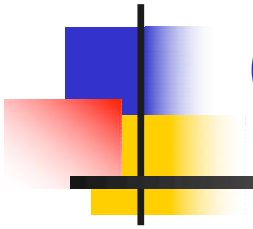


Chapter 4: Computer Prototype

Register Transfer and Microoperations

Register Transfer Language

- **Microoperation**
  - The operations executed on data stored in registers*(shift, clear, load, count)*
- **Internal H/W Organization**
  1. The set of registers
  2. The sequence of microoperations
  3. The sequence control of microoperations
- **Register Transfer Language**
  - The symbolic notation used to describe the microoperation transfer among registers
    - The use of symbols instead of a narrative explanation provides an organized and concise manner
  - A convenient tool for describing the internal organization of digital computers in concise and precise manner

Register Transfer

- **Registers :**
  - Designated by Capital Letter(sometimes followed by numerals) : *MAR(Memory Address Register), PC(Program Counter), IR(Instruction Register), R1(Processor Register)*
  - The numbering of bits in a 16-bit register : marked on top of the box
  - A 16-bit register partitioned into two parts : bit 0-7(*symbol "L" Low byte*), bit 8-15(*symbol "H" High byte*)
- **Register Transfer** : Information transfer from one register to another
  - (*transfer of the content of register R1 into register R2*)
    - The content of the source register R1 does not change after the transfer

$$R2 \leftarrow R1$$

Register Transfer

- **Control Function** : The transfer occurs only under a predetermined control condition
  - The transfer operation is executed by the hardware only if P=1

$$if\ (P = 1)\ then\ (R2 \leftarrow R1)$$
$$P : R2 \leftarrow R1$$

  - A comma is used to separate two or more operations(*Executed at the same time*)

$$T : R2 \leftarrow R1,\ R1 \leftarrow R2$$

Register Transfer

- **Basic Symbols for Register Transfer :**

Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow <--	Denotes transfer of information	R2 <-- R1
Comma ,	Separates two microoperations	R2 <-- R1, R1 <-- R2

Memory Transfers

- **Memory Transfer**
  - **Memory Read : A transfer information into register from the memory word M selected by the address in MAR**
  - **Memory Write : A transfer information from R1 into the memory word M selected by the address in MAR**

- **The 4 types of microoperation in digital computers**
  1. **Register transfer microoperation :**
  2. **Arithmetic microoperation**
  3. **Logic microoperation**
  4. **Shift microoperation**

Arithmetic Microoperation

- **Arithmetic Microoperation** :
  - Negate : 2's complement

$$R2 \leftarrow \overline{R2} + 1$$

  - Subtraction : R1 + 2's complement of R2

$$R3 \leftarrow R1 - R2 = R1 + (\overline{R2} + 1)$$

  - Multiplication(shift left), Division(shift right)

Logic Microoperation

- **Logic microoperation**
  - Logic microoperations consider ***each bit of the register separately*** and treat them as binary variables

$$P: R1 \leftarrow R1 \oplus R2$$

- **Special Symbols**
  - **Special symbols will be adopted for the logic microoperations**
  - ***OR($\vee$), AND($\wedge$),* and *complement(a bar on top)*, to distinguish them from the corresponding symbols used to express Boolean functions**

$$P + Q: R1 \leftarrow R2 + R3,\ \ R4 \leftarrow R5 \vee R6$$

Logic OR

Arithmetic ADD

Shift Microoperation

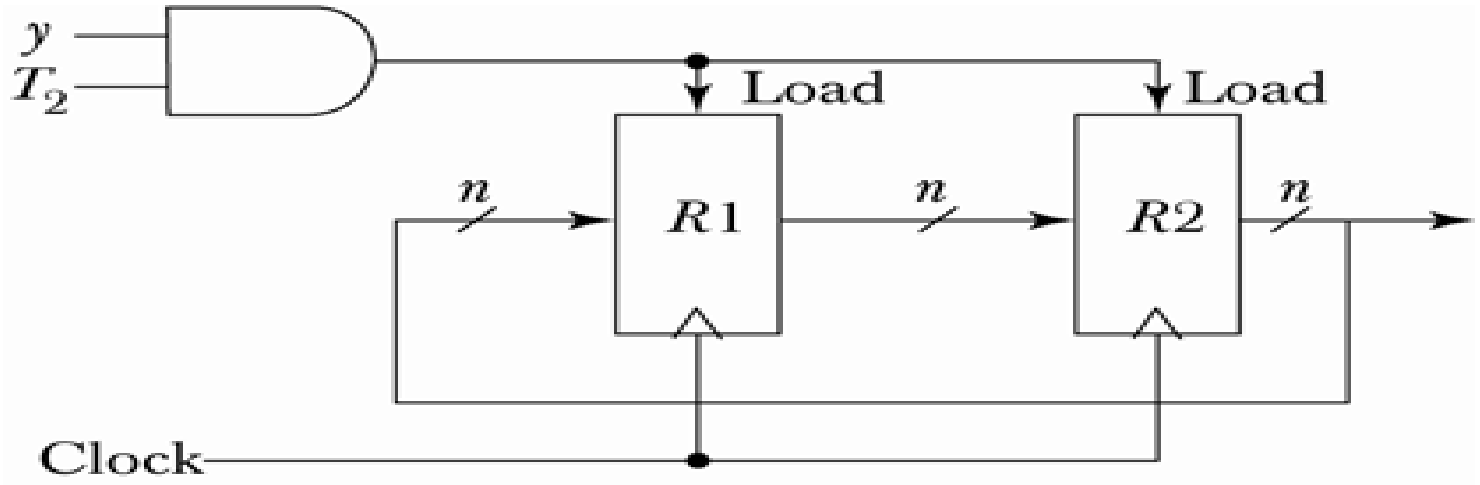
- **Shift Microoperations** :
  - Shift microoperations are used for serial transfer of data
  - Three types of shift microoperation : *Logical, Circular,* and *Arithmetic*
- **Logical Shift**
  - A *logical shift* transfers 0 through the serial input
  - The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift(*Zero inserted*)

$$R1 \leftarrow shl \quad R1$$
$$R2 \leftarrow shr \quad R2$$

Exercise 1

Show the block diagram of the hardware that implements the following register transfer statements:

yt_2: R2 ←R1, R1 ←R2



Exercise 2

2. The outputs of four registers , R0,R1,R2 and R3 , are connected through 4 to 1 line multiplexers to the inputs of a fifth register, R5. Each register is eight bits long . The required transfers are dictated by four timing variables T_0 through T_3 as follows:

$$T_0 : R5 \leftarrow R0$$

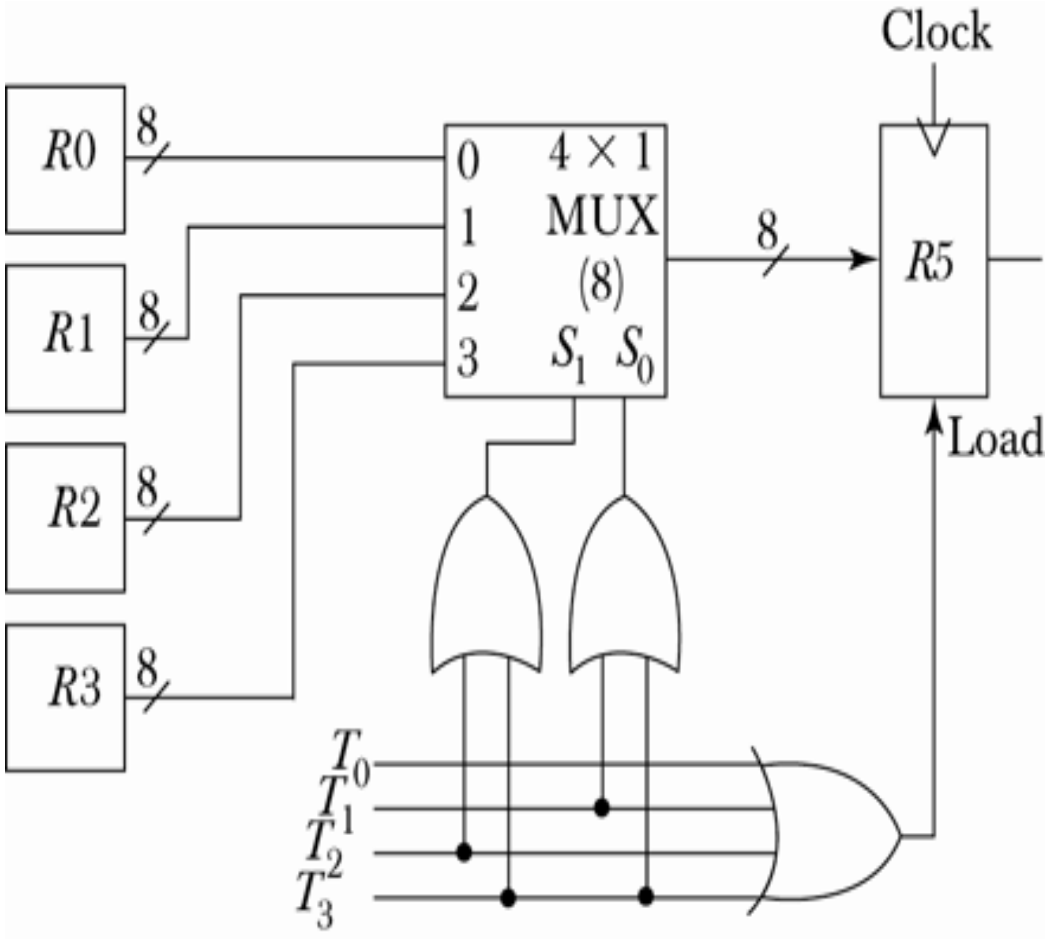
$$T_1 : R5 \leftarrow R1$$

$$T_2 : R5 \leftarrow R2$$

$$T_3 : R5 \leftarrow R3$$

The timing variables are mutually exclusive, which means that only one variable is equal to 1 at any given time. While the other three are equal to 0. draw a block diagram showing the hardware implementation of register transfers. Include the connections from the four timing variables to the selection inputs of the multiplexers and to the load input of register R5.

Solution



$T_0T_1T_2T_3$	S_1 S_0 R_3 load
0 0 0 0	X X 0
1 0 0 0	0 0 1
0 1 0 0	0 1 1
0 0 1 0	1 0 1
0 0 0 1	1 1 1

$S_1 = T_2 + T_3$

$S_0 = T_1 + T_3$

load = $T_0 + T_1 + T_2 + T_3$

Exercise 3

3.Represent the following conditional control statement by two register transfer statements with control functions.

If (P=1) then (R1 ← R2) else if (Q=1) then (R1 ← R3)

P: R1 ← R2

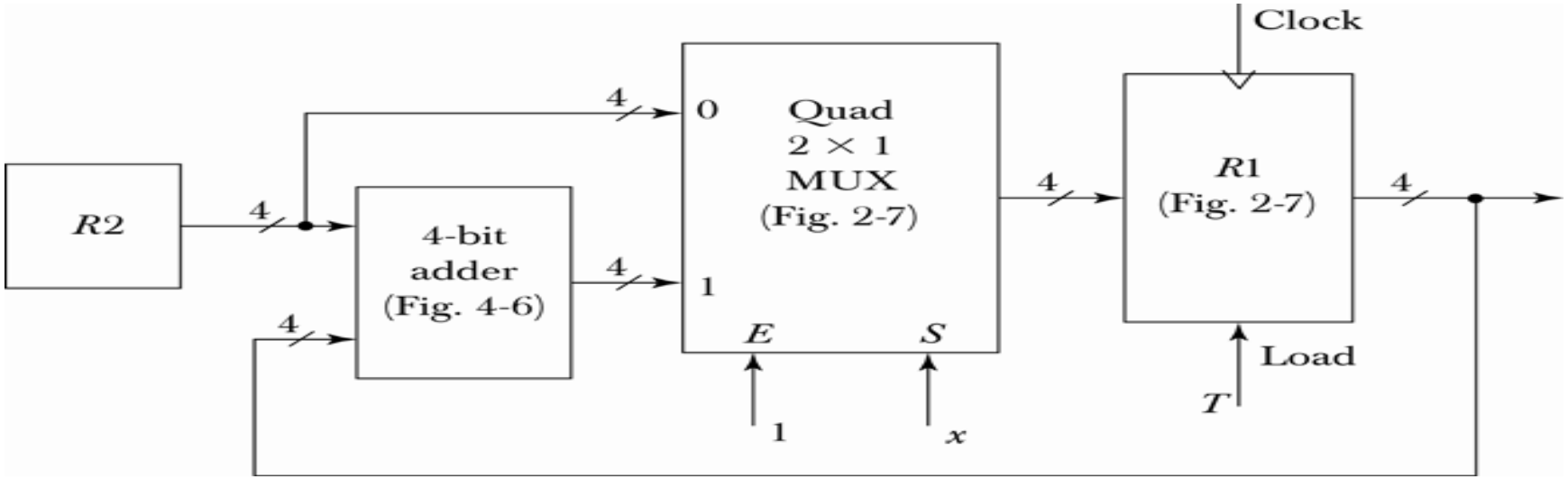
P'Q: R1 ← R3

Exercise 4

Draw the block diagram of the following statements:

xT: R1 ← R1+R2

x'T: R1 ← R2



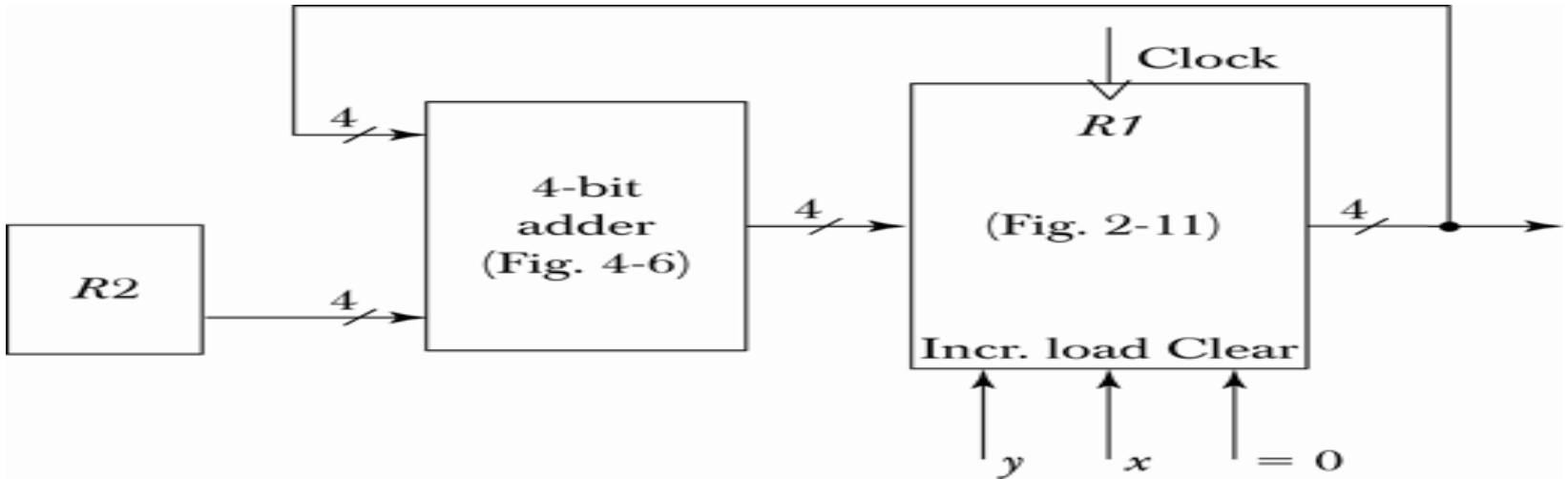
Exercise 5

Using a 4 bit counter with parallel load and a 4 bit adder draw block diagram that shows how to implement the following statements:

x: R1← R1+R2

x'y : R1 ← R1+1

Where R1 is a counter with parallel load and R2 is a 4 bit register



Exercise 6

What is wrong with the following statements?

a. **xT: AR ←$\overline{AR}$, AR ← 0**

b. **Yt: R1 ← R2, R1 ← R3**

c. **zT: PC ← AR, PC ← PC+1**

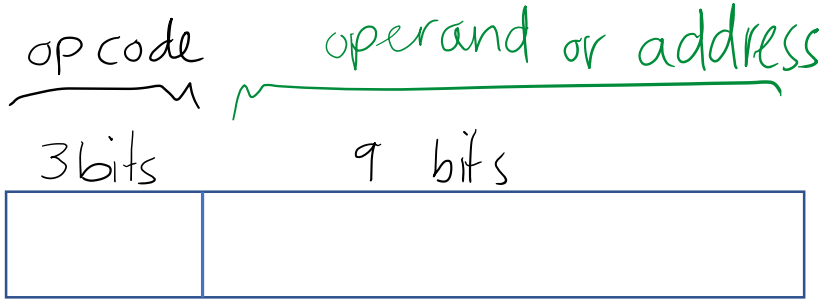
(a) Cannot complement and clear the same register at the same time.

(b) Cannot transfer two different values (R2 and R3) to the same register (R1) at the same time.

(c) Cannot transfer a new value into a register (PC) and increment the original value by one at the same time.

COMPUTER PROTOTYPE

- Every processor has an instruction set
- Each having a unique code (op code)

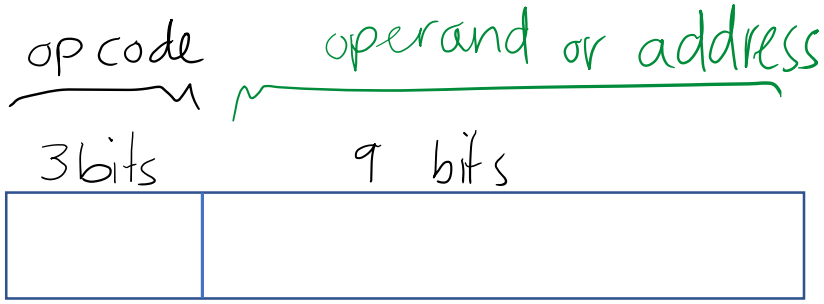


0 ADD
1 SUB
2 DIV
3 MULT
4 ADD

- Intel and IBM processors
- Machine Language
- High level language vs machine language
- Compiler

COMPUTER PROTOTYPE

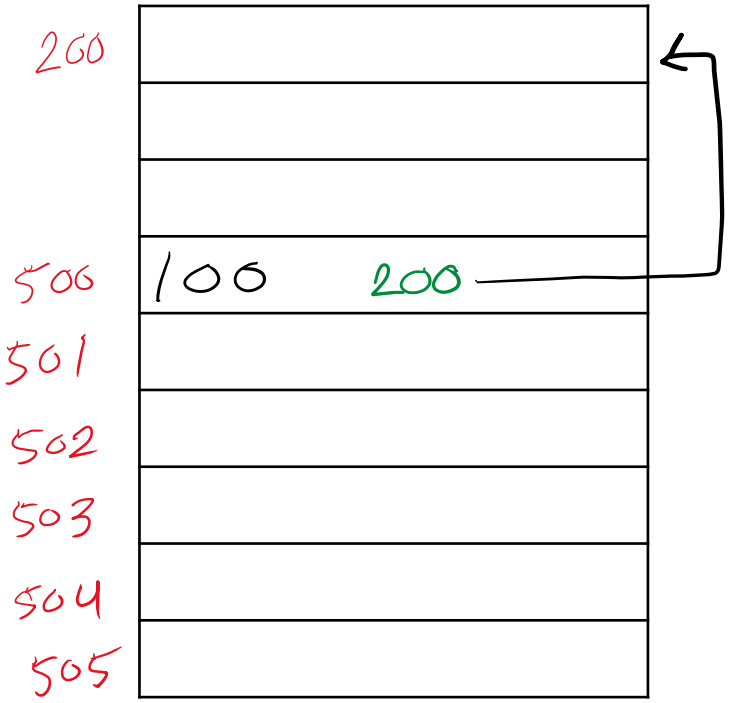
- Active programs are stored in RAM
- A program is a series of instructions
- Every processor has an instruction set
- Each having a unique code (op code)



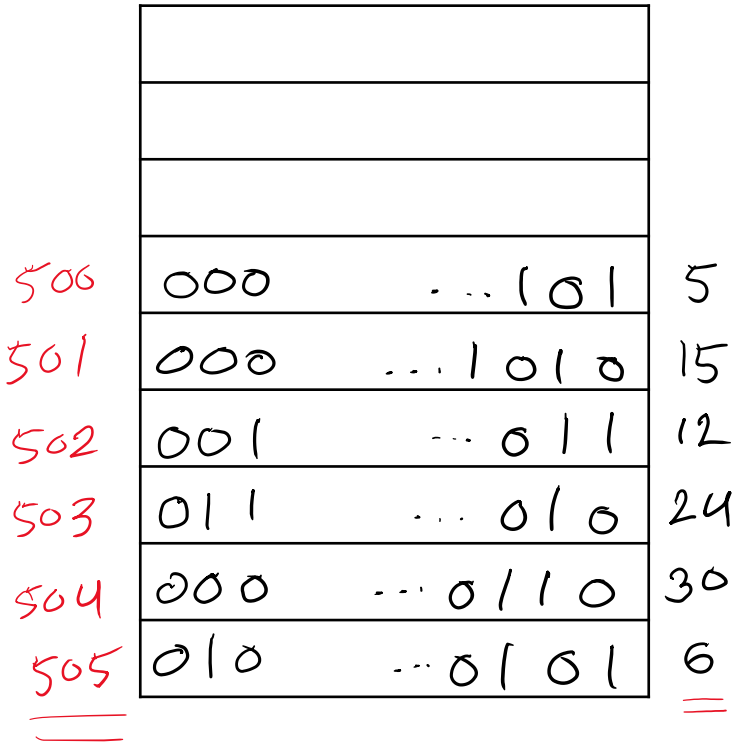
$2^9 = 512$

0 ADD
1 SUB
2 DIV
3 MULT
4 ADD

COMPUTER PROTOTYPE



0 ADD
1 SUB
2 DIV
3 MULT
4 ADD



What if I open another program, and come back to this one?

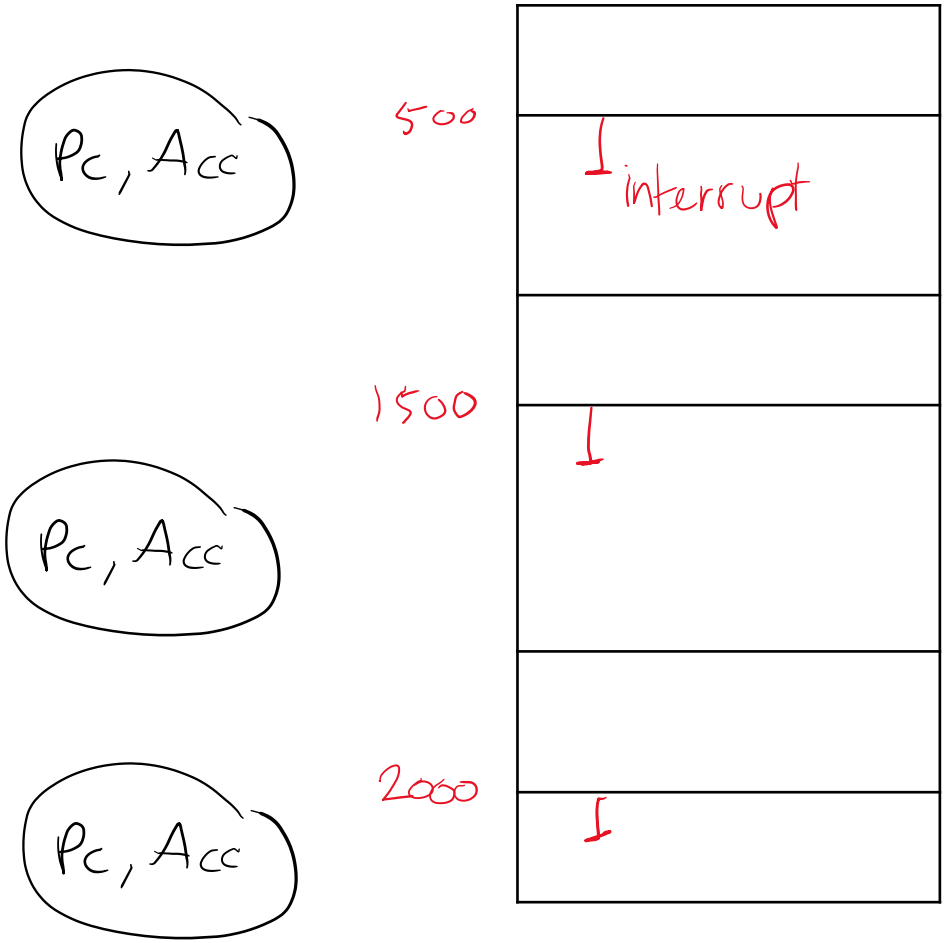
COMPUTER PROTOTYPE

Who places the programs in RAM?

- Begin and end address
- Accumulator
- PC = entry point (main)

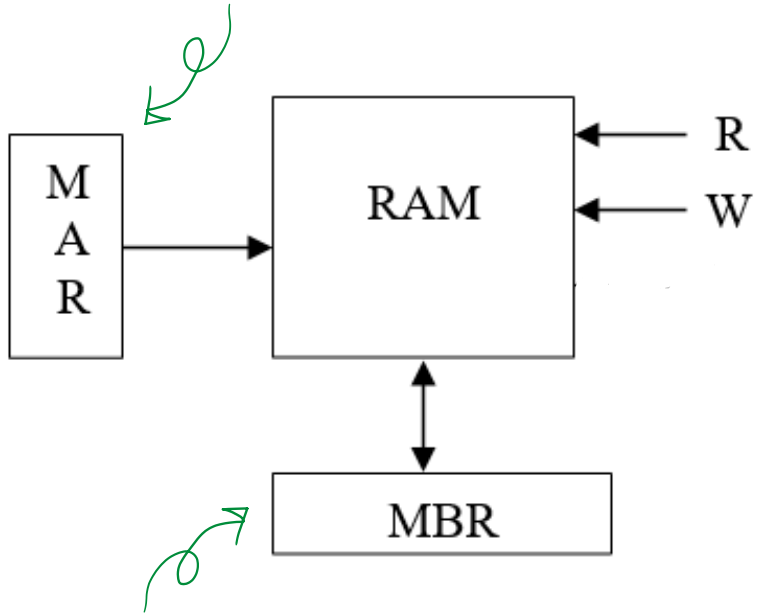
} Process control block

Interrupt every 1/10000



COMPUTER PROTOTYPE

Stores address of instruction to be executed or data to be read



Stores data coming out or going in to memory

Memory Address Register
Memory Buffer Register

Instructions are executed synchronously
T0 : put address
T1: send read signal
T2: take data

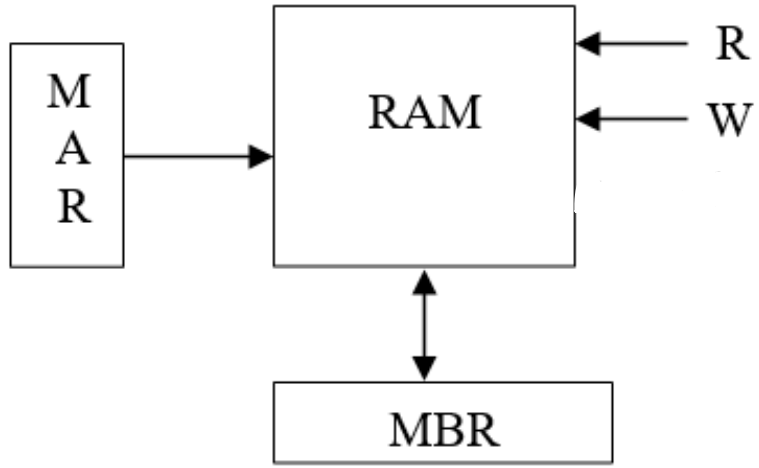
COMPUTER PROTOTYPE

- Read location 500 into Acc

T0: Put address in MAR

T1: Send read signal

T2: Take data from MBR to Acc



- Write x=5

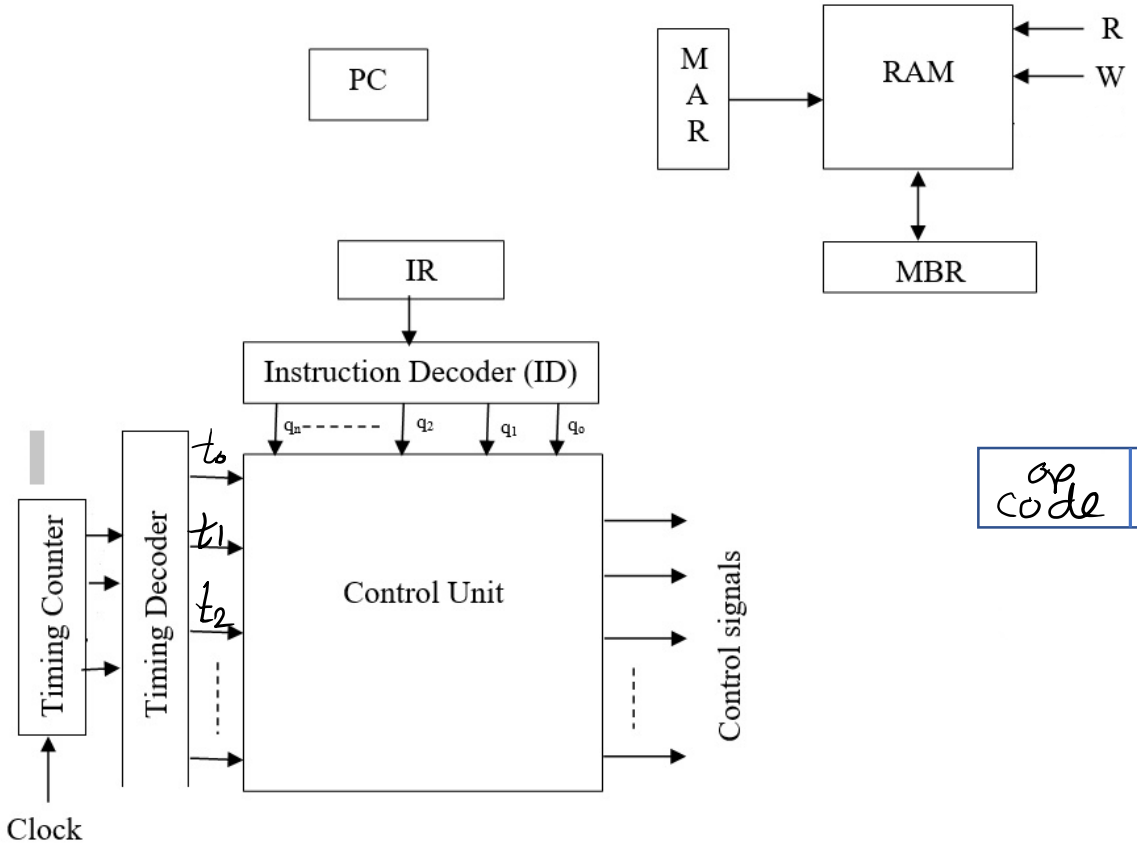
T0: Put address in MAR

T1:Put data to write in MBR

T2: Send write signal

COMPUTER PROTOTYPE

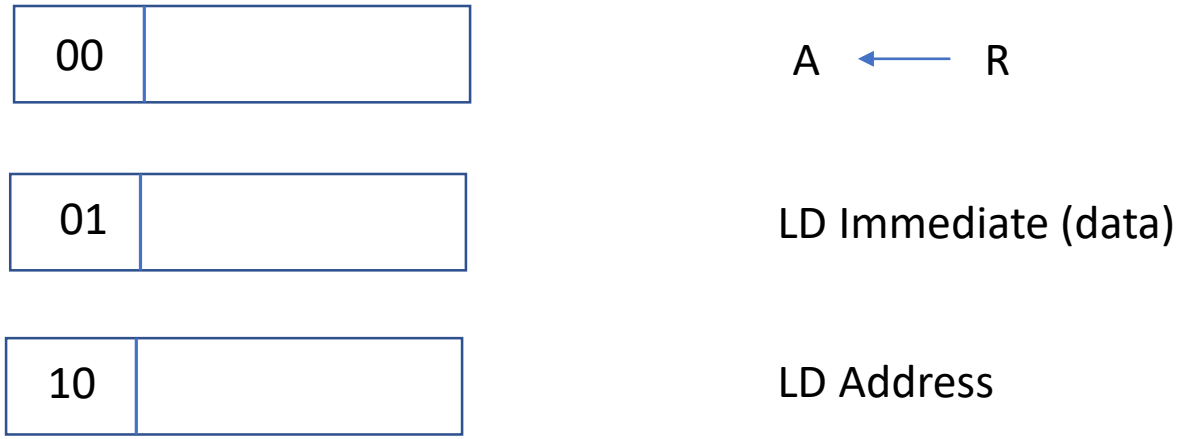
- Fetch
- Decode
- Execute



Address bus
Data bus
Control bus

COMPUTER PROTOTYPE

- Design a processor that can execute 3 instructions



A ← R

LD Immediate (data)

LD Address

COMPUTER PROTOTYPE

- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

Q0 T3: A ← R,
T ← 0

Q1 T3: A ← MBR [OPERAND],
T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

Q2 T5: A ← MBR
T ← 0

COMPUTER PROTOTYPE

- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

Q0 T3: A ← R,
T ← 0

Q1 T3: A ← MBR [OPERAND],
T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

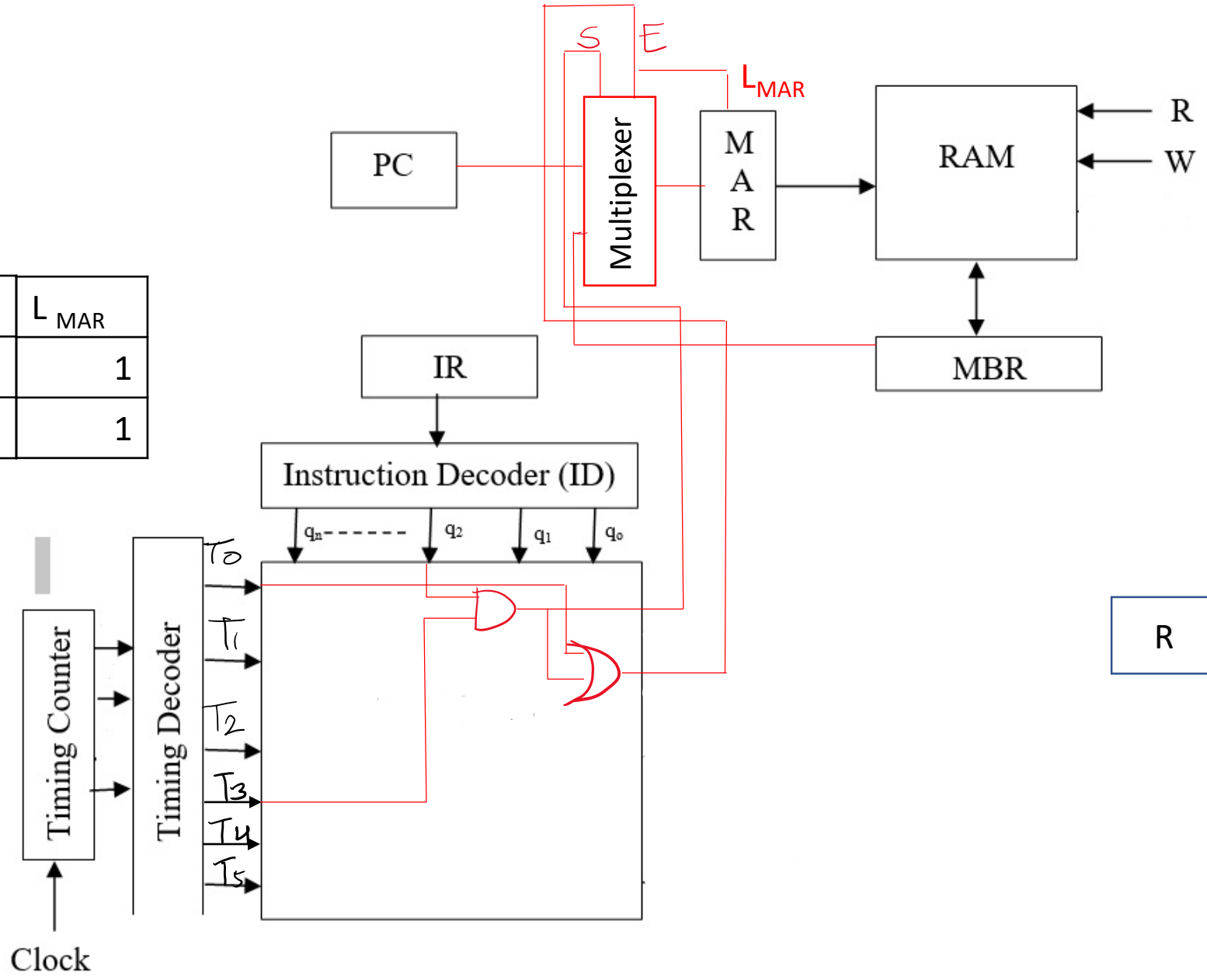
Q2 T5: A ← MBR
T ← 0

T0	Q2 T3	S	E	L_{MAR}
1	X	0	1	1
X	1	1	1	1

$S = Q2\ T3$

$E = Q2\ T3 + T0$

$L_{mar} = Q2T3 + T0$



R

A

- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

Q0 T3: A ← R,
T ← 0

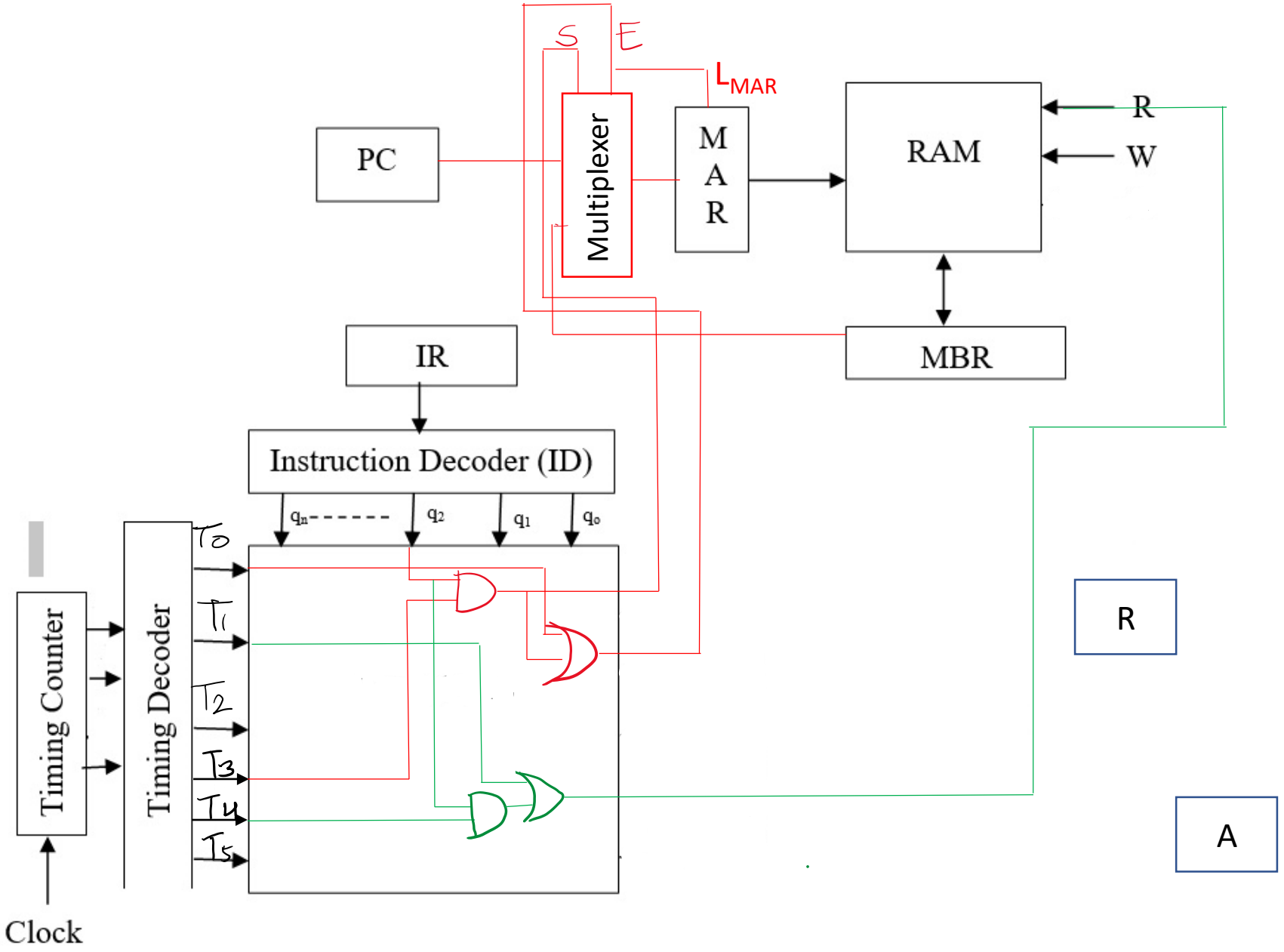
Q1 T3: A ← MBR [OPERAND],
T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

Q2 T5: A ← MBR
T ← 0

R= T1 + Q2 T4


S
E
L_{MAR}
PC
Multiplexer
M
A
R
RAM
R
W
MBR
IR
Instruction Decoder (ID)
q_n ------ q_2 q_1 q_0
T_0
T_1
T_2
T_3
T_4
T_5
Timing Counter
Timing Decoder
Clock
R
A

- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

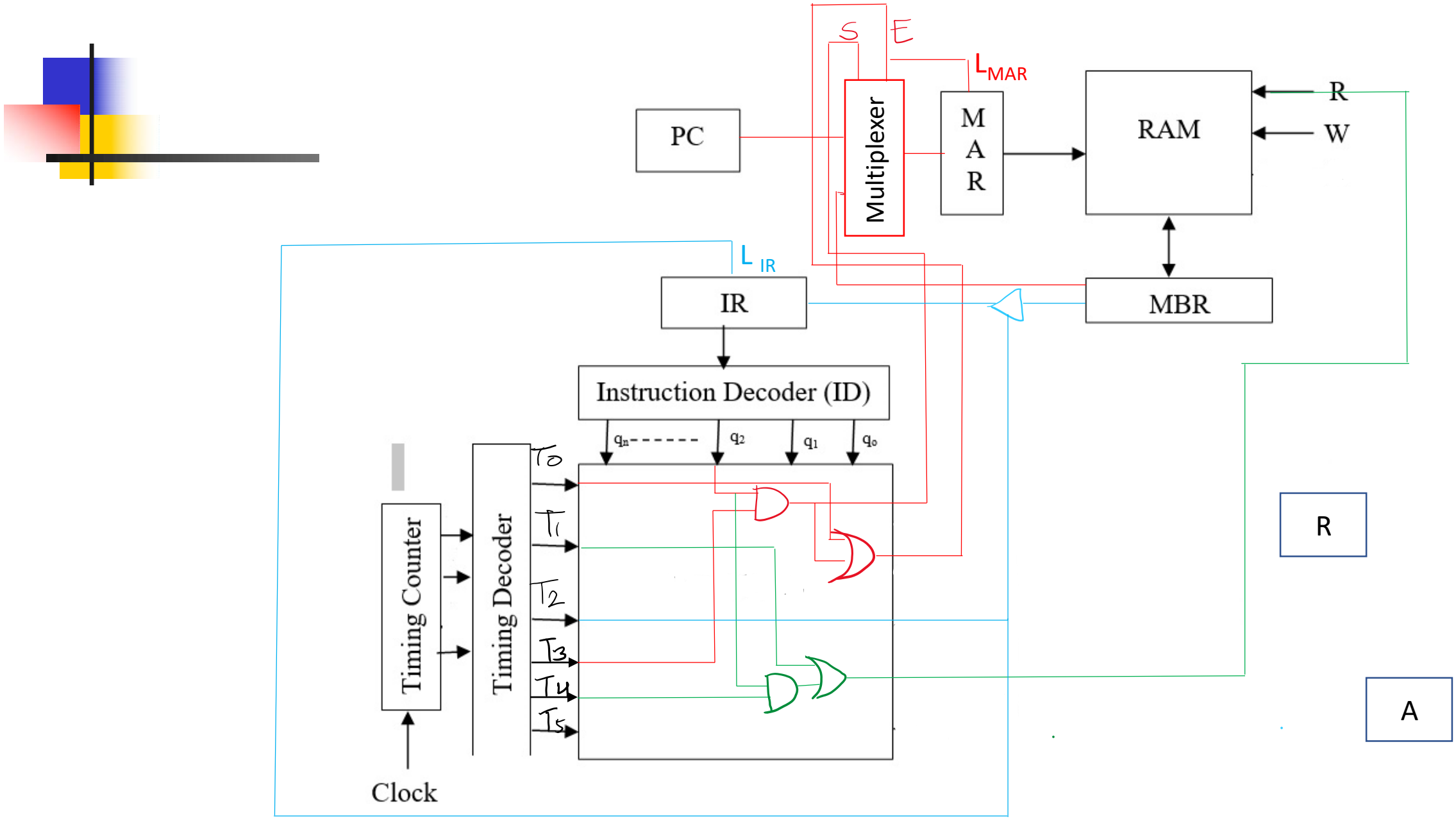
Q0 T3: A ← R,
T ← 0

Q1 T3: A ← MBR [OPERAND],
T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

Q2 T5: A ← MBR
T ← 0


S
E
L MAR
PC
Multiplexer
M
A
R
RAM
R
W
L IR
IR
MBR
Instruction Decoder (ID)
qn
q2
q1
q0
T0
T1
T2
T3
T4
T5
Timing Counter
Timing Decoder
Clock
R
A

- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

Q0 T3: A ← R,

T ← 0

Q1 T3: A ← MBR [OPERAND],

T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

Q2 T5: A ← MBR

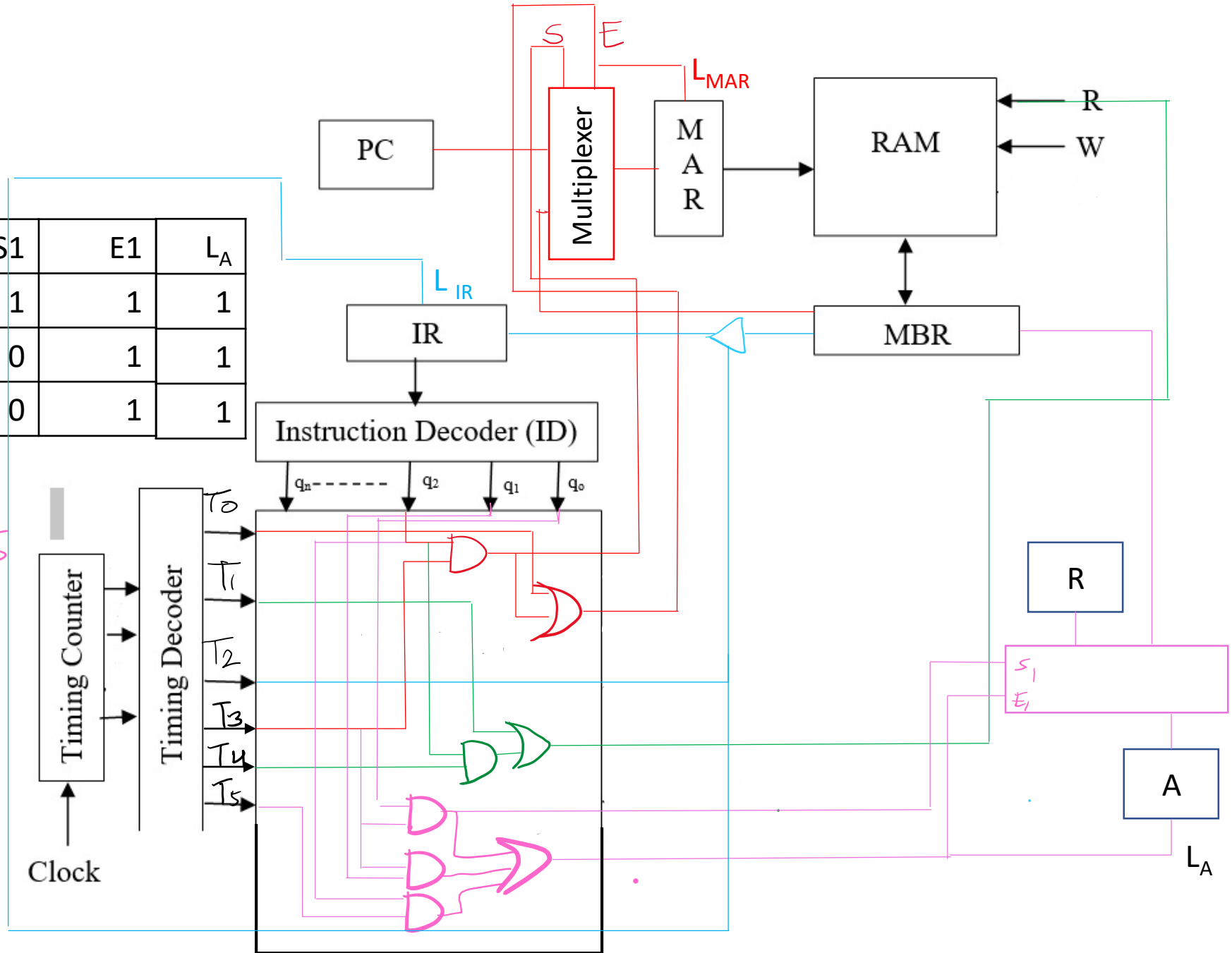
T ← 0

Q0 T3	Q1 T3	Q2 T5	S1	E1	L_A
1	X	X	1	1	1
X	1	X	0	1	1
X	X	1	0	1	1

$S_1 = Q0\,T3$

$E_1 = Q0T3 + Q1T3 + Q2\,T5$

$L_A =$ // // //



- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

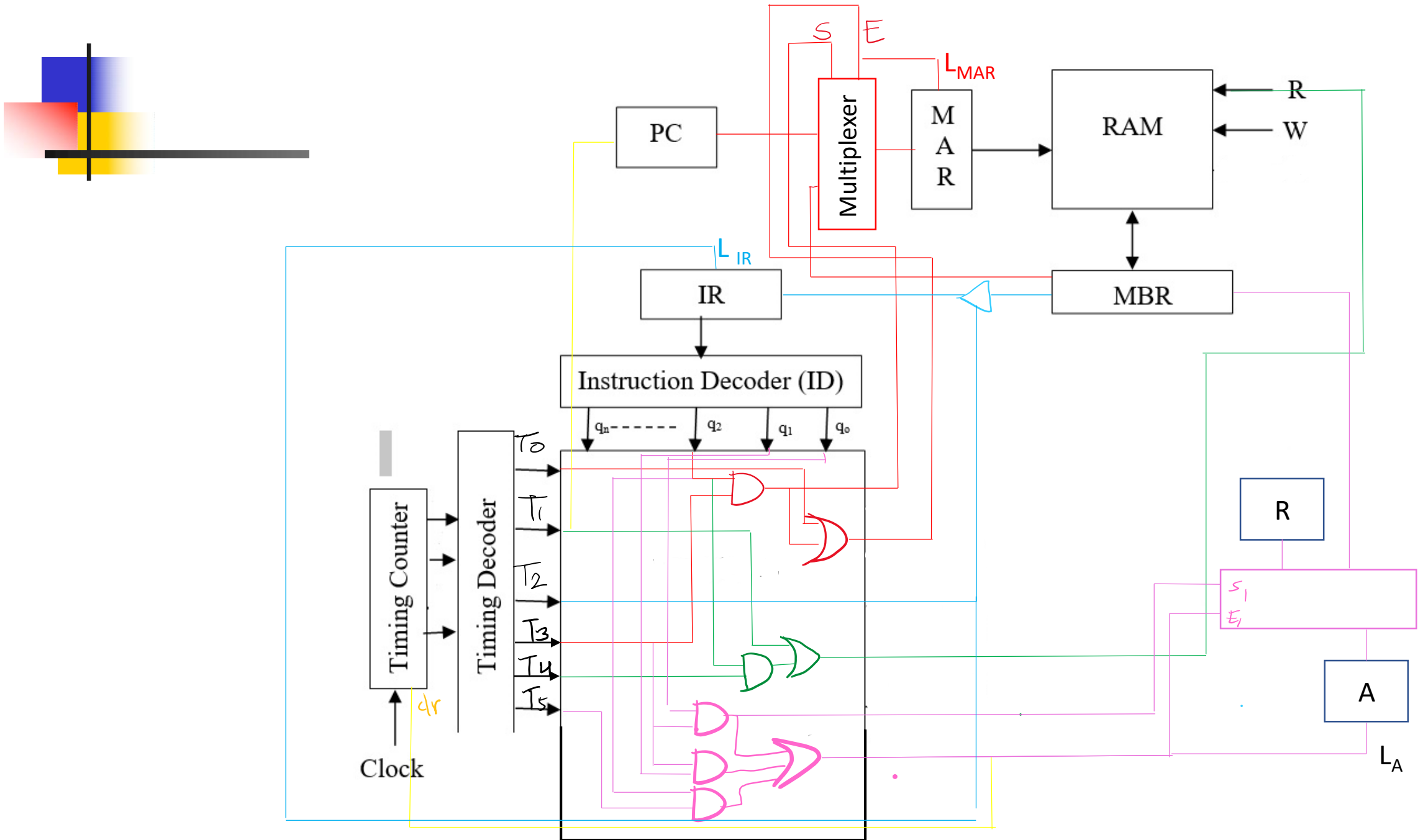
Q0 T3: A ← R,
T ← 0

Q1 T3: A ← MBR [OPERAND],
T ← 0

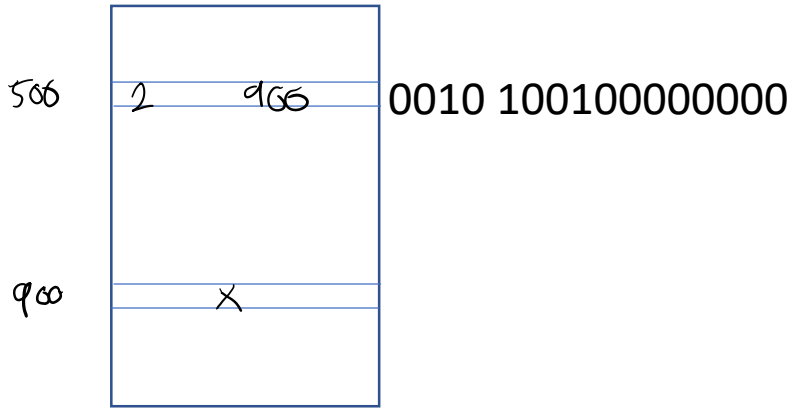
Q2 T3: MAR ← MBR [ADDRESS]

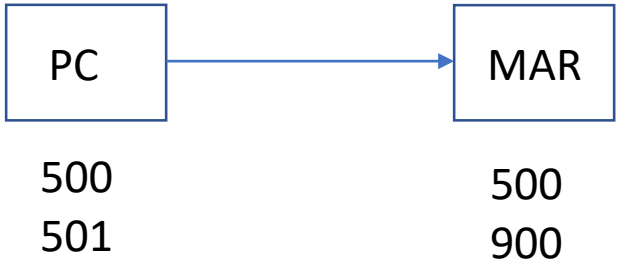
Q2 T4: MBR ← M

Q2 T5: A ← MBR
T ← 0


S
E
L_{MAR}
PC
Multiplexer
M
A
R
RAM
R
W
L_{IR}
IR
MBR
Instruction Decoder (ID)
q_n ------ q_2 q_1 q_0
T0
T1
T2
T3
T4
T5
Timing Counter
Timing Decoder
Clock
R
S_1
E_1
A
L_A

- PC and MAR both hold addresses, why not only use one?
- Ex LD X





High level language	Assembly language

If (x == 5)
  y = y + 1
else
  y= y - 1
	LD X
	Sub 5
	JNZ else (jump if not zero)
	LD y
	Add 1
	Store y
	JP end
	else LD y
	Sub 1
	Store y
	end

High level language	Assembly language

If (x == 5)
  y = y + 1
else
  y= y - 1
	0 LD X
	1 Sub 5
	2 JNZ else 7 (jump if not zero)
	3 LD y
	4 Add 1
	5 Store y
	6 JP end A
	else 7 LD y
	8 Sub 1
	9 Store y
	end A

x + +	LD x Add 1 Store x

Assembly language	Op code	Machine Language	
0 LD X	0	0 800	0000 1000 0000 0000
1 Sub 5	1	1 005	0001 0000 0000 0101
2 JNZ 7	2	2 007	0010 0000 0000 0111
3 LD y	0	0 801	
4 Add 1	3	3 001	
5 Store y	4	4 801	
6 JP A	5	5 00A	
7 LD y	0	0 801	
8 Sub 1	1	1 001	
9 Store y	4	4 801	
A			

RAM

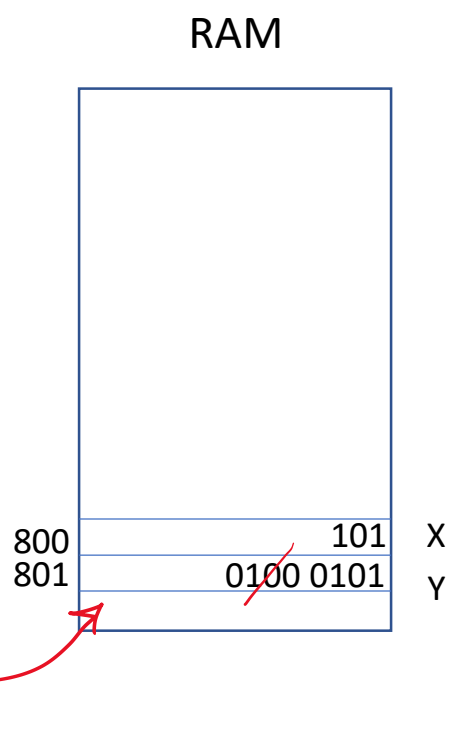
101	X
0100	Y

Assembly language
0 LD X
1 Sub 5
2 JNZ 7
3 LD y
4 Add 1
5 Store y
6 JP A
7 LD y
8 Sub 1
9 Store y
A

Op code
0
1
2
0
3
4
5
0
1
4

Machine language
0 800
1 005
2 007
0 801
3 001
4 801
5 00A
0 801
1 001
4 801

PC	Acc	IR
0000	0000	----
0001	0005	0800
0002	0000	1005
0003	0000	2007
0004	0100	0801
0005	0101	3001
0006	0101	4801
0007 A	0101	500A
-		
-		
-		
A		



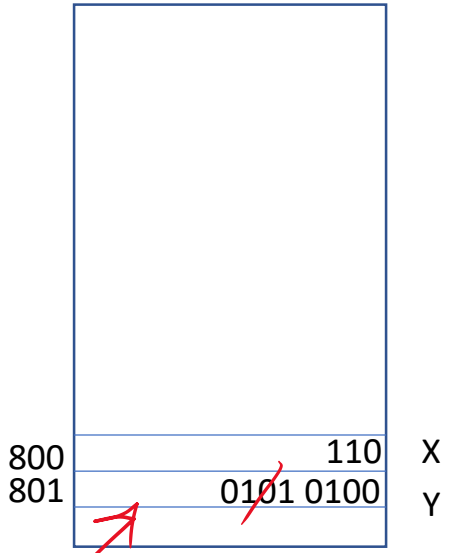
Assembly language
0 LD X
1 Sub 5
2 JNZ 7
3 LD y
4 Add 1
5 Store y
6 JP A
7 LD y
8 Sub 1
9 Store y
A

Op code
0
1
2
0
3
4
5
0
1
4

Machine language
0 800
1 005
2 007
0 801
3 001
4 801
5 00A
0 801
1 001
4 801

PC	Acc	IR
0000	0000	----
0001	0006	0800
0002	0001	1005
0003	0001	2007
-		
-		
-		
-		
0008	0005	0801
0009	0004	1001
000A	0004	4801

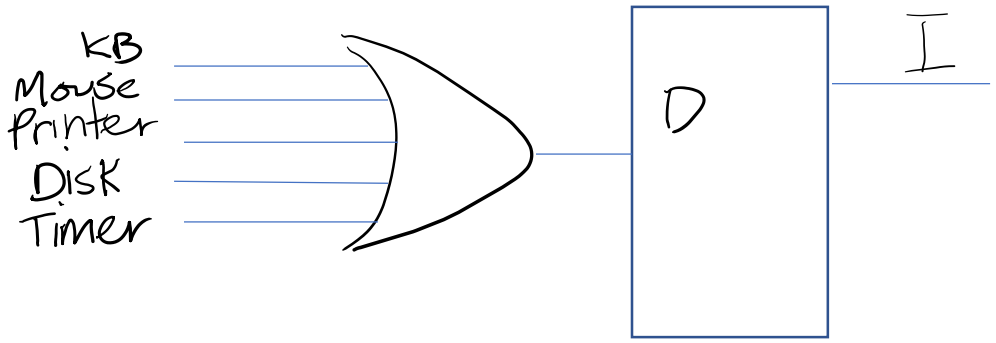
RAM



Interrupts

- We want the computer to respond to interrupts
- Button press
- Mouse movement
- Program icon
- Printer interrupt

These interrupts need to be handled.



- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

Q0 T3: A ← R,
T ← 0

$\bar{I}$Q1 T3: A ← MBR [OPERAND],
T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

Q2 T5: A ← MBR
T ← 0

- Micro Program

T0: MAR ← PC

T1: MBR ← M, PC ← PC+1

T2: IR ← MBR [OP CODE]

Q0 T3: A ← R,
T ← 0

I Q1 T3: A ← MBR [OPERAND],
save state (PC + A)
LD PC with I Handler
T ← 0

Q2 T3: MAR ← MBR [ADDRESS]

Q2 T4: MBR ← M

Q2 T5: A ← MBR
T ← 0

- Hardware instructions are not interruptible,

But software instructions are.

LD x

Add 1

Store x