CS322 Term Project Assignment

Name: Shubhajeet Dey

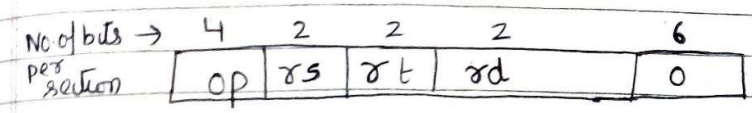
RollNo: 1901CS58

PAGE:

★ Term project Assignment ★

★ Instruction Available :→ (16 bits = 2 bytes) (Each Register can store 8bit info.)

→ R Type Instructions (ADD, SUB, AND, XOR) :→



op :→ operation code (4bits)

rs :→ Address of first register

rt :→ Address of second register

rd :→ Address of destination register.

★ 4-registers allowed R0..R3 so, 2bits

★ 4-bits for op-code.

★ op :→ ↳ after operation on contents of rs and rt

for ADD :→ 0000

for SUB :→ 0011

for AND :→ 0001

for XOR :→ 0010

Say $rs, $rt, $rd represents content of respective registers.

Process :→ $rd = $rs (op) $rt

∴ 4 Cycle Taken

∴ op = ADD, SUB, AND XOR

★ Load instruction :→

No of bits per section →

4	2	2	8
op	rs	rt	8 bit offset

op :→ operation code (4 bits)

rs :→ source register address

rt :→ destination register address

Offset :→ 8 bit offset [As the Data Memory has 8 bit addressing i.e it is capable of addressing 2^8 bytes in the memory]

* Here $rs and $rt represent content of respective registers.

Process :→ $rt = M[$rs + offset]

[Memory = $2^8 \times 8$]

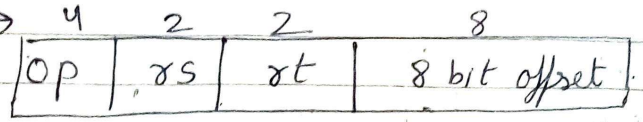
↳ M[x] returns value in Memory whose address is 'x'.

* OP code for this instruction is 010X

(X is don't care)

∴ 5 Cycle Taken

★ Store instruction :→



op → operation code (4 bits)

rs → source register address

rt :→ The register address, whose contents have to be stored.

offset → 8 bit offset [As the Data Memory has 8 bit addressing i.e it is capable of addressing 2^8 bytes in the memory]

★ Here $rs and $rt represent content of respective registers.

Process :→ M[$rs + offset] = $rt [Memory = 2^8 x 8]

↳ M[x'] represents content in Memory whose address is 'x'

★ OP code for this instruction is 011X

(X is don't care)

∴ 4 cycle Taken

★ Branch Type instructions:→ (BEQ, BLT, BGT)

No. of bits → per section	4	2	2	8
	op	rs	rt	8 bit offset

op:→ operation code (4 bits)

rs:→ Address of first register

rt:→ Address of second register

offset:→ 8 bit offset [~~B~~ instruction memory has 8 bit addressing i.e ~~B B B~~ ~~capacity address~~ Memory = $2^8 \times 16$]
16 bit instruction

★ Here $rs and $rt represent content of the respective registers:→

★ Op:→

for BEQ:→ 100x
for BLT:→ 1010
for BGT:→ 1011

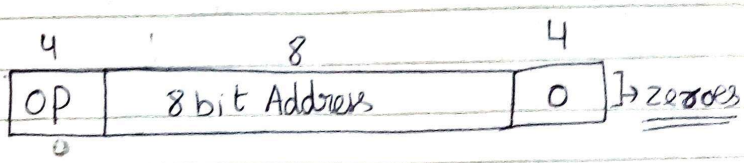
∴ 3 cycle Taken

if ($rs op $rt)
PC = PC + offset
↳ op = eq, Lt, Gt
↳ Here offset is already scaled to word size in ~~B B B~~ Logisim.

[Process]

★ Jump instruction :→

No. of bits → per section



op :→ operation code (4 bits)

Address :→ 8 bit Address to be jumped at

[As instruction memory has 8 bit addressing]

Process :→

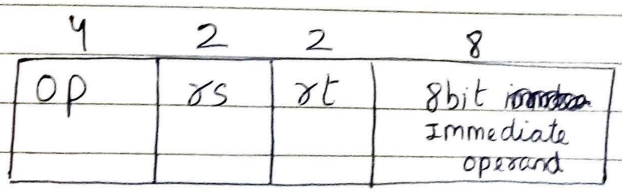
∴ PC = 8 bit Address

* OP code for this instruction is 1111 X (X is don't care)

∴ 3 Cycle Taken

★ ADDI instructions (Immediate) →

No. of bits → per section



op :→ operation code (4 bits)

rs :→ Address of source register

rt :→ Address of destination register where computed value have to be stored.

Immediate :→ 8 bit Number to be computed.
Operand

Process :→ Here $rs and $rt represent content of respective registers.

↓

$rt = $rs (op) imm

imm :→ immediate operand

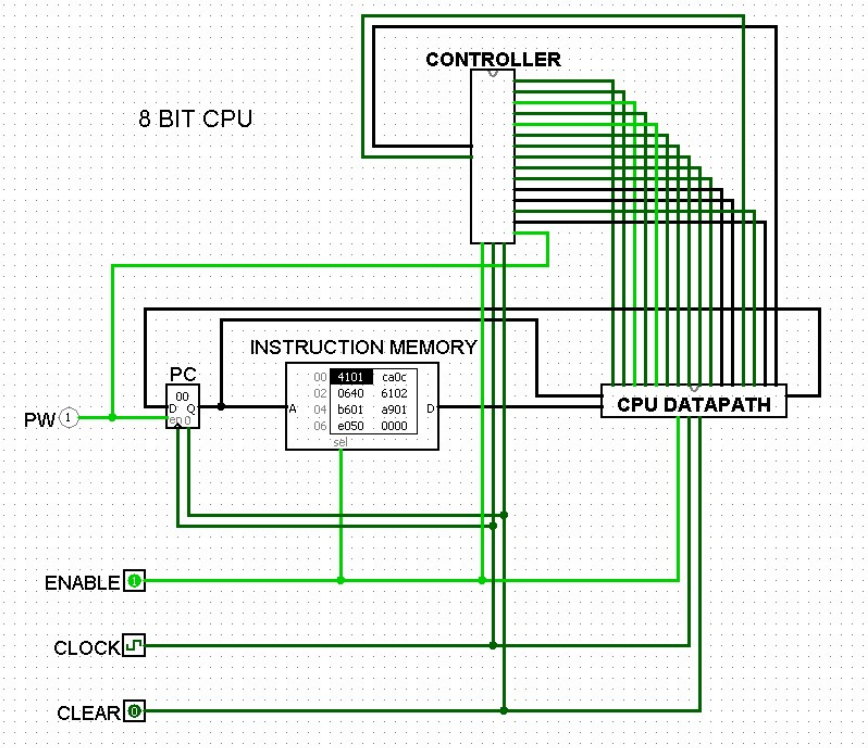
op :→ add,

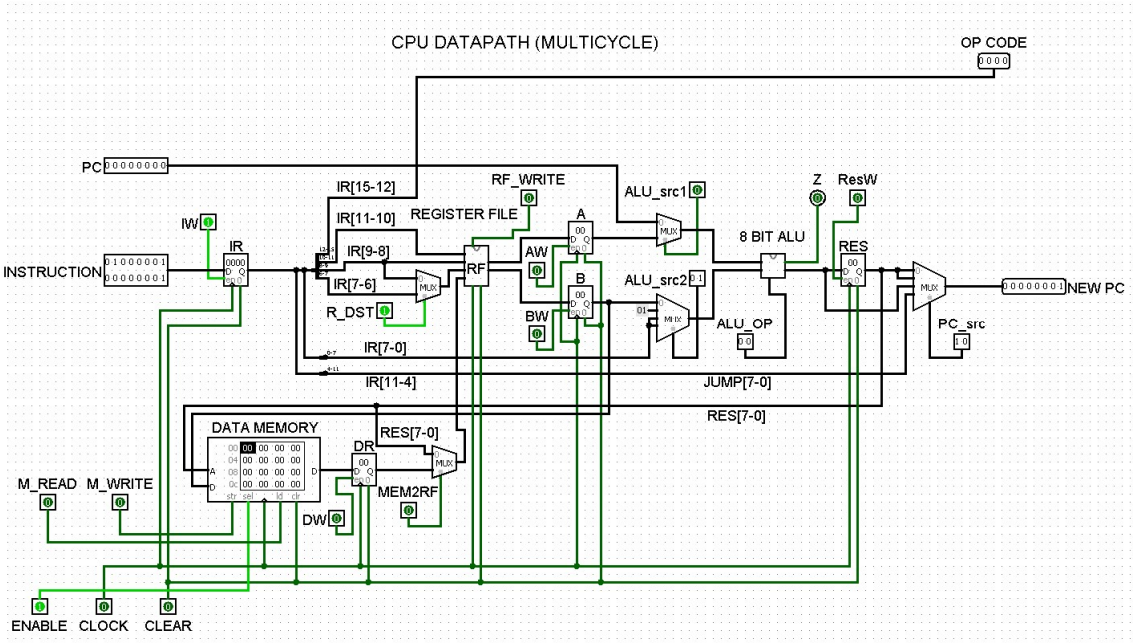
* Op :→

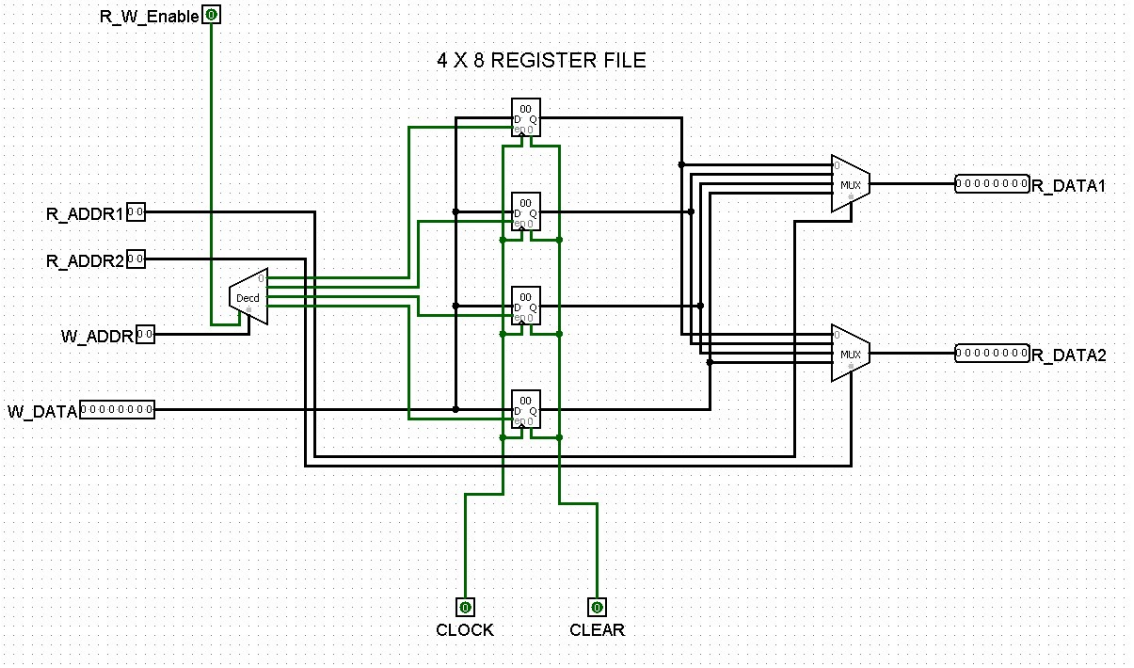
For ADDI :→ 110X (X is don't care)

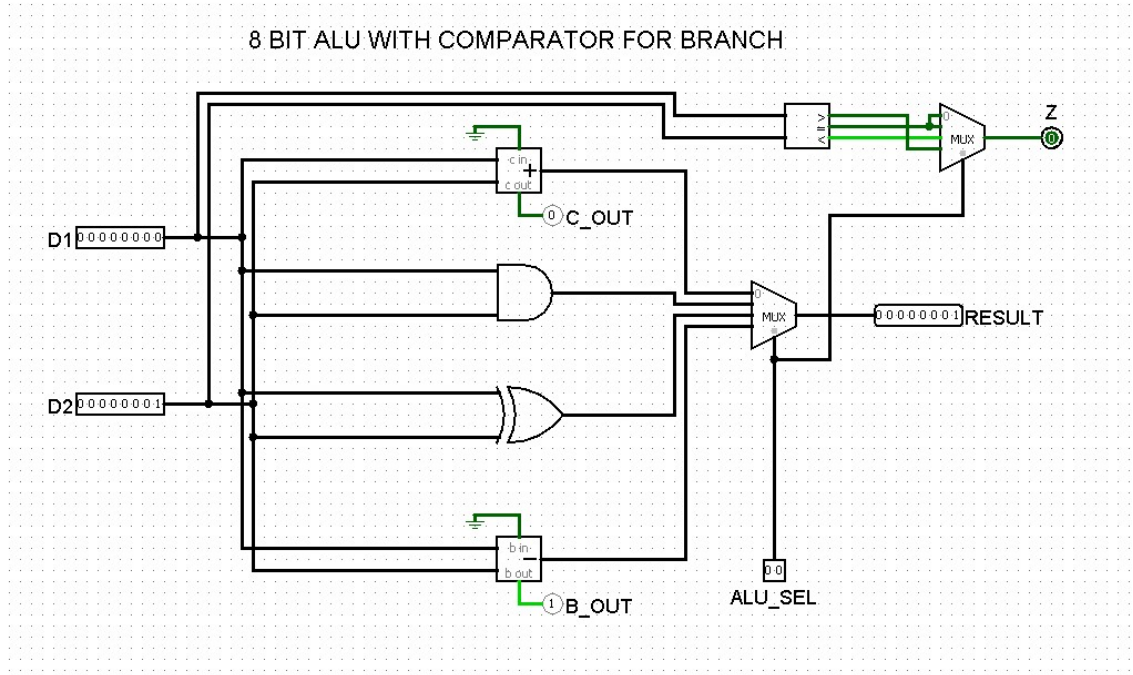
∴ 4 cycle Taken

CPU Datapath(Multicycle):

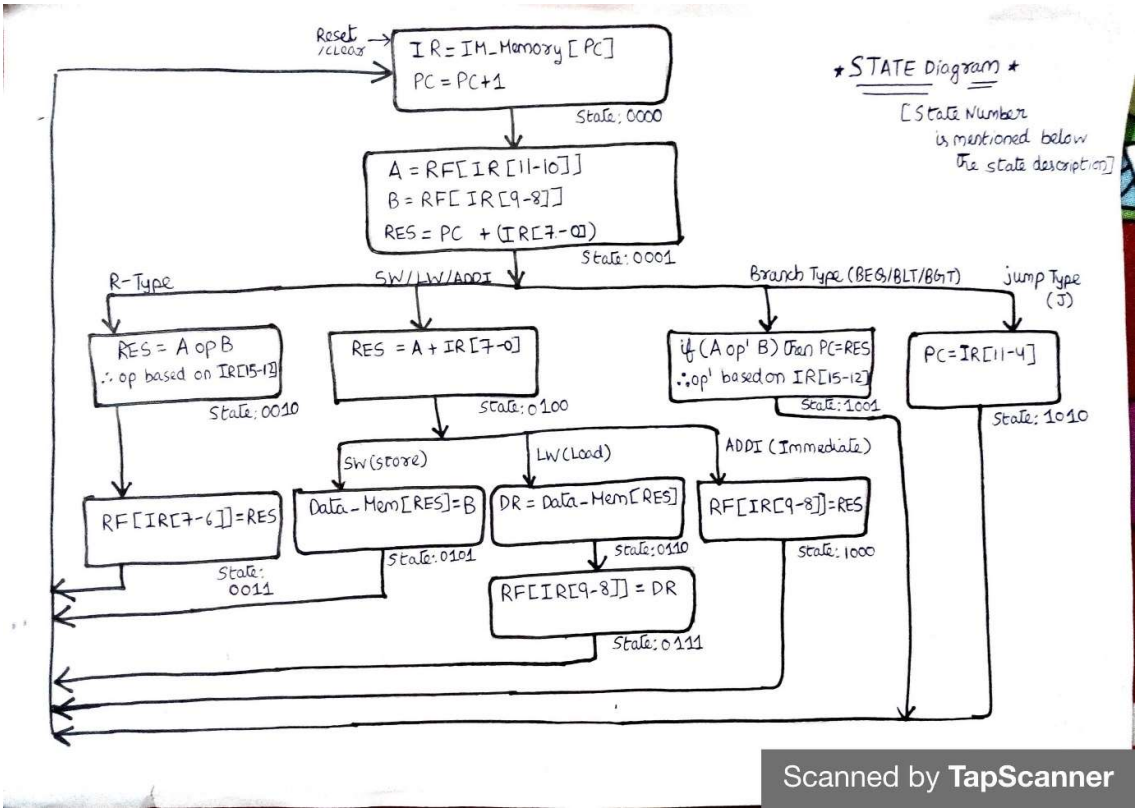



CPU DATAPATH (MULTICYCLE)
OP CODE
PC
IW
IR
INSTRUCTION
IR[15-12]
IR[11-10]
IR[9-8]
IR[7-6]
R_DST
IR[7-0]
IR[11-4]
RF_WRITE
REGISTER FILE
RF
AW
BW
A
B
ALU_src1
ALU_src2
8 BIT ALU
ALU_OP
Z
ResW
RES
PC_src
NEW PC
JUMP[7-0]
RES[7-0]
DATA MEMORY
DR
MEM2RF
M_READ
M_WRITE
DW
ENABLE
CLOCK
CLEAR


R_W_Enable
4 X 8 REGISTER FILE
R_ADDR1
R_ADDR2
W_ADDR
W_DATA
Decd
R_DATA1
R_DATA2
CLOCK
CLEAR



State Diagram with State Description:



In total of 4 bits are used to represent state of the System.

Here PC=PC+1, as logisim considers a word (16 bits) as a single block with a unique 8 bit address, because of which PC=PC+1 represents next instruction address.

Here, also Instruction Memory and Data Memory are taken different as the first one is 2^8 X 16 Memory with 8 bit addressing and 16 bit memory word and second one is 2^8X8 Memory with 8 bit addressing and 8bit memory word (8 bit value are being operated on). Also, it provides more security (as user can overwrite instructions with data unknowingly if both are merged)

Instruction Memory is read only and Data Memory is both read and write for this project.

(same reason) Here (branch) offset doesn't need to be scaled as it is already taken care by Logisim.

* State :→ S3 S2 S1 S0
* Op Code :→ op3 op2 op1 op0
* Next State :→ N3 N2 N1 N0

∴ op,op :→ Are controlled by ALU_Control signal, if 1 then op is derived directly from IR[15-12] or if 0 then '+' signal is send.
↳ (00)

* IR[13-12] can say about the ALU op.

PAGE:

State Table:→ X is don't care, remaining combinations are don't care.

S3	S2	S1	S0	Op3	Op2	Op1	Op0	N3	N2	N1	N0
0	0	0	0	X	X	X	X	0	0	0	1
0	0	0	1	0	0	X	X	0	0	1	0
0	0	0	1	0	1	X	X	0	1	0	0
0	0	0	1	1	1	0	X	0	1	0	0
0	0	0	1	1	0	X	X	1	0	0	1
0	0	0	1	1	1	1	X	1	0	1	0
0	0	1	0	X	X	X	X	0	0	1	1
0	0	1	1	X	X	X	X	0	0	0	0
0	1	0	0	0	X	1	X	0	1	0	1
0	1	0	1	X	X	X	X	0	0	0	0
0	1	0	0	0	X	0	X	0	1	1	0
0	1	1	0	X	X	X	X	0	1	1	1
0	1	1	1	X	X	X	X	0	0	0	0
0	1	0	0	1	X	X	X	1	0	0	0
1	0	0	0	X	X	X	X	0	0	0	0
1	0	0	1	X	X	X	X	0	0	0	0
1	0	1	0	X	X	X	X	0	0	0	0

The state and input combinations have already been optimized for minimization.

S_0, N_3:→ $\bar{S3}\bar{S2}\bar{S1}S0\ Op3\ \overline{Op2} + \bar{S3}\bar{S2}\bar{S1}S0\ Op3\ Op2\ Op1 + \bar{S3}S2\bar{S1}\bar{S0}\ Op3$

$\bar{S3}\bar{S2}\bar{S1}S0\ Op3(\overline{Op2} + Op2\ Op1) + \bar{S3}S2\bar{S1}\bar{S0}Op3$

$N_3 \Rightarrow \bar{S3}\bar{S2}\bar{S1}S0\ Op3\ \overline{Op2} + \bar{S3}\bar{S2}\bar{S1}S0\ Op3\ Op1 + \bar{S3}S2\bar{S1}\bar{S0}Op3$ | $x + \bar{x}y$

* x + x'y = x + y

$S_0, N_2 :\rightarrow$

$\bar{S_3}\bar{S_2}\bar{S_1}S_0\overline{OP_3}OP_2 + \bar{S_3}\bar{S_2}\bar{S_1}S_0\overline{OP_3}OP_2\overline{OP_1}$

$+ \bar{S_3}S_2\bar{S_1}\bar{S_0}\overline{OP_3}OP_1 + \bar{S_3}S_2\bar{S_1}\bar{S_0}\overline{OP_3}OP_1$

$+ \bar{S_3}S_2S_1\bar{S_0}$

$[X + \bar{X} = 1$

$X + \bar{X}Y = X + Y$

$AB + A\bar{B} = A]$

$N_2 \Rightarrow \bar{S_3}\bar{S_2}\bar{S_1}S_0\overline{OP_3}OP_2 +$

$\bar{S_3}\bar{S_2}\bar{S_1}S_0OP_2\overline{OP_1} + \bar{S_3}S_2\bar{S_0}\overline{OP_3} + \bar{S_3}S_2S_1\bar{S_0}$

$S_0, N_1 :\rightarrow$

$(\bar{S_3}\bar{S_2}\bar{S_1}\overline{OP_3}\overline{OP_2}\overline{OP_1}S_0\overline{OP_3}\overline{OP_2}) + \bar{S_3}\bar{S_2}\bar{S_1}S_0OP_3OP_2OP_1$

$+ \bar{S_3}\bar{S_2}S_1\bar{S_0} + \bar{S_3}S_2\bar{S_1}\bar{S_0}\overline{OP_3}\overline{OP_1}$

$+ \bar{S_3}S_2S_1\bar{S_0}$

$[AB + A\bar{B} = A$

$X + \bar{X}Y = X + Y]$

$N_1 \Rightarrow \bar{S_3}\bar{S_2}\bar{S_1}S_0\overline{OP_3}\overline{OP_2} + \bar{S_3}\bar{S_2}\bar{S_1}S_0OP_3OP_2OP_1$

$+ \bar{S_3}S_1\bar{S_0} + \bar{S_3}S_2\bar{S_0}\overline{OP_3}\overline{OP_1}$

S_0, N_0 :→

$\bar{S3}\,\bar{S2}\,\bar{S1}\,\bar{S0} + \bar{S3}\,\bar{S2}\,\bar{S1}\,S0\,Op3\,\overline{Op2} + \bar{S3}\,\bar{S2}\,S1\,\bar{S0}$

$+ \bar{S3}\,\bar{S2}\,\bar{S1}\,\bar{S0}\,\overline{Op3}\,Op1 + \bar{S3}\,\bar{S2}\,S1\,\bar{S0}$

[$X+\bar{X}Y = X+Y$
$A\bar{B}+AB = A$]

$N_0 \Rightarrow$

$\bar{S3}\,\bar{S2}\,\bar{S0} + \bar{S3}\,\bar{S2}\,\bar{S1}\,Op3\,\overline{Op2} +$

$\bar{S3}\,\bar{S0}\,\overline{Op3}\,Op1 + \bar{S3}\,S1\,\bar{S0}$

Output Logic :→ [Moore Machine] :→

State	M_READ	M_WRITE	IW	DW	R_DST	MEM2RF
0000	0	0	1	0	X	X
0001	0	0	0	0	X	X
0010	0	0	0	0	X	X
0011	0	0	0	0	1	0
0100	0	0	0	0	X	X
0101	0	1	0	0	X	X
0110	1	0	0	1	X	X
0111	0	0	0	0	0	1
1000	0	0	0	0	0	0
1001	0	0	0	0	X	X
1010	0	0	0	0	X	X

State	RF_WRITE	AW	BW	ALU_src1	ALU_src2
0000	0	0	0	0	1
0001	0	1	1	0	2
0010	0	0	0	1	0
0011	1	0	0	x	x
0100	0	0	0	1	2
0101	0	0	0	x	x
0110	0	0	0	x	x
0111	1	0	0	x	x
1000	1	0	0	x	x
1001	0	0	0	1	0
1010	0	0	0	x	x

State	ALU_control	ResW	PC_src	PW
0000	0	0	2	1
0001	0	1	x	0
0010	1	1	x	0
0011	x	0	x	0
0100	0	1	x	0
0101	x	0	x	0
0110	x	0	x	0
0111	x	0	x	0
1000	x	0	x	0
1001	1	0	0	1
1010	x	0	3	1

PAGE:

★ M-READ:→

S3S2 \ S1S0	00	01	11	10
00	0 (0)	0 (1)	0 (3)	0 (2)
01	0 (4)	0 (5)	0 (7)	1 (6)
11	X (12)	X (13)	X (15)	X (14)
10	0 (8)	0 (9)	X (11)	0 (10)

M-READ = $S_2 S_1 \bar{S_0}$

★ M-WRITE:→

S3S2 \ S1S0	00	01	11	10
00	0 (0)	0 (1)	0 (3)	0 (2)
01	0 (4)	1 (5)	0 (7)	0 (6)
11	X (12)	X (13)	X (15)	X (14)
10	0 (8)	0 (9)	X (11)	0 (10)

M-WRITE = $S_2 \bar{S_1} S_0$

★ IW:→

S3S2 \ S1S0	00	01	11	10
00	1 (0)	0 (1)	0 (3)	0 (2)
01	0 (4)	0 (5)	0 (7)	0 (6)
11	X (12)	X (13)	X (15)	X (14)
10	0 (8)	0 (9)	X (11)	0 (10)

IW = $\bar{S_3} \bar{S_2} \bar{S_1} \bar{S_0}$

★ DW :→

S3S2 \ S1S0	00	01	11	10
00	O_0	O_1	O_3	O_2
01	O_4	O_5	O_7	1 (6)
11	X_{12}	X_{13}	X_{15}	X_{14}
10	O_8	O_9	X	O_{10}

$DW = S2\,S1\,\overline{S0}$

★ R_DST :→

S3S2 \ S1S0	00	01	11	10
00	X_0	X_1	1 (3)	X_2
01	X_4	X_5	O_7	X_6
11	X_{12}	X_{13}	X_{15}	X_{14}
10	O_8	X	X_{11}	X_{10}

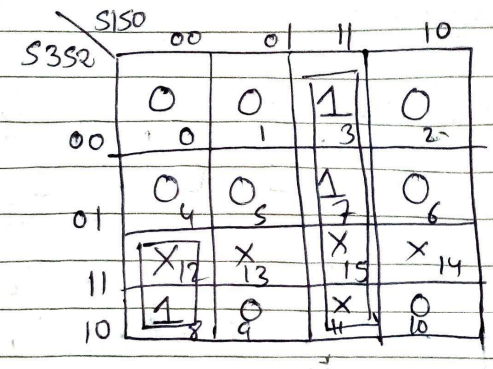
$R\_DST = \overline{S3}\,S2$

★ MEM2RF :→

S3S2 \ S1S0	00	01	11	10
00	X_0	X_1	O_3	X_2
01	X_4	X_5	1 (7)	X_6
11	X_{12}	X_{13}	X_{15}	X_{14}
10	O_8	X_9	X	X_{10}

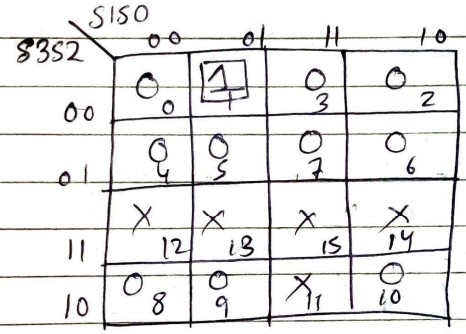
$MEM2RF = S2$

★ RF_WRITE :→



$RF\_WRITE = S1S0 + S3\bar{S1}\bar{S0}$

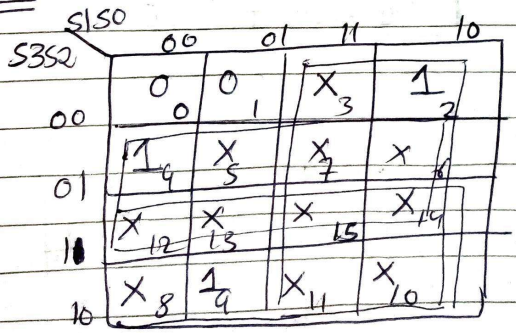
★ AW :→



$AW = \bar{S3}\bar{S2}\bar{S1}S0$

As $AW = BW = \bar{S3}\bar{S2}\bar{S1}S0$

★ ALU_Src1 :→



$ALU\_Src1 = S1 + S2 + S3$

PAGE:

★ ALU_Src2 :→

Bit0 :→

S3S2 \ S1S0	00	01	11	10
00	1 (0)	0 (1)	X (3)	0 (2)
01	0 (4)	X (5)	X (7)	X (6)
11	X (12)	X (13)	X (15)	X (14)
10	X (8)	0 (9)	X (11)	X (10)

Bit0 :→ $\bar{S2}\,\bar{S1}\,\bar{S0}$

Bit1 :→

S3S2 \ S1S0	00	01	11	10
00	0 (0)	1 (1)	X (3)	0 (2)
01	1 (4)	X (5)	X (7)	X (6)
11	X (12)	X (13)	X (15)	X (14)
10	X (8)	0 (9)	X (11)	X (10)

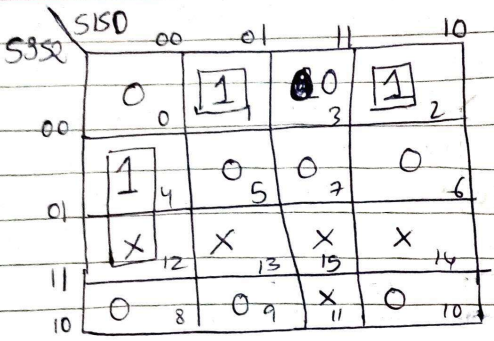
Bit1 :→ $S2 + \bar{S3}\,S0$

★ ALU_Control :→

S3S2 \ S1S0	00	01	11	10
00	0 (0)	0 (1)	X (3)	1 (2)
01	0 (4)	X (5)	X (7)	X (6)
11	X (12)	X (13)	X (15)	X (14)
10	X (8)	1 (9)	X (11)	X (10)

ALU_Control = S3 + S1

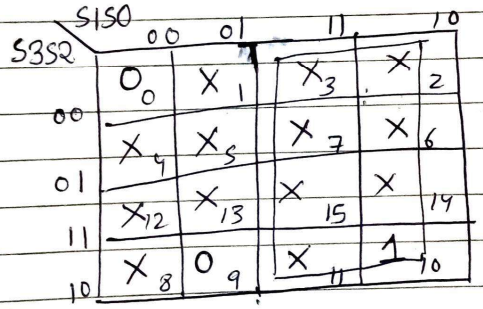
* ResW :→



ResW = $S_2\bar{S}_1\bar{S}_0 + \bar{S}_3S_2\bar{S}_1S_0 + \bar{S}_3\bar{S}_2S_1\bar{S}_0$

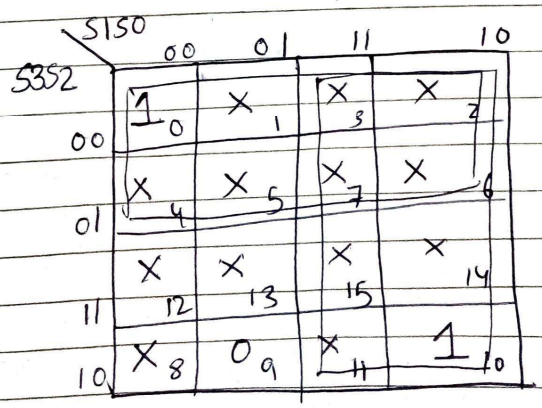
* PC_Src :→

Bit 0 :→

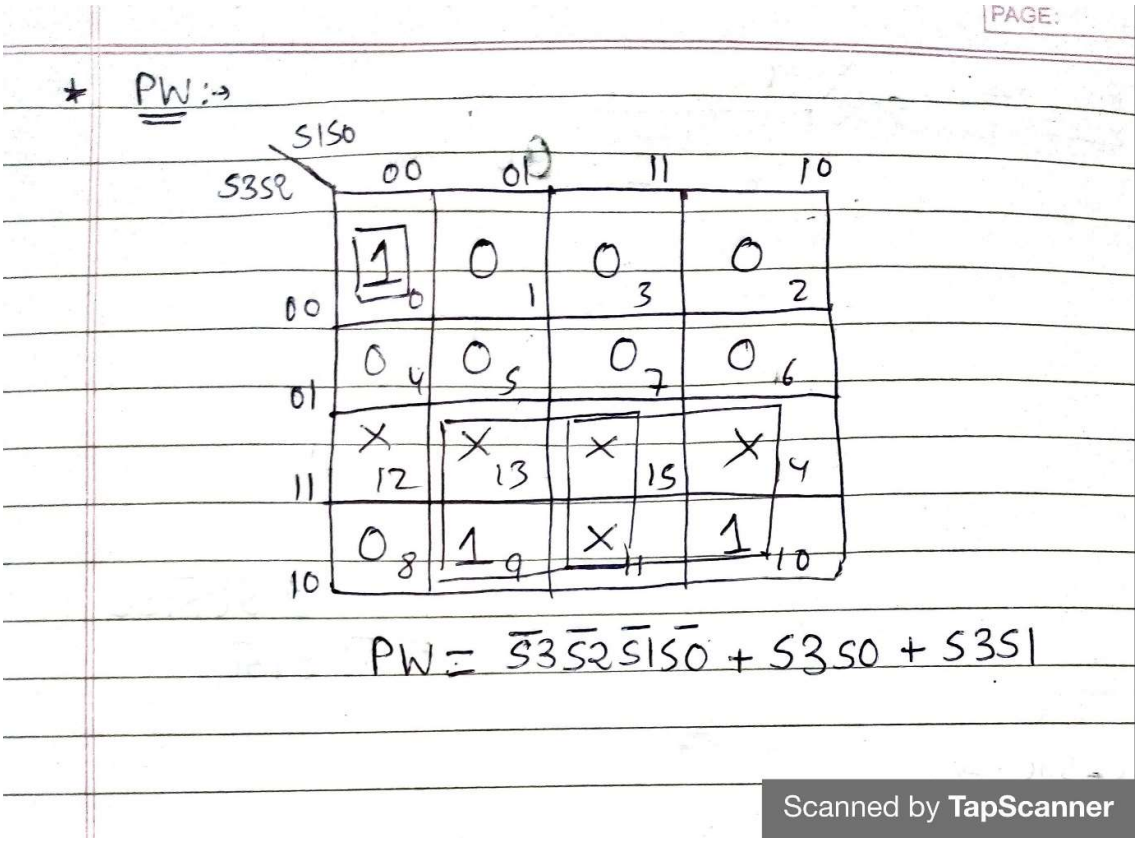


Bit0 :→ S_1

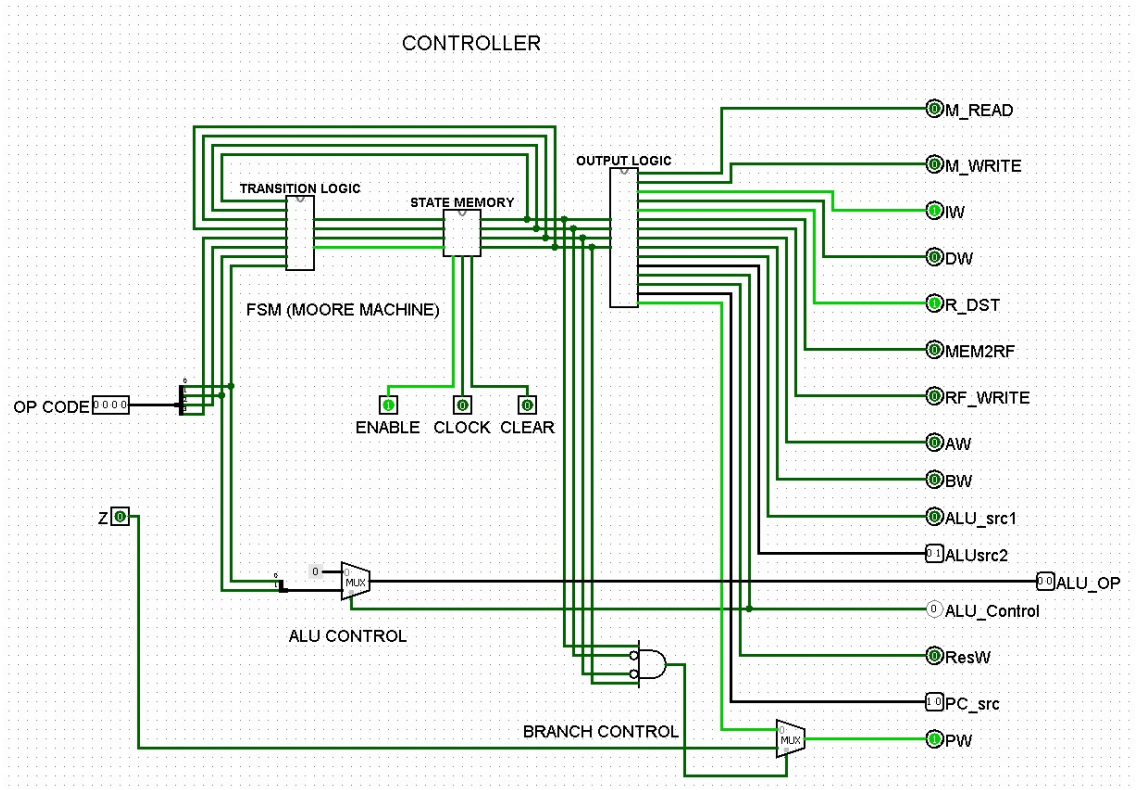
Bit 1 :→

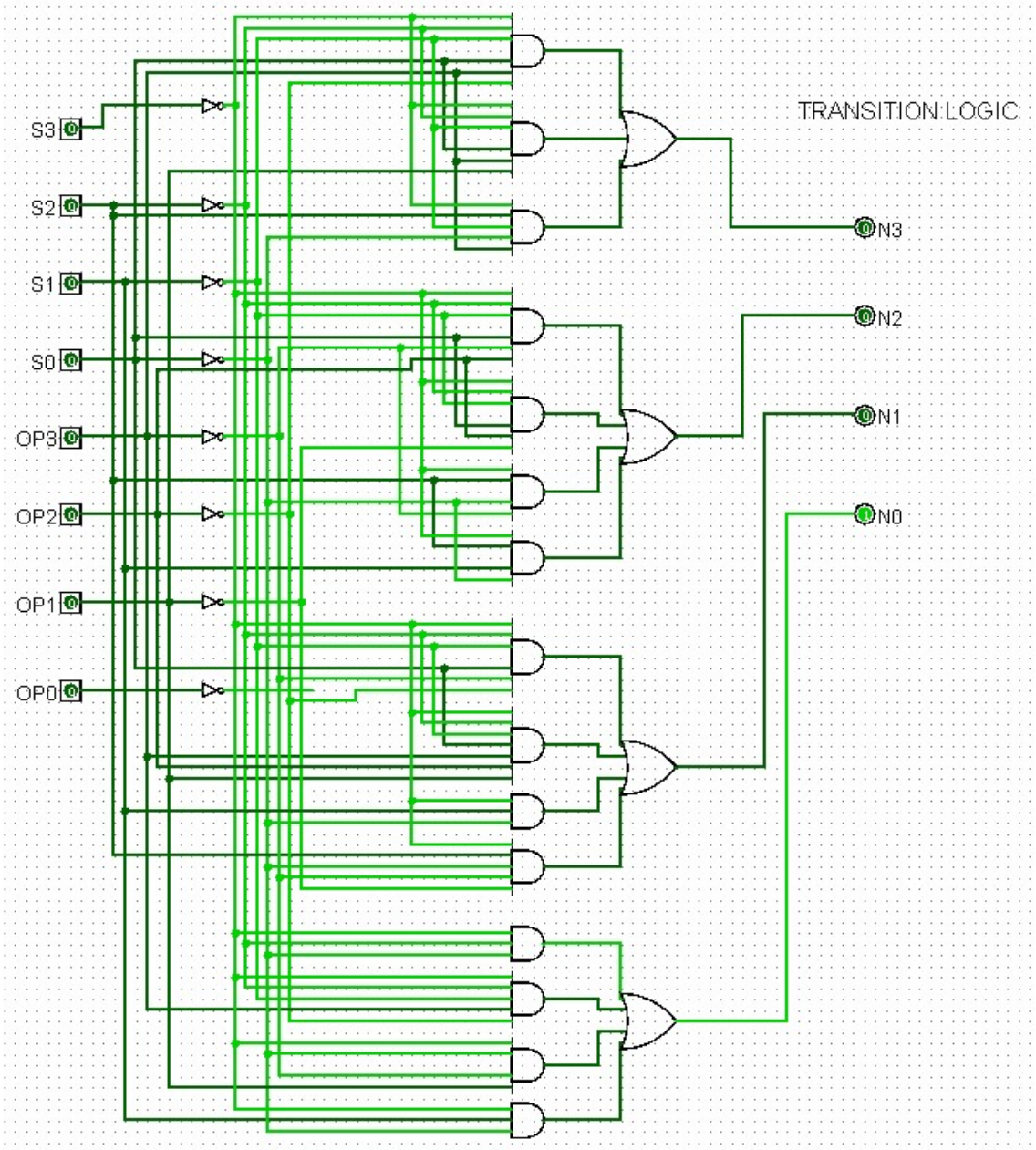


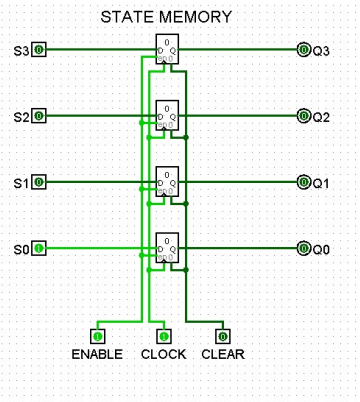
Bit 1 :→ $\bar{S}_3 + S_1$

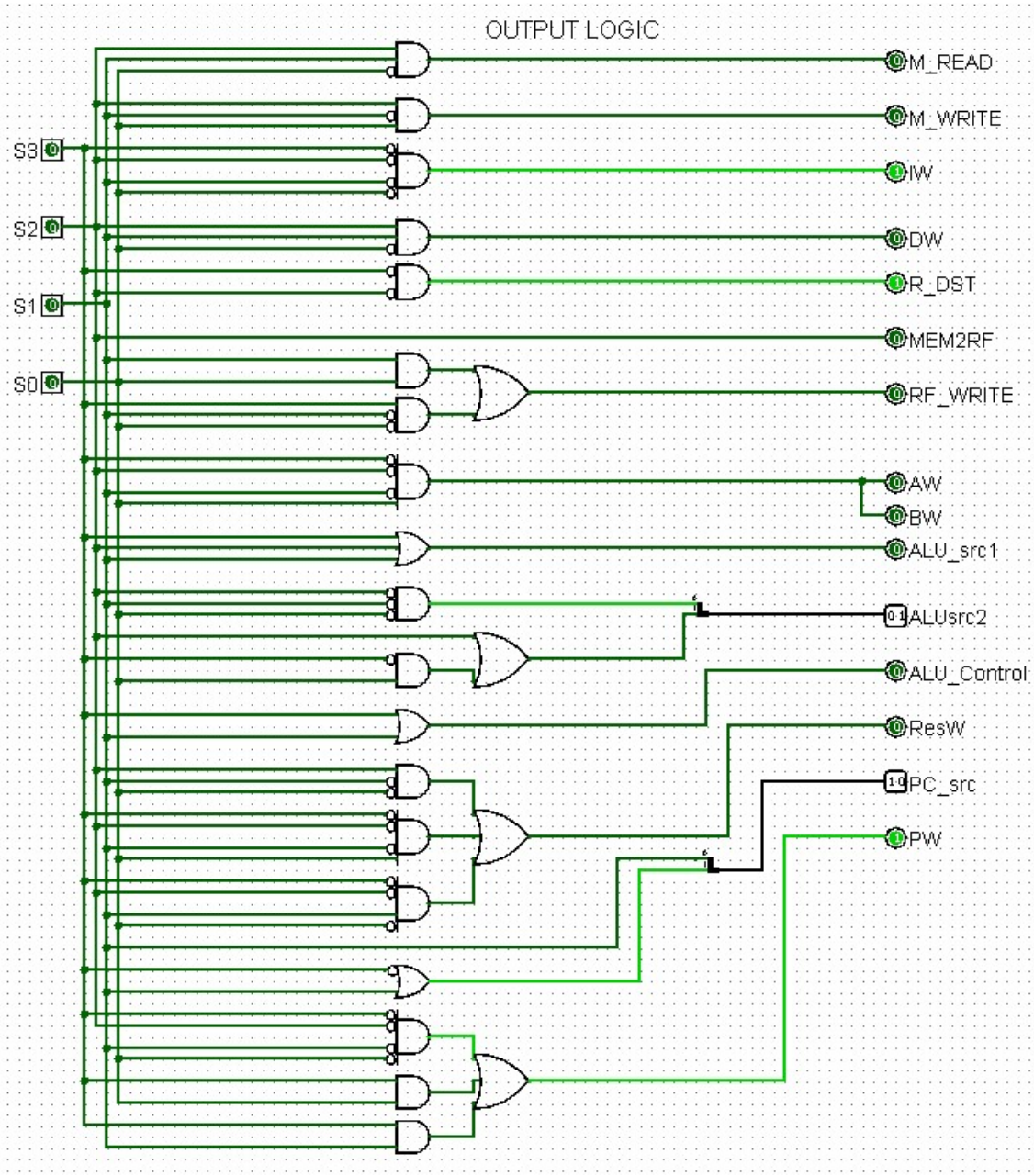


Controller Design:




TRANSITION LOGIC
S3
S2
S1
S0
OP3
OP2
OP1
OP0
N3
N2
N1
N0


STATE MEMORY
S3
S2
S1
S0
Q3
Q2
Q1
Q0
ENABLE
CLOCK
CLEAR



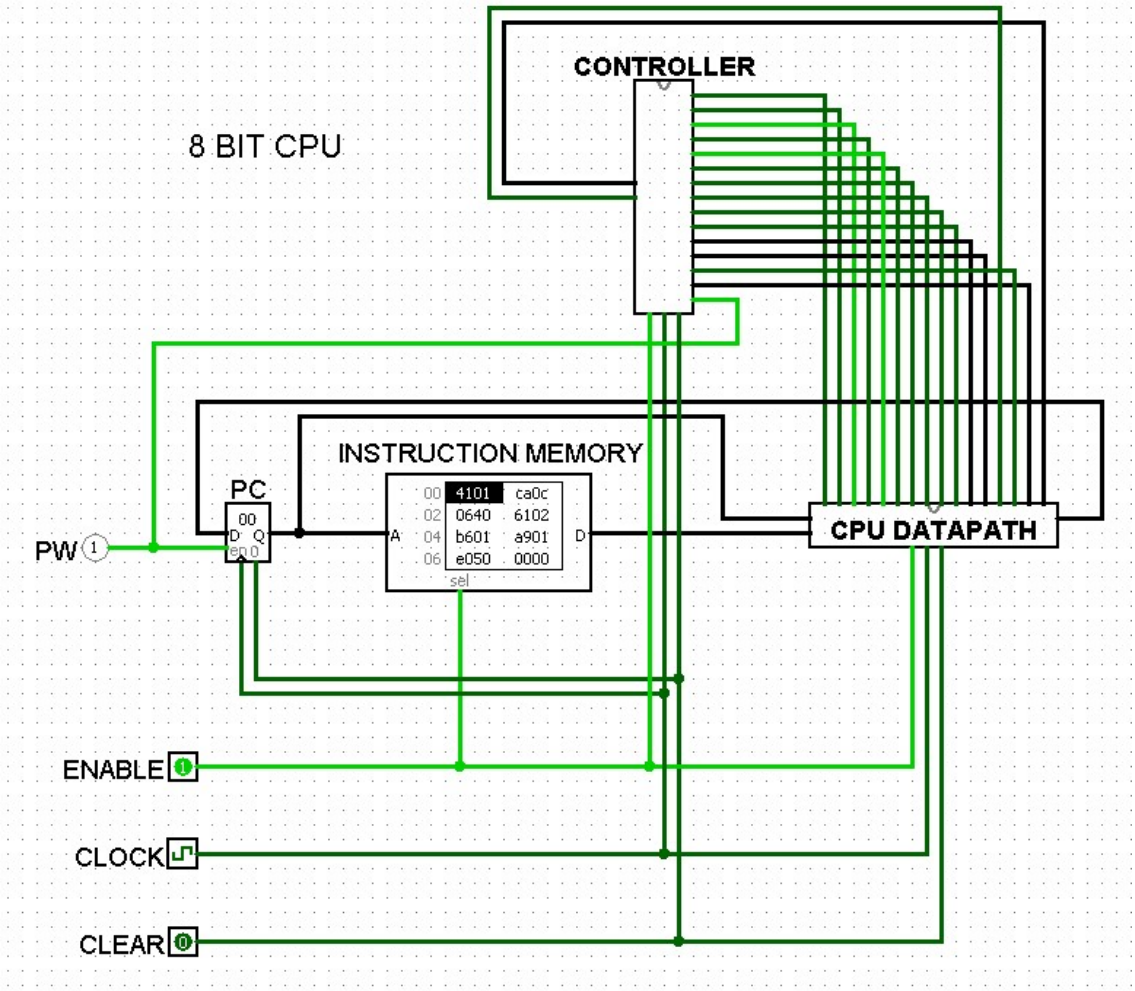
Output:

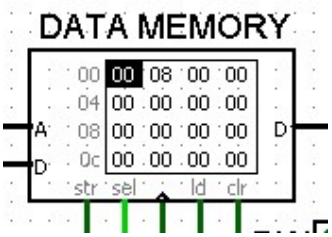
Data Memory position 0x01 is set to 0x08 for Load operation. All the registers are set to 0 by default. PC is set to 0

Now, the instructions inputted in Instruction Memory:

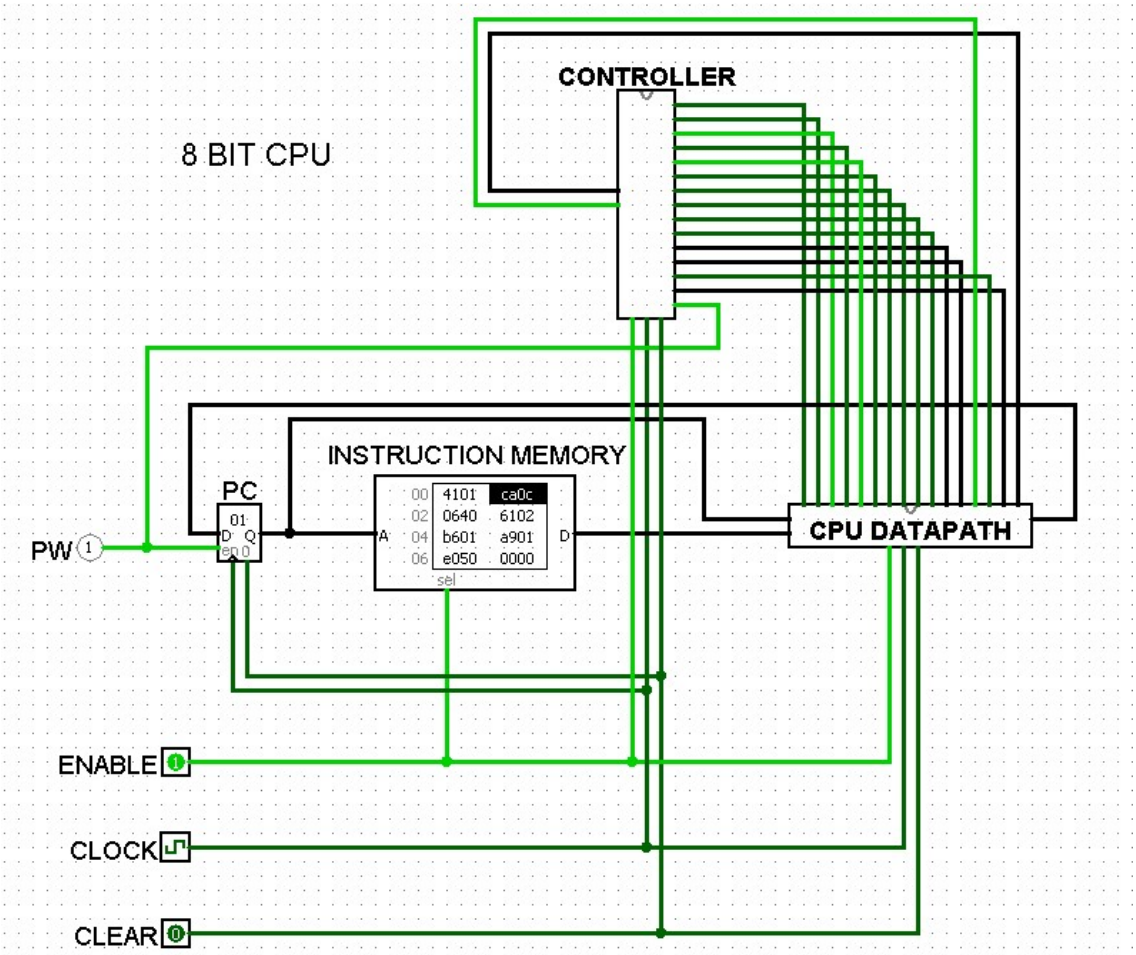
IM_Address	Instruction in Hex	Description	Remarks
0x00	0x4101	Load M[0x01+ ($r0)] = M[0x01] in $r1	Now, $r1=0x08
0x01	0xCA0C	ADD immediate $r2 and 12, store value in $r2	Now, $r2=0x0C
0x02	0x0640	ADD $r1 and $r2, store in $r1	Now, $r1=0x14
0x03	0x6102	Store $r1 in M[0x02+ ($r0)] = M[0x02]	Now, M[0x02]=0x14
0x04	0xB601	Branch if $r1 > $r2, offset = 1, meaning PC = PC* + 1, PC* = incremented PC	
0x05	0xA901	Branch if $r2 < $r1, offset = 1, meaning PC = PC* + 1, PC* = incremented PC	
0x06	0xE050	Jump to address 0x05	PC = 0x05

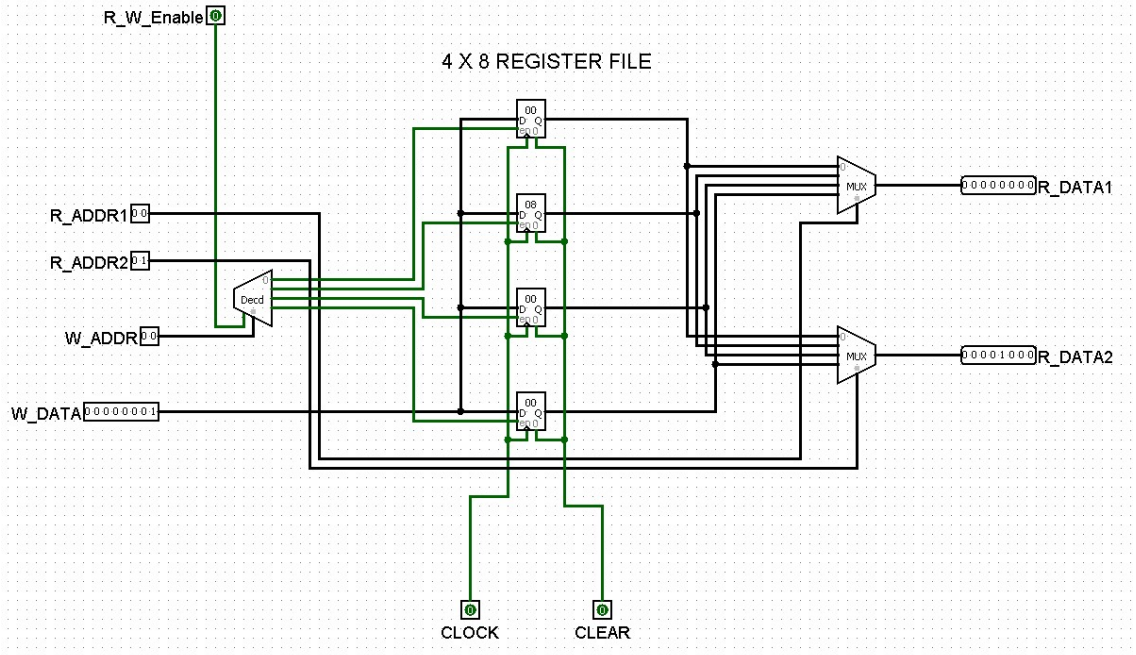
Initial State:





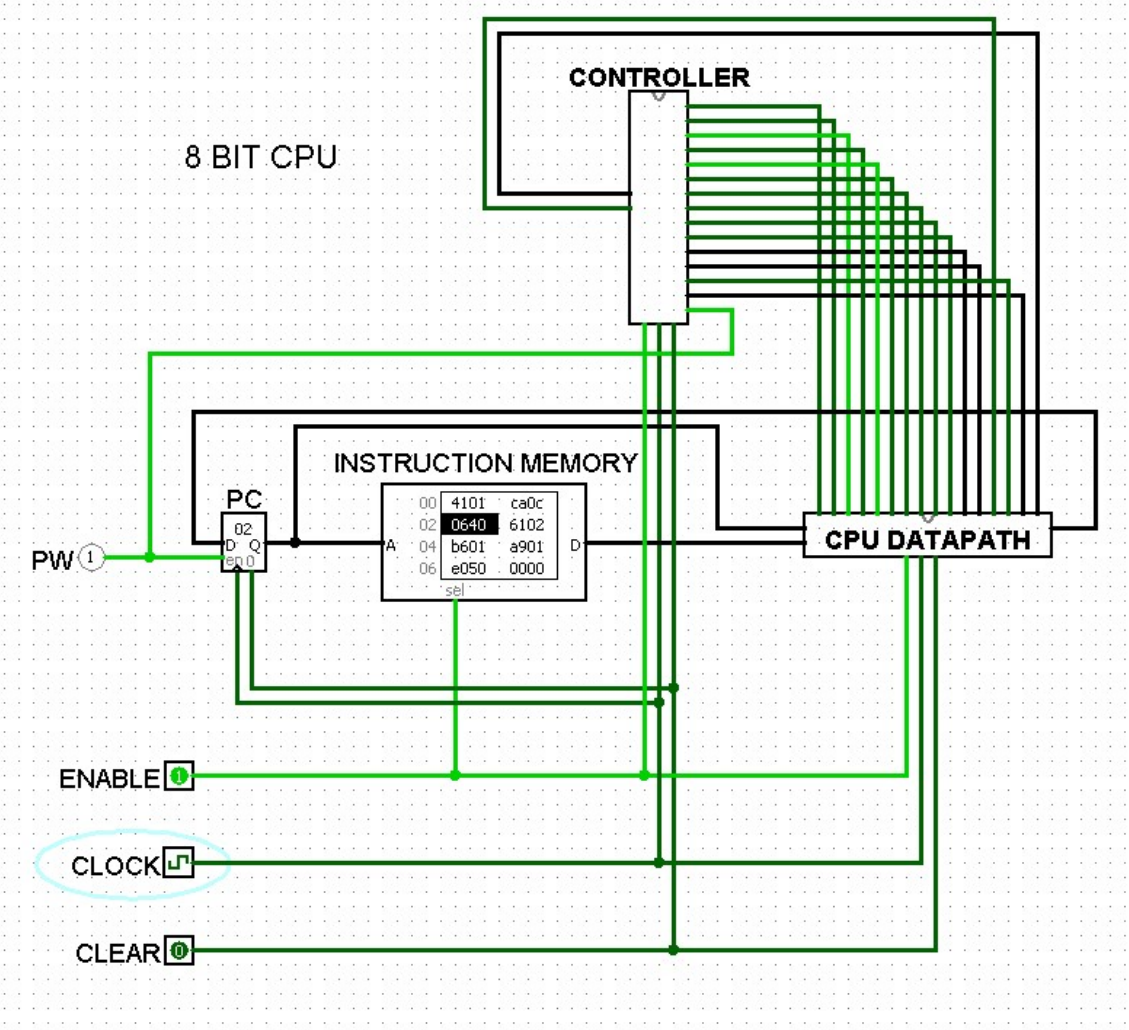
After 5 cycles for Load instruction:

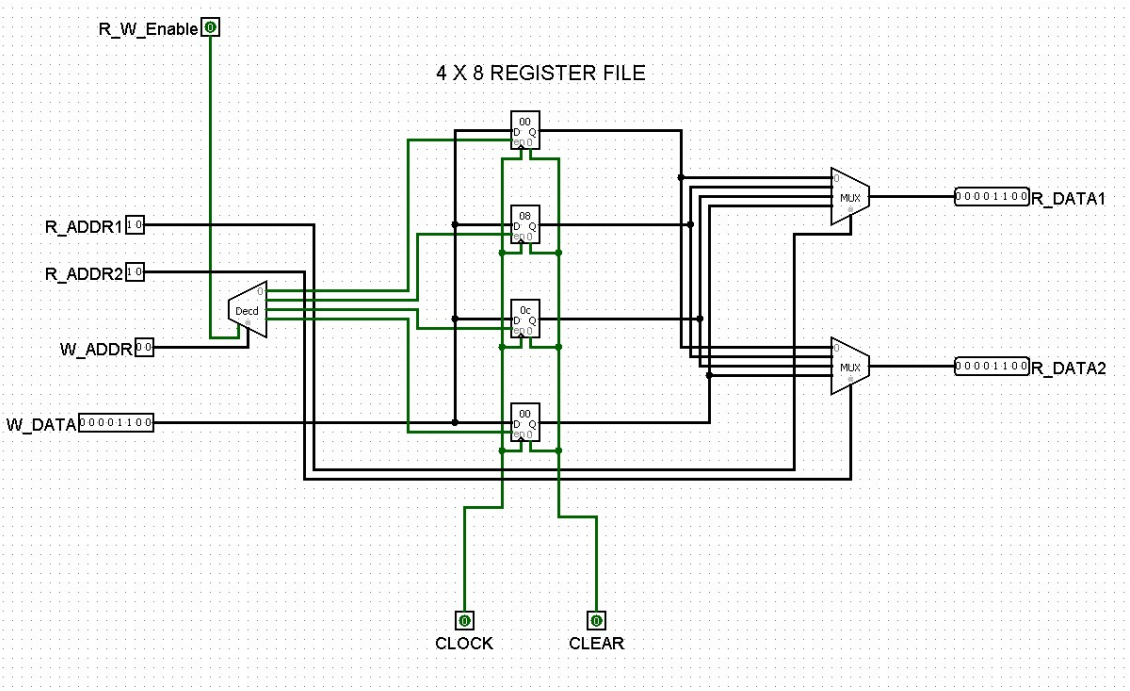

8 BIT CPU
CONTROLLER
INSTRUCTION MEMORY
PC
00 4101 ca0c
02 0640 6102
04 b601 a901
06 e050 0000
CPU DATAPATH
PW
ENABLE
CLOCK
CLEAR


R_W_Enable
4 X 8 REGISTER FILE
R_ADDR1
R_ADDR2
W_ADDR
W_DATA
Decd
MUX
MUX
R_DATA1
R_DATA2
CLOCK
CLEAR

Here $r1 is loaded with 0x08.

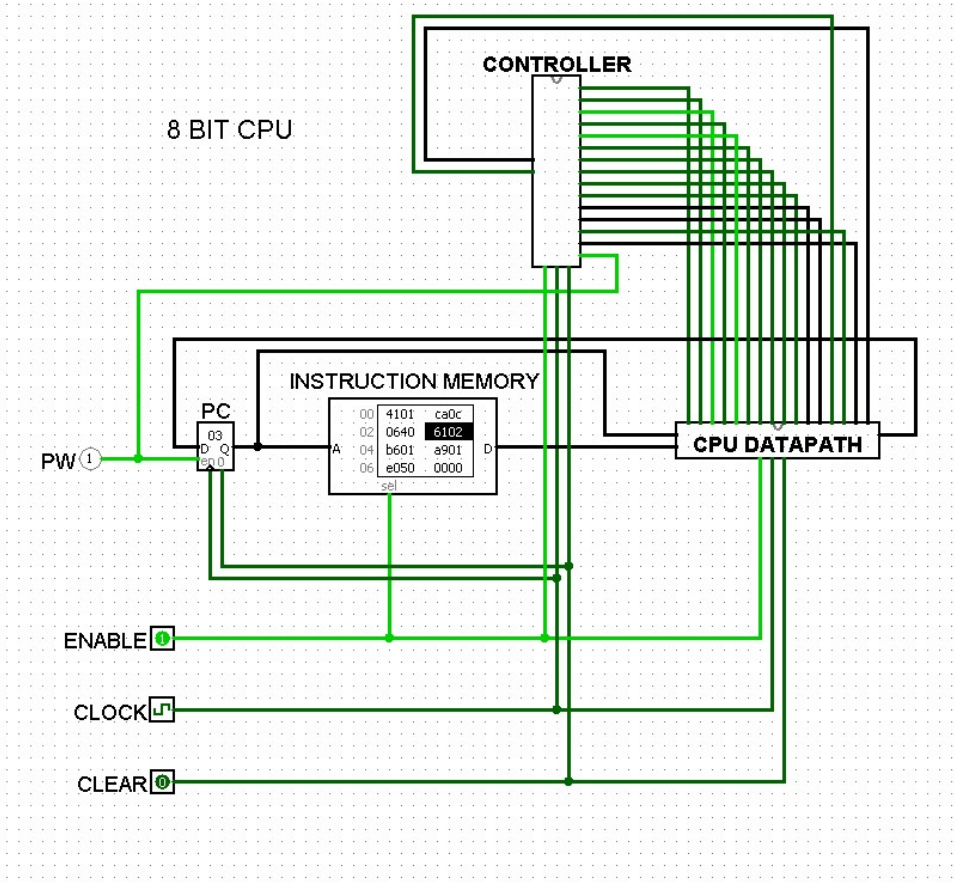
After 4 cycles for Add immediate:

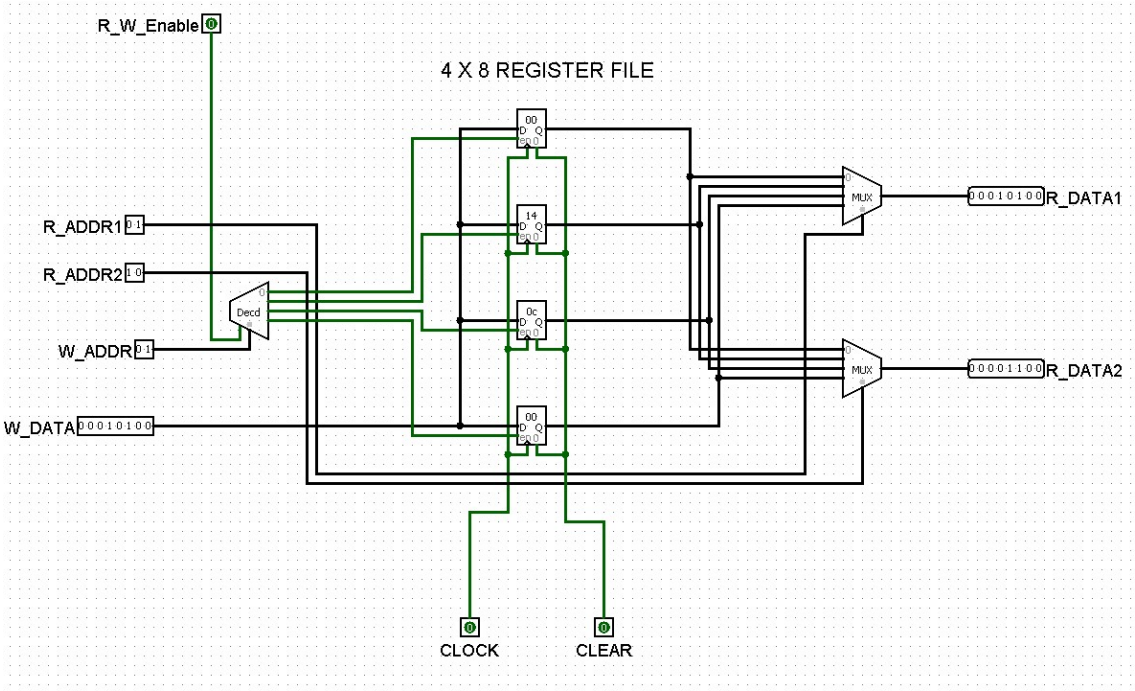




Here $r2 is loaded with 0x0C.

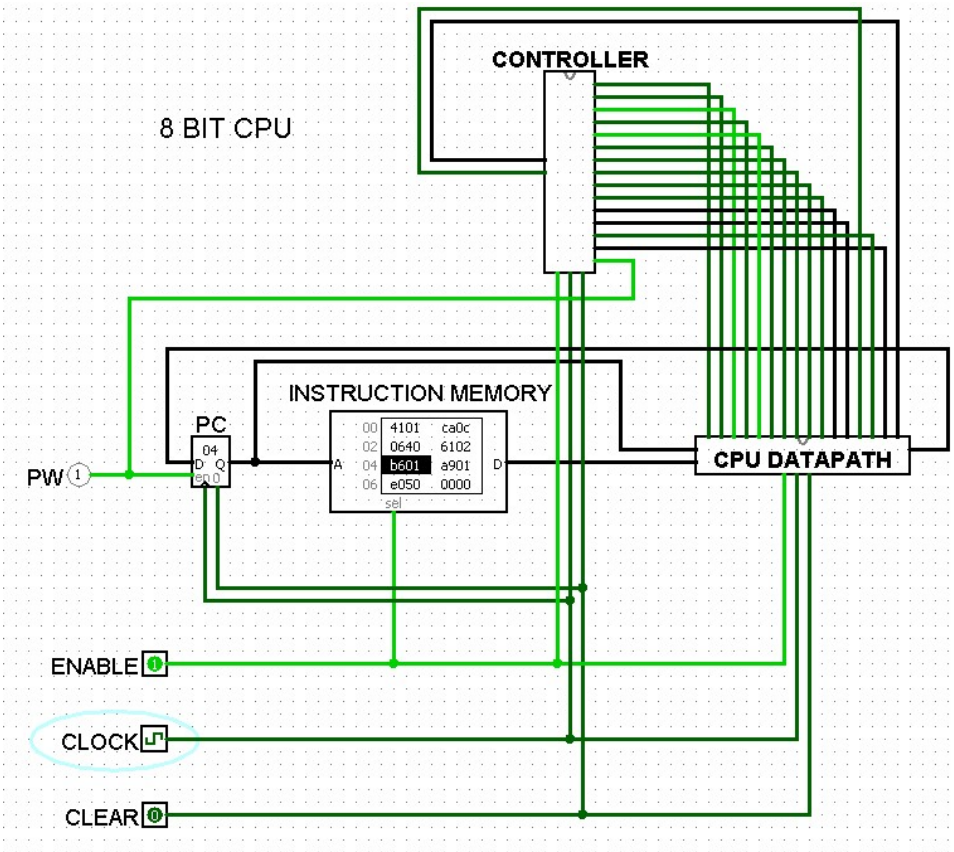
After 4 cycles for Add instruction:

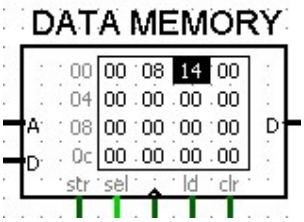




Here $r1 = 0x14 = 0x08 + 0x0C.

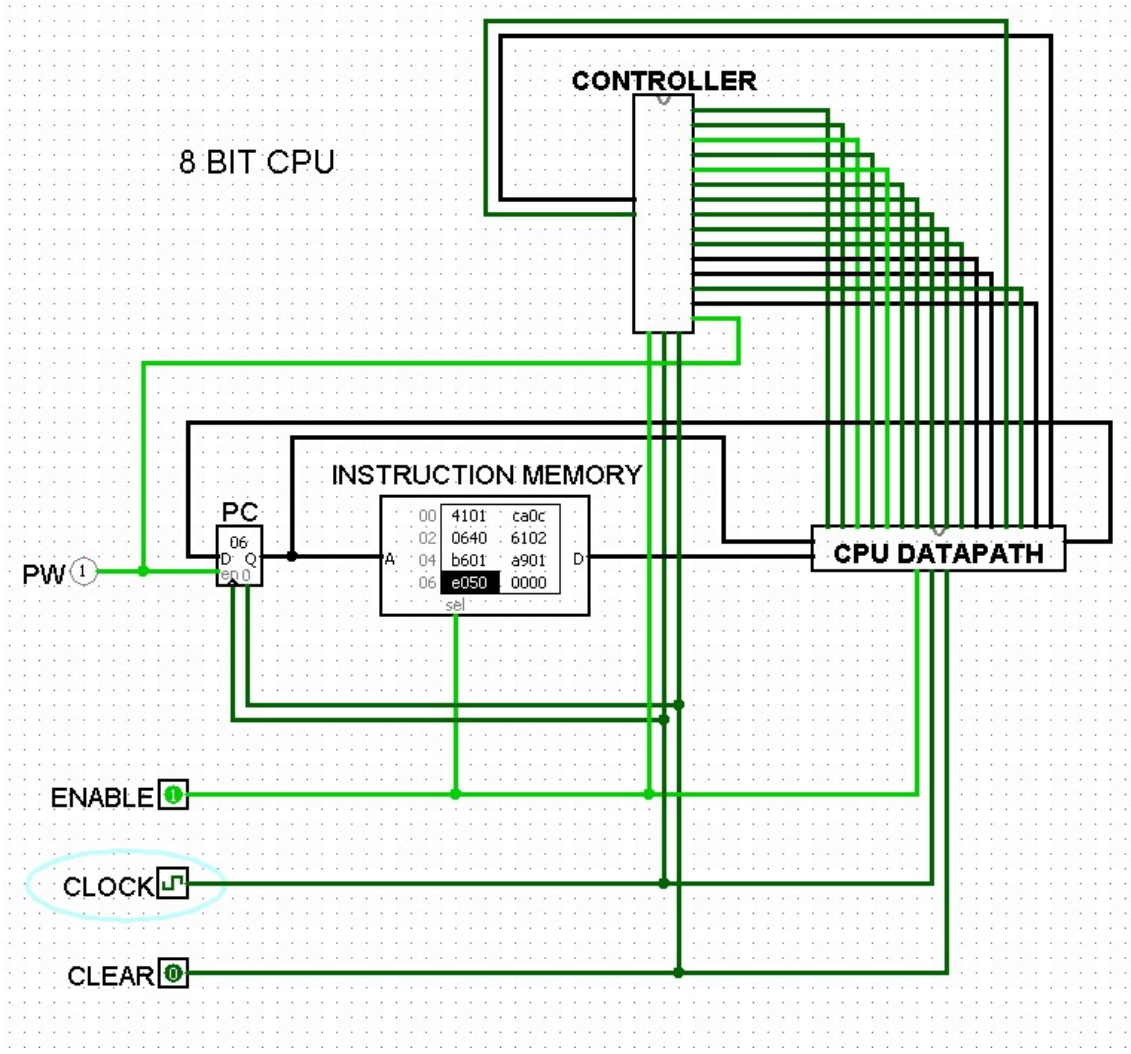
After 4 cycles for store:





Here, contents of $r1 = 0x14 is stored in M[0x02].

After 3 cycles for branch:



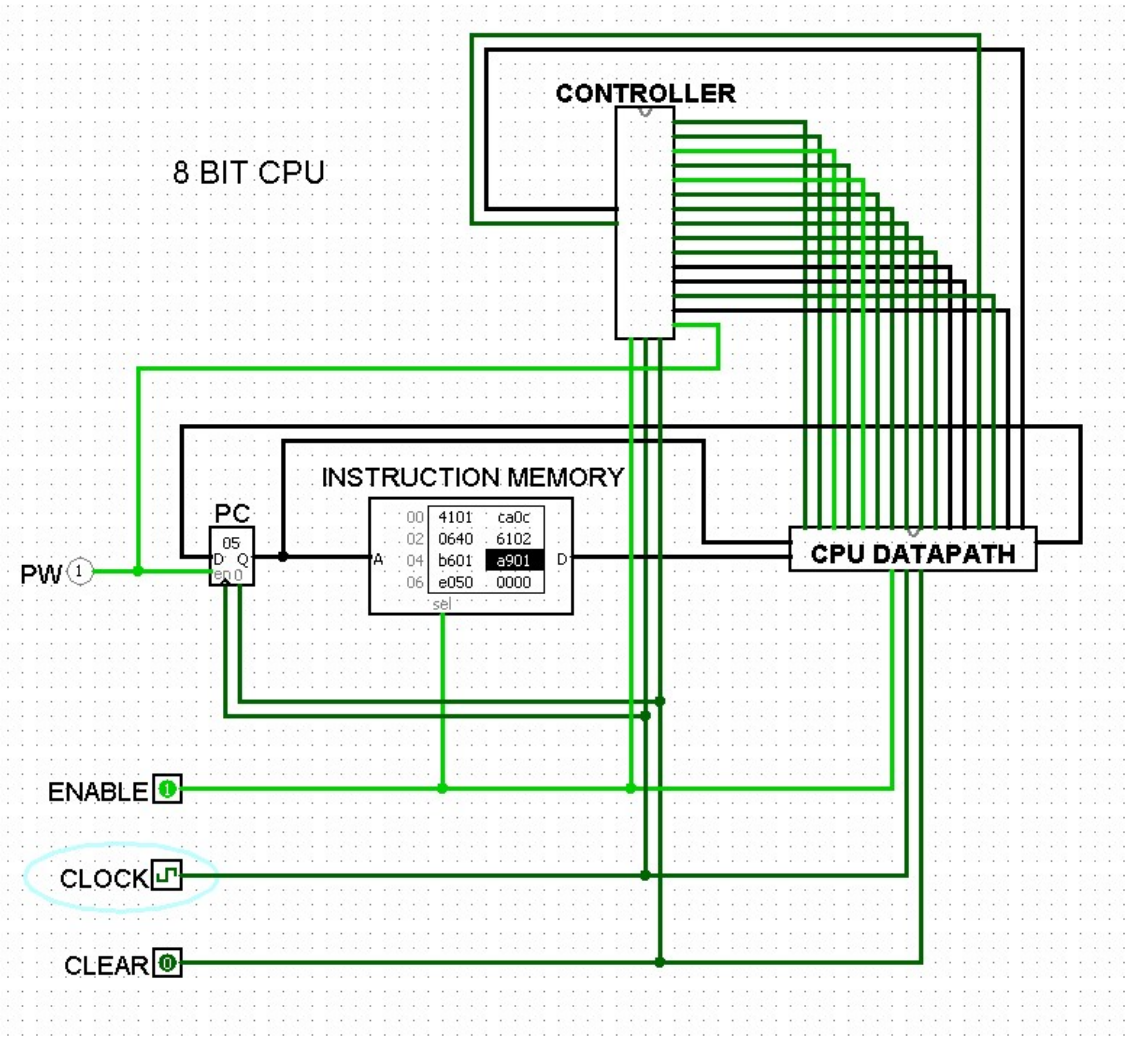
Here as $r1 > $r2, the branch operation is true,

So new PC = PC* + 0x01,

Where PC* = 0x04 + 0x01 = 0x05,

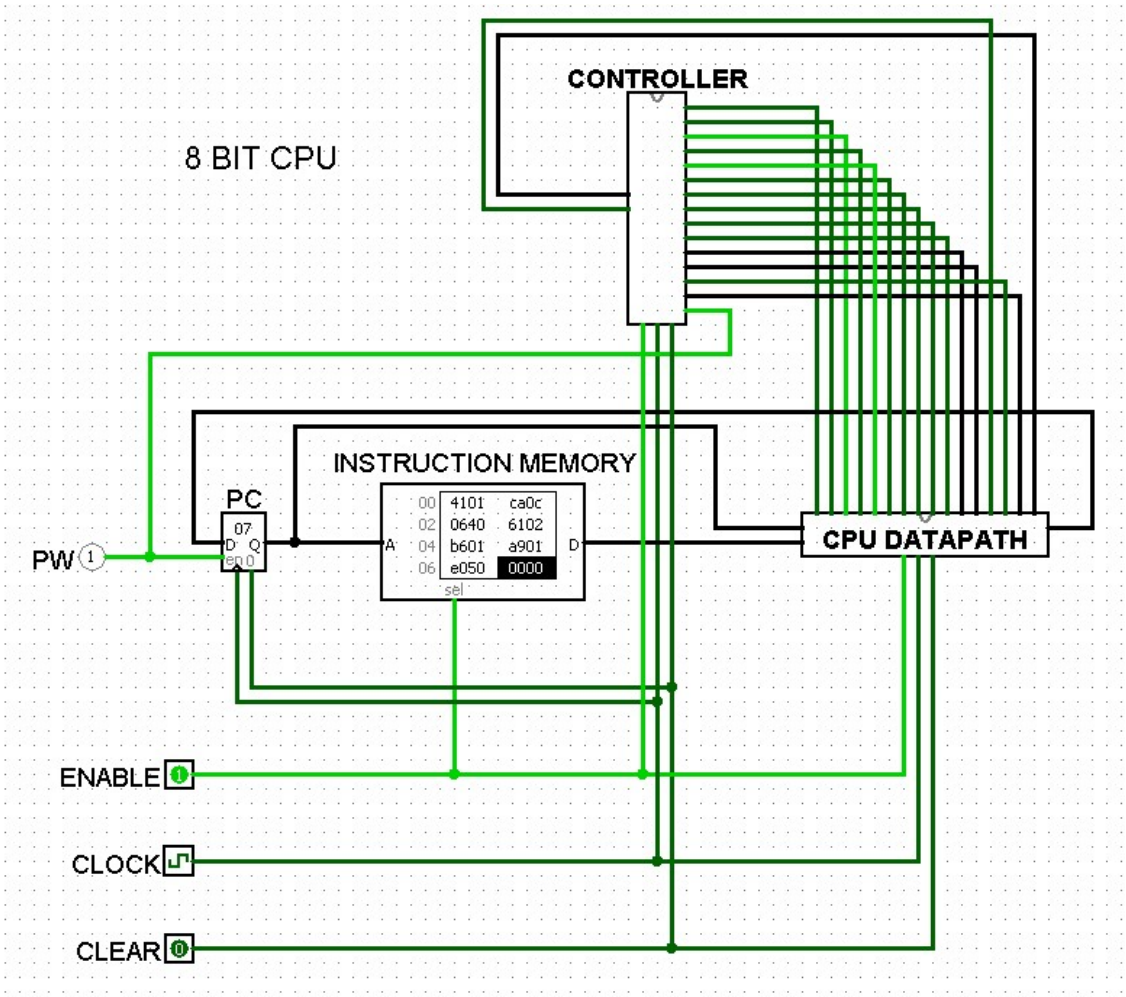
Hence, new PC = 0x06.

After 3 cycles for jump:



Here new PC = 0x05.

After 3 cycles for branch:



Here as $r2 < $r1, the branch operation is true,

So, new PC = PC* + 0x01,

Where PC* = 0x05 + 0x01 = 0x06,

Hence new PC = 0x07.
