2024

RISC Processor

Computer Architecture

Dr. Gihan Naguib
Eng.Jihad Awad

PART one

SINGLE CYCLE Processor

❖Group Members

- Belal Soliman Abd Elhalem
- Osama Ahmed Mahmoud
- Mohamed Ahmad Saleh

❖Content of Report

❖ Introduction

- In this project, we will design a simple 16-bit RISC processor with seven 16- bit general purpose registers: R1 through R7. R0 is hardwired to zero and cannot be written, we are left with seven registers. There is also one special-purpose 16-bit register, which is the program counter (PC). All instructions are only 16 bits. There are three instruction formats, R-type, I-type, and J-type

- The program counter PC is a special-purpose 16-bit register. That can address at most 2^16 instructions. All instructions are only 16 bits. There are three instruction formats, R-type, I-type, and J-type as shown below:

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 format

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

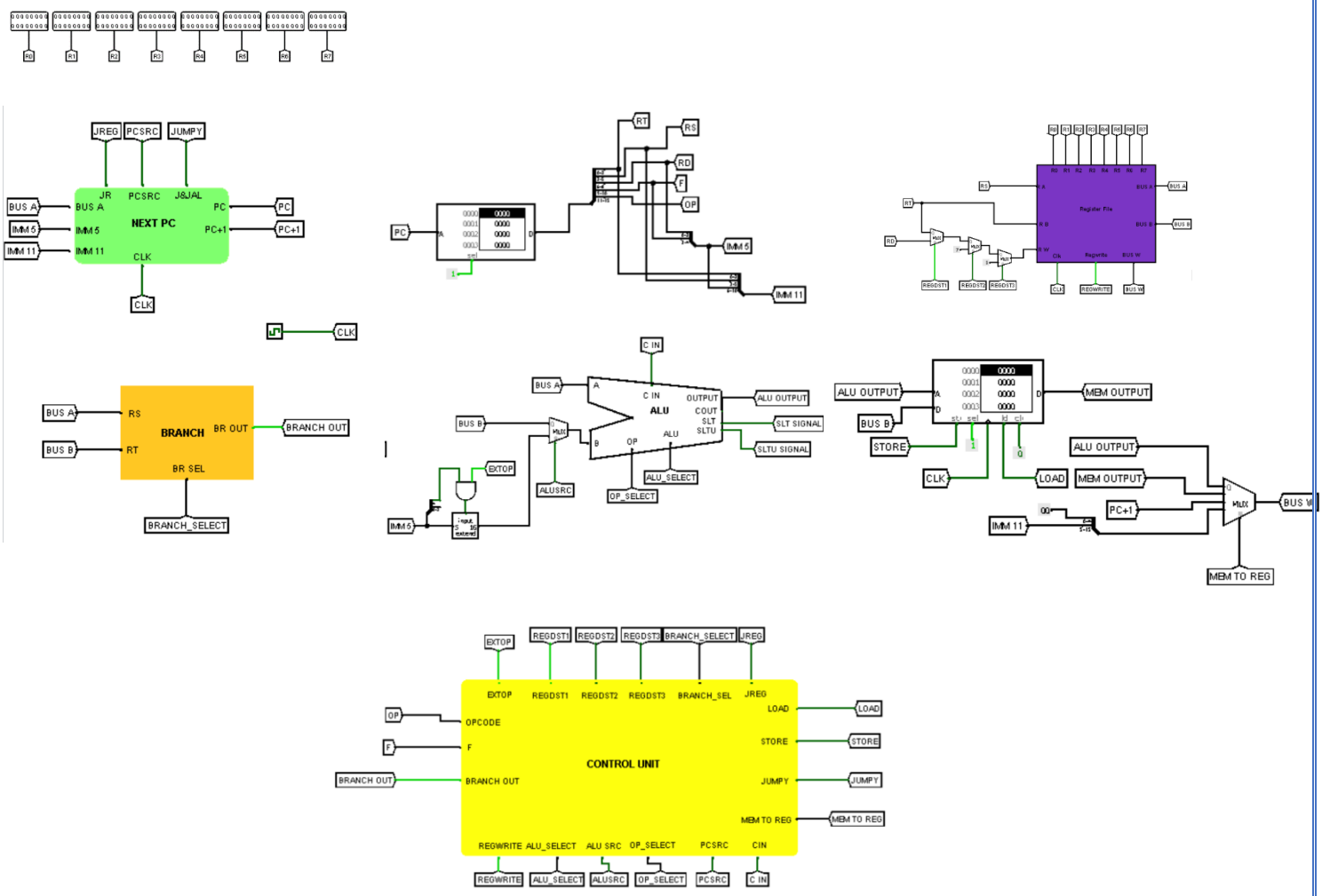
Op^5	Imm^5	Rs^3	Rt^3

J-type format :

5-bit opcode (Op) and11-bit Immediate

Op^5	Imm^{11}

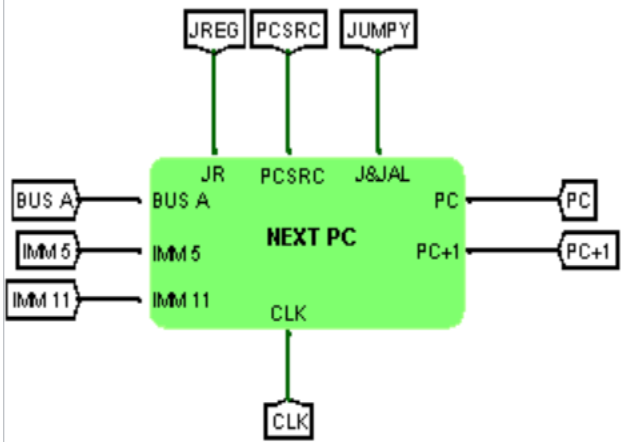
❖ Our design of processor

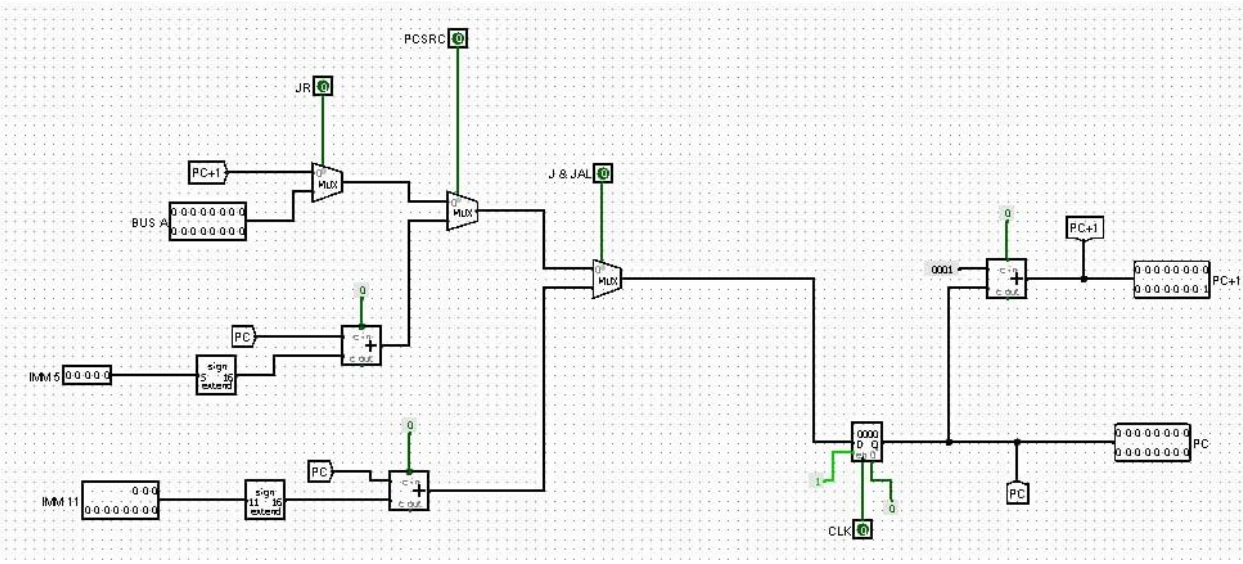


We will discuss the details of each part separately

❖ Next pc

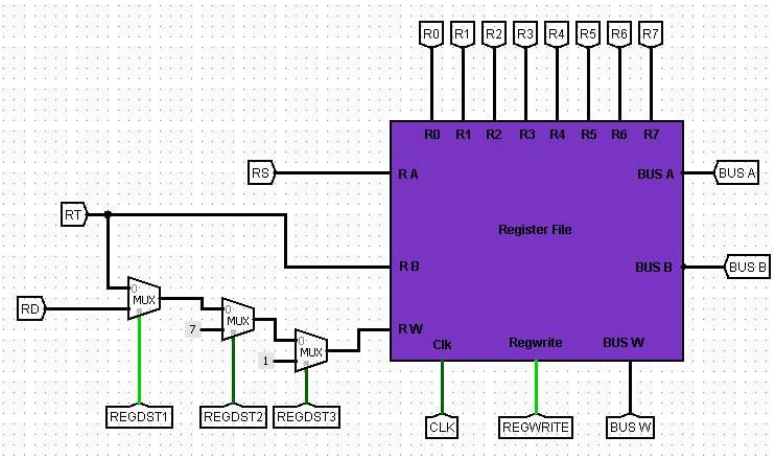
- the next pc :
  the circuit shown gives two outputs pc and pc+1 it has 4 control signals jr , pcsrc , j&jal snf clock and has input of bus A , imm5 and imm11



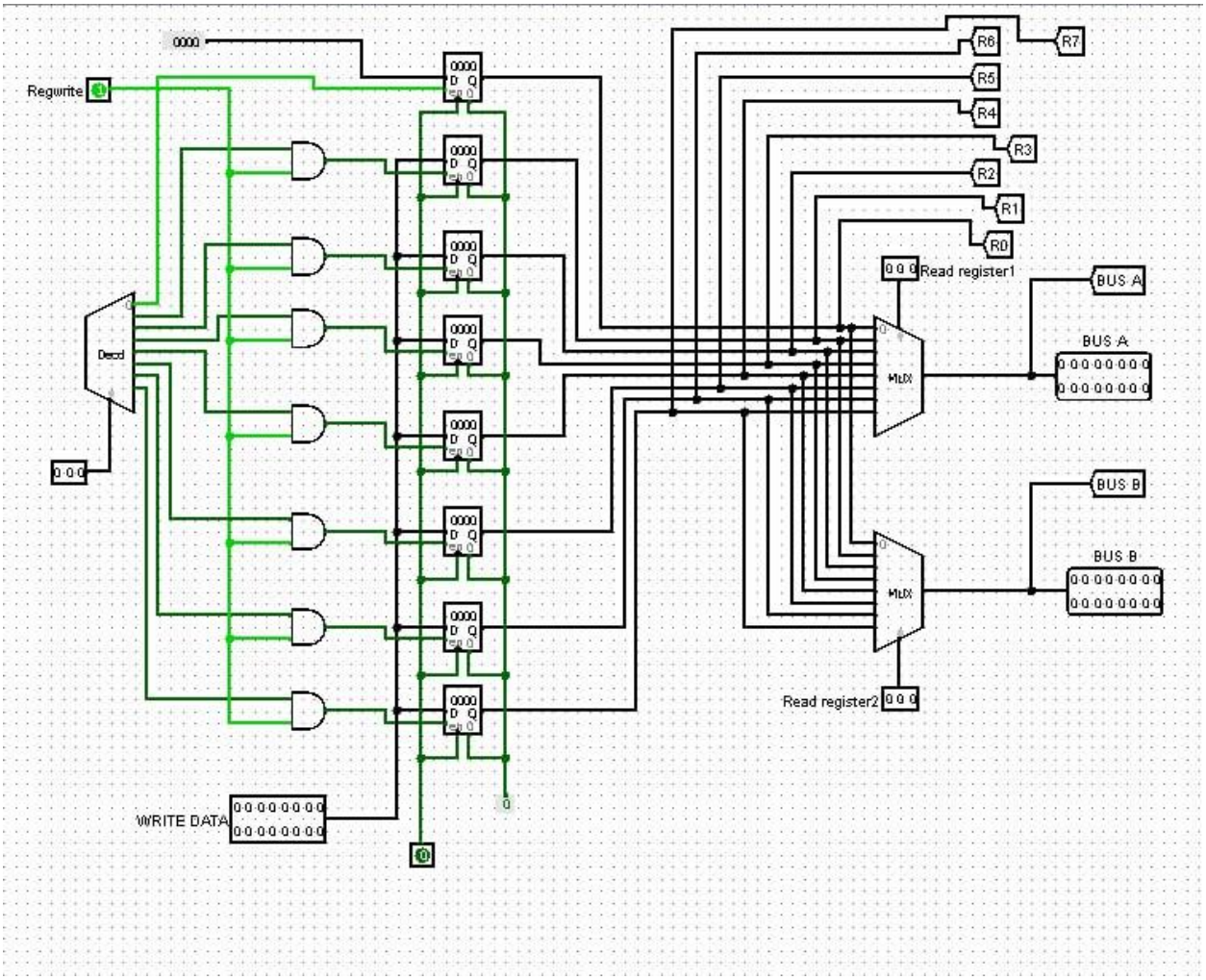


- an inner view of the next pc : if we looked at the top left of the page we'll find a mux with selector jump register to jump to the content of the register which come in the bus A and the address of the next instruction to be executed pc +1 the next mux is to select between the first mux and the address of the branch which is then for the next mux we choose between the first two muxes and the address of the jump which is , PC= PC + Signed(Imm11) then we save the address in the register lastly we get our outputs pc+1 by an adder and pc

❖ Register File

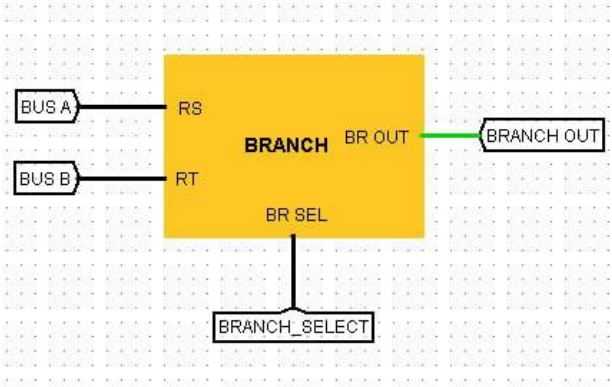


- Register File:
  A Register file containing eight 16-bit registers R0 to R7 With two read ports and one write port R0 is a register that can't be read nor written (hardwired to zero). The register file has two sources, one destination input ,Data Write Two output buses for data output from the register file, we also have one signal (REG write) that enables us to write in the register file, theclock signal & the clear signal which must be zero. In the next part, we will speak in detail



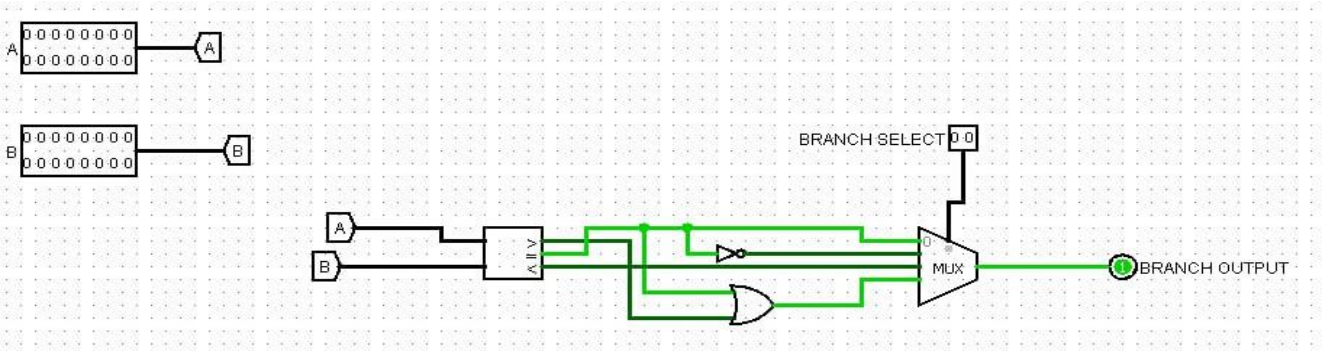
- We use decoder that select which register will be enabled and its selector is meanly the RD, we use eight and gates for input enable for all registers the input for ands is the output of decoder and REG write, we using two mux to choose which register will path through the buses A and B, the write data is input the data will be stored in RD the register and we put input zero at clear

❖ Branch Circuit



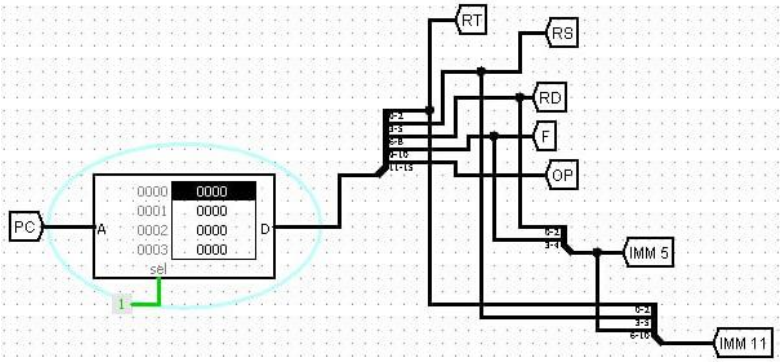
- for the branch circuit we use a 2'complement comparator to compare between two inputs

BUS A that represented RS and BUS B that represented Rt



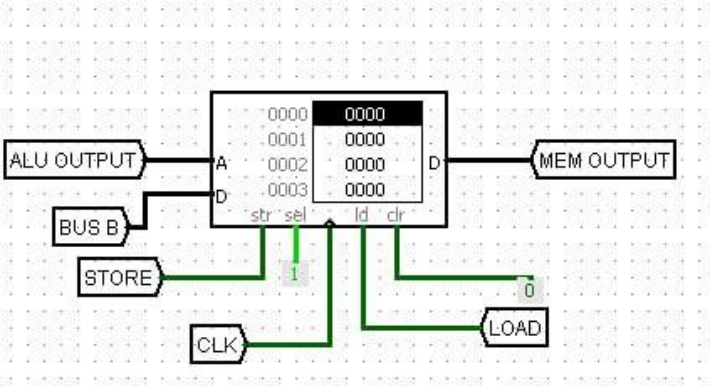
- we use a 4*1 MUX to select between the 4 branches possibilities BEQ , BNE , BLT and BGE . the output of the MUX will be used to specify if any of the branch conditions will be taken or not as we will see later at control unit .

❖ The Instruction Memory



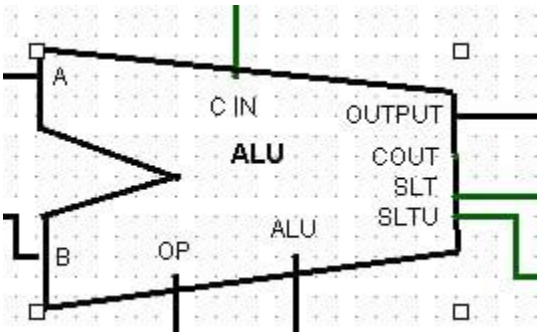
- The instruction memory : A rom to fetch our instructions to some outputs : Rt that holds the first 3 bits , Rs holds the next three bits , Rd holds the next 3 bits , op holds the next two bits , imm5 holds 5 bits which is composed of first three bits and bits numbered by 3 & 4 and lastly imm11 which is 11 bits composed of bits numbered by 0:2 , 3:5 and 6:10 and an input PC to fetch our instructions

❖ The Data Memory



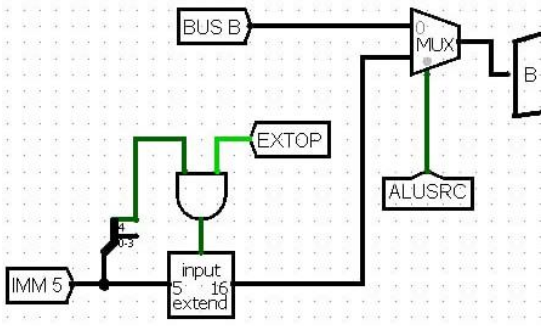
- The data memory : a RAM memory to store to and load from it our data which has two inputs the alu output which is used for calculating the address and BUS B to store some data into the data memory and some control signals store for writing in the memory and load for reading from the memory and the clock and lastly the output of the memory

❖ ALU (Arithmetic & Logic Unit)

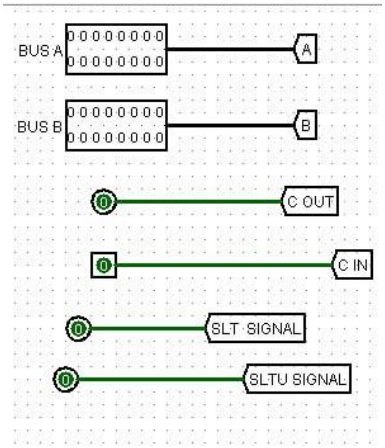


- An arithmetic-logic unit (ALU) is part of a computer processor (CPU) that carries out arithmetic and logic operations on the operands in computer instruction words.
- ALU has two inputs (A and B) in addition to one output (the result of the ALU operation).
- We have set less than signal , output and sltu and also we've three signals that come from the control unit op and alu and cin which we'll delve into on the next slides

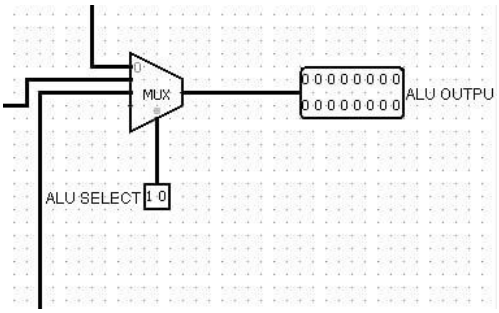
- Bus B includes the following image which is the output of the register file bus b muxed with the output of sign or zero extended imm 5 which has been controlled by a signal that is formed by a control signal extop with the sign bit (the last bit) anded with each other



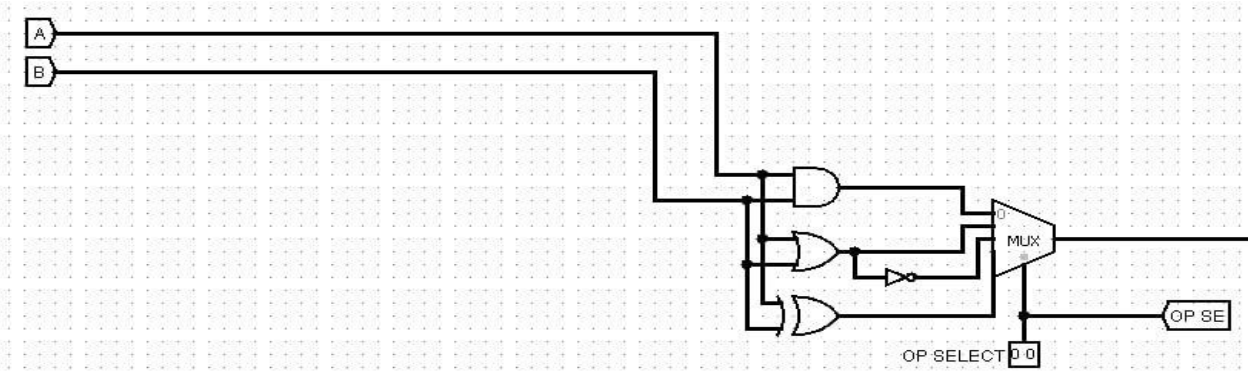
- now let's break down the alu part by part the first part is the inputs and the outputs of the alu which are bus a which we get it from the read data bus of register file and the other bus is b which we have discussed in the last slide and the other signal is output such as cout slt and sltu and a control signal c in .



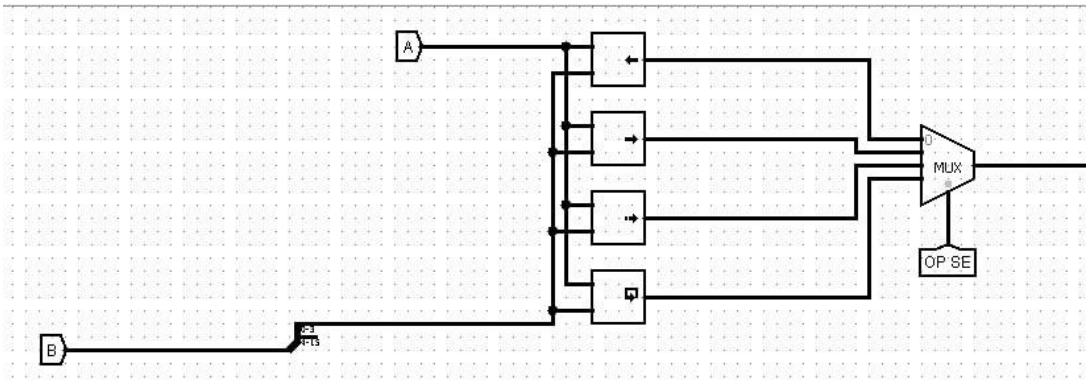
- we would start from the end to discuss each input of the mux , we can see the mux 4 input but won't get the fourth one, for 00 there's a shift operations for 01 we've some operations which are addition , subtraction , slt and sltu signal for 10 we've the logic operations



Logic operation: the logic operations which it's described by four operations (AND , OR , NOR and XOR) and we can select with op select signal coming from the control signal

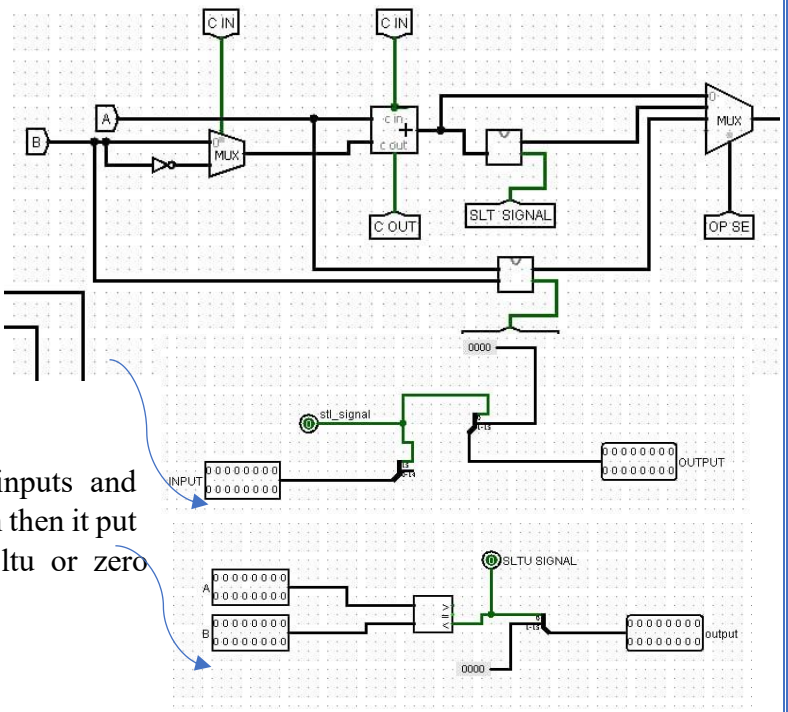


Shifting operations: we shift with bits of 0:3 of bus b and we shift bus A the operation we'll do is 00 shift left logical, 01 shift right logical, 10 shift right arithmetic and the last one is rotating right and we would select with signal coming from control signal op se



Arithmetic operations :

the some operations which are : addition or subtraction depending on the control signal cin or slt or sltu which will be discussed in the following.



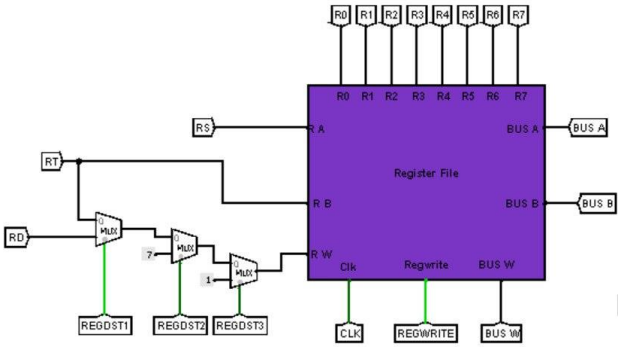
circuit (1) for slt : its input is the output of subtraction of the two inputs we take the last bit to be slt signal and then we put zeros to the other bits to get 1

circuit (2) is sltu this circuit take the two inputs and compare them and take the output on the less bin then it put the other bits zeros to set to 1 in case of sltu or zero otherwise

❖ Data Path

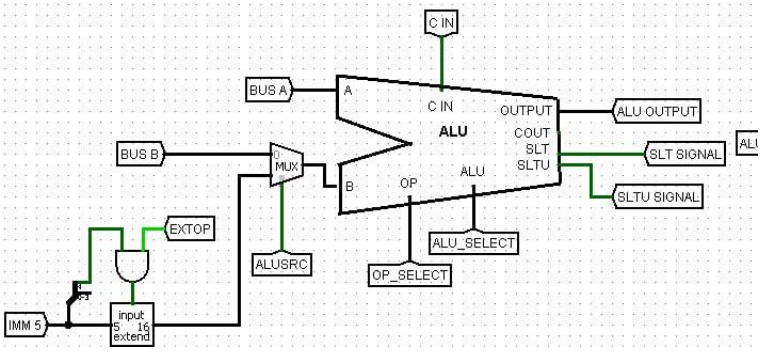
1) Register file:

the In this part, we will speak about the data path and how to connect all of the previous parts to make one block which will do our instruction, Let's start with the register file.
In the corresponding figure we denote :



- that the first source is (RS) and the second source is (RT).
- The distention can be (RD or Rt or R7 or R1)so we use three MUXs to select which address we will use and we have three control signals to do that, this signals are (REGDST1), (REGDST2) and (REGDST3).
- The output buses have the value of the registers and go to the ALU to do the operations.
- We also have a control signal (REGwrite) which enable to write in the register file.

2) ALU data path:



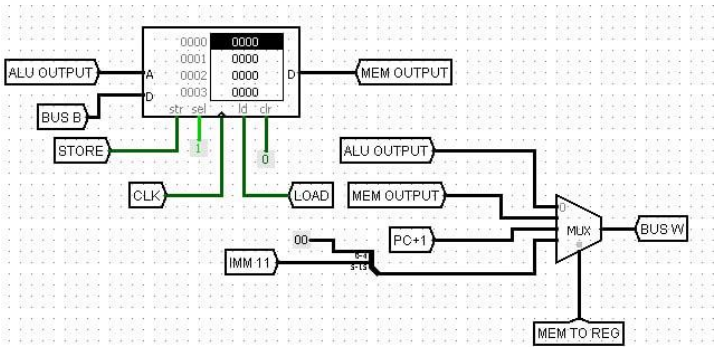
- the ALU has two inputs that come from the register file, the first bus A always come from BUS A from the register file which has the date of RT address.
- BUS B can be the data of the second bus from the register file or the immediate value (which content from the function and the RT that comes from the instruction) so we use a MUX to select one of them.
- 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 (extop).
- We also have five-bit control signal from the control unit we explained

before and we also have two output signals (alu output , slt and sltu).

- Finally, the output of the ALU (we will know where it is used in the next parts).

3) Data Memory:

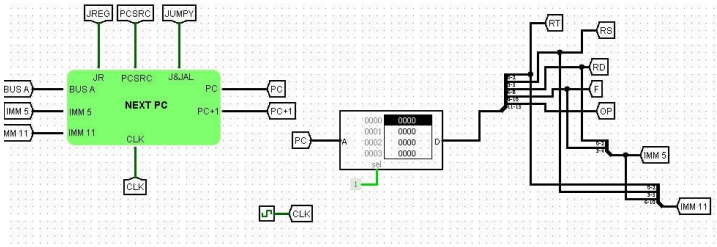
- The date that we will store in the memory is the content of the register RD which comes on the BUS B from the register file and as shown in the corresponding Figure:



- The date will store in the register file can be one of four types as we explained in the previous part (Data Memory) so we use a MUX to select one of them the selector comes from the shown control unit as we explained in the previous part

4) Next PC & Instruction Memory:

- As we see in the part of the next pc we need the data from RD registerwhich we got in BUS A.



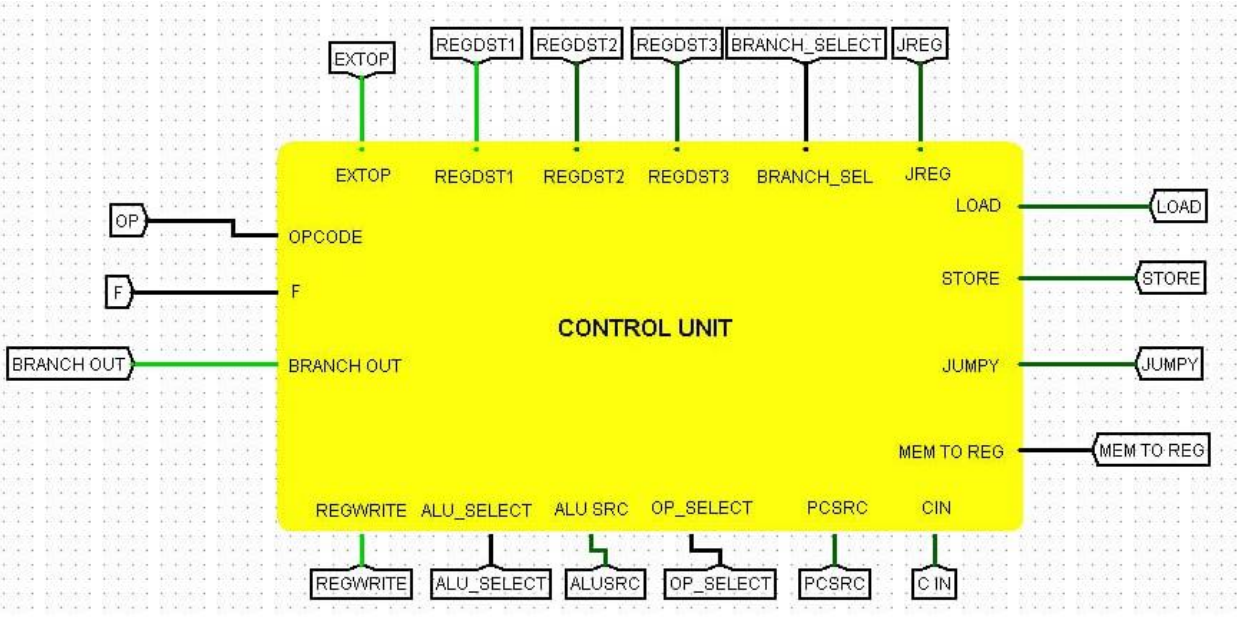
- We also need the immediate value (Denote it is for the jump instruction).

- The control signals that come from the control unit to select the next instruction.

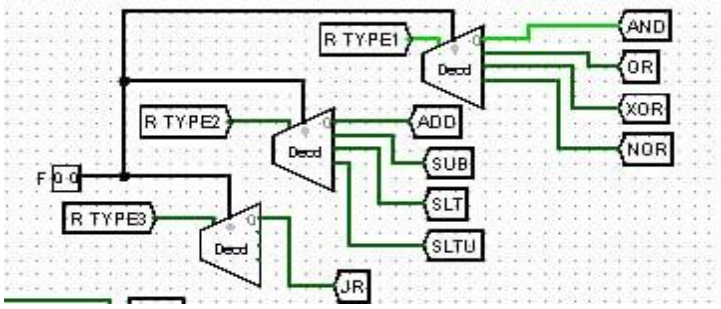
- The output of the next pc (PC) is the address of the ROM which store the instructions, the output (PC+1) is an input for register file and the output of the ROM is the instruction we will implement, we also format the instruction to (opcode, RD, RS, RT, function)

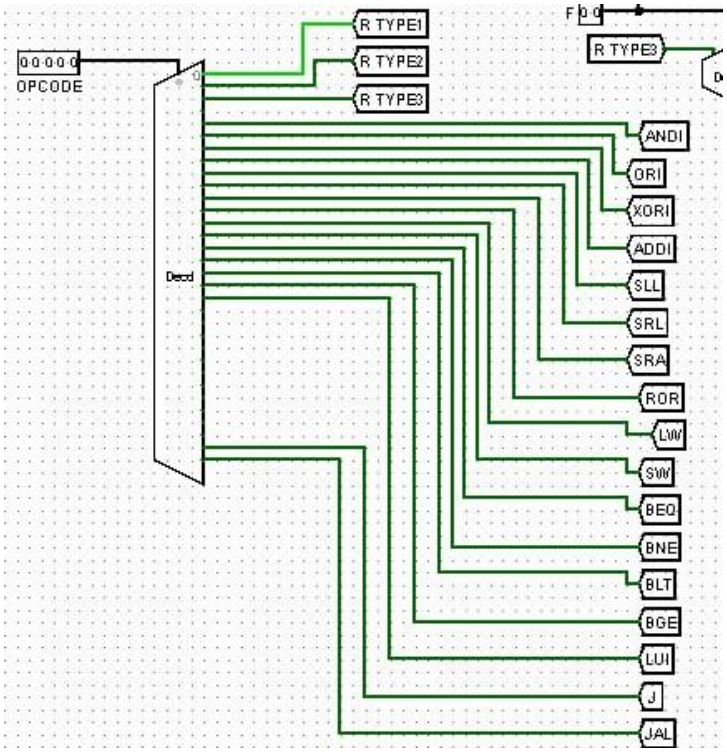
❖ Control Unit

- it is an important component that is responsible for generating all control signals to make all sub circuits work precisely.



- we use a 32*1 MUX to select one of the instructions according to the value of the 5 bits of the opcode ,for opcodes 0,1,2 we have differnet instructions due to the value of the 2 bits of function we use Rtype 1 and R type 2 and R type 3 as enables of the three decoders as shown in the figure and use the 2 bits of function as selectors of these decoders , so if the opcode is 0 or 1 or 2 one of these decoders is enabled and the other is disabled





❖ Instruction Truth Table

R-TYPE truth table : (AND, OR, XOR, NOR, ADD, SUB, SLT, SLTU, JR)

operation	op code	JREG	PCSRC	JUMPY	REGDST1	REGDST2	REGDST3	REGWRITE	EXTOP	ALUSRC	C IN	BRANCH	OP SELECT	ALU SELECT	MEM TO REG	LOAD	STORE
AND	0	0	0	0	1	0	0	1	x	0	x	xx	0	10	0	0	0
OR	0	0	0	0	1	0	0	1	x	0	x	xx	1	10	0	0	0
XOR	0	0	0	0	1	0	0	1	x	0	x	xx	11	10	0	0	0
NOR	0	0	0	0	1	0	0	1	x	0	x	xx	10	10	0	0	0
ADD	1	0	0	0	1	0	0	1	x	0	0	xx	0	1	0	0	0
SUB	1	0	0	0	1	0	0	1	x	0	1	xx	0	1	0	0	0
SLT	1	0	0	0	1	0	0	1	x	0	1	xx	1	1	0	0	0
SLTU	1	0	0	0	1	0	0	1	x	0	1	xx	10	1	0	0	0
JR	2	1	0	0	x	x	x	0	x	x	x	xx	xx	xx	xx	0	0

I-TYPE truth table : (ANDI, ORI, XORI, ADDI, SLL, SRL, SRA, ROR, LW, SW, BEQ, BNE, BLT, BGE)

ANDI	4	0	0	0	0	0	0	1	0	1	x	xx	0	10	0	0	0
ORI	5	0	0	0	0	0	0	1	0	1	x	xx	1	10	0	0	0
XORI	6	0	0	0	0	0	0	1	0	1	x	xx	11	10	0	0	0
ADDI	7	0	0	0	0	0	0	1	1	1	0	xx	0	1	0	0	0
SLL	8	0	0	0	0	0	0	1	x	1	x	xx	0	0	0	0	0
SRL	9	0	0	0	0	0	0	1	x	1	x	xx	1	0	0	0	0
SRA	10	0	0	0	0	0	0	1	x	1	x	xx	10	0	0	0	0
ROR	11	0	0	0	0	0	0	1	x	1	x	xx	11	0	0	0	0
LW	12	0	0	0	0	0	0	1	1	1	0	xx	0	1	1	1	0
SW	13	0	0	0	x	x	x	0	1	1	0	xx	0	1	xx	0	1
BEQ	14	x	1	0	x	x	x	0	x	x	x	0	xx	xx	xx	0	0
BNE	15	x	1	0	x	x	x	0	x	x	x	1	xx	xx	xx	0	0
BLT	16	x	1	0	x	x	x	0	x	x	x	10	xx	xx	xx	0	0
BGE	17	x	1	0	x	x	x	0	x	x	x	11	xx	xx	xx	0	0

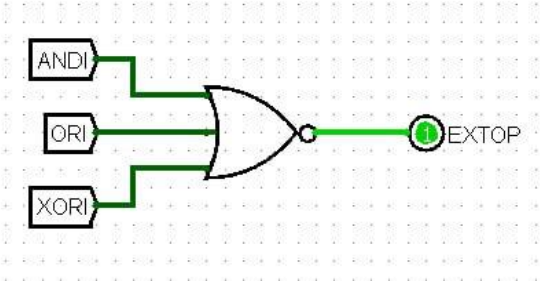
J-TYPE truth table : (LUI, J, JAL)

LUI	18	0	0	0	x	x	1	1	x	x	x	xx	xx	xx	11	0	0
J	30	x	x	1	x	x	x	0	x	x	x	xx	xx	xx	xx	0	0
JAL	31	x	x	1	x	1	0	1	x	x	x	xx	xx	xx	10	0	0

❖ Control Unit Signals

1) **The Extop Signal :** it determines the type of extension sign extend (extend due to the sign of the MSB) or zero extend we want it equal 0 in the immediate logic operations

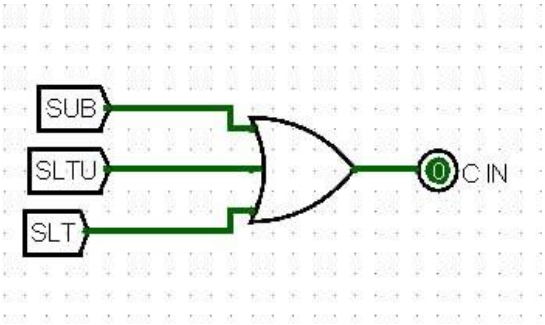
hint : we assume all don't cares equal one to simplify the design



$$EXTOP = \overline{(ANDI + ORI + XORI)}$$

2) **The C IN SIGNAL** : it refers to the the carry in of the full adder used in ALU . we want it equal 1 when we use the adder for subtraction

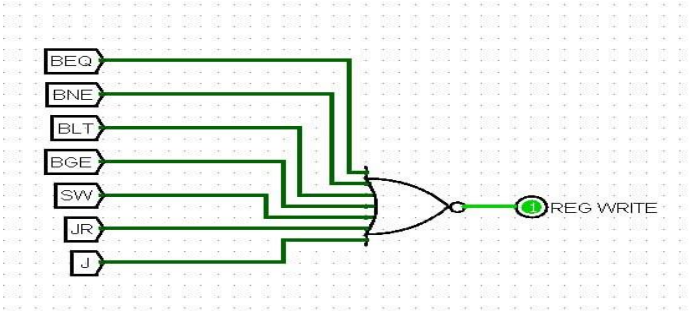
hint : we assume all don't cares equal zero to simplify the design



$$C\ IN = (SUB + SLT + SLTU)$$

3) **The REGWRITE Signal:** it determines if we will write in the register file or not

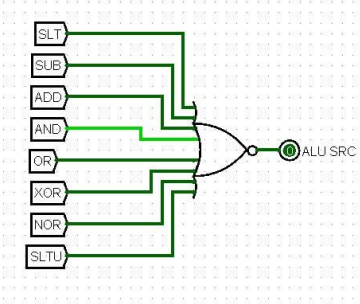
$$REGWRITE = \overline{(BEQ + BNE + BLT + BGE + SW + J + JR)}$$



4) **The ALUSRC Signal :** it selectes the second operand of the ALU is BUS B or IMM value

hint : we assume all don't cares equal one to simplify the design

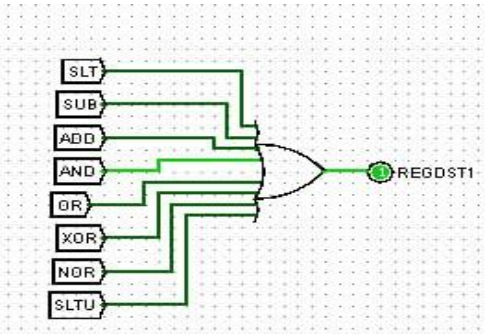
$$ALU\ SRC = \overline{(SLT + SUB + ADD + AND + OR + XOR + NOR + SLTU)}$$



5) **The REGDST1 Signal :** it selects the destination register (Rs or Rt)

hint : we assume all don't cares equal zero to simplify the design

REGDST1 = (SLT+SUB+ADD+AND+OR+XOR+NOR+SLTU)



6) **The REGDST2 Signal :** it selcts the destination between MUX 1 and R7

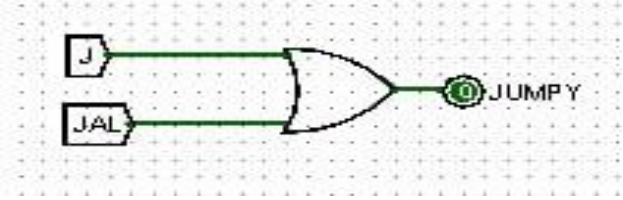
REGDST2 = JAL

7) **The REGDST3 Signal :** it selcts the destination between MUX2 and R1

REGDST3 = LUI

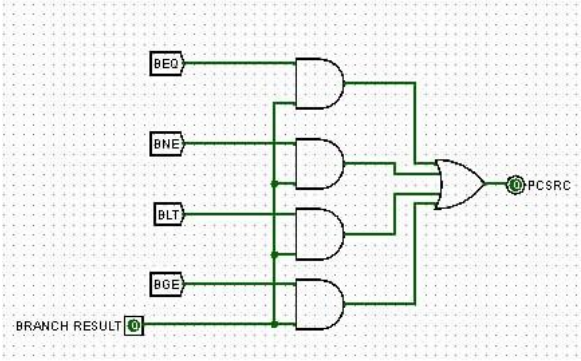
8) **The Jumpy Signal :** it helps to selcet the target address of the jump instruction

JUMPY = J + JAL



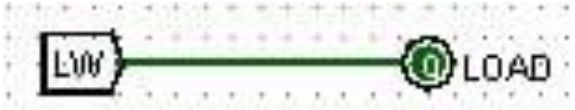
9) **The PCSRC Signal :** it determines if the branch taken or not, it equals one if BEQ = 1 and check of BEQ = 1 OR , BNE = 1 and check of BNE = 1 or BLT = 1 and check of BLT = 1 OR BGE =1 and check of BGE =1 .

PCSRC = (BRANCH OUT for equal .BEQ) + (BRANCH OUT FOR not equal .BNE)+ (BRANCH OUT for less than .BLT)+ (BRANCH OUT for greater or equal .BGE)



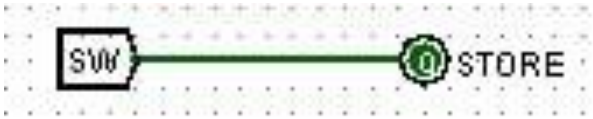
10) **The Load Signal :** it determines if we will load value from the data memory or not

LOAD = LW



11) **The Store Signal :** it determines if we will store value in the data memory or not

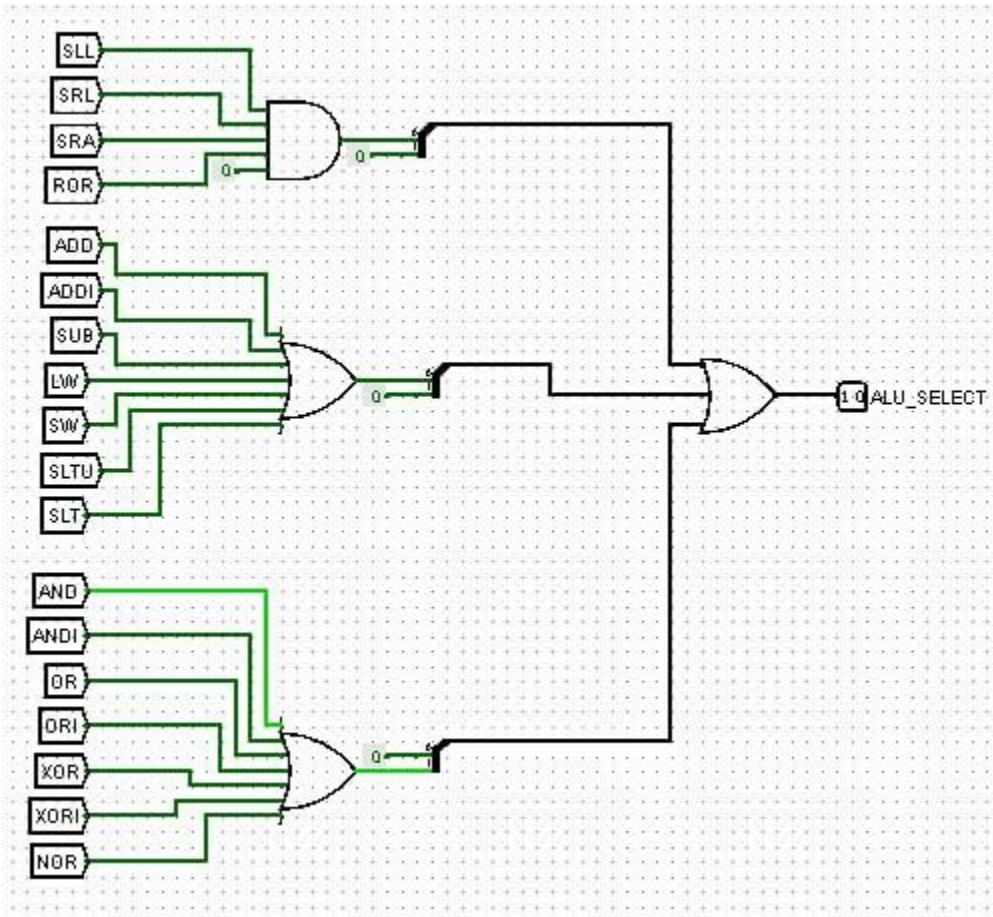
STORE = SW



12) **The JREG Signal :** it is used for the JR instruction

JREG = JR

13) **The ALU_SELECT :** it is a 2 bits signal that helps to select one operation from the mux (one operation from shift and rotate operations and one operation from arthimetic operations one operation from logic operation)



• The General Test Code

- This code is used for testing the processor by executing all instructions listed in the project by a consecutive sequence listed down below and we compared the output of the processor to the expected value we calculated and we found that they are typical and her down below the explanation for all instructions

- These instructions is used for storing the initial values of the memory for at the determined locations automatically instead of storing them by hand

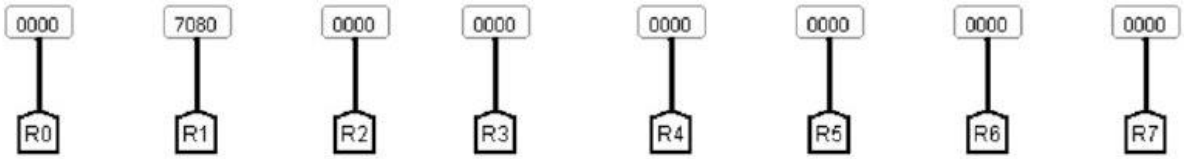
NO	instruction	code in hex
1	ADDI $2,$0,1	0x3842
2	SW $2,0($0)	0x6802
3	SW $2,1($0)	0x6842
4	ADDI $3,$0,10	0x3A83
5	SW $3,2($0)	0x6883
6	LUI 536	0x9218
7	ORI $4,$1,10	0x2A8C
8	ADDI $5,$5,15	0x3BED
9	ADDI $5,$5,15	0x3BED
a	ADDI $5,$5,15	0x3BED
b	ADDI $5,$5,1	0x386D
c	SW $4,14($5)	0x6BAC
d	ADD $1,$0,$0	0x0840
e	LUI 922	0x939A
f	ORI $6,$1,2	0x288E
10	SW $6,15($5)	0x6BEE

- The Code Instruction

NO	instruction	code in hex	expected value	final expected value
1	Lui 0x384	0x9384	R1 = 0X7080	R1 = 0x0037
2	Addi $5, $1,13	0x3B4D	R5 = 0X708D	R2=0x430a
3	Xor $3, $1, $5	0x04CD	R3 = 0X 000D	R3 =0xc280
4	Lw $1, 0($0)	0x6001	R1 = 0X0001	R4 = 0X4302
5	Lw $2, 1($0)	0x6042	R2 = 0X0001	R5 =0X000a
6	Lw $3, 2($0)	0x6083	R3 = 0X000a	R6 = 0X1850
7	Addi $4, $4, 10	0x3AA4	R4 = 0X000a	R7 =0X000f
8	Sub $4, $4, R4	0x0B24	R4 = 0X0000	
9	L2: Add $4, $2, $4	0x0914	R4 = 0X0001	
a	Slt $6, $2, $3	0x0D93	R6 = 0X0001	
b	Beq $6, $0, L1	0x70F0	BRANCH NOT TAKEN	
c	Add $2, $1, $2	0x088A	R2 = 0X0002	
d	Beq $0, $0, L2	0x7700	BRANCH TAKEN	
e	L1: Sw $4, 0($0)	0x6804	MEM{0} = 0X0073	
f	Jal func	0xF804	R7 = 0X000f	
10	Sll $3, $2, 6	0x4193	R3 = 0XC280	
11	ROR $6, $3, 3	0x58DE	R6 = 0X1850	
12	beq $0,$0,-1	0x7000	The program ends	
13	Func: or $5, $2, $3	0x0353	R5 = 0X000a	
14	Lw $1, 0($0)	0x6001	R1 = 0X0037	
15	Lw $2, 5($1)	0x614A	R2 = 0X430a	
16	Lw $3 ,6($1)	0x618B	R3 = 0X7342	
17	And $4, $2, $3	0x0113	R4 = 0X4302	
18	Sw $4, 0($0)	0x6804	MEM{0} = 0X4302	
19	Jr $7	0x1038	JUMP to 0010	

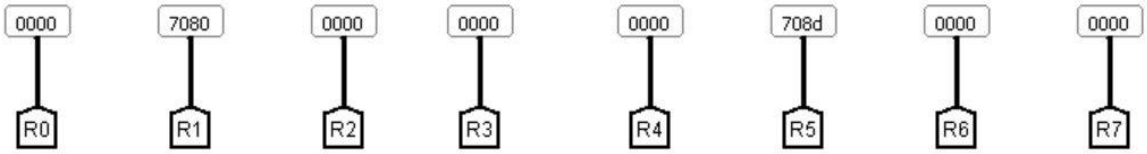
1) Lui ox384

the LUI instruction puts the value we want at the upper 11 bits of the register , so it takes the IMM 11 and shift it to the left five times and put the output value in the destination register (R1).The value 0x384 when we shift it five times it becomes 0x7080.



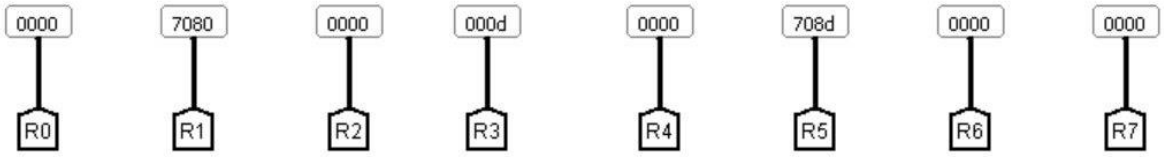
2) Addi R5,R1,13

The addi instruction makes an addition operation with two operands the first operand is the register Rs and the second operand is the the IMM 5 bits after we make a (5 to 11) bits sign extension to it and write the value in the destination register Rt , it adds the value of R1(0x7080) with 0x000d and write the result (0x708d) at R5



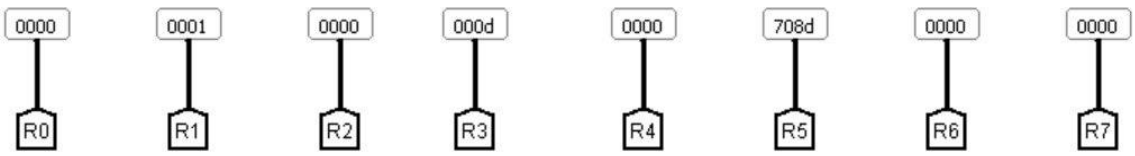
3) Xor R3,R1,R5

The xor instruction makes an xoring operation with two operands the first operand is the register Rs and the second operand is the register Rt and write the result in the destination register , it xor R1(0x7080) with R5(0x708d) and write the result (0x000d)



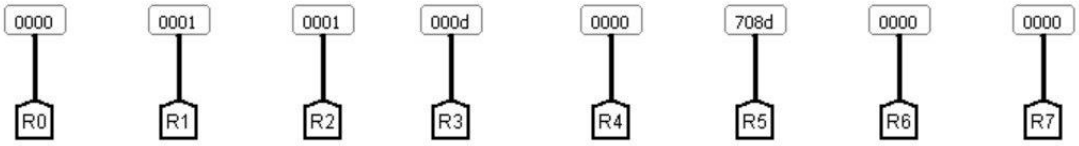
4) Lw R1,0(R0)

The lw instruction reads a value from the memory and writes it in the destination register (R1) , the address is determined through adding the base address (R0) with the sign extend of the IMM 5 (0) , so R1 will contain (0x0001)



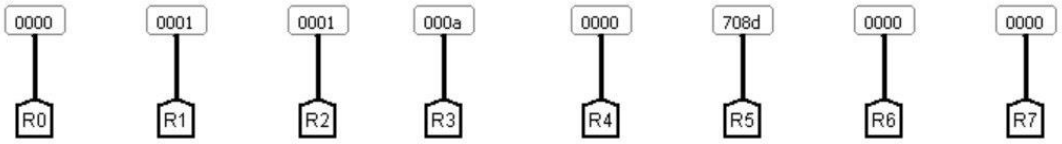
5) Lw R2,1(R0)

The lw instruction reads a value from the memory and writes it in the destination register (R2) , the address is determined through adding the base address (R0) with the sign extend of the IMM 5 (1) , so R2 will contain (0x0001)



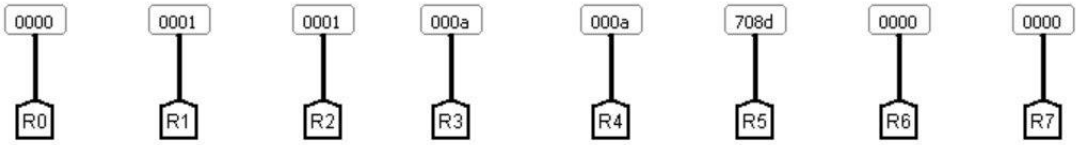
6) Lw R3,2(R0)

The lw instruction reads a value from the memory and writes it in the destination register (R3) , the address is determined through adding the base address (R0) with the sign extend of the IMM 5 (2) , so R3 will contain (0x000a)



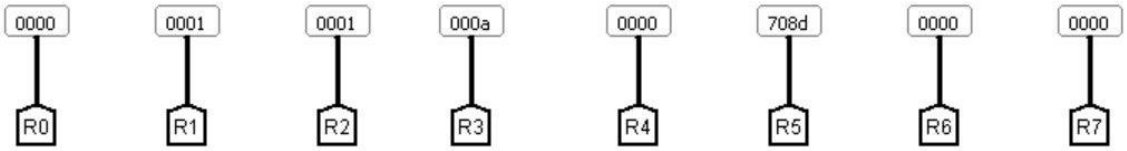
7) Addi R4,R4,10

The addi instruction makes an addition operation with two operands the first operand is the register Rs and the second operand is the the IMM 5 bits after we make a (5 to 11) bits sign extension to it and write the value in the destination register Rt , it adds the value of R4(0x0000) with 0x000a and write the result (0x000a) at R4.



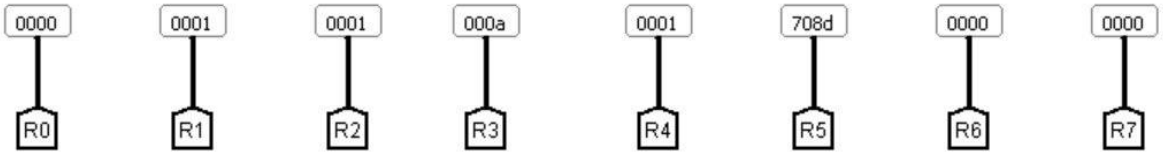
8) sub R4,R4,R4

The sub instructions makes an subtraction operation with two operands the first operand is the register Rs and the second operand is the register Rt and write in the destination register Rd , so it subtracts R4 (0x000a) from itself and write the result in R4(0X0000)



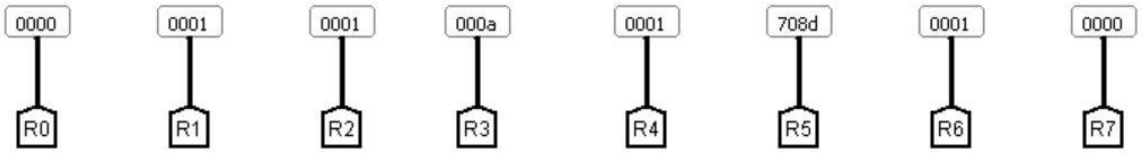
9) L2:Add R4,R2,R4

The add instructions makes an addition operation with two operands the first operand is the register Rs and the second operand is the register Rt and write in the destination register Rd , adding R2(0x0001) with R4(0X0000) and write the result in R4(0X0001)



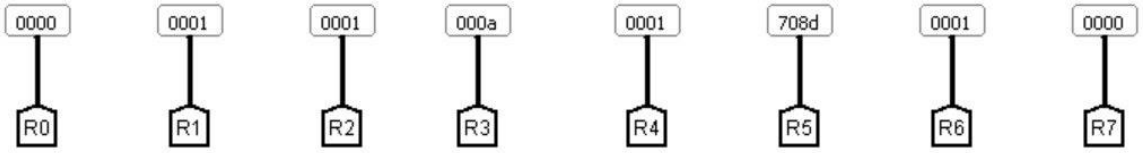
10) Slt R6,R2,R3

The slt instruction compares between two registers and if the first register is less than the second register it stores 1 at the destination register if the first is equal or bigger than the second it stores 0. Because of R2 (0x0001) is less than R3 (0x000a) so R6 will be set to (0x0001)



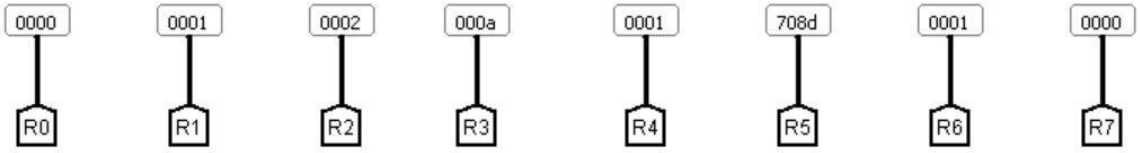
11) Beq R6,R0,L1

The beq instruction compares between two registers and if they are equal it branches to the determined label (branch target =) and if they are not equal the branch not taken and continue , R6(0x0001) does not equal to R0 so the branch is not taken and will go to the next instruction.



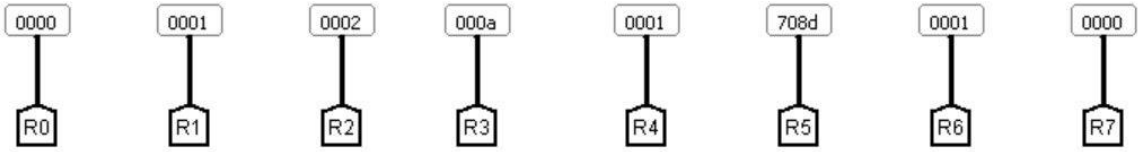
12) Add R2,R1,R2

The add instructions makes an addition operation with two operands the first operand is the register Rs and the second operand is the register Rt and write in the destination register Rd , adding R2(0x0001) with R1(0X0001) and write the result in R2(0X0002)

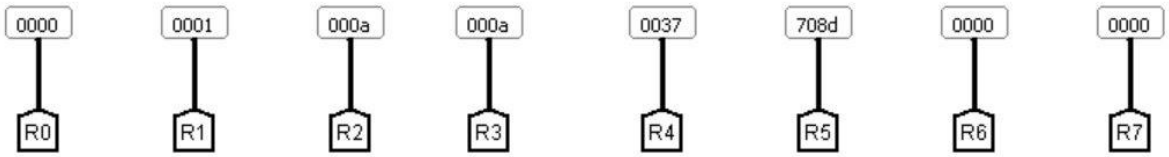


13) Beq R0,R0,L2

The beq instruction compares between two registers and if they are equal it branches to the determined label (branch target) and if they are not equal the branch not taken and continue , R0(0x0000) equal to itself so the branch is taken and will go to L2 . this branch will be repeated 9 times until R2 reaches to the same value of R3 so R6 is set to (0x0000) and the first branch will be taken and get out of that loop



The content of the registers after getting out from the loop



14) L1:Sw R4,0(R0)

The sw instruction it reads a value a the register Rt and store it at the memory at the location determined through the address is determined through adding the base address (R0) with the sign extend of the IMM 5,the value of R4(0X0037) will be written at MEM[0]

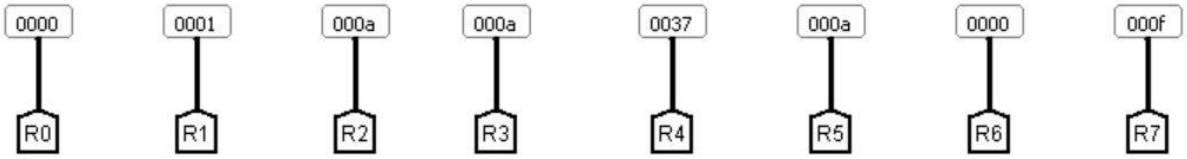
```
0000 0037 0001 000a 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0020 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0030 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 430a 7342 0000 0000
```

15) Jal func

The jal instruction writes the return address in the register R7 , PC = PC + sign extend (IMM 5)

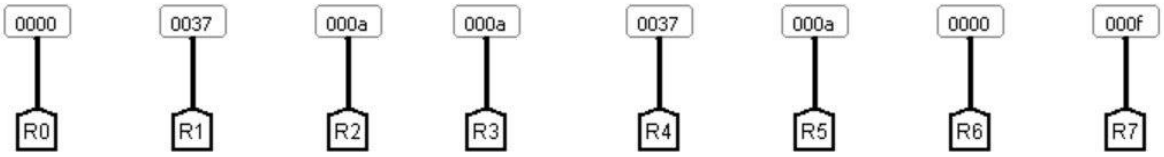
16) Or R5,R2,R3

Func: oring R2 (0X000a) with R3 (0X000a) and write the result in R5 (0X000a) .



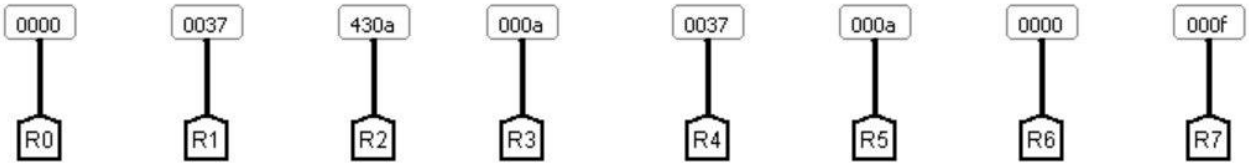
17) Lw R1,0(R0)

The lw instruction reads a value from the memory and writes it in the destination register (R1) , the address is determined through adding the base address (R0) with the sign extend of the IMM 5 (0) , so R1 will contain (0x0037)



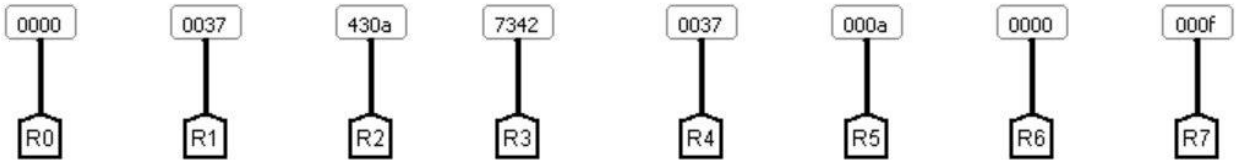
18) Lw R2,5(R1)

The lw instruction reads a value from the memory and writes it in the destination register (R2) , the address is determined through adding the base address (R0) with the sign extend of the IMM 5, so R2 will contain (0x430a) (content of MEM[60])



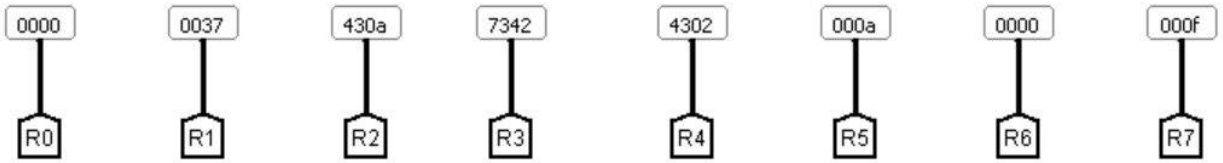
19) Lw R3,6(R1)

The lw instruction reads a value from the memory and writes it in the destination register (R3) , the address is determined through adding the base address (R0) with the sign extend of the IMM 5, so R3 will contain (0x7342) (content of MEM[61])



20) And R4,R2,R3

The and instruction makes an anding operation with two operands the first operand is the register Rs and the second operand is the register Rt and write the result in the destination register Rd. anding R2 (0x430a) with R3(0x7342) and write the result in R4(0x4302)



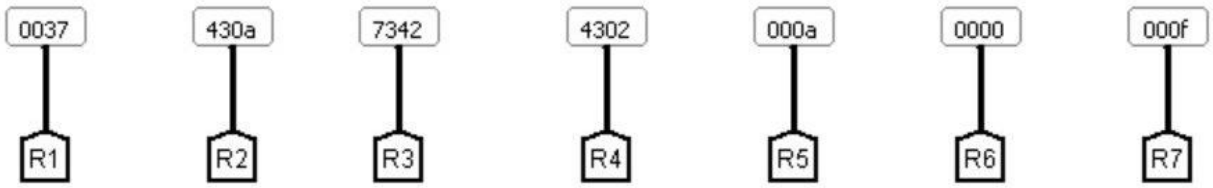
21) Sw R4,0(R0)

The sw instruction it reads a value a the register Rt and store it at the memory at the location determined through the address is determined through adding the base address (R0) with the sign extend of the IMM 5,the value of R4(0X4302) will be written at MEM[0]

```
0000 4302 0001 000a 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0020 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0030 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 430a 7342 0000 0000
```

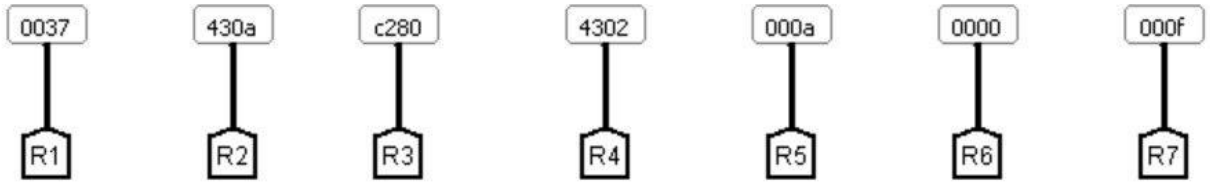
22) Jr R7

The jr instruction helps to return the next instruction after the function called PC = RS



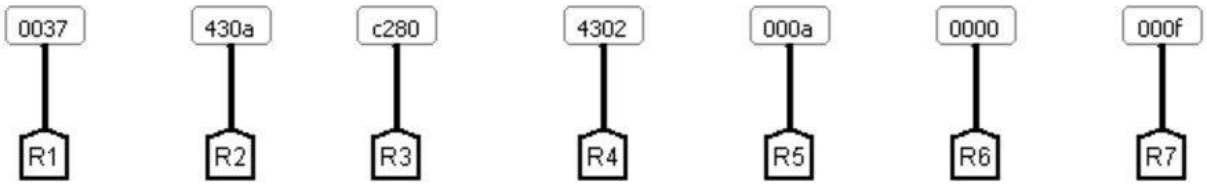
23) Sll R3,R2,6

The sll instruction it shifts the register RS by amount (lower IMM4) and write the result in the destination register Rt , shifting R2 (0x430a) by 6 and write the result in R3(0xc280)



24) ROR R6,R3,3

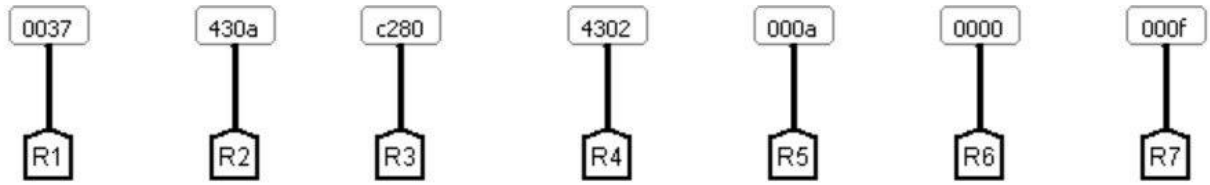
The ROR instruction makes a rotate to the right to the register Rs. the amount of shift is determined by the (IMM 4) the lower 4 bits of the imm value and write the result in the destination register Rt. Rotating R3 by 3 and write the result in R6 (0x1850)



25) Beq R0,R0,-1

After that the program ends

- This is the final result



- # The Array Program

- This program adds an array with size 3 of integers [1 , 2 , 3] using two procedures, the main procedures initializes the array elements, it then calls the second procedures after passing the array address and the number of elements in two registers to compute the sum of array elemnts and return the result in a register.

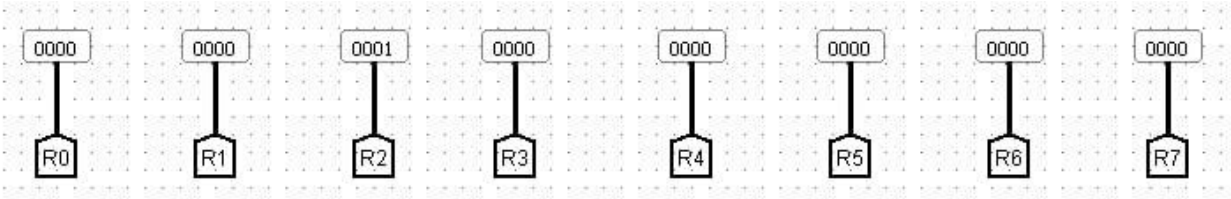
- Program Code

```
addi R2,R2,1
sw R2, 0(R0)
addi R2,R2,1
sw R2, 1(R0)
addi R2,R2,1
sw R2, 2(R0)
add R2,R0,R0
addi R3 , R0, 3
add R1,R0, R0
jal sum
loop: beq R0 , R0 , loop
sum: slt R4, R2 , R3
beq R4 , R0 , end
addi R2,R2, 1
lw R4, 0(R1)
addi R1, R1, 1
add R5 , R4 , R5
j sum
end: jr R7
```

- Execution Of Each Instruction

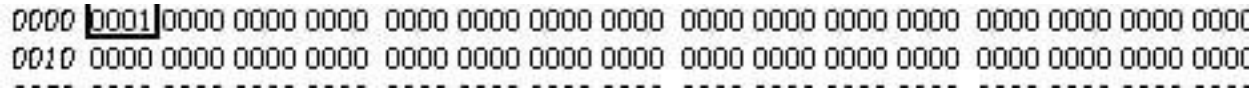
26) addi R2,R2,1

first load the register R2 with value 1



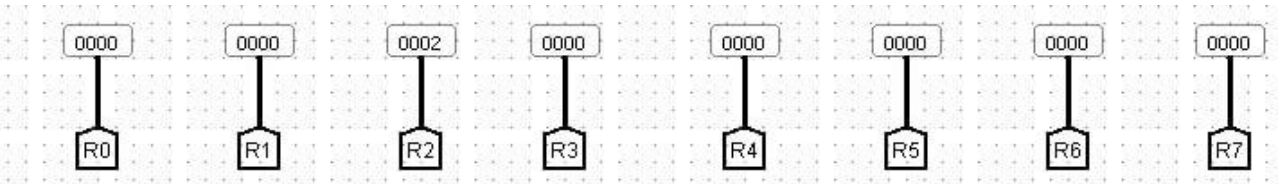
27) sw R2, 0(R0)

store the value in R2 to location 0 in memory



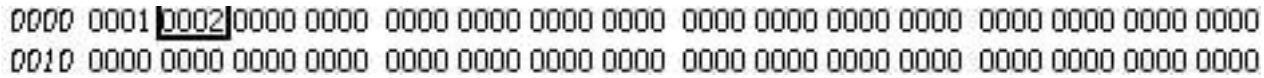
28) addi R2,R2,1

then add 1 to R2 to be 2



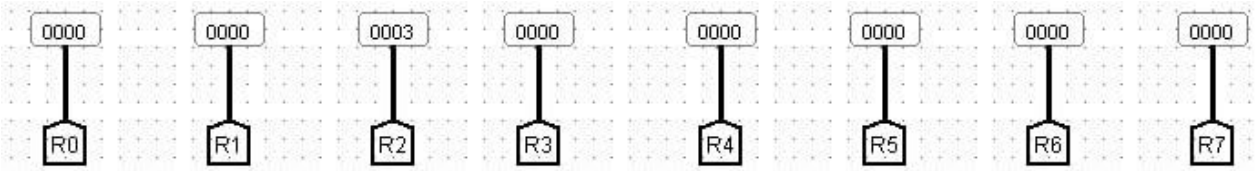
29) sw R2,1(R0)

store 2 in location number 1



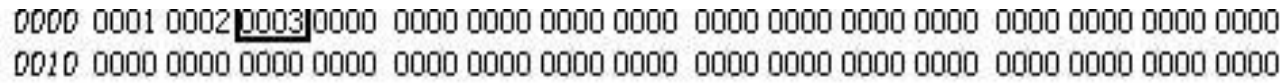
30) addi R2,R2,1

add 1 to R2 to be 3



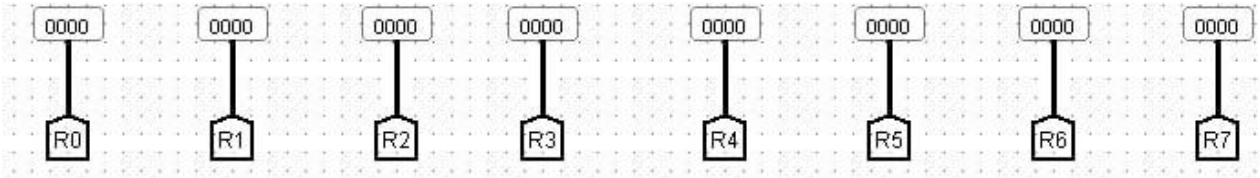
31) sw R2,2(R0)

then store R2 in location 2



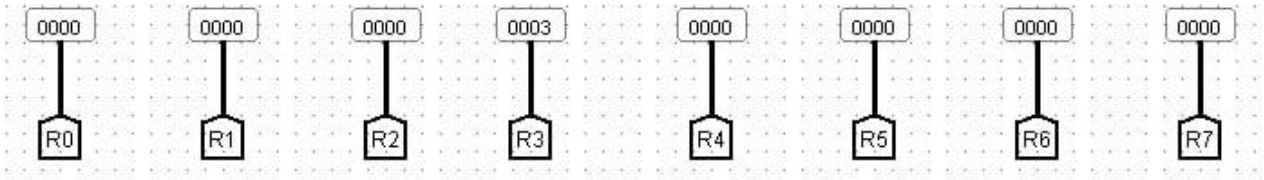
32) add R2,R0,R0

Initialize the counter R2 to 0



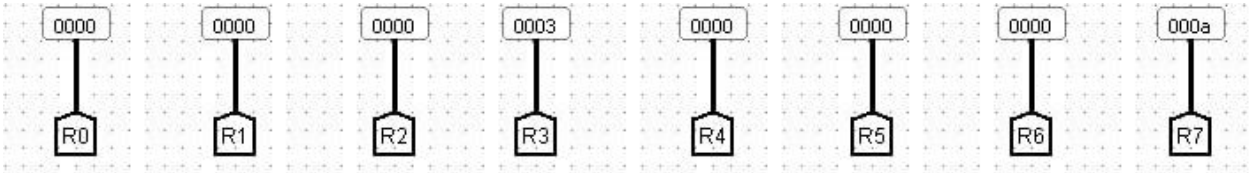
33) addi R3,R0,3

pass the number of elements to function by storing it into R3



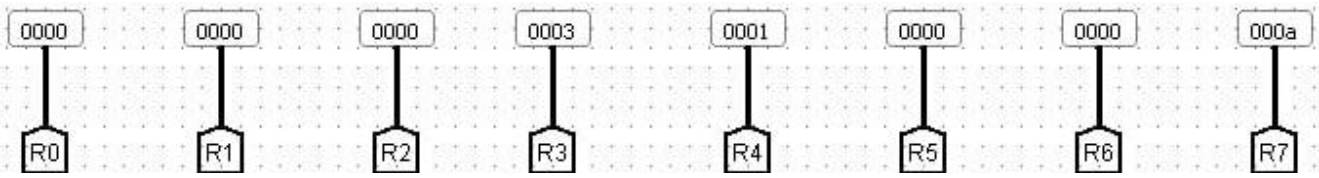
34) add R1,R0,R0

Initialize the base address to zero as in location 0



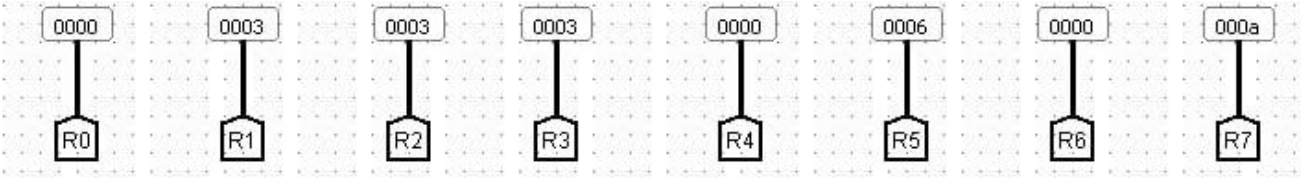
35) jal sum

jal the second procedure and store the return address into r7



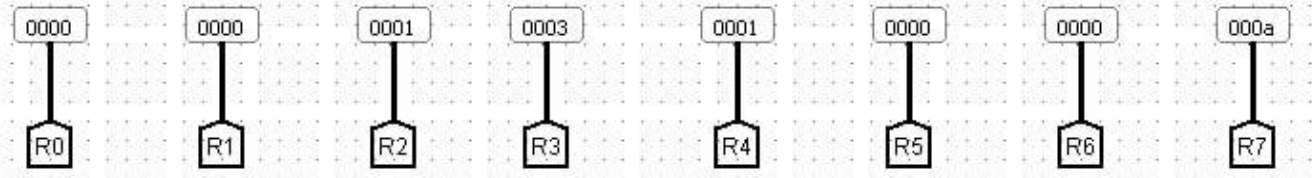
36) Loop : beq R0,R0,loop

The programe is over her to be set to this value after executing all instructions



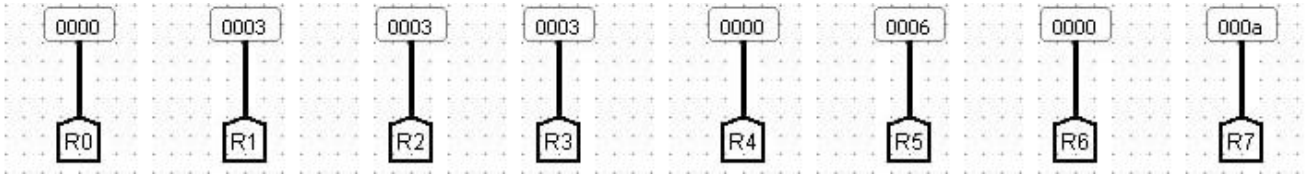
37) sum : slt R4,R2,R3

set R4 to 1 in case of R2<R3, and this is true condition so it will be set



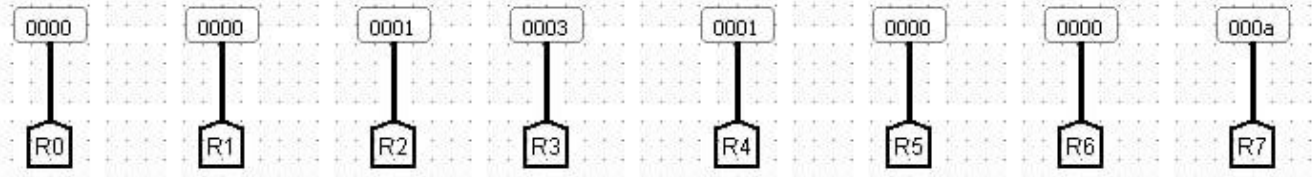
38) beq R4,R0,end

when R4 is set to zero after R2 to be 3 that's equal to number of elements in the array in R3 that's leads us out of the loop to end



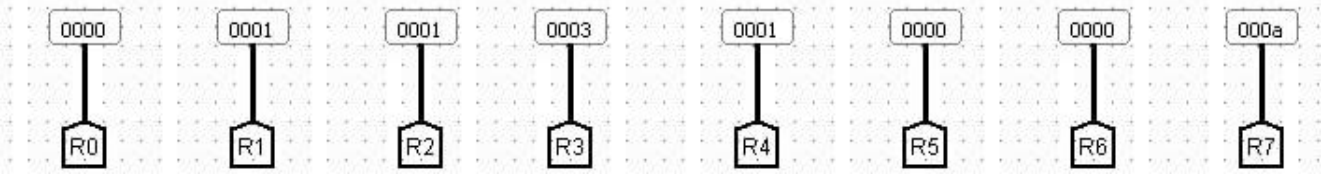
39) addi R2,R2,1

add 1 to R2 to increase the counter



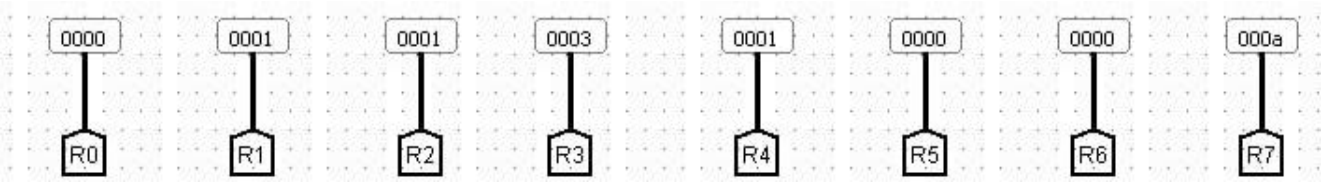
40) lw R4,0(R1)

lw the first location into r4 which is 1



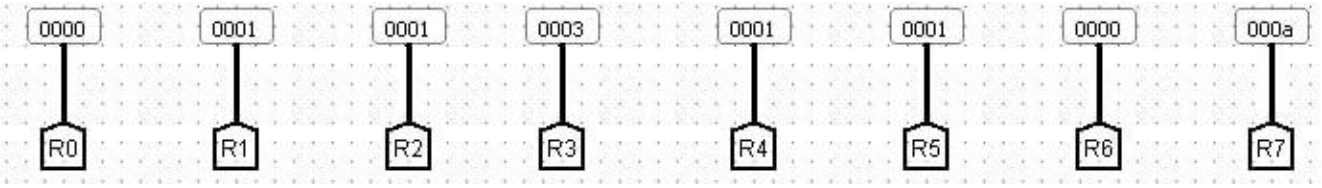
41) addi R1,R1,1

add then to the base address 1 to load the second location



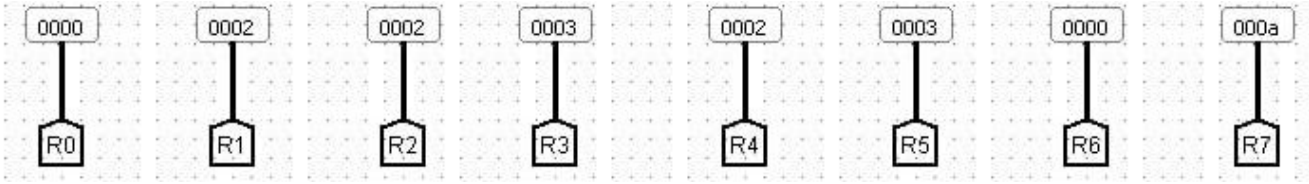
42) add R5,R4,R5

add R5 to R4 where R4 has 1 and R5 has 0 so R5 will have 1

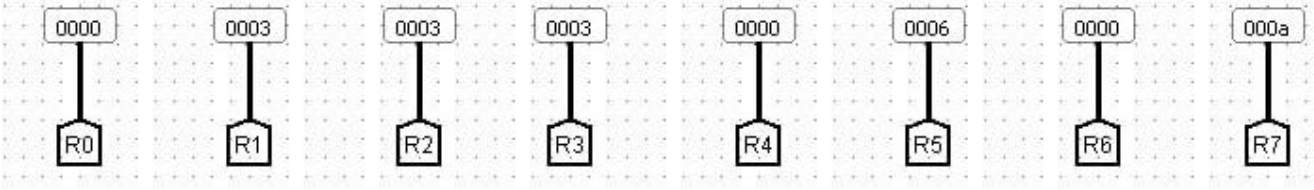


43) j sum

in the second loop we will load the second operand in the second location into r4 and then we will add it to r5 to be 3

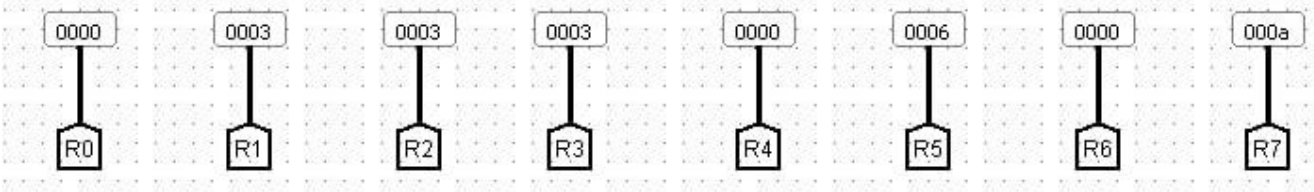


in the third loop we will add the third operand in the third location into r4 to be added to r5 to be 6



44) jr R7

we get finally R2 to be 3 to be equal to R3 which this can lead us out of the loop and jump th the address in R7 to be finally in beq to end the program



- The code Instruction

NO	Instruction	code in Hex	Expected values
1	addi R2,R2,1	3852	R2=0X0001
2	sw R2, 0(R0)	6802	MeM[0]=0X0001
3	addi R2,R2,1	3852	R2=0X0002
4	sw R2, 1(R0)	6842	MeM[1]=0X0002
5	addi R2,R2,1	3852	R2=0X0003
6	sw R2, 2(R0)	6882	MeM[2]=0X0003
7	add R2,R0,R0	880	R2=0
8	addi R3 , R0, 3	38C3	R3=3
9	add R1,R0, R0	840	R1=0
a	jal sum	F802	J sum ,R7=0X000A
b	loop: beq R0 , R0 , loop	7000	branch to the sam address
c	sum: slt R4, R2 , R3	0D13	R4=1 until last branch=0
d	beq R4 , R0 , end	71A0	not branch until last branch
e	addi R2,R2, 1	3852	R2=3
f	lw R4, 0(R1)	600C	R4=3
10	addi R1, R1, 1	3849	R1=2
11	add R5 , R4 , R5	965	R5=6
12	j sum	F7FA	jump to sum
13	end: jr R7	1038	R7=0X000A back to this address

PART two

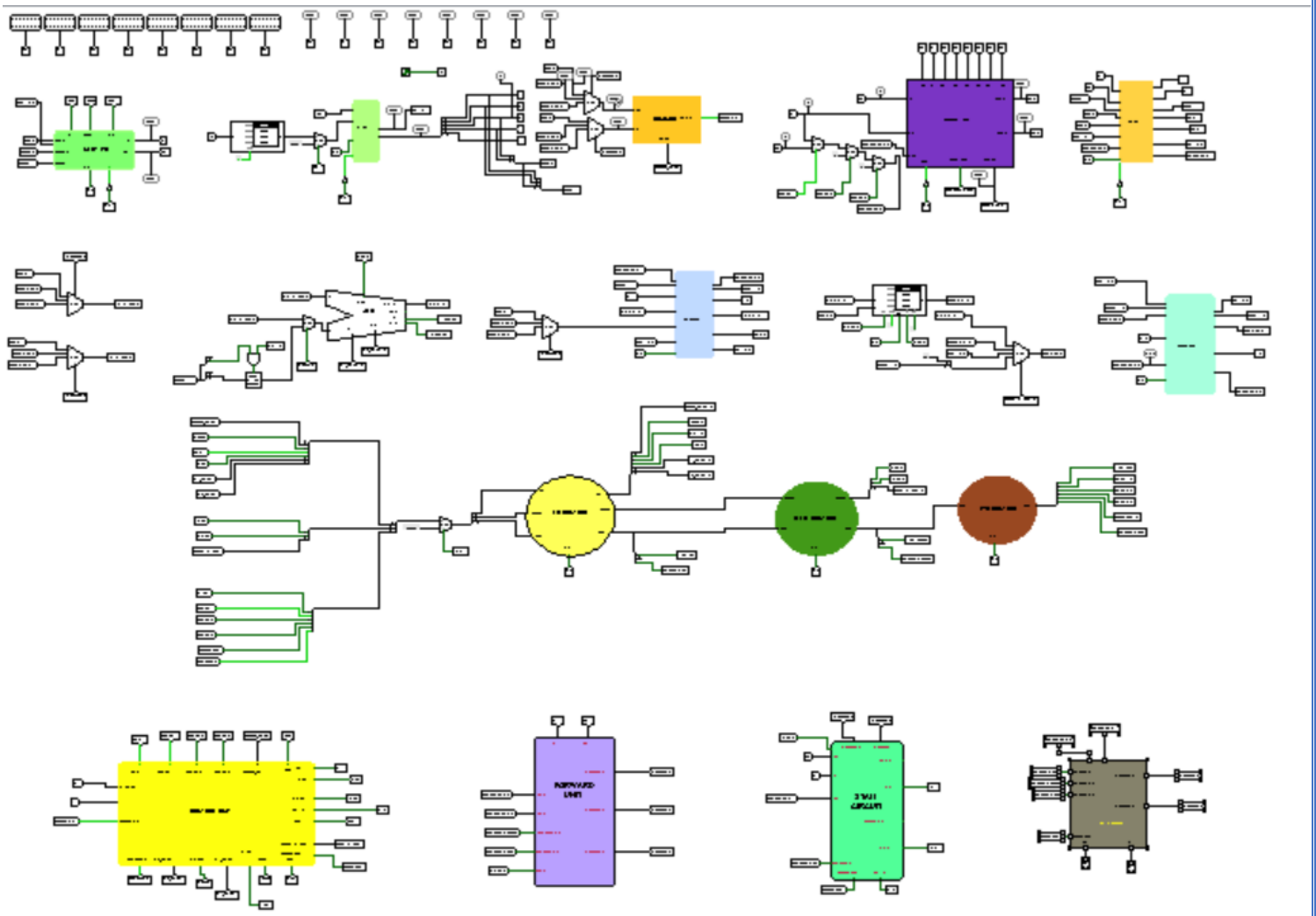
PIPELINE

Processor

❖ Introduction

- As we know in the single cycle Processor report, we design a simple 16 bit RISC processor with eight 16-bit general-purpose registers. • Pipelining is an implementation technique in which multiple instructions are over lapped in execution. • All steps in pipelining (called stages) operate concurrently As long as we have separate resources for each stage, we can pipeline the tasks. • To pipeline the single-cycle data path we add pipeline registers between stages (at end of each stage). • All the pipeline stages take a single clock cycle, so the clock cycle must be long enough to accommodate the slowest operation. • Pipelining doesn't improve latency of a single instruction; However, it improves throughput of entire workload so, Instructions are initiated and completed at a higher rate.

❖ OUR Design for pipe line processor



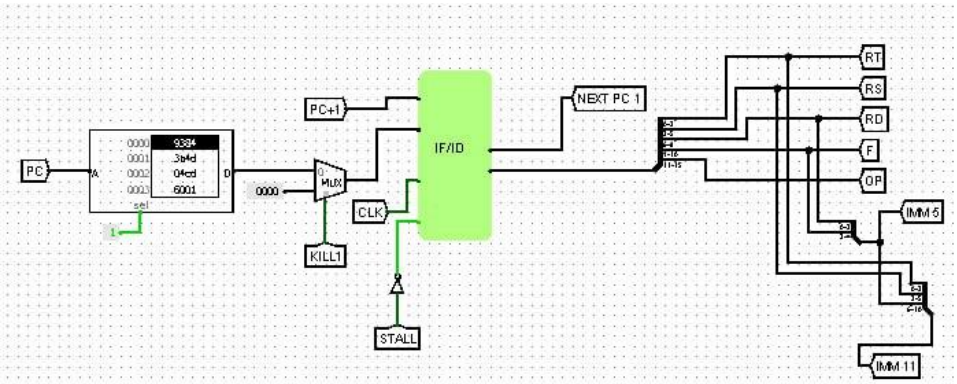
❖ The Pipeline Registers

- The Pipeline Registers (Clocked Registers)Allow Datapaths And Functional Units To Be Shared By Different Instructions During Different Stages While Retaining The Value Of An Individual Instruction.

- We Have Four Pipeline Registers (If/Id,Id/Ex,Ex/Mem,Mem/Wb).There Is Some Values That We Need To Wtore In Every Pipe Line Register Like Imm 11 ,Next Pc And The Destination Register.We Will Expalin Each Register Of Them In Details Below :

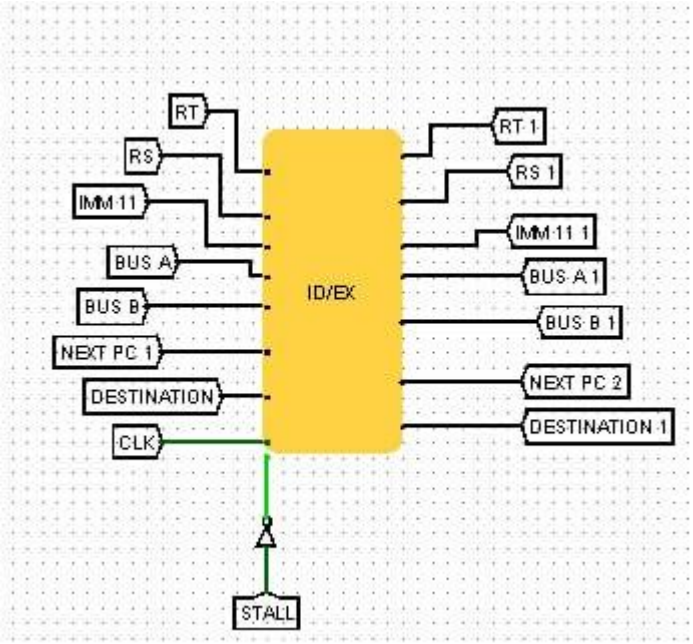
1) IF|ID Register

we store both of the instruction which is read from instruction memory or zeros in the case of flashing (no operation case) and the next pc value and from the output we split the instruction bits due to instruction format



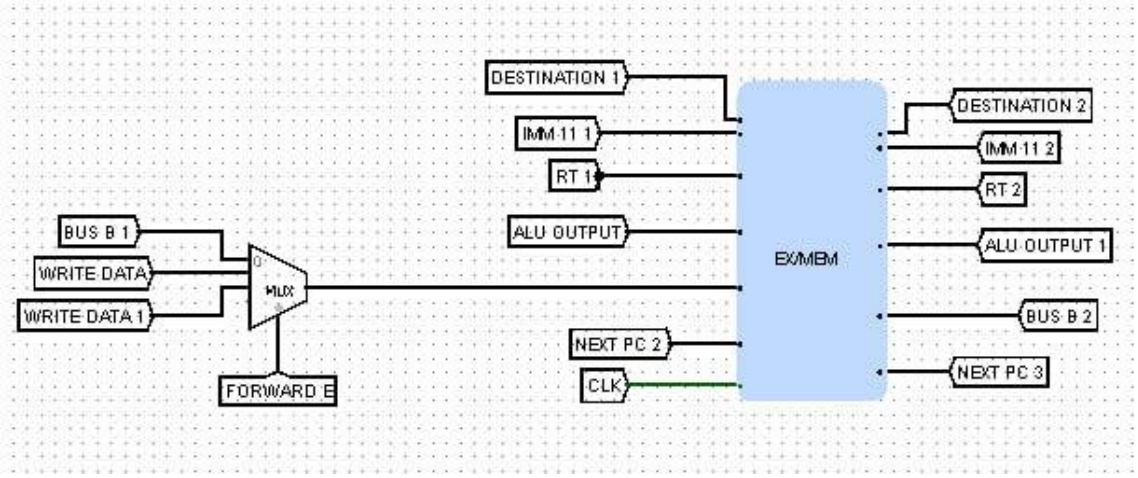
2) ID|EX Register

we store the two sources registers (RS,RT) as we need in the forward unit ,the destination register,BUS A ,BUS B ,the IMM 11 due to need of them in the next stages



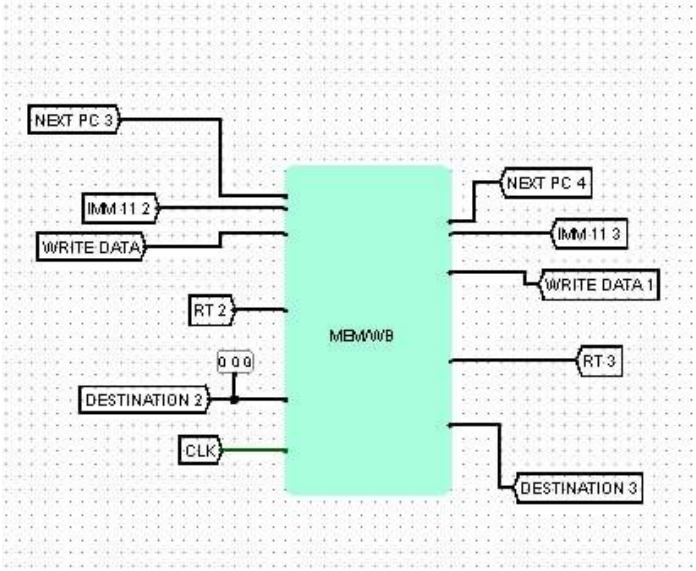
3) EX|MEM Register

we store the ALU output as we need as the address of a specific location in the data memory in the LW and SW instructions , BUS B1 as we need it to determine the value that should be written in the data memory in the SW instructions and the next pc 3 as we need it in the Jal instruction and the IMM 11 1 as we need it in the LUI instruction.



4) MEM|WB Register

we store destination 2 to be able to write in the right destiniation register and write data to be able to forward it from wb stage when we need it

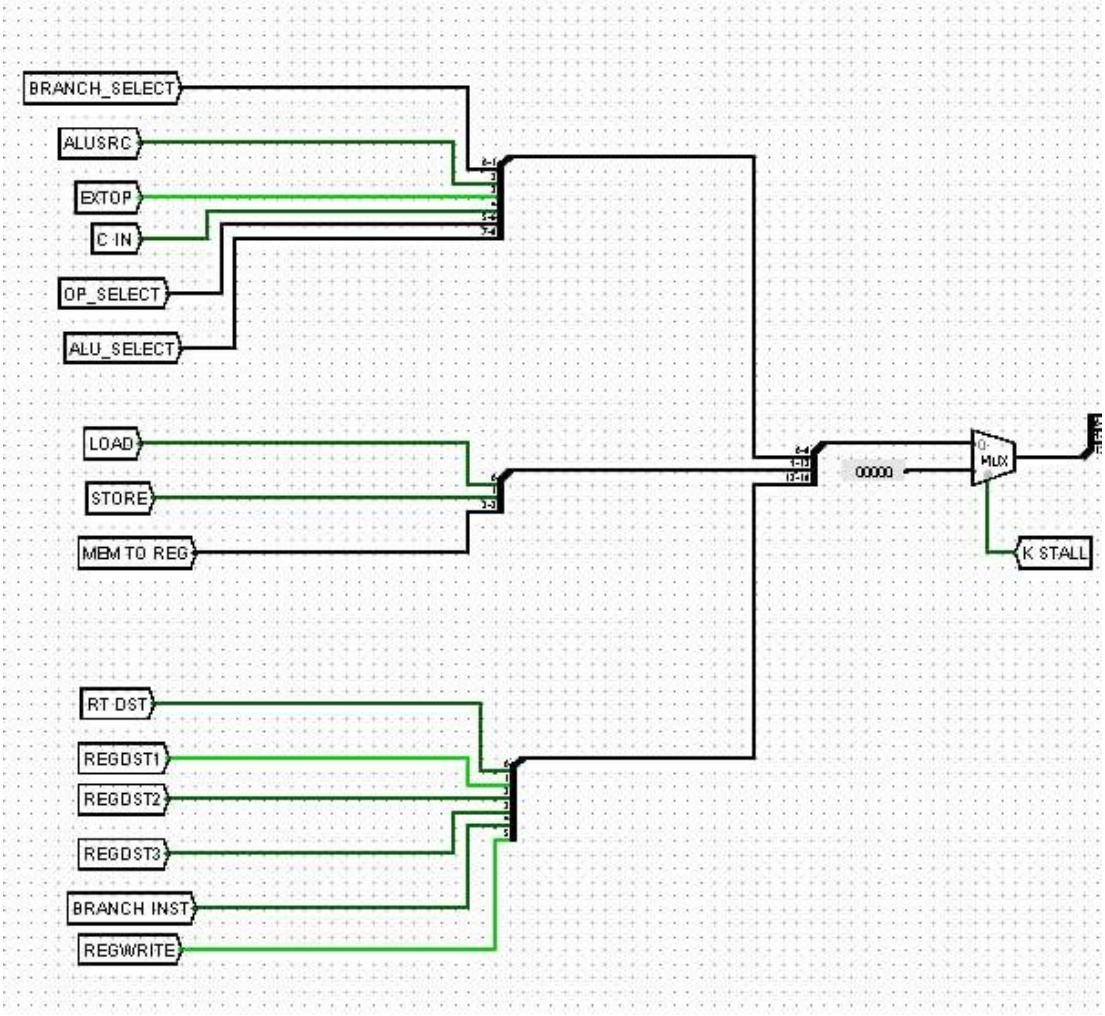


❖ Pipeline Control Signals

- Control signals in the pipeline processor uses the same signals generated from the control unit but at different stages and that is the difference between the control signals in single cycle and pipeline so we need to store these signals at registers to be able to use their correct values in the suitable stage

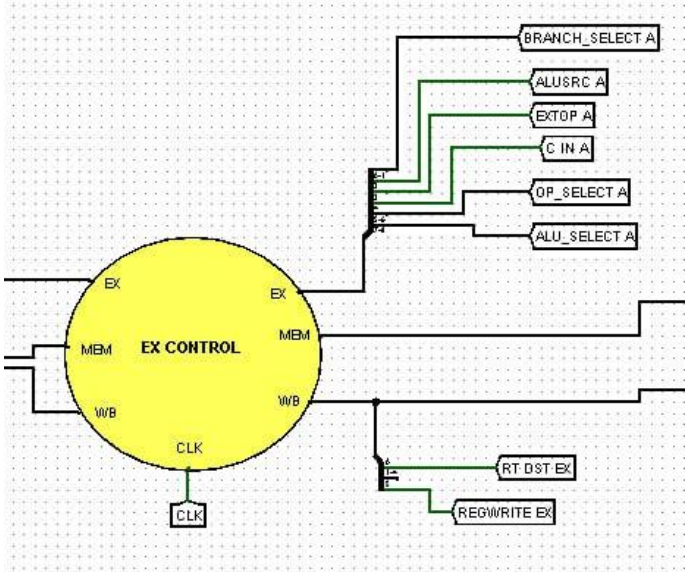
1) ID Stage

In this stage we use the direct signals from control unit and we do not need to store it because we use them in the decoding for branch circuit and we use them for the next pc circuit those signals are JUMPY ,PCSRC,JREG,BRANCH SELECT,KILL1 and KILL2



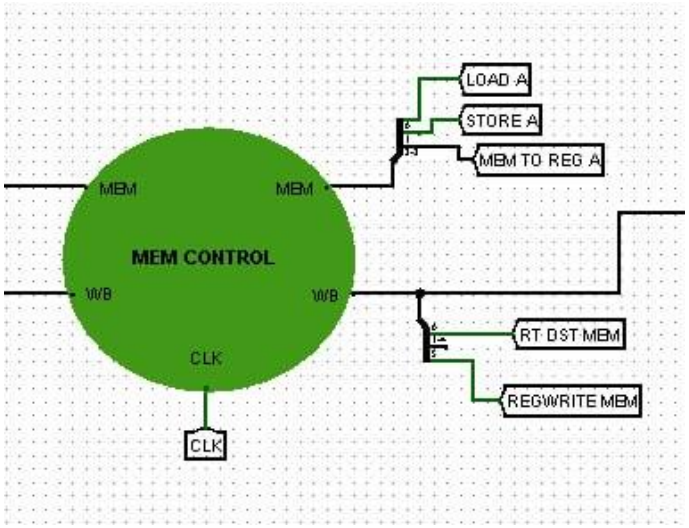
2) EX Control

we store the signals we need in the execution stage in the register and split what we need from them .those signals are ALUSRC A ,EXTOP A ,CIN A,OP_SELECT A,ALU_SELECT A and REGWRITE EX.



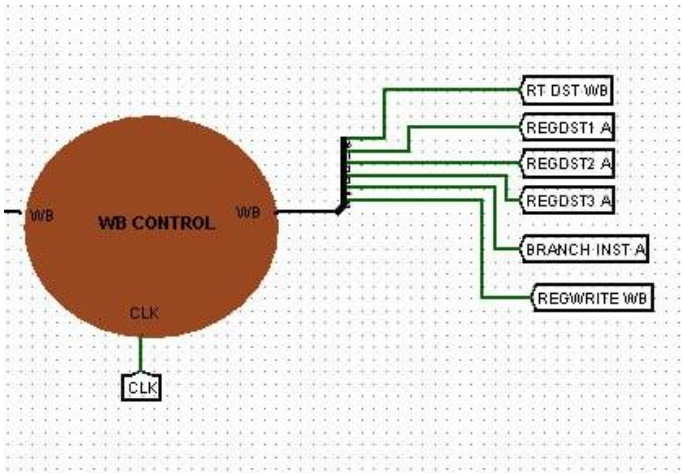
3) MEM Control

we store the signals we need in the memory stage in the register and split what we need from them .those signals are LOAD A , STORE A, MEM TO REG A,REGWRITE MEM.



4) WB Control

we store the signals we need in the write back stage in the register and split what we need from them .those signals are REGDST1 A,REGDST2 A,REGDST3 A,REGWRIRE WB.



❖ The Forward Unit

- The purpose of the forwarding unit is to guarantee that the instruction at the EX stage of the pipeline receives the correct values for its register operands RS & RT with the fewest possible stalls.

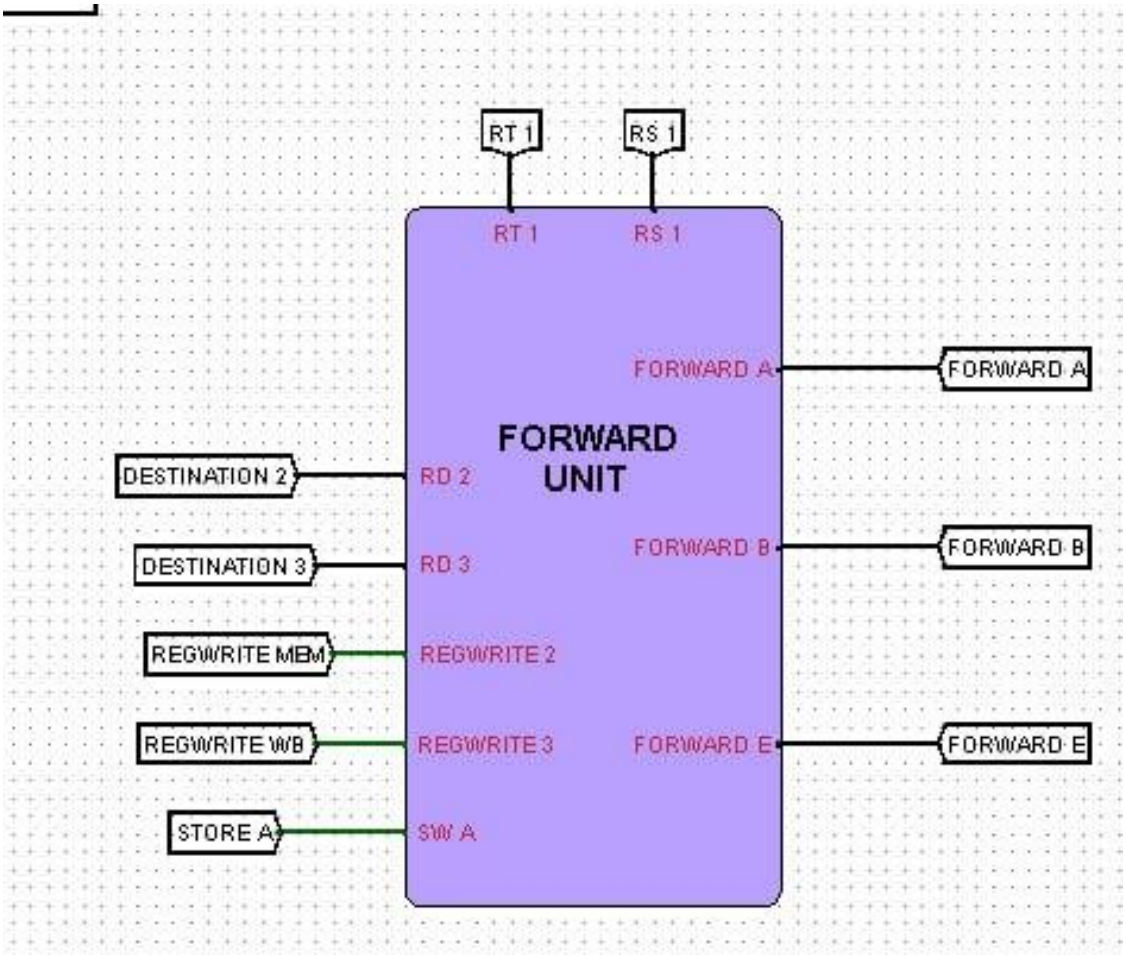
- Forwarding Reasons: • Forwarding happens when a new values "which are not already in the register file" are produced in the the processor pipeline in two locations. These locations are the EX and MEM stages. These new values are not written to the register file until the WB stage. • So, there are two cases at which old incorrect values are flowing through the pipeline. Forwarding can help cure this problem.

- The forward unit is responsible for 3 control signals A ,B and E

- A for forwarding to ALU from 2 possible stages and B the same like A but for different buses

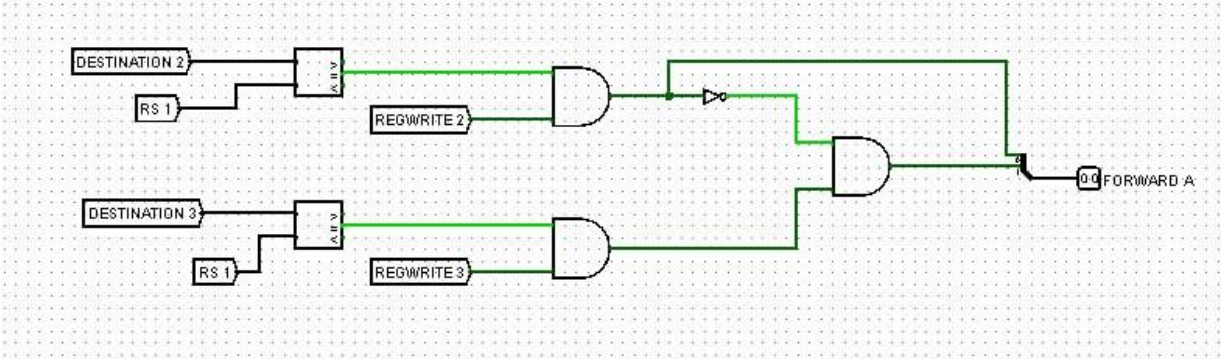
- E for forwarding to memory to choose what we will store in case of store signal

- we'll also use Destination 2 and Destination 3 to compare it with RS to see if there some kind of writing at the same time to forward it and make sure that registerwrite signal in WB and mem stage are 1 to be sure that we're writing in these stage where we are forwarding



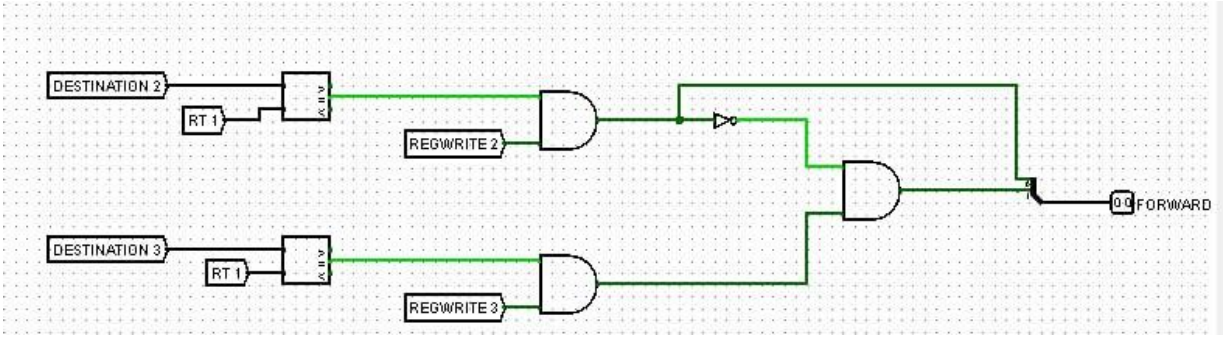
1) Forward A

The first control signal is A and we use it in case of RS equal is equal to the register destination in the mem stage or WB stage and make sure that in case of corresponding one registerwrite is 1



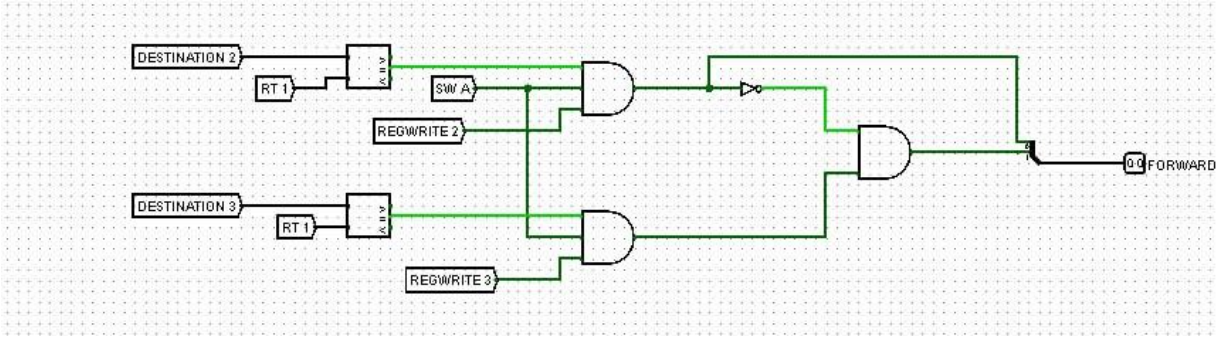
2) Forward B

The second control signal is B and we use it in case of RT equal is equal to the register destination in the mem stage or WB stage and make sure that in case of corresponding one registerwrite is 1



3) Forward E

The third control signal is E and we use it in case of RT equal is equal to the register destination in the mem stage or WB stage and make sure that in case of corresponding one registerwrite is 1 and also we should make sure that store is 1 that is the main reason for this forwarding



❖ The Branch Forward

- The purpose of the Branch Forward Unit is to guarantee that the instruction entering the comparator at ID stage receives the correct values for its register operands RS & RT with the fewest possible stalls.

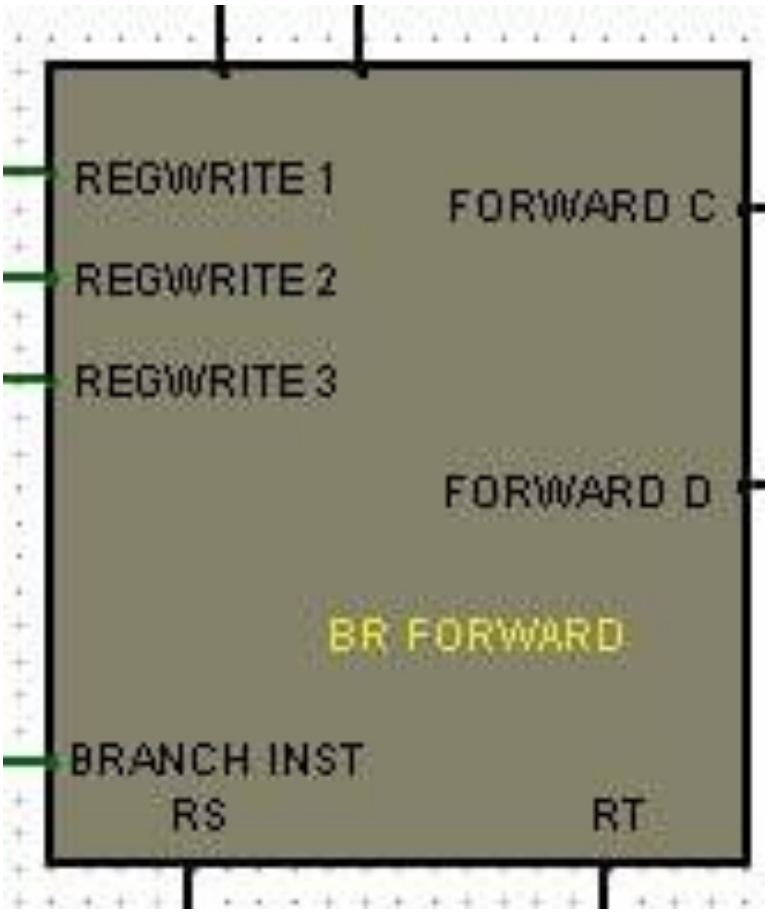
- We've made a control circuit for generating the control signals of the branch for forwarding to it by the two signals forward c and forward D

- We've some inputs to compare to check the case of forwarding to the branch

- We'll check the register write in all stage to be sure if we're writing or not

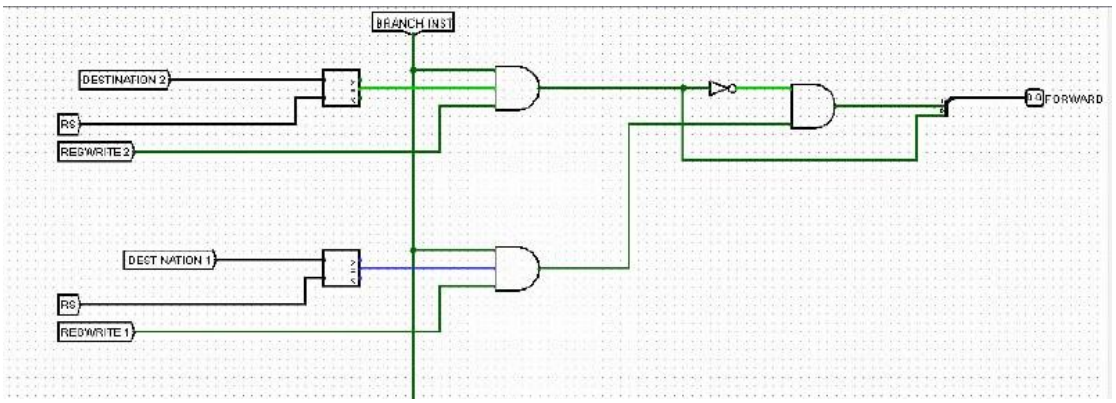
- We'll also check the the destination in two stages after decoding which is excuting and memory where we will forward from

- We'll also use RT and RS to compare them with others



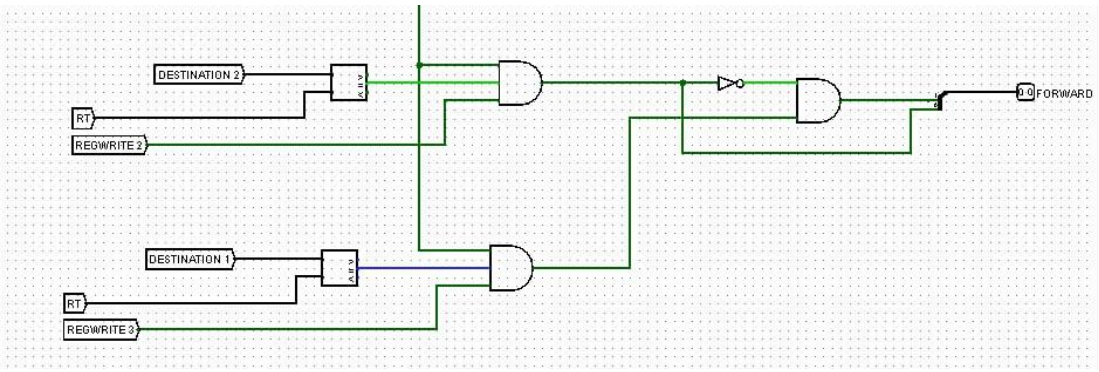
1) Forward C

for forwarding c we will compare destination 2 or 1 with RS and this will be anded to be sure that there's branch instruction and the register write in the memory stage and execution stage are 1 that's could lead us to branch into 4 cases which has implemented in the right half of the circuit



2) Forward D

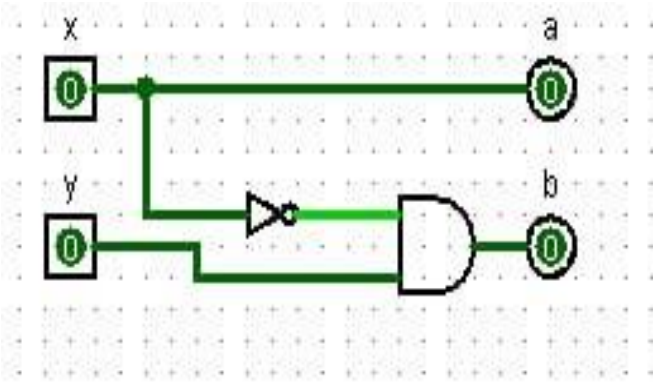
for forwarding D we will compare destination 2 or 1 with RT and this will be anded to be sure that there's branch instraction and the register write in the memory stage and excution stage are 1 that's could lead us to branch into 4 cases which has implemented in the right half of the circuite



➢ Special Circuit

We need to design a circuit with two inputs and two outputs when no forwarding is needed the out put should be 00, when the forwarding is from mem the output should be 01 and when the forwarding is from WB the output should be 10

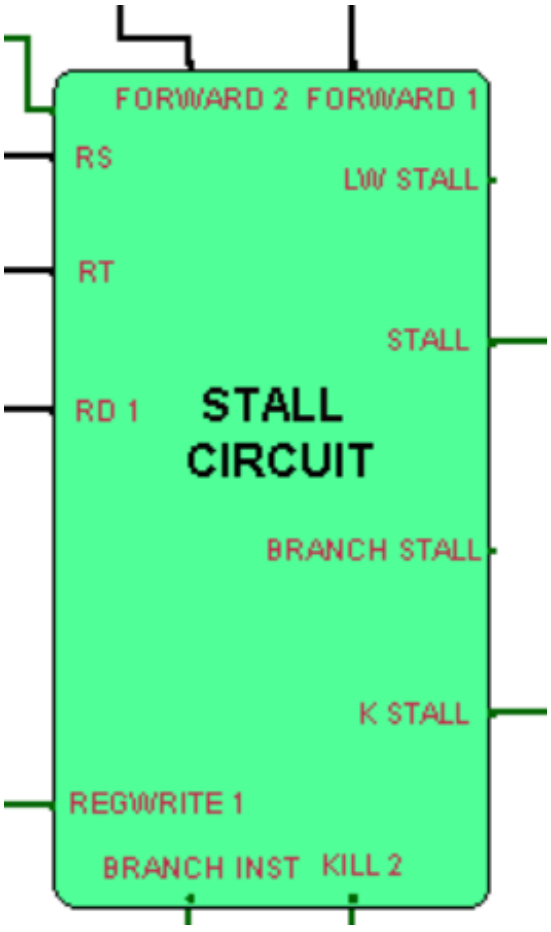
x	y	a	b
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	0



❖ The STALL Circuit

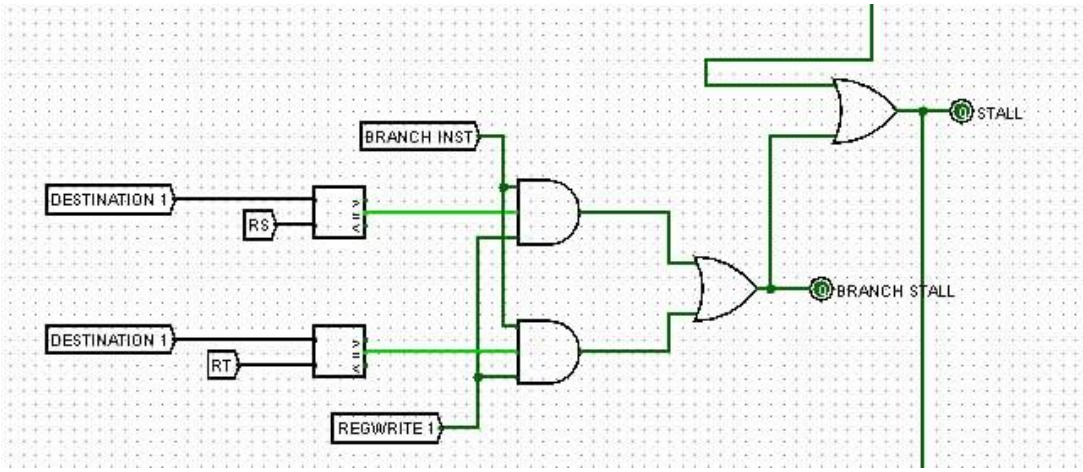
- We will need a stall circuit for stopping operations for one clock cycle for some case to be sure that all elements in registers are ready for the process

- The only case that we need a stall at is when our instruction depends on the previous instruction which new register value will not get ready until the EX stage, if that value is needed at the EX stage and because both of them are in the same cycle then it's a necessity to make a stall by Forcing control values in ID/EX register to Zero and Prevent update of PC and IF/ID register



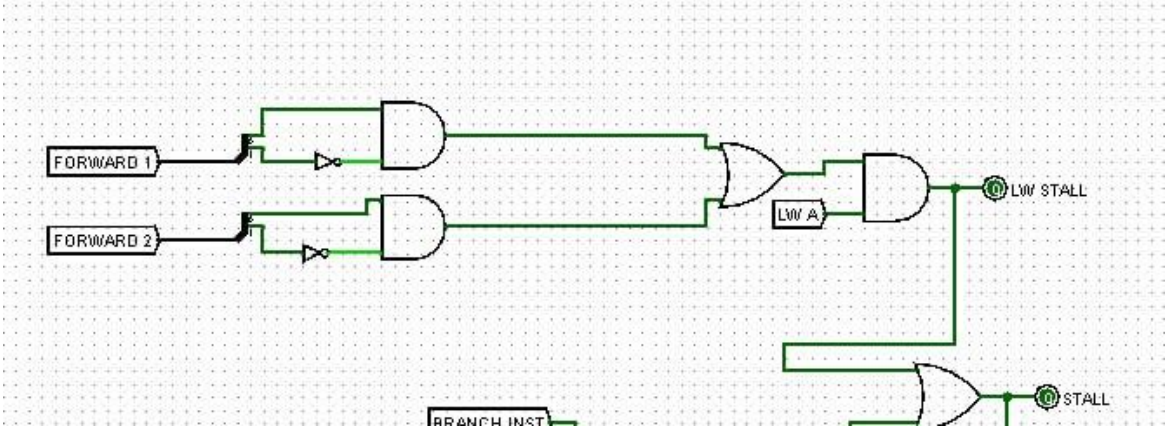
1) Branch Stall

The first case we'll stall for it is the branch stall whenever the register RT or RS which is in decoding stage with the destination in executing stage if they're equal this means I need to stall (Prevent update of PC Forcing control values in ID/EX to zero and IF/ID register)



2) LW Stall

The other case is loading case(loading A refers to loading in mem stage is 1) and we're forwarding to executing stage and from mem stage and this is un acceptable we will wait one clock cycle to make it ready to be used in executing stage

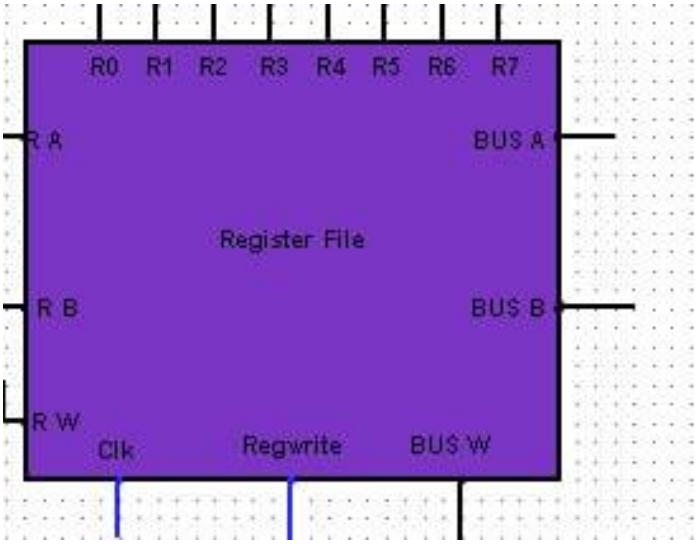


❖ Processor Components

- let's take a look on the changes we've done on the processor for pipelining it:

1) The Register File

The register file doesn't have changes from inside it has the same design as the single cycle processor the changes we have made in the data path will explain it in details in the data path

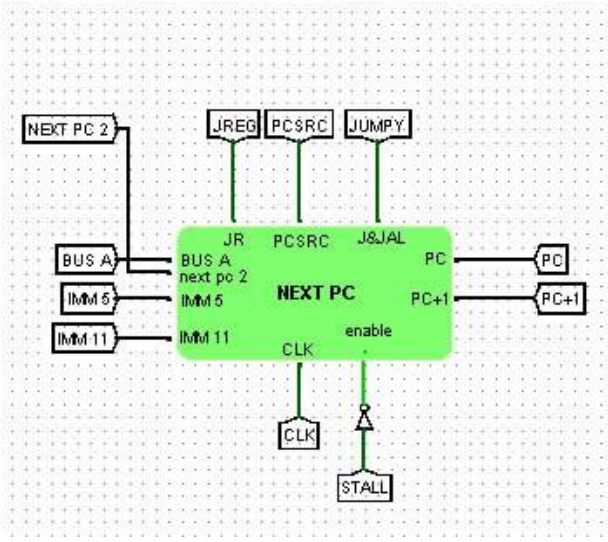


2) The Next PC

we make some changes in the next pc circuit:

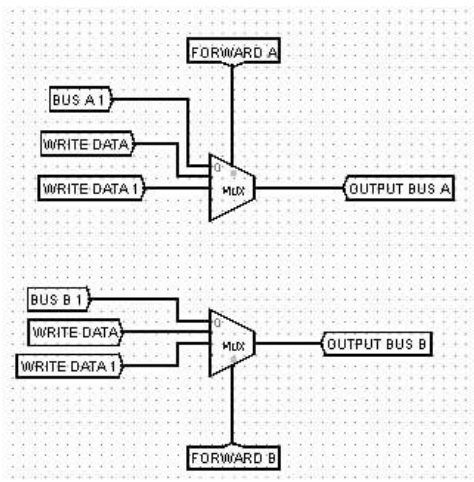
.firstly,we change the target address of the branch and jump instructions instead of adding pc with sign extend of IMM 5 we add the next pc 2 (from the ID/EX)with sign extend of IMM 5.

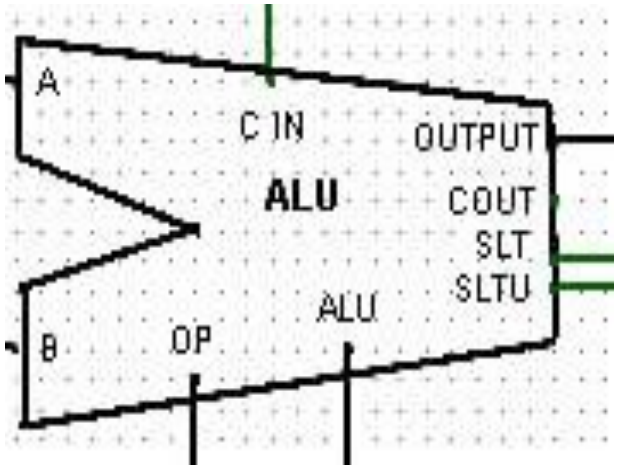
secondly, we replace the constant 1 at the enable of the pc register with the invert of the stall signal to make the pc retains its value and does not change it when there is a stall.



3) ALU Unit

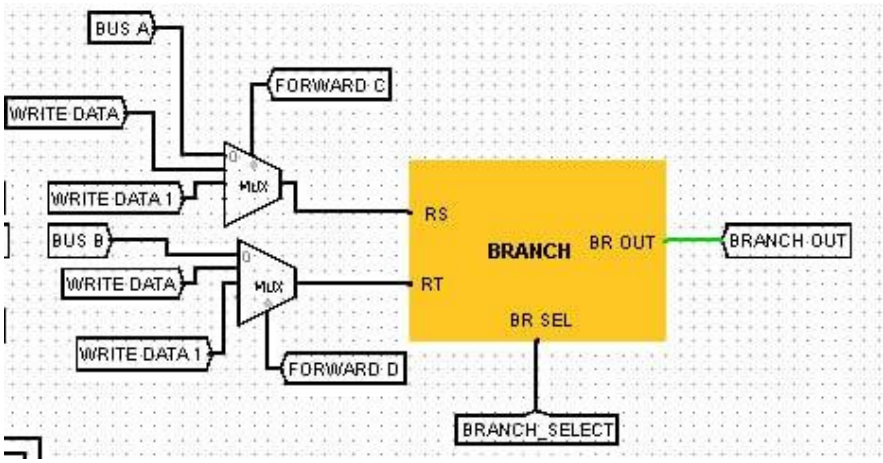
we the ALU circuit does not have any changes from inside drom the one in the single cycle because we have a seperate branch circuit from phase 1 and we do not change the slt circuit and still in the execution but we solve that problem through frowarding DATA MEMORY





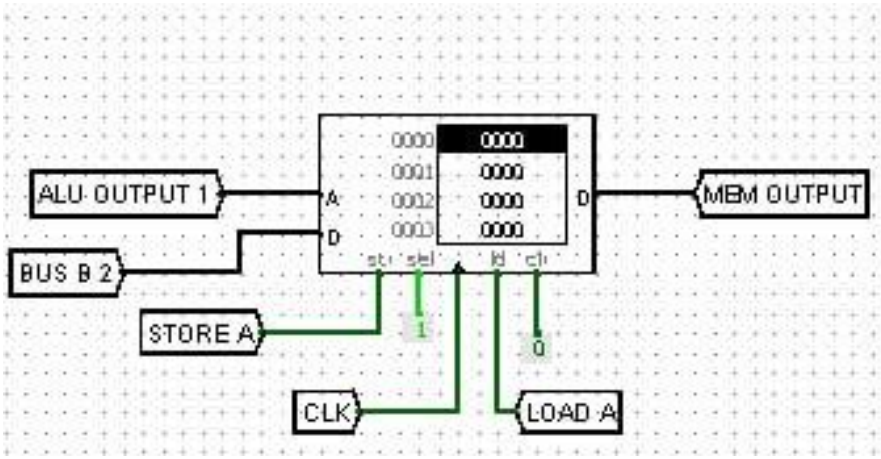
4) BRANCH Circuit

we do not have changes from the one in the single cycle only we use forwarding to determine the right value that we need to be compared to be able to know if the branches instructions are taken or nor taken



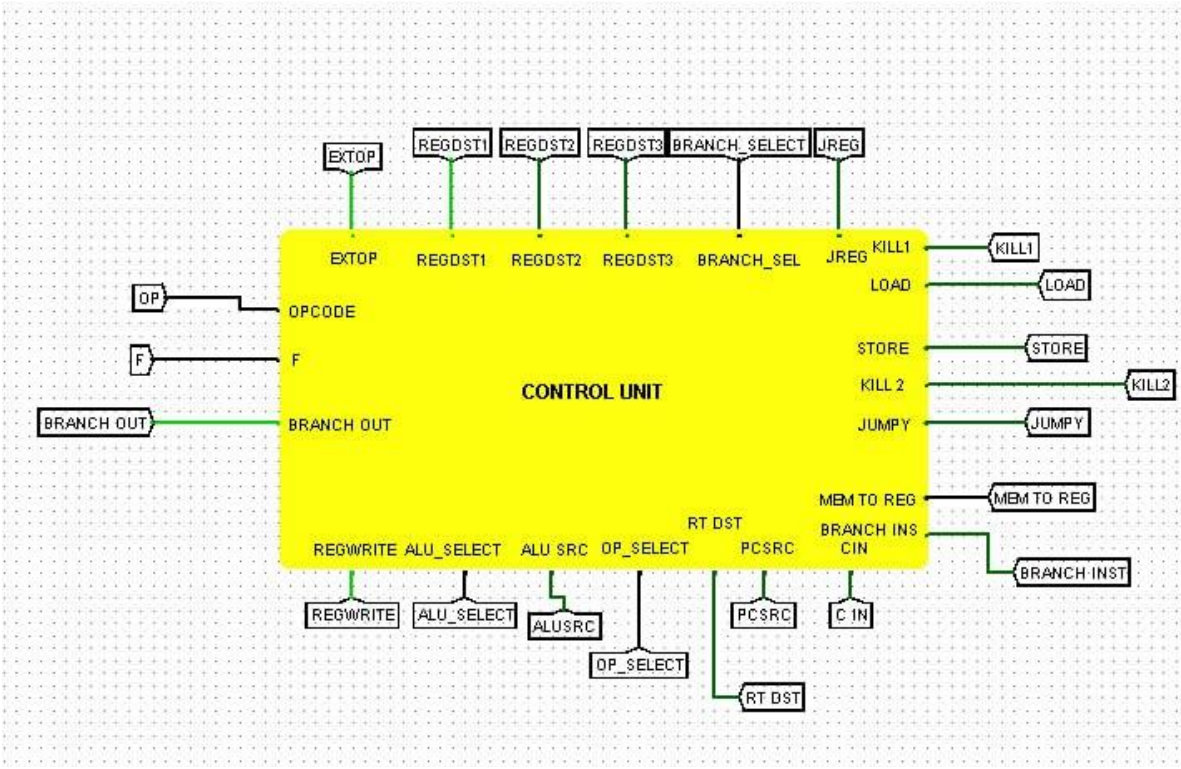
5) DATA MEMORY

we do not have changes from the one in the single cycle only we use forwarding to determine the right value that we need to be written in the memory for the SW instruction



6) CONTROL Unit

we do not change any signal that generated from the single cycle only we generate some new signals to be able to make the project work with no correctly



- ## KILL 1

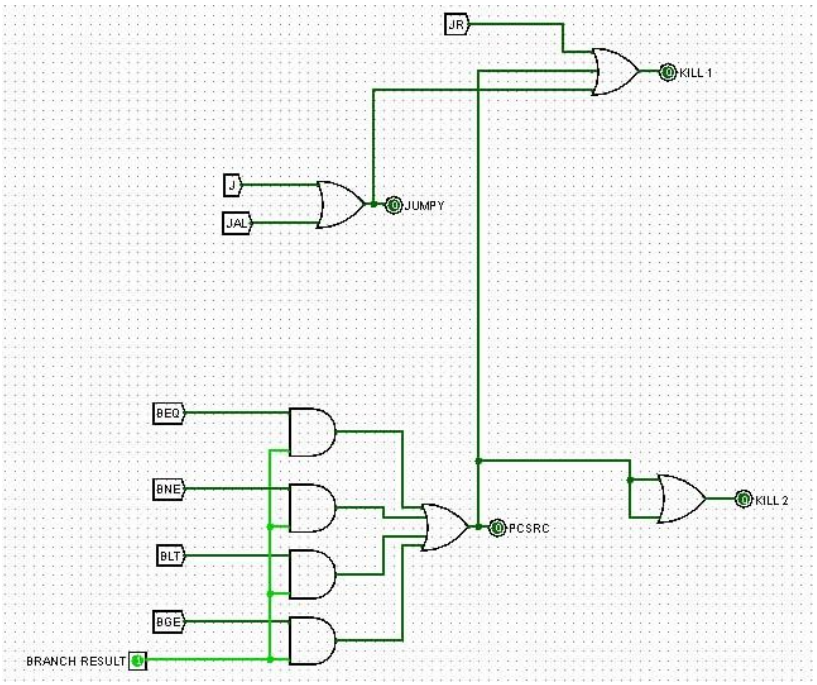
It is a signal that is used for selecting between the instruction out from instruction memory or 0000(flashing). it is used for branch and jump instructions

KILL1 = (PCSRC +J +JR+JAL)

- ## KILL 2

It is a signal that is used for determining if the control signals will pass or delayed and pass 0 . it is used for taken branches.

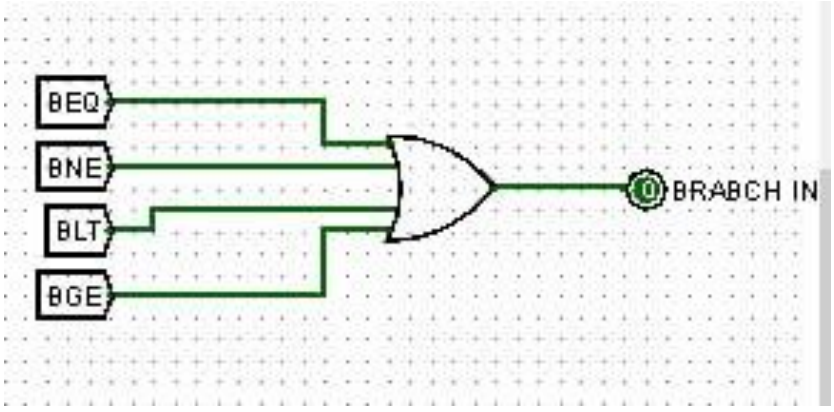
KILL2 = PCSRC



- ## BRANCH INST

It is a signal that is used for determining if there is a branch instruction or not to be able to make forwarding from the branch forward unit

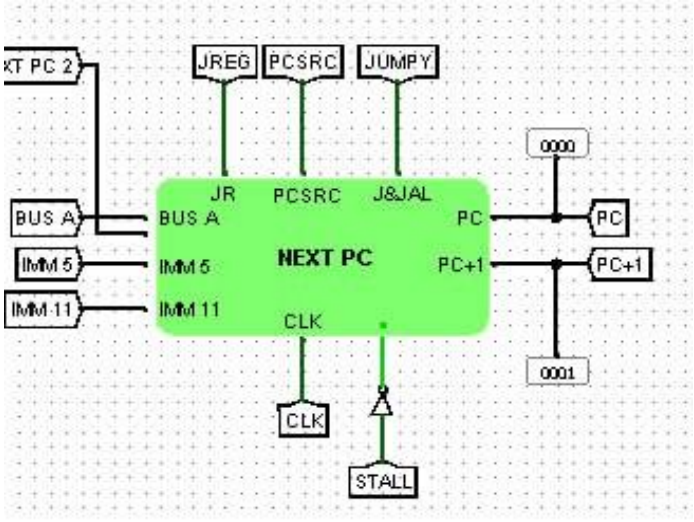
BRANCH INST = (BEQ+BNE+BLT+BGE)



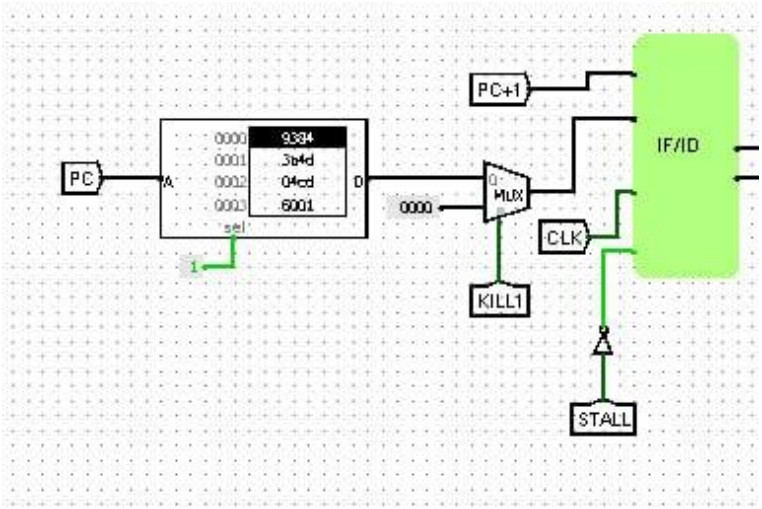
❖ The DATA PATH

1) NEXT PC

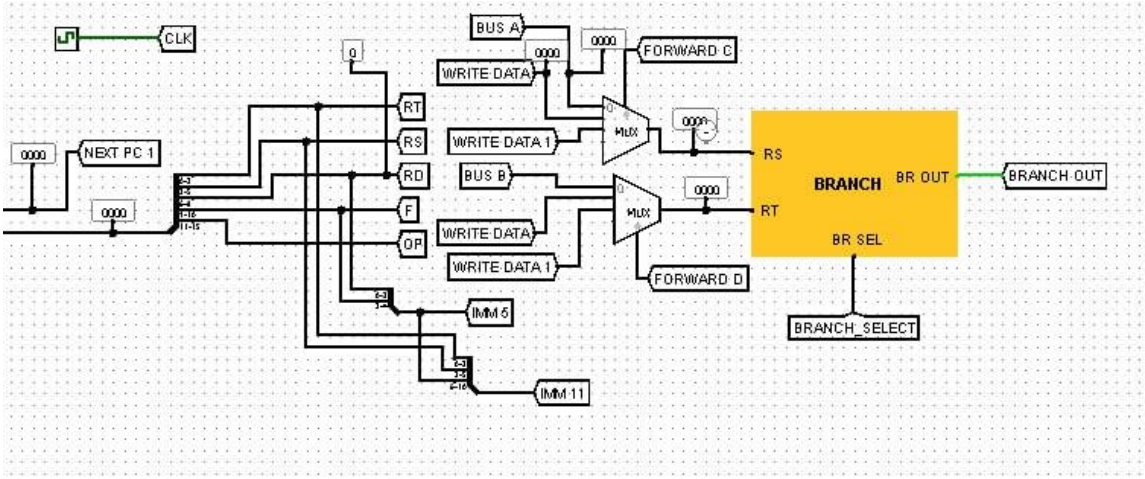
At first we'll talk about the next pc , We add a signal to make a stall come from oring of two signal stalls from the forward unit and a BR stall from BRforward unit and another data path is not changed about phase one and the other changes we've mentioned before



- Secondly we would talk about fetching stage where take the pc to fetch the require the instruction and the enter it on a mux that its signal is kill 1 to choose between 0 or the fetched instruction then we'll save it into register IF|ID with pc +1

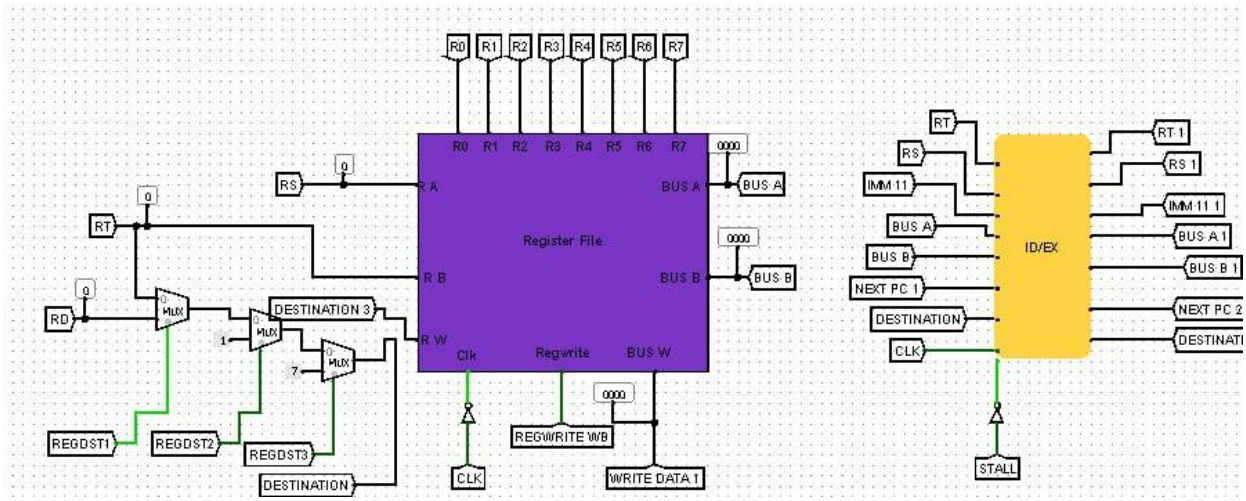


- we'll talk about the first part of the second stage the decoding stage where divide each part of the instruction of the fetched instruction and then we would take the forward element to compare it into the branch unit



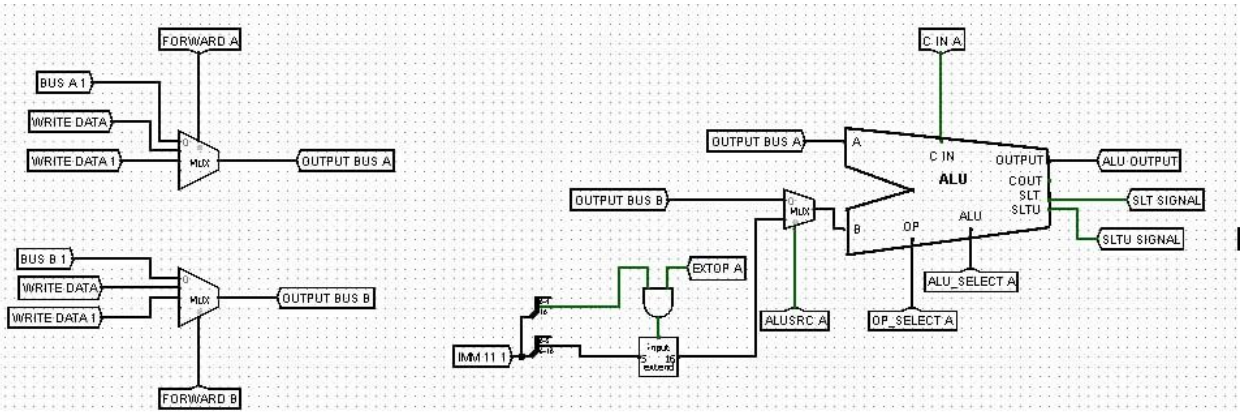
2) REGISTER FILE

The second part is register file where we choose our buses A and b and w but you should note that we make register write is what we return from the write back stage and the destination register is stored in each stage and then we save some buses needed in the next stage into the register ID|EX to use it later

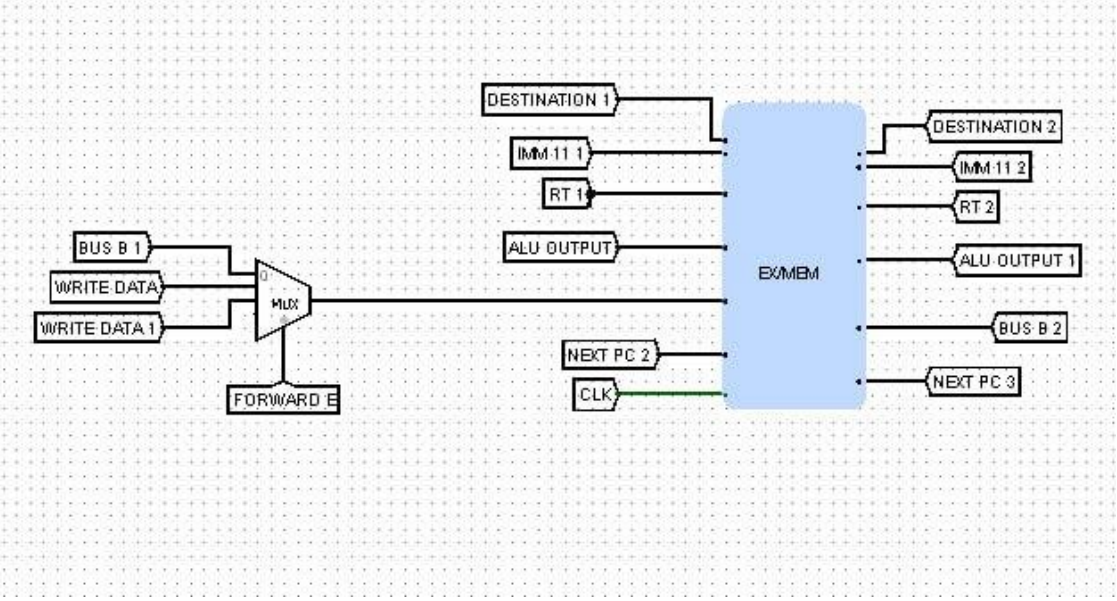


3) ALU Unit (Executing Stage)

- The third stage is executing stage where we execute our instruction but firstly we need to choose which element we would use as an input to the control signal by the forwarding mux as selector to choose between the the basic bus or what comes from the other stages memory and write back and the other part is memory where we take the right data bus to the ALU and execute the operation and get our output

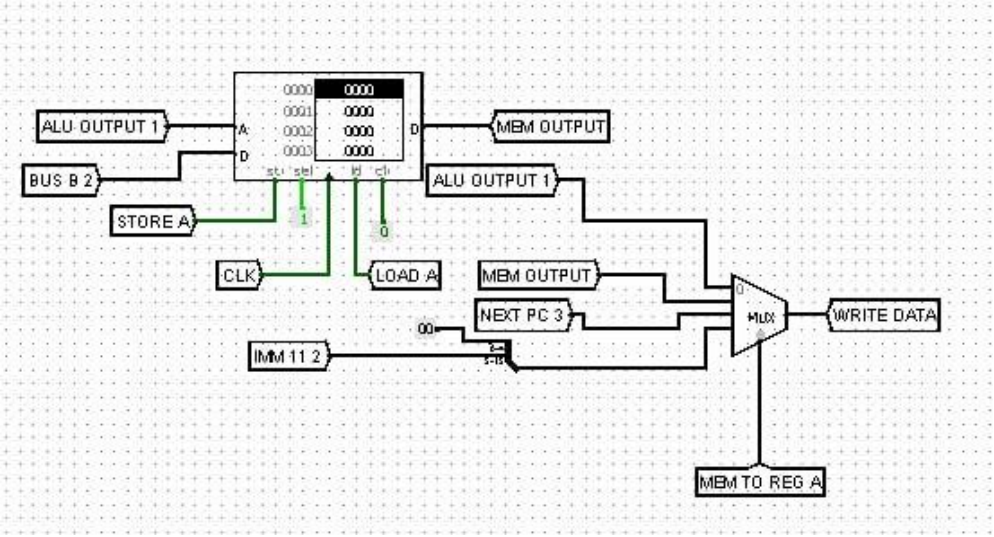


- The last part of this stage is the forwarding from the memory and the write back for store instruction and saving the other buses into the pipeline register

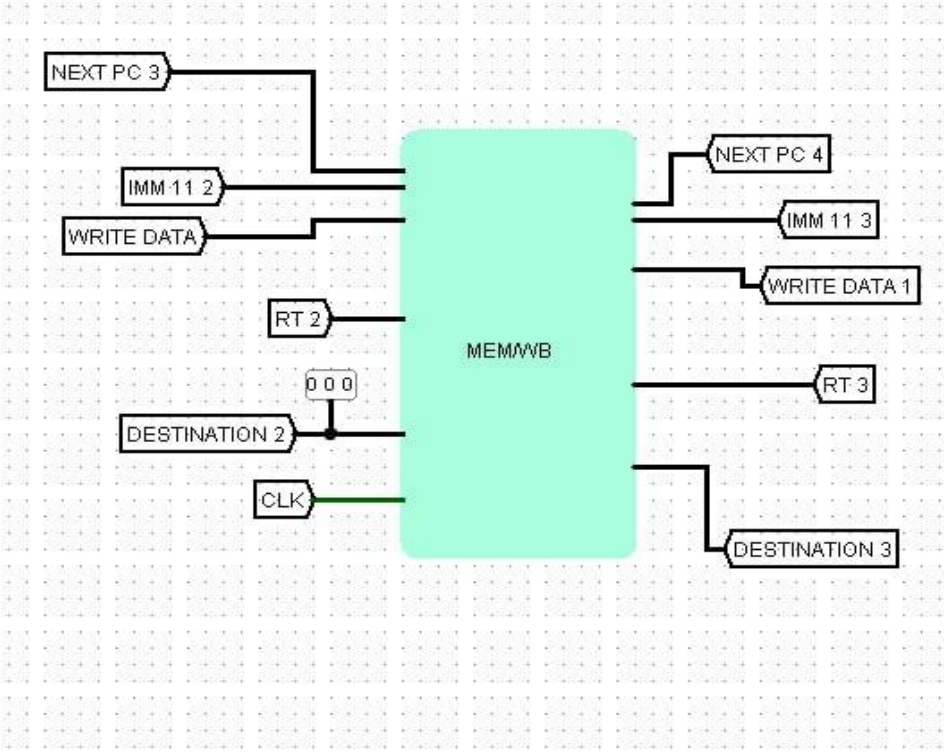


4) DATA MEMORY (Memory Stage)

- The fourth stage is memory stage where we take the output of the alu and target some address in the data memory and take the output muxed with alu output and the next pc and imm11 with 00 for lui to be on write data



- The last stage is write back stage where we take the output in the last stage and save it and them take them in the next clock cycle to use like write data 1 for bus w and destination 3 for targeting register in the register file



❖ Test Code

We will talk about the most important cases in our test code:

1) we can see in this instruction that register 5 is available in the previous instruction in the execution stage so we would need to forward from the MEM stage to the execution stage

```
Addi $5, $1,13

Xor $3, $1, $5
```

2) there's two case or even more in this instruction the first one is we need to forward to the branch register 6 the second one is we need to flash in case of the branch taken the next instruction or exciting the next instruction in case of the branch not taken

```
Slt $6, $2, $3
Beq $6, $0, L1
Add $2, $1, $2
```

3) In this instruction we need to flash the next because this branch is always taken and that what we put into our consideration

```
Beq $0, $0, L2
L1: Sw $4, 0($0)
```

4) we'll need her to forward register 3 from the mem stage to the execution stage as it was made in the previous instruction

```
Sll $3, $2, 6
ROR $6, $3, 3
```

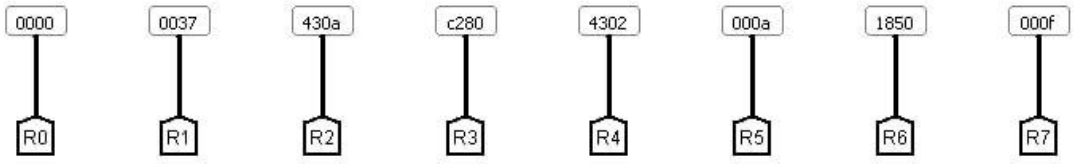
5) In this instructions we will flash the next instruction because the branch is always taken

```
beq $0,$0,-1
Func: or $5, $2, $3
```

6) In this instruction we need to forward to mem stage as we need to store it and that what we put into our consideration for store forward as you can see in the signal forward E to forward from mem stage

```
And $4, $2, $3
Sw $4, 0($0)
```

Here are the final results of the registers .the results are similar compared to the single cycle processor results:



❖ Team Working

- **Tasks per Name**

- *Osama Ahmed*
  Control unit
  Branch Circuit
  Control Unit signals
  Pipeline Registers
  Stall circuit
- *Belal Soliman Abd Elhalem*
  Register File
  Alu Arithmetic & Logic Unit
  Instructions Truth Table
  Forward Unit
  Pipeline Data Path
- *Mohamed Ahmed Saleh*
  Next PC
  Instruction Memory
  Data Memory
  Branch Forward Unit
  Pipeline Control Registers

- Common Tasks

Data Path

Testing all components

Testing each Instruction

Organizing the Report

- **Team Meetings**

During the project we made some meetings to discuss the project some were online and some were offline but here what we did in each meeting and what we discuss and how our project has been completed and the problems we met and how we solved :

- the first meeting was offline and through this meeting each one gave information about the project to complete our view about the project.
- the second meeting after collecting all information we started to distribute tasks to each one of us to complete and rearrange all blocks of the project to see how it works together to complete each part of the program.
- for sometimes we were working separately to accomplish our tasks but sometimes we ask each other the problem each one met to solve.
- then we started some online meetings to combine our tasks we implemented seperately and discuss about each part together through zoom meeting the meeting was sometimes taking 45 minutes to complete to 1.5 hour.
- we met some problems but we succeeded to solve such as the problem of test code to assemble , problems with microsoft word , the problem of time managing and some other problems.
- but we finally completed the project before the deadline