Bangladesh University of Engineering and Technology

Course No: CSE 404

Course Title: Digital System Design Sessional

A Report on 4-bit PC

Date of Submission: August 4, 2018

Submitted By:

Tahrina Tasnim Misha (1405100)
Nahiyaan Uddin (1405102)
Salsabil Arabi Shushmi (1405108)
Mirajul Islam (1405119)
Rajan Shrestha (1405121)
Saurav Gupta (1405122)

Introduction:

For 4-bit pc assignment we have to implement 25 instructions.
According to our 25 instructions the 4-bit PC has
-In Port
-Program Counter(PC),
-Memory address register(MAR)
- Memory
- Memory Data Register(MDR)
- Instruction Register(IR)
-Controller sequencer(CON)
- Accumulator(ACC)
- Arithmetic logic unit(ALU)
-Flag
-Temp reg
-B reg
-Stack pointer(SP)
-Output reg(OUT)
-Display

The Address bus is of 8-bit
The Data is 4-bit
We need several clock cycles and maximum 11 T-cycles for an instruction.
We need 40 signal pins on a whole.

Instruction set:

Instruction	Description
1. STA address	Memory[address] ← Acc
2. MOV Acc, B	Acc ← B
3. MOV B, Acc	B ← Acc
4. ADC B	Acc ← Acc + B + Carry
5. SUB B	Acc ← Acc - B
6. SBB address	Acc ← Acc - Memory[address] - Carry
7. SBB Immediate	Acc ← Acc - Immediate - Carry
8. IN	Acc ← Input_port
9. OUT	Output _port ← Acc
10. PUSH	Pushes the content of the Accumulator to the Stack
11. POP	Pops off top element of stack to Accumulator
12. AND B	Acc ← Acc.B
13. OR address	Acc ← Acc | Memory[address]
14. OR Immediate	Acc ← Acc | Immediate
15. XOR address	Acc ← Acc Memory[address]
16. CMP B	Accumulator will be unchanged. Set flags according to (Acc - B)
17. SHL	Acc ← Acc << 1, Carry ← Acc [MSB], Acc [LSB] ← 0
18. ROR	Acc ← Acc >> 1, Carry ← Acc [LSB], Acc [MSB] ← Acc [LSB]
19. JMP address	Jumps to the address
20. JNC address	Jumps to the address if Carry flag is not set
21. JG address	Jump if greater
22. JL address	Jump if less
23. CMC	Complements the Carry flag
24. CLC	Clears the Carry flag
25. HLT	Halts execution

Block Diagram:

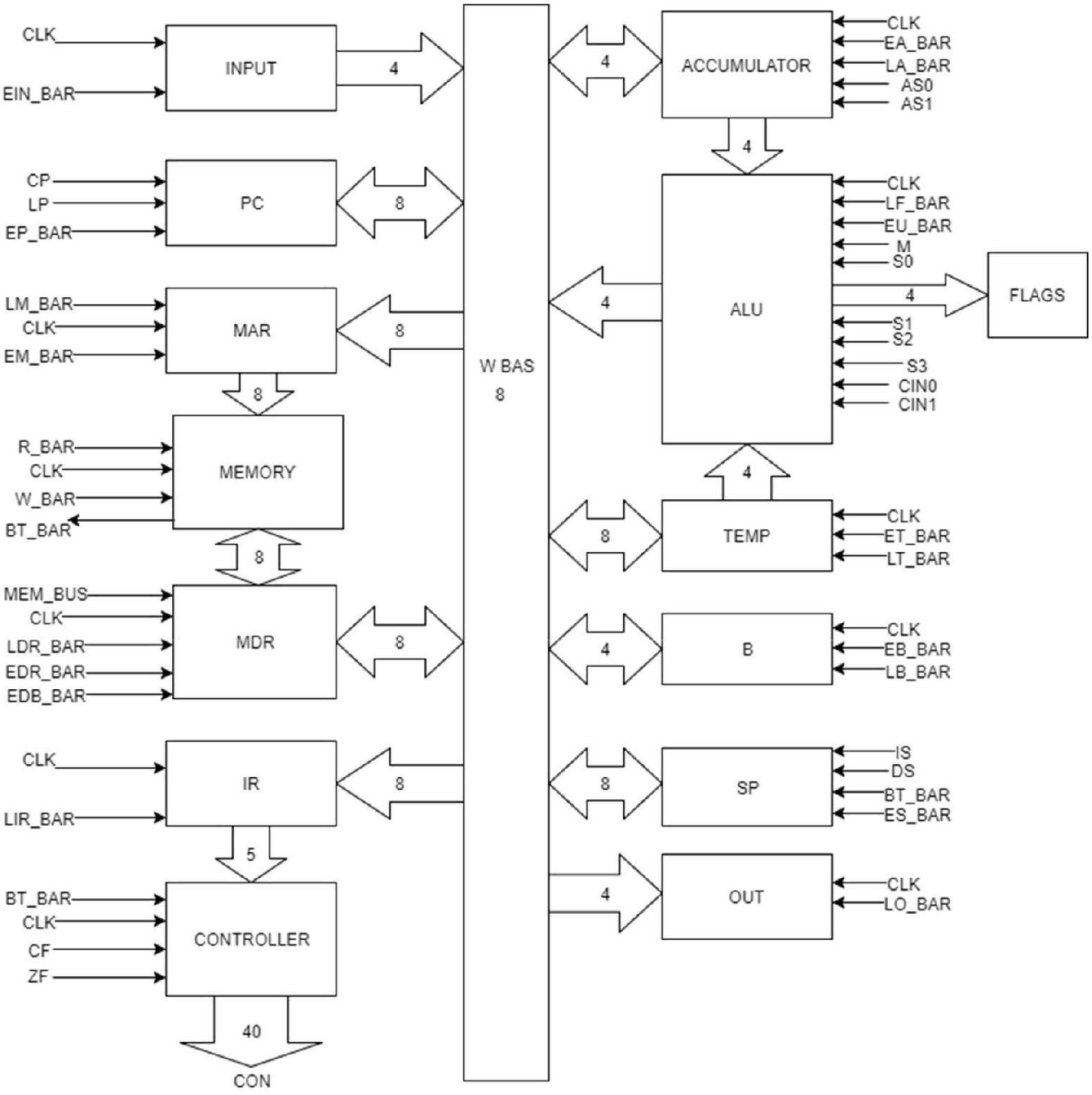


Fig: block diagram of 4-bit PC

Explanation of Blocks:

Input Register:

-Control Pin: E_IN

-4 bit register
-Used to take input from user and then store it in a buffer.
-When E_IN becomes high, the output of this register is sent to bus (W0-W3).

Program Counter:

-Control Pin: CP, L_PC, E_PC

-8 bit register
-Contains the address pointer value of the current instruction.
-After each instruction cycle it points to the next instruction in memory.
-For jump we need to load address to PC. This is also done by interacting with bus (W0-W7).
- The L_PC control pin is used to load the address of the instruction to PC.
- PC register passes its output to the bus, when E_PC is high.
-CP control pin is used to increment the PC to fetch the next instruction.

Memory Address Register:

-Control Pin: L_Mem

-8-bit register
-It takes 8-bit address as input from PC
-When L_Mem is high, address is loaded to MAR through W-bus.

Memory:

-Control Pin: W, R
Memory has two parts: RAM and Boot Loader.

RAM:

-RAM has following input lines, Mem0-Mem7, these lines connect RAM and MAR. These lines are used to provide address to RAM. And, bidirectional data line, RM0-RM7
-These lines connect MDR and RAM, using these lines either data is fed into RAM using the W signal, or Data is read from RAM using the R signal.

-To write content to RAM at a particular address, we must point at the interested address of RAM using MAR, we provide data in RM0-RM7 lines via MDR, then we control W signal to write the data.
-To read content from RAM at a particular address, we point that address using MAR, then we control the R signal to read content from RAM, later on this data is sent to MDR.

Boot loader:

-Boot loader resides inside the MEMORY module. It is responsible for loading program data from a ROM to RAM during boot period.
-Boot loader is consisted of two ROMs, and two counters. One of the ROMs contains the instructions and data and a special sentinel value FF, which is used to terminate the boot-loading process.
-The second ROM contains control words to drive the boot-loading process.
-Using one counter, we provide same address to both the program ROM and RAM, so that content in ROM at a particular address can be sent to that exact address in RAM.
-Using the second counter, we drive the boot control ROM.

Memory Data Register:

The Control Pin: L_MDR_RAM, L_MDR_BUS, E_MDR _RAM, E_MDR _BUS.

-8-bit register
- RM0-RM7 is used to read from or write into memory through data line.
-When L_MDR_RAM is HIGH, data is loaded to MDR from RAM. This is used to read data from RAM.
-When L_MDR_BUS is HIGH, data is loaded to MDR from the BUS.
-When E_MDR_BUS is HIGH, data from the MDR is sent to BUS via W0-W7.
-When E_MDR _RAM is HIGH, data from the MDR is sent to RAM via RM0-RM7.

Instruction Register:

Control Pin: L_IR

-When L_IR is HIGH, IR can load content from MDR via W0-W7 line.
-Instruction Register is an 8-bit register.
-During Fetch cycle, IR is loaded with 8-bit op-code of an instruction through W-bus (W0-W7 line) from MDR. Our 4-bit PC can execute 25 Instructions, so we only need 5 bits. IR register then sends the least significant 5 bits of the op-code to Control register using the CON0-C0N4 line.

Controller-Sequencer:

Generates the control word for each micro instructions.
It has two kinds of ROMs. They are:

1. Address ROM
2. Control ROMs

-The address ROM contains a list of starting address of the execution cycle of each macro instructions.
-We need only 5 bits because our 4-bit PC can execute 25 macro-instruction, so we need only 5 bits (2^5=32) to index all the macro instructions.
-According to these 5 op-code bits, starting address of the execution cycle of a particular micro instruction is generated.
-Our control ROMs consist of five cascading ROMs, because our control word is 40-bit length. The outputs of these ROMs are our control word for a particular micro instruction.
-The zero index refers to the first microinstruction of fetch cycle and corresponding control word is generated by the control ROMs.

-The control word for second, third and fourth micro instructions of fetch cycle are generated in the same way.
-When LOAD signal is active, our address ROM loads op-code from Instruction Register. On the next clock cycle, the counter starts counting from the address generated by address ROM which refers to the index first micro instruction of the execution cycle of that macro operation.
-We need to mark the end of a macro instruction. At the last micro instruction of each macro operation, we set a special RESET signal to 1. When this RESET signal is active, the counter resets to 0 which refers to the index of the first micro instruction of fetch cycle and the process continues.

Accumulator Register:

Control Pin: E_ACC, L_ACC, AS0, AS1

-4-bit register that can load data from bus or send data to bus using W0-W3 line. It also passes data to Arithmetic Logic Unit (ALU) through A0-A3 line to perform different arithmetic and logic operations.
-When E_ACC is HIGH, the data of Accumulator register is sent to W-bus.
-When L_ACC is low, Accumulator register loads data from W-bus.
-AS0 and AS1 are used to shift the content of Accumulator register.
-When both AS0 and AS1 are high, Accumulator contents are unchanged. This is why we need to set these two control pins every time we need to load data into accumulator.
-When AS0 is high and AS1 is low, Accumulator contents are shifted right.
- When AS0 is low and AS1 is high, Accumulator contents are shifted left.
-CACC is used to store the CARRY.

Arithmetic Logic Unit:

Control Pin: L_FLAG, E_ALU, M, S0, S1, S2, S3, CIN0, CIN1

ALU takes data from Accumulator register through A0-A3 line and from Temp Register through T0-T3 line and performs various arithmetic and logic operations.

ALU uses five control bits M, S0, S1, S2 and S3 to do various operations on word A and word B.
Data inputs for ALU come from Accumulator register (ACC) and Temp register (TEMP).

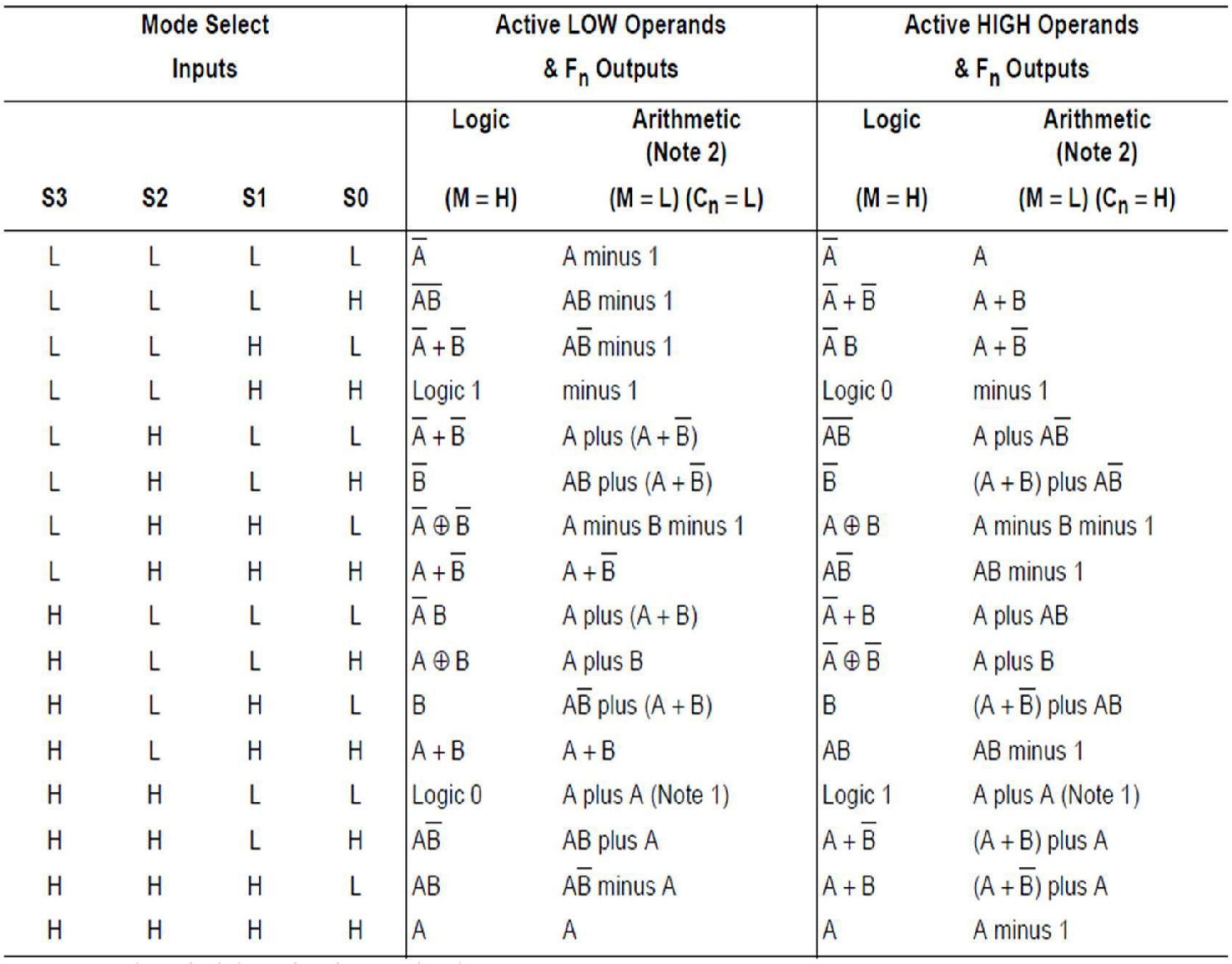
The function table is given in the next page.

-Arithmetic operations like ADD, ADC, SUB, SBB require us to put different values in CN bit of the ALU.
-In an ADD operation, we need to put 0 in CN pin.
-In an ADC operation, we need to put carry flag in CN pin.
-In a SUB operation we need to put 1 in CN.
-In a SBB operation we need to put the invert of carry flag in CN pin.
-To ensure these, we introduce two pins CIN0 and CIN1 to control the content of CN bit of ALU.

The combination is given below:

CIN1	CIN0	CN	Operation
0	0	0	ADD
0	1	Carry Flag (CF)	ADC
1	0	1	SUB
1	1	Invert of Carry Flag (CF_BAR)	SBB

-We store the carry flag (CF), zero flag (ZF), sign flag (SF) and overflow flag (OF) in a register.
-Carry, sign and zero flag can be found from ALU.
-For overflow flag (OF) we use a simple logic which is, when two positive numbers are added and the result is negative or when two negative numbers are added and the result is positive, overflow occurs.



Mode Select Inputs				Active LOW Operands & F_n Outputs		Active HIGH Operands & F_n Outputs	
				Logic	Arithmetic (Note 2)	Logic	Arithmetic (Note 2)
S3	S2	S1	S0	(M = H)	(M = L) (C_n = L)	(M = H)	(M = L) (C_n = H)
L	L	L	L	$\overline{A}$	A minus 1	$\overline{A}$	A
L	L	L	H	$\overline{AB}$	AB minus 1	$\overline{A} + \overline{B}$	$A + B$
L	L	H	L	$\overline{A} + \overline{B}$	$A\overline{B}$ minus 1	$\overline{A}\,B$	$A + \overline{B}$
L	L	H	H	Logic 1	minus 1	Logic 0	minus 1
L	H	L	L	$\overline{A} + \overline{B}$	A plus $(A + \overline{B})$	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	AB plus $(A + \overline{B})$	$\overline{B}$	$(A + B)$ plus $A\overline{B}$
L	H	H	L	$\overline{A} \oplus \overline{B}$	A minus B minus 1	$A \oplus B$	A minus B minus 1
L	H	H	H	$A + \overline{B}$	$A + \overline{B}$	$A\overline{B}$	AB minus 1
H	L	L	L	$\overline{A}\,B$	A plus $(A + B)$	$\overline{A} + B$	A plus AB
H	L	L	H	$A \oplus B$	A plus B	$\overline{A} \oplus \overline{B}$	A plus B
H	L	H	L	B	$A\overline{B}$ plus $(A + B)$	B	$(A + \overline{B})$ plus AB
H	L	H	H	$A + B$	$A + B$	AB	AB minus 1
H	H	L	L	Logic 0	A plus A (Note 1)	Logic 1	A plus A (Note 1)
H	H	L	H	$A\overline{B}$	AB plus A	$A + \overline{B}$	$(A + B)$ plus A
H	H	H	L	AB	$A\overline{B}$ minus A	$A + B$	$(A + \overline{B})$ plus A
H	H	H	H	A	A	A	A minus 1

Temporary Register:

Control Pin: L_TEMP

-8-bit register to store 4-bit data for our instructions
-This register can load data from W-bus using W0-W3 line.
-It sends data to Arithmetic Logic Unit (ALU) through T0-T3 line continuously to perform different arithmetic and logic operations.
-When L_TEMP is HIGH, Temp register loads data from W-bus.

B Register:

Associated Control Pin: L_B, E_B

- 4-bit register that can load data from W-bus or send data to W-bus using W0-W3 line.
-When E_B is HIGH, the data of B register is sent to W-bus.
-When L_B is HIGH, B register loads data from W-bus.

Stack Pointer:

Control Pin: E_SP, INC_SP, DEC_SP

-It is an 8-bit register that stores the address of the last program request in a stack.
-Initially, the Stack Pointer points to FF-the last memory location.
-For push operation, SP is first decremented and then loaded to MAR so that data can be written in that location.
-If a POP operation is done, SP is loaded to MAR so that data in that address can be read and after that value of SP is incremented.

-When E_SP is HIGH, the contents of Stack Pointer (SP) is sent to bus through W0-W7 line.
-When INC_SP is HIGH, the value of SP is incremented.
-When DEC_SP is HIGH, the value of SP is decremented.

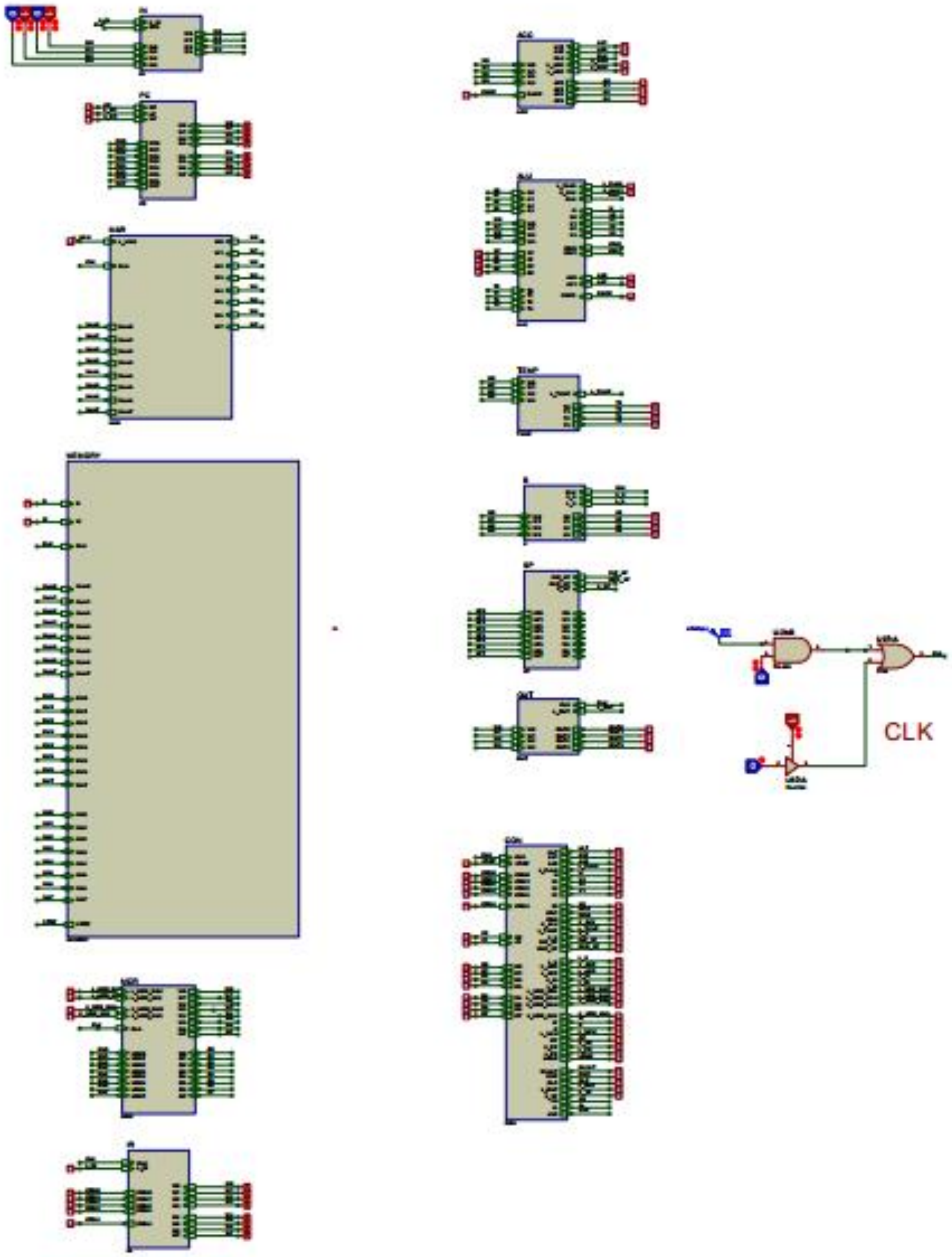
Output Register:

-Control Pin: L_OUT

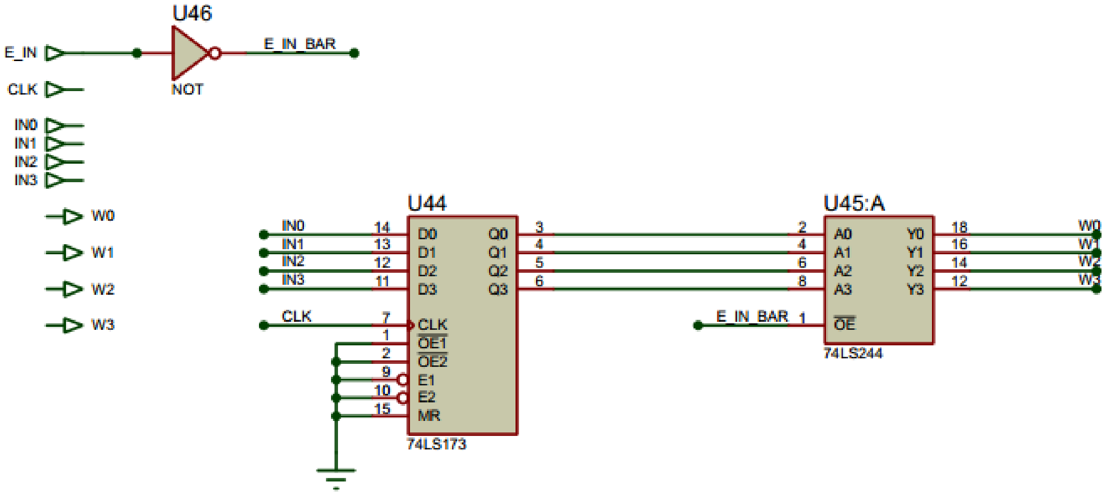
-It is 4-bit register that can load data from W-bus through W0-W3 line to show the result of different operations.

When L_OUT is HIGH, OUT register loads data from W-bus.

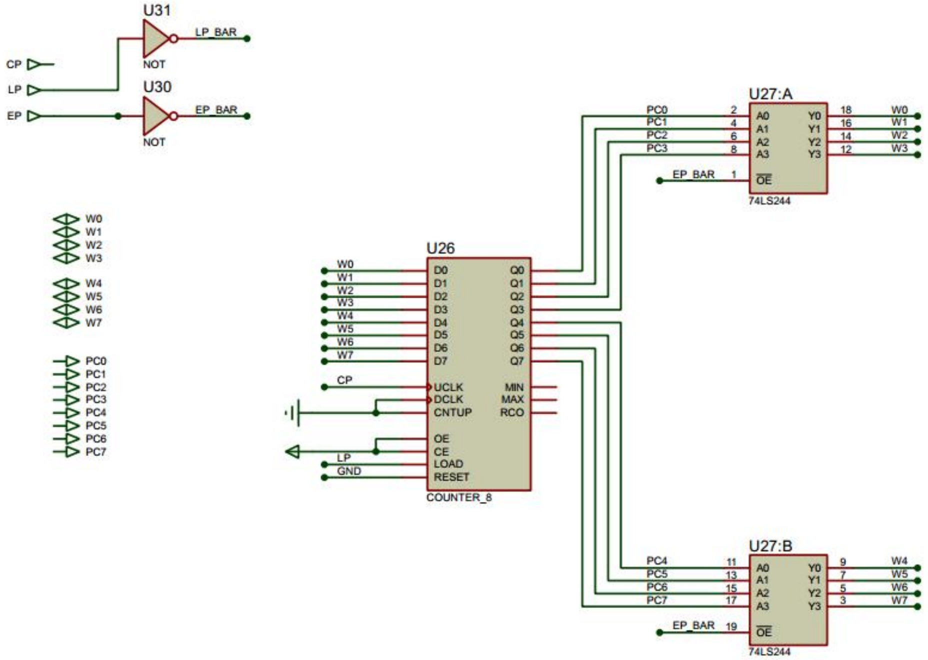
Circuit Diagrams:



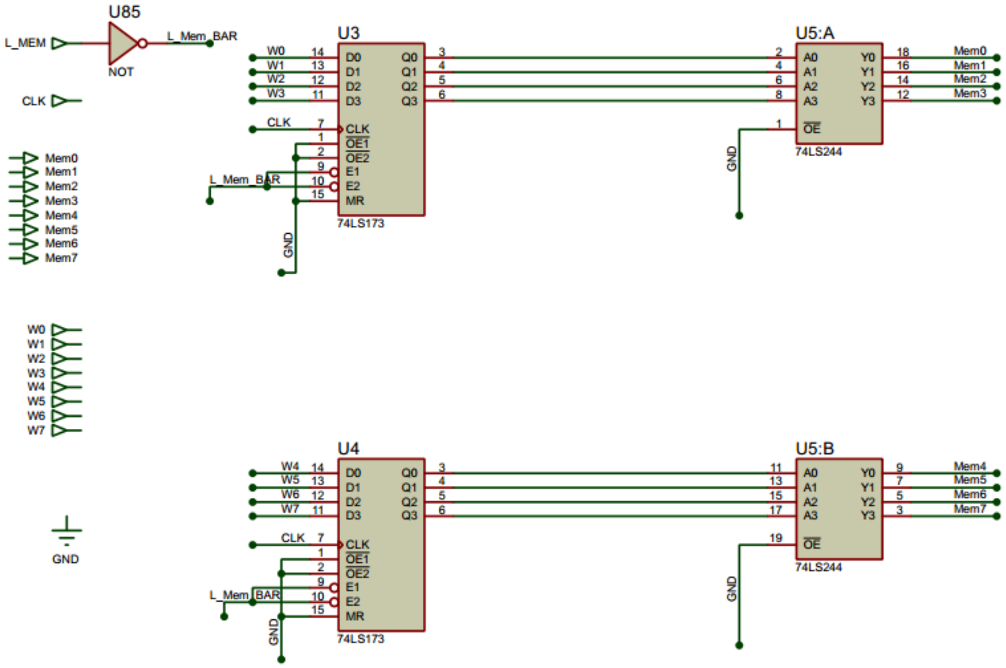
4 BIT PC


U46
E_IN
NOT
E_IN_BAR
CLK
IN0
IN1
IN2
IN3
W0
W1
W2
W3
U44
IN0
IN1
IN2
IN3
14
13
12
11
D0
D1
D2
D3
Q0
Q1
Q2
Q3
3
4
5
6
CLK
7
CLK
1
OE1
2
OE2
9
E1
10
E2
15
MR
74LS173
U45:A
2
4
6
8
A0
A1
A2
A3
Y0
Y1
Y2
Y3
18
16
14
12
W0
W1
W2
W3
E_IN_BAR
1
OE
74LS244

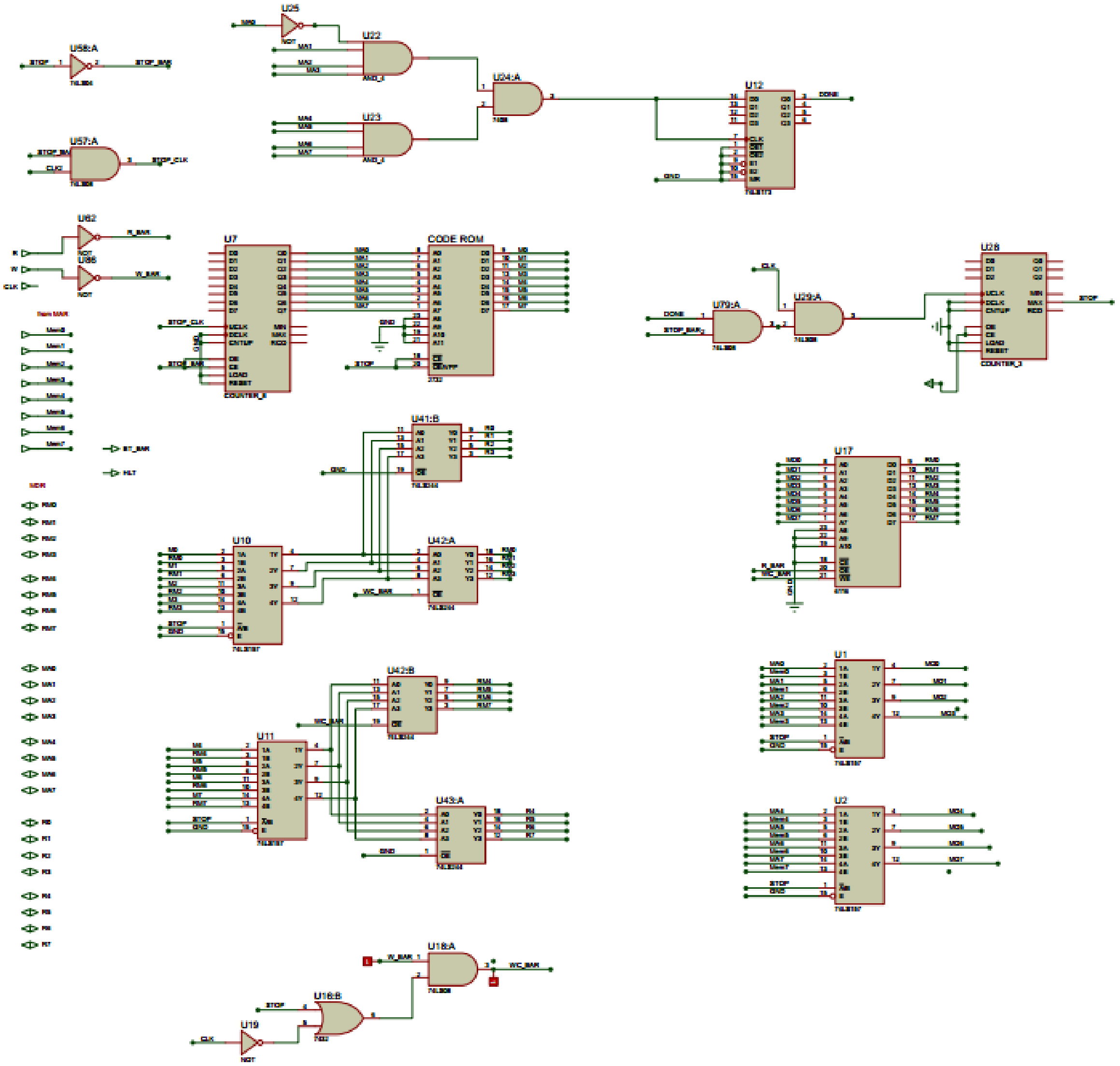
IN


U31
NOT
LP_BAR
CP
LP
EP
U30
NOT
EP_BAR
W0
W1
W2
W3
W4
W5
W6
W7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
U26
W0
W1
W2
W3
W4
W5
W6
W7
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
CP
UCLK
DCLK
CNTUP
MIN
MAX
RCO
OE
CE
LOAD
RESET
LP
GND
COUNTER_8
U27:A
PC0
PC1
PC2
PC3
2
4
6
8
A0
A1
A2
A3
Y0
Y1
Y2
Y3
18
16
14
12
W0
W1
W2
W3
EP_BAR
1
OE
74LS244
U27:B
PC4
PC5
PC6
PC7
11
13
15
17
A0
A1
A2
A3
Y0
Y1
Y2
Y3
9
7
5
3
W4
W5
W6
W7
EP_BAR
19
OE
74LS244

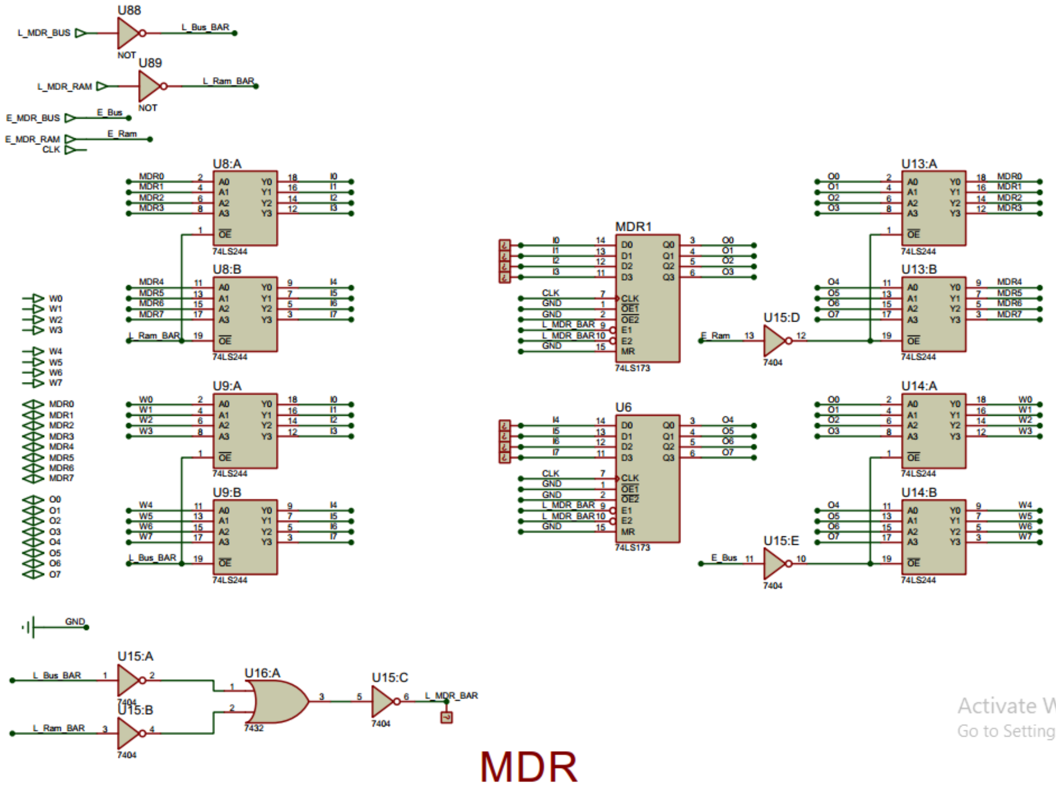
PC


U85
L_MEM
L_Mem_BAR
NOT
CLK
Mem0
Mem1
Mem2
Mem3
Mem4
Mem5
Mem6
Mem7
W0
W1
W2
W3
W4
W5
W6
W7
GND
U3
D0
D1
D2
D3
Q0
Q1
Q2
Q3
CLK
OE1
OE2
E1
E2
MR
74LS173
U5:A
A0
A1
A2
A3
Y0
Y1
Y2
Y3
OE
74LS244
U4
74LS173
U5:B
74LS244

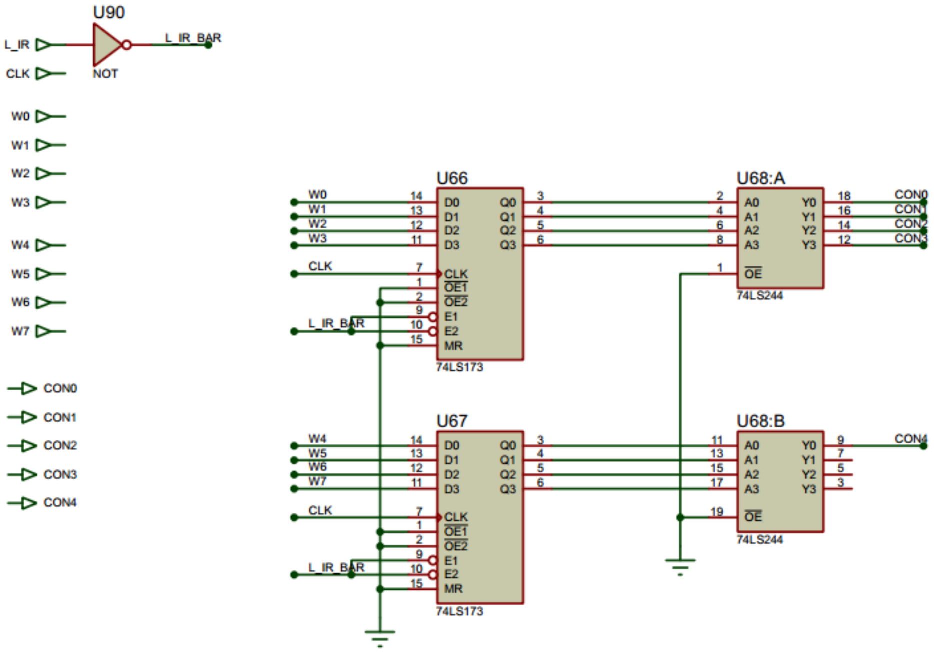
MAR


U58:A
STOP
STOP_BAR
U57:A
STOP_CLK
U25
NOT
U22
AND_4
U23
AND_4
U24:A
U12
DONE
GND
U62
R_BAR
U66
W_BAR
R
W
CLK
from MAR
U7
COUNTER_8
STOP_CLK
STOP_BAR
CODE ROM
2732
STOP
GND
U79:A
U29:A
U28
COUNTER_3
DONE
STOP_BAR
STOP
U41:B
74LS244
GND
U10
74LS157
U42:A
74LS244
WC_BAR
U17
6116
R_BAR
WC_BAR
U1
74LS157
U42:B
U11
74LS157
WC_BAR
U43:A
GND
U2
74LS157
U18:A
W_BAR
WC_BAR
U16:B
STOP
U19
CLK
NOT

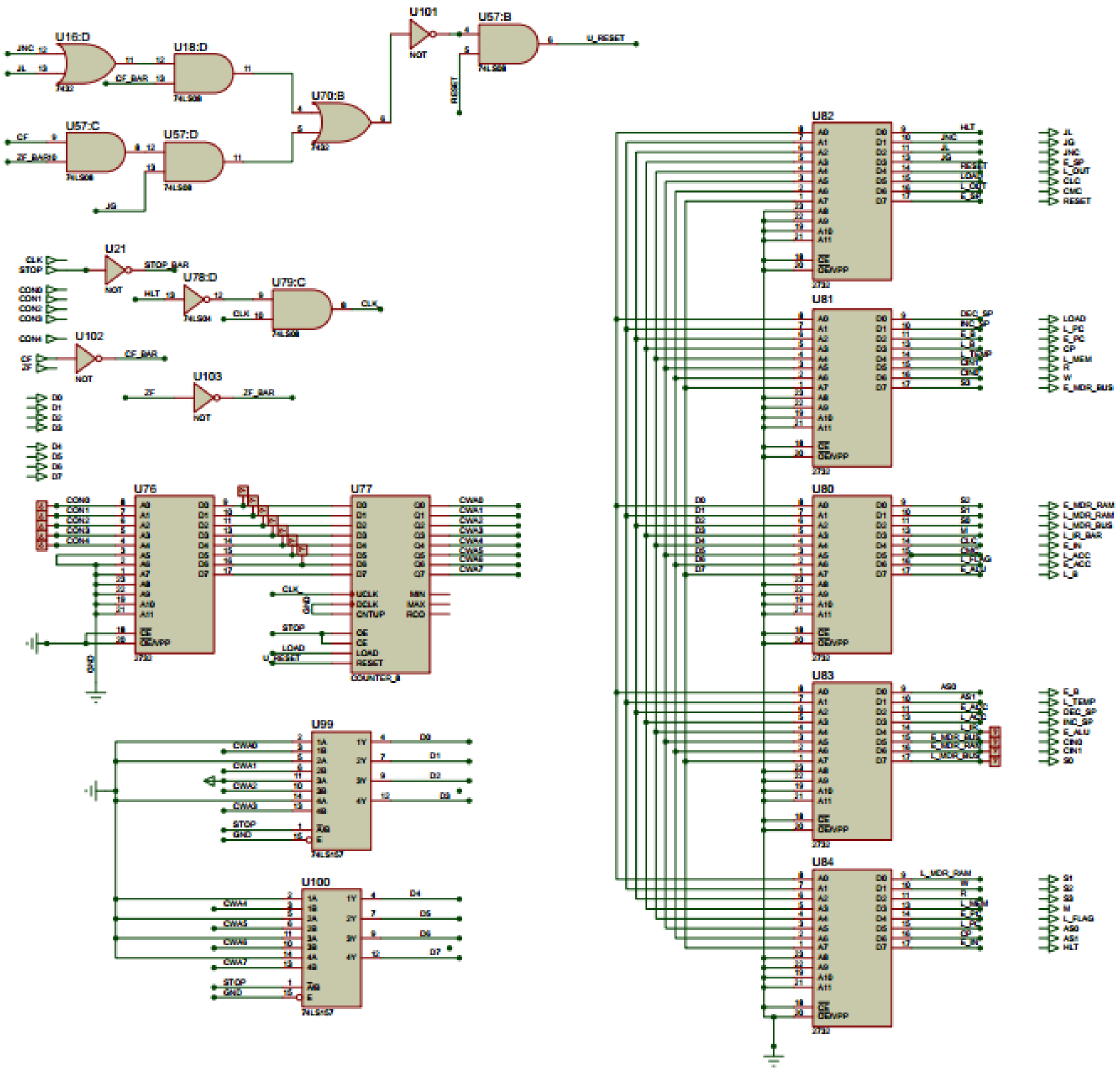
RAM


U88
L_MDR_BUS
NOT
L_Bus_BAR
U89
L_MDR_RAM
NOT
L_Ram_BAR
E_MDR_BUS
E_Bus
E_MDR_RAM
E_Ram
CLK
U8:A
MDR0
MDR1
MDR2
MDR3
A0
A1
A2
A3
Y0
Y1
Y2
Y3
OE
I0
I1
I2
I3
74LS244
U8:B
MDR4
MDR5
MDR6
MDR7
I4
I5
I6
I7
L_Ram_BAR
W0
W1
W2
W3
W4
W5
W6
W7
O0
O1
O2
O3
O4
O5
O6
O7
U9:A
U9:B
L_Bus_BAR
GND
MDR1
D0
D1
D2
D3
Q0
Q1
Q2
Q3
CLK
OE1
OE2
E1
E2
MR
L_MDR_BAR
74LS173
U6
U13:A
U13:B
U14:A
U14:B
U15:D
U15:E
7404
E_Ram
E_Bus
U15:A
U15:B
U16:A
7432
U15:C
L_MDR_BAR

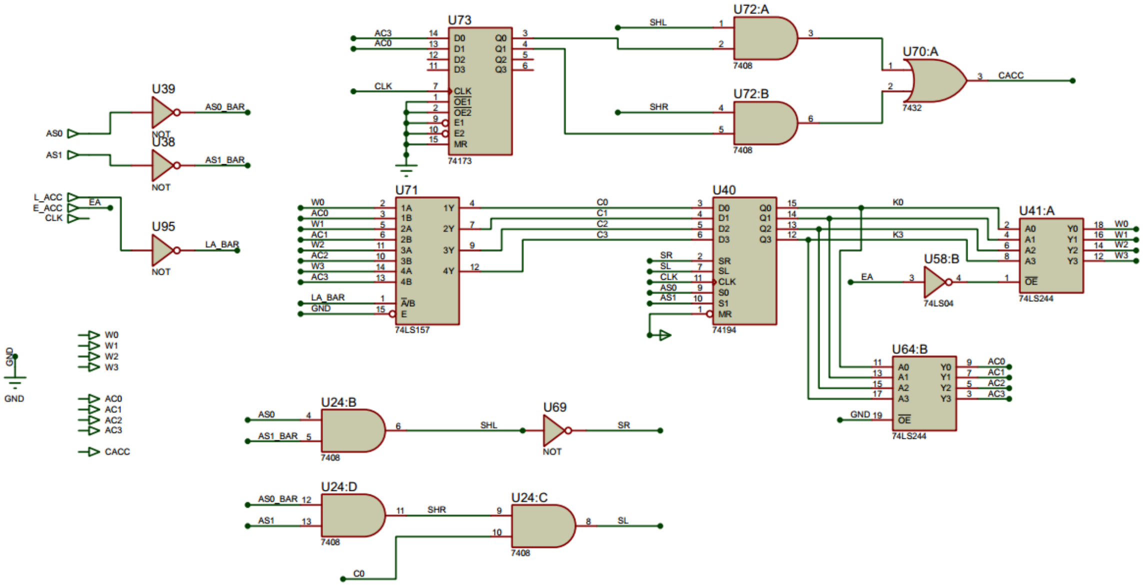
MDR


U90
L_IR
L_IR_BAR
NOT
CLK
W0
W1
W2
W3
W4
W5
W6
W7
CON0
CON1
CON2
CON3
CON4
U66
U67
74LS173
D0
D1
D2
D3
Q0
Q1
Q2
Q3
CLK
OE1
OE2
E1
E2
MR
U68:A
U68:B
74LS244
A0
A1
A2
A3
Y0
Y1
Y2
Y3
OE

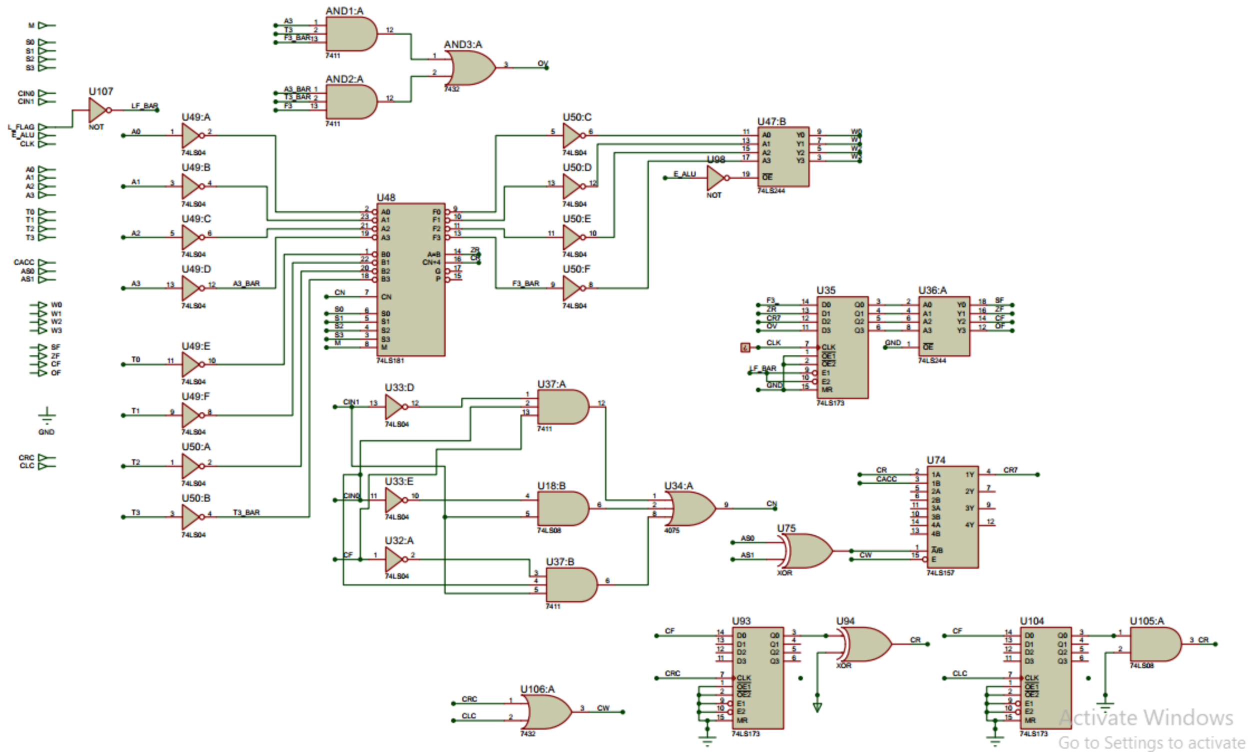
IR


U16:D
JNC
JL
7432
U18:D
CF_BAR
74LS08
U57:C
CF
ZF_BAR
74LS08
U57:D
74LS08
JG
U70:B
7432
U101
NOT
U57:B
RESET
74LS08
U_RESET
CLK
STOP
U21
NOT
STOP_BAR
CON0
CON1
CON2
CON3
U78:D
HLT
74LS04
U79:C
CLK
74LS08
CLK
CON4
U102
CF
ZF
NOT
CF_BAR
U103
ZF
NOT
ZF_BAR
D0
D1
D2
D3
D4
D5
D6
D7
U76
CON0
CON1
CON2
CON3
CON4
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
CE
OE/VPP
2732
GND
D0
D1
D2
D3
D4
D5
D6
D7
U77
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
CLK
GND
STOP
LOAD
U_RESET
CWA0
CWA1
CWA2
CWA3
CWA4
CWA5
CWA6
CWA7
U99
CWA0
CWA1
CWA2
CWA3
STOP
GND
1A
1B
2A
2B
3A
3B
4A
4B
A/B
E
1Y
2Y
3Y
4Y
74LS157
D0
D1
D2
D3
U100
CWA4
CWA5
CWA6
CWA7
STOP
GND
74LS157
D4
D5
D6
D7
U82
HLT
JNC
JL
JG
RESET
LOAD
L_OUT
E_SP
2732
JL
JG
JNC
E_SP
L_OUT
CLC
CMC
RESET
U81
DEC_SP
INC_SP
E_B
L_B
L_TEMP
CIN1
CIN0
S0
2732
LOAD
L_PC
E_PC
CP
L_MEM
R
W
E_MDR_BUS
U80
D0
D1
D2
D3
D4
D5
D6
D7
S0
S1
S2
M
CLC
CMC
L_FLAG
E_ALU
2732
E_MDR_RAM
L_MDR_RAM
L_MDR_BUS
L_IR_BAR
E_IN
L_ACC
E_ACC
L_B
U83
AS0
AS1
E_ACC
L_ACC
L_IR
E_MDR_BUS
E_MDR_RAM
L_MDR_BUS
2732
E_B
L_TEMP
DEC_SP
INC_SP
E_ALU
CIN0
CIN1
S0
U84
L_MDR_RAM
W
R
L_MEM
E_PC
L_PC
CP
E_IN
2732
S1
S2
S3
M
L_FLAG
AS0
AS1
HLT

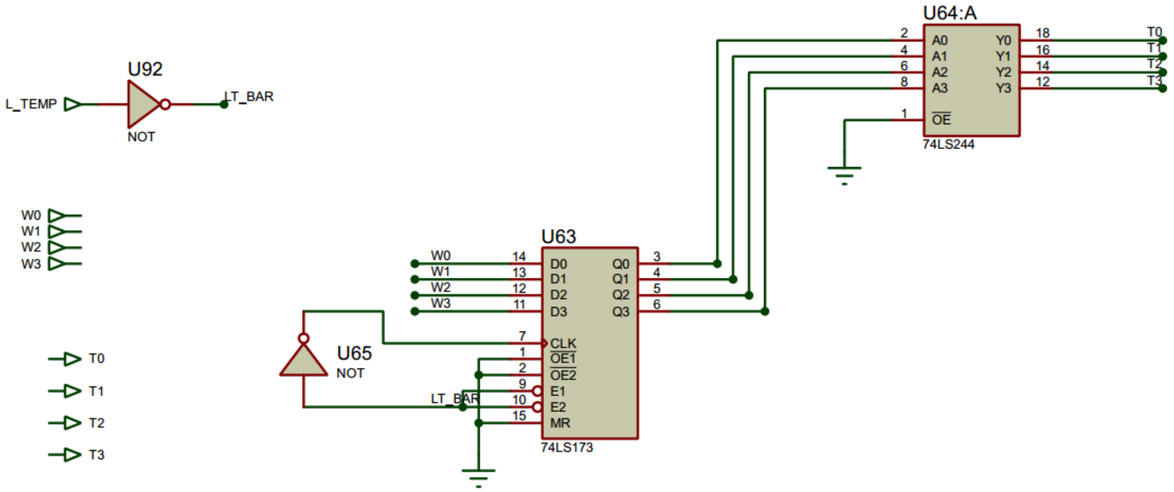
CONTROLLER


U73
AC3 14 D0 Q0 3
AC0 13 D1 Q1 4
12 D2 Q2 5
11 D3 Q3 6
CLK 7 CLK
1 OE1
2 OE2
9 E1
10 E2
15 MR
74173
U72:A
SHL 1
2
3
7408
U72:B
SHR 4
5
6
7408
U70:A
1
2
3 CACC
7432
U39
AS0
AS0_BAR
NOT
U38
AS1
AS1_BAR
NOT
L_ACC
E_ACC
CLK
EA
U95
LA_BAR
NOT
U71
W0 2 1A 1Y 4
AC0 3 1B
W1 5 2A 2Y 7
AC1 6 2B
W2 11 3A 3Y 9
AC2 10 3B
W3 14 4A 4Y 12
AC3 13 4B
LA_BAR 1 A/B
GND 15 E
74LS157
C0
C1
C2
C3
U40
3 D0 Q0 15
4 D1 Q1 14
5 D2 Q2 13
6 D3 Q3 12
SR 2 SR
SL 7 SL
CLK 11 CLK
AS0 9 S0
AS1 10 S1
1 MR
74194
K0
K3
U41:A
2 A0 Y0 18 W0
4 A1 Y1 16 W1
6 A2 Y2 14 W2
8 A3 Y3 12 W3
1 OE
74LS244
U58:B
EA 3 4
74LS04
U64:B
11 A0 Y0 9 AC0
13 A1 Y1 7 AC1
15 A2 Y2 5 AC2
17 A3 Y3 3 AC3
GND 19 OE
74LS244
GND
GND
W0
W1
W2
W3
AC0
AC1
AC2
AC3
CACC
U24:B
AS0 4
AS1_BAR 5
6 SHL
7408
U69
SR
NOT
U24:D
AS0_BAR 12
AS1 13
11 SHR
7408
U24:C
9
10
8 SL
7408
C0

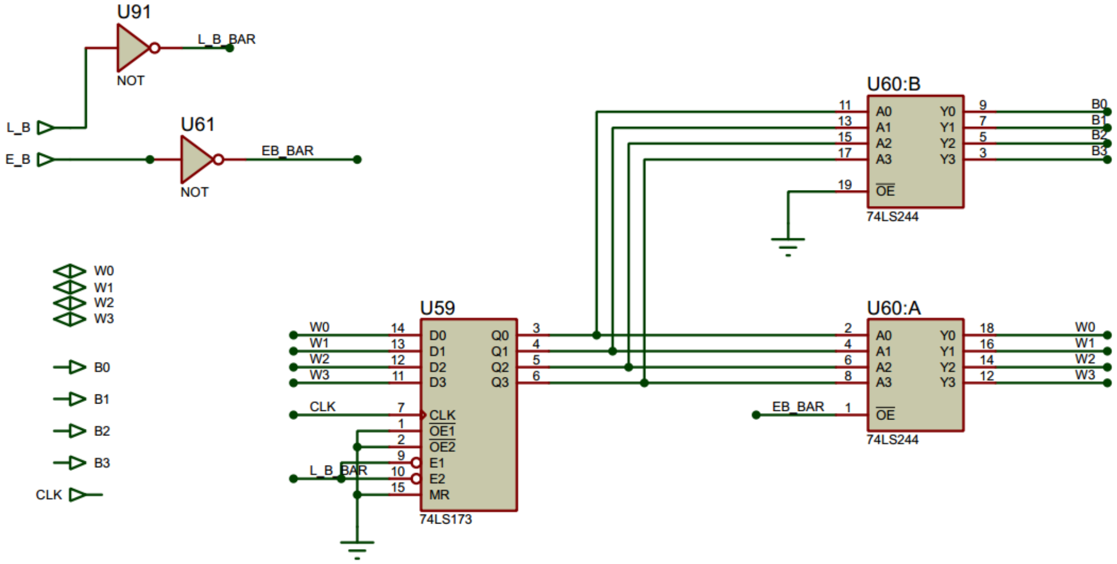
ACC


M
S0
S1
S2
S3
CIN0
CIN1
L_FLAG
E_ALU
CLK
A0
A1
A2
A3
T0
T1
T2
T3
CACC
AS0
AS1
W0
W1
W2
W3
SF
ZF
CF
OF
GND
CRC
CLC
U107
NOT
LF_BAR
AND1:A
A3
T3
F3_BAR
7411
AND2:A
A3_BAR
T3_BAR
F3
7411
AND3:A
7432
OV
U49:A
U49:B
U49:C
U49:D
A3_BAR
U49:E
U49:F
U50:A
U50:B
T3_BAR
74LS04
U48
A0
A1
A2
A3
B0
B1
B2
B3
CN
S0
S1
S2
S3
M
F0
F1
F2
F3
A=B
CN+4
G
P
ZR
CR
74LS181
U50:C
U50:D
U50:E
U50:F
F3_BAR
U47:B
OE
Y0
Y1
Y2
Y3
74LS244
U98
E_ALU
NOT
U35
F3_
ZR
CR7
OV
CLK
LF_BAR
GND
D0
D1
D2
D3
Q0
Q1
Q2
Q3
OE1
OE2
E1
E2
MR
74LS173
U36:A
74LS244
U33:D
CIN1
U37:A
7411
U33:E
CIN0
U18:B
74LS08
U32:A
CF
U37:B
7411
U34:A
4075
CN
U74
CR
CACC
1A
1B
2A
2B
3A
3B
4A
4B
A/B
E
1Y
2Y
3Y
4Y
CR7
74LS157
U75
AS0
AS1
XOR
CW
U106:A
CRC
CLC
7432
CW
U93
CF
CRC
74LS173
U94
XOR
CR
U104
CF
CLC
74LS173
U105:A
74LS08
CR

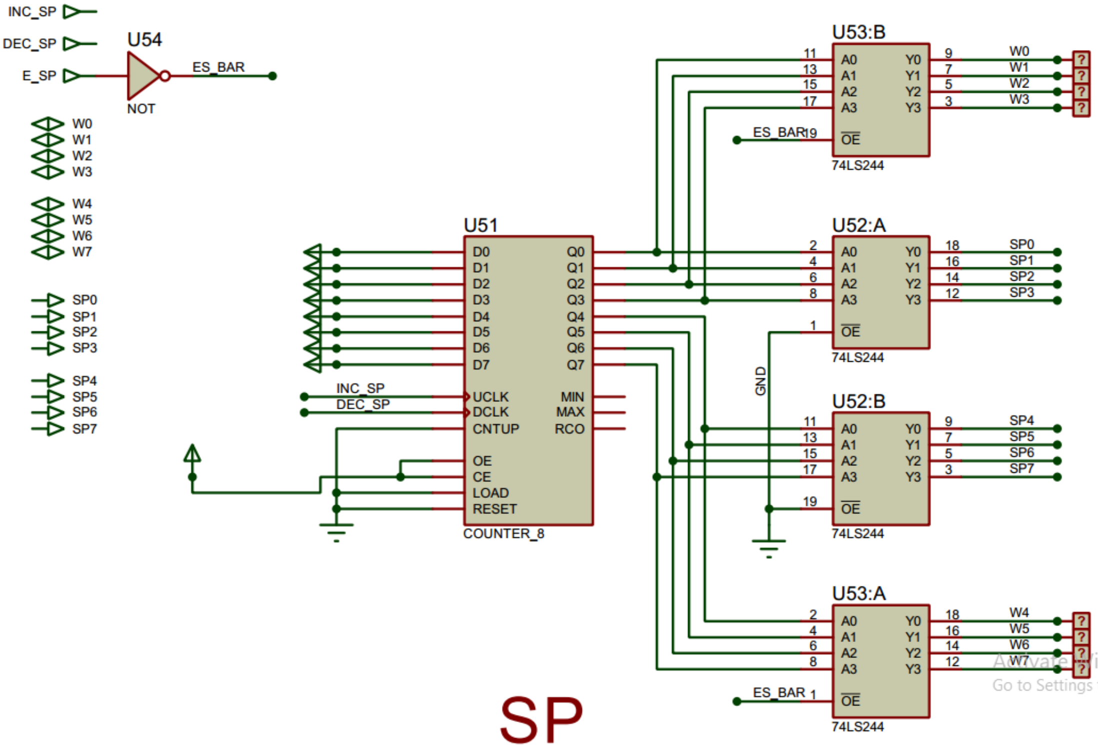
ALU


U92
L_TEMP
NOT
LT_BAR
W0
W1
W2
W3
T0
T1
T2
T3
U65
NOT
LT_BAR
U63
W0
W1
W2
W3
14
13
12
11
D0
D1
D2
D3
Q0
Q1
Q2
Q3
3
4
5
6
7
1
2
9
10
15
CLK
OE1
OE2
E1
E2
MR
74LS173
U64:A
2
4
6
8
1
A0
A1
A2
A3
OE
Y0
Y1
Y2
Y3
18
16
14
12
T0
T1
T2
T3
74LS244

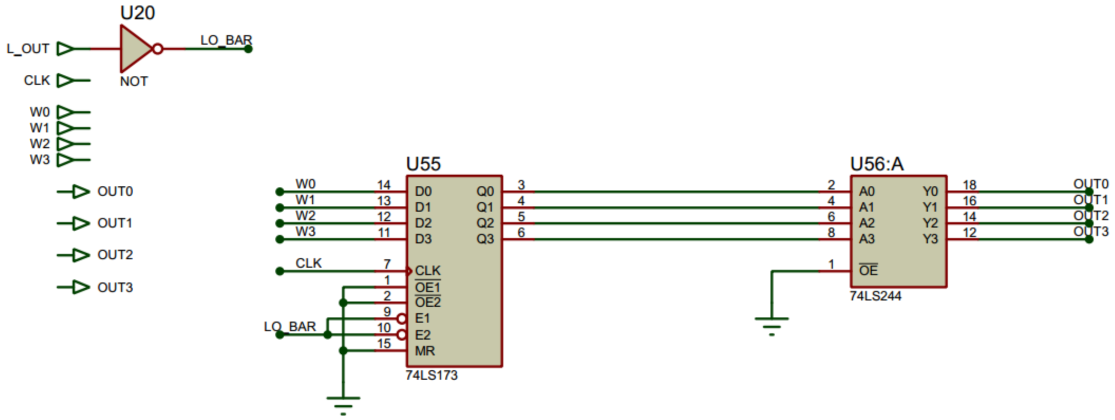
TEMP


U91
NOT
L_B_BAR
L_B
E_B
U61
NOT
EB_BAR
W0
W1
W2
W3
B0
B1
B2
B3
CLK
U59
D0
D1
D2
D3
CLK
OE1
OE2
E1
E2
MR
Q0
Q1
Q2
Q3
74LS173
L_B_BAR
U60:B
A0
A1
A2
A3
OE
Y0
Y1
Y2
Y3
74LS244
U60:A
A0
A1
A2
A3
OE
Y0
Y1
Y2
Y3
74LS244
EB_BAR

B


INC_SP
DEC_SP
E_SP
U54
NOT
ES_BAR
W0
W1
W2
W3
W4
W5
W6
W7
SP0
SP1
SP2
SP3
SP4
SP5
SP6
SP7
U51
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
INC_SP
DEC_SP
UCLK
DCLK
CNTUP
MIN
MAX
RCO
OE
CE
LOAD
RESET
COUNTER_8
U53:B
11
13
15
17
A0
A1
A2
A3
Y0
Y1
Y2
Y3
9
7
5
3
W0
W1
W2
W3
ES_BAR
19
OE
74LS244
U52:A
2
4
6
8
A0
A1
A2
A3
Y0
Y1
Y2
Y3
18
16
14
12
SP0
SP1
SP2
SP3
1
OE
74LS244
GND
U52:B
11
13
15
17
A0
A1
A2
A3
Y0
Y1
Y2
Y3
9
7
5
3
SP4
SP5
SP6
SP7
19
OE
74LS244
U53:A
2
4
6
8
A0
A1
A2
A3
Y0
Y1
Y2
Y3
18
16
14
12
W4
W5
W6
W7
ES_BAR
1
OE
74LS244
?
?
?
?
?
?
?
?

SP


U20
L_OUT
LO_BAR
NOT
CLK
W0
W1
W2
W3
OUT0
OUT1
OUT2
OUT3
U55
14
13
12
11
7
1
2
9
10
15
D0
D1
D2
D3
CLK
OE1
OE2
E1
E2
MR
Q0
Q1
Q2
Q3
3
4
5
6
74LS173
U56:A
2
4
6
8
1
A0
A1
A2
A3
OE
Y0
Y1
Y2
Y3
18
16
14
12
74LS244

OUT

Explanation of all instructions:

Description	Total T-States	T-State	Micro-Operation	Active	CROM0	CROM1	CROM2	CROM3	CROM4
Fetch	4	T1	MAR ← PC	L_MEM, E_PC	10	18	ff	df	0
		T2	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T3	IR ← MDR	E_MDR_BUS, L_IR	10	18	ff	9b	c8
		T4	LOAD from IR	LOAD	10	18	ff	df	ca
ACC ← RAM[address]	9	T5	MAR ← PC	L_MEM, E_PC	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R	10	18	ff	ce	98
		T7	MAR ← MDR	L_MEM, E_MDR_BUS,	10	18	ff	db	8
		T8	MDR ← RAM[MAR]	L_MDR_RAM, R	10	18	ff	ce	88
		T9	ACC ← MDR	L_ACC, E_MDR_BUS, AS0, AS1, RESET	70	18	fe	db	c9
Halts execution	5	T5	HALT	HLT	90	18	ff	df	c8
ACC ← input_port	5	T5	ACC ← input_port	L_ACC, AS0, AS1, E_IN, RESET	70	18	fe	5f	c9
RAM[address] ← ACC	12	T5	MAR ← PC	L_MEM, E_PC	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R	10	18	ff	ce	98
		T7	MAR ← MDR	L_MEM, E_MDR_BUS,	10	18	ff	db	8
		T8	MDR ← ACC	L_MDR_RAM, E_ACC, E_MDR_BUS,	10	18	fd	ef	88
		T9	RAM[MAR] ← MDR	E_MDR_RAM,	10	18	ff	d7	88
		T10		W, E_MDR_RAM,	10	18	ff	d5	88
		T11		E_MDR_RAM,	10	18	ff	d7	88
		T12		R, RESET,	10	18	ff	de	89
output_port ← ACC	5	T5	output_port ← ACC	L_OUT, E_ACC, RESET	10	18	ed	df	c9
B ← ACC	5	T5	B ← ACC	L_B, E_ACC, RESET	10	18	f9	df	c9
ACC ← B	5	T5	ACC ← B	L_ACC, AS0, AS1, E_B, RESET	70	18	f6	df	c9
ACC ← immediate	7	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	ACC ← MDR	L_ACC, AS0, AS1, E_MDR_BUS, RESET	70	18	fe	db	c9
NOP	5	T5	NOP	RESET	10	18	ff	df	c9
Jump to the address	8	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	PC ← MDR	LP, E_MDR_BUS	10	18	ff	db	cc
		T8		CP, LP, RESET	10	18	ff	df	dd
ACC ← ACC + B	7	T5	TEMP ← B	L_TEMP, E_B, S3, S0	14	98	b7	df	c8
		T6	ACC ← ACC + TEMP	S3, S0	14	98	ff	df	c8

		T7		L_ACC, AS0, AS1, E_ALU, S3, S0, L_FLAG, RESET	64	90	fe	df	c9
ACC ← ACC + B + C	7	T5	TEMP ← B	L_TEMP, E_B, S3, S0, CIN0	14	b8	b7	df	c8
		T6	ACC ← ACC + TEMP + C	S3, S0, CIN0	14	b8	ff	df	c8
		T7		L_ACC, AS0, AS1, E_ALU, S3, S0, CIN0, L_FLAG, RESET	64	b0	fe	df	c9
ACC ← ACC - B	7	T5	TEMP ← B	L_TEMP, E_B, S2, S1, CIN1	13	58	b7	df	c8
		T6	ACC ← ACC - TEMP	S2, S1, CIN1	13	58	ff	df	c8
		T7		L_ACC, AS0, AS1, E_ALU, S2, S1, CIN1, L_FLAG, RESET	63	50	fe	df	c9
ACC ← ACC - B - BO	7	T5	TEMP ← B	L_TEMP, E_B, S2, S1, CIN1, CIN0	13	78	b7	df	c8
		T6	ACC ← ACC - TEMP - BO	S2, S1, CIN1, CIN0	13	78	ff	df	c8
		T7		L_ACC, AS0, AS1, E_ALU, S2, S1, CIN1, CIN0, L_FLAG, RESET	63	70	fe	df	c9
ACC ← ACC + immediate	9	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	TEMP ← MDR	L_TEMP, E_MDR_BUS, S3, S0	14	98	bf	db	c8
		T8	ACC ← ACC + TEMP	S3, S0	14	98	ff	df	c8
		T9		L_ACC, AS0, AS1, E_ALU, S3, S0, L_FLAG, RESET	64	90	fe	df	c9
ACC ← ACC - RAM[address]	11	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	MAR ← MDR	L_MEM, E_MDR_BUS,	10	18	ff	db	8
		T8	MDR ← RAM[MAR]	L_MDR_RAM, R,	10	18	ff	ce	88
		T9	TEMP ← MDR	L_TEMP, E_MDR_BUS, S2, S1, CIN1	13	58	bf	db	c8
		T10	ACC ← ACC - TEMP	S2, S1, CIN1	13	58	ff	df	c8
		T11		L_ACC, AS0, AS1, E_ALU, S2, S1, CIN1, L_FLAG, RESET	63	50	fe	df	c9
ACC - B	7	T5	TEMP ← B	L_TEMP, E_B, S2, S1, CIN1	13	58	b7	df	c8
		T6	ACC - TEMP	S2, S1, CIN1	13	58	ff	df	c8
		T7		S2, S1, CIN1, L_FLAG, RESET	3	58	ff	df	c9
ACC & B	6	T5	TEMP ← B	L_TEMP, E_B	10	18	b7	df	c8
		T6	ACC & TEMP	L_FLAG, M, S3, S2, S1, RESET	f	18	ff	df	c9
ACC ← ACC & B	6	T5	TEMP ← B	L_TEMP, E_B	10	18	b7	df	c8
		T6	ACC ← ACC & TEMP	L_ACC, AS0, AS1, E_ALU, M, S3, S2, S1, RESET	7f	10	fe	df	c9
ACC ← ACC ⊕ immediate	8	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	TEMP ← MDR	L_TEMP, E_MDR_BUS	10	18	bf	db	c8
		T8	ACC ← ACC ⊕ TEMP	L_ACC, AS0, AS1, E_ALU, M, S3, S0, RESET	7c	90	fe	df	c9
RAM[SP] ← ACC	10	T5	SP ← SP - 1, MDR ← ACC	DEC_SP, L_MDR_RAM, E_ACC, E_MDR_BUS,	10	19	fd	ef	88
		T6	MAR ← SP	L_MEM, E_SP,	10	18	7f	df	8

		T7	RAM[MAR] ← MDR	E_MDR_RAM,	10	18	ff	d7	88
		T8		W, E_MDR_RAM,	10	18	ff	d5	88
		T9		E_MDR_RAM,	10	18	ff	d7	88
		T10		R, RESET,	10	18	ff	de	89
ACC ← RAM[SP]	7	T5	MAR ← SP	L_MEM, E_SP,	10	18	7f	df	8
		T6	MDR ← RAM[MAR]	L_MDR_RAM, R,	10	18	ff	ce	88
		T7	ACC ← MDR, SP ← SP + 1	L_ACC, AS0, AS1, E_MDR_BUS, INC_SP, RESET	70	1a	fe	db	c9
ACC ← ACC << 1	6	T5	ACC ← ACC << 1	E_ACC	10	18	fd	df	c8
		T6		L_ACC, AS0, RESET	50	18	fc	df	c9
ACC ← ACC >> 1	6	T5	ACC ← ACC >> 1	E_ACC	10	18	fd	df	c8
		T6		L_ACC, AS1, RESET	30	18	fc	df	c9
Jump if carry	8	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	PC ← MDR	LP, E_MDR_BUS	10	18	ff	db	Cc
		T8		CP, LP, RESET	10	18	ff	df	Dd
Jump if equal	8	T5	MAR ← PC	L_MEM, E_PC,	10	18	ff	df	0
		T6	PC← PC+1, MDR ← RAM[MAR]	CP, L_MDR_RAM, R,	10	18	ff	ce	98
		T7	PC ← MDR	LP, E_MDR_BUS	10	18	ff	db	Cc
		T8		CP, LP, RESET	10	18	ff	df	Dd

ICs used with count:

IC number	IC name	Number
74LS173	4 bit D-type register with 3 state output	14
74LS244	Octal 3 state buffer	18
74LS157	Quad 2-input Multiplexer	5
4539	Dual 4-input Multiplexer	2
74LS181	4 bit Arithmetic Logic Unit	1
2732	EPROM 32K (K * 8 bit)	8
6116	CMOS Static RAM 16K (2K * 8 bit)	1
COUNTER_4	4 bit binary up/down counter	1
COUNTER_8	8 bit Binary up/down counter	4
74LS04	Hex Inverter	4
74LS32	Quad 2-input OR gate	1
74LS08	Quad 2-input AND gate	2
7411	Tri 3-input AND gate	1
AND_4	4-input AND gate	3
4075	Tri 3-input OR gate	1

In total, 66 ICs are used. Many of the 74LS244 (buffer) ICs are used for debugging purpose. So, in the actual circuit the number of this IC would be much less.

How to write and Execute a Program in this computer

Writing:

Writing a program for this computer can be done in two steps,
i) Writing in mnemonics form of instructions.
ii) Converting the mnemonic form to hex code. This is done by merging the hex opcode of the instruction and the hex value of the operand(address or immediate value, if exists)
Each instruction will become a hex value representing 1 or 2 bytes of information, given whether they use address or immediate value operands or not.
Next, we arrange the hex values in a BIN file line by line such that each line has a two digit hex code(1 byte).
Each program must be terminated with the HALT instruction which has the hex opcode F.
To denote the end of the program file, we use a special value of FF, we put this value at the last line of the program BIN file.

A sample program:

```
LDA 03
MOV B,ACC
05
XOR B
OUT
```

Here LDA has opcode 00H, MOV B,ACC has opcode 04H, XOR B has opcode 13H, OUT has a opcode 03H.
So in Hex the program file will be:

```
00
02
05
13
03
```

We store these hex codes in a BIN file. We add, FF at the final line to denote end of FILE.

Execution:

We load this BIN file in the program ROM. When the PC starts, during the boot loader phase, each of these instructions from the program ROM is loaded into the RAM, afterwards during the fetch cycle, OP is fetched from the RAM and it is eventually sent to the instruction register and the execution phase starts

Discussion:

At first, starting the design was very difficult since we had minimum knowledge about it. Gradually by the passage of time, we could learn from mistakes and the process of designing became easier.

Designing the registers, program counter and stack pointer was easier. But, controller-sequencer and memory were quite difficult in this regards.

We faced problems while designing micro-operations for each instruction. Sometimes, we thought that particular operation could be done in single cycle, but performing them in one cycle resulted in failure. Such as, loading PC from TEMP or MDR took 2 cycles, but initially, we thought this could be done in one cycle. Writing data to an address of RAM requires 3 cycles normally. But when it is the last operation of an instruction, it takes 4 cycles since 3rd cycle of writing to RAM and RESET operation could not be done in a single cycle.

It was very difficult to understand the function of Boot_Loader. After studying more about the process of loading user program into the RAM, it became clear and we could implement it.

We faced problem in stopping the clock when a HLT instruction is executed. Because, it was interfering with the boot loading process what we did initially. If we corrected the boot loading, then HLT was not stopping the clock. By some trial and error, we could come up with a solution.

Initially, the stack pointer was not getting set to FF rather it was getting decremented twice and set to FE. This problem was also fixed later on.

For easier debugging, we used lots of buffers in the circuit to see the current value of different components which were not actually needed in the PC circuit.

We also used 7-Segment HEX display for observing the value of control word and different registers.

We wrote a java code for converting the hex code for the address ROM, control ROM's into binary file. We provided the active pins for each cycle of an instruction and our program could generate the corresponding control word from it and converted them into a binary file that could be easily used into the simulation circuit.

We also wrote a java program. It could take input a set of instructions (user program) and generate corresponding hex code of that input user program and convert them into a binary file which was loaded into the RAM during boot loading.

At the end, designing a complete PC from scratch had a great influence to all of us. It made us realize how difficult it is to design a multi-purpose PC and helped to clear our concept about so many internal issues of PC designing.