SECTION 6 - ARITHMETIC SECTION

6.1. X-D' ADDER

6.1-1. OBJECTIVES

To present the detailed theory of operation involved in the X-D' adder.

6.1-2. INTRODUCTION

The X-D' adder is used in most addition, subtraction, multiplication, and division operations as well as in many other operations where it is used simply as a transmission path.

6.1-3. REFERENCES

a. UNIVAC 1219 Technical Manual, Volume I, Paragraph 4-4b(3).

b. UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.1-4. INFORMATION

a. General Description. The X-D' adder is 18 bit positions in length. The outputs of the X and D registers are hard-wired to the adder. The logic is so arranged that the adder actually performs the subtraction of X-D'. The adder's output can be taken to AU, AL, P, S1, and Z1 registers.

The adder is of the end-around borrow type. When a borrow is needed from beyond bit position 2^{17}, it is taken end-around from bit position 2^{00}. In this sheet, a borrow being subtracted from a bit position is described as being applied to that bit position. Borrow and borrow request have the same meaning.

b. Adder Theory.

1. Half-Subtract Method of Subtraction. The X-D' adder employs the half-subtract method of subtraction. Refer to table 6.1-1 for the truth table description of half-subtraction.

The half-subtract is formulated by subtracting each bit position without considering borrows. Each bit position half-subtract is entirely independent of other bit positions.

After the half subtract for each bit position is produced, borrows are considered. If no borrow is applied to a bit position, the adder output is the half-subtract (HS). If a borrow is applied, the adder outputs the complement of the half-subtract (HS'). Refer to figure 6.1-1 for an example of the half-subtract method of subtraction with end-around borrow.

TABLE 6.1-1. HALF-SUBTRACT TRUTH TABLE

ADDER INPUTS	HALF SUBTRACT
00	0
01	1
10	1
11	0

BIT POSITIONS	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	0	1	0	1	0	1	0	0	1	0	0	1	1	0	0	1	1	0
−0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0	
HALF SUBTRACT =	0	0	1	0	0	1	0	0	1	1	1	1	1	1	0	0	1	0
OUTPUT TERM =	HS'	HS'	HS	HS	HS	HS	HS	HS	HS'	HS'	HS	HS	HS'	HS	HS	HS	HS'	HS'
BINARY OUTPUT =	1	1	1	0	0	1	0	0	0	0	1	1	0	1	0	0	0	1

Figure 6.1-1. Example of Half-Subtract Method of Subtraction

The result produced is the same as can be obtained by the normal pencil and paper method. A summation of the borrow conditions in this example is as follows:

a) A borrow is applied to bit position 05 from bit position 04.

b) A borrow is applied to bit positions 08 and 09 from bit position 07.

c) A borrow is applied to bit positions 17 and 16 and end-around to 01 and 00 from bit position 15.

Hereafter, the term stage is used in place of bit position.

2. Generation of a Borrow Request. There is only one bit configuration which will generate a request for a borrow, referred to simply as a borrow. A borrow is generated if, in the same stage, X is 0_2 and D' is 1_2. The number of stages affected by this borrow is dependent upon the configurations of the more significant stages.

3. Borrow Enable. There is only one bit configuration which will satisfy or absorb a borrow. If a borrow is applied to it, the bit inputs can supply the required 1_2 without propagating the borrow request to the next higher stage. A borrow enable condition exists if, in the same stage, X is 1_2 and D' is 0_2. In

this condition, a borrow will affect only this stage and will not be transmitted to higher stages.

c. Detailed Analysis.

1. Adder Logic Technique.

a) Formulation of HS and HS'. Each adder stage has logic circuitry which produces the half subtract and its complement. Refer to figure 6.1-2 for an example using stage 00.

In this example 10A00 can be considered to test X_{00} and D'_{00} for the two possible configurations resulting in HS = 1_2. With the use of 11A00, high levels for both HS = 1_2 and HS' = 1_2 are available.

b) Application of Borrow. Either the HS or HS' result is outputted from the adder depending upon the application of borrows to the particular stage. If no borrow is applied, HS is the final result. If a borrow is applied, HS' is the final result. Refer to figure 6.1-3 for an example.

If no borrow is applied to stage 00, 13A00 is a high level input and 12A00 is a low level input. This condition allows the adder output to be determined by input 10A00. 14A00 will output a low level if $HS_{00} = 1_2$.

In this example, the output is taken to AL. AL is initially cleared to 0's. Then the Adder $\rightarrow$AL signal occurs. AL_{00} is set to 1_2 only if 14A00 outputs a low level which represents the adder output of 1_2.

If a borrow is applied to stage 00, 12A00 is a high level input and 13A00 is a low level input. This condition allows the adder output to be determined by input 11A00. 14A00 will output a low level if $HS'_{00} = 1_2$.

2. Adder Layout. The X-D' adder is divided into three sections comprised of stages 05-00, 11-06, and 17-12. Each section has circuitry to sense borrows applied from other sections. Refer to figure 6.1-4 for a block diagram illustrating the section interconnection. Inter-section borrow requests are discussed later in this sheet.

Each stage of the adder senses borrows generated by the less significant stages in the section and the enable conditions of these bits in order to detect borrows which might affect its own output.

So as to speed the borrow propagation time, each 6-stage section is further separated into two groups.

3. Individual Stage.

a) Stage 00. Refer to figures 6.1-2, 6.1-3, and logic diagrams, figure 9-94. Review the operation of this stage as described previously. A borrow applied to this stage is always an end-around borrow.

b) Stage 01. Refer to figure 6.1-5 and logic diagrams, figure 9-94. The only difference from stage 00 is the borrow sensing logic. 12A01 outputs a high level if a borrow is applied. There are only two possible conditions which can apply a borrow to this stage. Refer to table 6.1-2 for these conditions.

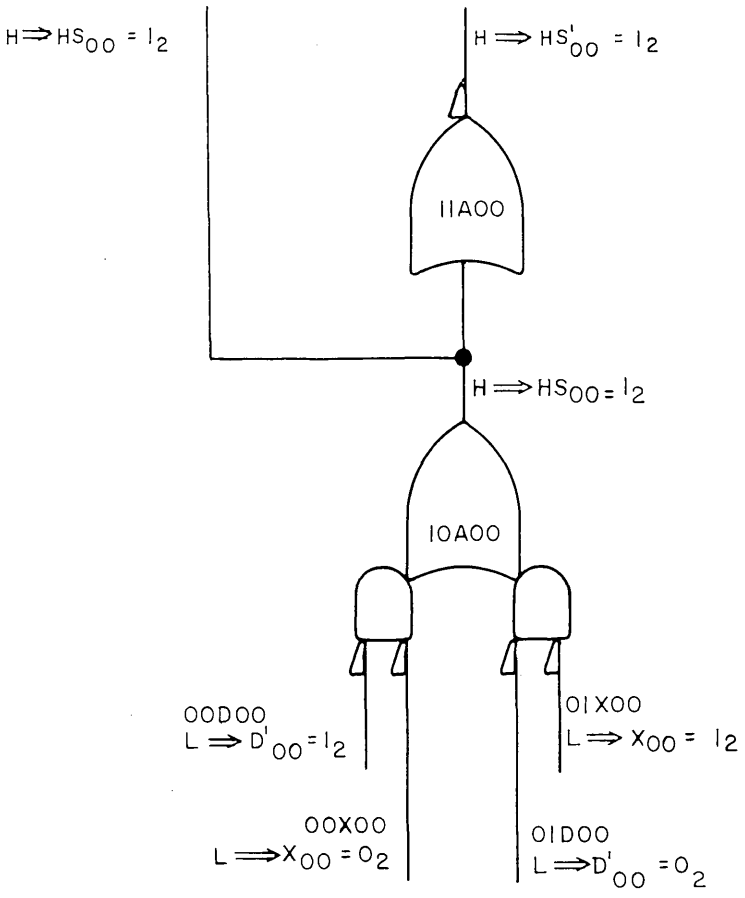


Figure 6.1-2. X-D' Adder HS and HS' Logic, Stage 00

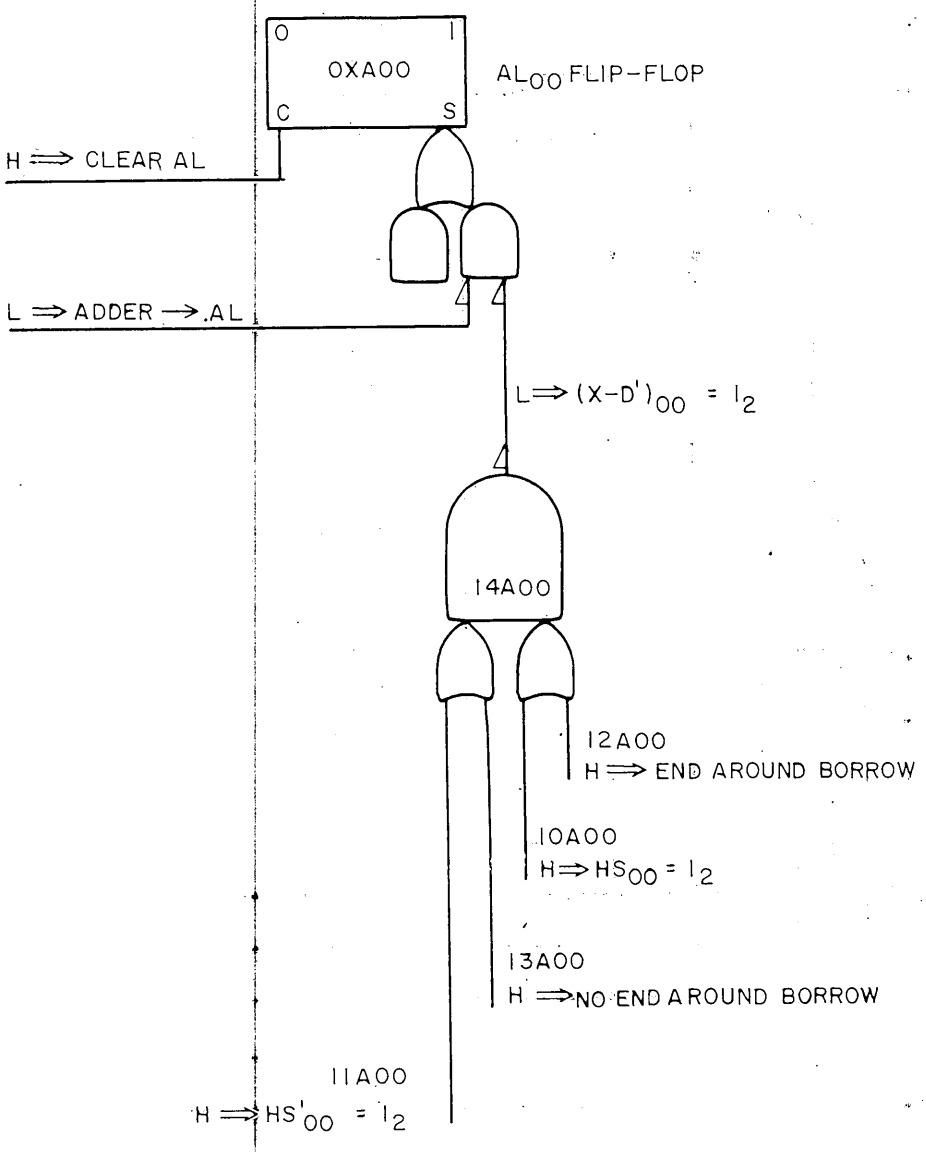


Figure 6.1-3. X-D' Adder Output Logic, Stage 00

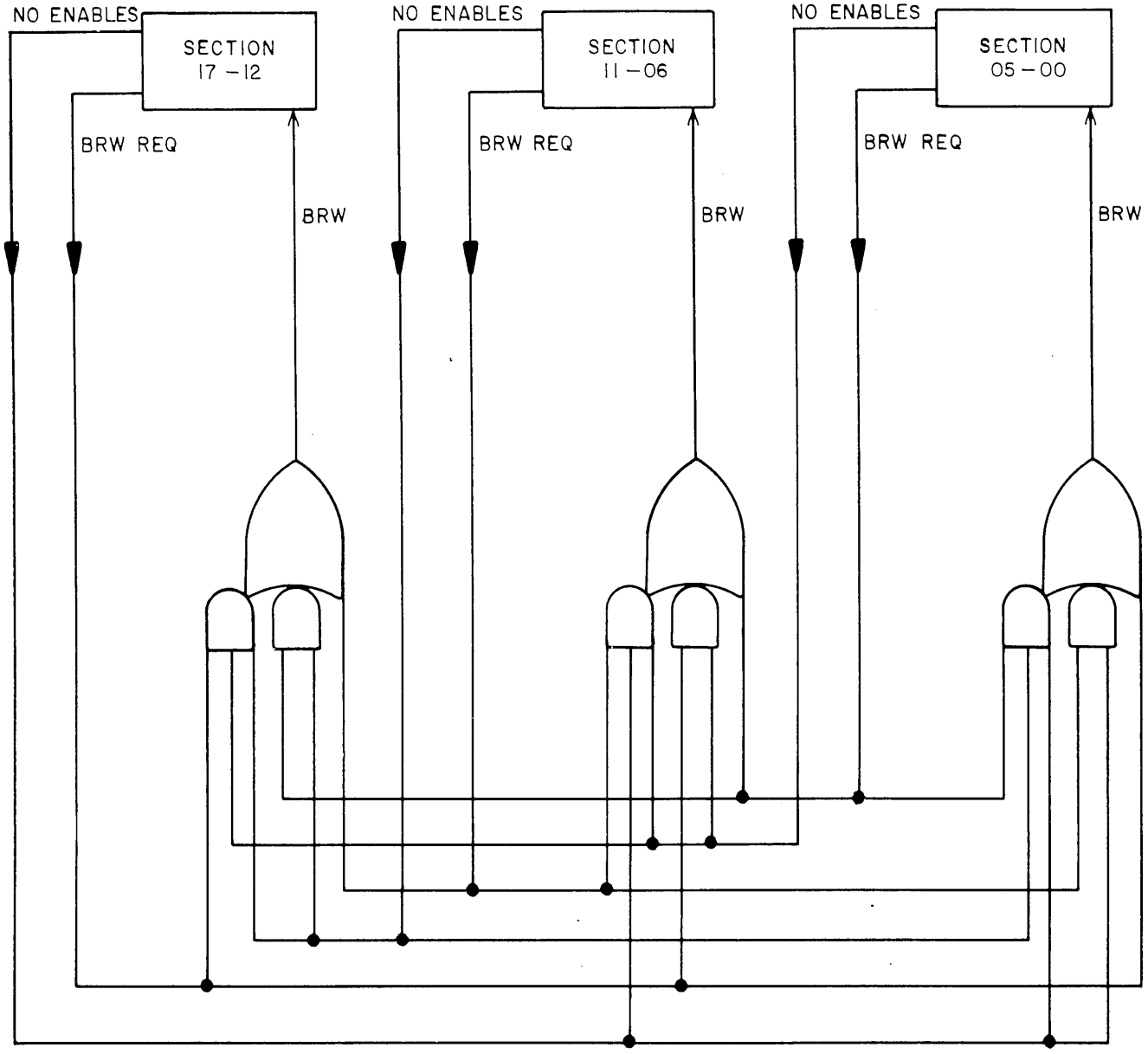


Figure 6.1-4. X-D' Adder Inter-Section Block Diagram

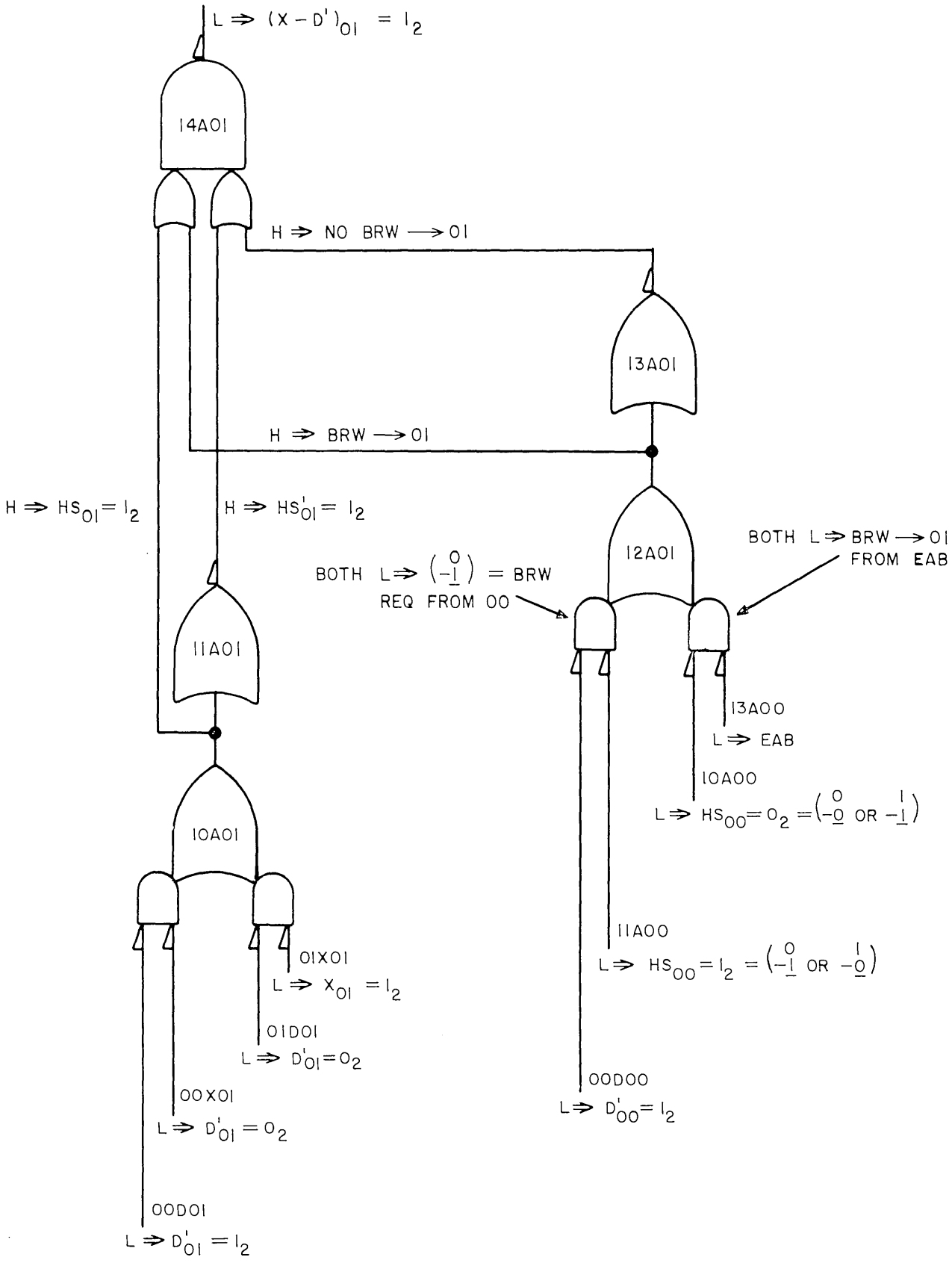


Figure 6.1-5. X-D' Adder, Stage 01

TABLE 6.1-2. X-D' ADDER, BORROWS APPLIED TO STAGE 01

CONDITIONS	INPUT LEVELS TO 12A01
Brw Req from 00	00D00 & 11A00 = L's
No enable & no Brw Req in 00, EAB	10A00 & 13A00 = L's

c) Stage 02. Refer to figure 6.1-6 and logic diagrams, figure 9-94. 12A02 outputs a high level if a borrow is applied. There are only three possible conditions which can apply a borrow to this stage. Refer to table 6.1-3 for these conditions.

TABLE 6.1-3. X-D' ADDER, BORROWS APPLIED TO STAGE 02

CONDITIONS	INPUT LEVELS TO 12A02
Brw Reg from 01	00D01 & 11A01 = L's
No Enable & no Brw Req in 01, Brw Req from 00	11A00, 00D00, & 10A01 = L's
No Enable & no Brw Req in 01 & 00, EAB	10A01, 10A00, & 13A00 = L's

d) Stage 03. Each 6-stage section is separated into two groups, as mentioned previously. The three stages discussed above comprise one group. Stage 03 is the start of the second three-stage group. So as to speed the borrow request propagation, the stage 03 borrow request detection logic is directly fed by the X and D register flip-flops for stages 02 through 00. Refer to figure 6.1-7 and logic diagrams, figure 9-91, for a portion of the stage 03 logic.

This logic simply provides stage 03 with the borrow request and enable status of stages 02 through 00. 20A00 determines if any of these lower three stages has an enable condition which would satisfy and stop a borrow request. An enable condition exists if X is 1_2 and D' is 0_2 in the same stage. If all input OR gates to 20A00 are satisfied, each of the stages contains no enable. 21A00 passes this information to the remainder of the stage 03 logic.

30A00 is used to detect borrows applied to stage 03 from within stages 02 through 00. Refer to table 6.1-4 for these borrow conditions.

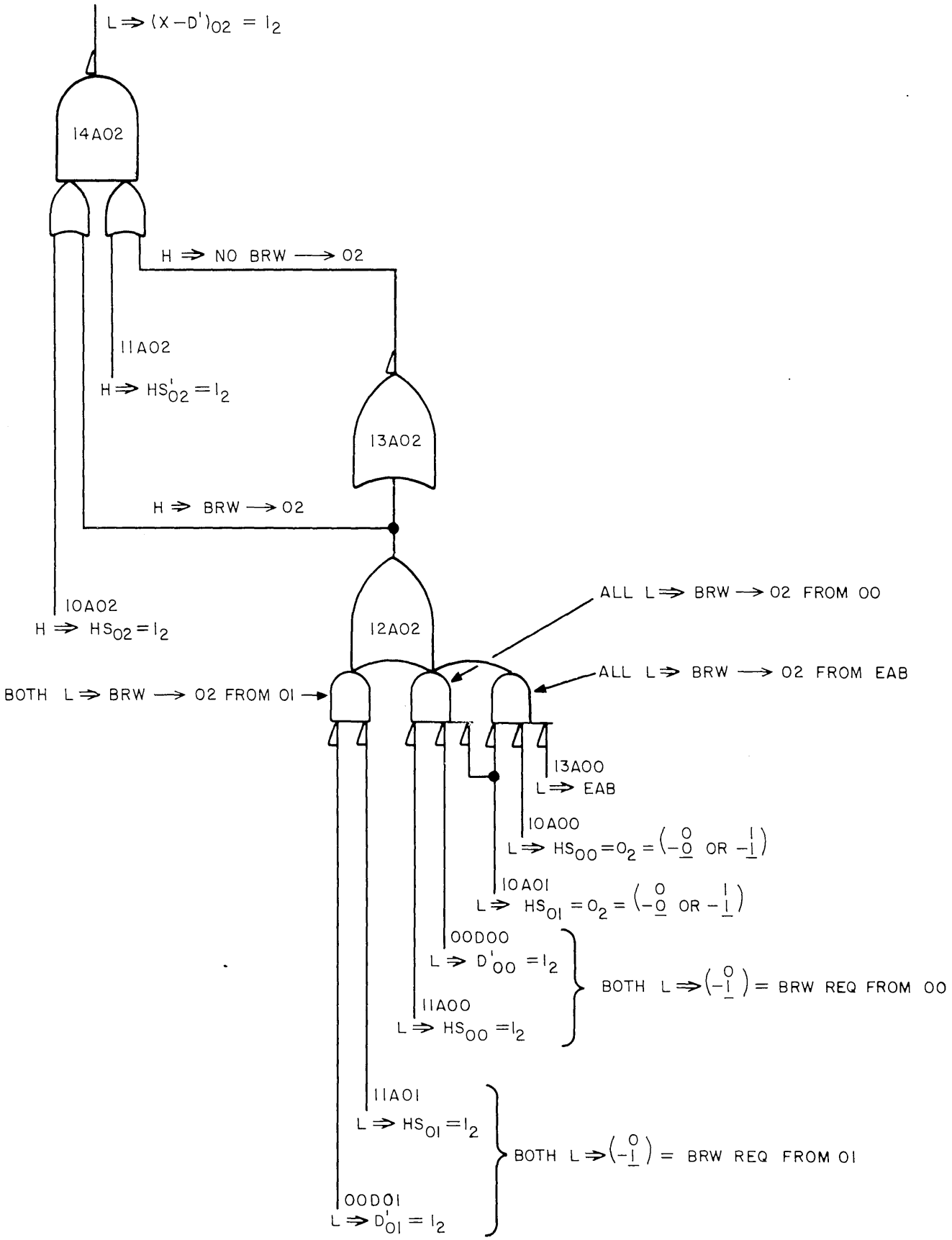


Figure 6.1-6. X-D' Adder, Stage 02

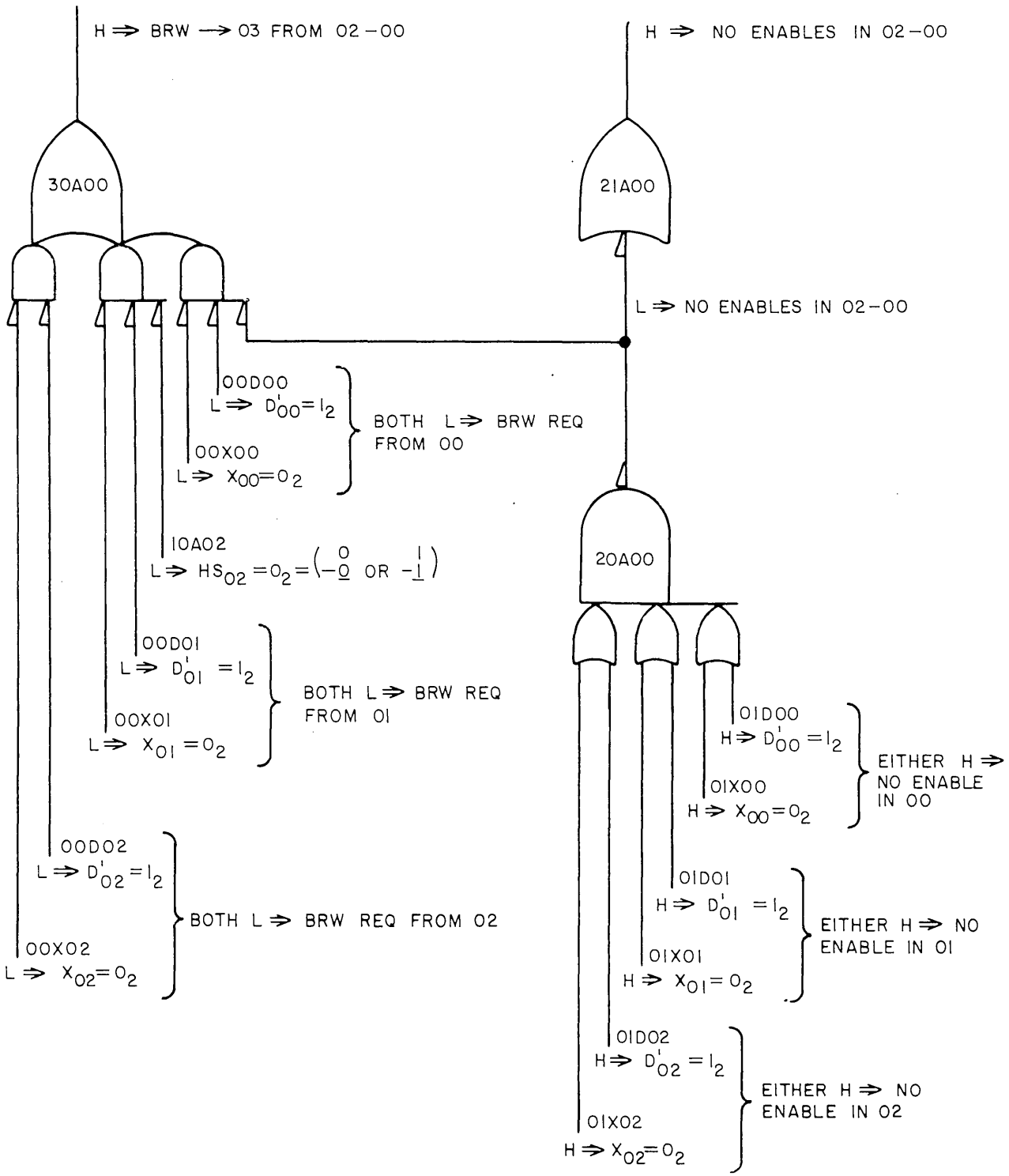


Figure 6.1-7. X-D' Adder, Stage 03 Preliminary Borrow Request Detection Logic

TABLE 6.1-4. X-D' ADDER, BORROWS APPLIED TO STAGE 03 FROM 02-00

CONDITIONS	INPUT LEVELS TO 30A00
Brw Req from 02	00X02 & 00D02 = L's
No enable & no Brw Req in 02, Brw Req from 01	00X01, 00D01, & 10A02 = L's
No Enables in 02-00, Brw Req in 00	00X00, 00D00, & 20A00 = L's

Refer to figure 6.1-8 and logic diagrams, figure 9-94.

12A03 receives the outputs of 12A00 and 30A00 discussed above. If a borrow is applied to stage 03, 12A03 outputs a low level. There are two conditions which can create this borrow. Input 30A00 satisfies 12A03 if a borrow is generated within stages 02 through 00 and is propagated from this group.

Input 12A00 applies a borrow to this stage if there is an end-around borrow which cannot be satisfied within stages 02 through 00. 21A00 inputs a high level to enable the borrow propagation to stage 03.

e) Stages 17-04. The logic for these stages is similar to that described above. Refer to logic diagrams, figures 9-91 through 9-96 for the logic concerning these stages. Inter-section borrows are discussed later in this sheet.

4. Generation of Inter-Section Borrow and Enable Signals. Section 05-00 is used for an example. Refer to figure 6.1-9 and logic diagrams, figure 9-91.

The logical functions of 30A03, 20A03, and 21A03 are the same as for 30A00, 20A00, and 21A00 described previously. Only the stage numbers are different. Therefore, 22A00 outputs a low level if there are no enables in this section (05-00). This signal indicates to other sections that if a borrow should be applied to section 05-00, it would not be satisfied and would be propagated to the next higher section.

31A00 outputs a low level to indicate that a borrow request is generated in section 05-00 and is not satisfied within this section. Therefore, this low level would apply a borrow to the next higher section. Refer to table 6.1-5 for the borrow generating conditions.

TABLE 6.1-5. X-D' ADDER, BORROW REQUESTS FROM SECTION 05-00

CONDITIONS	INPUT LEVELS TO 31A00
Brw Req from 05-03	30A03 = H
No Enables in 05-03, Brw Req from 02-00	30A00 & 21A03 = H's

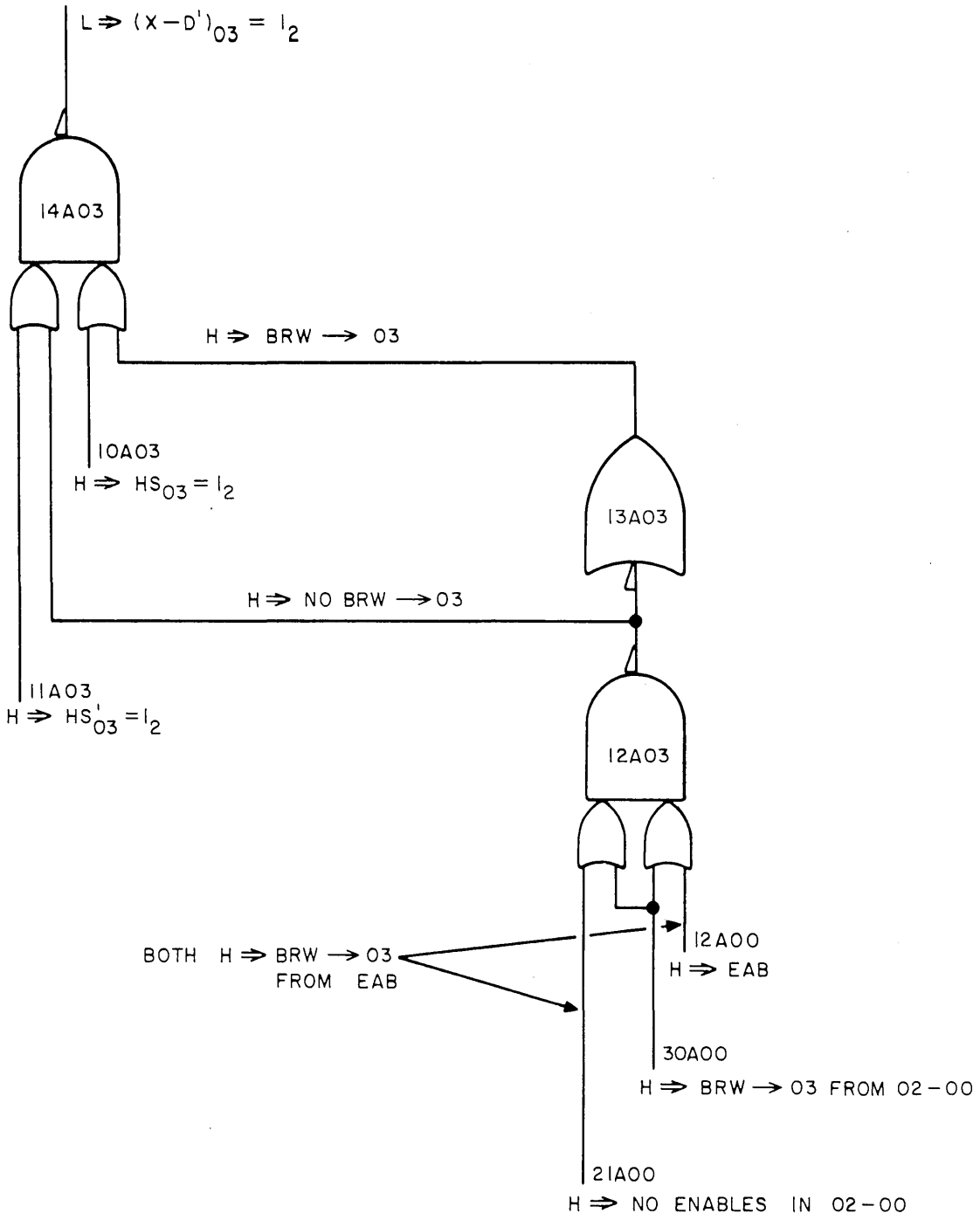


Figure 6.1-8. X-D' Adder, Stage 03 Final Portion

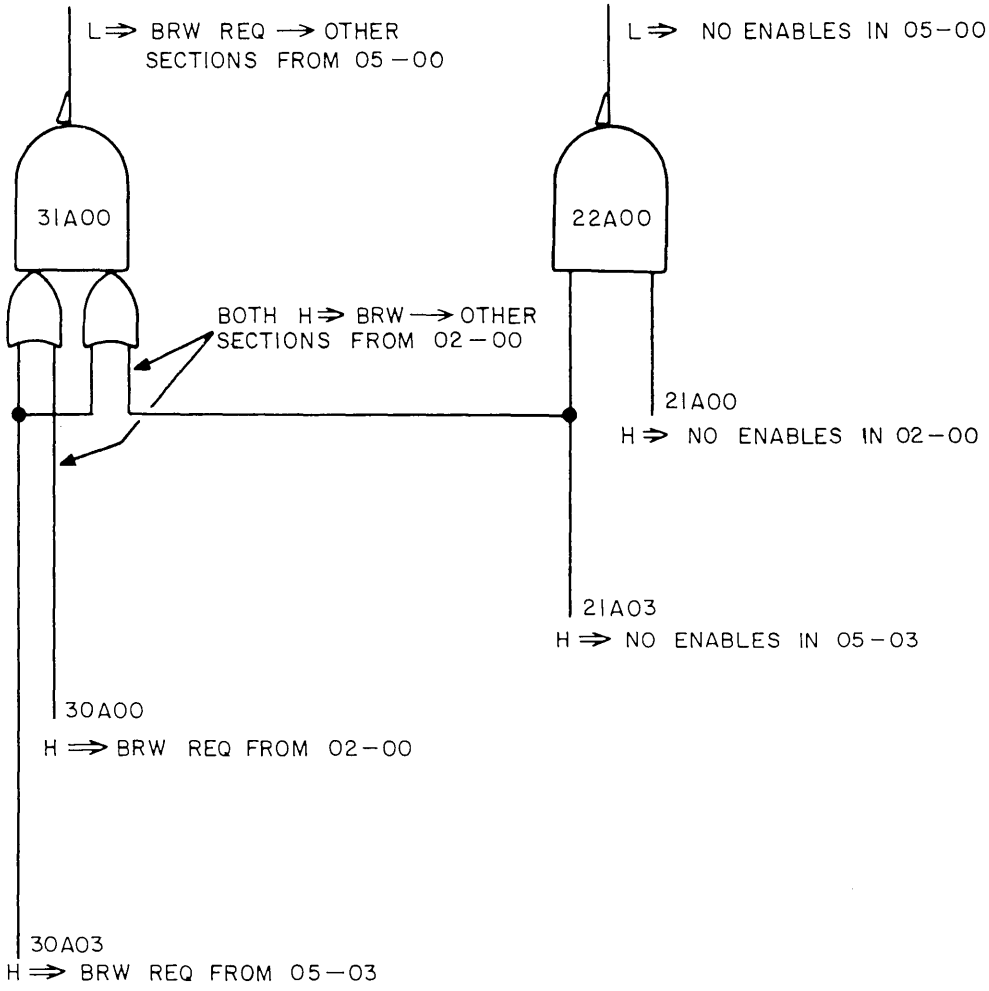


Figure 6.1-9. X-D' Adder, Inter-Section Borrow Request and Enable Generation

The other two sections generate these same borrow request and enable signals in the same manner. Section 17-12 considers two additional factors in developing these outputs. This section is affected by the status of the Inhibit EAB and Insert EAB flip-flops. If the Insert EAB flip-flop is set, a simulated borrow request is generated by section 17-12. If the Inhibit EAB flip-flop is set, a borrow request (other than the simulated request) from section 17-12 is inhibited. Also, this section is forced to indicate that it contains an enable condition. This simulated enable causes a borrow which is applied to this section from either section 11-06 or section 05-00 to be satisfied. Thus, an end-around borrow is inhibited.

Refer to logic diagrams, figure 9-93 for the effect of the Inhibit EAB and Insert EAB flip-flops.

22A12 outputs a low level if there are no enables in section 17-12 and the Inhibit EAB flip-flop is clear. 31A12 outputs a low level if there is a borrow request from section 17-12 and the Inhibit EAB flip-flop is clear or if the Insert EAB flip-flop is set.

Refer to logic diagrams, figure 9-92 for the borrow request and enable signals generation logic for section 11-06.

5. Detection of Inter-Section Borrow Requests. Each section evaluates the status of the other sections to determine whether there is a borrow applied to it. A borrow is applied to a section only if there are no enable conditions existing between that section and the section which generated the borrow request. Refer to figure 6.1-4 for a review of the section interconnection.

Refer to figure 6.1-10 and logic diagrams, figures 9-94 through 9-96 for the intersection borrow request detection logic.

Each gate shown in figure 6.1-10 senses three conditions which can apply a borrow to its section. Refer to table 6.1-6, 6.1-7 and 6.1-8 for these conditions.

6.1-5. SUMMARY

The X-D' adder is an 18-bit, end-around borrow adder. It is separated into three sections so as to speed the borrow request signal propagation. The X and D registers are hard-wired inputs to the adder logic. The adder's result is not used until the desired values have been entered into X and D and enough time has expired to allow propagation of any borrows.

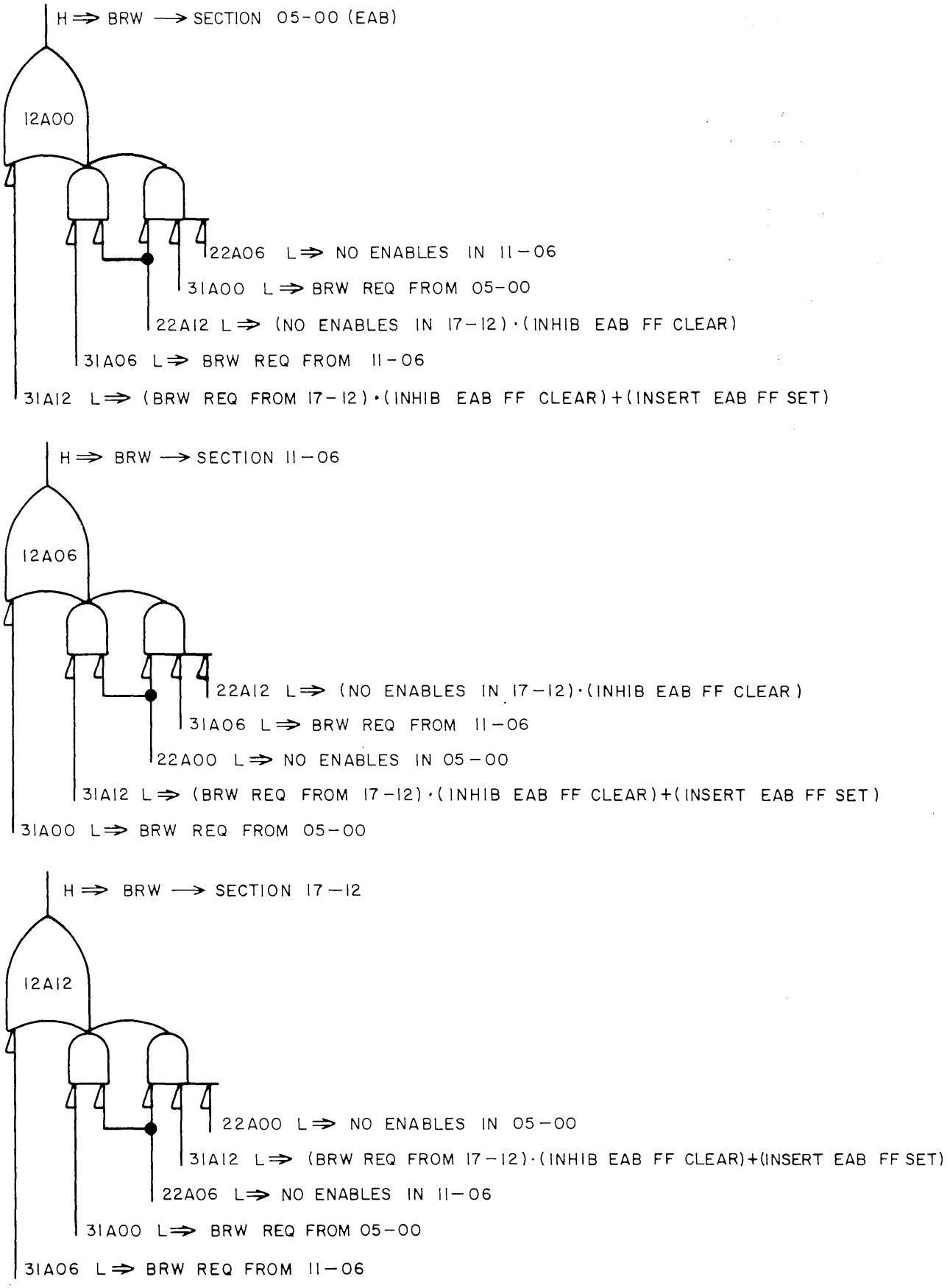


Figure 6.1-10. X-D' Adder, Inter-Section Borrow Request Detection

TABLE 6.1-6. X-D' ADDER, INTER-SECTION BORROWS APPLIED TO SECTION 05-00

SECTION 17-12	SECTION 11-06	SECTION 05-00	INPUT LEVELS TO 12A00
Brw Req			31A12 = L
No Enables	Brw Req		31A06 & 22A12 = L's
No Enables	No Enables	Brw Req	22A12, 31A00, & 20A06 = L's

TABLE 6.1-7. X-D' ADDER, INTER-SECTION BORROWS APPLIED TO SECTION 11-06

SECTION 17-12	SECTION 11-06	SECTION 05-00	INPUT LEVELS TO 12A06
		Brw Req	31A00 = L
Brw Req		No Enables	31A12 & 22A00 = L's
No Enables	Brw Req	No Enables	22A00, 31A06, & 22A12 = L's

TABLE 6.1-8. X-D' ADDER, INTER-SECTION BORROWS APPLIED TO SECTION 17-12

SECTION 17-12	SECTION 11-06	SECTION 05-00	INPUT LEVELS TO 12A12
	Brw Req		31A06 = L
	No Enables	Brw Req	31A00 & 22A06 = L's
Brw Req	No Enables	No Enables	22A06, 31A12, & 22A00 = L's

NAME: _____

6.1-6. STUDY QUESTIONS

a. Given: AL_i = 612160
SR = 01010_2
instruction = 141000
content of address 21000 = 161307

Refer to logic diagrams, figures 9-91 through 9-96. Give the output logic levels for the following gates which exist during the time that the X-D' adder is used in the execution of the above instruction to add AL + Y.

31A00 = _____ 31A12 = _____

22A00 = _____ 22A12 = _____

31A06 = _____ 12A00 = _____

22A06 = _____ 12A06 = _____

12A12 = _____

b. Given conditions same as above.

At the completion of the instruction, the final content of AL is 773463_8. Refer to logic diagrams, figure 9-94. Assume each of the following malfunctions to occur individually. Determine whether each malfunction would cause the erroneous AL result. Indicate your answers by writing "yes" or "no" beside each malfunction condition.

Grounded Output	Constant Low Level Output
_____ 12A00	_____ 12A00
_____ 10A02	_____ 10A02
_____ 11A02	_____ 11A02
_____ 12A02	_____ 12A02
_____ 13A02	_____ 13A02
_____ 14A02	_____ 14A02

c. Refer to logic diagrams, figures 9-94 and 9-96.

Given: 12A00 output is a low level.
12A12 output is a high level.
14A15 output is a low level.

bit positions	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}
X =	?	1	0	?	0	1
D =	?	0	0	?	1	0

There are no malfunctions. Use the above information to determine the binary contents of X and D bit positions 2^{17} and 2^{14}. Indicate your results below. Notice that D, rather than D', is referenced.

X_{17} = _____

X_{14} = _____

D_{17} = _____

D_{14} = _____

SECTION 6 - ARITHMETIC SECTION

6.2. INSTRUCTION EXECUTION OF SKPODD, SKPEVN AND PARITY EVALUATOR

6.2-1. OBJECTIVES

To present the detailed theory of operation involved in the execution of the parity instructions and the parity evaluator.

6.2-2. INTRODUCTION

Parity refers to the number of 1_2's, thus, every word has either odd or even parity. The parity evaluation function of the 1219 is useful in determining whether a data word has lost or gained a 1_2.

6.2-3. REFERENCES

UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.2-4. INFORMATION

a. SKPODD, SKPEVN Instructions.

1. General Description.

a) Instruction Interpretation.

1) SKPODD, f = 50:54. This instruction formulates the logical product (AND function) of AU · AL. The parity of their result is evaluated and causes a program skip of the next sequential instruction if the parity is odd. The contents of AU and AL are not disturbed.

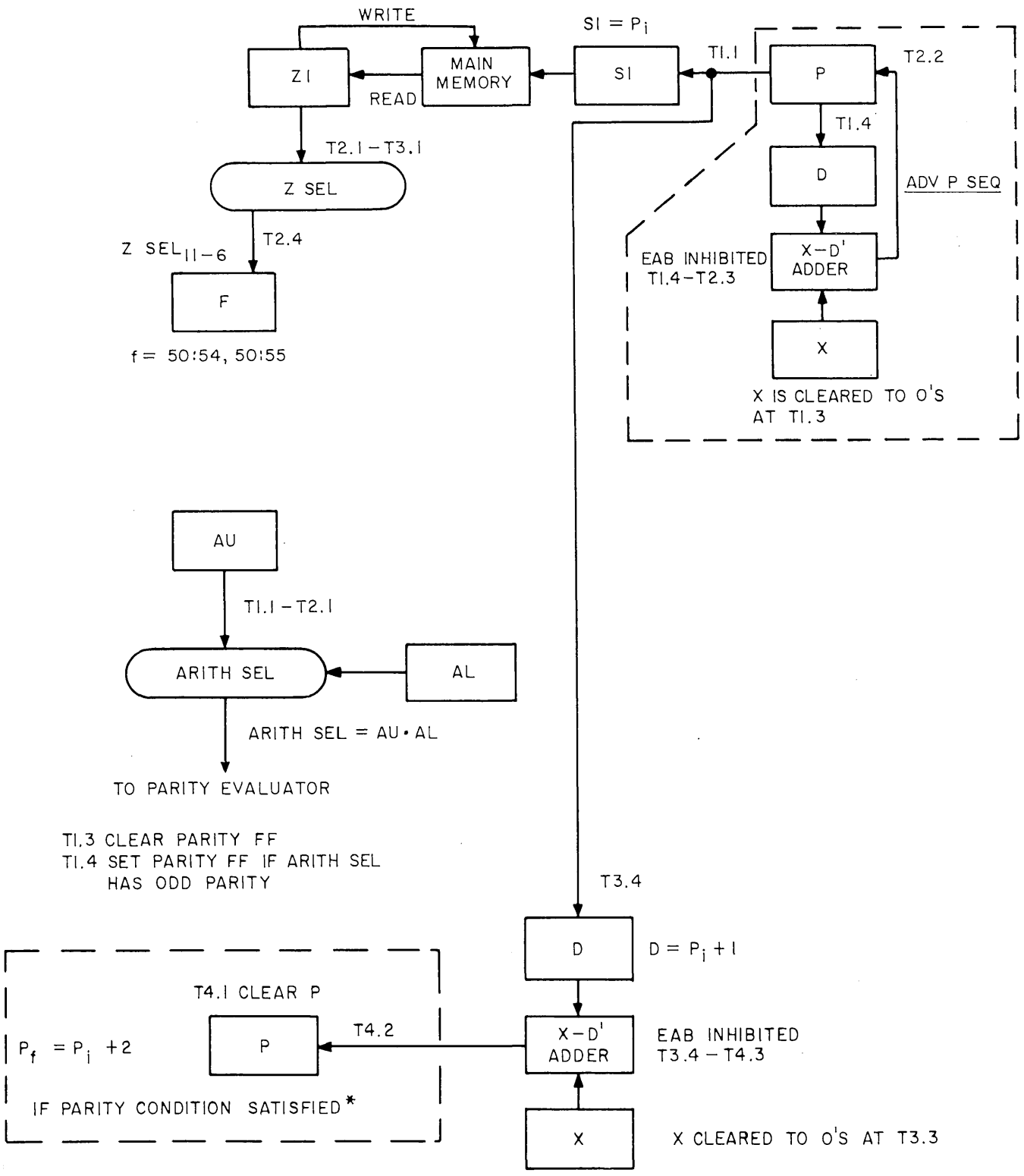
2) SKPEVN, f = 50:55. Except for the effect of the parity evaluation, this instruction is the same as f = 50:54. A program skip is performed if the parity is even.

b) Execution Sequence (I). All operations are performed within the I-sequence. Only the one memory reference to obtain the instruction is necessary.

2. Detailed Analysis.

a) Data Flow Block Diagram. Refer to figure 6.2-1 for a block diagram description of the execution of f = 50:54, 50:55.

Most of the I-sequence operations are as previously described. If necessary, refer to study guide sheet number 5.4 for a detailed analysis.



NOTE: * SKIP IF: (f = 50:54) · (PARITY FF SET)
(f = 50:55) · (PARITY FF CLEAR)

Figure 6.2-1. I-Sequence Data Flow for f = 50:54, 50:55

During the set time of the T14 flip-flop (T1.1-T2.1) both AU and AL are applied to arithmetic-select which formulates their logical product. The Parity flip-flop is used to record the parity of this result. This flip-flop affects only the f = 50:54, 50:55 skip evaluation.

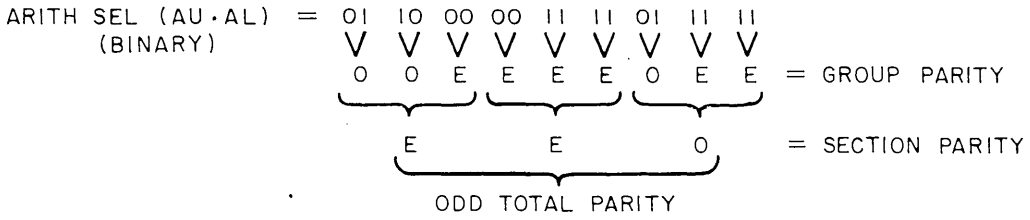
The X-D' adder is used to increment P by +1 a second time just as is done by the advance-P subsequence. If P receives the result of this second incrementation, the next sequential instruction will be skipped.

As discussed in a later sheet, the instruction could be obtained from bootstrap or control memory.

b) Essential Commands. Refer to table 6.2-1 for a sequential list of essential I-sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams. The parity evaluator is presented later in this sheet.

b. Parity Evaluator.

1. General Description. The parity evaluator has hard-wired inputs from arithmetic-select. The parity logic is comprised of several stages. The initial stages separate the arithmetic-select inputs into groups of two bits each. The parity indications from these groups are then combined into sections of three groups each. Total parity is then evaluated from the combination of the three sections. Refer to figure 6.2-2 for an example.



NOTES: "E" MEANS EVEN.
"O" MEANS ODD.

Figure 6.2-2. Parity Evaluation Example

2. Detailed Analysis.

a) Group Parity. Each group is comprised of two bits. Bits 1 and 0 are used for an example. Refer to figure 6.2-3 and logic diagrams, figure 9-101.

15X00 tests bits 1 and 0 of arithmetic-select for the two possible even parity conditions. Refer to table 6.2-2 for the four possible configurations of these bits.

TABLE 6.2-1. I SEQUENCE ESSENTIAL COMMANDS FOR f = 50:54, 50:55

TIME NOTATION	COMMANDS
T4.4	Clear S1
T1.1	P → S1, Init Memory, *set Incr P ff, AU → Arith Sel, AL → Arith Sel
T1.3	*Clear D, *clear X, clear Z1, clear F, *set OXL11 ff, clear Parity ff
T1.4	*P_L → D_L, *P_U → D_U, *set Inhib EAB ff, set Parity ff if Arith Sel = odd parity
T2.1	*Clear P, Z1 → Z Sel, *clear Incr P ff, drop AU Arith Sel, drop AL → Arith Sel
T2.2	*Adder → P
T2.3	*Clear OXL11 ff, *clear Inhib EAB ff
T2.4	Z Sel_{11-6} → F, set OXF06 ff
T3.1	Drop Z1 → Z Sel
T3.3	Clear D, clear X
T3.4	P_L → D_L, P_U → D_U, set Inhib EAB ff
T4.1	Clear P if skip satisfied**
T4.2	Adder → P if skip satisfied**
T4.3	Clear Inhib EAB ff

* These events are concerned with or are controlled by the advance-P subsequence.

** Skip condition is satisfied if: (f = 50:54) · (Parity ff set)
(f = 50:55) · (Parity ff clear)

TABLE 6.2-2. PARITY EVALUATOR, GROUP 1 & 0 CONDITIONS

2^1	2^0	GROUP PARITY	15X00 OUTPUT
0	0	Even	H
0	1	Odd	L
1	0	Odd	L
1	1	Even	H

The logic for the other groups is the same except for the bits being sensed.

b) Section Parity. Each section is comprised of three groups (six bits). Section 5-0 is used for an example. Refer to figure 6.2-3 and logic diagrams, figure 9-101.

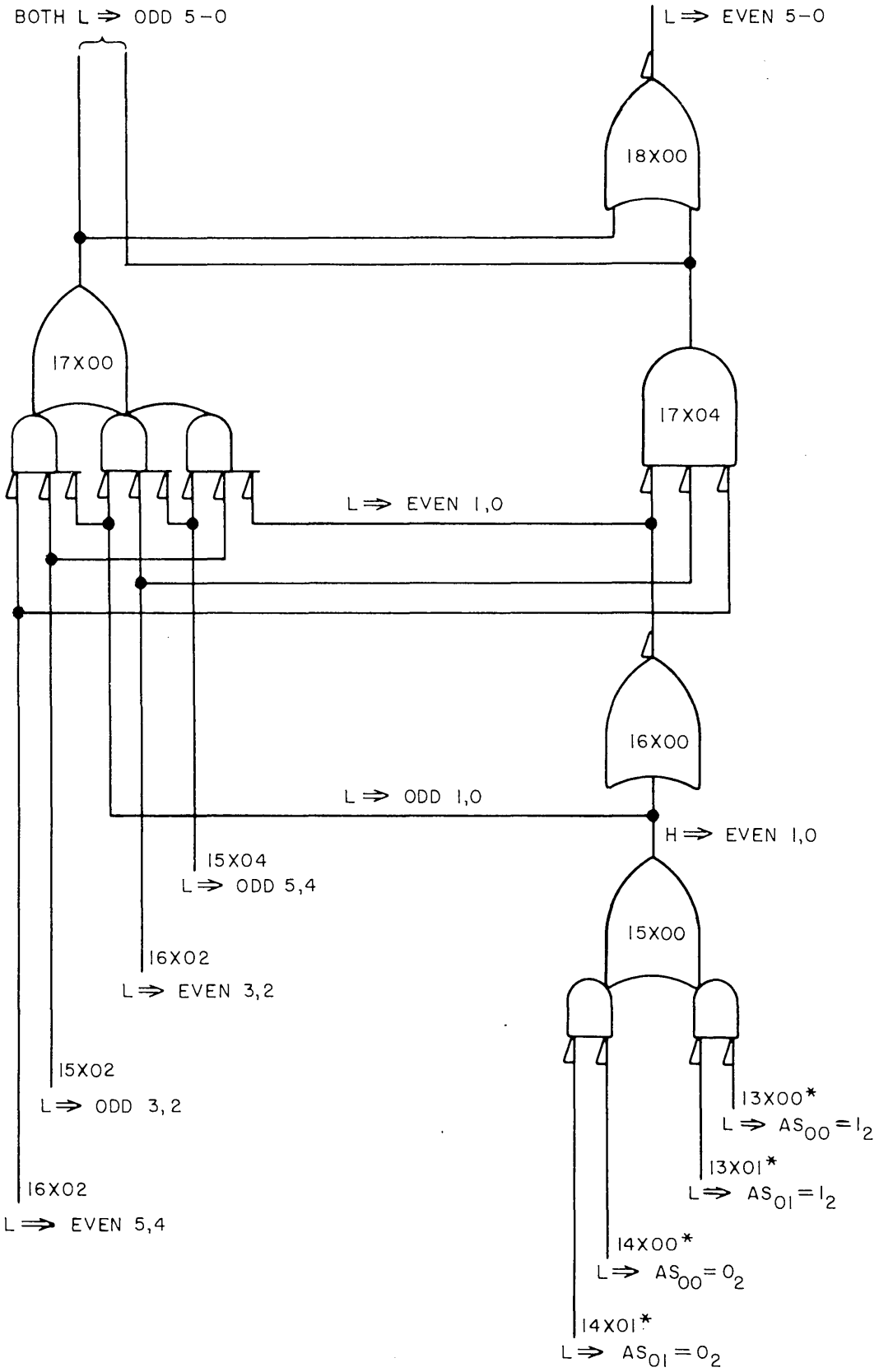
18X00, 17X00, and 17X04 test the parity conditions of the three groups (5,4; 3,2; and 1,0). Refer to table 6.2-3 for the eight possible parity configurations.

TABLE 6.2-3. PARITY EVALUATOR, SECTION 5-0 CONDITIONS

GROUPS			SECTION	OUTPUT LEVELS		
5,4	3,2	1,0	PARITY	17X04	17X00	18X00
Odd	Odd	Odd	Odd	L	L	H
Odd	Odd	Even	Even	L	H	L
Odd	Even	Odd	Even	L	H	L
Odd	Even	Even	Odd	L	L	H
Even	Odd	Odd	Even	L	H	L
Even	Odd	Even	Odd	L	L	H
Even	Even	Odd	Odd	L	L	H
Even	Even	Even	Even	H	L	L

The logic for the other sections is the same except for the bits being sensed.

c) Total Parity. Total parity is the combined evaluation of the three sections. Refer to figure 6.2-4 and logic diagrams, figure 9-101.



NOTES: *INPUTS LABELED "AS" ARE FROM ARITHMETIC SELECT.
"EVEN 1,0" MEANS EVEN PARITY IN 2^1 AND 2^0 OF ARITHMETIC SELECT.

Figure 6.2-3. Parity Evaluator, Section 5-0

The gates shown in figure 6.2-4 test the parity conditions of the three groups for the four possible odd parity configurations. Refer to table 6.2-4 for the eight possible parity configurations.

TABLE 6.2-4. PARITY EVALUATOR, TOTAL PARITY CONDITIONS

SECTIONS			TOTAL	OUTPUT LEVELS				
17-12	11-6	5-0	PARITY	19X00	19X04	19X08	19X12	20X00
Odd	Odd	Odd	Odd	H	L	L	L	L
Odd	Odd	Even	Even	L	L	L	L	H
Odd	Even	Odd	Even	L	L	L	L	H
Odd	Even	Even	Odd	L	L	L	H	L
Even	Odd	Odd	Even	L	L	L	L	H
Even	Odd	Even	Odd	L	L	H	L	L
Even	Even	Odd	Odd	L	H	L	L	L
Even	Even	Even	Even	L	L	L	L	H

6.2-5. SUMMARY

The output of 20X00 is the total parity indication of arithmetic-select (AU · AL). The Parity flip-flop is cleared at T1.3 time of the I-sequence and is set at T1.4 time if total parity is odd. The state of this flip-flop only affects the execution of the f = 50:54, 50:55 instructions to condition the program skip.

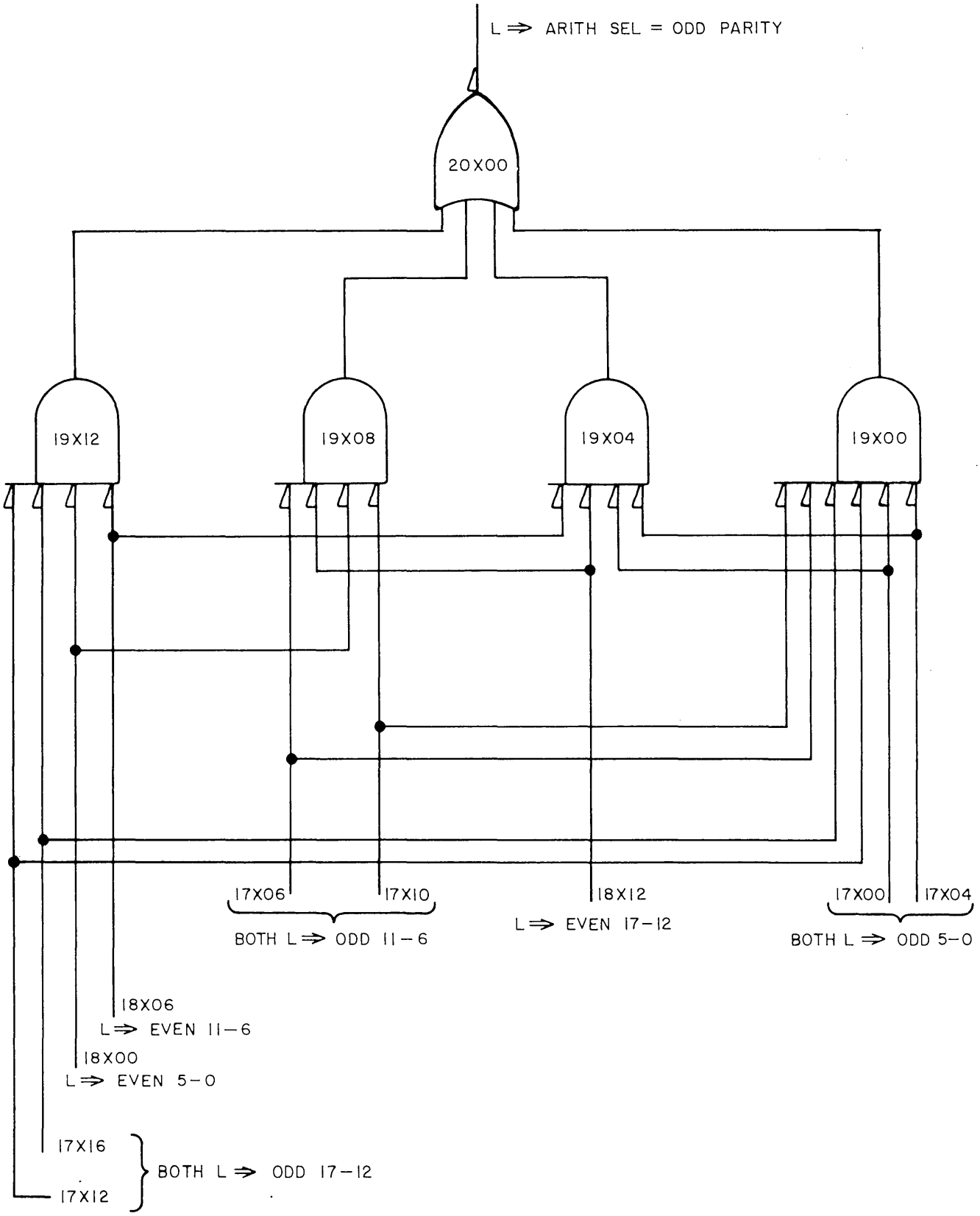


Figure 6.2-4. Parity Evaluator, Final Stage

NAME: _____

6.2-6. STUDY QUESTIONS

a. Given: AU = 461367
AL = 763752
instruction = 505400

Refer to logic diagrams, figure 9-101. Give the output logic levels for the following gates which exist at T1.4 time during the I-sequence of the above instruction.

15X00 = _____ 18X06 = _____

15X02 = _____ 17X12 = _____

15X04 = _____ 17X16 = _____

17X00 = _____ 18X12 = _____

17X04 = _____ 19X00 = _____

18X00 = _____ 19X04 = _____

17X06 = _____ 19X08 = _____

17X10 = _____ 19X12 = _____

20X00 = _____

b. Given conditions same as above.

When the instruction is executed, a program skip is performed. Refer to logic diagrams, figure 9-101. Determine whether each malfunction, occurring individually, would cause the erroneous program skip. Indicate your answers by writing "yes" or "no" beside each malfunction condition.

Grounded Output	Constant Low Level Output
_____ 15X00	_____ 15X00
_____ 16X02	_____ 16X02
_____ 16X04	_____ 16X04
_____ 16X00	_____ 16X00
_____ 17X04	_____ 17X04
_____ 17X10	_____ 17X10
_____ 18X12	_____ 18X12
_____ 19X00	_____ 19X00

SECTION 6 - ARITHMETIC SECTION

6.3. INSTRUCTION EXECUTION OF RSHAU, RSHAL, RSHA, LSHAU, LSHAL, LSHA

6.3-1. OBJECTIVES

To present the detailed theory of operation involved in the execution of instructions with f = 50:41-43, 50:45-47.

6.3-2. INTRODUCTION

These instructions perform either right shifts or left shifts of the content of AU, AL, or AU and AL together.

6.3-3. REFERENCES

a. UNIVAC 1219 Technical Manual, Volume I, Paragraphs 4-4c(2) and 4-7, table 4-11.

b. UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.3-4. INFORMATION

a. General Description.

1. Instruction Interpretation.

a) RSHAU, f = 50:41. This instruction right shifts with sign extension, the content of AU, the number of places specified by the six least significant bits (k) of the instruction word. The AU bit positions vacated are filled with the sign bit of the original value. Those bits shifted beyond AU_{17} are lost.

b) RSHAL, f = 50:42. Except for the value shifted, this instruction is the same as f = 50:41. The content of AL is right shifted with sign extension.

c) RSHA, f = 50:43. Except for the value shifted, this instruction is the same as f = 50:41. The combined content of AU and AL is right shifted with sign extension. AU is the more significant value. Those bits shifted beyond AU_{17} enter the left end of AL. The AU bits positions vacated are filled with the sign bit of the original AU value. Those bits shifted beyond AL_{00} are lost.

d) LSHAU, f = 50:45. This instruction circularly left shifts the content of AU the number of places specified by the six least significant bits (k) of the instruction word. Those bits shifted beyond AU_{35} enter the right end of AU.

e) LSHAL, f = 50:46. Except for the value shifted, this instruction is the same as f = 50:45. The content of AL is circularly left shifted. These bits shifted beyond AL_{17} enter the right end of AL.

f) LSHA, f = 50:47. Except for the value shifted, this instruction is the same as f = 50:45. The combined content of AU and AL is circularly left shifted. Those bits shifted beyond AL_{17} enter the right end of AU. Those bits shifted beyond AU_{35} enter the right end of AL.

2. Execution Sequences.

a) I-Sequence. During the I-sequence which obtains the instruction from memory, the shift count (k) is placed in KO.

b) Shift Sequence. The shift sequence uses a special timing which runs parallel to main timing and causes the shifting according to the shift count in KO.

b. Detailed Analysis.

1. I-Sequence. Most of the I-sequence operations are as previously described. If necessary, refer to study guide sheet number 5.4 for a detailed description.

KO receives the six least significant bits of the instruction word from Z-select. This value is the shift count. The function code in F determines the type of shift. The actual shifting is performed by the shift sequence which is initiated during the I-sequence. A detailed analysis of the shift sequence operations is presented later in this sheet.

2. Effect of Hold Flip-Flops. As is developed later in this sheet, the Hold I and Hold 2 flip-flops are set during the shifting operation. These flip-flops keep the machine in the I-sequence and prevent the reading of the next instruction until the shift operation is completed. Refer to logic diagrams, figure 9-28 for the Hold 1 and Hold 2 flip-flops.

These flip-flops both have outputs to the logic on figure 9-14. This logic supplies timing and enables for the I-sequence operations necessary to obtain next instruction. When the Hold flip-flops are set, their high level outputs disable these I-sequence operations. Among the events which are prevented are initiate-memory, advance-P subsequence, clear-F, Z-select F, clear-KO, and Z-select KO. Refer to the proper enable pages in the logic diagrams to develop the disable function of these events.

As long as the Hold flip-flops are set, the shift function code and the remaining shift count are retained in F and KO, respectively.

During the execution of a shift instruction, the F register indicates a shift function code which does not require any main timing sequence other than the I-sequence. Therefore, as the shifts are being performed, the I-sequence flip-flop is set for each main timing cycle. Refer to logic diagrams, figure 9-12 to develop the setting of the I-sequence flip-flop for each main timing cycle as long as F specifies f = 50:41-43, 50:45-47.

3. Shift Sequence.

a) Timing.

b) Shifting Operations.

1) Essential Commands. The commands which shift AU and AL by means of X and W, respectively, are enabled by the 0XL00 and 0XL01 Shift Sequence flip-flops and are timed by the master clock phases. Refer to table 6.3-4 for a sequential list of essential shift sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

TABLE 6.3-1. I AND SHIFT SEQUENCE ESSENTIAL COMMANDS WITH INITIAL SHIFT COUNT = 0

TIME NOTATION	COMMANDS
	I SEQUENCE
T1.4	Clear K0
T2.3	Clear 0XL00 ff
T2.4	Z $Sel_{5-0} \rightarrow$ K0 (K0 = 0)
T3.4	Set Hold 1 ff, clear Scale Factor ff (1XL00)
T4.1	Set 0XL01 ff
T4.3	Set 0XL02 ff, set Hold 2 ff
T1.1	Set Clear Hold ff, clear 0XL01 ff
T1.3	Clear 0XL02 ff
T4.2	Clear Hold 1 ff, clear Clear Hold ff
	I-SEQUENCE FOR NEXT INSTRUCTION
T1.3	Clear Hold 2 ff

2) Data Flow Block Diagram. Refer to figure 6.3-2 for a block diagram description of the shift sequence operations.

These diagrams illustrate the data flow for a one place shift for each of the shift function codes. One such flow of data can occur per master clock cycle. This data flow is continually repeated until the Shift Sequence flip-flops are cleared at the termination of the shift count.

TABLE 6.3-2. I AND SHIFT SEQUENCE ESSENTIAL COMMANDS WITH INITIAL SHIFT COUNT ≠ 0

TIME NOTATION	COMMANDS
	I SEQUENCE
T1.4	Clear K0
T2.3	Clear 0XL00 ff
T2.4	Z Sel_{5-0} → K0
T3.4	Set Hold 1 ff, clear Scale Factor ff (1XL00) - HOLD SEQUENCER UNTIL SHIFT IS COMPLETED
T4.1	Set 0XL01 ff
T4.2	Clear K1
T4.3	Set 0XL00 ff, set Hold 2 ff, K0-1 → K1, *clear X, *clear W
T4.4	Clear K0, *shift A → X/W
T1.1	K1 → K0, *clear A
T1.2	Clear K1, *X/W → A
T1.3	K0-1 → K1, *clear X, *clear W
T1.4	Clear K0, *shift A → X/W
T2.1	K1 → K0, *clear A
T2.2	Clear K1, *X/W → A
	(continue until K1 = 0)
TX.3	K0-1 → K1 (K1 = 0), *clear X, *clear W
TX.4	Clear K0, *shift A → X/W
TX.1	K1 → K0 (K0 = 0), set Clear-Hold ff, *clear A
TX.2	Clear K1, *X/W → A (A_f)
TX.3	K0-1 → K1, clear 0XL00 ff, set 0XL02 ff, *clear X, *clear W
TX.4	*Shift A → X/W (not used)
TX.1	Clear 0XL01 ff
TX.3	Clear 0XL02 ff
next T4.2	Clear Hold 1 ff, clear Clear-Hold ff
	I SEQUENCE FOR NEXT INSTRUCTION
T1.3	Clear Hold 2 ff

*These events pertain to the actual shifting of AU and AL and are discussed later in this sheet.

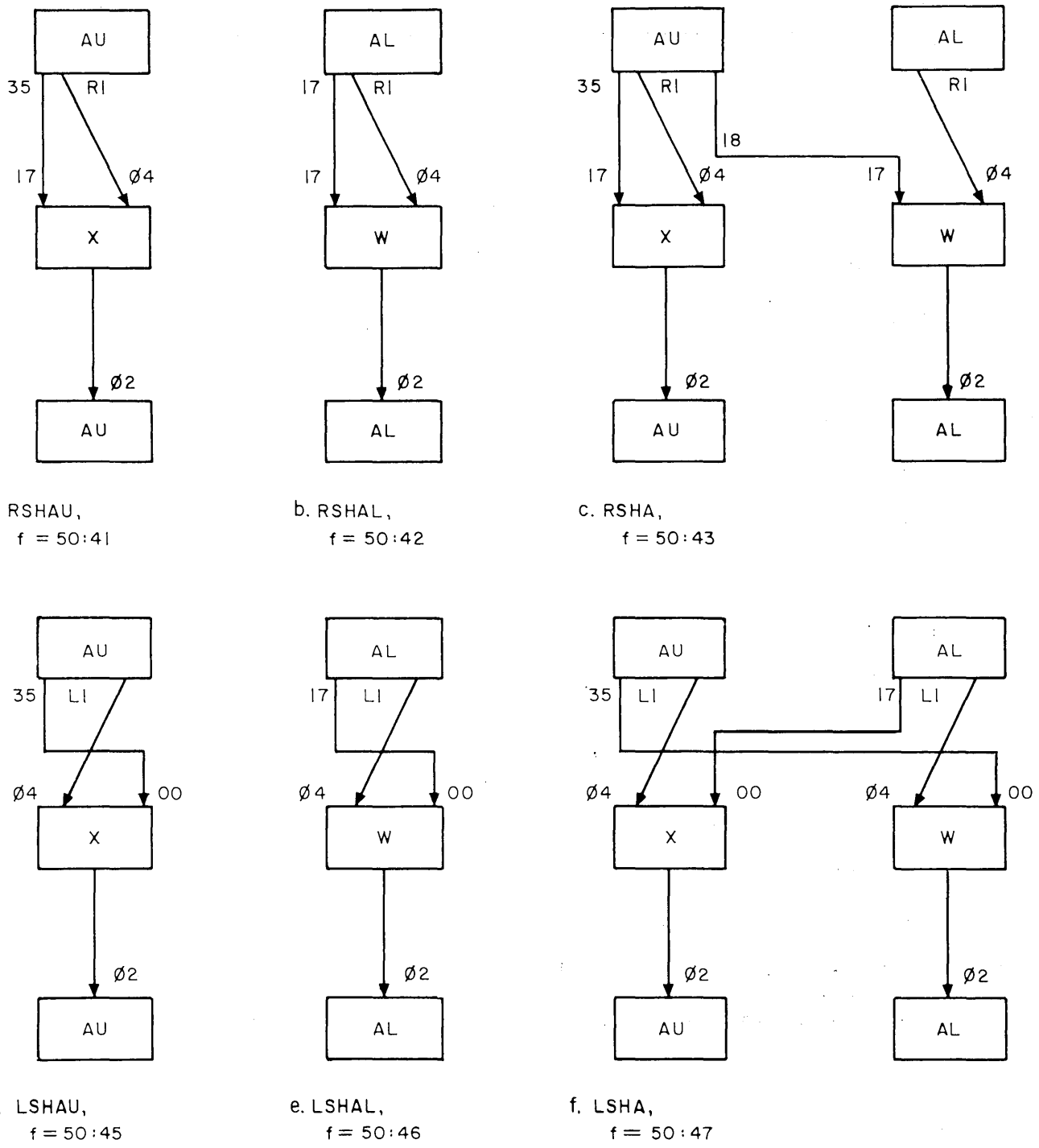


Figure 6.3-2. Shift Sequence Data Flow, One Place Shift

TABLE 6.3-3. EXECUTION TIME OF f = 50:41-43, 50:45-47

$SHIFT\ COUNT_{10}$	NUMBER OF MAIN TMG. CYCLES (I SEQ.)	EXECUTION TIME (μs)
0 through 4	2	4
5 through 8	3	6
9 through 12	4	8
13 through 16	5	10
17 through 20	6	12
21 through 24	7	14
25 through 28	8	16
29 through 32	9	18
33 through 36	10	20
37 through 40	11	22
41 through 44	12	24
45 through 48	13	28
49 through 52	14	30
53 through 56	15	32
57 through 60	16	34
61 through 63	17	36

6.3-5. SUMMARY

The f = 50:41 — 50:43, 50:45 — 50:47 instructions are format 2 and use the value k. The k value is available in K0 after T2.4 time of the I-sequence. The shift sequence is required to complete the executions of these instructions.

TABLE 6.3-4. SHIFT SEQUENCE ESSENTIAL COMMANDS

TIME	f=50:	41	42	43	45	46	47
0XL01 ff set·Ø3	Clear X, clear W	X	X	X	X	X	X
0XL01 ff set·Ø4	AUR1→X	X	X	X			
	*Set X_{17}=1 if AU_{35}=1 (AU_{35}→X_{17})	X	X	X			
	ALR1→W	X	X	X			
	*Set W_{17}=1 if AU_{18}=1 (AU_{18}→W_{17})	X		X			
	*Set W_{17}=1 if AL_{17}=1 (AL_{17}→W_{17})		X				
	AUL1→X				X	X	X
	*Set X_{00}=1 if AU_{35}=1 (AU_{35}→X_{00})				X		
	*Set X_{00}=1 if AL_{17}=1 (AL_{17}→X_{00})					X	X
	ALL1→W				X	X	X
	*Set W_{00}=1 if AU_{35}=1 (AU_{35}→W_{00})				X		X
	*Set W_{00}=1 if AL_{17}=1 (AL_{17}→W_{00})					X	
0XL00 ff set·Ø1	Clear AU	X		X	X		X
	Clear AL		X	X		X	X
0XL00 ff set·Ø2	X→AU	X		X	X		X
	**W→AL		X	X		X	X

*These commands are enabled by gates 31W00, 31W17, 31X00, and 31X17 in the logic diagrams, figure 9-33, and are timed by the AUR1→X, ARL1→W, AUL1→X, and ALL1→W commands.

**The transmission of bit 00 for the W→AL command is through gate 83A00 in the logic diagrams, figure 9-33.

NAME: _____

6.3-6. STUDY QUESTIONS

a. Given: 31X17 grounded output (logic diagrams, figure 9-33)
instruction = 504303
AU_i = 430012
AL_i = 234756

1. For the given conditions, draw below the block diagram of data flow for a 1-place shift like those in figure 6.3-2 of this sheet.

2. What are the contents of AU and AL at the completion of the given instruction considering the malfunction?

AU_f = _____ AL_f = _____

b. Given: 10F42 grounded output (logic diagrams, figure 9-47)
instruction = 504303
AU_i = 430012
AL_i = 534756

1. For the given conditions, draw below the block diagram of data flow for a 1-place shift like those in figure 6.3-2 of this sheet.

2. What are the contents of AU and AL at the completion of the given instruction considering the malfunction?

AU_f = _____ AL_f = _____

NAME: _____

c. Given: 11F45 grounded output (logic diagrams, figure 9-47)
instruction = 504503
AU_i = 430012

1. For the given conditions, draw below the block diagram of data flow for a 1-place shift like those in figure 6.3-2 of this sheet.

2. What is the content of AU at the completion of the given instruction considering the malfunction?

AU_f = _____

SECTION 6 - ARITHMETIC SECTION

6.4. KO-1 ADDER

6.4-1. OBJECTIVES

To present the detailed theory of operation involved in the KO-1 adder.

6.4-2. INTRODUCTION

The KO-1 adder is used to control the number of repeated operations used in shifting, scaling, multiplication, and division.

6.4-3. REFERENCES

a. UNIVAC 1219 Technical Manual, Volume I, Paragraph 4-2e(3).

b. UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.4-4. INFORMATION

a. General Description. The KO-1 adder is a 6-bit open-ended adder which continually subtracts 1 from the content of the KO register. The output of the adder is taken to the K1 register. The decremented value is then taken back to KO. Refer to figure 6.4-1 for a data flow block diagram of the K-counter configuration.

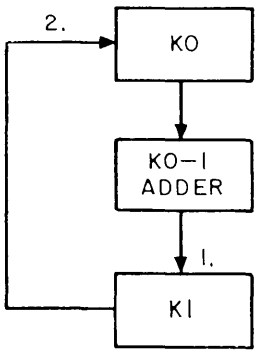


Figure 6.4-1. K Counter Block Diagram

Certain repeating type operations are controlled by the counter. KO is initially set to a specific count. Each time KO receives the new decremented value, one of the repeating operations is performed. When KO holds the value of all O's, the controlled operations are terminated.

b. Detailed Analysis

1. Effect of Borrow Request. If a request for a borrow is applied to a bit position of KO, the result of this subtraction produces the complement of the bit. Refer to figure 6.4-2 for examples.

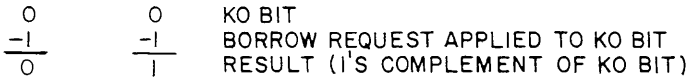


Figure 6.4-2. KO-1 Adder Borrow Examples

2. Adder Stage 00. Refer to logic diagrams, figure 9-37 for the adder logic.

The subtraction of KO-1 always affects KO_{00} to produce its complement. This complemented value is outputted by inverter 02K00 and is placed in $K1_{00}$ during the "KO-1 → K1" command.

3. Generation of Borrow Requests. If a borrow request is applied to a particular bit position, the complement of that bit is the result. If that same KO bit position contains 0_2, the borrow request is propagated to the next higher bit because the 0_2 cannot supply the 1_2 that the borrow requires. All borrow requests originate from the subtraction of KO_{00} - 1. A borrow request is applied to a particular bit position if all of the less significant KO bit positions contain O's. Refer to table 6.4-1 for the conditions necessary to apply borrow requests to the KO bits.

TABLE 6.4-1. KO-1 ADDER BORROW REQUEST CONDITIONS

	KO bit positions				
Borrows Applied	04	03	02	01	00
Brw → 01					0_2
Brw → 02				0_2	0_2
Brw → 03			0_2	0_2	0_2
Brw → 04		0_2	0_2	0_2	0_2
Brw → 05	0_2	0_2	0_2	0_2	0_2

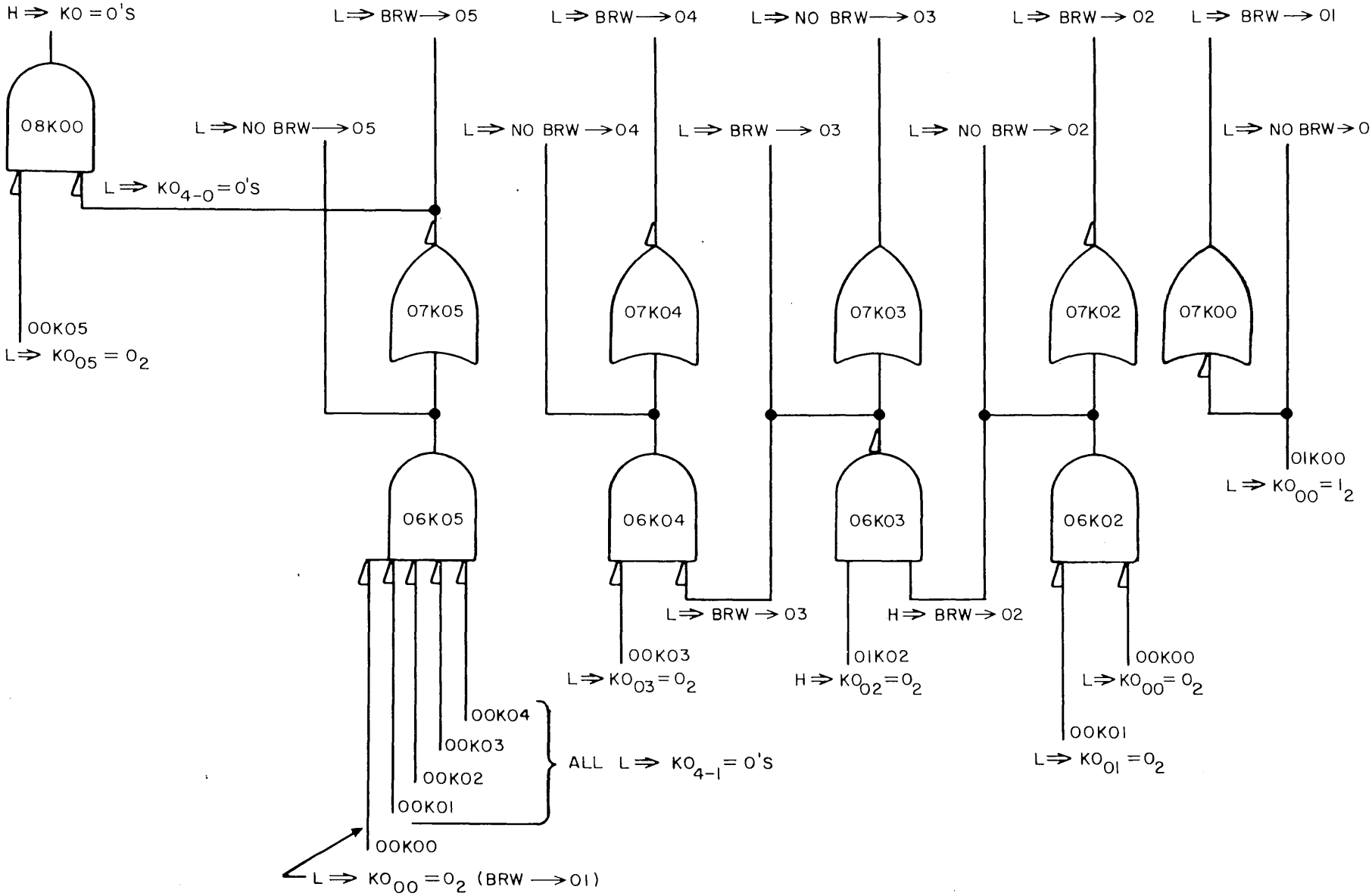
NOTE: "Brw → XX" means a borrow request is being applied to bit position XX.

Adder bits 05 through 01 have logic which tests all of the less significant K0 bits for the binary configurations described above. If a borrow request is applied, the complement of the K0 bit is the adder output as was shown for bit 00. Refer to figure 6.4-3 for the adder request logic. This is a portion of that shown in the logic diagrams, figure 9-37.

Gate 08K00 is used to test K0 for all 0's. This condition terminates the operation being controlled by the K-counter.

6.4-5. SUMMARY

The K-counter is comprised of K0 and K1 registers and the K0-1 adder. K0 is hard-wired to the adder input. The adder is 6 bits in length, and is of the open-ended type. It can only output to K1.



NOTES: "BRW → 0X" MEANS A BORROW REQUEST IS BEING APPLIED TO BIT POSITION 0X.
GATE 08K00 IS NOT ACTUALLY PART OF THE REQUEST LOGIC.

Figure 6.4-3. KO-1 Adder Borrow Request Generation Logic

1) Initial Shift Count = 0. If the shift count specified in the six least significant bits of the instruction word equals 0, no shifts are to be performed. The shift sequence is initiated but is terminated before it can effect any shifting.

Refer to table 6.3-1 for a sequential list of essential I and shift sequence control events. Develop these commands by referring to the proper enable pages in the logic diagrams. The commands shown are in addition to the normal I-sequence commands.

As shown, no shifting is performed. Two main timing cycles of the I-sequence are used before the Hold flip-flops are cleared. With the initial shift count equal to 0, the instruction execution time is 4 microseconds.

2) Initial Shift Count ≠ 0. If the shift count specified in the six least significant bits of the instruction word does not equal 0, one or more shifts are performed. The shift sequence is initiated and remains active until the shifting is completed at which time it is terminated.

The shift commands involving AU and AL are generated by the combination of the shift sequence flip-flops (0XL00 and 0XL01) being set and the occurrence of clock phases. 0XL00 and 0XL01 are set for the entire shifting operation. The clock phases alone provide the timing for the operation. For each cycle of the master clock, a shift of one place is executed.

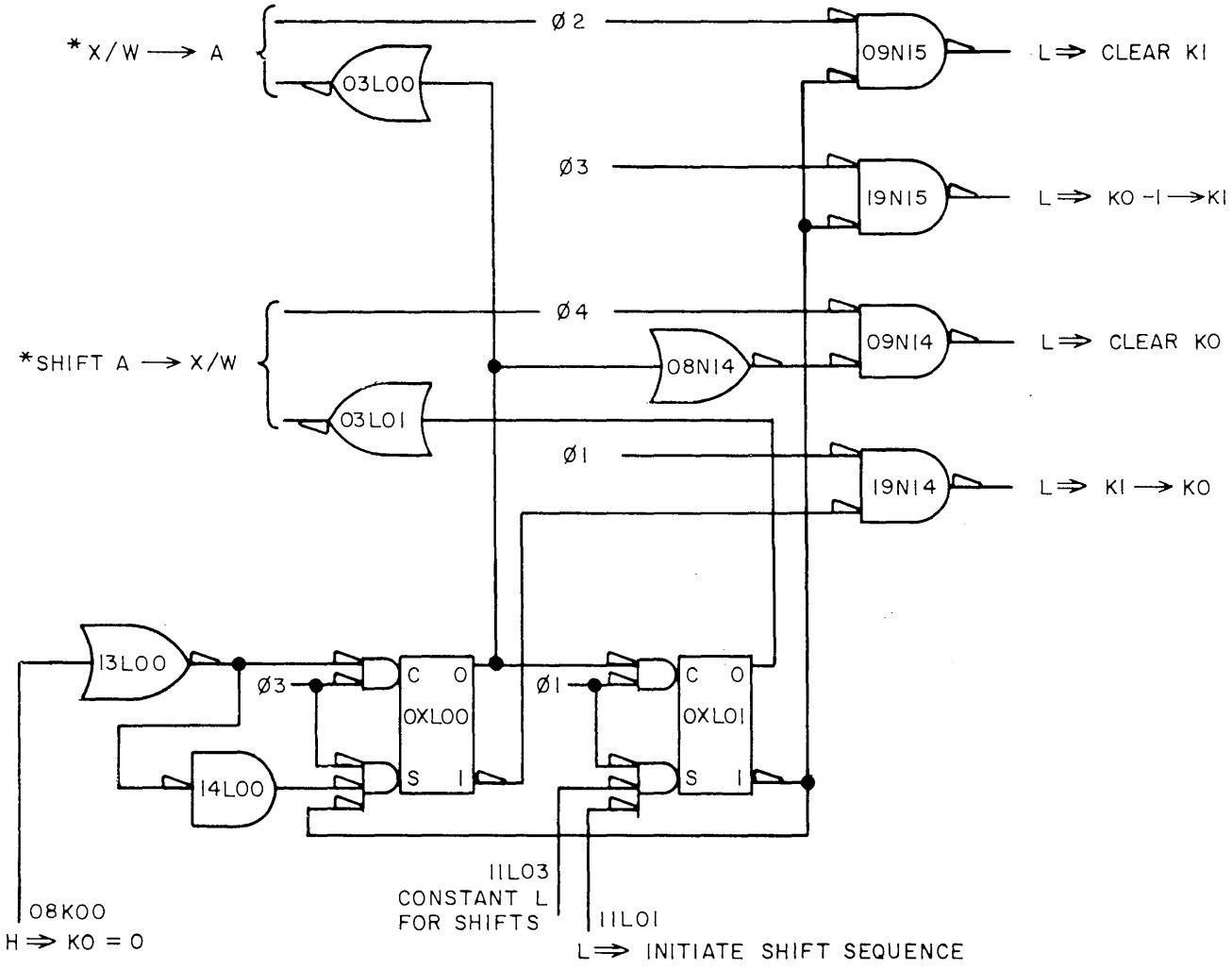
K0, K1, and K0-1 adder are used to control the number of shifts by determining the number of clock cycles during which the shift sequence flip-flops will remain set. The shift count which is held in K0 and K1 is decremented by 1 during each master clock cycle. When K0 reaches the count of 0, the Shift Sequence flip-flops are cleared and the AU and AL shift commands are disabled. The K0-1 adder is analyzed in a later sheet.

Refer to figure 6.3-1 for a simplified logic diagram of the shift sequence. Develop the events chart from the logic shown.

Refer to table 6.3-2 for a sequential list of essential I and shift sequence control events. Develop these commands by referring to the proper enable pages in the logic diagrams. The commands shown are in addition to the normal sequence commands.

As shown, shifting is performed until K1 = 0 which causes the clearing of the 0XL00 and 0XL01 flip-flops after the last shift. Flip-flop 0XL02 is set and cleared at the termination but is not used. The Hold 1 flip-flop is not cleared until T4.2 time of the main timing cycle during which the last shift occurred. The next main timing cycle is also under I-sequence control but is able to read-up the next instruction in the program. The last Hold flip-flop (2) is cleared at T1.3 time of the next instruction's I-sequence.

With the shift sequence active, each master clock cycle performs a one-place shift. Therefore, a maximum of four shifts can be performed during one main timing cycle which has a duration of two microseconds. Including the two microsecond I-sequence which reads-up the shift instruction, a shift instruction with a shift count from 1 through 4 has an execution time of four microseconds. Refer to table 6.3-3 for a complete list of shift count values with the corresponding instruction execution times.



SEQUENCE OF EVENTS ($KO_i \neq 0$)

Ø1	SET OXL01 FF (INITIATE SHIFT SEQUENCE)
Ø2	CLEAR KI
Ø3	SET OXL00 FF, KO − 1 → KI, *CLEAR X & W
Ø4	CLEAR KO, *SHIFT A → X/W
Ø1	KI → KO, *CLEAR A
Ø2	CLEAR KI, *X/W → A
¦	(CONTINUE UNTIL KI = 0)
Ø3	KO−1 → KI (KI = 0), *CLEAR X & W
Ø4	CLEAR KO, *SHIFT A → X/W
Ø1	KI → KO (KO = 0), *CLEAR A
Ø2	CLEAR KI (NOT USED), *X/W → A (A_f)
Ø3	CLEAR OXL00 FF, KO−1 → KI (NOT USED), *CLEAR X & W (NOT USED)
Ø4	*SHIFT A → X/W (NOT USED)
Ø1	CLEAR OXL01 FF

NOTE: *THESE EVENTS PERTAIN TO THE ACTUAL SHIFTING OF AU AND AL AND ARE DISCUSSED LATER IN THIS SHEET.

Figure 6.3-1. Shift Sequence Simplified Logic

NAME: _____

6.4-6. STUDY QUESTIONS

a. Given: instruction - 504605
06K03 grounded output (logic diagrams, figure 9-37)

Considering the above malfunction, describe the effect upon the execution of the given instruction. Fully explain your reasoning.

SECTION 6 - ARITHMETIC SECTION

6.5. INSTRUCTION EXECUTION OF SF

6.5-1. OBJECTIVES

To present the detailed theory of operation involved in the execution of the instruction with f = 50:44.

6.5-2. INTRODUCTION

This instruction normalizes the combined contents of AU and AL.

6.5-3. REFERENCES

a. UNIVAC 1219 Technical Manual, Volume I, Paragraphs 4-4c(4) and 4-7, tables 4-11 and 4-14.

b. UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.5-4. INFORMATION

a. General Description

1. Instruction Interpretation. This instruction, SR, circularly left shifts the combined content of AU and AL. AU is the more significant portion. Left shifts are performed until $AU_{35} \neq AU_{34}$ or until the maximum shift count specified in the six least significant bits of the instruction word has expired. This shift count dictates the maximum number of places that AU and AL can be shifted during the normalize operation. When shifting stops, the difference between the specified maximum shift count and the actual shift count is stored in control memory at the address 00017_8. The original content of this memory address is destroyed.

2. Execution Sequences.

a) I-Sequence. During the I-sequence which obtains the instruction from memory, the maximum shift count is placed in KO.

b) Scale Sequence. The scale sequence uses a special timing chain which runs in parallel to main timing and controls the actual normalize operation.

c) W-Sequence. The W-sequence is active throughout the normalize operation but is only effective at the completion of the operation in storing the difference between the maximum shift count specified and the actual shift count.

b. Detailed Analysis.

1. I-Sequence. The I-sequence operations are as previously described. If necessary, refer to study guide sheet number 5.4 for a detailed description. At the end of the I-sequence, KO contains the six least significant bits of the instruction word which specifies the maximum shift count allowed.

2. Effect of Hold Flip-Flops. The Hold 1 and Hold 2 flip-flops are set during the scale operation to prevent the normal W-sequence operations which store KO in control memory. These flip-flops prevent the setting of the final W-sequence flip-flop (lower bank). When the Hold flip-flops are cleared, the W-sequence is allowed to store KO.

3. W and Scale Sequences.

a) Data Flow Block Diagram.

1) Prior to Scale Termination. Refer to figure 6.5-1 for a block diagram description of one step of the scale operation.

During each master clock cycle with the scale sequence active, AU and AL are circularly left shifted one place as a 36 bit register. The count in KO is decremented by 1. When the scale sequence detects $AU_{35} \neq AU_{34}$ or KO = 0, the shift operation is terminated.

2) Scale Termination (KO Storage). Refer to figure 6.5-2 for a block diagram description of the KO storage operations.

When the scale sequence is terminated, the W-sequence stores KO in control memory at the address 00017_8. KO still contains its value which existed at the scale termination. This value is the difference between the maximum shift count allowed (KO_i) and the actual number of shifts executed by the scale sequence.

b) Essential Commands.

1) Aborted Scale Sequence. If the maximum shift count specified by the instruction equals 0 or if the initial value in AU is such that $AU35 \neq AU_{34}$, no shifting of AU and AL is to be performed. The scale sequence is disabled.

Refer to tables 6.5-1 and 6.5-2 for sequential lists of essential I, W, and scale sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

2) Normal Scale Sequence Prior to Termination. Refer to table 6.5-3 for a sequential list of essential I, W, and scale sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

The scale sequence commands which left shift AU and AL and decrement KO occur continuously until the scale sequence flip-flops 0XL00 and 0XL01 are cleared.

3) Scale Sequence Termination. Termination operations are initiated when either $AU_{34} \neq AU_{33}$ or K1 = 0. Refer to tables 6.5-4 and 6.5-5 for sequential lists of W and scale sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

4) W-Sequence Storage of KO. At the termination or abortion of the scale sequence, the Hold 1 and Hold 2 flip-flops are cleared which allows the W-sequence to perform the storage of KO in control memory at the address 00017_8. Refer to table 6.5-6 for a sequential list of essential W-sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

6.5-5. SUMMARY

The SF instruction is format 2 and uses the value k. The k value is available in KO after T2.4 time of the I-sequence. The scale and W-sequences are required to complete the execution of this instruction.

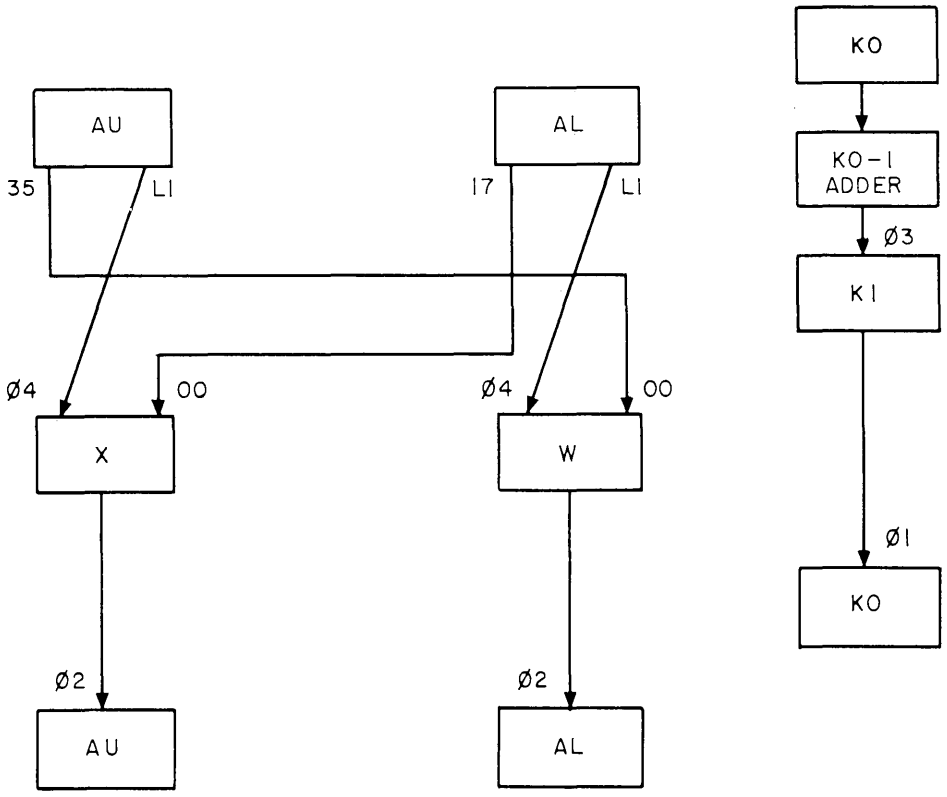
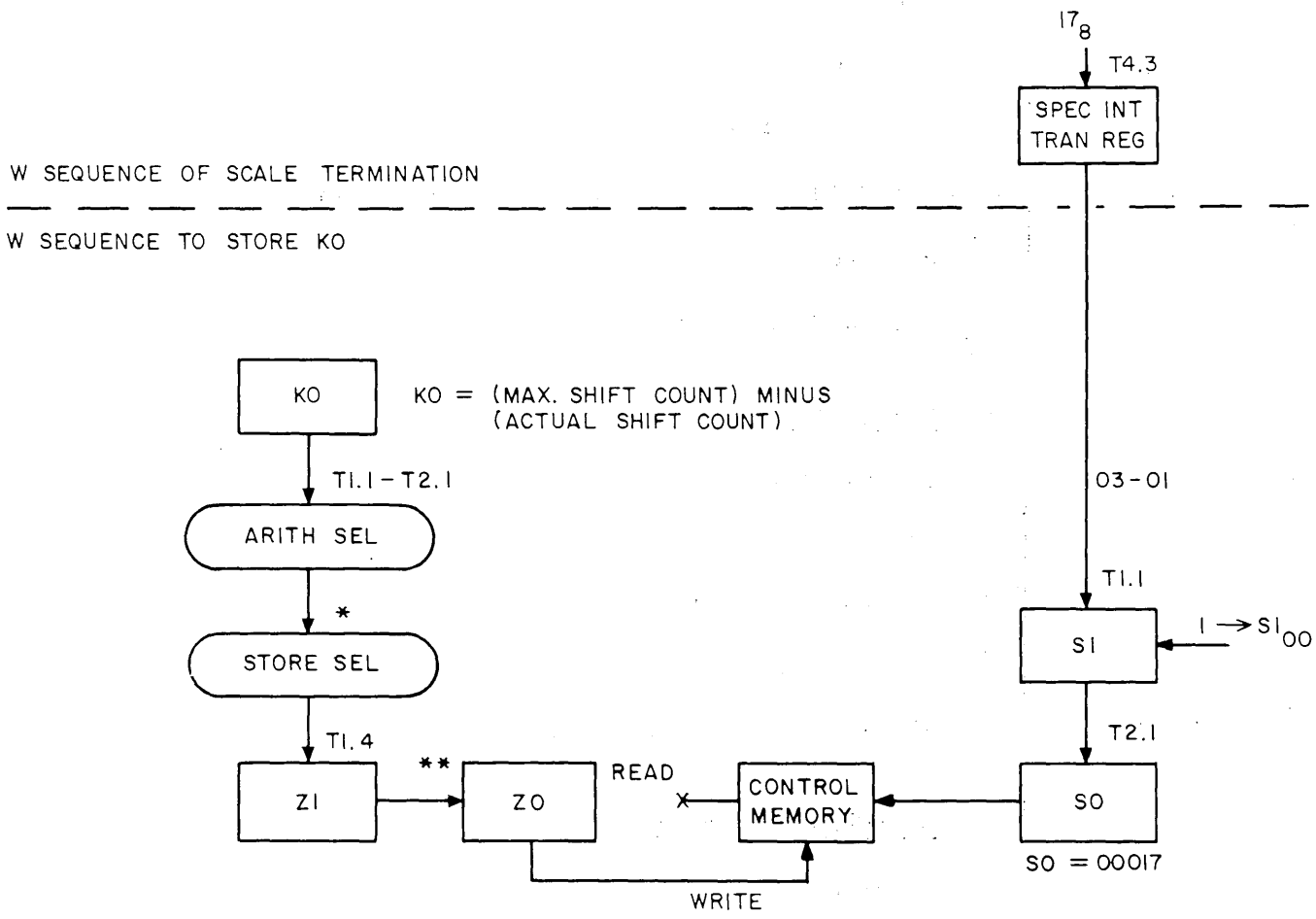


Figure 6.5-1. Scale Sequence, One Place Shift



NOTES: *ARITH SEL ⟶ STORE SEL OCCURS DURING THE ENTIRE W SEQUENCE.

**Z1 ⟶ Z0 IS TIMED BY CONTROL MEMORY TIMING.

Figure 6.5-2. Final Scale W and Last W Sequence Data Flow

TABLE 6.5-1. I, W, AND SCALE SEQUENCE ESSENTIAL COMMANDS WITH MAXIMUM SHIFT COUNT ALLOWED = 0*

TIME NOTATION	COMMANDS
	I SEQUENCE
T1.4	Clear KO
T2.3	Clear 0XL00 ff
T2.4	Z $Sel_{5-0} \rightarrow$ KO (KO = 0)
T3.4	Set Hold 1 ff, clear Scale Factor ff (1XL00)
T4.1	Set 0XL01 ff
T4.3	Set 0XL02 ff, set Hold 2 ff
	W SEQUENCE
T1.1	Set Clear Hold ff, clear 0XL01 ff
T1.3	Clear 0XL02 ff
T4.1	**Clear Spec Int Trans Reg.
T4.2	Clear Hold 1 ff, clear Clear-Hold ff
T4.3	**Set Spec Int Trans Reg. := 17_8
	W SEQUENCE TO STORE KO
T1.3	Clear Hold 2 ff

* The W-sequence events which store KO are not shown.

** These commands pertain to the events which store KO and are discussed later in this sheet.

TABLE 6.5-2. I, W, AND SCALE SEQUENCE ESSENTIAL COMMANDS WITH $AU_{35i} \neq AU_{34i}$

TIME NOTATION	COMMANDS
	I SEQUENCE
T1.3	Clear F
T1.4	Clear KO
T2.3	Clear OXL00 ff
T2.4	Z Sel_{11-6} → F, set OXF06 ff, Z Sel_{5-0} → KO
T3.1	Set Clear Hold ff
T3.4	Set Hold 1 ff, clear Scale Factor ff (1XL00)
T4.1	**Clear Spec Int Trans Reg
T4.2	Clear Hold 1 ff, clear Clear-Hold ff
T4.3	**Set Spec Int Trans Reg = 17_8
	W SEQUENCE TO STORE KO
T1.3	Clear Hold 2 ff

*The W-sequence events which store KO are not shown.

**These commands pertain to the events which store KO and are discussed later in this sheet.

TABLE 6.5-3. I, W, AND SCALE SEQUENCE ESSENTIAL COMMANDS WITH MAXIMUM SHIFT COUNT ALLOWED $\neq$ 0 AND $AU_{35i} = AU_{34i}$

TIME NOTATION	COMMANDS
	I SEQUENCE
T1.4	Clear KO
T2.3	Clear 0XL00 ff
T2.4	Z $Sel_{5-0} \rightarrow$ KO
T3.4	Set Hold 1 ff, clear Scale Factor ff (1XL00)
T4.1	Set 0XL01 ff
T4.2	Clear K1
T4.3	Set 0XL00 ff, set Hold 2 ff, KO-1 $\rightarrow$ K1, clear X, clear W
T4.4	Clear KO, AUL1 $\rightarrow$ X, ALL1 $\rightarrow$ W, *$AU_{35} \rightarrow W_{00}$, *$AL_{17} \rightarrow X_{00}$
	FIRST W SEQUENCE
T1.1	K1 $\rightarrow$ KO, clear AU, clear AL
T1.2	Clear K1, X $\rightarrow$ AU, **W $\rightarrow$ AL
T1.3	KO-1 $\rightarrow$ K1, clear X, clear W
T1.4	Clear KO, AUL1 $\rightarrow$ X, ALL1 $\rightarrow$ W, *$AU_{35} \rightarrow W_{00}$, *$AL_{17} \rightarrow X_{00}$
T2.1	K1 $\rightarrow$ KO, clear AU, clear AL
T2.2	Clear K1, X $\rightarrow$ AU, **W $\rightarrow$ AL
(continue until K1 = 0 or $AU_{34} \neq AU_{33}$)	

*These commands are enabled by gate 31W00 and 31X00 in the logic diagrams, figure 9-33, and are timed by the AUL1 $\rightarrow$ X and ALL1 $\rightarrow$ W commands.

**The transmission of bit 00 for the W $\rightarrow$ AL commands is through gate 83A00 in the logic diagrams, figure 9-33.

TABLE 6.5-4. TERMINATION OF SCALE SEQUENCE BY K1 = 0(W SEQUENCE)*

TIME NOTATION	COMMANDS
	W SEQUENCE
TX.3	KO-1 → K1 (K1 = 0), Clear X, Clear W
TX.4	Clear KO, AUL1 → X, ALL1 → W, **AU_{35} → W_{00}, **AL_{17} → X_{00}
TX.1	K1 → KO (KO = 0), set Clear Hold ff, clear AU, clear AL
TX.2	Clear K1, X → AU, ***W → AL
TX.3	KO-1 → K1, clear 0XL00 ff, set 0XL02 ff, clear X, clear W
TX.4	AUL1 → X, ALL1 → W, **AU_{35} → W_{00}, **AL_{17} → X00 (not used)
TX.1	Clear 0XL01 ff
TX.3	Clear 0XL02 ff
next T4.2	Clear Hold 1 ff, clear Clear-Hold ff
	W SEQUENCE TO STORE KO
T1.3	Clear Hold 2 ff

*The W-sequence events which store KO are not shown.

**These commands are enabled by gates 31W00 and 31X00 in the logic diagrams, figure 9-33, and are timed by the AUL1 → X, and ALL1 → W commands.

***The transmission of bit 00 for the W → AL command is through gate 83A00 in the logic diagrams, figure 9-33.

TABLE 6.5-5. TERMINATION OF SCALE SEQUENCE BY $AU_{34} \neq AU_{33}$ (W SEQUENCE)*

TIME NOTATION	COMMANDS
	W SEQUENCE
TX.3	KO-1 → K1
TX.4	Clear KO, AUL1 → X, ALL1 → W, **AU_{35} → W_{00}, **AL_{17} → X_{00}
TX.1	K1 → KO, clear AU, clear AL
TX.2	Clear K1, X → AU ($AU_{34} \neq AU_{33}$), ***W → AL
TX.3	KO-1 → K1
TX.4	Set Scale Factor ff (1XL00)
TX.1	K1 → KO
TX.2	Clear K1, X → AU ($AU_{35} \neq AU_{34}$), ***W → AL
TX.3	KO-1 → K1, clear 0XL00 ff, set 0XL02 ff
TX.4	AUL1 → X, ALL1 → W, AU_{35} → W_{00}, **AL_{17} → X_{00} (not used)
TX.1	Clear 0XL01 ff, set Clear Hold ff
TX.3	Clear 0XL02 ff
next T4.2	Clear Hold 1 ff, clear Clear-Hold ff
	W SEQUENCE TO STORE KO
T1.3	Clear Hold 2 ff

*The W-sequence events which store KO are not shown.

**These commands are enabled by gates 31W00 and 31X00 in the logic diagrams, figure 9-33, and are timed by the AUL1 → X, and ALL1 → W commands.

***The transmission of bit 00 for the W → AL command is through gate 83A00 in the logic diagrams, figure 9-33.

TABLE 6.5-6. FINAL SCALE W AND LAST W SEQUENCE ESSENTIAL COMMANDS

TIME NOTATION	COMMANDS
	W SEQUENCE OF SCALE TERMINATION
T4.1	Clear special interrupt translator Reg
T4.2	Clear Hold 1 ff
T4.3	Set Spec Int Trans Reg = 17_8
T4.4	Clear S1
	W SEQUENCE TO STORE K0
T1.1	Spec Int Trans $Reg_{03-01} \rightarrow S1_{03-01}$, $1 \rightarrow S1_{00}$, K0 $\rightarrow$ Arith Sel*
T1.3	Clear Hold 2 ff, clear Z1
T1.4	Disable CM $\rightarrow$ Z0, Store Sel $\rightarrow$ Z1**
T2.1	S1 $\rightarrow$ S0, drop K0 $\rightarrow$ Arith Sel
T2.4	Drop disable CM $\rightarrow$ Z0

*Arith Sel $\rightarrow$ Store Sel occurs during the entire W-sequence.

**Z1 $\rightarrow$ Z0 is timed by control memory timing.

SECTION 6 - ARITHMETIC SECTION

6.6. INSTRUCTION EXECUTION OF MULAL, MULALB

6.6-1. OBJECTIVES

To present the detailed theory of operation involved in the execution of instructions with f = 24, 25.

6.6-2. INTRODUCTION

These instructions multiply the content of AL by the content of memory.

6.6-3. REFERENCES

a. UNIVAC 1219 Technical Manual, Volume I, Paragraphs 4-4c(1)(c) and 4-7, tables 4-11 and 4-12.

b. UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.6-4. INFORMATION

a. General Description.

1. Instruction Interpretation.

a) MULAL, f = 24. This instruction multiplies the content of AL by the operand Y. Y is obtained from memory at the address U_P if SR is inactive or U_{SR} if SR is active. The final product is double length and appears in AU and AL. AU contains the more significant bits.

b) MULALB, f = 25. Except for the address of Y, this instruction is the same as f = 24. The address of Y is either $U_P + B$ or $U_{SR} + B$, depending upon the activeness of SR. The B register is specified by ICR.

2. Execution Sequences.

a) I-Sequence. During the I-sequence which obtains the instruction from memory, the address of the operand is formulated from U, P, SR, and B.

b) R1-Sequence. The R1-sequence uses a memory reference to obtain the operand Y.

c) Multiply Sequence. The multiply sequence uses a special timing chain which runs in parallel to main timing and controls the actual multiplication.

3. Multiplication Procedure.

a) Pencil and Paper Method. Multiplication with the binary number system is quite simple since during each multiplication step the multiplicand is multiplied by either 1_2 or 0_2. The multiplication by 1_2 is performed by adding the multiplicand to the partial product. Multiplying by 0_2 simply adds 0 to the partial product. Refer to figure 6.6-1 for a 4-bit example of the normal "pencil and paper" method.

```
  0101    MULTIPLICAND
x 0011    MULTIPLIER
  0101
 0101     } PARTIAL PRODUCTS
0000
0000
-------
0001111   PRODUCT
```

Figure 6.6-1. Example of Binary Multiplication, Pencil and Paper Method

b) 1219 Computer Method. In the 1219, the procedure is basically the same as described above. However, the result of each multiplication step is added to the previous partial product immediately instead of adding all of the partial products together at the end.

Also, as each new partial product is formulated, it is shifted right one place. Refer to figure 6.6-2 for the same numerical example using the 1219 Computer method.

In each step, either the multiplicand Y or +0 is added to the partial product in AU depending upon the value of AL_{00}. If $AL_{00} = 1_2$, it specifies 1 x Y which is accomplished by adding Y to the previous partial product.

As the process continues, the multiplier is shifted out of AL and the lower half of the product is shifted into AL. The final product is the content of AU and AL together.

b. Detailed Analysis.

1. I-Sequence. Most of the I-sequence operations are as previously described. If necessary, refer to study guide sheet number 5.4 for a detailed description. At the end of the I-sequence, the X-D' adder is outputting the address of the operand.

In addition to the normal operations, the Y Neg and A Neg flip-flops are cleared at T4.2 time. These flip-flops are shown in the logic diagrams, figure 9-33.

2. Effect of Hold Flip-Flops. The Hold 1 and Hold 2 flip-flops are set during the multiply operation to prevent the reading of the next instruction and the clearing of the multiply function code from F. The effect of these flip-flops is the same as described for the shift instructions in information sheet number 6.3. Since the Hold flip-flops are set during the R1-sequence, the R1-sequence remains active during the multiplication.

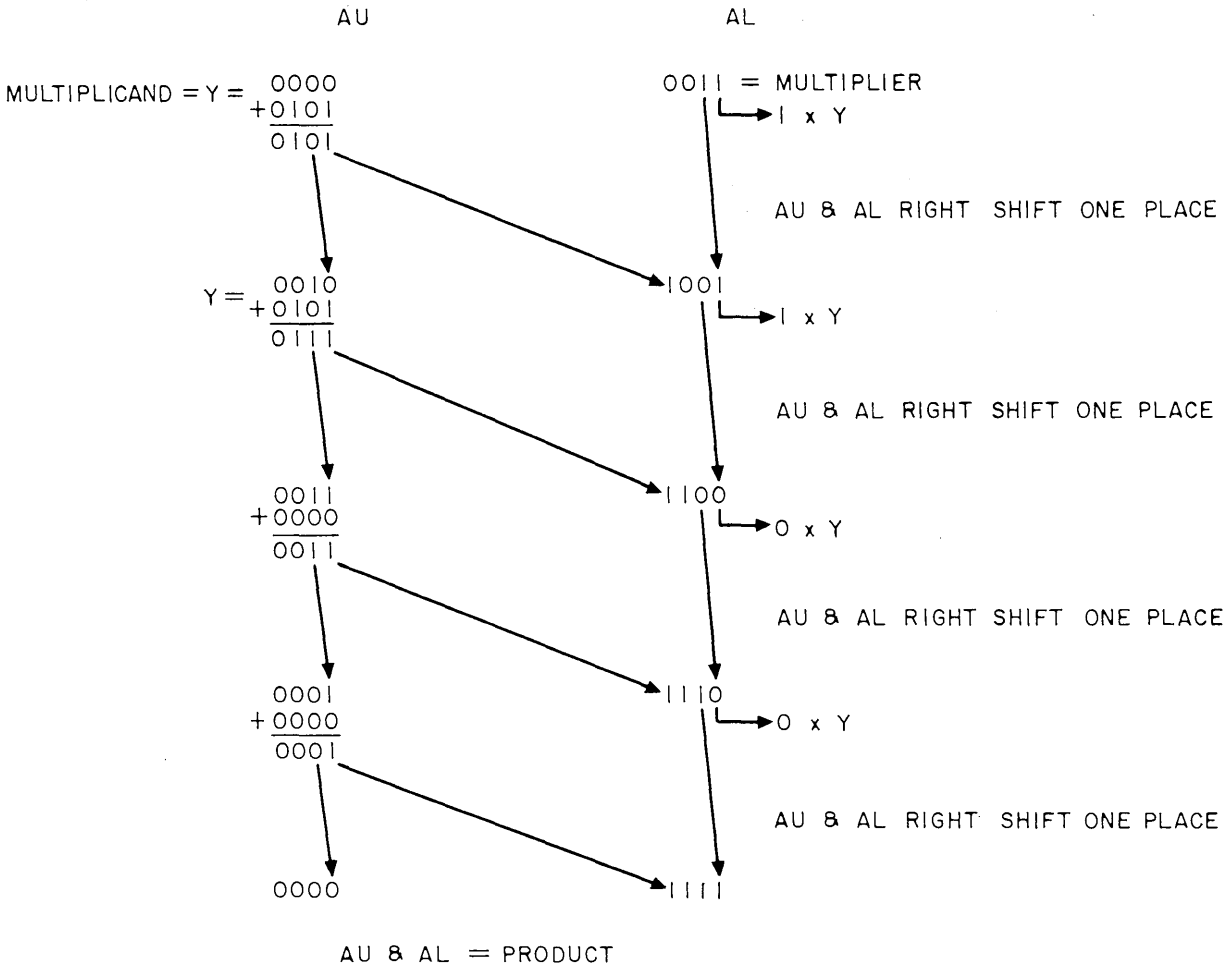


Figure 6.6-2. Example of Binary Multiplication, 1219 Computer Method

3. Rl and Multiply Sequences.

a) Data Flow Block Diagram.

1) Prior to Multiply Termination. Refer to figure 6.6-3 for a block diagram description of the execution of f = 24, 25, prior to the multiplication termination.

The Rl-sequence uses a memory reference to obtain the operand Y. This is the multiplicand. D receives either Y or its complement at T2.4 time. If Y is negative it appears complemented in D in its positive form. This is part of the initial sign connection operation which makes both the multiplicand and the multiplier positive numbers before the multiplication operation. At the completion of the multiply, the result is, if necessary, made negative according to the signs of the original multiplicand and multiplier.

The multipler in AL is also made positive by complementing at T2.2 time if its initial value is negative.

The actual multiplication of D x AL is effected by the multiplication sequence which runs in parallel with the R1-sequence. The 36-bit value in AU and AL is right shifted one place into X and W, respectively, at T2.4 time. The Multiplier Store flip-flop is used to record to value of AL_{00} which is the multiplier bit to be examined. If $AL_{00} = 1_2$, the multiplicand in D is added to the partial product in X (0's initially) and placed in AU at T3.2 time. If $AL_{00} = 0_2$, nothing is added to the partial product in X and the unchanged value is placed in AU.

During each multiplication step, AU and AL are right shifted one place and either the multiplicand or nothing is added to the partial product depending upon the current multiplier bit (AL_{00}). The number of steps is controlled by K0, K1, and the K0-1 adder. K0 is initially set to 19_{10} and is decremented during each multiplication step. When it contains 0, the operation is terminated. The resulting 19_{10} right shifts of AU and AL will have shifted the multiplier out of AL and properly positioned the final product in AU and AL.

As discussed in a later sheet, the operand could be obtained from bootstrap or control memory.

2) Multiply Termination (Final Sign Correction). When K0 = 0, the multiplication operation is terminated and AU and AL contain the final product. If the original signs of Y (multiplicand) and AL (multiplier) were unlike, the product must be made negative. Since both Y and AL were made positive prior to the multiplication, the product should be also positive. The product is left positive if both Y_i and AL_i had like signs. If the original signs are unlike, AU and AL are complemented to yield a negative product. Refer to figure 6.6-4 for a block diagram description of the final sign correction operation which occurs at the completion of the multiplication operation.

b) Essential Commands. The commands which effect the multiplication operations are enabled by the 0XL00 and 0XL01 Multiply Sequence flip-flops and are timed by the master clock phases.

Refer to table 6.6-1 for a sequential list of essential R1, next I, and multiply sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

6.6-5. SUMMARY

The MULAL and MULALB instructions use the Up or U_{SR} which is formulated in D during the I-sequence. The R1 and multiply sequences and the first portion of the next I-sequence are required to complete the executions of these instructions.

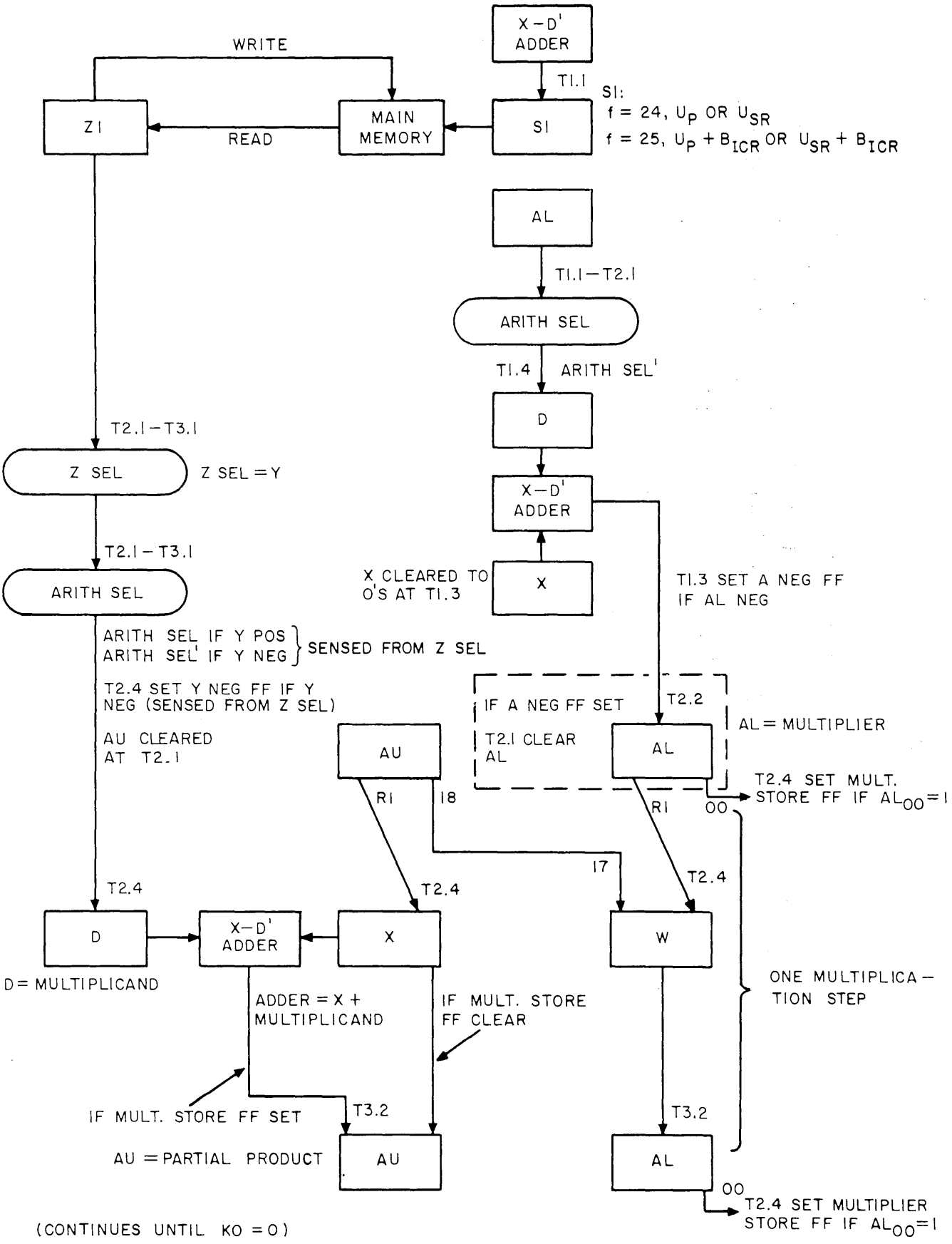


Figure 6.6-3. R1 and Multiply Sequence Data Flow

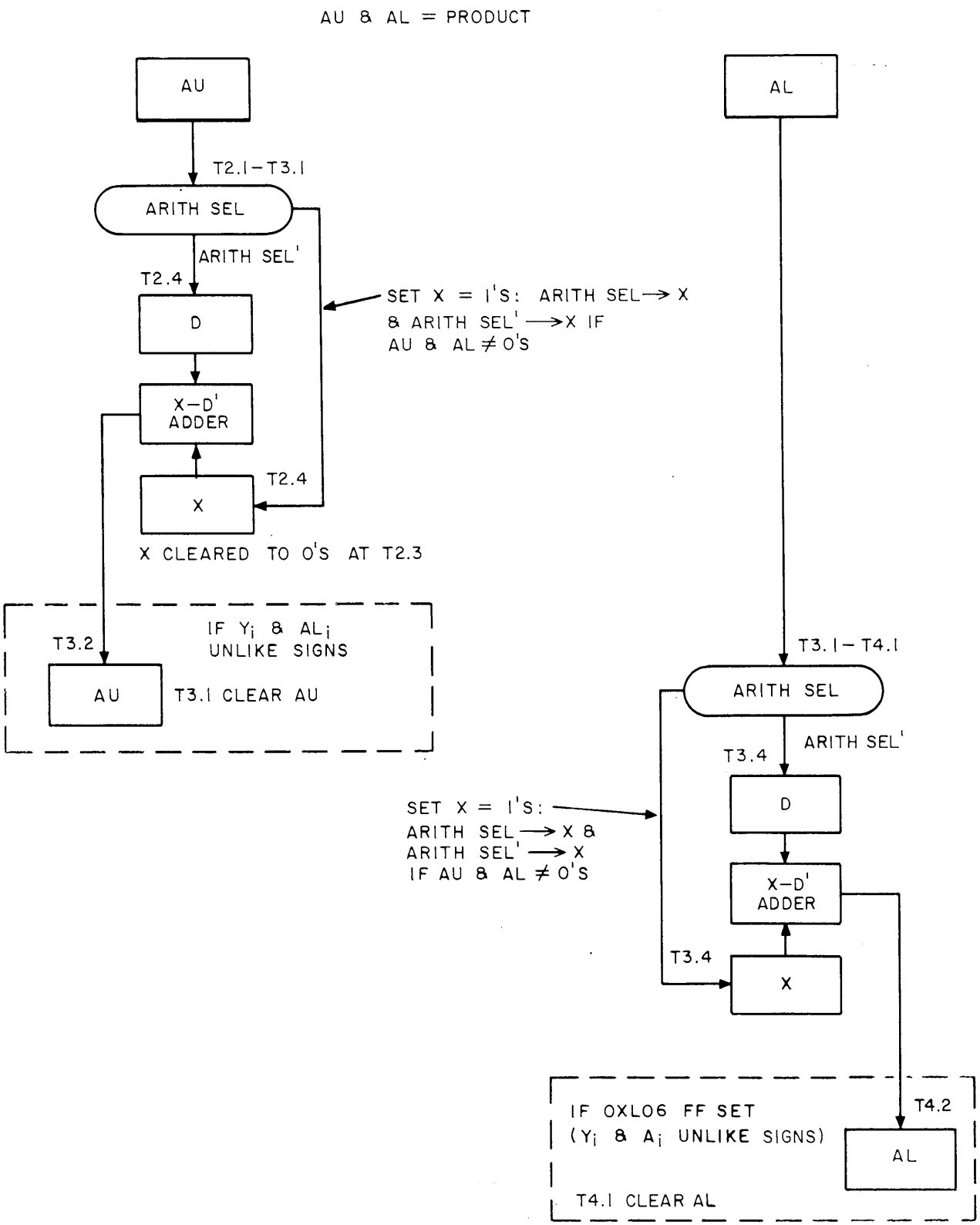


Figure 6.6-4. Multiply Final Sign Correction Data Flow

TABLE 6.6-1. R1, MULTIPLY AND NEXT I-SEQUENCE ESSENTIAL COMMANDS

TIME NOTATION	COMMANDS
	R1 SEQUENCE
T4.4	Clear S1
T1.1	AL→Arith Sel, Adder→S1, Init Memory
T1.3	Set A Neg ff if AL neg, clear D, clear X, clear Z1
T1.4	Arith Sel'→D, clear K0, clear Scale Factor ff (1XL00)
T2.1	Clear AU, clear AL if A Neg ff set, set K0 = 19_{10}, set 0XL01 ff, Z1→Z Sel, Z Sel ⟶ Arith Sel, drop AL→Arith Sel
T2.2	Adder→AL if A Neg ff set, Clear K1
T2.3	Clear D, set 0XL00 ff, clear X, clear W, clear Mult. Store ff, K0-1→K1
T2.4	Set Y Neg ff if Y neg *, Arith Sel→D if Y pos*, Arith Sel'→D if Y neg*, clear K0, AUR1→X, ALR1→W ***AU_{18}→W_{17}, set Mult. Store ff if AL_{00} = 1
T3.1	K1→K0 (K0 = 18), clear AL, clear AU
T3.2	Set Hold 1 ff, **W→AL, Adder→AU if Mult. Store ff set X→AU if Mult. Store ff clear, clear K1
T3.3	Set Hold 2 ff, K0-1→K1, clear X, clear W (continues until K0 = 0)
T1.1	Clear AL, clear AU, K1→K0 (K0 = 0)
T1.2	W→AL, Adder→AU if Mult. Store ff set X→AU if Mult. Store ff clear, clear K1
T1.3	Set 0XL02 ff, clear 0XL00 ff, K0-1→K1 clear X, clear W, clear Mult. Store ff
T1.4	AUR1→X, ALR1→W, ***AU_{18}→W_{17}, set Mult. Store ff if AL_{00} = 1
T2.1	Clear 0XL01 ff, set 0XL03 ff, AU→Arith Sel, set Clear-Hold ff
T2.3	Set 0XL04 ff, clear 0XL02 ff, clear D, clear X
T2.4	Arith Sel'→D, Arith Sel→X & Arith Sel'→X if AU & AL ≠ 0's (Set X = 1's)
T3.1	Set 0XL05 ff, if Y_i & AL_i unlike signs, clear 0XL03 ff, ****AL→Arith Sel, drop AU→Arith Sel, clear AU if Y_i & AL_i unlike signs
T3.2	Adder→AU if Y_i & AL_i unlike signs
T3.3	****set 0XL06 ff, clear 0XL04 ff, **** clear D
T3.4	****Arith Sel'→D, Arith Sel→X & Arith Sel'→X if AU & AL ≠ 0's (Set X = 1's)
T4.1	Clear 0XL05 ff, clear AL if 0XL06 ff set, drop AL→Arith Sel
T4.2	Adder→AL if 0XL06 ff set, clear Hold 1 ff, clear Clear-Hold ff
T4.3	Clear 0XL06 ff
	I-SEQUENCE OF NEXT INSTRUCTION
T1.3	Clear Hold 2 ff

*Sign of Y is sensed from Z select.

**The transmission of bit 00 for the W→AL command is through gate 83A00 in the logic diagrams, figure 9-33.

***AU_{18}→W_{17} data flow is through gate 31W17 in the logic diagrams, figure 9-33, and is enabled by the ALR1→W command.

****These events occur only if the 0XL05 ff is set to perform final sign correction.

SECTION 6 - ARITHMETIC SECTION

6.7. INSTRUCTION EXECUTION OF DIVA, DIVAB

6.7-1. OBJECTIVES

To present the detailed theory of operation involved in the execution of instructions with f = 26, 27.

6.7-2. INTRODUCTION

These instructions divide the combined content of AU and AL by the content of memory.

6.7-3. REFERENCES

a. UNIVAC 1219 Technical Manual, Volume I, Paragraphs 4-4c(1)(d) and 4-7, tables 4-11 and 4-12.

b. UNIVAC 1219 Technical Manual, Volume II, Section 9 (logic diagrams).

6.7-4. INFORMATION

a. General Description.

1. Instruction Interpretation.

a) DIVA, f = 26. This instruction divides the 36-bit value in AU and AL by the operand Y. AU contains the more significant bits. The origin of Y is memory at the address U_P if SR is inactive or U_{SR} if SR is active. The quotient appears in AL and the remainder is held in AU. The sign of the remainder is the same as the dividend (AU and AL).

b) DIVAB, f = 27. Except for the address of Y, this instruction is the same as f = 26. The address of Y is either U_P + B or U_{SR} + B depending upon the activeness of SR. The B register is specified by ICR.

2. Execution Sequences.

a) I-Sequence. During the I-sequence which obtains the instruction from memory, the address of the operand is formulated from U, P, SR, and B.

b) R1-Sequence. The R1-sequence uses a memory reference to obtain the operand Y.

c) Divide Sequence. The divide sequence uses a special timing chain which runs in parallel to main timing and controls the actual division.

3. Division Procedure.

a) Pencil and Paper Method. Division with the binary number system is quite simple since each division step produces a quotient bit of either 1_2 or 0_2. During the division step, the divisor is compared with the partial dividend. If it is less than or equal to the partial dividend, a 1_2 is set in the corresponding quotient bit position and the divisor is subtracted from the partial dividend. If the division cannot be performed (partial dividend less than the divisor), a 0_2 is set in the quotient bit and 0's are subtracted from the partial dividend which does not alter its value. Refer to figure 6.7-1 for an example of the normal "pencil and paper" method.

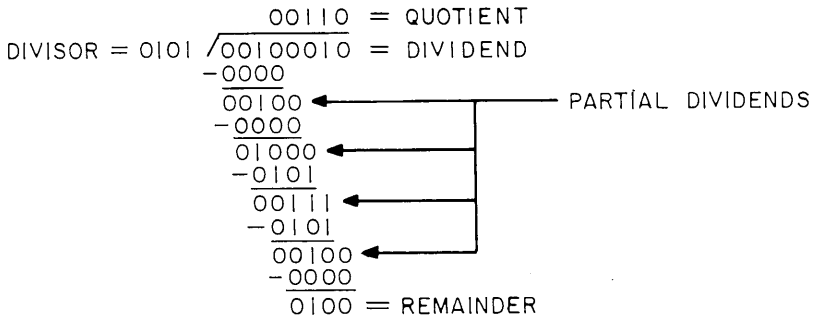


Figure 6.7-1. Example of Binary Division, Pencil and Paper Method

b) 1219 Computer Method. In the 1219, the procedure is basically the same as described above. However, instead of right shifting the divisor when subtracting from the partial dividend, the partial dividend is shifted left. Also an initial left shift of the dividend is performed. The most significant bit of the dividend which is shifted out is 0_2 because the dividend in AU and AL is made positive prior to the division operation. As the dividend is left shifted out of AL and into AU, the quotient is shifted into AL. Refer to figure 6.7-2 for the same numerical example using the 1219 Computer method.

b. Detailed Analysis.

1. I-Sequence. Most of the I-sequence operations are as previously described. If necessary, refer to study guide sheet number 5.4 for a detailed description. At the end of the I-sequence, the X-D' adder is outputting the address of the operand.

In addition to the normal I-sequence operations, the Y Neg and A Neg flip-flops are cleared at T4.2 time. These flip-flops are shown in the logic diagrams, figure 9-33.

2. Effect of Hold Flip-Flops. The Hold 1 and Hold 2 flip-flops are set during the divide operation to prevent the reading of the next instruction and the clearing of the divide function code from F. The effect of these flip-flops is the same as described in the shift instructions in information sheet number 6.3. Since the Hold flip-flops are set during the R1-sequence, the R1-sequence remains active during the division.

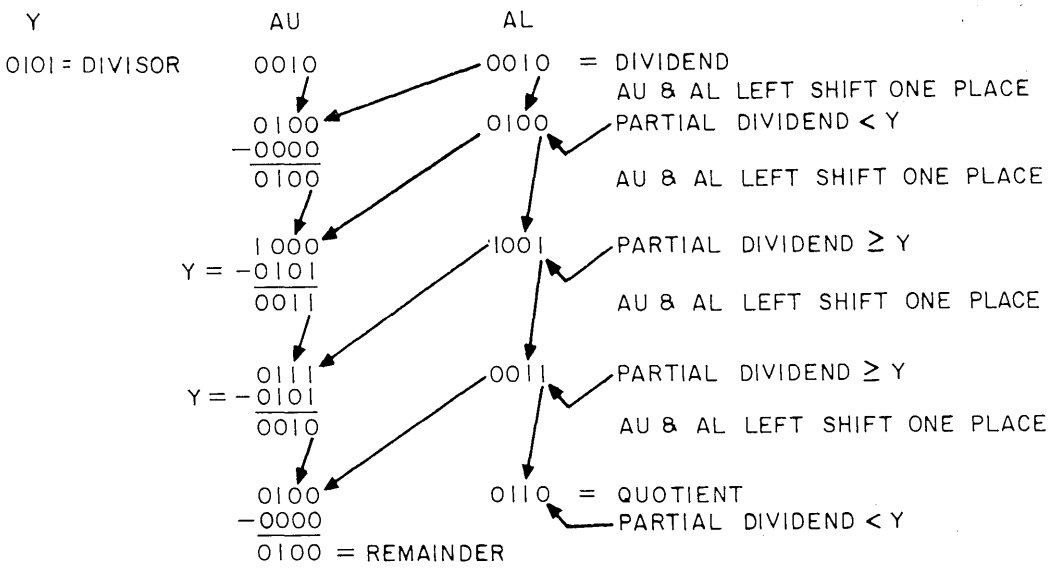


Figure 6.7-2. Example of Binary Division, 1219 Computer Method

3. Rl and Divide Sequences.

a) Data Flow Block Diagram.

1) Prior to Divide Termination. Refer to figure 6.7-3 for a block diagram description of the execution of f = 26, 27 prior to the division termination.

The Rl-sequence uses a memory reference to obtain the operand Y. This is the divisor. D receives either Y or its complement at T2.4 time. The X-D' adder is used to subtract X- divisor. The divisor is made positive as it appears presented to the adder. Therefore, if Y is positive, D receives Y' which means D' = Y; and the adder outputs X-D'. If Y is negative, D receives Y; and the adder output is X - Y'. In this case, the adder uses the operand in its complemented form which would cause it to become a positive value.

The dividend is also made positive prior to the division operation. The more significant half in AU is complemented at T2.2 time if AU_i is negative. The lower half of the dividend is not actually made positive in AL, but it is complemented if necessary as it is shifted into AU via X one bit at a time.

The actual division of AUAL ÷ D' is effected by the divide sequence which runs in parallel with the Rl-sequence. The 36-bit value in AU and AL is left shifted one place into X and W, respctively, at T2.4 time. The value in X (partial dividend) is compared with D' (divisor). If X is greater than or equal to D', the division of this step can occur and is indicated by the absence of an end-around borrow ($\overline{EAB}$). The divisor is then subtracted from the partial dividend and their difference is placed in AU. AL_{00} is set to 1_2 in this case which is the quotient bit value for this division step.

If X is less than D' as indicated by an end-around borrow, the division cannot occur. 0's are subtracted from the partial dividend and it is transferred unaltered from X to AU. In this case, nothing is set in AL_{00}, and it remains a 0_2.

During each division step, the operations described above occur. The number of steps is controlled by K0, K1, and the K0-1 adder. K0 is initially set to 18_{10} and is decremented by a -1 during each division step. When it contains 0, the operation is terminated. The resulting 18_{10} left shifts of AU and AL will have shifted the dividend out and properly positioned the quotient in AL. The remainder is the result of the operation with the last partial dividend and appears in AU.

As discussed in a later sheet, the operand could be obtained from bootstrap or control memory.

2) Divide Termination (Final Sign Correction). When K0 = 0, the division operation is terminated. AL contains the quotient. AU contains the remainder. If the original signs of Y (divisor) and AUAL (dividend) were unlike, the quotient must be made negative. Since both Y and AUAL were made positive prior to the division, the quotient should be also positive. The quotient is left positive if both Y_i and $AUAL_i$ had like signs. If the original signs were unlike, AL is complemented to yield a negative quotient. The sign of the remainder in AU is adjusted by complementing if necessary to make it the same as $AUAL_i$. Refer to figure 6.7-4 for a block diagram description of the final sign correction operation which occurs at the completion of the division operation.

b) Essential Commands. The commands which effect the division operations are enabled by the 0XL00 and 0XL01 Divide Sequence flip-flops and are timed by the master clock phases.

Refer to table 6.7-1 for a sequential list of essential R1, next I, and divide sequence events. Develop these commands by referring to the proper enable pages in the logic diagrams.

When the divisor and the first partial dividend are compared at T3.2 time, AL_{00} is set to 1_2 if the division can be done ($\overline{EAB}$). This bit position is the most significant bit position of the quotient; and, if set, causes the quotient to be a negative value prior to final sign correction. Since both the divisor and dividend are made positive prior to the division, the quotient should be also positive. Therefore, if the first division can be done, there is an error due to the size of the original numbers. This error condition is recorded by the Overflow flip-flop, which is set during the first divisor and dividend comparison if there is no end around borrow. The condition of this flip-flop can be later sensed by an f = 50:52, 50:53 instruction.

6.7-5. SUMMARY

The DIVA and DIVAB instructions use the value Up or U_{SR} which is formulated in D during the I-sequence. The R1 and divide sequences and the first portion of the next I-sequence are required to complete the executions of these instructions.

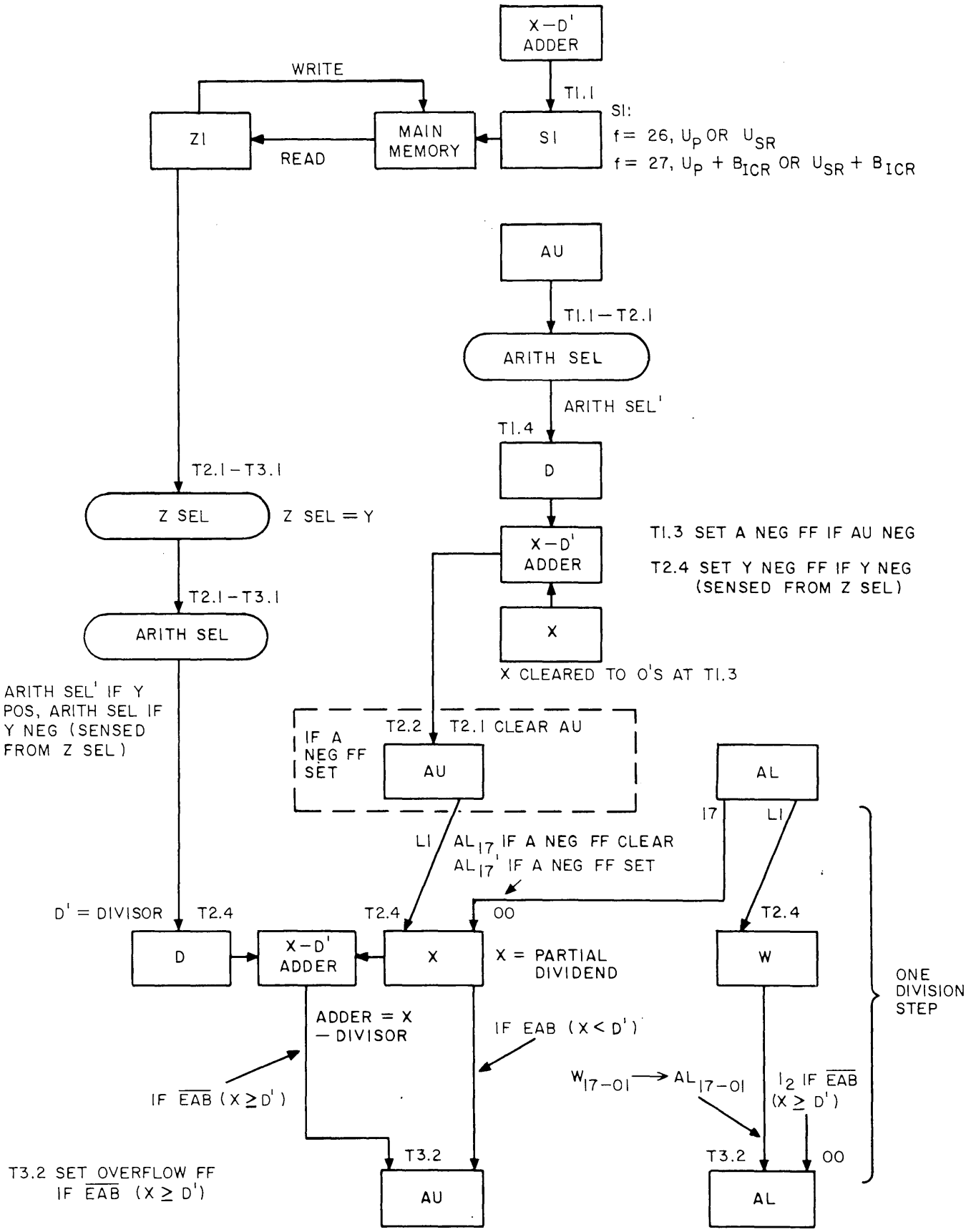


Figure 6.7-3. R1 and Divide Sequence Data Flow

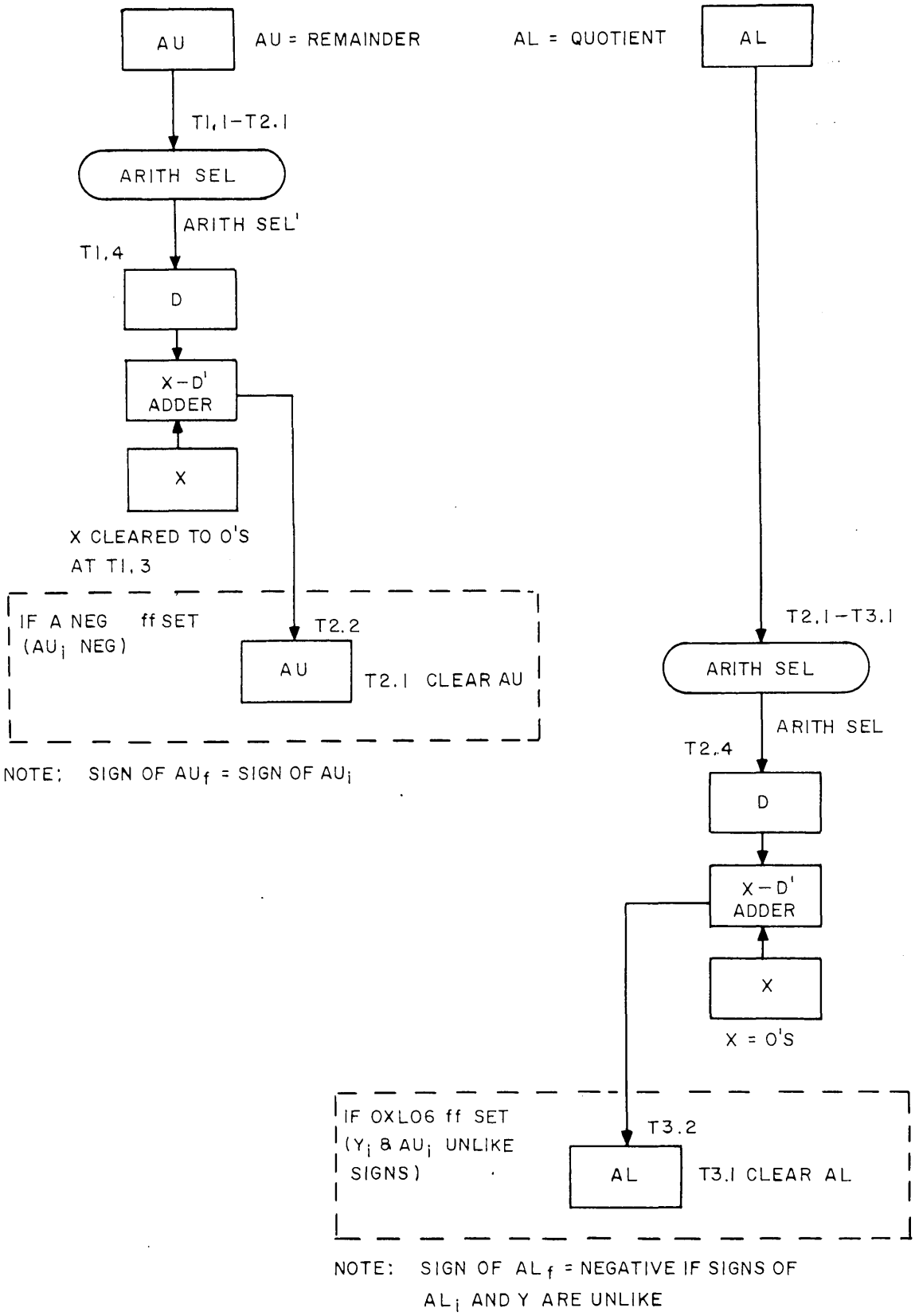


Figure 6.7-4. Divide Final Sign Correction Data Flow

TABLE 6.7-1. R1, DIVIDE, AND NEXT I SEQUENCE ESSENTIAL COMMANDS

TIME NOTATION	COMMANDS
	R1 SEQUENCE
T4.4	Clear S1
T1.1	AU→Arith Sel, Adder→S1, Init Memory
T1.3	Set A Neg ff if AU neg, clear D, clear X, clear Z1
T1.4	Arith Sel'→D, clear K0, clear Scale Factor ff (1XL00)
T2.1	Clear AU if A Neg ff set, set K0 = 18_{10}, set 0XL01 ff, Z1→ Z Sel, Z Sel→Arith Sel, drop AU→Arith Sel
T2.2	Adder→AU if A Neg ff set, clear K1
T2.3	Set 0XL00 ff, K0-1→K1, clear D, clear X, clear W
T2.4	Arith Sel'→D if Y pos*, Arith Sel→D if Y neg*, clear K0, set Y Neg ff if Y neg*, AUL1→X, ALL1→W, **AL_{17}→X_{00} if A Neg ff clear, **AL_{17}→X_{00} if A Neg ff set
T3.1	K1→K0, clear AU, clear AL, drop Z1→ Z Sel, drop Z Sel→Arith Sel
T3.2	Set Hold 1 ff, W_{17-01}→AL_{17-01}, ***1_2→AL_{00} if $\overline{EAB}$, X→AU if EAB, Adder→AU if EAB, clear K1, set Overflow ff if $\overline{EAB}$
T3.3 ⋮ T4.1	Set Hold 2 ff, K0-1→K1, clear X, clear W (continues until K0 = 0) Clear AU, clear AL, K1→K0 (K0 = 0)
T4.2	W_{17-01}→AL_{17-01}, ***1_2→AL_{00} if $\overline{EAB}$, X→AU if EAB, Adder→AU if $\overline{EAB}$, clear K1
T4.3	Set 0XL02 ff, clear 0XL00 ff, K0-1→K1, clear X, clear W
T4.4	AUL1→X, ALL1→W, **AL'_{17}→X_{00} if A Neg ff clear, **AL_{17}→X_{00} if A Neg ff set
T1.1	Clear 0XL01 ff, set 0XL03 ff, AU→Arith Sel, set Clear Hold ff
T1.3	Set 0XL04 ff, clear 0XL02 ff, clear D, clear X
T1.4	Arith Sel'→D
T2.1	Set 0XL05 ff if Y_i & AU_i unlike signs, clear 0XL03 ff, ****AL→Arith Sel, drop AU→Arith Sel, Clear AU if A Neg ff set
T2.2	Adder→AU if A Neg ff set
T2.3	****set 0XL06 ff, clear 0XL04 ff, ****clear D
T2.4	****Arith Sel'→D
T3.1	Clear 0XL05 ff, clear AL if 0XL06 ff set, drop AL→Arith Sel
T3.2	Adder→AL if 0XL06 ff set
T3.3	Clear 0XL06 ff
T4.2	Clear Hold 1 ff, clear Clear-Hold ff
	I-SEQUENCE OF NEXT INSTRUCTION
T1.3	Clear Hold 2 ff

*Sign of Y is sensed from Z-Select.

**AL_{17}→X_{00} and AL'_{17}→X_{00} data flow is through gate 31X00 in the logic diagrams, figure 9-33, and is enabled by the AUL1→X command.

***1_2→AL_{00} data flow is through gate 83A00 in the logic diagrams, figure 9-33, and is enabled by the W→AL command.

****These events occur only if the 0XL05 ff is set to perform final sign correction.