MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

TX-2 TECHNICAL MANUAL

LINCOLN MANUAL NO. 44

Volume 2

JUNE 1961

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the joint support of the U.S. Army, Navy and Air Force under Air Force Contract AF 19(604)-7400.

LEXINGTON MASSACHUSETTS

TX-2 TECHNICAL MANUAL

TABLE OF CONTENTS

VOLUME I

CHAPTER 1 INTRODUCTORY DESCRIPTION

CHAPTER 2 FUNCTIONAL DESCRIPTION OF TX-2

CHAPTER 3 CIRCUIT LOGIC ELEMENTS

CHAPTER 4 MEMORIES

CHAPTER 5 TIMING AND CONTROL

CHAPTER 6 FUNCTIONAL ORGANIZATION OF THE CONTROL ELEMENT

CHAPTER 7 OPERATION CODES

VOLUME II

CHAPTER 8 PULSE AND LEVEL NOTATION

CHAPTER 9 COMPUTER DYNAMICS

CHAPTER 10 CONTROL ELEMENT

CHAPTER 11 MEMORY ELEMENT

CHAPTER 12 PROGRAM ELEMENT

CHAPTER 13 EXCHANGE ELEMENT

CHAPTER 14 ARITHMETIC ELEMENT

CHAPTER 15 IN-OUT ELEMENT

VOLUME III

CHAPTER 16 TIMING CHARTS

CHAPTER 8

PULSE AND LEVEL NOTATION

TABLE OF CONTENTS

8-1 INTRODUCTION

8-2 REGISTERS AND FLIP-FLOPS

8-3 GENERAL PULSE NOTATION

- 8-3.1 CLOCK PULSES
- 8-3.2 REGISTER DRIVER PULSES
- 8-3.3 GATED REGISTER DRIVER PULSES

8-4 GENERAL LEVEL NOTATION

8-5 PULSE AND LEVEL NOTATION EXAMPLES

- 8-5.1 REGISTER DRIVER PULSES
- 8-5.2 LEVELS

8-6 REGISTER DRIVER LOGIC EQUATIONS

8-7 SUMMARY

LIST OF FIGURES

8-1 E BIT SYMBOLOGY

8-2 E REGISTER SYMBOLOGY

8-3 EXAMPLES OF GATED RD PULSES

8-4 XAS LOGIC

CHAPTER 8
PULSE AND LEVEL NOTATION

8-1 INTRODUCTION

This chapter will discuss the kinds of pulse and level notation used in the following chapters. This notation is the kind that is found on the TX-2 block schematic drawings.

There are several types of computer notation. However, certain forms of notation appear over and over again and serve as the basis for the pulse and level notation.

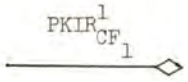
First, all the physical parts of the computer are identified, i.e., the flip-flops, registers, memories, etc. are given names. These names, wherever possible, are in the form of mnemonic abbreviations. However, since single letters are used to name registers, the mnemonic derivation is not always obvious.

Consider a typical build up of names and abbreviations. Normally the P register contains the address of the next instruction and the Q register the address of the next operand. PK (the P counter) distributes time levels during the instruction cycle and QK (the Q counter) distributes time levels during the operand cycle. Now consider the $PKIR_{CF}$ register. The roots of the abbreviation for this register are: PK (P counter), IR (instruction register), and CF (configuration). A "free" translation of the abbreviation might be: "The register in which the configuration bits are stored during the instruction cycle."

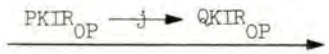
Once the significance of the $PKIR_{CF}$ abbreviation is known, it is natural to guess that $PKIR_{OP}$ is the register in which the operation code bits (OP) are held during the instruction cycle. Similarly, $QKIR_{OP}$ is the register in which the operation bits are held during the operand cycle. In this way a sizeable nomenclature is built up from a relatively small number of roots.

These names for the registers are in turn used as the roots in naming logical variables whose truth values depend on the state of associated registers, flip-flops, etc. For example, $PKIR^{1}_{CF_1}$ is the name of the logical variable which is "true" (i.e., has the logical value "ONE") when the first flip-flop in the $PKIR_{CF}$ register is a ONE. Assuming the significance of $PKIR_{CF}$ is known, it is only necessary to understand the effect of adding the subscript 1 and the superscript 1 on the $PKIR_{CF}$ root to form a comprehensive understanding of the full $PKIR^{1}_{CF_1}$ abbreviation.

The truth value of a variable can be determined in the computer by measuring the voltage on a wire whose voltage represents this variable, as described in Chapter 3. The relationship between the variable and the voltage is indicated in the following figure by labeling the wire with the variable and placing an arrowhead on the wire. If the arrowhead is hollow, then ground voltage on the wire corresponds to truth value for the variable. Similarly, a solid arrowhead indicated -3 volts on the wire corresponds to truth value for the variable.

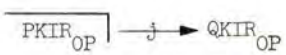


Still another type notation identifies the dynamic processes occurring in the computer. These processes are usually represented physically by 0.1 microsecond wide pulses on the wires which are labeled by this symbology. (Chapter 3 describes these dynamic processes.) For example, the following symbology is used to identify the process of jamming the contents of the $PKIR_{OP}$ register into the $QKIR_{OP}$ register and is represented by a 0.1 microsecond wide negative pulse on the associated wire.

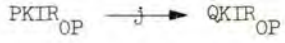


This pulse is usually the output of a register driver. The symbol is interpreted as the RD pulse which causes the contents of $PKIR_{OP}$ to be jammed (copied) into the $QKIR_{OP}$, i.e., the symbology "names" the pulse. This symbology brings up a convention which should be clarified. Normally the content of a register ( ) is symbolized by

However, what should be symbolized by



is frequently simplified to



The jargon used to describe the computer and its operation is based on the types of symbology just described.

Specifically, this chapter will discuss the notation for:

Register and Flip-Flops
Pulses
Flip-Flop Levels
Logic Net Levels
RD Logic Equations

8-2 REGISTERS AND FLIP-FLOPS

Most of the parts in the computer that are given logical identities are either flip-flops or assemblages of flip-flops. The assemblages are generally called registers.

Normally, the individual flip-flops never have single letter abbreviations. The mnemonics used to identify the flip-flops gives some hint of their function and sometimes indicates the type or subclass the flip-flop belongs to. Thus,

ST - STatus control flip-flop found in In-Out control units.

PI_3 - instruction cycle (P) interlock (I) flip-flop. Since there are more than one of these, the subscript indicates that this is the number three PI interlock.

EB - E register Busy interlock flip-flop. Since there is only one of these, no subscript is required.

The identification of flip-flops within a register is quite straightforward. The data registers such as A, B, C, D, E, etc. have ordered quarters and ordered bits within the quarter. The order reads from right to left. Fig. 8-1 shows the E bit symbology. The i.j flip-flop in this register, i.e., $E_{i.j}$, is the j-th flip-flop in the i-th quarter.

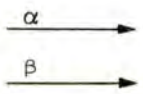
The counter registers are not quartered, so a single number ordering is sufficient. Most counters are made up of both an alpha and beta register. These are identical registers except the flip-flops in one are pulsed by alpha gated clock pulses and the flip-flops in the other are pulsed by beta gated clock pulses. $PK_{\alpha.3}$ and $PK_{\beta.3}$ are typical examples of flip-flops in the alpha and beta PK counter registers, respectively.

8-3 GENERAL PULSE NOTATION

There are three basic types of pulses:

1) Clock pulses.
2) Register driver pulses (gated clock pulses).
3) Gated register driver pulses (the pulse inputs that SET, CLEAR, and COMPLEMENT flip-flops).

8-3.1 CLOCK PULSES. These occur as a train of negative going pulses at 0.4 microsecond intervals. The β (beta) train of pulses lag the α (alpha) train of pulses by 0.2 microseconds. No identifying distinction is made between one alpha pulse and another or between one beta pulse and another. The notation for clock pulses is



It is important to realize that the alpha implies an uninterrupted train of alpha pulses and, similarly, that the beta implies an uninterrupted train of beta pulses.

8-3.2 REGISTER DRIVER PULSES. The pulses from any register driver occur at a specific time and initiate a specific process. For this reason, the symbol for the register driver pulse generally indicates, at least partially, the process initiated by the pulse. Register driver pulses are always negative. Two common types of register driver pulse notation are used depending on whether the process initiated does or does not involve an information transfer. Thus,

M $\xrightarrow{(\ )}$ E - is a pulse which transfers in some way the content of M into E.

$\vdash\!\xrightarrow{(\ )}$ E - is a pulse which SETS ($\vdash\!\xrightarrow{1}$), CLEARS ($\vdash\!\xrightarrow{0}$), COMPLEMENTS ($\vdash\!\xrightarrow{C}$), or PERMUTES ($\vdash\!\xrightarrow{p}$) the content of E.

The following are specific examples of register driver pulses taken from the TX-2 block schematics or from the timing charts in Chapter 17:

M $\xrightarrow{1}$ E - copies the contents of those flip-flops in the M register which contain ONES into the corresponding flip-flops in the E register, i.e., initiates a ONES transfer.

M $\xrightarrow{0}$ E - initiates a ZEROS transfer.

M $\xrightarrow{0,1}$ E - is really the symbol for two register driver pulses having the same input register driver logic, i.e., both register driver pulses are fired off at the same time even though they originate from different register drivers. The pulses initiate a ZEROS-ONES transfer.

M $\xrightarrow{j}$ E - initiates a jam (ZEROS, ONES) transfer.

Sometimes the type of input logic on the register driver producing the pulse is more completely identified by a subscript under the arrow. The subscript serves the additional function of hinting at the process in which the register driver pulse is used. Some specific examples of this are:

$\vdash\!\xrightarrow[se]{C}$ E - complements E "under sign extension control".

M $\xrightarrow[a_p]{0,1}$ E - copies the content of M into E "under permuted activity control". These pulses are used in the configuration process.

In these examples, the words "sign extension control" and "permuted activity control" are only meaningful when the person using the symbols has a detailed knowledge of the sign extension process and the configuration process (in which permuted activity takes place). These processes are discussed in detail in Chapter 13. Both sign extension control and permuted activity control take into account the configuration specified by the instruction. Fracture, activity, and permutation information are decoded from the configuration (CF) bits and combined with information decoded from the operation (OP) bits to generate configuration control levels. These levels find their way into the sign extension and permuted activity control nets. The output from these nets in turn find their way into the register driver logic initiating the pulses fired off during these processes.

Subscripting in the register abbreviations is used to indicate the specific quarters affected by the pulse. Fig. 8-2 shows how this notation is used in the permutation process.

8-3.3 GATED REGISTER DRIVER PULSES. These are the pulse inputs to the flip-flops themselves. Unlike the register driver pulses, these are positive going pulses. Usually these pulses are not distinguished by a name or notation of their own. They can be identified by examining the logic on the block schematics that produced them. Fig. 8-3 shows two examples.

8-4 GENERAL LEVEL NOTATION

Two basic types of level notation are used: one type identifies levels associated with flip-flops; the other type identifies levels associated with the output of logic nets. In the first type a superscript 0 or 1 is used to indicate the truth value of the variable, e.g., PI_3^0 or PI_3^1. In the second type the truth values are expressed by abbreviations with and without overbars, e.g., AEJ is an "Arithmetic Element Jump" level, while $\overline{AEJ}$ is a "not Arithmetic Element Jump" level. (The overbar is read as "not".) Logically, the overbar indicates the converse of the level represented by the abbreviation alone.

8-5 PULSE AND LEVEL NOTATION EXAMPLES

Typical examples of computer pulse and level notation are given below.

8-5.1 REGISTER DRIVER PULSES.

$PKIR_{CF_5} \xrightarrow{1} N_{2.9 - 1.5}$ - if $PKIR_{CF_5}$ contains a ONE, its contents are transferred into $N_{2.9 - 1.5}$. By means of this pulse, the sign bit of $PKIR_{CF}$ is expanded to fill 14 bits in N.

$\xrightarrow{PAD} A_i, C_i$ - initiates a "partial add" (PAD) which effects the i-th quarter of the A and C registers.

$\xrightarrow{0} N_{4,2,1}$ - clears the 1st, 2nd, and 4th quarter of the N register.

$\xrightarrow[ND]{SELECT} \diamond$ IOC - in the In-Out Element, 0.4 microsecond levels are pulses, hence the notation. The ND indicates that the pulse will go only to the specific IO unit determined by the output of the N Decoder, i.e., N_J determines the sequence. IOC is an abbreviation for "In-Out Control". SELECT hints at the function of the pulse.

$(PI_2^0 \cdot PKIR^{IND} + PI_5^1) \xrightarrow{j}$ XAS) - this is a somewhat unusual notation. Basically, the truth value of the statement on the left is copied into the XAS flip-flop. When either one of the terms in the bracket is true, XAS is set to ONE; if both are false, XAS is cleared to ZERO. (See Fig. 8-4.)

$\overline{PK|} + 1 \longrightarrow PK$ - indexes the PK counter by one, i.e., one is added to the contents of the PK counter by the pulse.

$\overline{PK|} + 1 \longrightarrow PK$ - does not index the P counter by one, i.e., the register driver pulse is not fired off. This notation is used to indicate inhibitory register driver logic.

$\xrightarrow{24} PK$ - presets the P counter to the PK^{24} time level state from whatever state it is in.

8-5.2 LEVELS.

A_i^1 - indicates the content of the i-th quarter of A is "all ONES".

f_i - indicates the fracture decoded from the configuration bits. There are four fractures: f_0 (36), f_1 (18,18), f_2 (27,9) and f_3 (9,9,9,9).

FD_i - indicates the count is "finished" in the i-th quarter of D.

IV - indicates the sign quarters of the subwords in the Arithmetic Element. In this case, the roman numeral indicates quarter 4 is the sign quarter.

$QKIR^{f_1 + f_2}$ - if either an f_1 or f_2 fracture is specified, this level will be decoded, during the operand cycle, from the contents of the QKIR register.

NP_{38}^{ev} - is generated by the instruction word (N) parity (P) check circuit. The subscript indicates that the parity count is taken over 38 bits. Whether an odd or even (ev) level is generated depends on the parity of the information in N.

$IOCM^{NORMAL}$ - is an abbreviation for an in-out-control-mixer level. The superscript is one of several and hints at the logical function of the level. The level is associated with the In-Out Element and is bound on the IOCM Bus.

$PK^{02\alpha}$ - is an alpha PK time decoder.

$PK^{02\beta}$ - is a beta PK time decoder. It occurs 0.2 microsecond after $PK^{02\alpha}$.

$PI^{START}{}_1$ - is an interlock level associated with the instruction cycle. Specifically, it is one of two start interlock levels involved in the logic for starting the PK counter.

$K^{eq\ JC}$ - is generated when the number of the new sequence specified by the output of the J coder is the same as the number of the current sequence specified by the contents of the K register.

8-6 REGISTER DRIVER LOGIC EQUATIONS

Boolean algebra equations are used to describe the way in which levels gate pulses in register drivers.

Consider the equation:

$$\alpha \cdot \overline{PI^{WAIT}} \cdot (PK^{22\alpha} \cdot PI^{0}_{2} + PK^{23\alpha} \cdot CSK^{11\alpha}) = \lfloor^{24}\!\rightarrow PK$$

CLOCK PULSE — LEVEL LOGIC — RD INPUT LOGIC — RD PULSE OUTPUT

An alpha clock pulse is ANDed with assorted level logic. When this level logic is satisfied, the clock pulse will be gated and given the name $\lfloor^{24}\!\rightarrow PK$. Note that in this equation both the alpha pulse and the $\overline{PI^{WAIT}}$ level are necessary conditions for generating ^{24} PK. The The equality sign (=) indicates that when one side of the equation is true, the other side of the equation is also satisfied.

Consider now how the above equation can be broken down into two other equations.

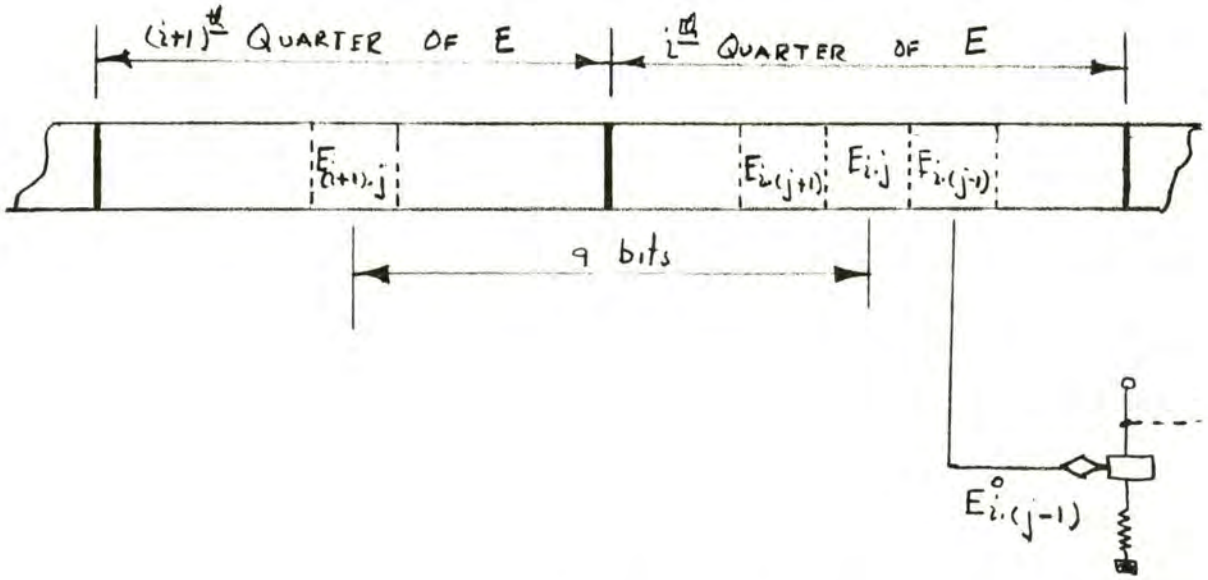
$$\alpha \cdot \overline{PI^{WAIT}} \cdot PK^{22\alpha} \cdot PI^{0}_{2} \supset \lfloor^{24}\!\rightarrow PK$$

$$\alpha \cdot \overline{PI^{WAIT}} \cdot PK^{23\alpha} \cdot CSK^{11\alpha} \supset \lfloor^{24}\!\rightarrow PK$$

In this case an implication sign ($\supset$) is used instead of an equality sign (=). The fact that the left hand side of the equation is satisfied "implies" that the right hand side is also satisfied, i.e., the fact that the left hand side of the equation is true is sufficient to make the right hand side also true. But, in this case the converse is not true, i.e., the fact that the right hand side of the equation is true is not sufficient to make the left hand side also true.

8-7 SUMMARY

It is important to realize that the significance of a given pulse or level lies strictly in the specific logic that produced it. The notation tries in a systematic way to hint at this logic. E.g., $PI^{CH\ SEQ}$ can be interpreted as an instruction interlock level calling for a change of sequence, but the full significance of $PI^{CH\ SEQ}$ can only be determined by examining the logic that produced the level. Similarly, the function of the level can only be determined by examining <u>all</u> the logic in which the level is used.



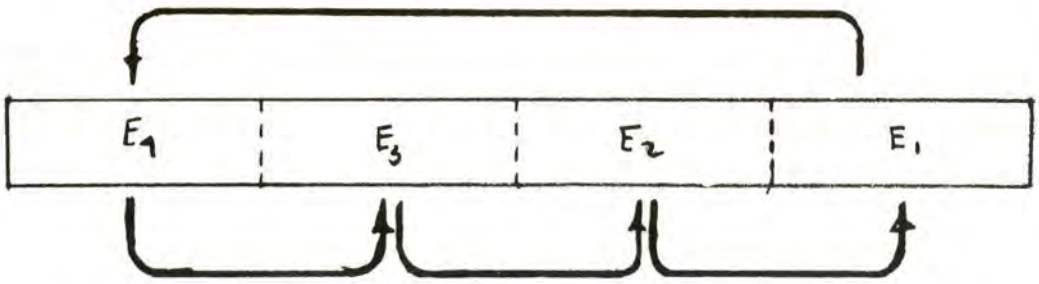
$E_{i,j}$ IS THE NAME OF THE j^{th} FLIP FLOP IN THE i^{th} QUARTER OF THE E REGISTER. NUMERICALLY i IS 1, 2, 3 OR 4 AND j IS 1, 2, 3, ... OR 9. THUS IF,

	$i=2$ and $j=5$	$i=4$ and $j=9$
$E_{i,j}$	$E_{2,5}$	$E_{4,9}$
$E_{i,(j-1)}$	$E_{2,4}$	$E_{4,8}$
$E_{i,(j+1)}$	$E_{2,6}$	$E_{1,1}$ (36), $E_{2,1}$ (27, 9*), $E_{3,1}$ (18, 18*), $E_{4,1}$ (9, 9, 9, 9*)
$E_{(i+1),j}$	$E_{3,5}$	$E_{1,9}$

* FRACTURE

$E°_{i,(j-1)}$ IS THE NAME OF THE VARIABLE REPRESENTING THE ZERO STATE OF THE $E_{i,(j-1)}$ FLIP FLOP. IN THE ILLUSTRATION ABOVE, THE ZERO STATE OF THE FLIP FLOP IS INDICATED AT THE TRANSISTOR BY A GROUND VOLTAGE.

FIG. 8.1 E BIT SYMBOLOGY



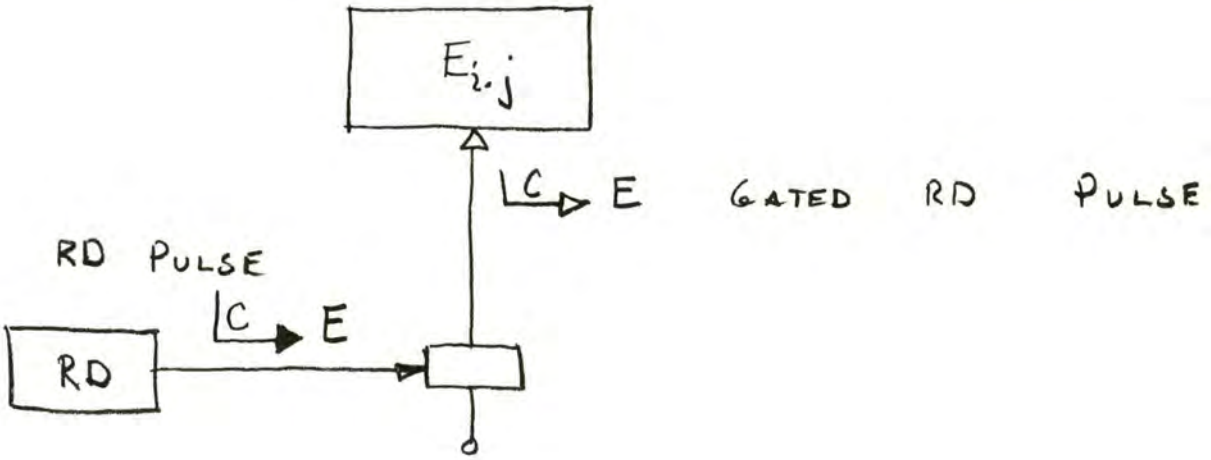
$E_{i+1} \xrightarrow{j} E_i$ (where $i = 1, 2, 3, 4$)

$E_{i+1} \xrightarrow{1} E_i$ IS THE SYMBOL FOR A <u>REGISTER DRIVER PULSE</u> WHICH CAUSES THE CONTENTS OF ALL THE FLIP FLOPS IN THE $(i+1)^{st}$ QUARTER OF E TO BE COPIED INTO THE CORRESPONDING FLIP FLOPS IN THE i^{st} QUARTER OF E. THE PULSE SYMBOL IS ALSO AN ACCURATE SYMBOLIC REPRESENTATION OF THE PROCESS EFFECTED BY THE PULSE.

<u>EXAMPLE</u>

	E_4	E_3	E_2	E_1
BEFORE PULSE	372	957	612	207
AFTER PULSE	207	372	957	612

FIG. 8.2 E REGISTER SYMBOLOGY



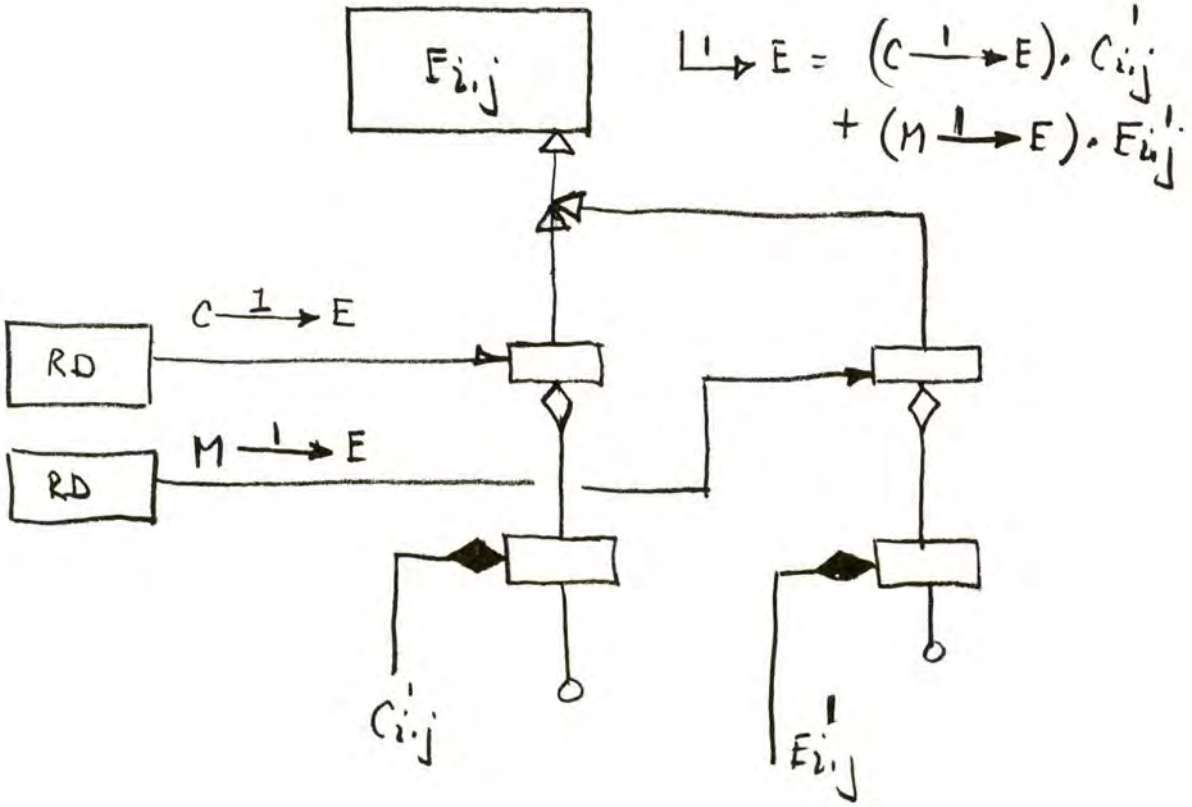


FIG. 8-3 EXAMPLES OF GATED RD PULSES.

TRUTH TABLE			
PI_2	$PKIR^{IND}$	PI_5	XAS
PI_2^0	$PKIR^{IND}$	PI_5^0	XAS^1
PI_2^0	$\overline{PKIR^{IND}}$	PI_5^0	XAS^0
PI_2^1	$PKIR^{IND}$	PI_5^0	XAS^0
PI_2^1	$\overline{PKIR^{IND}}$	PI_5^0	XAS^0
PI_2^0	$PKIR^{IND}$	PI_5^1	XAS^1
PI_2^0	$\overline{PKIR^{IND}}$	PI_5^1	XAS^1
PI_2^1	$PKIR^{IND}$	PI_5^1	XAS^1
PI_2^1	$\overline{PKIR^{IND}}$	PI_5^1	XAS^1

LOGICAL SIGNIFICANCE OF

$[PI_2^0 \cdot PKIR^{IND} + PI_5^1] \dashv\!\!\rightarrow XAS$

NOTATION

Fig. 8-4 XAS Logic

CHAPTER 9

COMPUTER DYNAMICS

TABLE OF CONTENTS

9-1 INTRODUCTION

9-2 INSTRUCTION CLASSIFICATION BY COUNTER ACTIVITY

- 9-2.1 CLASS A INSTRUCTIONS
- 9-2.2 CLASS B INSTRUCTIONS
- 9-2.3 CLASS C INSTRUCTIONS
- 9-2.4 SUB-CLASSIFICATION OF INSTRUCTIONS

9-3 EFFECT OF MEMORIES ON PK AND QK COUNTER ACTIVITY

- 9-3.1 MEMORY OVERLAP
- 9-3.2 MEMORY CYCLE TIME
  - 9-3.2.1 PK CYCLES
  - 9-3.2.2 QK CYCLES
- 9-3.3 EXAMPLES OF ELAPSED INSTRUCTION TIME AS A FUNCTION OF MEMORY LOCATION AND INSTRUCTION TYPE

9-4 SEQUENCE DYNAMICS

9-5 COUNTER DYNAMICS WHEN NO CHANGE OF SEQUENCE (CSK) OR DELAY SYNCHRONIZATION CYCLE(S) ARE INVOLVED

- 9-5.1 COUNTER STARTING CONDITIONS
  - 9-5.1.1 PI^{START}_{1}
  - 9-5.1.2 QI^{START}
  - 9-5.1.3 START XWK
  - 9-5.1.4 START FK
  - 9-5.1.5 START AK
- 9-5.2 SIMPLE PK AND QK WAITING LOGIC

9-6 COUNTER DYNAMICS WHEN A TRANSITION TO OR FROM A CHANGE OF SEQUENCE (CSK) OR DELAY SYNCHRONIZATION CYCLE (DSK) IS INVOLVED

- 9-6.1 DECISION AND WAITING LOGIC
- 9-6.2 CSK AND DSK COUNTER STARTING CONDITIONS

9-7 DEFERRED ADDRESSING CYCLES

9-8 IN-OUT TIME CONSIDERATIONS

9-9 PROGRAM EXAMPLE

LIST OF FIGURES

9-1 BASIC INSTRUCTION CLASSIFICATION BY COUNTER ACTIVITY

9-2 INSTRUCTION CLASSIFICATION BY COUNTER ACTIVITY

9-3 EFFECT OF MEMORY OVERLAP AND NO MEMORY OVERLAP SWITCH ON CLASS C1 INSTRUCTIONS

9-4 INFLUENCE OF MEMORIES AND INSTRUCTIONS ON PK AND QK COUNTING CYCLES

9-5 SUCCESSION OF IDENTICAL INSTRUCTIONS WITH INSTRUCTION AND OPERAND WORDS STORED IN DIFFERENT MEMORIES

9-6 POSSIBILITIES FOR TRANSITION FROM ONE INSTRUCTION TO THE NEXT INSTRUCTION
9-7 STARTING LOGIC FOR PK, QK, XWK, FK AND AK COUNTERS
9-8 PK, XWK, QK AND FK INTERLOCK EVENTS
9-9 SIMPLE WAITING STATE LOGIC
9-10 PI_3 AND CSK_4 INTERLOCK STATES FOR TRANSITION POSSIBILITIES FROM ONE INSTRUCTION TO THE NEXT
9-11 PK AND DSK DECISION LOGIC
9-12 CSK AND DSK STARTING CONDITIONS
9-13 CSK AND DSK INTERLOCK EVENTS
9-14 DEFERRED ADDRESS CYCLES
9-15 COUNTER ACTIVITY FOR SIMPLE PROGRAM EXAMPLE

CHAPTER 9
COMPUTER DYNAMICS

9-1 INTRODUCTION

This chapter will develop a detailed picture of the dynamic operation of the computer. The occurrence of events, i.e., pulse inputs to flip-flops, is determined by the occurrence of counter time levels. Bar graphs will be used to express the dynamic picture of counter activity. The bar graphs will show the operation and interlocking of the control counters which generate the time levels.

Two types of dynamic pictures are of interest: one type shows in detail the counter activity required to execute a specific instruction; the other type takes a broader view and looks at the counter activity occurring while a sequence of instructions is executed. In the process of developing these two kinds of pictures the chpater will answer the following types of questions:

1) During a given instruction, what specific counters will run? When will they start? How long will they run? What specific time states will the counters pass through?
2) When, during the execution of the current instruction, can the next instruction begin? What effect does the memory location of the instruction word and operand word have on this decision?
3) Where in the current instruction are decisions made that determine whether the computer will: (a) go on immediately to the instruction in the current sequence, (b) wait for awhile before going on in the current sequence, (c) change immediately to a different sequence, or (d) wait and then change to a different sequence? What specific factors determine these decisions?
4) What types of decisions does the computer make if the current instruction is held up because the execution logic requires some part of the computer that is currently busy with a previous instruction? E.g., what does the computer do if the Arithmetic Element is tied up executing a MULtiplication at the time an ADD instruction desires to use the Arithmetic Element?

This chapter will first classify all the operation codes according to the basic counter activity pattern required by the execution logic of the operation code. This picture establishes what counters are used by what operation codes and when the counters start with reference to the running of other counters.

Next, the effect of the memory location of the instruction word and operand word on the counter activity pattern will be discussed.

A third type picture will show the counter activity pattern for a sequence of instructions.

With these types of general pictures established, the chapter will then discuss in detail the logic that specifically determines the pattern of counter activity. First, the starting logic for each counter will be discussed. Then the logic determining the synchronization delays and change of sequence processes will be discussed.

The chapter will conclude with an example showing the pattern of counter activity for a specific sequence of instructions.

Logical definitions and descriptive discussions of the interlock levels used in this chapter can be found in Chapter 10.

9-2 INSTRUCTION CLASSIFICATION BY COUNTER ACTIVITY.

It is convenient to think of all the instructions as belonging to one of three basic classes, depending on the use they make of the PK and QK counters. The operation codes determine the class of instructions in that they determine the use of the counters. The use of the counters, and the corresponding operation codes are shown in Fig. 19-1.

9-2.1 CLASS A INSTRUCTIONS. These instructions are characterized by having no operand memory cycle, i.e., a QK cycle does not occur. A further peculiarity of this class is that the PK cycle always terminates in $PK^{31\alpha}$, instead of $PK^{24\alpha}$. States PK^{25} through PK^{31} are called the "execute instruction", or PKEI states. The execution logic for the jump instructions, which for the most part make up this class, requires this PKEI cycle. (Note that all three classes have a memory or PKM cycle which extends from PK^{00} through PK^{22}, and an added state PK^{24}.)

9-2.2 CLASS B INSTRUCTIONS. These instructions are like the Class A instructions in that they have a PKEI cycle. They are unlike the Class A instructions in that an operand word is obtained from memory during a QK cycle. Except for TSD, these are skip type instructions. Thus, most Class A and B instructions involve a possible change in the contents of the P register during the PKEI cycle.

9-2.3 CLASS C INSTRUCTIONS. This class contains the majority of instructions. In these instructions, PK terminates in PK^{24}, and an operand word is obtained from memory by a QK cycle.

9-2.4 SUBCLASSIFICATION OF INSTRUCTIONS. The three classes can be usefully broken down into subclasses which bring out in greater detail the activity of the PK, QK, FK, XWK, AK and ASK counters. This has been done in Fig. 9-2.

It should be noted that all instructions, except SKM and OPR, which use XWK at PK^{14} and do not again use XWK, use the contents of the X register for address modification. Also, all instructions, except FLF, FLG, SPG and SPF, which use FK use the contents of $QKIR_{CF}$ for standard configuration control.

The usual times for starting XWK and FK are PK^{14} and QK^{00}, respectively.

CLASS A

Class A1 (JMP, SKX, JPX, JNX). The XWK counter is not started until PK^{31} in these instructions. Note that if $PKIR^{1}_{CF_2}$, the execution logic for JMP does not require XWK at this time. None of these instructions uses the FK counter.

Class A2 (IOS). The XWK counter is started at the usual time. PK must wait in PK^{25} for EB^{0} in this instruction.

Class A3 (AOP). AOP starts the AK counter in PK^{26}. Because the instruction uses the Arithmetic Element beginning at PK^{26}, PK must wait in PK^{25} until the Arithmetic Element is free, i.e., the $\overline{AEB}$ condition exists. The last AK state used depends upon the Arithmetic Element instruction specified by the AOP.

Class A4 (JOV, JPA, JNA). This class uses FK, which can be started at PK^{13}. FK is started by an interlock start condition, and might not start immediately at PK^{13}. XWK is started at the usual time. PK must wait in PK^{25} until the waiting condition is satisfied.

CLASS B

Class B1 (SKM). XWK is started at the usual time. SKM does not use the FK counter. PK must wait in PK^{25} until QK reaches QK^{14}.

Class B2 (TSD). XWK and FK are started at the usual times. PK must wait in PK^{25} until QK reaches QK^{01}, if PI^{1}_{4}, or QK^{20}, if PI^{0}_{4}.

Class B3 (SED). XWK and FK are started at the usual times. PK must wait in PK^{25} until QK reaches QK^{14}.

CLASS C

Class C1 (LD-, ST-, DPX, ADX, ITA, ITE, UNA, EXA, INS, COM). These are "typical" instructions in that the PK cycle terminates in PK^{24}; a PK and a QK cycle occur; and XWK and FK are used and started at the usual times. Note that QK is loosely interlocked with PK^{24} (dashed line). The dashed line indicates that PK^{24} is the earliest time at which QK can begin. It is possible that QK may not actually start until later when certain other conditions are satisfied; e.g., QK may not have completed its cycle from the previous instruction when PK^{24} occurs in the current instruction.

Class C2 (SPF, SPG). XWK is started at the usual time. FK is started at QK^{13}. FK is not used for configuration control. The length of the FK cycle is determined by the operation code.

Class C3 (FLF, FLG). XWK is started at the usual time. FK is started at PK^{13}. FK is not used for configuration control. Note that QK is interlocked with FK and PK.

Class C4 (DSA, ADD, SUB). This Class is like Class C1, except that the AK counter is started at QK^{14}.

Class C5 (CY-, SC-, NO-, DIV, MUL, TLY). This class is like Class 4, except that the ASK counter is used as well as the AK counter. AK makes several iterated subcycles. With the exception of the CYcle and SCale instructions, the number of subcycles is determined or limited by ASK. (The operation of the ASK counter is described in Chapters 10 and 14.)

Class C6 (AUX, RSX, EXX). This class is like Class C1, except that an XWK cycle occurs in the QK cycle as well as in the PK cycle. The interlocks set by XWK can be the crucial factor determining when the next instruction can begin.

While Fig. 9-2 shows the general pattern of counter activity for the different classes of instructions, the specific picture depends on a variety of conditions that will be pointed out as the chapter develops.

9-3 EFFECT OF MEMORIES ON PK AND QK COUNTER ACTIVITY

The locations in memory of the instruction and operand words have an important effect on computer timing. This section will examine this effect.

The two basic situations that can exist are: (1) the operand words and instruction words are stored in the same memory, or, (2) conversely, the operand and instruction words are stored in different memories. In the first situation no overlap can exist between the operand and instruction cycles, while in the second situation an overlap is permitted.

9-3.1 MEMORY OVERLAP.

Fig. 9-3 shows the effect of memory overlap on Class C1 instructions. It should be noted that "memory overlap" refers to the overlap of the memory cycle for the next instruction word with the memory cycle for the current operand word, and not to the overlap of the current operand word with the current instruction word.

In Fig. 9-3(a), the instruction and operand words are stored in different memories. For this reason, the PK cycle of the next instruction can begin before the QK cycle of the current instruction ends. In Fig. 9-3(b), the instruction and operand words

are stored in the same memory (or the No Overlap interlock flip-flop is set to ONE (NO^1)). Note that a sequence of instruction cycles are considerably more compressed in time in Fig. 9-3(a) than in Fig. 9-3(b).

9-3.2 MEMORY CYCLE TIME. We shall now examine in some detail the operand and instruction word cycles. The following general facts should be kept in mind:

1) Each of the four memories, i.e., S, T, V_{FF} and V^T, have a basic instruction and operand word memory cycle time. Except for the V memories, the basic instruction and operand memory cycles for each memory are the same. However, the memory cycles differ among the memories. For example, the basic T Memory cycle is "faster" than the S Memory cycle. Fig. 9-4(a) tabulates the basic memory cycle times for each memory.

2) The actual elapsed PK and QK time is a function of the instruction itself as well as the memories used to store the instruction and operand words. This is the reason for speaking of a "basic" memory cycle time. The basic memory time is the time required to read a word out of memory and write the same word back into memory. Time may be consumed in the PK and QK cycles performing non-memory functions required by the execution logic of the instruction. Note that in the case of the PK cycle, this non-memory time always comes after the PKM cycle, i.e., after PK^{22}. In the case of the QK cycle, the non-memory time always comes in the middle of the QKM cycle, resulting in an "extended" QKM cycle. This time must be added to the basic memory time. Fig. 9-4(b) and 9-4(c) show the actual PK and QK time levels required as a function of memories and instructions.

The PK and QK counters proceed by a sort of "hop and skip" process. E.g., once the PK counter begins counting it hops from state to state because of the PK + 1 PK register driver pulses. Suppose that the instruction is stored in the S Memory; PK will hop from PK^{00} on up to PK^{06}. At PK^{06}, inhibitory logic will cause a PK + 1 PK condition to exist. PK can now proceed only by skipping into a "preset" state. In this case, a 09 PK pulse skips PK into PK^{09}. The bold face states on Figs. 9-4(b) and 9-4(c) indicate the "preset" states of the counter (see Chapter 10). Skipping always occurs into these "preset" states. PK now continues hopping from state to state up to PK^{16}. It then skips from PK^{16} to PK^{22} and (usually) then skips again from PK^{22} to PK^{24}.

The rules governing the skipping of counter time states are:

1) All skips are made to preset time states.

2) The skip is always in the forward direction (unless the execution of an instruction is abandoned and a change of sequence cycle occurs).

3) Skips are usually to the next preset state.

These rules are not inviolable, as indicated by the dashed lines on Figs. 9-4(b) and 9-4(c).

9-3.2.1 PK CYCLES. The variations possible during PK cycles are illustrated in Fig. 9-4(b). When an instruction word is obtained from memory, PK performs the basic memory cycle (PKM) as it runs from PK^{00} through PK^{22}. The succession of states followed is determined entirely by the memory involved. For example, if the instruction word is obtained from the A register, then a $PKM^{V}FF$ cycle is performed which is made up of PK states 00, 01, 02, 09, 10, 11, 12, 13, 14, 15 and 22. Thus the cycle lasts for 4.4 microseconds. Different PKM cycles are required by the other memories. If deferred address words are required by an instruction, then PK will go through similar cycles. The last deferred address word memory cycle will be followed by one more final PK cycle which does not use any memory and during which PKA^{0}. This final cycle always uses states 00, 01, 09, 10, 11, 12, 13, 14, 15 and 22.

After the instruction word memory cycle, if no deferred addresses are required, or after the final deferred address cycle (the one which does not use any memory), if deferred addresses are required then PK will attempt to enter PK^{24} from PK^{22}. Two situations can then occur:

1) Interlock conditions may require that the computer abandon the attempt to execute the instruction and instead perform a change of sequence cycle. In this case PK will go from PK^{22} back to PK^{00}. (A similar situation may arise during any $PKM^{V}FF$ cycle at PK^{02}. In this case PK goes from PK^{02} back to PK^{00}.)

2) On the other hand, PK may have to wait before a decision can be made as to whether to proceed executing the current instruction or to abandon the current instruction, i.e., perform a change of sequence cycle. In this situation PK goes from PK^{22} to PK^{23} and waits in this state while the delay synchronization counter (DSK) performs a number of cycles. If, eventually, a change of sequence cycle occurs, PK will go from PK^{23} back to PK^{00}. In this case the instruction is not executed and is abandoned. On the other hand, if the interlock control decides that PK need wait no longer and no change of sequence is required, then PK will finally proceed from PK^{23} to PK^{24}.

In the one case where PK can proceed from PK^{23} to PK^{24}, and in the other case where interlock conditions permit PK to proceed directly from PK^{22} to PK^{24} without any complications, the computer finally reaches a state from which it can proceed to execute the remainder of the instruction. Up to this point, all decisions have been made without regard to the class of the instruction. However, decisions about the succession of counter states are hereafter strongly influenced by the class of the instruction.

9-3.2.2 QK CYCLES. The variations possible during the QKM cycle are illustrated in Fig. 9-4(c). Note that the QK cycle always terminates in QK^{31}.

The basic QKM cycle for the V_{FF} memory involves states 00, 01, 02, 03, 09, 10, 11, 13, 14, 21, 22, 23 and 31. The other memories require different QKM cycles, which are again further modified by the requirements of the instruction being executed. In a memory modification type instruction, such as COM, the basic memory cycle may be "extended" by the insertion of intermediate states. This allows the word read out of memory to be modified before it is written back into memory. For example, in all non-load ($\overline{QKIR^{LOAD}}$) instructions involving the S and T memories, QKM is extended 0.8 microseconds by QK^{24} and QK^{25}. QKM can also be lengthened by the QK^{03} waiting state conditions. These can arise only when the operand word is located in the V_{FF} memory.

9-3.3 EXAMPLES OF ELAPSED INSTRUCTION TIME AS A FUNCTION OF MEMORY LOCATION AND INSTRUCTION TYPE. Three examples will be given illustrating the elapsed time required by a program consisting of a repetition of identical instructions.

Figs. 9-5(a) and 9-5(b) show a repetition of LOAD type instructions. In these two cases the PK cycle time is equal to the PKM time (see Fig. 9-4(a)) plus 0.4 microsecond for PK^{24}, while the QK cycle time is simply the basic QKM time.

In Fig. 9-5(a) the instruction words are located in the T Memory and the operand words are located in the S Memory. In this case (PKM^T + 0.4) QKM^S, since PKM^T = 4.4 microseconds and QKM^T = 6.4 microseconds. Note that QK cycles continuously, i.e., QK^{00} (which is the normal resting state) lasts only 0.4 microsecond. PK, on the other hand, rests in PK^{00} at the end of each PK cycle waiting for QK^{00} to occur. Note, also, that the first instruction time (4.8 microseconds) is shorter than the succeeding instruction times (6.4 microseconds).

In Fig. 9-5(b) the instruction words are located in the S Memory and the operand words are located in the T Memory. (The converse of the case shown in Fig. 9-5(a)). In this case PKM^S + 0.4 QKM^T, since PKM^S = 6.4 microseconds and QKM^T = 4.4 microseconds. Note that in this case, PK cycles continuously, while QK rests in QK^{00} waiting for PK^{24} to occur. Each instruction, including the first one, takes 6.8 microseconds.

The saving in time realized by storing the instruction words in the T Memory and the operand word in the S Memory, rather than vice versa, is thus approximately 0.4 microsecond per instruction. In either case, however, 6.4 microseconds is saved when compared with the case where both instructions and operands are located in the S Memory.

Fig. 9-5(c) illustrates the case where $PKM^S + 0.4 = QKM^T$, i.e., where the PK and QK cycles are the same length, by a sequence of INSert instructions. In the example, $PKM^S = 6.4$ microseconds and $QKM^T = 6.8$ microseconds. Note that both PK and QK cycle continuously and that each instruction takes 6.8 microseconds to execute. It can also be deduced from Fig. 9-4 that a series of INS instructions in which the instruction words were stored in the T Memory and the operand word were stored in the S Memory would require 8.8 microseconds per instruction.

9-4 SEQUENCE DYNAMICS

Thus far in the chapter counter activity patterns have been established for individual instructions (or at most for an uninterrupted succession of instructions in the same sequence). This section will take a broader view and consider all the basic possibilities for getting from one instruction to the next, and the next instruction in the same sequence or in another sequence.

Fig. 9-6 shows the four basic possibilities for proceeding from one instruction to the next. The normal situation is for the current instruction to be followed by another instruction in the same sequence. By the end of the current PK cycle a definite decision has been made to continue in the current sequence. As was pointed out earlier in the chapter, this does not necessarily mean that the next PK cycle will begin as soon as the current PK cycle is completed, i.e., PK will wait in its resting state, PK^{00}, until all the necessary interlock conditions for continuing are satisfied.

The second possibility is that a decision to change sequence will be made during the current instruction. The current instruction may or may not be completed before the change of sequence cycle begins. In any event, the change of sequence cycle cannot begin until the PK counter is in its PK^{00} resting state.

The third and fourth cases result from two different basic situations. Either some interlock condition has forced the PK counter to wait in PK^{02} or PK^{23}, or the current instruction has dismissed the sequence and PK is waiting in its PK^{00} resting state until a decision can be made as to whether to begin another instruction in the current sequence or to change sequence. During this waiting period a series of delay synchronization cycles are executed which examine the interlock conditions upon which the decision is based.

In case three, a decision is eventually made to go on in the current sequence. PK will either complete the current instruction or, if the current instruction has been completed, begin the next instruction in the current sequence.

In case four, a decision is eventually made to change sequence. If PK is in either the PK^{02} or PK^{23} waiting states, the current instruction will be abandoned. In this case PK will go back to PK^{00} and a change of sequence cycle will occur. If the delay synchronization cycle occurs while PK is in PK^{00}, the delay synchronization cycle in which the change of sequence decision is made will simply be followed by a change of sequence cycle, while PK remains in PK^{00}.

9-5 COUNTER DYNAMICS WHEN NO CHANGE OF SEQUENCE (CSK) OR DELAY SYNCHRONIZATION CYCLE(S) (DSK) ARE INVOLVED

This section will discuss Case 1 in Fig. 9-6 in detail, i.e., the case in which a series of instructions in the same sequence is executed. The counter activity patterns for the instructions themselves can have any of the forms shown on Fig. 9-2. It is necessary to examine only the counter starting conditions for PK, QK, XWK, FK and AK to determine the specific relative starting time of each counter for all the instruction variations. In order to establish an exact counter activity pattern, it is necessary to know (in addition to the counter starting conditions) the PK and QK time states used by the instruction and the time states in PK and QK in which waiting can occur.

Fig. 9-4 shows the PK and QK time states required as a function of instruction and memory. Waiting can occur in PK^{02}, PK^{25} and QK^{03}. The conditions under which simple waiting occurs in these states will be examined in this section. (Waiting can also occur in the 00 resting state of the PK and QK counters, but this is reflected in the PK and QK counter start interlock logic.)

9-5.1 COUNTER STARTING CONDITIONS. Fig. 9-7 shows the interlock start conditions for PK, QK, XWK, FK and AK. In this section, the following assumptions are made:

1) The $PI_3^0 \cdot CSK_4^0$ interlock condition is satisfied. PI_3 and CSK_4 are only of importance when a change of sequence or delay synchronization has just previously occurred. This will not be the case in this section.

2) All the alarm and pushbutton control conditions are satisfied, i.e., $\overline{AL}$, $START_2^1$, PKS_1^0 and PKS_2^0.

Fig. 9-8 shows the times at which the various interlocks involved in the counter starting conditions are set and cleared. The setting and clearing times are given as a function of instruction and counter. The time at which an interlock of interest is set or cleared is given at the intersection of the interlock column and the instruction row.

Even though all the interlock start conditions are satisfied, a counter will not start a new cycle until it is in its 00 resting state. Conversely, even though the counter is in its 00 resting state, it will not start a new cycle until the interlock start conditions are satisfied.

9-5.1.1 $PI^{START}1$. PK begins counting when either an instruction word is called for (PI_2^0) and the $PI^{START}1$ conditions are satisfied or when a deferred address word is called for (PI_2^1) and the $PI^{START}2$ conditions are satisfied. The deferred address situation will be considered a special topic and discussed at the end of the chapter. The primary interlocks of interest are PI_1, XB and QB.

$PI_3^0 \cdot CSK_4^0$ - The assumption in this section is that this interlock condition is satisfied.

PI_1^0 - PI_1 is set during the PK cycle in those instructions that have an operand (QK) cycle. It is then cleared in the QK cycle that follows. For most instructions, PI_1 is cleared to ZERO in QK^{00}. Some of the instructions that use the X and F memories for special purposes clear PI_1 later in the QK cycle. This is done in these cases to prevent the PK cycle from starting until the current QK cycle is finished with the X and F memories.

XB^0 - XB is set in the PK cycle at $PK^{12\alpha}$ and, in a few instructions, in the QK cycle at $QK^{13\alpha}$. The X write cycle that follows clears this interlock at $XWK^{02\alpha}$. XB^0 predicts that the X register will shortly be free. PK cannot start until there is assurance (XB^0) that the X Memory will be free at the time that it is required in the PK cycle.

QB^0 - For those instructions that have an operand (QK) cycle, QB is set at QK^{00}. It is then cleared in these instructions at $QK^{31\alpha}$. QB^1 prevents the PK cycle from starting until the current QK cycle is completed unless the computer is allowed to operate in the memory overlap condition, i.e., $NO^0 (\overline{P^S \cdot Q^S + P^T \cdot Q^T + P^U \cdot T^U + P^V \cdot Q^V})$.

The remainder of the logic in the $PI^{START}1$ level is normally satisfied. It is covered in detail in Chapter 10.

9-5.1.2 QI^{START}. This is the start interlock level for the QK counter when an operand word is called for. The primary interlocks of interest are PI_1 and FI.

PI_1^1 - This interlock is always set at $PK^{22\alpha}$ in those instructions that call for an operand. (See PI_1^0 discussion above.)

FI^1 - FI is cleared in the JPA, JOV, JNA, FLF, and FLG instructions in $PK^{13\alpha}$ at the time the FK counter is started. It is then set during the FK cycle. FI^0 prevents QK from starting if the FK counter is not available for configuration control during ordinary operand cycles.

The remainder of the logic in the QI^{START} level is normally satisfied. It is covered in detail in Chapter 10.

9-5.1.3 START XWK. XWK is normally started at $PK^{14\alpha}$ when the base address indexing process occurs. In certain of the jump instructions that use the X Memory for a different purpose, XWK is started at $PK^{31\alpha}$, i.e., at the end of the PKEI phase of the PK cycle. XWK is also started in the AUX, RSX and EXX instructions during the operand (QK) cycle when the contents of an X Memory register is being changed.

9-5.1.4 START FK. FK is normally started at QK^{00} as part of the configuration control process. In SPF and SPG, where the F Memory is used for non-configuration purposes, FK is started at $QK^{13\alpha}$. In FLF, FLG, JOV, JPA and JNA, where the F Memory is again used for non-configuration purposes, FK is started at $PK^{13\alpha}$. In this last case, FK starts because FI is cleared at $PK^{13\alpha}$. Note that the E register must be free (EB^{0}) before FK starts because the execution logic of these instructions uses the E register in the same process in which the F Memory is used.

9-5.1.5 START AK. AK is normally started at $QK^{14\alpha}$ in those instructions that use the AK counter in their execution logic. In the AOP instruction, AK is started at $PK^{26\alpha}$.

9-5.2 SIMPLE PK AND QK WAITING LOGIC. In addition to the normal waiting that can occur in the 00 resting state of the PK and QK counters, waiting can occur in PK^{02}, PK^{25} and QK^{03}. The interlock logic that causes this waiting is shown on Fig. 9-9. The basic reason for waiting is that the current cycle of the computer wants to use a part of the computer that is not currently available. The cycle waits in the "waiting state" until the cycle can go on. Note in these cases that there is no question of whether the cycle will or will not go on. The only question is when the cycle can proceed. PK waits in $PK^{02\alpha}$ if the selected word is located in the E register ($PKM^{V}FF \cdot \overline{VMD^{AE}}$) and either the E register is busy (EB^{1}) or the operand cycle associated with the previous instruction is not completed (QB^{1}). When the $EB^{0} \cdot QB^{0}$ condition is satisfied, PK proceeds to $PK^{09\alpha}$. A wait also occurs in PK^{02} if the instruction word is located in the Arithmetic Element ($PKM^{V}FF \cdot VMD^{AE}$) and the Arithmetic Element is still performing a previous instruction. In this case PK waits in PK^{02} until $\overline{AEB} \cdot QB^{0}$ occurs. A similar situation occurs in $QK^{03\alpha}$. QK cannot go on if the operand is located in the Arithmetic Element and the Arithmetic Element is busy with a previous instruction ($QKM^{V}FF \cdot VMD^{AE} \cdot AEB$). Waiting can also occur in $PK^{25\alpha}$ when non-operand type instructions (Class A) are executed. The actual waiting state logic in this case depends on the instruction executed. (See Chapter 17 for a discussion of the terms used in this logic in each instruction.)

9-6 COUNTER DYNAMICS WHEN A TRANSITION TO OR FROM A CHANGE OF SEQUENCE (CSK) OR DELAY SYNCHRONIZATION CYCLE (DSK) IS INVOLVED.

This section will discuss in detail Cases 2, 3 and 4 shown on Fig. 9-6, i.e., the cases where transitions to or from delay synchronization or change of sequence cycles are involved. The interlocks that determine these transitions are PI_3 and CSK_4. Fig. 9-10 shows the transition possibilities and the states of PI_3 and CSK_4 required for the transitions to occur.

By examining the conditions which set and clear PI_3 and CSK_4, it will be possible to establish the conditions which cause the transitions.

9-6.1 DECISION AND WAITING LOGIC.

Certain specific states in the PK cycle are called "decision states". The following alternative types of decisions are made in these states:

1) To immediately go on in the current sequence (or, more specifically, in some cases to go on in the current instruction), subject only to the interlock conditions just described in Section 9-5.

2) To immediately abandon the current instruction and perform a change of sequence.

3) To wait until the conditions for making a decision to go on with instructions in the current sequence are available.

4) To wait until the conditions for making a decision to change sequence are available.

If the decision to wait is made, PK will go into a "waiting state" associated with the PK "decision state". In this case, the decision to go on in the current sequence or to make a change of sequence will be made during a delay synchronization cycle(s), i.e., the decision will now be made by having the DSK counter sample the conditions on which the decision is based.

Fig. 9-11 summarizes this PK and DSK decision logic. If the instruction word is located in the V_{FF} memory, PK^{02} becomes a decision state. Note that at PK^{02}, the instruction word has not yet been placed in the N register. Thus the basic question on which a decision must be made is whether in fact the instruction word can be read out of memory and placed in N. This will occur if the logic for going on to PK^{09} is satisfied, i.e., if the instruction word is in a register that is currently accessible. If this decision cannot be made immediately, CSK_4 is set to ONE and a delay synchronization cycle(s) occurs. PK waits in PK^{02} until DSK clears CSK_4 to ZERO. If at the same time, PI_3 is set to ONE and PK is set back to PK^{00}, a change of sequence

cycle will occur. The change of sequence cycle will always clear PI_3 so that it can be followed by a PK cycle. If PI_3 is not set to ONE, PK will wait until the other conditions for going on to PK^{09} are satisfied (see Sect. 9-5).

Note in this case that the PK decision state and waiting state are the same, i.e., PK^{02}. Also note that the current sequence cannot be abandoned until after at least one delay synchronization cycle occurs.

PK^{22} is another decision state. Note that in this case the instruction word has already been placed in the N register. Thus the basic decision is whether to continue on in the current instruction or to abandon the instruction and perform a change of sequence. If the "wait" conditions are not generated, PK will immediately go on to PK^{24}. If the "leave sequence" conditions are generated, PI_3 will be set to ZERO and PK set back to PK^{00}, i.e., the current instruction will be abandoned and a change of sequence will occur. If the "wait" condition occurs, but the "leave sequence" condition is not generated, CSK_4 will be set to ONE and PK will wait in PK^{23} while the delay synchronization cycle(s) sample the "wait" and "leave sequence" conditions. If at some time the "not wait" conditions occur, PK will go on to PK^{24}. On the other hand, if the leave sequence conditions are generated, PI_3 will be set and PK set back to PK^{00}, i.e., the current instruction will be abandoned and a change of sequence will occur. Note that the flag of the current sequence can be dismissed (i.e., lowered) during a TSD in PK^{22}. This will occur if the IO buffer is busy or the QK cycle of a previous TSD is going on. This decision is made independently of the status of the hold bit on the TSD.

In PK^{24} all the $\overline{PKIR^{DIS}}$ instructions (i.e., all the Class C instructions) will cause PK to go ahead to PK^{00}. If the change sequence conditions are satisfied, PI_3 will be set to ONE in PK^{24}. If the instruction has a PKEI cycle, i.e., is a $PKIR^{DIS}$ instruction which terminates in PK^{31}, PI_3 will similarly be set in PK^{24} (so long as it is not an IOS instruction). In this case, the instruction will be completed before the change of sequence called for by the PI_3^1 condition occurs. The decisions made in PK^{25} and PK^{31} occur only in $PKIR^{DIS}$ type instructions. These instructions are of two basic types: those that "dismiss" ($PKIR^{DIS\ REQ}$) and those that "do not dismiss" ($\overline{PKIR^{DIS\ REQ}}$). Consider first the $PKIR^{DIS\ REQ}$ class. If the conditions for changing sequence are satisfied in PK^{31}, PI_3 will be set to ONE and the current PK cycle will be followed by a change of sequence cycle when PK reaches PK^{00}. If the conditions for changing sequence are not satisfied, the current instruction will simply be followed by the next instruction in the current sequence.

Consider now the instructions that can dismiss ($PKIR^{DIS\ REQ}$). While the JX type instructions are in this class, the logic requires that they be treated separately and they will, for the moment, be ignored. If the conditions for dismissing are satisfied in PK^{25}, the flag of the current sequence will be lowered. Note that in the case of TSD, which falls in this class, the flag may be dismissed twice during

the instruction, once in PK^{22} and again in PK^{25}. In PK^{31} a decision will be made to set CSK_4 to a ONE if the dismiss conditions are satisfied and no sequence requests attention. This means that the current PK cycle will be followed by a delay synchronization cycle(s). If the conditions for changing sequence are satisfied in PK^{31}, PI_3 will be set to ONE and the current PK cycle will be followed by a change of sequence cycle.

In the case of the JX type instructions, CSK_4 is set to ONE (if it is to be set) in PK^{25} instead of PK^{31} and the change sequence conditions are sampled only in PK^{24}. Note that, except for the JX and IOS instructions, the change sequence conditions are sampled at both PK^{24} and PK^{31} during those instructions that terminate in PK^{31}.

PK^{00} is the waiting state associated with the PK^{24}, PK^{25} and PK^{31} decision states.

PI_3 is always cleared during a change of sequence cycle at $CSK^{04\alpha}$, i.e., the CSK cycle is usually followed by a PK cycle. Only when a "trap" occurs on a sequence meta bit can two CSK cycles occur in succession. See Chapters 10 and 15. All the logic for setting and clearing PI_3 and CSK_4 has now been discussed.

The logical definitions of the factors and terms used on Fig. 9-11 are described in detail in Chapter 10.

The starting conditions for the CSK and DSK counters will now be examined.

9-6.2 CSK AND DSK COUNTER STARTING CONDITIONS. The interlock start conditions for the DSK and CSK counter are shown on Fig. 9-12. Note that CSK and DSK are physically the same counter. Which interpretation is given depends on the state of CSK_4. CSK_4^0 implies a change of sequence cycle, while CSK_4^1 implies a delay synchronization cycle. The interlocks that are set and cleared by the DSK and CSK counter are shown on Fig. 9-13.

9-6.2.1 CSI^{START}. CSK begins counting when a change of sequence is called for. CSK cannot start counting until PK is in its PK^{00} resting state and CSK_4 is cleared to ZERO. It is assumed that the $START_2^1$ pushbutton condition is satisfied.

XW^0 - This interlock is set and cleared in the XWK counter cycle. Since the change of sequence cycle uses the X Memory, the CSK counter cannot start until XW^0.

XB^0 - (See discussion earlier under $PI^{START}1$). CSK cannot start while the X Memory is in use. XB^1 covers such periods until XW^1.

EB^0 - This interlock is set and cleared in the QK cycle, except in the SPG instruction when it is cleared during the FK cycle. Since the change of sequence cycle uses the E register for temporary storage, there must be assurance that the E register is free (EB^0) before the CSK cycle can start.

$PI_3^1 \cdot CSK_4^0$ - This interlock condition was discussed earlier in this section.

9-6.2.2 DSK STARTING CONDITIONS. DSK begins counting when a delay synchronization cycle is called for. DSK cannot start counting unless CSK_4 is set to ONE. The conditions for CSK_4^1 were discussed earlier in this section. DSK will count only if the XWK counter is in its 00 resting state and PK is in one of its waiting states, i.e., PK^{02}, PK^{23} or PK^{00}.

9-7 DEFERRED ADDRESSING CYCLES

When the computer is ready for a new instruction, a PK cycle is used to read the instruction out of memory. If this instruction calls for a deferred address word, PK goes through another cycle, during which it reads out the deferred-address word from memory. If this deferred-address word calls for still another deferred-address word, the cycle is repeated. Finally, a deferred-address word is obtained which does not call for another deferred-address word. PK now performs the so-called ultimate deferred-address cycle, during which the final base address is computed and the index register specified by the instruction word is placed in X.

This section will examine the PK counter activity pattern during the deferred addressing process. The interlocks of primary interest are PI_2 and PI_5. The times at which these interlocks are set and cleared will determine the sequence of cycles.

Fig. 9-14 shows the basic deferred-address cycle and the times at which PI_2 and PI_5 are set and cleared.

The latest time at which the instruction is strobed into N during a PK cycle is $PK^{11\beta}$. The defer bit ($N_{2.9}$) is then examined at $PK^{13\alpha}$. If a deferred address is called for ($N_{2.9}^1$), PI_2 is set to ONE. Assuming the instruction is defined ($PKIR^{DEF}$), PI_5 will in turn be set to ONE in $PK^{14\alpha}$. The conditions ($PI_2^1 \cdot PI_5^1$) for an intermediate-deferred-address cycle to follow the current PK cycle have now been set up.

Once the instruction word memory cycle (PKM) is completed, PK is ready for an intermediate-deferred-address cycle. During the instruction word memory cycle, the instruction word's configuration, hold and OP code bits are placed in the $PKIR_{CF}$ and $PKIR_{OP}$ registers. This information will remain in these registers all during the succeeding intermediate and ultimate deferred cycles.

When the $PI^{START}{}_2$ conditions are satisfied, the first intermediate-deferred address cycle will begin. These cycles will continue until a deferred address word is read out which does not call for another deferred address word, i.e., $N^0_{2.9}$. The latest time at which this occurs is $PK^{11\beta} \cdot N^0_{2.9}$ causes PI_5 to be cleared to ZERO in $PK^{14\alpha}$. PI^0_5 insures that PK will execute next an ultimate deferred-address cycle.

The ultimate deferred-address cycle does not involve a memory, but simply the computation of the final deferred-address. Note that the $PI^1_2 \cdot PI^0_5$ interlock condition determines that no memory cycle is involved. PI_2 is cleared to ZERO in $PK^{13\alpha}$. The balance of the PK cycle is then like any normal instruction word cycle. The instruction is completed using the address computed in the ultimate cycle and the operation called for by the original instruction word.

9-8 IN-OUT TIME CONSIDERATIONS

Earlier in the chapter the effect of the In-Out Element on the interlocking decisions was implicitly examined. For example, the effect of levels like PI^{WAIT}, $PI^{CH\ SEQ}$, $PI^{LV\ SEQ}$, etc. on interlock decisions was analyzed. These levels are based on information from the Sequence Selector. This information, in turn, reflects events that have occurred in the In-Out Element. However, these events in the In-Out Element are generally initiated by the central computer. The time between the event in the central computer and the interlock condition in the central computer that reflects the chain of events in the In-Out Element initiated by this event can be considerably more than 0.4 microsecond.

Three types of central computer pulses can initiate action affecting the In-Out Element. These are: (1) IOI clock pulses, (2) IOS mode and select pulses, and (3) TSD data transfer pulses. A minimum of 1.6 microseconds must elapse before the interlock levels affected by these pulses can be sampled. The only other events occurring in the In-Out Element that can affect the central computer are events such as MISIND alarms, EIA alarm levels generated by switches, etc.

IOI clock pulses can be generated only at PK^{01}, PK^{12} and CSK^{11}. The decision and waiting states that occur at least 1.6 microseconds after these IOI clock pulses can be used to sample the interlock conditions affected by these pulses. Note that for this reason a decision to change sequence cannot be made in PK^{02} until after at least one delay synchronization cycle occurs, i.e., until at least 1.6 microseconds has elapsed since PK^{01}. Since PK^{22} occurs at least 1.6 microseconds after PK^{12}, a decision to change sequence can be made in PK^{22}.

The IOS mode and select pulses are generated at PK^{26}. These pulses can raise and lower flags in the Sequence Selector directly or change the mode of the In-Out Element, so that it in turn changes the status of flags in the Sequence Selector. The $PI^{CH\ SEQ}$ interlock level affected by these events cannot be sampled until at least 1.6 microseconds after PK^{26}. The interlock condition is in fact sampled at PK^{31} (see Fig. 9-11). PI_3 is not sampled at PK^{24}

by an IOS (see Fig. 9-11). This sampling is inhibited until after the IOS has a chance to change the mode of the In-Out Element, i.e., until after $PK^{26\alpha}$.

The buffer busy level ($IOCM^{BB}$) is used in the "wait" and "leave sequence" logic as well as in the FLAG dismissing logic in PK^{22}. This level is affected by the TSD data transfer pulses that occur in QK^{20}. For this reason decisions based on $IOCM^{BB}$ cannot be made until 1.6 microseconds after QK^{20}.

9-9 PROGRAM EXAMPLE

A specific example of the counter activity that occurs during a short program will be given. This example is designed to illustrate the effect of the interlock control on computer dynamics.

The assumption is made that the instruction word is stored in the S Memory and that the operand is stored in the T Memory. The program will consist of the following instructions:

```
.
.
DSK
CSK
TSD (H^1 = hold)
JMP (CF^1_5 = dismiss requested)
    (CF^1_2 = XWK at PK^31)
DSK
.
.
```

Fig. 9-15 shows the counter activity pattern for this program.

Assume that the initial DSK cycle starts while PK is in the PK^{00} resting state (a result of the previous instruction dismissing itself). Assume also that during this DSK cycle, CSK^{11} samples a $SS^{ATT\ REQ} \cdot (K^{eq\ JC} + KD^{00})$ condition. This condition at CSK^{11} causes PI_3 to be set to ONE and CSK_4 to be cleared to ZERO (see Fig. 9-11). This insures that a change of sequence will follow (see Fig. 9-10(b)). Note that all the CSI^{START} conditions are now satisfied. (It is assumed that XW, XB, EB and PI_1 were cleared to ZERO previously.)

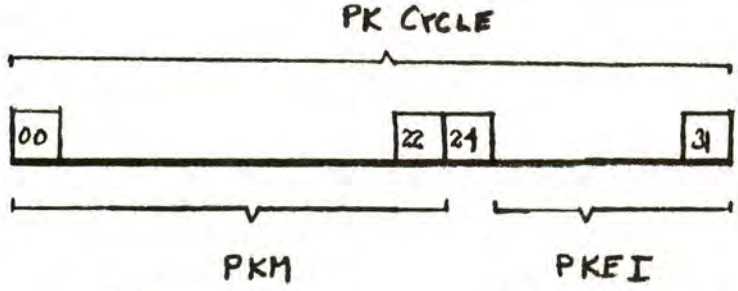
The CSK cycle clears PI_3, sets XB and starts XWK (see Fig. 9-13). XWK in turn clears XB in XWK^{02} (see Fig. 9-8). XB^0 is the crucial interlock in the $PI^{START}1$ level. PK starts counting as soon as XB is cleared.

During the TSD, the PK cycle sets XB and starts XWK counting. PK also sets PI_1 to ONE in PK^{22}. All the QI^{START} conditions are now satisfied (see Fig. 9-4) and QK begins counting. Note that the conditions that start QK are sufficient, in this case, to start FK counting.

QK immediately clears PI_1 to ZERO, so that the $PI^{START}{}_1$ condition is again satisfied. However, PK must finish the current TSD instruction before beginning the JMP instruction.

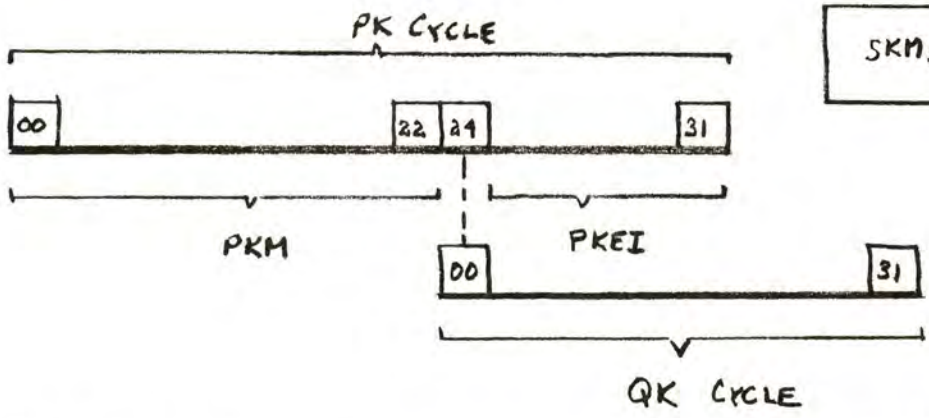
If the TSD instruction had dismissed instead of holding, PK may have had to wait in PK^{00} while DSK examined the conditions for going ahead in the program. The fact that the hold bit was a ONE, meant that the $PI^{START}{}_1$ conditions are immediately generated and that PK will go on to the next instruction in the current sequence.

The JMP instruction has no operand cycle. In the case chosen ($CF^1_2 \cdot CF^1_5$), the instruction dismisses. Fig. 9-2 and Fig. 9-8 show that an XWK cycle starts at PK^{31} in this instruction. PK^{31} in the JMP instruction sets CSK_4 to ONE because the $PKIR^0_H \cdot \overline{SS^{ATT\ REQ}} \cdot PKIR^{DIS\ REQ}$ condition is satisfied at that time (see Fig. 9-11). 1 CSK_4 places the CSK counter in the CSK^{08} state, i.e., the DSK resting state. DSK must wait in this state until XWK completes its cycle and returns to its XWK^{00} resting state. At that time delay synchronization cycles start being executed, and continue until some sequence again requests attention.



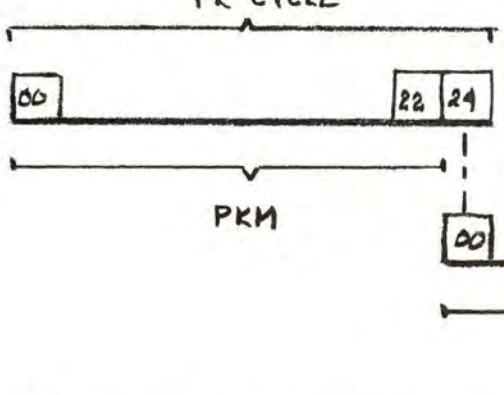
OPR	JNX	JPA
JMP	SKX	JNA
JPX	JOV	

(a) CLASS A INSTRUCTIONS - NO OPERAND (QK) CYCLE AND PK TERMINATES IN PK^{31}



SKM, SED, TSD

(b) CLASS B INSTRUCTIONS - OPERAND (QK) CYCLE AND PK TERMINATES IN PK^{31}

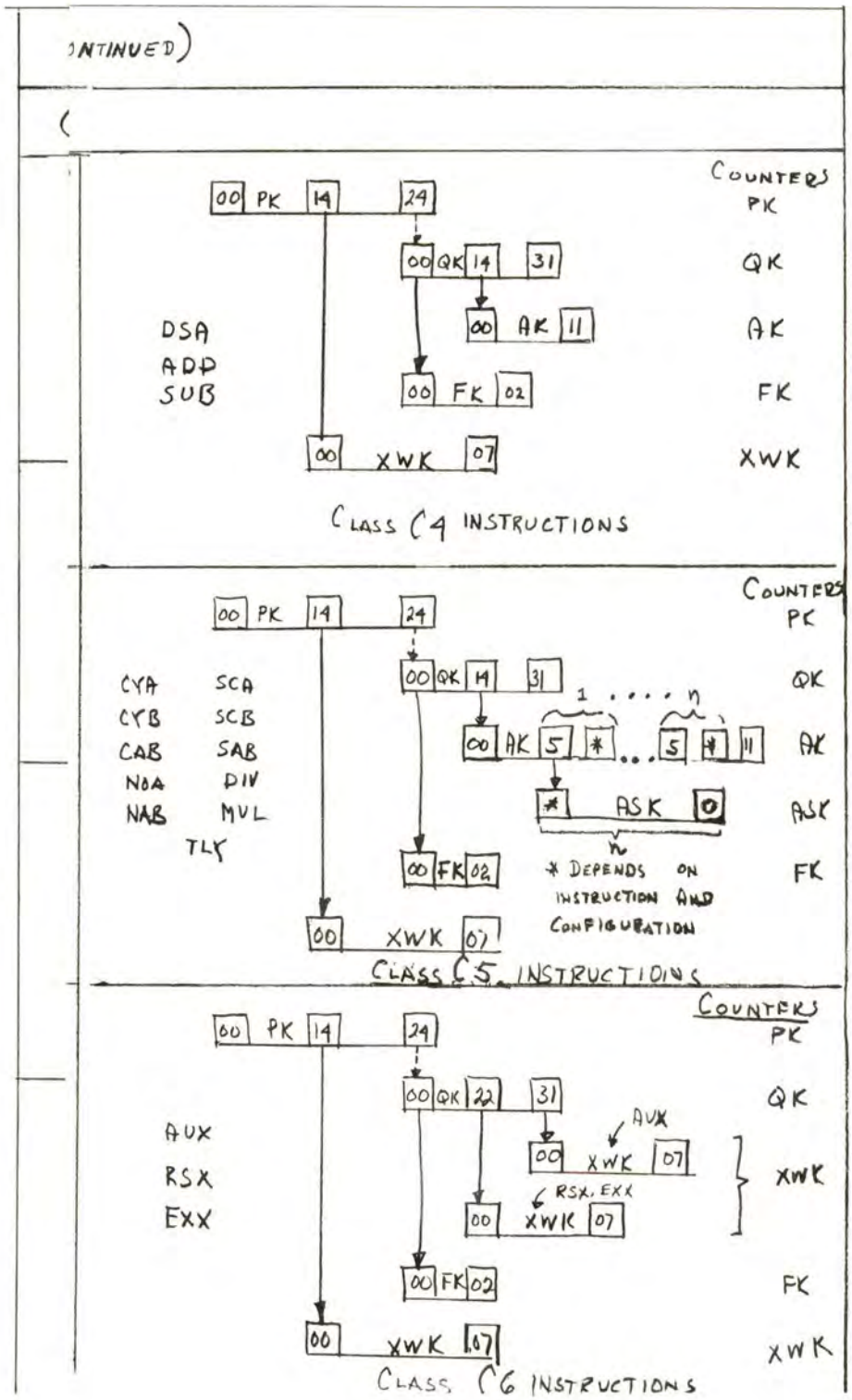


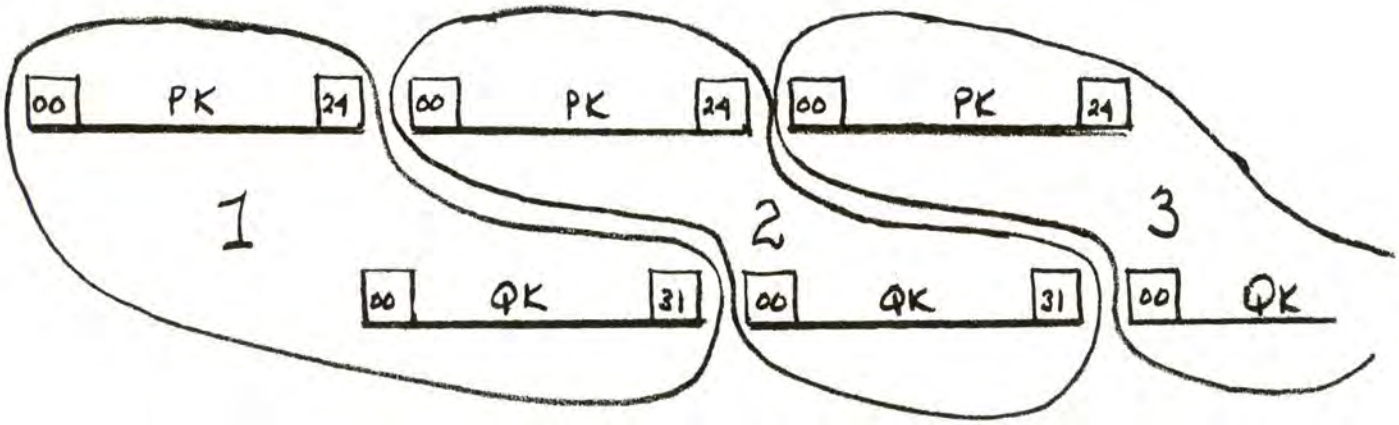
AUX	SPF	ITA	CYA	DSA	TLY	LDA/B/C/D/E
RSX	SPG	ITE	CAB	ADD	SCA	STA/B/C/D/E
EXX	FLF	UNA	CYB	SUB	SAB	
ADX	FLG	EXA	NOA	MUL	SCB	
DPX	COM	INS	NAB	DIV		

(C) CLASS C INSTRUCTIONS - OPERAND (QK) CYCLE AND PK TERMINATES IN PK^{24}

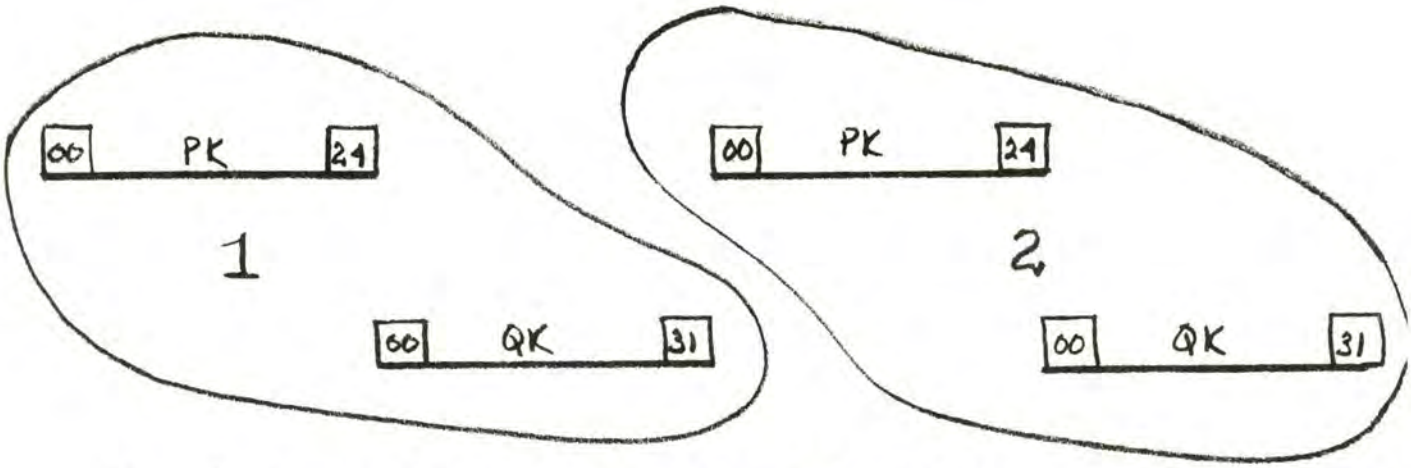
FIG. 9-1 BASIC INSTRUCTION CLASSIFICATION BY COUNTER ACTIVITY

(CONTINUED)


COUNTERS
PK
00 PK 14 24
00 QK 14 31
QK
00 AK 11
AK
DSA
ADD
SUB
00 FK 02
FK
00 XWK 07
XWK
CLASS C4 INSTRUCTIONS
COUNTERS
PK
00 PK 14 24
00 QK 14 31
QK
1 . . . . n
CYA SCA
CYB SCB
CAB SAB
NOA DIV
NAB MVL
TLY
00 AK 5 * . . . 5 * 11
AK
* ASK 0
ASK
n
00 FK 02
* DEPENDS ON INSTRUCTION AND CONFIGURATION
FK
00 XWK 07
CLASS C5 INSTRUCTIONS
COUNTERS
PK
00 PK 14 24
00 QK 22 31
QK
AUX
00 XWK 07
AUX
XWK
RSX
EXX
00 XWK 07
RSX, EXX
00 FK 02
FK
00 XWK 07
XWK
CLASS C6 INSTRUCTIONS



(a) TYPICAL PK, QK COUNTER SEQUENCE WHEN $\overline{NO^{0} \cdot (P^{S} \cdot Q^{S} + P^{T} \cdot Q^{T} + P^{U} \cdot Q^{U} + P^{V} \cdot Q^{V})}$

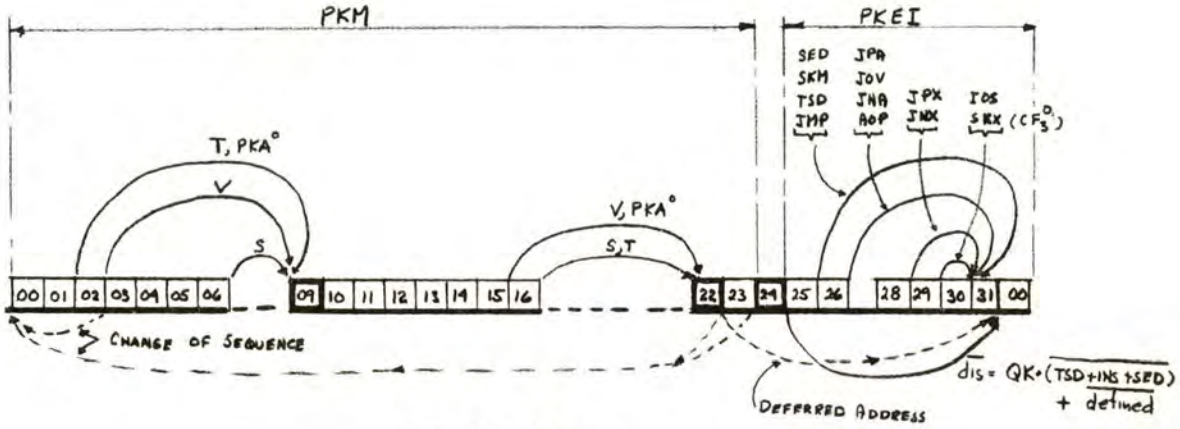


(b) TYPICAL PK, QK COUNTER SEQUENCE WHEN $NO^{1} + (P^{S} \cdot Q^{S} + P^{T} \cdot Q^{T} + P^{U} \cdot Q^{U} + P^{V} Q^{V})$

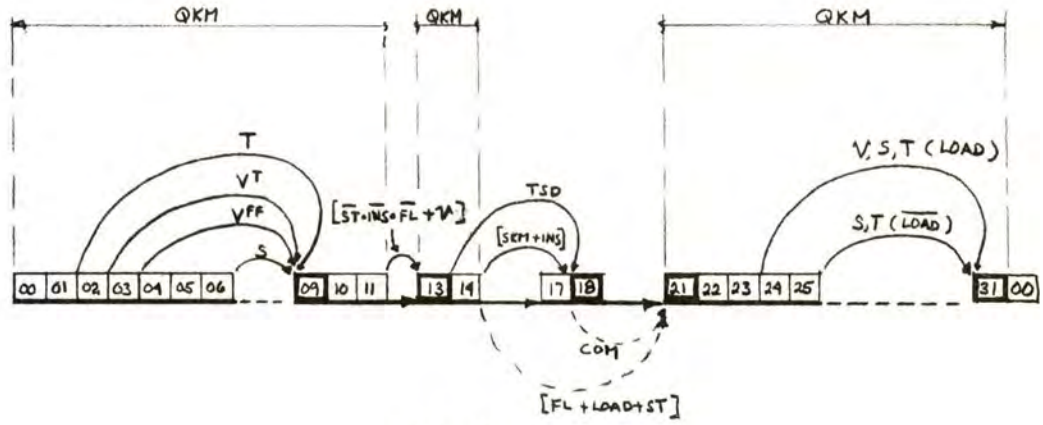
FIG. 9-3 EFFECT OF MEMORY OVERLAP AND NO MEMORY OVERLAP SWITCH ON CLASS C1 INSTRUCTIONS

MEMORIES	PKM	QKM		
	—	ALL LOAD	ALL SIMPLE STORES	TSD (NON SIMPLE STORE)
S	6.4	6.4	7.6	8.8
T	4.4	4.4	5.6	6.8
V^{FF}	4.4	5.2	4.8	6.0
V^{T}	4.4	4.8	5.2	6.4

(a) BASIC MEMORY TIMES IN MICROSECONDS



(b) PK CYCLE



(c) QK CYCLE

FIG. 9-4 INFLUENCE OF MEMORIES AND INSTRUCTIONS ON PK AND QK COUNTING CYCLES

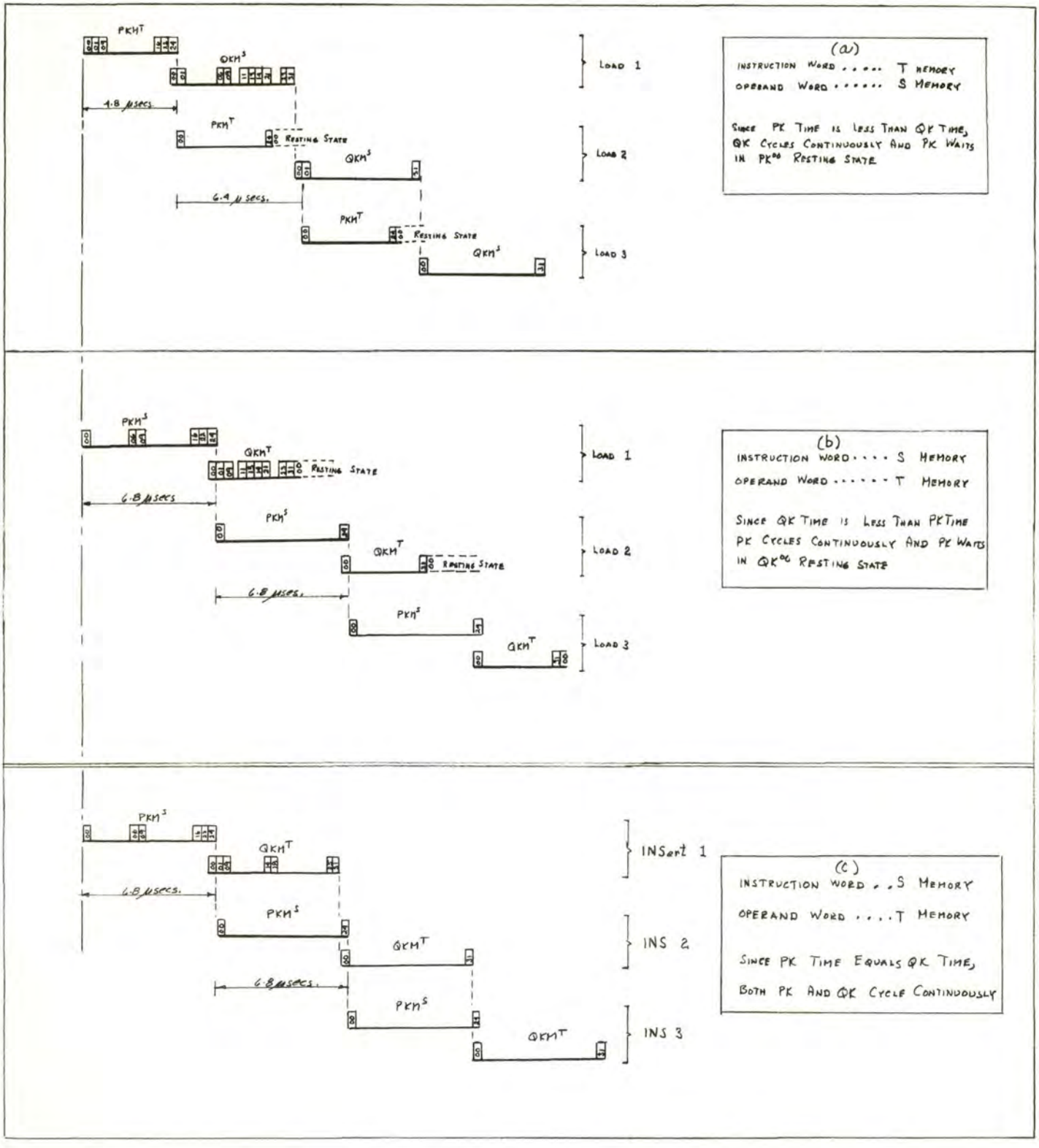


FIG 9-5 SUCCESSION OF IDENTICAL INSTRUCTIONS WITH INSTRUCTION AND OPERAND WORDS STORED IN DIFFERENT MEMORIES.

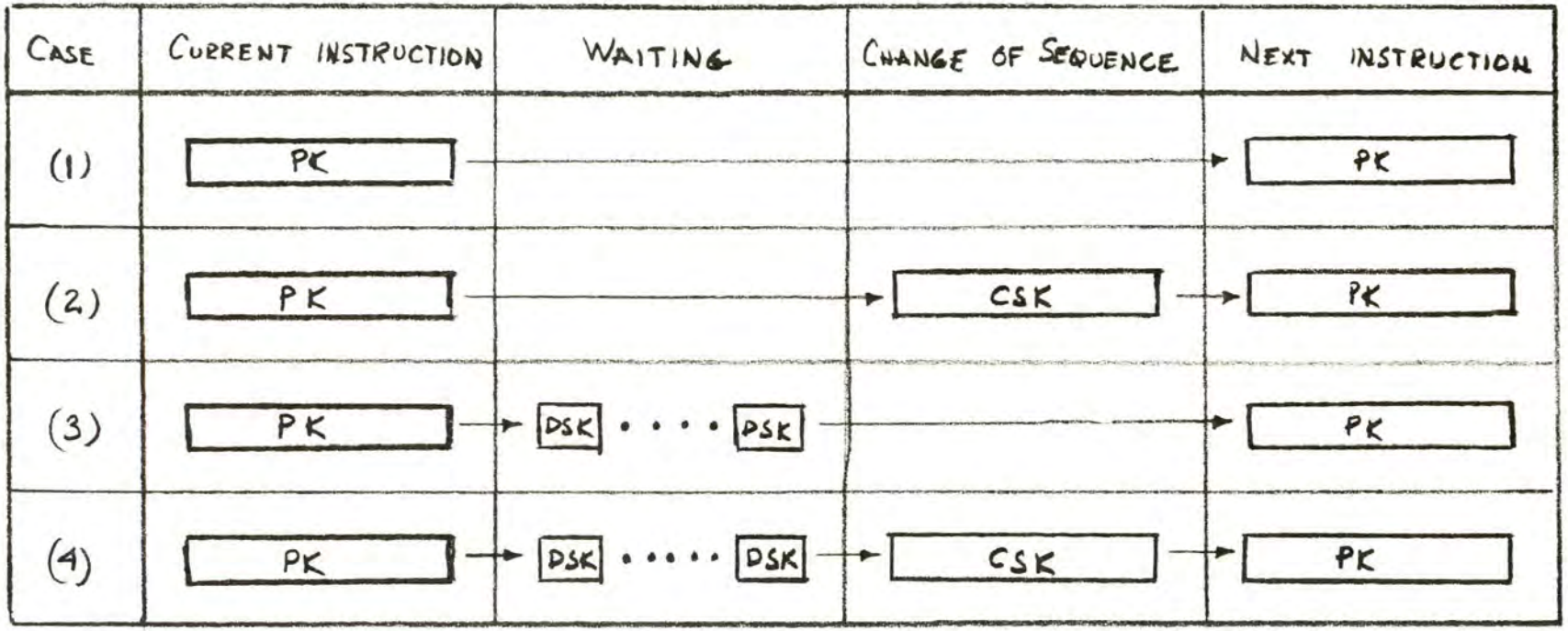


FIG. 9-6 POSSIBILITIES FOR TRANSITION FROM ONE INSTRUCTION TO THE NEXT INSTRUCTION

INTERLOCK START LEVEL LOGIC	COUNTER	COUNT LOGIC
$PI^{START_1} = PI_1^{0} \cdot PI_3^{0} \cdot CSK_4^{0} \cdot XB^{0} \cdot PKS_1^{0} \cdot \overline{AL} \cdot START_2^{1} \cdot [QB^{0} + NO^{0} \cdot (P^{1} \cdot Q^{1} + P^{T} Q^{T} + P^{U} Q^{U} + P^{V} \cdot Q^{V})]$ $PI^{START_2} = QB^{0} \cdot XB^{0} \cdot PKS_2^{0} \cdot START_2^{1}$	PK	$PK^{00d} \cdot [PI_2^{0} \cdot \overline{PI^{START_1}} + PI_2^{1} \cdot \overline{PI^{START_2}}] \supset \overline{PK}\vert+1 \rightarrow PK$
$QI^{START} = PI_1^{1} \cdot QKS^{0} \cdot START_2^{1} \cdot FI$	QK	$QK^{00d} \cdot \overline{QI^{START}} \supset \overline{QK}\vert+1 \rightarrow QK$
$\boxed{START}\ XWK = CSK^{09d} + CSK^{11d} \cdot PK^{22d} \cdot PKIR^{XM} \cdot PI^{LEAVE\ SEQ}$ $+ PK^{19d} \cdot (PI_2^{1} + \overline{PKIR^{XM}}) + PK^{31d} \cdot PKIR^{XM}$ $+ QK^{22d} \cdot QKIR^{LD} \cdot QKIR^{X} + QK^{31d} \cdot QKIR^{AUX}$	XWK	$XWK^{00d} \cdot \overline{\boxed{START}\ XWK} \supset \overline{XWK}\vert+1 \rightarrow XWK$
$\boxed{START}\ FK = QK^{00} \cdot QI^{START} \cdot \overline{PKIR^{f}} \cdot \overline{PKIR^{SEM}} + QK^{13d} \cdot (QKIR^{SPG} + QKIR^{SPF})$ $+ FI^{0} \cdot EB^{0} \cdot (PKIR^{sf} + PKIR^{ia})$	FK	$(FK^{00} \cdot FKB^{0}) \cdot \overline{\boxed{START}\ FK} \supset \overline{FK}\vert+1 \rightarrow FK$
$\boxed{START}\ AK = QK^{14d} \cdot QKIR^{AK} + PK^{26d} \cdot PKIR^{OPR\ AE}$	AK	$AK^{00d} \cdot \boxed{START}\ AK \supset \overline{AK}\vert+1 \rightarrow AK$

FIG. 9-7 STARTING LOGIC FOR PK, QK, XWK, FK AND AK COUNTERS

CLASS	INSTRUCTION	PK CYCLE: PK TERM STATE	PK: XB	PK: PI1	PK: FI	PK: START XWK	PK: START AK	XWK CYCLE: XB	XWK: XW	XWK: XW	INSTRUCTION	QK CYCLE: QB	QK: QB	QK: EB	QK: EB	QK: PI1	QK: XB	QK: START FK	QK: START XWK	QK: START AK	FK CYCLE: EB	FK: FI	FK: FK TERM STATE	INSTRUCTION	CLASS	SUB-CLASS
A	SKX	PK^{31}	PK^{12}			PK^{31} ③		XWK^{03}	XWK^{02}	XWK^{16}	SKX													SKX	A	1
	JPX, JNX										JPX, JNX													JPX, JNX		
	JMP (CF_2^1)										JMP (CF_2^1)													JMP (CF_2^1)		
	JMP (CF_2^0)										JMP (CF_2^0)													JMP (CF_2^0)		
	JPA, JOV, JNA				PK^{13}	PK^{19}					JPA, JOV, JNA											FK^{02}	FK^{02}	JPA, JOV, JNA		4
	IOS										IOS													IOS		2
	AOP						PK^{26}				AOP													AOP		3
B	SKM			PK^{22} ②							SKM	QK^{00}	QK^{31}	QK^{00}	*	QK^{19}		QK^{00}						SKM	B	1
	SED										SED				QK^{23}	QK^{00}							FK^{02}	SED		3
	TSD										TSD				*									TSD		2
C	AUX	PK^{29}									AUX				QK^{23}	QK^{19}	QK^{13}		QK^{31}					AUX	C	6
	RSX										RSX								QK^{32}					RSX		
	EXX										EXX				*	QK^{16}								EXX		1
	ADX										ADX					QK^{09} ①								ADX		
	DPX										DPX				QK^{33}	QK^{21}	QK^{13}							DPX		2
	SPF										SPF										FK^{06}		FK^{08}	SPF		
	SPG										SPG											FK^{07}		SPG		3
	FLG				PK^{13}						FLG				*	QK^{00}						FK^{02}	FK^{02}	FLG		
	FLF				↓						FLF													FLF		1
	STA, STB, STC, STD, STE										STA, STB, STC, STD, STE						QK^{00}							STA, STB, STC, STD, STE		
	EXA										EXA													EXA		
	INS										INS													INS		
	COM										COM													COM		
	ITA, UNA										ITA, UNA				QK^{23}									ITA, UNA		
	LDA, LDB, LDC, LDD, LDE										LDA, LDB, LDC, LDD, LDE													LDA, LDB, LDC, LDD, LDE		
	ITE										ITE													ITE		
	CYA, CYB, CAB										CYA, CYB, CAB									QK^{14}				CYA, CYB, CAB		5
	NOA, NAB										NOA, NAB													NOA, NAB		
	SCA, SCB, SAB										SCA, SCB, SAB													SCA, SCB, SAB		
	DIV, MUL										DIV, MUL													DIV, MUL		
	DSA										DSA													DSA		4
	ADD, SUB										ADD, SUB													ADD, SUB		
	TLY										TLY													TLY		5

① PI1 ALSO CLEARED REDUNDANTLY IN QK^{19}

② PI1 CAN ALSO BE SET IN PK^{23} WAITING STATE DURING INSTRUCTION

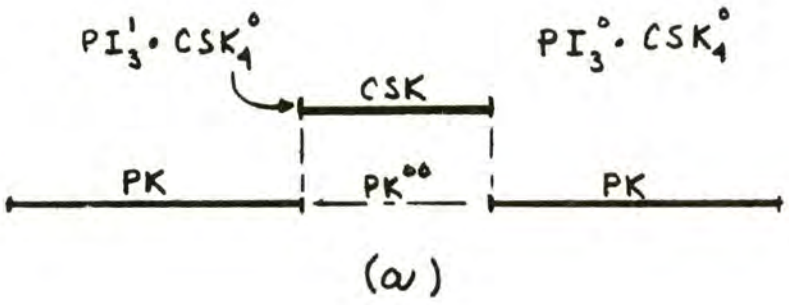
③ XWK CAN ALSO BE STARTED IN PK^{23} WAITING STATE DURING INSTRUCTION

* - QK^{21} UNLESS QKM^{VFF}, QK^{23} IF QKM^{VFF}

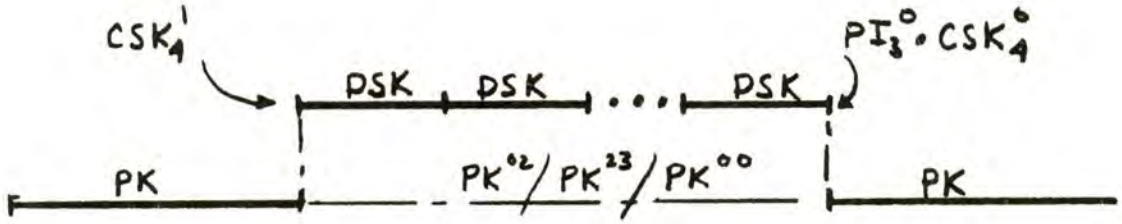
FIG. 9-8 PK, XWK, QK AND FK INTERLOCK EVENTS

WAITING STATE	OPERATION CODE	WAITING STATE LOGIC
PK^{02}	ALL OP CODES	$PKM^{VFF} \cdot [(\overline{AEB} + \overline{VMD^{AE}}) \cdot CSK_4^{\circ} \cdot EB^{\circ} \cdot QB^{\circ}] \supset$ 09 → PK
QK^{03}	ALL OP CODES	$QKM^{VFF} \cdot [\overline{AEB} + \overline{VMD^{AE}}] \supset$ 09 → QK
PK^{25}	JPX, JNX	$EB' \cdot XJ \supset \overline{PK} + 1 \not\rightarrow PK$
	JMP	$EB^{\circ} + PKIR_{CF_3}^{\circ} \cdot PKIR_{CF_4}^{\circ} \supset$ 31 → PK
	JOV, JPA, JNA	$AEB' + QB' + FI^{\circ} \supset \overline{PK} + 1 \not\rightarrow PK$
	AOP	$AEB' + QB' \supset \overline{PK} + 1 \not\rightarrow PK$
	JOS	$EB' + QB' \supset \overline{PK} + 1 \not\rightarrow PK$
	SKM, SED	$QK^{14d} \supset$ 31 → PK
	TSD	$PI_4^{\circ} \cdot QK^{20d} \supset$ 31 → PK $PI_4' \cdot QK^{01d} \supset$ 31 → PK

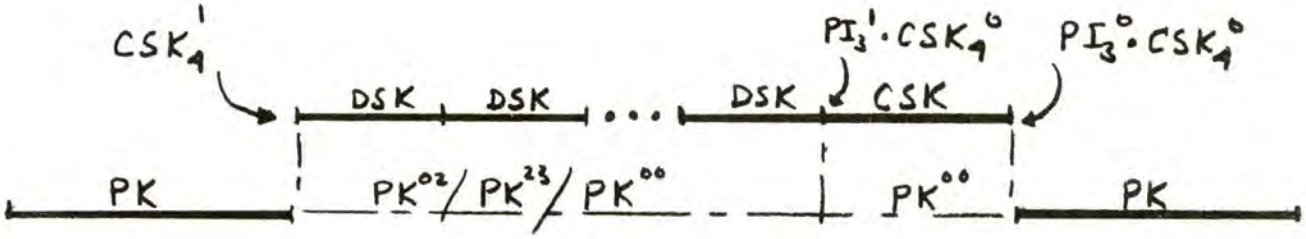
Fig. 9-9 Simple Waiting State Logic



(a)



(b)



(c)

FIG. 9-10 PI_3 AND CSK_4 INTERLOCK STATES FOR TRANSITION POSSIBILITIES FROM ONE INSTRUCTION TO THE NEXT. (SEE FIG. 9-6, CASES 2, 3 AND 4, RESPECTIVELY)

PK Decision State	PK Waiting State	PK Decision Logic	DSK Decision Logic
PK^{02}	→	$PKM^{VFF} \cdot [CSK_4^{\circ} \cdot EB^{\circ} \cdot QB^{\circ} \cdot (\overline{AEB} + \overline{VMD^{AE}})$ ⊃ 09→ PK $PKM^{VFF} \cdot [CSK_4^{\circ} \cdot AEI \cdot VMD^{AE}]$ ⊃ ↓↓→ CSK_4	
	PK^{02}	→	$CSK^{11d} \cdot (\overline{AEI} + PI^{AE\ Ch.Seg.})$ ⊃ 10→ CSK_4 $CSK^{11d} \cdot (PI^{AE\ Ch.\ Seg.})$ ⊃ 00→ PK, ↓↓→ PI_3
PK^{22}	→	$PI_2^{\circ} \cdot \overline{PI^{WAIT}}$ ⊃ 29→ PK $PI_2^{\circ} \cdot PI^{Leave\ Seg.}$ ⊃ ↓↓→ PI_3, 00→ PK $PI_2^{\circ} \cdot (\overline{PI^{Leave\ Seg.}} \cdot PI^{WAIT})$ ⊃ ↓↓→ CSK_4 $PI_2^{\circ} \cdot (IOCM^{BB} + QB' \cdot QKIR^{TSD})$ ⊃ Dismiss KD FLAG	
	PK^{23d}	→	$CSK^{11d} \cdot \overline{PI^{WAIT}}$ ⊃ 29→ PK $CSK^{11d} \cdot PI^{Leave\ Seg}$ ⊃ ↓↓→ PI_3, 00→ PK $CSK^{11d} \cdot (\overline{PI^{Leave\ Seg.}} + \overline{PI^{WAIT}})$ ⊃ 10→ CSK_4
PK^{29}	→	$\overline{PKIR^{DIS}}$ $\overline{PKIR^{IOS}} \cdot PI^{Ch\ Seg.}$ ⊃ 00→ PK ⊃ ↓↓→ PI_3	

Instruction Type →		$\overline{\text{Dismiss Request}}$	Dismiss Request ($\overline{jx}$)	jx
PK^{25}	→		$PKIR_h^{\circ} \cdot PKIR^{Dis\ reg.} \cdot \overline{KD^{00}}$ ⊃ Dismiss FLAG	$PKIR_h^{\circ} \cdot PKIR^{Dis\ reg.} \cdot \overline{KD^{00}}$ ⊃ Dismiss KD FLAG $PKIR_h^{\circ} \cdot PKIR^{DIS\ reg.} \cdot SS^{\overline{AH\ reg.}}$ ⊃ ↓↓→ CSK_4
PK^{31}	→	$PI^{Ch\ Seg}$ ⊃ ↓↓→ PI_3	$PKIR_h^{\circ} \cdot PKIR^{Dis\ Reg.} \cdot SS^{\overline{AH\ reg.}}$ ⊃ ↓↓→ CSK_4 $PI^{Ch\ Seg}$ ⊃ ↓↓→ PI_3	
	PK^{00}	→	$CSK^{11d} \cdot SS^{AH\ reg.}$ ⊃ 10→ CSK_4 $CSK^{11d} \cdot SS^{AH\ reg.} \cdot (\overline{K^{reg\ IC}} + KD^{00})$ ⊃ ↓↓→ PI_3	

Fig. 9-11 PK and DSK Decision Logic

COUNTER	CSK_4 STATE	COUNTER INTERLOCK START LEVEL LOGIC	COUNTER STARTING LOGIC
CSK	CSK_4^0	$CSI^{START} = PK^{00} \cdot PI_1^0 \cdot CSK_4^0 \cdot PI_3^1 \cdot XW^0 \cdot XB^0 \cdot EB^0 \cdot START_2^1$	$CSK^{00d} \cdot \overline{CSI^{START}} \supset \overline{CSK}\,]+1 \rightarrow CSK$
DSK	CSK_4^1		$CSK_4^1 \cdot [\overline{XWK^{00d}} + \overline{PK^{00d}} \cdot \overline{PK^{02d}} \cdot \overline{PK^{23d}}] \supset \overline{DSK}\,]+1 \rightarrow DSK$

FIG. 9-12 CSK AND DSK STARTING CONDITIONS

	CSK (CSK_4^0)			DSK (CSK_4')					
	CHANGE OF SEQUENCE CYCLE			DELAY SYNCHRONIZATION CYCLE					
INTERLOCK	START → XWK	11 → XB	10 → PI_3	START → XWK	11 → PI_3	10 → CSK_4	10 → PI_2	11 → PI_1	10 → PI_5
TIME LEVEL	CSK^{04}	CSK^{01} CSK^{04}	CSK^{04}	CSK^{11}	CSK^{11}	CSK^{11}	CSK^{11}	CSK^{11}	CSK^{11}

FIG 9-13 CSK AND DSK INTERLOCK EVENTS

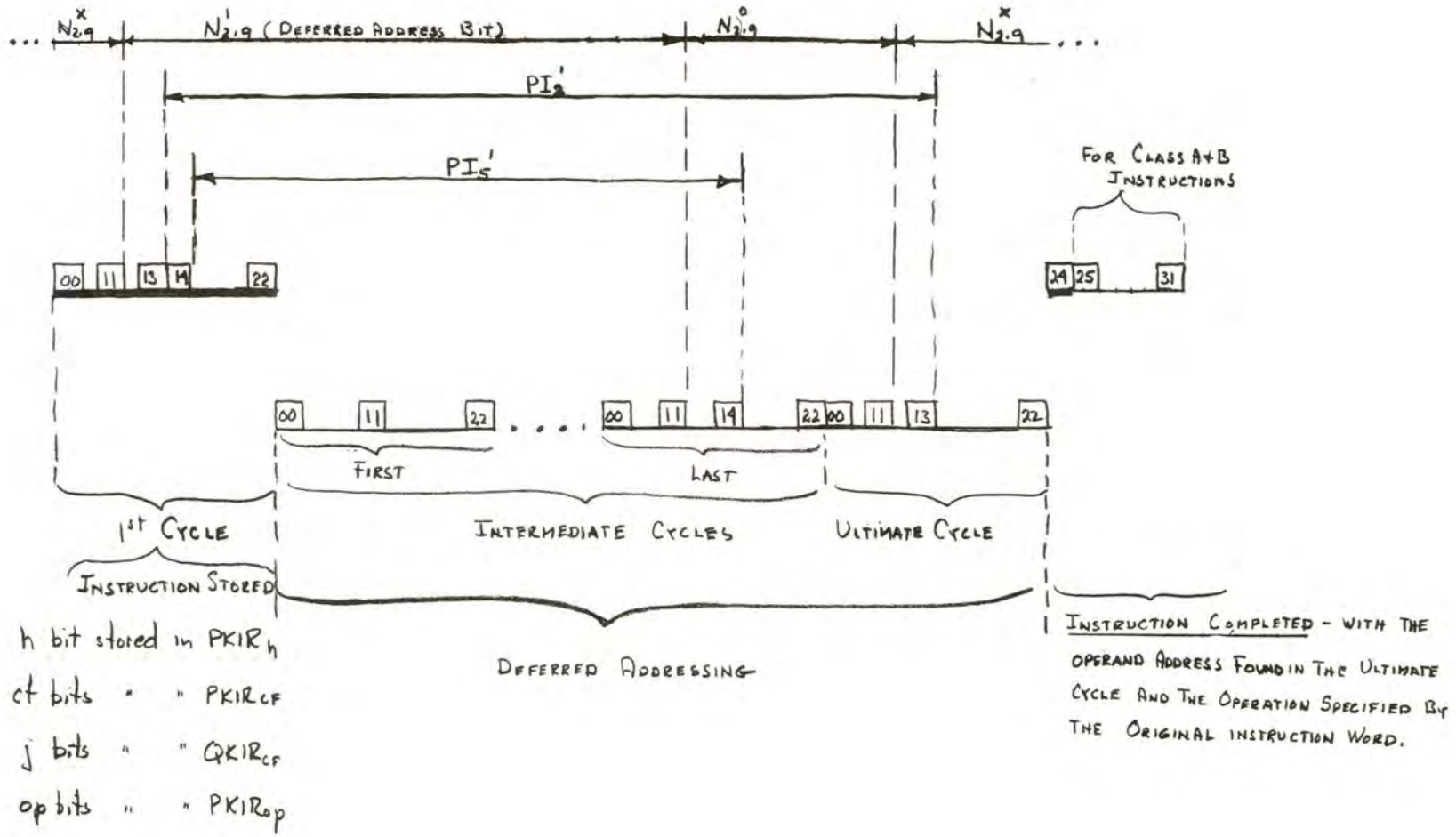


Fig 9-14 Deferred Address Cycles

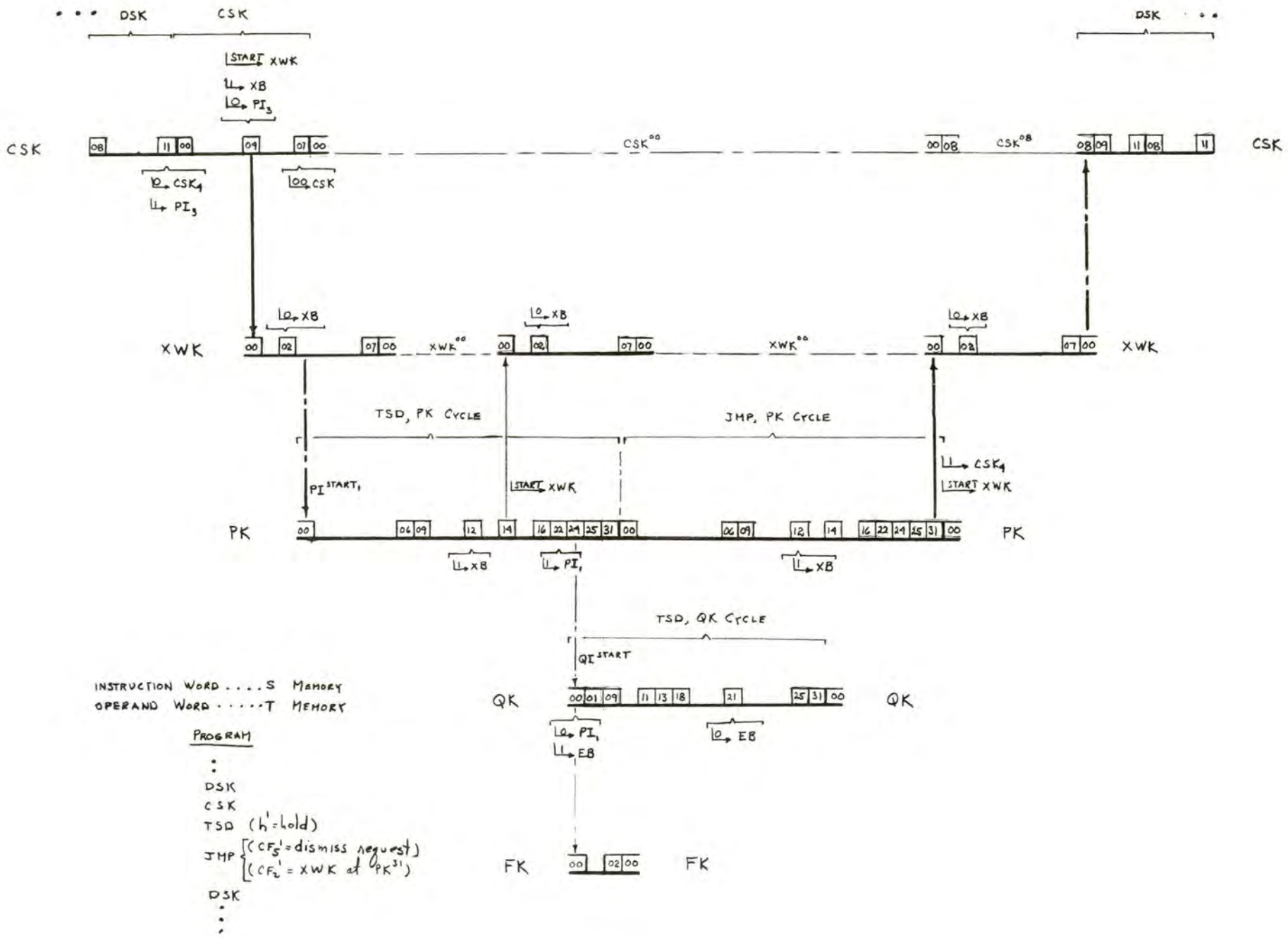


FIG 9-15 COUNTER ACTIVITY FOR SIMPLE PROGRAM EXAMPLE

CHAPTER 10

CONTROL ELEMENT

TABLE OF CONTENTS

10-1 INTRODUCTION

10-2 START-STOP CONTROL

10-2.1 GENERAL DESCRIPTION

10-2.2 START CONTROL

10-2.2.1 AL · $\overline{\text{AUTO START}}$ CONDITION

10-2.2.2 AL · AUTO START CONDITION

10-2.2.3 PB STOP PUSH BUTTON CONDITION

10-2.3 STOP CONTROL

10-2.3.1 LOW SPEED PUSH BUTTON MODE STOP CONTROL

10-2.3.2 LOW SPEED REPEAT MODE STOP CONTROL

10-2.3.3 HIGH SPEED MODE STOP CONTROL

10-2.4 SYNC SYSTEM

10-2.4.1 OUTPUT CONTROL SWITCHES

10-2.4.2 INPUT SELECTION SWITCHES

10-2.5 ALARMS AND ALARM INDICATORS

10-2.5.1 MEMORY SELECTION ALARMS

10-2.5.2 IN-OUT ALARMS

10-2.5.3 OPERATION CODE ALARMS

10-2.5.4 MEMORY PARITY ALARMS

10-2.5.5 MISCELLANEOUS ALARMS

10-2.5.6 ALARM LEVEL (AL)

10-2.5.7 CHIME CONTROL

10-2.5.8 ALARM DELAY COUNTER (ADK)

10-3 INTERLOCKS

10-3.1 GENERAL DESCRIPTION

10-3.2 INSTRUCTION INTERLOCKS

10-3.2.1 INSTRUCTION INTERLOCK$_1$ (PI$_1$)

10-3.2.2 INSTRUCTION INTERLOCK$_2$ (PI$_2$)

10-3.2.3 INSTRUCTION INTERLOCK$_3$ (PI$_3$)

10-3.2.4 INSTRUCTION INTERLOCK$_4$ (PI$_4$)

10-3.2.5 INSTRUCTION INTERLOCK$_5$ (PI$_5$)

10-3.3 ARITHMETIC ELEMENT INTERLOCKS

10-3.3.1 ARITHMETIC ELEMENT BUSY INTERLOCK LEVEL (AEB)

10-3.3.2 ARITHMETIC ELEMENT PREDICT INTERLOCK FLIP-FLOP (AEP)

10-3.3.3 ARITHMETIC ELEMENT INTERLOCK LEVEL (AEI)

10-3.3.4 ARITHMETIC ELEMENT INTERLOCK DURATIONS

10-3.4 MISCELLANEOUS INTERLOCKS

10-3.4.1 E REGISTER BUSY INTERLOCK (EB)

10-3.4.2 Q REGISTER BUSY INTERLOCK (QB)

10-3.4.3 F MEMORY INTERLOCK (FI)

10-3.4.4 X REGISTER BUSY INTERLOCK (XB)
10-3.4.5 X WRITE REGISTER INTERLOCK (XW)
10-4 INTERLOCK CONTROL LEVELS
10-4.1 INTRODUCTION
10-4.2 PK, QK AND CSK INTERLOCK START LEVELS
10-4.2.1 INSTRUCTION MEMORY CYCLE INTERLOCK START LEVEL (PI^{START_1})
10-4.2.2 DEFERRED ADDRESS MEMORY CYCLE INTERLOCK START LEVEL (PI^{START_2})
10-4.2.3 OPERAND MEMORY CYCLE INTERLOCK START LEVEL (QI^{START})
10-4.2.4 CHANGE OF SEQUENCE CYCLE INTERLOCK START LEVEL (CSI^{START})
10-4.3 SEQUENCE CHANGE INTERLOCK LEVELS
10-4.3.1 CHANGE SEQUENCE INTERLOCK LEVEL ($PI^{CH\ SEQ}$)
10-4.3.2 ARITHMETIC ELEMENT CHANGE SEQUENCE INTERLOCK LEVEL ($PI^{AE\ CH\ SEQ}$)
10-4.3.3 LEAVE SEQUENCE INTERLOCK LEVEL ($PI^{LV\ SEQ}$)
10-4.3.4 INTERLOCK WAIT LEVEL (PI^{WAIT})
10-4.4 MISCELLANEOUS COUNTER START INTERLOCK LEVELS
10-4.4.1 F MEMORY COUNTER START LEVEL ($\overset{START}{\longrightarrow}$ FK)
10-4.4.2 X MEMORY WRITE COUNTER START LEVEL ($\overset{START}{\longrightarrow}$ XWK)
10-4.4.3 ARITHMETIC ELEMENT COUNTER START LEVEL ($\overset{START}{\longrightarrow}$ AK)
10-4.4.4 ARITHMETIC ELEMENT STEP COUNTER START LEVEL ($\overline{ASK}$ + 1 $\longrightarrow$ ASK)
10-4.4.5 DELAY SYNCHRONIZATION COUNTER START LEVEL ($\overset{START}{\longrightarrow}$ DSK)
10-5 COUNTERS
10-5.1 GENERAL DESCRIPTION
10-5.2 INSTRUCTION COUNTER (PK)
10-5.3 OPERAND COUNTER (QK)
10-5.4 CHANGE SEQUENCE COUNTER (CSK)
10-5.5 X MEMORY WRITE COUNTER (XWK)
10-5.6 F MEMORY COUNTER (FK)
10-5.7 ALARM DELAY COUNTER (ADK)
10-5.8 ARITHMETIC ELEMENT COUNTER (AK)
10-5.9 ARITHMETIC ELEMENT STEP COUNTER (ASK)

LIST OF FIGURES

10-1 START CONTROL
10-2 STOP CONTROL
10-3 MEMORY SELECTION IN-OUT AND OPERATION CODE ALARMS
10-4 PARITY ALARMS AND DRIVERS
10-5 MISCELLANEOUS ALARMS
10-6 AL LEVEL LOGIC
10-7 CHIME LOGIC
10-8 ALARM DELAY COUNTER
10-9 PI_1 $INSTRUCTION_1$ INTERLOCK
10-10 PI_1^1 INTERLOCK DURATIONS
10-11 PI_2 $INSTRUCTION_2$ INTERLOCK
10-12 PI_3 $INSTRUCTION_3$ INTERLOCK
10-13 PI_4 $INSTRUCTION_4$ INTERLOCK

10-14 PI_5 $INSTRUCTION_5$ INTERLOCK
10-15 PI_2 AND PI_5 INTERLOCK DURATIONS VS. PK CYCLE
10-16 ARITHMETIC ELEMENT INTERLOCK LEVELS
10-17 AEP ARITHMETIC ELEMENT PREDICT INTERLOCK
10-18 AEB, AEI AND AEP^1 INTERLOCK DURATIONS ON $QKIR^{AK}$ INSTRUCTIONS
10-19 AEB, AEI AND AEP^1 INTERLOCK DURATIONS ON $PKIR^{OPR\ AE}$ INSTRUCTIONS
10-20 EB EXCHANGE ELEMENT BUSY INTERLOCK
10-21 QB INTERLOCK
10-22 QB^1 AND EB^1 INTERLOCK DURATIONS
10-23 FI CONFIGURATION INTERLOCK
10-24 XB INDEX REGISTER BUSY INTERLOCK
10-25 XW INTERLOCK
10-26 INTERLOCK START LEVELS
10-27 INTERLOCK LEVELS
10-28 FK COUNTER START LEVEL
10-29 XWK COUNTER START LEVEL
10-30 AK START AND ASK - COUNT LEVEL LOGIC
10-31 DSK COUNTER START LEVEL
10-32 PK INSTRUCTION COUNTER (MEMORY SECTION)
10-33 PK INSTRUCTION COUNTER (INSTRUCTION SECTION)
10-34 QK OPERAND COUNTER (MEMORY SECTION)
10-35 QK OPERAND COUNTER (INSTRUCTION SECTION)
10-36 CSK AND DSK CYCLES IN CSK COUNTER
10-37 CSK CHANGE SEQUENCE COUNTER
10-38 XWK INDEX WRITE COUNTER
10-39 FK CONFIGURATION COUNTER
10-40 FK COUNTER
10-41 AK REGISTER LOGIC
10-42 ASK SHIFT REGISTER LOGIC

CHAPTER 10
CONTROL ELEMENT

10-1 INTRODUCTION

This chapter will discuss the logical design of the Control Element. Chapter 6 gave brief functional descriptions of most of the components in the Control Element and indicated how the Control Element itself was related to the rest of the computer. While this chapter is organized in much the same way as Chapter 6, i.e., it covers the following general topics:

Start-Stop Control
Interlocks
Interlock Levels
Counter

It is unlike Chapter 6 in that it deals with these topics at the level of TX-2 Block Schematic information.

This chapter is also complementary to Chapter 9 which discusses the dynamics of the computer in terms of counter interlocking.

An elementary view of the needs the Control Element fills is given in Chapter 5, which discusses in broad terms the general timing and control problem.

10-2 START-STOP CONTROL

10-2.1 GENERAL DESCRIPTION. The output of the Start-Stop Control is a set of levels which enter into the interlock start level logic.

The Start-Stop Control system is divided into two subsystems: (1) a start control, and (2) a stop control.

The start control takes into account the factors which influence whether or not the computer runs, i.e., the state of the start-stop buttons on the console, and the condition of the alarms and the alarm suppress buttons. (The alarms, alarm controls, and the alarm delay counter (ADK) will be discussed in Section 10-2.5.)

The stop control takes into account the factors which influence the effective speed of operation of the computer, i.e., the modes of operation (high speed, low speed and low speed repeat), and the various stop buttons and switches.

The start system generates a level which is used by all the interlock start levels. The stop system generates individual levels for each of the interlock levels.

10-2.2 START CONTROL. The start control system is shown in Fig. 10-1. It consists primarily of two flip-flops that are used as a synchronizer. These flip-flops are $START_1$ and $START_2$.

$START_2^1$ is a necessary condition for the generation of the interlock start levels. When the PB START pulse is generated, the $START_1$ flip-flop will be set to ONE if the ADK counter is in its resting state. (ADK^{00} indicates that no alarm conditions exist.) The $START_2$ flip-flop is then synchronously set to ONE after the $START_1$ flip-flop is set, again providing that the ADK counter is in its ADK^{00} resting state.

The $START_1$ flip-flop is cleared when the PB STOP push button is actuated or when the AL · $\overline{\text{AUTO START}}$ condition exists. The $START_2$ flip-flop is cleared when the $START_1$ flip-flop is cleared or when the AL condition exists. AL indicates an alarm condition is present. (See Sect. 10-2.5.6.)

Note that in all cases the $START_2$ flip-flop changes state synchronously with the α (alpha) timing pulses.

10-2.2.1 AL · $\overline{\text{AUTO START}}$ CONDITION. When the START pulse is generated (see Fig. 10-1), the $START_1$ flip-flop is set. At the next α timing pulse, the $START_2$ flip-flop will be set. With both START flip-flops set, the computer will operate. If both the AL and $\overline{\text{AUTO START}}$ levels exist, then both the $START_1$ and $START_2$ flip-flops will be cleared. In this situation the computer will not start again until the alarms which generated AL have been cleared and the START push button is pressed again.

10-2.2.2 AL · AUTO START CONDITION. If when both START flip-flops are set the AL and AUTO START levels occur, only the $START_2$ flip-flop will be cleared. This stops the computer, but leaves the $START_1$ flip-flop set. This condition continues to exist until the ADK counter returns to ADK^{00}. At that time, the $START_2$ flip-flop is set again. This permits the computer to restart automatically after a delay equal to the duration of the operation of the ADK counter cycle.

10-2.2.3 PB STOP PUSH BUTTON CONDITION. If, while the START flip-flops are set, the PB STOP push button is actuated, the $START_1$ flip-flop will be cleared. At the next α timing pulse, the $START_2$ flip-flop will also be cleared and the computer will stop running.

10-2.3 STOP CONTROL. The stop control system is shown in Fig. 10-2. It consists primarily of two flip-flops used as a synchronizer. These flip-flops are $STOP_1$ and $STOP_2$. The system also includes the push button mode flip-flops, and the four flip-flops which stop or prevent an operand (QKS), change of sequence (CSKS), instruction (PKS_1) and/or a deferred address (PKS_2) cycle from starting. The condition of these "stop" flip-flops determines which mode the computer will stop in when the computer is running.

The $STOP_1$ flip-flop is set when the PB START push button is actuated and the ADK counter is in its ADK^{00} resting state. The $STOP_2$ flip-flop is then cleared by an α pulse after both the $STOP_1^1$ and $START_2^1$ conditions occur and certain other conditions are satisfied. These other conditions are that either the computer is not in the Low Speed Repeat (LSR) mode, or, if it is, that a Low Speed Oscillator (LSO) level is present. $STOP_2^0$ clears all of the stop flip-flops, i.e., CSKS, QKS, PKS_1 and PKS_2.

The $STOP_2$ flip-flop is set by an α pulse as soon as it is cleared if the computer is in either low speed mode. $STOP_2^1$ sets those stop flip-flops that have corresponding stop toggle switches on the console actuated. Since the stop flip-flops enter into the interlock start levels and inhibit these levels when they are set, it is clear that the stop system affects the operation of the computer only when it is in the low speed mode.

The mode of operation flip-flops are set by actuating push buttons on the console.

10-2.3.1 LOW SPEED PUSH BUTTON MODE STOP CONTROL. When the low-speed-push-button-mode push button is actuated, the LSPB flip-flop is set. Note that $STOP_2$ will be set at this time if it was previously clear. If the PB START push button is then actuated, the $STOP_1$ and $START_1$ flip-flops will be set. The following events will then be initiated by a succession of α pulses:

- α_n - $START_2$ flip-flop set
- α_{n+1} - $STOP_2$ flip-flop cleared
- α_{n+2} - All stop flip-flops and $STOP_2$ cleared
- α_{n+3} - The one interlock start level which has been generated allows the corresponding counter to start. At the same time, all the stop flip-flops are set which have the corresponding STOP switches set.

Thus the interlock start levels which correspond to the set STOP switches are inhibited. The counters corresponding to the interlock start levels cannot then be started until the START button is again actuated.

10-2.3.2 LOW SPEED REPEAT MODE STOP CONTROL. When the low-speed-repeat-mode push button is actuated, the LSR flip-flop is set. When the START push button is then pressed, the operation of the computer is identical to the Low Speed Push Button mode, except that $STOP_2$ will not be cleared until an LSO level occurs and that $STOP_1$ will not be immediately cleared by the next α pulse. The result is that $STOP_2$ is cleared for 0.4 microsecond whenever an LSO level occurs and that 0.4 microsecond later all the stop flip-flops are cleared for 0.4 microsecond. Immediately afterwards those stop flip-flops are set which correspond to set stop switches. The computer is hence able to run only until it tries to use an inhibited counter cycle. Since $START_2$ remains set until the STOP push button is actuated, the computer is essentially started every time the LSO level occurs.

Low Speed Oscillator (LSO). The low speed oscillator consists of two variable delay units, LSO_1 and LSO_2, coupled together so as to form an oscillator. When one unit turns itself off, it turns the other unit on. The two units are each set to be on for approximately the same amount of time. Although the two units as coupled together tend to oscillate by themselves, one of the units (LSO_1) is set whenever the Low Speed Repeat push button is pressed in order to guarantee oscillation. The output LSO level is generated once each complete cycle by the $LSO_1^1 \cdot LSO_2^0$ condition. The frequency of oscillation can be varied over the range 0-500KC by two knobs on the console.

10-2.3.3 HIGH SPEED MODE STOP CONTROL. The inputs to the LSR and LSPB flip-flops are arranged so that both flip-flops cannot be set at the same time. If both flip-flops are set when the power is turned on, the LSR flip-flop will be cleared by the first α pulse that occurs. If one flip-flop is set and the corresponding mode push button is pressed, both flip-flops will end up cleared. If one flip-flop is set and the push button for the other is pressed, then the first flip-flop is cleared and the other is set. If one flip-flop is set, then the computer is said to be in the corresponding low speed mode. If neither is set, then the computer is said to be in the high speed mode. When the computer is in the high speed mode and the PB START push button is actuated, the $STOP_1$ and $START_1$ flip-flops are set. The following events will then be initiated by a succession of α pulses:

α_n - $START_2$ is set

α_{n+1} - $STOP_2$ is cleared

α_{n+2} - The stop flip-flops CSKS, QKS, PKS_1 and PKS_2 are cleared. These stop flip-flops remain cleared and the operation of the computer is under control of the $START_2$ flip-flop.

10-2.4 SYNC SYSTEM. The Sync System provides the computer operator with a means of generating a pulse when certain specified states occur in parts of the computer. These states are specified by the position of selection switches on the Sync System control panel. The control panel and the computer console contain other switches which determine what effect the output pulses from the Sync System will have.

10-2.4.1 OUTPUT CONTROL SWITCHES. There are two sets of 31 selection switches on the Sync System control panel. Each set gates the same set of 31 input levels, but permits the operator to choose two different combinations of the levels. All the input levels selected by each set are separately "AND"ed. The two results are then "OR"ed in several ways to generate output pulse. Thus, if the input levels are designated by $L_1, \cdots, L_{31}$, and the two sets of selection switches are designated by $S_1, \cdots, S_{31}$, and $T_1, \cdots, T_{31}$ then the logical quantities

$$A_1 = (\overline{S}_1 + L_1) \cdots (\overline{S}_{31} + L_{31})$$

and

$$A_2 = (\overline{T}_1 + L_1) \cdots (\overline{T}_{31} + L_{31})$$

are formed.

An output pulse from the system can be used to stop the computer. Two different switches on the computer console, SYNC $STOP_1$ and SYNC $STOP_2$, determine, via two SYNC STOP flip-flops, which of the above two quantities will stop the computer. Specifically, the quantity

$$\text{SYNC STOP} = \text{SYNC STOP}_1^1 \cdot A_1 + \text{SYNC STOP}_2^1 \cdot A_2$$

is used to clear the START synchronizer and to generate a SYAL. After such an alarm the computer can be restarted by pressing the CALACO button.

The output pulse from the Sync System can also be used to sync test oscilloscopes. In this case two switches, called SELECTED $SYNC_1$ and SELECTED $SYNC_2$ determine directly which of the above two quantities will generate a sync pulse. Specifically, the quantity

$$\text{SELECTED SCOPE SYNC} = \text{SELECTED SYNC}_1 \cdot A_1 + \text{SELECTED SYNC}_2 \cdot A_2$$

is sent to various output BNC connectors on the main horizontal bar of the computer frame. The sync input to a test scope can then be easily connected so as to receive a sync pulse when specified states occur in the computer.

One of the output BNC connectors is connected directly to the Trap Sequence (No. 42 (o)). An output pulse can then be used to raise the flag of the Trap Sequence, as described in Chapter 15.

10-2.4.2 INPUT SELECTION SWITCHES. Sixteen of the thirty-one switches in each set of selection switches gate levels received from BNC connectors of the main horizontal bar of the computer frame. These levels are called B_1, B_2, C_1, C_2, D_1, D_2, E_1, E_2, F_1, F_2, MT_1, MT_2, MT_3, MT_4, IOI_1, and IOI_2, where the names indicate the section of the frame of the computer which contain the BNC connectors.

The other fifteen inputs correspond to wired in nets which detect coincidence between the state of a particular part of the computer and a corresponding set of toggle switches on the Sync System control panel. These fifteen inputs are PK_α, PK_{OP} ($PKIR_{OP}$), PK_{CF} ($PKIR_{CF}$), PK_H ($PKIR_H$), P, QK_α, QK_{OP} ($QKIR_{OP}$), Q, $N_{4.10}$, $M_{4.10}$, N_J, AK_α, AK_{OP} ($AKIR_{OP}$), ASK and $X_{2.9}$. For example, a coincidence net determines whether the five PK_α flip-flops agree in value with the settings of the five PK_α toggle switches. If so, then the corresponding input level to the selection switches is generated.

All fifteen of these nets are similar, aside from the number of switches and flip-flops involved, except for $N_{4.10}$ and $M_{4.10}$. In these two cases the additional switches are somewhat redundant since only the "ONE" value of the corresponding flip-flop can be detected by the coincidence net. Note, however, that either state of $X_{2.9}$ can be selected.

10-2.5 ALARMS AND ALARM INDICATORS. The general function of the various alarms was described in Chapter 6. In addition to the alarm flip-flop indicators, each alarm has an associated flashing indicator, which flashes each time the corresponding alarm condition occurs. The flashing indicators are variable delay units which clear themselves automatically after a 70 milliseconds delay. The clearing logic for the alarm flip-flops, as well as the logic for generating the alarm conditions, is shown on Figs. 10-3, 10-4 and 10-5.

An alarm flip-flop (and the associated flashing indicator) is set immediately when the corresponding alarm condition is generated. However, individual alarm suppression switches in the console determine whether the alarm conditions affect the operation of the computer.

All suppressed alarms are cleared when the PB Clear Suppressed Alarms push button is pressed. The pulse generated by the button sets a variable delay unit, CA_2, which generates a 0.4 microsecond level. This level is ANDed with the console alarm suppress switch levels to clear the corresponding alarm flip-flops. For example, PSAL is cleared by $\alpha \cdot CA_2^1 \cdot PSAL_{SUP}$. In this way all the suppressed alarms are cleared simultaneously.

All unsuppressed alarms are cleared when the Alarm Delay Counter (ADK) reaches state ADK^{10}. For example, PSAL is cleared by $\alpha \cdot ADK^{10} \cdot \overline{PSAL_{SUP}}$. The occurrence of any unsuppressed alarm causes the generation of the AL level. AL stops the computer and starts ADK. When ADK reaches state ADK^{11}, it will not proceed to state ADK^{10} unless one of the following two conditions are satisfied:

1) The AUTO START switch is turned on.
2) The PB Clear Unsuppressed Alarms push button is pressed.

The presence of the AUTO START level permits ADK to proceed through state ADK^{10} to state ADK^{00}, whereupon the computer is promptly allowed to restart. On the other hand, if the AUTO START switch is not turned on, so that the computer stops, and the PB Clear Unsuppressed Alarms is pressed, then the CA_1 flip-flop will be set. The next α pulse after CA_1 is set will clear both CA_1 and ADK_1. The computer then proceeds as in the first case. Note that CA_1 acts as a synchronizer so that ADK will not be affected by the pulse generated from the push button except when ADK is in state ADK^{11}.

10-2.5.1 MEMORY SELECTION ALARMS. (See Fig. 10-3.)

P Memory Cycle Selection Alarm (PSAL). The PSAL flip-flop is set whenever a memory cycle is performed in which P is used as the memory address register and the address in P does not refer to any of the memories logically connected to the computer at the time $(\overline{PKM^{LEGAL}})$. The alarm occurs at $PK^{09\alpha}$ during instruction cycles.

Q Memory Cycle Selection Alarm (QSAL). The QSAL flip-flop is set whenever a memory cycle is performed in which Q is used as the memory address register and the address in Q does not refer to any of the memories logically connected to the computer at the time $(\overline{QKM^{LEGAL}})$. The alarm occurs at $PK^{09\alpha}$ during deferred address cycles and at $QK^{09\alpha}$ during operand cycles.

10-2.5.2 IN-OUT ALARMS. (See Fig. 10-3.)

In-Out Selection Alarm (IOSAL). The IOSAL flip-flop is set whenever an IOS instruction is performed which tries to change the mode (IOS 3XXXX) or select a new drive (IOS 6XXXX) of an In-Out unit which is in the maintenance mode. The alarm occurs at $PK^{24\alpha}$ of the IOS instruction.

In-Out Miss Indication Alarm (MISAL). The MISAL flip-flop is set by the $IOCM^{MISIND}$ level. The alarm indicates that some In-Out unit has missed a line of data.

10-2.5.3 OPERATION CODE ALARM (OCSAL). (See Fig. 10-3.) The OCSAL flip-flop is set whenever the computer attempts to execute an instruction with an undefined operation code. The alarm can occur if an instruction word with an undefined OP code is read out of memory or if an AOP instruction specifies an undefined OP code in bits $N_{2.6 - 2.1}$. In the first case, the alarm is generated at $PK^{15\alpha}$ of the PK cycle in which the OP code is interpreted, i.e., when PI_2^0. (PI_2^0 indicates that no deferred address cycles remain to be performed.) In the second case, the alarm occurs at the time the AK counter is started during the AOP, i.e., when the content of $AKIR_{OP}$ is being interpreted.

10-2.5.4 MEMORY PARITY ALARMS. (See Fig. 10-4.)

M Parity Alarm (MPAL). The MPAL flip-flop is set whenever the parity check circuit in the M register indicates that the operand word just read out of memory into M has an even parity.

The alarm is generated 1.2 microseconds after $QK^{11\beta}$ ($QK^{11\beta}$ is the latest time at which a strobe can occur during a QK cycle) and hence occurs at varying QK states depending on the instruction being executed. The $\boxed{1}\rightarrow$ MPAL logic determines the time at which the parity is checked. Note that the alarm is not generated when a QSAL is generated (i.e., when $\overline{QKM^{LEGAL}}$), nor when the V Memory is used.

N Parity Alarm (NPAL). An NPAL is generated whenever an instruction word or deferred address word which has an incorrect parity is read out of memory into the N register. This alarm is generated in a manner similar to the MPAL discussed above. However, in this case $PK^{13\alpha}$ always occurs 1.2 microseconds after $PK^{11\beta}$. ($PK^{11\beta}$ is the latest strobe time.)

F Parity Alarm (FPAL). An FPAL is generated whenever a word is read out of the F Memory into the $QKIR_{CF}$ register that has an incorrect parity. The parity check is made 0.5 microseconds after a word is strobed into $QKIR_{CF}$. Only one readout occurs during a normal FK cycle but four readouts occur during a FLG. Note that during SPF and SPG instructions, and when register 00 is selected, the words readout are not used. In these cases the parity is not checked.

X Parity Alarm (XPAL). The XPAL flip-flop is set whenever the parity check circuit in the X register indicates that the X Memory word just read out of memory into the X register has an even parity. The alarm is generated at $PK^{15\alpha}$ or $CSK^{04\alpha}$. This is never less than 0.8 microsecond after the word is strobed into the X register from the X Memory. Note that, although a zero word is placed in the X register whenever X Memory register 00 is selected, the parity of the word is correct since XP is set.

10-2.5.5 MISCELLANEOUS ALARMS (See Fig. 10-5.) All the previous alarms have virtually identical design, except for the individual alarm condition logic which distinguishes them. The following alarms have few similar features, although each has an alarm flip-flop and can stop the computer.

T Memory Selection Alarm (TSAL). This alarm is designed to protect the circuitry in the T Memory by turning off the read-write currents whenever the TSAL alarm flip-flop is set. This occurs whenever a voltage transition takes place on one of the memory address MAS_T lines while the read-write currents are turned on. The TSAL flip-flop can be cleared only by a $\llcorner$PRESET$\blacklozenge$ CE level. There is no flashing indicator associated with this alarm and it cannot be suppressed, but on the other hand the alarm does not stop the computer.

Synch System Alarm (SYAL). This alarm is generated whenever the Synch System generates a SYNCH STOP pulse. This level sets both the SYAL alarm flip-flop and a flashing indicator. The flip-flop can be cleared only by pressing the CLEAR UNSUPPRESSED ALARMS push button. This alarm stops the computer, but does it directly by clearing the $START_1$ flip-flop, rather than by starting the alarm delay counter ADK.

Mousetrap Alarm (Mousetrap). This alarm is used to detect and remember various malfunctions of the computer. Currently it determines whether the S Memory read-write flip-flops SR_U and SR_V are cleared at a time when they should remain set during an S Memory cycle. Such an event will generate a $\overset{1}{\longrightarrow}$ MOUSETRAP level. There is neither a flashing indicator nor a suppression switch associated with the Mousetrap alarm. When the alarm is set it will always stop the computer in the same way that a normal unsuppressed alarm would. It is also cleared in the normal manner.

10-2.5.6 ALARM LEVEL (AL). (See Fig. 10-6.) Most of the alarm conditions which stop the computer do so by generating the AL level. This level clears the $START_2$ flip-flop in the start control system and starts the alarm delay counter ADK. It will also clear the $START_1$ flip-flop if the AUTO START switch is not turned on, so that the computer will not restart by itself when ADK returns to ADK^{00}.

The AL level is simply the OR of the MOUSETRAP alarm and all of the following alarms which are not unsuppressed: PSAL, QSAL, MISAL, IOSAL, OCSAL, MPAL, NPAL, FPAL and XPAL. The AL level can be removed only by clearing all the set alarm flip-flops which generate it. Note that SYAL can also stop the computer, but it doesn't use the AL level to do this.

10-2.5.7 CHIME CONTROL. Two different audible indications of an alarm condition are generated using a two-tone chime. One tone indicates that a suppressed alarm, i.e., one which does not stop the computer, has occurred. The other tone indicates that an unsuppressed alarm, i.e., one which stops the computer (at least momentarily), has occurred. Each of these audible indications can be suppressed by switches on the console.

Chime on Suppressed Alarms. As shown on Fig. 10-7, this chime is generated by the flashing indicators associated with the alarm flip-flops. Only QSAL, PSAL, MISAL, IOSAL, OCSAL, MPAL, NPAL, FPAL and XPAL can generate this indication, and then only when the Suppress Chime on Unsuppressed Alarm switch is turned off.

Chime on Unsuppressed Alarms. This chime is generated when either the ADK counter has been started or when a SYAL has been generated. In the first case the chime level lasts as long as ADK is in state ADK^{01}, which is determined by ALD_1, whenever a MOUSETRAP or an unsuppressed QSAL, PSAL, MISAL, IOSAL, OCSAL, MPAL, NPAL, FPAL or XPAL occurs. In the second case the level is generated directly by the flashing indicator associated with SYAL.

10-2.5.8 ALARM DELAY COUNTER (ADK). This counter is started whenever an AL alarm level is generated. After it has started no other PK, QK or CSK cycle can begin until it has returned to its resting state. The purpose of the counter is to stop the computer, and then after the necessary operations have been performed, to allow the computer to be started again in a controlled manner. The counter uses variable delay units to slow down its rate of counting to about 0.1 seconds per cycle.

The logic of the counter is illustrated in Fig. 10-8. The AL level sets the first delay unit ALD_1. ADK_1 is then set and the counter remains in the ADK^{01} state until ALD_1 clears itself synchronously with an α pulse. During this time the CHIME ON UNSUPPRESSED ALARMS is sounded. Presumably transient conditions, which might have caused the AL level, have by then had a chance to disappear.

The counter next enters the ADK^{11} state by setting ADK_2, and also sets the second delay unit, ALD_2. While ALD_2 is set the first delay unit recovers. During this time the computer simulates the pressing of the STARTOVER push button, if the PASOFA switch is on, and simulates the PRESET button, if both the PASOFA and the AUTO START switches are on. Note that in the latter case only the Control Element exclusive of the start-stop control is preset.

When ALD_2 clears itself the counter will wait in the ADK^{11} state until the CLEAR UNSUPPRESSED ALARMS button is pressed unless the AUTO START switch is on. The CA_1 flip-flop is set when this button is pressed. When either CA_1^1 or the AUTO START switch is on, ADK can proceed to state ADK^{10} wherein the unsuppressed alarms which generated the AL level are cleared, and then to state ADK^{00} where it remains until AL is generated again.

Note that CA_1 and ADK_2 together act as a synchronizer for CLEAR UNSUPPRESSED ALARMS pulses.

10-3 INTERLOCKS

10-3.1 GENERAL DESCRIPTION. The general function of the various interlock flip-flops was described in Chapter 6. This chapter will discuss the specific logic that sets and clears these interlocks.

In certain cases two or more interlocks will have a related function. The times at which these interlocks are set can overlap. To clarify the time relation of the interlocks, figures will be given to show the relative times at which these interlocks are set and cleared in terms of the basic counter cycles.

10-3.2 INSTRUCTION INTERLOCKS

10-3.2.1 INSTRUCTION INTERLOCK$_1$ (PI_1). This interlock determines whether a PK instruction word or QK operand word memory cycle can occur.

The logic for setting and clearing PI_1 is shown in Fig. 10-9. A graphic illustration of the duration of PI_1^1 is shown in Fig. 10-10.

PI_1 is set if an instruction requires a QK cycle. It is then cleared after the QK cycle starts, thereby indicating that the next instruction (PK) or change of sequence (CSK) cycle can begin. (See also PI_3 discussion.) PI_1 is always set at essentially the same time during a PK cycle, but it is cleared at various times during the QK cycle that follows. The specific time depends on the kind of operation being performed. Thus PI_1^0 provides additional timing information about when the next PK or CSK cycle is permitted to start.

PI_1 is set during instructions which require QK cycles either at the end of the instruction word PK memory cycle or, if deferred address is required, at the end of the PK ultimate cycle. The set pulse occurs at $PK^{22\alpha}$ if the computer can proceed with the execution of the instruction, i.e., if the $\overline{PI^{WAIT}}$ level exists, or $PK^{23\alpha}$ if the computer has been forced to wait before the decision is made to continue.

PI_1 is usually cleared as soon as the QK cycle begins, i.e., when QK is in the QK^{00} state and the QI start level exists. However, any 1X (AUX, RSX, EXX, ADX, DPX or SKM) or LF (SPF or SPG) operation code postpones clearing PI_1 until later during the QK cycle. For example, in the case of ADX, PI_1 is not cleared until $QK^{16\alpha}$. In the case of SPF and SPG, $PKIR_{CF}$ is protected from the effects of the next PK or CSK cycle until the FK cycle is finished with it. In the case of SKM, N_J is protected. In the remaining instructions PI_1^1 helps protect the $N_{2,1}$ input to the X Adder until the XA output has been used. Note that although in some cases PI_1 is cleared twice during a QK cycle only the first of such pulses has any affect on the interlock.

10-3.2.2 INSTRUCTION INTERLOCK$_2$ (PI_2). PI_2 is set during a PK instruction word memory cycle whenever deferred address cycles are required. It then remains set, and serves to distinguish subsequent PK cycles which are associated with the deferred addresses. It is not cleared until the ultimate deferred address cycle (the one which does not obtain a word from memory).

The logic that sets and clears PI_2 is shown in Fig. 10-11. Note that the term $CSK^{07\alpha} \cdot SS^{CH\ SEQ}$ is present only for reasons of wiring convenience and has no effect on the operation of the interlock. (See also PI_5 discussion.)

10-3.2.3 INSTRUCTION INTERLOCK$_3$ (PI_3). This interlock determines whether a PK instruction word memory cycle or a CSK change sequence cycle is to occur next. The logic that sets and clears PI_3 is illustrated in Fig. 10-12.

PI_3 must be set before a change sequence cycle can begin. It can be cleared during normal operation only at $CSK^{04\alpha}$ of a CSK cycle.

On the other hand, there are three kinds of occasions at which PI_3 can be set:

1) One of these times is simply at $CSK^{07\alpha}$ during a change of sequence cycle. A change of sequence always takes place to the highest priority sequence which wants attention and usually no flags can be set during the CSK cycle. Thus the $SS^{CH\ REQ}$ level usually exists at $CSK^{07\alpha}$ when the CSK cycle is ending. However, if the sequence meta bit on the program counter of the new sequence is set, and the Trapping Sequence is trapping on such bits (see Chapter 15), then the flag of the Trapping Sequence will be raised and $SS^{CH\ REQ}$ might exist at $CSK^{07\alpha}$. In this case PI_3 is set and the change of sequence is followed by another change of sequence, this time the change of sequence is to the Trapping Sequence.

2) The second class of timing covers the situations when PK is in a waiting state and the delay synchronizer counter DSK is running. In these cases DSK will stop when interlock conditions determine either that PK can continue from where it was stopped when DSK started or that PK must go to PK^{00} (if it is not already there) and a change of sequence cycle should start. There are three such situations:

a) PK can already be in PK^{00} while DSK is counting if an instruction dismissed the current sequence at a time when no sequence wanted attention. In this case DSK cycles until some sequence wants attention ($SS^{ATT\ REQ}$) at $CSK^{11\alpha}$. At that time, PI_3 is set and a change of sequence occurs next. Two special situations must be taken into account. One situation is that if the highest priority sequence which wants attention is sequence zero (KD^{00}) then the change of sequence will always occur, even though the computer is already in sequence zero. This means that sequence zero will, once it has been dismissed, start up again at the TSP address when its flag is raised again. The other situation is that if the highest priority sequence which wants attention is the current sequence ($K^{eq\ JC}$), then, with the above exception about sequence zero, PI_3 will not be set since no CSK cycle is required.

b) PK can be waiting in PK^{02} while DSK is counting if the Arithmetic Element is busy when PK attempts to obtain an instruction or deferred address word from it. In this case PI_3 will be set if the $PI^{AE\ CH\ SEQ}$ level exists at $CSK^{11\alpha}$. $PI^{AE\ CH\ SEQ}$ indicates, as we will shortly see, that a higher priority sequence wants attention and the last instruction executed did not "hold".

c) PK can be waiting in PK^{23} while DSK is counting if either a TSD instruction tries to use an In-Out unit which is not ready ($IOCM^{BB}$) or because an instruction tries to use the Arithmetic Element while it is busy (AEI). In either case, PI_3 will be set if the $PI^{LV\ SEQ}$ level exists.

3) The third class of time covers the situations when PK is counting and a change of sequence condition occurs.

One of these conditions can occur at the end of the PK memory cycle at PK^{22} and covers the same situations which are covered at PK^{23} above and again involve the $PI^{LV\ SEQ}$ level.

The others occur either at $PK^{24\alpha}$ or $PK^{31\alpha}$ of instructions which do not hold or which dismiss. If a sequence which has its flag up can generate the $PI^{CH\ SEQ}$ level, then PI_3 is set and the PK cycle of the instruction is followed by a CSK change of sequence cycle.

10-3.2.4 INSTRUCTION INTERLOCK$_4$ (PI_4). PI_4 simply remembers the hold value of the last instruction executed. The logic that sets and clears PI_4 is shown in Fig. 10-14.

Since the decision to execute an instruction is indicated at the earliest by the decision for PK to advance to state $PK^{24\alpha}$, this is also the earliest time at which the PI_4 flip-flop can be changed. Whatever hold value is then placed in PI_4 is held until the next time a decision is made to execute an instruction.

The PI_4 interlock is used in the interlock level logic to decide whether the $SS^{CH\ REQ}$ level can contribute to sequence change decisions.

10-3.2.5 INSTRUCTION INTERLOCK$_5$ (PI_5). PI_5 is similar to PI_2 in that it is set during a PK instruction word cycle whenever deferred address cycles are required. The logic for setting and clearing PI_5 is shown in Fig. 10-14.

However, there are two kinds of deferred address cycles: (1) the deferred address cycles which require memory words, and (2) the ultimate deferred address cycle which does not require a memory word.

PI_2 and PI_5 serve to distinguish the three kinds of PK cycles, as shown in Fig. 10-15.

Initially both PI_2 and PI_5 are zero during the instruction word cycle. If the instruction word requires a deferred address, i.e., $N^1_{2.9}$, then PI_2 is set at $PK^{12\alpha}$ and PI_5 at $PK^{14\alpha}$. Both interlocks remain set during subsequent deferred address memory cycles until finally a deferred address word is obtained in which $N^0_{2.9}$. At $PK^{14\alpha}$ during this cycle, PI_5 is cleared. The next PK cycle is then identified as the "ultimate" cycle, since the PI^1_2 and PI^0_5 interlock condition exists at the beginning of the cycle. PKA (see Chapter 11) then remains cleared throughout this PK cycle so that no memory is selected and no memory word is read out. During the first part of the ultimate PK cycle the sum of the base address and the selected index register in the last deferred address is placed in $N_{2,1}$. The index bits from the original instruction word are retrieved from $QKIR_{CF}$ and placed in $N_{3.6\ -\ 3.1}$.

The logic performed in the ultimate cycle after PI_2 is cleared at $PK^{13\alpha}$ is, aside from memory logic, identical to the logic which would have been performed after $PK^{13\alpha}$ in the initial instruction word memory cycle if no deferred addresses were required. This can be seen by examining the logic performed in $PK^{14\alpha}$ through $PK^{22\alpha}$; nearly all of the logic will be seen to contain a factor of PI_2^0.

After the ultimate cycle, PK continues on with the execution of the instruction using the final effective base address.

10-3.3 ARITHMETIC ELEMENT INTERLOCKS. Registers in the Arithmetic Element can be used either as Memory Element flip-flop storage registers or as storage registers for the intermediate and final results of arithmetic computations. The instructions involved in this second case can be divided into two classes: (1) the simple load and store type instructions, and (2) the more complex add and shift type instructions. The more complex instructions generate either the $PKIR^{OPR}AE$ or $QKIR^{AK}$ levels (see Chapter 14), and make use of a variety of Arithmetic Element interlocks.

10-3.3.1 ARITHMETIC ELEMENT BUSY INTERLOCK LEVEL (AEB). This interlock level simply indicates whether or not the Arithmetic Element control counter AK is in its AK^{00} resting state. This is shown in Fig. 10-16. When the $\overline{AEB}$ level exists, i.e., AK is in its AK^{00} resting state, then it is permissible for immediate use to be made of the flip-flop registers in the Arithmetic Element.

Note that AEB is an interlock control level and not an interlock flip-flop. It is discussed here instead of Section 10-4 only for convenience in grouping together the Arithmetic Element interlock conditions.

10-3.3.2 ARITHMETIC ELEMENT PREDICT INTERLOCK FLIP-FLOP (AEP). The AEP flip-flop optimizes the speed of the Arithmetic Element instructions by predicting when the current use of the Arithmetic Element will end. The logic setting and clearing this flip-flop is shown in Fig. 10-17.

The interlock is set whenever an instruction which might use both the AK and ASK counters starts to make use of the Arithmetic Element. This occurs at $PK^{26\alpha}$ in all AOP instructions and at $QK^{14\alpha}$ in all other instructions which use both AK and ASK.

The interlock is then cleared at some time during the AK cycle of these instructions. This time is at most 2.8 microsecond before the end of the AK cycle, i.e., at most 2.8 microseconds before the $\overline{AEB}$ level occurs.

10-3.3.3 ARITHMETIC ELEMENT INTERLOCK LEVEL (AEI). This interlock level is discussed here, rather than later to keep it in the context of the two other Arithmetic Element interlocks. AEI is generated by the logic shown on Fig. 10-16. During AOP instructions, AEI is simply equal to AEP^{1}, but during $QKIR^{AESK}$ instructions it starts with $QK^{01\alpha}$, earlier then $QK^{14\alpha}$.

10-3.3.4 ARITHMETIC ELEMENT INTERLOCK DURATIONS. The duration of the Arithmetic Element interlock levels during $QKIR^{AE}$ instructions is illustrated in Fig. 10-18. Since AEP is needed only to help generate AEI, it will be ignored in the following discussions.

AEI is used in the various PK decision states of a subsequent instruction in order to help determine whether or not the PK cycle should continue. The PK cycle of a subsequent instruction can begin as early as 0.4 microseconds after the QK cycle of a $QKIR^{AK}$ instruction begins. The earliest PK decision state occurs at $PK^{02\alpha}$. Hence the AEI level must be generated earlier then the $QK^{14\alpha}$ time at which AEP is set. For this reason, the AEI level is started, as indicated by the logic, with $QK^{01\alpha}$. Also, the logic of the PK decision states is designed with a bias towards continuing the PK cycles, rather then towards causing a change of sequence. For this reason the AEI level ends when AEP is cleared, before the $\overline{AEB}$ level occurs. In all cases when the new instruction tries to make use of the Arithmetic Element before the Arithmetic Element is actually free for a new use, the AEB level is itself used in waiting state logic to hold up the new instruction. However, the AEB level itself is never used by a new PK cycle until late enough in a new instruction so that it doesn't need to be generated before $QK^{14\alpha}$ when AK is started.

The duration of the Arithmetic Element interlock levels during an AOP instruction is illustrated in Fig. 10-19. In this case AEI is simply equal to AEP^{1}. AEB and AEI are again used to influence the same PK decisions just described. Here, though, the levels reflect a different situation since AOP does not require a QK cycle and AK is started directly by PK during the execution of the AOP instruction. Both the AEB and AEI levels begin when AK is started. This is adequate for interlocking purposes since no subsequent PK cycle can begin until after the current PK cycle has ended, i.e., after the two levels have been generated. The termination of the levels is the same as in the previous case, since the levels end in a manner which is independent of whether the operation being performed in the Arithmetic Element was originated by an AOP or an ordinary OP code.

Note, however, that an AOP can call for an operation which does not use ASK, nor even AK. This can occur since the full six bits in $N_{2.6 - 2.1}$ of the AOP specify the operation. Some of these, the $11XXXX \cdot \overline{ADD} \cdot \overline{SUB} \cdot \overline{DSA}$, are just like the corresponding $QKIR^{AESK}$ operations which use both AK and ASK. Others, ADD + SUB + DSA, use AK but not ASK. In these, AEP is cleared as soon as AK starts, so that AEI ends before the PK cycle ends. Finally, the $\overline{11XXXX}$ codes are not defined and do not actually use AK. Hence, neither AK or ASK is used and both the interlock levels end before the PK cycle ends. In this last case, however, an OCSAL alarm is also generated at the present time.

10-3.4 MISCELLANEOUS INTERLOCKS

10-3.4.1 E REGISTER BUSY INTERLOCK (EB). The EB interlock indicates when the E register is busy during an operand cycle and can not yet be used for a new purpose. The logic for setting and clearing EB is shown in Fig. 10-20.

EB is set whenever a QK operand cycle starts, since the E register is used by all instructions which require an operand.

The interlock is then cleared as soon as the register is no longer needed during the operand cycle. This occurs at $QK^{21\alpha}$ during all store type instructions which are not placing the operand in the V_{FF} Memory. All other instructions, except SPG, use the E register until $QK^{23\alpha}$, at which time EB is cleared. SPG alone clears EB during a non-QK state. This situation will be understood after it is discussed in Chapter 16.

EB need be set only during operand cycles, and not during other uses of the E register, since conflicting demands for the use of the E register can arise only when one of these demands already is an operand cycle use of the E register.

10-3.4.2 Q REGISTER BUSY INTERLOCK (QB). The QB interlock indicates when the Q register is busy during an operand cycle. The logic for setting and clearing QB is shown in Fig. 10-21, and simply shows that QB is set when a QB cycle begins, and is cleared when it ends. As in the case with EB, if these are conflicting demands for Q, one of them always already involves an operand cycle.

Note that QB also indicates whether the M register is busy, since nearly all QK cycles require the use of M until $QK^{31\alpha}$.

The relative durations of EB and QB are graphically illustrated in Fig. 10-22.

10-3.4.3 F MEMORY INTERLOCK (FI). Before any SF (FLF or FLG) or JA (JPA, JNA and JOV) instructions can be executed, the F Memory interlock FI must be in a cleared state. Fig. 10-23 shows the logic for setting and clearing the FI interlock.

FI is cleared when a JA or SF type instruction is executed. This occurs at $PK^{13\alpha}$ either during the instruction word memory cycle if no deferred address cycles are required or during the ultimate deferred address cycle.

The FI flip-flop is set at $FK^{2\alpha}$ during an FLF or a JA type instruction. During a FLG instruction, FI is set at $FK^{7\alpha}$.

The FI interlock permits the contents of the F Memory registers to be obtained during the execution of these instructions earlier than they would otherwise be.

Note that FI is set during FLF and FLG in such a manner that FI can contribute to the QI^{START} level.

10-3.4.4 X REGISTER BUSY INTERLOCK (XB). The XB interlock serves two functions. The first is to indicate that the X Memory is busy with a READ-WRITE cycle. This is done by setting it whenever an X Memory READ cycle is initiated, and then clearing it when the WRITE cycle is performed. Thus XB is set whenever XR is set, and it is cleared during the XWK cycle. (See Fig. 10-24.)

The second function of the interlock is to predict, by 1.6 microsecond when the X Memory WRITE cycle will end. This prediction ability enables a PK cycle to start that much time before the WRITE cycle ends. Thus, XB is cleared at $XWK^{02\alpha}$, 1.6 microsecond before XW is cleared at $XWK^{06\alpha}$.

10-3.4.5 X WRITE REGISTER INTERLOCK (XW). This flip-flop is described more throughly in Chapter 12 in the section on the X Memory. XW is used as an interlock to indicate when an X Memory WRITE cycle has ended. The flip-flop is set at $XWK^{02\alpha}$ to turn on the X Memory write current and is cleared at $XWK^{06\alpha}$. (See Fig. 10-25.)

10-4 INTERLOCK CONTROL LEVELS

10-4.1 INTRODUCTION. The general function of the various interlock levels was described in Chapter 6. This section will examine their function in greater detail by examining the logic that generates these interlock control levels.

10-4.2 PK, QK AND CSK INTERLOCK START LEVELS. The following interlock start levels determine when instruction word, deferred address word, operand word and change of sequence counter cycles can begin. These are the basic counter cycles in the computer and are the only such start levels which are influenced by the console stop-start controls. The logic that generates these interlock start levels is shown on Fig. 10-26.

10-4.2.1 INSTRUCTION MEMORY CYCLE INTERLOCK START LEVEL ($PI^{START}{}_1$). The logic governing the start of a PK instruction word memory cycle depends on the $PI^{START}{}_1$ level. The fact that such a PK cycle is required is indicated by the fact that PI_2^0.

The logic first of all requires that the stop-start system permit such a cycle. This is indicated by $START_2^1 \cdot PKS_1^0 \cdot \overline{AL}$. Also, there must be neither a CSK nor a DSK cycle required, i.e., the $PI_3^0 \cdot CSK_4^0$ condition must be satisfied. The QK operand cycle, if one is required by the previous instruction, must have already started and progressed to the point where PI_1 is cleared. Also, any previous use of the X Memory (or X Adder) must be almost over (XB^0). Finally, the memory overlap conditions must be settled. The memory selected by P from which the instruction will be obtained must not be the same memory used by the previous QK operand cycle, i.e., the $\overline{(P^S Q^S + \cdots + P^V \cdot Q^V)}$ condition must be satisfied and the No Overlap switch must be off (NO^0). If these overlap conditions are not satisfied the previous QK cycle must be over (QB^0).

The $PI^{START}{}_1$ level is used by PK when it attempts to start an instruction word memory cycle, and also by the Memory Address Selector in order to turn on the selected memory (see Chapter 11).

10-4.2.2 DEFERRED ADDRESS MEMORY CYCLE INTERLOCK START LEVEL ($PI^{START}{}_2$). The logic governing the start of a PK deferred address word memory cycle depends on the $PI^{START}{}_2$ level. The fact that such a PK cycle is required is indicated by the fact that PI_2^1.

The stop-start control must permit such a cycle, i.e., the $START_2^1 \cdot PKS_2^0$ condition must be satisfied. The XWK cycle initiated by the previous PK cycle must be almost finished (XB^0), and the Q register must be available (QB^0).

The $PI^{START}{}_2$ level is used both to start PK and to turn on the selected memory in all the intermediate deferred address cycles. However, during the "ultimate" cycle it is used only to start PK. In this case only the stop-start control conditions are really relevant as start conditions. Note that PI_5^1 during all intermediate cycles and that PI_5^0 during the ultimate cycle.

10-4.2.3 OPERAND MEMORY CYCLE INTERLOCK START LEVEL (QI^{START}). The logic governing the start of a QK operand word memory cycle depends on the QI^{START} level.

The stop-start control must permit such a cycle, i.e., the $START_2^1 \cdot QKS^0$ condition must be satisfied. A QK cycle must be required (PI_1^1), and, in the case of SPF and SPG, the FK counter cycle must be almost over (FI^1).

The QI^{START} level is used both to start QK and to turn on the selected memory.

10-4.2.4 CHANGE OF SEQUENCE CYCLE INTERLOCK START LEVEL (CSI^{START}). The logic governing the start of a CSK change of sequence cycle depends on the CSI^{START} level.

The stop-start control must permit such a cycle, i.e., the $START_2^1 \cdot CSKS^0$ condition must be satisfied. Any QK cycle required by a previous instruction must have already started (PI_1^0), and PK must be in its $PK^{00\alpha}$ resting state. Also, since CSK makes immediate use of the X Memory and the E register, these must both be available, i.e., the $XW^0 \cdot XB^0$ and EB^0 conditions must be satisfied, respectively.

The CSI^{START} level is used only in the starting of CSK when a change of sequence cycle is required.

10-4.3 SEQUENCE CHANGE INTERLOCK LEVELS. These interlock levels are used in the "decision" states of PK (see Chapter 9), and also in the DSK cycles, to determine whether the computer should continue executing instructions in the current sequence, or change to a new sequence.

The logic generating these sequence change interlock levels is shown on Fig. 10-27.

10-4.3.1 CHANGE SEQUENCE INTERLOCK LEVEL ($PI^{CH\ SEQ}$). Towards the end of the PK cycle of each instruction executed the decision must be made whether to execute the next instruction in the current sequence. A change of sequence is made usually when either the current instruction does not hold ($PKIR_H^0$) and some other sequence of <u>higher</u> priority wants attention ($SS^{CH\ SEQ}$), or when the current instruction dismisses ($PKIR_H^0 \cdot PKIR^{DIS\ REQ}$) and <u>any</u> other sequence wants attention ($SS^{ATT\ REQ}$). The Sequence Selector levels can also indicate other reasons for changing sequences, as described earlier and in Chapter 12.

10-4.3.2 ARITHMETIC ELEMENT CHANGE SEQUENCE INTERLOCK LEVEL ($PI^{AE\ CH\ SEQ}$). If the computer attempts to obtain an instruction, deferred address or operand word from an Arithmetic Element flip-flop register and the Arithmetic Element is performing a $QKIR^{AESK}$ type instruction, indicated by the AEI level, then the computer is forced to wait and perhaps to make a change of sequence.

The $PI^{AE\ CH\ SEQ}$ level reflects the conditions for changing sequence. If the last instruction executed did not hold (PI_4^0) and some higher priority sequence wants attention ($SS^{CH\ REQ}$) while the Arithmetic Element is busy (AEI), then the level is generated.

10-4.3.3 LEAVE SEQUENCE INTERLOCK LEVEL ($PI^{LV\ SEQ}$). The $PI^{AE\ CH\ SEQ}$ level alone is used in the PK^{02} decision state. In the $PK^{22/23}$ decision state a more complex situation exists for determining whether to change sequences.

Here the change will occur if, when the $PI^{AE\ CH\ SEQ}$ level exists, the computer is attempting to execute an instruction which requires the Arithmetic Element ($PKIR^{AE}$) or obtains an operand from the Arithmetic Element ($PKIR^{QK} \cdot XA^{AE}$). The change will also occur if the computer is attempting to execute a TSD and either the selected IO unit is not available ($IOCM^{BB}$) or the QK cycle of a previous TSD is not finished ($QKIR^{TSD} \cdot QB^{1}$) when some other sequence wants attention ($SS^{ATT\ REQ}$).

10-4.3.4 INTERLOCK WAIT LEVEL (PI^{WAIT}). The computer can arrive in the $PK^{22/23}$ decision state and be unable to change sequence ($\overline{PI^{LV\ SEQ}}$) and also be unable to continue to execute the instruction. This latter condition is indicated by the PI^{WAIT} level. In fact, the instruction cannot be executed unless the $\overline{PI^{WAIT}}$ level exists.

It can be seen that PI^{WAIT} is similar to $PI^{LV\ SEQ}$ except for the absence of the Sequence Selector flag information (see Fig. 10-27). PI^{WAIT} simply indicates that the computer is attempting to perform an instruction which can not be done at the moment and that no sequence change condition exists either.

10-4.4 MISCELLANEOUS COUNTER START INTERLOCK LEVELS. The FK, XWK, AK, ASK and DSK counters are slaved to the PK, QK and CSK counters in the sense that their start conditions are usually generated at particular times during PK, QK and CSK cycles and last for only 0.4 microseconds. Hence, these counters usually must be in their resting states and must start immediately when the start conditions are generated. FK is the only one of these counters that uses an interlock flip-flop in its start level and even then only in a restricted class of start situations.

None of the start conditions for these counters directly reflect the console stop-start controls.

ADK was discussed earlier in the chapter in Sect. 10-2.5.8.

10-4.4.1 F MEMORY COUNTER START LEVEL ($\overset{\text{START}}{\longrightarrow}$FK). The logic governing the FK level logic is shown on Fig. 10-28.

Normally FK is started when the QK cycle starts during the execution of instructions which use the F Memory in order to obtain a configuration word. The two instructions which specify configurations (SPF and SPG) do not start FK until $QK^{13\alpha}$ after the new configurations have been placed in the E register. The Arithmetic Element jump instructions ($PKIR^{JA}$) and the instructions which file configurations ($PKIR^{SF}$) start FK by clearing the FI interlock in $PK^{13\alpha}$ and waiting for EB to be cleared. In these cases, FK then starts as soon as it returns to FK^{00}. Note that the QK cycles of FLF and FLG wait in QK^{00} until FI is set. Similarly, the PKEI cycles of JOV, JPA and JNA wait in $PK^{25\alpha}$ until FI is set.

10-4.4.2 X MEMORY WRITE COUNTER START LEVEL ($\overset{\text{START}}{\longrightarrow}$XWK). The XWK counter is started only when a word is to be written in the X Memory. The logic for starting the counter is shown in Fig. 10-29.

The counter is always started at $PK^{14\alpha}$ in a deferred address cycle (PI_2^1), and in the same state during instructions which do not make special use of the X Memory ($PKIR^{XM}$) during a PKEI cycle. These PKEI instructions all start XWK at $PK^{31\alpha}$, unless execution of the instruction is abandoned at $PK^{22/23}$ because a $PI^{LV\ SEQ}$ level occurs.

During the QK cycle of QK instructions which change the contents of an X Memory register, XWK is started as soon as the new word is placed in the X register. This occurs at $QK^{22\alpha}$ for RSX and EXX and at $QK^{31\alpha}$ for AUX. Note that in all these instructions XWK goes through an earlier cycle started at $PK^{14\alpha}$. This extra earlier cycle is performed so that the X Memory register read out during the PK cycle is not left in the cleared state in case the stop-start or alarm controls inhibit the QK cycle of the instruction. When the QK cycle does occur the X register is cleared out again at $QK^{13\alpha}$ in preparation for the QK XWK cycle.

XWK is also started in change of sequence cycles at $CSK^{04\alpha}$ in order to store the old program counter in the X Memory.

10-4.4.3 ARITHMETIC ELEMENT COUNTER START LEVEL ($\xrightarrow{\text{START}}$ AK). The logic for this level is shown in Fig. 10-30. AK is started at $PK^{26\alpha}$ during AOP and at $QK^{14\alpha}$ for all $QKIR^{AK}$ instructions.

10-4.4.4 ARITHMETIC ELEMENT STEP COUNTER START LEVEL ($\overline{ASK}$ + 1 ⟶ ASK). The logic for this level is shown in Fig. 10-30. This counter counts only during $QKIR^{AESK}$ instructions. It does not count in the usual manner (every 0.4 microseconds) since it counts once each time AK goes through a subcycle during $QKIR^{AESK}$ instructions. It starts from the state it was preset to by the ^{PRESET} ASK condition, and advances to state zero by this ASK count level.

10-4.4.5 DELAY SYNCHRONIZATION COUNTER START LEVEL ($\xrightarrow{\text{START}}$ DSK). The delay synchronization counter is started only when the computer cannot continue executing instructions in the current sequence and no change sequence condition exists. In these situations the DSK counter is started in order to synchronize signals arriving in the Sequence Selector from the In-Out Element.

CSK_4 can be set in any of the "decision" states of PK, but PK must be in one of the associated waiting states before DSK can start counting. These waiting states are $PK^{02\alpha}$, $PK^{23\alpha}$ and $PK^{00\alpha}$. XWK must also be in state zero when DSK starts counting.

The logic generating the DSK start level is shown on Fig. 10-31.

10-5 COUNTERS

10-5.1 GENERAL DESCRIPTION. The general function of the various counters was discussed in Chapter 6. In this section, the function of each counter will be discussed in greater detail. The actual count logic for each counter will also be discussed.

The dynamic interlocking of the counters is discussed in Chapter 9.

10-5.2 INSTRUCTION COUNTER (PK). The count logic for the PK counter is of two types. One type is used during the memory cycle when an instruction or deferred address word is obtained from memory. This is the part of the PK cycle which extends from PK^{00} to PK^{22}. The second type of count logic is used in $PK^{23/24}$ and in the additional PK states (PKEI) used in the execution of special instructions. The memory PK count logic is shown on Fig. 10-32 and the special instruction PK count logic is shown on Fig. 10-33.

Basically, one of three types of action can occur during the PK cycle:

1) PK can count into the next state.
2) PK can skip to some "preset" state.
3) PK can "wait" in a state until a decision to go on can be made.

The PK memory cycle carries the PK counter from PK^{00}, when the appropriate start condition is satisfied, through to PK^{22}. Skips occur from states 01, 02 or 06 to state 09, and from states 15 or 16 to 22, depending upon the memory selected. During the "ultimate" deferred address cycle (PKA^{0}) a similar sort of cycle occurs except that no memory is selected. The starting condition used when PK is in state zero depends upon whether an instruction cycle (PI^{0}_{2}) or a deferred address cycle (PI^{1}_{2}) is to occur.

Only one decision state occurs during a PK memory cycle. This is at $PK^{02\alpha}$ during a V_{FF} cycle where the AEI level is examined if the selected register is in the Arithmetic Element.

When PK finishes the instruction and all the deferred address cycles by arriving at $PK^{22\alpha}$ with PI^{0}_{2} another decision must be made about whether to execute the instruction. At this time, the PI^{WAIT} and $PI^{LV\ SEQ}$ levels are examined. PK cannot advance to PK^{24} and actually go on to execute the instruction until the $\overline{PI^{WAIT}}$ exists.

Once PK reaches state PK^{24} the computer is committed to executing the new instruction. This usually does not involve further use of PK. All $\overline{PKIR^{DIS}}$ instructions send PK back to state zero from state 24, and start a QK cycle. The $PKIR^{DIS}$ instructions send PK through a PKEI cycle from state 25 through to state 31, and start a QK cycle only if they require an operand from memory.

Some of the $PKIR^{DIS}$ instructions can make PK wait in state 25 if certain interlock conditions are not satisfied. The interlock involved is usually EB, but can also be QK, AEB or FI.

10-5.3 OPERAND COUNTER (QK). The count logic for the QK counter, like that of the PK counter, is of two types. One type is used during the memory cycle while an operand word is obtained from memory. The second type of count logic reflects the special timing required by the execution logic of certain instructions. Unlike the PK count logic, the two QK types of count logic overlap, i.e., the type that reflects the instruction requirements usually occurs in the middle of the memory cycle and, in part, at the same time as some of the memory count logic.

The memory count logic is shown on Fig. 10-34 and the instruction count logic is shown on Fig. 10-35.

The QK counter can skip and jump states, and wait in states, as in the case of the PK counter. However, there are no "decision" states in QK, and waiting can occur only in QK^{03}.

The memory count logic is primarily a function of which memory is selected. QK waits in state zero until the QI^{START} level is generated. From QK^{00} to QK^{13} the succession of states is determined entirely by the memory selected. The counter then enters the instruction section. The memory section of the QK cycle does not occur until QK^{21}. From QK^{21} to QK^{31} the count logic is again determined by the memory selected, except that $QKIR^{LOAD}$ type instructions can cause QK to skip states 22 and 23.

The only waiting state in QK, other than QK^{00}, is QK^{03} when the operand is located in the V_{FF} Memory. QK will wait in QK^{03}, if an Arithmetic Element register is selected, until the $\overline{AEB}$ level indicates that the Arithmetic Element is available for use.

A jump to $QK^{13\alpha}$ from $QK^{11\alpha}$ takes place if the operand is in the V Memory, since no parity check is required. ($QK^{13\alpha}$ is used primarily to allow time for the parity check circuits to stabilize.)

The instruction section of the QK cycle does not depend on the memory selected but rather on the particular instruction (or class of instructions) being executed. (See Fig. 10-35.) The instructions involved are: TSD, INS, SKM, ST-, LD-, FLF, FLG and COM. Each of these instructions requires QK to go through a different sequence of states in the instruction section of the QK cycle.

10-5.4 CHANGE SEQUENCE COUNTER (CSK). The change of sequence counter actually functions as two counters, as described in Chapter 6. One of these is the change of sequence counter which uses states zero through seven. This counter is usually referred to as the CSK counter. The other counter is the delay synchronization counter which uses states 8 through 11. This counter is referred to as the DSK counter even though the states are labelled $CSK^{08\alpha}$ through $CSK^{11\alpha}$. This point of view can be better understood, as illustrated in Fig. 10-36, by considering CSK as a three stage counter and CSK_4 as an interlock flip-flop. The logic controlling the counter is shown in Fig. 10-37.

When CSK_4^0, the counter can perform a change of sequence cycle when the CSI^{START} level occurs and CSK is in state $CSK^{00\alpha}$. Since the $\overline{CSI^{START}}$ level inhibits the counter only when CSK is in its $CSK^{00\alpha}$ resting state, the counter, when started, will run through to state $CSK^{07\alpha}$ and then back to state $CSK^{00\alpha}$ without interruption. Note that the essential interlock condition in CSI^{START} for a change of sequence cycle is PI_3^1.

CSK_4 is never set unless DSK is to perform a delay synchronization cycle. In this case the count logic does not permit DSK to count, once CSK_4 is set, until XWK is in state zero and PK is in state 02, 23 or 00. The count logic again inhibits the count when CSK_4 is cleared in $CSK^{11\alpha}$, since the CSK count circuit on CSK inhibits the carry from CSK_2 to CSK_3 when CSK_4^1 so that the next state of CSK is $CSK^{00\alpha}$. Hence, if CSK_4 remains set in $CSK^{11\alpha}$, DSK counts from $CSK^{11\alpha}$ to $CSK^{08\alpha}$, and if CSK_4 is cleared in $CSK^{11\alpha}$ then DSK counts from $CSK^{11\alpha}$ to $CSK^{00\alpha}$.

CSK_4 is set in any of the PK decision states when the computer is unable to decide whether to continue executing the instruction or to make a change of sequence.

In $PK^{02\alpha}$ this occurs when the computer is attempting to obtain an instruction from an Arithmetic Element flip-flop register and the AEI level is present. At least one DSK cycle is performed before the $PI^{AE\ CH\ SEQ}$ level is examined.

In $PK^{22\alpha}$, CSK_4 is set when the PI^{WAIT} level is present and the $PI^{LV\ SEQ}$ level is not. PK then enters $PK^{23\alpha}$ where DSK cycles occur. CSK_4 can not be cleared until either the $\overline{PI^{WAIT}}$ or $PI^{LV\ SEQ}$ level occurs.

Instruction which dismiss ($PKIR_H^0 \cdot PKIR^{DIS\ REQ}$) cause CSK_4 to be set during the PKEI cycle if no other sequence wants attention. This occurs at $PK^{31\alpha}$ for most of these instructions. However, it occurs at $PK^{25\alpha}$ for JPX and JNX because the $PKIR^{DIS\ REQ}$ level does not exist after $PK^{25\alpha}$ during these instructions. DSK then begins to cycle when PK reaches $PK^{00\alpha}$ and CSK_4 is then cleared when some sequence wants attention ($SS^{ATT\ REQ}$).

Note that CSK_4 is cleared and the interlock condition examined only when DSK is in state $CSK^{11\alpha}$. Note also that when another DSK cycle is to follow the level $\overline{\text{IO}}$ → CSK_4 is used to generate IO clock pulses.

10-5.5 X MEMORY WRITE COUNTER (XWK). The START → XWK level causes the XWK counter to start counting by setting XWK_1, as shown in Fig. 10-38. The counter will then continue to count through to XWK^{07} and then back to XWK^{00}.

While XWK is counting the X Memory parity compute circuit first stablizes and then the X Memory write current is turned on and off via XW.

10-5.6 F MEMORY COUNTER (FK). The FK counter logic is shown on Fig. 10-39. The FK counter will start counting, when the START → FK level occurs, if it is in its FK^{00} resting state and the FK8 flip-flop is cleared.

If the counter is not executing a FF type instruction (SPG or FLG), the counter will cause one complete F Memory read-write cycle while it is counting through to state $FK^{02\alpha}$ and then back to $FK^{00\alpha}$.

When either a SPG or FLG instruction is performed, FK counts from $FK^{00\alpha}$ through to $FK^{07\alpha}$ and through one extra state, in which $FK8^{1} \cdot FK^{00\alpha}$. In the process FK executes four complete read-write cycles.

As shown in Fig. 10-40, FK is only a 9 state counter even though it has four stages. The special FK8 flip-flop of the counter is set when the computer reaches state $FK^{07\alpha}$, and then is cleared on the next α pulse while the other stages remain cleared.

10-5.7 ALARM DELAY COUNTER (ADK). See Sec. 10-2.5.8 for a discussion of this counter.

10-5.8 ARITHMETIC ELEMENT COUNTER (AK). AK differs from all the other control counters in that it is actually a shift register and the flip-flops of the counter are used directly to generate the AK control time levels. Thus AK is in state $AK^{03\alpha}$ when $AK^{1}_{\alpha.3}$. The logic for the counter is shown in Fig. 10-41.

AK is placed in state $AK^{00\alpha}$, before it is started, whenever an operation code is placed in AKIR. This also occurs whenever a preset level or a Synch System AE Stop condition is generated, or whenever an undefined AKIR operation code occurs ($AKIR^{AOP}$).

The counter is also cleared at the end of the various AK instruction cycles. For ADD, SUB and MUL this occurs at $AK^{09\alpha}$, for DIV at $AK^{11\alpha}$, and for DSA at $AK^{03\alpha}$. For TLY it occurs in $AK^{02\alpha}$ when ASK has reached state zero ($ASK^{0}_{7} \cdot ASK^{0}_{6}$). For NOA, NOB and NAB it occurs in $AK^{04\alpha}$ when ASK reaches state zero in the case where the number being normalized is zero. If the number being normalized is not zero or if a SH type instruction (SCA, SCB, SAB, CYA, CYB or CAB) is being performed, a different clearing logic is used. In all these NOR and SH type instruction the clear pulse occurs in $AK^{04\alpha}$, but in the case of the NOR type instruction only when <u>all</u> the numbers being normalized are actually normalized and in the case of the SH type only when all the counts in D are finished.

AK starts counting from $AK^{00\alpha}$ when the START → AK level occurs. It then continues to count until either a skip or waiting state is reached or the counter returns to $AK^{00\alpha}$ and there is no Synch System AE Stop condition. During NOR and SH type instruction AK simply counts through to $AK^{04\alpha}$ where it remains until AK is cleared as described above. During DSA, AK counts through to $AK^{03\alpha}$ (and then returns to $AK^{00\alpha}$). In the case of ADD, SUB or MUL, AK counts until it reaches either $AK^{03\alpha}$ or $AK^{09\alpha}$. These instructions return AK to $AK^{00\alpha}$ from $AK^{09\alpha}$, but the $AK^{03\alpha}$ situation varies and will be covered below.

DIV and TLY are more complex. Both of them stop the AK count pulses when AK is in states $AK^{02\alpha}$, $AK^{03\alpha}$, $AK^{09\alpha}$ or $AK^{11\alpha}$, or when $ASK_2^1 \cdot ASK_1^0$, or when AK is in state 08α and ASK_7^0. Note that AK is not cleared until AK reaches $AK^{11\alpha}$ during a DIV, but that during a TLY AK is cleared at $AK^{02\alpha}$. Hence in the latter case none of the logic in AK states later than $AK^{02\alpha}$ apply.

As will be seen in Chapter 16, AK can jump forwards or backwards in ADD, SUB, MUL and DIV. This state jumping is covered in the $\lfloor$PRESET$\rightarrow$ AK logic. Note that this preset logic never places AK in state $AK^{00\alpha}$.

In the case of ADD and SUB, AK jumps from $AK^{03\alpha}$ to either state $AK^{05\alpha}$ or $AK^{06\alpha}$ depending on the length of the subword in the Arithmetic Element. The amount of time allowed for carries to propagate in the carry circuits is controlled in this manner.

During a MUL, AK waits in $AK^{03\alpha}$ until ASK_7^0, while the multiplication is performed, and then jumps to $AK^{05\alpha}$ or $AK^{06\alpha}$ to do the final carry.

During a DIV, AK jumps ahead to $AK^{05\alpha}$ or $AK^{06\alpha}$ in order to enter the subcycle in which the divide steps are performed. This subcycle extends from $AK^{05\alpha}$ or $AK^{06\alpha}$ through to $AK^{09\alpha}$. After the subcycle is first entered, AK jumps back to $AK^{05\alpha}$ or $AK^{06\alpha}$ from $AK^{09\alpha}$ until $ASK_7^0 \cdot ASK_2^1 \cdot ASK_1^0$ in $AK^{08\alpha}$. AK then jumps ahead out of the subcycle to $AK^{10\alpha}$ from $AK^{08\alpha}$ (and continues on to $AK^{11\alpha}$ before returning to $AK^{00\alpha}$).

10-5.9 ARITHMETIC ELEMENT STEP COUNTER (ASK). ASK is used to control the number of times a subcycle in AK is repeated during most of the $QKIR^{AESK}$ and AOP instructions. The ASK counter logic is shown in Fig. 10-42.

ASK is cleared when the $\lfloor$START$\rightarrow$ AK level occurs. It is then preset to some negative number at $AK^{01\alpha}$. This value is dependent on the length of the longest active subword in the Arithmetic Element and on whether the instruction uses single ($AKIR^{N}$) or double ($AKIR^{2N}$) length subwords.

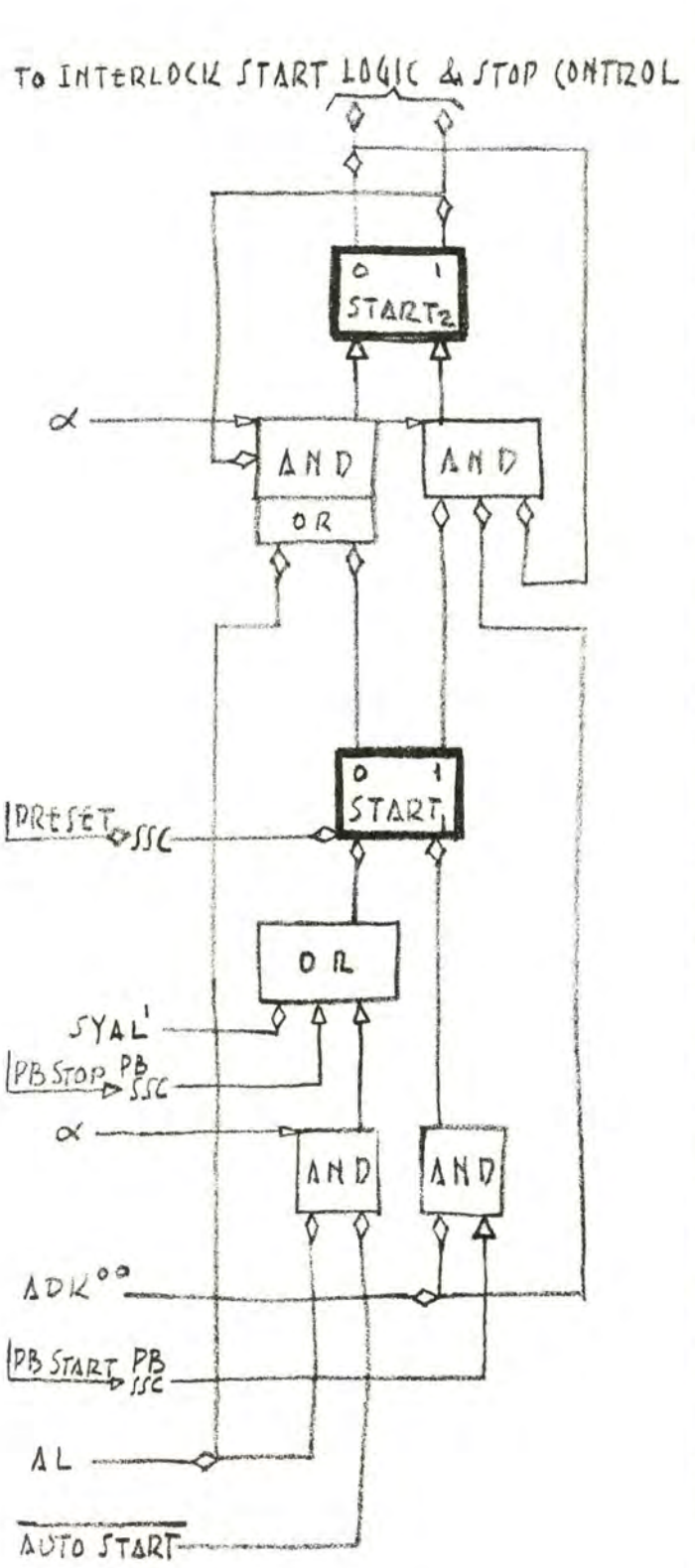
An ASK count pulse is generated whenever AK performs a subcycle. These subcycles can be only one AK state long so that AK simply waits until the ASK count is "complete". Usually this completion of the ASK count occurs when ASK goes positive, i.e., when ASK_7 becomes a zero. During some instructions, however, ASK can end with a positive number as large as two.

During NOR and SH type instructions, ASK counts while AK waits in $AK^{04\alpha}$. Note that the content of ASK does not influence the number of AK subcycle repetitions during SH type instructions.

During MUL, ASK counts once in $AK^{02\alpha}$, and then counts each time AK goes through the subcycle in $AK^{03\alpha}$. Thus, the subcycle is repeated one less time than the length of the subword.

During TLY, ASK counts each time AK goes through the subcycle in $AK^{02\alpha}$.

During DIV, ASK counts each time AK goes through the subcycle which passes through $AK^{07\alpha}$. Note that this subcycle actually begins in $AK^{03\alpha}$ or $AK^{04\alpha}$ and goes through to $AK^{08\alpha}$.



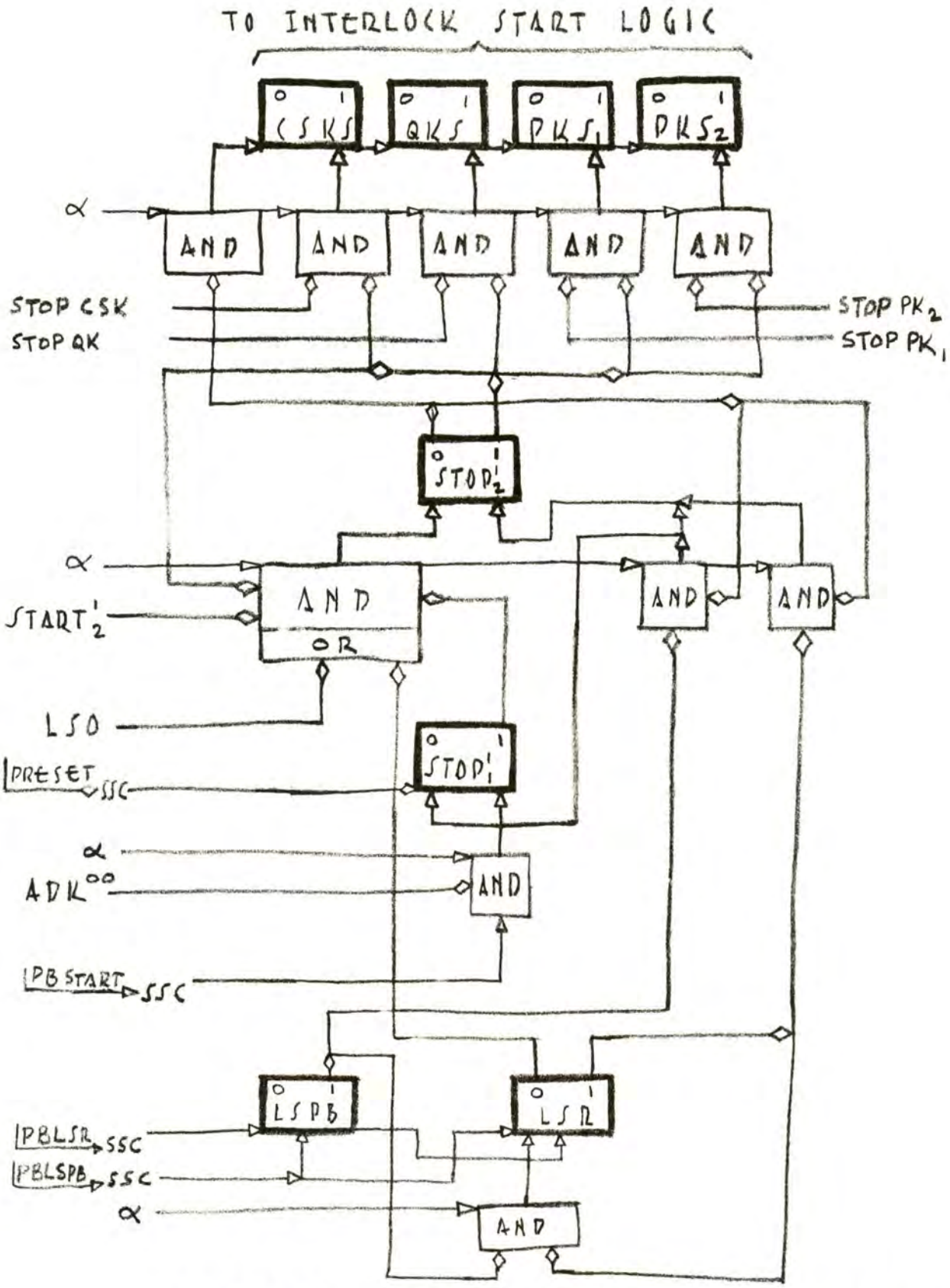
START CONTROL

Fig 10-1

HM 6-3-60

		START 1	START 2
AL · AUTO START	PB START PB SSC	1	0
	α	1	1
	↓	↓	↓
	AL · AUTO START	0	0
	α	0	0
	↓	↓	↓
AL · AUTO START	PB START PB SSC	1	0
	α	1	1
	↓	↓	↓
	AL · AUTO START	1	0
	↓	↓	↓
	ADK°° · α	1	1
	↓	↓	↓
PB STOP → SSC PB	PB START PB SSC	1	0
	α	1	1
	↓	↓	↓
	PB STOP PB SSC	0	1
	α	0	0
	↓	↓	↓

FLIP-FLOP SETTINGS



STOP CONTROL HM 6-3-60

Fig 10-2

SELECTION ALARMS

	1 → FF				0 → FF		
	RD PULSE	TIME LEVEL	INSTRUCTIONS	OTHERS	RD PULSE	TIME LEVEL	SELECTION ALARM
PSAL	α	$PK^{09\alpha}$.		$\cdot \overline{PKM^{LEGAL}} \cdot PI_2^0$	α	CA_2^1 ADK^{10}	$PSAL_{SUP}$ $\overline{PSAL_{SUP}}$
PD	α	$PK^{09\alpha}$		$\cdot \overline{PKM^{LEGAL}} \cdot PI_2^0$	α	AUTOMATICALLY CLEARED	
QSAL	α	$QK^{09\alpha}$. $PK^{09\alpha}$.		$\cdot \overline{QKM^{LEGAL}}$ $\cdot \overline{PKM^{LEGAL}} \cdot PI_5^1$	α	CA_2^1 ADK^{10}	$QSAL_{SUP}$ $\cdot \overline{QSAL_{SUP}}$
QD	α	$QK^{09\alpha}$. $PK^{09\alpha}$.		$\cdot \overline{QKM^{LEGAL}}$	α	AUTOMATICALLY CLEARED	
IOSAL	α	$PK^{24\alpha}$.	$PKIR^{IOS}$	$\cdot \left(N_{2.6-2.4}^{011} + N_{2.6-2.4}^{110}\right) \cdot IOCM^{MAINT}$	α	CA_2^1 ADK^{10}	$\cdot IOSAL_{SUP}$ $\cdot \overline{IOSAL_{SUP}}$
ID	α	$PK^{24\alpha}$.	$PKIR^{IOS}$	$\cdot \left(N_{2.6-2.4}^{011} + N_{2.6-2.4}^{110}\right) \cdot IOCM^{MAINT}$		AUTOMATICALLY CLEARED	
MISAL	α			$IOCM^{MISIND}$		CA_2^1 ADK^{10}	$\cdot MISAL_{SUP}$ $\cdot \overline{MISAL_{SUP}}$
MID	α			$IOCM^{MISIND}$		AUTOMATICALLY CLEARED	
OCSAL	α	$PK^{15\alpha}$.	$\overline{PKIR^{DEF}}$ $AKIR^{OCSAL}$	$\cdot PI_2^0$		CA_2^1 ADK^{10}	$\cdot OCSAL_{SUP}$ $\cdot \overline{OCSAL_{SUP}}$
OD	α	$PK^{15\alpha}$.	$\overline{PKIR^{DEF}}$ $AKIR^{OCSAL}$			AUTOMATICALLY CLEARED	

MEMORY SELECTION IN-OUT AND OPERATION CODE ALARMS

FIG 10-3

HM6-B-60

PARITY ALARMS

	1 → FF					0 → FF		
	RD PULSE	PARITY	LEVELS	INSTRUCTIONS	OTHERS	RD PULSE	TIME LEVEL	PARITY ALARM
MPAL	α	MP_{38}^{EV} · $\llcorner 1 \lrcorner\diamond$ MPAL			· QKM^{LEGAL} · $\overline{QKM^{V}}$	α	CA_2^1 · MPAL SUP ADK^{10} · $\overline{\text{MPAL}}$ SUP	
MD	α	MP_{38}^{EV} · $\llcorner 1 \lrcorner\diamond$ MPAL			· QKM^{LEGAL} · $\overline{QKM^{V}}$	α	AUTO CLEARED	
NPAL	α	NP_{38}^{EV} · $PK^{13\alpha}$			· PKM^{LEGAL} · $\overline{PKM^{V}}$	α	CA_2^1 · NPAL SUP ADK^{10} · $\overline{\text{NPAL}}$ SUP	
ND	α	NP_{38}^{EV} ·			· PKM^{LEGAL} · $\overline{PKM^{V}}$	α	AUTO CLEARED	
FPAL	α				$\llcorner 1 \lrcorner\diamond$ FPAL	α	CA_2^1 · FPAL SUP ADK^{10} · $\overline{\text{FPAL}}$ SUP	
FD	α				$\llcorner 1 \lrcorner\diamond$ FPAL	α	AUTO CLEARED	
XPAL	α	XP_{19}^{EV} · $[PK^{15\alpha} + CSK^{04\alpha}]$				α	CA_2^1 · XPAL SUP ADK^{10} · $\overline{\text{XPAL}}$ SUP	
XD	α	XP_{19}^{EV} · $[PK^{15\alpha} + CSK^{04\alpha}]$				α		

$\llcorner 1 \lrcorner\diamond MPAL = [(QK^{13\alpha} \cdot QKIR^{ST}) + QK^{14\alpha} \cdot (QKIR^{LOAD} + QKIR^{COM}) + (QK^{13\alpha} \cdot \overline{QKIR^{INS}}) + QK^{13\alpha} \cdot (QKIR^{2NS} + QKIR^{FL})]$

$QKM^{LEGAL} = [QKM^{V} + (QKM^{U} \cdot UMOFF^{\circ}) + (QKM^{T} \cdot TMOFF^{\circ}) + (QKM^{S} \cdot SMOFF^{\circ})]$

$PKM^{LEGAL} = [PKM^{V} + (PKM^{U} \cdot UMOFF^{\circ}) + (PKM^{T} \cdot TMOFF^{\circ}) + (PKM^{S} \cdot SMOFF^{\circ})]$

$\llcorner 1 \lrcorner\diamond FPAL = FP_{10}^{EV} \cdot \overline{PKIR_{CF}^{00}} \cdot (FKB^{1} + \overline{FK^{00\alpha}} \cdot FK_{\alpha,1}^{\circ})$

FIG 10-4

PARITY ALARMS AND DRIVERS

HM 6-8-60

PULSE	1 → FF		0 → FF	
	RD PULSE	OTHER	RD PULSE	OTHER
TSAL		MAS$_{T,J}$ TRANSITION		PRINT → CE
SYAL		SYNCH STOP		CLEAR ALARM UNSUPPRESSED → SSC
SD		SYNCH STOP		AUTOMATICALLY CLEARED
MOUSETRAP	∝	1 → MOUSETRAP	∝	ADK^{10}

MISCELLANEOUS ALARMS

FIG 10-5

MA 12-28-60

AL - ALARM CONDITION

ALARM CONDITION	
ALARM #	
MPAL'	• MVAL$_{SUP}$
NPAL'	• NPAL$_{SUP}$
FPAL'	• FPAL$_{SUP}$
XPAL'	• XPAL$_{SUP}$
PSAL'	• PKSAL$_{SUP}$
QSAL'	• QKSAL$_{SUP}$
MISAL'	• MISAL$_{SUP}$
IOSAL'	• IOSAL$_{SUP}$
OCSAL'	• OCSAL$_{SUP}$
MOUSETRAP'	•

AL LEVEL LOGIC

FIG 10-6

HM 6-8-60

CHIME ON ALARMS

CHIME ON ALARMS$_{SUP}$			CHIME ON ALARMS$_{USUP}$	
CHIME	ALARM	INDICATOR DRIVER	CHIME	TIME LEVEL
(SUPPRESS CHIME ON SUPPRESSED ALARMS)	$\cdot MPAL_{SUP}$	$\cdot MD'$	(SUPPRESS CHIME ON UNSUPPRESSED ALARMS)	$\cdot ADK^{01}$
(ditto)	$\cdot NPAL_{SUP}$	$\cdot ND'$	(ditto)	$\cdot SYD'$
(ditto)	$\cdot FPAL_{SUP}$	$\cdot FD'$		
(ditto)	$\cdot XPAL_{SUP}$	$\cdot XD'$		
(ditto)	$\cdot PSAL_{SUP}$	$\cdot PD'$		
(ditto)	$\cdot QSAL_{SUP}$	$\cdot QD'$		
(ditto)	$\cdot MISAL_{SUP}$	$\cdot MID'$		
(ditto)	$\cdot IOSAL_{SUP}$	$\cdot ID'$		
(ditto)	$\cdot OCSAL_{SUP}$	$\cdot OD'$		

CHIME LOGIC

FIG 10-7

HM 6-8-60

ALARM DELAY COUNTER (ADK) FUNCTION

TIMING STATE		ALARM FF COMBINATIONS FF	FF SUBSCRIPT 2	FF SUBSCRIPT 1	LOGICAL CONDITIONS	FUNCTIONAL DESCRIPTION
ASSYNCHRONOUS	00α	ALD	0	0	AL ⊃ L1→ ALD1	|PB START→ SSC ⊃ L1→ STOP1 , L1→ START1 START2° · START1' ⊃ L1→ START2
		ADK	0	0		
		ALD	0	1	ALD1' ⊃ L1→ ADK1	
		ADK	0	0		
SYNCHRONOUS	01α	ALD	0	1		ADK2° · ADK1' · $\overline{\text{SUPPRESS CHIME ON UNSUPPRESSED ALARM}}$ ⊃ CHIME ON UNSUPPRESSED ALARMS
		ADK	0	1		
		TIME DELAY - DURATION FOR OPERATION OF CHIMES AND FOR TX-2 TO STABILIZE				
		ALD	0	0	ADK2° · ADK1' · ALD1° ⊃ L1→ ALD2 , L1→ ADK2	
		ADK	0	1		
	11α	ALD	1	0		ALD2' · PASOFA ⊃ L1→ SYNCH OF SEQUENCE 00 (RAISE FLAG) ALD2' · AUTO-START · PASOFA ⊃ |PRESET→ CE
		ADK	1	1		
		TIME DELAY				
		ALD	0	0	ADK2' · ALD2° · ADK1' · [CA1' + AutoStart] ⊃ L0→ ADK1	|PB CLEAR UNSUPPRESSED ALARMS→ SSC ⊃ L1→ CA1 WAITING STATE
		ADK	1	1		
	10α	ALD	0	0	L0→ ADK2	L0→ CA1 ⊃ CLEAR ALL $\overline{\text{SUPPRESSED ALARMS}}$
		ADK	1	0		
	00α	ALD	0	0		RETURN TO STARTING STATE FOR NEXT OPERATION
		ADK	0	0		

ALARM DELAY COUNTER

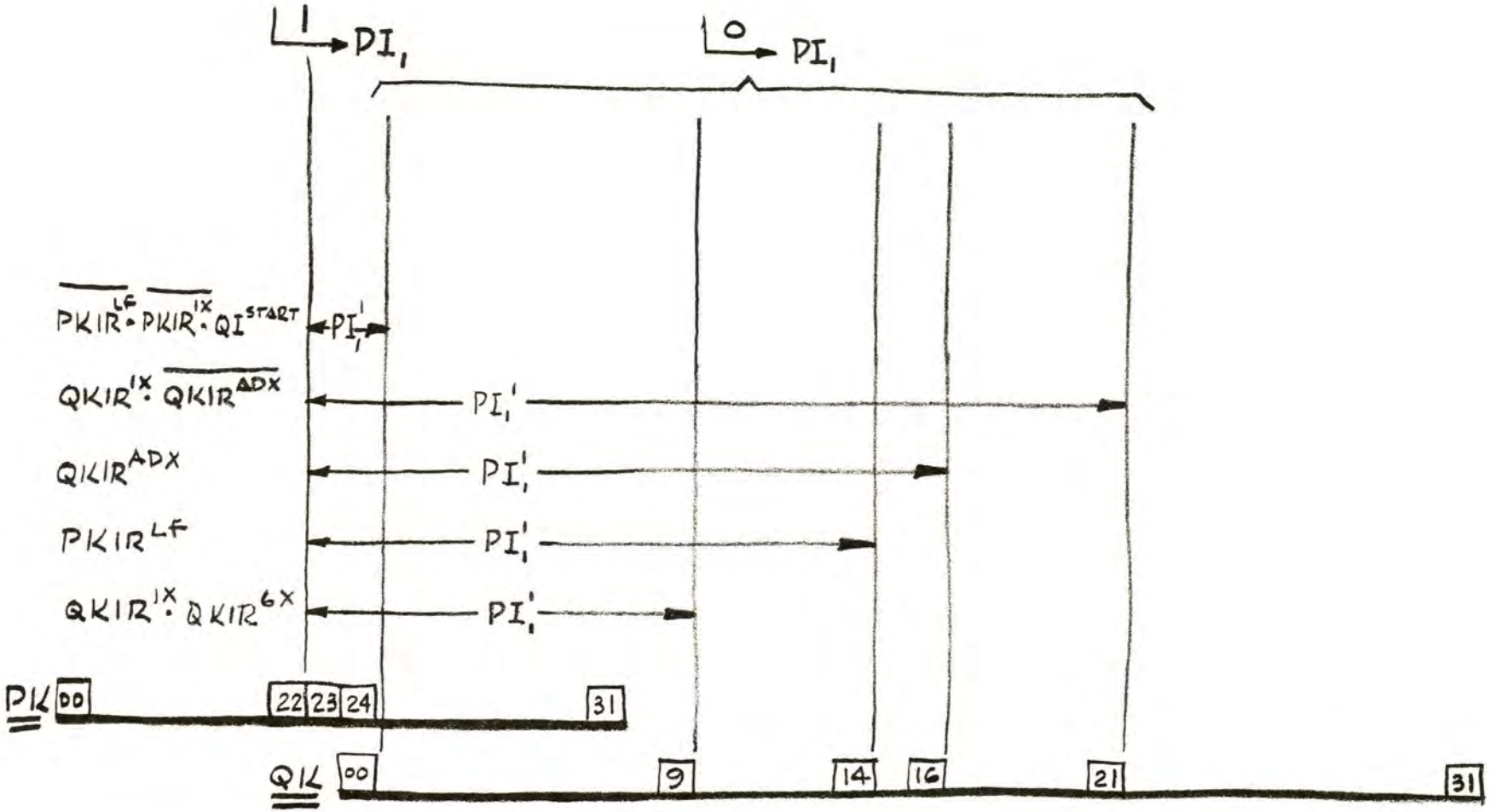
FIG 10-8
HM 4-28-60

PI_1 - INSTRUCTION$_1$ INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
$\xrightarrow{1} PI_1$	α	$\overline{\text{PRESET}}$ CE	$PK^{22\alpha}$	$\cdot PKIR^{QK}$	$\cdot \overline{PI^{WAIT}} \cdot PI_2^{\circ}$
			$CSK^{11\alpha}$	$\cdot PKIR^{QK}$	$\cdot PK^{23\alpha} \cdot \overline{PI^{WAIT}}$
$\xrightarrow{0} PI_1$	α	$\overline{\text{PRESET}}$ CE	$QK^{00\alpha}$	$\cdot \overline{PKIR_{op}^{IX}} \cdot \overline{PKIR^{LF}}$	$\cdot QI^{START}$
			$QK^{09\alpha}$	$\cdot QKIR_{op}^{IX} \cdot QKIR_{op}^{X6}$	
			$QK^{14\alpha}$	$\cdot QKIR_{op}^{IX} \cdot \overline{QKIR^{ADX}}$	
			$QK^{16\alpha}$	$\cdot QKIR^{ADX}$	
			$QK^{21\alpha}$	$\cdot PKIR^{LF}$	
$\xrightarrow{0} PI_1$		PRESET CE			

FIG 10-9

MM 5-31-60



PI1' INTERLOCK DURATIONS

FIG 10-10

HM 5-20-60

PI_2 - INSTRUCTION$_2$ INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
1 → PI_2	α	$\overline{\text{PRESET}}$ CE	$PK^{13\alpha}$ · $PKIR^{DEF}$		· $N^1_{2,9}$ · PI^0_2
0 → PI_2	α	$\overline{\text{PRESET}}$ CE	$PK^{13\alpha}$		· PI^1_2 · PI^0_5
			$(SK^{07\alpha}$		· $SS^{CH\ REQ}$ (NO RELEVANCE)
			$(SK^{11\alpha}$		· $PK^{02\alpha}$ · $PI^{A6\ CH.\ SEQ}$
0 → PI_2		PRESET CE			

Fig 10-11

HM 5-31-60

PI$_3$ - INSTRUCTION$_3$ INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC: TIME LEVEL	PULSE GATE LOGIC: INSTRUCTION	PULSE GATE LOGIC: OTHERS
$\xrightarrow{1}$ PI$_3$	α	$\overline{\text{PRESET}}$ $\rightarrow$ CE	$CSK^{07\alpha}$		$\cdot SS^{CH\ REQ}$
			$CSK^{11\alpha}$		$\cdot PK^{00\alpha} \cdot SS^{AT\ REQ} \cdot KD^{00}$
			$CSK^{11\alpha}$		$\cdot PK^{00\alpha} \cdot SS^{AT\ REQ} \cdot \overline{K^{EQ\ JC}}$
			$CSK^{11\alpha}$		$\cdot PK^{02\alpha} \cdot PI^{AE\ CH\ SEQ}$
			$CSK^{11\alpha}$		$\cdot PK^{23\alpha} \cdot PI^{LEAVE\ SEQ}$
			$PK^{22\alpha}$		$\cdot PI_2^0 \cdot PI^{LEAVE\ SEQ}$
			$PK^{24\alpha} \cdot$	$\overline{PKIR^{IOS}}$	$\cdot PI^{CH\ SEQ}$
			$PK^{31\alpha} \cdot$	$\overline{PKIR^{3X}}$	$\cdot PI^{CH\ SEQ}$
$\xrightarrow{0}$ PI$_3$	α	$\overline{\text{PRESET}}$ $\rightarrow$ CE	$CSK^{04\alpha}$		
$\xrightarrow{0}$ PI$_3$		PRESET $\rightarrow$ CE			

FIG 10-12

HM 5-31-60

PI_4 - INSTRUCTION$_4$ INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
$\rightarrow 1 \rightarrow PI_4$	$\propto$	$\overline{\text{PRESET}} \diamond CE$	$PL^{24\alpha} \cdot PKIR_h^1$		
$\rightarrow 0 \rightarrow PI_4$	$\propto$	$\overline{\text{PRESET}} \diamond CE$	$PK^{24\alpha} \cdot PKIR_h^0$		

FIG 10-13

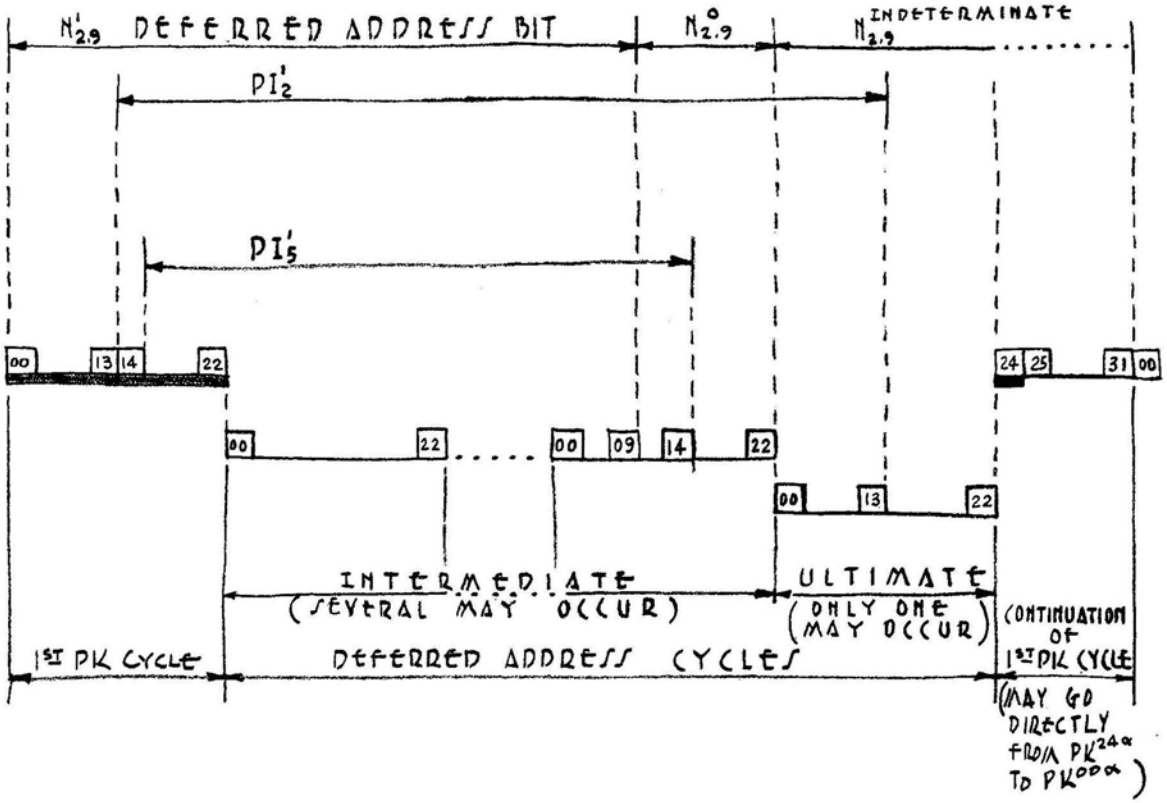
HM 5-31-60

PI_5 - INSTRUCTION$_5$ INTERLOCK

PULSE	α	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
$\xrightarrow{1} PI_5$	α	$\overline{\text{PRESET}}$ CE	$PK^{14\alpha}$	$\cdot PKIR^{DEF}$	$\cdot N^{1}_{2.9} \cdot PI^{1}_{2}$
$\xrightarrow{0} PI_5$	α	$\overline{\text{PRESET}}$ CE	$PK^{14\alpha}$ $CSK^{07\alpha}$ $CSK^{11\alpha}$		$\cdot N^{0}_{2.9} \cdot PI^{1}_{5}$ $\cdot$ SS CH REQ (NO RELEVANCE) $\cdot PK^{02\alpha} \cdot PI^{AE}$ CH SEQ
$\xrightarrow{0} PI_5$		$\overline{\text{PRESET}}$ CE			

FIG 10-14

TM 5-31-60



PI$_2$ & PI$_5$ INTERLOCK DURATIONS VS PK CYCLE

FIG 10-15

HM 5-20-60

ARITHMETIC ELEMENT INTERLOCK LEVELS

LEVEL	LEVEL LOGIC		
	TIME LEVEL	INSTRUCTIONS	OTHERS
AEB	$AK^{\circ}_{\alpha.0}$		
AEI	$\overline{QK^{00\alpha}}$	$\cdot QKIR^{AESK}$	AEP' $\cdot QK^{\circ}_{5\alpha}$

$AK^{\circ}_{\alpha.0} = \overline{AK^{00\alpha}}$

FIG 10-16

HM 6-2-60

AEP - ARITHMETIC ELEMENT PREDICT INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
$\rightarrow^{1}$ AEP	$\propto$	$\overline{\text{PRESET CE}}$	$PK^{26\alpha} \cdot PKIR^{OPR\,AG}$ $QK^{14\alpha} \cdot QKIR^{AESK}$		
$\rightarrow^{0}$ AEP	$\propto$	$\overline{\text{PRESET CE}}$		$\rightarrow^{0}$ AEP	
$\rightarrow^{0}$ AEP		PRESET CE			

$$\rightarrow^{0}_{AEP} = AK'_{\alpha.4} \cdot AKIR^{NOR} \cdot \left(\bar{A}^{0}_{1} \cdot \bar{A}^{1}_{1} + I\right) \cdot \left(\bar{A}^{0}_{2} \cdot \bar{A}^{1}_{2} + II\right) \cdot \left(\bar{A}^{0}_{3} \cdot \bar{A}^{1}_{3} + III\right) \cdot \left(\bar{A}^{0}_{4} \cdot \bar{A}^{1}_{4} + \overline{IV}\right)$$

$$+ AK'_{\alpha.4} \cdot AKIR^{SH} \cdot \left(LAD + \bar{J}\right) \cdot \left(LAD_2 + \overline{II}\right) \cdot \left(LAD_3 + \overline{III}\right) \cdot \left(LAD_4 + \overline{IV}\right)$$

$$+ AK'_{\alpha.1} \cdot \left(AKIR^{ADD} + AKIR^{DSA}\right)$$

$$+ AK'_{\alpha.9} \left(ASK'_{1} \cdot ASK^{0}_{7}\right) \cdot AKIR^{DIV}$$

$$+ \left[\left(AK'_{\alpha.2} \cdot AKIR^{TLY}\right) + \left(AK'_{\alpha.4} \cdot AKIR^{NOR}\right)\right] \cdot \left[ASK'_{7} \cdot ASK'_{6} \cdot ASK'_{5} \cdot ASK'_{4} \cdot ASK^{0}_{3} \cdot ASK'_{2}\right]$$

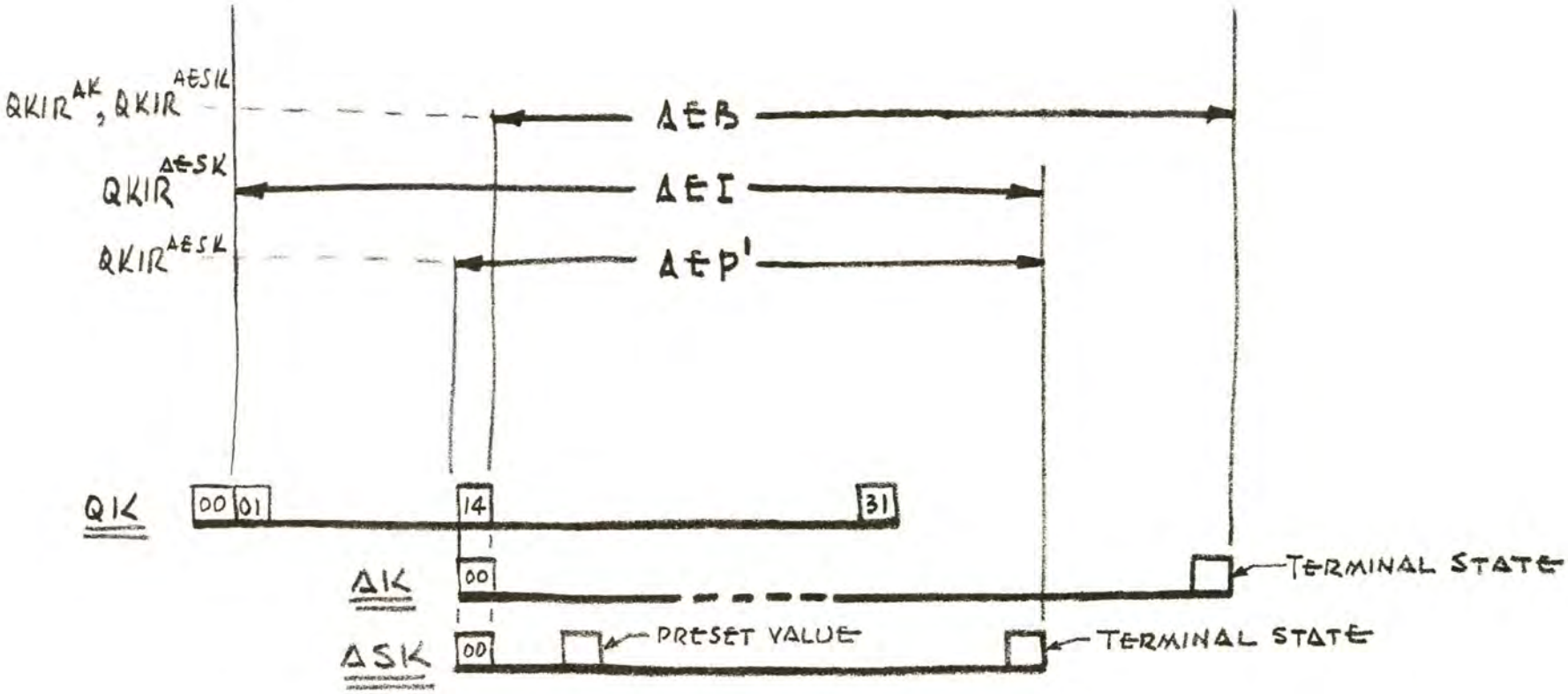
$$+ AK'_{\alpha.3} \cdot ASK^{0}_{7} \cdot AKIR^{MUL}$$

$$+ AKIR^{OCSAL}$$

AND WHERE $LAD_i = D_{i4} \cdot D_{i5} \cdot D_{i6} \cdot D_{i7} \cdot D_{i8} = D_{ij}\ (j = 4\text{-}8)$

FIG 10-17

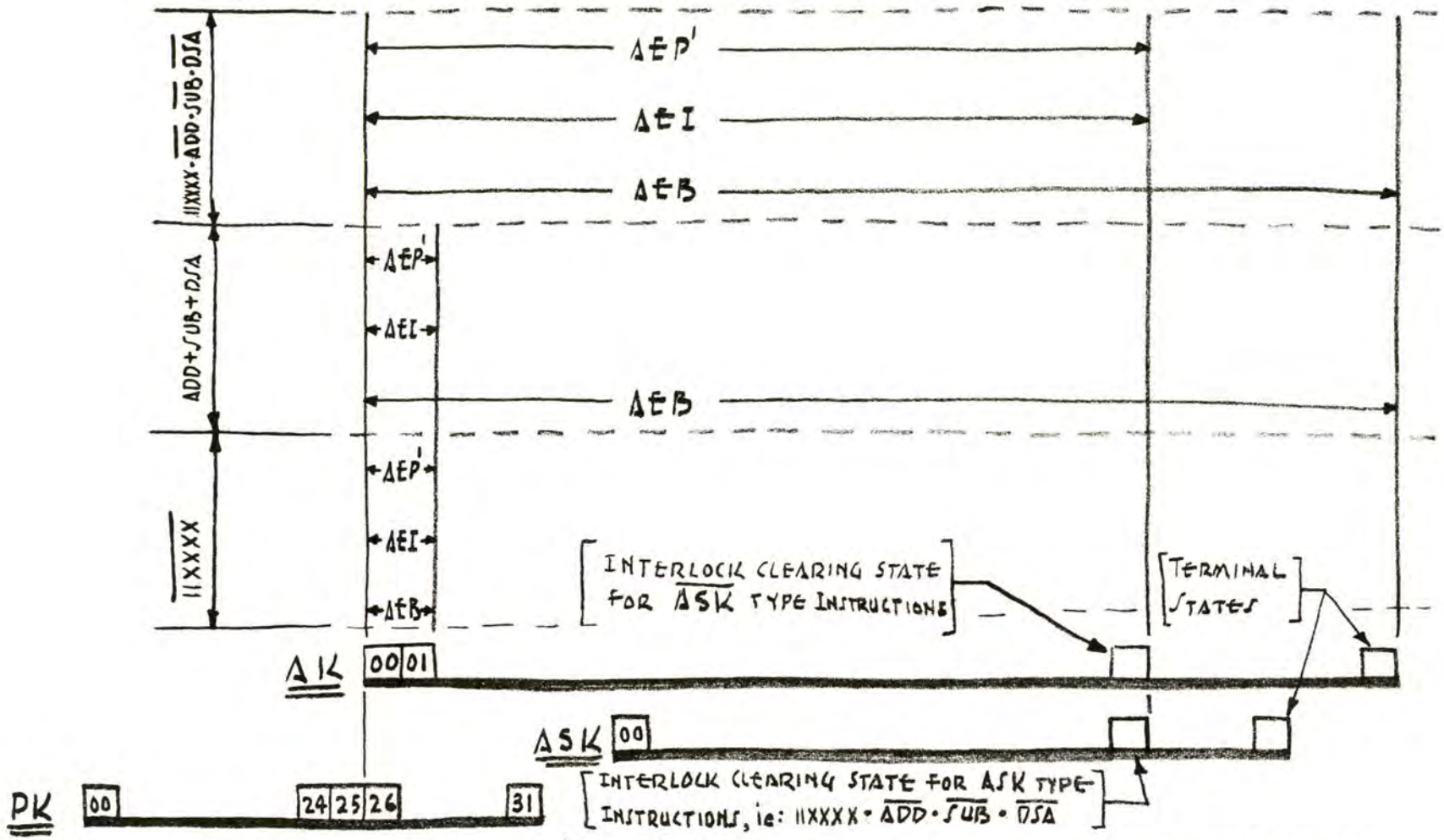
HA 5-31-60



AEB, AEI, & AEP' INTERLOCK DURATIONS
ON QKIR^AK INSTRUCTIONS

FIG 10-18

HM 6-1-60



ΔEB, ΔEI & ΔEP' INTERLOCK DURATIONS ON PKIR OPRAT INSTRUCTIONS

FIG 10-19 TM 6-2-60

EB - EXCHANGE ELEMENT BUSY INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
11 → EB	α	PRESET CE	$QK^{00\alpha}$		$\cdot QI^{START}$
10 → EB	α	PRESET CE	$QK^{21\alpha} \cdot QKIR^{STORE}$ $QK^{23\alpha} \cdot \overline{QKIR^{SPG}}$ $FK^{06\alpha} \cdot QKIR^{SPG}$		$\cdot \overline{QKM^{VF}}$
10 → EB		PRESET CE			

FIG 10-20

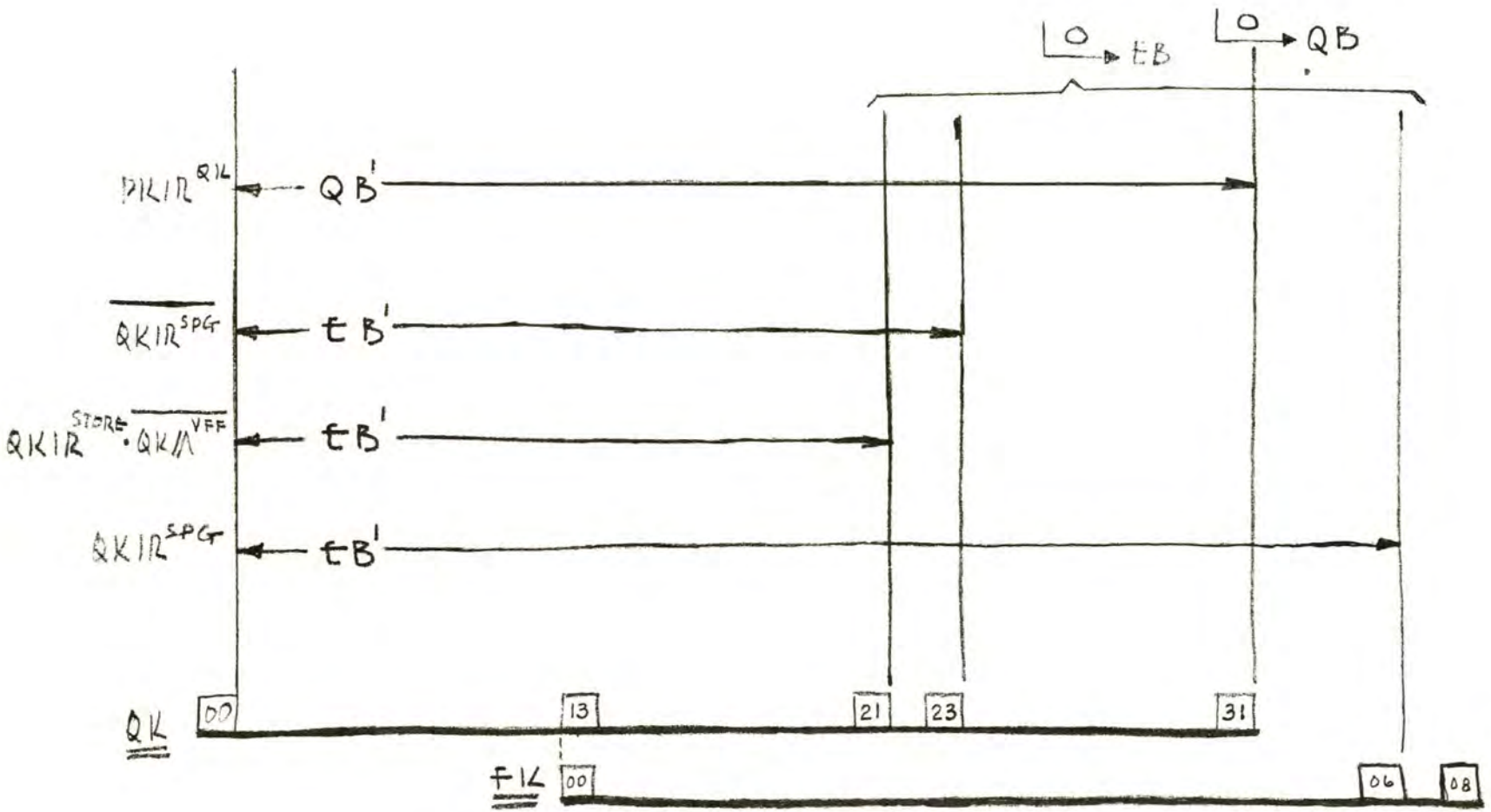
HM 5-31-60

QB INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIMELEVEL	INSTRUCTION	OTHER
1 → QB	α	$\overline{\text{PRESET}}$ CE	$QK^{00\alpha}$		QI^{START}
0 → QB	α	$\overline{\text{PRESET}}$ CE	$QK^{31\alpha}$		
0 → QB		PRESET CE			PRESET CE

FIG 10-21

HM 5-31-60



QB' & tB' INTERLOCK DURATIONS

FIG 10-22 HM 5-20-60

FI - CONFIGURATION INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
1 → FI	α	$\overline{\text{PRESET}}$ ◇ CE	$FK^{2\alpha}$	$\cdot PKIR^{JA}$	
			$FK^{2\alpha}$	$\cdot PKIR^{SF} \cdot \overline{PKIR^{FF}}$	
			$FK^{7\alpha}$	$\cdot PKIR^{SF}$	
0 → FI	α	$\overline{\text{PRESET}}$ ◇ CE	$PK^{13\alpha}$	$\cdot PKIR^{SF}$	$\cdot PI_5^0 \cdot PI_2^1$
			$PK^{13\alpha}$	$\cdot PKIR^{SF}$	$\cdot PI_5^0 \cdot \quad \cdot N_{2.9}^0$
			$PK^{13\alpha}$	$\cdot PKIR^{JA}$	$\cdot PI_5^0 \cdot PI_2^1$
			$PK^{13\alpha}$	$\cdot PKIR^{JA}$	$\cdot PI_5^0 \cdot \quad \cdot N_{2.9}^0$
		PRESET ◇ CE			

Fig 10-23

HM 5-31-60

XB - INDEX REGISTER BUSY INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHERS
1 → XB	α	PRESET CE	$QK^{13\alpha}$ · $QKIR^{AUX}$ $QK^{13\alpha}$ · $QKIR^{LD}$ · $QKIR^{X}$ $CSK^{01\alpha}$ $CSK^{04\alpha}$ $DK^{12\alpha}$		
0 → XB	α	PRESET CE	$XWK^{02\alpha}$		
0 → XB		PRESET CE			

FIG 10-24

#M 5-31-60

XW - INTERLOCK

PULSE	RD PULSE	PRESET	PULSE GATE LOGIC		
			TIME LEVEL	INSTRUCTION	OTHER
1 → XW	α	$\overline{\text{PRESET}}$ CE	XWK$^{02\alpha}$		
0 → XW	α	$\overline{\text{PRESET}}$ CE	XWK$^{06\alpha}$		
0 → XW		PRESET CE			

FIG 10-25

HM 5-31-60

LEVEL	LEVEL LOGIC		
	TIME LEVEL	INSTRUCTION	OTHERS
PI^{START_1}			$\left[QB^{\circ} + NO^{\circ} \cdot \overline{(P^{S} \cdot Q^{S} + P^{T} \cdot Q^{T} + P^{U} \cdot Q^{U} + P^{V} Q^{V})}\right] (START_2^{1} \cdot PKS_1^{\circ} \cdot PI_1^{\circ} \cdot PI_3^{\circ} \cdot XB^{\circ} \cdot (CSK_4^{\circ} \cdot \overline{AL})$
PI^{START_2}			$PKS_2^{\circ} \cdot XB^{\circ} \cdot QB^{\circ} \cdot START_2^{1}$
QI^{START}			$PI_1^{1} \cdot FI^{1} \cdot QKS^{\circ} \cdot START_2^{1}$
CSI^{START}	$PK^{\circ\circ\alpha}$		$PI_1^{\circ} \cdot PI_3^{1} \cdot XW^{\circ} \cdot XB^{\circ} \cdot EB^{\circ} \cdot CSKS^{\circ} \cdot START_2^{1}$

INTERLOCK START LEVELS

FIG 10-26

HM 6-1-60

LEVEL	INTERLOCK LEVELS		
	TIME LEVEL	INSTRUCTION	OTHERS
$PI^{LEAVE\ SEQ}$		$PKIR^{TSD}$	$SS^{ATT\ REQ} \cdot IOCM^{BB}$
		$PKIR^{TSD} \cdot QKIR^{TSD}$	$SS^{ATT\ REQ} \cdot QB'$
		$PKIR^{AG}$	$SS^{CH\ REQ} \cdot AEI \cdot PI^{\circ}_{4}$
		$PKIR^{QK}$	$SS^{CH\ REQ} \cdot AEI \cdot PI^{\circ}_{4} \cdot XA^{At}$
$PI^{AE\ CH\ SEQ}$			$SS^{CH\ REQ} \cdot AEL \cdot PI^{\circ}_{4}$
$PI^{CH\ SEQ}$		$PKIR^{\circ}_{h}$	$SS^{CH\ REQ}$
		$PKIR^{\circ}_{h} \cdot PKIR^{DIS\ REQ}$	$SS^{ATT\ REQ}$
PI^{WAIT}		$PKIR^{TSD}$	$\cdot IOCM^{BB}$
		$PKIR^{TSD} \cdot QKIR^{TSD}$	$\cdot QB'$
		$PKIR^{AG}$	AEI
		$PKIR^{QK}$	$AEI \cdot XA^{At}$

fig 10-27

HM 6-1-60

LEVEL	LEVEL LOGIC		
	TIME LEVEL	INSTRUCTION	OTHERS
START → FK	$QK^{00\alpha}$	$\overline{PKIR^{F}} \cdot \overline{PKIR^{SKM}}$	$\cdot QI^{START}$
	$QK^{13\alpha}$	$QKIR^{SPF}$	
	$QK^{13\alpha}$	$QKIR^{SPG}$	
		$PKIR^{JA}$	$FI^{\circ} \cdot EB^{\circ}$
		$PKIR^{SF}$	$FI^{\circ} \cdot EB^{\circ}$

FL-COUNTER START LEVEL

FIG 10-28

HM 6-1-60

LEVEL	LEVEL LOGIC		
	TIME LEVEL	INSTRUCTION	OTHERS
START ◇ XWIL	$PK^{14\alpha}$	$\cdot \overline{PKIR^{XM}}$	
	$PK^{14\alpha}$		$\cdot PI'_2$
	$PK^{31\alpha}$	$\cdot PKIR^{XM}$	
	$CSK^{11\alpha}$	$\cdot PKIR^{XM}$	$\cdot PK^{23\alpha} \cdot PI^{LEAVE\ SEQ}$
	$CSK^{04\alpha}$		
	$QK^{31\alpha}$	$\cdot QKIR^{AUX}$	
	$QK^{22\alpha}$	$\cdot QKIR^{X} \cdot QKIR^{LD}$	

XWIL - COUNTER START LEVEL

FIG 10-29

#M 6-1-60

LEVEL	LEVEL LOGIC		
	TIME LEVEL	INSTRUCTIONS	OTHERS
START → AK	$PK^{26\alpha}$	$\cdot PKIR^{OPRAE}$	
	$QK^{14\alpha}$	$\cdot QKIR^{AK}$	
$\overline{ASK}+1 \rightarrow ASK$	$AK'_{\alpha,2}$		
	$AK'_{\alpha,2}$	$\cdot (AKIR^{MUL} + AKIR^{TLY})$	
	$AK'_{\alpha,4}$	$\cdot (AKIR^{SH} + AKIR^{NOR})$	
	$AK_{\alpha 7}$	$\cdot AKIR^{DIV}$	

AK-START AND ASK-COUNT LEVEL LOGIC

Fig 10-30

MM 12-30-60

LEVEL	LEVEL LOGIC		
	TIME LEVEL	INSTRUCTION	OTHERS
START DSK	$PIK^{00\alpha}$		$CSK'_4 \cdot XWK^{00\alpha}$
	$PIK^{02\alpha}$		$CSK'_4 \cdot XWK^{00\alpha}$
	$PIK^{23\alpha}$		$CSK'_4 \cdot XWK^{00\alpha}$

DSK-COUNTER START LEVEL

fig 10-31

HM 6-1-60

PK - INSTRUCTION COUNTER (MEMORY SECTION)

PULSE	MEMORY	RD PULSE	PRESET	REGISTER DRIVER LOGIC		
				TIME LEVEL	MEMORY	OTHERS
$\overline{PK}+1\rightarrow PK$	VFF	α		$PK^{00\alpha}$		$\cdot PI_2^1 \cdot \overline{PI^{START_2}}$
				$PK^{00\alpha}$		$\cdot PI_2^0 \cdot \overline{PI^{START_1}}$
				$PK^{02\alpha} \cdot PKM^{VFF}$		
$\overline{109} \rightarrow PK$ $\overline{PK}+1\rightarrow PK$	ALL	α		$PK^{01\alpha}$		$\cdot PKA^0$
	S	α		$PK^{06\alpha} \cdot PKM^S$		
	T	α		$PK^{01\alpha} \cdot PKM^T$		
	U	α		$PK^{01\alpha} \cdot PKM^U$		
	VFF	α		$PK^{02\alpha} \cdot PKM^{VFF} \cdot CSK_4^0 \cdot QB^0 \cdot EB^0 \cdot \overline{A \neq B}$		
				$PK^{02\alpha} \cdot PKM^{VFF} \cdot CSK_4^0 \cdot QB^0 \cdot EB^0 \cdot \overline{VMD^{4\neq}}$		
	$\overline{VFF}$	α		$PK^{02\alpha} \cdot PKM^{\overline{VFF}}$		
$\overline{122} \rightarrow PK$ $\overline{PK}+1\rightarrow PK$	ALL	α		$PK^{15\alpha}$		$\cdot PKA^0$
	S	α		$PK^{16\alpha} \cdot PKM^S$		
	T	α		$PK^{16\alpha} \cdot PKM^T$		
	U	α		$PK^{16\alpha} \cdot PKM^U$		
	V	α		$PK^{15\alpha} \cdot PKM^V$		
$\overline{100} \rightarrow PK$ $\overline{PK}+1\rightarrow PK$	VFF	α		$PK^{02\alpha}$		$\cdot CSK^{11\alpha} \cdot PI^{\neq CH.SEQ}$

Fig 10-32

HA 5-27-60

PK - INSTRUCTION COUNTER (INSTRUCTION SECTION)

PULSE	MEMORY	RP PULSE	PRESET	REGISTER DRIVER LOGIC		
				TIME LEVEL	INSTRUCTION	OTHERS
$\overline{PK}+1 \rightarrow PK$	ALL	α		$PK^{25\alpha}$		
				$PK^{25\alpha}$	$\cdot PKIR^{JMP}$	
				$PK^{25\alpha}$	$\cdot PKIR^{JX}$	$\cdot EB' \cdot XJ$
				$PK^{25\alpha}$	$\cdot PKIR^{JA}$	$\cdot QB'$
				$PK^{25\alpha}$	$\cdot PKIR^{JA}$	$\cdot AEB$
				$PK^{25\alpha}$	$\cdot PKIR^{JA}$	$\cdot FI^{\circ}$
				$PK^{25\alpha}$	$\cdot PKIR^{TSD}$	
				$PK^{25\alpha}$	$\cdot PKIR^{IOS}$	$\cdot EB'$
				$PK^{25\alpha}$	$\cdot PKIR^{IOS}$	$\cdot QB'$
				$PK^{25\alpha}$	$\cdot PKIR^{OPL\,AE}$	$\cdot QB'$
				$PK^{25\alpha}$	$\cdot PKIR^{OPR\,AE}$	$\cdot AEB$
				$PK^{25\alpha}$	$\cdot PKIR^{SED}$	
				$PK^{25\alpha}$	$\cdot PKIR^{SKM}$	
			PRESET CE			
$\boxed{24} \rightarrow PK$ $\overline{PK}+1 \rightarrow PK$	ALL	α		$PK^{22\alpha}$		$\cdot PI_2^{\circ} \cdot \overline{PI^{WAIT}}$
				$PK^{23\alpha}$		$\cdot CSK^{11\alpha} \cdot \overline{PI^{WAIT}}$
$\boxed{31} \rightarrow PK$ $\overline{PK}+1 \rightarrow PK$	ALL	α		$PK^{25\alpha}$	$\cdot PKIR^{JMP}$	$\cdot EB^{\circ}$
				$PK^{25\alpha}$	$\cdot PKIR^{JMP}$	$\cdot PKIR^{\circ}_{cf3} \cdot PKIR^{\circ}_{cf4}$
				$PK^{28\alpha}$	$\cdot PKIR^{JX}$	
				$PK^{26\alpha}$	$\cdot PKIR^{JA}$	
				$PK^{25\alpha}$	$\cdot QKIR^{TSD}$	$\cdot QK^{01\alpha} \cdot PI_4'$
				$PK^{25\alpha}$	$\cdot QKIR^{TSD}$	$\cdot QK^{20\alpha} \cdot PI_4^{\circ}$
				$PK^{29\alpha}$	$\cdot PKIR^{IOS}$	
				$PK^{26\alpha}$	$\cdot PKIR^{OPR\,AE}$	
				$PK^{29\alpha}$	$\cdot PKIR^{SKX}$	$\cdot PKIR^{\circ}_{cf3}$
				$QK^{14\alpha}$	$\cdot QKIR^{SED}$	
				$QK^{14\alpha}$	$\cdot QKIR^{SKM}$	
$\boxed{100} \rightarrow PK$ $\overline{PK}+1 \rightarrow PK$	ALL	α		$PK^{23\alpha}$		$\cdot CSK^{11\alpha} \cdot PI^{LEAVE\,SEQ}$
				$PK^{22\alpha}$		$\cdot \quad \cdot PI^{LEAVE\,SEQ}$
				$PK^{24\alpha}$	$\cdot \overline{PKIR^{DIS}}$	
				$PK^{22\alpha}$		$\cdot PI_2'$
			PRESET CE			

QK - OPERAND COUNTER (MEMORY SECTION)

PULSE	MEMORY	RD PULSE	PRESET	REGISTER DRIVER LOGIC		
				TIME LEVEL	MEMORY	OTHERS
$\overline{QK}+1 \rightarrow QK$	ALL	α		$QK^{00\alpha}$		$\cdot \overline{QI^{START}}$
	VF	α		$QK^{03\alpha} \cdot QKM^{VF}$		
$\overset{09}{\rightarrow} QK$ $\overline{QK}+1 \rightarrow QK$	S	α		$QK^{06\alpha} \cdot QKM^{S}$		
	T	α		$QK^{01\alpha} \cdot QKM^{T}$		
	U	α		$QK^{01\alpha} \cdot QKM^{U}$		
	VFF	α		$QK^{03\alpha} \cdot QKM^{VFF} \cdot (\overline{AEB} + \overline{VMD^{AE}})$		
	$\overline{VFF}$	α		$QK^{02\alpha} \cdot QKM^{\overline{VFF}}$		
$\overset{13}{\rightarrow} QK$ $\overline{QK}+1 \rightarrow QK$	V	α		$QK^{11\alpha} \cdot QKM^{V}$		
$\overset{31}{\rightarrow} QK$ $\overline{QK}+1 \rightarrow QK$	ALL	α		$QK^{25\alpha}$		
	S	α		$QK^{23\alpha} \cdot QKM^{S} \cdot QKIR^{LOAD}$		
	T	α		$QK^{23\alpha} \cdot QKM^{T} \cdot QKIR^{LOAD}$		
	U	α		$QK^{23\alpha} \cdot QKM^{U} \cdot QKIR^{LOAD}$		
	V	α		$QK^{23\alpha} \cdot QKM^{V}$		

FIG 10-34 HM 5-27-60

QK - OPERAND COUNTER (INSTRUCTION SECTION)

PULSE	MEMORY	RD PULSE	PRESET	REGISTER DRIVER LOGIC		
				TIME LEVEL	INSTRUCTION	OTHER
13 → QK $\overline{QK}$+1 → QK	ALL	α		$QK^{11\alpha} \cdot \overline{QKIR^{INS}} \cdot \overline{QKIR^{ST}} \cdot \overline{QKIR^{FL}}$		
18 → QK $\overline{QK}$+1 → QK	ALL	α		$QK^{13\alpha} \cdot QKIR^{TSD}$ $QK^{14\alpha} \cdot QKIR^{INS}$ $QK^{14\alpha} \cdot QKIR^{SKM}$		
21 → QK $\overline{QK}$+1 → QK	ALL	α		$QK^{14\alpha} \cdot QKIR^{ST}$ $QK^{14\alpha} \cdot QKIR^{LOAD}$ $QK^{14\alpha} \cdot QKIR^{FLF}$ $QK^{14\alpha} \cdot QKIR^{FLG}$ $QK^{17\alpha} \cdot QKIR^{COM}$		
00 → QK	ALL	α	PRESET CE			

FIG 10-35

HM 5-27-60

CSK^{0}_{4} DURING
CHANGE OF SEQUENCE CYCLE

CSK (CSK CYCLE)			
4	3	2	1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

CSK'_{4} DURING
DELAY SYNC. CYCLES

CSK (DSK CYCLE)			
4	3	2	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1

CSK AND DSK CYCLES IN CSK COUNTER

FIG 10-36

HA 12-28-60

CSK CHANGE SEQUENCE COUNTER

PULSE	MEMORY	RD PULSE	DRESET	REGISTER DRIVER LOGIC				PULSE GATE LOGIC			
				TIME LEVEL	CSK4 CONDITION	INSTRUCTION	OTHERS	TIME LEVEL	CSK4 CONDITION	INSTRUCTION	OTHERS
$\overline{CSK}+1 \rightarrow CSK_{3,2,1}$		α	$\overline{DRESET_{CE}}$	$CSK^{00\alpha}$			$\cdot \overline{CSI^{START}}$				
					CSK_4^1		$\cdot \overline{XWK^{00\alpha}}$				
					CSK_4^1		$\cdot \overline{DK^{00\alpha} \cdot DK^{02\alpha} \cdot DK^{23\alpha}}$				
$00 \rightarrow CSK_{3,2,1}$			$DRESET_{CE}$								
$0 \rightarrow CSK_4$		α	$\overline{DRESET_{CE}}$					$CSK^{11\alpha}$		$\cdot DK^{00\alpha} \cdot SS^{\Delta T\ REQ}$	
								$CSK^{11\alpha}$		$\cdot DK^{02\alpha} \cdot DI^{\Delta CH\ SEQ}$	
								$CSK^{11\alpha}$		$\cdot DK^{02\alpha} \cdot \overline{\Delta TI}$	
								$CSK^{11\alpha}$		$\cdot DK^{23\alpha} \cdot DI^{LEAVE\ SEQ}$	
								$CSK^{11\alpha}$		$\cdot DK^{23\alpha} \cdot \overline{DI^{WAIT}}$	
$1 \rightarrow CSK_4$		α	$\overline{DRESET_{CE}}$							$DKIR^{JX} \cdot PKIR^{DIS\ REQ} \cdot PKIR_h^0 \cdot DK^{25\alpha} \cdot \overline{SS^{\Delta T\ REQ}}$	
										$\overline{DKIR^{JX}} \cdot DKIR^{DIS\ REQ} \cdot PKIR_h^0 \cdot DK^{31\alpha} \cdot \overline{SS^{\Delta T\ REQ}}$	
										$\cdot CSK_4^0 \cdot DK^{02\alpha} \cdot PKM^{VF} \cdot VMD^{\Delta T} \cdot \Delta TI$	
										$\cdot DK^{22\alpha} \cdot DI^{WAIT} \cdot DI_2^0 \cdot \overline{DI^{LEAVE\ SEQ}}$	
			$DRESET_{CE}$								

FIG 10-37 HM 5-27-60

XWK- INDEX WRITE COUNTER

PULSE	MEMORY	RD PULSE	PRESET	REGISTER DRIVER LOGIC			
				TIME LEVEL	XWK 3 CONDITION	INSTRUCTIONS	OTHERS
$\overline{XWK}+1 \rightarrow XWK$		α		$XWK^{00\alpha}$			
$00 \rightarrow XWK$		α		$XWK^{06\alpha}$			
$1 \rightarrow XWK_1$		α	$\overline{\text{PRESET CE}}$			START XWK	

FIG 10-38

MM 5-26-60

FK - CONFIGURATION COUNTER

PULSE	MEMORY	RD PULSE	PRESET	REGISTER DRIVER LOGIC			
				TIME LEVEL	FK8 CONDITION	INSTRUCTION	OTHERS
$\overline{FK}+1 \rightarrow FK$		α		$FK^{0\alpha}$	$\cdot FK8^{0}$		$\cdot \overline{\text{START} \rightarrow FK}$
$00 \rightarrow FK$ $\overline{FK}+1 \rightarrow FK$		α		$FK^{2\alpha}$	$\cdot$	$\overline{PKIR^{FF}}$	

FIG 10-39

HM 10-25-60

		FK COUNTER			
		8	3	2	1
FK_8^0 ↑	00d	0	0	0	0
	01d	0	0	0	1
	02d	0	0	1	0
	03d	0	0	1	1
	04d	0	1	0	0
	05d	0	1	0	1
	06d	0	1	0	0
↓	07d	0	1	1	1
FK_8^1	00d	1	0	0	0
FK_8^0	00d	0	0	0	0

FK COUNTER

FIG 10-40

	REGISTER DRIVER LOGIC		PULSE GATE LOGIC		
	RD PULSE	OTHER	RD PULSE	TIME LEVEL	OTHER
$\overline{AK}+1\rightarrow AK$	α	$AK'_{\alpha,0}\cdot$ START$_{\diamond AK}$ $AK^{\circ}_{\alpha,0}\cdot\left(\overline{\text{SYNC STOP}}+\overline{\text{STOP AE ON SEL'D SYNC}}\right)\cdot\Big[AK^{\circ}_{\alpha,1}\left(AKIR^{SH}+AKIR^{NOR}\right)+\left(AKIR^{DSA}\cdot AK^{\circ}_{\alpha,3}\right)$ $+AK^{\circ}_{\alpha,3}\cdot AK^{\circ}_{\alpha,9}\cdot\left(AKIR^{MUL}+AKIR^{ADD}\right)$ $+\left(AK^{\circ}_{\alpha,8}+ASK'_{7}\right)\cdot\left(AKIR^{DIV}+AKIR^{TLV}\right)\cdot\left(AK^{\circ}_{\alpha,2}\cdot AK^{\circ}_{\alpha,3}\cdot AK^{\circ}_{\alpha,9}\cdot AK^{\circ}_{\alpha,11}\right)$ $+\left(ASK'_{1}+ASK'_{2}\right)\cdot(\text{ditto})\cdot(\text{ditto})\Big]$			
$\lfloor C\rfloor\rightarrow AK_{\alpha,2}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,2}$	
$\lfloor C\rfloor\rightarrow AK_{\alpha,3}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,3}$	
$\lfloor C\rfloor\rightarrow AK_{\alpha,5}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,5}\cdot\bar{f}_4$	
$\lfloor C\rfloor\rightarrow AK_{\alpha,6}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,6}\cdot f_4$	
$\lfloor C\rfloor\rightarrow AK_{\alpha,8}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,8}$	
$\lfloor C\rfloor\rightarrow AK_{\alpha,9}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,9}$	
$\lfloor C\rfloor\rightarrow AK_{\alpha,10}$	α	PRESET$_{\rightarrow AK}$		$AK'_{\alpha,10}$	
$AK_{\alpha,2}\rightarrow AK_{\beta,2}$		$AK_{\alpha}\xrightarrow{\ f\ }AK_{\beta}$	β	$AK_{\alpha,2}$	
$AK_{\alpha,3}\rightarrow AK_{\beta,3}$		$AK_{\alpha}\xrightarrow{\ f\ }AK_{\beta}$	β	$AK_{\alpha,3}$	
$AK_{\alpha,9}\rightarrow AK_{\beta,9}$		$AK_{\alpha}\xrightarrow{\ f\ }AK_{\beta}$	β	$AK_{\alpha,9}$	

WHERE $\lfloor$PRESET$\rfloor_{\rightarrow AK} = AK'_{\alpha,3}\cdot AKIR^{ADD}\left(AKIR^{MUL}+ASK^{\circ}_{7}\right)$
$+AKIR^{DIV}\cdot\left(AK'_{\alpha,2}+AK'_{\alpha,9}\right)$
$+AKIR^{DIV}\cdot\left(ASK^{\circ}_{1}\cdot ASK'_{2}\cdot ASK^{\circ}_{7}\cdot AK'_{\alpha,8}\right)$

$AK_{\alpha}\xrightarrow{\ f\ }AK_{\beta}=\beta$

$\lfloor$START$\rfloor_{\diamond AK} = PK^{2\alpha}\cdot PKIR^{\text{OPRAE}}+QK^{4\alpha}\cdot QKIR^{AK}$

Fig 10-41

#/A 12-30-60

SHIFT REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHER	TIME LEVEL	INSTRUCTION	OTHER
$\downarrow^{1}_{\triangleright ASK_1}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{2N}$	$(F_1 + f_2)$
$\downarrow^{1}_{\triangleright ASK_2}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{N}$ $AKIR^{2N}$	$\cdot f_2$ $\cdot (f_4 + f_3 \cdot a_2^0 \cdot a_3^0 \cdot a_4^0)$ $\cdot f_3 \cdot (a_2' + a_3' + a_4')$
$\downarrow^{1}_{\triangleright ASK_3}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{N}$ $AKIR^{2N}$	$\cdot [f_1 + f_3 (a_2' + a_3' + a_4')]$ $\cdot (f_4 + f_3 \cdot a_2^0 \cdot a_3^0 \cdot a_4^0)$ $\cdot f_2$
$\downarrow^{1}_{\triangleright ASK_4}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{2N}$	$\cdot f_3 (a_2' + a_3' + a_4')$
$\downarrow^{1}_{\triangleright ASK_5}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{N}$ $AKIR^{2N}$	$\cdot f_4 + (f_3 \cdot a_2 \cdot a_3 \cdot a_4)$ $\cdot f_2$ $\cdot f_1$
$\downarrow^{1}_{\triangleright ASK_6}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{N}$ $AKIR^{2N}$	$\cdot \overline{f_1}$ $\cdot f_1$ $\cdot f_4 + (f_3 \cdot a_2^0 \cdot a_3^0 \cdot a_4^0)$
$\downarrow^{1}_{\triangleright ASK_7}$	$\propto$	$AK'_{\propto,1}$				$AKIR^{N}$	$\cdot \overline{f_1}$
$\downarrow^{0}_{\triangleright ASK}$	$\propto$	$AK'_{\propto,0}$	$\cdot \downarrow^{START}_{\triangleright AK}$				

WHERE: $\downarrow^{START}_{\triangleright AK} = QK^{14\propto} \cdot QKIR^{AK} + PK^{26\propto} \cdot PKIR^{OPR\,AC}$

ASK SHIFT REGISTER LOGIC

FIG 10-42

HA 12-30-60

CHAPTER 11

MEMORY ELEMENT

TABLE OF CONTENTS

11-1 INTRODUCTION

11-2 MEMORY ADDRESS SELECTOR

11-3 STROBE SELECTOR

11-4 INHIBIT SELECTOR

11-5 S MEMORY

- 11-5.1 ADDRESS DECODING
- 11-5.2 READ-WRITE OPERATION

11-6 T MEMORY

- 11-6.1 ADDRESS DECODING
- 11-6.2 READ-WRITE OPERATION

11-7 V MEMORY DECODING

- 11-7.1 PLUGBOARD STORAGES A AND B
- 11-7.2 TOGGLE SWITCH STORAGE
- 11-7.3 SHAFT ENCODER
- 11-7.4 REAL TIME CLOCK
- 11-7.5 INPUT MIXER
- 11-7.6 V_{FF} MEMORY

11-8 PARITY

LIST OF FIGURES

11-1 MEMORY ELEMENT BLOCK DIAGRAM

11-2 TYPICAL MEMORY ADDRESS SELECTOR DIGIT STAGE

11-3 INSTRUCTION MEMORY ADDRESS SELECTOR CONTROL LEVEL

11-4 PKA MEMORY SELECTOR CONTROL FLIP-FLOP

11-5 DFA MEMORY SELECTOR CONTROL FLIP-FLOP

11-6 OPERAND AND DEFERRED ADDRESS MEMORY ADDRESS SELECTOR CONTROL LEVEL

11-7 QKA MEMORY SELECTOR CONTROL FLIP-FLOP

11-8 INSTRUCTION AND DEFERRED ADDRESS, READ-WRITE, STROBE AND INHIBIT SELECTOR LEVEL CONTROL

11-9 OPERAND, READ-WRITE, STROBE AND INHIBIT SELECTOR LEVEL CONTROL

11-10 MEMORY STROBE INTO M AND N REGISTERS FOR S, T, U AND $V_{\overline{FF}}$ MEMORIES

11-11 MEMORY CLEAR AND STROBE INTO N REGISTER

11-12 MEMORY CLEAR AND STROBE INTO M REGISTER

11-13 TYPICAL MEMORY DIGIT INHIBIT LOGIC STAGE

11-14 S, T AND U MEMORY INHIBIT SELECTOR CONTROL FOR INSTRUCTION WORD REWRITE

11-15 S, T AND U MEMORY INHIBIT SELECTOR CONTROL FOR OPERAND WORD REWRITE

11-16 S MEMORY ADDRESS DECODER, READ-WRITE UNIT AND STACK INHIBIT SELECTOR

11-17 READ-WRITE UNITS OF S MEMORY

11-18 S MEMORY READ-WRITE SR_U FLIP-FLOP CONTROL LOGIC

11-19 S MEMORY READ-WRITE SR_V FLIP-FLOP CONTROL LOGIC

11-20 S MEMORY INHIBIT FLIP-FLOP CONTROL LOGIC
11-21 T MEMORY 1ST AND 2ND LEVEL X AND Y ADDRESS DECODERS
11-22 READ-WRITE UNIT OF T MEMORY
11-23 T MEMORY READ FLIP-FLOP CONTROL LOGIC
11-24 T MEMORY WRITE FLIP-FLOP CONTROL LOGIC
11-25 T MEMORY INHIBIT FLIP-FLOP CONTROL LOGIC
11-26 V MEMORY DECODER
11-27 PLUGBOARD STORAGE A
11-28 PLUGBOARD STORAGE B
11-29 TOGGLE SWITCH STORAGE
11-30 SHAFT ENCODER
11-31 REAL TIME CLOCK
11-32 TYPICAL STAGE OF V MEMORY INPUT MIXER
11-33 V_{FF} MEMORY INSTRUCTION AND DEFERRED ADDRESS WORD TRANSFER
11-34 V_{FF} MEMORY OPERAND WORD TRANSFER
11-35 TYPICAL PARITY STAGE
11-36 M PARITY COUNT WITH S, T AND U MEMORIES
11-37 N PARITY COUNT WITH S, T AND U MEMORIES
11-38 X MEMORY PARITY COUNT

CHAPTER 11
MEMORY ELEMENT

11-1 INTRODUCTION

The primary function of the Memory Element is to store programs and data while they are not being used.

The Memory Element consists of four separate memories. Three of these are magnetic core memories (S, T and U). The fourth, or V Memory, is divided into two groups: a static memory called $V_{\overline{FF}}$ (or V_T) which can be altered manually only; and a flip-flop memory called V_{FF} which can be altered by the machine. The $V_{\overline{FF}}$ Memory consists of several different devices: plugboards, toggle switch registers, a shaft encoder and a real time clock. The V_{FF} Memory consists of the A, B, C and D registers in the Arithmetic Element and the E register in the Exchange Element. The general structure of these memories was discussed in Chapter 4.

There are several units in the Memory Element, each designed to control some aspect of the over-all memory cycle. The more important of these units are shown in Fig. 11-1. Since there is more than one memory in the Memory Element, it is necessary to have a unit that determines which memory is selected and when. Both of these questions are answered by the Memory Address Selector. There is also the problem of determining which register in the selected memory is selected. This is determined by the address decoder associated with each memory. The S, T and U memories each have read-write units that control the READ and WRITE processes in these memories. A Memory Strobe Selector is used to read out the content of the selected register and similarly a Memory Inhibit Selector is used to write information into the selected register. Finally, there are two parity check circuits: one on the N Memory buffer register and the other on the M Memory buffer register.

11-2 MEMORY ADDRESS SELECTOR

The function of the Memory Address Selector is to select the proper memory during an instruction, deferred address, or operand memory cycle. The Memory Address Selector is made up of the Memory Address Digit Selector and the Memory Address Control. The leftmost bits in the P and Q registers are used by the Memory Address Control; while the remaining bits in the P and Q registers are used by the Memory Address Digit Selector.

The Memory Address Digit Selector is made up of 16 similar stages. The i.jth stage is associated with the i.jth bits in the P and Q registers. A typical stage is shown in Fig. 11-2. The output levels (MAS) of each stage of the selector are routed to the address decoders of the memories. There are usually four outputs from each stage, one for each of the four memories, S, T, U and V. The exception is that not all 16 bits in the P and Q registers are used by each memory. The S Memory uses 16 bits; the T and U memories each use 12 bits; and the V Memory uses only 7 bits.

There are two situations which generate MAS levels. The first situation occurs during an instruction memory address cycle, when the contents of the P register are combined in the Memory Address Digit Selector with a $PM^{S(T,\ U\ or\ V)}$ level to generate a set of MAS levels. The $PM^{S(T,\ U\ or\ V)}$ levels are generated by the logic shown on Fig. 11-3. This logic involves the state of the PKA and DFA interlocks and the state of the leftmost bits in the P register. PKA must be set to ONE. This occurs at the start of either an instruction cycle or an intermediate deferred address cycle. (See Fig. 11-4.) DFA must be cleared to ZERO. This occurs at the start of an instruction cycle. (See Fig. 11-5.) PKA^{1} and DFA^{0} then allow a $PM^{S(T,\ U\ or\ V)}$ level to be generated.

The second situation arises during an operand address cycle or during a deferred address cycle. Either kind of cycle will generate $QM^{S(T,\ U\ or\ V)}$ levels. These levels are then combined in the Memory Address Digit Selector with the content of the Q register to generate a set of MAS levels. The $QM^{S(T,\ U\ or\ V)}$ levels are generated by the logic shown on Fig. 11-6. This logic involves the state of the QKA, PKA and DFA interlocks and the state of the leftmost bits in the Q register. If an operand cycle is executed, QKA is set to ONE at the start of the operand cycle. (See Fig. 11-7.) QKA^{1} is one of the interlock conditions that allows a $QM^{S(T,\ U\ or\ V)}$ level to be generated. Only the different states of the QKA, PKA and DFA interlocks distinguish between an operand address and a deferred address. In a deferred address, both the PKA and DFA interlocks are set to ONE. The PKA interlock is set when either starting an instruction cycle or executing an intermediate deferred address cycle (see Fig. 11-4). The DFA interlock is set to ONE when a deferred address cycle is executed (see Fig. 11-5). PKA^{1} and DFA^{1} is another interlock condition that allows a $QM^{S(T,\ U\ or\ V)}$ level to be generated.

Another set of levels, $PKM^{S(T,\ U\ or\ V)}$ and $QKM^{S(T,\ U\ or\ V)}$ are formed in a manner similar to the $PM^{S(T,\ U\ or\ V)}$ and $QKM^{S(T,\ U\ or\ V)}$ levels. The former levels control the function of the read-write, strobe and inhibit selectors during the execution of an instruction, operand or deferred address cycle. Generally speaking, these levels control the occurrence of events during the operation of the specified memories by the indicated control counter. E.g., PKM^{S} is used to control events during an S Memory read-write cycle that uses the PK counter (this would be either an instruction or deferred address cycle), and QKM^{S} is used to control events during an S Memory read-write cycle that uses the QK counter (this would be an operand cycle).

The logic that generates the $PKM^{S(T,\ U\ or\ V)}$ levels is shown in Fig. 11-8. There are two situations which generate these levels. The first situation occurs in an instruction cycle and requires that PKA be set to ONE and DFA be cleared to ZERO. In this case, the content of the P register is used to select the proper memory. The second situation occurs in a deferred address cycle and requires that both PKA and DFA be set to ONE. In this case the contents of the Q register are used to select the proper memory.

The logic that generates the $QKM^{S(T,\ U\ or\ V)}$ levels is shown in Fig. 11-9. The logic requires that QKA be set to ONE. The contents of the Q register are used to select the proper register.

The QKM^V level (and similarly the PKM^V level) used for the V Memory is further divided into levels for the V_{FF} and $V_{\overline{FF}}$ memories. The V_{FF} Memory level $QKM^V{}_{FF}$ is formed by ANDing the VMD_{FF} level and the QKM^V level. The $V_{\overline{FF}}$ memory level $QKM^V{}_{\overline{FF}}$ is formed by ANDing the $\overline{VMD_{FF}}$ level and the QKM^V level.

11-3 STROBE SELECTOR

The Memory Strobe Selector determines whether information coming out of the selected Memory should be strobed into the M or N registers. The selected register depends upon whether an instruction, operand, or deferred address cycle is being executed. Fig. 11-1 shows the information flow paths involving the computer and the Memory Strobe Selector.

The Memory Strobe Selector is basically a double gating circuit. A typical stage for one memory is shown in Fig. 11-10. Four such gating circuits are used, one for each memory. The first gate routes the information coming from the memory sense amplifiers to the M and N pulse gate inputs. Which specific strobe pulse then occurs depends on the memory selected and whether an instruction, deferred address, or operand cycle is being executed.

During instruction or deferred address cycles, a memory strobe pulse routes information from the selected memory into the N register. The logic governing the memory strobe pulses is shown in Fig. 11-11. The pulse which transfers ONES usually consists of two pulses, one for each pair of quarters. The pulse which transfers ZEROES occurs in the third quarter only. The first, second and fourth quarters of the N register are usually cleared at $PK^{10\alpha}$. For the S Memory, the strobe pulses occur at $PK^{10\beta}$ during the READ cycle of the instruction or deferred address cycle (i.e., when the PKM^S level exists). They occur at $PK^{11\alpha}$ for the T, U and V memories.

During an operand cycle, the operand strobe pulse routes information from the selected memory into the M register. The logic governing the memory strobe pulse is shown in Fig. 11-12. The pulse which transfers ONES usually consists of two pulses, one for each pair of quarters. The whole M register is usually cleared at $QK^{09\alpha}$. For the S Memory, the strobe pulses occur at $QK^{10\beta}$ during the READ cycle of the operand cycle. For the T, U and V memories, these pulses occur at $PK^{11\beta}$ during the READ cycle.

The timing of other pulses during both PK and QK cycles assumes, if there is any question, that the last memory strobe pulse will occur in the 11β state of both counters.

Note that the Strobe Selector does not influence memory read-outs from the V_{FF} Memory since there are no strobe pulses per se when this memory is selected.

11-4 INHIBIT SELECTOR

The Inhibit Selector is used to route inhibit currents to the memory cores in the selected memory. A read-out from core memories is destructive, i.e., all the bits in the selected

memory register are left cleared by the reading process. During the WRITE part of a read-write cycle, inhibit currents are generated in the cores in which ZEROES are to be written. The inhibit currents prevent the core from changing state during the writing process. As we shall see, the Inhibit Selector effectively routes information from the buffer register to the selected memory register in order that these inhibit currents can be generated in the selected cores.

The Inhibit Selector consists of 38 similar stages (one for each bit). A typical stage is shown in Fig. 11-13. There are three possible output levels in each stage, one for each of the three memories, S, T and U. The $V_{\overline{FF}}$ Memory does not require a WRITE cycle, thus no inhibit logic is necessary. The corresponding fourth position in the Inhibit Selector is used for non-memory purposes.

There are two situations in which the Inhibit Selector generates $S(T \text{ or } U)_{i.j}$ INH levels. The first situation occurs during the execution of an instruction or deferred address cycle, when the contents of the N register are written back into the selected memory register. The inputs to the Inhibit Selector in this case are the $N_{i.j}$ levels, representing the contents of the N register, and the $N_{i.j} \longrightarrow\!\diamond SM_{i.j}$ levels.

The $N_{i.j} \longrightarrow\!\diamond SM_{i.j}$ levels are generated by the logic shown on Fig. 11-14. The inputs in this logic are the PKM^S levels (see Fig. 11-8) and levels from the SINH flip-flops. Note that the $SINH_{i.j}$ levels shown on Fig. 11-13 are completely different from the $SINH^1$ levels shown on Fig. 11-14. As we have seen, the latter are used in generating the former. Similar logic generates the $TINH_{i.j}$ and $UINH_{i.j}$ levels, using the $N_{i.j} \longrightarrow\!\diamond TM_{i.j}$ and $N_{i.j} \longrightarrow\!\diamond UM_{i.j}$ level, respectively. Delay lines are used in generating the $N \longrightarrow\!\diamond SM$ levels only, since the S Memory requires that the $N \longrightarrow\!\diamond SM$ levels vary sequentially (i.e., "ripple"). The delay line is designed so that the level for each successive bit is turned on after a delay step of 0.015 microsecond.

The second situation occurs during the execution of an operand cycle when the contents of the M register are written back into the selected memory register. In this case, the inputs to the memory digit inhibit selector (Fig. 11-13) are the $M_{i.j}$ levels representing the contents of the M register and the $M_{i.j} \longrightarrow\!\diamond SM_{i.j}$ levels.

The $M_{i.j} \longrightarrow\!\diamond SM_{i.j}$ levels are generated by the logic shown on Fig. 11-15. The inputs in this logic are QKM^S levels (see Fig. 11-9) and levels from the $SINH^1$ flip-flops. The logic generating the $M_{i.j} \longrightarrow\!\diamond SM_{i.j}$ levels is very similar to that generating the $N_{i.j} \longrightarrow\!\diamond SM_{i.j}$ levels.

11-5 S MEMORY

11-5.1 ADDRESS DECODING. The MAS_S lines from the memory address decoder are channelled into four decoders and associated read-write units where they produce four sets of 10 decoder lines (YU, YV, XU and XV). Each set of decoder lines contains eight

lines decoded from three MAS lines. Every fourth MAS line is associated with either an SR_U^1 or SR_V^1 level. These levels are the inputs to the read-write unit. This unit generates the remaining two lines in each set of 10.

Bits $MAS_{1.8}$ and $MAS_{1.7}$ are also decoded in the memory stack inhibit selector into four selection lines. These four selection lines are amplified and split into four outputs per selection line. This gives a total of sixteen inhibit selection lines.

11-5.2 READ-WRITE OPERATION. The read-write units produce levels used by the current regulators in the XU, XV, YU and YV switch core drivers. (See Fig. 4-10, Chapter 4.) These levels are generated by combining MAS_S levels and the SR_U and SR_V flip-flop levels as shown in Fig. 11-17.

The S Memory read and write flip-flops, SR_U and SR_V, determine when the READ or WRITE operation should take place. The READ operation takes place when both flip-flops are set to ONES. The WRITE operation takes place when both flip-flops are cleared to ZEROES.

The logic for setting and clearing SR_U and SR_V is shown in Figs. 11-18 and 11-19 respectively. Although the pulse setting SR_U is generated at $PK^{03\beta}$ or $QK^{03\beta}$, the pulse doesn't actually get to the flip-flop until after the delays shown in Figs. 11-18 and 11-19.

The logic for setting and clearing the SINH flip-flop is shown in Fig. 11-20.

11-6 T MEMORY

11-6.1 ADDRESS DECODING. The MAS_T lines from the memory address selector are channelled into four first level decoders as shown in Fig. 11-21. Each set of three MAS_T lines is decoded into eight lines. The pair of eight decoder lines generated from the MAS_T lines from 1.1 to 1.6 become the inputs to a second level decoder that in turn generates 64 X selection levels. 64 Y selection levels are generated in a similar manner from the MAS_T lines from 1.7 to 2.3. The X and Y levels select the core in the T Memory itself.

11-6.2 READ-WRITE OPERATION. Actually each second level decoder has three inputs: two coordinate selection levels and a read-write level. (See Fig. 11-21.) All three levels must be present before an output level is generated.

Fig. 11-22 shows the read-write unit which generates the read-write level. Note that this unit contains two read-write generators for each of the X and Y coordinates. The inputs to this unit are the TR^1 and TW^1 levels and the 1.2 and 1.8 bits of the T Memory address selector. If a READ operation is occurring, TR is set to ONE and

if a WRITE operation is occurring, TW is set to ONE. Note that the $MAS_{T1.2}$ and $MAS_{T1.8}$ lines are used redundantly, i.e., they are inputs to both the first level decoders and the read-write unit. This is done so that the second level decoder can be split in half and each half driven by one read-write generator. This scheme uses the selection logic to reduce the load on each read-write generator.

The logic that sets and clears TR is shown in Fig. 11-23. A T Memory read pulse is generated at $PK^{01\alpha}$ and $QK^{01\alpha}$. After a time delay of 0.4 microsecond, the T Memory read flip-flop is set by this pulse. The same pulse clears the flip-flop after a delay of 1.6 microseconds.

The logic that sets and clears TW is shown in Fig. 11-24. The delay logic is similar to that for TR.

The logic that sets and clears TINH is shown in Fig. 11-25. It is identical to that for TW except for the different time delays used.

11-7 V MEMORY ADDRESS DECODING

The V Memory decoder requires two levels of decoding to select the proper V Memory.

The first level decoder is shown in Fig. 11-25. It consists of two decoders. The first decoder decodes bits $MAS_{V1.1}$ to $MAS_{V1.3}$ into eight lines plus $MAS^{1}_{V1.3}$ and $MAS^{0}_{V1.3}$. The second decoder decodes bits $MAS_{V1.4}$ to $MAS_{V1.7}$ into sixteen lines plus VMD_{FF}. VMD_{FF} is a decoding of bits $MAS_{V1.5}$ to $MAS_{V1.7}$ only.

The second level decoder is actually an "AND" circuit which combines the outputs of each of the two first level decoders in order to produce levels which will select the proper memory or the proper register in the proper memory. The second level decoders for the Real Time Clock and the Shaft Encoder are shown in Fig. 11-26. The second level decoders for the other V memories are shown in the figures illustrating those memories.

11-7.1 PLUGBOARD STORAGES A AND B.

Each plugboard contains 16 registers of 37 bits each. The Plugboard Storage A register-selection is shown in Fig. 11-27. The registers are divided into two groups of eight registers. VMD^{16X}_{PBA} selects registers 0 through 7, while VMD^{17X}_{PBA} selects register 10 through 17. The specific register within the group is selected by VMD^{XX0} through VMD^{XX7}.

Plugboard Storage B register-selection is shown in Fig. 11-28. It is similar to that described above except that VMD^{14X}_{PBB} and VMD^{15X}_{PBB} levels are used to select the two groups of eight registers.

11-7.2 TOGGLE SWITCH STORAGE. The toggle switch storage contains 24 registers of 37 bits each. The register selection logic is shown on Fig. 11-29. The registers are divided into three groups of eight registers. VMD_{TSS}^{12X} selects registers 0 through 7; VMD_{TSS}^{13X} selects registers 10 through 17; and VMD_{TSS}^{14X} selects registers 20 through 27. The specific register within the group is selected by VMD^{XX0} through VMD^{XX7}. Note that only registers 0-17 currently exist.

11-7.3 SHAFT ENCODER. The Shaft Encoder is a device which converts an analog input into a digital electrical representation by means of a dual brush-disc device. The output of each Shaft Encoder represents a 9 bit binary number. Four Shaft Encoders generate a 36 bit number. A toggle switch is used for the meta-bit.

The output of the Shaft Encoder is selected by the VMD^{020} level as shown on Fig. 11-30.

11-7.4 REAL TIME CLOCK. The Real Time Clock is a 36 bit counter plus a meta-bit. The output of the Real Time Clock is selected by the VMD_{CK}^{030} level as shown on Fig. 11-31.

The counter is divided into four quarters. A carry occurs from one quarter to the next with an end-around carry from the fourth quarter into the first quarter.

The inputs to the counter are a clear pulse, beta clock pulse, and 100 kilocycle pulse.

The outputs of the counter are combined in an output mixer with the VMD_{CK}^{030} level from the V Memory decoder to form 36 VMD_{CK} levels. The $VMD_{CK\ 4.10}$ level is set by a toggle switch.

11-7.5 INPUT MIXER. The output levels of the various $V_{\overline{FF}}$ memories are routed through a central input mixer. The output of the mixer then communicates with the M and N registers in the central computer in the same manner the Memory Element sense amplifiers do. There is one input mixer stage for each bit, making a total of 38 such stages. A typical stage and its inputs are shown on Fig. 11-32.

11-7.6 V_{FF} MEMORY. The V_{FF} Memory consists of the A, B, C and D registers in the Arithmetic Element and the E register in the Exchange Element. Read-out from these memory registers is non-destructive.

When an instruction or a deferred address word is read-out, the contents of the selected register are transferred into the N register via the E register as shown in Fig. 11-33. Similarly, when an operand word is read-out, the contents of the selected register are transferred into the M register via the E register as shown in Fig. 11-34.

During these transfers through the E register, the original contents of the E register are temporarily saved in the M register until they can be returned to the E register.

The logic governing these transfers is found in the chapters on the elements in which the transfers occur.

11-8 PARITY

The function of the Parity Count circuits is to check the validity of the read-outs from the S, T and U memories. All the bits of the full memory word are checked by a parity count circuit in the M or N register. This circuit is made by pyramiding stages of individual parity circuits. A typical stage in this pyramid is shown in Fig. 11-35. In a typical pyramid there are 16 such circuits.

The six bits 4.6 to 4.10, and 2.10 are not in the pyramid. Instead they are tied in as shown in Fig. 11-36 (for the M Parity Count).

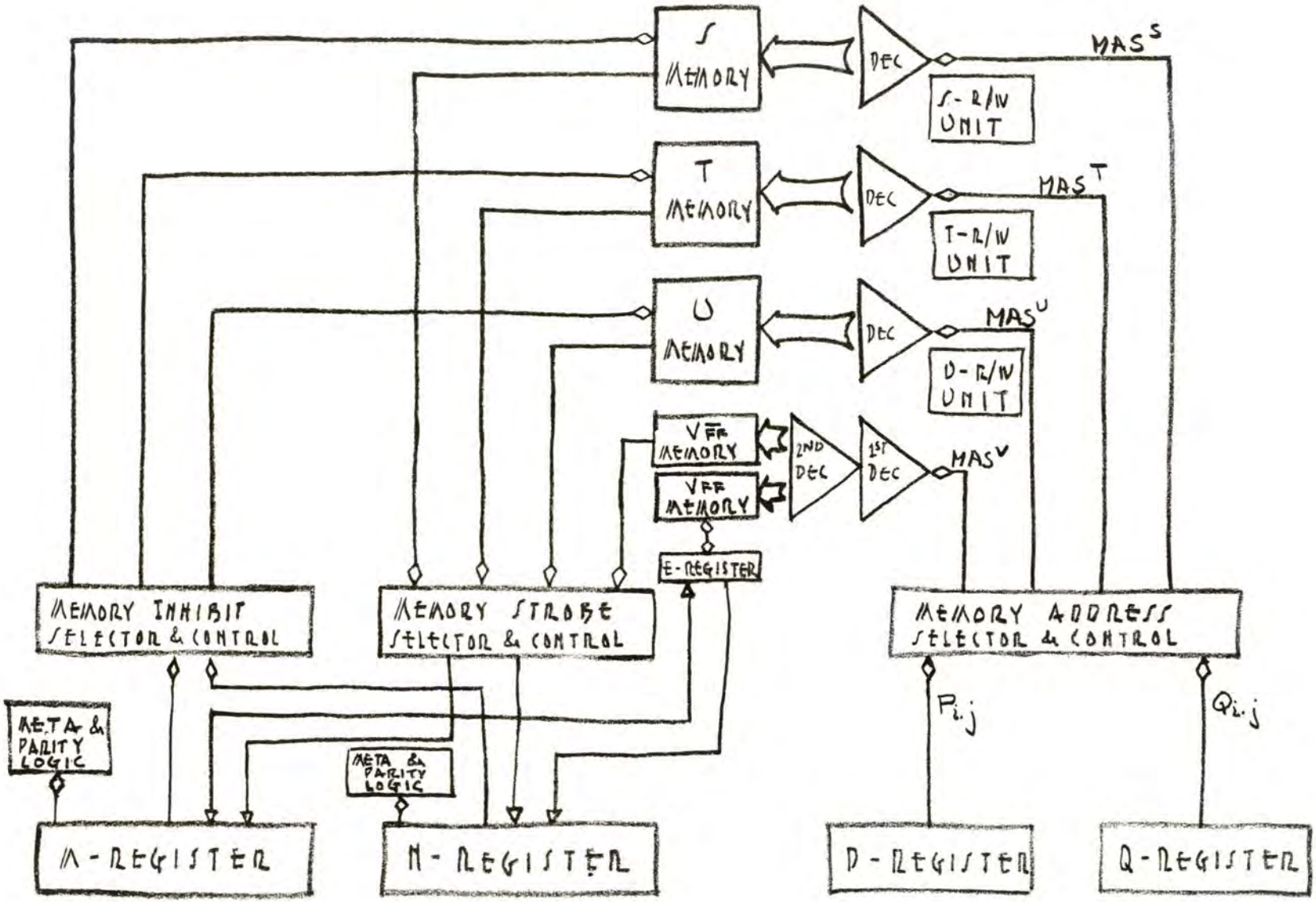
Two outputs are generated by each parity check circuit. One is a "check parity level" which determines the correctness of the parity of the entire word. This level is used to generate an alarm when the parity is incorrect. The other level is a "compute parity level". This level determines the parity bit inhibit current when the word in the buffer register is written back in memory. This level forces the parity of the word written in memory to always be a correct parity.

The M and N parity circuits also contain elements which control the value of the bit written in the 4.10 position, as specified by the Trapping Sequence. This bit is written either as a ONE, or according to the contents of the 4.10 bit in the buffer register itself. The logic is described in Chapter 15. The logic for $INHM_{4.10}$ and $INHN_{4.10}$ is given below.

The output of the larger pyramid, along with the output of another pyramid covering bits 4.6 to 4.10 and 2.10 provide the "check parity level" MP_{38}^{EVEN}. The "compute parity level", MP_{37}^{EVEN}, consists of the outputs of the larger pyramid and of another pyramid formed from bits 4.6 to 4.9 and the $INHM_{4.10}$ (or $\overline{M_{4.10}^{1} \cdot SMB \cdot SM^{1}}$) level.

The N Parity Count circuit, as shown in Fig. 11-37, is similar to the M Parity circuit shown in Fig. 11-36 but with two incidental differences. The first difference is that the bits of the N register are used instead of the M register. The second difference is in the "compute parity level", NP_{37}^{EVEN}. The secondary pyramid is also formed by bits 4.6 to 4.9 and $INHN_{4.10}$ level, but here $INHM_{4.10}$ is $\overline{(N_{4.10}^{1}) + (DFA^{1} \cdot SMB \cdot SND^{1}) + (DFA^{0} \cdot SMB \cdot SNI^{1})}$.

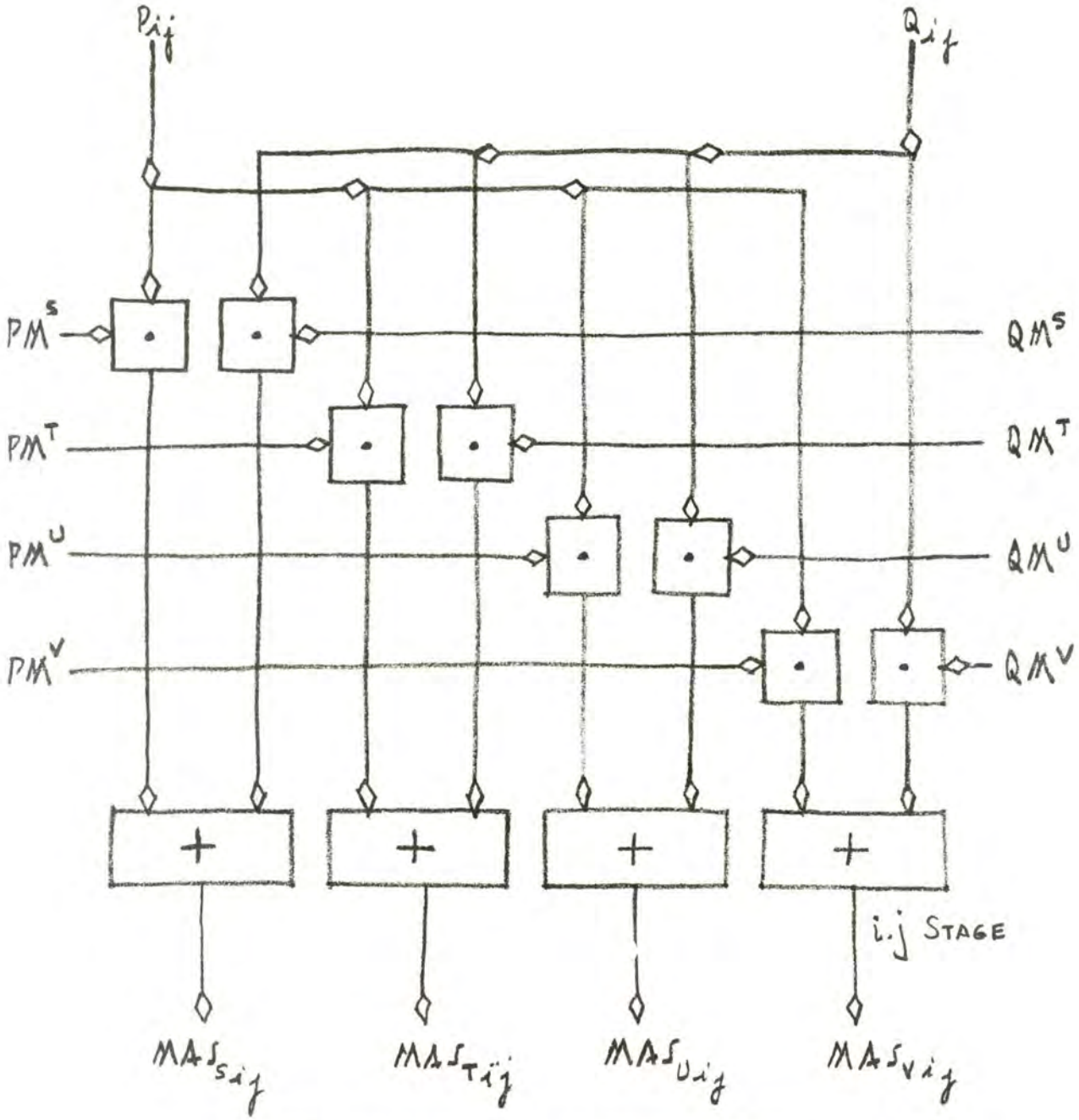
The X Memory parity circuit shown in Fig. 11-38 uses a smaller pyramid with a base of 8 stages for the 16 bits of the X Memory word. The output of this pyramid is pyramided with the output of the stage whose inputs are the 2.8 and 2.9 bits. The outputs of this final pyramid are the "compute parity levels" (XP_{18}^{ODD} and XP_{18}^{EV}). These levels are then fed into another parity stage with the outputs of the XP flip-flop to form the "check parity levels" (XP_{19}^{ODD} and XP_{19}^{EV}).



MEMORY ELEMENT BLOCK DIAGRAM

FIG 11-1

HM 9-15-60

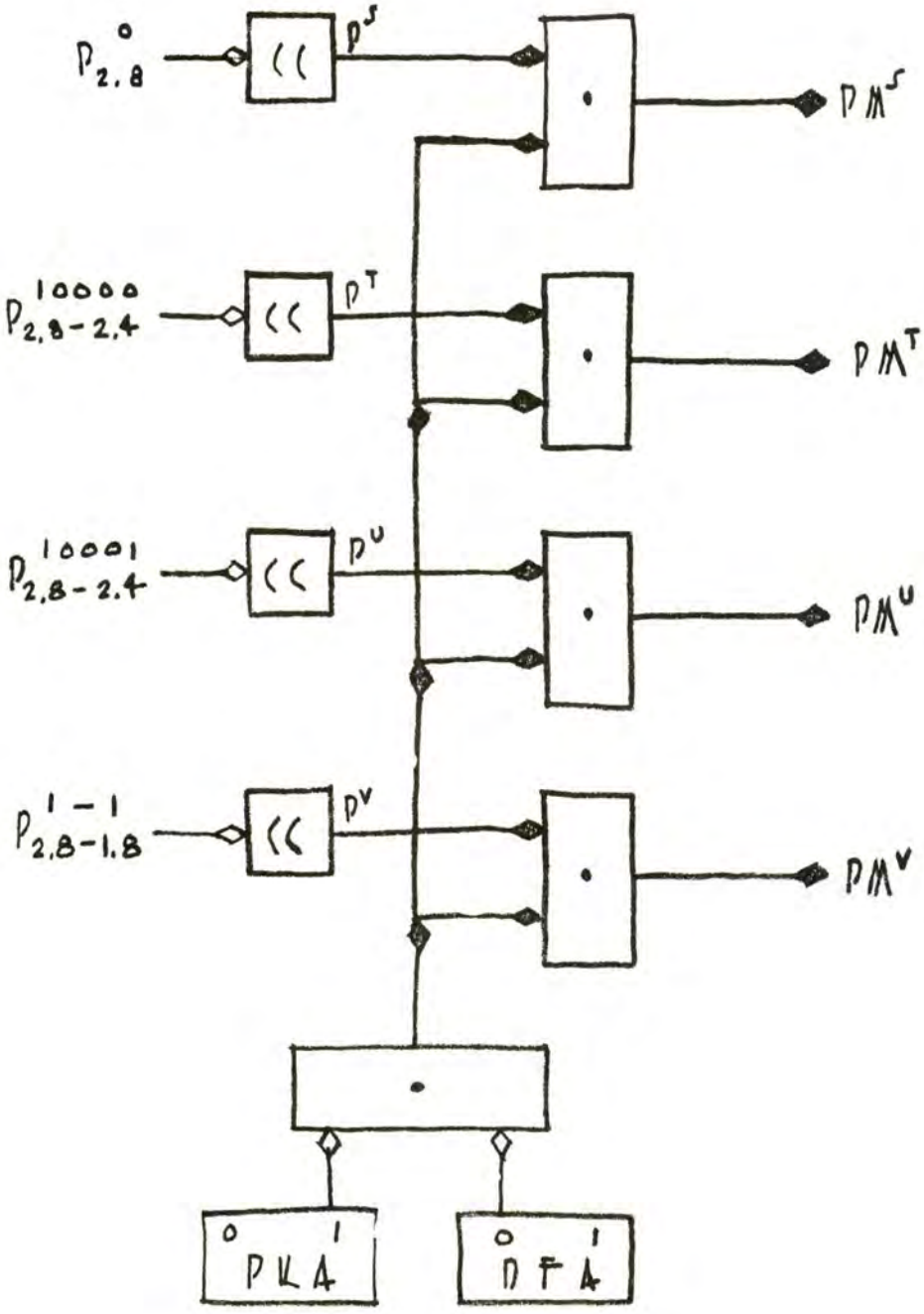


LEVEL	BITS SELECTED
MAS_{Sij}	2,7 ← ij → 1.1
MAS_{Tij}	2,3 ← ij → 1.1
MAS_{Uij}	2,3 ← ij → 1.1
MAS_{Vij}	17 ← ij → 1.1

TYPICAL MEMORY ADDRESS SELECTOR
DIGIT STAGE

HM 8-30-60

FIG 11-2



INSTRUCTION MEMORY ADDRESS SELECTOR CONTROL LEVEL

FIG 11-3 MA 9-1-60

SD67716

PKA FF LOGIC							
PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIMELEVEL	INSTRUCTION	OTHERS	TIMELEVEL	INSTRUCTION	OTHERS
1 → PKA	α			$\overline{\text{PRESET}}$ CE	$PK^{00\alpha}$ $PK^{00\alpha}$		$\cdot DI^{START_2} \cdot DI_5'$ $\cdot DI^{START_1} \cdot DI_2^0$
0 → PKA	α			$\overline{\text{PRESET}}$ CE	$PK^{00\alpha}$ $PK^{00\alpha}$ $PK^{23\alpha}$ $PK^{24\alpha}$		$\cdot \overline{DI^{START_2}} \cdot DI_5'$ $\cdot DI_2^0 \cdot DI_5'$
	PRESET CE						

Fig. 11-4 PKA MEMORY SELECTOR CONTROL FLIP-FLOP

FIG 11-

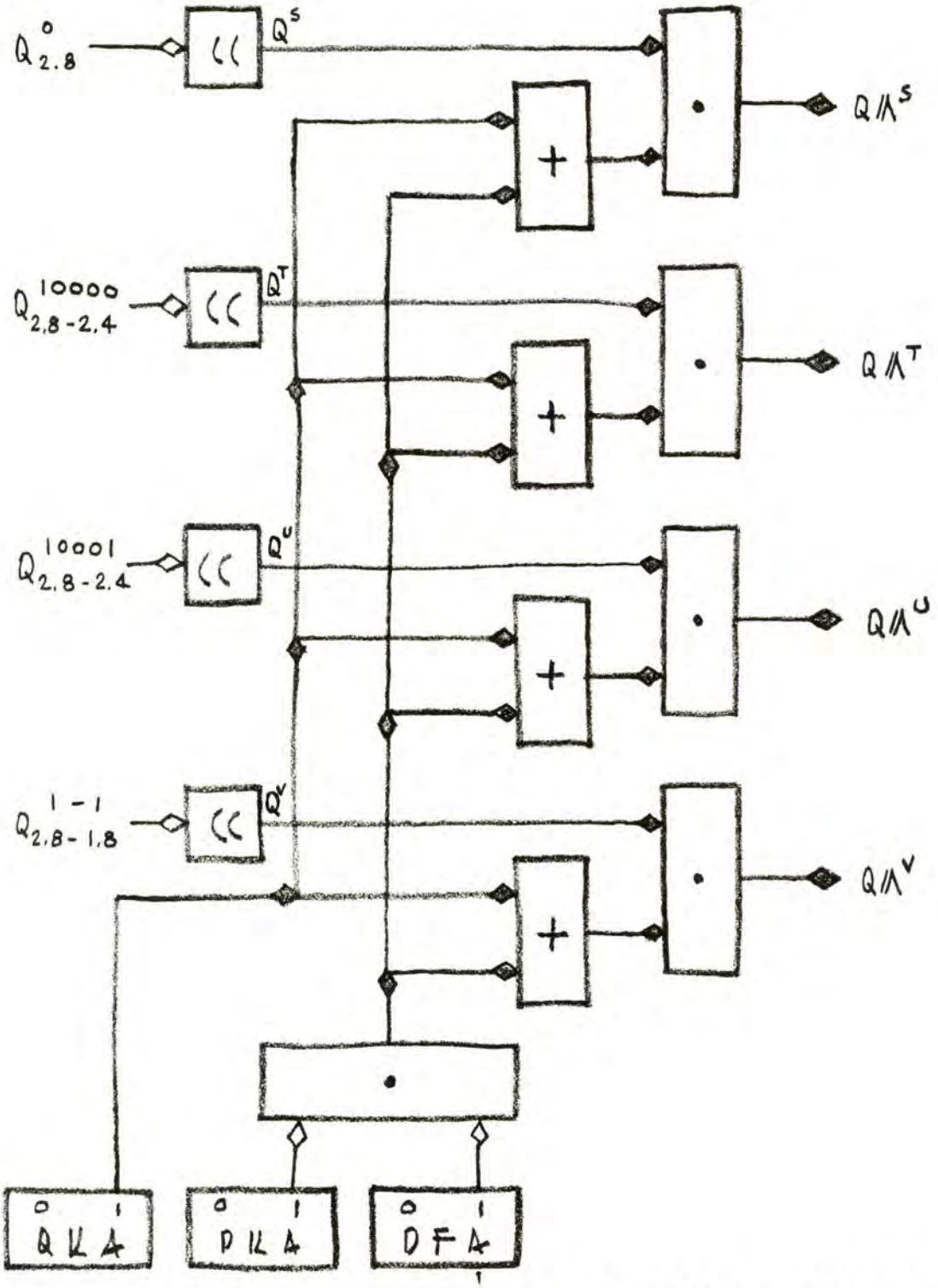
HM 9-13-60

	DFA FF LOGIC						
PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
1 → DFA	$\propto$			$\overline{\text{PRESET}}$ CE	$PR^{00\propto}$		$\cdot PI^{START_2} \cdot PI_2^1$
0 → DFA	$\propto$			$\overline{\text{PRESET}}$ CE	$PR^{00\propto}$		$\cdot PI^{START_1} \cdot PI_2^0$
	$\overline{\text{PRESET}}$ CE						

Fig. 11-5 DFA MEMORY SELECTOR CONTROL FLIP-FLOP

FIG 11-

HA 9-13-60



OPERAND AND DEFERRED ADDRESS
MEMORY ADDRESS SELECTOR
CONTROL LEVEL

Fig 11-6
HM 8-31-60

SD67716

QKA FF LOGIC							
PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
		TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
1 → QKA	α			$\overline{\text{PRESET}}_{CE}$	$QK^{00\alpha}$		$\cdot QI^{START}$
0 → QKA	α			$\overline{\text{PRESET}}_{CE}$	$QK^{00\alpha}$		$\cdot \overline{QI^{START}}$
	PRESET_{CE}						

Fig. 11-7 QKA MEMORY SELECTOR CONTROL FLIP-FLOP

FIG 11-

#M 9-13-60

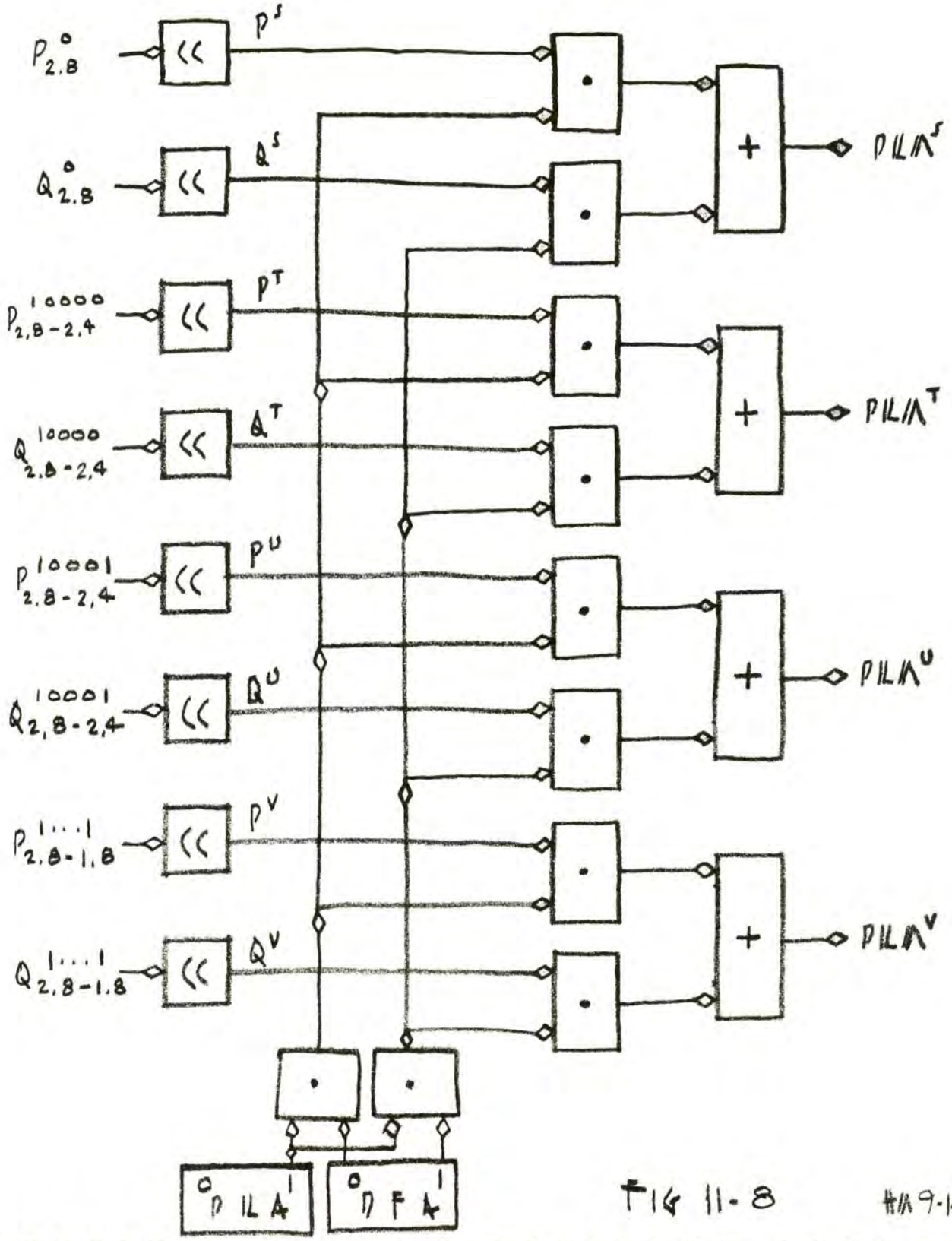
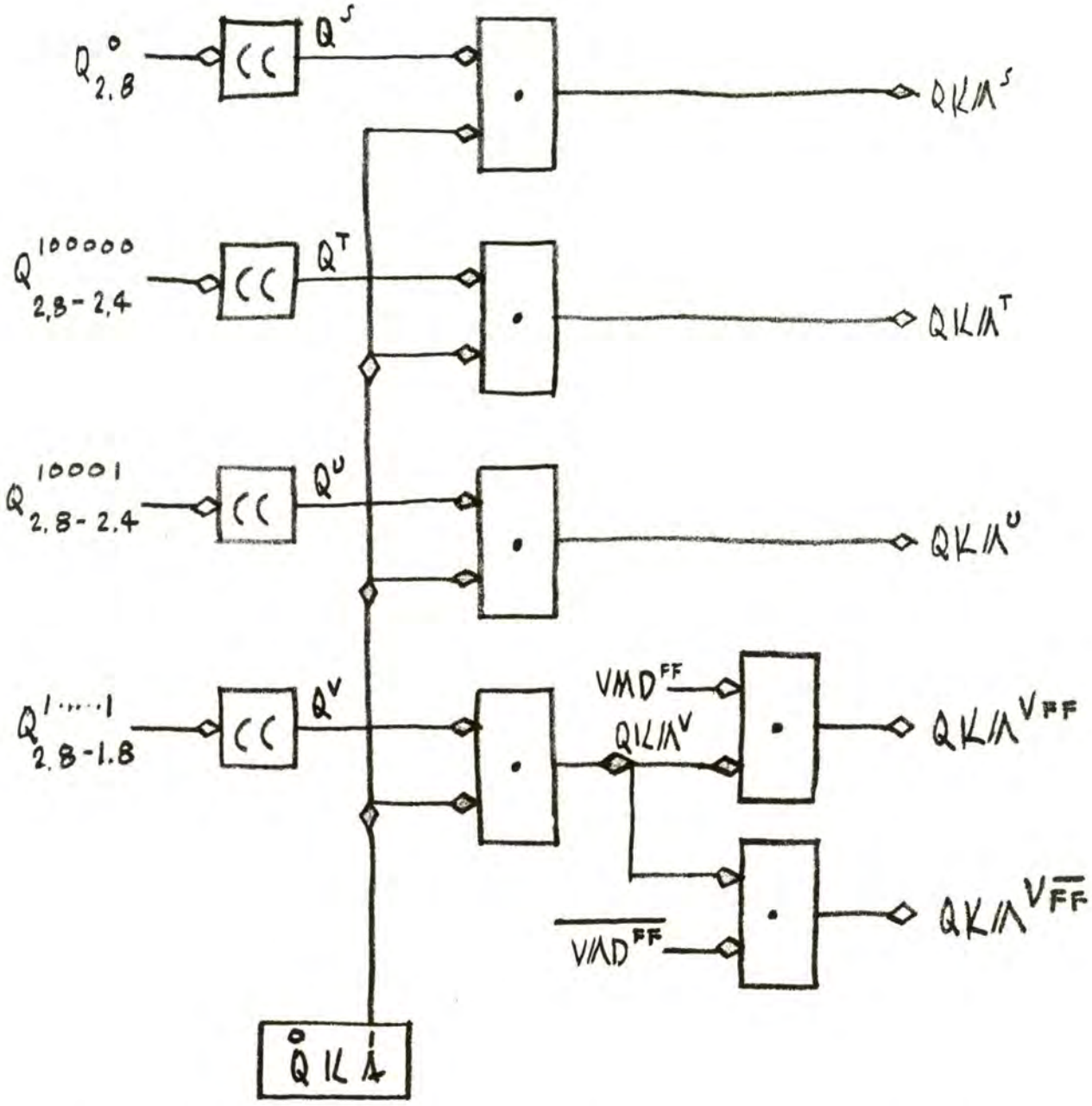


FIG 11-8 HM 9-14-62

INSTRUCTION AND DEFERRED ADDRESS READ-WRITE, STROBE AND INHIBIT SELECTOR LEVEL CONTROL

SD67716

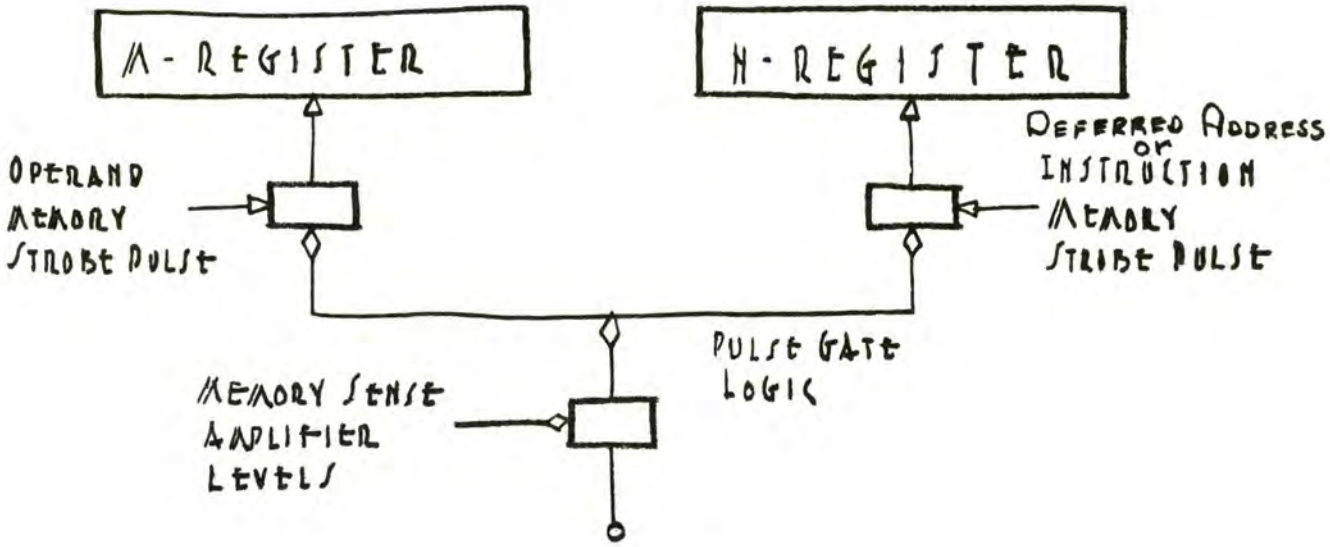


OPERAND, READ-WRITE, STROBE AND INHIBIT SELECTOR LEVEL CONTROL

FIG 11-9

AM 9-14-60

SD67716



MEMORY STROBE INTO M AND N REGISTERS FOR S, T, U AND V $\overline{FF}$ MEMORIES

Fig 11-10

MM 9-8-60

N-REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC					PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$SM_{P,1,2} \xrightarrow{1} N_{P,1,2}$ $SM_{M3,4} \xrightarrow{1} N_{M3,4}$ $SM_3 \xrightarrow{0} N_3$	β	$PK^{10\beta}$	$\cdot PKM^S$					$\cdot SSA^1_{1,2}$ $\cdot SSA^1_{3,4}$ $\cdot SSA^0_3$
$TM_{P1,2} \xrightarrow{1} N_{P1,2}$ $TM_{M3,4} \xrightarrow{1} N_{M3,4}$ $TM_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha}$	$\cdot PKM^T$					$\cdot TSA^1_{P,1,2}$ $\cdot TSA^1_{M3,4}$ $\cdot TSA^0_3$
$UM_{P1,2} \xrightarrow{1} N_{P1,2}$ $UM_{M3,4} \xrightarrow{1} N_{M3,4}$ $UM_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha}$	$\cdot PKM^U$					$\cdot USA^1_{P,1,2}$ $\cdot USA^1_{M3,4}$ $\cdot USA^0_3$
$VM_{1,2} \xrightarrow{1} N_{1,2}$ $VM_{3,4} \xrightarrow{1} N_{3,4}$ $YM_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha}$	$\cdot PKM^{V\overline{F}}$					$\cdot VSA^1_{1,2}$ $\cdot VSA^1_{3,4}$ $\cdot VSA^0_3$
$L \xrightarrow{0} N_{1,2}$ $L \xrightarrow{0} N_4$	α	$PK^{10\alpha}$ $PK^{25\alpha}$ $QK^{01\alpha}$ $CSK^{01\alpha}$	$\cdot$ $\cdot$ $\cdot$	$\cdot$ $\cdot PKIR^{JX}$ $\cdot QKIR^{IX}$	$\cdot\left(PKM^{LEGAL} + PI^1_2 \cdot DI^0_5\right)$ $\cdot\left(EB^0 + \overline{XJ}\right)$			

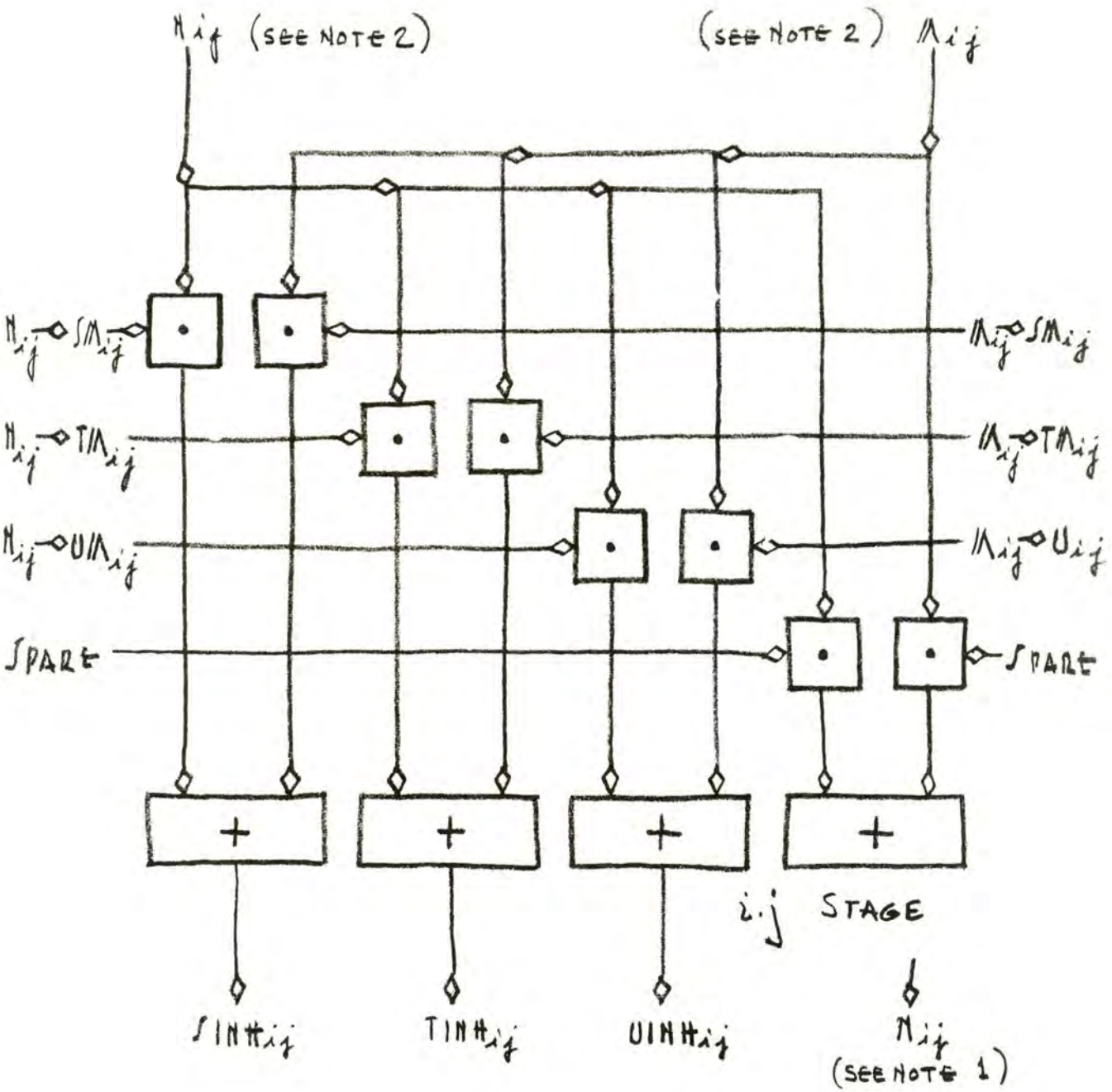
FIG. 11-11 MEMORY CLEAR AND STROBE INTO N-REGISTER

FIG 11-
HM 9-8-60

PULSE	REGISTER DRIVER LOGIC					PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$SM_1 \xrightarrow{1} M_1$ $SM_2 \xrightarrow{1} M_2$ $SM_3 \xrightarrow{1} M_3$ $SM_4 \xrightarrow{1} M_4$	β	$QK^{10\beta} \cdot QKM^S$						SSA_1 SSA_2 SSA_3 SSA_4
$TM_{1,2} \xrightarrow{1} M_{1,2}$ $TM_{3,4} \xrightarrow{1} M_{3,4}$	β	$QK^{11\beta} \cdot QKM^T$						$TSA_{1,2}$ $TSA_{3,4}$
$UM_{1,2} \xrightarrow{1} M_{1,2}$ $UM_{3,4} \xrightarrow{1} M_{3,4}$	β	$QK^{11\beta} \cdot QKM^U$						$USA_{1,2}$ $USA_{3,4}$
$VM_{1,2} \xrightarrow{1} M_{1,2}$ $VM_{3,4} \xrightarrow{1} M_{3,4}$	β	$QK^{11\beta} \cdot QKM^{\overline{VFF}}$						$VSA_{1,2}$ $VSA_{3,4}$
$LO \rightarrow M_{1,2,3,4}$	α	$QK^{09\alpha} \cdot \overline{QKM^{VFF}}$ $QK^{18\alpha}$ · $QKIR^{SKM} \cdot PKIR_{CH_3'}$ $QK^{18\alpha}$ · $QKIR^{TSD} \cdot IOCM^{ASS'Y}$						

FIG. 11-12 MEMORY CLEAR AND STROBE INTO M-REGISTER

FIG 11-
#M 9-13-60

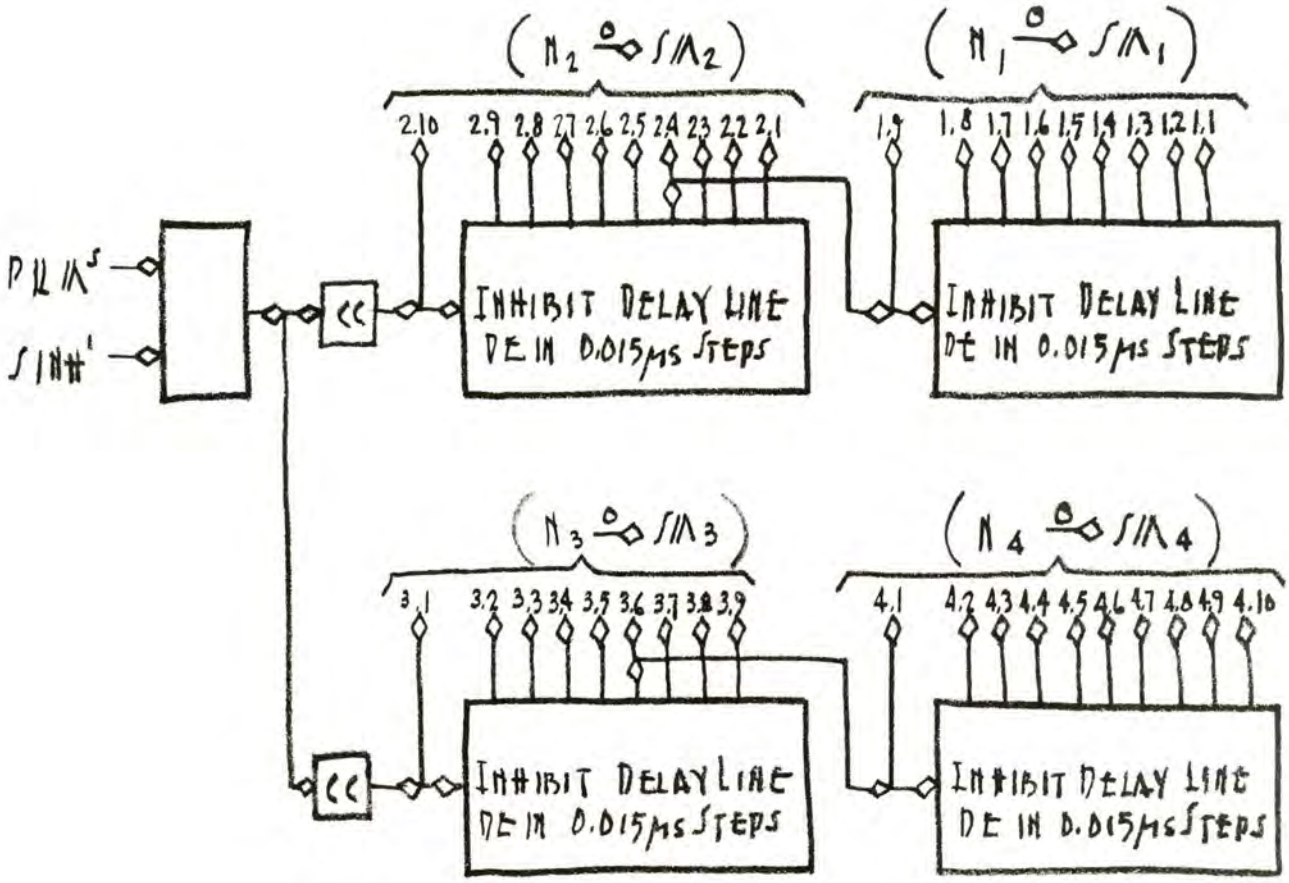


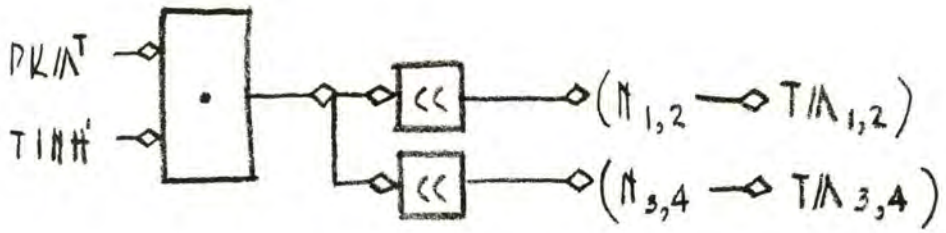
NOTES

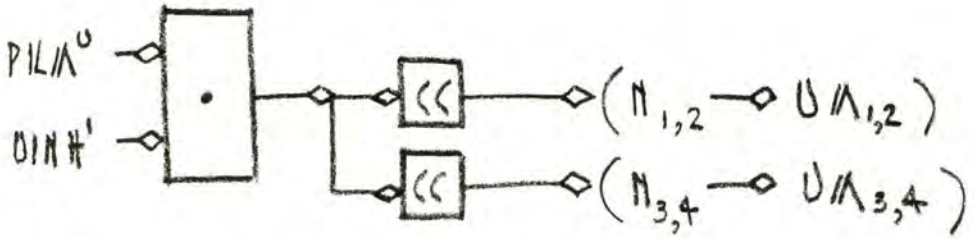
1. M_{ij} LEVEL
2. META BIT IS OMITTED

TYPICAL MEMORY DIGIT INHIBIT LOGIC STAGE

FIG 11-13 MM 9-8-60

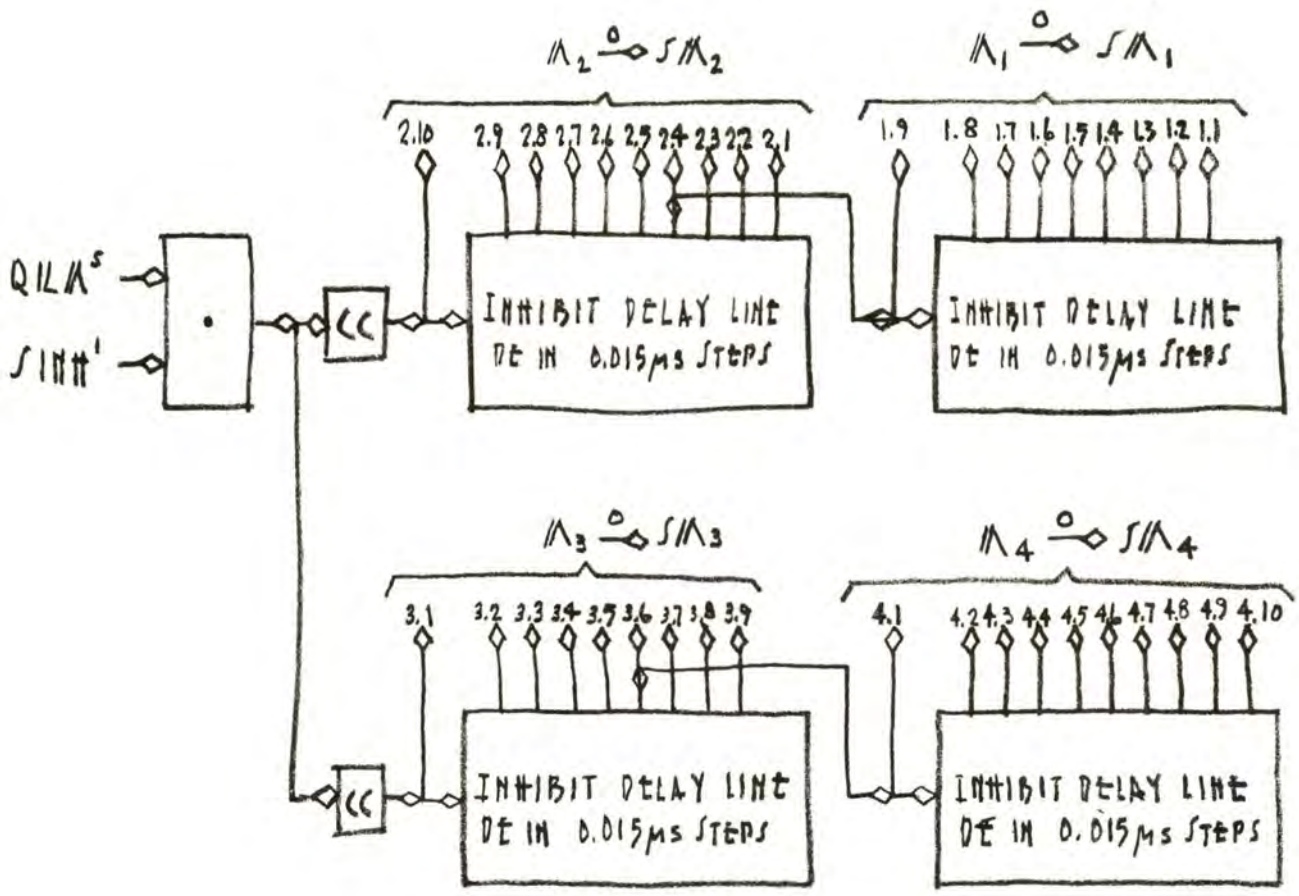


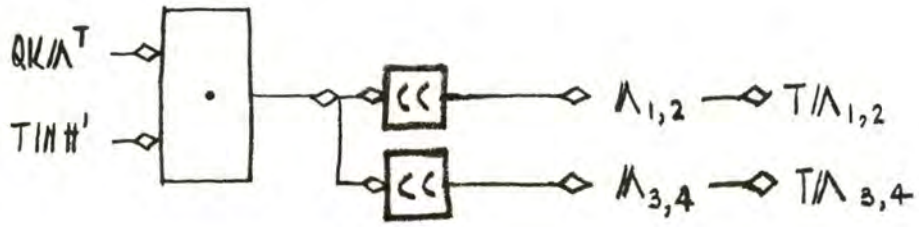


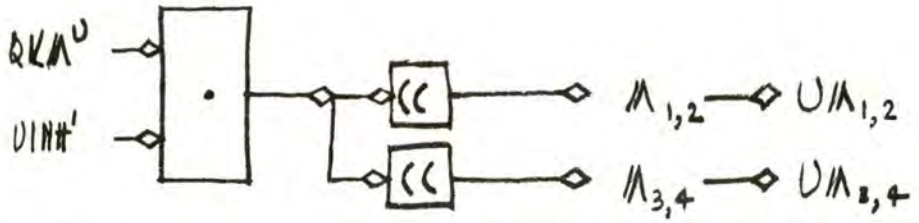


S, T & U MEMORY INHIBIT SELECTOR CONTROL FOR INSTRUCTION WORD REWRITE

SD 67704 FIG 11-14 HA 9-6-60

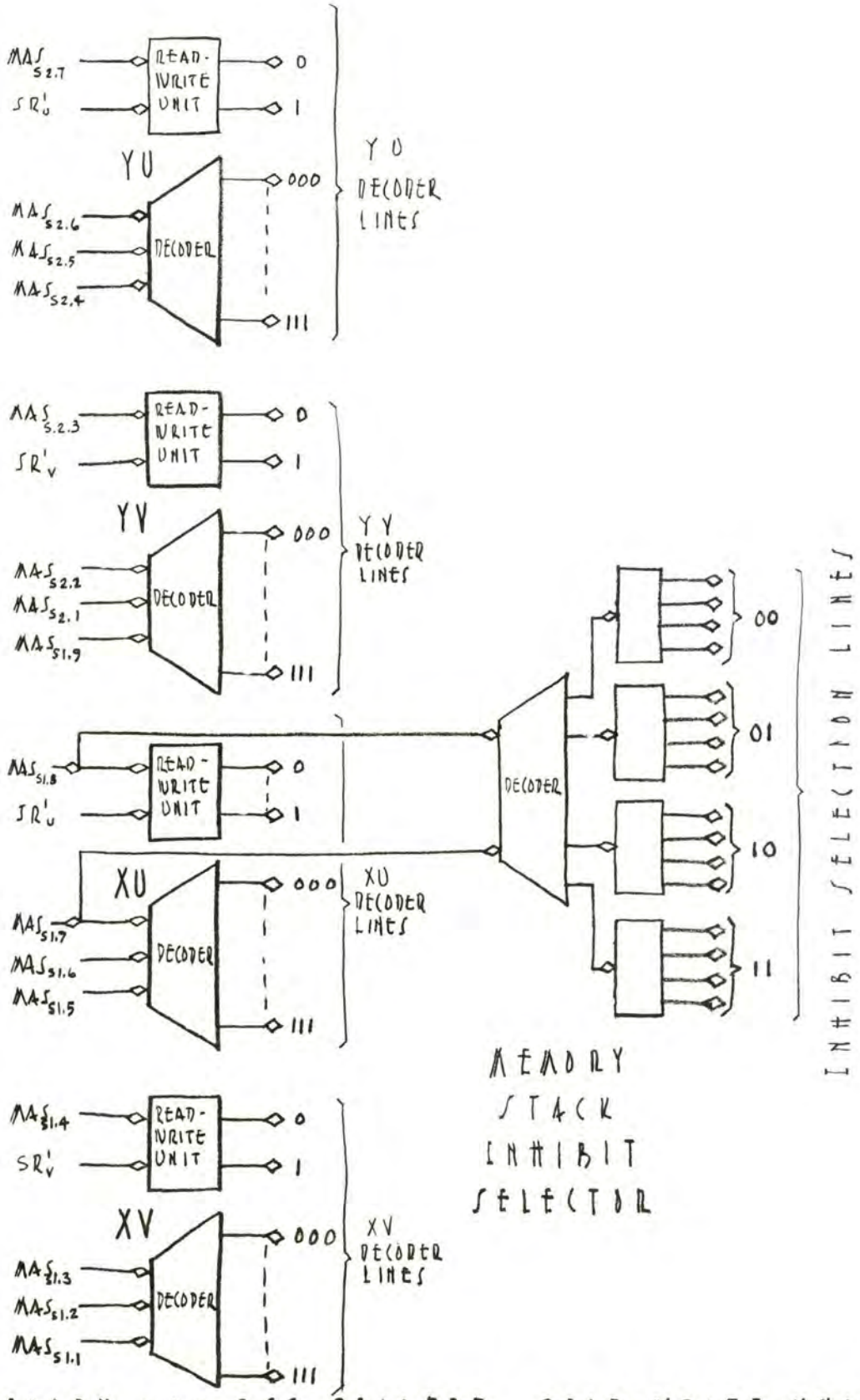






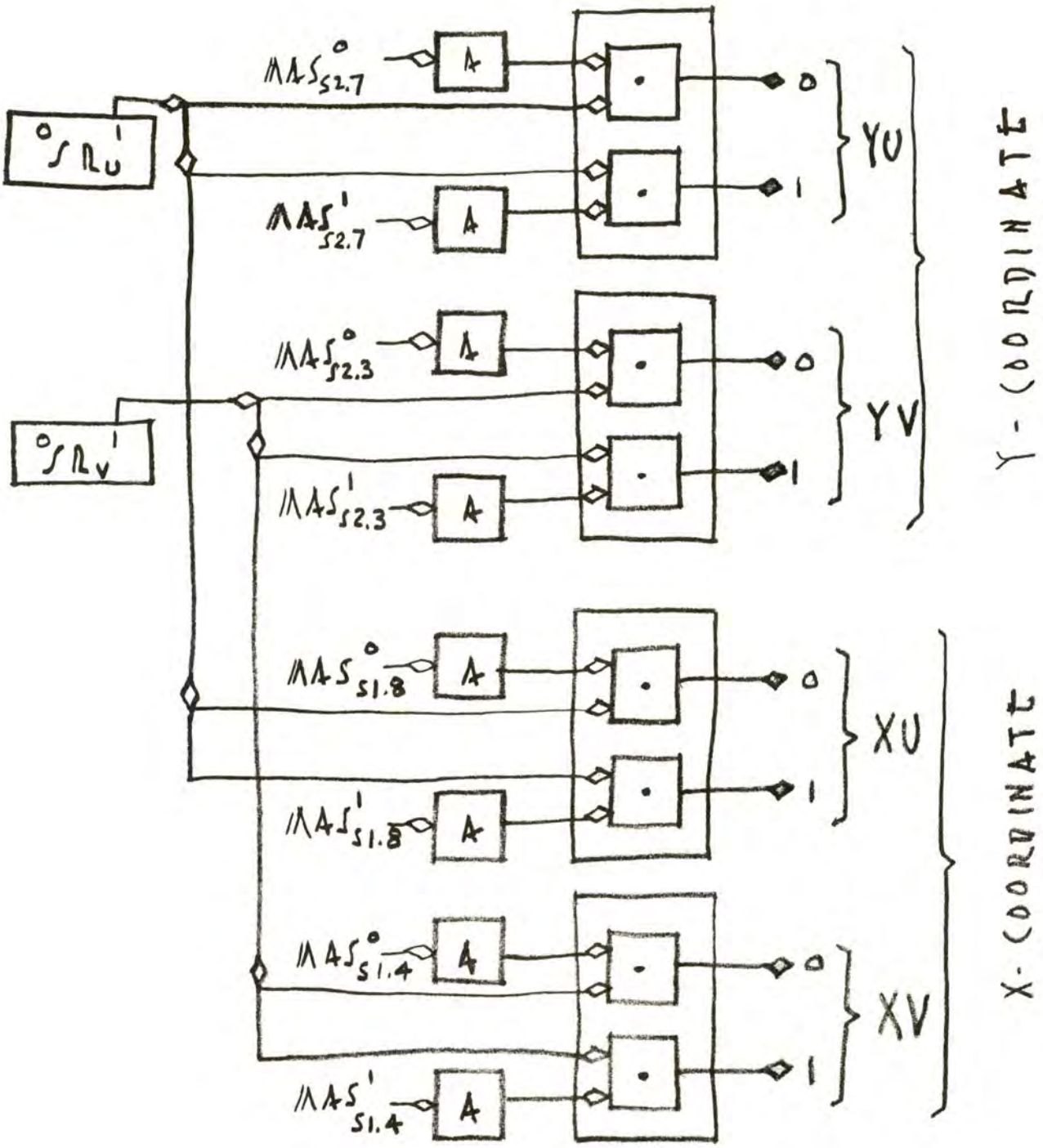
S, T & U MEMORY INHIBIT SELECTOR
CONTROL FOR OPERAND WORD REWRITE

SD 67704 FIG 11-15 HM 9-6-60



S-MEMORY ADDRESS DECODER, READ-WRITE UNIT AND STACK INHIBIT SELECTOR

FIG 11-16
HM 9-2-60



READ-WRITE UNITS OF S-MEMORY

FIG 11-17 #M9-13-60

SD67689

								SR_0 FF LOGIC
PULSE	REGISTER DRIVER LOGIC				DELAY LOGIC	PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	OTHERS	DELAY	TIME LEVEL	MEMORY	OTHERS
$\rightarrow SR_0$	β	$PK^{03\beta} \cdot PKM^S \cdot SMOFF^0$		$\overline{PRESET}_{CE}$	0.24 μs DE			
		$QK^{03\beta} \cdot QKM^S \cdot SMOFF^0$		$\overline{PRESET}_{CE}$	0.24 μs DE			
$\rightarrow SR_0$	β	$PK^{12\beta} \cdot PKM^S \cdot$			0.16 μs DE			
		$QK^{13\beta} \cdot QKM^S \cdot QKIR^{LOAD}$			0.16 μs DE			
		$QK^{21\beta} \cdot QKM^S \cdot QKIR^{STORE}$			0.16 μs DE			
	$\overline{PRESET}_{CE}$							

Fig. 11-18 S-MEMORY READ-WRITE SR_0 FLIP-FLOP CONTROL LOGIC

FIG 11-
HM 9-2-60

SUv FF LOGIC								
PULSE	REGISTER DRIVER LOGIC				DELAY LOGIC	PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	OTHERS	DELAY	TIME LEVEL	MEMORY	OTHERS
$\llcorner 1 \triangleright SR_v$	β	$PK^{02\beta} \cdot PKM^{s} \cdot SMOFF^{0} \cdot \overline{PRESET}_{CE}$ $QK^{02\beta} \cdot QKM^{s} \cdot SMOFF^{0} \cdot \overline{PRESET}_{CE}$			0.2μs DE 0.2μs DE			
$\llcorner 0 \triangleright SR_v$	α	$PK^{11\alpha} \cdot PKM^{s}$ $QK^{11\alpha} \cdot QKM^{s}$			0.2μs DE 0.2μs DE			
	$\overline{PRESET}_{CE}$							

FIG. 11-19 S-MEMORY READ-WRITE SR_v FLIP FLOP CONTROL LOGIC

FIG 11-

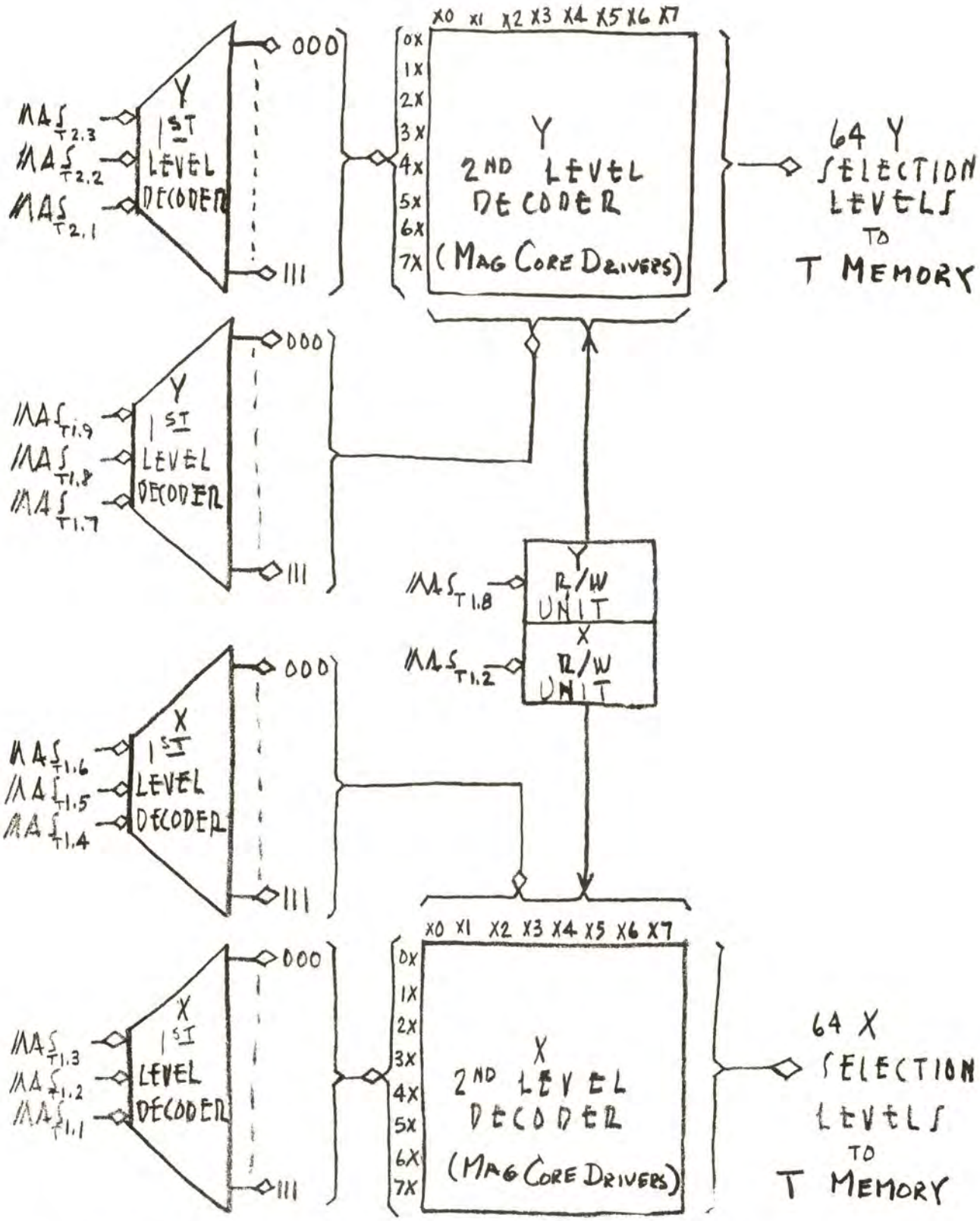
HM 9-2-60

						SINH FF LOGIC		
PULSE	REGISTER DRIVER LOGIC				DELAY LOGIC	PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	OTHERS	DELAY	TIME LEVEL	MEMORY	OTHERS
1 → SINH	α	$PIL^{12\alpha}$ · $PIKM^{S}$ · · $SMOFF^{0}$ · $\overline{PRESET}_{CE}$ $QK^{13\alpha}$ · QKM^{S} · $QKIR^{LOAD}$ · $SMOFF^{0}$ · $\overline{PRESET}_{CE}$ $QK^{21\alpha}$ · QKA^{S} · $QKIR^{STORE}$ · $SMOFF^{0}$ · $\overline{PRESET}_{CE}$						
0 → SINH	α	$PIL^{22\alpha}$ · $PIKM^{S}$ $QK^{31\alpha}$ · QKA^{S}						
	$\overline{PRESET}_{CE}$							

Fig. 11-20 S-MEMORY INHIBIT FLIP-FLOP CONTROL LOGIC

FIG 11-

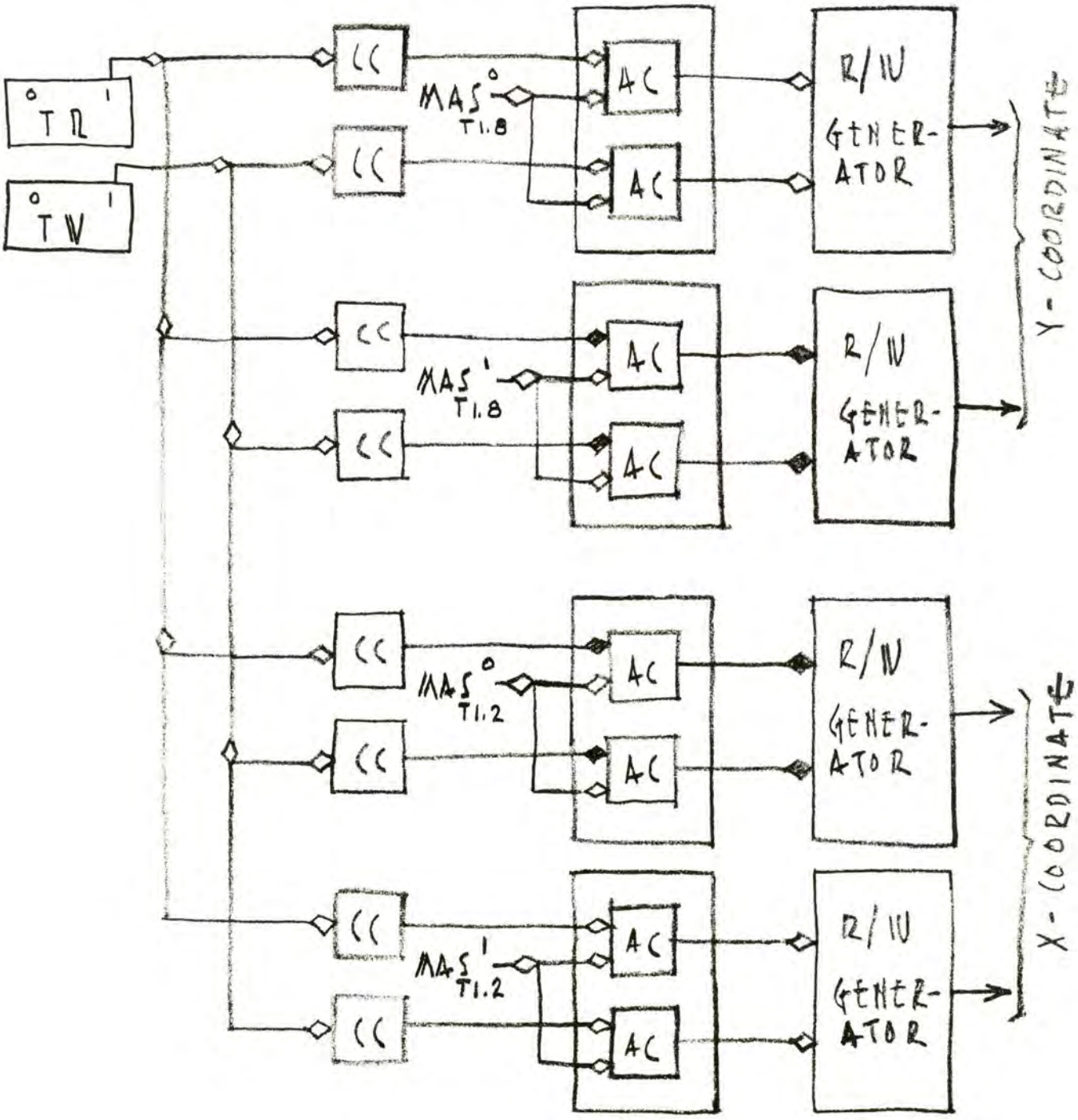
HM 9-2-60



T-MEMORY 1ST AND 2ND LEVEL
X AND Y ADDRESS DECODERS

FIG 11-21

HM 9-1-60



READ-WRITE UNIT OF T-MEMORY

FIG 11-22 HM 9-1-60

TR FF LOGIC								
PULSE	REGISTER DRIVER LOGIC				DELAY LOGIC	PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	OTHERS	DELAY	TIME LEVEL	INSTRUCTION	OTHERS
1 → TR	α	$PL^{01\alpha} \cdot PLM^{T}$		$\cdot \overline{\text{PRESET}}_{CE}$	0.4 μs DE			TMOFF°
		$QL^{01\alpha} \cdot QLM^{T}$		$\cdot \overline{\text{PRESET}}_{CE}$	0.4 μs DE			TMOFF°
0 → TR	α	$PK^{01\alpha} \cdot PKM^{T}$		$\cdot \overline{\text{PRESET}}_{CE}$	1.6 μs DE			
		$QK^{01\alpha} \cdot QKM^{T}$		$\cdot \overline{\text{PRESET}}_{CE}$	1.6 μs DE			
	PRESET → CE							

Fig. 11-23 T-MEMORY READ FLIP-FLOP CONTROL LOGIC

FIG 11-

MM 9-2-60

TW FF LOGIC

PULSE	REGISTER DRIVER LOGIC				DELAY LOGIC	PULSE GATE LOGIC		
	RD PULSE	TIME PULSE	MEMORY	OTHERS	DELAY	TIME PULSE	MEMORY	OTHERS
1 → TW	$\propto$	$PK^{12\propto} \cdot PKM^{T}$		$\cdot \overline{\text{PRESET}_{CE}}$	0.8 μs DE			TMOFF°
		$QK^{13\propto} \cdot QKM^{T}$		$\cdot QKIR^{LOAD} \cdot \overline{\text{PRESET}_{CE}}$	0.8 μs DE			TMOFF°
		$QK^{21\propto} \cdot QKM^{T}$		$\cdot QKIR^{STORE} \cdot \overline{\text{PRESET}_{CE}}$	0.8 μs DE			TMOFF°
0 → TW	$\propto$	$PK^{12\propto} \cdot PKM^{T}$		$\cdot \overline{\text{PRESET}_{CE}}$	2.0 μs DE			
		$QK^{13\propto} \cdot QKM^{T}$		$\cdot QKIR^{LOAD} \cdot \overline{\text{PRESET}_{CE}}$	2.0 μs DE			
		$QK^{21\propto} \cdot QKM^{T}$		$\cdot QKIR^{STORE} \cdot \overline{\text{PRESET}_{CE}}$	2.0 μs DE			
	PRESET CE							

Fig 11-29 T-MEMORY WRITE FLIP FLOP CONTROL LOGIC

FIG 11

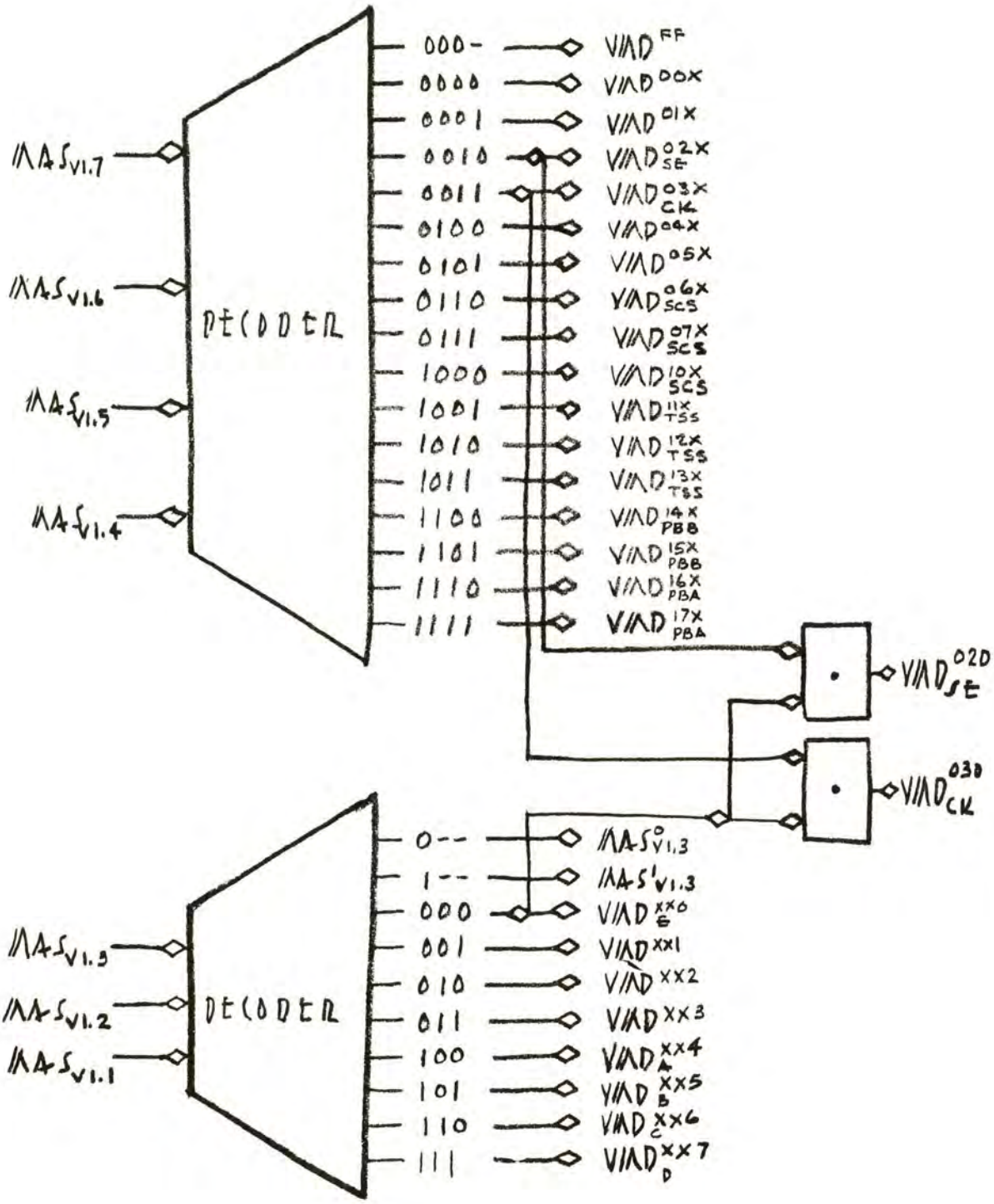
HM 9-2-60

TINH FF LOGIC

PULSE	REGISTER DRIVER LOGIC				DELAY LOGIC	PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	DELAY	TIME LEVEL	INSTRUCTION	OTHERS
1 → TINH	α	$PK^{12\alpha}$ · PKM^{T} ·		· $\overline{\text{PRESET}_{CE}}$	0.4μs DE			TMOFF°
	α	$QK^{13\alpha}$ · QKM^{T} ·		$QKIR^{LOAD}$ · $\overline{\text{PRESET}_{CE}}$	0.4μs DE			TMOFF°
	α	$QK^{21\alpha}$ · QKM^{T} ·		$QKIR^{STORE}$ · $\overline{\text{PRESET}_{CE}}$	0.4μs DE			TMOFF°
0 → TINH	α	$PK^{12\alpha}$ · PKM^{T} ·		· $\overline{\text{PRESET}_{CE}}$	2.0μs DE			
	α	$QK^{13\alpha}$ · QKM^{T} ·		$QKIR^{LOAD}$ · $\overline{\text{PRESET}_{CE}}$	2.0μs DE			
	α	$QK^{21\alpha}$ · QKM^{T} ·		$QKIR^{STORE}$ · $\overline{\text{PRESET}_{CE}}$	2.0μs DE			
	$\overline{\text{PRESET}_{CE}}$							

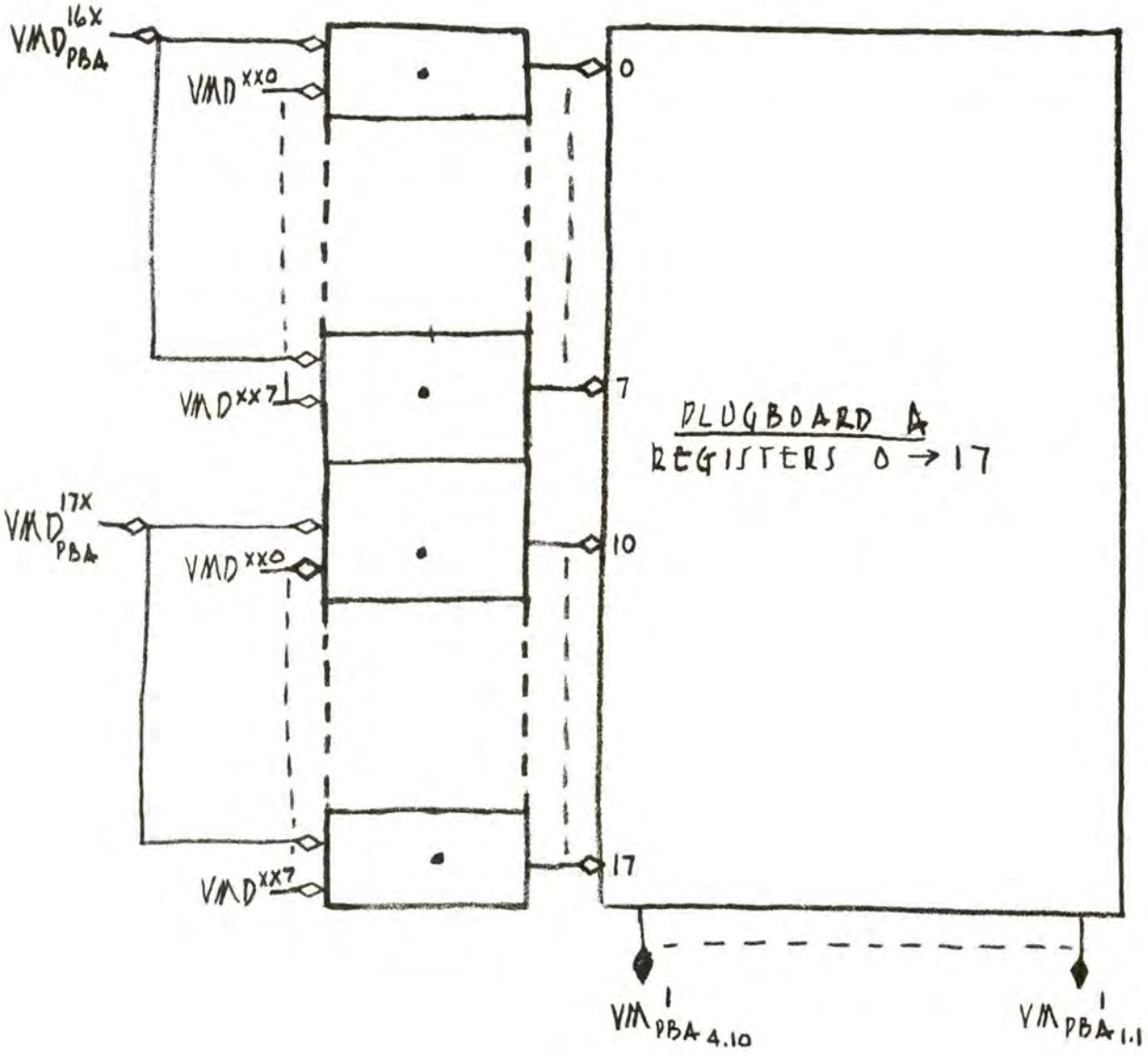
Fig. 11-25 T-MEMORY INHIBIT FLIP FLOP CONTROL LOGIC

FIG 11
HA 9-2-60



V-MEMORY DECODER

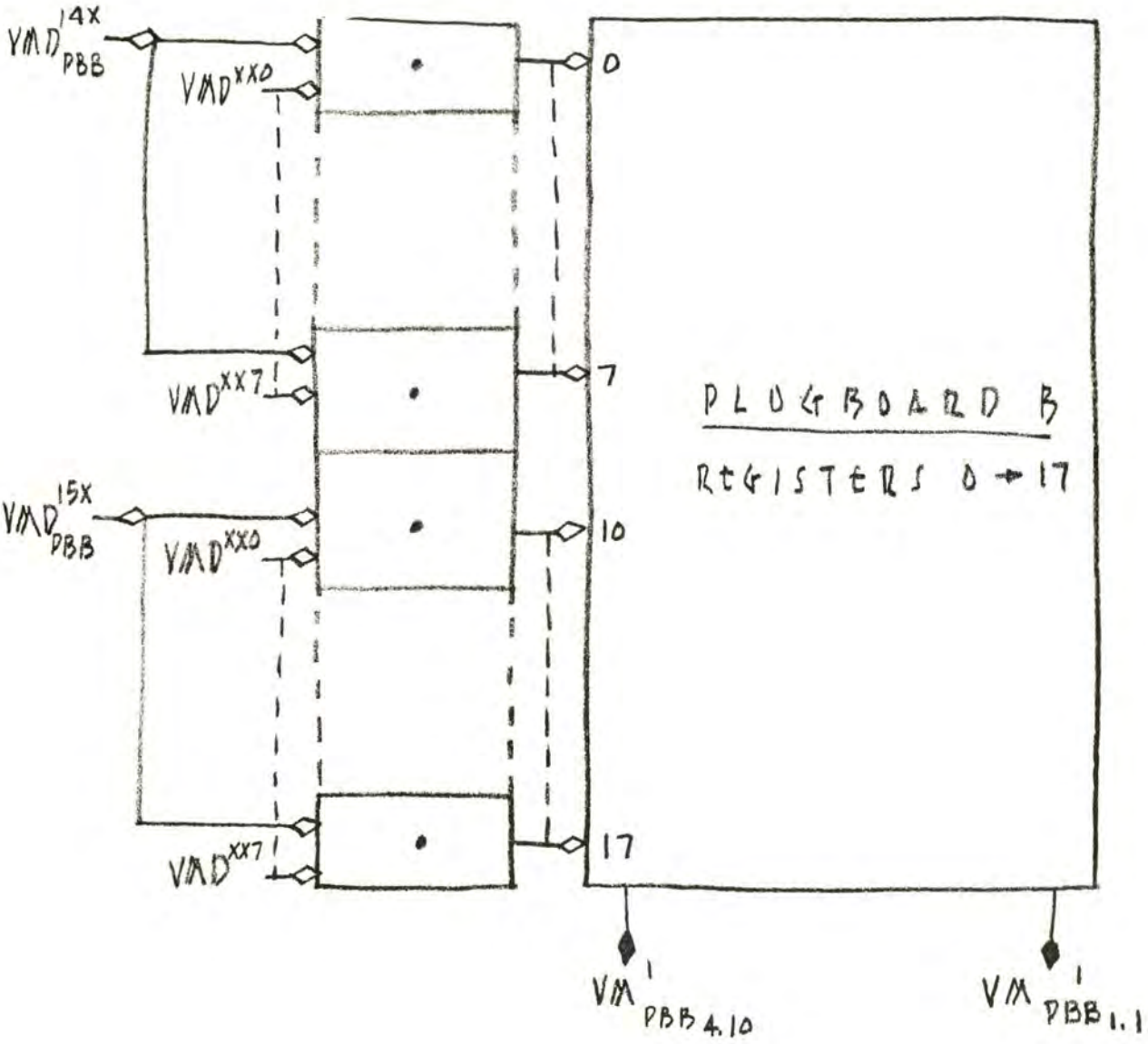
FIG 11-26 #M 9-6-60



PLUGBOARD STORAGE A

FIG 11-27 #M 9-7-60

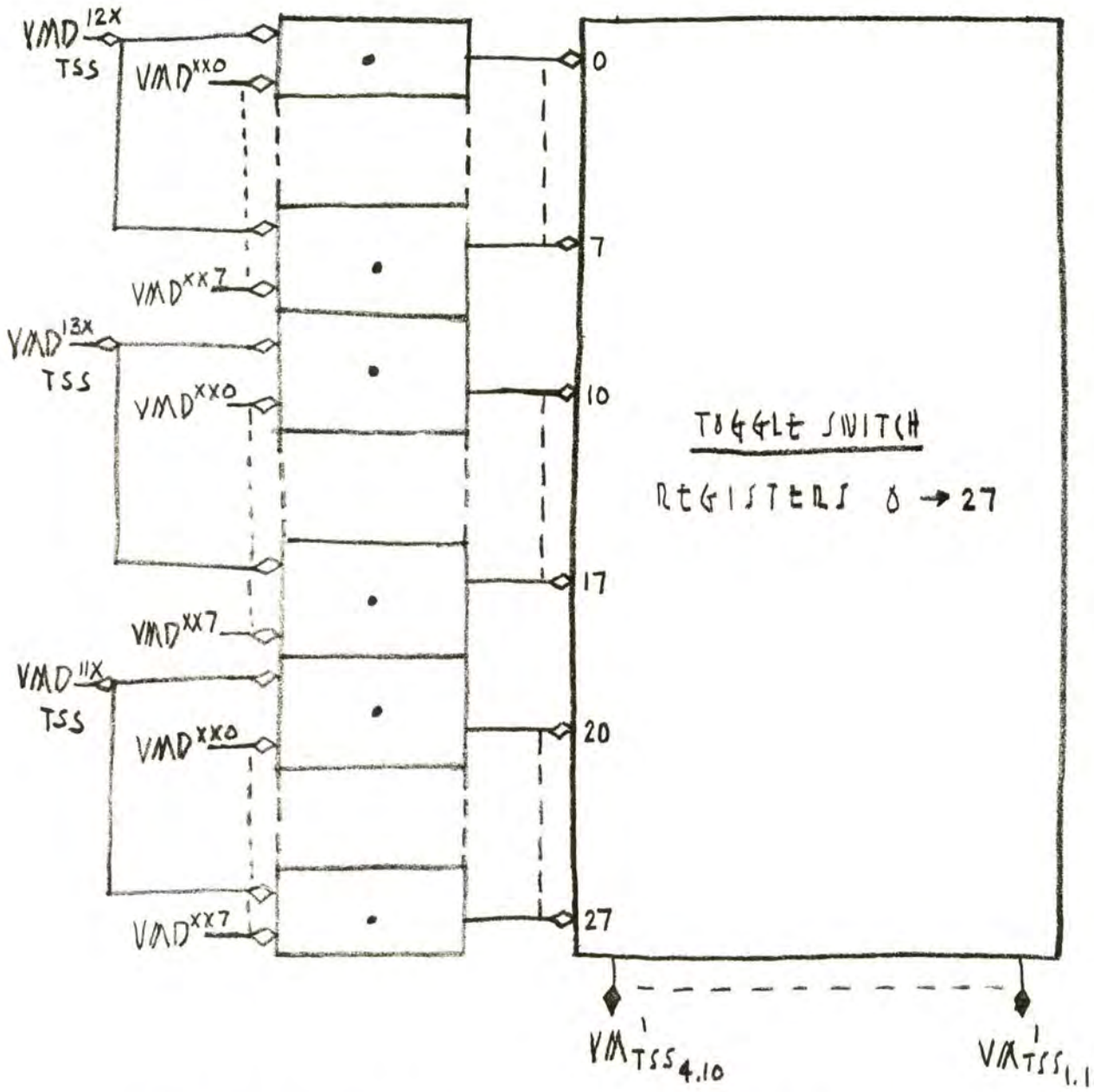
SD87B27



PLUGBOARD STORAGE B

FIG 11-28 HM 9-13-60

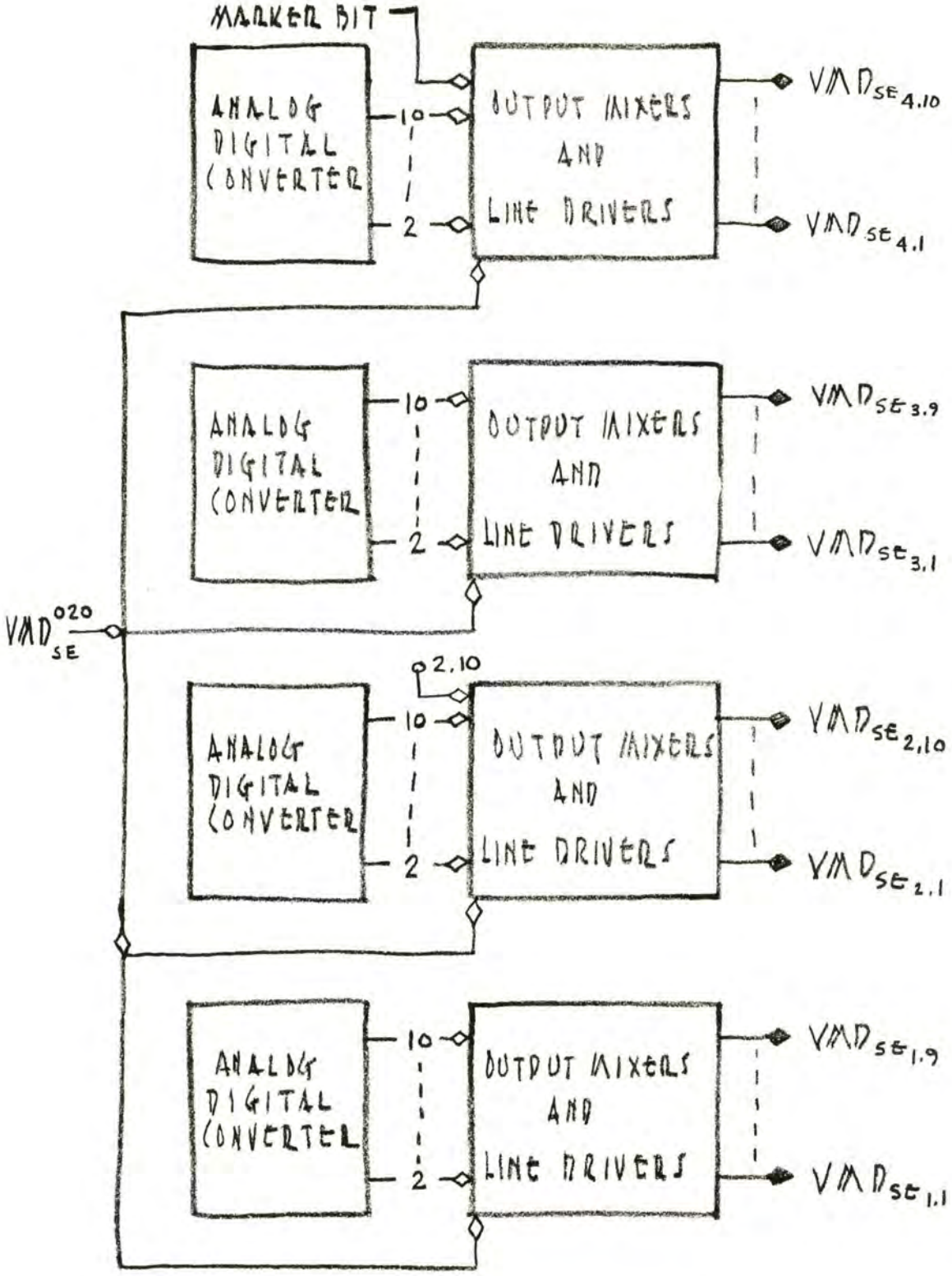
SD87827



TOGGLE SWITCH STORAGE

fig 11-29 HM 9-7-60

SD67720

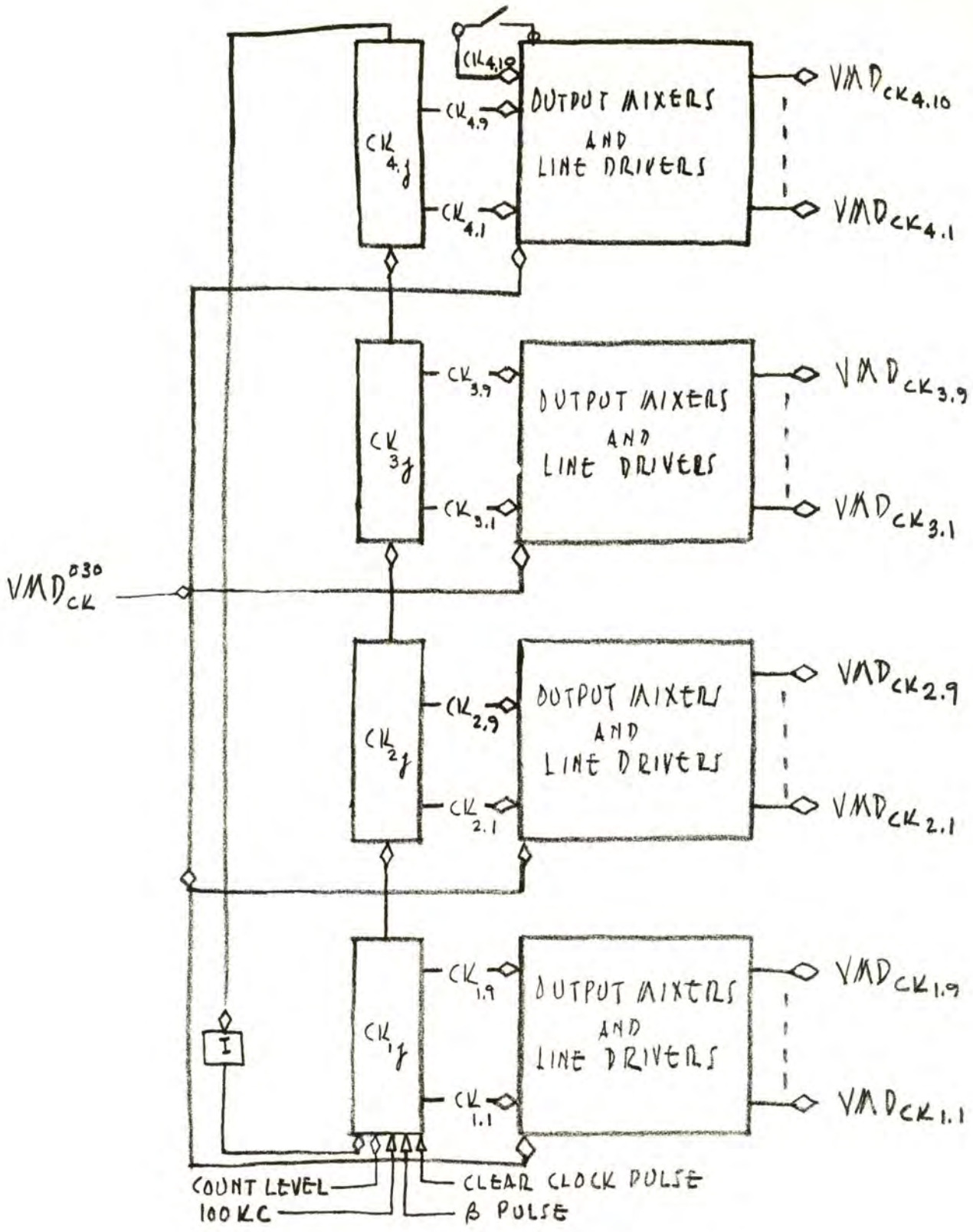


SHAFT ENCODER

Fig 11-30

HM 9-7-60

SD87821
SD87822

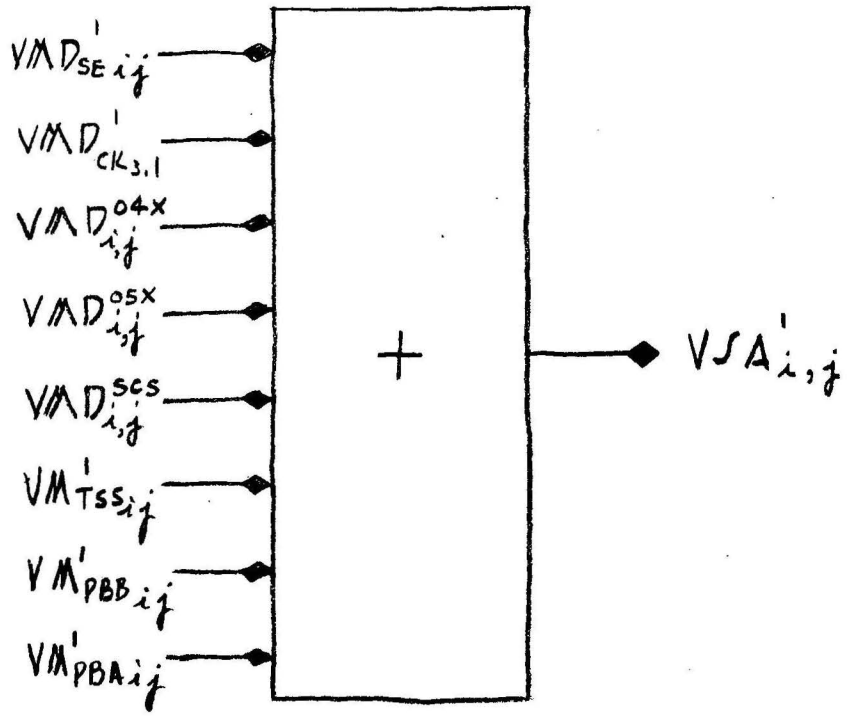


REAL TIME CLOCK

FIG 11-3

HM 9-7-60

SD87823



TOTAL OF 38 STAGES WHERE: $i = (1 \rightarrow 4)$

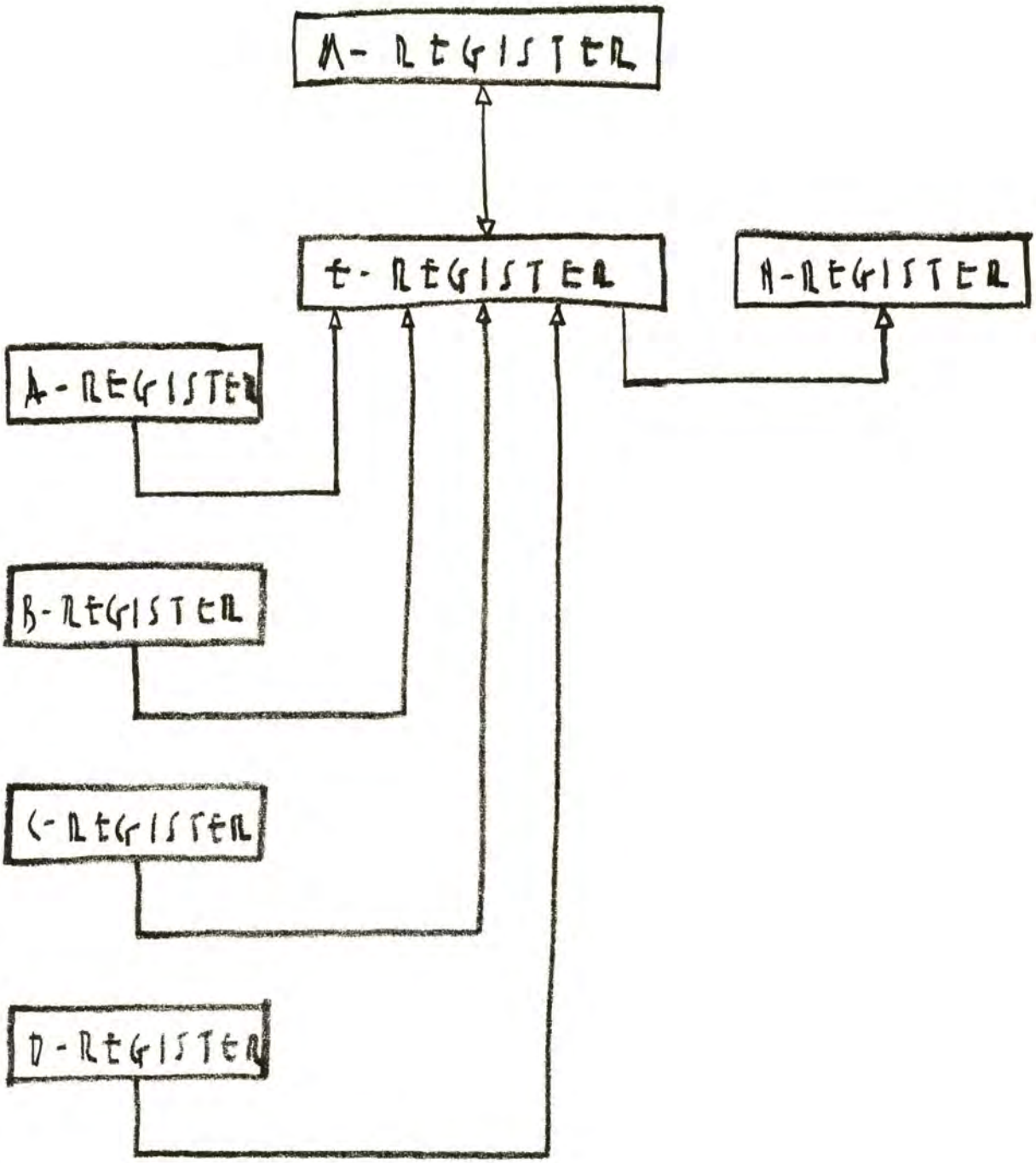
AND $j = (1 \rightarrow 10)$

TYPICAL STAGE OF V-MEMORY INPUT MIXER

FIG 11-32

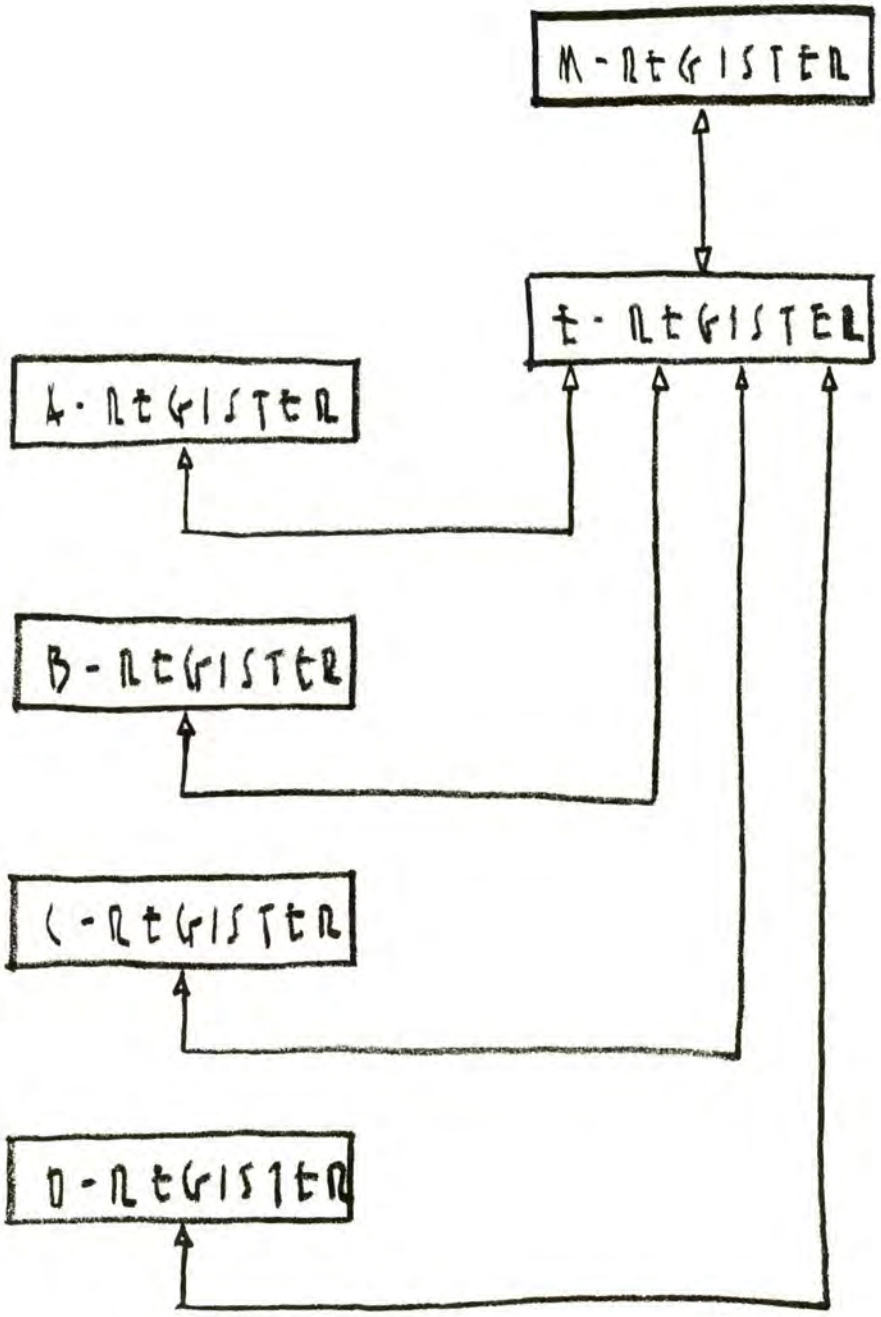
HM 9-7-60

SD87826



VMC-MEMORY INSTRUCTION AND DEFERRED ADDRESS WORD TRANSFER

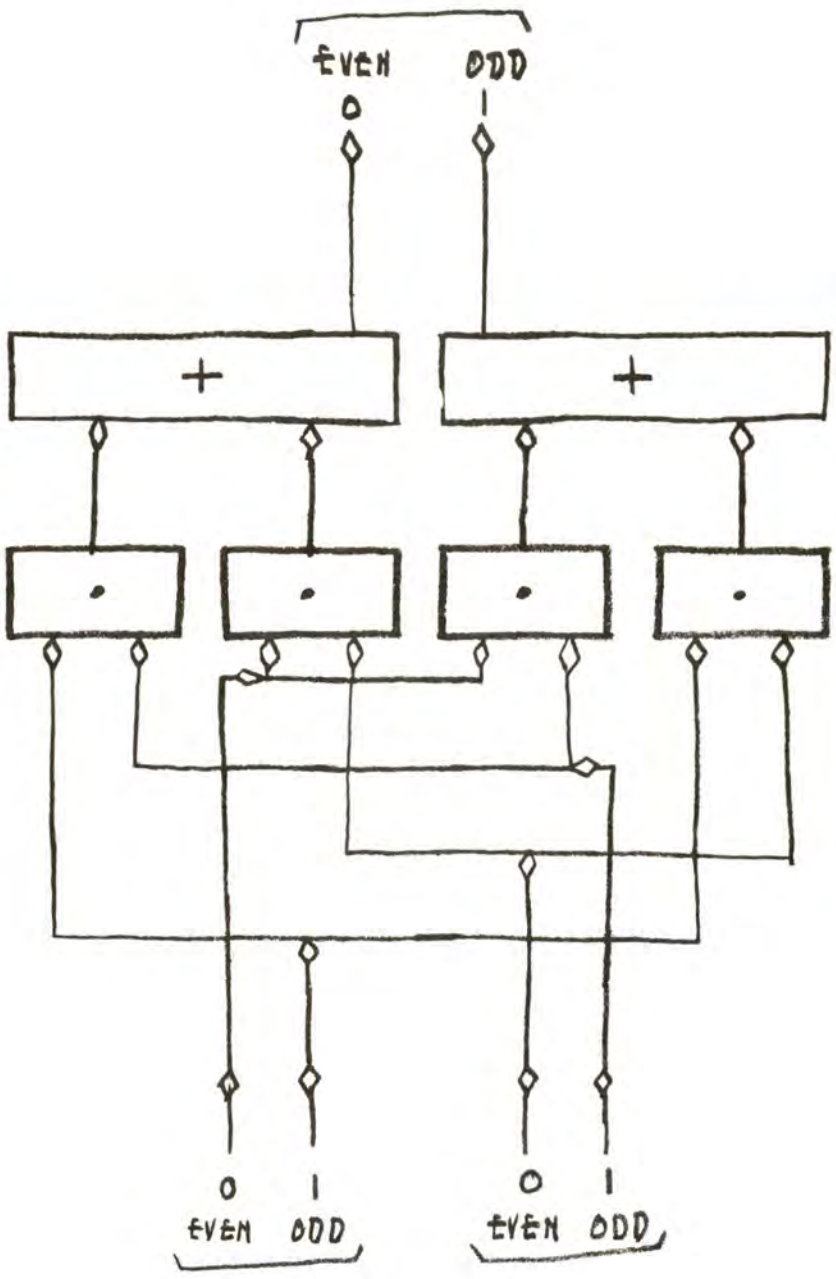
FIG 11-33 MA 10-3-60



VFF-MEMORY OPERAND WORD TRANSFER

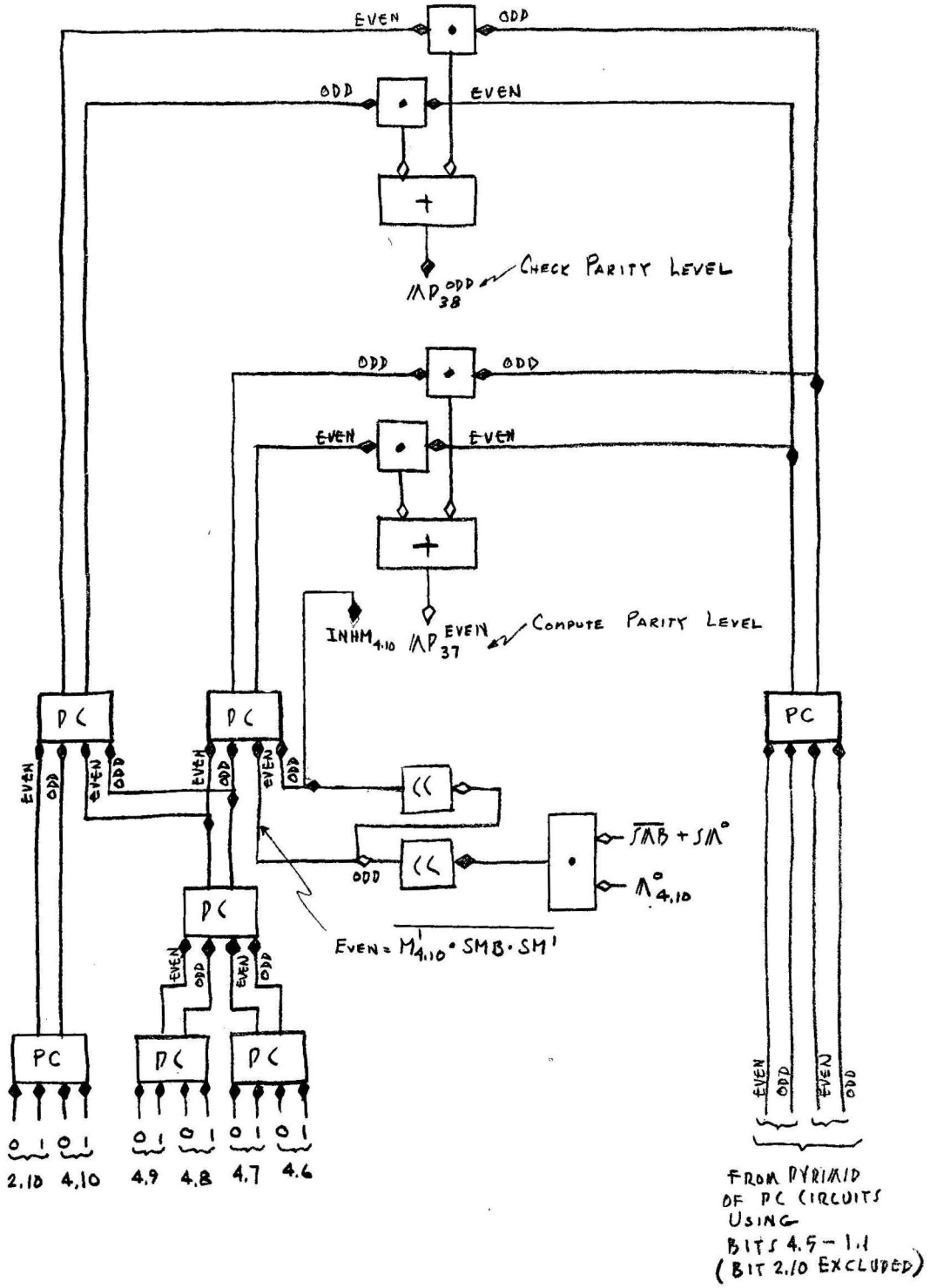
FIG 11-31

HM 10.3.60



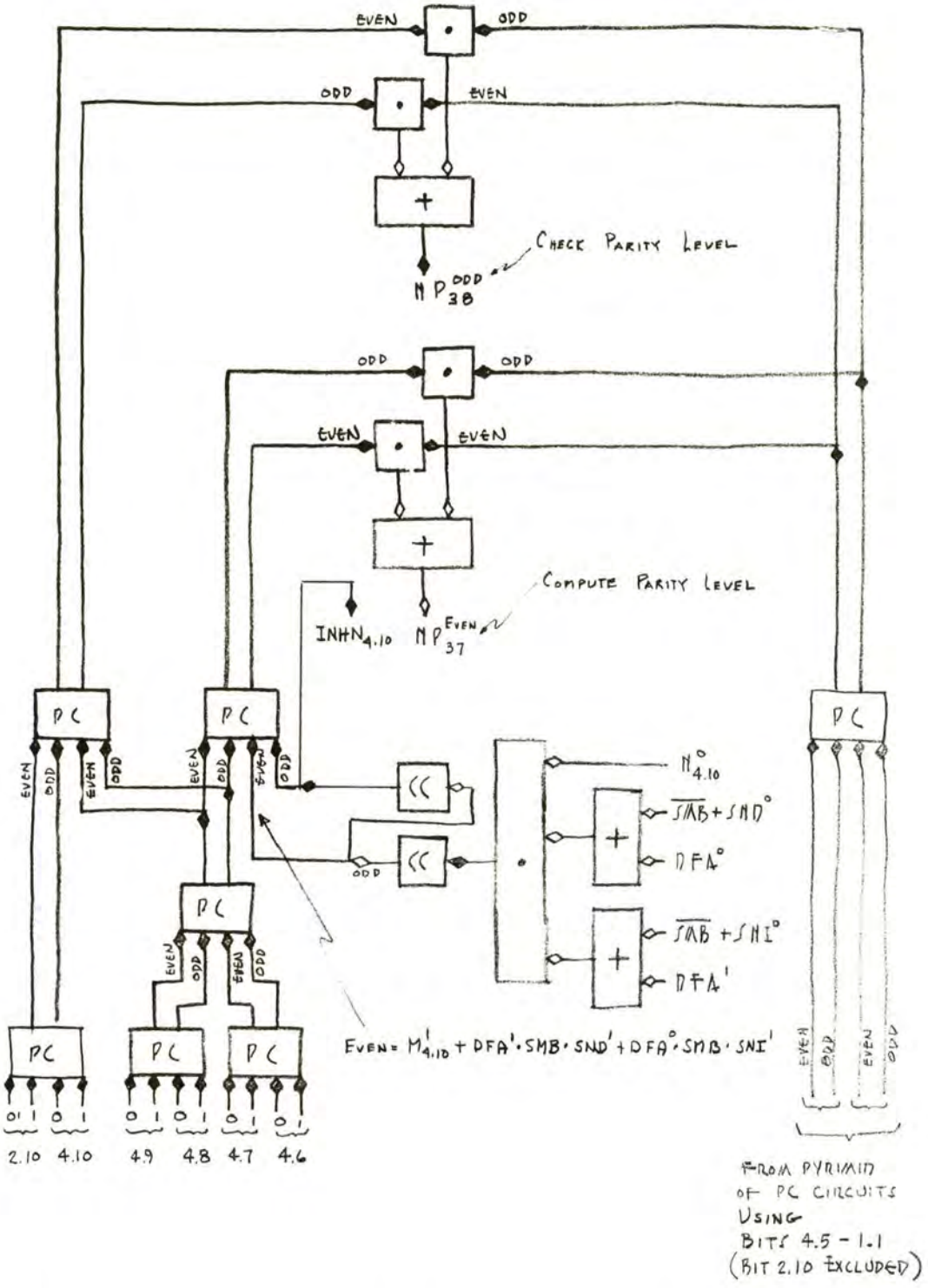
TYPICAL PARITY STAGE

FIG 11-35 #M 9-14-60



M-PARITY COUNT WITH S, T & U MEMORIES

FIG 11-36 MM 9-16-60



N-PARITY COUNT WITH S, T & U MEMORIES

Fig 11-37 #M 9-16-60

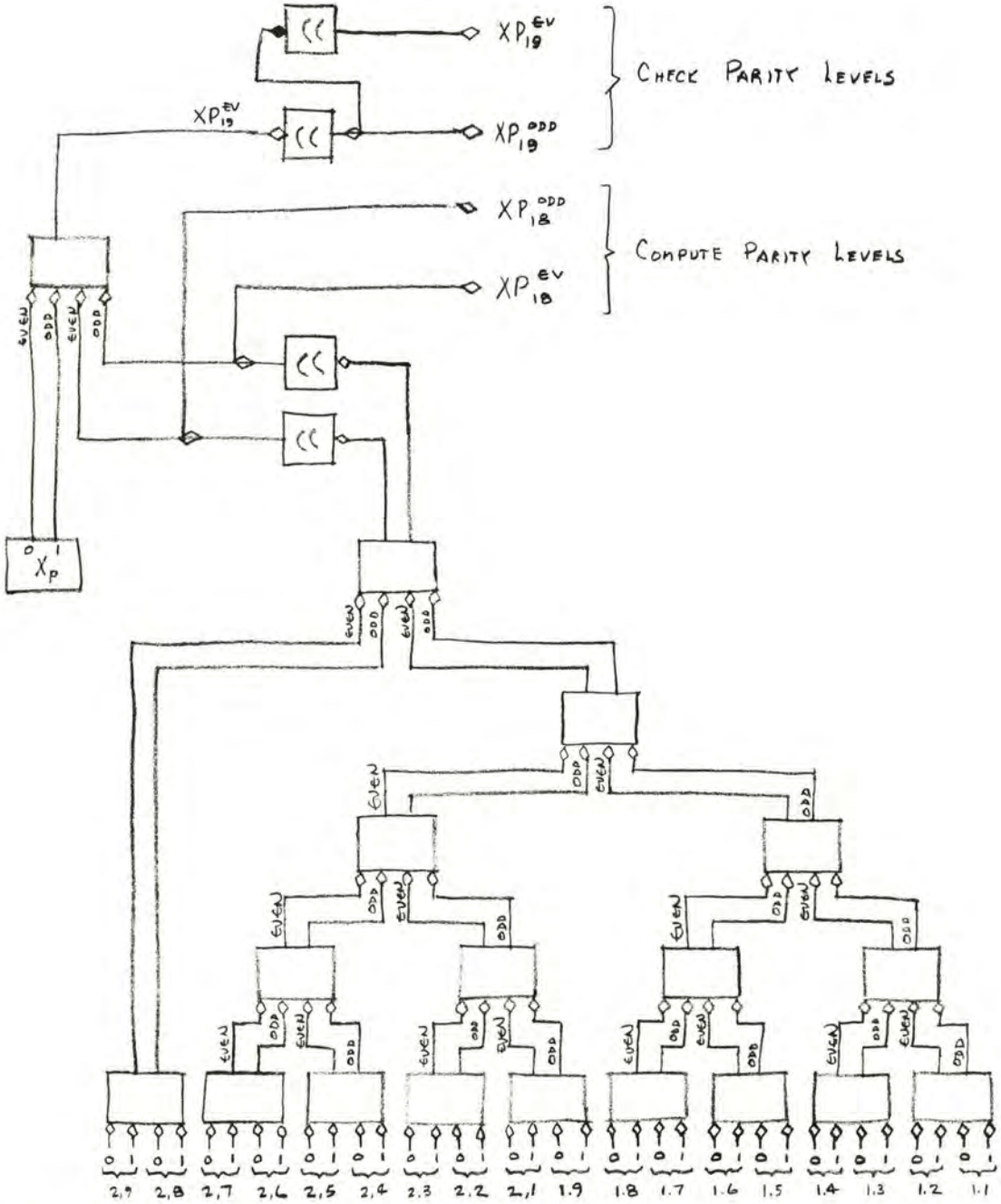


FIG. 11-38X MEMORY PARITY COUNT

FIG 11-
#M 9-16-60

CHAPTER 12

PROGRAM ELEMENT

TABLE OF CONTENTS

12-1 INTRODUCTION

12-2 PROGRAM ELEMENT REGISTER DRIVER LOGIC

12-2.1 GENERAL DESCRIPTION

12-2.2 N REGISTER REGISTER DRIVER LOGIC

12-2.2.1 MEMORY TRANSFERS INTO THE N REGISTER

12-2.2.2 CLEAR N REGISTER LOGIC

12-2.2.3 E REGISTER TRANSFERS INTO THE N REGISTER

12-2.3 P REGISTER REGISTER DRIVER LOGIC

12-2.4 Q REGISTER REGISTER DRIVER LOGIC

12-2.5 K REGISTER REGISTER DRIVER LOGIC

12-2.6 X REGISTER REGISTER DRIVER LOGIC

12-2.6.1 X MEMORY TRANSFERS INTO THE X REGISTER

12-2.6.2 XPS FLIP-FLOP LOGIC

12-2.6.3 P REGISTER TRANSFERS INTO THE X REGISTER

12-2.6.4 X ADDER TRANSFERS INTO THE X REGISTER

12-2.6.5 CLEAR X REGISTER AND SET X PARITY FLIP-FLOP LOGIC

12-2.6.6 COMPLEMENT X REGISTER LOGIC

12-2.7 X ADDER SELECT FLIP-FLOP (XAS) LOGIC

12-2.8 X ADDER CARRY FLIP-FLOP (XAC) LOGIC

12-2.9 OP REGISTERS REGISTER DRIVER LOGIC

12-2.9.1 $PKIR_{OP}$ REGISTER DRIVER LOGIC

12-2.9.2 $QKIR_{OP}$ REGISTER DRIVER LOGIC

12-2.9.3 $AKIR_{OP}$ REGISTER DRIVER LOGIC

12-2.10 CF REGISTERS REGISTER DRIVER LOGIC

12-2.10.1 $PKIR_{CF}$ REGISTER DRIVER LOGIC

12-2.10.2 $QKIR_{CF}$ REGISTER DRIVER LOGIC

12-2.10.3 $AKIR_{CF}$ REGISTER DRIVER LOGIC

12-3 X MEMORY SYSTEM

12-3.1 GENERAL DESCRIPTION

12-3.2 X MEMORY

12-3.2.1 X MEMORY READ LOGIC

12-3.2.2 X MEMORY WRITE LOGIC

12-3.3 X ADDER

12-4 F MEMORY SYSTEM

12-4.1 GENERAL DESCRIPTION

12-4.2 F MEMORY

12-5 OPERATION DECODING PROCESS

12-5.1 GENERAL DESCRIPTION

12-5.2 OP REGISTERS AVAILABILITY FOR DECODING

12-5.3 OP REGISTERS 1ST AND 2ND LEVEL DECODING

12-5.4 CLASS DECODERS
12-5.4.1 $PKIR_{OP}$ CLASS LEVELS
12-5.4.2 $QKIR_{OP}$ CLASS LEVELS
12-5.4.3 $AKIR_{OP}$ CLASS LEVELS
12-6 CONFIGURATION DECODING PROCESS
12-6.1 GENERAL DESCRIPTION
12-6.2 SUBWORD FORM
12-6.3 ACTIVITY
12-6.3.1 EXTENDED ACTIVITY
12-6.4 PERMUTATION
12-6.4.1 PERMUTED ACTIVITY
12-6.5 SIGN EXTENSION
12-7 SEQUENCE SELECTION
12-7.1 GENERAL DESCRIPTION
12-7.2 PRIORITY PATCH PANEL
12-7.3 SEQUENCE SELECTOR
12-7.4 FLAG REGISTER
12-7.5 K DECODER
12-7.6 J CODER
12-7.7 $K^{eq\ JC}$ NET
12-7.8 $K^{eq\ J}$ NET

LIST OF FIGURES

12-1 DIAGRAM OF MEMORY STROBE INTO N REGISTER
12-2 MEMORY STROBE INTO N REGISTER
12-3 N REGISTER CLEARING PULSE
12-4 E REGISTER TRANSFER INTO N REGISTER (3RD QUARTER IS A JAM TRANSFER)
12-5 X ADDER TRANSFER INTO P REGISTER
12-6 X ADDER TRANSFER INTO Q REGISTER
12-7 N REGISTER TRANSFER INTO K REGISTER
12-8 X MEMORY TRANSFER INTO X BUFFER REGISTER
12-9 P REGISTER JAM INTO X BUFFER REGISTER
12-10 X ADDER REGISTER JAM INTO X BUFFER REGISTER
12-11 X PARITY FLIP-FLOP SET PULSE
X BUFFER REGISTER CLEAR PULSE
12-12 X BUFFER REGISTER COMPLEMENT PULSE
12-13 X ADDER SELECT FLIP-FLOP LOGIC
12-14 X ADDER CARRY FLIP-FLOP LOGIC
12-15 N REGISTER TRANSFER INTO $PKIR_{OP}$ REGISTER AND HOLD BIT
12-16 $PKIR_{OP}$ REGISTER TRANSFER INTO $QKIR_{OP}$ REGISTER
12-17 N_2 REGISTER AND $QKIR_{OP}$ REGISTER TRANSFER INTO $AKIR_{OP}$ REGISTER
12-18 N REGISTER TRANSFER INTO $PKIR_{CF}$ REGISTER $PKIR_{CF}$ REGISTER PULSE LOGIC
12-19 CF MEMORY INTO $QKIR_{CF}$ REGISTER
12-20 E_1 REGISTER TRANSFER INTO $QKIR_{CF}$ REGISTER

12-21 N REGISTER TRANSFER INTO $QKIR_{CF}$ REGISTER
12-22 $QKIR_{CF}$ REGISTER CLEARING PULSE
12-23 N_1 REGISTER AND $QKIR_{CF}$ REGISTER TRANSFER INTO $AKIR_{CF}$ REGISTER
12-24 X MEMORY REGISTER SELECTION
12-25 TYPICAL X MEMORY WRITE CIRCUIT STAGE (i,j BIT OF REGISTER 00 ILLUSTRATED)
12-26 TYPICAL X MEMORY READ CIRCUIT STAGE (i,y BIT OF REGISTER 00 ILLUSTRATED)
12-27 X MEMORY READ LOGIC
12-28 X MEMORY WRITE LOGIC
12-29 TYPICAL X MEMORY ADDER STAGES
12-30 DEFER BIT STAGE OF X MEMORY ADDER
12-31 CF MEMORY SYSTEM
12-32 CF MEMORY REGISTER SELECTION
12-33 TYPICAL CF MEMORY WRITE CIRCUIT STAGE
12-34 TYPICAL CF MEMORY READ CIRCUIT STAGE
12-35 OPERATION CODE BLOCK DIAGRAM
12-36 SIMULTANEOUS EXECUTION OF INSTRUCTIONS
12-37 $PKIR_{OP}$ CLASS DECODER LEVEL LOGIC
12-38 $QKIR_{OP}$ CLASS DECODER LEVEL LOGIC
12-39 CONFIGURATION BLOCK DIAGRAM
12-40 SUBWORD FORM
12-41 PARTIALLY ACTIVE SUBWORD (36 BIT SUBWORD WITH ONLY QUARTER 1 ACTIVE)
12-42 EXTENDED ACTIVITY WITH SECOND QUARTER ACTIVE
12-43 QKIR EXTENDED ACTIVITY LOGIC
12-44 QKIR EXTENDED ACTIVITY
12-45 PERMUTATION AS DEFINED BY $QKIR_{CF}$
12-46 QKIR PERMUTED ACTIVITY LOGIC
12-47 SIGN EXTENSION LOGIC AND NETS
12-48 SEQUENCE SELECTOR (FLAG LOGIC OMITTED)
12-49 SEQUENCE SELECTOR STAGE ($H \neq 0$) AND FLAG LOGIC
12-50 SEQUENCE SELECTOR STAGE ($H = 0$) AND FLAG LOGIC
12-51 FLAG REGISTER LOGIC

CHAPTER 12

PROGRAM ELEMENT

12-1 INTRODUCTION

Two basic operations are performed in the Program Element. One operation is the determination of the addresses of instructions, deferred addresses and operands, and the subsequent interpretation of the instruction and deferred address words after they appear in the N register. The second operation is the change of sequence, i.e., the change of program counter in the P register, when indicated by the Sequence Selector.

The results of interpreting an instruction are usually a set of static levels used by the remainder of the computer. Since the Program Element can be interpreting as many as three instructions at once, there can be that many sets of levels about instructions, as well as another set generated about sequence selection.

This chapter begins by discussing the register pulse logic associated with each of the registers in the Program Element. The X and F memory systems are then explained. Next, the decoding process, by means of which the data transferred into the Program Element is interpreted, is discussed. Finally the sequence selection process is examined.

12-2 PROGRAM ELEMENT REGISTER DRIVER LOGIC

12-2.1 GENERAL DESCRIPTION. The following registers or flip-flops in the Program Element are controlled by register drivers: K, P, Q, N, PKIR, QKIR, AKIR, X, X Adder carry flip-flop (XAC), X Adder select flip-flop (XAS), and FLAG. The FLAG register is discussed in Sect. 12-7.4. The X Adder (XA) is treated as a register even though it does not contain any flip-flops and hence does not have any associated register drivers. The functions of the Program Element registers and their paths of communication with other registers in the computer were discussed in Chapter 2. A block diagram of the Program Element was given in Fig. 2-4. This section will discuss the register driver and pulse gate control of each of these registers.

12-2.2 N REGISTER REGISTER DRIVER LOGIC. This logic controls the transfer of information from the main memory sense amplifiers and from the E register into the N register.

12-2.2.1 MEMORY TRANSFERS INTO THE N REGISTER. Fig. 12-1 illustrates how data is transferred from the memory sense amplifiers into either the M or N register. A word in memory may be an instruction, deferred address, or an operand. If the word is an instruction or deferred address, it will be transferred into the N register during a PK cycle by a memory strobe pulse. This strobe pulse is formed from a PK time level and a memory selection level. The memory strobe logic was discussed in Chapter 11, but will be briefly reviewed.

The logic for the N register strobe pulse is shown in Fig. 12-2. An instruction or deferred address word is strobed out of the memory sense amplifiers at $PK^{11\alpha}$. In the case of the S Memory, the strobe pulse is routed through a delay line. Thus even though the pulse is initiated at $PK^{10\beta}$, it is not finished until $PK^{11\beta}$.

12-2.2.2 CLEAR N REGISTER LOGIC. (See Fig. 12-3.) The N register (with the exception of the third quarter) is cleared at $PK^{10\alpha}$ in preparation for receiving a word from memory. The "clear" pulse is not fired unless the selection address is legal (PKM^{LEGAL}) or unless this cycle is the final deferred address cycle. At other times, only quarters 2 and 1 of the N register are cleared. For $PKIR^{JX}$ type instructions, the clear pulse is fired at $PK^{25\alpha}$ when PK need no longer wait in $PK^{25\alpha}$. For $QKIR^{1X}$ type instructions (AUX, RSX, SKX, EXX, ADX, DPX and SKM), the clear pulse is fired at $QK^{01\alpha}$. The clear pulse is also fired at $CSK^{01\alpha}$ during a change of sequence cycle.

The third quarter of the N register is never affected by the clear N register pulses. All transfers into N_3 are jammed. This is done in order to reduce the amount of noise in the X Memory selection lines decoded from the $N_{3.6\ -\ 3.1}$ bits.

12-2.2.3 E REGISTER TRANSFERS INTO THE N REGISTER. The register driver logic for these transfers is shown in Fig. 12-4. During an instruction or deferred address cycle using the V_{FF} Memory, the contents of the E register are transferred into the N register at $PK^{11\alpha}$. During the final deferred address cycle, the final base address is copied from E into N. During various instructions which make use of the X Memory as an operand memory, the contents of $E_{2,1}$ are transferred into $N_{2,1}$. These transfers occur at $QK^{15\alpha}$ or $QK^{21\alpha}$.

12-2.3 P REGISTER REGISTER DRIVER LOGIC. Information can be transferred into the P register only from the X Adder. In addition to this single transfer path, the P register has a counter which can index the contents of the P register by one. Note that count circuit does not alter the contents of $P_{2.9}$.

The contents of both quarters of the X Adder (with the exception of $XA_{2.9}$) are jammed into the P register during jump type instructions or during a change of sequence. The logic is shown in Fig. 12-5. For these transfers to occur, the computer must be in either an AUTO START or $\overline{AL}$ (no alarm) condition.

The XA $\xrightarrow{j}$ P pulse occurs in the following situations:

1) When PK need no longer wait in $PK^{25\alpha}$ for EB^{0} and the index jump condition is satisfied (XJ).
2) During a JMP instruction at $PK^{31\alpha}$. (This logic has a redundant term due to wiring considerations.)
3) Whenever the Arithmetic Element jump condition (AEJ) is satisfied at $PK^{31\alpha}$.
4) At $CSK^{04\alpha}$ when a new program counter is placed in the P register.

$XA_{2.9}$ is copied into $P_{2.9}$ only in case 4, i.e., during a change of sequence. This is the only situation in which the $P_{2.9}$ bit is altered, since the sequence meta-bit must be remembered, with the program counter, when the latter is placed in P.

The indexing circuit on the P register is used to add one to the contents of the P register each time an instruction is read out of memory and executed. This indexing pulse occurs at $PK^{24\alpha}$. During skip type instructions (SKX, SKM and SED), the contents of P are indexed a second time if the skip condition is satisfied.

12-2.4 Q REGISTER REGISTER DRIVER LOGIC. This logic controls the transfer of information from the X Adder into the Q register as shown in Fig. 12-6. Q can hold the address of either a deferred address or an operand.

The content of the X Adder is jam transferred into the Q register at the beginning of the QK operand cycle, i.e., when the QI^{START} interlock level is present and QK is in $QK^{00\alpha}$. This jam transfer also takes place when a deferred address cycle begins, i.e., when the $PI^{START}{}_{2}$ and PI^{1}_{2} interlock levels are present and PK is in $PK^{00\alpha}$.

12-2.5 K REGISTER REGISTER DRIVER LOGIC. This logic controls the jam transfer of information from $N_{3.6 - 3.1}$ into $K_{3.6 - 3.1}$. The transfer occurs during a change of sequence at $CSK^{03\alpha}$ as shown in Fig. 12-7.

12-2.6 X REGISTER REGISTER DRIVER LOGIC. The X register is the X Memory buffer register. The register driver logic controls the transfer of information from the X Memory, X Adder and P register into the X register. This logic also controls the set X parity pulse, clear X pulse and complement X pulse.

12-2.6.1 X MEMORY TRANSFERS INTO THE X REGISTER. The content of an X Memory register is jam-transferred into the X register during either of the two situations shown on Fig. 12-8.

The first situation occurs at $PK^{13\alpha}$ during the read-out of the X Memory register specified by the J bits of an instruction or deferred address word. The second situation occurs during the read-out of a new program counter in a change of sequence. In neither situation is the content of the X Memory register actually placed in the X register if the 00 register is specified.

If the selected X Memory register has the same address as the current program counter (i.e., if the $K^{eq\ J}$ level is present), then XPS^{1} must also be present if the memory read-out is to occur.

12-2.6.2 XPS FLIP-FLOP LOGIC. This flip-flop inhibits the X Memory strobe pulse into X when the register selected has the same address or the current program counter, is not register 0, and this is the first reference to this register since the last sequence change. In this case all the cores of the register are cleared and only "junk" (with a 50-50 chance of a bad parity) would be strobed into X. If XPS^{1}, then a clear pulse is substituted for the strobe pulse.

The flip-flop is set whenever a sequence change occurs, and is cleared the first time thereafter that the program counter register is referenced during a PK cycle (if ever). See Fig. 12-8.

12-2.6.3 P REGISTER TRANSFERS INTO THE X REGISTER. The content of the P register is jam-transferred into the X register in the two situations shown in Fig. 12-9.

The first situation occurs at $PK^{31\alpha}$ during the execution of a $PKIR^{JMP}$ instruction, when the content of the P register is placed in the X Memory. However, the transfer will not take place unless either the toggle switch producing the $XPAL_{SUP}$ level is turned on or the XPAL flip-flop is cleared (indicating that no X parity alarm condition exists).

The second situation occurs during a change of sequence at $CSK^{04\alpha}$. Again, the transfer will not take place unless the XPAL condition is satisfied.

12-2.6.4 X ADDER TRANSFERS INTO THE X REGISTER. The content of the X Adder is jam-transferred into the X register by the logic shown on Fig. 12-10. The path through the X Adder is the only one by which data can be transferred to the X Memory from the Memory Element. The XPAL condition must be satisfied before any of these transfers can take place. These transfers occur only during SKX, JPX, JNX, RSX, EXX and AUX instructions.

12-2.6.5 CLEAR X REGISTER AND SET X PARITY FLIP-FLOP LOGIC. The clear X register pulse serves as a substitute for the X Memory strobe pulse. For this reason, the clear pulse also sets the X parity flip-flop to a ONE. This guarantees that the X register contains a number with the correct parity.

Note that the register driver logic shown on Fig. 12-11 is, aside from the time level gating, the inverse of the strobe pulse logic shown on Fig. 12-8. The clear pulse occurs if either X Memory register 0 is selected or if the register has the same address as the current program counter ($K^{eq\ J}$) and XPS^{1}.

12-2.6.6 COMPLEMENT X REGISTER LOGIC. During a JPX instruction ($PKIR^{0X}_{OP} \cdot PKIR^{X6}_{OP}$), the X register is complemented twice; once at $PK^{15\alpha}$, if PI^{0}_{2}, and again at $PK^{25\alpha}$, if EB^{0}. These complement pulses are the only features which distinguish JPX from JNX.

During a SKX instruction, the X register is complemented at $PK^{15\alpha}$, if PI^{0}_{2} and $PKIR_{CF_3}$ differs from $PKIR_{CF_1}$. The X register is then recomplemented either at $PK^{27\alpha}$ if $PKIR^{1}_{CF_3}$, or at $PK^{31\alpha}$ if $PKIR^{1}_{CF_1}$.

In both of the above cases, a pair of complement pulses are generated which contribute to the computation of the desired result in the X register. The PI^{0}_{2} condition permits the first of the pair of complement pulses to occur only in the PKM cycle just before the PKEI cycle of the instructions.

12-2.7 X ADDER SELECT FLIP-FLOP (XAS) LOGIC. This logic is shown in Fig. 12-13. XAS determines whether the output of the X Adder is the sum of $N_{2,1}$ and X (when XAS^{1}), or is simply the contents of $N_{2,1}$ (when XAS^{0}).

The register driver is controlled only by the PRESET level from the Control Element. The remainder of the logic is pulse gate logic.

XAS is set to ONE at $PK^{14\alpha}$ during PK cycles which call for an indexed base address. This occurs during instruction cycles (when no deferred address cycle is called for), or when the final deferred address cycle is reached ($PKIR^{IND} \cdot PI^{0}_{2}$), or during all intermediate deferred cycles (PI^{1}_{5}). Otherwise XAS is left cleared to ZERO. XAS is always set during JX type instructions at $PK^{26\alpha}$ in order that the increment can be added to the index register.

During AUX, ADX and all $QKIR^{X}$ type instructions, XAS is set at $QK^{10\alpha}$. This pulse is not always necessary, but guarantees that the X Adder generates the desired output. XAS is cleared at $QK^{21\alpha}$ for RSX and EXX, since in these cases the sum is not desired.

During a change of sequence cycle, XAS is set at $CSK^{02\alpha}$ so that the X Adder output is the program counter coming from the X Memory when any index register other than number 00 is selected; and is simply $N_{2,1}$ (which contains the value of TSP), if register 00 is selected.

12-2.8 X ADDER CARRY FLIP-FLOP (XAC) LOGIC. The logic is shown in Fig. 12-14. After the terms to be summed in the X Adder have been placed in X and $N_{2,1}$, setting XAC to ONE will "clear" the carry circuit of the X Adder. The carry logic then insures that the X Adder output will be the correct ONE's complement sum. The pulse gate

logic covers the situation when this is desired. The set pulse occurs at $PK^{14\alpha}$ and $PK^{26\alpha}$ during a SKX; at $PK^{25\alpha}$ during a JX type instruction; at $QK^{01\alpha}$ for all $QKIR^{1X}$ type instructions; and at $QK^{01\alpha}$ and $QK^{10\alpha}$ for AUX and ADX. In the case of the last two instructions, the pulse initiated at $QK^{10\alpha}$ is required since the contents of $N_{2,1}$ are not set up until after $QK^{01\alpha}$. XAC is also set at $CSK^{01\alpha}$.

XAC is automatically cleared 0.4 microsecond after it is set.

While the sum of a base address in $N_{2,1}$ and an index register in X is being formed between PK^{13} and PK^{22}, the X Adder carry circuit is forced into a "set" condition. This causes the sum of an 18 bit number and its 18 bit ONE's complement to be all ZEROS, rather than all ONES, if this sum should be formed. The computed address of an operand, deferred address, or next instruction then becomes the first register of the S Memory (address 0), rather than the last register of the V Memory (address 377 777 (octal)), when, for example, the base address is 000 004 and the index is 777 773. The logic for obtaining this result simply uses the $PK^{13\beta}$ 0.4 microsecond time level to set the X Adder carry circuit at the time that XAC would ordinarily have been used to clear it.

It should also be noted that the $N_{2.9}$ bit is presented as an input to the X Adder only when no deferred address cycles are called for. When PI_2^1, the input to the X Adder from the $N_{2.9}$ position is forced to appear as a ZERO.

12-2.9 OP REGISTERS REGISTER DRIVER LOGIC. The operation registers are $PKIR_{OP}$, $QKIR_{OP}$ and $AKIR_{OP}$. These registers are used during the process of interpreting an operation code.

12-2.9.1 $PKIR_{OP}$ REGISTER DRIVER LOGIC. This logic is shown on Fig. 12-15. The contents of $N_{4.3 - 3.7}$ (OP bits) are jam-transferred into $PKIR_{OP}$ at $PK^{12\alpha}$. Simultaneously the content of $N_{4.9}$ (hold bit) is transferred into $PKIR_H$.

12-2.9.2 $QKIR_{OP}$ REGISTER DRIVER LOGIC. This logic is shown on Fig. 12-16. The content of $PKIR_{OP}$ is jam-transferred into $QKIR_{OP}$ at $QK^{00\alpha}$ when the QI^{START} interlock permits the QK counter to start.

12-2.9.3 $AKIR_{OP}$ REGISTER DRIVER LOGIC. This logic is shown on Fig. 12-17. There are two paths over which information can be jammed into the $AKIR_{OP}$ register. The first path is from the N register. During an AOP instruction, the six bits in $N_{2.6 - 2.1}$ are jam-transferred into $AKIR_{OP}$ at $PK^{25\alpha}$ (providing that AK is in $AK^{00\alpha}$ at this time). The second path is from the $QKIR_{OP}$ register. The content of $QKIR_{OP}$ is jam-transferred into $AKIR_{OP}$ at $QK^{13\alpha}$ during $QKIR^{AK}$ type instructions.

12-2.10 CF REGISTERS REGISTER DRIVER LOGIC. The configuration registers are $PKIR_{CF}$, $QKIR_{CF}$ and $AKIR_{CF}$. These are registers used during the process of interpreting the CF bits in an instruction and the configuration word selected.

12-2.10.1 $PKIR_{CF}$ REGISTER DRIVER LOGIC. This logic is shown on Fig. 12-18. The contents of the five CF bits $N_{4.8 - 4.4}$ are jam transferred into $PKIR_{CF}$ at $PK^{13\alpha}$ of the instruction word cycle.

A counting circuit is incorporated in the $PKIR_{CF}$ register that allows instructions which make use of more than one F Memory register to address four successive F Memory registers. These instructions are SPG and FLG, as indicated by the $PKIR^{FF}$ class level ("FF" denotes "Four conFigurations").

The F Memory "master" pulse is generated when FK is in its resting state ($FK^{0\alpha} \cdot FK8^{0}$) and the START → FK level occurs. The master pulse occurs thereafter at 0.8 microsecond intervals at $FK^{2\alpha}$, $FK^{4\alpha}$ and $FK^{6\alpha}$ (i.e., when $\overline{FK^{0\alpha}} \cdot FK^{0}_{\alpha.1}$) if certain further conditions are satisfied. These further conditions are that the instruction is a SPG or FLG ($PKIR^{FF}$), and that either the correct check parity condition exists (FP^{ODD}_{10}), or register 00 is used ($PKIR^{00}_{CF}$), or that the F parity alarms are suppressed ($FPAL_{SUP}$). This pulse is delayed 0.1 microsecond, and, if $\overline{FK^{0\alpha}}$ (i.e., when $FK^{2\alpha}$, $FK^{4\alpha}$ or $FK^{6\alpha}$), is used to index $PKIR_{CF}$. The master pulse is also used in other F Memory system logic (see below).

12-2.10.2 $QKIR_{CF}$ REGISTER DRIVER LOGIC. There are three paths over which information can be transferred into the $QKIR_{CF}$ register. These paths are from the F Memory, E register and N register.

The F Memory strobe pulse is generated (see Fig. 12-19) by delaying the F Memory master pulse 0.38 microsecond and gating it with $(\overline{PKIR^{00}_{CF}} \cdot \overline{PKIR^{LF}})$, i.e., the strobe pulse is permitted only if register 00 is not used and the instruction is not a SPF or SPG. The pulse is further gated by $PKIR_{CF_5}$, so that account can be taken of the difference in polarity of the memory sense amplifier signals from the two halves of the F Memory (see 12-4.2). In every existing use, this strobe pulse is really a ONE's transfer, even though it is written as a jam transfer. This pulse becomes a jam transfer only when the (unused) Complement Mode (FC^{1}) of operation of the F Memory is used.

The path from the E register involves only E_1. This path is used during SPF and SPG instructions ($PKIR^{LF}$) as part of the route from the Memory Element to the F Memory. (See Fig. 12-20.) The logic is identical to the memory strobe pulse logic except for the $PKIR^{LF}$ factor.

Here, of course, the pulse gate logic involves E rather than the memory sense amplifiers. Note that this pulse causes only a ONE's transfer. Note also that the input to $QKIR_{CF_P}$ is from a compute parity circuit based upon E_1.

The third path is actually a pair of paths leading from the N register into the $QKIR_{CF}$ register. (See Fig. 12-21.) At $PK^{09\alpha}$ of the first deferred address cycle ($PI_2^1 \cdot XAS^0$) the $N_{3.6 - 3.1}$ bits are stored in $QKIR_{CF_{9-4}}$ for use later in the final deferred address cycle. Also, the contents of $N_{3.6 - 3.5}$ are transferred into $QKIR_{CF_{2-1}}$ at $QK^{01\alpha}$ during an SKM instruction. The pulse gate logic here subtracts one from the value of the two CF bits. A permutation is then formed from the right three bits of QKIR (the rest of $QKIR_{CF}$ is cleared) that allows SKM to select the desired quarter of the operand word.

The clear pulse for $QKIR_{CF}$ is separated into two pulses. The first pulse clears bits $QKIR_{CF_{P,\ 9-3}}$, whereas the second pulse clears $QKIR_{CF_{2,1}}$. The logic for the two pulses is identical except for an additional term in the $QKIR_{CF_{P,\ 9-3}}$ logic. This term is shown separately in Fig. 12-22. The remaining logic is shown for both pulses together. The extra term in the clear pulse for bits $QKIR_{CF_{9-3}}$ occurs when the transfer of bits $N_{3.6 - 3.5}$ into $QKIR_{CF_{2,1}}$ takes place during an SKM.

The first clear $QKIR_{CF}$ situation occurs by delaying the F Memory master pulse 0.1 microsecond and, in the case of $PKIR^{LF}$ type instructions (SPF, SPG), clearing the $QKIR_{CF}$ register in anticipation of the E_1 ONE's transfer. An additional term involving FC^0 can be neglected since the circuitry is wired as if the flip-flop FC does not exist.

The second clear $QKIR_{CF}$ situation occurs by delaying the F Memory master pulse 0.5 microsecond so that when register 00 is being read out ($PKIR_{CF}^{00}$), a clear $QKIR_{CF}$ pulse can substitute for a memory strobe pulse.

The third situation arises when $QKIR_{CF}$ is cleared at $PK^{01\alpha}$ of the first deferred address cycle ($PI_2^1 \cdot XAS^0$) in anticipation of the $N_{3.6 - 3.1}$ bits being placed in $QKIR_{CF}$.

12-2.10.3 $AKIR_{CF}$ REGISTER DRIVER LOGIC. Information is transferred into $AKIR_{CF}$ from both the N and $QKIR_{CF}$ registers.

The contents of $N_{1.9 - 1.4}$ are jam-transferred into the $AKIR_{CF}$ register at $PK^{25\alpha}$ during an AOP instruction. The contents of $QKIR_{CF_{9-8}}$ are jam-transferred into $AKIR_{CF_{9-8}}$ at $QK^{13\alpha}$ during $QKIR^{AK}$ type instructions. Also, under these same conditions, the extended activity levels $QKIR_{4-1}^{EXT\ ACT}$ are jam-transferred into $AKIR_{CF_{7-4}}$.

12-3 X MEMORY SYSTEM

12-3.1 GENERAL DESCRIPTION. The X Memory system consists of the X Memory, X buffer register, register selector, sense amplifiers, digit drivers, read-write drivers and J Decoder (JD). The function and structure of the X Memory system was covered in Chapters 2 and 4.

The register selector, X Memory and read-write drivers are illustrated in Fig. 12-24. The register selector uses the $N_{3.6}$ bit and the JD levels obtained by decoding $N_{3.5 - 3.1}$ to determine which register (X_J) is selected in the X Memory. The read-write drivers use the outputs of the X Read (XR) and the X Write (XW) flip-flops, in conjunction with the state of the $N_{3.6}$ bit, to control when read or write currents should occur.

12-3.2 X MEMORY. The X Memory is a 64 register, 19-bit word magnetic core memory with two cores per bit. It is a one-dimensional selection memory with registers selected by the outputs of a two-stage address decoder. Each register selector wire (see Fig. 12-24) is connected to both the read and write drivers. The direction and magnitude of the current in this wire depends on whether a READ or WRITE cycle is occurring.

In order to understand the WRITE cycle, it is essential to realize that only one of the two cores per bit may be in the "set" state at any one time. However, both cores may be in a "cleared" state. Before a WRITE cycle is executed, both cores must be "cleared". Fig. 12-25 illustrates the current flow during the WRITE cycle. In any given digit column, the X register flip-flop (or the XP_{18} level) feeds into the digit drivers and determines the direction of the current flow in the digit winding. When the write driver is turned on, the current in the register winding induces a field in one of the two digit cores of the selected register in the same direction as the flux induced by the current in the digit winding. The core which then switches to the "set" state remembers whether a ONE or a ZERO is written. The other core does not switch to the "set" state since the flux induced in the core by the current in the register winding is in opposition to that of the digit winding. The parity digit position uses the output of the parity compute circuit to determine the direction of current flow in the digit winding so that a word is always written with the correct parity.

During the READ cycle (see Fig. 12-26), a large current flows in the register winding in a direction opposite to that that occurred during the WRITE cycle. This read current is large enough by itself to switch all the cores of the selected register which are in the "set" state to the "cleared" state. In the given digit column, the change in flux resulting from the switching of a "set" core induces a current, or really a voltage pulse, in a direction opposite to the one which existed in the digit winding during the WRITE cycle. Thus, if the ONE core was switched during the write operation, then a positive pulse will appear in the figure at the lower right end of the digit winding. This pulse is fed through the differential amplifier and appears as a negative gate level on one of the two transistors whose emitters are pulsed by the XM $\xrightarrow{j}$ X strobe pulse. This strobe pulse then can get through to hit the ONE side of the flip-flop. After the READ cycle, all the cores in the selected register are left in the "cleared" state in readiness for a new WRITE cycle.

Note that at all times a current is flowing in the digit winding in one direction or the other. This current does not influence the state of either core in any bit position unless a read or write current also exists in the digit winding. During a WRITE cycle, the write current is large enough so that one core will switch, but not both cores. During a READ cycle the read current is large enough to switch whichever core is "set" by swamping out the effect of the digit current. The read-out signal itself is observed as a voltage pulse superimposed on the digit current.

Also, since the register selector is directly connected to the $N_{3.6 - 3.1}$ bits, some register winding is always selected. In order to reduce the amount of noise on these windings caused by selection and deselection, the content of the third quarter of N is altered only when a new X Memory register is to be selected. Thus, the logic for the third quarter of N is quite different from that for the other quarters of N.

12-3.2.1 X MEMORY READ LOGIC. The logic for the X Memory read flip-flop (XR) is shown on Fig. 12-27. XR turns on the read current in the selected register for 0.46 microsecond. It is set no sooner than 0.2 microsecond after the $N_{3.6 - 3.1}$ bits are changed. It is turned on at $PK^{13\alpha}$ and $CSK^{01\alpha}$ in order to actually read out the contents of the selected index register. It is turned on at $CSK^{04\alpha}$, and at $QK^{13\alpha}$ during AUX, RSX and EXX in order to clear the selected register before a word is written in the register.

12-3.2.2 X MEMORY WRITE LOGIC. The logic for the X Memory write flip-flop (XW) is shown in Fig. 12-28. The write current is turned on for 1.6 micro-seconds during XWK cycles.

12-3.3 X ADDER. The X Adder performs an 18 bit ONE's complement full sum addition. All carrys and partial additions are internal and do not require separate pulses.

The X Adder consists of 18 bits or stages. Alternating stages of the X Adder are identical in construction although all stages have the same function. The last stage of the X Adder, stage 2.9, must take into account the special nature of the defer bit whenever $N_{2.9}$ is interpreted as the defer bit. However, it assumes the function of just another adder stage at all other times.

A typical pair of X Adder stages is shown in Fig. 12-29. Each stage contains:

1) One or two partial add circuits
2) A carry-out circuit
3) A carry-in circuit
4) Either a force-carry circuit or an enable (or kill-carry) circuit
5) A full sum circuit
6) A selector and driver circuit for the output

The stages which contain only one partial-add circuit also contain a force-carry circuit.

The partial-add circuit forms the partial sum of the contents of $N_{2,1}$ and the contents of the X register. The partial sum logic is:

$$PS_{i.j} = N^0_{i.j} \cdot X^1_{i.j} + N^1_{i.j} \cdot X^0_{i.j} \left(= N_{i.j} \neq X_{i.j} \right)$$

Each stage also generates a carry-out (CYO) which essentially forms the carry input (CYI) to the next stage to the left. The carry logic for even numbered stages ($XA_{2.8, 2.6, 2.4, 2.2, 1.9, 1.7, 1.5, 1.3, 1.1}$) is:

$$CYO_{i.j} = X^1_{i.j} \cdot N^1_{i.j} + CYI_{i.j} \cdot (X_{i.j} \neq N_{i.j})$$

The carry logic for odd numbered stages ($XA_{2.9, 2.7, 2.5, 2.3, 2.1, 1.8, 1.6, 1.4, 1.2}$) is:

$$CYO_{i.j} = (X^1_{i.j} + N^1_{i.j}) \cdot \left[CYI_{i.j} + (X_{i.j} \neq N_{i.j}) \right]$$

The two forms for the CYO logic are necessary because each stage acts as an inverter, and alternate stages must use dual forms of the logic.

The carry-in circuit (CYI) consists of an amplifier and an inverter to provide both polarities of the carry level. Since inversion of a level takes a significant amount of time, the inverted carry level is used only when the delay will not have a cumulative effect on the over-all carry time.

In the odd-numbered stages, the carry-in circuit is tied to the enable, or kill-carry circuit. The enable or kill-carry circuit turns off (kills) the carry circuit when the XAC flip-flop is set. However, the XAC flip-flop clears itself 0.4 microsecond after being set. (See Fig. 12-29.) When the XAC flip-flop is cleared, the carry-in circuit is enabled to transmit carry information.

The full sum circuit takes the outputs of the partial-add and carry-in circuits and completes the addition process. The full sum logic for even numbered stages is:

$$SUM_{i.j} = CYI_{i.j} \cdot (X_{i.j} \neq N_{i.j}) + \overline{CYI_{i.j}} \cdot (X_{i.j} \neq N_{i.j})$$

The full sum logic for odd numbered stages is:

$$SUM_{i.j} = \left[CYI_{i.j} + (X_{i.j} \neq N_{i.j})\right] \cdot \left[\overline{CYI_{i.j}} + (X_{i.j} = N_{i.j})\right]$$

The duality of the circuits as described by these two equations is occasioned by the inversion which takes place in each stage of the carry circuit.

The 2.9 stage of the X Adder must take into account the defer bit character of the $N_{2.9}$ bit. (See Fig. 12-30.) The logic is identical to other similar stages in the X Adder, except that the $\lfloor 0 \diamond XA_{2.9}$ and $\lfloor 1 \diamond XA_{2.9}$ levels are substituted for $N^0_{2.9}$ and $N^1_{2.9}$, respectively, in the adder inputs. If the $N_{2.9}$ bit is a ONE during PK cycles between PK^{13} and PK^{22}, this stage of the adder should receive a ZERO in order for correct address modification to occur. At these times PI_2 is in the ONE state and because of this the $\lfloor 0 \diamond XA_{2.9}$ input occurs. At other times, PI_2 is in the ZERO state, and because of this either the $\lfloor 0 \diamond XA_{2.9}$ or the $\lfloor 1 \diamond XA_{2.9}$ input is generated, reflecting the value of $N_{2.9}$.

12-4 F MEMORY SYSTEM

12-4.1 GENERAL DESCRIPTION. The F Memory system consists of the F Memory, the $QKIR_{CF}$ and $PKIR_{CF}$ registers, and the $PKIR_{CF}$ address decoder. The function and structure of the F Memory system were covered in Chapters 2 and 4.

As shown in Fig. 12-31, the value of the five CF bits in the N register is transferred into the $PKIR_{CF}$ register. The content of the $PKIR_{CF}$ register is used for two purposes. Normally it is interpreted as an address in the F Memory. However in certain instructions it has a special purpose, e g., it is used to increment index (X Memory) registers during JX type instructions.

The $PKIR_{CF}$ register has an indexing circuit which indexes the register during execution of SPG and FLG instructions.

The information read out of the F Memory is transferred into the $QKIR_{CF}$ register. The content of this register is decoded into activity, coupling and permutation information. (See Sect. 12-6.)

12-4.2 F MEMORY. The F Memory is a 32 register, 10 bit word magnetic film memory. It is a one-dimensional memory array. Register selection is by a two stage selector whose inputs are the $PKIR_{CF}$ bits.

Each word selection line is connected to a switch core selected by the decoder. (See Fig. 12-32.) A switch core is turned "ON" when the FR flip-flop is set. The five $PKIR_{CF}$ bits then select the decoded register. When the core is switched "ON" a current is induced in the word selection line. When the core is turned off a diode in the word selection line prevents an opposite current from flowing.

During a WRITE cycle, the content of the entire buffer including the parity bit is written back into the F Memory. The direction of the current in the digit winding determines whether a ONE or ZERO is written (see Fig. 12-33). A ZERO is written if the $QKIR_{CF}$ bit is ZERO. The TW flip-flop controls the current in the digit windings and permits a ONE to be written during the WRITE cycle when the $QKIR_{CF}$ bit is a ONE.

While the digit current is flowing, a ONE or ZERO is written only if the magnetic film spot is pulsed by the flux from the word selection line. If the spot contains a ZERO and a ONE is to be written, the digit winding produces a flux in the opposite direction to that existing in the magnetic film spot. However, the film will not switch unless the external field exceeds a certain threshold. This threshold is attained when the field from the pulsed word line is added to the field induced by the digit line. If the spot contains a ZERO and a ZERO is to be written, the external field induced by the digit winding is in the same direction as that of the magnetic film spot and the ZERO remains.

During a READ cycle as shown in Fig. 12-34, the digit current flows in the same direction as in the writing of a ZERO. This occurs since the TW flip-flop is turned off. The read is effected when the word current is turned on to aid the digit current. This switches the ONES to ZEROS. The ZEROS remain as ZEROS and are not read out. To minimize noise in the sense lines, the sense lines are crossed between the two arrays. This occasions a bi-polar output. The output of either of the two arrays must be properly interpreted. This is effected by the strobe unit which contains a dual transistor strobe circuit. Two gate pulse amplifiers divide the strobe pulse according to whether the first or second array is used. The $PKIR_{CF_5}$ bit is used to select the proper gate pulse amplifier. Array No. 1 is selected when this flip-flop is a ZERO, and Array No. 2 when this flip-flop is a ONE.

Originally, a "complement" mode of operation was considered which required an extra transistor circuit in the strobe unit on the ZERO output side. Since this feature has been abandoned, the complement flip-flop (COMP) output is now tied to the zero state at all times. Much of the physical circuitry for this mode is still intact but is not used.

12-5 OPERATION DECODING PROCESS

12-5.1 GENERAL DESCRIPTION. The paths along which the operation code information flows are shown on Fig. 12-35. The general interpretation of the six OP code bits in the instruction word was discussed in Chapters 2 and 7.

The op code bits are transferred sequentially during the instruction execution from the N register into the $PKIR_{OP}$ register; from the $PKIR_{OP}$ register into the $QKIR_{OP}$ register; and from the $QKIR_{OP}$ register into the $AKIR_{OP}$ register. However, the picture is quite different for AOP instructions. In these instructions, the contents of $N_{2.6 - 2.1}$ are transferred directly into the $AKIR_{OP}$ register.

The information in each of these registers is decoded through several levels. $PKIR_{OP}$ and its associated decoder are used principally during PK cycle operations. However, they can also be used during QK and AK cycles. The $QKIR_{OP}$ system is used principally during QK and AK cycles. The $AKIR_{OP}$ system is used only during AK cycles.

The fact that there are three OP registers permits three different instructions to be executed simultaneously. An illustration of this situation is shown on Fig. 12-36. The bar graph in the figure shows the overlapping of several counter cycles.

12-5.2 OP REGISTER AVAILABILITY FOR DECODING. $PKIR_{OP}$, $QKIR_{OP}$ and $AKIR_{OP}$ receive their information, and hence are available for decoding at the following times. The six OP bits in the N register are transferred into $PKIR_{OP}$ at $PK^{12\alpha}$ and are available for decoding in the PK cycle after that time. When the QK cycle starts, the content of $PKIR_{OP}$ is transferred into $QKIR_{OP}$. During $QKIR^{AK}$ instructions, the content of $QKIR_{OP}$ is transferred into $AKIR_{OP}$ at $QK^{13\alpha}$. During AOP instructions, the contents of $N_{2.6 - 2.1}$ are transferred into $AKIR_{OP}$ at $PK^{25\alpha}$. Thus when $AKIR_{OP}$ is decoded depends on the instruction.

12-5.3 OP REGISTER 1ST AND 2ND LEVEL DECODING. The first level decoding of each of the three OP registers consists of a complete decoding of each pair of three bits. This results in two sets of eight lines. E.g., the $PKIR_{OP}$ 1st level decoder lines are:

DECODED FROM $PKIR_{OP}$ 6-4	DECODED FROM $PKIR_{OP}$ 3-1
$PKIR_{OP}^{0X}$	$PKIR_{OP}^{X0}$
.	.
.	.
.	.
$PKIR_{OP}^{7X}$	$PKIR_{OP}^{X7}$

The second level OP decoder combines pairs of outputs from the first level decoder to generate an OP line. E.g.,

$$PKIR^{TSD} = PKIR_{OP}^{0X} \cdot PKIR_{OP}^{X7}$$

Such OP lines are generated for each OP register, but only as many lines are decoded as are actually needed.

12-5.4 CLASS DECODERS. The $PKIR_{OP}$ class decoder combines the outputs of the $PKIR_{OP}$ register and the first and second level $PKIR_{OP}$ decoders into levels which represent specific classes of OP codes. These OP class lines are used in level logic, when it is convenient to represent a class of OP codes exhibiting common properties by a single level. $QKIR_{OP}$ and $AKIR_{OP}$ class decoder levels are generated in a similar manner.

12-5.4.1 $PKIR_{OP}$ CLASS LEVELS. A brief description of each of these levels is given below.

$PKIR^{F}$ (FLF, FLG, SPF, SPG). These OP codes require the use of the content of a register in the F Memory as an operand.

$PKIR^{FF}$ (SPG, FLG). These OP codes require the use of the contents of four successive registers in the F Memory as operands.

$PKIR^{LF}$ (SPF, SPG). These OP codes load the F Memory with Memory Element information.

$PKIR^{SF}$ (FLF, FLG). These OP codes store F Memory information in the Memory Element.

$PKIR^{XM}$ (JPX, JNX, ^{XXX1X}JMP, SKX). These OP codes postpone starting XWK until after $PK^{14\alpha}$. These are $PKIR^{QK}$ instructions which use an X Memory operand cycle, i.e., the XWK operand cycle replaces the QK operand cycle. Other instructions, such as AUX, require a QK operand cycle.

$PKIR^{AE}$ (CYA, CYB, CAB, SCA, SCB, SAB, NOA, NAB, TLY, ITA, UNA, EXA, DSA, INS, ADD, SUB, MUL, DIV, LDA (-B, -C, -D), STA (-B, -C, -D), JPA, JNA, JOV and AOP). These OP codes use the Arithmetic Element.

$\overline{PKIR^{QK}}$ (OPR, JMP, JPX, JNX, JPA, JNA, JOV, SKX and the undefined OP codes whose octal numbers are: 00, 01, 02, 03, 13, 23, 33, 45, 50 51, 52, 53, 63 and 73. These OP require no QK operand cycle.

$PKIR^{OPR\ AE}$ (AOP). This level is decoded during an OPR instruction when no deferred address cycles are requested and $N^0_{2.8} \cdot N^1_{2.7}$.

$PKIR^{IND}$ (LDA (-B, C, D, E), STA (-B, -C, -D, -E), SPF, SPG, FLF, FLG, ITE, ITA, UNA, EXA, DSA, INS, SED, JPA, JNA, JOV, PCM, TSD, CYA, CYB, CAB, SCA, SCB, SAB, NOA, NAB, TLY, ADD, SUB, MUL, DIV, $^{xxxxl}JMP$, $^{xxxlx}SKX$ and $^{xxlxx}SKX$). These are OP codes in which the base address is indexed.

$PKIR^{IOS}$ (IOS). This level is decoded during an OPR instruction when $N^0_{2.8} \cdot N^0_{2.7}$.

$PKIR^{JX}$ (JPX, JNX). These OP codes specify jump instructions which are conditional on the X Memory.

$PKIR^{JA}$ (JPA, JNA, JOV). These OP codes specify jump instructions which are conditional on the Arithmetic Element.

$PKIR^{DIS}$ (OPR, TSD, JMP, JPA, JNA, JOV, JPX, JNX, SED, SKM and SKX). These are OP codes in which PK runs through to $PK^{31\alpha}$, i.e., require a PKEI cycle. (In $\overline{PKIR^{DIS}}$ instructions, PK runs through to $PK^{24\alpha}$.)

$PKIR^{DIS\ REQ}$ (TSD, JPX, JNX, JMP, IOS and SKX). This class level is decoded during instructions in which a "dismiss request" is generated. (The hold bit is not included in the level.) A "dismiss request" occurs:

1) In a TSD.
2) In an index jump, when the jump condition is satisfied.
3) In an IOS, which has its dismiss bit set (CF^1_5) and which is not raising the flag of the current sequence (IOS 50000, with $K^{eq\ J}$).
4) In a SKX, which has its dismiss bit set and which is not raising the flag of the current sequence ($^{xlxxx}SKX$, with $K^{eq\ J}$).

$\overline{PKIR^{DEF}}$ (00, 01, 02, 03, 04, 13, 23, 33, 45, 50, 51, 52, 53, 63, 73). This OP code class level is decoded during instructions which are not defined. This class includes the OPR code (04) only when $N^{1}_{2.8}$.

The logic generating the $PKIR_{OP}$ class levels is shown on Fig. 12-37.

12-5.4.2 $QKIR_{OP}$ CLASS LEVELS. A brief description of each of these levels is given below. Note that no undefined OP codes ever appear in the $QKIR_{OP}$ register.

$QKIR^{FL}$ (FLF, FLG). These OP codes store F Memory information in the Memory Element.

$QKIR^{ST}$ (STA (-B, -C, -D, -E), EXA, EXX and DPX). These OP codes perform simple storing operations in the Memory Element. FLF and FLG are not included.

$QKIR^{LD}$ (LDA (-B, -C, -D, -E), EXA, EXX, RSX, AUX, ITA, UNA and the $QKIR^{AK}$ type OP codes (see below)). These OP codes perform simple loading operations using operands from the Memory Element. Note that SPF and SPG are not included.

$QKIR^{STORE}$ (ADX, FLF, FLG, INS, PCM, SKM, TSD and $QKIR^{ST}$ type instructions). These OP codes include all those which can change a word in the Memory Element.

$QKIR^{LOAD}$ ($\overline{QKIR^{STORE}}$). The $QKIR^{LOAD}$ OP code class level reflects all the OP codes which do not change a word in the Memory Element (i.e., $\overline{QKIR^{STORE}}$ OP codes).

$QKIR^{X}$ (DPX, EXX, RSX). These are OP codes which obtain an operand from the X Memory, but which do not use the X Adder for summing in this process.

$QKIR^{AK}$ (SCA, SCB, SAB, CYA, CYB, CAB, NOA, NAB, ADD, SUB, DSA, MUL, DIV and TLY). These are the OP codes which use the AK counter.

$QKIR^{AESK}$ (SCA, SCB, SAB, CYA, CYB, CAB, NOA, NAB, MUL, DIV and TLY). These are the OP codes which use the AE step counter. Note that these OP codes are a subclass of the $QKIR^{AK}$ class.

$QKIR^{A}$ (STA, EXA and TLY). These are the OP codes which use the A register as an operand or data register.

$QKIR^{B}$ (LDB, STB, INS). See $QKIR^{A}$ above.

$QKIR^{C}$ (LDC and STC). See $QKIR^{A}$ above.

$QKIR^{D}$ (LDD, STD, ADD, SUB, MUL, DIV, NOA, NAB, SCA, SCB, SAB, CYA, CYB and CAB). See $QKIR^{A}$ above.

$QKIR^{E}$ (ITE, LDE, STE and SED). See $QKIR^{A}$ above.

The logic generating the $QKIR_{OP}$ class levels is shown on Fig. 12-38.

12-5.4.3 $AKIR_{OP}$ CLASS LEVELS. These levels are discussed in detail in Chapter 14.

12-6 CONFIGURATION DECODING PROCESS

12-6.1 GENERAL DESCRIPTION. The paths along which the configuration information flows are shown on Fig. 12-39. The CF bits in the instruction word are transferred from the N register into the $PKIR_{CF}$ register. The content of the $PKIR_{CF}$ register is then decoded and used to select a word in the F Memory. The selected word is strobed into the $QKIR_{CF}$ buffer register. The content of the $QKIR_{CF}$ register is then interpreted by various decoders which specify subword forms, activities and permutations. The general interpretation of the five CF bits in the instruction word in terms of subword form, activity and permutation was discussed in Chapter 2.

12-6.2 SUBWORD FORM. The subword forms in the A, B, C, D and E register during PK and QK cycles are defined by the value of the bits in $QKIR_{CF9,8}$ as shown in Table 12-1.

TABLE 12-1
SUBWORD INTERPRETATION OF $QKIR_{CF9,8}$

$QKIR_{CF9,8}$	FRACTURE ($QKIR^{f}i$)	SUBWORD FORM
00	f_1	36
01	f_2	18,18
10	f_3	27,9
11	f_4	9,9,9,9

A graphical representation of the subword forms is shown in Fig. 12-40. Any combination of subwords of a given form is allowed. E.g., in the (9,9,9,9) subword form, quarters 3 and 1 can be used simultaneously.

During AK cycles, the subword forms of the data in the Arithmetic Element are defined by $AKIR_{CF9,8}$ as shown in Table 12-2. Note that in nearly all cases, the information in $AKIR_{CF9,8}$ is a copy of the information in $QKIR_{CF9,8}$.

TABLE 12-2
SUBWORD INTERPRETATION OF $AKIR_{CF9,8}$

$AKIR_{CF9,8}$	FRACTURE ($AKIR^{f}i$)	SUBWORD FORM
00	f_1	36
01	f_2	18,18
10	f_3	27,9
11	f_4	9,9,9,9

12-6.3 ACTIVITY. The "activity" of each quarter in the A, B, C, D and E registers during PK and QK cycles is determined by the value of the bits in $QKIR_{CF7-4}$ as shown in Table 12-3.

TABLE 12-3

$QKIR_{CF}$ (CF_7 CF_6 CF_5 CF_4)	ACTIVITY ($QKIR^{ACT}i$)
x x x 0	ACT_1
x x 0 x	ACT_2
x 0 x x	ACT_3
0 x x x	ACT_4

Note that the quarter is "active" when the controlling $QKIR_{CF}$ flip-flop is a ZERO, and "latent" (i.e., not active) when the flip-flop is a ONE.

During AK cycles, the quarter activity of the Arithmetic Element is defined by $AKIR_{CF7-4}$. Note that usually the information in $AKIR_{CF7-4}$ is a copy of the information in $QKIR_{CF7-4}$.

TABLE 12-4

$AKIR_{CF}$ $(CF_7\ CF_6\ CF_5\ CF_4)$	ACTIVITY $(AKIR^{a^1}_i)$
x x x 0	a^1_1
x x 0 x	a^1_2
x 0 x x	a^1_3
0 x x x	a^1_4

12-6.3.1 EXTENDED ACTIVITY. Subwords can be defined in which not all the quarters of the subword are active. These are referred to as partially active subwords. An example of a partially active subword is given in Fig. 12-41. In this example only one quarter of an 18 bit subword is active. Activity extension is the process by which an entire subword is made active if the subword is partially active, i.e., activity is extended into the inactive (latent) quarters in the subword. When the subword form has an influence upon the execution of an instruction, as in an ADD or MUL, the partially active subwords are usually made fully active. This is done by nets which extend the activity of quarters of a partially active subword to all quarters of the subword. The result is that the subwords are either wholly active or wholly inactive.

Activity is extended in preparation for sign extension operations in the Exchange Element. Partially active subwords have inactive quarters filled by the sign digits of active quarters.

Fig. 12-42 illustrates sign extension for all four subword forms. In each case the second quarter is active. For an f_1 (36) subword form, the activity is extended through the third and fourth quarters and then around through the first quarter. For an f_2 (18,18) subword form, the activity is extended around through the first quarter. The subword containing the third and fourth quarters are not influenced. For an f_3 (27,9) subword form, the activity is extended through the third and fourth quarters. The subword containing the first quarter is not touched. For an f_4 (9,9,9,9) subword form, the activity cannot be extended since each

subword contains only one quarter each. Therefore the three subwords containing the first, third and fourth quarters are not influenced.

Fig. 12-43 shows the logic generating the extended activity levels.

Level $QKIR^{EXT\ ACT}{}_1$ indicates how activity can be extended into the first quarter for various combinations of quarter activities and subword forms. As an example,

$$QKIR^{EXT\ ACT}{}_1 = QKIR^{ACT}{}_2 \cdot QKIR^{f}{}_1 + {}^{f}{}_2$$

shows that activity is extended into the first quarter when the second quarter is active, providing the subword form is f_1 (36) or f_2 (18,18). Both these subword forms contain the active second quarter and the inactive first quarter through which the activity is to be extended. Subword forms f_3 (27,9) and f_4 (9,9,9,9) do not appear since the first quarter is not in the subword which contains the active quarter.

Level $QKIR^{EXT\ ACT}{}_{2,1}$ is a combination of levels $QKIR^{EXT\ ACT}{}_2$ and $QKIR^{EXT\ ACT}{}_1$. It specifies activity in the first and second quarters. This insures that the first two quarters of the E register are active during the execution of an exchange index instruction.

Level $QKIR^{ALL\ ACT}$ is a combination of levels $QKIR^{ACT}{}_1$, $QKIR^{ACT}{}_2$, $QKIR^{ACT}{}_3$ and $QKIR^{ACT}{}_4$. This level is not actually used any more, but says that the whole word is active.

Fig. 12-44 shows the relationship between a subword form with a specific activity and the $QKIR^{EXT\ ACT}{}_j$ levels which are generated.

12-6.4 PERMUTATION. The permutation of the operand word in the Exchange Element is determined by $QKIR_{CF3-1}$. These flip-flops are decoded to generate $QKIR^{PRM}{}_i$ levels. The eight possible permutations are shown in Fig. 12-45. This figure also illustrates the connection between activity in the central computer and in the Memory Element.

The effective permutation paths between the M register and the E register are graphically shown in Fig. 12-45. The actual mechanics of the permutation process in the Exchange Element are discussed in Chapter 13.

12-6.4.1 PERMUTED ACTIVITY. The permuted activity level $QKIR^{PRM\ ACT}{}_i$ indicates that the i-th quarter of memory is connected to an active quarter of the central machine via the specified permutation path.

Fig. 12-46 shows the logic generating the various permuted activity levels.

For a specified permuted activity level there are various possible combinations of permutations and active quarters in the central machine. As an example,

$$QKIR^{PRM\ ACT}{}_1 = QKIR^{ACT}{}_2 \cdot QKIR^{PRM}\ 3 + 4$$

shows that the first quarter in memory is active when the second quarter of the central machine is active, providing that either the $QKIR^{PRM}{}_3$ or $QKIR^{PRM}{}_4$ permutation paths are specified. Fig. 12-45 shows that these are the only two permutation paths possible for this situation.

12-6.5 SIGN EXTENSION. In sign extension, the inactive quarters of subwords are filled with the sign bit of active quarters of that subword. Inactive quarters to the left of an active quarter in the subword are first cleared, and then complemented if the sign bit of the active quarter is a ONE. (See Fig. 12-47.)

The logic governing the clearing of quarters of E under sign extension control is,

$$\left\lfloor \frac{0}{SE} \right\rfloor E \cdot QKIR^{EXT\ ACT}{}_i \cdot \overline{QKIR^{ACT}{}_i} \supset \lfloor 0 \rfloor E_i,\ i = 1,2,3,4$$

The $\frac{0}{SE}$ E level contains OP code and time level information. The remaining logic in this level simply says that the given quarter is itself inactive, but that the quarter forms part of a partially active subword.

If the sign which is being extended is negative, i.e., a ONE, then a complement pulse must be fired. The logic for these complement pulses is given by,

$$\left\lfloor \frac{C}{SE} \right\rfloor E \cdot QKIR^{EXT\ ACT}{}_i \cdot \overline{QKIR^{ACT}{}_i} \cdot S_i \supset \lfloor C \rfloor E_i,\ i = 1,2,3,4$$

Here the logic for $\left\lfloor \frac{C}{SE} \right\rfloor \rightarrow E$ is identical to that for $\left\lfloor \frac{0}{SE} \right\rfloor \rightarrow E$ except that it occurs 0.2 microsecond later. S_i specifies that Quarter i is itself inactive, but lies to the left of an active quarter whose leftmost bit is a ONE, where both quarters are in the same subword and there are no intervening active quarters. The logic for the S_i's includes the quantitites A, B, C and D. (See Fig. 12-47.) These quantities are actually the output of a carry-like sign extension net.

For example, suppose that the f_1 (36) subword form is specified with quarters 2 and 4 active and $E^1_{2.9}$ and $E^0_{4.9}$. Then Quarters 3 and 1 will be cleared, since they have extended activities (see Fig. 12-43). Since Quarter 2 is active and $E^1_{2.9}$, quantity C is generated. This indicates that there is a negative sign to be extended to the left out of Quarter 2, and causes quantity D to be generated. Note that since $E^1_{4.9}$ and Quarter 4 is active, quantity A is not generated, and hence neither is S_1 and S_2. Of the four S_i's, only S_3 is generated. Thus, only Quarter 3 of E is complemented under sign extension control. The negative sign of Quarter 2 is extended to the left to fill Quarter 3, and the positive sign of Quarter 4 is extended to the left to fill Quarter 1.

12-7 SEQUENCE SELECTION

12-7.1 GENERAL DESCRIPTION. The Sequence Selector is the unit in the Program Element which determines whether the next instruction to be executed is taken from the current program sequence or from some new program sequence.

The components which comprise the sequence selector are:

1) The individual Sequence Selector stages
2) The Priority Patch Panel
3) The K Decoder
4) The J Coder
5) The FLAG register
6) The $K^{eq\ JC}$ net
7) The $K^{eq\ J}$ net

The Priority Patch Panel fixes the relative priority relationships among the program sequences. As its name implies, the panel is a plugboard with patch cords which are used to provide any desired arrangement of priorities among the sequences.

The Sequence Selector stages determine which is the highest priority sequence desiring attention. The Priority Patch Panel of course influences this decision.

The K Decoder provides the Sequence Selector with the number of the current sequence being executed.

The J Coder encodes the thirty-three outputs of the sequence selector stages and specifies the number of the next sequence. The encoded number can then be inserted into $N_{3.6\ -\ 3.1}$, the J bits in the N register.

The FLAG register indicates which sequences request attention. It contains one FLAG flip-flop for each of the 33 program sequences.

The two nets on the K register compare the number in the K register with the output of the J Coder and the number in the J bits.

12-7.2 PRIORITY PATCH PANEL. This panel consists of a plugboard with two sets of 3 X 33 jacks. One set of jacks is associated with the priority number. The other set of jacks is associated with the Sequence Selector stages.

The Priority Patch Panel, as shown in Fig. 12-48, is divided into four sections. Three of the sections are each composed of eight stages which coincide with the Sequence Selector stages. Section 3 has 9 stages instead of 8 stages due to the startover sequence. Each stage of the Priority Patch Panel contains two sets of three jacks. The upper set of jacks of each stage are interconnected throughout a section.

One jack from each stage is connected in parallel. The other two jacks are connected in series. The series connections are used to transmit information serially through a section (with the initial input tied down to represent no information coming in). The parallel connection is used to feed information in from outside a section simultaneously to all stages in a section. The output from the last stage of the series connections of section represents the only piece of information coming out of a section. These outputs are connected as shown to all lower priority sections and to a last OR net to generate the $SS^{ATT\ REQ}$ level. Note that the input to the parallel connection of section 3 (the highest priority) is also tied off to represent no information. The effect of all these connections is that attention request information generated by any Sequence Selector stage is transmitted through at most one full section (8 stages) before contributing to the $SS^{ATT\ REQ}$ level. Two full sections (16 stages) is the maximum delay met before such information gets to any other lower priority Sequence Selector stage. (This should be compared with a maximum delay of 32 stages in a wholly serial net.)

The lower set of three jacks in each stage of each section is connected to the corresponding Sequence Selector stage.

The priority of a given Sequence Selector stage is determined by which (upper) set of three jacks is connected to the (lower) set of three jacks of the Sequence Selector stage. The upper set of three jacks may be in the same section as the lower set of three jacks or even in another section. All these interconnections are accomplished by patch cords and can be changed whenever the need arises.

12-7.3 SEQUENCE SELECTOR. The Sequence Selector consists of 33 stages. All of the stages are identical with the exception of the first stage. The first stage corresponds to the Startover Sequence (octal 00).

Fig. 12-49 illustrates a typical stage of the Sequence Selector.

There are three levels that the Sequence Selector may generate. They are $SS^{CH\ REQ}$ (or $SS^{CH\ SEQ}$), $SS^{NEXT\ SEQ}$ and $SS^{ATT\ REQ}$.

The $SS^{ATT\ REQ}$ level is generated by ORing the $PP^{ATT\ REQ}_{g,0;\ -}$ outputs from the patch panel sections. These outputs from the patch panel are formed by cascoding within the individual sections, the $SS^{ATT\ REQ}_{g,\ h;\ H}$ inputs from the Sequence Selector stages. These individual inputs state that either the flag of a sequence, which is not the current sequence, is up; or that some sequence of higher priority within the same quarter requests attention. Note that if the highest priority stage in a quarter requests attention, it must cascade through, at most, 8 other stages before contributing to the $SS^{ATT\ REQ}$ level. Note also that when the current sequence is in a "waiting" state and no instructions are being executed, then KD is disconnected from the K register and $SS^{ATT\ REQ}$ can indicate whether the current sequence requests attention also.

The $SS^{CH\ REQ}$ level indicates whether a sequence of higher priority than the current sequence has its flag up. This level is used to determine whether a change of sequence to a higher priority sequence can occur when the hold bit on an instruction, being executed in the current sequence, is a ZERO. This is generated by ORing the $SS^{CH\ REQ}_{H}$ levels produced by the Sequence Selector stages. Only one of these levels, the one coming from the current sequence, can ever be on, and even this cannot occur unless KD is connected to K. The individual levels are generated by simply determining whether the attention request levels, coming in from the priority panel connections, state that a higher priority sequence requests attention. The maximum delay met by attention request information before contributing to the $SS^{CH\ REQ}$ level is two full sections (16 stages).

The $SS^{NEXT\ SEQ}_{H}$ levels are generated by the individual selector stages. Only one, at most, of these can be turned on at a given time. If one is on it indicates that the corresponding sequence is the highest priority sequence with its flag raised. The level is formed in a selector stage when the flag is up but no sequence of higher priority requests attention. The current sequence, identified by the KD level, is usually excluded.

The Startover Sequence Selector stage is similar to the others, except that it can occupy only the highest priority position. (See Fig. 12-50.)

All the Sequence Selector stages contain logic to complete the decoding of the $N_{3.6\ -\ 3.1}$ bits. This circuitry is called the N Decoder, even though its inputs are $N_{3.6}$ and JD. The outputs go to the In-Out Element.

12-7.4 FLAG REGISTER. The FLAG register is composed of 33 flip-flops, one for each sequence. Each flip-flop is set, i.e., each flag is raised, when its associated in-out unit requests attention. This is indicated by the $\llcorner 1 \rightarrow$ FLAG pulse shown in Fig. 12-51.

Certain instructions can also affect the flag of the sequence specified by the $N_{3.6 - 3.1}$ bits of the instruction. An IOS instruction can raise or lower (clear) a flag if bits $N_{2.6 - 2.4}$ in the address section have the value 101 or 100, respectively. An SKX can raise a flag if $PKIR^{1}_{CF_4}$. All the pulses which do this are gated by ND and occur at $PK^{26\alpha}$.

Another group of pulses which are gated by KD can also lower flags. However, the gating by KD means that only the flag of the current sequence is affected. These pulses are called "dismiss" pulses. Whenever a change of sequence is made to sequence 00, the Startover Sequence, the flag is lowered in order to allow pulses from the Startover button to recognize which sequence 00 is running. This pulse occurs at $CSK^{05\alpha}$. If the computer attempts to execute a TSD and either it is still executing a previous TSD, or it finds the In-Out unit of the sequence is busy, then it will lower the flag. This pulse occurs at $PK^{22\alpha}$ and is called "dismiss and wait". The other pulse of this type occurs during an ordinary "dismiss". Here the pulse is given at $PK^{25\alpha}$ during instructions which request a dismiss ($PKIR^{DIS\ REQ}$) in sequences other than sequence 00. This last exclusion exists because the dismiss pulse for sequence 00 was given at $CSK^{05\alpha}$ when the sequence was entered and a new flag raising in this sequence might have occurred while sequence 00 was running.

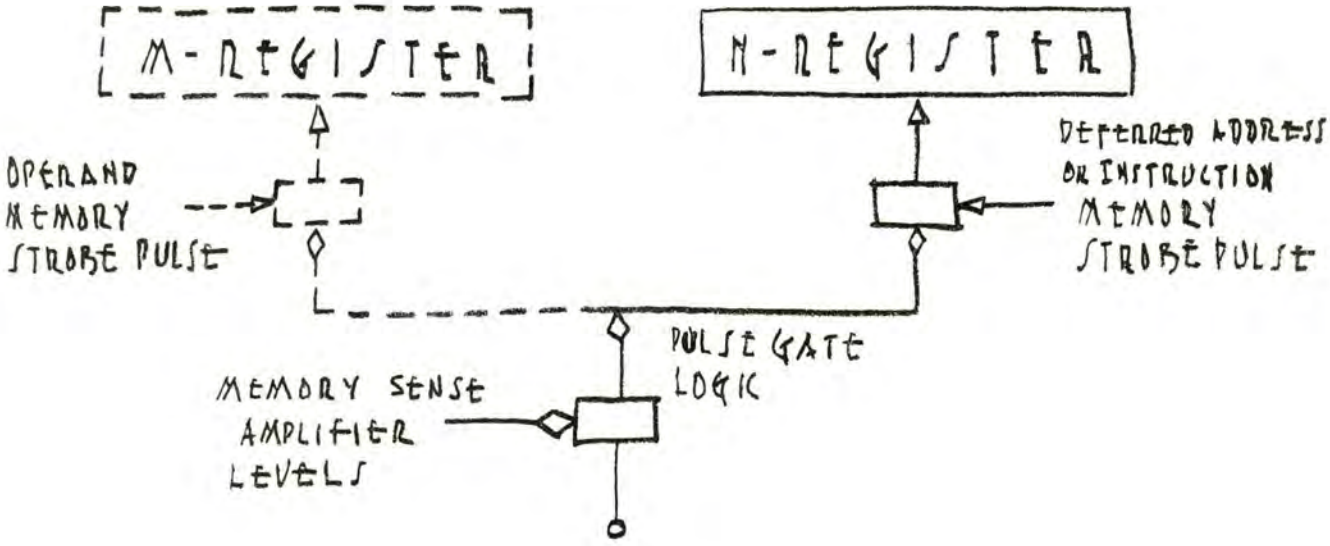
Another level that clears the flags is the $\llcorner$ PRESET $\blacklozenge$ SS level. This level is generated by the start-stop control.

12-7.5 K DECODER. The K Decoder interprets the $K_{3.6 - 3.1}$ bits. It generates 33 lines, one for each of the Sequence Selector stages. These lines also go to the In-Out units. The $K_{3.6}$ bit is used for a special purpose. $K^{1}_{3.6} \cdot \overline{(CSK^{1}_{4} \cdot PK^{00})}$ is substituted for the $K^{1}_{3.6}$ input to the K Decoder. This logic says that when the current sequence is not sequence 00, and the computer is waiting (CSK^{1}_{4}) after an ordinary dismiss ($PK^{00\alpha}$), then the K Decoder is disconnected from K, i.e., KD $\neq$ K. This logic permits the Sequence Selector to request the raising of the flag of the current sequence, after it has been dismissed.

12-7.6 J CODER. The J Coder serves the function of encoding all the $SS^{NEXT\ SEQ}_{14}$ levels into the six bits to be inserted into the $N_{3.6 - 3.1}$ bit position of the N register.

12-7.7 $K^{eq\ JC}$ NET. This net determines whether the number of the current sequence is the same as the highest priority sequence which requests attention. This information is used when the wait cycle (DSK) ends because some sequence wants attention and the possibility exists that this situation might exist because KD was disconnected from K (KD $\neq$ K).

12-7.8 $K^{eq\ J}$ NET. This net determines whether the contents of K and J are equal. It is used, for example, to determine whether IOS and SKX instructions are raising the flag of the current sequence at the same time that they are dismissing. In these cases the $PKIR^{DIS\ REQ}$ level is not generated.



MEMORY STROBE INTO N-REGISTER

FIG 12-1

TM 8-8-60

N-REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	MEMORY	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$SM_{1,2} \xrightarrow{1} N_{1,2}$ $SM_{3,4} \xrightarrow{1} N_{3,4}$ $SM_3 \xrightarrow{0} N_3$	β	$PK^{10\beta} \cdot PKM^S$					$SSAG'_{1,3}$ $SSAG'_{2,4}$ $SSA^{\circ}_{3,3}$
$TM_{1,2} \xrightarrow{1} N_{1,2}$ $TM_{3,4} \xrightarrow{1} N_{3,4}$ $TM_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha} \cdot PKM^T$					$TSAG'_{1,2}$ $TSAG'_{3,4}$ $TSA_{3,3}$
$UM_{1,2} \xrightarrow{1} N_{1,2}$ $UM_{3,4} \xrightarrow{1} N_{3,4}$ $UM_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha} \cdot PKM^U$					$USAG_{1,2}$ $USAG_{3,4}$ $USA^{\circ}_{3,3}$
$VM_{1,2} \xrightarrow{1} N_{1,2}$ $VM_{3,4} \xrightarrow{1} N_{3,4}$ $VM_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha} \cdot PKM^{\overline{VF}}$					$VSAG'_{1,2}$ $VSAG'_{3,4}$ $VSA^{\circ}_{3,3}$

MEMORY STROBE INTO N-REGISTER

FIG 12-2

HM 6-24-60

N-REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$\rightarrow M_{2,1}$ $\rightarrow M_4$	α	$PK^{10\alpha}$		$\cdot\left(PKM^{LEGAL} + PI_2' \cdot PI_5^0\right)$			
$\rightarrow M_{2,1}$	α	$PK^{25\alpha} \cdot PKIR^{JX}$ $QK^{01\alpha} \cdot QKIR^{IX}$ $CSK^{01\alpha}$		$\cdot\left(EB^0 + \overline{XJ}\right)$			

N-REGISTER CLEARING PULSE

FIG 12-3

HM 6-24-60

N-REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	MEMORY	OTHERS
$E_{2,1} \xrightarrow{1} N_{2,1}$	α	$PK^{11\alpha}$ ·		$\cdot\left(PK^{VFF} + PI_2^1 \cdot PI_5^0\right)$			
		$QK^{15\alpha}$ ·	$QKIR^{ADX}$				$\cdot E^1_{1,2;f}$
		$QK^{21\alpha}$ ·	$QKIR^{AUX}$				
		$QK^{21\alpha}$ ·	$QKIR^{X}$	$\cdot QKIR^0_{OP_2}$			
$E_{4,3} \xrightarrow{1} N_{4,3}$	α	$PK^{11\alpha}$		$\cdot PK^{VFF} + PI_2^1 \cdot PI_5^0$			$\cdot E^1_{3,4;f}$
$E_3 \xrightarrow{0} N_3$	α	$PK^{11\alpha}$		$\cdot PK^{VFF} + PI_2^1 \cdot PI_5^0$			$\cdot E^0_{3;f}$

E-REGISTER TRANSFER INTO N-REGISTER
(3RD QUARTER IS A JAM TRANSFER)

FIG 12-4

HM 6-24-60

P-REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIMELEVEL	INSTRUCTION	OTHERS	TIMELEVEL	INSTRUCTION	OTHERS
$\overline{P1}+1 \longrightarrow P$	α	$PK^{27\alpha} \cdot PKIR^{SKX}$ $PK^{28\alpha} \cdot PKIR^{SKX}$ $PK^{31\alpha} \cdot PKIR^{SKM}$ $PK^{31\alpha} \cdot PKIR^{SKM}$ $PK^{31\alpha} \cdot PKIR^{SKD}$ $PK^{24\alpha}$		$\cdot PKIR_{cf2}^{0} \cdot PKIR_{cf3}^{1} \cdot XJ$ $\cdot PKIR_{cf3}^{1} \cdot XJ$ $\cdot PKIR_{cf4}^{0} \cdot PKIR_{cf5}^{1} \cdot \overline{E_J}$ $\cdot PKIR_{cf4}^{1} \cdot \overline{(PKIR_{cf5}^{1} \cdot \overline{E_J})}$ $\cdot E^{SKIP\ \overline{ZERO}}$		$P_{2.8-1.1}$	
$XA_1 \overset{J}{\diamond\!\!\rightarrow} P_1$	α			$(XA \overset{J}{\diamond\!\!\rightarrow} P) \cdot (\overline{AL} + AUTO\ START)$		XA_{1J}	
$XA_{2.8-2.1} \overset{J}{\diamond\!\!\rightarrow} P_{2.8-2.1}$	α			$(XA \overset{J}{\diamond\!\!\rightarrow} P) \cdot (\overline{AL} + AUTO\ START)$		$XA_{2.8-2.1}$	
$XA_{2.9} \overset{J}{\diamond\!\!\rightarrow} P_{2.9}$	α	$CSK^{04\alpha}$		$(XA \overset{J}{\diamond\!\!\rightarrow} P) \cdot (\overline{AL} + AUTO\ START)$ $(\overline{AL} + AUTO\ START)$		$XA_{2.9}^{0}$	

WHERE:

$XA \overset{J}{\diamond\!\!\rightarrow} P \supset PK^{25\alpha} \cdot PKIR^{JX} \cdot EB^{0} \cdot XJ + CSK^{04\alpha} + PK^{31\alpha} \cdot (PKIR^{JMP} + \Delta EJ)$

$XJ = [\, X_{1.1}^{0} + \cdots X_{1.9}^{0} + X_{2.1}^{0} + \cdots X_{2.8}^{0} \,] \cdot [X_{2.9}^{1}]$

X-ADDER TRANSFER INTO P-REGISTER

FIG 12-5

HM 6-24-60

Q-REGISTER LOGIC							
PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$XA_{1,2} \xrightarrow{f} Q_{1,2}$	α	$QK^{00\alpha}$ · · QI^{START} $PK^{00\alpha}$ · · $PI^{START_2} \cdot PI_2'$			$XA_{1,2,f}$		

X-ADDER TRANSFER INTO Q-REGISTER

FIG 12-6

HM 6-24-60

K-REGISTER LOGIC							
PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$N_{3.6-3.1} \xrightarrow{T_P} K_{3.6-3.1}$	α	$CSK^{03\alpha}$			$N_{3.6-3.1}$		

N-REGISTER TRANSFER INTO K-REGISTER

FIG 12-7

HM 6-24-60

XPS & X-BUFFER REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				MEMORY GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIMELEVEL	MEMORY	OTHERS
$XM_{P,29-1.1} \xrightarrow{J} X_{P,29-1.1}$	$\propto$	$PK^{13\propto}$ ·		$\cdot \overline{N_J^{00}} \cdot (XPS^{\circ} + \overline{K^{EQJ}})$			
		$CSK^{02\propto}$ ·		$\cdot \overline{N_J^{00}} \cdot (XPS^{\circ} \overline{+ K^{EQJ}})$			CORE $POLSE_{P,2.9-1.1}$
$L1 \rightarrow XPS$	$\propto$	$CSK^{04\propto}$					
$L0 \rightarrow XPS$	$\propto$	$PK^{15\propto}$		K^{EQJ}			

X-MEMORY TRANSFER INTO X-BUFFER REGISTER

FIG 12-8

#M 6-28-60

X-BUFFER REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$P_{2.9-1.1} \xrightarrow{\frac{1}{2}} X_{2.9-1.1}$	α	$PK^{31\alpha} \cdot PKIR^{JMP} \cdot PKIR_{cf_2^{1,0}} (XPAL_{SOP} + XPAL^{0})$	$CSK^{04\alpha}$.	$\cdot XPAL_{SOP} + XPAL^{0})$			$\cdot P_{2.9-1.1}$

P-REGISTER JAM INTO X-BUFFER REGISTER

FIG 12-9

HM 6-28-60

X-BUFFER REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$XA_{29-1.1} \xrightarrow{\delta} X_{29-1.1}$	α	$PK^{26\alpha}$	$\cdot PKIR^{SKX}$	$\cdot(XPAL_{SUP} + XPAL^{\circ})$			$\cdot XA_{2.9-1.1}$
		$PK^{30\alpha}$	$\cdot PKIR^{SKX}$	$\cdot(XPAL_{SUP} + XPAL^{\circ})$			
		$PK^{31\alpha}$	$\cdot PKIR^{JX}$	$\cdot(XPAL_{SUP} + XPAL^{\circ})$			
		$QK^{22\alpha}$	$\cdot QKIR^{LD} \cdot QKIR^{X}$	$\cdot(XPAL_{SUP} + XPAL^{\circ})$			
		$QK^{31\alpha}$	$\cdot QKIR^{AUX}$	$\cdot(XPAL_{SUP} + XPAL^{\circ})$			

X-ADDER REGISTER JAM INTO X-BUFFER REGISTER

FIG 12-10

#M 6-28-60

X-BUFFER REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
1 → XP		PK$^{13\alpha}$ ·		$\cdot\left(N_j^{00} + K^{EQJ} \cdot XPS'\right)$			
0 → $X_{2.9-1.1}$		CSK$^{02\alpha}$.		$\cdot\left(N_j^{00} + K^{EQJ} \cdot XPS'\right)$			

X-PARITY FLIP-FLOP SET PULSE
X-BUFFER REGISTER CLEAR PULSE

FIG 12-11
HM 6-28-60

X-BUFFER REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$\xrightarrow{C}$ X	α	$PK^{15\alpha} \cdot PKIR_{op}^{ox} \cdot PKIR_{op}^{x6} \cdot PI_2^{\circ}$ $PK^{15\alpha} \cdot PKIR^{skx} \cdot (PKIR_{cf_1}^{\circ} \; PKIR_{cf_3}^{1}) \cdot PI_2^{\circ}$ $PK^{15\alpha} \cdot PKIR^{skx} \cdot (PKIR_{cf_1}^{1} \cdot PKIR_{cf_3}^{\circ}) \cdot PI_2^{\circ}$ $PK^{25\alpha} \cdot PKIR_{op}^{ox} \cdot PKIR_{op}^{x6} \cdot EB^{\circ}$ $PK^{27\alpha} \cdot PKIR^{skx} \cdot PKIR_{cf_3}^{1}$ $PK^{31\alpha} \cdot PKIR^{skx} \cdot PKIR_{cf_1}^{1}$				

X-BUFFER REGISTER COMPLEMENT PULSE

FIG 12-12

HM 6-28-60

X-ADDER SELECT LOGIC

PULSE	RD PULSE	REGISTER DRIVER LOGIC			PULSE GATE LOGIC		
		TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
1 → XAS	α			PRESET CE	$PK^{14\alpha} \cdot (PKIR^{IND} \cdot PI_2^0 + PI_5^1)$ $PK^{26\alpha} \cdot PKIR^{JX}$ $QK^{10\alpha} \cdot QKIR^{ADX}$ $QK^{10\alpha} \cdot QKIR^{AUX}$ $QK^{10\alpha} \cdot QKIR^{X}$ $CSK^{02\alpha}$: $\cdot \overline{M_f^{00}}$		
0 → XAS	α			PRESET CE	$PK^{14\alpha} \cdot (PKIR^{IND} \cdot PI_2^0 + PI_5^1)$ $QK^{21\alpha} \cdot QKIR^{LD} \cdot QKIR^{X}$ $CSK^{02\alpha}$: $\cdot M_f^{00}$		

X-ADDER SELECT FLIP-FLOP LOGIC

FIG 12-13

#M 6-27-60

X-ADDER CARRY LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$\overline{1}$ → XAC	α			PRESET$_{ct}$	$PK^{14\alpha} \cdot PKIR^{SKX}$ $PK^{25\alpha} \cdot PKIR^{JX}$ $PK^{26\alpha} \cdot PKIR^{SKX}$ $QK^{01\alpha} \cdot QKIR^{IX}_{OP}$ $QK^{14\alpha} \cdot (QKIR^{ADX} + QKIR^{ADX})$ $CSK^{01\alpha}$		
$\overline{0}$ → XAC	α			XAC'			

X-ADDER CARRY FLIP-FLOP LOGIC

Fig 12-14

HM 6-27-60

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$N_{4.3-3.7} \rightarrow PKIR_{op6-1}$	α	$PK^{12\alpha}$		PI_2^0		$N_{4.3-3.7}$	
$N_{4.9} \rightarrow PKIR_h$	α	$PK^{12\alpha}$		PI_2^0		$N_{4.9}$	

PKIR$_{op}$ REGISTER LOGIC

N-REGISTER TRANSFER INTO PKIR$_{op}$-REGISTER & HOLD BIT

FIG 12-15

HM 6-27-60

QKIR$_{op}$ REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
PKIR$_{op6-1}$ $\xrightarrow{i}$ QKIR$_{op6-1}$	α	QK$^{00\alpha}$		$\cdot$ QI^{START}		PKIR$_{op6-1}$	

PKIR$_{op}$ - REGISTER TRANSFER INTO QKIR$_{op}$ - REGISTER

FIG 12-16

HM 6-27-60

$AKIR_{op}$ REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$N_2 \rightarrow AKIR_{op}$	α	$AK'_{\alpha,0}$		$\cdot N \rightarrow AKIR$			$\cdot N_{2.6-2.1}$
$QKIR_{op} \rightarrow AKIR_{op}$	α			$QKIR \rightarrow AKIR$			$\cdot QKIR_{op\,6-1}$

WHERE -

$N \rightarrow AKIR = PIL^{25\alpha} \cdot PKIR^{OPR\ AE}$

$QKIR \rightarrow AKIR = QK^{13\alpha} \cdot QKIR^{AK}$

N_2-REGISTER AND $QKIR_{op}$-REGISTER TRANSFER INTO $AKIR_{op}$ REGISTER

FIG 12-17

HM 6-28-60

												PKIR REGISTER LOG	
PULSE		REGISTER DRIVER LOGIC			DELAY	G.P.A. LOGIC			PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHER	DELAY	TIME LEVEL	INSTRUCTION	OTHER	TIME LEVEL	INSTRUCTION	OTHER
$N_{4.8-4.4} \rightarrow PKIR_{cf-1}$	α	$PK^{13\alpha}$		$\cdot PI_2^0$					$N_{4.8-4.4}$		
$\overline{PKIR_{cf}}+1 \rightarrow PKIR_{cf}$		$FK^{0\alpha}$ $\overline{FK^{0\alpha}}$		$\cdot FKB^0 \cdot$ START FK $\cdot FK^0_{\alpha,1} \cdot PKIR^{FF} \cdot (FP^{ODD}_{10} + PKIR^{00}_{cf} + FPAL_{sup})$	0.1 MS	$\overline{FK^{0\alpha}}$					

Y-REGISTER TRANSFER INTO $PKIR_{cf}$ REGISTER

$PKIR_{cf}$ REGISTER PULSE LOGIC

FIG 12-18

HM 12-20-60

PKIR$_{cf}$ REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				DELAY	G. P. A. LOGIC			PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHER	DELAY	TIME LEVEL	INSTRUCTION	OTHER	TIME LEVEL	INSTRUCTION	OTHER
↓ QKIR$_{cf}$	α	$\frac{FIL^{0\alpha}}{FIL^{00\alpha}}$		FIL 8° ← START FIL $FIL^{0}_{\alpha.1} \cdot PKIR^{FF} \cdot (FP^{ODD}_{10} + PKIR^{00}_{cf} + FPAL_{SUP})$	0.4 MS			$\overline{PKIR^{00}_{cf}} \cdot \overline{PKIR^{LF}}$			$PKIR_{cf_5}$

CF-MEMORY INTO QKIR$_{cf}$ REGISTER

FIG 12-19

HM 12-20-60

$QKIR_{cf}$-REGISTER LOGI

	REGISTER DRIVER LOGIC			DELAY	GATED PULSE AMPLIFIER LOGIC			PULSE GATE LOGIC		
		TIME LEVEL	INSTRUCTION	OTHERS	DELAY	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$E_1 \xrightarrow{1} QKIR_{cf}$	α	$FK^{0\alpha}$ / $\overline{FK^{0\alpha}}$	· $FIL\,8^{\circ}$ · START ◇ FK · $FIL^{0}_{\alpha,1} \cdot PKIR^{FF} \cdot \left(FP^{ODD}_{10} + PKIR^{00}_{cf} + FPAL_{SUP}\right)$	0.4 μsec	$\overline{PKIR^{00}_{cf}} \cdot PKIR^{LF}$				$E'_{1 \cdot j}$		

E_1 - REGISTER TRANSFER INTO $QKIR_{cf}$-REGISTER

FIG 12-20
HA 6-29-60

QKIR$_{cf}$ REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$N_{3.6-3.1} \xrightarrow{1} QKIR_{cf\ 9-4}$	α	$PK^{09\alpha}$ ·		· PI_2' · XAS^0			$N'_{3.6-3.1}$
$N_{3.6-3.5} \xrightarrow{1} QKIR_{cf_{2-1}}$	α	$QK^{01\alpha}$ · $PKIR^{SKM}$					$\overline{N_{3,6,5}} - 1$

N - REGISTER TRANSFER INTO QKIR$_{cf}$ - REGISTER

FIG 12-21

HM 6-29-60

	REGISTER DRIVER LOGIC				DELAY	GPA PULSE LOGIC			GPA LEVEL LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	DELAY	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$\to QKIR_{CFP,9-3}$	α	$QK^{01\alpha}$	$\cdot PKIR^{SKM}$								
$\to QKIR_{CFP,9-3}$ $\to QKIR_{CF2,1}$	α	$FK^{0\alpha}\cdot$	$\cdot FKB^{0}\cdot \overline{START} \Diamond FK$		0.1 μs					$(PKIR^{LP} + COMP^{0})$	
		$\overline{FK^{0\alpha}}\cdot$	$\cdot FK^{0}_{\alpha,1}\left[PKIR^{FF}\cdot\left(FP^{ODD}_{10} + PKIR^{00}_{CF} + FPAL_{SUP}\right)\right]$								
		$FK^{0\alpha}\cdot$	$\cdot FKB^{0}\cdot \overline{START} \Diamond FK$		0.5 μs		$PKIR^{00}_{CF}$				
		$\overline{FK^{0\alpha}}\cdot$	$\cdot FK^{0}_{\alpha,1}\left[PKIR^{FF}\cdot\left(FP^{ODD}_{10} + PKIR^{00}_{CF} + FPAL_{SUP}\right)\right]$								
		$PK^{01\alpha}\cdot$	$\cdot PI^{1}_{2}\cdot XAS^{0}$								

$QKIR_{CF}$ - REGISTER CLEARING PULSE

FIG 1

HM 6-

DEDLEVEL	PKIR$_{op}$ FLIP-FLOP & DECODER LEVELS	OP CODE FORM
$(PKIR)^{F}$	$PKIR^{LF} + PKIR^{SF}$	SPF, SPG, FLF, FLG
$(PKIR)^{FF}$	$(PKIR)^{F} \cdot (PKIR)^{X2}_{op}$	SPG, FLF
$(PKIR)^{LF}$	$(PKIR)^{ZX}_{op} \cdot \left[(PKIR)^{X1}_{op} + (PKIR)^{X2}_{op} \right]$	SPF, SPG
$(PKIR)^{SF}$	$(PKIR)^{3X}_{op} \cdot \left[(PKIR)^{X1}_{op} + (PKIR)^{X2}_{op} \right]$	FLF, FLG
$(PKIR)^{IOS}$	$N^{0}_{2.7} \cdot N^{0}_{2.8} \cdot (PKIR)^{OPR}$	IOS
$(PKIR)^{OPR\,AE}$	$N^{1}_{2.7} \cdot N^{0}_{2.8} \cdot (PKIR)^{OPR}$	AOP
$(PKIR)^{AE}$	$(PKIR)^{4X}_{op} \cdot \left[(PKIR)^{X1}_{op} + (PKIR)^{X2}_{op} \right] + (PKIR)^{5X}_{op} \cdot \left[(PKIR)^{X4}_{op} + (PKIR)^{X5}_{op} \right] + (PKIR)^{DEF} \cdot (PKIR)^{1}_{OP5} \cdot \left[(PKIR)_{op_3^1} + (PKIR)_{op_6^1} \right] + (PKIR)^{OPRAE} + (PKIR)^{JA}$	ITA, UNA, EXA, INS, SPG, LDB, LDD, FLG, STB, STD, INS, TSD, SAB, DIV, SUB, CYA, CYB, CAB, NOA, DSA, NAB, A AOP, JPA, JNA, JOV — SCA, SCB, SAB, TLY, DIV, MUL
$(PKIR)^{AK}$	$(PKIR)^{4X}_{op} \cdot (PKIR)^{1}_{op_3} + (PKIR)^{0X}_{op} + (PKIR)^{SKX} + (PKIR)^{DEF}$	JOV, JPA, JNA, IOS, AOP, JMP, JPX, JNX, SKX, 00-03, 13, 23, 33, 45, 63, 73, 50-53
$(PKIR)^{IND}$	$(PKIR)^{SKX} \cdot \left[(PKIR)_{cf_2^1} + (PKIR)_{cf_3^1} \right] + (PKIR)^{JMP} \cdot (PKIR)_{cf_1^1} + (PKIR)_{OP5^1} + (PKIR)_{OP6^1}$	^{XXXIX}SKX, ^{XXIXX}SKX, ^{XXXXI}JMP, LDE, SPF, SPG, LDA, LDB, LDC, LDD, STE, FLF, FLG, STA, STB, STC, STD, ITE, ITA, UNA, SED, JOV, JPA, JNA, EXA, IN. PCM, TSD, CYA, CYB, CAB, NOA, DSA, NAB, ADD, SCA, SCB, SAB, TLY, DIV, MUL, SUB
$(PKIR)^{XA}$	$(PKIR)^{JMP} \cdot (PKIR)_{cf_2^1} + (PKIR)^{JX} + (PKIR)^{SKX}$	JPX, JNX, SKX, ^{XXXIX}JMP
$(PKIR)^{JX}$	$(PKIR)^{0X}_{op} \cdot \left[(PKIR)^{X6}_{op} + (PKIR)^{X7}_{op} \right]$	JPX, JNX
$(PKIR)^{JA}$	$(PKIR)^{JPA} + (PKIR)^{JNA} + (PKIR)^{JOV}$	JPA, JNA, JOV
$(PKIR)^{DIS}$	$(PKIR)^{OPR} + (PKIR)^{TSD} + (PKIR)^{JMP} + (PKIR)^{JA} + (PKIR)^{JX} + (PKIR)^{SED} + (PKIR)^{SKM} + (PKIR)^{SKX}$	OPR, TSD, JMP, JOV, JPA, JNA, JPX, JNX, SED, SKM, SKX
$\overline{(PKIR)^{DEF}}$	$(PKIR)^{0}_{OP3} \cdot \left[(PKIR)^{0X}_{op} + (PKIR)^{5X}_{op} \right] + (PKIR)^{X3}_{op} \cdot \overline{(PKIR)^{SED}} + (PKIR)^{OPR} \left[N^{1}_{2.8} \cdot N^{0}_{2.9} \right] + (PKIR)^{UDF_2}$	00-03, 50-53, 13, 23, 33, 53, 63, 73, AOP, UDF_2
$(PKIR)^{DIS\,RER}$	$(PKIR)^{1}_{cf_5} \cdot \left\{ (PKIR)^{IOS} \cdot \left[\overline{N^{101}_{2.6-2.4}} + \overline{K^{EQJ}} \right] + (PKIR)_{cf_4^0} \cdot \left[(PKIR)^{SKX} + \overline{(PKIR)^{EQJ}} \right] \right\} + (PKIR)^{JMP} \cdot (PKIR)_{CF_5^1} + (PKIR)^{TSD} + (PKIR)^{JX} \cdot XJ$	^{IXXXX}IOS$\left(\overline{N^{101}_{2.6-2.4}} + \overline{K^{ERJ}}\right)$, ^{XOXXX}SKX$\left(\overline{K^{ERJ}}\right)$, ^{IXXXX}JMP, TSD, JPX^{XJ}, JN^{XJ}

PKIR$_{op}$ CLASS DECODER LEVEL LOGIC

FIG 12-37
#M 12-28

DECODED LEVEL	QKIR$_{op}$ FLIP-FLOP & DECODER LEVELS	OP CODE FORM
$(QKIR)^{ST}$	$\{[QKIR_{op}^{3X}\cdot(QKIR_{op}^{X0}+QKIR'_{op3})]+[QKIR_{op}^{X4}\cdot(QKIR_{op}^{3X}+QKIR_{op}^{1X})]+[(QKIR_{op}^{X6}\cdot QKIR_{op}^{1X})]\}$	STE, FLF, EXA, EXX, DPX
$(QKIR)^{LD}$	$\{[(QKIR)_{op}^{2X}\cdot(\quad ditto \quad)]+[\quad ditto \quad]+[(QKIR)_{op}^{X1}\cdot(QKIR)_{op}^{1X}+(QKIR)^{AK}]\}$	LDE, SPF, EXA, EXX, RSX, AIL
$(QKIR)^{STORE}$	$\{[(QKIR)_{op}^{1X}\cdot(QKIR)_{op3}^{1}]+[(QKIR)_{op}^{5X}+(QKIR)_{op}^{3X}]\}$	EXX, ADX, DPX, SKM, STE, FLF, FLCF, STA, STB, STC, STD + TXA, INS, PCM, TSD
$(QKIR)^{LOAD}$	$[\overline{(QKIR)^{STORE}}]$	ISO, ADP, JMP, JPX, JNX, AUX, RSX, SKX, LDE, SPF, SPG, LDA, LDB, LDC, LDD, ITE, ITA, UM, SED, JDV, JPA, JNA, CYA, CYB, CAB, NOA, DSA, NADD, SCA, SCB, SAB, TLY, DIV, MUL, SUB, 00–03, 13, 45, 50–53, 63, 73
$(QKIR)^{FL}$	$[(QKIR)^{FLG}+(QKIR)^{FLF}]$	FLG, FLF
$(QKIR)^{X}$	$\{(QKIR)^{1X}\cdot[(QKIR)_{op}^{X1}+(QKIR)_{op1}^{0}\cdot(QKIR)_{op3}^{1}]\}$	RSX, EXX, DPX
$(QKIR)^{MESK}$	$[(QKIR)^{AK}\cdot\overline{(QKIR)^{DSA}}\cdot\overline{(QKIR)_{op}^{X7}}]$	
$(QKIR)^{AK}$	$[(QKIR)_{op6}^{1}\cdot(QKIR)_{op5}^{1}]$	CYA, CYB, CAB, NOA, DSA, NAB, ADD, SCA, SCB, SAB, TLY, DIV, MUL, SUB
$(QKIR)^{A}$	$\{(QKIR)_{op}^{X4}\cdot[(QKIR)_{op}^{2X}+(QKIR)_{op}^{3X}+(QKIR)_{op6}^{1}\cdot(QKIR)_{op4}^{1}]\}$	LDA, STA
$(QKIR)^{B}$	$\{(QKIR)_{op}^{X5}\cdot[(QKIR)_{op}^{2X}+(QKIR)_{op}^{3X}+(QKIR)_{op}^{5X}]\}$	LDB, STB, INS
$(QKIR)^{C}$	$\{(QKIR)_{op}^{X6}\cdot[(QKIR)_{op}^{2X}+(QKIR)_{op}^{3X}]\}$	LDC, STC
$(QKIR)^{D}$	$\{(QKIR)_{op}^{X7}\cdot[(QKIR)_{op}^{2X}+(QKIR)_{op}^{3X}]+[(QKIR)^{AK}\cdot\overline{(QKIR)^{A}}]\}$	LDD, STD, $AK\cdot(\overline{LDA}+\overline{STA})$
$(QKIR)^{E}$	$\{(QKIR)_{op}^{X0}\cdot[(QKIR)_{op}^{2X}+(QKIR)_{op}^{3X}]+[(QKIR)^{ITE}+(QKIR)^{SED}]\}$	LDE, STE, ITE, SED

QKIR$_{op}$ CLASS DECODER LEVEL LOGIC

FIG
#A

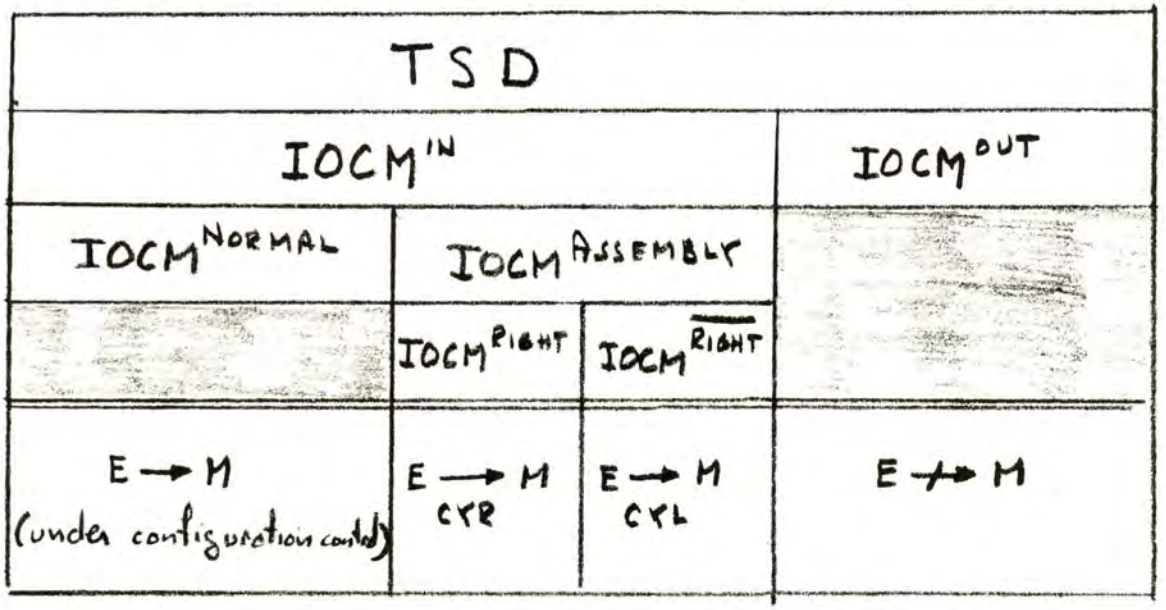


FIG. 13-4 TRANSFER LOGIC FROM E TO M DURING TSD

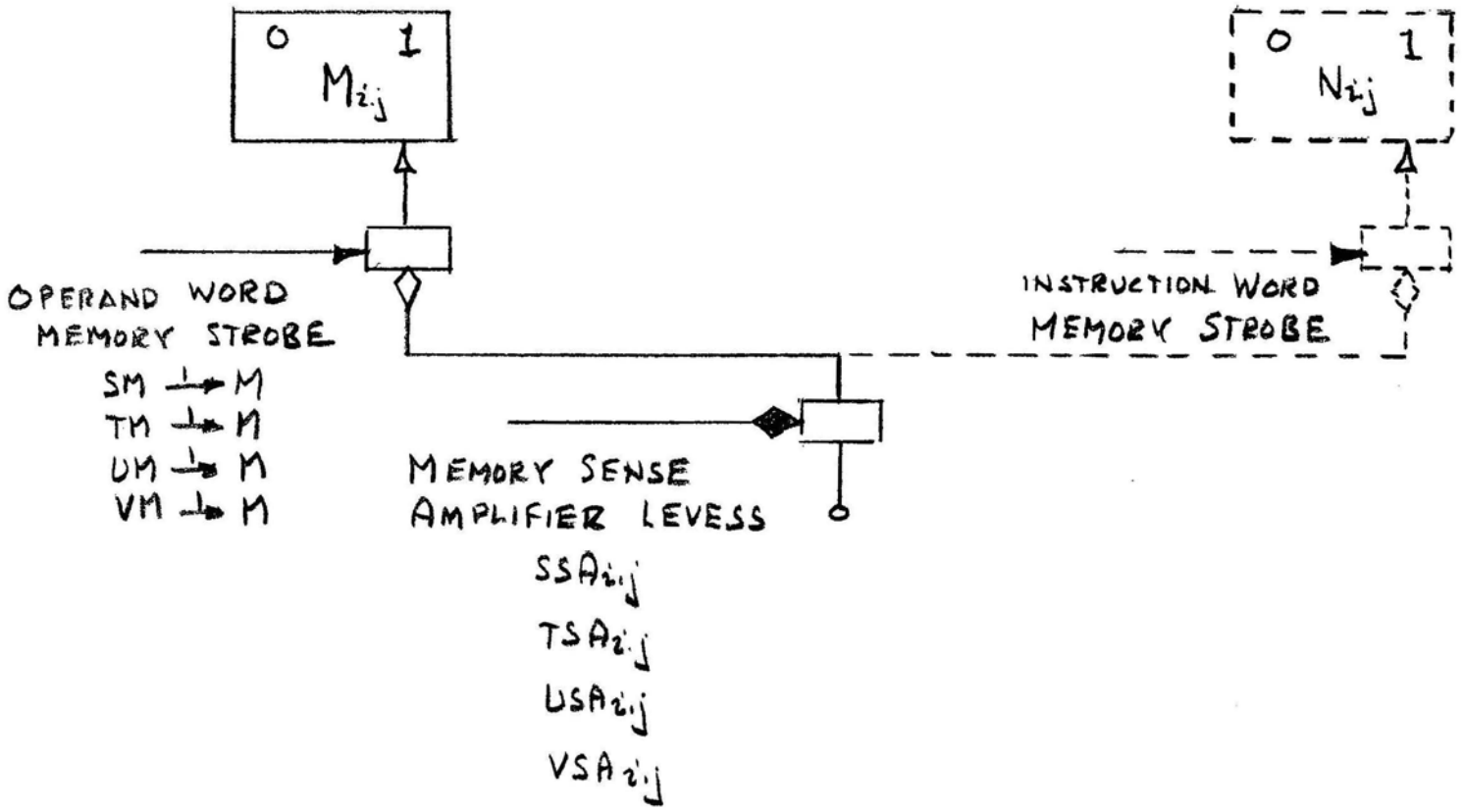
	QK^{18d}				$\overline{QK^{18d}}$	
PULSE	TIME LEVEL	INSTRUCTION	CF BIT CONTROL	PARITY ALARM	TIME LEVEL	SEE FIG 13-5
$\rightarrow M_{4.10}$	$(QK^{18d} \cdot QKIR^{SKM}) \cdot (PKIR'_{CF_2} \cdot PKIR^{0}_{CF_1}) \cdot \overline{MPA}$				$+ \overline{QK^{18d}} \cdot (\rightarrow M_{4,3,2,1})$	
$\rightarrow M_{4.10}$	(← Ditto →) $\cdot (PKIR^{0}_{CF_2} \cdot PKIR'_{CF_1}) \cdot \overline{MPA}$					
$\rightarrow M_{4.10}$	(← Ditto →) $\cdot (PKIR'_{CF_2} \cdot PKIR'_{CF_1}) \cdot \overline{MPA}$					
$\rightarrow E_j$	(← Ditto →) $\cdot (PKIR'_{CF_2} \cdot PKIR^{0}_{CF_1})$					
$\rightarrow E_j$	(← Ditto →) $\cdot (PKIR^{0}_{CF_2} \cdot PKIR'_{CF_1})$					
$\rightarrow E_j$	(← Ditto →) $\cdot (PKIR'_{CF_2} \cdot PKIR'_{CF_1})$					

FIG. 13-3 CLEAR, COMPLMENT AND SET REGISTER DRIVER CONTROL OF M AND E REGISTERS FOR SKM INSTRUCTION.

PULSE	M RD LOGIC
SM $\perp\!\!\!\rightarrow$ M	$QKM^{S} \cdot QK^{10B}$
TM $\perp\!\!\!\rightarrow$ M	$QKM^{T} \cdot QK^{11B}$
UM $\perp\!\!\!\rightarrow$ M	$QKM^{U} \cdot QK^{11B}$
VM $\perp\!\!\!\rightarrow$ M	$QKM^{\overline{VFF}} \cdot QK^{11B}$

FIG 13-2 MEMORY STROBE INTO M REGISTER

REGISTER DRIVER LOGIC



$$[(SM \perp\!\!\!\rightarrow M) \cdot SSA_{i,j}] + [(TM \perp\!\!\!\rightarrow M) \cdot TSA_{i,j}] + [(UM \perp\!\!\!\rightarrow M)] \cdot USA_{i,j} + [(VM \perp\!\!\!\rightarrow M) \cdot VSA_{i,j}]$$

$$\supset \perp\!\!\!\rightarrow M_{i,j}$$

FIG. 13-1 MEMORY STROBE INTO M REGISTER

Thus, neglecting sign extension, the basic difference between a store and load type instruction, as far as the Exchange Element is concerned, is the transfer pulse occurring at $QK^{13\alpha}$. In the store case, $QK^{13\alpha}$ initiates an E ⟶ M transfer. In the load case, $QK^{13\alpha}$ initiates an M ⟶ E transfer.

13-4.3 SIGN EXTENSION. Fig. 13-14 shows the basic concept of sign extension as it applies to the E register. The general rule for extending the sign is also given.

Fig. 13-15 gives a specific example of sign extension. This figure is in reality an extension of Fig. 13-12. The E register was permuted at $QK^{13\beta}$ on Fig. 13-12(d). Fig. 13-15(a) shows that the third quarter of E is cleared at $QK^{14\alpha}$. For the case chosen, the fact that the fourth quarter of E is active ($QKIR^{ACT}{}_4$) and the coupling is f_2 ($f_2 \supset \overline{f_4}$) generates the $QKIR^{EXT\ ACT}{}_3$ level. The logic for clearing E_3 is shown in Fig. 13-15(a).

At $QK^{14\beta}$, the sign of the active fourth quarter is extended into the inactive third quarter by the simple operation of complementing the third quarter of E. The operation and the complement logic are shown in Fig. 13-15(b). Note that, if the sign bit of the fourth quarter were a ZERO ($E^0_{4.9}$), then the complement pulse would not have occurred and the inactive third quarter would have been left with all ZEROS in it.

2) Figs. 13-10 shows that at $QK^{11\beta}$, E is also inversely permuted. This can be seen since the "OP" and "CF" bits are decoded to generate $QKIR^{LD}$ and $QKIR^{PRM}{}_3$ levels. If we assume that this is not a SPF or SPG instruction, then the logic on Fig. 13-10 is satisfied and all the quarters of E will be shifted one quarter to the right at $QK^{11\beta}$, i.e., E will be inversely permuted.

3) Figs. 13-7 and 13-12(c) show that a transfer will occur from M to E under permuted activity control at $QK^{13\alpha}$. All quarters of M are transferred, except the second quarter. This selection occurs because of the $QKIR^{PRM\ ACT}{}_2$ level in the transfer logic. This level is generated by logic that looks at both the permutation and activity required by the instruction and decides in which quarters an M ⟶ E transfer should occur. Note that E now contains the correct data but the data is all shifted one quarter to the right.

4) Figs. 13-10 and 13-12(d) indicate that at $QK^{13\beta}$ (0.2 microsecond after the M⟶E transfers) a direct permutation occurs in which data is finally shifted to the left into the desired quarters. Compare the E register in Figs. 13-12(a) and (d).

The sign extension process which follows step 4 is described in 13-4.3.

13-4.2 STORE TYPE INSTRUCTIONS. The store type instruction will now be examined. Fig. 13-13(a) shows the effect of the instruction on the M and E register. The M register contains the content of the E register shifted one quarter to the right, except for the second quarter of M which contains whatever data was in it at the beginning of the instruction.

The sequence of transfers that accomplishes the store instruction is as follows:

1) If the instruction is STA, the content of A is placed in E at $QK^{11\alpha}$ as indicated on Fig. 13-8.

2) Figs. 13-13(b) and (d) are exactly the same transfers and occur at exactly the same time as the transfers shown in Figs. 13-12(b) and (d).

3) Figs. 13-5 and 13-13(c) indicate that a transfer under permuted activity control from E to M occurs at $QK^{13\alpha}$. Note that at the end of this transfer, M contains the word that is to be stored in memory. E however must still be unscrambled by the direct permutation pulse that occurs at $QK^{13\beta}$.

13-4 ILLUSTRATIVE EXAMPLES

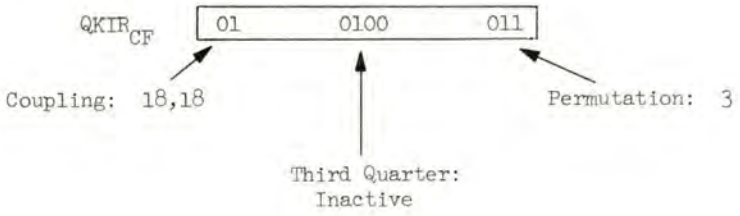
Some examples will be given to illustrate the use of the register driver logic tabulations given in this chapter. These examples illustrate the configuration and sign extension operation which is processed in the Exchange Element. In these examples, the transfers that occur in the Exchange Element during a configured load and a store type instruction will be examined. Only those transfers illustrating configuration will be examined in detail.

In the example, it is assumed that the programmer specified a configuration that caused the third quarter to be inactive and that calls for an f_2 (18,18) subword form. The specified configuration makes use of permutation 3.

This permutation has the effect of shifting the quarters of M one quarter to the left into the E register, during load type instructions; and the quarters of E one quarter to the right into M, during store type instructions. In the example, it is also assumed that the sign bit of the active quarter of the partially active subword is a ONE in the load type instruction.

Let m_i and e_i represent the original contents of the quarters of the M and E registers, respectively.

For the examples cited, the configuration bits are as follows:



13-4.1 LOAD TYPE INSTRUCTIONS. The load type instruction will be examined first. Fig. 13-12(a) shows the effect of the instruction (neglecting the effects of sign extension) on the M and E registers. At the end of the instruction, the operand appears in M just as it was read out of memory. The E register contains the word in M shifted one quarter to left, except for the third quarter of E which contains whatever was in E_3 before the instruction began.

The sequence of transfers that accomplishes this operation is as follows:

1) Fig. 13-2 indicates that the operand is usually strobed out of memory into M at $QK^{11\beta}$.

Several observations can be made by looking at the individual terms for directly and inversely permuting the quarters of E. First observe that most of the instructions are included in the $QK^{11\beta}$, $QK^{13\beta}$ and $QK^{18\beta}$ terms. However, while the load and store type instructions go through $QK^{11\beta}$ and $QK^{13\beta}$, they do not go through $QK^{18\beta}$. Thus, the $QK^{18\beta}$ term will include far fewer instructions than the factor ANDed with $QK^{18\beta}$. Note that, in most configured instructions, both an inverse and direct permutation will occur.

13-3.6 CLEAR AND COMPLEMENT E REGISTER. These operations are involved in combination in the process of sign extension (see Fig. 13-11). The logic involved in extending the sign of an active quarter into the inactive quarters of a partially active subword causes the inactive quarters to be cleared at $QK^{14\alpha}$. If the sign bit of the active quarter is a ONE, the inactive quarters are then complemented at $QK^{14\beta}$. The sign extension control term of the register driver logic includes factors which take into account the activity and coupling involved in the instruction.

In the case of the COM instruction, the sign is extended at $QK^{14\beta}$, and then the active quarters themselves are complemented at $QK^{15\beta}$. During an INS or ITA instruction, the content of the entire E register is complemented as a basic step in the execution of the instruction.

Earlier in the chapter, it was mentioned that the content of XA is copied into $E_{2,1}$ at $QK^{11\alpha}$, during the execution of X Memory type instructions (RSX, ADX, EXX or DPX). If the sign bit in XA ($X_{2.9}$) is a ONE, $E_{4,3}$ will be complemented at $QK^{10\beta}$ as part of the sign extension logic in the E register. Effectively, the content of the X register is extended to fill the E register.

Certain miscellaneous instructions require that the E register be cleared before a data transfer into the E register can take place. This occurs:

1) For most instructions using $QK^{10\alpha}$ in an operand cycle.
2) As a preliminary step to $N_{3.6 - 3.1} \xrightarrow{1} E_{3.6 - 3.1}$, during an IOS when $PKIR_{CF_1}$ is a ONE.
3) For all instructions involving the V_{FF} Memory, except when the instruction is placed in E.
4) As a preliminary step to placing data in E, during a deferred address cycle.
5) As a preliminary step to placing the contents of Q in E, during a JMP instruction when the $PKIR_{CF_4}$ bit is a ONE.

13-3.4 MISCELLANEOUS USES OF E REGISTER. Fig. 13-9 also shows the register driver logic involved in several miscellaneous transfers into E. The transfers occur:

1) During a deferred address cycle, when the content of $QKIR_{CF_{9-4}}$ is placed in $E_{3.6 - 3.1}$, and the content of XA (X Adder register) is placed in $E_{2,1}$.

2) During a change of sequence cycle, when the content of $N_{3.6 - 3.1}$ is placed in $E_{3.6 - 3.1}$; the content of $K_{3.6 - 3.1}$ is placed in $E_{4.6 - 4.1}$; and the content of P is stored in $E_{2,1}$.

3) During the X Memory instructions, when the content of XA is placed in $E_{2,1}$.

4) During a JMP instruction (if $PKIR_{CF_4}$ contains a ONE) when the content of Q is placed in $E_{2,1}$. Also during a JMP (if $PKIR_{CF_3}$ contains a ONE), when the content of P is placed in $E_{4,3}$.

5) During an IOS instruction (if $PKIR_{CF_1}$ contains a ONE), when the content of $N_{3.6 - 3.1}$ is placed in $E_{3.6 - 3.1}$.

6) During a JPA, JNA or JOV instruction (if the jump conditions are satisfied), when the content of P is placed in $E_{2,1}$.

7) During a FLF or FLG instruction, when the content of $QKIR_{CF_{9-1}}$ is placed in $E_{4.9 - 4.1}$.

13-3.5 INTERCHANGE OF E REGISTER QUARTERS. The register driver logic for this interchange is shown in Fig. 13-10.

There are two general circumstances in which the quarters of E are interchanged. In one case, the interchange is a by-product of configuration control and is indirectly under the control of the programmer. The programmer may select one of several configurations for the same basic instruction. The interchanges in the E register for the configuration will then take place. In the second case, the interchange is a basic step in the instruction. During store and load type instructions involving the F Memory (SPF, SPG, FLF and FLG), the quarters of the E register are interchanged in such a way as to cycle the content of the E register either one quarter to the right or one quarter to the left. The interchange is initiated by the FK counter.

During the execution of an ITE instruction, the ZEROS of M are transferred into E at $QK^{13\alpha}$, but not the ONES.

2) Broadside Transfers. A jam transfer from M to E will occur for most store type instructions at $QK^{21\alpha}$ and for most load type instructions at $QK^{23\alpha}$.

COM, SPF, SPG or a TSD in the ASSEMBLY mode causes a jam transfer to occur from M to E at $QK^{13\alpha}$.

In addition to the above, a jam transfer from M to E occurs whenever the V_{FF} Memory is involved.

Finally a jam transfer from M to E occurs at $PK^{11\alpha}$ during a deferred address cycle.

3) "Exclusive or" Transfer between M and E Under Permuted Control. This transfer occurs twice during an SED instruction. The second transfer has the effect of restoring the E register to its original state because of the logical characteristics of the "exclusive or".

13-3.2 ARITHMETIC ELEMENT TO E TRANSFERS. The register driver logic for these transfers is shown in Fig. 13-8. Two cases exist: either the V_{FF} Memory is or is not involved in the instruction.

1) In the first case, if an instruction is stored in either the A, B, C or D registers, the instruction word will be read into the E register at $PK^{10\alpha}$. Note that if the Arithmetic Element is busy, PK will not get to $PK^{10\alpha}$ until the $\overline{AEB}$ condition is satisfied. If an operand is stored in the Arithmetic Element and that element is not busy, the operand will be read into E at $QK^{03\alpha}$.

2) The second case includes load and store type instructions involving the Arithmetic Element registers. In the case of INS, ITA and UNA, data is transferred specifically from the A register to the E register.

13-3.3 IOBM TO E TRANSFERS. During the execution of a TSD in the IN mode or during the execution of an IOS when $PKIR_{CF_1}$ is a ONE, a jam transfer occurs from the selected $IOBM_i$ (In-Out Buffer Mixer) to E. The register driver logic for this transfer is shown in Fig. 13-9.

2) Transfers Under Permuted Activity Control. In this case, the pulses on the gates between each quarter of E and M are independently controlled by $QKIR^{PRM\ ACT}_{i}$ levels. The contents of E are transferred to M under permuted activity control during all the store type instructions at $QK^{13\alpha}$. In the case of the INS instruction, the ZEROS are transferred at $QK^{13\alpha}$ and the ONES are transferred at $QK^{19\alpha}$. Even though the store type instructions are included, these instructions do not go through $QK^{19\alpha}$, so that this condition is satisfied by only a few instructions.

If a TSD is executed in the NORMAL mode or if $PKIR_{CF_3}$ is equal to ONE during an SKM instruction, the logic is satisfied and an E to M transfer occurs under permuted activity control.

3) Transfer of ZEROS from E_1 to M_1. This is one of the Exchange Element transfers involved in the execution of the FLF instruction.

13-3 E REGISTER

The following types of transfers into the E register can take place:

1) Data can be transferred from the M register into the E register.
2) Data can be transferred from the A, B, C and D register into the E register.
3) Data in the IOBM (In-Out Buffer Mixer) can be transferred into the E register.
4) Data from the P, Q and XA registers can be transferred into the E register.
5) Certain bits of miscellaneous registers can be transferred into the E register for temporary storage.

In addition to the above, the quarters of E can be independently cleared and complemented, and the data in the quarters can be permuted.

13-3.1 M TO E TRANSFERS. The register driver logic for these transfers is shown in Fig. 13-7. It falls into three general categories.

1) Transfers Under Permuted Activity Control. Just as store type instructions transferred data from E to M at $QK^{13\alpha}$, load type instructions transfer data from M to E at $QK^{13\alpha}$. Note that a TSD or an SKM instruction may also transfer data from M to E at $QK^{13\alpha}$.

The ADX instruction has some of the characteristics of a load type instruction and some of the characteristics of a store type instruction. For this reason, it is treated separately and not lumped with the store type instructions.

13-2.2 E TO M TRANSFERS. The register driver logic tabulated on Fig. 13-5 indicates the various conditions under which E to M transfers take place. The conditions are determined by: the OP decoder class levels, which indicate in what instruction or type of instruction the transfer occurs; the time levels, which determine when the transfers occur; and the levels reflecting configuration control, sign extension control, parity, alarm control, etc.

Certain IOCM (In-Out Control Mixer) level logic associated with the TSD instruction is found on Fig. 13-5. This logic is discussed in detail in Chapter 15. Fig. 13-4 summarizes the aspects of this logic that are important in the discussion that follows. Note that only the $IOCM^{IN}$ logic (which indicates a TSD is transferring data between the In-Out Element and the central computer) is involved. Data may be transferred in both the NORMAL and ASSEMBLY mode during a TSD. In the NORMAL mode, data from the In-Out Element is transferred from E to M under configuration control, while in the ASSEMBLY configuration control is not used. Instead the data is cycled (shifted) one place to the right if an $IOCM^{RIGHT}$ level is present, or to the left if, an $\overline{IOCM^{RIGHT}}$ level is present, during the E to M transfer.

13-2.2.1 $\xrightarrow{0} M_{4,3,2,1}$. This clear pulse occurs whenever the parity alarm inhibition is absent ($\overline{MPA}$) and any one of the following three conditions is satisfied:

1) The instruction is a TSD in the ASSEMBLY mode.
2) The instruction is an SKM and $PKIR_{CF_3}$ is a ONE. (Note that this condition is not sufficient to clear $M_{4.10}$.)
3) All instructions having an operand cycle will normally clear the M register at $QK^{09\alpha}$, except those using the V_{FF} Memory during the operand cycle. Thus, TSD and SKM may clear the M register twice during the QK cycle.

13-2.2.2 $E \xrightarrow[CYL]{1} M$ AND $E \xrightarrow[CYR]{1} M$. Conditions 1 and 2 above, which cleared M at $QK^{18\alpha}$, also cycle E into M at $QK^{19\alpha}$. The only difference in the clear and cycle logic is the parity alarm condition and the added control logic for determining whether the shift is to the right or left. (See Fig. 13-6.)

13-2.2.3 $E \xrightarrow{0,1} M$. There are three categories of conditions under which this transfer takes place:

1) Broadside Transfers. Certain types of instructions transfer the ZEROS and ONES of all the quarters of E into the corresponding quarters of M simultaneously. These instructions include FLF, FLG, COM, and instructions involving the V_{FF} Memory.

13-2.1.2 PARITY ALARM. Normally, if the parity alarm flip-flop (MPAL) is set, the content of the M register should not be destroyed. (The operator may nullify the effects of this alarm by means of the parity alarm suppress pushbutton ($MPAL_{SUP}$).) However, the condition of the check parity circuit must be ignored at certain times, e.g., just before memory strobe when M contains all ZEROS. For this reason, a parity alarm level (MPA) is generated which controls the M register driver logic. If MPA is generated, then no pulses are allowed to change the content of M.

This MPA circuit looks at MPAL, $MPAL_{SUP}$ and the parity alarm inhibitory logic involving MPS. The net effect is that $MPAL^{1} \cdot \overline{MPAL_{SUP}}$ is a necessary but not sufficient condition for MPA. Pulses generated between $QK^{01\alpha}$ and $QK^{11\alpha}$ are always allowed to change the content of M.

13-2.1.3 META BIT (4.10). This bit is read into the M register from the Memory Element with the other operand bits. It is rewritten into memory, just as it was read out, for all instructions except SKM. The SKM instruction may complement, set to ONE, or clear to ZERO $M_{4.10}$ before the contents of $M_{4.10}$ are rewritten into memory.

Fig. 13-3 shows the register driver logic for complementing, setting or clearing the meta bit under SKM control. The state of the $PKIR_{CF_1}$ and $PKIR_{CF_2}$ bits determines which modification of the meta bit will take place. Note that the meta bit cannot be modified unless the memory parity alarm level is absent, i.e., an $\overline{MPA}$ condition exists. During an SKM instruction, the quarters of E are complemented, cleared, and set by the same register driver logic (except for the parity alarm inhibition) that correspondingly modifies $M_{4.10}$.

The second term in the $\xrightarrow{0} M_{4.10}$ register driver logic indicates that the meta bit is cleared when the M register as a whole is cleared, except in those cases where the M register is cleared at $QK^{18\alpha}$. The $QK^{18\alpha}$ inhibition guarantees that the meta bit will be rewritten in memory just as it was read out unless an SKM instruction is being executed. During most instructions, the meta bit will be cleared at $QK^{09\alpha}$ by the following logic:

$$\overline{QKM_{VFF}} \cdot \overline{MPA} \cdot QK^{09\alpha} \supset \xrightarrow{0} M_{4.10}$$

It should be realized, however, that $M_{4.10}$ is cleared by special circuitry and not in the direct manner indicated by the above equation.

The operand meta bit can be transferred between the Memory Element and the M register only.

2) The following instructions may temporarily store data in the E register during a PK cycle for reuse at a later time in the instruction:

IOS
JMP
JPX and JNX

Data may also be stored temporarily during deferred addressing (PK) and during a change of sequence (CSK).

3) During the execution of the SPF, SPG, FLF and FLG (F Memory) instructions, the FK counter initiates several data transfers in the Exchange Element.

13-2 M REGISTER (OPERAND MEMORY BUFFER)

Data is transferred into the M register from either the E register or the Memory Element. There are no other transfer paths into the M register.

13-2.1 OPERAND MEMORY STROBE. Fig. 13-1 shows the logic involved in strobing a word out of the Memory Element. The strobe logic for M and N are similar and is covered in greater detail in Chapter 11. The data may be transferred from a given memory sense amplifier into either the M or N register. If an instruction word is involved, it will be placed in the N register during a PK cycle. In the case of an operand, the word is placed in the M register during a QK cycle.

The operand strobe pulse logic is shown in Fig. 13-2. This logic consists of an operand memory selection level and a QK time level. The operand is strobed at $QK^{11\beta}$. In the case of the S Memory, the strobe pulse is routed through a "ripple" delay line. Thus, although the pulse is initiated at $QK^{10\beta}$, it does not finish strobing until $QK^{11\beta}$.

13-2.1.1 PARITY BIT (2.10). The parity bit is read out of memory into the M register along with the other operand bits. However, it is not written into memory with the other M register bits.

Before the content of the M register is written into memory, the parity of the word in the M register is computed. The output of the compute parity circuit is written into memory in place of the $M_{2.10}$ bit. Once the M register is cleared, the original parity bit is permanently lost.

During a normal load type instruction, the output of the check parity circuit will be equal to the $M_{2.10}$ bit if there is no read error. However, if a bit of the word is lost during the memory strobe, a parity alarm flip-flop (MPAL) will be set, since the check parity will not equal the value of the $M_{2.10}$ bit in this case.

CHAPTER 13

EXCHANGE ELEMENT

13-1 INTRODUCTION

During the execution of an instruction a complex series of data transfers may take place in the Exchange Element. Both the transfers themselves and the order in which they occur is important in determining the net effect of the transfers. This chapter will emphasize the register driver logic for the individual transfers and mention only in passing the time ordering of a sequence of transfers. A detailed dynamic picture of the transfers is developed in Chapter 16.

Manipulation of data in the Exchange Element involves the following register driver pulses:

$\lfloor 0 \rightarrow$, $\lfloor 1 \rightarrow$ or $\lfloor C \rightarrow$ pulses to $M_{4,3,2,1}$ or $E_{4,3,2,1}$

$M_i \xrightarrow{0} E_i$; $M_i \xrightarrow{1} E_i$; $M_i \xrightarrow{j} E_i$

$E_i \xrightarrow{0} M_i$; $E_i \xrightarrow{1} M_i$; $E_i \xrightarrow{j} M_i$

} $(i = 1,2,3,4)$

$E_{i+p} \longrightarrow E_i$

$E_i \longrightarrow E_{i+p}$

} $(p = 1,2,3;$ where $3 + 1 = 0$, $3 + 2 = 1$, etc.)

$E \xrightarrow[CYR]{1} M$ (CYR = cycle right)

$E \xrightarrow[CYL]{1} M$ (CYL = cycle left)

$\lfloor 0 \rightarrow$, $\lfloor 1 \rightarrow$, $\lfloor C \rightarrow$ pulses to $M_{4.10}$ ($M_{4.10}$ = meta bit)

In addition to the above, there are register driver pulses which transfer the contents of registers in other elements into the E and M registers.

Normally, data transfers occur in the Exchange Element during an operand cycle. This means that the transfer pulses are usually initiated by a QK time level. The following are exceptions to this rule:

1) If an instruction word is read out of the V_{FF} Memory, information will be transferred through the Exchange Element during the instruction cycle. Hence the transfer pulses will be initiated by PK time levels.

CHAPTER 13

EXCHANGE ELEMENT

TABLE OF CONTENTS

13-1 INTRODUCTION

13-2 M REGISTER (OPERAND MEMORY BUFFER)

13-2.1 OPERAND MEMORY STROBE

13-2.1.1 PARITY BIT (2.10)

13-2.1.2 PARITY ALARM

13-2.1.3 META BIT (4.10)

13-2.2 E TO M TRANSFERS

13-2.2.1 $\xrightarrow{0}$ $M_{4,3,2,1}$

13-2.2.2 E $\xrightarrow[CYL]{1}$ M AND E $\xrightarrow[CYR]{1}$ M

13-2.2.3 E $\xrightarrow{0,1}$ M

13-3 E REGISTER

13-3.1 M TO E TRANSFERS

13-3.2 ARITHMETIC ELEMENT TO E TRANSFERS

13-3.3 IOBM TO E TRANSFERS

13-3.4 MISCELLANEOUS USES OF E REGISTER

13-3.5 INTERCHANGE OF E REGISTER QUARTERS

13-3.6 CLEAR AND COMPLEMENT E REGISTER

13-4 ILLUSTRATIVE EXAMPLES

13-4.1 LOAD TYPE INSTRUCTION

13-4.2 STORE TYPE INSTRUCTION

13-4.3 SIGN EXTENSION

LIST OF FIGURES

13-1 MEMORY STROBE INTO M REGISTER

13-2 MEMORY STROBE INTO M REGISTER REGISTER DRIVER LOGIC

13-3 REGISTER DRIVER LOGIC OF M AND E REGISTERS FOR SKM INSTRUCTION

13-4 TRANSFER LOGIC FROM E TO M DURING TSD

13-5 E ⟶ M, M REGISTER RD CONTROL

13-6 CYCLIC REGISTER TRANSFER BETWEEN REGISTERS E AND M

13-7 M ⟶ E, E REGISTER RD CONTROL

13-8 ARITHMETIC ELEMENT REGISTERS TO E RD CONTROL

13-9 MISCELLANEOUS E REGISTER RD CONTROL

13-10 E REGISTER PERMUTATION RD CONTROL

13-11 $\xrightarrow{C}$ E, $\xrightarrow{0}$ E E REGISTER RD CONTROL

13-12 CONFIGURED LOAD TYPE INSTRUCTION ILLUSTRATING BASIC PERMUTATION AND M ⟶ E TRANSFERS

13-13 CONFIGURED STORE TYPE INSTRUCTION ILLUSTRATING BASIC PERMUTATION AND M ⟶ E TRANSFERS

13-14 SIGN EXTENSION IN E REGISTER

13-15 CONFIGURED LOAD TYPE INSTRUCTION ILLUSTRATING SIGN EXTENSION

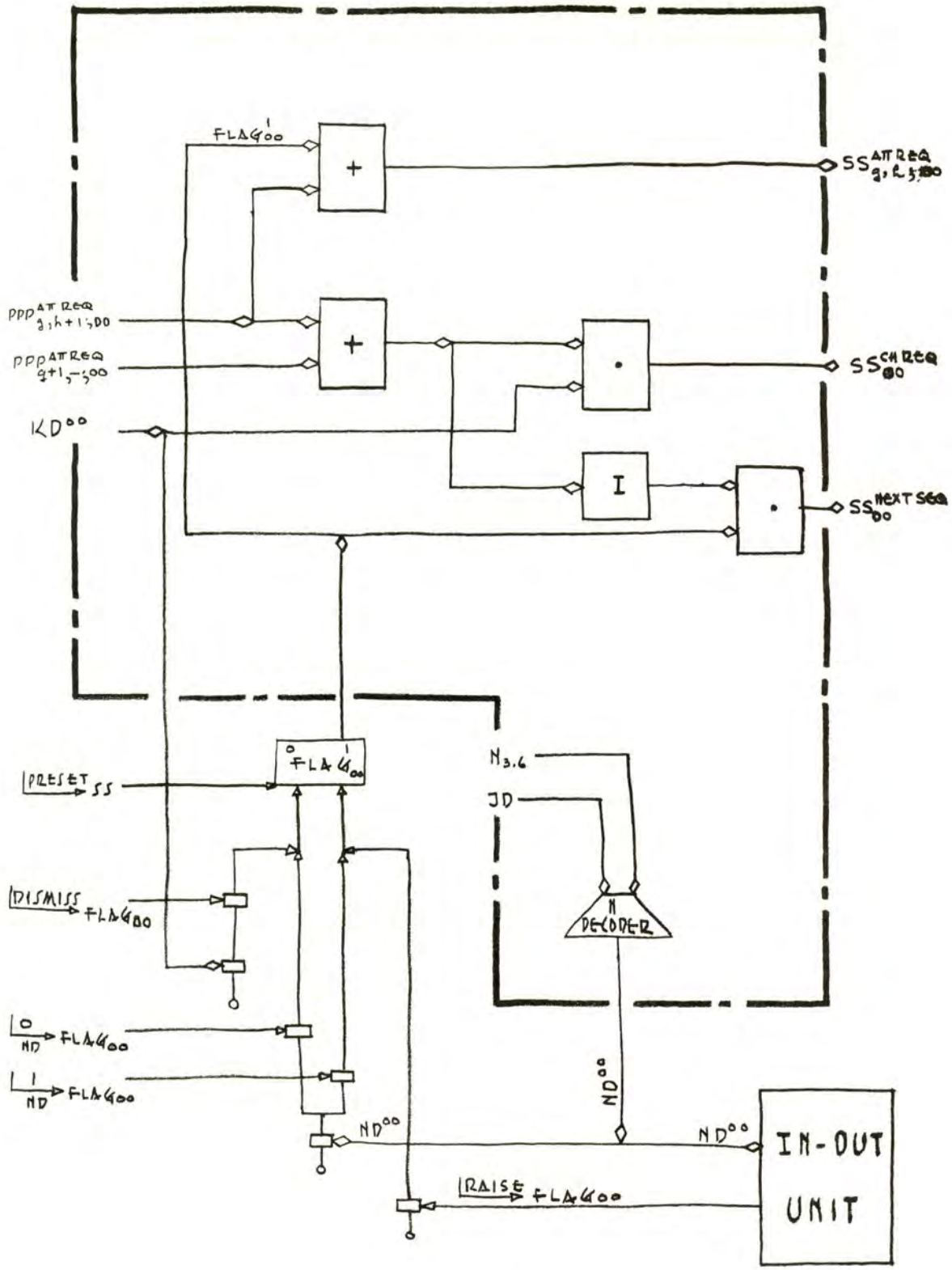
FLAG REGISTER LOGIC

PULSE	REGISTER DRIVER LOGIC					PULSE GATE LOGIC		
		RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
	COMPLETION PULSE FROM IO UNIT		RAISE → FLAG#					
I → FLAG	GATED BY ND	α	$PK^{26\alpha}$ · $PKIR^{105}$		· $N^{101}_{2,6-2,4}$			· $ND^{\#}$
			$PK^{26\alpha}$ · $PKIR^{SKX}$		· $PKIR_{cf'_4}$			
	GATED BY ND	α	$PK^{26\alpha}$ · $PKIR^{105}$		· $N^{100}_{2,6-2,4}$			· $ND^{\#}$
IO → FLAG	GATED BY KD	α	$CSK^{05\alpha}$		· KD^{00}			· $KD^{\#}$
			$PK^{25\alpha}$ · $PKIR^{DISREG}$		· $\overline{KD^{00}}$ · $PKIR^{0}_{h}$			
			$PK^{22\alpha}$ · $PKIR^{TSD}$ · $QKIR^{TSD}$		· PI^{0}_{2} · QB'			
			$PK^{22\alpha}$ · $PKIR^{TSD}$		· PI^{0}_{2} · $IOCM^{BB}$			
	CLEARS ALL FLAGS	α	PRESET → SS					

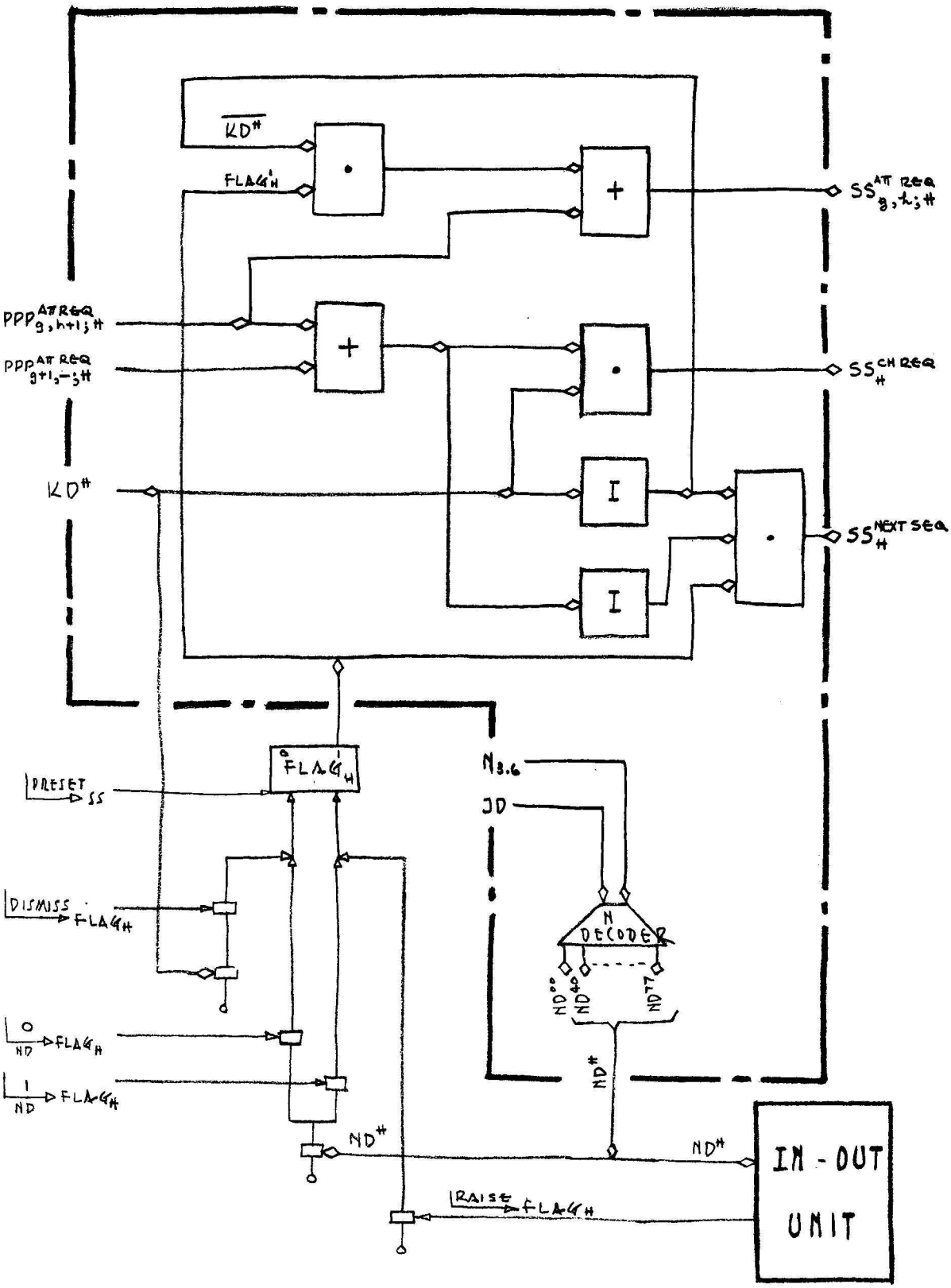
FLAG REGISTER LOGIC

FIG 12-51

HM 7-6-60

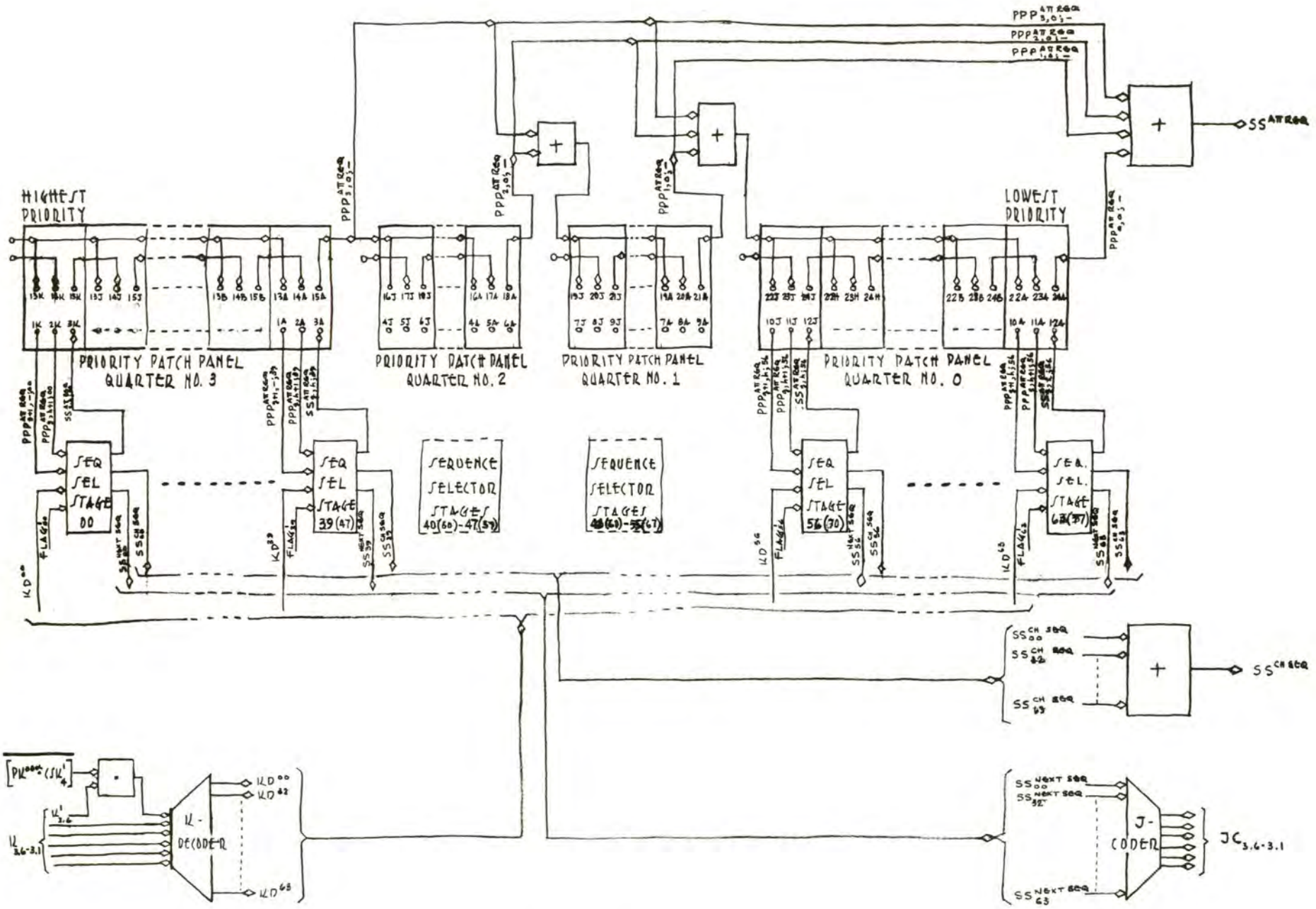


SEQUENCE SELECTOR STAGE (N = 0)
& FLAG LOGIC FIG 12-50 MM 7-13-60



SEQUENCE SELECTOR STAGE ($\#\neq 0$)
& FLAG LOGIC

FIG 12-49 HM 7-13-60



SEQUENCE SELECTOR

(FLAG LOGIC OMITTED)

FIG 12-48 HM 7-13-60

$$\underset{SE}{\lfloor^{O}} \circ E \cdot \overline{QKIR^{ACT_i}} \cdot QKIR^{EXT\,ACT_i} \supset \underset{SE}{\lfloor^{O}} \rightarrow E_i$$

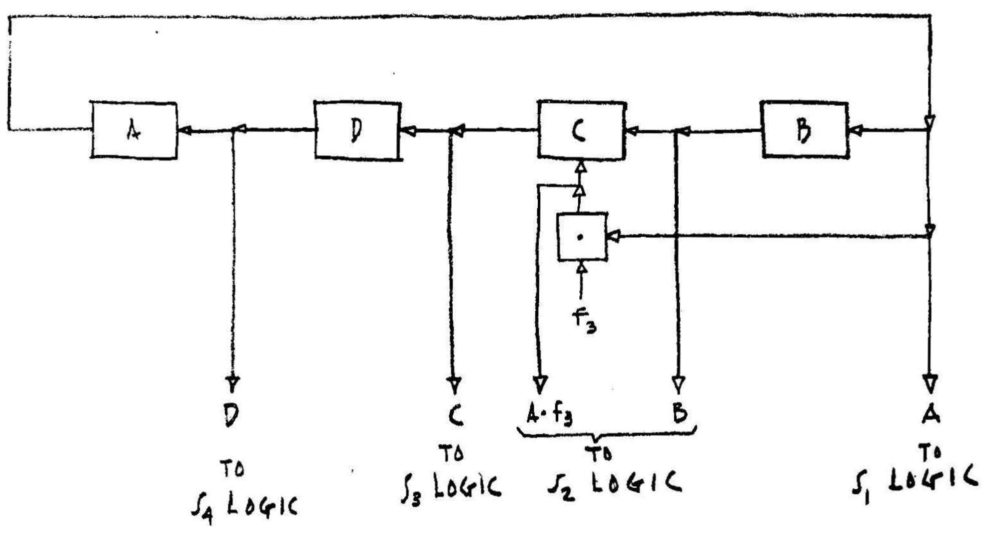
$$\underset{SE}{\lfloor^{C}} \circ E \cdot \overline{QKIR^{ACT_i}} \cdot QKIR^{EXT\,ACT_i} \cdot S_i \supset \underset{SE}{\lfloor^{C}} \rightarrow E_i$$

WHERE S_i :

$$S_1 = A \cdot QKIR^{f_1} + E'_{2,9} \cdot QKIR^{f_2}$$
$$S_2 = A \cdot QKIR^{f_3} + E'_{1,9} \cdot QKIR^{f_2} + B$$
$$S_3 = C \cdot QKIR^{f_1+f_3} + E'_{4,9} \cdot QKIR^{f_2}$$
$$S_4 = D \cdot QKIR^{f_1+f_3} + E'_{3,9} \cdot QKIR^{f_2}$$

WHERE

$$A = D \cdot \overline{QKIR^{ACT4}} + E'_{4,9} \cdot QKIR^{ACT4}$$
$$B = \left(A + QKIR^{ACT_1}\right) \cdot \left(E'_{1,9} + \overline{QKIR^{ACT_1}}\right) \cdot QKIR^{f_1}$$
$$C = \left(B + A \cdot QKIR^{f_3}\right) \cdot \overline{QKIR^{ACT_2}} + E'_{2,9} \cdot QKIR^{ACT_2}$$
$$D = \left(C + QKIR^{ACT_3}\right) \cdot \left(E'_{3,9} + \overline{QKIR^{ACT_3}}\right)$$



SIGN EXTENSION LOGIC AND NETS

FIG 12-47

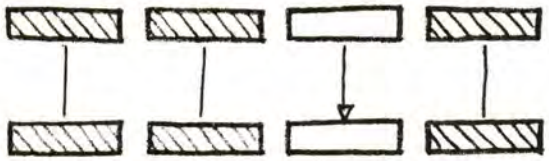
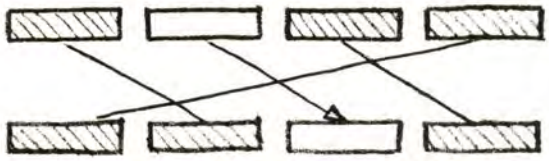
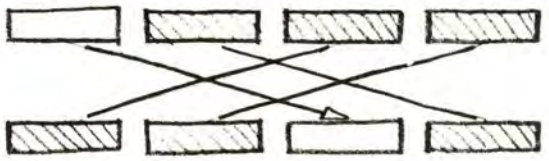
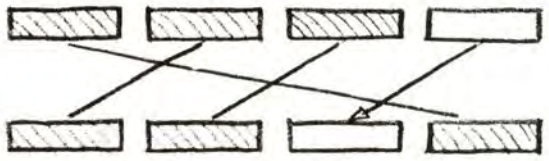
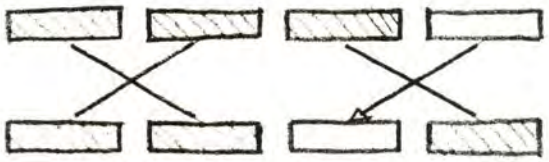
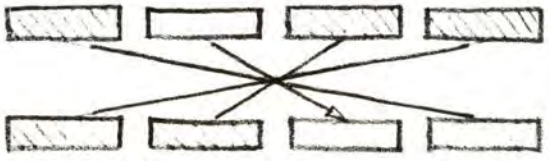
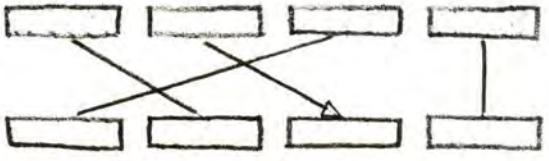
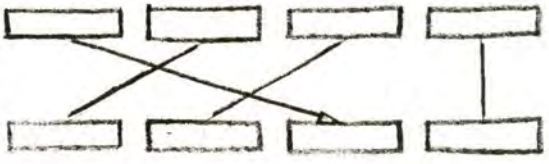
HM 12-21-60

LEVEL	=	ACTIVE QUARTER	PERMUTATION
$QKIR^{PRM\ ACT_1}$	=	$QKIR^{ACT_1}$	$\cdot QKIR^{PRM\ 0+1+7}$
		$QKIR^{ACT_2}$	$\cdot QKIR^{PRM\ 3+4}$
		$QKIR^{ACT_3}$	$\cdot QKIR^{PRM\ 2}$
		$QKIR^{ACT_4}$	$\cdot QKIR^{PRM\ 1+5}$
$QKIR^{PRM\ ACT_2}$	=	$QKIR^{ACT_1}$	$\cdot QKIR^{PRM\ 1+4}$
		$QKIR^{ACT_2}$	$\cdot QKIR^{PRM\ 0}$
		$QKIR^{ACT_3}$	$\cdot QKIR^{PRM\ 3+5+7}$
		$QKIR^{ACT_4}$	$\cdot QKIR^{PRM\ 2+6}$
$QKIR^{PRM\ ACT_3}$	=	$QKIR^{ACT_1}$	$\cdot QKIR^{PRM\ 2}$
		$QKIR^{ACT_2}$	$\cdot QKIR^{PRM\ 1+5+6}$
		$QKIR^{ACT_3}$	$\cdot QKIR^{PRM\ 0}$
		$QKIR^{ACT_4}$	$\cdot QKIR^{PRM\ 3+4+7}$
$QKIR^{PRM\ ACT_4}$	=	$QKIR^{ACT_1}$	$\cdot QKIR^{PRM\ 3+5}$
		$QKIR^{ACT_2}$	$\cdot QKIR^{PRM\ 2+7}$
		$QKIR^{ACT_3}$	$\cdot QKIR^{PRM\ 1+4+6}$
		$QKIR^{ACT_4}$	$\cdot QKIR^{PRM\ 0}$

QKIR PERMUTED ACTIVITY LOGIC

Fig 12-46

HM 7-20-60

PERMUTATION	=	$QKIR_{CF}$ SETTINGS	PERMUTATION PATHS AND ACTIVITY REPRESENTATION
$QKIR^{PRM_0}$	=	$QKIR_{cf321}^{000}$	
$QKIR^{PRM_1}$	=	$QKIR_{cf321}^{001}$	
$QKIR^{PRM_2}$	=	$QKIR_{cf321}^{010}$	
$QKIR^{PRM_3}$	=	$QKIR_{cf321}^{011}$	
$QKIR^{PRM_4}$	=	$QKIR_{cf321}^{100}$	
$QKIR^{PRM_5}$	=	$QKIR_{cf321}^{101}$	
$QKIR^{PRM_6}$	=	$QKIR_{cf321}^{110}$	
$QKIR^{PRM_7}$	=	$QKIR_{cf321}^{111}$	

PERMUTATION AS DEFINED BY $QKIR_{CF}$.

ALSO ILLUSTRATES THE EFFECT OF PERMUTATIONS UPON THE RELATIONSHIP BETWEEN ACTIVITY IN THE CENTRAL MACHINE AND THE MEMORY ELEMENT

(SHOWN WITH SECOND QUARTER OF CENTRAL MACHINE ACTIVE)

FIG 12-45
HM 7-19-60

SUBWORD FORM	ACTIVITY	⊃	$QKIR^{EXT\,ACT_4}$	$QKIR^{EXT\,ACT_3}$	$QKIR^{EXT\,ACT_2}$	$QKIR^{EXT\,ACT_1}$
f_1	1	⊃	×	×	×	
	2		×	×		×
	3		×		×	×
	4			×	×	×
f_2	1	⊃			×	
	2					×
	3		×			
	4			×		
f_3	1	⊃				
	2		×	×		
	3		×		×	
	4			×	×	
f_4	1	⊃				
	2					
	3					
	4					

QKIR EXTENDED ACTIVITY

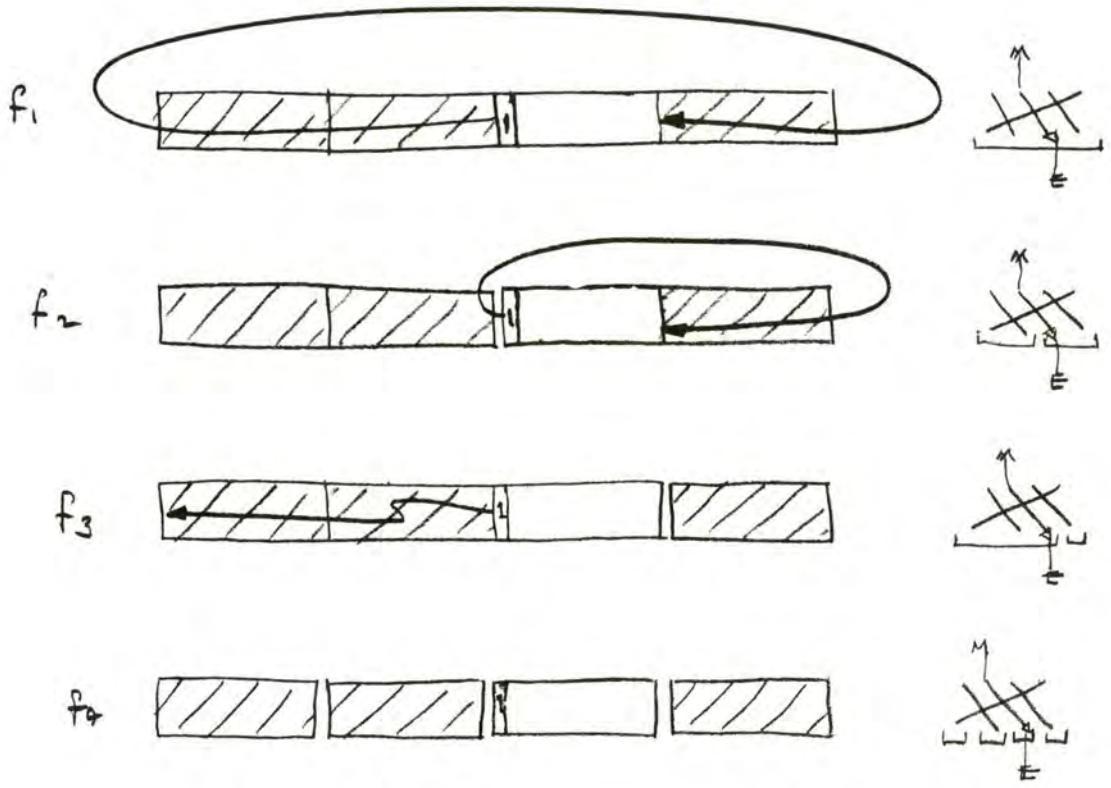
FIG 12-44 HM 7-20-60

LEVEL	=	ACTIVE QUARTER	SUBWORD FORM
$QKIR^{EXT\ ACT_1}$	=	$QKIR^{ACT_1}$	
		$QKIR^{ACT_2}$	$\cdot\ QKIR^{f_1+f_2}$
		$QKIR^{ACT_3}$	$\cdot\ QKIR^{f_1}$
		$QKIR^{ACT_4}$	$\cdot\ QKIR^{f_1}$
$QKIR^{EXT\ ACT_2}$	=	$QKIR^{ACT_1}$	$\cdot\ QKIR^{f_1+f_2}$
		$QKIR^{ACT_2}$	
		$QKIR^{ACT_3}$	$\cdot\ QKIR^{f_1+f_3}$
		$QKIR^{ACT_4}$	$\cdot\ QKIR^{f_1+f_3}$
$QKIR^{EXT\ ACT_3}$	=	$QKIR^{ACT_1}$	$\cdot\ QKIR^{f_1}$
		$QKIR^{ACT_2}$	$\cdot\ QKIR^{f_1+f_3}$
		$QKIR^{ACT_3}$	
		$QKIR^{ACT_4}$	$\cdot\ \overline{QKIR^{f_4}}$
$QKIR^{EXT\ ACT_4}$	=	$QKIR^{ACT_1}$	$\cdot\ QKIR^{f_1}$
		$QKIR^{ACT_2}$	$\cdot\ QKIR^{f_1+f_3}$
		$QKIR^{ACT_3}$	$\cdot\ \overline{QKIR^{f_4}}$
		$QKIR^{ACT_4}$	
$QKIR_{cf}^{EXT\ ACT_3\ +}$	=	$QKIR^{EXT\ ACT_2} \cdot QKIR^{EXT\ ACT_1}$	
$QKIR^{ALL\ ACT}$	=	$QKIR_{cf\ 7}^{ACT_4} \cdot QKIR_{cf\ 6}^{ACT_3} \cdot QKIR_{cf\ 5}^{ACT_2} \cdot QKIR_{cf\ 4}^{ACT_1}$	

QKIR EXTENDED ACTIVITY LOGIC

Fig 12-43

HM 7-20-60



a_2

f_1: $QKIR^{EXT\ ACT_4} \cdot QKIR^{EXT\ ACT_3} \cdot QKIR^{EXT\ ACT_1}$

f_2: $\cdot\ QKIR^{EXT\ ACT_1}$

f_3: $QKIR^{EXT\ ACT_4} \cdot QKIR^{EXT\ ACT_3}$

f_4: —

VARIABLES : ACTIVITY & ⇩ COUPLING ⌊

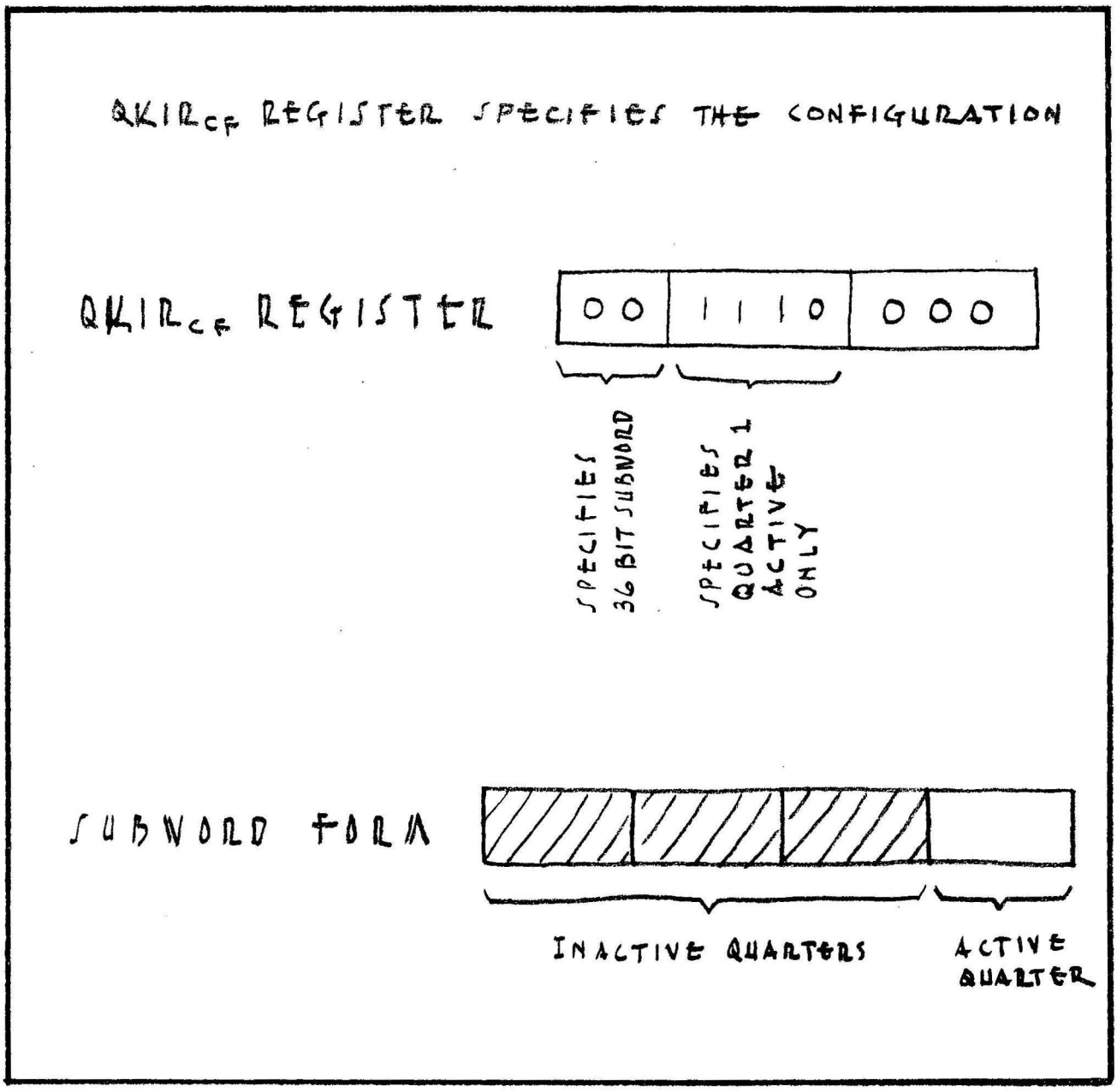
INDEPENDENT OF PERMUTATION

EXTENDED ACTIVITY

WITH SECOND QUARTER ACTIVE

FIG 12 - 42

HM 7-19-60

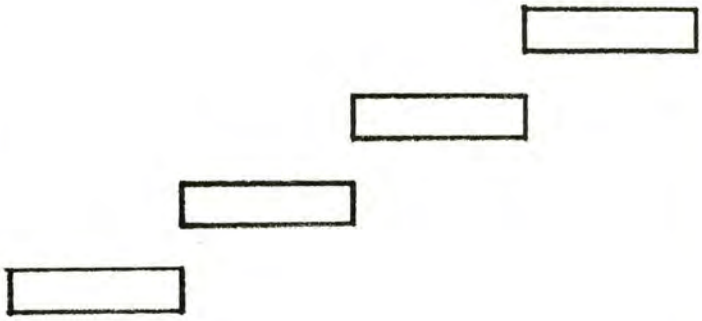
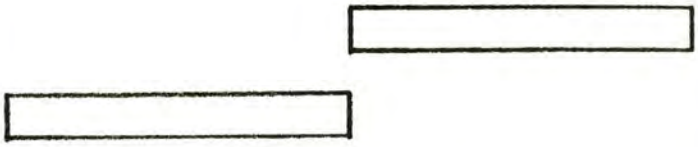


PARTIALLY ACTIVE SUBWORD

(36-BIT SUBWORD WITH ONLY QUARTER 1 ACTIVE)

FIG 12-41

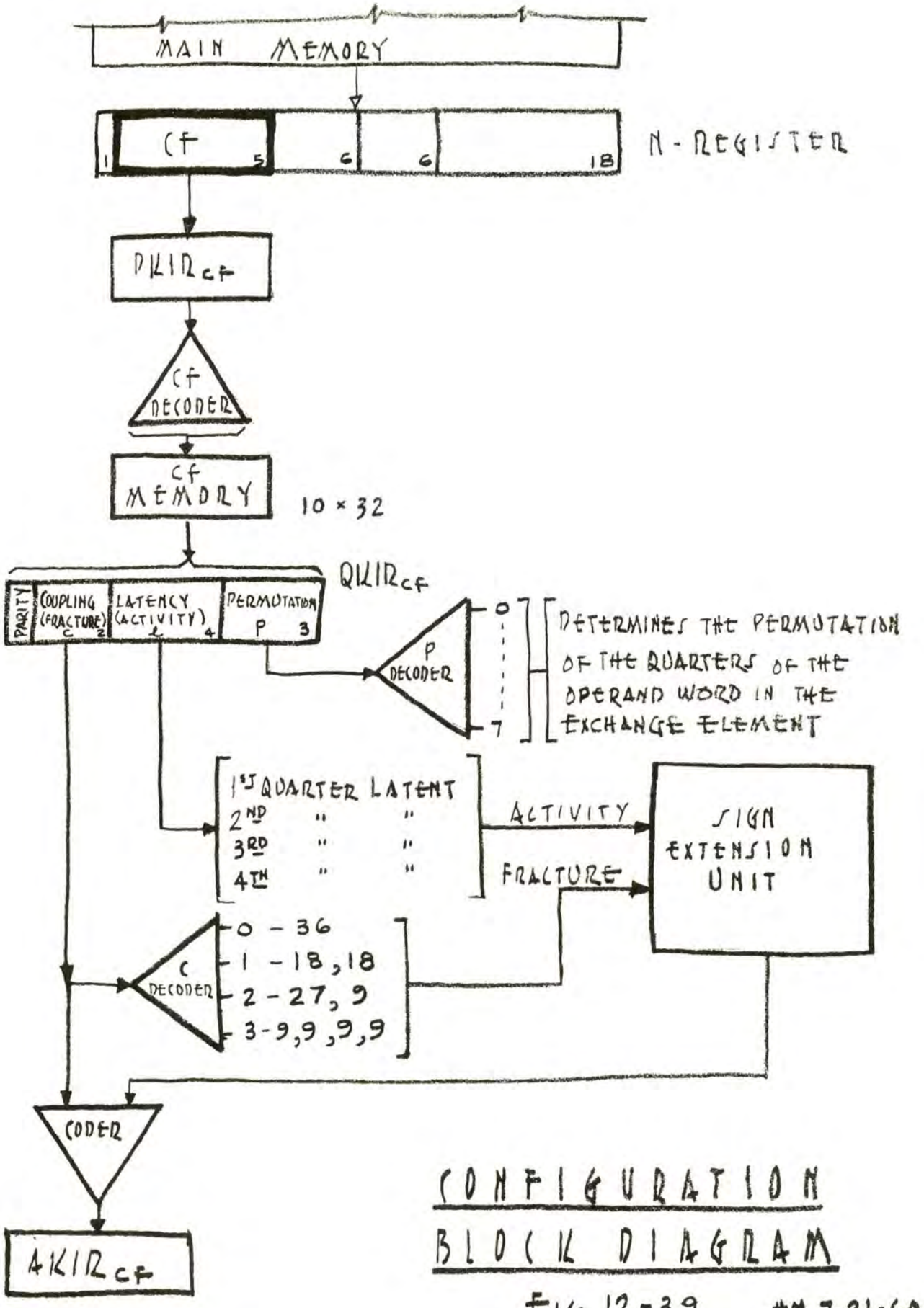
HM 12-21-60

FRACTURE NOTATION	NO OF BITS IN SUBWORDS	SUBWORD FORM
f_3	9, 9, 9, 9	
f_2	18, 18	
f_1	27, 9	
f_0	36	

SUBWORD FORM

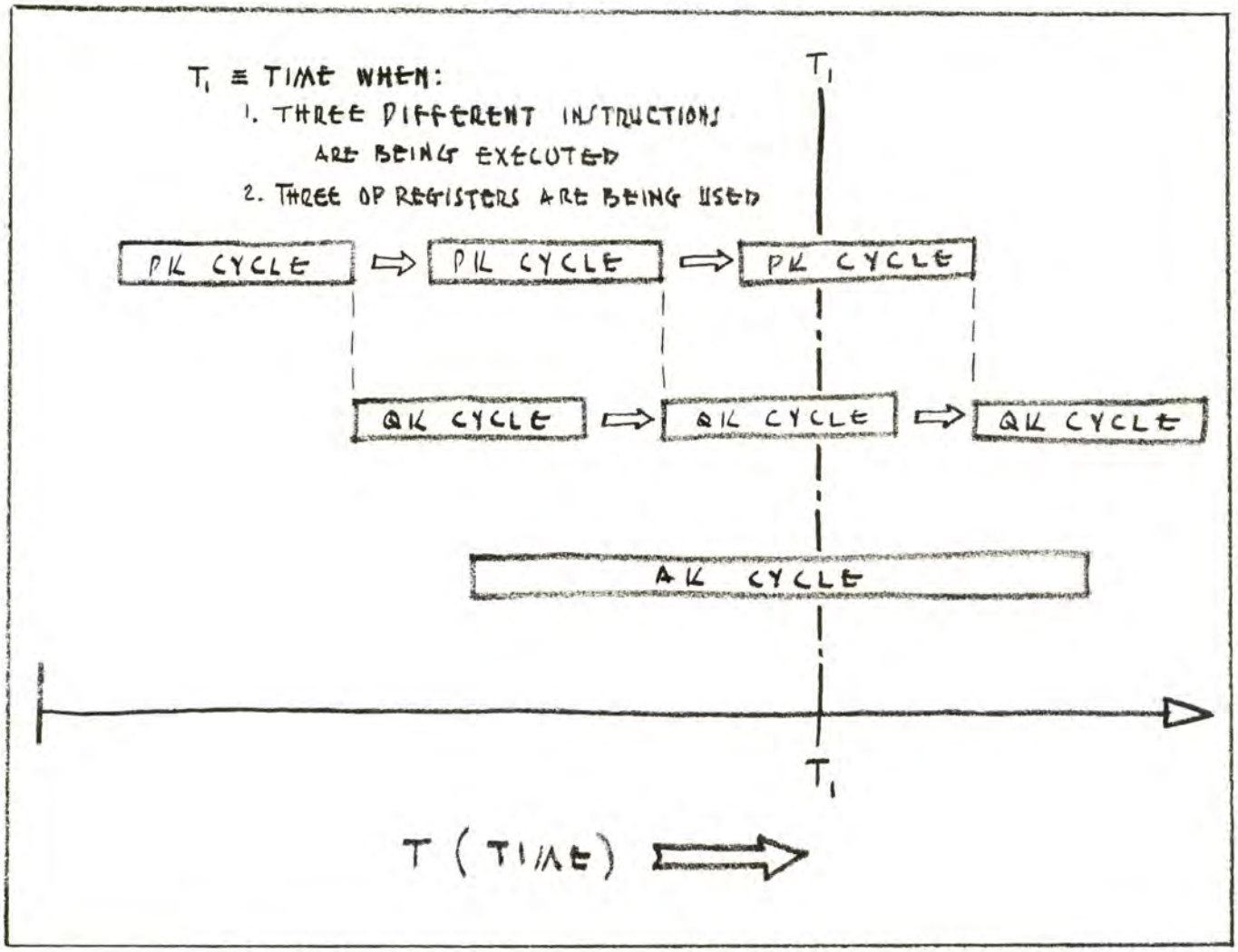
FIG 12-40

HM 12-20-60



CONFIGURATION BLOCK DIAGRAM

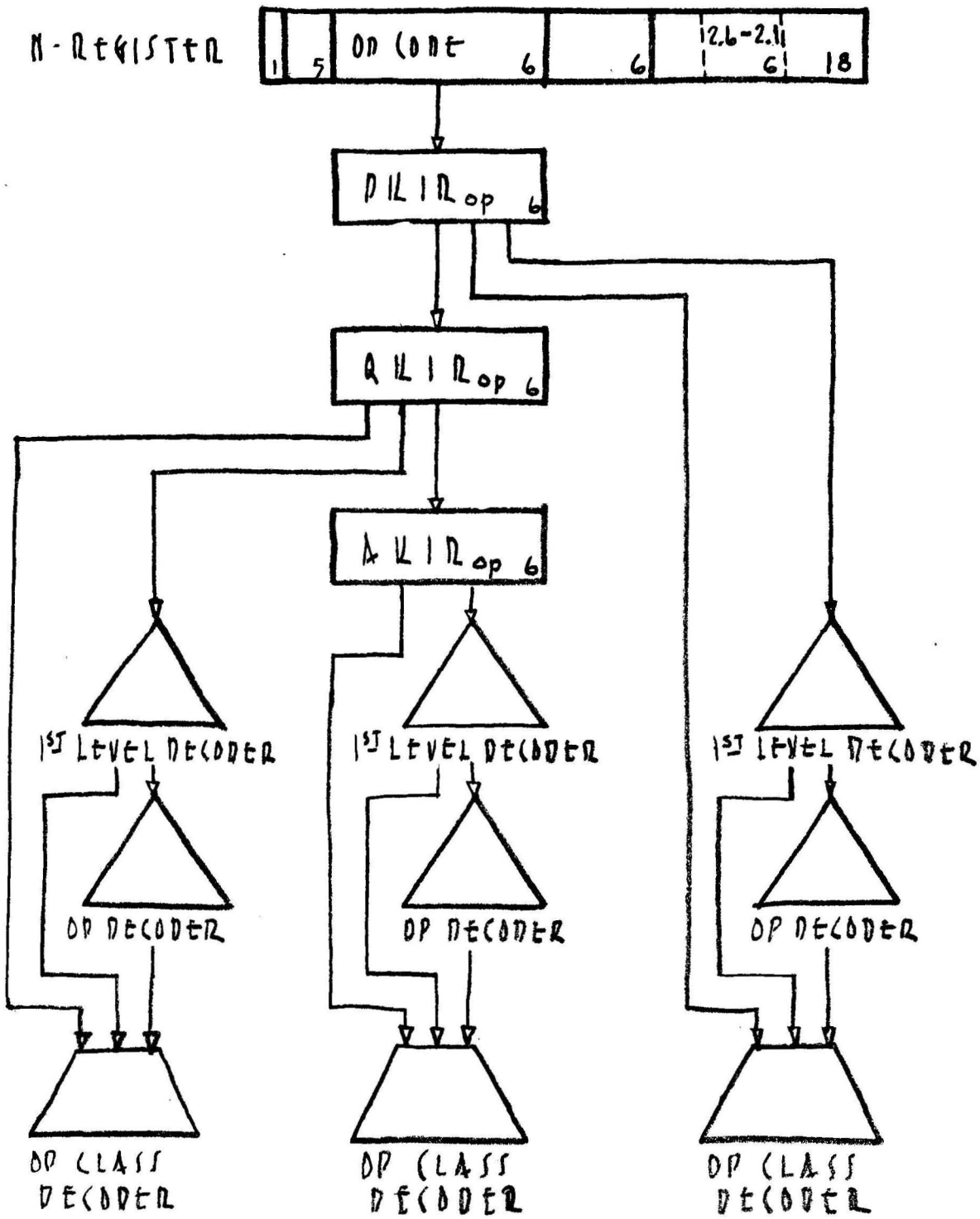
FIG 12-39 HM 7-21-60



SIMULTANEOUS EXECUTION OF INSTRUCTIONS

FIG 12-36

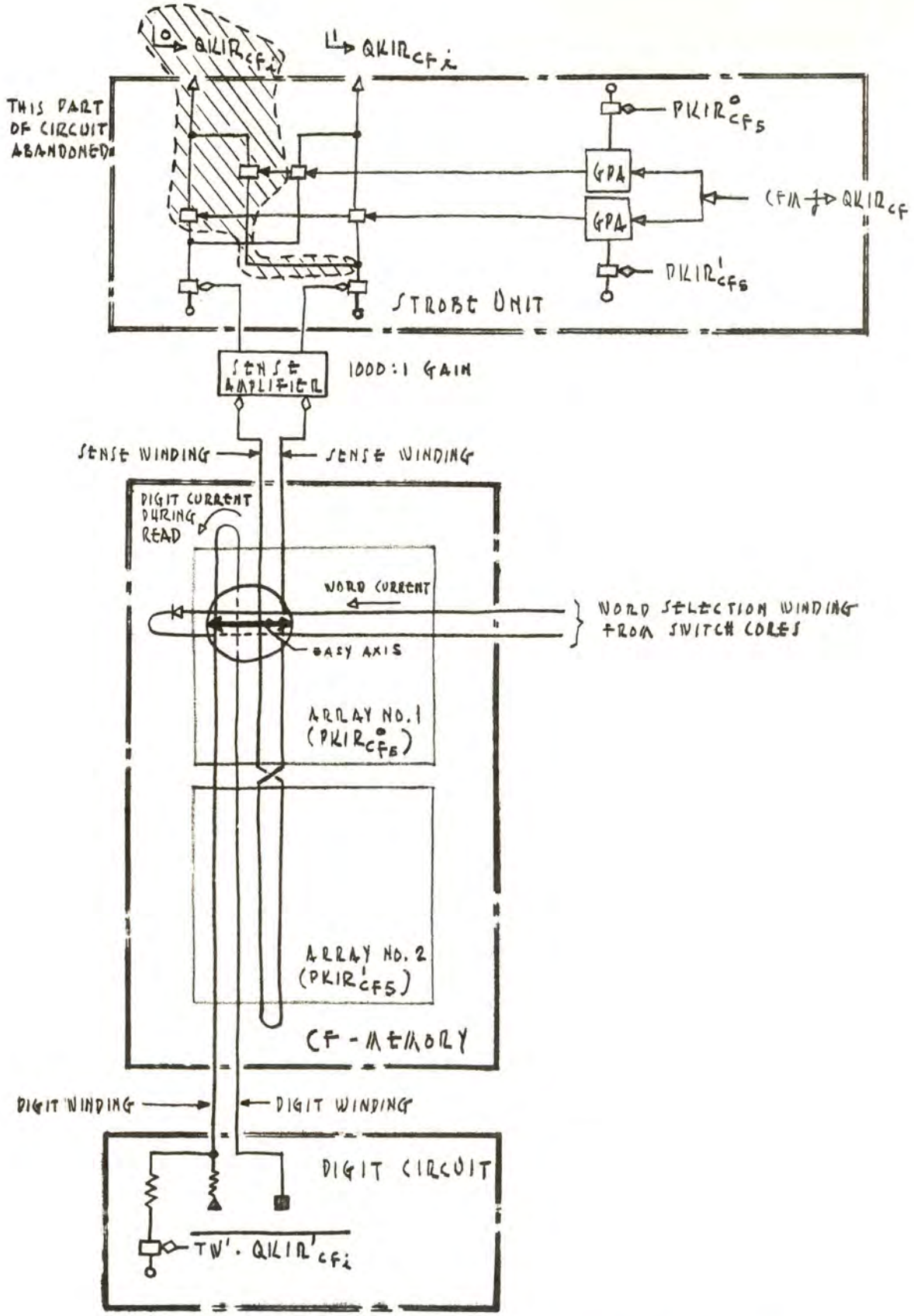
HM 12-20-60



OPERATION CODE BLOCK DIAGRAM

FIG 12-35

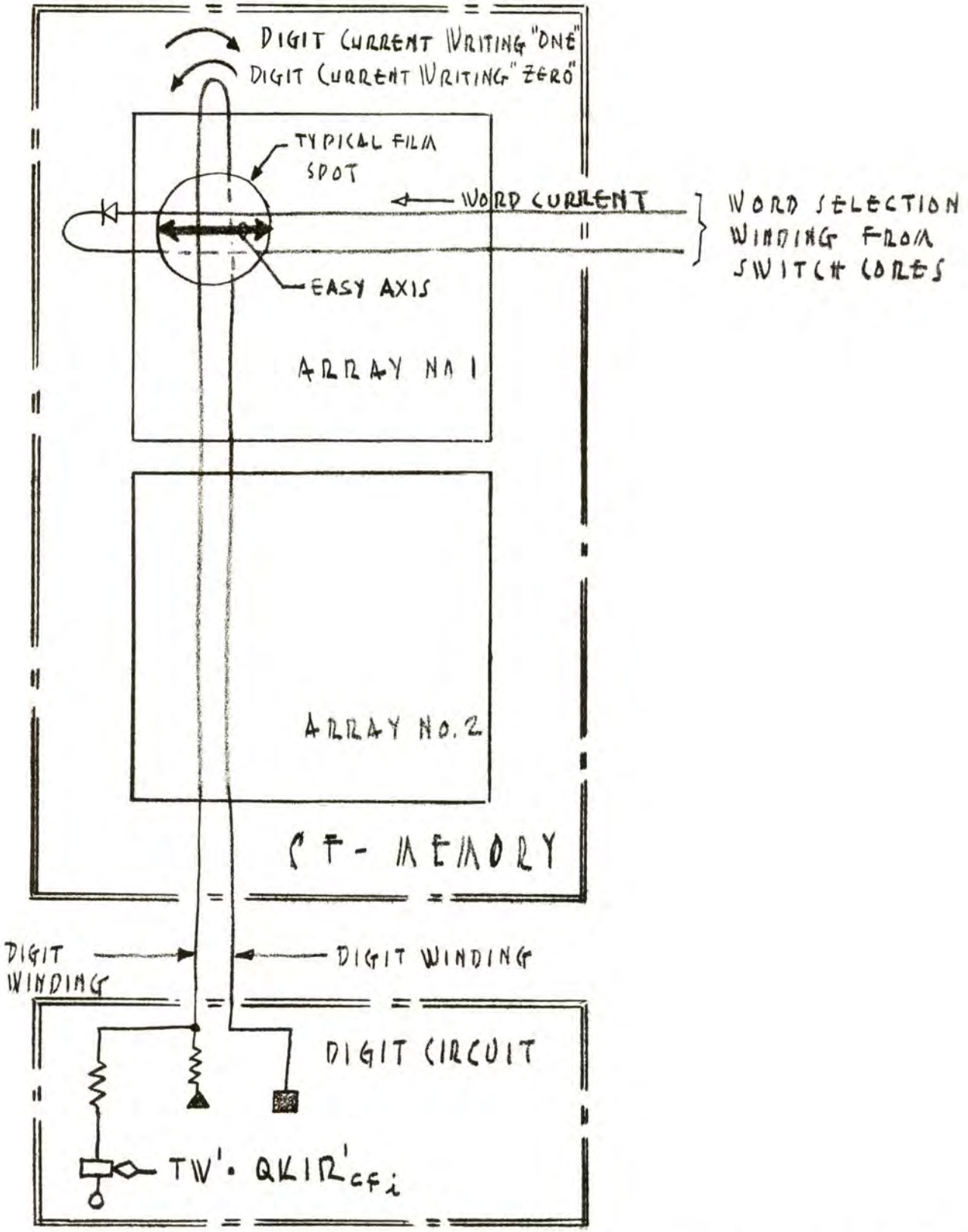
MA 8-3-60



TYPICAL CF-MEMORY READ CIRCUIT STAGE

FIG 12-34

HM 10-20-60



TYPICAL CT-MEMORY WRITE CIRCUIT STAGE

FIG 12-33
HM 10-20-60

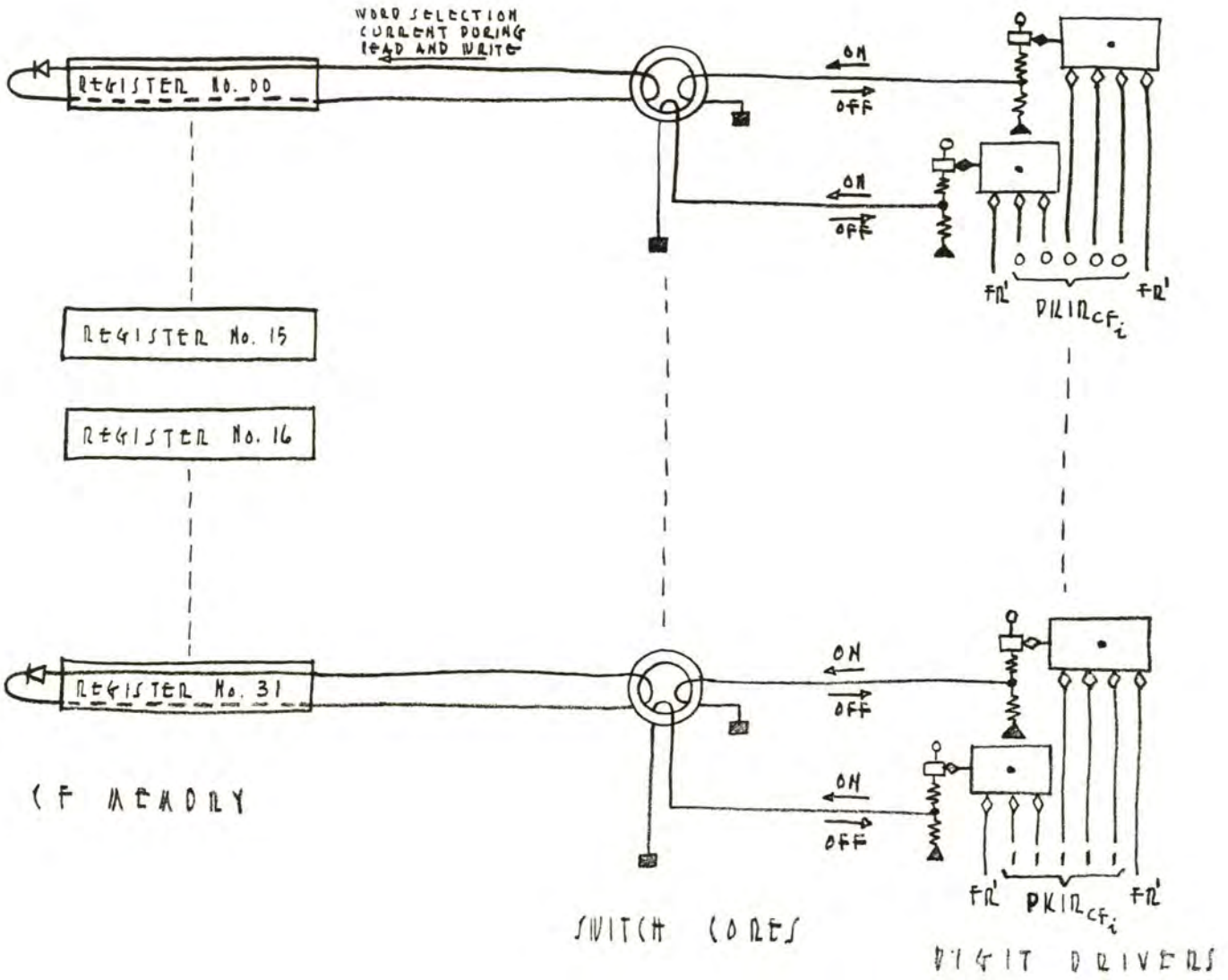
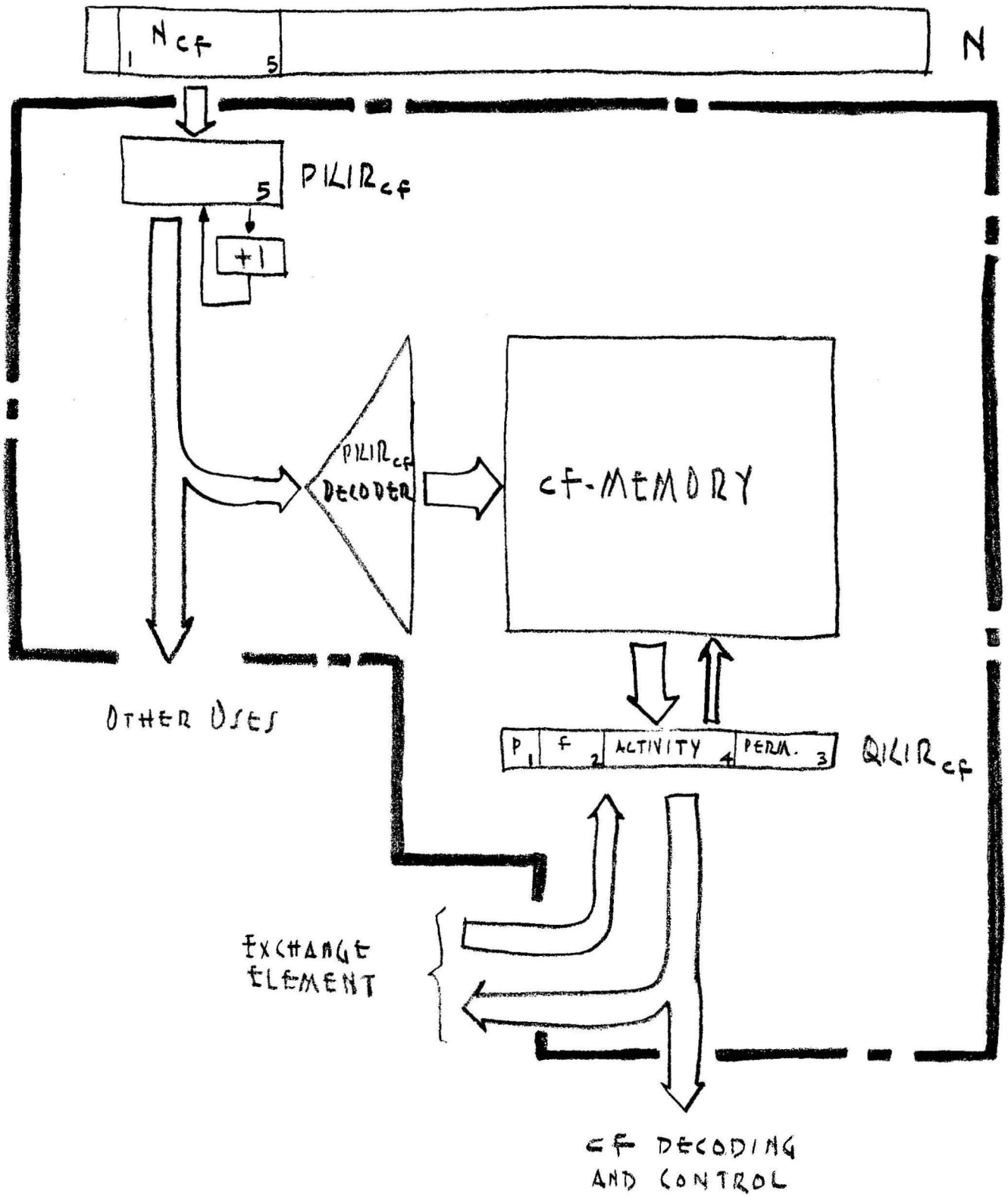


FIG 12-32

CF MEMORY REGISTER SELECTION

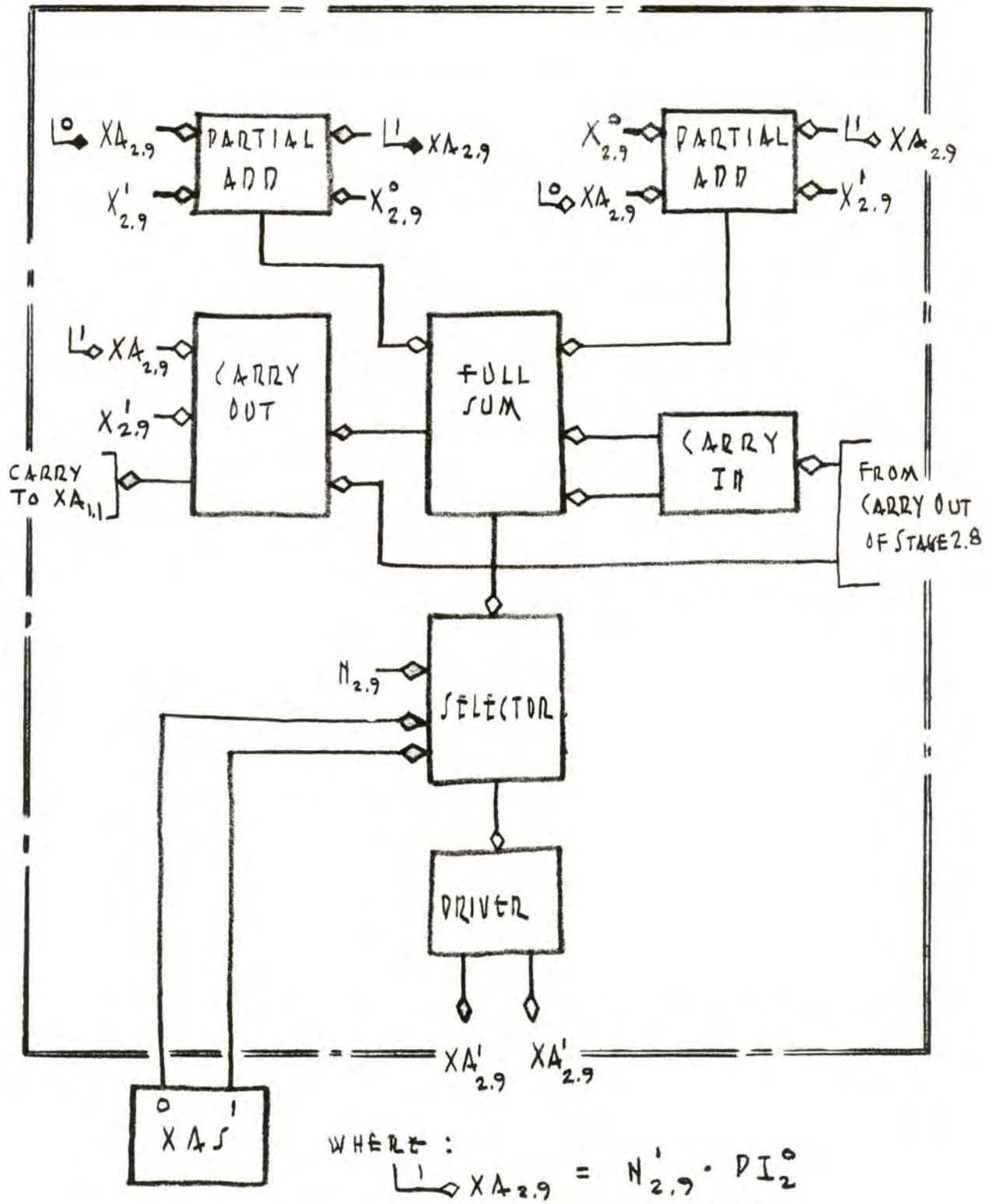
HA 10/19/60



CF-MEMORY SYSTEM

FIG 12-31

HM 10-21-60



WHERE :

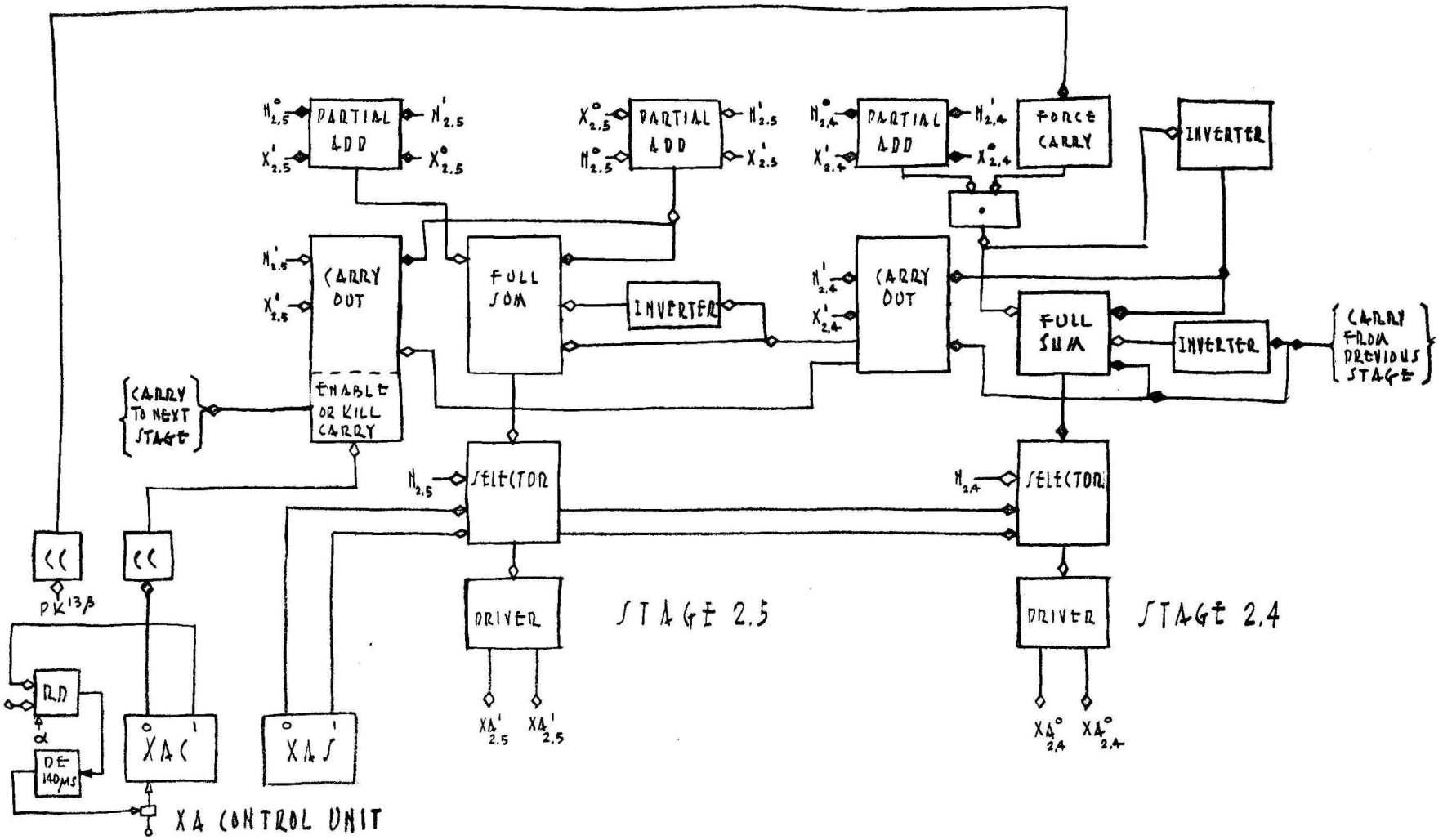
$^1_{\diamond}$ XA$_{2.9}$ = N'$_{2.9}$ · DI$^0_2$

$^0_{\diamond}$ XA$_{2.9}$ = $\overline{\ ^1_{\diamond}\text{XA}_{2.9}\ }$

DEFER BIT STAGE OF X-MEMORY ADDER

FIG 12-30

HM 7-27-60



TYPICAL X-MEMORY ADDER STAGES

(STAGES 2.4 AND 2.5 SHOWN)

FIG 12-29

HM 7-27-60

X-MEMORY LOGIC

PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHER	TIME LEVEL	INSTRUCTION	OTHER
L 1 → XW	α	PRESET → CE			XWK$^{02\alpha}$		
L 0 → XR	α	PRESET → CE			XWK$^{06\alpha}$		
		PRESET → CE					

X-MEMORY WRITE LOGIC

FIG 12-28
#M 12-20-60

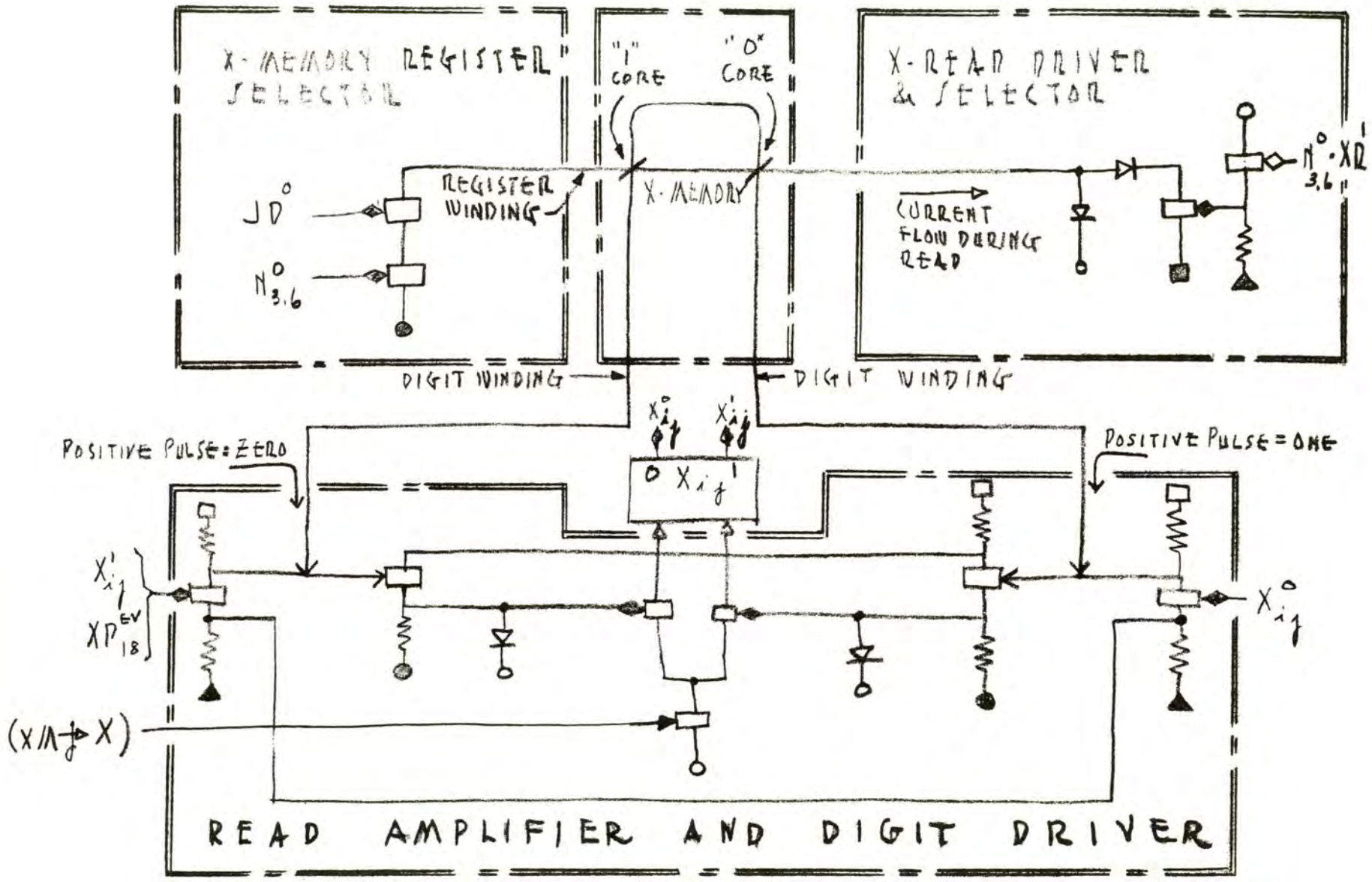
X-MEMORY LOGIC

PULSE	REGISTER DRIVER LOGIC				DELAY	PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHER	DELAY	TIME LEVEL	INSTRUCTION	OTHER
1 → XR	α	PRESET → CE				$PK^{12\alpha}$ $CSK^{01\alpha}$ $CSK^{04\alpha}$ $QK^{13\alpha} \cdot QKIR^{AUX}$ $QK^{13\alpha} \cdot QKIR^{LD} \cdot QKIR^{X}$		
10 → XR						PRESET → CE		
	α				0.06μs	XR'		

X-MEMORY READ LOGIC

FIG 12-27

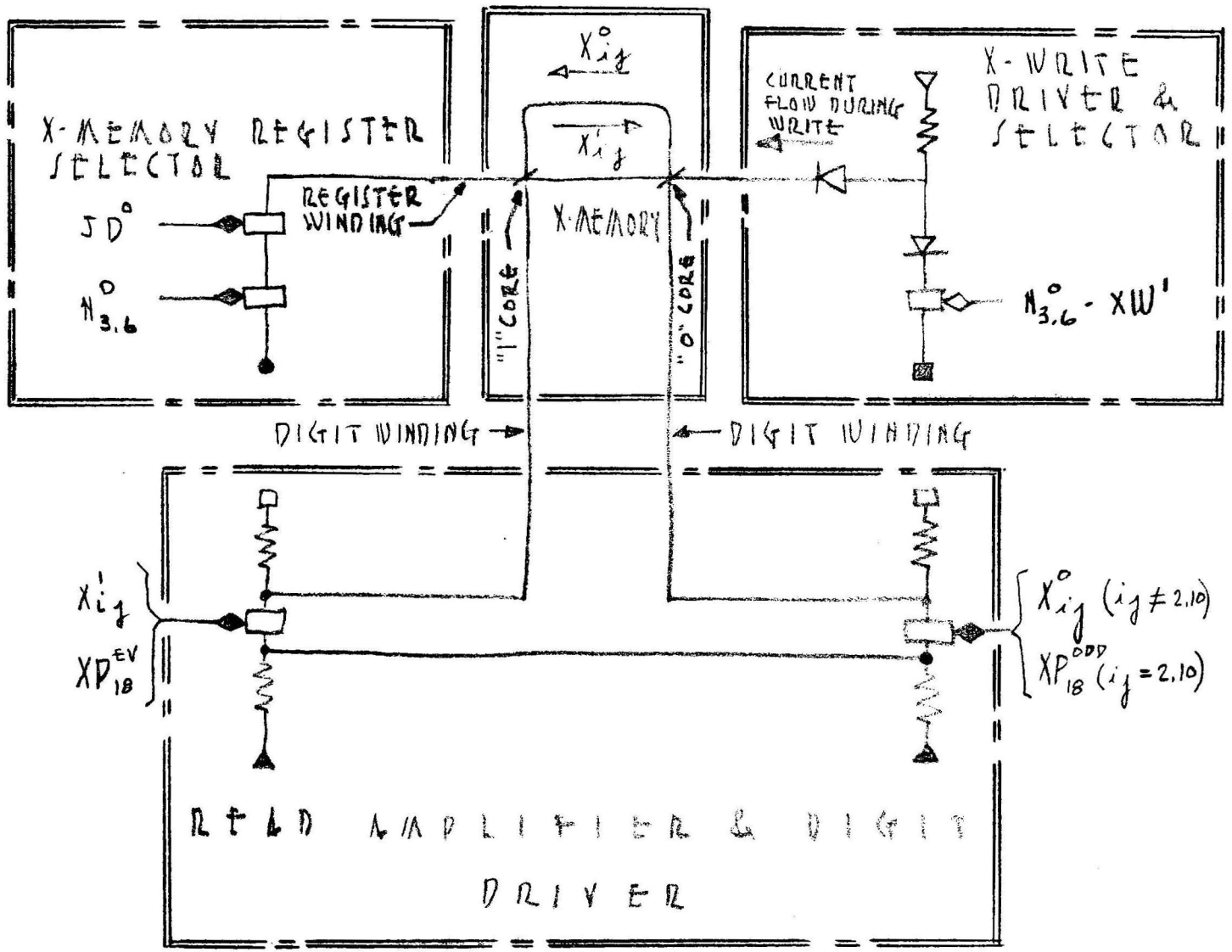
TM 12-20-60



TYPICAL X-MEMORY READ CIRCUIT STAGE
(i, j BIT OF REGISTER DO ILLUSTRATED)

FIG 12-26

TM 10-17-60

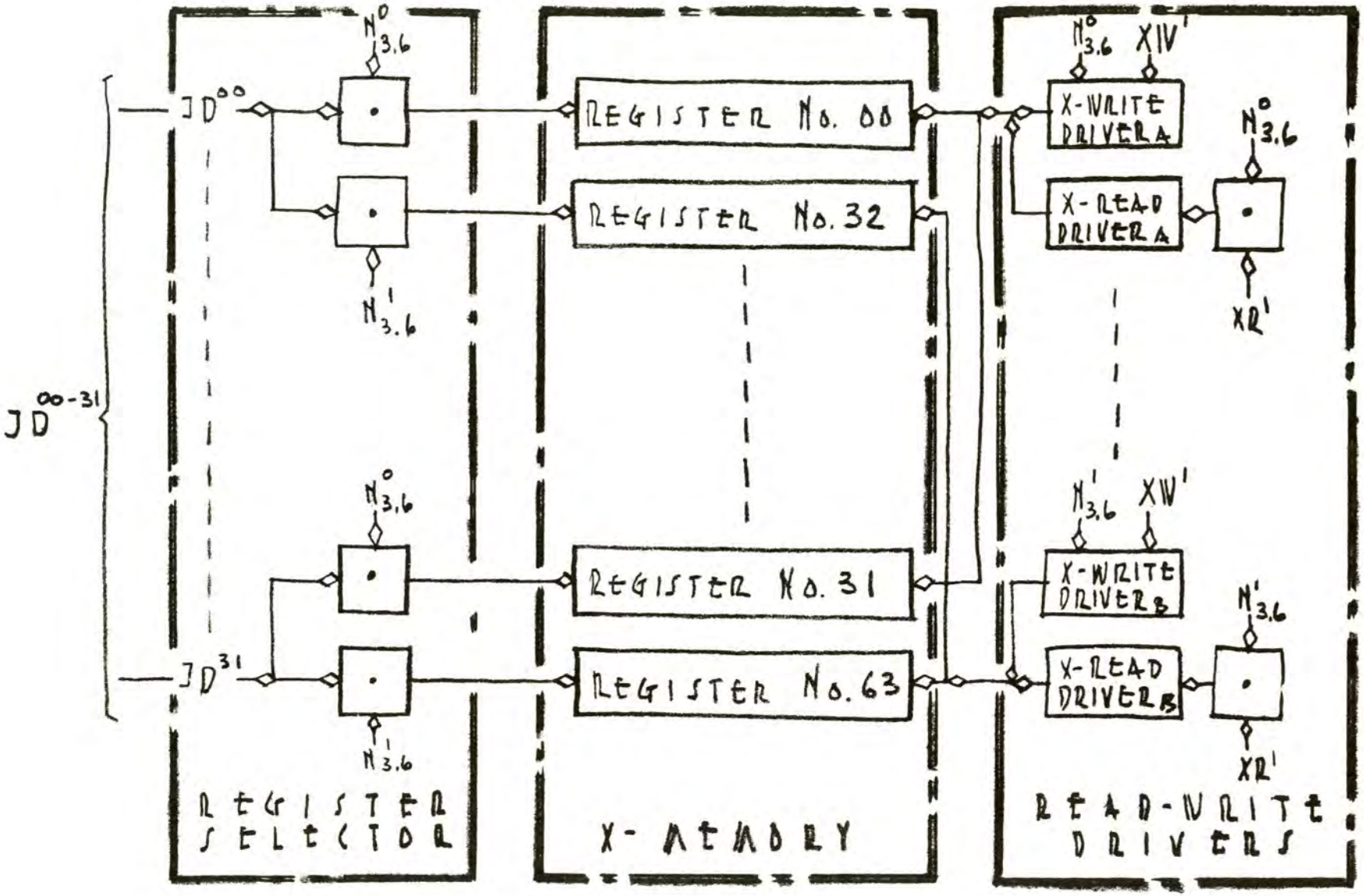


TYPICAL X-MEMORY WRITE CIRCUIT STAGE

(i,j BIT OF REGISTER 00 ILLUSTRATED)

FIG 12-25

HM 10-17-60



X-MEMORY REGISTER SELECTION

FIG 12-24

HA 10-18-60

AKIR$_{cf}$ REGISTER LOGIC							
PULSE	REGISTER DRIVER LOGIC				PULSE GATE LOGIC		
	RD PULSE	TIME LEVEL	INSTRUCTION	OTHERS	TIME LEVEL	INSTRUCTION	OTHERS
$N_{1.9-1.4} \xrightarrow{\downarrow} AKIR_{cf}$	α	$AK'_{\alpha,0}$		$PL^{25\alpha} \cdot PKIR^{OPR\ A\leftarrow}$			$\cdot N_{1.9-1.4}$
$QKIR_{cf_{9-8}} \xrightarrow{\downarrow} AKIR_{cf_{9-8}}$	α			$\cdot AK^{13\alpha} \cdot QKIR^{AIL}$			$\cdot QKIR^{EXT\ ACT_{4-1}}$

N_1-REGISTER AND $QKIR_{CF}$-REGISTER TRANSFER INTO $AKIR_{CF}$ REGISTER

FIG 12-23

MM L-28-60

	Clear And Cycle into M Transfers									
Pulse	Time Level	Inst	Misc.	Time Level	Inst	Misc.	Time Level	Inst	Parity Alarm	Cycle Direction (Left or Right)
(See Note 1) $\underline{Lo} \rightarrow M_{4,3,2,1}$	$[QK^{18d}\cdot(QKIR^{TSD}\cdot IOCM^{Ass'r})+$			$QK^{18d}\cdot(QKIR^{SKM}\cdot PKIR^{1}_{CF_3})+$			$QK^{09d}\cdot\overline{QKM^{VFF}}]\cdot$		$[\overline{MPA}]$	
$E \xrightarrow[CYC]{1} M$	$[QK^{19d}\cdot($← Ditto →$)+$			$QK^{19d}\cdot($← Ditto →$)$			$]\cdot$		$[MPAL^{0}+MPAL_{SUP}]\cdot$	$[\overline{QKIR^{SKM}+IOCM^{Right}}]$
$E \xrightarrow[CYC]{1} M$	$[QK^{19d}\cdot($← Ditto →$)+$			$QK^{19d}\cdot($← Ditto →$)$			$]\cdot$		$[$← Ditto →$]\cdot$	$[QKIR^{SKM}+IOCM^{Right}]$

Note 1 $\underline{Lo} \rightarrow M_{4,3,2,1}$ does not clear $M_{4,10}$. See Fig. 13-3 for $\underline{Lo} \rightarrow M_{4,10}$ logic.

	E → M Transfers Under Permuted Activity Control (Time Level, Inst, Time Level, Inst, Time Level, Inst, Permuted Activity Control, Parity Alarm)		E → M Broadside Transfers During VFF Operand Cycle (Time Level, Time Level, Inst, Time Level, Inst, VFF, Parity Alarm)		E → M Misc. Broadside Transfers (Time Level, Inst, Time Level, Inst., Time Level, Inst, Parity Alarm)		$E_1 \xrightarrow{0} M_1$ FLF (Time Level, Inst, Parity Alarm)
Pulse							
$E_1 \xrightarrow{0} M_1$	$[(QK^{19d}\cdot A+QK^{13d}\cdot QKIR^{ST}+QK^{13d}\cdot QKIR^{INS})\cdot QKIR^{PRM\ ACT_1}\cdot MPA]$	+	$[(QK^{02d}+QK^{09d}\cdot\overline{VMD^{XXO}_{E}}+QK^{21d}\cdot QKIR^{STE})\cdot QKM^{VFF}\cdot\overline{MPA}]$	+	$[PK^{09d}\cdot PKM^{VFF}+QK^{13d}\cdot QKIR^{SF}+QKIR^{16d}\cdot QKIR^{COM})\cdot\overline{MPA}]$	+	$[QK^{13d}\cdot QKIR^{FLF}\cdot\overline{MPA}]$
$E_1 \xrightarrow{1} M_1$	$[($← Ditto → $+QK^{19d}\cdot QKIR^{INS})\cdot$ ← Ditto →$]$	+	[← Ditto →]	+	[— Ditto —]		
$E_2 \xrightarrow{0} M_2$	$[($← Ditto → $+QK^{13d}\cdot QKIR^{INS})\cdot QKIR^{PRM\ ACT_2}\cdot MPA]$	+	[← Ditto →]	+	[— Ditto —]		
$E_2 \xrightarrow{1} M_2$	$[($← Ditto → $+QK^{19d}\cdot QKIR^{INS})\cdot$ ← Ditto →$]$	+	[← Ditto →]	+	[— Ditto —]		
$E_3 \xrightarrow{0} M_3$	$[($← Ditto → $+QK^{13d}\cdot QKIR^{INS})\cdot QKIR^{PRM\ ACT_3}\cdot MPA]$	+	[← Ditto →]	+	[— Ditto —]		
$E_3 \xrightarrow{1} M_3$	$[($← Ditto → $+QK^{19d}\cdot QKIR^{INS})\cdot$ ← Ditto →$]$	+	[— Ditto —]	+	[— Ditto —]		
$E_4 \xrightarrow{0} M_4$	$[($← Ditto → $+QK^{13d}\cdot QKIR^{INS})\cdot QKIR^{PRM\ ACT_4}\cdot MPA]$	+	[← Ditto →]	+	[— Ditto —]		
$E_4 \xrightarrow{1} M_4$	$[($← Ditto → $+QK^{19d}\cdot QKIR^{INS})\cdot$ — Ditto →$]$	+	[← Ditto →]	+	[— Ditto —]		
	Where $A=[IOCM^{IN\cdot NORMAL}+\overline{QKIR^{TSD}}]\cdot[PKIR^{0}_{CF_3}+\overline{QKIR^{SKM}}]$						

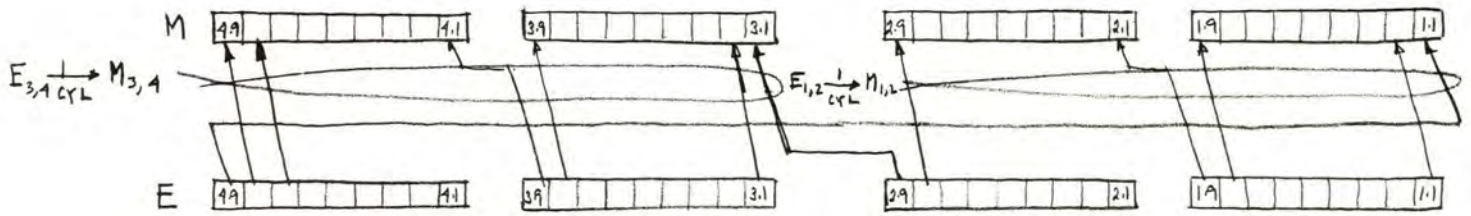
FIG. 13-5 E → M, M REGISTER RD CONTROL

	$M \xrightarrow{0,1} E$ TRANSFERS UNDER PERMUTED ACTIVITY CONTROL			$M \xrightarrow{0} E$ TRANSFERS UNDER PERMUTED ACTIVITY CONTROL		
PULSE	TIME LEVEL	INSTRUCTIONS	PERMUTED ACTIVITY CONTROL	TIME LEVEL	INST	PERMUTED ACTIVITY CONTROL
$M_1 \xrightarrow{0} E_1$	$[QK^{13d} \cdot (QKIR^{TSD} \cdot IOCM^{NORMAL} + QKIR^{SKM} + QKIR^{LD} + QKIR^{APX})] \cdot QKIR^{PRM\ ACT_1}$			$+ [QK^{13d} \cdot QKIR^{ITE}] \cdot QKIR^{PRM\ ACT_1}$		
$M_1 \xrightarrow{1} E_1$	[← DITTO →]		• DITTO			
$M_2 \xrightarrow{0} E_2$	[← DITTO →]		$\cdot QKIR^{PRM\ ACT_2}$	$+ [$← DITTO →$] \cdot QKIR^{PRM\ ACT_2}$		
$M_2 \xrightarrow{1} E_2$	[← DITTO →]		• DITTO			
$M_3 \xrightarrow{0} E_3$	[← DITTO →]		$\cdot QKIR^{PRM\ ACT_3}$	$+ [$← DITTO →$] \cdot QKIR^{PRM\ ACT_3}$		
$M_3 \xrightarrow{1} E_3$	[← DITTO →]		• DITTO			
$M_4 \xrightarrow{0} E_4$	[← DITTO →]		$\cdot QKIR^{PRM\ ACT_4}$	$+ [$← DITTO →$] \cdot QKIR^{PRM\ ACT_4}$		
$M_4 \xrightarrow{1} E_4$	[← DITTO →]		• DITTO			

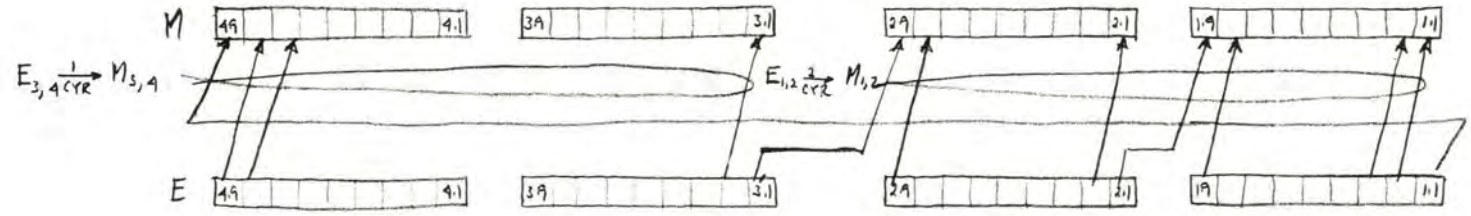
	$M \xrightarrow{0,1} E$ BROADSIDE TRANSFERS
PULSE	
$M_i \xrightarrow{0} E_i$ $M_i \xrightarrow{1} E_i$ $i = 1,2,3,4$	$\{QK^{21d} \cdot [(QKIR^{STORE} \cdot \overline{QKIR^{E}}) + (QKIR^{E} \cdot QKM^{VFF})] + QK^{23d} \cdot [(\overline{QKIR^{SPG}} \cdot \overline{QKIR^{E}} \cdot QKIR^{LOAD}) + (QKIR^{STE} \cdot VHD_E^{XXO} \cdot QKM^{VFF})]$ $+ QK^{13d} \cdot [QKIR^{COM} + QKIR^{SPF+SPG} + QKIR^{TSD} \cdot IOCM^{ASS'T}] + QK^{09d} \cdot QKM^{VFF} + PK^{12d} \cdot PKM^{VFF} + PK^{11d} \cdot [PI_2^{1} \cdot PI_5^{0}]$

	$M \oplus E \rightarrow E$ TRANSFER UNDER PERMUTED ACTIVITY CONTROL		
PULSE	TIME LEVELS	SED	PERMUTED ACTIVITY CONTROL
$M_1 \oplus E_1 \rightarrow E_1$	$[(QK^{13d} + QK^{23d}) \cdot QKIR^{SED}] \cdot$		$QKIR^{PRM\ ACT_1}$
$M_2 \oplus E_2 \rightarrow E_2$	[← DITTO →] $\cdot$		$QKIR^{PRM\ ACT_2}$
$M_3 \oplus E_3 \rightarrow E_3$	[← DITTO →] $\cdot$		$QKIR^{PRM\ ACT_3}$
$M_4 \oplus E_4 \rightarrow E_4$	[← DITTO →] $\cdot$		$QKIR^{PRM\ ACT_4}$

FIG. 13-7 $M \rightarrow E$; E REGISTER RD CONTROL



a) Cycle Left



b Cycle Right

Fig 13-6 Cyclic Register Transfer Between Registers E and M

RD Pulse	AE To E Transfers										
	VFF Instructions				AE Register Selection	Special Insts.		Misc. Insts.			
	Time Level	Inst Memory	Time Level	Operand Memory		Time Level	Inst	Time Level	Inst	Time Level	Inst
$A_{2,1} \rightarrow E_{2,1}$	$[PK^{10a} \cdot PKM^{VFF} + QK^{03a} \cdot QKM^{VFF} \cdot \overline{AEB}] \cdot VMD^{XX4} + [QK^{IN} \cdot QKIR^{A}] + [QK^{19a} \cdot QKIR^{INS}] + [QK^{22d} \cdot QKIR^{NA+UNA}]$										
$A_{4,3} \rightarrow E_{4,3}$	[← Ditto →]				$\cdot VMD^{XX4}$	+ [Ditto]					
$B_{2,1} \rightarrow E_{2,1}$	[← Ditto →]				$\cdot VMD^{XX5}$	$+ [QK^{19a} \cdot QKIR^{B}]$					
$B_{4,3} \rightarrow E_{4,3}$	[← Ditto →]				$\cdot VMD^{XX5}$	+ [Ditto]					
$C_{2,1} \rightarrow E_{2,1}$	[← Ditto →]				$\cdot VMD^{XX6}$	$+ [QKIR^{IN} \cdot QKIR^{C}]$					
$C_{4,3} \rightarrow E_{4,3}$	[← Ditto →]				$\cdot VMD^{XX6}$	$+ [QKIR^{IN} \cdot QKIR^{D}]$					
$D_{2,1} \rightarrow E_{2,1}$	[← Ditto →]				$\cdot VMD^{XX7}$	$+ [QKIR^{IN} \cdot QKIR^{D}]$					
$D_{4,3} \rightarrow E_{4,3}$	[← Ditto →]				$\cdot VMD^{XX7}$	$+ [QKIR^{IN} \cdot QKIR^{D}]$					

Fig 13-8 Arithmetic Element Registers To E RD Control

	IO BUFFER → E TRANSFERS					
PULSE	TIME LEVEL	INST	MISC.	TIME LEVEL	INST	MISC.
$IOBM_i \rightarrow E_i$	$(QK^{18d} \cdot QKIR^{TSD} \cdot IOCH^{IN}) + (PK^{26d} \cdot PKIR^{10S} \cdot PKIR'_{CF_1})$					

	MISCELLANEOUS TRANSFERS INTO E						
PULSE	TIME LEVEL	INST	MISC.	TIME LEVEL	INST	MISC	SPECIAL
$QKIR_{CF_{9-9}} \xrightarrow{1} E_{3.6-3.1}$							$PK^{09d}(PI'_1 \cdot PI^0_5)$
$XA_{2,1} \xrightarrow{1} E_{2,1}$	$(QK^{18d} \cdot QKIR^{ADX})$			$+(QK^{11d} \cdot QKIR^{X})$			$+PK^{09d}(PI'_2 \cdot PI^0_5)$
$Q_{2,1} \xrightarrow{1} E_{9,3}$	$(PK^{31d} \cdot PKIR^{JMP} \cdot PKIR'_{CF_4})$						
$N_{3.6-3.1} \xrightarrow{1} E_{3.6-3.1}$	$(PK^{26d} \cdot PKIR^{10S} \cdot PKIR'_{CF_1})$						$+(CSK^{02d})$
$K_{3.6-3.1} \xrightarrow{1} E_{4.6-4.1}$							(CSK^{02d})
$P_{2,1} \xrightarrow{\dot{1}} E_{2,1}$	$(PK^{25d} \cdot PKIR^{JX} \cdot EB^{0} \cdot XJ) + [PK^{31d} \cdot (PKIR^{JMP} \cdot PKIR'_{CF_3} + AEJ)] + (CSK^{09d})$						
$QKIR_{CF} \rightarrow E_4$	$(FK'_{8.1} \cdot PKIR^{SF})$						

FIG. 13-9 MISC. E REGISTER RD CONTROL

RD Pulse	⊔▸E Interchange Quarters Under Direct Permutation Control: Time Level	Instruction	Time Level	Inst	Permutation Control	⊔̄▸E Interchange Quarters Under Inverse Permutation Control: Time Level	Instruction	Time Level	Instruction	Time Level	Instruction	Permutation Control	Cycle Q'T'R's Right: Time Level	Instruction	Cycle Q'T'R's Left: Time Level	Instruction
$E_2 \dagger E_1$	$[QK^{13\beta} \cdot (\overline{QKIR^{FLF+FLG+SPF+SPG}}) \cdot (IOCM^{NORMAL} + \overline{QKIR^{TSD}}) + QK^{23\beta} \cdot QKIR^{SFD}]$				$\cdot QKIR^{PRM_{1+4}}$	$+ [QK^{11\beta} \cdot (\overline{QKIR^{FLF+FLG+SPF+SRE}}) \cdot \overline{QKIR^{SKM}} + QK^{18\beta} \cdot (\overline{QKIR^{TSD}} + IOCM^{IN\cdot NORMAL}) + QK^{21\beta} \cdot QKIR^{SFD}]$					$\cdot QKIR^{PRM_{2+4}}$	$+ [FK'_{\beta,1} \cdot PKIR^{FF}]$		
$E_3 \dagger E_1$	[Ditto]				$\cdot QKIR^{PRM_2}$	+ [Ditto]						$\cdot QKIR^{PRM_{1+3}}$				
$E_4 \dagger E_1$	[Ditto]				$\cdot QKIR^{PRM_{3+5}}$	+ [Ditto]						$\cdot QKIR^{PRM_{1+5}}$			$+ [FK'_{\beta,2} \cdot PKIR^{FLF}]$	
$E_5 \dagger E_2$	[Ditto]				$\cdot QKIR^{PRM_{1+5+6}}$	+ [Ditto]						$\cdot QKIR^{PRM_{3+5+7}}$	+ [Ditto]			
$E_4 \dagger E_2$	[Ditto]				$\cdot QKIR^{PRM_{2+7}}$	+ [Ditto]						$\cdot QKIR^{PRM_{2+6}}$				
$E_1 \dagger E_2$	[Ditto]				$\cdot QKIR^{PRM_{3+4}}$	+ [Ditto]						$\cdot QKIR^{PRM_{1+4}}$			+ [Ditto]	
$E_4 \dagger E_3$	[Ditto]				$\cdot QKIR^{PRM_{1+4+6}}$	+ [Ditto]						$\cdot QKIR^{PRM_{3+4+7}}$	+ [Ditto]			
$E_1 \dagger E_3$	[Ditto]				$\cdot QKIR^{PRM_2}$	+ [Ditto]						$\cdot QKIR^{PRM_2}$				
$E_2 \dagger E_3$	[Ditto]				$\cdot QKIR^{PRM_{3+5+7}}$	+ [Ditto]						$\cdot QKIR^{PRM_{1+5+6}}$			+ [Ditto]	
$E_1 \dagger E_4$	[Ditto]				$\cdot QKIR^{PRM_{1+5}}$	+ [Ditto]						$\cdot QKIR^{PRM_{2+5}}$	+ [Ditto]			
$E_2 \dagger E_4$	[Ditto]				$\cdot QKIR^{PRM_{2+6}}$	+ [Ditto]						$\cdot QKIR^{PRM_{2+7}}$				
$E_3 \dagger E_4$	[Ditto]				$\cdot QKIR^{PRM_{3+4+7}}$	+ [Ditto]						$\cdot QKIR^{PRM_{1+4+6}}$			+ [Ditto]	

Fig 13-10 E Register Permutation RD Control

Pulse	LC→E SE Complement E Under Sign Extension Control — Time Level	LC→E SE Complement E Under Sign Extension Control — Inst.	Sign Extension Control	LC→E 1,3,2,1 Complement E — Time Level	Inst	Extended Activity Control	Time Level	Inst	Time Level S	Inst	LC→E 1,3 Only — Time Level	Inst	Misc.
LC→E1	$[(QK^{19B} \cdot QKIR^{LD+ADX+COM}) \cdot (QKIR^{Ext\ Act_1} \cdot \overline{QKIR^{Act_1}}) \cdot (A \cdot QKIR^{f_1} + E'_{2,9} \cdot QKIR^{f_2})]$			$+[(QK^{15B} \cdot QKIR^{COM}) \cdot QKIR^{Ext\ Act_1}]$			$+[QK^{19B} \cdot QKIR^{INS}]$		$+[QK^{10\alpha+21\beta+22\beta} \cdot QKIR^{ITA}]$				
LC→E2	$[(\leftarrow$ Ditto $\rightarrow) \cdot (QKIR^{Ext\ Act_2} \cdot \overline{QKIR^{Act_2}}) \cdot (A \cdot QKIR^{f_3} + E'_{1,9} \cdot QKIR^{f_2} + B)]$			$+[(\leftarrow$ Ditto $\rightarrow) \cdot QKIR^{Ext\ Act_2}]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$				
LC→E3	$[(\leftarrow$ Ditto $\rightarrow) \cdot (QKIR^{Ext\ Act_3} \cdot \overline{QKIR^{Act_3}}) \cdot (C \cdot QKIR^{f_1+f_3} + E'_{4,9} \cdot QKIR^{f_2})]$			$+[(\leftarrow$ Ditto $\rightarrow) \cdot QKIR^{Ext\ Act_3}]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow] + [QK^{10\beta} \cdot QKIR^{X} \cdot X'_{2,9}]$				
LC→E4	$[(\leftarrow$ Ditto $\rightarrow) \cdot (QKIR^{Ext\ Act_4} \cdot \overline{QKIR^{Act_4}}) \cdot (D \cdot QKIR^{f_1+f_3} + E'_{3,9} \cdot QKIR^{f_2})]$			$+[(\leftarrow$ Ditto $\rightarrow) \cdot QKIR^{Ext\ Act_4}]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow] + [\leftarrow$ Ditto $\rightarrow]$				

Where $A = D \cdot \overline{QKIR^{Act_4}} + E'_{4,9} \cdot QKIR^{Act_4}$

$B = (A + QKIR^{Act_1})(E'_{1,9} + \overline{QKIR^{Act_1}}) \cdot QKIR^{f_1}$

$C = (B + A \cdot QKIR^{f_3})\overline{QKIR^{Act_2}} + E'_{2,9} \cdot QKIR^{Act_2}$

$D = (C + QKIR^{Act_3})(E'_{3,9} + \overline{QKIR^{Act_3}})$

Pulse	LO→E SE Clear E Under Sign Extension Control — Time Level	LO→E SE Clear E Under Sign Extension Control — Inst.	Sign Extension Control	LO→E 1,3,2,1 — Time Level	Inst	Time Level	Inst	Misc.	Clear E — Time Level	Inst	Time Level	Inst	Time Level	Inst	LO→E 1,3 Only — Time Level	Inst	Misc.
LO→E1	$[(QK^{19\alpha} \cdot QKIR^{LD+ADX+COM}) \cdot (QKIR^{Ext\ Act_1} \cdot \overline{QKIR^{Act_1}})]$			$+[QK^{10\alpha} \cdot QKIR^{\bar{E} \cdot \overline{FLG} \cdot \overline{FLF} \cdot \overline{SKM}}]$		$+[PK^{25\alpha} \cdot PKIR^{10S} \cdot PKIR'_{CF_1} \cdot EB^{0}]$			$+[PK^{09\alpha} \cdot \overline{VMD_E^{XXO}} \cdot PKM^{VFF}]$		$+[QK^{02\alpha} \cdot QKM^{VFF}]$		$+[PK^{01\alpha} \cdot PI'_{S} \cdot PI_S^{0}]$				
LO→E2	$[(\leftarrow$ Ditto $\rightarrow) \cdot (QKIR^{Ext\ Act_2} \cdot \overline{QKIR^{Act_2}})]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$				
LO→E3	$[(\leftarrow$ Ditto $\rightarrow) \cdot (QKIR^{Ext\ Act_3} \cdot \overline{QKIR^{Act_3}})]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$				
LO→E4	$[(\leftarrow$ Ditto $\rightarrow) \cdot (QKIR^{Ext\ Act_4} \cdot \overline{QKIR^{Act_4}})]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$			$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$		$+[\leftarrow$ Ditto $\rightarrow]$				

FIG. 13-11 LC→E, LO→E E REGISTER RD CONTROL

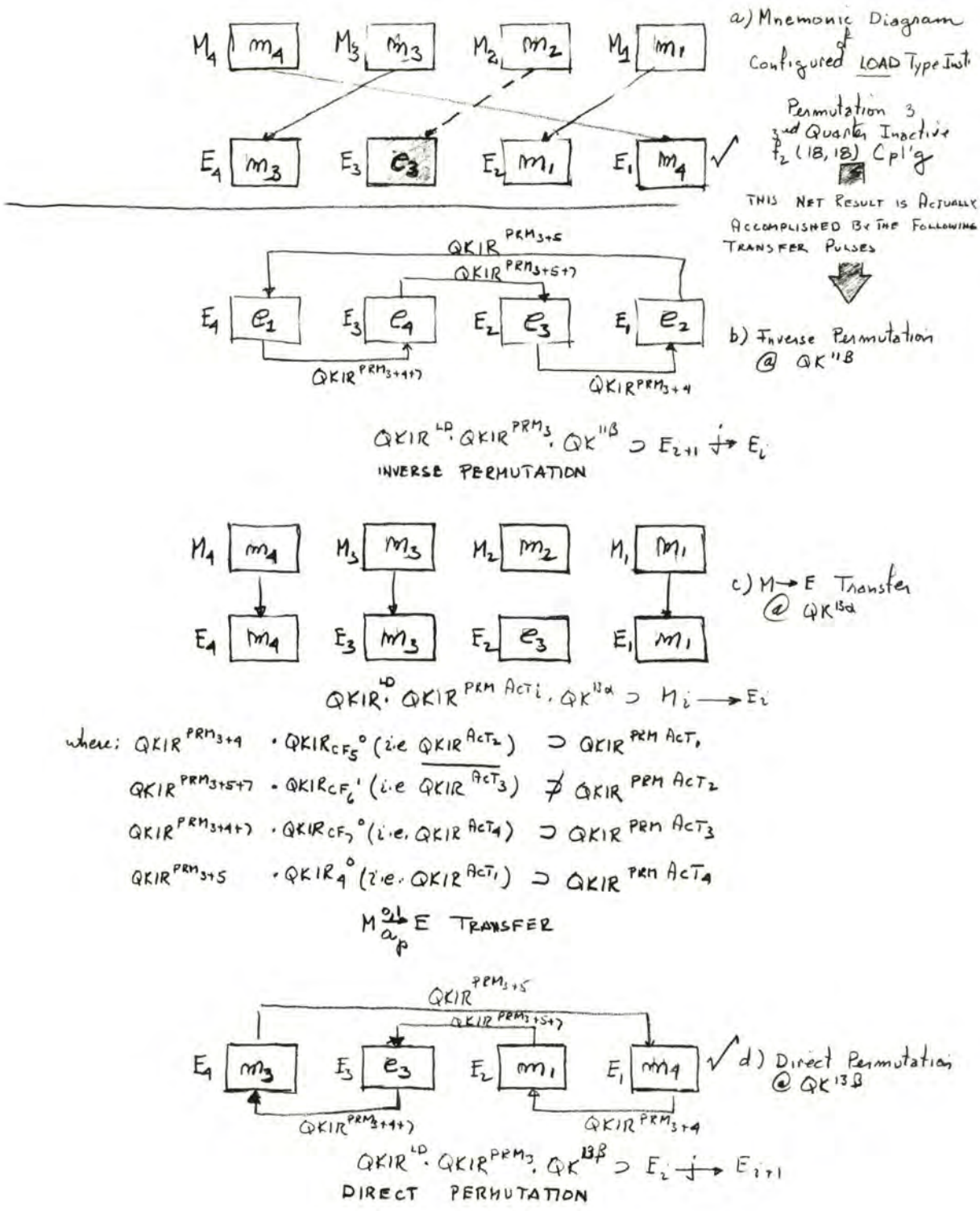


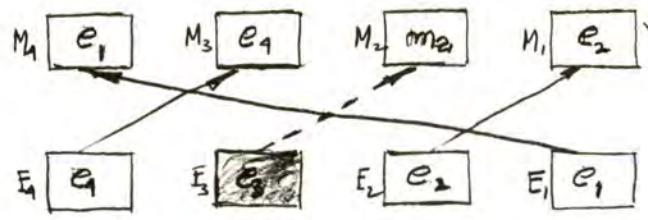
Fig 13-12 CONFIGURED LOAD TYPE INSTRUCTION

ILLUSTRATING BASIC PERMUTATION AND M → E TRANSFERS IN THE EE

Permutation 3

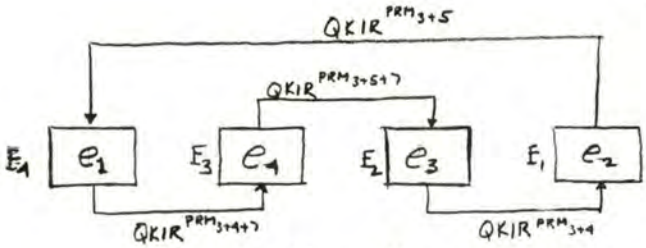
3rd Quarter Inactive

Coupling t_2 (18, 18)



a) Mnemonic Diagram of Configured STORE Type Inst. Permutation 3 3rd Quarter Inactive

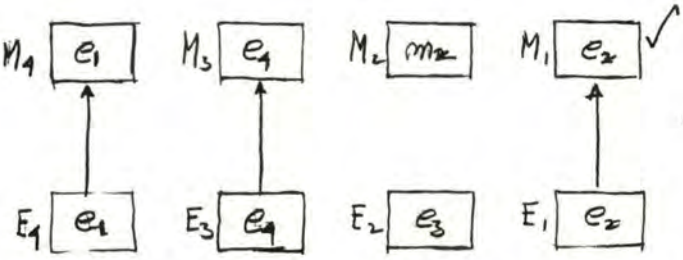
THIS NET RESULT IS ACTUALLY ACCOMPLISHED BY THE FOLLOWING TRANSFER PULSES.



b) Inverse Permutation @ QK 11β

$QKIR^{ST} \cdot QKIR^{PRM_3} \cdot QK^{11\beta} \supset E_{i+1} \overset{j}{\rightarrow} E_i$

INVERSE PERMUTATION

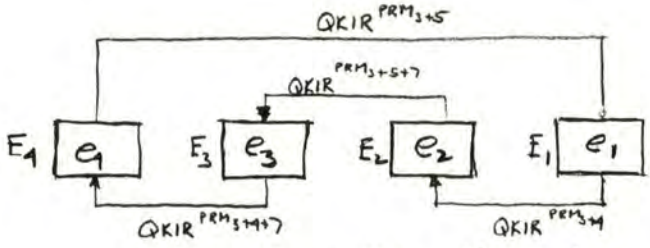


c) E→M Transfer @ QK 13α

$QKIR^{ST} \cdot QKIR^{PRM\ Acti} \cdot QK^{13\alpha} \supset E_i \rightarrow M_i$

when $QKIR^{PRM\ Acti}$ (See Fig 13-12)

$E \xrightarrow[\alpha_p]{\alpha_1} M$ TRANSFER



d) Direct Permutation @ QK 13β

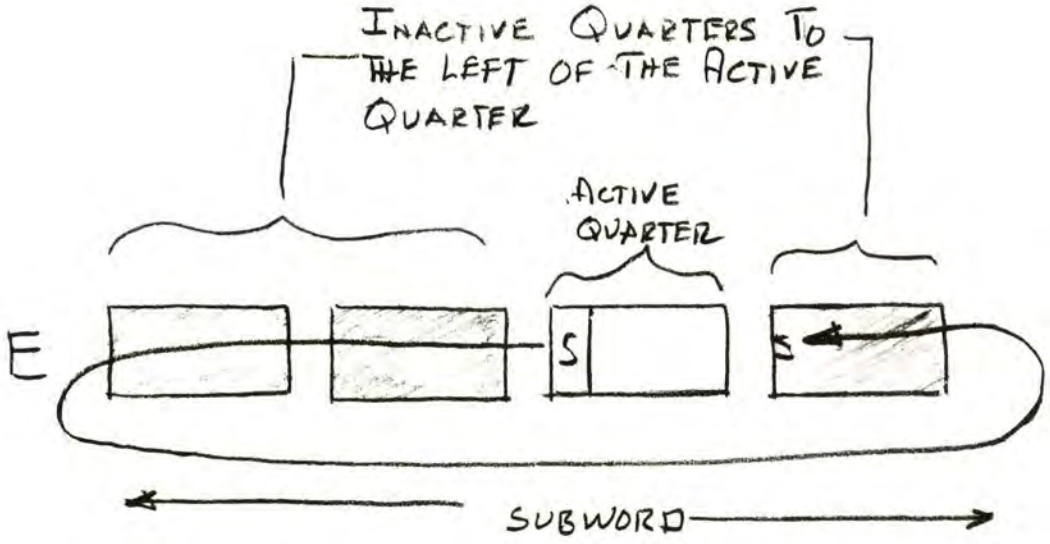
$QKIR^{ST} \cdot QKIR^{PRM_3} \cdot QK^{13\beta} \supset E_i \overset{j}{\rightarrow} E_{i+1}$

DIRECT PERMUTATION

FIG 13-13 CONFIGURED STORE TYPE INSTRUCTION ILLUSTRATING BASIC PERMUTATION AND M→E TRANSFERS IN THE EE

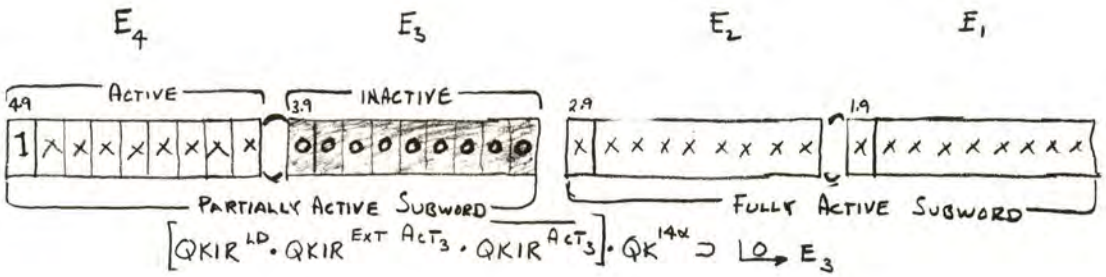
PERMUTATION 3

3RD QUARTER INACTIVE



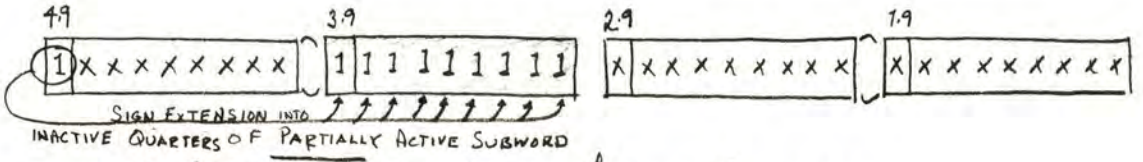
GENERAL RULE: SIGN DIGIT OF AN ACTIVE QUARTER OF A PARTIALLY ACTIVE SUBWORD EXTENDS TO THE LEFT FILLING INACTIVE QUARTERS UNTIL AN ACTIVE QUARTER IS AGAIN MET, THIS MUST BE INTERPRETED IN TERMS OF THE POSSIBLE PARTIALLY ACTIVE SUBWORDS.

FIG 13-14 SIGN EXTENSION IN E REGISTER



$[QKIR^{LD} \cdot QKIR^{EXT\ ACT_3} \cdot QKIR^{ACT_3}] \cdot QK^{19\alpha} \supset \underline{LO} \rightarrow E_3$

a) CLEAR PULSE @ $QK^{19\alpha}$



$[QKIR^{LD} \cdot QKIR^{EXT\ ACT_3} \cdot QKIR^{ACT_3}] \cdot [E'_{4.9} \cdot QKIR^{f_2}] \cdot QK^{19\beta} \supset \underline{LC} \rightarrow E_3$

b) COMPLEMENT PULSE @ $QK^{19\beta}$

where: $[QKIR^{ACT_4} \cdot \overline{QKIR^{f_4}}] + [QKIR^{ACT_3} + QKIR^{ACT_2} \cdot QKIR^{f_1+f_3} + QKIR^{ACT_1} \cdot QKIR^{f_1}] \supset QKIR^{EXT\ ACT_3}$

TRUE (FOR THE CASE CHOSEN)

FALSE (FOR THE CASE CHOSEN)

FIG 13-15 CONFIGURED LOAD TYPE INSTRUCTION ILLUSTRATING SIGN EXTENSION

(OCCURS AFTER THE PERMUTATIONS AND M→E TRANSFERS IN THE EE.)

3rd Quarter Inactive

Coupling f_2 (18, 18)

PERMUTATION 3

CHAPTER 14

ARITHMETIC ELEMENT

TABLE OF CONTENTS

14-1 INTRODUCTION

14-2 ARITHMETIC ELEMENT INSTRUCTION CLASSIFICATION

- 14-2.1 $\overline{AK}$ TYPE INSTRUCTIONS
- 14-2.2 AK TYPE INSTRUCTIONS
  - 14-2.2.1 Z FLIP-FLOPS
- 14-2.3 $\overline{ASK}$ TYPE INSTRUCTIONS
  - 14-2.3.1 ADD AND SUB
  - 14-2.3.2 DSA
- 14-2.4 ASK TYPE INSTRUCTIONS
- 14-2.5 $\overline{\text{COUNT IN D}}$ TYPE INSTRUCTIONS
  - 14-2.5.1 MULTIPLICATION
  - 14-2.5.2 DIVISION
- 14-2.6 COUNT IN D TYPE INSTRUCTIONS
  - 14-2.6.1 SCA, SCB, SAB, CYA AND CAB
  - 14-2.6.2 NOA AND NAB
  - 14-2.6.3 $\overline{\text{TLY}}$
- 14-2.7 OPR^{AE} VS $\overline{OPR^{AE}}$ ARITHMETIC ELEMENT INSTRUCTIONS

14-3 ARITHMETIC ELEMENT REGISTER OPERATION

- 14-3.1 STANDARD TRANSFERS
- 14-3.2 SPECIAL LOGICAL TRANSFERS
- 14-3.3 D REGISTER COUNTER
- 14-3.4 SHIFTING OPERATION
- 14-3.5 CARRYING OPERATION

14-4 ARITHMETIC ELEMENT LEVEL LOGIC

- 14-4.1 $AKIR_{OP}$ $AKIR_{CF}$ REGISTERS
  - 14-4.1.1 $AKIR_{OP}$ DECODING
  - 14-4.1.2 $AKIR_{CF}$ DECODING
- 14-4.2 ALL ZEROS (A_i^0), ALL ONES (A_i^1) LEVEL LOGIC
- 14-4.3 SIGMA (σ) LEVEL LOGIC
- 14-4.4 SHIFT COUPLING UNITS
- 14-4.5 CARRY COUPLING UNITS
- 14-4.6 AEJ LEVEL LOGIC
- 14-4.7 $\overset{0}{\longrightarrow}$ AEP LEVEL LOGIC

14-5 ARITHMETIC ELEMENT REGISTER DRIVER LOGIC

- 14-5.1 $\overline{D_i}$ + 1 $\longrightarrow$ D_i RD LOGIC
- 14-5.2 A REGISTER SHIFT RD LOGIC
- 14-5.3 B REGISTER SHIFT RD LOGIC
- 14-5.4 MULTIPLY STEP RD LOGIC
- 14-5.5 COMPLEMENT C RD LOGIC
- 14-5.6 CARRY RD LOGIC

14-5.7 PARTIAL ADD RD LOGIC
14-5.8 A, B, C AND D CLEAR RD LOGIC
14-5.9 Z PULSE GATE AND RD LOGIC
14-5.10 A REGISTER COMPLEMENT RD LOGIC
14-5.11 B REGISTER COMPLEMENT RD LOGIC
14-5.12 D REGISTER COMPLEMENT RD LOGIC
14-5.13 E → A, B, C AND D RD LOGIC
14-5.14 A —j→ B AND B —j→ A RD LOGIC

LIST OF FIGURES

14-1 AE INSTRUCTIONS CLASSIFIED BY COUNTER ACTIVITY
14-2 FUNCTIONAL BLOCK DIAGRAM OF THE AE FOR $\overline{AK}$ TYPE INSTRUCTIONS
14-3 FUNCTIONAL BLOCK DIAGRAM OF THE AE FOR AK TYPE INSTRUCTIONS
14-4 EFFECT OF AK TYPE INSTRUCTIONS ON Z FLIP-FLOP
14-5 ADDITION OVERFLOW LOGIC
14-6 SAB EXAMPLE SHOWN FOR f_2 FRACTURE AND POSITIVE AND NEGATIVE OPERAND SUBWORDS
14-7 CAB EXAMPLE SHOWN FOR f_2 FRACTURE AND POSITIVE AND NEGATIVE OPERAND SUBWORDS
14-8 FUNCTION OF ASK AND D COUNTERS IN ASK TYPE INSTRUCTIONS
14-9 AE REGISTER FUNCTION FOR AK TYPE INSTRUCTIONS
14-10 AOP AND $\overline{AOP}$ AE INSTRUCTIONS
14-11 TYPICAL AE REGISTER OPERATIONS
14-12 JAM TRANSFERS BETWEEN A AND B
14-13 TRANSFERS GATED BY |PAD → PULSE
14-14 TRANSFERS GATED BY |MUL STEP → PULSE
14-15 D REGISTER COUNT CIRCUIT
14-16 A REGISTER SHIFT LOGIC
14-17 A REGISTER CARRY LOGIC
14-18 $AKIR_{OP}$ 1ST LEVEL DECODERS
14-19 $AKIR_{OP}$ OP DECODERS
14-20 $AKIR_{OP}$ CLASS DECODER
14-21 AE CONFIGURATION LOGIC
14-22 A_i ALL ZEROS, ALL ONES NETS
14-23 SIGMA LOGIC
14-24 A SHIFT COUPLING UNIT LOGIC
14-25 B SHIFT COUPLING UNIT LOGIC
14-26 EXAMPLES OF SHIFT COUPLING
14-27 CARRY COUPLING UNIT LOGIC
14-28 EXAMPLES OF CARRY COUPLING
14-29 AE JUMP NET LOGIC
14-30 |0 → AEP LOGIC
14-31 AE RD PULSES
14-32 AE PULSE GATE LOGIC
14-33 D COUNTER RD LOGIC

14-34 A REGISTER SHIFT RD LOGIC

14-35 B REGISTER SHIFT RD LOGIC

14-36 MULTIPLY-STEP RD LOGIC

14-37 COMPLEMENT C RD LOGIC

14-38 CARRY RD LOGIC

14-39 PARTIAL ADD RD LOGIC

14-40 CLEAR A, B, C AND D RD LOGIC

14-41 Z PULSE GATE AND RD LOGIC

14-42 COMPLEMENT A RD LOGIC

14-43 COMPLEMENT B RD LOGIC

14-44 COMPLEMENT D RD LOGIC

14-45 E $\xrightarrow{1}$ A, B, C, D RD LOGIC

14-46 E $\xrightarrow{j}$ B AND B $\xrightarrow{j}$ A RD LOGIC

CHAPTER 14
ARITHMETIC ELEMENT

14-1 INTRODUCTION

The Arithmetic Element has a two-fold nature: (1) it is used in the execution of a specific group of operation codes, (2) its A, B, C and D registers are part of the V_{FF} Memory. Since the V_{FF} Memory is discussed in detail in Chapter 11, this chapter only briefly describes the V_{FF} register driver logic affecting the A, B, C and D registers and instead emphasizes the role of the Arithmetic Element in processing instructions.

The chapter proceeds by first classifying all the Arithmetic Element operation codes according to the use they make (or do not make) of the AK, ASK and D counters. The execution logic of each operation code in the sub-classes is then discussed. The net effect is to bring into focus the ways in which the Arithmetic Element can process data.

The logic circuits integrated into the A, B, C and D registers are next discussed. This includes a discussion of the shift and carry circuits and the D counter circuit.

Since there are a large number of special purpose logic nets, these are then itemized and discussed.

Finally all the Arithmetic Element register driver logic is tabulated and discussed.

14-2 ARITHMETIC ELEMENT INSTRUCTION CLASSIFICATION

The computer currently has twenty-nine operation codes that make use of the Arithmetic Element. It is convenient to classify these operation codes in terms of the use they make of the AK, ASK and D counters.

Fig. 14-1 shows the Arithmetic Element operation codes classified in this manner.

14-2.1 $\overline{\text{AK}}$ TYPE INSTRUCTIONS. The execution logic for these instructions has the following features:

1) During the operand cycle, the Arithmetic Element register driver logic is controlled entirely by PK or QK time levels.

2) The register transfers in the Arithmetic Element are simple, i.e., they do not involve complex logic. For example, the only Arithmetic Element register driver pulses involved in these instructions are those that clear, complement, and transfer the contents of the E register into the Arithmetic Element registers. (The complementing pulses are used only in two operation codes, INS and ITA.)

Fig. 14-2 shows a functional block diagram of the Arithmetic Element for $\overline{AK}$ type instructions.

The load and store type instructions involve simple register transfers between E and the Arithmetic Element registers (or vice versa). These instructions have the following features:

1) During an Arithmetic Element load or store type instruction, the operand always passes through the Exchange Element. This path is followed so that even though the operand is stored or is to be stored in one of the Arithmetic Element registers of the V_{FF} Memory, it still may be configured. Thus in a LDA instruction it is possible to load A_4 with the content of A_1. The initial content of A is placed in the Exchange Element, configured, and the configured operand placed back in A.

2) When a LDD instruction is executed (or for that matter <u>any</u> instruction which places an operand in D), the bit placed in $D_{1.9}$ is transferred by a parallel path into Y_1. The Y bits are used in sign control only. The programmer has no access to the Y bits.

Generally the procedure in the logical instructions INS, ITA and UNA involves transferring Arithmetic Element data into the Exchange Element; logically processing the data in the Exchange Element; and then, usually transferring the result back into the Arithmetic Element. (In INS and ITA the execution logic for the instruction does require certain complementing logic to occur in the Arithmetic Element.)

The Z flip-flops are not affected by any of the $\overline{AK}$ type instructions.

14-2.2 AK TYPE INSTRUCTIONS. The execution logic for these instructions involves quite different Arithmetic Element features than the $\overline{AK}$ instructions. For example:

1) In these instructions, the Arithmetic Element is "loosely" coupled to the central computer, i.e., once the operand is transferred into the Arithmetic Element, the Arithmetic Element time control is turned over to the AK counter. While the AK type instruction is being executed, the central computer can execute any additional instructions which do not involve the Arithmetic Element.

2) In addition to the standard register driver logic for clearing, complementing, etc., the AK type instructions can make use of the following "multiply step" and "partial addition" register driver logic:

$$A_{i.(j+1)} \oplus C_{i.j} \xrightarrow{j} A_{i.j}$$
$$A_{i.(j+1)} \xrightarrow{0} C_{i.j}$$

} Multiply Step Logic

$$A_{i.j} \cdot D_{i.j} \xrightarrow{1} C_{i.j}$$
$$D_{i.j} \oplus A_{i.j} \xrightarrow{j} A_{i.j}$$

} Partial Add Logic

3) These instructions can use the shift and carry circuits integrated into the A and B registers.

4) These instructions can use the D register to count.

5) These instructions make functional distinctions among the Arithmetic Element registers.

Fig. 14-3 shows a functional block diagram of the Arithmetic Element for AK type instructions.

In all of the AK type instructions, an operand is brought from memory into the Arithmetic Element before the instruction is executed. The fact that the operand may be located in the A, B, C or D register of the V_{FF} Memory is not a contradiction of the above statements, but merely emphasizes the fact that the register hardware may wear the V_{FF} hat as well as the Arithmetic Element hat.

Notice that in all the AK instructions but TLY, the operand brought from the Memory Element is placed in the D register. In the case of TLY, the operand is placed in the A register. This means that before AK actually executes the instruction, the only data found in non-operand registers will be that left from a previous instruction.

Just as the D register generally contains the instruction operand, the A register generally "accumulates" the result of the instruction, e.g. if a series of additions are performed, the running sum is found in A.

14-2.2.1 Z FLIP-FLOPS. The ability of A to accumulate the results of a series of instructions leads to the possibility of A overflowing. In the case of addition, overflow occurs when the accumulation exceeds the size of the subword in which the sum or difference is accumulated. One of the functions of the Z flip-flops is to indicate these overflows. The Z flip-flops are also used in sign control. Only the Z flip-flops in the sign quarters of A are used for any of these purposes. (The sign quarter is the left most quarter of an active subword.)

Fig. 14-4 shows how the different instructions affect and make use of Z. Z is cleared at the beginning of the four arithmetic instructions: ADD, SUB, MUL and DIV. In the case of the other instructions, except SCale and NOrmalize, Z is left in the state determined by the previous instruction.

In the case of the SCale and NOrmalize instructions involving A, the content of Z may be shifted into A. (This will be discussed in greater detail when the SCale and NOrmalize instructions are described.) Z is used in the MULtiply and DIVide instructions for sign control. If Z_i and Y_i (which remember the sign of A and D, respectively) are the same, the product or quotient is given a positive sign; if Z and Y differ, the product or quotient is given a negative sign. Note that Z is also used in the DIVide instruction as an overflow indicator. (The logic for accomplishing these two functions in the same instruction will be discussed later in the chapter.) The other two instructions which use Z as an overflow indicator are ADD and SUB.

Certain instructions always leave Z in a cleared state. These instructions are: MULtiply, which uses Z just for sign control; and SCale and NOrmalize (involving A), which can shift the contents of Z into A. The other AK type instructions neither make use of nor affect the Z flip-flops. These instructions are DSA, TLY, CY-, and SCB.

14-2.3 $\overline{\text{ASK}}$ TYPE INSTRUCTIONS. The ADD, SUB and DSA instructions which make up this class are executed by a common basic logic. Note that the B register is not used in the $\overline{\text{ASK}}$ instructions which use AK.

14-2.3.1 ADD AND SUB. A ONE's complement ring adder is formed in which an end around carry occurs. E.g., suppose it is desired to perform the following subtraction:

0 1 1 1	subtrahend
0 1 0 0	minuend
0 0 1 1	difference

In the computer the minuend will be complemented and the terms added, i.e.,

sign bits	0 1 1 1	subtrahend
	1 0 1 1	complemented minuend
1	0 0 1 0	
end around carry	1	
sign bit	0 0 1 1	difference

As we saw earlier in the chapter, it is possible for overflow to occur in the addition and subtraction processes. Fig. 14-5 summarizes the basic overflow logic for ADDition. Note that SUBtraction is simply addition with the minuend complemented.

The key fact is that overflow can only occur when the sign of the augend and addend are the same. The overflow rules are as follows:

1) If the signs of the terms are positive and the sign of the sum is positive, overflow has not occurred.

2) If the signs of the terms are positive and the sign of the sum is negative, overflow has occurred.

3) If the signs of the terms are negative and the sign of the sum is negative, overflow has not occurred.

4) If the signs of the terms are negative, and the sign of the sum is positive, overflow has occurred.

5) If the signs of the terms differ, then overflow can not occur.

Another way of saying this is: "Overflow can occur in ADDition only when the sign of the sum differs from the sign of both terms".

The computer logic for ADDition overflow is also shown in Fig. 14-5. Z is cleared at the beginning of the instruction. A "partial sum" of the signs of the augend and addend is then stored in Z. Note that ONE's in the Z flip-flops indicate that the signs of the terms are similar. A reset Z pulse is then fired which clears Z to ZERO if the sign of the sum is the same as the sign of the augend. Note that in the non-overflow cases where this is true, Z already contains ZERO and thus the clear pulse is not necessary.

14-2.3.2 DSA. This instruction is very similar to the ADD instruction except that the complete carry required in the addition process is not executed. For this reason no overflow problems arise. Hence the Z flip-flops are left unaffected by the instruction.

In DSA the logical sum of A and D is placed in A and the accumulated logical product is placed in C. That is,

Logical Sum (or "partial add"): $A \oplus D \longrightarrow A$

Accumulated logical product (carry): $C + (A \cdot D) \longrightarrow C$

Suppose that a DSA is executed with the following data:

Before Processing Data

0	1	0	1	1	D	} Operand Data
0	1	1	0	1	A	
0	0	0	0	0	C	

The DSA instruction leaves the D register unaltered and the A and C with the following results:

After Processing Data

0	1	0	1	1	D	} Result
0	0	1	1	0	A	
0	1	0	0	1	C	

14-2.4 ASK TYPE INSTRUCTIONS. These instructions are characterized by an operation or series of operations which (somewhere in the execution of the instruction) are repeated a finite number of times. The usual function of the ASK counter is to keep track of the number of iterations. Because of this iterative characteristic, the execution of the instruction generally requires considerably more than the usual instruction time.

14-2.5 COUNT IN D TYPE INSTRUCTIONS. This subclass is made up of the MUL and DIV instruction. In these instructions all the Arithmetic Element registers are used as well as the Y and Z flip-flops.

14-2.5.1 MULTIPLICATION. In this instruction, the multiplicand, which is the operand from memory, is loaded into D. The multiplier is the data left in A from a previous instruction.

Early in the execution of the instruction, the multiplier is transferred from A into B and A is then cleared. There then begins an iteration of "partial add - multiply step" cycles. It is the function of the ASK counter to see that the correct number of iterations occur. The actual number depends on the size of the subwords used, i.e., on the fracture.

The partial add performs the following logic:

Partial Add

$$A_{i.j} \cdot D_{i.j} \xrightarrow{1} C_{i.j}$$

$$A_{i.j} \oplus D_{i.j} \xrightarrow{j} A_{i.j}$$

The before and after states of the Arithmetic Element registers during a partial add might typically be as follows:

Before PAD

D [1 1 1 1]

C [0 1 0 0]

A [0 0 1 1] [0 1 1 1] B

PAD conditional on right most bit being 1

After PAD

D [1 1 1 1]

C [0 1 1 1]

A [1 1 0 0] [0 1 1 1] B

The multiply step pulse processes the carries in C and shifts the running sum in AB one bit to the right. This is done by a single pulse. The logic involved in this operation is as follows:

Multiply Step

$$A_{i.(j+1)} \oplus C_{i.j} \xrightarrow{j} A_{i.j.}$$

$$A_{i.(j+1)} \xrightarrow{0} C_{i.j}$$

$$A_{i.1} \xrightarrow{j} B_{i.9}$$

$$B_{i.(j+1)} \xrightarrow{j} B_{i.j}$$

The effect on the Arithmetic Element registers of following the PAD described above with an MS (multiply step) is as follows:

After MS

D | 1 1 1 1 |

C | 0 1 1 0 |

A | 0 0 0 1 | | 0 0 1 1 | B

This process is iterated the correct number of times and then the carries left in C are absorbed into the A register by a carry pulse ($\xrightarrow{CRY}$ A). Note that this complete carry is performed only once during the execution of the MULtiply instruction. At the end of the instruction, AB contains the product. The major half of the product (most significant bits) is in A and the minor half of the product (least significant bits) is in B.

The fracture (f) specified in the MULtiply instruction determines the AB subword length. The AB possibilities are shown in the table on Fig. 14-3. For example, if an f_2 (18,18) fracture is specified, two independent products will be formed if there is more than one active subword involved. A 36 bit product will be contained in A_4 - A_3 - B_4 - B_3 with A_4 the sign quarter. At the same time, a 36 bit product will be formed in A_2 - A_1 - B_2 - B_1 with A_2 the sign quarter. However, in the case of f_3 (27,9), the 9 bit subword product will not be correctly generated.* The f_4 (9,9,9,9) form must be used to obtain the correct product in the right quarter.

14-2.5.2 DIVISION. In this instruction, the divisor, which is the operand, is loaded into D. The dividend is the data left in AB from a previous instruction. The major half of the dividend is located in A and the minor half of the dividend is located in B. The dividend must have the same form as the product left by a MUL.

The C register is used to keep track of the carries involved in the partial adds, just as in MULtiplication.

At the end of the DIVide instruction, the A register contains the signed quotient and the B register contains the signed remainder. The sign logic is based on the following simple algebra:

* At the moment the ASK counter can be used for only one subword length at a time. A modification will be made so that ASK can count both for 27 and 9 bit subwords simultaneously.

$$\frac{\text{DIVIDEND}}{\text{DIVISOR}} = \text{QUOTIENT} + \frac{\text{REMAINDER}}{\text{DIVISOR}}$$

or

$$\frac{AB}{D} = A + B$$

The sign rules are:

1) The sign of the quotient is positive if the signs of the dividend and divisor are the same, and negative if these signs are different.

2) The remainder always has the same sign as the dividend.

The mechanics of the DIVide instruction will be explained by examining the example below. The example has been chosen because it produces an overflow. Let the operand in D and data in AB be the following:

D | 0 1 0 | 0 0 0 | 0 0 1 |

AB | 0 1 0 | 0 0 0 | 0 0 1 | | 0 0 0 | 0 0 0 | 0 1 0 |

If AB is positive, AB is complemented. (The signs of D and AB are remembered by Y and Z respectively.) Thus,

Y | 0 | D | 0 1 0 | 0 0 0 | 0 0 1 | ASK = 170

Z | 0 | AB | 1 0 1 | 1 1 1 | 1 1 0 | | 1 1 1 | 1 1 1 | 1 0 1 |

Because of the f_4 fracture, ASK is preset to 170. ASK now counts out the "partial add - carry" loops as they are executed.

The first PAD pulse leaves the A register looking as follows:

A | 1 1 1 | 1 1 1 | 1 1 1 |

If the sign of A now differs from that of D, D is left unchanged and AB is shifted one bit to the left by the CRY pulse. This leaves the registers as follows:

	D	010 000 001	ASK = 171
PAD	A	111 111 111	
CRY	AB	111 111 111	111 111 011

This process is repeated. Note that in the next iteration, the sign of A is the same as the sign of D, therefore D is complemented. This is done so that the PAD pulse always adds terms of unlike signs, i.e., a subtraction always takes place. Thus,

	D	101 111 110	ASK = 172
PAD	A	010 000 000	
CRY	AB	100 000 001	111 110 110

Again the process is repeated. In the next partial add, $D_{i.9}$ is "carried" into A because a final "fix up" pad is executed on the basis of the sign bit in A being ZERO (i.e., positive). Thus,

	D	101 111 110	ASK = 002
	A	111 111 111	
	AB	111 111 111	100 000 000

A and B are now interchanged, so that the quotient is now in A and the remainder is in B.

AB	100 000 000	111 111 111

Up to this point both Y and Z have been in the ZERO state. Several things now occur simultaneously:

1) Since Z is ZERO, B is complemented.

2) Since Z = Y, A is <u>not</u> complemented.

3) Since $A_{i.9}$ is ONE, Z is set to ONE (indicating an overflow).

Thus,

Z	1	AB	100 000 000	000 000 000

Note that since Z indicates an overflow, A does not contain the right quotient. However, the right quotient can be obtained by following the DIVide instruction with a NOrmalize instruction.

14-2.6 COUNT IN D TYPE INSTRUCTIONS

14-2.6.1 SCA, SCB, SAB, CYA, CYB AND CAB. SCaling and CYcling are similar type instructions. SCaling effectively multiplies the data by a positive or negative power of 2 while preserving the significance of the sign bit. CYcling simply rotates the data (including the sign bit) left or right within the subword. In both SCale and CYcle type instructions, the sign quarters of the operand in D prescribe the number of shifts to occur.

Fig. 14-6 shows an example of a SCale AB instruction (SAB) in which an f_2 (18,18) fracture was specified. It is assumed that one of the subwords in the operand has a negative sign and that the other subword has a positive sign.

Two cases are illustrated in Fig. 14-6. In case 1, it is assumed that overflow indicators Z_2 and Z_4 were left in a ZERO state by the previous instruction. Case 2 assumes that the same overflow indicators were left in the ONE state by the previous instruction, i.e., that an overflow occurred in these instructions.

The following events take place in the SAB instruction. The sign quarter of the left subword in D is complemented. This occurs because the original sign bit of the operand, now stored in Y_4, is positive (ZERO). The sign quarter of the right subword in D is <u>not</u> complemented, since the operand is already negative (ONE). The operand in D_4 specifies the number of shifts that will occur in the data in the associated AB subword A_4 - A_3 - B_4 - B_3. The fact that Y_4 is ZERO means that the data will be shifted to the left. Similarly the operand in D_2 specifies the number of shifts that will occur in the data in AB subword A_2 - A_1 - B_2 - B_1. The fact that Y_2 is ONE means that the data will be shifted to the right.

First consider Case 1. Here the overflow bits in Z are both ZERO. The logic of the SCale instruction requires that the sign bit be left unchanged, therefore no shift into the sign bit occurs. Consider now the left subword. If the sign bit is ZERO, shifting should fill up the right end of the subword with ZEROS. If the sign bit is ONE, shifting should fill up the right end of the subword with ONES. Shifting the sign bit ($A_{4.9}$) into the right end ($B_{3.1}$) accomplishes just this result.

In the case of the right subword, shifting right should fill up the left end of the subword with ZEROS, if the sign bit is ZERO and ONES if the sign bit is 1. Shifting the sign bit ($A_{2.9}$) into the left end ($A_{2.8}$) accomplishes just this result.

Now consider Case 2. Here the overflow indicators Z_4 and Z_2 have both been left set to ONE by the previous instruction, i.e., an overflow has occurred. The overflow has caused an error in the sign, therefore the sign must be complemented before the data is shifted. The mechanics of the instruction are then the same as in Case 1. The ONES in Z_4 and Z_2 are cleared to ZERO by the SAB instruction, since the overflow is taken care of by the instruction.

Fig. 14-7 shows an example of a CYcle AB instruction (CAB) in which the same operand, data, fracture and overflow conditions are used as were used in the SAB example. The example illustrates the basic differences between the two types of instructions.

In the case of CAB, the entire subword is shifted in a closed ring. The sign bits are given no special treatment. In Case 2, in which the overflow indicators have been set to ONE by a previous instruction, the CAB instruction does not affect and is not affected by the state of the Z flip-flops.

In both SCale and CYcle instruction ASK performs no useful function during the execution of the instructions. It simply is indexed once each time a shift occurs. The number of shifts which take place are determined by the D counter.

14-2.6.2 NOA AND NAB. These instructions take the data left in A or AB and multiply it by that positive or negative power of 2 required to make the value of the data lie between 1/2 and 1. The sign quarter of D counts the number of shifts to the left or right required to do this. Effectively, the number of shifts to the left required is subtracted from the sign quarter of the operand brought from memory and placed in D.

In this instruction ASK prevents unlimited shifting from occurring when A or AB contains all ZEROS or all ONES.

Overflows are handled exactly as they were in the SAB instruction previously described.

14-2.6.3 TLY. This instruction is unique in that the operand from memory is placed in the A register. Neither the B or C register is used in this instruction.

In this instruction the data placed in A is completely rotated once and left in its original position. The ASK counter is used to count the number of shifts required to completely "cycle" the subwords. The D counter "tallies" or counts the number of ONES that are contained in the data. The number of ONES is added to the contents of the sign quarter in D left from the previous instruction. The TLY instruction has no effect on the overflow indicators.

Fig. 14-8 summarizes the use of the ASK and D counters for ASK type instruction.

Fig. 14-9 summarizes the function of the Arithmetic Element registers and the Y and Z flip-flops for all the AK instructions.

14-2.7 OPR^{AE} ("AOP") VS $\overline{OPR^{AE}}$ ARITHMETIC ELEMENT INSTRUCTIONS. All of the basic AK instructions discussed so far can be specified by the AOP instruction. This instruction has several characteristics:

1) The Y bits of the N register are used to specify the instruction and its configuration. The Y bits do this by setting the state of the $AKIR_{OP}$ and $AKIR_{CF}$ registers directly with no intervening decoding.

2) In this instruction there is no memory operand or operand cycle. The data in what would ordinarily be the Arithmetic Element operand register is the data left there from a previous instruction.

3) Because the part of an Arithmetic Element instruction in which an operand is taken from memory, configured in the Exchange Element and then loaded into an Arithmetic Element register is absent in the AOP instruction, the effects of configuration specification are different.

Fig. 14-10 shows a comparison of an AOP and $\overline{AOP}$ instruction.

The $\overline{AOP}$ instruction is a CYA in which only quarter 2 of an f_1 (36) fracture is specified active. Similarly, the AOP instruction is used to specify a cycle A in which quarter 2 of an f_1 fracture is active.

Consider first the CYA instruction. The sign bit in quarter 2 of the operand from memory is negative. This means that after the sign is extended in the Exchange Element, quarters 1, 3 and 4 will contain ONES. The $QKIR^{EXT\ ACT}{}_i$ levels

generated in the Program Element and used in the Exchange Element for sign extension are also used by the Arithmetic Element to determine which quarters are active. For example, the fact that $QKIR^{EXT\ ACT}{}_3$, $QKIR^{EXT\ ACT}{}_4$ and $QKIR^{EXT\ ACT}{}_1$ levels were generated to extend the sign into the third, fourth and first quarter means that the a_3, a_4 and a_1 levels will be generated by $AKIR_{CF}$. This will activate the 3rd, 4th and 1st quarters of the Arithmetic Element. Note that because of this all the quarters of the Arithmetic Element are active. Any shift that occurs because of the CYA instruction will cycle a 36 bit word in a closed ring.

In an f_1 (36) fracture the contents of quarter 4 of D (i.e., the sign quarter) will determine how many shifts are to take place. Sign extension has filled quarter 4 with ONES, however. This means that no shifts will occur in the instruction as specified.

Now consider the AOP equivalent of the $\overline{AOP}$ instruction. Assume that the same operand brought from memory in the $\overline{AOP}$ instruction previously discussed, is, in the present instance, left in D from a previous instruction.

Assume that the Y bits of the N register specify an f_1 fracture and quarter 2 is active. This means that $AKIR_{CF}$ will generate an f_1 level and an a_2 level.

Coupling units in the Arithmetic Element connect the various quarters of A and B on the basis of instruction, activity and fracture. In the present case, the fact that f_1 and a_1 in a cycle A to the right instruction (assume that the sign bit ($Y_4 = D_{4.9}$) is a ONE) are specified means that $A_{3.1}$ will be coupled to $A_{2.9}$. However, no shifting will occur into A_1, A_3, and A_4. If $A_{3.1}$ contains a ZERO and D_4 specifies more than 9 shifts, A_2 will fill up with ZEROS.

This example suggests the main differences between AOP and $\overline{AOP}$ instructions.

14-3 ARITHMETIC ELEMENT REGISTER OPERATIONS

This section will discuss the various types of register transfers that can occur in the Arithmetic Element. The basic transfers, which are finite in number, are used to execute all the Arithmetic Element instructions.

14-3.1 STANDARD TRANSFERS. Fig. 4-11 shows the standard register operations common to A, B, C and D. These operations are:

$\xrightarrow{0}$ A (B, C and D and Y)

$\xrightarrow{C}$ A (B, C and D)

E $\xrightarrow{1}$ A (B, C and D and Y)

The execution of certain instructions requires that the contents of A be jammed into B (e.g., in the MULtiply instruction), or that the contents of A and B be interchanged in a jam transfer (e.g., in the DIVide instruction). Fig. 14-12 shows these jam transfers.

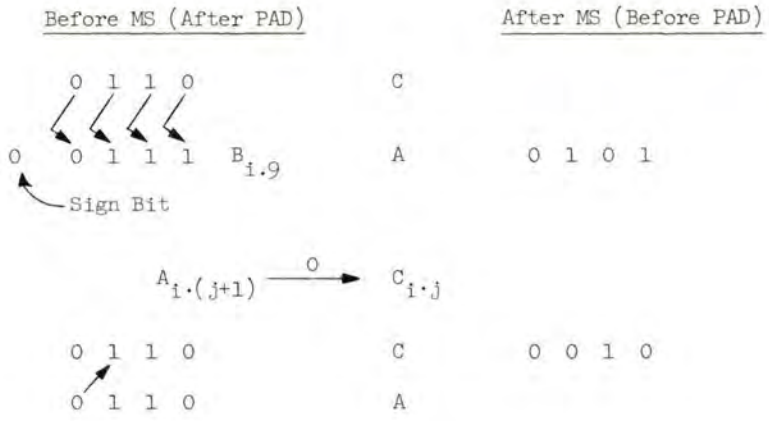
14-3.2 SPECIAL LOGICAL TRANSFERS. Fig. 14-13 and 14 show two special logical transfer circuits in the Arithmetic Element.

The transfers shown in Fig. 14-13 are initiated by a "partial add" pulse ($\xrightarrow{\text{PAD}}$). This pulse complements $A_{i.j}$ if the corresponding $D_{i.j}$ bit is a ONE. This is the "exclusive OR" transfer used to perform a partial add. At the same time, the PAD pulse sets $C_{i.j}$ to ONE, if the corresponding A and C bits are ONES. This is a carry operation based on accumulating in C the logical product of A and D. Note that if $C_{i.j}$ were in the ONE state when a ONE was "carried" into it during a partial addition, the logic of the arithmetic would break down. Later we shall see that, except during DSA, $C_{i.j}$ is <u>always</u> in the ZERO state when a ONE is carried into it due to the over-all instruction logic.

The transfers shown in Fig. 14-14 are initiated by a "Multiply Step" pulse. The Multiply Step operation really does several things at the same time. Basically the operation shifts the content of A and performs a partial addition of the contents of C and A.

The example below illustrates the main features of the Multiply Step operation:

$$A_{i\,.\,(j+1)} \oplus C_{i.j} \xrightarrow{j} A_{i.j}$$



Since both these transfers occur at the same time, the net effect is as follows:

$$A_{i\cdot(j+1)} \oplus C_{i\cdot j} \longrightarrow A_{i\cdot j}$$

$$A_{i\cdot(j+1)} \xrightarrow{0} C_{i\cdot j}$$

The following truth table shows all the possible effects of the Multiply Step operation on the A and C register:

Before MS		After MS	
$A_{i \cdot (j+1)}$	$C_{i \cdot j}$	$A_{i \cdot j}$	$C_{i \cdot j}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Note that $A_{i \cdot j}$ and $C_{i \cdot j}$ are never both left in the ONE state by the Multiply Step operation.

From the arithmetic point of view, the partial addition logic adds the contents of A and D bitwise, leaving the carries in the C register. No inter-bit logic occurs in this partial addition. The Multiply Step operation performs a partial addition between the content of the C register (i.e., the carries left in the C register) and the content of the A register. The carries from this partial addition are placed in the C register and the content of the A register are shifted to the right.

14-3.3 D REGISTER COUNTER. Fig. 14-15 shows the operation of the D register counter. As pointed out earlier, the D register is always preset to a negative number and then counts up to a negative zero, i.e., all ONES. The counter logic says that $D_{i \cdot j}$ will not be complemented unless all the bits to the right of $D_{i \cdot j}$ (i.e., $D_{i \cdot 1}$ the rough $D_{i \cdot (j-1)}$) are ONES.

14-3.4 SHIFTING OPERATION. Fig. 14-16 shows the circuitry arrangement for shifting left and right. The A register shift circuits are shown in Fig. 14-16 (the B register shift circuits are similar in arrangement).

The shift circuits must have sufficient flexibility to accommodate all the possible fractures and instructions. To provide this flexibility, the quarters are designed with "shift coupling units" at the ends. The shift operation involves a bit-wise jam transfer.

Note that the shift right circuitry has a coupling unit at the left end. This unit determines what bit (if any) will have its content shifted into $A_{i \cdot 9}$ when the $\xrightarrow{\text{SHR}}$A pulse is fired. There are eight possible transfer paths into $A_{i \cdot 9}$: $A_{K \cdot 1} \xrightarrow{-j-} A_{i \cdot 9}$ (K = 1, 2, 3, 4) and (in AB type coupling) $B_{K \cdot 1} \xrightarrow{-j-} A_{i \cdot 9}$ (K = 1, 2, 3, 4). Which of the eight possibilities used is determined by the fracture and the instruction. The coupling unit contains the necessary logical circuitry for making the decision.

The shift left circuitry is similar to the shift right circuitry with one slight variation. In some instructions, e.g., SCA, and fractures, $A_{i \cdot 9}$ is the sign bit. For these instructions the instruction logic may not shift $A_{i \cdot 8}$ into $A_{i \cdot 9}$. For this reason, the shift left circuitry has two Shift Left Coupling Units: one at the right end of the quarter and the other between the $A_{i \cdot 8}$ and $A_{i \cdot 9}$ bits.

14-3.5 CARRYING OPERATION. Fig. 14-17 shows a logical block diagram and transistor block schematic of the A register carry scheme.

First consider the logic of the carry scheme. The right end of the carry out in the quarter is driven by a Carry Coupling Unit. This coupling unit has a similar function to the shift coupling unit described above. It determines what data will be carried into (or through) the quarter. The carry inputs are the $D_{i \cdot 9}$ states of each quarter and the "carry outs" (CYO_i) of other appropriate quarters. The selection logic is based on the instruction (DIV or $\overline{DIV}$) and the fracture.

The logic can be broken down into two parts: the bitwise carry logic within a quarter, and the logic involved in carrying between quarters.

Consider a typical stage for $i \neq 1$. The carry into this stage ($CYI_{i \cdot j}$) is identically the same as the carry out of the previous stage ($CYO_{i \cdot (j-1)}$).

This carry circuit is used to propagate a "complete carry" through the A register. For example, in MULtiplication a number of partial adds and multiply steps is followed by a final complete carry. At any one stage in the execution of the MULtiplication, the A register contains the current accumulation of partial additions and the C register contains the current accumulation of partial carries. In the complete carry, the carry bits in C must be correctly brought down into the A register. The $\boxed{\text{CRY}}\!\!\rightarrow$ pulse does this.

If a carry level into the $i \cdot j$ stage ($CYI_{i \cdot j}$) is present, it will complement $A_{i \cdot j}$ when the $\boxed{\text{CRY}}\!\!\rightarrow$ A pulse is fired.

A carry out of the $i \cdot j$ stage will occur whenever $C^1_{i \cdot j}$, or $A^1_{i \cdot j}$ and a carry in ($CYI_{i \cdot j}$), occurs.

A carry into a quarter (CYI_i) will occur whenever the carry coupling logic is satisfied. The i.1 stage will then treat CYI_i as a normal $CYI_{i \cdot j}$ is treated.

A carry out of the quarter (CYO_i) will occur if either a carry into the quarter occurs (CYI_i) and the quarter contains all ONES (A^1_i), or if a carry out of the i.9 stage occurs. Note that $CYO_{i \cdot 9}$ will not be generated unless the quarter does not contain all ONES ($\overline{A^1_i}$). This logic allows the carry to bypass the quarter if it is already loaded with ONES. This saves the time required to propagate the carry through the quarter.

14-4 ARITHMETIC ELEMENT LEVEL LOGIC

This section will discuss the interpretation of the control information found in the $AKIR_{OP}$ and $AKIR_{CF}$ registers. It will also discuss in detail some of the special level logic nets found in the Arithmetic Element. For example, the logic details of the shift and carry circuits will be examined.

14-4.1 $AKIR_{OP}$ AND $AKIR_{CF}$ REGISTERS. The Arithmetic Element receives instruction control commands for AK type instructions from the $AKIR_{OP}$ and $AKIR_{CF}$ registers. These registers are actually located in the Program Element. Chapter 12 describes how the $AKIR_{OP}$ and $AKIR_{CF}$ registers are set up. This chapter discusses the decoding of these registers. Note that $\overline{AK}$ type instructions which use the Arithmetic Element are controlled by $QKIR_{OP}$ and $QKIR_{CF}$. These two registers are also discussed in Chapter 12.

14-4.1.1 $AKIR_{OP}$ DECODING. The $AKIR_{OP}$ register is decoded into $AKIR_{OP}^{XX}$ levels by 1st level decoders. Fig. 14-18 shows the names of the decoded lines.

Fig. 14-19 shows how OP decoders in turn combine the outputs of the 1st level decoders to generate OP code lines. For example, $AKIR^{DIV}$ is generated by a net that ANDs $AKIR^{7X}$ and $AKIR^{X7}$. Note that not all the Arithmetic Element instructions are decoded in this way, e.g., SUB (77) is decoded, but ADD (67) is not.

Still another set of levels is generated in the $AKIR_{OP}$ decoding process by class decoder nets. These class levels group the Arithmetic Element instructions by common characteristics. For example, one level can be used to indicate a class of instructions in which shifting takes place.

Fig. 14-20 tabulates the logic used to generate the class levels. The significance of these levels will become apparent when the logic which uses them is discussed. For reference purposes, a brief description of each class level is given below:

$AKIR^{SH}$ - is generated when any one of the CYcle or SCale instructions is specified. These instructions shift data in A, B or AB as specified by the sign quarters of the operand in D.

$AKIR^{SHA}$ AND $AKIR^{SHB}$ - are both subclasses of $AKIR^{SH}$. The necessary condition for their generation is that shifting occur in register A or B, respectively. Note that if either CAB or SAB is specified, both $AKIR^{SH}$, $AKIR^{SHA}$ and $AKIR^{SHB}$ are generated.

$AKIR^{CY}$ - implies an instruction in which data is cycled in a closed ring. The form of the closed ring is determined by the specific instruction and the fracture specified. The instructions which cycle data are the CYcle instructions and the TLY and DIV instructions.

$AKIR^{AB}$ - implies an instruction in which A and B are coupled. In certain instructions the subwords of A are extended by joining them to the corresponding subwords in B. In this way an AB subword is formed. This occurs in the CAB, NAB and SAB instructions, as well as in the DIV and MUL instructions.

$AKIR^{A+B}$ $(= \overline{AKIR^{AB}})$ - self explanatory.

$AKIR^{CY \cdot AB}$ - is an example of class levels being combined to form a new class level. This level is generated by those instructions which cycle data from A into B (or B into A) in a closed ring. The instructions that do this are CAB and DIV.

$AKIR^{CY \cdot (A+B)}$ - is generated by the CYA, CYB and TLY instructions. In these instructions data is cycled in register A or in register B, but not in AB.

$AKIR^{2N}$ - controls allowable shifting. Earlier in the chapter (Fig. 14-8), we saw that ASK is used in the CYcle, SCale, NOrmalize and TLY instruction to prevent excess shifting from occurring. In the case of CAB, NAB and SAB, ASK limits the shifting to (approximately) twice the subword length specified by the fracture. $AKIR^{2N}$ is generated in these instructions to indicate the double length shift allowed.

$AKIR^{N}$ $(= \overline{AKIR^{2N}})$ - self explanatory.

$AKIR^{NOR}$ - is generated by the two NOrmalize instructions, i.e., NOA and NAB.

$AKIR^{ADD}$ - is generated by either the ADD or SUB instruction. Note that when SUB is specified, both $AKIR^{SUB}$ and $AKIR^{ADD}$ are generated, but that when ADD is specified only $AKIR^{ADD}$ is generated.

$AKIR^{OCSAL}$ - is used in the OCSAL alarm logic. The $AKIR^{OCSAL}$ level is generated when $AKIR_{OP}$ specifies an undefined Arithmetic Element instruction. All the Arithmetic Element instructions are in the 60's and 70's, but 63 and 73 are not defined. Note that this level includes $AK^{1}_{\alpha,1}$ as a factor.

$AKIR^{AOP}$ - is generated whenever an undefined AOP instruction is specified. Note that currently the defined AOP instructions are limited to the Arithmetic Element instructions, hence the same logic that generates $AKIR^{OCSAL}$ also generates $AKIR^{AOP}$.

14-4.1.2 $AKIR_{CF}$ DECODING. The $AKIR_{CF}$ register is decoded to generate fracture (f) and activity (a) levels. Bits $AKIR_{CF_{7-4}}$ determine the activity, and bits $AKIR_{CF_{9-8}}$ determine the fracture. The table on Fig. 14-21 shows the $AKIR_{CF}$ decoding. Note that a quarter is activated by an a_i^1 level and that the a_i^1 level is in turn generated by an associated $AKIR^0_{CF_J}$ level.

Fracture decoders use the a's and f's as inputs to generate Roman numeral levels. The unsubscripted Roman numerals (RN) indicate the sign quarter of a subword which contains at least one active quarter. For example, II indicates that quarter 2 is a sign quarter and that it is part of a subword which is at least partially active.

The subscripted Roman numerals (RN_i) indicate that the i-th quarter is active under a certain special condition. This condition is that the i-th quarter is part of a subword whose sign quarter is given by the Roman numeral. For example, IV_1 indicates that quarter 1 is active and is part of a subword which has quarter 4 as its sign quarter.

A pictorial representation of these Roman numeral levels is shown in Fig. 14-21. The conditions for generating these levels are:

Roman Numeral I. The only occasion when quarter 1 is the leftmost quarter of a subword which contains at least one active quarter is when quarter 1 itself is the subword. Thus, $I = I_1$. (Note this same argument makes $III = III_3$.) I is generated in both the f_3 (27,9) and f_4 (9,9,9,9) fractures when quarter 1 is active (a_1^1).

Roman Numeral II. A threefold possibility exists: either the first quarter is active and there is an f_2 (18,18) fracture, or the second quarter is active and there is an f_2 (18,18) or f_4 (9,9,9,9) fracture.

Roman Numeral III and IV. The logic here is similar to that described for I and II. Note that $IV_4 = a_4^1$.

14-4.2 ALL ZEROS (A_i^0), ALL ONES (A_i^1) LEVEL LOGIC. The logic generating the A_i^0 and A_i^1 levels is shown in Fig. 14-22. These levels indicate that the quarters are filled with all ZEROS (or all ONES). A_i^0 and A_i^1 are used in the carry and jump logic.

14-4.3 SIGMA (σ) LEVEL LOGIC. $A_{i.9} = A_{i.8}$ implies sigma (σ). Fig. 14-23 shows the explicit logic generating the sigma (σ) level. This level is used in the NOrmalizing instructions to indicate that $A_{i.9} = A_{i.8}$ and that consequently the normalization process is not yet finished.

14-4.4 SHIFT COUPLING UNITS. These units are logic nets which determine how the quarters of A and B are coupled together during shift instructions. In (A + B) type instructions the quarters of A are always coupled to other quarters of A, and similarly the quarters of B are always coupled to other quarters of B. In AB type instructions cross coupling between the registers can occur.

Fig. 14-24 shows the logic that relates the inputs and outputs of the Shift Coupling Units shown on Fig. 14-16. Fig. 14-25 shows the corresponding shift coupling logic for register B.

For the moment consider Fig. 14-24. Note that all of the shift right logic involves inter-quarter shifting. If the instruction is an (A+B) type, $A_{i.1's}$ will be shifted into $A_{i.9's}$. The specific bits shifted will depend on the fracture. If the instruction is an AB type, $B_{i.1's}$ will be shifted into $A_{i.9's}$. Again the specific bits shifted will depend on the fracture. Regardless of which type instruction is involved, i.e., (A+B) or (AB), certain quarters of A will be coupled to other quarters of A by the last term in the shift coupling logic.

The logical format for shifting in B is similar and shown in Fig. 14-25.

Fig. 14-26 shows a specific example of shift right coupling for both an (AB) type instruction and an (A+B) type instruction that have an f_2 fracture. Note that in both instructions the same logic couples A_4 to A_3 and A_2 to A_1. In the (AB) type instruction, the sign quarters of the subwords are A_4 and A_2, respectively. No shift into $A_{4.9}$ or $A_{2.9}$ will occur unless the instruction is a shifting instruction which ignores the sign bit, i.e., CY · AB = CAB + DIV. Since B_4 and B_2 are <u>not</u> sign quarters, it is only necessary that the instruction be an AB type, i.e., (CAB + NAB + SAB) + (MUL + DIV) in order that $A_{3.1}$ be coupled to $B_{4.9}$ and $A_{1.1}$ be coupled to $B_{2.9}$.

Note in Figs. 14-24 and 14-25 that the fractures in the column "independent of AB/(A+B)" are always the complement of the fractures in the "AB and A+B" columns.

The inter-quarter shift left logic shown on Fig. 14-24 and 14-25 is similar in format to the shift right logic with one important exception. Since in a shift left instruction we are always shifting information into i.1, there are no sign bit considerations. This means that the shift is independent of whether the instruction is or is not of the CY type and depends only on whether it is of the (AB) or (A+B) type.

In the case of shift left, coupling units are also required between the i.8 and i.9 bits, i.e., intra-register coupling exists. If a CY type instruction is involved, no sign bit consideration is involved and i.8 is always coupled to i.9. If the fracture is f_1 (36) or f_2 (18,18), 1.8 will always be coupled to 1.9, since in either fracture 1.9 is not the sign bit. Similar logic is involved for coupling into 2.9, 3.9 and 4.9 during a shift left. Note that 4.9 is always a sign bit regardless of the fracture.

If an AB instruction is involved, $B_{i.9}$ will never be a sign quarter, therefore $B_{i.8}$ is always shifted into $B_{i.9}$ on a shift left instruction.

In passing, it might be mentioned that the speed of the coupling logic nets themselves becomes critical if the shifting rate approaches 5 megacycles.

14-4.5 CARRY COUPLING UNITS. Fig. 14-27 shows the logic that relates the inputs and outputs of the Carry Coupling Units shown on Fig. 14-17. Note that CYI_i represents a carry into the i-th quarter, while CYO_i represents a carry out of the i-th quarter.

For $\overline{DIV}$ type instructions, the carries are propagated in a ring whose constituents are determined by the fracture. For example, suppose an f_2 fracture is specified for a $\overline{DIV}$ type instruction. The quarter wise carry picture would then look as shown in Fig. 14-28(a).

In the DIVide instruction a 2's complement-like arithmetic is used. The logic of this arithmetic requires that a 1 be added if the sign bit in D is negative. This is shown in Fig. 14-28(b). The inter-quarter carry logic is the same for both DIV and $\overline{DIV}$ instructions. Only the end around carry differs for these two cases.

Note that the carry logic format shown on Fig. 14-27 is very similar to the shift logic format shown on Fig. 14-24 and 14-25.

14-4.6 AEJ LEVEL LOGIC. AEJ is a level which indicates whether a jump is to be made, based on the contents of the Arithmetic Element. It can be generated during a JPA, JNA or JOV instruction. The function of AEJ for each of these instructions is as follows:

JPA - If some active subword contains a positive non-zero (arithmetically) number, AEJ will be generated and cause the output of the XA (X Adder) to be copied into P.

JNA - If some active subword contains a negative non-zero (arithmetically) number, AEJ will be generated and cause the output of the XA to be copied into P.

JOV - If some active subword has a Z flip-flop in the sign digit position set to ONE, AEJ will be generated and cause the output of the XA (X Adder) to be copied into P.

Fig. 14-29 shows the logic generating the AEJ level. Note that the presence of any one of the terms is sufficient to generate AEJ.

As an illustrative example, the conditions for causing a jump based on the contents of A_2 will be discussed. If a JPA is executed, $PKIR^{JPA}$ will be generated. $QKIR^{EXT\ ACT}{}_2$ indicates that the quarter 2 of the Arithmetic Element is active, while $A^0_{2.9}$ indicates that the sign bit is positive. Note that $A_{2.9}$ is the sign bit in an f_2 (18,18) or f_4 (9,9,9,9) fracture. The additional factor in the term on Fig. 14-29 insures that the active subword contains a non-zero number. Note that in an f_2 (18,18) fracture it is sufficient that quarter one not contain all ONES or all ZEROS , i.e., $A^{\overline{1}}_1 \cdot A^{\overline{0}}_1 \cdot QKIR^{f_2}$; while it is sufficient in an f_2 (18,18) or f_4 (9,9,9,9) fracture that quarter 2 not contain all ONES or all ZEROS, i.e., $A^{\overline{1}}_2 \cdot A^{\overline{0}}_2 \cdot QKIR^{f_2+f_4}$.

The logic for JNA is the same, except that $A_{2.9}$ must be in the ONE state.

The JOV logic is also similar. Z_1 is in a sign quarter in an f_3 (27,9) or f_4 (9,9,9,9) fracture. Z_2 is in a sign quarter in an f_2 (18,18) or f_4 (9,9,9,9) fracture, etc. Note that Z_4 is always in a sign quarter regardless of the fracture.

14-4.7 $\llcorner^0\!\!\blacklozenge$ AEP LEVEL LOGIC. $\llcorner^0\!\!\blacklozenge$ AEP is used to clear to ZERO the AEP (Arithmetic Element Predict) interlock flip-flop. (See Chapter 10.) AEP^0 indicates that the Arithmetic Element will soon be available for use by another Arithmetic Element instruction. For each Arithmetic Element instruction and configuration an event, before the completion of the AK cycle, is chosen to generate the $\llcorner^0\!\!\blacklozenge$ AEP level. The AEP^0 level effectively predicts the maximum time required for the balance of the AK cycle. This maximum of the maximum times is 2.8 microseconds and occurs during an ADD instruction with an $\overline{f_4}$ fracture. During other instructions and with other configuration the time can be less.

Fig. 14-30 gives the anticipatory logic for generating the $\text{AEP}^{0}_{\text{L}}$ level. The first term in this logic is concerned with the NOrmalize instructions. In these instructions the data in the subword is shifted until the value of the data lies between 1/2 and 1, i.e., the left-most bits in the sign quarter must be 01 or 10. If the sign quarter contains neither all ZEROS nor all ONES, the greatest number of shifts that can occur before the data is normalized is eight. For example, suppose the subword contains the following data:

SIGN QUARTER

1 1 1 1 1 1 1 1 0	X X X X X X X X X

If a NOrmalize instruction is executed, after seven shifts the data will be normalized, i.e., the sign quarter will look as follows:

1 0 X X X X X X X

The example just given was a "worst condition" case. The data to be normalized might have been:

1 1 0 X X X X X X

In this case only one shift is required to normalize the data.

Note that if the sign quarter is quarter 1, then a Roman numeral I will be generated, and it is necessary to know only that this quarter does not contain all ZEROS or all ONES to know that a maximum of seven shifts will occur before the data is normalized.

In the case where the subword contains the following data,

SIGN QUARTER

1 1 1 1 1 1 1 1 1	1 1 1 1 1 0 X X X

six shifts occur before the $\text{AEP}^{0}_{\text{L}}$ level is generated. AEP^{0} then indicates that a maximum of seven additional shifts will follow before the data is fully normalized.

The second term in the $\text{AEP}^{0}_{\text{L}}$ logic is concerned with the SH instructions (i.e., CYcle and SCale). Since the D counter always counts up to zero from some negative value in these instructions, it is always possible to know how long the count will take to complete from an arbitrary but predetermined counter state. LAD is used as the reference event in the count. LAD_i anticipates how long it will take to complete the count in the i-th quarter of D. If a Roman numeral II is generated, we are interested in the LAD_2 level, etc. The logic here is very similar to that for the NOR instructions.

The third term in the $\boxed{0}\longrightarrow$ AEP logic is associated with the ADD, SUB and DSA instructions. In this case, the balance of the AK instruction time depends only on the reference AK state.

In the DIVide instruction, it is necessary to know both the state of the AK counter and the ASK counter to predict the balance of the AK instruction time.

The fifth term is concerned with the case where the ASK counter overrides the D counter. This is the case where some active subword contains all ZEROS or all ONES. In the NOrmalize instructions, after a certain number of shift counts have occurred, AEP^0 will indicate that there are <u>at least</u> no more than 6 additional shifts to occur. In the TLY instruction, after a certain number of shifts counts have occurred, AEP^0 will indicate that there are <u>exactly</u> 6 additional shifts to occur.

The MUL term logic is similar to the DIV term logic.

Finally, in the case of an undefined AOP instruction, a finite time exists between the generation of the $AKIR^{AOP}$ level and the completion of the AK cycle. Note that $AKIR^{AOP}$ includes an AK time level, i.e.,

$$AKIR^{AOP} = AK^{1}_{\alpha.1}\ ((\overline{AKIR^{6X}_{OP}}\ \cdot\ \overline{AKIR^{7X}_{OP}})\ +\ AKIR^{X3})$$

ARITHMETIC ELEMENT REGISTER DRIVER LOGIC

The remainder of the chapter will discuss the specific register driver pulses found in the Arithmetic Element and the logic generating these pulses.

All of these register driver pulses are tabulated on Fig. 14-31. This figure shows the various logic and counter time levels that are found in the register driver logic. For example, the sigma (σ) levels are found in both the D counter register driver logic and in the shift register driver logic.

Fig. 14-32 tabulates all the pulse gate logic. In some cases the gating logic is quite simple, in other cases it is quite complex, e.g., the gating logic for the Z flip-flops. Once the logic generating the register driver pulse is known and the pulse gating logic is known a comprehensive picture of the register operation can be established.

14-5.1 $\boxed{\overline{D_i}}\ +\ 1 \longrightarrow D_i$ RD LOGIC. (See Fig. 14-33). Counting will occur in the quarter of D corresponding to the sign quarter of the subword in A. The Roman numerals indicate the sign quarters. The FD_i levels indicate when the D counter has counted up to zero. The count pulses are then inhibited by the FD_i levels.

In the SH type instructions (CYcle and SCale), counting is initiated at $AK^1_{\alpha.3}$. Overflow control occurs at $AK^1_{\alpha.3}$ and then AK counts on to $AK^1_{\alpha.4}$. The major portion of the counting in D then occurs in $AK^1_{\alpha.4}$.

In the TLY instruction, D counts the ONES that appear in the sign bit of A at $AK^1_{\alpha.2}$ as the subword is cycled (rotated).

During the NOrmalize instructions, D counts, i.e., continues to normalize, as long as $A_{i.9} = A_{i.8}$. This equality is indicated by the σ (sigma) levels.

Note that ASK can override the D counters by forcing AK into a new time state even though the D counter register driver logic is not satisfied. D then stops counting, even though the FD_i levels are not generated.

14-5.2 A REGISTER SHIFT RD LOGIC. (See Fig. 14-34). During a TLY instruction, shifting to the right occurs in all the active quarters of A. The shift decision is independent of any fracture considerations.

During a NOrmalize type instruction, the active quarters of a subword are shifted to the right if the Z flip-flop in the sign quarter of the subword indicates an overflow. Note that there are three possibilities that can cause a shift right to occur in A.

$Z^1_1 \cdot I$ Quarter 1 is active and is also the sign quarter. $Z^1_1 \cdot I$ indicates that quarter 1 is part of a subword in which an overflow condition exists.

$Z^1_2 \cdot II_1$ Quarter 1 is active and the sign quarter is quarter 2. $Z^1_2 \cdot II_1$ indicates that quarter 1 is part of a subword in which an overflow condition exists.

$Z^1_4 \cdot IV_1$ Quarter 1 is active and the sign quarter is in quarter 4. $Z^1_4 \cdot IV_1$ indicates that quarter 1 is part of a subword in which an overflow condition exists.

The shift right logic for the other quarters of A during NOrmalize type instructions is similar to that just described. The shift left decision is made if $A_{i.9} = A_{i.8}$ as indicated by the σ (sigma) levels.

Note that if there is to be a shift right it will occur at $AK^1_{\alpha.2}$. At the completion of the shift right, $A_{i.9} \neq A_{i.8}$ (see Fig. 14-43 for the NOR logic that complements $A_{i.9}$ at $AK^1_{\alpha.2}$), therefore the σ (sigma) levels will be absent and no shift left will occur at $AK^1_{\alpha.4}$.

The shift logic for the SHA type instructions is similar to that for the NOR type instructions. The shift right decision is made if the operand sign bit is negative ($Y_i^1 = D_{i.9}^1$), and left if the sign bit is positive ($Y_i^0 = D_{i.9}^0$). Note that $\overline{FD}_1$ is used both for inhibiting the counting in D and for inhibiting the shifting in the SH instruction.

A shift left in the active quarters of A occurs in the DIVide instruction. A shift occurs during each iteration as ASK counts up to zero. During the count up to zero, ASK is in the ONE state. One more shift occurs after ASK reaches the zero state. This shift is taken care of by the ASK_1^0 term.

14-5.3 B REGISTER SHIFT RD LOGIC. (See Fig. 14-35). The B register shift logic for the NAB and SHB instructions is identical to the A register shift logic for the NOR and SHA instructions, respectively. During the MUL instruction, a shift right occurs in all the active quarters of B at $AK_{\alpha.3}^1$. Similarly, during the DIV instruction, a shift left occurs in all the active quarters of B at $AK_{\alpha.9}^1$.

14-5.4 MULTIPLY STEP RD LOGIC. (See Fig. 14-36). The Multiply Step pulses are fired off repeatedly during the MUL instruction. The two pulses involved are:

$\llcorner\overset{\text{MULT STEP}}{\longrightarrow} A_i, C_i$. This pulse is fired off in all the active quarters. Note that this pulse effects only bits i.1 through i.8.

$\llcorner\overset{\text{MULT STEP SIGN}}{\longrightarrow} A_{i.9}, C_{i.9}$. This pulse is fired off only for those i.9 bits which are <u>not</u> sign bits. The logic for the MULT STEP SIGN pulses indicates that the pulse will be fired off for the i.9 bits, only if the sign quarter of the subword is to the left of the quarter in which the i.9 bits are located.

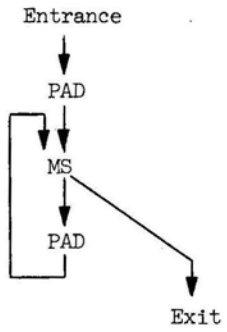
14-5.5 COMPLEMENT C RD LOGIC. (See Fig. 14-37). This logic is not currently used. The complement nets have, however, been partially incorporated in the computer.

14-5.6 CARRY RD LOGIC. (See Fig. 14-38). During a DIV instruction, a $\llcorner\overset{\text{CRY}}{\longrightarrow}$ pulse will be fired off in all the active quarters of A. Note that DIV uses a twos - complement arithmetic. The logic of this arithmetic leaves the carry loop open (see Fig. 14-28(b)).

The $\overline{\text{DIV}}$ instructions involve an "end around carry". The logic involved closes the carry loop (see Fig. 14-28(a)). If the subword in A contains all ONES, the loop tends to exhibit instability in the presence of noise. That is, it can generate a carry level by itself. For this reason, the $\llcorner\overset{\text{CRY}}{\longrightarrow}$ pulse is not fired off in a quarter if the subword containing it holds all ONES. For example, the $\llcorner\overset{\text{CRY}}{\longrightarrow} A_1$ pulse is not fired off unless: (1) either A_1 does not contain all ONES; or (2) A_2 does not contain all ONES and there is an f_1 (36) or f_2 (18,18)

fracture; or (3) A_3 does not contain all ONES and there is an f_1 (36) fracture; or (4) if A_4 does not contain all ONES and there is an f_1 (36) fracture.

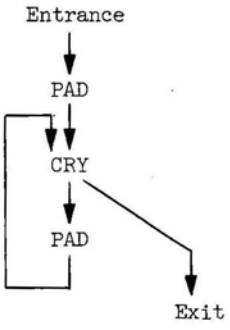
14-5.7 PARTIAL ADD RD LOGIC. (See Fig. 14-39). The execution of the MUL instruction involves the iteration of a "partial add - multiply step" loop. The loop looks somewhat as follows:



The PAD pulse, encountered before the loop is entered, occurs at $AK^1_{\beta.2}$, while the PAD pulses in the loop occur at $AK^1_{\beta.3}$.

PAD $\lceil_1$ on Fig. 14-39 pertains to the MUL instruction. The logic ANDed with PAD $\lceil_1$ says that the i-th PAD pulse is fired off if the rightmost bit in the subword in B is in the ONE state and the i-th quarter is active. For example, a partial add pulse is fired off in quarter 4 if an f_3 (27,9) fracture is involved and the $B_{2.1}$ bit is in the ONE state at ($AK^1_{B.3}$ + $AK^1_{B.2}$).

The execution of the DIV instruction involves the iteration of a "partial add - carry". The loop in this case looks somewhat as follows:



The PAD pulse, encountered before the loop is entered, occurs at $AK^1_{B.3}$; while the PAD pulses in the loop occur at $AK^1_{B.9}$.

PAD $\lceil_2$ pertains to the DIV PAD pulse that is fired off before the loop is entered. Note that this pulse is fired off in all active quarters.

PAD ϕ pertains to the DIV PAD pulses that are fired off in the loop. The early pulses in the loop are conditional on the state of the ASK counter and the fact that the quarter is active. Notice that $RN_i \supset a_1^1$. The last PAD pulse fired off is conditional not on ASK, but on the sign of the subword. This is why the PAD ϕ's must be ANDed with the subscripted Roman numerals and not just the activity levels. As ASK counts, PAD ϕ looks at ASK_7, ASK_1 and $A_{i.9}$. Thus,

ASK COUNTER

.
.
.

1 1 1 1 1 1 0 $\quad ASK_7^1 \supset PAD\ \phi_i$ where i = 1,2,3,4

1 1 1 1 1 1 1 ditto

0 0 0 0 0 0 0 $\quad ASK_1^0 \supset PAD\ \phi_i$ where i = 1,2,3,4

0 0 0 0 0 0 1 $\quad ASK_7 \neq ONE,\ ASK_1 \neq ZERO,\ \therefore PAD\ \phi_i \supset A_{i.9}^0$

PAD $\lceil 2$ also takes care of the single partial add pulse that is fired off in the DSA, ADD and SUB instructions at $AK_{\beta.2}^1$.

14-5.8 A,B,C AND D CLEAR RD LOGIC. (See Fig. 14-40). First consider the $\llcorner^0$ A,B,C and D levels which cause the $\llcorner^0$ A,B,C and D pulses to be fired off. These levels are triggered by QK time levels. If an operand is to be stored in the Arithmetic Element registers of the V_{FF} Memory, the selected register will first be cleared at $QK^{22\alpha}$. The A register is also cleared in the ITA and INS instruction at $QK^{22\alpha}$. If the Arithmetic Element registers are to be loaded, they will first be cleared at $QK^{14\alpha}$.

The active quarters of A will be cleared in a MUL instruction at $AK_{\alpha.2}^1$. This occurs immediately after the contents of A have been transferred into B, i.e., at $AK_{\alpha.1}^1$.

The active quarters of C will be cleared in an ADD or SUB instruction at $AK_{\alpha.1}^1$ just before the PAD pulse is fired at $AK_{\beta.2}^1$. These quarters of C will also be cleared in the MUL and DIV instructions at both $AK_{\alpha.1}^1$ and $AK_{\alpha.8}^1$. The $\llcorner^0 C_i$ at $AK_{\alpha.1}^1$ sets up C for the first PAD pulse. The $\llcorner^0 C_i$ at $AK_{\alpha.8}^1$ leaves C cleared in both the MUL and DIV instruction at the end of the instruction. In the case of the DIV instruction, it also clears C at the same time the carry occurs.

14-5.9 Z PULSE GATE LOGIC AND RD LOGIC. (See Fig. 14-41). See Fig. 14-4 for the function of Z in the various instructions. Z is cleared in the sign quarter at the beginning of the ADD, SUB and MUL instruction. This occurs at $AK^{1}_{\alpha.1}$. Note that in the case of DIV an $\lfloor A_{i.9} \xrightarrow{\text{Sign}} Z_i$ pulse is fired at $AK^{1}_{\alpha.1}$. If the sign bit is positive ($A^{0}_{i.9}$), Z is cleared to ZERO; if the sign bit is negative ($A^{1}_{i.9}$), Z is set to ONE.

In the case of SCA, SAB, NOA and NAB, the instruction leaves Z cleared. The clearing occurs at $AK^{1}_{\alpha.3}$. MULtiplication always leaves Z cleared. In the MUL case, Z is cleared at $AK^{1}_{\alpha.9}$.

In the case of MULtiplication, Z is cleared at $AK^{1}_{\alpha.1}$ and then the sign of A is placed in Z at $AK^{1}_{\alpha.2}$. In the case of DIV, an overflow can occur. This is taken care of by reading the sign of A into Z near the end of the instruction, i.e., at $AK^{1}_{\alpha.11}$.

The rest of the Z logic is used in the ADD and SUB overflow logic (see Fig. 14-5). Z is first cleared at $AK^{1}_{\alpha.1}$. The signs of D and A are compared at $AK^{1}_{\alpha.3}$. If they are the same, Z is set to ONE. The A register contains the addend or subtrahend at $AK^{1}_{\alpha.3}$. At the same time the signs are examined, the PAD pulse is fired off. After the carry is completed, the A register contains the sum or difference. At $AK^{1}_{\alpha.9}$ the sign of D and A are again examined. If they are the same, Z is cleared by the Reset Z logic. If they are different, the ONE left in Z indicates an overflow condition.

14-5.10 A REGISTER COMPLEMENT RD LOGIC. (See Fig. 14-42). The subword will be complemented at the end of the MUL and DIV instructions if $Z \neq Y$ in the sign quarter, i.e., at $AK^{1}_{\alpha.9}$ and $AK^{1}_{\alpha.11}$, respectively.

In the DIV instruction, A is complemented at the beginning of the instruction at $AK^{1}_{\alpha.1}$, if the sign of the subword is positive.

A is also complemented during the INS and ITA instructions as part of the execution logic.

Note that $\lfloor C \longrightarrow A_i$ complements $A_{i.1}$ through $A_{i.9}$.

In addition to complementing the quarters of A, it is possible to complement the $A_{i.9}$ bits individually. In SCale instructions, if an overflow from a previous instruction exists in the sign quarter, the sign bit in A is complemented. If the SCale instruction calls for a shift left (Y^{0}_{1}), the sign bit is complemented at $AK^{1}_{\alpha.2}$. If the SCale instruction calls for a shift right (Y^{1}_{1}), the sign bit is complemented at $AK^{1}_{\alpha.3}$.

Finally, in a NOR instruction, the sign bit of A is complemented if an overflow from a previous instruction exists in the sign quarter. This occurs at $AK^{1}_{\alpha.2}$.

14-5.11 B REGISTER COMPLEMENT RD LOGIC. (See Fig. 14-43). If a negative subword is placed in the B register in a MUL instruction, the subword will be complemented in the B register. This occurs just after the multiplier in the A register has been transferred into the B register, i.e., at $AK^{1}_{\alpha.2}$. Thus, MUL is always executed with a positive multiplier in the B register.

During a MUL instruction, the B register is also complemented at the end of the AK cycle, i.e., at $AK^{1}_{\alpha.9}$, if Z is not equal to Y. This is part of the sign control logic.

In a DIV instruction, the minor half of the dividend, located in the B register, is complemented at the beginning of the instruction, i.e., at $AK^{1}_{\alpha.1}$, if the sign of the major half of the dividend located in the A register is positive. This is the significance of the $\overset{C}{\longrightarrow} B_i \cdot RN_i$ logic. The B register is also complemented at the end of the instruction when it contains the remainder, if the Z flip-flop in the sign quarter of A is in the ZERO state at $AK^{1}_{\alpha.11}$.

The B register is also complemented in the INS instruction at $QK^{10\alpha}$ and $QK^{21\alpha}$ as part of the execution logic of that instruction.

14-5.12 D REGISTER COMPLEMENT RD LOGIC. (See Fig. 14-44). Consider first the complement D logic used in the NOR instructions. Since D counts up to zero, the memory operand in D is always complemented at the beginning of the NOR instruction, i.e., at $AK^{1}_{\alpha.3}$. The D register is then complemented at the end of the normalizing to restore the operand to its original value. The end of the normalizing occurs when $A_{i.8} \neq A_{i.9}$ in the sign quarter, i.e., $\overline{\sigma}_i$, hence the $(\overline{RN} + \overline{\sigma})$ factors. If the data to be normalized should contain all ZEROS or all ONES, $ASK^{0}_{6} \cdot ASK^{0}_{7}$ would indicate the end of the normalizing, i.e., ASK overrides the D counter and the σ condition is ignored.

The logic for complementing the D register in other instructions is basically covered by the ϕ terms. First consider the SH type instructions, i.e., the SCale and CYcle instructions. The sign quarter of D is complemented if the sign of that quarter is positive, i.e., the sign quarter (which is the quarter in which the counting occurs) is always made negative. The logic shown on Fig. 14-44 may look peculiar for these instructions until the following facts are realized: Q_1 and Q_3 are used only in the logic for quarters 1 and 3, respectively; whereas Q_2 and Q_4 are each used in the logic for more than one quarter. Therefore, the SH terms for quarters 1 and 3 can be included in Q_1 and Q_3, respectively; whereas separate terms are needed for quarters 2 and 4.

Now consider the balance of the $\emptyset$ logic. Remember that $AKIR^{ADD}$ covers both the ADD and SUB instructions. In a SUB instruction the active quarters of D are complemented at $AK^{1}_{\alpha.2}$. At $AK^{1}_{\alpha.9}$, the active quarters of the subword in D are complemented in the ADD and SUB instruction, if the sign of the subword at this time does not equal Y, i.e., $Y_i \neq D_{i.9}$. Note that in SUB D is complemented twice; whereas in ADD D is complemented just once.

In the DIV instruction, the data in the D register is always made opposite in sign to the data in the A register before the partial addition occurs, i.e., D is complemented if $D_{i.9} = A_{i.9}$ at $AK^{1}_{\alpha.2}$. Remember that the PAD pulse is fired off both at $AK^{1}_{\beta.3}$ and $AK^{1}_{\beta.9}$. Therefore, D is complemented at $AK^{1}_{\alpha.9}$ for the same reason. Finally, D is complemented at $AK^{1}_{\alpha.11}$ as part of the sign control logic if $Y_1 \neq D_{i.9}$. This makes the sign of D equal to its original value.

In the MUL instruction the D register is complemented at $AK^{1}_{\alpha.2}$ if the subword in D is negative, i.e., if Y^{1}_{1}. The D register is complemented again at $AK^{1}_{\alpha.9}$ if Y^{1}_{1} in order to restore D to its original value.

14-5.13 E ⟶ A, B, C AND D RD CONTROL. (See Fig. 14-45). The only way that data can be placed in the A, B, C and D registers in the Arithmetic Element is via the Exchange Element, more specifically via the E register. This occurs in the following situations:

1) During LD type instructions, when the A, B, C or D registers are specified, the transfer occurs at $QK^{21\alpha}$.

2) In the execution logic of the ITA and UNA instructions, the data found in E at $QK^{23\alpha}$ is transferred into the A register.

3) If a STORE instruction involving the V_{FF} Memory specifies one of the Arithmetic Element registers, data is transferred from E into the register at $QK^{23\alpha}$.

As we saw earlier in the chapter the data that is transferred from $E_{i.9}$ into $D_{i.9}$ is also transferred by the same register driver pulse into Y_i.

14-5.14 A —j⟶ B AND B —j⟶ A RD CONTROL. (See Fig. 14-46). Note that these transfers are of the jam type. They occur in the MUL and DIV instruction under the following circumstance:

1) One of the first things that happens in a MUL instruction is that the data in the active quarters of A (left from a previous instruction) is transferred into the corresponding quarters of B. This occurs at $AK^{1}_{\alpha.1}$.

2) The execution logic of the DIV instruction generates the quotient in the B register and the remainder in the A register. At $AK^{1}_{\alpha.10}$ the active quarters of A and B are interchanged so that A contains the quotient and B contains the remainder.

AE INSTRUCTIONS			
$\overline{AK}$ TYPE	AK TYPE		
	$\overline{ASK}$ TYPE	ASK TYPE	
		$\overline{\text{COUNT IN D TYPE}}$	COUNT IN D TYPE
LDA, -B, -C, ξ -D STA, -B, -C, ξ -D EXA INS ITA UNA	ADD SUB PSA	MUL DIV	SCA, -B, -AB CYA, -B, -AB NOA, -AB TLY

Fig 14-1 AE INSTRUCTIONS CLASSIFIED BY COUNTER ACTIVITY

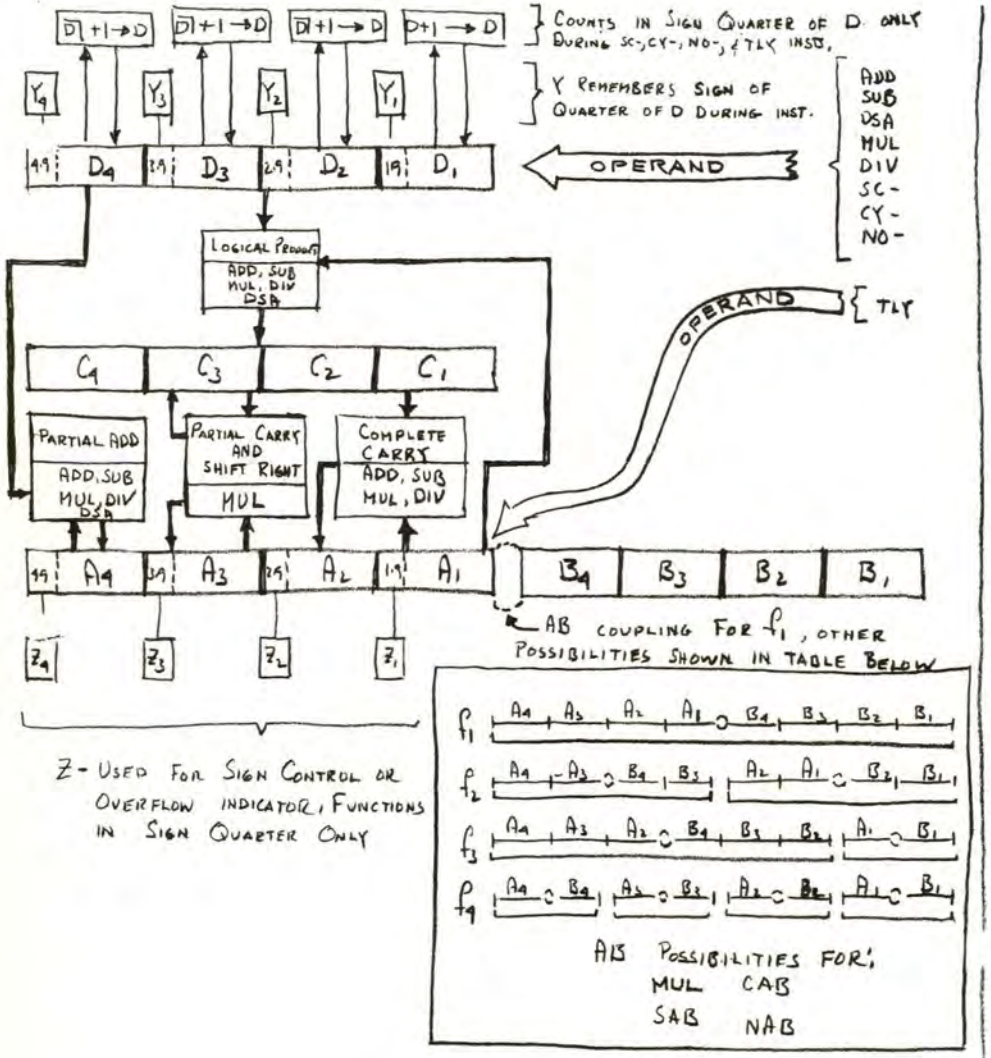


Fig 14-3 Functional Block Diagram of the AE for AK Type Instructions

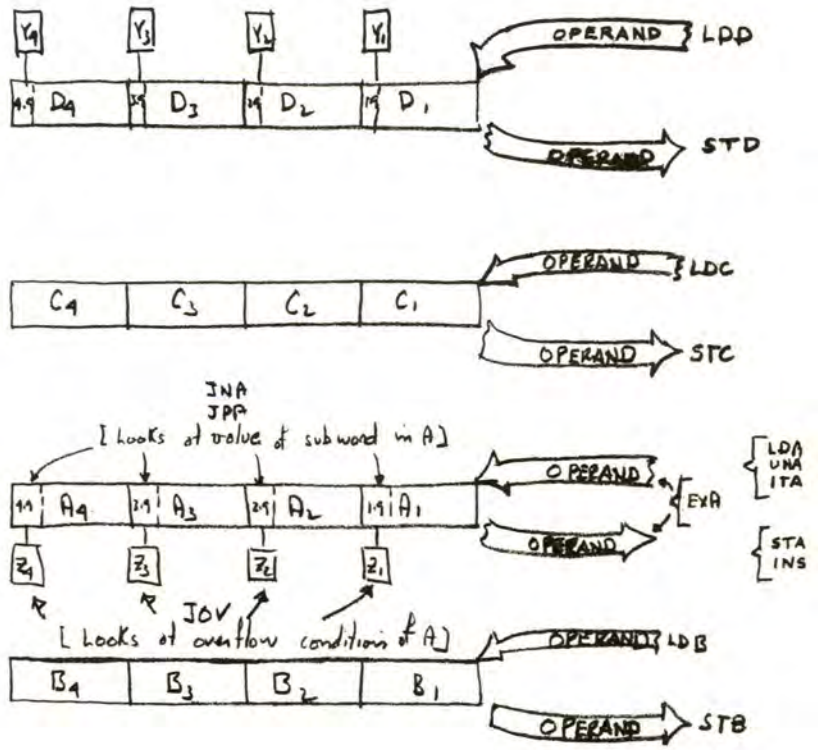


Fig 14-2 Functional Block Diagram of the AE for $\overline{AK}$ Type Instructions

	0 → Z (at beginning)	Z → A	Z Used For Sign Control	Overflow Maybe Generated / Left in Z	Z Left Cleared By Instruction	Z Left Unchanged By Inst
ADD	X			X		
SUB	X			X		
PSA						X
MUL	X		X		X	
DIV	X		X	X		
TLY						X
CY-SCB						X
SCA, SAB		X			X	
NOA, NAB		X			X	

Fig 14-4 Effect of A.K. Type Instructions on Z Flip-Flop

ADDITION						
BASIC OVERFLOW LOGIC				COMPUTER OVERFLOW LOGIC		
D_{SIGN}	A_{SIGN}		Z	$\boxed{0} \rightarrow Z$	$\boxed{PAD} \rightarrow Z$	$\boxed{RESET} \rightarrow Z$
AUGEND	ADDEND	SUM				
0	0	0	0	0	1	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
1	1	0	1	0	1	1
0	0	1	1	0	1	1
1	0	1	0	0	0	0
0	1	1	0	0	0	0
1	1	1	0	0	1	0

Fig 14-5 Addition Overflow Logic

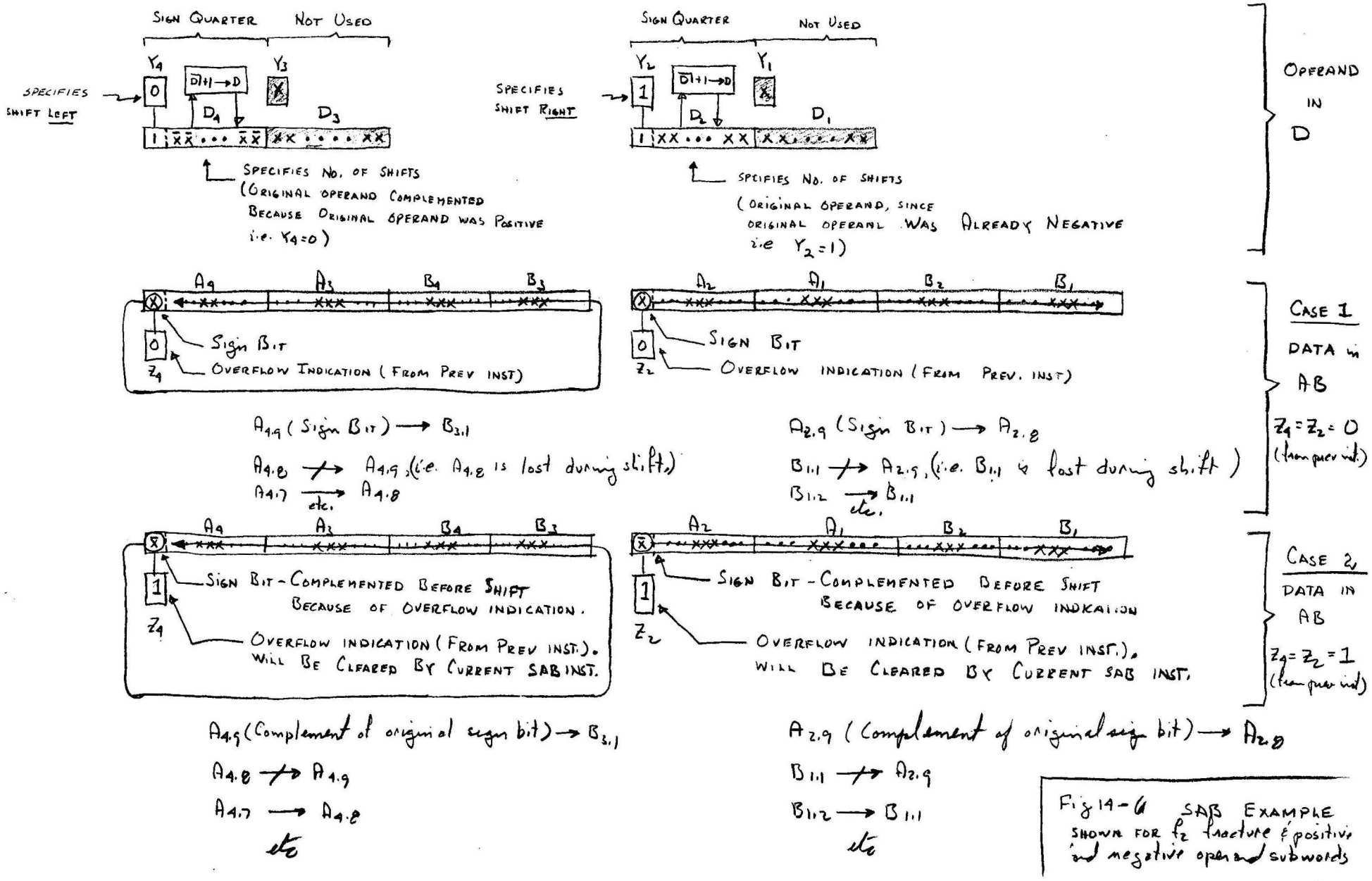
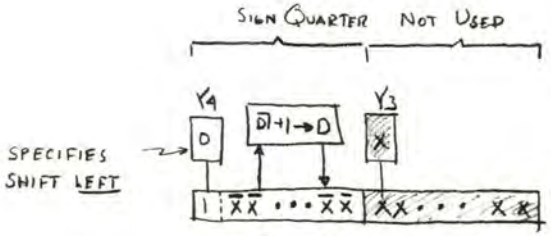
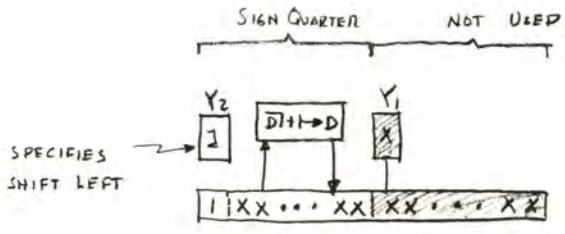
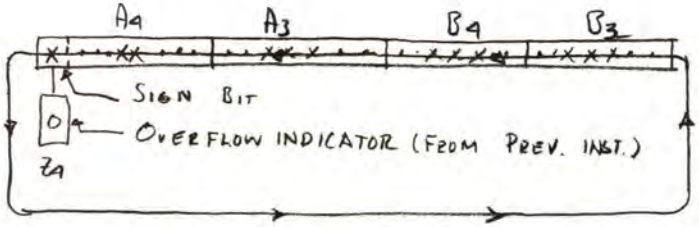


Fig 14-6 SAB EXAMPLE SHOWN FOR f_2 fracture & positive and negative operand subwords

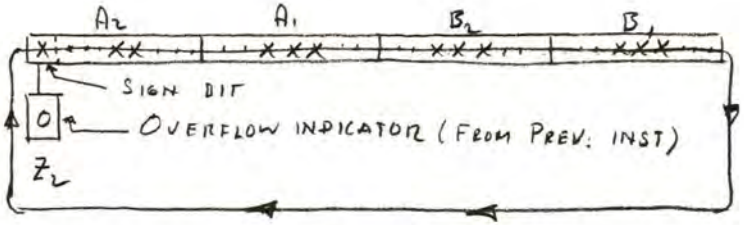




OPERAND IN D



$A_{4.9} \rightarrow B_{3.1}$
$A_{4.8} \rightarrow A_{4.9}$
$A_{4.7} \rightarrow A_{4.8}$
etc



$A_{2.9} \rightarrow A_{2.8}$
$B_{1.1} \rightarrow A_{2.9}$
$B_{1.2} \rightarrow B_{1.1}$
etc.

CASE 1
DATA IN AB
$Z_4 = Z_2 = 0$
(from previous)

CASE 2 IS THE SAME AS CASE 1. THE OVERFLOW INDICATORS ARE NOT AFFECTED BY THE INSTRUCTION

CASE 2
DATA IN AB
$Z_4 = Z_2 = 1$
(from previous)

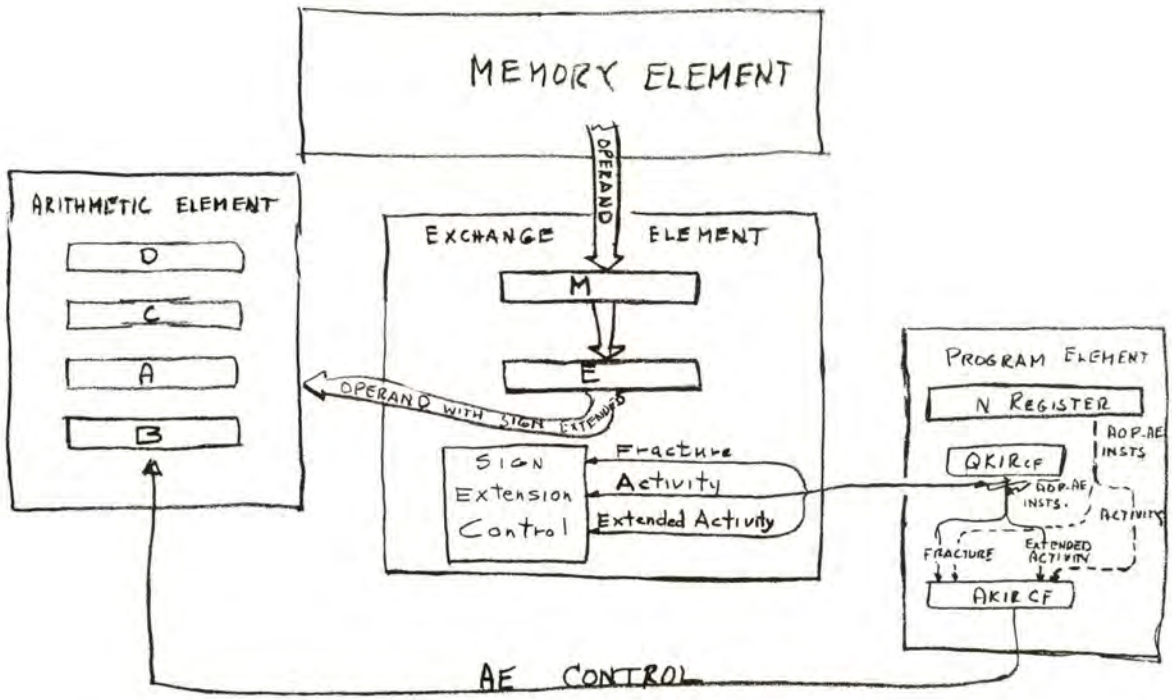
Fig 14-7 CAB Example shown for the fractional part of operand subwords.

ASK INSTRUCTIONS	ASK COUNTER	D COUNTER [In sign quarter of active subword]
MUL DIV	COUNTS NUMBER OF MULT STEP-PAD ITERATIONS	
SC- CY-		COUNTS OUT NUMBER OF SHIFTS SPECIFIED BY OPERAND. COUNTS UP TO ZERO FROM PRESET NEGATIVE VALUE.
NO-	LIMITS NUMBER OF SHIFTS IN CASES WHERE SUBWORD CONTAINS ALL ZEROES OR ALL ONES	SUBTRACTS NUMBER OF SHIFTS REQ'D TO NORMALIZE DATA FROM OPERAND BROUGHT FROM MEMORY. DATA IS NORMALIZED WHEN A OR AB LIES BETWEEN ½ AND 1
TLY	COUNTS NUMBER OF SHIFTS REQ'D TO COMPLETELY ROTATE SUBWORD.	ACCUMULATES NUMBER OF 1'S APPEARING IN DATA EXAMINED ACCUMULATION IS ADDED TO PREVIOUS CONTENTS OF D REGISTER

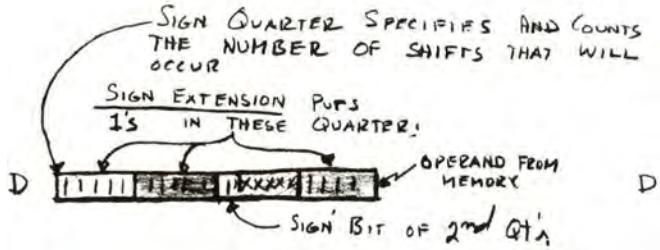
Fig 14-8 FUNCTION OF ASK AND D COUNTERS IN ASK TYPE INSTRUCTIONS

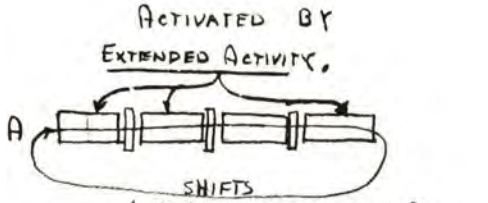
	A		Z (Sign g'r only)		B		C	D		Y
	BEFORE INST.	AFTER INST.	FUNCTION	FINAL VALUE	BEFORE INST	AFTER INST		BEFORE INST.	AFTER INST.	
ADD	AUGEND (from prev. inst.)	SUM	OVERFLOW INDICATION	OVERFLOW STATE			Keeps track of carries during inst. (Left cleared)	... XXX ...	ADDEND (Inst. operand)	Remembers Dig during inst.
SUB	SUBTRAHEND (from prev inst)	DIFFERENCE	Ditto	Ditto			Ditto	... XXX ...	MINUEND (INST. operand)	Ditto
DSA	DATA (from prev inst.	A ⊕ D → A (BEFORE INST, OPER. AND, AFTER INST)	NOT USED	UNCHANGED			C + (A · D) → C (C BEFORE INST., OPER. AND, C AFTER INST.)	... XXX ...	DATA (INST. OPERAND)	Ditto
MUL	MULTIPLIER (from prev inst)	Major half of PRODUCT	SIGN CONTROL	0	... XXX ...	Minor half of PRODUCT	Keeps track of carries during inst.	... XXX ...	MULTIPLICAND (Inst Operand)	Ditto
DIV	Major half of DIVIDEND (from prev inst.)	QUOTIENT	SIGN CONTROL & OVERFLOW	OVER FLOW STATE	Minor half of DIVIDEND (from prev inst)	REMAINDER	Ditto	... XXX ...	DIVISOR (Inst. Operand)	Ditto
SC-	SCA & SAB: DATA (from prev inst.)	Scaled data	OVERFLOW INDICATION	0	SCB + SAB: DATA (from prev inst)	Scaled data		BEGIN'G OF INST: SIGN QUARTER GIVES NO. OF SHIFTS TO OCCUR (INST. OPERAND)	END OF INST: MINUS ZERO	Ditto
CY-	CYA & CAB: DATA (from prev inst.)	Cycled data	OVERFLOW INDICATION	0	CYB + CAB: DATA (from prev inst)	Cycled data		Ditto	Ditto	Ditto
NO-	NOA & NAB: DATA (from prev inst.)	Normalized data	OVERFLW INDICATION	0	NOA & NAB: DATA (from prev inst)	Normalized Data		BEGIN'G OF INST: DATA (INST. OPERAND)	END OF INST: DATA plus number of shifts that occurr in An AB	Ditto
TLY	... XXX ...	DATA (INST. OPERAND)	NOT USED	UNCHANGED				DATA (from prev inst)	DATA plus number OF 1's in A.	UNCHANGED

Fig 14-9 AE Register Function For AK type instructions



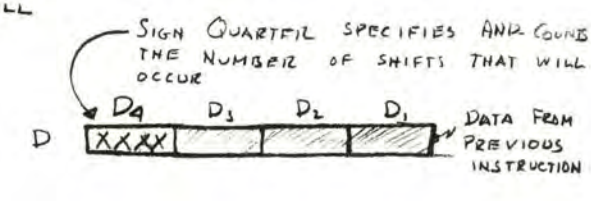
a) EFFECT OF CONFIGURATION CONTROL DURING AOP & $\overline{\text{AOP}}$ AE INSTRUCTIONS





EXA INSTRUCTION WITH 11↓1 configuration specification

NOTE THAT THE SIGN QUARTER (D_4) SPECIFIES THE NUMBER OF SHIFTS THAT WILL OCCUR, BUT THAT SIGN EXTENSION HAS PLACED 1's IN THIS QUARTER. THEREFORE NO SHIFTS WILL OCCUR.



AOP instruction specifying cycle A operation with 11↓1 configuration

NOTE THAT NO SHIFTS OCCUR IN A1, A3, & A4 SINCE THE AOP INSTRUCTION SPECIFIES ONLY A_2 WILL BE ACTIVE, HOWEVER THE COUPLING UNIT COUPLES $A_{3.1}$ to $A_{2.9}$. THIS MEANS THAT ON EACH SHIFT, $A_{3.1} \rightarrow A_{2.9}$. (e.g. if $A_{3.1}=0$ and D4 specifies 9 SHIFTS, A2 WOULD BE FILLED WITH ZEROES AT THE END OF THE INSTRUCTION.)

Fig 14-10
AOP & $\overline{\text{AOP}}$ AE INSTRUCTIONS

b) ILLUSTRATIVE EXAMPLE OF AOP & $\overline{\text{AOP}}$ AE INSTRUCTION SHOWING EFFECTS OF SIGN EXTENSION & ACTIVITY EXTENSION.

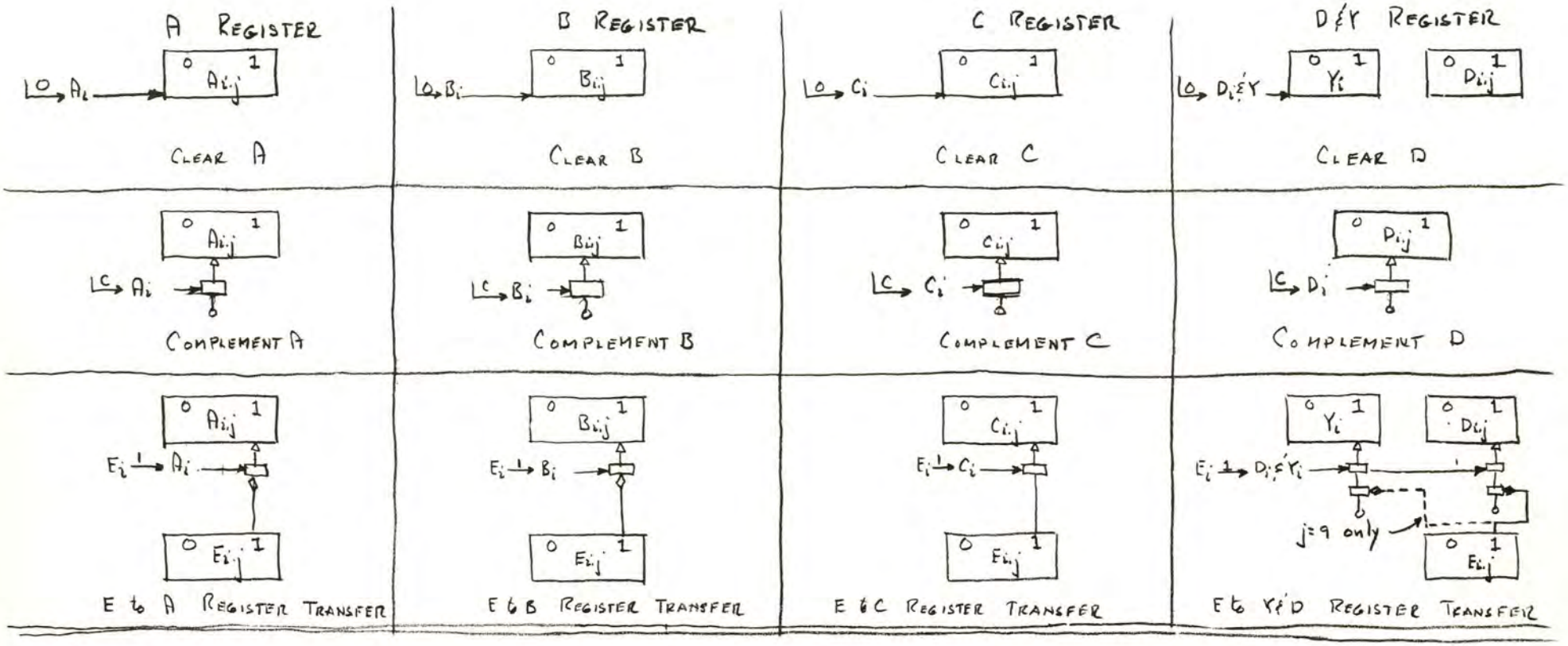
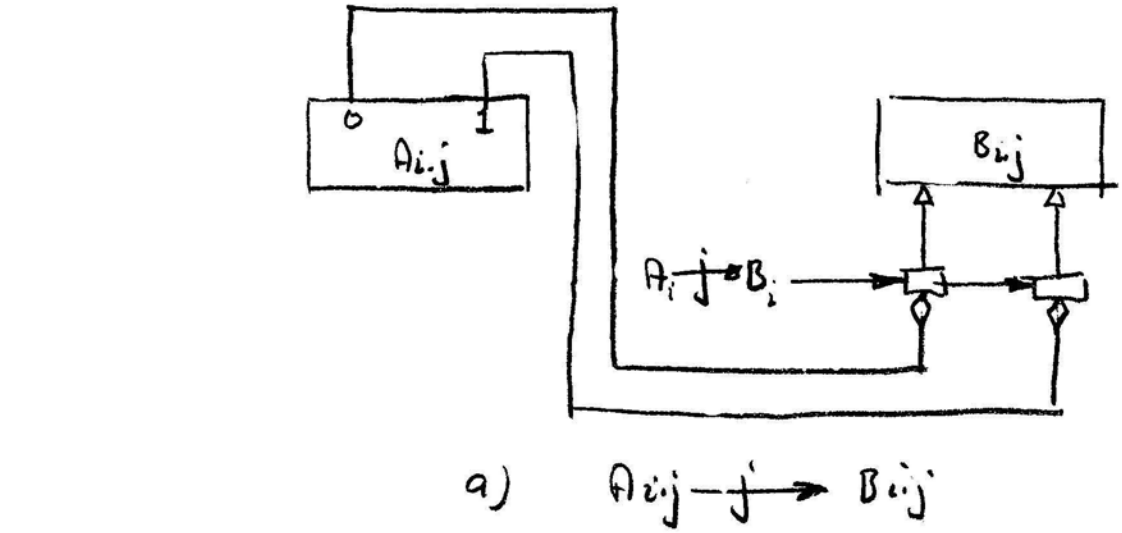
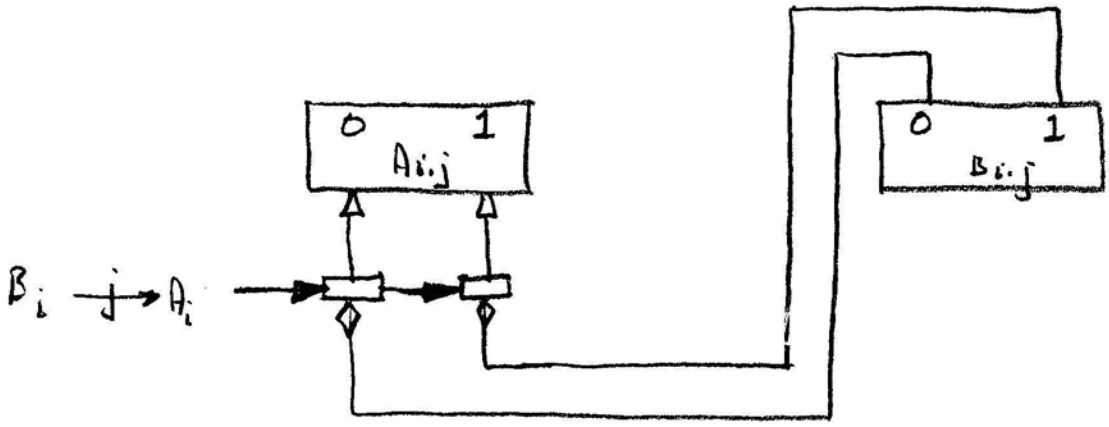


Fig. 19-11

TYPICAL AE REGISTER OPERATIONS



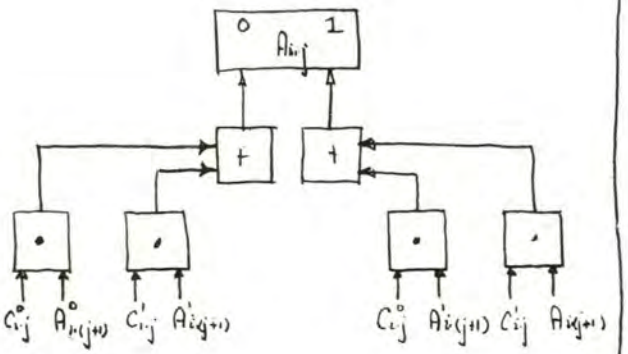
a) $A_{i.j} \xrightarrow{j} B_{i.j}$



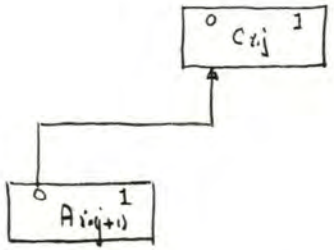
b) $B_{i.j} \xrightarrow{j} A_{i.j}$

FIG. 14-12 JAM TRANSFERS BETWEEN A & B

NOTE: IF BOTH THE $A_i \xrightarrow{j} B_i$ AND $B_i \xrightarrow{j} A_i$ PULSES ARE FIRED OFF SIMULTANEOUSLY, THE EFFECT IS TO INTERCHANGE A & B.

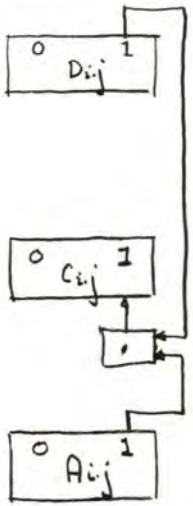


a) $A_{i(j+1)} \oplus C_{ij} \longrightarrow A_{ij}$

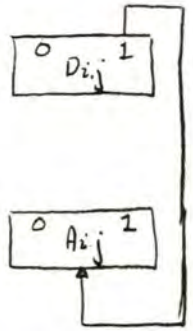


b) $A_{i(j+1)} \xrightarrow{0} C_{ij}$

Fig 19-14 TRANSFERS GATED BY MULTIPLY STEP Pulse



a) $C_{ij} + A_{ij} \cdot D_{ij} \longrightarrow C_{ij}$



b) $D_{ij} \oplus A_{ij} \longrightarrow A_{ij}$

Fig 19-13 TRANSFERS GATED BY Pad Pulse

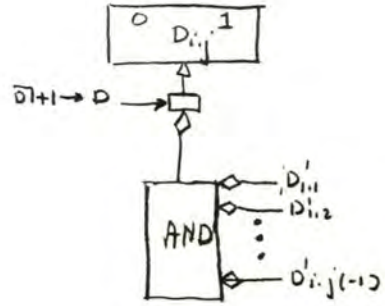


Fig 19-15 D REGISTER COUNT CIRCUIT

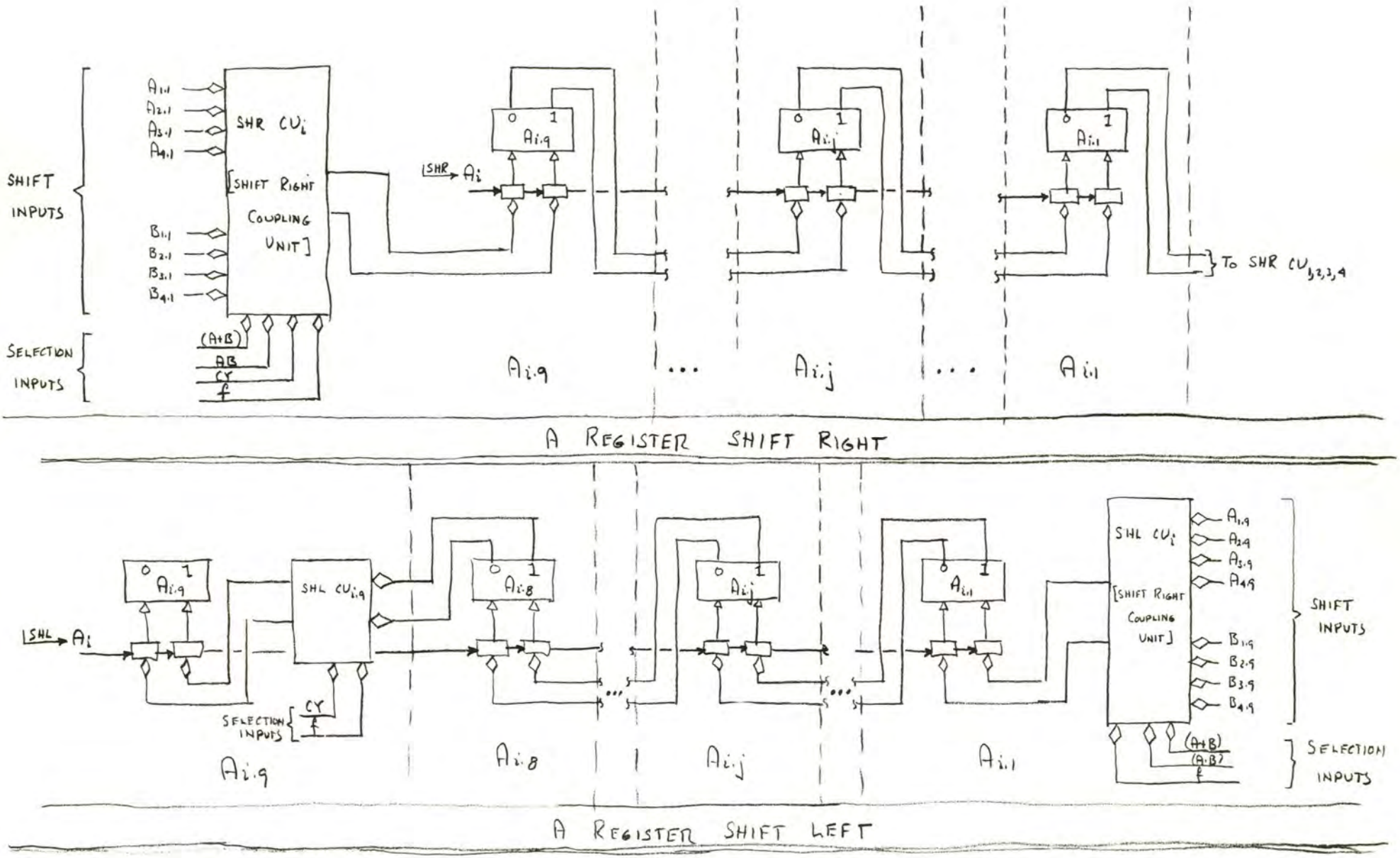


Fig 14-16 A REGISTER SHIFT LOGIC

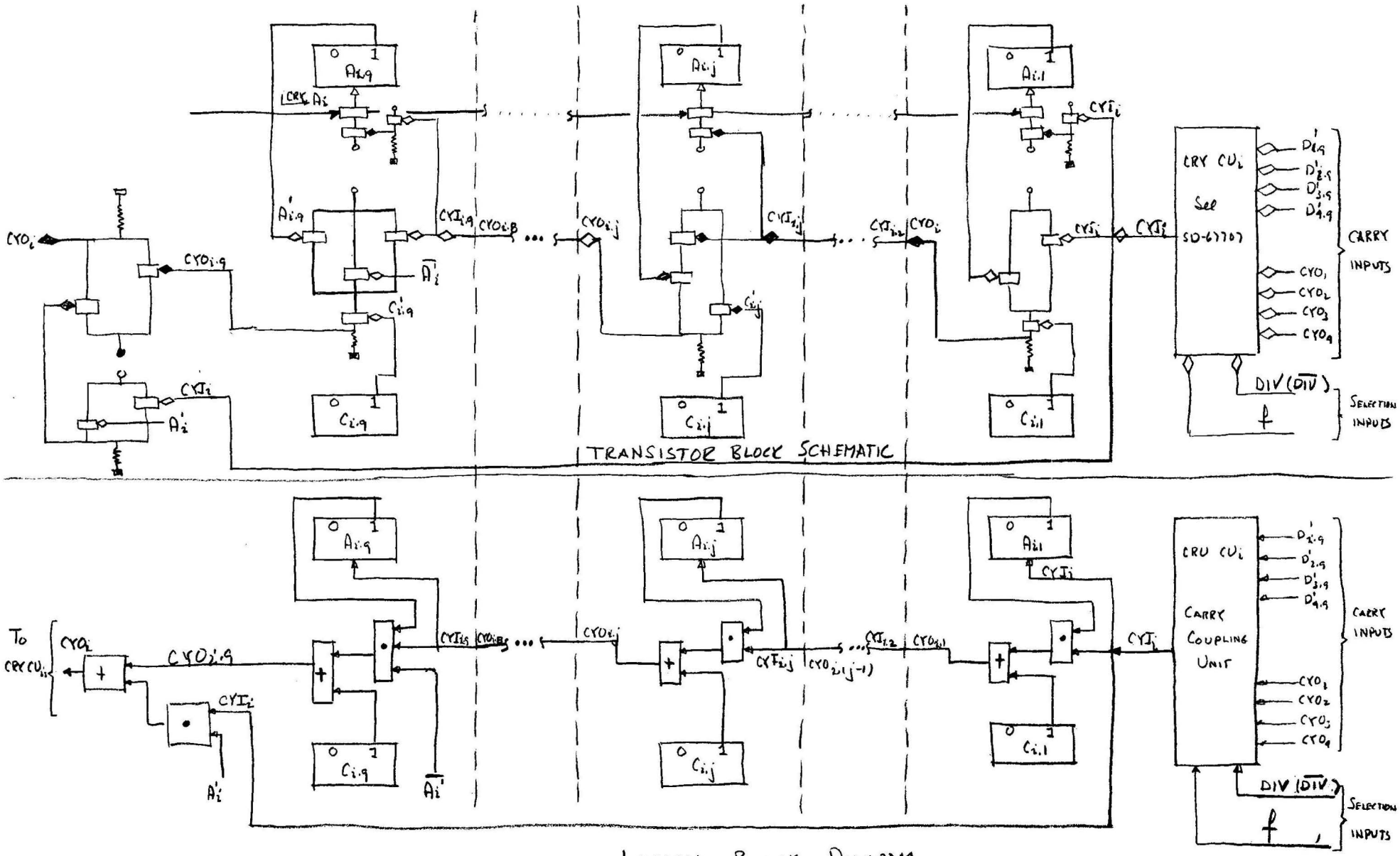


FIG 14-17 A REGISTER CARRY LOGIC

DECODED LEVEL	$AKIR_{OP1-6}$ 6	5	4	3	2	1
$AKIR_{op}^{0X}$	0	0	0	X	X	X
$(\quad)^{1X}$	0	0	1	X	X	X
$(\quad)^{2X}$	0	1	0	X	X	X
$(\quad)^{3X}$	0	1	1	X	X	X
$(\quad)^{4X}$	1	0	0	X	X	X
$(\quad)^{5X}$	1	0	1	X	X	X
$(\quad)^{6X}$	1	1	0	X	X	X
$(\quad)^{7X}$	1	1	1	X	X	X
$(\quad)^{X0}$	X	X	X	0	0	0
$(\quad)^{X1}$	X	X	X	0	0	1
$(\quad)^{X2}$	X	X	X	0	1	0
$(\quad)^{X3}$	X	X	X	0	1	1
$(\quad)^{X4}$	X	X	X	1	0	0
$(\quad)^{X5}$	X	X	X	1	0	1
$(\quad)^{X6}$	X	X	X	1	1	0
$(\quad)^{X7}$	X	X	X	1	1	1

Fig 4-18 $AKIR_{op}$ 1st LEVEL DECODERS

REF DRWGS: 87800

OP CODES	DECODED OP LEVEL	DECODER LOGIC
DSA = 65	$AKIR^{DSA}$	$AKIR_{op}^{6X} \cdot AKIR_{op}^{X5}$
NAB = 66	$(\quad)^{NAB}$	$(\quad)^{6X} \cdot (\quad)^{X6}$
SCA = 70	$(\quad)^{SCA}$	$(\quad)^{7X} \cdot (\quad)^{X0}$
SCB = 71	$(\quad)^{SCB}$	$(\quad)^{7X} \cdot (\quad)^{X1}$
SAB = 72	$(\quad)^{SAB}$	$(\quad)^{7X} \cdot (\quad)^{X2}$
Spare = 73	$(\quad)^{-}$	$(\quad)^{7X} \cdot (\quad)^{X3}$
TLY = 74	$(\quad)^{TLY}$	$(\quad)^{7X} \cdot (\quad)^{X4}$
DIV = 75	$(\quad)^{DIV}$	$(\quad)^{7X} \cdot (\quad)^{X5}$
MUL = 76	$(\quad)^{MUL}$	$(\quad)^{7X} \cdot (\quad)^{X6}$
SUB = 77	$(\quad)^{SUB}$	$(\quad)^{7X} \cdot (\quad)^{X7}$

Fig 4-19 $AKIR_{op}$ OP DECODERS

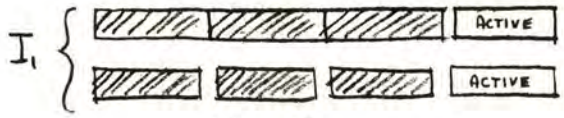
REF DRWGS: 87800

DECODED CLASS LEVEL	DECODER LOGIC	DECODER LOGIC EXPRESSED IN OCTAL OP CODES	DECODER LOGIC EXPRESSED IN MNEUMONIC OP CODES
$(AKIR)^{SH}$	$[AKIR_{OP}^{6X} + AKIR_{OP}^{7X}] \cdot [AKIR_{OP}^{X0} + AKIR_{OP}^{X1} + AKIR_{OP}^{X2}]$	$[60 + 61 + 62] + [70 + 71 + 72]$	SH = [CYA + CYB + CAB] + [SCA + SCB + SAB]
$(\quad)^{SHA}$	[ditto] · [ ]^{X0} + ( )^{X2}]	[60 + 62] + [70 + 72]	SHA = [CYA + CAB] + [SCA + SAB]
$(\quad)^{SHB}$	[ditto] · [ ]^{X1} + ( )^{X2}]	[61 + 62] + [71 + 72]	SHB = [CYB + CAB] + [SCB + SAB]
$(\quad)^{CY}$	$\{(\quad)^{6X} \cdot [(\quad)^{X0} + (\quad)^{X1} + (\quad)^{X2}]\} + \{(\quad)^{7X} \cdot [(\quad)^{X4} + (\quad)^{X5}]\}$	[60 + 61 + 62] + [74 + 75]	CY = [CYA + CYB + CAB] + [TLY + DIV]
$(\quad)^{AB}$	$\{(\quad)^{6X} \cdot [(\quad)^{X2} + (\quad)^{X6}]\} + \{(\quad)^{7X} \cdot [(\quad)^{X2} + (\quad)^{X5} + (\quad)^{X6}]\}$	[62 + 66] + [72 + 75 + 76]	AB = [CAB + NAB] + [SAB + DIV + MUL]
$(\quad)^{(A+B)}$	$\overline{(\quad)^{AB}}$	———	———
$(\quad)^{CY \cdot AB}$	$(\quad)^{CY} \cdot (\quad)^{AB}$	[62 + 75]	CY · AB = CAB + DIV
$(\quad)^{CY \cdot (A+B)}$	$(\quad)^{CY} \cdot (\quad)^{(A+B)}$	[60 + 61 + 74]	CY · (A+B) = CYA + CYB + TLY
$(\quad)^{2N}$	$\{(\quad)^{6X} \cdot [(\quad)^{X2} + (\quad)^{X6}]\} + [(\quad)^{7X} \cdot (\quad)^{X2}]$	[62 + 66 + 72]	2N = [CAB + NAB + SAB]
$(\quad)^{N}$	$\overline{(\quad)^{2N}}$	———	———
$(\quad)^{NOR}$	$\{(\quad)^{6X} \cdot [(\quad)^{X4} + (\quad)^{X6}]\}$	[64 + 66]	NOR = [NOA + NAB]
$(\quad)^{ADD}$	$\{[(\quad)^{6X} + (\quad)^{7X}] \cdot (\quad)^{7X}\}$	[61 + 77]	ADD = [ADD + SUB]
$(\quad)^{OCSAL}$	$AK_{d.1}' \cdot [\overline{(\quad)^{6X} \cdot (\quad)^{7X}} + (\quad)^{X3}]$	———	* OCSAL = $AK_{d.1}'$ · (UNDEFINED AE INST)
$(\quad)^{AOP}$	{ditto}	———	

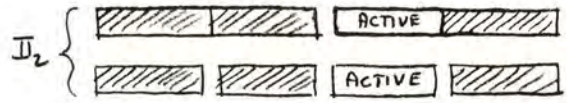
* AOP and OCSAL ARE CURRENTLY REDUNDANT

Fig 14-20 $AKIR_{OP}$ CLASS DECODER

REF. DRWG. 87801

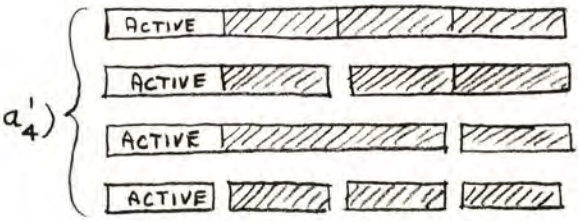


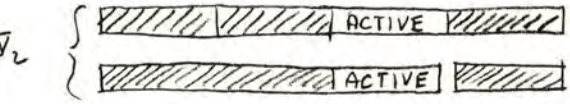
$$I = I_1 = a_1' \cdot (f_3 + f_4)$$

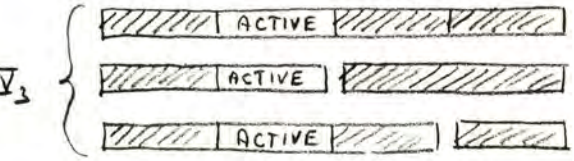


$$II = II_1 + II_2 = [a_1' \cdot f_2] + [a_2' \cdot (f_2 + f_4)]$$

$$III = III_3 = a_3' \cdot f_4$$







$$IV = IV_1 + IV_2 + IV_3 + \overline{IV_4} = a_1' \cdot f_1 + a_2' \cdot (f_1 + f_3) + a_3' \cdot \overline{f_4} + a_4'$$

FRACTIONS	$AKIR_{CF_j}$ 9	8	7	6	5	4	ACTIVITY
f_1	0	0					
f_2	0	1					
f_3	1	0					
f_4	1	1					
			X	X	X	0	a_1'
			X	X	0	X	a_2'
			X	0	X	X	a_3'
			0	X	X	X	a_4'

$AKIR_{CF}$ DECODING

THE ROMAN NUMERALS I, II, III, & IV INDICATE THE LEFTMOST QUARTER OF A SUBWORD WHICH CONTAINS AT LEAST ONE ACTIVE QUARTER.

REF DRWGS: 81804

FIG 14-21 AE CONFIGURATION LOGIC

A_i^0 & A_i^1 LOGIC	
A_1^0 =	$A^0_{1.1-1.9}$
A_1^1 =	$A^1_{1.1-1.9}$
A_2^0 =	$A^0_{2.1-2.9}$
A_2^1 =	$A^1_{2.1-2.9}$
A_3^0 =	$A^0_{3.1-3.9}$
A_3^1 =	$A^1_{3.1-3.9}$
A_4^0 =	$A^0_{4.1-4.9}$
A_4^1 =	$A^1_{4.1-4.9}$

Fig 14-22
A_i All zeroes, all ones Nets

REF DRWGS: 87816

G	$Ai.9 = Ai.8$
G_1	$= (A^0_{1.9} + A^1_{1.8}) \cdot (A^1_{1.9} + A^0_{1.8})$
G_2	$= (A^0_{2.9} + A^1_{2.8}) \cdot (A^1_{2.9} + A_{2.8})$
G_3	$= (A^0_{3.9} + A^1_{3.8}) \cdot (A^1_{3.9} + A^0_{3.8})$
G_4	$= (A^0_{4.9} + A^1_{4.8}) \cdot (A^1_{4.9} + A^0_{4.9})$

FIG 14-23 Sigma LOGIC

REF DRWG: 87816

CPLG UNIT	INTER QUARTER SHIFT RIGHT COUPLING (A+B) TYPE INSTRUCTIONS	AB TYPE INSTRUCTIONS	INSTRUCTIONS INDEPENDENT OF (A+B) or (AB) CONSIDERATIONS
SHR CU 1,2,0 ◇ $A^{1,0}_{1,9}$	$[CY\cdot(A+B)]\cdot[A^{1,0}_{1,1}\cdot f_3 + A^{1,0}_{1,9}\cdot f_4] + [CY\cdot AB]\cdot[B^{1,0}_{1,1}\cdot f_3 + B^{1,0}_{1,1}\cdot f_4] + [A^{1,0}_{2,1}(f_1 + f_2)]$		
SHR CU 1,2,0 ◇ $A^{1,0}_{1,5}$	$[\text{ditto}]\cdot[A^{1,0}_{1,1}\cdot f_2 + A^{1,0}_{2,1}\cdot f_4] + [\text{ditto}]\cdot[B^{1,0}_{1,1}\cdot f_2 + B^{1,0}_{2,1}\cdot f_4] + [A^{1,0}_{3,1}(f_1 + f_3)]$		
SHR CU 1,2,0 ◇ $A^{1,0}_{3,5}$	$[\text{ditto}]\cdot[A^{1,0}_{3,1}\cdot f_4] + [\text{ditto}]\cdot[B^{1,0}_{3,1}\cdot f_4] + [A^{1,0}_{4,1}(f_1 + f_2 + f_3)]$		
SHR CU 1,2,0 ◇ $A^{1,0}_{4,5}$	$[\text{ditto}]\cdot[A^{1,0}_{1,1}\cdot f_1 + A^{1,0}_{2,1}\cdot f_2 + A^{1,0}_{2,1}\cdot f_3 + A^{1,0}_{4,1}\cdot f_4] + [\text{ditto}]\cdot[B^{1,0}_{1,1}\cdot f_1 + B^{1,0}_{3,1}\cdot f_2 + B^{1,0}_{2,1}\cdot f_3 + B^{1,0}_{4,1}\cdot f_4]$		

CPLG UNIT	INTER QUARTER SHIFT LEFT COUPLING (A+B) TYPE INSTRUCTIONS	AB TYPE INSTRUCTIONS	INSTRUCTIONS INDEPENDENT OF (A+B) or (AB) CONSIDERATIONS
SHL CU 1,2,0 ◇ $B^{1,0}_{1,1}$	$[A+B]\cdot[A^{1,0}_{4,5}\cdot f_1 + A^{1,0}_{2,5}\cdot f_2 + A^{1,0}_{1,9}\cdot f_3 + A^{1,0}_{1,9}\cdot f_4] + [AB]\cdot[B^{1,0}_{4,5}\cdot f_1 + B^{1,0}_{2,9}\cdot f_2 + B^{1,0}_{1,9}\cdot f_3 + B^{1,0}_{1,9}\cdot f_4]$		
SHL CU 1,2,0 ◇ $A^{1,0}_{2,1}$	$[\text{ditto}]\cdot[A^{1,0}_{4,5}\cdot f_3 + A^{1,0}_{2,9}\cdot f_4] + [\text{ditto}]\cdot[B^{1,0}_{4,9}\cdot f_3 + B^{1,0}_{2,9}\cdot f_4] + [A^{1,0}_{1,9}(f_1 + f_2)]$		
SHL CU 1,2,0 ◇ $A^{1,0}_{3,1}$	$[\text{ditto}]\cdot[A^{1,0}_{4,9}\cdot f_2 + A^{1,0}_{3,9}\cdot f_4] + [\text{ditto}]\cdot[B^{1,0}_{4,9}\cdot f_2 + B^{1,0}_{3,5}\cdot f_4] + [A^{1,0}_{2,5}(f_1 + f_3)]$		
SHL CU 1,2,0 ◇ $A^{1,0}_{4,1}$	$[\text{ditto}]\cdot[A^{1,0}_{4,9}\cdot f_4] + [\text{ditto}]\cdot[B^{1,0}_{4,9}\cdot f_4] + [A^{1,0}_{3,5}(f_1 + f_2 + f_3)]$		

CPLG UNIT	INTRA QUARTER SHIFT LEFT COUPLING
SHL CU 1,2,0 ◇ $A^{1,0}_{1,9}$	$[CY + f_1 + f_2]\cdot A^{1,0}_{1,8}$
SHL CU 1,2,0 ◇ $A^{1,0}_{2,9}$	$[CY + f_1 + f_3]\cdot A^{1,0}_{2,8}$
SHL CU 1,2,0 ◇ $A^{1,0}_{3,9}$	$[CY + f_1 + f_2 + f_3]\cdot A^{1,0}_{3,8}$
SHL CU 1,2,0 ◇ $A^{1,0}_{4,9}$	$[CY]\cdot A^{1,0}_{4,8}$

	BLOCK SCHEMATIC REF.	CHAPTER 14 REF
SHR CU		
SHL CU		
CY		
AB		
A+B		

WHERE: $CY = AKIR^{CY} = (CYA + CYB + CAB) + (TLY + DIV)$

$AB = AKIR^{AB} = (CAB + NAB + SAB) + (MUL + DIV)$

$(A+B) = AKIR^{(A+B)}$

$CY\cdot(AB) = AKIR^{CY\cdot AB} = CAB + DIV$

$CY\cdot(A+B) = AKIR^{CY\cdot(A+B)} = (CYA + CYB) + TLY$

REF Dwgs: 67699
67707

Fig 14-24
A SHIFT COUPLING UNIT LOGIC

INTER QUARTER SHIFT RIGHT COUPLING

	(A+B) TYPE INSTRUCTION	AB TYPE INSTRUCTIONS	INSTRUCTIONS INDEPENDENT OF (A+B) or (AB) CONSIDERATIONS
SHR CU $\overset{1,0}{\diamond}$ $B^{1,0}_{1,9}$	$[CY\cdot(A+B)]\cdot[B^{1,0}_{1,1}\cdot f_3 + B^{1,0}_{1,1}\cdot f_4]$	$+[AB]\cdot[A^{1,0}_{1,1}\cdot f_3 + A^{1,0}_{1,1}\cdot f_4]$	$+[B^{1,0}_{2,1}\cdot(f_1 + f_2)]$
SHR CU $\overset{1,0}{\diamond}$ $B^{1,0}_{2,9}$	$[\text{ditto}]\cdot[B^{1,0}_{1,1}\cdot f_2 + B^{1,0}_{2,1}\cdot f_4]$	$+[\text{ditto}]\cdot[A^{1,0}_{1,1}\cdot f_2 + A^{1,0}_{2,1}\cdot f_4]$	$+[B^{1,0}_{3,1}\cdot(f_1 + f_3)]$
SHR CU $\overset{1,0}{\diamond}$ $B^{1,0}_{3,9}$	$[\text{ditto}]\cdot[B^{1,0}_{3,1}\cdot f_4]$	$+[\text{ditto}]\cdot[A^{1,0}_{1,3}\cdot f_4]$	$+[B^{1,0}_{4,1}\cdot(f_1 + f_2 + f_3)]$
SHR CU $\overset{1,0}{\diamond}$ $B^{1,0}_{4,9}$	$[\text{ditto}]\cdot[B^{1,0}_{1,1}\cdot f_1 + B^{1,0}_{2,1}\cdot f_2 + B^{1,0}_{3,1}\cdot f_3 + B^{1,0}_{4,1}\cdot f_4]$	$+[\text{ditto}]\cdot[A^{1,0}_{1,1}\cdot f_1 + A^{1,0}_{1,2}\cdot f_2 + A^{1,0}_{1,3}\cdot f_3 + A^{1,0}_{1,4}\cdot f_4]$	

INTER QUARTER SHIFT LEFT COUPLING

	(A+B) TYPE INSTRUCTIONS	AB TYPE INSTRUCTIONS	INSTRUCTIONS INDEPENDENT OF (A+B) or (AB) CONSIDERATIONS
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{1,1}$	$[A+B]\cdot[B^{1,0}_{4,9}\cdot f_1 + B^{1,0}_{2,9}\cdot f_2 + B^{1,0}_{1,9}\cdot f_3 + B^{1,0}_{1,9}\cdot f_4]$	$+[AB]\cdot[A^{1,0}_{4,9}\cdot f_1 + A^{1,0}_{2,9}\cdot f_2 + A^{1,0}_{1,9}\cdot f_3 + A^{1,0}_{1,9}\cdot f_4]$	
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{2,1}$	$[\text{ditto}]\cdot[B^{1,0}_{4,9}\cdot f_3 + B^{1,0}_{2,9}\cdot f_4]$	$+[\text{ditto}]\cdot[A^{1,0}_{4,9}\cdot f_3 + A^{1,0}_{2,9}\cdot f_4]$	$+[B^{1,0}_{1,9}(f_1 + f_2)]$
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{3,1}$	$[\text{ditto}]\cdot[B^{1,0}_{3,1}\cdot f_2 + B^{1,0}_{3,9}\cdot f_4]$	$+[\text{ditto}]\cdot[A^{1,0}_{4,9}\cdot f_2 + A^{1,0}_{3,9}\cdot f_4]$	$+[B^{1,0}_{2,9}(f_1 + f_3)]$
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{4,1}$	$[\text{ditto}]\cdot[B^{1,0}_{4,9}\cdot f_4]$	$+[\text{ditto}]\cdot[A^{1,0}_{4,9}\cdot f_4]$	$+[B^{1,0}_{3,9}(f_1 + f_2 + f_3)]$

INTRA QUARTER SHIFT LEFT COUPLING

SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{1,9}$	$[CY + AB + (f_1 + f_2)]\ B^{1,0}_{1,8}$
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{2,9}$	$[CY + AB + (f_1 + f_3)]\ B^{1,0}_{2,8}$
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{3,9}$	$[CY + AB + (f_1 + f_2 + f_3)]\ B^{1,0}_{3,8}$
SHL CU $\overset{1,0}{\diamond}$ $B^{1,0}_{4,9}$	$[CY + AB]\cdot B^{1,0}_{4,8}$

WHERE: $CY = AKIR^{CY}$ =

$AB = AKIR^{AB}$

$(A+B) = AKIR^{(A+B)}$

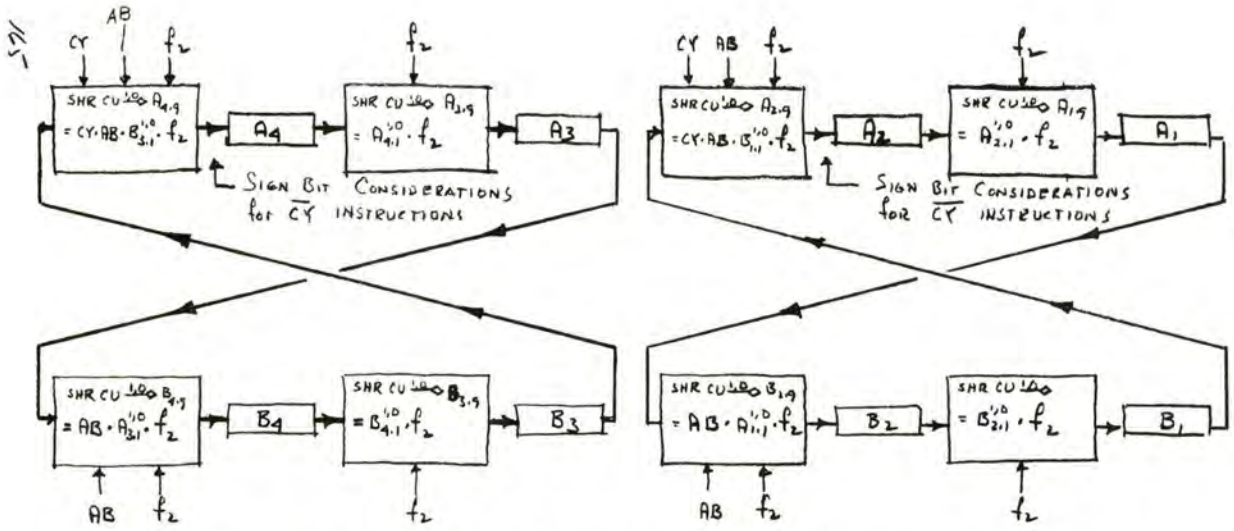
$CY\cdot(AB) = AKIR^{CY\cdot AB}$

$CY\cdot(A+B) = AKIR^{CY\cdot(A+B)}$

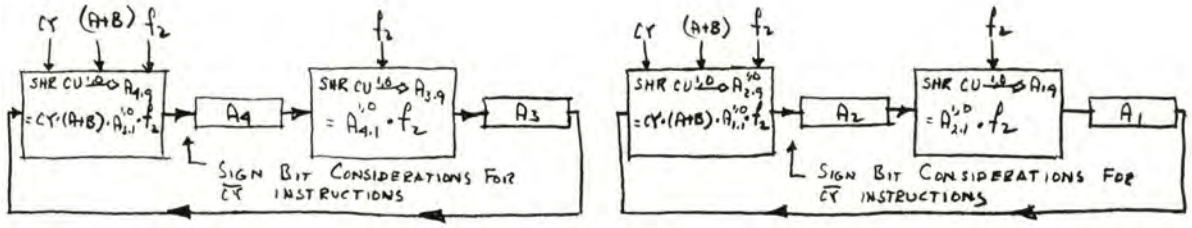
	Block Schematic Ref.	Chapter 14 Ref.
SHR CU		
SHL CU		
CY		
AB		
A+B		

REF DWGS: 67699
67708

Fig 19-25
B SHIFT COUPLING UNIT LOGIC



a) SHIFT RIGHT COUPLING IN REGISTERS A and B
for
CY·AB TYPE INSTRUCTION (e.g. CAB)
WITH f_2 FRACTURE



b) SHIFT RIGHT COUPLING IN REGISTER A
for
CY·(A+B) TYPE INSTRUCTION (e.g. CYA)
WITH f_2 FRACTURE

Fig 14-26 EXAMPLES OF SHIFT COUPLING
(See Figs 14-24 & 14-25 for coupling logic)

CARRY COUPLING UNIT LOGIC

	$\overline{DIV}$	DIV	INDEPENDENT OF DIV or $\overline{DIV}$ CONSIDERATIONS
CYI_1	$\overline{DIV}[CYO_4 \cdot f_1 + CYO_2 \cdot f_2 + CYO_1 \cdot f_3 + CYO_1 \cdot f_4] + DIV[D'_{4.9} \cdot f_1 + D'_{2.9} \cdot f_2 + D'_{1.9} \cdot f_3 + D'_{1.9} \cdot f_4]$		
CYI_2	$\overline{DIV}[CYO_4 \cdot f_3 + CYO_2 \cdot f_4] + DIV[D'_{4.9} \cdot f_3 + D'_{2.9} \cdot f_4]$		$+[CYO_1(f_1 + f_2)]$
CYI_3	$\overline{DIV}[CYO_4 \cdot f_2 + CYO_3 f_4] + DIV[D'_{4.9} \cdot f_2 + D'_{3.9} \cdot f_4]$		$+[CYO_2(f_1 + f_3)]$
CYI_4	$\overline{DIV}[CYO_4 \cdot f_4] + DIV[D'_{4.9} f_4]$		$+[CYO_3(f_1 + f_2 + f_3)]$

WHERE: $CYO_i = CYO_{i.9} + CYI_i \cdot A'_i$

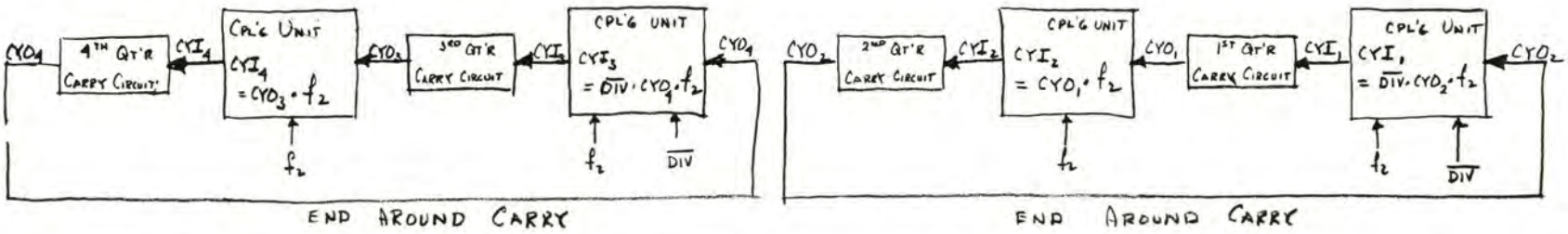
$CYO_{i.9} = C'_{i.9} + CYI_{i.9} \cdot A'_{i.9} \cdot \overline{A_i}$

$CYI_{i.j} = CYO_{i.(j-1)}$

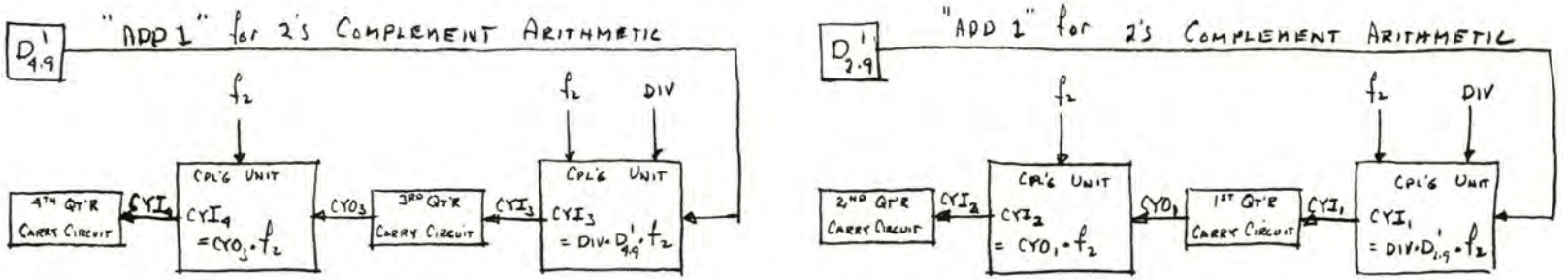
NOTE ① IN ANY GIVEN SUBWORD, the INTER QUARTER CARRY IN THE SUBWORD IS THE SAME FOR BOTH DIV AND $\overline{DIV}$, WHEREAS THE END AROUND CARRY IS WHAT DIFFERS FOR THESE TWO CASES.

	BLOCK SCHEMATIC REF	CHAPTER 14 REF
CYI_i		
CYO_i		

Fig 14-27
CARRY COUPLING UNIT LOGIC



a) CARRY COUPLING for $\overline{DIV}$ TYPE INSTRUCTION with f_2 Fracture.



b) CARRY COUPLING for DIV TYPE INSTRUCTION WITH f_2 FRACTURE

Fig 14-28 EXAMPLES OF CARRY COUPLING
(See Fig 14-27 for coupling logic)

$$AEJ =$$

$$\Big[(PKIR^{JPA} \cdot A^{\circ}_{1,9} + PKIR^{JNA} \cdot A^{\prime}_{1,9}) \cdot QKIR^{EXT\ ACT_1} \cdot \overline{A}^{\prime}_1 \cdot \overline{A}^{\circ}_1 \cdot QKIR^{f_3+f_4}$$
$$+ (PKIR^{JPA} \cdot A^{\circ}_{2,9} + PKIR^{JNA} \cdot A^{\prime}_{2,9}) \cdot QKIR^{EXT\ ACT_2} \cdot [\overline{A}^{\prime}_1 \cdot QKIR^{f_2} + \overline{A}^{\prime}_2 \cdot QKIR^{f_2+f_4}] \cdot [\overline{A}^{\circ}_1 \cdot QKIR^{f_2} + \overline{A}^{\circ}_2 \cdot QKIR^{f_2+f_4}]$$
$$+ (PKIR^{JPA} \cdot A^{\circ}_{3,9} + PKIR^{JNA} \cdot A^{\prime}_{3,9}) \cdot QKIR^{EXT\ ACT_3} \cdot \overline{A}^{\circ}_3 \cdot \overline{A}^{\prime}_3 \cdot QKIR^{f_4}$$
$$+ (PKIR^{JPA} \cdot A^{\circ}_{4,9} + PKIR^{JNA} \cdot A^{\prime}_{4,9}) \cdot QKIR^{EXT\ ACT_4} \cdot [\overline{A}^{\circ}_4 + \overline{A}^{\circ}_3 \cdot \overline{QKIR}^{f_4} + \overline{A}^{\circ}_2 \cdot QKIR^{f_1+f_3} + \overline{A}^{\circ}_1 \cdot QKIR^{f_1}] \cdot [\overline{A}^{\prime}_4 + \overline{A}^{\prime}_3 \cdot \overline{QKIR}^{f_4} + \overline{A}^{\prime}_2 \cdot QKIR^{f_1+f_3} + \overline{A}^{\prime}_1 \cdot QKIR^{f_1}] \Big]$$

$$+$$

$$\Big[PKIR^{JOV} \cdot Z^{\prime}_1 \cdot QKIR^{EXT\ ACT_1} \cdot QKIR^{f_3+f_4}$$
$$PKIR^{JOV} \cdot Z^{\prime}_2 \cdot QKIR^{EXT\ ACT_2} \cdot QKIR^{f_2+f_4}$$
$$PKIR^{JOV} \cdot Z^{\prime}_3 \cdot QKIR^{EXT\ ACT_3} \cdot QKIR^{f_4}$$
$$PKIR^{JOV} \cdot Z^{\prime}_4 \cdot QKIR^{EXT\ ACT_4} \Big]$$

Fig 14-29
AE JUMP NFT LOGIC

REF DWGS: 87811

LO◆ AEP =

$$AK'_{d.4}\cdot AKIR^{NOR}\cdot\left(\overline{A_1^0}\cdot\overline{A_1'}+\overline{I}\right)\cdot\left(\overline{A_2^0}\cdot\overline{A_2'}+\overline{II}\right)\cdot\left(\overline{A_3^0}\cdot\overline{A_3'}+\overline{III}\right)\cdot\left(\overline{A_4^0}\cdot\overline{A_4'}+\overline{IV}\right)$$

$$+\ AK'_{d.4}\cdot AKIR^{SH}\cdot\left(LAD_1+\overline{I}\right)\cdot\left(LAD_2+\overline{II}\right)\cdot\left(LAD_3+\overline{III}\right)\cdot\left(LAD_4+\overline{IV}\right)$$

$$+\ AK'_{d.1}\cdot\left(AKIR^{ADD}+AKIR^{DSA}\right)$$

$$+\ AK'_{d.9}\cdot\left(ASK'_1\cdot ASK^0_7\right)\cdot AKIR^{DIV}$$

$$+\ \left[\left(AK'_{d.2}\cdot AKIR^{TLY}\right)+\left(AK'_{d.4}\cdot AKIR^{NOR}\right)\right]\cdot\left[ASK'_7\cdot ASK'_6\cdot ASK'_5\cdot ASK'_4\cdot ASK^0_2\cdot ASK'_2\right]$$

$$+\ AK'_{d.3}\cdot ASK^0_7\cdot AKIR^{MUL}$$

$$+\ AKIR^{OCSAL}$$

WHERE:

$$LAD_i = D'_{i.4}\cdot D'_{i.5}\cdot D'_{i.6}\cdot D'_{i.7}\cdot D'_{i.8} = D'_{i.j}\ (j=4-8)$$

LAD = (Long) anticipation of the completion of the count in the i^th quarter of D

	Block Schematic Ref	Chapter 14 Ref
LO◆ AEP	87817	
AKIR	87801	
Roman Numerals	87804	
AK	87803	
ASK	87802	
LAD	67680	

Ref Drwgs: 87817

FIG 14-30 LO◆ AEP Logic

AE RD Pulse	RD Pulse Parameters										
	RN	RN_i	a_i	f	Y_i	Z_i	FD_i	G_i	AK	ASK	QK
$\overline{D}+1 \rightarrow D$	X						X	X	X		
SHR → A		X	X		X	X			X		
SHL → A		X	X		X			X	X		
SHR → B		X	X		X	X			X		
SHR → B		X	X		X			X	X		
Mult Step A & C			X						X		
Mult Step Sign $A_{i,9}$ & $C_{i,9}$		X							X		
CRY → A			X	X					X		
PAD → A & C		X	X						X	X	
PAD → Z	X								X		
$A_{i,9}$ Sign → Z	X								X		
Reset → Z	X								X		
0 → Z	X								X		
0 → A			X						X		X
0 → B			X								X
0 → C			X						X		X
0 → D			X								X
C → A		X							X		X
C → $A_{i,9}$	X								X		
C → B		X							X		X
C → D	X	X							X	X	
E → A			X								X
E → B			X								X
E → C			X								X
E → D			X								X
A → B			X						X		
B → A			X						X		

RN — Roman Numeral: Sign Quarter (leftmost quarter of subword which contains at least one active quarter)
RN_i — Roman Numeral indicates the sign quarter, i indicates active quarter in the subword.
a_i — Activity: i indicates quarters in AE activated by "extended activity" or AOP instruction
f — Fracture: As specified by configuration
Y_i — D Sign Bits: Remember State of $D_{i,9}$ throughout instruction
Z_i — A Sign Bit or Overflow Indicator
FD_i — Finished ND: Indicates count is finished in i^{th} quarter of D
G_i — Indicates that $A_{i,9} = A_{i,8}$ in i^{th} quarter

Fig 14-31.
AE RD Pulses

AE PULSE GATE LOGIC

RD PULSE	GATING LOGIC	FF INPUT
$[\overline{D}+1 \rightarrow D]$	$\cdot [D'_{i,(j-1)} \cdots D'_{i,1}]$	C $\lfloor C \rfloor$ $D_{i,j}$
$[\lfloor \overset{SHR}{\rightarrow} A]$	$\cdot [SHR\ CU_i^{0,1}]$	C $\lfloor 0,1 \rfloor$ $A_{i,9}$
[ditto]	$\cdot [A_{i,j}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $A_{i,(j-1)}$
$[\lfloor \overset{SHL}{\rightarrow} A]$	$\cdot [SHL\ CU_i^{0,1}]$	C $\lfloor 0,1 \rfloor$ $A_{i,1}$
[ditto]	$\cdot [SHL\ CU_{i,9}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $A_{i,9}$
[ditto]	$\cdot [A_{i,j}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $A_{i,(j+1)}$
$[\lfloor \overset{SHR}{\rightarrow} B]$	$\cdot [SHR\ CU_i^{0,1}]$	
[ditto]	$\cdot [B_{i,j}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $B_{i,(j-1)}$
$[\lfloor \overset{SHL}{\rightarrow} B]$	$\cdot [SHL\ CU_i^{0,1}]$	C $\lfloor 0,1 \rfloor$ $B_{i,1}$
[ditto]	$\cdot [SHL\ CU_{i,9}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $B_{i,9}$
[ditto]	$\cdot [A_{i,j}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $B_{i,(j+1)}$
$[\lfloor \overset{MULT\ STEP}{\rightarrow} A,C]$	$\cdot [C_{i,j} = A_{i,(j+1)}]$	C $\lfloor 0 \rfloor$ $A_{i,j}$
[ditto]	$\cdot [C_{i,j} \neq A_{i,(j+1)}]$	C $\lfloor 1 \rfloor$ $A_{i,j}$
[ditto]	$\cdot [A_{i,(j+1)}^{0}]$	C $\lfloor 0 \rfloor$ $C_{i,j}$
$[\lfloor \overset{MULT\ STEP\ SIGN}{\rightarrow} A_{i,9}, C_{i,9}]$	$\cdot [C_{i,9} = A_{(i+1),1}]$	C $\lfloor 1 \rfloor$ $A_{i,9}$
[ditto]	$\cdot [C_{i,9} \neq A_{(i+1),1}]$	C $\lfloor 0 \rfloor$ $A_{i,9}$
[ditto]	$\cdot [A_{(i+1),1}^{0}]$	C $\lfloor 0 \rfloor$ $C_{i,e}$ $i \neq$

RD PULSE	GATING LOGIC	FF INPUT
$[\lfloor \overset{CRY}{\rightarrow} A]$	$\cdot [CYI_i]$	C $\lfloor C \rfloor$ $A_{i,1}$
[ditto]	$\cdot [CYI_{i,j}]$	C $\lfloor C \rfloor$ $A_{i,j}$
$[\lfloor \overset{PAD}{\rightarrow} A, C]$	$\cdot [A'_{i,j} \cdot D'_{i,j}]$	C $\lfloor 1 \rfloor$ $C_{i,j}$
[ditto]	$\cdot [D'_{i,j}]$	C $\lfloor C \rfloor$ $A_{i,j}$
$[\lfloor \overset{PAD}{\rightarrow} Z]$	See Fig. 19-41	
$[\lfloor \overset{A_{i,9}\ SIGN}{\rightarrow} Z]$		
$[\lfloor \overset{RESET}{\rightarrow} Z]$		
$[\lfloor 0 \rightarrow Z]$		
$[\lfloor 0 \rightarrow A]\ B]\ C]\ D]$		C $\lfloor 0 \rfloor$, A, B, C, D
$[\lfloor C \rightarrow A]\ B]\ C]\ D]$		C $\lfloor C \rfloor$ A, B, C, D
$[\lfloor C \rightarrow A_{i,9}]$		C $\lfloor C \rfloor$ $A_{i,9}$
$[E \overset{1}{\rightarrow} A]$	$\cdot [E_{i,j}]$	C $\lfloor 1 \rfloor$ $A_{i,j}$
$[E \overset{1}{\rightarrow} B]$	$\cdot [$ ditto $]$	C $\lfloor 1 \rfloor$ $B_{i,j}$
$[E \overset{1}{\rightarrow} C]$	$\cdot [$ ditto $]$	C $\lfloor 1 \rfloor$ $C_{i,j}$
$[E \overset{1}{\rightarrow} D]$	$\cdot [$ ditto $]$	C $\lfloor 1 \rfloor$ $D_{i,j}$
[ditto]	$\cdot [E'_{i,9}]$	C $\lfloor 1 \rfloor$ Y_i
$[A \overset{j}{\rightarrow} B]$	$\cdot [A_{i,j}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $B_{i,j}$
$[B \overset{j}{\rightarrow} A]$	$\cdot [B_{i,j}^{0,1}]$	C $\lfloor 0,1 \rfloor$ $A_{i,j}$

Fig 19-32
AE PULSE GATE LOGIC

D Counter RD Logic

RD Pulse	RD Clock Pulse	CYA, CYB, CAB; SCA, SCB, SAB			TLY			NOA, NAB					
		Time Levels	INST	OTHER	Time Level	INST	OTHER	Time Level	INST	OTHER	Time Level	INST	OTHER
$\overline{D_1}+1 \rightarrow D_1$	α	$[(AK'_{J.3}+AK'_{J.4})\cdot AKIR^{SH}]\cdot\overline{FD_1}\cdot I$	+		$[AK'_{J.2}\cdot AKIR^{TLY}]\cdot A'_{1.9}\cdot I$	+		$[AK'_{J.4}\cdot AKIR^{NOR}]\cdot G_1\cdot I$	+		$[AK'_{J.2}\cdot AKIR^{NOR}]\cdot Z'_1\cdot I$	+	
$\overline{D_2}+1 \rightarrow D_2$	α	[← ditto →]$\cdot\overline{FD_2}\cdot II$	+		[← ditto →]$\cdot A'_{2.9}\cdot II$	+		[← ditto →]$\cdot G_2\cdot II$	+		[← ditto →]$\cdot Z'_2\cdot II$	+	
$\overline{D_3}+1 \rightarrow D_3$	α	[← ditto →]$\cdot\overline{FD_3}\cdot III$	+		[← ditto →]$A'_{3.9}\cdot III$	+		[← ditto →]$\cdot G_3\cdot III$	+		[← ditto →]$\cdot Z'_3\cdot III$	+	
$\overline{D_4}+1 \rightarrow D_4$	α	[← ditto →]$\cdot FD_4\cdot IV$	+		[← ditto →]$A'_{4.9}\cdot IV$	+		[← ditto →]$\cdot G_4\cdot IV$	+		[← ditto →]$\cdot Z'_4\cdot IV$	+	

WHERE; $FD_i = D'_{i\cdot(1-8)} = D'_{i.8}\cdot D'_{i.7}\cdot D'_{i.6}\cdot D'_{i.5}\cdot D'_{i.4}\cdot D'_{i.3}\cdot D'_{i.2}\cdot D'_{i.1}$

$G_i \supset A_{i.9} = A_{i.8}$

N2 $Z_i = AK'_{J.2}\cdot AKIR^{NOR}\cdot Z'_i$

	Block Schematic REF	Chapter 14 REF
$\overline{D_i}+1 \rightarrow D_i$	67680 87807	
AK		
AKIR( )		
$\overline{G_i}$		
FD_i		
RN		

Fig. 14-33

D Counter RD Logic

SHIFT RIGHT IN A RD LOGIC

RD ULSE	RD CLOCK PULSE	NOA, NAB: TIME LEVEL	NOA, NAB: INST.	NOA, NAB: OTHERS	CYA, CYB ; SCA, SAB: TIME LEVELS	CYA, CYB ; SCA, SAB: INST	CYA, CYB ; SCA, SAB: OTHERS	TLY: TIME LEVEL	TLY: INST.	TLY: OTHERS
SHR → A_1	α	$[AK'_{d.2} \cdot AKIR^{NOR}] \cdot Z'_1 \cdot I_1$			$+[(AK'_{d.3} + AK'_{d.4}) \cdot AKIR^{SHA}] \cdot Y'_1 \cdot \overline{FD_1} \cdot I_1$					
		$+[\leftarrow$ ditto $\rightarrow] \cdot Z'_2 \cdot II_1$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_2 \cdot \overline{FD_2} \cdot II_2$			$+ [AK'_{d.2} \cdot AKIR^{TLY}] \cdot a'_1$		
		$+[\leftarrow$ ditto $\rightarrow] \cdot Z'_4 \cdot IV_1$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_1$					
SHR → A_2	α	$[\leftarrow$ ditto $\rightarrow] \cdot Z'_2 \cdot II_2$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_2 \cdot \overline{FD_2} \cdot II_2$			$+ [\leftarrow$ ditto $\rightarrow] \cdot a'_2$		
		$+[\leftarrow$ ditto $\rightarrow] \cdot Z'_4 \cdot IV_2$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_2$					
SHR → A_3	α	$[\leftarrow$ ditto $\rightarrow] \cdot Z'_3 \cdot III$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_3 \cdot \overline{FD_3} \cdot III$			$+ [\leftarrow$ ditto $\rightarrow] \cdot a'_3$		
		$+[\leftarrow$ ditto $\rightarrow] \cdot Z'_4 \cdot IV_3$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_3$					
SHR → A_4	α	$[\leftarrow$ ditto $\rightarrow] \cdot Z'_4 \cdot IV$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV$			$+ [\leftarrow$ ditto $\rightarrow] \cdot a'_4$		

SHIFT LEFT IN A RD LOGIC

RD PULSE	RD CLOCK PULSE	NOA, NAB: TIME LEVEL	NOA, NAB: INST	NOA, NAB: OTHERS	CYA, CYB ; SCA, SAB: TIME LEVELS	CYA, CYB ; SCA, SAB: INST	CYA, CYB ; SCA, SAB: OTHERS	DIV: TIME LEVELS	DIV: INST	DIV: OTHERS
SHL → A_1	α	$[AK'_{d.2} \cdot AKIR^{NOR}] \cdot G_1 \cdot I$			$+[(AK'_{d.3} + AK'_{d.4}) \cdot AKIR^{SHA}] \cdot Y_1^0 \cdot \overline{FD_1} \cdot I_1$					
		$+[$ ditto $] \cdot G_2 \cdot II_1$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_2^0 \cdot \overline{FD_2} \cdot II_2$			$+ [ASK'_1 + ASK_1^0] \cdot AK'_{d.4} \cdot AKIR^{DIV}] \cdot a'_1$		
		$+[$ ditto $] \cdot G_4 \cdot II_1$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_4^0 \cdot \overline{FD_4} \cdot IV_1$					
SHL → A_2	α	$[$ ditto $] \cdot G_2 \cdot II_2$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_2^0 \cdot \overline{FD_2} \cdot II_2$			$+ [\leftarrow$ ditto $\rightarrow] \cdot a'_2$		
		$+[$ ditto $] \cdot G_4 \cdot IV_2$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_4^0 \cdot \overline{FD_4} \cdot IV_2$					
SHL → A_3	α	$[$ ditto $] \cdot G_3 \cdot III$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_3^0 \cdot \overline{FD_3} \cdot III$			$+ [\leftarrow$ ditto $\rightarrow] \cdot a'_3$		
		$+[$ ditto $] \cdot G_4 \cdot IV_3$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_4^0 \cdot \overline{FD_4} \cdot IV_3$					
SHL → A_4	α	$[$ ditto $] \cdot G_4 \cdot IV$			$+[\leftarrow$ ditto $\rightarrow] \cdot Y_4^0 \cdot \overline{FD_4} \cdot IV$			$+ [\leftarrow$ ditto $\rightarrow] \cdot a'_4$		

Ref dwg 87805

Fig M-54 A REGISTER SHIFT RD LOGIC

SHIFT RIGHT IN B RD LOGIC

RD PULSE	RD Clock Pulse	NAB: Time Level	NAB: INST.	NAB: OTHER	CYB, CAB ; SCB, SAB: Time Levels	CYB, CAB ; SCB, SAB: INST.	CYB, CAB ; SCB, SAB: OTHER	MUL: Time Level	MUL: INST.	MUL: OTHER
SHR → B_1	α	$[AK'_{0,2} \cdot AKIR^{NAB}] \cdot Z'_1 \cdot I$ + $[(AK'_{0,3}+AK'_{0,4}) \cdot AKIR^{SHB}] \cdot Y'_1 \cdot \overline{FD_1} \cdot I$ + $[\leftarrow ditto \rightarrow] \cdot Z'_2 \cdot II_1$ + $[\leftarrow ditto \rightarrow] \cdot Y'_2 \cdot \overline{FD_2} \cdot II_1$ + $[\leftarrow ditto \rightarrow] \cdot Z'_4 \cdot IV_1$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_1$								+ $[AK'_{0,3} \cdot AKIR^{MUL}] \cdot a'_1$
SHR → B_2	α	$[\leftarrow ditto \rightarrow] \cdot Z'_2 \cdot II_2$ + $[\leftarrow ditto \rightarrow] \cdot Y'_2 \cdot \overline{FD_2} \cdot II_2$ + $[\leftarrow ditto \rightarrow] \cdot Z'_4 \cdot IV_2$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_2$								+ $[\leftarrow ditto \rightarrow] \cdot a'_2$
SHR → B_3	α	$[\leftarrow ditto \rightarrow] \cdot Z'_3 \cdot III$ + $[\leftarrow ditto \rightarrow] \cdot Y'_3 \cdot \overline{FD_3} \cdot III$ + $[\leftarrow ditto \rightarrow] \cdot Z'_4 \cdot IV_2$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_2$								+ $[\leftarrow ditto \rightarrow] \cdot a'_3$
SHR → B_4	β	$[\leftarrow ditto \rightarrow] \cdot Z'_4 \cdot IV$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV$								+ $[\leftarrow ditto \rightarrow] \cdot a'_4$

SHIFT LEFT IN B RD LOGIC

RD PULSE	RD Clock Pulse	NAB: Time Level	NAB: INST.	NAB: OTHER	CYB, CAB ; SCB, SAB: Time Levels	CYB, CAB ; SCB, SAB: INST	CYB, CAB ; SCB, SAB: OTHER	DIV: Time Level	DIV: INST	DIV: OTHER
SHL → B_1	α	$[AK'_{0,4} \cdot AKIR^{NAB}] \cdot G_1 \cdot I$ + $[(AK'_{0,3}+AK'_{0,4}) \cdot AKIR^{SHB}] \cdot Y'_1 \cdot \overline{FD_1} \cdot I$ + $[\leftarrow ditto \rightarrow] \cdot G_2 \cdot II_1$ + $[\leftarrow ditto \rightarrow] \cdot Y'_2 \cdot \overline{FD_2} \cdot II_1$ + $[\leftarrow ditto \rightarrow] \cdot G_4 \cdot IV_1$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_1$								+ $[AC'_{0,4} \cdot AKIR^{DIV}] \cdot a'_1$
SHL → B_2	α	$[\leftarrow ditto \rightarrow] \cdot G_2 \cdot II_2$ + $[\leftarrow ditto \rightarrow] \cdot Y'_2 \cdot \overline{FD_2} \cdot II_2$ + $[\leftarrow ditto \rightarrow] \cdot G_4 \cdot IV_2$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_2$								+ $[\leftarrow ditto \rightarrow] \cdot a'_2$
SHL → B_3	α	$[\leftarrow ditto \rightarrow] \cdot G_3 \cdot III$ + $[\leftarrow ditto \rightarrow] \cdot Y'_3 \cdot \overline{FD_3} \cdot III$ + $[\leftarrow ditto \rightarrow] \cdot G_4 \cdot IV_3$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV_3$								+ $[\leftarrow ditto \rightarrow] \cdot a'_3$
SHL → B_4	β	+ $[\leftarrow ditto \rightarrow] \cdot G_4 \cdot IV$ + $[\leftarrow ditto \rightarrow] \cdot Y'_4 \cdot \overline{FD_4} \cdot IV$								+ $[\leftarrow ditto \rightarrow] \cdot a'_4$

Ref Drawing 87805

FIG. 14-35 B REGISTER SHIFT RD LOGIC

RD Pulse		Multiply Step RD Logic		
		Time Level	Inst.	Other
Mult. Step Sign →	$A_{1.9}$ $C_{1.9}$	$[AK_{0,3}' \cdot AKIR^{MUL}) \cdot (II_1 + \overline{IV}_1)$		
Mult. Step Sign →	$A_{2.9}$ $C_{2.9}$	(ditto) $\cdot \overline{IV}_2$		
Mult. Step Sign →	$A_{3.9}$ $C_{3.9}$	(ditto) $\cdot \overline{IV}_3$		
Mult. Step →	A_1 C_1	(ditto) $\cdot a_1'$		
Mult. Step →	A_2 C_2	(ditto) $\cdot a_2'$		
Mult. Step →	A_3 C_3	(ditto) $\cdot a_3'$		
Mult. Step →	A_4 C_3	(ditto) $\cdot a_4'$		

Fig. 14-36 Multiply-Step RD Logic

	Block Schematic Ref	Chapter 14 Ref
Mult Step → A, C	87808	
Mult Step Sign → $A_{1,9}$ & $C_{1,9}$	87808	
Roman Numerals		

Pulse		RD Logic
C →	C_1	$a_1' \cdot [(\ ) + (\ )] \cdot [(\ ) + (\ )]$
C →	C_2	$a_2' \cdot [$ ditto $]$
C →	C_3	$a_3' \cdot [$ ditto $]$
C →	C_4	$a_4' \cdot [$ ditto $]$

Fig 14-37 Complement C RD Logic

Ref Dwgs. 87808

RD PULSE	RD CLOCK PULSE	TIME LEVEL	CARRY RD LOGIC — INSTRUCTION: DIV	CARRY RD LOGIC — INSTRUCTION: $\overline{DIV}$	CARRY RD LOGIC — OTHER
$\lfloor CRY_{\rightarrow} \; A_1$	α	$(AK'_{0,8}) \cdot \{[AKIR^{DIV}] + [$	$\overline{A}_1' + \overline{A}_2' \cdot (f_1 + f_2) + \overline{A}_3' \cdot f_1 + \overline{A}_4' \cdot f_1]\} \cdot a_1'$		
$\lfloor CRY_{\rightarrow} \; A_2$	α	$(\text{ditto}) \cdot \{[$	$\text{ditto}] + [\overline{A}_1' \cdot (f_1 + f_2) + \overline{A}_2' + \overline{A}_3' \cdot (f_1 + f_3) + \overline{A}_4' \cdot (f_1 + f_3)]\} \cdot a_2'$		
$\lfloor CRY_{\rightarrow} \; A_3$	α	$(\text{ditto}) \cdot \{[$	$\text{ditto}] + [\overline{A}_1' \cdot f_1 + \overline{A}_2' \cdot (f_1 + f_3) + \overline{A}_3' + \overline{A}_4' \cdot (f_1 + f_2 + f_3)]\} \cdot a_3'$		
$\lfloor CRY_{\rightarrow} \; A_4$	α	$(\text{ditto}) \cdot \{[$	$\text{ditto}] + [\overline{A}_1' \cdot f_1 + \overline{A}_2' \cdot (f_1 + f_3) + \overline{A}_3' \cdot (f_1 + f_2 + f_3) + \overline{A}_4']\} \cdot a_4'$		

	BLOCK SCHEMATIC REF	CHAPTER 14 REF
$\lfloor CRY_{\rightarrow} \; A$	87815	
A_i'		
AK		
a_i		

FIG 14-38 CARRY RD LOGIC

PARTIAL ADD RD LOGIC

RD PULSE	RD CLOCK PULSE	MUL		DIV; DSA, ADD, SUB		DIV							
		TIME LEVEL • INST PAD Γ_1	OTHER	TIME LEVEL • INST PAD Γ_2	OTHER	TIME LEVEL • INST PAD ϕ_1	OTHER	TIME LEVEL • INST PAD ϕ_2	OTHER	TIME LEVEL • INST PAD ϕ_3	OTHER	TIME LEVEL • INST PAD ϕ_4	OTHER
$\lfloor$PAD$\rightarrow A_1, C_1$	β	$(\text{PAD}\,\Gamma_1)\cdot a_1'\cdot[B_{1,1}'$		$]+(\text{PAD}\,\Gamma_2)\cdot a_1'+$		$[\text{PAD}\,\phi_1\cdot \text{I}]+$		$[\text{PAD}\,\phi_2\cdot \text{II}_1]$				$+[\text{PAD}\,\phi_4\cdot \text{IV}_1]$	
$\lfloor$PAD$\rightarrow A_2, C_2$	β	$(\text{ditto})\cdot a_2'$	$[(B_{1,1}'\cdot f_1)+(B_{2,1}'\cdot f_3)+B_{3,1}'\cdot(f_2+f_4$	$]+(\text{ditto})\cdot a_2'$				$+[\text{PAD}\,\phi_2\cdot \text{II}_2]$				$+[\text{PAD}\,\phi_4\cdot \text{IV}_2]$	
$\lfloor$PAD$\rightarrow A_3, C_3$	β	$(\text{ditto})\cdot a_3'$	$[B_{1,1}'\cdot(f_1+f_2)+B_{2,1}'\cdot(f_3+f_4)$	$]+(\text{ditto})\cdot a_3'$						$+[\text{PAD}\,\phi_3\cdot \text{III}_3]+$		$[\text{PAD}\,\phi_4\cdot \text{IV}_3]$	
$\lfloor$PAD$\rightarrow A_4, C_4$	β	$(\text{ditto})\cdot a_4'$	$[(B_{1,1}'\cdot f_1)+(B_{2,1}'\cdot f_3)+(B_{3,1}'\cdot f_2)+(B_{4,1}'\cdot f_4)]$	$+(\text{ditto})\cdot a_4'$								$+[\text{PAD}\,\phi_4\cdot \text{IV}_4]$	

WHERE:

$$\text{PAD}\,\Gamma_1 = [(AK_{B,2}' + AK_{B,3}')\cdot AKIR^{MUL}]$$

$$\text{PAD}\,\Gamma_2 = [AK_{B,2}'\cdot AKIR^{DIV}]+[AK_{B,3}'\cdot AKIR^{DSA+ADD}]$$

$$\text{PAD}\,\phi_1 = AK_{B,9}'\cdot[A_{1,9}^{\circ}+(ASK_1'+ASK_1^{\circ})]\cdot AKIR^{DIV}$$

$$\text{PAD}\,\phi_2 = AK_{B,9}'\cdot[A_{2,9}^{\circ}+(ASK_1'+ASK_1^{\circ})]\cdot AKIR^{DIV}$$

$$\text{PAD}\,\phi_3 = AK_{B,9}'\cdot[A_{3,9}^{\circ}+(ASK_1'+ASK_1^{\circ})]\cdot AKIR^{DIV}$$

$$\text{PAD}\,\phi_4 = AK_{B,9}'\cdot[A_{4,9}^{\circ}+(ASK_1'+ASK_1^{\circ})]\cdot AKIR^{DIV}$$

	BLOCK SCHEMATIC REF	CHAPTER 14 REF
AKIR( )		
$\lfloor$PAD$\rightarrow A_i, C_i$	87808	
PAD Γ		
PAD ϕ		
RN		
AK		
ASK		

FIG. 14-39 PARTIAL ADD RD LOGIC

CLEAR A, B RD LOGIC

RD PULSE	RD CLOCK PULSE	MUL: TIME LEVEL	MUL: INST	MUL: OTHER	ITA, UNA LD, STORE (See Below)
$LO \rightarrow A_1$	α	$[AK'_{9.2} \cdot AKIR^{MUL}] \cdot a'_1$	+		$+ [LO \rightarrow A]$
$LO \rightarrow A_2$	α	[ditto] $\cdot a'_2$	+	[ditto]	
$LO \rightarrow A_3$	α	[ditto] $\cdot a'_3$	+	[ditto]	
$LO \rightarrow A_4$	α	[ditto] $\cdot a'_4$	+	[ditto]	
$LO \rightarrow B_1$	α				$[LO \rightarrow B]$
$LO \rightarrow B_2$	α				[ditto]
$LO \rightarrow B_3$	α				[ditto]
$LO \rightarrow B_4$	α				[ditto]

CLEAR C, D RD LOGIC

RD PULSE	RD CLOCK PULSE	ADD, SUB: TIME LEVEL	ADD, SUB: INST	ADD, SUB: OTHER	MUL, DIV: TIME LEVEL	MUL, DIV: INST	MUL, DIV: OTHER	LD, STORE (See Below)
$LO \rightarrow C_1$	α	$[AK'_{9.1} \cdot AKIR^{ADD}] \cdot a'_1$	+	$[(AK'_{9.1} + AK'_{9.8}) \cdot (AKIR^{MUL} + AKIR^{DIV}] \cdot a'_1$	+	$[LO \rightarrow C]$		
$LO \rightarrow C_2$	α	[ditto] $\cdot a'_2$	+	[ditto]	+	[ditto]		
$LO \rightarrow C_3$	α	[ditto] $\cdot a'_3$	+	[ditto]	+	[ditto]		
$LO \rightarrow C_4$	α	[ditto] $\cdot a'_4$	+	[ditto]	+	[ditto]		
$LO \rightarrow D_1, Y_1$	α						+	$[LO \rightarrow D]$
$LO \rightarrow D_2, Y_2$	α						+	[ditto]
$LO \rightarrow D_3, Y_3$	α						+	[ditto]
$LO \rightarrow D_4, Y_4$	α						+	[ditto]

WHERE: $LO \rightarrow A = QK^{22\alpha} \cdot [(QKIR^{ITA} + QKIR^{UNA}) + (QKN^{VFF} \cdot QKIR^{STORE}) \cdot VMD^{XX4}] + [(QK^{14\alpha} \cdot QKIR^{LD}) \cdot QKIR^{A}]$

$LO \rightarrow B = QK^{22\alpha} \cdot [$ (ditto) $\cdot VMD^{XX5}] + [($ $) \cdot QKIR^{B}]$

$LO \rightarrow C = QK^{22\alpha} \cdot [$ (ditto) $\cdot VMD^{XX6}] + [($ $) \cdot QKIR^{C})$

$LO \rightarrow D = QK^{22\alpha} \cdot [$ (ditto) $\cdot VMD^{XX7}] + [($ $) \cdot QKIR^{D}]$

	BLOCK SCHEMATIC REF	CHAPTER 14 REF
a'_i		
AK		
$AKIR^{(\ )}$		
$LO \rightarrow$ A,B,C,D	87812,67721	
$LO \leftarrow$ A,B,C,D		

Fig 14-40 CLEAR A, B, C, and D RD LOGIC

Z PULSE GATE LOGIC

Z FF INPUT	RD PULSE (See Below)	GATING LOGIC	RD PULSE (See Below)	GATING LOGIC	RD PULSE (See Below)	GATING LOGIC
0 → Z_1	[0 → Z_1]	+	[Reset Z_1 · ($D^{0}_{1.9} \cdot A^{0}_{1.9} + D'_{1.9} \cdot A'_{1.9}$)]	+	[$A_{1.9}$ Sign Z_1 · $A^{0}_{1.9}$]	
0 → Z_2	[0 → Z_2]		[Reset Z_2 · ($D^{0}_{2.9} \cdot A^{0}_{2.9} + D'_{2.9} \cdot A'_{2.9}$)]	+	[$A_{2.9}$ Sign Z_2 · $A^{0}_{2.9}$]	
0 → Z_3	[0 → Z_3]		[Reset Z_3 · ($D^{0}_{3.9} \cdot A^{0}_{3.9} + D'_{3.9} \cdot A'_{3.9}$)]	+	[$A_{3.9}$ Sign Z_3 · $A^{0}_{3.9}$]	
0 → Z_4	[0 → Z_4]		[Reset Z_4 · ($D^{0}_{4.9} \cdot A^{0}_{4.9} + D'_{4.9} \cdot A'_{4.9}$)]	+	[$A_{4.9}$ Sign Z_4 · $A^{0}_{4.9}$]	
1 → Z_1	[PAD Z_1 · ($D^{0}_{1.9} \cdot A^{0}_{1.9} + D'_{1.9} \cdot A'_{1.9}$)]			+	[$A_{1.9}$ Sign Z_1 · $A'_{1.9}$]	
1 → Z_2	[PAD Z_2 · ($D^{0}_{2.9} \cdot A^{0}_{2.9} + D'_{2.9} \cdot A'_{2.9}$)]			+	[$A_{2.9}$ Sign Z_2 · $A'_{2.9}$]	
1 → Z_3	[PAD Z_3 · ($D^{0}_{3.9} \cdot A^{0}_{4.9} + D'_{3.9} \cdot A'_{3.9}$)]			+	[$A_{3.9}$ Sign Z_3 · $A'_{3.9}$]	
1 → Z_4	[PAD Z_4 · ($D^{0}_{4.9} \cdot A^{0}_{4.9} + D'_{4.9} \cdot A'_{4.9}$)]			+	[$A_{4.9}$ Sign Z_4 · $A'_{4.9}$]	

	Block Schem Ref	Chap 14 Ref
Z Logic	8)809	
RN		
AK		
AKIR( )		

Fig 14-41
Z Pulse Gate & RD Logic

RD PULSE (See above)	Z RD LOGIC: RD Clock Pulse	ADD, SUB: Time Level	ADD, SUB: INST.	ADD, SUB: OTHER
PAD → Z_1	α	$[AK'_{d.3} \cdot AKIR^{ADD}]$	· I	
PAD → Z_2	α	[ditto]	· II	
PAD → Z_3	α	[ditto]	· III	
PAD → Z_4	α	[ditto]	· IV	

RD PULSE (See above)	Z RD LOGIC: RD Clock Pulse	ADD, SUB: Time Level	ADD, SUB: INST.	ADD, SUB: OTHER
Reset → Z_1	α	$[AK'_{d.9} \cdot AKIR^{ADD}]$	· I	
Reset → Z_2	α	[ditto]	· II	
Reset → Z_3	α	[ditto]	· III	
Reset → Z_4	α	[ditto]	· IV	

RD PULSE (See above)	Z RD LOGIC:	DIV: Time Levels	DIV: INST	DIV: OTHER	MUL: Time Level	MUL: INST.	MUL: OTHER
$A_{1.9}$ Sign → Z_1	α	$[(AK'_{d.1} + AK'_{d.11}) \cdot AKIR^{DIV}]$	· I +		$[AK'_{d.2} \cdot AKIR^{MUL}]$	· I	
$A_{2.9}$ Sign → Z_2	α	[ditto]	· II +		[ditto]	· II	
$A_{3.9}$ Sign → Z_3	α	[ditto]	· III +		[ditto]	· III	
$A_{4.9}$ Sign → Z_4	α	[ditto]	· IV +		[ditto]	· IV	

		ADD, SUB: Time Level	ADD, SUB: INST	ADD, SUB: OTHER	MUL: Time Levels	MUL: INST	MUL: OTHER	SCA, SAB: Time Level	SCA, SAB: INST	SCA, SAB: OTHER	NOA, NAB: Time Level	NOA, NAB: INST	NOA, NAB: OTHER
0 → Z_1	α	$[AK'_{d.1} \cdot AKIR^{ADD}]$	· I +		$[(AK'_{d.1} + AK'_{d.9}) \cdot AKIR^{MUL}]$	· I +		$[AK'_{d.3} \cdot (AKIR^{SCA} + AKIR^{SAB})]$	· I +		$[AK'_{d.3} \cdot AKIR^{NOR}]$	· I	
0 → Z_2	α	[ditto]	· II +		[ditto]	· II +		[ditto]	· II +		[ditto]	· II	
0 → Z_3	α	[ditto]	· III +		[ditto]	· III +		[ditto]	· III +		[ditto]	· III	
0 → Z_4	α	[ditto]	· IV +		[ditto]	· IV +		[ditto]	· IV +		[ditto]	· IV	

COMPLEMENT A RD LOGIC

RD Pulse	RD Clock Pulse	MUL			DIV						INS, ITA (See Below)
		Time Level	INST	OTHER	Time Level	INST	OTHER	Time Level	INST	OTHER	
LC→ A_1	α	$[(AK'_{0,11} \cdot AKIR^{MUL}) \cdot (Z'_1 \cdot Y'^{0}_1 + Z'^{0}_1 \cdot Y'_1) \cdot I]$			$+[(AK'_{0,9} \cdot AKIR^{DIV}) \cdot (Z'_1 Y'^{0}_1 + Z'^{0}_1 \cdot Y'_1) \cdot I]$			$+[(AK'_{0,1} \cdot AKIR^{DIV}) \cdot A^{0}_{1,9} \cdot I]$			
		$+[($ ditto $) \cdot (Z'_2 \cdot Y'^{0}_2 + Z'^{0}_2 \cdot Y'_2) \cdot II_1]$			$+[($ ditto $) \cdot (Z'_2 \cdot Y'^{0}_2 + Z'^{0}_2 \cdot Y'_2) \cdot II_1]$			$+[(\leftarrow$ ditto $\rightarrow) \cdot A^{0}_{2,9} \cdot II_1]$			$+[$ LC→ A $]$
		$+[($ ditto $) \cdot (Z'_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y'_4) \cdot IV_1]$			$+[($ ditto $) \cdot (Z'_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y'_4) \cdot IV_1]$			$+[(\leftarrow$ ditto $\rightarrow) \cdot A^{0}_{4,9} \cdot IV_1]$			
LC→ A_2	α	$[($ ditto $) \cdot (Z'_2 \cdot Y'^{0}_2 + Z'^{0}_2 \cdot Y'_2) \cdot II_2]$			$+[($ ditto $) \cdot (Z'_2 \cdot Y'^{0}_2 + Z'^{0}_2 \cdot Y'_2) \cdot II_2]$			$+[($ ditto $) \cdot A^{0}_{2,9} \cdot II_2]$			$+[$ ditto $]$
		$[($ ditto $) \cdot (Z'_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y'_4) \cdot IV_2]$			$+[($ ditto $) \cdot (Z'_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y'_4) \cdot IV_2]$			$+[($ ditto $) \cdot A^{0}_{4,9} \cdot IV_2]$			
LC→ A_3	α	$[($ ditto $) \cdot (Z'_3 \cdot Y'^{0}_3 + Z'^{0}_3 \cdot Y'_3) \cdot III]$			$+[($ ditto $) \cdot (Z'_3 \cdot Y'^{0}_3 + Z'^{0}_3 \cdot Y'_3) \cdot III]$			$+[($ ditto $) \cdot A^{0}_{3,9} \cdot III]$			$+[$ ditto $]$
		$[($ ditto $) \cdot (Z'_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y'_4) \cdot IV_3]$			$+[($ ditto $) \cdot (Z'_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y'_4) \cdot IV_3]$			$+[($ ditto $) \cdot A^{0}_{4,9} \cdot IV_3]$			
LC q_4	α	$[($ ditto $) \cdot (Z_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y_4) \cdot IV]$			$+[($ ditto $) \cdot (Z_4 \cdot Y'^{0}_4 + Z'^{0}_4 \cdot Y_4) \cdot IV]$			$+[($ ditto $) \cdot A^{0}_{4,9} \cdot IV]$			$+[$ ditto $]$

COMPLEMENT $A_{1,9}$ RD LOGIC

RD Pulse	RD Clock Pulse	SCA, SAB						NOA, NAB			
		Time Level	INST.	OTHER	Time Level	INST.	OTHER	Time Level	INST	OTHER	
LC→ $A_{1,9}$	α	$[(AK'_{0,2} \cdot AKIR^{SCA+SAB}) \cdot Y'^{0}_1 \cdot Z'_1 \cdot I]$			$+[(AK'_{0,3} \cdot AKIR^{SCA+SAB}) \cdot Y'_1 \cdot Z'_1 \cdot I]$			$+[(AK'_{0,2} \cdot AKIR^{NOR}) \cdot Z'_1 \cdot I]$			
LC→ $A_{2,9}$	α	$[($ ditto $) \cdot Y'^{0}_2 \cdot Z'_2 \cdot II]$			$+[($ ditto $) \cdot Y'_2 \cdot Z'_2 \cdot II]$			$+[($ ditto $) \cdot Z'_2 \cdot II]$			
LC→ $A_{3,9}$	α	$[($ ditto $) \cdot Y'^{0}_3 \cdot Z'_3 \cdot III]$			$+[($ ditto $) \cdot Y'_3 \cdot Z'_3 \cdot III]$			$+[($ ditto $) \cdot Z'_3 \cdot III]$			
LC→ $A_{4,9}$	α	$[($ ditto $) \cdot Y'^{0}_4 \cdot Z'_4 \cdot IV]$			$+[($ ditto $) \cdot Y'_4 \cdot Z'_4 \cdot IV]$			$+[($ ditto $) \cdot Z'_4 \cdot IV]$			

WHERE: LC→ $A = QK^{10\alpha} \cdot QKIR^{INS} + QK^{24} \cdot (QKIR^{INS+ITA})$

	Block Schem Ref	Chap M Ref
Comp A	67721, 87814	
AKIR( )		
RN		
AK		
QK		

FIG. 14-42 COMPLEMENT A RD LOGIC

COMPLEMENT B RD LOGIC

RD PULSE	RD CLOCK PULSE	MUL: TIME LEVEL	MUL: INST	MUL: OTHER	DIV: TIME LEVEL	DIV: INST	DIV: OTHER	INS (See Below)
LC→ B_1	ϕ_1	$[(AK'_{d,12} \cdot AKIR^{MUL}_2) \cdot B'_{1,9} \cdot I] + [(AK'_{3,9} \cdot AKIR^{MUL}_2) \cdot (Z'_1 \cdot Y_1^{\circ} + Z_1^{\circ} \cdot Y'_1) \cdot I] + [(AK'_{3,11} \cdot AKIR^{DIV}) \cdot Z_1^{\circ} \cdot I] + [(LC \rightarrow B_1) \cdot I]$ $[(\text{ditto}) \cdot B'_{2,9} \cdot II_1] + [(\text{ditto}) \cdot (Z'_2 \cdot Y_2^{\circ} + Z_2^{\circ} \cdot Y'_2) \cdot II_2] + [(\text{ditto}) \cdot Z_2^{\circ} \cdot II_1] + [(LC \rightarrow B_2) \cdot II_1] + [LC \rightarrow B]$ $[(\text{ditto}) \cdot B'_{4,9} \cdot IV] + [(\text{ditto}) \cdot (Z'_4 \cdot Y_4^{\circ} + Z_4^{\circ} \cdot Y'_4) \cdot IV_1] + [(\text{ditto}) \cdot Z_4^{\circ} \cdot IV_1] + [(LC \rightarrow B_4) \cdot IV_1]$						
LC→ B_2	ϕ_1	$[(\text{ditto}) \cdot B'_{2,9} \cdot II_2] + [(\text{ditto}) \cdot (Z'_2 \cdot Y_2^{\circ} + Z_2^{\circ} \cdot Y'_2) \cdot II_2] + [(\text{ditto}) \cdot Z_2^{\circ} \cdot II_2] + [(LC \rightarrow B_2) \cdot II_2]$ $[(\text{ditto}) \cdot B'_{4,9} \cdot IV_2] + [(\text{ditto}) \cdot (Z'_4 \cdot Y_4^{\circ} + Z_4^{\circ} \cdot Y'_4) \cdot IV_2] + [(\text{ditto}) Z_4^{\circ} \cdot IV_2] + [(LC \rightarrow B_4) \cdot IV_2]$						+ [ditto]
LC→ B_3	ϕ_2	$[(\text{ditto}) \cdot B'_{3,9} \cdot III] + [(\text{ditto}) \cdot (Z'_3 \cdot Y_3^{\circ} + Z_3^{\circ} \cdot Y'_3) \cdot III] + [(\text{ditto}) Z_3^{\circ} \cdot III] + [(LC \rightarrow B_3) \cdot III]$ $[(\text{ditto}) \cdot B'_{4,9} \cdot IV_3] + [(\text{ditto}) \cdot (Z'_4 \cdot Y_3^{\circ} + Z_4^{\circ} \cdot Y'_4) \cdot IV_3] + [(\text{ditto}) Z_4^{\circ} \cdot IV_3] + [(LC \rightarrow B_4) \cdot IV_3]$						+ [ditto]
LC→ B_4	ϕ_2	$[(\text{ditto}) \cdot B'_{4,9} \cdot IV] + [(\text{ditto}) \cdot (Z'_4 \cdot Y_4^{\circ} + Z_4^{\circ} \cdot Y'_4) \cdot IV] + [(\text{ditto}) Z_4^{\circ} \cdot IV] + [(LC \rightarrow B_4) \cdot IV] + [\text{ditto}]$						

WHERE:
- $LC \rightarrow B_1 = (AK'_{3,1} \cdot AKIR^{DIV}) \cdot A_{1,9}^{\circ}$
- $LC \rightarrow B_2 = (\text{ditto}) \cdot A_{2,9}^{\circ}$
- $LC \rightarrow B_3 = (\text{ditto}) \cdot A_{3,9}^{\circ}$
- $LC \rightarrow B_4 = (\text{ditto}) \cdot A_{4,9}^{\circ}$
- $LC \rightarrow B = (QK^{10\phi} + QK^{2\phi}) \cdot QKIR^{INS}$

Block Schem Ref 67721 — Chap 19 Ref

COMP B
AKIR( )
RN;
*
QK

FIG 19-13 COMPLEMENT B RD LOGIC

COMPLEMENT D RD LOGIC

		ADD, SUB, DIV, MUL; CYA, CYB CAB; SCA, SCB, SAB				CYA, CYB, CAB; SCA, SCB, SAB (See also Φ_1 & Φ_2)			NOA, NAB (See Below)
		Φ_1	Φ_2	Φ_3	Φ_4	TIME LEVEL	INST.	OTHER	
LC→ D_1	d	$\Phi_1 \cdot I$	$+ \Phi_2 \cdot II_1$		$+ \Phi_4 \cdot IV_1$				$+ (\quad) \cdot I$
LC→ D_2	d		$\Phi_2 \cdot II_2$		$+ \Phi_4 \cdot IV_2$	$+ AK'_{d.2} \cdot AKIR^{SH} \cdot Y_2^0 \cdot II$			$+ (\quad) \cdot II$
LC→ D_3	d			$+ \Phi_3 \cdot III$	$+ \Phi_4 \cdot IV_3$				$+ (\quad) \cdot III$
LC→ D_4	d				$+ \Phi_4 \cdot IV$	$+ AK'_{d.2} \cdot AKIR^{SH} \cdot Y_4^0 \cdot IV$			$+ (\quad) \cdot IV$

WHERE: $\Phi_1 = [(AK'_{d.9} \cdot AKIR^{ADD}) + (Y_1 \neq D_{1.9})' + [AK'_{d.2} \cdot AKIR^{SUB}] + [(AK'_{d.2} + AK'_{d.9}) \cdot AKIR^{MUL}] \cdot Y_1' + [(AK'_{d.2} + AK'_{d.9}) \cdot AKIR^{DIV}] \cdot (D_{1.9} = A_{1.9}) + [AK'_{d.11} \cdot AKIR^{DIV}] \cdot (Y_1 \neq D_{1.9}) \times AK'_{d.2} \cdot AKIR^{SH} \cdot Y_1^0$

$\Phi_2 = [$ ditto $] \cdot (Y_2 \neq D_{2.9}) + [$ ditto $] + [$ ditto $] \cdot Y_2' + [$ ditto $] \cdot (D_{2.9} = A_{2.9}) + [$ ditto $] \cdot (Y_2 \neq D_{2.9})$

$\Phi_3 = [$ ditto $] \cdot (Y_3 \neq D_{3.9}) + [$ ditto $] + [$ ditto $] \cdot Y_3' + [$ ditto $] \cdot (D_{3.9} = A_{3.9}) + [$ ditto $] \cdot (Y_3 \neq D_{3.9})$

$\Phi_4 = [$ ditto $] \cdot (Y_4 \neq D_{4.9}) + [$ ditto $] + [$ ditto $] \cdot Y_4' + [$ ditto $] \cdot (D_{4.9} = A_{4.9}) + [$ ditto $] \cdot (Y_4 \neq D_{4.9})$

$$(\quad) = \left\{ AK'_{d.3} + AK'_{d.4} \cdot [(ASK_6^0 \cdot ASK_7^0) + (\overline{IV} + \overline{6}_4) \cdot (\overline{III} + \overline{6}_3)] \cdot [(ASK_6^0 \cdot ASK_7^0) + (\overline{II} + \overline{6}_2) \cdot (\overline{I} + \overline{5}_1)] \right\} \cdot AKIR^{NOR}$$

	Block Schematic Ref	Chapter 14 Ref
Comp D	87806	
RN	87804	
AKIR( )		
AK	87803	
ASK		
G		

Fig. 14-44 Complement D RD Logic

RD Pulse	E→A,B,C,D RD Logic: RD Clock Pulse	E→A,B,C,D RD Logic: E→A,B,C,D Level
$E_1 \xrightarrow{1} A_1$	α	[E → A]
$E_2 \xrightarrow{1} A_2$	α	[ditto]
$E_3 \xrightarrow{1} A_3$	α	[ditto]
$E_4 \xrightarrow{1} A_4$	α	[ditto]
$E_1 \rightarrow B_1$	α	[E → B]
$E_2 \rightarrow B_2$	α	[ditto]
$E_3 \rightarrow B_3$	α	[ditto]
$E_4 \rightarrow B_4$	α	[ditto]
$E_1 \xrightarrow{1} C_1$	α	[E → C]
$E_2 \xrightarrow{1} C_2$	α	[ditto]
$E_3 \xrightarrow{1} C_3$	α	[ditto]
$E_4 \xrightarrow{1} C_4$	α	[ditto]
$F_1 \xrightarrow{1} D_1$	α	[E → D]
$E_2 \xrightarrow{1} D_2$	α	[ditto]
$E_3 \xrightarrow{1} D_3$	α	[ditto]
$E_4 \xrightarrow{1} D_4$	α	[ditto]

E → A, B, C, D LEVEL LOGIC

$E \rightarrow A = [QK^{21\alpha} \cdot QKIR^{LD}] \cdot QKIR^{A} + [QK^{23\alpha} \cdot QKIR^{ITA+UNA}] + [QK^{23\alpha} \cdot QKIR^{STORE\ QKM^{VFF}}] \cdot VMD^{XX4}$

$E \rightarrow B = [$ ditto $] \cdot QKIR^{B} + [$ ditto $] + [$ ditto $] \cdot VMD^{XX5}$

$E \rightarrow C = [$ ditto $] \cdot QKIR^{C} + [$ ditto $] \pm [$ ditto $] \cdot VMD^{XX6}$

$E \rightarrow D = [$ ditto $] \cdot QKIR^{D} + [$ ditto $] \pm [$ ditto $] \cdot VMD^{XX7}$

	Block Schematic Ref	Chap 14 Ref
$E_i \rightarrow A_i, B_i, C_i, D_i$	87813	
$F_i \rightarrow A, B, C, D$	67721	

Fig 14-45
E → A, B, C, D RD LOGIC

RD Pulse	A → B, & B → A RD LOGIC						
	RD Clock Pulse	DIV			MUL		
		Time Level	Inst.	Other	Time Level	Inst.	Other
$A_1 \rightarrow B_1$	α	$[AK'_{x,10} \cdot AKIR^{DIV}] \cdot a_1'$			$+ [AK'_{y,1} \cdot AKIR^{MUL}] \cdot a_1'$		
$B_1 \rightarrow A_1$	α	[ditto		] $\cdot a_1'$			
$A_2 \rightarrow B_2$	α	[ditto		] $\cdot a_2'$	+ [ditto		] $\cdot a_2'$
$B_2 \rightarrow A_2$	α	[ditto		] $\cdot a_2'$			
$A_3 \rightarrow B_3$	α	[ditto		] $\cdot a_3'$	+ [ditto		] $\cdot a_3'$
$B_3 \rightarrow A_3$	α	[ditto		] $\cdot a_3'$			
$A_4 \rightarrow B_4$	α	[ditto		] $\cdot a_4'$	+ [ditto		] $\cdot a_4'$
$B_4 \rightarrow A_4$	α	[ditto		] $\cdot a_4'$			

	Block Schematic Ref	Chapter 14 Ref
$A_i \rightarrow B_i$	87810	
$B_i \rightarrow A_i$	87810	
AK		

Fig 14-46
E → B & B → A RD LOGIC

CHAPTER 15

IN-OUT ELEMENT

TABLE OF CONTENTS

15-1 INTRODUCTION

15-2 IN-OUT ELEMENT BLOCK DIAGRAM

15-3 PHYSICAL LAYOUT OF THE IN-OUT ELEMENT

- 15-3.1 GENERAL DESCRIPTION
- 15-3.2 IN-OUT BUS
- 15-3.3 CABLE INTERCONNECTIONS BETWEEN CENTRAL COMPUTER AND IN-OUT SECTION
  - 15-3.3.1 SEQUENCE SELECTION CONTROL CABLES
- 15-3.4 CABLE INTERCONNECTIONS BETWEEN IN-OUT SECTION AND IN-OUT UNIT

15-4 TYPICAL IN-OUT UNIT

- 15-4.1 GENERAL DESCRIPTION
- 15-4.2 CONTROL FLIP-FLOPS
- 15-4.3 SYNCHRONIZER

15-5 TYPICAL SEQUENCE SWITCH

- 15-5.1 GENERAL DESCRIPTION
- 15-5.2 LOGICAL STRUCTURE
- 15-5.3 SEQUENCE SWITCH LOGIC

15-6 IN-OUT ELEMENT OPERATION CODES (TSD AND IOS)

- 15-6.1 GENERAL DESCRIPTION
- 15-6.2 TSD AND IOS TIME CONTROL SIGNALS
- 15-6.3 IOCM CONTROL LEVELS
- 15-6.4 IOS
  - 15-6.4.1 IOS TYPES
  - 15-6.4.2 IOS 3X XXX FLOW DIAGRAM
- 15-6.5 TSD
  - 15-6.5.1 TSD TRANSFER MODES
  - 15-6.5.2 TSD FLOW DIAGRAM FOR OUTPUT DEVICE
  - 15-6.5.3 TSD FLOW DIAGRAM FOR INPUT DEVICE

15-7 ALARMS

15-8 DESCRIPTION OF IN-OUT UNITS

- * (00) STARTOVER
- * (41) IO ALARM
- (42) TRAP
- (46) MAGNETIC TAPE
- (47) MISCELLANEOUS INPUTS
- (50) DATRAC
- (51) XEROX
- * (52) PETR
- (54) INTERVAL TIMER
- (55) LITE PEN

(60) DISPLAY NO. 1

(61) RANDOM NUMBER GENERATOR

* (63) PUNCH

(65) LINCOLN WRITER INPUT

(66) LINCOLN WRITER OUTPUT

(72) X-Y PLOTTER

* Discussions currently included in chapter.

LIST OF FIGURES

15-1 BLOCK DIAGRAM OF IN-OUT ELEMENT SHOWING INTERCONNECTIONS WITH CENTRAL COMPUTER

15-2 INTERCONNECTIONS BETWEEN SEQUENCE SWITCHES ON IN-OUT FRAME AND CENTRAL COMPUTER

15-3 INTERCONNECTIONS BETWEEN SEQUENCE SWITCHES ON IN-OUT FRAME AND INDIVIDUAL IN-OUT UNITS

15-4 INTERCONNECTIONS BETWEEN CENTRAL COMPUTER AND IN-OUT FRAME

15-5 TYPICAL SEQUENCE SWITCH

15-6 LOGICAL OPERATION OF INPUT MIXERS AND OUTPUT DISTRIBUTORS USED IN SEQUENCE SWITCH

15-7 SIMPLIFIED FLOW DIAGRAM FOR TSD AND IOS INSTRUCTIONS

15-8 IN-OUT TIME CONTROL LEVELS GENERATED IN CENTRAL COMPUTER

15-9 $IOCM^{(xx)}$ LEVELS AS A FUNCTION OF SELECTED SEQUENCE

15-10 IOS TYPES

15-11 IOS REPORT TABLE

15-12 SIGNIFICANCE OF $N^{(xx)}_{1.1 - 2.3}$ DURING IOS 3X XXX AND IOS 6X XXX

15-13 FLOW DIAGRAM FOR IOS 3X XXX WHEN REPORT BIT IS SET (CF^{1}_{1})

15-14 IOCM MODE LEVELS ASSOCIATED WITH EACH IN-OUT UNIT

15-15 TSD DATA TRANSFER TABLE

15-16 PETR READING INTO MEMORY BLOCK OF 36 BITS IN ASSEMBLY MODE

15-17 TSD FOR OUTPUT DEVICE

15-18 TSD FOR INPUT DEVICE

15-STARTOVER-1 STARTOVER SEQUENCE

15-IO ALARM-1 BLOCK DIAGRAM OF IO ALARM SYSTEM

15-IO ALARM-2 ALARM DATA REPORTED BY TSD

15-PETR-1 PETR TAPE TRANSPORT SYSTEM

15-PETR-2 PETR BLOCK DIAGRAM

15-PETR-3 PETR MOTION CONTROL LOGIC

15-PUNCH-2 PUNCH BLOCK DIAGRAM

15-PUNCH-2 HIGH SPEED PUNCH SYNCHRONIZATION

CHAPTER 15
IN-OUT ELEMENT

15-1 INTRODUCTION

The In-Out Element provides a communication link between the external world and the computer. Before events occurring in the external world can communicate with the computer they must be synchronized, i.e., brought in step with the periodicity of the computer. This synchronizing occurs in the In-Out Element.

All of the in-out data transmission devices, as well as certain asynchronous events such as in-out alarms and those events initiated by certain manual controls, communicate with the computer via the In-Out Element. However, some asynchronous events, such as most of those initiated at the control console, communicate directly with the Control Element.

The In-Out Element must provide the necessary logic for accommodating the special operating characteristics of many different devices. It is this accommodation requirement that makes the In-Out Element such a complex communication link.

As described in Chapter 3, the solid state circuitry used in the In-Out Element is slower than the circuitry in the central computer. The logic reflects this, and also the fact that level transitions and 0.4 microsecond levels are used to generate pulses.

The chapter begins with a block diagram discussion of the In-Out Element. This discussion establishes the basic components and communication paths.

Because the in-out frame has a rather complex physical structure, the layout of the In-Out Element is discussed in some detail.

A discussion of a typical In-Out unit and sequence switch then follows, since these components are found in all sequences and have common characteristics regardless in which sequence they are found.

The two In-Out OP codes, IOS and TSD, are then discussed. The logic and communication paths these OP codes use are discussed in detail.

The chapter concludes with a logical description of the individual In-Out units.

15-2 IN-OUT ELEMENT BLOCK DIAGRAM

Fig. 15-1 shows a block diagram of the In-Out Element. The diagram indicates the communication paths between the In-Out Element and the central computer.

The In-Out Element includes:

1) Input-Output Devices
2) In-Out Bus
3) Sequence Switches

Since only one In-Out device can communicate with the central computer at a time, it is possible to have all the devices use a shared In-Out Bus. An elaborate switching arrangement is required to properly connect the selected In-Out device to the bus. This switching is taken care of by the individual sequence switches associated with each In-Out device.

Once an In-Out device is properly connected to the In-Out Bus, it can communicate with the central computer. While it is difficult to make a sharp distinction, generally the communication will involve transmitting either data information or control information. Each bus between the central computer and the In-Out Element is used to transmit a specific type of information. For example:

IOBM Bus - This bus is used to transmit data information from the In-Out Buffers to the E register. It is also used to "report" the control state of the In-Out devices to the E register.

E Bus - This bus is used to transmit data information from the E register to the In-Out Buffers.

N Bus - This bus is used to transmit mode control information from the N register to the In-Out devices.

IOCM Bus - This bus is used to transmit control information from the In-Out devices to the Control Element in the central computer.

Hi Speed Bus - This bus is used to transmit control information from the Control Element to the In-Out Devices.

Note that these buses are "shared" by all the sequences. In addition to these shared buses, there are individual cables that run between each sequence switch and the Sequence Selector in the central computer. These cables transmit the "Raise Flag" signals to the Sequence Selector from the In-Out devices. They also transmit KD^{i} and ND^{i} sequence selection levels to the sequence switches during TSD's and IOS's, respectively. (The function of $KD^{41(0)}$ will be discussed later.)

15-3 PHYSICAL LAYOUT OF THE IN-OUT ELEMENT

15-3.1 GENERAL DESCRIPTION. An elaborate cabling and busing arrangement is required to interconnect all the parts of the In-Out Element to the central computer. This results in the In-Out system being physically complex as well as logically complicated.

The heart of the cabling and busing arrangement is the In-Out Frame. This frame or section is located at the far left end of the central computer structure (facing the front). The frame contains an open wire bus structure and all the sequence switches associated with the various In-Out devices. Flexible cables connect the In-Out section to the central computer and to the individual In-Out devices. Note that the individual In-Out devices themselves can consist of several chassis and control panels and require extensive interconnection. An example of this is the XEROX printer. Fig. 15-2, 15-3 and 15-4 show the cable interconnections for all the sequences.

15-3.2 IN-OUT BUS. There are two open-wire buses running horizontally down the In-Out section (see Fig. 15-2). The top bus has 100 wires and the bottom bus has 72 wires. Each bus has 37 positions where 104-pin receptacles and 75-pin receptacles, respectively, provide access to the buses for external connection. Since all 37 receptacle locations are logically identical, convenience alone determines which sequence is assigned to which location. Signals from the central computer are routed into the In-Out Bus at the left end of the In-Out section (looking at the section from the rear). Signals to the central computer are routed from the In-Out Bus at the right end of the In-Out section.

A 10-conductor video cable is jumpered between the T bars on the In-Out section occupied with sequence switches (see the jumper between Sequence 66 and 72 identified on Fig. 15-2). This video cable is the In-Out High-Speed Control Bus. It also runs horizontally down the In-Out section.

In addition to the cable terminating devices found at either end of the In-Out section, there are two logic nets involving $IOBM^{1}_{2.9}$ and $IOCM^{EIA + MISIND}$. These nets will be discussed later in the chapter.

15-3.3 CABLE INTERCONNECTIONS BETWEEN CENTRAL COMPUTER AND IN-OUT SECTION. These interconnections are shown in Figs. 15-2 and 15-4. The E cables from Section BC in the central computer terminate in cascodes and cable drivers on the In-Out section. The E cables consist of four 10-conductor video cables, which are used for transmitting $E^{1}_{4.9 - 1.1}$ (36 wires). These cables connect into the 100 wire bus.

The N cables from Section BC terminate in cascodes and cable drivers. These in turn connect into the same 100-wire bus that the E cables connected into. The N cables consist of two 10-conductor video cable for transmitting $N^{1}_{2.9 - 1.1}$ (18 wires). Although all 18 bits appear on the bus, only bits 2.3 - 1.1 are currently used. Note that both ZEROS and ONES appear on the N bus since the N cables are tied both directly and through inverting amplifiers to the N bus.

The IOCM cable to the Control Element in Section C of the central computer is driven by cascodes and cable drivers. These amplifiers are plugged into the 100-wire bus at the right-hand end. The E, N and IOCM cables all interconnect with the 100-wire bus.

The 75-wire bus transmits only $IOBM^{0,1}_{4.9 - 1.1}$ (72 wires). Information is carried back from the IOBM bus to the E register via section BC in the central computer. This is done using eight 10-conductor IOBM video cables.

15-3.3.1 SEQUENCE SELECTION CONTROL CABLES. Each sequence switch on the In-Out section is connected by an individual cable to the Sequence Selector in Section D of the central computer. These cables transmit the KD^{i}, ND^{i} and $\overline{\text{RAISE}} \rightarrow \text{FLAG}_{i}$ signals. In addition to these, a $KD^{41(0)}$ wire is found in each cable.

15-3.4 CABLE INTERCONNECTIONS BETWEEN IN-OUT SECTION AND IN-OUT UNITS. These interconnections are shown on Fig. 15-4. As the figure indicates, considerable variation occurs in the number of cables required for each sequence. Basically, the cables are used to connect the In-Out units to the associated sequence switches on the In-Out section.

The sequence switches themselves provide the link between the cables shown on Fig. 15-3 and the In-Out Bus itself.

15-4 TYPICAL IN-OUT UNIT

15-4.1 GENERAL DESCRIPTION. A typical In-Out Unit consists of the following types of devices:

1) Data Conversion Devices. For example, a photoelectric tape reader, a paper punch, etc.

2) Control Boxes. These boxes usually contain the In-Out buffer, the synchronizer, the control flip-flops and other special purpose circuitry.

3) Non-logical Controls. These are found in various chassis and control panels and include such items as power supplies, motor switches, etc.

15-4.2 CONTROL FLIP-FLOPS. These flip-flops determine the logical operation of the In-Out unit. The standard In-Out control flip-flops are:

C (Connect Flip-Flop). The In-Out unit is logically connected to the computer by setting the C flip-flop to ONE. This is done by an IOS "connect" instruction. C^1 gates the RAISE FLAG signals and, usually, certain other signals such as those caused by the Equipment Inability Alarm (EIA) flip-flop and the MISINDication flip-flop being set. Almost all In-Out units have a connect flip-flop.

ST (STatus Flip-Flop). When this flip-flop is set to ONE, it is permissible for the computer to perform a TSD in the unit's program sequence. The STatus flip-flop is set to ONE by the In-Out unit generating a "completion pulse", indicating that the unit is ready for another TSD. Almost all In-Out units have a STatus flip-flop.

EIA (Equipment Inability Alarm Flip-Flop). This flip-flop is set to ONE as a result of some difficulty such as overheating, low paper supply, etc., in the associated In-Out unit. Not all units have an EIA flip-flop.

MISIND (MISINDication Flip-Flop). This flip-flop is only found in free-running units such as the Magnetic Tape unit. When MISIND is set to ONE, it indicates that the unit is getting ahead of the computer, i.e., a line of data has been missed by the computer.

M (Maintenance Circuit). This is not a flip-flop, but rather a circuit which may include a manually operated maintenance switch. A "fail-safe" design has been incorporated in the circuit, so that an M (Maintenance) level is generated when any one of several conditions occur. Thus an M level is generated when the switch is open, the unit is not powered or the unit is physically disconnected. The transition of this level does not have to be synchronized.

15-4.3 SYNCHRONIZER. Normally when an In-Out unit has completed its cycle, it will generate a completion pulse. This pulse indicates that the unit is ready for the central computer to execute another TSD. These completion pulses occur asynchronously, since in many cases they occur as a function of the mechanical cycle of the data conversion device itself. The central computer synchronizes these asynchronous events by means of IOI clock pulses and a synchronizer. As we shall see later in the chapter, the output of the synchronizer becomes the synchronous RAISE FLAG signal that is transmitted to the central computer. The function of the synchronizer is to insure that the In-Out buffer state will not change until the central computer has completed its communication with the buffer.

15-5 TYPICAL SEQUENCE SWITCH

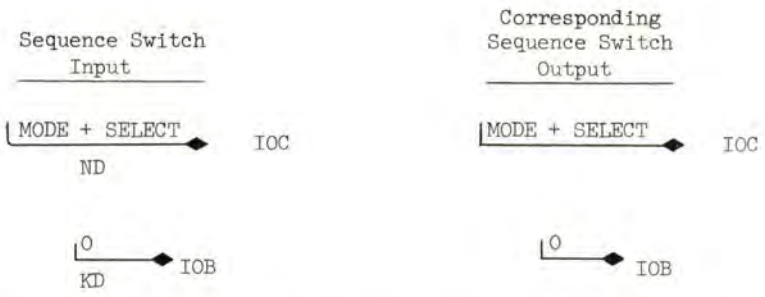
15-5.1 GENERAL DESCRIPTION. Since a large amount of information is transmitted between the In-Out Element and the central computer, it is important to understand how this information is routed to its correct destination. Most of this routing occurs in the sequence switches. The sequence switches provide a method of multiplexing a number of In-Out units onto a "shared" bus.

One side of the sequence switch is tied to the central computer. The other side of the sequence switch is tied to the associated In-Out unit. All the information on the "shared" buses will appear at the input to every sequence switch. It is then only necessary to provide a logical means for selecting the specific sequence switch that will pass the information through to the In-Out unit. In certain cases information will be transmitted right through the sequence switch without any gating occurring, e.g., this occurs in the case of IOI clock pulses, PRESET IOC levels, etc.

15-5.2 LOGICAL STRUCTURE. The sequence switches vary in complexity, depending on the nature of the specific sequence. However, there is much that is common to all the sequence switches. Fig. 15-5 is a block diagram of a typical sequence switch. It shows in composite form most of the communication that is possible between the central computer and the In-Out Element, and indicates the kinds of sequence switch gating which can occur.

The two standard "mixing" packages that are used in the sequence switches are the input mixers (IM) and the output distributors (OD). (The output mixers (OM) are logically similar to the output distributors.) The logical operation of these packages is shown in Fig. 15-6. These are level logic devices, although a 0.4 microsecond "pulse" is often used as one of the input signals.

15-5.3 SEQUENCE SWITCH LOGIC. No action can take place in the sequence switch during a TSD or IOS unless the sequence is selected by the KD^i or ND^i levels, respectively. The control levels gated by KD^i and ND^i are shown in Fig. 15-5. They include the control inputs transmitted to the sequence switch over the In-Out High Speed Control Bus and the IOCM levels transmitted from the In-Out units to the central computer. Note that in the case of the input signals, the levels retain their identity after passing through the selection logic nets, except that the KD's and ND's are dropped. For example:



The IOCM levels are expressed in terms of the control logic producing them. For example:

Sequence Switch Input	Corresponding Sequence Switch Output
$\overline{M} \cdot C^{1} \cdot ST^{1}$	$\overline{IOCM^{BB}}$

As mentioned before, certain control signals pass through the sequence switch without being gated by KD^{i} or ND^{i}. These are the IOI clock pulses, and the Stop Unit and Preset levels to the In-Out unit; and the RAISE FLAG pulses, and the $IOCM^{MATT}$, $IOCM^{MISIND}$ and $IOCM^{EIA}$ levels to the alarm sequence. The reset signal is gated by $\overline{M}$ so that, when the In-Out unit is in the maintenance state, the PRESET button on the console will not disturb the unit.

The specific logic used to connect the In-Out buffer to the E bus, the IOBM bus to the In-Out buffer and the In-Out control, and the N bus to the In-Out control will be discussed later in the chapter. However, the general features of this logic will be pointed out at this time.

Data can be transferred to and from the In-Out buffer in three possible modes: the NORMAL mode, the ASSEMBLY AND FORWARD mode and the ASSEMBLY AND REVERSE mode. Once the mode transfer is determined, the sequence switch is set up accordingly.

Data is always transmitted to the In-Out unit by first clearing the unit's buffer and then transferring ONES. For this reason, the output distributors have only one output wire. Data is usually transferred from the E bus to the In-Out unit in the form of a 0.4 microsecond ground level. The strobing is performed by a 0.4 microsecond negative (-3 volts) level.

Information is "jammed" into the E register from the IOBM bus. For this reason, input mixers are designed with two outputs. When the gating level is at ground, both outputs are at ground regardless of the input. When the gating level is negative and the input is at ground, one of the outputs will be at ground and one will be negative. If the input goes negative, the output wires will both reverse their signal levels. Hence data is visualized as represented by the negative output wires.

15-6 IN-OUT ELEMENT OPERATION CODES (TSD AND IOS)

15-6.1 GENERAL DESCRIPTION. Fig. 15-7 shows a simplified flow diagram of the TSD and IOS instructions.

During the execution of a program there is constant communication back and forth between the In-Out Element and the central computer. That this can occur in a variety of ways will become apparent in the discussion that follows.

Suppose that an IOS is executed which logically connects the i-th sequence. This can occur as follows. The IOS will cause the ND^i level to be generated in the Sequence Selector. The ND^i level will then allow the N register to communicate with the i-th control flip-flop via the i-th sequence switch. In this way the information in the N register during an IOS can be used to set the state of the i-th control flip-flops and in particular logically connect (C^1) the i-th sequence.

At certain specific times the Control Element will transmit IOI clock pulses to all the synchronizers in the different In-Out units. These IOI clock pulses "synchronize" the asynchronous events in the In-Out device that are used to indicate the devices have completed their cycle and are ready to communicate with the central computer. If a sequence is logically connected (C^1), the output of its synchronizer will be transmitted to the Sequence Selector in the form of a $\lfloor\overrightarrow{\text{RAISE}}$ FLAG pulse. At the same time, if an input device is involved, the output of the synchronizer will be used to gate data from the input device into the In-Out buffer in preparation for a TSD.

The Sequence Selector may receive "Raise Flag" signals from several sequences, since more than one sequence can be logically connected at a time. The Sequence Selector logic then determines which sequence has the highest priority. When all the necessary conditions are satisfied a change of sequence will occur to the selected sequence. At that time the KD^i level will be generated. When a TSD now occurs in the program of the current sequence, KD^i will allow TSD timing control information to connect the E register to the i-th In-Out buffer at the right logical time, if an output In-Out device is involved. If an input device is involved, the i-th In-Out buffer will be connected to the IOBM bus by the KD^i level and the TSD time control. The precise time at which the buffer content is read into the E register will then be determined by the IOBM $\xrightarrow{j}$ E pulse that is generated by an E register driver.

Note that the mode in which data is transmitted during the TSD is determined by an earlier IOS. The states of the i-th control flip-flops are transmitted back to the central computer in the form of IOCM levels. These IOCM levels set up the necessary logic for transmitting data in the mode called for by the IOS.

15-6.2 TSD AND IOS TIME CONTROL SIGNALS. The logic generating the TSD and IOS time control signals shown on Fig. 15-7 is tabulated on Fig. 15-8. The logic generating the IOI clock pulses and the |PRESET➔ IOC and STOP UNIT levels is also shown on Fig. 15-8.

The time signals originate in the Control Element of the central computer and are transmitted over the IO High Speed Control bus to the sequence switches. The IOI clock pulse and Stop Unit signals pass through all the sequence switches without any gating occurring there. The |PRESET➔ IOC level is gated through the sequence switches by an $\overline{M}$ ($\overline{\text{Maintenance}}$) level that indicates the sequence is not disabled. The rest of the time control signals enter only those sequence switches selected by ND^i during an IOS or KD^i during a TSD.

Note that normally two IOI clock pulses occur during each PK cycle. These clock pulses are inhibited, or prevented from occurring, if the QK cycle of a previous TSD overlaps the current TSD-PK cycle at $PK^{01\alpha}$ or at $PK^{12\alpha}$. An IOI clock pulse will also occur at $CSK^{11\alpha}$ during a delay synchronization cycle if the DSK cycle is to be followed by another DSK cycle, i.e., if the ($\overline{\text{|0➔ CSK}_4}$) at $CSK^{11\alpha}$ logic is satisfied. These IOI clock pulses are used to synchronize the asynchronous completion pulses in the In-Out devices and, in so doing, to generate the "Raise Flag" signals at the proper time.

During an output TSD, the In-Out buffer is cleared during the operand cycle by the $\overset{|0}{KD}$➔ IOB level at $QK^{18\alpha}$. The data in the E register is then transferred into the In-Out buffer by a $\overset{|DO}{KD}$➔ IOU level at $QK^{20\alpha}$. During an input TSD, the In-Out buffer is connected to the IOBM bus by the IOB $\overset{}{\underset{KD}{\longrightarrow}}$ E level. The data is then pulsed into the E register by RD logic at the E register. It should be noted that only the E bits corresponding to In-Out buffer bits are affected by this strobe. The other E bits are left undisturbed. During an IOS, the control levels generated depend on which of several possible IOS instructions is being executed, i.e., on the value of $N_{2.6 - 2.4}$. If the IOS is to do anything to an In-Out device, the bits must have the value 011 or 110. The IOS time control level then generated will be $\overset{|\text{MODE + SELECT}}{ND}$➔ IOC. The $\overset{|\text{SELECT}}{ND}$➔ IOC level, or its inverse ($\overline{\overset{|\text{SELECT}}{ND}\text{➔ IOC}}$), gates the |MODE + SELECT➔ IOC level at the In-Out unit in such a way as to distinguish whether a "mode" or "select" operation is involved. Note that this gating is significant only at those In-Out units which have subunits (currently this only includes the magnetic tape sequence). For all other In-Out units no distinction is made between $N^{011}_{2.6 - 2.8}$ and $N^{110}_{2.6 - 2.8}$. The $\overset{|0}{ND}$➔ C level is used by the "disconnect" IOS to clear C to ZERO. Note, that, by definition,

$$IOB \xrightarrow[KD]{} E = QKIR^{TSD} \cdot EB^{1}$$

$$IOC \xrightarrow[ND]{} E = \overline{IOB \xrightarrow[KD]{} E}$$

Hence,

$$IOC \xrightarrow[ND]{} E = \overline{QKIR^{TSD}} + EB^{0}$$

These levels essentially determine whether the sequence switch is to be set up to execute an IOS or a TSD. The logic is such that the $IOC \xrightarrow[]{} E$ level is always present, except during the QK cycle of a TSD when EB^{1}. In the TSD case, the $IOB \xrightarrow[KD]{} E$ level is generated. Thus the sequence switch is biased towards performing an IOS rather than a TSD.

The $\overset{PRESET}{\longrightarrow}$ IOC level is initiated by the PRESET pushbutton on the console. This level sets all the In-Out control flip-flops to a prescribed state, which in turn, in effect, places each In-Out unit in a predetermined state. The preset state of the In-Out unit and its associated control flip-flops will vary from sequence to sequence, but generally EIA, C, and MISIND will be cleared. The STatus flip-flop will be set to ONE for an output unit, and cleared to ZERO for an input unit. Normally, the In-Out unit itself will also be stopped.

15-6.3 IOCM CONTROL LEVELS. As shown in Fig. 15-7, the IOCM levels are used to inform the central computer of the state of the In-Out unit. The central computer reaches in to command the state of the In-Out unit by means of the N bus; the IOCM bus feeds back to the central computer the In-Out unit's actual state at any moment.

The interpretation of the IOCM levels is, to an extent, a function of the sequence selected. Note that if there is no sequence switch for the sequence selected by the Sequence Selector, all the wires on the IOCM bus will float at ground. A ground level indicates a "not" condition; for example, ground = $\overline{IOCM^{BB}}$ = buffer not busy. If a sequence switch is provided for the sequence selected, three possibilities exist: (1) the level is tied to ground; (2) the level is tied to -3 volts; or (3) the level depends on the state of the In-Out control.

Fig. 15-9 tabulates the IOCM levels for all the sequences. The IOCM levels and the logic that generates them are described below:

$\underline{IOCM^{BB}\ (Buffer\ Busy)}$. In most sequences, this level is generated by C^{0} + ST^{0} (or, in some sequences, by $M + C^{0} + ST^{0}$). ST^{0} indicates that the In-Out buffer is being used by the In-Out unit or that the In-Out unit is in some transient state and should not be disturbed.

IOCM^{MISIND} (Misindication). In free-running input devices, $C^1 \cdot MISIND \cdot \overline{M}$ will generate this level.

IOCM^{EIA} (Equipment Inability Alarm). In sequences that have an EIA flip-flop, $\overline{M} \cdot C^1 \cdot EIA^1$ generates this level.

IOCM^{NORMAL} (note that $\overline{NORMAL}$ = Assembly). For sequences that operate in a single mode, this level (or its negation) is prewired in the sequence switch. In the sequences that can operate in more than one mode, the IOCM^{NORMAL} level is determined by the state of the mode control flip-flops.

When the In-Out unit is in the NORMAL mode, data is transferred in "blocks" during a TSD, that is, adjacent bits in the In-Out buffer correspond to a block of adjacent bits in the E register. In the Exchange Element, the data is under normal permuted activity control (normal configuration control, excluding sign extension).

If a TSD is performed in the ASSEMBLY mode, the In-Out buffer data is splayed when it is transferred into the E register. That is, if there are six bits in the buffer word, the bits will be spread out so that they correspond to every sixth bit in the E register. Similarly, if there are nine bits in the buffer word, the bits will be spread out to correspond to every fourth bit in the E register. When a TSD is performed in the ASSEMBLY mode, the Exchange Element is not under configuration control.

IOCM^{RIGHT} (note that $\overline{RIGHT}$ = Left). This is a level used by the Exchange Element in conjunction with the $\overline{IOCM^{NORMAL}}$ level to determine whether data in the E register will be shifted to the left or right into the M register during an assembly TSD. If the In-Out unit operates in the forward direction (REV^0), the $\overline{IOCM^{RIGHT}}$ level is generated; conversely, if the In-Out unit operates in the reverse direction (REV^1), the IOCM^{RIGHT} level is generated.

IOCM^{IN} (note that $\overline{IN}$ = Out). This level indicates whether the In-Out unit is an input or output device. Note that for all sequences, except magnetic tape, this level is prewired in the sequence switch. The IOCM^{IN} level is used in the Exchange Element as one of the conditions for gating IOBM into the E register during an input TSD instruction. The level is also involved in the E to M transfer logic.

IOCM^{MAINT} (Maintenance). This level is generated whenever the In-Out unit's maintenance switch is turned on, or the power is turned off. Note that the level is not generated synchronously.

$\overline{IOCM^{MISIND + EIA}}$. This level is generated in the In-Out frame or section by ORing all the EIA and MISIND levels from the In-Out units. The function of this level will be discussed in the section describing the In-Out Alarm Sequence.

Note that all of the above IOCM levels (or their converse) can be generated by each and every sequence, and that all of these levels are transmitted to the central computer at any given time only from the sequence selected by KD^i. However, the central computer may or may not make use of the levels. For example, during a normal TSD, no use is made of the $IOCM^{RIGHT}$ or $\overline{IOCM^{RIGHT}}$ levels.

15-6.4 IOS. This instruction is used to control and/or report on the state of the In-Out system, as well as to raise and lower flags in the Sequence Selector. It is one of the variations of the OPR instruction. The instruction has the following characteristics:

1) An IOS in any sequence can logically connect any other squence. For example, an IOS in the PETR Sequence (52) can connect (⌊1◆ C) the Lite Pen Sequence (55).

2) An IOS is always possible, i.e., the IOS is never prevented from occurring, except when the selected In-Out unit is in the MAINTenance state. An IOS 30,000 or 60,000 will cause an IOSAL in this case.

3) An IOS instruction is always one of three types: i.e., it either (a) affects the controls of an In-Out unit, (b) has no effect on any In-Out unit, but raises or lowers a flag in the Sequence Selector, or (c) has no control effect on either the In-Out Element or the central computer, but is used for reporting.

15-6.4.1 IOS TYPES. IOS is an instruction in which some of the instruction word bits are used in a special way. Fig. 15-10 shows how the content of the N register is interpreted during an IOS.

The OP code bits 000100 (04) specify an OPR instruction. $N^{00}_{2.8 - 2.7}$ indicates that an OPR^{IOS} instruction instead of an OPR^{AE} instruction is specified. Bits $N_{4.8 - 4.7}$ are not used at all. The hold and defer bits are interpreted in the usual way. The sequence selected by the IOS is decoded from the J bits. CF_5 (or $N_{4.8}$) is used as a "dismiss" bit, i.e., if it is a ONE then the instruction reports a dismiss.

CF_1 (or $N_{4.4}$) is a "report" bit. If CF_1 is a ONE, the state of certain In-Out control flip-flops is reported to the E register. Fig. 15-11 tabulates the specific report made to the E register for each of the sequences. For example, suppose that an IOS 40,000, specifying the Datrac Sequence (50), is performed with CF_1^1. Then the content of FS is placed in $E_{1.4}$; the value of $\overline{M} \cdot C^1 \cdot EIA^1$ in $E_{2.4}$; the content of C in $E_{2.6}$; etc. Note that the transfer of the contents of the control flip-flops to the E register is via the IOBM bus, and that, when IOBM is gated into E, the E register bits take on the same state as the corresponding IOBM bits. Here again the only E bits affected are those that receive a report.

The Y bits are used to specify the IOS type. (Note that the decision to dismiss or to report is independent of the IOS type.) Bits $N_{2.3 - 1.1}$ are used in only two of the eight basic IOS types determined by bits $N_{2.6 - 2.4}$.

The basic IOS types are:

IOS 00 000, 10 000 and 70 000. If these IOS types have CF_5^0 and CF_1^0, they become dummy instructions in which nothing happens, i.e., these IOS types can be used only for reporting.

IOS 20 000. This IOS type is used to logically disconnect the selected In-Out unit from the computer.

IOS 3X XXX. This IOS type is used to logically connect the selected In-Out unit to the computer and to specify the operating mode of the In-Out unit. (Bits $N_{2.3 - 1.1}$ specify the operating mode.) Fig. 15-12 tabulates the mode specified by the $N_{2.3 - 1.1}$ bits. For example, if the Punch Sequence (63) is selected and $N_{1.2}^1$, then the ASSemblY flip-flop in the punch unit will be set to ONE. If now a TSD is performed in the Punch Sequence, the data will be transferred in the "assembly" mode.

IOS 40 000. This IOS type is used to lower the flag of the specified sequence. It communicates directly with the Sequence Selector and has no effect on the In-Out Element.

IOS 50 000. This IOS type is similar to IOS 40 000, except that it raises the flag of the specified sequence.

IOS 6X XXX. This IOS type is used to select the subunit of a multiple unit sequence. Currently only the Magnetic-tape Sequence uses this instruction. Fig. 15-13 tabulates the magnetic-tape subunits selected by the N bits. Note that this IOS type does not specify the operating mode of the selected subunit. This must be done by an IOS 3X XXX. However, as noted earlier in the chapter, IOS 6X XXX is equivalent to 3X XXX in those sequences which do not specify subunits.

15-6.4.2 IOS 3X XXX FLOW DIAGRAM. Fig. 15-13 shows an over-all flow diagram for an IOS 3X XXX type instruction when the CF_1 report bit is a ONE. Note that certain of the N bits are used by the Control Element in the logic that generates the IOS timing control.

The report data is gated onto the IOBM bus by the IOC ——◆ E level. Note that this level occurs as soon as ND^i is decoded in the Sequence Selector, i.e., the logic that generates IOC ——◆ E does not include a time level. The report data is then gated into the E register by the IOBM —j→ E pulse at the same time that the "mode commands" are gated into the In-Out control flip-flops by the MODE + SELECT ——◆ IOC pulse, i.e., at PK^{26}.

Note also the fact that the IOC ——◆ E level is generated is sufficient to logically connect ($\overset{1}{\longrightarrow}$ C) the sequence, i.e., C is set independent of the content of N. Since $C^1 \cdot ST^1$ indicates that the In-Out buffer is not busy the STatus flip-flop is always set to ONE for an output unit and to ZERO for an input unit whenever the unit is logically connected. Note that it is the transition to C^1 ($<C^1>$) that sets or clears the STatus flip-flop.

If the maintenance switch is turned on (M), and either an IOS 30 XXX or an IOS 6X XXX is attempted, an IOSAL alarm will be generated at $PK^{24\alpha}$.

15-6.5 TSD. This instruction transfers data between the specified In-Out buffer and the selected Memory Element register. It is unlike the IOS instruction in the following respects:

1) The computer must perform the TSD in the sequence associated with the In-Out device into or out of which data is being transferred. This sequence is determined by the content of the K register.

2) If the In-Out unit selected for a TSD is not ready to receive or transmit data, the TSD is not executed. In this case, a "dismiss and wait" takes place. The central computer is informed of this condition by the "buffer busy" ($IOCM^{BB}$) level.

3) The activity occurring in the In-Out Element during a TSD must be synchronized with the central computer.

15-6.5.1 TSD TRANSFER MODES. One of the fundamental considerations in a TSD is the mode in which data is transferred. A summary of the modes for each sequence is given in Fig. 15-14. Most of the sequences transfer data in the NORMAL mode. The specific bits transferred in each sequence and in each mode are given on Fig. 15-15.

A TSD in the PETR Sequence (52) can cause a data transfer in the ASSEMBLY mode. This type of transfer is shown on Fig. 15-16. It is used to store a "block" of six 6-bit lines on the paper tape as one 36-bit word in memory by means of six successive TSD instructions. Basically this is accomplished by transferring the In-Out buffer word into the E register in a splayed form and then shifting the content of the E register one bit to the left during the transfer from E into the M register. In this manner, a series of six TSD's packs the six lines into one memory word. The timing of the transfers and the logic of the packing process are shown in Fig. 15-16.

The Punch Sequence (63) is much like the PETR Sequence in that data can be transferred in the ASSEMBLY mode. The logic involved is very similar to that for the PETR Sequence, except that the direction of data flow is reversed.

The Magnetic-Tape Sequence is unique in that it can transfer data in any of the six possible modes:

1) Data can be transferred into the computer in the NORMAL mode.

2) Data can be transferred into the computer in the ASSEMBLY mode with the tape traveling in the forward direction.

3) Data can be transferred into the computer in the ASSEMBLY mode with the tape traveling in the reverse direction.

4) All three modes above are also possible when data is transferred out from the computer.

15-6.5.2 TSD FLOW DIAGRAMS FOR OUTPUT DEVICE. Fig. 15-17 shows a flow diagram for a TSD when an output device is involved.

When the input device has completed the action requested by a TSD, an IOI clock pulse will synchronize the "completion" pulse. A RAISE FLAG will then be generated which will set the STatus flip-flop to ONE. The RAISE FLAG pulse will also raise the sequence's flag in the Sequence Selector.

When the Sequence Selector causes the central computer to change to the In-Out unit's sequence, KD^{i} will connect the In-Out unit to the In-Out bus for a data transfer. As soon as the K Decoder decodes the content of the K register, the IOCM mode levels and "buffer not busy" signal will be transmitted to the central computer.

The $\overline{IOCM^{BB}}$ level is generated, because the STatus flip-flop was set before the KD^{i} level was generated. The IOCM mode levels and the mode control flip-flops will set up the Exchange Element and the sequence switch, respectively, for the desired mode of data transfer.

The fact that this is an output device (that is, that the $\overline{IOCM^{IN}}$ level is generated) means that a clear In-Out buffer pulse will be generated at $QK^{18\alpha}$. The data in the E register is actually gated into the In-Out buffer by a $\llcorner^{DO}\!\!\rightarrow$ IOU level occurring at $QK^{20\alpha}$. If required, this $\llcorner^{DO}\!\!\rightarrow$ IOU level can also be used to initiate the actual data output conversion.

The $\llcorner^{DO}\!\!\rightarrow$ IOU pulse will also clear the STatus flip-flop to ZERO and, in so doing, will generate an $IOCM^{BB}$ (buffer busy) signal. This level is used to inhibit the PK counter and cause a "dismiss and wait". A "completion" pulse from the output device will later indicate that the TSD has been completed and that the unit is ready for another TSD by setting the STatus flip-flop to ONE again.

15-6.5.3 TSD FLOW DIAGRAM FOR INPUT DEVICE. Fig. 15-18 shows a flow diagram for a TSD when an input device is involved. The process illustrated is similar to that for an output device, except that the direction of data flow is reversed. In this case, data is transferred into the In-Out buffer from the input device when a completion pulse occurs. This pulse is synchronized by the IOI clock pulses to generate a RAISE FLAG signal. This RAISE FLAG signal sets the STatus flip-flop to ONE, thus generating an $\overline{IOCM^{BB}}$ (buffer not busy) level.

Note that the buffer is connected to the IOBM bus by the IOB ——◆ E level. There is no time level term in the logic that generates this level. Consequently, the In-Out buffer is tied to the bus for virtually the entire QK cycle of TSD's. The actual time gating of the data into the E register from the bus is performed by the IOBM —j→ E level. This gating occurs at $QK^{18\alpha}$. At $QK^{20\alpha}$, a DO ——◆ IOU pulse is generated which clears the STatus flip-flop and generates the $IOCM^{BB}$ level.

In a free-running device it is possible that a second word will arrive at the input buffer before the first word has been transferred into the computer. In this case, the second word will take the place of the first word and the first word will be lost. The MISIND (misindication) level will be generated in order to inform the computer and the operator of the lost line of data.

15-7 ALARMS

There are three alarm situations associated with the In-Out Element:

1) An IOSAL alarm will occur during an IOS if the M (maintenance) level is present. The logic for this alarm is shown in Fig. 15-13. Note that the alarm is synchronous and can only occur at $PK^{24\alpha}$ during an IOS instruction.

2) When a free-running device is being operated, it is possible for the device to request data transfers by TSD's faster than TSD's are performed by the computer. The magnetic tape, PETR and DATRAC units have this characteristic. Under these conditions, the In-Out unit's MISIND flip-flop will be set. The setting of MISIND is synchronized by the IOI clock pulses. The corresponding $IOCM^{MISIND}$ level generated may then cause a MISAL alarm.

3) In the third situation, an $IOCM^{EIA}$ or $IOCM^{MISIND}$ level can raise the flag of the In-Out Alarm. The In-Out Alarm Sequence program is not unique and depends on the particular way in which the programmer desires to handle these alarms.

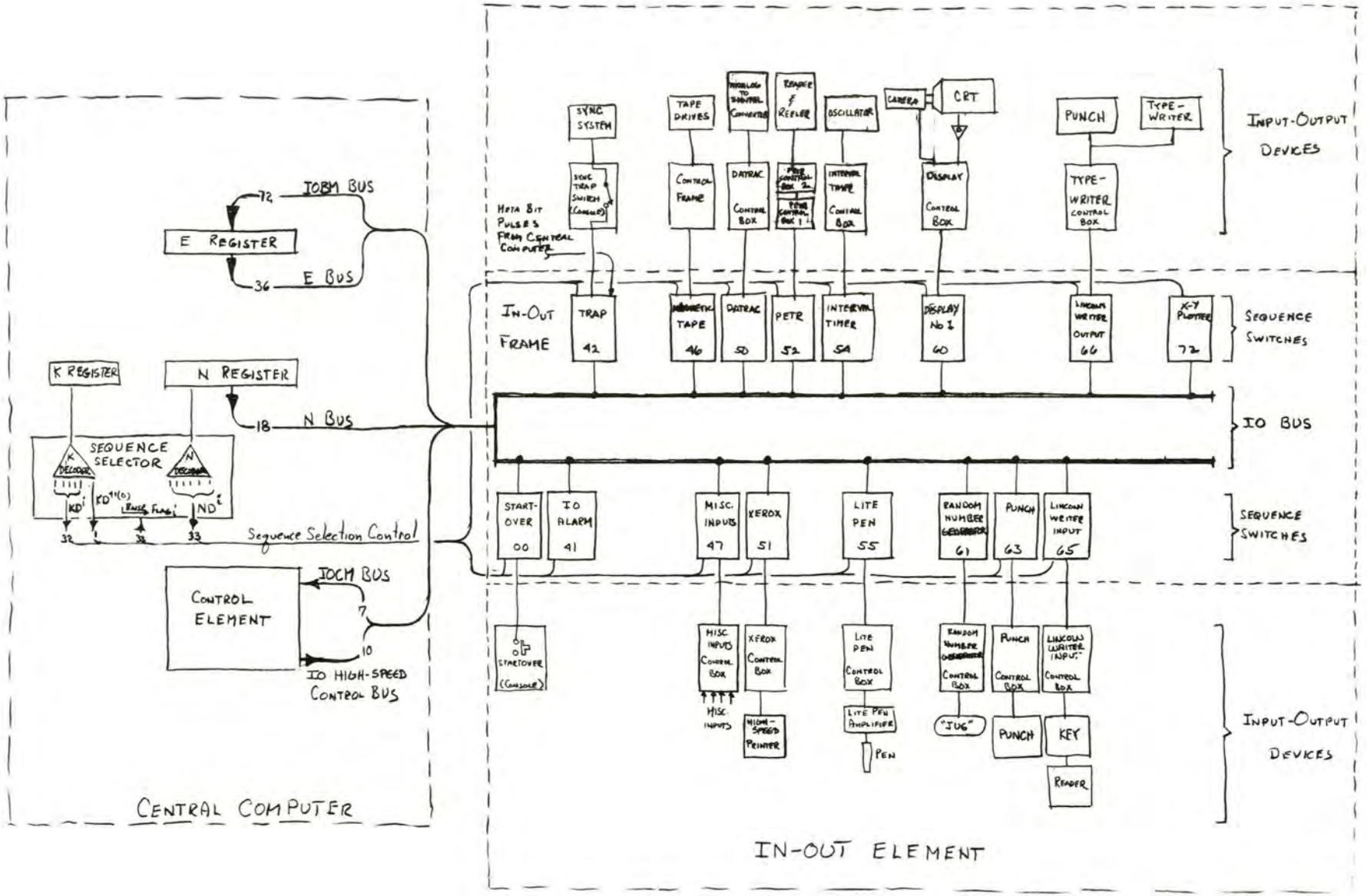
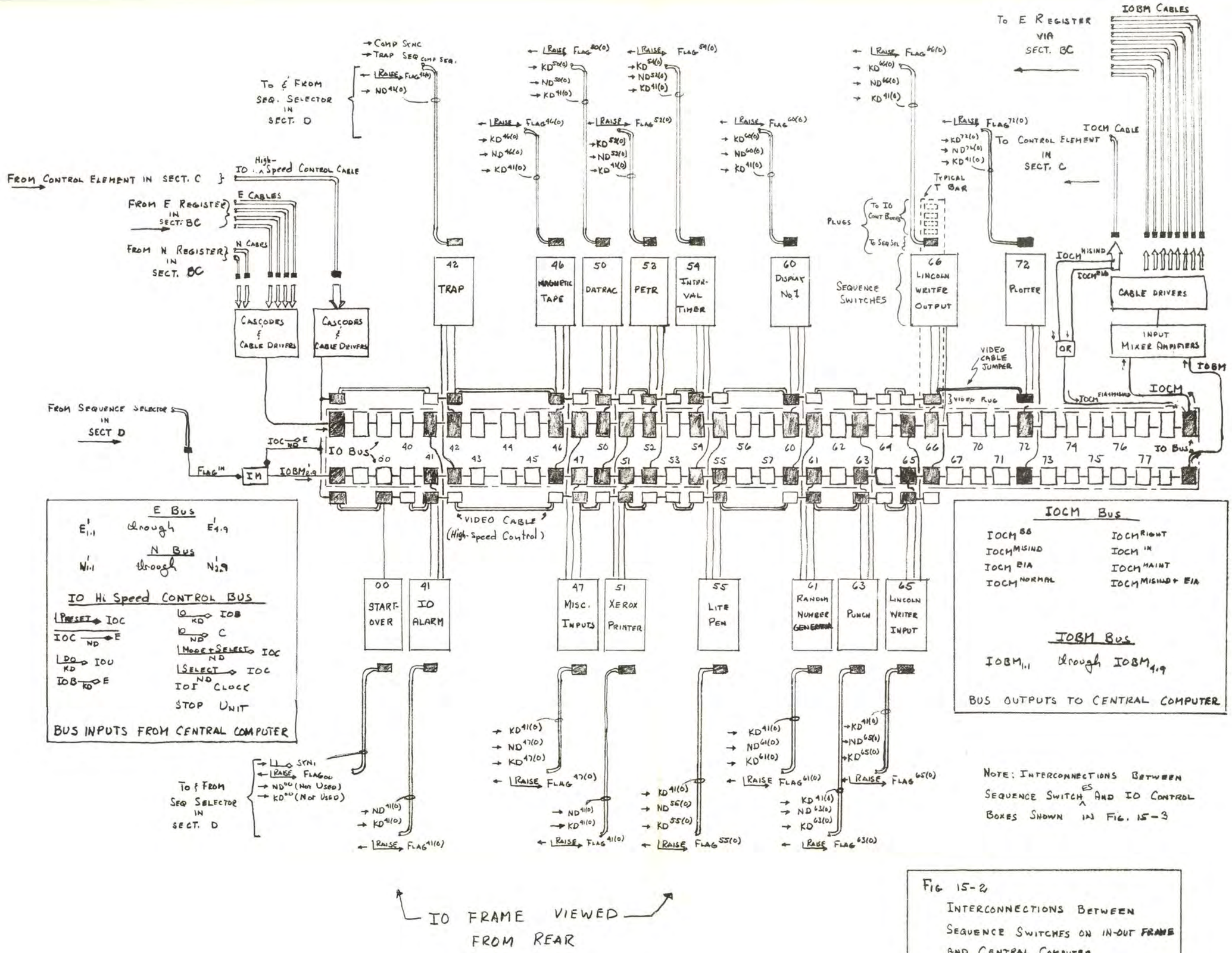
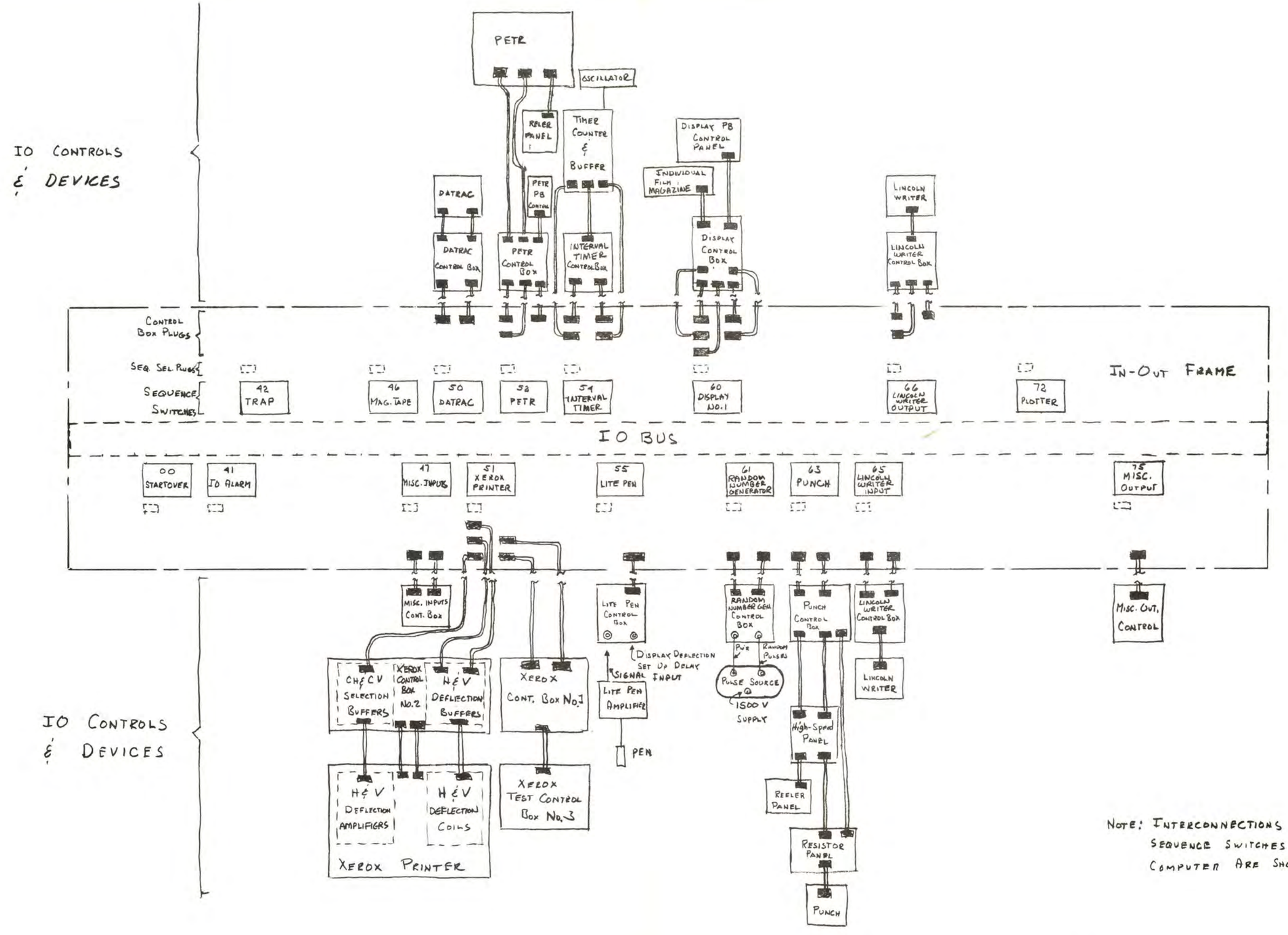


FIG. 15-1
BLOCK DIAGRAM OF IN-OUT ELEMENT SHOWING INTERCONNECTIONS WITH CENTRAL COMPUTER



Note: Interconnections Between Sequence Switches And IO Control Boxes Shown in Fig. 15-3

Fig 15-2
Interconnections Between Sequence Switches on IN-OUT Frame and Central Computer



NOTE: INTERCONNECTIONS BETWEEN SEQUENCE SWITCHES AND CENTRAL COMPUTER ARE SHOW IN FIG. 15-1.

FIG 15-3
INTERCONNECTIONS BETWEEN SEQUENCE SWITCHES ON IO FRAME AND INDIVIDUAL IN-OUT UNITS

IO CONTROLS & DEVICES

PETR

OSCILLATOR

RELER PANEL

TIMER COUNTER & BUFFER

DISPLAY PB CONTROL PANEL

INDIVIDUAL FILM MAGAZINE

DATRAC

PETR PB CONTROL

LINCOLN WRITER

DATRAC CONTROL BOX

PETR CONTROL BOX

INTERVAL TIMER CONTROL BOX

DISPLAY CONTROL BOX

LINCOLN WRITER CONTROL BOX

CONTROL BOX PLUGS

SEQ. SEL. PLUGS

SEQUENCE SWITCHES

42 TRAP

46 MAG. TAPE

50 DATRAC

52 PETR

54 INTERVAL TIMER

60 DISPLAY NO. 1

66 LINCOLN WRITER OUTPUT

72 PLOTTER

IN-OUT FRAME

IO BUS

00 STARTOVER

41 IO ALARM

47 MISC. INPUTS

51 XEROX PRINTER

55 LITE PEN

61 RANDOM NUMBER GENERATOR

63 PUNCH

65 LINCOLN WRITER INPUT

75 MISC. OUTPUT

MISC. INPUTS CONT. BOX

LITE PEN CONTROL BOX

RANDOM NUMBER GEN CONTROL BOX

PUNCH CONTROL BOX

LINCOLN WRITER CONTROL BOX

MISC. OUT. CONTROL

CH & CV SELECTION BUFFERS

XEROX CONTROL BOX NO. 2

H & V DEFLECTION BUFFERS

XEROX CONT. BOX NO. 1

DISPLAY DEFLECTION SET UP DELAY

SIGNAL INPUT

LITE PEN AMPLIFIER

PEN

PWR

RANDOM PULSES

PULSE SOURCE

1500 V SUPPLY

LINCOLN WRITER

High-Speed PANEL

REELER PANEL

RESISTOR PANEL

PUNCH

IO CONTROLS & DEVICES

H & V DEFLECTION AMPLIFIERS

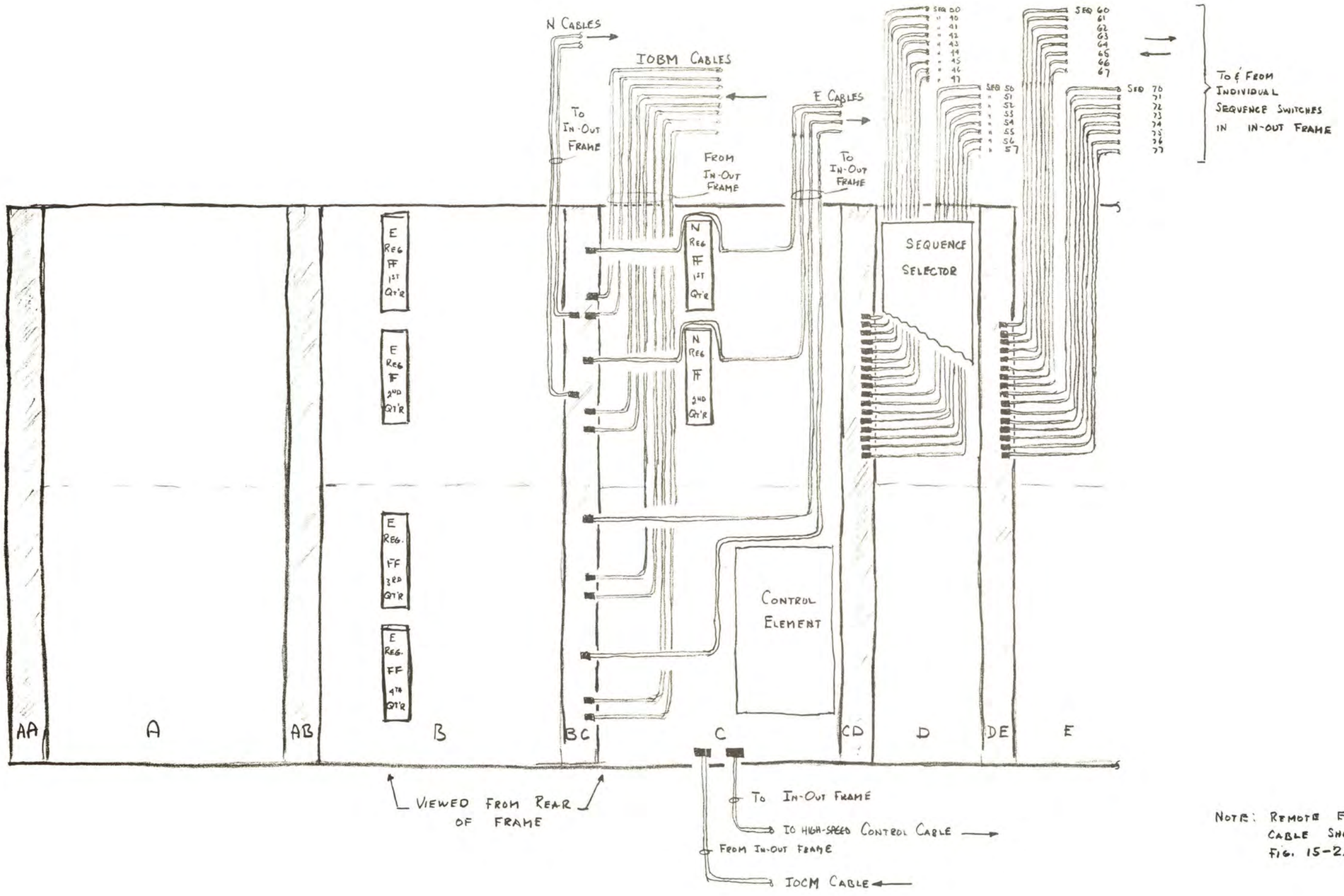
H & V DEFLECTION COILS

XEROX PRINTER

XEROX TEST CONTROL BOX NO. 3

NOTE: INTERCONNECTIONS BETWEEN SEQUENCE SWITCHES AND CENTRAL COMPUTER ARE SHOW IN FIG. 15-1.

FIG 15-3
INTERCONNECTIONS BETWEEN SEQUENCE SWITCHES ON IO FRAME AND INDIVIDUAL IN-OUT UNITS



NOTE: REMOTE END OF CABLE SHOWN IN FIG. 15-2.

FIG. 15-4
INTERCONNECTIONS BETWEEN CENTRAL COMPUTER AND IN-OUT FRAME

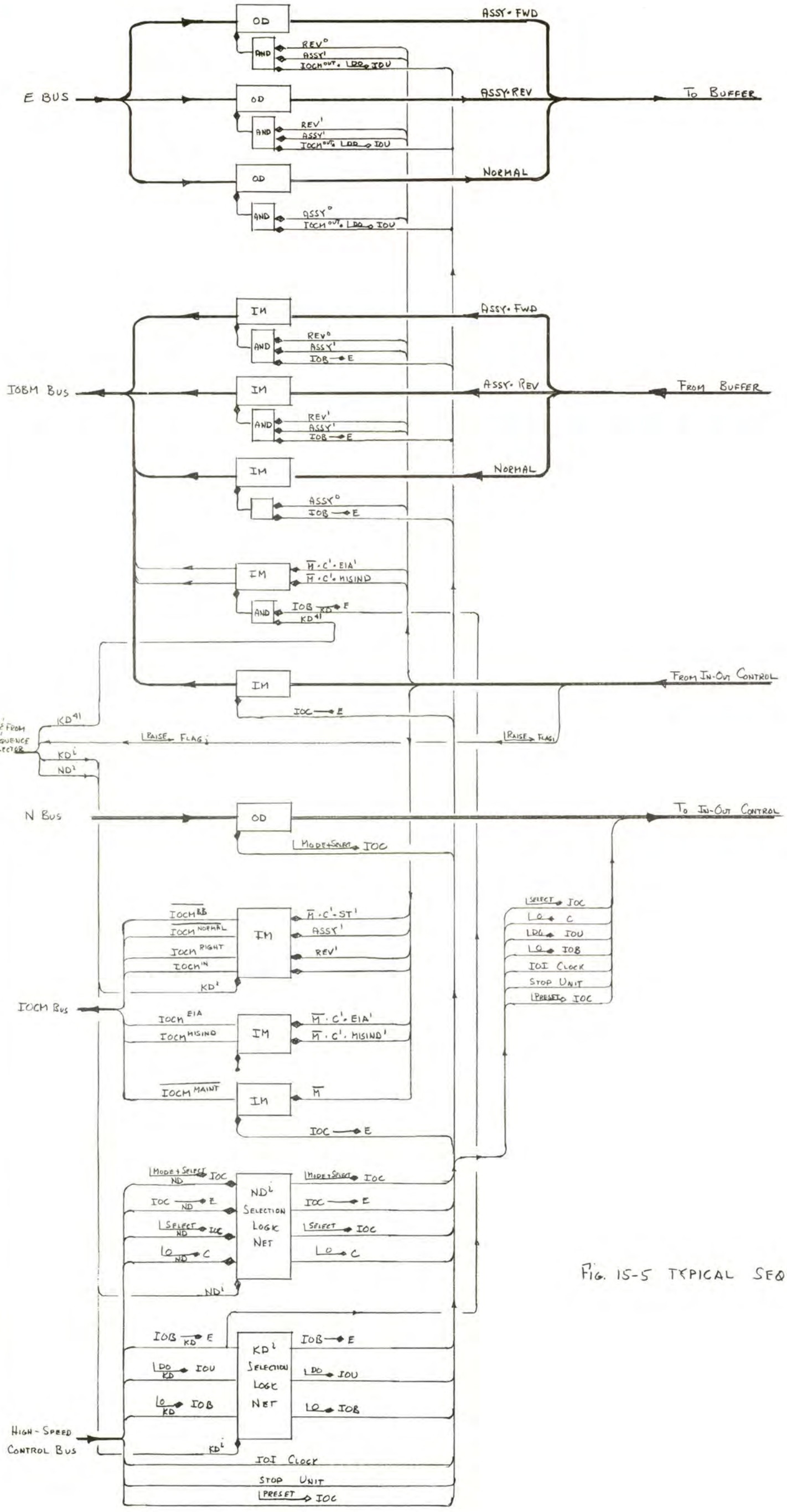


Fig. 15-5 Typical Sequence Switch

GRADE I FOR REFERENCE ONLY
GRADE II PRELIMINARY DESIGN
GRADE III FINAL DESIGN
GRADED BY: DATE:

ITEM | MATERIAL - DESCRIPTION | PART NO. | QTY.

LINCOLN LABORATORY DIV. 5
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON 73, MASS.

SCALE: DR. APPD.
ENG. CK. D-

CHG.	CN#	DATE	APPD.
-1			
-2			
-3			
-4			
-5			
-6			
-7			
-8			
-9			
-10			
-11			
-12			
-13			
-14			
-15			
-16			
-17			
-18			
-19			
-20			

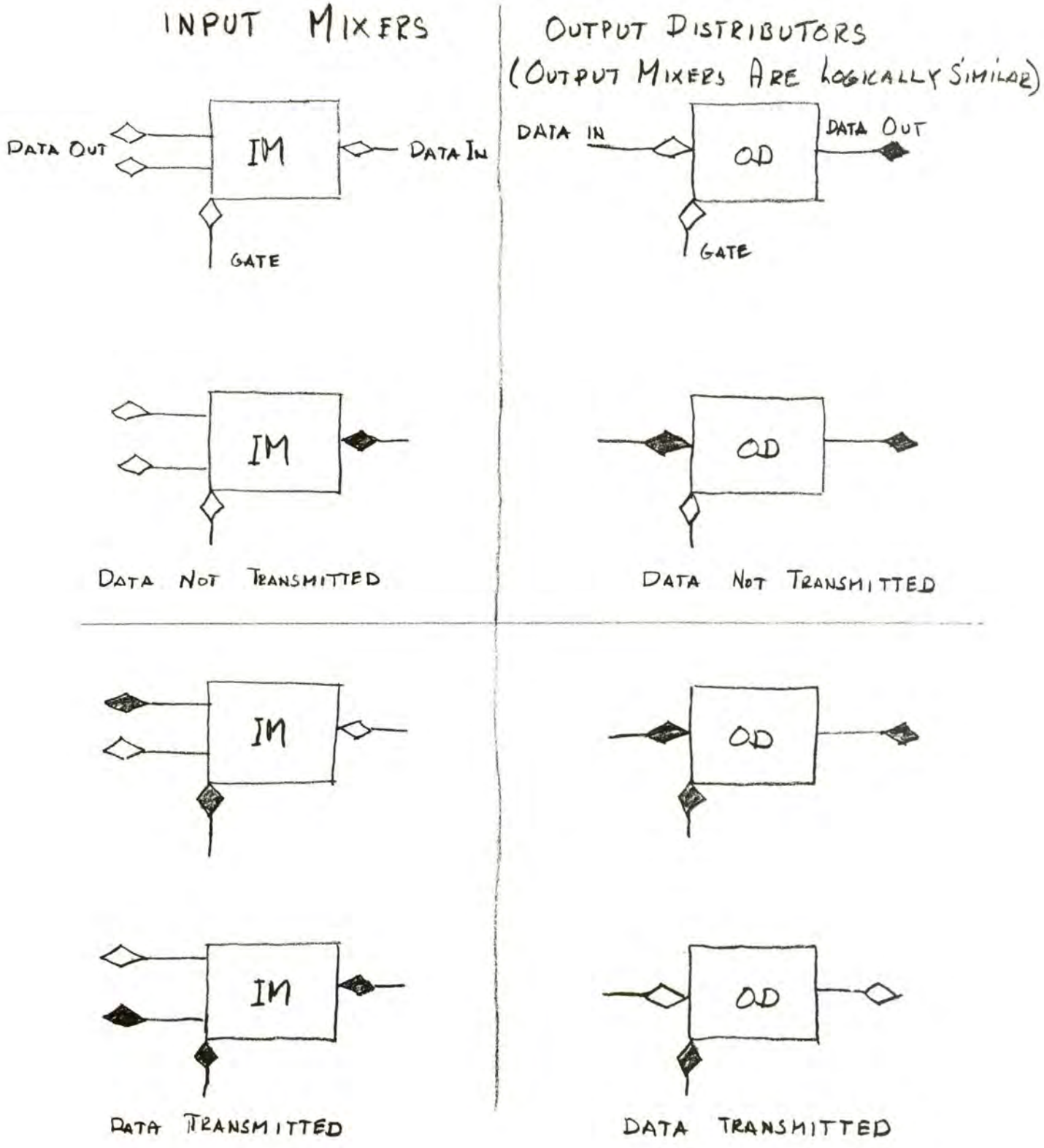


FIG. 15-6 LOGICAL OPERATION OF INPUT MIXERS AND OUTPUT DISTRIBUTORS USED IN SEQUENCE SWITCH

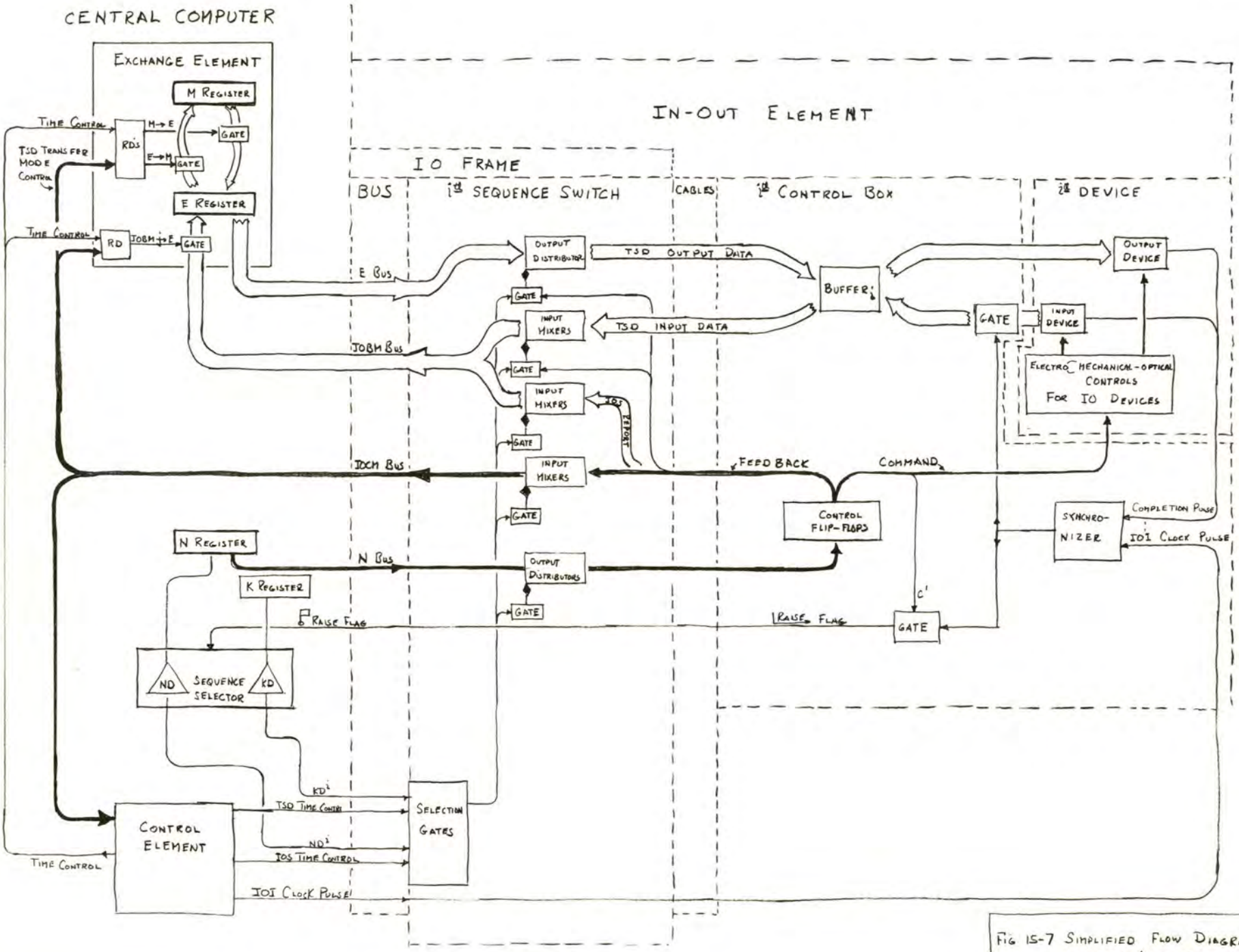


FIG 15-7 SIMPLIFIED FLOW DIAGRAM FOR TSD & IOS INSTRUCTIONS

TYPE SIGNAL	IO CONTROL LEVEL	IO CONTROL LEVEL LOGIC
IO SYNCHRONIZING SIGNAL	IOI CLOCK	$(PK^{ON} + PK^{12\alpha} + CSK^{11\alpha}) \cdot \overline{(LO \diamond CSK_4)} \cdot \overline{(QKIR^{TSD} \cdot QB')}$
TSD	IOB $\xrightarrow[KD]{}$ E	$QKIR^{TSD} \cdot EB'$
	LO $\xrightarrow[KD]{}$ IOB	$QKIR^{TSD} \cdot \overline{IOCM^{IN}} \cdot QK^{18\alpha}$
	DO $\xrightarrow[KD]{}$ IOU	$QKIR^{TSD} \cdot QK^{20\alpha}$
IOS	IOC $\xrightarrow[ND]{}$ E	$\overline{IOB \xrightarrow[KD]{} E}$
	MODE+SELECT $\xrightarrow[ND]{}$ IOC	$(N^{011}_{2.6-2.4} + N^{110}_{2.6-2.4}) \cdot PKIR^{IOS} \cdot PK^{26\alpha} \cdot \overline{IOCM^{MAINT}}$
	SELECT $\xrightarrow[ND]{}$ IOC	$N^{110}_{2.6-2.4} \cdot PKIR^{IOS}$
	O $\xrightarrow[ND]{}$ C	$N^{010}_{2.6-2.4} \cdot PKIR^{IOS} \cdot PK^{26\alpha}$
PUSH BUTTON & ALARM CONTROL	PRESET IOC	START-STOP CONTROL LOGIC
	STOP UNIT	

FIG 15-8 IN-OUT TIME CONTROL LEVELS GENERATED IN CENTRAL COMPUTER

Ground	No Sequence Switch	Startover	To Alarm	Trap	Mag. Tape	Misc. Inputs	Datrac	Xerox Printer	PETR	Interval Timer	Lite Pen	Display No 1	Random Number Generator	Punch	Lincoln Writer Input	Lincoln Writer Output	Plotter
Levels	40 53 64 43 56 67 44 57 70 45 62 71	00	41	42	46	47	50	51	52	54	55	60	61	63	65	66	72
$\overline{IOCM^{BB}}$ (tied to ground)	Floating ground	Floating ground	$IOCM^{BB}$ -3 ($\supset C^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'$)	Floating ground	$IOCM^{BB}$ -3V / $\overline{IOCM^{BB}}$ GRD	$IOCM^{BB}$ -3V ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	$IOCM^{BB}$ -3 ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	$IOCM^{BC}$ -3V ($\supset M+C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset \bar{M}\cdot C'\cdot ST'$)	$IOCM^{BB}$ -3 ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	$IOCM^{BB}$ -3V ($\supset C^0+(ST^0\cdot CT')$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot(ST'+CT^0)$)	$IOCM^{BB}$ -3 ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	$IOCM^{BB}$ -3V ($\supset M+C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset \bar{M}\cdot C'\cdot ST'$)	$IOCM^{BB}$ -3V ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	$IOCM^{BB}$ -3V ($\supset M+C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset \bar{M}\cdot C'\cdot ST'$)	$IOCM^{BB}$ -3V ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	$IOCM^{BB}$ -3V ($\supset C^0+ST^0$) / $\overline{IOCM^{BB}}$ GRD ($\supset C'\cdot ST'$)	— —
$\overline{IOCM^{MISIND}}$ (tied to ground)	Floating ground	Floating ground	Floating ground	Floating ground	$IOCM^{MISIND}$ -3V / $\overline{IOCM^{MISIND}}$ GRD	Floating ground	$IOCM^{MISIND}$ -3V ($\supset C'\cdot MISIND'$) / $\overline{IOCM^{MISIND}}$ GRD ($\supset C^0+MISIND^0$)	Floating ground	$IOCM^{MISIND}$ -3V ($\supset C'\cdot MISIND'$) / $\overline{IOCM^{MISIND}}$ GRD ($\supset C^0+MISIND^0$)	Floating ground	Floating ground	Tied to ground	Floating ground	Floating ground	Floating ground	Floating ground	
$\overline{IOCM^{EIA}}$ (tied to ground)	Floating ground	Floating ground	Floating ground	Floating ground	$IOCM^{EIA}$ -3V / $\overline{IOCM^{EIA}}$ GRD	Floating ground	Floating ground	$IOCM^{EIA}$ -3V ($\supset \bar{M}\cdot C'\cdot EIA'$) / $\overline{IOCM^{EIA}}$ GRD ($\supset M+C^0+EIA^0$)	Floating ground	Floating ground	Floating ground	$IOCM^{EIA}$ -3V ($\supset \bar{M}\cdot C'\cdot EIA'$) / $\overline{IOCM^{EIA}}$ GRD ($\supset M'+C^0+EIA^0$)	Floating ground	$IOCM^{EIA}$ -3V ($\supset \bar{M}\cdot C'\cdot EIA'$) / $\overline{IOCM^{EIA}}$ GRD ($\supset M+C^0+EIA^0$)	Floating ground	Floating ground	
$\overline{IOCM^{NORMAL}}$ (tied to ground)	Floating ground	Floating ground	$IOCM^{NORMAL}$ tied to -3V	Floating ground	$IOCM^{NORMAL}$ -3V / $\overline{IOCM^{NORMAL}}$ GRD	$IOCM^{NORMAL}$ tied to -3V	$IOCM^{NORMAL}$ tied to -3V	$IOCM^{NORMAL}$ tied to -3V	$IOCM^{NORMAL}$ -3V ($\supset ASSY^0$) / $\overline{IOCM^{NORMAL}}$ GRD ($\supset ASSY'$)	$IOCM^{NORMAL}$ tied to -3V	Floating ground	$IOCM^{NORMAL}$ tied to -3V	$IOCM^{NORMAL}$ tied to -3V	$IOCM^{NORMAL}$ -3V ($\supset ASSY^0$) / $\overline{IOCM^{NORMAL}}$ GRD ($\supset ASSY'$)	$IOCM^{NORMAL}$ tied to -3V	$IOCM^{NORMAL}$ tied to -3V	
$\overline{IOCM^{RIGHT}}$ (tied to ground)	Floating ground	Floating ground	Floating ground	Floating ground	$IOCM^{RIGHT}$ -3V / $\overline{IOCM^{RIGHT}}$ GRD	Floating ground	Floating ground	Floating ground	$IOCM^{RIGHT}$ -3V ($\supset REV'$) / $\overline{IOCM^{RIGHT}}$ GRD ($\supset REV^0$)	Floating ground	Floating ground	Floating ground	Floating ground	Floating ground	Floating ground	Floating ground	
$\overline{IOCM^{IN}}$ (tied to ground)	Floating ground	$IOCM^{IN}$ tied to -3V	$IOCM^{IN}$ tied to -3V	Floating ground	$IOCM^{IN}$ -3V / $\overline{IOCM^{IN}}$ GRD	$IOCM^{IN}$ tied to -3V	$IOCM^{IN}$ tied to -3V	Floating ground	$IOCM^{IN}$ tied to -3V	Floating ground	$IOCM^{IN}$ tied to -3V	Floating ground	$IOCM^{IN}$ tied to -3V	Floating ground	$IOCM^{IN}$ tied to -3V	Floating ground	
$\overline{IOCM^{MAINT.}}$ (tied to ground)	Floating ground	Floating ground	Floating ground	Floating ground	$IOCM^{MAINT}$ -3V / $\overline{IOCM^{MAINT}}$ GRD	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	Floating ground	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	$IOCM^{MAINT}$ -3V ($\supset M$) / $\overline{IOCM^{MAINT}}$ GRD ($\supset \bar{M}$)	— —
$\overline{IOCM^{MISIND+EIA}}$ (tied to ground)			$\overline{IOCM^{EIA}}$ + $\overline{IOCM^{MISIND}}$ (from other sequence switches)														

SYMBOLS:

- Floating ground ⊃ $\overline{IOCM^{(\ )}}$
- Tied to ground ⊃ $\overline{IOCM^{(\ )}}$
- Tied to -3 volts ⊃ $IOCM^{(\ )}$

NOTES:

① IOCM SUPERSCRIPT IS TRUE WHEN BUS IS DRIVEN TO -3 VOLTS,
e.g. $IOCM^{BB}$ = -3 volt ⊃ "BUFFER BUSY"
$\overline{IOCM^{BB}}$ = GRD ⊃ "BUFFER NOT BUSY"

② STATE OF BUS IS GIVEN WHEN SEQUENCE SWITCH IS SELECTED BY KD'. SELECTED SEQUENCE WILL ALWAYS DETERMINE BUS STATE

③ WHERE BUS IS SINGLE-VALUED WHEN SEQUENCE IS SELECTED, BUS IS "TIED" TO -3 volts OR "FLOATED" AT GROUND

④ WHERE BUS IS DOUBLE-VALUED WHEN SEQUENCE IS SELECTED, BUS VALUE [illegible]

FIG 15-9
$IOCM^{(\ )}$ LEVELS AS A FUNCTION OF SELECTED SEQUENCE

IOS TYPES	IOS FUNCTION					N REGISTER										
	CF_1 ($=N_{4.4}$) REPORT BIT		j BITS	y BITS		H	CF			OP CODE	j		SPECIAL USE	y		
	CF_1'	CF_1^0		$N_{23.4-2.4}$	$N_{23-1.1}$	49	48	47 45	44	43 38	37 32	31 30	29 28	27 26	25 24	23 12
						1	1	3	1	6	6	1	2	3		12
IOS 0 0 0 0 0 IOS 1 0 0 0 0 IOS 7 0 0 0 0	STATE OF IO CONTROL FF's PRIOR TO CURRENT IOS STORED IN E REG.	NO REPORT	SPECIFIES SEQUENCE SELECTED	NOT USED	NOT USED	X	X			000100	XXXXXX	X	00	000		000000000000
IOS 2 0 0 0 0	D.H0	D.H0	D.H0	DISCONNECTS (IO ⊘ C) IO UNIT. PHYSICALLY STOPS UNIT IF IT IS FREE-RUNNING	NOT USED	X	X			000100	XXXXXX	X	00	010		000000000000
IOS 3 X X X X	D.H0	D.H0	D.H0	CONNECTS (IO ⊘ C) IO UNIT.	SPECIFIES OPERATING MODE OF IO UNIT. SEE FIG 15-12	X	X			000100	XXXXXX	X	00	011		XXXXXXXXXXXX
IOS 4 0 0 0 0	D.H0	D.H0	D.H0	LOWERS FLAG IN SEQUENCE SELECTOR OF SPECIFIED SEQUENCE. HAS NO EFFECT ON IO ELEMENT.	NOT USED	X	X			000100	XXXXXX	X	00	100		000000000000
IOS 5 0 0 0 0	D.H0	D.H0	D.H0	RAISES FLAG IN SEQUENCE SELECTOR OF SPECIFIED SEQUENCE. HAS NO EFFECT ON IO ELEMENT	NOT USED	X	X			000100	XXXXXX	X	00	101		000000000000
IOS 6 X X X X	D.H0	D.H0	D.H0	SELECTS ONE OF SEVERAL SIMILAR UNITS. CURRENTLY EFFECTIVE ONLY W/ MAG. TAPE SEQUENCE.	SPECIFIES SELECTED SUB UNIT SEE FIG 15-12	X	X			000100	XXXXXX	X	00	110		XXXXXXXXXXXX

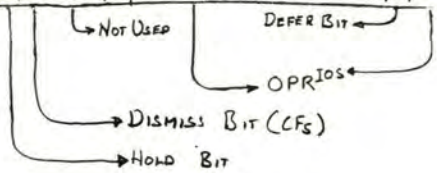


FIG 15-10
IOS TYPES

	IO ALARM 41	TRAP 42	MAG TAPE 46	MISC. INPUTS 47	DATRAC 50	XEROX PRINTER 51	PETR 52	INTERVAL TIMER 54	LITE PEN 55	DISPLAY NO. 1 60	DISPLAY NO. 2 60	RANDOM NUMBER GENERATOR 61	PUNCH 63	LINCOLN WRITER INPUT 65	LINCOLN WRITER OUTPUT 66
$IOBM'_{1.1}$		TNI'	Spare												
$IOBM'_{1.2}$		TND'	ASSY'				ASSY'						ASSY'		
$IOBM'_{1.3}$		TM'	REV'				REV'			CAMERA'			7TH HOLE		
$IOBM'_{1.4}$		Spare	SP_0			FS'				$INTENSIFY'_2$					
$IOBM'_{1.5}$			SP_1							$INTENSIFY'_1$					
$IOBM'_{1.6}$			SP_2												
$IOBM'_{1.7}$			F_0				CLUTCH'	COUNT'		SQ_M'					
$IOBM'_{1.8}$			F_1					(RAISE FLAG)'		SQ_H'					
$IOBM'_{1.9}$			F_2												
$IOBM'_{2.1}$			SBA'												
$IOBM'_{2.2}$			RS'												
$IOBM'_{2.3}$															
$IOBM'_{2.4}$			$\overline{M} \cdot C' \cdot MISIND$		$C' \cdot MISIND'$		$C' \cdot MISIND$								
$IOBM'_{2.5}$			$\overline{M} \cdot C' \cdot EIA'$			$\overline{M} \cdot C' \cdot EIA'$				$\overline{M} \cdot C' \cdot EIA'$	$\overline{M} \cdot C' \cdot EIA'$		$\overline{M} \cdot C' \cdot EIA'$		
$IOBM'_{2.6}$	C'		C' →												
$IOBM'_{2.7}$			M' →												
$IOBM'_{2.8}$			ST' →												
$IOBM'_{2.9}$	FLAG' →														
$IOBM'_{3.1}$															
$IOBM'_{3.2}$															
$IOBM'_{3.3}$															
$IOBM'_{3.4}$															
$IOBM'_{3.5}$															
$IOBM'_{3.6}$															
$IOBM'_{3.7}$			UBD												
$IOBM'_{3.8}$			TOS'												
$IOBM'_{3.9}$			TOF'												
$IOBM'_{4.1}$			REF'												
$IOBM'_{4.2}$			SBOK'												
$IOBM'_{4.3}$			SLD'												
$IOBM'_{4.4}$			BM												
$IOBM'_{4.5}$			UAB												
$IOBM'_{4.6}$			UNAV												
$IOBM'_{4.7}$			$\overline{\text{FORWARD}}$												
$IOBM'_{4.8}$			FEOT'												
$IOBM'_{4.9}$			REOT'												

NOTE: The following logic will gate the state of certain In-Out control levels into the E Register

$$PKIR_{CF_1'} \cdot PKIR^{IOS} \cdot PK^{2ax} \supset IOBM \rightarrow E$$

(a) the significance of the bits in the E Register during an IOS REPORT (for the sequence selected by the IOS) is given by the table above.

Fig. 15-11
IOS REPORT TABLE

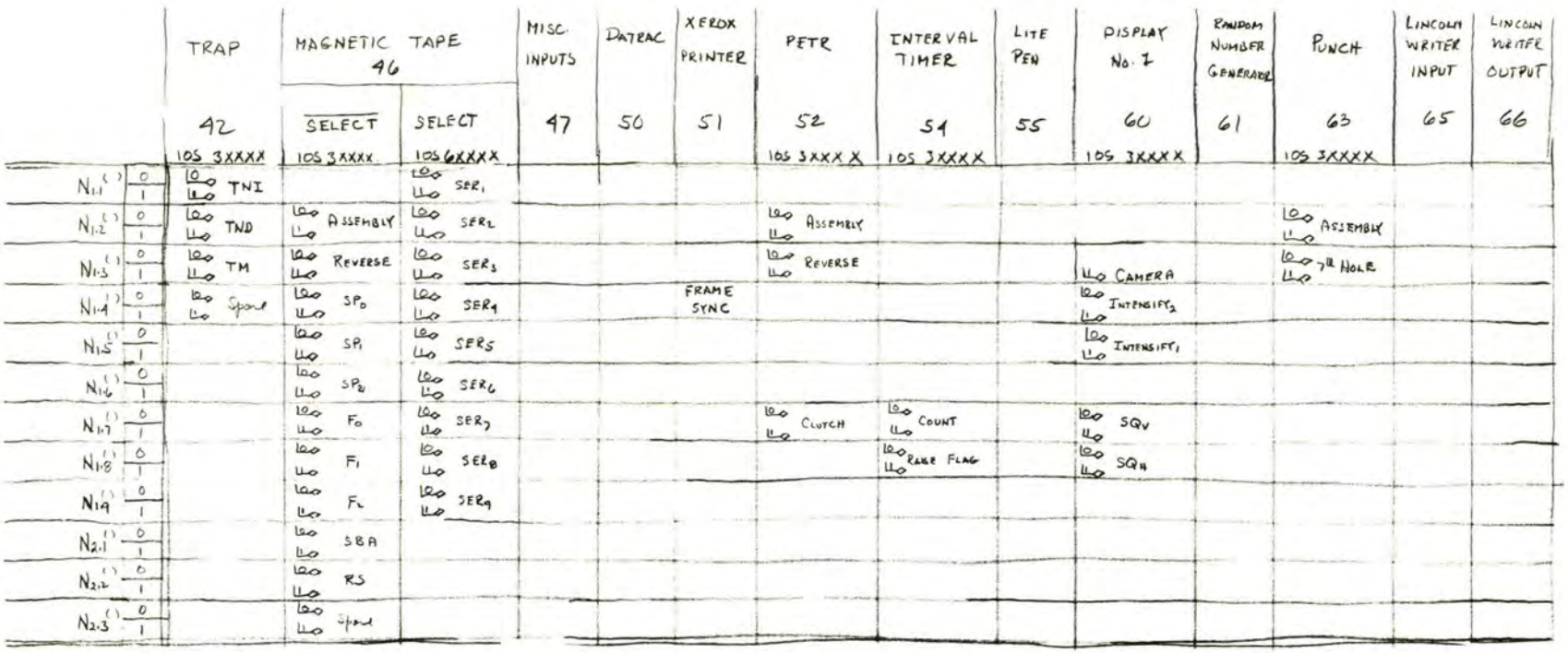


FIG 15-12
SIGNIFICANCE OF $N_{1,1-2,3}^{(\ )}$
DURING IOS 3X XXX AND IOS 6X XXX

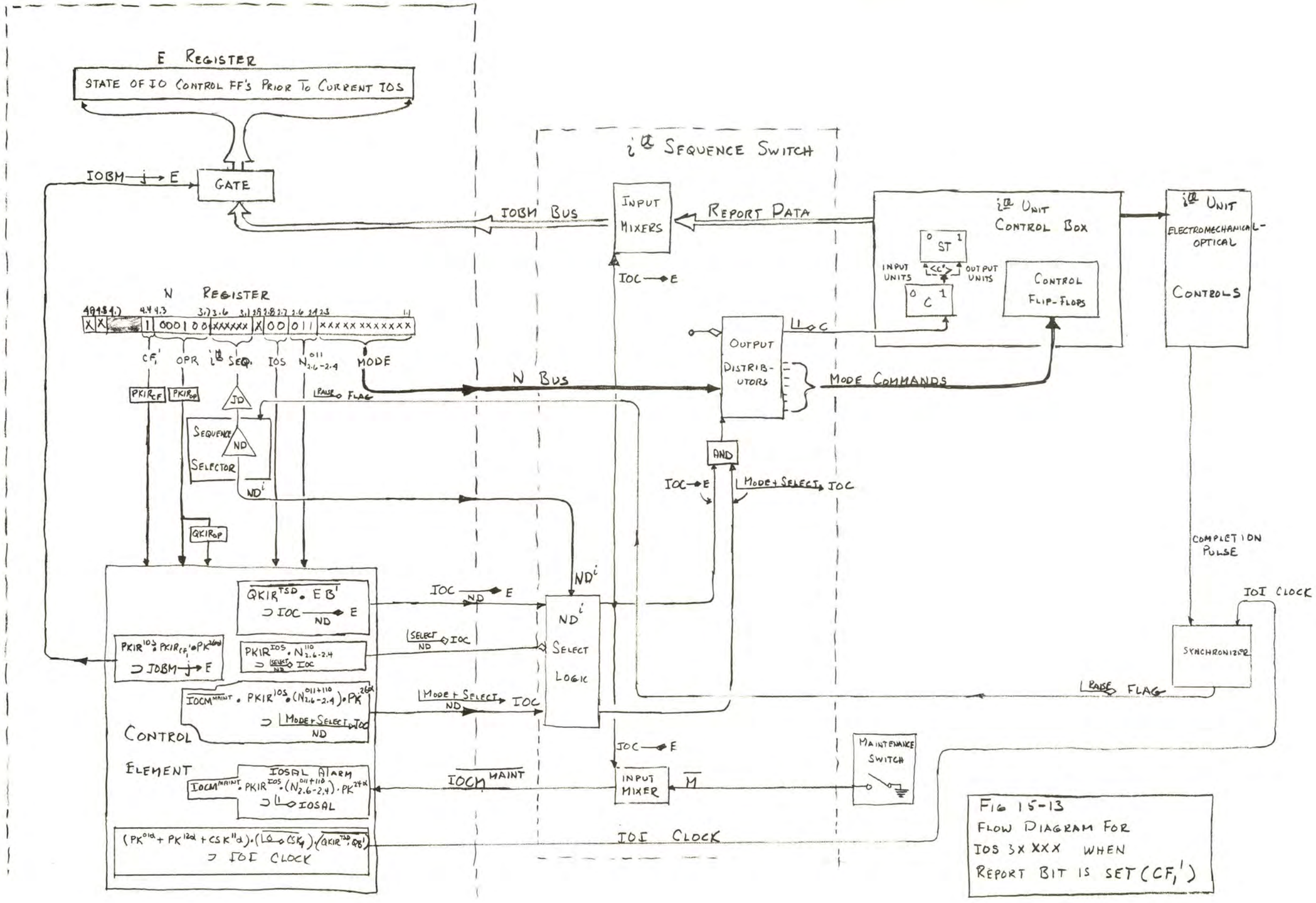


Fig 15-13
Flow Diagram For
IOS 3X XXX When
Report Bit is Set (CF1')

IOCM MODE LEVELS					
$IOCM^{IN}$			$\overline{IOCM}^{IN}$ (OUT)		
$IOCM^{NORMAL}$	$\overline{IOCM}^{NORMAL}$ (ASSEMBLY)		$IOCM^{NORMAL}$	$\overline{IOCM}^{NORMAL}$ (ASSEMBLY)	
	$IOCM^{RIGHT}$	$\overline{IOCM}^{RIGHT}$ (LEFT)		$IOCM^{RIGHT}$	$\overline{IOCM}^{RIGHT}$ (LEFT)
MAG TAPE	MAG TAPE (REV)	MAG TAPE (FWD)	MAG TAPE	MAG TAPE (REV)	MAG TAPE (FWD)
PETR		PETR (FWD)	PUNCH		PUNCH (FWD)
IN-OUT ALARM			XEROX		
MISC. INPUTS		STARTOVER	INTERVAL TIMER		TRAP
RANDOM NUMBER GENERATOR		LITE PEN	DISPLAY NO. 1		
DATRAC			LINCOLN WRITER OUTPUT		
LINCOLN WRITER INPUT					

FIG 15-14 IOCM MODE LEVELS ASSOCIATED WITH EACH IN-OUT UNIT

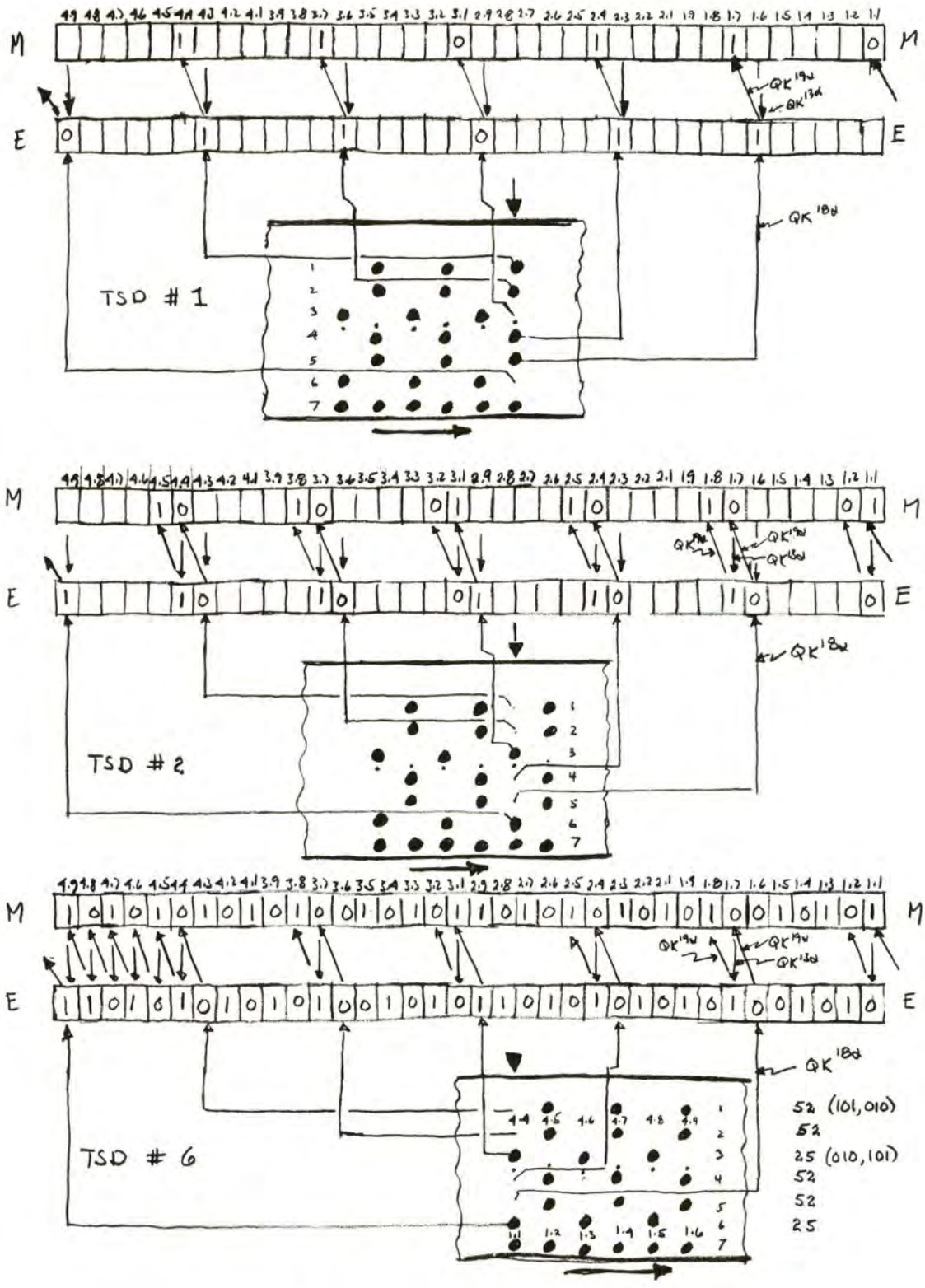


FIG. 15-16 PETR READING INTO MEMORY
BLOCK OF 36 BITS
IN
ASSEMBLY MODE

	MAGNETIC TAPE (46)						MISC. INPUTS 47	DATBAC 50	XEROX PRINTER 51	PETR 52		INTERVAL TIMER 54	LITE PEN 55	DISPLAY NO.1 60	RANDOM NUMBER GEN. 61	PUNCH 62		LINCOLN WRITER INPUT 65	LINCOLN WRITER OUTPUT 66
	IN IOBM→E			OUT E→IOB						IN IOBM→E						OUT E→IOB			
	ASSY FWD.	ASSY REV	NORMAL	ASSY FWD	ASSY REV	NORMAL	IN JOBM→E	IN IOBM→E	OUT E→IOB	NORMAL	ASSEMBLY	OUT E→IOB		OUT E→IOB	IN IOBM→E	NORMAL	ASSEMBLY	IN IOBM→E	OUT E→IOB
E1.1		IOB_1'	IOB_9'		IOB_9	IOB_9	MIB_1'	$DATB_{11}'$	CH_4	$HOLE_6'$		ITB_1			$SR1'$	HOLE 6		KB_1'	LW_1
E1.2			IOB_8'			IOB_8	MIB_2'	$DATB_{11}'$	CH_3	$HOLE_5'$		ITB_2			$SR2'$	HOLE 5		KB_2'	LW_2
E1.3			IOB_7'			IOB_7	MIB_3'	$DATB_{11}'$	CH_2	$HOLE_4'$		ITB_3			$SR3'$	HOLE 4		KB_3'	LW_3
E1.4	IOB_8'		IOB_6'	IOB_6		IOB_6	MIB_4'	$DATB_{11}'$	CV_4	$HOLE_3'$		ITB_4			$SR4'$	HOLE 3		KB_4'	LW_4
E1.5		IOB_9'	IOB_5'		IOB_8	IOB_5	MIB_5'	$DATB_{11}'$	CH_1	$HOLE_2'$		ITB_5			$SR5'$	HOLE 2		KB_5'	LW_5
E1.6			IOB_4'			IOB_4	MIB_6'	$DATB_{11}'$	CV_3	$HOLE_1'$	$HOLE_5'$	ITB_6			$SR6'$	HOLE 1	HOLE 6	KB_6'	LW_6
E1.7			IOB_3'			IOB_3	MIB_7'	$DATB_{11}'$	CV_2			ITB_7			$SR7'$				
E1.8	IOB_7'		IOB_2'	IOB_8		IOB_2	MIB_8'	$DATB_1'$	CV_1			ITB_8			SRB'				
E1.9		IOB_8'	IOB_1'		IOB_7	IOB_1	MIB_9'	$DATB_2'$	SC			ITB_9	$DV_{1.9}$		$SR9'$				
E2.1								$DATB_3'$	V_5			ITB_{10}	$DV_{2.1}$						
E2.2								$DATB_4'$	V_4			ITB_{11}	$DV_{2.2}$						
E2.3	IOB_6'			IOB_7				$DATB_5'$	V_3		$HOLE_4'$	ITB_{12}	$DV_{2.3}$				HOLE 5		
E2.4		IOB_7'			IOB_6			$DATB_6'$	V_2			ITB_{13}	$DV_{2.4}$						
E2.5								$DATB_7'$	V_1			ITB_{14}	$DV_{2.5}$						
E2.6								$DATB_8'$	V_0			ITB_{15}	$DV_{2.6}$						
E2.7	IOB_5'			IOB_6				$DATB_9'$				ITB_{16}	$DV_{2.7}$						
E2.8		IOB_6'			IOB_5			$DATB_{10}'$				ITB_{17}	$DV_{2.8}$						
E2.9								$DATB_{11}'$			$HOLE_3'$	ITB_{18}	$DV_{2.9}$				HOLE 4		
E3.1																			
E3.2	IOB_4'			IOB_5															
E3.3		IOB_5'			IOB_4														
E3.4																			
E3.5																			
E3.6	IOB_3'			IOB_4							$HOLE_2'$						HOLE 3		
E3.7		IOB_4'			IOB_3														
E3.8																			
E3.9														$DH_{3.9}$					
E4.1	IOB_2'			IOB_3					H_9					$DH_{4.1}$					
E4.2		IOB_3'			IOB_2				H_8					$DH_{4.2}$					
E4.3									H_7		$HOLE_1'$			$DH_{4.3}$			HOLE 2		
E4.4									H_6					$DH_{4.4}$					
E4.5	IOB_1'			IOB_2					H_5					$DH_{4.5}$					
E4.6		IOB_2'			IOB_1				H_4					$DH_{4.6}$					
E4.7									H_3					$DH_{4.7}$					
E4.8									H_2					$DH_{4.8}$					
E4.9	IOB_9'			IOB_1					H_1		$HOLE_6'$			$DH_{4.9}$			HOLE 1		

Fig 15-15
TSD Data Transfer Tab

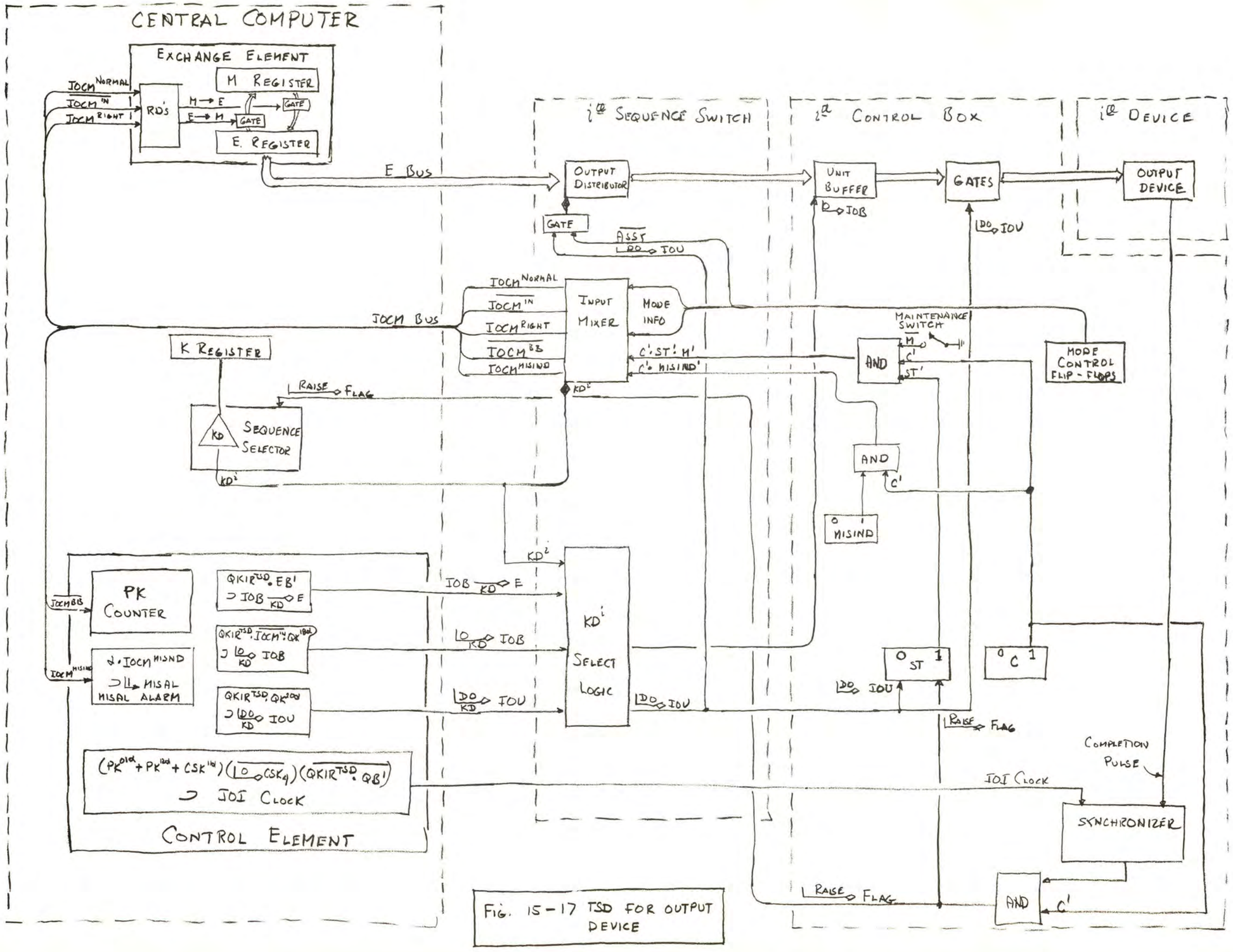


FIG. 15-17 TSD FOR OUTPUT DEVICE

CENTRAL COMPUTER

EXCHANGE ELEMENT

M REGISTER

F REGISTER

RD's

RD

GATE

IOCM NORMAL

IOCM RIGHT

M → E

F → M

IOBM → E

IOBM BUS

i^th SEQUENCE SWITCH

i^th CONTROL BOX

i^th DEVICE

INPUT MIXER

UNIT BUFFER

GATE

INPUT DEVICE

GATE

ASSY

IOB KD E

IOCM NORMAL

IOCM IN

IOCM RIGHT

IOCM BB

IOCM MISIND

INPUT MIXER

MODE INFO

IOCM BUS

K REGISTER

C'ST'M

C' MISIND

AND

M

C'

ST'

MODE CONTROL FLIP-FLOPS

RAISE FLAG

KD

SEQUENCE SELECTOR

AND

C'

MLSIND

KD^i

PK COUNTER

IOCM BB

QKIR^TSD · EB' ⊃ IOB KD E

IOB KD E

KP^i SELECT LOGIC

IOB → E

QKIR^TSD · IOCM^IN · QK^10a ⊃ LO KD IOB

LO KD IOB

IOCM MISIND

Q · IOCM MISIND ⊃ L → MISAL

MISAL ALARM

QKIR^TSD · QK^20a ⊃ DO KD IOU

DO KD IOU

DO IOU

ST

C

DO IOU

RAISE FLAG

COMPLETION PULSE

(PK^0a + PK^12a + CSK^10a)(LO CSK_4)(QKIR^TSD · QB') ⊃ IOI CLOCK

IOI CLOCK

CONTROL ELEMENT

SYNCHRONIZER

RAISE FLAG

AND

C'

Fig 15-18 TSD FOR INPUT DEVICE

(00) STARTOVER SEQUENCE

The Startover Sequence has several unique features:

1) It has top sequence priority (00).

2) The sequence is always connected (no C flip-flop).

3) The sequence switch consists of a synchronizer which synchronizes pulses from the STARTOVER button.

Fig. 15-Start'r Seq-1 shows a block diagram of the Startover Sequence. When the STARTOVER pushbutton on the console is depressed, it initiates a $\xrightarrow{1}$ SYN_1 level through the console control logic. This is an asynchronously generated level which is synchronized by the IOI clock pulses. The output of the synchronizer is the $\xrightarrow{\text{RAISE}}$ $FLAG_{(00)}$ pulse.

As soon as it is permissible, a change of sequence into the Startover Sequence will occur. Note that during the change of sequence to this sequence the flag of sequence 00 is lowered. Thus, if the STARTOVER button is pressed again while a STARTOVER sequence program is operating, another change of sequence to the STARTOVER sequence will occur as soon as the operating program dismisses.

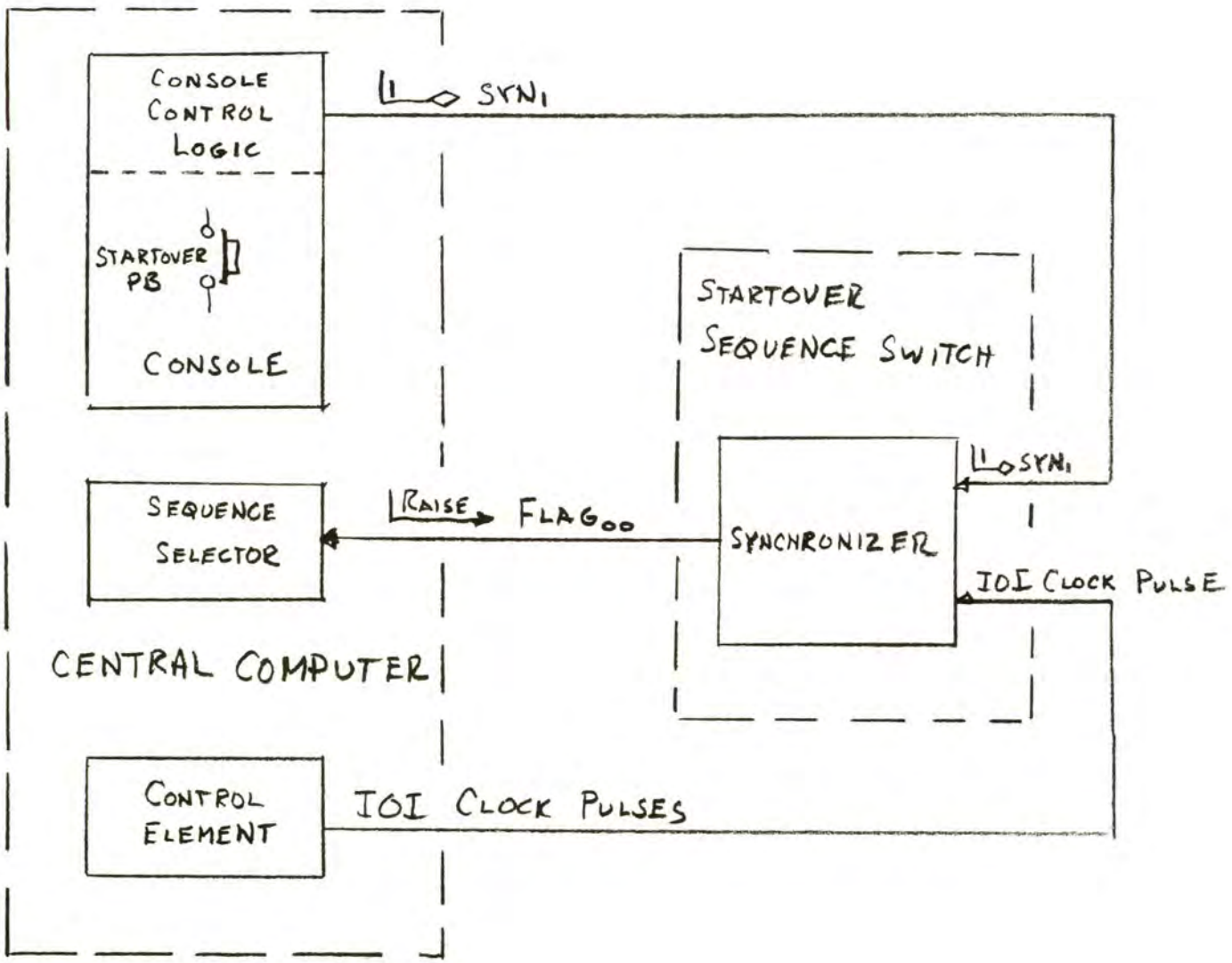


FIG 15- START'R - 1 STARTOVER SEQUENCE

(41) IN-OUT ALARM SEQUENCE

A block diagram of the In-Out Alarm system appears in Fig. 15-IO A1 Seq-1. This sequence has several unusual features:

1) It has no STatus flip-flop.

2) The $\overline{\text{IOCM}^{\text{BB}}}$ level is always generated as long as the sequence is connected (i.e., as long as C^{1}).

3) A TSD in this sequence transfers the data shown in Fig. 15-IO A1 Seq-2 into the E register.

An $\text{IOCM}^{\text{MISIND}}$ or IOCM^{EIA} level from any of the sequences shown will cause the $\text{IOCM}^{\text{EIA + MISIND}}$ level to be generated. Note that this level can be generated only by In-Out units which can cause an EIA or a MISIND. If the In-Out Alarm Sequence is connected (i.e., turned on by an IOS 30 000), the synchronized $\text{IOCM}^{\text{EIA + MISIND}}$ level will generate the $\xrightarrow{\text{RAISE}}$ FLAG_{41} pulse. Since the In-Out Alarm Sequence has a higher priority than nearly all the other priorities, the computer will quickly change to this sequence. What occurs thereafter depends on the program for the In-Out Alarm Sequence. If a TSD is included in the program, the status of the MISIND and EIA flip-flops of all the connected sequence will be transferred to the E register.

One peculiarity of this sequence is that, once an In-Out unit generates an alarm and raises the flag of the In-Out Alarm Sequence, no other unit's alarm will raise the flag until the first alarm is cleared, i.e., the EIA or MISIND flip-flop causing the alarm must be cleared to ZERO.

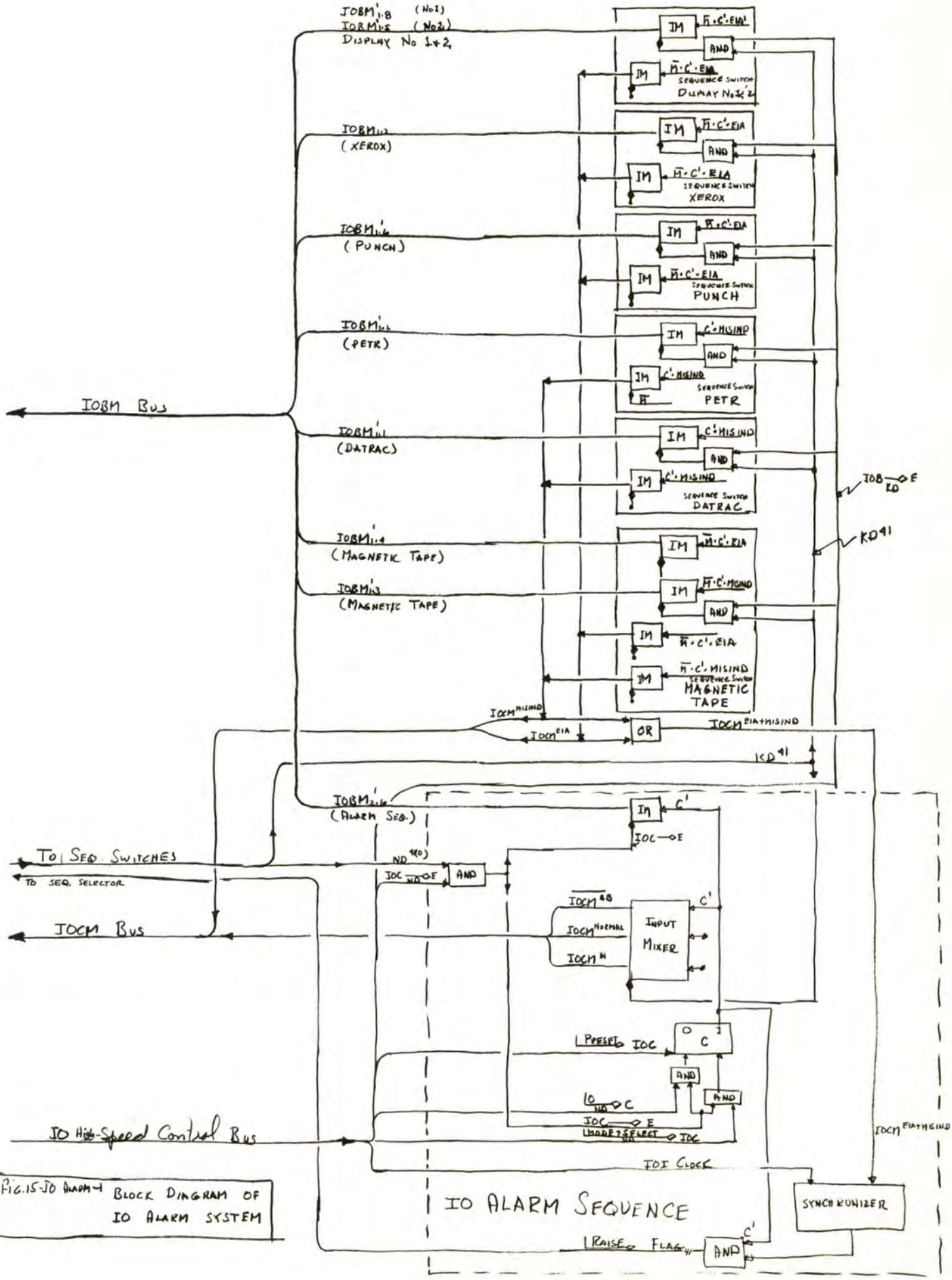


FIG. 15-10 ALARM 4 BLOCK DIAGRAM OF IO ALARM SYSTEM

SEQUENCE	ALARM REPORT
XEROX	$IOBM'_{1.7} = \overline{M} \cdot C' \cdot EIA$
PUNCH	$IOBM'_{1.6} = \overline{M} \cdot C' \cdot EIA$
DISPLAY No. 1	$IOBM'_{1.5} = \overline{M} \cdot C' \cdot EIA$
MAG TAPE	$IOBM'_{1.4} = \overline{M} \cdot C' \cdot EIA$
MAG TAPE	$IOBM'_{1.3} = \overline{M} \cdot C' \cdot MISIND$
PETR	$IOBM'_{1.2} = C' \cdot MISIND$
DATRAC	$IOBM'_{1.1} = C' \cdot MISIND$

FIG. 15-IOALARM-2. ALARM DATA REPORTED BY TSD IN ALARM SEQUENCE.

PETR is a photoelectric paper tape reader. This device uses photoelectric diodes to sense a tape that has been punched with 7 possible holes, plus a feed hole. Only 6 of the holes are used to store the data which is transmitted to the central computer. The seventh hole is used in the logic that indicates the end of the tape has been reached, i.e., it is used for control purposes only. The feed hole is used to generate the "completion" pulse that is used in the synchronizing process.

Data Transfer Modes. Data may be transmitted from the PETR buffer to the central computer in either the normal or assembly modes. It requires six TSD's to pack a 36-bit word in the central computer when the assembly mode is used. Data is never read into the PETR buffer, except when the tape is advanced in the forward direction.

Mechanical Tape Transport. Fig. 15-PETR-1 shows the major mechanical features of the PETR tape transport system. The tape may be transported in either the REEL or STRIP mode. When the STRIP mode is used the tape motion is determined entirely by the capstan. In this case the reel is not used, i.e., the tape is not wound on the reel. The reel clutch is left disengaged and the brake partially on.

Both capstan and reel assembly are belt driven by a single reversing drive motor as shown on the figure. Motion of the reel and capstan is then controlled by individual reel and capstan magnetic clutch and brake units. The direction of the drive motor is controlled by the REV flip-flop in the PETR control box.

The drive controls, when the tape is transported in the REEL mode, are designed to prevent the tape from accumulating slack between the reel and capstan. When the tape is running "binward" in the steady-state REEL mode, both the capstan and reel are driven by the motor. However, when the tape is running in the "forward" direction, the capstan clutch is disengaged and the capstan brake is partially engaged. The effect of the slippage in the capstan brake is to provide the reel with a light drag load.

Tape Transport Cycle. The basic tape transport cycle used in reading a tape into the computer is as follows:

First, the tape is advanced in the binward (REV^1) direction. During this phase, the data on the tape is sensed, but not gated into the buffer. (See Fig. 15-PETR-2.) When the end of the tape is reached, on octal 73 character (this is a character without a 7-th hole) is sensed. The octal 73 is ANDed with REV^1 to generate an End Mark (EM) level. EM is used to gate a feed-hole transition. $EM \cdot \langle H_f \rangle$ then clears the REV flip-flop to ZERO, thus reversing the direction of the drive motor. The tape now begins running in the forward direction. Note that if the PETR had not been logically connected, then $C^0 \cdot EM \cdot \langle H_f \rangle$ would have cleared the CLUTCH flip-flop to ZERO thus stopping the tape motion. Note also that $EM \cdot \langle H_f \rangle$ is not a synchronized signal.

Each feed hole (H_f) that the PETR senses is synchronized by an IOI clock pulse in the PETR synchronizer. If the tape is running in the forward direction and a seventh hole is present on the tape ($H_f \cdot REV^0$), then the output of the synchronizer will gate the tape data into the PETR buffer. The fact that PETR is connected (C^1) means that the output of the synchronizer will also be transmitted to the Sequence Selector as a $\boxed{\text{RAISE}} \rightarrow FLAG_{52}$ pulse.

Motion Control Logic. Fig. 15-PETR-2 is a block diagram of the PETR sequence switch and control unit. Most of the logic found on this figure has been previously described. However the motion control logic is unique to the PETR and requires explanation.

The motion control logic must be able to run the tape in both the forward and reverse direction in either the STRIP or REEL mode. In addition, the motion control logic must take into account the inertia transient effects during tape reversals and run-stop operations. Fig. 15-PETR-2 shows how this logic is generated. First, a level is generated, indicating that the computer wants the tape to move. This level is called M_v. M_v will not be present ($\overline{M_v}$) when the tape is slowing down, prior to stopping or reversing direction. A second level is generated and used when the tape is operated in the REEL mode and the tape is traveling in the bin direction. This level is called B.

Consider now how M_v and B are generated. Whenever the state of the REV flip-flop is changed, a term called VD_1^1 is generated. This is the output of a variable delay unit. VD_1^1 will persist for a predetermined length of time, after which the output of the variable delay unit will revert to VD_1^0. The function of this level (VD_1^1) is to stop the motion of the tape while the drive motor is changing its direction. Assuming that the unit is connected (C^1) and the Stop Unit level is not present, M_v will be generated as long as the CLUTCH flip-flop is set to ONE and the VD_1^0 level is present. Whenever a transition to $\overline{M_v}$ ($<\overline{M_v}>$) occurs, a variable delay level called VD_2^1 is generated. This level is similar to VD_1^1 and will become VD_2^0 after a predetermined length of time. The primary function of VD_2 is to apply the booster brakes when they are needed. Note that VD_2^1 occurs whenever VD_1^1 occurs (that is, during reversal operations), but that VD_1^1 does not necessarily occur whenever VD_2^1 occurs (that is, during $CLUTCH^1 \longrightarrow CLUTCH^0$ or $RUN \longrightarrow STOP$ operations).

There are two situations which will generate B. If the tape has been traveling in the binward direction for some time, the $REV^1 \cdot VD_1^0$ condition will be satisfied. This is sufficient to generate B. If the REV flip-flop is suddenly cleared to ZERO while the tape is traveling in the binward direction, B will persist until the tape actually comes to a stop and reverses direction. This happens because clearing the REV flip-flop to ZERO initiates VD_1^1, and $REV^0 \cdot VD_1^1$ generates B. Here again, the VD_1^1 serves as a digital memory for the mechanical system during the motor reversing period.

Consider next the logic used in operating the capstan and reel clutches and brakes. In addition to M_v and B, another level must be considered. This is the level initiated by the REEL-STRIP switch on the PETR PB control panel. The S level indicates the STRIP mode, and the $\overline{S}$ level indicates the REEL mode.

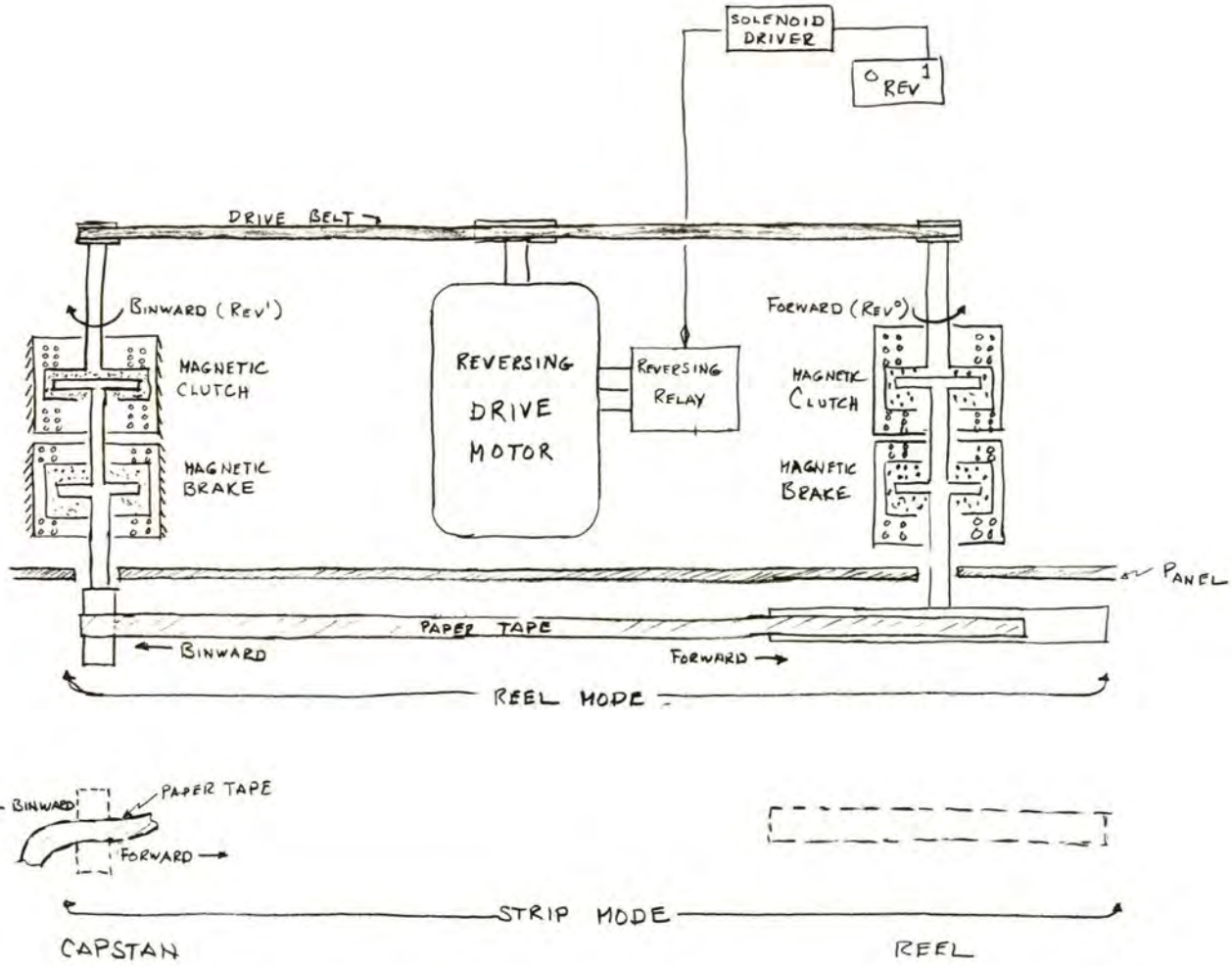
The presence of S is sufficient to disengage the reel clutch and engage the reel brake. The actual engagement and disengagement of the capstan clutch and brake occur conversely and, in the STRIP mode, depend only on M_v.

When the PETR is operated in the REEL mode, a slightly more complicated drive logic is used. When the tape is transported binward, the capstan clutch will be engaged whenever M_v is present. It will also be engaged during $\overline{M}_v$ while VD_2^1. The logic that is engaging the capstan clutch will always be disengaging the capstan brake. The reel clutch is engaged whenever M_v is present and, similarly, the reel brake operates whenever M_v is not present ($\overline{M_v}$). The reel brake booster is present only while VD_2^1. Fig. 15-PETR-3 shows the time relationship of these levels during a typical operating cycle.

MISAL Alarm. Since the PETR is a free-running input device, it has a MISIND flip-flop. Fig. 15-PETR-2 shows that the MISIND flip-flop is cleared to ZERO whenever the device is connected ($<C^1>$) or the PETR $\xleftarrow{\text{PRESET}}$ IOC level is generated.

The MISIND flip-flop is set under the following circumstances: Suppose that the Raise Flag signal has just gated data into the PETR buffer. This same Raise Flag signal will set the STatus flip-flop to ONE. Note that the MISIND flip-flop cannot be set to ONE because the STatus flip-flop is in the ZERO state when the Raise Flag signal arrives. If a TSD now occurs, data will be gated into the E register by an IOBM $\xrightarrow{j}$ E pulse at $QK^{18\alpha}$. The $\xleftarrow{DO}$ IOU pulse will then clear the STatus flip-flop at $QK^{20\alpha}$ and the cycle may be repeated. However, if another Raise Flag signal occurs before a TSD has read the content of the buffer into the computer and cleared the STatus flip-flop with a $\xleftarrow{DO}$ IOU pulse, the initial content of the PETR buffer will become permanently lost. In this case, the STatus flip-flop will be in the ONE state when the Raise Flag signal arrives. If, in addition, the STOP UNIT level is absent ($\overline{\text{STOP UNIT}}$), the MISIND flip-flop will be set to ONE. When this occurs the MISAL alarm flip-flop in the central computer will be set and, if the In-Out Alarm Sequence is turned on its flag will be raised.

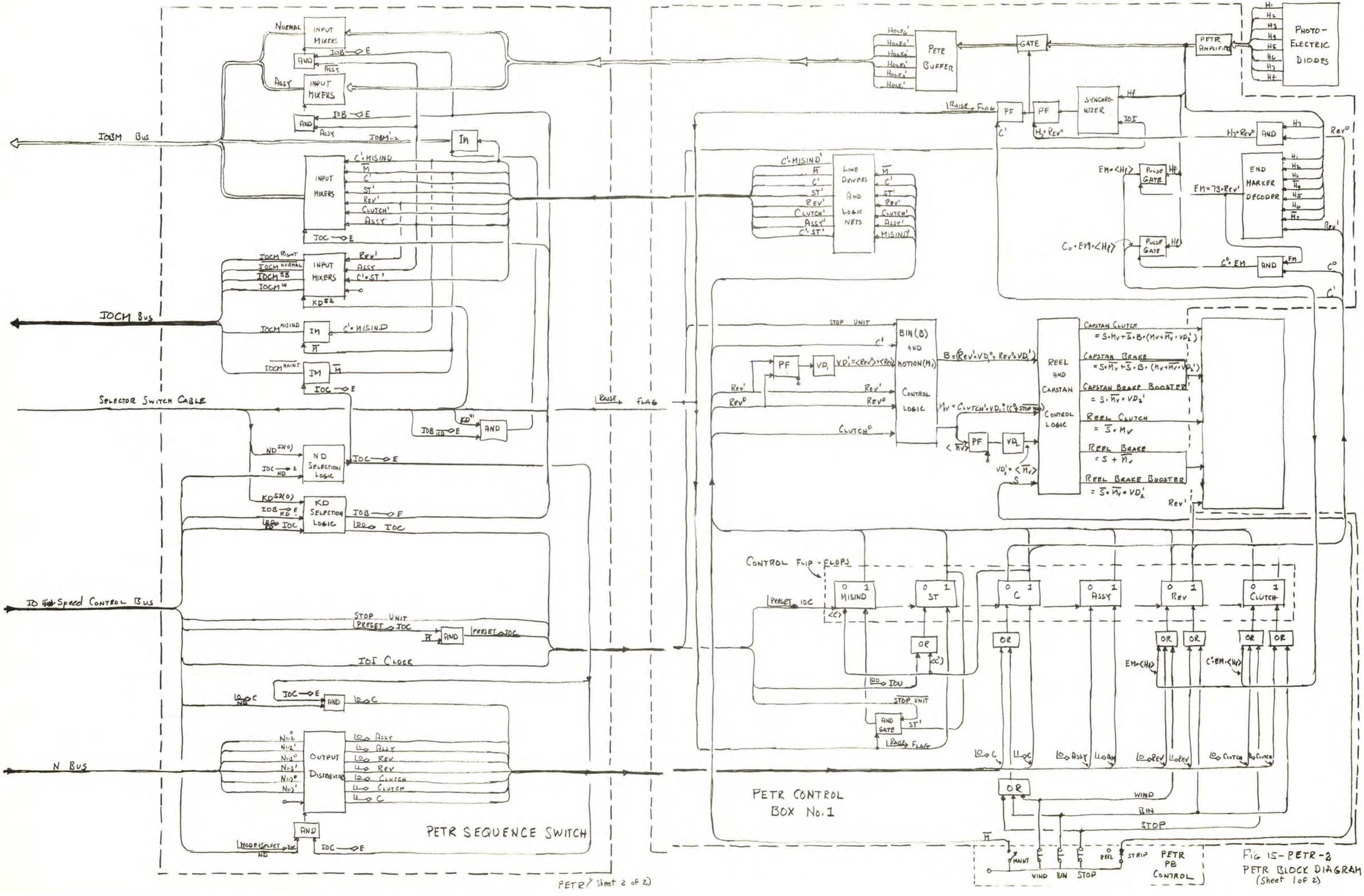
The reason for including the STOP UNIT level in the $\xleftarrow{1}$ MISIND logic is as follows: Suppose that some other sequence (magnetic-tape, for example) generates a MISIND which in turn sets the MISAL alarm and stops the computer. Even though the computer is stopped, the PETR will continue to generate Raise Flag signals until the tape can be brought to a stop by the $\overline{M_v}$ level. If the STOP UNIT level were not included in the $\xleftarrow{1}$ MISIND logic, it would be impossible to determine whether the Magnetic-tape Sequence or the PETR Sequence had caused the original MISAL alarm. The STOP UNIT level inhibits the PETR MISIND flip-flop from being set in this case.



		REEL MODE		STRIP MODE	
		BINWARD	FORWARD	BINWARD	FORWARD
CAPSTAN	CLUTCH	ENGAGED	DISENGAGED	ENGAGED	ENGAGED
	BRAKE	DISENGAGED	ENGAGED	DISENGAGED	DISENGAGED
REEL	CLUTCH	ENGAGED	ENGAGED	DISENGAGED	DISENGAGED
	BRAKE	DISENGAGED	DISENGAGED	ENGAGED	ENGAGED

STEADY-STATE CLUTCH AND BRAKE ENGAGEMENTS FOR PETR MODES

Fig. 15-PETR-1
PETR TAPE TRANSPORT SYSTEM



PETR/ Sheet 2 of 2)

FIG 15-PETR-2
PETR BLOCK DIAGRAM
(Sheet 1 of 2)

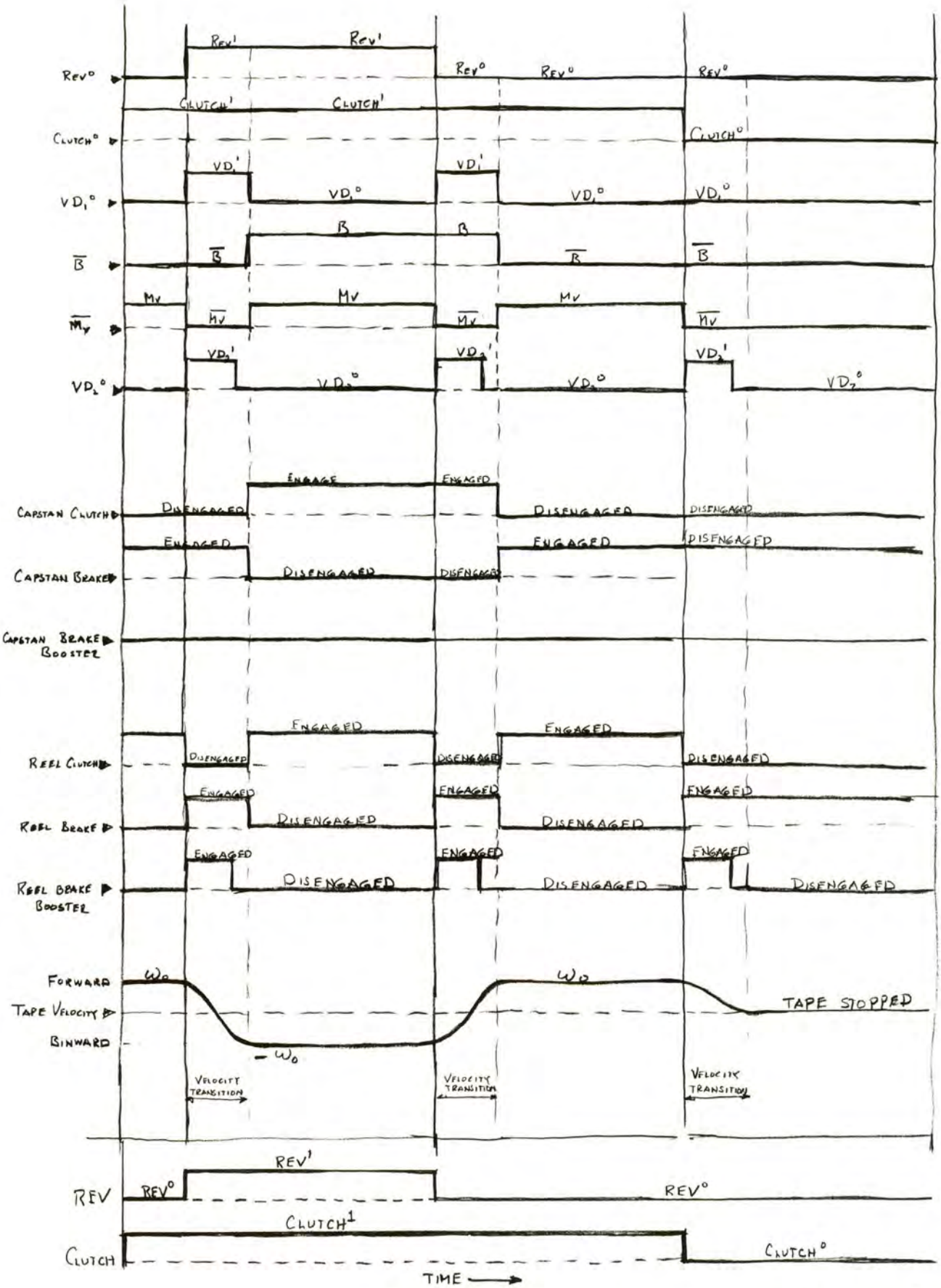


FIG. 15-PETR-3 PETR MOTION CONTROL LOGIC FOR REEL MOD

(63) HI SPEED PUNCH

This is an output device which punches holes in paper tape. Six bits of information are transferred from the central computer to the tape during each TSD.

Modes. Any of 4 possible modes can be selected for punching. In 7-th hole mode a 7-th hole is punched with each line of data resulting from a TSD.

Normal or assembly mode can also be specified. In the normal mode bits 1.1 through 1.6 of the register specified by the TSD are punched in that order on a line of tape with 1.6 going into hole 1. The line can be read as it existed in memory by viewing the tape with the 7-th hole on the right.

In the assembly mode each line punched is made up of every 6-th bit of the 36 bit word in memory starting with 1.6, viz. 1.6, 2.3, 2.9, 3.6, 4.3 and 4.9. After a TSD in this mode is performed the memory word is cycled one place to the left, so that successive TSD's referring to the same memory word record different bits of the word even though they are taken from the same bit positions. In this way a full 36 bit word is disassembled into 6 successive lines of tape. Normal or assembly modes can be used either with or without the 7-th hole mode.

IOS instruction bits which specify modes are as follows:

$$1.2 \quad \begin{cases} 0 = \text{NORMAL} \\ 1 = \text{ASSEMBLY} \end{cases}$$

$$1.3 \quad \begin{cases} 0 = \text{7-th HOLE}^{0} \\ 1 = \text{7-th HOLE}^{1} \end{cases}$$

The bits in the punch control transmitted to the E register by an IOS report instruction are as follows:

$IOBM_{1.2}$ ——— ASSY
$IOBM^{1}_{1.3}$ ——— 7-th HOLE
$IOBM_{2.5}$ ——— $\overline{M}.C^{1} \cdot$ EIA
$IOBM_{2.6}$ ——— C
$IOBM_{2.7}$ ——— M
$IOBM_{2.8}$ ——— ST

Mechanical Punch Cycle. The TSD instruction serves to start the drive motor as well as to initiate punching. No actual punching can occur, however, until the motor is up to speed. This involves a delay of about 1 second. The drive motor will continue to run as long as TSD commands are given at a rate exceeding one every 5 seconds. The motor will stop about 5 seconds after the last TSD command.

The basic mechanical cycle of the punch consists of: (1) punching the tape with the data stored in the punch buffer, and (2) advancing the tape while the buffer is loaded with more data from the central computer.

The punching mechanism has two built in pickups which generate "punch" and "feed" sync signals. The punch sync generates a positive going pulse at the beginning of the punch cycle and a negative going pulse at the end of the punch cycle. These pulses are identified as <START PUNCH> and <END PUNCH> , respectively. Similarly the feed sync generates a positive going pulse at the beginning of the feed cycle and a negative going pulse at the end of the feed cycle. These pulses are identified as <START FEED> and <END FEED> respectively. Since the <START FEED> and <END PUNCH> pulses are essentially coincident, the <END PUNCH> pulse is used to indicate both conditions.

Punch and Feed Control Details. Fig. 15-PUNCH-1 is a block diagram of the punch sequence switch, control box and mechanism. Fig. 15-PUNCH-2 shows the time relation of the events that occur during the punch feed cycle.

Assume that the central computer is in the punch sequence, and the sequence is connected, but that the punch motor is off. Suppose now that the program calls for a punch TSD. During the operand cycle (QK) the punch buffer will first be cleared by a $\lfloor^{0}\rightarrow$ IOB pulse and then a $\lfloor^{DO}\rightarrow$ IOU pulse will occur. This pulse does several things:

1) It is used in the sequence switch to gate data from the central computer into the punch buffer in the specified mode.

2) It clears the STatus flip-flop to ZERO. ST^{0} causes the $IOCM^{BB}$ level to be generated and in so doing tells the central computer the punch buffer is now busy.

3) It sets the PUNCH flip-flop to ONE, indicating a punch-feed cycle is to follow.

4) It causes the MOTOR ON level to be generated. This level comes from a variable delay unit. The $\lfloor^{DO}\rightarrow$ IOU pulse starts the variable delay unit timing. After the preset variable delay, the unit will generate a $\overline{\text{MOTOR ON}}$ level unless in the mean time another $\lfloor^{DO}\rightarrow$ IOU pulse (or $\langle FEED^{0}\rangle$) pulse has reset the variable delay. Actually two VD units are used to handle the variable delay logic.

The MOTOR ON level causes the motor to begin coming up to rated speed. The punch and feed sync signals start occurring. However these signals have no effect until the motor is up to rated speed (ω^{0}).

The first <START PUNCH> sync pulse sets CODE to a ONE (assuming the PUNCH is now set to ONE and the 1-second MOTOR ON delay has ended). $CODE^{1}$ permits the data in the buffer to be punched onto the tape. The <END PUNCH> sync pulse that follows sets the FEED flip-flop to ONE. $FEED^{1}$ causes the tape to be advanced in preparation for the next punch cycle.

The <END PUNCH> sync pulse also clears the CODE flip-flop to ZERO. $CODE^{0}$ in turn clears the PUNCH flip-flop to ZERO.

Finally the asynchronous <END PUNCH> sync pulse is synchronized in the synchronizer by an IOI clock pulse. The output of the synchronizer is then used to set the STatus flip-flop to a ONE. ST^{1} causes the $\overline{IOCM^{BB}}$ level to be generated which indicates to the central computer that the punch buffer is now not busy. The output of the synchronizer also causes the punch raise flag signal to be generated.

Suppose now that the program calls for another punch TSD to be executed. Another $\xrightarrow{DO}$ IOU pulse is generated during the second TSD operand cycle. The $\xrightarrow{DO}$ IOU pulse again sets the PUNCH flip-flop to ONE and pulses the variable delay unit. Note that the unit is pulsed before the delay has ended, i.e., the motor is still energized and operating at rated speed.

Finally the <END FEED> sync pulse associated with the first TSD occurs. This pulse clears the FEED flip-flop. The <START PUNCH> sync pulse again sets the CODE flip-flop to ONE.

The punch-feed cycles repeat in this manner until the program ceases to generate TSD's. The variable delay units will then time out and the $\overline{\text{MOTOR ON}}$ level will be generated. The drive motor will now coast to a stop.

Special Control Features. By means of the TAPE FEED switch on the punch panel, the tape may be advanced independent of the central computer. The TAPE FEED level causes the MOTOR ON level to be generated. After 1 second has elapsed, to allow the motor to come up to speed, the <END PUNCH> sync pulse will set the FEED flip-flop to a ONE. The tape will then be advanced.

Alarms. Alarm circuits have been provided to indicate the presence of conditions requiring human attention. All alarms will manifest themselves by a buzzer sounding and a red light turning on. The EIA flip-flop will also set when the next TSD occurs. If the alarm ACK-RESET switch is now set to ACK, (acknowledge) the buzzer will be suppressed and the red light and EIA condition will continue as long as the switch remains in this position, even after the cause of the alarm is corrected. If the alarm condition is corrected and the switch is turned to RESET, the light will go off, and the buzzer is stopped. However, the EIA flip-flop can only be cleared by a connection process.

The most common type of alarm results from the amount of tape on the supply reel diminishing to about 100 feet. This will cause the LOW TAPE level to be generated. However this will not prevent further punching.

The following conditions will generate an ALARM level which will prevent further punching:

1) If the tape handler fails to supply tape to the punch as required a switch above the slack loop will sense this and cause an alarm. This can happen if the bulb providing the light beam burns out. It can also happen if the tape is loaded improperly, or if the end of the tape roll is reached and is glued securely to the form. This alarm will inhibit further punching and manual feeding and allow the motor to stop after 5 second delay. The feed button or a TSD can restart the motor in this condition but it will not cause punching.

2) The end of the tape passing through the end of tape sensor will create the same effect as alarm 1 described above. This prevents the very end of the tape, which is thickened by a paper glued to it, from going into the punch and jamming it.

3) It is necessary to lubricate the punch after each 4 hours of running. To prevent it from being operated for longer intervals without lubrication, a timer is provided to shut the machine off once this period has elapsed since the last lubrication. Only maintenance personnel are authorized to reset this timer, which times out after 4.5 hours.

D-

TOLERANCES NOT OTHERWISE SPECIFIED
DECIMAL ± .005 FRACTIONAL ± 1/64 ANGULAR ± ½°
DIMENSIONS ENCLOSED THUS [.000] FOR REFERENCE ONLY

To & From Central Computer

E Bus

N Bus

IOBM Bus

IOCM Bus

To Hi-Speed Control Bus

Hi Speed Punch Sequence Switch

Output Mixers

Input Mixer

Input Mixers

Output Distributors

Logical Circuitry

Buffer

Punch

Punch Mechanism

Punch Pick Up

Feed Mechanism

Feed Pick Up

Time Relation of Punch & Feed Sync Signals

Manual Tape Feed Control

Interlocks Misc Controls with Punch

Misc Controls (Motor, Capstan, Etc.)

Switch Generated Levels

Hi Speed Punch Control Box

Fig 15 — Punch-1 Punch Block Diag.

GRADE I FOR REFERENCE ONLY
GRADE II PRELIMINARY DESIGN
GRADE III FINAL DESIGN
GRADED BY: DATE:

ITEM	MATERIAL - DESCRIPTION	PART NO.	QTY.

LINCOLN LABORATORY DIV. 6
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON 73, MASS.

PUNCH (63) BLOCK DIAGRAM

SCALE: DR.
ENG. CK. APPD.

D-

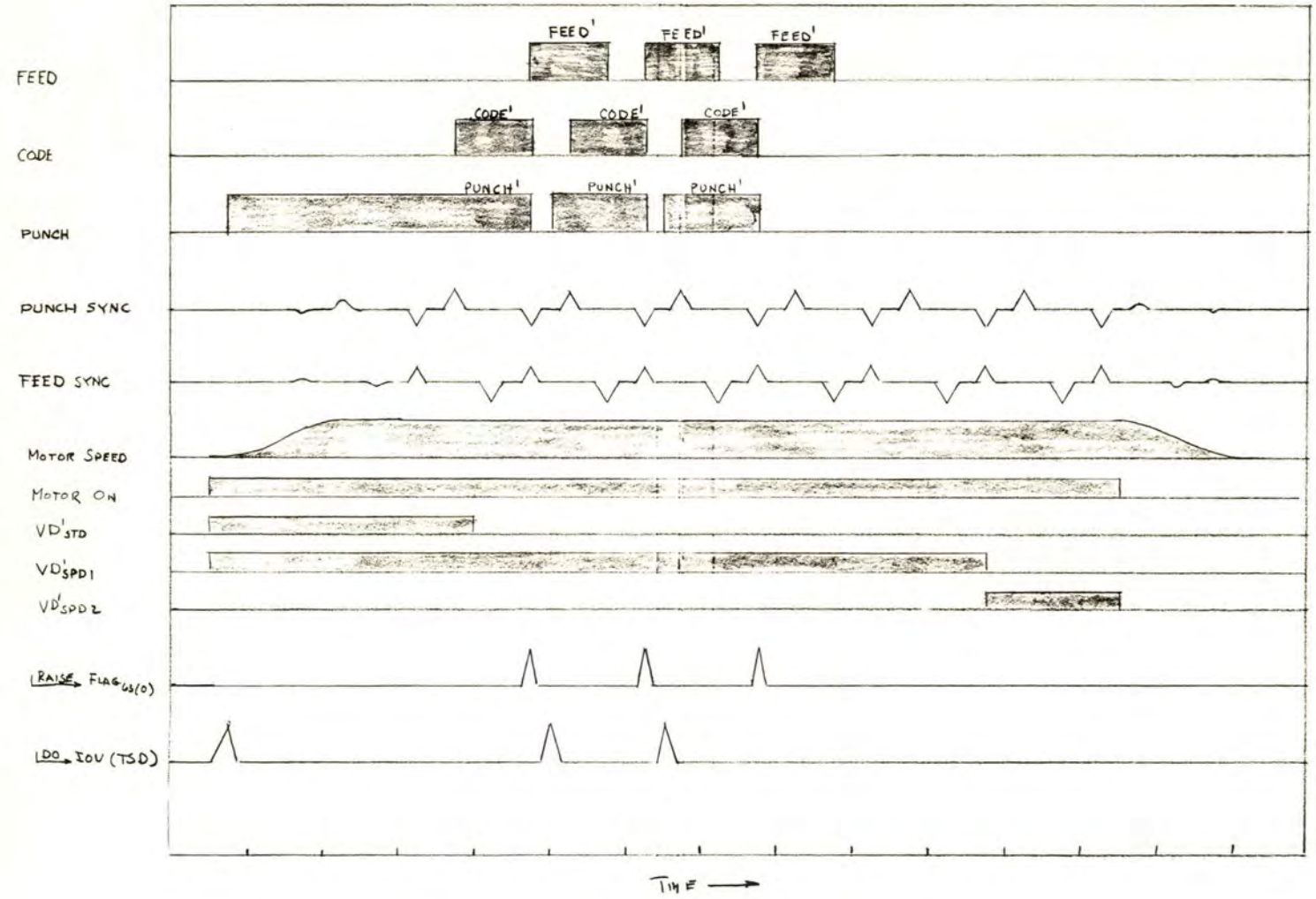


FIG. 15 — PUNCH — 2 Hi SPEED PUNCH SYNCHRONIZATION