INSTRUCTION MANUAL

EXTENDED ARITHMETIC ELEMENT

KE09A

PDP-9

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

Corley

DEC-09-I2AA-D

INSTRUCTION MANUAL

KE09A

EXTENDED ARITHMETIC ELEMENT

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing July 1968
2nd Printing February 1969

Instruction times, operating speeds and the like are included in this manual for reference only; they are not to be taken as specifications.

CONTENTS

CONTENTS (Cont)

TABLES (Cont)

CHAPTER 1
INTRODUCTION

This manual contains operation and maintenance information for the KE09A Extended Arithmetic Element (EAE) of the Programmed Data Processor PDP-9, manufactured by Digital Equipment Corporation, Maynard, Massachusetts. For a complete understanding of the option and its relation to the basic PDP-9 system, the user must be thoroughly familiar with the contents of the PDP-9 Maintenance Manual, F-97.

1.1 PURPOSE

The EAE option facilitates high-speed multiplication, division, shifting, normalizing, and register manipulation. Installation of the EAE adds an 18-bit multiplier-quotient register (MQ) and a 6-bit step counter (SC) to the basic PDP-9 system. The option logic occupies space in the central processor wing of the basic PDP-9 system, as indicated in the CP UML drawing KC8. All logic module locations have been prewired into the system. The contents of the MQ can be selected by the REGISTER DISPLAY switch on the PDP-9's operator console for display in the REGISTER indicator.

The EAE operates asynchronously with the basic system, permitting computations to be performed in the shortest possible time. Furthermore, instructions can be microcoded so that several nonconflicting EAE operations can be performed by one instruction, thereby simplifying arithmetic programming. Maximum multiplication and division time is 12 μs.

1.2 RELATED DOCUMENTS

The PDP-9 library offers a complete package of single- and multiple-precision programming routines for use with the EAE. These and other related documents and tapes are listed in Chapter 1 of the PDP-9 Maintenance Manual.

1.3 POWER REQUIREMENTS

The EAE needs no source of primary or dc power other than that already furnished with the basic PDP-9 system. All necessary power is prewired to the module locations.

1.4 ENGINEERING DRAWINGS AND REFERENCES

Throughout this manual all references to EAE option drawings and basic PDP-9 system drawings are abbreviated as in the PDP-9 Maintenance Manual. Refer to Chapter 1 of the Maintenance Manual for abbreviation codes. As an aid to understanding the EAE, a simplified version of LINC Control drawing KC15 along with a portion of EAE logic appears on an illustration at the end of this manual.

Chapter 5 of this option manual contains a complete set of EAE option drawings indexed by their full drawing number codes, along with all module circuit schematics.

1.5 SPECIFICATIONS

1.5.1 Functional Characteristics

The EAE enables fast, flexible, hardware execution of the following signed or unsigned functions.

a. Shifting the contents of the primary arithmetic registers (AC, MQ) right or left, requires 4 to 18 μs.

b. Normalizes the quantity in the primary arithmetic registers, i.e., shifts the contents left to remove leading binary 0s for the purpose of preserving as many significant bits as possible. The time required is 4 to 18 μs.

c. Multiplication is performed in 5 to 12 μs.

d. Division including integer divide and fraction divide require 5 to 12 μs. Divide overflow indication is furnished by the LINK when signed division produces a quotient exceeding $\pm 377777_8$ in magnitude, or unsigned division produces a quotient exceeding 777777_8 in magnitude.

e. Basic setup instructions to manipulate the data in the registers preparatory to execution of the above instructions requires 2 μs.

1.5.2 Operating Characteristics

Heat Dissipation	108 BTU/hr
Power Dissipation	0.032 kW

CHAPTER 2
INSTALLATION AND OPERATION

2.1 INSTALLATION

Complete installation of the EAE option merely involves plugging the logic modules into their assigned locations in the central processor wing, and ascertaining that certain jumpers are removed. The following jumpers are in place to allow FORTRAN programming without the EAE. They must be removed for EAE operations (refer to drawing KC27).

a. AC0 → LINK from E04R to E04B.

b. ADRL(B) from B03D to B03N.

c. MQI(1)/EAE OR ARO from D22P to D23J.

d. TEMP1(1) from B03C to B03T.

e. SCO(1) from B31C to B31P.

2.2 MANUAL CONTROLS AND INDICATORS

The EAE option contains no manual controls and indicators other than those prewired into the PDP-9 operator's console. Table 2-1 lists and describes these controls and indicators. Refer to the PDP-9 Maintenance Manual for details.

Table 2-1
Operating Controls and Indicators

Control/Indicator	Function
REGISTER DISPLAY switch and REGISTER indicator	MQ position displays contents of the MQ register in the REGISTER indicator when the computer is in a stop condition. EAE position is presently not used (not wired).

2.3 PROGRAMMING CONSIDERATIONS

The EAE option adds the instructions listed in Table 2-2 to the basic PDP-9 instruction repertoire. See Table 2-3 for execution times.

Table 2-2
EAE Instructions

Octal Code	Mnemonic	Operation
640000	EAE	Basic EAE instruction. Acts as a NOP instruction.
640001	OSC	Inclusive-OR the SC with the AC. The contents of the AC are inclusive-ORed with the contents of the 6-bit SC on a bit-for-bit basis, and the results are left in AC12 through 17. If corresponding SC and AC bits are 0, the result is 0. If corresponding bits are 1 or differ, the result is 1. The previous contents of the AC are lost, the LINK and the SC remain unchanged.
640002	OMQ	Inclusive-OR the MQ with the AC. The contents of the AC are inclusive-ORed with the contents of the MQ on a bit-for-bit basis, and the results are left in the AC. If corresponding MQ and AC bits are 0, the result is 0. If corresponding bits are 1 or differ, the result is 1. The previous contents of the AC are lost, the LINK and the MQ remain unchanged.
640004	CMQ	Complement the MQ. The previous contents of the MQ are lost, the LINK and the AC remain unchanged.
641001	LACS	Load AC12 through 17 with the contents of the SC. The previous contents of AC12 through 17 are lost, the LINK and the SC remain unchanged.
641002	LACQ	Load the AC with the contents of the MQ. The previous contents of the AC are lost, the LINK and the MQ remain unchanged.
644000	ABS	Get the absolute value of the AC. If the sign (AC00) of the contents of the AC is negative, the contents are 1s complemented. The LINK remains unchanged.
650000	CLQ	Clear the MQ. The previous contents of the MQ are lost, the LINK and the AC remain unchanged.
652000	LMQ	Load the MQ with the contents of the AC. The previous contents of the MQ are lost, the LINK and the AC remain unchanged.
664000	GSM	Get the sign and magnitude of the AC. Places the sign (AC00) of the AC contents in the LINK, and if negative, 1s complements the contents.
6405XX	LRS	Long Right Shift. Shifts the contents of the LINK, AC, and MQ right the number of positions indicated in bits XX. The LINK is usually initialized to 0 and shifted unchanged on each step.
6605XX	LRSS	Long Right Shift, Signed. Shifts the contents of the LINK, AC and MQ right the number of positions indicated in bits XX. AC00 is initially stored in the LINK, then shifted unchanged on each step.

Table 2-2 (cont)
EAE Instructions

Octal Code	Mnemonic	Operation
6406XX	LLS	Long Left Shift. Shifts the contents of the LINK, AC and MQ left the number of positions indicated in bits XX. The LINK is usually initialized to 0 and shifted unchanged on each step.
6606XX	LLSS	Long Left Shift, Signed. Shifts the contents of the LINK, AC and MQ left the number of positions indicated in bits XX. AC00 is initially stored in the LINK, then shifted unchanged on each step.
6407XX	ALS	Accumulator Left Shift. Shifts the contents of the LINK and AC left the number of positions indicated in bits XX. The LINK is usually initialized to 0 and shifted unchanged on each step.
6607XX	ALSS	Accumulator Left Shift, Signed. Shifts the contents of the LINK and AC left the number of positions indicated in bits XX. AC00 is initially stored in the LINK, then shifted unchanged on each step.
640444	NORM	Normalize. Shifts the contents of the LINK, AC and MQ left until AC00 and AC01 differ or until the maximum of 36 shifts (44_8) occur. The LINK is usually initialized to 0 and shifted unchanged on each step.
660444	NORMS	Normalize, Signed. Shifts the contents of the LINK, AC and MQ left until AC00 and AC01 differ or until the maximum of 36 shifts (44_8) occur. AC00 is initially stored in the LINK and then shifted unchanged on each step.
6531XX	MUL	Multiply. Multiplies the number in the AC (multiplier) by the number in the next core memory location (multiplicand) to form a product in the AC and MQ. MUL transfers the multiplier to the MQ, clears the AC, and fetches the multiplicand from memory. Bits XX command the desired precision of the product (22_8 or 18_{10} steps for maximum 36-bit precision). The LINK must be cleared previously and remains unchanged.
6571XX	MULS	Multiply, Signed. Multiplies the number in the AC (multiplier) by the number in the next core memory location (absolute value multiplicand) to form a signed product in the AC and MQ. AC00 and AC01 receive the product sign. A previous LAC/GSM/DAC CAND sequence places the multiplicand sign in the LINK and the absolute value in memory. MULS transfers the multiplier to the MQ, performs 1s complements of the multiplier if its sign is negative, fetches the absolute value multiplicand from memory, and clears the LINK. Bits XX command the desired precision of the product (22_8 or 18_{10} steps for maximum 36-bit precision).

Table 2-2 (cont)
EAE Instructions

Octal Code	Mnemonic	Operation
6403XX	DIV	Divide. Divides the number in the AC and MQ (dividend) by the number in the next core memory location (divisor) to form a quotient in the MQ and remainder in the AC. DIV fetches the divisor from memory. Bits XX command the desired precision of the quotient and remainder (23_8 or 19_{10} steps for maximum 36-bit precision). The LINK must be cleared previously and remains unchanged unless divide overflow occurs. Overflow occurs if the divisor is not numerically greater than the AC portion of the dividend.
6443XX	DIVS	Divide, Signed. Divides the number in the AC and MQ (36-bit double-signed dividend) by the number in the next core memory location (absolute value divisor) to form a signed quotient in the MQ and remainder in the AC. MQ00 receives the sign of the quotient and AC00 receives the original sign of the dividend. A previous LAC/GSM/DAC sequence places the divisor sign in the LINK and the absolute value in the memory. DIVS fetches the absolute value divisor, 1s complements the MQ portion of the dividend if the dividend sign is negative, and clears the LINK. Bits XX command the desired precision of the quotient and remainder (23_8 or 19_{10} steps for maximum 36-bit precision). The LINK remains cleared unless divide overflow occurs. Divide overflow occurs if the divisor is not numerically greater than the AC portion of the dividend.
6533XX	IDIV	Integer Divide. Divides the number in the AC (integer dividend) by the number in the next core memory location (divisor) to form a quotient in the MQ and remainder in the AC. IDIV fetches the divisor from memory, transfers the contents of the AC to the MQ, then clears the AC. Bits XX command the desired precision of the quotient and remainder (23_8 or 19_{10} steps for maximum 36-bit precision). The LINK must be previously cleared and remains unchanged unless divide overflow occurs. Overflow occurs only if the divisor is 0.
6573XX	IDIVS	Integer Divide, Signed. Divides the number in the AC (signed integer dividend) by the number in the next core memory location (absolute value divisor) to form a signed quotient in the MQ and remainder in the AC. MQ00 receives the sign of the quotient and AC00 receives the original sign of the dividend. A previous LAC/GSM/DAC sequence places the sign of the divisor in the LINK and the absolute value in memory. IDIVS fetches the absolute value divisor, transfers the contents of the AC to the MQ, 1s complements them if the dividend sign is negative, and clears the AC and LINK. Bits XX command the desired precision of the quotient and remainder (23_8 or 19_{10} steps for maximum 36-bit precision). The LINK remains cleared unless divide overflow occurs. Overflow occurs only if the divisor is 0.

Table 2-2 (cont)
EAE Instructions

Octal Code	Mnemonic	Operation
6503XX	FRDIV	Fraction Divide. Divides the number in the AC (fraction dividend) by the number in the next core memory location (divisor) to form a quotient in the MQ and remainder in the AC. The binary point is assumed to be at the left of AC00. FRDIV fetches the divisor from memory and clears the MQ. Bits XX command the desired precision of the quotient and remainder (23_8 or 19_{10} steps for maximum 36-bit precision). The LINK must be previously cleared and remains unchanged unless divide overflow occurs. Overflow occurs if the divisor is not numerically greater than the dividend.
6543XX	FRDIVS	Fraction Divide, Signed. Divides the number in the AC (signed fraction dividend) by the number in the next core memory location (absolute value divisor) to form a signed quotient in the MQ and remainder in the AC. The binary point is assumed at the left of AC01. MQ00 receives the sign of the quotient and AC00 receives the original sign of the dividend. A previous LAC/GSM/DAC sequence places the sign of the divisor in the LINK and the absolute value in memory. FRDIVS fetches the absolute value divisor, clears the MQ and LINK, and 1s complements the contents of the AC if the dividend is negative. Bits XX command the desired precision of the quotient and remainder (23_8 or 19_{10} steps for maximum 36-bit precision). The LINK remains cleared unless divide overflow occurs. Overflow occurs if the divisor is not numerically greater than the dividend.

Table 2-3
EAE Operation Times

Number of Shifts*	SETUP, SHIFT, NORM Instructions	MUL, DIV Instructions
0	2**	5***
1	4	5
2,3,4	5	6
5,6,7	6	7
8,9,10	7	8
11,12,13	8	10
14,15,16	10	11
17,18,19	11	12
20,21,22	12	
23,24,25	13	
26,27,28	14	
29,30,31	16	
32,33,34	17	
35,36	18	

*Initial step count.

**SETUP Instructions.

***DIV OV causes divide operation to stop here. MUL and DIV instructions containing initialized step count of 0 stop here with no arithmetic operations undertaken.

CHAPTER 3
PRINCIPLES OF OPERATION

This chapter describes the EAE option in terms of its instruction repertoire and the logic that implements those instructions. The discussions include references to the logic drawings in Chapter 5 and to pertinent drawings of the basic PDP-9 system.

3.1 INSTRUCTION FETCH AND OP CODE DECODING

EAE instructions are fetched from core memory through the fetch cycle processes as are all PDP-9 instructions. The PDP-9 Maintenance Manual explains the fetch cycle processes in detail. Briefly, the BGN process word (10) which concludes a previous execute cycle transfers the current address held in the PC to the MB and starts the next core memory and control memory read operations. MA JAM transfers the current address from the MB to the MA, the core memory cycle starts, and the fetch entry process word (21) is extracted from control memory. Process word 21 increments the address in the MB and transfers it to the PC for the next following fetch cycle (MBO, +1, PCI).

The next CM process word(12) occurs while the core memory reads the addressed memory word into the sense amplifiers. Processes evolved from process word 12 transfer this (instruction) word from the sense amplifiers to the MB, and also gate the op code portion into the IR (SAO, MBI, IRI). The contents of the AC are gated into the AR (ACO, ARI).

The next process word address held in the address portion (CMA00 through 05) of process word 12 is 24. On drawing KC12, the op code detection circuits decode the op code bits IR00, IR01, IR03. These bits, all in the 1 state for an EAE op code of 64_8, produce the REP signal. REP allows the IR bits to modify the control memory address on drawing KC17, boosting this next CM address from 24 to 75. This is the third and last process word extracted during the normal, 1-μs fetch cycle. All EAE operations start from this "EAE execute entry" process word.

3.2 EAE COMMAND DECODING

The EAE option contains an instruction register (see drawing KE4) which accepts bits SA09 through 11 of the instruction word during process 12. These bits contain the code for a particular EAE instruction class, and are fed directly from the register EIR09-11 into the Binary-to-Octal Decoder S151-H02. The S151 module decodes the octal class code to supply an output command level denoting one of the following seven EAE instruction classes.

0_8	SETUP instructions
1	MUL (Multiply) instructions

2	Not used
3	DIV (Divide) instructions
4	NORM (Normalize) instructions
5	LRS (Long Right Shift) instructions
6	LLS (Long Left Shift) instructions
7	ALS (Accumulator Left Shift) instructions

The pertinent command level remains on throughout the succeeding EAE execution processes to determine the particular execute operation, starting with process word 75. The paragraphs that follow discuss each instruction class in detail.

3.3 TIMING AND FLOW

Figure 3-1 is a composite timing diagram for all EAE instruction classes, showing machine cycle time versus process word branching for the various classes. The diagram can be correlated with the operation times listed in Table 2-3 and the flow diagrams KE5 and KE6. Examination of Figure 3-1 reveals the following general features on operating times.

a. All SETUP instructions require two machine cycles, progressing toward the BGN process word (10) that starts the next instruction fetch cycle.

b. All SHIFT instructions, including NORM, branch to process word 50 and continue in accordance with the number of shifts (steps) programmed in bits 12 through 17 of the shift instruction word.

c. All MUL and DIV instructions branch to process word 51 and continue in accordance with the number of shifts (steps) programmed in bits 12 through 17 of the instruction word.

Important features not apparent in Figure 3-1 are: for all instructions other than MUL or DIV, core memory is idle after the initial instruction fetch; for MUL and DIV instructions a core memory cycle occurs during process word 51 in which a multiplicand or divisor is fetched. Thereafter, core memory is not needed by the EAE during the execute cycles, and may be accessed by the DMA channel as a time-saving feature. Ordinarily, the last process word in the fetch cycle contains an SM (start memory) bit in order to read an operand from memory during the execute cycle. In process word 75 this SM bit is absent (0), leaving the memory idle. In process word 51, the SM bit is present (1) to start a memory cycle for MUL or DIV.

3.4 SETUP INSTRUCTIONS

Nine 2-cycle SETUP instructions manipulate the data in the prime arithmetic registers (AC, MQ) in preparation for execution of the arithmetic operations commanded by succeeding MUL and DIV instructions. Table 3-1 shows the instruction format. Table 3-2 through 3-10 list the logic functions that implement the instructions, referencing the appropriate logic drawings.

a. "ADVP" Checks that the memory location following the multiply and/or divide instruction is not modified by the execution of the instruction and that the program address counter is properly incremented during the execution of the instruction.

b. "NEAE" Set up check - Checks the set-up of all EAE signed, unsigned, integer and fraction, multiply and divide instructions. These instructions are executed with a shift count of zero.

c. "SHCT" Shift Counter Test - Executes the Multiply instruction sequentially starting at a shift count of 1 and incrementing it up to a shift count of 22.

d. "STMUL" Sign multiply and divide test - Test all signed multiply and divide instructions.

e. "MULTST" Multiply and Divide Test - This test using worse-case number patterns acts as both a EAE and Adder Test.

f. "MSPEED" Speed Multiply and Divide - This test is in three operations: (1) a sequence of multiply instructions are executed back to back, (2) then a sequence of divide instructions are executed, (3) followed by a sequence of MUL, DIV, MUL, and DIV executed back to back.

4.2.2 Section 2 Random Data Multiply and Divide Test - The Random Data Test verifies that the EAE will multiply and divide random numbers at shift counts 1 through maximum (22 for multiply, 23 for divide) and checks that the LINK is set on divide overflow.

The sequence of testing is as follows:

a. Test the Multiply
   (1) Generate a random number
   (2) Do a software multiply
   (3) Do a hardware multiply
   (4) Compare the results of both operations
   (5) LOOP BACK TO 1 TILL DONE

b. Test the Divide
   (1) Generate a random number
   (2) Do a software divide
   (3) Do a hardware divide
   (4) Compare the results of both operations
   (5) LOOP BACK TO 1 TILL DONE

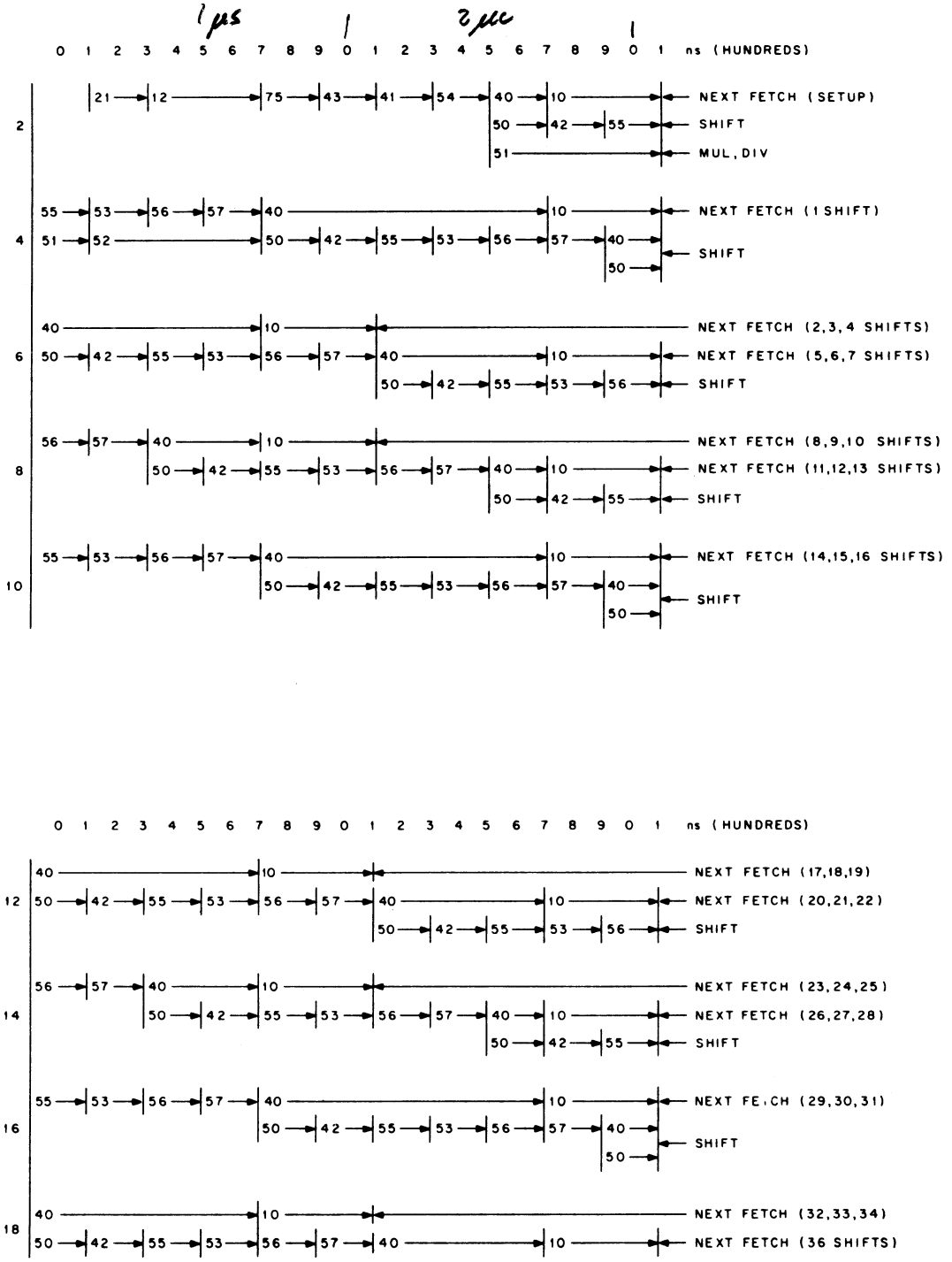


Figure 3-1 EAE Timing

Table 3-1
EAE SETUP Instruction Format

Op Code 64_8			SETUP 0_8	Not Used		
0 1 2	3 4 5	6 7 8	9 10 11	12 13 14	15 16 17	
6	4	0	0	0	1	OSC
6	4	0	0	0	2	OMQ
6	4	0	0	0	4	CMQ
6	4	1	0	0	1	LACS
6	4	1	0	0	2	LACQ
6	4	4	0	0	0	ABS
6	5	0	0	0	0	CLQ
6	5	2	0	0	0	LMQ
6	6	4	0	0	0	GSM

Table 3-2
OSC Functions

640001 Inclusive-OR the SC with the AC

Process	Function	Drawing No.
75	(ACO,ARI,EAE,LI,CONT,CMA43)	KC18
	ACO(1) = AC00-17 → A BUS00-17	KC20
	A BUS00-17 → ADR00-17	KC21
	NOSH = ADR00-17 → O BUS00-17	KC20
	ARI(1) = O BUS00-17 → AR00-17	KC20
	LI(1) = ADRL = LINK → LAR	KC15
	LI(1) = ADRL = LINK → TEMP3	KE3
	SA09(0)∧SA10(0)∧SA11(0) = SETUP	KE4
	EAE(1)∧ARI(1) = SU1(1)	KE3
	SU1(1) = 0 → SCOV,SCOV2,FIRST,EAE RUN,EAE SIGN,MQ SIGN	KE2-3
	SU1(1)∧MB05(0) = EAE OR MQO	KE3
	CM STROBE∧CONT(1) = GO TO 43	KC16
43	(ACI,EAE,CONT,CMA41)	KC18
	CM STROBE∧EAE OR MQO = MQO(1)	KC19
	MQO(1) = MQ00-17 → A BUS00-17	KC20
	A BUS00-17 → ADR00-17	KC21
	NOSH = ADR00-17 → O BUS00-17	KC20
	ACI(1) = O BUS00-17 → AC00-17	KC20
	LI(0) = LAR → LINK	KC15
	CM STROBE∧CONT(1) = GO TO 41	KC16

Table 3-2 (cont)
OSC Functions

640001 Inclusive-OR the SC with the AC

Process	Function	Drawing No.
41	(ACO,MQI,EAE,CONT,CMA54)	KC18
	ACO(1) = AC00-17 → A BUS00-17	KC20
	A BUS00-17 → ADR00-17	KC21
	NOSH = ADR00-17 → O BUS00-17	KC20
	MQI(1) = O BUS00-17 → MQ00-17	KC20
	EAE(1)∧MQI(1)∧SETUP = SU3(1)	KE3
	SU3(1) = SCOV(1)	KE3
	SU3(1) = SCOV2(1)	KE3
	MQI(1)∧MB08(0)∧EAE(1) = EAE OR ARO	KE3
	CM STROBE∧CONT(1) = GO TO 54	KC16
54	(ACI,EAE-R,CONT,CMA40)	KC18
	CM STROBE∧EAE OR ARO = ARO(1)	KC19
	EAE-R(1)∧MB17(1)∧SETUP = SCO	KE2
	ARO(1) = AR00-17 → A BUS00-17	KC20
	A BUS00-17 → ADR00-17	KC21
	NOSH = ADR00-17 → O BUS00-17	KC20
	SCO = SC12-17 → O BUS12-17	KC22
	ACI(1) = O BUS00-17 → AC00-17	KC20
	EAE-R(1) = O BUS L → TEMP2	KE3
	CM STROBE∧CONT(1) = GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	CLK(B) + 670 ns ∧ EAE(1)∧DONE(1) = INPUT IO RESTART	KD3(3)
	INPUT IO RESTART = IO RESTART	KD3(3)
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

Table 3-3
OMQ Functions

640002 Inclusive-OR the MQ with the AC

Process	Function	Drawing No.
75	Same as OSC	
43	Same as OSC	
41	Same as OSC plus	
	SU3(1)∧MB16(1) = EAE OR MQO	KE3
54	(ACI,EAE-R,CONT,CMA40)	KC18
	CM STROBE∧EAE OR ARO = ARO(1)	KC19
	CM STROBE∧EAE OR MQO = MQO(1)	KC19

Table 3-3 (cont)
OMQ Functions

640002 Inclusive-OR the MQ with the AC (cont)

Process	Function	Drawing No.
54 (cont)	ARO(1) = AR00-17 → A BUS00-17	KC20
	MQO(1)= MQ00-17 → A BUS00-17	KC20
	A BUS00-17 → ADR00-17	KC21
	NOSH = ADR00-17 → O BUS00-17	KC20
	ACI(1) = O BUS00-17 → AC00-17	KC20
	EAE-R(1) = O BUS L → TEMP2	KE3
	CM STROBE∧CONT(1) = GO TO 40	KC16
40	Same as OSC	
10	Same as OSC	

Table 3-4
CMQ Functions

640004 Complement the MQ

Process	Functions	Drawing No.
75	Same as OSC	
43	Same as OSC	
41	Same as OSC plus:	
	SU3(1)∧MB15(1) = CMPL	KE3
	CMPL = $\overline{\text{ADR00-17}}$ → O BUS00-17	KC20
54	Same as OSC except:	
	MB17(0) = $\overline{\text{SCO}}$	
40	Same as OSC	
10	Same as OSC	

Table 3-5
LACS Functions

641001 Load the AC with the SC

Process	Function	Drawing No.
75	Same as OSC	
43	Same as OSC	
41	Same as OSC except:	
	MQI(1)∧MB08(1)∧EAE(1) = $\overline{\text{EAE OR ARO}}$	

Table 3-5 (cont)
LACS Functions

641001 Load the AC with the SC

Process	Functions	Drawing No.
54	Same as OSC except: CM STROBE∧$\overline{\text{EAE OR ARO}}$ = ARO(0)	
40	Same as OSC	
10	Same as OSC	

Table 3-6
LACQ Functions

641002 Load the AC with the MQ

Process	Function	Drawing No.
75	Same as OSC	
43	Same as OSC	
41	Same as OSC plus:	
54	MQI(1)∧MB08(1)∧EAE(1) = $\overline{\text{EAE OR ARO}}$ SU3(1)∧MB16(1) = EAE or MQO	KE3
	(ACI, EAE-R, CONT, CMA40)	KC18
	CM STROBE∧EAE OR MQO = MQO(1) MQO(1) = MQ00-17 → A BUS00-17 A BUS00-17 → ADR00-17 NOSH = ADR00-17 → O BUS00-17 ACI(1) = O BUS00-17 → AC00-17 EAE-R(1) = O BUS L → TEMP2 CONT(1)∧CM STROBE = GO TO 40	KC19 KC20 KC21 KC20 KC20 KE3 KC16
40	Same as OSC	
10	Same as OSC	

Table 3-7
ABS Functions

644000 Get Absolute Value of AC

Process	Function	Drawing No.
75	Same as OSC plus:	
	If AC00 = 1, then SU1(1)∧MB06(1)∧MB07(0)∧AC00(1) = CMPL CMPL = $\overline{\text{ADR00-17}}$ → O BUS00-17	KE3 KC20
43	Same as OSC	
41	Same as OSC	

Table 3-7 (cont)
ABS Functions

644000 Get Absolute Value of AC

Process	Function	Drawing No.
54	Same as OSC except: MB17(0) = $\overline{SCO}$	
40	Same as OSC	
10	Same as OSC	

Table 3-8
CLQ Functions

650000 Clear the MQ

Process	Function	Drawing No.
75	Same as OSC except: MB05(1) = $\overline{EAE\ OR\ MQO}$	
43	Same as OSC except: CM STROBE∧$\overline{EAE\ OR\ MQO}$ = MQO(0) MQO(0) = 0 → A BUS00-17	
41	Same as OSC	
54	Same as OSC except: MB17(0) = $\overline{SCO}$	
40	Same as OSC	
10	Same as OSC	

Table 3-9
LMQ Functions

652000 Load the MQ with the AC

Process	Function	Drawing No.
75	Same as OSC except: MB05(1) = $\overline{EAE\ OR\ MQO}$ MB07(1) = EAE OR ARO	 KE3
43	(ACI, EAE, CONT, CMA41)	KC18
	CM STROBE∧EAE OR ARO = ARO(1)	KC19
	ARO(1) = AR00-17 → A BUS00-17	KC20
	A BUS00-17 → ADR00-17	KC21
	NOSH = ADR00-17 → O BUS00-17	KC20
	ACI(1) = O BUS00-17 → AC00-17	KC20
	LI(0) = LAR → LINK	KC15
	CM STROBE∧CONT(1) = GO TO 41	KC16

Table 3-9 (cont)
LMQ Functions

652000 Load the MQ with the AC

Process	Function	Drawing No.
41	Same as OSC	
54	Same as OSC except: MB17(0) = $\overline{SCO}$	
40	Same as OSC	
10	Same as OSC	

Table 3-10
GSM Functions

664000 Get Sign and Magnitude of AC

Process	Function	Drawing No.
75	Same as OSC except: If AC00 = 1, then SU1(1)∧MB06(1)∧MB07(0)∧AC00(1) = CMPL CMPL = $\overline{ADR00\text{-}17}$ → O BUS00-17 SU1(1)∧MB04(1)∧AC00(1) = A BUS LINK A BUS LINK = ADRL $\overline{SHIFT}$ = ADRL → O BUS L LI(1) = O BUS L → LAR(1)	 KE3 KC20 KE3 KC15 KC15 KC15
43	Same as OSC	
41	Same as OSC	
54	Same as OSC except: MB17(0) = $\overline{SCO}$	
40	Same as OSC	
10	Same as OSC	

3.5 SHIFT INSTRUCTIONS

Long left, long right, and accumulator-left shift instructions include a step count in bits 12 through 17 which commands the number of bit positions to be shifted. Preliminary operations governed by the early shift entry process words transfer the 2s complement of the step count into the step counter SC12 through 17 in the EAE logic, drawing KE2. The SC, then, becomes binary up-counter which steps toward 0 with each shift process. When the SC reaches 0, it sets a pair of overflow flip-flops SCOV and SCOV2, in turn, which shut off the shift processes and cause the computer to branch to the BGN next fetch process word.

The data to be shifted may be signed or unsigned. For signed data shifts, an early process word (43) transfers the sign (AC00) into the LINK, and the LINK is shifted thereafter unchanged. For unsigned data shifts, the LINK is usually initialized to 0 and shifted thereafter unchanged. Table 3-11 shows the SHIFT instruction format. Bit 04 of the instruction commands the signed or unsigned operation.

Table 3-11
EAE Shift Instruction Format

Op Code 64_8			Shift Code			Commands Number of Shifts						
0 1 2	3 4 5	6 7 8	9 10 11			12 13 14			15 16 17			
6	4	0*	5			X			X			LRS
6	6	0*	5			X			X			LRSS
6	4	0*	6			X			X			LLS
6	6	0*	6			X			X			LLSS
6	4	0*	7			X			X			ALS
6	6	0*	7			X			X			ALSS

*May be used for same functions as EAE SETUP.

Bits 12 through 17 can contain step codes of up to 44_8 for long register shifts of up to 36 bit positions. For accumulator left shifts (ALS, ALSS) bits 12 through 17 can contain step codes of up to 22_8 for AC left shifts of up to 18 bit positions.

Table 3-12 through 3-14 and Figures 3-2 through 3-4 illustrate the operations involved for LRSS, LLSS, and ALSS instructions calling for one, two, and three shift steps, respectively. A comparison of the three reveals the pattern for shifting the data and terminating the instruction.

While the NOSH level generated on drawing KC13 commands direct bit-for-bit transfers between registers, the shift operations make use of the SHL1 and SHR1 levels on the same drawing to shift a bit one position left or right into the receiving register. Register input/output gating and data flow is as usual from output register to A bus to ADR to O bus to input register. These functions are abbreviated in the tables for convenience.

of notes from class

Table 3-12
LRSS Functions

660501 Long Right Shift Signed (One Position)

Process	Function	Drawing No.
75	(ACO,ARI,EAE,LI,CONT,CMA43)	KC18
	ACO(1)∧ARI(1)∧NOSH = AC → AR	KC20-21
	SA09(1)∧SA10(0)∧SA11(1) = LRS	KE4
	EAE(1)∧ARI(1) = SU1(1)	KE3
	SU1(1) = 0 → SCOV,SCOV2,FIRST,EAE RUN,EAE SIGN,MQ SIGN	KE2-3
	SU1(1)∧$\overline{\text{SETUP}}$ = SC CLR	KE2
	SC CLR = 0 → SC	KE2
	SU1(1)∧MB05(0) = EAE OR MQO	KE3
	If AC00 = 1, then SU1(1)∧MB04(1)∧AC00(1) = A BUS LINK	KE3
	A BUS LINK → ADRL	KC15
	LI(1) = ADRL → LAR	KC15
	LI(1) = ADRL → TEMP3	KE3
	CM STROBE∧CONT(1) = GO TO 43	KC16
43	(ACI,EAE,CONT,CMA41)	KC18
	CM STROBE∧EAE OR MQO = MQO(1)	KC19
	MQO(1)∧NOSH∧ACI(1) = MQ → AC	KC20-21
	EAE(1)∧ACI(1)∧$\overline{\text{SETUP}}$ = SU2(1)	KE3
1s Comp to SC	SU2(1) = $\overline{\text{MB12-17}}$ = 111110 → SC (one's complement of 01_8)	KE2
	LI(0) = LAR → LINK	KC15
	CM STROBE∧CONT(1) = GO TO 41	KC16
41	(ACO,MQI,EAE,CONT,CMA54)	KC18
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	EAE(1)∧MQI(1)∧MB08(0) = EAE OR ARO	KE3
	CM STROBE∧CONT(1) = GO TO 54	KC16
54	(ACI,EAE-R,CONT,CMA40)	KC18
	CM STROBE∧EAE OR ARO = ARO(1)	KC19
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
2s Comp to SC	R-PULSE = 111111 → SC = SC FULL	KE2
	EAE-R(1)∧SCOV2(0) = ADDR 10	KE3
	EAE-R(1) = O BUS L = LINK → TEMP2 (not used)	KE3
	CMA40∧ADDR 10 = CMA50	KC17
	CM STROBE∧CONT(1) = GO TO 50	KC16
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	EAE-P(1)∧EAE RUN(0) = FIRST(1)	KE3
	EAE-P(1)∧SCOV2(0) = EAE RUN(1)	KE3
	EAE-P(1) = O BUS L = LINK → TEMP1 (not used)	KE3
	EAE-P(1) = TEMP2 = LINK → END BIT00 (not used)	KC15
	EAE-P(1) = TEMP3 = LINK → END BIT17 (not used)	KC15
	CM STROBE∧CONT(1) = GO TO 42	KC16

Table 3-12 (cont)
LRSS Functions

Process	Function	Drawing No.
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 000000 → SC	KE2
	EAE-R(1)∧SC FULL = SCOV(1)	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
Shift 1	ACO(1)∧SHR1∧MQI(1) = ACn → MQ(n+1)	KC20-21
	SHR1 = ADR17 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2	KE3
	EAE-R(1) = ADRL → END BIT00	KC15
	EAE-R(1) = TEMP1 = LINK → END BIT17 (not used)	KC15
	MQI(1)∧SHR1 = END BIT00 → MQ00	KC20
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53	KC18
	EAE-P(1)∧EAE RUN(1) = FIRST(0)	KE3
	FIRST(0)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
	ARO(1)∧SHR1∧ACI(1) = ARn → ACn+1	KC20-21
	SHR1 = ADR17 → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1 (not used)	KE3
	EAE-P(1) = TEMP2 → END BIT00	KC15
	EAE-P(1) = TEMP3 → END BIT17 (not used)	KC15
	SHR1 = END BIT00 → AC00	KC20
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	EAE-R(1)∧SCOV(1) = SCOV2(1)	KE2
	SCOV2(1) = $\overline{\text{IN SHR1}}$	KE4
	SCOV(1) = $\overline{\text{R-PULSE}}$	KE2
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 56	
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	EAE-R(1)∧SCOV2(1) = $\overline{\text{EAE RUN(0)}}$	KE3
	EAE RUN(0)∧SCOV2(1) = $\overline{\text{ADDR 10}}$	KE3
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	CM STROBE CONT(1) = GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	~~CLK(B)+670 ns∧EAE(1)∧DONE(1)~~ = INPUT IO RESTART	KD3(3)
	INPUT IO RESTART = IO RESTART	KD3(3)
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

NOTE

CML 42 Set SCOV, CML 53 Set SCOV2, and CML 57 reset EAE RUN which inhibited the generation of ADDR 10. If the shift process has not reset EAE RUN when CML 40 is pointed to, it will go back through CML's 50, 42, 55, 53, 56, 57, and then to 40.

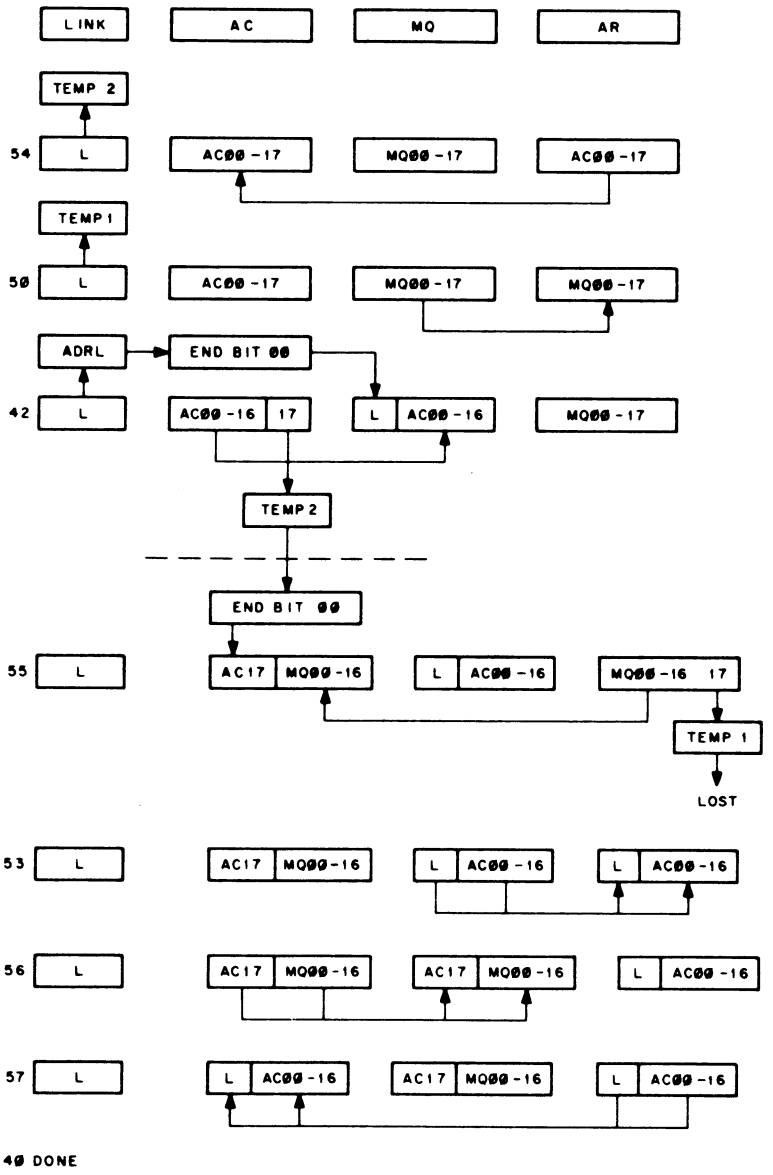


Figure 3-2 LRS, LRSS Register Manipulation (One Position)

Table 3-13
LLSS Functions

660602 Long Left Shift, Signed (Two Positions)

Process	Function	Drawing No.
75	Same as LRSS except:	
	SA09(1)∧SA10(1)∧SA11(0) = LLS	KE4
43	Same as LRSS except:	
	SU2(1) = 111101→SC	
41	Same as LRSS	
54	Same as LRSS except:	
	R-PULSE= 111110→SC	
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	EAE-P(1)∧EAE RUN(0) = FIRST(1)	KE3
	EAE-P(1)∧SCOV2(0) = EAE RUN(1)	KE3
	EAE-P(1)∧SCOV(0)∧EIR09(1)∧EIR11(0)= IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	MQO(1)∧SHL1∧ARI(1)= MQn→ARn-1	KC20-21
Shift 1	SHL1 = ADR00 →O BUS L	KC15
	EAE-P(1) = O BUS L→TEMP1	KE3
	EAE-P(1) = TEMP2→END BIT00	KC15
	EAE-P(1) = TEMP3→END BIT17	KC15
	SHL1= END BIT17→AR17	KC20
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV(0)= R-PULSE	KE2
	R-PULSE = 111111→SC = SC FULL	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR09(1)∧$\overline{\text{LRS}}$ = IN SHL1	KE4
	IN SHL1= SHL1	KC13
	ACO(1)∧SHL1∧MQI(1)= ACn→MQn-1	KC20-21
	SHL1 = ADR00→O BUS L	KC15
	EAE-R(1)= O BUS L→TEMP2 (lost)	KE3
	EAE-R(1)= TEMP1→END BIT17	KC15
	SHL1= END BIT17→MQ17	KC20
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	EAE-P(1)∧EAE RUN(1)= FIRST(0)	KE3
	EAE-P(1)∧SCOV(0)∧EIR09(1)∧EIR11(0) = IN SHL1	KE4
	IN SHL1= SHL1	KC13
	ARO(1)∧SHL1∧ACI(1)= ARn→ACn-1	KC20
Shift 2	SHL1= ADR00→O BUS L	KC15
	EAE-P(1)= O BUS L→TEMP1	KE3
	EAE-P(1)= TEMP2→END BIT00 (lost)	KC15
	EAE-P(1)= TEMP3→END BIT17	KC15
	SHL1= END BIT17→AC17	KC20
	CM STROBE∧CONT(1)= GO TO 53	

Table 3-13 (cont)
LLSS Functions

660602 Long Left Shift, Signed (Two Positions)

Process	Function	Drawing No.
53	(MQO, ARI, EAE-R, CONT, CMA56)	KC18
↑	EAE-R(1) ∧ SCOV(0) = R-PULSE	KE2
	R-PULSE = 000000 → SC	KE2
	R-PULSE ∧ SC FULL = SCOV(1)	KE2
	EAE-R(1) ∧ SCOV2(0) ∧ EAE RUN(1) ∧ EIR09(1) ∧ $\overline{\text{LRS}}$ = IN SHL1	KE4
	MQO(1) ∧ SHL1 ∧ ARI(1) = MQn → ARn-1	KC20
Shift 2	SHL1 = ADR00 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2 (lost)	KE3
	EAE-R(1) = TEMP1 → END BIT17	KC15
	SHL1 = END BIT17 → AR17	KC20
↓	CM STROBE ∧ CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	EAE-R(1)∧SCOV(1) = SCOV2(1)	KE2
	SCOV(1) = $\overline{\text{R-PULSE}}$	KE2
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	ARO(1)∧NOSH∧ACI = AR → AC	KC20-21
	EAE-R(1)∧EAE RUN(1) = ADDR 10	KE3
	CMA40∧ADDR 10 = CMA50	KC17
	CM STROBE∧CONT(1) = GO TO 50	KC16
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV2(1) = EAE RUN(0)	KE3
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 56	KC16

Table 3-13 (cont)
LLSS Functions

660602 Long Left Shift, Signed (Two Positions)

Process	Function	Drawing No.
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EAE RUN(0)∧SCOV2(1) = $\overline{\text{ADDR 10}}$	KE3
	CM STROBE∧CONT(1) = GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	~~CLK(B)+670 ns∧EAE(1)∧DONE(1)~~ = INPUT IO RESTART	KD3(3)
	INPUT IO RESTART = IO RESTART	KD3(3)
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

Table 3-14
ALSS Functions

660703 Accumulator Left Shift Signed (Three Positions)

Process	Function	Drawing No.
75	Same as LRSS except:	
	SA09(1)∧SA10(1)∧SA11(1) = ALS	KE4
43	Same as LRSS except:	
	SU2(1) = 111100 → SC	KE2
41	Same as LRSS	
54	Same as LRSS except:	
	R-PULSE = 111101 → SC	KE2
50	Same as LRSS	
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111110 → SC	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR09(1)∧$\overline{\text{LRS}}$ = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	ACO(1)∧SHL1∧MQI(1) = ACn → MQn-1	KC20-21
	SHL1 = ADR00 → O BUS L	KC15

Table 3-14 (cont)
ALSS Functions

660703 Accumulator Left Shift, Signed (Three Positions)

Process	Function	Drawing No.
42(cont)	EAE-R(1) = O BUS L → TEMP2	KE3
	EAE-R(1) = TEMP1 → END BIT17	KC15
	SHL1 = END BIT17 → MQ17	KC20
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	EAE-P(1)∧EAE RUN(1) = FIRST(0)	KE3
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EIR11(1) = $\overline{\text{IN SHL1}}$	KE4
	$\overline{\text{SHIFT}}$ = ADRL → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1	KE3
	EAE-P(1) = TEMP2 → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3 → END BIT17 (not used)	KC15
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KE18
↑	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111111 → SC = SC FULL	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR09(1)∧$\overline{\text{LRS}}$ = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	MQO(1)∧SHL1∧ARI(1) = MQn → ARn-1	KC20-21
Shift 2	SHL1 = ADR00 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2	KE3
	EAE-R(1) = TEMP1 → END BIT17	KC15
	SHL1 = END BIT17 → AR17	KC20
↓	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	EIR11(1) = $\overline{\text{IN SHL1}}$	KE4
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	$\overline{\text{SHIFT}}$ = ADRL → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1	KE3
	EAE-P(1) = TEMP2 → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3 → END BIT17 (not used)	KC15
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
↑	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 000000 → SC	KE2
	R-PULSE∧SC FULL = SCOV(1)	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR09(1)∧$\overline{\text{LRS}}$ = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	ARO(1)∧SHL1∧ACI(1) = ARn → ACn-1	KC20-21
Shift 3	SHL1 = ADR00 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2 (lost)	KE3
	EAE-R(1) = TEMP1 → END BIT17	KC15
↓		

Table 3-14 (cont)
ALSS Functions

660703 Accumulator Left Shift, Signed (Three Positions)

Process	Function	Drawing No.
57(cont) ↓	SHL1 = END BIT17 → AC17	KC20
	EAE-R(1)∧EAE RUN(1) = ADDR 10	KE3
	CMA40∧ADDR 10 = CMA50	KC17
	CM STROBE∧CONT(1) = GO TO 50	KC16
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV(1) = SCOV2(1)	KE2
	SCOV(1) = $\overline{\text{R-PULSE}}$	KE2
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	EAE-R(1)∧SCOV2(1) = EAE RUN(0)	KE3
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	SCOV2(1) = $\overline{\text{IN SHL1}}$	KE4
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EAE RUN(0)∧SCOV2(1) = $\overline{\text{ADDR 10}}$	KE3
	CM STROBE∧CONT(1) = GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	~~CLK(B)+670 ns∧EAE(1)∧DONE(1)~~ = INPUT IO RESTART	KD3(3)
	INPUT IO RESTART = IO RESTART	KD3(3)
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

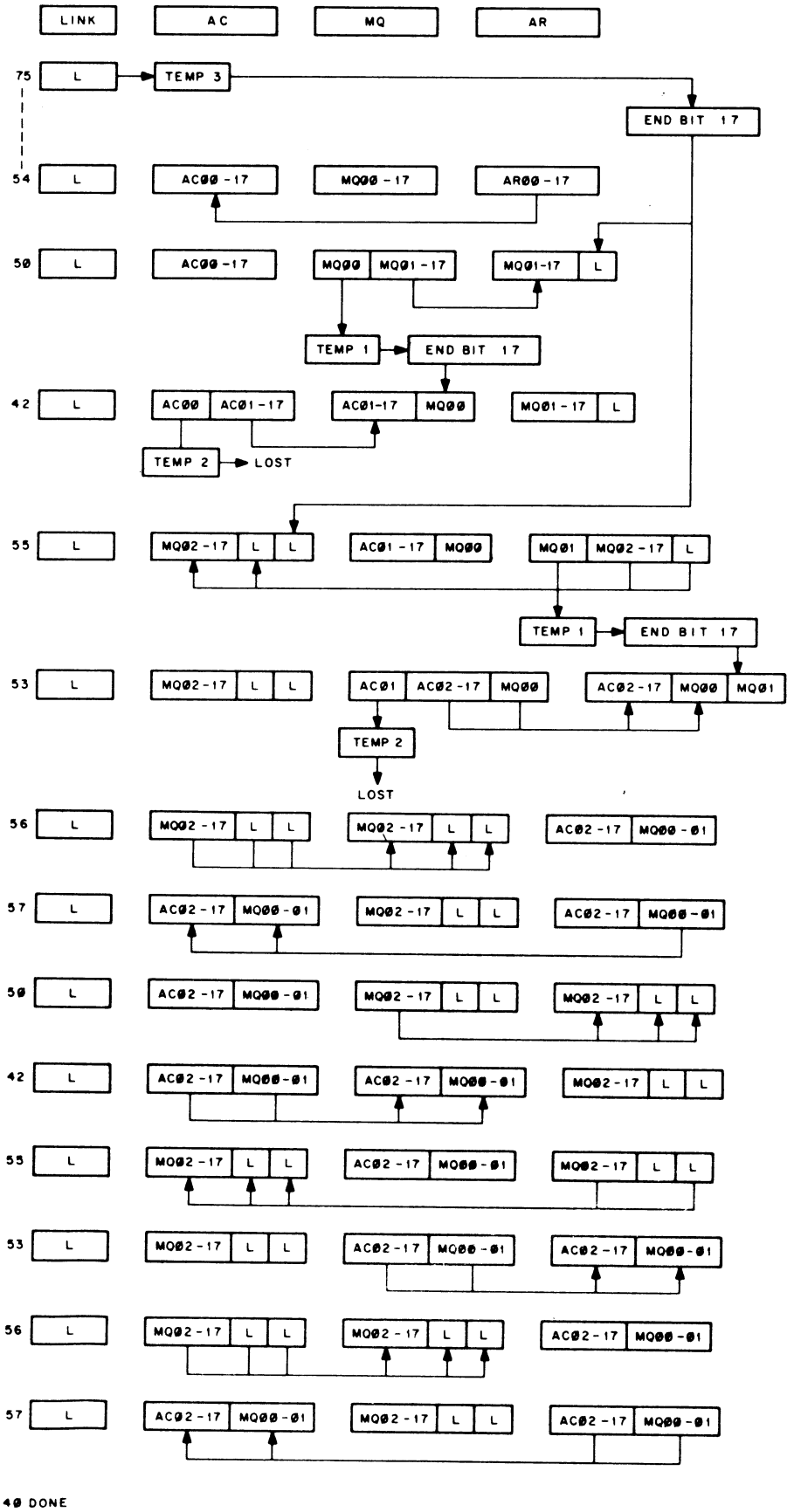


Figure 3-3 LLS, LLSS Register Manipulation (Two Positions)

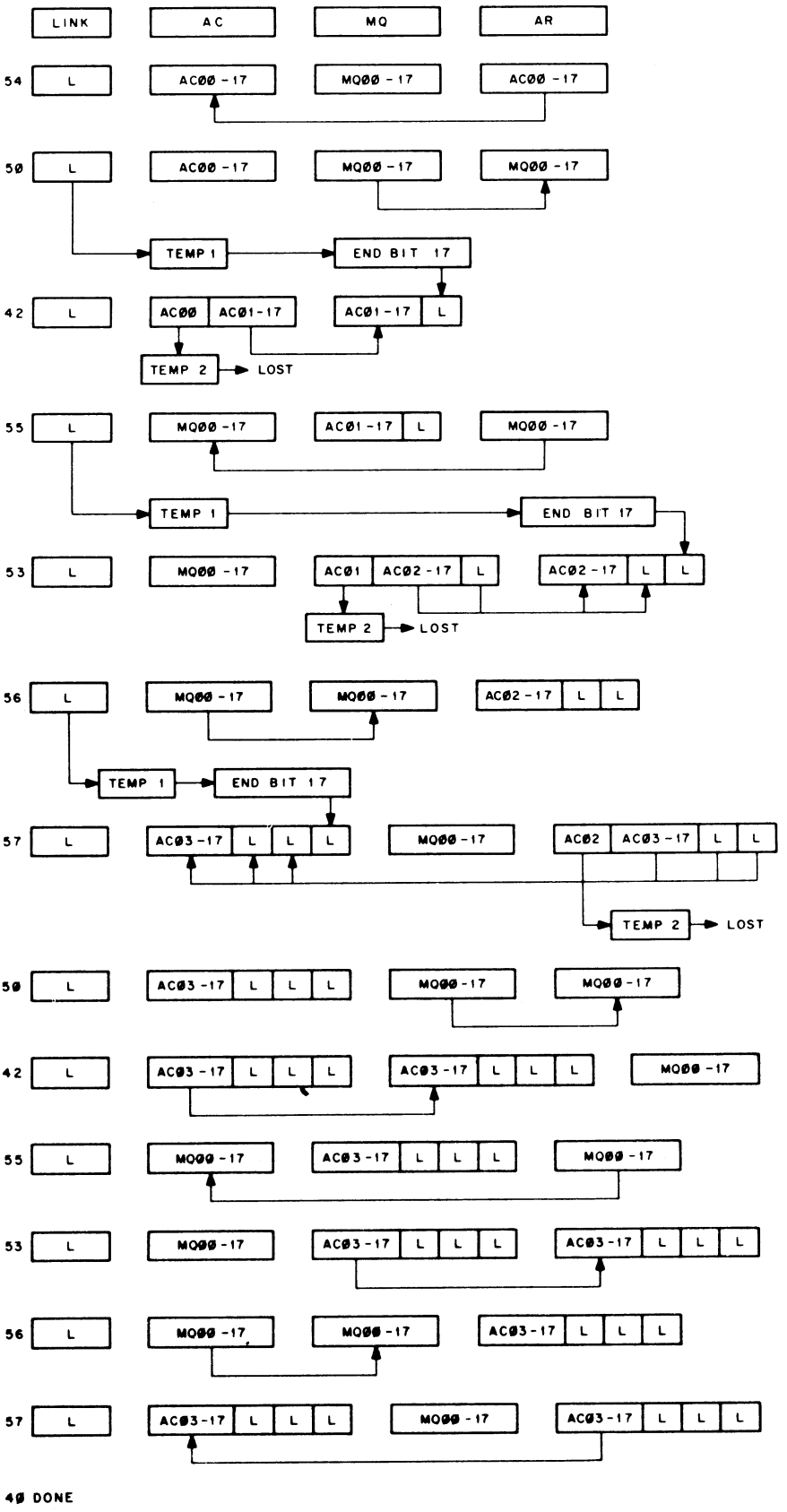


Figure 3-4 ALS, ALSS Register Manipulation (Three Positions)

3.6 NORMALIZE INSTRUCTIONS

The NORM and NORMS instructions, Table 3-15, are commonly used within a subroutine to convert an integer into a fraction and exponent for use in floating-point arithmetic. The algorithm for normalize is to shift the contents of the AC and MQ left until AC00 differs with AC01. For signed, normalized positive numbers this results in AC00(0) and AC01(1). For signed, normalized negative numbers the result is AC00(1) and AC01(0). For signed normalized numbers the sign (AC00) is first duplicated in the LINK. For unsigned numbers the LINK is usually initialized to 0. In both cases the content of MQ00 enters AC17, the content shifted out of AC00 is lost, and the content of the LINK enters MQ17, on each shift. When shifting halts, the contents of the SC reflect the number of shifts executed to reach the normalized condition. The SC contents are available through the use of the EAE OSC or EAE LACS instruction.

Table 3-15
EAE NORM Instruction Format

Op Code 64_8			Not Used			NORM 4_8			Number of Shifts						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
6			4			0			4			4			4			NORM
6			6			0			4			4			4			NORMS

For normalized numbers, the binary point is assumed to be between AC00 and AC01, the mantissa of the fraction extends from AC01 to MQ17, the sign is in AC00, and the value of the exponent is in the SC. The number in the SC after normalize is actually the sum of the pre-established characteristic and the exponent (n) in 2s complement form. The characteristic is a number equivalent to the total number of bit positions in the AC and MQ, 36_{10} or 44_8. The NORM(S) instruction contains this number in bits 12 through 17 and loads it into the SC in 2s complement to establish the exponent in excess 44 code. This means that the exponential range of the fraction when normalized is 2^0 to 2^{35}, or $-44_8 + n$.

For example, if the integer +3 is stored in the MQ (MQ16, MQ17 are 1s) and it is desired to convert this to a fraction and exponent, the following program sequence is required.

```
NORM(S)     /NORMALIZE CONTENTS OF AC, MQ
DAC         /DEPOSIT AC IN MEMORY
LACQ        /MOVE MQ TO AC
DAC         /DEPOSIT MQ IN MEMORY
LACS        /MOVE SC TO AC
TAD (44     /SUBTRACT CHARACTERISTIC FROM STEP COUNT
DAC         /DEPOSIT RESULT (EXPONENT) IN MEMORY
```

In the process of normalizing, a total of 33 shifts is required to shift MQ16(1) into AC01. This leaves the SC with a step count of:

011100	initialized step count
100001	plus 33 steps
111101	final step count

Since the step count is in 2s complement, the TAD (44_8 instruction (2s complement add) in effect subtracts the characteristic from the final step count to arrive at the exponent:

111101	final step count
100100	TAD characteristic
100001	exponent

The NORM(S) logic functions are very similar to the LLS(S) functions. Table 3-13 lists the functions for a two-position LLSS instruction. The functions for a NORMS instruction requiring only two shifts to normalize can be correlated with those of Table 3-13.

In the NORMS case, any positive integer whose most-significant 1 bit is located in AC03 requires two shifts to normalize. Likewise, any negative integer whose most-significant 0 bit is in AC03 requires two shifts to normalize. Substituting the positive-integer NORMS case in the listings of Table 3-13, the following NORMS functions become apparent.

75	SA09(1)∧SA10(0)∧SA11(0) = NORM	KE4
43	SU2(1) = 011011 → SC	KE2
41	Same	
54	R-PULSE = 011100 → SC	KE2
50	Same, first shift	
42	Same, first shift, plus:	
	R-PULSE = 011101 → SC	
	EAE STROBE DLYD∧EAE-R(1)∧NORM∧O BUS00∧O BUS01 = $\overline{\text{SCOV(1)}}$	KE2
55	Same, second shift	
53	Same, second shift, plus:	
	R-PULSE = 011110 → SC	KE2
	EAE STROBE DLYD∧EAE-R(1)∧NORM∧OBUS00∧$\overline{\text{O BUS01}}$ = SCOV(1)	KE2
56,57,50,42,55,53,56,57,40,10	Same	

Although the execution of a NORM(S) instruction cannot be interrupted by a program interrupt (PI) or an automatic priority interrupt (API) request, the central processor can grant such a request before the executed NORM(S) results can be extracted from the EAE registers and processed. Therefore, if interrupt-accessed subroutines are to make use of the EAE, the following instruction sequences are suggested to preserve the register contents during the interrupt and to restore them to the EAE upon completion of the interrupt service routine.

```
/SAVE EAE REGISTERS DURING INTERRUPT
                JMS SUBENTR
SUBENTR,          0
                DAC ACSAVE          /SAVE AC CONTENTS
                LACQ                /MOVE MQ TO AC
                DAC MQSAVE          /SAVE MQ CONTENTS
                LACS                /MOVE SC TO AC
                DAC SCSAVE          /SAVE SC CONTENTS
                    .
                    .
                    .
                LAC SCSAVE
                XOR (77             /COMPLEMENT STEP COUNT
                TAD (640402         /DEVELOP PSEUDO NORM
                AND (640477         /DELETE POSSIBLE STEP COUNT OVERFLOW
                DAC.+1              /PLACE NORM IN SEQUENCE
                HLT*                /STEP COUNT TO SC
                LAC MQSAVE          /
                LMQ                 /LOAD THE MQ
                LAC ACSAVE          /LOAD THE AC
                DBR                 /RESTORE PC,LINK,ETC
                JMP I SUBENTR
```

Restoration of the step count to the SC requires that the 2s complemented quantity, taken from the SC at the time of interrupt, be complemented, then combined with the pseudo NORM instruction. The step count following TAD,AND is one less (1s complement) than the actual value produced by the previous normalization (2s complement). Execution of the pseudo NORM instruction, then, 2s complements this step count into the SC, and in shifting the AC and MQ left one bit position adds the necessary 1 to the SC to produce the correctly restored step count (the 6404XX present in the AC from TAD, AND shifts to become 501XXX). From the previous two-shift NORM(S) sample:

```
                011110      LAC ACSAVE
                111111      XOR (77
                100001
6404(8)         000010      TAD (640402
                100011
6404(8)         111111      AND (640477
6404(8)         100011      DEPOSIT IN HLT* = 640443 = NORM
NORM =          011100      1s complement → SC
                011101      2s complement → SC
                011110      shift once, step SC
```

The DBR instruction preceding the JMP I subroutine termination primes the computer for restoration of the interrupted program. This restoration occurs during JMP I. During this time, the PC and

*Good programming practices dictate that instructions to be developed at "run" time be represented by HLT instructions in the source program. If the development does not occur, the HLT will facilitate debugging the program.

LINK are restored to the contents existing at the time of interrupt. The memory protect and extended memory options, if in the system, are restored to their on or off status. Refer to the PDP-9 Maintenance Manual and option manuals for details.

3.7 MULTIPLY INSTRUCTIONS

The MUL(S) instruction, Table 3-16, multiplies the contents of the AC (multiplier) by the contents of the next sequential core memory location (multiplicand) to form a product in the AC and MQ. Bits 12 through 17 in the instruction are usually programmed for a step count of 22_8 (18_{10}), representing the multiplication of one 18-bit quantity (sign bit and 17 magnitude bits for MULS) by another to produce a 36-bit product. When such precision is not required, the microprogrammed step count can be decreased by subtracting the appropriate number "n" from the instruction code. The product is always scaled 18-n from MQ17. If "n" is programmed in the instruction, the 18-n lower order bits in the long register are meaningless.

Table 3-16
EAE MUL Instruction Format

Op Code 64_8			MUL 1_8	Commands Product Precision						
0 1 2	3 4 5	6 7 8	9 10 11	12 13 14			15 16 17			
6	5	3	1	X			X			MUL
6	5	7	1	X			X			MULS

For a MUL instruction the LINK must previously have been initialized to 0 and remains 0. During the preparatory phase the multiplier is transferred from the AC to the MQ, the AC is cleared, and the SC is set to the 2s complement of the step count in bits 12 through 17 of the instruction. A core memory cycle takes place to read the multiplicand into the MB. The arithmetic phase, executed as multiplication of one unsigned quantity by another (binary point of no consequence), halts when the SC counts up to 0.

For a MULS instruction a previous LAC/GSM/DAC CAND sequence stores the absolute value of the multiplicand in memory and places the original sign of the multiplicand in the LINK. During the preparatory phase of MULS, a core memory cycle reads the absolute value multiplicand into the MB, transfers the LINK content to a TEMPorary storage flip-flop in the EAE, and resets the LINK. The multiplier is transferred to the MQ and is 1s complemented if negative, the AC is cleared to 0, and the SC is initialized to the 2s complement of the step count in bits 12 through 17 of the instruction. The arithmetic phase, executed as multiplication of one signed quantity by another (sign bit plus 17 magnitude

bits, binary point of no consequence), halts when the SC counts up to 0. Bits AC00 and AC01 each receive the sign of the product; the remaining AC and MQ bits represent the magnitude.

From the above description of MULS, it can be seen that the arithmetic phase always starts with positive, like-signed quantities in the MQ (multiplier) and the MB (multiplicand). The TEMPorary storage flip-flop which receives the original sign of the multiplicand (TEMP3, drawing KE3) acts upon the MQ SIGN and EAE SIGN flip-flops which perform certain complementary functions during the arithmetic phase to arrive at the correctly signed product.

Thus, the complementary functions govern the four signed multiply situations as follows.

+ x + = +	(behaves as simple unsigned multiply, no complementing of the final product)
+ x - = -	(negative multiplier is first complemented in preparatory phase, final product complemented after arithmetic phase)
- x + = -	(EAE GSM sets LINK, complements multiplicand; MULS complements final product after arithmetic phase)
- x - = +	(EAE GSM sets LINK, complements multiplicand; MULS complements multiplier in preparatory phase; no complementing of final product)

The algorithm for multiplication using the EAE is sample, add, and shift right. Each bit of the multiplier is sampled, starting with the least significant bit. If the sampled bit is a 1, the multiplicand is added to the partial product. The partial product and the multiplier are then shifted right one position for the next multiplier bit sampling. If the sampled bit is a 0, zeros are added to the partial product. With each shift the content of the least significant bit is lost. Multiplication ends when the SC, up-counted with each shift, reaches 0.

A sample program for signed multiplication of two positive numbers, $2_8 \times 5_8$ follows. The logic functions that perform the MULS operations are tabulated in Table 3-17. Table 3-18 is a listing of the arithmetic operations by process word functions.* The sample program and the microprogrammed bits 12 through 17 in the MULS instruction reflect an initial step count of 04_8, resulting in a product precision of eight bits. The MULS instruction is used here to explain EAE SIGN operations; actually, the sample program can be modified for MUL by eliminating the GSM sequence if dealing with unsigned numbers. Tables 3-19, 3-20, and 3-21 list the ramifications of Table 3-17 for different sign situations.

```
/MULTIPLY 2₈ x 5₈
ST,     0200    200100    LAC CAND      /LOAD MULTIPLICAND INTO AC
        0201    100500    JMS MPY       /STORE MAIN PROGRAM ADDRESS IN 0500
                                        /AND JUMP TO MPY SUBROUTINE
        0202    200101    LAC PLIER     /LOAD MULTIPLIER INTO AC
        0203                            /MAIN PROGRAM RE-ENTRY
```

*Table 3-18 utilizes 4-bit binary numbers for simplicity. The actual result obtained in multiplying $2_8 \times 5_8$ is 000000_8 in the AC and 500000_8 in the MQ. Fourteen more shifts to the right would align the answer as 12_8 ($MQ000012_8$).

```
MPY     0500    000202    PC          /MAIN PROGRAM ADDRESS
        0501    664000    GSM         /STORE CAND SIGN IN LINK AND
                                      /ABSOLUTE VALUE IN AC
        0502    040505    DAC .+3     /DEPOSIT CAND IN 0505
        0503    420500    XCT*MPY     /LOAD MULTIPLIER INTO AC
        0504    657122    MULS        /FETCH CAND AND MULTIPLY

        0505    000002    MPY
        0506    440500    ISZ         /INCREMENT MAIN PROGRAM ADDRESS
        0507    620500    JMP MPY     /JUMP TO MAIN PROGRAM

CAND    0100    000002    MULTIPLICAND
PLIER   0101    000005    MULTIPLIER
```

Table 3-17
MULS Functions

657104 Multiply, Signed (Four Steps)

$$\begin{array}{r} 0010 \\ \times\ 0101 \\ \hline 2_8 \times 5_8 \end{array}$$

Process	Function	Drawing No.
75	(ACO,ARI,EAE,LI,CONT,CMA43)	KC18
	ACO(1)∧NOSH∧ARI(1) = AC → AR	KC20-21
	SA09(0)∧SA10(0)∧SA11(1) = MUL	KE4
	EAE(1)∧ARI(1) = SU1(1)	KE3
	SU1(1) = 0 → SCOV,SCOV2,FIRST,EAE RUN,EAE SIGN,MQ SIGN	KE2-3
	SU1(1)∧$\overline{\text{SETUP}}$ = SC CLR	KE2
	SC CLR = 0 → SC	KE2
	SU1(1)∧MB07(1) = EAE OR ARO	KE3
	LI(1) = ADRL → LAR(0)	KC15
	LI(1) = ADRL → TEMP3(0)	KE3
	EAE(1) = 0 → EN CMPL	KE3
	TEMP3(0) = condition MQ SIGN MUL = condition MQ SIGN	KE3
	CM STROBE∧CONT(1) = GO TO 43	KC16
43	(ACI,EAE,CONT,CMA41)	KC18
	CM STROBE∧EAE OR ARO = ARO(1)	KC19
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EAE(1)∧ACI(1)∧$\overline{\text{SETUP}}$ = SU2(1)	KE3
	SU2(1) = $\overline{\text{MB12-17}}$ → SC = 111011	KE2
	LI(0) = LAR(0) → LINK(0)	KC15
	CM STROBE∧CONT(1) = GO TO 41	KC16
41	(ACO,MQI,EAE,CONT,CMA54)	KC18
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 54	KC16
54	(ACI,EAE-R,CONT,CMA40)	KC18
	ACI(1) = 0 → AC	KC20
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111100 → SC	KE2
	EAE-R(1) = O BUS L = LINK → TEMP2(0)	KE3
	EAE(0)∧TEMP3(0) = MQ SIGN(1)	KE3

Table 3-17 (cont)
MULS Functions

657104 — Multiply, Signed (Four Steps) — $2_8 \times 5_8$

Process	Function	Drawing No.
54(cont)	MQ SIGN(1) = condition EAE SIGN	KE3
	EAE-R(1)∧SCOV2(0) = ADDR 10	KE3
	EAE-R(1)∧EIR09(0)∧SCOV2(0)∧EAE RUN(0) = ODD ADDR	KE3
	CM STROBE∧CONT(1)∧CMA40∧ADDR 10∧ODD ADDR = GO TO 51	KC16
51	(PCO,SM,MBI,CMA52)	KC18
	PCO(1)∧NOSH∧MBI(1) = PC → MB (CAND ADDRESS)	KC20-21
	SM(1)∧CLK = FETCH CAND	MC2
	SM(1)∧CLK = CM STROBE	KC16
	CM STROBE = GO TO 52	KC17
52	(MBO,+1,PCI,LI,CMA50)	KC18
	+1(1) = CI17	KC14
	MBO(1)∧NOSH∧CI17∧PCI(1) = MB (CAND ADDRESS) +1 → PC	KC20-21
	+1(1) = $\overline{\text{A BUS LINK}}$ → $\overline{\text{ADRL}}$	KC15
	LI(1) = $\overline{\text{ADRL}}$ → LAR(0) LI(1) = $\overline{\text{ADRL}}$ → TEMP3(0)	KC15
	LI(1)∧$\overline{\text{CONT}}$ = EAE CLR RQ	KE3
	EAE CLR RQ = IN CLR, CLR	KC16
	IN CLR = CLR I = 0 → PCI, MBO	KC19
	CLR = 0 → +1, 1 → SAO	KC19
	IN CLR = 1 → MBI	KC19
	SAO(1) = $\overline{\text{A BUS LINK}}$ → $\overline{\text{ADRL}}$ (Since +1 is cleared by CLR, SAO(1) inhibits erroneous setting of LAR)	KC15
	SAO(1)∧NOSH∧MBI(1) = SA(CAND) → MB	KC20-21
	MEM STROBE = GO TO 50	KC16
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	EAE-P(1)∧EAE RUN(0) = FIRST(1)	KE3
	EAE-P(1)∧SCOV2(0) = EAE RUN(1)	KE3
	FIRST(1)∧EAE RUN(1)∧MQ SIGN(1)=CMPL EAE SIGN=EAE SIGN(1)	KE3
	FIRST(1)∧MUL = MQ SIGN (1)	KE3
	MQ SIGN(1) = condition EAE SIGN	
Sample	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	EAE-P(1)∧MUL∧SCOV(0)∧O BUS17(1) = EAE OR MBO	KE3
	EAE-P(1) = O BUS L = LINK = ADRL → TEMP1 (not used)	KE3
	LI(0) = LAR(0) → LINK(0)	KC15
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111101 → SC	KE2
	CM STROBE∧EAE OR MBO = MBO(1)	KC19
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
ADD, Shift 1	ACO(1)∧SHR1∧MQI(1) = ACn → MQn+1	KC20-21
	MBO(1)∧SHR1∧MQI(1) = MBn → MQn+1 AC+MB → MQ	KC20-21

Table 3-17 (cont)
MULS Functions

657104 — Multiply, Signed (Four Steps) — $2_8 \times 5_8$

Process	Function	Drawing No.
42 (cont)	EAE-R(1) = ADRL → END BIT00 (~~C00=0~~)	KC18
	SHR1 = END BIT00 → MQ00	KC20
	SHR1 = ADR17 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2	KE3
	EAE-R(1) = TEMP1 = LINK → END BIT17 (lost)	KC15
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
Shift 1, Sample	EAE-P(1)∧EAE RUN(1) = FIRST(0)	KE3
	EAE-P(1)∧FIRST(0)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
	ARO(1)∧SHR1∧ACI(1) = ARn → ACn+1	KC20-21
	EAE-P(1)∧MUL∧SCOV(0)∧O BUS17(0) = $\overline{\text{EAE OR MBO}}$	KE3
	SHR1 = ADR17 → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1 (lost)	KE3
	EAE-P(1) = TEMP2 → END BIT00	KC15
	SHR1 = END BIT00 → AC00	KC20
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
Shift 2, Add Zeros	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111110 → SC	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
	MQO(1)∧SHR1∧ARI(1) = MQn → ARn+1	KC20-21
	EAE-R(1) = ADRL → END BIT00	KC15
	SHR1 = END BIT00 → AR00	KC20
	SHR1 = ADR17 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2	KE3
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
Shift 2, Sample	EAE-P(1)∧FIRST(0)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
	ACO(1)∧SHR1∧MQI(1) = ACn → MQn+1	KC20-21
	EAE-P(1)∧MUL∧SCOV(0)∧O BUS17(1) = EAE OR MBO	KE3
	SHR1 = ADR17 → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1 (lost)	KE3
	EAE-P(1) = TEMP2 → END BIT00	KC15
	SHR1 = END BIT00 → MQ00	KC20
	CM STROBE∧CONT(1) = GO TO 57	KC16

Table 3-17 (cont)
MULS Functions

657104 — Multiply, Signed (Four Steps) — $2_8 \times 5_8$

Process	Function	Drawing No.
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111111 → SC = SC FULL	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = SHR1	KE4
	IN SHR1 = SHR1	KC13
	CM STROBE∧EAE OR MBO = MBO(1)	KC19
Add, Shift 3	ARO(1)∧SHR1∧ACI(1) = ARn → ACn+1	KC20-21
	MBO(1)∧SHR1∧ACI(1) = MBn → ACn+1	KC20-21
	EAE-R(1) = ADRL → END BIT00	KC15
	SHR1 = END BIT00 → AC00	KC20
	SHR1 = ADR17 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2	KE3
	EAE-R(1) = TEMP1 → END BIT17 (lost)	KC15
	EAE-R(1)∧SCOV2(0) = ADDR10	KE3
	CM STROBE∧CONT(1)∧CMA40∧ADDR 10 = GO TO 50	KC16
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	EAE-P(1)∧FIRST(0)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHRI = SHR1	KC13
	MQO(1)∧SHR1∧ARI(1) = MQn → ARn+1	KC20-21
Shift 3, Sample	EAE-P(1)∧MUL∧SCOV(0)∧O BUS17(0) = $\overline{\text{EAE OR MBO}}$	KE3
	SHR1 = ADR17 → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1 (lost)	KE3
	EAE-P(1) = TEMP2 → END BIT00	KC15
	SHR1 = END BIT00 → AR00	KC20
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 000000 → SC	KE2
	EAE-R(1)∧SC FULL = SCOV(1)	KE2
	EAE-R(1)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
Shift 4, Add Zeros	ACO(1)∧SHR1∧MQI(1) = ACn → MQn+1	KC20-21
	EAE-R(1) = ADRL → END BIT00	KC15
	SHR1 = END BIT00 → MQ00	KC20
	SHR1 = ADR17 → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2	KC15
	EAE-R(1) = TEMP1 → END BIT17 (lost)	KC15
	CM STROBE∧CONT(1) = GO TO 55	KC16

Table 3-17 (cont)
MULS Functions

657104 — Multiply, Signed (Four Steps) — $2_8 \times 5_8$

Process	Function	Drawing No.
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	EAE-P(1)∧FIRST(0)∧SCOV2(0)∧EAE RUN(1)∧EIR10(0)∧EIR11(1) = IN SHR1	KE4
	IN SHR1 = SHR1	KC13
Shift 4	ARO(1)∧SHR1∧ACI(1) = ARn → ACn+1	KC20-21
No Sample	EAE-P(1)∧MUL∧SCOV(1) = $\overline{\text{EAE OR MBO}}$	KE3
	SHR1 = ADR17 → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1 (lost)	KE3
	EAE-P(1) = TEMP2 → END BIT00	KC15
	SHR1 = END BIT00 → AC00	KC20
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	EAE-R(1)∧SCOV(1) = $\overline{\text{R-PULSE}}$	KE2
	EAE-R(1)∧SCOV(1) = SCOV2(1)	KE3
	SCOV2(1) = $\overline{\text{IN SHR1}}$	KE4
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	SCOV2(1) = $\overline{\text{IN SHR1}}$	KE4
	EAE-P(1)∧ACO(1)∧MQI(1)∧EIR09(0)∧SCOV2(1) = EN CMPL(1)	KE3
	EN CMPL(1)∧MUL∧MQ SIGN(1)=CMPL EAE SIGN=EAE SIGN(0)	KE3
	EAE SIGN(0) = $\overline{\text{CMPL}}$	KE3
	ACO(1)∧NOSH∧MQI(1)∧$\overline{\text{CMPL}}$ = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	SCOV2(1) = $\overline{\text{IN SHR1}}$ EAE	KE4
	EAE-R(1)∧SCOV2(1) = RUN(0)	KE3
	EAE-R(1)∧SCOV2(1)∧RUN (0) = $\overline{\text{ADDR 10}}$	KE3
	EN CMPL∧EAE SIGN (0) = $\overline{\text{CMPL}}$	KE3
	ARO(1)∧NOSH∧ACI(1)∧$\overline{\text{CMPL}}$= AR → AC	KC20-21
	CM STROBE∧CONT(1)∧$\overline{\text{ADDR 10}}$= GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	CLK(B) DLYD∧EAE(1)∧DONE(1) = INPUT IO RESTART	KD3
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

Table 3-18
MULS Arithmetic

$2_8 \times 5_8$

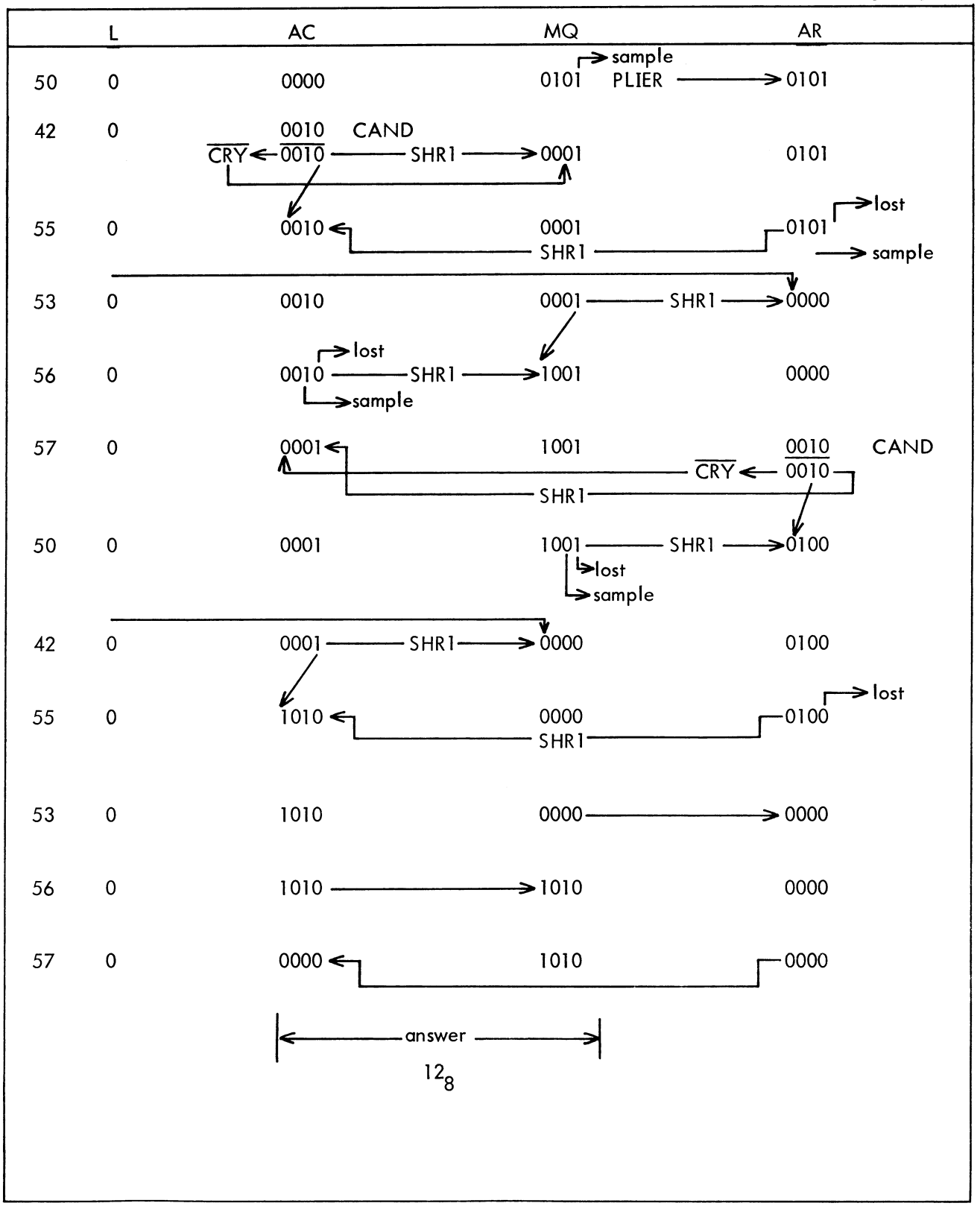


Table 3-19
MULS Functions

657104 Multiply, Signed (Four Steps) $2_8 - 5_8$

Process	Function
75	TEMP3(0) = condition MQ SIGN MUL = condition MQ SIGN AC00(1) = condition EAE SIGN EAE(1) = 0 → EN CMPL
43	SU2(1)∧MB06(1)∧AC00(1) = EAE SIGN (1) SU2(1)∧EIR09(0)∧EAE SIGN(1)∧EIR11(1) = CMPL CMPL = $\overline{AR}$ → AC
41	AC → MQ
54	EAE(0)∧TEMP3(0) = MQ SIGN(1) MQ SIGN(1) = condition EAE SIGN 0 → AC
51	CAND fetch
52	MB+1 → PC
50	FIRST(1)∧EAE RUN(1)∧MQ SIGN(1)∧EAE SIGN (1) = EAE SIGN(0) FIRST(1)∧MUL = MQ SIGN(1)
42,55,53	same as MULS $2_8 \times 5_8$
56	EAE-P(1)∧SCOV2(1)∧MQI(1)∧EIR09(0)∧ACO(1) = EN CMPL(1) MUL∧EN CMPL∧MQ SIGN(1)∧EAE SIGN (0) = EAE SIGN(1) EN CMPL(1)∧EAE SIGN(1) = CMPL CMPL = $\overline{AC}$ → MQ
57	EN CMPL∧EAE SIGN (1) = CMPL CMPL = $\overline{AR}$ → AC

Table 3-20
MULS Functions

657104 Multiply, Signed (Four Steps) $-2_8 \times 5_8$

Process	Function
75	TEMP3(1) = no conditioning of MQ SIGN AC00(0) = no conditioning of EAE SIGN MUL = condition MQ SIGN EAE(1) = 0 EN CMPL
43	AR → AC
41	AC → MQ
54	0 → AC
51	CAND fetch
52	MB+1 → PC
50	FIRST(1)∧MUL = MQ SIGN (1) FIRST(1)∧EAE RUN(1) = no effect on EAE SIGN

Table 3-20 (cont)
MULS Functions

657104 Multiply, Signed (Four Steps) $-2_8 \times 5_8$

Process	Function
42,55,53	same as MULS $2_8 \times 5_8$
56	EAE-P(1)∧SCOV2(1)∧MQI(1)∧EIR09(0)∧ACO(1) = EN CMPL(1) EN CMPL(1)∧MUL∧MQ SIGN(1)∧EAE SIGN (0) = EAE SIGN (1) EN CMPL(1)∧EAE SIGN (1) = CMPL CMPL = $\overline{AC}$ → MQ
57	EN CMPL∧EAE SIGN (1) = CMPL CMPL = $\overline{AR}$ → AC

Table 3-21
MULS Functions

657104 Multiply, Signed (Four Steps) $-2_8 \times -5_8$

Process	Function
75	TEMP3(1) = no conditioning of MQ SIGN AC00(1) = condition EAE SIGN MUL = condition MQ SIGN EAE(1) = 0 → EN CMPL
43	SU2(1)∧MB06(1)∧AC00(1) = EAE SIGN (1) SU2(1)∧EIR09(0)∧EAE SIGN(1)∧EIR11(1) = CMPL CMPL = $\overline{AR}$ → AC
41	AC → MQ
54	0 → AC
51	CAND fetch
52	MB+1 → PC
50	FIRST(1)∧MUL = MQ SIGN(1) FIRST(1)∧EAE RUN(1) = no effect on EAE SIGN
42,55,53	same as MULS $2_8 \times 5_8$
56	EAE-P(1)∧SCOV2(1)∧MQI(1)∧EIR09(0)∧ACO(1) = EN CMPL(1) EN CMPL(1)∧MUL∧MQ SIGN(1)∧EAE SIGN (1) = EAE SIGN (0) EN CMPL(1)∧EAE SIGN(0) = $\overline{CMPL}$ AC → MQ
57	EN CMPL(1)∧EAE SIGN(0) = $\overline{CMPL}$ AR → AC

3.8 DIVIDE INSTRUCTIONS

Six divide instructions including integer divide and fraction divide, Table 3-22, divide the contents of the AC and MQ (integer dividend, fraction dividend, long register dividend) by the contents

of the next sequential core memory location (divisor) to form a quotient in the MQ and remainder in the AC. Bits 12 through 17 in the instruction are usually programmed for a step count of 23_8 (19_{10}), representing division of a 36-bit dividend (actual or implied) by an 18-bit divisor. When such precision is not required, the microprogrammed step count can be decreased by subtracting the appropriate number "n" from the instruction code. The quotient is always right-justified in the MQ and the remainder right-justified in the AC. If "-n" is programmed in the instruction, the n high-order bits in the MQ and AC are meaningless.

Table 3-22
EAE DIV Instruction Format

Op Code 64_8			DIV 3_8			Commands Precision of QUOT/Remainder						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
6			4			0			3			X			X			DIV
6			4			4			3			X			X			DIVS
6			5			3			3			X			X			IDIV
6			5			7			3			X			X			IDIVS
6			5			0			3			X			X			FRDIV
6			5			4			3			X			X			FRDIVS

Instructions may be programmed for division of signed or unsigned quantities. Divide overflow occurs if the quotient exceeds the capacity of the MQ (777777_8, unsigned; $\pm 377777_8$, signed). The LINK sets to indicate an overflow, divide execution ends in 5 computer cycles, and the register contents are meaningless. The computer goes on to the next instruction.

3.8.1 DIV(S) Instruction

The DIV(S) instruction divides the contents of the AC and MQ (long register dividend) by the contents of the next sequential core memory location to form a quotient in the MQ and remainder in the AC.

For a DIV instruction the LINK must previously have been set to 0 and remains 0 unless divide overflow occurs (Section 3.8.4). During the preparatory phase, the SC is set to the 2s complement of the step count in bits 12 through 17 of the instruction. A core memory cycle takes place to read the divisor into the MB. The arithmetic phase, executed as the division of one unsigned quantity by another (binary point of no consequence), halts when the SC counts up to 0.

For a DIVS instruction, a previous LAC/GSM/DAC DIVR sequence stores the absolute value of the divisor in memory and places the original sign of the divisor in the LINK. During the preparatory phase of DIVS, a core memory cycle reads the absolute value divisor into the MB, transfers the LINK

content to the temporary storage register TEMP3 in the EAE, and resets the LINK. The SC is set to the 2s complement of the step count in bits 12 through 17 of the instruction. The arithmetic phase, executed as the division of one signed quantity by another (binary point of no consequence), halts when the SC counts up to 0. The dividend contains a double sign in bits AC00 and AC01. MQ00 receives the sign of the quotient, and AC00 receives the original sign of the dividend.

As with the execution of MULS, the arithmetic phase of DIVS starts with positive, like-signed quantities in the divisor and dividend. TEMP3, MQ SIGN, and EAE SIGN flip-flops act to 1s complement the MQ portion of a negative dividend during the preparatory phase and to perform other complementary functions during the arithmetic phase to arrive at the correctly signed quotient as follows.

+ ÷ + = +	(behaves as simple unsigned divide, final quotient complemented after arithmetic phase)
+ ÷ - = -	(EAE GSM sets LINK, complements divisor; final quotient not complemented)
- ÷ + = -	(MQ portion of dividend complemented during preparatory phase; quotient not complemented; remainder complemented after arithmetic phase)
- ÷ - = +	(EAE GSM sets LINK, complements divisor; MQ portion of dividend complemented during preparatory phase, quotient complemented after arithmetic phase).

The algorithm for divide using the EAE is sample, add or subtract, and shift left. The divisor is first subtracted from the AC portion of the dividend, and the result is shifted left. The LINK and TEMP3 receive the most significant bit of the result for sampling. If the result is a negative number, the divisor is added to the quotient; if the result is a positive number, the divisor is subtracted from the quotient. The result is then shifted left one position for the next sampling. If in the first subtraction the divisor is not greater than the AC portion of the dividend, divide overflow occurs, stopping divide operations (Section 3.8.4). The subtract operation takes the form of a 2s complement add.

Following is a sample program for the signed division of two positive numbers, $12_8 \div 5_8$. The logic functions that perform the DIVS operations are listed in Table 3-23. Table 3-24 is a listing of the arithmetic operations by process word functions. The sample program and the microprogrammed bits 12 through 17 in the DIVS instruction reflect an initial step count of 05_8, resulting in a four-bit precision of the quotient and remainder. The DIVS instruction is used here for purposes of explanation of the EAE SIGN operations; actually, the sample program can be modified for DIV by eliminating the GSM sequence if dealing with unsigned numbers. Tables 3-25, 3-26, and 3-27 list the ramifications of Table 3-23 for different sign situations.

/DIVIDE $12_8 \div 5_8$

```
ST,     0500    200100    LAC DIVR      /LOAD DIVISOR INTO AC
        0501    100200    JMS DIV       /STORE PROGRAM ADDRESS IN 0200 AND
                                        /JUMP TO DIV SUBROUTINE
        0502                            /MAIN PROGRAM RE-ENTRY
```

```
DIV     0200    000502    PC          /PROGRAM ADDRESS
        0201    664000    GSM         /STORE DIVR SIGN IN LINK AND ABSOLUTE
                                      /VALUE IN AC
        0202    040207    DAC. +5     /DEPOSIT DIVR IN 0207
        0203    200101    LAC DIVD1   /LOAD HALF DIVIDEND INTO AC
        0204    652000    LMQ         /MOVE TO MQ
        0205    200102    LAC DIVD2   /LOAD HALF DIVIDEND INTO AC
        0206    644323    DIVS        /FETCH DIVR AND DIVIDE
DIVR    0207    000005
        0210    620200    JMP I 200   /RETURN TO MAIN PROGRAM

        0100    000005    DIVISOR
        0101    000012    DIVIDEND (LEAST SIGNIFICANT)
        0102    000000    DIVIDEND (MOST SIGNIFICANT)
```

NOTE: The following discussion of a divide signed operation is using a 4 bit divisor and 8 bit dividend instead of 18 and 36. References to a given register bit 17 are referring to the least significant bit of the applicable register.

Table 3-23
DIVS Functions

644305 Divide, Signed (Five Steps)

$0101\overline{)0000\ 1010}$

$12_8 \div 5_8$

Process	Function	Drawing No.
75	(ACO,ARI,EAE,LI,CONT,CMA43)	KC18
	ACO(1)∧NOSH∧ARI(1) = AC → AR	KC20-21
	SA09(0)∧SA10(1)∧SA11(1) = DIV	KE4
	EAE(1)∧ARI(1) = SU1(1)	KE3
	SU1(1) = 0 → SCOV,SCOV2,FIRST,EAE RUN,MQ SIGN,EAE SIGN	KE2-3
	SU1(1)∧$\overline{\text{SETUP}}$ = SC CLR	KE2
	SC CLR = 0 → SC	KE2
	SU1(1)∧MB05(0) = EAE OR MQO	KE3
	LI(1) = O BUS L = ADRL → LAR(0)	KC15
	LI(1) = ADRL = LINK → TEMP3(0)	KE3
	TEMP3(0) = condition MQ SIGN	KE3
	EAE(1) = 0 → EN CMPL	KE3
	AC00(0) = no conditioning of EAE SIGN	KE3
	CM STROBE∧CONT(1) = GO TO 43	KC16
43	(ACI,EAE,CONT,CMA41)	KC18
	EAE(1)∧ACI(1)∧$\overline{\text{SETUP}}$ = SU2(1)	KE3
	SU2(1) = $\overline{\text{MB12-17}}$ = 111010 → SC	KE2
	SU2(1)∧MB06(1)∧AC00(0) = no effect on EAE SIGN (EAE SIGN 0)	KE3
	CM STROBE∧EAE OR MQO = MQO(1)	KC19
	MQO(1)∧NOSH∧ACI(1) = MQ → AC	KC20-21
	LI(0) = LAR(0) → LINK(0)	KC15
	CM STROBE∧CONT(1) = GO TO 41	KC16
41	(MQI,ACO,EAE,CONT,CMA54)	KC18
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	MQI(1)∧MB08(0) = EAE OR ARO	KE3
	CM STROBE∧CONT(1) = GO TO 54	KC16

Table 3-23 (cont)
DIVS Functions

644305 — Divide, Signed (Five Steps) — $12_8 \div 5_8$

Process	Function	Drawing No.
54	(ACI,EAE-R,CONT,CMA40)	KC18
	CM STROBE∧EAE OR ARO = ARO(1)	KC19
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111011 → SC	KE2
	EAE-R(1)∧SCOV2(0) = ADDR 10	KE3
	EAE-R(1)∧EIR09(0)∧SCOV2(0)∧EAE RUN(0) = ODD ADDR	KE3
	EAE(0)∧TEMP3(0) = MQ SIGN(1)	KE3
	MQ SIGN(1) = condition EAE SIGN	KE3
	EAE-R(1) = O BUS L = LINK → TEMP2(0)	KE3
	CM STROBE∧CONT(1)∧CMA40∧ADDR 10∧ODD ADDR = GO TO 51	KC16
51	(PCO,SM,MBI,CMA52)	KC18
	PCO(1)∧NOSH∧MBI(1) = PC → MB (DIVR ADDRESS)	KC20-21
	SM(1)∧CLK = FETCH DIVR	MC2
	SM(1)∧CLK = CM STROBE	KC16
	CM STROBE = GO TO 52	KC16
52	(MBO,+1,PCI,LI,CMA50)	KC18
	+1(1) = CI17	KC14
	MBO(1)∧NOSH∧CI17∧$\overline{\text{PCI(1)}}$ = MB (DIVR ADDRESS) +1 → PC	KC20-21
	+1(1) = $\overline{\text{A BUS LINK}}$ → $\overline{\text{ADRL}}$	KC15
	LI(1) = ADRL → LAR(0)	KC15
	LI(1)∧$\overline{\text{CONT(0)}}$ = EAE CLR RQ	KE3
	LI(1)∧$\overline{\text{ADRL}}$ = TEMP3(0)	KE3
	EAE CLR RQ = IN CLR, CLR	KC16
	IN CLR = CLR I = 0 → PCI, MBO	KC19
	CLR = 0 → +1, 1 → SAO	KC19
	IN CLR = $\overline{1}$ → MBI	KC19
	SAO(1) = $\overline{\text{A BUS LINK}}$ → ADRL	KC15
	SAO(1)∧NOSH∧MBI(1) = SA (DIVR) → MB	KC20-21
	MEM STROBE = GO TO 50	KC16
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
↑ Shift 1, Sample ↓	EAE-P(1)∧SCOV2(0) = EAE RUN(1)	KE3
	EAE-P(1)∧EAE RUN(0) = FIRST(1)	KE3
	FIRST(1)∧EAE RUN(1)∧MQ SIGN(1)=CMPL EAE SIGN=EAE SIGN(1)	KE3
	EAE-P(1)∧SCOV2(0)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	MQO(1)∧SHL1∧ARI(1) = MQn → ARn-1	KC20-21
	SHL1 = ADR00 = MQ00(1) → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1(1)	KE3
	EAE-P(1) = TEMP2(0) → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3(0) → END BIT17	KC15
	SHL1 = END BIT17 → AR17(0)	KC20
	LI(0) = LAR(0) → LINK(0)	KC15

Table 3-23 (cont)
DIVS Functions

644305 — Divide, Signed (Five Steps) — 12_8 2_8

Process	Function	Drawing No.
50 (cont)	EAE-P(1)∧SCOV(0)∧TEMP3(0)∧DIV = EAE OR SUB	KE3
	EAE-P(1)∧SCOV2(0)∧DIV = EAE OR LI	KE3
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	CM STROBE∧EAE OR SUB = SUB(1)	KC19
	CM STROBE∧EAE OR LI = LI(1)	KC19
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111100 → SC	KE2
	EAE-R(1)∧SCOV(0)∧EAE RUN(1)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	EAE-R(1)∧SUB(1) = CI17	KE3
Sub, Shift1	SUB(1)∧SHL1∧CI17∧MQI(1) = $\overline{MB}$+1 → MQn-1	KC20-21
	ACO(1)∧SHL1∧MQI(1) = ACn → MQn-1	KC20-21
	SHL1 = ADR00(1) → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2(1)	KE3
	LI(1) = O BUS L → LAR(1)	KC15
	EAE-R(1) = TEMP1(1) → END BIT17	KC15
	SHL1 = END BIT17 → MQ17(1)	KC20
	LINK(0)∧SUB(1)∧EAE R(1) = A BUS LINK	KC15
	A BUS LINK∧$\overline{CO00}$ = ADRL	KC15
	LI(1) = ADRL → TEMP3(1)	KE3
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	EAE-P(1)∧SCOV2(0)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	EAE-P(1)∧EAE RUN(1) = FIRST(0)	KE3
	ARO(1)∧SHL1∧ACI(1) = ARn → ACn-1	KC20-21
	SHL1 = ADR00(0) → O BUS L	KC15
Shift 2, Sample	EAE-P(1) = O BUS L → TEMP1(0)	KE3
	EAE-P(1) = TEMP2(1) → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3(1) → END BIT17	KC15
	SHL1 = END BIT17 → $\overline{AC17(1)}$	KC20
	EAE-P(1)∧SCOV2(0)∧$\overline{EAE\ OR\ SUB}$∧O BUS 17∧DIV = EAE OR MBO	KE3
	LI(0) = LAR(1) → LINK(1)	KC15
	EAE-P(1)∧SCOV2(0)∧DIV = EAE OR LI	KE3
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	CM STROBE∧EAE OR MBO = MBO(1)	KC19
	CM STROBE∧EAE OR LI = LI(1)	KC19
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111101 → SC	KE2
Add, Shift 2	EAE-R(1)∧SCOV(0)∧EAE RUN(1)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	MQO(1)∧SHL1∧ARI(1) = MQn → ARn-1	KC20-21

Table 3-23 (cont)
DIVS Functions

644305 — Divide, Signed (Five Steps) — $12_8 \div 5_8$

Process	Function	Drawing No.
53 (cont)	MBO(1)∧SHL1∧ARI(1) = MBn → ARn-1	KC20-21
	SHL1 = $\overline{\text{ADR00}}$(1) → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2(1)	KE3
	LI(1) = O BUS L → LAR(1)	KC15
	LI(1) = ADRL → TEMP3(1)	KE3
	EAE-R(1) = TEMP1(0) → END BIT17	KC15
	SHL1 = END BIT17 → AR17(0)	KC20
	LINK(1)∧$\overline{\text{SUB}}$ = A BUS LINK	KC15
	A BUS LINK∧$\overline{\text{CO00}}$ = ADRL	KC15
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	EAE-P(1)∧SCOV2(1)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	ACO(1)∧SHL1∧MQI(1) = ACn → MQn-1	KC20-21
	SHL1 = ADR00(1) → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1(1)	KE3
Shift 3,	EAE-P(1) = TEMP2(1) → END BIT00 (lost)	KC15
Sample	EAE-P(1) = TEMP3(1) → END BIT17	KC15
	SHL1 = END BIT17 → MQ17(1)	KC20
	EAE-P(1)∧SCOV2(0)∧$\overline{\text{EAE OR SUB}}$∧O BUS 17∧DIV = EAE OR MBO	KE3
	LI(0) = LAR(1) → LINK(1)	KC15
	EAE-P(1)∧SCOV2(0)∧DIV = EAE OR LI	KE3
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	CM STROBE∧EAE OR MBO = MBO(1)	KC19
	CM STROBE∧EAE OR LI = LI(1)	KC19
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111110 → SC	KE2
	EAE-R(1)∧SCOV(0)∧EAE RUN(1)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
Add,	ARO(1)∧SHL1∧ACI(1) = ARn → ACn-1	KC20-21
Shift 3	MBO(1)∧SHL1∧ACI(1) = MBn → ACn-1	KC20-21
	SHL1 = ADR00(1) → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2(1)	KE3
	EAE-R(1) = TEMP1(1) → END BIT17	KC15
	SHL1 = END BIT17 → AC17(1)	KC20
	LI(1) = O BUS L → LAR(1)	KC15
	LINK(1)∧$\overline{\text{SUB}}$ = A BUS LINK	KC15
	A BUS LINK∧$\overline{\text{CO00}}$ = ADRL	KC15
	LI(1) = ADRL → TEMP3(1)	KC15
	EAE-R(1)∧SCOV2(0) = ADDR 10	KE3
	CM STROBE∧CONT(1)∧CMA40∧ADDR 10 = GO TO 50	KC16

Table 3-23 (cont)
DIVS Functions

644305 — Divide, Signed (Five Steps) — 12_8 5_8

Process	Function	Drawing No.
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
Shift 4, Sample	EAE-P(1)∧SCOV2(0)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	MQO(1)∧SHL1∧ARI(1) = MQn → ARn-1	KC20-21
	SHL1 = ADR00(0) → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1(0)	KE3
	EAE-P(1) = TEMP2(1) → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3(1) → END BIT17	KC15
	SHL1 = END BIT17 → AR17(1)	KC20
	LI(0) = LAR(1) → LINK(1)	KC15
	EAE-P(1)∧SCOV2(0)∧$\overline{\text{EAE OR SUB}}$∧O BUS 17∧DIV = EAE OR MBO	KE3
	EAE-P(1)∧SCOV2(0)∧DIV = EAE OR LI	KE3
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
Add, Shift 4	CM STROBE∧EAE OR MBO = MBO(1)	KC19
	CM STROBE∧EAE OR LI = LI(1)	KC19
	EAE-R(1)∧SCOV(0) = R-PULSE	KE2
	R-PULSE = 111111 → SC = SC FULL	KE2
	EAE-R(1)∧SCOV(0)∧EAE RUN(1)∧DIV = SHL1	KE4
	IN SHL1 = SHL1	KC13
	ACO(1)∧SHL1∧MQI(1) = ACn → MQn-1	KC20-21
	MBO(1)∧SHL1∧MQI(1) = MBn → MQn-1	KC20-21
	SHL1 = ADR00(0) → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2(0)	KE3
	LI(1) = O BUS L → LAR(0)	KC15
	EAE-R(1) = TEMP1(0) → END BIT17	KC15
	SHL1 = END BIT17 → MQ17(0)	KC20
	LINK(1)∧$\overline{\text{SUB}}$ = A BUS LINK	KC15
	A BUS LINK∧CO00 = $\overline{\text{ADRL}}$	KC15
	LI(1) = $\overline{\text{ADRL}}$ → TEMP3(0)	KE3
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
Shift 5, Sample	EAE-P(1)∧SCOV2(0)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	ARO(1)∧SHL1∧ACI(1) = ARn → ACn-1	KC20-21
	SHL1 = ADR00(0) → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1(0)	KE3
	EAE-P(1) = TEMP2(0) → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3(0) → END BIT17	KC15
	SHL1 = END BIT17 → AC17(0)	KC20
	EAE-P(1)∧SCOV2(0)∧DIV = EAE OR LI	KE3
	EAE-P(1)∧SCOV(0)∧TEMP3(0)∧DIV = EAE OR SUB	KE3
	LI(0) = LAR(0) → LINK(0)	KC15
	CM STROBE∧CONT(1) = GO TO 53	KC16

Table 3-23 (cont)
DIVS Functions

644305 — Divide, Signed (Five Steps) — $12_8 \div 5_8$

Process	Function	Drawing No.
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	CM STROBE∧EAE OR SUB = SUB(1)	KC19
	CM STROBE∧EAE OR LI = LI(1)	KC19
	EAE-R(1) = R-PULSE	KE2
	R-PULSE = 000000 → SC	KE2
	R-PULSE∧SC FULL = SCOV(1)	KE2
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
Sub	SUB(1)∧EAE-R(1) = CI17	KE3
	SUB(1)∧NOSH∧CI17∧ARI91) = $\overline{\text{MB}}$+1 → AR	KC20-21
	MQO(1)∧NOSH∧ ARI(1) = MQ → AR	KC20-21
	SUB(1)∧EAE-R(1)∧LINK(0) = A BUS LINK	
	A BUS LINK∧$\overline{\text{CO00}}$ = ADRL	KC15
	$\overline{\text{SHIFT}}$= ADRL → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2(1)	KE3
	LI(1) = O BUS L → LAR(1)	KC15
	LI(1) = ADRL → TEMP3(1)	KE3
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	EAE-P(1)∧SCOV2(0)∧DIV = IN SHL1	KE4
	IN SHL1 = SHL1	KC13
	ACO(1)∧SHL1∧MQI(1) = ACn → MQn-1	KC20-21
	SHL1 = ADR00(1) → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1(1)	KE3
Shift 5,	EAE-P(1) = TEMP2(1) → END BIT00 (lost)	KC15
Sample	EAE-P(1) = TEMP3(1) → END BIT17	KC15
	SHL1 = END BIT17 → MQ17(1)	KC20
	EAE-P(1)∧SCOV2(0)∧$\overline{\text{EAE OR SUB}}$∧O BUS 17∧DIV = EAE OR MBO	KE3
	EAE-P(1)∧SCOV2(1)∧DIV = EAE OR LI	KE3
	LI(0) = LAR(1) → LINK(1)	KC15
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	CM STROBE∧EAE OR MBO = MBO(1)	KC19
	CM STROBE∧EAE OR LI = LI(1)	KC19
	EAE-R(1)∧SCOV(1) = SCOV2(1)	KE2
	SCOV(1) = $\overline{\text{IN SHL1}}$	KE4
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	MBO(1)∧NOSH∧ACI(1) = MB → AC	KC20-21
Add	A BUS LINK∧CO00 = $\overline{\text{ADRL}}$	KC15
	$\overline{\text{SHIFT}}$ = $\overline{\text{ADRL}}$ → $\overline{\text{O BUS L}}$	KC15
	EAE-R(1) = $\overline{\text{O BUS L}}$ → TEMP2(0)	KE3
	LI(1) = $\overline{\text{O BUS L}}$ → LAR(0)	KC15
	LI(1) = ADRL → TEMP3(0)	KE3
	EAE-R(1)∧RUN(1) = ADDR 10	KE3
	CM STROBE∧CONT(1)∧CMA40∧ADDR 10 = GO TO 50	KC16

Table 3-23 (cont)
DIVS Functions

644305 — Divide, Signed (Five Steps) — $12_8 \div 5_8$

Process	Function	Drawing No.
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	SCOV2(1) = $\overline{\text{IN SHLT}}$	KE4
	SCOV2(1) = $\overline{\text{EAE OR MBO}}$, $\overline{\text{EAE OR SUB}}$, $\overline{\text{EAE OR LI}}$	KE3
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	LI(0) = LAR(0) → LINK(0)	KC15
	LINK(0) = $\overline{\text{ADRL}}$	KC15
	$\overline{\text{SHIFT}}$ = $\overline{\text{ADRL}}$ → $\overline{\text{O BUS L}}$	KC15
	EAE-P(1) = $\overline{\text{O BUS L}}$ → TEMP1(0)	KE3
	EAE-P(1) = TEMP2(0) → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3(0) → END BIT 17	KC15
	CM STROBE∧CONT(1) → GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	EAE-R(1)∧SCOV2(1) = EAE RUN(0)	KE3
	ACO(1)∧NOSH∧MQI(1) = AC → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	EAE-P(1)∧MQI(1)∧ACO(1)∧EIR09(0)∧SCOV2(1) = EN CMPL(1)	KE3
	EN CMPL(1)∧EAE SIGN(1) = CMPL	KE3
	ACO(1)∧NOSH∧MQI(1)∧CMPL = $\overline{\text{AC}}$ → MQ	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	EAE-R(1)∧DIV∧EN CMPL∧MQ SIGN(1)∧EAE SIGN(1) = EAE SIGN(0)	KE3
	EAE SIGN(0) = $\overline{\text{CMPL}}$	KE3
	ARO(1)∧NOSH∧ACI(1)∧$\overline{\text{CMPL}}$ = AR → AC	KC20-21
	EAE-R(1)∧SCOV2(1)∧RUN(0) = $\overline{\text{ADDR 10}}$	KE3
	CM STROBE∧CONT(1) = GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	CLK(B) DLYD∧EAE(1)∧DONE(1) = INPUT IO RESTART	KD3
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

Table 3-24
DIVS Arithmetic

$12_8 \div 5_8$

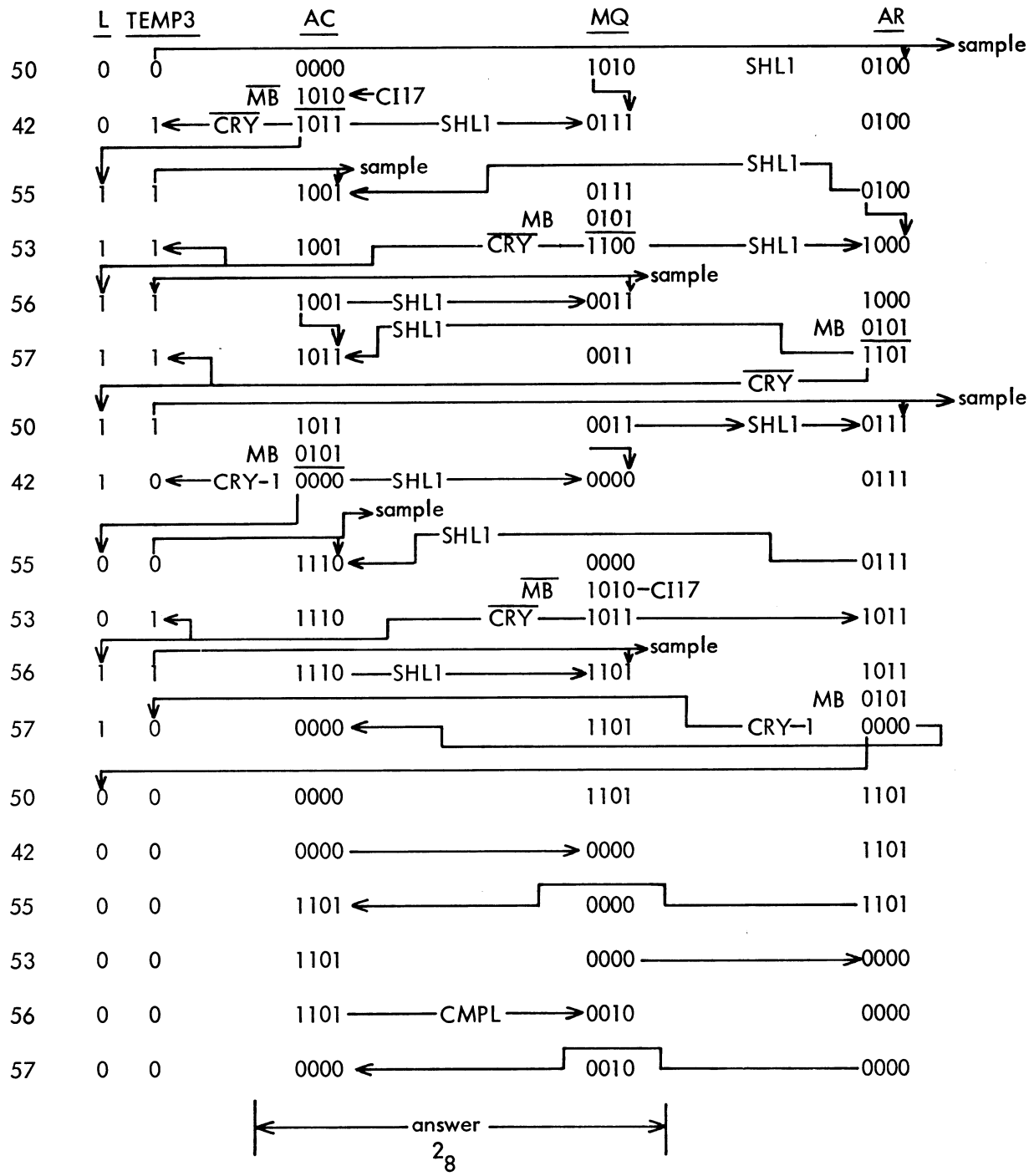


Table 3-25
DIVS Functions

$12_8 \div -5_8$

Process	Function
75	TEMP3(1) = no conditioning of MQ SIGN AC00(0) = no conditioning of EAE SIGN EAE(1) = 0 → EN CMPL
43	MQ → AC
41	AC → MQ
54	EAE(0)∧TEMP3(1) = no effect on MQ SIGN
51 through last 53 same as DIVS $12_8 \div 5_8$	
56	EN CMPL(1)∧EAE SIGN(0) = $\overline{\text{CMPL}}$ AC → MQ
57	CMPL = AR → AC

Table 3-26
DIVS Functions

$-12_8 \div +5_8$

Process	Function
75	TEMP3(0) = condition MQ SIGN AC00(1) = condition EAE SIGN EAE(1) = 0 → EN CMPL
43	MB06(1)∧SU2(1) = EAE SIGN(1) SU2(1)∧$\overline{\text{EAE}}$ SIGN(1) = CMPL CMPL = $\overline{\text{MQ}}$ → AC
41	AC → MQ
54	EAE(0)∧TEMP3(0) = MQ SIGN(1) MQ SIGN(1) = condition EAE SIGN
51,52	same as DIVS $12_8 \div 5_8$
50	FIRST(1)∧EAE RUN(1)∧MQ SIGN(1)∧EAE SIGN(1) = EAE SIGN(0)
42 through last 53 same as DIVS $12_8 \div 5_8$	
56	EN CMPL(1)∧EAE SIGN(0) = $\overline{\text{CMPL}}$ AC → MQ
57	EAE-R(1)∧EN CMPL(1)∧DIV∧MQ SIGN(1)∧EAE SIGN(0) = EAE SIGN(1) EAE SIGN(1)∧EN CMPL(1) = CMPL CMPL = $\overline{\text{AR}}$ → AC

Table 3-27
DIVS Functions

$-12_8 \div -5_8$

Process	Function
75	TEMP3(1) = no conditioning of MQ SIGN AC00(1) = condition EAE SIGN EAE(1) = 0 → EN CMPL
43	MB06(1)∧SU2(1) = EAE SIGN(1) SU2(1)∧EAE SIGN(1) = CMPL CMPL = $\overline{MQ}$ → AC
41	AC → MQ
54	EAE(0)∧TEMP3(1) = no effect on MQ SIGN
51	through last 53 same as $12_8 \div 5_8$
56	EN CMPL(1)∧EAE SIGN(1) = CMPL CMPL = $\overline{AC}$ → MQ
57	EAE-R(1)∧EN CMPL(1)∧DIV∧MQ SIGN(0) = no effect on EAE SIGN(1) EAE SIGN(1)∧EN CMPL(1) = CMPL $\overline{AR}$ → AC

3.8.2 IDIV(S) Instruction

The IDIV(S) instruction divides the contents of the AC (integer dividend) by the contents of the next sequential core memory location to form a quotient in the MQ and a remainder in the AC.

The arithmetic phase of the instruction(s) is identical to that of DIV(S). The preparatory phase transfers the contents of the AC to the MQ and clears the AC. Thereafter the arithmetic phase in reality performs the division on the long register dividend just as for DIV. The exception here is that the most significant portion of the dividend (AC) is at 0.

Therefore, the DIV(S) functions of Table 3-23 hold true for IDIV(S) with the following preparatory exceptions.

75) SU1(1)∧MB07(1) = EAE OR ARO
AC → AR (same)

43) AR → AC

41) MB08(1) = $\overline{\text{EAE OR ARO}}$
AC → MQ (same)

54) ACI(1) = 0 → AC

The rule for divide overflow, Section 3.8.4 is the same. In the IDIV(S) case overflow occurs only if the computer attempts to divide by 0, since this is the only quantity not larger than the AC portion of the dividend.

The sample divide in Table 3-23, although performed by a DIVS instruction, could in fact be used as a sample IDIVS operation since the arithmetic phase also starts with a zero quantity in the AC.

3.8.3 FRDIV(S) Instruction

The FRDIV(S) instruction divides the contents of the AC (fraction dividend) by the contents of the next sequential core memory location to form a quotient in the MQ and a remainder in the AC.

The arithmetic phase of the instruction(s) is identical to that of DIV(S). The preparatory phase clears the MQ. The arithmetic phase thereafter is in reality a division of the long register with the MQ at 0. For FRDIV the binary point is assumed at the left of AC00. For FRDIVS the binary point is assumed between AC00 and AC01. The divide overflow rule, Section 3.8.4, is the same.

The DIV(S) functions of Table 3-23 hold true for FRDIV(S) therefore, with the following exceptions.

75) $SU1(1) \wedge MB05(1) = \overline{EAE \text{ OR } MQO}$
$SU1(1) \wedge MB07(0) = \overline{EAE \text{ OR } ARO}$
AC → AR (same)

43) ACI(1) = 0 → AC

41) AC → MQ (same)

54) AR → AC (same)

3.8.4 Divide Overflow

For all divide instructions the first subtract operation of the arithmetic phase checks for a divide overflow situation. Divide overflow exists when the computer attempts to divide a dividend by a divisor which is not numerically greater than the most significant portion (AC) of the dividend. If the divide operations were carried out, the result would exceed the capacity of the 18-bit MQ register, and the MQ contents would be erroneous. For unsigned division, the capacity of the MQ is $2^{18}-1$, or 777777_8. For signed division the capacity is $+2^{17}-1$, or $+377777_8$.

For all divide instructions process word 52 during the divisor fetch from memory blocks the recirculation of the LINK into the LAR; process word 50 transfers the LAR content(0) into the LINK and starts the arithmetic phase of the instruction. The arithmetic phase therefore always starts with the LINK in the reset state. The LINK returns to the reset state at the end of all valid divide instructions. If, however, the EAE logic encounters the divide overflow situation, the LINK sets and the instruction execution is halted after five machine cycles as a time-saving feature. The computer will then go on to the next instruction, which is usually an instruction which tests the status of the LINK (OPR SZL, OPR SNL, etc.).

Table 3-28 lists the functions that provide the overflow indication to the LINK and stop the divide operations. The listing starts with process word 50, at which point the preparatory phase has been completed, the divisor is in the MB, and the dividend is correctly placed in the AC and MQ. The operation attempts to divide 32_{10} by 2_{10} for a quotient of 16 using a 4-bit MQ register, resulting in overflow since the register capacity is 15 for unsigned divide.

Note from Table 2-3 that a valid five-step arithmetic divide operation requires seven machine cycles for completion, whereas divide overflow stops the operation after the first step and five cycles. For the overflow situation the step count in the SC does not matter since the DIV OV flip-flop controls the SCOV, SCOV2, and RUN functions.

Table 3-28
DIV OV Functions

640305 — Divide, Unsigned (Five Steps) — $32_{10} \div 2_{10}$

Process	Function	Drawing No.
50	(MQO,ARI,EAE-P,CONT,CMA42)	KC18
	EAE-P(1)∧EAE RUN(0) = FIRST(1)	KE3
	EAE-P(1)∧SCOV2(0) = EAE RUN(1)	KE3
	EAE-P(1) etc. = SHL1	KE4
	FIRST(1)∧EAE RUN(1)∧MQ SIGN(1)∧EAE SIGN(0) = EAE SIGN(1)	KE3
	LI(0) = LAR(0) → LINK(1)	KC15
	EAE-P(1)∧SCOV(0)∧TEMP3(0)∧DIV = EAE OR SUB	KE3
	EAE-P(1)∧SCOV2(0)∧DIV = EAE OR LI	KE3
	MQO(1)∧SHL1∧ARI(1) = MQn → ARn-1	KC20-21
	SHL1 = ADR00(0) → O BUS L	KC15
	EAE-P(1) = O BUS L → TEMP1(0)	KE3
	EAE-P(1) = TEMP2 → END BIT00 (lost)	KC15
	EAE-P(1) = TEMP3(0) → END BIT17	KC15
	SHL1 = END BIT17 → AR 17(0)	KC20
	CM STROBE∧CONT(1) = GO TO 42	KC16
42	(ACO,MQI,EAE-R,CONT,CMA55)	KC18
	CM STROBE∧EAE OR SUB = SUB(1)	KC19
	EAE-R(1)∧SUB(1) = CI17	KE3
	CM STROBE EAE OR LI = LI(1)	KC19
	EAE-R(1), etc. = SHL1	KE4
	ACO(1)∧SHL1∧MQI(1) = ACn → MQn-1	KC20-21
	SUB(1)∧SHL1∧MQI(1)∧CI17 = $\overline{MB+1}$ → MQn-1	KC20-21
	EAE-R(1)∧SUB(1)∧LINK(0) = A BUS LINK	KC15
	A BUS LINK∧CO00 = $\overline{ADRL}$	KC15
	$\overline{ADRL}$ = $\overline{ADRL(B)}$	KC15
	EAE-R(1)∧FIRST(1)∧$\overline{ADRL(B)}$∧DIV = DIV OV(1)	KE3
	SHL1 = ADR00(0) → O BUS L	KC15
	EAE-R(1) = O BUS L → TEMP2(0)	KE3
	EAE-R(1) = TEMP1(0) → END BIT17	KC15
	SHL1 = END BIT17 → MQ17(0)	KC20
	LI(1) = DIV OV(1) → LAR(1)	KC15
	LI(1) = $\overline{ADRL}$ → TEMP3(0)	KE3
	CM STROBE∧CONT(1) = GO TO 55	KC16
55	(ARO,ACI,EAE-P,CONT,CMA53)	KC18
	EAE-P(1)∧RUN(1) = FIRST(0)	KE3
	EAE-P(1)∧DIV OV(1) = DIV NO GO	KE2
	DIV NO GO = SCOV(1),SCOV2(1),EAE RUN(0)	KE2-3

Table 3-28(cont)
DIV OV Functions

640305 — Divide, Unsigned (Five Steps) — $32_{10} \div 2_{10}$

Process	Function	Drawing No.
55 (cont)	LI(0) = LAR(1) → LINK(1)	KC15
	SCOV2(1) = $\overline{\text{IN SHLT}}$	KE4
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	CM STROBE∧CONT(1) = GO TO 53	KC16
53	(MQO,ARI,EAE-R,CONT,CMA56)	KC18
	MQO(1)∧NOSH∧ARI(1) = MQ → AR	KC20-21
	CM STROBE∧CONT(1) = GO TO 56	KC16
56	(ACO,MQI,EAE-P,CONT,CMA57)	KC18
	ACO(1)∧NOSH∧MQI(1)∧CMPL = $\overline{\text{AC}}$ → MA	KC20-21
	CM STROBE∧CONT(1) = GO TO 57	KC16
57	(ARO,ACI,EAE-R,CONT,CMA40)	KC18
	ARO(1)∧NOSH∧ACI(1) = AR → AC	KC20-21
	EAE-R(1)∧SCOV2(1)∧EAE RUN(0) = $\overline{\text{ADDR 10}}$	KE3
	CM STROBE∧CONT(1)∧CMA40∧$\overline{\text{ADDR 10}}$ = GO TO 40	KC16
40	(EAE,DONE,CMA10)	KC18
	CLK(B) DLYD EAE(1) DONE(1) = INPUT IO RESTART	KD3
	IO RESTART = GO TO 10	KC16
10	(PCO,SM,CMA21)	KC18
	BGN next fetch	

3.9 EAE INSTRUCTION DEVELOPMENT

The addition of n_8 bits to the basic EAE op code 64_8 converts the basic instruction to a microcoded instruction to accomplish a setup, shift, or arithmetic operation not already in the instruction repertoire. Refer to Table 3-29 for descriptions of the functional use of the individual bits. The sole restriction for development of "n" is that the microcoded operations must not occur during the same process word if they logically conflict.

Table 3-29
EAE Microinstructions

Bit	Binary Code	Function
4	1	Enters AC00 into the LINK for signed operations.
5	1	Clears the MQ.

Table 3-29 (cont)
EAE Microinstructions

Bit	Binary Code	Function
6	1	Reads AC00 into the EAE SIGN register prior to a signed multiply or divide operation.
6,7	10	Takes the absolute value of the AC after the AC00 bit is read into the EAE SIGN register.
7	1	Inclusive-ORs the AC with the MQ and places the result in the MQ.
8	1	Clears the AC.
9,10,11	000	SETUP instruction code. Accompanies code in bits 15, 16, 17.
9,10,11	001	MUL instruction code.
9,10,11	010	Unused instruction code.
9,10,11	011	DIV instruction code.
9,10,11	101	LONG RIGHT SHIFT instruction code.
9,10,11	110	LONG LEFT SHIFT instructions code.
9,10,11	100	NORMALIZE instruction code.
9,10,11	111	ACCUMULATOR LEFT SHIFT instruction code.
12-17		Specifies the step count for all EAE codes (9-11) except SETUP.
15	1	For SETUP instruction code only, complements the MQ contents.
16	1	For SETUP instruction code only, inclusive-ORs the MQ with the AC and places the result in the AC.
17	1	For SETUP instruction code only, inclusive-ORs the AC with the SC and places the result in the AC.

CHAPTER 4
MAINTENANCE

4.1 GENERAL MAINTENANCE

The general maintenance practices described in the PDP-9 Maintenance Manual also apply to the EAE option.

4.2 MAINTENANCE PROGRAM TAPES

Chapter 1 of the PDP-9 Maintenance Manual lists the diagnostic tapes and documents for use with the EAE.

4.3 REPLACEABLE PARTS

Table 4-1 lists all logic modules used in the EAE option by DEC type and quantity. The CP UML drawing KC8 shows the module locations in the central processor wing of the PDP-9 frame. DEC has available a spare modules kit, SP09A, for use with the basic PDP-9 system and including spares for the EAE option. If the kit is not on hand, it is recommended that one spare module of each logic type be stocked to reduce equipment down-time while repairing faulty modules.

Table 4-1
EAE Module Complement

DEC Type	Module Type	Quantity
B105	Inverter	1
B133	Inverter	1
B213	Flip-Flop	15
R002	Diode Network	8
R111	NAND/NOR Gate	11
S151	Binary-to-Octal Decoder	1
S181	DC Carry Chain	1
S206	Flip-Flop	6
W005	Clamped Load	1

CHAPTER 5
ENGINEERING DRAWINGS

This chapter contains a complete set of engineering drawings pertaining to the EAE option along with circuit schematics of all logic modules. DEC engineering drawings are encoded as to type, major assembly, and series. Drawing number codes and signal conventions are explained in Chapter 5 of the PDP-9 Maintenance Manual.

5.1 SIGNAL MNEMONIC INDEX

All signals originating on the EAE logic drawings are listed below in alphanumeric order. The Origin column locates the source of the signals to the specific logic drawing, using the abbreviated drawing number system.

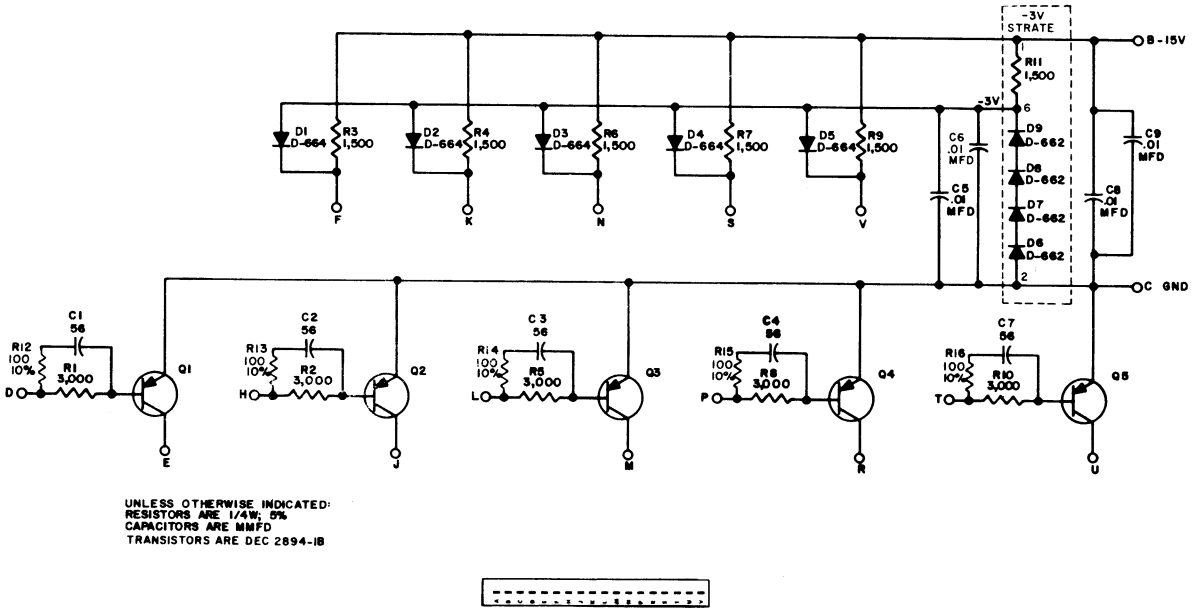
Signal	Origin	Description
A BUS LINK	KE3	Enter AC00 into LINK
AC0 → LINK	KE3	Recirculate LINK via LAR
ADDR 10	KE3	Add 10 to next Control Memory address
ALS	KE4	Accumulator Left Shift command
CMPL	KE3	Complement the register contents in transfer
CI17	KE3	Initiate a carry into the Adder
DIV	KE4	Divide command
DIV NO GO	KE2	Stop divide operations
DIV OV	KE3	Divide Overflow
EAE CLR RQ	KE3	Clear CM gating bits for argument fetch
EAE OR ARO	KE3	Set ARO bit on next CM STROBE
EAE OR LI	KE3	Set LI bit on next CM STROBE
EAE OR MBO	KE3	Set MBO bit on next CM STROBE
EAE OR MQO	KE3	Set MQO bit on next CM STROBE
EAE OR SUB	KE3	Set SUB bit on next CM STROBE
EAE PWR CLR	KE3	Clear flip-flops on power turn-on
EAE RUN	KE3	Start EAE instruction execution
EAE SIGN	KE3	Store AC00
EIR09-11	KE4	EAE instruction register
EN CMPL	KE3	Enable complement function
FIRST	KE3	Start first arithmetic operation

Signal	Origin	Description
IN SHL1	KE4	Enable Shift Left Function
IN SHR1	KE4	Enable Shift Right function
LLS	KE4	Long Left Shift command
LRS	KE4	Long Right Shift command
MQ SIGN	KE3	Store divisor or multiplicand sign
MUL	KE4	Multiply command
NORM	KE4	Normalize command
ODD ADDR	KE3	Add 1 to next CM address
O BUS17(B)	KE3	END Bit shifted into next register
R-PULSE	KE2	Up-date the Step Count
SC12-17	KE2	Step Counter register
SC CLR	KE2	Clear the Step Counter
SC FULL	KE2	Step Counter up-dated to 77_8
SCO	KE2	Step Counter output gate
SCOV	KE2	Step Counter up-dated to 00_8
SCOV(1)	KE2	Set SCOV on normalize condition
SCOV2	KE2	Step Counter up-dated to 00_8
SETUP	KE4	Setup command
SU1-3	KE3	Setup or preparatory instruction phase
TEMP1-3	KE3	Temporary LINK and END Bit storage

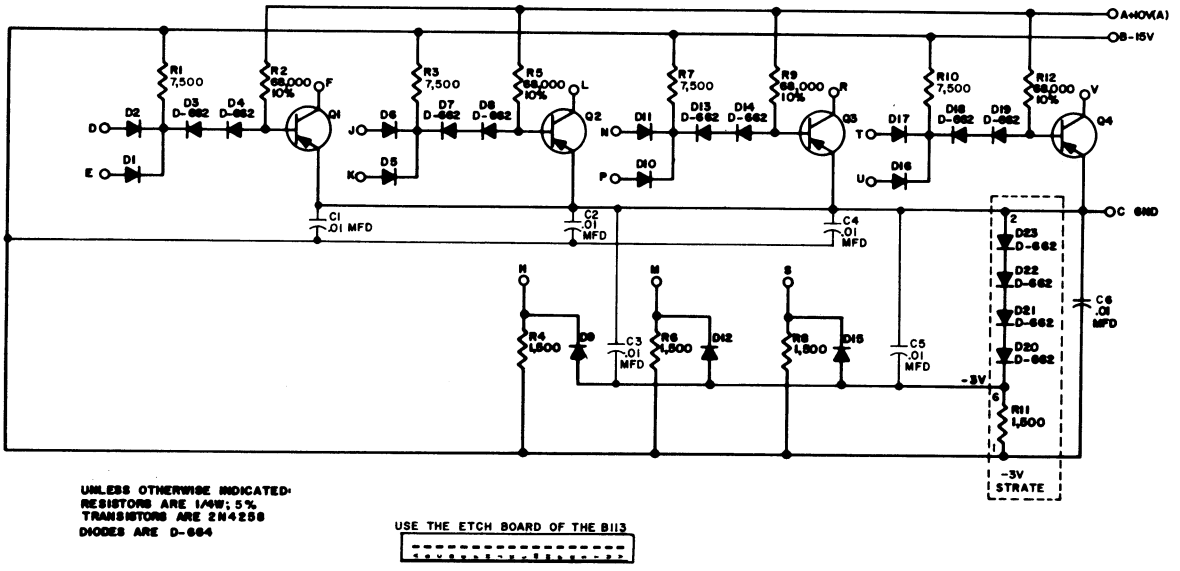
5.2 DRAWING LIST

Below is a list of all drawings included in this chapter. Other related EAE logic is included in the Chapter 5 drawings of the PDP-9 Maintenance Manual as part of the prewired, basic system.

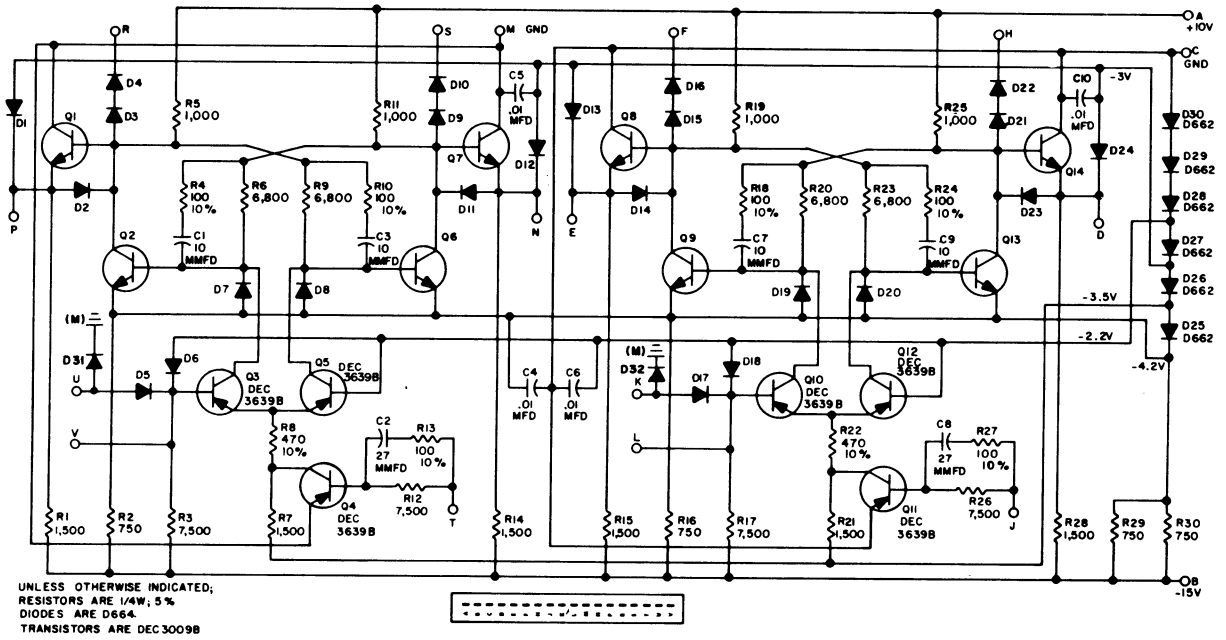
Drawing Number	Title	Revision	Page
B-CS-B105-0-1	Inverter B105, Circuit Schematic	E	5-4
B-CS-B133-0-1	Inverter B133, Circuit Schematic	B	5-4
B-CS-B213-0-1	Flip-Flop B213, Circuit Schematic	F	5-5
B-CS-R002-0-1	Diode Network R002, Circuit Schematic	A	5-5
B-CS-R111-0-1	NAND/NOR Gate R111, Circuit Schematic	F	5-6
B-CS-S151-0-1	Binary-to-Octal Decoder S151, Circuit Schematic	C	5-6
B-CS-S181-0-1	DC Carry Chain S181, Circuit Schematic	A	5-7
B-CS-S206-0-1	Flip-Flop S206, Circuit Schematic	B	5-7



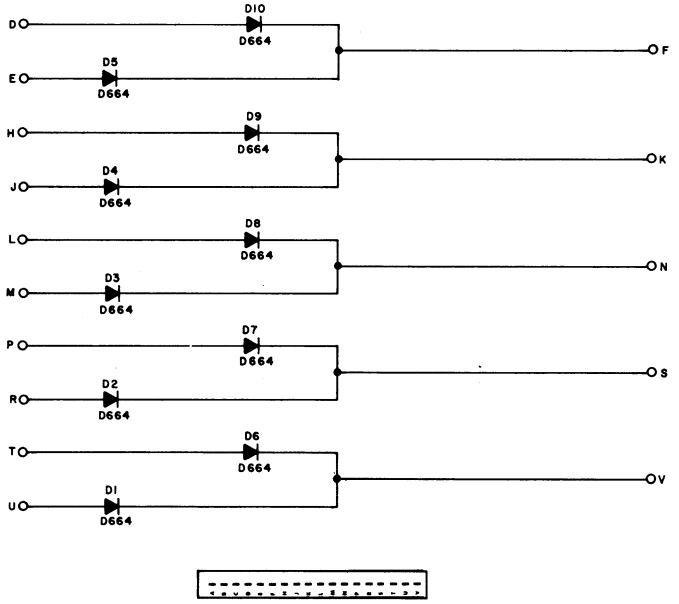
B-CS-B105-0-1 Inverter B105, Circuit Schematic



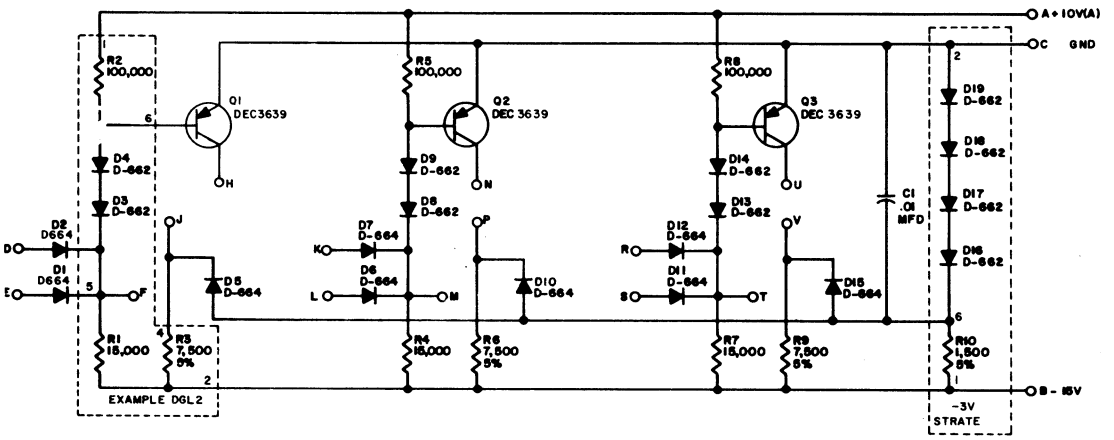
B-CS-B133-0-1 Inverter B133, Circuit Schematic



B-CS-B213-0-1 Flip-Flop B213,
Circuit Schematic

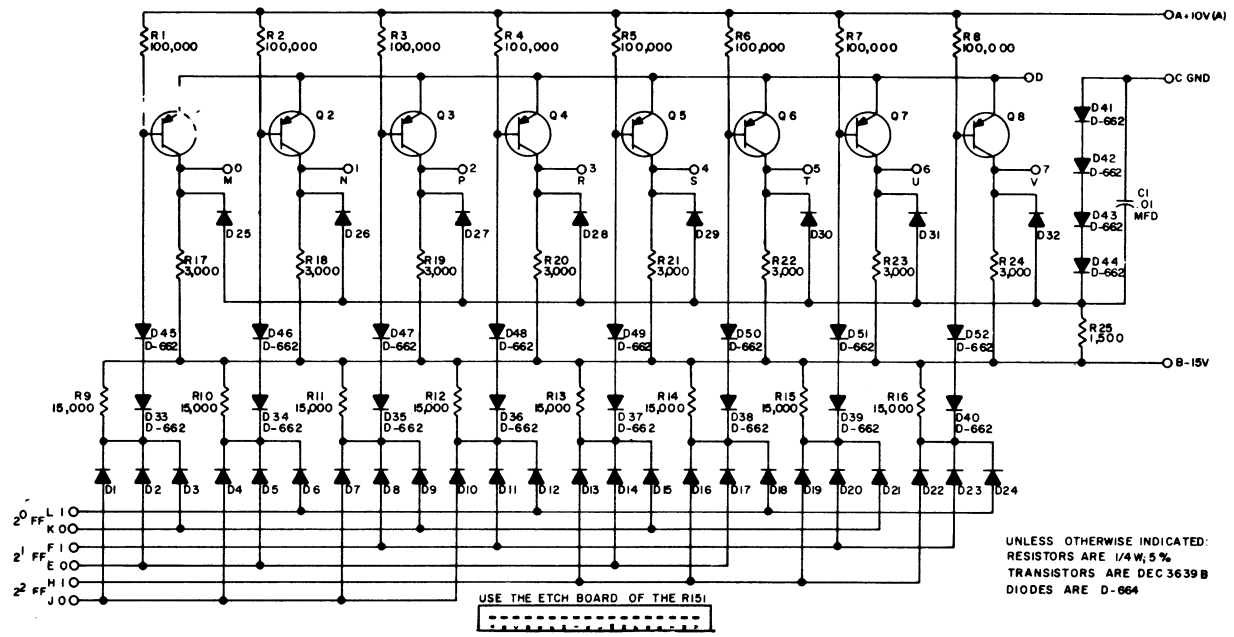


B-CS-R002-0-1 Diode Network R002,
Circuit Schematic



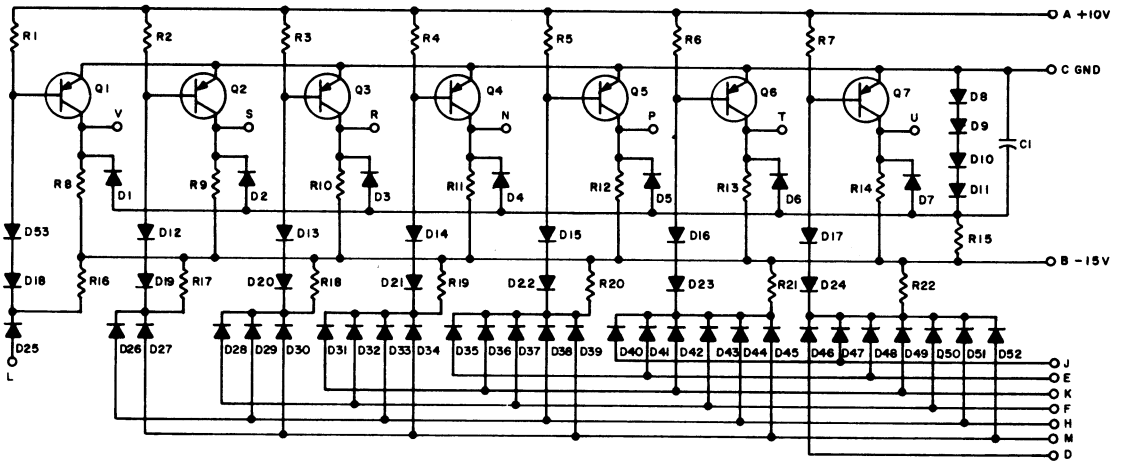
UNLESS OTHERWISE INDICATED:
RESISTORS ARE 1/4W; 5%
PRINTED CIRCUIT REV. FOR
DGL BOARD IS SIB

B-CS-R111-0-1 NAND/NOR Gate R111,
Circuit Schematic

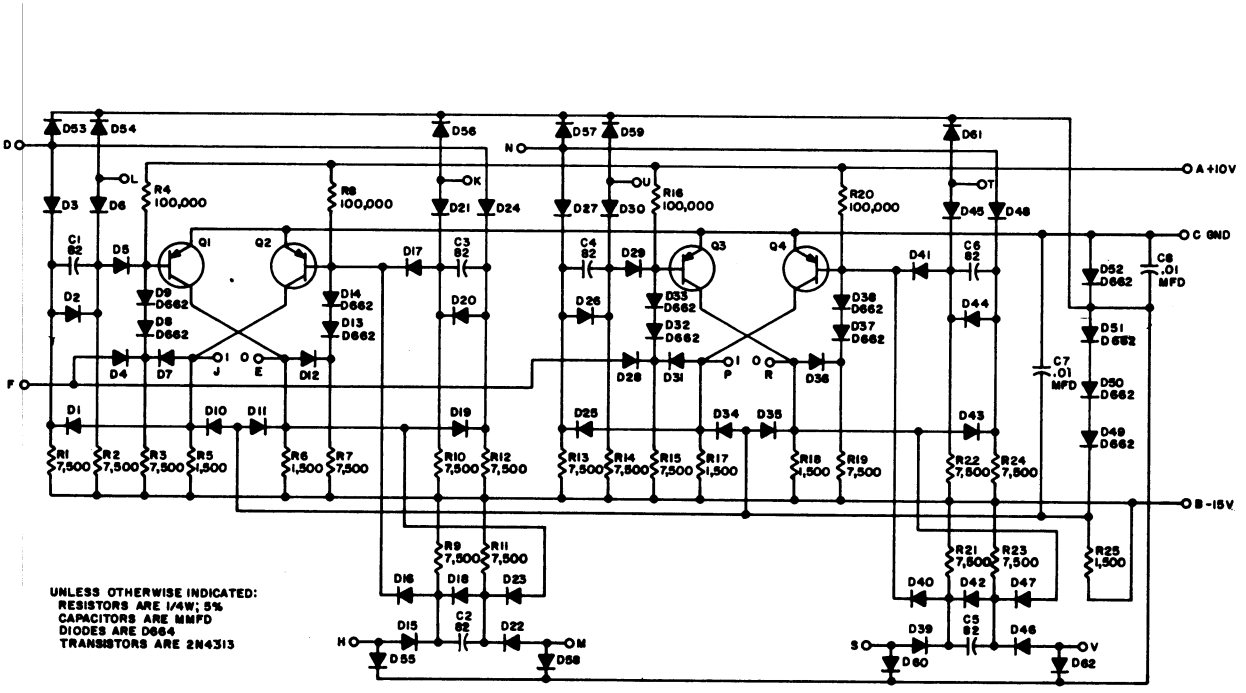


UNLESS OTHERWISE INDICATED:
RESISTORS ARE 1/4 W, 5 %
TRANSISTORS ARE DEC 3639 B
DIODES ARE D-664

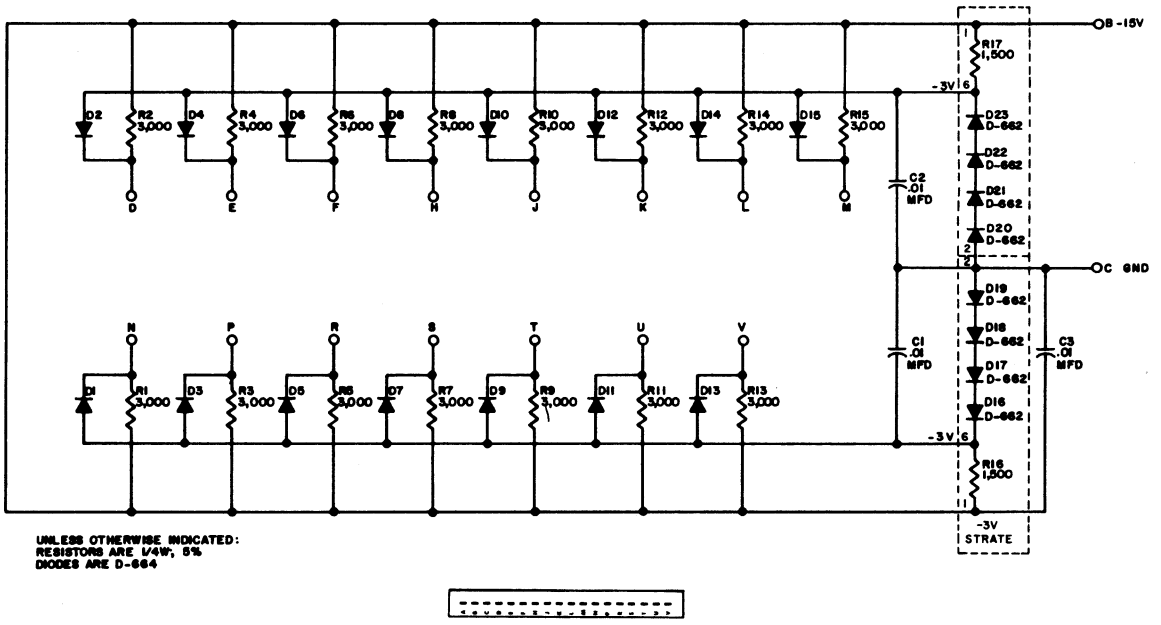
B-CS-S151-0-1 Binary-to-Octal Decoder S151,
Circuit Schematic



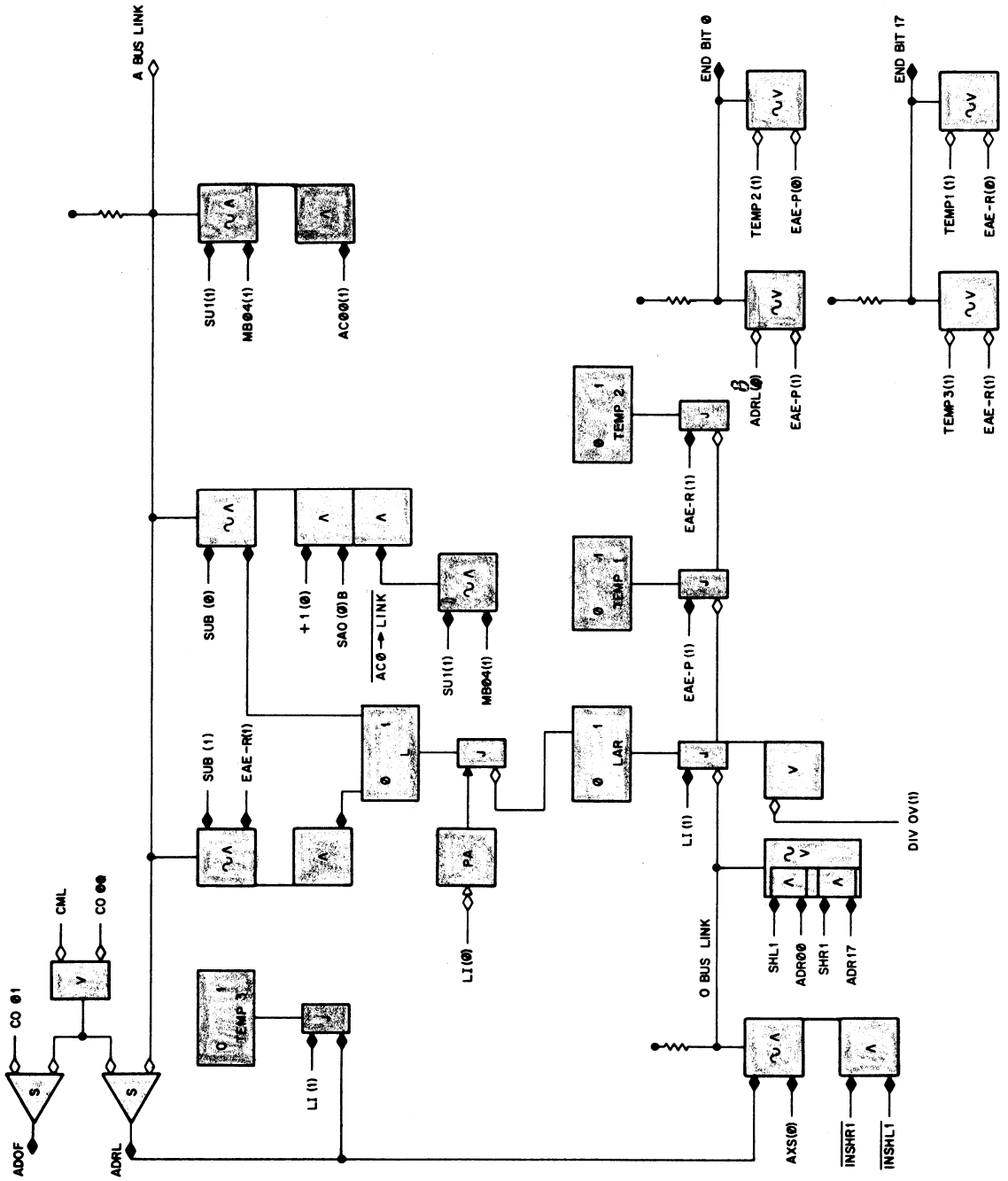
B-CS-S181-0-1 DC Carry Chain S181,
Circuit Schematic



B-CS-S206-0-1 Flip-Flop S206,
Circuit Schematic



B-CS-W005-0-1 Clamped Load W005,
Circuit Schematic



Link Control for EAE Instructions

digital

DIGITAL EQUIPMENT CORPORATION ▫ MAYNARD, MASSACHUSETTS

Printed in U.S.A.