IBM Customer Engineering

Manual of Instruction

7100 Central Processing Unit

Issued to:_____

Department or Branch Office_____ Telephone Number_____

Address_____ City_____ State_____

Home Address_____ City_____ State_____

If this manual is mislaid, please notify the above address.

Printed in U.S.A.
Form 223-6860

CONTENTS

1.0.00 GENERAL INFORMATION

1.1.00 INTRODUCTION

The Central Processing Unit (CPU) is the center of activity in the 7090 system. Its functions are much the same as the central processing units in the 704 and 709 systems. The CPU controls and does most of the information processing for the system except some input-output (I-O) operations.

The instructions in the program control the CPU. The logic of the circuits of the CPU control how the information is processed to perform an instruction.

1.2.00 GENERAL MACHINE LOGIC

Figure 1.2-1 shows the paths that information can follow as it goes from one point to another in the CPU. When the information will be moved and where it will be moved to is controlled according to the instruction being operated on by the CPU.

1.3.00 PHYSICAL LAYOUT

The CPU for the 7090 system occupies two SMS Sliding Gate Modules. The modules are called CPU 1 and CPU 2.

1.4.00 MACHINE LANGUAGE

Instructions and data used by the CPU must be in the form of binary numbers. Inputs to the 7090 system can be in many forms but, for the CPU to use the information, the numbers first have to be decoded by a program and put into binary form.

2.0.00 EXTERNAL FUNCTIONS OR OPERATIONS

2.1.00 INPUT-OUTPUT

The direct data input and output for the 7090 system is core storage. Information can be read from cards or tape, and information can be taken out of the system on to cards, tape, or a printed form. However, as data passes between I-O and the CPU it must go through core storage.

There is only one core storage and it must be used by both the CPU and the I-O system. Only one of the two can use core storage at one time; interlocking circuits control this usage. Some control lines into or out of the CPU may also be considered inputs or outputs. These control lines run mostly between the CPU and the I-O devices. These control lines allow the CPU program to control the whole system, yet allow the I-O system operate independently of the CPU.

2.2.00 CONTROL

The program controls the 7090 system. Instructions of the program are brought into the CPU one at a time. The bit configuration of the instruction word determines what the CPU is to do. When the instruction is brought into the CPU, the operation code goes to the program register. From there the instruction is decoded to direct operation of the units concerned.

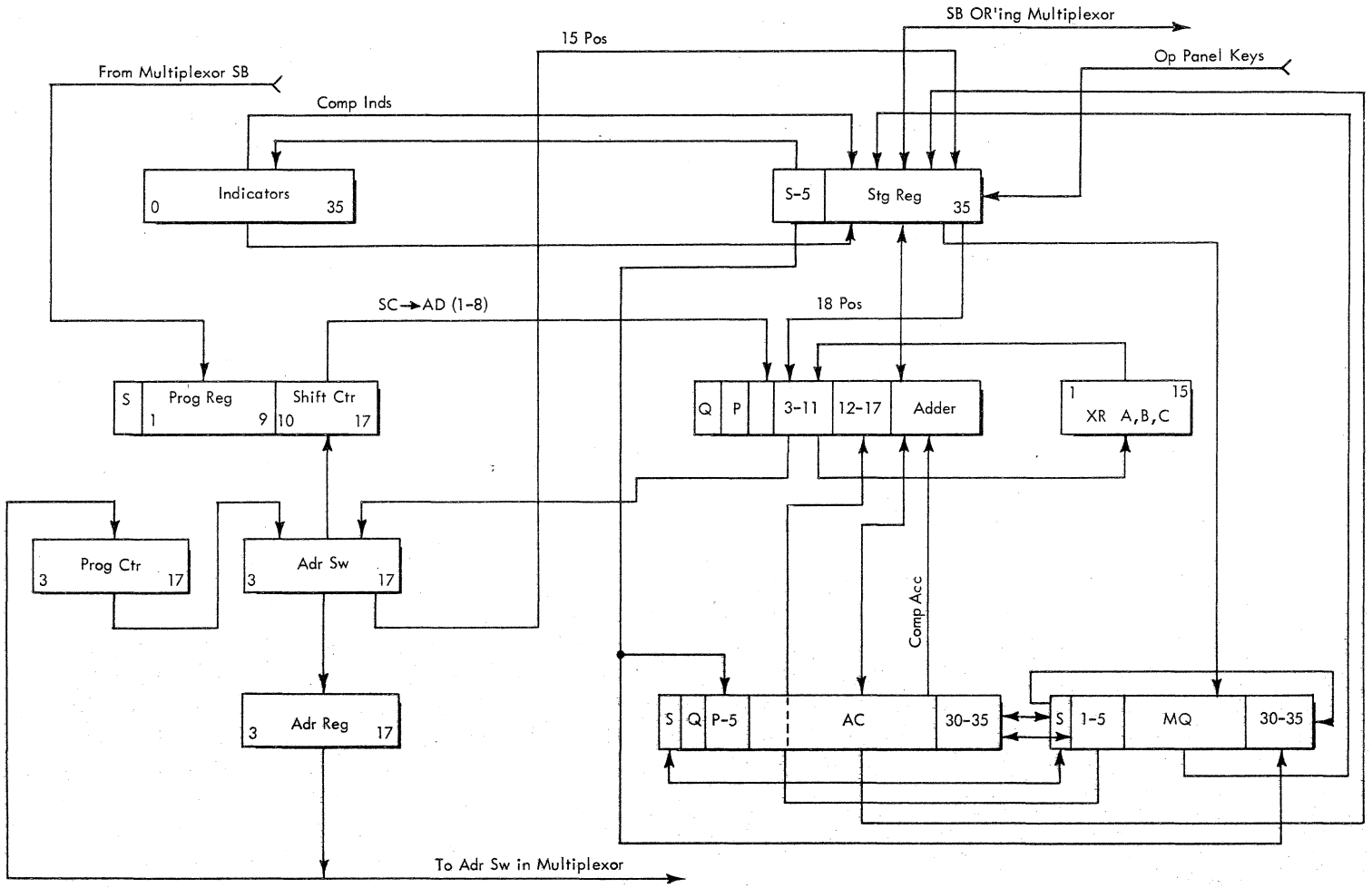


FIGURE 1.2-1. 7090 CENTRAL PROCESSING UNIT

3.0.00 INTERNAL FUNCTIONS

3.1.00 FUNCTIONAL COMPONENTS

Figure 1.2-1 shows the functional components of the CPU. Some components are made up of shift cells so information can be shifted from one position to another or so information can be put into and taken out of the register at the same time.

A basic shift cell is shown in Figure 3.1-1. Assume that one of the outputs of this shift cell is connected to an input. This condition would be the same as putting information into the shift cell, while at the same time sending information from this shift cell to another. An input and a set pulse are required to turn the top trigger on. The drop of the hold turns the lower trigger off. Even though the hold drops slightly before the set rises, circuit delays keep the top trigger conditioned. Because of these delays the top circuit will be turned on when the set pulse occurs. The hold coming up allows the lower trigger to turn on. The end of the set pulse will turn the top trigger off. If the set and hold pulse occur and there is no input, the lower trigger is turned off.

The registers that contain shift cells are:

1. Storage register (SR)
2. Accumulator register (AC)
3. Multiplier-Quotient register (MQ)
4. Index register (XR)

The program register (PR) and the address register (AR) have one standard trigger for each register position.

The sense indicators have one standard trigger for each position in addition to another trigger used to invert.

The adders and address switches are groups of AND and OR connected to perform the indicated function.

The program counter and shift counter are binary counters.

Binary Counters

A binary counter unit is a group of counter positions connected together so the input pulses to the low order position are counted. A counter unit can be designed to count up or down, determined by intended use. The program counter is an example of a count up counter. It normally receives a stepping impulse once for each instruction and this is used to obtain the address of the next sequential instruction in core storage. The shift counter is an example of a count down counter. It is preset to a desired number of shifts and, each time a shift occurs, a stepping pulse is sent to the low order position of the counter. When the counter has been stepped down to zero, it signals the CPU that the desired number of shifts have been taken.

A basic counter position is shown in Figure 3.1-2. This counter position can be turned on to give a one output and turned off to give a zero output. To give a one output, the counter must receive a plus input to +0(2). This input is supplied by +A(1) or +A(3).

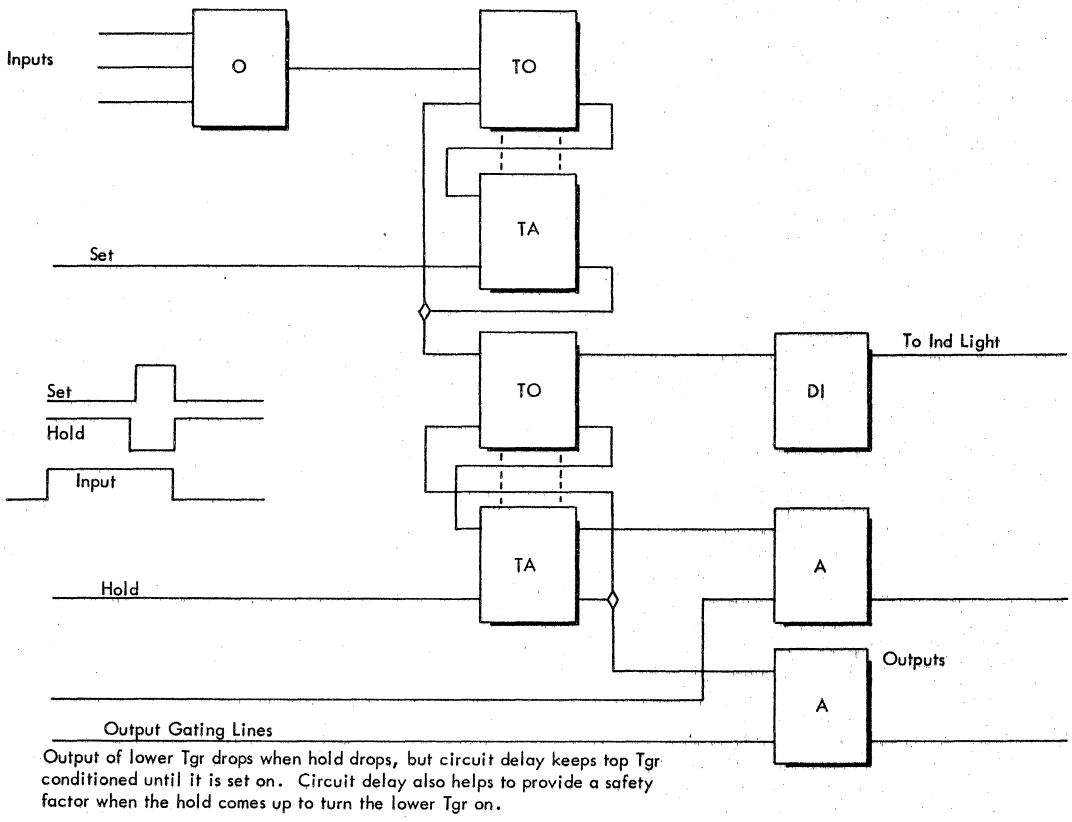


FIGURE 3.1-1. SHIFT CELL

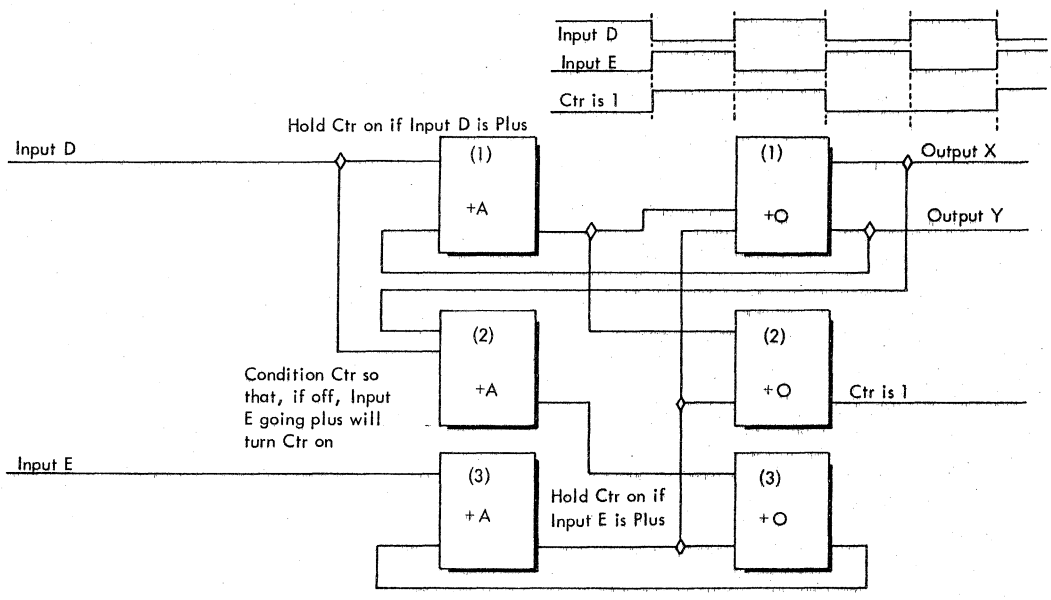


FIGURE 3.1-2. BASIC COUNTER POSITION-STEPPING

The counter position is held on by either the input D or the input E lines being plus. If input D is plus, input E is minus; if input D is minus, input E is plus.

Assuming that the counter position is on and holding through +A(1), it will be turned off when input D goes down and input E goes up.

If the counter position is being held on through +A(2), it will stay on when input E goes minus and input D goes plus. Under these conditions, the hold used to keep the counter position on is switched from +A(3) to +A(1).

When the counter position has just been turned off, input D is minus and input E is plus. The next shift of the inputs will make input D plus and input E minus. This does not turn on the counter position, however, +A(2) supplies a plus input to the bottom of +A(3) through +0(3). The next plus shift on input E will once again turn on the counter position through +A(3).

Referring to Figure 3.1-3 it can be seen that a simple counter unit can be made by connecting the X and Y outputs of one position to the D and E inputs of the next higher position. For a count up counter, output X is connected to input E and output Y is connected to input D. For a count down counter, output X is connected to input D and output Y is connected to input E. A counter unit can be made as large as needed by adding more counter positions.

Each time a counter position is turned off in a count up counter, the condition of the next higher counter position is reversed (if on, it is turned off; if off, it is turned on). In a count down counter, each time a counter position is turned on, the condition of the next higher position is reversed.

The chart on Figure 3.1-4 illustrates the conditions of the counter positions in a three position count up counter for a series of stepping pulses supplied to the low order position (position 3).

Figure 3.1-5 illustrates the conditions of the counter positions in a three-position count down counter for a series of stepping pulses supplied to the low order position (position 3).

Counter Set and Reset. The basic counter circuit, Figure 3.1-2, shows only the stepping inputs. However, a counter unit also has a set and a reset input. The set input is necessary so the counter can be set to some predetermined number. The reset is used to put the counter in some initial starting condition, such as resetting all counter positions to zero.

In Figure 3.1-2, the set input would be a plus pulse going to the input of +0(1) and +0(3). The reset for a count up counter would be a minus pulse sent to the input of +A(3) and the reset for a count down counter would be a minus pulse at the input of +A(1).

Speed-Up for Counter Circuits. Connecting the output of one counter position to the input of the next higher position does produce a counter. The stepping of a low order counter position may step all positions above it. The counter circuits introduce delay into this action so the speed of operation is not acceptable for the 7090. Therefore, circuits are attached to the counter unit to speed up counter operation. Figure 3.1-6 and Figure 3.1-7 show the logic of the speed-up circuits used with the shift counter and program counter.

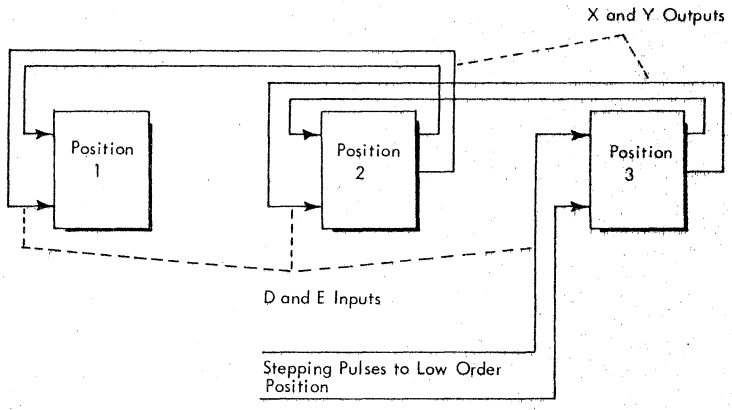


FIGURE 3.1-3. SIMPLE THREE-POSITION COUNTER UNIT

	Pos 1	Pos 2	Pos 3	In Ctr
	Off	Off	Off	0
Step 1	Off	Off	On	1
Step 2	Off	On	Off	2
Step 3	Off	On	On	3
Step 4	On	Off	Off	4
Step 5	On	Off	On	5
Step 6	On	On	Off	6
Step 7	On	On	On	7
Step 8	Off	Off	Off	0

FIGURE 3.1-4. CONDITIONS OF COUNTER POSITIONS, THREE-POSITION COUNT UP COUNTER

	Pos 1	Pos 2	Pos 3	In Ctr
	On	On	On	7
Step 1	On	On	Off	6
Step 2	On	Off	On	5
Step 3	On	Off	Off	4
Step 4	Off	On	On	3
Step 5	Off	On	Off	2
Step 6	Off	Off	On	1
Step 7	Off	Off	Off	0
Step 8	On	On	On	7

FIGURE 3.1-5. CONDITIONS OF COUNTER POSITIONS, THREE-POSITION COUNT DOWN COUNTER

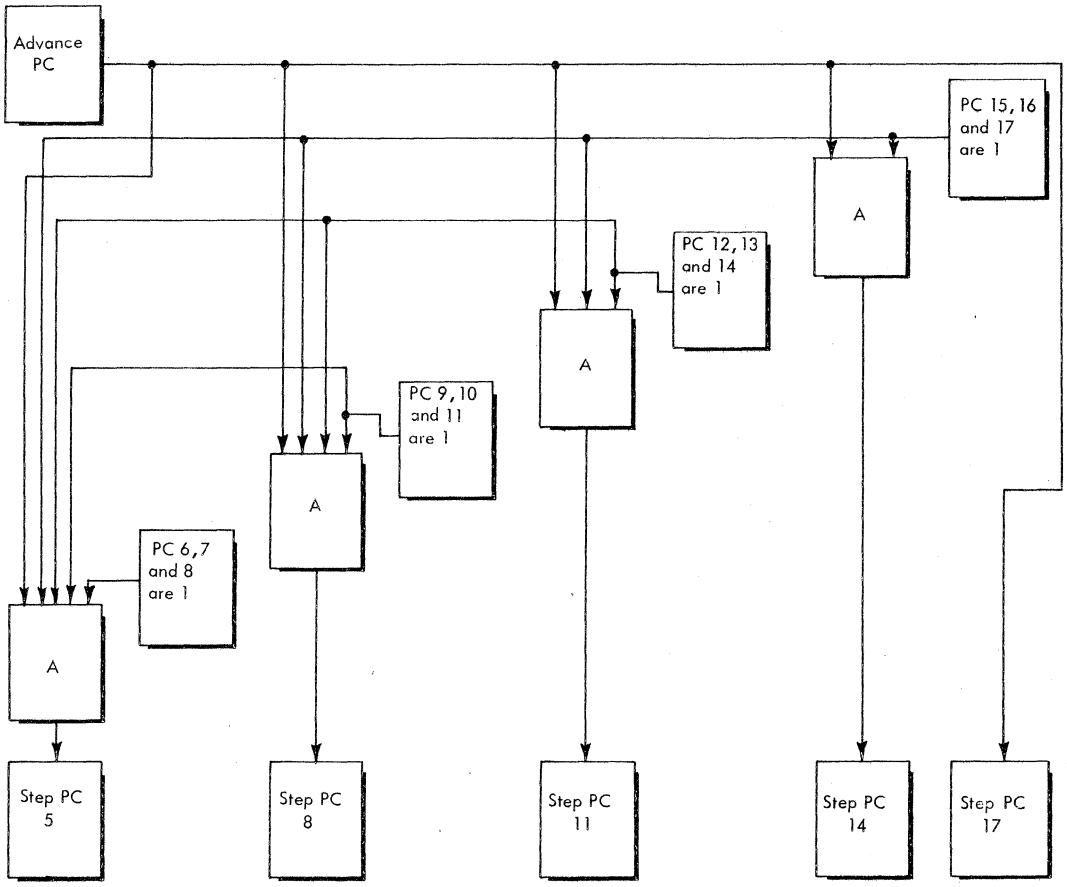


FIGURE 3.1-6. PROGRAM COUNTER STEP GENERATOR (3.05.08.1)

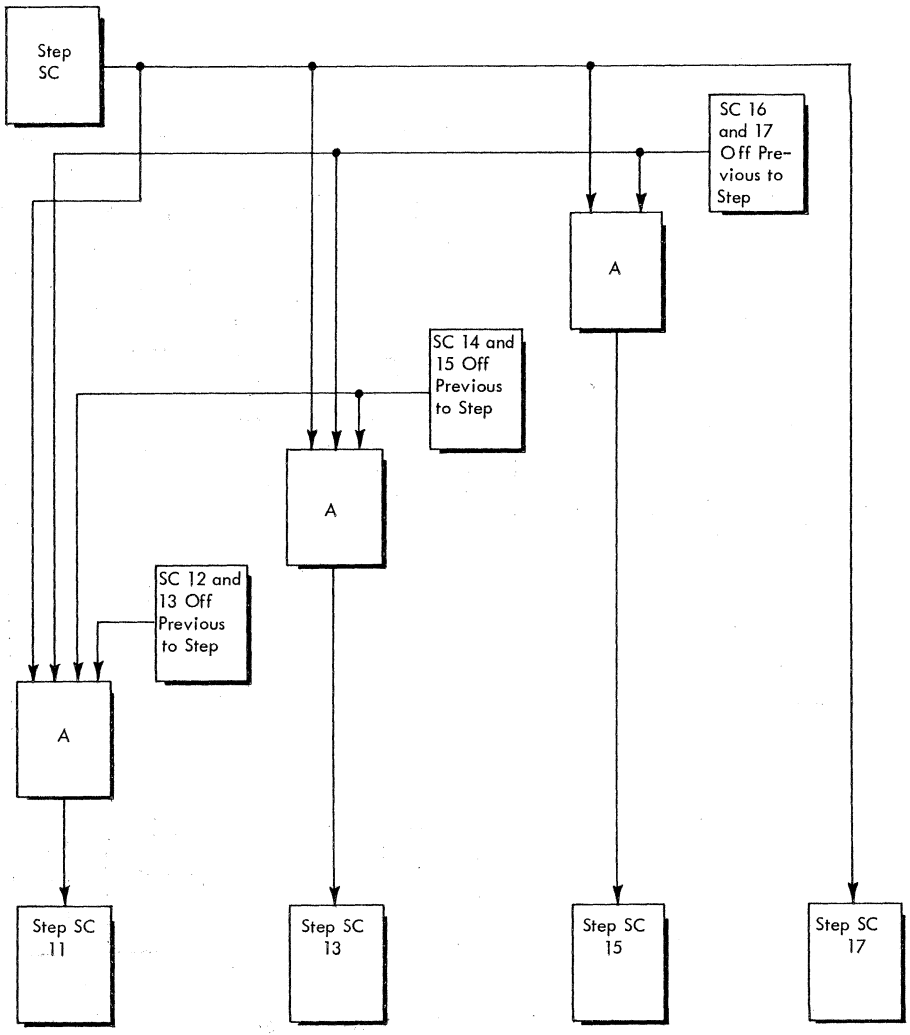


FIGURE 3.1-7. SHIFT COUNTER STEP CONTROL

3.1.01 Storage Register (SR)

The storage register is a 37-position shift cell register and the register positions are labeled (S), (1-35), and (Q). All instructions and data coming into or leaving the CPU go through the storage register, positions (S), (1-35), except a portion of the instruction that goes directly to the program register. Storage register positions (S), (1-35), and (Q) are also used for holding factors and partial results during the execution of many instructions.

3.1.02 Accumulator Register (AC)

The accumulator is a 38-position shift cell register used in nearly every arithmetic operation. The register positions are labeled (S), (Q), (P), (1-35). Positions (S), (1-35) accommodate the word in standard operations. Positions (Q), and (P) are used as overflow positions because the sum of two 35-position numbers can be greater than 35 positions. Postion P also holds the S bit of a word during logical operations. As the name implies, the accumulator is the unit into which results accumulate; it does not perform the addition.

3.1.03 Multiplier-Quotient Register (MQ)

The multiplier-quotient register is a 36-position shift cell register with the positions labeled (S), (1-35). This register has several uses. During a multiply operation it holds the multiplier; after the multiply operation it holds the least significant half of the product. During a divide operation, it holds the least significant half of the dividend; after the divide operation it holds the quotient. In several floating point operations, the MQ holds the least significant 35 bits of the result.

3.1.04 Sense Indicator Register (SI)

The sense indicator register is a 36-position shift cell register, labeled (0-35). Each position of the register contains two triggers. One of the triggers is used to retain information in the register, and the other trigger is used as a remembering device during an invert operation. The sense indicator register is controlled completely by the program and is not used by the computer as a part of its arithmetic operations. It can be used as a set of switches which are set and tested by the program to check the progress or direction of the program. The SI register may also be used to store words or parts of words temporarily and, in this way, it is useful for altering and testing words.

3.1.05 Index Registers (XR)

There are three 15-position shift cell registers called index registers A, B, and C. The register positions are labeled (3-17). The three registers are identical in operation and are used for temporary instruction address modification. They are activated by the tag positions of an instruction and can be used singly or in parallel or combinations. They modify an address by adding the compliment of their contents to the address; in effect, the address is reduced by the contents of the index register.

There are many instructions which operate on the index registers and make them useful programming tools for counting, word alteration, program loop control and so forth.

3.1.06 Program Register (PR)

The program register is a ten-position trigger register whose positions are labeled (S), (1-9). The program register receives the operation code of the instruction directly from the storage bus and holds the operation code throughout the execution of the instruction. The output of the program register is decoded to initiate and control the CPU so it will execute the instruction. Positions (1-5) contain the primary operation part of the operation code; positions (6-9) contain the secondary part.

3.1.07 Shift Counter (SC)

The shift counter is an eight-position counter, labeled (10-17). It sometimes is considered part of the program register because, for some instructions, the shift counter will contain part of the code that will determine system operation. The shift counter has two additional main functions. The number of shifts required for a shifting operation can be set into the shift counter; when stepped down with each shift, it will indicate when shifting is complete. It is also used to hold the unit and class address of an I-O unit during a read or write select instruction.

3.1.08 Program Counter (PC)

The program counter is a 15-position counter, labeled (3-17), used to locate instructions in storage. The storage addressing circuits are set to the address contained in the program counter at the beginning of each instruction cycle. The program counter is normally stepped at the end of an instruction cycle to cause the sequential selection of instructions. This counter can also be changed by transfer, trap, or skip instructions, giving flexibility to the programming.

3.1.09 Address Register (AR)

The address register is a 15-position register whose positions are labeled (3-17). The address register gets the desired core storage address before a word is brought out of core storage. The address goes from the address register to the address selection components in core storage.

3.1.10 Address Switches (AS)

The address switch is a 15-position gating circuit with positions labeled (3-17). The switches receive information from the program counter or the adders and this information can be gated through the switches to the address register, shift counter, or storage register.

3.1.11 Adders (AD)

The adders are transistor switching circuits used to combine binary numbers or words into a binary sum. There are 37 separate adders (Q), (P), (1-35) in the adder unit capable of combining two binary words.

The adders do the arithmetic calculation of the system. Basically all the system will do is add. Subtraction is done by adding a complement number. Multiplication is accomplished by a series of additions and shifts. Division is accomplished by a series of complement additions and shifts.

Figure 3.1-8 shows a basic adder circuit. The adder circuit is designed to use three inputs at one time with one of the three inputs being a carry from the next low order position. The adder produces two outputs, a sum and a carry. The adders are designed to follow the example of binary addition illustrated in Figure 3.1-9.

Some adders have several inputs because the adders have multiple uses. However, only two of the outside inputs and the carry in can be active at the same time. The adding function of these multipurpose adders is the same as shown in the basic adder circuit, Figure 3.1-8.

Adder Unit and Look Ahead Circuits (LAC). The basic principle behind connecting individual adders together to make an adder unit is to take the carry out of one adder and connect to the carry in of the next high order position adder. This would mean that, if all adder positions contained a one and a one was added to the low order position, a carry would have to ripple through the adder unit from the low order position to the high order position. As the carry ripples through the adders the logic circuits introduce a delay into this action. The 7090 is too fast to accept these conditions. To overcome this slow ripple condition in the adders, the 7090 adders have associated circuits that speed up the carry operation. These circuits determine how many adder positions the carry must go through, then send the carry almost immediately to a higher order adder a few positions from the initial carry impulse. These carry speed-up circuits are called look ahead circuits (LAC). The logic of the operation of the look ahead circuits is shown in Figure 3.1-10 and Figure 3.1-11. Notice that for look ahead purposes the adders are grouped into six groups of five adders each and one group of six adders. The output of the first stage LAC feeds the inputs of the second stage LAC.

All adder positions, except the Q position, have two LAC outputs. One is the OR LAC and the other is the AND LAC. The OR LAC gives a usable output to the first stage LAC when a bit is sent to either input 1 or input 2 or both. The AND LAC gives a usable output to the first stage LAC when bits are sent to both inputs 1 and 2.

There are two outputs from the first stage LAC. One output is the carry out LAC and the other output is active when all adders in the group are receiving at least one input. This latter output is simply called look-ahead. The CO LAC is active when any combination of bits in that particular group of adders would cause a carry out of the high order adder of the group.

The second stage LAC uses the outputs of the first stage LAC, along with a line to indicate a carry in or a bit to adder 35. These LAC initiate a carry into the low order adder of the next higher group.

Following through the first and second stage LAC, note that if adder 35 has an AND LAC output and adders 34-30 have OR LAC outputs, a carry in would be sent to adder 29. If all adders have an OR LAC output, these outputs would cause a carry in to be sent to adders 29, 24, 19, 14, 9, 4, and Q almost simultaneously if a carry into adder 35 occurred.

Note that a carry has to ripple through a maximum of 6 adders rather than through 37 because of the look ahead circuits, and the carry operation through the adders has been greatly speeded up.

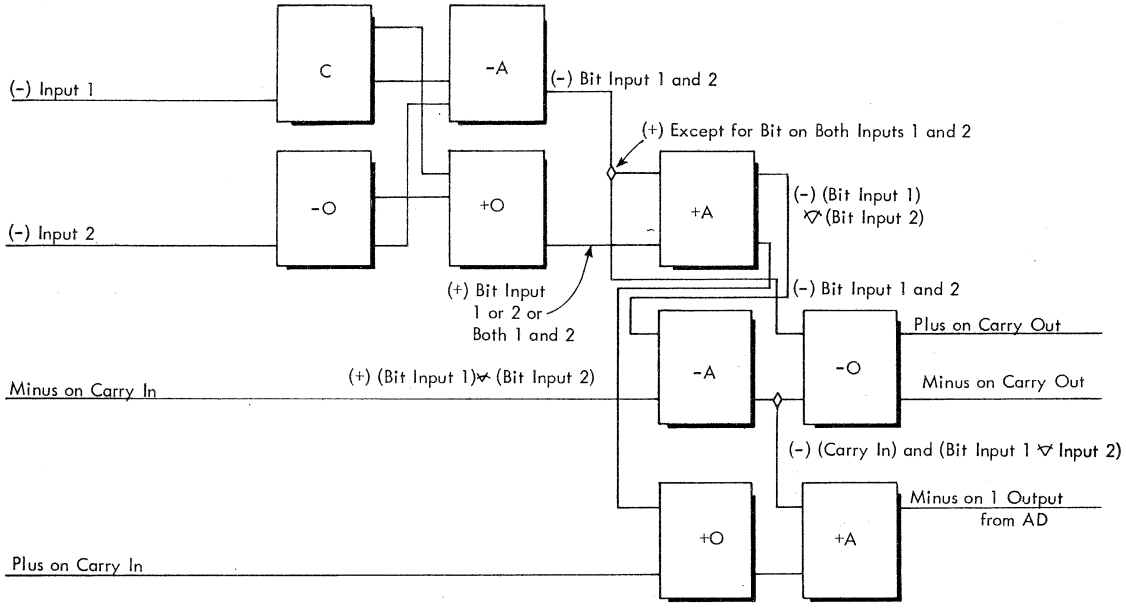
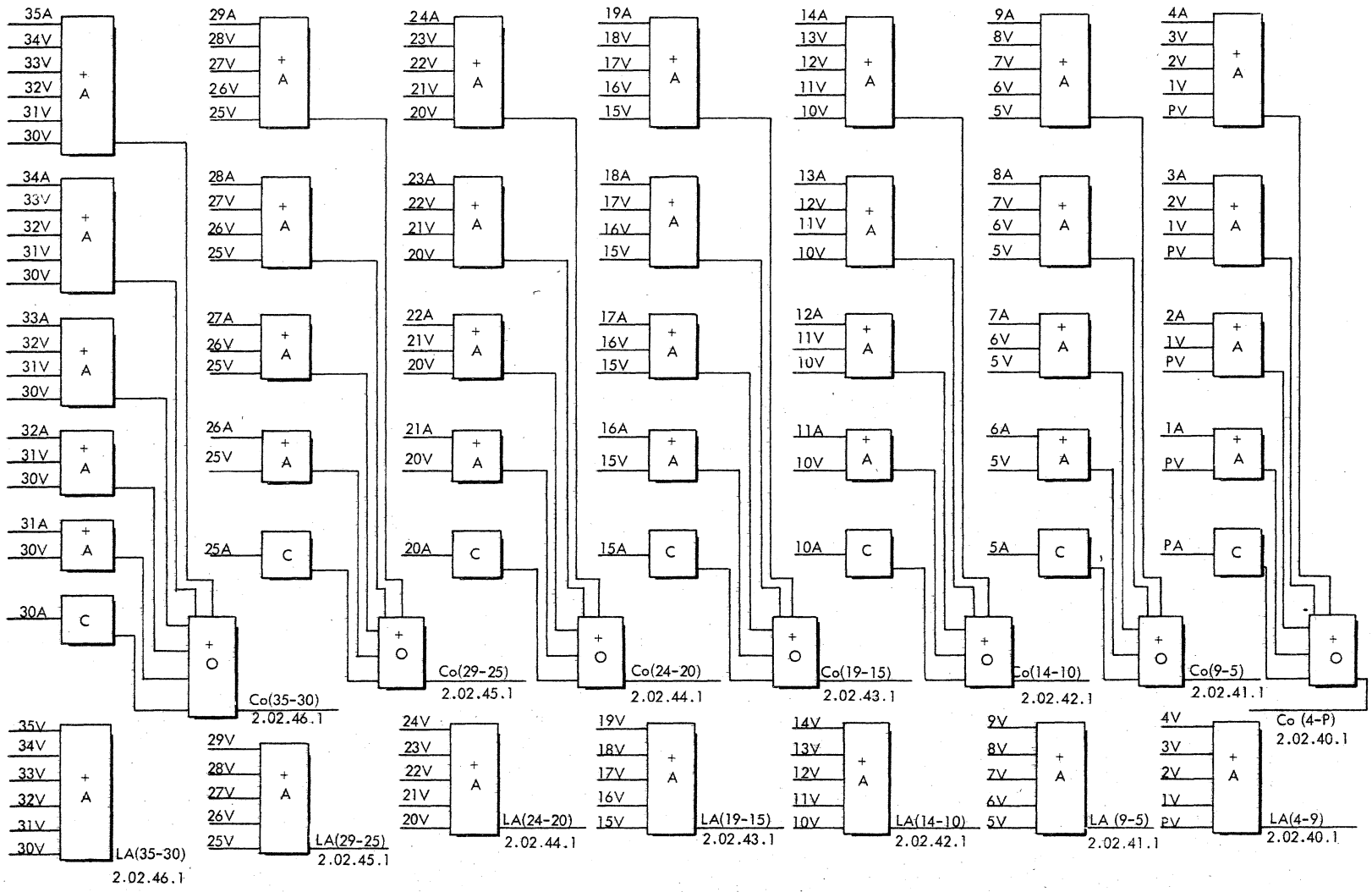


FIGURE 3.1-8. BASIC ADDER CIRCUITS - 7090 CPU

INPUTS	Input 1	0	1	0	0	1	0	1	1
	Input 2	0	0	1	0	1	1	0	1
	Carry In	0	0	0	1	0	1	1	1
OUTPUTS	Sum	0	1	1	1	0	0	0	1
	Carry	0	0	0	0	1	1	1	1

FIGURE 3.1-9. EXAMPLE OF BINARY ADDITION



A = 2 Bits V = At Least 1 Bit Co = Carry Out LA = Look Ahead

FIGURE 3.1-10. FIRST STAGE LOOK AHEAD CIRCUITS

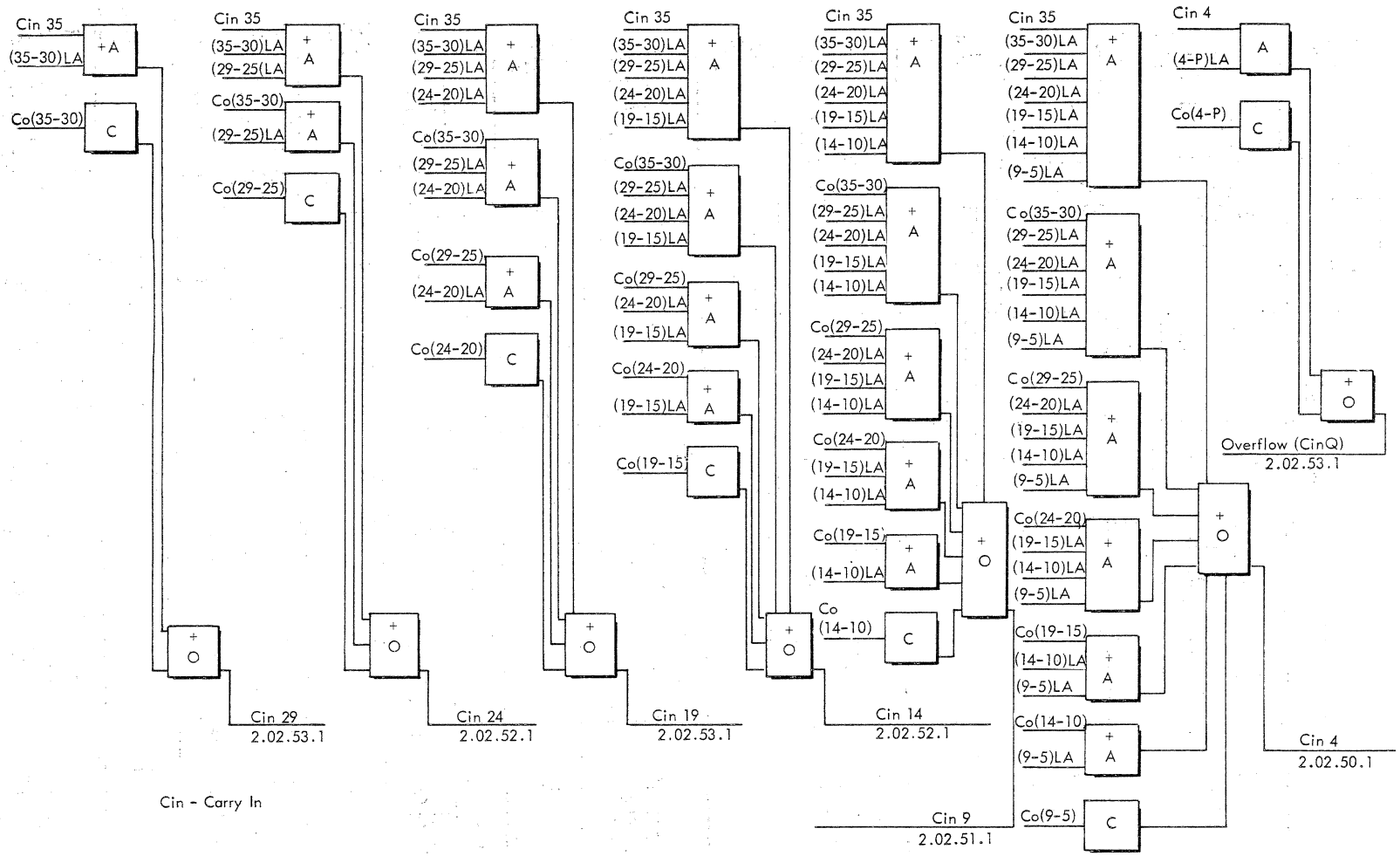


FIGURE 3.1-11. SECOND STAGE LOOK AHEAD CIRCUITS

Carry From Adder 9 to Adder 8. A special circuit can prevent or allow a carry from adder 9 to adder 8. The reason for this is that a floating-point word has two separate parts, a characteristic (positions 1-8) and a fraction (positions 9-35). During some floating-point operations, a carry out of the fraction must not be allowed to gate to the characteristic.

3.2.00 INSTRUCTION DECODING AND PROCESSING

The output of the program counter is gated to the address register through the address switches. The output of the address register is then used to locate the address of the instruction in core storage. The output of core storage is sent to the storage register and the operation code is also sent to the program register. The output of the program register then is decoded to instruct the system.

3.2.01 Operation Decoders

The output of the program register feeds two decoders. The outputs of positions 1-5 feed the primary operation decoders (POD), and the outputs of positions 6-9 feed the secondary operation decoders (SOD). The primary operation is used to establish the basic execution control routing. The secondary operation decoder is used only for the sense, or primary operations 76 instructions.

Some similar instructions have the same primary operation. Direct outputs of the program register or storage register are sent to individual machine circuits to cause the machine to operate according to the minor differences among these similar instructions.

3.2.02 Control Circuits

The control circuits cause the machine to operate according to the decoding of the instruction. Specific control circuits cause data to be moved and functions to be performed so the machine gives the desired results of the instruction.

3.2.03 Pulses

The control circuits require a great number of pulses and gates. These pulses and gates, obtained by mixing clock pulses from the multiplexor clock, function in the 7090 as the circuit breakers function in electro-mechanical card machines. They are used to establish proper sequence of operation in the system.

3.3.00 BASIC CYCLE

The basic cycle in the CPU is 2.4 microseconds. It is divided into 12 equal times of 200 milli-microseconds each. The 12 times are given to the CPU by the multiplexor.

There are three different types of cycles used by the CPU. These are:

1. Instruction I cycle
2. Execution E cycle
3. Logic L cycle

Every instruction has an I cycle. This cycle brings the instruction from core storage to the CPU, where the instruction is decoded. Some instructions require only the I cycle for their execution; most instructions require additional cycles.

E cycles are used when information is to be moved between core storage and the CPU to execute an instruction.

L cycles are used to execute an instruction when information from core storage is not needed.

Most instructions use only one E or L cycle for their execution, but some instructions require multiple E or L cycles. For example: a convert instruction requires several E cycles; a multiply instruction requires several L cycles.

3.4.00 COMMON DATA FLOW AND TIMING

3.4.01 I Cycle

I cycle operation is much the same for all instructions. E and L cycle operations vary depending on the instruction.

I cycle operation is shown in Figure 3.4-1. The end operation trigger (END OP TGR) is turned on during the last cycle of an operation and for an initial starting condition. This causes the I time trigger to be turned on. Also with the end operation trigger on, the address of the instruction is sent to core storage. The instruction is then taken from that address in core storage and sent to the CPU on the storage bus (SB). The instruction is put into the storage register (SR) and the operation code is sent to the program register (PR). Note the difference between instructions with a bit in positions 1 and 2 and those without a bit in 1 or 2. Few instructions have a bit in positions 1 or 2.

When the instruction is decoded the computer will decide what kind of a cycle is to be taken. Most instructions requiring an L cycle have operation codes 0-177 or from 700 on. Instructions with operation codes between these limits usually require an E cycle. The exceptions to these general L and E time call rules are in the MF systems pages.

Some instructions require only one cycle; therefore, the end operation trigger is turned on during I time. This forces "go to I time" on Systems 8.00.12.1. "Go to I time" will turn on the master I time trigger and will prevent turning on of the master E or L time triggers.

The program counter is advanced to give the location of the next sequential instruction in core storage. The address portion of the instruction is sent to the address register. For primary operations 76 it is sent to the shift counter. The address in the address register is used to locate a particular address in storage if the next cycle is an E cycle. Part of the address is sent to the shift counter on primary operations 76. The shift counter is used on these operations along with the program register to instruct the system.

3.4.02 Indirect Addressing

Indirect addressing is a programming device which permits changing an instruction address. Bits in positions 12 and 13 of an instruction are signals for indirect addressing. This causes an E cycle, during which the word located at the original instruction address is read out of storage and its address portion is substituted for the instruction address. The instruction then operates on the new address. Only indexable instruc-

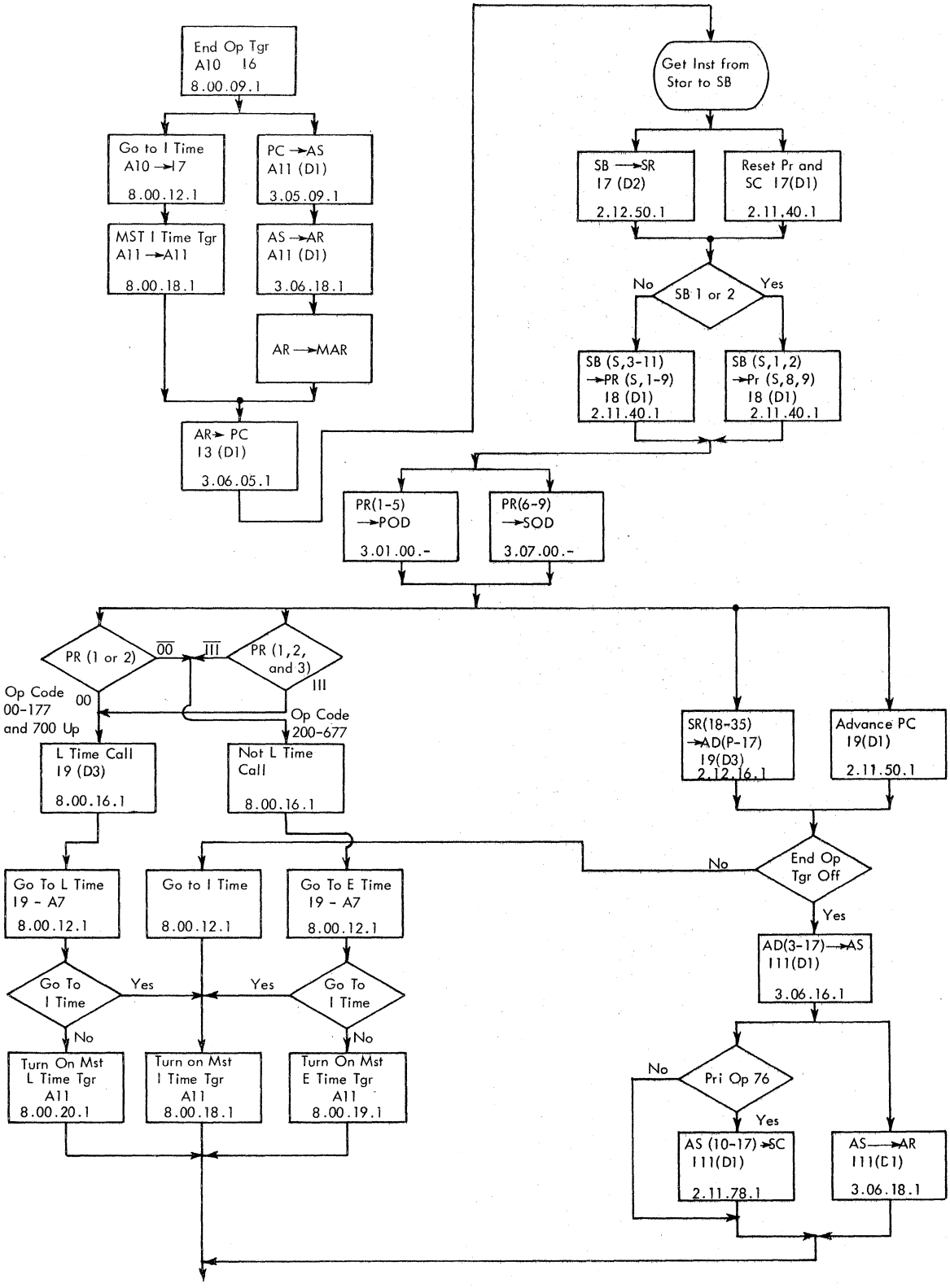


FIGURE 3.4-1. INSTRUCTION CYCLE

tions may be indirectly addressed. The only change in operation of an instruction caused by indirect addressing is the insertion of an E cycle. As with direct address instructions, indexing is automatic.

Indirect addressing controls are shown in Figure 3.4-2. Once an indirect addressing condition is recognized, the IA control trigger is turned on to block normal operations and an E cycle is initiated. When the E cycle is completed, the trigger is turned off, and normal execution is resumed.

3.5.00 INSTRUCTIONS

3.5.01 Word Transmission Instructions

Word transmission instructions are necessary to move information into and out of the CPU. Factors must be brought in from core storage and results must be returned. Instructions are available to move parts of words or whole words to or from core storage.

Store STO +0601 Figure 3.5-1

This instruction moves a full word to core storage. The contents of the AC(S,1-35) replace the contents of storage location X. The AC is unchanged. The store prefix, decrement, tag, address, and MF store control lines are activated so a full word can be put into core storage.

Store Logical Word SLW +0602

This instruction replaces the contents of storage location X with the contents of the AC(P-35). The AC is unchanged. Execution of this instruction is identical to that of store except for the routing of AC(P) to the storage register-sign position. AC(P-35) is gated to SR(S-35) on Systems 2.12.02.1.

Store MQ STQ -0600 Figure 3.5-1

The contents of core storage location X are replaced by the contents of the MQ register. The contents of the MQ are unchanged. This instruction operates similarly to store, except that the word sent to the SR comes from the MQ rather than from the AC.

Store Zero STZ +0600 Figure 3.5-1

This instruction causes zeros to be placed in all positions of storage location X. The operation of this instruction is similar to store, except that nothing is put on the SB. Therefore, when the storage bus is gated to core storage, zeros are put into that location of storage.

Store Prefix STP +0603 Figure 3.5-2

This instruction places the contents of AC positions (P, 1 and 2) into core storage location X, positions (S, 1 and 2). The contents of the AC and storage positions (3 through 35) are unchanged. MF store prefix and MF store control are activated to cause core storage to store what is on SB(S, 1 and 2). To get the information to the storage bus, it must be moved from the accumulator to the SR.

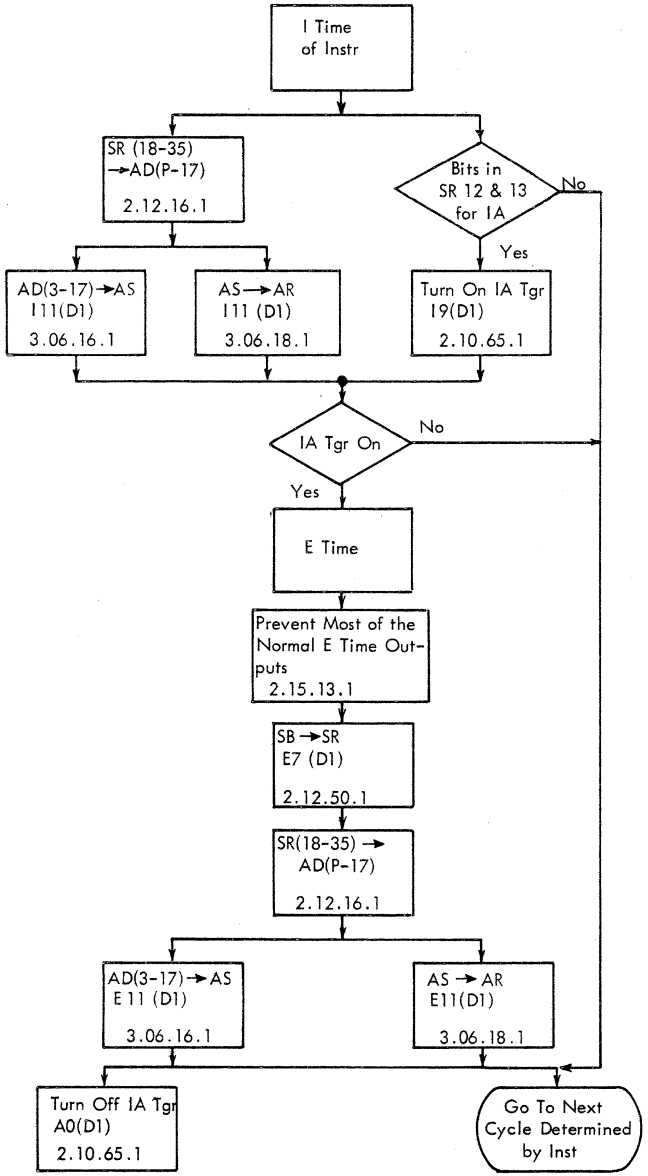


FIGURE 3.4-2. INDIRECT ADDRESSING

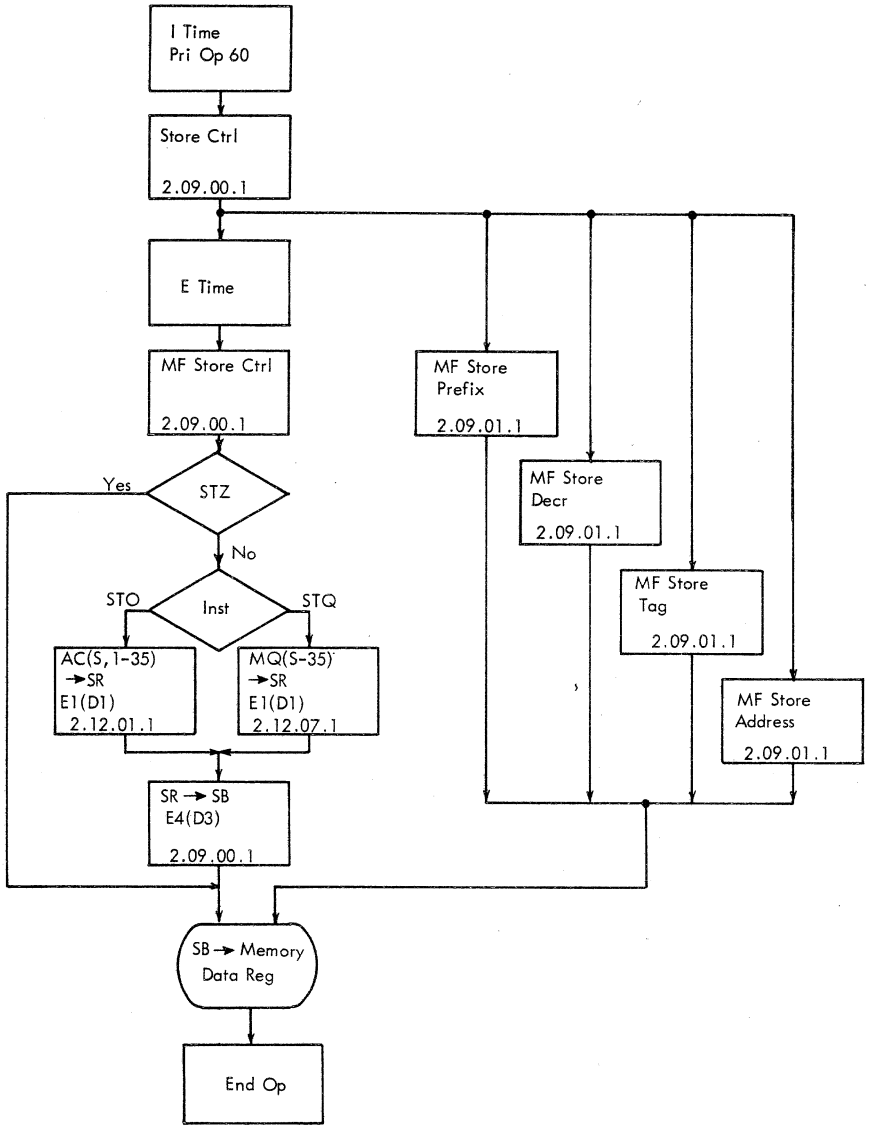


FIGURE 3.5-1. STO +0601; STQ -0600; STZ +0600

Store Decrement STD +0622 Figure 3.5-2

The contents of AC(3-17) replace the contents of positions (3-17) of core storage location X. The remaining storage positions are unchanged, and the AC is unchanged. This instruction operates similarly to store prefix, except that MF store decrement rather than MF store prefix is made active.

Store Tag STT +0625 Figure 3.5-2

The contents of AC(18-20) replace the contents of positions (18-20) of core storage location X. The AC and the remaining positions of core storage are unchanged. This instruction operates similarly to store prefix, except that MF store tag rather than MF store prefix is made active.

Store Address STA +0621 Figure 3.5-2

The contents of AC(21-35) replace the contents of positions (21-35) of core storage location X. The remaining positions of storage and the contents of the AC are unchanged. This instruction also operates similarly to store prefix, except that MF store address rather than MF store prefix is made active.

Store Left Half MQ SLQ -0620 Figure 3.5-3

The contents of positions (S-17) of the MQ replace the contents of positions (S-17) of storage location X. The remaining storage positions and the contents of the MQ are unchanged. MF store prefix, MF store decrement, and MF store control lines are activated so SB(S-17) can be read into storage. The word is sent from the MQ to the SR so it can be put on the SB.

Store Instruction Location Counter STL -0625 Figure 3.5-4

The contents of the program counter, which contains the location of the STL instruction plus one, replace the contents of positions (21-35) of storage location X. The contents of the PC and the remaining positions of storage are unchanged. The MF store address and MF store control lines are activated so the address portion of the storage word can be changed. The PC contents are sent to core storage on the SB. The contents of the PC are sent to the SR through the AS and the output of the SR feeds the SB.

Load MQ LDQ +0560 Figure 3.5-5

The contents of the MQ are replaced by the contents of storage location X. The storage word is put into the SR, and the output of the SR is sent to the MQ.

Exchange AC and MQ XCA +0131 Figure 3.5-6

The contents of the AC(S, 1-35) are interchanged with the contents of the MQ(S, 1-35). AC positions Q and P are reset.

Exchange Logical Accumulator and MQ XCL -0130 Figure 3.5-7

This instruction interchanges the contents of AC(P-35) and the contents of the MQ (S-35). Positions (S) and (Q) of the AC are cleared. Execution is identical to that of XCA except for the handling of AC positions (S) and (P). The contents of the MQ are put into the SR. The contents of the SR(S, 1-35) and the AC(S, 1-35) are interchanged, putting the original MQ in the AC. Gating the output of the SR(S, 1-35) to the MQ(S, 1-35) places the original AC contents in the MQ. AC positions (Q) and (P) are cleared because of normal shift cell operations.

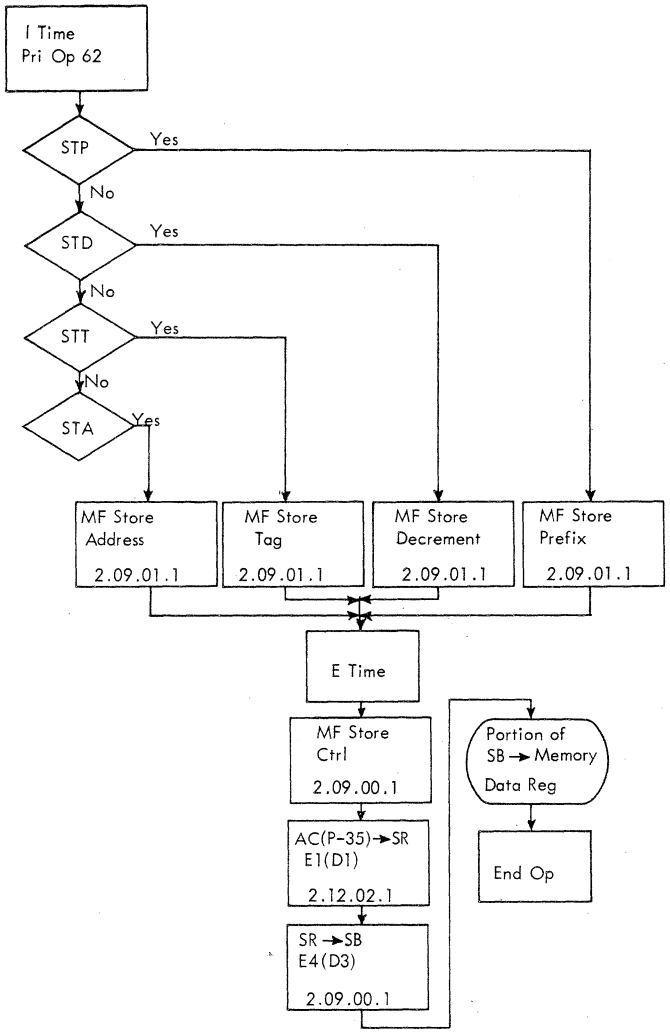


FIGURE 3.5-2. STP +0630; STD +0622; STT +0625; STA +0621

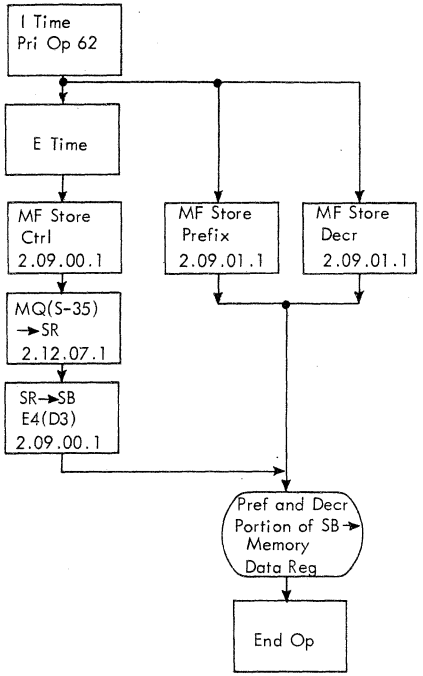


FIGURE 3.5-3. SLQ - 0620

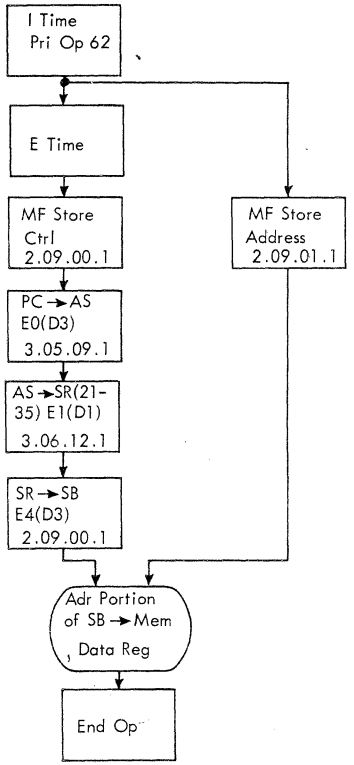


FIGURE 3.5-4. STL -0625

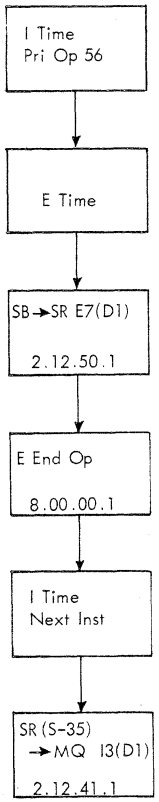


FIGURE 3.5-5. LDQ +0560

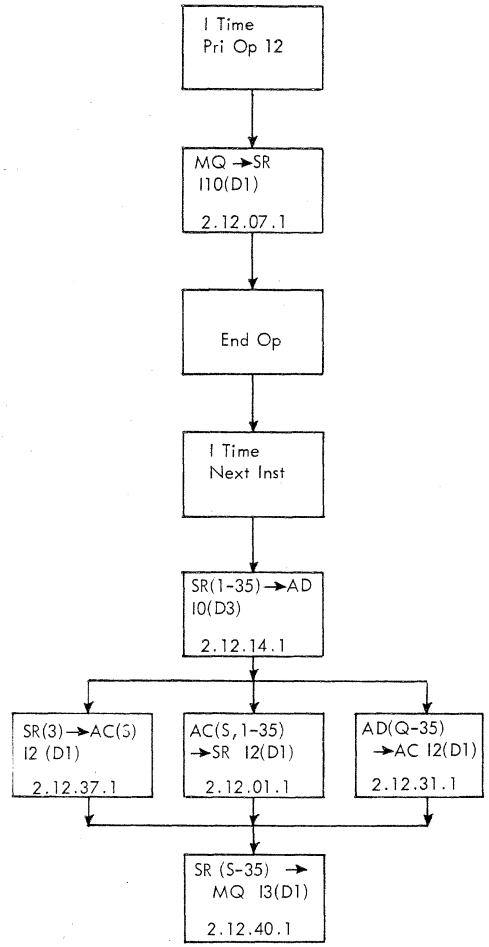


FIGURE 3.5-6. XCA +0131

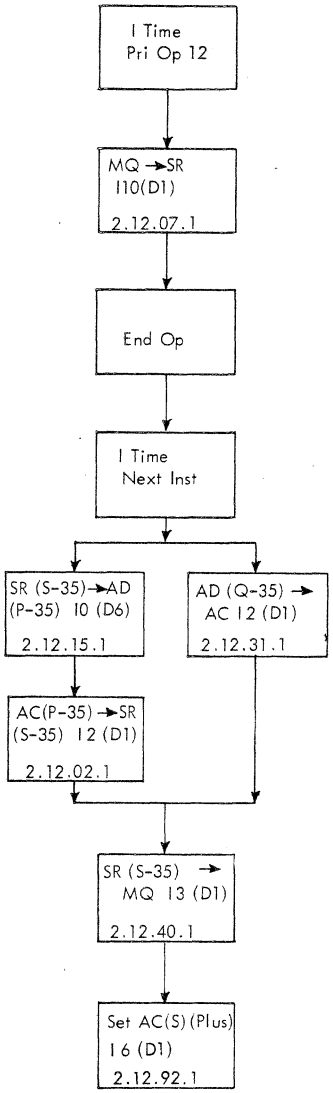


FIGURE 3.5-7. XCL -0130

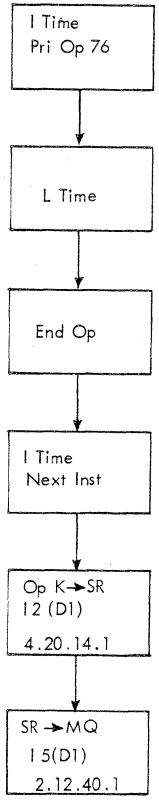


FIGURE 3.5-8. ENK +0760...0004

Enter Keys ENK +0760...0004 Figure 3.5-8

The word represented in console keys (S, 1-35) replaces the contents of the MQ. This is a primary operation 76 instruction. The word in the keys is put into the SR and the output of the SR is gated into the MQ.

3.5.02 Shifting Instructions

Shift instructions are used to align words, or for fast multiplication or division by a power of 2. Shifting moves the bits of the AC or MQ, or both, to the right or left within the registers. Bits shifted out of the end of a register are lost and bits shifted away from either end of a register are replaced by zeros. Because the shift counter receives only the last eight positions of the instruction address, the maximum number of shifts possible is $225_{10}(11\ 111\ 111_2)$.

Shifting to the left is the same as multiplying by a power of 2; shifting to the right reduces or divides by a power of 2. The number of shifts is equal to the exponent.

Accumulator Left Shift ALS +0767 Figure 3.5-9

This instruction causes the contents of the AC(Q-35)* to be shifted left a number of places equal to the eight low order positions of the address. Zeros replace any bits shifted away from position (35). Bits shifted past Q are lost. A "1" shifted into the P position turns on the AC overflow indicator.

This instruction has a primary operation 76 which causes the SC to receive the eight low order positions of the address. As long as the SC value is not zero the AC is shifted and the SC is stepped down once for each master clock pulse. The computer will continue in L cycles until the SC is stepped down to zero. This stops shifting and turns on the end operation trigger.

Long Left Shift LLS +0763 Figure 3.5-10

For this instruction the contents of the MQ and AC (except the sign positions) are shifted left the number of places designated by the eight low order positions of the address. The AC sign is set to agree with the MQ sign. Bits shifted past Q are lost and bits shifted away from MQ(35) are replaced by zeros. Bits shifted into P cause the AC overflow indicator to be turned on.

The number of shifts to be taken is set into the SC. As long as the SC value is not zero, the MQ and AC are shifted left and the SC is stepped down once for each clock pulse. Shifting and L cycles continue until the SC equals zero. Shifting is then stopped and the end operation trigger is turned on.

Logical Left Shift LGL -0763 Figure 3.5-10

This instruction shifts the contents of AC(Q-35) and MQ(S-35) left the number of places designated by the address. Bits shifted from MQ(1) enter MQ(S), and from MQ (S) enter AC(35). Bits shifted into AC(P) cause the AC overflow indicator to be turned on. Bits shifted past AC(Q) are lost and bits shifted away from MQ(35) are replaced by zeros. The operation of LGL is similar to LLS except for handling of the MQ sign.

* This text section uses (Q - 35) to represent (Q, P, 1 - 35).

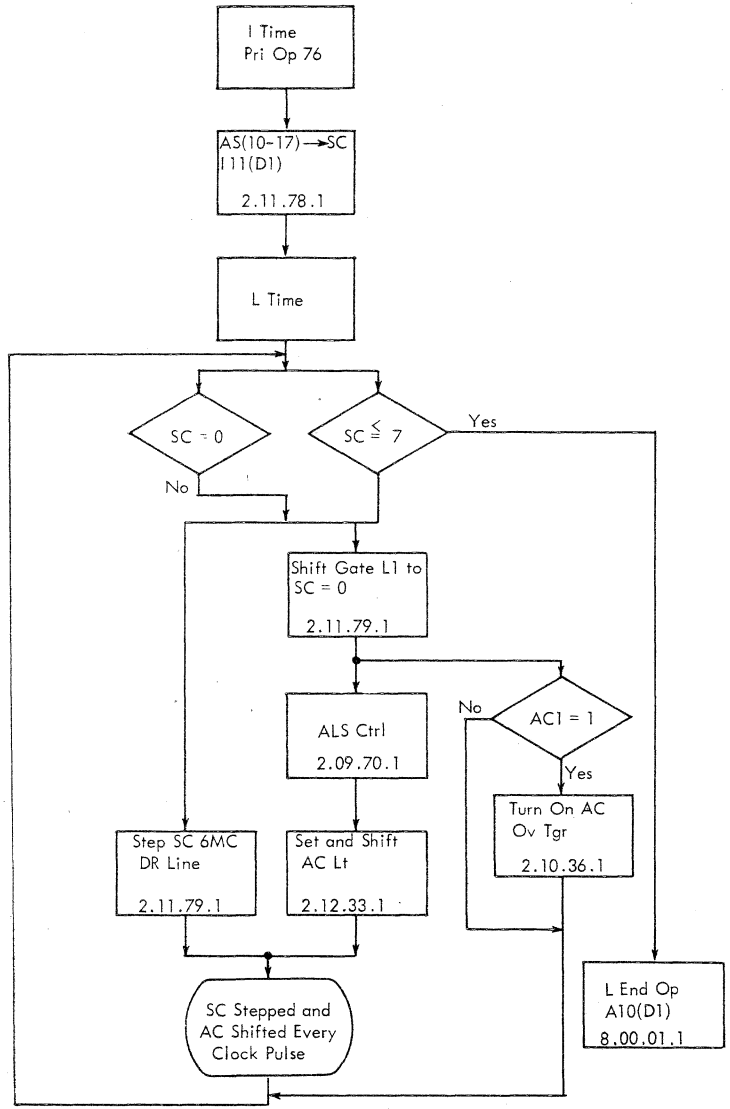


FIGURE 3.5-9. ALS +0767

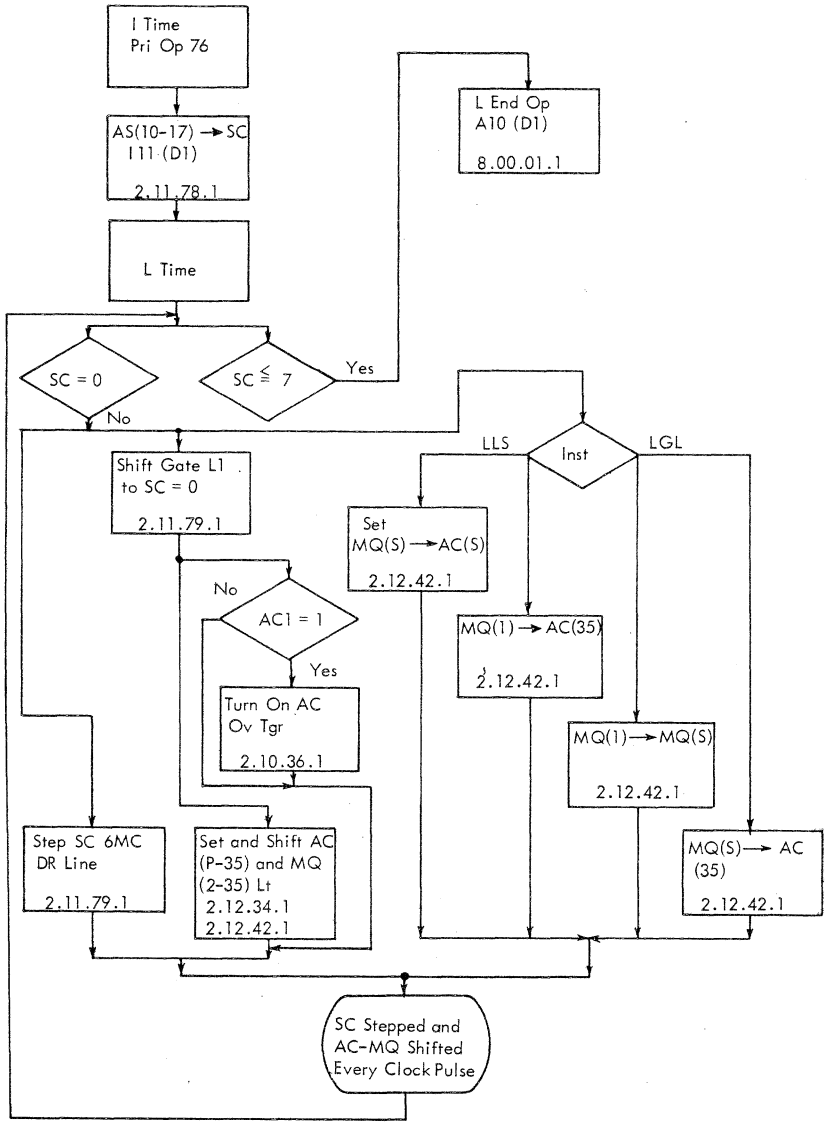


FIGURE 3.5-10. LLS +0763; LGL -0763

Accumulator Right Shift ARS +0771 Figure 3.5-11

This instruction causes the contents of AR(Q-35) to be shifted right the number of places indicated by the address. Bits shifted away from Q are replaced by zeros; bits shifted past position (35) are lost. ARS is similar to ALS except for direction of the shifting.

Long Right Shift LRS +0765 Figure 3.5-12

The contents of the MQ and AC (except the sign positions) are shifted right the number of places indicated by the address. The MQ sign is set to agree with the AC sign. Bits shifted away from Q are replaced by zeros and shifted past AC(35) enter MQ(1). Bits shifted past MQ(35) are lost. The SC is stepped down and the AC and MQ are shifted once for each clock pulse, until the SC equals zero. When the shift counter steps down to zero, shifting is stopped and the end operation trigger is turned on.

Logical Right Shift LGR -0765 Figure 3.5-12

This instruction shifts the contents of the AC(Q-35) and MQ(S-35) right the number of places designated by the address. Bits shifted out of AC(35) are entered into MQ(S) and from MQ(S) to MQ(1). Bits shifted past MQ(35) are lost. The operation of LGR is the same as LRS except for the handling of the MQ sign.

Rotate MQ Left RQL -0773 Figure 3.5-13

This instruction shifts the contents of the MQ, including the sign, left the number of places designated by the address. Bits shifted out of MQ(1) enter the sign position and from the sign position enter MQ(35). The SC is stepped and a shift occurs once for each clock pulse. When the SC equals zero, shifting is stopped and the end operation trigger is turned on.

3.5.03 Fixed-Point Arithmetic Instructions

Fixed-point arithmetic instructions are the basic calculating instructions. Addition is the foundation of all of these instructions, and parts of the ADD instruction will be found in all instructions. All arithmetic instructions use the accumulator.

Clear and Add CLA +0500 Figure 3.5-14

The instruction clears the AC(S, Q-35) and loads positions (S, 1-35) with the contents of the storage location indicated by the address. The word that is to be put into the AC is brought from core storage to the SR during the E cycle. In the I cycle of the next instruction, positions (1-35) are gated from the SR through the AD to the AC. The sign position is gated directly from the SR to the AC.

Clear and Subtract CLS +0502 Figure 3.5-14

This instruction replaces the contents of the AC with the word from the core storage location indicated by the address. The sign of the word is reversed. Execution of this instruction is the same as the CLA except that the sign of the word is inverted during the E cycle after the word has entered the SR.

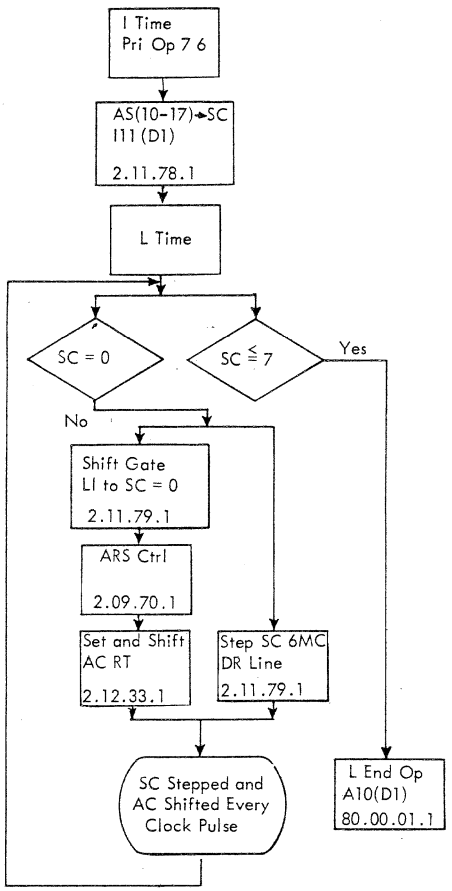


FIGURE 3.5-11. ARS +0771

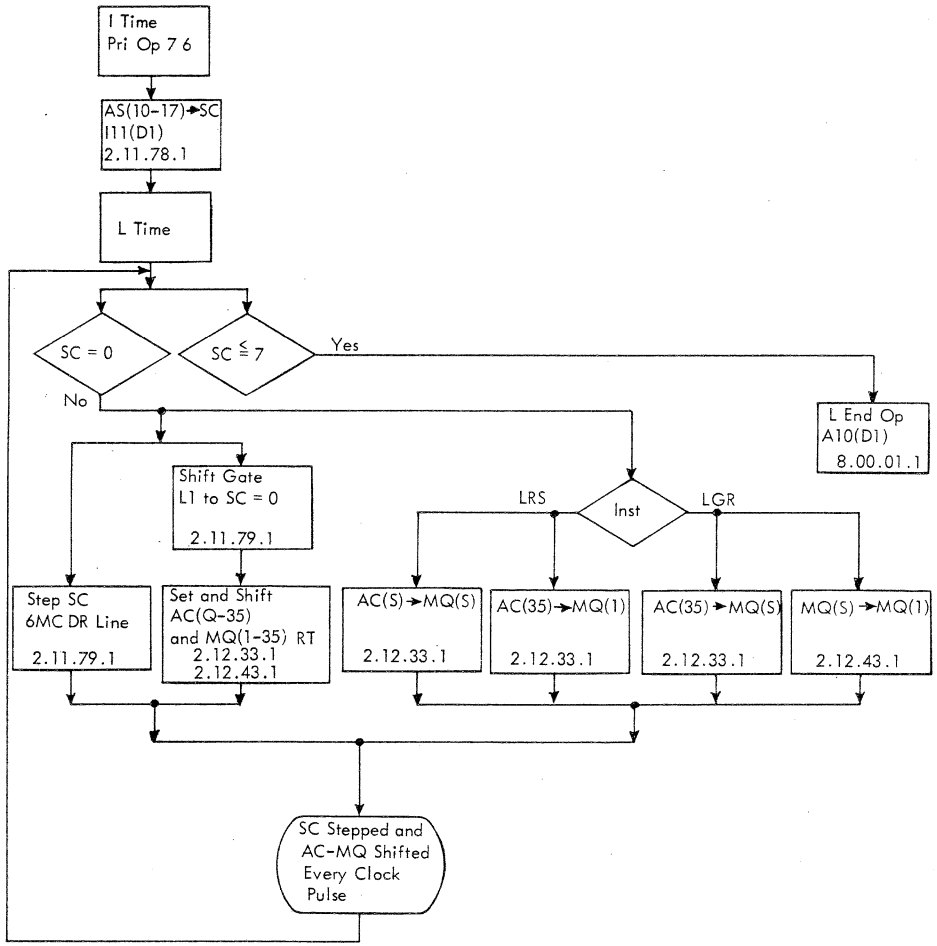


FIGURE 3.5-12. LRS +0765; LGR -0765

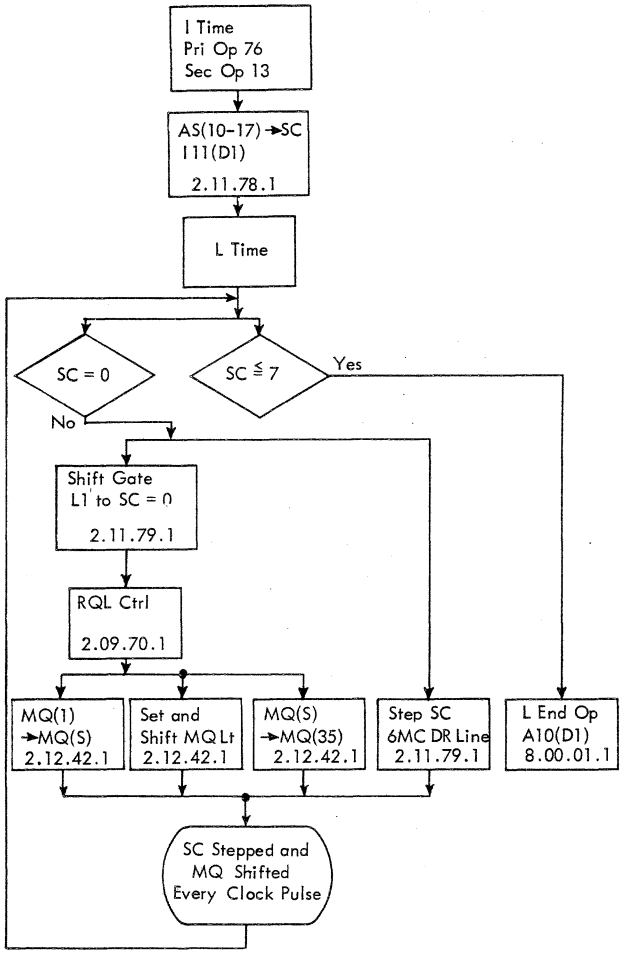


FIGURE 3.5-13. RQL - 0773

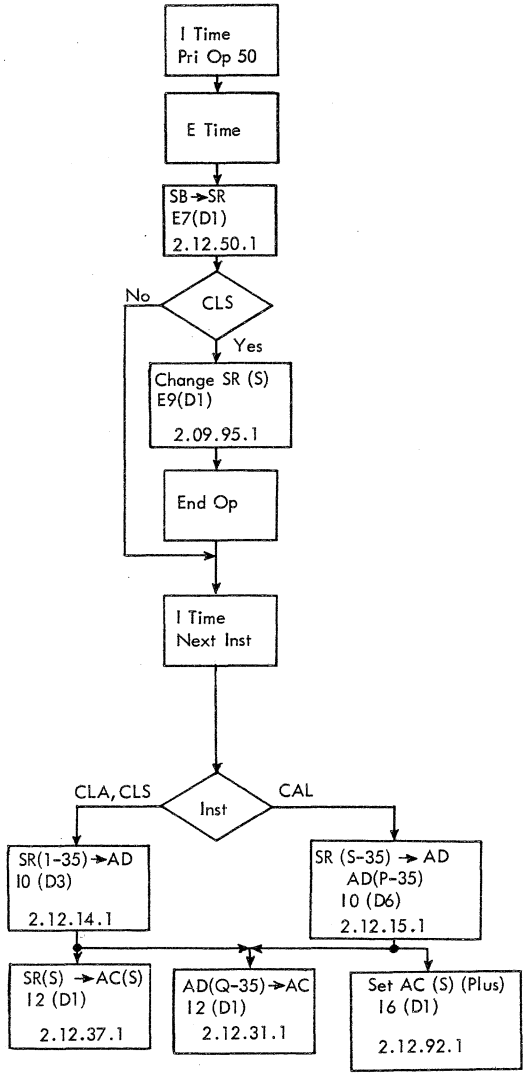


FIGURE 3.5-14. CLA +0500; CLS +0502; CAL - 0500

Clear and Add Logical Word CAL -0500 Figure 3.5-14

This instruction clears the AC(S,Q,P,1-35) and enters the word stored at the location indicated by the address into AC(P,1-35). This instruction operates the same as CLA except for the handling of the S and P positions. The sign of the word in the AC is positive.

Add ADD +0400 Figure 3.5-15

This instruction causes the word, stored at the location indicated by the address, to be added algebraically, to the contents of the AC. The resulting sum or difference replaces the AC factor.

The following rules of addition are used during the execution of the add instruction:

1. Accumulator and storage register signs alike:
   a. Add true accumulator factor to the storage register factor.
   b. The accumulator sign is unchanged.
2. Accumulator and storage register signs unlike:
   a. Add 1's complement of the accumulator factor to the storage register factor.
      (1) If no end-carry results, complement the accumulator factor and leave the accumulator sign unchanged.
      (2) If an end-carry results, add one to the result and change the accumulator sign.

Except for bringing a word from storage to the SR, most of the execution of this instruction is accomplished during the I cycle of the next instruction. The contents of the AC or the 1's complement of the AC and the contents of the SR are added in the adders. Whether to use true AC or complemented AC is determined by the comparison between the AC and SR signs. Complement addition is used to obtain the difference between the contents of the SR and the contents of the AC.

The difference between the SR and AC contents can be a complement number or a true number. The result will be in complement form if the AC is larger than the SR factor. A true number will result if the AC factor is smaller than the SR factor. During the addition a carry out of AD position Q indicates that the AC factor is smaller and no Q carry indicates that the AC factor is larger. To remember the carry, a Q carry trigger is turned on by a carry out of AD position (Q).

If the result of the complement addition is a true number it is one less than it should be because the 1's complement rather than the 2's complement was used in the addition. Therefore, a one is added to the result in the AC to get the correct difference. If the result of the addition is a complement number, it must be re-complemented to get the correct true number. The sign of the result in the AC is set the same as the sign of the largest original factor, as determined by the status of the Q carry.

Example 1
Signs Alike
-7 + (-6) = -13
-0111 SR(7)
-0110 AC(6)
-1101 Result in AC (13)

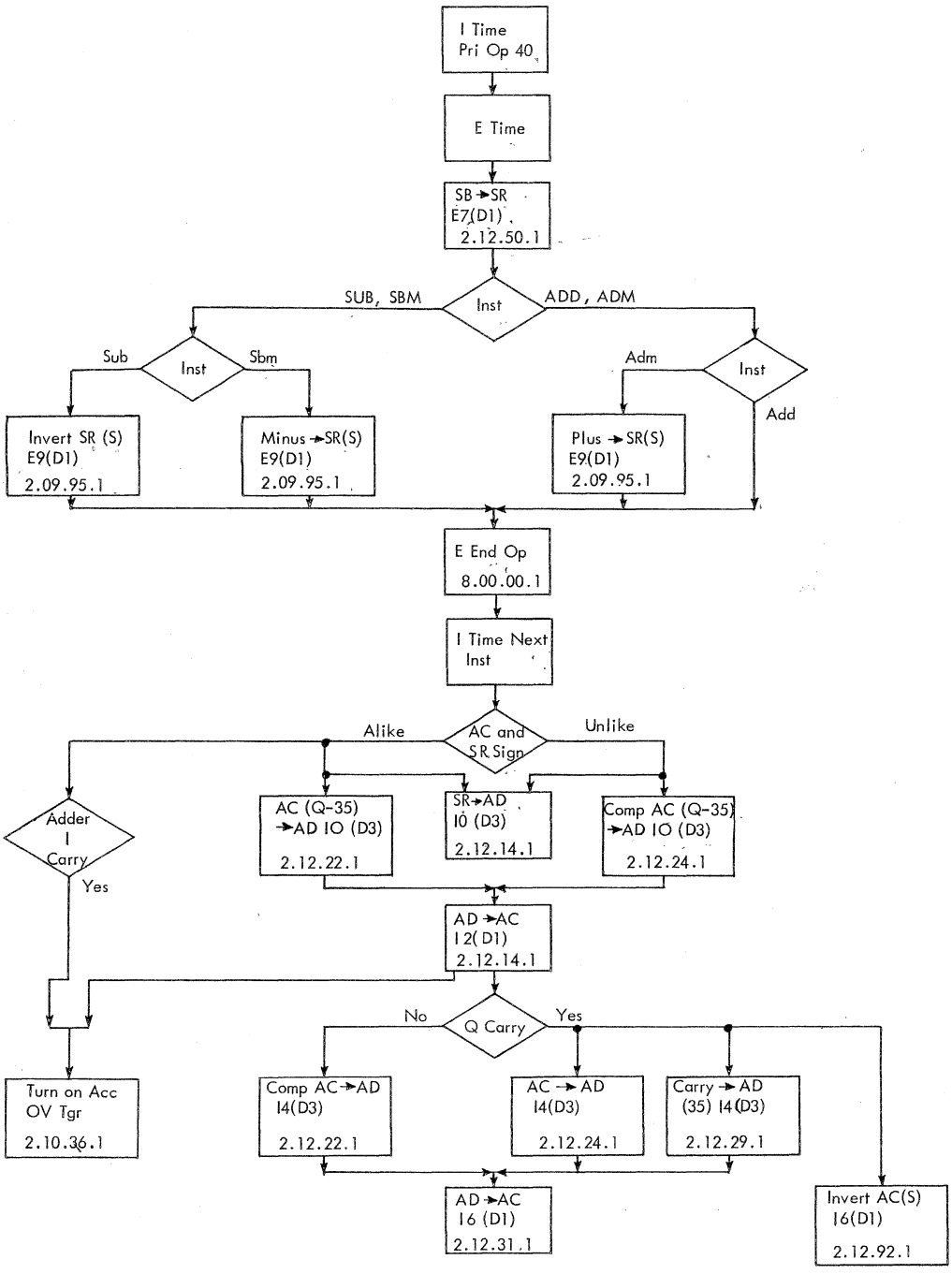


FIGURE 3.5-15. ADD +0400; ADM +0401; SUB +0402; SBM -0400

Example 2

Signs Unlike, AC Smaller

7 + (-6) = +1

```
+0111  SR(7)
-1001  1's comp of AC(6)
-0000  Q carry
    1  Add one
-0001  Result in AC
+0001  Change sign
```

Example 3

Signs Unlike, AC Greater

6 + (-7) = -1

```
+0110  SR(6)
-1000  1's comp of AC(7)
-1110  No Q carry, Result in AC
-0001  Comp AC
```

Add Magnitude ADM +0401 Figure 3.5-15

For this instruction, the word at the storage location indicated by the address is to be considered positive. This positive word is added, algebraically, to the contents of the AC. ADM operates the same as ADD, except that the sign of the word is set positive after it has entered the SR.

Subtract SUB +0402 Figure 3.5-15

This instruction algebraically subtracts the word at the storage location indicated by the address from the contents of the AC. The execution of SUB is similar to ADD, except that the sign of the word in the SR is inverted.

Subtract Magnitude SBM -0400 Figure 3.5-15

For this instruction the sign of the storage word indicated by the address is considered negative. This negative word is added algebraically to the contents of the AC. The execution of SBM is the same as ADD, except that the sign of the SR word is forced negative.

Add and Carry Logical Word ACL +0361 Figure 3.5-16

This instruction adds the 36-bit logical word stored at the location indicated by the address to the contents of the AC(P-35). AC(S) is ignored; AC(Q) is cleared. A carry out of AC(P) is added to position 35 in the adders. The operation of this instruction is the same as ADD, except for the handling of the (S), (Q), and (P) positions.

Multiply MPY +0200 Figure 3.5-17

This instruction multiplies the word stored at the location indicated by the address by the contents of the MQ. The 35 most significant bits of the 70-bit product replace the AC(1-35), and the 35 least significant bits replace the MQ(1-35). Positions (Q) and (P) of the accumulator are cleared, and the signs of the AC and MQ are set to the algebraic sign of the product.

```
00  0110  Multiplicand (SR)
    1011  Multiplier (MQ)
    0110  Partial Product
   0110     "       "
  0000      "       "
 0110       "       "
1000010  Product (AC and MQ)
```

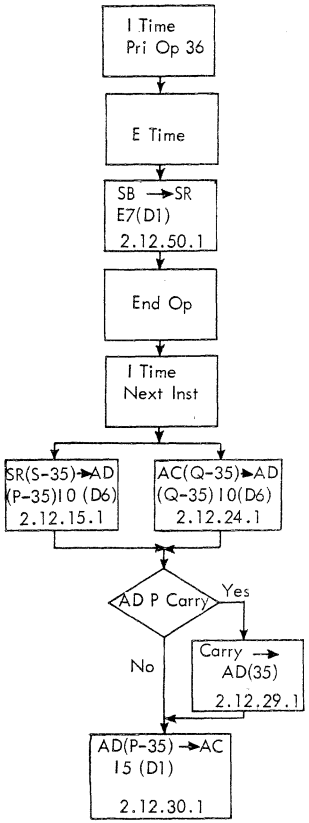


FIGURE 3.5-16. ACL +0361

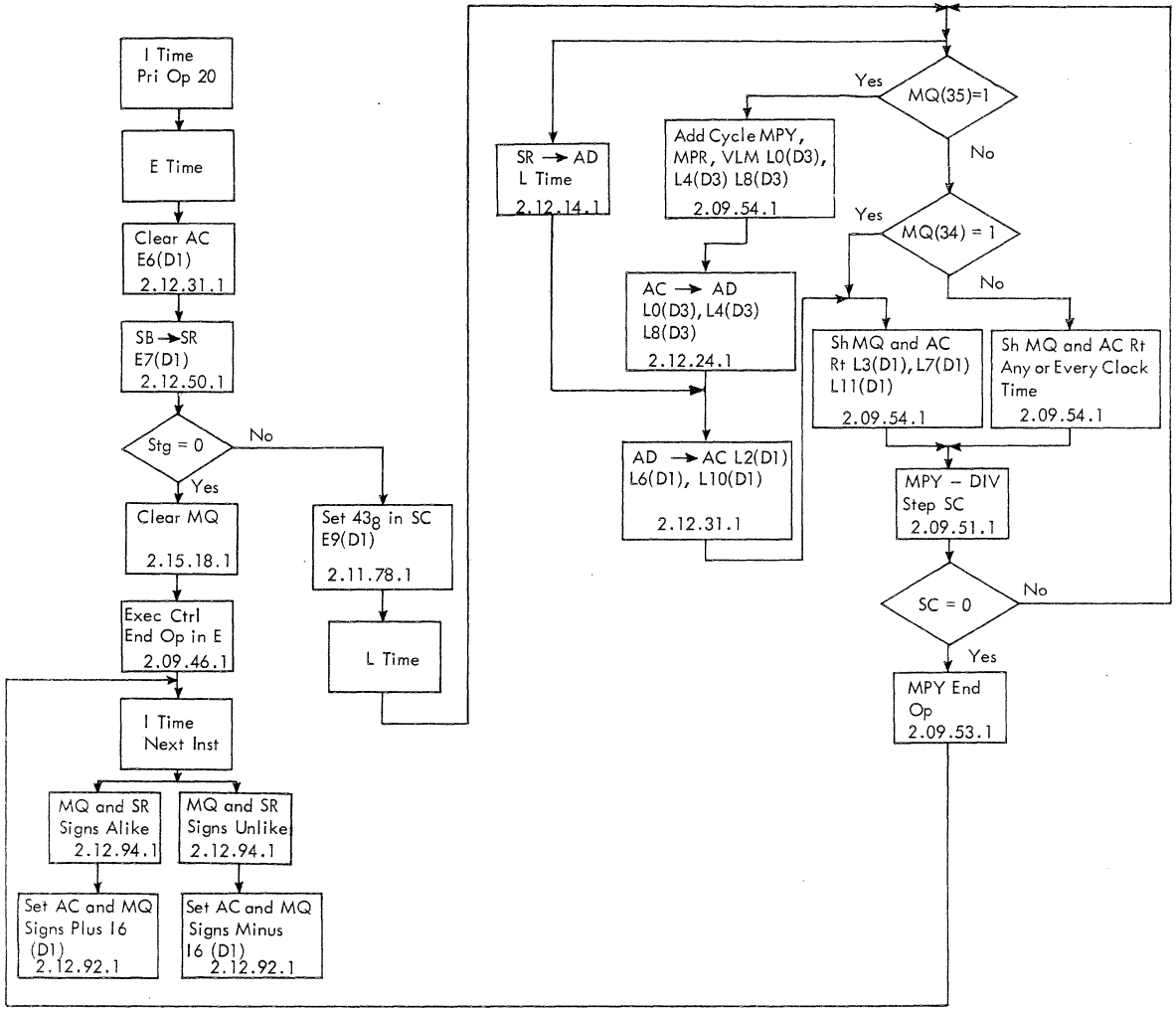


FIGURE 3.5-17. MPY ±0200

In the 7090, multiplication follows this procedure closely. Starting at the right of the multiplier in the MQ, only one position of the multiplier is scanned at a time. In the 7090 there is no means of remembering partial products so they may be added after each position of the multiplier has been used. Therefore, if the multiplier position being scanned is a one, the multiplicand is added to any previous partial product that is in the AC. In the 7090, multiplication is done as follows:

```
   0110   Multiplicand (SR)
   1011   Multiplier (MQ)
   0000   Start with zeros in AC
  +0110   Right MQ position scanned
   0110   Partial Product
  +0110   Next MQ position scanned
  10010   Partial Product
   0000   Next MQ positions scanned
 010010   Partial Product
  +0110   Next MQ position scanned
1000010   Product (AC and MQ)
```

In this example, each time the multiplicand is added to the previous partial product, the multiplicand must be moved to the left to correctly line up the bit positions. In the 7090, the partial product is moved to the right rather than the multiplicand moving to the left. The end result is the same.

Multiplication in the 7090 breaks to two major procedures:

1. Addition of SR and AC contents.
2. Shifting the contents of the AC and MQ right.

Figure 3.5-17 shows a MPY operation in the 7090. During the E cycle, in addition to bringing the multiplicand to the SR, the AC is cleared. This allows the MPY to start with the AC containing zeros. Also during the E cycle the word coming from storage is tested to see if all positions contain a zero. If the multiplicand is zero the product is to be zero. To save machine time, the MQ is cleared and the instruction ends operation. For a non-zero multiplicand, 43_8 (35_{10}) is put into the SC. The SC will be used to indicate when all bits of the MQ have been tested. Also, because the storage word was not zero, the computer is put into L time for multiplication.

Now MQ(35) is tested for a bit. If there is a bit in MQ(35), the SR and AC contents are added and the result is put into the AC. If MQ(35) had no bit, this addition would not have been done because the sum of zero and a number equals the same number. The AC and MQ are shifted right to put the next position of the multiplier in MQ(35) to be tested, and put the partial product in correct alignment with the multiplicand. This type of shift is called a slow-speed shift. High-speed shifting (shifting once for each clock time) will occur if both MQ(34) and MQ(35) contain zeros. This is done to speed the multiplication operation. Each time the AC and MQ are shifted, the shift counter is stepped and, as long as the shift counter does not equal zero, MQ(35) is once more tested and the process starts again. When the SC has been stepped down to zero, the computer signals that multiplication is complete. Because all positions of the multiplier have been scanned and all additions have been completed, MPY then ends operation.

All that remains is to put the algebraic sign of the product in the AC and MQ. This is done during the I cycle of the next instruction.

Note that three additions can be done during each cycle of MPY. The maximum number of shifts during an L cycle can be 12, depending on positions (34) and (35) of the MQ. Maximum number of machine cycles required for MPY is 14; minimum is two.

Multiply and Round MPR -0200 Figure 3.5-18

This instruction executes a multiplication followed by rounding the AC contents. The multiplication is identical to MPY; the rounding is accomplished by adding one to AC (35) if MQ(1) contains a one.

Variable-Length Multiply VLM +0204 Figure 3.5-19

This instruction operates much the same as MPY, except that the number of multiplier bits to be used is specified by a number in the decrement portion of the instruction. The difference between MPY and VLM occurs in the E cycle. A count from the decrement portion of the instruction is put into the SC rather than 43_8. Usually this count will be less than 43_8. Therefore, VLM usually takes less time than MPY because end operation still occurs when the SC equals zero.

Divide or Halt DVH +0220 Figure 3.5-20

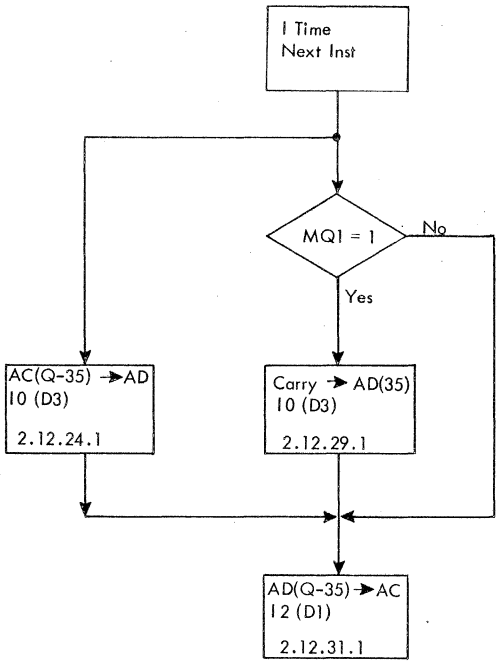
This instruction divides the contents of the AC and MQ, taken together as the dividend, by the word stored at the location specified by the address. A 35-position quotient is developed in the MQ, and the remainder of the dividend is left in the AC. The sign of the MQ is set to the algebraic sign of the quotient, as determined by the SR and AC signs. The sign of the remainder remains the same as the sign of the dividend. The size of the registers restricts the size of the factors to be divided. If the AC portion of the dividend were greater than or equal to the divisor, the quotient would be too large for the MQ. In this case, division cannot take place, and the computer is stopped with the divide check indicator on.

The following demonstrates binary division:

```
          0110      Quotient (MQ)
1011 ) 1000010      Dividend (AC and MQ)
       1011
       01011
        1011
        00000
         0000
         0000       Remainder (AC)
```

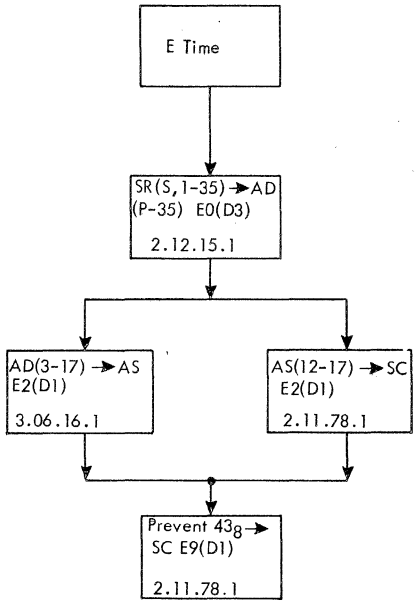
Note that the divisor will go once or not at all into the high order positions of the dividend. Therefore, it is only necessary to determine if the divisor is equal to or smaller than these positions of the dividend. If the divisor is equal to or smaller than the selected positions of the dividend, a one is put in the quotient and the divisor is subtracted from that portion of the dividend. If the divisor is larger than the selected portion of the dividend, a zero remains in the quotient. Another position of the dividend is now taken into account, and the procedure starts again. This continues until all positions in the dividend have been used.

In the 7090, the SR and AC are complement added to determine if a reduction of the high order positions of the dividend is possible. If the reduction is possible, these



Operates the Same as MPY, Except that the Above is also Done During the I Cycle after MPY.

FIGURE 3.5-18. MPR - 0200



Operates the Same as MPY, Except that the Above is also Done During the E Cycle.

FIGURE 3.5-19. VLM + 0204

positions of the dividend are reduced by the amount of the divisor and the difference is put into the AC. If a reduction is not possible the AC remains the same. A successful reduction causes a one to be put into MQ(35). After the reduction time, the AC and MQ are shifted left to bring the next position of the dividend into the AC and another reduction is tried. This operation continues until all positions of the MQ portion of the dividend have been moved to the AC.

Figure 3.5-20 shows the sequence of a DVH operation. Because subtraction in the 7090 is accomplished through complement addition, the AC contents are put into complement form early in the E cycle. This complement and the contents of the SR are added to determine if the quotient will be small enough for the MQ register. A Q carry indicates that division is possible and a no Q carry indicates that division is not possible. Also, during the E cycle, 43_8 is put into the SC and this will be used to indicate when all dividend positions have been used.

When there is no Q carry as a result of the E cycle test, the divide check trigger is turned on and the computer is signalled to divide check end operation. An L cycle occurs before actual end operation because the divide check end operation signal comes too late in the E cycle. During the I cycle of the next instruction, the master stop trigger is turned on and this causes B cycle interrupt to be activated. B cycle interrupt prevents I, E, and L cycles.

A Q carry as the result of the E cycle test allows normal divide operation to take place. Toward the end of the E cycle the AC and MQ are shifted left and the SC is stepped. This brings the next position of the dividend into the MQ. Note that while shifting from MQ(1) to AC(35), the information is complemented because a complement number is used in the AC.

The computer then goes into L cycles. A reduction is attempted and a 1 is put into MQ(35) if the reduction is successful. Again there is a left shift of the AC and MQ and the SC is stepped. This attempted reduction and shifting procedure continues until the SC equals zero. When the SC=0, divide end operation is actuated.

During the I cycle of the next instruction the AC is again complemented to make the remainder a true number. The sign of the MQ is set to the algebraic sign of the quotient.

Divide or Proceed DVP +0221 Figure 3.5-20

The execution of this instruction is identical to DVH except that the computer is not stopped for the divide check violation, but proceeds to the next instruction. The divide check trigger is not allowed to turn on the master stop trigger during the I cycle of the next instruction.

Variable-Length Divide or Halt VDH +0224 Figure 3.5-20

This instruction operates the same as DVH, except that the number of reductions to be taken is specified by the count in positions (12-17) of the instruction. The number of positions in the quotient is equal to the count and will be contained in the low-order positions of the MQ. The count should be restricted to a number between 0 and 43_8. A zero count ends operation in E time and prevents the shift at the end of the E cycle. A count of 43_8 will give the same result as DVH. A count greater than 43_8 causes part of the quotient to be shifted into the AC, where it can be altered.

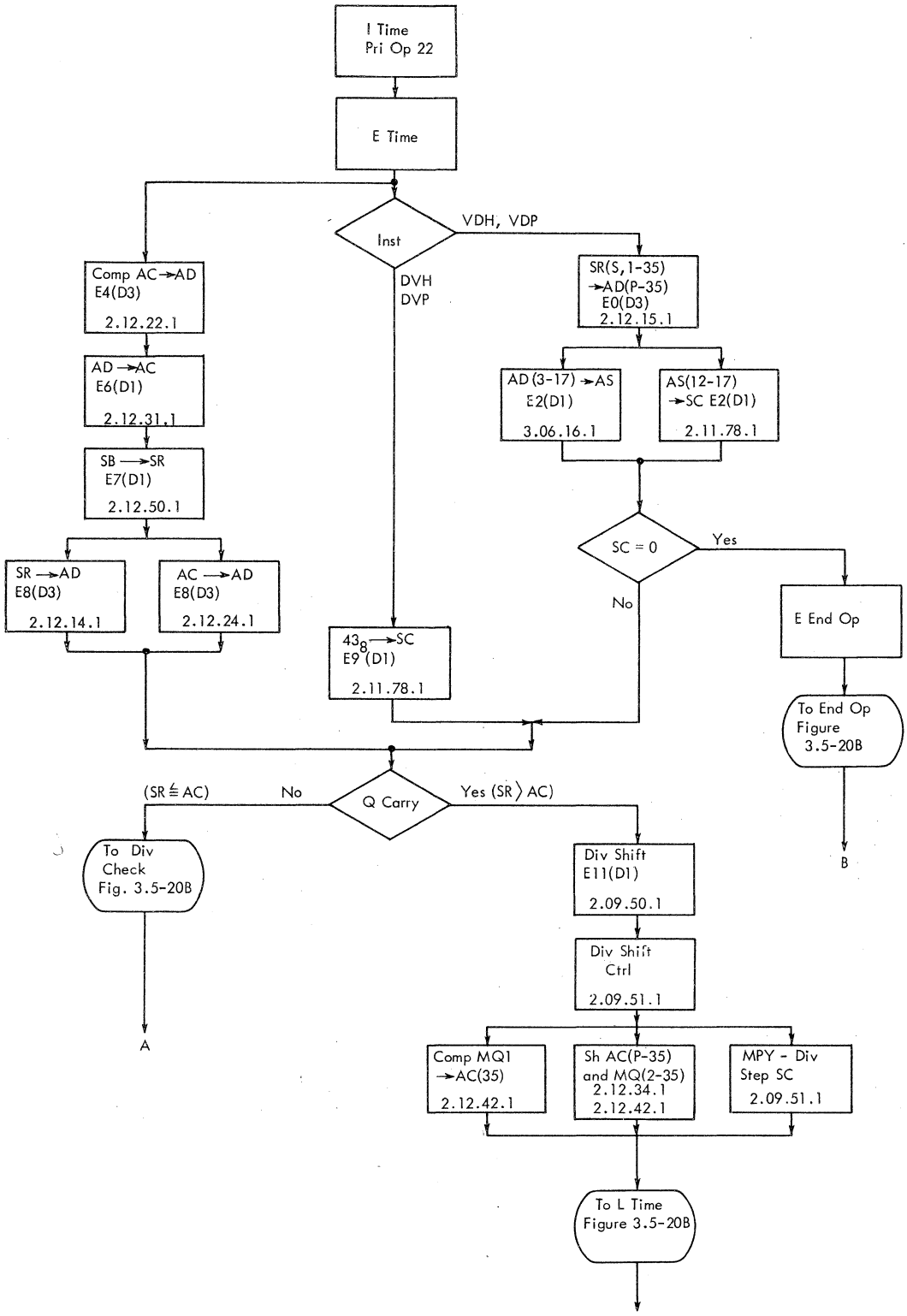


FIGURE 3.5-20A. DVH +0220; DVP +0221; VDH +0224; VDP + 0225

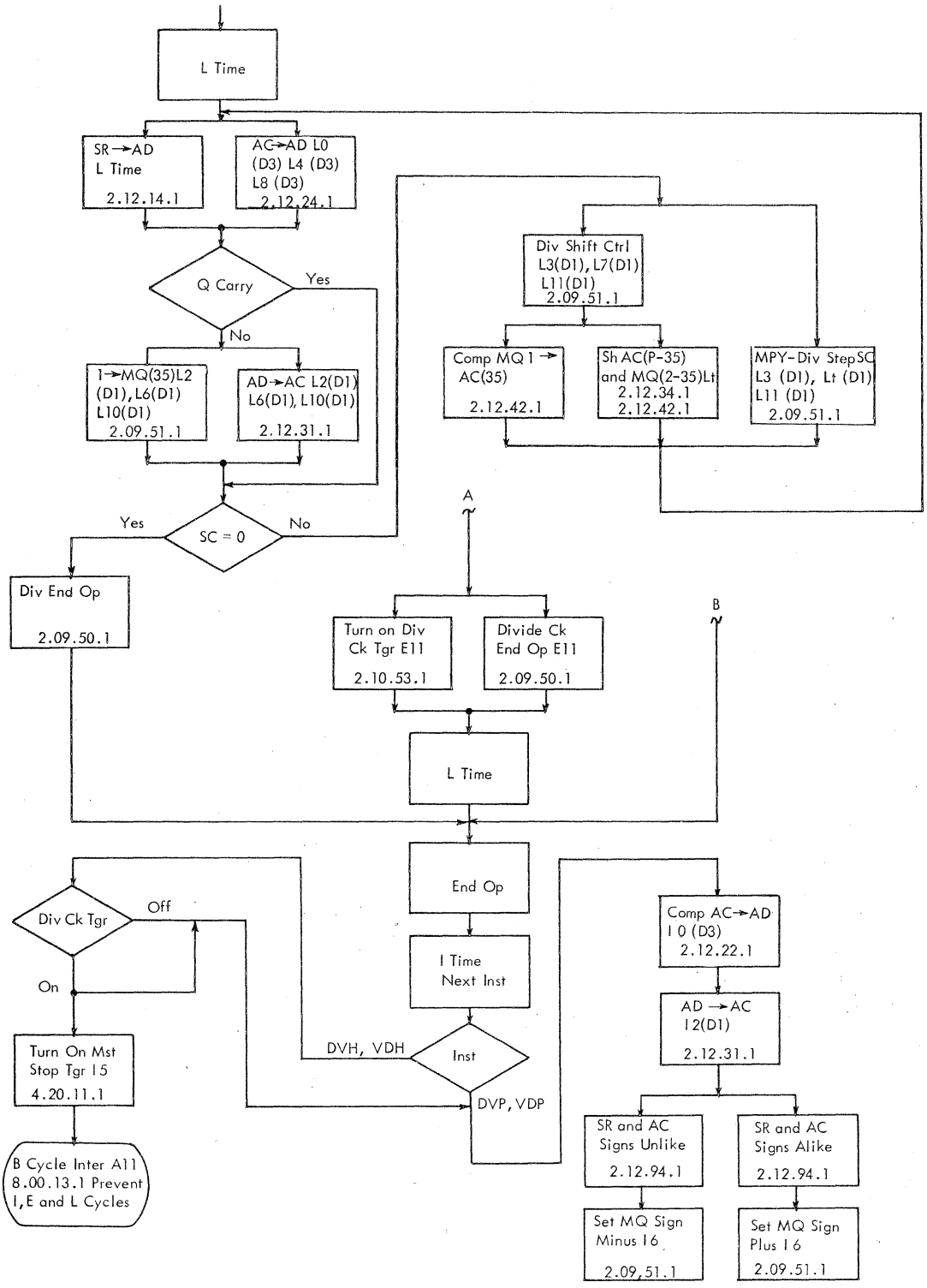


FIGURE 3.5-20B. DVH + 0220; DVP + 0221; VDH + 0224; VDP + 0225

Variable-Length Divide or Proceed	VDP +0225	Figure 3.5-20

The execution of this instruction is the same as VDH, except that the computer will not stop for a divide check, but will proceed to the next instruction.

Round	RND +0760...0010	Figure 3.5-21

This instruction examines the contents of MQ(1) and if it contains a one, the magnitude of the AC is increased by one. If MQ(1) contains a zero, the AC is unchanged. The MQ is not changed in either case. AC overflow is possible. This is a primary operation 76 instruction. The contents of the AC are sent to the AD and, if MQ(1) contains a one, a one is sent to AD(35) and the output of the AD replaces the contents of the AC.

Clear Magnitude	CLM +0760...0000	Figure 3.5-22

This instruction puts zeros in AC(Q-35). CLM is a primary operation 76 instruction. The operation is accomplished by gating the AD to the AC, with nothing in the adders.

Complement Magnitude	COM +0760...0006	Figure 3.5-22

The contents of the AC(Q-35) are complemented. Positions containing ones are changed to zeros and positions containing zeros are changed to ones. This instruction is executed by complementing the AC to the AD and replacing the contents of the AC with this complement.

3.5.04 Floating-Point Arithmetic Instructions

The range of numbers anticipated during a calculation may be extremely large, extremely small or, in some cases, unpredictable. Such situations make fixed-point arithmetic difficult to work with for two reasons;

1. The size of the number is limited by the size of the register (35 binary bits or 12 decimal digits).
2. The programmer must keep track of the point in all numbers throughout the calculation.

To meet the needs of large numbers and to automatically keep track of the point, an alternative set of arithmetic instructions, called floating-point arithmetic instructions, are available.

Floating-point arithmetic is merely arithmetic dealing with numbers in exponential form. The numbers 5, 6×10^{3} or 56000×10^{-4} have a familiar form. The numbers are made of three parts; a fraction (5.6 or 56000), an exponent (3 or -4), and a base (10).

Floating-point numbers in binary are similar to decimal floating-point numbers. The major difference is the base. Numbers in the 7090 use 2 as a base, because it is a binary computer. The other difference is one of terms. Instead of a decimal point, we will call it a binary point.

The following chart gives a comparison of fixed-point binary numbers and floating-point binary.

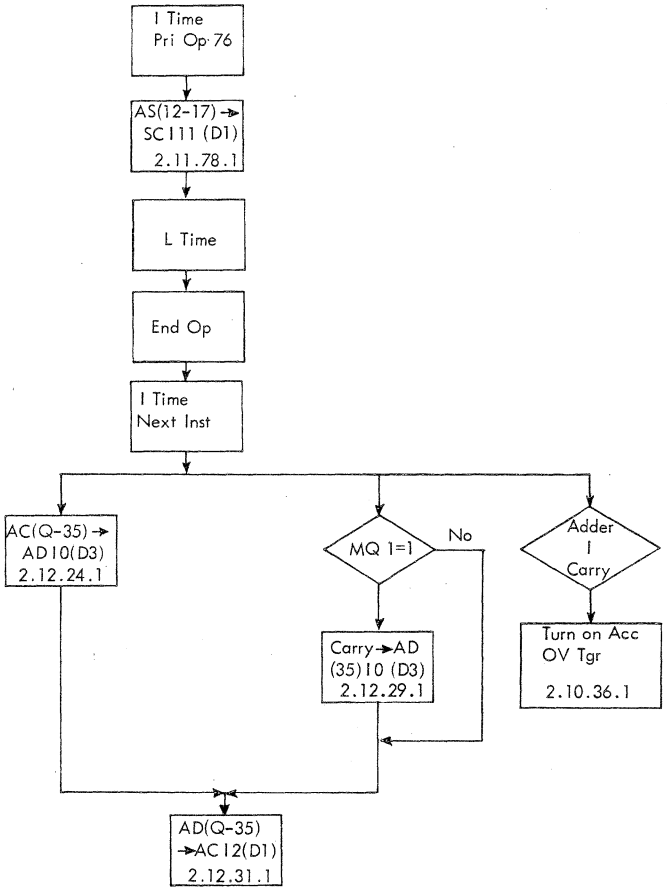


FIGURE 3.5-21. RND +0760...0010

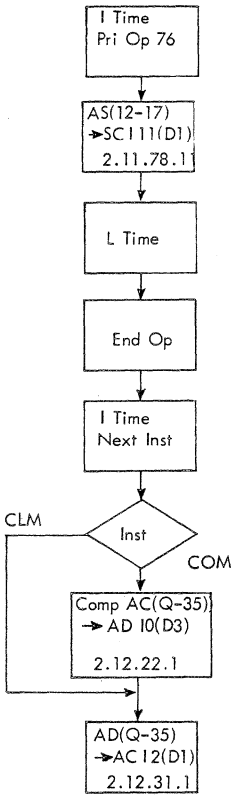


FIGURE 3.5-22. CLM +0760...0000; COM +0760...0006

Fixed-Point Binary	Floating Point
(4) 000100	$.1 \times 2^{011}$
(11) 001001	$.1001 \times 2^{100}$

Because the 7090 works in binary, all floating-point numbers will be to the base 2. Therefore, to represent a floating-point number in the computer, there is no need to carry the base along with the number. This cuts our need to representing the fraction and the exponent. The exponent is represented in positions (1-8) of the word and is now called the characteristic. The fraction is contained in positions (9-35). The binary point is to the left of the 9 bit. The sign position is used to sign the fraction. Word layout takes this format:

S 1 ---------- 8 .9----------35

Characteristic Fraction

The value of the number in the characteristic field signifies the exponent and its sign. The characteristic is derived by adding 200_8 to the exponent. If the characteristic is 200_8 the exponent is zero. If the number is 201 to 377, the exponent is positive. If it is 0 to 177, the exponent is negative. The following chart gives examples of exponential numbers and their floating point representation:

Exponential		Floating Point	
Binary	S	1 - 8	9 - 35
$+.1 \times 2^{011}$	+	10000011	10000----0
$-.01 \times 2^{001}$	-	10000001	0100-----0
$+.1 \times 2^{-011}$	+	01111101	1000-----0

Normal and Unnormal Forms

A floating-point number is said to be in normal form when the digit immediately to the right of the point is a significant bit (1). If the number is a zero, it is said to be in unnormal form. The exception to this rule is a normal zero: a normal zero is a floating-point number whose characteristic and fraction are both zero.

To go along with the two type of numbers, the instructions are also divided into two categories, normal and unnormal. The difference in computer operation is that the normal instructions always attempt to produce a normal answer and the unnormal instructions do not.

Arithmetic of Floating Point

Addition of floating-point numbers is done by adding the fractions of floating-point numbers which have equal characteristics. The characteristics are set equal preceding the addition by placing the number with the smallest characteristic in the AC. The fraction is then shifted right, and for each right shift one is added to the characteristic. When the characteristics of the AC equals the characteristic of the SR, shifting stops, and the fractions of the AC and SR are added. Bit shifted out of AC(35) enter MQ(9). The sum appears in the AC and forms the most significant part of the answer. The least significant part is the bits that were shifted into the MQ. The MQ characteristic is set 27_{10} less than the AC characteristic to complete an unnormalized floating

add. If it were a normalizing instruction, a check would be made to see if a 1 were in AC position nine. If AC(9) does contain a 1, the operation would be complete; if not, the AC would shift left until a one did appear in position nine. Shifting increases the number, so to keep it the same, the characteristic is reduced by the number of left shifts taken. Floating-point subtraction works the same except that the fractions are subtracted.

Floating-point divide is accomplished by dividing the fraction of the dividend by the fraction of the divisor and subtracting the characteristics. During the subtraction of the characteristics, the 200_8 that is added to all exponents is lost. Therefore, before the answer is final, 200_8 must be added to the quotient characteristic.

Floating multiply is accomplished by multiplying the fraction in the SR by the fraction in the MQ. The exponents in multiply are added, so in a floating multiply, the computer adds the characteristics. Because 200_8 had been added to each exponent originally, the characteristic is increased by 400_8 after the addition. Before the answer is final, 200_8 must be subtracted from the characteristic. The most significant part of the product is in the AC and the least significant part in the MQ.

Sign control is as follows:

Multiplication and Division	Signs of factors alike; answer plus Signs of factors unlike; answer minus
Addition and Subtraction	Answer always has sign of the largest factor

Floating-Point Tally Counter

The floating-point add, multiply, and divide type instructions require an I, an E, and several L cycles. Execution of the instruction is accomplished during the L cycles. To differentiate between the different types of L cycles, there is a tally counter. This is a five stage counter whose output is added with L time to direct the various phases in the execution of the instruction. When the required operations are complete for first step L time, the tally counter is stepped to produce second step L time. This causes the computer to go into the next phase in the execution of the instruction. The operation continues to the next step, and so on. All five steps of the tally counter are not used for every instruction; each instruction uses as many as it requires.

Floating Add FAD +0300 Figure 3.5-23

This instruction adds, algebraically, the floating-point number stored at the location indicated by the address and the floating-point number contained in the accumulator. The most significant portion of the result appears as a normalized floating-point number in the accumulator. The least significant portion of the result appears in the MQ as a floating-point number with a characteristic 27_{10} less than the characteristic of the number in the accumulator. The signs of the AC and MQ are set to the sign of the larger factor. If both the resulting MQ and AC fractions are zero, the registers will be reset to contain normal zeros with the signs corresponding to the original factor having the smaller characteristic. If the characteristics were equal, the resulting signs will correspond to the original AC sign.

The result in the AC is always normalized, whether the original factors were normal or not. No attempt is made to normalize the MQ.

First Step L Time. As first step L time is entered, one of the floating-point words to be added is in the AC and the other is in the SR. The objectives of 1st step L time are to:

1. Put the word with the smallest characteristic in the AC.

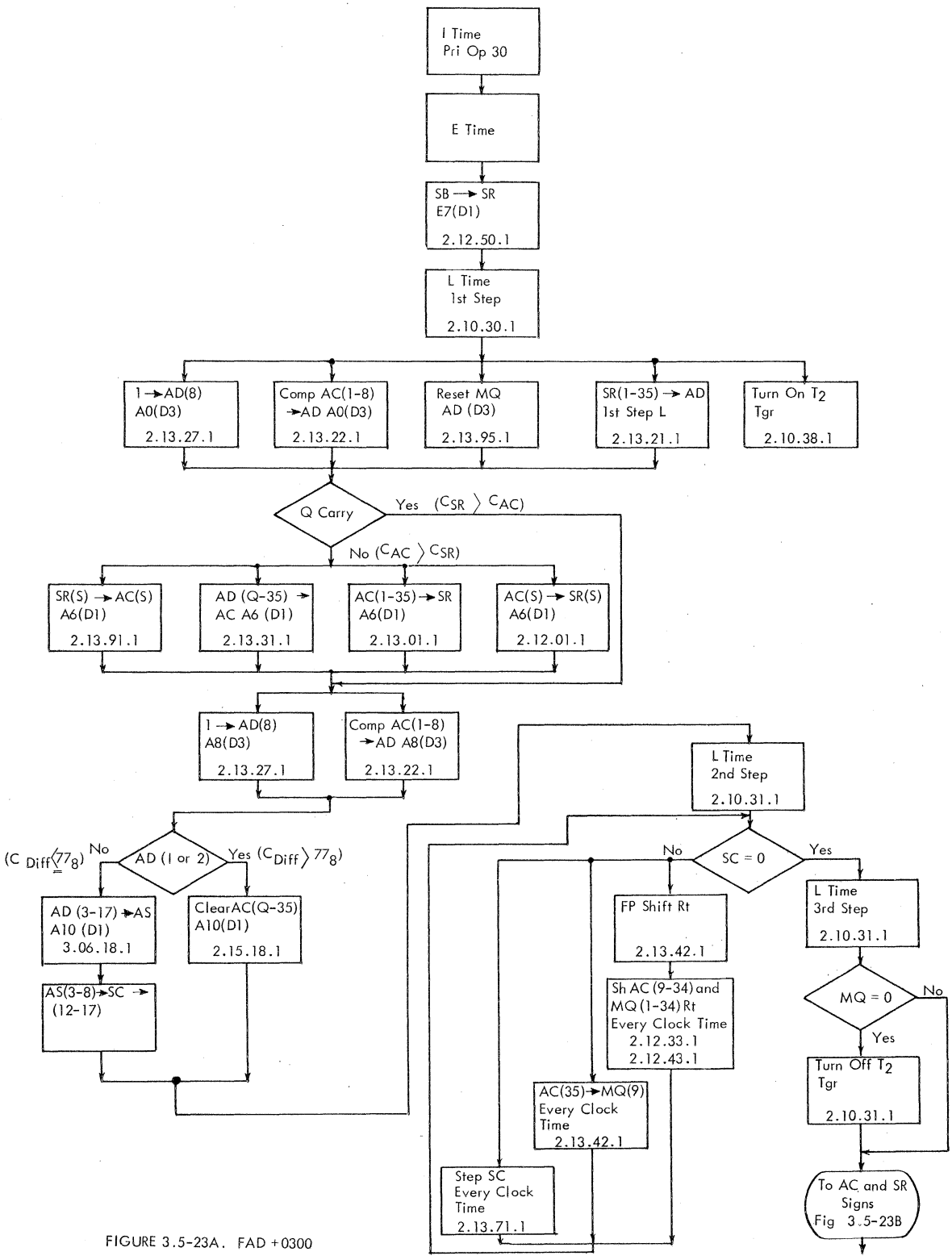


FIGURE 3.5-23A. FAD +0300

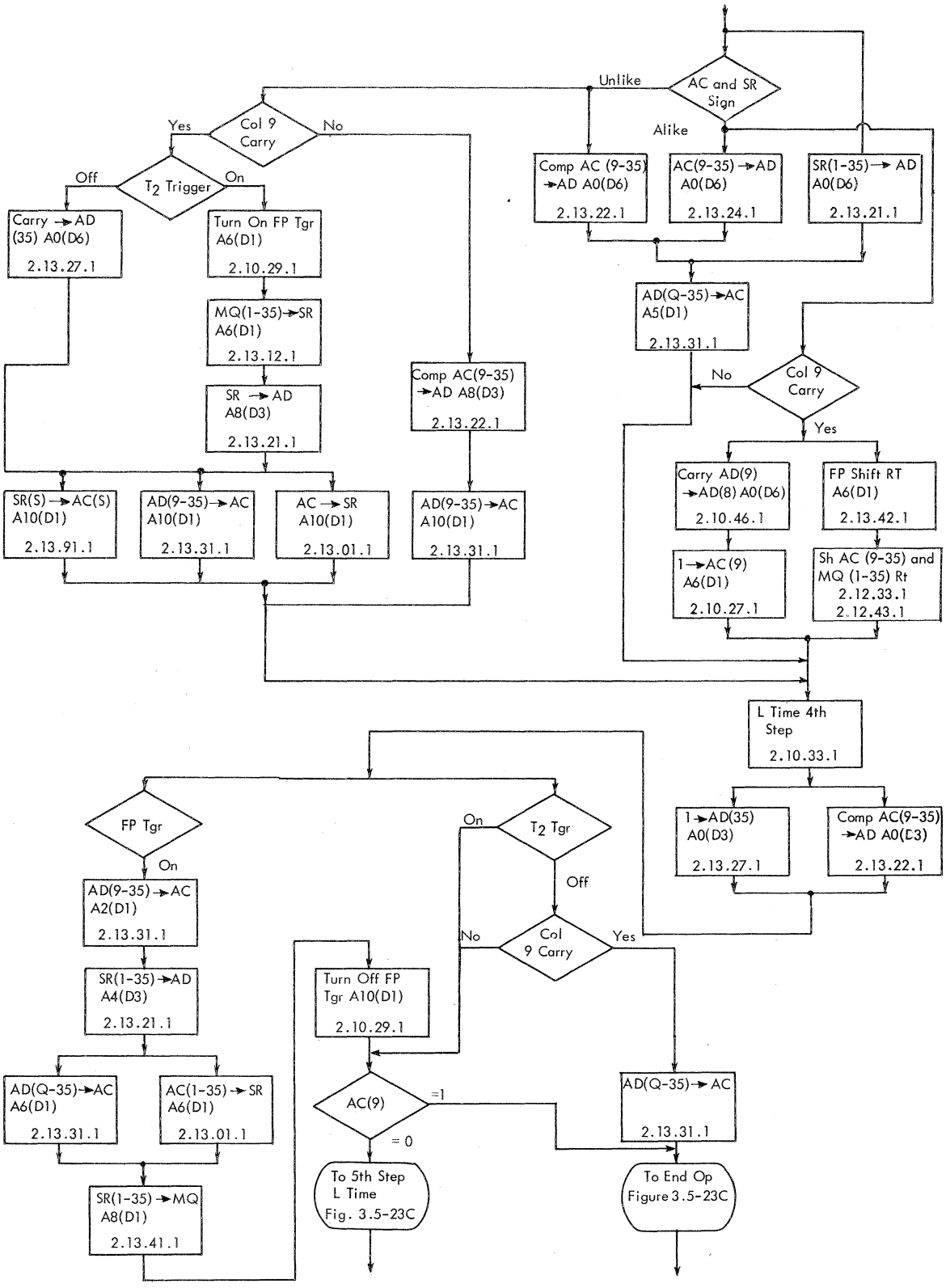


FIGURE 3.5-23B. FAD +0300

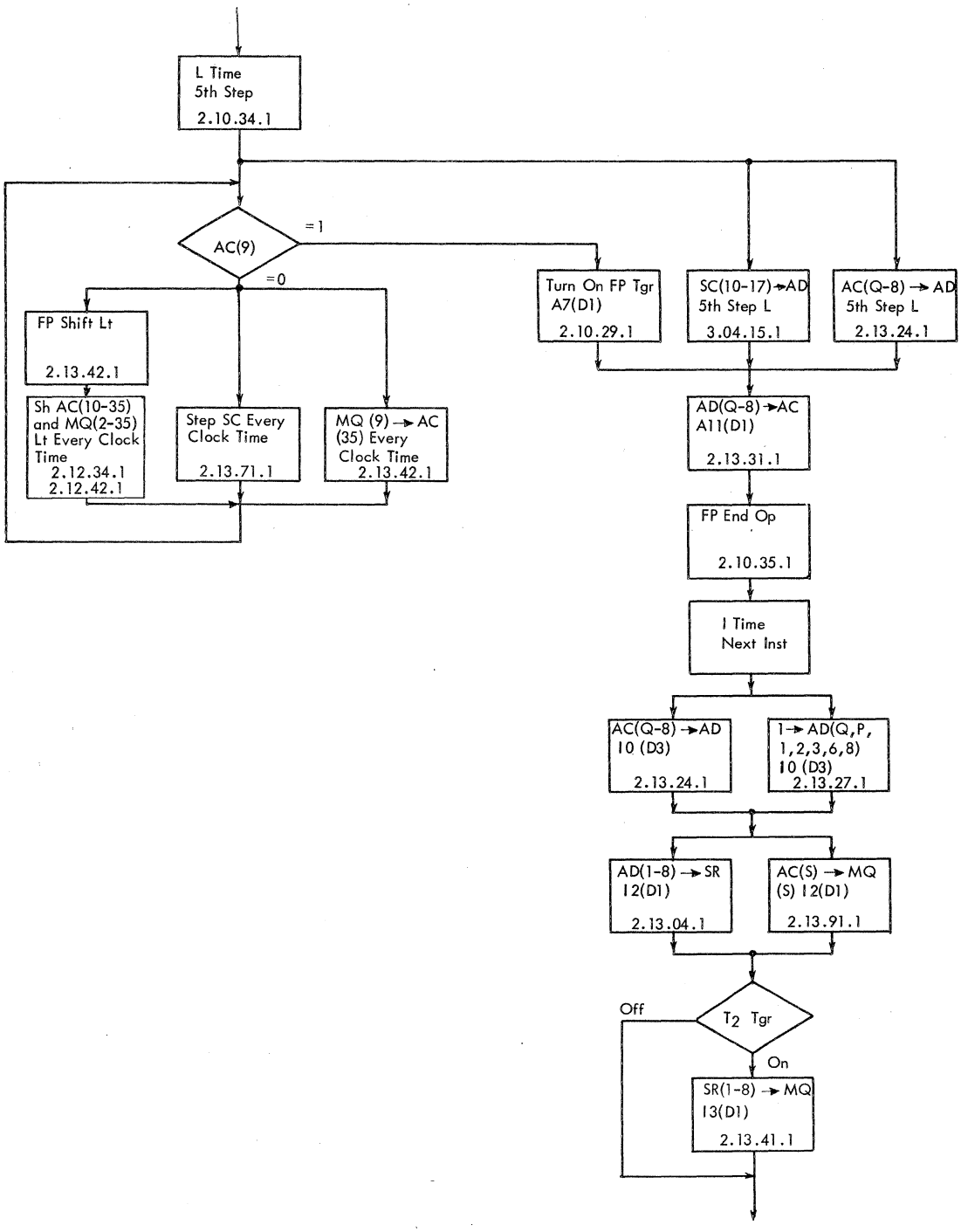


FIGURE 3.5-23C. FAD +0300

2. Determine the difference between AC and SR characteristics.
   a. If the difference is less than 100_8, put the difference in the SC.
   b. If the difference is greater than 77_8, clear the AC.

To determine which characteristic is smaller, the 2's complement of the AC characteristic is added to the SR characteristic. Also, during this time the MQ is reset and T2 trigger is turned on. These are the initial starting conditions for T2 and the MQ. Trigger T2 will be used later, during the execution of the instruction, if the MQ contains a zero or not.

A Q carry, because of the complement addition of the AC characteristic to the SR characteristic, means that the SR characteristic is larger. No Q carry means that the AC characteristic is larger. Therefore, the word in the SR is moved to the AC and the word in the AC is moved to the storage register.

Again, the 2's complement of the AC characteristic is added to the SR characteristic to determine the exact difference between the characteristics. A characteristic difference of less than 100_8 causes the difference to be put into the SC and a difference greater than 77_8 causes the AC to be cleared. The AC is cleared for a difference greater than 77_8 because, due to this difference, the information eventually would be shifted out of the AC and MQ anyway. Shifting the MQ and AC fractions right makes possible setting the characteristics of the SR to the characteristic of the AC. It takes 70_8 shifts to move a bit from AC fraction position (9) out through MQ fraction position (35). The characteristic difference is checked for 77_8 rather than 70_8 because this is easier to do in the computer. The AC is cleared rather than let shifting take place because machine time is saved.

Second Step L Time. Second step L time is used to shift the AC and MQ fractions right a number of places equal to the difference between the AC and MQ characteristics. The principle here is the same as moving the point to the left and increasing the exponent by one for each position that the point is moved. The changing of the characteristic is done later. However, the fraction is adjusted for this change during second step. The AC and MQ fractions are shifted right until the SC is stepped down to zero.

Third Step L Time. Third step L time occurs when the SC equals 0. The objectives of the third step are to:

1. Set the AC characteristic equal to the SR characteristic.
2. Add, algebraically, the AC fraction to the SR fraction.
3. Make AC sign equal to the algebraic sign of the result of the addition.

The AC characteristic is set equal to the SR characteristic as a result of the addition because the AC(9-35) or the 1's complement of AC(9-35) is sent to the adders with SR (1-35). Therefore, when AD(Q-35) are gated back to the AC, the correct characteristic is set into the AC along with the result of the addition.

When binary fractions with like signs are added, there is a possibility of a carry out of the high order fraction position. This is the same as a carry across a decimal point. The 7090 has no register position to hold this carry (1), therefore, the fraction and characteristic of the sum must be adjusted. This is done by moving the AC and MQ fractions one position to the right, putting the one in the high order position of the AC fraction, and increasing the characteristic by one. The characteristic is automatically increased by one during the addition because the carry out of AD(9) goes to the AD(8).

When signs are unlike, a complement addition occurs. A column 9 carry resulting from this addition means the AC fraction was smaller than the SR fraction and the result placed in the AC is a true number. Assuming the MQ to be zero, this true number is 1 less than the sum, because the 1's of the AC fraction were used for the addition. Therefore, a one is added to the AC fraction. If the MQ contains something other than zero, the result of the addition is 1 minus MQ fraction less than it should be. Therefore, one is not added to the AC fraction, but the MQ is subtracted from this one. This is done by, in effect, complementing the MQ. Because no provisions are in the circuits to complement the MQ, the MQ contents are moved to the AC through the SR. The result of the addition is moved to the SR when the MQ contents are brought into the AC.

Fourth Step L Time. Fourth step L time is used to:

1. Complement the AC fraction.
   a. AC contains original MQ fraction if the MQ was not zero.
   b. If original MQ fraction was zero, it was not moved to the AC; therefore, this complementing is used to test the AC fraction for a zero result of the addition.
2. If the MQ and AC are zero, clear the AC to give a zero characteristic.
3. End operation if normalizing is not needed.
   a. If AC and MQ are zero.
   b. If AC(9) contains a one.

After complementing the AC fraction with the FP trigger on (MQ was not zero), the complement is sent to the MQ and the result of the addition is moved from the SR back to the AC. Again available circuits must be used to get the complement to the MQ. Therefore, the AC is gated to the SR and the SR is gated to the MQ.

Fifth Step L Time. Fifth step L time is used for normalizing the result and ending operation. Normalizing is done by (1) shifting the AC and MQ fractions left until AC(9) receives a one, (2) counting the number of shifts, and (3) subtracting the number of shifts from the characteristic. The SC is used to count the number of shifts. Since the SC is a count down counter which starts from zero, the 2's complement of the number of shifts will be in the SC when shifting is stopped. Adding this complement to the characteristic of the AC gives the correct adjusted characteristic.

I Time. I time of the next instruction sets the MQ(S) and characteristic. The AC(S) is gated to the MQ(S) so both signs are the same.

For an MQ fraction that is not zero, the characteristic of the MQ is set $27_{10}(33_8)$ less than the AC characteristic. This is done by adding the 2's complement of 33_8 to the AC characteristic and putting the result into the MQ characteristic.

Unnormalized Floating Add UFA -0300

This instruction algebraically adds two floating-point numbers in the same manner as FAD. The result, however, is not normalized. The sequence of operation for UFA is the same as FAD with these exceptions:

1. For like SR and AC signs, UFA ends operation at the end of the third step because a normalizing step is not needed.
2. UFA ends operation at the end of the third step if AC and SR signs are unlike and there is no column 9 carry as a result of the addition. This is because the correct unnormalized result is in the AC and MQ at the end of the 3rd step.

Floating Add Magnitude FAM +0304

This instruction algebraically adds the magnitude of the floating-point number stored at the location designated by the address to the floating-point number in the accumulator. The sequence of operations is identical to that of FAD except that the SR sign is set positive during the E cycle. This control line is on Systems 2.09.95.1.

Unnormalized Add Magnitude UAM -0304

This instruction operates the same as FAD, with the exceptions pointed out in UFA and FAM.

Floating Subtract FSB +0302

This instruction algebraically subtracts the floating point number stored at the location indicated by the address from the floating-point number in the accumulator. FSB operates the same as FAD, except that the sign of the word in the SR is inverted during the E cycle. This is shown on Systems 2.09.95.1.

Unnormalized Floating Subtract UFS -0302

This instruction algebraically subtracts two floating-point numbers without normalizing the result. Execution is the same as FAD with the exceptions noted in UFA and FSB.

Floating Subtract Magnitude FSM +0306

This instruction algebraically subtracts the magnitude of the floating-point number stored at the location indicated by the address from the floating-point number in the accumulator. Execution of this instruction is the same as FAD, except that the sign of the word in the SR is forced minus during the E cycle. This is shown on Systems 2.09.95.1.

Unnormalized Subtract Magnitude USM -0306

This instruction operates the same as FSM, with the exceptions explained under UFA.

Floating Multiply FMP +0260 Figure 3.5-24

This instruction multiplies the floating-point number stored at the location designated by the address by the floating-point number stored in the MQ. The product appears as two floating-point numbers: the most significant part in the accumulator, and the least significant part in the MQ. The signs of both registers are set to the algebraic sign of the product. If the multiplicand is zero, the product will be two normal zeros with proper algebraic signs. If the AC fraction is zero, the AC is reset to a normal zero, and no characteristic is assigned to the MQ. If the AC fraction is not zero, the MQ is assigned a characteristic 27_{10} less then the AC characteristic.

Remember that when multiplying numbers using exponents, the fractions are multiplied and the exponents are added. Multiplying the fractions is done much the same as in a MPY operation, by shifting and adding. Adding the characteristic is not enough

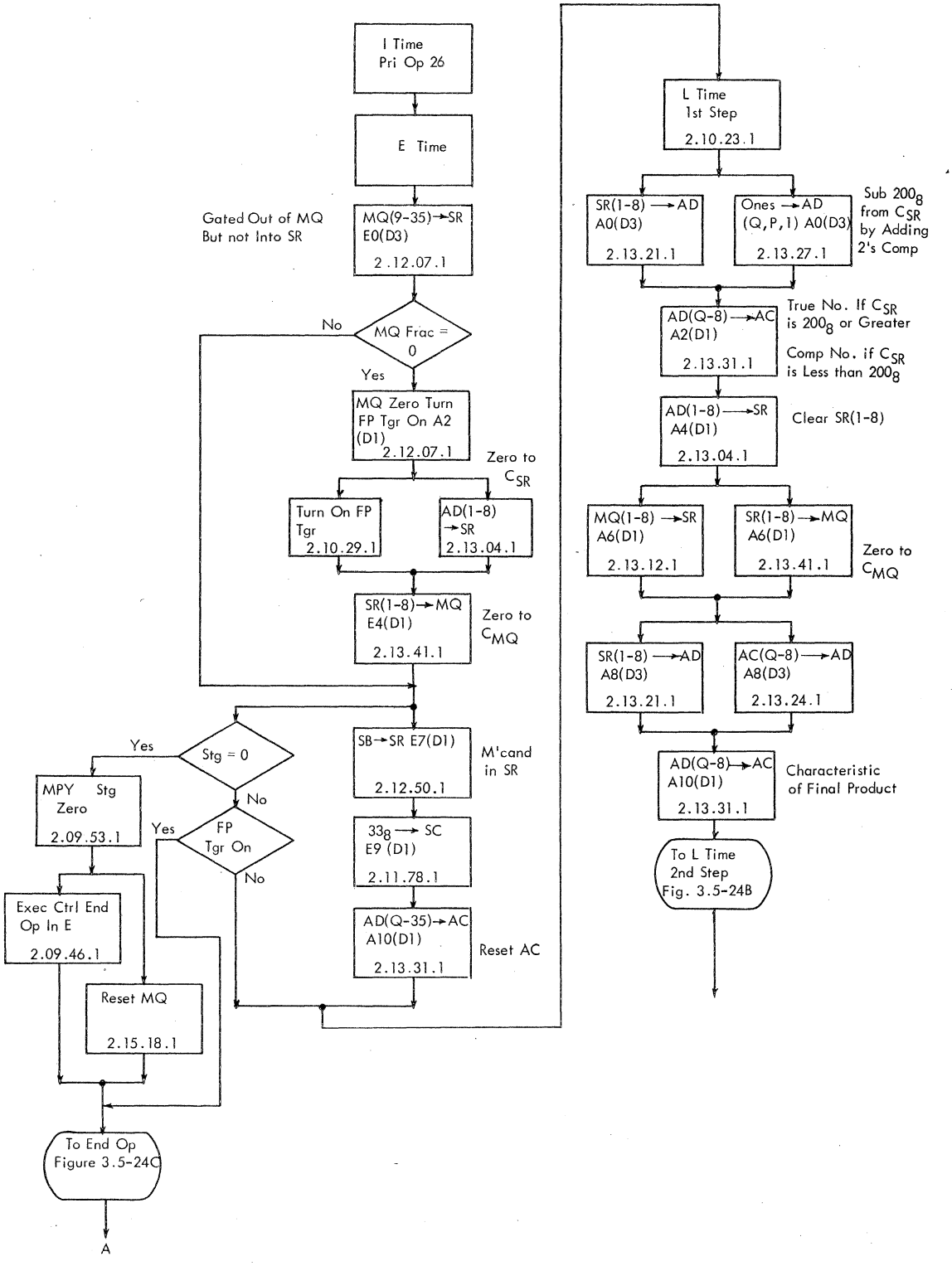


FIGURE 3.5-24A. FMP + 0260

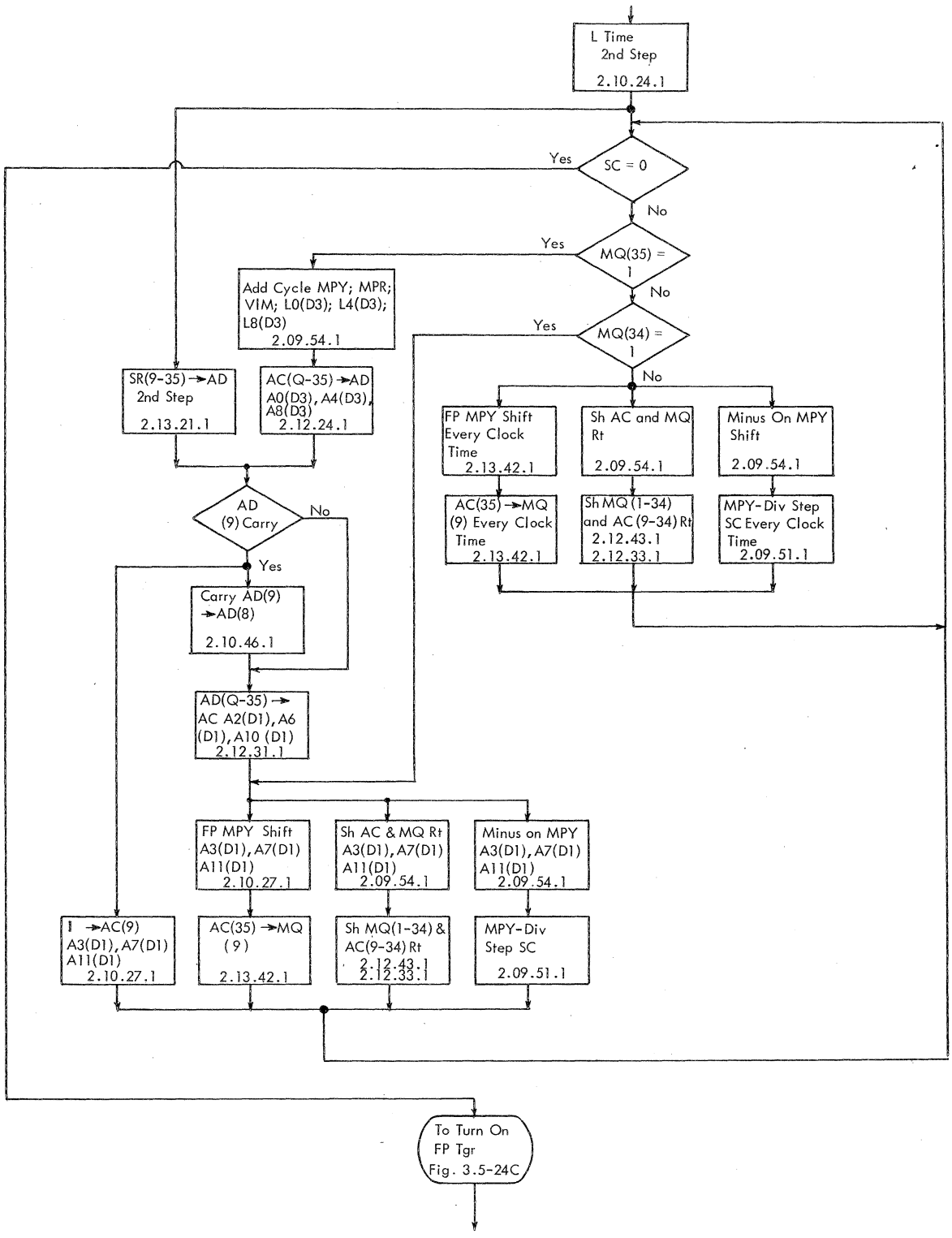


FIGURE 3.5-24B. FMP + 0260

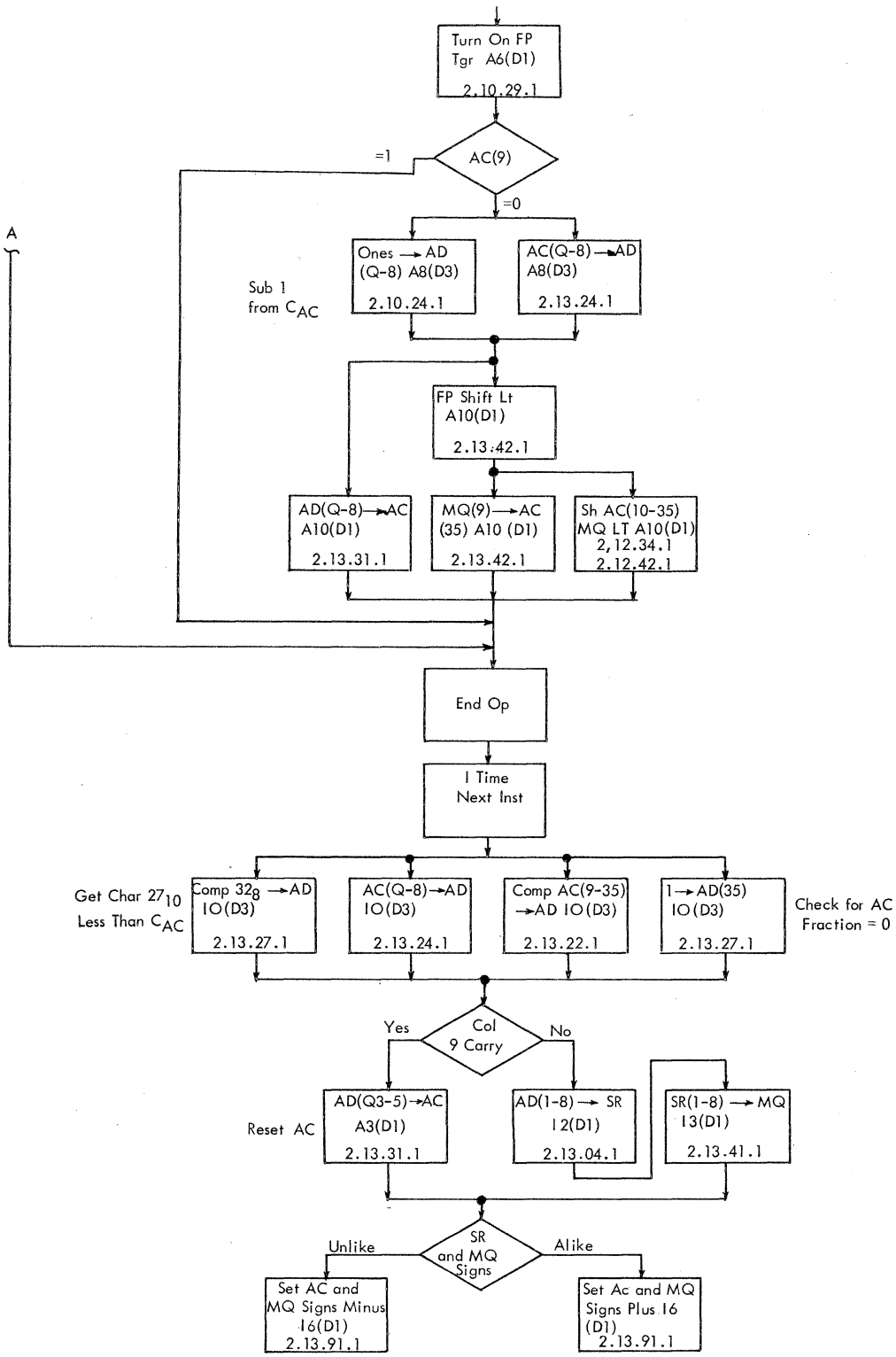


FIGURE 3.5-24C. FMP + 0260

to produce the proper characteristic of the product. Because both of the original characteristics are the exponent plus 200_8, adding these characteristics would produce a resultant characteristic 200_8 too large. Therefore, the addition of the characteristics must be adjusted by 200_8.

The execution of this instruction is accomplished by I, E, and L cycles. The L cycles are divided into first step L time and second step L time.

The E cycle, in addition to bringing the word from storage, has the following objectives:

1. For a zero MQ fraction
   a. Put zeros in the characteristic of the SR and the MQ
   b. End operation at the end of the E cycle
2. For a zero storage fraction
   a. Reset the MQ
   b. End operation at the end of the E cycle
3. Set 33_8 in the SC (used to count the number of shifts)
4. Reset the AC

If either the MQ or the storage fraction is zero, the product must also be zero. Therefore, for this zero condition the MQ(1-35) and the AC(Q-35) contain zeros at the end of the E cycle. This is a normal zero product. For this zero condition end operation comes at the end of E time because no further multiplication is necessary.

To check the MQ fraction for zero, MQ(9-35) is gated to the SR. The set and hold of the SR are not operated; the information does not go to the SR but to the zero checking circuits on Systems 2.12.47.1. The FP trigger is used to remember the MQ zero condition and to cause end operation at the end of the E cycle.

First step L time is used to get the initial characteristic of the final product. This characteristic may be changed at the end of FMP if normalizing is required.

The characteristic of the SR is reduced by 200_8 so the characteristic of the product will be only 200_8 greater than the exponent. The multiplicand characteristic minus 200_8 is put into the AC, and the multiplier characteristic is put into the SR. The two characteristics are added to produce the AC characteristic for the product. During this time the MQ characteristic is set to zero.

Second step L time is used to do the fraction multiplication and to normalize the product. The multiplication is done the same as in MPY, by shifting and adding. For no bits in MQ(34-35) fast shifting occurs, otherwise only three shifts per cycle take place. If there is a carry out of AC(9) during the add cycle it goes to AD(8) to increase the characteristic by one. To correct the fraction for this carry, a one is put into AC (9) when the AC and MQ are shifted right.

When 33_8 shifts are complete, the fraction multiplication is accomplished and the shift counter equals zero. The shift counter going to zero turns on the FP trigger, and the trigger is used to instruct the system that the multiplication part of the instruction is finished.

If AC(9) is zero, a single normalizing step is performed. One step is the maximum needed if two normal numbers were used for the multiplication.

During I time of the next instruction, the AC fraction is checked for zero by adding one to the 1's complement of the fraction. If the fraction is zero, the addition will cause a column 9 carry. For a zero AC fraction, the whole AC is set to zero and the characteristic of the MQ is left at zero. If the AC fraction is not zero, there is no column 9 carry and the MQ characteristic is set 27_{10} less than the AC characteristic. During I time of the next instruction, the AC and MQ signs are set to the algebraic sign of the product.

Unnormalized Floating Multiply UFM -0260

This instruction multiplies the floating-point number stored at the location indicated by the address by the floating-point number in the MQ. This instruction operates the same as FMP except that the result is not normalized or tested. The minus PR(S) prevents FP normalizing gate on Systems 2.10.2.4.1. The PR(S) not being plus prevents the reset of the AC during the following I cycle, on Systems 2.10.25.1.

Floating Round FRN +0760...0011 Figure 3.5-25

This instruction examines the contents of MQ(9) and if MQ(9) contains a bit, a one is added to the contents of the AC. A carry out of AC(9) increases the characteristic by one, causes the fraction to be shifted right, and a one to be placed in AC(9).

Floating Divide or Halt FDH +0240 Figure 3.5-26

This instruction divides the floating-point number stored in the AC by the floating-point number stored at location X. The result is a floating-point quotient in the MQ and the floating-point remainder of the dividend in the AC. The sign of the MQ is the algebraic sign of the quotient. The sign of the AC is the sign of the dividend. The characteristic of the MQ is equal to the difference between the dividend and divisor characteristics, increased by 128_{10}. The characteristic of the remainder is 27_{10} less than the original dividend characteristic.

If two normalized numbers are used as divisor and dividend fractions, the dividend fraction cannot be twice as large as the divisor fraction. Therefore, the 7090 is designated to halt only when the dividend fraction is at least twice as large as the divisor fraction. The divide check indicator is turned on to cause the halt.

A zero dividend fraction causes the division to be skipped, but does not halt the computer. A normal zero quotient results. If the initial factors are normal floating-point numbers, the quotient is also normal.

As in any division using exponents, the fractions are divided and the exponent of the divisor is subtracted from the exponent of the dividend. The result is the correct quotient factor and exponent. Division of the fractions is done in the same manner as DVH, by attempting reduction and shifting left. When the reduction is successful, a bit is put into MQ(35). Only 27_{10} reductions are attempted in FDH because the fraction is contained in 27_{10} positions.

FDH uses an E cycle and several L cycles. The L cycles are divided into three types (first, second, and third step) through use of the tally counter.

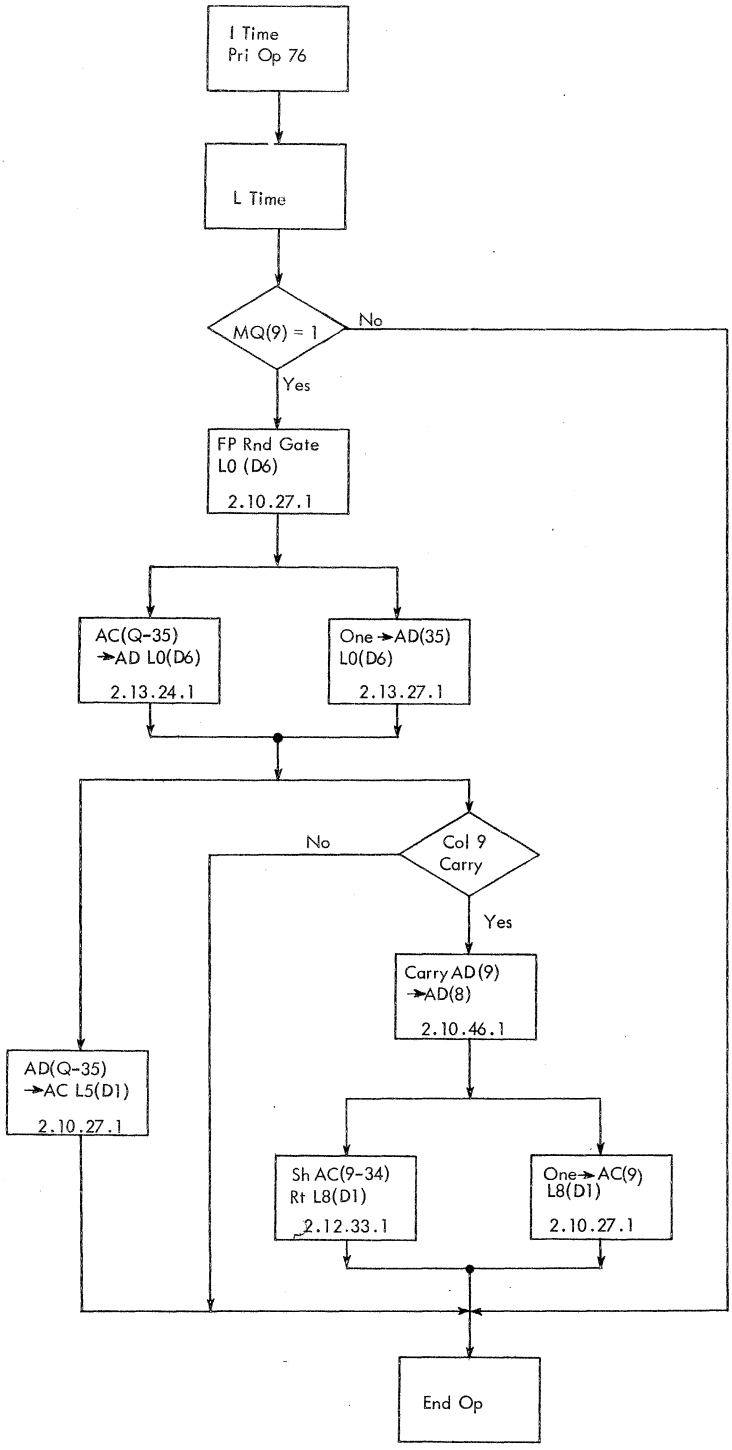


FIGURE 3.5-25. FRN +0760...0011

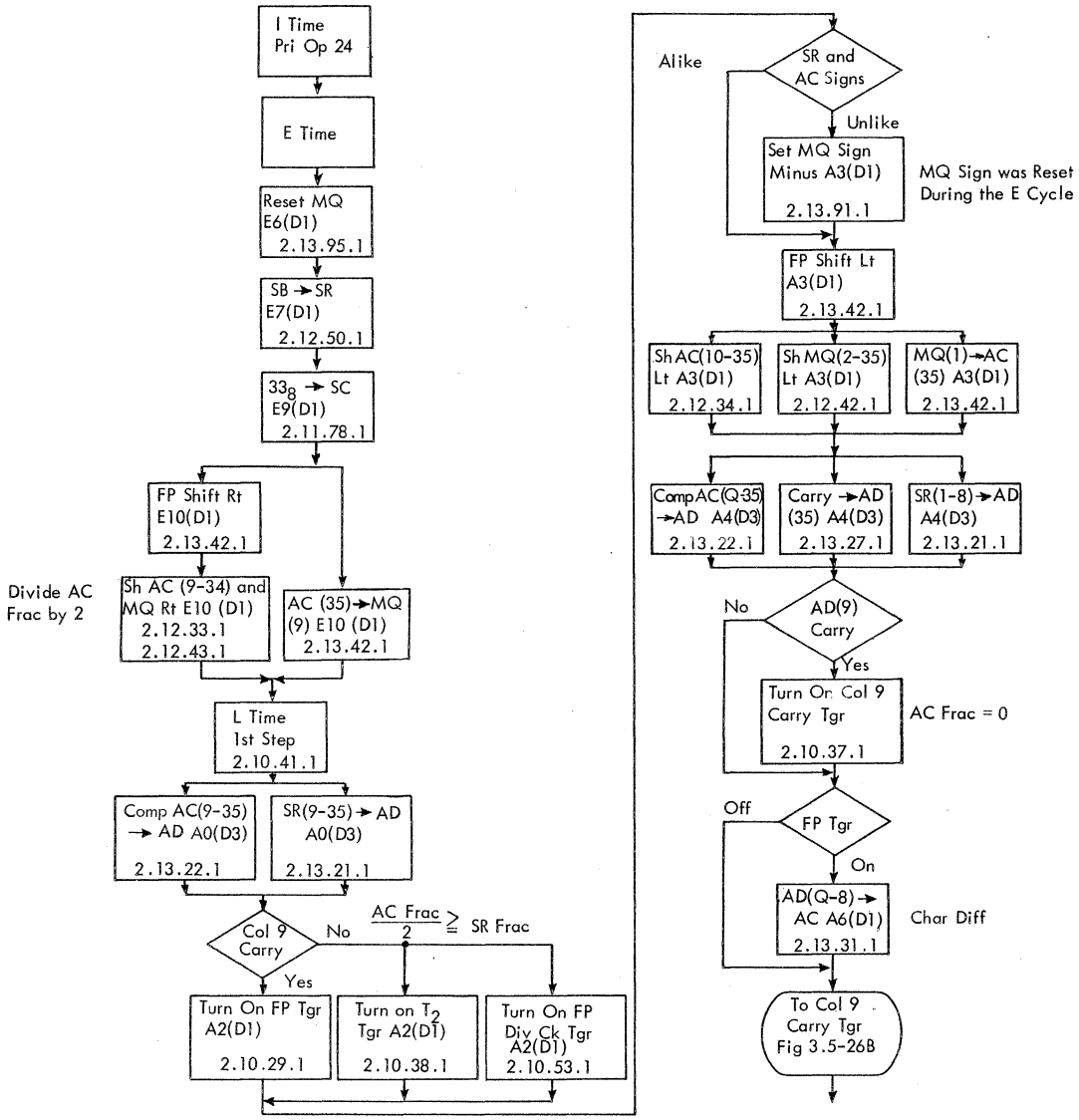


FIGURE 3.5-26A. FDH +0240

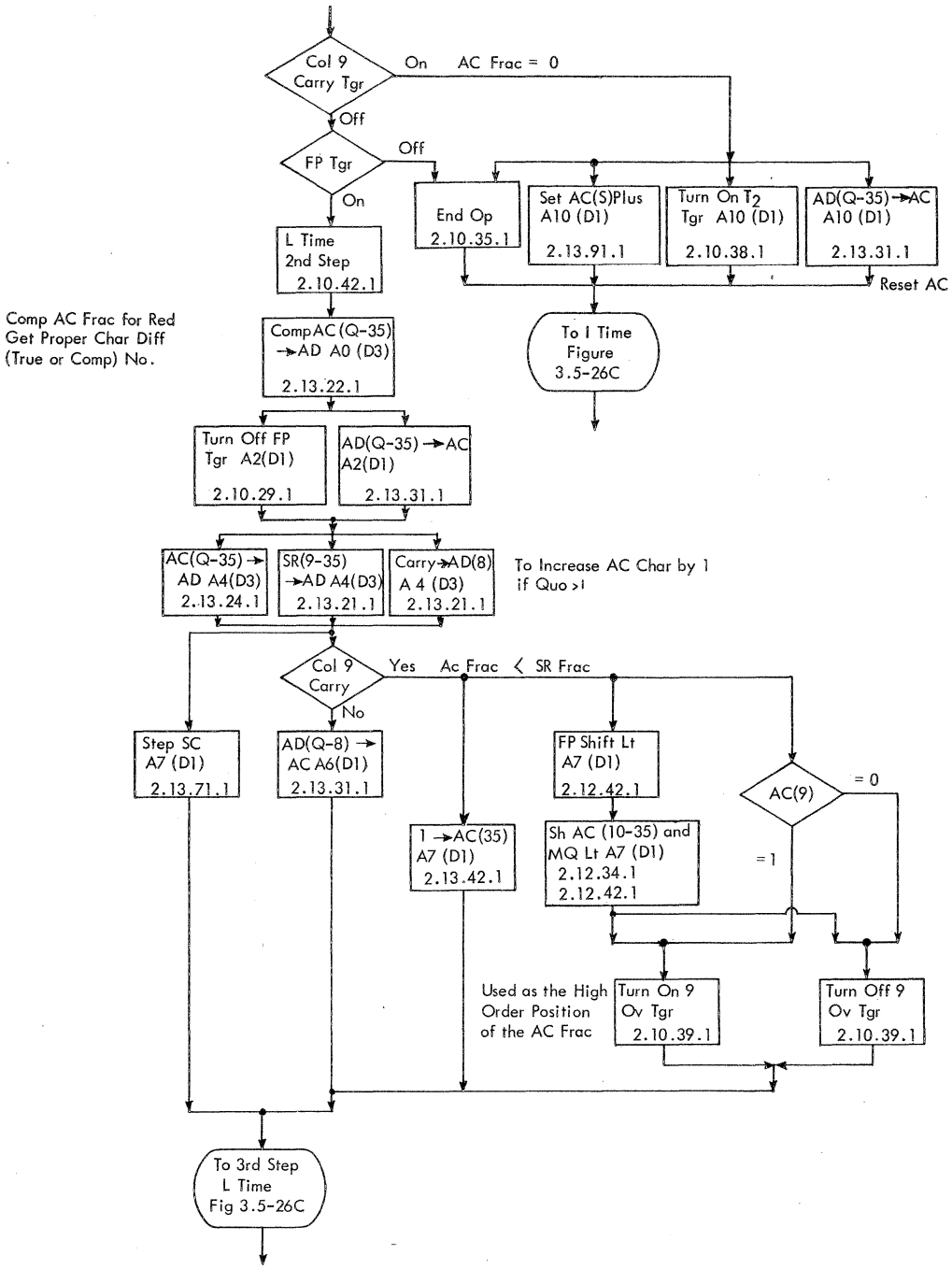


FIGURE 3.5-26B. FDH + 0240

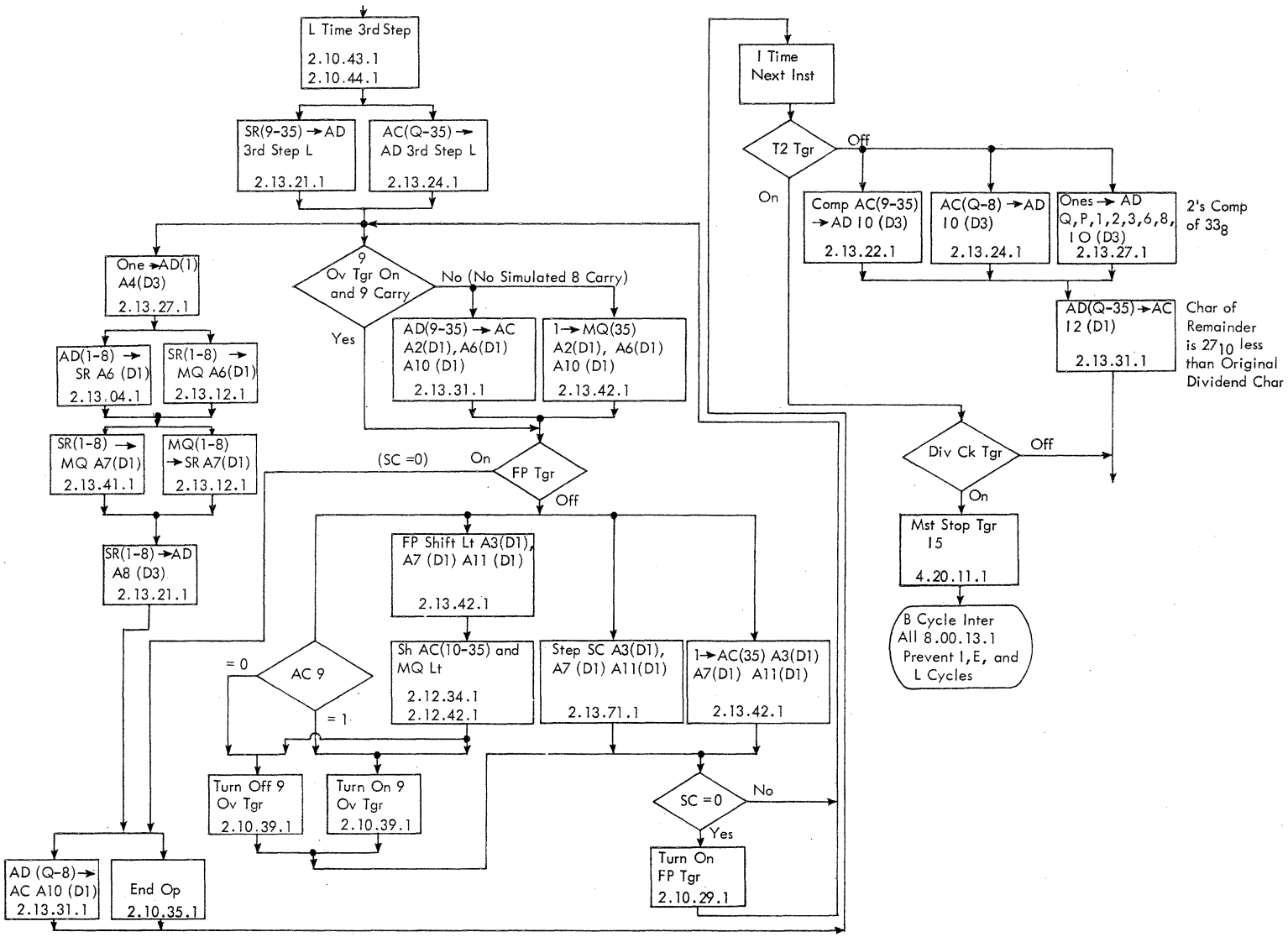


FIGURE 3.5-26C. FDH + 0240

The E cycle is used to:

1. Reset the MQ
2. Bring the divisor from storage
3. Set 33_8 (27_{10}) into the SC (used to count the number of shifts during divide)
4. Divide dividend fraction by 2 by shifting the fraction right one place (to be used for comparison of dividend fraction to divisor fraction)

First step L time is used to:

1. Determine if dividend fraction is more or less than two times the divisor fraction
2. Set the MQ(S) to the algebraic sign of the quotient
3. Shift dividend left one place to put dividend back to initial position
4. Check AC fraction for zero
5. Get the difference between divisor and dividend characteristics
6. End operation if the AC fraction is zero or if the dividend fraction is twice as large or larger than the divisor fraction
7. Reset the whole AC if the AC fraction is zero

Second step L time is used to:

1. Increase the characteristic difference by one if dividend fraction is equal to or greater than the divisor fraction
2. Shift the dividend fraction left one place if the dividend fraction is less than the divisor fraction
3. Step the shift counter down to 26_{10}--the desired number of shifts to move a bit from MQ(35) to MQ(9).

Third step L time is used to do the actual division. The objectives of third step L time are to:

1. Attempt to reduce the AC fraction by the amount of the divisor fraction
2. If the reduction is successful put a 1 in MQ(35)
3. Shift AC and MQ left one place, step shift counter, and put a 1 in AC(35)
4. Repeat steps 1, 2, and 3 until the SC equals zero
5. Compute the characteristic of the quotient and put it into the MQ
6. End operation when the shifting and reductions are complete

I time of the next instruction is used to:

1. If there was no divide check and the dividend fraction was not zero:
   a. AC fraction is re-complemented to get true number
   b. Characteristic of the remainder is set to 27_{10} less than the original characteristic of the dividend
2. Stop the machine if there was a divide check

Floating Divide or Proceed FDP +0241

This instruction operates the same as FDH, except that a divide check does not halt the computer.

3.5.05 Transfer Instructions

Transfer instructions are used to alter the sequence of instructions. The conditional transfers allows automatic testing of problem conditions without stopping the

computer. The transfer instructions greatly reduce program length by allowing program loops and subroutine operation.

Transfer TRA +0020 Figure 3.5-27

The transfer instruction causes the computer to take its next instruction from location X and resets the instruction counter to X. The address portion of the transfer instruction is substituted for the instruction counter when setting the address register to locate the next instruction. The instruction counter is then set to this new address, and the instruction sequence continues from the new location.

Transfer on MQ Plus TQP +0162 Figure 3.5-28

If the sign of the MQ register is positive, a transfer will be taken to storage location X. If the MQ sign is minus, the computer proceeds to the next instruction in sequence.

Transfer on Plus TPL +0120

If the sign of the accumulator is plus, a transfer is taken to storage location X. If the sign is minus, the computer proceeds to the next instruction in sequence. TPL is executed in the same manner as TQP, except that the AC sign rather than the MQ sign is tested. See Systems 2.10.08.1.

Transfer on Minus TMI -0120

If the sign of the AC is minus, a transfer is taken to storage location X. If the AC sign is plus, the computer proceeds to the next instruction in sequence. The execution of this instruction is the same as TQP, except that the AC sign rather than the MQ sign is tested. See Systems 2.10.08.1.

Transfer on Overflow TOV +0140

If the AC overflow trigger is on as a result of a previous operation, a transfer is taken to storage location X, and the overflow trigger is turned off. If the overflow trigger is off, the computer proceeds to the next instruction in sequence. Execution of TOV is much like TQP, except that the overflow trigger rather than the MQ sign is tested. See Systems 2.10.08.1. The AC overflow trigger is turned off on Systems 2.10.36.1.

Transfer on No Overflow TNO -0140

If the AC overflow trigger is off, the next instruction is taken from storage location X. If the overflow trigger is on, no transfer is taken. Operation of this instruction is the same as TQP, except that the overflow trigger rather than the MQ sign is tested.

Transfer on Zero TZE +0100 Figure 3.5-29

If the contents of the AC (including the overflow positions) are zero, a transfer is taken to storage location X. If the contents are not zero, the computer proceeds to the next instruction in sequence. In either case the contents of the AC are not changed.

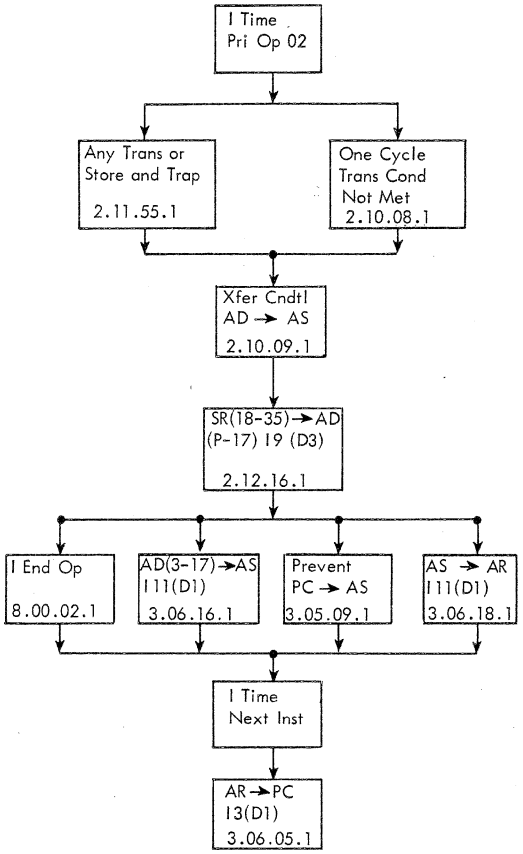


FIGURE 3.5-27. TRA + 0020

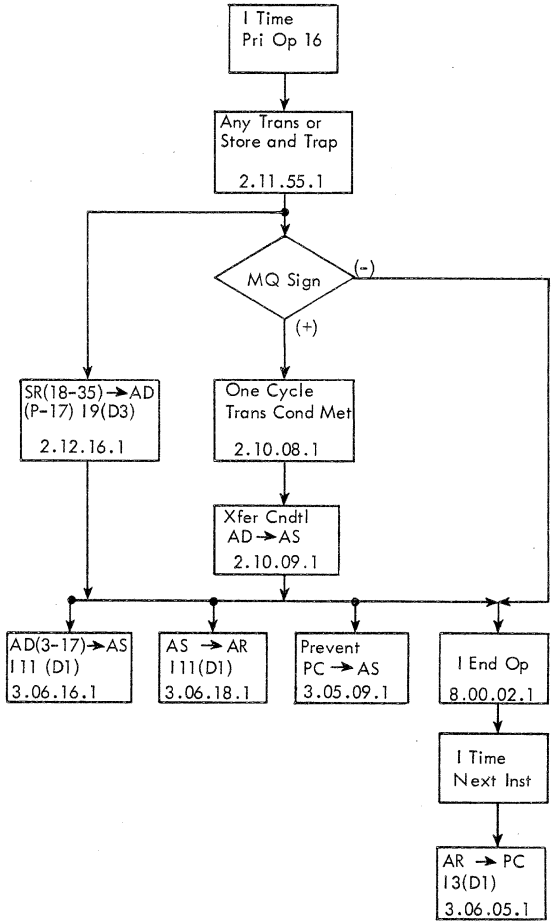


FIGURE 3.5-28. TQP + 0162

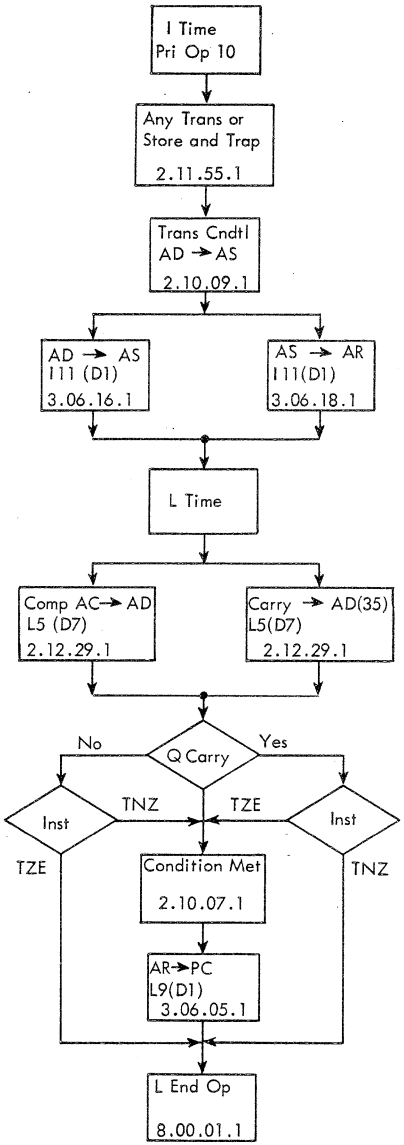


FIGURE 3.5-29. TZE + 0100; TNZ - 0100

To test for a zero condition, the accumulator is complemented to the adders, and a one is added to position 35. If a zero condition exists, a carry results which ripples through all of the adders and turns on the Q carry trigger. The Q carry trigger is used to condition the transfer circuits.

Transfer on No Zero TNZ -0100 Figure 3.5-29

If the contents of the AC are not zero, the next instruction is taken from storage location X. If the contents are zero, the computer proceeds to the next instruction in sequence.

Transfer on Low MQ TLQ +0040 Figure 3.5-30

An algebraic comparison is made between the MQ and the accumulator contents. If the MQ contents are lower than the accumulator contents, an instruction transfer is made to storage location X. If the AC is greater or equal, no transfer is taken. For this instruction a +0 is considered to be larger than a -0. The contents of both registers are left unchanged.

The operation is performed by adding the contents of the MQ to the complemented accumulator contents. The Q carry trigger is then matched with the register signs to determine whether the conditions for transfer have been met. To prevent transfers when the factors are equal, a one is added to AC(35) to produce a Q carry when the AC factor is plus.

The following table illustrates comparisons which might be made, along with the desired result:

No. in AC	No. in MQ	Q Carry	Transfer
-7	-6	No	No
-6	-6	No	No
-0	-6	Yes	Yes
-0	-0	No	No
+0	-0	Yes	Yes
-0	+0	No	No
-6	+6	No	No
+0	+6	Yes	No
+6	+6	Yes	No
+7	+6	No	Yes

Transfer on Channel in Operation TCO +XXXX Figure 3.5-31

This instruction causes a transfer to storage location X if the particular data channel is in use. Operation codes +0060 through +0067 are used to check data channels A through H, respectively.

Transfer on Channel Not in Operation TCN -XXXX Figure 3.5-31

TCN causes a transfer to storage location X if the data channel is not in operation. Operation codes -0060 through -0067 are used to select data channels A through H, respectively.

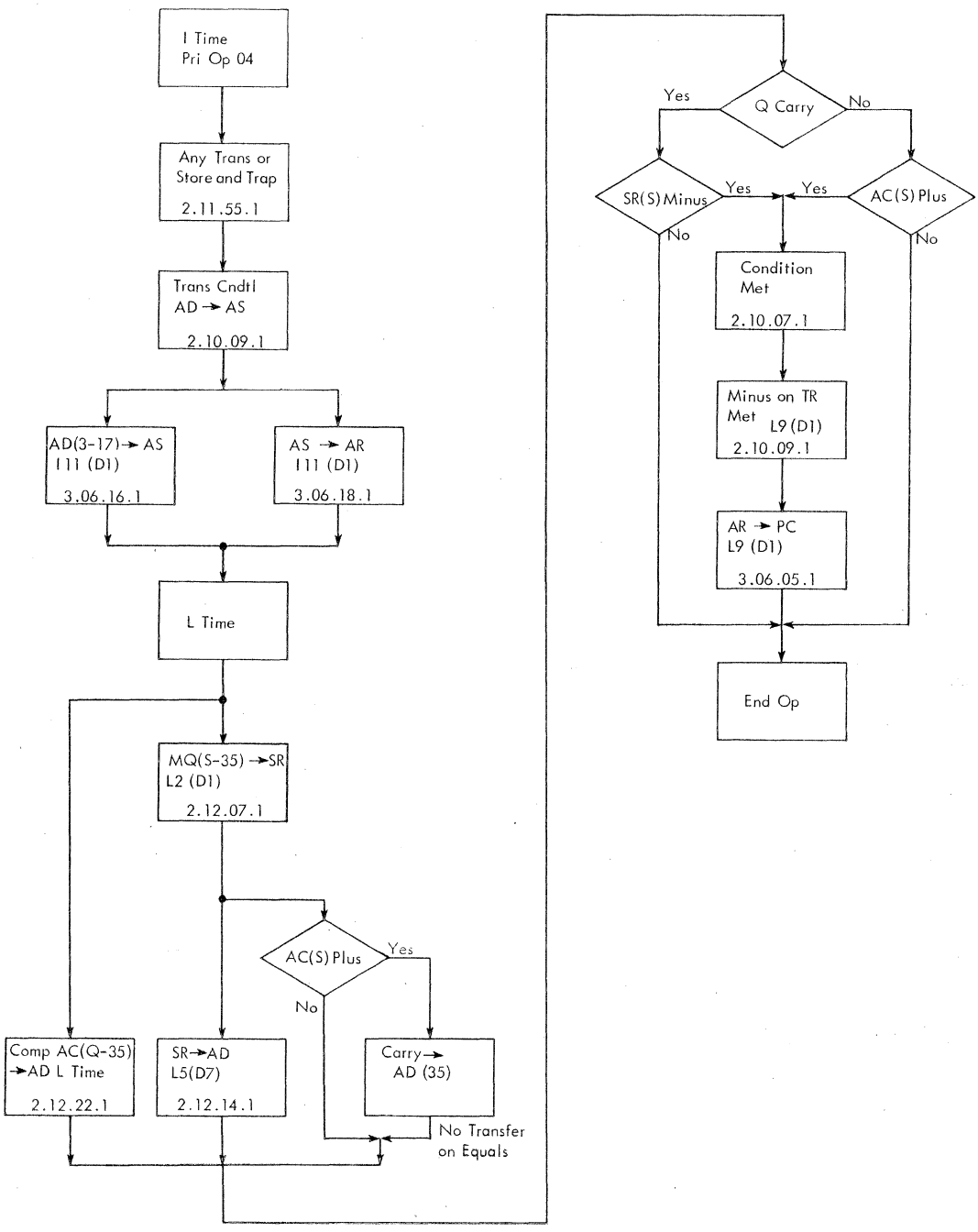


FIGURE 3.5-30. TLQ + 0040

Transfer on Data Channel Redundancy Check TRC ±XXXX Figure 3.5-32

If the data channel redundancy check trigger is on, a transfer is taken to location X and the trigger is turned off. If the trigger is off, the computer takes the next sequential instruction. Operation codes +0022, -0022, +0024, -0024, +0026, -0026, +0027, and -0027 are used to select data channels A through H, respectively.

Transfer on Data Channel End of File TEF ±XXXX Figure 3.5-32

If the data channel end-of-file trigger is on, a transfer is taken to storage location X and the trigger is turned off. If the trigger is off, the computer takes the next instruction in sequence. Operation codes +0030, -0030, +0031, -0031, +0032, -0032, +0033, and -0033 are used to select data channels A through H, respectively.

3.5.06 Trap Mode Instructions

The 7090 can be operated in either of two modes, normal or trapping. Entrance to trapping mode is gained by executing the ETM instruction. Exit is accomplished by using the LTM instruction or the clear or reset keys on the console. Trapping mode effects the operation of transfer instructions. In trapping mode, the location of each successful transfer instruction is stored in the address portion of location 0000. Transfers are not executed; instead, an instruction transfer, or trap, is taken to location 0001. One instruction, trap transfer, is immune to trapping mode. The locating of successful transfers and the trap to a common check point make trapping mode useful for debugging program flow.

Enter Trapping Mode ETM +0760...0007

This instruction places the computer in trapping mode by turning on the trap mode trigger on Systems 2.10.53.1. The computer remains in trapping mode until a LTM instruction is executed or the clear or reset buttons are depressed. ETM is a primary operation 76 instruction and requires an I and an L cycle.

TRA in Trapping Mode TRA +0020

The effect of trapping mode on a transfer is shown in Figure 3.5-33.

Leave Trapping Mode LTM -0760...0007

This instruction returns the computer to normal mode by turning off the trap mode trigger on Systems 2.10.53.1. This is a primary operation 76 instruction, and an I and an L cycle are required.

Trap Transfer TTR +0021

This is the only instruction which provides an instruction transfer to location X regardless of the operating mode of the computer. The TTR instruction nullifies the transfer blocking circuits of trapping mode and operates like TRA.

Store Location and Trap STR -1000 Figure 3.5-34

This instruction stores its location plus one in the address portion of storage loca-

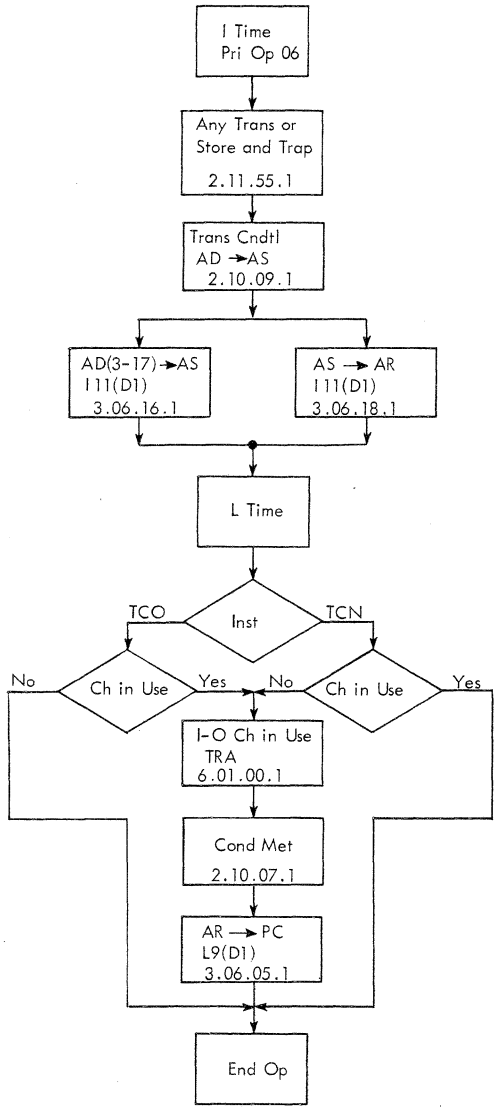


FIGURE 3.5-31. TCO + 0060; + 0061; + 0062; Etc.
TCN − 0060; − 0061; - 0062; Etc.

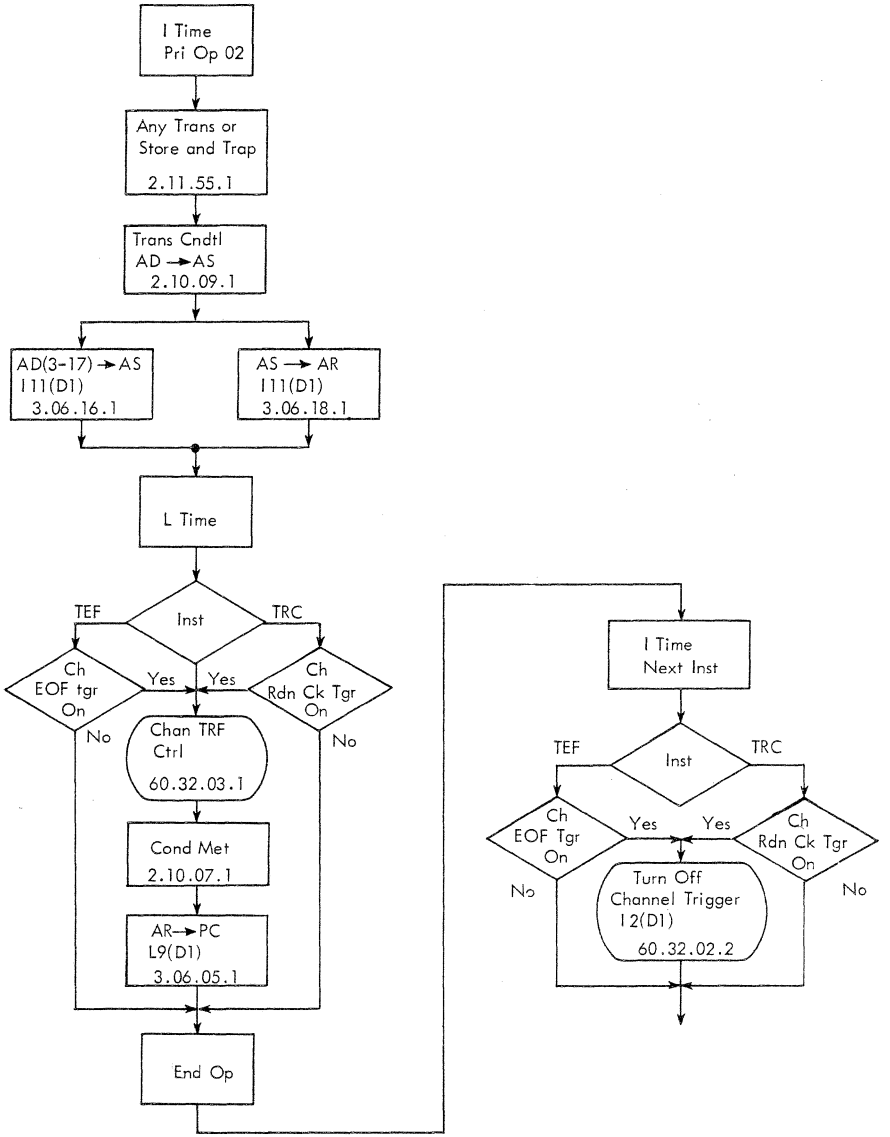


FIGURE 3.5-32. TRC + 0022; -0022; + 0024; Etc.
TEF + 0030; -0030; + 0031; Etc.

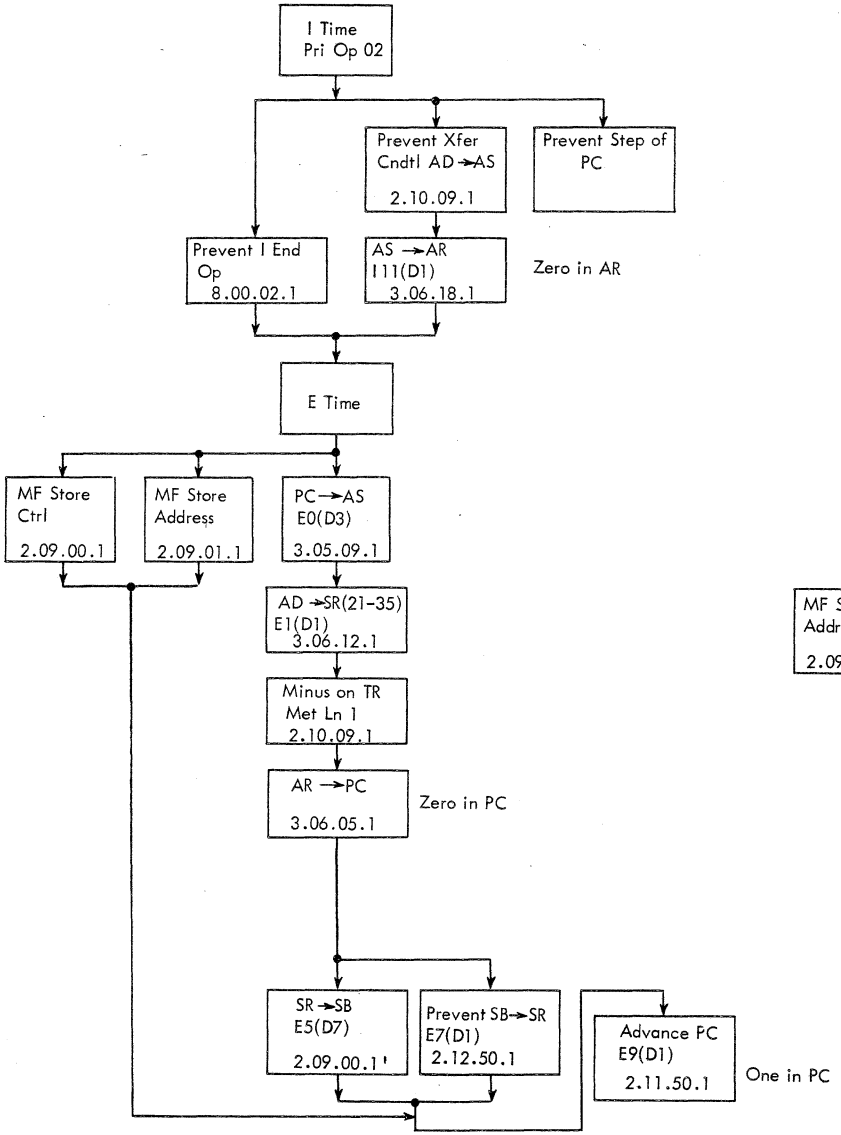


FIGURE 3.5-33. TRA +0020 TRAP MODE

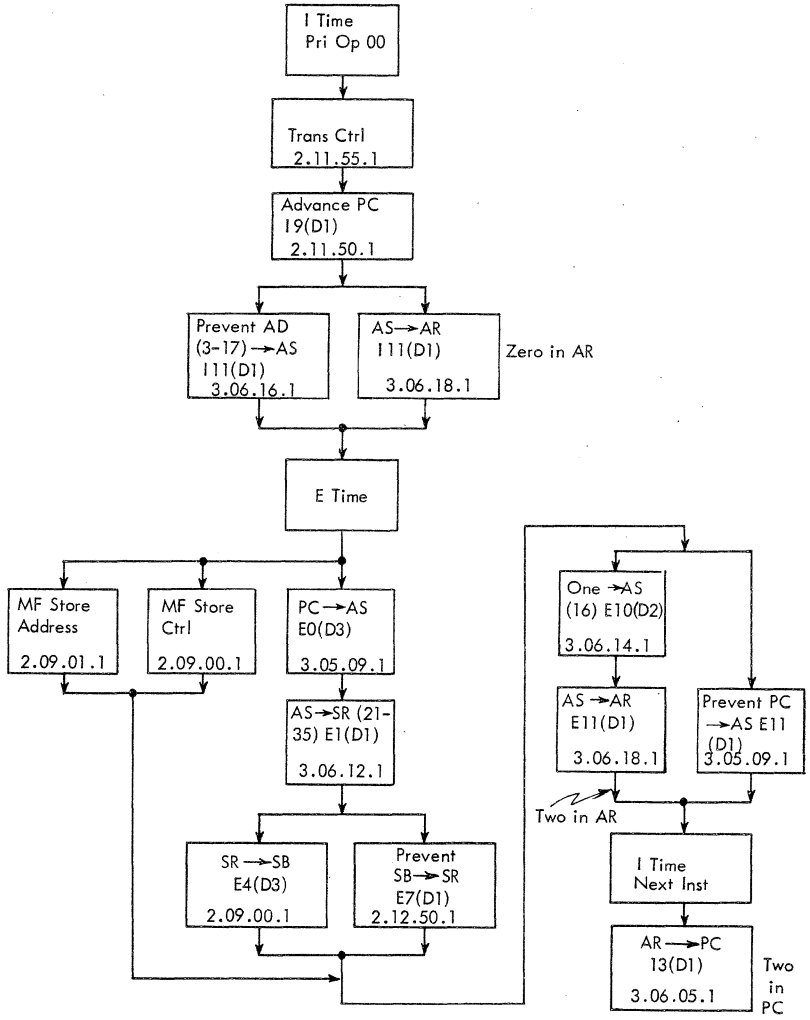


FIGURE 3.5-34. STR - 1000

tion 0000. It then traps or transfers the computer to location 0002 where the instruction sequence is resumed. STR does not place the computer in trapping mode.

3.5.07 Skip Instructions

The skip instructions allow the programmer to alter the program to meet special conditions without stopping the computer. These instructions are similar to the conditional transfer instructions but, instead of transferring, they cause one or two instructions to be skipped. Skipping is accomplished by supplying an extra advance pulse to the instruction counter.

Most skip instructions cause the computer to skip when the condition being tested is met. The exceptions are the error testing and I-O testing instructions, which cause skipping when the condition being tested is not met; e.g., when there are no errors. This exception (skip on no error) allows a straight-line program until an error is detected. To process an error, an instruction transferring to an error subroutine usually follows the test instruction. Another exception, the CAS instruction, has three possible results: it can fail to skip for AC greater, skip once for equal, or skip twice for AC less.

P Bit Test PBT -0760...0001 Figure 3.5-35

A bit in accumulator (P) position causes the computer to skip one instruction. If there is no bit in (P), the computer takes the next instruction in sequence.

Low-Order Bit Test LBT +0760...0001 Figure 3.5-35

A bit in accumulator (35) causes the computer to skip one instruction. If there is no bit in (35), the computer takes the next instruction in sequence.

Storage Zero Test ZET +0520 Figure 3.5-36

If the contents of storage location X, except the sign, are zero, the computer will skip one instruction. If storage is not zero, the computer proceeds to the next instruction in sequence. Storage is unchanged. The information from storage on the SB is tested for zero as shown on Systems 2.12.52.1.

Storage Non-Zero Test NZT -0520 Figure 3.5-36

If positions (1-35) of storage location X are not zero, the computer skips the next instruction. If the contents of storage location X are zero, no skip is taken. Storage is unchanged.

Compare Accumulator with Storage CAS +0340 Figure 3.5-37

The accumulator is compared with the word at storage location X. Comparison is accomplished by taking an algebraic difference. If the accumulator is greater than the word in storage, no skip is taken. If the accumulator is equal to the word in storage, one instruction is skipped. If the accumulator is less than the word in storage, the next two instructions are skipped. Neither the accumulator nor the word in storage is changed.

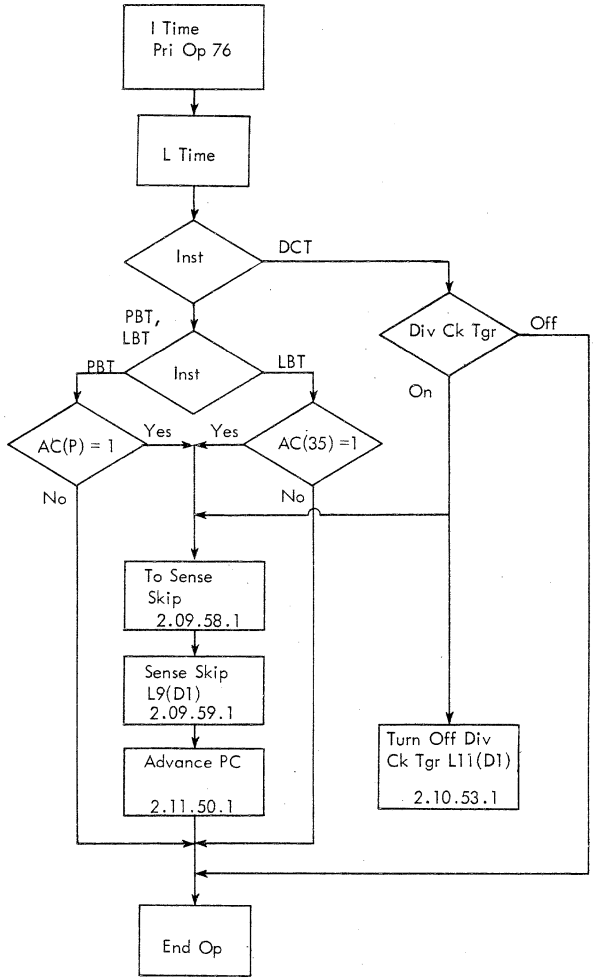


FIGURE 3.5-35. PBT - 0760...0001; LBT +0760...0001; DCT +0760...0012

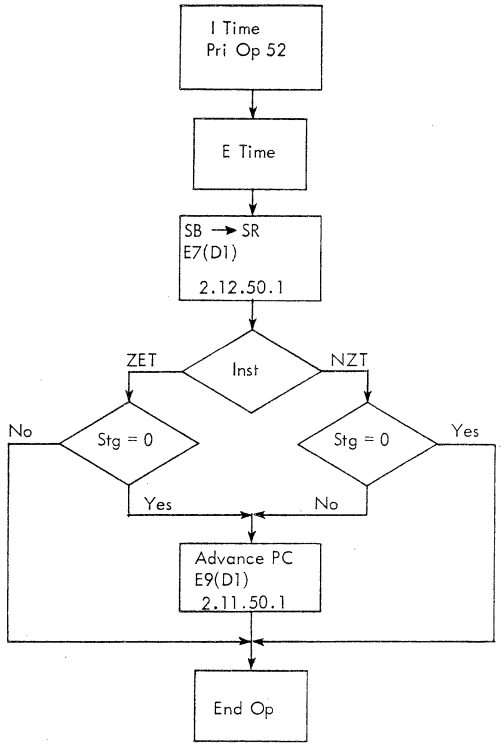


FIGURE 3.5-36. ZET +0520; NZT - 0520

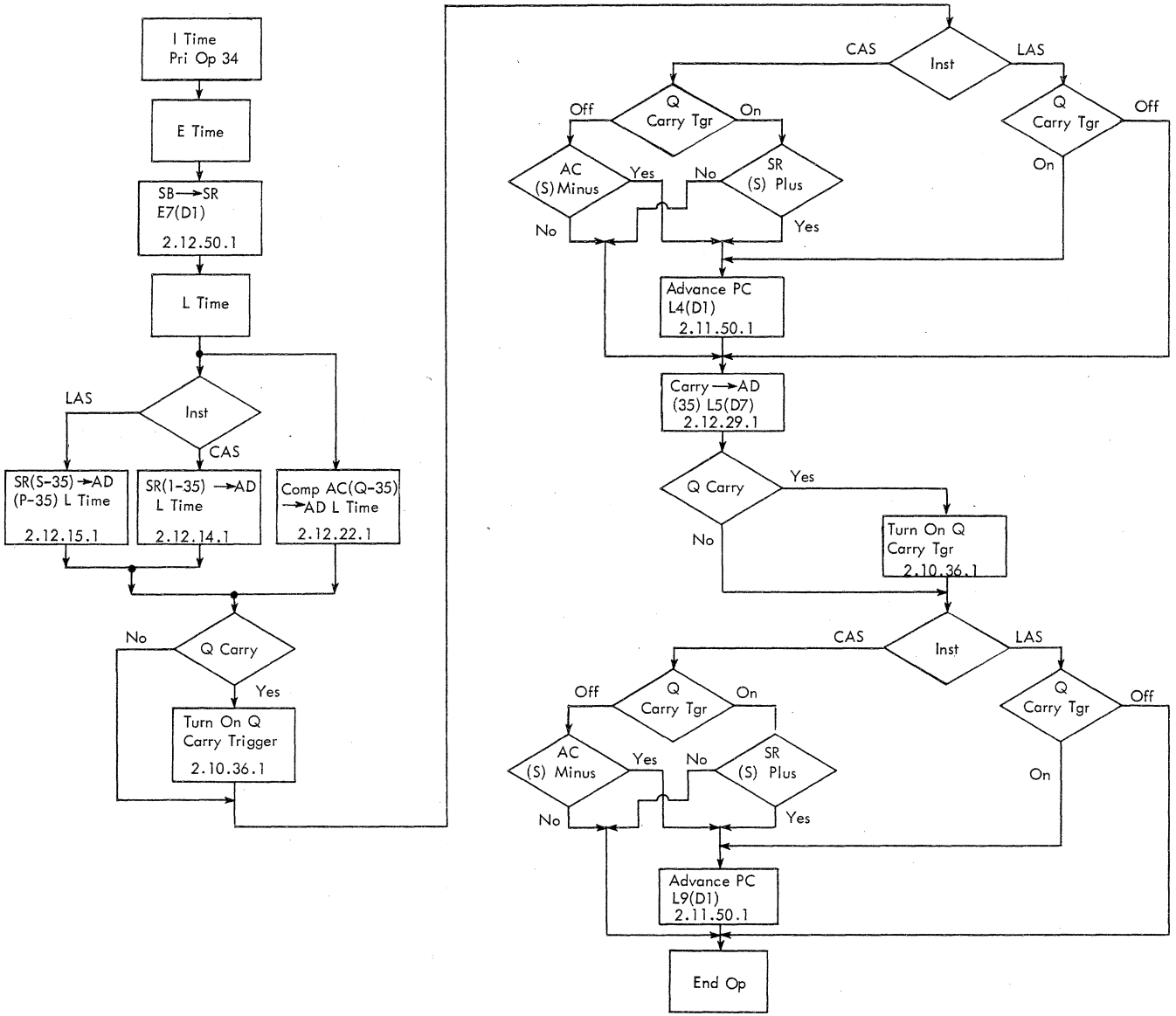


FIGURE 3.5-37. CAS +0340; LAS - 0340

To execute this instruction, an E cycle is required to bring the word in storage to the storage register; then an L cycle is used for two comparisons in the adders. For the first comparison, the SR and the complement of the AC are fed to the adders; the Q carry and sign conditions are matched to condition the first possible skip. The second comparison is made after a one has been added to the difference, to differentiate words of equal magnitude.

The following illustrate some of the possible combinations:

Number in Accumulator	Number in Storage	Q Carry Tgr 1st Comp	Q Carry Tgr 2nd Comp	Result
-6	-7	On	On	Next Instruction
-6	-6	Off	On	Skip 1 Instruction
-6	-4	Off	Off	Skip 2 Instructions
-6	+6	Off	On	Skip 2 Instructions
-0	+0	Off	On	Skip 2 Instructions
+0	-0	Off	On	Next Instruction
+6	-6	Off	On	Next Instruction
+6	+4	Off	Off	Next Instruction
+6	+6	Off	On	Skip 1 Instruction
+6	+7	On	On	Skip 2 Instructions

Logical Compare Accumulator with Storage LAS -0340 Figure 3.5-37

This instruction compares the contents of the AC(P, 1-35) with the logical word (S, 1-35) stored at location X. The sign of the AC is disregarded; the contents of the AC and storage are unchanged.

If the contents of the AC are greater than the contents of storage location X, the computer takes the next instruction in sequence. If the AC equals storage, the computer skips one instruction. If the contents of the AC are less than the contents of storage, the computer will skip the next two instructions.

This instruction is executed the same as CAS, except that the signs are not used for an algebraic comparison and AC(P) is compared against SR(S).

Plus Sense PSE +0760...XXXX

This instruction provides a means to test the status of any of the six sense switches, to turn on or off the four console sense lights, and to permit the transmission of an impulse to or from the exit or entry hubs of either the printer or punch.

The address portion of the instruction determines whether a light, switch, printer, or card punch is being sensed; further, it determines which light, switch, or hub is being sensed. The octal addresses for the different sense instructions are:

Address	Instruction
0140	Turn off all sense lights (Figure 3.5-38)
0141-0144	Turn on sense light 1, 2, 3, or 4, respectively (Figure 3.5-39)

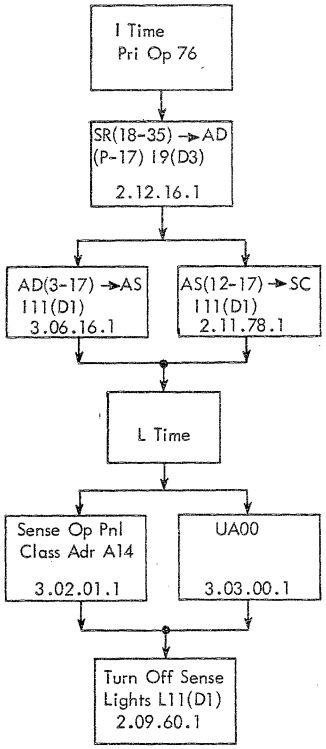


FIGURE 3.5-38. PSE +0760...0140

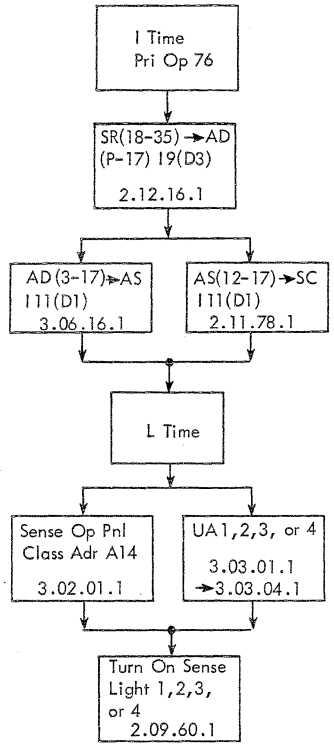


FIGURE 3.5-39. PSE +0760...0141, 0142, 0143, 0144

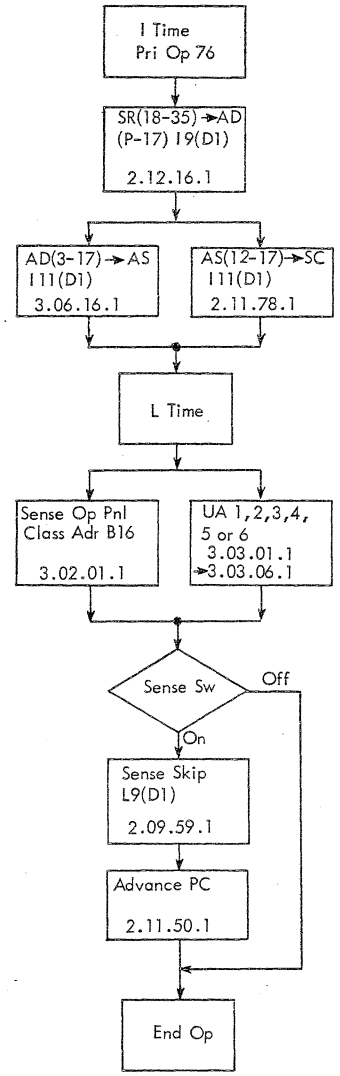


FIGURE 3.5-40. PSE +0760...0161, 0162, ETC.

Address	Instruction
0161- 0166	If the corresponding sense switch is down (on), the computer skips the next instruction. If the sense switch is up (off), the computer takes the next instruction in sequence (Figure 3.5-40).
1341- 1342 2341- 2342 3341- 3342 4341- 4342 5341- 5342 6341- 6342 7341- 7342 10341-10342	The computer causes an impulse to appear at the specified exit hub of the control panel of the card punch attached to Data Channel A, B, C, D, E, F, G, or H respectively (Figure 3.5-41).
1360 2360 3360 4360 5360 6360 7360 10360	If an impulse is present at the entry hub of the control panel of the printer attached to Data Channel A, B, C, D, E, F, G, or H respectively, the computer skips the next instruction. If there is no impulse, the computer takes the next instruction in sequence (Figure 3.5-42).
1361- 1372 2361- 2372 3361- 3372 4361- 4372 5361- 5372 6361- 6372 7361- 7372 10361-10372	The computer causes an impulse to appear at the specified exit hub of the control panel of the printer attached to Data Channel, A, B, C, D, E, F, G, or H, respectively (Figure 3.5-43).

Minus Sense MSE -0760...XXXX Figure 3.5-44

If the sense light, on the operator's console, corresponding to the address portion of the instruction is on, this light is turned off and the computer skips the next instruction. If the sense light is off the computer takes the next instruction in sequence. Addresses 0141-0144 correspond to sense lights 1-4, respectively.

Input-Output Check Test IOT +0760...0005 Figure 3.5-45

If the I-O check trigger is on, the indicator is turned off and the computer takes the next instruction in sequence. If the I-O check indicator is off, the computer skips the next instruction.

Divide Check Test DCT +0760...0012 Figure 3.5-35

This instruction examines the status of the divide check trigger. If the trigger is off, the next instruction is skipped. If the trigger is on, it is turned off and the computer takes the next instruction in sequence.

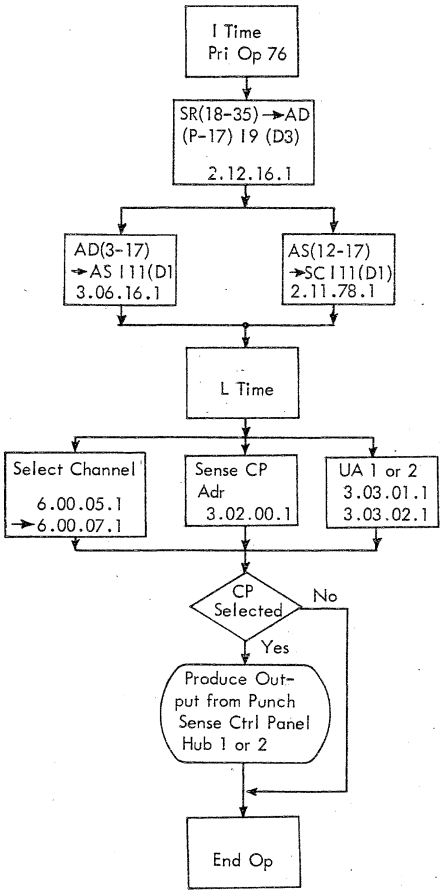


FIGURE 3.5-41. PSE +0760...1341, 2342, 3341, ETC.

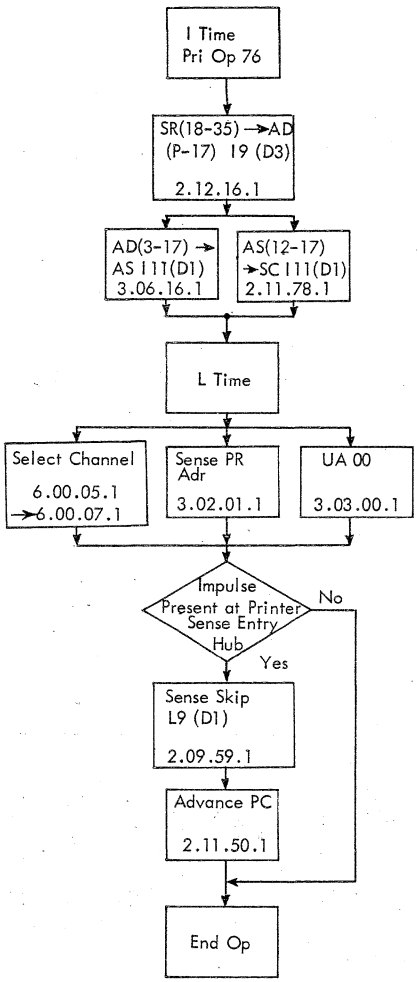


FIGURE 3.5-42. PSE +0760...1360, 2360, ETC.

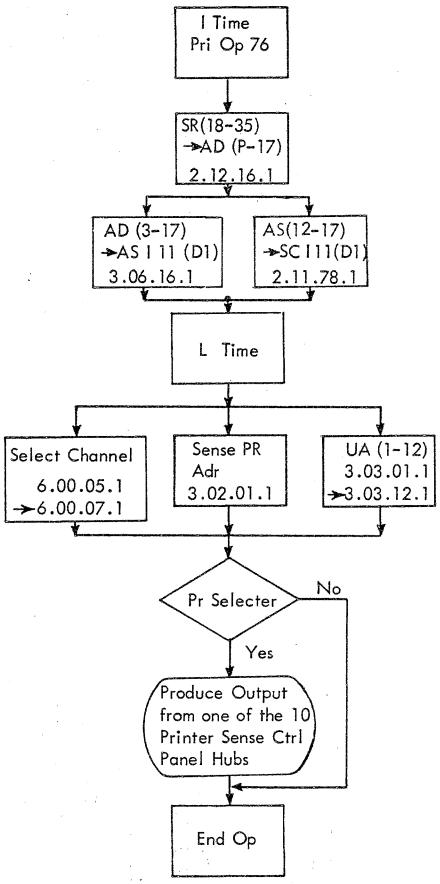


FIGURE 3.5-43. PSE +0760...1361, 1372, 2361, 2362, ETC.

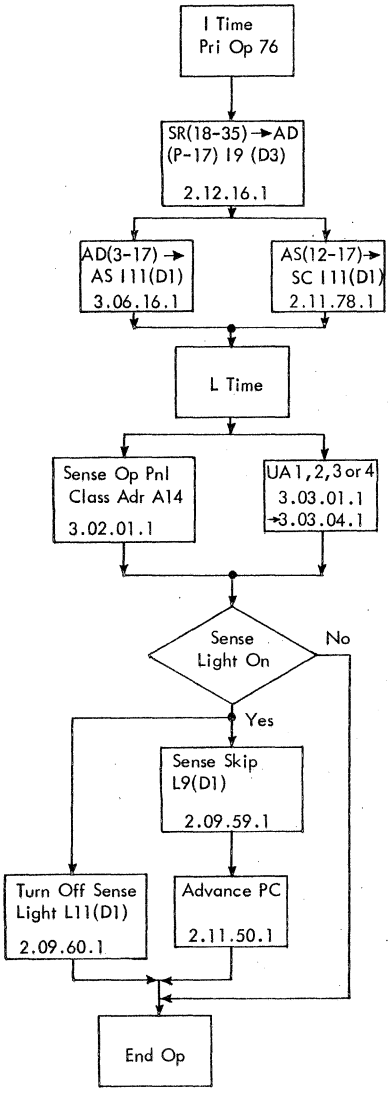


FIGURE 3.5-44. MSE -0760...0141, 0142, 0143, 0144

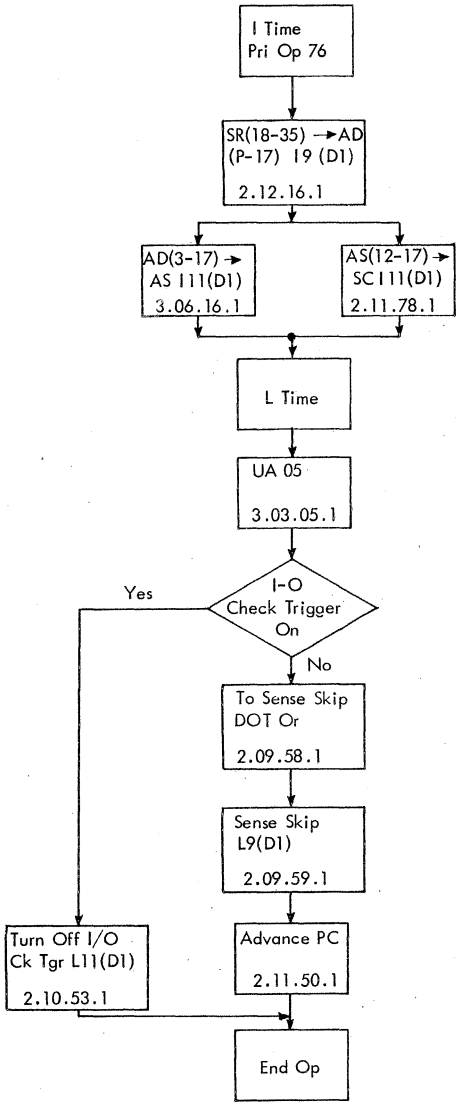


FIGURE 3.5-45. IOT +0760...0005

Beginning-of-Tape Test BTT +0760...XXXX Figure 3.5-46

This instruction tests the status of the beginning-of-tape indicator in a particular data channel. Address 1000-10000 selects data channels A-H respectively. If the beginning-of-tape indicator for the selected channel has been turned on by a previous instruction, the indicator is turned off and the computer takes the next instruction in sequence. If the indicator is already off, the computer skips the next instruction.

End-of-Tape Test ETT -0760...XXXX Figure 3.5-46

This instruction uses address 1000-10000 to select the data channel in which the end-of-tape indicator is to be tested. If the indicator is on, it is turned off and the computer takes the next instruction in sequence. If the indicator is off, the computer skips the next instruction.

3.5.08 Control Instructions

Control instructions are provided so the programmer may change the problem conditions or service the computer.

Halt and Proceed HPR +0420 Figure 3.5-47

This instruction causes the computer to stop at the end of I time. When the start button is depressed, the computer proceeds to the next instruction in sequence.

Halt and Transfer HTR +0000 Figure 3.5-48

This instruction causes the computer to stop at the end of I time by turning on the master stop trigger at I_{11}(D1). Depressing the start button causes the computer to proceed in L time of a transfer operation and the computer takes an instruction transfer to location X.

No Operation NOP +0761

The NOP instruction performs no active function, but is used to reserve space for other instructions. Since this instruction has a primary operation 76, an I and an L cycle are required. The only function of this instruction is to turn on the end operation trigger to allow the computer to proceed to I time of the next sequential instruction. SOD O1 on Systems 3.07.01.1 causes L END OP on Systems 8.00.09.1.

Execute XEC +0522 Figure 3.5-49

This instruction causes the computer to perform the instruction at location X. The program counter is not altered; therefore, after the instruction at location X has been executed, the computer proceeds to the next sequential instruction (instruction in next position beyond the XEC instruction). XEC prevents AR to PC on Systems 3.06.05.1.

Set Sign Plus SSP +0760...0003 Figure 3.5-50

This instruction places a zero (a plus) in the accumulator sign position. Positions (Q-35) of the accumulator are unchanged.

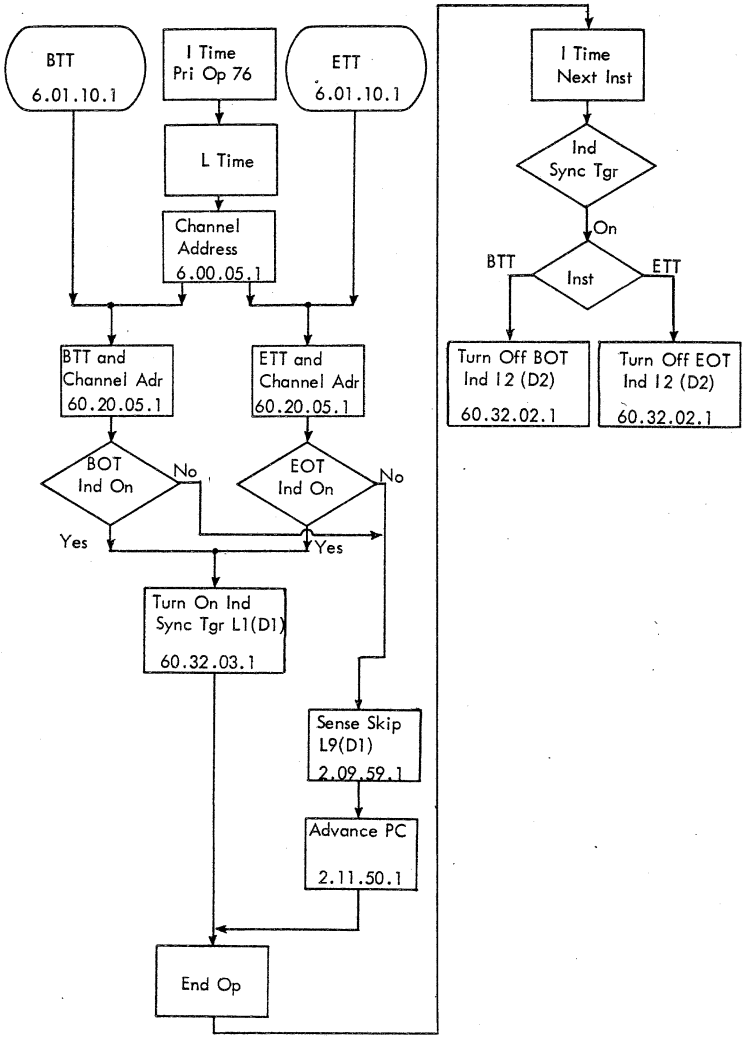


FIGURE 3.5-46. BTT +0760, ETT -0760

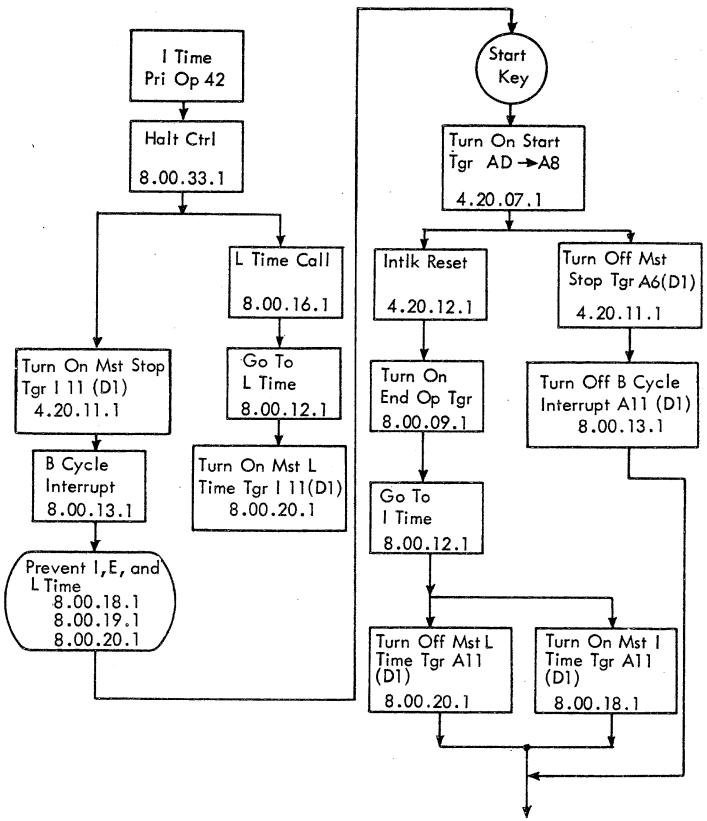


FIGURE 3.5-47. HPR +0420

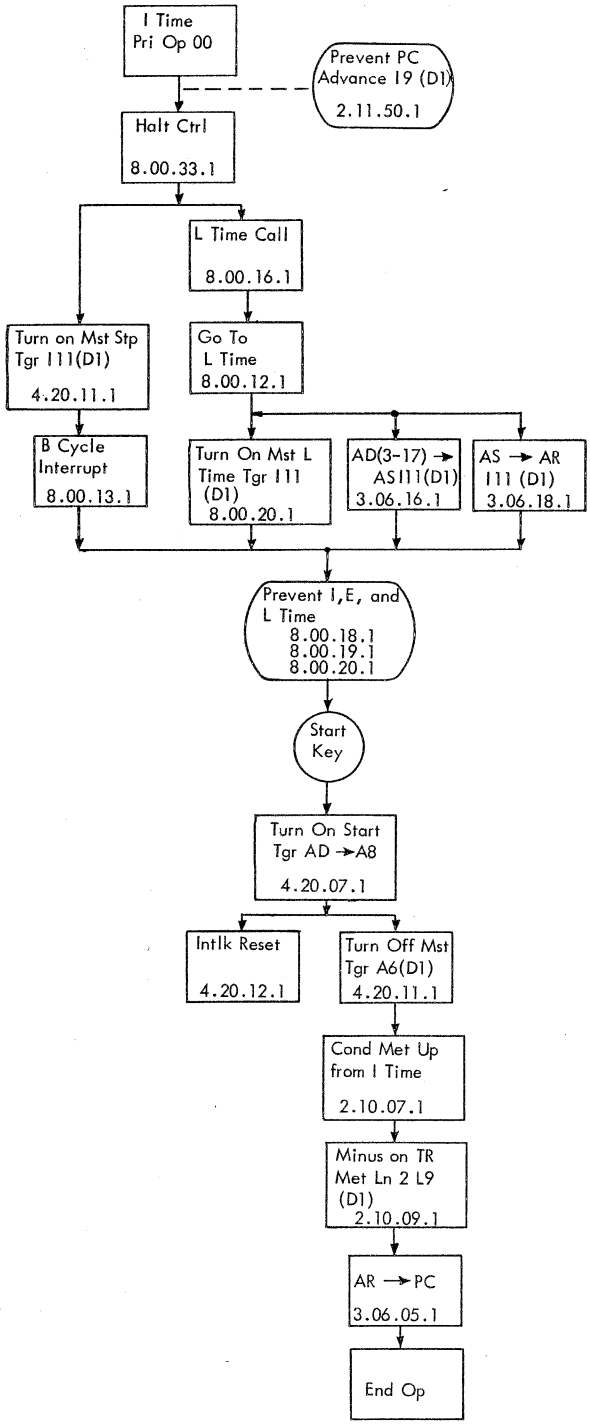


FIGURE 3.5-48. HTR + 0000

Set Sign Minus SSM -0760...0003 Figure 3.5-50

This instruction places a one (a minus) in the accumulator sign position. Positions (Q-35) of the accumulator are unchanged.

Change Sign CHS +0760...0002 Figure 3.5-51

This instruction complements the sign position of the accumulator. A one is replaced by a zero and a zero is changed to a one. Positions (Q-35) of the accumulator are unchanged.

3.5.09 Sense Indicator Instructions

The sense indicator instructions are a group of instructions which operate on the sense indicator register. These instructions enable the computer to set and test the indicators under program control.

Load Indicators LDI +0441 Figure 3.5-52

The contents of storage location X (S, 1-35) are placed in indicator positons (0-35). The contents of storage are unchanged.

Store Indicators STI +0604

The contents of indicator positions (0-35) replace the contents of storage location X. The indicators are unchanged. Execution of this instruction is the same as STO (Figure 3.5-1, Section 3.5.01) except that SI(0-35) is taken to the SR rather than AC (S, 1-35). SI(0-35) to the SR(S, 1-35) is shown on Systems 2.12.13.1.

OR Storage to Indicators OSI +0442 Figure 3.5-52

This instruction places the logical OR of the word at storage location X and the contents of the indicators in the sense indicator register. Storage is unchanged.

Invert Indicators from Storage IIS +0440 Figure 3.5-52

This instruction inverts the positions of the SI register which have corresponding "1" bits in the word at storage location X.

Reset Indicators from Storage RIS +0445 Figure 3.5-53

This instruction utilizes the word at storage location X to reset the sense indicator register position. A "1" bit in any position of the word causes the corresponding sense indicator trigger to be turned off. Indicator positions which correspond to "0" bits are unchanged. Storage is unchanged.

Set Indicators of Right Half SIR +0055 Figure 3.5-54

For this instruction, the control field (18-35) of the instruction is OR'ed with the right half of the sense indicator register. A "1" bit in either the control field or the indicator places a "1" bit in the corresponding sense indicator. Because the only

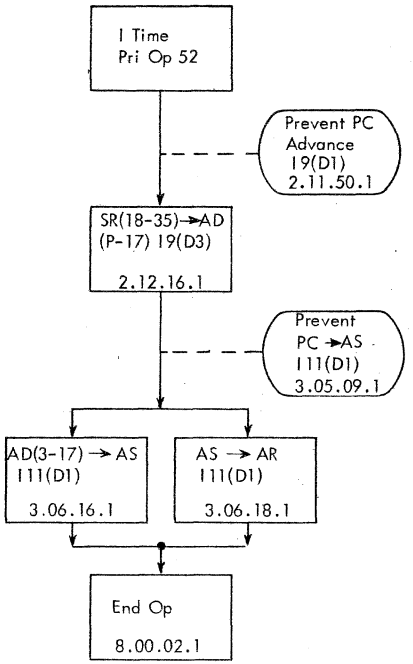


FIGURE 3.5-49. XEC + 0522

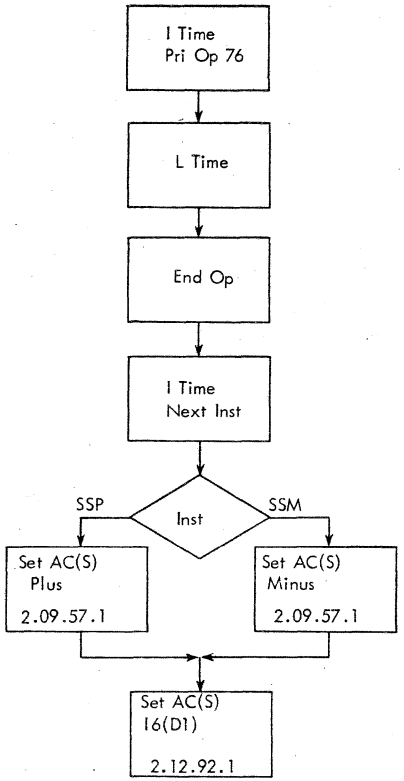


FIGURE 3.5-50. SSM - 0760...0003; SSP + 0760...0003

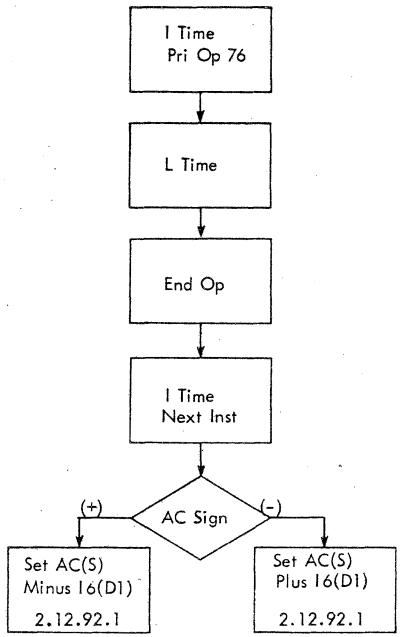


FIGURE 3.5-51. CHS +0760...0002

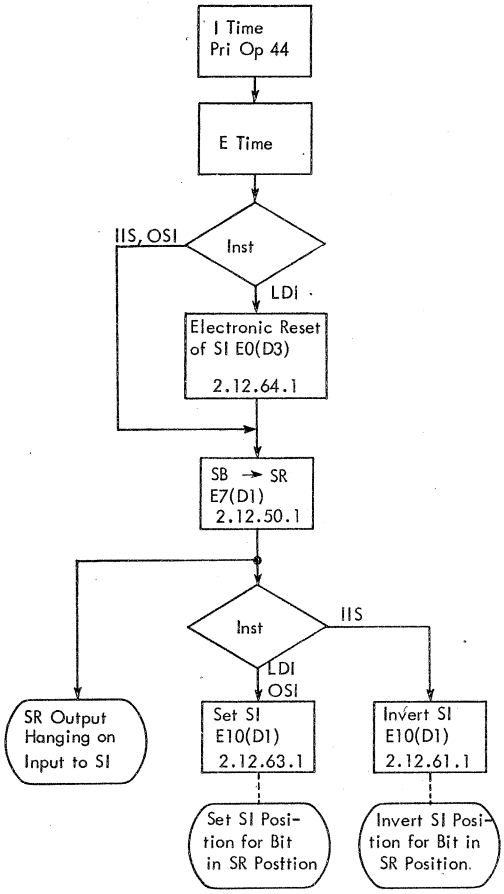


FIGURE 3.5-52. LDI +0441; OSI +0442; IIS +0440

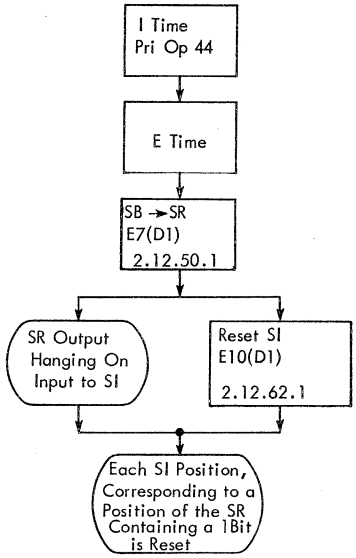


FIGURE 3.5-53. RIS +0445

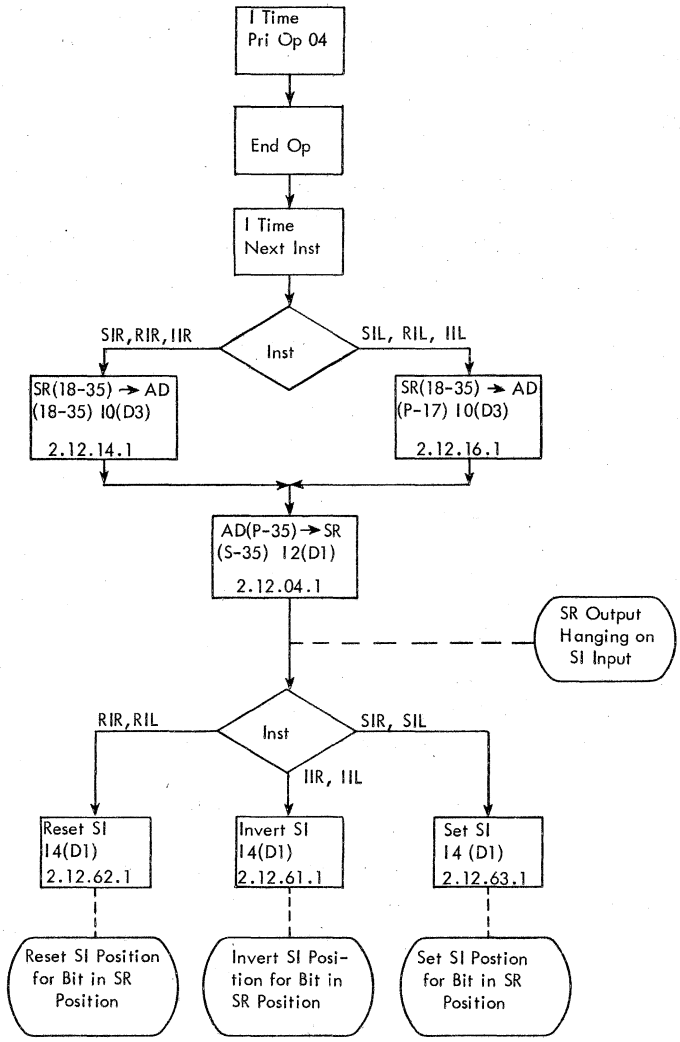


FIGURE 3.5-54 SIR + 0055; SIL - 0055; RIR + 0057;
RIL - 0057; IIR + 0051; IIL - 0051

input to the sense indicator register is from the SR, the control field must be placed in the right half of the SR, and the left half of the SR must be cleared before setting the indicators. This is accomplished by gating only the right half of the SR to the adders and then gating adders (P, 1-35) to SR(S, 1-35).

Set Indicators of Left Half SIL -0055 Figure 3.5-54

This instruction OR's the control field (18-35) of the instruction with the left half of the sense indicator register. Execution is the same as SIR except that the control field must be switched to the left half of the SR, and the right half of the SR must be cleared before the indicators can be set. This is done by routing SR(18-35) to AD(P-17).

Reset Indicators of Right Half RIR +0057 Figure 3.5-54

This instruction causes the reset of the positions of the right half of the sense indicator register which correspond to ones in the control field (18-35) of the instruction. Indicator positions corresponding to zeros are unchanged. Execution of this instruction is identical to SIR, except that the reset-indicators input is used instead of set indicators.

Reset Indicators of Left Half RIL -0057 Figure 3.5-54

For this instruction, the control field of the instruction is used as a reset mask for the left half of the indicator register. A one in the control field resets the corresponding indicator position. Execution of this instruction is similar to that of SIL except that the reset-indicators pulse is used.

Invert Indicators of Right Half IIR +0051 Figure 3.5-54

This instruction inverts positions of the right half of the indicator register which correspond to ones in the control field of the instruction. Indicator positions corresponding to zeros are unchanged. Execution is identical to that of SIR except that the invert-indicators pulse to the indicator input is used.

Invert Indicators of Left Half IIL -0051 Figure 3.5-54

This instruction inverts positions of the left half of the indicator register which correspond to ones in the control field (18-35) of the instruction. Indicator positions corresponding to zeros are unchanged. Execution of this instruction is identical to that of SIL, except that an invert-indicators pulse is used.

Place Indicator in Accumulator PIA -0046 Figure 3.5-55

The contents of sense indicators (0-35) are placed in positions (P, 1-35) of the accumulator. The sense indicators are unchanged.

Place Accumulator in Indicators PAI +0044 Figure 3.5-55

The contents of the accumulator (P, 1-35) are placed in the sense indicator register. The accumulator is unchanged.

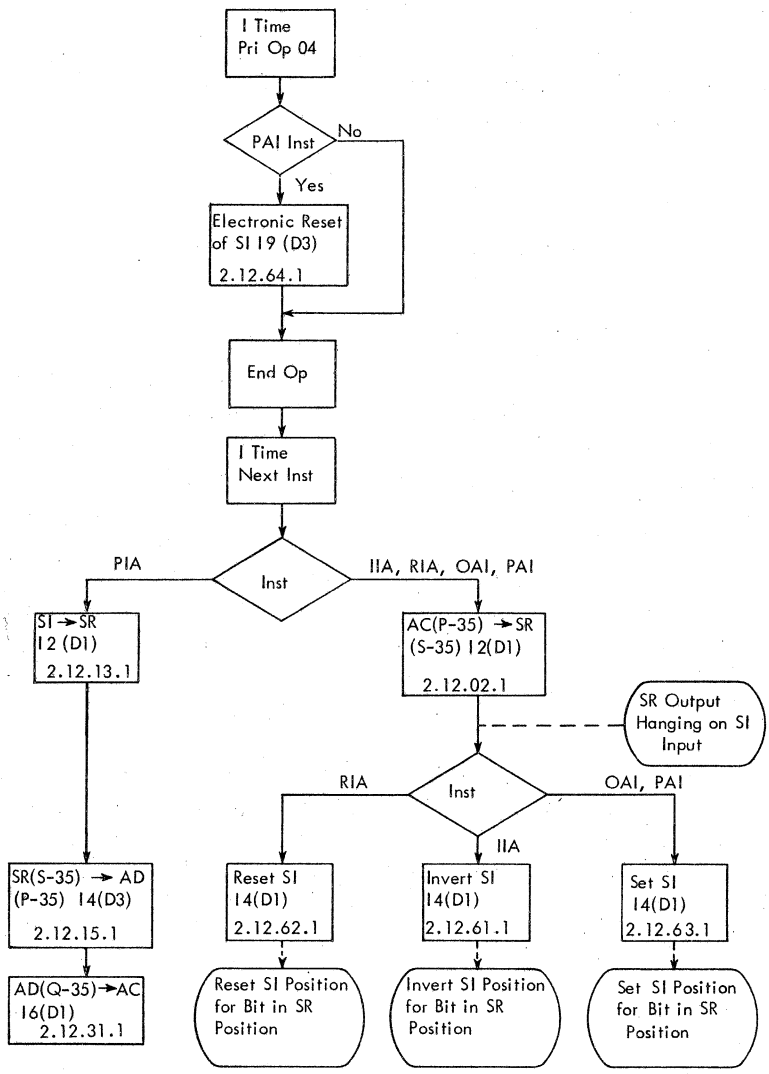


FIGURE 3.5-55. PAI +0044; PIA - 0046; OAI +0043;
RIA - 0042; ITA +0041

OR Accumulator to Indicators OAI +0043 Figure 3.5-55

The logical OR of the contents of the accumulator and the indicators is placed in the sense indicator register. If a position in either register contains a one, a one will be placed in that position of the indicator. The accumulator is unchanged. Execution is accomplished using the sequence of PAI but omitting the indicator reset.

Reset Indicators from Accumulator RIA -0042 Figure 3.5-55

The positions of the sense indicator register which correspond to the positions of the AC(P, 1-35) having "1" bits are reset to zero. Indicator positions which correspond to "0" bits are unchanged. The AC is unchanged.

Invert Indicators from Accumulator IIA +0041 Figure 3.5-55

This instruction inverts any sense indicator position for which there is a corresponding "1" bit in the accumulator.

Transfer if Indicators On TIO +0042 Figure 3.5-56

If all ones in the AC are matched by ones in the indicator register, an instruction transfer is taken to location X. AC positions containing zeros are not compared with indicator positions. If all of the ones are not matched, the computer takes the next instruction in sequence. To execute this instruction, the complement of the AC is OR'ed with the indicator. If the ones match, the result will be all ones. The result of the OR is stored in the SR and fed to the adder. A carry to adder (35) will ripple down the adder and carry out of the AC(P). The adder (P) carry is used to condition the transfer.

Transfer if Indicators Off TIF +0046 Figure 3.5-56

If all of the ones in the AC are matched by zeros in the indicator register, an instruction transfer is taken to location X. AC positions containing zeros are not compared with indicator positions. If all of the ones are not matched, the computer takes the next instruction in sequence. Execution of this instruction is identical to that of TIO, except that the complement of the indicators is OR'ed to the complement of the AC. Test procedure remains the same.

On Test for Indicators ONT +0446 Figure 3.5-57

If the ones contained in the word stored at location X are matched by ones in the corresponding indicator register positions, the next instruction will be skipped. If all of the ones are not matched, the computer will take the next instruction in sequence. Positions of storage location X which contain zeros are not compared. Execution of this instruction requires an E cycle to obtain the test word from storage and two L cycles to complete the test. The test is accomplished by moving the test word to the accumulator, where it can be complemented. The complement is OR'ed with the indicators and returned to the SR. The result of the OR will be all ones if the ones in the test word match the indicator. To test for all ones a carry is added to the OR which will result in an adder (P) carry to condition the advance instruction counter. The contents of the accumulator are saved and restored to normal during execution of the instruction.

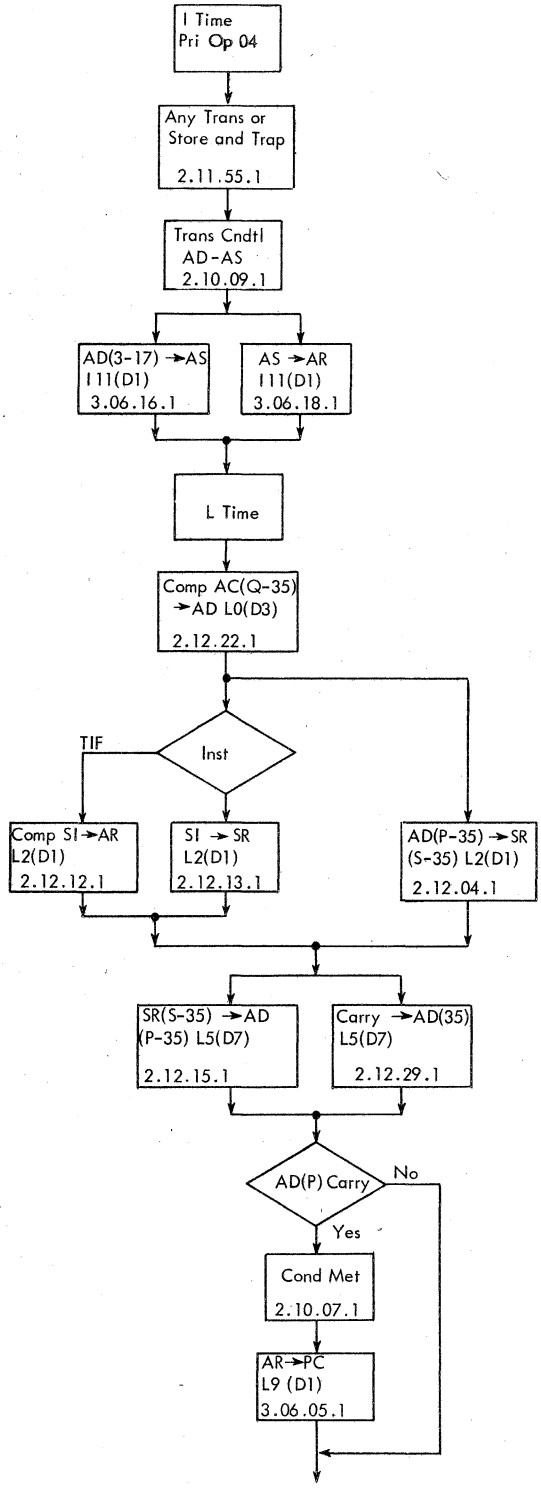


FIGURE 3.5-56. TIO + 0042; TIF + 0046

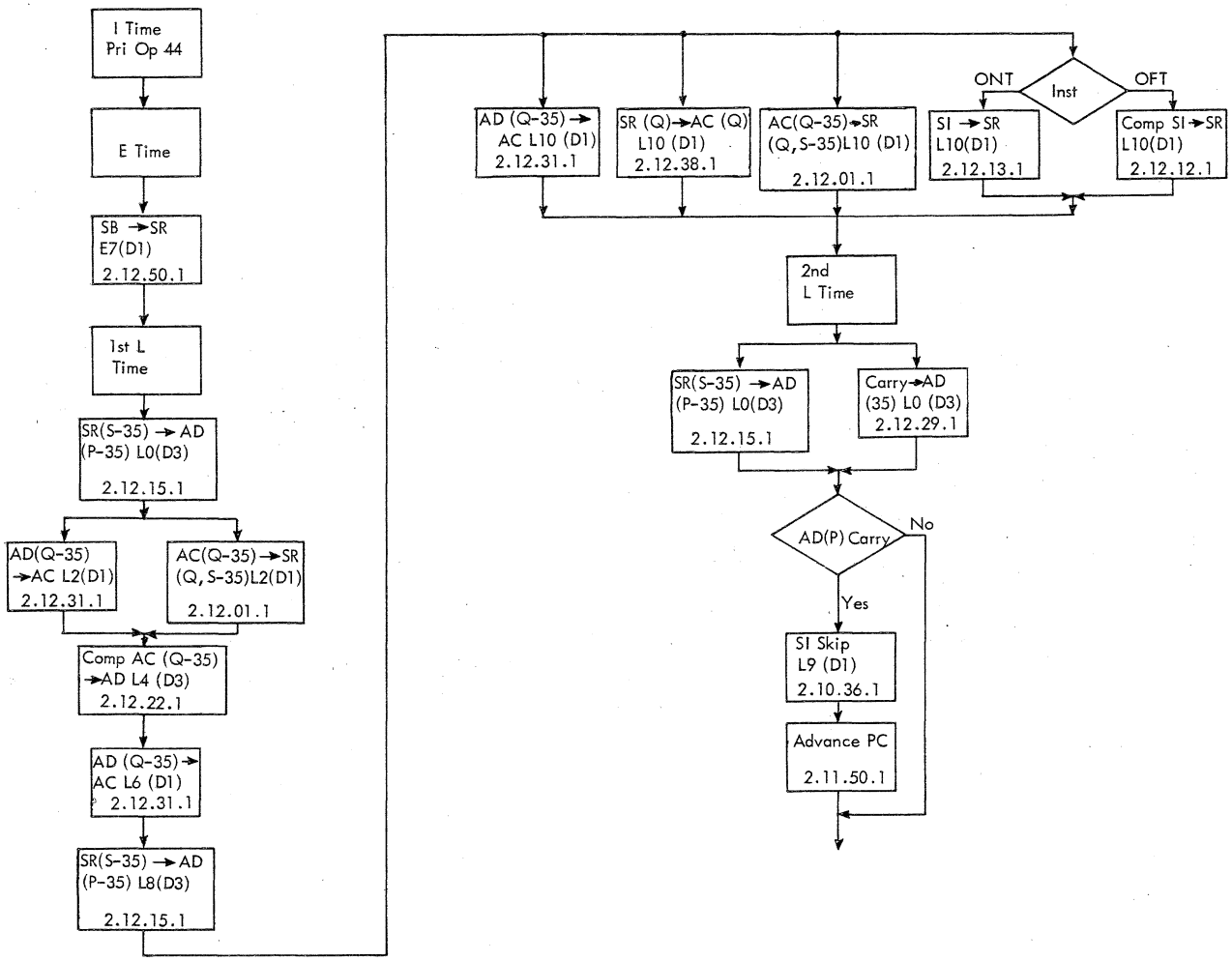


FIGURE 3.5-57. ONT +0446, OFT +0444

Off Test for Indicators OFT +0444 Figure 3.5-57

If the ones contained in the word stored at location X are matched by zeros in the corresponding indicator positions, the computer will skip one instruction. If all of the ones are not matched by zeros, the computer will take the next instruction in sequence. Positions in the test word from storage which contain zeros are not compared. Execution of this instruction is identical to that of ONT, except that the complement of the indicator is used for the OR operation instead of the true indicator.

Right-Half Indicators, On Test RNT +0056 Figure 3.5-58

This instruction matches the ones in the control field (18-35) of the instruction against the corresponding positions of the right half of the indicator register. If all of the ones are matched by ones, the computer skips one instruction. If all are not matched, the computer takes the next instruction in sequence. Positions of the control field containing zeros are not compared. Execution of this instruction is the same as ONT except that only positions (18-35) of SR are used.

Left-Half Indicators, On Test LNT -0056 Figure 3.5-58

If the ones in the control field of the instruction are matched by ones in the corresponding positonis of the left half of the indicator register, the computer will skip the next instruction. If all ones are not matched, the computer takes the next instruction in sequence. Execution of this instruction is the same as ONT, except that the SR (18-35) is compared against SI(0-17).

Right-Half Indicators, Off Test RFT +0054 Figure 3.5-58

If the ones in the control field of this instruction are matched by zeros in the right half of the indicator register, the computer will skip one instruction. If all ones are not matched by zeros, the computer will take the next instruction in sequence. Positions of the control field containing zeros are not compared. This instruction is executed the same as OFT, except that only positions (18-35) of the SR are used.

Left-Half Indicators, Off Test LFT -0054 Figure 3.5-58

If the ones in the control field of this instruction are matched by zeros in the left half of the indicator register, the computer will skip one instruction. If all are not matched, the computer will take the next instruction in sequence. Positions of the control field containing zeros are not compared. The control field and the indicator register are unchanged. Execution of this instruction is the same as OFT, except that SR(18-35) is compared with SI(0-17).

3.5.10 Index Transmission Instructions

These are the instructions which operate on or with the index registers. They are used to load or modify the index registers, to alter instructions from the index registers, or to control program flow.

There are three shift cell index registers containing 15 positions each. The tag bit positions (18-20) of the instruction control which index registers are to be used. Any

index register or combination of index registers may be selected. The tag bits are retained by the tag register on Systems 2.08.01.1. SB(18-20) outputs are used to turn on the tag register triggers during I time of the instruction. The output of the index register, when gated to the adders, is always complemented. To make this a 2's complement, a carry is gated to adder (17) at the same time the XR is gated to the AD.

Many instructions are indexable, as controlled by their tag positions, allowing the address portion of the instruction to be modified. Appendix A shows which instructions are indexable.

Transfer with Index Incremented TXI +1000 Figure 3.5-59

This transfer adds its decrement to the specified index register and causes an unconditional transfer to location X. The index register outputs are always complemented to the adders. Thus, to add the decrement, the index register must be cycled through the adders so they contain a complement before the addition.

Transfer on Index TIX +2000 Figure 3.5-60

If the number in the specified index register is greater than the decrement, the contents of the index register will be reduced by the amount of the decrement, and an instruction transfer will be taken to storage location X. When the number in the index register is equal to or less than the decrement, no reduction is made and the computer takes the next instruction in sequence. The comparison between the index register contents and the decrement is made by gating the decrement and the 2's complement of the index register contents to the adders. No carry from adder (3) indicates that the index register contents are greater and that the transfer and index register reduction are to be made. A carry from adder (3) will, therefore, block the transfer and index register reduction.

Transfer on No Index TNX -2000 Figure 3.5-60

If the number in the specified index register is greater than the decrement, the contents of the index register will be reduced by the amount of the decrement, and the computer will proceed to the next instruction in sequence. When the number in the index register is equal to or less than the decrement, no reduction is made, but an instruction transfer will be taken to storage location X.

The sequence of operations for this instruction is like that of TIX, except that the conditional transfer circuits are activated with the adder (3) carry trigger on.

Transfer on Index High TXH +3000 Figure 3.5-60

If the number in the specified index register is greater than the decrement, an instruction transfer is taken to storage location X. If the number in the index register is less than or equal to the decrement, the computer takes the next instruction in sequence. Execution of this instruction is identical to that of TIX except that the difference in the adders is not routed to the index register, because no reduction is to be made.

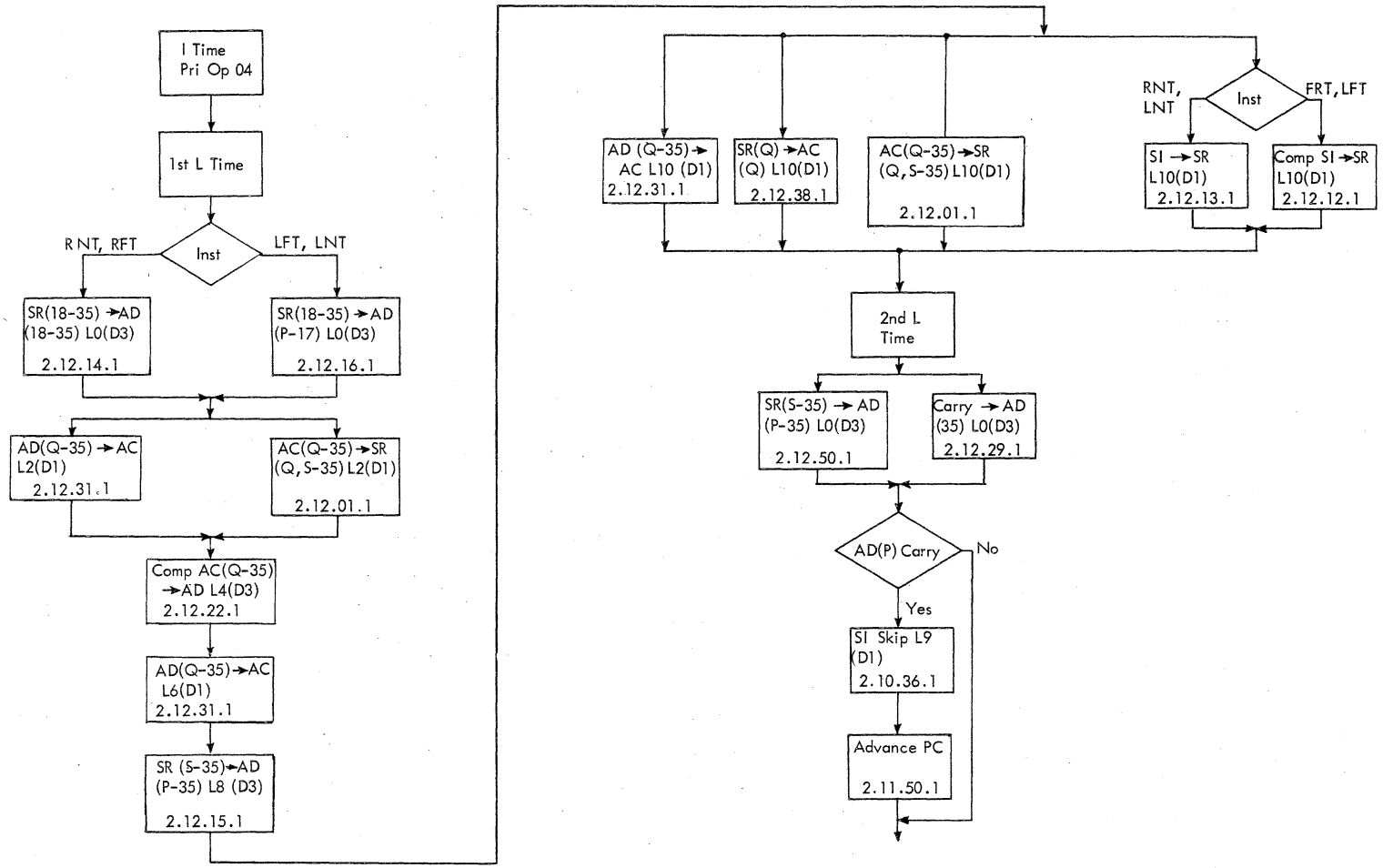


FIGURE 3.5-58. RNT +0056; LNT - 0056; RFT +0054; LFT - 0054

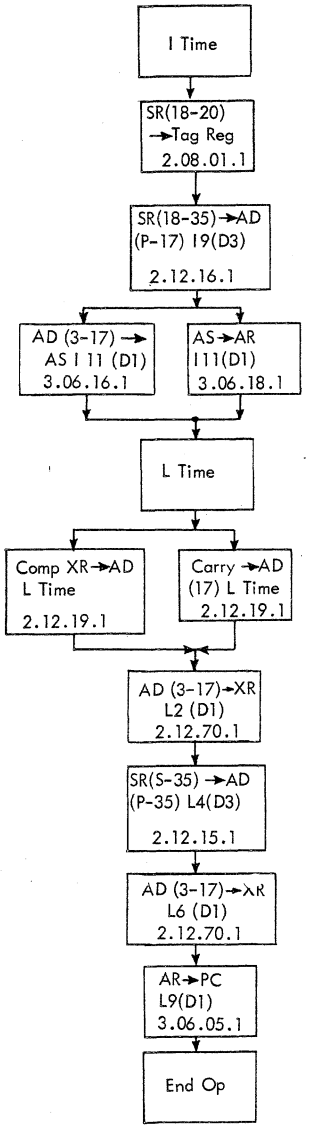


FIGURE 3.5-59. TXI +1000

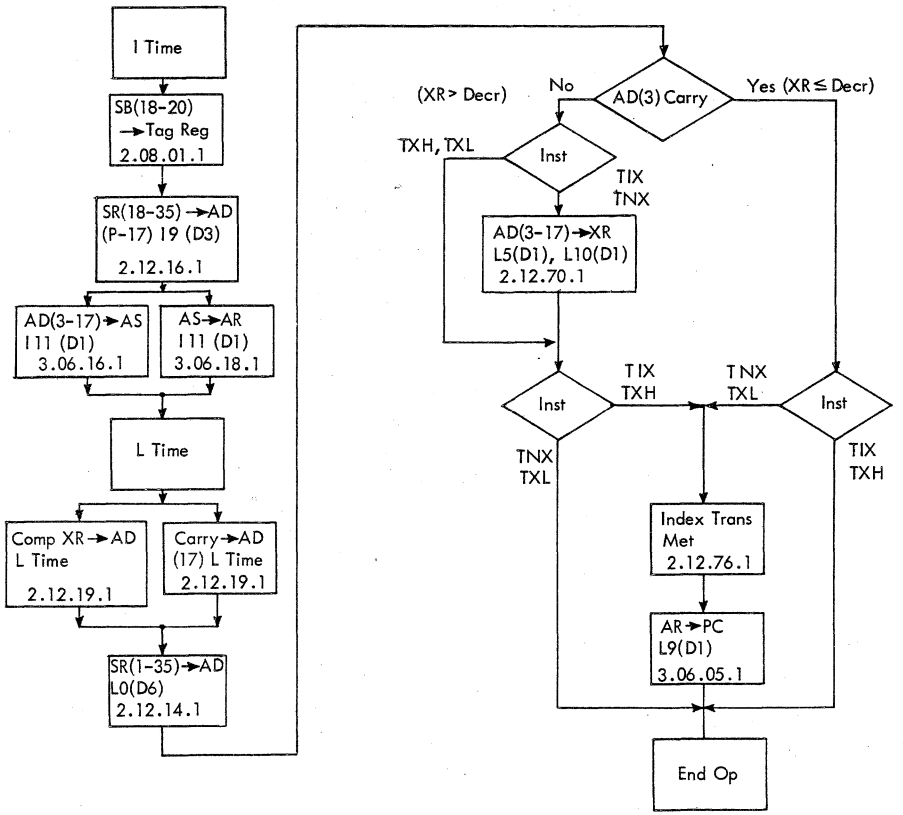


FIGURE 3.5-60. TIX +2000; TNX - 2000; TXH +3000; TXL - 3000

Transfer on Index Low or Equal TXL -3000 Figure 3.5-60

If the number in the specified index register is greater than the decrement, the computer takes the next instruction in sequence. If the number in the index register is equal to or less than the decrement, an instruction transfer will be taken to storage location X. Execution of this instruction is identical to that of TNX except that, because no reduction is required, the adders will not be routed to the index register.

Transfer and Set Index TSX +0074 Figure 3.5-61

This instruction places the 2's complement of the instruction counter (the location of the TSX instruction) in the specified index register and causes an instruction transfer to storage location X. Execution of this instruction requires, in addition to the normal transfer controls, the routing of the instruction counter to the index register. This involves use of the address switch, storage register, and adders. To obtain the required 2's complement, the index register contents must also be cycled through the adders after receiving the contents of the instruction counter.

Place Address in Index PAX +0734 Figure 3.5-62

This instruction places the contents of AC(21-35) in the specified index register in true form. To shift the address field to the decrement positions of the adders, AC (21-35) is routed through the SR to AD(P-17).

Place Decrement in Index PDX -0734 Figure 3.5-62

This instruction places the true contents of AC(3-17) in the specified index register. Because no shift is required to execute this instruction, AC(3-17) is sent to AD(3-17) through the SR.

Place Complement of Address in Index PAC +0737 Figure 3.5-62

This instruction places the 2's complement of AC(21-35) in the specified index register. The AC is unchanged. The complementing is accomplished by putting the address in the XR and then cycling the XR through the adders.

Place Complement of Decrement in Index PDC -0737 Figure 3.5-62

This instruction loads the 2's complement of AC(3-17) in the specified index register. The AC is unchanged. The complementing is accomplished in the same manner as in PAC.

Place Index in Address PXA +0754 Figure 3.5-63

This instruction places the true contents of the specified index register in AC(21-35). Positions (S, Q, P-20) of the AC are cleared. To obtain the true contents of the index register for executing this instruction, the index register contents are cycled through the adders. The index register contents are then rerouted to the adders and routed to the storage register by way of the address switch to shift the index register contents to the address field.

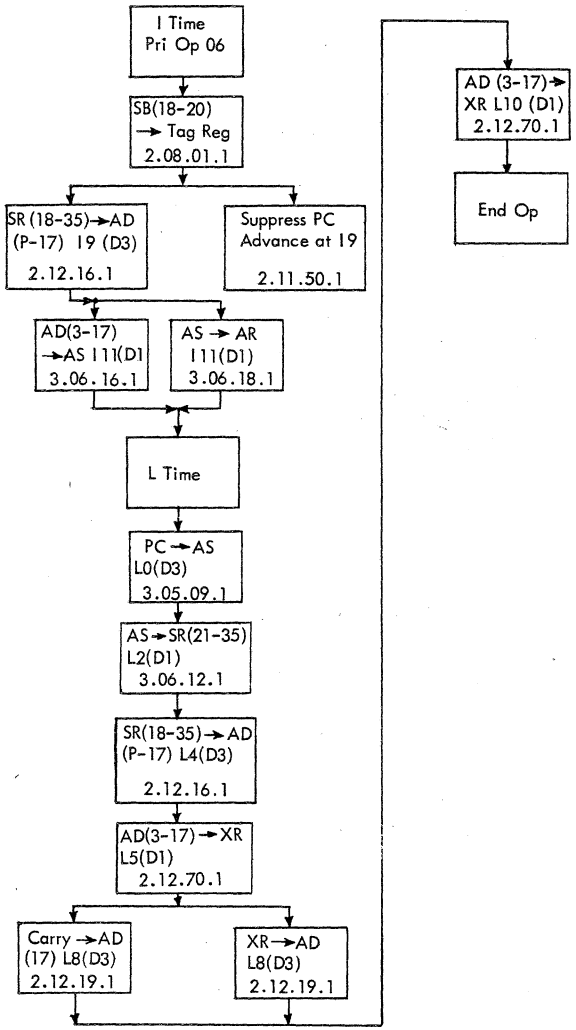


FIGURE 3.5-61. TSX +0074

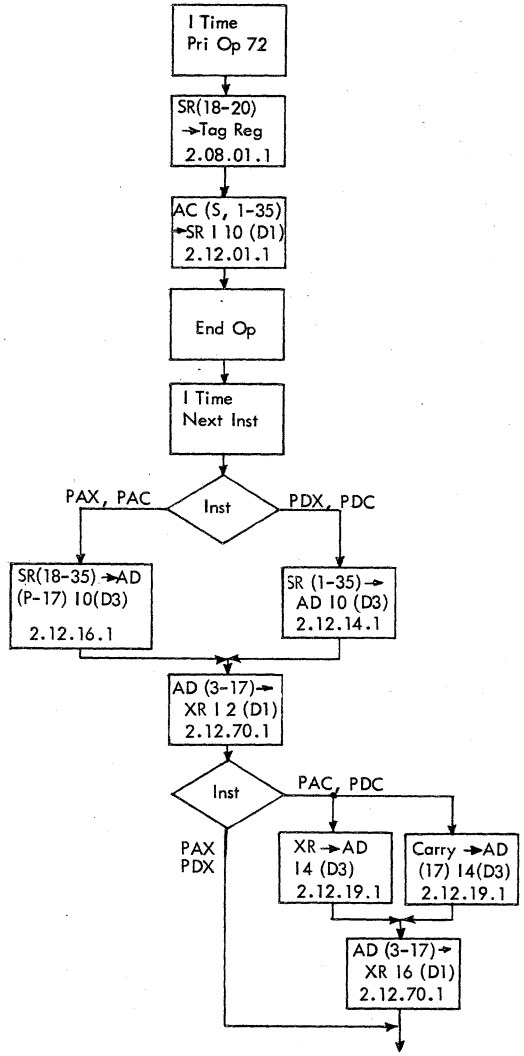


FIGURE 3.5-62. PAX +0734; PDX -0734; PAC +0737; PDC -0737

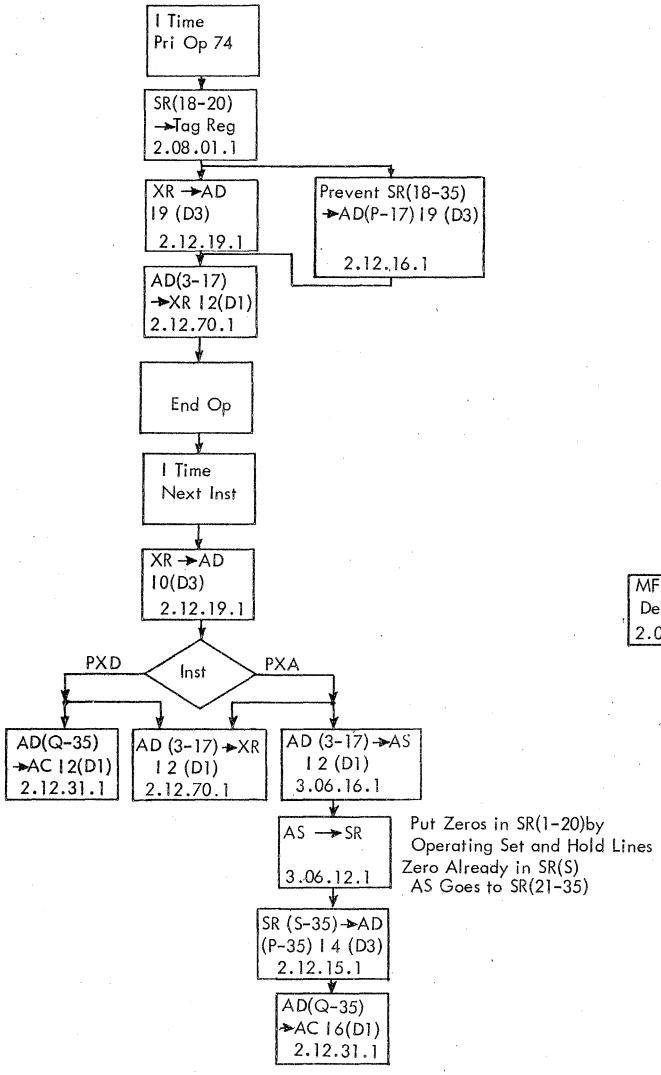


FIGURE 3.5-63. PXA +0754; PXD - 0754

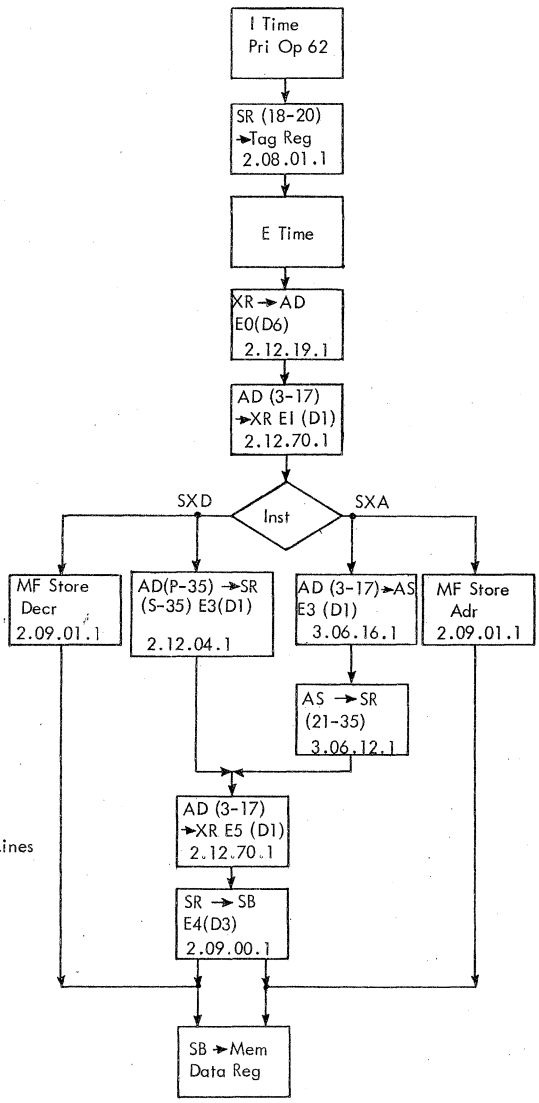


FIGURE 3.5-64. SXA +0634; SXD - 0634

Place Index in Decrement PXD -0754 Figure 3.5-63

This instruction places the true contents of the specified index register in positions (3-17) of the AC. The remainder of the AC is cleared. Execution of this instruction is similar to PXA. However, because no shift is required, the index register contents can be routed directly from the adders to the AC.

Store Index in Address SXA +0634 Figure 3.5-64

This instruction stores the true contents of the specified index register in the address field (21-35) of storage location X. The remaining positions of location X are unchanged. The logic of the execution of this instruction is similar to PXA. However, an E cycle is needed to put the information in core storage.

Store Index in Decrement SXD -0634 Figure 3.5-64

This instruction stores the true contents of the specified index register in the decrement (3-17) of storage location X. The remaining positions of location X are unchanged. Execution of this instruction is similar to SXA but, because there is no shift required, the output of the adders goes directly to the SR.

Address to Index True AXT +0774 Figure 3.5-65

This instruction loads the specified index register with the address field (21-35) in true form.

Address to Index Complemented AXC -0774 Figure 3.5-65

This instruction loads the specified index register with the 2's complement of the address portion (21-35) of the instruction. Execution of AXC is similar to that of AXT except that the index register contents must be cycled through the adders during the following I cycle to obtain the 2's complement.

Load Complement of Address in Index LAC +0535 Figure 3.5-66

This instruction loads the 2's complement of the address (21-35) of storage location X into the specified index register. Execution of this instruction requires an E cycle to obtain the word from storage. The address portion is transferred to the index register by way of the adders at the end of the cycle,and, to obtain the 2's complement, the index register is cycled during the following I cycle.

Load Complement of Decrement in Index LDC -0535 Figure 3.5-66

This instruction loads the 2's complement of the decrement field (3-17) of storage location X into the specified index register. Execution of this instruction is identical to that of LAC except that the SR decrement, instead of the address, goes to the adders.

Load Index from Address LXA +0534 Figure 3.5-66

This instruction loads the specified index register with the contents of the address field (21-35) of storage location X in true form. Execution of this instruction is identical to that of LAC, except that the complementing operation is not required.

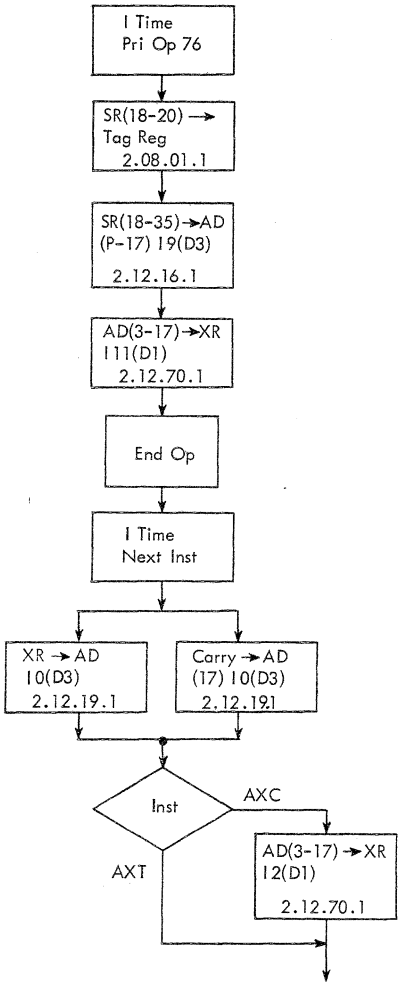


FIGURE 3.5-65. AXT +0774; AXC - 0774

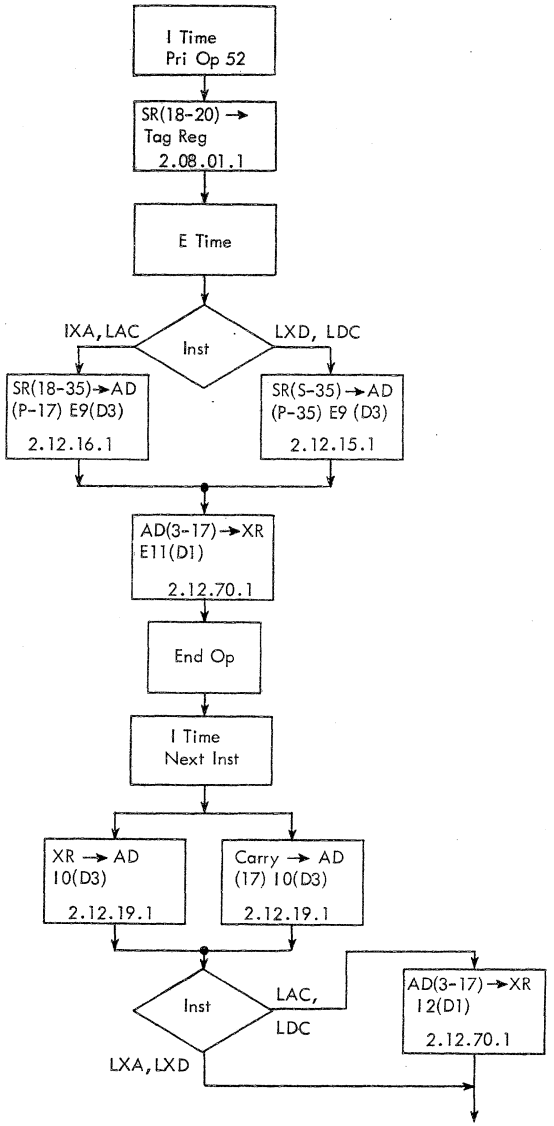


FIGURE 3.5-66. LXA +0543; LXD - 0534; LAC +0535; LDC - 0535

Load Index from Decrement LXD -0534 Figure 3.5-66

This instruction loads the specified index register with the contents of the decrement field (3-17) of storage location X in true form. Execution of LXD is identical to that of LDC, except that the complementing operation is not used.

3.5.11 AND and OR Instructions

The AND and OR instructions are used to produce logical combinations of bits. These are useful for masking or matching words.

OR to Storage ORS -0602 Figure 3.5-67

This instruction stores the logical OR of the word stored at location X and the AC (P, 1-35) in location X. The logical OR is obtained by matching the two words and storing ones in all positions which have ones in either word. For this instruction, the OR is developed in the memory data register during the E cycle. The words from core storage and the SB both go the memory data register and the result in the memory data register is put back into core storage.

OR to Accumulator ORA -0501 Figure 3.5-68

This instruction places the logical OR of the word stored at location X and AC(P-35) in the accumulator. The OR is produced by matching the two words and placing ones in all positions of the AC which have ones in either word. The OR is developed in the SR by gating both the storage and AC words into the SR at the same time.

AND to Accumulator ANA -0320 Figure 3.5-69

This instruction places the logical AND of the word stored at location X and the AC (P, 1-35) in the accumulator (P, 1-35). The logical AND is obtained by matching the two words and placing ones in all positions of storage location X which have corresponding ones in both the AC and storage. This instruction is executed by adding the complement of both words and then complementing this sum.

AND to Storage ANS +0320 Figure 3.5-69

This instruction stores the logical AND of the word stored at location X and the contents of the AC(P, 1-35) in location X. The logical AND is obtained by matching the two words and placing ones in all positions of storage location X which have corresponding ones in both the AC and storage. The contents of the AC(S, Q, P, 1-35) are unchanged. The AND is accomplished the same for this instruction as for ANA. An additional E cycle is required to put the AND in storage. The complement of the original AC (P, 1-35) is returned to the AC and re-complemented to restore the contents of AC(P, 1-35). During the execution, the original AC(Q) is saved in SR(Q) and then returned to AC(Q).

Exclusive OR to Accumulator ERA +0322 Figure 3.5-70

This instruction matches AC(P, 1-35) with the logical word stored at location X. Zeros replace all positions of the AC which match the corresponding positions of the logical word. Accumulator positions (S) and (Q) are cleared.

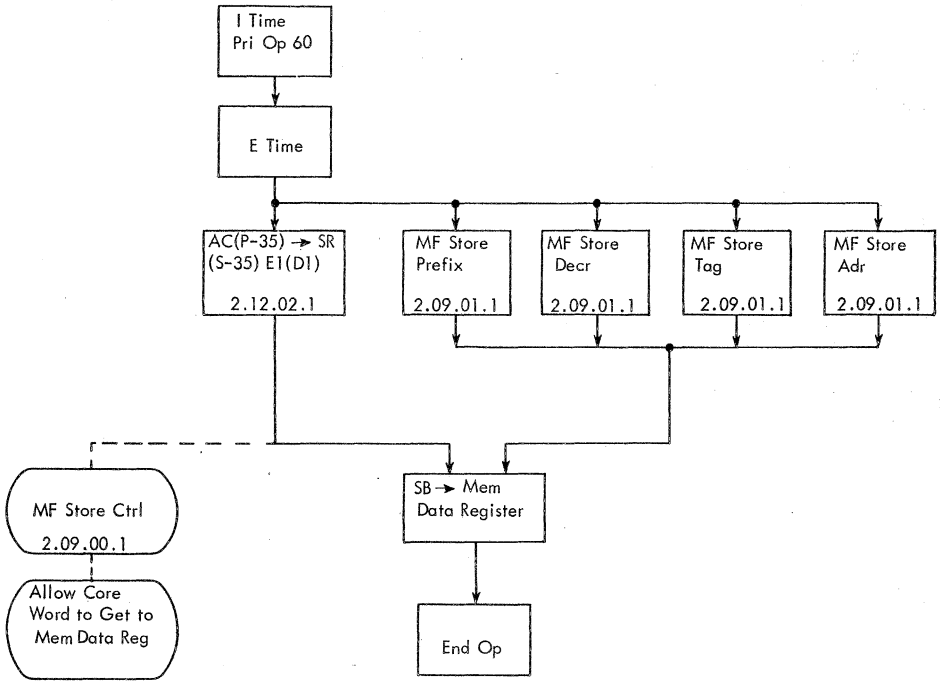


FIGURE 3.5-67. ORS - 0602

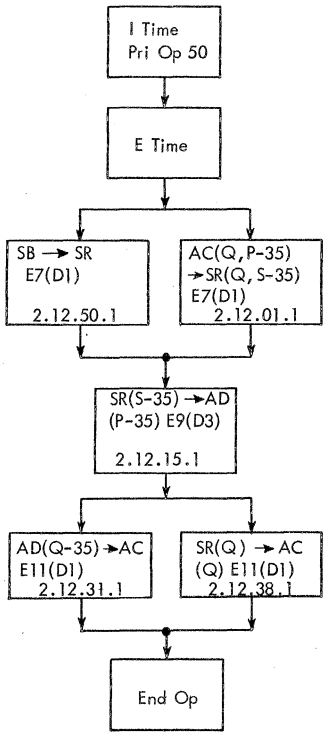


FIGURE 3.5-68. ORA - 0501

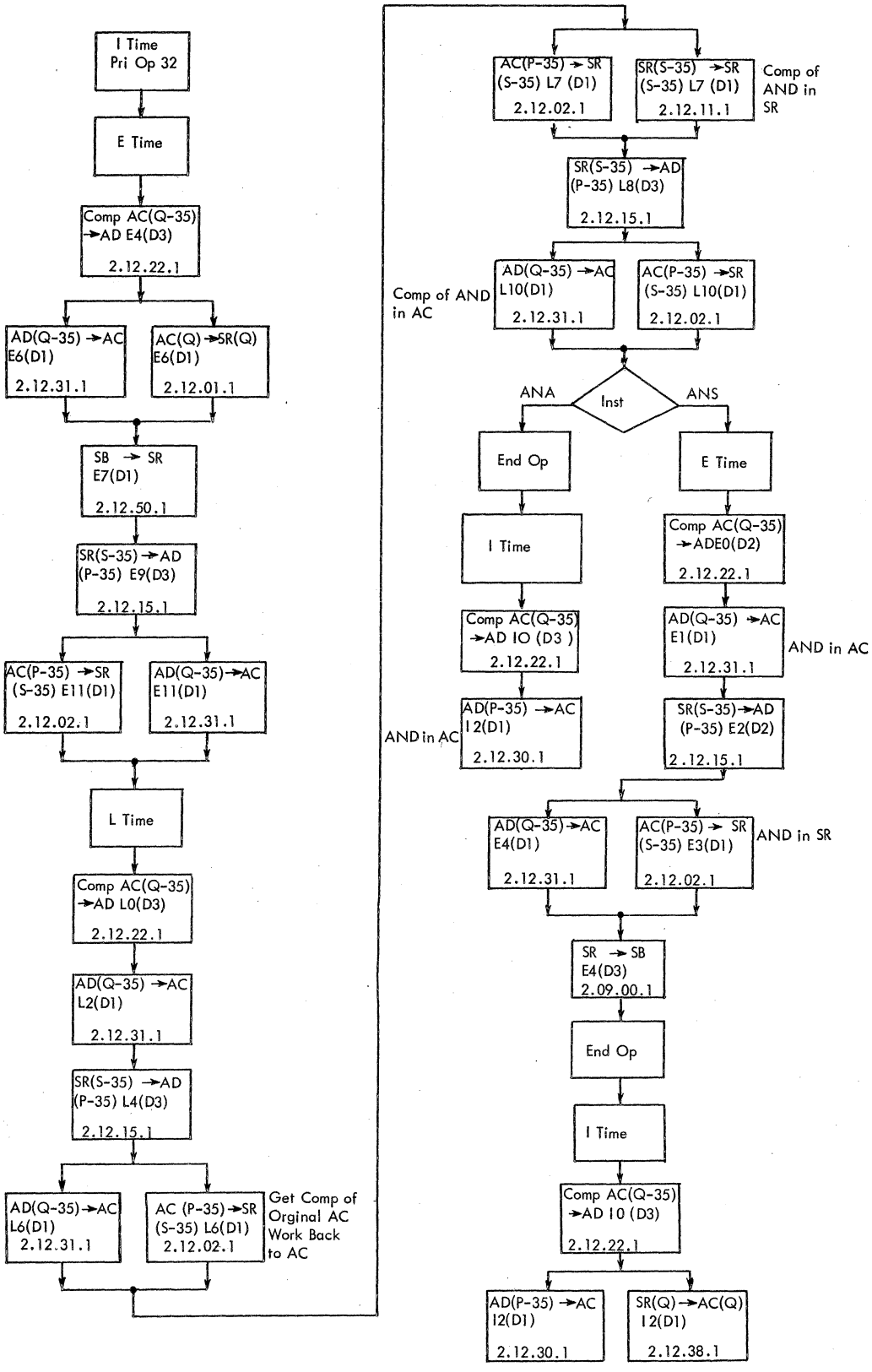


FIGURE 3.5-69. ANA - 0320; ANS +0320

Logically speaking, the EXCLUSIVE OR is the result of OR'ing bits from either one word or the other, but not both. The computer, which can only AND and OR, makes use of the following logical equation to develop the EXCLUSIVE OR:

EXCLUSIVE OR = 2 (A or B) - (A + B)

The derivation of the above equation can be shown with the following adder table:

Factor A	0	0	0	1	1
Factor B	0	0	1	0	1
A + B (carry not blocked)	0	1	0	0	0
Carry to be blocked	0	0	0	1	0
A + B (carry blocked)	0	0	1	1	0
EXCLUSIVE OR (A + B carry blocked)	0	0	1	1	0
A or B	0	0	1	1	1
2 (A + B)	1	0	0	0	0
2 (A or B)	0	1	1	1	0

From the table it can be seen that the EXCLUSIVE OR is equal to the sum output or the adders with all carries blocked. The carries cannot be blocked, but the EXCLUSIVE OR can be obtained by subtracting an amount equal to the blocked carries from the sum of two words:

1. EXCLUSIVE OR = (A + B) - blocked carries
   The blocked carries can be simulated by subtracting twice the OR from twice the sum of two words:
2. Blocked carries = 2 (A + B) - 2 (A or B) = 10000 - 01110 = 00010
   Substituting the equation 2 in equation 1:
3. EXCLUSIVE OR = (A + B) - [2 (A + B) - 2 (A or B)] = 01000 - 00010 = 00110
   And simplifying equation 3:
4. EXCLUSIVE OR = 2 (A or B) - (A + B) = 01110 - 01000 = 00110

3.5.12 Convert Instructions

The three convert instructions can materially reduce the time required for many "housekeeping" and table-look-up routines. They can be used for number conversions, for preparing print fields, and even for adding numbers in systems other than binary.

The convert instructions, like variable-length instructions, include a count field as well as the operation code, address and tag bit. The convert instructions cause a series of references to be made (maximum of six) and the address specifies the starting location of the first storage table. The register (accumulator or MQ) from which the reference is controlled is considered to be made up of six 6-bit groups. The first of these groups is added to the instruction address to give the location of the first storage reference. The word stored at this location must contain, in addition to its conversion information, the starting location of the next storage table. The convert-by-replacement instructions shift the controlling register six places, clearing the six places on the

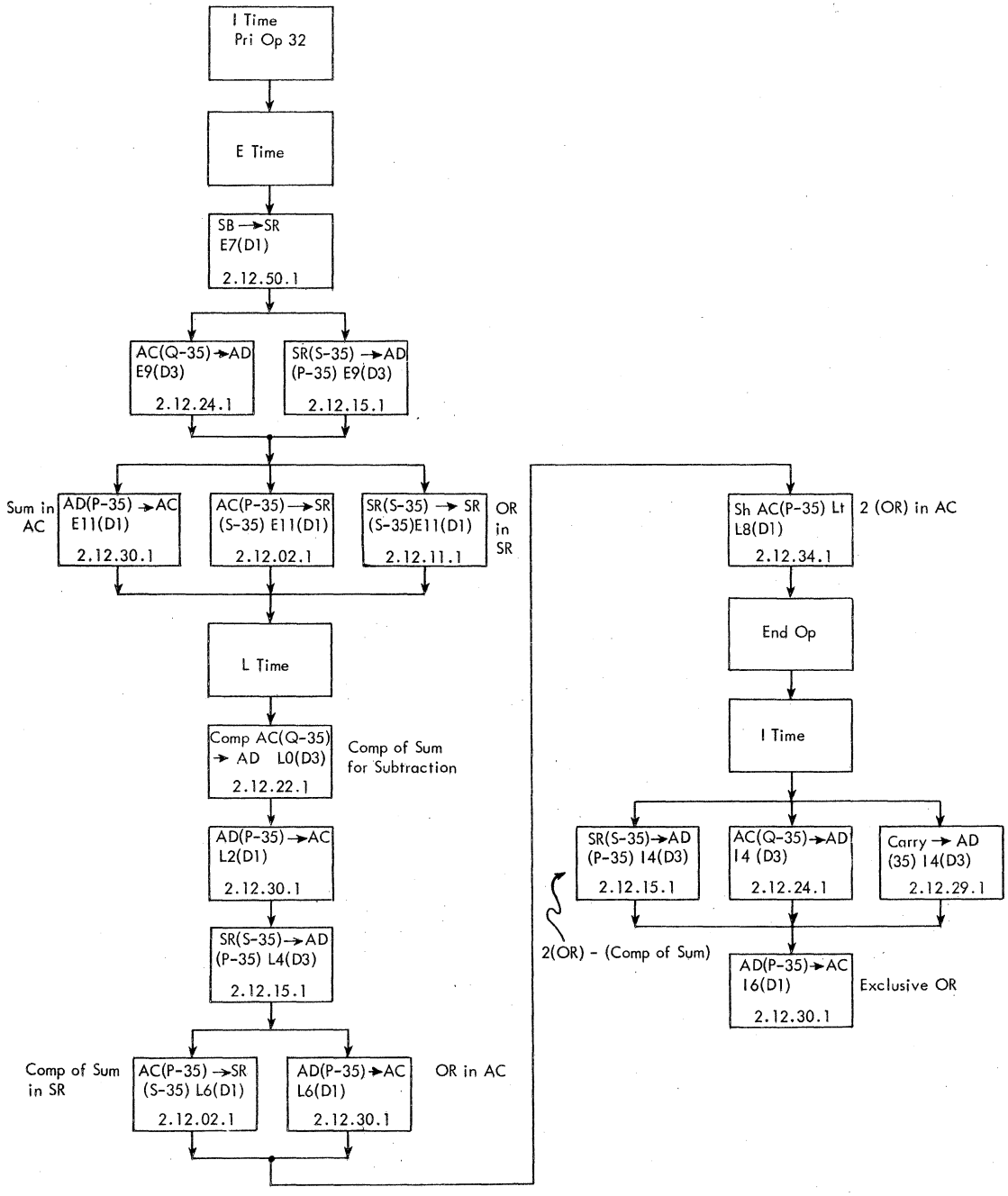


FIGURE 3.5-70. ERA +0322

opposite end of the register. Positions(S-5)* of the table word are entered into the six cleared positions, and the next group of six bits is in position to add to the starting location of the next table. The process continues until up to six references have been made. The controlling register is gradually replaced by six-bit entries from the storage tables. When the number of references specified by the count has been made, the conversion is complete.

The tag of the instruction has a unique function for the convert instructions. Positions (18) and (19) are not used, but a bit in (20) will cause the storage table starting location contained in the last reference word to be stored in index register A.

Convert by Replacement from Accumulator CVR +0114 Figure 3.5-71

This instruction treats the contents of the AC(P, 1-35) as six 6-bit representations. The instruction replaces a number of these representations, equal to the count of the instruction, with the contents of positions (S-5) of a like number of words from storage. These words are found by adding AC(30-35) (the first six-bit group) to SR(21-35) (initially the instruction address) and directing storage to this modified address. The word thus found is brought to the SR; the AC is shifted right six places; SR(S-5) replaces AC(P, 1-5). SR(21-35) adds to the next six-bit group in AC(30-35) to locate the next word in storage. The process is then repeated. After the required number of replacements, the address portion of the last storage word can be stored in index register A by including a "tag" bit in position (20) of the instruction.

The following illustrates the use and operation of CVR:

Direct Addition of BCD Numbers:

A	+	B	=	C
134589	+	691593	=	826182

Table required in storage for this example:

Storage Location	Contents	
	(S-5)	(21-35)
1000	0	1000
1001	1	1000
1002	2	1000
1003	3	1000
1004	4	1000
1005	5	1000
1006	6	1000
1007	7	1000
1008	8	1000
1009	9	1000

Storage Location	Contents	
	(S-5)	(21-35)
1010	0	1001
1011	1	1001
1012	2	1001
1013	3	1001
1014	4	1001
1015	5	1001
1016	6	1001
1017	7	1001
1018	8	1001
1019	9	1001

Instructions required for operation:

```
CAL  A              (First BCD word)
ADD  B              (Second BCD word)
CVR  6, 0, 1000     (Convert sum to BCD word)
```

* This text section uses (S-5) to represent (S, 1-5).

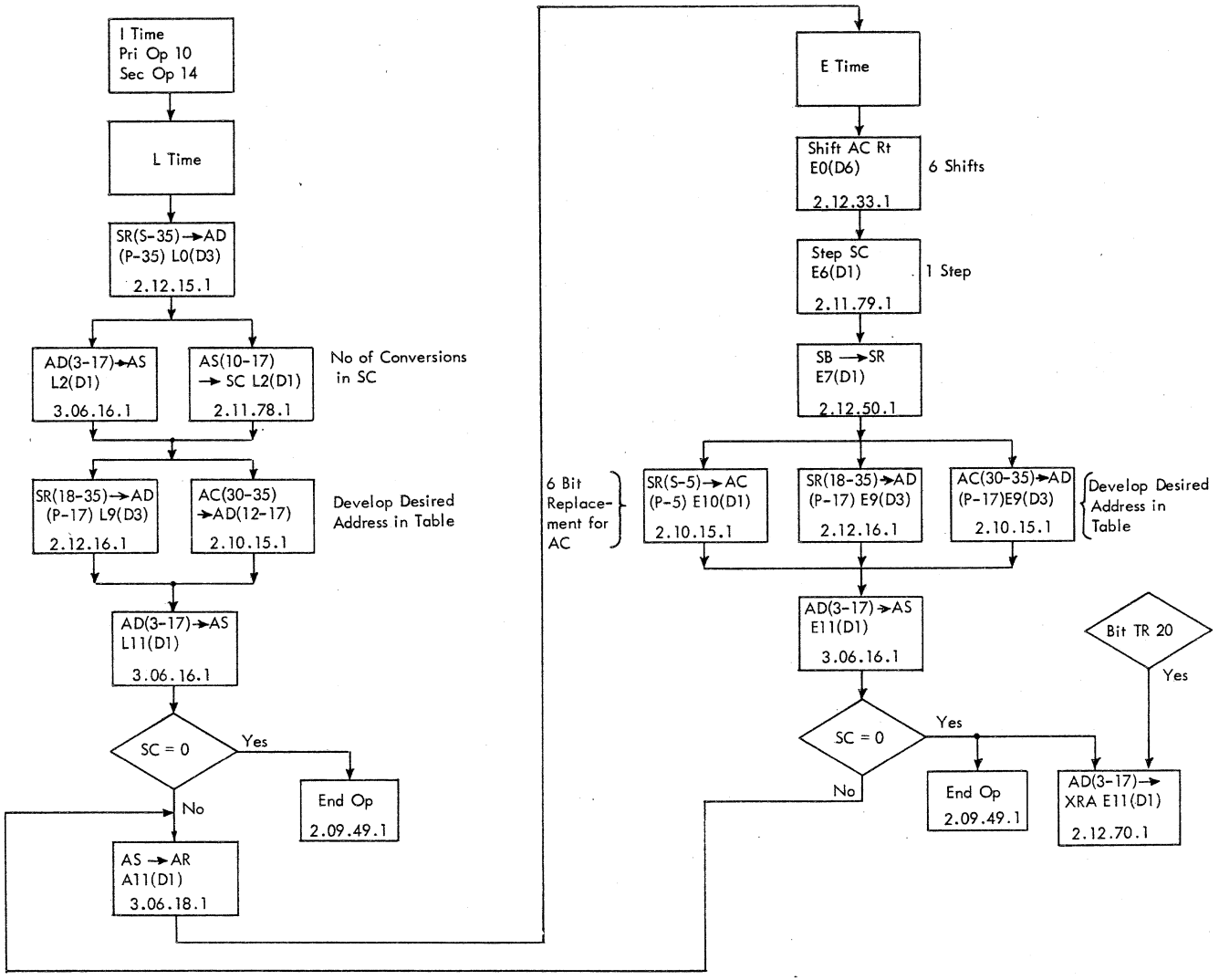


FIGURE 3.5-71. CVR +0114

Development of Program

CAL A(Clear AC, add logical word A)
Add B
(Unconverted binary sum)
CVR 1000, 0, 6

Series of Steps within CVR

C(AC) 30-35		Start Loc		Table Refer	Table C(-) 1-5	Next Start Loc C(-) 21-35
*12	+	1000	- 1012	2	1001	
17	+	1001	- 1018	8	1001	
10	+	1001	- 1011	1	1001	
5	+	1001	- 1006	6	1000	
12	+	1000	- 1012	2	1001	
7	+	1001	- 1008	8	1000	

CONTENTS OF ACCUMULATOR						
P1-5	6-11	12-17	-23	-29	-35	
1	3	4	5	8	9	1st Char
+6	9	1	5	9	3	
=7	12	5	10	17	12	Conv
						ARS 6
2	7	12	5	10	17	Count = 6
8	2	7	12	5	10	= 5
1	8	2	7	12	5	= 4
6	1	8	2	7	12	= 3
2	6	1	8	2	7	= 2
8	2	6	1	8	2	= 1

Count = zero: if Tag = 1, 1000 XRA
SLW C (Store converted BCD sum as logical word)

The execution of CVR requires one L cycle to set the count in the shift counter and to calculate the initial table location. The instruction is completed in as many E cycles as specified by the count.

Convert by Replacement from MQ CRQ -0154 Figure 3.5-72

This instruction operates on the MQ, considered to be composed of six 6-bit representations. The instruction replaces a number of these representations equal to the count of the instruction, with the contents of positions (S-5) of a like number of words from storage. The location of the first of these storage words is found by adding the contents of MQ(S-5) to SR(21-35) (initially the instruction address). The word stored at this modified location is brought to the SR, and the MQ is shifted left six places. Positions (S-5) of the stored word are placed in MQ(30-35), and the location of the next storage word is computed by again adding MQ(S-5) to SR(21-35) which is now the address portion of the previous storage word. The process continues until the required number of replacements have been made. At this time the presence of a tag bit in position 20 of the instruction will cause the address portion of the final storage word to be stored in index register A.

The following illustrates the use and operation of CRQ:

Prepare a BCD number for printing be replacing leading zeros with blanks:

Convert BCD number 007109 to BL BL 7109
Instructions required for this operation:

```
LDQ       A      (BCD number in storage)
CRQ 6,0, 2000    (Convert from MQ)
STQ       B      (Store converted number)
```

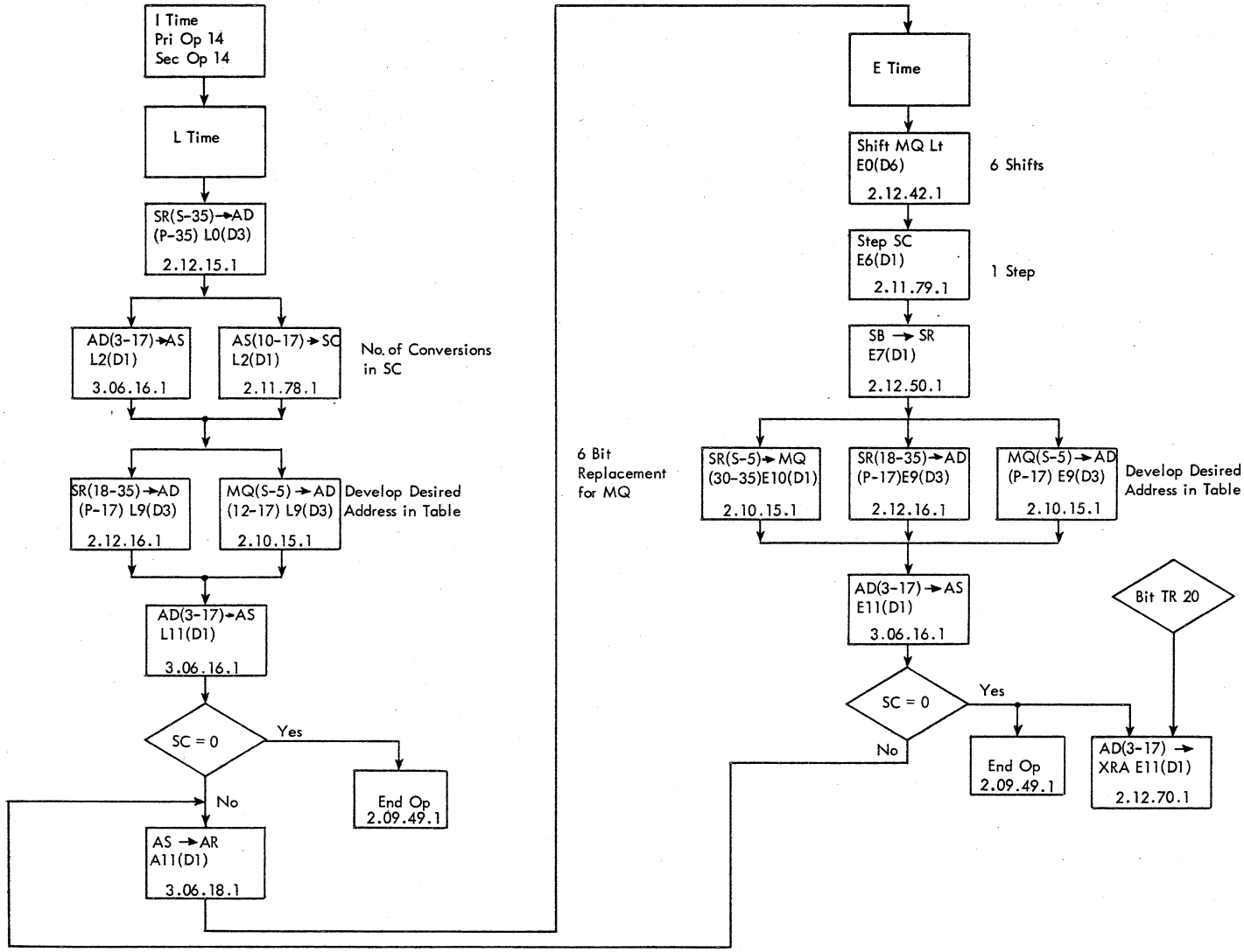


FIGURE 3.5-72. CRQ - 0154

Storage table required for CRQ (in decimal):

Storage Location	Contents (S-5)	Contents (21-35)
2000	BL	2000
2001	1	2010
2002	2	2010
2003	3	2010
2004	4	2010
2005	5	2010
2006	6	2010
2007	7	2010
2008	8	2010
2009	9	2010

Storage Location	Contents (S-5)	Contents (21-35)
2010	0	2010
2011	1	2010
2012	2	2010
2013	3	2010
2014	4	2010
2015	5	2010
2016	6	2010
2017	7	2010
2018	8	2010
2019	9	2010

Development of Program

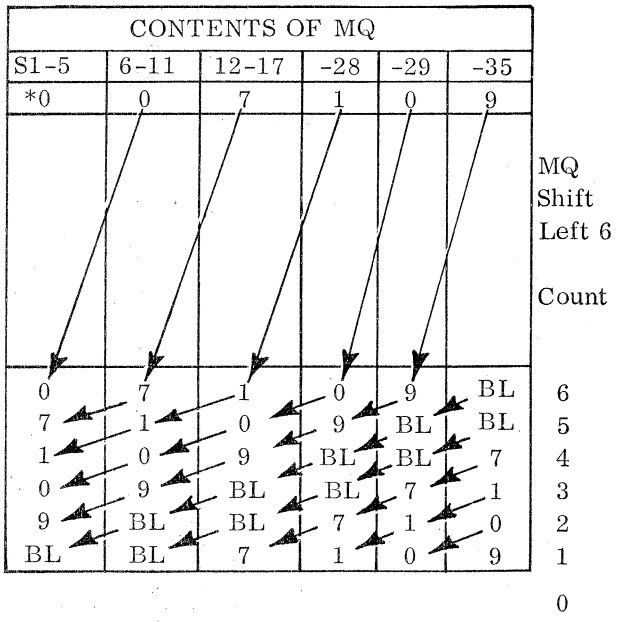
LDQ A — 1st Char Conv

CRQ 2000, 0, 6

Series of Steps within CRQ

C(MQ)s 1-5		Start Loc	Table Refer	Table C(-)s 1-5	Next Starting Loc C(-) 21-35
*0	+	2000 =	2000	BL	2000
0	+	2000 =	2000	BL	2000
7	+	2000 =	2007	7	2010
1	+	2010 =	2011	1	2010
0	+	2010 =	2010	0	2010
9	+	2010 =	2019	9	2010

Count = 0; if Tag = 1, 2010 → XRA



The execution of CRQ is accomplished in one L cycle and a number of E cycles equal to the count.

Convert by Addition from MQ — CAQ -0114 — Figure 3.5-73

This instruction treats the MQ as six 6-bit representations, as does CRQ. This instruction does not replace any of these representations, but uses them to locate a number of storage words equal to the count of the instruction. The storage words are located at the address developed by adding MQ(S-5) to SR(21-35) (initially the instruction address) and are brought to the SR. The storage words are then added to the contents of the accumulator. The MQ is not replaced, but is merely rotated left six places to allow the next six-bit representation to be added to the address portion of the previous storage word. The process continues until a number of additions (to the AC) equal to the count have been made. The operation is then complete. The address portion of the last storage word used can be stored in index registers for future reference by adding a tag bit in position 20 of the instruction.

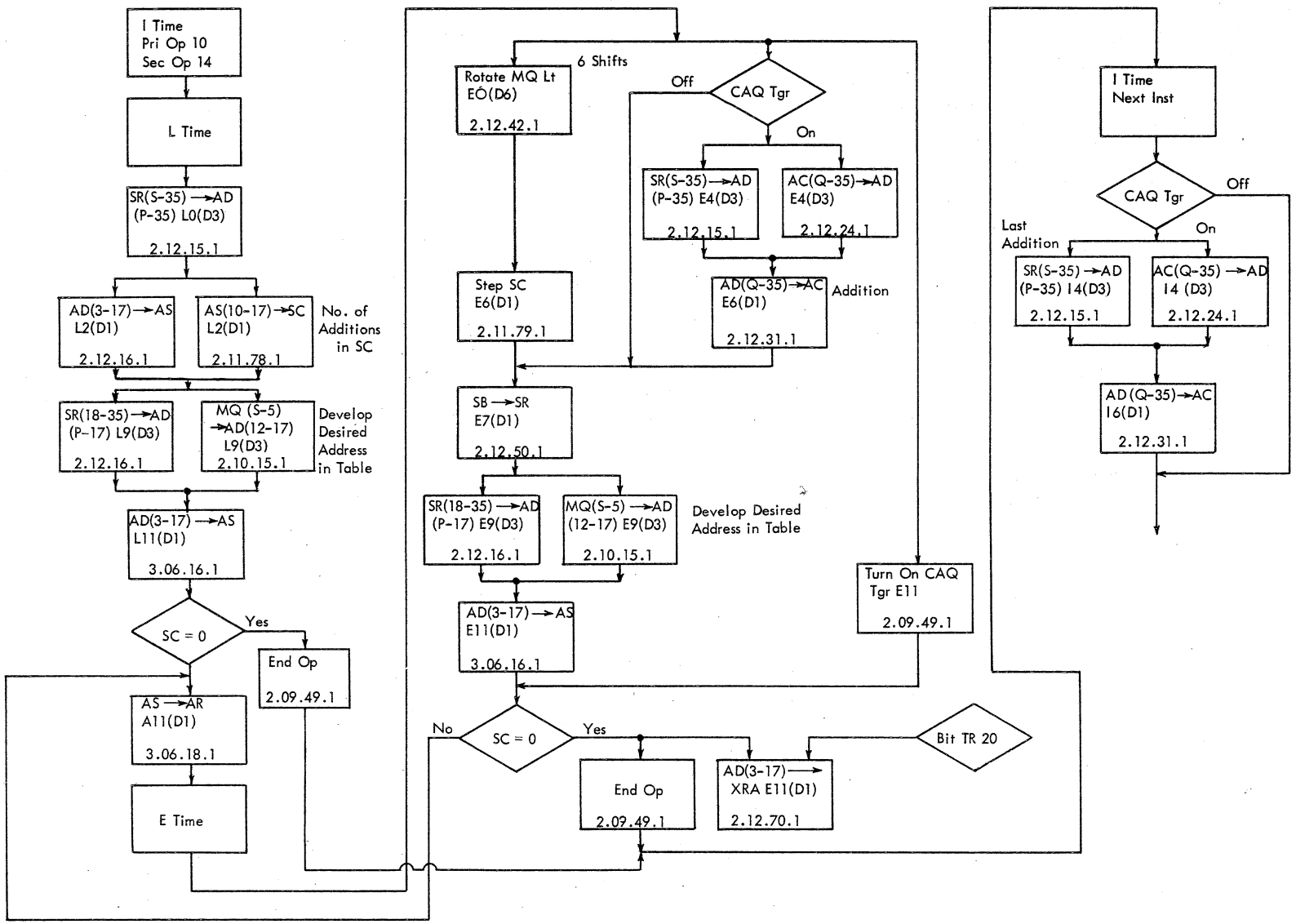


FIGURE 3.5-73. CAQ - 0114

To illustrate the operation and use of CAQ, a program which will convert BCD to binary follows.

Convert BCD Word to Binary

709542 (BCD) converted to 2,551,646 (Octal)

Instructions required for this operation:

```
LDQ          A      (BCD word)
CLM                 (Clear AC)
CAQ 6, 0, 3000      (Convert to binary)
ARS 16_10           (Position result in AC)
SLW          B      (Store result)
```

Storage table required for CAQ:

Storage Location	Contents (S-19)	Contents (21-35)
Decimal	Octal	Decimal
3000	0	3100
3001	303,240	3100
3002	606,500	3100
3003	1,111,740	3100
3004	1,415,200	3100
3005	1,720,440	3100
3006	2,223,700	3100
3007	2,527,140	3100
3008	3,032,400	3100
3009	3,335,640	3100
3100	0	3200
3101	23,420	3200
3102	47,040	3200
3103	72,460	3200
3104	116,100	3200
3105	141,520	3200
3106	165,140	3200
3107	210,560	3200
3108	234,200	3200
3109	257,620	3200
3200	0	3300
3201	1,750	3300
3202	3,720	3300
3203	5,670	3300
3204	7,640	3300
3205	11,610	3300
3206	13,560	3300
3207	15,530	3300
3208	17,500	3300
3209	21,450	3300

Storage Location	Contents (S-19)	Contents (21-35)
Decimal	Octal	Decimal
3300	0	3400
3301	144	3400
3302	310	3400
3303	454	3400
3304	620	3400
3305	764	3400
3306	1,130	3400
3307	1,274	3400
3308	1,440	3400
3309	1,604	3400
3400	0	3500
3401	12	3500
4302	24	3500
3403	36	3500
3404	50	3500
3405	62	3500
3406	74	3500
3407	106	3500
3408	120	3500
3409	132	3500
3500	0	
3501	1	
3502	2	
3503	3	
3504	4	
3505	5	
3506	6	
3507	7	
3508	10	
3509	11	

Development of Program

Binary Equivalent

BCD Word

LDQ BCD WORD
CLM
CAQ 3000, 0, 6

Series of Steps within CAQ

C(MQ)s 1-5		Start Loc		Table Refer	Table C(-)s 1-19	Next Start Loc C(-) 21-35)		CONTENTS OF ACCUMULATOR P, 1-19	21-35	Count
								XX	XX	
								00	00	
									RQL 6	
7	+	3000	=	3007	2,527,140	3100	+	2,527,140	3100	6
0	+	3100	=	3100	0	3200	+	0	3200	5
							=	2,527,140	6300	
9	+	3200	=	3209	21,450	3300	+	21,540	3300	4
							=	2,550,610	9600	
5	+	3300	=	3305	764	3400	+	764	3400	3
							=	2,551,574	13000	
4	+	3400	=	3404	50	3500	+	50	3500	2
							=	2,551,644	16500	
2	+	3500	=	3502	2	-	+	2	-	1
							=	2,551,646	16500	
ARS 16 (Binary Equivalent) →								2,551,646		0

CONTENTS OF MQ

S1-5	6-11	-17	-23	-29	-35
*7	0	9	5	4	2
0	9	5	4	2	7
9	5	4	2	7	0
5	4	2	7	0	9
4	2	7	0	9	5
2	7	0	9	5	4
7	0	9	5	4	2

SLW_

Appendix A

Alphabetic Listing and Index of 7090 Instructions

OPERATION CODE ALPHA	OCTAL		INSTRUCTION	INDEXABLE	INDIRECTLY ADDRESSABLE	PAGE
ACL	0361		Add and Carry Logical Word	X	X	32
ADD	0400		Add	X	X	30
ADM	0401		Add Magnitude	X	X	32
ALS	0767		Accumulator Left Shift	X		25
ANA	—0320		AND to Accumulator	X	X	96
ANS	0320		AND to Storage	X	X	96
ARS	0771		Accumulator Right Shift	X		27
AXC	—0774		Address to Index Complemented			94
AXT	0774		Address to Index True			94
BSF	—0764		Backspace File	X		*
BSR	0764		Backspace Record	X		*
BTT	0760..	xxxx	Beginning of Tape Test	X		75
CAL	—0500		Clear and Add Logical Word	X	X	30
CAQ	—0114		Convert by Addition from MQ			105
CAS	0340		Compare AC with storage	X	X	67
CHS	0760..	0002	Change Sign	X		78
CLA	0500		Clear and Add	X	X	27
CLM	0760..	0000	Clear Magnitude	X		40
CLS	0502		Clear and Subtract	X	X	27
COM	0760..	0006	Complement Magnitude	X		40
CRQ	—0154		Convert by Replacement from MQ			103
CVR	0114		Convert by Replacement from AC			101
DCT	0760..	0012	Divide Check Test	X		72
DVH	0220		Divide or Halt	X	X	35
DVP	0221		Divide or Proceed	X	X	37
ECTM	—0760..	0006	Enter Copy Trap Mode	X	X	*
EFTM	—0760..	0002	Enter Floating Trap Mode	X		*
ENB	0564		Enable	X	X	*
ENK	0760..	0004	Enter Keys	X		25
ERA	0322		Exclusive OR to Accumulator	X	X	96
ESNT	—0021		Enter Storage Null. and Transfer	X	X	*
ESTM	—0760..	0005	Enter Select Trap Mode	X		*
ETM	0760..	0007	Enter Trapping Mode	X		64
ETT	—0760..	xxxx	End of Tape Test	X		75
FAD	0300		Floating Add	X	X	43
FAM	0304		Floating Add Magnitude	X	X	49
FDH	0240		Floating Divide or Halt	X	X	54
FDP	0241		Floating Divide or Proceed	X	X	59
FMP	0260		Floating Multiply	X	X	49
FRN	0760..	0011	Floating Round	X		54
FSB	0302		Floating Subtract	X	X	49
FSM	0306		Floating Subtract Magnitude	X	X	49
HPR	0420		Halt and Proceed			75
HTR	0000		Halt and Transfer	X	X	75
IIA	0041		Invert Indicators from AC			84
IIL	—0051		Invert Indicators of Left Half			82
IIR	0051		Invert Indicators of Right Half			82
IIS	0440		Invert Indicators from Storage	X	X	78
IOT	0760..	0005	Input-Output Check Test	X		72
LAC	0535		Load Complement of Address in Index			94
LAS	—0340		Logical Compare Accumulator with Storage	X	X	69
LBT	0760..	0001	Low-Order Bit Test	X		67
LCHA	0544		Load Channel A	X	X	*
LCHB	—0544		Load Channel B	X	X	*
LCHC	0545		Load Channel C	X	X	*

OPERATION CODE ALPHA	OCTAL		INSTRUCTION	INDEXABLE	INDIRECTLY ADDRESSABLE	PAGE
LCHD	—0545		Load Channel D	X	X	*
LCHE	0546		Load Channel E	X	X	*
LCHF	—0546		Load Channel F	X	X	*
LCHG	0547		Load Channel G	X	X	*
LCHH	—0547		Load Channel H	X	X	*
LDC	—0535		Load Complement of Decrement in XR			94
LDI	0441		Load Indicators	X	X	78
LDQ	0560		Load MQ	X	X	22
LFT	—0054		Left Half Indicators, Off Test			87
LFTM	—0760..	0004	Leave Floating Trap Mode	X		*
LGL	—0763		Logical Left Shift	X		25
LGR	—0765		Logical Right Shift	X		27
LLS	0763		Long Left Shift	X		25
LNT	—0056		Left Half Indicators, On Test			87
LRS	0765		Long Right Shift	X		27
LSNM	—0760..	0010	Leave Storage Nullification Mode	X		*
LTM	—0760..	0007	Leave Trapping Mode	X		64
LXA	0534		Load Index from Address			94
LXD	—0534		Load Index from Decrement			96
MPR	—0200		Multiply and Round	X	X	35
MPY	0200		Multiply	X	X	32
MSE	—0760..	xxxx	Minus Sense	X		72
NOP	0761		No Operation			75
NZT	—0520		Storage Non-zero Test	X	X	67
OAI	0043		OR Accumulator to Indicators			84
OFT	0444		Off Test for Indicators	X	X	87
ONT	0446		On Test for Indicators	X	X	84
ORA	—0501		OR to Accumulator	X	X	96
ORS	—0602		OR to Storage	X	X	96
OSI	0442		OR Storage to Indicators	X	X	78
PAC	0737		Place Complement of Address in XR			91
PAI	0044		Place Accumulator in Indicators			82
PAX	0734		Place Address in XR			91
PBT	—0760..	0001	P-bit Test	X		67
PDC	—0737		Place Complement of Decrement in XR			91
PDX	—0734		Place Decrement in Index			91
PIA	—0046		Place Indicator in Accumulator			82
PSE	0760..	xxxx	Plus Sense	X		70
PXA	0754		Place Index in Address			91
PXD	—0754		Place Index in Decrement			94
RCHA	0540		Reset and Load Channel A	X	X	*
RCHB	—0540		Reset and Load Channel B	X	X	*
RCHC	0541		Reset and Load Channel C	X	X	*
RCHD	—0541		Reset and Load Channel D	X	X	*
RCHE	0542		Reset and Load Channel E	X	X	*
RCHE	—0542		Reset and Load Channel F	X	X	*
RCHG	0543		Reset and Load Channel G	X	X	*
RCHH	—0543		Reset and Load Channel H	X	X	*
RCT	—0760..	0014	Restore Channel Traps	X		*
RDS	0762		Read Select	X		*
REW	0772		Rewind	X		*
RFT	0054		Right Half Indicators, Off Test			87
RIA	—0042		Reset Indicators from Accumulator			84
RIL	—0057		Reset Indicators of Left Half			82
RIR	0057		Reset Indicators of Right Half			82

* Description not included in text.

OPERATION CODE ALPHA	OCTAL	INSTRUCTION	INDEXABLE	INDIRECTLY ADDRESSABLE	PAGE
RIS	0445	Reset Indicators from Storage	X	X	78
RND	0760..0010	Round	X		40
RNT	0056	Right Half Indicators, On Test			87
RQL	—0773	Rotate MQ Left	X		27
SBM	—0400	Subtract Magnitude	X	X	32
SCHA	0640	Store Channel A	X	X	*
SCHB	—0640	Store Channel B	X	X	*
SCHC	0641	Store Channel C	X	X	*
SCHD	—0641	Store Channel D	X	X	*
SCHE	0642	Store Channel E	X	X	*
SCHF	—0642	Store Channel F	X	X	*
SCHG	0643	Store Channel G	X	X	*
SCHH	—0643	Store Channel H	X	X	*
SIL	—0055	Set Indicator of Left Half			82
SIR	0055	Set Indicator of Right Half			78
SLQ	—0620	Store Left Half MQ	X	X	22
SLW	0602	Store Logical Word	X	X	20
SSM	—0760..0003	Set Sign Minus	X		78
SSP	0760..0003	Set Sign Plus	X		75
STA	0621	Store Address	X	X	22
STD	0622	Store Decrement	X	X	22
STI	0604	Store Indicators	X	X	78
STL	—0625	Store Instruction Location Counter	X	X	22
STO	0601	Store	X	X	20
STP	0630	Store Prefix	X	X	20
STQ	—0600	Store MQ	X	X	20
STR	—1000	Store Location and Trap			64
STT	0625	Store Tag	X	X	22
STZ	0600	Store Zero	X	X	20
SUB	0402	Subtract	X	X	32
SXA	0634	Store Index in Address			94
SXD	—0634	Store Index in Decrement			94
TCNA	—0060	Transfer on DSC A Not in Operation	X	X	62
TCNB	—0061	Transfer on DSC B Not in Operation	X	X	62
TCNC	—0062	Transfer on DSC C Not in Operation	X	X	62
TCND	—0063	Transfer on DSC D Not in Operation	X	X	62
TCNE	—0064	Transfer on DSC E Not in Operation	X	X	62
TCNF	—0065	Transfer on DSC F Not in Operation	X	X	62
TCNG	—0066	Transfer on DSC G Not in Operation	X	X	62
TCNH	—0067	Transfer on DSC H Not in Operation	X	X	62
TCOA	0060	Transfer on DSC A in Operation	X	X	62
TCOB	0061	Transfer on DSC B in Operation	X	X	62
TCOC	0062	Transfer on DSC C in Operation	X	X	62
TCOD	0063	Transfer on DSC D in Operation	X	X	62
TCOE	0064	Transfer on DSC E in Operation	X	X	62
TCOF	0065	Transfer on DSC F in Operation	X	X	62
TCOG	0066	Transfer on DSC G in Operation	X	X	62
TCOH	0067	Transfer on DSC H in Operation	X	X	62
TEFA	0030	Transfer on DSC A End of File	X	X	64
TEFB	—0030	Transfer on DSC B End of File	X	X	64
TEFC	0031	Transfer on DSC C End of File	X	X	64
TEFD	—0031	Transfer on DSC D End of File	X	X	64
TEFE	0032	Transfer on DSC E End of File	X	X	64
TEFF	—0032	Transfer on DSC F End of File	X	X	64
TEFG	0033	Transfer on DSC G End of File	X	X	64
TEFH	—0033	Transfer on DSC H End of File	X	X	64
TIF	0046	Transfer if Indicators Off	X	X	84
TIO	0042	Transfer if Indicators On	X	X	84
TIX	2000	Transfer on Index			88
TLQ	0040	Transfer on Low MQ	X	X	62
TMI	—0120	Transfer on Minus	X	X	60
TNO	—0140	Transfer on No Overflow	X	X	60
TNX	—2000	Transfer on No Index			88
TNZ	—0100	Transfer on No Zero	X	X	62
TOV	0140	Transfer on Overflow	X	X	60
TPL	0120	Transfer on Plus	X	X	60
TQO	0161	Transfer on Quotient Overflow	X	X	*
TQP	0162	Transfer on MQ Plus	X	X	60
TRA	0020	Transfer	X	X	60
TRCA	0022	Transfer on DSC A Redundancy Check	X	X	64
TRCB	—0022	Transfer on DSC B Redundancy Check	X	X	64
TRCC	0024	Transfer on DSC C Redundancy Check	X	X	64
TRCD	—0024	Transfer on DSC D Redundancy Check	X	X	64
TRCE	0026	Transfer on DSC E Redundancy Check	X	X	64
TRCF	—0026	Transfer on DSC F Redundancy Check	X	X	64
TRCG	0027	Transfer on DSC G Redundancy Check	X	X	64
TRCH	—0027	Transfer on DSC H Redundancy Check	X	X	64
TSX	0074	Transfer and Set Index			91
TTR	0021	Trap Transfer	X	X	64
TXH	3000	Transfer on Index High			88
TXI	1000	Transfer with XR Incremented			88
TXL	—3000	Transfer on XR Low or Equal			91
TZE	0100	Transfer on Zero	X	X	60
UAM	—0304	Unnormalized Add Magnitude	X	X	49
UFA	—0300	Unnormalized Floating Add	X	X	48
UFM	—0260	Unnormalized Floating Multiply	X	X	54
UFS	—0302	Unnormalized Floating Subtract	X	X	49
USM	—0306	Unnormalized Subtract Magnitude	X	X	49
VDH	0224	Variable Length Divide or Halt	X		37
VDP	0225	Variable Length Divide or Proceed	X		40
VLM	0204	Variable Length Multiply	X		35
WEF	0770	Write End of File	X		*
WRS	0766	Write Select	X		*
XCA	0131	Exchange AC and MQ			22
XCL	—0130	Exchange Logical AC and MQ			22
XEC	0522	Execute	X	X	75
ZET	0520	Storage Zero Test	X	X	67

* Description not included in text.