الجامعة اللبنانية الأميركية
Lebanese American University

School of Arts and Sciences

- **Course name: Computer Organization**
- **Course number: CSC320**
- **Section 11**
- **Instructor: Dr. Sanaa Sharafeddine Dawy**
- **Date: 28/04/2022**

Design Phase 2 - CO

Presented by:

Hussien Ali Ahmad	**ID: 202104969**
Bilal Delbani	**ID: 202104998**
Mohammad Al Fallah	**ID: 202105098**
Mohammad Alizzi	**ID: 202104792**

MIPS Sheet:

Opcode(bin)	Opcode(hex)	Format	Function	Name
00000	00_{hex}	*B*	*ASi*	*Assign Immediate*
00001	01_{hex}	*B*	*i++*	*Add Immediate*
00010	02_{hex}	*B*	*i&*	*And Immediate*
00011	03_{hex}	*B*	*i||*	*Or Immediate*
00100	04_{hex}	*B*	*i~*	*Not Immediate*
00101	05_{hex}	*B*	*i/*	*Division Immediate*
01011	$0B_{hex}$	*B*	*i**	*Multiplication Immediate*
01100	$0C_{hex}$	*B*	*i--*	*Subtract Immediate*
10000	10_{hex}	*M*	*++*	*Add*
10001	11_{hex}	*M*	*--*	*Subtract*
10010	12_{hex}	*M*	*LW*	*Load word*
10011	13_{hex}	*M*	*SW*	*Store word*
10100	14_{hex}	*M*	*	*Multiplication*
10101	15_{hex}	*M*	/	*Division*
10110	16_{hex}	*M*	*&*	*And*
10111	17_{hex}	*M*	||	*Or*
11000	18_{hex}	*M*	~	*Not*
11010	$1A_{hex}$	*M*	<<	*Shift Left Logical*
11011	$1B_{hex}$	*M*	>>	*Shift Right Logical*
11100	$1C_{hex}$	*M*	*B=*	*Branch on equal*
11101	$1D_{hex}$	*M*	*B!=*	*Branch on not equal*
11110	$1E_{hex}$	*M*	*B<*	*Branch less than*
11111	$1F_{hex}$	*M*	*B>*	*Branch greater than*

Registers	Number	Use
!Z	**0**	**The constant value 0**
!S0 - !S20	**1 -21**	**Saved Registers**
!T0 - !T20	**22 - 42**	**Temporary Registers**
!A0 - !A16	**43 - 59**	**Address Registers**
!GP	**60**	**Global Pointer**
!SP	**61**	**Stack Pointer**
!FP	**62**	**Frame Pointer**
!RA	**63**	**Register Address**

B – Format:

31 26 20 14 0

Opcode	Var1	Var2	Constant/Address
5	6	6	15

M – Format:

31 26 20 14 8 4 0

Opcode	Var1	Var2	Dest	Shamt	Unused
5	6	6	6	4	5

C-Code:

```
int x = 10;
int y = 0;
int w = 0 << 2;
int j = 5;
for (int i = 1 ; i < x ; i++){
        x += ((i + 2) - (i * 3))/j ;
}
if(x > w){
    w = x|999;
    y = x&w;
    A[1] = y;
}
```

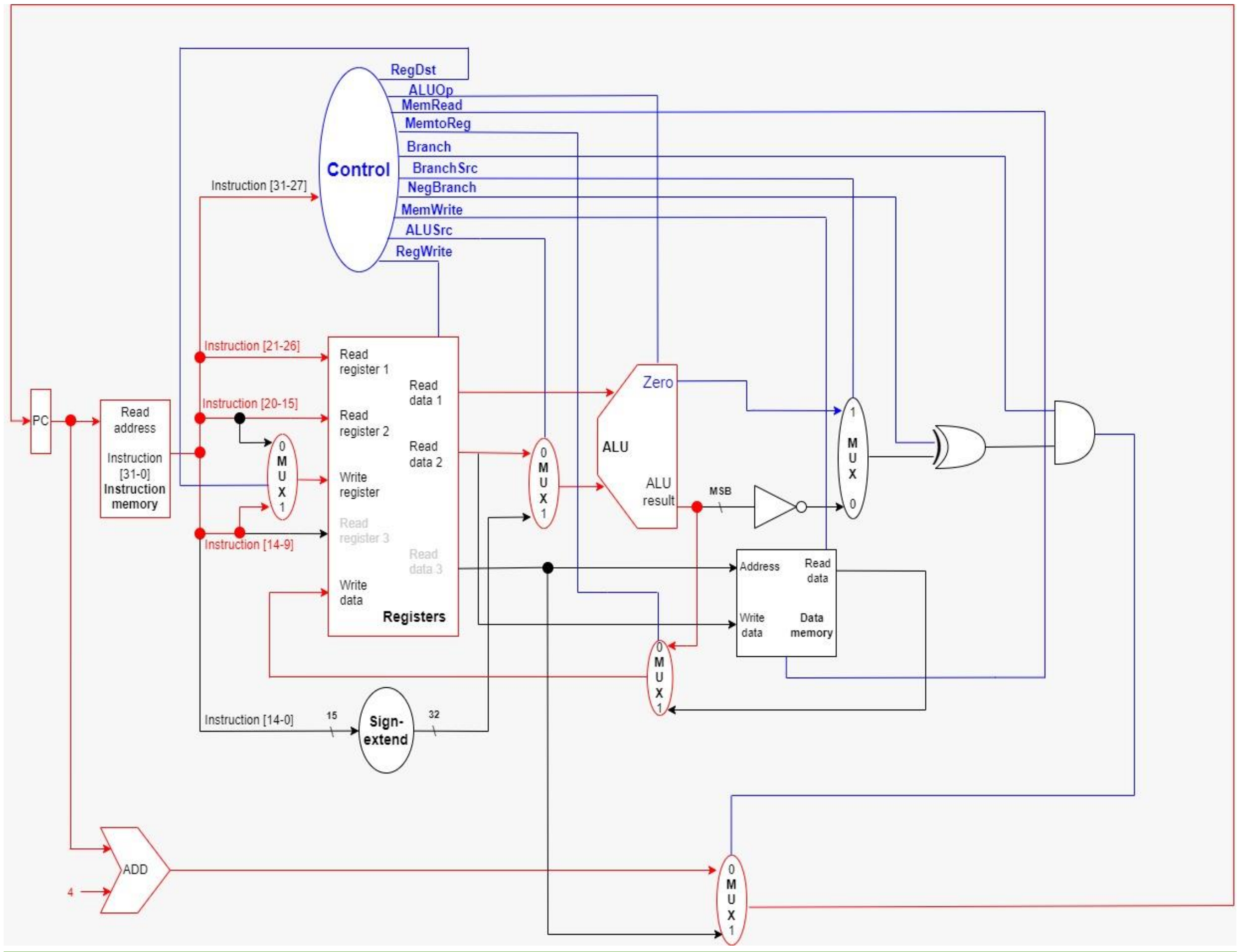
ASSEMBLY CODE:

```
//assume that the address of the array is in register !A0
 ASi !S0, 10   // initialize x=10
 ASi !S1, 0    // initialize y=0
ASi !T0, 0    //initial a temp holding the value to be shifted left logically by 2 bits
<<!S2,!T0,2    //shifting the temp0 by 2 and storing it in S2
ASi !S3, 5    // initialize j=5
 ASi !S4, 1    // initialize i=1

 B = !S1, !Z, Condition  // !S1 is always equal to 0 as we initialize it, this is just to branch to
Condition label.
 Loop:
  i + + !T2, !S4,2 // i+2
 i * !T3, !S4,3 // i*3
 − − !T2, !T2, !T3 // (i+2)-(i*3)
  / !T2 , !T2, !S3 // (i+2)-(i*3)/j
  + + !S0, !S0, !T2 //x+=(i+2)-(i*3)/j
   i + !S4, !S4,1 // i++
Condition: B <  !S4, !S0, Loop //check if i<x branch to Loop
B< !S0,!S2, Exit // if x<w exits else if x>w continues with code and enters the code in the if
statement (addToArray)
addToArray:
                    i| | !S2, !S0,999 // w = x or 999
                    & !S1, !S0, !S2 // y = x and w
                    i + !T4, !A0,4 //adding to get the full address of the array at index 1
                    SW !S1, !T4 //storing the value of y in index 1 of the array
 Exit: ...
```

Note: For the Datapath, the lines highlighted in red are those important to the Datapath, and the others we don't care about or has no significant role in the Datapath of the specific type of instruction.

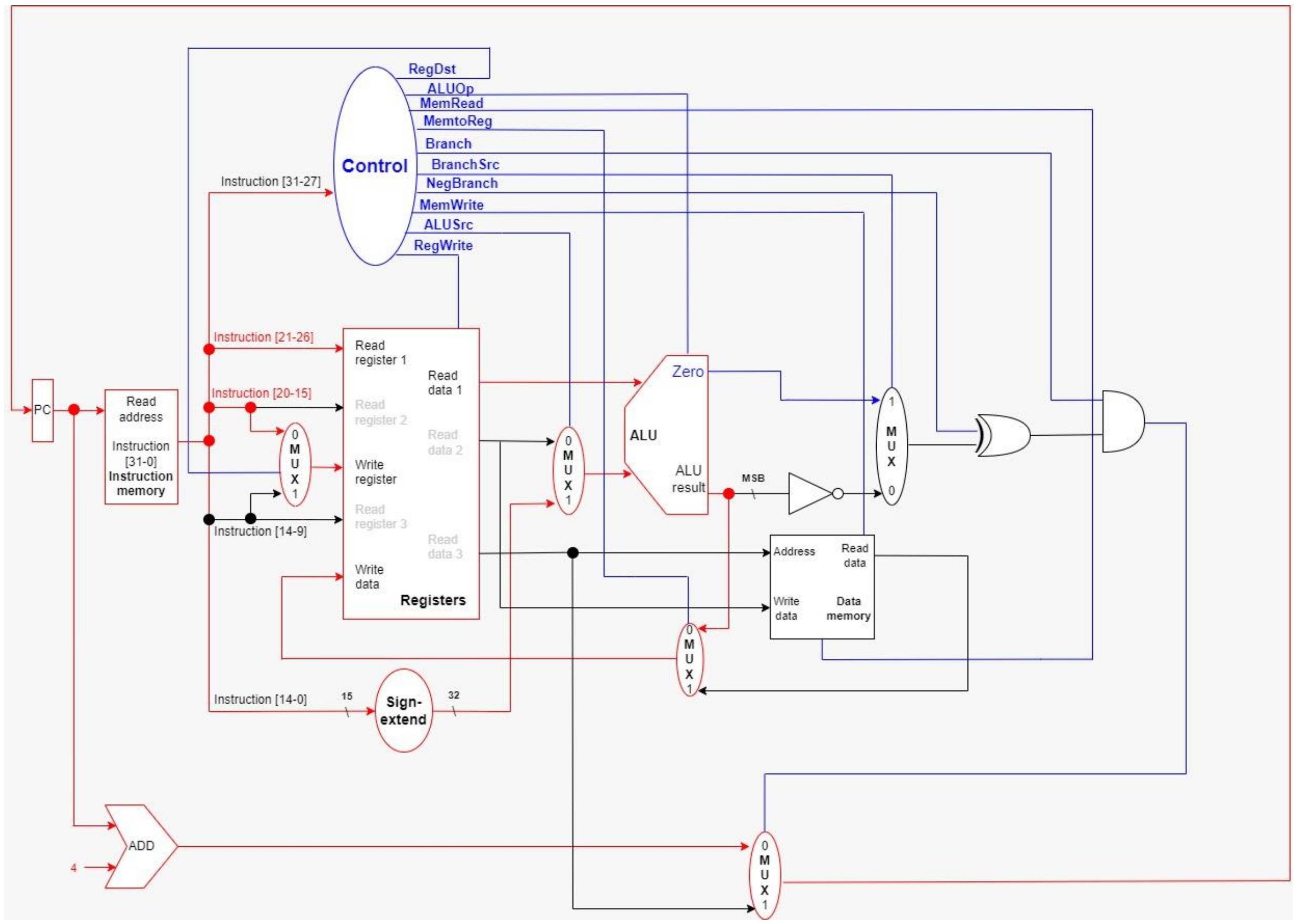
Arithmetic Datapath:



Control:

- **RegDst: 1**
- **ALUOp: X**
- **MemRead: 0**
- **MemtoReg: 0**
- **Branch: 0**
- **BranchSrc: 1 (Don't Care)**
- **NegBranch: 1 (Don't Care)**
- **MemWrite: 0**
- **ALUSrc: 0**
- **RegWrite: 1**
- **Shifting: 0**

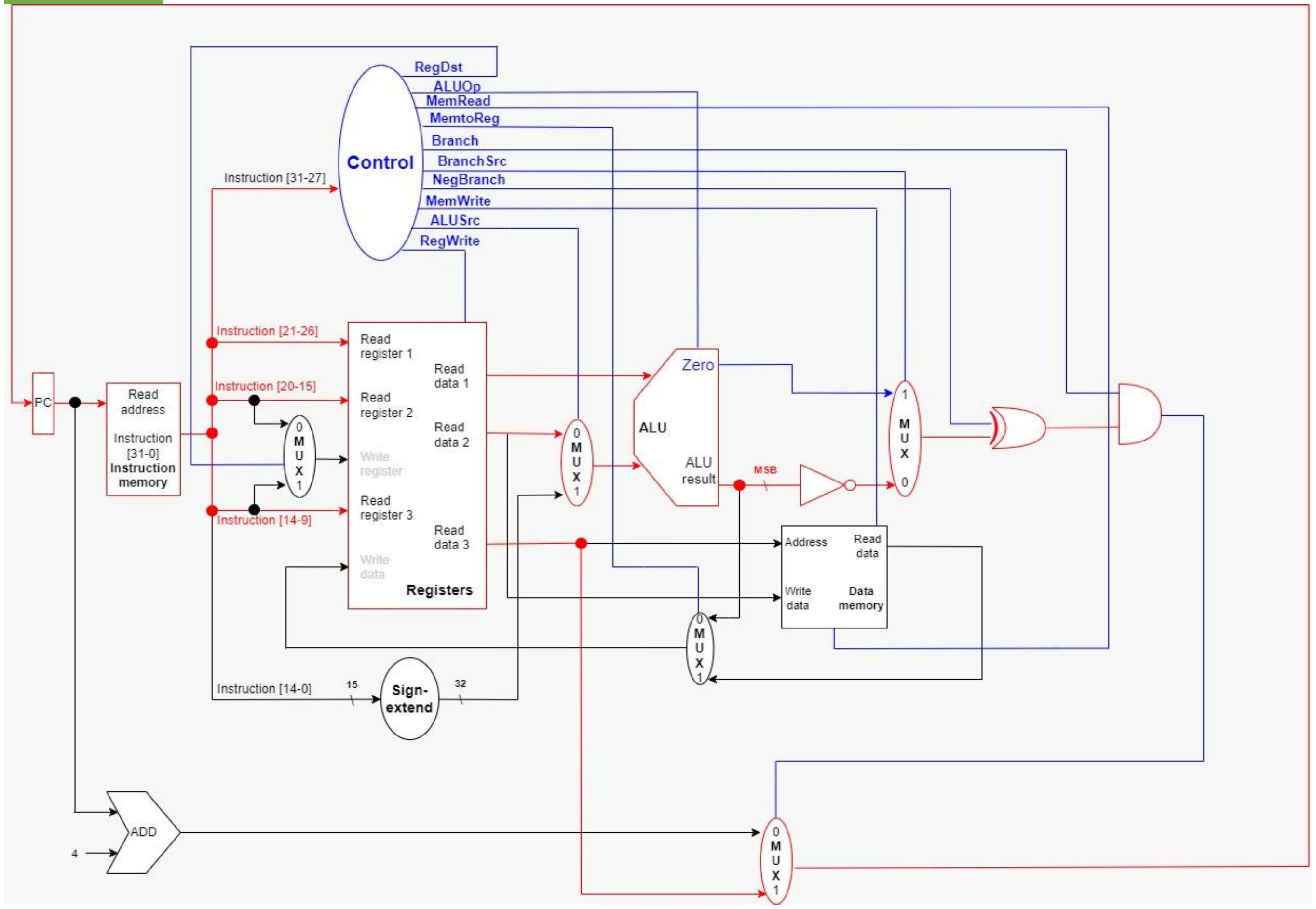
Arithmetic Immediate:



Control:

- **RegDst: 0**
- **ALUOp: X**
- **MemRead: 0**
- **MemtoReg: 0**
- **Branch: 0**
- **BranchSrc: 1 (Don't Care)**
- **NegBranch: 1 (Don't Care)**
- **MemWrite: 0**
- **ALUSrc: 1**
- **RegWrite: 1**
- **Shifting: 0**

Branch:



Legend: MSB is Most Significant Bit

Control:

- **RegDst: 0 (Don't Care)**
- **ALUOp: 001**
- **MemRead: 0**
- **MemtoReg: 0 (Don't Care)**
- **Branch: 1**
- **BranchSrc: X (depends, below the full explanation, briefly if beq or bnq it is 1, while for bgt or blt it is 0)**
- **NegBranch: X (depends, below the full explanation, briefly if bnq or blt it is 1,while for beq or bgt it is 0)**
- **MemWrite: 0**
- **ALUSrc: 0**
- **RegWrite: 1**
- **Shifting: 0**

Full explanation for the Branch Datapath:

Branch Instructions : B=, B!=, B<, B>.

B< and B>

After fetching the instruction,

The control signals will be:

- RegDst=X // we don't need a write register
- ALUOp= 001 // subtract
- MemRead = X // we are not interesting if it reads from memory or not
- MemtoReg = X // we don't need to write in any register
- Branch=1 // it is branch instruction
- BranchSrc = 0 // choose the most significant bit
- NegBranch equals '1' for (B<) and '0' for (B>)
- MemWrite= X // we are not interesting if it writes to memory or not
- ALUSrc = 0 // to choose read data 2 and not the sign extends
- RegWrite = X //we are not interesting for writing in a register

Var1 will be at (Read register 1), Var2 will be at (Read register 2), Dest will be at (Read register 3), then, **Read data 2** is chosen (ALUSrc) and the ALU will subtract the data from **Read data 1 and Read data 2** (ALUOp), the cases that we have are (+/-/0), if we are working on **B<** and the **condition is true** then it should branch to its target; so the answer of "sub" should be **negative** and we choose the **negation of the Most Significant bit (1→0)** with the "MUX (BranchSrc=0 since it is not **branch equal or branch not equal**)" (//note that the other input of MUX is 0 since the answer is not zero so the condition is false) and the **NegBranch** will give '1' since it is **B<** → 1 xor 0 = 1 and then the **Branch** is always 1 for all branches instructions → 1 & 1= 1 → therefore the new address given with **read register 3** will be chosen as a target address for the **PC ;** now if the **condition** of **B<** is **not true** then the negation of the most significant bit (0→1) then 1 xor 1=0 → 0&1 will be zero and the value of PC will be its initial value incremented by 4. Now we shift for **B>** if the **condition is true** then it should branch to its target; so the answer of "sub" should be **positive** and we choose the **negation of the Most Significant bit (0→1)** with the "MUX (BranchSrc=0 since it is not **branch equal or branch not equal**)" (//note that the other input of MUX is 0 since the answer is not zero so the condition is false) and the **NegBranch** will give '0' since it is not **B<** → 0 xor 1 = 1 and then the **Branch** is always 1 for all branches instructions → 1 & 1= 1 → therefore the new address given with **read register 3** will be chosen as a target address for the **PC ;** now if the **condition** of **B>** is **not true** then the negation of the most significant bit (1→0) then 0 xor 0=0 → 0&1 will be zero and the value of PC will be its initial value incremented by 4.

B= and B!=

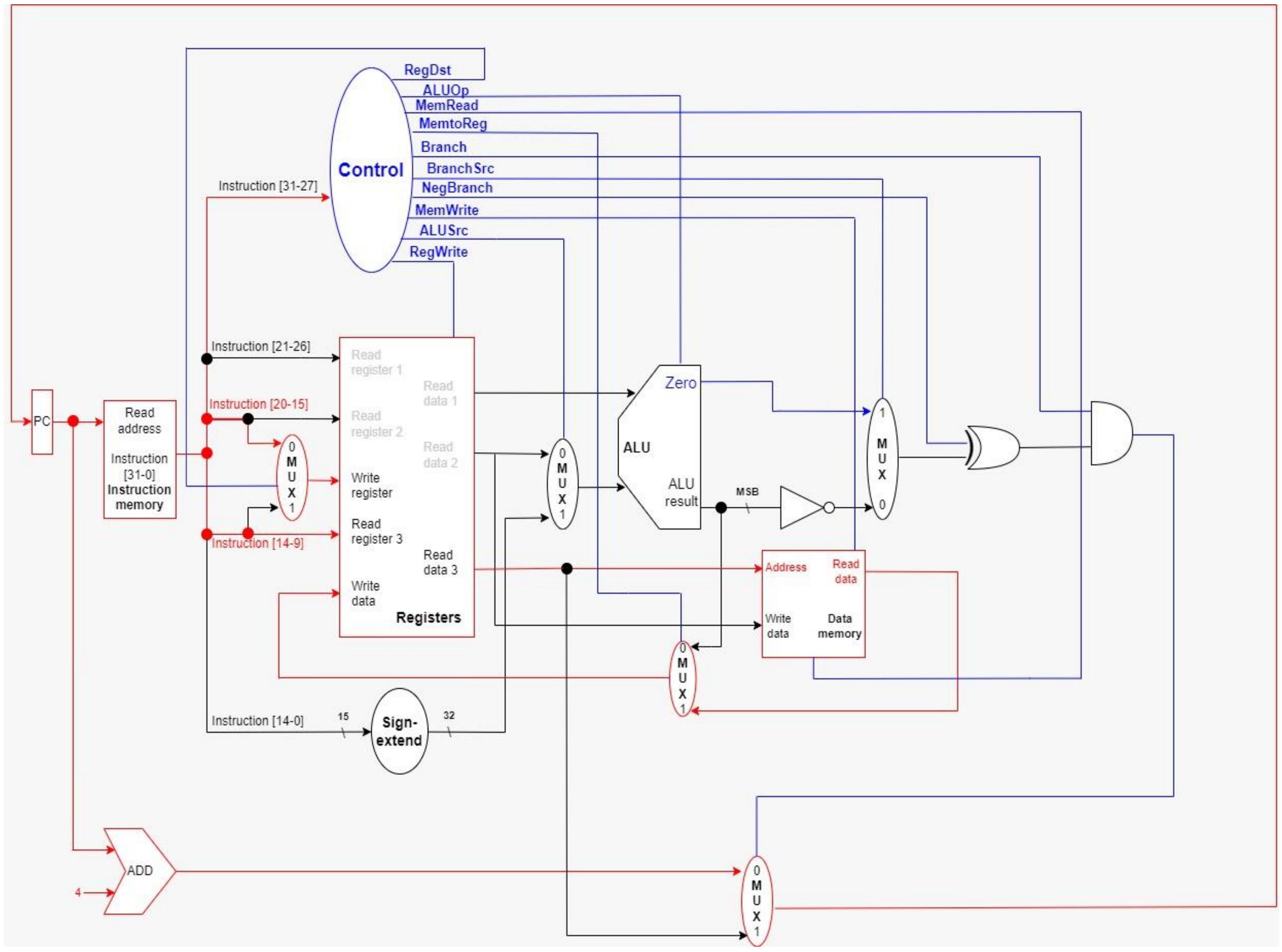
After fetching the instruction,

The control signals will be:

- RegDst=X // we don't need a write register
- ALUOp= 001 // subtract
- MemRead = X // we are not interesting if it reads from memory or not
- MemtoReg = X // we don't need to write in any register
- Branch=1 // it is branch instruction
- BranchSrc = 1 // choose the side where it is compared with 'zero'
- NegBranch equals '1' for (B!=) and '0' for (B=)
- MemWrite= X // we are not interesting if it writes to memory or not
- ALUSrc = 0 // to choose read data 2 and not the sign extend
- RegWrite = X //we are not interesting for writing in a register

Var1 will be at (Read register 1), Var2 will be at (Read register 2), Dest will be at (Read register 3), then, **Read data 2** is chosen (ALUSrc) and the ALU will subtract the data from **Read data 1 and Read data 2** (ALUOp), the cases that we have are (+/-/0), if we are working on **B=** and the **condition is true** then it should branch to its target; so the answer of "sub" should be **zero** and we choose the **zero side (since answer is zero and it is compared with zero so condition is true → 1)** with the "MUX (BranchSrc=1 since it is **branch equal or branch not equal**)" (//note that the other input of MUX is the negation of the most significant bit and we are not interested with its value as MUX will always choose the zero side in our cases) and the **NegBranch** will give '0' since it is **B=** → 0 xor 1 = 1 and then the **Branch** is always 1 for all branches instructions → 1 & 1= 1 → therefore the new address given with **read register 3** will be chosen as a target address for the **PC ;** now if the **condition** of **B=** is **not true** then we choose the **zero side (since answer is not zero(+/-) and it is compared with zero so condition is false → 0)**then 0 xor 0=0 → 0&1 will be zero and the value of PC will be its initial value incremented by 4. Now we shift for **B!=** if the **condition is true** then it should branch to its target; so the answer of "sub" should be **not zero** and we choose the **zero side (since answer is not zero and it is compared with zero so condition is false→ 0)**with the "MUX (BranchSrc=1 since it is **branch equal or branch not equal**)")" (//note that the other input of MUX is the negation of the most significant bit and we are not interested with its value as MUX will always choose the zero side in our cases) and the **NegBranch** will give '1' since it is **B!=** → 0 xor 1 = 1 and then the **Branch** is always 1 for all branches instructions → 1 & 1= 1 → therefore the new address given with **read register 3** will be chosen as a target address for the **PC ;** now if the **condition** of **B!=** is **not** then we choose the **zero side (since answer is zero and it is compared with zero so condition is true → 1)** then 1 xor 1=0 → 0&1 will be zero and the value of PC will be its initial value incremented by 4.

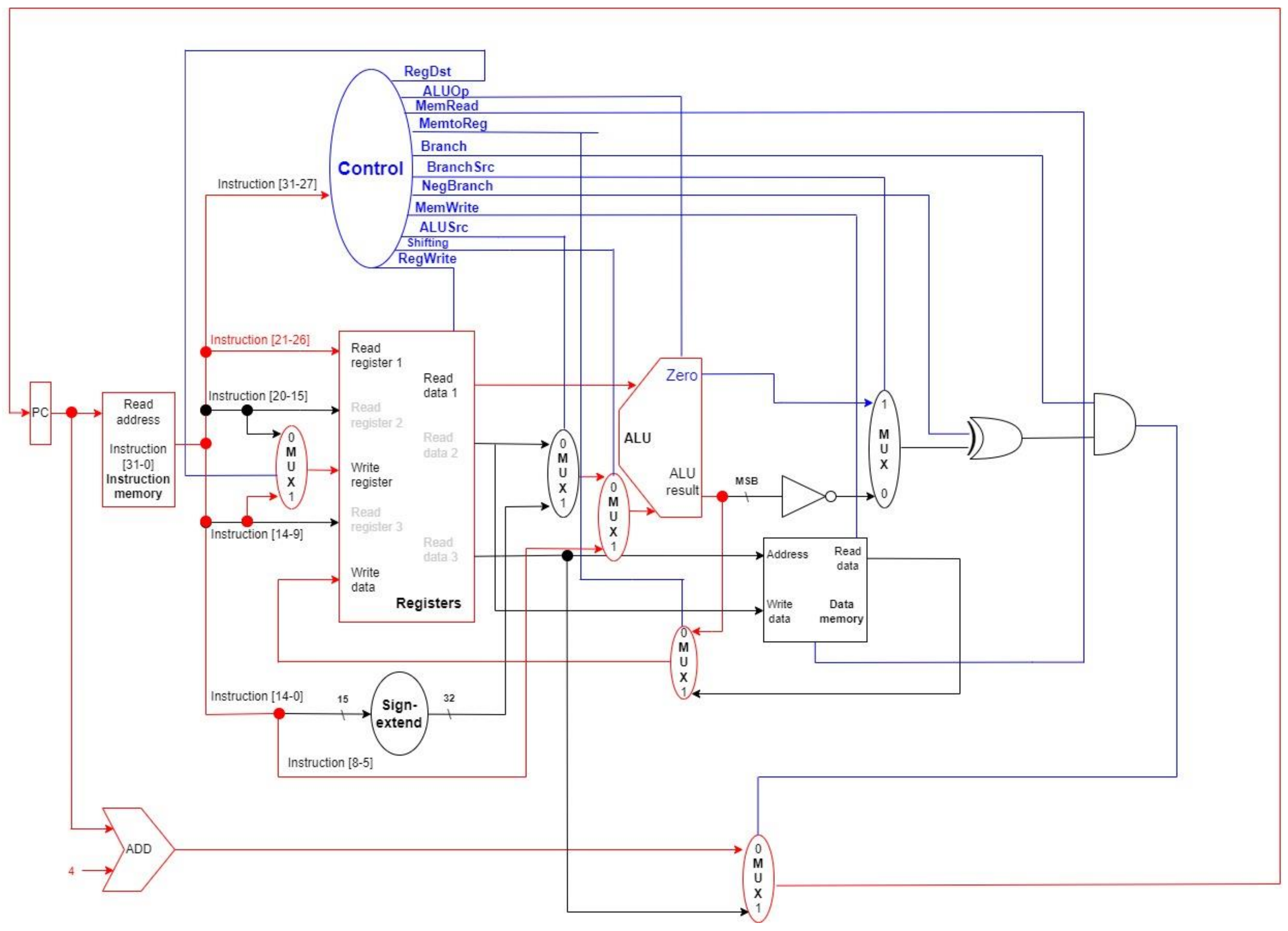
Load:



Control:

- **RegDst: 0**
- **ALUOp: X (Don't Care, it will not pass the ALU)**
- **MemRead: 1**
- **MemtoReg: 1**
- **Branch: 0**
- **BranchSrc: 1 (Don't Care)**
- **NegBranch: 1 (Don't Care)**
- **MemWrite: 0**
- **ALUSrc: 0 (Don't Care, will not pass the ALU)**
- **RegWrite: 1**
- **Shifting: 0**

Shift Left Logical:



Control:

- **RegDst: 1**
- **ALUOp: 110**
- **MemRead: 0**
- **MemtoReg: 0**
- **Branch: 0**
- **BranchSrc: 1 (Don't Care)**
- **NegBranch: 1 (Don't Care)**
- **MemWrite: 0**
- **ALUSrc: 1**
- **RegWrite: 1**
- **Shifting: 1**

Table Of the ALU:

Opcode	ALUOp	Operation	ALU performed Function
00001	000	Add immediate	Addition
00000	000	Assign Immediate	Addition
10000	000	Add	Addition
01100	001	Subtract Immediate	Subtraction
10001	001	Subtract	Subtraction
11100	001	Branch on equal	Subtraction
11101	001	Branch on not equal	Subtraction
11110	001	Branch less than	Subtraction
11111	001	Branch greater than	Subtraction
00010	010	And immediate	and
10110	010	And	and
00011	011	Or immediate	or
10111	011	Or	or
00100	010	Not immediate	not
11000	010	Not	not
00101	100	Division immediate	Division
10101	100	Division	Division
01011	101	Multiplication immediate	Multiplication
10100	101	Multiplication	Multiplication
11010	110	Shift Left Logical	Sll
11011	111	Shift Right Logical	Srl