Architecture project

Phase 1

# Names Sec B.N.

Hossam Ahmed 1 15

Sarah Mohamed 1 24

Nada Adel 2 29

Nourhan Gamal 2 33

# Instruction format

* Width of instruction = 16 bits
* 5 bits for opcode for all kind of instruction

## Opcodes:

### One operand

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode  5 bits | 000  3 bits | Rdst  3 bits | 00000  5 bits |

### Two operand (ADD, SUB, OR, AND)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode  5 bits | Rsrc1  3 bits | Rscr2  3bits | Rdst  3 bits | 0  2 bit |

### Two operand (SWAP)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode  5 bits | Rsrc1  3 bits | Rscr2  3bits | 000  3 bits | 0  2 bit |

### Two operand (IADD, SHL, SHR)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode  5 bits | 000  3 bits | Rscr  3bits | 0  1 bits | 0001  4 bit |

|  |
| --- |
| IMM  16 bit |

### Memory (PUSH, POP)

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode  5 bits | 000  3 bits | Rdst  3 bits | 00000  5 bit |

### Memory (LDM)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode  5 bits | 000  3 bits | Rdst  3 bits | 0  1 bit | 0001  4 bit |

|  |
| --- |
| IMM  16 bit |

### Memory (LDD, STD)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode  5 bits | 000  3 bits | Rdst  3 bits | EA(19:16)  4 bit | 1  1 bit |

|  |
| --- |
| EA(0:15)  16 bit |

### Branch and Change of Control Operations

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode  5 bits | 000  3 bits | Rdst  3bits | 00000  5 bits |

## Opcodes for every Instruction

|  |  |
| --- | --- |
| **Instruction** | **Opcode** |
| **NOP** | **00000** |
| **NOT  Rdst** | **00001** |
| **INC Rdst** | **00010** |
| **DEC Rdst** | **00011** |
| **OUT Rdst** | **00100** |
| **IN Rdst** | **00101** |
| **SWAP Rsrc, Rdst** | **00110** |
| **ADD Rscr1, Rscr2, Rdst** | **00111** |
| **IADD Rscr1, Rdst, Imm** | **01000** |
| **SUB Rscr1, Rscr2, Rdst** | **01001** |
| **AND Rscr1, Rscr2, Rdst** | **01010** |
| **OR Rscr1, Rscr2, Rdst** | **01011** |
| **SHL Rscr, Imm** | **01100** |
| SHR Rsrc,Imm | 01101 |
| **PUSH Rdst** | **01110** |
| **POP Rdst** | **01111** |
| **LDM Rdst, Imm** | **10000** |
| **LDD Rdst, EA** | **10001** |
| **STD Rscr, EA** | **10010** |
| **JZ Rdst** | **10011** |
| **JMP Rdst** | **10100** |
| **CALL Rdst** | **10101** |
| **RET** | **10110** |
| **RTI** | **10111** |

## Schematic diagram:

### Fetch and decode:

## 

Notes:

There is a bus from instruction fetched to the second 16 bits of instruction section of ID/EX reg, to load the instruction fetched which is the address in case of 32 bit instruction and use it in Execution stage.

There is enable signals from prediction feedback and from check branches to enable the BHT to read or write, (check branches. JZ OR prediction Feedback. Enable) to disable the BHT in the ordinary instructions and enable it in case of JZ to read the state and the instruction next to JZ to write then new state.

**Decision circuit**: is used to determine the next pc value, to be loaded to the pc in falling edge after fetching the current instruction at the beginning of cycle (rising edge).

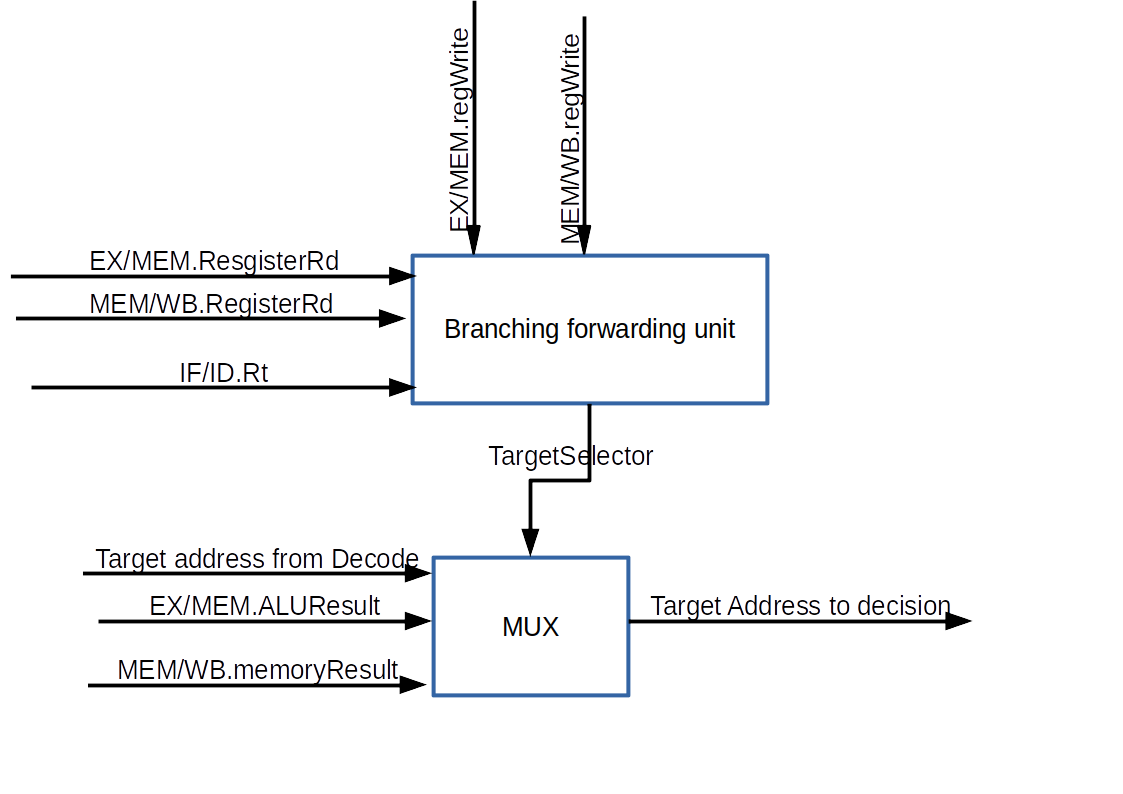
In case of branch instructions (jz, jmp, call, ret, RTI), at first we check for jz instruction which needs prediction, if the instruction is JZ then we load the current value of the pc to the branch pc.

**Branch pc**: used to save it to use in case of miss prediction (predict token and the prediction is wrong then we need to load the next instruction of branch instruction not the next of the token prediction target address).

**BHT**: used to save the state of JZ instructions, it has R/W input which determined by JZ signal.

**Check RRI**: to check for RET, RTI, interrupt instructions that need to write back the read PC from the memory stage, so it stalls the fetch stage.

**Hazard Detection Unit of branch**: needed to stall the fetch and decode in case of branch WAR data dependency (if.flush=1,pcWrite =0).

****

**1bit Counter**: used to determine if the current fetch stage is fetching address not instruction . 1 => address

* enable of counter is ORING of decode signal of 32/16 signal , 0th bit in instruction fetched.

**decision circuit at IF/ID:** its input is the fetched instruction and counter to determine the output which is zeros or the instruction. In case of counter =1 then the output will be the instruction forwarded to ID/EX is the instruction(address) and the another output is zeros to stored , if.flush = 1 ,,if counter =0 then the instruction stored in IF/ID and if.flush = 0.

IF/ID => interrupted 0 in normal flow.

1=> when stage receive interrupt signal it stall fetch make pcwrite =0, put interrupt signal =1 in IF/ID.

Other stages put interrupt signal in the next intermediate register =1, then the next cycle to this stage it will stall itself.

In memory we would operate 4 cycles:

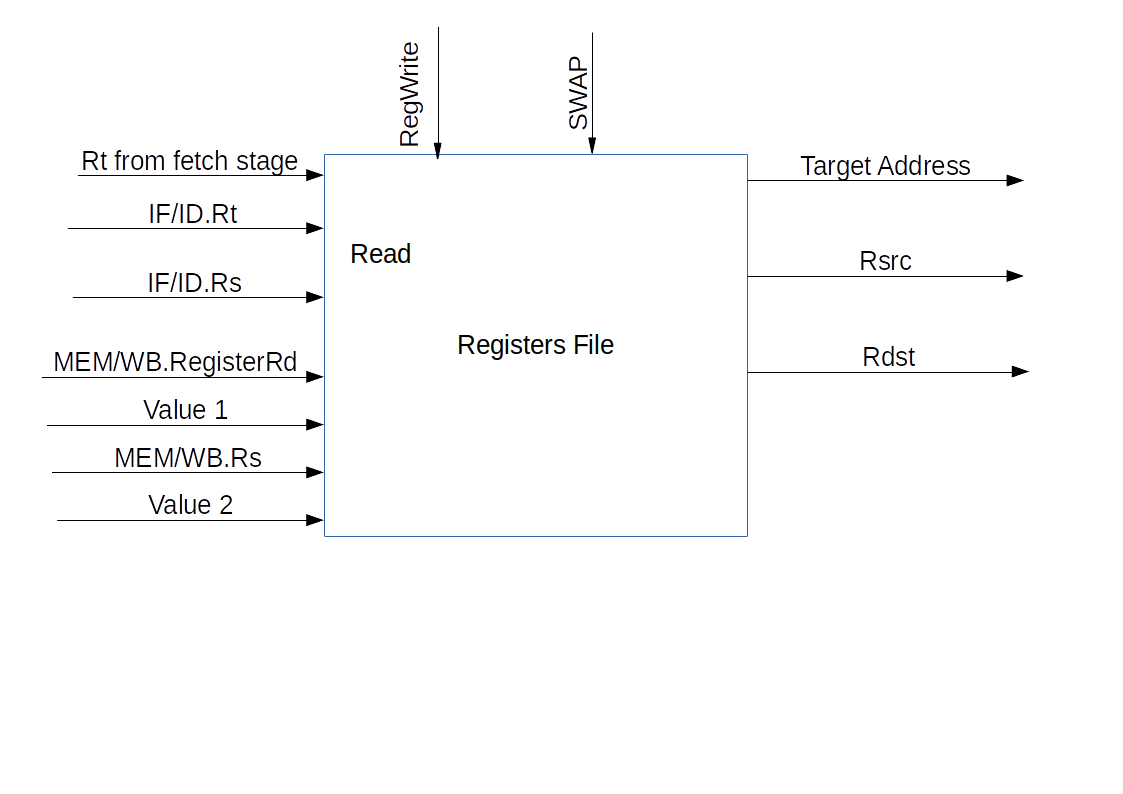
First cycle: to perform the current instruction.

Second cycle: to write PC value in the stack, after this cycle stops WB.

Third cycle: to put the address of M (2), M (3) to PC.

Fourth cycle: to store the flags in stack.

**Decode:**



at swap instruction : swap signal =1 then it enable the write functionality of write at Rs register.

**Hazard detection unit**

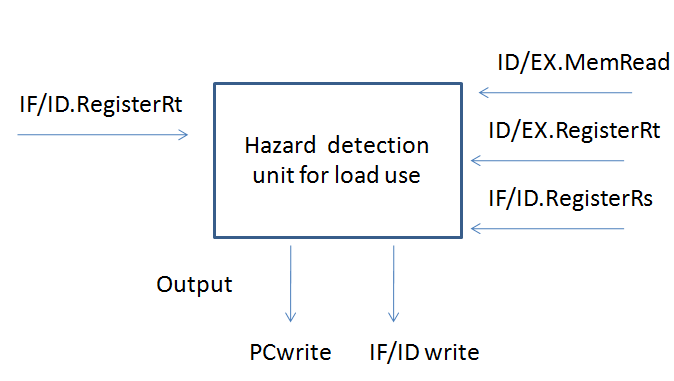
 Checking for load instructions, the control for the hazard detection unit is this single condition:

if (ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or

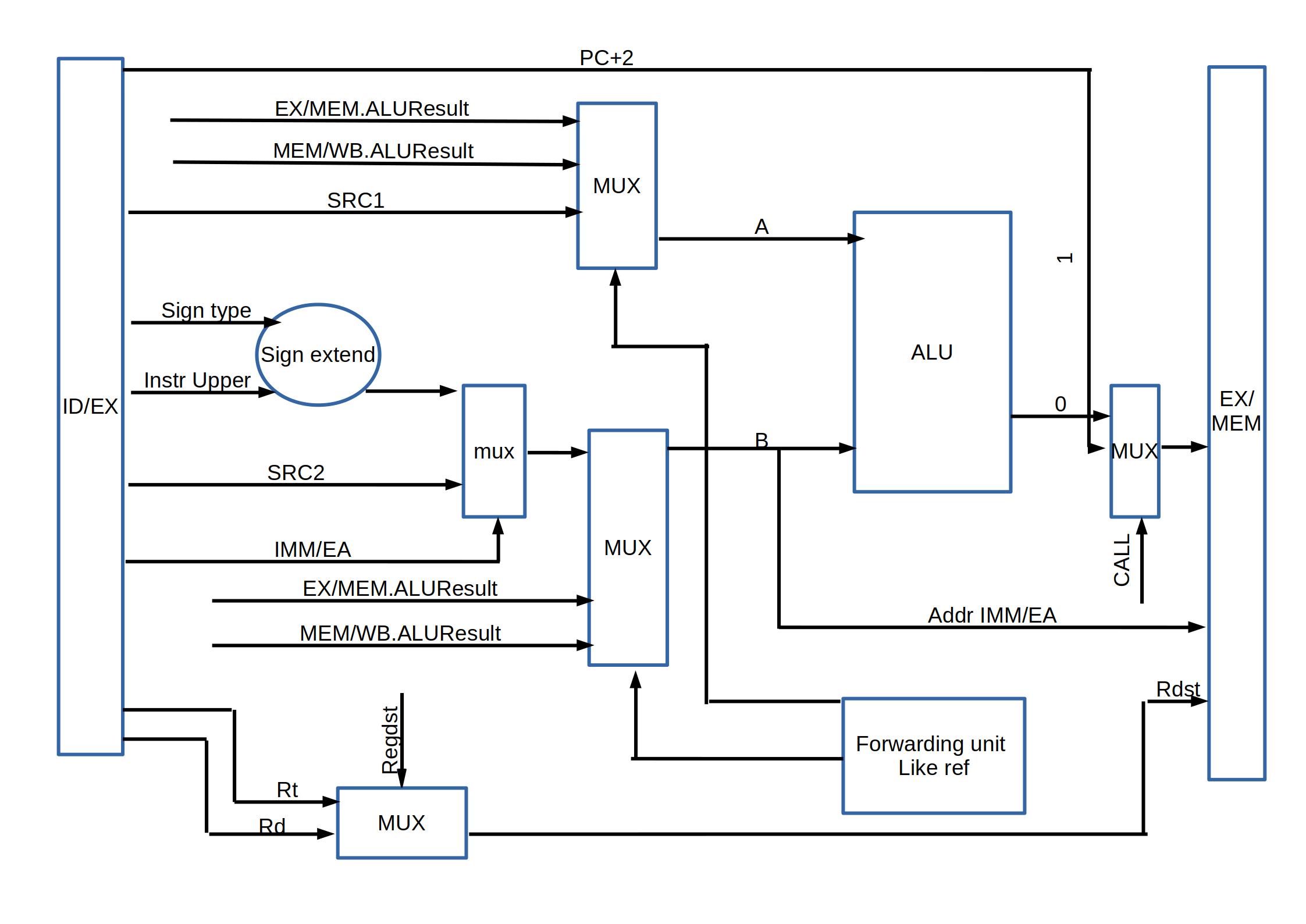
(ID/EX.RegisterRt = IF/ID.RegisterRt)))   stall the pipeline

ID stage is stalled, then the instruction in the IF stage must also be stalled by preventing the PC register and the IF/ID pipeline register from changing. a stall bubble delays everything behind it.

 The hazard detection unit controls the writing of the PC and IF/ID registers plus the multiplexor that chooses between the real control values and all 0s. The hazard detection unit stalls and deasserts the control fields if the load-use hazard test above is true.

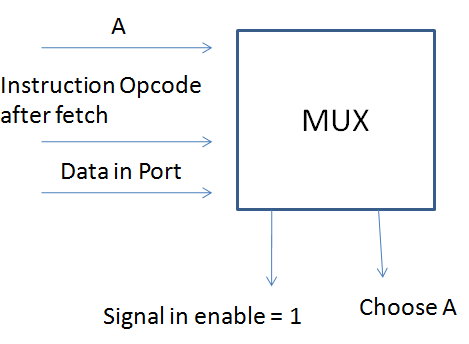


Execution :



* We add multiplexer before ALU takes A and input port value and selects between them.

In this case selector will be one (signal **in enable =1).**



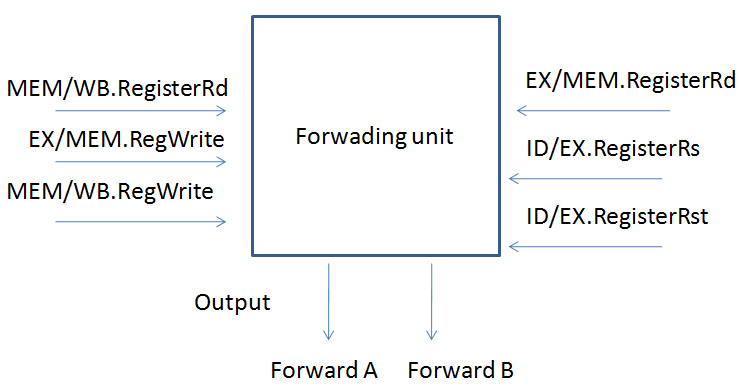
### MUX to load to data to be written in the memory take the data from ALUResult , PC+2 from IF/ID register. The selector is CRR signal which indicate that the instruction is CALL or RET or RTI which need to write the pc value

### ALU selectors bits (4 bits):

* NOP: 0000
* NOT: 0001
* INC: 0010
* DEC: 0011
* SHL: 0100
* SHR: 0101
* ADD: 0110
* SUB: 0111
* AND: 1000
* OR: 1001
* A: 1010
* B: 1011

### 1- Data Hazard Forwarding Unit

**Forwarding unit diagram**



Data Hazard Conditions:

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

### 1. EX hazard:

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and

(EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and

EX/MEM.SWAP and

(EX/MEM.RegisterRs = ID/EX.RegisterRs)) ForwardA = 10

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and

(EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and

EX/MEM.SWAP and

(EX/MEM.RegisterRs = ID/EX.RegisterRt)) ForwardB = 10

### 2. MEM hazard:

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and

(MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and

MEM/WB.SWAP and

(MEM/WB.RegisterRs = ID/EX.RegisterRs)) ForwardA = 01

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and

(MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and

MEM/WB.SWAP and

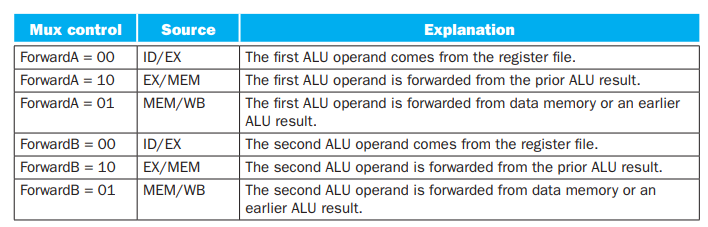
(MEM/WB.RegisterRs = ID/EX.RegisterRt)) ForwardB = 01

In this case, the result is forwarded from the MEM stage because the result in the MEM stage is the more recent result. The control for the MEM hazard would be :

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and

(EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01 if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt))

 and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01



## **Control hazards and branching :**

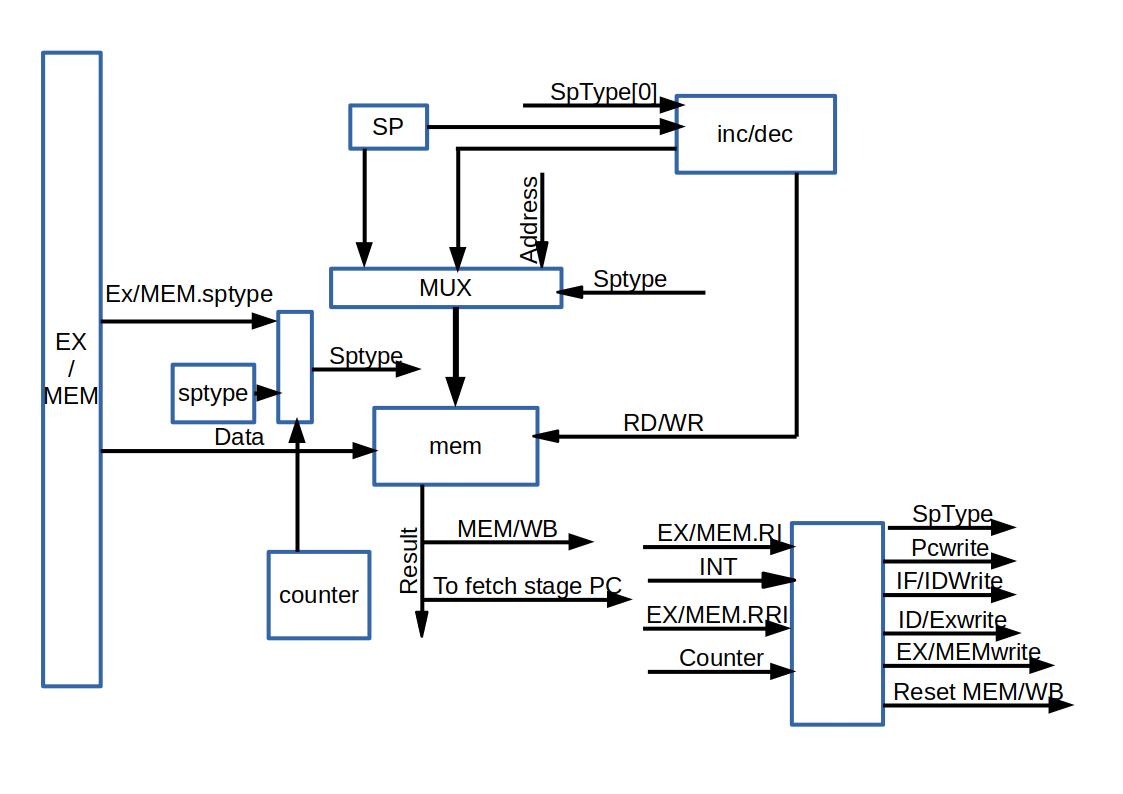
In fetch stage we add check branching and check RTI to determine the current instruction and check if it has control hazard or not ,

in case of unconditional branching JMP ,CALL, it gets the pc new value from decoding stage.

in case of JZ which need prediction at first it will read the prediction state from BHT then according to it it determine whether pc new will be PC+2(NT) or target address from decode,then in the next cycle in fetching of new instruction and decode of JZ instruction , then the prediction feedback circuit out the T/NT/notJZ to determined if the prediction was right or not ,and to determine the new state to store in BHT.

RET,RTI : they need the pc from memory stage then in each stage if RRI =1 then it will stall itself and the previous stage to prevent any new instruction from execution , and in memory stage it will stall itself and the previous stages and determine the logic of memory according to the instruction.

### Memory:



* Counter enable determined by RRI, RI, interrupt signal.
* There is a Decision circuit to determine data input to memory.
* Inputs are counter, RI, RRI, interrupt signal to be selectors, and the data to be selected pc from fetch, flags from ALU, EX/MEM.Data
* There is a Decision circuit to determine address of selector.
* Inputs are  counter ,RI,RRI,interrupt signal, reset signal  to be selectors, and the addresses to be selected 0 for reset ,2 for interrupt, EX/MEM.address.

## Pipeline registers details:

### IF/ID register

Inputs of this register are IR (2 bytes) + PC new (after incremented) (4 bytes), interrupt signal (1 bit), RRI (1 bit), reset (1 bit),. Instruction of fetch stage is navigated to ID/IE to be stored in the lower 16 bits of the register. (51 bits)

### In case of data hazards:

* PC needs an enable in case of stall the pipeline.
* IF/ID has an enable so that it will not be changed.

In case of control hazard

In jz instruction when decode decision is opposite to prediction then prediction FSM make make if.flush = 1

### ID/EX:

Needs the values are read from the register file of Rscr1 and Rscr2 so  4 Bytes for Rscr1 + 4 Bytes for Rscr2, instruction (4 bytes),interrupt signal (1 bit ) ,RRI(1 bit), reset (1 bit ),Rt(3bits),Rs(3bits),SWAP, CALL

* WBsignals = 3 signals => memtoreg, regwrite, outenable =>3bits.
* EXsignals =ALUop=>4bits.
* SignExtend, IMM/EA, regDst, CRR=>8bits.
* MEM signals = memRead, memWrite, spType=>4bits.

(126 bits)

## EX/MEM:

It must be big enough to hold all possible situations as the following:

* Rt(3bits),Rs(3bits),SWAP
* ALUresult/pc+2 =>4bytes, address =>4bytes, Rdst (3 bit).
* Interrupt signal (1 bit), RRI (1 bit), reset (1 bit).
* MEM signals = memRead, memWrite, spType=>4bits.
* WBsignals = 3 signals => memtoreg, regwrite, outenable =>3bits.

(84 bits)

## MEM/WB:

* Rt(3bits),Rs(3bits),SWAP,address(4 bytes)
* Result from the memory (.i.e. in load instruction it will result 4bytes for the value to be WB in DST reg).
* Result from ALU (4bytes).

(103 bits)

**Fetching 2 word instructions:**

Bit 4 in instruction is indicator to the type of instruction:

* ‘0’ if the instruction is 16 bits.
* ‘1’ if the instruction is 32 bits.

In fetch stage we know the instruction then use the 4th bit of instruction and the 32/16 signal from decode to enable the counter.

When counter =0 then the stage fetch the first 16 bits of the instruction , then it make decision circuit will store the instruction in IF/ID , and forward zeros to ID/EX upper word of instruction section ,then in the next cycle we decode the first part of instruction and we know that the instruction is 32 bits then 32/16 =1 then counter will be 1 and when 32/16 =1 then if.flush = 1 to insert bubble to the remaining of the fetched instruction which is address. When counter =1 then instruction forwarded to ID/EX instruction section.

## Control signal of each instruction:

## One operand:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instr\  signals | RegWr | RegDST | Mem  To  Reg | Mem  Rd | MemWR | SP | ALU | PC  write | Imm/  EA | sign | CRR | in enable | Out enable |
| NOP | 0 | 0 | 0 | 0 | 0 | 11 | nop | 1 | 0 | 0 | 0 | 0 | 0 |
| NOT Rdst | 1 | 1 | 0 | 0 | 0 | 11 | A | 1 | 0 | 0 | 0 | 0 | 0 |
| INC Rdst | 1 | 1 | 0 | 0 | 0 | 11 | A | 1 | 0 | 0 | 0 | 0 | 0 |
| DEC Rdst | 1 | 1 | 0 | 0 | 0 | 11 | A | 1 | 0 | 0 | 0 | 0 | 0 |
| OUT Rdst | 0 | 1 | 0 | 0 | 0 | 11 | nop | 1 | 0 | 0 | 0 | 0 | 1 |
| IN Rdst | 1 | 1 | 0 | 0 | 0 | 11 | nop | 1 | 0 | 0 | 0 | 1 | 0 |

## Branch:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instr\  signals | Reg  WR | Reg  DST | Mem  To  Reg | out  enable | Mem  Rd | memWR | SP | AlU | 32  /16 | Imm/  EA | sign | CRR |
| PUSH | 0 | x | 0 | 0 | 0 | 1 | 00 | B | 0 | x | 1 | 0 |
| POP | 1 | 1 | 0 | 0 | 1 | 0 | 01 | nop | 0 | 0 | 1 | 0 |
| LDM | 1 | 1 | 0 | 0 | 0 | 0 | 10 | B | 1 | 1 | 1 | 0 |
| LDD | 1 | 1 | 0 | 0 | 1 | 0 | 10 | nop | 1 | 1 | 0 | 0 |
| STD | 0 | 1 | 0 | 0 | 0 | 1 | 10 | A | 1 | 1 | 0 | 0 |
| JZ | 0 | 0 | 0 | 0 | 0 | 0 | 11 | nop | 0 | 0 | x | 0 |
| JMP | 0 | 0 | 0 | 0 | 0 | 0 | 11 | nop | 0 | 0 | x | 0 |
| CALL | 0 | 0 | 0 | 0 | 0 | 1 | 00 | nop | 0 | 0 | x | 1 |
| RET | 0 | x | 0 | 0 | 1 | 0 | 01 | nop | 0 | 0 | x | 1 |
| RTI | 0 | x | 0 | 0 | 1 | 0 | 01 | nop | 0 | 0 | x | 1 |
| instr\signals | RegWR | RegDST | Mem  To  Reg | out  enable | Mem  Rd | memWR | SP | ALU | 32/  16 | Imm/  EA | sign | CRR |
| Swap | 1 | 1 | 0 | 0 | 0 | 0 | 11 | nop | 0 |  | 0 | 0 |
| ADD | 1 | 1 | 0 | 0 | 0 | 0 | 11 | Add | 0 | 0 | 0 | 0 |
| IADD | 1 | 1 | 0 | 0 | 0 | 0 | 11 | Add | 0 | 1 | 1 | 0 |
| SUB | 1 | 1 | 0 | 0 | 0 | 0 | 11 | Sub | 0 | 0 | 0 | 0 |
| AND | 1 | 1 | 0 | 0 | 0 | 0 | 11 | And | 0 | 0 | 0 | 0 |
| OR | 1 | 1 | 0 | 0 | 0 | 0 | 1111 | or | 0 | 0 | 0 | 0 |
| SHL | 1 | 0 | 0 | 0 | 0 | 0 | 1111 | shl | 1 | 1 | 0 | 0 |
| SHR | 1 | 0 | 0 | 0 | 0 | 0 | 1111 | shr | 1 | 1 | 0 | 0 |