Cairo University

Faculty of Engineering

Computer Architecture Project Document

Phase - 1

Team Number : **7**

Team Members : SEC:

1. **Israa Achraf Younis** **11**
2. **Israa Mamdouh** **12**
3. **Sara Ahmed** **27**
4. **Sara Maher** **28**

Table of Contents

Note:

All registers are edge triggered

Instruction Set Architecture:

One operand instructions:

	op type(2bits)	opcode(4bits)	op(3bits)	(6bits)	issue bit(1bit)
NOP	00	0000	-	-	0
SETC	00	0001	-	-	0
CLRC	00	0010	-	-	0
NOT Rdst	00	0011	xxx	-	0
INC Rdst	00	0100	xxx	-	0
DEC Rdst	00	0101	xxx	-	0
OUT Rdst	00	0110	xxx	-	1
IN Rdst	00	0111	xxx	-	1

2 operand instructions:

	Op type(2bits)	opcode(3bits)	op1(3bits)	op2(3bits)	(3bits)	Issue bit(1bit)
MOV Rsrc, Rdst	01	000	xxx	xxx	-	0
ADD Rsrc, Rdst	01	001	xxx	xxx	-	0
SUB Rsrc, Rdst	01	010	xxx	xxx	-	0
AND Rsrc, Rdst	01	011	xxx	xxx	-	0
OR Rsrc, Rdst	01	100	xxx	xxx	-	0
SHL Rsrc, Rdst	01	101	xxx	xxx	-	0
SHR Rsrc, Rdst	01	110	xxx	xxx	-	0

Memory instructions:

	Op type(1bit)	opcode(2bits/ 3bits)	op1(3bits)	op2(3bits)	(6bits)	Issue bit(1bit)
PUSH Rdst	010	000	xxx	-	-	1
POP Rdst	010	001	xxx	-	-	1
LDM Rdst, Imm	010	01	xxx	-	-	1
LDD Rsrc, Rdst	010	10	xxx	xxx	-	1
STD Rsrc, Rdst	010	11	xxx	xxx	-	1

Jump instructions:

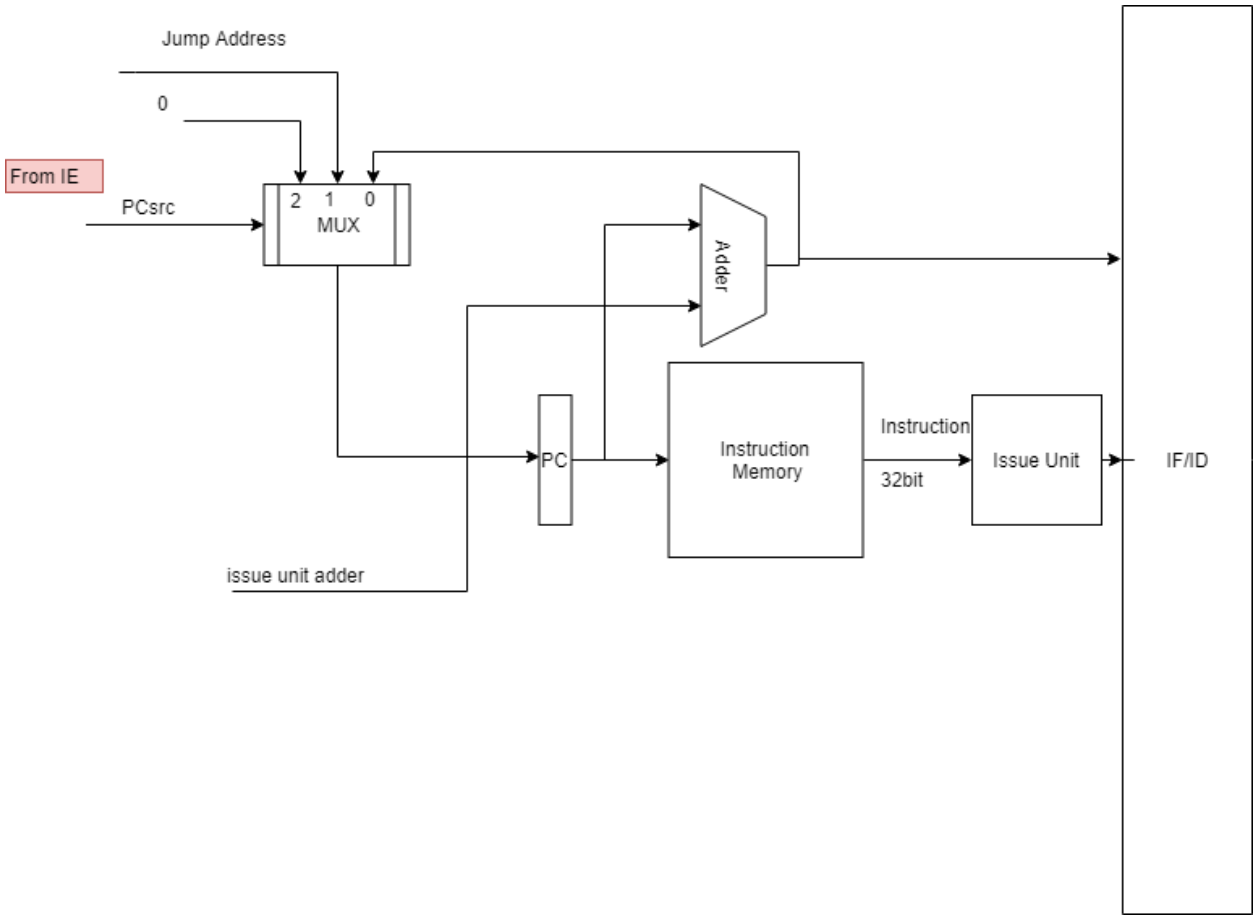
	Op type(2bits/3bits)	opcode(2bits//4 bits)	op1(3bits)	(7bits)	Issue bit(1bit)
JZ Rdst	011	00	xxx	-	0
JN Rdst	011	01	xxx	-	0
JC Rdst	011	10	xxx	-	0
JMP Rdst	011	11	xxx	-	0
CALL Rdst	00	11	xxx	-	1
RET	00	1010	-	-	1
RTI	00	1011	-	-	1

	Op type(2bits)	opcode(4bits)	(9bits)	Issue bit(1bit)
Reset	00	1000	-	1
Interrupt	00	1001	-	1

Pipeline Stages

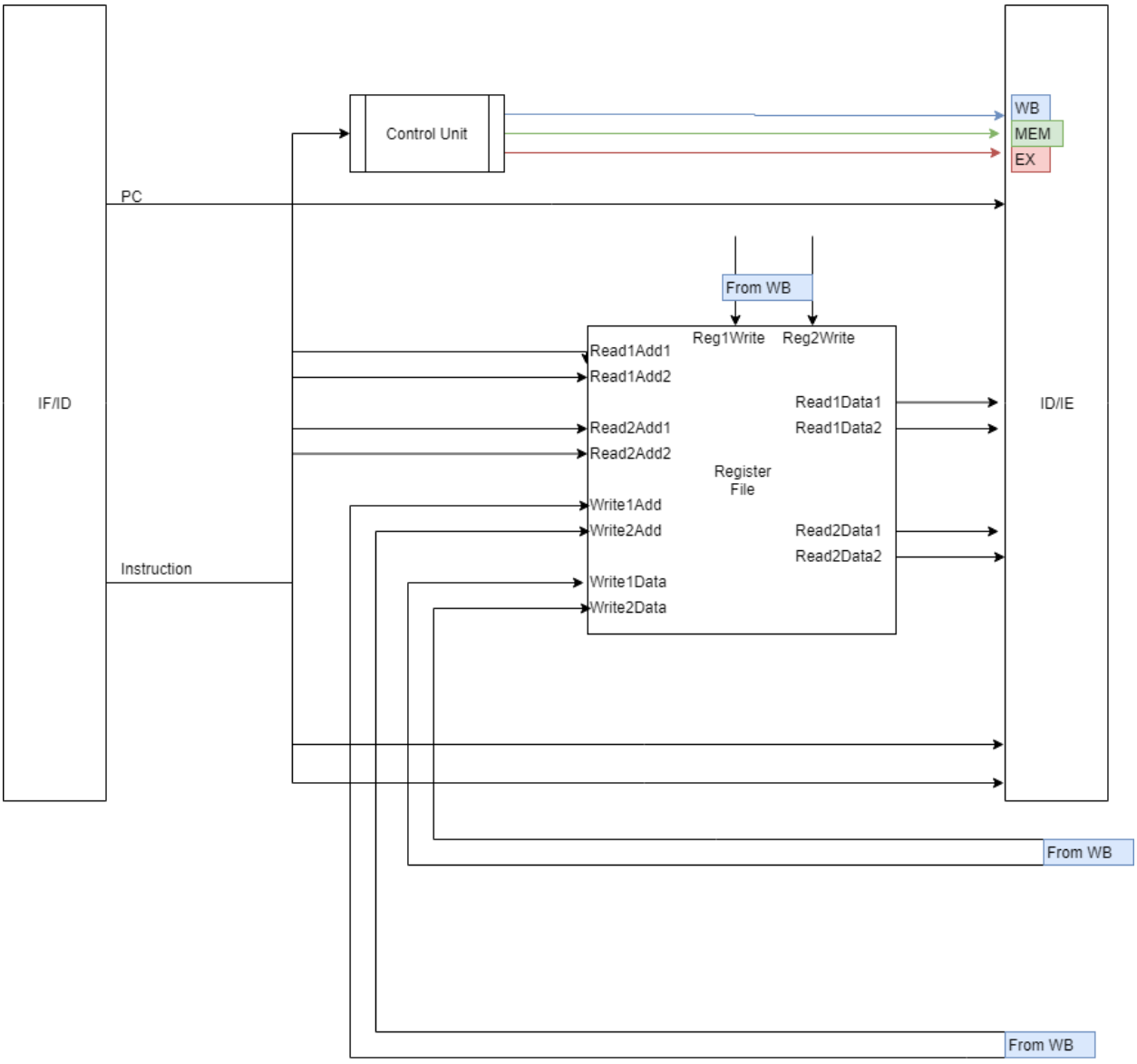
1. Fetch Stage (IF):

In this stage the CPU reads two instructions from the address in the instruction memory [32 bit] whose value is present in the program counter and then update the program counter either increment it by 2 or make it jump to the Jump Address or reset it to 0 .
The issue takes the first instruction [16 bit] and checks if it's LD/STR instruction and store it in the first 16-bit in instruction Reg if not it put NOP instruction,
And also the second instruction [16 bit] and checks if it's ALU/JMP instruction and store it in the second 16-bit in instruction Reg if not it put NOP instruction.

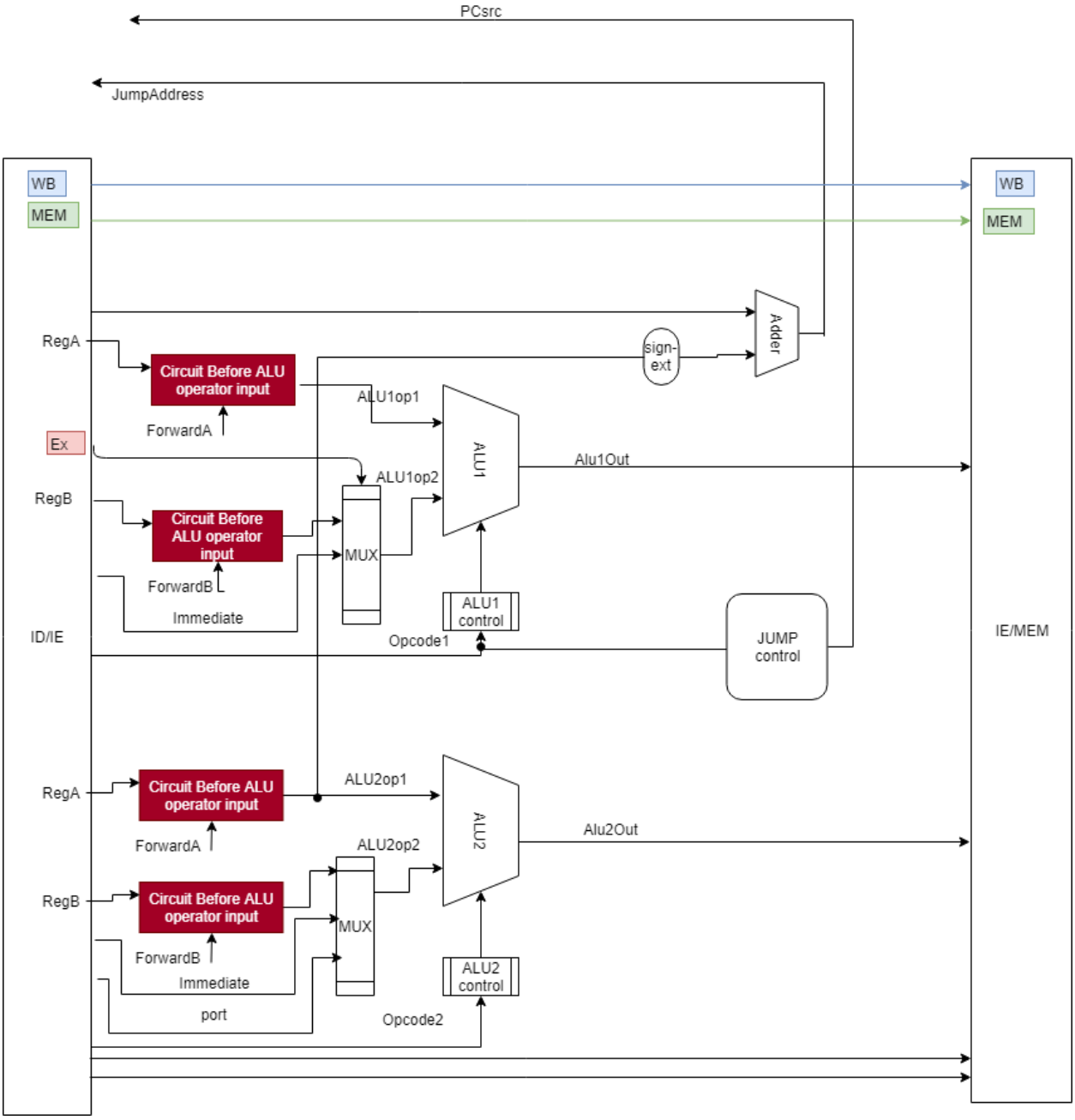


2. Decode Stage (ID):

In this stage, the two instructions are decoded as { LD/STR | ALU/JMP } and the register file is accessed to get the values from the registers used in the instructions and the control unit do operations on the instructions to out the signal which control the flow of each instruction in all upcoming stages

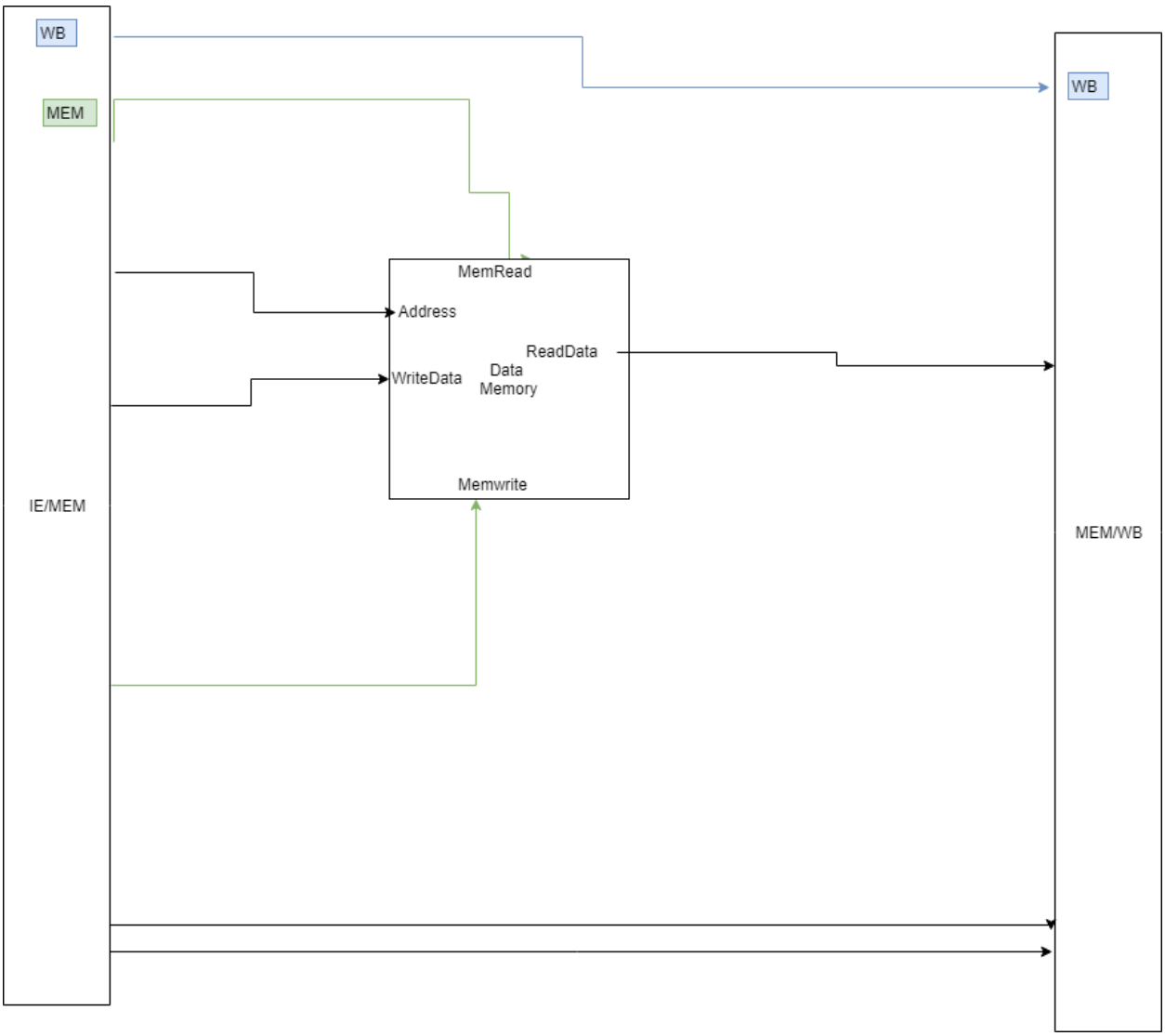


3. Execute Stage (IE):



4. Memory Stage (MEM):

In this stage, if we have load/store instruction memory operand is read or written from/to the memory that is present in the instruction.
MemRead is the enable which indicates that the instruction need to read from the data memory.
Address holds the which address needs to be accessed from the memory.
WriteData is the data which'll be written in the specified address in the Data memory.
MemWrite is the enable which indicates that if the instruction needs to write in the data memory.
ReadData is the fetched data from the data memory.



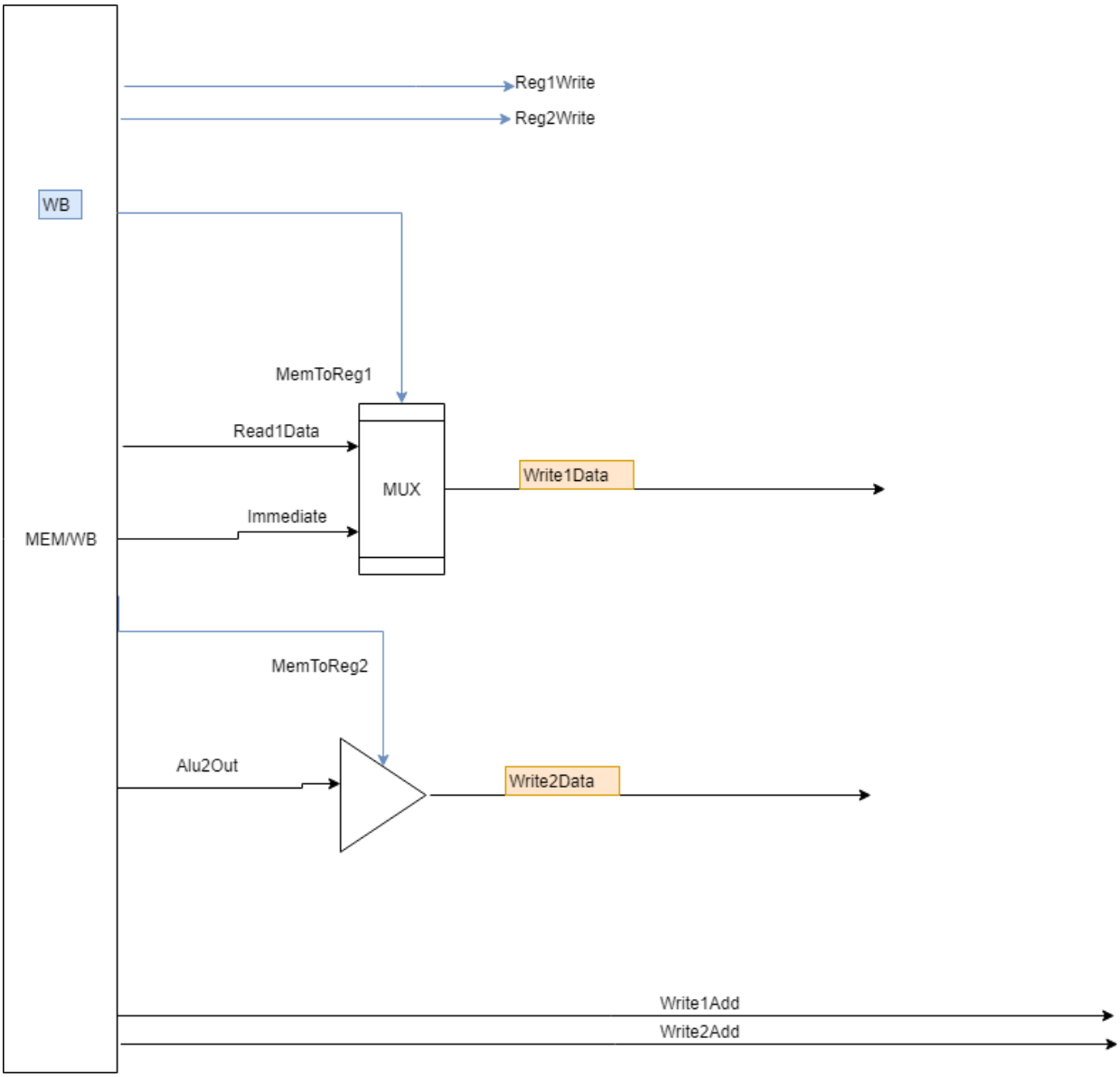
5. Write Back Stage (WB):

In this stage, computed/fetched values are written back to the registers present in the instructions.

Reg1Write,Reg2Write are the enabled of dst registers in the registers file.

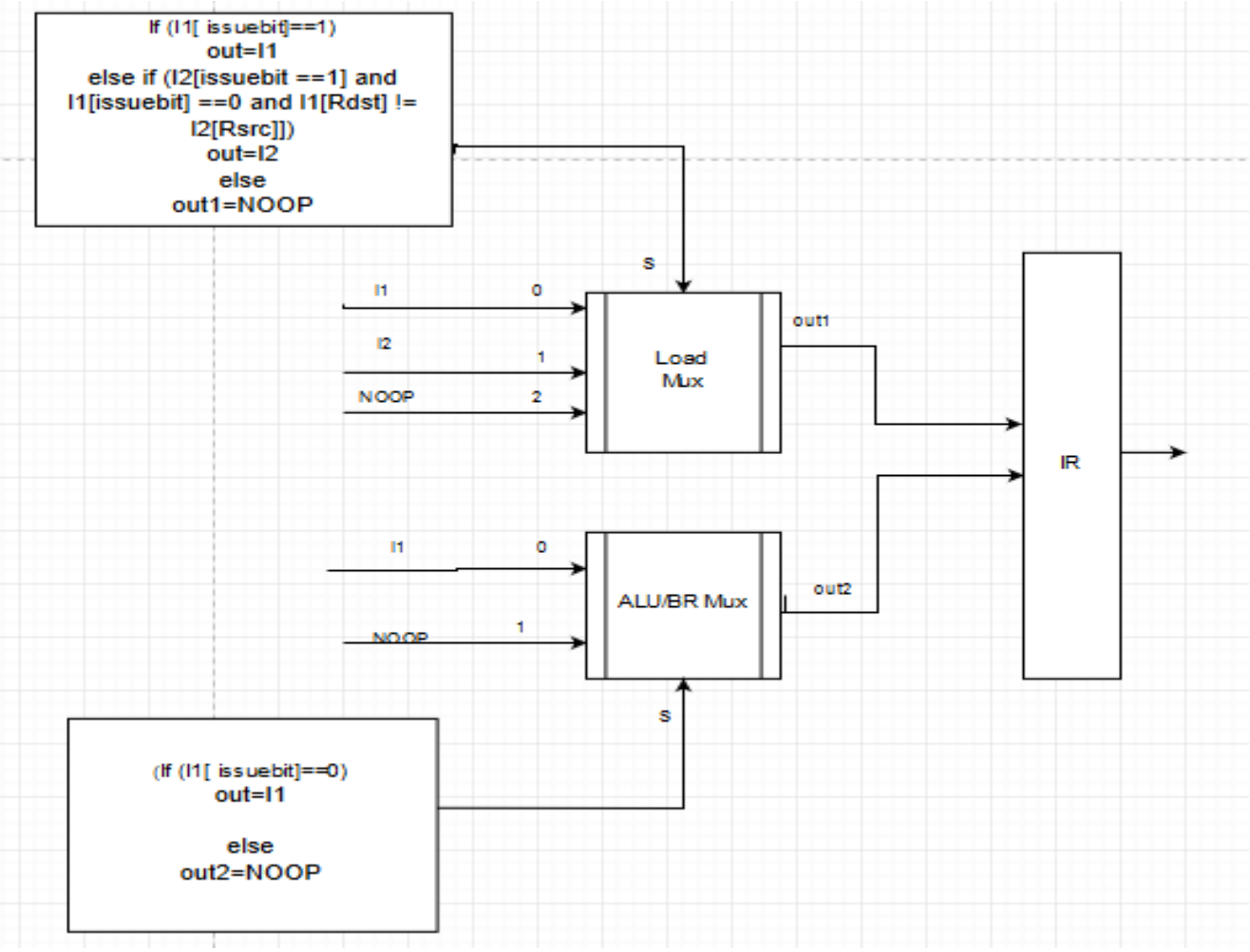
Write1Data,Write2Data are the computed/fetched values.

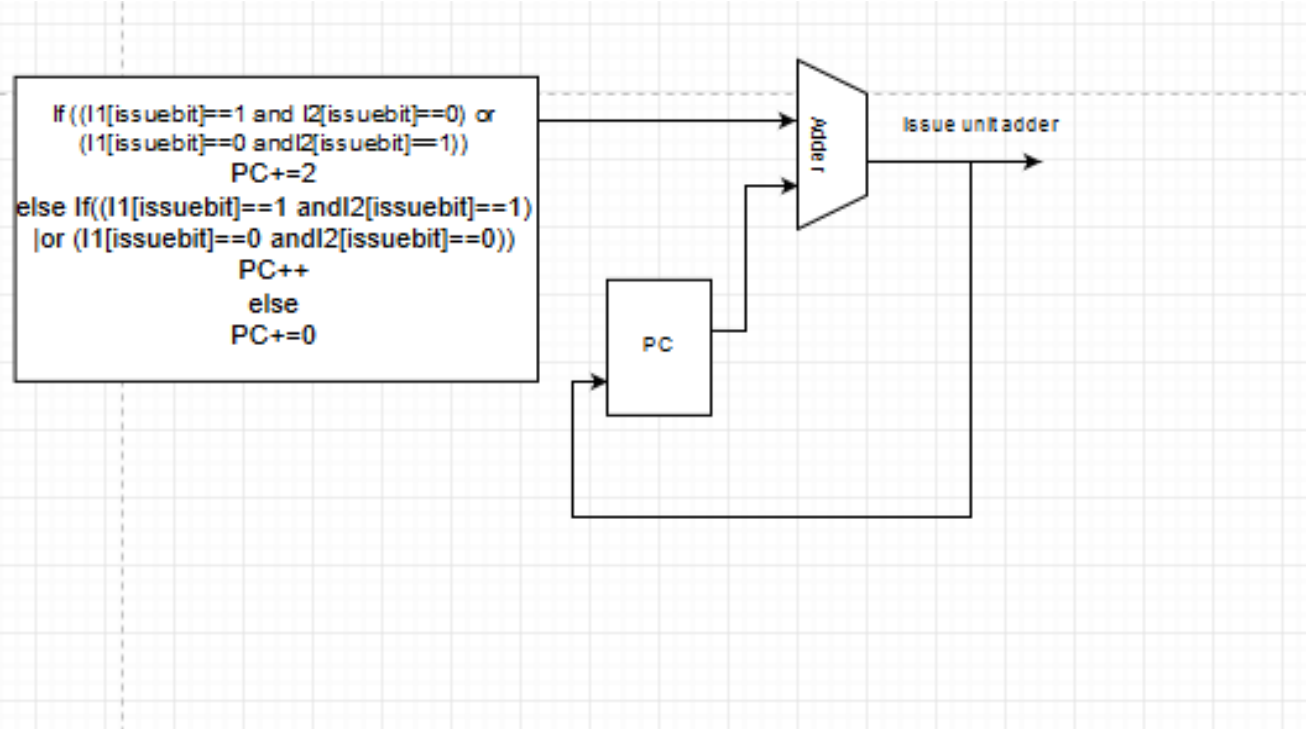
Write1Add,Write2Add are the addresses / names of the dst registers.



Issue unit:

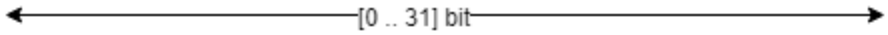
First half of the IR is a ld/store instruction and the second one is ALU instruction, we check the issue bit if 1(Ld/store) and if 0(ALU) and set PC according to result.
2Instructions at a time→ PC+=2
1Instruction at a time→ PC++
No Instructions → PC+=0
By this, we guarantee no data hazard(no two instruction dependency at same time).

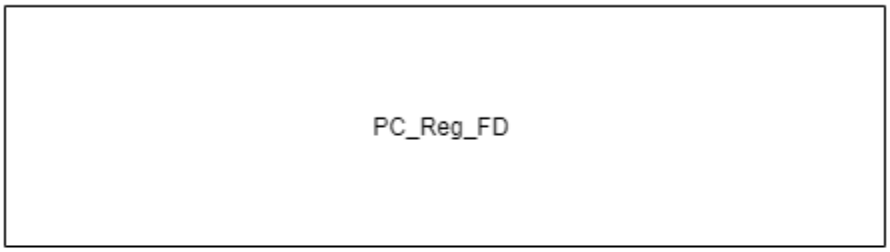




PipeLine Stages' Buffers:

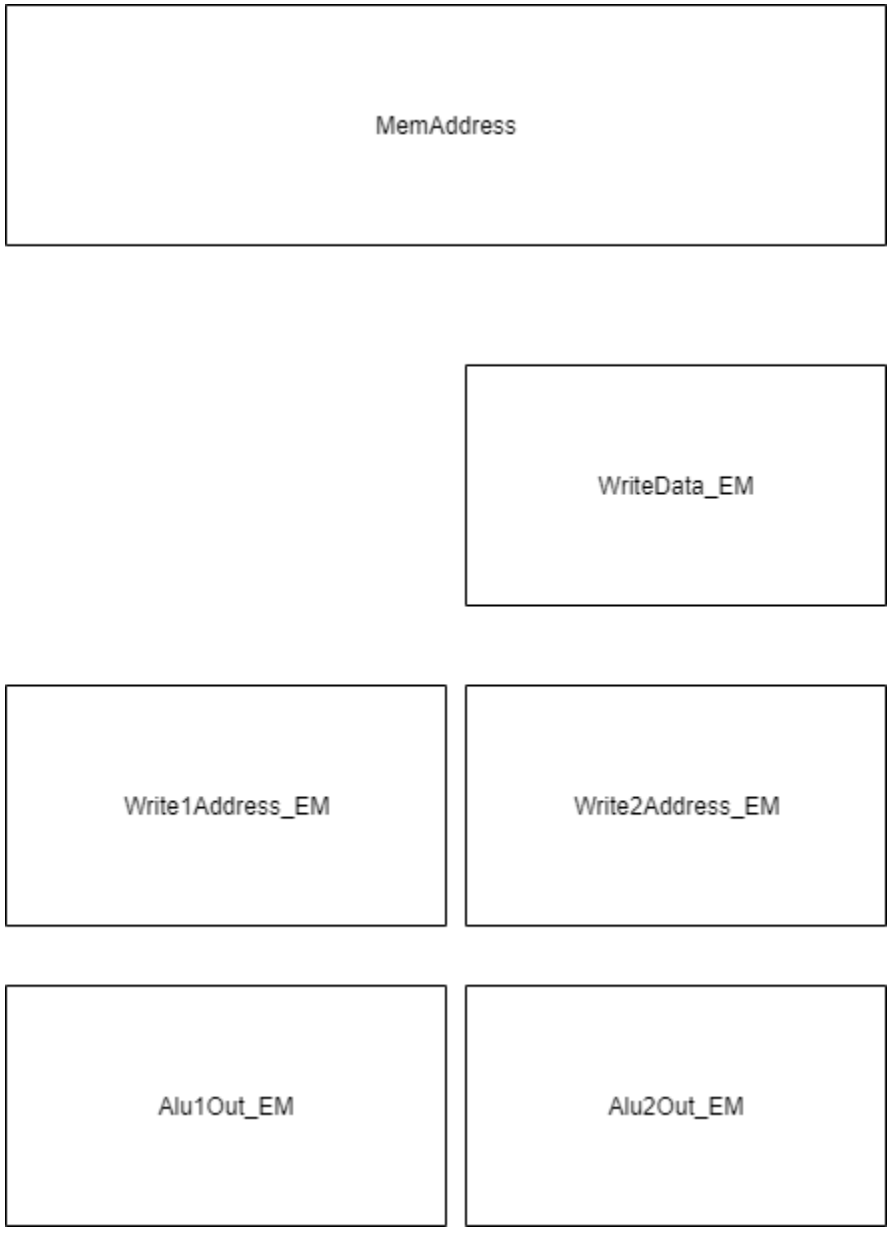
1. **IF/ID Buffer**





2. ID/IE Buffer

3. IE/MEM Buffer



4. MEM/WB Buffer

Pipeline Hazards Handling:

Structural Hazards:

Using 2 Alus, two memories and registers are edge triggered prevent the occurrence of structural hazards

Data Hazards:

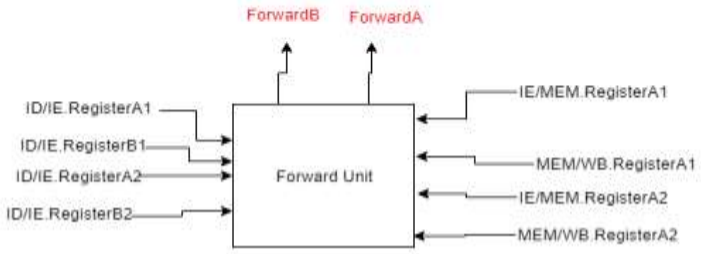
Forward Unit:

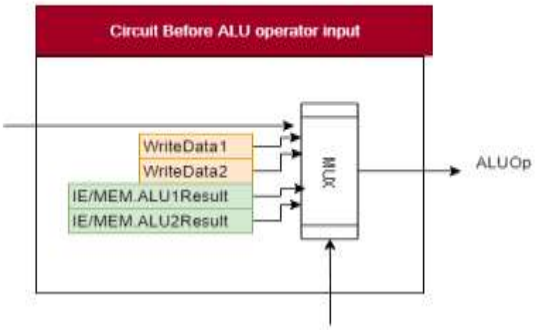
1. Ex hazard

if (EX/MEM.RegWrite and (EX/MEM.RegisterA1== ID/EX.RegisterA1)) ForwardA = 001
if (EX/MEM.RegWrite and (EX/MEM.RegisterA1 == ID/EX.RegisterB1)) ForwardB = 001
if (EX/MEM.RegWrite and (EX/MEM.RegisterA2== ID/EX.RegisterA2)) ForwardA = 010
if (EX/MEM.RegWrite and (EX/MEM.RegisterA2 == ID/EX.RegisterB2)) ForwardB = 010

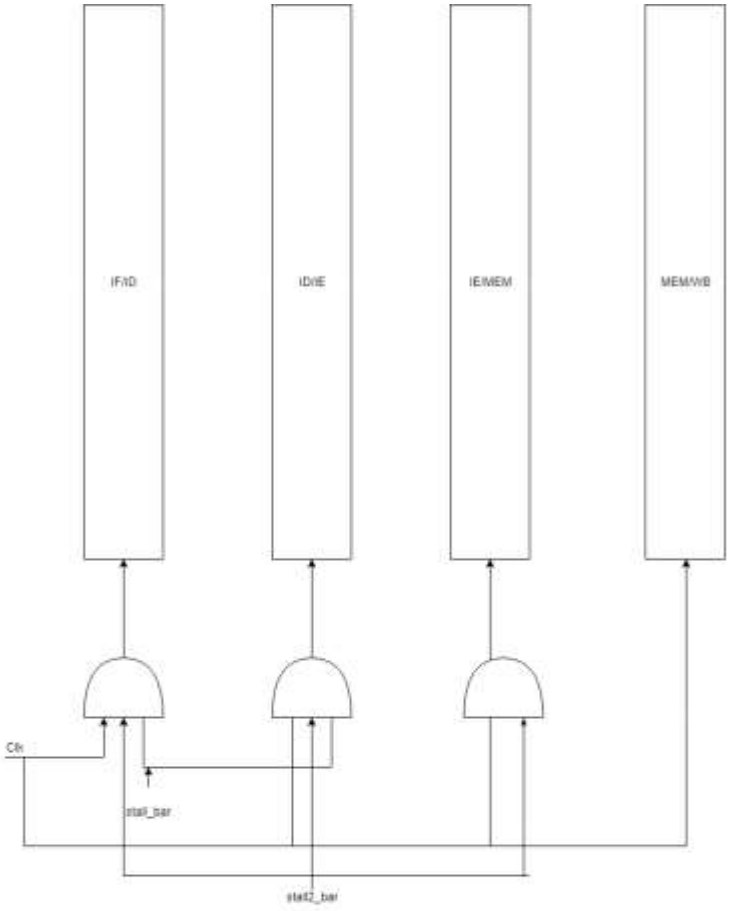
2. MEM hazard:

if (MEM/WB.RegWrite and (MEM/WB.RegisterA1 == ID/EX.RegisterA1)) ForwardA = 011
if (MEM/WB.RegWrite and (MEM/WB.RegisterA1== ID/EX.RegisterB1)) ForwardB = 011
if (MEM/WB.RegWrite and (MEM/WB.RegisterA2 == ID/EX.RegisterA2)) ForwardA = 100
if (MEM/WB.RegWrite and (MEM/WB.RegisterA2== ID/EX.RegisterB2)) ForwardB = 100



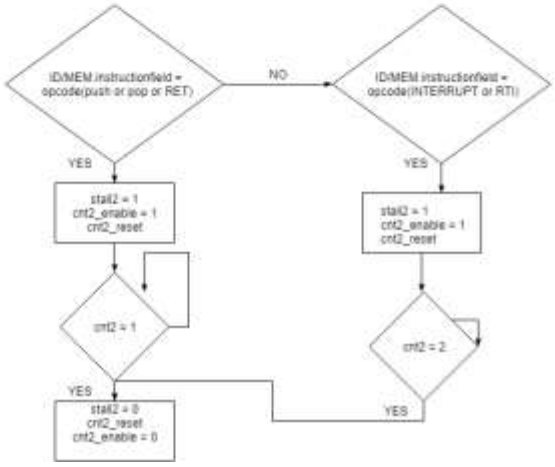


Stalling and "load-Use Data Hazard"



If(ID/IE.MemRead1 and ((ID/IE.RegisterA1 = IF/ID.ReisterA2) or (ID/IE.RegisterA1= IF/ID.RegisterB2))
stall = 1
cnt1_enable = 1
cnt1_reset

If(cnt1 = 1) stall = 0

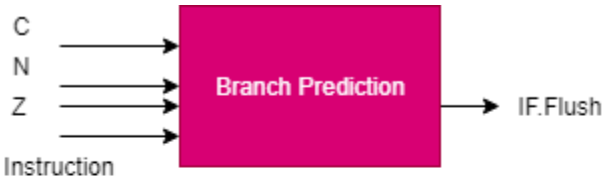


Control Hazards:

Static Branch Prediction:

We predict branches as always not taken. If branch is taken, then we flush

This circuit is in Decoding stage



If(IF.Flush = 1)
IF/ID.instructionField = 0

Control Unit:

```
--This is for the execution stage
--A and B are the 2 ops of the ALU
if (ir(15 downto 14) = "00") then
  if (ir(13 downto 10) = "0000") then
     -- NOP  00  0000    -      0
     -- ALUop =0
  elsif (ir(13 downto 10) = "0001") then
     -- SETC 00  0001    -      0
     -- ALUop = 1
  elsif (ir(13 downto 10) = "0010") then
     -- CLRC 00  0010    -      0
     -- ALUop = 2
  elsif (ir(13 downto 10) = "0011") then
     -- NOT Rdst 00  0011    xxx     0
     -- ALUop = 3
  elsif (ir(13 downto 10) = "0100") then
     -- INC Rdst 00  0100    xxx     0
     -- ALUop = 4
```

```
  elsif (ir(13 downto 10) = "0101") then
     -- DEC Rdst 00  0101    xxx     0
     -- ALUop = 5
  elsif (ir(13 downto 10) = "0110") then
     -- OUT Rdst 00  0110    xxx     1
     -- ALUop = 6
  elsif (ir(13 downto 10) = "0111") then
     -- IN Rdst  00  0111    xxx     1
     -- ALUop = 8
     -- ALUsrc = 11
  elsif (ir(13 downto 12) = "11") then
     --CALL Rdst 00  11  xxx     1
     --(X[SP] ← PC + 1; sp-2; PC ← R[ Rdst ])
  elsif (ir(13 downto 10) = "1010") then
     --RET   00  1010    -       1
     --sp+2, PC ←X[SP]
  elsif (ir(13 downto 10) = "1011") then
     --RTI   00  1011    -       1
     --sp+2; PC ← X[SP]; Flags restored
  elsif (ir(13 downto 10) = "1000") then
     --Reset 00  1000    -       1
     -- pcsrc = 2
  elsif (ir(13 downto 10) = "1001") then
     --Interrupt 00  1001    -       1
     --sp+2; PC ← X[SP]; Flags restored
  end if ;
elsif (ir(15 downto 13) = "011") then
  if (ir(12 downto 11) = "00") then
     --JZ Rdst   011 00  xxx     0
     --if (z ='0') pcsrc=1
     -- send jumpaddress to pc calculation circuit
  elsif (ir(12 downto 11) = "01") then
     --JN Rdst   011 01  xxx     0
     --if (n= '0')    pcsrc=1
     -- send jumpaddress to pc calculation circuit
  elsif (ir(12 downto 11) = "10") then
     --JC Rdst   011 10  xxx     0
     -- if (c ='0')    pcsrc=1
```

```
    -- send jumpaddress to pc calculation circuit
  elsif (ir(12 downto 11) = "11") then
    --JMP Rdst  011 11  xxx    0
    --pcsrc=1
    -- send jumpaddress to pc calculation circuit
  else --error code
  --do nothing
  end if ;
end if ;
```

```
if (ir [15]==1) then
        if(ir(14 downto 12)=="000") then
                --PUSH Rdst
                --sp=sp-1
                --mov sp,Rdst (ALUop =01000)
        elsif (ir(14 downto 12)=="001") then
                --POP Rdst
                --mov sp,Rdst (ALUop =01000)
                --sp=sp+1
        elsif(ir(14 downto 13)=="01") then
                --LDM Rdst, Imm
                --mov Rdst,Imm (ALUop =01000)
        elsif(ir(14 downto 13)=="10") then
                --LDD Rsrc, Rdst
                --Mem read at address Rsre
                --Reg write
        elsif (ir(14 downto 13 =="11")) then
                --STD Rsrc, Rdst
                --Mem write at address Rdst
```