Faculty of Engineering
Fayoum University

Architecture Report

Phase 1: Single Cycle Processor

Under the Supervision of:

Dr. Gihan Naguib

Eng. Jihad Awad

Made By:

- Tasneem Eid
- Shahd Mahmoud
- Maged Aziz

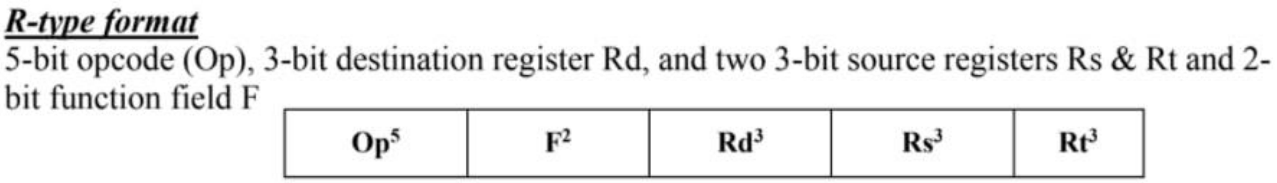
Content of the Report:-

Introduction:

A 16-bit RISC processor with 7 general-purpose registers is to be designed. R0 is fixed to zero and cannot be written to. Additionally, there is one register for the program counter (PC).

All instructions are only 16 bits long, and there are three instruction formats: R-type, I-type, and J-type.

>R-type Structure

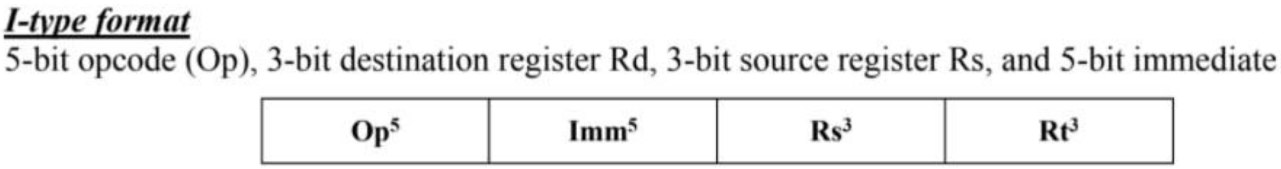


R-type format

5-bit opcode (Op), 3-bit destination register Rd, and two 3-bit source registers Rs & Rt and 2-bit function field F

Op^5	**F**^{2}	**Rd**^{3}	**Rs**^{3}	**Rt**^{3}

>I-type Structure

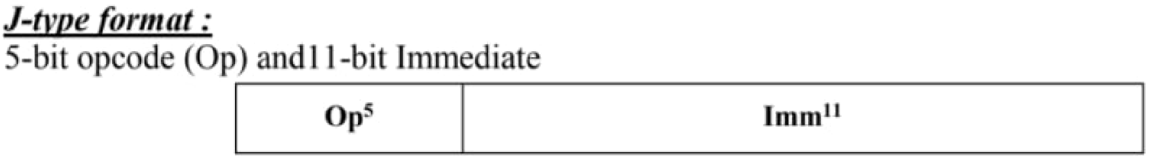


I-type format

5-bit opcode (Op), 3-bit destination register Rd, 3-bit source register Rs, and 5-bit immediate

Op^5	**Imm**^{5}	**Rs**^{3}	**Rt**^{3}

>J-type Structure



J-type format :

5-bit opcode (Op) and11-bit Immediate

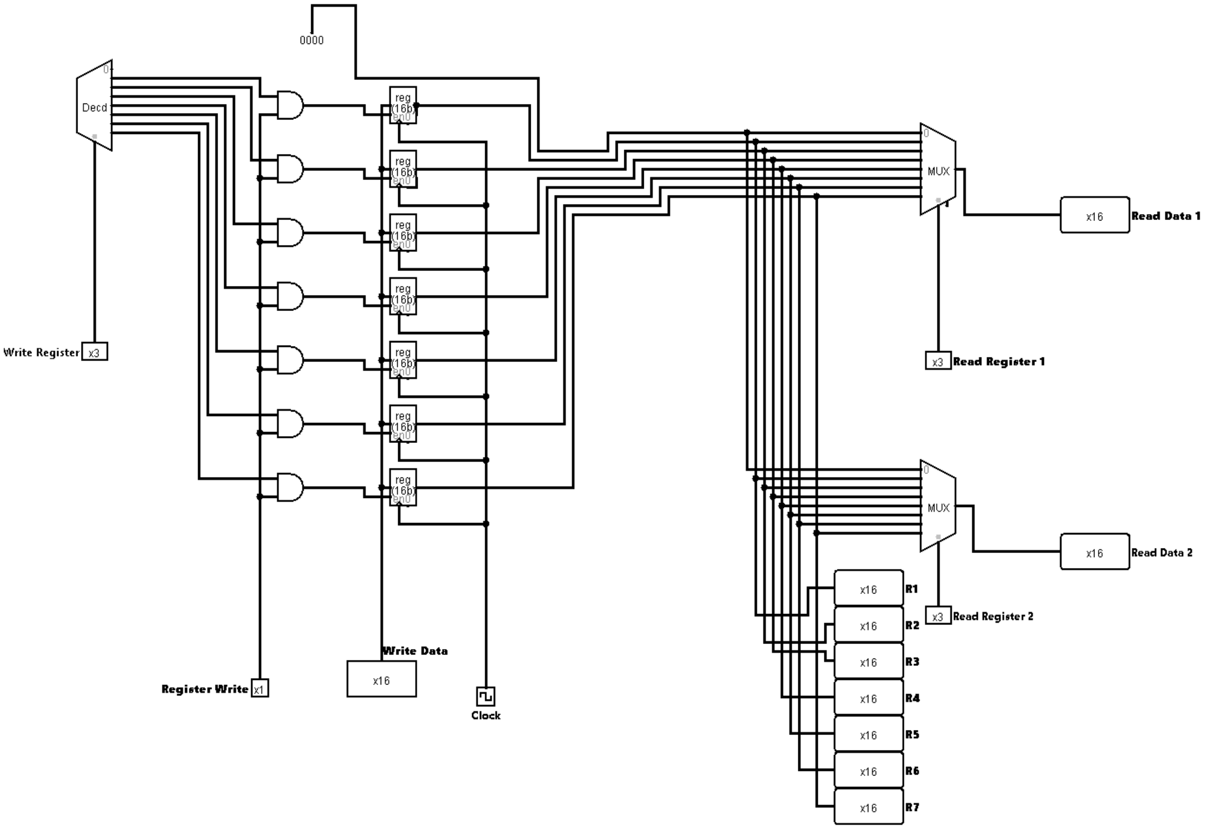
Op^5	**Imm**^{11}

- In the next few pages we will talk about each part and its usage and role in the whole project in the same order of [Content of Project]

⌘ Register File:

- It includes 7 registers of 16 bits each, labeled R1 to R7 (bearing in mind that R0 is permanently hardwired to zero), each with read ports and one write port.

- There are two output buses responsible for carrying the output from the register file, along with the control signal (Register Write), and a clock signal for the register block to announce when a rising edge is incoming.



- The decoder's main usage in this situation is to select which register will be written on it so the decoder lines interact with an AND gate alongside the Write signal to execute the same process previously discussed. Then, on the left side of the register, which serves as the output, we connect it to two multiplexers to select the data that will be on bus A and bus B (Rs, Rt).

⌘ ALU:

- The Arithmetic Logic Unit (ALU) is a fundamental component of a computer's central processing unit (CPU). It performs arithmetic and logic operations on input data according to instructions from the CPU.
- It has two inputs: A and B, and one output, which is the result of the operation performed on A and B.
- control signal is used to select which operation to perform based on each operation's instruction.

-ALUControl 🡻🡻🡻

Operations of ALU (With Mux Pins):

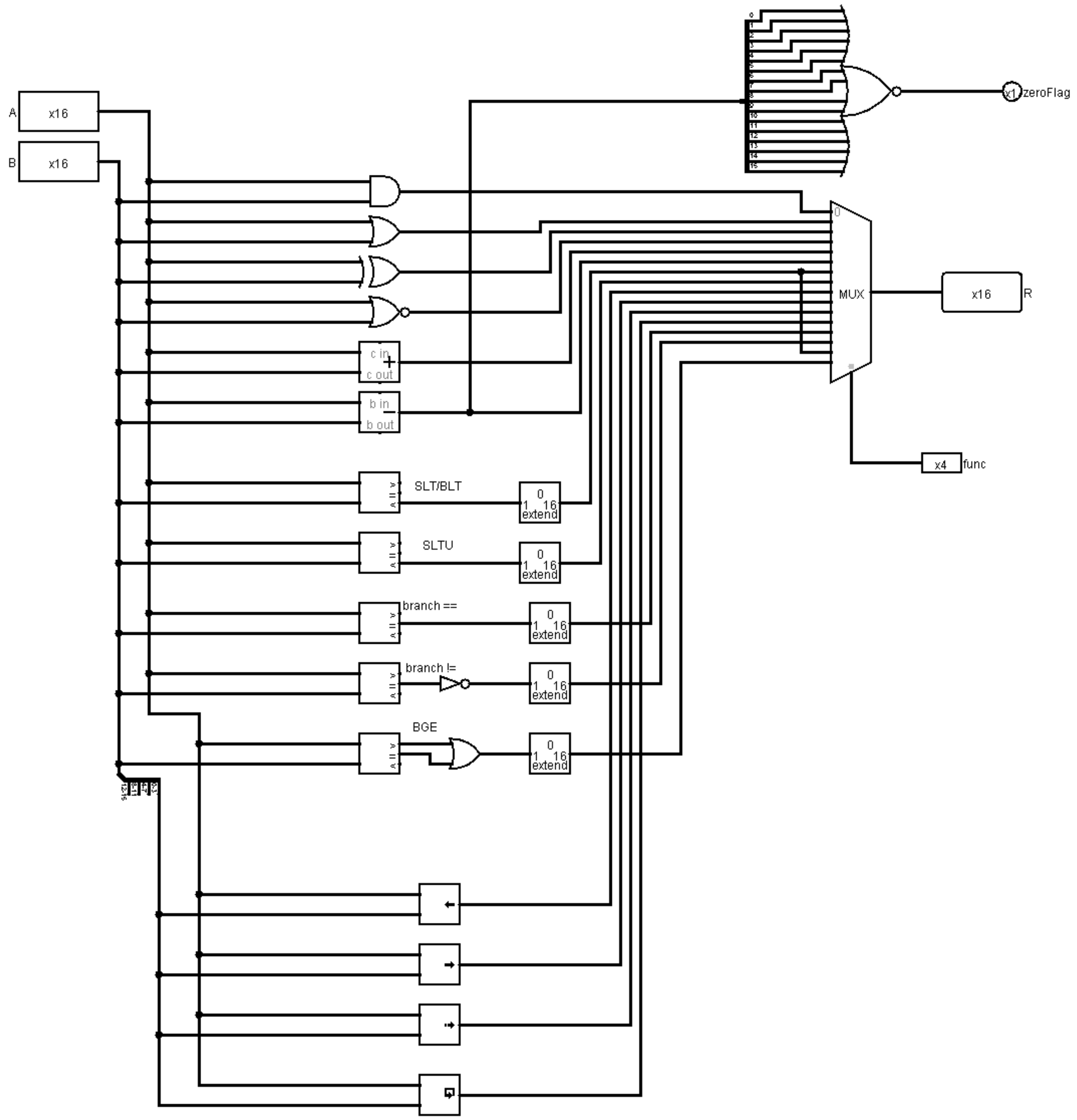
- Pin0 : A and B
- Pin1 : A or B
- Pin2 : A Xor B
- Pin3 : A Nor B
- Pin4 : A add B
- Pin5 : A Sub B
- Pin6: A < B then Output = 1 otherwise Output = 0 [Signed]
- Pin7/14 : A < B then Output = 1 otherwise Output = 0 [Unsigned] / Set Less Than
- Pin8 : A shift left logically by 4
- Pin9 : A Shift Right Logically by 4
- Pin10 : A Shift Right Arithmetically by 4
- Pin11 : A Rotate Right by 4
- Pin12 : A = B then Output = 1 otherwise Output = 0 [Signed]
- Pin13 : A != B [Not Equal] then Output = 1 otherwise Output = 0 [Signed]
- Pin15 : [A < B] OR [A >= B] then Output = 1 otherwise Output = 0 [Signed]

Note: From Pin12 to Pin15 Related to the Branch instruction

- ➢ Notes About Branch:
  - To apply branch instruction we have used the comparator block to compare between the two input then set the operation to give the needed output
  - The output of the comparator is always 1-bit so we did use a bit extender to make it 16-bit to match with the mux and results data bits

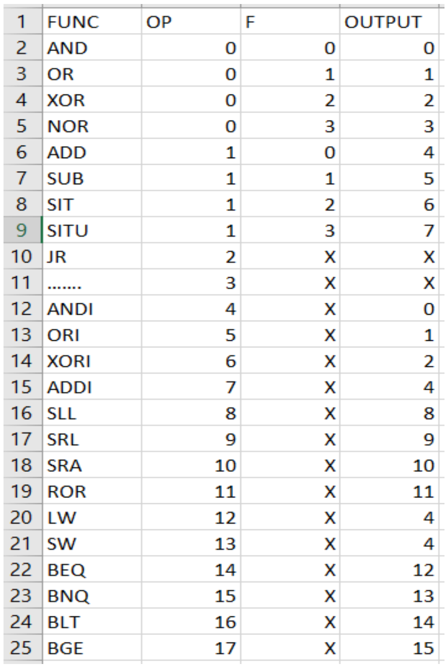
- Note about R-type and I-type:
  - The operation of I type that also exsist at Rtype as ADDI and ORI and XORI … will be at the same pin as R type means that the operation of and ,, andi will be at the same pin (pin 0) and the same for OR and ORI and so on

Main ALU :-

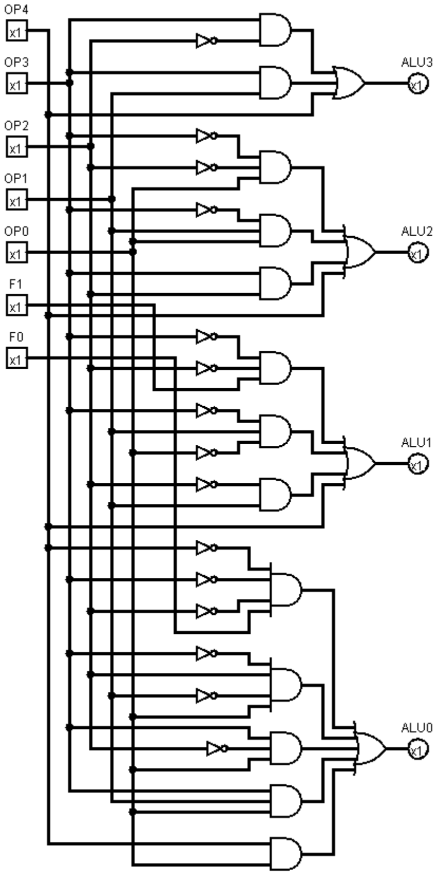


The ALUcontrol will be the signal at the selector of this mux to select the proper operation

And here is it’s truth table:-

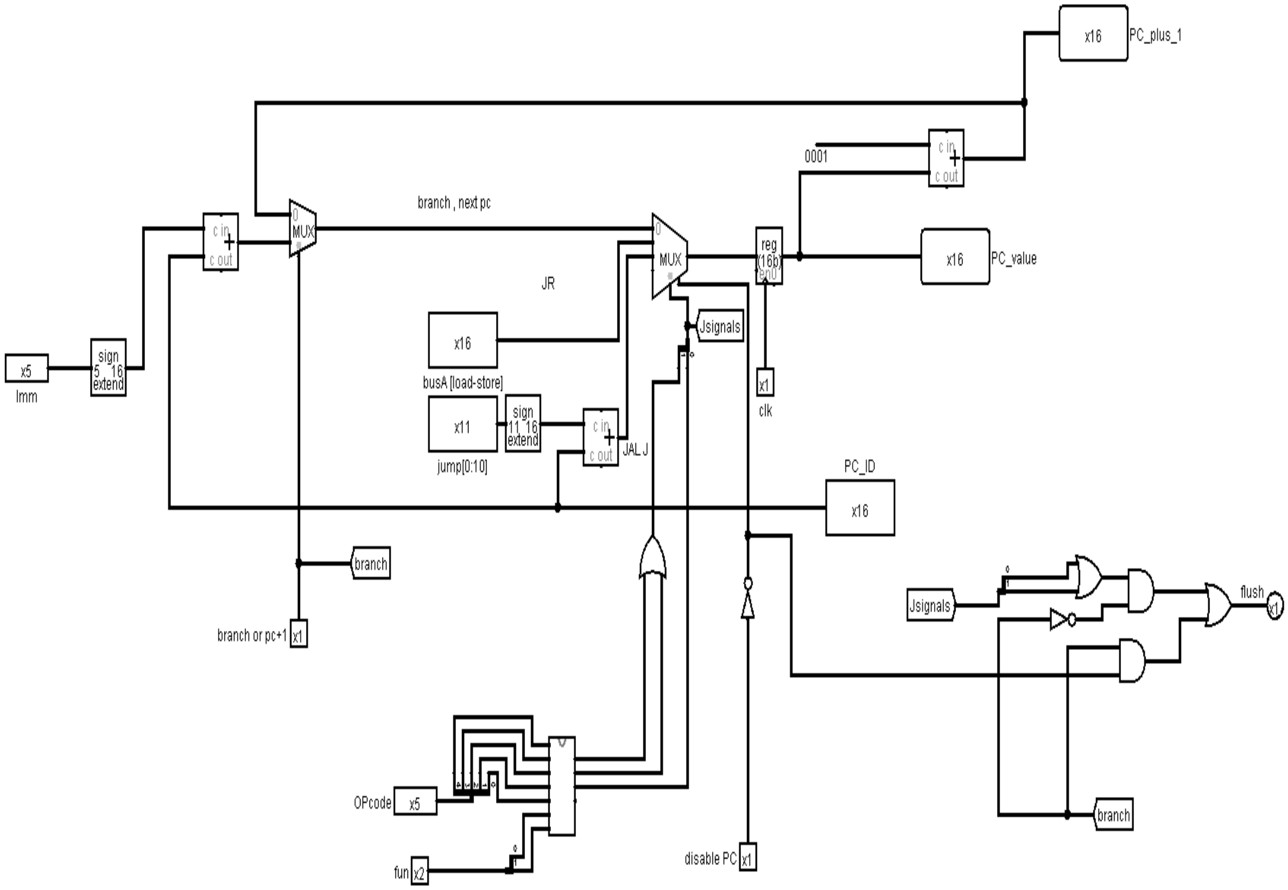


1	FUNC	OP	F	OUTPUT
2	AND	0	0	0
3	OR	0	1	1
4	XOR	0	2	2
5	NOR	0	3	3
6	ADD	1	0	4
7	SUB	1	1	5
8	SIT	1	2	6
9	SITU	1	3	7
10	JR	2	X	X
11	…….	3	X	X
12	ANDI	4	X	0
13	ORI	5	X	1
14	XORI	6	X	2
15	ADDI	7	X	4
16	SLL	8	X	8
17	SRL	9	X	9
18	SRA	10	X	10
19	ROR	11	X	11
20	LW	12	X	4
21	SW	13	X	4
22	BEQ	14	X	12
23	BNQ	15	X	13
24	BLT	16	X	14
25	BGE	17	X	15



⌘ Program Counter [PC] :

The PC, or Program Counter, is a component used to calculate the next program counter during program execution, including for branching or jumping.



Left Mux:

Select 1 if >>>>>	Last 2Bit Of AluOP= 11	the instruction isn't LUI [usage of OR]	AluResult [first bit]= 1	Select 0 Otherwise: PC + 1

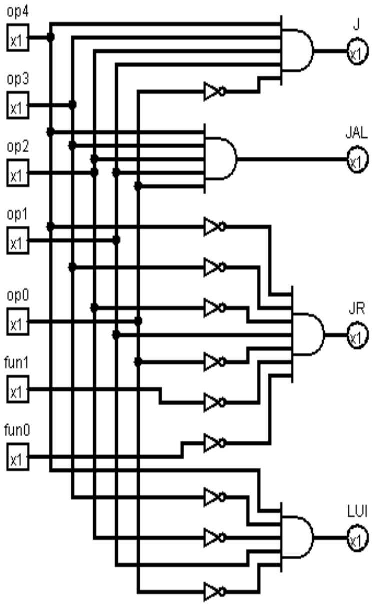
Before explaining how the right Mux works we should first break down the signals which are the main thing for selecting the appropriate operation

Jump Signals:

Jump signal's main operation is to choose which to activate between

[J, JAL , JR , LUI] depending on the OP code and Function code

Note: [Function code] is mainly related to JR instruction only



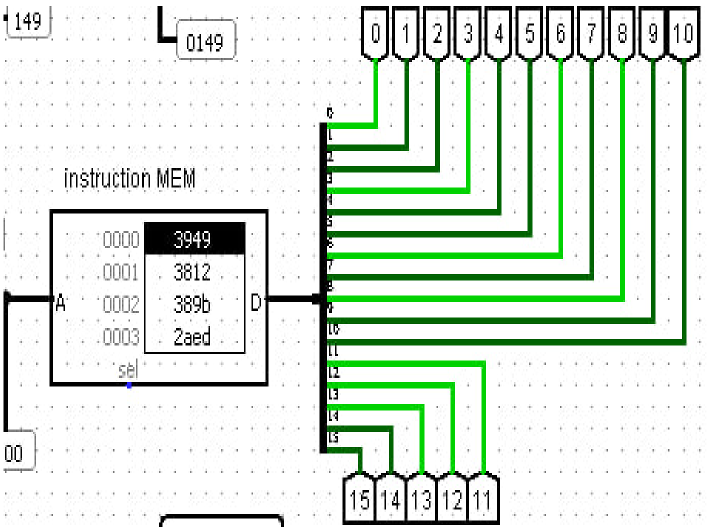
Right Mux:

Select 2 if >>>>>	OP of JAL or J	JAL |> R7 = PC +1 , PC =PC + signed[imm11]
Select 1 if >>>>>	OP and FN of JR	JR |> PC = Reg(Rs)
Select 0 if >>>>>	otherwise	Branch or PC + 1 from left MUX

After finishing the muxes explanation now it's turn for the register, PC register is to save the value of the current program counter and after executing the instruction we increase the PC by 1 to point to the next instruction

⌘ Instruction Memory:

- It is a ROM Read-Only Memory that we need for users to store data permanently so it is so important for us, we saved data in ROM as four hex numbers.
- It operates on a word-by-word basis, and we receive input A from the succeeding program counter [PC]. Upon activating the chip select signal, it yields a 16-bit output,



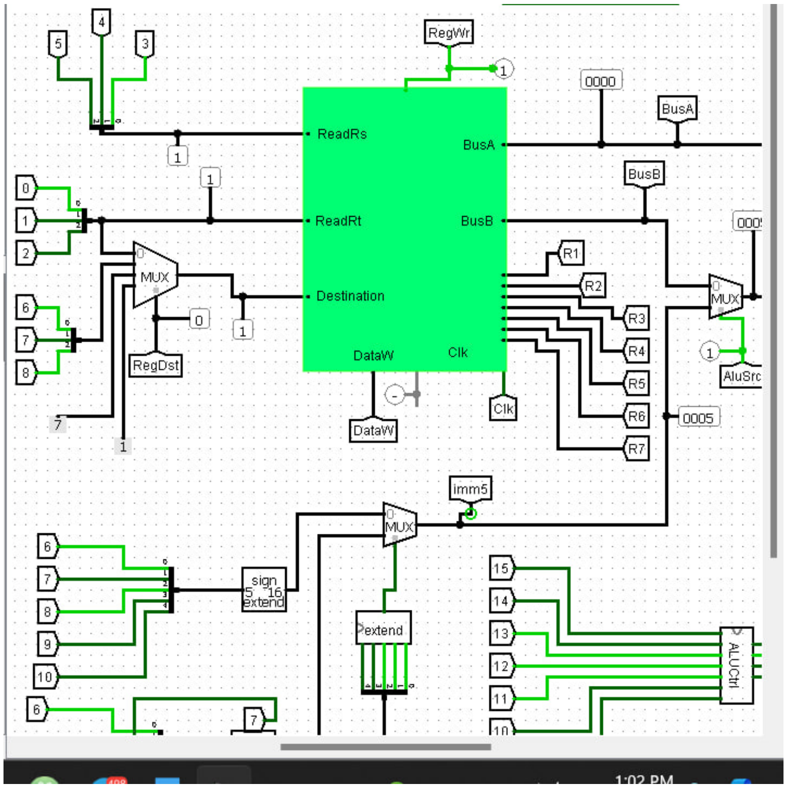
⌘ DataPath:

The data path in a CPU refers to the component responsible for the movement and manipulation of data during instruction execution. It comprises various functional units such as registers, arithmetic logic units (ALUs), multiplexers, and buses. The data path facilitates the flow of data between these units, allowing the CPU to perform arithmetic and logical operations, access memory, and execute instructions. It plays a crucial role in the overall operation and performance of the CPU.

Register File Data Path:

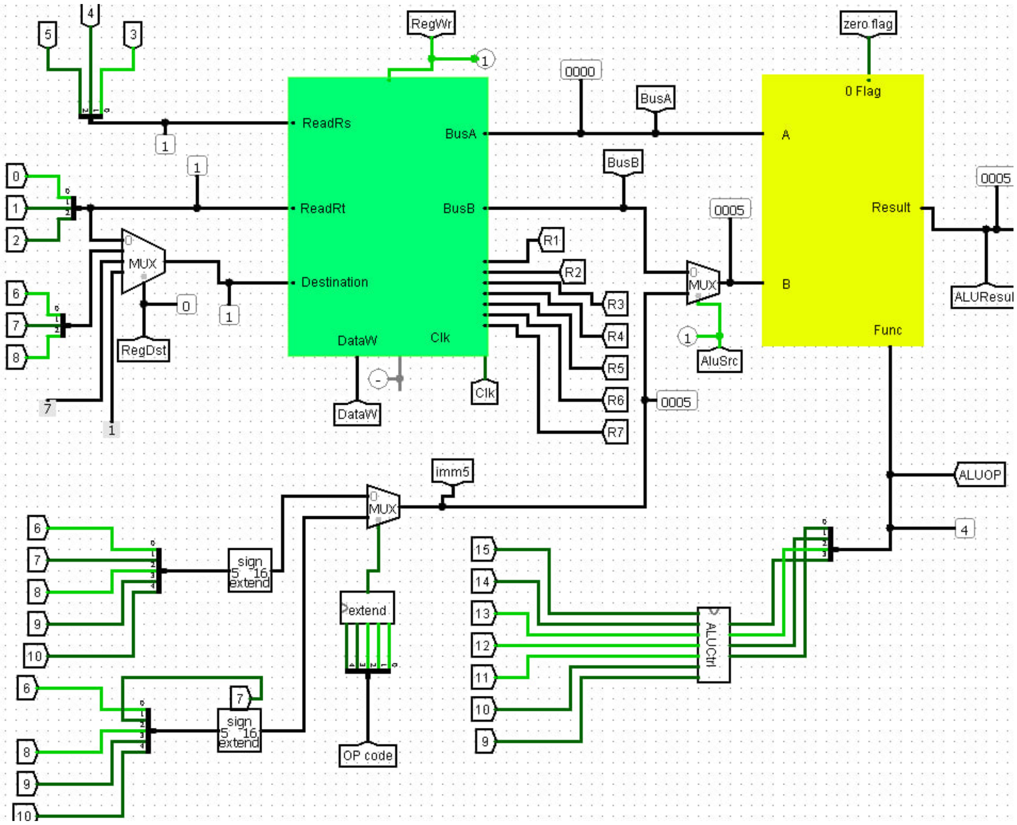
• As we can see, the left mux chooses between Rt or Rd or R7 or R1 to be the DESTINATION register. Therefore, we use a Mux to select which register to write on. We have one control signal for that, which is [RegDst].

- Another signal that’s crucial for the register file is RegW at the top of it, it permits the register to write on it



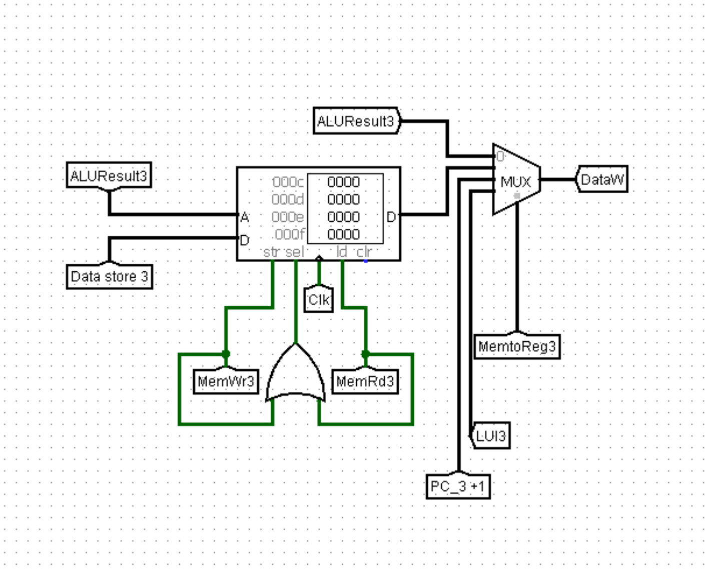
ALU Data Path:

- ALU has two inputs that come from the register file , the first bus A always come from Read data 1 [BUS A] which has Rs value
- Bus B can be the data of the second Bus from the register file or the immediate value so we should use a mux to select one to access the ALU
- If the control signal (ALU src) is set to be one the immediate value will enter the Alu from BUS B Except that the data from the register file will enter we also denote that we should extend the immediate sign extend for Arithmetic operations and extend with zero for non-Arithmetic operations before entering the ALU to do that we AND The sign bit with the control signal (extend).



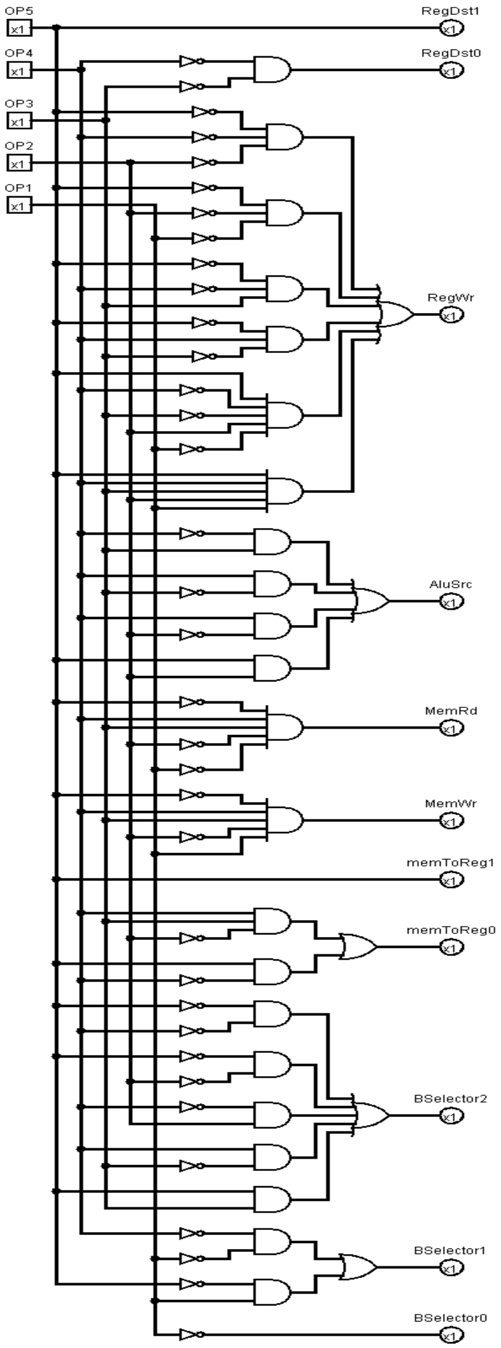
⌘ Instruction Memory:

- The result of the ALU will be the address in the memory to load or store, store in the same address we have got, or load from it
- And there is a mux that choose from [PC + 1, LUI , loaded value from memory, result of ALU] all that depends on the signal that comes to mux (MemtoReg)

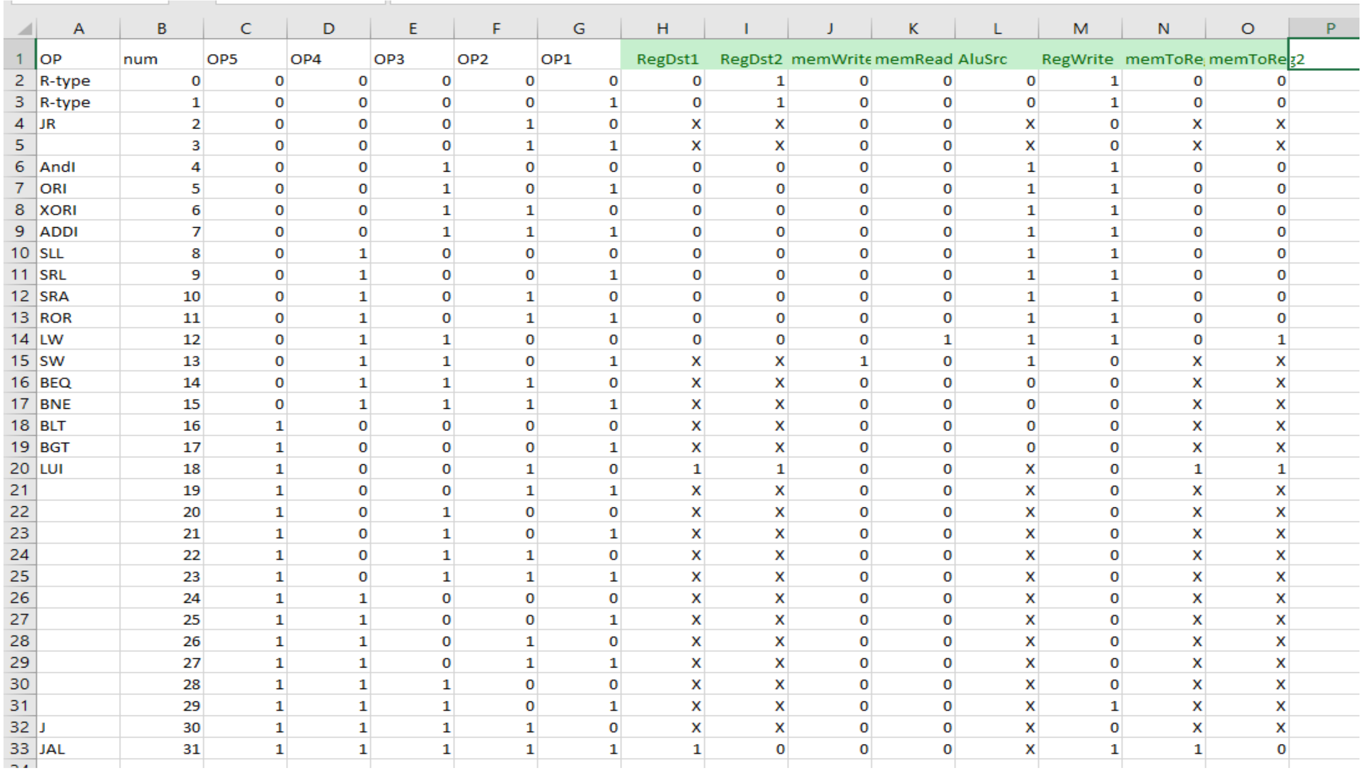


⌘ Control Unit:

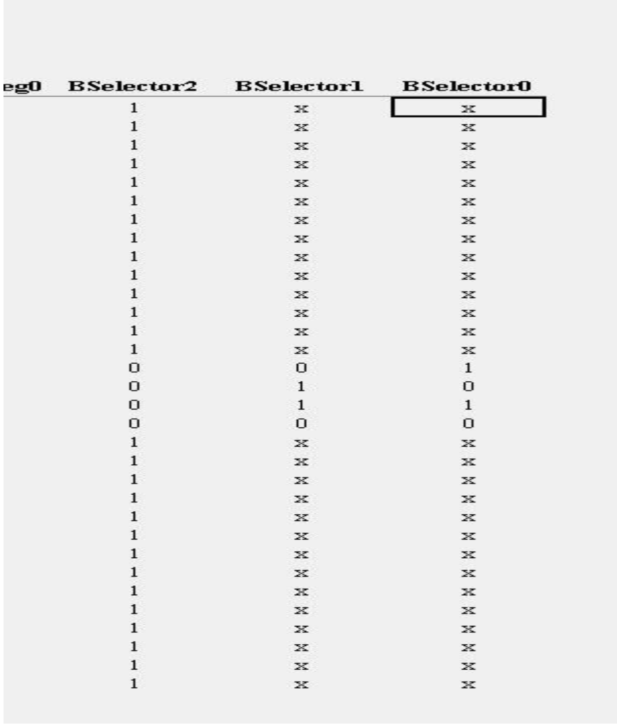
- The Control unit’s input is 5-bit for Opcode
- The outputs are the values for all the control signals in the data path.
- We did use 2 bits for RegDst and MemtoReg because both of them have more than 2 options to choose from



And here is the truth table of the circuit:-



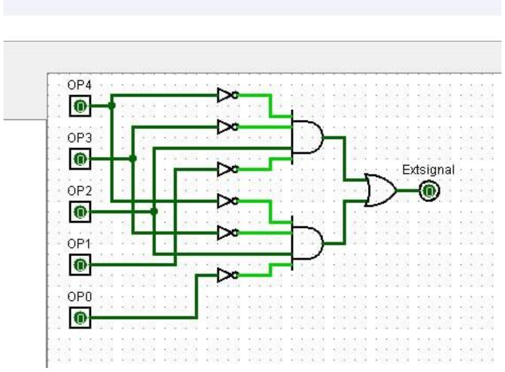
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	OP	num	OP5	OP4	OP3	OP2	OP1	RegDst1	RegDst2	memWrite	memRead	AluSrc	RegWrite	memToRe	memToReg2
2	R-type	0	0	0	0	0	0	0	1	0	0	0	1	0	0
3	R-type	1	0	0	0	0	1	0	1	0	0	0	1	0	0
4	JR	2	0	0	0	1	0	X	X	0	0	X	0	X	X
5		3	0	0	0	1	1	X	X	0	0	X	0	X	X
6	AndI	4	0	0	1	0	0	0	0	0	0	1	1	0	0
7	ORI	5	0	0	1	0	1	0	0	0	0	1	1	0	0
8	XORI	6	0	0	1	1	0	0	0	0	0	1	1	0	0
9	ADDI	7	0	0	1	1	1	0	0	0	0	1	1	0	0
10	SLL	8	0	1	0	0	0	0	0	0	0	1	1	0	0
11	SRL	9	0	1	0	0	1	0	0	0	0	1	1	0	0
12	SRA	10	0	1	0	1	0	0	0	0	0	1	1	0	0
13	ROR	11	0	1	0	1	1	0	0	0	0	1	1	0	0
14	LW	12	0	1	1	0	0	0	0	0	1	1	1	0	1
15	SW	13	0	1	1	0	1	X	X	1	0	1	0	X	X
16	BEQ	14	0	1	1	1	0	X	X	0	0	0	0	X	X
17	BNE	15	0	1	1	1	1	X	X	0	0	0	0	X	X
18	BLT	16	1	0	0	0	0	X	X	0	0	0	0	X	X
19	BGT	17	1	0	0	0	1	X	X	0	0	0	0	X	X
20	LUI	18	1	0	0	1	0	1	1	0	0	X	0	1	1
21		19	1	0	0	1	1	X	X	0	0	X	0	X	X
22		20	1	0	1	0	0	X	X	0	0	X	0	X	X
23		21	1	0	1	0	1	X	X	0	0	X	0	X	X
24		22	1	0	1	1	0	X	X	0	0	X	0	X	X
25		23	1	0	1	1	1	X	X	0	0	X	0	X	X
26		24	1	1	0	0	0	X	X	0	0	X	0	X	X
27		25	1	1	0	0	1	X	X	0	0	X	0	X	X
28		26	1	1	0	1	0	X	X	0	0	X	0	X	X
29		27	1	1	0	1	1	X	X	0	0	X	0	X	X
30		28	1	1	1	0	0	X	X	0	0	X	0	X	X
31		29	1	1	1	0	1	X	X	0	0	X	1	X	X
32	J	30	1	1	1	1	0	X	X	0	0	X	0	X	X
33	JAL	31	1	1	1	1	1	1	0	0	0	X	1	1	0



eg0	BSelector2	BSelector1	BSelector0
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	0	0	1
	0	1	0
	0	1	1
	0	0	0
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x

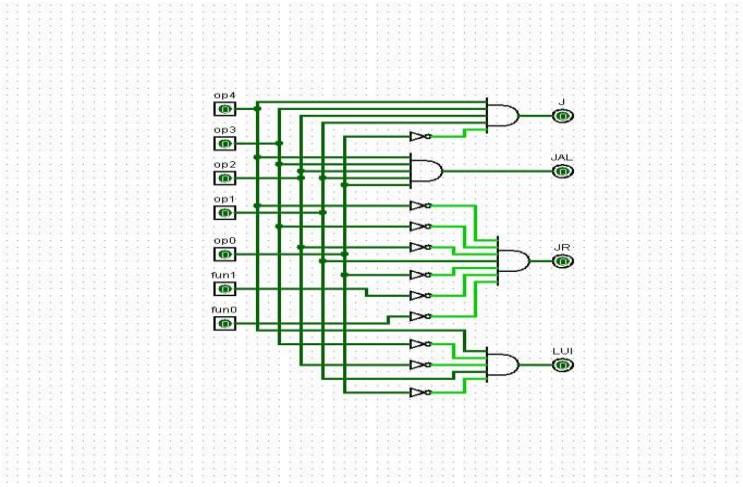
⌘ Extend Signal

this circuit is used to determine if we will make zero extend in case of input opcode is the opcode for andi or ori or xori and sign extend for other instructions



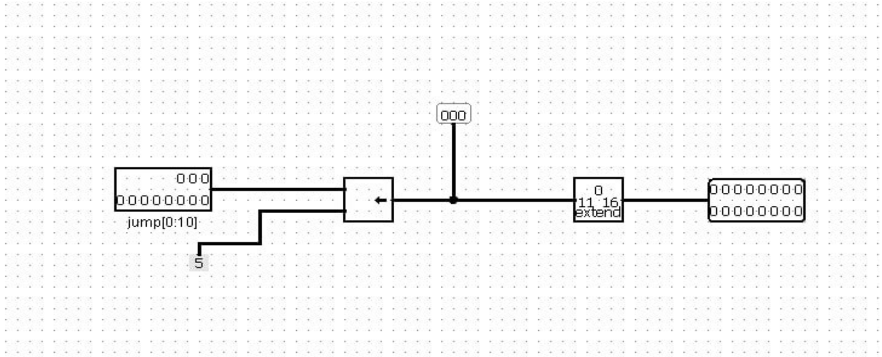
⌘ Signals

➢ the output of this circuit is one for j or jal or jr or lui based on input opcode and input function. we use this circuit to control in second mux in the PC.



⌘ LUI Class

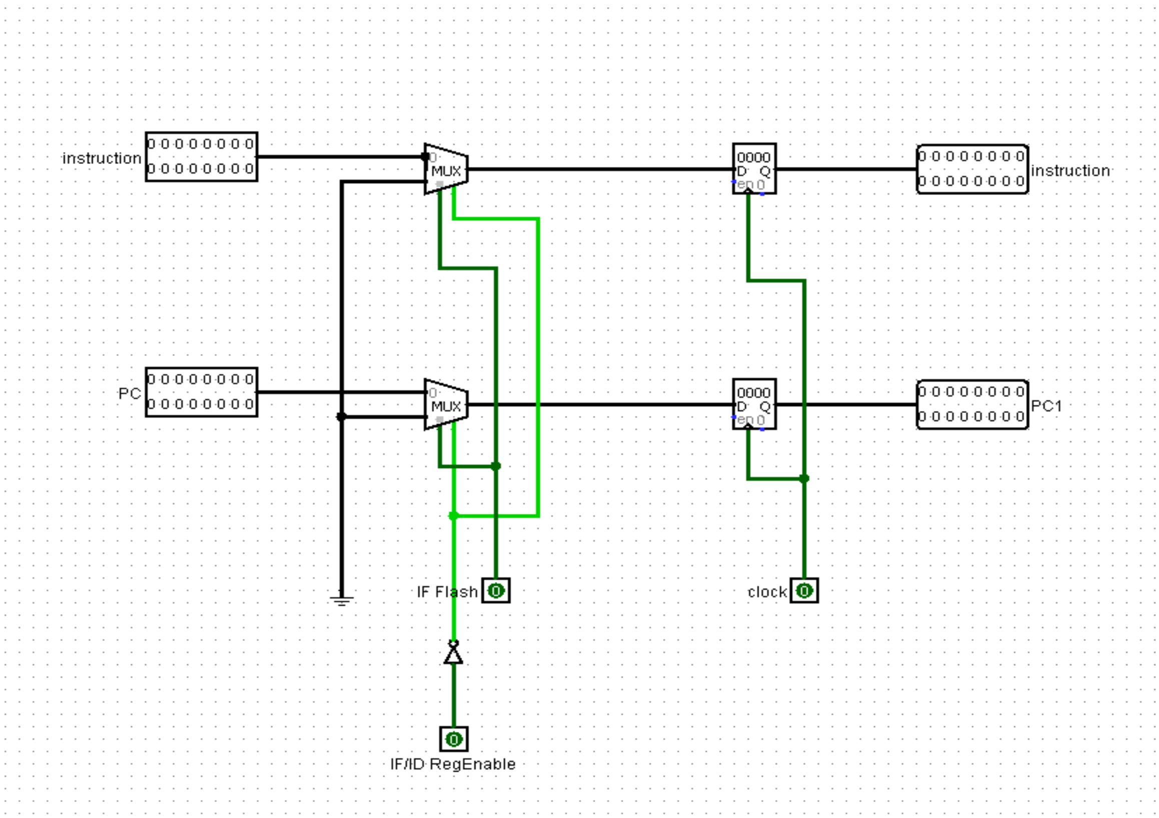
this circuit is used to execute LUI instruction by making 5 bits shift left for bits from 0 to 10.



⌘ Pipeline Stages:-

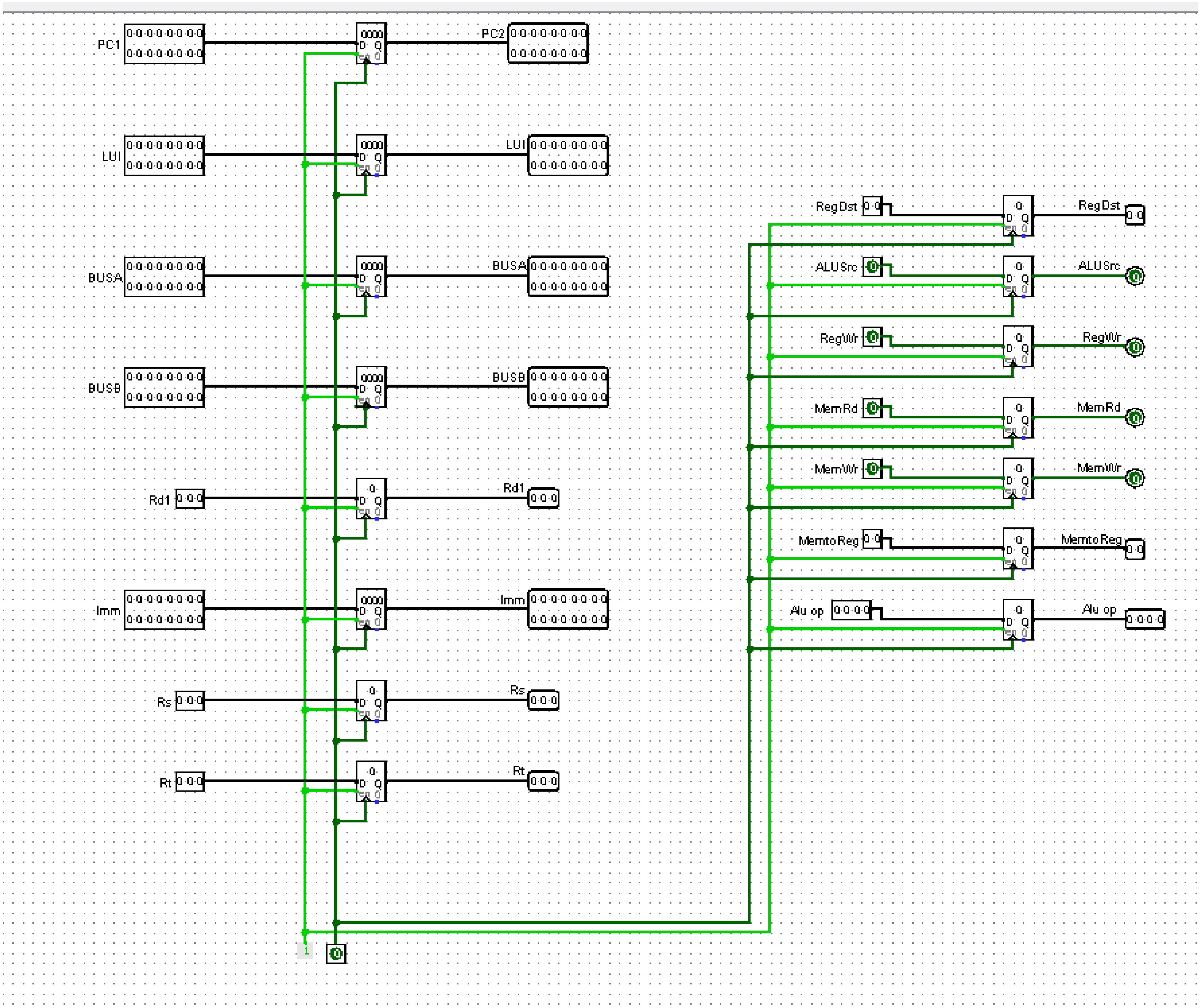
- Registers between (IF/ID – ID/EX – EX/MEM – MEM/WB)
- Registers Rd address and the next program counter are all saved in all registers we have just mentioned

○ IF/ID Register:



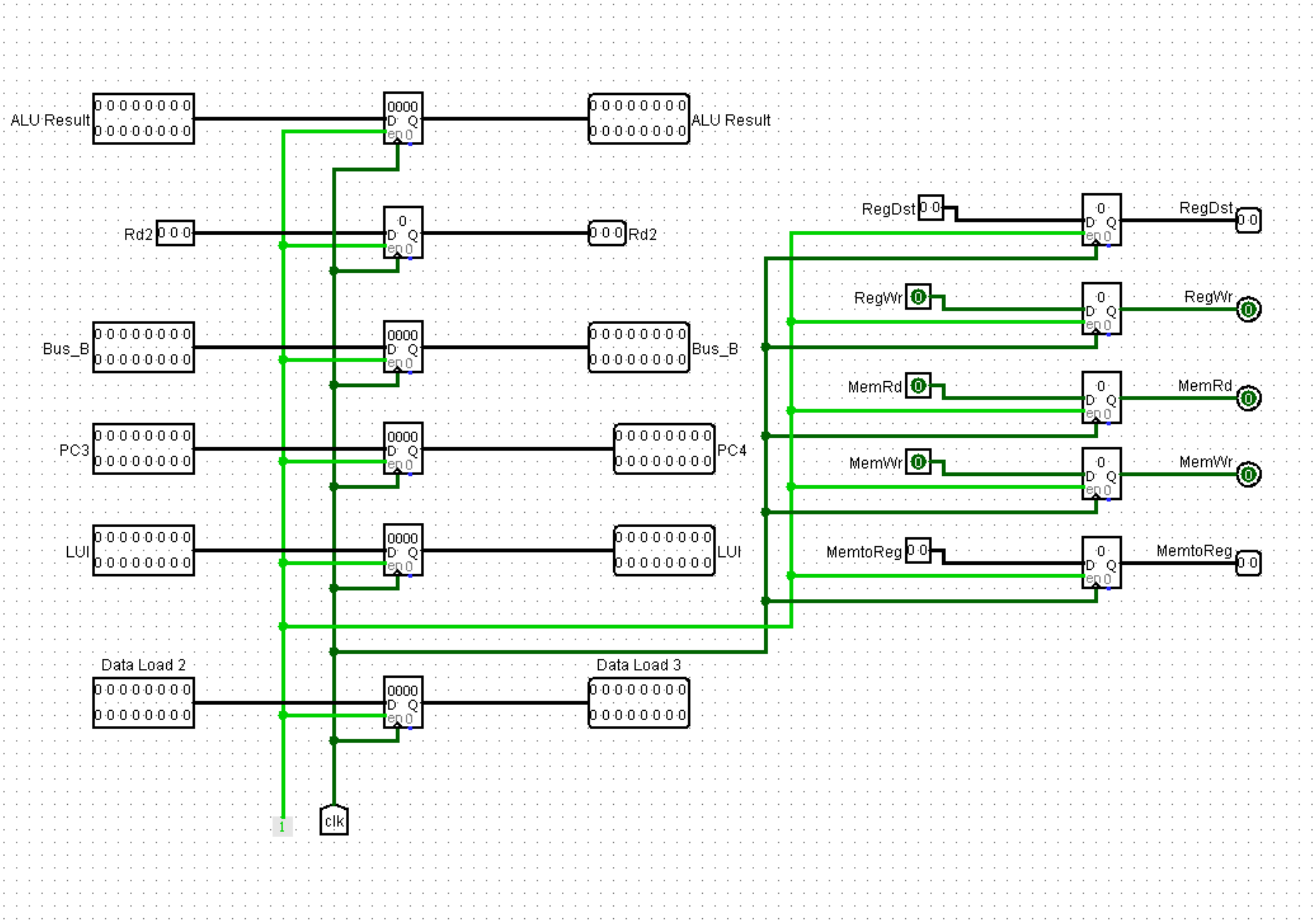
The first mux represents if the instruction is taken from the ROM or it will be killed because of the Flush signal, the other mux represents something similar because it will take the new program counter otherwise if there is a flush or stall (IF/ID regEnable is taken from the stall) it will be zero, and take nothing and signal IF/ID RegEnable will be 1 if there is a stall.

ID/EX Register:



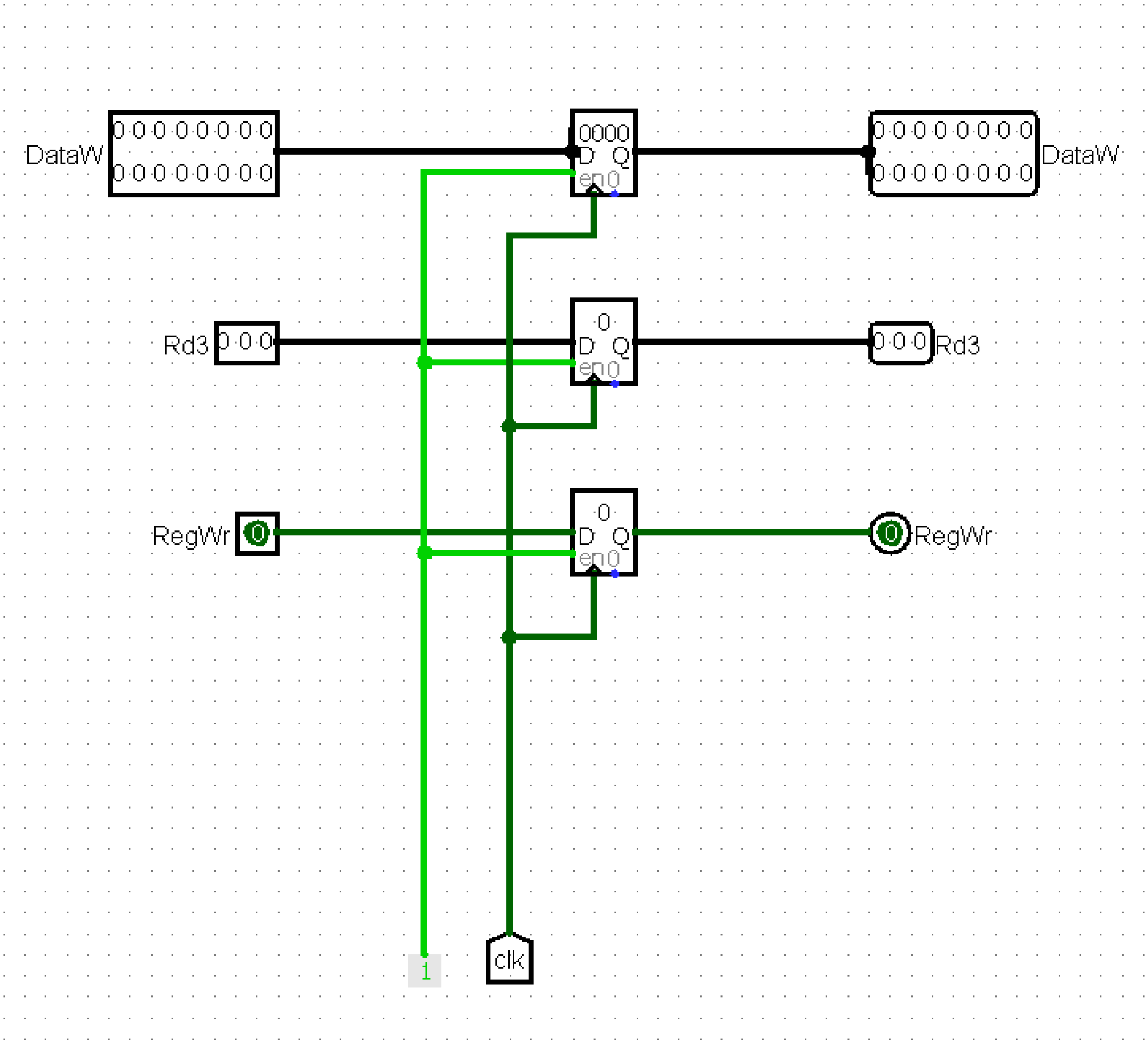
Storing the values of Bus A, Bus B, and immediate as long as they are needed in the next stages in addition to transferring needed signals for the next register and pc from the lasr register

- EX/MEM Register:



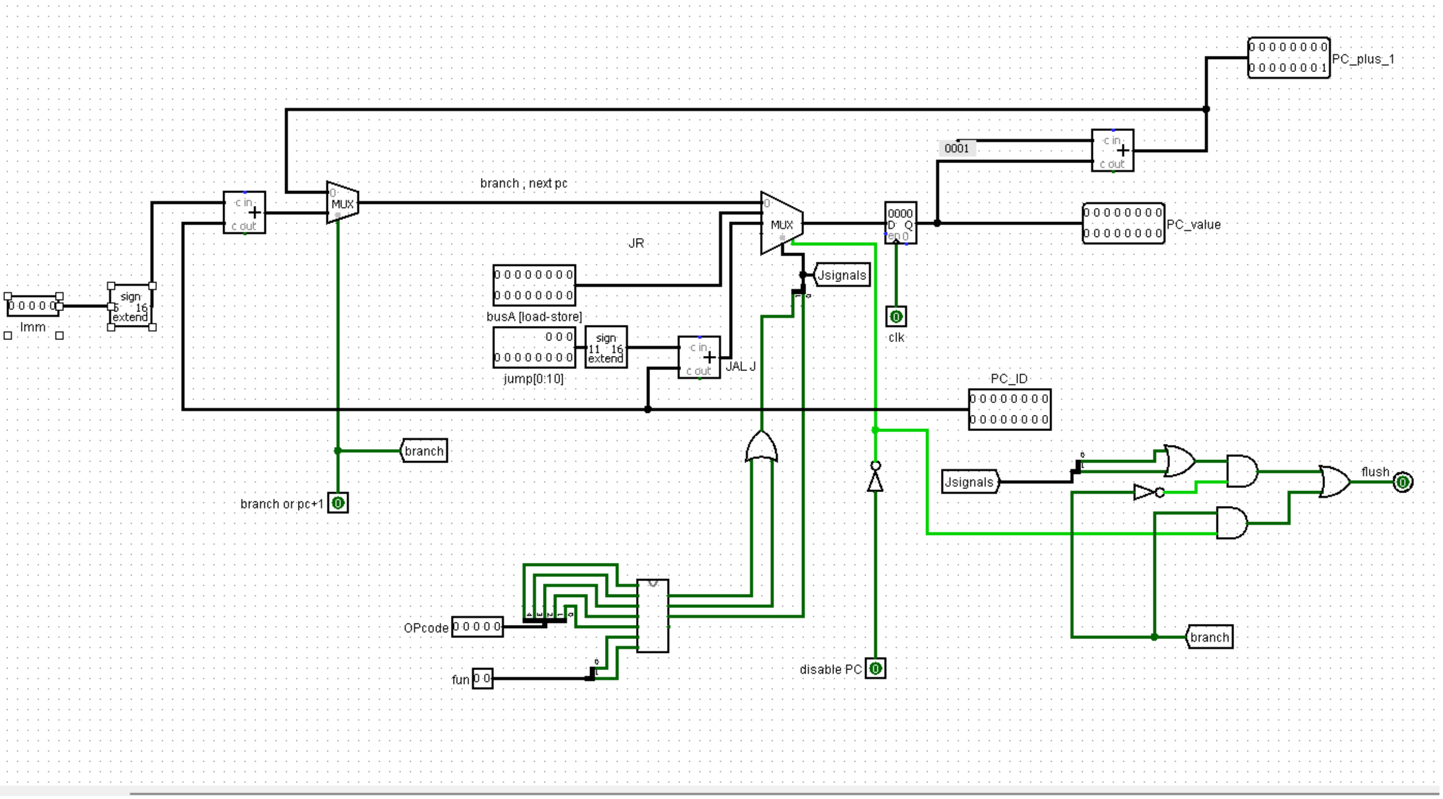
- Storing the ALU result because we need its value as an address to access RAM or saving it in MEM/WB register to write it back.
- Storing buses after delaying by 1 clock cycle one more time as we need Data load to be stored in RAM if the instruction is SW.
- In addition to transferring the needed data and signals like LUI , PC and the other unit signals [used in the next stages]

○ MEM/WB:



- ➢ Data is coming from the Multiplexer which chooses what will be the DataW [ALURes, Load from memory, PC+1 , LUI]
- ➢ As we mentioned before register Rd is saved in the whole journey of registers in case we have Rd as our destination to write back Data.

⌘ Modified PC:-



- ➢ Only a single modification but acts in different ways: the modification is we add an enable to choose if we active the mux or make a Stall because the signal called disable PC is coming from Stall block so if disable PC = 1 then Stall happen , neither the new PC nor flush is activated in this case
- ➢ Flush signal is a signal which is activated when we have a JUMP signal in addition we have not a Branch because we can’t do jump and branch at the same time or when we have jump signal

⌘ Forward unit 1 :-

The out of this circuit indicates which kind of forward will be choosen for registers Rs , Rt it's output is FORWARD A and FORWARD B each are two bits

This circuit is at ID/EX stage we first make sure RD EX/MEM(RD3) and RD MEM/WB(RD4) is not equal zero (not flash instruction) then:-

If ID/EX Rs = RD3 → FORWARD A= 01

If ID/EX Rs = RD4 → FORWARD A= 10

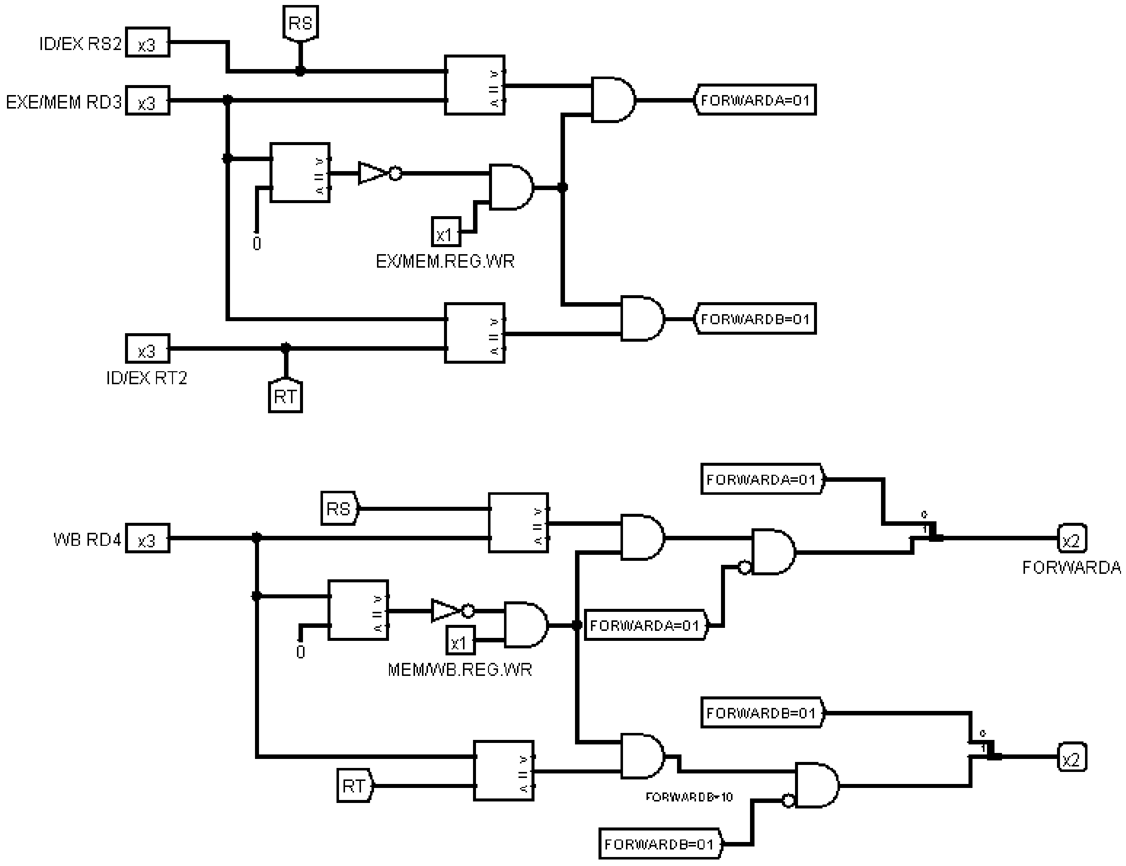
And we but an and gate to make sure these two conditions won't happen at the same time if happened it chooses FORWARD A= 01 because the last update will be at RD3

And same for RT :-

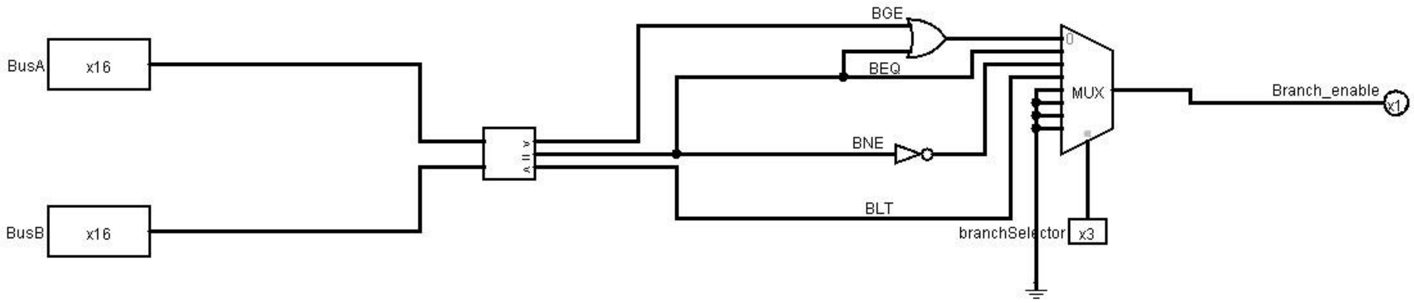
If ID/EX Rt = RD3 → FORWARD B= 01

If ID/EX Rt = RD4 → FORWARD B= 10

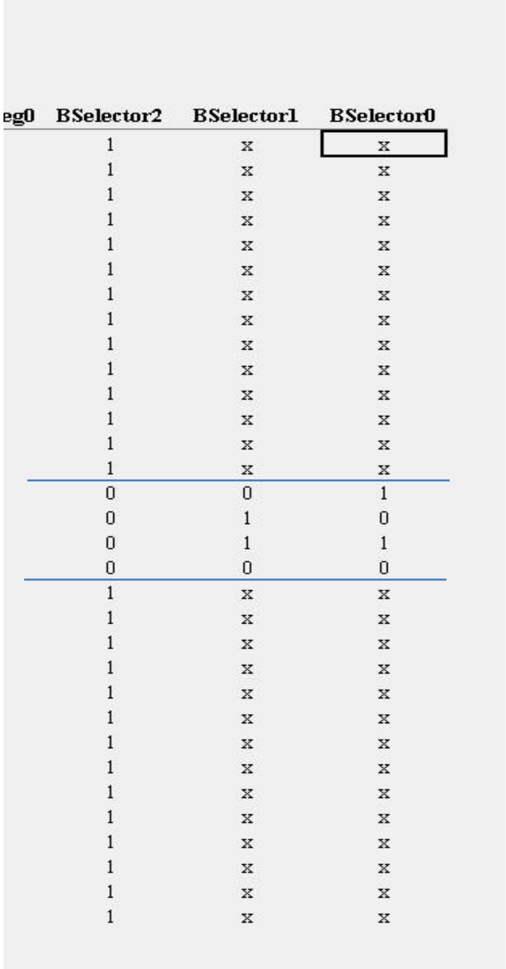
And if there is no forward for RS then FORWARD A=00 and if no forward for RT FORWARD B=00



⌘ Branch unit :-



This circuit to enable a one bit called “Branch_enable” when the instruction is one of the four kinds of branches (BEQ,BNE,BLT,BGE) we used a mux with a selector 3 bits called “branchSelector” and this is it’s truth table using the op code



eg0	BSelector2	BSelector1	BSelector0
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	0	0	1
	0	1	0
	0	1	1
	0	0	0
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x
	1	x	x

If the op code is 14 to 17 bit 2(BSelector2) is zero and for any another op code it is 1 and the other two bits don’t care and we put there pins ground at the mux

then the first four pins are four are for the branches:

branchSelector=000 →BGE

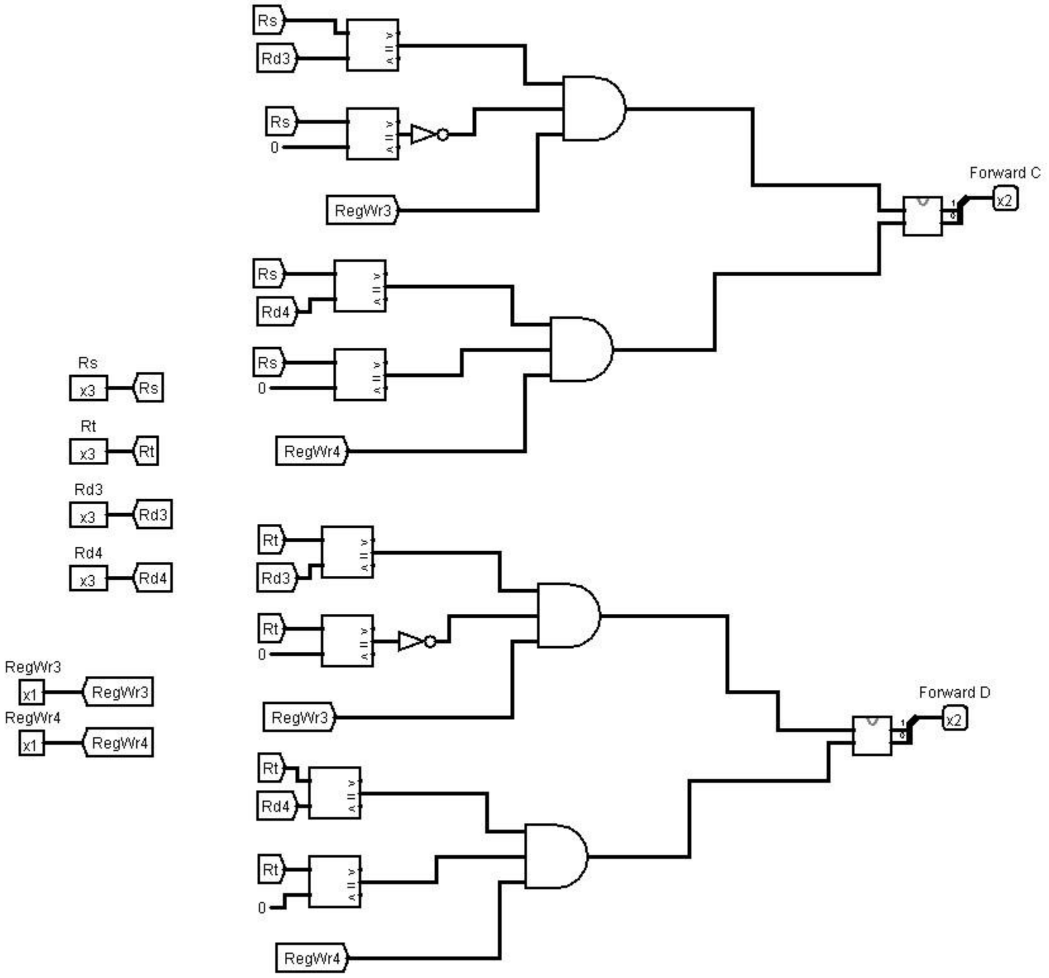
branchSelector=001 →BEQ

branchSelector=010 →BNE

branchSelector=011 →BLT

⌘ Forward unit2 :-

The out of this circuit indicates which kind of forward will be choosen for registers Rs , Rt at branch instructions and it is at IF/ID stage it's output is FORWARD C and FORWARD D each are two bits first make sure RD EX/MEM(RD3) and RD MEM/WB(RD4) is not equal zero (not flash instruction) then:-



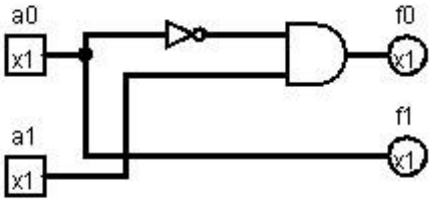
If IF/ID Rs = RD3 → FORWARD C= 01

If IF/ID Rs = RD4 → FORWARD C= 10

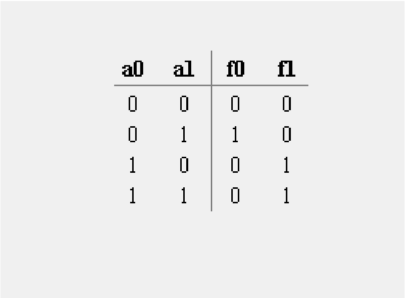
And same for RT :-

If IF/ID Rt = RD3 → FORWARD D= 01

If IF/ID Rt = RD4 → FORWARD D= 10



This circuit to make sure these two conditions won’t happen at the same time if happened it chooses FORWARD A= 01 because the last update will be at RD3 and here is it’s truth table



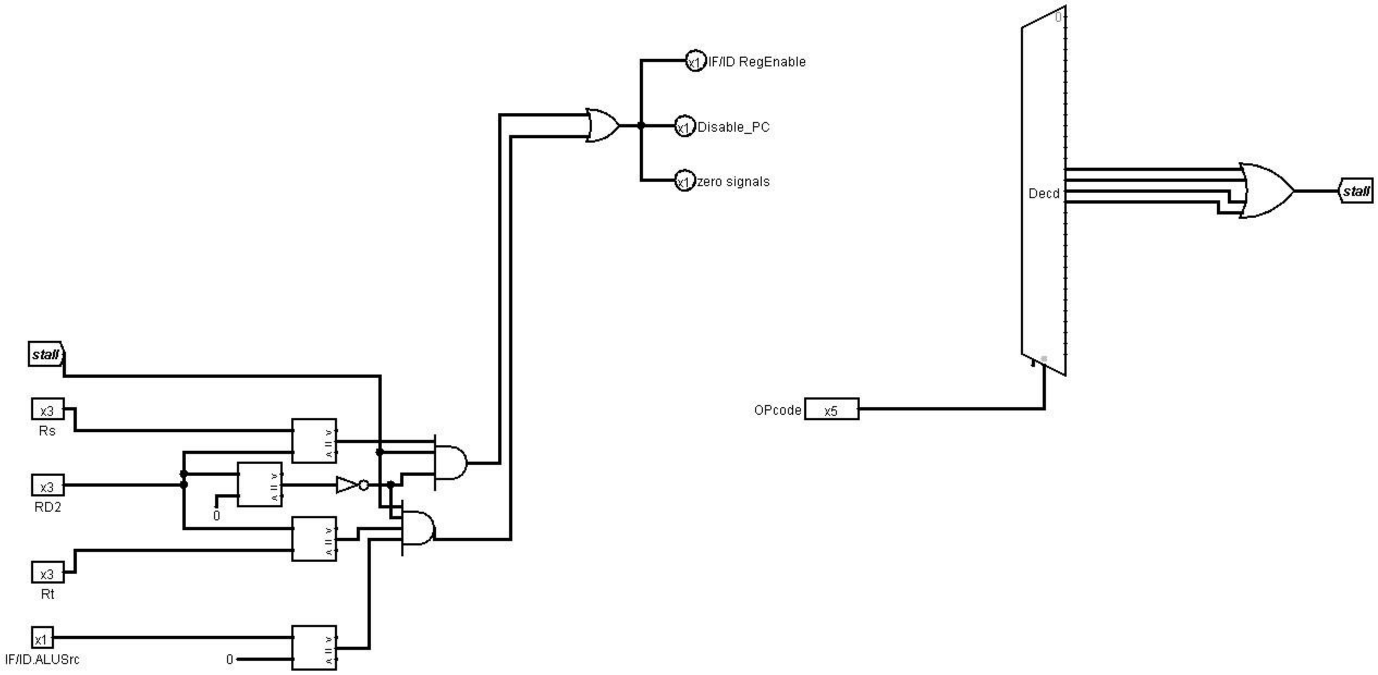
a0	a1	f0	f1
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	1

And if there is no forward for RS then FORWARD A=00 and if no forward for RT FORWARD B=00

⌘ Stall unit :-

This unit is at the IF/ID stage and is used to delay the processor for one clock If the branch instruction depends on the register RD of the last instruction cycle by disable the IF/ID register and the pc counter unit and making the control signals all zeros

We check is if IF/ID RS or IF/ID Rt equals ID/EX RD (RD2) which means that the branch instruction depends on register RD of the instruction before it so we stall the processor for on the instruction right before it and we check if RD2 not equal zero (not a flash instruction) and we check that it is an R-TYPE instruction with signal ALUsrc=0

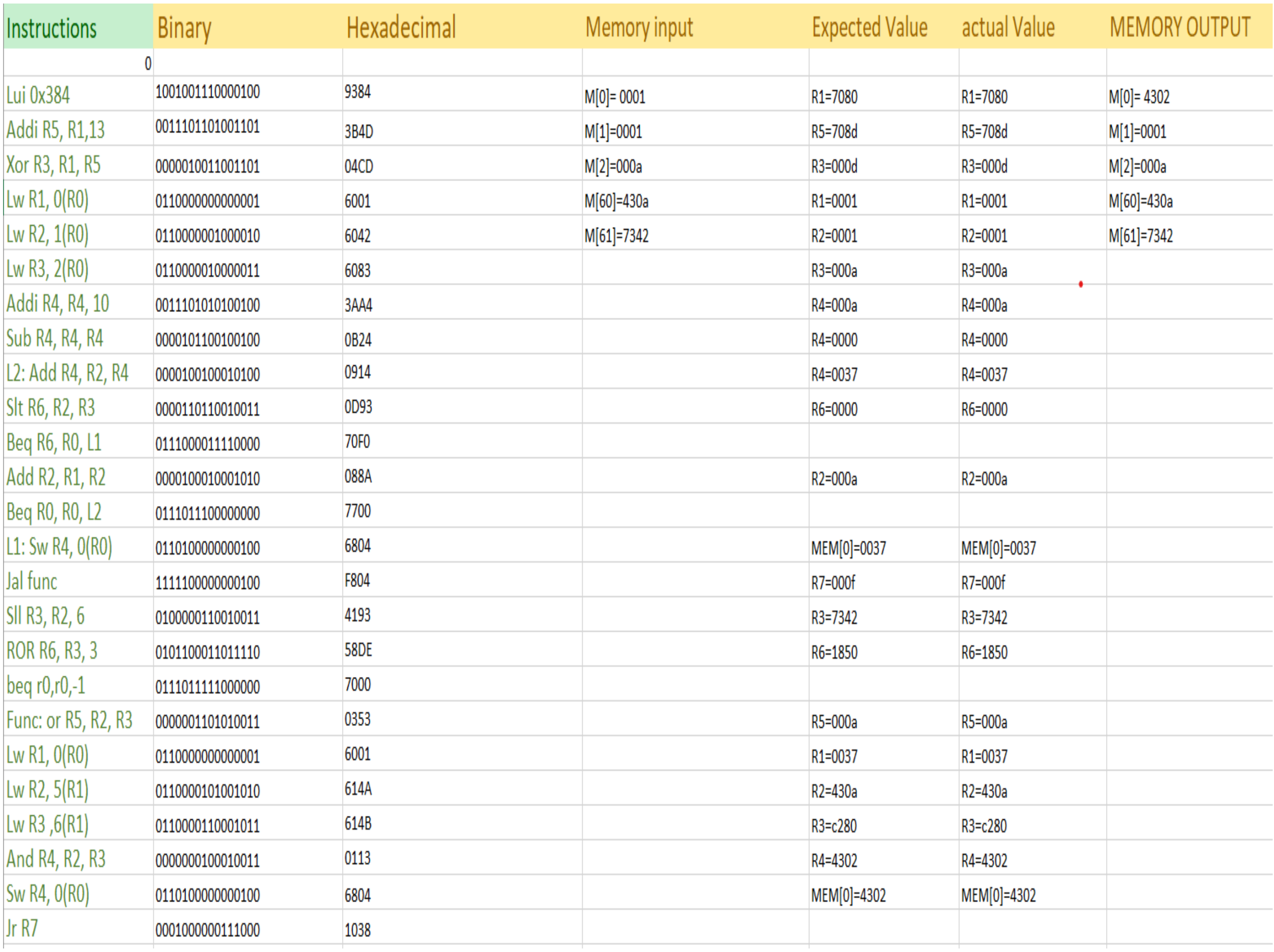


⌘ Participation in parts of the Project:

[According to Classes]

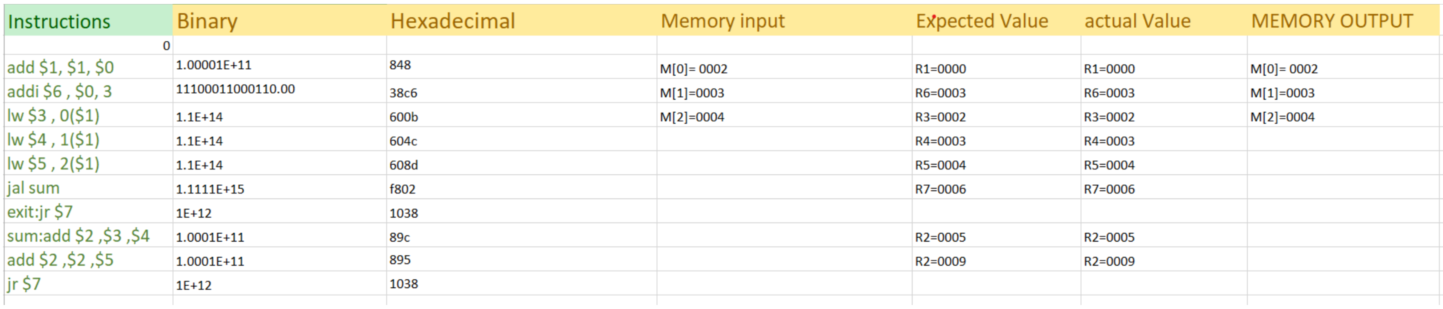
Tasneem Eid	Shahd Mahmoud	Maged Aziz
Control Unit	Alu	Program Counter [PC]
Signals	Alu control	Extend Signal
	LUI	Register File
Test array	Test1	Test 2
Main circuit	Main circuit	Main circuit
Forward unit1	Forward unit2	Branch unit
EX/MEM Register	IF/ID Register	MEM/WB Register
Stall unit circuit	Stall unit circuit	ID/EX Register

The test :-



Instructions	Binary	Hexadecimal	Memory input	Expected Value	actual Value	MEMORY OUTPUT
0						
Lui 0x384	1001001110000100	9384	M[0]= 0001	R1=7080	R1=7080	M[0]= 4302
Addi R5, R1,13	0011101101001101	3B4D	M[1]=0001	R5=708d	R5=708d	M[1]=0001
Xor R3, R1, R5	0000010011001101	04CD	M[2]=000a	R3=000d	R3=000d	M[2]=000a
Lw R1, 0(R0)	0110000000000001	6001	M[60]=430a	R1=0001	R1=0001	M[60]=430a
Lw R2, 1(R0)	0110000001000010	6042	M[61]=7342	R2=0001	R2=0001	M[61]=7342
Lw R3, 2(R0)	0110000010000011	6083		R3=000a	R3=000a	
Addi R4, R4, 10	0011101010100100	3AA4		R4=000a	R4=000a	
Sub R4, R4, R4	0000101100100100	0B24		R4=0000	R4=0000	
L2: Add R4, R2, R4	0000100100010100	0914		R4=0037	R4=0037	
Slt R6, R2, R3	0000110110010011	0D93		R6=0000	R6=0000	
Beq R6, R0, L1	0111000011110000	70F0				
Add R2, R1, R2	0000100010001010	088A		R2=000a	R2=000a	
Beq R0, R0, L2	0111011100000000	7700				
L1: Sw R4, 0(R0)	0110100000000100	6804		MEM[0]=0037	MEM[0]=0037	
Jal func	1111100000000100	F804		R7=000f	R7=000f	
Sll R3, R2, 6	0100000110010011	4193		R3=7342	R3=7342	
ROR R6, R3, 3	0101100011011110	58DE		R6=1850	R6=1850	
beq r0,r0,-1	0111011111000000	7000				
Func: or R5, R2, R3	0000001101010011	0353		R5=000a	R5=000a	
Lw R1, 0(R0)	0110000000000001	6001		R1=0037	R1=0037	
Lw R2, 5(R1)	0110000101001010	614A		R2=430a	R2=430a	
Lw R3 ,6(R1)	0110000110001011	614B		R3=c280	R3=c280	
And R4, R2, R3	0000000100010011	0113		R4=4302	R4=4302	
Sw R4, 0(R0)	0110100000000100	6804		MEM[0]=4302	MEM[0]=4302	
Jr R7	0001000000111000	1038				

Report test :-



Instructions	Binary	Hexadecimal	Memory input	Expected Value	actual Value	MEMORY OUTPUT
0						
add $1, $1, $0	1.00001E+11	848	M[0]= 0002	R1=0000	R1=0000	M[0]= 0002
addi $6 , $0, 3	11100011000110.00	38c6	M[1]=0003	R6=0003	R6=0003	M[1]=0003
lw $3 , 0($1)	1.1E+14	600b	M[2]=0004	R3=0002	R3=0002	M[2]=0004
lw $4 , 1($1)	1.1E+14	604c		R4=0003	R4=0003	
lw $5 , 2($1)	1.1E+14	608d		R5=0004	R5=0004	
jal sum	1.1111E+15	f802		R7=0006	R7=0006	
exit:jr $7	1E+12	1038				
sum:add $2 ,$3 ,$4	1.0001E+11	89c		R2=0005	R2=0005	
add $2 ,$2 ,$5	1.0001E+11	895		R2=0009	R2=0009	
jr $7	1E+12	1038				