IBM Customer Engineering

Instruction-Maintenance

7094 II Data Processing System - Volume 2

Arithmetic Instructions

IBM®

Customer Engineering

Instruction-Maintenance

7094 II Data Processing System - Volume 2

Arithmetic Instructions

Preface

This is the second (Volume 2) of three volumes that make up the final version of the IBM 7094-II Customer Engineering Instruction-Maintenance manual. This volume contains all the arithmetic instructions for the IBM 7094-II Data Processing System and is arranged in four sections:

1. Fixed-Point
2. Floating-Point
3. Double-Precision Floating-Point
4. Reference Section (Flow Charts)

The material in this volume is written at engineering change level 253405; however, future engineering changes may change the logic and machine operations from their presentation here.

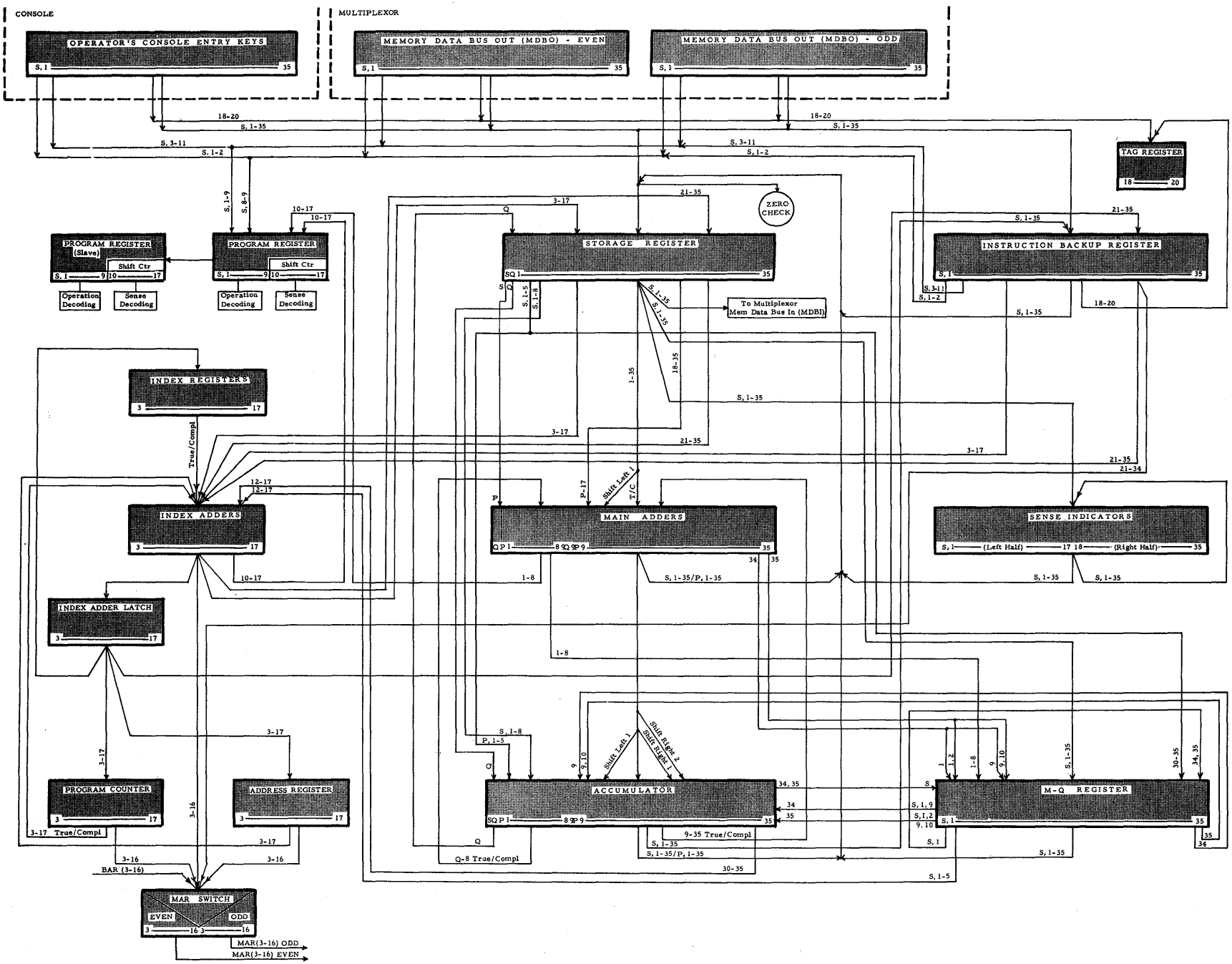
Volume 1 of the *Customer Engineering Instruction-Maintenance manual, IBM 7094-II,* Form 223-2721, contains information concerning: System Organization, Component Circuits, System and Functional Components, and Timing.

Volume 3 of the *Customer Engineering Instruction-Maintenance manual, IBM 7094-II,* Form 223-2723, contains information concerning: Non-Arithmetic Instructions, Overlap, Trapping, Channel Instructions, the IBM 7151-2 Console Control Unit, and Compatibility. Material in Volume 3 is presently available in the *Customer Engineering Instruction-Maintenance manual, IBM 7094-II,* Form Z22-2723, and Supplement, Form S23-4019.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, CE Manuals, Dept. B96, PO Box 390, Poughkeepsie, N. Y. 12602

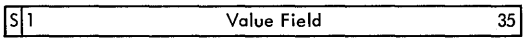
Contents

Illustrations



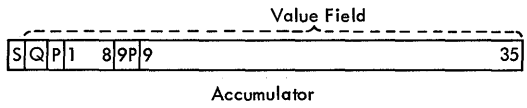
7094-II CPU Data Flow

Fixed-point arithmetic is the most basic form of arithmetic. Simply stated, it is the process of computation using quantities whose magnitude is completely expressed by a single (value) field. The relationship of the magnitude to zero is expressed by a sign position. In fixed-point arithmetic, the length of an operand is generally determined by the size of the word that occupies one location in core storage. In the 7094-II, fixed-point arithmetic operands have the following basic format:



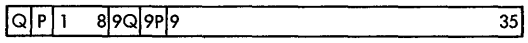
The sign bit (S) determines whether the magnitude is positive or negative. When S is a 0, the magnitude is positive; when S is a 1, the magnitude is negative. The value field is 35 bits long and states the magnitude of the number. A fixed-point operand can then be defined as a unit of data 36 bits long, containing a sign bit and 35 magnitude bits.

Fixed-point arithmetic in the 7094-II includes addition, subtraction, multiplication, and division. All these operations involve only two operands. One operand is explicitly addressed (addressed operand) and one operand is implied (implied operand). In all four operations, the explicitly addressed operand is obtained from the core storage location (Y) specified by the instruction. The implied operand varies with the operation: addition or subtraction implies the accumulator register (AC); multiplication, the multiplier-quotient register (MQ); division, the combined AC-MQ registers. The implied or accumulator operand has the following format:



The accumulator value field is 37 bits long. The additional bits, Q and P of the AC are provided primarily to handle conditions which result in an overflow out of position 1. Bits P and Q are therefore known as *overflow bits* and are treated as the two highest order accumulator bits during the execution of fixed-point arithmetic. Position 9P of the AC is not used in fixed-point arithmetic, but is used in floating-point arithmetic.

The actual arithmetic takes place in the adder which has the following format:

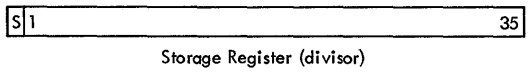


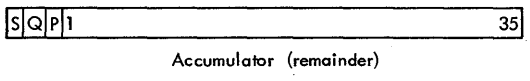
Note: 9Q and 9P not used in fixed point.

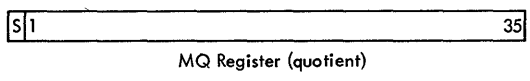
Basically, when the contents of the storage register (SR) are gated into the adders simultaneously with the true or complement form of the AC contents, an addition or subtraction is effected, and the result may be placed in the AC.

In multiplication, the addressed operand is obtained from the core storage location (Y) specified by the instruction; the implied operand is obtained from the MQ register. The addressed operand is placed in the SR, which has the basic format of a sign bit and a 35-bit value field. SR contents become the multiplicand. MQ contents form the multiplier, which has a format identical with the multiplicand. Multiplication is effected by a combination of right shifts and additions. A multiplication result is placed in the combined AC-MQ registers with MQ(35) the lowest order bit. Multiplication is algebraic, and the resultant sign is placed in both the AC(S) and MQ(S) positions.

In division, the addressed operand is obtained from the core storage location (Y) specified by the instruction; the implied operand is obtained from the combined AC-MQ registers. The addressed operand is placed in the SR and becomes the divisor; the combined AC-MQ registers become the dividend. Divisor format is the basic single sign bit and 35 value-field bits. The dividend format is a single sign bit and 72 value-field bits:







The result or quotient is placed in the MQ register and has a format identical with the divisor. Remainder bits, if any, go into the AC with a format of one sign bit and

37 value-field bits; AC(35) is the lowest order remainder bit. Division is effected by a combination of subtractions and left shifts.

Addition

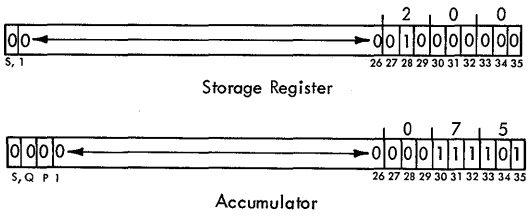
When performing addition in the 7094-II, the general rules of algebra must first be applied to the signs of the quantities involved to determine whether the sum or the difference of the quantities involved is to be obtained. Therefore, when adding two positive quantities, the result is the sum of those quantities with a positive sign. When adding a positive and a negative quantity, the sum is actually the difference of the two quantities with the resultant sign being the sign of the larger magnitude. Finally, when adding two negative quantities, the result is the sum of the quantities with a negative sign.

Assume the quantity $+200_8$ is to be added to the accumulator, which contains $+75_8$. The result is $+275_8$. To satisfy machine operand format, convert the quantities to their binary equivalents:

$+200_8 = +\ 010\ 000\ 000$

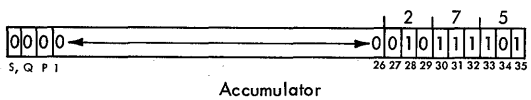
$+\ 75_8 = +\ 000\ 111\ 101$

Insert these binary numbers into respective data words with the lowest order bit going into bit 35:



Bits 1 through 26 are not needed to express the quantities and are therefore all 0's. Because accumulator bits Q and P are treated as part of the value field and the accumulator value is assumed as $+75_8$, bits P and Q are 0's. Because each number is positive, a 0 is placed in the respective sign bit (S).

Fixed-point addition in the 7094-II is identical with that described in binary addition: 0 + 0 = 0; 0 + 1 = 1; 1 + 1 = 0 with a 1 carry to the next higher position. Adding the two operands produces a resultant magnitude of 010 111 101, with a resultant sign of 0. In machine operand format, the result is as follows:



If the same magnitudes are used but the signs are changed to negative, the entire handling of the magnitude remains unchanged in performing the addition. The 7094-II treats the sign bits separately. To represent the negative values correctly, insert a 1 in the sign bit position of each of the operands and the result; this is what is done in the computer.

Because algebraic principles are employed, addition of two quantities with unlike signs is effectively a subtraction. Using the same values, but changing the sign of the accumulator operand to a minus, the problem becomes $(+\ 200_8) + (-\ 75_8)$. To accomplish addition, line up the octal points and subtract:

$+\ 200_8$

$-\ 075_8$

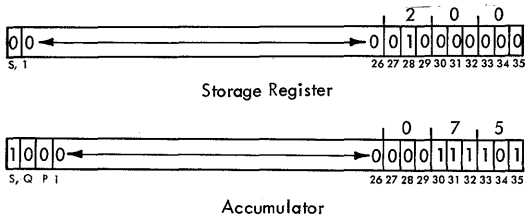
$+\ 103_8$

To satisfy machine operand format, convert the values to their binary equivalent:

$+\ 200_8 = +\ 010\ 000\ 000$

$-\ 075_8 = -\ 000\ 111\ 101$

Insert these binary numbers into their respective data words with the lowest order bit in each value going into bit 35 :



Bits 1 through 26 are not needed to express the quantities and are therefore all 0's. Accumulator bits Q and P are implied 0's by the assumed accumulator value.

The computer adds values having unlike signs as follows:

1. Complements the accumulator value field.
2. Adds the 1's complemented accumulator value field and storage register value field.
3. Places the result in the accumulator.
4. Compares the accumulator and storage register signs:
   a. If alike, check for a carry-out value field position
      1. The coincidence of like signs and a carry-out of position 1 indicates an overflow.
   b. If unlike, checks for a Q carry:
      1. If there is a Q carry, adds 1 to the accumulator in the lowest order position (bit 35), inverts the accumulator sign, and places the resultant operand in the accumulator.
      2. If there is no Q carry, complements the accumulator value field.

The addition is then performed as follows:

1. Storage Register = + 200_8 = + 010 000 000
   Accumulator = − 075_8 = − 000 111 101
2. Complementing the accumulator value field results in its containing 111 000 010, with bits Q-26 all 1's.
3. Add: 010 000 000
   111 000 010
   001 000 010 with a 1 carry propagated through the rest of the bits (Q-26) and out of Q.
4. The intermediate result is placed in the accumulator, which now contains − 001 000 010. Bits Q-26 are all 0's because of the propagated carry.
5. Checking the accumulator and storage register signs reveals they are unlike.
6. Checking for a Q carry reveals one.
7. Adding 1 to the accumulator lowest order bit makes the value field 001 000 011, and inverting the sign makes it positive (0).
8. The resultant value in the accumulator is + 001 000 011, which equals + 103_8.

Repeating the problem with + 200_8 as the accumulator operand and − 75_8 as the addressed operand causes the following:

1. Storage Register = − 75_8 = − 000 111 101
   Accumulator = + 200_8 = + 010 000 000
2. Complementing the accumulator value field results in its containing 101 111 111, with bits Q-26 all 1's.
3. Add: 000 111 101
   101 111 111
   110 111 100 with bits Q-26 unaffected.
4. The intermediate result is placed in the accumulator, which now contains + 110 111 100. Bits Q-26 are all 1's.
5. Checking the accumulator and storage register signs reveals that they are unlike.
6. Checking for a Q carry reveals none.
7. Complementing the accumulator value field yields a final result of + 001 000 011.

The term *overflow* means that the capacity of the machine has been exceeded. The arithmetic result cannot be represented by the machine because it contains more than 35 value field positions. As previously stated, the accumulator bits Q and P are called *overflow bits.* The name, however, only provides an easy means of identifying these bits as a pair. Because they could originally contain 00, 01, 10, or 11, their significance depends on the problem. When dealing with values having like signs, a resultant 1 in either bit or in both bits indicates an overflow. In this case, the overflow is remembered but subsequent action depends on the program being executed.

When dealing with unlike signs, the overflow bits are significant as a pair and, in this sense, they either generate a Q carry or they do not generate a Q carry. If a carry is generated, it indicates that the accumulator operand was the smaller operand and that the number presently in the accumulator value field is a true number equal to one less than the correct answer. If a Q carry is not generated, its absence indicates that the accumulator operand was the larger operand and that the number presently in the accumulator value field is the correct answer in complement form.

Clear and Add — CLA + 0500
(I, E)

The contents of AC(S, 1-35) are replaced with the contents of storage location (Y), as indicated by the address portion of the instruction. AC(Q, P) are set to zero. See Figure 8.

Clear and Add Logical Word — CAL − 0500
(I, E)

The logical contents of Y replace the contents of AC(P, 1-35), the sign of Y replacing AC(P). AC(S, Q) are set to zero. See Figure 8.

Add — ADD + 0400
(I, E)

The contents of Y are algebraically added to the contents of the AC. The resulting sum replaces the contents of the AC. AC overflow is possible. See Figure 9.

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 Q carry results, complement the accumulator factor and leave the accumulator sign unchanged.
      2. If a Q carry results, add one to the result and change the accumulator sign.

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(Q) indicates that the AC factor is

smaller. No Q carry indicates that the AC factor is larger. To remember the carry, a carry trigger is turned on by a carry-out of AD(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 1 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 recomplemented to get the correct true number. The sign of the result in the AC is set the same as the sign of the larger original factor, as determined by the status of the Q carry.

Example 1

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

Example 2

Signs Unlike, AC Smaller
−6+(+7) = +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
−7+(+6) = −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
(I, E)

The sign of Y is ignored and the contents of Y are treated as a positive number. This positive number is then added algebraically to the contents of the AC. The resulting sum replaces the contents of the AC. With a minus AC sign, a subtractive process (signs unlike) will occur. AC overflow is possible. See Figure 9.

Add and Carry Logical Word — ACL + 0361
(I, E)

The logical contents of Y are added to the contents of AC(P, 1-35), the sign of Y being added to AC(P). The resulting sum, including a carry to AD(35) if a carry-out of AD(P) occurs, replaces the contents of AC(P, 1-35). The AC sign is ignored; AC(Q) is unchanged; AC overflow is not possible. See Figure 10.

Subtraction

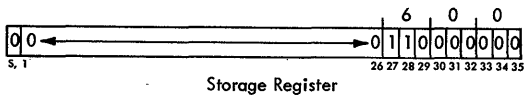
Subtraction in the 7094-II is algebraic and is accomplished as follows:

1. Invert the storage register sign.
2. Compare the accumulator and storage register signs:
   a. If alike, add the contents of the accumulator and storage register.
   b. If unlike, complement the accumulator and then add the contents of the accumulator and the storage register.
3. Place the addition result in the accumulator.
4. Compare the accumulator and storage register signs:
   a. If alike, check for a carry-out of value field position 1. The coincidence of like signs and a carry-out of value field position 1 indicates an overflow.
   b. If unlike, check for a Q carry:
      1. If there is a Q carry, add 1 to the present accumulator value field in the low-order position and invert the accumulator sign.
      2. If there is no Q carry, complement the accumulator value field.

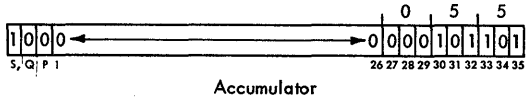
Assume the problem $-55_8 - (+600_8)$, where $+600_8$ is the addressed operand and -55_8 is the implied operand. The result is -655_8. To satisfy machine operand format, convert the quantities to their binary equivalents:

$+ 600_8 = + 110\ 000\ 000$
$- 55_8 = - 000\ 101\ 101$

Insert these binary numbers into respective data words with the lowest order bit going into bit 35.



Storage Register



Accumulator

Bits 1 through 26 are not needed to express the quantities and are therefore all 0's. Because accumulator bits Q and P are treated as part of the value field and the accumulator value is assumed as -55_8, bits P and Q are 0's. The addressed operand is positive, so its sign bit is a 0, whereas the implied operand is negative, so its sign bit is a 1.

Following this procedure, the subtraction is accomplished as follows:

1. Storage Register = $+ 600_8 = + 110\ 000\ 000$
   Accumulator = $- 55_8 = - 000\ 101\ 101$

2. Complementing the storage register sign results in the register containing − 110 000 000.

3. Comparing the operand signs reveals they are alike.

4. Add: 110 000 000
000 101 101
110 101 101 with bits Q-26 all 0's.

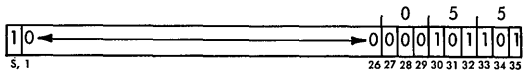
5. The addition result is placed in the accumulator, which now contains − 110 101 101.

6. Comparing the accumulator and storage register signs reveals they are alike.

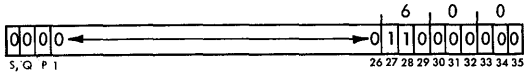
7. Checking for a Q carry reveals none.

8. The final answer in the accumulator is − 110 101 101 which equals -655_8.

Repeating the problem, but with -55_8 the addressed operand, the operand formats are as follows:



Storage Register



Accumulator

In accordance with the procedure, the following takes place:

1. Storage Register = -55_8 = − 000 101 101
Accumulator = $+600_8$ = + 110 000 000

2. Complementing the storage register sign results in the register containing + 000 101 101.

3. Comparing the accumulator and storage register sign reveals they are alike.

4. Add: 000 101 101
110 000 000
110 101 101 with bits Q-26 all 0's

5. The addition result is placed in the accumulator, which now contains + 110 101 101.

6. Comparing the accumulator and storage register signs reveals they are alike.

7. Checking for a Q carry reveals none.

8. The final answer in the accumulator is + 110 101 101, which equals $+655_8$.

Note the identical manner in which the two problems were handled. In each case, the arithmetic was addition. In each case, the sign of the subtrahend (storage register operand) was inverted. Subtraction of unlike signs becomes addition and it is not significant whether the accumulator is the larger or smaller operand.

Clear and Subtract **CLS + 0502**
(I, E)

With the sign position of Y sent in inverted form, the contents of Y replace the contents of AC(S, 1-35). AC(Q, P) are set to zero. See Figure 8.

Subtract **SUB + 0402**
(I, E)

The contents of Y are algebraically subtracted from the contents of the AC. The difference replaces the contents of the AC. This instruction operates the same as ADD, except the sign of Y is used in inverted form. AC overflow is possible. See Figure 9.

Subtract Magnitude **SBM − 0400**
(I, E)

The sign of Y is ignored and the contents of Y are treated as a negative number. This negative number is then added algebraically to the contents of the AC. The resulting sum replaces the contents of the AC. With a minus AC sign, an additive process (signs alike) will occur. AC overflow is possible. See Figure 9.

Multiplication

In order to simplify the 7094-II multiplication process, a review of the basic machine process using only one digit of the multiplier at a time is as follows:

Binary computers perform multiplication by repetitive addition and shifting. The process is similar to that used when performing binary multiplication using pencil and paper. The basic rule is to add and shift when a 1 is decoded in the low-order position of the multiplier and to shift without addition when a zero is decoded.

Assume the problem is to multiply 15_8 by 5_8. On paper we would do the following:

$15_8 =$ 1101_2 (multiplicand)
$5_8 =$ 101_2 (multiplier)
1101 (first multiply by 1)
0000 (multiply by zero—no add—shift)
1101 (second multiply by 1—shift and add)
1000001 $= 101_8$

Proof: $15_8 \times 5_8 = 13_{10} \times 5_{10} = 65_{10}$
$101_8 = 65_{10}$

In the first step, a 1 is contained in the low-order position of the multiplier. With a 1 in this position, the first partial product is equal to the value of the multiplicand. The second step requires a multiplication by 0. To accomplish this, 0's are added to the first partial product formed. The relative position of the partial product is maintained by displacing the 0's left one place before the summation.

The final iteration is a multiplication by 1. The multiplicand is shifted left and added to the partial prod-

uct formed as a result of the previous two multiply iterations.

Thus, to perform a multiplication of two binary numbers, the state of the low-order position of the multiplier is examined to determine whether that iteration of the multiply cycle is to be a multiplication by 0 or by 1. If the contents of the low-order position is a 1, a multiplication by 1 is required and the multiplicand is added to any previous partial product formed. If no bit is detected in the low-order position of the multiplier, a multiplication by 0 is accomplished by adding 0 to the partial product.

For convenience, three registers are used to program a multiplication. The multiplicand is contained in the SR, the multiplier in the MQ, and the partial product is formed in the AC. The AC and the MQ are shifted right after each multiplication by 1 or 0 to sense the next higher order position of multiplier and to maintain the proper relationship between the partial product and the multiplicand.

Because of the cost to store each partial product separately before summation, an addition is performed after each iteration and the answer is gradually built up in the AC and MQ. A shift counter (SC) is used to indicate when the proper number of multiplier bits have been processed, and the multiplication is complete. The value to which the SC is set is either the length of the multiplier, 35_{10} bits in the 7094-II or a value determined by the decrement field in a variable length instruction.

Assume we are to perform the same problem ($15_8 \times 5_8$), using a binary computer with five-position registers. At the start of the problem the registers would contain:

SC = 101 (5_{10}) SR = 01101 AC = 00000 MQ = 00101

MQ(5) is sensed to determine if its contains a 1 or a 0. If MQ(5) is a 1, the contents of the SR are added to the contents of the AC and the result is put into the AC. The AC and MQ are shifted right one place to align the registers for the next step. This shift puts bit 4 of the MQ into position 5 for sensing and also puts the least significant bit of the answer into MQ(1). The registers now contain:

SC = 100 SR = 01101 AC = 00110 MQ = 1]0010

The bracket around the first bit in the MQ indicates this bit is part of the partial product. MQ(5) is again sensed to determine if its contains a 1 or 0. Because a 0 is encountered, no addition takes place but the AC and MQ are shifted right one place. The registers now look like this:

SC = 011 SR = 01101 AC = 00011 MQ = 01]001

The 1 in MQ(5) requires an addition and a shift. The contents of the SR and AC are added and the result placed in the AC. The AC and MQ are shifted right one place and the registers now contain:

SC = 010 SR = 01101 AC = 01000 MQ = 001]00

MQ(4,5) now contain the last two bits of the original multiplier—both 0's. These 0's will result in shifting without addition and at the end of the problem the registers will contain:

SC = 0 SR = 01101 AC = 00010 MQ = 00001]

The operation is halted because SC = 0. The answer is contained in both the AC and MQ which equal $0001000001_2 = 101_8 = 65_{10}$.

Binary multiplication is performed by examining the low-order position of the multiplier to determine whether that iteration of the multiply cycle is to be a multiplication by 0 or 1. If the contents of the low-order multiplier position is a 1, a multiplication by 1 is indicated and the multiplicand (storage register) is added to any previous partial product in the AC. The contents of the AC and MQ are shifted right one place to align the new partial product, and to place the next higher order position of the multiplier in the low-order position of the MQ. If 0 had been detected in MQ(35), a multiplication by 0 would have been indicated and accomplished by shifting the partial product and multiplier without adding the multiplicand.

Thus, by examining one position of the multiplier and adding and shifting for the proper number of iterations, the multiplication is performed. Because one position is examined, each iteration performed may be considered as a multiplication by either 0 or 1. The result of each multiplication is added to the partial product which, in turn, is shifted to maintain the proper relationship between the partial product and the multiplicand.

In the 7094-II, instead of looking at one digit of the multiplier, two digits of the multiplier are decoded at a time. It then becomes apparent that on any one iteration, a multiply by 0, 1, 2, or 3 is possible. A multiplier decoder is used which senses the four possible states of MQ(34,35) which correspond to four numbers, 0, 1, 2, 3 in the base four number system. Because time is required to decode the states of MQ(34,35), it is necessary on all iterations, except the initial or static one, to presense the state of these registers by examining the condition of MQ(32,33).

The presensing (or sampling) of MQ(32,33) results in the setting of the pre-MQ(34,35) triggers. The condition of these triggers, along with the pre-string bit trigger, determines the decoding for the current iteration (multiply cycle). These decoded values indicate the number of times the multiplicand is to be added to the partial product.

A multiplication by 0 ($\times$ 0) indicates that the multiplicand is not added to the partial product. In a multiplication by 1 ($\times$ 1), the partial product is increased by the amount of the multiplicand. In a multiplication by 2 ($\times$ 2), the multiplicand is doubled by shifting it left one place to the adders. Thus, the partial product is increased by an amount equal to two times the multiplicand. A multiplication by 3 ($\times$ 3) is performed by subtracting the multiplicand from the partial product once and adding an additional four times the multiplicand to the partial product during the next iteration.

While it is possible to multiply by four by shifting the multiplicand left two places, and additional iteration would be required to perform the necessary left shift before adding, and to perform the complement add cycle. To obviate the necessity for this additional iteration, a multiplication by 4 is performed by increasing, by 1, the next two higher order positions of the multiplier after the complement add cycle has been performed. Thus a multiplication by 3 is accomplished in one multiply iteration, consisting of the 2's complement addition of the multiplicand, and the adjustment of the next two higher order positions of the multiplier.

A string bit trigger is used to "remember" that a 1 is to be added to the next two higher order bits of the multiplier before they are decoded. This has the effect of shifting the multiplicand left two places for each 2's complement add iteration performed. Once the string bit trigger is set on, it stays on until a $\times$ 1 or $\times$ 2 addition occurs.

The ability to perform base 4 multiplication requires the computer to scan multiplier bits at a rate twice as fast as for a binary multiplier decoding scheme. Because of the irregular method of a multiplication by 3, additional circuitry is required to remember that a complement addition was performed. The multiplier may be adjusted for the next iteration, and so the complement form of the high-order partial product positions is maintained.

The complement form of the high-order positions of the partial product is maintained by utilizing accumulator overflow positions Q and P, which are set during each complement add iteration, and remain set until a true add is performed. With AC(Q,P) on, 1's are gated to the high-order position (Q and P) of the adders. These positions, gated right to the accumulator registers one and two, provide the means of setting these registers on during the subtraction and during each subsequent multiply iteration, until a true add takes place. A true add will occur only when a multiplication by 1 or 2 is decoded. Because the partial product formed as a result of a $\times$ 3 iteration contains 1's in the high-order positions, it is always necessary to terminate a complete multiplication with a true add iteration. This "string" of 1's in the partial product is, during a multiplication by 1, added to the multiplicand. As the partial product has been shifted right two places each iteration, the highest order position of the multiplicand containing a 1 corresponds to a position in the partial product included within this string of 1's, and a $\times$ 1 addition generates a 0 in AD(Q). The output of AD(Q) is gated directly to AC(P) and serves to set 0 in that position at the end of each $\times$ 1 multiplictaion following a $\times$ 3 cycle. On a $\times$ 2 multiplication with SR gated left one place, there could be a carry into AD(Q) which would become a part of the partial product in AC(1) when shifted right two places. To prevent this, the output of AD(Q) is blocked to AC(P), resulting in the reset of AC(P) and indicating that we have just performed a true add.

The cyclic make-up of the MPY instruction is I, E, L, L. Two multiply iterations are performed during E time and sixteen iterations are performed during the following two L cycles. To allow for an effective continuous shift of the AC and MQ, the adder outputs are gated right two places to the AC; the MQ is shifted right two places; and the SC is stepped by two on each iteration (each clock pulse) except the last. The last iteration occurs on the second L7 clock pulse when the SC value is equal to one. On this last iteration, the adder outputs are gated right one place to the AC; the MQ is shifted right one place; and the SC is stepped by one to zero. See Figure 11.

Multiply **MPY + 0200**
(MIN I, E; MAX I, E, 2L)

The contents of Y are multiplied 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). AC(Q, P) are set to zero and the signs of the AC and MQ are set to the algebraic sign of the product. See Figure 11.

Multiply and Round **MPR – 0200**
(MIN I, E; MAX I, E, 3L)

Multiply and round operates the same as the multiply (MPY) instruction, and also adds 1 to the AC contents if MQ(1) equals one after the multiplication is complete. See Figure 11.

Round **RND + 0760 . . . 0010**
(I, L)

If MQ(1) equals one, the AC contents are increased by 1. If MQ(1) equals zero, the AC contents are unchanged. In either case, the MQ contents are unchanged. Note that positions 24-35 of this instruction

represent part of the operation code. Modification by indexing may change the operation code itself. AC overflow is possible. See Figure 13.

Variable-Length Multiplication

Variable-length multiplication is fixed-point multiplication with an operand other than 35 bits. The decrement of the instruction is used as the count field in variable length multiply. This count (c) is entered into the SC to control the number of multiply iterations performed, thus specifying the size of the product. The most significant portion of the product is placed in the AC. The least significant portion of the product is placed in the high-order positions of the MQ. The number of MQ positions used will equal the value of the count field contents (c). The count specified is usually less than 43_8. With a count less than 43_8, the low order of MQ will contain a number of unused positions equal to 43_8 minus the count field. See Figure 11.

If a count of 43_8 is used, the variable-length instruction will perform as a fixed-length instruction. A count of 60_8 or greater will cause an I/A cycle, (count extending into positions 12 and 13 of instruction) and the count of the I/A word will be set into the SC. However, this is a program error as variable-length instructions are not indirectly addressable.

Variable-length instructions are used to conserve machine time. For example, if the multiplier is never more than six digits long, one L cycle can be saved during each multiply operation. Figure 12 shows a X-Y recording of the multiply cycles which occur during a typical VLM instruction. It should be noted that many of the lines shown are the same for both fixed and floating multiply.

Variable Length Multiply VLM + 0204 (MIN I, E; MAX I, E, 3L)

Variable-length multiply operates the same as the multiply (MPY) instruction with the following exceptions: The number of multiplier positions to be tested is specified by the number in the decrement portion (count field) of the instruction. The 35 most significant bits of the product replace the contents of AC(1-35) and the number of least significant bits, as specified by the count field (c), replace the contents of MQ(1-c). The remaining low-order positions of the MQ will contain the original 35-c high-order positions of the MQ. See Figure 11.

Division

Fixed-point binary division in the 7094-II is accomplished by dividing the contents of the AC and MQ, taken together as the dividend, by the contents of the SR, the divisor. A 35-position quotient is developed in the MQ with the remainder, if any, 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. The quotient can never exceed 35 bits, the maximum length of the MQ. Therefore, the SC is set to 43_8 (35_{10}) to control the number of iterations of the divide process and is stepped after each iteration.

If the AC portion of the dividend is equal to or greater than the divisor, the quotient will be too large for the MQ. This condition prohibits division and turns the divide check indicator on. The computer will then either stop or proceed, depending on the type of divide instruction.

The following problem illustrates a hand performed binary division. Assuming a 4-bit register, it shows that had the AC portion of the dividend been equal to or greater than the divisor, a significant bit of the quotient would have had to have been entered into the AC; this would constitute a divide check condition.

```
                    AC   MQ
SR = 15₈ = 13₁₀          0110     Quotient (MQ) = 6₈ = 6₁₀
  (SR) Divisor
      1101 ) 0100 1110            Dividend
               11 01              (AC and MQ) = 116₈ = 78₁₀
             ---------
               01 101
                1 101                      6
             ---------          13 ) 78
                0 0000
                  0000
             ---------
                  0000           Remainder (AC)
```

Note in the problem that the divisor will go once, or not at all, into the high-order position 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 1 is put into 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 0 remains in the quotient. Another position of the dividend is now taken into account and the procedure starts again. These iterations continue until all positions in the dividend have been tested.

In the 7094-II, the SR and AC are complement-added (subtracted) to determine if a reduction of the high-order positions of the dividend is possible. If a reduction is possible, these 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 re-

sults in a 1 being put into the low-order of the MQ. Following the reduction attempt, the AC and MQ are shifted one place left to bring the next position of the AC into alignment and another reduction is attempted. The repetitive process continues until all positions of the MQ portion of the dividend have been moved to the AC. The SC will equal 0 when the division is complete. See Figure 14.

In fixed-point division, four instructions are used: divide or halt (DVH), divide or proceed (DVP), variable-length divide or halt (VDH), and variable-length divide or proceed (VDP). Note that three conditions are involved in these instructions. First, there are two halt-type instructions (DVH) and (VDH). These two instructions will stop the computer in I time of the next instruction if a divide check occurred. Second, there are two proceed-type instructions (DVP) and (VDP). These two instructions allow the computer to proceed in I time of the next instruction even though a divide check occurred, and give the programmer the option of testing the divide check indicator with a DCT instruction. Third, there are two variable-length type instructions (VDH) and (VDP) which allow a count other than 43_8 to be entered into the SC. These two instructions are used by the programmer usually when his quotient is a fixed length and is less than 35 digits long. For more information on these, see "Variable-Length Division."

Divide or Halt **DVH + 0220**
(MIN I, E; MAX I, E, 5L)

The contents of AC(Q-35) and MQ(1-35) are divided by the contents of storage location Y(1-35). The 35-bit quotient replaces the contents of MQ(1-35) and the remainder replaces the contents of AC(1-35). The MQ sign is the algebraic sign of the quotient and the AC sign is the sign of the dividend.

If the magnitude of Y is greater than the magnitude of the AC, division takes place. If the magnitude of Y is equal to or less than the magnitude of the AC, division does not occur and the computer stops with the divide check indicator on. For example, if AC(P) contains a 1, the magnitude of Y is less than the AC contents. If division does not occur, the dividend remains unchanged in the AC and MQ. See Figure 14.

Divide or Proceed **DVP + 0221**
(MIN I, E; MAX I, E, 5L)

Divide or proceed is the same as the divide or halt (DVH) instruction with one exception: When division does not occur (divide check condition), the computer proceeds to the next sequential instruction. See Figure 14.

Variable-Length Division

Variable-length division is fixed-point division with an operand of a length other than 35 bits. The decrement of the instruction is used as the count field in variable-length instructions. This count (c) is entered into the SC to control the number of divide iterations performed, thus specifying the number of significant digits of the quotient. The count is usually less than 43_8.

If a count of 43_8 is used, the variable-length instruction will perform as a fixed-length instruction. A count of 60_8 or greater will cause an I/A cycle (count extending into positions 12 and 13 of instruction), and the count of the I/A word will be set into the SC. However, this is a program error as variable-length instructions are not indirectly addressable.

Variable-length instructions are used to conserve machine time. The number of positions in the quotient is equal to the count and will be contained in the low-order end of the MQ. See Figure 14.

Variable-Length Divide or Halt **VDH + 0224**
(MIN I, E; MAX I, E, 5L)

Variable-length divide or halt is the same as the divide or halt (DVH) instruction with the following exceptions: The contents of the count field (c) determines the size of the quotient in the low-order positions of the MQ. The remainder replaces the contents of AC(1-35) and the 35-c high-order positions of the MQ. If the count field is zero, the computer will interpret the instruction as a no-operation, end op in E time, and proceed to the next instruction. See Figure 14.

Variable-Length Divide or Proceed **VDP + 0225**
(MIN I, E; MAX I, E, 5L)

Variable-length divide or proceed is the same as the divide or proceed (DVP) instruction with the following exceptions: The contents of the count field (c) determines the size of the quotient in the low-order positions of the MQ. The remainder replaces the contents of AC(1-35) and the 35-c high-order positions of the MQ. If the count field is zero, the computer will interpret the instruction as a no-operation, end op in E time, and proceed to the next instruction. See Figure 14.

Floating-Point Arithmetic

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 10 decimal digits).
2. The programmer must keep track of the point in all numbers throughout the calculation.

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

Floating-point arithmetic is 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).

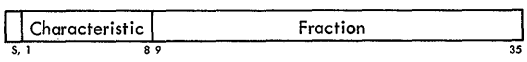
Floating-point numbers in binary are similar to decimal floating-point numbers. The major difference is the base. Numbers in the 7094-II use 2 as a base because it is a binary computer. The other difference is one of terms. Instead of a decimal point, we will call it a binary point.

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

DECIMAL	FIXED-POINT BINARY	FLOATING-POINT BINARY
4	000 100	0.1×2^{001}
11	001 001	0.1001×2^{100}

Characteristic and Fraction

Because the 7094-II 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 limits our need to represent 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:



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 0. 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 BINARY	FLOATING-POINT		
	S	1—8	9—35
$+ 0.1 \times 2^{011}$	+	10000011	1000-----0
$- 0.01 \times 2^{001}$	−	10000001	0100-----0
$+ 0.1 \times 2^{-011}$	+	01111101	1000-----0

Sign Control

1. Addition: With unlike signs and equal factors, the answer equals the sign of the original AC. In all other cases, the answer equals the sign of the larger factor.
2. Subtraction: After the sign of the storage word is inverted to the SR, the rules for addition apply.
3. Multiplication: Signs of factors alike, answer plus. Signs of factors unlike, answer minus.
4. Single-precision division: Signs of factors alike, quotient sign plus. Signs of factors unlike, quotient sign minus. The remainder sign equals the original dividend sign unless the dividend is zero. When the dividend is zero, the remainder sign is set plus.
5. Double-precision division: On a divide check condition, the MQ sign is set to equal the AC sign. In all other cases, rules for addition apply.

Normal and Unnormal Numbers

A floating-point number is in normal form when the digit immediately to the right of the point is a significant bit (1). If this digit is a zero, the number is 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 process these two types of numbers, instructions are 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.

Zero Fraction

A floating-point number having a zero fraction is treated in a variety of ways because the significance of a zero fraction operand depends on the arithmetic process to be performed. In addition and subtraction, if one operand has a zero fraction, the fraction portion of the answer will be the same as the non-zero fraction operand. In the computer, a zero fraction operand has

no effect on the operation; the arithmetic is performed, allowing normalization of the non-zero operand fraction, if specified. If both operands contain a zero fraction, the answer has no meaning and can not be normalized, so the AC and MQ are reset to contain normal zeros.

In multiplication, a zero fraction has a different meaning and is treated differently. A zero fraction multiplicand results in a product containing a zero fraction: anything times zero equals zero. Likewise, a zero raised to some power is still zero; thus the operation is not performed as the result would be meaningless. Also, a zero fraction can not be normalized. Consequently, in a single-precision multiplication, a zero fraction multiplicand causes the operation to be terminated and the AC and MQ registers (both characteristic and fraction) to be reset (a normal zero condition). However, the sign of this normal zero will be set plus or minus as determined by the algebraic sign of the product.

Effectively, a multiplier with a zero fraction has the same meaning as a multiplicand with a zero fraction: the result fraction will be zero. In a normalized single-precision floating multiply (FMP) with a zero fraction multiplier, the AC and MQ registers are reset to a normal zero with an affixed algebraic sign. In an unnormalized single-precision floating multiply (UFM) with a zero fraction multiplier, the fractions are not multiplied but the characteristics are added and the product (AC and MQ) has the properly signed characteristics with zero fractions.

In division, the divisor or the dividend could contain a zero fraction. Each case has a different meaning and is treated differently. If the divisor has a zero fraction, the quotient cannot be determined; a divide check condition results and the operation is ended. The dividend, however, remains unaltered in this case. When the dividend contains a zero fraction, the quotient will be zero and the operation is ended. However, in this case, the associated characteristic positions of the AC and MQ registers, which hold the result of a division, are cleared.

The preceding discussion pertains only to zero fraction operands. In multiplication and division, zero fraction results are due to zero fraction operands and have already been covered. In addition and subtraction, a zero fraction result is possible with non-zero fraction operands. On a normalized addition or subtraction, a zero fraction result causes the AC and MQ characteristics to be reset to zero. A zero fraction result of a unnormalized add or subtract does not reset the characteristics, but sets the MQ characteristic 27_{10} less than the AC characteristic.

Arithmetic Operations

Addition of floating-point numbers is done by adding the fractions of floating-point numbers that have equal characteristics. The characteristics are set equal before the addition by placing the number with the smallest characteristic in the AC. The AC fraction is then shifted right the number of places equal to the difference between the SR and AC characteristics (Δ). Bits shifted out of AC(35) enter MQ(9), and bits shifted out of MQ(35) are lost. After shifting stops, the AC and SR fractions are added. 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(9). If AC(9) does contain a 1, the operation would be complete; if not, the AC would shift left until a 1 did appear in AC(9). 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 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, 200_8 must be subtracted from the characteristic. The most significant part of the product is in the AC and the least significant part is in the MQ.

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. The quotient appears in the MQ and the remainder of the dividend in the AC. The remainder characteristic will equal the original dividend characteristic -27_{10} unless a quotient equal to or greater than one condition ($Q \geq 1$) existed in E time. When $Q \geq 1$ in E time, the remainder characteristic will equal the original dividend characteristic -26_{10}.

Floating-Point Controls

Because of the additional operations performed in floating-point arithmetic, several control devices are necessary. Adder separation circuits; a two-stage tally counter; seven floating-add control triggers; a four-stage double-precision synchronizer; and a characteristic checking circuit are used to control the operations.

Adder Separation

To separate the characteristic and fraction during a floating-point instruction, the ability to send the AD(9) carry to AD(8) is blocked; Systems 03.01.02.1 (4D). On a floating-point instruction, the minus output of the above block causes input to Systems 02.14.27.1 (1G), pin Q, to go plus; thus blocking the "one to AD(9P)" that is used during fixed-point operation. Systems 02.14.27.1 (2G) shows the four ways an AD(9) carry (fraction carry) can increase the characteristic on floating-point instructions.

Tally Counter

The tally counter (TC), which is normally reset to one, is used for both single and double-precision floating divide instructions. The final quotient characteristic is computed on the first L cycle of such an instruction after which the TC is stepped to a value of two. This step occurs on the first L7(D1) pulse on Systems 02.10.20.1 (3B). When the instruction operation is ended, the TC is reset to one with an A1(D1) pulse on Systems 08.00.32.1 (3F).

FACT Triggers

The seven floating-add control triggers (FACT 1-7) are used for various operations of single-precision floating-point addition, subtraction, and multiplication. The FACT triggers are also used in the control of all double-precision floating-point arithmetic instructions. Following is a list of the FACT triggers and their main functions:

FACT 1: used for fraction alignment as determined by the characteristic difference (Δ).

FACT 2: controls addition of the operands and sets the SR characteristic into the AC.

FACT 3: complements the AC or adds one to its contents, as determined by a fraction carry with unlike signs during the previous FACT 2.

FACT 4: controls normalization, including adjustment of the characteristic.

FACT 5: controls fraction overflow shifting, MQ characteristic development, and end operation. FACT 5 is used by all floating-point instructions excepting single-precision floating divide and floating round.

FACT 6: controls complementing of the MQ fraction for cases of unlike signs, where the SR fraction was greater than the AC fraction, and the MQ fraction contains significant data.

FACT 7: controls placement of the MQ data, which was corrected in FACT 6, back into the MQ.

All FACT triggers are normally in the reset (off) condition. See Figure 1 for FACT usage.

Type of Instruction	FACT Number							
	1	2	3	4	5	6	7	3&7
FAD, FSB	A	C	A	A	B	A	A	
DFAD, DFSB	C	C	A	A	B	A	A	A
FMP					C			
DFMP					C			
DP FP Divide		C	A	A	C	A	A	

A = Possible
B = Always
C = Always, unless E-End Op Condition

Figure 1. Fact Usage Chart

Double-Precision Sync (DPS)

The double-precision sync controls the different phases of all double-precision FP instructions. When DPS = 0, various gates for single-precision FP operations are operative, so it may be said that the DPS controls both single and double-precision instructions. The operations performed under DPS control are explained in the general descriptions of the three main types of double-precision instructions.

Floating-Point Trap

During the execution of floating-point instructions, the resultant characteristic in the AC and MQ may exceed eight bit positions. The capacity is exceeded if the exponent goes above + 177_8 or below − 200_8. Over + 177_8 is termed *overflow* while below − 200_8 is termed *underflow*. Floating-point overflow or underflow (spill) can occur in either (or both) of the AC and MQ registers.

A unique system of spill identification called *floating-point trap* is used to identify the instruction and the register condition which cause a floating-point overflow or underflow. A floating-point trap is possible only when operating in floating-point trap mode. The FP trap mode trigger (Systems 02.10.71.1) is reset on, thus the computer normally operates in this mode. To leave this condition, a LFTM (leave floating-trap mode) instruction must be executed. The normal mode of operation may then be re-entered by resetting the computer or executing a EFTM (enter floating-trap mode) instruction.

When in floating-point trap mode and upon sensing an overflow or underflow, the computer puts the location plus one (of the FP instruction causing the spill) into the address portion of location 0000. An identifying code, telling whether an underflow or overflow occurred, which registers are involved, and whether the most significant result is in the AC or MQ, is put in the decrement portion of location 0000. The decrement

positions used and the meaning of a 1 bit in these positions are:

POSITION	MEANING
14	Single-precision divide (MQ register is not an extension of the AC factor.)
15	Overflow in AC or MQ, or both registers.
16	AC overflow or underflow.
17	MQ overflow or underflow.

Refer to Figure 2 for possible spill codes resulting from floating-point instructions. The following steps show how characteristic overflow and underflow is recognized and how the spill code is developed:

1. Underflow or overflow of the MQ characteristic is detected by a bit in AD(P) as the MQ characteristic is computed in the adders. MQ overflow trigger 1 is set on if AD(P) equals 1 at this time, Systems 02.10.50.1 (3C). If AD(Q) equals zero at this time, FP overflow trigger 1 is also set on to indicate it is an overflow condition, Systems 02.10.50.1 (4F).

2. During the I cycle following the FP instruction, underflow or overflow of the AC characteristic is detected by a bit in AC(P). The FP trap trigger is set on if AC(P) equals 1 at this time, Systems 02.10.50.1 (3A). If AC(Q) equals zero at this time, FP overflow trigger 1 is also set on to indicate it is an overflow condition, Systems 02.10.50.1 (3B). Note that the FP trap trigger is also turned on in the I cycle following any FP instruction that turns on MQ overflow trigger 1 (as in the preceding step 1), Systems 02.10.50.1 (3A).

3. When the FP trap trigger is on, as a result of steps 1 or 2 (or both), the trap sequence begins. Only the development of the spill codes will be discussed. Spill code development is the result of AC(P) status and the various triggers that are on as a result of steps 1 and 2.

a. Bit 14: FP divide trigger, Systems 02.10.52.1 (3B). This trigger is on whenever a single-precision divide takes place.

b. Bit 15: FP overflow trigger 1, Systems 02.10.52.1 (3D).
This trigger on as a result of AD(P) and not AD(Q) as the MQ characteristic is computed during the FP instruction; or AC(P) and not AC(Q) in the I cycle following the FP instruction.

c. Bit 16: AC(P) equals 1 in the I cycle following the FP instruction.

d. Bit 17: MQ overflow trigger 1, Systems 02.10.51.1 (3G).
This trigger on as a result of AD(P) when the MQ characteristic is computed during the FP instruction.

Single-Precision Floating-Point Addition and Subtraction

These instructions algebraically add (or subtract) the floating-point numbers in Y and in the AC placing their sum (or difference) in the AC and MQ, with the resultant characteristic in the AC and a characteristic smaller by 27_{10} in the MQ. The most significant portion of the result is found in the AC and the least significant portion in the MQ. Floating-point underflow or overflow is possible. Refer to the FAD flow chart (Figure 15) for the following discussion.

If AC(Q and P) are not equal to zero before the execution of these instructions, the result will usually be incorrect. Non-zero bits in AC(Q or P), initially interpreted as part of the AC characteristic, make the AC

FP Operation (with possible types)	Type	AC	MQ	Decrement Position 14	15	16	17
Floating Round (C)	A	—	Unfl	0	0	0	1
Single and Double-Precision: Add or Subtract (A, B, C);	B	Unfl	Unfl	0	0	1	1
Multiply (A, B, C, D)	C	Ovfl	—	0	1	1	0
Double-Precision Divide (A, B, C)	D	Ovfl	Ovfl	0	1	1	1
Single-Precision Divide (E, F, G, H)	E	—	Unfl	1	0	0	1
	F	Unfl	—	1	0	1	0
	G	Unfl	Unfl	1	0	1	1
	H	—	Ovfl	1	1	0	1

Figure 2. Floating-Point Spill Codes

characteristic larger than the SR characteristic so that the SR and AC are always exchanged during E time. During this exchange, a 1 will be placed in SR(S) position if there is a 1 in AC(S or P) so that the sign of the number may be changed. Any AC(Q) bit is lost and both AC(Q and P) are cleared when the contents of SR replace the contents of the AC.

E time objectives of a single-precision floating add (or subtract) instruction are as follows:

1. Check the AC characteristic for a possible floating-point trap condition (MQ underflow or AC overflow). If the preceding condition is possible, set the overlap conflict trigger.
2. Put storage location Y into the SR, setting the SR (S) position as determined by the specific instruction.
3. Reset the MQ.
4. Determine the characteristic difference (Δ).

a. If Δ equals 0, signs are alike, and one of the fractions is normalized (it contains a 1 in the high-order bit position); set FACT 2 to add the fractions and "end op."

b. If Δ is less than 100_8 but does not meet all of the above conditions, place the larger word into the SR and the smaller word into the AC, set Δ into the SC and if Δ is greater than 0, set FACT 1 for lining up the fractions.

c. If Δ is greater than 77_8 and the larger fraction is normalized, "end op," place the larger word into the AC and set FACT 5 to compute the MQ characteristic and sign.

d. If Δ is greater than 77_8 and the larger word is not normalized, place the larger word into the SR, reset the AC and MQ, and set FACT 2 which will add the larger fraction to zero.

To determine the characteristic difference, the 2's complement of the AC characteristic is added to the SR characteristic. Because of the complement addition of the AC characteristic to the SR characteristic, a Q carry indicates that the SR characteristic is equal to or greater than the AC characteristic. No Q carry indicates 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 SR to place the smaller word in the AC.

If Δ equals 0, signs are alike and one of the fractions is normalized; the operation can be ended in E time. The operation will be completed during E7 because the sum will always be a normalized fraction, which requires no post shifting for normalization.

If Δ is less than 100_8, a normal addition is performed with one or more L time required.

If Δ is greater than 77_8, the resulting sum of the two words is equal to the larger word because the smaller word would eventually be shifted out of the AC and the MQ. It takes 66_8 shifts to shift a bit from AC fraction position (9) out through MQ fraction position (35). Δ is checked for 77_8 rather than 66_8 because it is easier to do in the computer. To save machine time, the AC is cleared rather than to allow shifting to take place. If the larger fraction is normalized, the operation can be ended in E time because the sum will be a normalized number and the operation will be completed during E7.

5. Start FACT sequence. See Figure 3 for the FACT sequence flow chart.

FACT 1 is used to equalize the characteristics and will occur only if the characteristic difference (Δ) equals 1 or more. For correct operation, Δ should not equal more than 100_8 in order to set FACT 1; however, if AC(Q and P) are not equal to zero at E5 time, it is possible to have a Δ of 101_8 or more and to set FACT 1. This is possible as AC(Q or P) equal to 1 blocks turning on the reset add trigger. With the preceding thought in mind, it can be seen that a Δ of 377_8 is possible, with a resulting FACT 1 duration of 128_{10} clock pulses.

Equalizing the characteristics is accomplished by right shifting the AC and MQ fractions the number of places equal to Δ. With the SC equal to Δ at the start of FACT 1, the following operation occurs: On any clock pulse that the SC equals 3 or more, the AC and MQ are shifted right 2 places and the SC stepped down 2 places. On any clock pulse that the SC equals 2, the AC and MQ are shifted right 2 places, the SC stepped to zero, and FACT 2 is set. On any clock pulse that the SC equals 1, the AC and MQ are shifted right 1 place, the SC stepped to zero, and FACT 2 set.

FACT 2 controls the algebraic addition of the two fractions as follows:

1. Regardless of signs: The MQ fraction is sent into the SR for zero testing and to facilitate recomplementing the MQ in FACT 6 if necessary.
2. Signs alike: Add the SR fraction to the AC fraction and set the sum into the AC along with the SR characteristic. If a fraction carry occurs as a result of the addition, turn on the carry trigger to remember that the AC and MQ are to be shifted right 1 place in FACT 5. AD(9Q) is sent to AC(9P) to hold any fraction carry so it can be shifted back into AC(9) in FACT 5. AD(9P) is allowed to carry into AD(8) to increase the characteristic of the sum if a fraction carry occurs. On a normal instruction, with no fraction carry, and without a two-cycle add condition, FACT 4 will be set. If it is an unnormal instruction, or a normal instruction with a fraction carry, or a two-cycle add condition (pre-end op), FACT 5 will be set.
3. Signs unlike: Add the SR fraction to the 1's complement of the AC fraction and set the sum into the AC along with the SR characteristic.

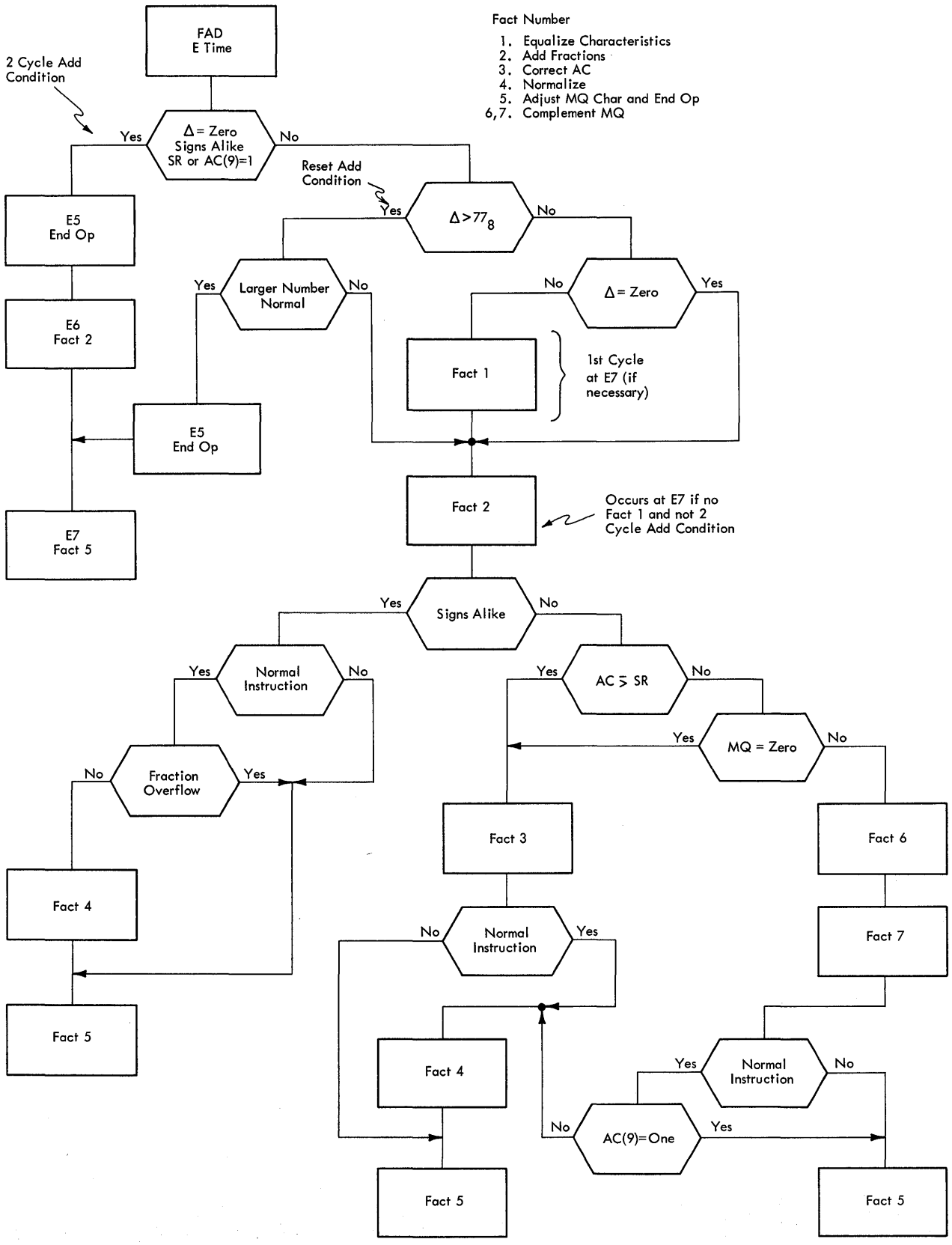


Figure 3. FAD; Fact Sequence Chart

a. With fraction carry: An AD(9P) carry turns on the carry trigger to remember that in FACT 3 the addition result in the AC is in true form. With the AC in true form, the AC sign must be made the same as the SR sign, so the SR(S) to AC(S) trigger is turned on. If the MQ fraction equals zero, FACT 3 is set to adjust the AC, making it the result of a 2's complement addition instead of a 1's complement addition. This is done by adding 1 to the AC during FACT 3. If the MQ fraction is not zero, FACT 6 is set to recomplement the MQ fraction.

b. No fraction carry: No AD(9P) carry means the addition result in the AC is in complement form, so FACT 3 is set to recomplement the AC fraction.

FACT 3 can only occur with unlike signs and is used to correct the AC contents as determined by carry trigger status. With the carry trigger off, FACT 3 recomplements the AC fraction and sets FACT 4 or 5 as determined by the specific instruction. With the carry trigger on, FACT 3 adds 1 to the true 1's complement result in the AC, thus making it a true 2's complement result. FACT 4 or 5 is then set as determined by the instruction.

FACT 4 is used to normalize the sum in the AC by shifting the AC and MQ left until AC(9) equals 1. AC characteristic underflow is possible as each left shift of one decreases the AC characteristic by one.

If AC(9) equals 1 as FACT 4 is entered, no shifting takes place and FACT 5 is set. This is possible under the three following sets of conditions:

1. Normal instruction, like signs, and no fraction carry in FACT 2.
2. Normal instruction, unlike signs, and no fraction carry in FACT 2.
3. Normal instruction, unlike signs, fraction carry in FACT 2, and MQ equal to zero. Steps 2 and 3 came from FACT 3, step 1 from FACT 2. The only other entry point to FACT 4 is from FACT 7, where AC(9) must equal zero on a normal instruction.

If the AC and MQ fractions equal zero as FACT 4 is entered, FACT 5 is set. Since AC(9 and 10) are also zero, the AC and MQ are shifted left two places and the AC characteristic is decreased by 2; however, shifting zeros has no effect and reducing the AC characteristic has no effect as FACT 5 resets the AC and MQ characteristics when the AC and MQ fractions equal zero.

On any clock pulse where AC(10) equals 1 and AC(9) equals zero, only one shift is taken; FACT 5 is set. On any clock pulse where AC(11) equals 1 and AC(9 and 10) equals zero, two shifts are taken; FACT 5 is set.

FACT 5 is used to set the pre-end op trigger if it is not already on; to set the correct sign into the AC and MQ; to adjust the MQ characteristic (or reset the AC and MQ characteristics to zero); to shift the AC and MQ fractions right 1 place if there was a fraction carry with like signs in FACT 2; and to check for a floating-point trap condition.

1. Set pre-end op trigger to end op on the next L5 pulse.
2. With unlike signs in FACT 2, if a fraction carry occurs (AC < SR), the SR(S) to AC(S) trigger is set on. In FACT 5, with the SR(S) to AC(S) trigger on, and the AC and MQ fractions are not zero, the SR(S) is set into the AC and MQ signs; or, if the preceding conditions are not met, the AC(S) is set into the MQ(S).
3. On a normal instruction, with no fraction carry in FACT 2, and when the AC and MQ fractions equal zero, the AC and MQ characteristics are reset; or, if the preceding conditions are not met, the MQ characteristic is set to a value 27_{10} less than the AC characteristic.
4. With signs alike, and a fraction carry occurs in FACT 2, FACT 5 shifts the AC and MQ right 1 place. This puts the carry bit located in AC(9P) into AC(9) thus normalizing the AC contents.
5. When the MQ characteristic is set 27_{10} less than the AC characteristic, MQ underflow is possible. FACT 5 checks for this condition and if present, sets the FAD MQ overflow trigger on.

FACT 6 is used to complement the MQ fraction and can only occur when the three following conditions are met:

1. Signs unlike (causes 1's complement addition in FACT 2).
2. AC sum in true form (fraction carry).
3. MQ fraction not equal to zero (FACT 1 occurred and shifted at least one significant bit into the MQ).

Complementing the MQ with the 2's complement is the same as making a 2's complement addition of the MQ to all zeros. The MQ contains the low-order positions of the original AC. Since the 1's complement addition of the AC gave a true result, a 2's complement addition to zeros will give the correct result for the MQ portion of the answer.

The following example shows a computer using six-bit registers, with signs unlike, and with significant bits in the MQ as a result of FACT 1. At the start of FACT 2, the register contents are:

SR 001 101 = 15_8
AC 001 010 = 12_8 (high-order position of original AC)
MQ 110 000 = 60_8 (low-order position of original AC)

FACT 2 operation	FACT 6 operation
AC 001 010	MQ 110 000
$\overline{\text{AC}}$ 110 101 (signs unlike)	$\overline{\text{MQ}}$ 001 111 } 2's complement MQ
SR 001 101	1 }
AC 000 010 = 2_8	MQ 010 000 = 20_8
9P carry MQ ≠ zero } Set FACT 6	

The final result found in the combined AC-MQ registers is: $2|20_8$.

PROOF

SR	$15|00_8 = 13|00_{10}$
AC-MQ	$12|60_8 = 10|75_{10}$
AC-MQ	$2|20_8 = 2|25_{10}$

FACT 7 always follows FACT 6 and puts the corrected MQ fraction back into the MQ. FACT 4 or 5 is then set as conditions require.

Floating Add **FAD + 0300**
(MIN I, E; MAX I, E, 8L)

The floating-point numbers in Y and the AC are algebraically added together. The most significant portion of the result appears as a normal floating-point number in the AC. The least significant portion of the result appears in the MQ as a floating-point number with a characteristic 27_{10} less than the AC characteristic. The signs of the AC and MQ are set to the sign of the larger factor. The sum in the AC and MQ is always normalized whether the original factors were normal or not. If the contents of AC(1-35) contain zeros, the FAD instruction may be used to normalize an unnormal floating-point number. Floating-point underflow or overflow is possible. Refer to the FAD flow chart (Figure 15) and to the preceding discussion on single-precision FP addition and subtraction for detailed machine operation.

Unnormalized Floating Add **UFA − 0300**
(MIN I, E; MAX I, E, 5L)

The floating-point numbers in Y and the AC are algebraically added together as in a FAD instruction. No attempt is made to normalize; thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 15.

Floating Add Magnitude **FAM + 0304**
(MIN I, E; MAX I, E, 8L)

This instruction algebraically adds the positive magnitude of the floating-point number in Y to the signed floating-point number in the AC, and normalizes the result. Floating-point underflow or overflow is possible. See Figure 15.

Unnormalized Floating Add Magnitude **UAM − 0304**
(MIN I, E; MAX I, E, 5L)

This instruction algebraically adds the positive magnitude of the floating-point number in Y to the signed floating-point number in the AC. No attempt is made to normalize; thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 15.

Floating Subtract **FSB + 0302**
(MIN I, E; MAX I, E, 8L)

This instruction algebraically subtracts the floating-point number in Y from the floating-point number in the AC and normalizes the result. Floating-point underflow or overflow is possible. See Figure 15.

Unnormalized Floating Subtract **UFS − 0302**
(MIN I, E; MAX I, E, 5L)

This instruction algebraically subtracts the floating-point number in Y from the floating-point number in the AC. No attempt is made to normalize; thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 15.

Floating Subtract Magnitude **FSM + 0306**
(MIN I, E; MAX I, E, 8L)

This instruction algebraically subtracts the positive magnitude of the floating-point number in Y from the signed floating-point number in the AC and normalizes the result. Floating-point underflow or overflow is possible. See Figure 15.

Unnormalized Floating Subtract Magnitude
(MIN I, E; MAX I, E, 5L) **USM − 0306**

This instruction algebraically subtracts the positive magnitude of the floating-point number in Y from the signed floating-point number in the AC. No attempt is made to normalize; thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 15.

Floating Round **FRN + 0760 . . . 0011**
(I, L)

Floating-point add, subtract, and multiply instructions produce a double-word result. The instruction FRN adds 1 to AC(35) if MQ(9) equals 1. (When MQ(9) equals 1, the MQ fraction is equal to or exceeds half the magnitude of a 1-bit in AC position 35.) If adding a 1 to AC(35) results in a 9P carry, the AC is corrected by adding 1 to the AC characteristic; shifting AC(9-34) right one place; and putting the 9P carry into AC(9). Floating-point overflow is possible. See Figure 17.

Single-Precision Floating-Point Multiplication

In single-precision FP multiplication, the floating-point number in the SR (multiplicand) is multiplied algebraically by the floating-point number in the MQ (multiplier). The product is placed in the AC and MQ. The characteristic of the product is contained in AC(1-8),

the most significant portion of the product fraction in AC(9-35), and the least significant portion in MQ(9-35). MQ(1-8) contains the AC characteristic minus 27_{10}. With like signs, the product signs are set plus. On unlike signs, the product signs are set minus.

Two separate operations are performed during a floating-point multiplication; the characteristics are added and the fractions are multiplied. As in all floating-point operations, the value 200_8 is the dividing line between positive and negative characteristics (exponents). Because both characteristics contain this value, 200_8 is subtracted from the sum during the characteristic addition. Fraction multiplication during floating-point arithmetic is similar to fixed-point multiplication with AD(9P,9Q) corresponding to AD(P,Q) and AC(9P) serving the same function as AC(P) in fixed-point; MQ(9,10) corresponds to MQ(1,2) in fixed-point arithmetic.

Floating Multiply **FMP + 0260**
(MIN I, E; MAX I, E, 2L)

The contents of Y are multiplied by the contents of the MQ. The most significant part of the product appears in the AC and the least significant part appears in the MQ. The product of two normalized numbers is in normalized form. If either of the numbers is not normalized, the product may be in unnormalized form. Floating-point underflow or overflow is possible. Refer to the FMP flow chart (Figure 16) for the following discussion.

1. The product signs are algebraically set whether multiplication takes place or not.

2a. If the MQ fraction (multiplier) or the SR fraction (multiplicand) are equal to zero, AC(Q-35) and MQ(1-35) are reset to zero and the computer proceeds to the next instruction.

2b. If the MQ or SR are not zero, the sum of the characteristics, minus 200_8, is placed in AC(1-8) forming the final product characteristic.

3. If multiplication takes place, SR(9-35) is multiplied by MQ(9-35). The 27 most significant bits of the 54-bit product replace the contents of AC(9-35) and the 27 least significant bits replace the contents of MQ(9-35).

4. If AC(9) equals zero after multiplication takes place, the contents of AC(10-35) and MQ(9-35) are shifted left one place and the AC characteristic is reduced by 1.

5a. After multiplication is finished AC(9-35) is tested for zero in FACT 5. If this high-order portion of the product equals zero, the AC and MQ characteristics are reset to zero.

5b. If AC(9-35) does not equal zero, the MQ characteristic is set 27_{10} less than the AC characteristic.

Unnormalized Floating Multiply **UFM – 0260**
(MIN I, E; MAX I, E, 2L)

The floating-point number in Y is multiplied by the floating-point number in the MQ. No attempt is made to normalize; thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 16. This instruction is the same as FMP with the following exceptions:

1. If the MQ fraction (multiplier) is zero, the preend op trigger is not set, as on a FMP instruction, but instead the SC is reset to zero. The SC equal to zero prevents fraction multiplication but does allow the addition of the characteristics.

2. No attempt is made to normalize the multiplication result.

3. If AC(9-35) equals zero after multiplication is finished, FACT 5 does not reset the AC and MQ characteristics as in FMP, and the MQ characteristic is set 27_{10} less than the AC characteristic.

Single-Precision Floating-Point Division

In single-precision FP division, the floating-point number in the AC (dividend) is divided by the floating-point number in storage (divisor) as designated by the address of the instruction. The MQ register (S-35) is initially reset, whereas this is not true in fixed-point or double-precision division. In floating-point division, the fraction of the dividend is divided by the fraction of the divisor to obtain the quotient fraction of the result. Division of the fractions is similar to fixed-point division. A floating-point remainder of the dividend, if any, is left in the AC. The sign of the quotient (MQ) is set algebraically as determined by the signs of the AC and SR. The sign of the remainder (AC) is the same as the sign of the original dividend unless the dividend fraction was zero, in which case the AC sign is set positive.

The exponents in a division are subtracted. Therefore, the characteristic of the MQ (quotient) is determined by subtracting the SR characteristic (divisor) from the AC characteristic (dividend). The extra 200_8 in each exponent is lost in the subtraction of the AC and SR characteristics so 200_8 must be added to the final MQ characteristic. The characteristic of the remainder in the AC is set 27_{10} less than the original dividend characteristic.

If the initial factors are normalized floating-point numbers, the quotient will also be normal. However, no attempt is made to normalize the result.

The dividend fraction (AC) cannot be twice as large as the divisor fraction (SR). If it is two or more times greater, the divide check indicator is turned on, divi-

sion does not take place and the contents of the AC will remain unchanged.

If the dividend fraction, AC(9-35), is zero, actual division does not take place, and the AC is reset to a normal zero, with a plus sign. If a divide check condition existed along with the AC fraction being zero, the divide check indicator is turned on in addition to resetting the AC.

Division of the fractions is done in the same manner as a fixed-point division except only 27_{10} reductions are attempted in floating-point because the fraction is contained in 27_{10} positions.

Single-precision floating divide uses the first step of the tally counter to develop the final quotient characteristic. The minimum cyclic time of a single-precision floating divide is I,E where division does not take place, or a maximum I,E, and 4L cycles when division does take place. One reduction attempt is performed each clock pulse, starting with the first L0 pulse. The last reduction occurs on the fourth L2 pulse at which time the shift counter is stepped to zero. See Figure 21.

Floating Divide or Halt (MIN I, E; MAX I, E, 4L) FDH + 0240

The floating-point number in the AC is divided by the floating-point number in Y. The quotient appears in the MQ and the remainder in the AC. The quotient is in normal form if both the dividend and divisor are in normal form. Floating-point underflow or overflow is possible. See Figure 21 for the following discussion.

1. The MQ sign is set to the algebraic sign of the quotient under all conditions.

2. Unless the following step 3 occurs, the AC sign remains unchanged, so that the signs of the remainder and dividend always agree.

3. If the AC fraction (dividend) equals zero, the AC sign and the AC and MQ characteristics are set to zero. The computer then proceeds to the next instruction, unless a divide check condition exists, in which case the computer halts with the divide check light on. (Note: With the AC fraction equal to zero, a divide check condition can occur only if the SR fraction is also zero.)

4. If the AC fraction is equal to or greater than twice the divisor fraction (AC $\geq$ 2·SR), the divide check light turns on and the computer halts. The dividend is left unchanged and the MQ is left a signed normal zero.

5. If division does take place, the quotient characteristic is set into MQ(1-8). This characteristic value is the result of subtracting the SR characteristic from the AC characteristic and then adding 200_8 to the difference. Refer to step 7.

6. After division is completed, the original dividend characteristic minus 27_{10} is set into AC(1-8), thus setting the remainder characteristic. Refer to step 7.

7. In steps 5 and 6, the characteristic values may be 1 higher than stated. If so, it is the result of the AC being equal to or greater than the SR before the first divide reduction cycle. This condition is referred to as a quotient equal or greater than one (Q $\geq$ 1) condition.

On a Q $\geq$ 1 condition, the AC characteristic is increased by 1 before any characteristic computing is done. Increasing the AC characteristic by 1 is effectively the same as shifting the AC left 1 place. If the Q $\geq$ 1 condition does not exist, the AC and MQ are shifted left 1 place before divide reduction cycles take place.

Floating Divide or Proceed (MIN I, E; MAX I, E, 4L) FDP + 0241

The floating-point number in the AC is divided by the floating-point number in Y. This instruction operates the same as the FDH instruction except that on a divide check condition, the computer does not halt but proceeds to the next instruction with the divide check light on. The divide check condition may be tested at some later time by the DCT instruction. See Figure 21.

Double-Precision Floating-Point Arithmetic

The purpose of floating-point arithmetic is to improve the handling of very large or very small numbers rapidly and accurately. When a fixed-point fraction is changed to floating-point form, the resulting fraction may exceed 27 bits. In the 7094-II, circuits which accommodate the longer-length fraction are used when operating with double-precision instructions. Double-precision doubles the fraction-handling capacity thereby doubling the precision of the result. The end result of a double-precision floating-point number is a product consisting of characteristic and fraction, the fraction being 54 bits long.

In all double-precision arithmetic instructions, the most significant 27 bits of the answer are contained in the AC and the least significant in the MQ. The characteristic is contained in AC(1-8), and the characteristic −27 is contained in MQ(1-8).

When a double-precision instruction is referenced to a location in memory (Y), that location will be placed in the storage register (SR) and the next location (Y+1) will be placed in the instruction backup register (IBR). This double-precision number, consisting of two sequential memory locations, is called the *addressed operand.*

The first memory location referenced (Y) contains the addressed operand sign, characteristic, and high-order fraction. The second memory location is automatically referenced and will be one address higher than the first location referenced. It is assumed that the sign of the second location equals the sign of the first location and that the characteristic equals the characteristic −27 of the first location. Only the fraction bits of the second location are used; these bits form the low-order fraction of the addressed operand. The addressed operand is placed in the storage register (SR) and the instruction backup register (IBR).

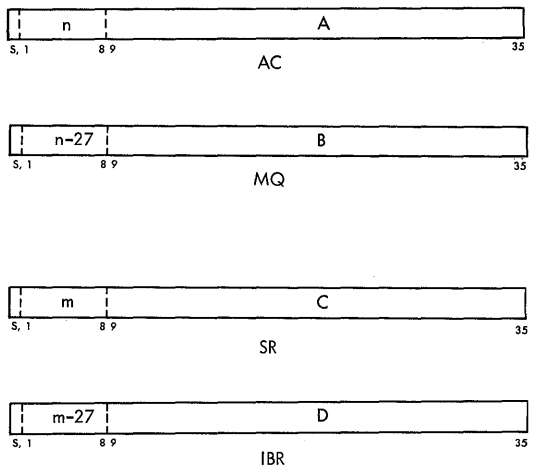
The second double-precision floating-point number is called the implied operand and will be initially located in the AC and MQ. This implied operand consists of the sign, characteristic, and high-order fraction located in the AC, and the low-order fraction located in the MQ. The MQ sign is assumed equal to the AC sign, and the MQ characteristic is assumed equal to the AC characteristic −27. The most significant portion of the implied operand will be in the AC and the least significant in the MQ.

The MQ is assigned a characteristic 27_{10} less than the AC characteristic because the fraction contained in MQ(9-35) is displaced 27 positions to the right of the accumulator binary point, which is the point just to the left of AC(9).

On all double-precision instructions, the sense indicator (SI) register is used for temporary storage during register exchanging and various addition, multiplication, and divide operations. This causes the destruction of any information contained in the SI register at the beginning of a double-precision instruction.

Since the IBR is used for all double-precision instructions, overlap is not possible during their E or L cycles.

The format at the start of a double-precision instruction, showing register contents, is as follows:



Where:

1-8 contains the characteristic, and 9-35 the fraction.

AC = A · 2^n, the most significant part of the implied operand.

MQ = B · 2^{n-27}, the least significant part of the implied operand.

SR = C · 2^m, the most significant part of the addressed operand.

IBR = D · 2^{m-27}, the least significant part of the addressed operand.

For convenience, regroup the numbers into characteristic and fraction where:

A = A characteristic and A fraction (A · 2^n)

B = B characteristic and B fraction (B · 2^{n-27})

C = C characteristic and C fraction (C · 2^m)

D = D characteristic and D fraction (D · 2^{m-27})

Double-Precision Floating-Point Addition and Subtraction

These instructions add (or subtract) $A \cdot 2^n + B \cdot 2^{n-27}$ to $C \cdot 2^m + D \cdot 2^{m-27}$ and place their sum (or difference) in the AC and MQ with the resultant characteristic in the AC and a characteristic smaller by 27_{10} in the MQ. These instructions assume that $A \cdot 2^n$ and $B \cdot 2^{n-27}$ have been previously placed in the AC and MQ respectively and that $C \cdot 2^m$ and $D \cdot 2^{m-27}$ are in consecutive locations in memory. For a simplified flow chart of these operations, refer to Figure 4.

Two sets of controls are used in double-precision addition and subtraction. The floating-add control triggers (FACT 1-7) are used as in single-precision but with certain modifications. The double-precision sync (DPS) modifies FACT operations and also controls data movement before and after additions. FACT 4 is the only FACT not modified by the DPS.

DPS 0 controls the E time and first L0 movements of data in preparation for the first add. At the end of L0 time, the larger of the two operands will be in the SI and SR, with the smaller operand in the MQ and AC. They will appear in one of the following two forms.

	SI	SR	AC	MQ	Figure Reference
1.	C	$D \cdot 2^m$	$B \cdot 2^n$	A	Figure 18, Chart 1
2.	A	$B \cdot 2^n$	$D \cdot 2^m$	C	Figure 18, Chart 2

At the end of L0, FACT 1 is set for pre-normalization and the DPS is stepped to one. An E time end op may occur if the characteristic difference is greater than 77_8 and the larger of the two operands is normalized. When this occurs, the larger of the two operands is placed in the AC and MQ and FACT 5 is set to complete the operation. See Figure 18, Charts 3 and 4.

DPS 1 controls pre-normalization (FACT 1) and first add (FACT 2). During FACT 1, MQ(35) is shifted to AC(9) and the AC(35) to MQ(9) gates are blocked. During FACT 2, a two's complement add is performed if the signs are unlike instead of the one's complement add of single-precision. Single-precision controls place the contents of MQ(9-35) in the SR during FACT 2. The AC and SR fractions are exchanged following FACT 2 and the DPS is stepped to complete the operand relocation in preparation for the second addition. Figure 18 (Chart 5) shows all the register exchanges that occur when the DPS is not zero. DPS 2 is used to complete the operand relocation in preparation for the second add. SI(9-35) and SR(9-35) are exchanged, the DPS is stepped and FACT 2 is set for the second addition.

DPS 3 controls the second add, MQ adjust, normalization, and end operation functions. During FACT 2, any AD(9) carries which were generated under DPS 2 control are now treated as carries into AD (35). If a true add is being performed, the operation will proceed to FACT 4 for normalization or to FACT 5 if normalization is not required. All single-precision decisions during these controls are valid. If a complement add is being performed in FACT 2, the single-precision decisions do not apply as they are based on a one's complement addition. If a 9 carry occurs during this complement add, it indicates that the answer in both the MQ and AC are in true form except that their contents must be checked for zero. If this condition occurs, FACT 4 is set for post normalization. If a complement add is being performed and an AD(9) carry does not occur, it indicates that the MQ and AC are in two's complement form and must be corrected. MQ(9-35) is set into SR(9-35) by the single-precision controls of FACT 2 and FACT 6 is set. During FACT 6, the complement of SR(9-35) is gated to AD(9-35) along with a carry to AD(35) and AD(9-35) is gated to SR(9-35). The carry trigger is turned on if a carry resulted and FACT 7 and FACT 3 are set. During FACT 7, SR(9-35) is gated to MQ(9-35) and comp AC(9-35) is gated to AD(9-35) along with the carry trigger. AD(9-35) is then gated to AC(9-35) and FACT 4 or 5 is set, depending on normalization requirements. The instruction is then completed under single-precision controls. See Figure 19.

Double-Precision FP Add (MIN I, E; MAX I, E, 11L) — DFAD + 0301

The double-precision floating-point number in Y and Y + 1 is algebraically added to the double-precision floating-point number in the AC and MQ. The most significant portion of the result appears in the AC, and the least significant in the MQ. The sign of the result is the sign of the larger operand. The result is always normalized whether the original operands were normal or not. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision Unnormalized FP Add (MIN I, E; MAX I, E, 8L) — DUFA − 0301

The double-precision floating-point number in Y and Y + 1 is algebraically added to the double-precision floating-point number in the AC and MQ. No attempt is made to normalize, thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision FP Add Magnitude (MIN I, E; MAX I, E, 11L) — DFAM + 0305

The positive magnitude of the double-precision floating-point number in Y and Y + 1 is algebraically added to the signed double-precision floating-point number in the AC and MQ. The result is normalized. Floating-point underflow or overflow is possible. See Figure 19.

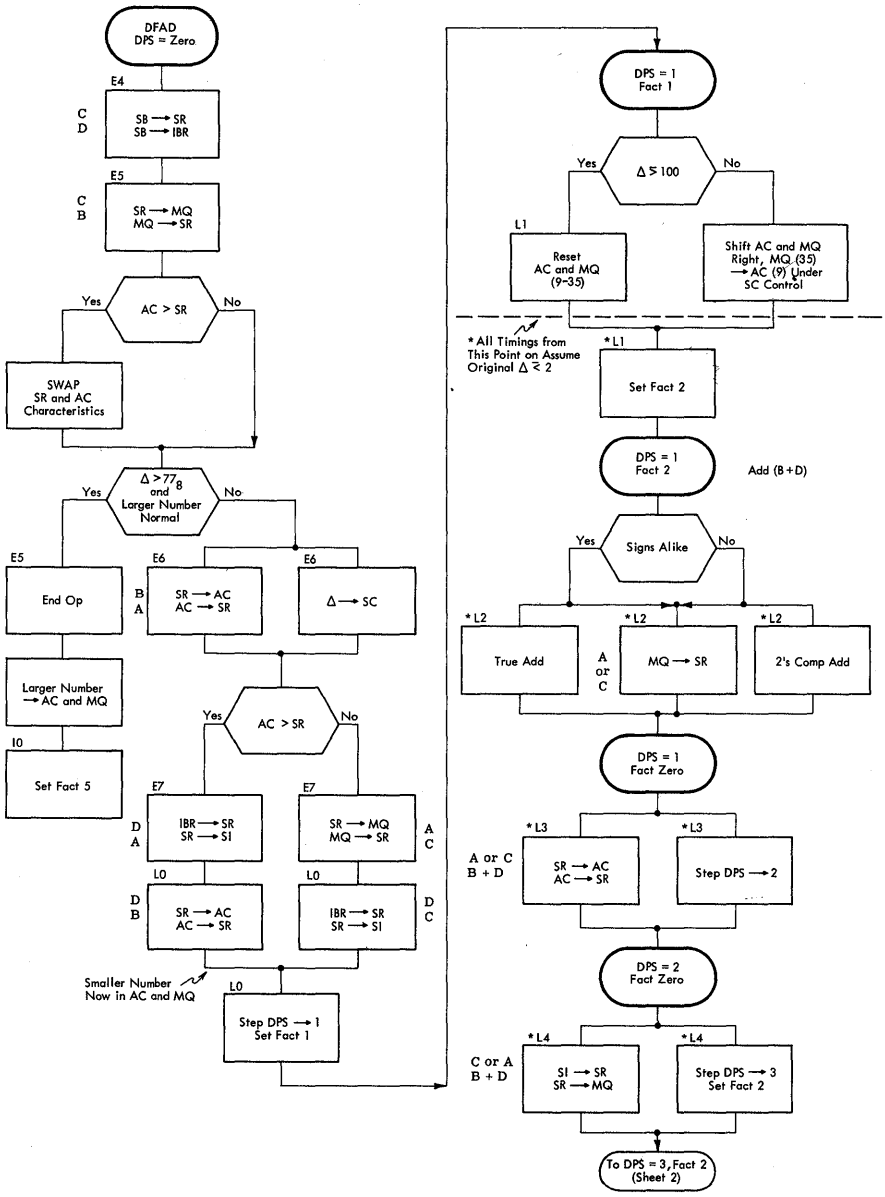


Figure 4. DFAD; Simplified Flow Chart—Sheet 1 of 2

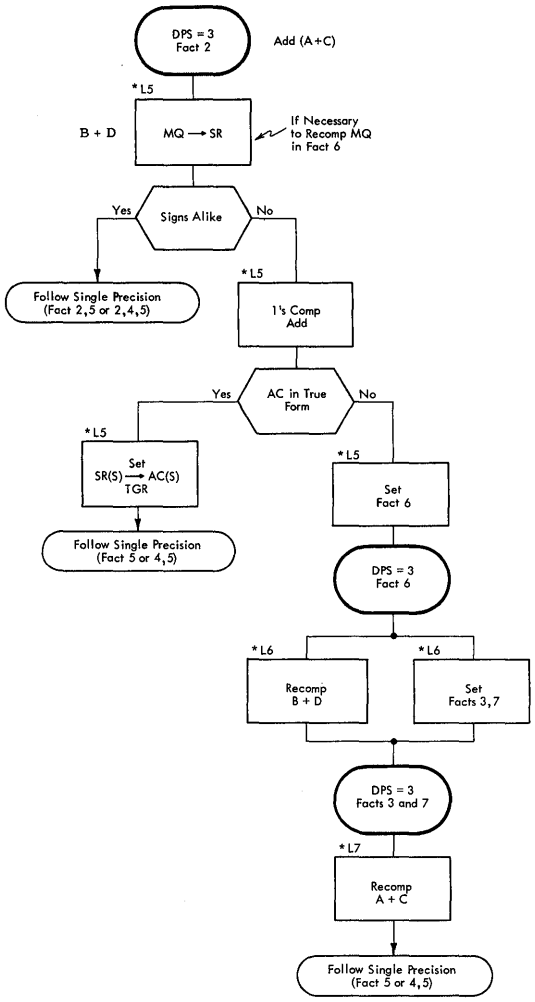


Figure 4. DFAD; Simplified Flow Chart—Sheet 2 of 2

Double-Precision Unnormalized FP Add Magnitude
(MIN I, E; MAX I, E, 8L) DUAM – 0305

The positive magnitude of the double-precision floating-point number in Y and Y + 1 is algebraically added to the signed double-precision floating-point number in the AC and MQ. No attempt is made to normalize, thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision FP Subtract DFSB + 0303
(MIN I, E; MAX I, E, 11L)

The double-precision floating-point number in Y and Y + 1 is algebraically subtracted from the double-precision floating-point number in the AC and MQ. The result is normalized. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision Unnormalized FP Subtract
(MIN I, E; MAX I, E, 8L) DUFS – 0303

The double-precision floating-point number in Y and Y + 1 is algebraically subtracted from the double-precision floating-point number in the AC and MQ. No attempt is made to normalize, thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision FP Subtract Magnitude
(MIN I, E; MAX I, E, 11L) DFSM + 0307

The positive magnitude of the double-precision floating-point number in Y and Y + 1 is algebraically subtracted from the signed double-precision floating-point number in the AC and MQ. The result is normalized. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision Unnormalized FP Subtract Magnitude
(MIN I, E; MAX I, E, 8L) DUSM – 0307

The positive magnitude of the double-precision floating-point number in Y and Y + 1 is algebraically subtracted from the signed double-precision floating-point number in the AC and MQ. No attempt is made to normalize, thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 19.

Double-Precision Floating-Point Multiplication

The object of a double-precision floating-point multiply instruction is to multiply the addressed operand located in Y and Y + 1 by the implied operand in the AC and MQ.

If the multiplier is denoted by $A \cdot 2^n + B \cdot 2^{n-27}$ and the multiplicand is denoted by $C \cdot 2^m + D \cdot 2^{m-27}$ the following arithmetic operation is performed.

$$(C \cdot 2^m + D \cdot 2^{m-27}) \cdot (A \cdot 2^n + B \cdot 2^{n-27})$$

The computer performs two multiplications, an add, a third multiplication, a second add, and ends with the answer in the AC and MQ. A possible fourth multiplication (B · D) is not performed as the product is insignificant with a 54-bit fraction.

During E time, A and B, C and D, and A and C are tested to see if they are equal to zero. A zero combination of any of these will result in a zero product, so the operation is terminated at the end of E time and the registers are set to a normal zero. The signs of the product are also set during E time and the fractions are moved in preparation for the first multiply (B · C).

DPS 0 controls the first multiply and the following register swap in preparation for the second multiply (A · D).

DPS 1 controls the second multiply and the following register swaps in preparation for the third multiply (A · C). The addition of (B · C) + (A · D) is also done during the register swapping. If a carry results from

this addition, it is remembered by stepping the DPS one extra time.

DPS 2 OR 3 controls the third multiply (A · C). If the DPS equals 3 at the end of the third multiply, it means a carry occurred on the sum of (B · C) + (A · D) and must be added to the final product. If no carry occurred, the DPS will equal 2 at the end of the third multiply with the product in its final form except for normalizing which is done during DPS 2 or 3 if required. FACT 5 controls the MQ characteristic computation and the zero testing of the fractions.

The DFMP flow chart (Figure 20) can be briefly summarized as follows:

1. Add characteristics of the two floating-point operands (m + n − 200).
2. Multiply the original MQ fraction by the fraction of Y (B · C).
3. Multiply the original AC fraction by the fraction of Y + 1 (A · D).
4. Add the products of steps 2 and 3 (B · C + A · D).
5. Multiply the original AC fraction by the fraction of Y (A · C).
6. Add the product of step 5 to the sum of step 4 [A · C + (B · C + A · D)].
7. Adjust MQ characteristic.

Three types of instructions use floating-point multiply cycles (Figure 16, sheet 2 of 2) to obtain their desired result: single and double-precision multiplication, and double-precision division. The timings involved are shown in Figure 5.

33_8 → SC (14 Clk Pulses Each Mpy)		
Instruction Type	Turn On Mpy Cycle Tgr #1	Step SC By 1 → Zero
Single-Precision Multiply	E6	2nd L4
Double-Precision Multiply		
1st Mpy (B·C)	E6	2nd L4
2nd Mpy (A·D)	3rd L3	5th L1
3rd Mpy (A·C)	5th L7	7th L5
Double-Precision Divide (Q1·D)	5th L0	6th L6

Figure 5. Timing of Floating-Point Multiply Cycles

Double-Precision FP Multiply (MIN I, E; MAX I, E, 7L) DFMP + 0261

The double-precision floating-point number in Y and Y + 1 is multiplied by the double-precision floating-point number in the AC and MQ. The most significant part of the product appears in the AC and the least significant in the MQ. The product of two normalized numbers is in normalized form. If either number is not normalized, the product may be in unnormalized form. Floating-point underflow or overflow is possible. Refer to Figure 20 for the following steps:

1. The product signs are algebraically set whether multiplication takes place or not.

2a. If the multiplier (A, B) or the multiplicand (C, D) is a normal zero, or the high-order fractions of both (A, C) are zero, the AC and MQ are reset to a signed normal zero and the computer proceeds to the next instruction.

2b. If step 2a is not true, three multiplications take place: (B · C), (A · D), and (A · C). The low-order parts of the products (B · C) and (A · D) are discarded and the remaining high-order parts are added to the low-order part of the product (A · C).

3. At the end of step 2b, if the high-order part of the final product is not a normal number, the AC and MQ fractions are shifted left one position and the characteristic of the AC is reduced by 1.

4a. At the end of step 3, if the final product in the AC and MQ is zero, the AC and MQ characteristics are reset, giving a signed normal zero.

4b. If the final product is not zero, the AC characteristic equals the sum of the characteristics of Y and the original AC, minus 200_8. The MQ characteristic will equal the AC characteristic minus 27_{10}.

Double-Precision Unnormalized FP Multiply (MIN I, E; MAX I, E, 7L) DUFM − 0261

The double-precision floating-point number in Y and Y + 1 is multiplied by the double-precision floating-point number in the AC and MQ. No attempt is made to normalize, thus the result may be an unnormal number. Floating-point underflow or overflow is possible. See Figure 20. This instruction is the same as DFMP with the following exceptions:

1. If the AC and MQ fractions (multiplier) are zero, the pre-end op trigger is not set as on DFMP, but instead the SC is reset to zero. The SC equal to zero prevents the first multiply (B · C) from taking place but all other operations are performed as usual. When this condition occurs, FACT 5 will be set on the sixth L7 pulse, which is one cycle earlier than under normal conditions.
2. No attempt is made to normalize the final result.
3. If the final product in the AC and MQ is zero, FACT 5 does not reset the AC and MQ characteristics as in DFMP, and the MQ characteristic is set 27_{10} less than the AC characteristic.

Double-Precision Floating-Point Division

The objective of a double-precision floating-point divide instruction is to divide the implied operand in the AC and MQ by the addressed operand located in Y and Y + 1.

If the dividend is denoted by $A \cdot 2^n + B \cdot 2^{n-27}$ and the divisor is denoted by $C \cdot 2^m + D \cdot 2^{m-27}$ the following arithmetic operation is performed:

$$\frac{A \cdot 2^n + B \cdot 2^{n-27}}{C \cdot 2^m + D \cdot 2^{m-27}}$$

This can be separated into two single-precision divide operations:

$$\frac{A \cdot 2^n + B \cdot 2^{n-27}}{C \cdot 2^m} = Q_1 \cdot 2^{n-m} + R_1 \cdot 2^{n-27}$$

where Q_1 is the high-order quotient and R_1 is the remainder from this division.

$$\frac{R_1 \cdot 2^{n-27} - Q_1D \cdot 2^{n-27}}{C \cdot 2^m} = Q_2 \cdot 2^{n-m-27} + R_2 \cdot 2^{n-54}$$

where Q_2 is the low-order quotient and R_2 is the remainder from this division. Because the fraction of R_2 is 54 bits removed from the original dividend, it is not used.

The development of the high-order quotient (Q_1) and remainder (R_1) is accomplished by performing a single-precision divide, except that the SI register is reset rather than the MQ.

The final quotient characteristic is developed and put into AC(1-8) during the first L cycle of this divide.

The development of the second quotient cannot be accomplished in the form shown in equation 2, therefore an intermediate step is necessary.

The factors Q_1 and D are multiplied together. This is accomplished by turning on the block divide trigger, placing Q_1 in the SR, D in the MQ, resetting AC(9-35), and turning on the multiply cycle trigger. The multiply cycle trigger will cause multiplication of the fraction in the SR by the fraction in the MQ and will place the product in the AC and MQ. Because the low-order digits of the product in the MQ are 54 bits removed from the original double-precision dividend, it is not used. After the multiply cycle trigger is turned off by the SC going to 0, the fraction R_1 is placed in the storage register and the factor Q_1D located in the AC is alegbraically subtracted from it. This places the fraction $R_1 - Q_1D$ in the AC.

At the end of the development of $R_1 - Q_1D$, the factor C is placed in the SR, MQ(9-35) is reset, and the block divide trigger is reset so that a second single-precision divide may take place. This time $R_1 - Q_1D$ is divided by C, which develops Q_2 in the MQ and R_2 in the AC. At the end of the second divide, the block divide trigger is again turned on, Q_1 is placed in the SR, AC(9-35) is reset, and FACT 2 is turned on. With FACT 2 on, a single-precision normalized floating-add is performed ($Q_1 + Q_2$). Normalization in FACT 4 takes place if needed. FACT 5 then computes Q_2 characteristic placing it in MQ(1-8), and sets the pre-end op trigger to end the operation. Figure 6 shows the characteristics and signs of a double-precision divide operation. See Figure 7 for a simplified flow chart of double-precision division.

CHARACTERISTICS

A	B	C	D	Q_1	R_1	Q_1D	Q_2
n	n-27	m	m-27	n-m+200	n-27	n-27	n-m+200-27

SIGNS

A+B	C+D	Q_1	R_1*	Q_1D*	$R_1 - Q_1D$		Q_2	
					$R_1 > Q_1D$	$R_1 < Q_1D$	$R_1 > Q_1D$	$R_1 < Q_1D$
+	+	+	+	+	+	-	+	-
+	-	-	+	+	+	-	-	+
-	+	-	-	-	-	+	-	+
-	-	+	-	-	-	+	+	-

* Note that R_1 and Q_1D Always Follow the Sign of A+B

Figure 6. Double-Precision FP Divide; Characteristic and Sign Determination Tables

Double-Precision FP Divide or Halt (MIN I, E; MAX I, E, 15L) DFDH – 0240

The double-precision floating-point number in the AC and MQ is divided by the double-precision floating-point number in Y and Y + 1. The quotient is a normalized double-precision floating-point number in the AC and MQ. Floating-point underflow or overflow is possible.

If the AC fraction (at the start of either divide operation) is equal to or greater than twice the divisor fraction (AC $\geq$ 2 · SR), the divide check turns on and the computer halts. Since this procedure contains two floating-point divide operations, a divide check could occur at two different times: (1) If a divide check occurs on the first division, the dividend remains unchanged and the sense indicator register is cleared. (2) If a divide check occurs on the second division, the AC fraction contains ($R_1 - Q_1D$) and the sense register contains the quotient (Q_1) of the first division.

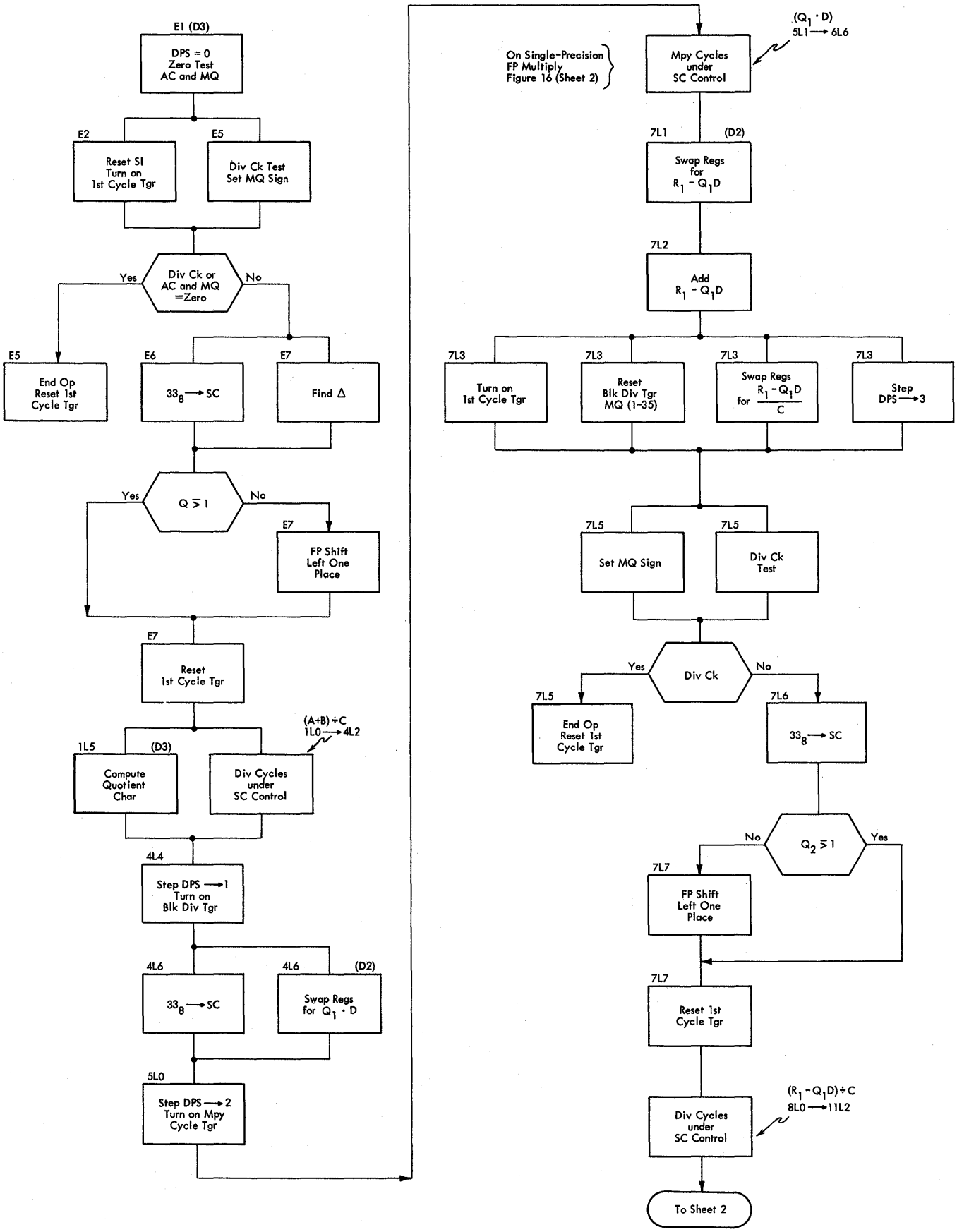


Figure 7. Double-Precision FP Divide; Simplified Flow Chart—Sheet 1 of 2

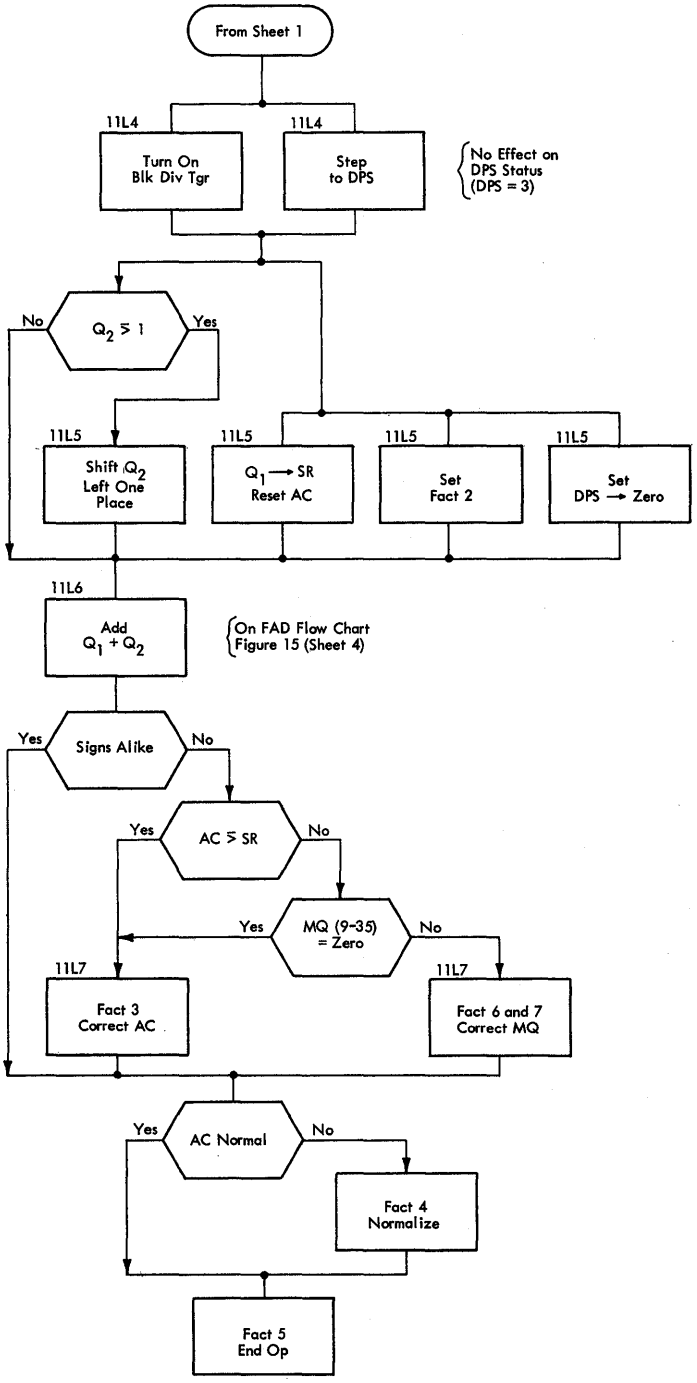


Figure 7. Double-Precision FP Divide; Simplified Flow Chart—Sheet 2 of 2

The overflow or underflow indication will be lost if a divide check occurs. See Figure 21 for the following steps:

1. Signs resulting from this instruction are always alike:
   a. Before the first divide, with or without a divide check, when A and B fractions are zero; with like signs, the AC and MQ signs are set plus; with unlike signs, the AC and MQ signs are set minus.
   b. Before the first or second divide, with a divide check only, the AC and MQ signs equal the sign of the original AC.
   c. When both divisions take place, the AC and MQ signs are the algebraic result of the original operands.
2. Quotient characteristics are developed and retained only when the dividend fractions (A,B) are not zero and a divide check does not occur:
   a. If the original AC and MQ fractions (A,B) equal zero, the AC and MQ characteristics are set to zero.
   b. If A and B fractions are not zero, and a divide check occurs before the first divide, the AC and MQ characteristics remain unchanged.
   c. If a divide check occurs before the second divide, the AC and MQ characteristics will equal zero.
   d. When none of the preceding conditions occur, a successful division takes place, and the AC characteristic will equal the characteristics of A minus C, plus 200_8. The MQ characteristic will equal the AC characteristic $- 27_{10}$.

Double-Precision FP Divide or Proceed DFDP – 0241 (MIN I, E; MAX I, E, 15L)

The double-precision floating-point number in the AC and MQ is divided by the double-precision floating-point number in Y and Y + 1. The quotient is a normalized double-precision floating-point number in the AC and MQ. This instruction is the same as DFDH except on a divide check condition. If a divide check occurs, the computer does not halt but proceeds to the next instruction. See Figure 21.

Abbreviations

AC	Accumulator Register
AD	Adder/Adders
Adr	Address
AR	Address Register
Auto	Automatic
Blk	Block (stop/bar)
c	The number of iterations to be performed in variable-length instructions (in decrement portion of instruction)
Char	Characteristic
Clk	Clock Pulse
Comp	Complement
Cond	Condition
Ctrl	Control
Div Ck	Divide Check
Dlyd	Delayed
DP	Double-Precision
DPS	Double-Precision Sync
FP	Floating-Point
Gt	Gate
IA	Indirect Address
IBR	Instruction Backup Register
Inst	Instruction
Lt	Left
Mpy	Multiply
MQ	Multiplier-Quotient Register
MST	Master Stop
Op	Operation
Ovlp	Overlap
Ovfl	Overflow
PC	Program Counter
POD	Primary Operation Decoder
Ppg	Propagate
PR	Program Register
Prec	Precision
Pri	Primary
Recomp	Recomplement
Reg	Register
Rt	Right
SB	Storage Bus
SC	Shift Counter
Sh	Shift
SI	Sense Indicator Register
SR	Storage Register
SOD	Secondary Operation Decoder
SS	Single-Shot
Stg	Storage
TC	Tally Counter
Tgr	Trigger
Unfl	Underflow
XAD	Index Adders
Y	Storage Word (referred to by address portion of current instruction)

Symbols

Δ	Characteristic Difference
$>$	(A>B) A "greater than" B
$<$	(A<B) A "less than" B
$\overline{>}$	(A$\overline{>}$ B) A "equals or greater than" B
$\overline{<}$	(A$\overline{<}$B) A "equals or less than" B
$\overline{AB}$	Complement of (AB)
*	Block location shown at exit of feeding block (used in flow charts thus: 02.13.47.1*)
A·B	A Times B (multiplication)
$\neq$	Not Equal (opposite of equal)

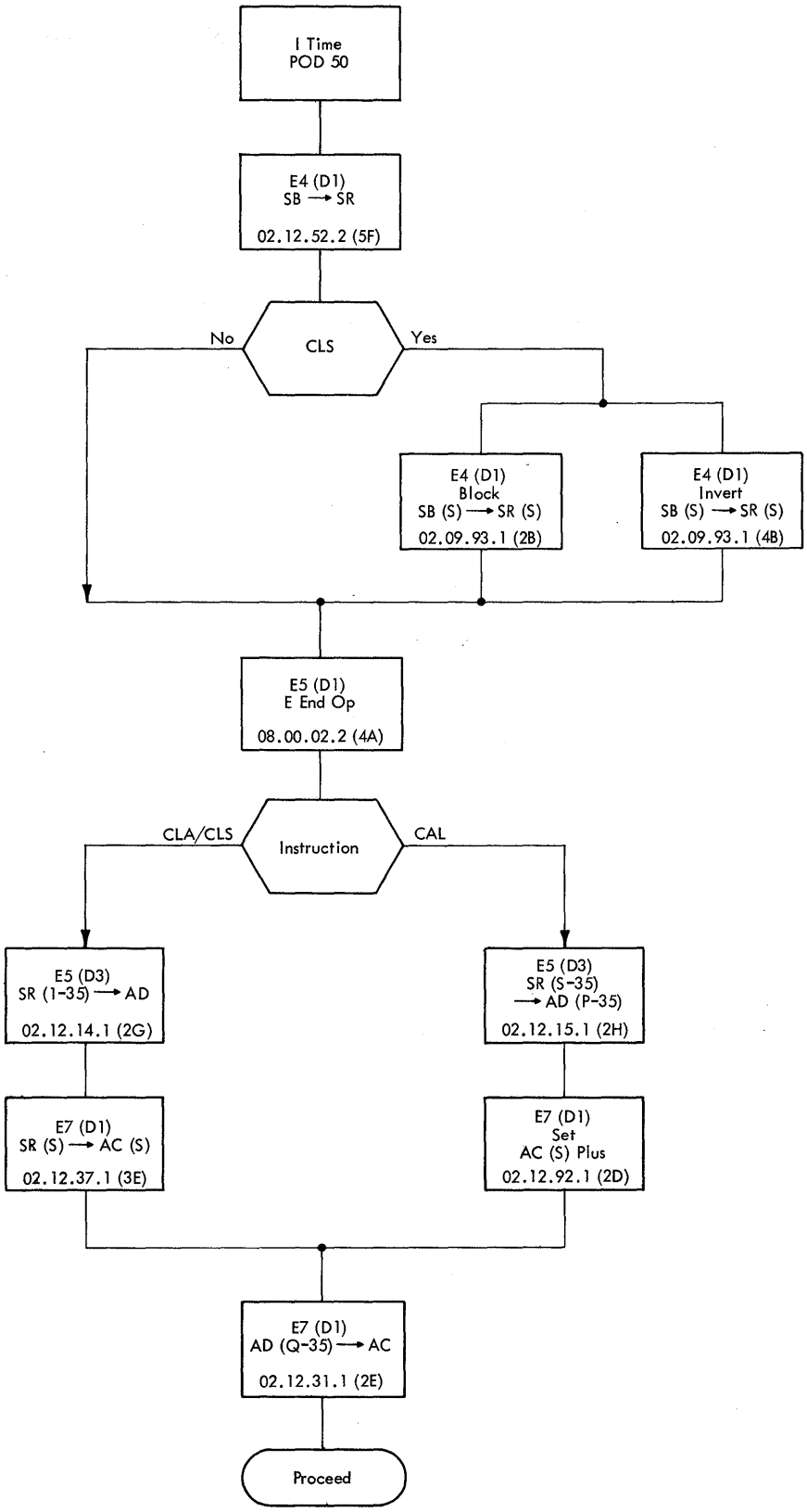


Figure 8. Clear and Add; Clear and Subtract; Clear and Add Logical Word

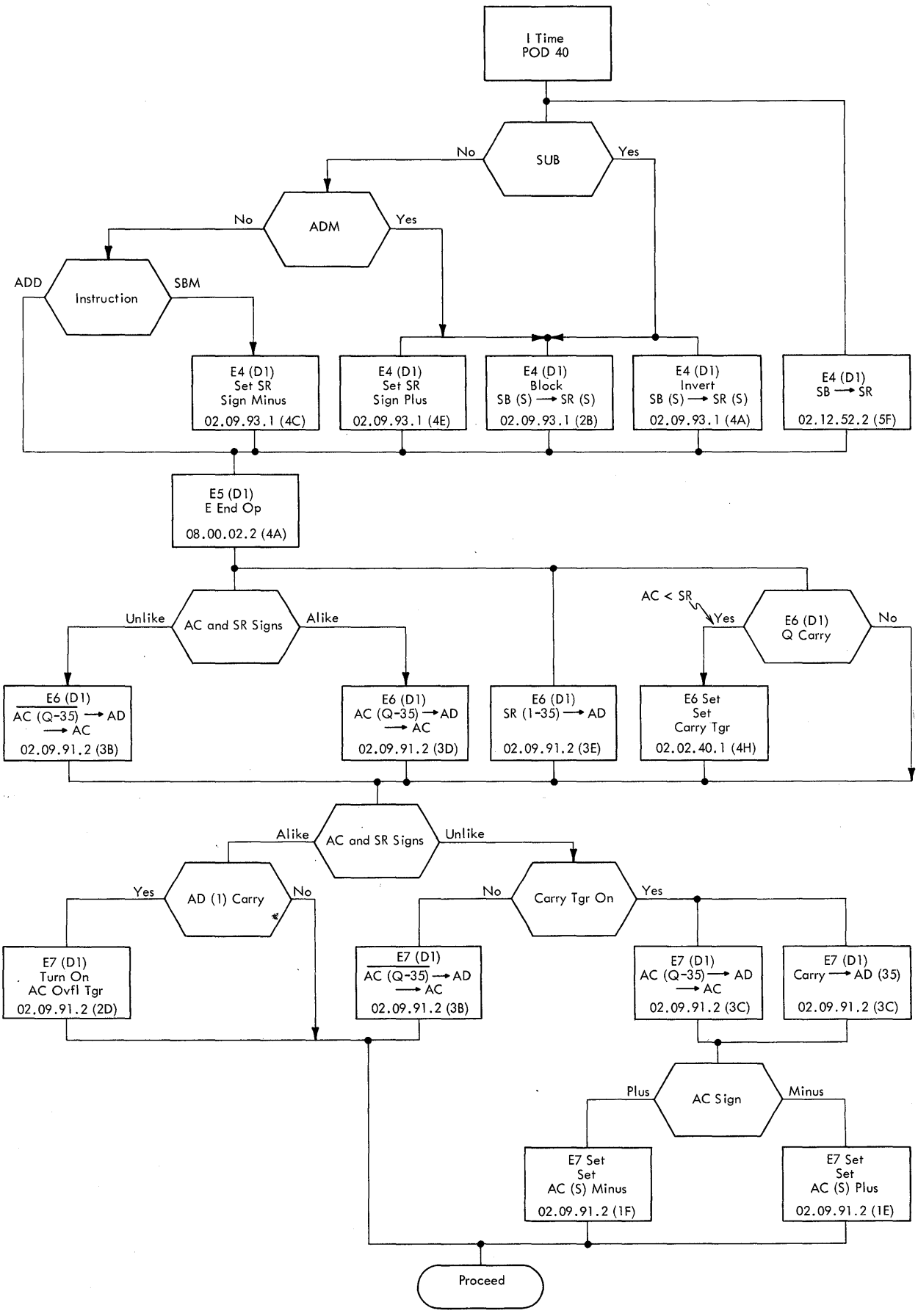


Figure 9. Add; Add Magnitude; Subtract; Subtract Magnitude

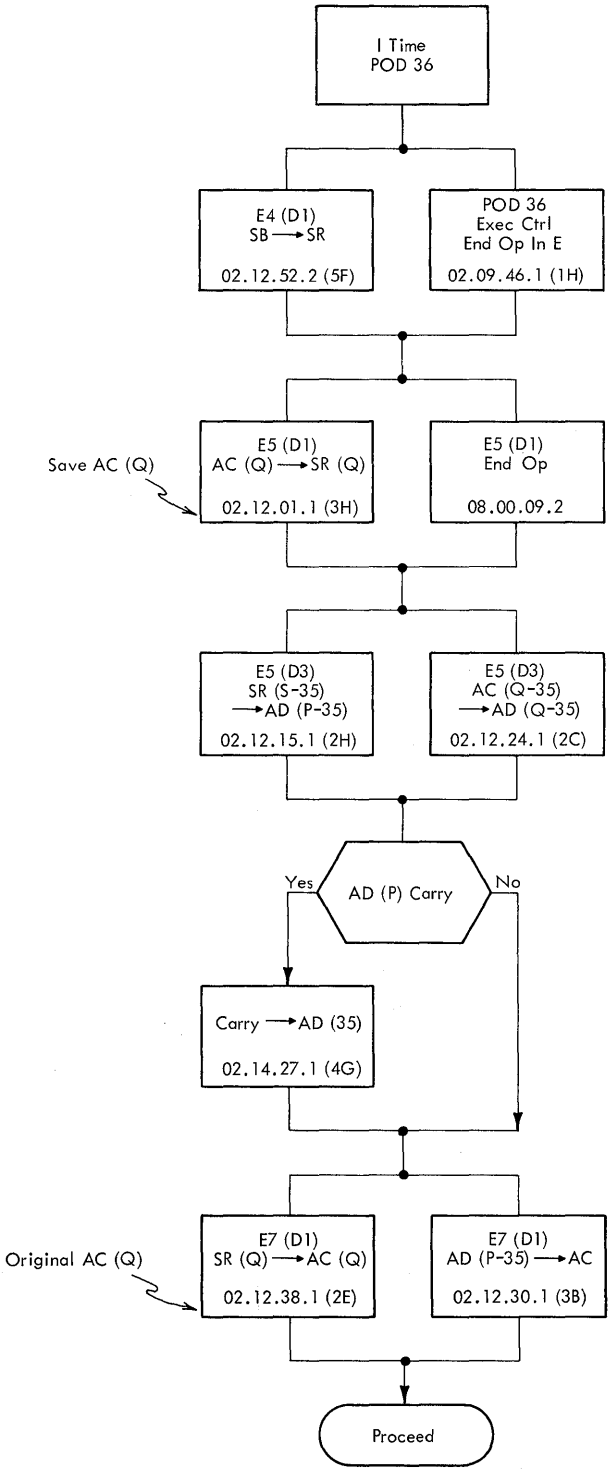


Figure 10. Add and Carry Logical Word

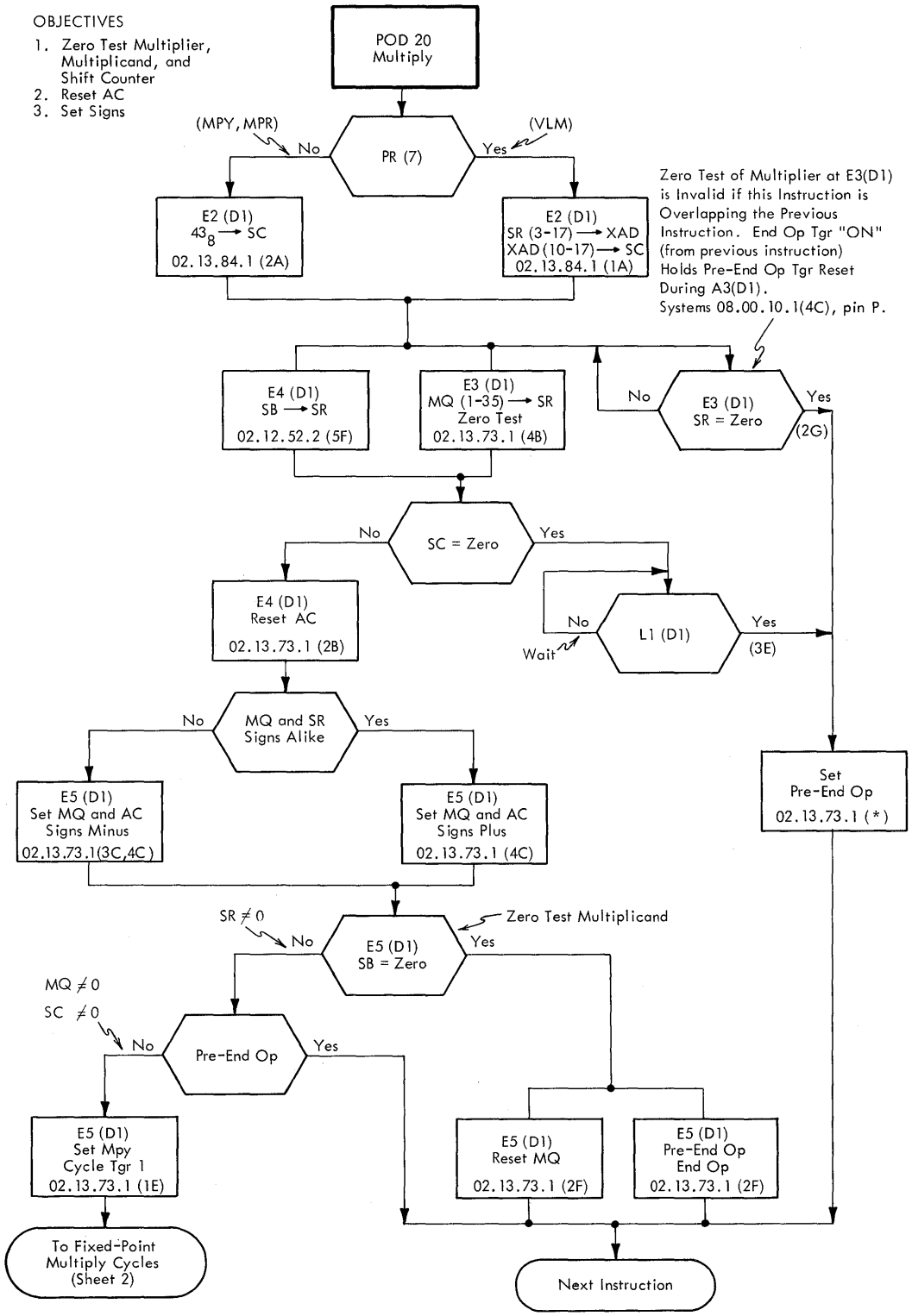


Figure 11. Fixed-Point Multiply—Sheet 1of 3

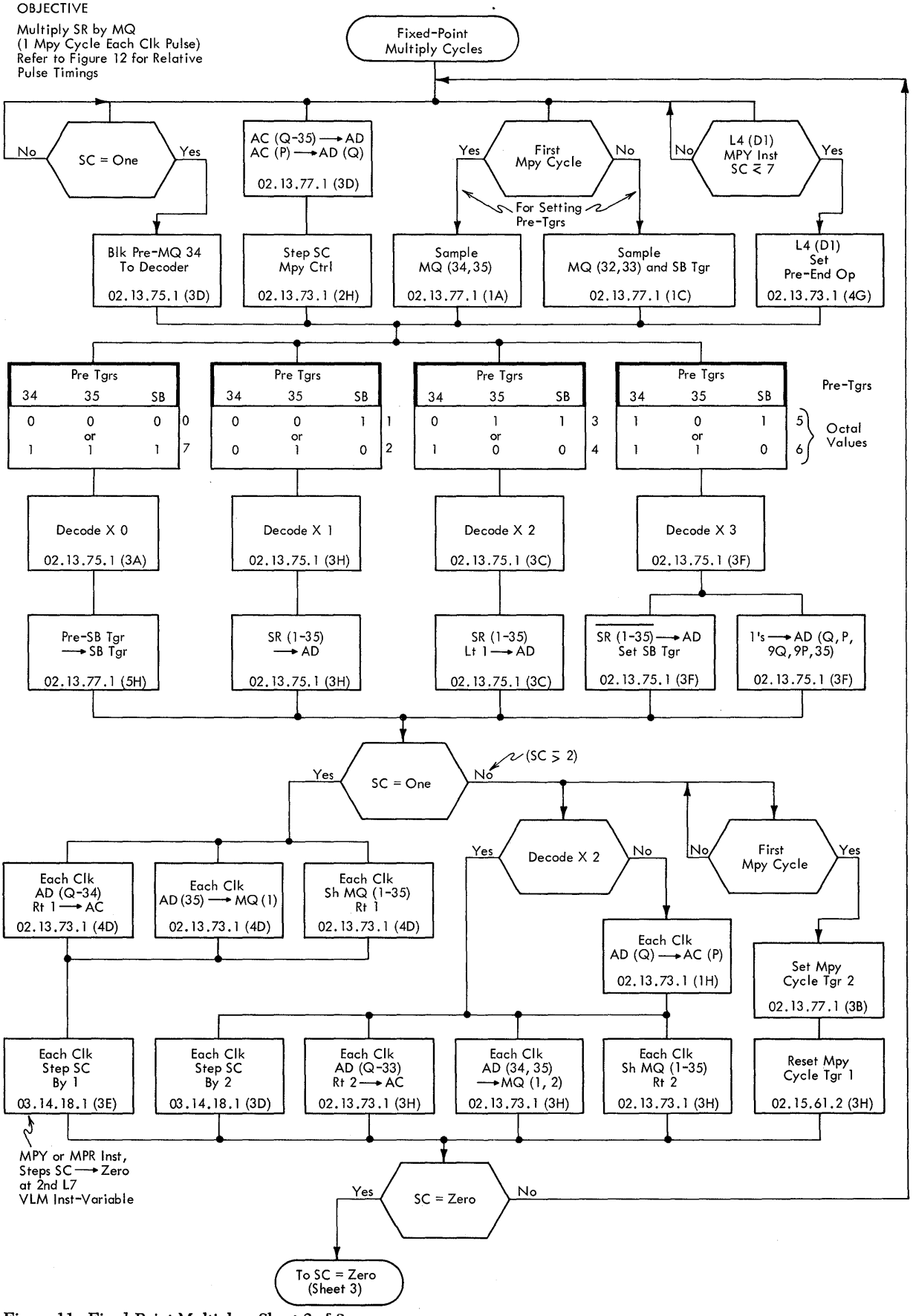


Figure 11. Fixed-Point Multiply—Sheet 2 of 3

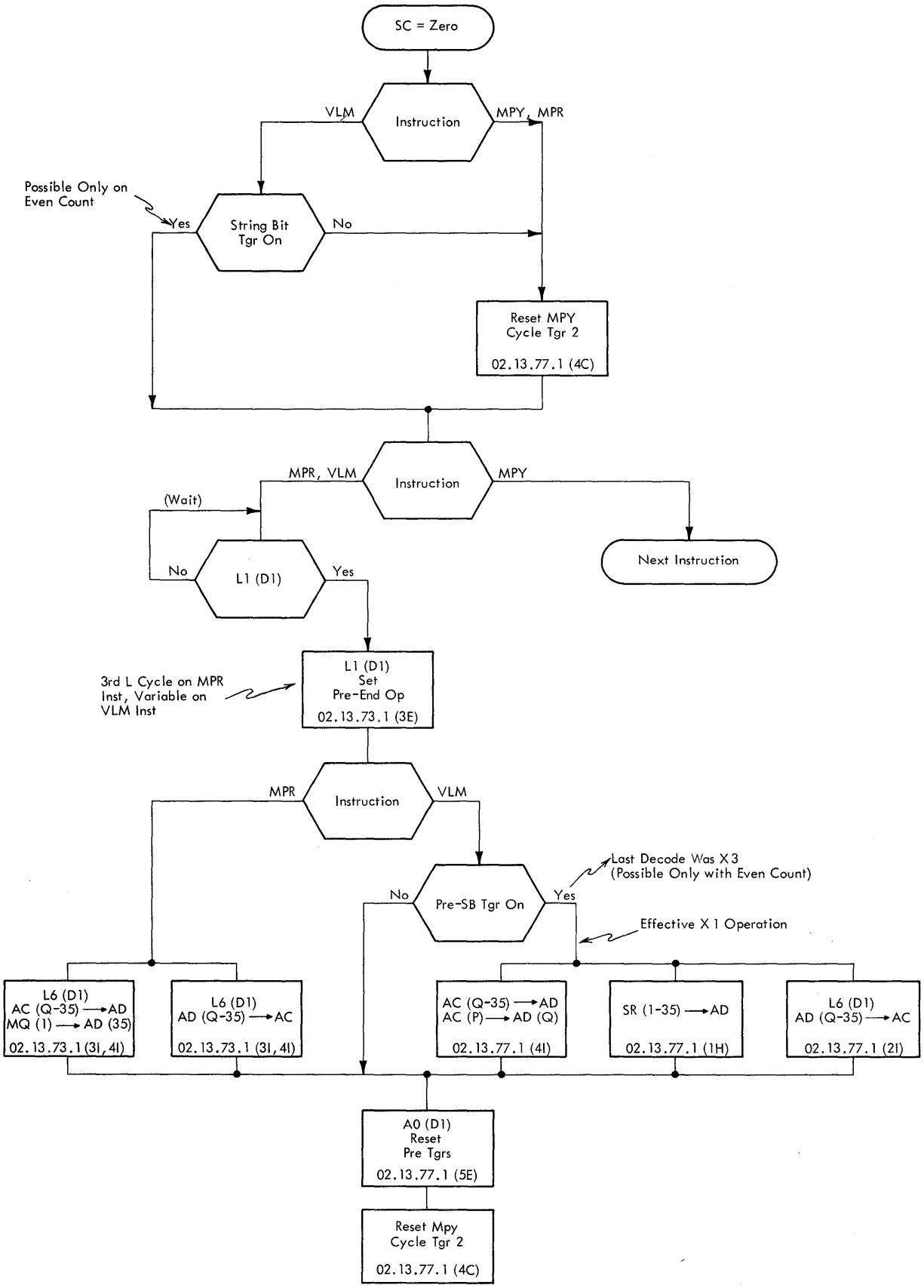


Figure 11. Fixed-Point Multiply—Sheet 3 of 3

PROGRAM (All Numbers Octal)

Location	Instruction	
0	LDQ	10
1	VLM	15 11
2	TRA	0
3	HTR	
10	15771	
11	5	

Line #	TEST POINT	LEVEL	LINE NAME	SYSTEMS
1.	1B2D23C	+F	Mpy Cycle Trigger	02.13.77.1(3D)
2.	1B3F04M	-F	Gate to MQ 35	02.04.06.1(2F)
3.	1B2E28D	+F	Sample MQ 34 and 35	02.13.77.1(1A)
4.	1B2E28C	+F	Sample MQ 32 and 33	02.13.77.1(1C)
5.	1B3F04P	-F	Set MQ 35	02.04.06.1(2F)
6.	1B3F04N	-F	Hold MQ 35	02.04.06.1(2F)
7.	1B3F07L	+F	MQ 32	02.04.05.1(2H)
8.	1B3F06L	+F	MQ 33	02.04.06.1(2A)
9.	1B3F05L	+F	MQ 34	02.04.06.1(2C)
10.	1B3F04L	+F	MQ 35	02.04.06.1(2F)
11.	1B2F04B	+F	Pre MQ 34 Trigger	02.13.77.1(2E)
12.	1B2F04C	+F	Pre MQ 35 Trigger	02.13.77.1(2F)
13.	1B2F04D	+F	Pre String Bit Tgr	02.13.77.1(2G)
14.	1B2F14H	+F	String Bit Trigger	02.13.77.1(3H)
15.	1B2J06B	+F	Decode X0	02.13.75.1(2A)
16.	1B2J08C	-F	Decode X1	02.13.75.1(3H)
17.	1B2J07C	-F	Decode X2	02.13.75.1(3C)
18.	1B2J09C	-F	Decode X3	02.13.75.1(3F)
19.	1A4D24D	-F	Step SC by 2	03.14.18.1(3D)
20.	1B2E26D	+F	SH RT 2	02.13.73.1(3H)
21.	1A4G25B	-F	Step SC by 1	03.14.18.1(3E)
22.	1B2E26C	+F	SH RT 1	02.13.73.1(4D)

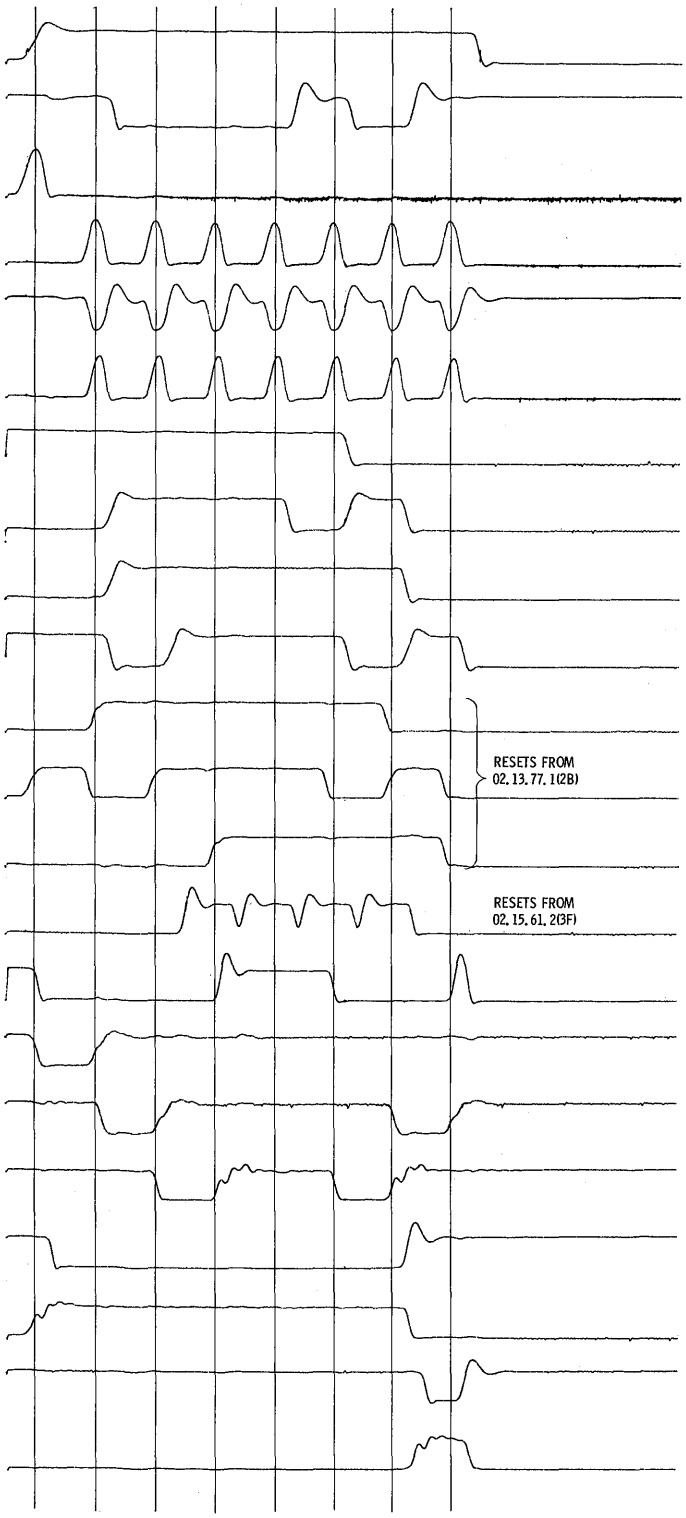


Figure 12. Fixed-Point Multiply Cycles; X-Y Recording

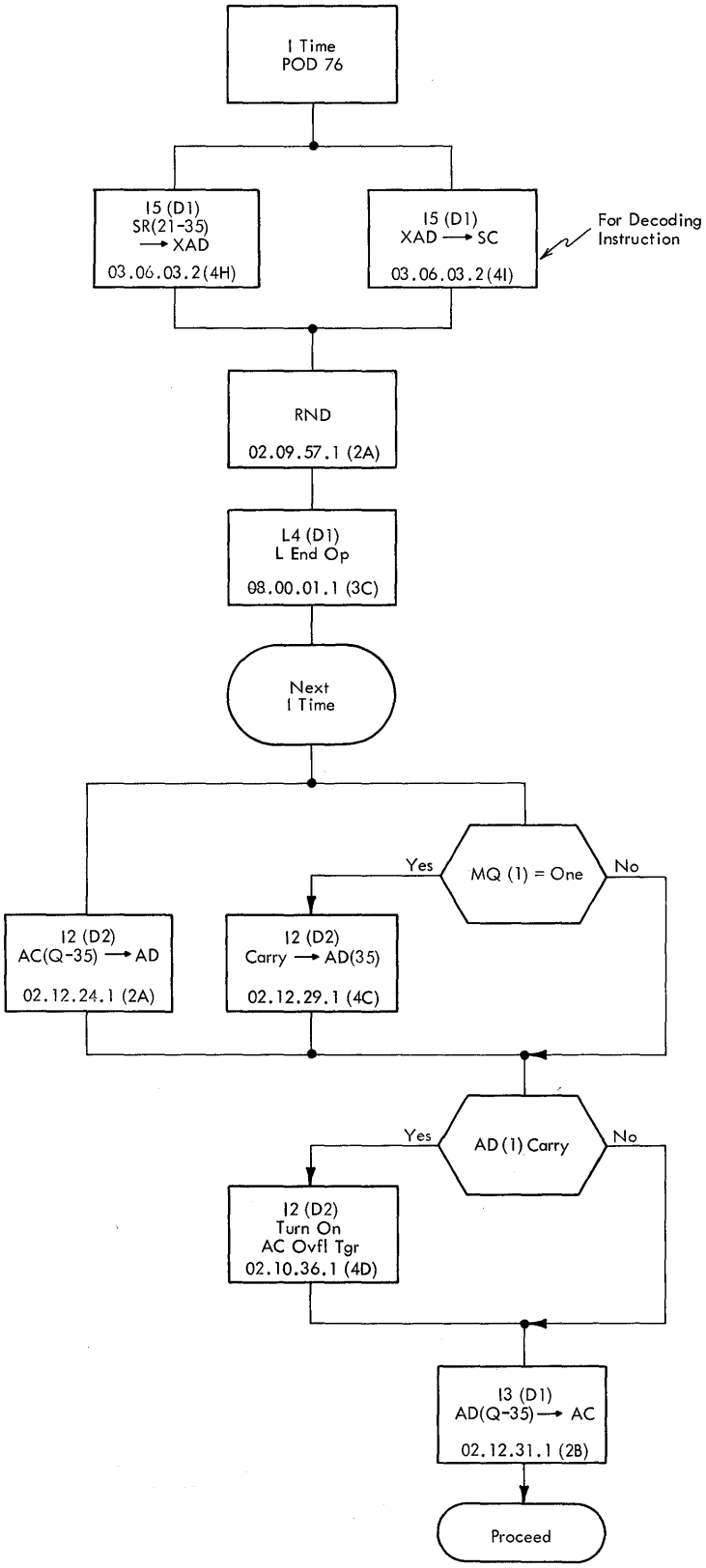


Figure 13. Round

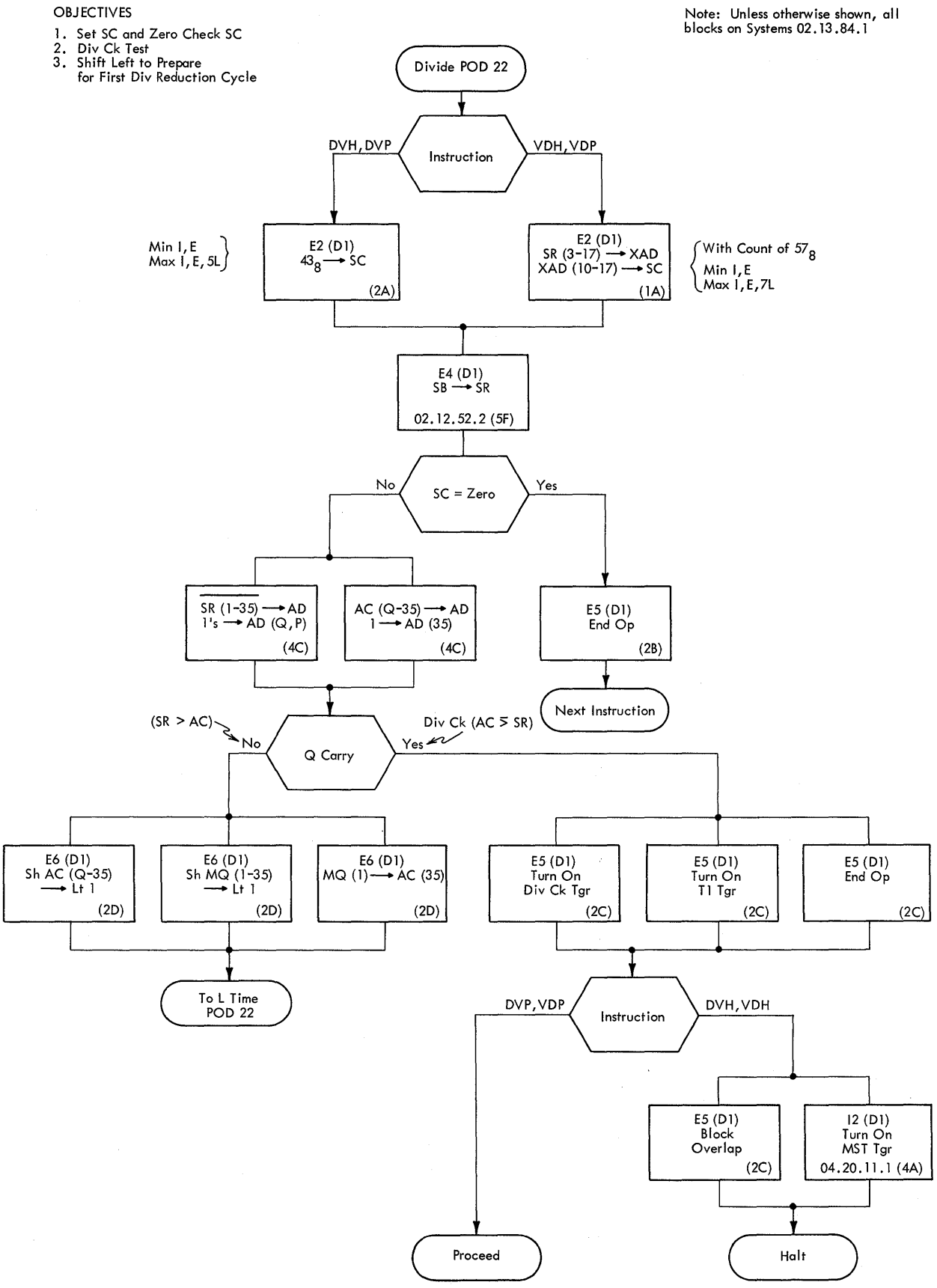


Figure 14. Fixed-Point Division—Sheet 1 of 2

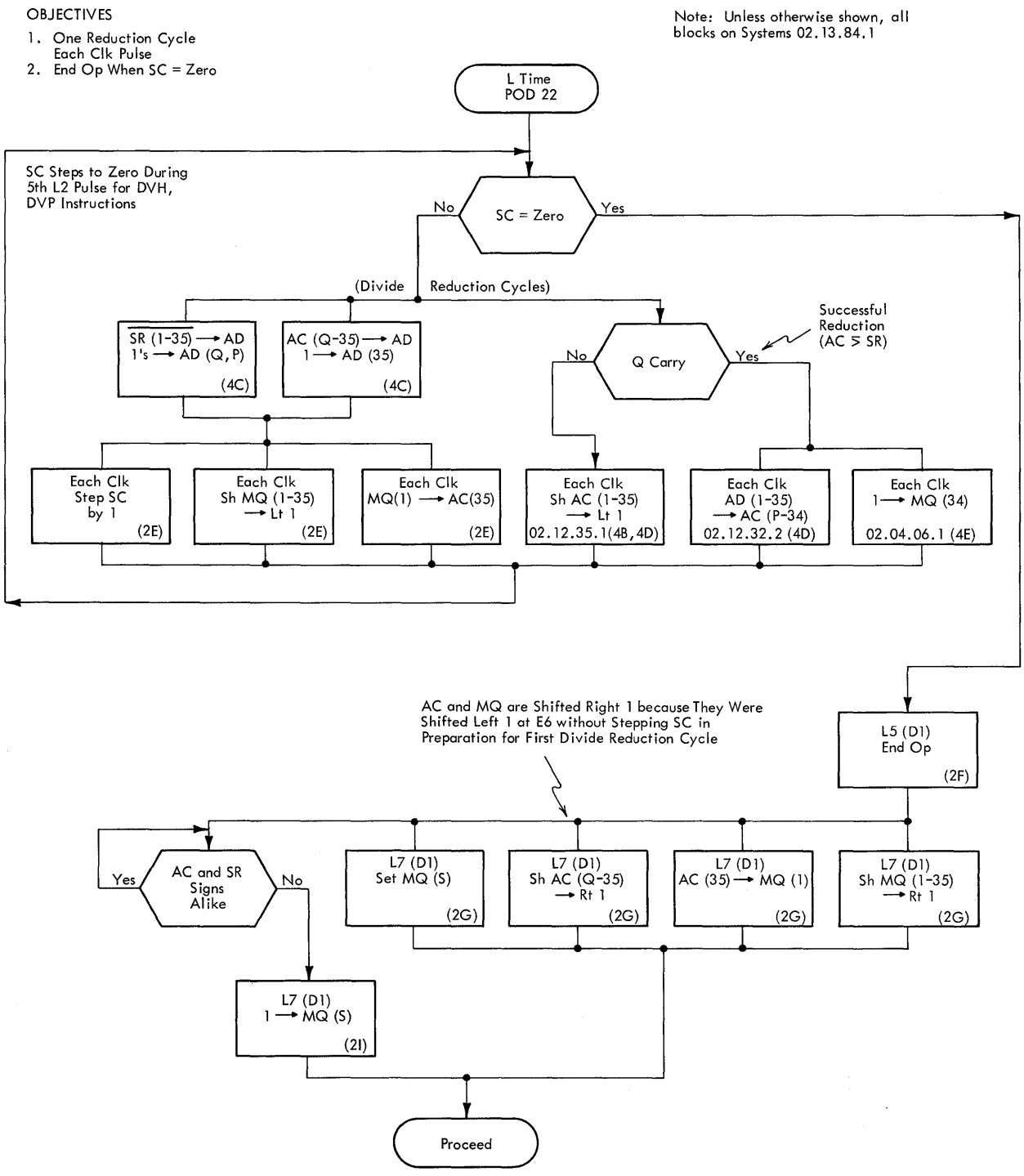


Figure 14. Fixed-Point Division—Sheet 2 of 2

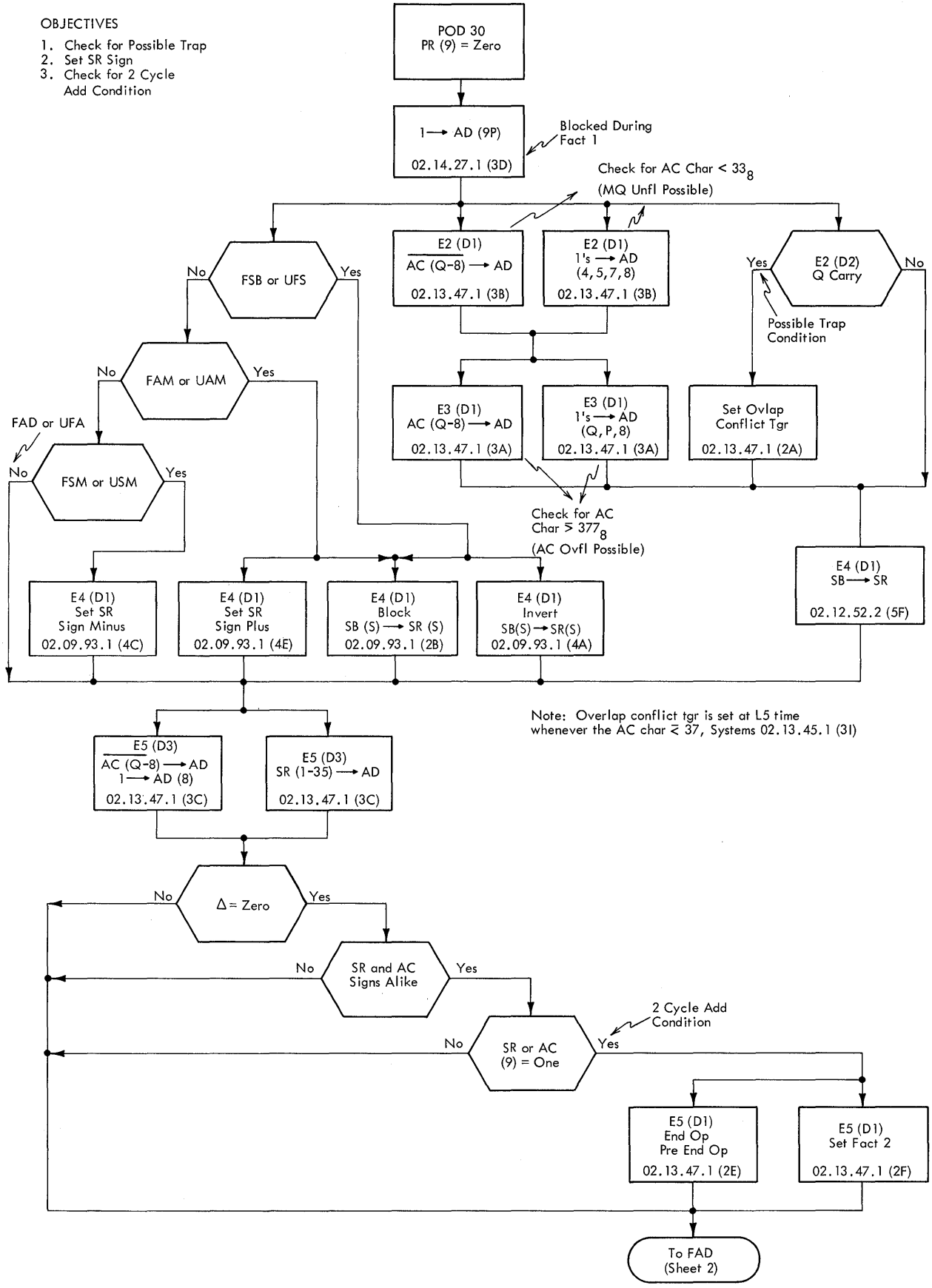


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 1 of 8

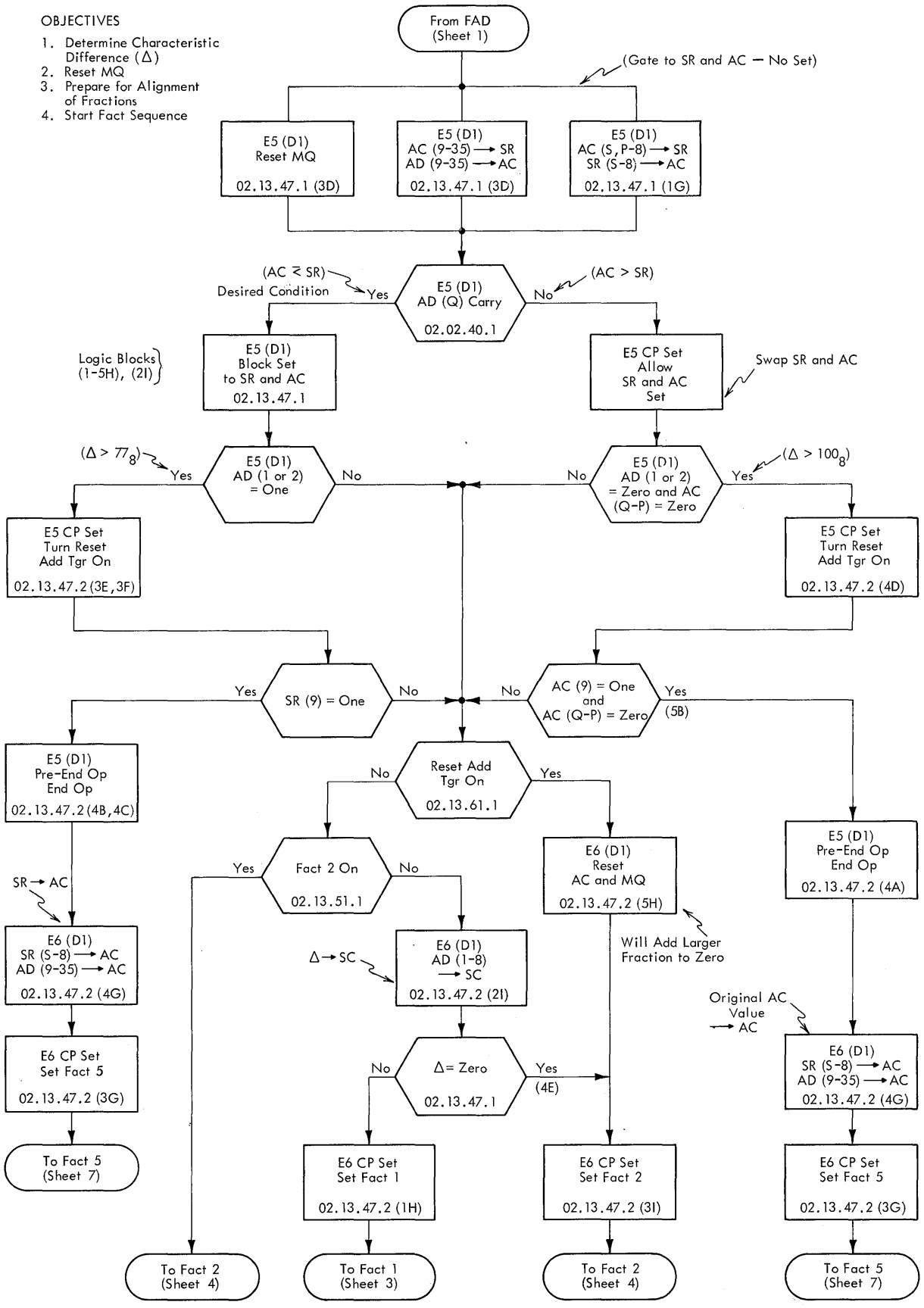


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 2 of 8

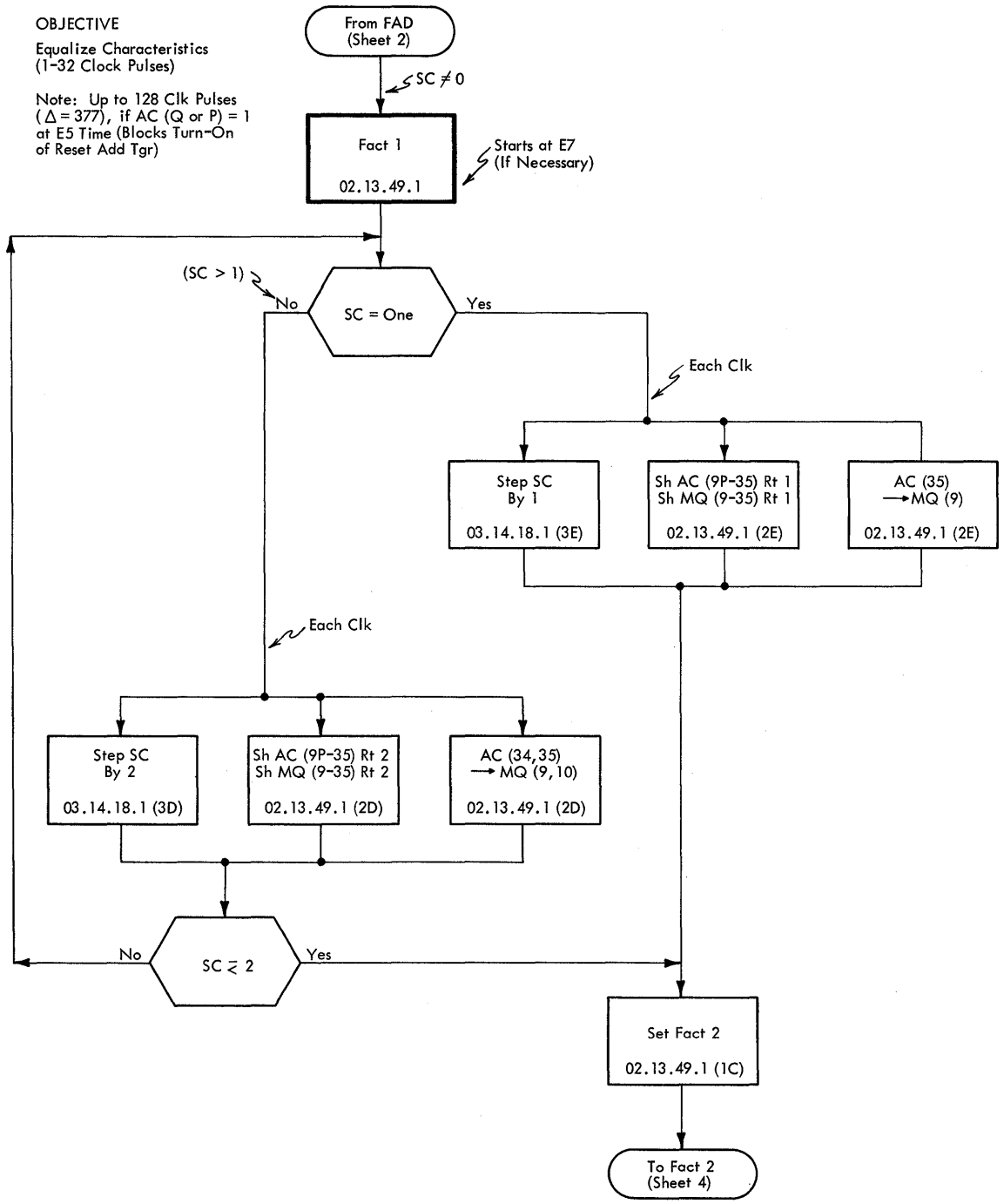


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 3 of 8

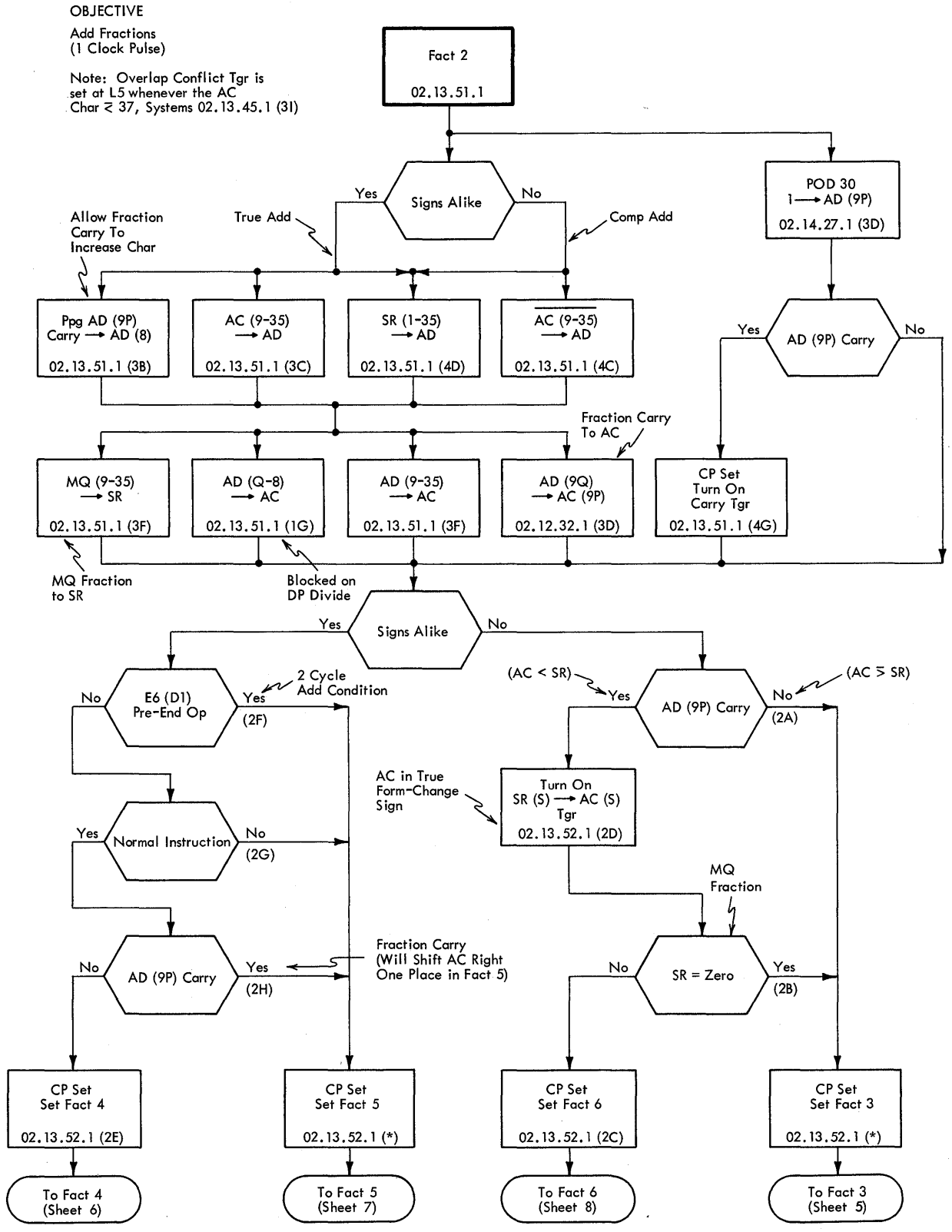


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 4 of 8

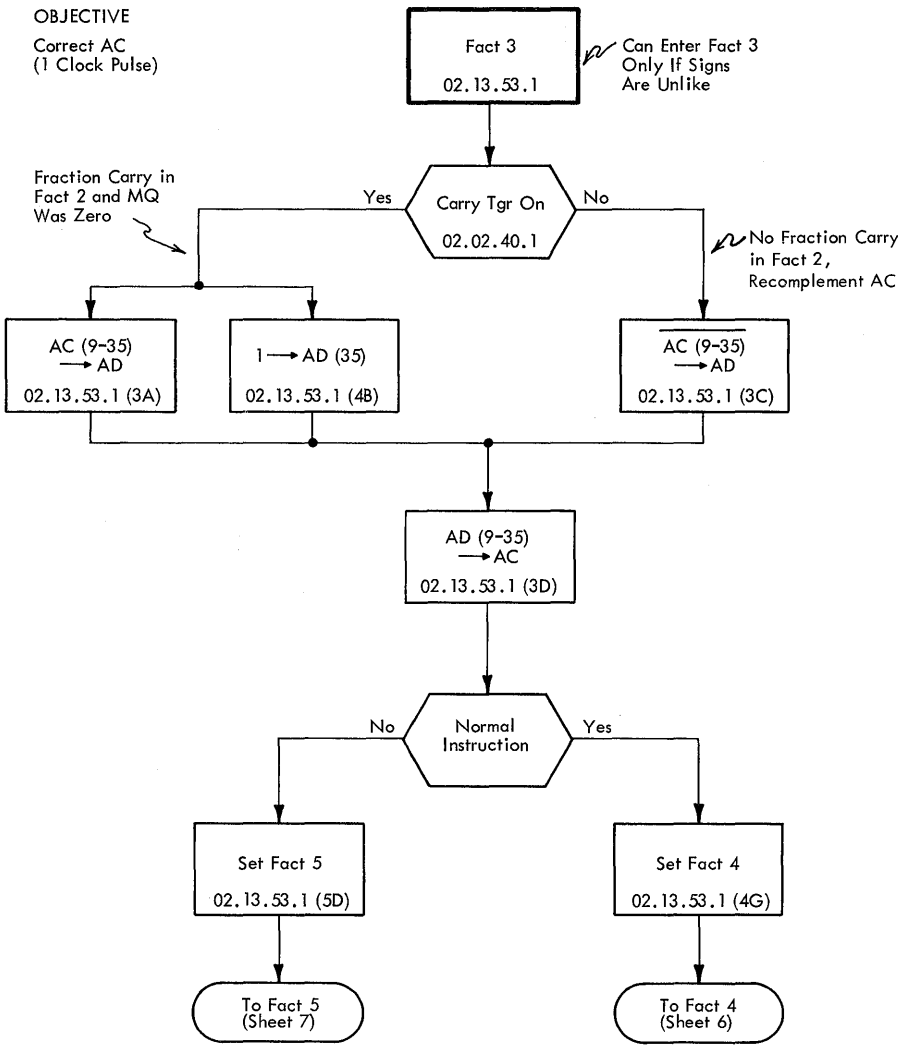


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 5 of 8

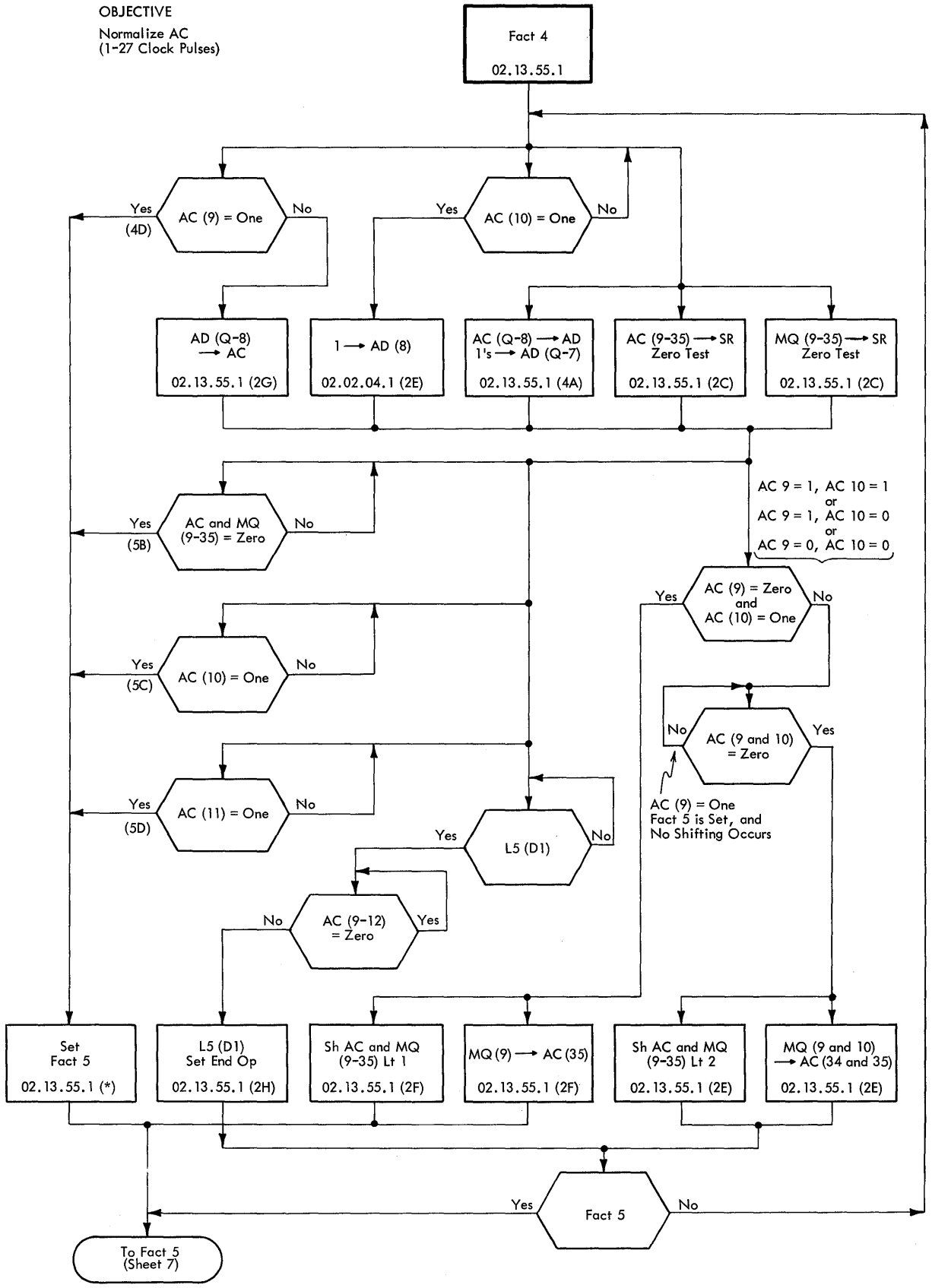


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 6 of 8

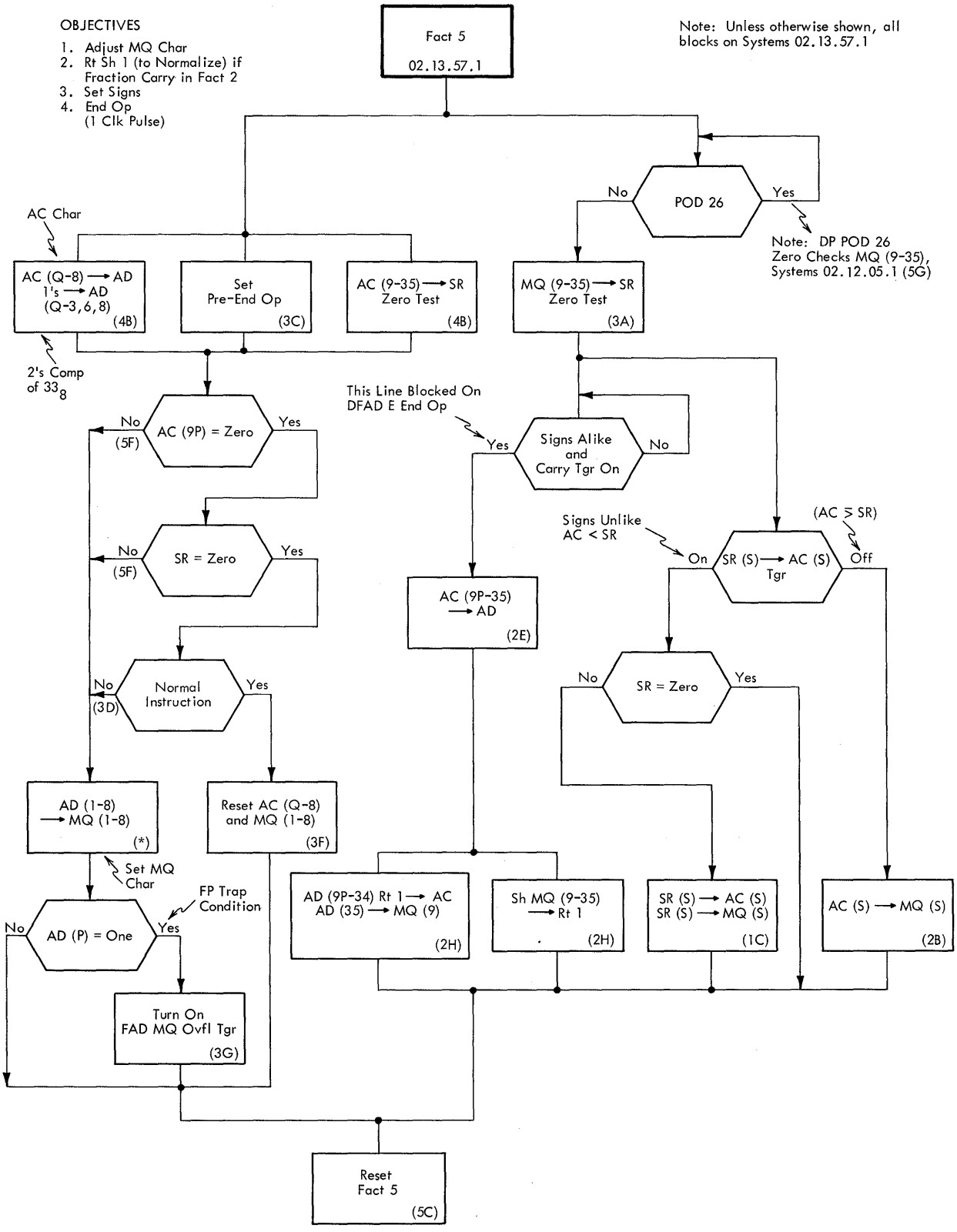


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 7 of 8

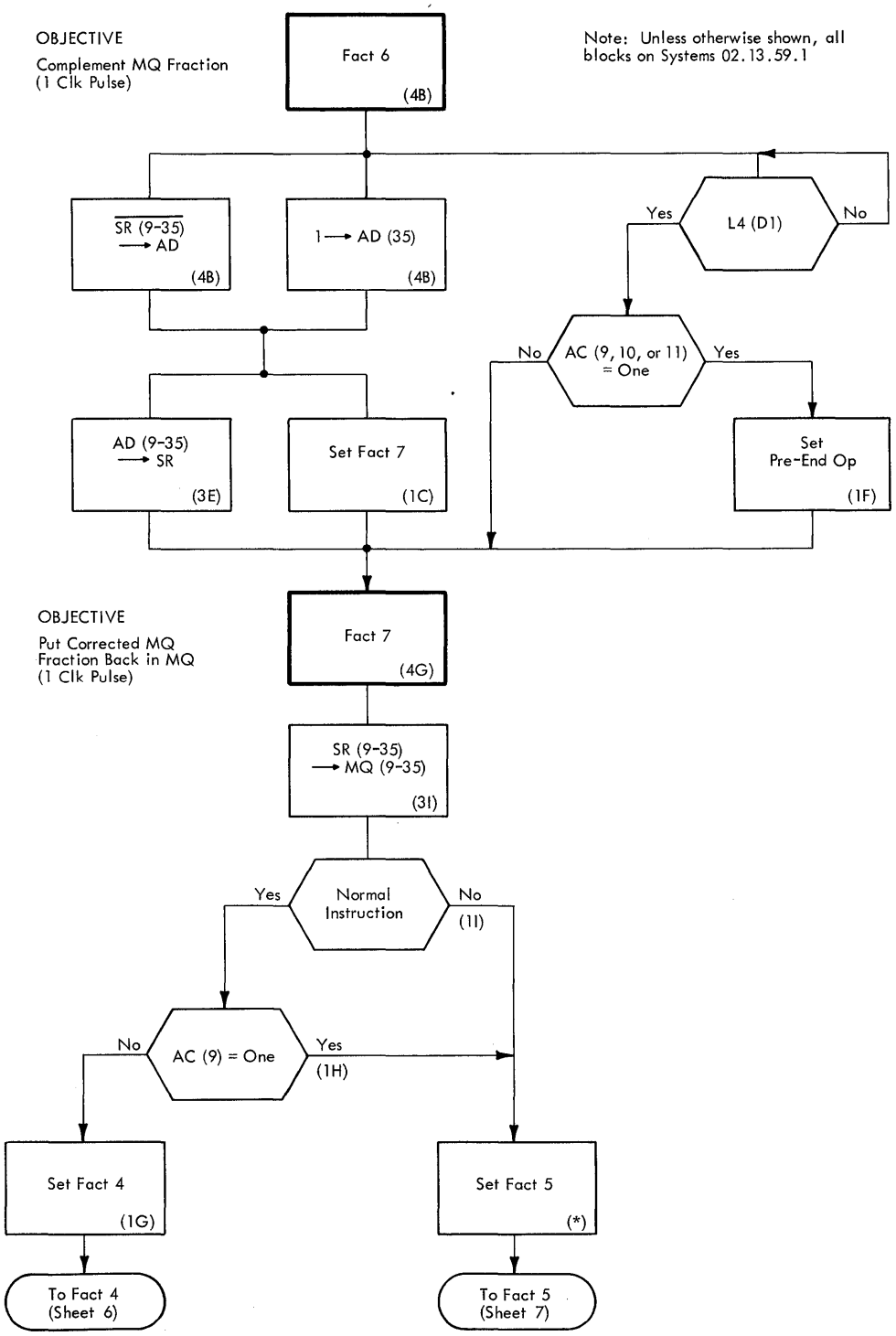


Figure 15. Single-Precision FP Addition and Subtraction—FAD; Sheet 8 of 8

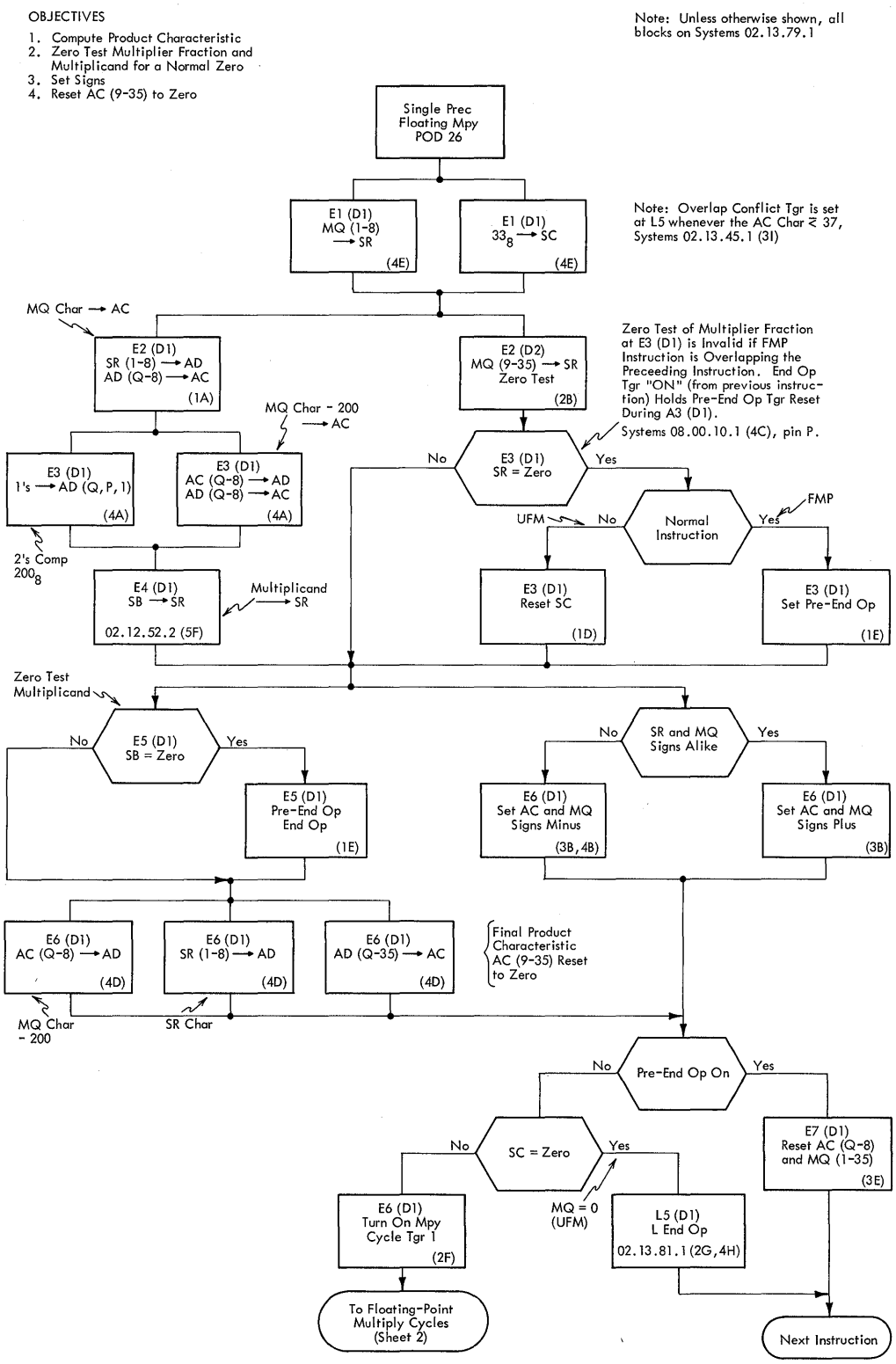


Figure 16. Single-Precision FP Multiply—FMP; Sheet 1 of 2

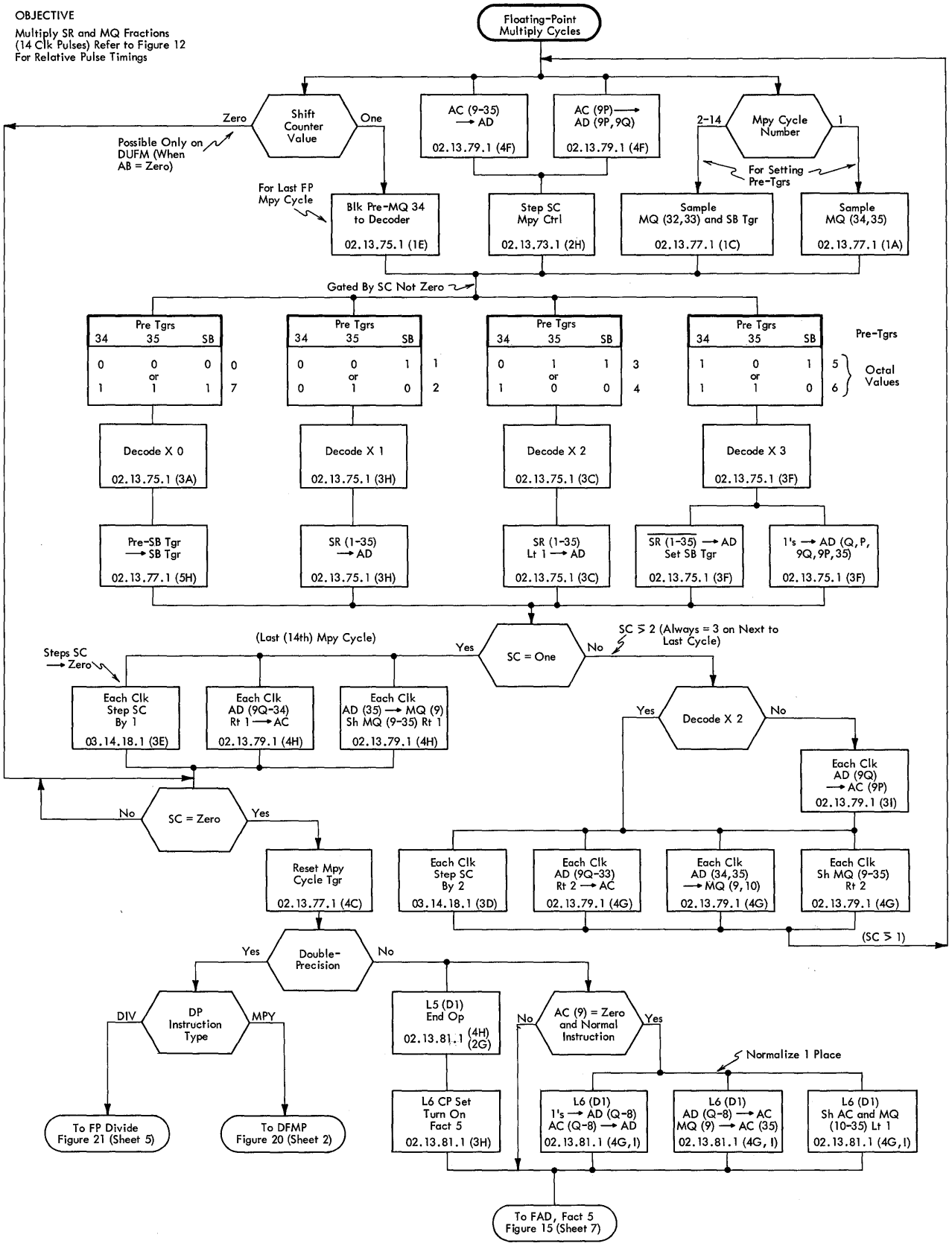


Figure 16. Single-Precision FP Multiply—FMP; Sheet 2 of 2

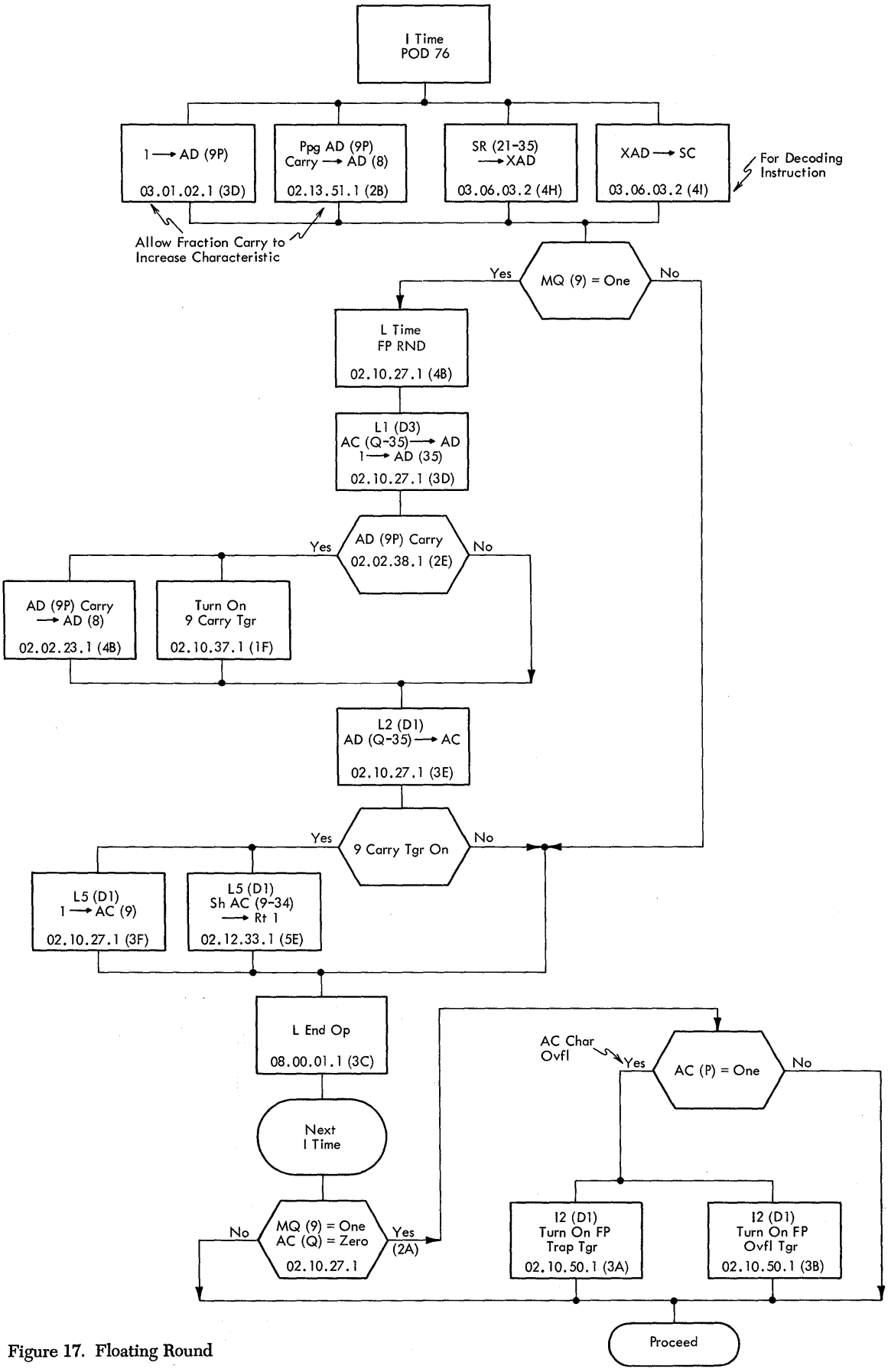


Figure 17. Floating Round

	E5 S, 1-8, 9-35			E6 S, 1-8, 9-35			E7 S, 1-8, 9-35			L0 S, 1-8, 9-35			L1 S, 1-8, 9-35		
	SR (9-35) → MQ MQ (9-35) → SR			SR (9-35) → AC AC (9-35) → SR			SR (9-35) → MQ MQ (9-35) → SR			IBR (9-35) → SR SR → SI					
AC	A	A	A	A	A	A	A	A	B	A	A	B	A	A	B.
MQ	B	B	B	B	B	C	B	B	C	B	B	A	B	B	A
SR	C	C	C	C	C	B	C	C	A	C	C	C	C	C	D
SI													C	C	C
IBR	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

Chart 1 Conditions: Carry Tgr On (AC ≷ SR) Normal Add (Δ ≷ 77) or Carry Tgr On (AC ≷ SR) Reset Add (Δ > 77) SR (9) = Zero

	E5			E6			E7			L0			L1		
	SR (9-35) → MQ MQ (9-35) → SR SR (S-8) → AC AC (S-8) → SR			SR (9-35) → AC AC (9-35) → SR			IBR (9-35) → SR SR → SI			SR (9-35) → AC AC (9-35) → SR					
AC	A	A	A	C	C	A	C	C	B	C	C	B	C	C	D
MQ	B	B	B	B	B	C	B	B	C	B	B	C	B	B	C
SR	C	C	C	A	A	B	A	A	A	A	A	D	A	A	B
SI										A	A	A	A	A	A
IBR	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

Chart 2 Conditions: Carry Tgr Off (AC > SR) Normal Add (Δ ≷ 77) or Carry Tgr Off (AC > SR) Reset Add (Δ > 100) AC (9) = Zero

	E5			E6			E7			I0			I1		
	SR (9-35) → MQ MQ (9-35) → SR			IBR (9-35) → SR SR → SI			SR (9-35) → MQ MQ (9-35) → SR SR (S-8) → AC			SR (9-35) → AC AC (9-35) → SR			Fact 5		
AC	A	A	A	A	A	A	A	A	A	C	C	A	C	C	C
MQ	B	B	B	B	B	C	B	B	C	B	B	D	B	B	D
SR	C	C	C	C	C	B	C	C	D	C	C	C	C	C	A
SI							C	C	B	C	C	B	C	C	B
IBR	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

Chart 3 Conditions: Carry Tgr On (AC ≷ SR) Reset Add (Δ> 77) SR (9) = One (End Op Tgr On)

	E5			E6			E7			I0			I1		
	SR (9-35) → MQ MQ (9-35) → SR SR (S-8) → AC AC (S-8) → SR						SR (9-35) → MQ MQ (9-35) → SR SR (S-8) → AC						Fact 5		
AC	A	A	A	C	C	A	C	C	A	A	A	A	A	A	A
MQ	B	B	B	B	B	C	B	B	C	B	B	B	B	B	B
SR	C	C	C	A	A	B	A	A	B	A	A	C	A	A	C
SI															
IBR	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

Chart 4 Conditions: Carry Tgr Off (AC > SR) Reset Add (Δ > 100) AC (9) = One (End Op Tgr On)

Note: On charts 1-4, register contents are shown at beginning of indicated clock pulses.

	Carry Tgr On (AC ≷ SR)					Carry Tgr Off (AC > SR)				
	AC	MQ	SR	SI	IBR	AC	MQ	SR	SI	IBR
DPS = 1, Fact 1	B	A	D	C	D	D	C	B	A	D
DPS = 1, Fact 2	B	A	D	C	D	D	C	B	A	D
DPS = 1, Fact 0	B + D	A	A	C	D	B + D	C	C	A	D
DPS = 2	A	A	B + D	C	D	C	C	B + D	A	D
DPS = 3, Fact 2	A	B + D	C	C	D	C	B + D	A	A	D
DPS = 3, $\overline{\text{Fact 2}}$	A + C	B + D	B + D	C	D	A + C	B + D	B + D	A	D

Chart 5: DPS > Zero

Figure 18. DFAD; Register Exchange Charts

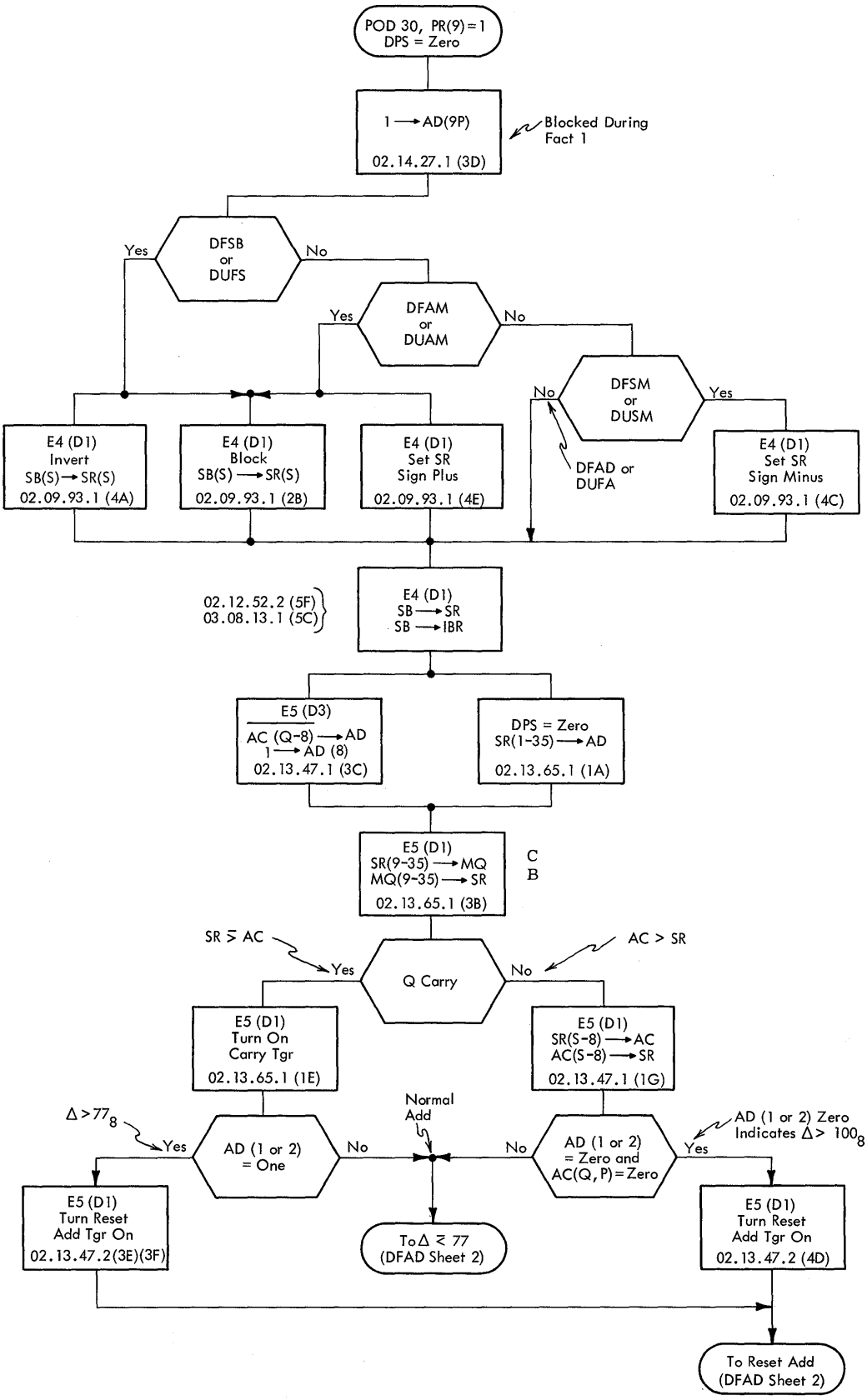


Figure 19. Double-Precision FP Addition and Subtraction—DFAD; Sheet 1 of 6

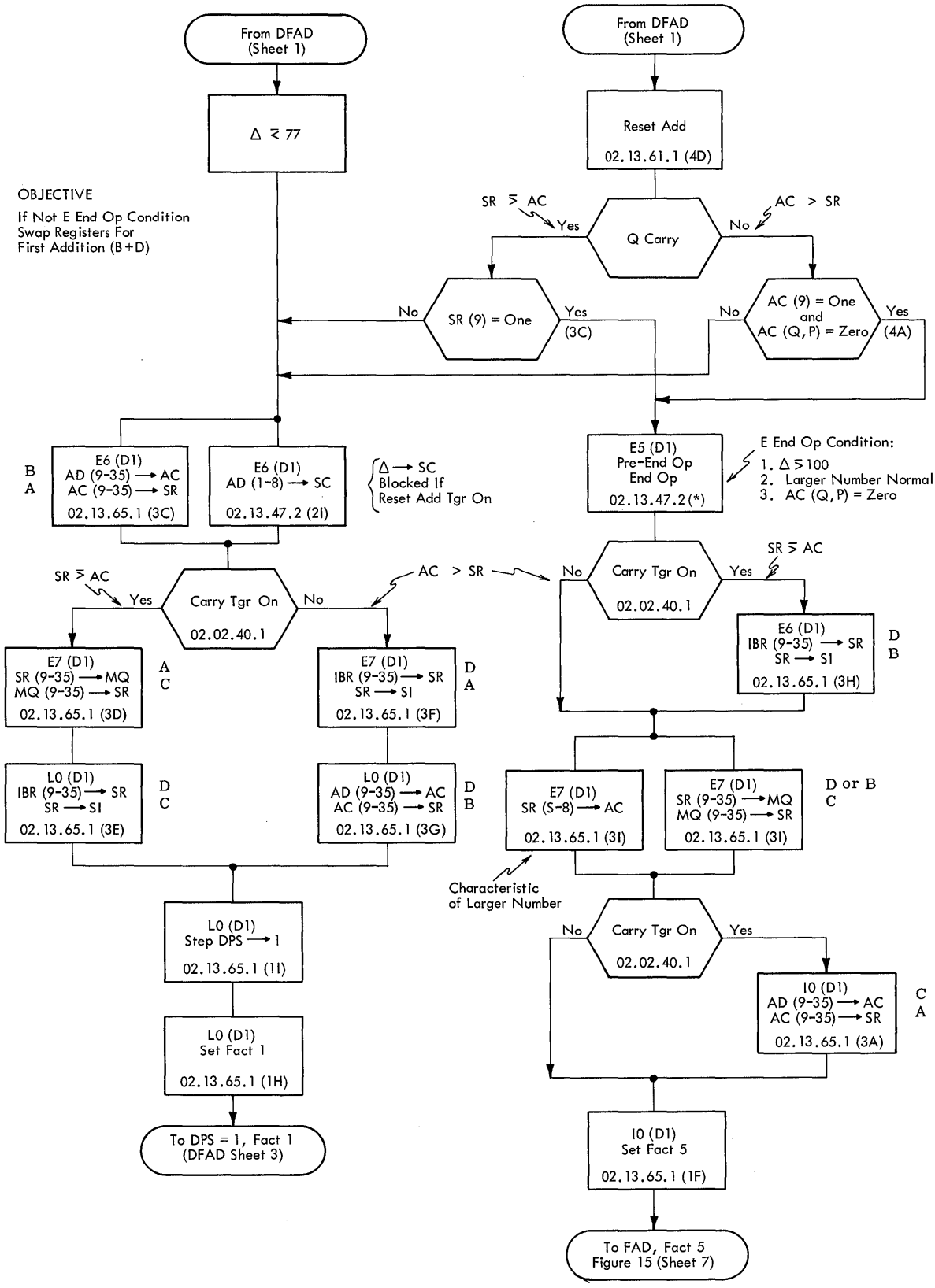


Figure 19. Double-Precision FP Addition and Subtraction—DFAD; Sheet 2 of 6

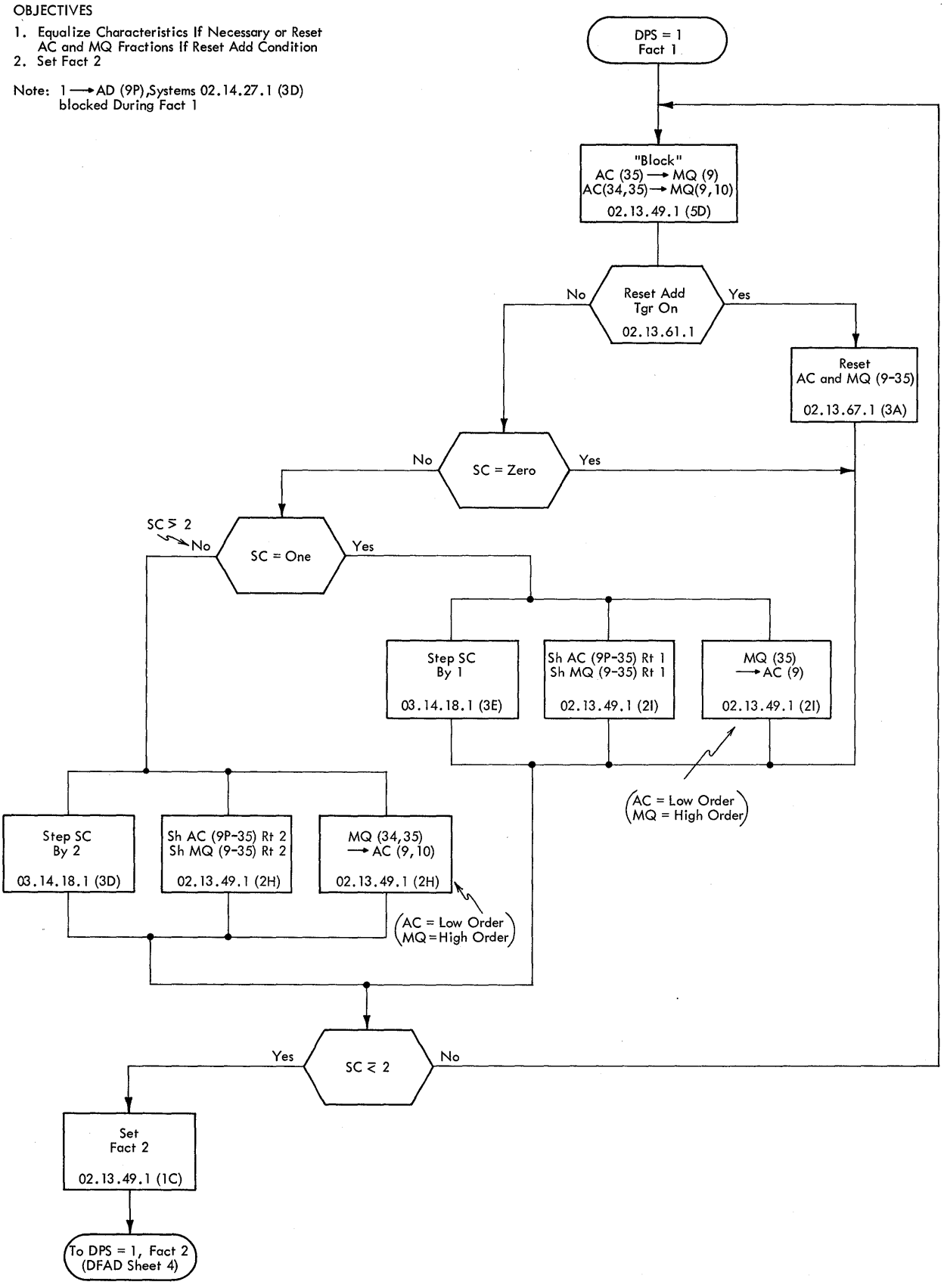


Figure 19. Double-Precision FP Addition and Subtraction—DFAD; Sheet 3 of 6

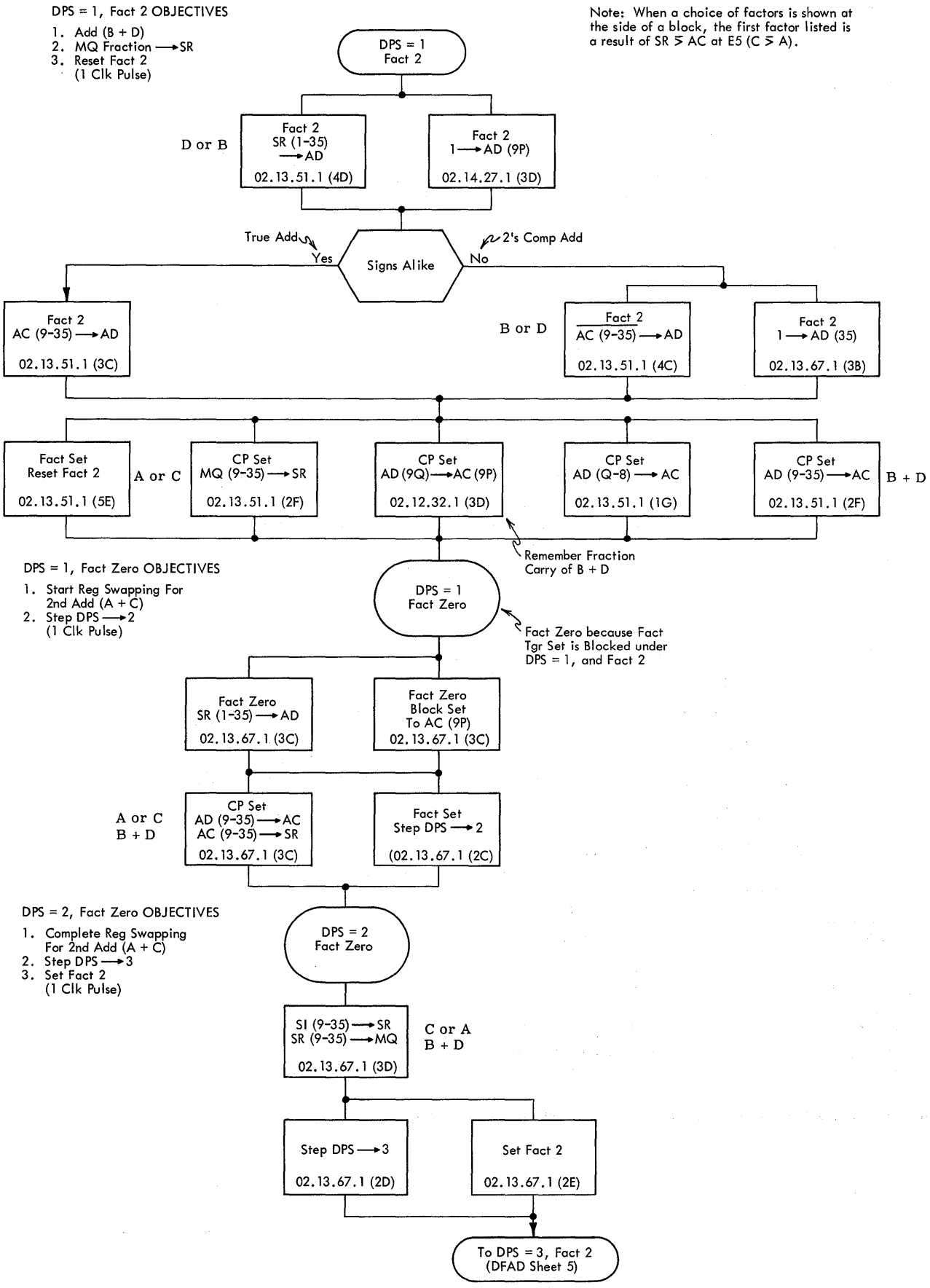


Figure 19. Double-Precision FP Addition and Subtraction—DFAD; Sheet 4 of 6

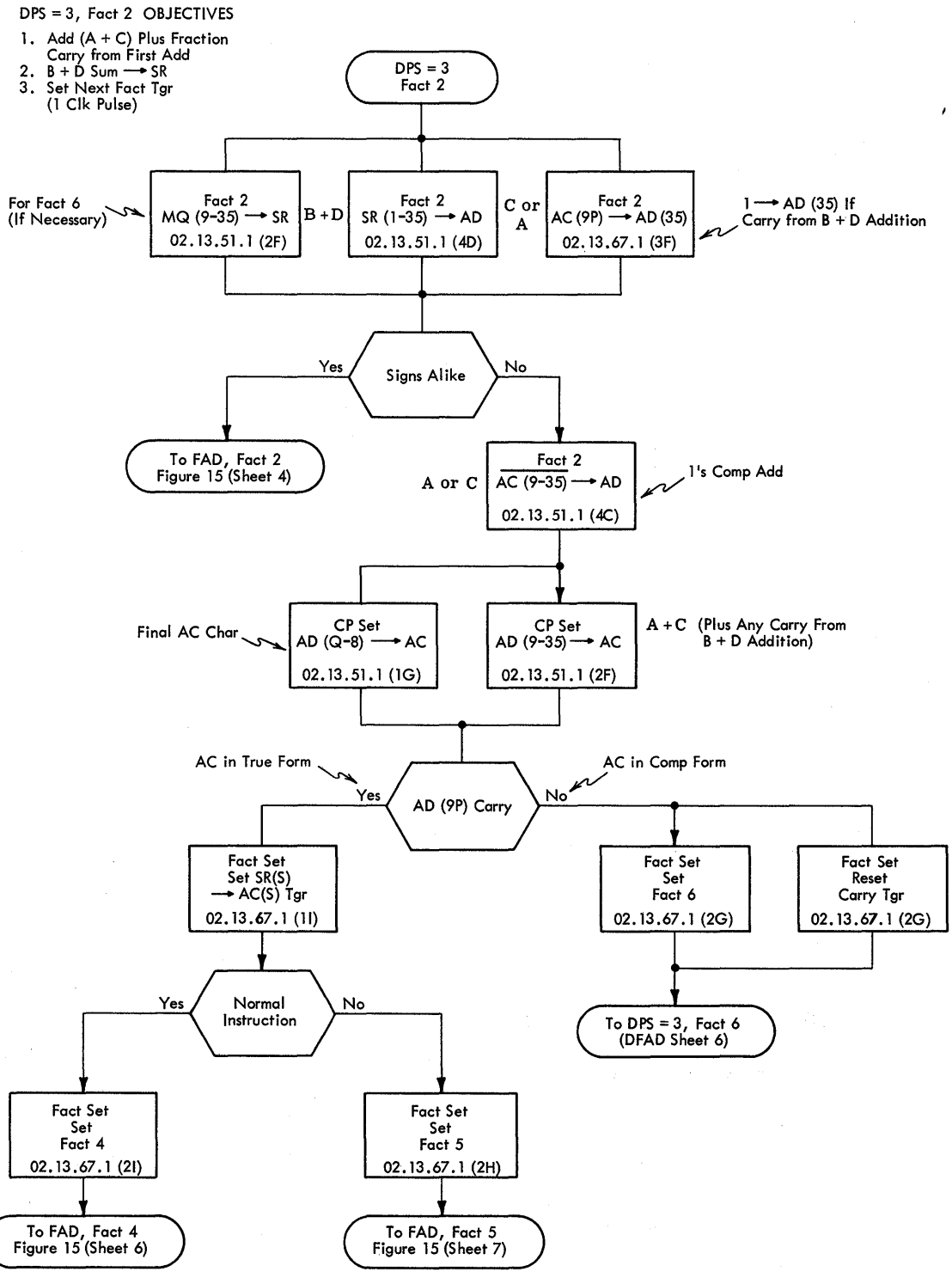


Figure 19. Double-Precision FP Addition and Subtraction—DFAD; Sheet 5 of 6

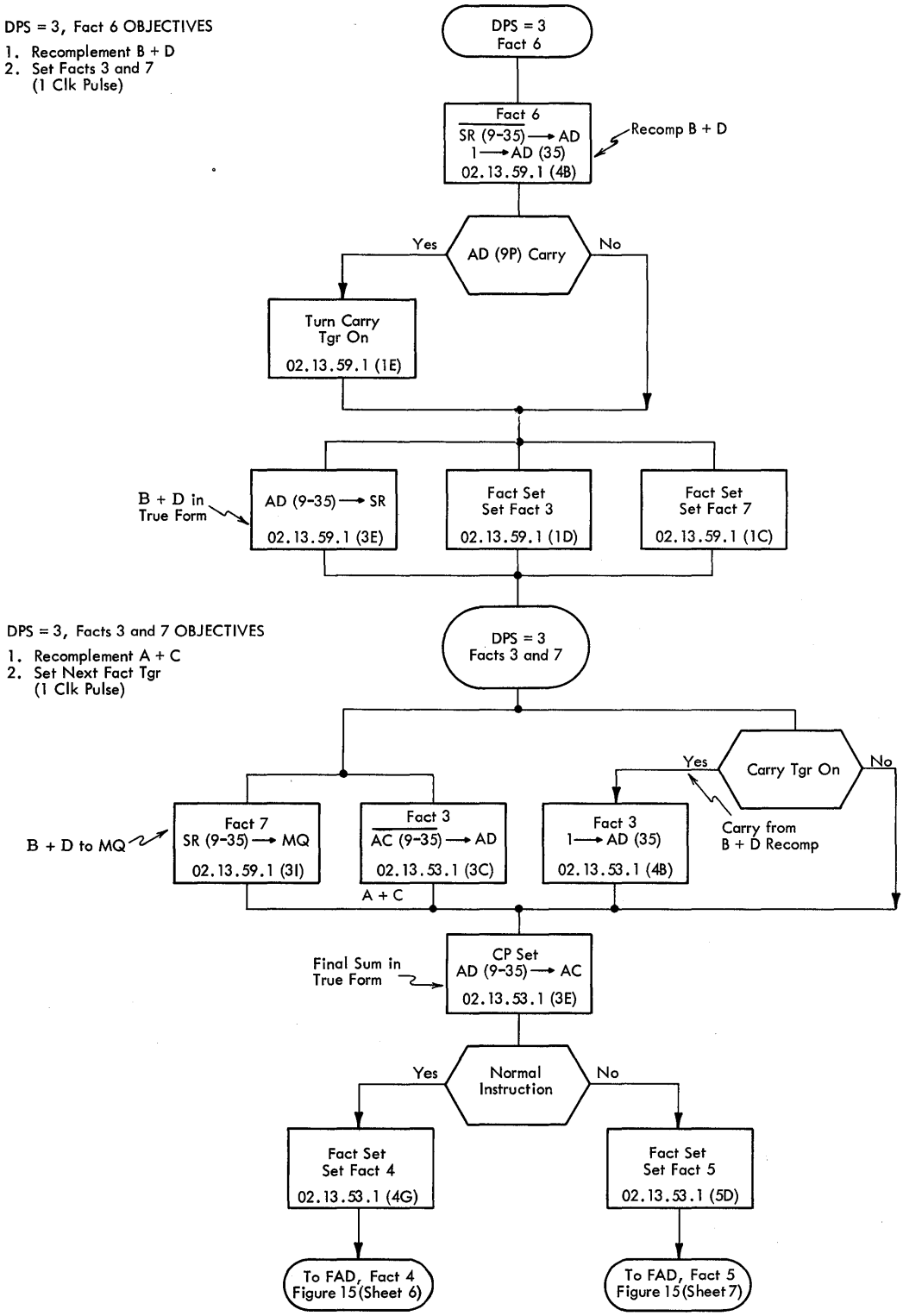


Figure 19. Double-Precision FP Addition and Subtraction—DFAD; Sheet 6 of 6

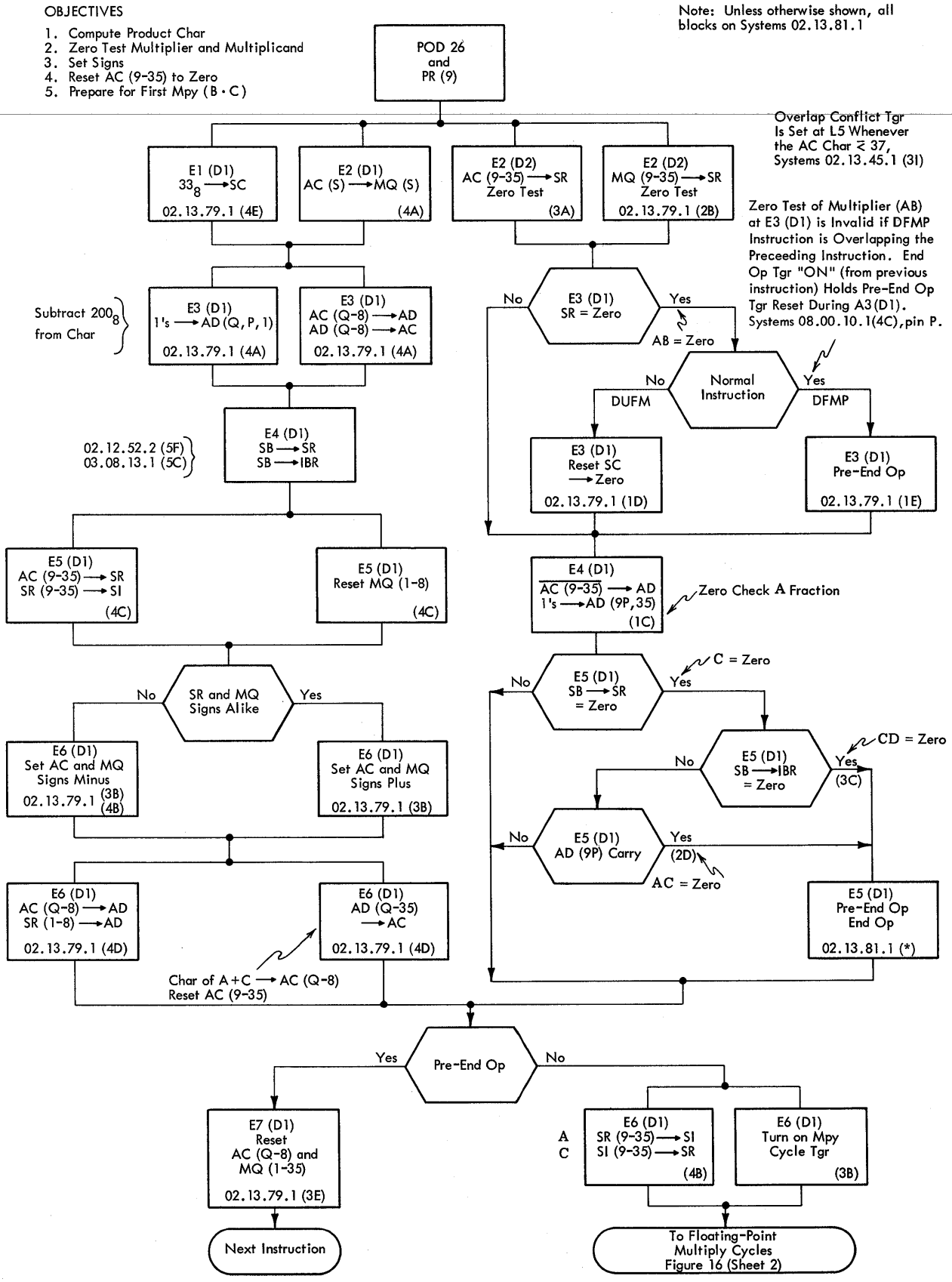


Figure 20. Double-Precision FP Multiply—DFMP; Sheet 1 of 2

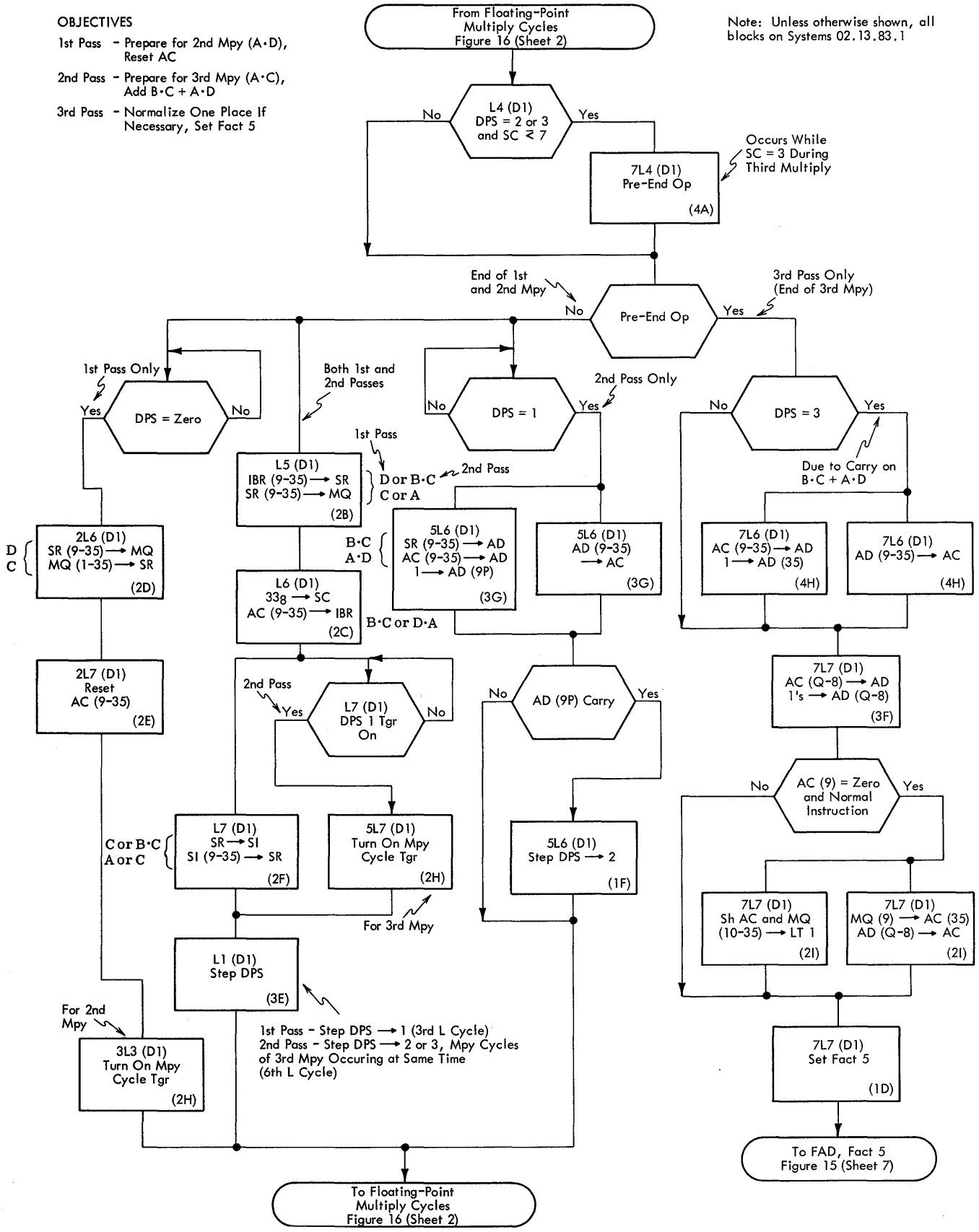


Figure 20. Double-Precision FP Multiply—DFMP; Sheet 2 of 2

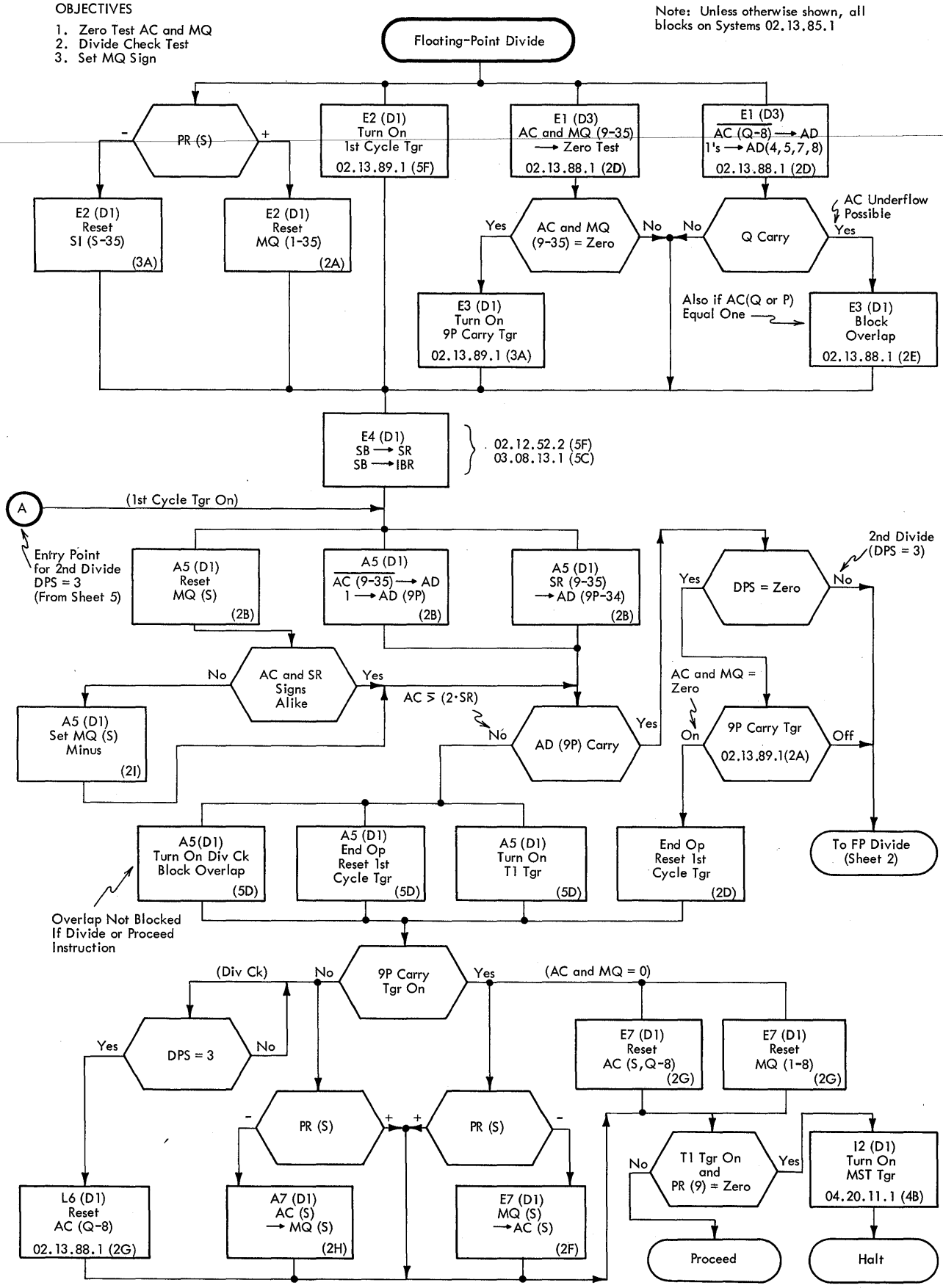


Figure 21. Floating-Point Divide (Single and Double-Precision)—Sheet 1 of 6

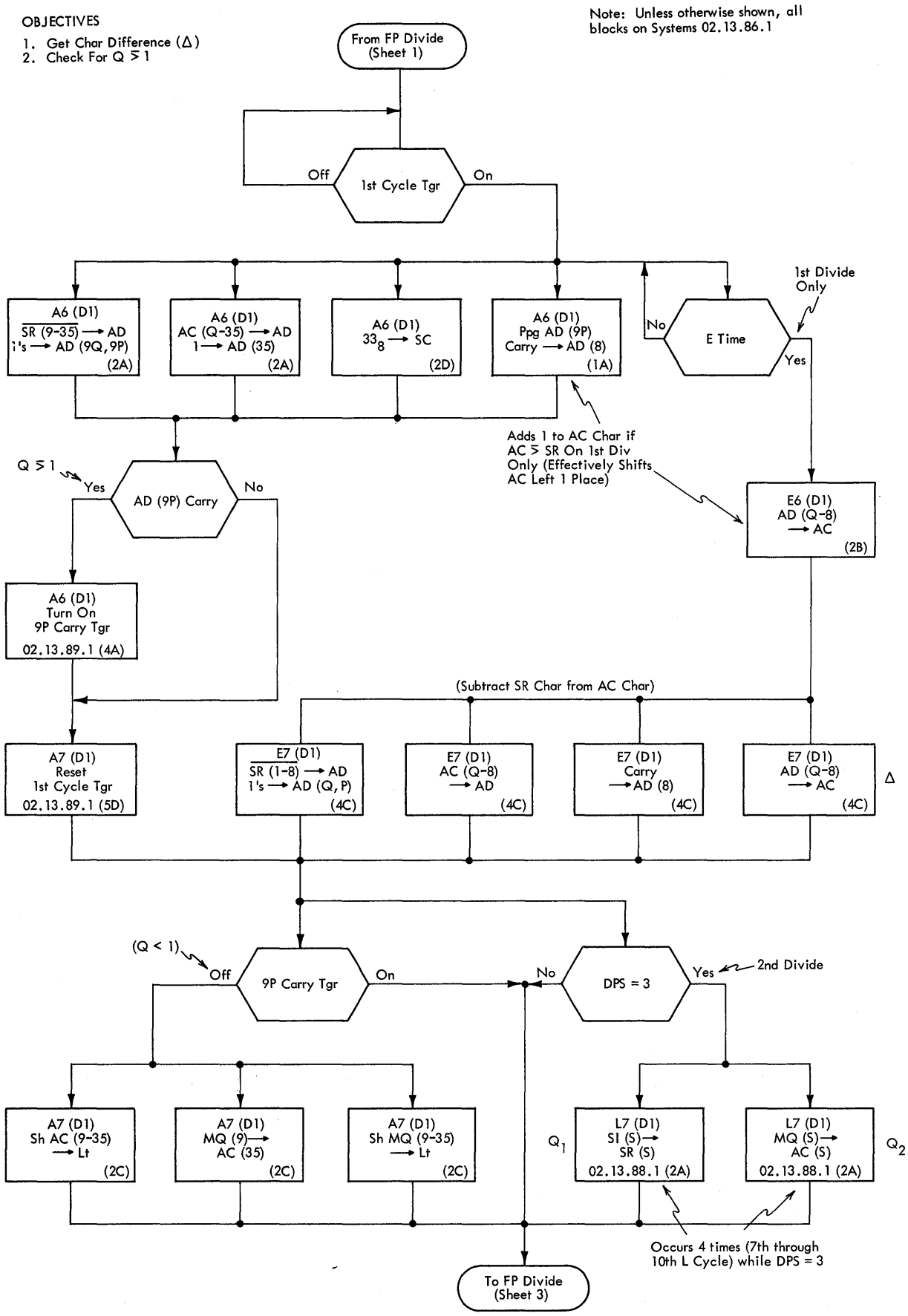


Figure 21. Floating-Point Divide (Single and Double-Precision)—Sheet 2 of 6

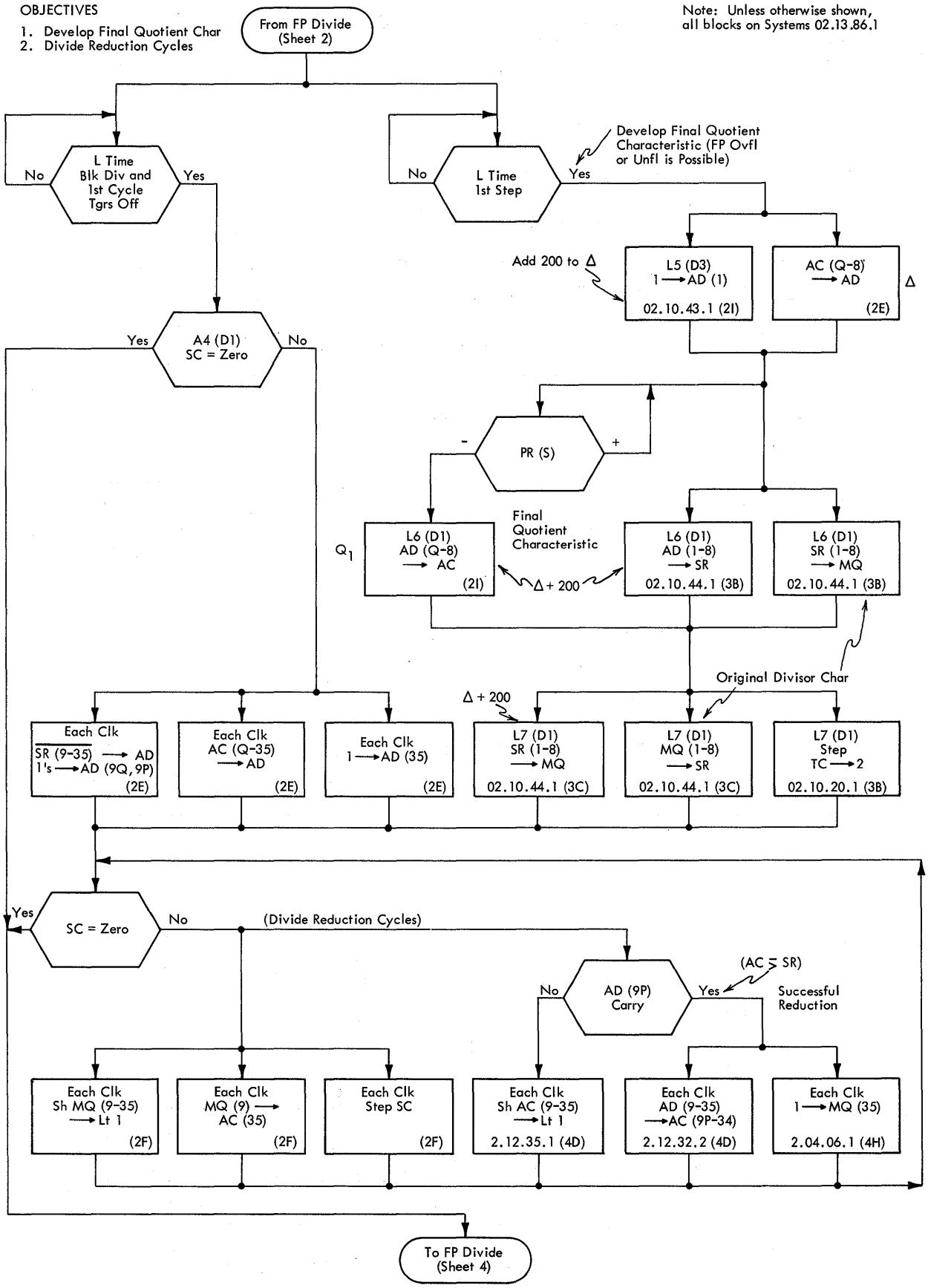


Figure 21. Floating-Point Divide (Single and Double-Precision)—Sheet 3 of 6

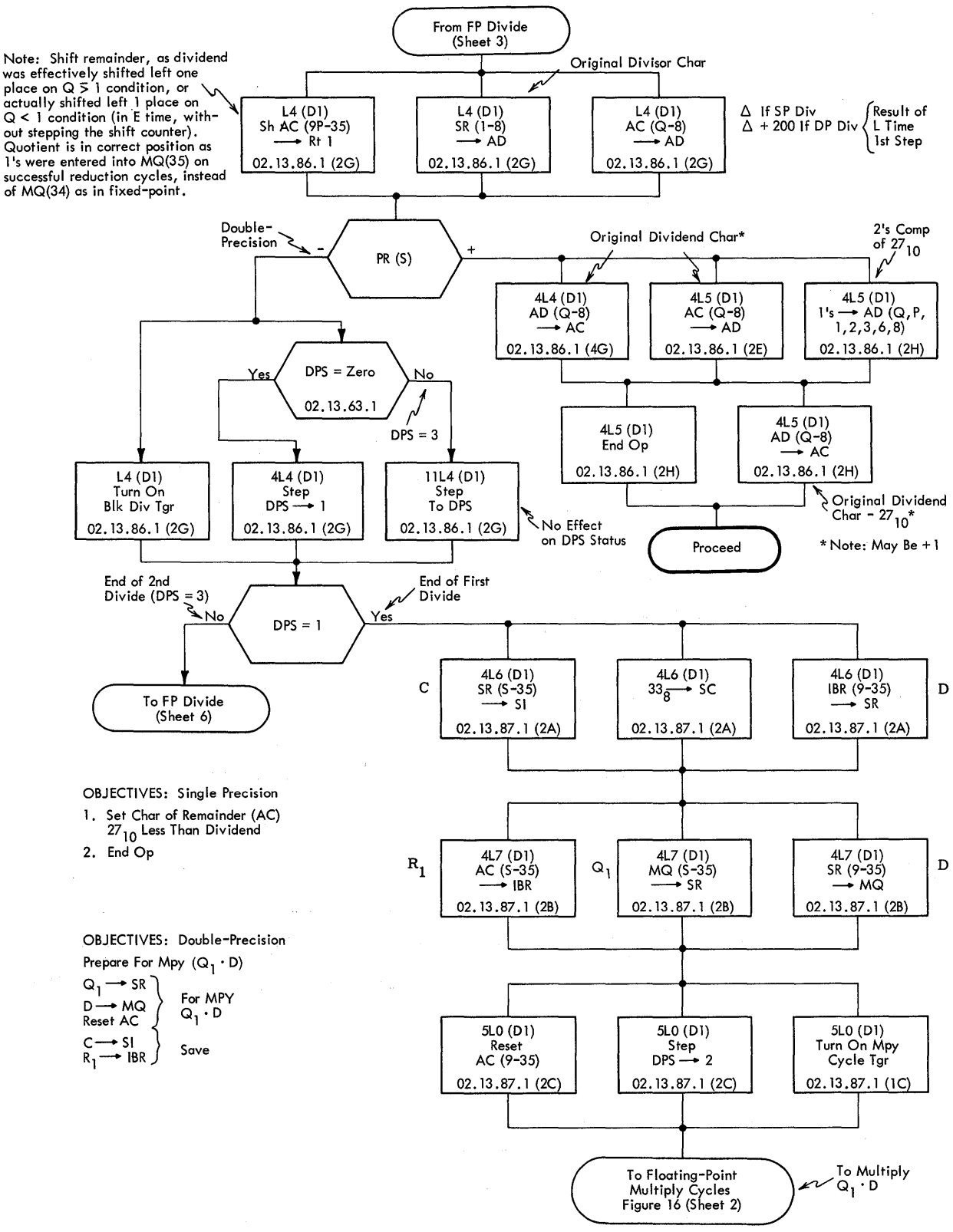


Figure 21. Floating-Point Divide (Single and Double-Precision)—Sheet 4 of 6

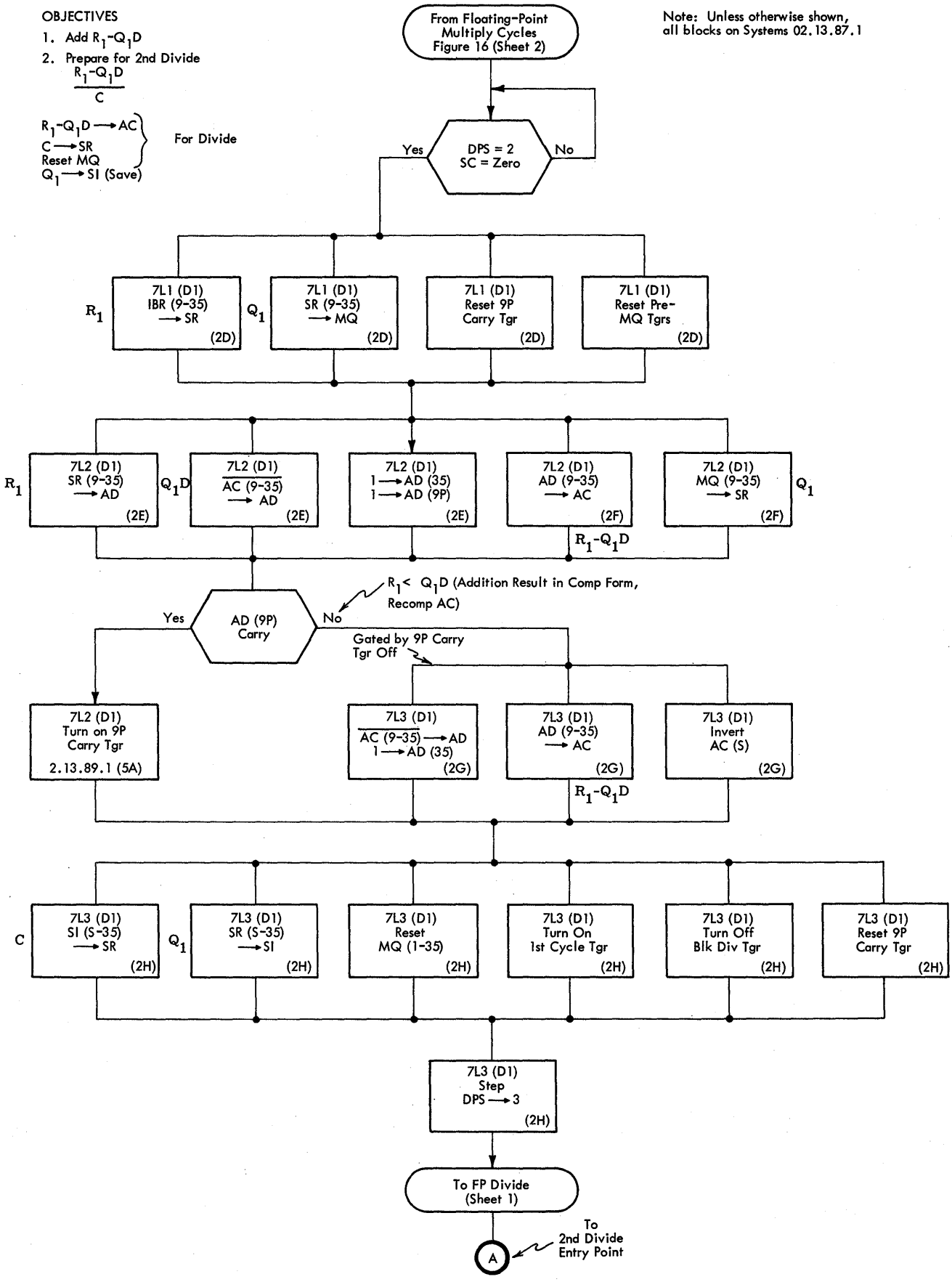


Figure 21. Floating-Point Divide (Single and Double-Precision)—Sheet 5 of 6

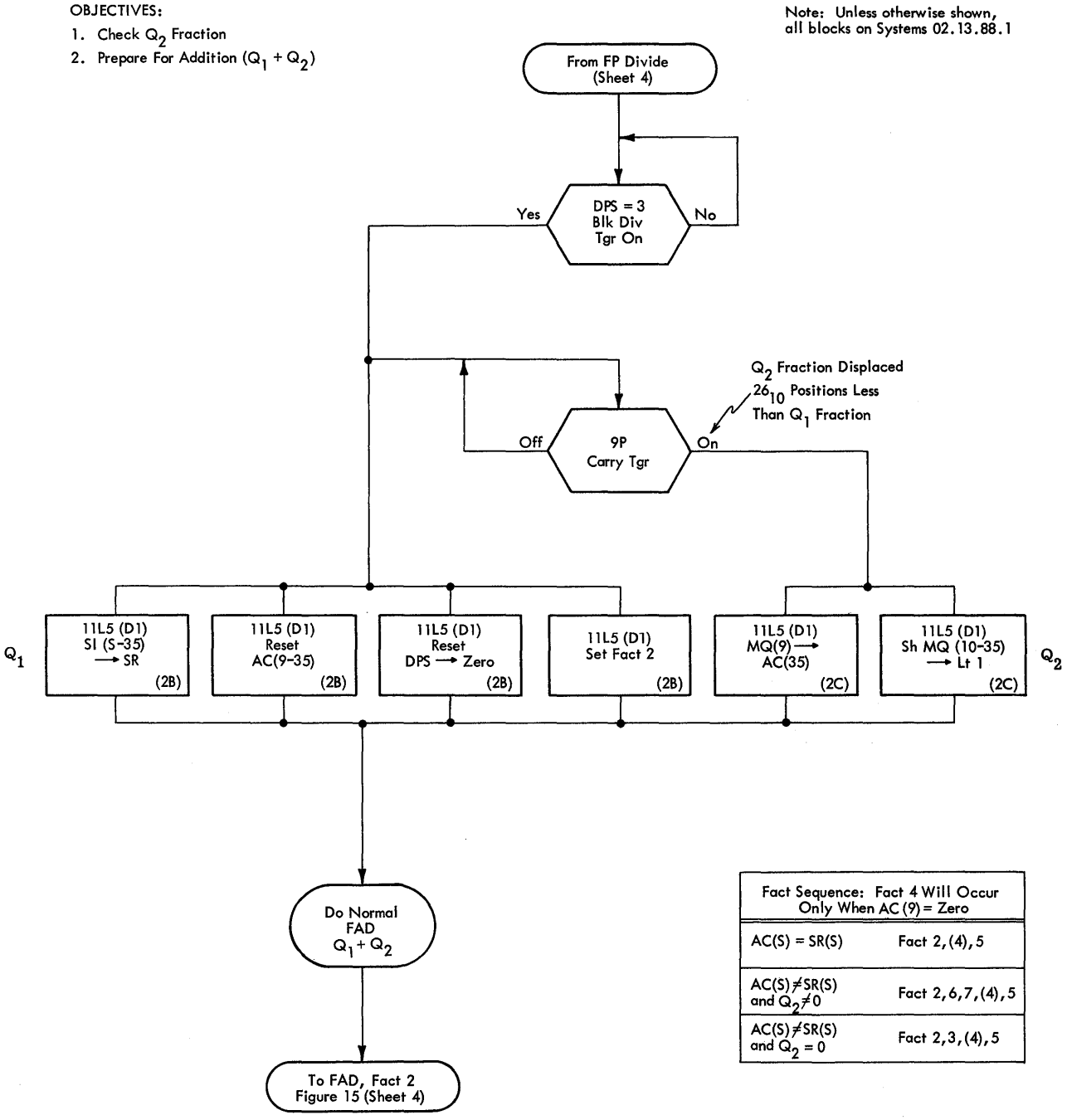


Figure 21. Floating-Point Divide (Single and Double-Precision)—Sheet 6 of 6

Appendix

Principal Triggers Used During Arithmetic Operations

TRIGGER NAME	SYSTEMS LOCATION
AC Ovfl	02.10.36.1
Blk FP Div	02.13.89.1
Carry Tgr	02.02.40.1
9 Carry	02.10.37.1
9P Carry	02.13.89.1
Clk Drive	02.13.44.1
DPS 1-3	02.13.63.1
Div Ck	02.10.53.1
End Op	08.00.09.2
FACT 1-7	02.13.49.1-02.13.59.1
FAD MQ Ovfl	02.13.57.1
First Cycle	02.13.89.1
FP Div	02.10.52.1
FP Ovfl #1	02.10.52.1
FP Ovfl #2	02.10.50.1
FP Trap	02.10.51.1
FP Trap Mode	02.10.71.1
MST	04.20.11.1
Mpy Cycle #1	02.13.77.1
Mpy Cycle #2	02.13.77.1
MQ Ovfl #1	02.10.51.1
MQ Ovfl #2	02.13.04.1
MQ String Bit	02.13.77.1
Ovlp Conflict	03.08.17.2
Pre-End Op	08.00.10.1
Pre MQ 34	02.13.77.1
Pre MQ 35	02.13.77.1
Pre MQ String Bit	02.13.77.1
Reset Add	02.13.61.1
SR (S)⟶AC (S)	02.13.61.1
T1	02.10.53.1

CUT ALONG LINE

COMMENT SHEET

IBM 7094 II DATA PROCESSING SYSTEM – VOLUME II
ARITHMETIC INSTRUCTIONS

CUSTOMER ENGINEERING INSTRUCTION–MAINTENANCE, FORM 223–2722–0

FROM

NAME _____

OFFICE NO. _____

FOLD

CHECK ONE OF THE COMMENTS AND EXPLAIN IN THE SPACE PROVIDED

FOLD

☐ SUGGESTED ADDITION (PAGE , TIMING CHART, DRAWING, PROCEDURE, ETC.)

☐ SUGGESTED DELETION (PAGE)

☐ ERROR (PAGE)

EXPLANATION

FOLD

FOLD

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES, STAPLE, AND MAIL

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: CE MANUALS, DEPARTMENT B96

CUT ALONG LINE

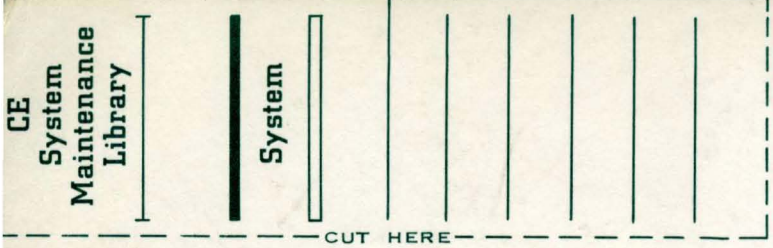
FOLD

FOLD

6/64:1.5M-FAW-80

STAPLE

STAPLE



223-2722-0

IBM 7094 II Printed in U.S.A. 223-2722-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601