In the name of God

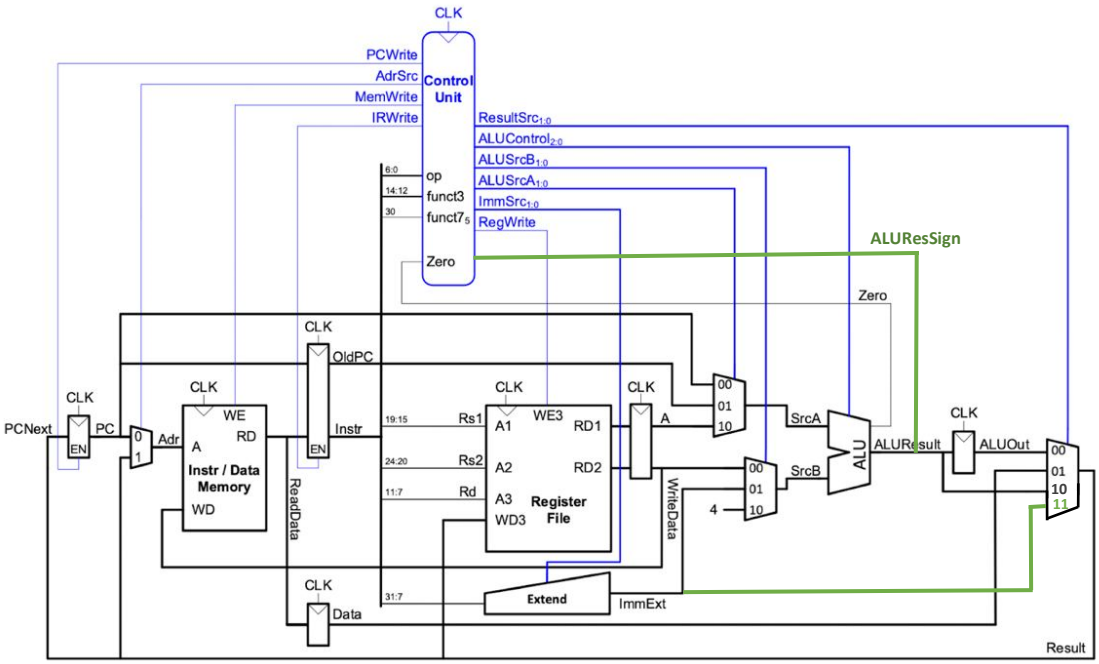
Computer Architecture Computer Assignment 3

Contributors (student number):
Mohammad mahdi Doust mohamadi (810100142)

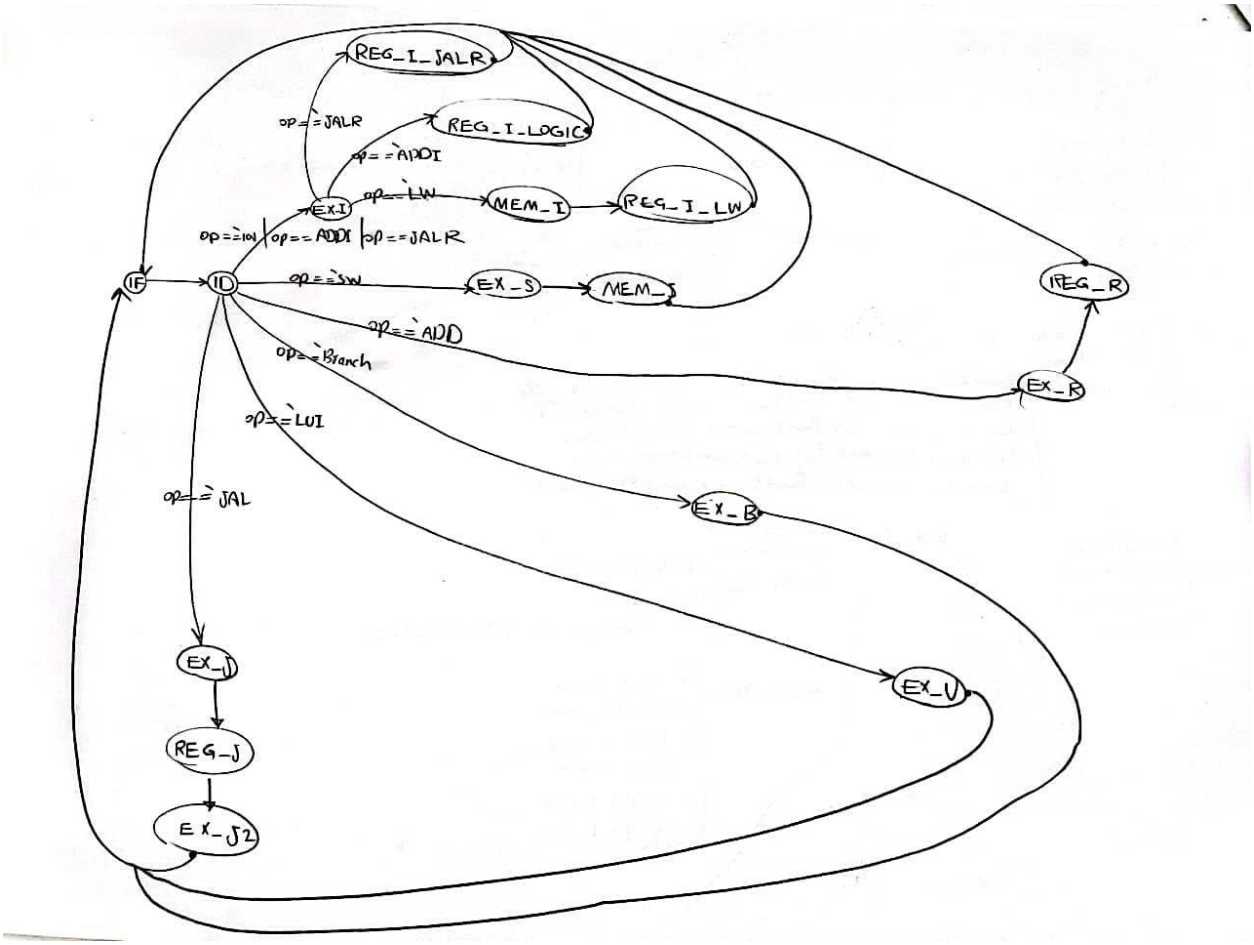
Farnaz Sedaghati (810100177)

Multicycle RISC-V Processor Design

Datapath:



Control signals:



States :

Defualt state:

```
default: {AdrSrc, PCWrite, MemWrite, IRWrite, ALUSrcA,
          ALUSrcB, ALUControl, ResultSrc, RegWrite, ImmSrc} = 16'b0;
```

Instruction fetch (IF):

```
IF : begin
        AdrSrc = 1'b0;
        IRWrite = 1'b1;
        ALUSrcA = 2'b00;
        ALUSrcB = 2'b10;
        ALUControl = 3'b000;
        ResultSrc = 2'b10;
        PCWrite = 1'b1;
end
```

Instruvtion Decode (ID):

```
ID : begin
        ALUSrcA = 2'b01;
        ALUSrcB = 2'b01;
        ALUControl = 3'b000;
        ImmSrc = 3'b010;
end
```

Execution B-type (EX-B):

```
EX_B : begin
        ALUSrcA = 2'b10;
        ALUSrcB = 2'b00;
        ALUControl = 3'b001;
        ResultSrc = 2'b00;
        PCWrite = (funct3 == 3'b000 && Zero == 1'b1) ? 1'b1 ://beq branches
                  (funct3 == 3'b001 && Zero == 1'b0) ? 1'b1 :  //bne branches
                  (funct3 == 3'b100 && ALUResSign == 1'b1) ? 1'b1 :  //blt branches
                  (funct3 == 3'b101 && ALUResSign == 1'b0) ? 1'b1 : 1'b0; //bge branches
end
```

Execution R-type (EX-R):

```
EX_R : begin
        ALUSrcA = 2'b10;
        ALUSrcB = 2'b00;
        ALUControl = (funct3 == 3'b000 && funct7 == 7'b0000000) ? 3'b000 : //ADD
                     (funct3 == 3'b000 && funct7 == 7'b0100000) ? 3'b001 ://SUB
                     (funct3 == 3'b010 && funct7 == 7'b0000000) ? 3'b101 : //SLT
                     (funct3 == 3'b110 && funct7 == 7'b0000000) ? 3'b011 : //OR
                     (funct3 == 3'b111 && funct7 == 7'b0000000) ? 3'b010 : 3'bx; //AND
end
```

Execution S-type (EX-S):

```
EX_S : begin
        ImmSrc = 3'b001;
        ALUSrcA = 2'b10;
        ALUSrcB = 2'b01;
        ALUControl = 3'b000;
end
```

Execution I-type (EX-I):

```
EX_I : begin
        ImmSrc = 3'b000;
        ALUSrcA = 2'b10;
        ALUSrcB = (op == `JALR && funct3 == 3'b000) ? 2'b10 : // JALR
                  (op == `LW && funct3 == 3'b010) ? 2'b01 : // LW
                  (op == `ADDI) ? 2'b01 : 2'bxx; // ADDI - ORI - SLTI - XORI -

        ALUControl = (op == `ADDI && funct3 == 3'b000) ? 3'b000 : //ADDI
                     (op == `JALR && funct3 == 3'b000) ? 3'b000 ://JALR
                     (op == `LW && funct3 == 3'b010) ? 3'b000 : //LW
                     (op == `ORI && funct3 == 3'b110) ? 3'b011 : //ORI
                     (op == `XORI && funct3 == 3'b100) ? 3'b100 : //XORI
                     (op == `SLTI && funct3 == 3'b010) ? 3'b101 : 3'bx; //SLTI
end
```

Execution J-type (EX-J): saving return address

```
EX_J : begin
        ALUSrcA = 2'b01;
        ALUSrcB = 2'b10;
        ALUControl = 3'b000;
end
```

Execution J-type (EX-J2):

```
EX_J2 : begin
        ImmSrc = 3'b011;
        ALUSrcA = 2'b01;
        ALUSrcB = 2'b01;
        ALUControl = 3'b000;
        ResultSrc = 2'b10;
        PCWrite = 1'b1;
end
```

Execution U-type (EX-U):

```
EX_U : begin
        ImmSrc = 3'b100;
        ResultSrc = 3'b011;
        RegWrite = 1'b1;
```

Memory S-type(MEM-S):

```
MEM_S : begin
        ResultSrc = 2'b00;
        AdrSrc = 1'b1;
        MemWrite = 1'b1;
end
```

Memory I-type(MEM-I):

```
MEM_I : begin
        ResultSrc = 2'b00;
        AdrSrc = 1'b1;
end
```

Registerfile R-type(REG-R):

```
REG_R : begin
        ResultSrc = 2'b00;
        RegWrite = 1'b1;
end
```

Registerfile I-type -> LW (REG-I-LW):

```
REG_I_LW : begin
        ResultSrc = 2'b01;
        RegWrite = 1'b1;
end
```

Registerfile I-type -> Logic (REG-I-LOGIC):

```
REG_I_LOGIC : begin
        ResultSrc = 2'b00;
        RegWrite = 1'b1;
end
```

Registerfile I-type -> Jalr (REG-I-JALR):

```
 REG_I_JALR : begin
         ResultSrc = 2'b00;
         RegWrite = 1'b1;
 end
```

Registerfile J-type (REG-J):

```
REG_J : begin
        ResultSrc = 2'b00;
        RegWrite = 1'b1;
end
```

TESTING:

To find biggest number of an array of size 10 :

Following assembly code will find the largest number of the array which are mentioned bellow :

Numbers of array :

10 , 12 , 8 , 9 , 4 , -268435445 , 1 , 162 , 13 , 6

Largest number : 162

```
Assembly Instructions :
00000333 -> addi x6, x0, 0
00000413 -> addi x8, x0, 0
02832393 -> addi x7, x6, 40
00038e63 -> beq x1, x7, 8
02832483 -> lw x9, 40(x6)
009420b3 -> add x0, x8, x9
00008463 -> beq x0, x8, 16
00900433 -> add x8, x8, x9
00430313 -> addi x6, x6, 4
fe5ff06f -> jal x0, -20
```

Through applying this instruction …

1. It initializes registers X6 and X8 to 0 .
2. It sets register X7 to 40.
3. It enters a loop where it performs the following operations:
    - It compares the values in X1 and X7. If they are equal, it skips ahead to the instruction after the loop.
    - It loads a word from memory at an address calculated by adding 40 to the value in X6 , and stores that word in X9.
    - It adds the values in X8 and X9 , and stores the result in X8.
    - It compares X0 (which is always 0) and X8. If they are equal, it skips ahead to the instruction after the loop.
    - It adds the values in X8 and X9 , and stores the result in X8.
    - It increments X6 by 4.
    - It jumps back to the start of the loop.
4. After the loop, it jumps back to an instruction 20 steps before the end of the code.

Finally the largest number found through the instructions would be in register X8 .