CSE404

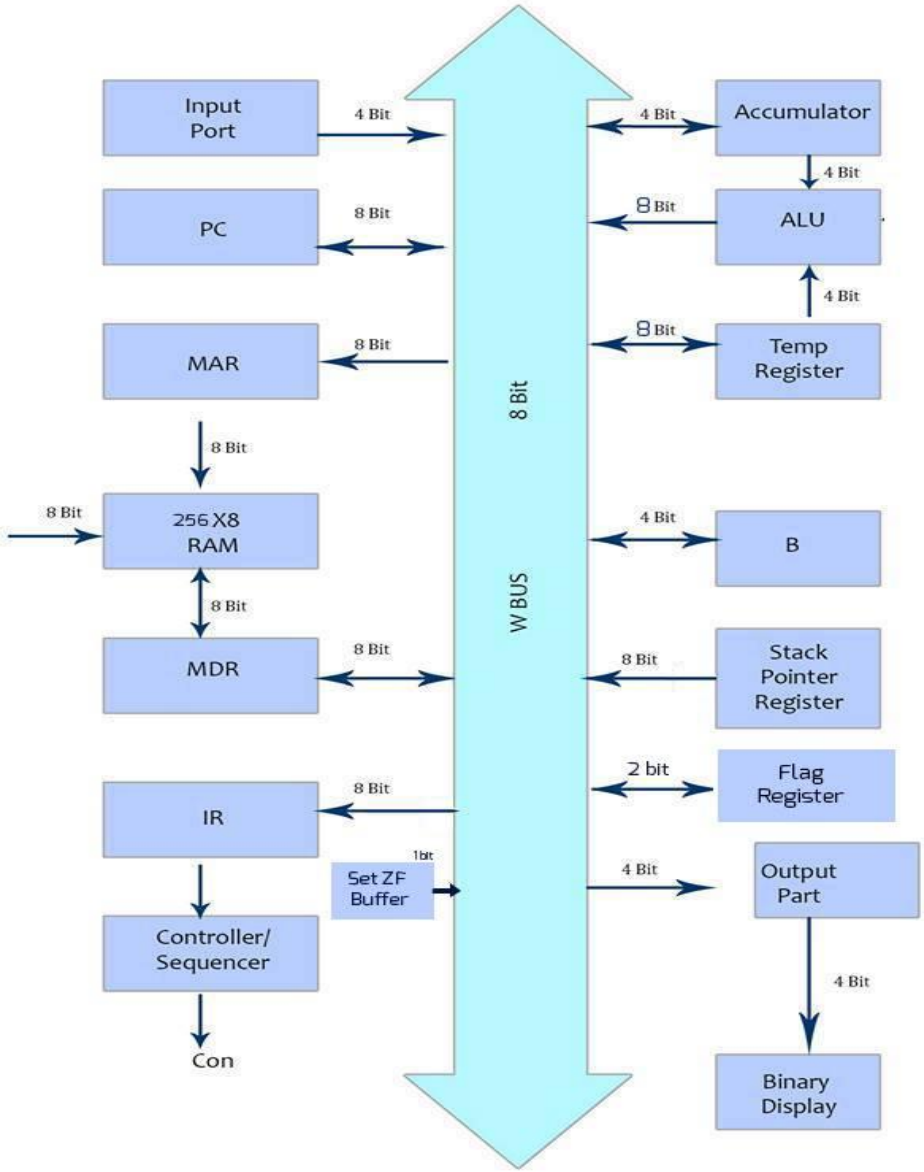
Digital Systems Design Sessional

Design of 4-bit SAP

Section	B1
Group No.	03
Writer's Roll	1205016
Writer's Name	Rashid Abid Rafi
Group Members	1205016 1205026 1205028 1205034 1205052
Date of Submission:	15 November 2016

General Discussion & BLOCK DIAGRAM:

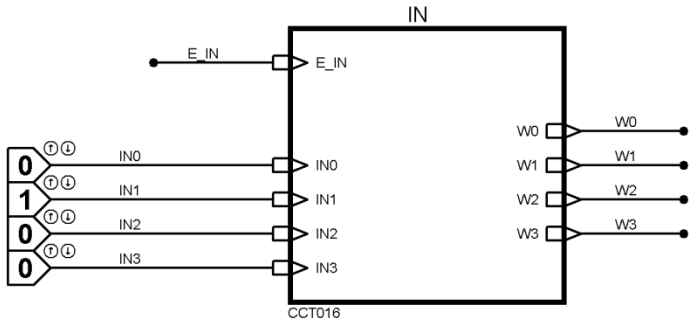
- Address BUS 8bit
- DATA bus 4 bit
- All data loading from BUS are sequential (L_{sth}) and all data emitting to BUS are combinational (E_{sth}) (e.g: sth= MAR,MDR,IR etc.)
- The machine is edge triggered and both falling edge and rising edge are used in specific cases to perform specific purposes to avoid race conditions.


Input Port
4 Bit
PC
8 Bit
MAR
8 Bit
8 Bit
8 Bit
256 X8 RAM
8 Bit
MDR
8 Bit
IR
8 Bit
Controller/ Sequencer
Con
1bit
Set ZF Buffer
8 Bit
W BUS
4 Bit
Accumulator
4 Bit
8 Bit
ALU
4 Bit
8 Bit
Temp Register
4 Bit
B
8 Bit
Stack Pointer Register
2 bit
Flag Register
4 Bit
Output Part
4 Bit
Binary Display

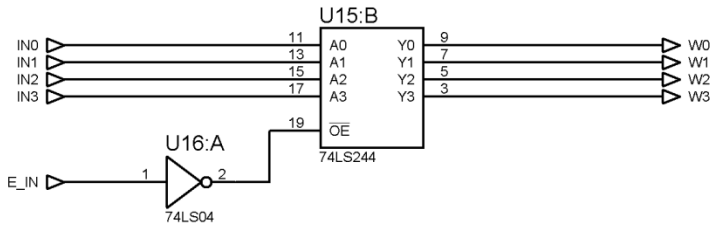
Input Register:

It is unidirectional with the bus: it only gives its data to the bus. That's why only E_{IN}. The task of input register is to take user input and pass the data to Accumulator via bus.

It is a data register, so only 4 bit transaction will suffice.



INPUT REGISTER



INPUT REGISTER CHILD

PROGRAM COUNTER (PC):

PC is bidirectional with the bus. L_{PC} loads data from the bus in a clock cycle and E_{PC} lets it emit his data to the bus.

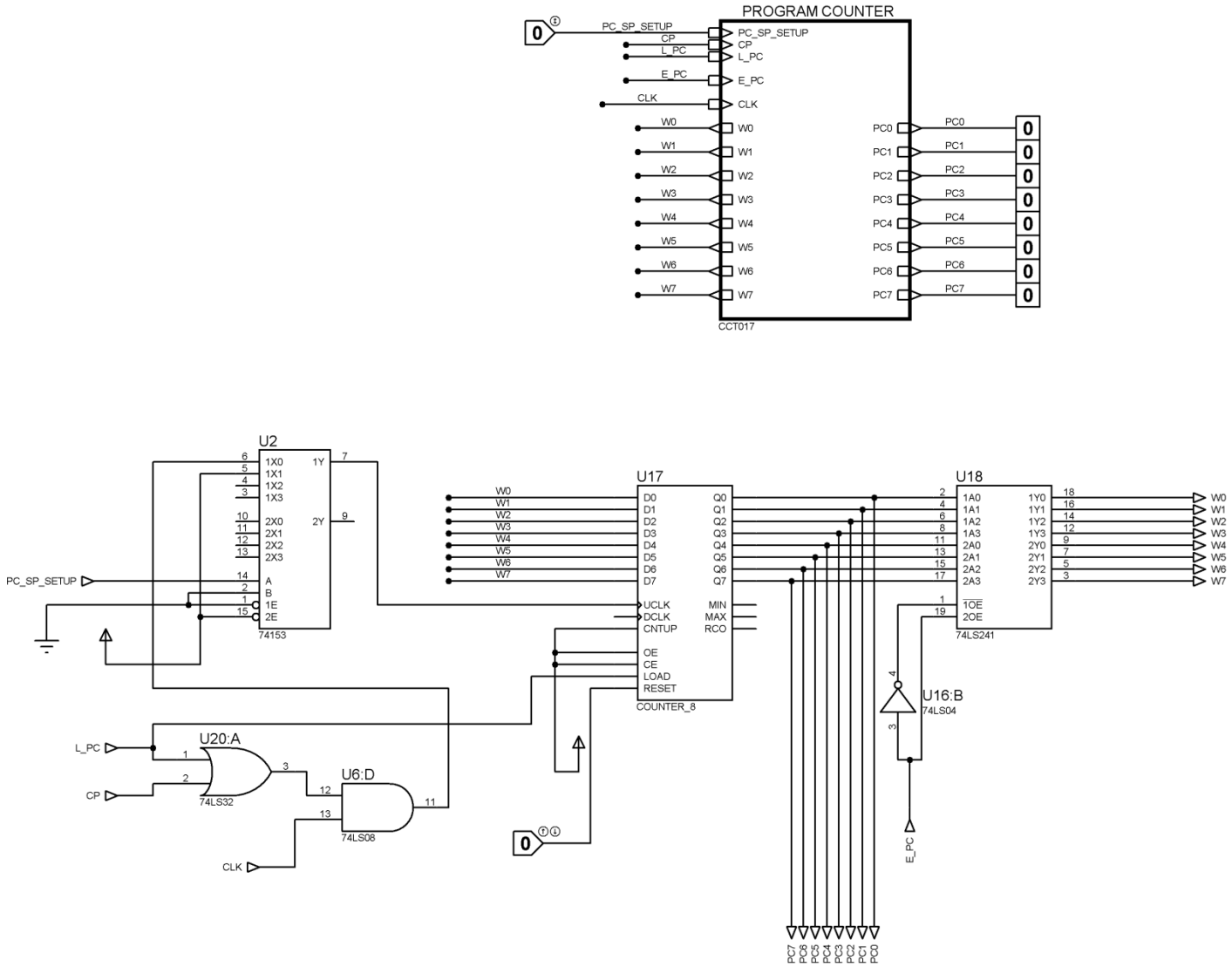
PC is an address register. That's why it performs transactions in 8 bit.

There is a special bit PC_SP_SETUP in the design. There is a reason it is there. Eventually, when we will discover our RAM, we will see that the RAM stores data from the 1^{st} address instead of 0^{th} address. For this reason, PC is incremented by 1 forcefully at the beginning of running any program.

So, the incrementing needs a choice:

i) PC gets incremented by one at the beginning of the program by turning PC_SP_SETUP 1 forcefully.
ii) PC gets incremented when C_P is 1 at the time of program running.

The MUX in this design selects between these two choices.

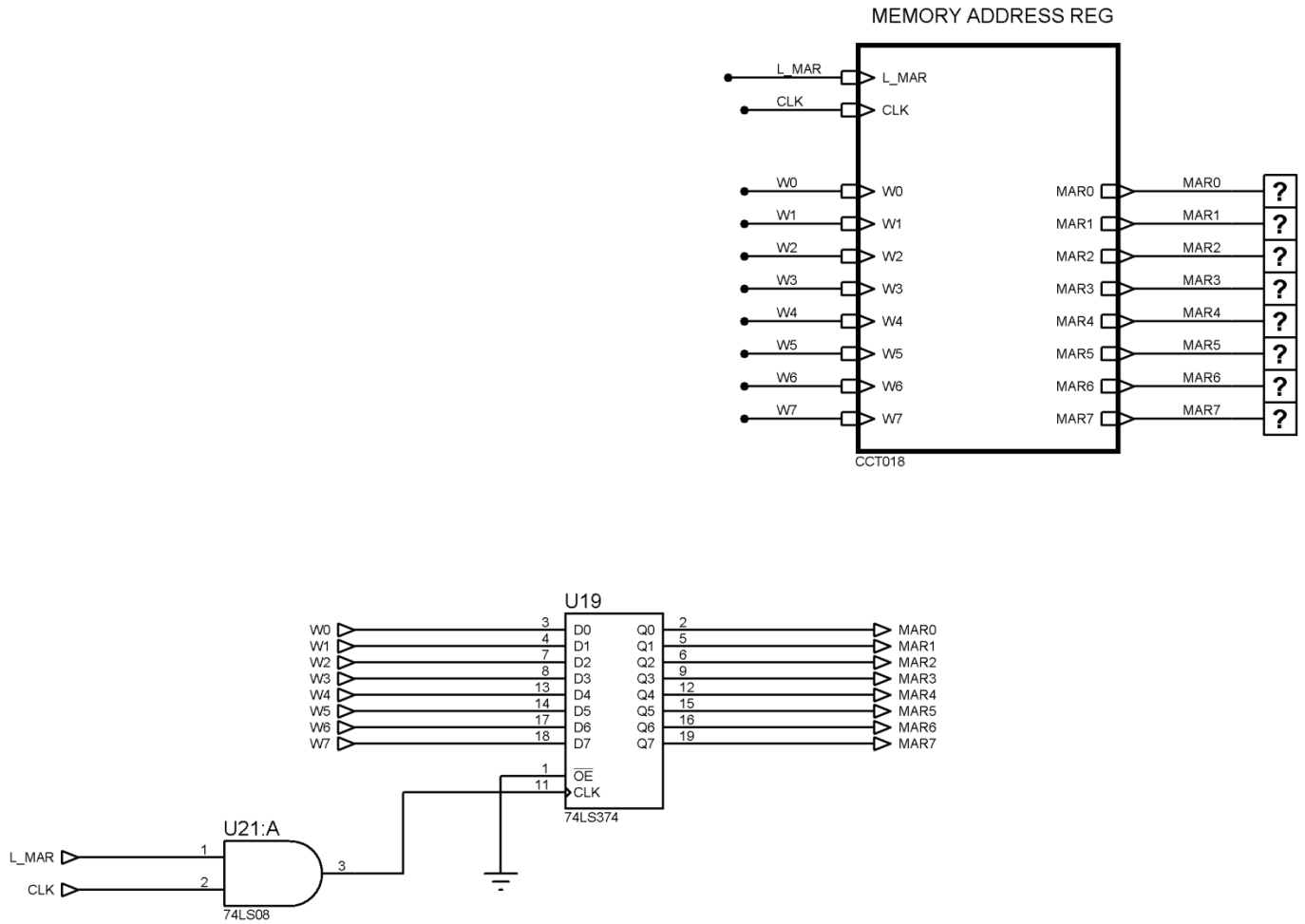

PROGRAM COUNTER
PC_SP_SETUP
CP
L_PC
E_PC
CLK
W0
W1
W2
W3
W4
W5
W6
W7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
CCT017
U2
74153
U17
COUNTER_8
U18
74LS241
U16:B
74LS04
U20:A
74LS32
U6:D
74LS08

MAR (Memory Address Register):

It is unidirectional with the bus: it only loads data from the bus. That's why only L_{MAR}.

The part of MAR with RAM is combinational. That means, whenever there is a data in MAR, RAM points to that address in the same clock cycle.

It is also an address register, hence 8 bit transaction.


MEMORY ADDRESS REG
L_MAR
CLK
W0
W1
W2
W3
W4
W5
W6
W7
MAR0
MAR1
MAR2
MAR3
MAR4
MAR5
MAR6
MAR7
CCT018
U19
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
OE
CLK
74LS374
U21:A
74LS08

RAM (Random Access Memory):

Our address bit is 8 bit. So, we need to implement a 256x8 RAM.

There is no connection of BUS with RAM directly. In fact, the design was specific about this not to have any direct connection with the BUS. For this reason, the two registers: MAR & MDR are used.

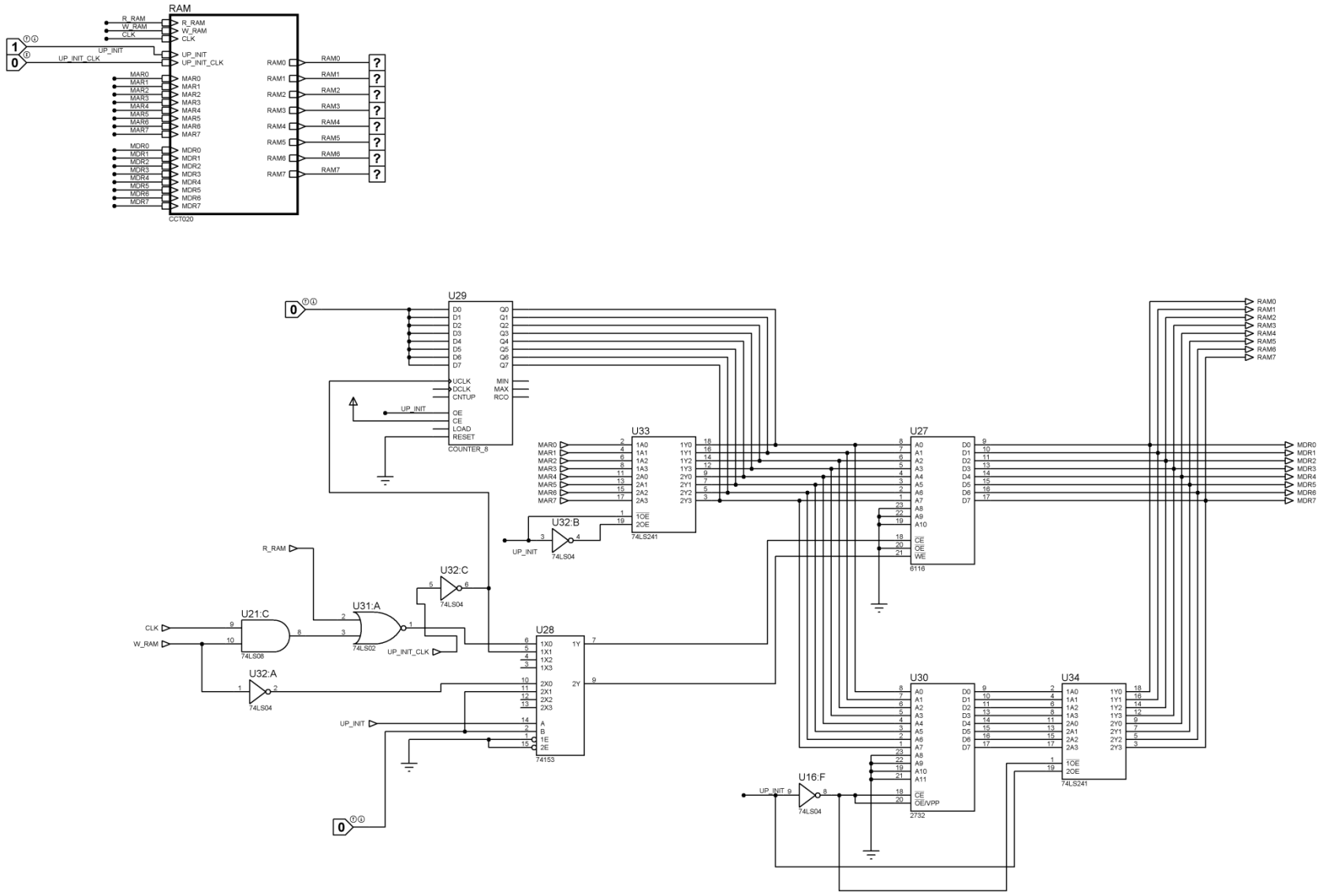
The whole design constitutes of two parts:

i) Uploader/ Boot-Loader : When Up_INIT is on, with every clock pulse at UP_INIT_CLK, the assembly program, which the PC is about to run, pre-saved in the EPROM in the form of Machine Language gets loaded byte by byte and written into RAM.

ii) Normal Read/Write Operation: Only after the whole assembly program is loaded into RAM, typical read/write operations starts according to the CLK and R_RAM, W_RAM bit.

Functionality:

- Read Operation: When there is CE(Chip Enable) on and WE(Write Enable) off, provided that OE(output Enable) always remains on, a read operation happens. It reads the data from the address A0-A7 and outputs data into the D0-D7 pins.
- Write Operation: When there is CE(Chip Enable) on and WE(Write Enable) on, provided that OE(output Enable) always remains on, a write operation happens. It writes the data from the data pins D0-D7 into the address specified in A0-A7 pins.
- MUX: The ram operates to serve two purposes. One is for boot-loading the program and the other is the normal read/write operation. For this choice making, a MUX is used along with proper combinational logic to control the RAM.


RAM
R_RAM
W_RAM
CLK
UP_INIT
UP_INIT_CLK
MAR0
MAR1
MAR2
MAR3
MAR4
MAR5
MAR6
MAR7
MDR0
MDR1
MDR2
MDR3
MDR4
MDR5
MDR6
MDR7
RAM0
RAM1
RAM2
RAM3
RAM4
RAM5
RAM6
RAM7
CCT020
U29
COUNTER_8
U33
74LS241
U27
6116
U32:B
74LS04
U32:C
U21:C
74LS08
U31:A
74LS02
U28
74153
U32:A
U30
2732
U34
U16:F

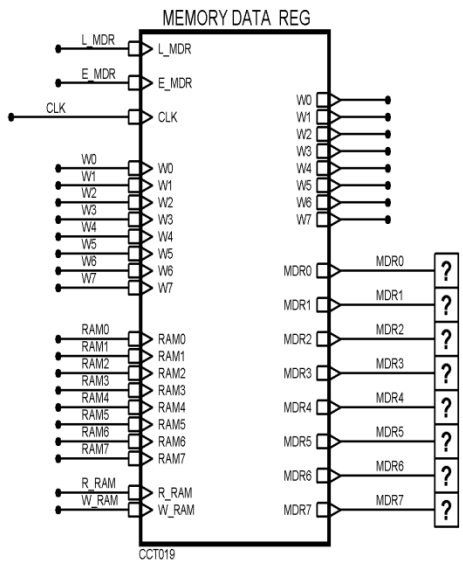
MDR (Memory Data Register):

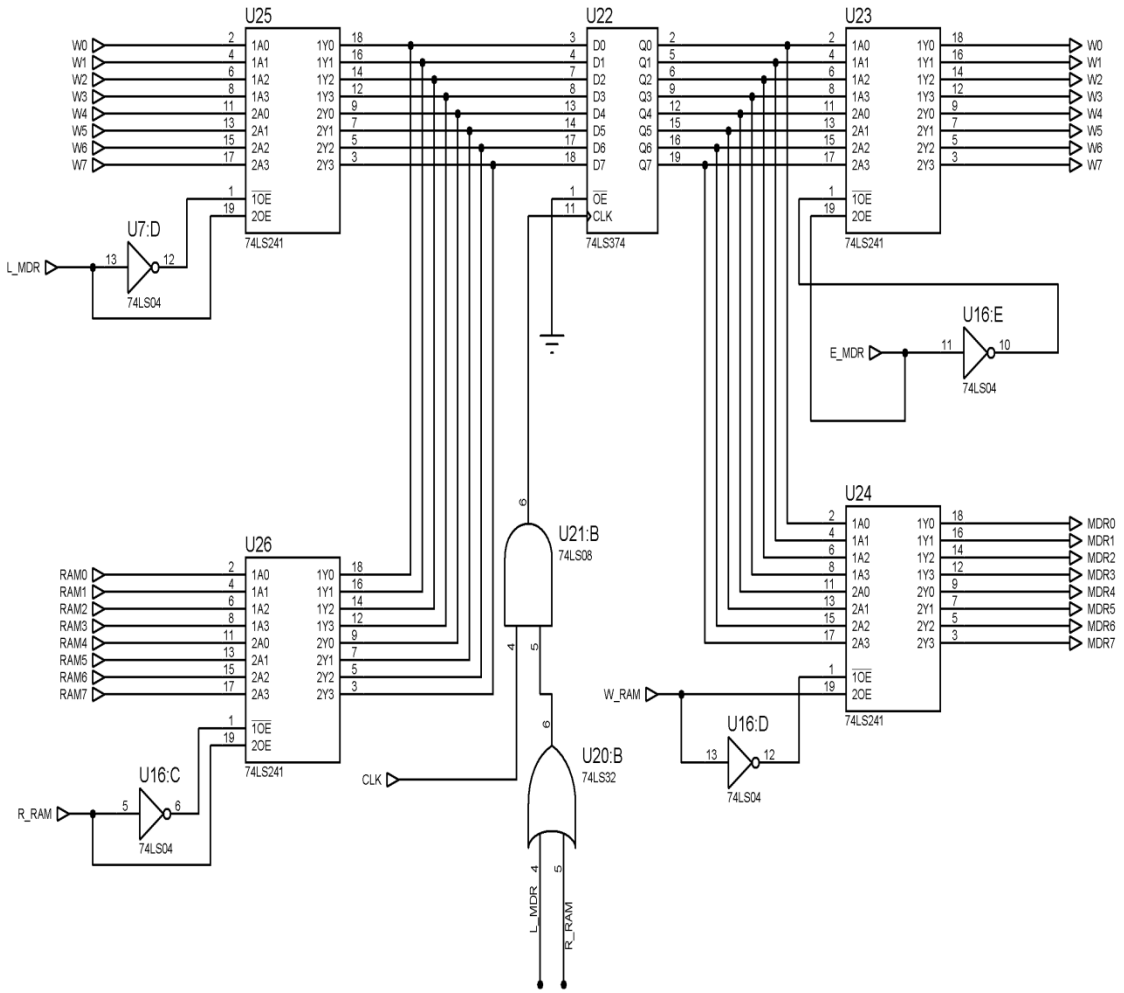
MDR is bidirectional with the bus. That's why L_{MDR} & $E_{MDR.}$

It can contain 4 bit data or 8 bit address as a data, too. That's why it is 8 bit register.

MDR is bidirectional with RAM, too.

i) When R_RAM is on, MDR reads data from RAM and stores within. It needs a clock cycle.
ii) When W_RAM is on, MDR writes its data at RAM at the address pointed by MAR. It needs a clock cycle.


MEMORY DATA REG
L_MDR
E_MDR
CLK
W0
W1
W2
W3
W4
W5
W6
W7
RAM0
RAM1
RAM2
RAM3
RAM4
RAM5
RAM6
RAM7
R_RAM
W_RAM
MDR0
MDR1
MDR2
MDR3
MDR4
MDR5
MDR6
MDR7
CCT019

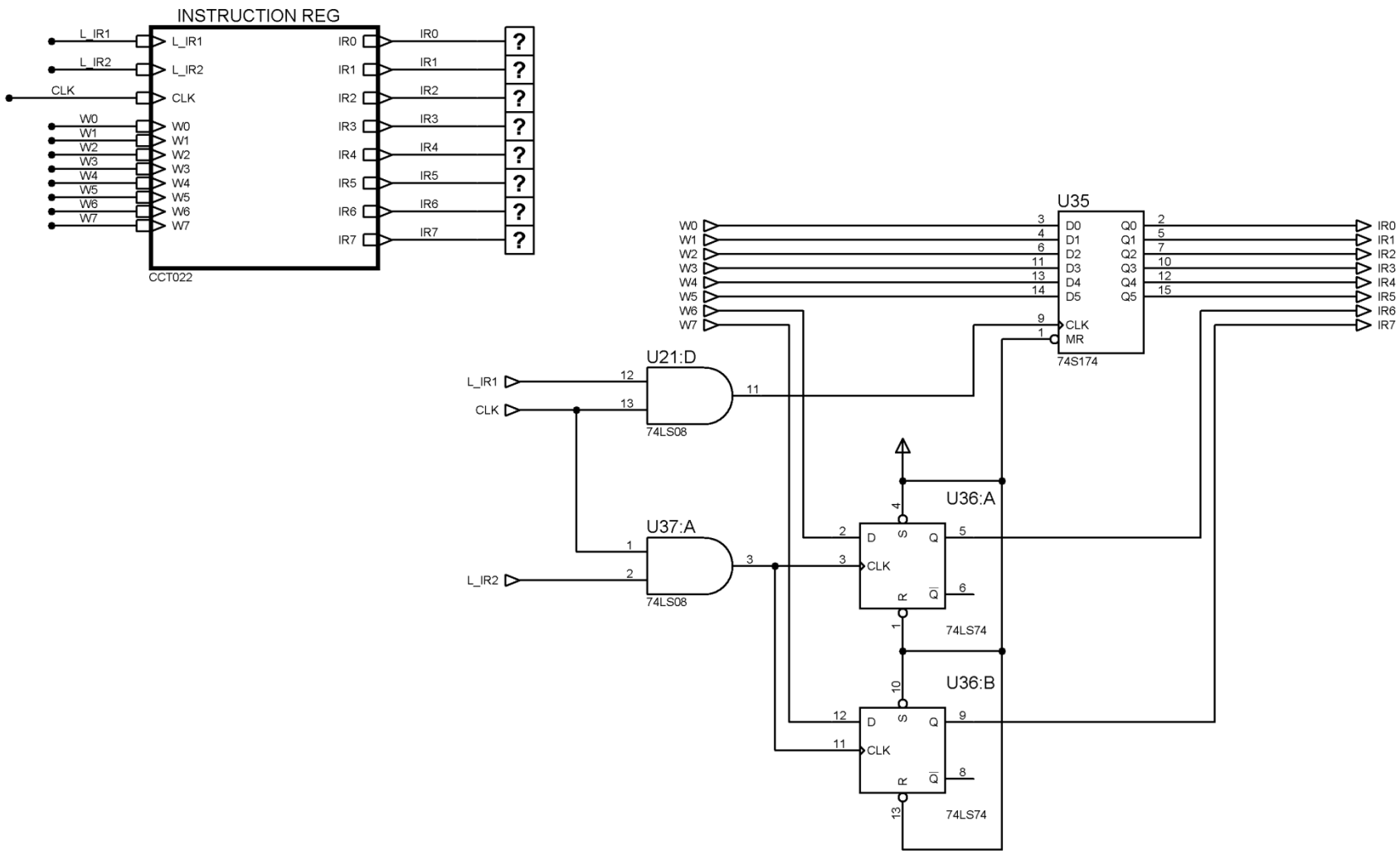

U25
U22
U23
U24
U26
U7:D
U16:E
U16:C
U16:D
U21:B
U20:B
74LS241
74LS374
74LS04
74LS08
74LS32
L_MDR
E_MDR
R_RAM
W_RAM
CLK

IR (Instruction Register) :

It is unidirectional with the bus. It only loads data from bus.

i) L_IR1 loads 6 bit from bus. The first 5 bit(bit0-bit4) is the opcode.
ii) L_IR2 loads 2 bit from bus. These are the two flag bits: Carry Flag(bit6) and the Zero Flag (MSB or bit7).

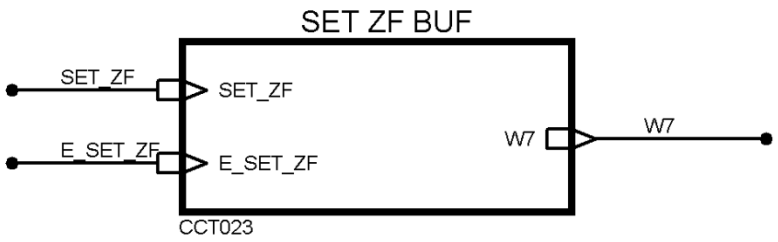
It has a combinational connection with the CONTROLLER. That means, it passes its 8 bit data directly to the controller.

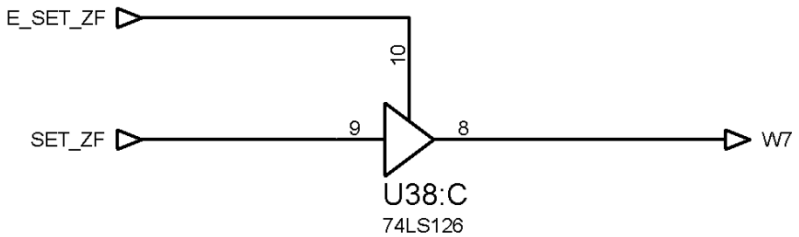

INSTRUCTION REG
L_IR1
L_IR2
CLK
W0
W1
W2
W3
W4
W5
W6
W7
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
CCT022
U35
D0
D1
D2
D3
D4
D5
Q0
Q1
Q2
Q3
Q4
Q5
MR
74S174
U21:D
74LS08
U37:A
74LS08
U36:A
U36:B
D
S
Q
R
74LS74
74LS74

SET ZF BUF (Set Zero Flag Buffer):

It is a tri-state buffer. Its job is to set/ clear the W7 bit (MSB) of bus.

It is a buffer, so unidirectional with the bus and hence $E_{SET\_ZF\_BUF}$


SET ZF BUF
SET_ZF
SET_ZF
E_SET_ZF
E_SET_ZF
W7
W7
CCT023

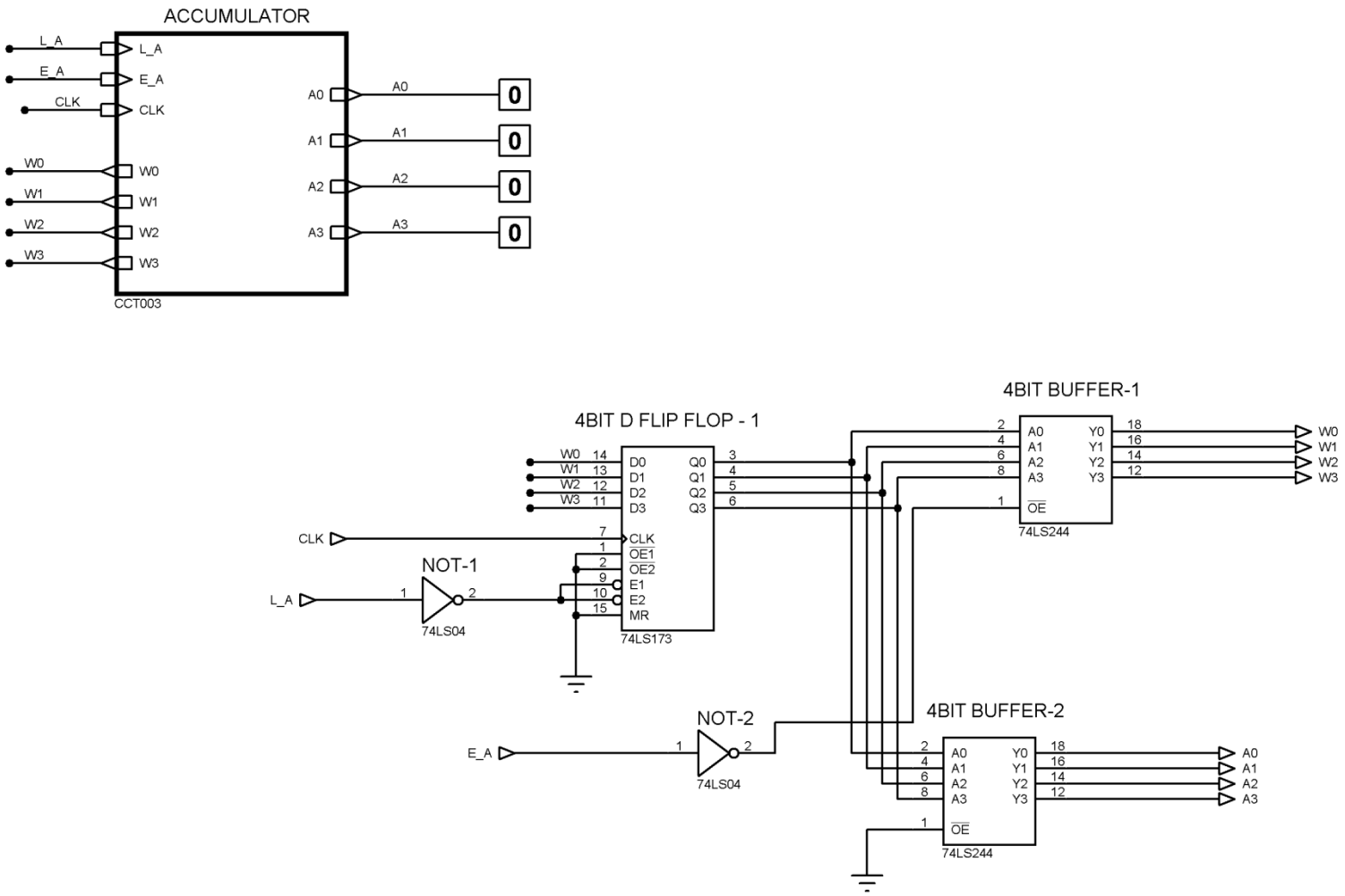

E_SET_ZF
10
SET_ZF
9
8
W7
U38:C
74LS126

ACCUMULATOR:

It is bi-directional with the bus. Hence, L_A & E_A.

It has a combinational connection with the ALU. That means, whenever there is data in Accumulator, it passes to ALU in the same clock cycle.

It is a data register, hence 4 bit transaction.

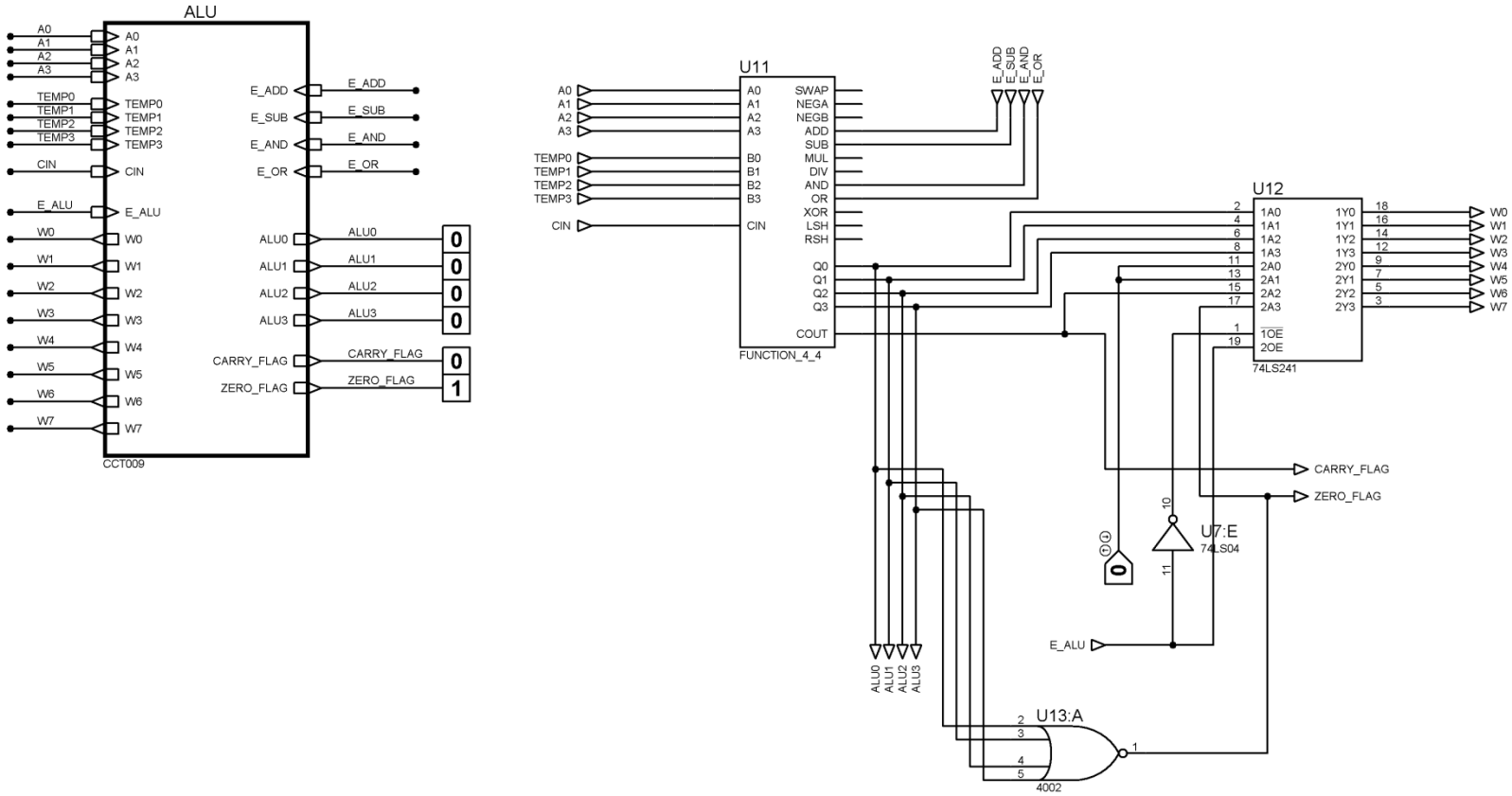

ACCUMULATOR
L_A
E_A
CLK
W0
W1
W2
W3
A0
A1
A2
A3
0
CCT003
4BIT D FLIP FLOP - 1
4BIT BUFFER-1
4BIT BUFFER-2
NOT-1
NOT-2
74LS04
74LS173
74LS244
D0
D1
D2
D3
Q0
Q1
Q2
Q3
CLK
OE1
OE2
E1
E2
MR
A0
A1
A2
A3
Y0
Y1
Y2
Y3
OE

ALU (Arithmetic Logic Unit):

The design is called 4-bit PC because our ALU operates on 4 bit data. There are direct connections from Accumulator & Temp Reg to ALU. That means, as soon as the data in these two updates, ALU performs its operation according to its control bits.

i) E_{ALU} lets ALU emit its data to bus
ii) E_{ADD} & E_{SUB} performs the arithmetic operation.
iii) E_{AND} & E_{OR} performs the logical operation.

If we notice, we can see that there is no direct connection from ALU to FLAG Register. That means, whenever ALU performs an operation, the corresponding flags are to set by us manually. For this reason, ALU computes the flags each time an operation happens and it sends them in W6-W7 (Carry Flag and Zero Flag respectively), whereas the computed 4 bit data is sent to W0-W3.

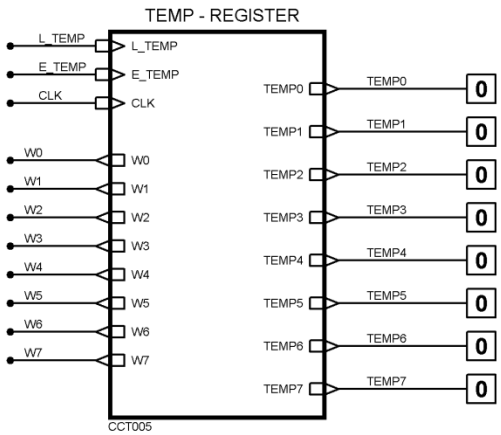

ALU
A0
A1
A2
A3
TEMP0
TEMP1
TEMP2
TEMP3
CIN
E_ALU
W0
W1
W2
W3
W4
W5
W6
W7
E_ADD
E_SUB
E_AND
E_OR
ALU0
ALU1
ALU2
ALU3
CARRY_FLAG
ZERO_FLAG
0
0
0
0
0
1
CCT009
U11
SWAP
NEGA
NEGB
ADD
SUB
MUL
DIV
AND
OR
XOR
LSH
RSH
Q0
Q1
Q2
Q3
COUT
FUNCTION_4_4
U12
1A0
1A1
1A2
1A3
2A0
2A1
2A2
2A3
1OE
2OE
1Y0
1Y1
1Y2
1Y3
2Y0
2Y1
2Y2
2Y3
74LS241
U7:E
74LS04
U13:A
4002

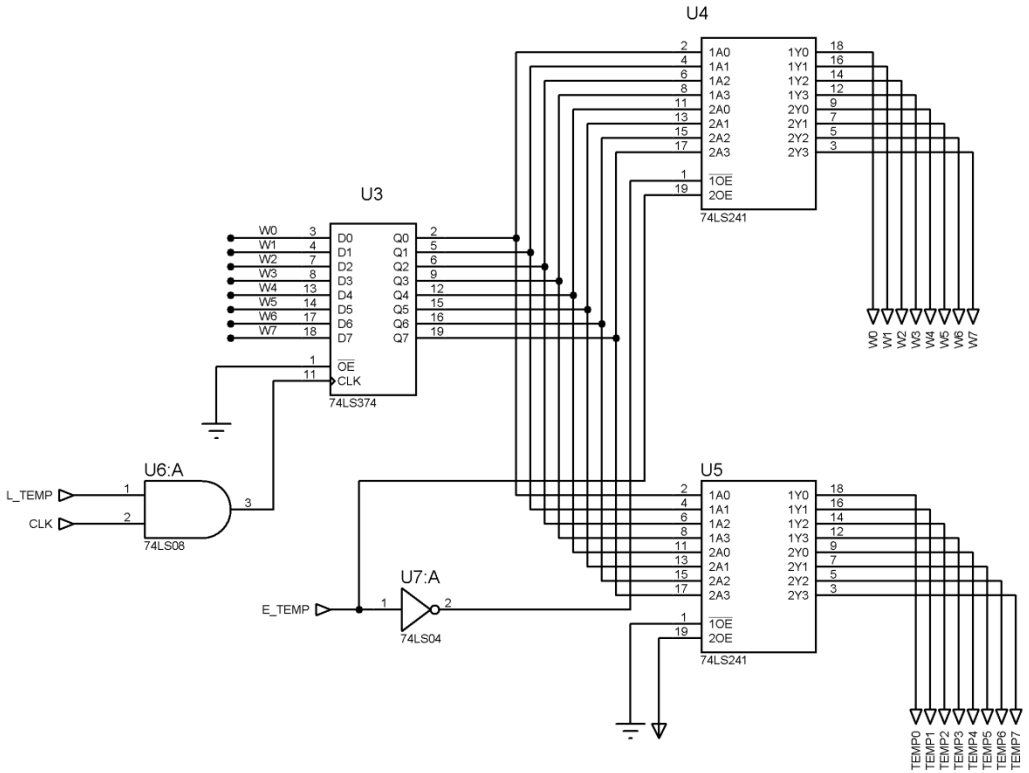
TEMP Register:

It is bidirectional with the bus, hence L_{TEMP} & $E_{TEMP.}$

It has a direct connection with ALU. That means, whenever the data in TEMP changes, ALU updates.

In our design, TEMP Register can also hold address value temporary. That's why its 8 bit register in design.

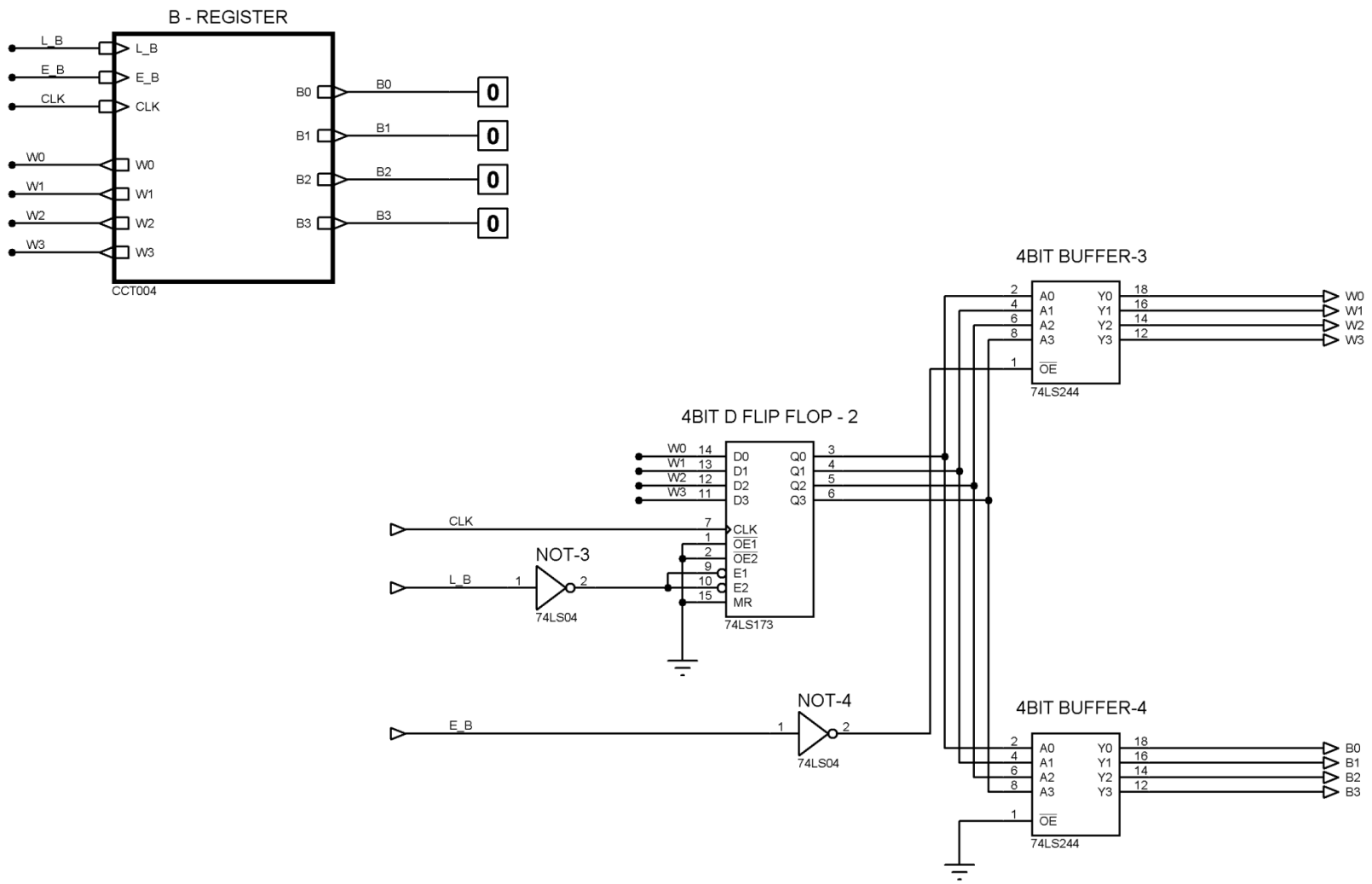

TEMP - REGISTER
L_TEMP
E_TEMP
CLK
W0
W1
W2
W3
W4
W5
W6
W7
TEMP0
TEMP1
TEMP2
TEMP3
TEMP4
TEMP5
TEMP6
TEMP7
0
CCT005


U4
U3
U6:A
U5
U7:A
74LS241
74LS374
74LS08
74LS04
L_TEMP
CLK
E_TEMP
W0
W1
W2
W3
W4
W5
W6
W7
TEMP0
TEMP1
TEMP2
TEMP3
TEMP4
TEMP5
TEMP6
TEMP7

B Register:

It is also a bidirectional data register like Accumulator unlike it has no connection with ALU directly. Any data from B Register goes to ALU via Temp Register.

Its bidirectional with bus, hence L_B & $E_{B.}$


B - REGISTER
L_B
E_B
CLK
W0
W1
W2
W3
B0
B1
B2
B3
0
CCT004
4BIT BUFFER-3
A0
A1
A2
A3
OE
Y0
Y1
Y2
Y3
74LS244
4BIT D FLIP FLOP - 2
D0
D1
D2
D3
Q0
Q1
Q2
Q3
CLK
OE1
OE2
E1
E2
MR
74LS173
NOT-3
74LS04
NOT-4
74LS04
4BIT BUFFER-4
74LS244

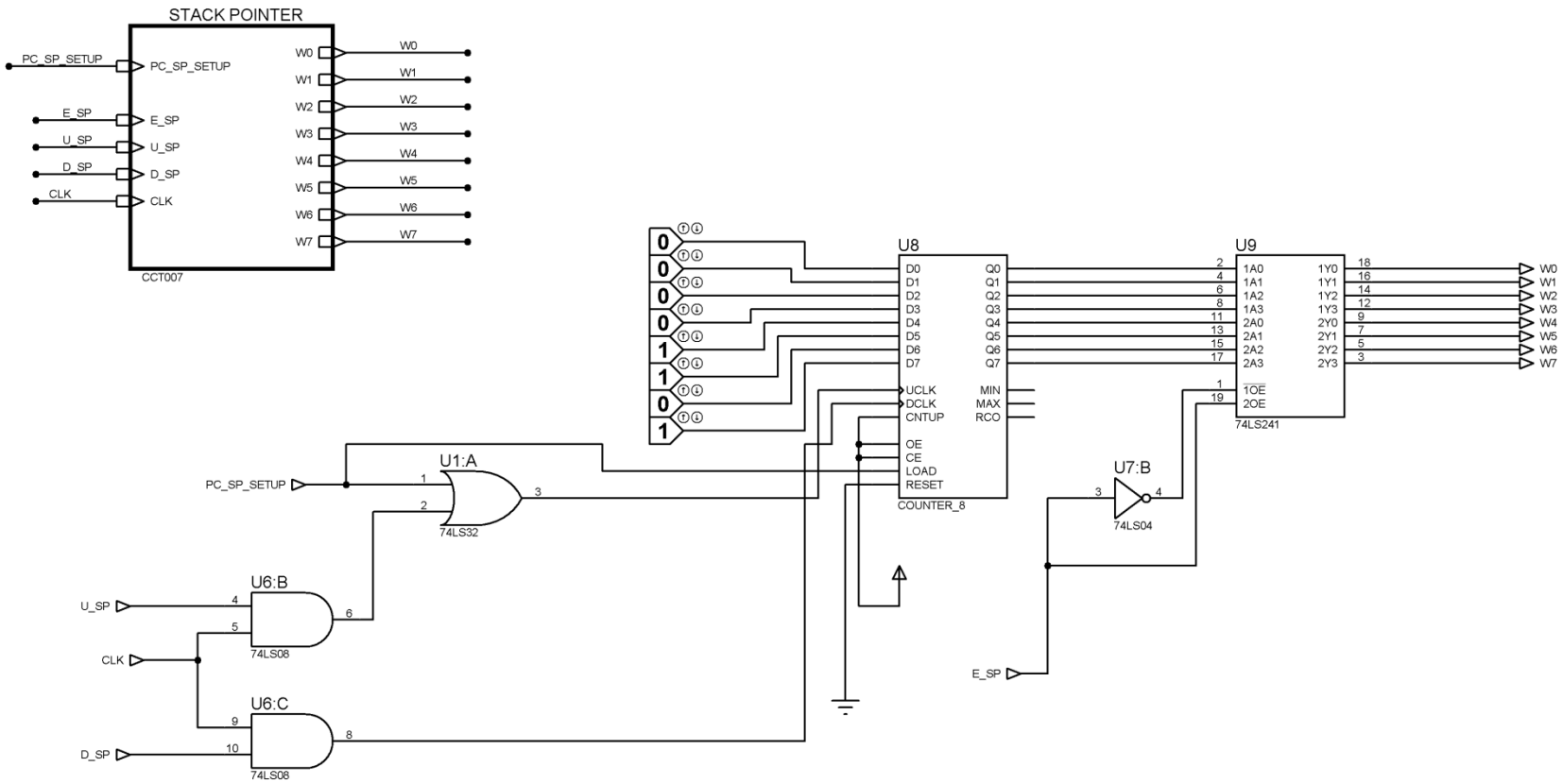
STACK POINTER:

This register manages the TOP of the stack. It always holds the address of the TOP of the stack.

It has a special bit PC_SP_SETUP. It initially sets SP address to the desired address location at the beginning of the program running.

It can both increase and decrease, that's why U_{SP} (Up Counter) & D_{SP} (Down Counter).

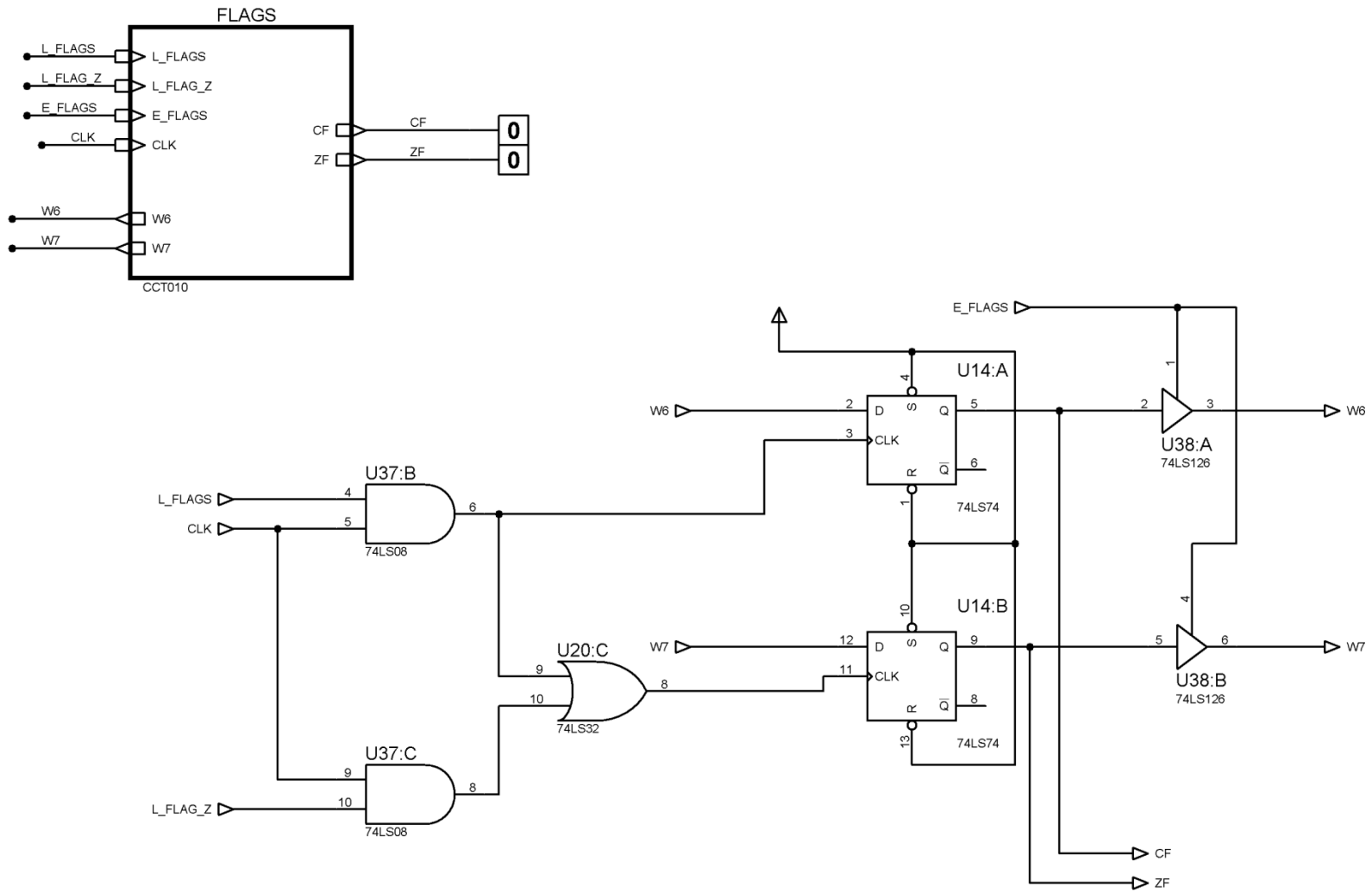
It is unidirectional with the bus, it can only give its data to the Bus. Hence, E_{SP}.


STACK POINTER
PC_SP_SETUP
E_SP
U_SP
D_SP
CLK
W0
W1
W2
W3
W4
W5
W6
W7
CCT007
U8
D0
D1
D2
D3
D4
D5
D6
D7
UCLK
DCLK
CNTUP
OE
CE
LOAD
RESET
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
MIN
MAX
RCO
COUNTER_8
U9
1A0
1A1
1A2
1A3
2A0
2A1
2A2
2A3
1OE
2OE
1Y0
1Y1
1Y2
1Y3
2Y0
2Y1
2Y2
2Y3
74LS241
U1:A
74LS32
U6:B
74LS08
U6:C
74LS08
U7:B
74LS04

FLAGS REGISTER:

It is 2 bit data register. It just loads W6 & W7 bit bus data and saves in it as Carry Flag(W6) and Zero Flag (W7).

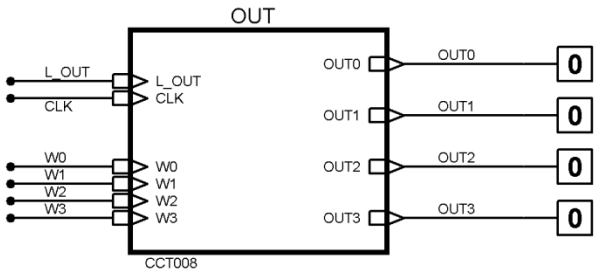
i) L_FLAGS loads both CF & ZF
ii) L_FLAGS_Z loads only W7 data (ZF)
iii) E_FLAGS lets it emit its 2 flags in 2 bits.

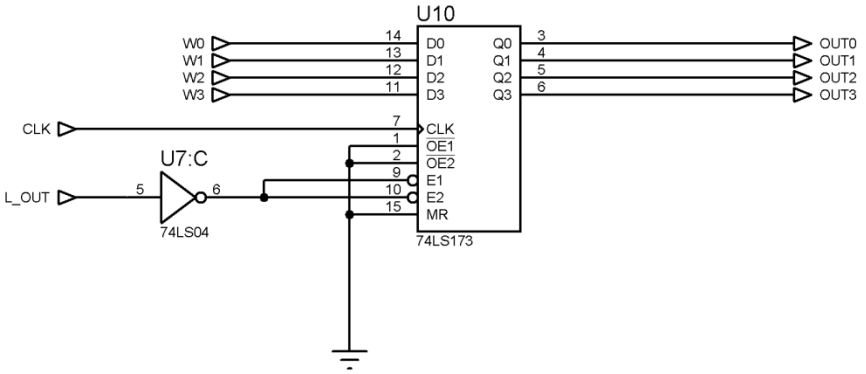

FLAGS
L_FLAGS
L_FLAG_Z
E_FLAGS
CLK
W6
W7
CF
ZF
0
0
CCT010
E_FLAGS
U14:A
W6
D
S
Q
CLK
R
Q
74LS74
U38:A
74LS126
U37:B
74LS08
U14:B
W7
U20:C
74LS32
U37:C
74LS08
U38:B
74LS126
CF
ZF

OUT:

OUT register loads 4 bit data (W0-W3) from the bus and shows it to the binary display.

It is unidirectional with the bus, it only loads data from the bus. Hence, L_{OUT} only.


OUT
L_OUT
CLK
W0
W1
W2
W3
L_OUT
CLK
W0
W1
W2
W3
OUT0
OUT1
OUT2
OUT3
OUT0
OUT1
OUT2
OUT3
0
0
0
0
CCT008


U10
W0
W1
W2
W3
14
13
12
11
D0
D1
D2
D3
Q0
Q1
Q2
Q3
3
4
5
6
OUT0
OUT1
OUT2
OUT3
CLK
7
CLK
1
OE1
2
OE2
9
E1
10
E2
15
MR
U7:C
L_OUT
5
6
74LS04
74LS173

CONTROLLER:

This is the core part of the design or “Heart of Design.”

It generates 40 bit control word for each T-state of each instruction. Each Control word gets generated into the falling edge of the clock whereas each sequential execution gets executed in the rising edge of the clock.

CONTROL WORD (CON):

$E_{IN}C_PL_{PC}E_{PC}|L_{MAR}R_{RAM}W_{RAM}L_{MDR}||$ (EPROM1)

$E_{MDR}L_{IR1}L_{IR2}SET_{ZF}|E_{SET\_ZF}L_AE_AC_{IN}||$(EPROM2)

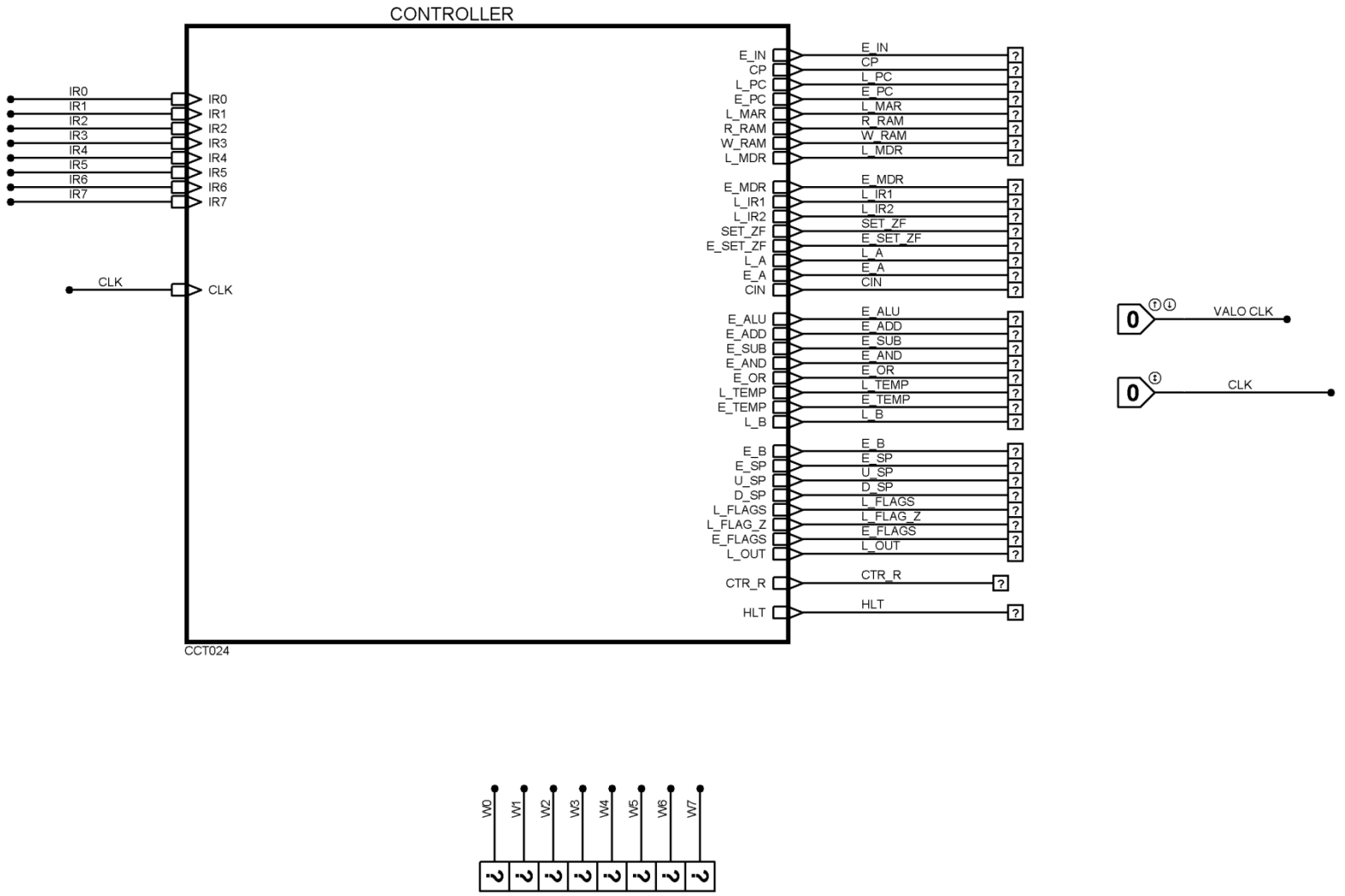
$E_{ALU}E_{ADD}E_{SUB}E_{AND}|E_{OR}L_{TEMP}E_{TEMP}L_B||$(EPROM3)

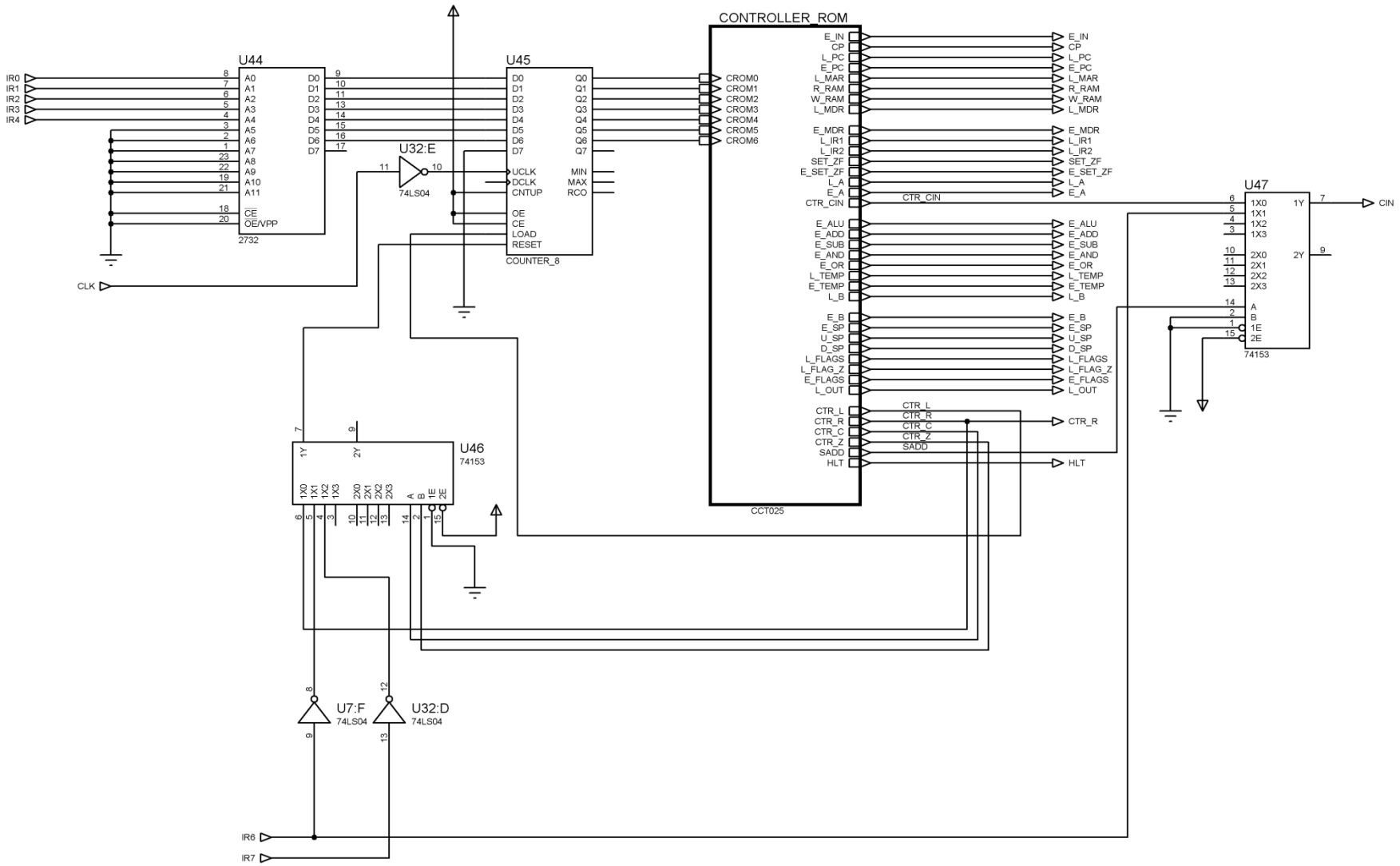
$E_BE_{SP}U_{SP}D_{SP}|L_{FLAGS}L_{FLAG\_Z}E_{FLAGS}L_{OUT}||$(EPROM4)

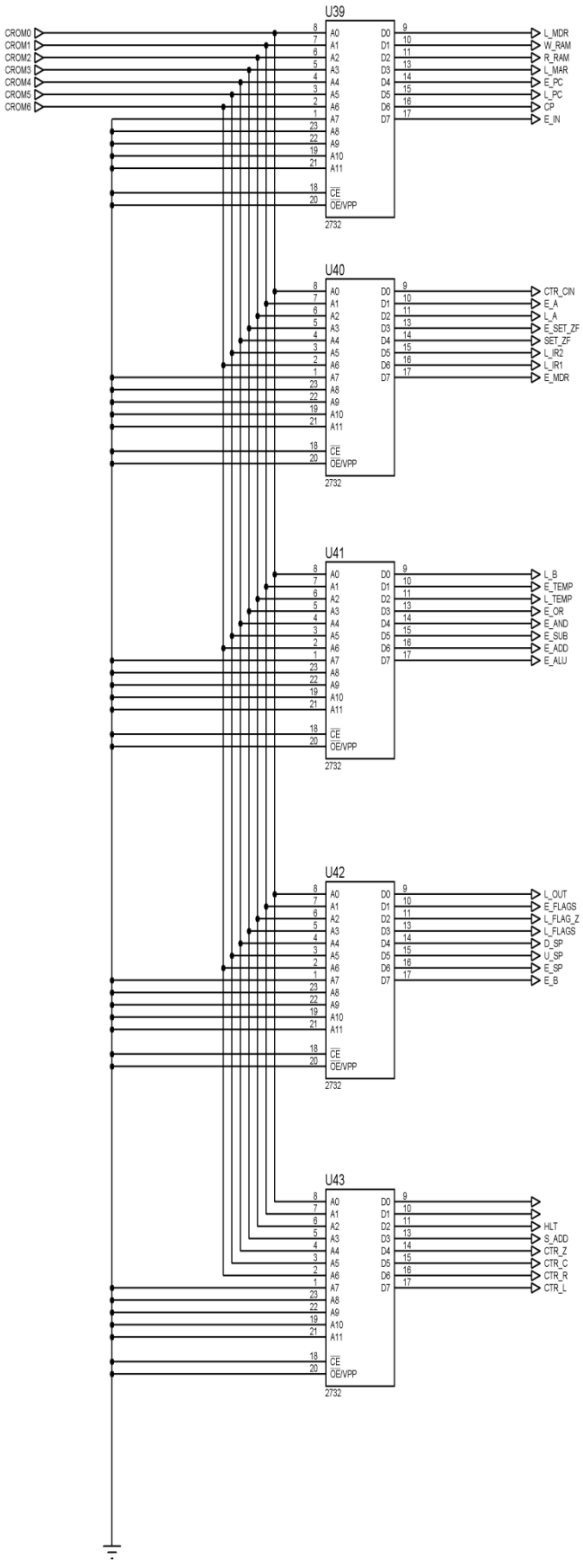
$CTR_LCTR_RCTR_CCTR_Z|S_{ADD}$HLT 0 0 (EPROM5)

Functionality:

- There was a direct connection with the instruction register to the controller. For this reason, it could take opcodes from IR as direct inputs.
- After taking opcodes(IR0-IR4) it searched an EPROM for address where the control word for the associated instruction was loaded.
- There is a MUX for iterating into the CONTROLLER_ROM for fetching pre-saved control word associated with that particular T-state
- CTR_C,CTR_Z,CTR_L, ,CTR_R these four control bits were used to control the MUX in order to take decision whether to reset the counter or not.
- In CONTROLLER_ROM, 5 EPROMs were cascaded ot produce a 40 bit control word according to the definite format stated before.
- We had 93 T-states in total. For this reason, CONTROLLER_ROM had to have 7 bits to iterate within. (93< 2^7=128)
- Another MUX was used in order to control C_{IN}. In case of Special add instructions (like ADC, SBB) we had to control $C_{IN.}$ For this reason, a sepearte control bit SADD was introduced to control this MUX.


CONTROLLER
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
CLK
E_IN
CP
L_PC
E_PC
L_MAR
R_RAM
W_RAM
L_MDR
E_MDR
L_IR1
L_IR2
SET_ZF
E_SET_ZF
L_A
E_A
CIN
E_ALU
E_ADD
E_SUB
E_AND
E_OR
L_TEMP
E_TEMP
L_B
E_B
E_SP
U_SP
D_SP
L_FLAGS
L_FLAG_Z
E_FLAGS
L_OUT
CTR_R
HLT
CCT024
VALO CLK
CLK
W0
W1
W2
W3
W4
W5
W6
W7

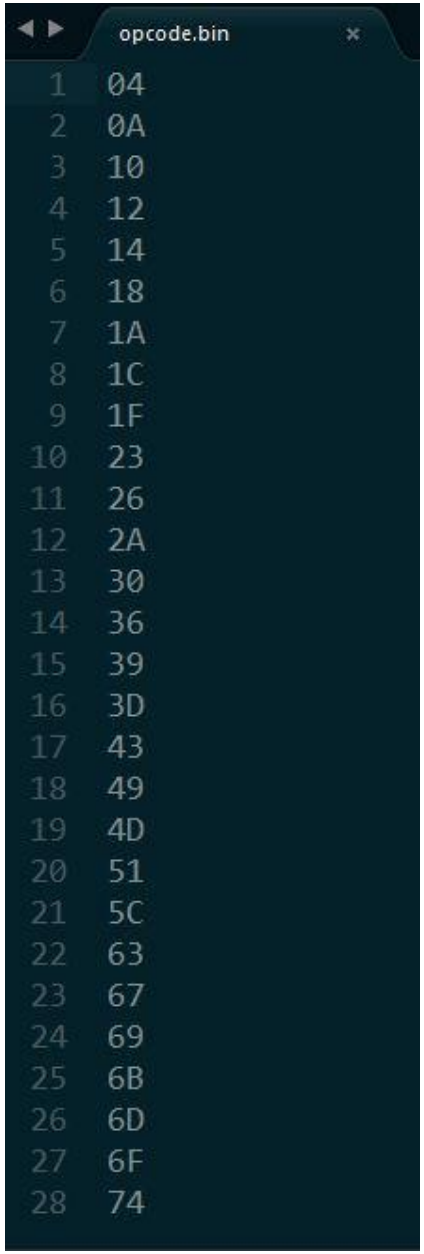

CONTROLLER_ROM
U44
2732
U45
COUNTER_8
U32:E
74LS04
U46
74153
U47
74153
U7:F
74LS04
U32:D
74LS04
CCT025
IR0
IR1
IR2
IR3
IR4
IR6
IR7
CLK
CIN
CTR_CIN
CTR_R
HLT
E_IN
CP
L_PC
E_PC
L_MAR
R_RAM
W_RAM
L_MDR
E_MDR
L_IR1
L_IR2
SET_ZF
E_SET_ZF
L_A
E_A
E_ALU
E_ADD
E_SUB
E_AND
E_OR
L_TEMP
E_TEMP
L_B
E_B
E_SP
U_SP
D_SP
L_FLAGS
L_FLAG_Z
E_FLAGS
L_OUT
CTR_L
CTR_R
CTR_C
CTR_Z
SADD
HLT
CROM0
CROM1
CROM2
CROM3
CROM4
CROM5
CROM6

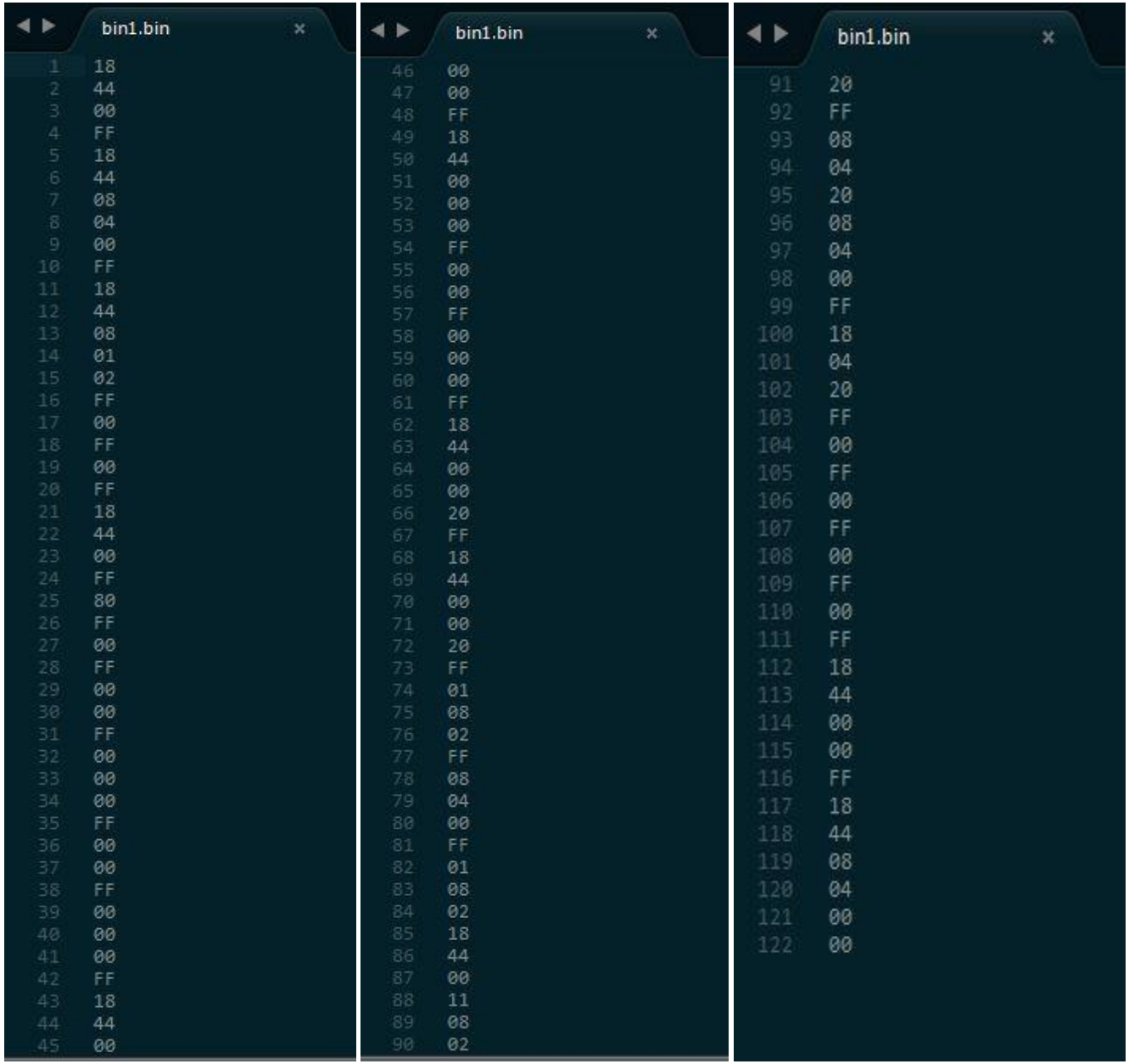

U39
CROM0
CROM1
CROM2
CROM3
CROM4
CROM5
CROM6
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
CE
OE/VPP
D0
D1
D2
D3
D4
D5
D6
D7
L_MDR
W_RAM
R_RAM
L_MAR
E_PC
L_PC
CP
E_IN
2732
U40
CTR_CIN
E_A
L_A
E_SET_ZF
SET_ZF
L_IR2
L_IR1
E_MDR
2732
U41
L_B
E_TEMP
L_TEMP
E_OR
E_AND
E_SUB
E_ADD
E_ALU
2732
U42
L_OUT
E_FLAGS
L_FLAG_Z
L_FLAGS
D_SP
U_SP
E_SP
E_B
2732
U43
HLT
S_ADD
CTR_Z
CTR_C
CTR_R
CTR_L
2732

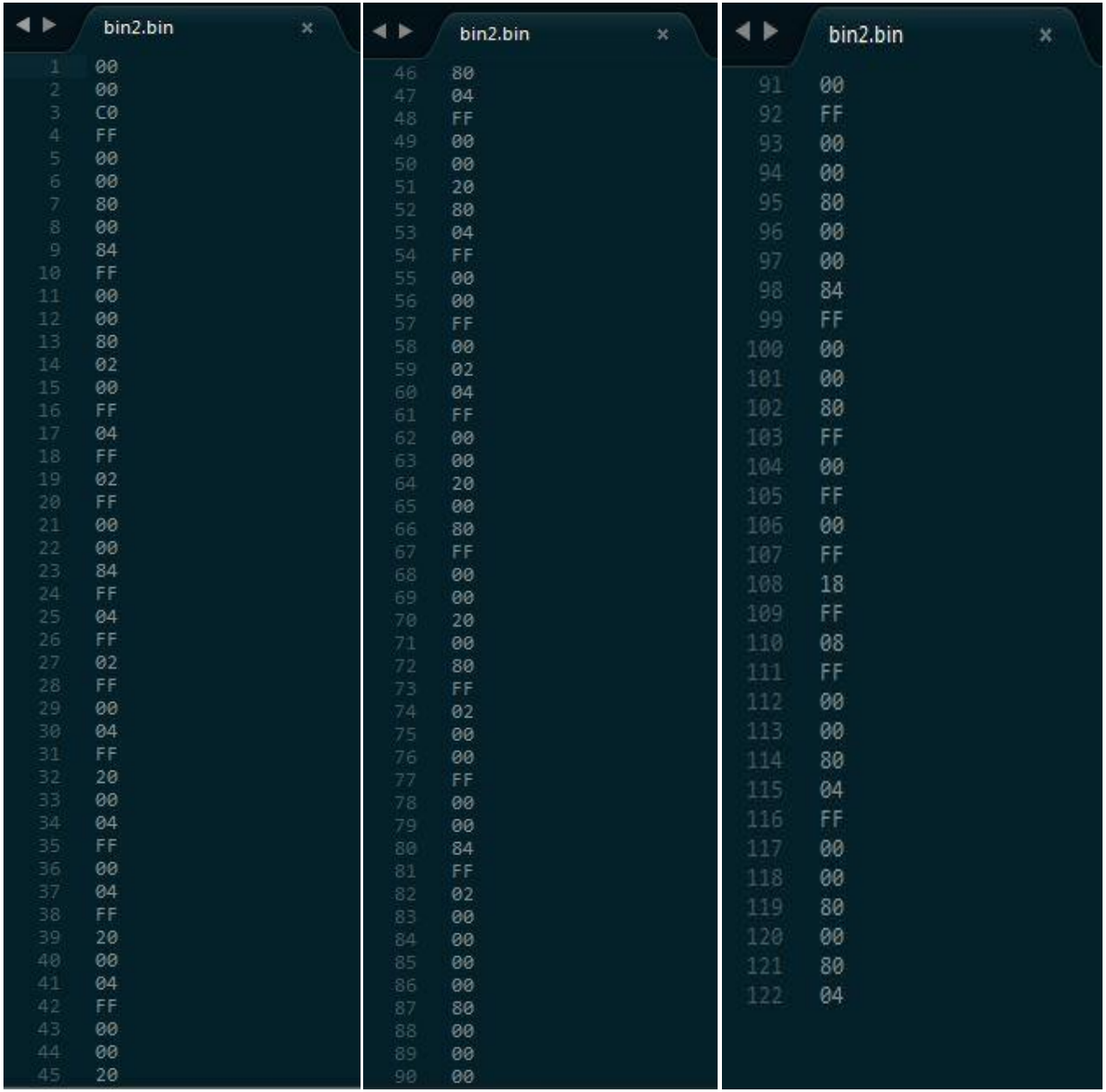
BIN FILES:

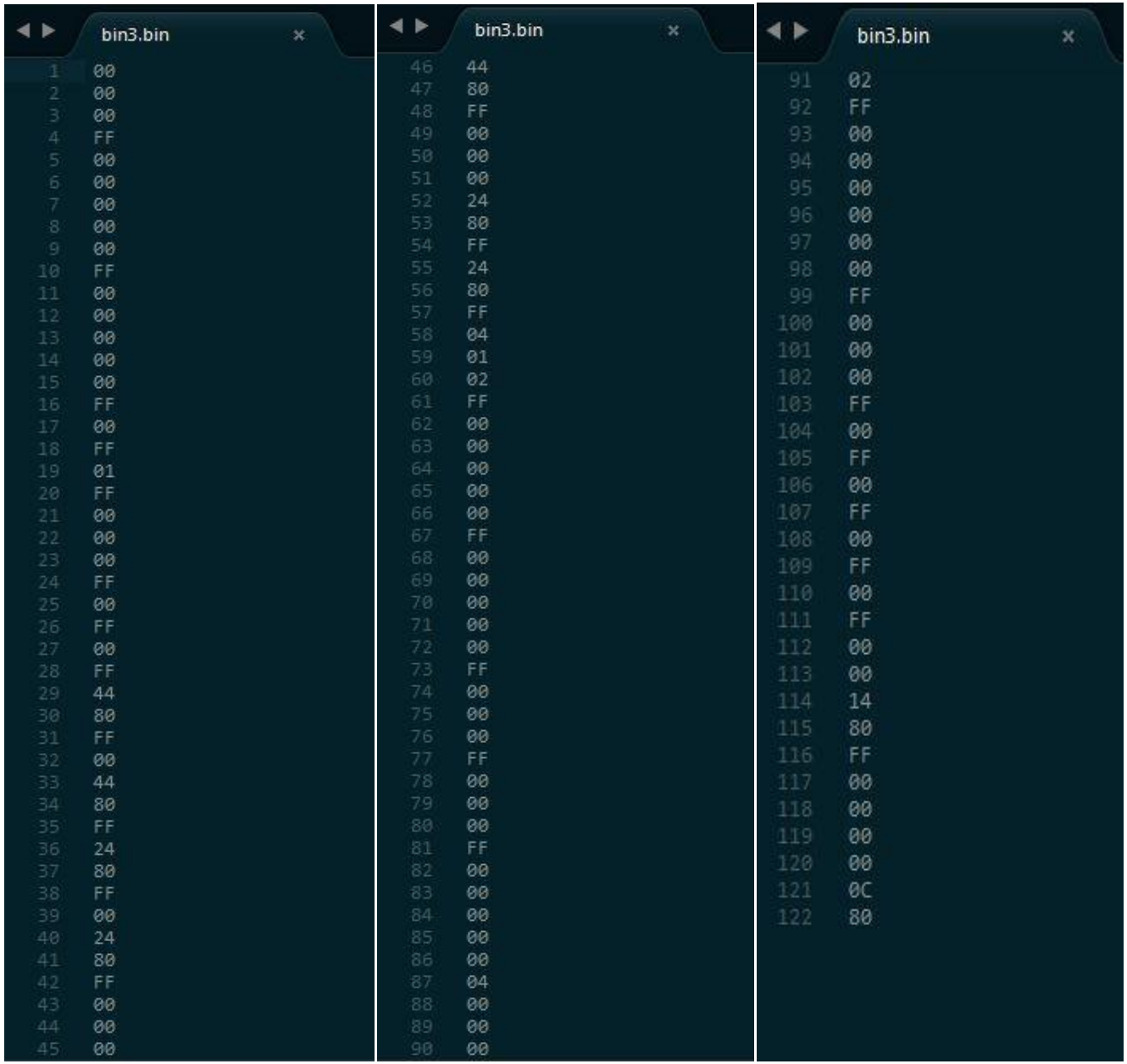
Bin files are saved into specific folder. One can modify it but he/she has to understand thoroughly how the bin files were pre-saved at the first case.

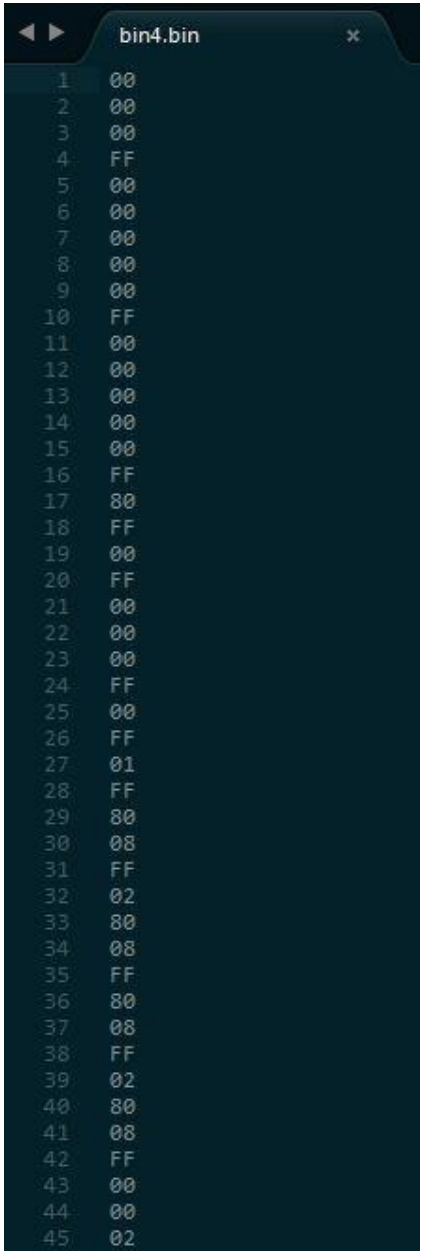
- bin1.bin : EPROM1
- bin2 .bin: EPROM2
- bin3 .bin: EPROM3
- bin4 .bin: EPROM4
- bin5 .bin: EPROM5
- opcode.bin: keeps the starting address of Execution cycle of that particular opcoded instruction.
- Assembly.bin: stores the HEX Code of assembly program (with these 28 instruction sets.)

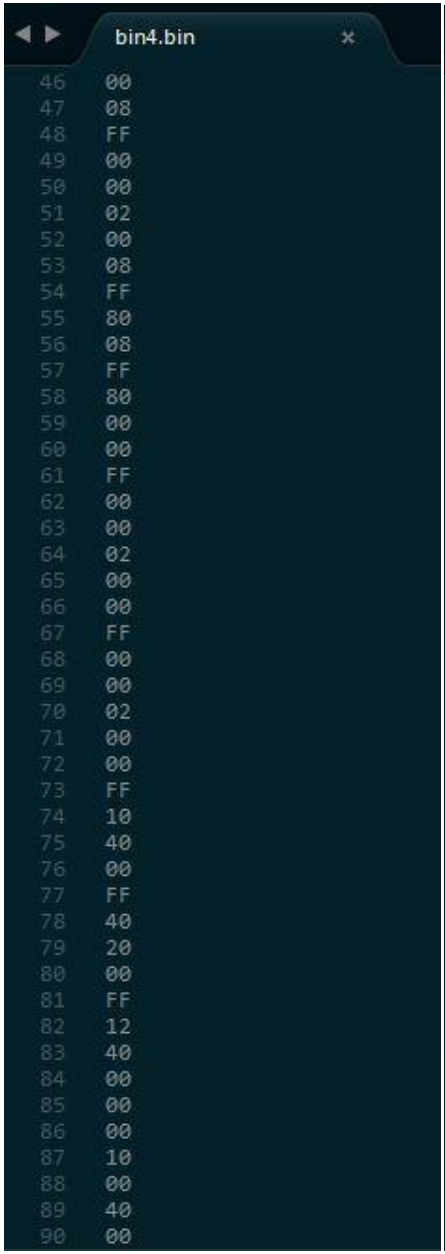

opcode.bin
1 04
2 0A
3 10
4 12
5 14
6 18
7 1A
8 1C
9 1F
10 23
11 26
12 2A
13 30
14 36
15 39
16 3D
17 43
18 49
19 4D
20 51
21 5C
22 63
23 67
24 69
25 6B
26 6D
27 6F
28 74

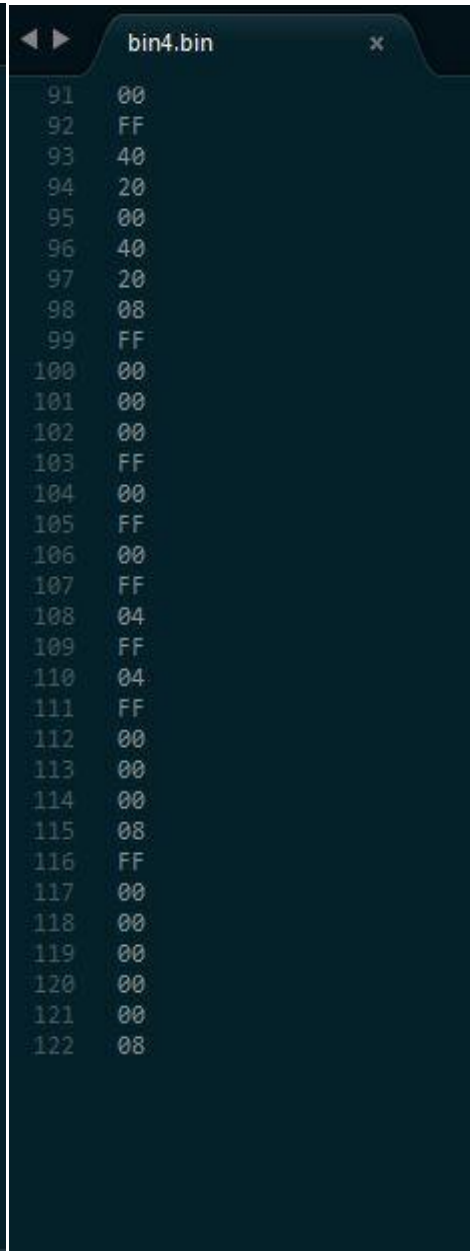

bin1.bin
18
44
00
FF
18
44
08
04
00
FF
18
44
08
01
02
FF
00
FF
00
FF
18
44
00
FF
80
FF
00
FF
00
00
FF
00
00
00
FF
00
00
FF
18
44
00
bin1.bin
00
00
FF
18
44
00
00
00
FF
00
00
FF
00
00
FF
18
44
00
00
20
FF
18
44
00
00
20
FF
01
08
02
FF
08
04
00
FF
01
08
02
18
44
00
11
08
02
bin1.bin
20
FF
08
04
20
08
04
00
FF
18
04
20
FF
00
FF
00
FF
00
FF
00
FF
18
44
00
00
FF
18
44
08
04
00
00

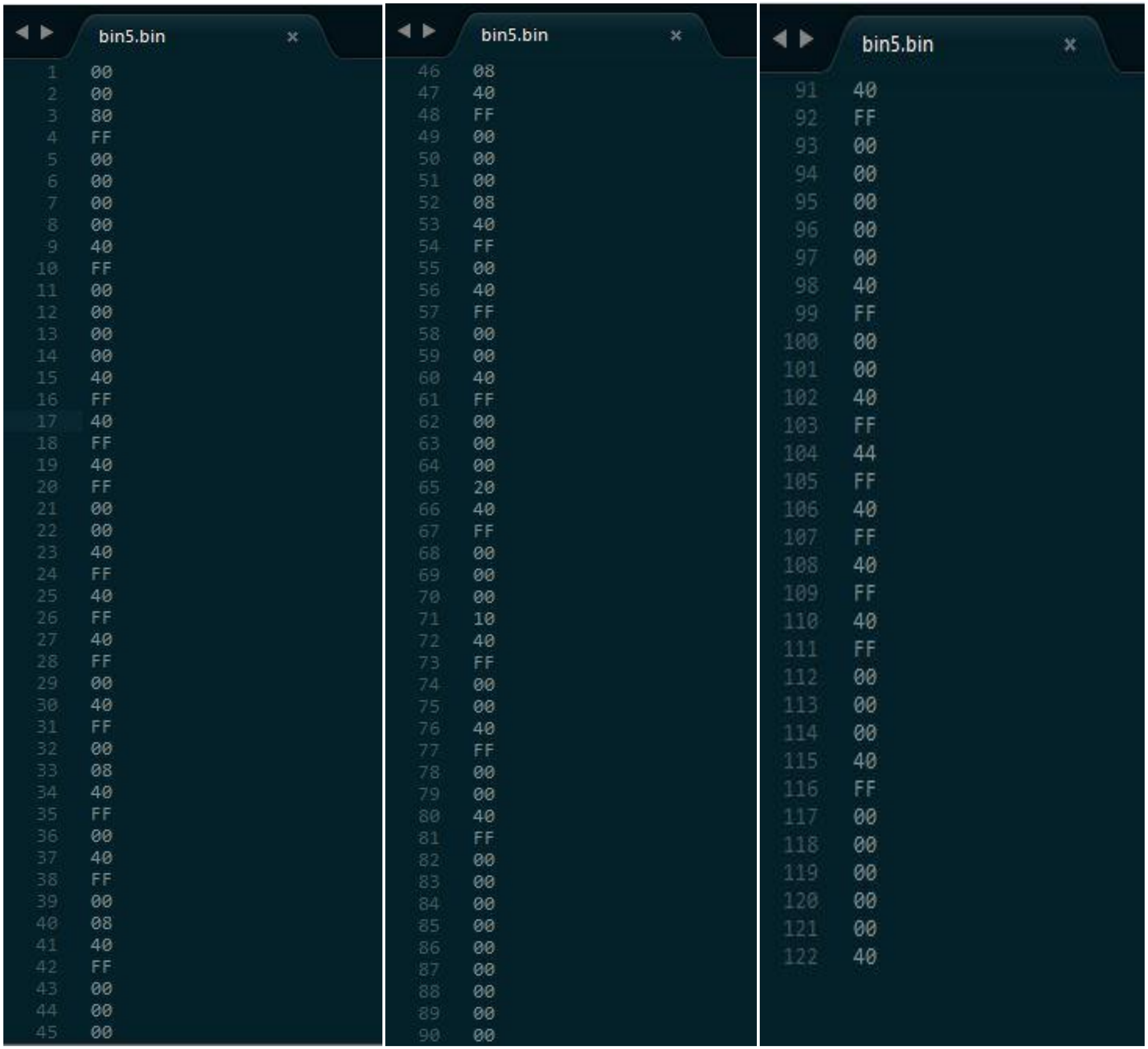

bin2.bin
00
00
C0
FF
00
00
80
00
84
FF
00
00
80
02
00
FF
04
FF
02
FF
00
00
84
FF
04
FF
02
FF
00
04
FF
20
00
04
FF
00
04
FF
20
00
04
FF
00
00
20
bin2.bin
80
04
FF
00
00
20
80
04
FF
00
00
FF
00
02
04
FF
00
00
20
00
80
FF
00
00
20
00
80
FF
02
00
00
FF
00
00
84
FF
02
00
00
00
80
00
00
00
bin2.bin
00
FF
00
00
80
00
00
84
FF
00
00
80
FF
00
FF
00
FF
18
FF
08
FF
00
00
80
04
FF
00
00
80
00
80
04


bin3.bin
00
00
00
FF
00
00
00
00
00
00
FF
00
00
00
00
00
00
FF
00
FF
01
FF
00
00
00
FF
00
FF
00
FF
44
80
FF
00
44
80
FF
24
80
FF
00
24
80
FF
00
00
00
bin3.bin
44
80
FF
00
00
00
24
80
FF
24
80
FF
04
01
02
FF
00
00
00
00
00
FF
00
00
00
FF
00
00
00
FF
00
00
00
FF
00
00
00
04
00
00
00
bin3.bin
02
FF
00
00
00
00
00
00
00
FF
00
00
00
FF
00
FF
00
FF
00
FF
00
FF
00
00
00
14
80
FF
00
00
00
00
0C
80


bin4.bin
1 00
2 00
3 00
4 FF
5 00
6 00
7 00
8 00
9 00
10 FF
11 00
12 00
13 00
14 00
15 00
16 FF
17 80
18 FF
19 00
20 FF
21 00
22 00
23 00
24 FF
25 00
26 FF
27 01
28 FF
29 80
30 08
31 FF
32 02
33 80
34 08
35 FF
36 80
37 08
38 FF
39 02
40 80
41 08
42 FF
43 00
44 00
45 02


bin4.bin
46 00
47 08
48 FF
49 00
50 00
51 02
52 00
53 08
54 FF
55 80
56 08
57 FF
58 80
59 00
60 00
61 FF
62 00
63 00
64 02
65 00
66 00
67 FF
68 00
69 00
70 02
71 00
72 00
73 FF
74 10
75 40
76 00
77 FF
78 40
79 20
80 00
81 FF
82 12
83 40
84 00
85 00
86 00
87 10
88 00
89 40
90 00


bin4.bin
91 00
92 FF
93 40
94 20
95 00
96 40
97 20
98 08
99 FF
100 00
101 00
102 00
103 FF
104 00
105 FF
106 00
107 FF
108 04
109 FF
110 04
111 FF
112 00
113 00
114 00
115 08
116 FF
117 00
118 00
119 00
120 00
121 00
122 08



bin5.bin

```
00
00
80
FF
00
00
00
00
40
FF
00
00
00
00
40
FF
40
FF
40
FF
00
00
40
FF
40
FF
40
FF
00
40
FF
00
08
40
FF
00
40
FF
00
08
40
FF
00
00
00
```

bin5.bin

```
08
40
FF
00
00
00
08
40
FF
00
40
FF
00
00
40
FF
00
00
00
20
40
FF
00
00
00
10
40
FF
00
00
40
FF
00
00
40
FF
00
00
00
00
00
00
00
00
00
```

bin5.bin

```
40
FF
00
00
00
00
00
40
FF
00
00
40
FF
44
FF
40
FF
40
FF
40
FF
00
00
00
40
FF
00
00
00
00
00
40
```

dummy.cpp

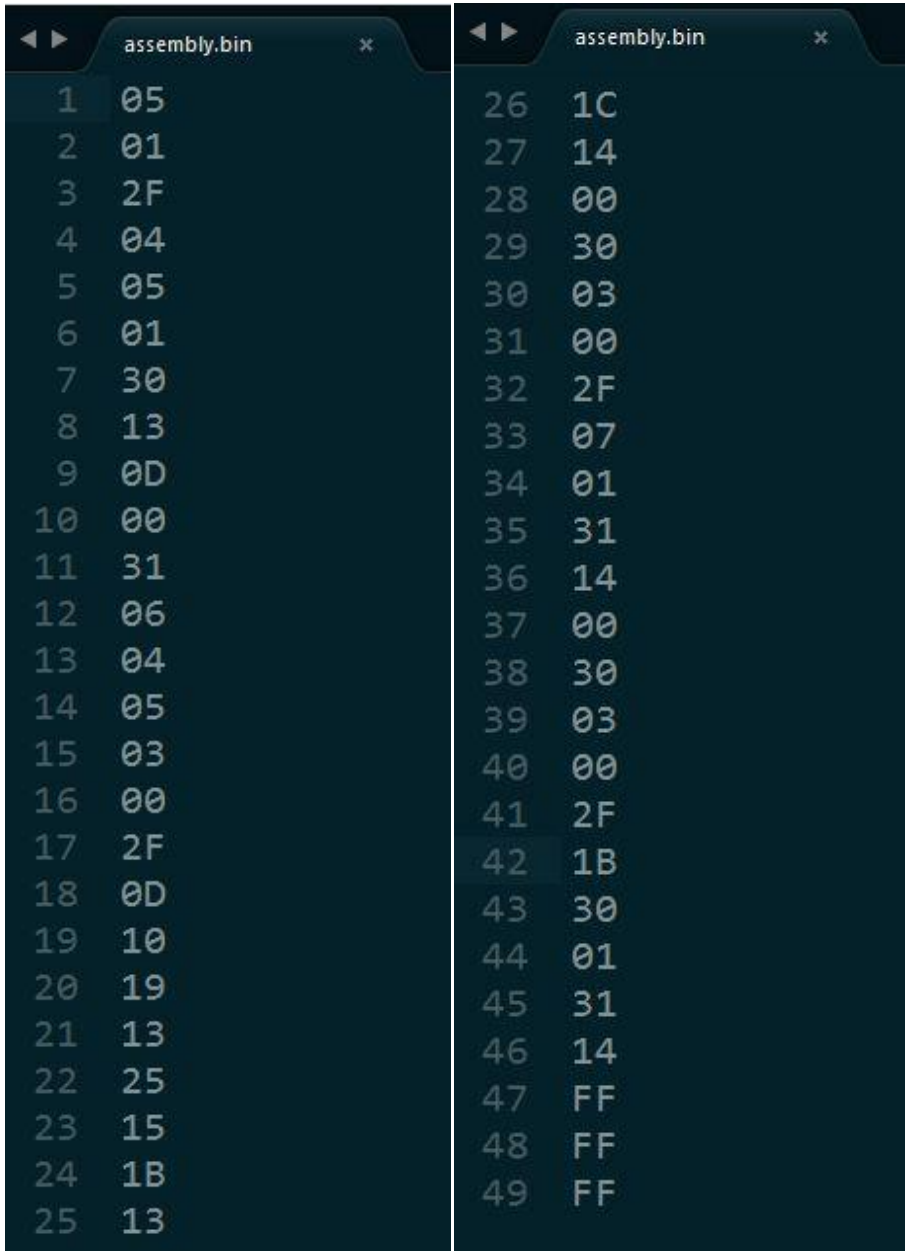
```
main()
{
    read X;
    store X in RAM;
    store 5 in #30; //address 30H

    call compare();
    OUT;
}

compare()
{
    if (X=5) call add();
    else call logical();
}

add()
{
    X=X+[30];
}

logical()
{
    X= X logicalOP [30];        //for our case, logicalOP= OR
}
```


assembly.bin
1 05
2 01
3 2F
4 04
5 05
6 01
7 30
8 13
9 0D
10 00
11 31
12 06
13 04
14 05
15 03
16 00
17 2F
18 0D
19 10
20 19
21 13
22 25
23 15
24 1B
25 13
assembly.bin
26 1C
27 14
28 00
29 30
30 03
31 00
32 2F
33 07
34 01
35 31
36 14
37 00
38 30
39 03
40 00
41 2F
42 1B
43 30
44 01
45 31
46 14
47 FF
48 FF
49 FF

How to Operate this 4-bit machine:

- Convert the pseudocode to corresponding Assembly Language Instructions
- Convert the Instructions into HexCoded Machine Language
- Save them into “assembly.bin” file
- Simulate the Proteus File (RUN)
- Press UP_INIT once to turn it ON
- Press UP_INIT_CLOCK repeatedly until all hex codes in “assembly.bin” gets boot-loaded to the RAM.
- Press PC_SP_SETUP once.
- Press CLK repeatedly to provide clock pulse to the machine until all instructions finish execution.

Special Features:

- We have boot-loader to upload the assembly program to the RAM initially.
- Our memory is divided into three separate segments:
  CODE Segment (CS), DATA Segment(DS), STACK Segment(SS).
  CS: 1-100------->100
  DS: 101-175-----> 75
  SS: 176-255 ----> 80

  SS is fixed but CS & DS are flexible and user-controlled.

- We have detailed output pins to show what is really going on inside the 4bit Machine. It helps to DEBUG at a large extent.
- To provide more accurate debugging, we intentionally left CLK and UP_INIT_CLK logical toggle instead of using a clock pulse. It helped to see even the CONTROL WORDS generated in each T-State.
- The W-BUS was left neat & clean instead of messy circuitry.

Chip Count:

- AND(74LS08): 11
- OR (74LS32): 04
- NOR (74LS02): 1
- NOT (74LS04): 21
- 4-INPUT-NOR (4002): 1
- 1-BIT-D-FLIP-FLOP (74LS74): 4
- 4-BIT-D-FLIP-FLOP (74LS173): 3
- 6-BIT-D-FLIP-FLOP (74S174): 1
- 8-BIT-D-FLIP-FLOP (74LS374): 1
- 1-BIT-TRISTATE-BUFFER (74LS126): 3
- 4-BIT-TRISTATE-BUFFER (74LS244): 5
- 8-BIT-TRISTATE-BUFFER (74LS241): 8
- EPROM (2732): 7
- RAM (6116): 1
- DUAL 4x1 MUX (74153): 4
- UP-DOWN COUNTER (COUNTER_8): 4
- ALU(FUNCTION_4_4): 1

Discussion:

- This 4 bit pc is functional for 28 instructions only.
- The opcodes for these 28 instructions are fixed and cannot be changed.
- RAM loads address from 1^{st} address instead of 0^{th}.
- PC initially starts from 0. So, we had to manually increase "PC_SP_SETUP" to 1 before running the assembly program for synching issues with RAM.
- We put one extra clock "UP_INIT_CLK" for boot-loading into RAM.
- COUNTER_8 & FUNCTION_4_4 are not real chips but are Proteus Packages. For this reason, some control bits in ALU like E_{ADD}, $E_{SUB,}$ $E_{AND,}$ E_{OR} are actually implemented with permutated sequence of real status bits.
- The whole machine works in two separate clock edges.
  i) In falling edge, controller generates control signals for next T-state of instruction.
  ii) In rising edge, the sequential operations like load, write etc are performed for that T-state.

- If more than one component tries to emit data at bus, it gets garbage and we get “logic contention error”.
- Sometimes, keeping a pin floating also gave logic contention error.
- In another approach, it could be done that, the boot-loader could use PC instead of a new counter inside RAM. Then it would need respective design calibrations & modifications.
- In case of Call we PUSH accumulator data, Flag data into Stack and POP when RET.
- In case of returning any value, instead of keeping it into STACK segment (as per as the ideal case), we keep it into the DATA segment of the memory.

- Combinational parts:
    i) MAR->RAM
    ii) Ram->MDR combinational, but operates in the next clock cycle.
    iii) IR->CONTROLLER
    iv) ACCUMULATOR->ALU
    v) TEMP->ALU
    vi) OUT->BINARY DISPLAY
- Sequential parts:
    i) IN->BUS
    ii) PC<-BUS
    iii) BUS->MAR
    iv) MDR->RAM
    v) BUS->MDR,
    vi) IR<-BUS
    vii) ACCUMULATOR<-BUS
    viii) TEMP<-BUS
    ix) B<-BUS
    x) FLAG<-BUS
    xi) BUS->OUT

T-State Analysis

Special Thanks to:

- Md. Iftekharul Islam Sakib
  Lecturer, CSE,BUET.
- Md Ishtiyaque Ahmed
  Lecturer, CSE,BUET.