograms, may each be considered as a separate, but complete block, containing everything, except COMMON data, necessary for execution of the program. These blocks are placed contiguously by the FORTRAN BSS loader in lower storage with a variable length area separating them from COMMON in upper storage.

Each program block consists of program instructions, constants, erasable storage, and data, which are stored in that order in ascending storage locations. The data is separated into non-dimensioned variables, dimensioned variables, and variables appearing in EQUIVALENCE statements. The first program loaded will start at location $11_8$.

COMMON data starts at $77461_8$, and continues downward in storage. The area above $77461_8$ is available for erasable storage for library and hand-coded subroutines.

When a source program is compiled, FORTRAN produces a printed "storage map" of the arrangement of storage locations in the object program.

## FIXED POINT ARITHMETIC

The use of fixed point arithmetic is governed by the following considerations:

1. Fixed point constants specified in the source program must have magnitudes $< 2^{17}$.

2. Fixed point data read in by the object program itself is treated modulo $2^{17}$.

3. The output from fixed point arithmetic in the object program is modulo $2^{17}$. However, if, during computation of a fixed point arithmetic expression, an intermediate value occurs which is $\geq 2^{19}$, it is possible that the final result will be inaccurate. (The inaccuracy will occur only when the arithmetic expression contains a divide.)

4. Indexing in the object program is modulo (size of core storage) — never greater than $2^{15}$.

## OPTIMIZATION OF ARITHMETIC EXPRESSIONS

Considerable attention is given by FORTRAN to the efficiency of the object program instructions arising from an arithmetic expression, regardless of how the expression is written. Thus, although the expression

A• B• C• D• E

is taken to mean

((((A• B)• C)• D)• E)

(where · represents / or *, or + or -)

FORTRAN assumes that mathematically equivalent expressions are computationally equivalent. Hence, a sequence of consecutive multiplications and/or divisions (or additions and/or subtractions) not grouped by parentheses will be reordered, if necessary, to minimize the number of storage accesses in the object program.

Although the assumption concerning mathematical and computational equivalence is virtually true for floating point expressions, special care must be taken to indicate the order of fixed point multiplication and division, since fixed point arithmetic in FORTRAN is "greatest integer" arithmetic (i.e., truncated or remainderless). Thus, the expression

5*4/2

which is by convention taken to mean ((5 x 4)/2), is computed in a FORTRAN object program as

((5/2)*4)

i.e., it is computed from left to right after permutation of the operands to minimize storage accesses. The result of a FORTRAN computation in this case, would be 8. On the other hand, the result of the expression (5 x 4)/2 is 10. Therefore, to insure accuracy of fixed point multiplication and division, it is suggested that parentheses be inserted into the expression involved.

One important type of optimization, involving common sub-expressions, takes place only if the expression is suitably written. For example, the arithmetic statement

```
Y = A*B*C + SINF(A*B)
```

will cause the object program to compute the product A*B twice. An efficient object program would compute the product A*B only once. The statement is correctly written

```
Y = (A*B) * C + SINF (A*B)
```

By parenthesizing the common subexpression, A*B will be computed only once in the object program.

In general, when common sub-expressions occur within an expression, they should be parenthesized.

There is one case in which it is not necessary to write the parentheses, because FORTRAN will assume them to be present. These are the type discussed in "Hierarchy of Operations," page 10), and need not be given. Thus

```
Y = A*B+C+SINF(A*B)
```

is, for optimization purposes, as suitable as

```
Y   (A*B)+C+SINF(A*B)
```

However, the parentheses discussed in "Ordering within a Hierarchy," on page 10, must be supplied if optimization of common sub-expressions is to occur.

## SUBROUTINES ON THE SYSTEMS TAPE

Various library subroutines in relocatable binary form are available on the FORTRAN master tape. As mentioned on page 15, further subroutines can be placed on the tape by each installation in accordance with its own requirements. To do so, the following steps are necessary:

1. Produce the subroutine in the form of relocatable binary cards.

2. Produce a program card in accordance with specifications outlined in the FORTRAN Operation's manual.

3. Transcribe the resulting card deck onto the master tape by means of the 9LIB program included in the FORTRAN Editing Program.

Tape subroutines may include Fortran functions and Subroutine subprograms. The program card compiled by FORTRAN with these programs will be in the format required for tape subroutines.

If the name of a function defined by a library tape subroutine is encountered while FORTRAN is processing a source program, that subroutine will be included in the object program. Only one such inclusion will be made for a particular function, regardless of how many times that function occurs in the source program.

## INPUT AND OUTPUT OF ARGUMENTS

When control is transferred to a library subroutine, other than a Fortran function or Subroutine subprogram, the argument(s) will be located as follows: $\text{Arg}_1$ will be located in the Ac, $\text{Arg}_2$ (if any) in in the MQ, $\text{Arg}_3$ (if any) in relocatable location $77775_8$, $\text{Arg}_4$ in relocatable location $77774_8$, etc. Locations down through $77462_8$ are available for common erasable storage for library subroutines.

The output of any function called by an arithmetic statement which is a single value, must be in the Accumulator when control is returned to the calling program. All Index Registers which were stored at the beginning of the subroutine, must be restored prior to returning control.

The arguments for Fortran functions and Subroutine subprograms are listed in the object program after the transfer to the subroutine (see Appendix D).

## RELATIVE CONSTANTS

A relative constant is defined as a variable in a subscript, which is not under control of a DO, or a DO-implying parentheses in a list. For example, in the sequence:

```
A = B(K)

DO  10 I = 1,  10

X = B(I) + C(I,3J+2)
```

K and J are relative constants, but I is not.

The appearance of a relative constant in any of the following ways will be called a relative constant definition.

1. On the left side of an arithmetic statement.
2. In the list of an input statement.
3. As an argument for a Fortran function of Subroutine subprogram.
4. In a COMMON statement.

The following paragraphs describe methods for assuring that the computation for relative constants occur at the proper point between the definition and the use of the relative constant.

### Relative Constants in an Input List

In the object program, some computation will take place at each such defintion. In the case of READ, READ TAPE, and READ INPUT TAPE lists, the computation may not precede the use of a relative constant in the list unless the relative constant appearance is handled properly.

Where the relative constant definition appears in the same READ, READ TAPE, or READ INPUT TAPE list with its relative constant and precedes it, extra parentheses may be required in the list. In such a list, it is necessary that there be a left parenthesis, other than the left parenthesis of a subscript combination, between the relative constant definition and its relative constant. If the list does not contain the parenthesis, it should be obtained by placing parentheses around the symbol subscripted by the relative constant.

Examples:

```
A, B, K, M,  (C(J),  J = 1,  10),  G(K)

A, B, K, M,  G(K)
```

The first of these two input lists is correct. The second is incorrect, but may be made correct with extra parentheses; i.e.,

```
A, B, K, M,  (G(K))
```

A relative constant definition must not appear to the left of the name of an array in the list of a READ DRUM statement.

### Relative Constants in an Argument List

A variable defined in one program may have its value transmitted to another program, where it is a relative constant and where, consequently, the value is used. This may be done by placing it in an argument list. The appearance of a relative constant in an argument list is sufficient to provide the necessary computation for the relative constant.

## Relative Constants in Common Statements

A relative constant value may be transmitted from one program to another by placing it in COMMON, but only if it is being transmitted from the calling to the called subprogram.

Example

Main Program

```
    .
    .
    .                                SUBROUTINE ABC
COMMON K                             COMMON I
                                     DIMENSION B(10)
K = 5                                A = B(I)
CALL ABC                                 .
    .                                    .
    .                                    .
    .                                    .
```

## FURTHER DETAILS ABOUT DO STATEMENTS

Triangular Indexing

Indexing such as

```
           DO   I = 1, 10
           DO   J = I, 10
or
           DO   I = 1, 10
           DO   J = 1, I
```

is permitted and simplifies work with triangular arrays. These are simply special cases of the fact that an index under control of a DO is available for general use as a fixed point variable.

The diagonal elements of an array may be picked out by the following type of indexing:

```
           DO   I = 1, 10
           A(I, I, I) = (some expression)
```

Status of the Cell Containing I

A DO loop with index I does not affect the contents of the object program storage location for I, except under the following circumstances:

1. An IF-type or GO TO-type transfer exit occurs from the range of the DO.

2. I is used as a variable in the range of the DO.

3. I is used as a subscript in combination with a relative constant whose value changes within the range of the DO.

Therefore, if a normal exit occurs from a DO to which cases 2 and 3 do not apply, the I cell contains what it did before the DO was encountered. After normal exit where 2 or 3 do apply, the I cell contains the current value of I.

What has just been said applies only when I is referred to as a variable. When it is referred to as a subscript, I is undefined after any normal exit and is the current value after any transfer exit.

# CHAPTER 2 — LIMITATIONS ON SOURCE PROGRAM SIZE

In translating a source program into an object program, FORTRAN internally forms and utilizes various tables containing certain items of information about the source program. These tables are of finite size and thus place restrictions on the volume of certain kinds of information which the source program may contain. If a table size is exceeded, a halt will occur while the object program is being compiled.

The relevant tables and the limitations are given below. In the following, the term "literal appearance" indicates that if the same item appears more than once, it must be counted more than once. Maximum table sizes are given first for the 32,000 word (32K) system, and then for the 8,000 word (8K) system.

### Statements with Statement Numbers

TEIFNO Table. The number of source statements which have statement numbers must not exceed 3000 (32K) or 750 (8K) in any single source program. (An input/output statement which has a statement number and whose list contains controlling parentheses counts as 2.)

### Fixed Point Constants

FIXCON Table. The number of different fixed point constants must not exceed 400 for 32K systems, or 100 for 8K systems. (For this purpose, constants differing only in sign are not considered different.)

### Floating Point Constants

FLOCON Table. The number of different floating point constants must not exceed 200 for the 32K system (50 for the 8K system) in any one arithmetic statement, nor more than 450 in any one source program. (Constants differing only in sign are not considered different, neither are numbers such as 4., 4.0, 40.E-1, etc., considered different.)

### Subscripted Variables

FORTAG Table. The total number of literal appearances of subscripted variables must not exceed 6000 for the 32K system, or 1500 for the 8K system.

### Subscripts

TAU Tables. The total number of different 1-, 2-, and 3-dimensional subscript combinations must not exceed 400, 360, and 300 respectively for the 32K system, or 100, 90, and 75 for the 8K system. Subscript combinations are considered different if corresponding subscripts, exclusive of addends, or corresponding "leading dimensions" of the subscripted arrays differ. "Leading dimensions" are the first dimension of a 2-dimensional array, and the first and second dimensions of a 3-dimensional array.

SIGMA Tables. The number of literal appearances of variables whose subscripts contain one or more unique addends, must not exceed 120 for the 32K system, or 30 in the 8K system in any one arithmetic expression.

Arithmetic Statements

LAMBDA Table. This table, and the BETA Table discussed immediately below, limit the size of arithmetic expressions both on the right-hand side of arithmetic statements, and as the arguments of IF and CALL statements. For each expression, λ must not exceed 1440 for the 32K system, or 360 for the 8K system (709 FORTRAN) in the equation

$$\lambda = n+4b+4a-3f+3p+2t+e+3, \text{ where}$$

n = number of literal appearances of variables and constants, except those in subscripts.

b = number of open parentheses, except those introducing subscripts.

p = number of appearances of + or -, except in subscripts or as unary operators (the + in A*(+B) is a unary operator).

t = number of appearances of * or /, except in subscripts.

e = number of appearances of **.

f = number of literal appearances of function names.

a = number of arguments of functions (for SINF(SINF(X)), a = 2).

BETA Table. With the above definitions, $\beta$ must not exceed 1080 for the 32K system, or 270 for the 8K system (709 FORTRAN) where

$$\beta = \lambda + 1-n-f$$

ALPHA Table. To determine whether an ALPHA table overflow will occur during the course of translation of an arithmetic statement, the following procedure should be carried out. Set the initial value of a counter to 3. Scanning the right-hand side of the statement in question, add 4 to the value of this counter for each left parenthesis encountered and subtract 4 for each right parenthesis encountered. This statement is compilable by the 709 FORTRAN System if and only if the counter value never exceeds 556 for 32K systems and 139 for 8K systems.

Arithmetic Statements: Fixed Point Variables

FORVAL Table. The total number of literal appearances of non-subscripted fixed point variables on the left-hand side of arithmetic statements, in input lists, in COMMON statements, and in argument lists for Fortran functions and Subroutine Subprograms must not exceed 2000 for the 32K system, or 500 for the 8K system.

FORVAR Table. The total number of literal appearances of non-subscripted fixed point variables on the right-hand side of arithmetic statements, and in the arguments of IF and CALL statements must not exceed 3000 for the 32K system, or 750 for the 8K system.

Arithmetic Statements: Functions

FORSUB Table. For 709 FORTRAN, the total number of distinct Arithmetic Statement functions must not exceed 140 for the 32K system, or 35 for the 8K system.

Transfer Statements

TRAD Table. The number of literal appearances of statement numbers mentioned in assigned GO TO and computed GO TO statements must not exceed 1, 000 for the 32K system, or 250 for the 8K system.

NLIST Table. The total number of different fixed point variables in assigned GO TO statements must not exceed 200 for the 32K system, or 50 for the 8K system in 709 FORTRAN.

TIFGO Table. The number of ASSIGN, and If- and GO TO-type statements in a source program must not exceed 1200 for the 32K system, or 300 for the 8K system in 709 FORTRAN.

STOP

TSTOPS Table. The total number of STOP statements in a source program must not exceed 1200 for the 32K system, or 300 for the 8K system.

DO

TDO Table. The total number of DOs must not exceed 600 for the 32K system, or 150 for the 8K system in any one source program. (A DO-implying parenthesis counts as a DO.)

DOTAG Table. The number of DOs must not exceed 200 for the 32K system, or 50 for the 8K system in any one nest of DOs.

CALL

CALLFN Table. The total number of CALL statements appearing in one source program must not exceed 2400 for the 32K system, or 600 for the 8K system.

Alphanumerical Arguments

HOLARG Table. Entries are made in this table when a CALL statement lists alphanumerical arguments. For every nH in a CALL statement, divide n by 6. Add 1 to the quotient if there is a remainder. Add 1 to this. The total of all such calculations must not exceed 3600 for the 32K system, or 600 for the 8K system.

FORMAT

FMTEFN Table. The total number of input-output statement references to FORMAT with numerical statement numbers must not exceed 2000 for the 32K system, or 500 for the 8K system.

FORMAT Table. For each FORMAT statement compute f as follows: Count all characters, including blanks, following the word FORMAT, up to and including the final right parenthesis.

Divide this count by 6. Add 1 to the quotient if there is a remainder.

The total of all the f values thus computed, must not exceed 6000 for one source program in 32K systems or 1500 for 8K systems.

Subprogram
Arguments

SUBDEF Table. The SUBDEF table arises from the SUBROUTINE and FUNCTION statements. An entry is made for the name of the subprogram being defined, and for each "dummy" argument contained in the argument list. The number of entries must not exceed 720 for the 32K system, or 180 for the 8K system.

Subprograms,
Functions
and
Input/Output
Statements

CLOSUB Table. One entry is made in this table for each closed subroutine, Fortran function, and Subroutine subprogram called in the source program. In addition, as many as three entries may be made for each input/output statement. The table must not exceed 6000 (32K), or 1500 (8K) entries, of which no more than 3000 or 750 respectively may be different.

Non-Executable
Statements

NONEXC Table. The number of non-executable statements in a source program must not exceed 1200 for the 32K system, or 300 for the 8K system.

DIMENSION

DIM Tables. The total number of 1-, 2-, and 3-dimensional variables mentioned in DIMENSION statements must not exceed 400, 400, and 360, respectively, for the 32K system, or 100, 100, and 90, respectively, for the 8K system.

SIZE Table. The number of arrays mentioned in a source program must not exceed 1160 for the 32K system, or 290 for the 8K system.

EQUIVALENCE

EQUIT Table. The total number of literal appearances of variables in EQUIVALENCE statements must not exceed 3000 (32K), or 750 (8K) in any one source program, nor 1200 (32K), or 300 (8K) in any one EQUIVALENCE statement.

FREQUENCY

FRET Table. The total number of numbers mentioned in FREQUENCY statements must not exceed 3000 for the 32K system, or 750 for the 8K system. For example FREQUENCY 30 (1, 2, 1) mentions four numbers.

COMMON

COMMON Table. The number of literal appearances of variables in COMMON statements must not exceed 2400 for the 32K system, or 600 for the 8K system.

# APPENDIX A — SOURCE PROGRAM STATEMENTS AND SEQUENCING

The precise rules which govern the order in which the source program statements of a 709 FORTRAN program will be executed can be stated as follows:

1. Control originates at the first executable statement.

2. If control has been with statement S, then control will pass to the statement indicated by the normal sequencing properties of S. (The normal sequencing properties of each FORTRAN statement are given below. If, however, S is the last statement in the range of one or more DO's which are not yet satisfied, then the normal sequencing of S is ignored and DO-sequencing occurs.)

## Non-Executable Statements

The statements FORMAT, DIMENSION, EQUIVALENCE, FREQUENCY, and COMMON are non-executable statements. In questions of sequencing they can simply be ignored.

If the last executable statement in the source program is not a STOP, RETURN, IF-type, or GO TO-type statement, then the object program is compiled to give the effect of depressing the Load Cards key following the last executable statement.

Every executable statement in a FORTRAN source program (except the first) must have some path of control leading to it.

Table of Source Program Statement Sequencing

| Statement | Normal Sequencing |
|---|---|
| a = b | Next executable statement |
| GO TO n | Statement n |
| GO TO n, $(n_1, n_2, \ldots, n_m)$ | Statement last assigned to n |
| ASSIGN i TO n | Next executable statement |
| GO TO $(n_1, n_2, \ldots, n_m)$, i | Statement $n_i$ |
| IF (a) $n_1$, $n_2$, $n_3$ | Statement $n_1$, $n_2$, or $n_3$ if (a) < 0, (a) = 0, or if (a) > 0, respectively. |

| Statement | Normal Sequencing |
|---|---|
| SENSE LIGHT i | Next executable statement. |
| IF (SENSE LIGHT i) $n_1$, $n_2$ | Statement $n_1$, $n_2$ if Sense Light i is On or Off, respectively. |
| IF (SENSE SWITCH i) $n_1$, $n_2$ | Statement $n_1$, $n_2$ if Sense Switch i is Down or Up, respectively. |
| IF ACCUMULATOR OVERFLOW $n_1$, $n_2$ | Statement $n_1$, $n_2$ if the 709 FORTRAN internal overflow indicator is On or Off, respectively. |
| IF QUOTIENT OVERFLOW $n_1$, $n_2$ | Statement $n_1$, $n_2$ if the 709 FORTRAN internal overflow indicator is On or Off, respectively. |
| IF DIVIDE CHECK $n_1$, $n_2$ | Statement $n_1$, $n_2$ if the Divide Check indicator is On or Off, respectively. |
| PAUSE or PAUSE n | Next executable statement. |
| STOP or STOP n | Terminates program. |
| DO n i = $m_1$, $m_2$ or<br>DO n i = $m_1$, $m_2$, $m_3$ | Do-sequencing, then next executable statement. |
| CONTINUE | Next executable statement. |
| END ($I_1$, $I_2$, $I_3$, $I_4$, $I_5$) | No sequencing; this statement terminates a problem. |
| CALL Name ($a_1$, $a_2$,..., $a_n$) | First statement of subroutine Name. |
| SUBROUTINE Name ($a_1$, $a_2$,..., $a_n$) | Next executable statement. |

| Statement | Normal Sequencing |
|---|---|
| FUNCTION Name ($a_1$, $a_2$,..., $a_n$) | Next executable statement. |
| RETURN | The statement or part of statement following call. |
| READ n, List | Next executable statement. |
| READ INPUT TAPE i, n, List | Next executable statement. |
| PUNCH n, List | Next executable statement. |
| PRINT n, List | Next executable statement. |
| WRITE OUTPUT TAPE i, n, List | Next executable statement. |
| FORMAT (Specification) | Not executed. |
| READ TAPE i, List | Next executable statement. |
| READ DRUM i, j, List | Next executable statement. |
| WRITE TAPE i, List | Next executable statement. |
| WRITE DRUM i, j, List | Next executable statement. |
| END FILE i | Next executable statement. |
| REWIND i | Next executable statement. |
| BACKSPACE i | Next executable statement. |
| DIMENSION v, v, v, ... | Not executed. |
| EQUIVALENCE (a, b, c, ...). (d, e, f, ...), .... | Not executed. |
| FREQUENCY n (i, j, ...). m(k, l, ...), ..... | Not executed. |
| COMMON A, B,... | Not executed. |

# APPENDIX B — TABLE OF SOURCE PROGRAM CHARACTERS

| CHARACTER | CARD | BCD TAPE | STORAGE | CHARACTER | CARD | BCD TAPE | STORAGE | CHARACTER | CARD | BCD TAPE | STORAGE | CHARACTER | CARD | BCD TAPE | STORAGE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 01 | 01 | A | 12<br>1 | 61 | 21 | J | 11<br>1 | 41 | 41 | / | 0<br>1 | 21 | 61 |
| 2 | 2 | 02 | 02 | B | 12<br>2 | 62 | 22 | K | 11<br>2 | 42 | 42 | S | 0<br>2 | 22 | 62 |
| 3 | 3 | 03 | 03 | C | 12<br>3 | 63 | 23 | L | 11<br>3 | 43 | 43 | T | 0<br>3 | 23 | 63 |
| 4 | 4 | 04 | 04 | D | 12<br>4 | 64 | 24 | M | 11<br>4 | 44 | 44 | U | 0<br>4 | 24 | 64 |
| 5 | 5 | 05 | 05 | E | 12<br>5 | 65 | 25 | N | 11<br>5 | 45 | 45 | V | 0<br>5 | 25 | 65 |
| 6 | 6 | 06 | 06 | F | 12<br>6 | 66 | 26 | O | 11<br>6 | 46 | 46 | W | 0<br>6 | 26 | 66 |
| 7 | 7 | 07 | 07 | G | 12<br>7 | 67 | 27 | P | 11<br>7 | 47 | 47 | X | 0<br>7 | 27 | 67 |
| 8 | 8 | 10 | 10 | H | 12<br>8 | 70 | 30 | Q | 11<br>8 | 50 | 50 | Y | 0<br>8 | 30 | 70 |
| 9 | 9 | 11 | 11 | I | 12<br>9 | 71 | 31 | R | 11<br>9 | 51 | 51 | Z | 0<br>9 | 31 | 71 |
| blank | blank | 20 | 60 | + | 12 | 60 | 20 | - | 11 | 40 | 40 | 0 | 0 | 12 | 00 |
| = | 8-3 | 13 | 13 | . | 12<br>8-3 | 73 | 33 | $ | 11<br>8-3 | 53 | 53 | , | 0<br>8-3 | 33 | 73 |
| - | 8-4 | 14 | 14 | ) | 12<br>8-4 | 74 | 34 | * | 11<br>8-4 | 54 | 54 | ( | 0<br>8-4 | 34 | 74 |

NOTE: There are two - signs. Only the 11-punch minus sign can be used in FORTRAN source program cards. Either minus sign may be used in input data to the object program; object program output uses the 8-4 minus sign.

The character $ can be used in FORTRAN only as Hollerith text in a FORMAT statement.

# APPENDIX C — TABLE OF LIBRARY FUNCTIONS

| Type of Function | Definition | No. of Args. | Name | Mode of Argument | Mode of Function |
|---|---|---|---|---|---|
| Natural Logarithm | Computes natural log for X, where X>0. | 1 | LOGF | Floating | Floating |
| Trigonometric Sine | Computes sine of an angle given in radians. | 1 | SINF | Floating | Floating |
| Trigonometric Cosine | Computes cosine of an angle in radians. | 1 | COSF | Floating | Floating |
| Exponential | X< 87.3 | 1 | EXPF | Floating | Floating |
| Square Root | Computes $\sqrt{\lvert X\rvert}$ | 1 | SQRTF | Floating | Floating |
| Arctangent | Computes floating arctangent | 1 | ATANF | Floating | Floating |
| Hyperbolic Tangent | Computes TANH X. Result has sign of X. | 1 | TANHF | Floating | Floating |

# APPENDIX D — SENSE SWITCH SETTINGS FOR 709 FORTRAN

| | | |
|---|---|---|
| Sense Switch 1 | UP | Cards containing the object program(s) are punched on-line. Actual tape unit B3 contains the object program of the source program compiled, or, if under monitor control, of the last source program compiled. |
| | DOWN | Actual tape unit B3 contains the object program for the last or only source program compiled. If under monitor control, tape unit B4 contains the object programs for all the source programs compiled, in the order compiled. No cards are punched. |
| Sense Switch 2 | UP | Produces, on actual tape unit B2, two files for the source program compiled, containing the source program and a map of object program storage. If under monitor control, actual tape unit A3 will contain two files for each program compiled and actual tape unit B2 will contain two files for the last program compiled. |
| | DOWN | Adds a third file for each program compiled (see above) containing the object program in terms of the symbolic code FAP (FORTRAN Assembly Program) on actual tape unit B2 (and A3, if under monitor control). |
| Sense Switch 3 | UP | No on-line listings are produced. |
| | DOWN | Lists on-line the first two or three files of tape unit B2, depending upon the setting of Sense Switch 2. |
| Sense Switch 4 | UP | Punched output, if any, is relocatable row binary cards. |
| | DOWN | Punched output is relocatable columnar binary cards. |

| | | |
|---|---|---|
| Sense Switch 5 | UP | Library subroutines will not be punched on cards or written on actual tape unit B3. |
| | DOWN | Causes library subroutines to be punched on cards or written on actual tape unit B3, depending upon whether Sense Switch 1 is Up or Down. |

# APPENDIX E — USING HAND-CODED SUBROUTINES WITH 709 FORTRAN COMPILED OBJECT PROGRAMS

Fortran function subprograms and Subroutine subprograms coded by hand or by a system other than FORTRAN can also be linked to FORTRAN programs. If coded in FAP and assembled through the FORTRAN Monitor, the linkage instructions will occur automatically. For hand-coding other than by FAP, rules for providing this linkage are given below.

It is necessary for hand-coded subprograms to conform to FORTRAN programs with regard to five conditions.

1. Transfer lists to called subroutines, if any.
2. Method of obtaining the variables (arguments) given in the calling sequence.
3. Saving and restoring index registers.
4. Storing results.
5. Method of returning to the calling program.

## Calling Sequence

A calling sequence for a subprogram, produced by FORTRAN consists of the following:

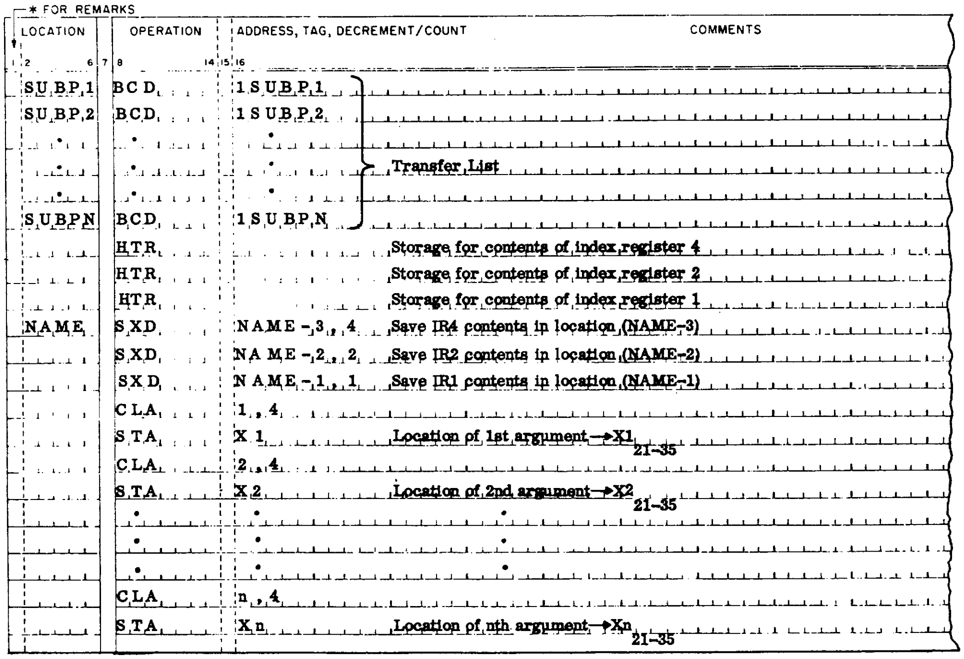
```
TSX       NAME, 4

TSX       LOCX1

TSX       LOCX2

 .          .

 .          .

 .          .

TSX       LOCXn
```

The calling sequence consists of n+1 words. The first is an instruction which causes transfer of control to the subprogram. The remaining n words include one for each argument. The "TSX" in these words is never executed. In case an argument consists of an array, one instruction determines the entire array; the address of that instruction specifies the location of the first element of the array, i.e., element $A_{1,1,1}$. If the argument is Hollerith data, the location given is that of the first word of the block containing the data.

### Transfer List, Prologue, and Index Register Saving

The first instructions of a subprogram will consist of a transfer list and a prologue in that order. The transfer list contains the symbolic names of the lower level subprograms and functions, if any, that the subprogram calls. The prologue obtains and stores the locations given in the calling sequence. It will consist of the CLA and STA instructions necessary for each argument. If it is desired, index registers may be saved.

The instructions below show such a transfer list and prologue.



```
* FOR REMARKS
LOCATION  OPERATION  ADDRESS, TAG, DECREMENT/COUNT   COMMENTS
1 2    6 7 8     14 15 16
 SUBP1    BCD        1SUBP1      ⎫
 SUBP2    BCD        1SUBP2      ⎪
   .       .            .        ⎪
   .       .            .        ⎬ Transfer List
   .       .            .        ⎪
 SUBPN    BCD        1SUBPN      ⎭
          HTR                      Storage for contents of index register 4
          HTR                      Storage for contents of index register 2
          HTR                      Storage for contents of index register 1
 NAME     SXD        NAME-3,4      Save IR4 contents in location (NAME-3)
          SXD        NAME-2,2      Save IR2 contents in location (NAME-2)
          SXD        NAME-1,1      Save IR1 contents in location (NAME-1)
          CLA        1,4
          STA        X1            Location of 1st argument→X1
                                                              21-35
          CLA        2,4
          STA        X2            Location of 2nd argument→X2
                                                              21-35
           .          .                       .
           .          .                       .
           .          .                       .
          CLA        n,4
          STA        Xn            Location of nth argument→Xn
                                                              21-35
```

### RESULTS

A FORTRAN function must place its (single) result in the Accumulator prior to returning control to the calling program.

A Subroutine subprogram must place each of its results in a storage location. (Such a subprogram need not, of course, return results.) A result represented by the n th argument of a CALL statement is stored in the location specified by the address field of location (n, 4).

### Return

Transfer of control to the calling program is effected by

1. Restoring the Index Registers to their condition prior to transfer of control to the subprogram.

2. Transferring to the calling program. The required steps are:

* FOR REMARKS

| LOCATION | | OPERATION | ADDRESS, TAG, DECREMENT/COUNT | COMMENTS |
|---|---|---|---|---|
| 1 2 6 | 7 | 8 14 | 15 16 | |
| | | LXD | NAME-3,4 | RESTORE CONTENTS OF XR4 |
| | | LXD | NAME-2,2 | RESTORE CONTENTS OF XR2 |
| | | LXD | NAME-1,1 | RESTORE CONTENTS OF XR1 |
| | | TRA | n+1,4 | RETURN. n=NUMBER OF ARGUMENTS. |
| | | | | |

### Entry

Unlike a Fortran compiled subprogram, a hand-coded subprogram may have more than one entry point. A hand-coded subprogram used with a FORTRAN calling program may be entered at any desired point, provided that a subprogram name acceptable to FORTRAN is assigned to each selected entry point. All the above mentioned conditions, must of course, be satisfied at each entry point. The entry point name by which a FORTRAN calling program refers to a DAP subprogram need not have been used in the original symbolic DAP coding.

### System Tape Subroutines

As discussed on page 59, hand-coded subprograms as well as Library functions, may be placed on the system tape of the FORTRAN system. When a FORTRAN source program mentions the name of such a subprogram, it is handled in exactly the same way as a library function.

### Alphanumerical Information

Hand-coded subprograms may handle alphanumerical information. This information is supplied as an argument of a CALL statement. The form for an alphanumerical argument is

$$nHx_1x_2. \dots x_n$$

The following example illustrates the method of storing alphanumerical information.

Example:

CALL TRMLPH (8, C, 13HFINAL RESULTS)

the characters 13H are dropped, and the remaining information stored:

| Location | Contents |
|---|---|
| X | FINALb |
| X+1 | RESULT |
| X+2 | Sbbbbb (b represents a blank - $60_8$) |
| X+3 | $777777777777_8$ |

The address X is given in the calling sequence for the CALL statement.