IIT Bhubaneswar
School of Electrical Sciences
COA Lab (0 - 0 - 3)
Autumn 2020
Lab Schedule: Thu (10AM -1PM)

Instructor: Debi Prosad Dogra (dpdogra@iitbbs.ac.in)
Teaching Assistant : Shivani and Rohit

Submission Deadline: 11-11-2020 (midnight)

Assignment 6 (Mini Project)

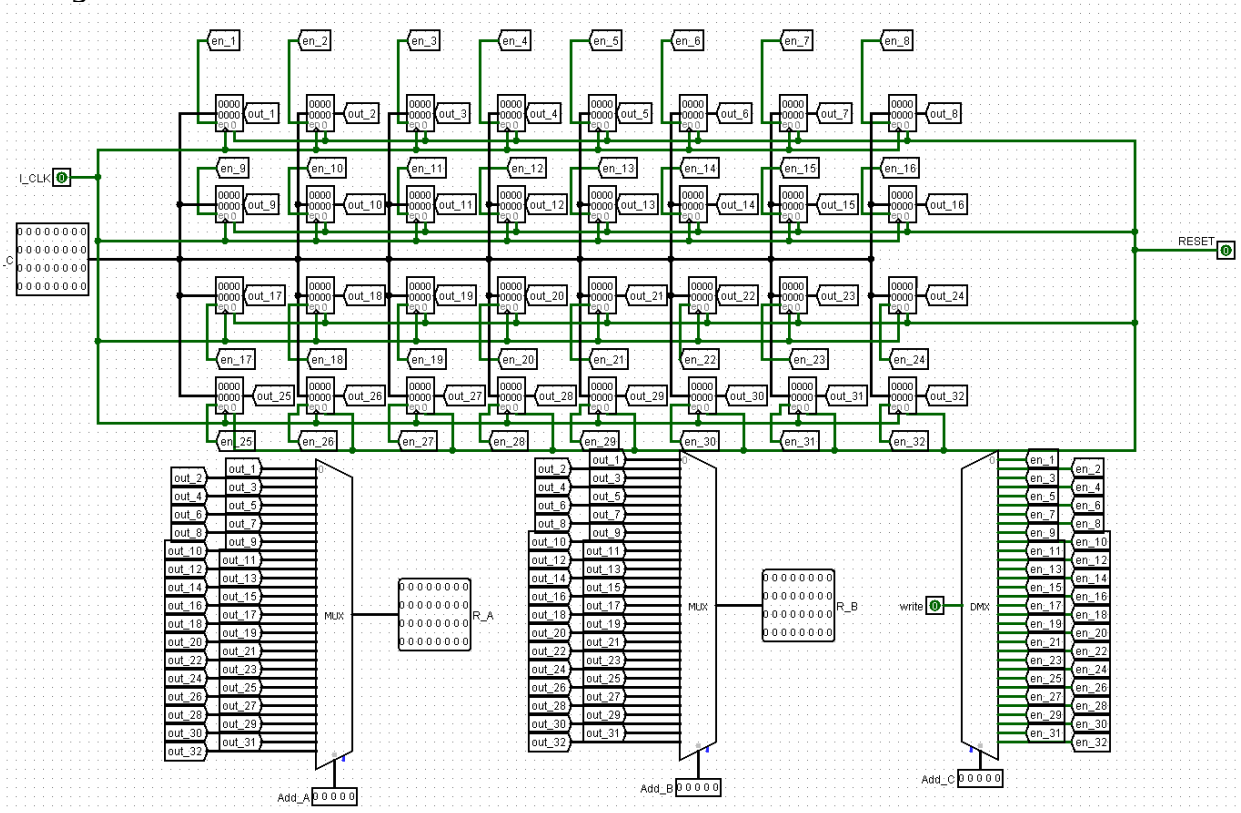
Points: 200

Name: Dushyanth

Roll No.:18CS01009

Block-level diagram of the implementation and over-all 5-stage architecture with all necessary stages, intermediate registers, and other components.

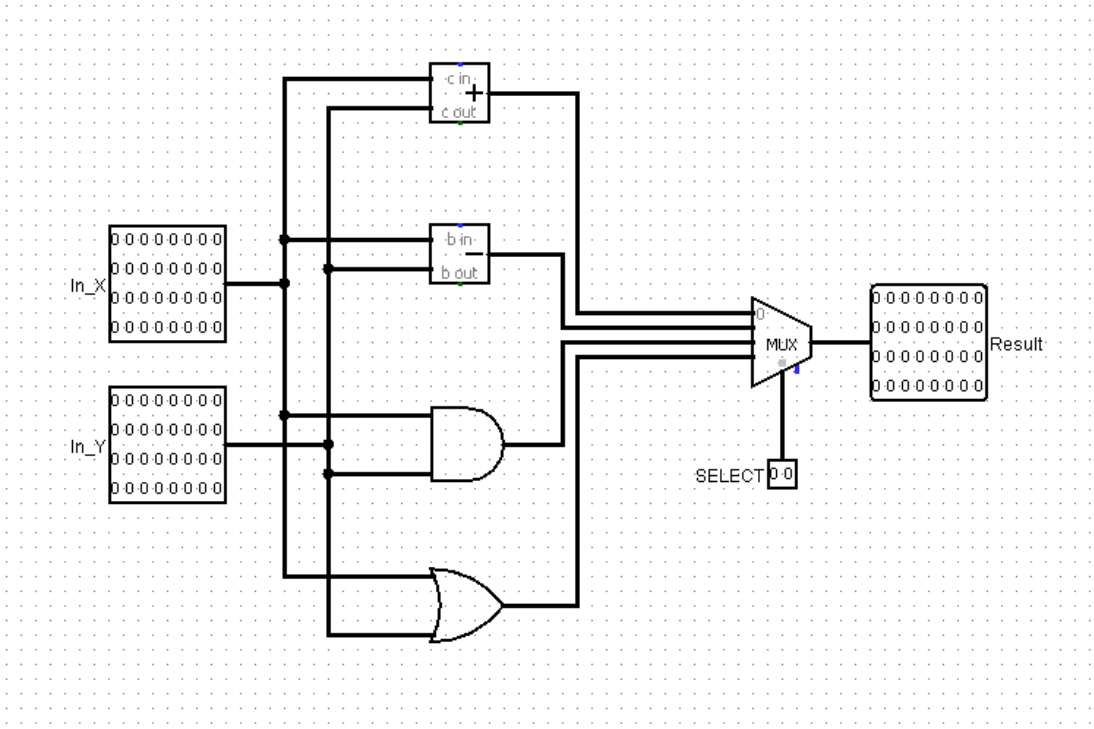
Logisim circuit diagram for Register file: (Tunnel feature of Logisim is used to reduce the wiring)



R_A and R_B contains the data at Add_A, Add_B registers(used 1 for each R_A and R_B).
If write signal is zero then the 32-GPRs of the register file aren't enabled.
If write signal is one then a register represented by Add_C is enabled and the data is written into it.
Reset is used to clear the data in register file after program execution.

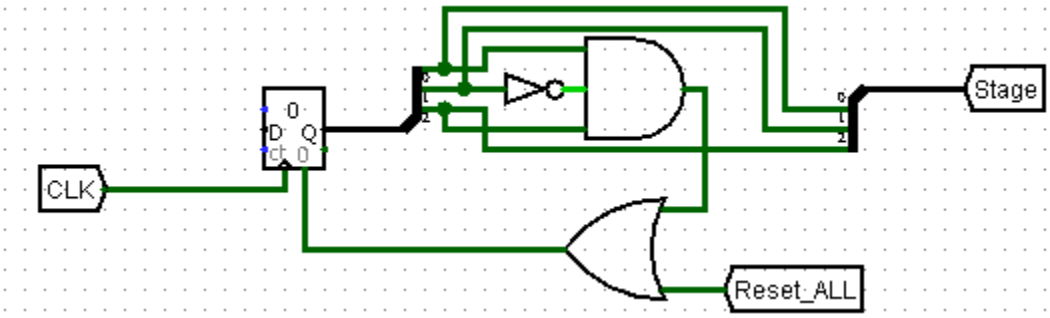
The above circuit is implemented in separate file-“Register_File” and is used in main circuit(hence during program execution the register values aren’t visible in the registers of “Register_File” circuit).

Logisim circuit diagram for ALU:



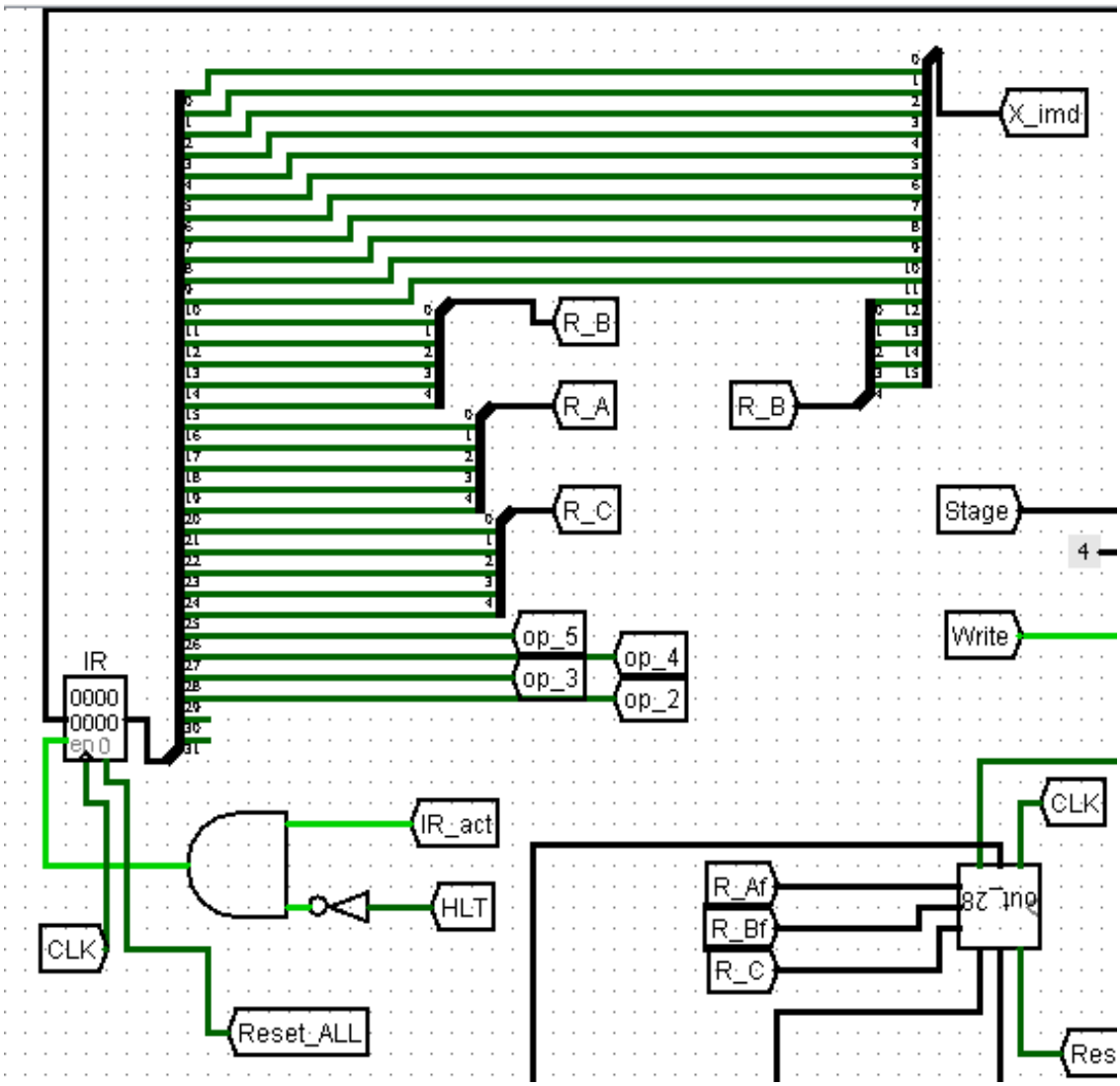
2bit-Select for MUX
0 0 - ADD
0 1 - SUB
1 0 - AND
1 1 – OR
If one of the operands of ADD/SUB/AND/OR is immediate then the 16-bit immediate is extended to 32-bit using zero padding and sent to register in the main circuit.

To keep track of five stages the below counter is used in “main” circuit:



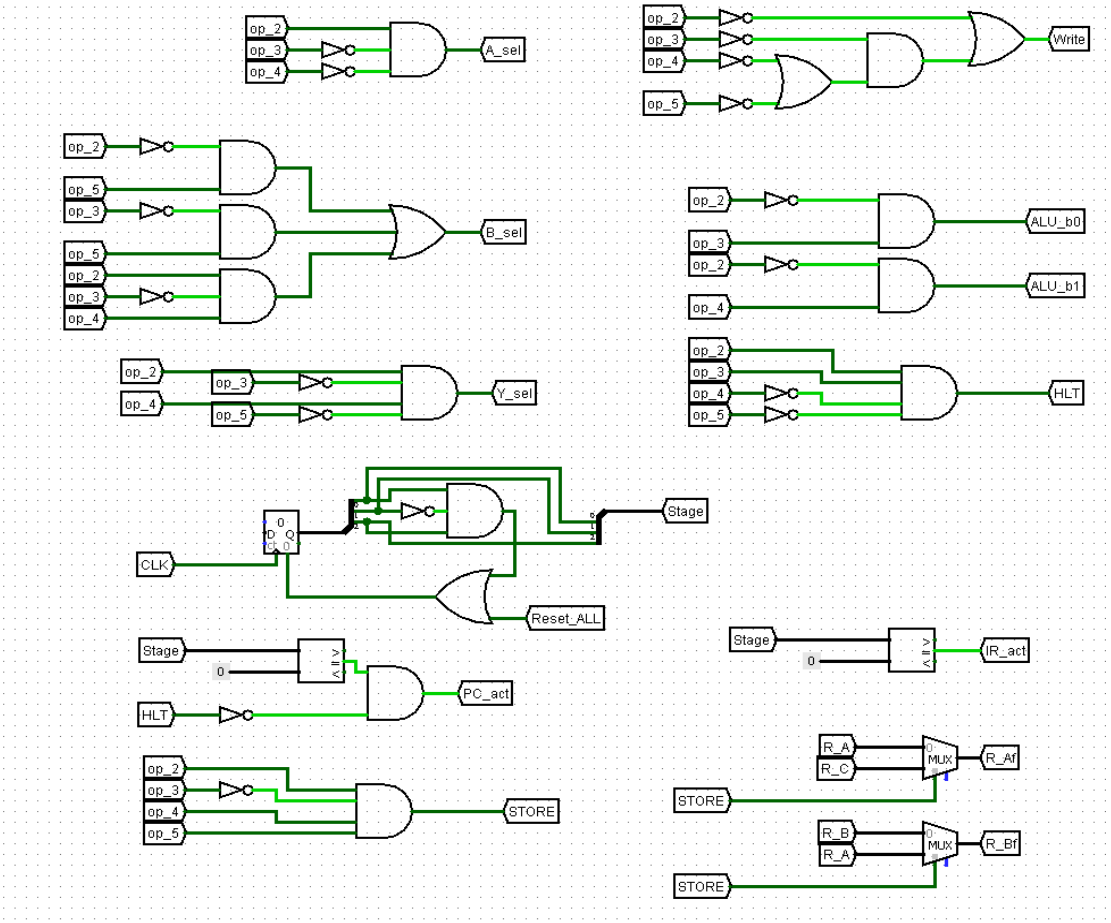
Stages are numbered as 0,1,2,3,4

Decode stage:



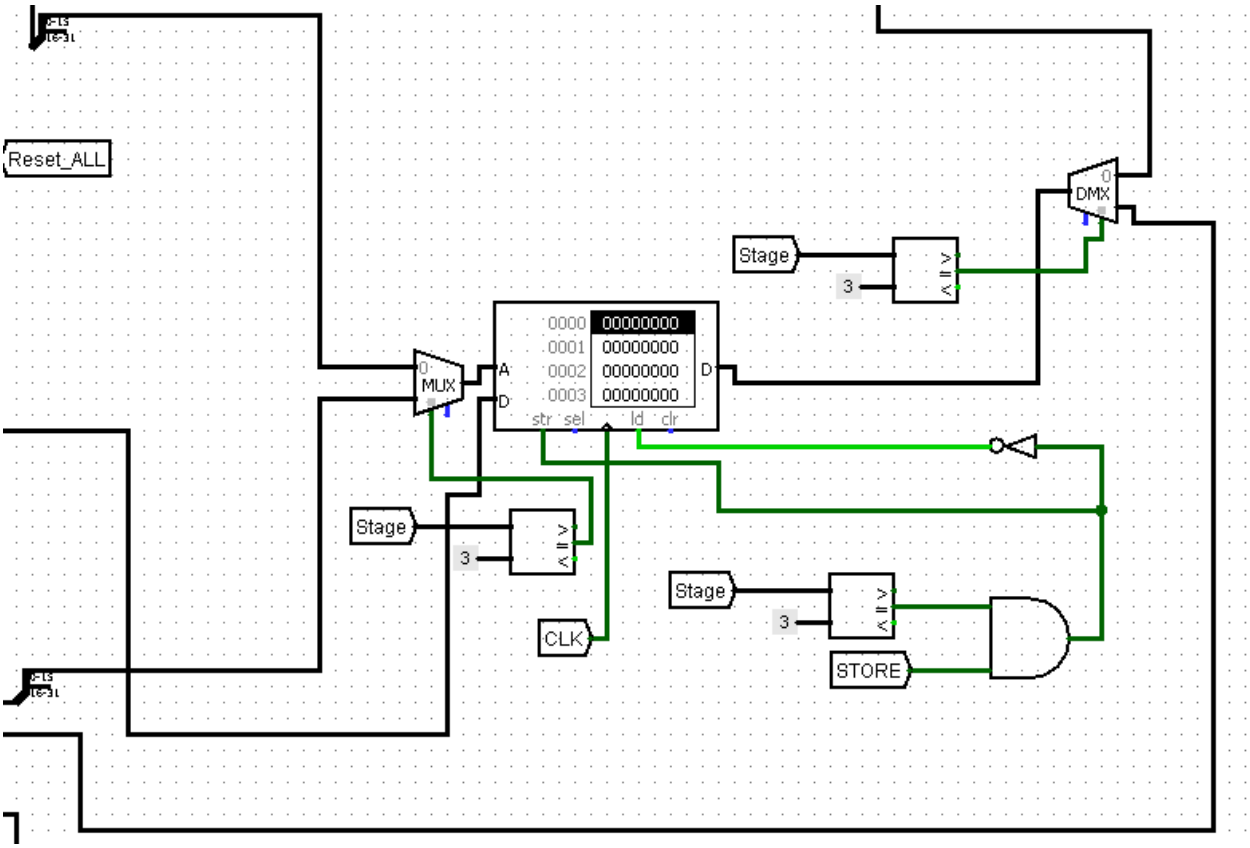
The op_code, operands, Immediate are collected from 32-bit Instruction.

Controls Signal generation:



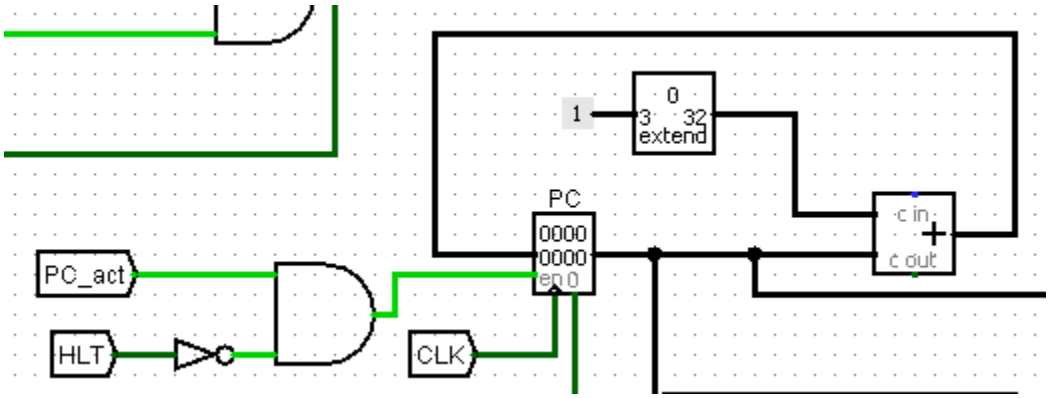
- A_sel is used as select bit for MUX-A that loads first operand into ALU(either 0 or Ra)
- Zero is selected by MUX-A in case we need to pass a value from stage 2 to stage 3 without performing any ALU operation(Rz = 0 + Rb/X, Rb (or) X is passed to Rz)
- B_sel is used as select bit for MUX-B that loads second operand into ALU(either immediate or Rb)
- Y_sel is used as select bit for MUX-Y
- ALU_b0 and ALU_b1: decides the ALU operation
- IR_act: enable for Instruction register
- PC_act: enable for program counter
- HLT, STORE instructions are decoded.
- WRITE is used to generate a write signal for register file

Memory stage:

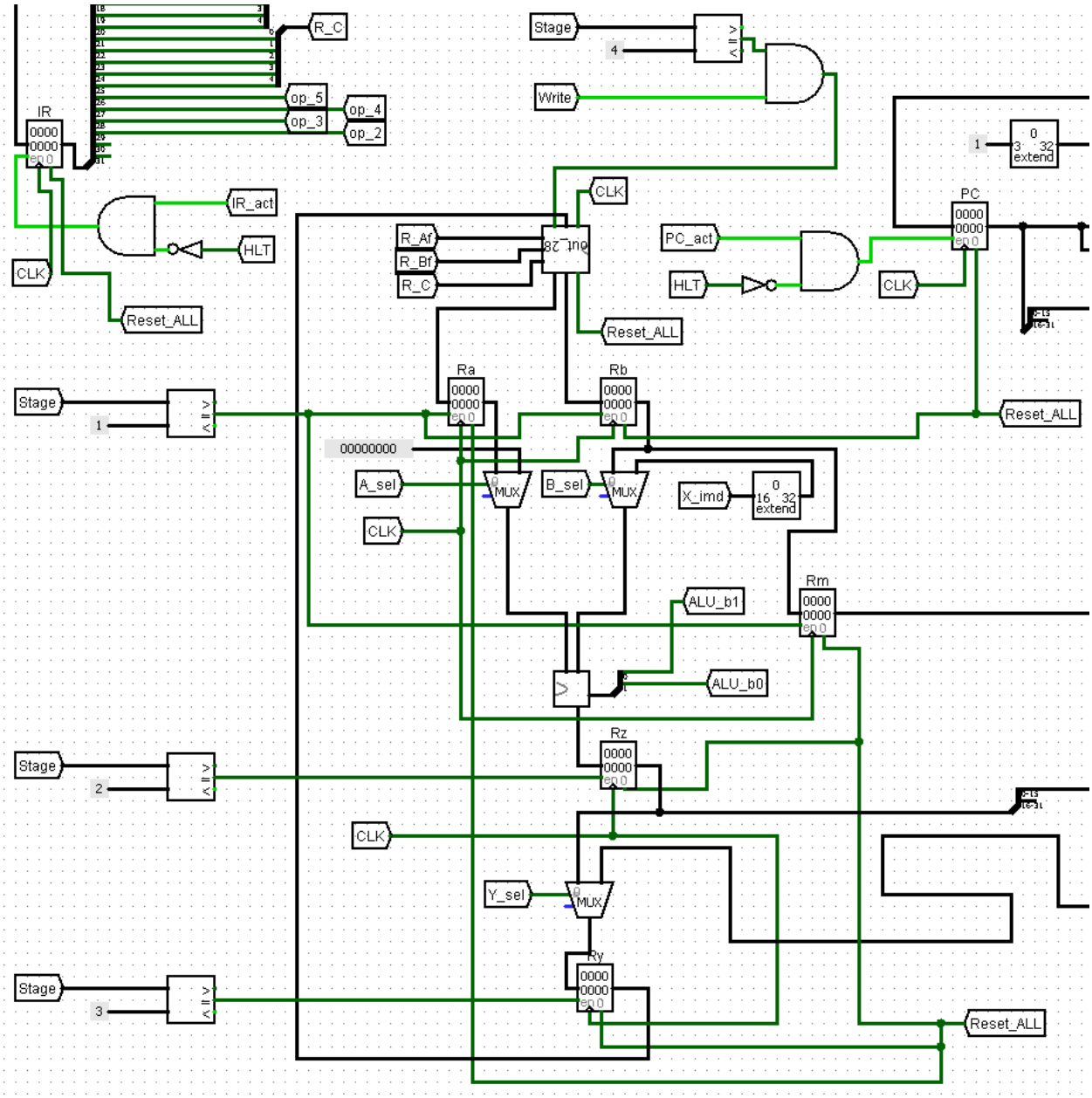


- The address from PC is split into 0-15 and 16-31 bits because the RAM has only 16-bit address.
- The input Address to RAM is multiplexed between PC and Memory address(required in stage 4, hence a comparator with 3 is used as a select for MUX)
- The data from RAM is de-multiplexed to send to IR and the Ry using a select bit.

Incrementing PC unless a HLT instruction is encountered:

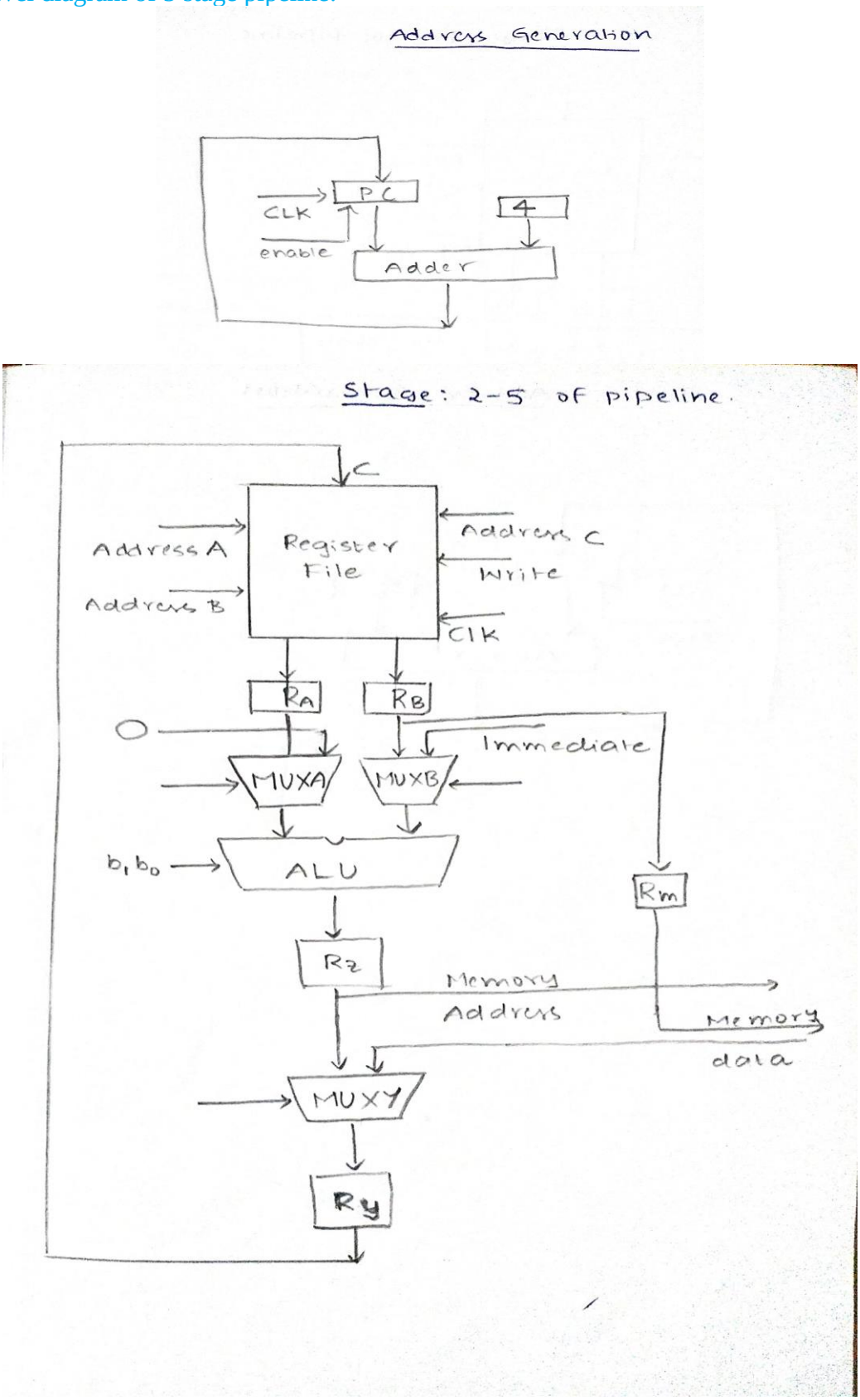


Stages 2-5:



Comparators are used to keep track of stage.

Block level diagram of 5 stage pipeline:



Clock and enable signals are used for registers.

Sample program-1:

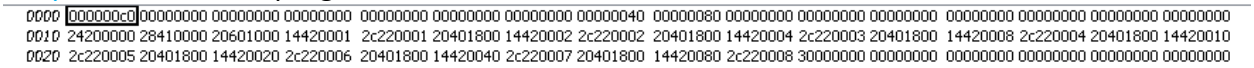
```
MOVE R1, 00
LOAD R2, 0(R1)
MOVE R3, R2
ANI R2, R2, 1
STORE R2, 1(R1)
MOVE R2, R3
ANI R2, R2, 2
STORE R2, 2(R1)
MOVE R2, R3
ANI R2, R2, 4
STORE R2, 3(R1)
MOVE R2, R3
ANI R2, R2, 8
STORE R2, 4(R1)
MOVE R2, R3
ANI R2, R2, 16
STORE R2, 5(R1)
MOVE R2, R3
ANI R2, R2, 32
STORE R2, 6(R1)
MOVE R2, R3
ANI R2, R2, 64
STORE R2, 7(R1)
MOVE R2, R3
ANI R2, R2, 128
STORE R2, 8(R1)
HLT
```

The first instruction MOVE R1, 00 is Move immediate instruction, handled differently(op_code is different from MOVE Ri,Rj instruction)

The machine code for above program is as follows:

00100100001000000000000000000000 (or) 24200000
00101000010000010000000000000000 (or) 28410000
00100000011000000001000000000000 (or) 20601000
00010100010000100000000000000001 (or) 14420001
00101100001000100000000000000001 (or) 2c220001
00100000010000000001100000000000 (or) 20401800
00010100010000100000000000000010 (or) 14420002
00101100001000100000000000000010 (or) 2c220002
00100000010000000001100000000000 (or) 20401800
00010100010000100000000000000100 (or) 14420004
00101100001000100000000000000011 (or) 2c220003
00100000010000000001100000000000 (or) 20401800
00010100010000100000000000001000 (or) 14420008
00101100001000100000000000000100 (or) 2c220004
00100000010000000001100000000000 (or) 20401800
00010100010000100000000000010000 (or) 14420010
00101100001000100000000000000101 (or) 2c220005
00100000010000000001100000000000 (or) 20401800
00010100010000100000000000100000 (or) 14420020
00101100001000100000000000000110 (or) 2c220006
00100000010000000001100000000000 (or) 20401800
00010100010000100000000001000000 (or) 14420040
00101100001000100000000000000111 (or) 2c220007
00100000010000000001100000000000 (or) 20401800
00010100010000100000000010000000 (or) 14420080
10110000100010000000000000001000 (or) 2c220008
11000000000000000000000000000000 (or) 30000000

Output of the above program:



Input: 000000c0 (192 in decimal system)

Output: from location '00000001'

00000000
00000000
00000000
00000000
00000000
00000000
00000040(4*16=64)
00000080(8*16=128)

Note: The program is loaded at 00000010 location and the input is given at 00000000 location in RAM (addresses and Instructions are represented as hexa-decimal numbers).

Sample program-2:

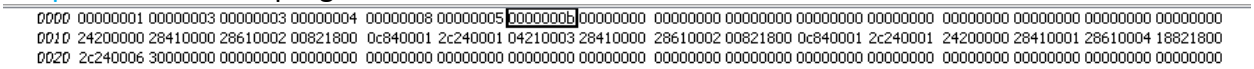
```
MOVE R1, 00
LOAD R2, 0(R1)
LOAD R3, 2(R1)
ADD R4, R2, R3
SUI R4, R4, 1
STORE R4, 1(R1)
ADI R1, R1, 3
LOAD R2, 0(R1)
LOAD R3, 2(R1)
ADD R4, R2, R3
SUI R4, R4, 1
STORE R4, 1(R1)
MOVE R1, 00
LOAD R2, 1(R1)
LOAD R3, 4(R1)
OR R4, R2, R3
STORE R4, 6(R1)
HLT
```

The first instruction MOVE R1, 00 is Move immediate instruction, handled differently (op_code is different from MOVE Ri,Rj instruction)

Machine code for above program:

00100100001000000000000000000000	(or) 24200000
00101000010000010000000000000000	(or) 28410000
00101000011000010000000000000010	(or) 28610002
00000000100000100001100000000000	(or) 00821800
00001100100001000000000000000001	(or) 0c840001
00101100001001000000000000000001	(or) 2c240001
00000100001000010000000000000011	(or) 04210003
00101000010000010000000000000000	(or) 28410000
00101000011000010000000000000010	(or) 28610002
00000000100000100001100000000000	(or) 00821800
00001100100001000000000000000001	(or) 0c840001
00101100001001000000000000000001	(or) 2c240001
00100100001000000000000000000000	(or) 24200000
00101000010000010000000000000001	(or) 28410001
00101000011000010000000000000100	(or) 28610004
00011000100000100001100000000000	(or) 18821800
00101100001001000000000000000110	(or) 2c240006
30000000000000000000000000000000	(or) 30000000

Output of the above program:



Input: seven numbers starting from location '00000000'

1,2,3,4,5,5,6

Output:1,3(=1+3-1),3,4,8(=4+5-1),5,11(b= 3|8;'|' is 'or' operation)

The input is modified by the given program:

e1=e0+e2-1

e4=e3+e5-1

e6=e1 or e4

Note: The program is loaded at 00000010 location and the input is given from 00000000 location in RAM