9H-00310S5/04

PDP-15/76
HARDWARE FAMILIARIZATION

Student Guide

Revision 1

COURSE: PDP-15/76

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
8:15		LOGIC	CONSOLE 8 CONTROL TIMING	9 ME-15 MEMORY (DIFFERENCE)	LOADING A DOS RESTORE
9:00	1 SYSTEM BLOCK DIAGRAM	FAMILIARIZATION CPU 2	CONSOLE KEY FLOW	10 EAE INST. SET	USE OF EDIT
10:00	EQUIPMENT	BLOCK DIAGRAM. CPU TIMING. CPU DECODE of MRI. 3	CONSOLE SWITCH BOARD & DISPLAY PANEL		USE OF ASSEMBLER LOADING PROGRAMS
11:00	FAMILIARIZATION	MEMORY (MM-15) 4	LOGIC FLOW OF A DEPOSIT FUNCTION	11 E.A.E HARDWARE	PROGRAMMING ASSIGNMENT
12:00	L	U	N	C	H
1:00	POWER DISTRIBUTION	5 MRI OPERATION FLOW B.D. (DAC & LAC)	LAB	E.A.E FLOWS	LAB
2:00	LAB	6 CPU INTERFACING WITH MEMORY. LOGIC DAC & LAC	PROJECT	LAB	PROJECT
3:00	PROJECT	7 CPU INSTRUCTION SET a) MRI	#2	PROJECT	#4
4:00	#1	b) OPERATE c) REGISTER CONT. d) INDEX. REGISTER XFER INST.	USE OF CONSOLE	#3 SCOPE PROJECT	LOADING SYSTEM SOFTWARE AND USER PROGRAMS
5:00					

COURSE: PDP-15/76 WEEK 2 OF 3

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
8:15	13 BLOCK DIAGRAM OF PROGRAM CONTROL	15 PROGRAM	TTY CONTROLLER	16 HARDWARE READ-IN	18 AUTOMATIC
9:00	DEVICES IOT 14	INTERRUPT	a) RECEIVER b) TRANSMITTER		PROGRAM INTERRUPT
10:00	INPUT OUTPUT	FACILITY	16 BA-15 BLOCK DIAGRAM PC-15	17 MEMORY	19 POWER
11:00	FACILITY		PAPER TAPE READER/PUNCH	PROTECT & RELOCATE	FAIL CONTROL
12:00	L	U	N	C	H
1:00	LAB	LAB	LAB	LAB	20 REAL
2:00	PROJECT	PROJECT	PROJECT	PROJECT	TIME CLOCK
3:00	#	#	#	#	REVIEW
4:00	5	6	7	8	
5:00					

COURSE: PDP-15/76

WEEK 3 OF 3

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
8:15	21 INTRODUCTION TO DATA CHANNEL	22 ADD	23 XVM	24 INTRODUCTION TO UNICHANNEL	
9:00	3 CYCLE	TO MEMORY REVIEW		BLOCK DIAGRAM BLOCK DIAGRAM OF	
10:00	BREAK	THE SYSTEM		MEMORY CONTROL	
11:00	DAT1 & DATO	BLOCK DIAGRAM			
12:00	L	U	N	C	H
1:00	22 SINGLE	LAB			
2:00	CYCLE BREAK	PROJECT			
3:00	INC MB	#9			
4:00					
5:00					

TABLE OF CONTENTS

TABLE OF CONTENTS

SECTION

TABLE OF CONTENTS

SECTION

TABLE OF CONTENTS

SECTION

TABLE OF CONTENTS

TABLE OF CONTENTS

MAIN GOAL OF COURSE

Have field service engineer or customer install, fix, and maintain PDP-15 systems.

1. System Block Diagram of PDP-15 System Objectives:

   a) Using the block diagram of the PDP-15 system, define the function of each block

   b) Define the purpose of the DW-15

   c) List the devices in the BB-15 and explain their function

   d) List the options installed in the BB-15

   e) List the two general types of Data Transfers.

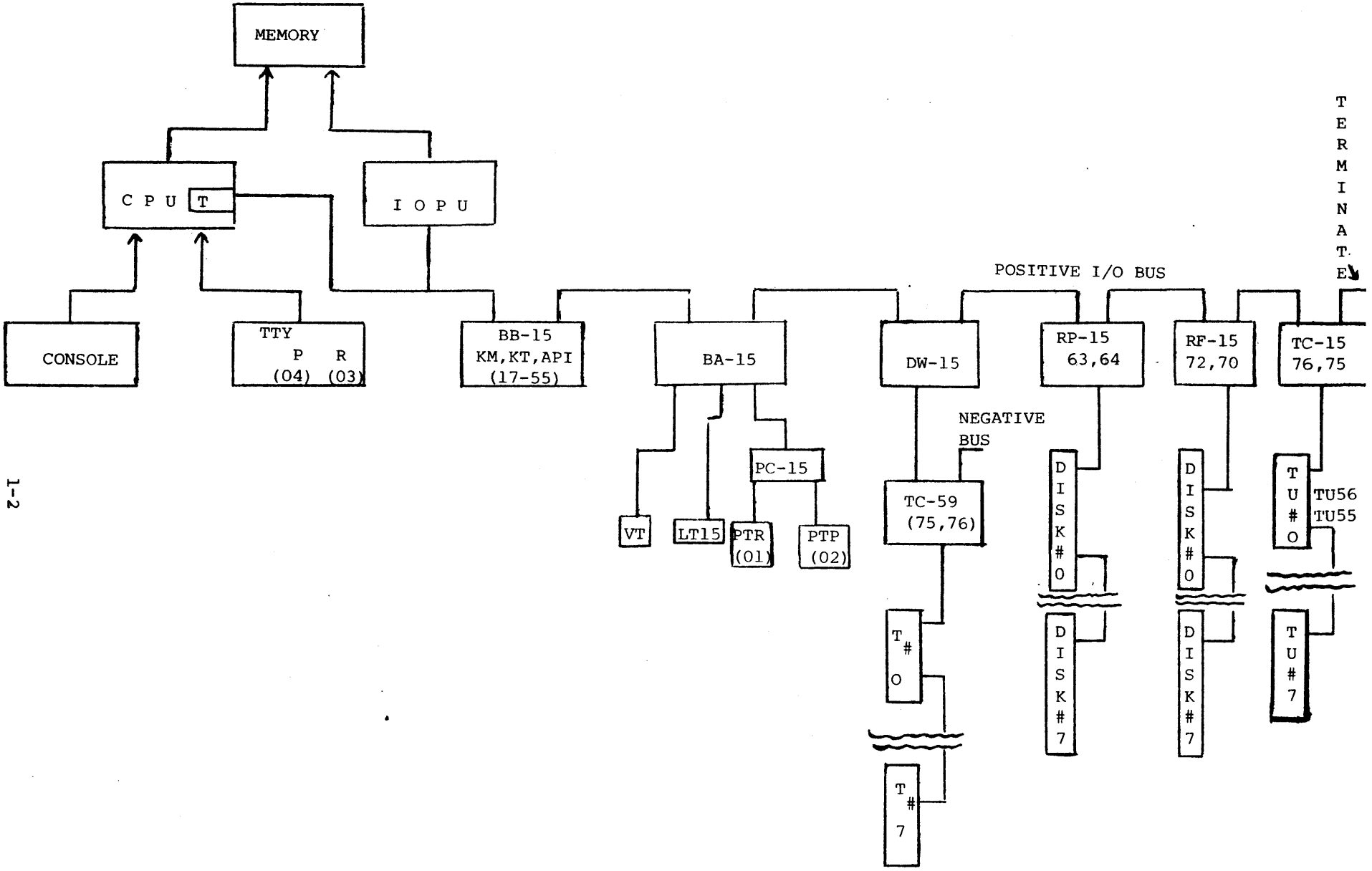
   f) Define program control transfers

   g) List high speed block transfers devices

   h) Define the type of data transfers high speed devices use.

   i) Using the equipment layout prints for the PDP-15 system have the student install this system on paper then check work against the PDP-15 in the lab. Student correct his own work.

   j) Define priority if any on the BUS



BASIC PDP-15 SYSTEM BLOCK DIAGRAM

2. CENTRAL PROCESSOR BLOCK DIAGRAM

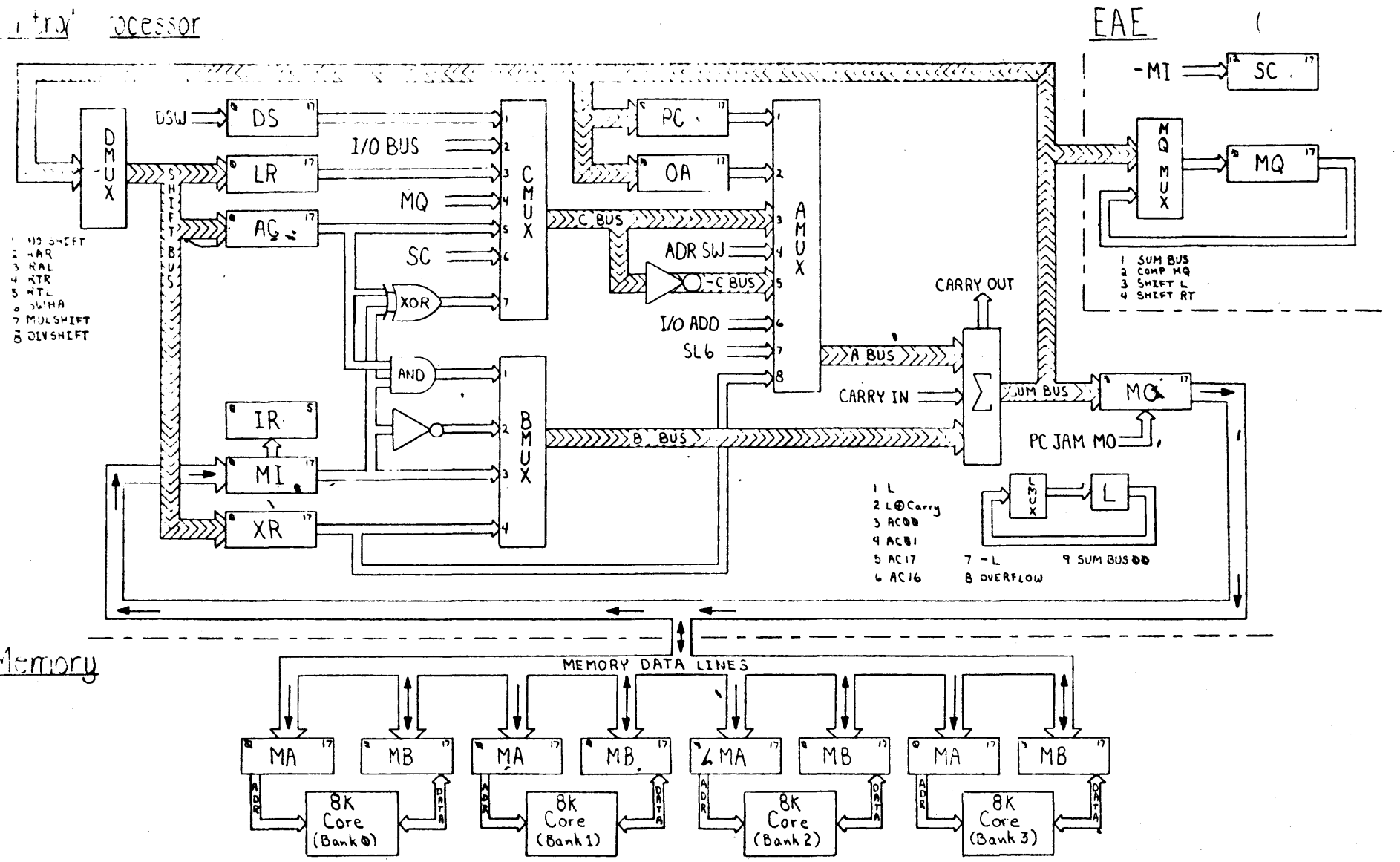
OBJECTIVES:

1. List the registers in the central processor and explain their functions.

2. Define in writing how the multiplexor in the central processor works.

3. Define in writing the address scheme for the MM-15 memory.

4. Draw the block diagram of the memory processor interface.



Instruction Execution Block

3. CENTRAL PROCESSING TIMING

OBJECTIVES:

1. Upon request be able to explain during which major states the central processor can receive an instruction from memory.

2. Upon request be able to explain in what major states does the central processor decode and process the instruction.

3. List the three ways in which the central processor, running under program control, cna be halted.

4. List the GRP decodes for any given memory reference instruction.

CPU-RUNNING INSTRUCTIONS

MAJOR STATES

	FETCH	INC	DEFER	EAE	EXECUTE
TS01					
TS02					
TS03					
	1→FETCH 1→INC 1→DEFER 1→EAE 1→EXECUTE	1→FETCH 1→EXECUTE	1→FETCH 1→INC 1→EXECUTE	1→EXECUTE	1→FETCH

INSTRUCTION

FLOW CHART

KP 76

QUESTIONS:

When does the CPU receive an instruction?

When does the CPU decode & process the instruction?

DECODES FOR MEMORY REFERENCE INSTRUCTIONS (MRI)

MRI	GROUPS 1	2	3	4	5	6
CAL		✓		✓		
DAC	✓	✓				
JMS	✓	✓		✓		
DZM	✓	✓				
LAC	✓		✓		✓	
XOR	✓		✓		✓	
ADD	✓		✓		✓	✓
TAD	✓		✓		✓	✓
XCT	✓					
ISZ	✓		✓			
AND	✓		✓		✓	
SAD	✓		✓		✓	
JMP	✓		✓			
REGISTERS	MI	IR	IR	IR	IR	IR

DECODES LOCATED ON

KP 30

CPU-RUNNING INSTRUCTIONS

MAJOR STATES

	FETCH	INC	DEFER	EAE	EXECUTE
TS01					
TS02					
TS03					
	1→ FETCH 1→ INC 1→ DEFER 1 — EAE 1→ EXECUTE	1→ FETCH 1→ EXECUTE	1→FETCH 1→INC 1→EXECUTE	1→EXECUTE	1→FETCH

INSTRUCTION

FLOW CHART

KP 76

QUESTIONS:
When does the CPU receive an instruction?

When does the CPU decode & process the instruction?

DECODES FOR MEMORY REFERENCE INSTRUCTIONS (MRI)

MRI	GROUPS 1	2	3	4	5	6
CAL		✓		✓		
DAC	✓	✓				
JMS	✓	✓		✓		
DZM	✓	✓				
LAC	✓		✓		✓	
XOR	✓		✓		✓	
ADD	✓		✓		✓	✓
TAD	✓		✓		✓	✓
XCT	✓					
ISZ	✓		✓			
AND	✓		✓		✓	
SAD	✓		✓		✓	
JMP	✓		✓			
REGISTERS	MI	IR	IR	IR	IR	IR

DECODES LOCATED ON

KP 30

MEMORY - CENTRAL PROCESSOR BLOCK DIAGRAM

1. Name the registers in the CPU and their functions

   a) PC - contains address in memory of next instruction to be executed

   b) OA - temporary storage of the operand address for all MRI

   c) DS - console data switches

   d) LR - used to set a limit on operations with the index register or for storage

   e) AC - retains results of arithmetic/logical operations program controlled I/O transfers go thru AC

   f) IR - 6 BITS 0-3 OPCODE BIT 4 indirect BIT
      BIT 5 index BIT

   g) MI - receives all data & instructions read from core

   h) XR - used in indexed addressing operations, as a counter or for storage

   i) LINK - 1 BIT extension of accumulator

   j) MO - contains all information going from CPU to memory data lines

2. Memory Organization

   a) Bank Mode
      Programs run in 8K partition

   b) Page Mode
      Programs run in 4K partition

   c) Data paths between CPU and memory

4. MM-15 Memory Block Diagram

1. Define a memory cycle

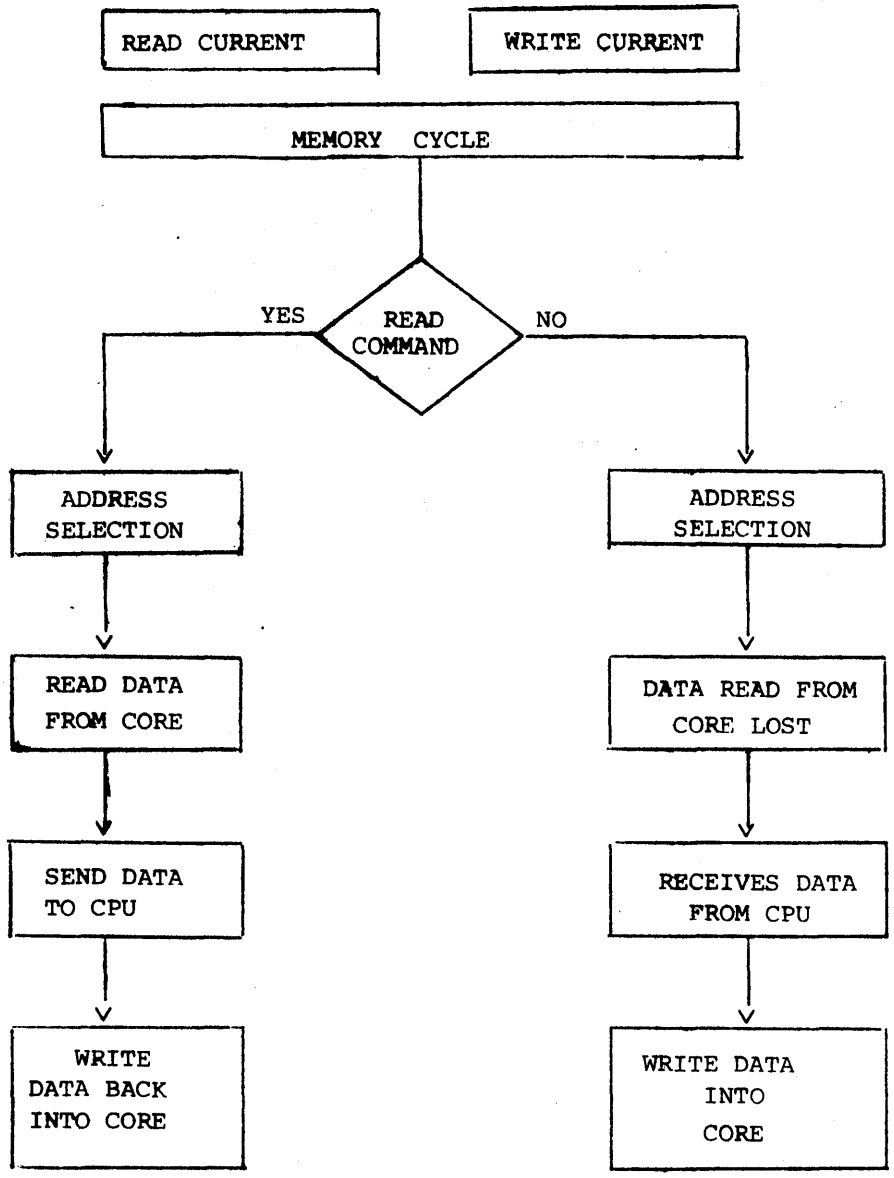
2. List the events that takes place in a read cycle

3. List the events that takes place in a write cycle

4. Draw a block diagram of memroy that can perform a read, write cycle.

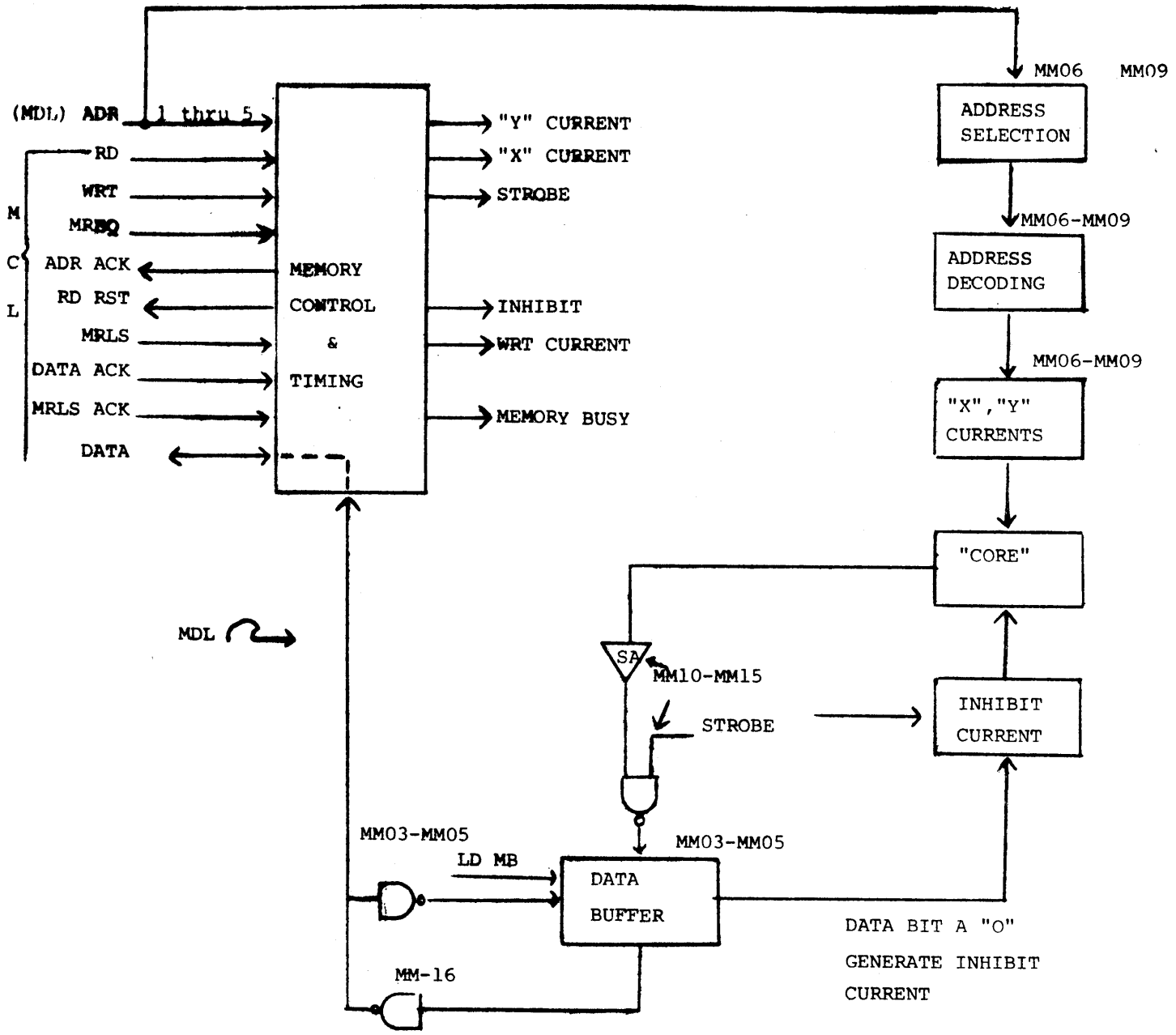
5. Upon request be able to orally explain address decoding

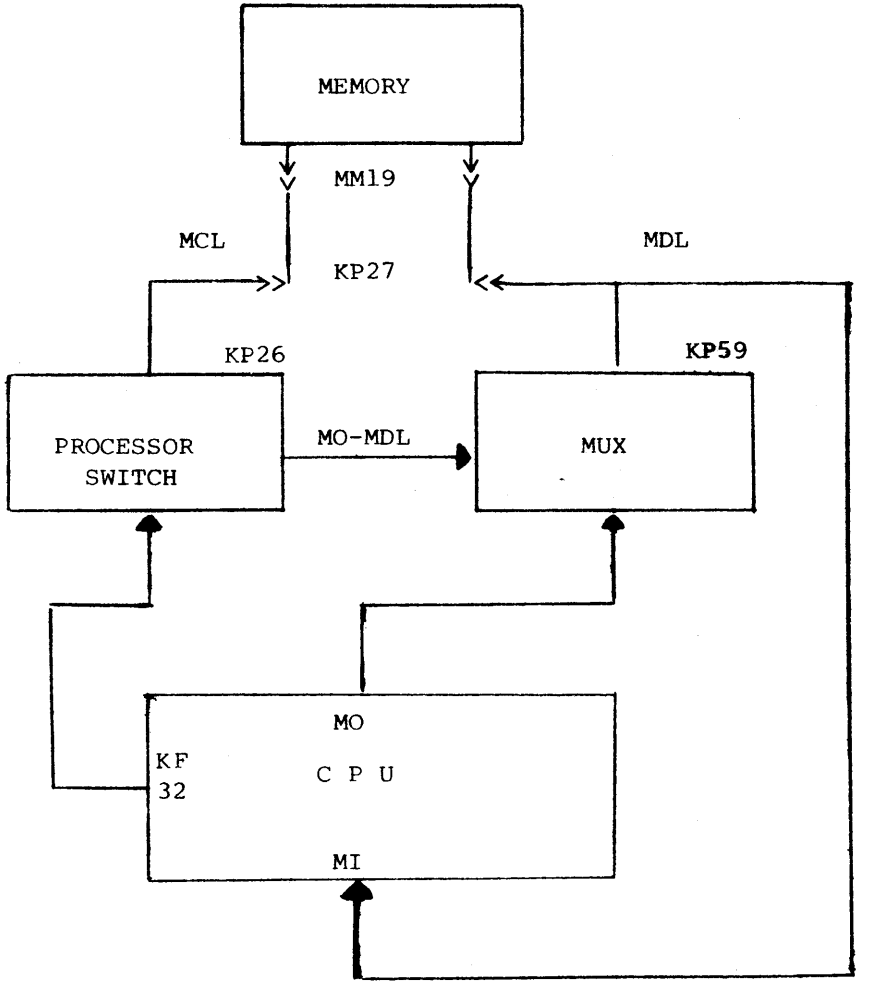
6. Identify the dialogue response to any signal on the memory control lines (MCL).


READ CURRENT
WRITE CURRENT
MEMORY CYCLE
READ COMMAND
YES
NO
ADDRESS SELECTION
READ DATA FROM CORE
SEND DATA TO CPU
WRITE DATA BACK INTO CORE
ADDRESS SELECTION
DATA READ FROM CORE LOST
RECEIVES DATA FROM CPU
WRITE DATA INTO CORE

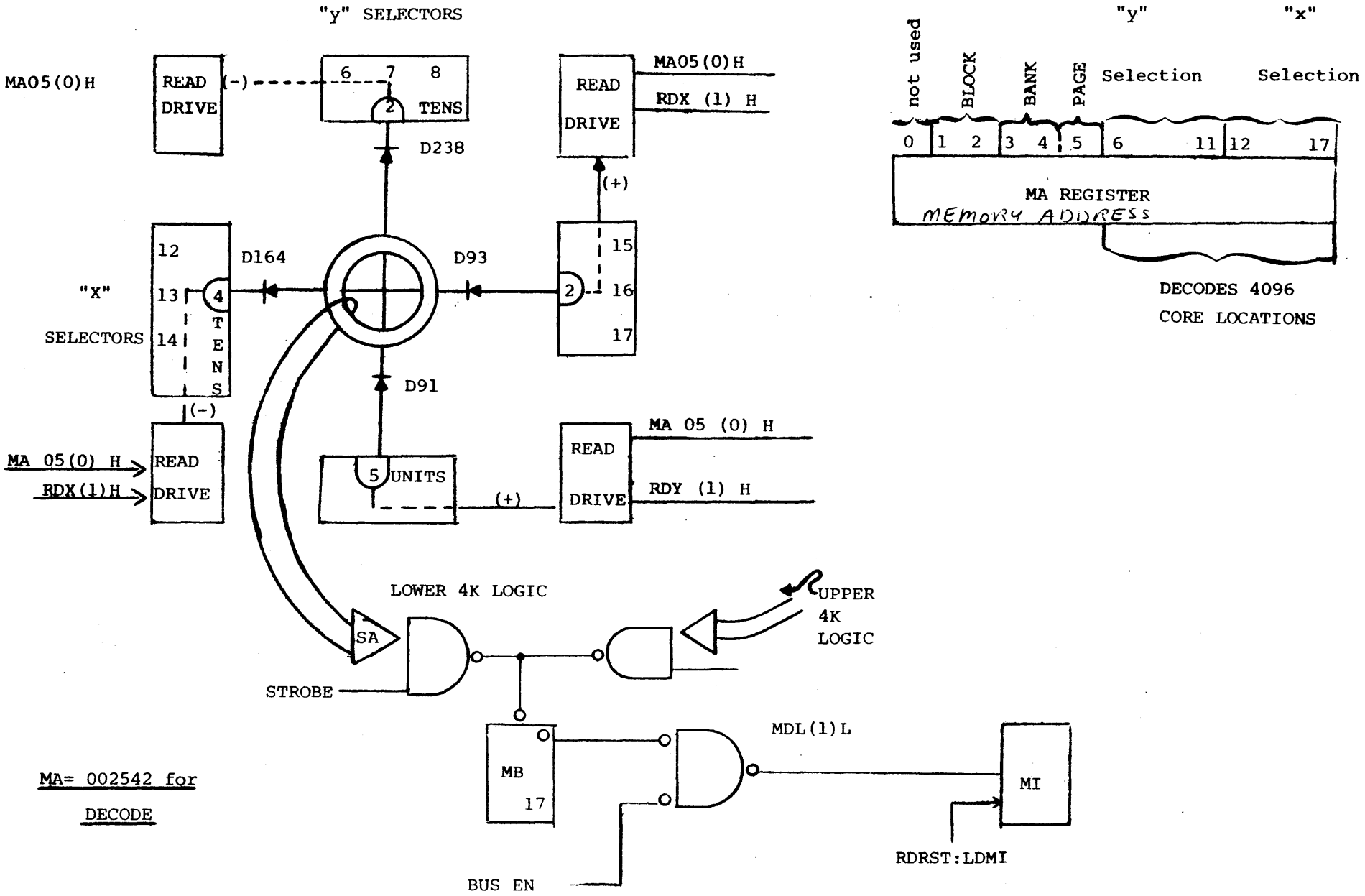
MEMORY BLOCK DIAGRAM

ADDRESS BITS 6 - 17



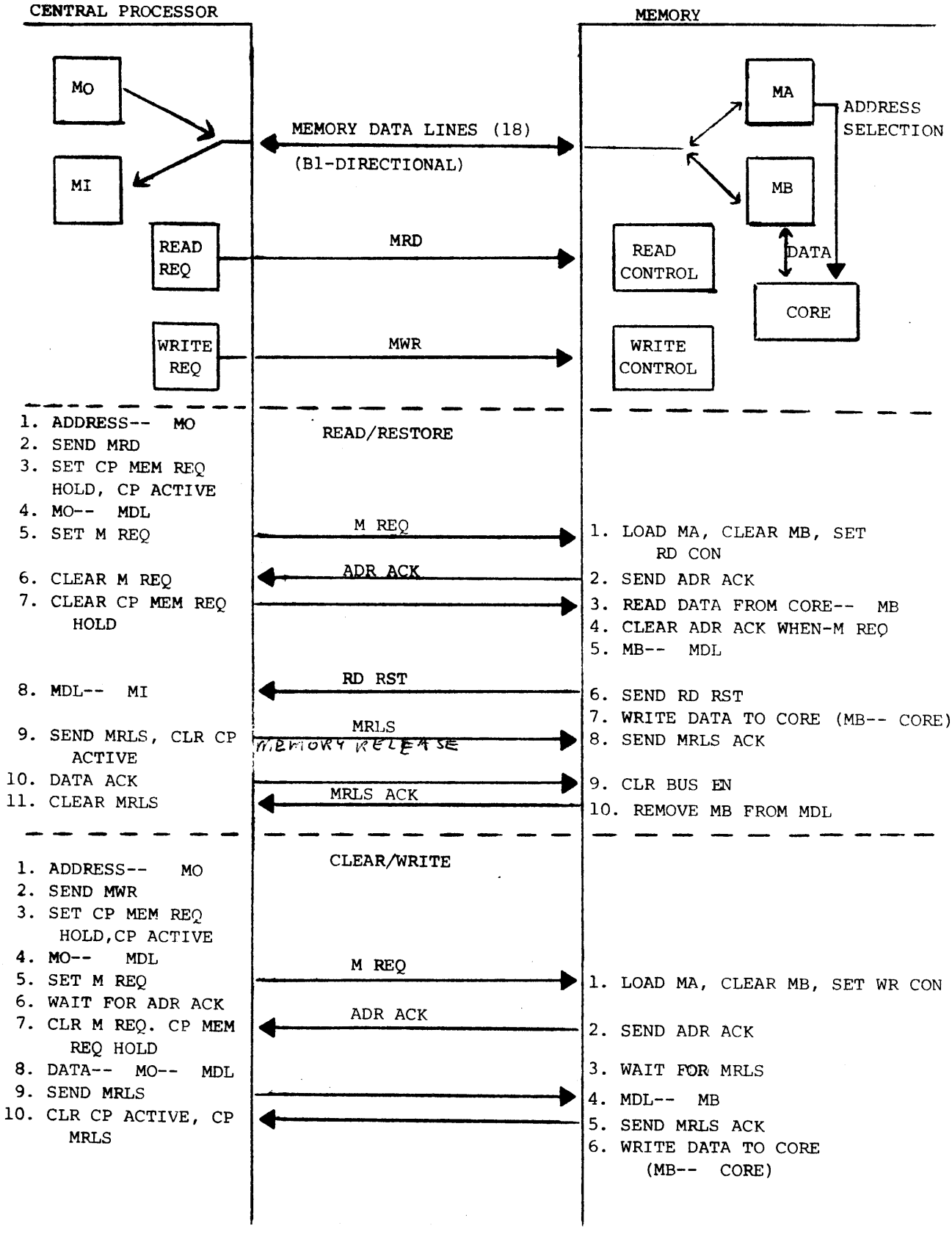


MEMORY CPU INTERFACE



READ CYCLE OF MEMORY - TAKES DATA OUT OF CORE, STROBES IT INTO THE MEMORY BUFFER, SENDS DATA TO CPU.

MEMORY CPU INTERFACE



5. MEMORY REFERENCE INSTRUCTIONS

OBJECTIVES:

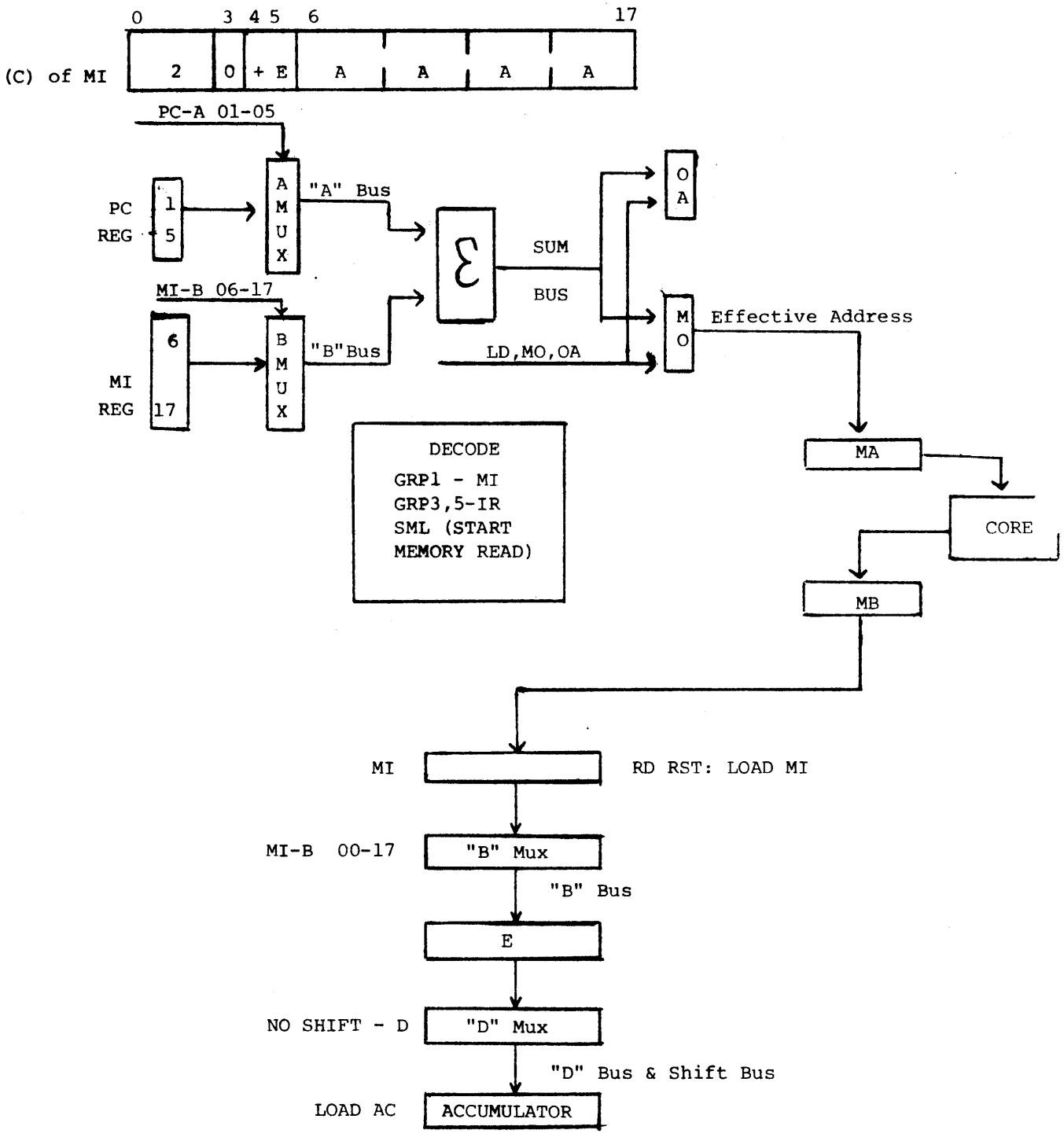
1. Using the central processor block diagram, describe the flow of addressing core, receiving the data and transfer it to the accumulator.

2. Using the central processor block diagram descirbe the flow of addressing core, and transferring data from the accumulator into core.

3. List the address modes and describe their function.

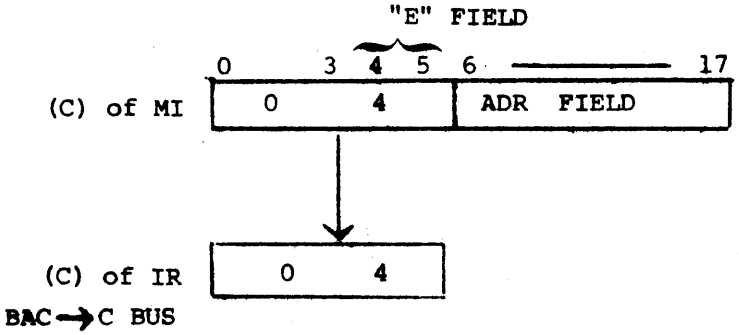
LAC: Contents of the memory location specified by the effective address replaces the contents of the accumulator.

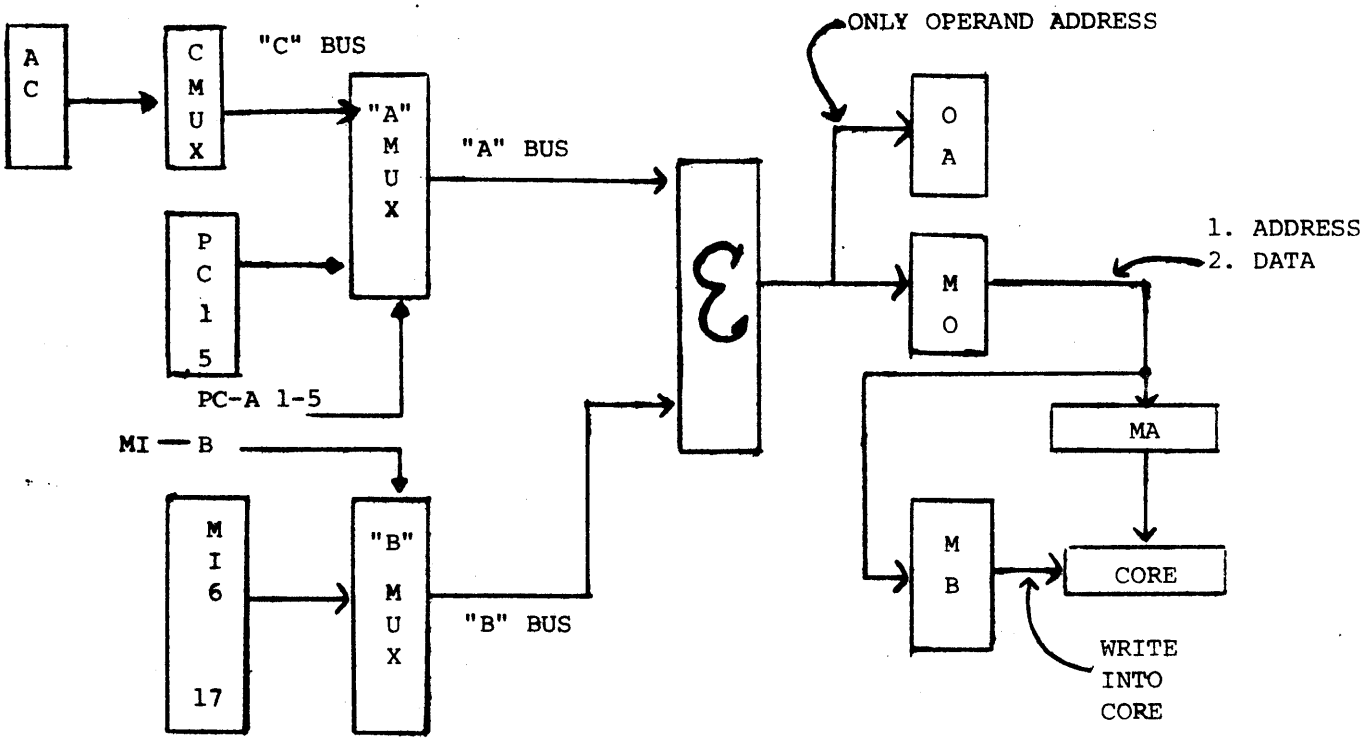
LAC: LOAD THE ACCUMULATOR



DAC - DEPOSIT ACCUMULATOR

The contents of the accumulator is deposited in the memory location specified by the effective address. The accumulator remains unchanged.

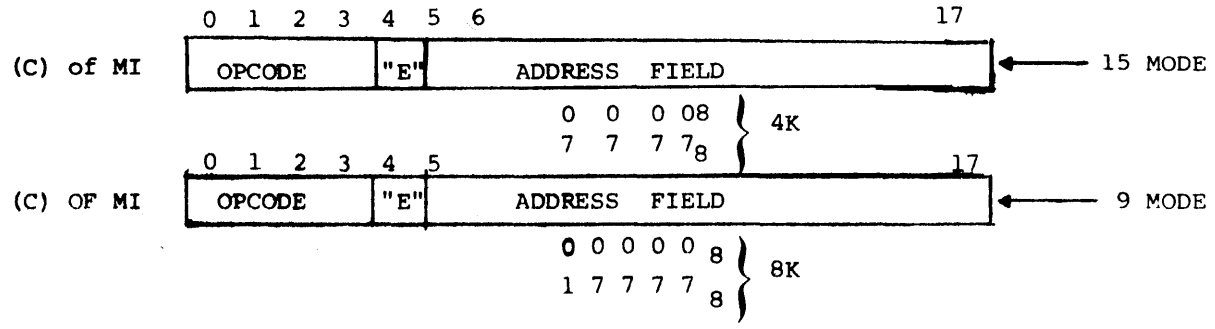




DECODE DAC

GRP 1 → MI
GRP 2 → IR
SMW - START MEMORY WRITE

MEMORY REFERENCE INSTRUCTION



E FIELD

	BITS 4 & 5		
DIRECT ADR	0	0	Determines addressing mode Address field is the effective address.
INDEX ADR	0	1	Index REG(XR) + PC 1-5, m16-17 is the effective address.
INDIRECT ADR	1	0	PC 1-5, M16-17 is the address of the effective address.
INDEXED INDIRECT ADR	1	1	PC 1-5, M16-17 is the address of an address. CPU reads (C of the address into MI (Address of the address) then adds the contents of the index register to this value the result is the effective address

BITS 4 5

0 0 — Direct Addressing
- 15 mode 4K
- 9 mode 8K

0 1 Index Addressing
- 15 mode 128K
- 9 mode can't use index addressing

1 0 Indirect Addressing
- 15 Mode 32K
- 9 Mode 32K

MEMORY REFERENCE INSTRUCTIONS
"15 MODE"

LAC 500 - Direct Addressing - 200500

LAC 500,X - Index Addressing - 210500

LAC * 500 - Indirect Addressing - 220500

LAC * 500,X - Indirect Index Addressing - 230500

FETCH MAJOR STATE

TS01-MI [200500] → Decode Group 1 "DIRECT ADDRESSING"

IR [20] → Decode Group 3,5

START MEMORY READ (SMR)

PC 1-5 M16-17 MO,OA (effective ADR)

TS02 MREQ

TS03 Wait for Rd RST (Stop CLK hangs CPU INTS03 & Ø3 (C) of 500 loaded into MI

1 EXECUTE

EXECUTE MAJOR STATE

TS01 PC JAM TO MO (SET FETCH)
START MEMORY RD
(GET THE NEXT INSTRUCTION)

TS02 MREQ

MI → AC

TS03 Wait for RD RST (stop CLK hangs CPU in TSO3&Ø3) instruction loaded into MI register

1 FETCH

Always enter the FETCH majorstate with an instruction in the MI Register

FETCH MAJOR STATE

TS01-MI [210500] DECODE GRP1 "INDEX ADDRESSING"

IR [21] DECODE GRP3,5

START MEMORY READ (SMR)
PC 1-5, MI 6-17 PLUS XRO-17 = EFFECTIVE ADR

TS02 MREQ

TS03 WAIT for RD RST (STOP CLK - HANGS CPU IN TS03 * Ø3) (C)
OF EFFECTIVE ADDRESS LOADED INTO MI

1 ⟶ EXECUTE

EXECUTE MAJOR STATE

TS01 PC JAM MO (SET FETCH)
START MEMORY RD
(GET THE NEXT INSTRUCTION)

TS02 MREQ
MI ⟶ AC

TS03 WAIT FOR RD RST (STOP CLK HANGS CPU TS03 * Ø3)
INSTRUCTION LOADED INTO MI REGISTER

1 ⟶ FETCH

ALWAYS ENTER THE FETCH MAJOR STATE WITH AN INSTRUCTION IN THE MI REGISTER.

FETCH MAJOR STATE

"INDIRECT ADDRESS"

MI | 220500 | ⟶ DECODE GRP1

TS01

| 22 | ⟶ DECODE GRP 3,5

START MEMORY READ (SMR)

PC 1-5, M16-17-(THE ADDRESS OF THE EFFECTIVE ADDRESS)⟶MO,OA

TS02 MREQ

TS03 WAIT FOR RD RST (STOP CLK-HANGS UP CPU IN TS03*Ø3)
(C) EFFECTIVE ADDRESS NOW IN MI REGISTER

1 ⟶ DEFER

START MEMORY READ (SMR)

DEFER MAJOR STATE

TS01 PC 1-2, M13-17⟶MO,OA

TS02 MREQ

TS03 Wait for RD RST (STOP CLK- HANGS CPU INTS03*Ø3)
DATA TO BE TRANSFERRED TO ACCUMULATOR NOW IN REGISTER

1⟶EXECUTE

EXECUTE MAJOR STATE

START MEMORY READ (SMR)

TS01 PC JAM⟶MO "SET FETCH"

TS02 MREQ
MI⟶AC

TS03 WHAT FOR RD RST (STOP CLK HANGS CPU TS03*Ø3) INSTRUCTION LOADED INTO MI REGISTER

1⟶FETCH

ALWAYS ENTER THE FETCH MAJOR STATE WITH AN INSTRUCTION IN THE MI REGISTER

FETCH MAJOR STATE

"INDIRECT, INDEXED ADDRESSING"

MI [230500] ⟶ DECODE GRP 1

IR [23] ⟶ DECODES GRPS 3,5

TS01

START MEMORY READ (SMR)
PC 1-5, M16-17 MO,OA (THE ADDRESS OF A LOCATION IN CORE)

TS02 MREQ

TS03 WAIT FOR RD RST (STOP CLK - HANGS UP CPU IN TS03*Ø3) (C) of THE CORE LOCATION LOADED INTO THE MI REGISTER

1 ⟶ DEFER

DEFER MAJOR STATE

TS01 PC1-2 MI 3-17

+XRO ——— 17

RESULT = EFFECTIVE ADDRESS ⟶ MO OA

START MEMORY READ (SMR)

TS02 MREQ

TS03 WAIT FOR RD RST (STOP CLK - HANGS CPU TS03 Ø3) DATA TO BE TRANSFERED TO THE ACCUMULATOR IS NOW LOADED INTO THE MI REGISTER

1 ⟶ EXECUTE

EXECUTE MAJOR STATE

TS01 START MEMORY READ (SMR) "SET FETCH"
PC JAM ⟶ MO

TS02 MREQ
MI ⟶ AC

TS03 WAIT FOR RD RST (STOP CLK HANGS CPU TS03 * Ø3)
INSTRUCTION LOADED INTO MI REG.

1 ⟶ FETCH

ALWAYS ENTER THE FETCH MAJOR STATE WITH AN INSTRUCTION IN THE MI REGISTER

6. CENTRAL PROCESSOR INTERFACING WITH MEMORY

OBJECTIVES:

1. Define how the central processor initiates a memory cycle.

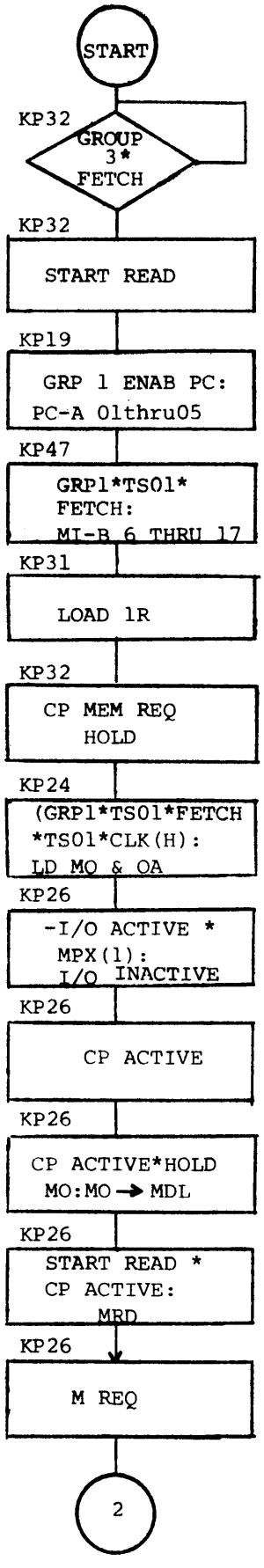
2. List the ways the central processor waits for memory responses
   a) read type instructions
   b) write type instructions

3. Explain how the control signal MO-MDL is made active.

4. Demonstrate ability to use instruction flow chart

5. Explain how an instruction is decoded & processed through the central processor.

MEMORY REFERENCE READ FLOW



FETCH
TS01

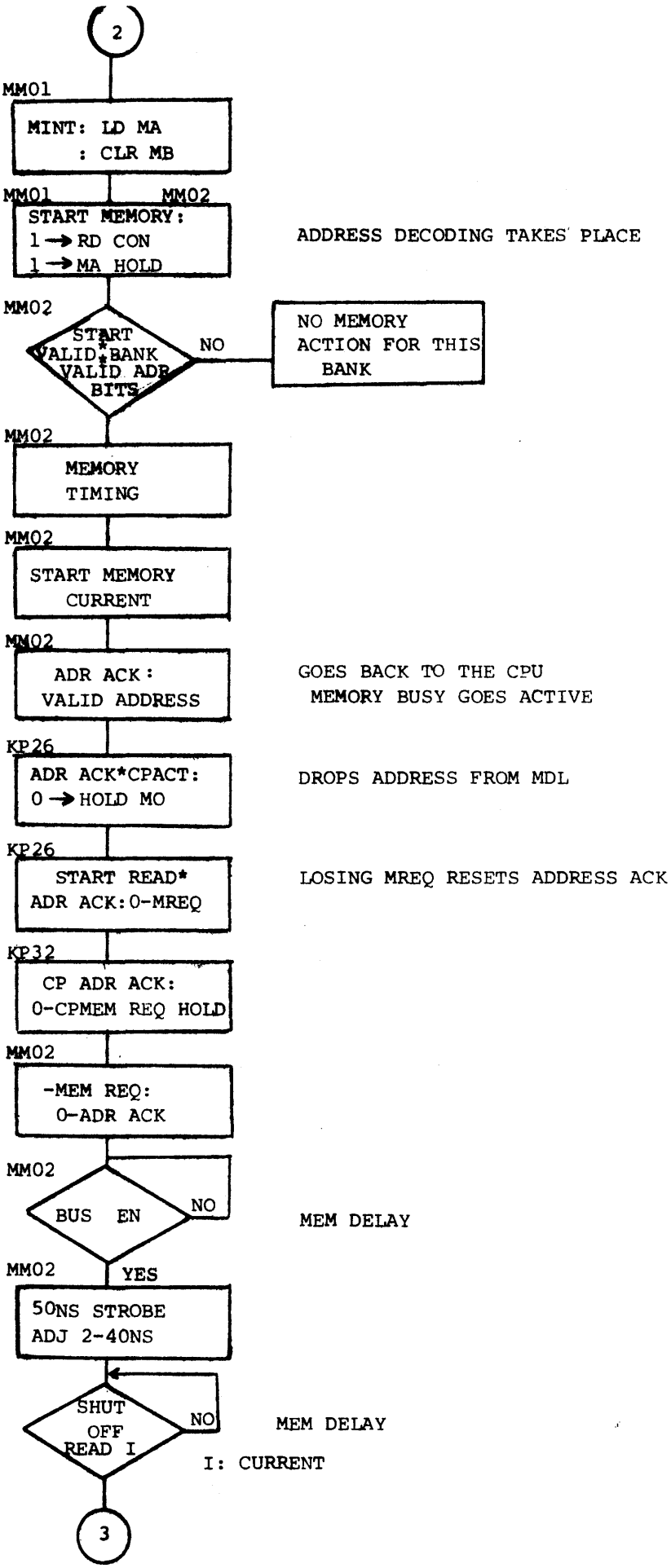
CLK H : LAST HALF OF Ø3
ADDRESS LOADED

ADDRESS ON MDL LINES

COMMAND TO DO
READ/RESTORE MEMORY CYCLE

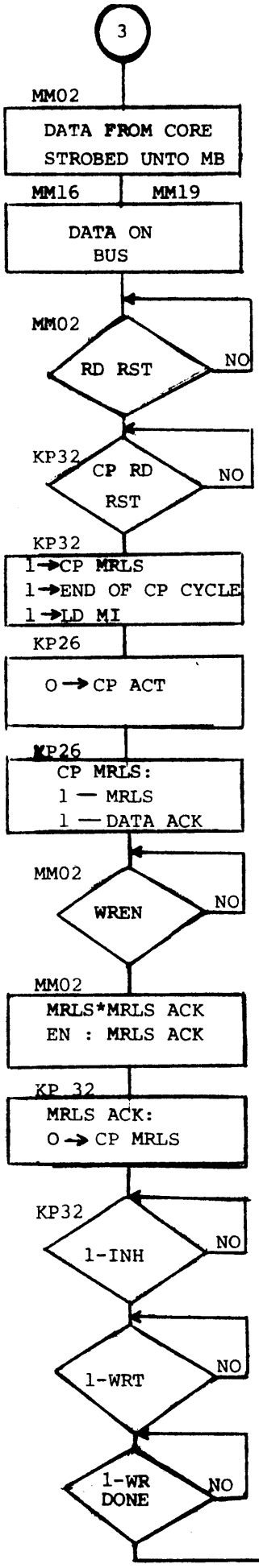
TS02

START THE MEMORY CYCLE


2
MM01
MINT: LD MA
: CLR MB
MM01
MM02
START MEMORY:
1 → RD CON
1 → MA HOLD
ADDRESS DECODING TAKES PLACE
MM02
START
VALID*BANK
VALID ADR
BITS
NO
NO MEMORY
ACTION FOR THIS
BANK
MM02
MEMORY
TIMING
MM02
START MEMORY
CURRENT
MM02
ADR ACK:
VALID ADDRESS
GOES BACK TO THE CPU
MEMORY BUSY GOES ACTIVE
KP26
ADR ACK*CPACT:
0 → HOLD MO
DROPS ADDRESS FROM MDL
KP26
START READ*
ADR ACK:0-MREQ
LOSING MREQ RESETS ADDRESS ACK
KP32
CP ADR ACK:
0-CPMEM REQ HOLD
MM02
-MEM REQ:
0-ADR ACK
MM02
BUS EN
NO
MEM DELAY
MM02
YES
50NS STROBE
ADJ 2-40NS
SHUT
OFF
READ I
NO
MEM DELAY
I: CURRENT
3

TS02 Ø3

TS03Ø3



KP20 CPACT*TS03*START READ*RD RST WAIT: STOP CLOCK HANGS UP CPU UNTIL RD RST OCCURS

CP ACTIVE * Ø3 * TS03 * RD RST: CP RD RST

DATA READ FROM CORE LOADED INTO MI

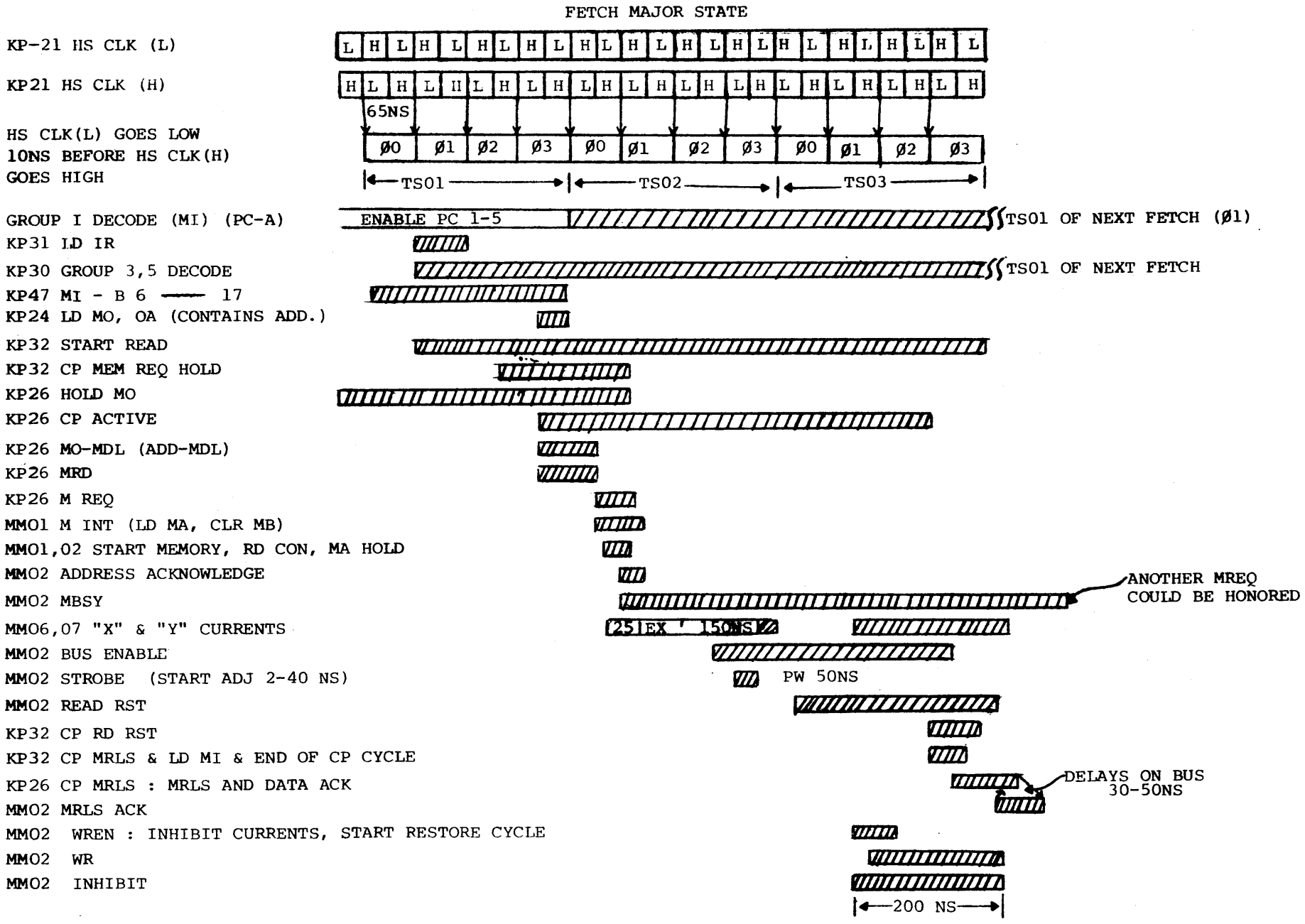
PROC DONE WITH MEMORY CYCLE

← DATA ACK RESETS BUS EN TAKES DATA OFF THE MDL LINES.

0→RD RST
1 BUS DONE

→ 0 - MRLS ACK MM02

WRITE CURRENT RESTORES CONTENTS OF MB BACK INTO CORE.


FETCH MAJOR STATE
KP-21 HS CLK (L)
KP21 HS CLK (H)
65NS
HS CLK(L) GOES LOW
10NS BEFORE HS CLK(H)
GOES HIGH
Ø0 Ø1 Ø2 Ø3 Ø0 Ø1 Ø2 Ø3 Ø0 Ø1 Ø2 Ø3
TS01
TS02
TS03
KP30
KP19
GROUP I DECODE (MI) (PC-A)
ENABLE PC 1-5
TS01 OF NEXT FETCH (Ø1)
KP31 LD IR
KP30 GROUP 3,5 DECODE
TS01 OF NEXT FETCH
KP47 MI - B 6 —— 17
KP24 LD MO, OA (CONTAINS ADD.)
KP32 START READ
KP32 CP MEM REQ HOLD
KP26 HOLD MO
KP26 CP ACTIVE
KP26 MO-MDL (ADD-MDL)
KP26 MRD
KP26 M REQ
MM01 M INT (LD MA, CLR MB)
MM01,02 START MEMORY, RD CON, MA HOLD
MM02 ADDRESS ACKNOWLEDGE
MM02 MBSY
ANOTHER MREQ
COULD BE HONORED
MM06,07 "X" & "Y" CURRENTS
MM02 BUS ENABLE
MM02 STROBE (START ADJ 2-40 NS)
PW 50NS
MM02 READ RST
KP32 CP RD RST
KP32 CP MRLS & LD MI & END OF CP CYCLE
KP26 CP MRLS : MRLS AND DATA ACK
DELAYS ON BUS
30-50NS
MM02 MRLS ACK
MM02 WREN : INHIBIT CURRENTS, START RESTORE CYCLE
MM02 WR
MM02 INHIBIT
200 NS

7. CENTRAL PROCESSOR BLOCK DIAGRAM

OBJECTIVES:

1. Explain what an instruction does

2. Define by central processor block diagram how the instruction or instructions will be processed.

JMS INSTRUCTION OPCODE 10

JMP TO SUBROUTINE

PC INSTRUCTION NEUMONICS

100 - 100500 JMS 500
101

FETCH MAJOR STATE

THE JMS INSTRUCTION IS DECODED. THE EFFECTIVE ADDRESS IS: (000500)

TS01 PC 1-5 M1-6-17→ MO,OA (000500)
SMW-MEMORY COMMAND

TS02 MREQ
DATA- {0 1 2 3 ------- 17 ⟶ MO⟶ MB THEN WRITE
L B U 0 0 100
I A U
N N E
K K R PC

TS03

1→ EXECUTE

EXECUTE MAJOR STATE

TS01 0A + 1→MO, PC 000500 + 1 = 000501→ MO,OA
501 NOW IS THE EFFECTIVE ADDRESS

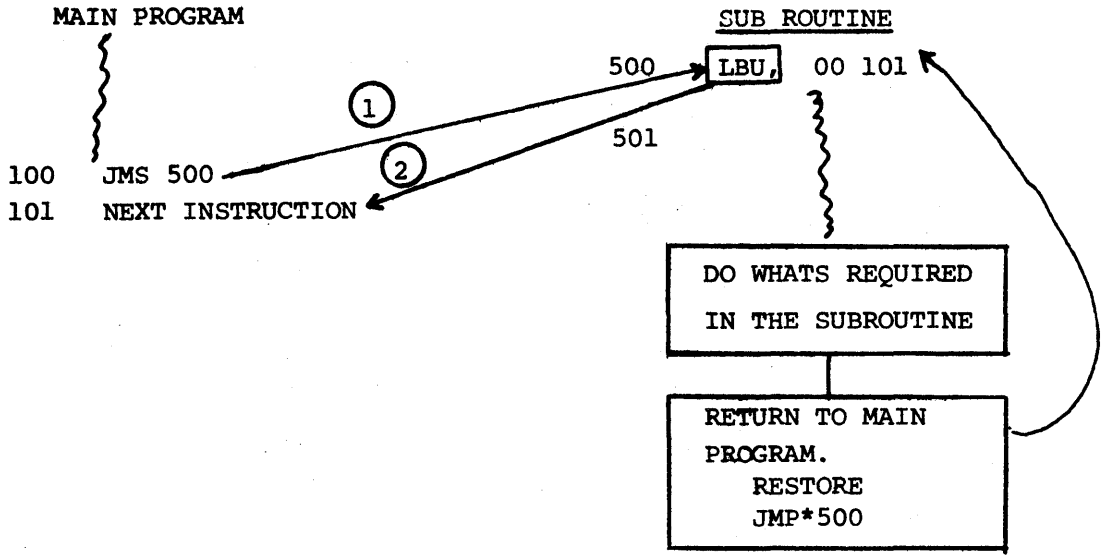
SMR (READ COMMAND - READ NEXT INSTRUCTION
OUT OF MEMORY)

TS02 MREQ

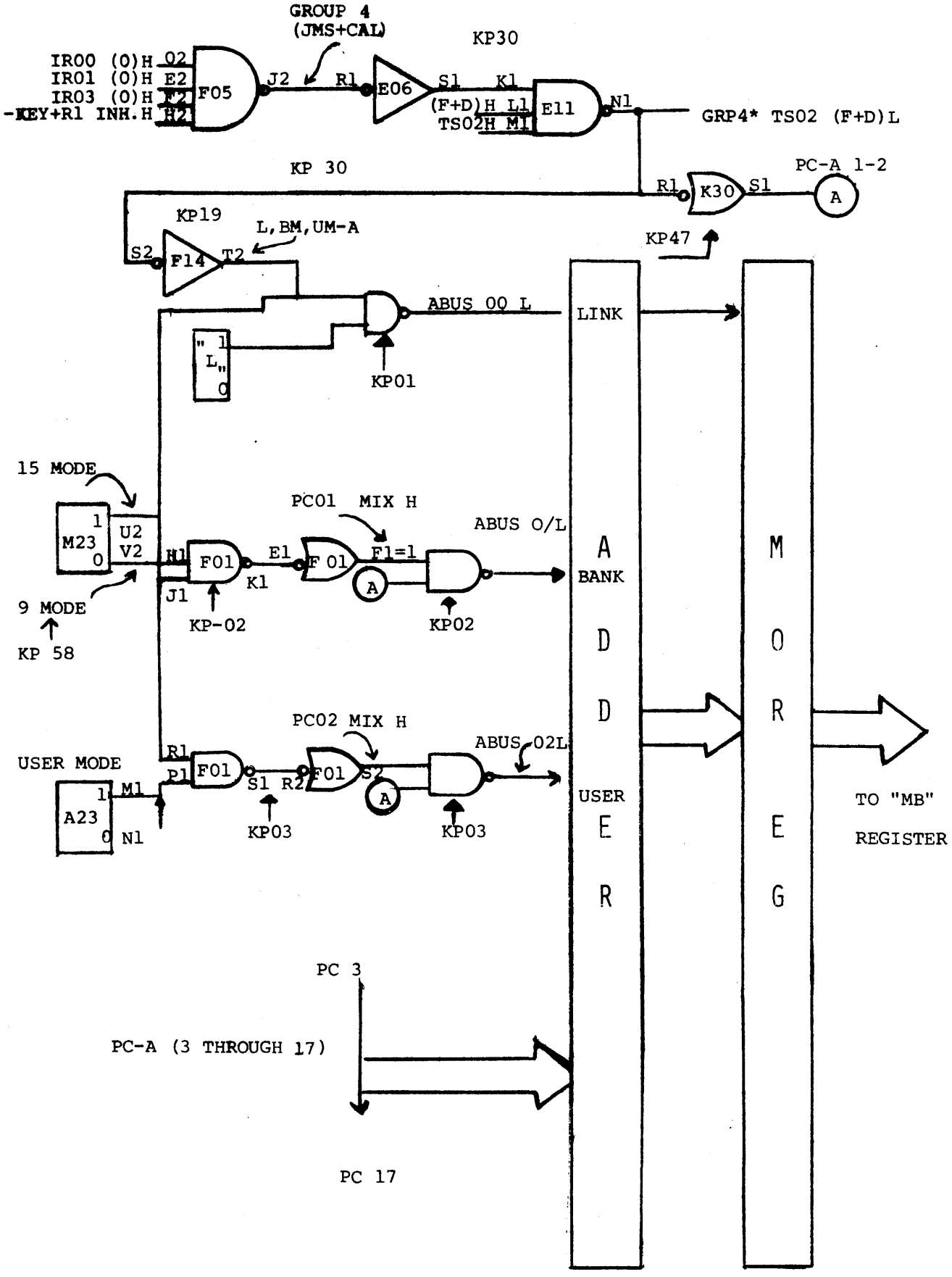
TS03 * Ø3 WAIT FOR READ RESTART - LOAD CPU (MI REGISTER)
WITH THE CONTENTS OF 501 WHICH IS THE NEXT
INSTRUCTION.
BUMP THE PC PC+1→ PC = 502

1→ FETCH

NOW PROGRAM IN SUBROUTINE IS WORKING



NOTE: THE RESTORE INSTRUCTION ENABLES THE RESTORATION OF THE $LINK_{FF}$, $BANK_{FF}$, AND USER $MODE_{FF}$ TO THEIR ORIGINAL CONDITIONS. IF THE PROGRAMMER DOESN'T NEED TO USE THESE THREE BITS, THE RESTORE INSTRUCTION IS ELIMINATED FROM HIS PROGRAM.

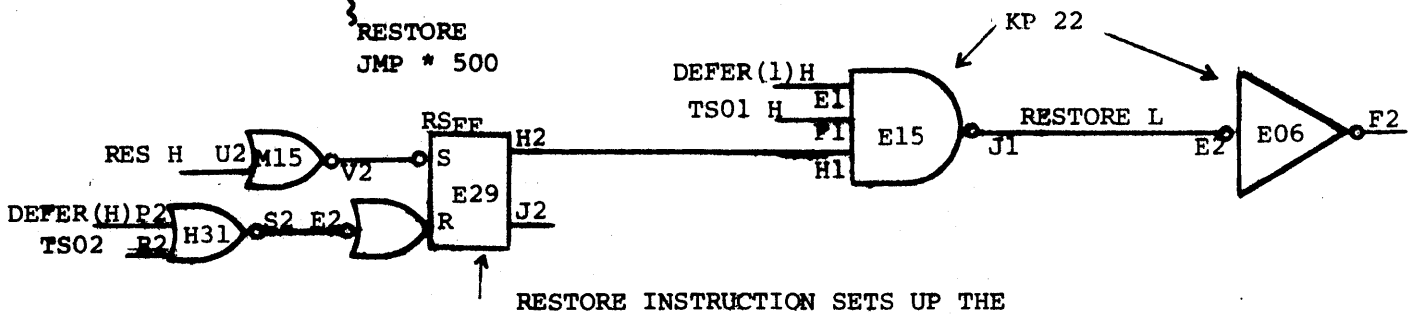


WRITE CONTROL DATA IN THE 1ST LOCATION OF THE SUBROUTINE (L,B,U, AND PC 3-17)

ROUTINE TO GET BACK INTO THE MAIN PROGRAM

500 - L,B,U, AND PC (101 0000 101)

5_8



RESTORE INSTRUCTION SETS UP THE RESTORATION OF L,B,U MODE

EX L=1, B=0, U=1 PC 3-17 = 00101

FETCH MAJOR STATE

TS01 JMP * 500 (PC 1-5,MI 6-17 → MO, OA, AND PC)
SMR

TS02 MREQ

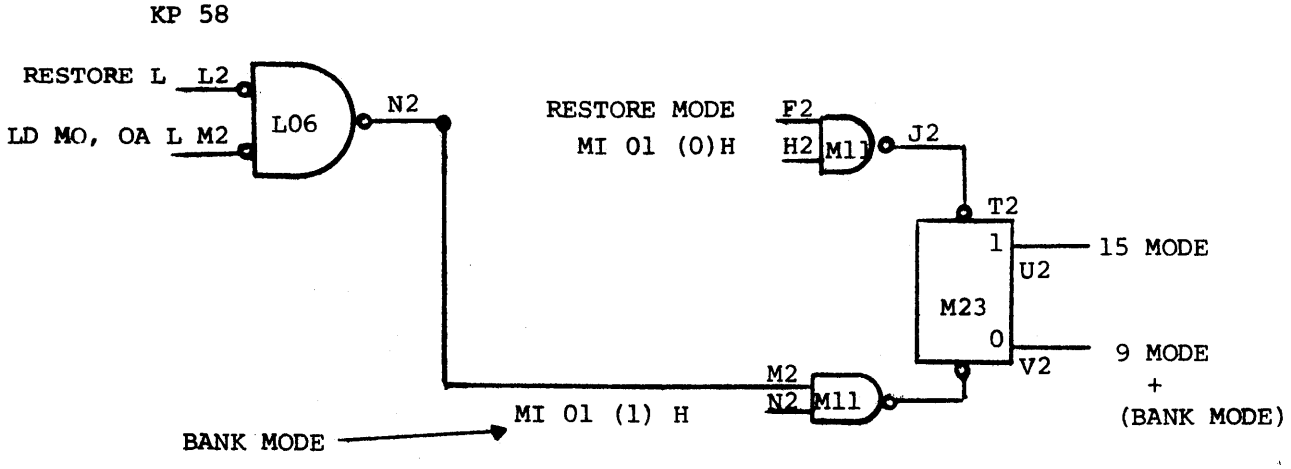
TS03 RD THE CONTENTS OF LOCATION 500 INTO M1 REG.

MI = 101 00101
5_8

1 → DEFER

DEFER MAJOR STATE

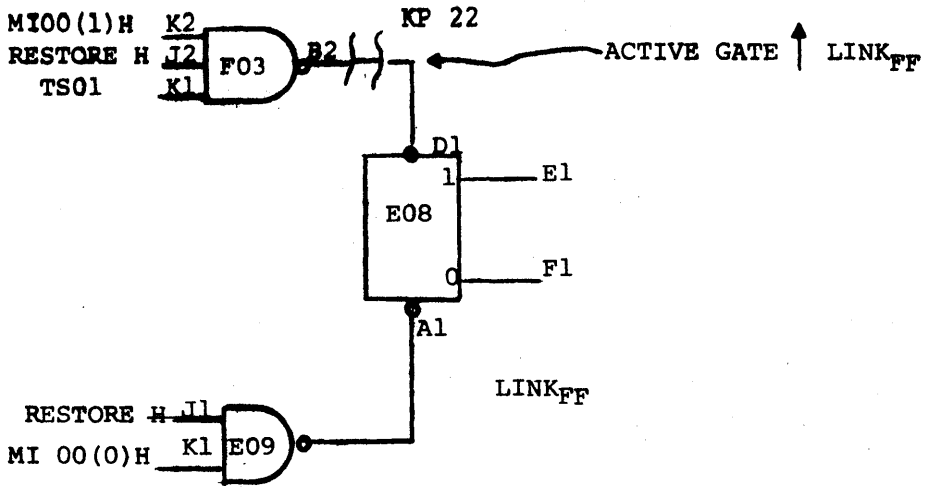
TS01 PC 0-2 +MI 3-17 TO THE OA, MO AND PC
SMR



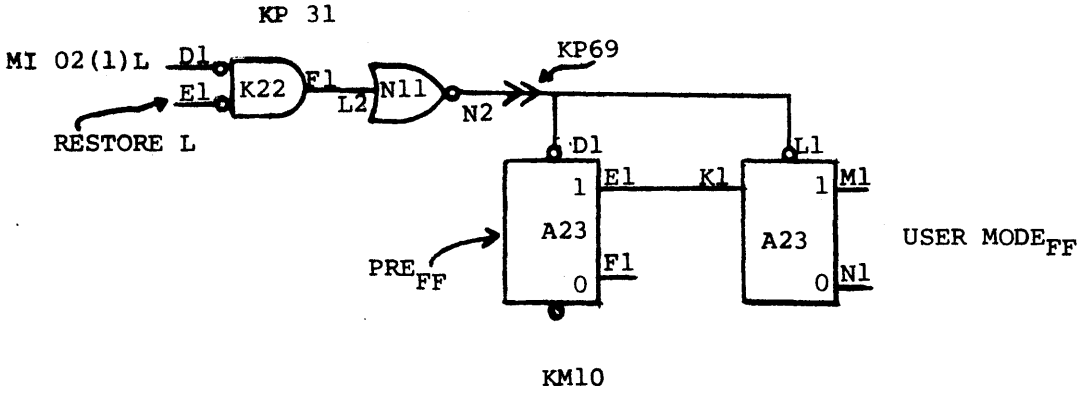
BANK $MODE_{FF}$ WAS INITIALLY ON A ZERO: RESTORE ↓ $MODE_{FF}$

RESTORATION OF THE BANK $MODE_{FF}$

RESTORATION OF THE $LINK_{FF}$



RESTORATION OF THE USER $MODE_{FF}$



DEFER MAJOR STATE

EFFECTIVE ADD (101) BACK TO MAIN PROGRAM

TS02 MREQ

TS03 READ INSTRUCTION, CONTENTS OF 101 INTO THE MI REGISTER

BUMP PC 000101 + 1→ PC

1→ FETCH

XOR INSTRUCTION OPCODE 24

THIS INSTRUCTION READS INTO THE MI REGISTER DATA. THE CONTENTS OF THE MI REGISTER ARE EXCLUSIVELY ORED WITH THE CONTENTS OF THE ACCUMULATOR THE RESULT IS LOADED INTO THE ACCUMULATOR.
IF THE BITS ARE THE SAME, EG. AC =0 * M10=0, RESULT LOADED INTO THE ACCUMULATOR ACO WILL BE A "O". IF THE BITS ARE DIFFERENT ACO WILL BE "1".

FETCH MAJOR STATE

PC 1-5 MI6-17 MO,OA (EFFECTIVE ADDRESS)
SMR (READ COMMAND)
MREQ- WILL BE SENT TO MEMORY AND STARTS THE MEMORY CYCLE

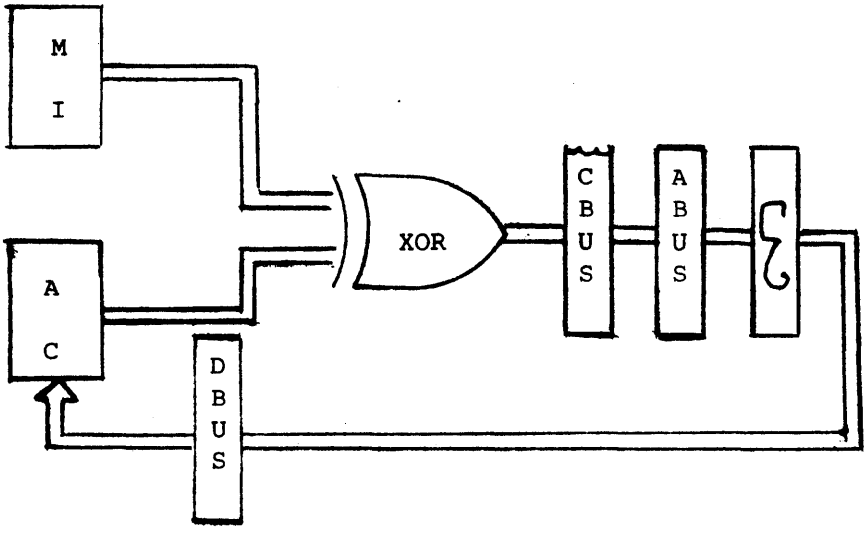
DATA READ INTO THE MI REGISTER

1 →EXECUTE

EXECUTE MAJOR STATE

GET THE NEXT INSTRUCTION

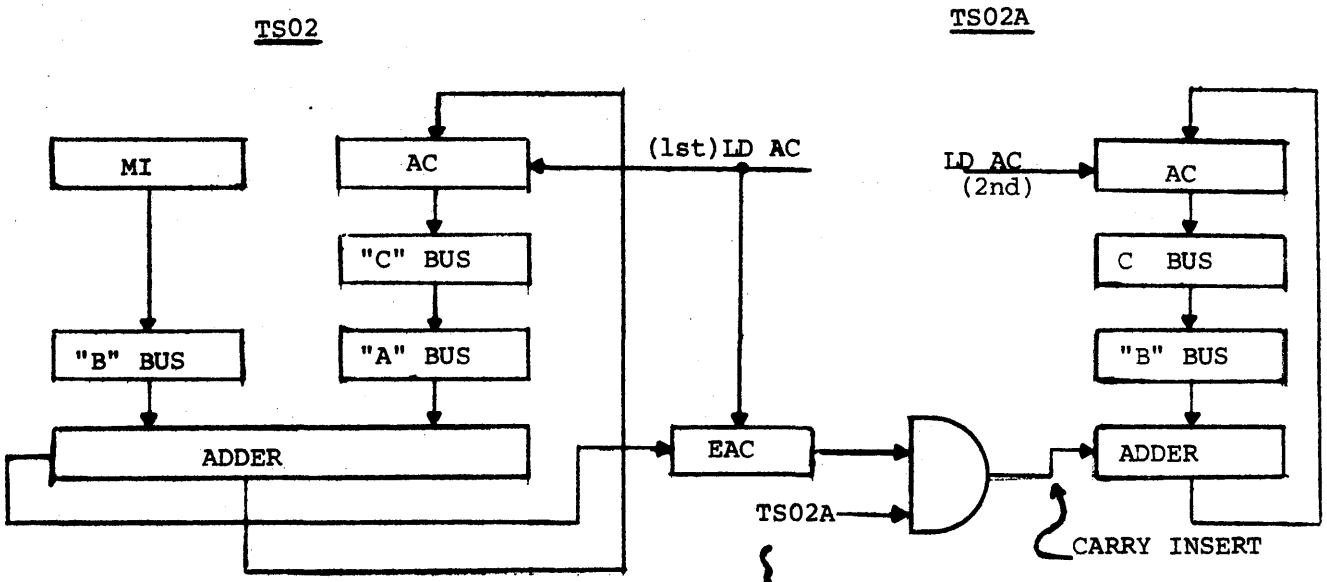
IN TS02 THE "XOR" FUNCTION TAKES PLACE



ADD INSTRUCTION OPCODE 30

THIS INSTRUCTION READS DATA INTO THE MI REGISTER DURING THE FETCH MAJOR STATE.
GOES TO THE EXECUTE MAJOR STATE AND READS INTO THE PROCESSOR THE NEXT INSTRUCTION.

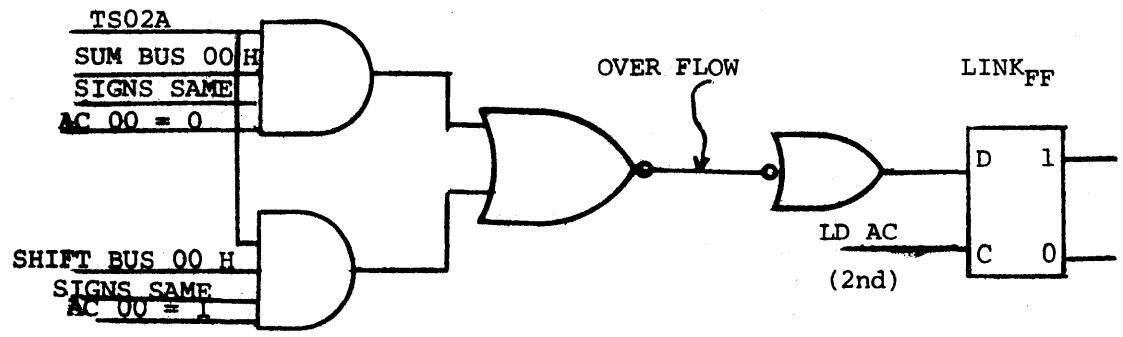
TS02A OF THE EXECUTE MAJOR STATE
OVERFLOW AS A RESULT OF THE ADD IMPLIES SET THE $LINK_{FF}$. THIS MEANS ITS A GOOD IDEA TO CLEAR THE $LINK_{FF}$ BEFORE DOING THE ADD.



TS02 - ADD MI + AC.
THE RESULT IS LOADED INTO THE AC. AT THE SAME TIME SET THE EAC_{FF} IF THE ADD RESULTS IN A CARRY.

TS02A
BRING THE CONTENTS OF THE AC THROUGH THE ADDER IF EAC_{FF} IS SET ADD +1 TO THE CONTENTS OF THE AC THE RESULT LOADED INTO THE AC.

OVERFLOW CHECK



TAD INSTRUCTION OP CODE 34

TWO'S COMPLEMENT ADD. THIS MEANS NEGATIVE NUMBERS ARE ENTERED IN TWO'S COMPLEMENT FORM. AN ARITHMETIC CARRY COMPLEMENTS THE $LINK_{FF}$.

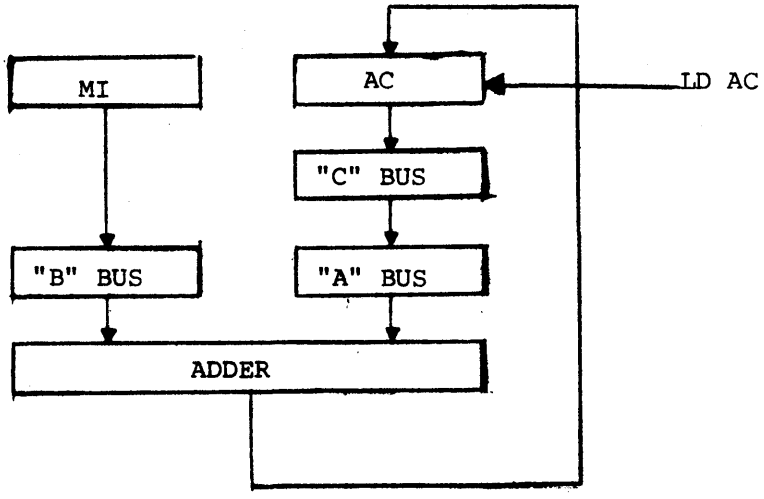
TO CHECK FOR OVERFLOW CONDITION AS A RESULT OF A TAD INSTRUCTION MUST BE DONE BY SOFTWARE.

THIS INSTRUCTION READS DATA INTO THE MI REGISTER DURING THE FETCH MAJOR STATE

GO TO THE EXECUTE MAJOR STATE AND READ THE NEXT INSTRUCTION.

EXECUTE MAJOR STATE

TS02



MI + AC ⟶ AC

XCT INSTRUCTION OPCODE 40

THIS INSTRUCTION GOES THROUGH THE FETCH MAJOR STATE. PURPOSE TO READ AN INSTRUCTION FROM MEMORY AND OPERATE ON IT. DOES NOT CHANGE THE PC. THAT IS, THE NEXT INSTRUCTION FOLLOWING THE XCT INSTRUCTION WILL BE OPERATED ON.

```
100 XCT 4000          XCT INSTRUCTION IS A
101 LAC 500           1 STEP SUBROUTINE
```

```
4000 DAC * 0400
```

FETCH MAJOR STATE

TS01 PC 1-5 MI 6-17→MO,OA (MO/OA=4000)

TS02 MREQ

TS03 READ CONTENTS OF 4000 INTO MI REGISTER
ALSO SET SCT REM (KP48,→KP24). THIS PREVENTS PC+1 TO PC FROM OCCURRING. PC STILL AT 101.
1→FETCH

FETCH MAJOR STATE

TS01 DECODE DAC * 0400 AND OPERATE ON IT
SMR

TS02

TS03 READ CONTENTS OF 400 INTO MI
1→DEFER

DEFER MAJOR STATE

TS01 PC 1-2, MI 3-17→MO,OA (EFFECTIVE ADDRESS)
SMW

TS02 AC→MO→MEMORY

TS03
1→EXECUTE

EXECUTE MAJOR STATE

TS01 PC JAM→MO (MO=0000101)
SMR (READ THE NEXT INSTRUCTION)

TS02

TS03 READ NEXT INSTRUCTION→MI REGISTER
1→FETCH

MEMORY REFERENCE INSTRUCTIONS

ISZ INSTRUCTION OPCODE 44

INCREMENT AND SKIP IF ZERO

FETCH MAJOR STATE

PC 1-5 MI 6-17 MO, OA→(EFFECTIVE ADDRESS)
SMR - (READ COMMAND)
MREQ WILL BE SENT TO MEMORY AND STARTS THE MEMORY CYCLE

CONTENTS OF THE EFFECTIVE ADDRESS LOCATION ARE LOADED INTO THE MI REGISTER IN THE CPU
1 →INC

INCREMENT MAJOR STATE

OA→MO (EFFECTIVE ADDRESS)
SMW (WRITE COMMAND)
MREQ WILL BE SENT TO MEMROY AND START THE MEMORY CYCLE.

MI + CARRY INSERT (+1)→MO
SEND THE UPDATED DATA AND LOAD IT BACK INTO THE EFFECTIVE LOCATION IN MEMORY

SKIP:CARRY=1
1→EXECUTE

$\overline{SKIP}$: CARRY=0
(SET FETCH)

GO TO NEXT INSTRUCTION
(PC+1 PC, MO) + (PO JAM→MO)=(EFFECTIVE ADDRESS)
SMR (READ COMMAND)
MREQ WILL BE SENT TO MEMORY AND START THE MEMORY CYCLE.

CONTENTS OF THE EFFECTIVE ADDRESS LOCATION ARE LOADED INTO THE MI REGISTER IN THE CPU

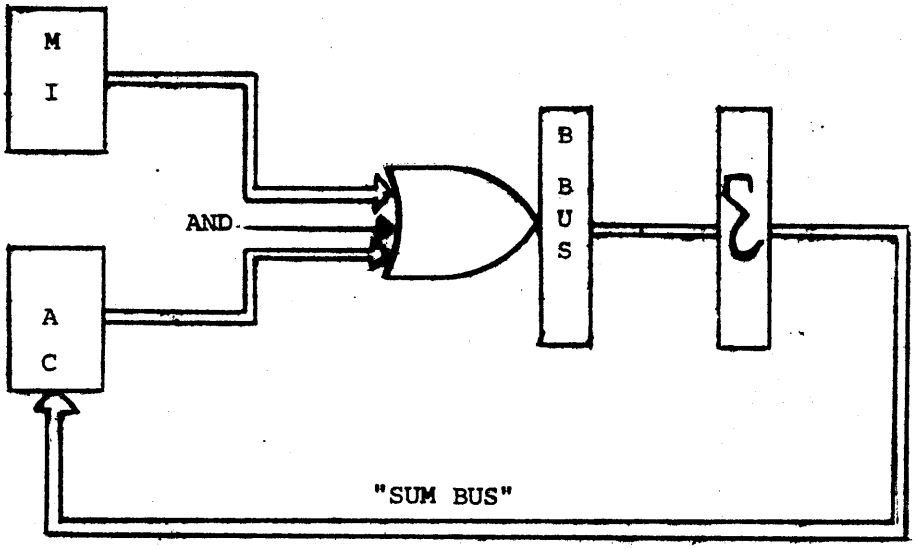
1→FETCH
INSTRUCTION IN MI REGISTER

AND INSTRUCTION OP CODE 50

THIS INSTRUCTION READS DATA INTO THE MI REGISTER DURING THE FETCH MAJOR STATE. GO TO THE EXECUTE MAJOR STATE AND READ IN THE NEXT INSTRUCTION.

EXECUTE MAJOR STATE

TS02



MI 0 = 1 * AC 0 = 1 : LD AC WITH A "1"
MI 1 = 0 * AC 1 = 1 : LD AC WITH A "0"

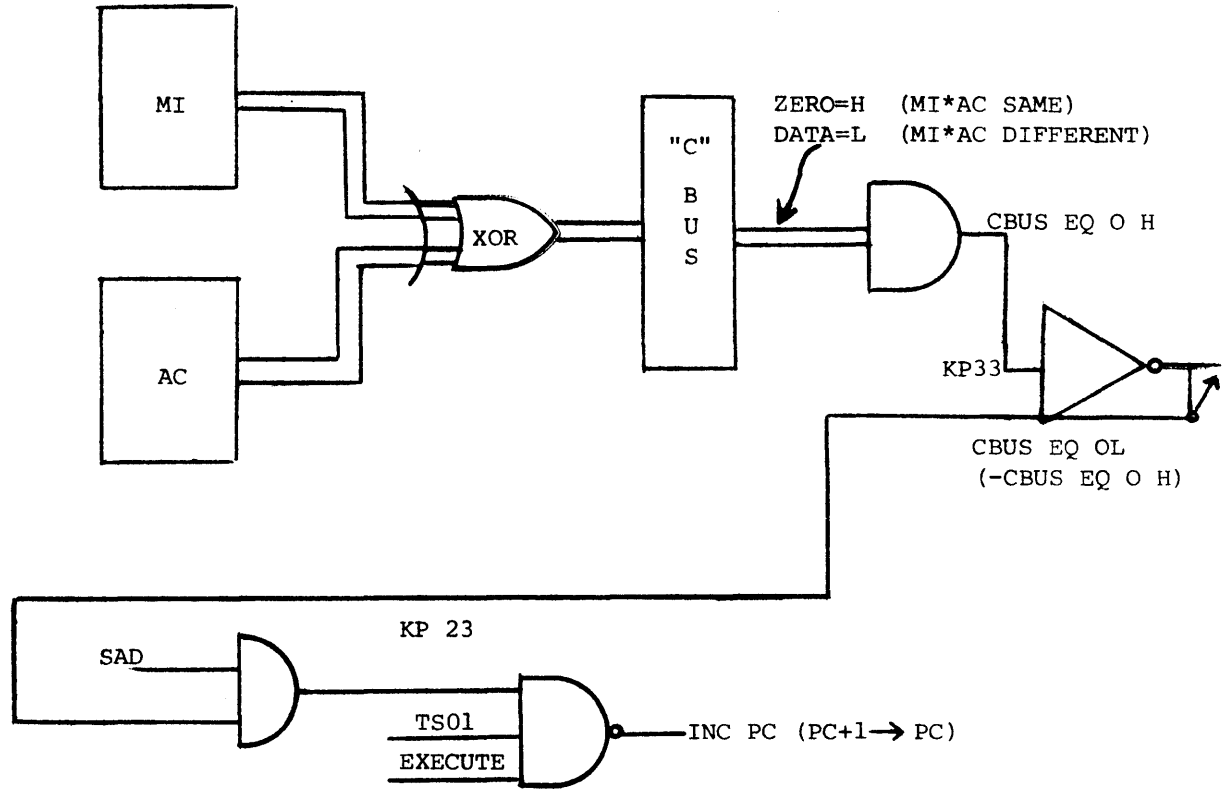
AND INSTRUCTION CAN BE USED AS A MASK

SAD INSTRUCTION OPCODE - 54

THIS INSTRUCTION READS DATA INTO THE MI REG DURING THE FETCH MAJOR STATE.
IF THIS DATA IS DIFFERENT THEN THE CONTENTS OF THE ACCUMULATOR SKIP THE NEXT INSTRUCTION.
(PC+1→PC-MO IN TS01 OF THE EXECUTE MAJOR STATE)
IF THIS DATA IS THE SAME AS THE CONTENTS OF THE ACCUMULATOR, DO THE NEXT INSTRUCTION.
(PC JAM→MO IN TS01 OF THE EXECUTE MAJOR STATE)

EXECUTE MAJOR STATE

TS02



SKIP CONDITION

PC→A

CARRY INSERT: LD PC AND MO

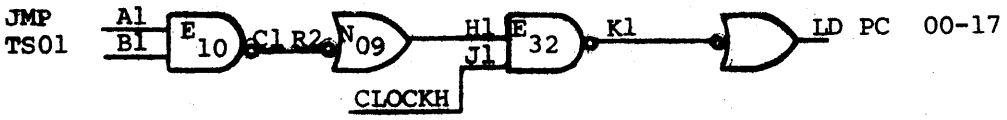
JMP INSTRUCTION OPCODE 60

THIS INSTRUCTION MODIFIES THE PC.
FETCH MAJOR STATE:

TS01 PC 1→5 M16-17 →MO, OA, AND PC

LOAD PC

KP 24



SMR (READ CYCLE)

TS02 MREQ

TS03 WAIT FOR RD RST (LOADS INSTRUCTION INTO MI REGISTER)

BUMP→PC (PC→A+1)→ PC-
PAGE MODE: LOAD PC 6-17
BANK MODE: LOAD PC 5-17

1→ FETCH

INDEX INSTRUCTIONS: OPCODES - 72, 73

ALLOWS PROGRAMMER TO TRANSFER INFORMATION BETWEEN THE AC, LIMIT REGISTER, AND INDEX REGISTER, ADD A NUMBER CONTAINED IN THE INSTRUCTION ITSELF (±256) TO THE AC, LIMIT REGISTER, OR INDEX REGISTER AND TEST TO DETERMINE IF THE INDEX REGISTER IS GREATER THAN OR EQUAL TO THE LIMIT REGISTER.

MOVE TYPE INSTRUCTIONS

AC = 00
XR = 01
LR = 10

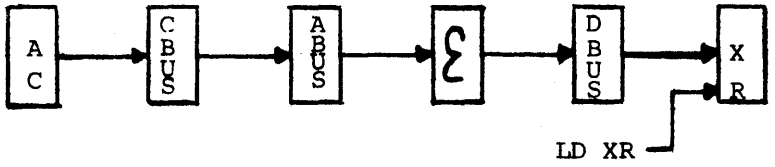
PAX — DECODES ON KP 29

OP CODE 721000 : AC → XR

FROM SOURCE			TO DEST.	
5	6	7	8	
0	0	0	1	

AC → XR

"BUS CONTROL"

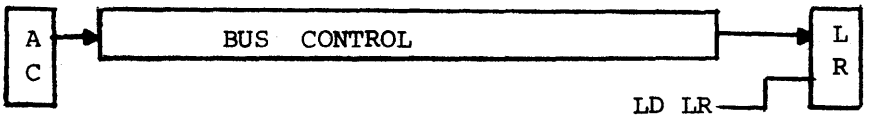


PAL INSTRUCTION

OP CODE 722000 : AC → LR

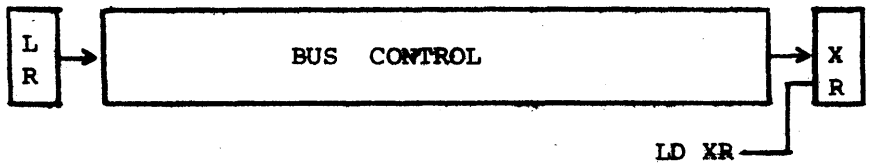
BITS 5 6 7 8
0 0 1 0

AC → LIMIT REGISTER



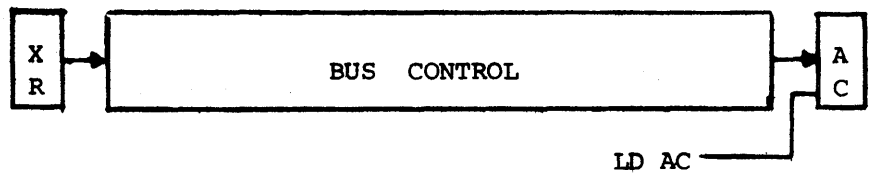
PLX INSTRUCTION

OP CODE 731000 : LR → XR
BITS 5678
1001
LR→XR



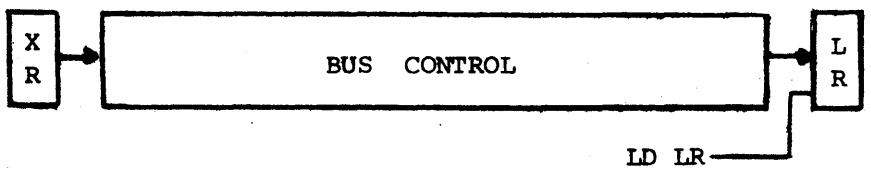
PXA INSTRUCTION

OP CODE 724000 : XR → AC
BITS 5678
0100
XR → AC



PXL INSTRUCTION

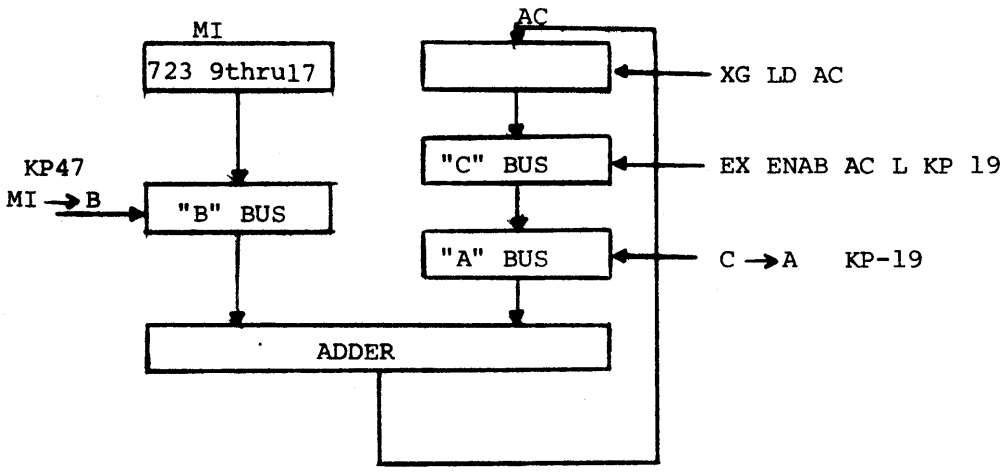
OP CODE 726000 : XR → LR
BITS 5678
0110
XR→LR



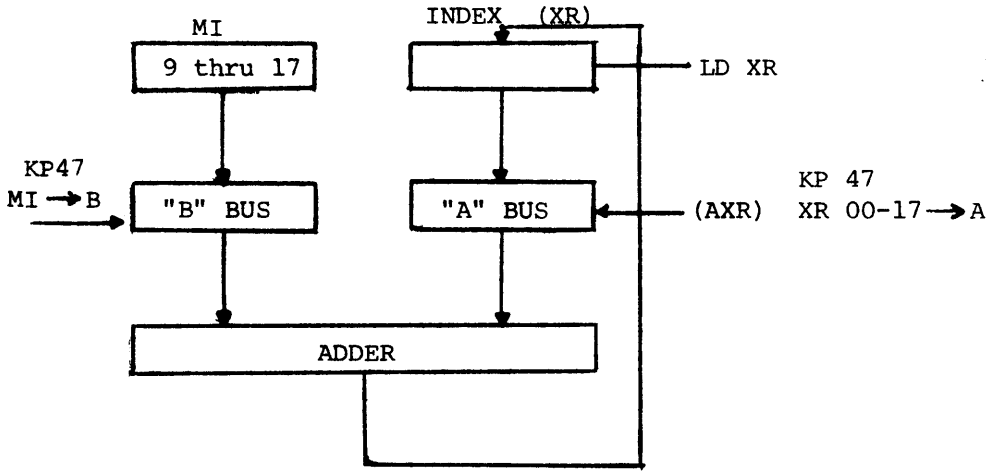
INSTRUCTION 723n

ADD N TO ACCUMULATOR (AAC+ n)

72 3+N (N = 9 BITS)
N = SIGN + 8 BITS.
NEGATIVE NUMBERS IN TWO'S COMPLEMENT FORM



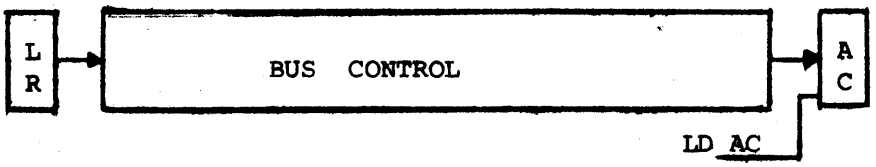
INSTRUCTION: ADD TO INDEX REGISTER AXR + n
737 + N (n = 9 BITS)



PLA INSTRUCTION

730000_8 LR → AC

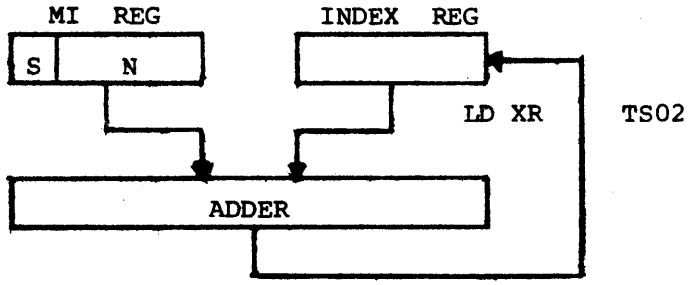
BITS 5678
1000
LR → AC

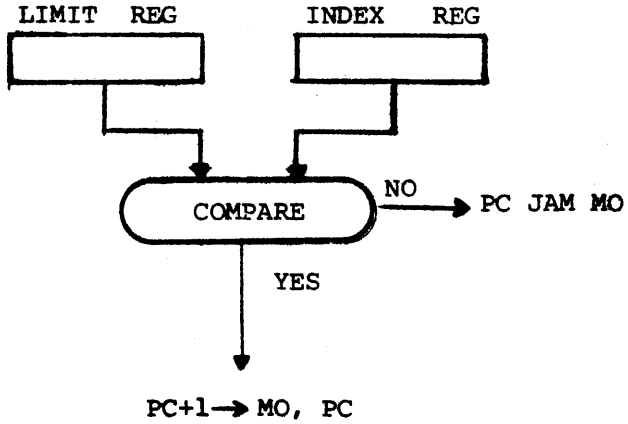


INCREMENT TYPE INSTRUCTIONS

AXS (N) 725 N N = 8 BITS + SIGN

AXS (N) INSTRUCTION IMPLIES ADD OR SUBTRACT THE "N" TO THE INDEX REGISTER AND SKIP IF EQUAL TO OR GREATER THAN THE LIMIT REGISTER

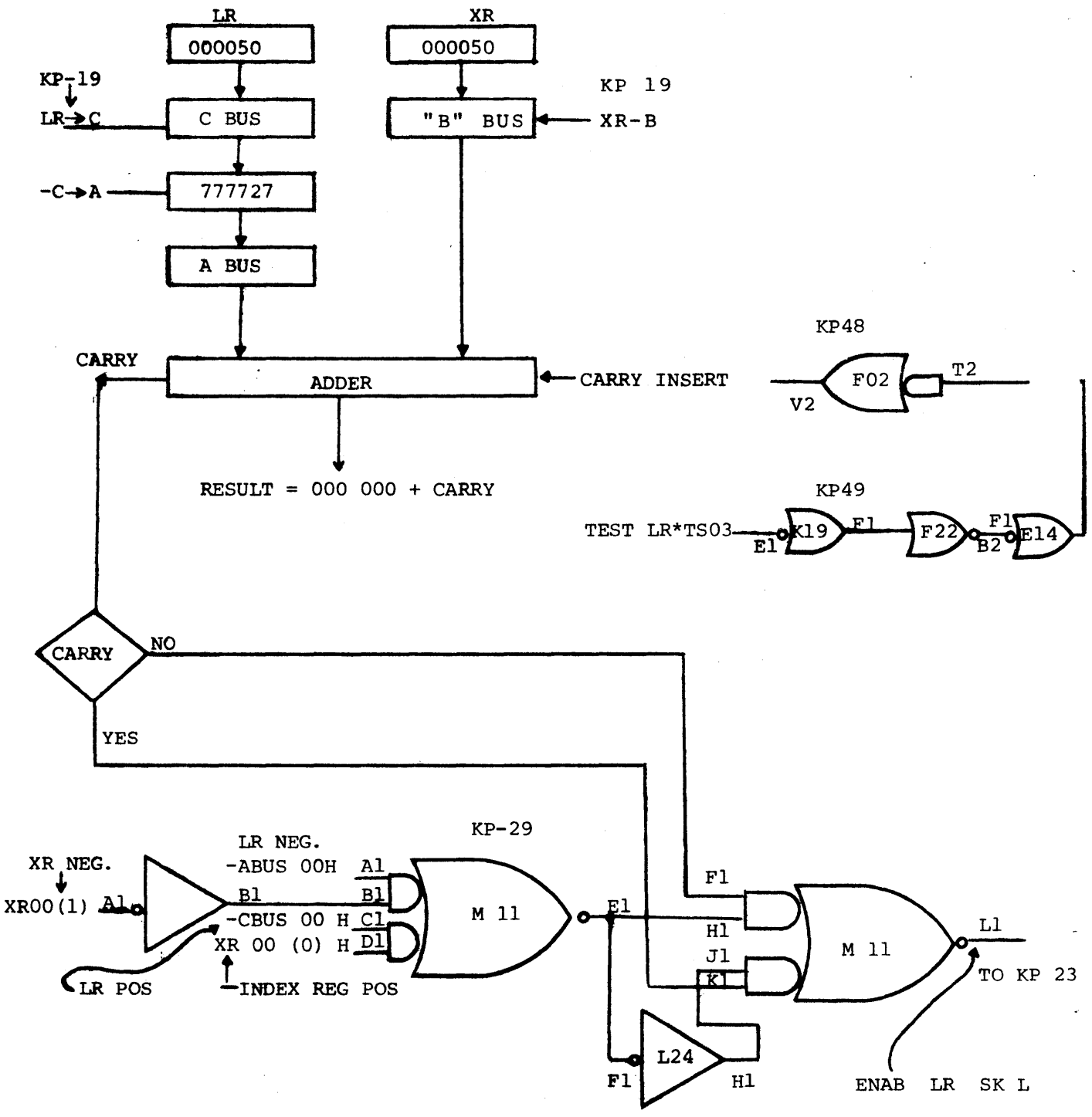




INSTRUCTION: COMPARE OF INDEX REGISTER AND LIMIT REGISTER

AXSn

GIVEN: XR = 000050, LR = 000050



OPERATE INSTRUCTIONS: OPCODE - 74

OPERATE INSTRUCTIONS ARE USED TO SENSE AND/OR ALTER THE CONTENTS OF THE AC AND LINK. THESE INSTRUCTIONS CAN BE MICRO-PROGRAMMED.

								BIT 7=0				
								RAR	RAL			
CLA	CLL	ADDITIONAL ROTATE	0=Or of 1=And of	SNL SZL	SZA SNA	SMA SPA	HLT	RTR	RTL	OAS	CML	CMA
								BIT 7=1				
5	6	7	8	9	10	11	12	13	14	15	16	17

MAIN ORDER OF EVENTS

TS01 - SKIP CONDITIONS
CLEAR AC

TS02 - CLEAR LINK
OPERATION ON AC AND LINK

TS03 - HLT

	← TS01 →		← TS02 →	← TS03 →
ORDER OF EVENTS	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4
LEVEL 1	SNL, SZA, SMA	CLA, CLL	OAS CMA CML / IAC	HLT
LEVEL 2	SZL, SNA, SPA		RAR OR RAL	
LEVEL 3	SKIP		RTR RTL OR SWHA	

NOTE: CAN'T COMBINE INSTRUCTIONS ON DIFFERENT LEVELS IF THEY ARE IN THE SAME COLUMN

CAN'T COMBINE CML AND IAC, (KP22 GATING OF $LINK_{FF}$)

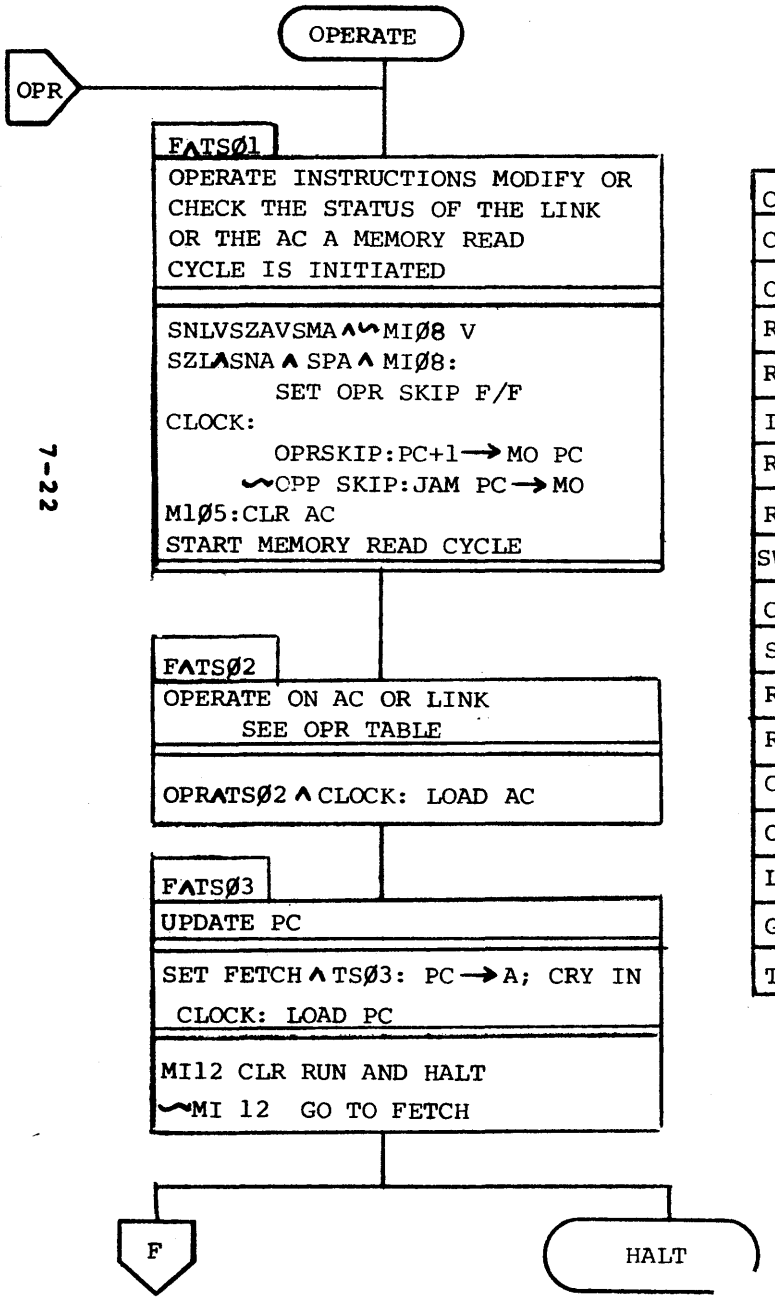
COMBINED ROTATES BECOMES A SWHA OR IAC

SW HA : MI13, MI14 ON A (1)H * MI07 (1)H

IAC : MI13, MI14 ON (1)H * MI07 (0)H

OPERATE INSTRUCTIONS - FLOW CHART KP76

SHOWS A MEMORY CYCLE. THIS IS ONLY TO RETRIEVE THE NEXT INSTRUCTION



OPERATE INSTRUCTIONS

		BAC→C BUS	DSW→C BUS	NC→A BUS	C→A BUS	CARRY INSERT	RAL→D BUS	RAR→D BUS	RTL→D BUS	RTR→D BUS	NO SHIFT→D BUS	SWA→D BUS	MIØ6	MI 16	RESULT
CMA	740001	●		●						●					∿AC→AC
CML	740002													●	∿ L→L
OAS	740004	●	●		●					●					ACV DSW→AC
RAL	740010					●									ACO→L L→Ac17 Acn→Acn-1
RAR	740020														L→Aco Acn→Acn+1 Ac17→L
IAC	740030	●			●	●				●					AC+1→AC
RTL	742010	●			●			●							Ac1→L AcØ-Ac16 L AC16 Acn→Acn-2
RTR	742020	●			●				●						L→AC1 Ac17 Aco Ac16→L Acn→Acn+2
SWHA	742030										●				AcØ-8toAc9-17 AC9-17toAcØ-8
CLL	744000											●			Ø→L
STL	744002											●		●	Ø→L, ∿ L→L
RCL	744010					●						●			CLL ! RAL
RCR	744020						●					●			CLL ! RAR
CLA	750000														Ø→AC
CLC	750001	●		●						●					ALL 1's→AC
LAS	750004		●		●					●					Ø→ AC DSW→ AC
GLK	750010					●									Ø→ AC L→AC17
TCA	740031	●		●		●				●					∿ AC + 1→AC

8. CONSOLE OPERATION

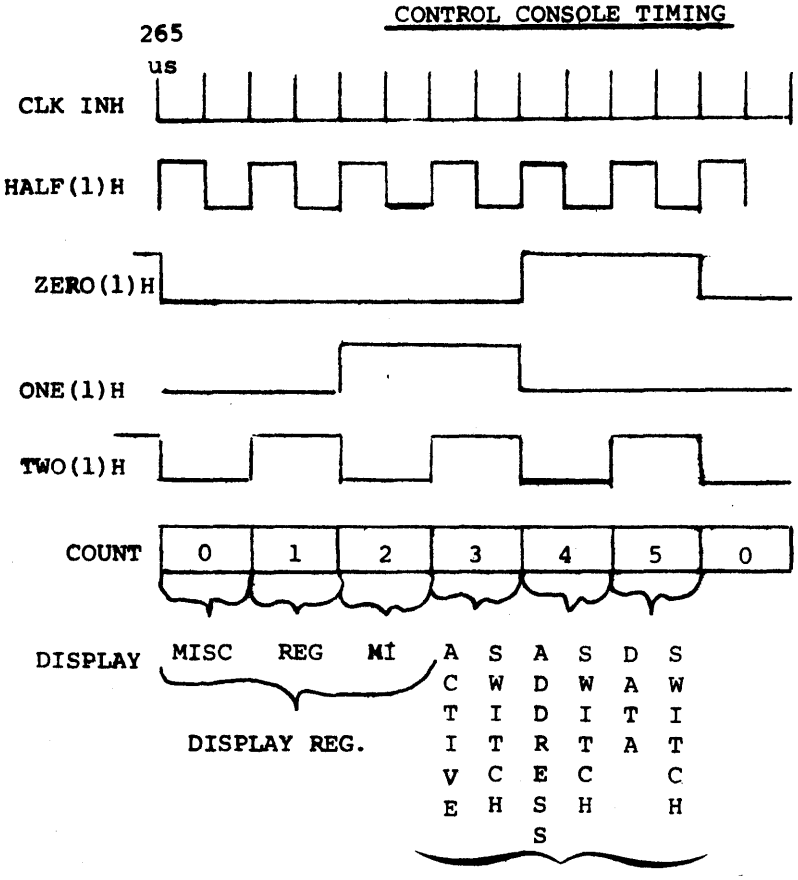
OBJECTIVES:

1. Manually load a program from the console switches.
2. Manually examine program
3. Run the program
4. Explain the functions of the control switches
5. Analyze console display
6. Load and run a hardware paper tape read in.

CONSOLE LOGIC LOCATIONS

1. CONSOLE SWITCH BOARD
2. CONSOLE INDICATOR PANEL
3. KP-18 REGISTER - I BUS
4. KP-34 CONSOLE INTERFACE (CONSOLE TO PROCESSOR)
5. KP-38,39,40,41,42,43 - I BUS BITS 0-23
6. KP-44 CONSOLE CONTROL #1
7. KP-45 CONSOLE CONTROL #2
8. KP-46 MIS CONSOLE LOGIC
9. KP-72 SIGNAL CHART IND BUS
10. KP-73 KEY FLOW
11. FLOW CHART HANDOUT

CONTROL CONSOLE TIMING

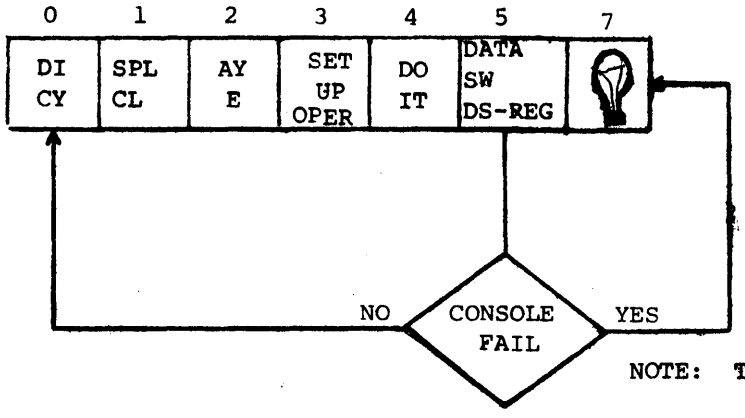


Note Active SW on
IBUS: CNT 3

1. Deposit switch or what ever switch depressed

2. Set 1 of 12ff to be controlled by rotary SW.
   AC selected sets 1st ff and at a count of one the register lights display whats in the AC.

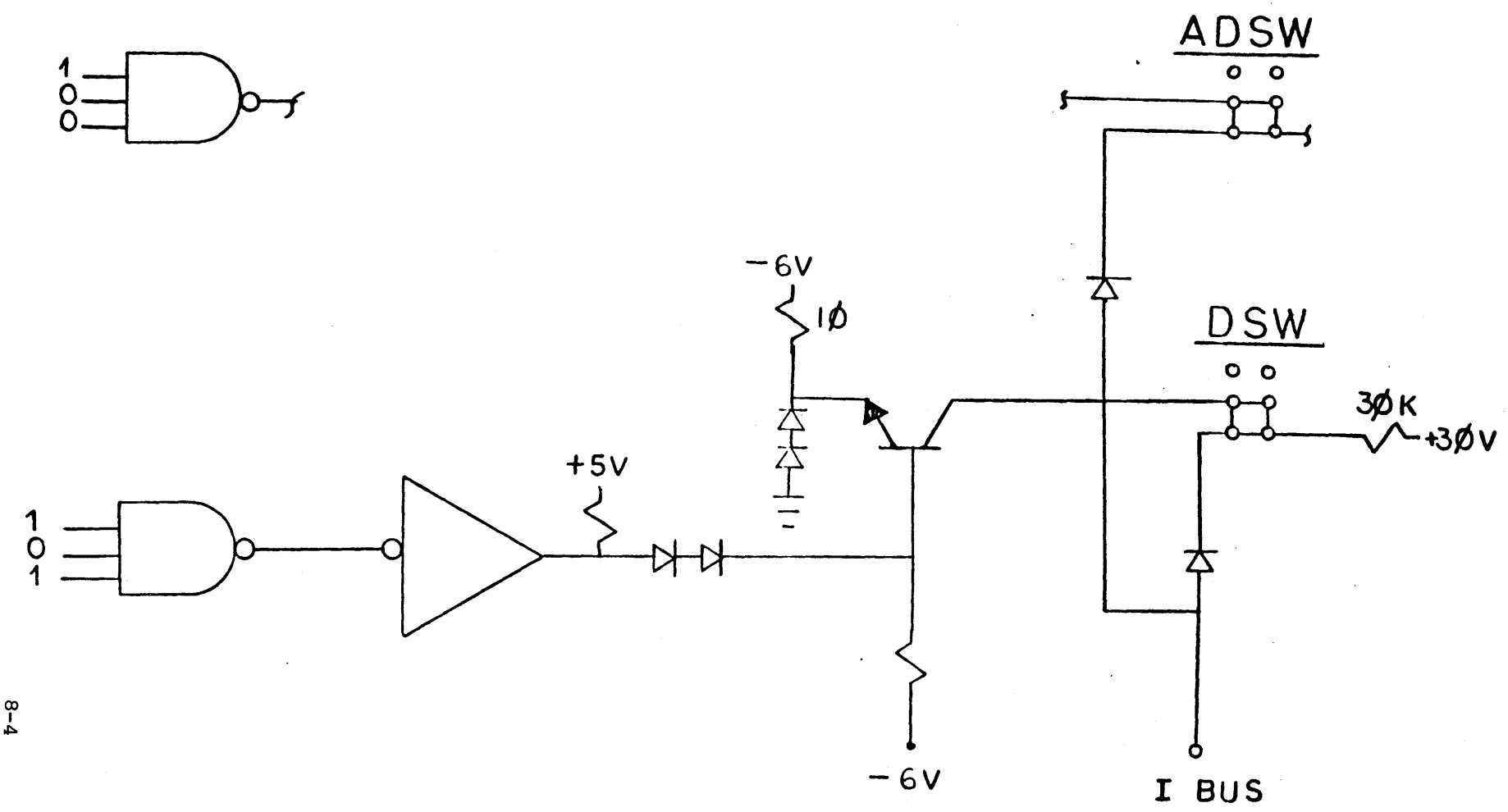
3. NOTE: ADDR SW to IBUS: CNT4



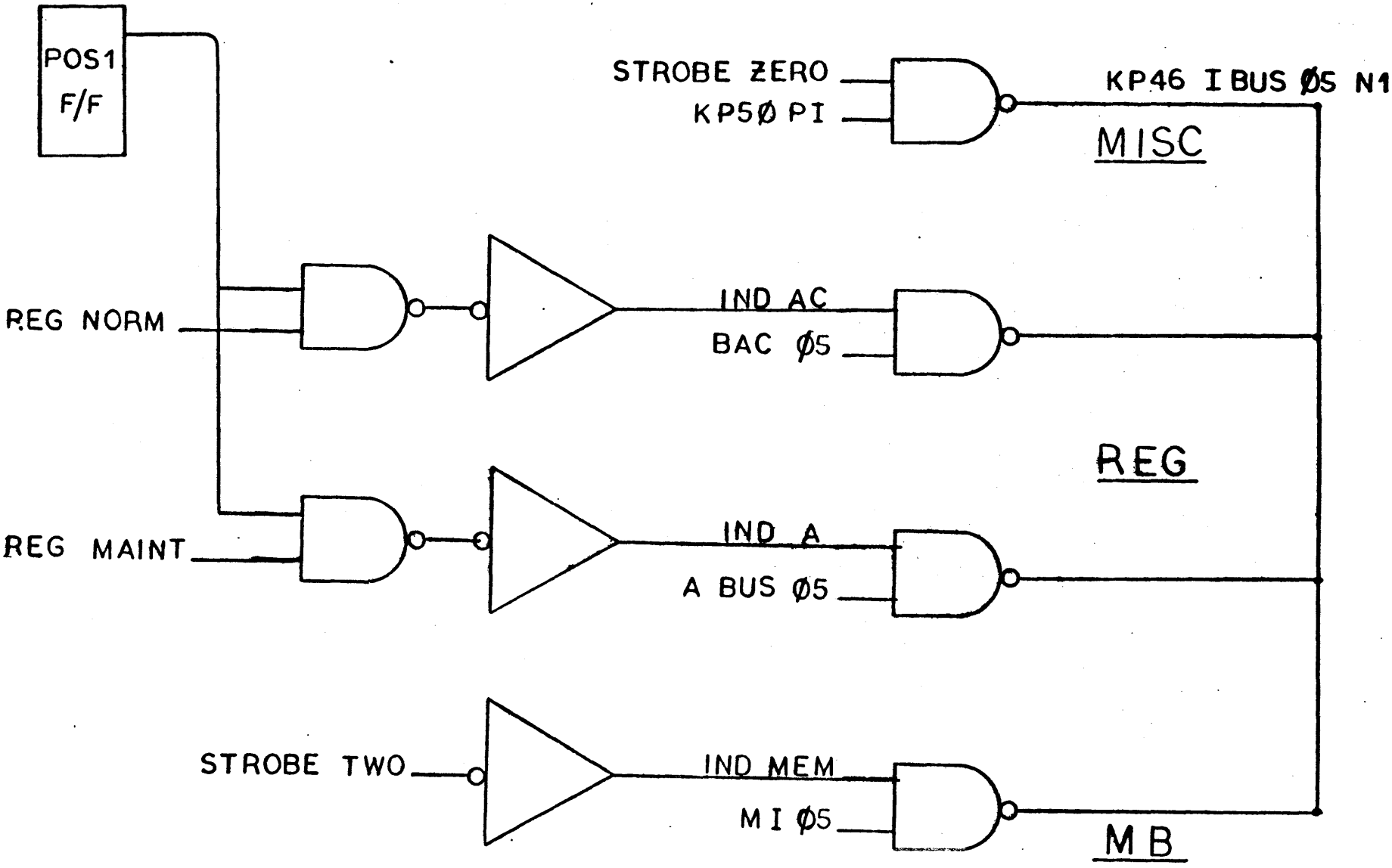
1. Start CPU timing and do the given operation. (deposit)

2. Data switch are stored in DS REG and

NOTE: TURN OFF DISPLAY LIGHTS (PROTECTION)



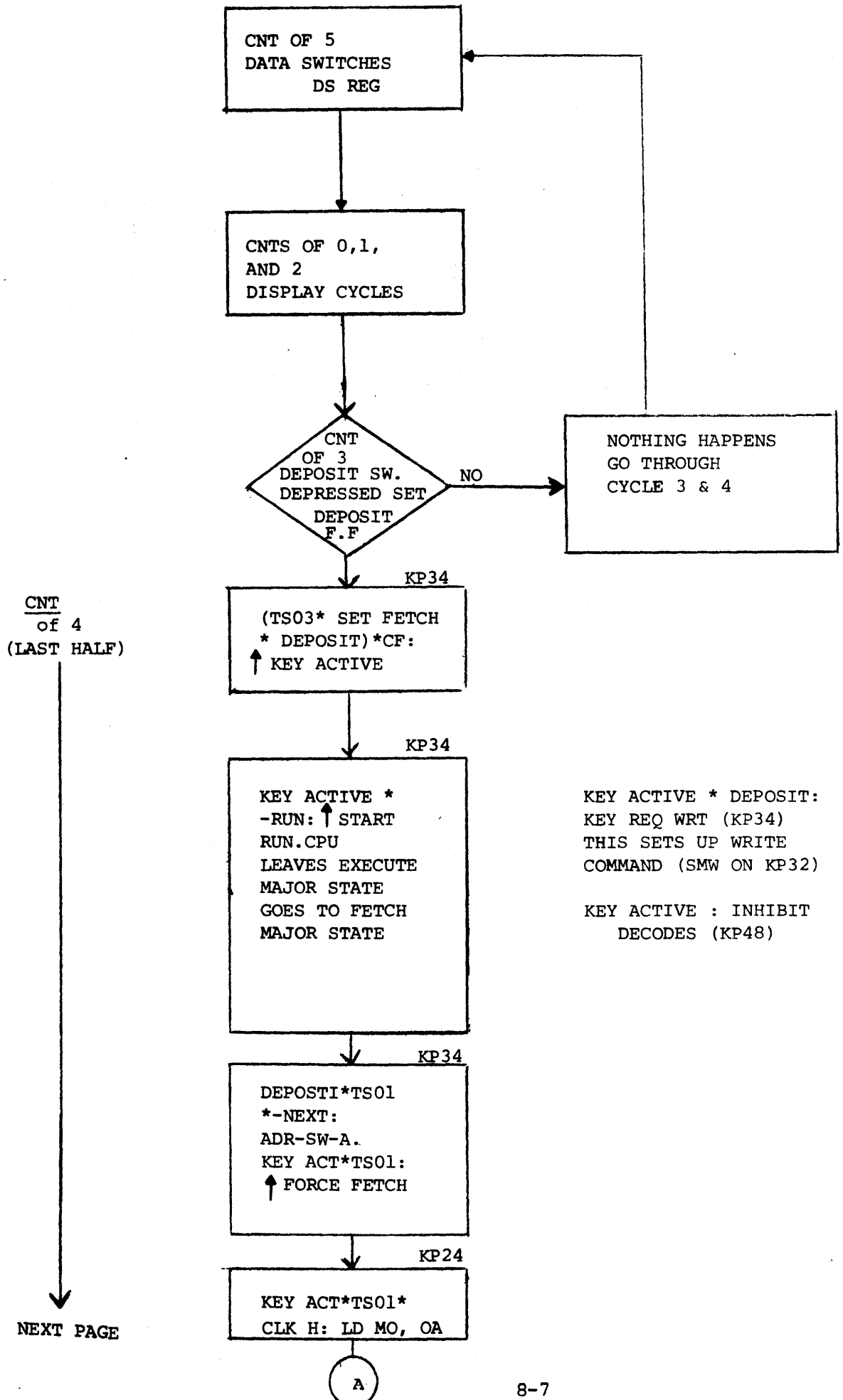
I BUS SWITCHES

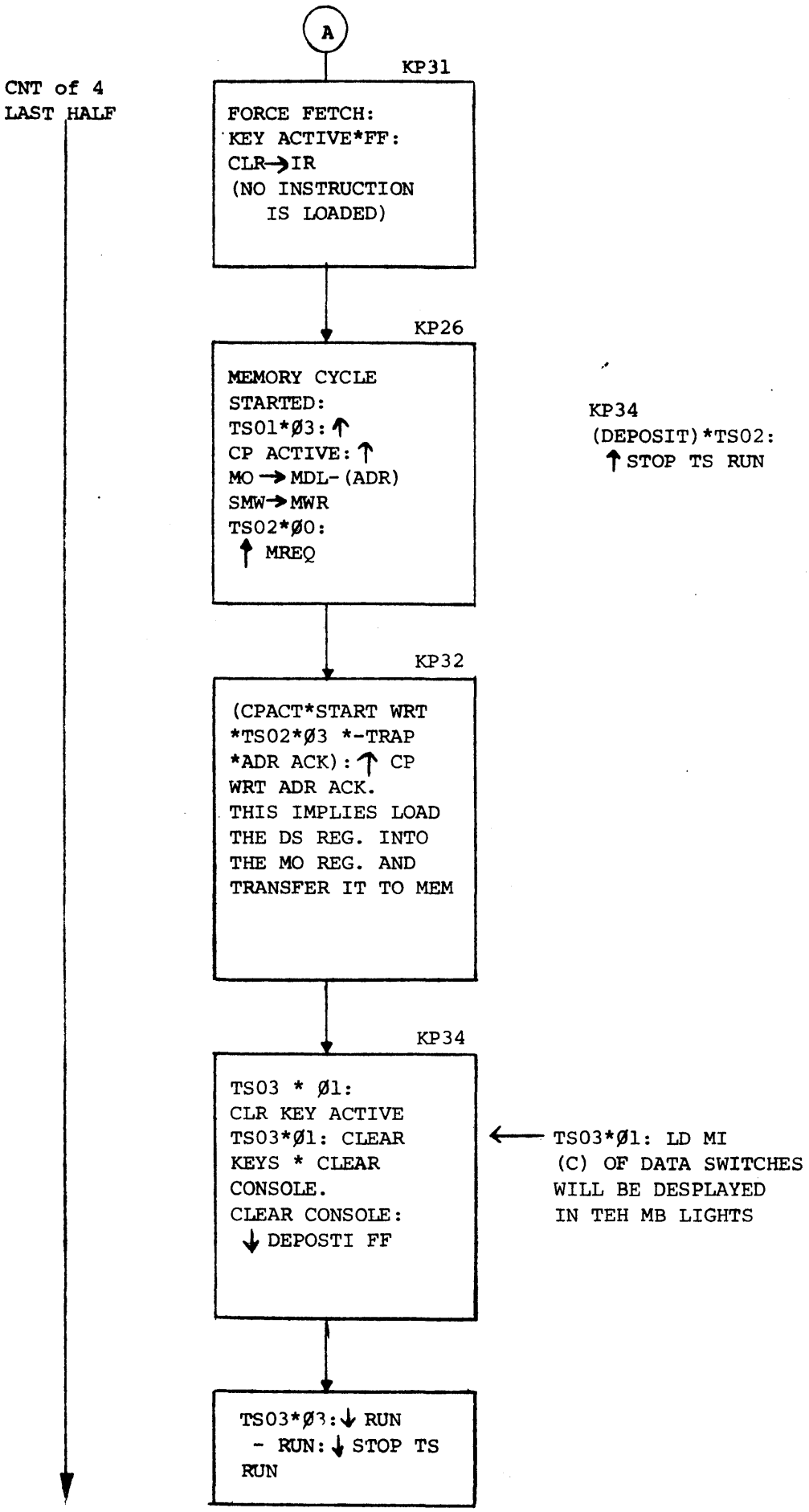

POS1
F/F
STROBE ZERO
KP5Ø PI
KP46 I BUS Ø5 N1
MISC
REG NORM
IND AC
BAC Ø5
REG
REG MAINT
IND A
A BUS Ø5
STROBE TWO
IND MEM
MI Ø5
MB

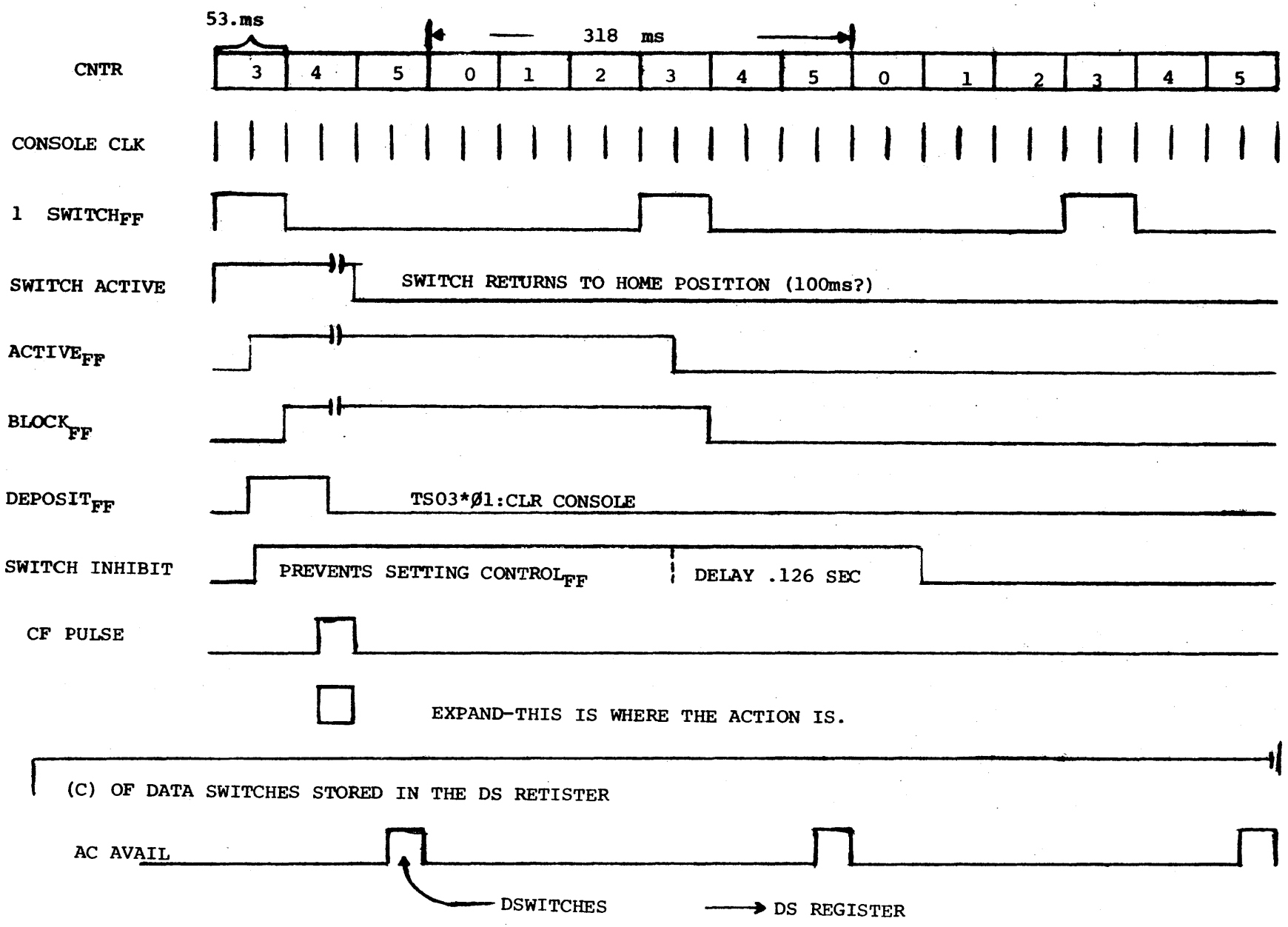
I BUS LIGHTS

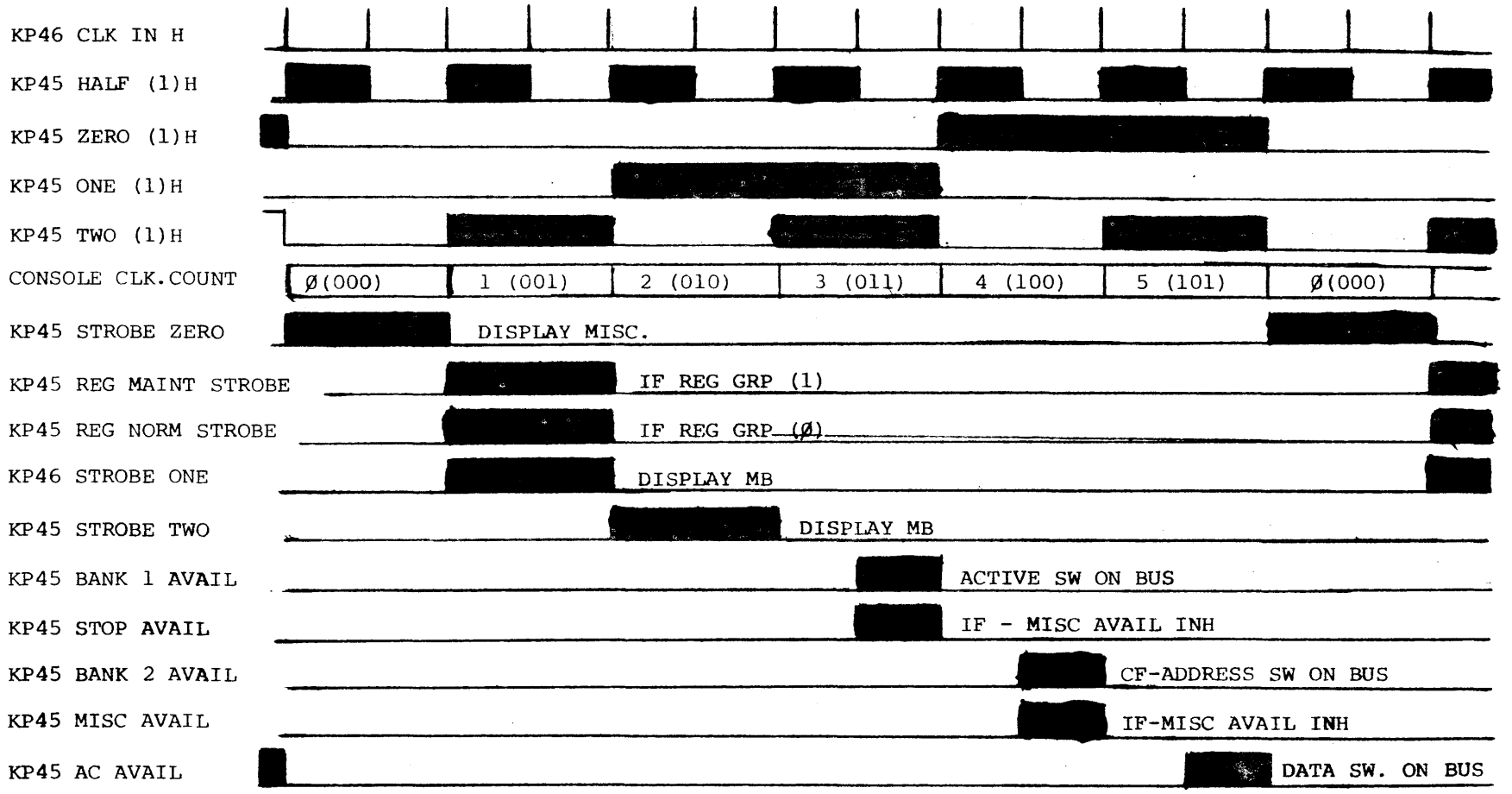
SWITCH REG

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
IND AC	AC Ø	AC 1	AC 2	AC 3	AC 4	AC 5	AC 6	AC 7	AC 8	AC 9	AC 10	AC 11	AC 12	AC 13	AC 14	AC 15	AC 16	AC 17	
PC	PC Ø	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8	PC 9	PC 10	PC 11	PC 12	PC 13	PC 14	PC 15	PC 16	PC 17	
OA	OA Ø	OA 1	OA 2	OA 3	OA 4	OA 5	OA 6	OA 7	OA 8	OA 9	OA 10	OA 11	OA 12	OA 13	OA 14	OA 15	OA 16	OA 17	
MQ	MQ Ø	MQ 1	MQ 2	MQ 3	MQ 4	MQ 5	MQ 6	MQ 7	MQ 8	MQ 9	MQ 10	MQ 11	MQ 12	MQ 13	MQ 14	MQ 15	MQ 16	MQ 17	
PL/SC	API REQ ØØ KAØ5	API REQ Ø1 KAØ5	API REQ Ø2 KAØ5	API REQ Ø3 KAØ5	API REQ Ø4 KAØ1	API REQ Ø5 KAØ1	API REQ Ø6 KAØ1	API REQ Ø7 KAØ1				SC OFLO KEØ3	SC 12 KEØ3	SC 13 KEØ3	SC 14 KEØ3	SC 15 KEØ3	SC 16 KEØ3	SC 17 KEØ3	
XR	XR Ø	XR 1	XR 2	XR 3	XR 4	XR 5	XR 6	XR 7	XR 8	XR 9	XR 10	XR 11	XR 12	XR 13	XR 14	XR 15	XR 16	XR 17	
LR	LR Ø	LR 1	LR 2	LR 3	LR 4	LR 5	LR 6	LR 7	LR 8	LR 9	LR 10	LR 11	LR 12	LR 13	LR 14	LR 15	LR 16	LR 17	
EAE	A (0) H KEØ4	B (0) H KEØ4	C (0) H KEØ4	D (0) H KEØ4	E (0) H KEØ4	F (0) H KEØ4	-SU KEØ4	-MUL KEØ4	-DIV SHIFT KEØ4	-EAE NORMS KEØ4	-EAE LRS KEØ4	-EAE LLS KEØ4	-EAE ACLS KEØ4	-EAE SIGN KEØ5	-P/Q KEØ5	-DIV OFLO KEØ3	-Full KEØ3	-EAE NO SHF KEØ4	LOW TRUE AS SHOWN
DSR	DS Ø	DS 1	DS 2	DS 3	DS 4	DS 5	DS 6	DS 7	DS 8	DS 9	DS 10	DS 11	DS 12	DS 13	DS 14	DS 15	DS 16	DS 17	
I/O B	I_O Ø	I_O 1	I_O 2	I_O 3	I_O 4	I_O 5	I_O 6	I_O 7	I_O 8	I_O 9	I_O 10	I_O 11	I_O 12	I_O 13	I_O 14	I_O 15	I_O 16	I_O 17	
STATUS	PIE KP51	I/O B Ø1 KP53	I/O B Ø2 KP53	KBD FLAG KP60	TELE-PRINT FLAG KP50	I/O B Ø5 KP53	CLK FLAG KP57	CLK EN KP57	I/O B Ø8 KP53	I/O B Ø9 KP52	I/O B 10 KP52	I/O B 11 KP52	I/O B 12 KP52	I/O B 13 KP52	I/O B 14 KP52	I/O B 15 KP52	I/O B 16 KP52	I/O B 17 KP52	
MO	MO Ø	MO 1	MO 2	MO 3	MO 4	MO 5	MO 6	MO 7	MO 8	MO 9	MO 10	MO 11	MO 12	MO 13	MO 14	MO 15	MO 16	MO 17	
A BUS	A Ø	A 1	A 2	A 3	A 4	A 5	A 6	A 7	A 8	A 9	A 10	A 11	A 12	A 13	A 14	A 15	A 16	A 17	NOT
B BUS	B Ø	B 1	B 2	B 3	B 4	B 5	B 6	B 7	B 8	B 9	B 10	B 11	B 12	B 13	B 14	B 15	B 16	B 17	PRESENT
C BUS	C Ø	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10	C 11	C 12	C 13	C 14	C 15	C 16	C 17	when
SHFT BUS	D Ø	D 1	D 2	D 3	D 4	D 5	D 6	D 7	D 8	D 9	D 10	D 11	D 12	D 13	D 14	D 15	D 16	D 17	LIT
I/O A	KP68			I/O_A 3	I/O_A 4	I/O_A 5	I/O_A 6	I/O_A 7	I/O_A 8	I/O_A 9	I/O_A 10	I/O_A 11	I/O_A 12	I/O_A 13	I/O_A 14	I/O_A 15	I/O_A 16	I/O_A 17	
SUM	SM Ø	SM 1	SM 2	SM 3	SM 4	SM 5	SM 6	SM 7	SM 8	SM 9	SM 10	SM 11	SM 12	SM 13	SM 14	SM 15	SM 16	SM 17	
M1	DIV SHIFT → D KP19	MULT SHIFT → D KP19	RAL → D KP19	RAR → D KP19	RTL → D KP19	RTR → D KP19	NO SHIFT → D KP19	SWHA → D KP19	C bus → A KP19	-C bus → A KP19	XR → A KP19	READ IN KP66	SLG → A (TTY)	I/O ADD → A KP19	Addr SW → A KP19	OA → A KP19	DATA IN KDØ5	DATA OUT KDØ5	
M2	SKIP KP23	-MI → B KP19	LR → C KP19	AND → B KP19	Ld AC KP24	Ld MO KP24	Ld PC KP24	Ld OA KP24	Ld LR KP24	Ld XR KP24	BAC → C 09-17 KP19	XR → B Ø-5 KP19	I/O BUS → C KP19	XOR → C KP19	CPMEM REQ (Gate) KP32	START READ KP32	START WRITE KP32	CP MRLS KP32	
MDL	ML Ø	ML 1	ML 2	ML 3	ML 4	ML 5	ML 6	ML 7	ML 8	ML 9	ML 10	ML 11	ML 12	ML 13	ML 14	ML 15	ML 16	ML 17	
MMA	MA Ø	MA 1	MA 2	MA 3	MA 4	MA 5	MA 6	MA 7	MA 8	MA 9	MA 10	MA 11	MA 12	MA 13	MA 14	MA 15	MA 16	MA 17	
MMB	MB Ø	MB 1	MB 2	MB 3	MB 4	MB 5	MB 6	MB 7	MB 8	MB 9	MB 10	MB 11	MB 12	MB 13	MB 14	MB 15	MB 16	MB 17	
MST	ADR ACK MMØ2	BUS EN MMØ2	RD RST MMØ2	WR EN MMØ2	MRLS ACK MMØ2	MEM BUSY MMØ2		MDL Ø3	MDL Ø4			READ CON MMØ1	WRIT CON MMØ1		PAR MB	PAR EXIST			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	


CNT OF 5
DATA SWITCHES
DS REG
CNTS OF 0,1,
AND 2
DISPLAY CYCLES
CNT
OF 3
DEPOSIT SW.
DEPRESSED SET
DEPOSIT
F.F
NO
NOTHING HAPPENS
GO THROUGH
CYCLE 3 & 4
KP34
(TS03* SET FETCH
* DEPOSIT)*CF:
↑ KEY ACTIVE
CNT
of 4
(LAST HALF)
KP34
KEY ACTIVE *
-RUN: ↑ START
RUN.CPU
LEAVES EXECUTE
MAJOR STATE
GOES TO FETCH
MAJOR STATE
KEY ACTIVE * DEPOSIT:
KEY REQ WRT (KP34)
THIS SETS UP WRITE
COMMAND (SMW ON KP32)
KEY ACTIVE : INHIBIT
DECODES (KP48)
KP34
DEPOSTI*TS01
*-NEXT:
ADR-SW-A.
KEY ACT*TS01:
↑ FORCE FETCH
KP24
KEY ACT*TS01*
CLK H: LD MO, OA
NEXT PAGE
A


A
KP31
CNT of 4
LAST HALF
FORCE FETCH:
KEY ACTIVE*FF:
CLR→IR
(NO INSTRUCTION
IS LOADED)
KP26
MEMORY CYCLE
STARTED:
TS01*Ø3: ↑
CP ACTIVE: ↑
MO → MDL- (ADR)
SMW→ MWR
TS02*Ø0:
↑ MREQ
KP34
(DEPOSIT)*TS02:
↑ STOP TS RUN
KP32
(CPACT*START WRT
*TS02*Ø3 *-TRAP
*ADR ACK): ↑ CP
WRT ADR ACK.
THIS IMPLIES LOAD
THE DS REG. INTO
THE MO REG. AND
TRANSFER IT TO MEM
KP34
TS03 * Ø1:
CLR KEY ACTIVE
TS03*Ø1: CLEAR
KEYS * CLEAR
CONSOLE.
CLEAR CONSOLE:
↓ DEPOSTI FF
← TS03*Ø1: LD MI
(C) OF DATA SWITCHES
WILL BE DESPLAYED
IN TEH MB LIGHTS
TS03*Ø3: ↓ RUN
- RUN: ↓ STOP TS
RUN


53.ms
318 ms
CNTR
3 4 5 0 1 2 3 4 5 0 1 2 3 4 5
CONSOLE CLK
1 SWITCH$_{FF}$
SWITCH ACTIVE
SWITCH RETURNS TO HOME POSITION (100ms?)
ACTIVE$_{FF}$
BLOCK$_{FF}$
DEPOSIT$_{FF}$
TS03*Ø1:CLR CONSOLE
SWITCH INHIBIT
PREVENTS SETTING CONTROL$_{FF}$
DELAY .126 SEC
CF PULSE
EXPAND-THIS IS WHERE THE ACTION IS.
(C) OF DATA SWITCHES STORED IN THE DS RETISTER
AC AVAIL
DSWITCHES
DS REGISTER

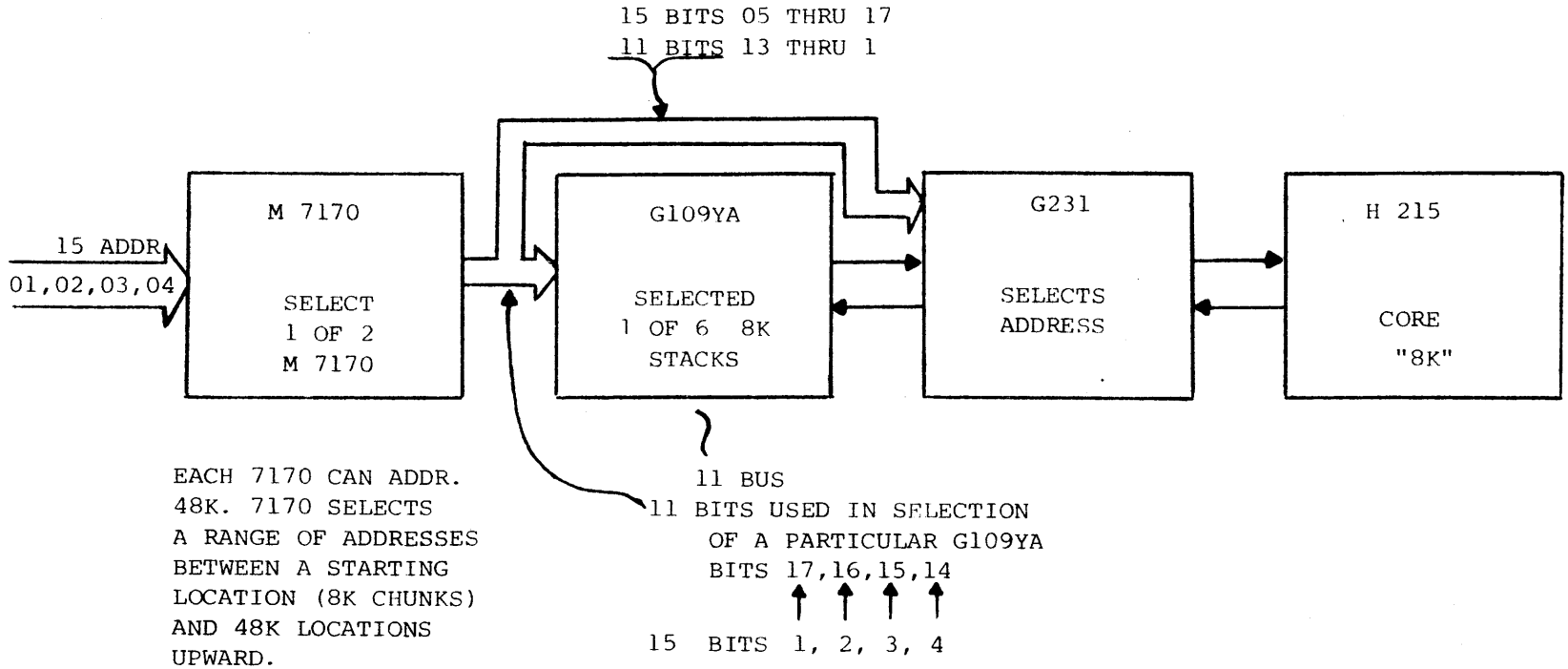


CONTROL CONSOLE TIMING

9. ME-15 MEMORY BLOCK DIAGRAM (11 MEMORY)

OBJECTIVE:

1. List the functions of the M7170 interface control
2. List the functions of the G109-YA
3. List the functions of the G-231
4. Describe the dialogue between the PDP-15 and the ME 15 Memory


15 BITS 05 THRU 17
11 BITS 13 THRU 1
15 ADDR
01,02,03,04
M 7170
SELECT
1 OF 2
M 7170
G109YA
SELECTED
1 OF 6 8K
STACKS
G231
SELECTS
ADDRESS
H 215
CORE
"8K"
EACH 7170 CAN ADDR.
48K. 7170 SELECTS
A RANGE OF ADDRESSES
BETWEEN A STARTING
LOCATION (8K CHUNKS)
AND 48K LOCATIONS
UPWARD.
11 BUS
11 BITS USED IN SELECTION
OF A PARTICULAR G109YA
BITS 17,16,15,14
15 BITS 1, 2, 3, 4

C1	C0	
0	0	DAT1
0	1	DAT1P
1	0	DAT0
1	1	DATOB

C1=0 : READ
C1=1 : WRITE
CO IN CONJUNCTION WITH C1:
WHAT TYPE OF READ + WRITE

TRUE BUS COMMAND

UNIBUS IS POSITIVE WITH LOGIC LEVELS OF:
OV ("1"), +3V("0")
WITH THE EXCEPTION OF BUS GRANT AND NPG, =
3V ("1"), OV (0).

UNIBUS IS LIMITED TO 20 "UNIT LOADS

UNIT LOAD = 1 RECEIVER AND 2 DRIVERS.

7170

15MWR	15MRD	FUNCTION	CO	C1	11CO	11C1
H	L	READ	0(L)	1(H)	1(H)	1(H)
L	H	WRITE	0	0	1(H)	0(L)
L	L	READ/WRITE	1/0	1/0	0(L)/1H	1(H)/0(L)

DIALOGUE P-2-32

EAE INSTRUCTIONS

To be included at a later date.

To be included at a later date.

ASSEMBLY LANGUAGE PROGRAMMING

To facilitate our programming efforts, we will make use of system programs supplied with the Disk Operating System (DOS). They will be the Editor (EDIT), the assembler (MACRO) and the Peripheral Interchange Program (PIP).

The Editor	allows us to set up an area on the disk containing our program (the "source" program). We use the Editor to both create the text and to modify it because of typing errors or changes in program design.
The Assembler	is used to read the "source program" and translate it into a format (the binary program) which can be loaded into memory and subsequently executed.
The PIP program	allows for manipulation and transfer of data files from any input to any output device.

In order to make use of these facilities, we must first load the Disk Operating System's executive program ("The Monitor") into memory so that we can request the Editor and the Assembler through it.

DOSSAV OPERATING INSTRUCTIONS

DOSSAV is the save/restore system for DOS-15.

DOSSAV saves and restores to/from DECdisk, Disk Cartridges, Disk Packs, DECtape and magtape. A DECdisk system can be saved on and restored from DECtape, magtape, Disk Cartridge and Disk Pack. A Disk Pack or Disk Cartridge system can use DECtape and magtape.

Once loaded, DOSSAV asks for all necessary information, such as input and output device, unit numbers and, in the case of magtape, parity and density.

GENERAL INSTRUCTION:

The user must type a Carriage Return after all entries, including the character typed to restart after errors. For UC15 system, start up PIREX as indicated below.

To load PIREX, place the ABS11 paper tape in the PDP-15's paper tape reader. Place the ENABLE/HALT switch on the PDP-11 in the HALT position. Press the STOP and RESET switches on the PDP-15 simultaneously. Set the ADDRESS switches on the PDP-15 to 177ØØ. Press the READIN switch on the PDP-15. When the readin operation is completed and the PDP-15 has halted, set the PDP-11 switch register to:

6ØØØØ for 4K local memory on the PDP-11
1ØØØØØ for 8K local memory on the PDP-11
12ØØØØ for 12K local memory on the PDP-11

and depress the PDP-11 LOAD ADDR switch, then set the ENABLE/HALT switch on the PDP-11 to ENABLE, and finally depress the PDP-11 START switch.

Remove ABS11 from the paper tape reader, and reload it with the PIREX paper tape. Press CONTINUE on the PDP-15. This will cause the ABS11 program (which has two segments: A PDP-11 segment, and a PDP-15 segment) to read in PIREX (which is a PDP-11 absolute binary tape) via the PDP-15 segment and load it into PDP-11 lower memory via the PDP-11 segment.

When the PIREX paper tape has been read in, the PDP-15 will halt, and the PDP-11 will be running PIREX. Remove the PIREX paper tape from the reader. At this point the UNICHANNEL Peripheral Processor has been loaded and is waiting for an I/O request from DOS-15.

A.1 RESTORING SYSTEMS

The following examples illustrate how to put the systems distributed by Digital on DECtape or magtape onto a DECdisk, Disk Pack or Disk Cartridge. The user responses are underlined. The RK05 based systems start up PIREX as described in GENERAL INSTRUCTION, above, before starting up DOSSAV. DOSSAV resides on a paper tape, which must be (HRM) loaded at 37720 (restart 34200).

1. To restore a DECdisk system from DECtape (1 of 2 on Unit 1 and 2 of 2 on Unit 2)

```
DOSSAV Vnn
INPUT DEVICE?  DT↵
UNIT NO?  1↵
OUTPUT DEVICE?  DK↵
DATE CREATED: 06 Jun 73     /Note that if DK is typed no
                            /unit number is requested.
TAPE DONE. MOUNT ANOTHER    /At this point,
                            /type 2 on the key-
2↵                          /board followed by Carriage
                            /RETURN.
```

2. To restore a DECdisk system from magtape (on Unit 0):

```
DOSSAV Vnn
INPUT DEVICE?  MT↵
UNIT NO? 0↵
TRACK (7 OR 9)?  7↵
DENSITY (2,5,8)?  8↵
PARITY (E OR O)?  O↵
OUTPUT DEVICE:  DK↵
DATE CREATED:  06-JUN-73
```

NOTE

All DOS-15 System Restore magtapes distributed by Digital are 800 BPI, odd parity. For 9 track units, DOSSAV assumes 800 BPI.

3. To restore a Disk Pack system from DECtape (1 of 2 on Unit 1 and 2 of 2 on Unit 2):

```
DOSSAV Vnn
INPUT DEVICE?  DT↵
UNIT NO?  1↵
OUTPUT DEVICE?  DP↵
UNIT NO? 0↵
DATE CREATED:  06-JUN-73

TAPE DONE, MOUNT ANOTHER    At this point, type 2 on the
                            teleprinter followed by a
2↵                          Carriage RETURN.
```

4. To restore a Disk Pack system from magtape (on Unit 1):

```
DOSSAV Vnn
INPUT DEVICE?  MT↵
UNIT NO?  1↵
TRACK (7 OR 9)?  7↵
DENSITY (2,5,8)?  8↵
PARITY (E OR O)?  O↵
OUTPUT DEVICE?  DP↵
UNIT NO?  Ø↵
DATE CREATED:  06-JUN-73
```

5. To restore a Disk Cartridge system from DECtapes on Units 1, 2, 3, and 4:

```
DOSSAV Vnn
INPUT DEVICE?  DT↵
UNIT NO?  1↵
OUTPUT DEVICE?  RK↵
UNIT NO?  0↵
DATE CREATED:  Ø6-JUN-73
TAPE DONE.  MOUNT ANOTHER  (The user mounted the next tape on
2↵                          unit number 2, then typed 2↵
                            to continue)
TAPE DONE.  MOUNT ANOTHER  (The user mounted the next tape on
3↵                          unit number 3, then typed 3↵
                            to continue)
TAPE DONE.  MOUNT ANOTHER  (The user mounted the next tape on
4↵                          unit Number 4, then typed 4↵
                            to continue)
DOSSAV  Vnn
INPUT DEVICE?              (Operation complete)
```

6. To restore a Disk Cartridge from magtape Unit 1:

```
DOSSAV  Vnn
INPUT DEVICE?  MT↵
UNIT NO?  1↵
TRACK (7 OR 9)?  7↵
DENSITY (2,5,8)?  8↵
PARITY (E OR O)?  O↵
OUTPUT DEVICE?  RK↵
UNIT #?  Ø↵
DATE CREATED:  Ø6-JUN-73

DOSSAV Vnn
INPUT DEVICE?                    (Operation complete)
```

It is possible to restore to the DECdisk a software system which was created for a machine smaller (different number of DECdisk platters) than the one being restored to. DOSSAV does all the necessary adjustments of the SAT's[1]. Therefore, the restore tapes issued by Digital for a 1-platter system can be restored to any system. Note that this should only be done with the master tape(s) which have block 1775_8

[1]SAT's: Storage Allocation Tables - i.e., bit maps.

free. That block is needed during the restore for five or more DECdisk platters. It is not possible to restore a software system which is larger than the hardware. (For example, one cannot restore a 3-platter system onto a 1-platter configuration.)

The system can then be bootstrapped from the appropriate disk. See the DOS Keyboard Command Guide (DEC-15-ODKCA-A-D).

A.2 SAVING SYSTEMS

Once the user has tailored the system to his specific configuration, he will want to save that system for future restorations. To do that, simply reverse the procedure above. To illustrate, consider Example 1 above and the changes necessary to it to create a restore tape.

To save a DECdisk system to DECtape (on Units 1 and 2);

```
DOSSAV Vnn
INPUT DEVICE?  DK)
OUTPUT DEVICE?  DT)
UNIT No?  1)
TAPE DONE.  MOUNT ANOTHER          At this point, type 2 on the
                                   keyboard followed by a Carriage
2)                                 RETURN.
```

Note that DOSSAV allows for as many DECtapes and magtapes as are necessary to hold the system.

A.3 ERROR CONDITIONS AND MESSAGES

Recoverable errors during command string decoding: If a question is answered incorrectly, DOSSAV outputs an appropriate error message and then repeats the question. These error messages are:

ILLEGAL DEVICE	An illegal device mnemonic was typed (something other than DP, DK, RK, DT, or MT) or an illegal combination of devices was typed (i.e., input = DT and output = MT).
BAD TRACK	Something other than 7 or 9 was typed.
BAD DENSITY	Something other than 2 (200), 5 (556), or 8 (800) was typed.
BAD PARITY	Something other than E (even) or O (odd) was typed.

Recoverable errors during operations: If it is possible to recover from an error, DOSSAV attempts to do it. The error message is output to the console. After the problem has been corrected, any character on the keyboard followed by a Carriage RETURN resumes operation.

TAPE NOT READY — The DECtape or magtape unit is off line or not write enabled.

DISK NOT READY — DECdisk is write locked.

DISK PACK NOT READY — The Disk Pack or Disk Cartridge unit is not ready.

Unrecoverable errors: Primarily hardware errors, from which DOSSAV cannot recover. After the error message has been output, DOSSAV restarts. DOSSAV retries five times on parity error, before issuing an unrecoverable error message.

DECTAPE ERROR

MAGTAPE ERROR

DISK ERROR

DISK PACK ERROR

ATTEMPT TO RESTORE SYSTEM TO WRONG DISK

To protect users who have access to more than one type of disk and who may have several sets of restore tapes, all restore tapes are created with the mnemonic of the disk type in the first SAT. DOSSAV checks this code against the output device code. If they differ, this message is output.

BLK 1775 OCCUPIED. NO 2ND SAT CREATED

A DECdisk system created for 4 or fewer platters is restored to a machine with 5 or more platters and block 1775 is already used. Therefore, no second SAT is created. A master tape was not used to make the restore.

XX ERR IGN

where xx = DK or DP or RK.

This error is typed on the console, and the PDP-15 halts. This reports that "Read/Write check" errors occurred more than 12_8 time during a save or restore process. The bad block number is present in the PDP-15 AC. Users can continue the save or restore process by pressing the continue switch on the console of the machine.

A.4 TAPE STRUCTURE

The restore tapes are structured as follows: The first SAT of the system is the first block put on the tape. This SAT, which is never restored to the disk, has two words modified: word 2 contains the creation date (taken from .SCOM+47) and word 376 contains the device mnemonic (.SIXBT , right justified). All the occupied blocks referenced by this SAT are then put sequentially on the tape. The second SAT, if there is one, is then put on, and so on. This structure enables use of magtape, which is a sequential only device.

A.5 DOSSAV Restrictions

1. It is not possible to save or restore magtapes with even parity.

2. DOSSAV fails when two DECtapes are on line with the same unit number. It is necessary to restart under such circumstances

MACRO-15 SUMMARY

I. INTRODUCTION

MACRO-15 is a system program which facilitates writing programs symbolically because it converts symbolic representation to machine code.

INPUT: ASCII text

OUTPUT: Binary

.FULL	.ABS
Output to paper tape for HARDWARE READIN	output to paper tape for use with Absolute Loader

II. FIELD DEPENDENT ASSEMBLER

A MACRO source program is composed of a sequence of source lines where each line (terminated by a carriage return) may contain one or more assembly language statements. If several statements are written on a single line, they are to be separated by semicolons.

STATEMENT

or

STATEMENT;STATEMENT;STATEMENT

Statements may contain up to four fields that are separated by space(s) or tab (but not both except between address and comment fields). The four fields are:

LABEL (OR TAG) field

OPERATION (OR OPERATOR) field

OPERAND (OR ADDRESS) field

COMMENTS field

and are identified by order of appearance and delimited by certain terminating characters. The general format of a MACRO assembly language statement is:

LABEL or OPERATION FIELD or OPERAND FIELD or /COMMENTS
(ADDRESS)

where each field is delimited by a space(s) OR a tab and each statement is terminated by a semicolon or carriage return. The comments field is preceded by a tab or space(s) and forward slash (/). The label and comments field are optional. The operation and operand fields are interdependent; either may be omitted depending upon the contents of the other. However, blank lines are illegal. (NOTE that if several statements are on a line, only the last may have comments).

FIELD CONTENTS

LABEL or TAG FIELD	OPERATION FIELD	ADDRESS FIELD	COMMENT FIELD
Symbolic Label	Machine Mnemonic Instructions	Symbol	/.....
Direct Assignment	MACRO-15 Assembler Directive	Number	
	Macro Name	Expression	
	Number Symbol Expression		

LABEL FIELD

1- a label is a symbolic address created by the programmer to identify the statement (usually for reference as addresses for jump instructions, data locations, and for debugging).

2- labels are optional but if present, a label always occurs first in a statement (column 1)

3- a label takes as its numeric value, the address of the location it names (the value of the current location counter is entered into the user defined symbol table).

4- a label may be defined only once (therefore, a given label may be used on only one statement).

5- note that even if a label is not used in a statement, the delimiter for the label field still must be given.

OPERATION FIELD

1- the operation field follows the label field

2- the operation field may contain a(n):

--instruction mnemonic
--assembler directive
--number
--symbol
--expression
--Macro Call

3- the operator may be preceded by none or one label and may be followed by none,one or more operands and/or a comment.

4- when the operator is a macro call, the assembler inserts the appropriate code to expand the macro. When the operator is an instruction mnemonic, it specifies the instruction to be generated and the action to be performed on any operand(s) that follow. When the operator is an assembler directive, it specifies a certain function or action to be performed during assembly.

OPERAND FIELD

1- an operand is that part of a statement that is manipulated by the operator.

2- an operand is usually a symbolic or numeric address of data to be accessed when an instruction is executed or the arguments of an assembler directive or Macro Call.

3- the interpretation of the operands depends upon the operation field.

Macro source statements may use the tab to align the statement fields according to the following format:

label-- column 1

operation--column 9

operand-- column 17

comment-- column 33

(tabs are set up to move modulo 8 spaces; columns 1,9,17.25,33,41,49,57 etc.)

EXAMPLE

```
        .LOC 10500
START   CLX
        LAC     10600           /PICK UP NEG TAX RATE
        DAC*    10601           /PUT IT IN TAX TABLE
        LAC     DATAB           /PICK UP GROSS PAY
        DAC     TAB,X           /STORE IT
        HLT
DATAB   20000                   /NOT SO GOOD--OCTAL
        .LOC 10600
        62
        TAXTAB
TAXTAB  .BLOCK 6
        .END START
```

COMMENT FIELD

1 - a comment is a short explanatory note which the programmer adds to a statement as an aid in later analysis, debugging or documentation.

2 - comments are optional

3 - comments do not affect the assembly process or the object program (and hence do not affect program execution). They are merely printed in the lising.

4 - the comment field must be preceded by a forward slash (/). The slash may be the first character on a line (if the entire line is to be taken as a comment) or may be preceded by:

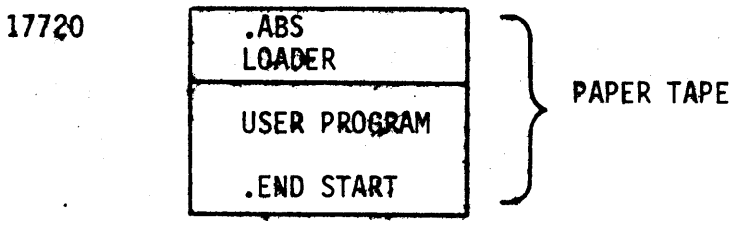
a. space(s)
b. tab(s)
c. semicolon

```
 ↑C
$LOGIN CES


DOS-15 V3A000
$A PP -13

$A LP -12

$K ON

$EDIT


 EDITOR V3A000
>OPEN ZTABLE
FILE ZTABLE SRC NOT FOUND.
INPUT
          .ABS
          .LOC 100
          LAC K400                  /LOAD AC WITH STARTING LOC
          DAC A                     /DEPOSIT AC INTO STARTING LOC
          LAW -100                  /LOAD AC WITH NO. OF LOC TO CLR
          DAC CNT#                  /DEPOSIT AC INTO NO. CNTR
LOOP      DZM* A                    /ZERO CURRENT ADDRESS
          ISZ A                     /INC ADDRESS
          ISZ CNT                   /INC NO. COUNT, SKIP IF DONE
          JMP LOOP                  /NOT DONE, GET NEXT ADDRESS
          HLR\T                     /DONE
A         400
K400      400
          .END

EDIT
>EXIT


DOS-15 V3A000
$MACRO

BMACRO-15 V3A000
>B N←ZTABLE
 END OF PASS 1
SIZE=00114      NO ERROR LINES
BMACRO-15 V3A000
>↑C

DOS-15 V3A000
$LOGOUT


DOS-15 V3A000
$
```

```
                                    .ABS
                                    .LOC 100
                    200112          LAC K400        /LOAD AC WITH STARTING LOC
                    040111          DAC A           /DEPOSIT AC INTO STARTING LOC
                    777700          LAC -100        /LOAD AC WITH NO. OF LOC TO CLR
                    040113          DAC CNT#        /DEPOSIT AC INTO NO. CNTR
          000124    160111    LOOP  DZM* A          /ZERO CURRENT ADDRESS
          000125    440111          ISZ A           /INC ADDRESS
          000126    440113          ISZ CNT         /INC NO. COUNT, SKIP IF DONE
          000127    600124          JMP LOOP        /NOT DONE, GET NEXT ADDRESS
          000130    740040          HLT             /DONE
          000111    000400    A     400
          000112    000400    K400  400
                    000000          .END
                SIZE=000114       NO ERROR LINES
```

OBJECT PROGRAM OUTPUT

The absolute (.ABS) pseudo-ops cause absolute, checksuirned binary code to be output. If no value is specified in the address field and if the output device is the paper tape punch, the assembler will precede the output with the Absolute Binary Loader (ABL), which will load the punched output at object time. The ABL is loaded via hardware readin into location 17720 of any memory bank.



To Load your program:

Set address switches to 17720
STOP, RESET
READIN

To start your Program:

Set Address Switches to your
PROGRAM STARTING ADDRESS
STOP, RESET
START

PROGRAMMING ASSIGNMENT

Write at least one of the following programs:

A. Write a program that will read two values from the console data switches.

If equal Halt with the AC = Ø
If Unequal Halt with the AC = -1

NOTE: Use the two's complement version of -1, I.E. 777777.

B. Write a program which reads a value from the console data switches and counts the number of bits that are set to 1 in the value.

Have the program halt with the final answer left in the AC so that it may be checked via the console switches.

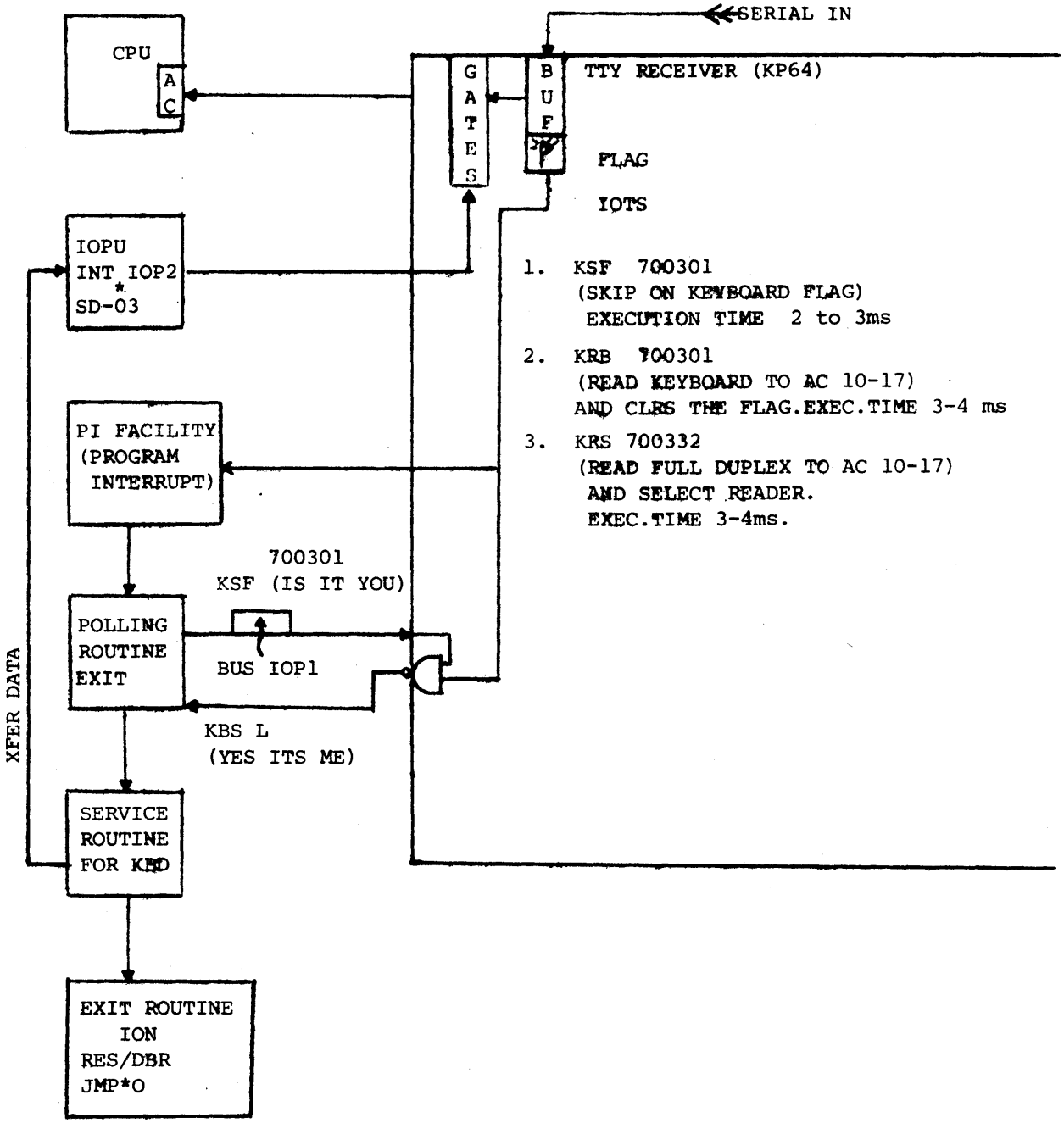
Example: DATA SWITCHES = 102376 OCTAL

NUMBER OF BITS SET = 11 OCTAL

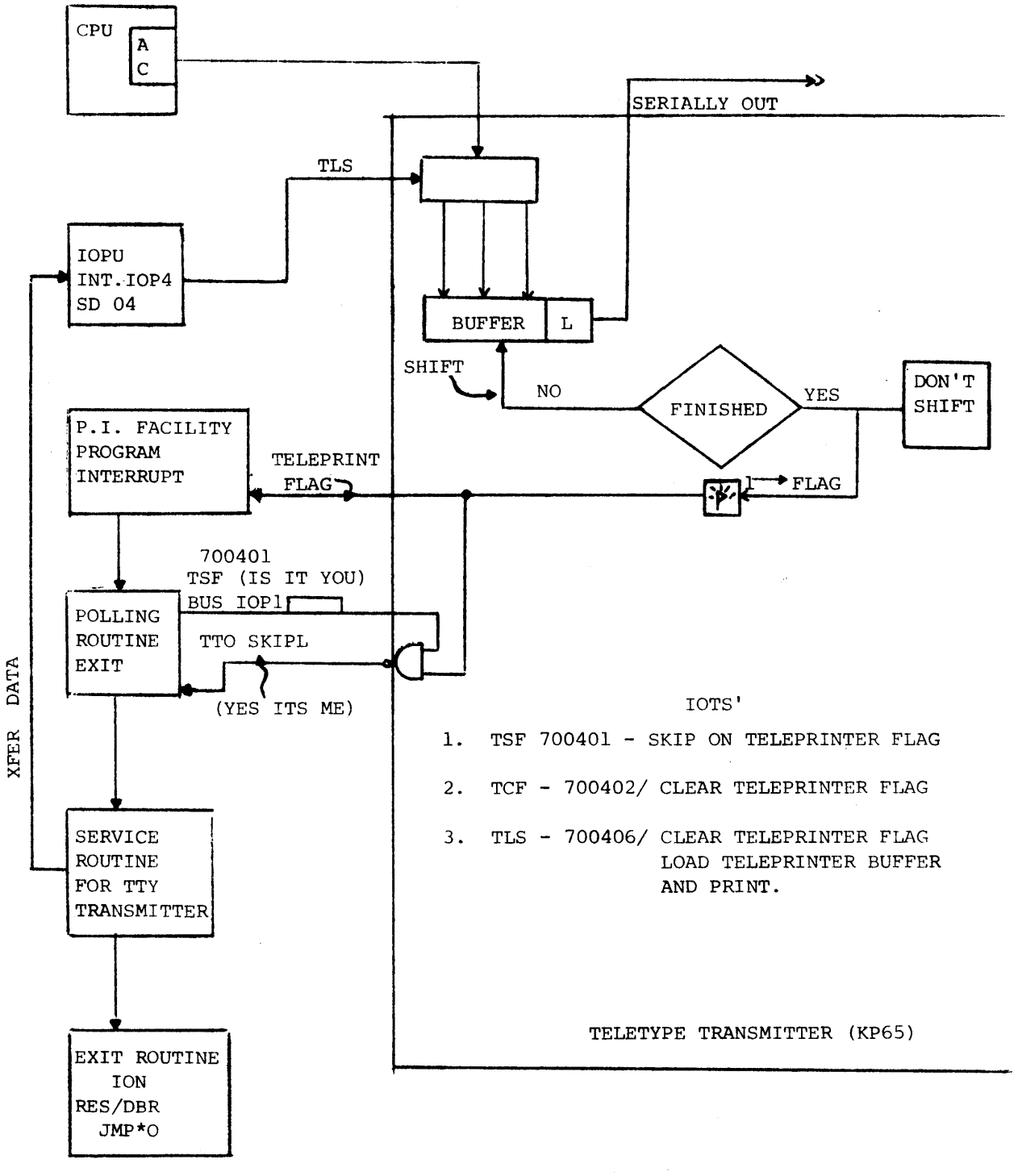
C. Write a program which reads a value from the console switches and reverses the octal number. Have the porgram halt with the reverse value in the AC so that it may be checked via the console data switches.

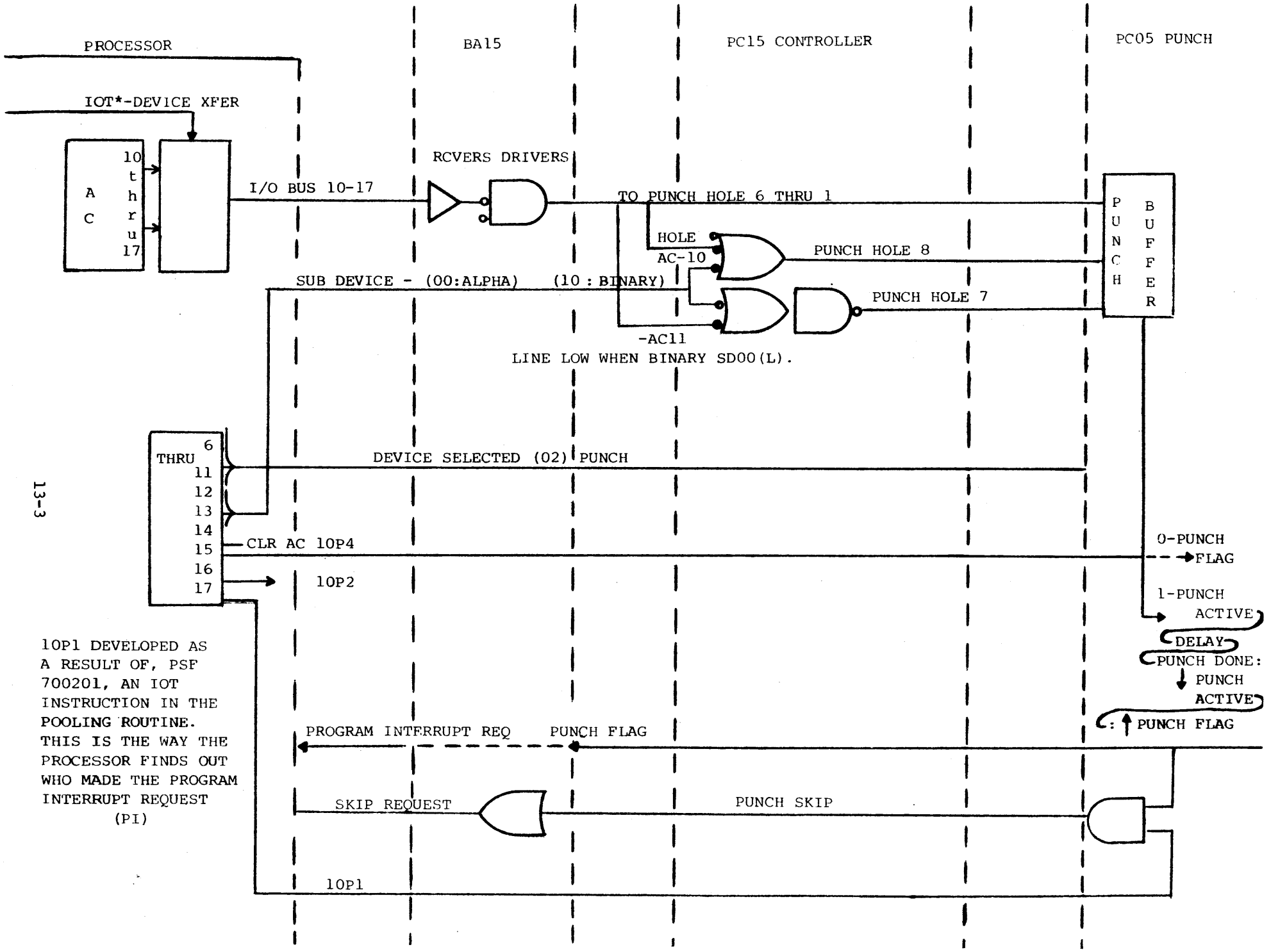
Example: DATA SWITCHES = 762415 reversed value = 514267

DEVICE: TTY RECEIVER



DEVICE: TTY TRANSMITTER




PROCESSOR
BA15
PC15 CONTROLLER
PC05 PUNCH
IOT*-DEVICE XFER
AC
10 thru 17
I/O BUS 10-17
RCVERS DRIVERS
TO PUNCH HOLE 6 THRU 1
PUNCH BUFFER
HOLE AC-10
PUNCH HOLE 8
SUB DEVICE - (00:ALPHA) (10 : BINARY)
PUNCH HOLE 7
-AC11
LINE LOW WHEN BINARY SD00(L).
6 THRU 11 12 13 14 15 16 17
DEVICE SELECTED (02) PUNCH
CLR AC 10P4
0-PUNCH FLAG
10P2
1-PUNCH ACTIVE
DELAY
PUNCH DONE: PUNCH ACTIVE
: PUNCH FLAG
10P1 DEVELOPED AS A RESULT OF, PSF 700201, AN IOT INSTRUCTION IN THE POOLING ROUTINE. THIS IS THE WAY THE PROCESSOR FINDS OUT WHO MADE THE PROGRAM INTERRUPT REQUEST (PI)
PROGRAM INTERRUPT REQ
PUNCH FLAG
SKIP REQUEST
PUNCH SKIP
10P1

IOT

OBJECTIVES:

1. Describe the format of IOT instructions.

2. Describe in general terms the function of device select codes, sub-device select codes, and IOP signals.

3. Describe the logical operation and timing of a basic IOT instruction.

IOT FACILITY

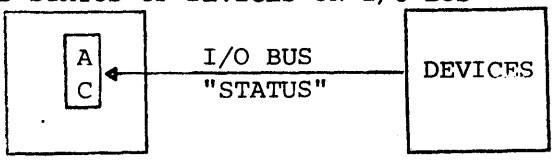
IOTS USE:

1. CHECK FOR SKIP CONDITIONS

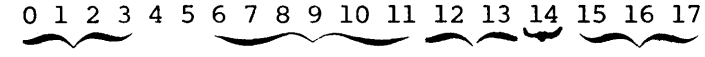
2. PROGRAM CONTROLLED DATA TRANSFERS, TELETYPE, PAPER TAPE READER & PUNCH, ETC.

3. SETTING UP CONTROL OF BLOCK TRANSFER DEVICES. DATA TRANSFERS FOR BLOCK TRANSFER DEVICES ARE DONE BY THE DATA CHANNEL FACILITY

4. READ STATUS OF DEVICES ON I/O BUS



IOT INSTRUCTION



OP CODE

DEVICE SELECT

TO ADDRESSABLE I/O BUS

SUB DEV. SEL.

C L R A C

IOT CONTROL TIMING
BIT 17=1 GENERATE IOP1
BIT 16=1 GENERATE IOP2
BIT 15=1 GENERATE IOP4
BITS 15&17=1 GENERATE IOP1 FOLLOWED BY IOP4
BITS 15&16=1 GENERATE IOP2 FOLLOWED BY IOP4

(C) of MI REG CODE FOR IOT

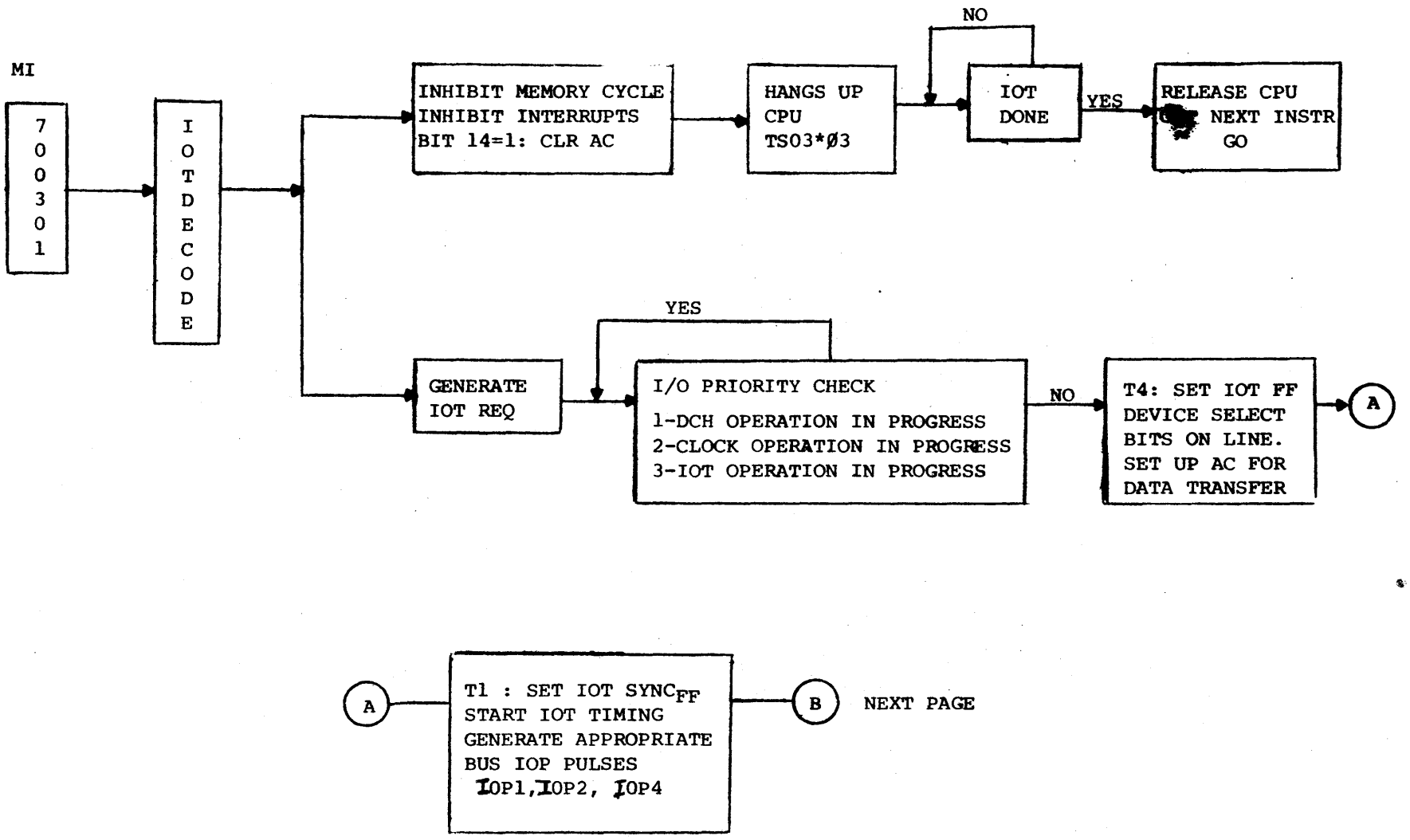
70 0301

THE ACCUMULATOR REGISTER IS USED IN CONJUNCTION WITH THE IOT.

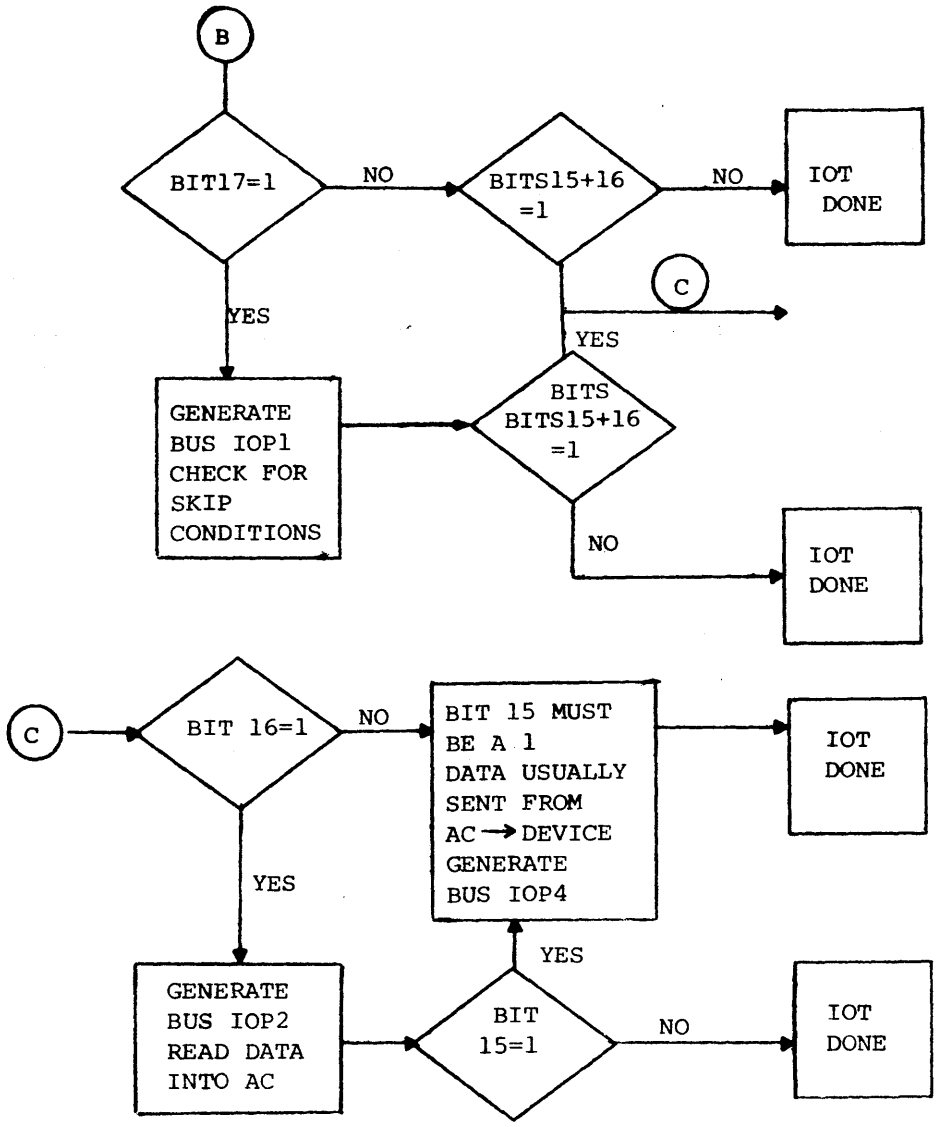
ACCUMULATOR IS USED TO:

a) TRANSFER COMMANDS TO A PARTICULAR DEVICE
b) DATA TRANSFERRED TO A DEVICE. AC → DEVICE
   DATA TRANSFERRED FROM A DEVICE. DEVICE → AC

IOT BLOC[illegible]IAGRAM



IOT DONE BLOCK DIAGRAM



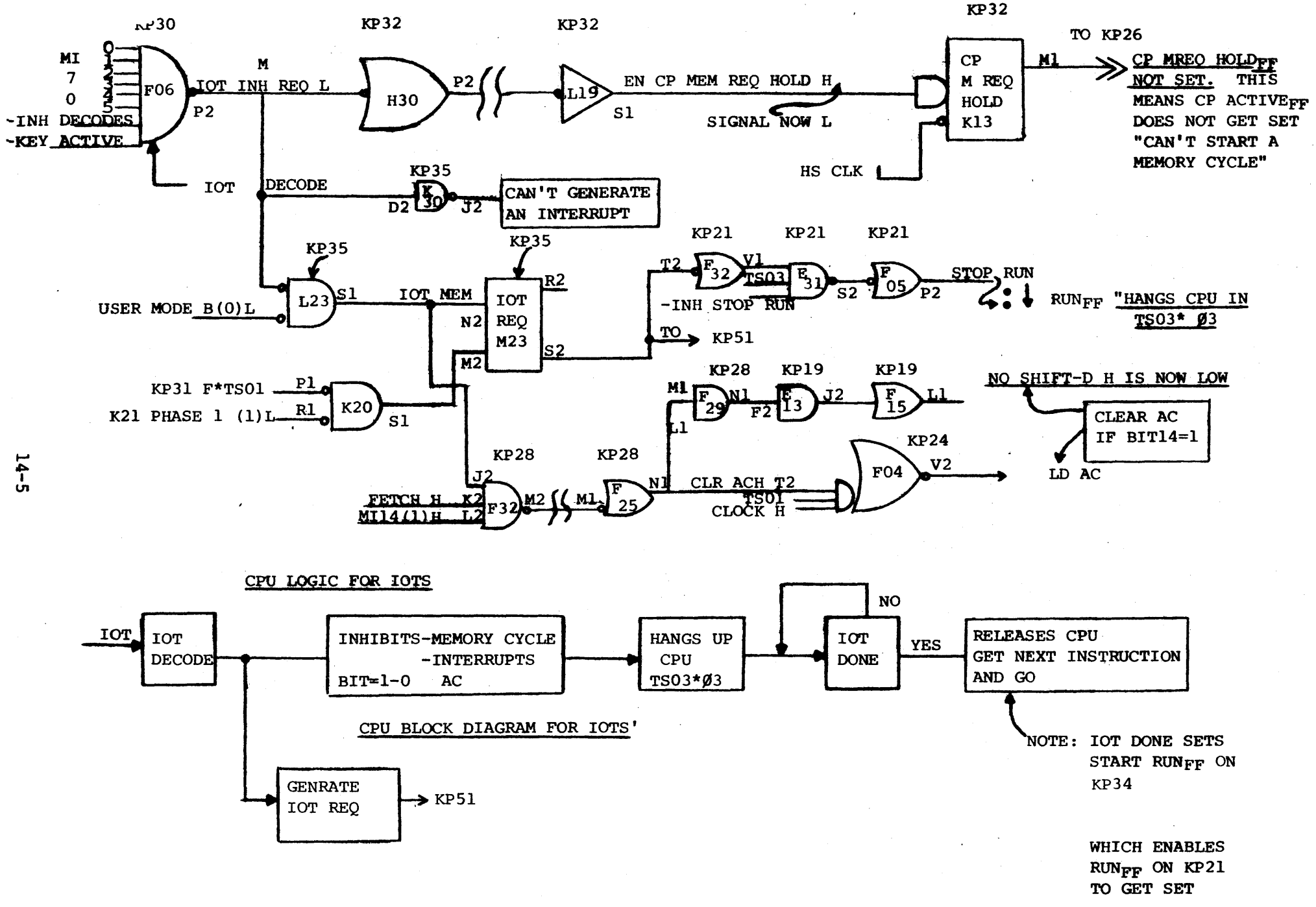
THIS CONDITION IS A NOP

IOT DONE TERMINATES THE IOT OPERATION

IOT DONE TERMINATES BY THE FOLLOWING MANNER:

KP51 RESETS IOT_{FF}
RESETS IOT $SYNC_{FF}$
KP35 RESETS IOT REQ_{FF}
KP34 RESETS START RUN
KP21 ENABLES CPU TO START RUN

KP76 SHOWS CPU GOES TO EXECUTE MAJOR STATE TO READ IN THE INSTR. RD RST IS USED TO LD M1. AT THIS TIME IOT DECODE DROPS.

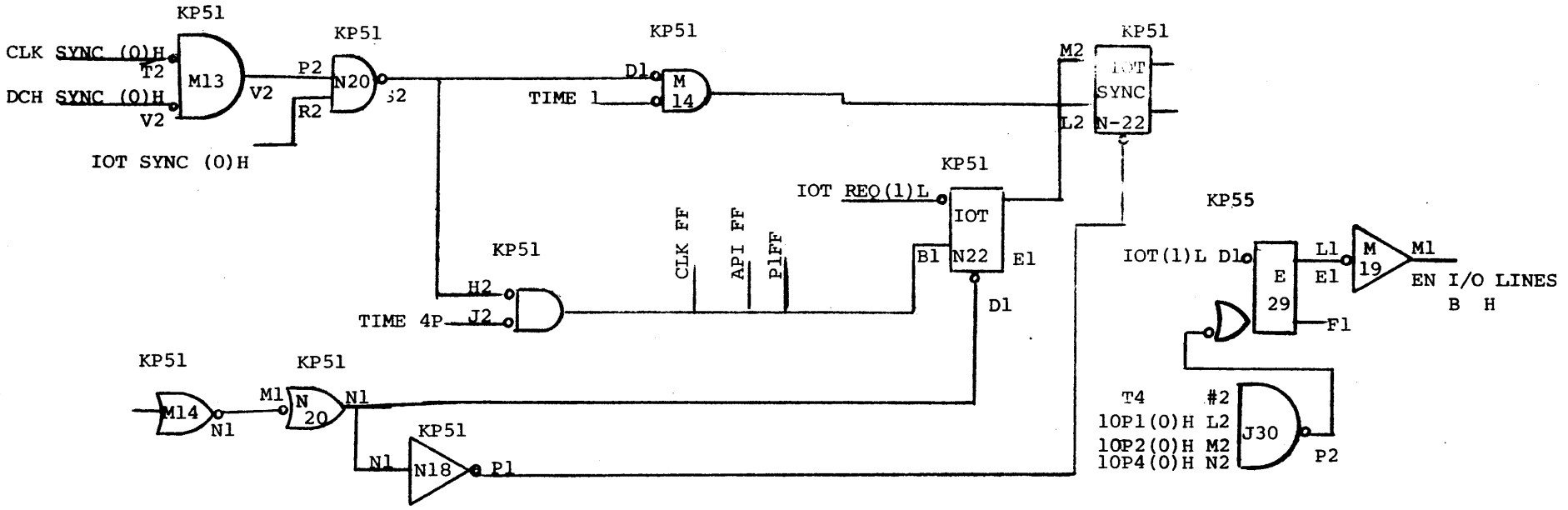


CPU LOGIC FOR IOTS

CPU BLOCK DIAGRAM FOR IOTS'

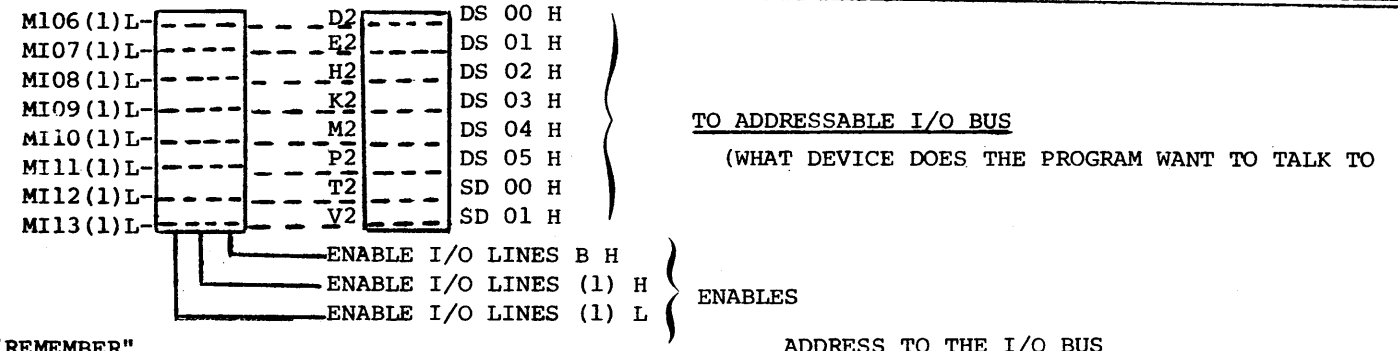
NOTE: IOT DONE SETS START RUN FF ON KP34

WHICH ENABLES RUN FF ON KP21 TO GET SET



KP51 KP-51 M05

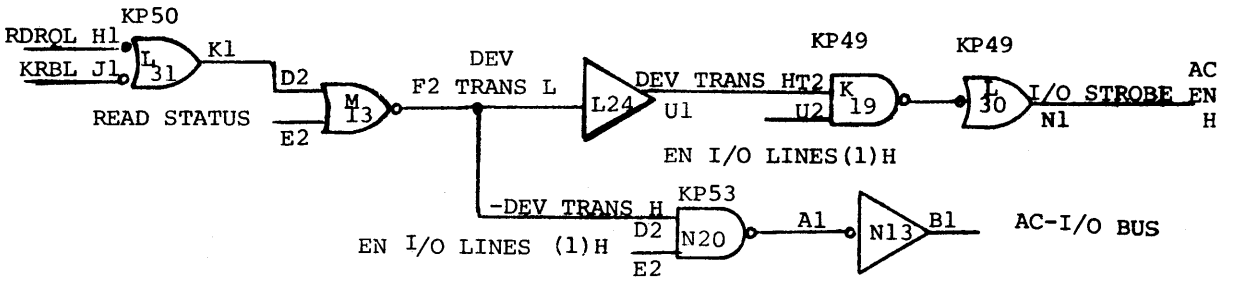
PRIORITY CHECK OPERATION



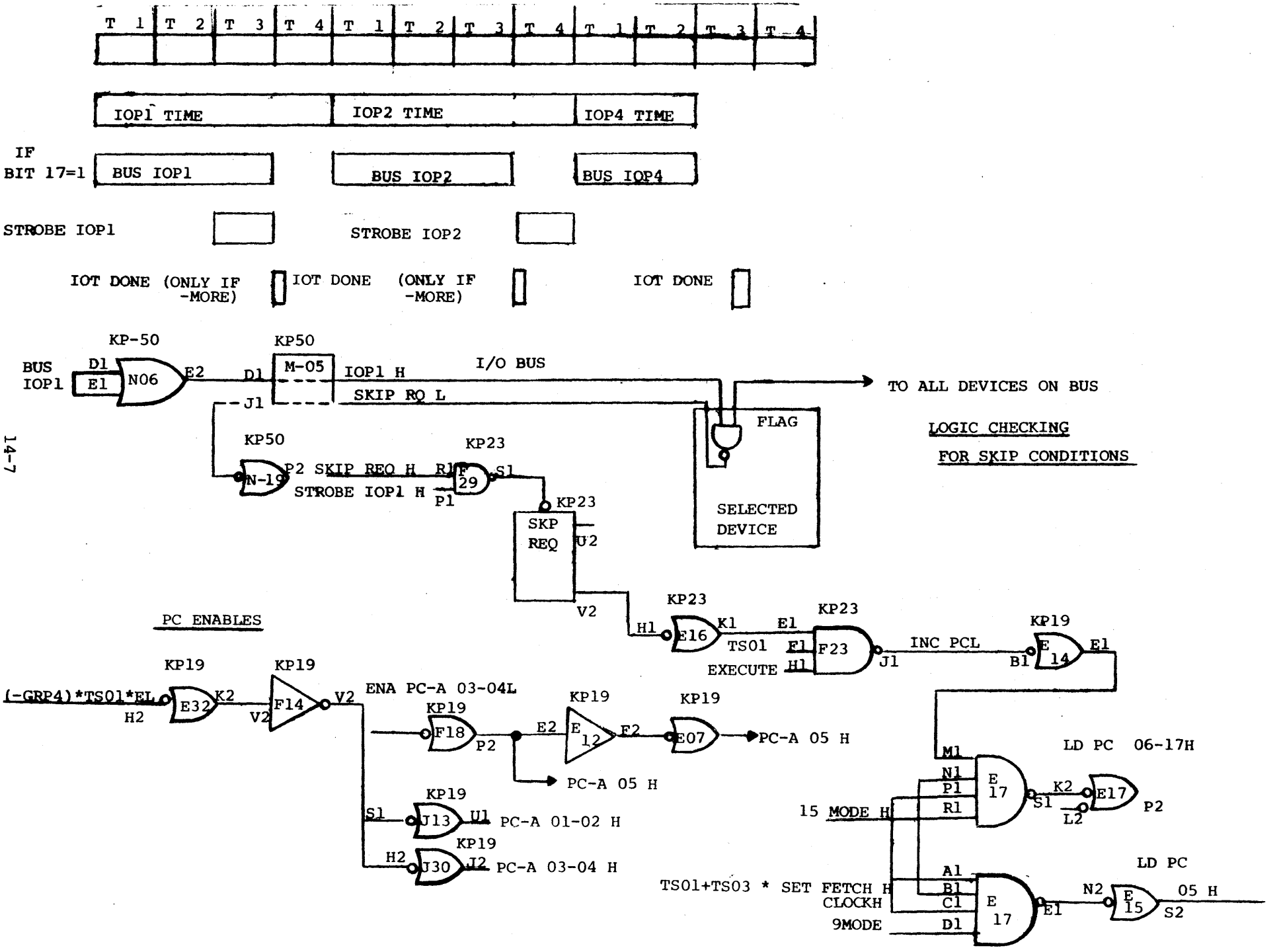
"REMEMBER"

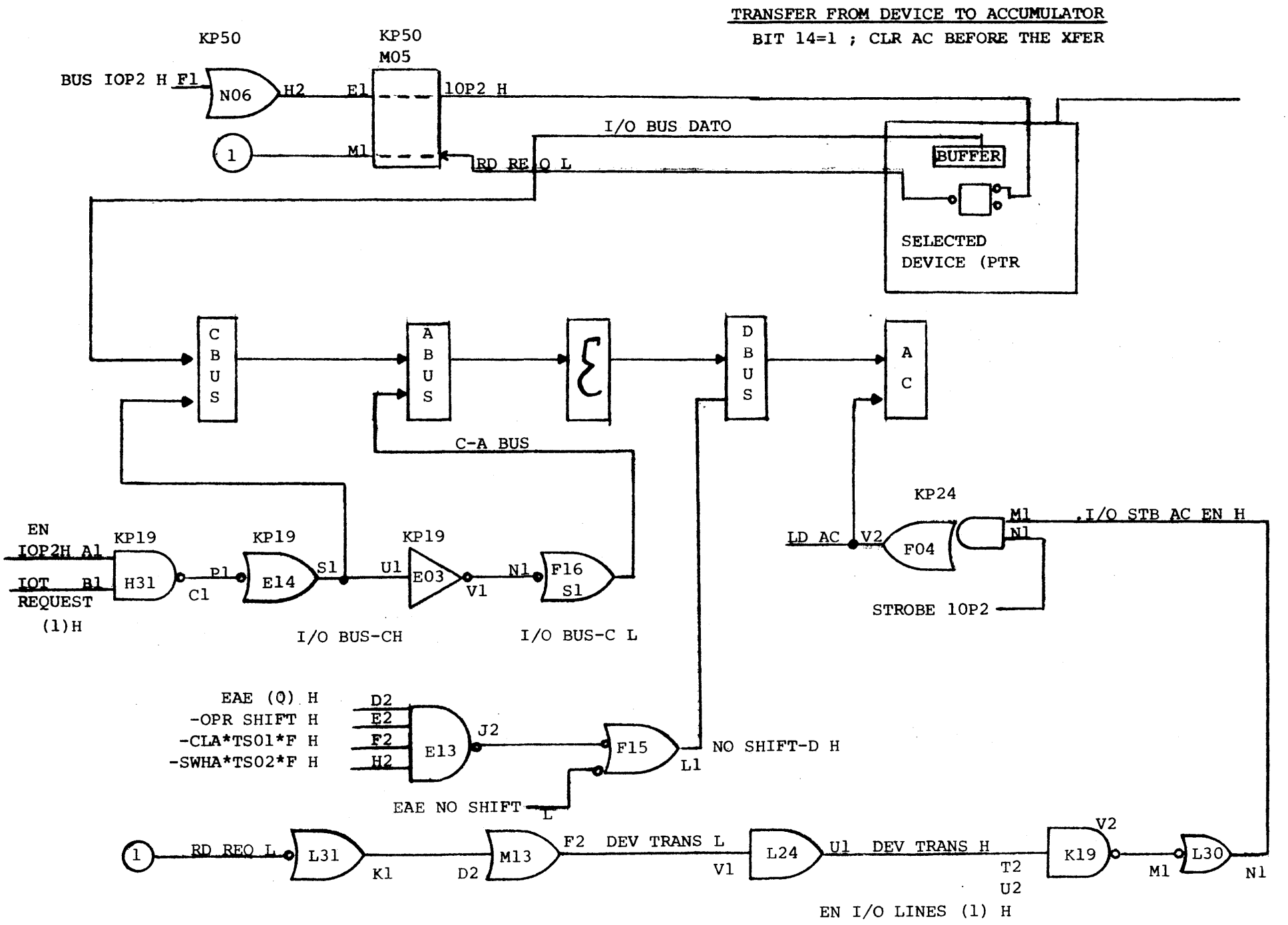
IOT INSTRUCTION
IS IN THE MI REGISTER

ADDRESS TO THE I/O BUS

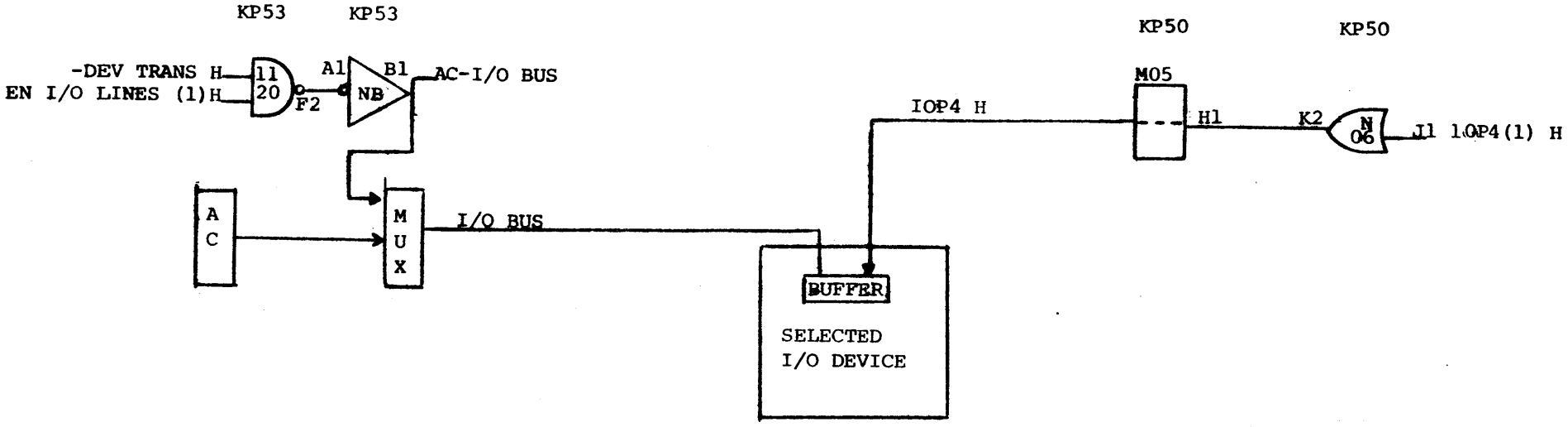


I/O STROBE AC EN USED FOR DATA TRANSFERS
COMING INTO THE ACCUMULATOR


T 1 T 2 T 3 T 4 T 1 T 2 T 3 T 4 T 1 T 2 T 3 T 4
IOP1 TIME
IOP2 TIME
IOP4 TIME
IF
BIT 17=1
BUS IOP1
BUS IOP2
BUS IOP4
STROBE IOP1
STROBE IOP2
IOT DONE (ONLY IF -MORE)
IOT DONE
(ONLY IF -MORE)
IOT DONE
KP-50
KP50
BUS IOP1
D1
E1
N06
E2
D1
M-05
IOP1 H
I/O BUS
TO ALL DEVICES ON BUS
J1
SKIP RQ L
FLAG
LOGIC CHECKING
FOR SKIP CONDITIONS
KP50
KP23
N-19
P2 SKIP REQ H
R1
F 29
S1
STROBE IOP1 H
P1
KP23
SKP REQ
U2
SELECTED DEVICE
V2
KP23
KP23
H1
E16
K1
E1
TS01
F1
F23
J1
INC PCL
KP19
E 14
E1
EXECUTE
H1
B1
PC ENABLES
KP19
KP19
(-GRP4)*TS01*EL
H2
E32
K2
V2
F14
V2
ENA PC-A 03-04L
KP19
KP19
KP19
F18
P2
E2
E 12
F2
E07
PC-A 05 H
PC-A 05 H
LD PC 06-17H
M1
N1
P1
R1
E 17
K2
S1
E17
P2
L2
15 MODE H
KP19
S1
J13
U1
PC-A 01-02 H
KP19
H2
J30
J2
PC-A 03-04 H
LD PC
A1
B1
C1
D1
E 17
E1
N2
E 15
05 H
S2
TS01+TS03 * SET FETCH H
CLOCKH
9MODE


TRANSFER FROM DEVICE TO ACCUMULATOR
BIT 14=1 ; CLR AC BEFORE THE XFER
KP50
KP50
M05
BUS IOP2 H
F1
N06
H2
E1
IOP2 H
I/O BUS DATO
1
M1
RD REQ L
BUFFER
SELECTED
DEVICE (PTR
C
B
U
S
A
B
U
S
D
B
U
S
A
C
C-A BUS
KP24
EN
IOP2H A1
KP19
IOT
REQUEST
(1)H
B1
H31
P1
C1
KP19
E14
S1
KP19
U1
E03
V1
N1
F16
S1
I/O BUS-CH
I/O BUS-C L
LD AC
V2
F04
M1
N1
.I/O STB AC EN H
STROBE IOP2
EAE (Q) H
-OPR SHIFT H
-CLA*TS01*F H
-SWHA*TS02*F H
D2
E2
F2
H2
E13
J2
F15
L1
NO SHIFT-D H
EAE NO SHIFT L
1
RD REQ L
L31
K1
M13
D2
F2 DEV TRANS L
V1
L24
U1 DEV TRANS H
T2
U2
V2
K19
M1
L30
N1
EN I/O LINES (1) H

PROGRAM CONTROLLED TRANSFER TO DEVICE

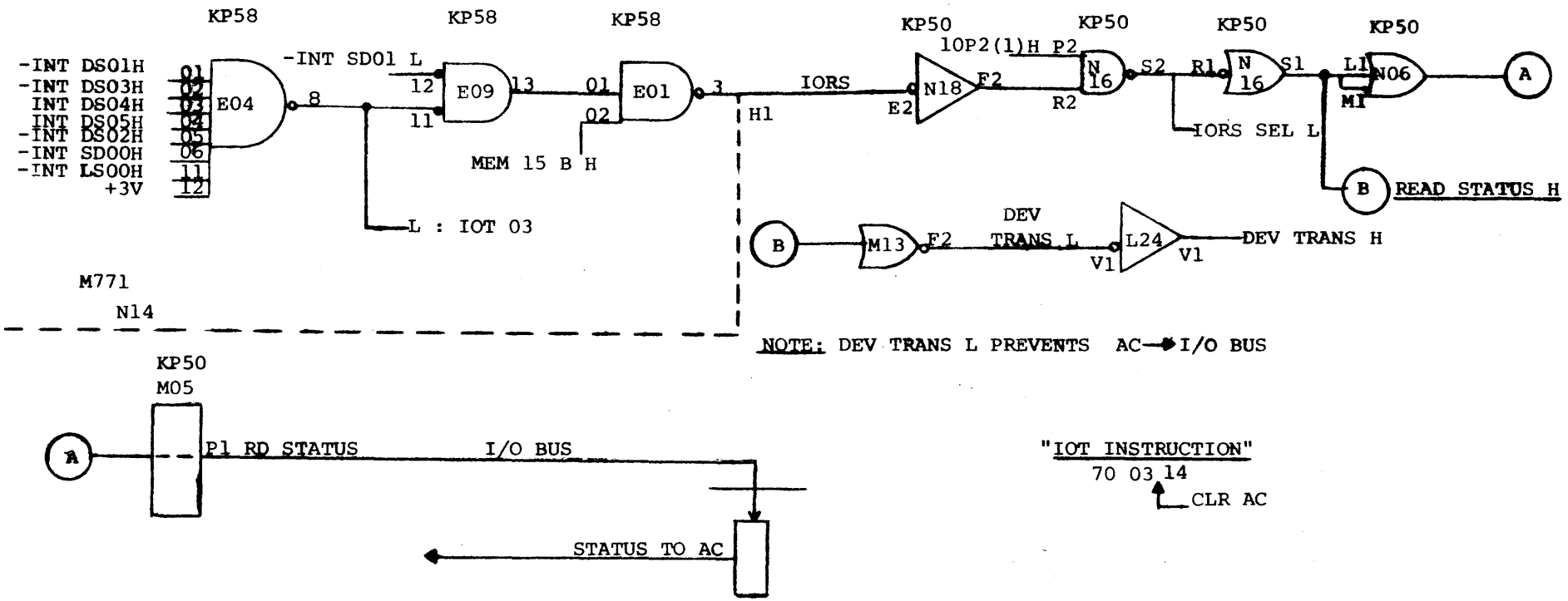


PROGRAM CONTROLLED TRANSFER TO DEVICE
AC ⟶ DEVICE
LOAD BUFFER WITH IOP4

IOT INSTRUCTION:

70 02 04	PUNCH ALPH
70 02 44	PUNCH BINARY

READ STATUS



NOTE: BITS THAT ARE USED TO SELECT A DEVICE "03" SPECIFIES THE CONSOLE TELETYPE KEYBOARD. THIS DEVICE DOES NOT USE THE IOP4 SIGNAL. IT IS ALLOWED TO GO OUT ON THE BUS BUT NOTHING HAPPENS. THE STATUS IS READ INTO THE ACCUMULATOR IN 10P2 TIME.

PROGRAM INTERRUPT

OBJECTIVES:

a) DESCRIBE IN GENERAL TERMS THE PROCESSING OF A PROGRAM INTERRUPT BY THE CENTRAL PROCESSOR.

b) STATE THE PROGRAMMING CONSIDERATIONS NECESSARY WHEN USING THE PROGRAM INTERRUPT FACILITY.

c) DESCRIBE THE LOGICAL OPERATION OF THE ION, IOF, AND DBR, RES IOT INSTRUCTIONS.

d) EXPLAIN THE LOGICAL OPERATION AND TIMING OF A PROGRAM INTERRUPT USING THE LOGIC PRINTS.

PROGRAM INTERRUPT FACILITY (PI)

QUESTION:

WHAT IS THE PURPOSE OF THE PROGRAM INTERRUPT?

ANSWER- DO AWAY WITH THE WAIT LOOP IN A SYSTEM PROGRAM.

```
WAIT LOOP
      KSF              DEDICATED I/O
      JMP.-1
```

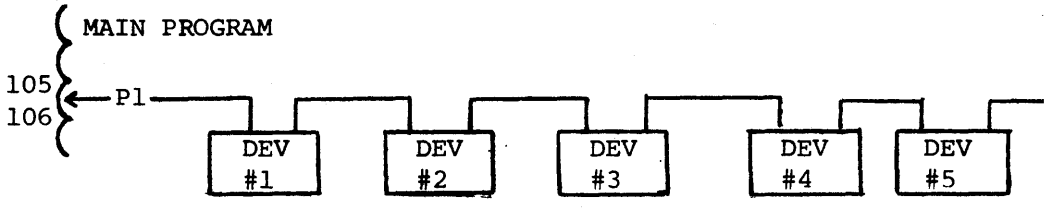
TAKE NOTICE NOTHING ELSE HAPPENS EXCEPT THE PROGRAM HANGS IN A TIGHT LOOP WAITING FOR THE DEVICE TO RAISE ITS FLAG.

WAIT LOOP - INEFFICIENT

QUESTION:

HOW DOES THE PROGRAM INTERRUPT IMPROVE OVER THE WAIT LOOP?

ANSWER- BY ALLOWING THE MAIN PROGRAM TO RUN UNTIL THE DEVICE INTERRUPTS THE MAIN PROGRAM TELLING THE PROCESSOR THE DEVICE NEEDS SERVICING.



QUESTION:

WHO MADE THE INTERRUPT ?

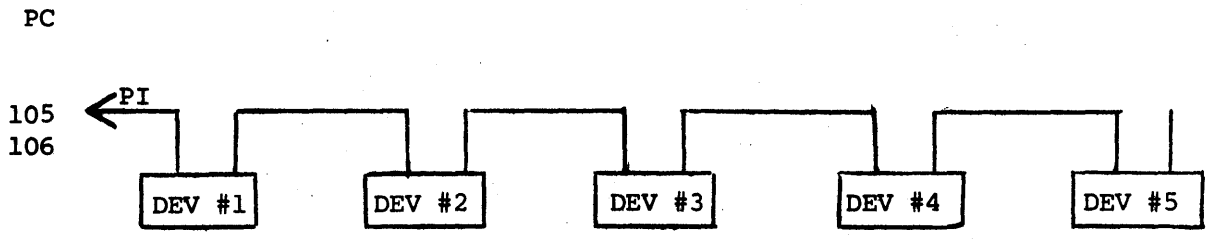
ANSWER - DON'T KNOW

QUESTION:

HOW CAN WE FIND OUT WHO MADE THE INTERRUPT?

ANSWER - BY SENDING OUT IOTS THAT CHECK THE FLAGS OF EACH DEVICE. RESPONDING TO THE DEVICE THAT HAS ITS FLAG RAISED OR BY READING STATUS (IORS)

PROGRAM INTERRUPT FACILITY (P1)



WHEN USING THE IOT SKIP CHECK PROCESS, A SKIP CHAIN IS USED. IN THE SKIP CHAIN PLACE THE DEVICES WHICH HAVE HIGHEST PRIORITY 1st AND LOWEST PRIORITY LAST.

EXAMPLE OF SKIP CHAIN

PFSF - 703201/SKIP ON PWR LOW FLAG
JMP. +2
JMP TO PWR FAIL ROUTINE
KPSF - 706301/SKIP ON DISK FLAG (RP-15)
JMP. +2
JMP TO DISK PACK SERVICE ROUTINE (ERROR + JOB DONE)
DTDF - 707601/SKIP ON DECTAPE FLAG
JMP. +2
JMP TO DECTAPE SERVICE ROUTINE (ERROR + JOB DONE)
MTSF - 700301/SKIP ON MAGTAPE FLAG
JMP. +2
JMP TO SERVICE ROUTINE FOR MAG TAPE (ERROR + JOB DONE)
KSF - 700301/SKIP ON KEYBOARD FLAG
JMP. +2
JMP TO KEYBOARD TTY SERVICE ROUTINE
JMP TO ERROR ROUTINE

KEYBOARD SERVICE ROUTINE

1. SAVE REGISTERS THE MAIN PROGRAM IS USING.
2. KRB/READ THE BUFFER INTO THE ACCUMULATOR
3. DAC INTO INPUT BUFFER
4. JMP TO EXIT ROUTINE

EXIT ROUTINE

1. ION TURN PROGRAM INTERRUPT FACILITY BACK ON.
2. RES/DBR - SET UP RESTORING OF L,B,UM
3. JMP*0 - GET BACK TO LOCATION 106

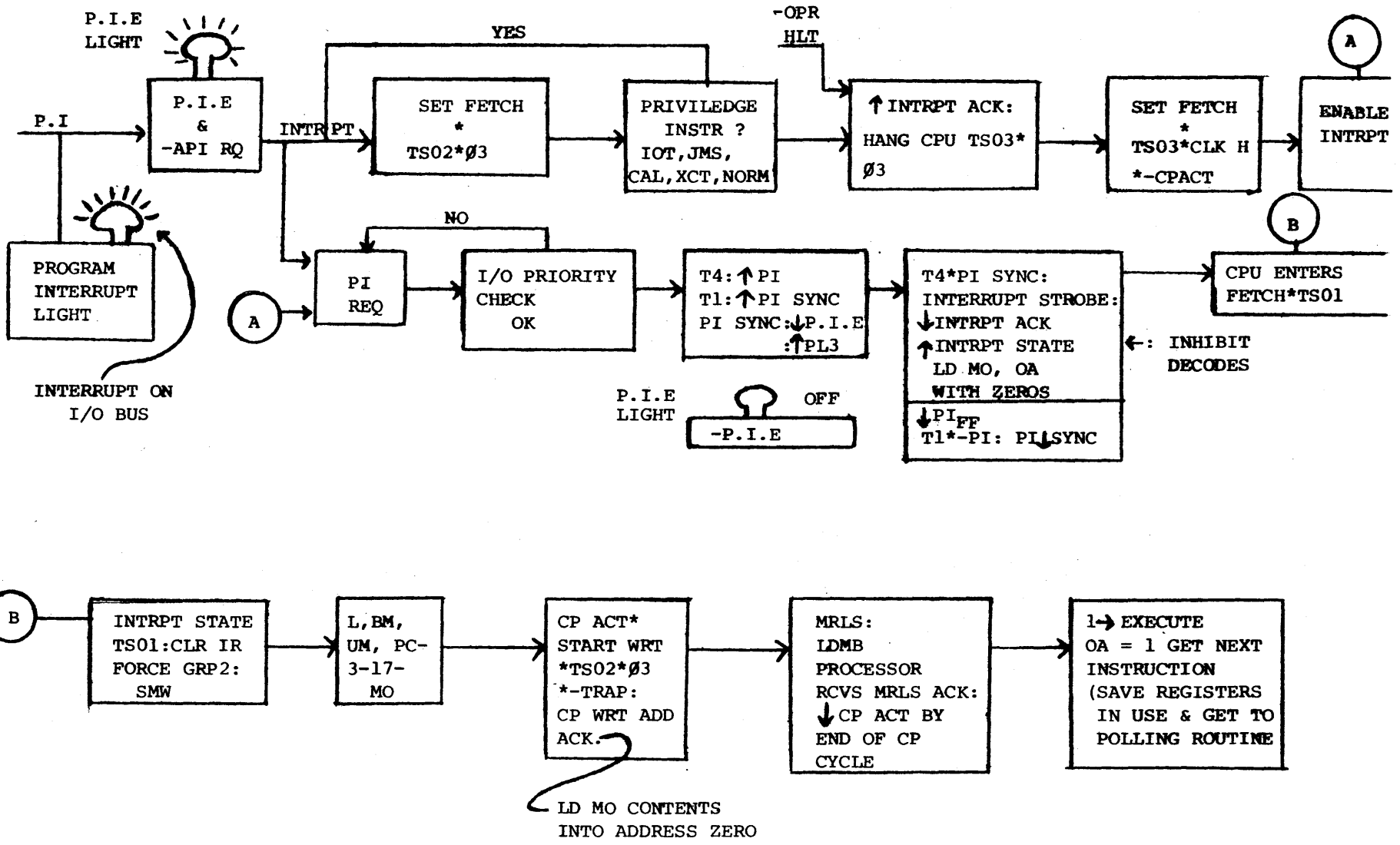
NOW IT CAN BE SEEN WHAT HAS TO HAPPEN WHEN AN INTERRUPT IS RECEIVED.

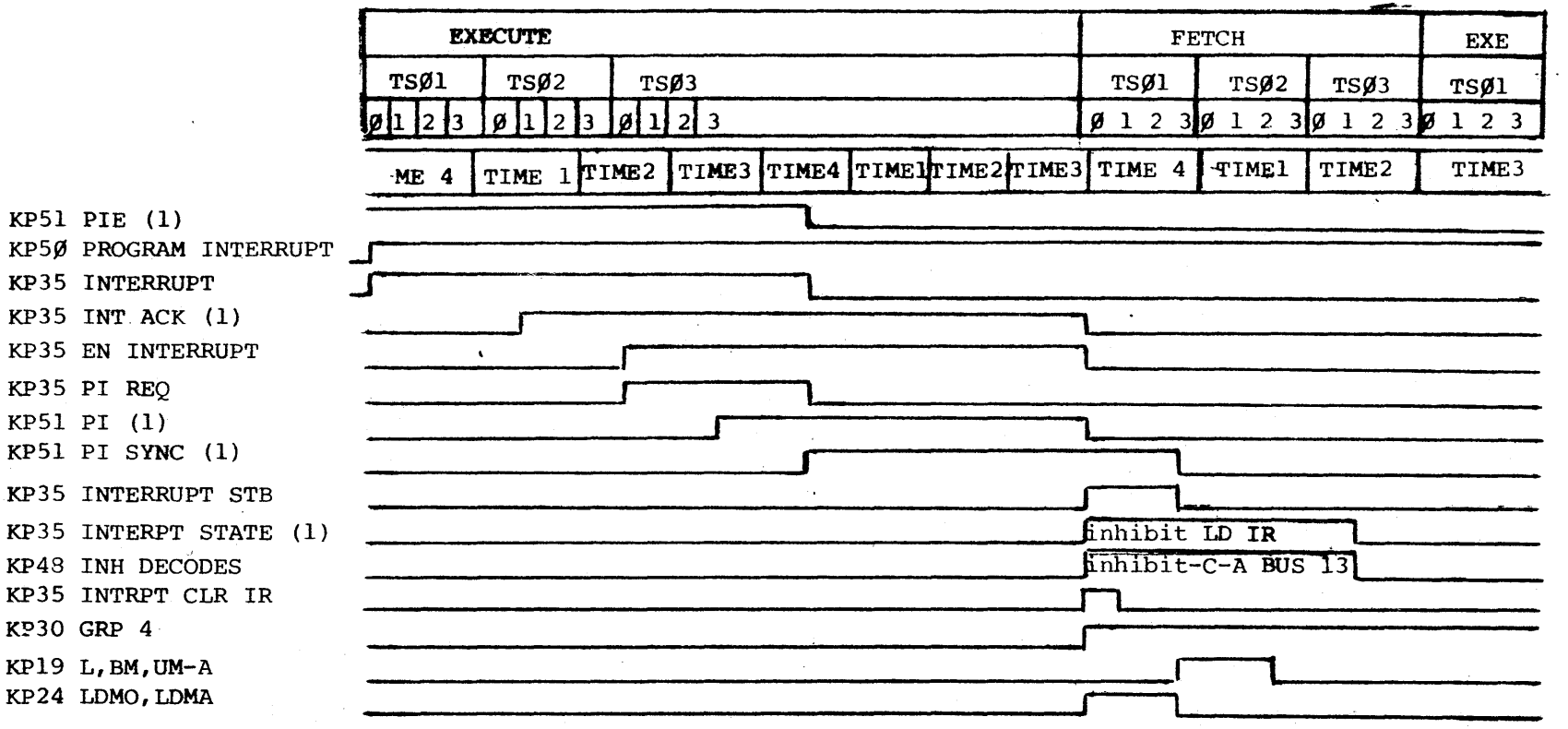
1 - LET CPU FINISH INSTRUCTION

2 - HANDLE INTERRUPT (TURN OFF INTERRUPT FACILITY)

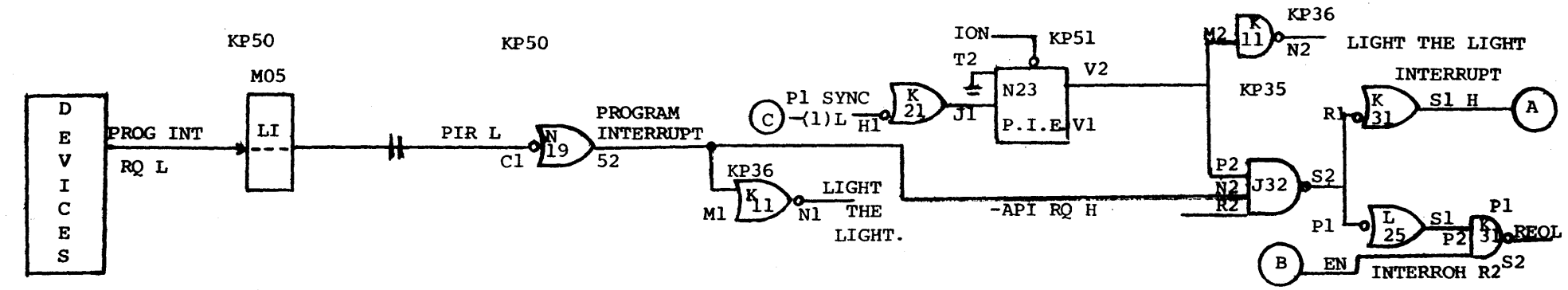
a) WRT L,BM,UM, PC3-17 INTO ADDRESS ZERO
b) GET NEXT INSTRUCTION - GET TO SKIP CHAIN
c) FIND OUT WHO RAISED THE FLAG
d) SERVICE DEVICE
e) EXIT THIS ROUTINE BY TURNING ON THE INTERRUPT FACILITY & IF NEEDED RESTORE L,BM,UM
f) JMP * 0 GET THE NEXT ADDRESS AND PROCEED IN THE MAIN PROGRAM.

PROGRAM INTERRUPT FACILITY

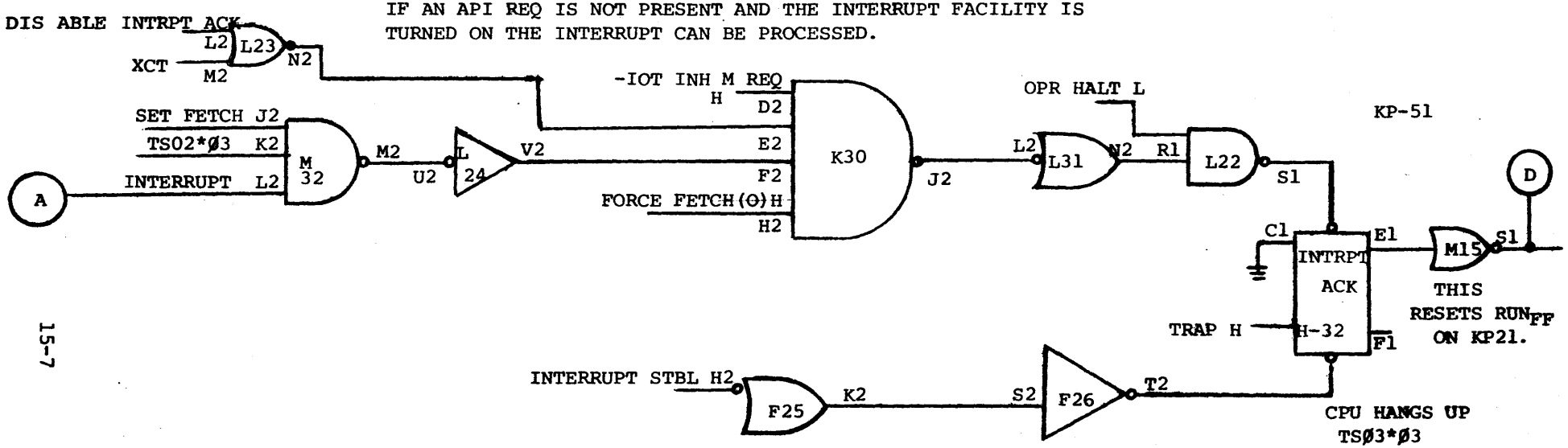




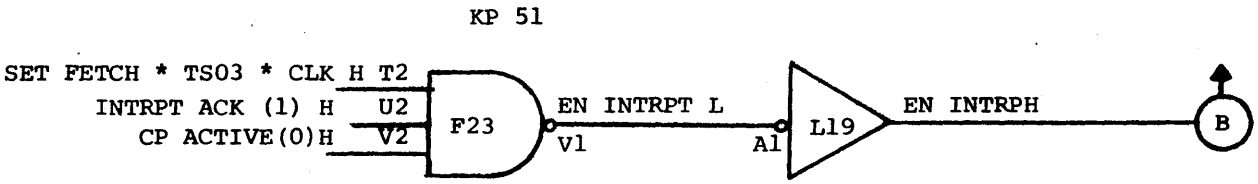
PROGRAM INTERRUPT TIMING



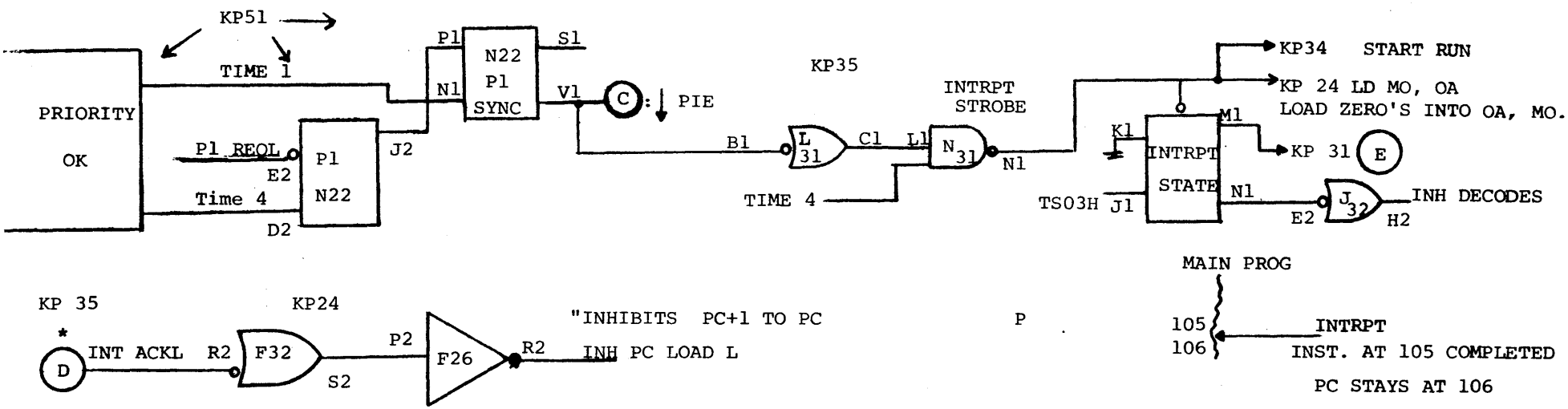
STATEMENT: DEVICE RAISES PROGRAM INTERRUPT SENTS TO PROCESSOR AND IF AN API REQ IS NOT PRESENT AND THE INTERRUPT FACILITY IS TURNED ON THE INTERRUPT CAN BE PROCESSED.



STATEMENT: CPU LOOKS FOR INTERRUPTS. (INSTRUCTION HAS BEEN PROCESSED). CHECK TO SEE IF THE INSTRUCTION WAS A PRIVILEGED INSTRUCTION, IF SO, DON'T SET INTRPT ACK, IF NOT A PRIVILEGED INSTRUCTION SET INTRPT ACK.



STATEMENT: CPU IS DONE WITH THE MEMORY CYCLE GENERATE A PI REQ.



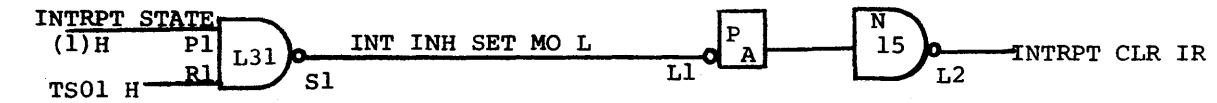
INTRPT ACK PREVENTS THE PC VROM BEING INCREMENTED

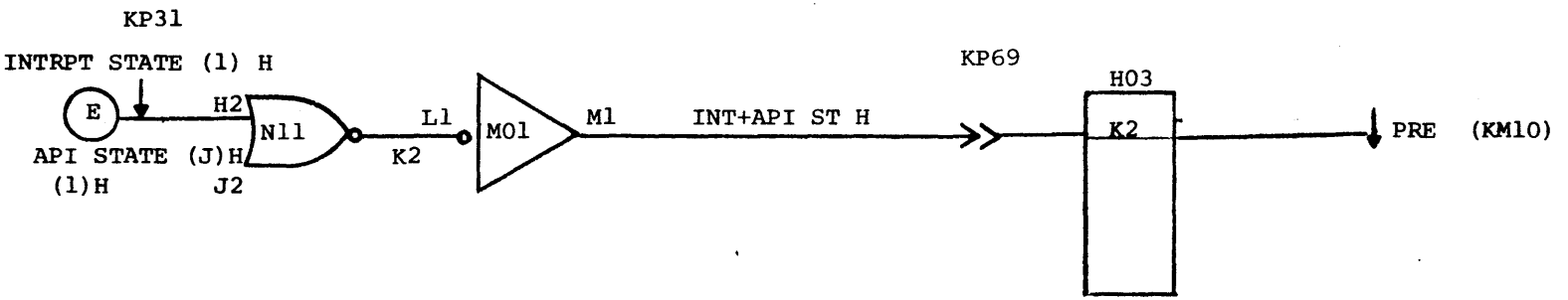
STATEMENT: PRIORITY CHECK MADE & SETTING PISYNC TURNS OFF PIE FACILITY. THE NEXT T4: INTRPT STROBE

INTRPT STROBE: LD MO, OA WITH ZERO

: ↑ START RUN ALLOW CPU TO GO TO FETCH MAJOR STATE

: ↓ INTRPT ACK

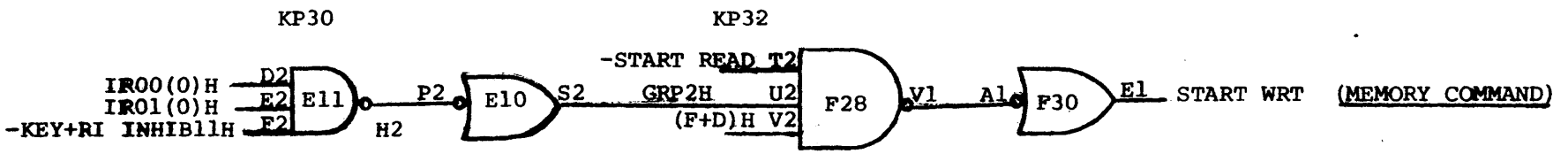




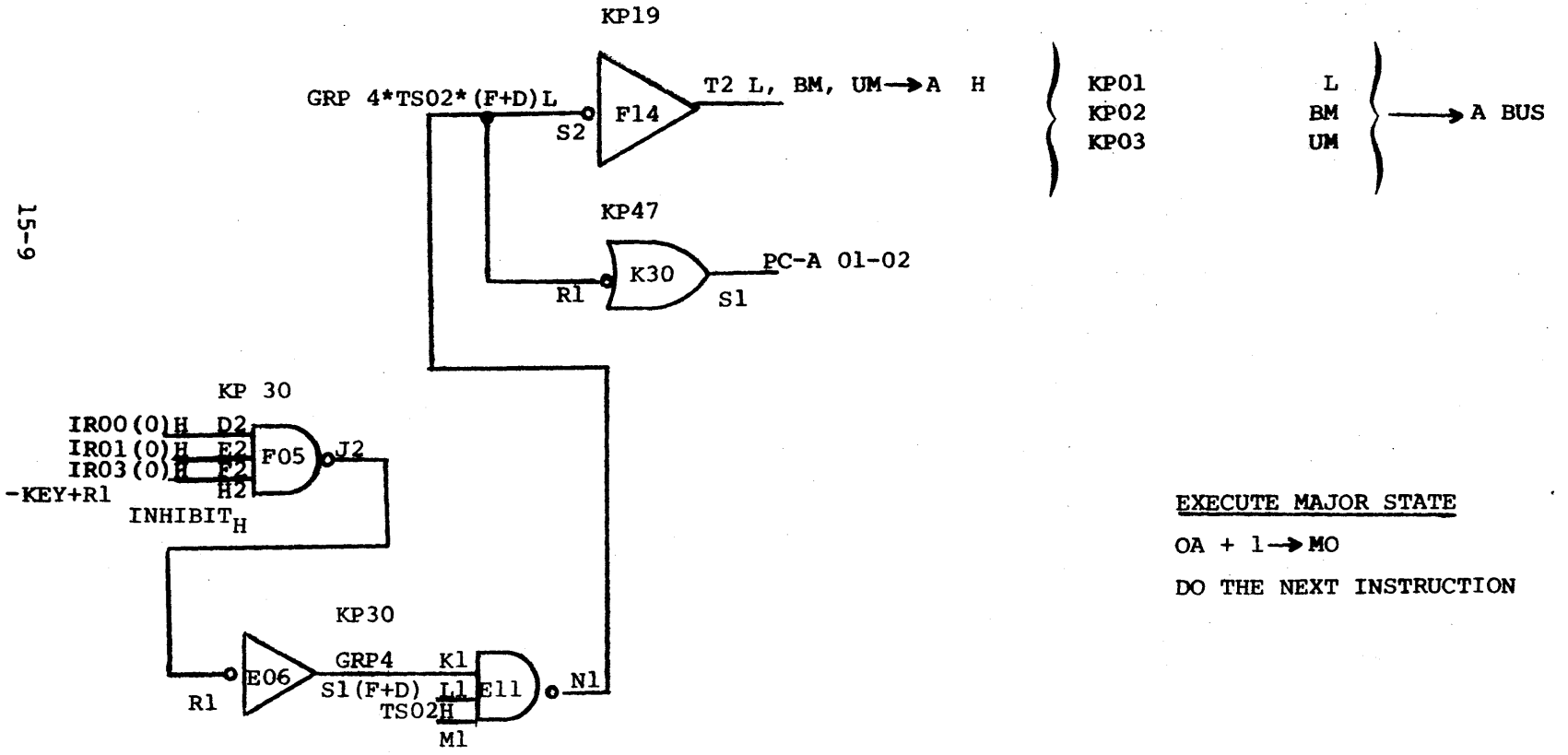
CLR IR 00-05

KP30- INH DECODES

ADDRESS - ZERO IS IN MO, OA



DO THE MEMORY CYCLE ROUTINE



EXECUTE MAJOR STATE

OA + 1 → MO

DO THE NEXT INSTRUCTION

TOPIC BA-15 PHERIPHERAL EXPANDER

OBJECTIVES: Upon completion of this unit the student will be able to:

1. State the function of the BA
2. Draw a block diagram of the BA
3. Construct the relationship between the BA & PC
4. Differentiate between the systems included in the BA

A. PERIPHERAL EXPANDER

1. Purpose

BA is a universal controller used to minimize bus loading by serving as a control for options that are infrequently used.

2. Options

a) PC-15 high speed paper tape/reader, punch
   (1) Reads 300 characters/sec
   (2) Punches 50 characters/sec

b) LT15A - Single teletype control
   (1) Background/foreground

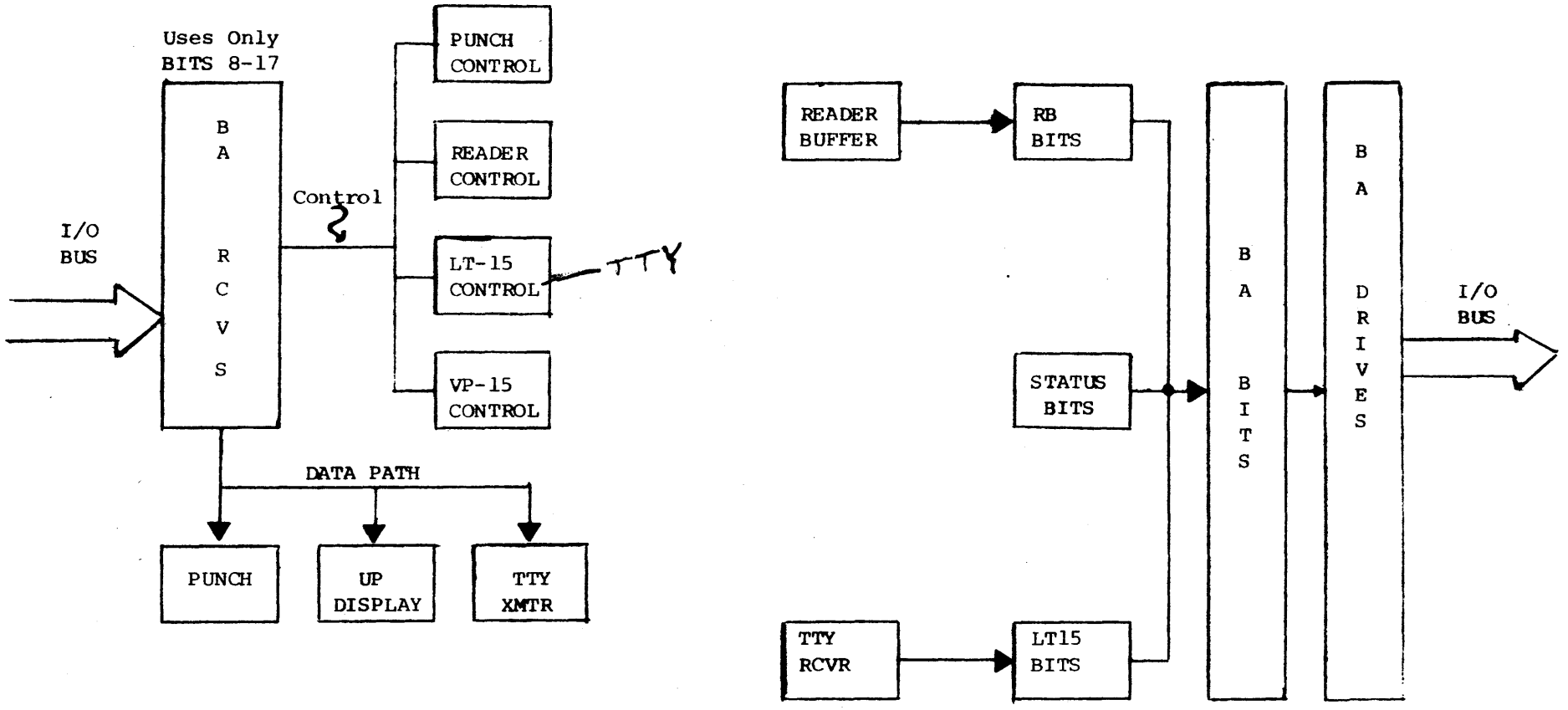
c) VP15 Display Console
   (1) Interfaces various display devices by providing D/A convertors and control logic for X-Y positioning as well as intensity.

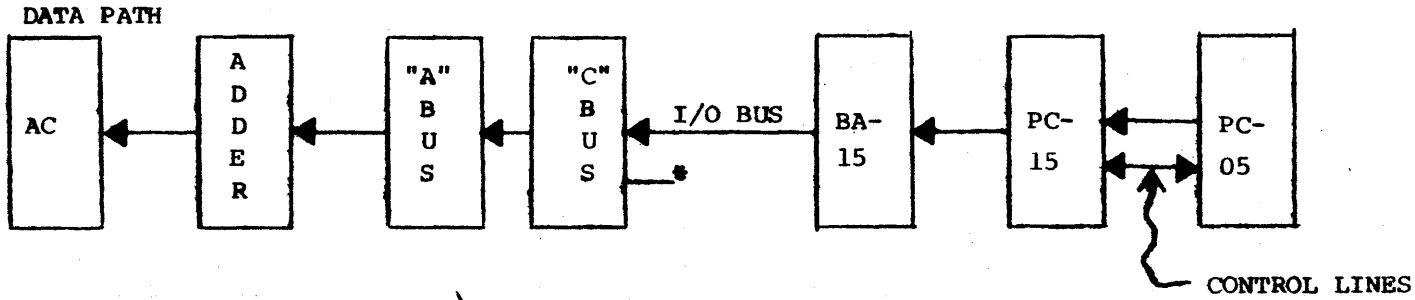
d) Discuss the block diagram of the BA

WRITE OPERATIONS

BA-15 BLOCK DIAGRAM

READ OPERATIONS





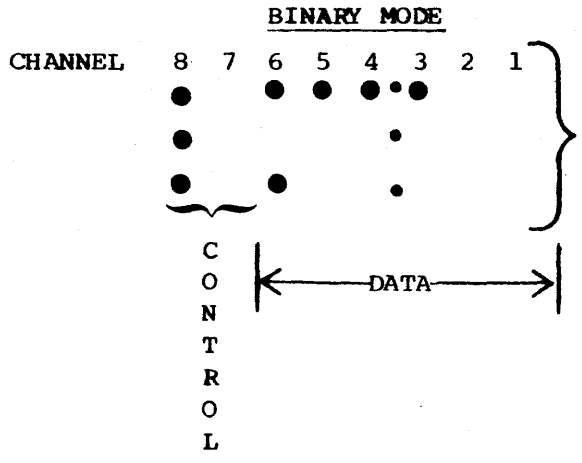
* EN 10P2:BAC→C BUS
EN 10P2*IOT REQ:I/O BUS→C BUS

Contents of AC ored with data on I/O Bus. Result loaded into AC

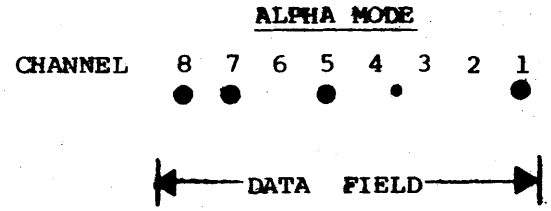
PC15 - 2 modes of reading RSA - 700104 - Select Aplhanumeric Mode
RSA - 700144 - Select Binary Mode

TAPE FORMAT

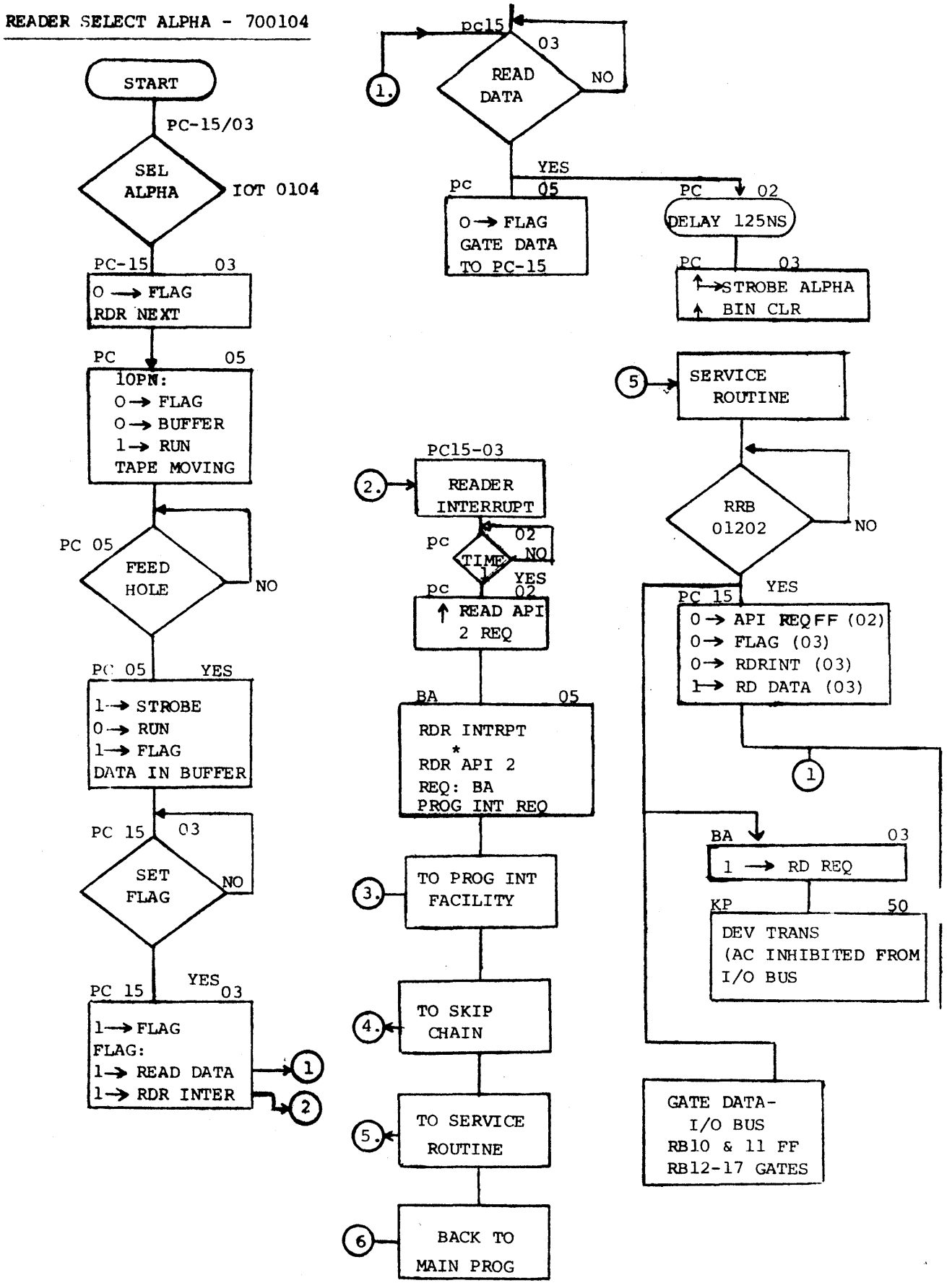
BINARY MODE Assembles One 18 Bit Word

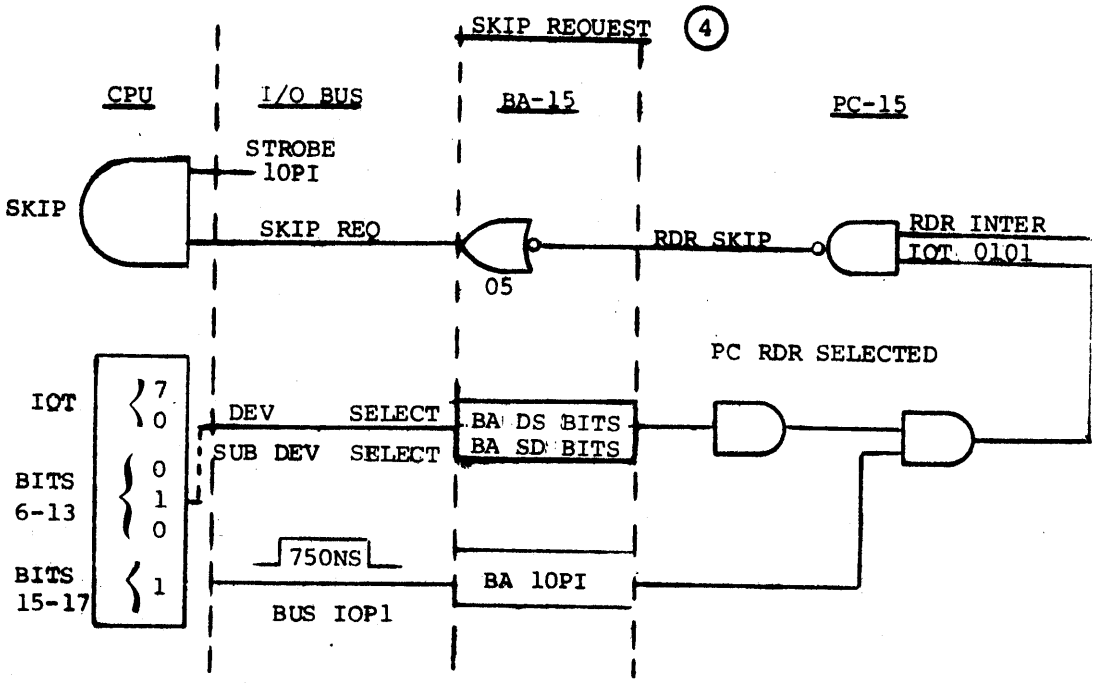


ALPHA Mode Assembles One 8 Bit Word



READER SELECT ALPHA - 700104



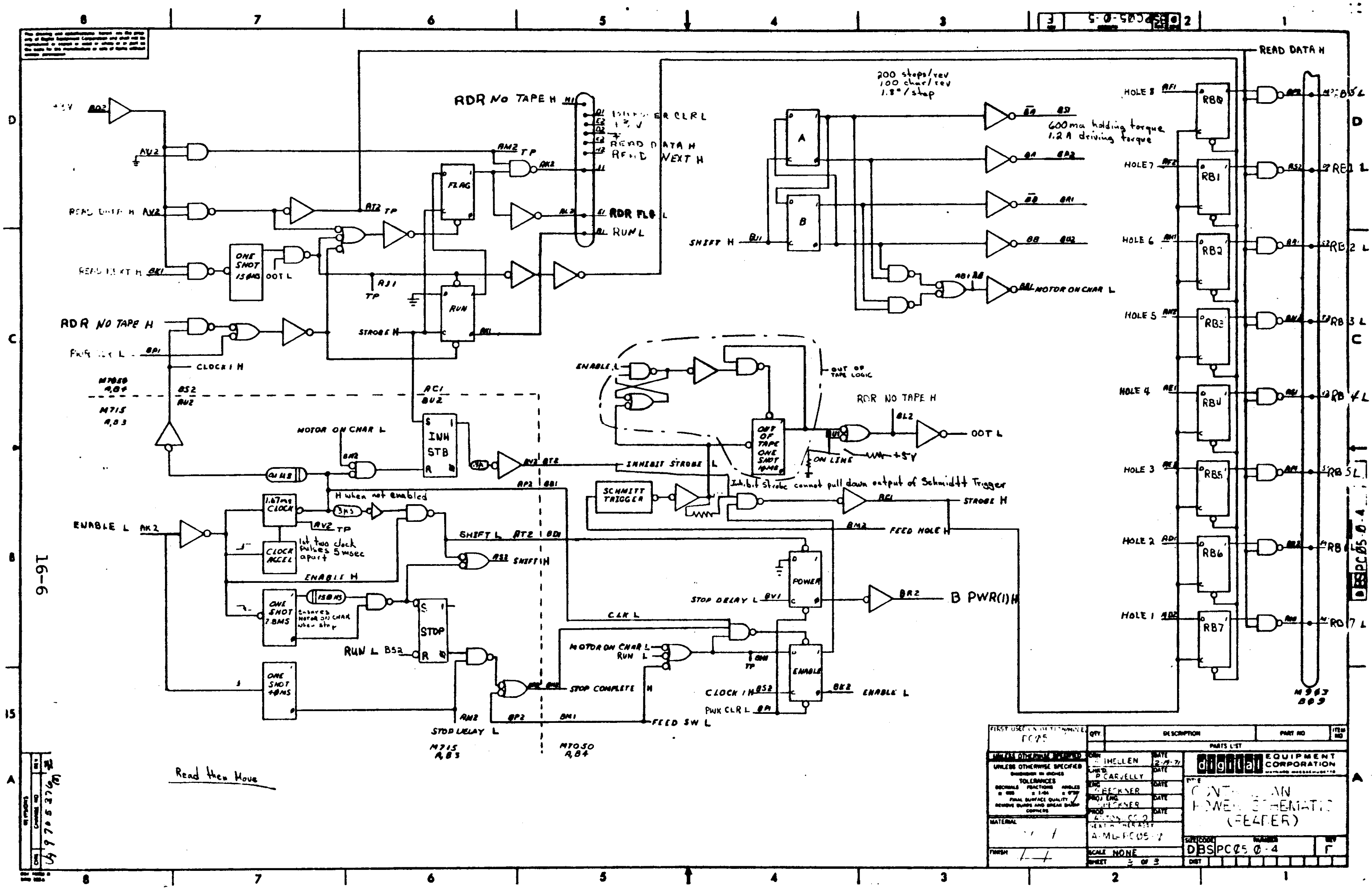


SKIP CHECK WHO MADE THE INTERRUPT

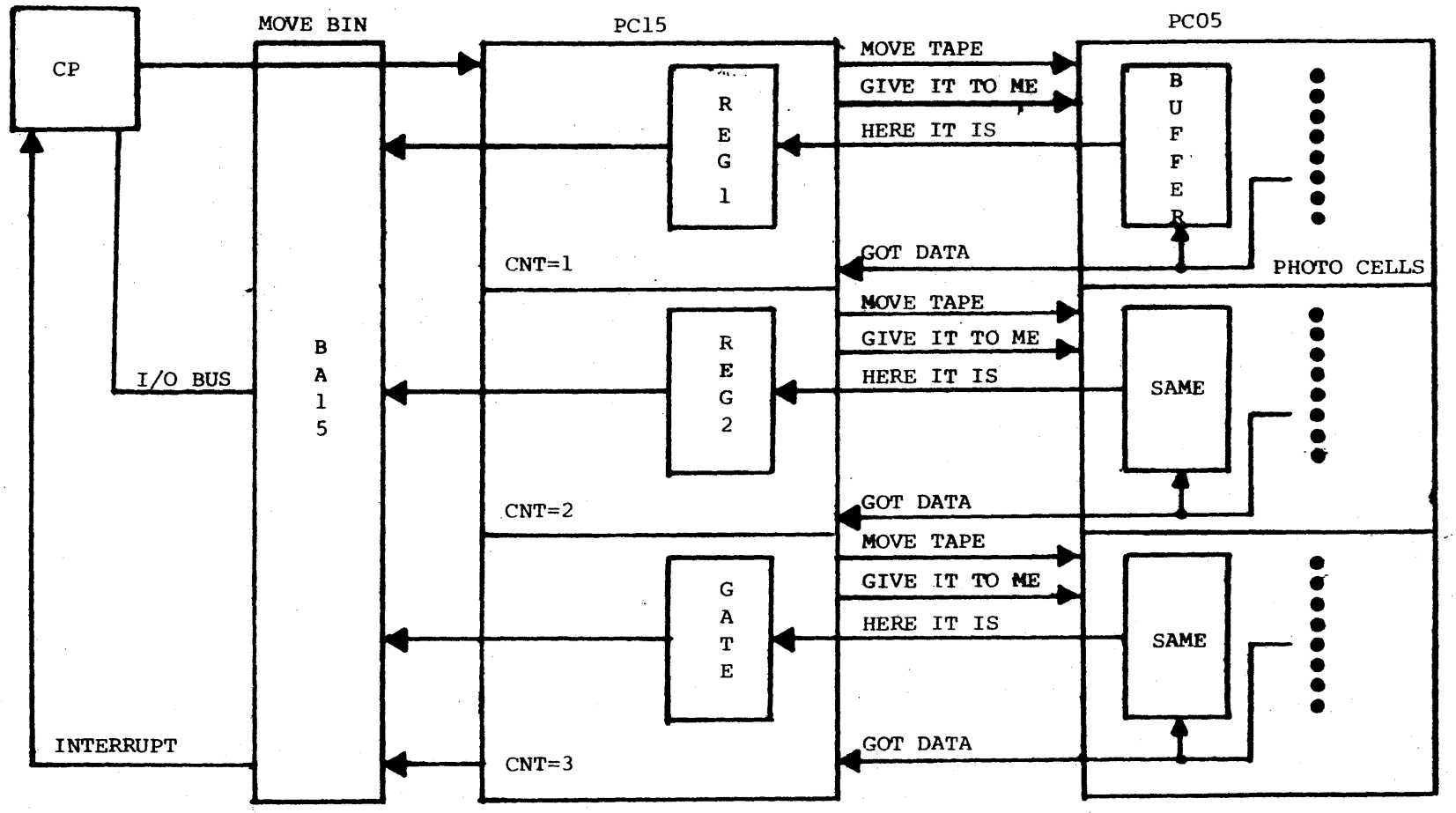
ANSWER THE PAPER TAPE READER

GO TO THE SERVICE ROUTINE FOR PAPER

TAPE READER RRB (700112) READ READER BUFFER


READ DATA H
RDR NO TAPE H
READ DATA H
READ NEXT H
RDR FLG L
RUN L
200 steps/rev
100 char/rev
1.8° / step
600 ma holding torque
1.2 A driving torque
HOLE 8
HOLE 7
HOLE 6
HOLE 5
HOLE 4
HOLE 3
HOLE 2
HOLE 1
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
RB0 L
RB1 L
RB2 L
RB3 L
RB4 L
RB5 L
RB6 L
RB7 L
M903
B09
A
B
SHIFT H
MOTOR ON CHAR L
+5V
FLAG
RUN
ONE SHOT 15 MS
OOT L
CLOCK I H
STROBE H
RDR NO TAPE H
M7050
A,B4
M715
A,B3
ENABLE L
OUT OF TAPE LOGIC
OUT OF TAPE ONE SHOT
ON LINE
RDR NO TAPE H
OOT L
MOTOR ON CHAR L
INH STB
INHIBIT STROBE L
Inhibit Strobe cannot pull down output of Schmidtt Trigger
H when not enabled
SCHMITT TRIGGER
STROBE H
FEED HOLE H
1.67 ms CLOCK
CLOCK ACCEL
1st two clock pulses 5 msec apart
ENABLE H
SHIFT L
SHIFT H
ONE SHOT 7.8 MS
ensures motor on char when stop
STOP
RUN L
ONE SHOT 40 MS
STOP DELAY L
POWER
B PWR(1) H
CLK L
MOTOR ON CHAR L
RUN L
STOP COMPLETE H
CLOCK 1 H
PWR CLR L
ENABLE
ENABLE L
FEED SW L
Read then Move
16-6
DIGITAL EQUIPMENT CORPORATION
CONTROL AND POWER SCHEMATIC (READER)
SCALE NONE
SHEET 3 OF 3
D BS PC05 0 4
REV F

PAPER T READER



CONTROLLER		DEVICE	
MOVE TAPE	→	READ NEXT	(10P4)
GOT DATA	←	RDR FLAG	
GIVE IT TO ME	→	READ DATA	(10P2)
HERE IT IS	←	DATA	

CONTROLLER SEQUENCE

READ DATA : STARTS CONTROLLER SEQUENCE
DELAYS 125NS

1. READ STROBE:LD REG
2. DELAY 75NS;ADV.STROBE#1-CNTS THE COUNTER
   CHECKS FOR CNT OF 3*FLAG:RDR INTRPT
3. DELAY 50NS;ADV.STROBE 2: NOT CNT 3 MOVE TAPE

DLE 20

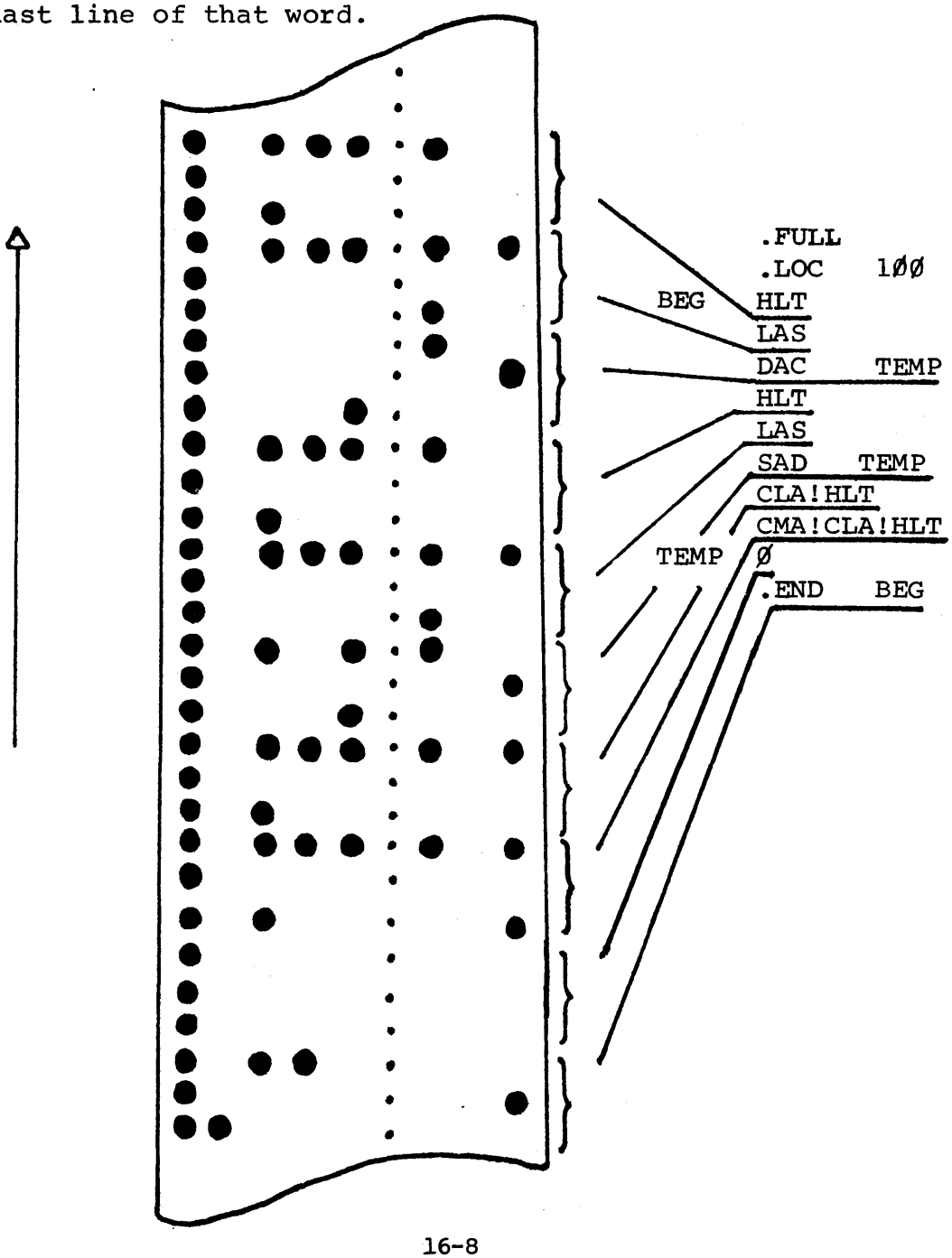
PAPER TAPE FORMATS

Program tapes are supplied in one of two formats:

1. HRI (HRM) Hardware Read-In Mode (.FULL assembly parameter)

2. BINARY or ABS (.ABS assembly parameter)

HRI Tapes consist of 18 bit data and instructions punched in binary mode (PSB), which are loaded in sequential memory locations via the HARDWARE READ-IN feature. The last word is an instruction which is to be executed when read (i.e., HLT or JMP). The last word is indicated by hole #7 being punched in the last line of that word.



ABS or BINARY paper tapes consist of 3 basic parts:

1. ABS loader program in HRI

2. Data blocks

3. Start block

The ABS loader is a program in HRI format which when read via the READ-IN key will load the remainder of the tape under program control (BIN LOADER). The BINARY LOADER expects the rest of the tape to be in a block format.

DATA BLOCK - consists of 3 control words:

1. LOAD ADDRESS

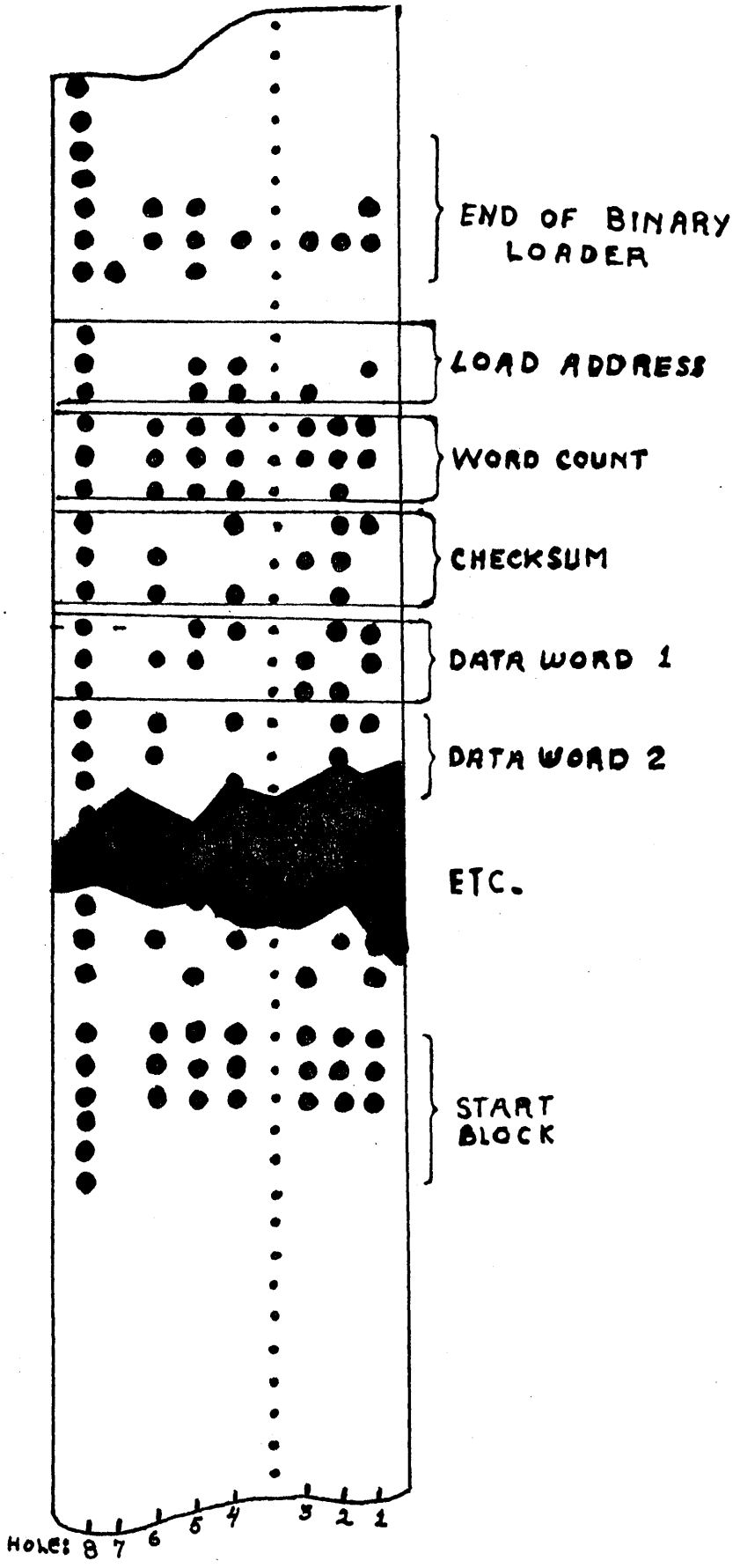
2. WORD COUNT (not exceeding 25)

3. CHECKSUM

START BLOCK - A two word block at the end of tape. It is distinguished from a data block because bit Ø of the first word is a one.

1. STARTING ADDRESS (-1=hlt)

2. DUMMY WORD (not used)



.ABS PAPER TAPE FORMAT

SUBJECT: HARDWARE READ-IN

SET ADDRESS IN SWITCHES

KP66 DEPRESS READ IN, SET PCO READ IN
KP50 PCO READ IN: DS05*SDOO : PCO SEL * BINARY
KP34 KEY ACTIVE * START RUN : ENABLE CPU TIMING TO RUN

KP66 KEY ACTIVE * PCO READ IN : RD IN START
KP50 RE IN START : I/O PWR CLR WHICH RESETS FLAG & RUN IN PC05

PC03 PCO SEL: ENABLE SET OF RI FF * PWR CLR : ↑ RI_{FF}
RI * $\overline{10P2}$: RI START
PC02 RI START PULSE : RI START + (SET ALPH)
PC03 RI START + (SET ALPH):
SELECT BINARY : ↓ ALPHA FF
RESET CNTR TO A COUNT OF ZERO
CAUSE READ NEXT TO GO HIGH

THIS CHANGE GETS SENT TO THE DEVICE (READ NEXT WILL BE SEEN AS A 10P4 WHICH FIRES A SINGLE SHOT AND PROVIDED YOUR NOT OUT OF TAPE, THE $FLAG_{FF}$ WILL BE RESET AND THE RUN_{FF} WILL BE SET. THIS IMPLIES TAPE IS MOVING.

FETCH IN THE MEANTIME THE CPU HAS MOVED INTO TS01 OF THE
TS01 FETCH MAJOR STATE.

KP34 READ IN * ($\overline{\text{NEXT}}$ * TS01) : ADD SW-A
ADDRESS IS NOW ON SUM BUS

KP49 KEY ACTIVE + READ IN (1) H: KEY + READ IN INHIBIT/
INHIBITS GRP 2,4
KP48 READ IN : INHIBIT DECODES

KP24 TIME STATE 1 * CLK H * KEY ACTIVE : LD MO & OA
WITH THE ADDRESS ON SUM BUS BUT AT THIS TIME A START WRITE CYCLE IS NOT INITIATED BECAUSE OF THE INHIBIT GROUP 2,4.

FETCH KP49 READ IN * $\overline{\text{READ IN EXEC}}$: RI PAUSE
TS01 KP34 READ IN PAUSE ENABLES "D" INPUT OF STOP TS RUN
TS02 SETS "STOP TS RUN"
KP21 STOP TS RUN & OTHER LEVELS WILL CAUSE THE RUN_{FF} TO BE RESET, THE CPU HANGS UP IN TS03, 03.

WAITS FOR READER TO ASSEMBLE 3 FRAMES AND GENERATE AN INTERRUPT

DURING THIS TIME TAPE HAS BEEN MOVING AND FOR EACH FEED HOLE, PROVIDING IN THE DEVICE CONTROLLER THE $ENABLE_{FF}$ IS SET, A STROBE WILL OCCUR WHICH GATES WHAT EVER IS BEING READ INTO A BUFFER REGISTER AND SETS THE $FLAG_{FF}$ (IN THE DEVICE)

THIS GOES OVER TO THE PC15-02 AS READER FLAG
AND IS CHANGED TO SET FLAG IN THE PC15-02

PC15-03 SET FLAG : ↑ FLAG FF
FLAG : READ DATA (DATA IS FOR THE TAKING)
PC15-02 READ DATA : 10P2 ON CABLE BACK TO PC05

BACK IN THE PC05 THE 10P2 ↓ FLAG AND ENABLE THE DATA FROM THE BUFFERS ONTO THE CABLE TO PC02 RB0 THROUGH RB7

RB0 : HOLE 8 : THIS IS DATA
RB1 : HOLE 7 : WITH MORE TO COME

RB0 : HOLE 8 : THIS IS THE LAST WORD
RB1 : HOLE 7 : EXECUTE THE INSTRUCTION
READ SEQUENCE
PC15-02 125ns AFTER READ DATA : READ STROBE
PC15-03 READ STROBE * CNT OF 0 * ALPHA : STROBE 1

STROBE 1 : LOAD THE SIX DATA BITS FROM PC05 BUFFER INTO RB00 THROUGH RB05

PC15-02 200ns AFTER READ DATA : ADVANCE STROBE 1
PC15-03 ADVANCE STROBE 1 * RB0 (HOLE 8) : CLK CNTR

CNT OF 1 : GOT THE FIRST 6 BITS

PC15-02 250ns AFTER READ DATA : ADVANCE STROBE 2 PULSE
ADVANCE STROBE 2 : DO WE HAVE A CNT OF "3" YET.
NO
AT THIS TIME READER NEXT GOES HIGH
AT THE PC05 THIS LEVEL FIRES THE SINGLE SHOT
PC15-03 ADVANCE STROBE 2 : ↓ FLAG FF IN PC03

IN THE PC05 READER NEXT IS RECEIVED AS 10P4 WHICH RESETS FLAG * SETS RUN TAPE IS NOW MOVING TOWARDS THE SECOND CHARACTER.
RESETTING OF THE $FLAG_{FF}$ ON PC15-03 REMOVES READ DATA FROM ITS ACTIVE STATE (10P2 AT PC05 REMOVES BUFFER OUTPUT FROM LINES TO PC15-02.)

NEXT FEED HOLE * ENABLE : STROBE
STROBE :DATA IN PC 05 BUFFER
: ↓ RUN $_{FF}$ IN PC05
: ↑ FLAG

AS CAN BE SEEN THE SECOND 6 BIT FRAME IS NOW IN THE BUFFER AND WE FOLLOW THE SAME SEQUENCE AS IN THE PRECEEDING PAGE; EXCEPT WHEN READ DATA PUTS THE DATA ON THE LINE.

STROBE * CNT OF 1 * $\overline{\text{ALPH}}$: STROBE 2 :
STROBE ALPH RB06 THROUGH RB11 WILL BE
LOADED

ADVANCE STROBE 1 * 8 HOLE : CNT =2

ADVANCE STROBE 2 : CNT OF 3 ANSWER NO, FIRE
CKTRY TO SET RUN & RESET
FLAG IN THE PC05. ALSO
CLEAR BUFFER IN PC-05

PC03 RD START (L) RESETS FLAG

TAPE MOVING TOWARD LAST FRAME OF ONE COMPUTER
WORK (18 BITS)

AS SOON AS FLAG COMES UP THE PC05 THIS IMPLIES
DATA IS NOW IN THE PC05 BUFFER AND THE RUN
FF IS RESET

PC02 FLAG FROM PC05 GENERATES SET FLAG WHICH
IN TURN SETS THE FLAG FF ON PC03

PC03 FLAG : READ NEXT (10P2 to PC05) : ↓ FLAG
IN PC05 &

ENABLE DATA ON LINES RBO THROUGH
RB7 to PC02

PC02 125 ns AFTER READ DATA : STROBE 1

DATA DOESN'T GET LOADED INTO ANY REGISTER

RB 2 : RB 12
RB 3 : RB 13
RB 4 : RB 14
RB 5 : RB 15
RB 6 : RB 16
RB 7 : RB 17

} LAST 6 BITS THROUGH GATES

200 ns AFTER READ DATA : ADVANCE STROBE 1

ADVANCE STROBE 1 * HOLE 8 : CNT 3

PC15-03 CNT 3 * FLAG : RD INTERRUPT

PC15-02 M104

RD INTERRUPT * I/O SYNC : RDR AP1 2 REQ

BA-05 RDR INTERRUPT * RDR AP1 2 REQ : PROGRAM
INTERRUPT

NOTE: 1 MS AFTER RDR AP1 2 REQ AN 10P2 WILL
BE GENERATED

250 ns AFTER READ DATA : ADVANCED STROBE 2

PC15-03 ADVANCE STROBE 2 * CNT 3 * $\overline{\text{ALPH}}$:

	RDR START H : WILL NOT RESET FLAG FF (PC15-03)
	RDR NEXT L : THE RUN FF WON'T GET SET
PC15-03	$\overline{\text{ADVANCE STROBE}}$ 2 : NO EFFECT
	CNT OF 3 OVER RIDES
KP51	READ IN READY * TIME 1 : ↑ 10P2
KP55	$\overline{\text{ENABLE I/O LINES}}$ * 10P2 : DCH R1 IOP2
KP50	DCH + R1 10P2 : BUS IOP2 (ONTO I/O BUS)
PC15-02	10P2 * PC SEL : IOT 0102
BA15-04	IOT 0102 : RBOO-RB17 : BA BITOO-BA BIT 17
BA15-03	DATA ON I/O BUS
	I/O BUS 00 THRU 17
KP49	READ IN * 10P2: PCO I/O CONT
	PCO I/O CONT : R1 ADV
	" " " : I/O STROBE AC EN
KP19	" " " : I/O BUS - C
	I/O BUS - C BUS : C-A DATA NOW ON SUM BUS
KP19	NO SHIFT - D BUS IS ACTIVE
	THE ACCUMULATOR IS ENABLED AND AT STROBE 10P2 TIME THE DATA WILL BE LOADED INTO THE ACCUMULATOR KP24 I/O STB AC EN * STROBE 10P2 : LD AC (KP51 10P2 * TIME 4 : STROBE 10P2
	750 ns AFTER 10P2 INITIATION DATA IS TUCKED SAFELY IN THE ACCUMULATOR RDR INTERRUPT ON PC15-03 GOT RESET AS A RESULT OF IOT 01 02
KP66 KP51	THIS MEANS ON BA15-03 $\overline{\text{PROG}}$ INTER PROG INTRPT : READ IN READY 250 ns AFTER THE DATA HAS BEEN TUCKED AWAY IN THE ACCUMULATOR THE NEXT TIME 1 THE 10P2 FF WILL BE RESET.
PC15-03	$\overline{\text{BA10P2}}$ * READ IN (1) L : GO THROUGH THE READER CYCLE FOR ANOTHER CHARACTER
	REMEMBER THE PROCESSOR HAS BEEN HUNG UP IN TS03 Ø3
KP34	READ IN ADVANCE * STROBE 10P2 * $\overline{\text{RUN}}$: START RUN

KP21 NEXT CLK : ↑ RUN

NEXT CLK : TS01 Ø0

KP66 STORE * $\overline{\text{TIME STATE}}$ 3 * 10P2 : ↓ FIRST $CHAR_{FF}$

$\overline{\text{FIRST CHAR}}$ * PCO READ IN : RI STORE

KP30 RI STORE : GRP 2

KP32 $\overline{\text{START READ}}$ * GRP 2 F : (START WRT)
NOTE: ADDRESS FROM SWITCHES HAD BEEN LOADED INTO MO & OA

TS02 FETCH * TS02 * RI STORE : DAC * TS02 (F + D) L

ACCUMULATOR CONTENTS MUST BE LOADED INTO THE MO REG

BAC - C, C-A : ON SUM BUS
KP 24 CP WR ADDR ACK : LD MO

DO THE WRT

TS03

KP49 RI STORE * TS03 * $\overline{\text{10P2}}$ * $\overline{\text{READ IN EXE}}$:

RI INC OA : RI + TEST LR CRY

KP48 RI + TEST LR CRY : CARRY INSERT

KP19 RI INCR OA : OA - A WITH CARRY INSERT UP DATES THE ADDRESS

KP24 RI INCR OA * CLOCK H * RL INCR OA : LDMO & OA

MO IS NOW SET UP FOR THE NEXT ADDRESS. THE ABOVE INFO (PAGE 1 THRU 5) OCCURS FOR EACH CHARACTER OF THE HARDWARE

876————1 READ IN
•0
•0
••

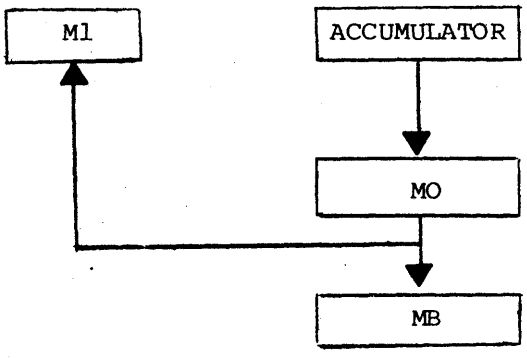
THE HANDLING OF THE LST CHARACTER, WHICH WILL BE A HLT OR

A JMP, WHICH GETS LOADED INTO THE ACCUMULATOR AS NORMAL IN

TS02:

FORCE A DAC TO STROE AC IN SPECIFIED LOC A WRITE OPERATION IS PERFORMED

TS03



MO-MDL * READ IN
EXECUTE *

TS03 * PHASE 1 : CP + R1
RD RST
R1 RD RST : CP MRLS
: END CP CYCLE
KP66: LD M1

GO TO FETCH IN INSTRUCTION IS IN THE MI DO IT !!

TS01, 01 LOAD IR

SIGNAL USED TO TERMINATE ARE

PC15-03 READ IN FINISH

BA15-03 READ IN FINISH : R1 SKP

KP50 SKP REQUEST

KP66 PCO READ IN & SKP : ENABLE "READ IN EXECUTE $_{FF}$"

10P2 & TIME 3 : "READ IN EXECUTE $_{FF}$"

NOTE : LAST INSTRUCTION READ INTO THE ACC AS NORMAL"

TS01 DO THE NORM (WRT)

TS02 FORCE THE DAC

TS03 KP32 MO - MDL * READ IN EXE * TS03 * 01 : CP + R1 RD RST

R1 RD RST : CP MRLS, END OF CP CYCLE, LD MI, RL RESET

R1 RESET : READ IN, PCO READ IN 1 - FETCH

XCT THE INSTRUCTION IN THE M1 REGISTER.

MEMORY PROTECT

OBJECTIVES:

A) LIST THE USE AND FUNCTIONS OF THE MEMORY PROTECT OPTIONS

B) SHOWN THE BLOCK DIAGRAM, EXPLAIN THE COMMUNICATION SIGNAL SEQUENCE AND DATA FLOW OF THE MEMORY PROTECT OPTION.

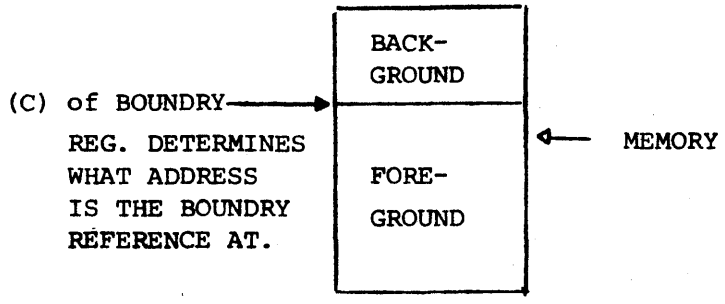
C) EXPLAIN THE FORMAT AND USE OF THE MEMORY PROTECT IOT INSTRUCTIONS

D) LIST THE PROGRAMMING CONSIDERATIONS NECESSARY WHEN USING THE MEMORY PROTECT OPTION.

E) EXPLAIN THE KM-MEMORY PROTECT FLOW DIAGRAM.
STATING THE LOGICAL
OPERATION OF A MEMORY PROTECT TRAP
CAUSED BY
1) BOUNDARY VIOLATION
2) ILLEGAL INSTRUCTION
3) NON-EXISTANT MEMORY

KM - MEMORY PROTECT

PROVIDES PDP-15 THE CAPABILITY OF RUNNING IN A BACKGROUND FOREGROUND ENVIRONMENT.

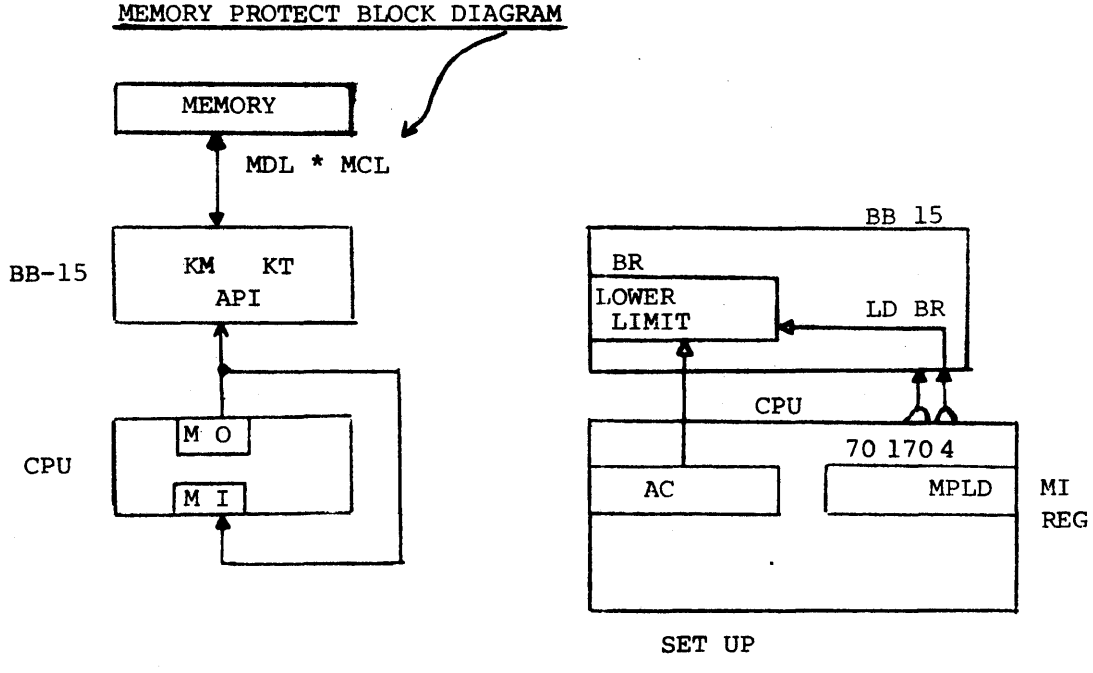


THERE IS A SPECIFIC SET OF INSTRUCTIONS TERMED ILLEGAL, BECAUSE THEY INTERFERE WITH BACKGROUND FOREGROUND OPERATIONS.

INSTRUCTIONS ARE: OAS, IOT, HLT, AND XCT OF XCT. ILLEGAL INSTRUCTIONS CAUSE TRAPS (INTERRUPTS). SO DO

1) REFERENCING BELOW THE BOUNDARY REGISTER-CHECK ADDRESSES ON ALL WRITE INSTRUCTIONS, JMP, AND ISZ INSTRUCTIONS.
2) REFERENCES TO NON-EXISTENT MEMORY

MEMORY PROTECT BLOCK DIAGRAM



SET UP

TRAP INVALID ADDRESS

KM-10 — IN TSØ3* PHASE 3, WITH SET FETCH ACTIVE, THE CENTRAL PROCESSOR IS WAITING TO RECEIVE THE NEXT INSTRUCTION. RD RST TELLS THE CPU THE INSTRUCTION IS ON THE MDL TAKE IT.

IF THE MEMORY PROTECT OPTION IS IN THE SYSTEM THE INSTRUCTION WILL ALSO BE LOADED INTO THE INSTRUCTION REGISTER IN THE KM LOGIC. THIS INSTRUCTION IS DECODED AND CHECKED FOR VALIDITY, (NO IOTS, HLT, OAS, XCT02), IF INSTRUCTION IS INVALID TRAP. ASSUMING THE INSTRUCTION IS A VALID INSTRUCTION, EXAMPLE TAKE A DAC INSTRUCTION, THE CPU GOES TO THE FETCH MAJOR STATE AND PROCESSES THE DAC INSTRUCTION, SENDING TO MEMORY THROUGH THE MEMORY PROTECT LOGIC, THE ADDRESS AND COMMAND (MWR,) FOLLOWED BY THE MREQ.

KM-12 — THE COMMAND MWR: PMODE ENABLE (CHECK THE ADDRESS.)

KM13-14 — THE ADDRESS ON MDL 1-9 + BOUNDARY REG 1-9 RESULTS IN NO CARRY IMPLIES THIS ADDRESS IS VALID PROCESS THE MEMORY CYCLE. IF THE ADD RESULTS IN A CARRY, THIS IMPLIES TRAP.

KP31 — WRT*TRAP:MEM TRAP
MEM TRAP: TRAP P A PULSE 110NS.

KP26 — TRAP P: ↓ MREQ

KP32 — TRAP P: ↑ ADR ACK
TRAP PREVENTS CP WR ADR ACK FROM OCCURING. ADR ACK: ↓ CP MREQ $HOLD_{FF}$

KP26 — ADR ACK: ↓ HOLD MO

KP31 — MEM TRAP*MREQ: TRAP CLR MEMORY

KP32 — TRAP CLR MEMORY: END OF CP CYCLE

KP26 — END OF CP CYCLE: ↓ CP ACTIVE

TRAP INVALID ADDRESS

CPU TO MEMORY CONTROL$_{FF}$ ARE NOW CLEARED.

KP35 CLR KEYS (TS03*Ø1)*TRAP*FORCE FETCH: ↑ INTRPT ACK

KP31 TRAP*TS03*Ø1: ↑ TRAP ADDRESS EN IF P.I.E=0 TRAP ADDRESS EN: CAL ADDR EN THIS FORCES ADDRESS 20. IF P.I.E. IS ENABLED GO TO LOCATION ZERO.

KP19 TS03* INTRPT ACK: I/O ADDRESS TO "A" BUS
FETCH TS01

KP21 INTRPT ACK*TS03*Ø2* HS CLK: ↓ RUN$_{FF}$. THIS HANGS THE CPU IN TS03, Ø3.

KP20 TRAP*INTRPT ACK*TS03*CLK H: MS PWR CLR. WHICH CHANGES THE CPU FROM THE FETCH MAJOR STATE TO THE EXECUTE MAJOR STATE. SET FETCH ALSO GOES ACTIVE.

KP35 (SET FETCH*TS03*CLK H)* INTRPT ACK* $\overline{\text{CP ACTIVE}}$:
ENABLE INTERRUPTS.

ENABLE INTERRUPT*TRAP: PI REQ

KP51 TIME4* PRIORITY CHECK OKAY: ↑ PI$_{FF}$
TIME1* PRIORITY CHECK OKAY: ↑ PI SYNC$_{FF}$
PI SYNC: ↓ P.I.E.$_{FF}$

KP35 PI SYNC*TIME4: ↑ INTERRUPT STROBE
INTRPT STB: ↑ INTRPT STATE
INTRPT STB: ↓ INTRPT ACK .

KP34 INTRPT STROBE*$\overline{\text{RUN}}$: ↑ START RUN

KP24 INTRPT STROBE: LD MO, OA

KP48 INTRPT STATE: INH DECODES

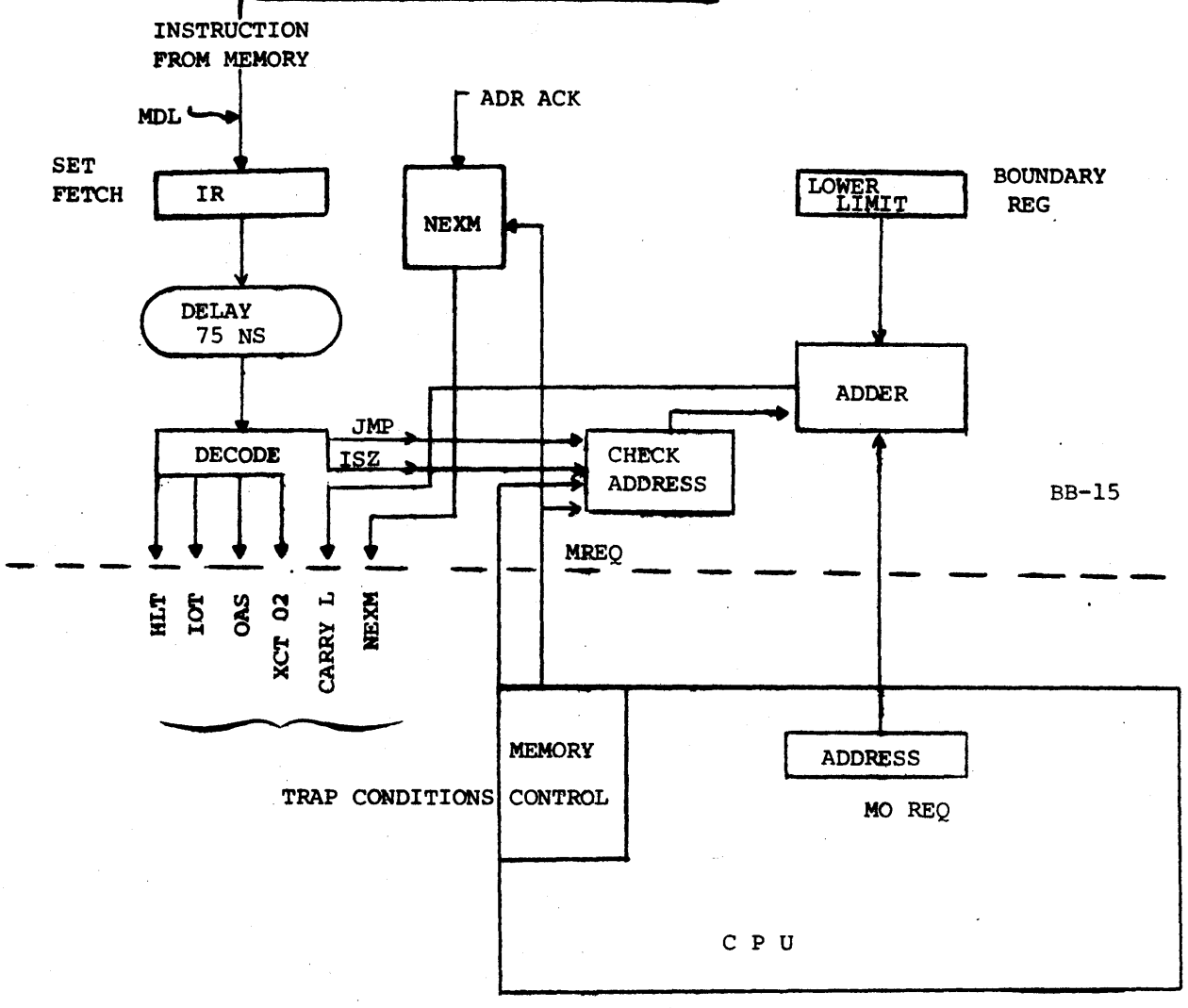
KP35 INTRPT ACK+INTRPT STATE: INH PC LOAD
THIS PREVENTS THE PC FROM INCREMENTING

KP31 INTRPT STATE + API STATE: ↓ TRAP$_{FF}$, ↓ PRE$_{FF}$, ↓ UM$_{FF}$
THIS TAKES THE SYSTEM OUT OF USER MODE AND ADDRESS O TAKES THE SYSTEM INTO MONITOR MODE

KP35 INTRPT STATE*TS01: INT. INH. SET MO.
INT. INH. SET MO: INTRPT CLR IR

FROM THIS POINT PROCESS AS IN PI
WRITE-L,B,U, AND PC 3-17 INTO LOCATION 20 or ZERO.

BLOCK DIAGRAM OF TRAP CONDITIONS



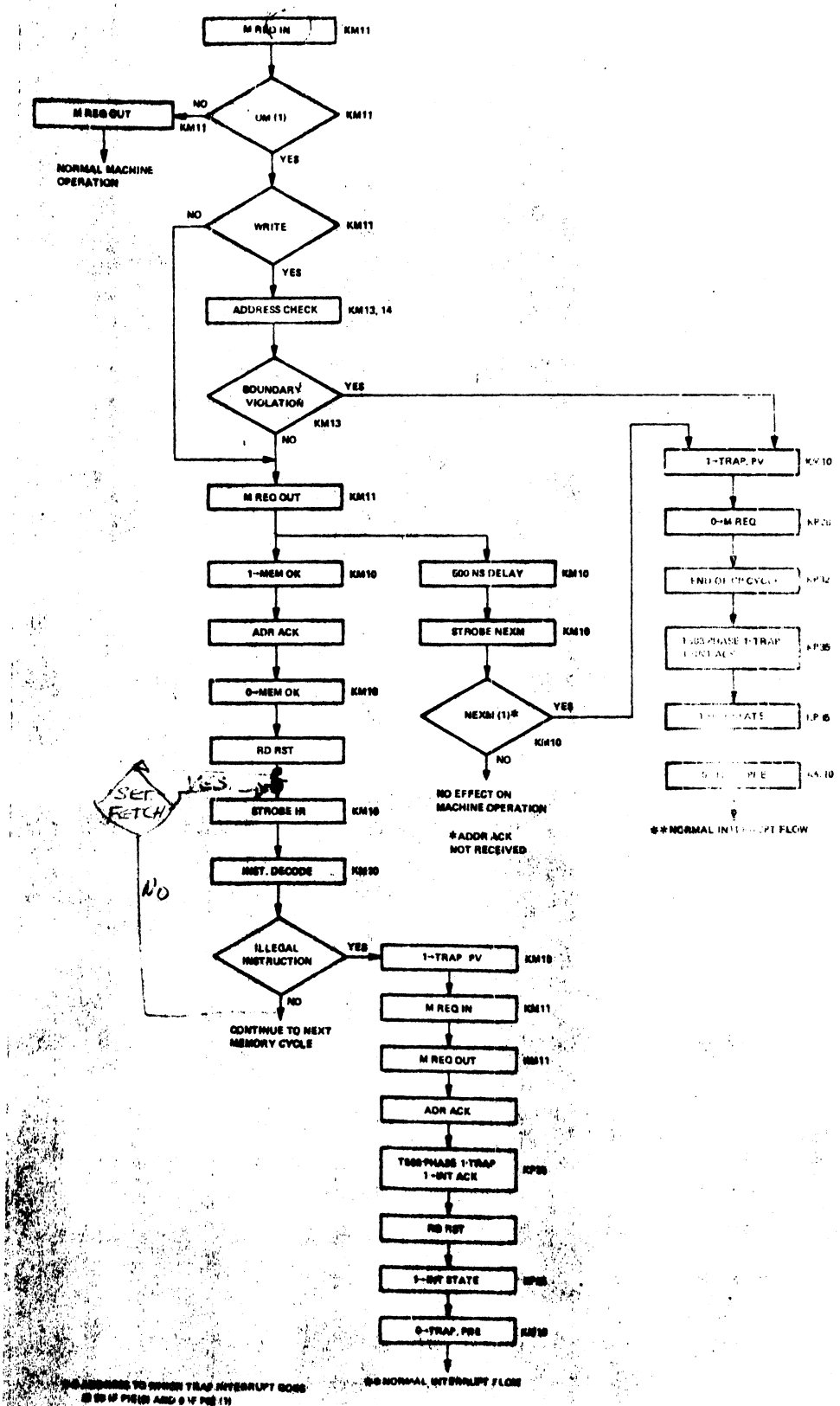


Figure 6-12 KM15 Memory Protect Flow Diagram

MEMORY RELOCATE - KM/KT

OBJECTIVES:

A) LIST THE USE AND FUNCTIONS OF THE MEMORY RELOCATE OPTION.

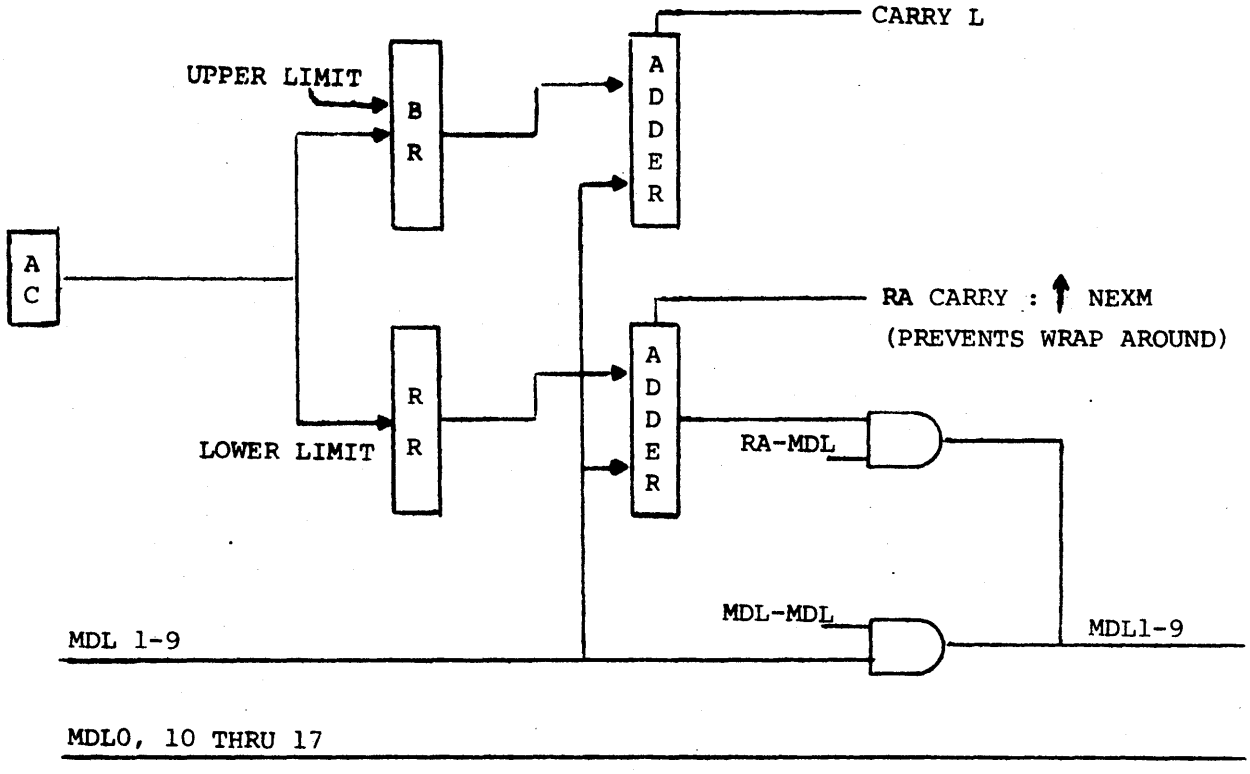
B) SHOWN THE BLOCK DIAGRAM, EXPLAIN THE COMMUNICATION SIGNAL SEQUENCE AND DATA FLOW OF THE MEMORY PROTECT OPTION.

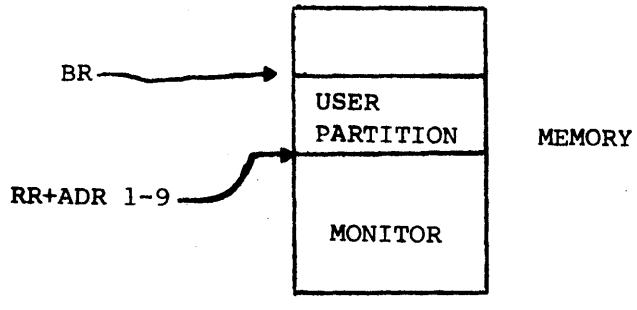
C) EXPLAIN THE FORMAT AND USE OF THE MEMORY RELOCATE OPTION.

D) LIST THE PROGRAMMING CONSIDERATIONS NECESSARY WHEN USING THE MEMORY RELOCATE OPTION.

E) EXPALIN THE KT - MEMORY RELOCATE FLOW DIAGRAM. STATING THE LOGICAL OPERATION OF A MEMORY RELOCATE TRAP CAUSED BY
   1) BOUNDARY VIOLATION
   2) ILLEGAL INSTRUCTION
   3) NON-EXISTANT MEMORY

MEMORY PROTECT AND RELOCATE BLOCK DIAGRAM





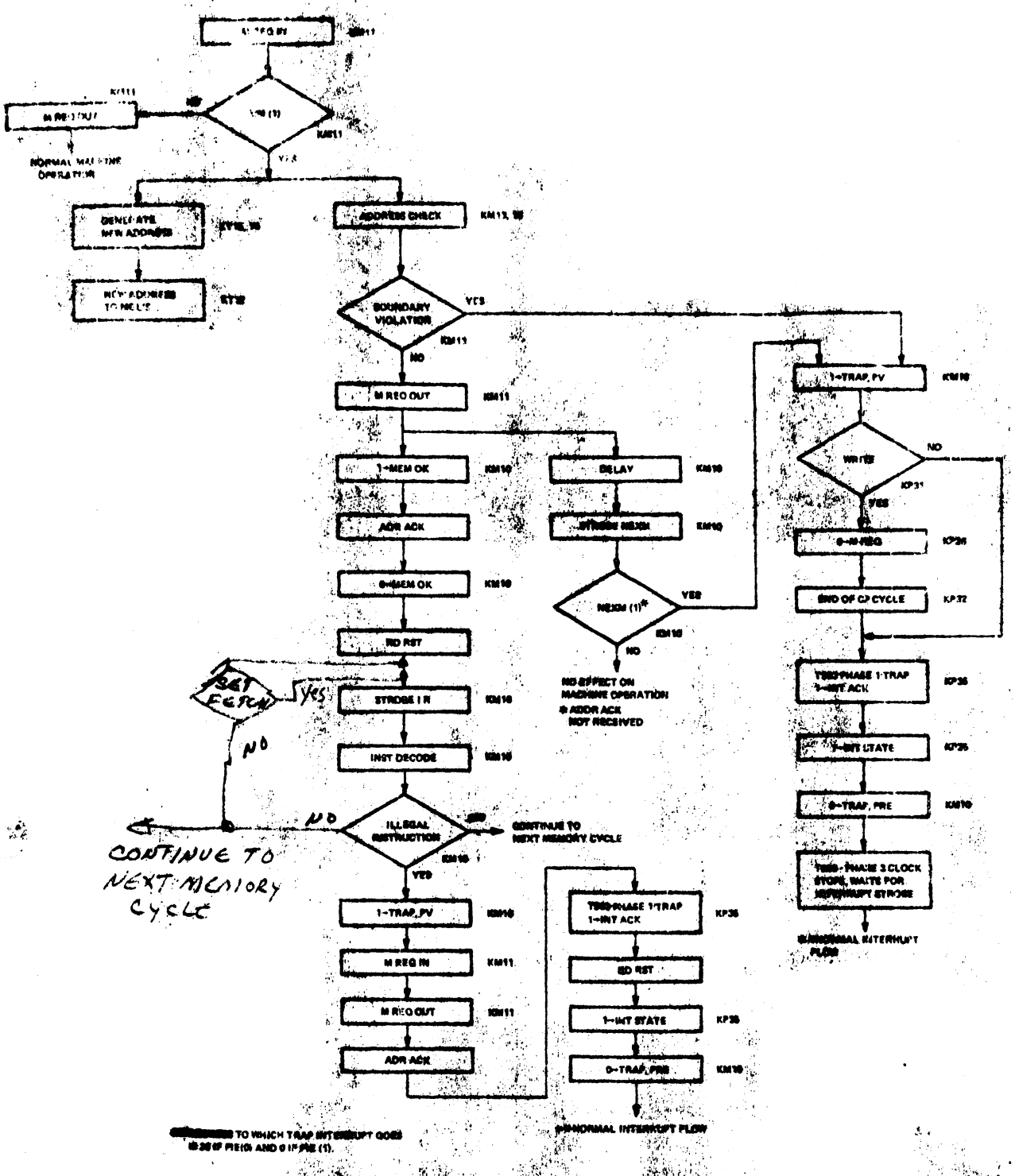


Figure 6-13 KT15 Memory Protect/Relocate Flow Diagram

TOPIC API

OBJECTIVES:

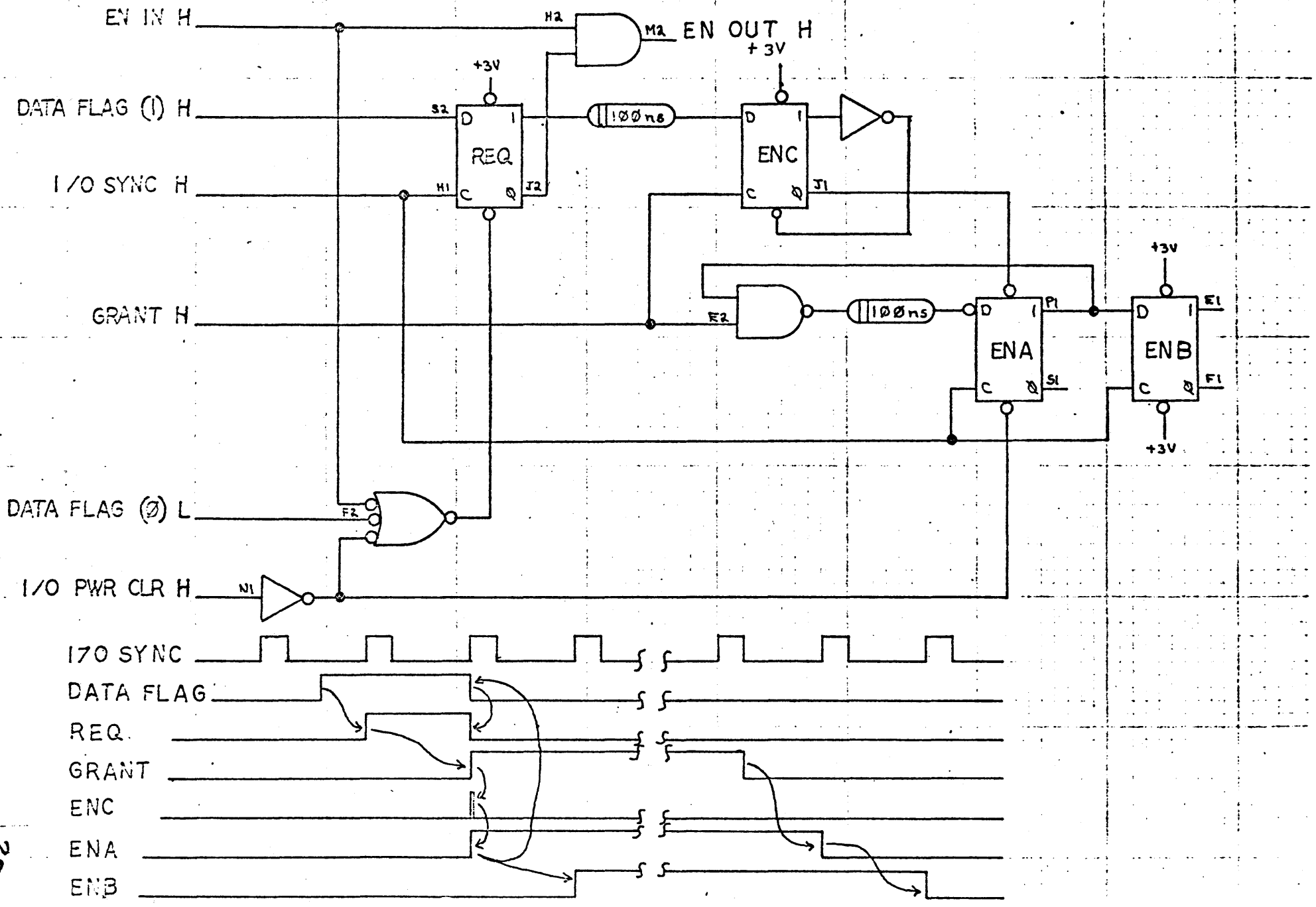
A) EXPLAIN ON A BLOCK DIAGRAM LEVEL THE FUNCTIONAL PARTS OF THE AUTOMATIC PRIORITY INTERRUPT OPTION.

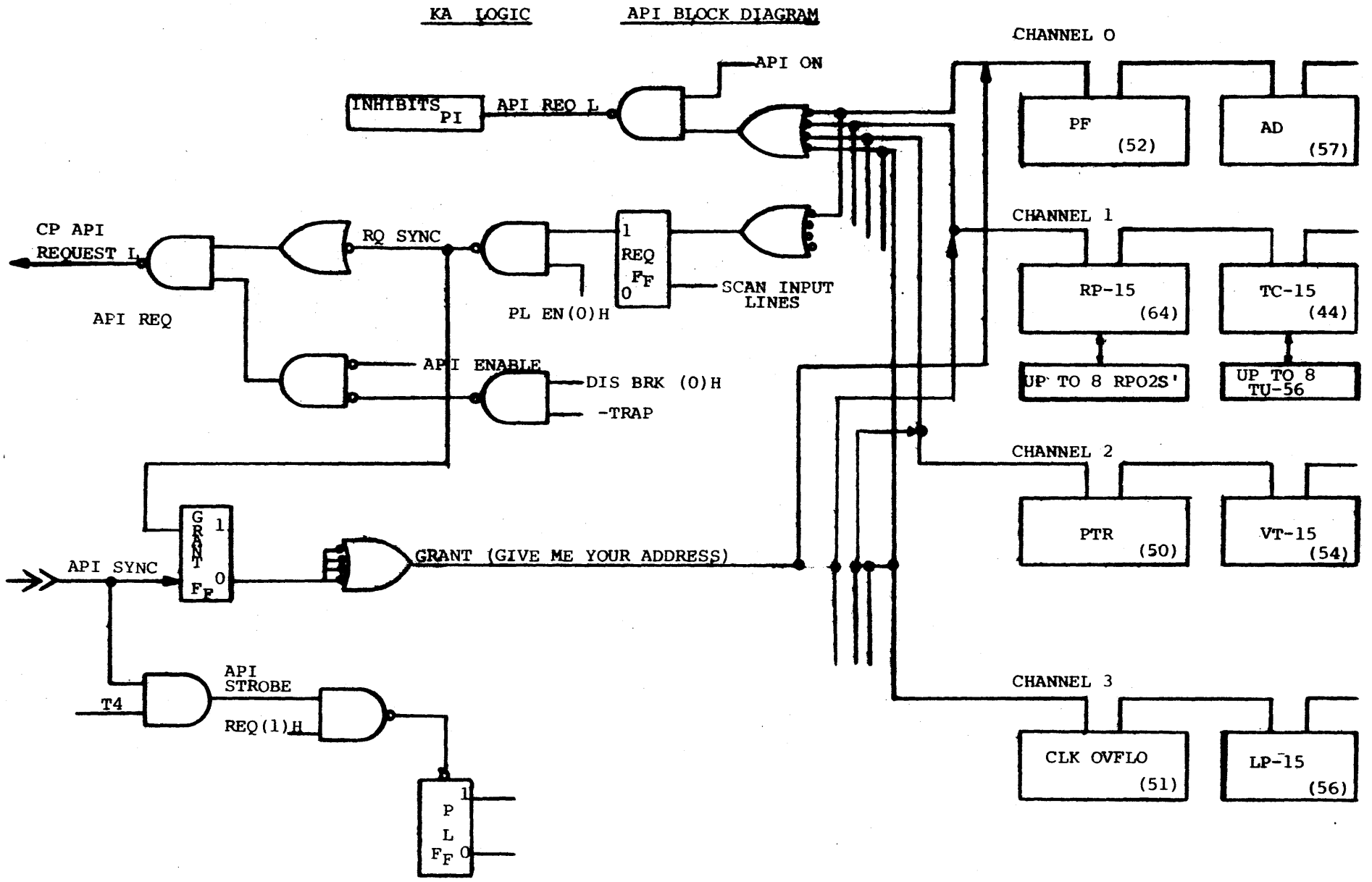
B) LIST ON A BLOCK DIAGRAM LEVEL THE COMMUNICATION SIGNAL SEQUENCE OF THE API OPTION.

C) STATE THE FUNCTIONS OF THE API IOT INSTRUCTIONS.

D) STATE THE PROGRAMMING CONSIDERATIONS NECESSARY WHEN USING API.

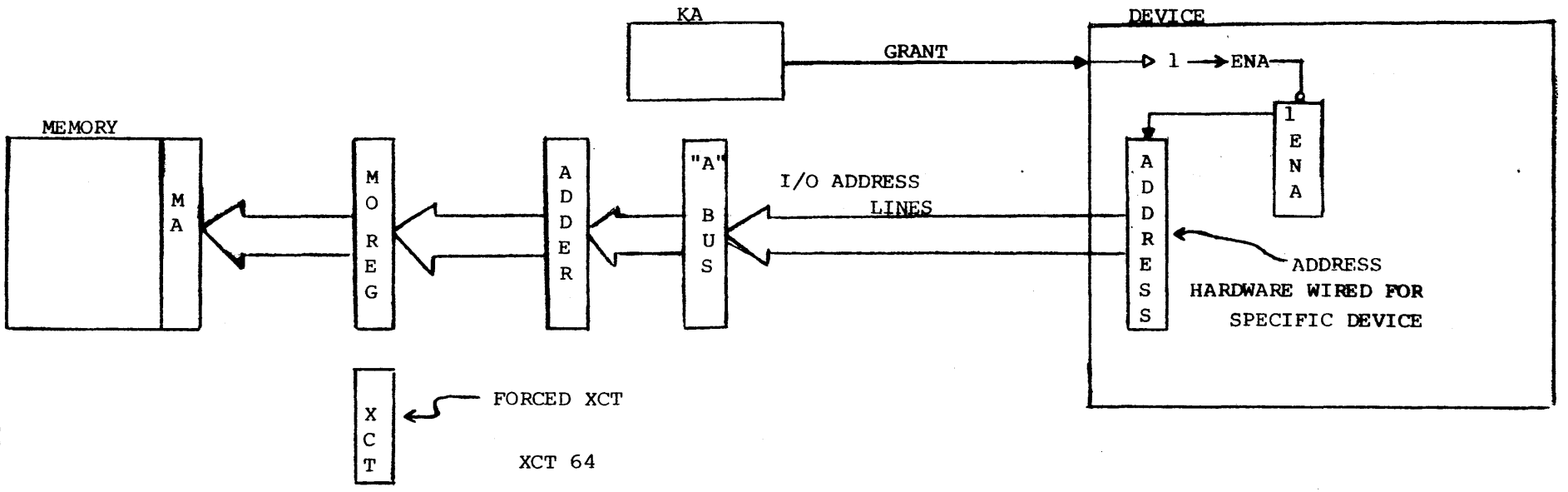
M104 SIMPLIFIED




KA LOGIC
API BLOCK DIAGRAM
CHANNEL 0
API ON
INHIBITS PI
API REQ L
PF (52)
AD (57)
CHANNEL 1
CP API REQUEST L
RQ SYNC
REQ FF
SCAN INPUT LINES
PL EN(0)H
API REQ
RP-15 (64)
TC-15 (44)
API ENABLE
DIS BRK (0)H
-TRAP
UP TO 8 RPO2S'
UP TO 8 TU-56
CHANNEL 2
PTR (50)
VT-15 (54)
GRANT FF
API SYNC
GRANT (GIVE ME YOUR ADDRESS)
CHANNEL 3
API STROBE
T4
REQ(1)H
CLK OVFLO (51)
LP-15 (56)
PLFF

API GIVE ME YOUR ADDRESS

USE M104 SLIDE IST



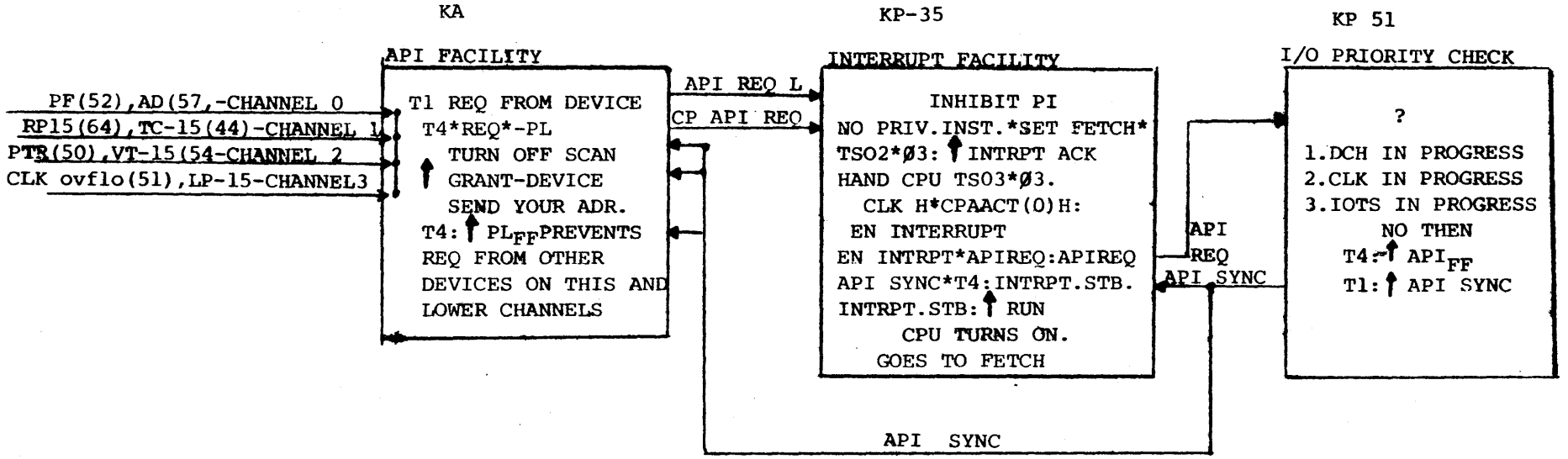
LOCATION 64 JMS TO SUBROUTINE FOR RP-15

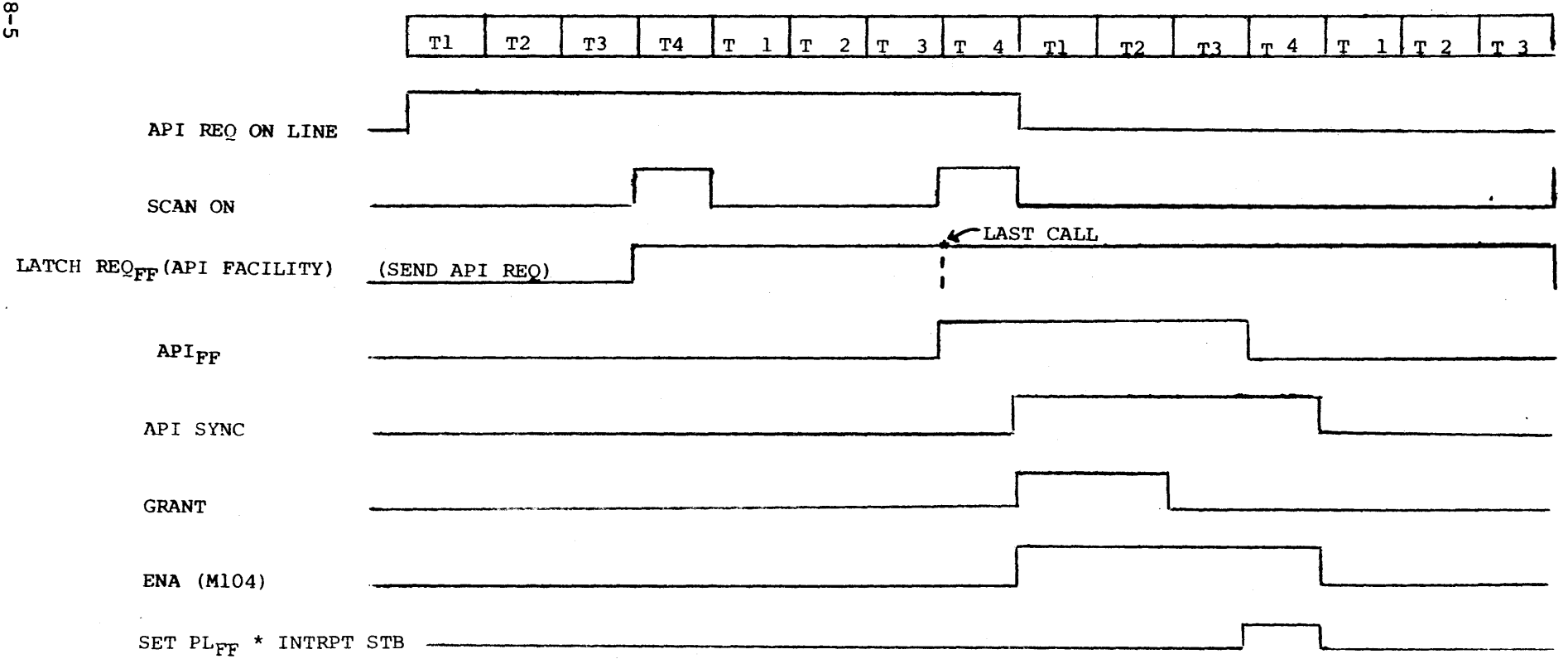
INSTRUCTION - 103000

3000 Ø L,B,U, PC 3-17

DBR

JMP * 3000


KA
API FACILITY
PF(52),AD(57,-CHANNEL 0
RP15(64),TC-15(44)-CHANNEL 1
PTR(50),VT-15(54-CHANNEL 2
CLK ovflo(51),LP-15-CHANNEL3
T1 REQ FROM DEVICE
T4*REQ*-PL
TURN OFF SCAN
GRANT-DEVICE
SEND YOUR ADR.
T4: PLFF PREVENTS
REQ FROM OTHER
DEVICES ON THIS AND
LOWER CHANNELS
API REQ L
CP API REQ
KP-35
INTERRUPT FACILITY
INHIBIT PI
NO PRIV.INST.*SET FETCH*
TSO2*Ø3: INTRPT ACK
HAND CPU TSO3*Ø3.
CLK H*CPAACT(0)H:
EN INTERRUPT
EN INTRPT*APIREQ:APIREQ
API SYNC*T4:INTRPT.STB.
INTRPT.STB: RUN
CPU TURNS ON.
GOES TO FETCH
API
REQ
API SYNC
KP 51
I/O PRIORITY CHECK
?
1.DCH IN PROGRESS
2.CLK IN PROGRESS
3.IOTS IN PROGRESS
NO THEN
T4: APIFF
T1: API SYNC
API SYNC


T1 T2 T3 T4 T 1 T 2 T 3 T 4 T1 T2 T3 T 4 T 1 T 2 T 3
API REQ ON LINE
SCAN ON
LAST CALL
LATCH REQFF (API FACILITY)
(SEND API REQ)
APIFF
API SYNC
GRANT
ENA (M104)
SET PLFF * INTRPT STB

SYSTEM POWER CONTROL

OBJECTIVES:

A) TRACE THE ROUTE OF THW PWR OK BUS THROUGH THE MAIN FRAME AND THE BACK DOOR RACK FROM THE 715 P.S.

B) EXPLAIN HOW A POWER FAIL GENERATES AN INTERRUPT.

C) EXPLAIN THE DIFFERENCE BETWEEN -POWER OK L AND POWER LOW.

D) EXPLAIN HOW CONSOLE LOCKED EFFECTS THE POWER FAIL SEQUENCE.

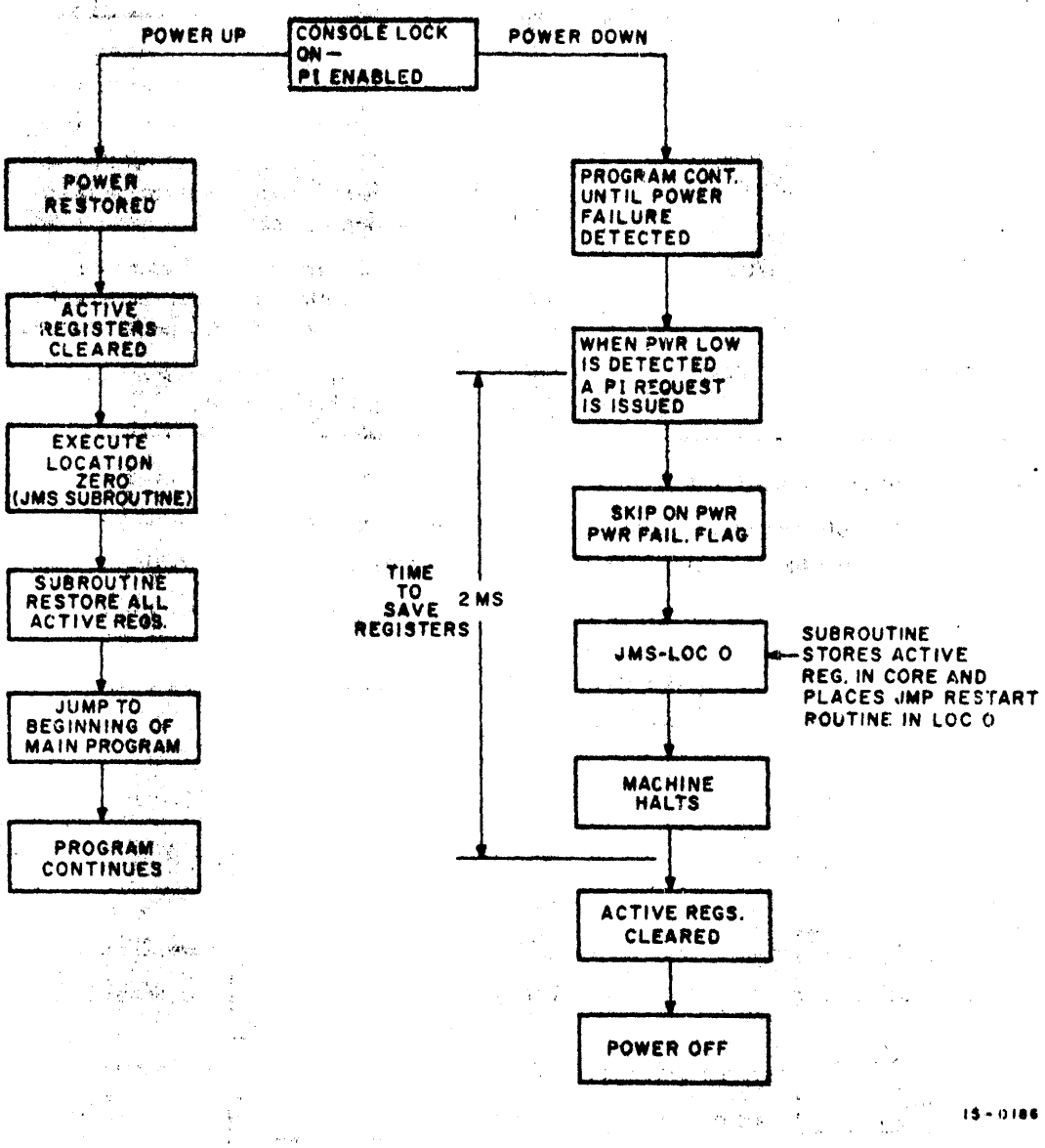


Figure 6-8 Power Fail Up/Down Sequence

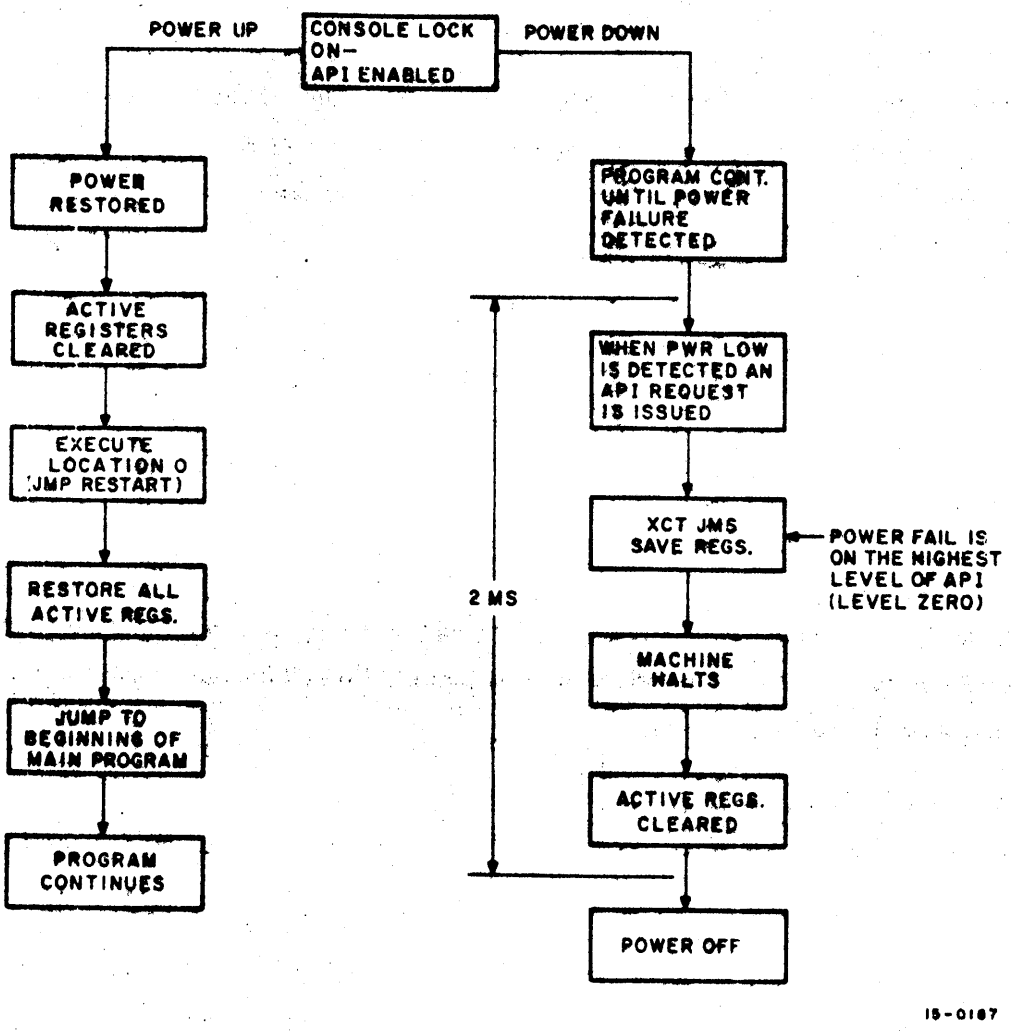
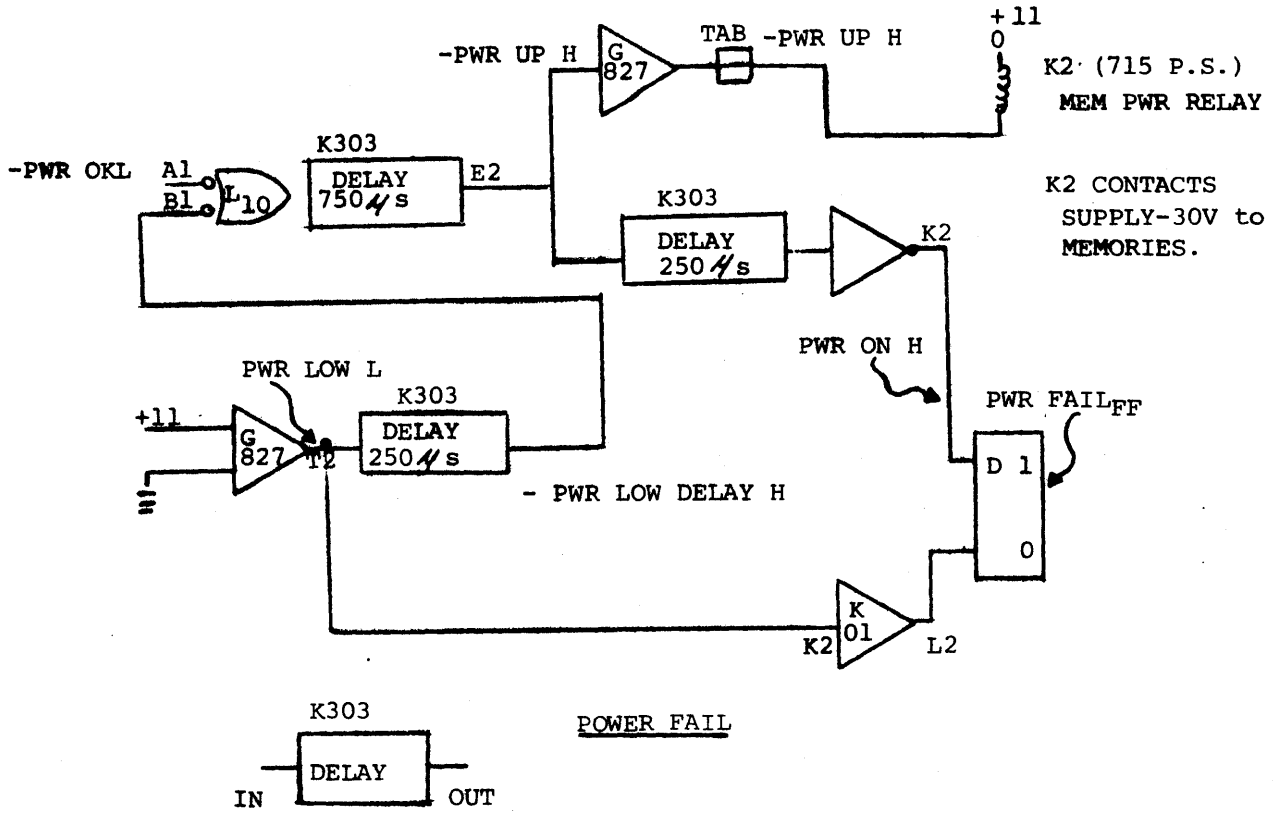


Figure 6-9 Power Fail Up/Down Sequence



POWER FAIL

H IN: H OUT IMMEDIATELLY NO DELAY
L IN: STAYS HIGH UNTIL THE DELAY TIMES OUT THEN OUTPUT GOES LOW

FAILURES IN THE REGULATORS, G821-G822-G823, DROP THE MEMORY PWR RELAY IMMEDIATELY TO PROTECT THE MEMORY- "CAN'T SAVE ANY REG." THE FAILURES IN THE REGULATORS ARE DETECTED BY THE SIGNAL - PWR OK L.NOTE PWR $FAIL_{FF}$ IS NOT SET.

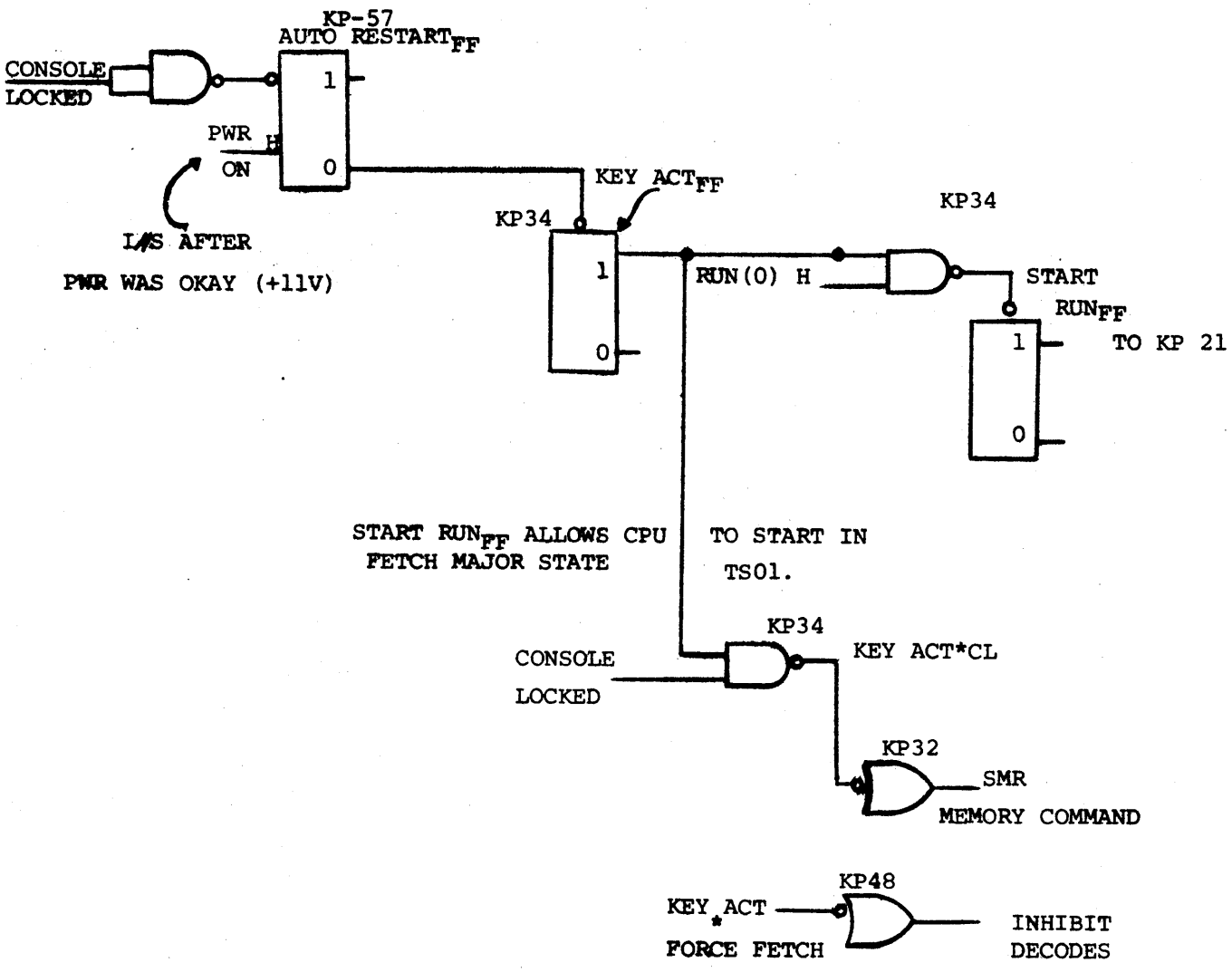
G827 SAMPLES +11V. THIS IS THE VOLTAGE OFF THE CAPACITORS IN 715 PS. IF THE +11 VOLTS DROPS FOR ANY REASON, THE G827 GENERATES A PWR LOW SIGNAL WHICH IMMEDIATELY SETS THE PWR $FAIL_{FF}$.

SETS UP FOR INTERRUPT. TIME BEFORE THE MEMORY PWR RELAY DROPS IS 250μs THEREFORE IT CAN BE SEEN IN THIS TIME CAN SAVE A LOT OF REGISTERS.

IF CONSOLE LOCKED IS NOT ON PROCESS POWER DOWN AS IN CONSOLE LOCKED POWER DOWN FLOW. HAVE TO USE MANUAL RESTART SEQUENCE.

POWER FAIL - AUTOMATIC RESTART

ON POWER DOWN THE LAST THING THAT HAPPENED WAS A JMS TO START UP ROUTINE WAS ENTERED INTO LOCATION ZERO.



NOTHING IS ENABLED ON THE "A" BUS AND THE "B" BUS: SUM BUS = ZERO's

KP24

KEY ACTIVE*TS01*CLK H: LD MO, OA THE ADDRESS OF ZERO IS LOADED INTO THE MO AND OA REGISTER.

MEMORY CYCLE READS INTO THE CPU THE INSTRUCTION LOCATED AT LOCATION ZERO (JMS TO START UP ROUTINE)

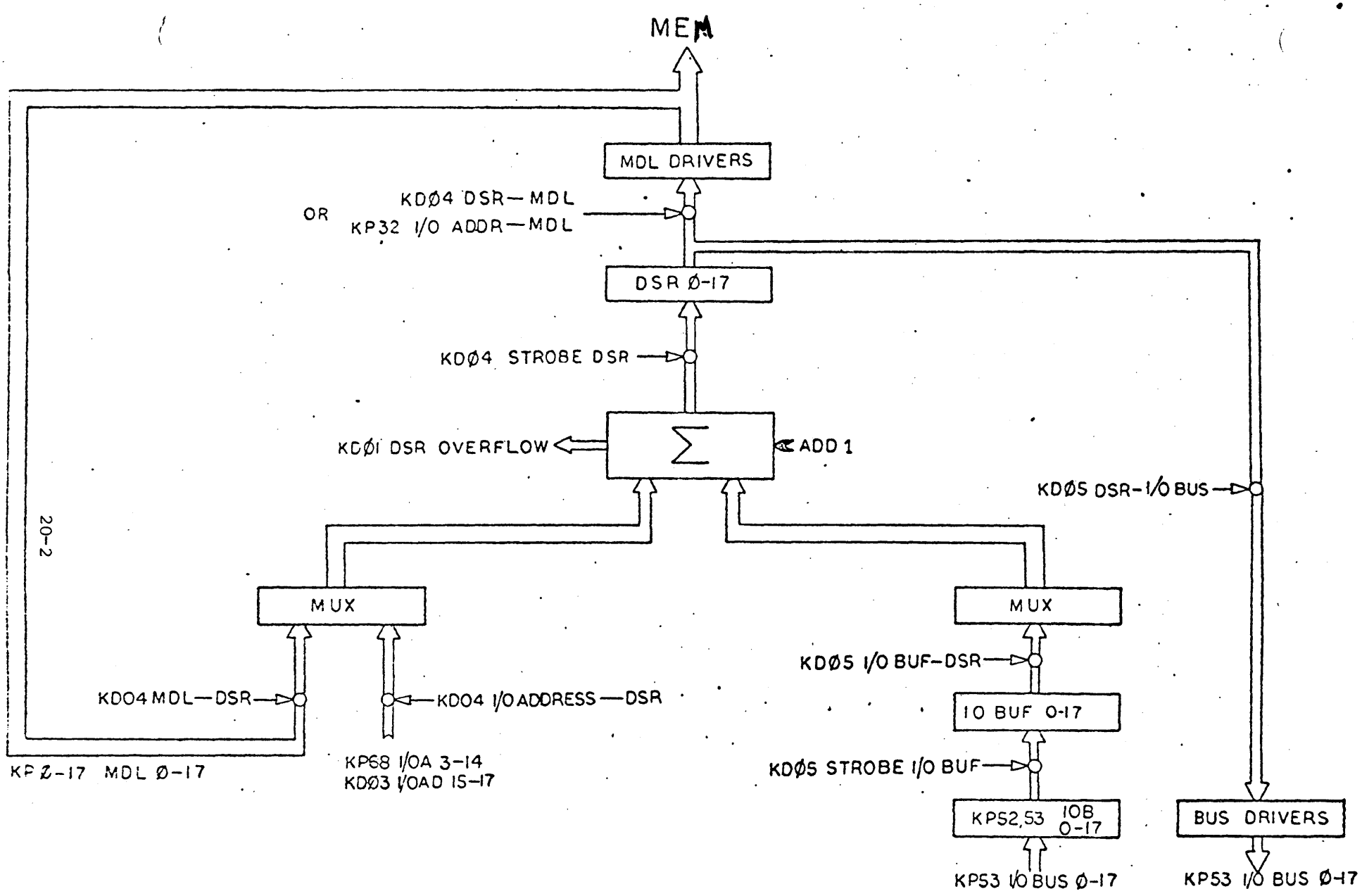
REAL TIME CLOCK

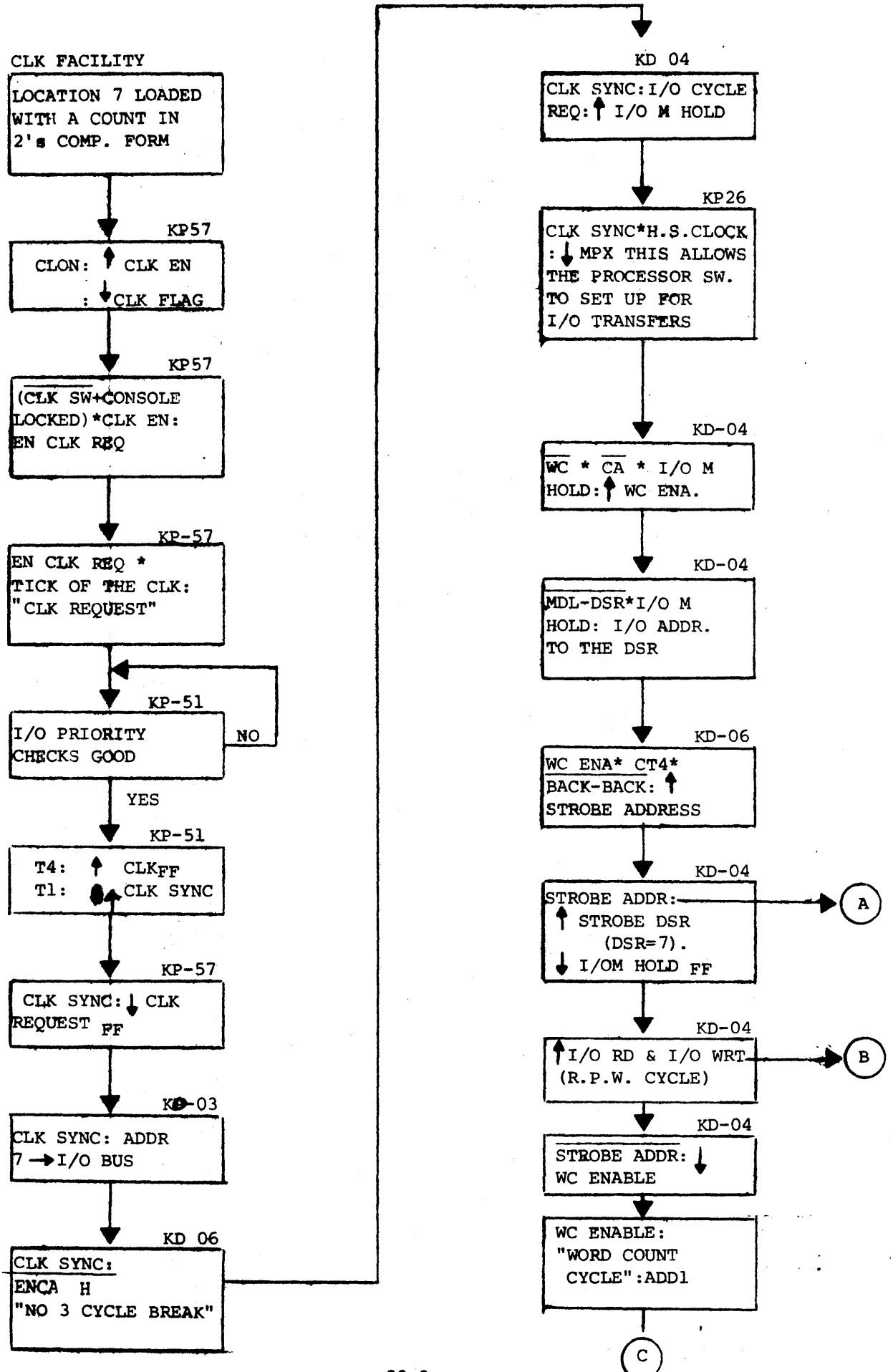
OBJECTIVES:

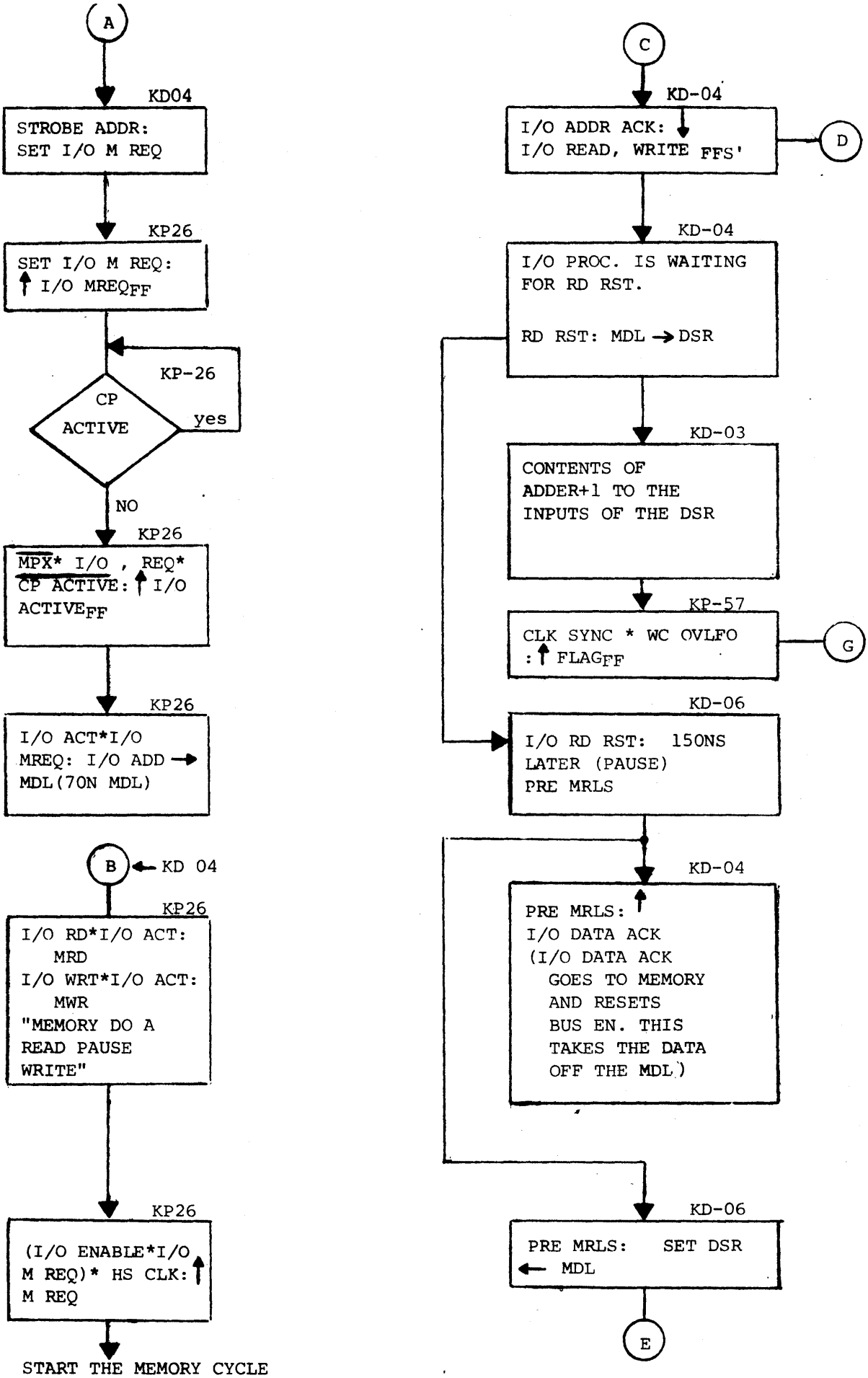
A) WRITE A PROGRAM UTILIZING THE REAL TIME CLOCK.

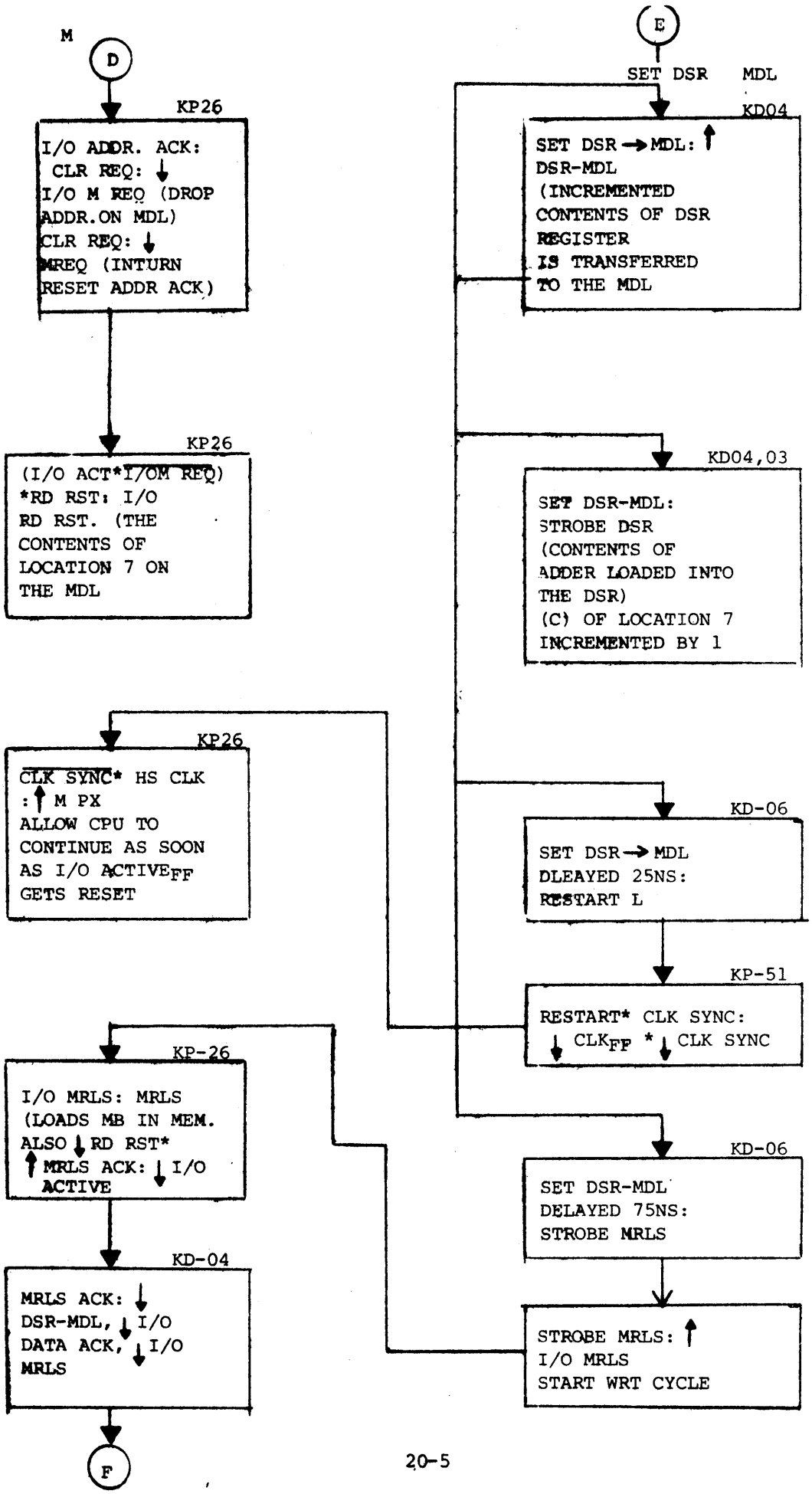
B) STATE HOW THE REAL TIME CLOCK FACILITY IS SET UP.

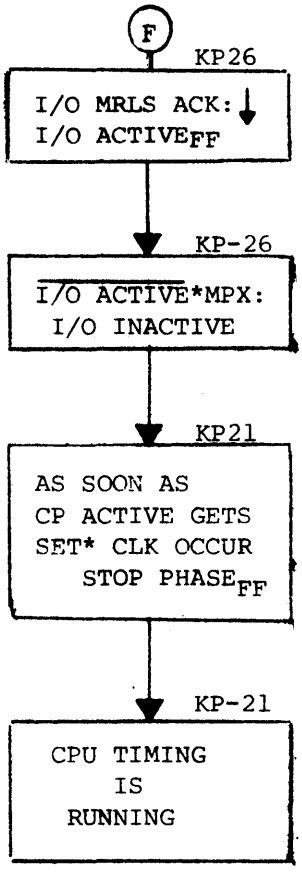
C) LIST THE SEQUENCE OF EVENTS THAT CAUSE A CLK PI.

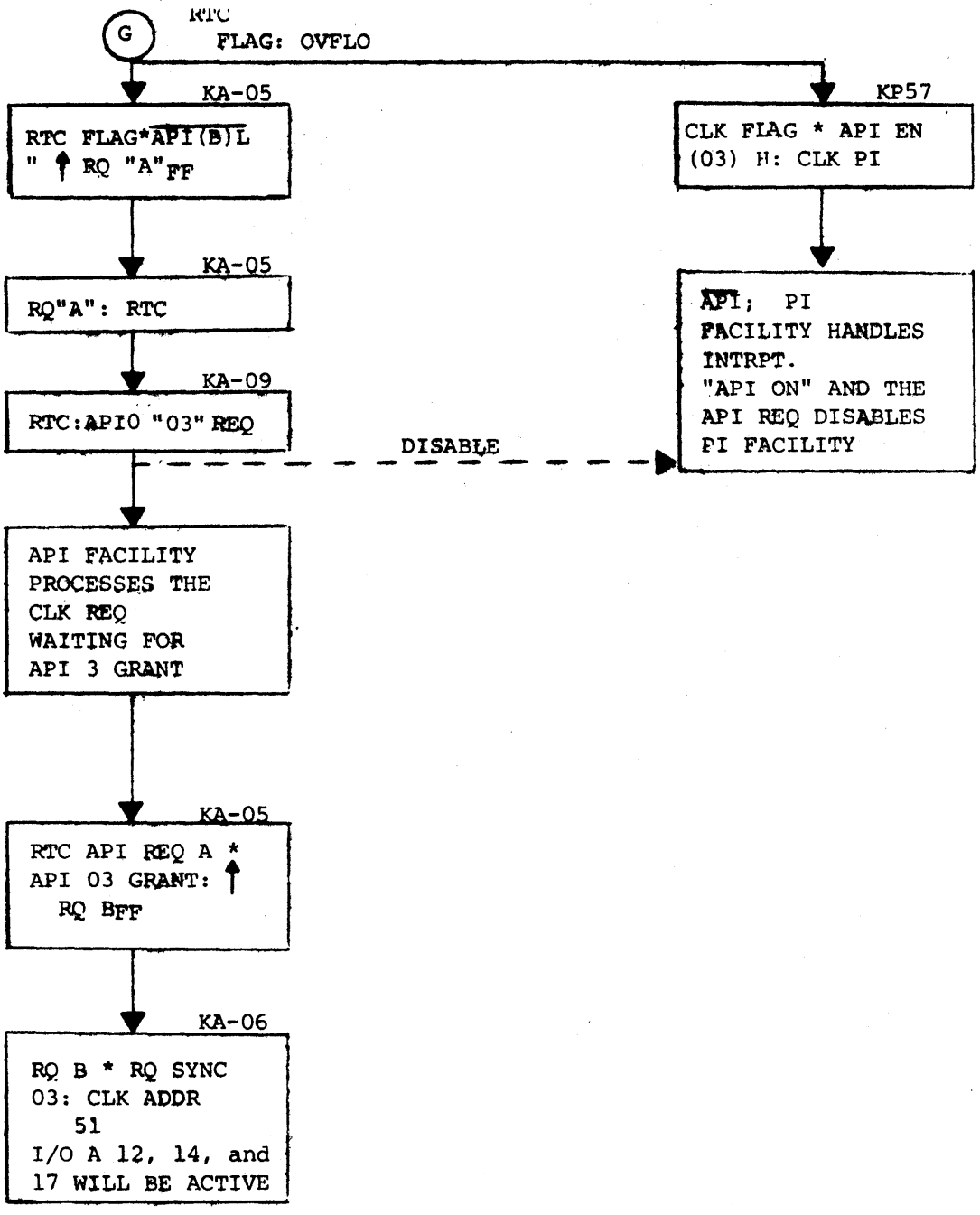

MEM
MDL DRIVERS
KDØ4 DSR—MDL
OR
KP32 I/O ADDR—MDL
DSR Ø-17
KDØ4 STROBE DSR
KDØI DSR OVERFLOW
Σ
ADD 1
KDØ5 DSR-I/O BUS
20-2
MUX
MUX
KDØ5 I/O BUF-DSR
KDO4 MDL—DSR
KDO4 I/O ADDRESS—DSR
IO BUF 0-17
KDØ5 STROBE I/O BUF
KP68 I/OA 3-14
KDØ3 I/OAD IS-17
KP Ø-17 MDL Ø-17
KPS2,53 IOB 0-17
BUS DRIVERS
KP53 I/O BUS Ø-17
KP53 I/O BUS Ø-17


CLK FACILITY
LOCATION 7 LOADED WITH A COUNT IN 2's COMP. FORM
KP57
CLON: ↑ CLK EN
: ↓ CLK FLAG
KP57
(CLK SW+CONSOLE LOCKED)*CLK EN: EN CLK REQ
KP-57
EN CLK REQ * TICK OF THE CLK: "CLK REQUEST"
KP-51
I/O PRIORITY CHECKS GOOD
NO
YES
KP-51
T4: ↑ CLKFF
T1: CLK SYNC
KP-57
CLK SYNC: ↓ CLK REQUEST FF
KD-03
CLK SYNC: ADDR 7 → I/O BUS
KD 06
CLK SYNC:
ENCA H
"NO 3 CYCLE BREAK"
KD 04
CLK SYNC: I/O CYCLE REQ: ↑ I/O M HOLD
KP26
CLK SYNC*H.S.CLOCK : ↓ MPX THIS ALLOWS THE PROCESSOR SW. TO SET UP FOR I/O TRANSFERS
KD-04
WC * CA * I/O M HOLD: ↑ WC ENA.
KD-04
MDL-DSR*I/O M HOLD: I/O ADDR. TO THE DSR
KD-06
WC ENA* CT4* BACK-BACK: ↑ STROBE ADDRESS
KD-04
STROBE ADDR: ↑ STROBE DSR (DSR=7). ↓ I/OM HOLD FF
A
KD-04
↑ I/O RD & I/O WRT (R.P.W. CYCLE)
B
KD-04
STROBE ADDR: ↓ WC ENABLE
WC ENABLE: "WORD COUNT CYCLE":ADD1
C


A
KD04
STROBE ADDR:
SET I/O M REQ
KP26
SET I/O M REQ:
↑ I/O MREQFF
KP-26
CP
ACTIVE
yes
NO
KP26
MPX* I/O , REQ*
CP ACTIVE: ↑ I/O
ACTIVEFF
KP26
I/O ACT*I/O
MREQ: I/O ADD →
MDL(70N MDL)
B
← KD 04
KP26
I/O RD*I/O ACT:
MRD
I/O WRT*I/O ACT:
MWR
"MEMORY DO A
READ PAUSE
WRITE"
KP26
(I/O ENABLE*I/O
M REQ)* HS CLK: ↑
M REQ
START THE MEMORY CYCLE
C
KD-04
I/O ADDR ACK: ↓
I/O READ, WRITE FFS'
D
KD-04
I/O PROC. IS WAITING
FOR RD RST.
RD RST: MDL → DSR
KD-03
CONTENTS OF
ADDER+1 TO THE
INPUTS OF THE DSR
KP-57
CLK SYNC * WC OVLFO
: ↑ FLAGFF
G
KD-06
I/O RD RST: 150NS
LATER (PAUSE)
PRE MRLS
KD-04
PRE MRLS: ↑
I/O DATA ACK
(I/O DATA ACK
GOES TO MEMORY
AND RESETS
BUS EN. THIS
TAKES THE DATA
OFF THE MDL)
KD-06
PRE MRLS: SET DSR
← MDL
E


M
D
KP26
I/O ADDR. ACK:
CLR REQ: ↓
I/O M REQ (DROP
ADDR.ON MDL)
CLR REQ: ↓
MREQ (INTURN
RESET ADDR ACK)
KP26
(I/O ACT*I/OM REQ)
*RD RST: I/O
RD RST. (THE
CONTENTS OF
LOCATION 7 ON
THE MDL
KP26
CLK SYNC* HS CLK
: ↑ M PX
ALLOW CPU TO
CONTINUE AS SOON
AS I/O ACTIVE FF
GETS RESET
KP-26
I/O MRLS: MRLS
(LOADS MB IN MEM.
ALSO ↓ RD RST*
↑ MRLS ACK: ↓ I/O
ACTIVE
KD-04
MRLS ACK: ↓
DSR-MDL, ↓ I/O
DATA ACK, ↓ I/O
MRLS
F
E
SET DSR MDL
KD04
SET DSR → MDL: ↑
DSR-MDL
(INCREMENTED
CONTENTS OF DSR
REGISTER
IS TRANSFERRED
TO THE MDL
KD04,03
SET DSR-MDL:
STROBE DSR
(CONTENTS OF
ADDER LOADED INTO
THE DSR)
(C) OF LOCATION 7
INCREMENTED BY 1
KD-06
SET DSR → MDL
DLEAYED 25NS:
RESTART L
KP-51
RESTART* CLK SYNC:
↓ CLK FF * ↓ CLK SYNC
KD-06
SET DSR-MDL
DELAYED 75NS:
STROBE MRLS
STROBE MRLS: ↑
I/O MRLS
START WRT CYCLE


F
KP26
I/O MRLS ACK: ↓
I/O ACTIVE$_{FF}$
KP-26
$\overline{\text{I/O ACTIVE}}$*MPX:
I/O INACTIVE
KP21
AS SOON AS
CP ACTIVE GETS
SET* CLK OCCUR
STOP PHASE$_{FF}$
KP-21
CPU TIMING
IS
RUNNING

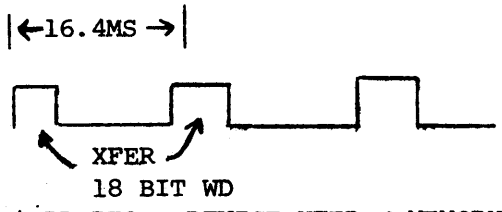


NOTE: CLK FLAG WILL GET CLEARED BY CLON + CLOF

RF 15- 3 CYCLE BREAK DEVICE
WC ADDRESS 36 (C) 36= TWO's COMP OF NUMBER OF WDS XFER
CA 37 (C) 37= LOAD INTO LOCATION 37 STARTING ADDRESS MINUS ONE.

API BREAK ADDRESS - 63
SUGGESTED PRIORITY LEVEL 1

SECTOR CONTAINS, 256,-18 BIT WDS

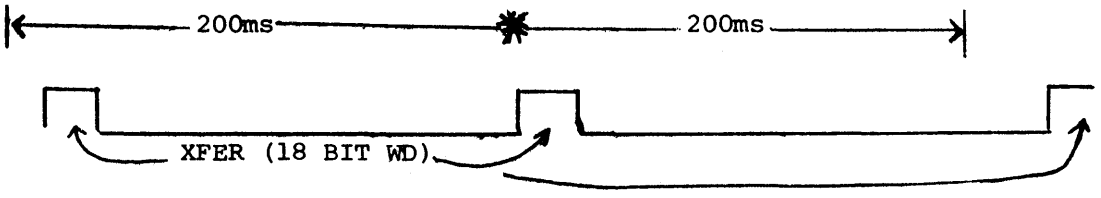


DCH REQ * RD REQ : DEVICE XFER → MEMORY 10P2
DCH REQ * WRT REQ: MEMORY XFER → DEVICE 10P4

1. SET UP
2. DCH XFERS / 1 WORD PER DCH REQUEST
3. JOB DONE OR ERROR - API OR PI

TC 15 3 CYCLE BREAK DEVICE
WC LOCATION 30 (C) 30 = 2's COMP. OF NO OF WDS XFER
CA LOCATION 31 (C) 31 = LOAD INTO LOCATION 31
STARTING ADDRESS MINUS ONE

API BREAK ADDRESS 44
SUGGESTED PRIORITY LEVEL 1



1. SET UP
2. DCH XFERS / 1 WORD PER DCH REQUEST
3. JOB DONE OR ERROR - API OR PI

DCH REQ * RD REQ : DEVICE XFER TO MEMORY 10P2
DCH REQ * WRITE REQ: MEMORY XFER TO DEVICE 10P4

RP-15

SET UP INSERT AND EXECUTE

1. INSERT 2's COMPL. OF NO OF WORDS
   LOAD WORD COUNT REGISTER (18 BITS)

2. INSERT DISK ADDRESS
   a) CYLINDER ADDRESS REGISTER (7 BITS)
   b) HEAD ADDRESS REGISTER (5 BITS)
   c) SECTOR ADDRESS REGISTER (4 BITS)

3. INSERT INITIAL MEMORY ADDRESS LOAD CURRENT ADDRESS
   REGISTER (17 BITS)

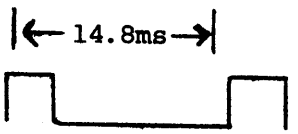
4. INSERT FUNCTION & GO
   LOAD STATUS REGISTER "A" (9 BITS)

RP15 CONTROLS UP TO 8 RP02 OR RP03 - DISK PACK
WD SIZE - 36 BITS/WD
SECTOR CONTAINS 128, 36BIT WDS OR 256, 18 BIT WDS

HAS A WC REGISTER } LOADED DURING SETUP
HAS A CA REGISTER }

USES SINGLE CYCLE XFERS FOR ODD NUMBER OF XFERS

BACK TO BACK XFERS FOR EVEN NUMBER OF XFERS

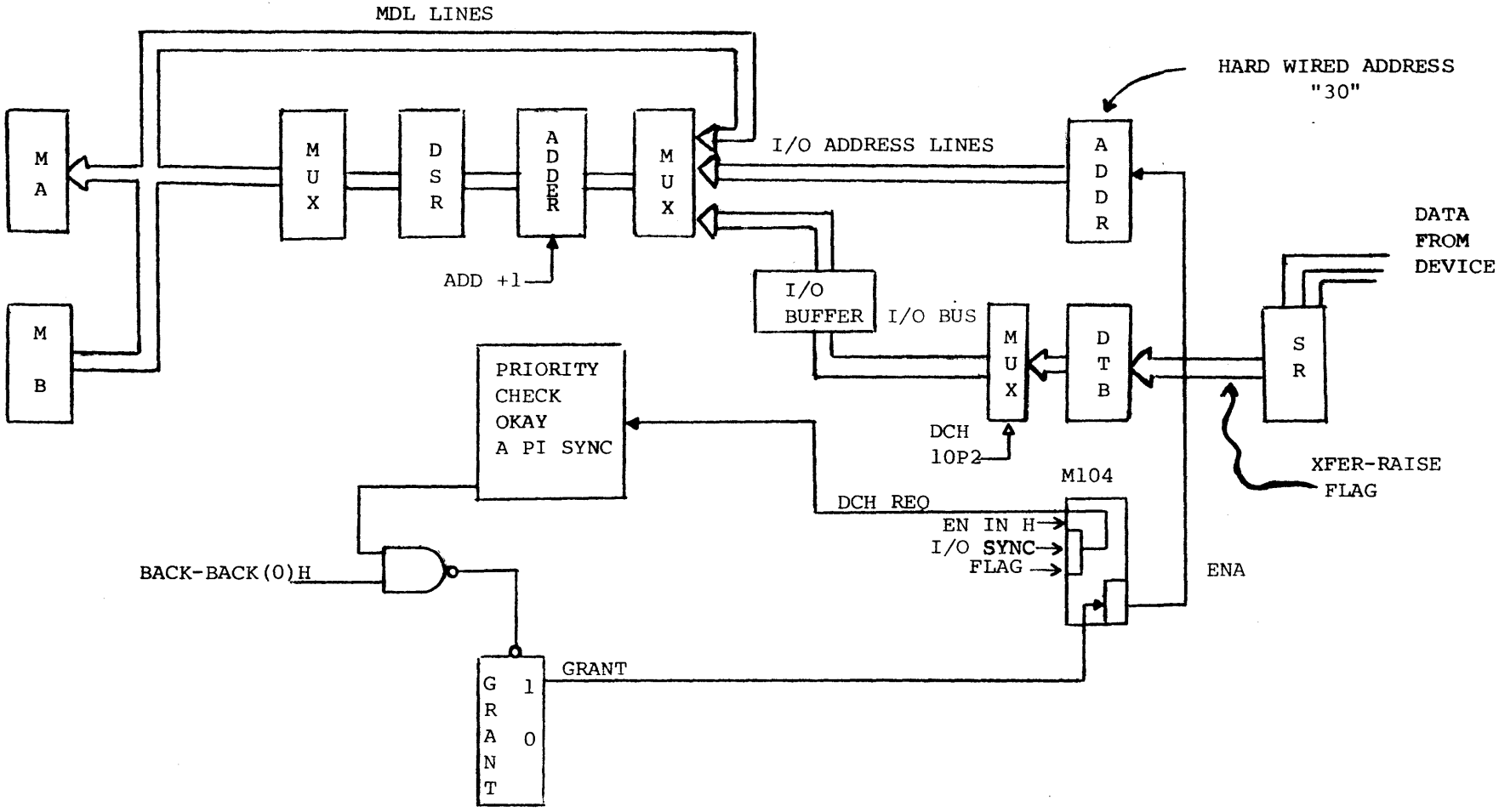


API BREAK ADDRESS 64
API LEVEL - SUGGESTED PRIORITY LEVEL 1

1. SET UP
2. DCH XFERS/ 2 WDS - (BACK→ BACK)
+
1 WD (SINGLE CYCLE)

DCH REQ L	SING CYL REQ L	FUNCTION
0	1	SINGLE CYCLE XFER OUT
1	1	SINGLE CYCLE XFER IN

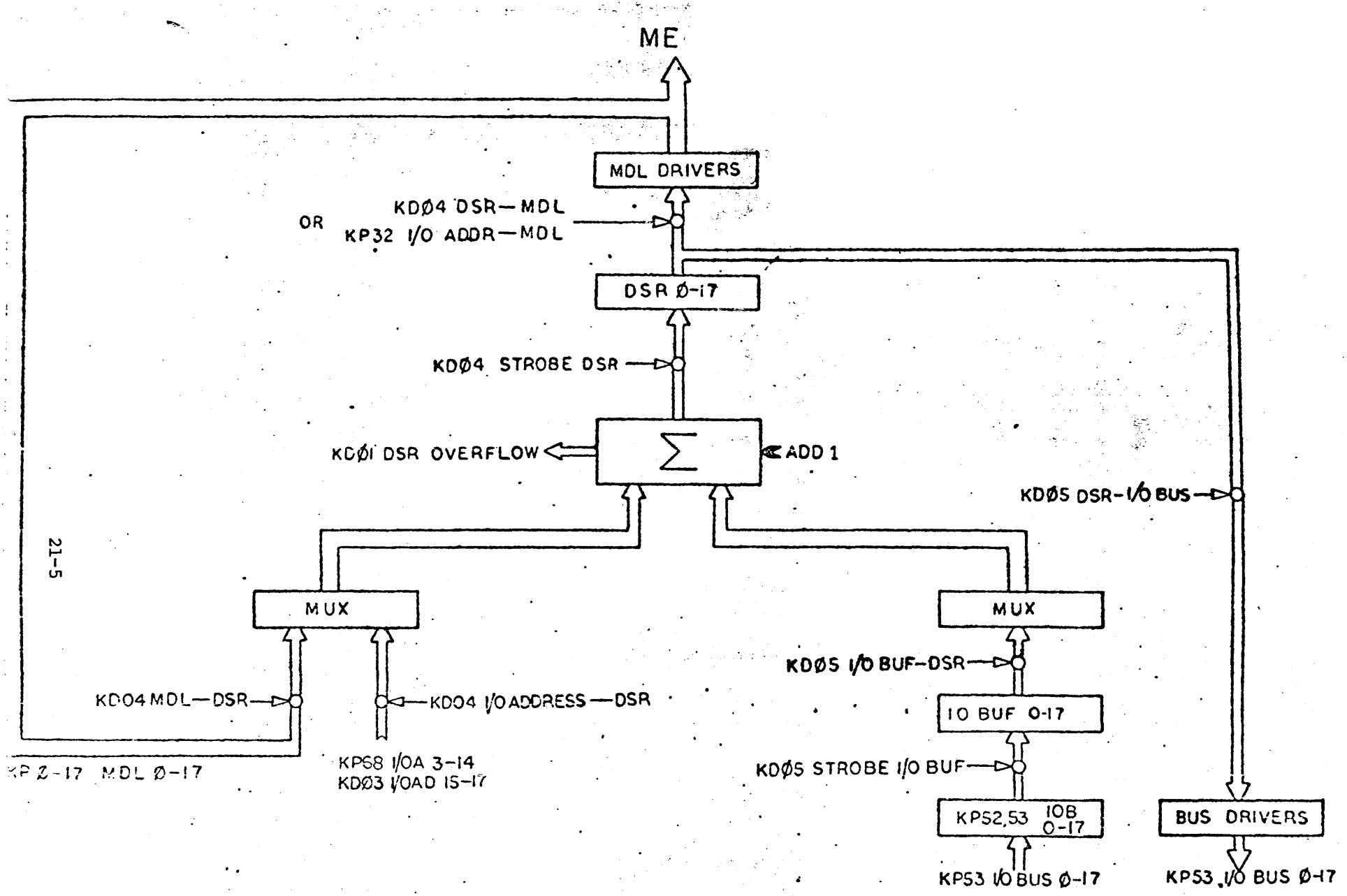
IF SING CYL REQ L IS HELD ACTIVE AFTER $GRANT_{FF}$ IS RESET, THIS IMPLIES ANOTHER XFER IS COMING (BACK→BACK)



BLOCK DIAGRAM OF DATA IN TRANSFER

3 CYCLE BREAK FROM THE DECK TAPE
DATA IN XFER.

1. WORD COUNT CYCLE (WC)
2. CURRENT ADDRESS CYCLE (CA)
3. DATA CYCLE


ME
MDL DRIVERS
OR
KDØ4 DSR—MDL
KP32 I/O ADDR—MDL
DSR Ø-17
KDØ4 STROBE DSR
KDØI DSR OVERFLOW
Σ
ADD 1
KDØ5 DSR-I/O BUS
21-5
MUX
MUX
KDØ5 I/O BUF-DSR
KDO4 MDL—DSR
KDO4 I/O ADDRESS—DSR
IO BUF 0-17
KDØ5 STROBE I/O BUF
KP 2-17 MDL Ø-17
KP58 I/OA 3-14
KDØ3 I/OAD 15-17
KPS2,53 IOB 0-17
BUS DRIVERS
KP53 I/O BUS Ø-17
KP53 I/O BUS Ø-17

3 CYCLE BREAK

EXAMPLE:

DATA IN XFER FROM TC-15

1. WORD COUNT CYCLE
   READ PAUSE WRITE

NEEDS A DCH REQUEST FROM THE DEVICE TO TRANSFER THE DATA INTO MEMORY.

NEEDS ADDRESS FROM DEVICE 30 FROM TC-15

NEEDS COMMAND
   I/O PROCESSOR GENERATES I/O READ*I/O WRT
   THIS SETS MEMORY CONTROL TO SET UP FOR A READ-PAUSE WRITE

NEEDS MREQ
   THIS LOADS THE ADDRESS IN MEMORY AND STARTS THE MEMORY CYCLE.

NEEDS STOP THE CPU WHILE PROCESSING THE DEVICES DATA XFER.

   CPU IS HUNG UP BY STOP PHASES
   CPU DECODES INSTRUCTION IT IS GOING TO PROCESS AND
      SMW: CPU HANGS IN TS02*Ø2
      SMR: CPU HANGS IN TS03*Ø2

WORD COUNT MEMORY CYCLE CONTENTS OF LOCATION 30 INCREMENTED BY 1 WRITTEN BACK INTO LOCATION 30.

(C) LOCATION 30 = 777523 = THE TWO's COMPLEMENT OF 256 WDS. ONE BLOCK OF DATA TO BE XFERED.

777523 IS ON THE MDL
RD RST: MDL → DSR ADD+1 IS ALSO ACTIVE OUTPUT OF THE ADDER IS 777524

RD RST: DELAY 150NS (PAUSE) AND GENERATE PRE MRLS AND SET DSR-MDL

PRE MRLS: ↑ I/O DATA ACK: ↑ DATA ACK: ↓ BUS EN
BUS EN RESET TAKES 777523 OFF THE MDL.

SET DSR-MDL: ↑ DSR-MDL ALLOWS (C) OF DSR → MDL
:STROBE DSR 777524 IS CLOCKED INTO THE DSR AND XFERED ONTO THE MDL.
(SET DSR-MDL* WC)*EN CA: ↑ CA
STROBE DSR: CHECK THE ADDER FOR OVERFLOW. IF OVERFLOW IS PRESENT THIS IMPLIES 256 WORDS HAVE BEEN TRANSFERED. SEND OVFLO BACK TO DEVICE WHICH TERMINATES BY RAISING A JOB DONE FLAG.

SET DSR-MDL DELAYED 75NS : STROBE MRLS

STROBE MRLS: ↑ I/O MRLS : ↑ MRLS :MB LOAD
MB = 777524
I/O MRLS: ↓ WC
DELAY OF 50NS: MRLS ACK: ↓ RD RST AND START THE WRITE TIMING IN MEMORY

MRLS ACK : ↓ I/O DATA ACK, ↓ I/O MRLS, ↓ DSR-MDL, ↓ I/O ACTIVE

$\overline{\text{I/O MRLS}}$: ↓ MRLS ACK

MEMORY WRITE CYCLE FINISHES & DROPS MBSY

CURRENT ADDRESS CYCLE (CA)

THE STARTING LOCATION TO STORE THE DATA WORD WILL BE LOCATION 500. THEREFORE, THE PROGRAMMER PLACES IN LOCATION 31,000477.

STARTING ADDRESS MINUS ONE

2. CA CYCLE
READ PAUSE WRITE

NEEDS ADDRESS 30 FROM DEVICE TO BE STILL ON THE LINE.

NEEDS ADD+1 ACTIVE WHICH IT STILL IS

NEEDS COMMAND
I/O PROCESSOR GENERATES I/O READ*I/O WRITE
THIS SETS MEMORY CONTROL TO SET UP FOR A READ, PAUSE, WRITE.

NEEDS MREQ
THIS LOADS THE ADDRESS IN MEMORY AND STARTS THE MEMORY CYCLE.

NEEDS - CPU STILL HUNG UP BY STOP PHASES

CURRENT ADDRESS CYCLE

CONTENTS OF LOCATION 31 INCREMENTED BY 1
WRITTEN BACK INTO LOCATION 31
000477+1 = 000500 NOW UPDATED CONTENTS OF LOCATION 31 ALSO 000500 IS STILL STORED IN THE DSR. DSR CONTAINS THE ADDRESS WHERE THE DATA WILL BE STORED.

(C) LOCATION 31 = 000477

000477 IS ON THE MDL

RD RST: MDL-DSR ADD+1 IS ACTIVE OUTPUT OF THE ADDER IS 000500

RD RST: DELAY 150 NS (PAUSE) AND GENERATE PREMRLS AND SET DSR-MDL

PRE MRLS: ↑ I/O DATA ACK, : ↑ DATA ACK: ↓ BUS EN
BUS EN RESET TAKES 000477 OFF THE MDL LINES

SET DSR-MDL: ↑ DSR-MDL THIS ALLOWS (C) OF DSR-MDL
SET DSR-MDL: STROBE DSR 000500 IS CLOCKED INTO THE DSR AND XFERED ONTO THE MDL

SET DSR-MDL DELAYED 75 NS: STROBE MRLS

STROBE MRLS: ↑ I/O MRLS, ↑ MRLS : MB LOAD
MB = 000500

IOMRLS*DATA CYCLE: ↓ CA

$\overline{WC}*\overline{CA}$: $\overline{ADD+1}$

MRLS DELAYED 50 NS: ↑ MRLS ACK : ↓ RD RST AND START THE WRITE TIMING IN MEMORY.

MRLS ACK: ↓ I/O DATA ACK, ↓ I/O MRLS, ↓ DSR-MDL, ↓ I/O ACTIVE

$\overline{I/O\ MRLS}$: ↓ MRLS ACK

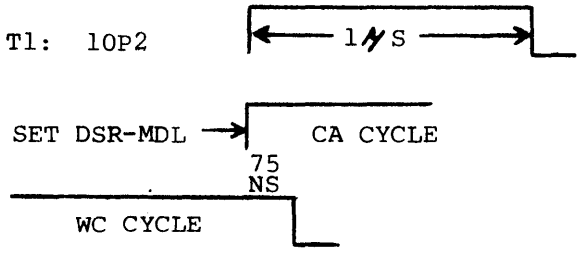
MEMORY WRITE CYCLE FINISHES AND DROPS MBSY

DATA CYCLE:

SETTING UP FOR THE DATA CYCLE STARTS DURING THE WC CYCLE. THE TIME AS FAR AS A MEMORY IS CONCERNED IS ABOUT MREQ TIME (TAIL END). MORE SPECIFIC TIME IS THE I/O TIME T1: IF WC*ENCA*$\overline{\text{DATA IN}}$*
$\overline{\text{DATA OUT}}$: ↑ ENB

THE DEVICE RECEIVES AN I/O SYNC PULSE (T1) WHICH SETS ENB IN THE M104. AS A RESULT THE DEVICE SENDS RD REQ BACK TO THE I/O PROCESSOR.

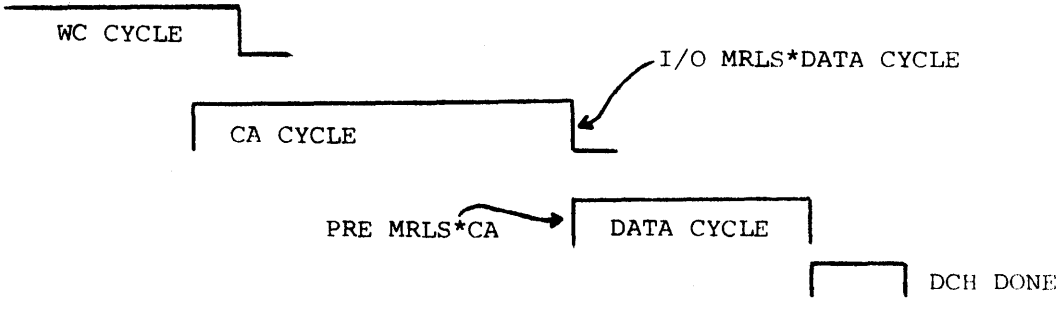
T4: ↑ DATA IN, ↑ IN XFER CYCLE, DCH EN 10P2, THIS OCCURS BEFORE RD RST A SIGNAL FROM MEMORY.



A SHORT TIME AFTER THE I/O ADDRESS HAS BEEN RECEIVED STROB 10P2 GATES THE DATA OFF THE BUS INTO THE I/O BUFFER.

CA * PRE MRLS : ↑ DATA CYCLE

NOTE: THE UPDATED CURRENT ADDRESS WILL BE TRANSFERRED FROM THE DSR → MB 000477+1 → DSR = 000500 → MDL → MB
DSR = 500 THE STARTING ADDRESS



DATA CYCLE

DATA CYCLE * I/O MRLS ACK: ↑I/O MREQ

(DATA CYCLE * I/O MREQ) * DATA IN): ↑I/O WRITE

DSR CONTAINS 500

DATA CYCLE * I/O MRLS ACK: ↑I/O MREQ
NEXT HS CLK: ↑I/O ACTIVE
I/O ACTIVE * I/O MREQ: I/O ADDR-MDL
NEXT HS CLK: ↑MREQ
LOADS 500 INTO MA & STARTS THE MEMORY CYCLE

$\overline{\text{ADD-MEM}}$ * DATA CYCLE * DATA IN) * I/O ADDR ACK:
RETURN DATA
RETURN DATA: ↑DSR-MDL: ↑STROBE DSR
DSR CONTAINS A DATA WORD.
ALSO SET DCH DONE
SET DSR-MDL = ↑DSR-MDL MDL=DATA
SET DSR-MDL DELAYED 75 NS : STROBE MRLS

STROBE MRLS: ↑I/O MRLS: MRLS: LD MB
50 NS LATER : ↑MRLS ACK, ↓RD RST
ALSO START THE WRITE PORTION OF A MEMORY CYCLE

MRLS ACK↓I/O DATA ACK, ↓I/O MRLS, ↓DSR-MDL
MRLS ↓: ↓MRLS ACK

MEMORY WRITE CYCLE FINISHES AND DROPS MBSY.

RESTARTING CPU

$\overline{\text{(ADD-MEM}}$ * DATA CYCLE * DATA IN) * I/O ADDR ACK:
RETURN DATA

RETURN DATA * $\overline{\text{BACK TO BACK}}$: DCH DONE

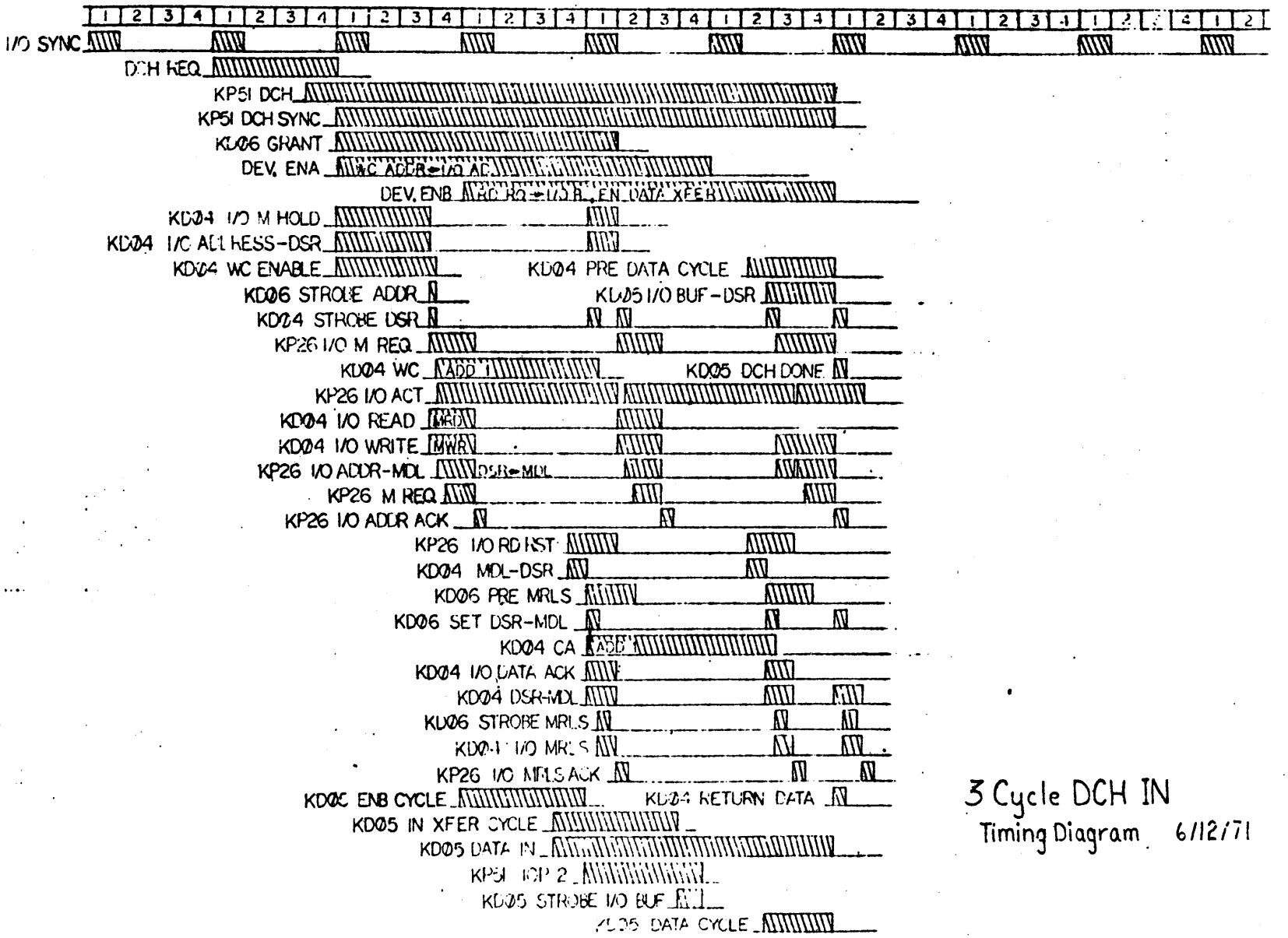
DCH DONE : $\downarrow$ DCH_{FF}, $\downarrow$ DCH $SYNC_{FF}$

THE NEXT TICK OF THE HS CLK : $\uparrow$ MPX
WHEN I/O MRLS ACK, COMES BACK FROM THE DATA
MEMORY CYCLE, IT $\downarrow$ I/O ACTIVE

MPX * $\overline{\text{I/O ACTIVE}}$: I/O INACTIVE

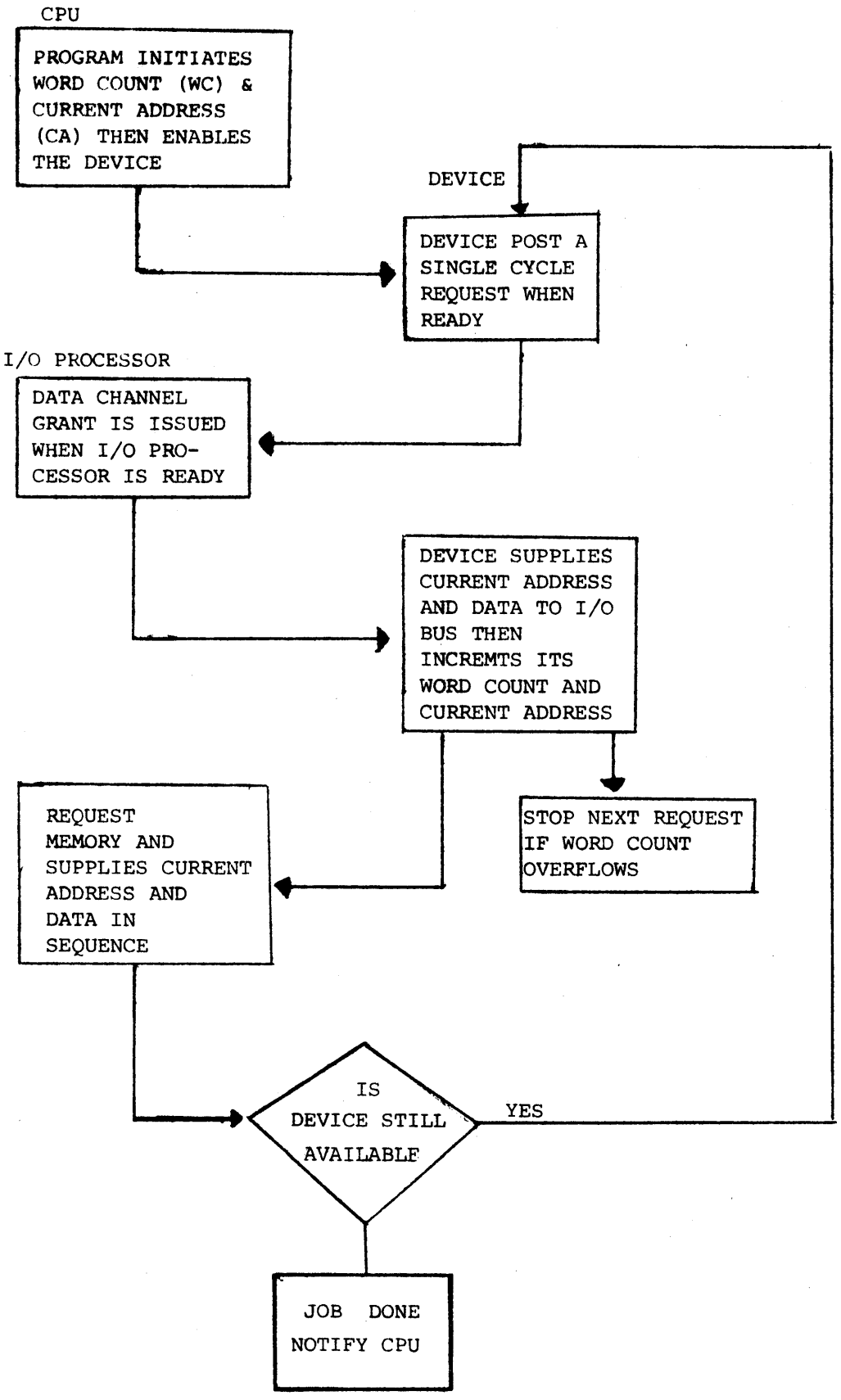
I/O INACTIVE : AS SOON AS CP ACTIVE IS SET AND THE NEXT
TICK OF THE HS CLK:$\uparrow$ STOP PHASE

THE NEXT TICK OF THE HS CLK CHANGES PHASE THE CPU IS NOW RUNNING.



3 Cycle DCH IN
Timing Diagram 6/12/71

SINGLE - CYCLE BLOCK TRANSFER



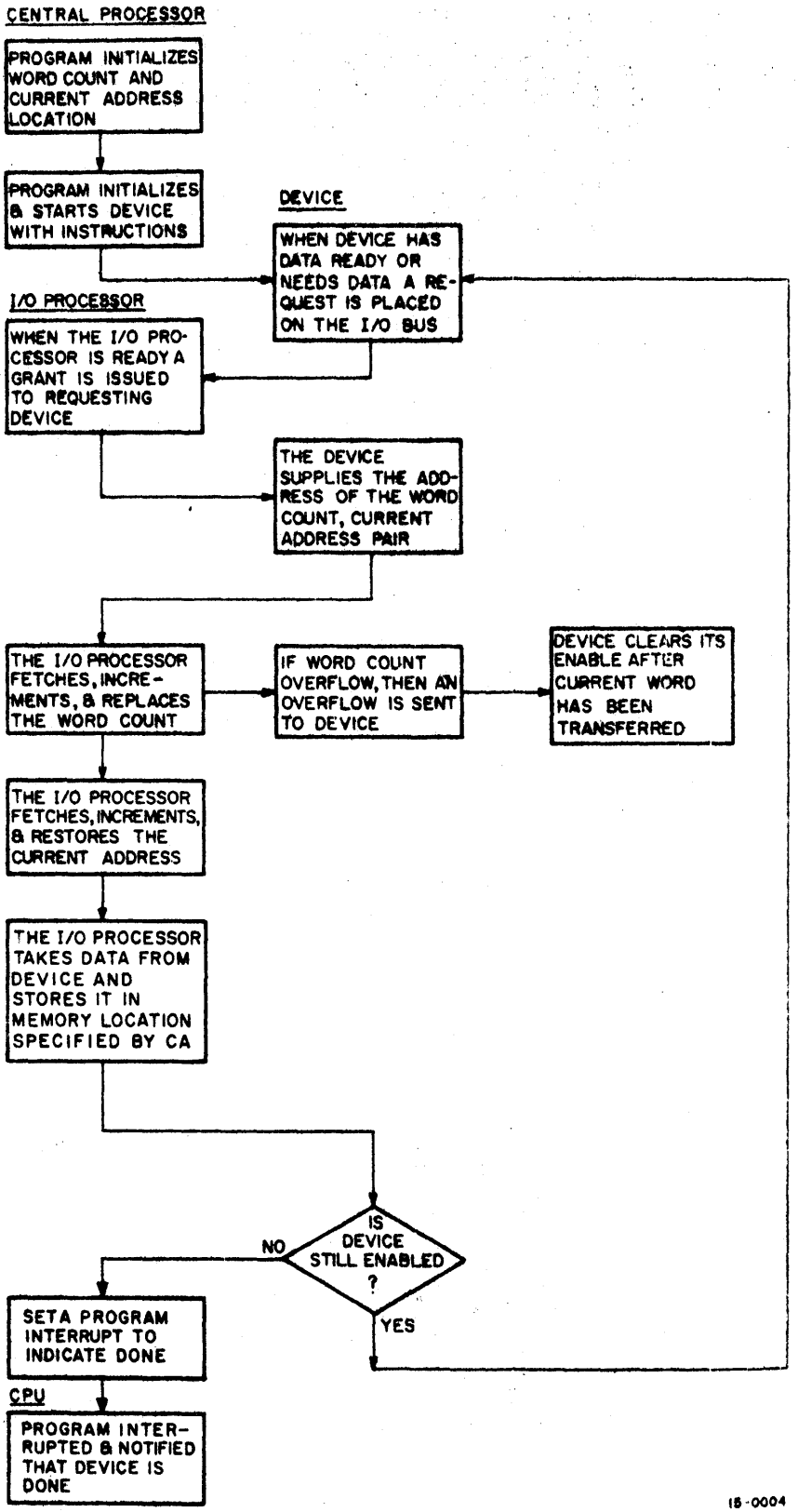


Figure 5-2 Multicycle In Block Transfer, Flowchart

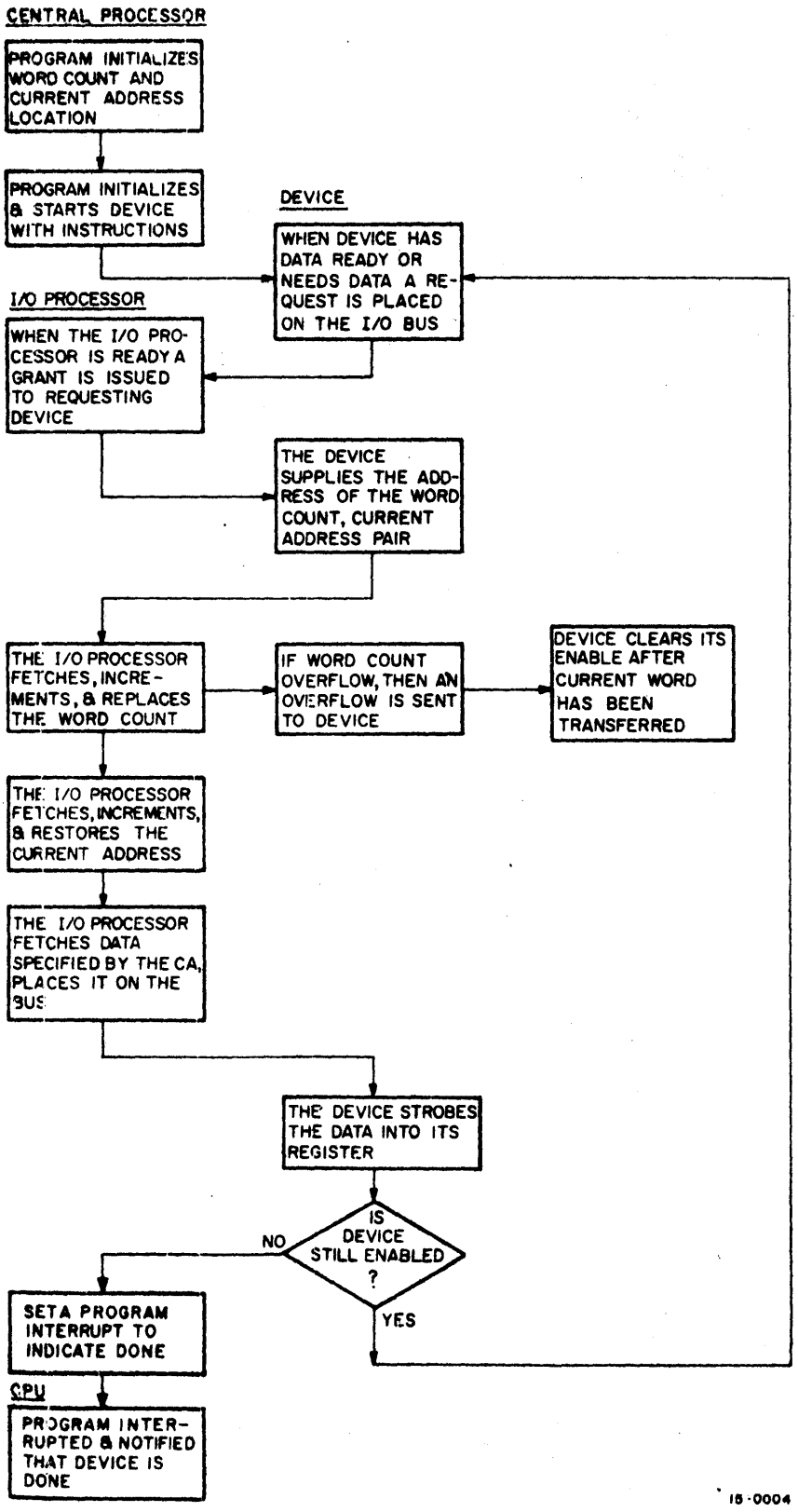
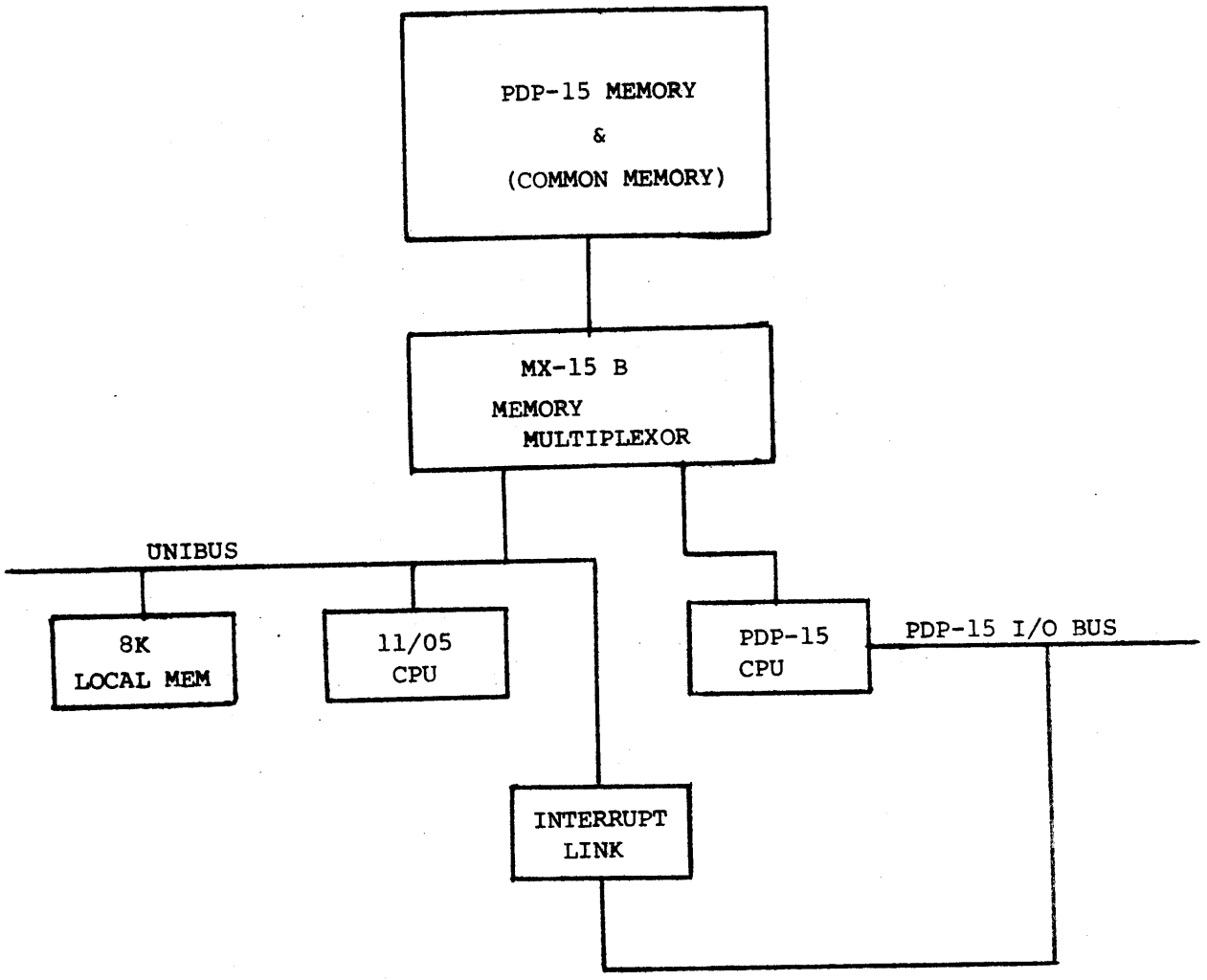
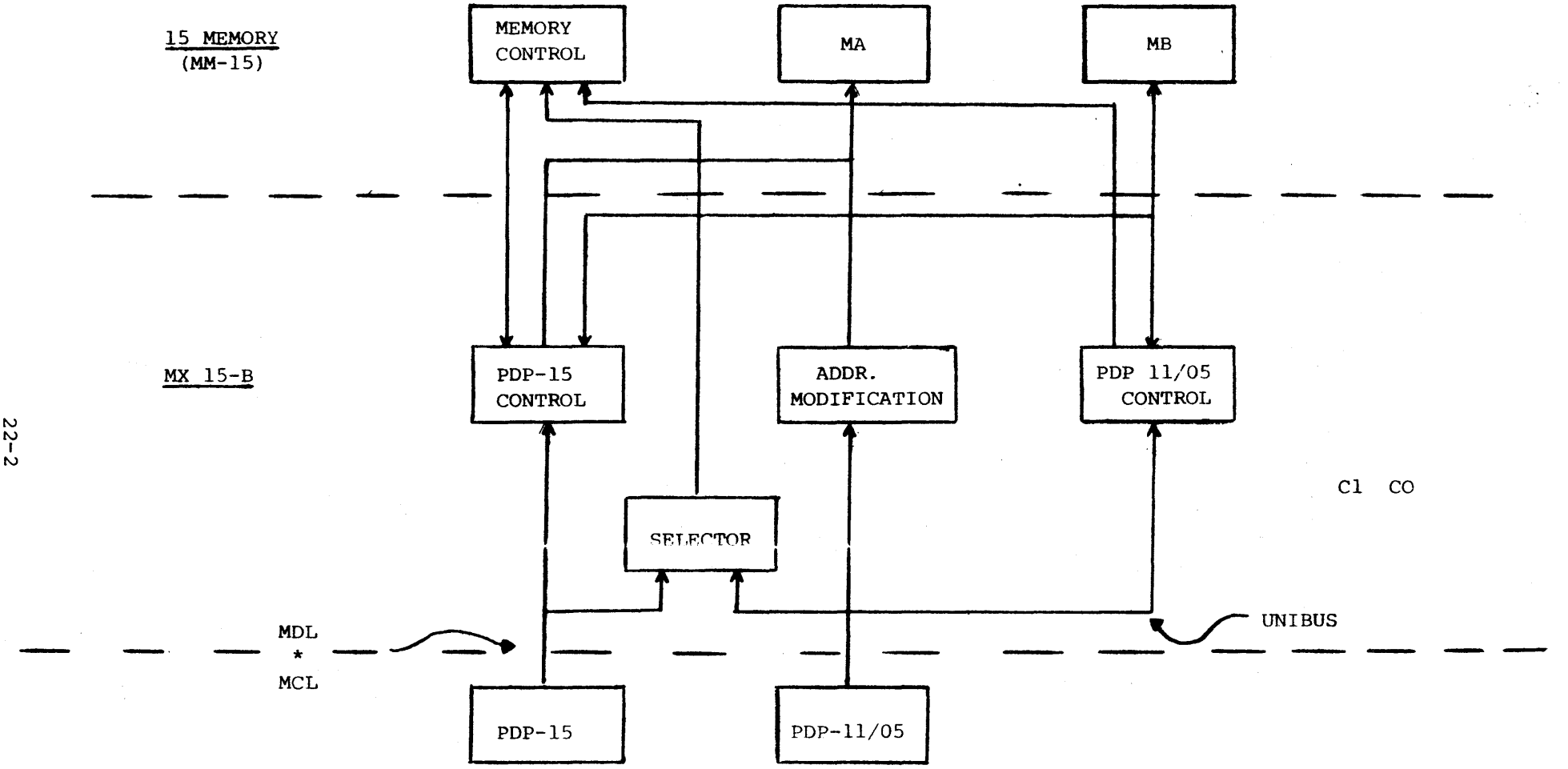


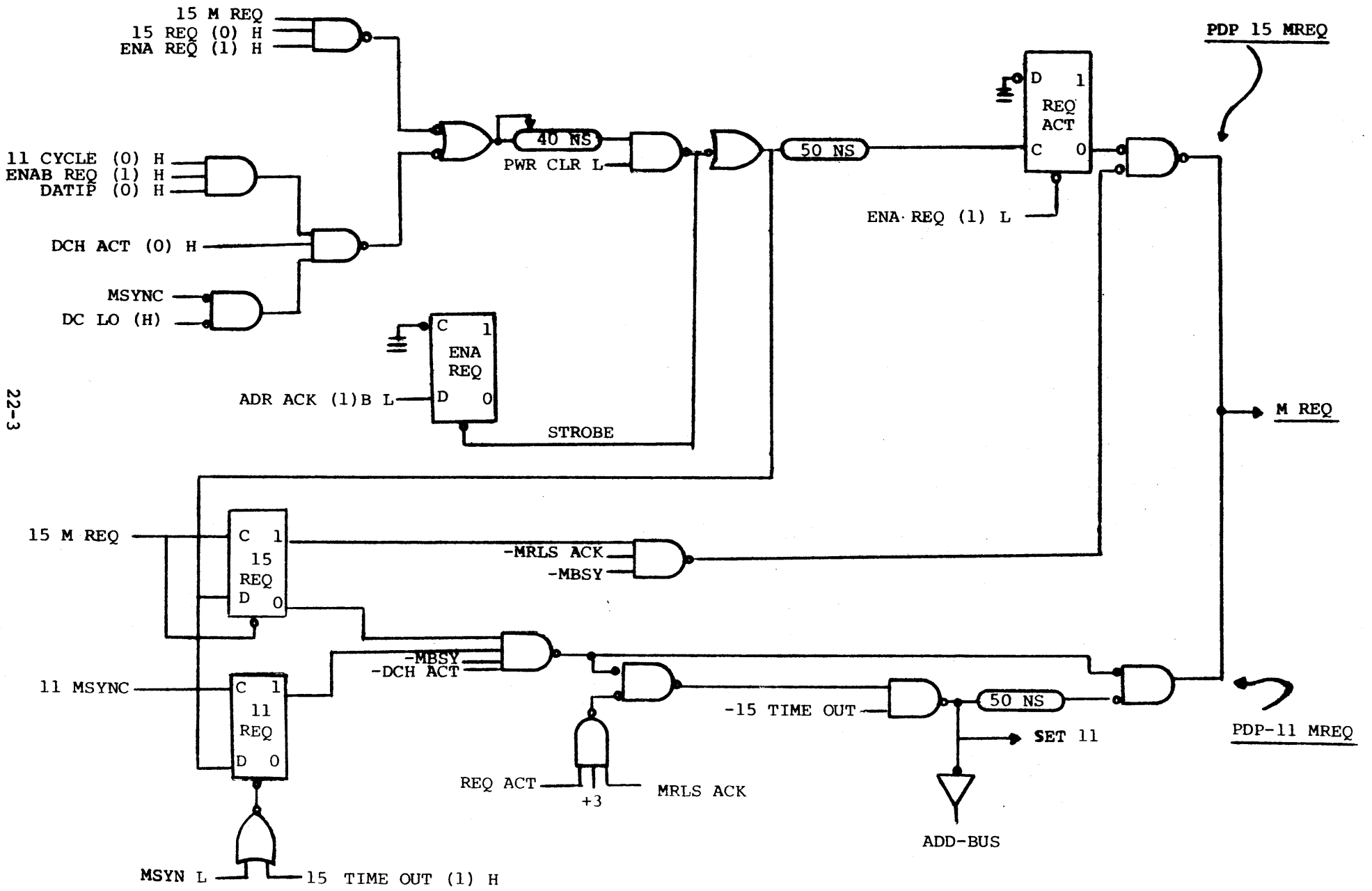
Figure 5-1 Multicycle Out Block Transfer, Flowchart



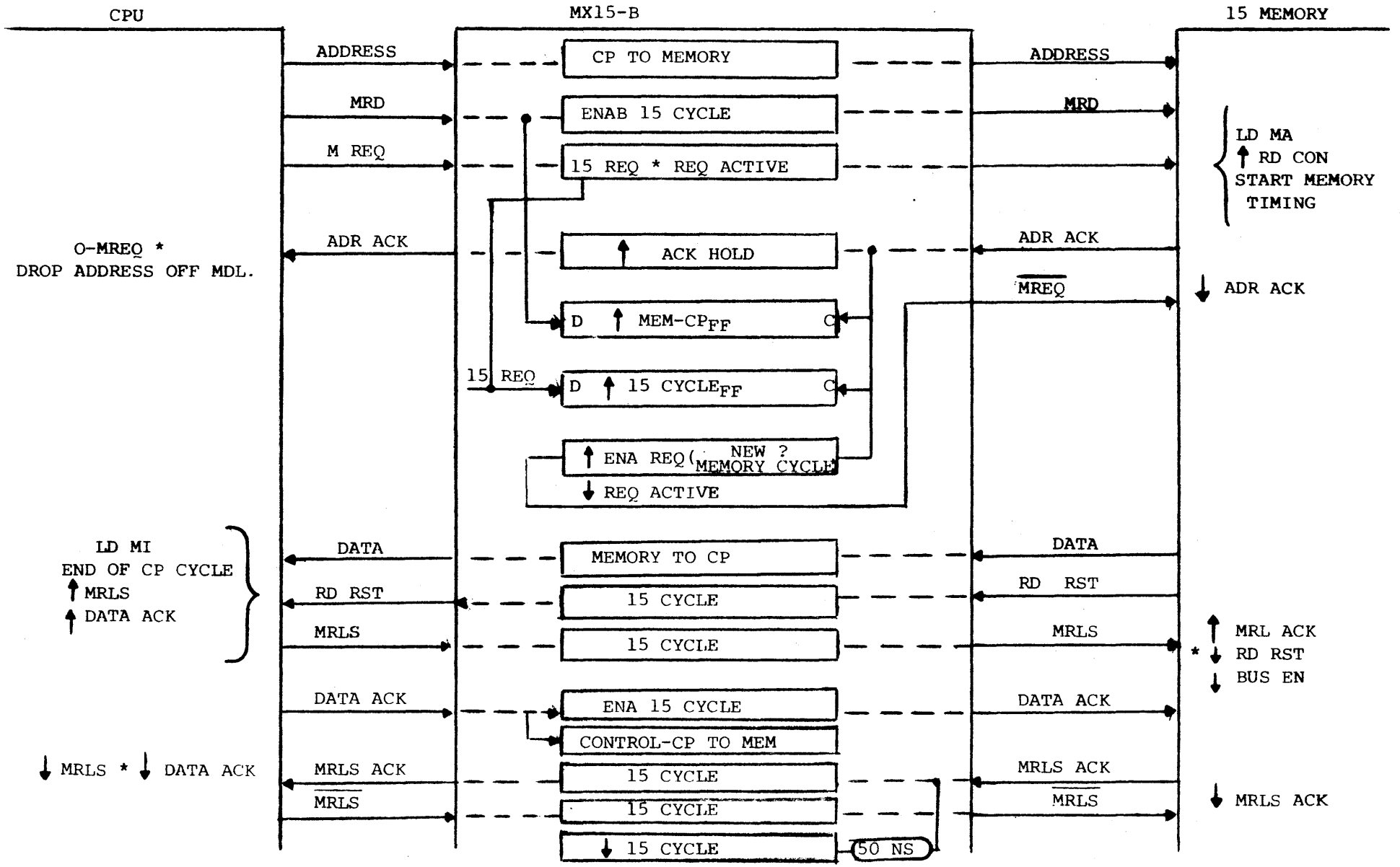
BASIC BLOCK DIAGRAM

BLOCK DIAGRAM OF MEMORY CONTROL PDP-15/76

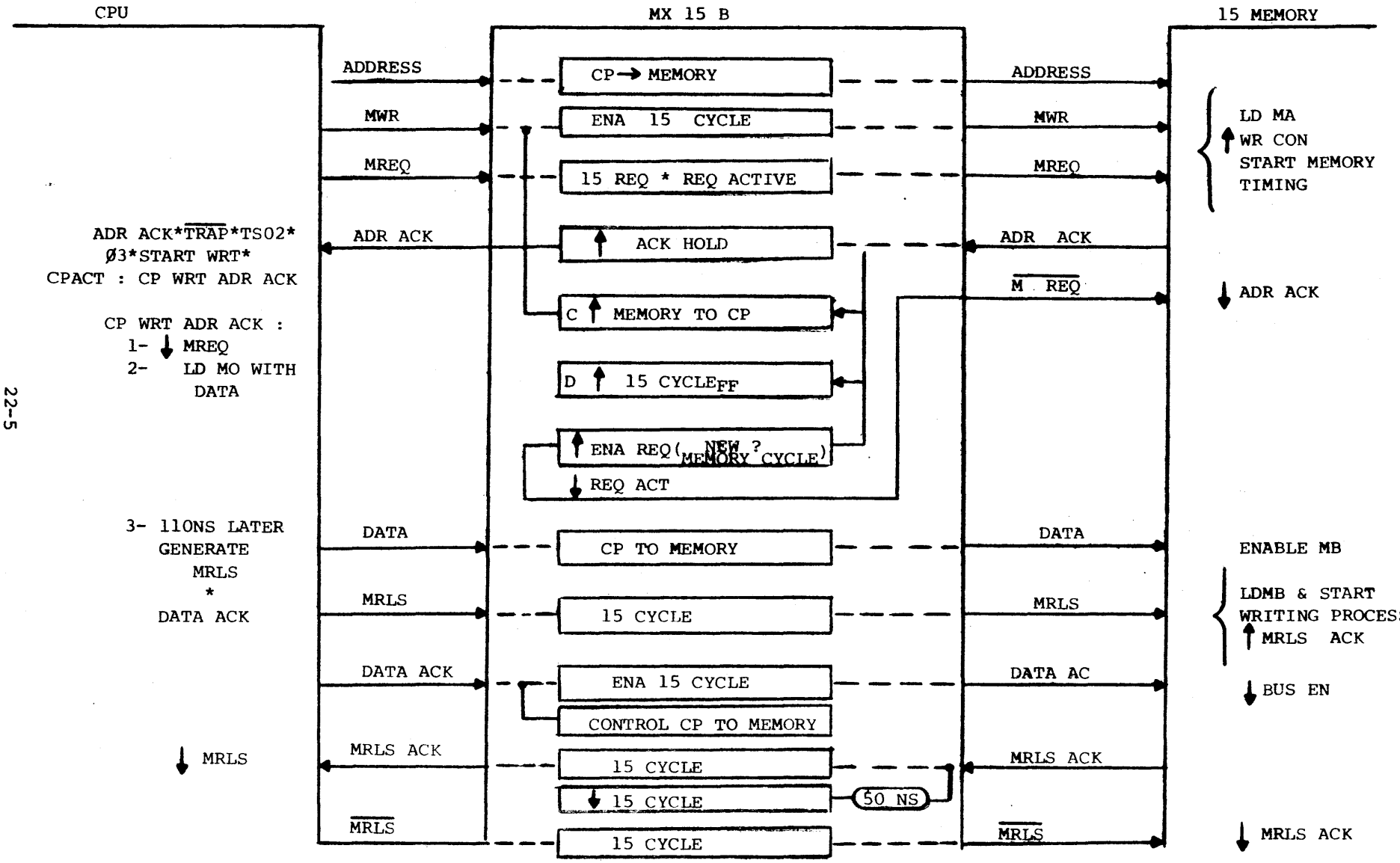



15 M REQ
15 REQ (0) H
ENA REQ (1) H
11 CYCLE (0) H
ENAB REQ (1) H
DATIP (0) H
DCH ACT (0) H
MSYNC
DC LO (H)
40 NS
PWR CLR L
50 NS
D 1
REQ ACT
C 0
ENA REQ (1) L
PDP 15 MREQ
C 1
ENA REQ
D 0
ADR ACK (1)B L
STROBE
M REQ
15 M REQ
C 1
15 REQ
D 0
-MRLS ACK
-MBSY
11 MSYNC
C 1
11 REQ
D 0
-MBSY
-DCH ACT
-15 TIME OUT
50 NS
SET 11
PDP-11 MREQ
REQ ACT
+3
MRLS ACK
ADD-BUS
MSYN L
15 TIME OUT (1) H

PDP-15 READ OPERATION



PDP-15 WRITE OPERATION



THE ADDRESS CONSIDERATIONS ARE AS FOLLOWS;

1. PDP-11 ADDRESS BITS ARE

   BUS A17 THRU BUS AØ

   BUS A17 IS THE MSD
   BUS AØ IS THE LSD

   BUS AØ IS NOT USED FOR ADDRESS DECODE.
   PDP-15 ADDRESS BITS ARE AS FOLLOWS;

   MDL ØØ THRU MDL 17

   MDL ØØ IS NEVER USED FOR ADDRESS DECODING

2. PDP 11 ADDRESSES ARE INCREMENTED BY 2.

   PDP-15 ADDRESSES ARE INCREMENTED BY 1

3. PDP 11 CAN ADDRESS PDP-15 MEMORY THE SECTION OF MEMORY THE PDP 11
   CAN ADDRESS IS CALLED COMMON MEMORY.

4. PDP 11 SYSTEM ASSSUMES ANY ADDRESS ABOVE 124K AS I/O ADDRESS.

5. PDP-11 LOCAL MEMORY CAN BE 8K, 12K, 16K.

6. PDP-11 ADDRESS 28K

AMOUNT OF CORE ADDRESSES	28K
AMOUNT OF LOCAL CORE ADDRESSES	8K
PDP-11 CAN ADDRESS 20K OF COMMON MEMORY.	20K

PDP 11 ADDRESS 160 000 CAUSES
THIS ADDRESS TO BE MODIFY

IN THE PDP-11 IF BITS 13, 14, and 15 are 1's
JAM BITS 16 AND 17 TO A ONE.
ADDRESS 160 000 NOW BECOMES ADDRESS
760 000 WHICH IS THE START OF I/O ADDRESSES.

PDP-11 CAN ONLY ADDRESS COMMON MEMORY AND LOCAL MEMORY.

UNIBUS DEVICES CAN ADDRESS LOCAL MEMORY AND PDP-15 MEMORY
LOCATIONS UP TO 757776 WHICH IS BLOCK 3 PAGE 4.

QUESTION:

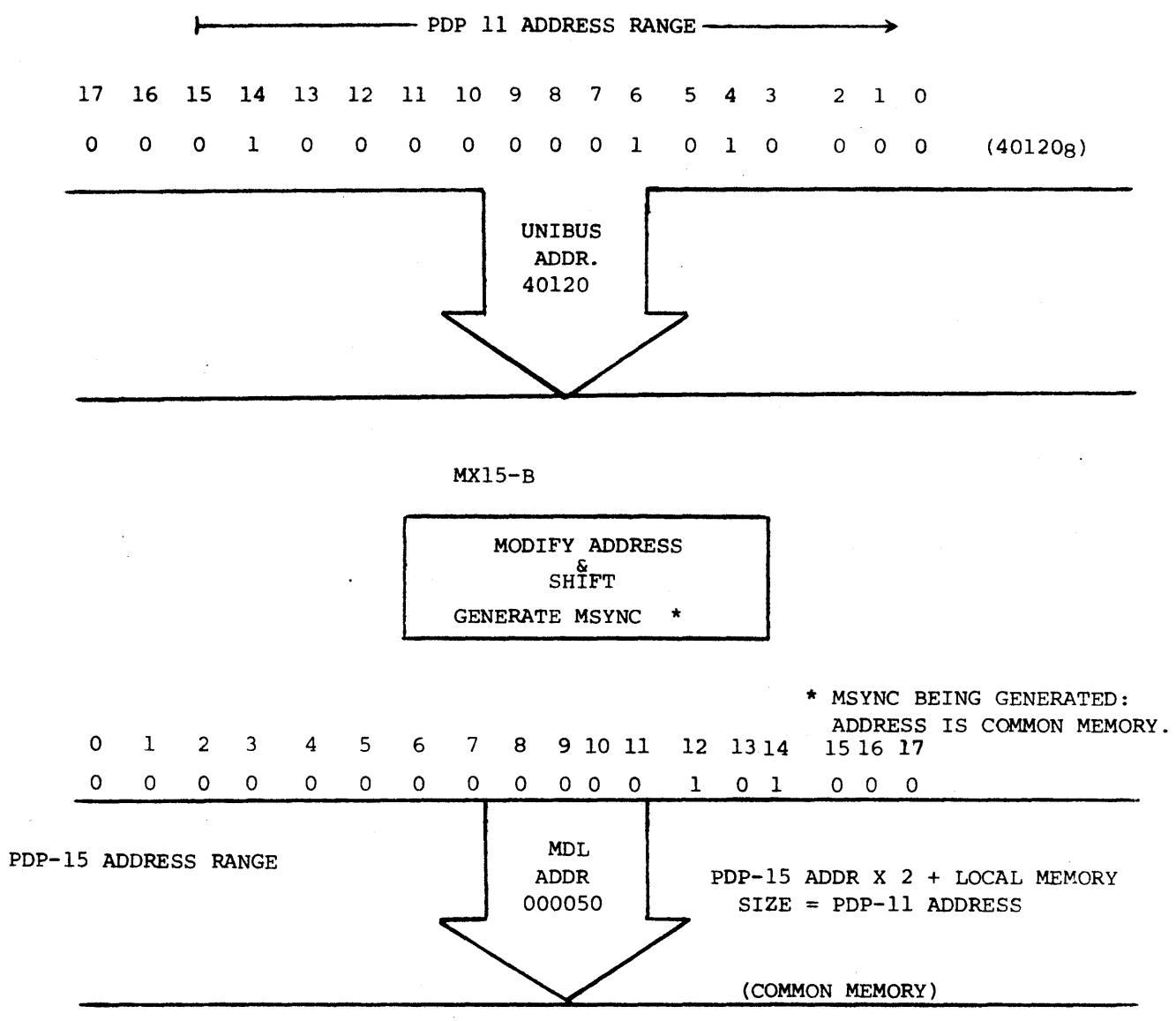
WHAT IS THE PDP-11 ADDRESS THAT WILL ADDRESS
LOCATION 50 IN COMMON MEMORY ?

ANSWER:

15 ADDRESS X2 + LOCAL MEMORY SIZE = PDP-11 ADDRESS

50 x 2 + 40000 =
120 + 40000 = 40120

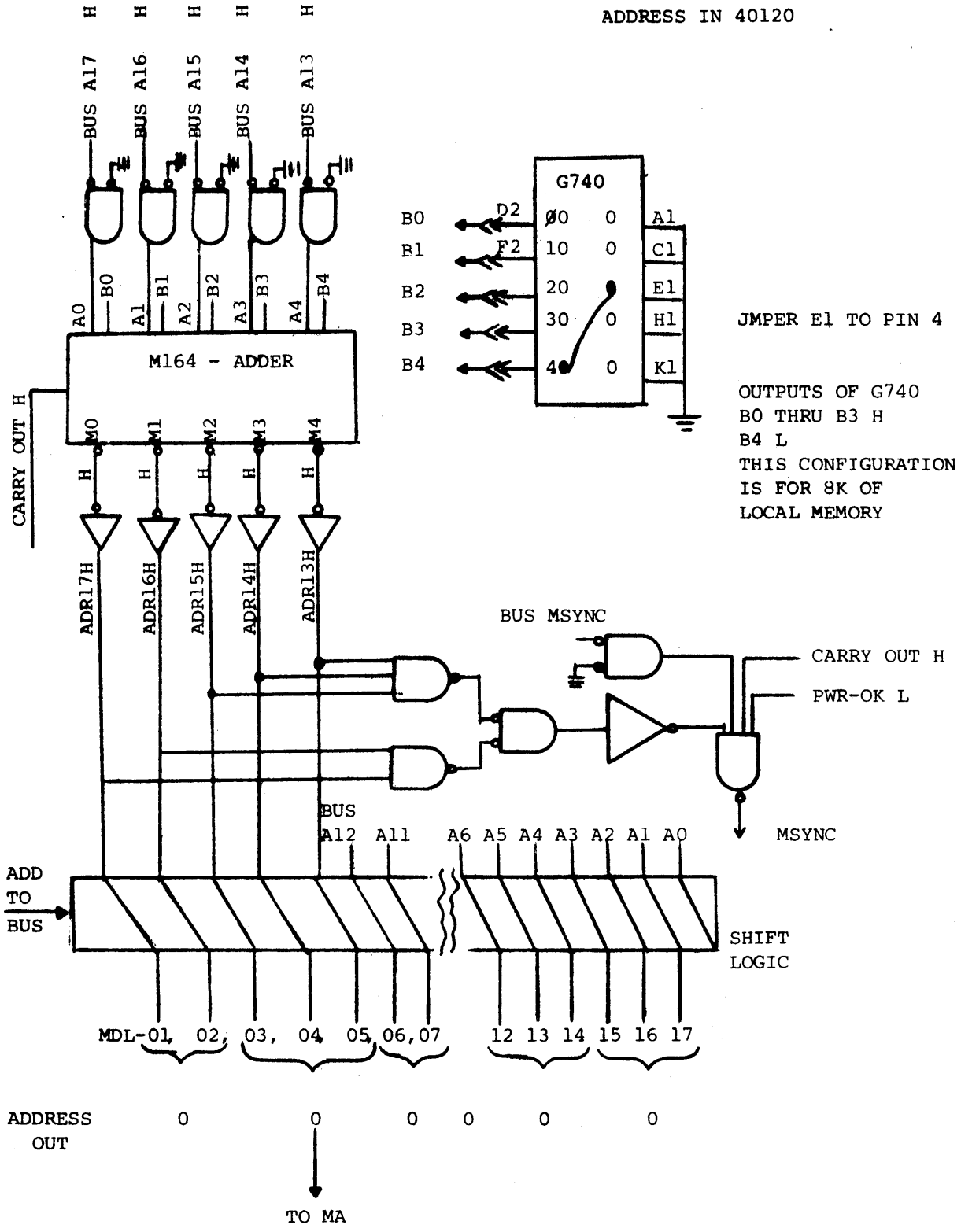
PDP 11 ADDR = PDP 15 ADDR
40120 = 000050

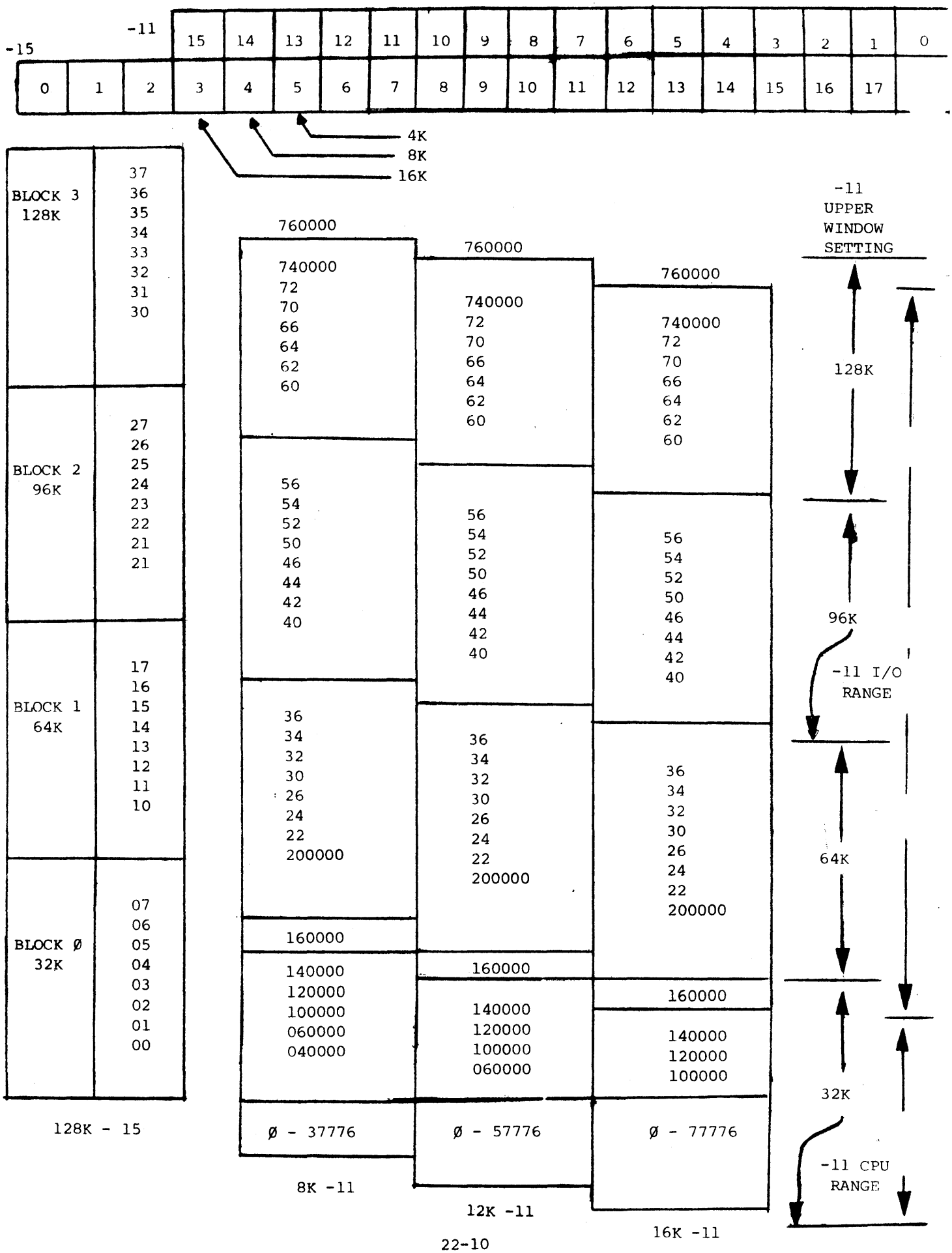


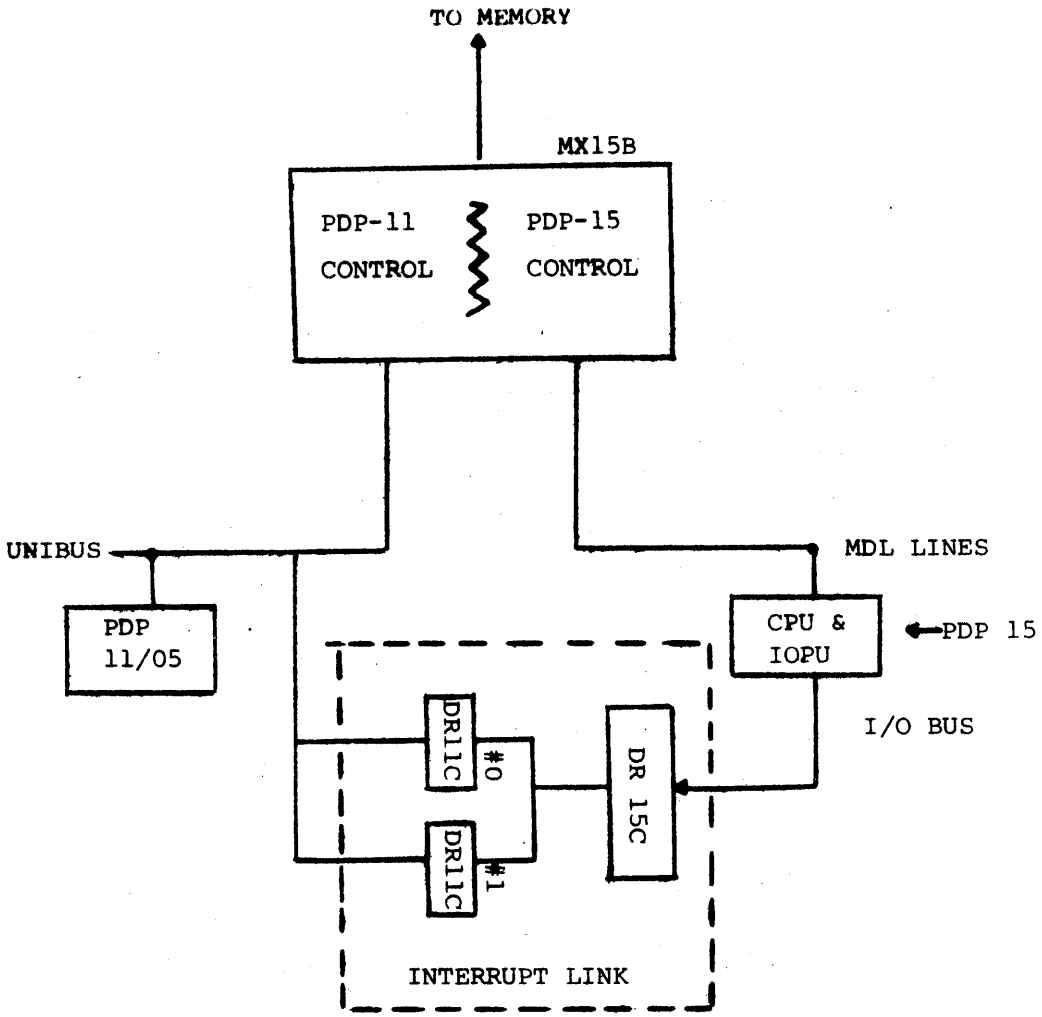
PDP-15 MEMORY

MX 15-B ADDRESS MODIFICATION

ADDRESS IN 40120

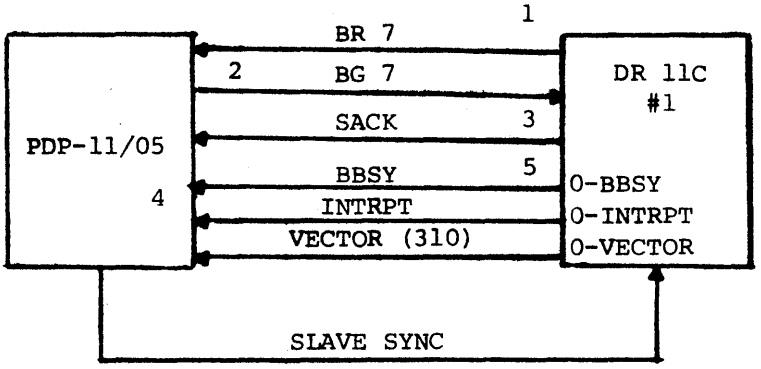



-15
-11
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
4K
8K
16K
BLOCK 3
128K
37
36
35
34
33
32
31
30
BLOCK 2
96K
27
26
25
24
23
22
21
21
BLOCK 1
64K
17
16
15
14
13
12
11
10
BLOCK Ø
32K
07
06
05
04
03
02
01
00
128K - 15
760000
740000
72
70
66
64
62
60
56
54
52
50
46
44
42
40
36
34
32
30
26
24
22
200000
160000
140000
120000
100000
060000
040000
Ø - 37776
8K -11
760000
740000
72
70
66
64
62
60
56
54
52
50
46
44
42
40
36
34
32
30
26
24
22
200000
160000
140000
120000
100000
060000
Ø - 57776
12K -11
760000
740000
72
70
66
64
62
60
56
54
52
50
46
44
42
40
36
34
32
30
26
24
22
200000
160000
140000
120000
100000
Ø - 77776
16K -11
-11
UPPER
WINDOW
SETTING
128K
96K
-11 I/O
RANGE
64K
32K
-11 CPU
RANGE



EXPANDED INTERRUPT LINK

BUS REQUEST



(1) DEVICE ASSERTS BR
GRANTED AT END OF INSTRUCTION
PRIORITY ARBITRATION WITH PSW
AND OTHER BRS
(BR 7-THRU BR4)

(2) CPU ASSERTS
BUS GRANT

(3) DEVICE ASSERTS SACK
DROP BR REQ

(4) CPU DROPS
BBSY & GRANT

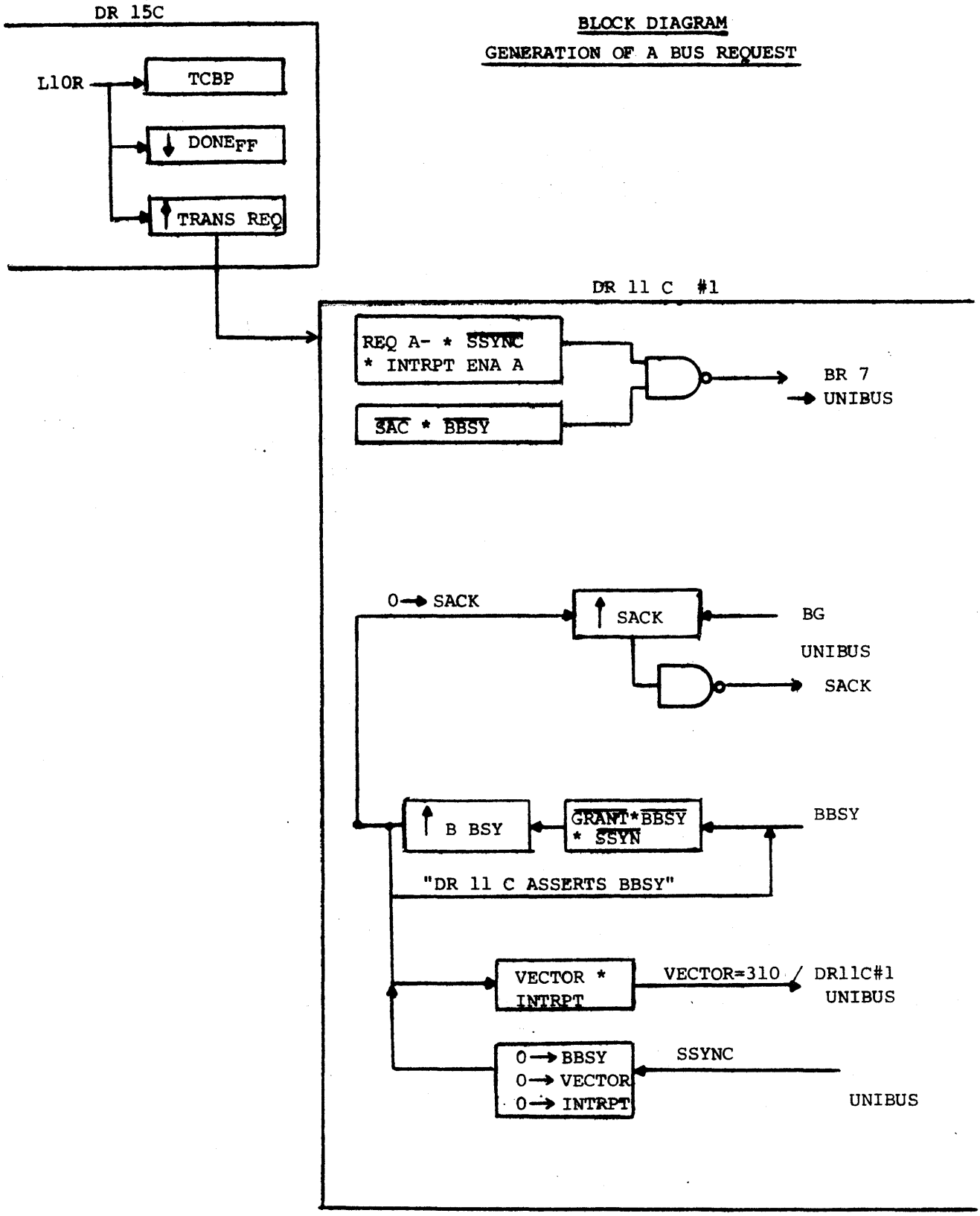
(5) DEVICE ASSERTS BBSY
DROPS SACK BRINGS UP
VECTOR ADDRESS AND INTRPT

(6) CPU RECEIVES
VECTOR * INTRPT
PROCESSES INTRPT
IF PRIORITY IS
HIGH ENOUGH -
GO TO ROUTINE
THAT WILL READ IN
TCBP LOCATED IN
ADDR 767764.

SSYNC

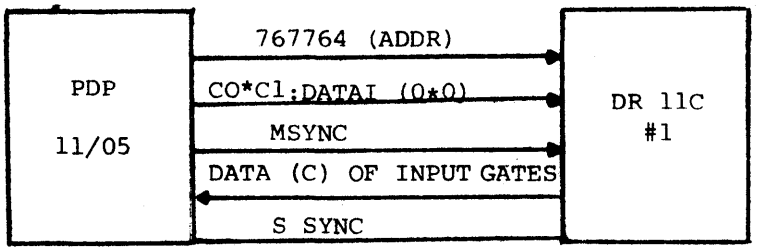
0 → BBSY
0 → INTRPT
0 → VECTOR

BLOCK DIAGRAM
GENERATION OF A BUS REQUEST

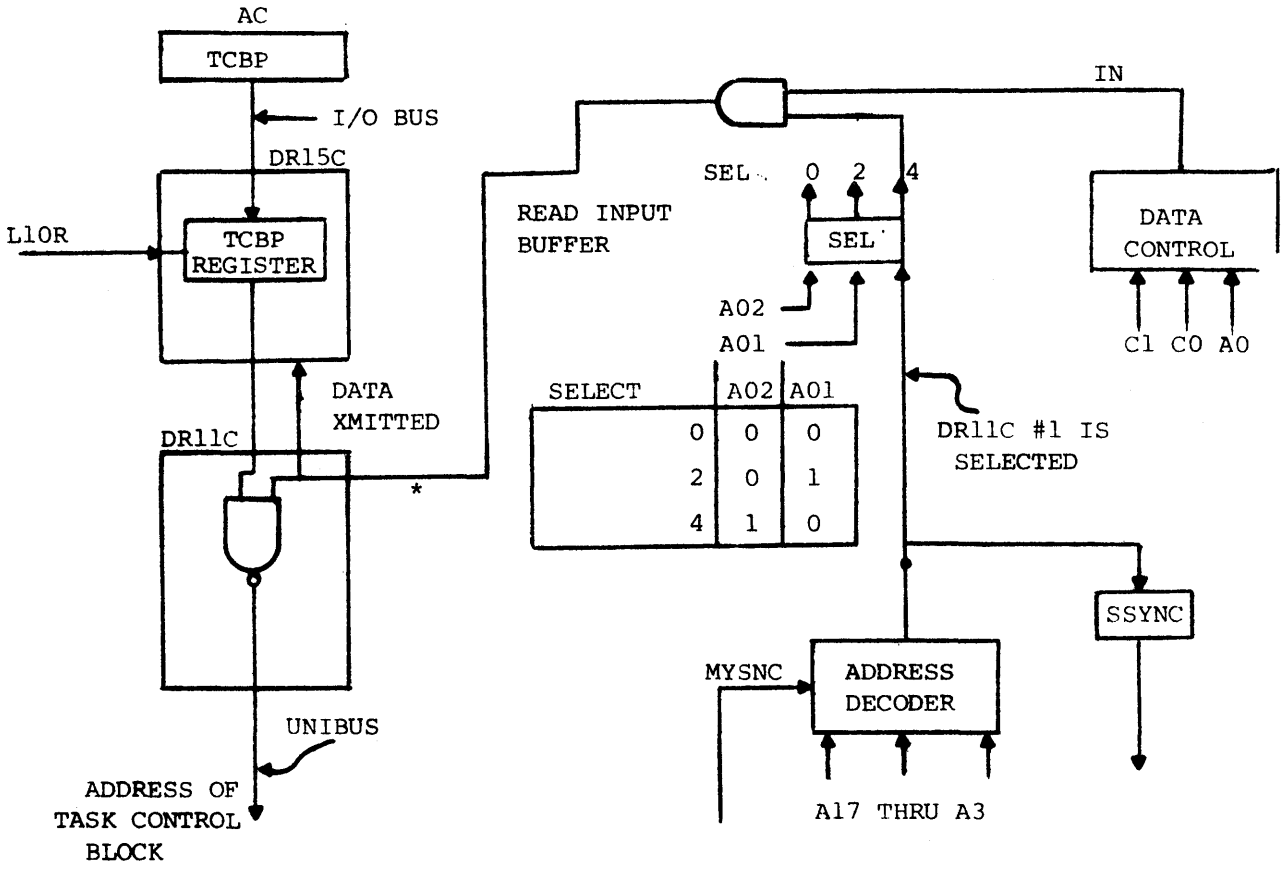


DATA IN TRANSFER

GET THE TASK CONTROL BLOCK ADDRESS



PDP 11/05 READS CONTENTS OF THE INPUT GATES WHICH CONTAINS THE CONTENTS OF THE TASK CONTROL BLOCK POINTER REGISTER IN THE DR-15C.



* READ IN LINES : READS CONTENTS OF INPUT GATES TO THE UNIBUS

READ IN LINE : DATA TRANSMITTED
DATA TRANSMITTED : ↓ TRANSFER REQ IN DR15C
↑ DONE FF

DATA CONTROL CHART

	C1	C0	A0	CONTROL
DATA IN	0	0	0	IN
	0	0	1	IN
DATA IN	0	1	0	IN
PAUSE	0	1	1	IN
DATA OUT	1	0	0	OUT LOW OUT HIGH
	1	0	1	OUT HIGH
DATA OUT	1	1	0	OUT LOW
BYTE	1	1	1	OUT HIGH

PDP-11 GENERATES API REQ (JOB DONE)

TASK CONTROL BLOCK POINTER → TO WORD ZERO OF THE BLOCK

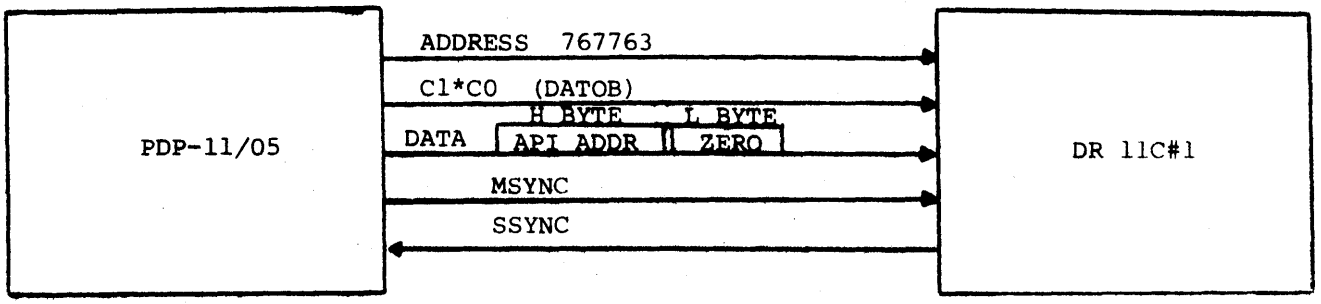
	HIGH BYTE	LOW BYTE
WORD ZERO	API PRIORITY LEVEL 0,1,2,3	API ADDRESS
WORD ONE	FUNCTION	TASK CODE
WORD TWO	REQUEST EVENT VARIABLE	
WORD THREE	THRU 3=N DEPEND UPON PARTICULAR TASK	

	HIGH BYTE	LOW BYTE
WORD ZERO	001	200
WORD ONE	FUNCTION: INTERRUPT WHEN JOB DONE	TASK CODE RK 05

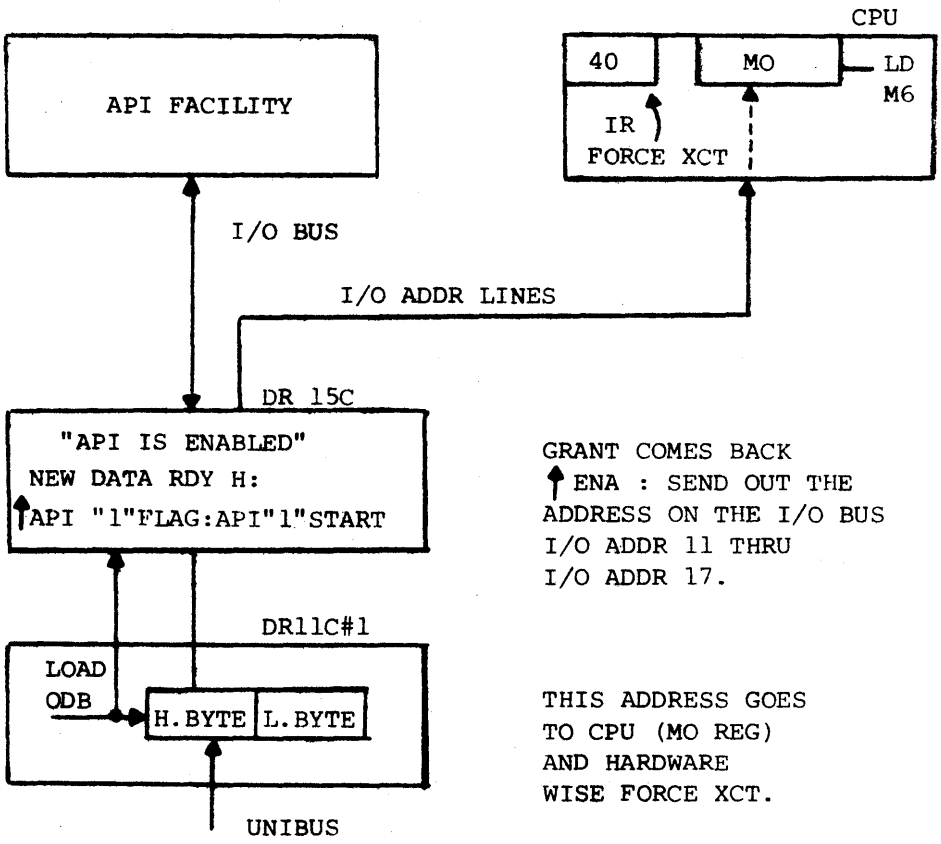
API LEVEL ONE ADDRESS IS ASSOCIATED WITH DR 11C #1 OUTPUT BUFFER, WHOSE ADDRESS IS 767773, HIGH BYTE.

		OUTPUT BUFFER ADDRESS
API LEVEL ZERO	: DR 11C HI LOW BYTE	767762
API LEVEL ONE	: DR 11C#1 HIGH BYTE	767763
API LEVEL TWO	: DR 11C#0 LOW BYTE	767772
API LEVEL THREE	: DR 11C#0 HIGH BYTE	767773

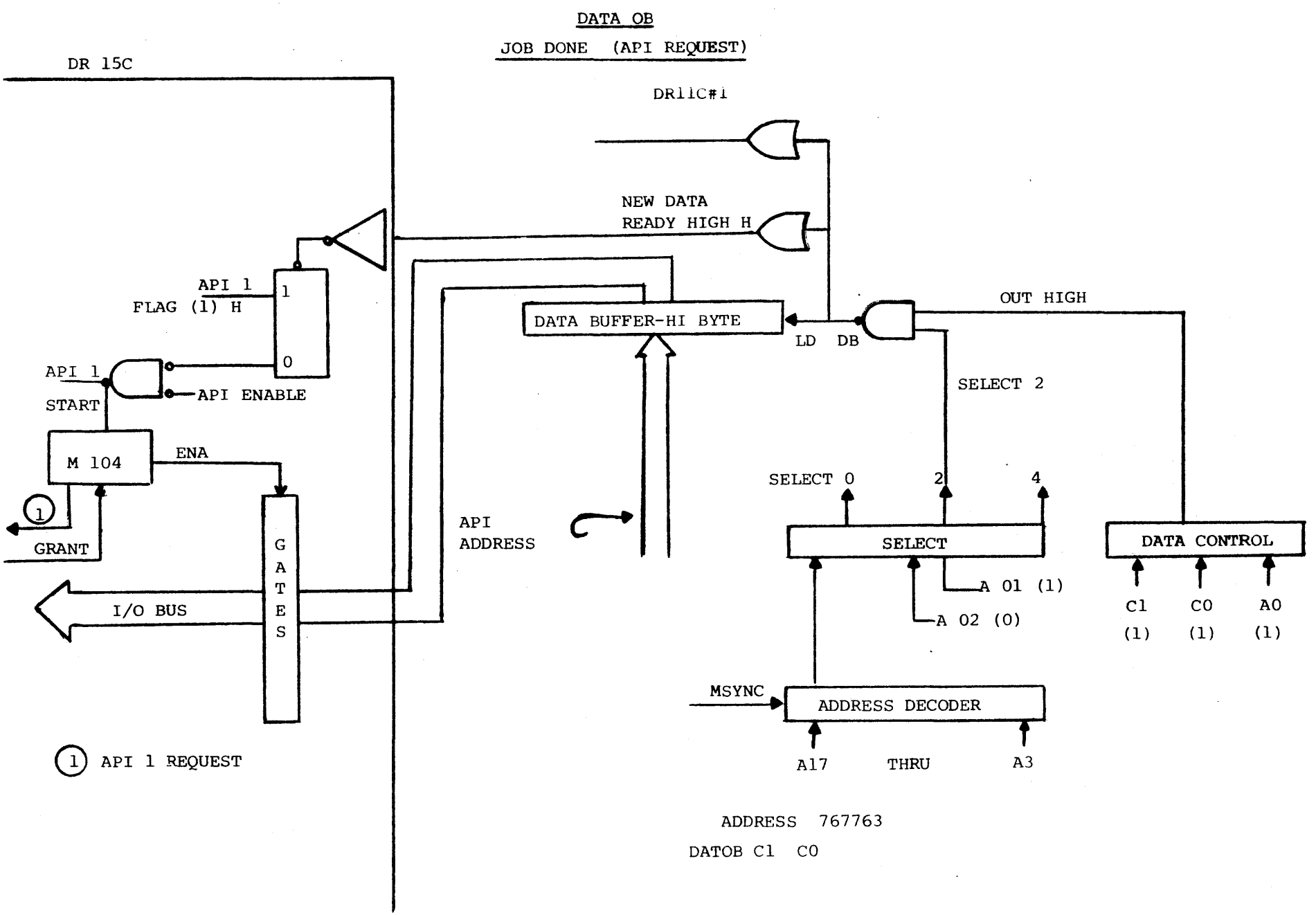
DATO B



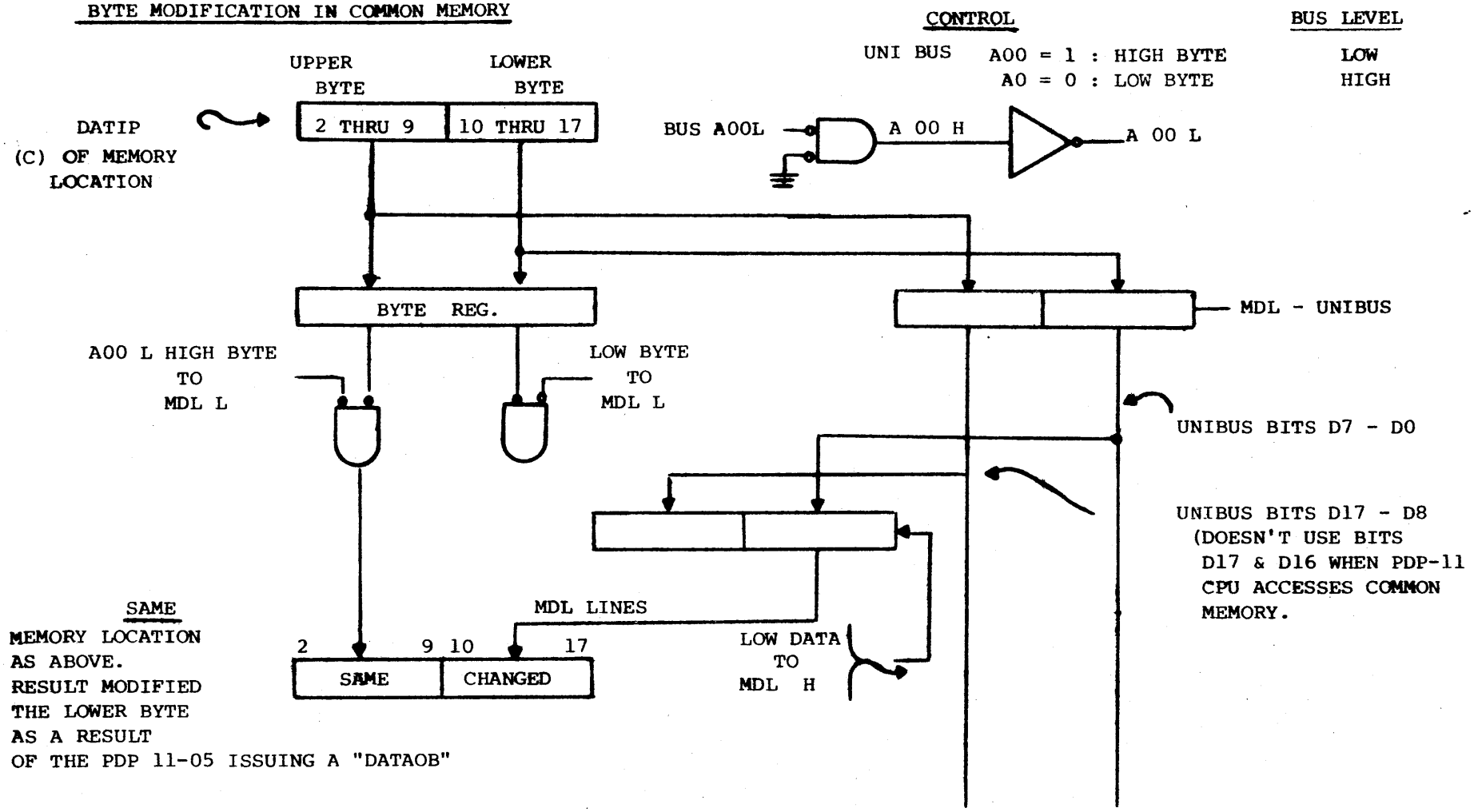
PURPOSE: LOAD THE DATA OUT BUFFER HIGH BYTE WITH THE API ADDRESS.
GENERATE API REQ ON CHANNEL ONE. THIS GOES THRU THE API FUNCTION WHICH WILL SEND BACK A GRANT BY WAY OF THE I/O BUS



THIS MEANS AS SOON AS THE CPU ENTERS THE FETCH MAJOR STATE AND WORKS ON THE OPERAND XCT 200
(C) 200 = JMS TO SUBROUTINE TO HANDLE THE JOB DONE INTERRUPT.


DATA OB
JOB DONE (API REQUEST)
DR 15C
DR11C#1
NEW DATA
READY HIGH H
API 1
FLAG (1) H
1
0
API 1
START
API ENABLE
M 104
ENA
GRANT
1
G
A
T
E
S
I/O BUS
API
ADDRESS
DATA BUFFER-HI BYTE
LD DB
OUT HIGH
SELECT 2
SELECT 0
2
4
SELECT
A 01 (1)
A 02 (0)
DATA CONTROL
C1
C0
A0
(1)
(1)
(1)
MSYNC
ADDRESS DECODER
A17
THRU
A3
1 API 1 REQUEST
ADDRESS 767763
DATOB C1 C0

BYTE MODIFICATION IN COMMON MEMORY

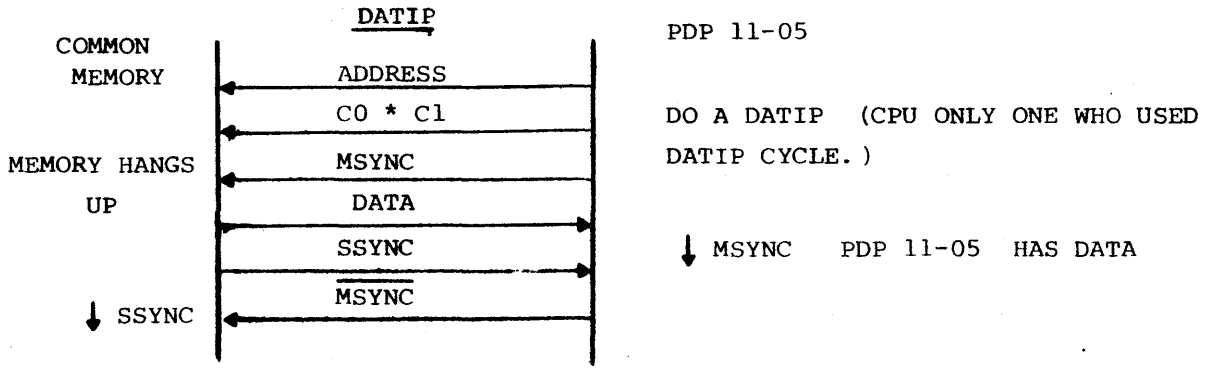


SAME

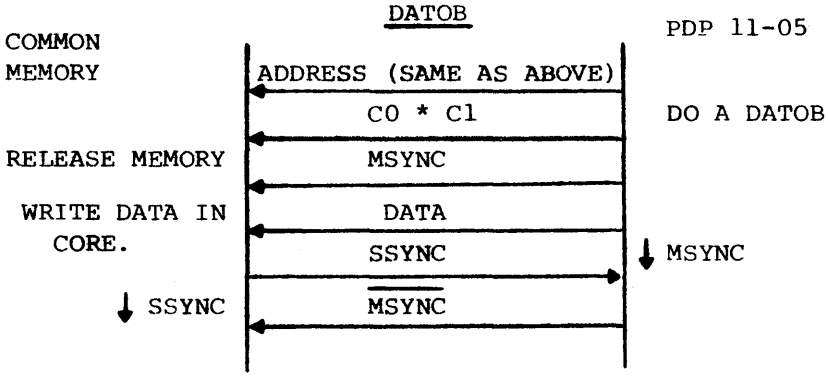
MEMORY LOCATION AS ABOVE. RESULT MODIFIED THE LOWER BYTE AS A RESULT OF THE PDP 11-05 ISSUING A "DATAOB"

MODIFY LOW BYTE

ADDRESS BIT A00=0 : THE BUS LEVEL IS HIGH THERE FOR GATE HI BYTE TO MDL * LOW DATA TO MDL H



NOTE: DATIP ALWAYS FOLLOWED BY A DATO OR $DATO_B$



XVM HARDWARE

To be supplied at a later date.