PIPELINE-PROCESSOR

Team 10

Names

- **Moamen Hassan Attia**
- **Ahmed Salama Hamed**
- **Ahmed Maher**
- **Mohamed Talaat**

Instruction Format

- IR register (16 bit)

Op code (2 bits)	Func (3 bits)	Rsrc (4 bits)	Rdst (4 bits)

- Op code

00	One operand
01	Two operands
10	Memory
11	Branch

- One operand function (00)

000	No op
001	Set carry
010	Clear carry
011	Not rdst
100	Increment rdst
101	Decrement rdst
110	Out rdst
111	In rdst

- Two operand functions (01)

000	Mov
001	add
010	Sub
011	And
100	Or
101	Shift left
110	Shift right

- Memory functions (10)

000	push
001	pop
010	Load imm
011	Load from memory
100	Store to memory

- Branch functions (11)

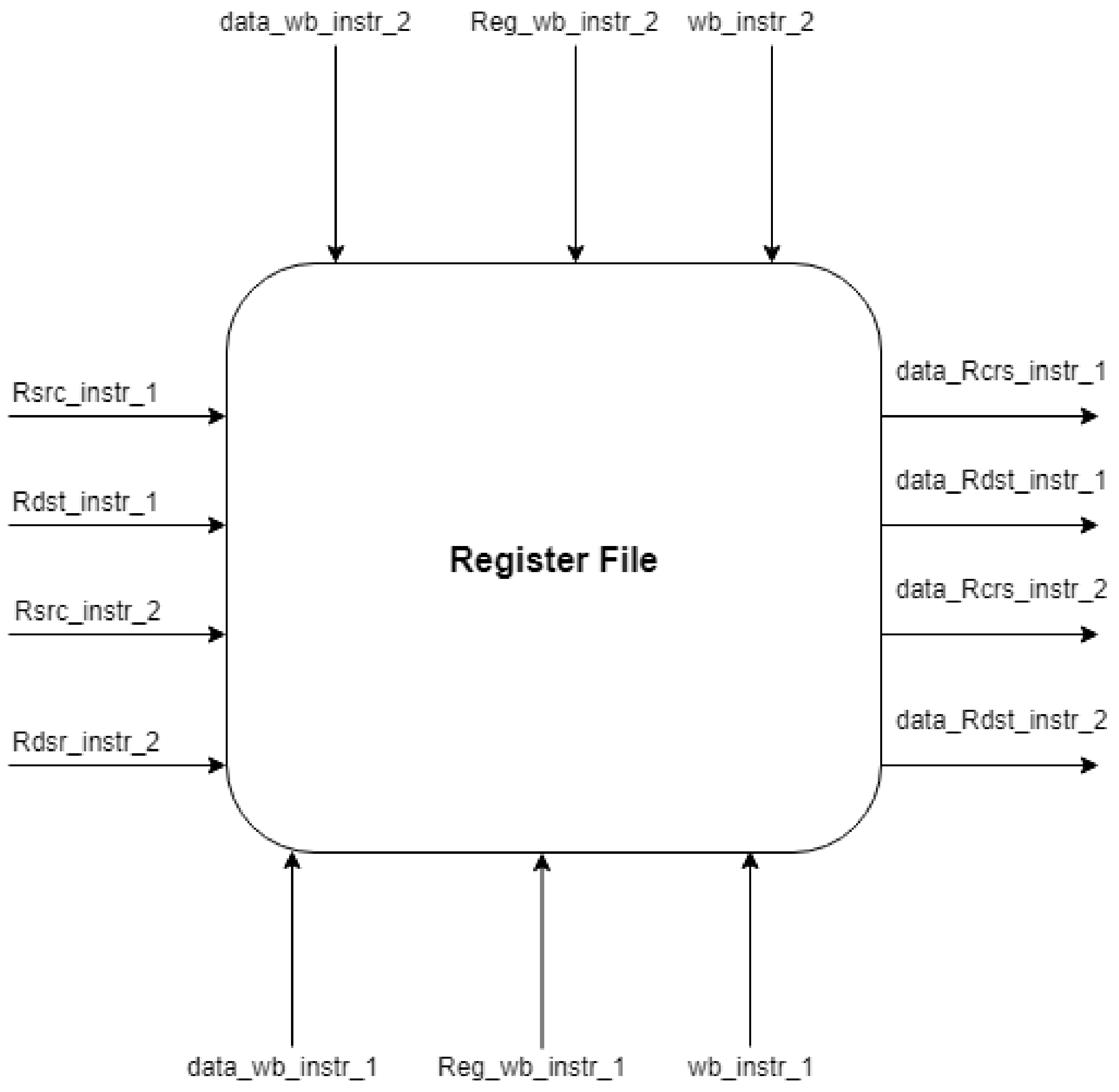
000	jump if zero
001	jump if negative
010	jump if carry
011	jump (unconditional)
100	call rdst(call subrutine)
101	RET (return from subroutine)
110	RTI (return form interrupt)

- Registers

R0	0000
R1	0001
R2	0010
R3	0011
R4	0100
R5	0101
R6	0110
R7	0111
SP	1000
PC	1001
Flag	1010

Schematic diagram Blocks

- **Register File**

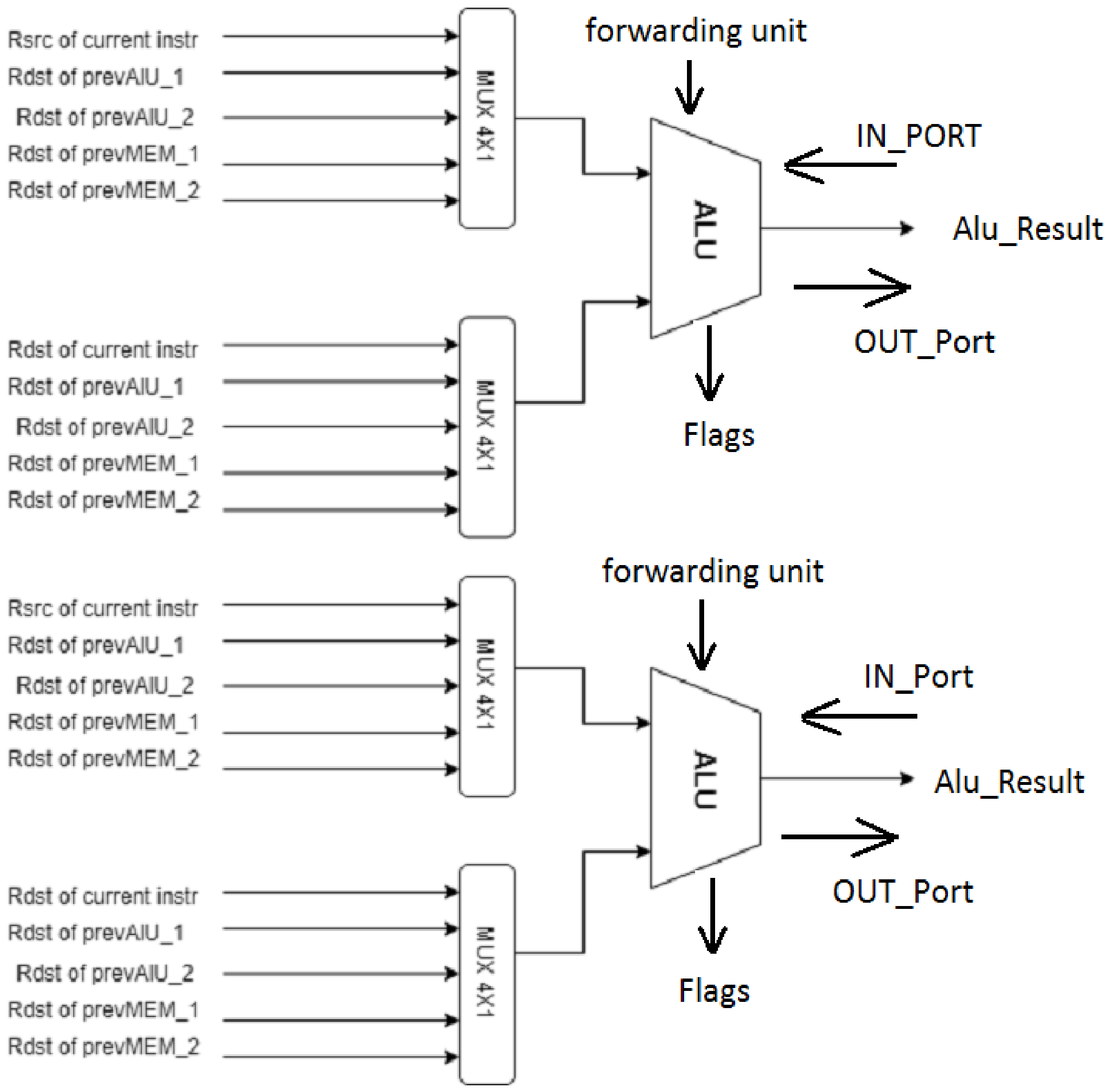


- Alu

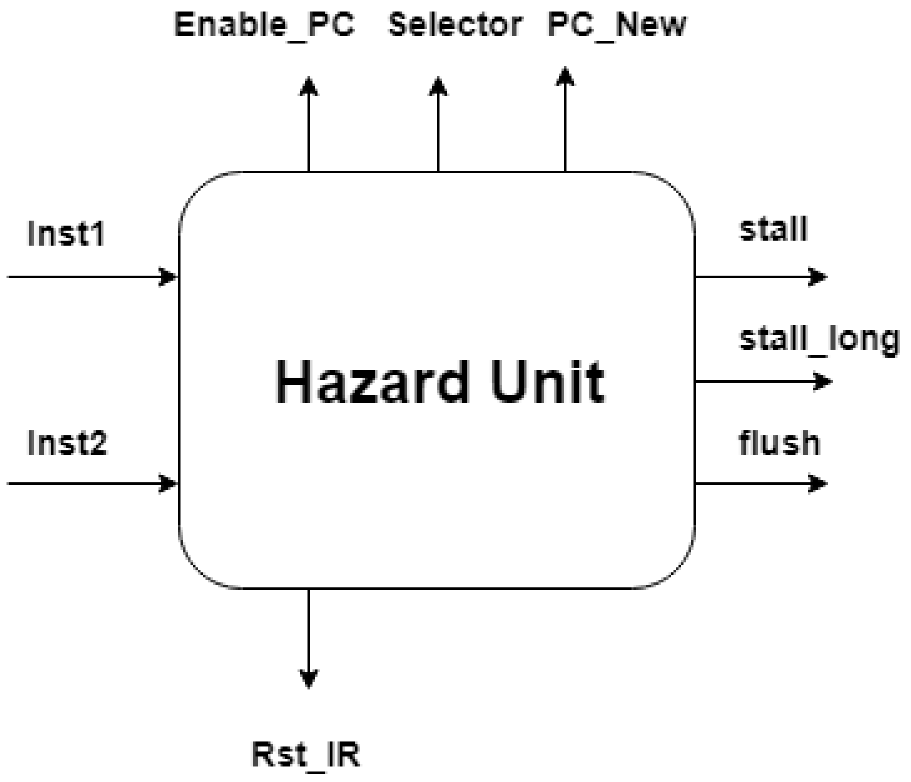
  -changes the flags register directly

  -Mux selectors form forwarding unit

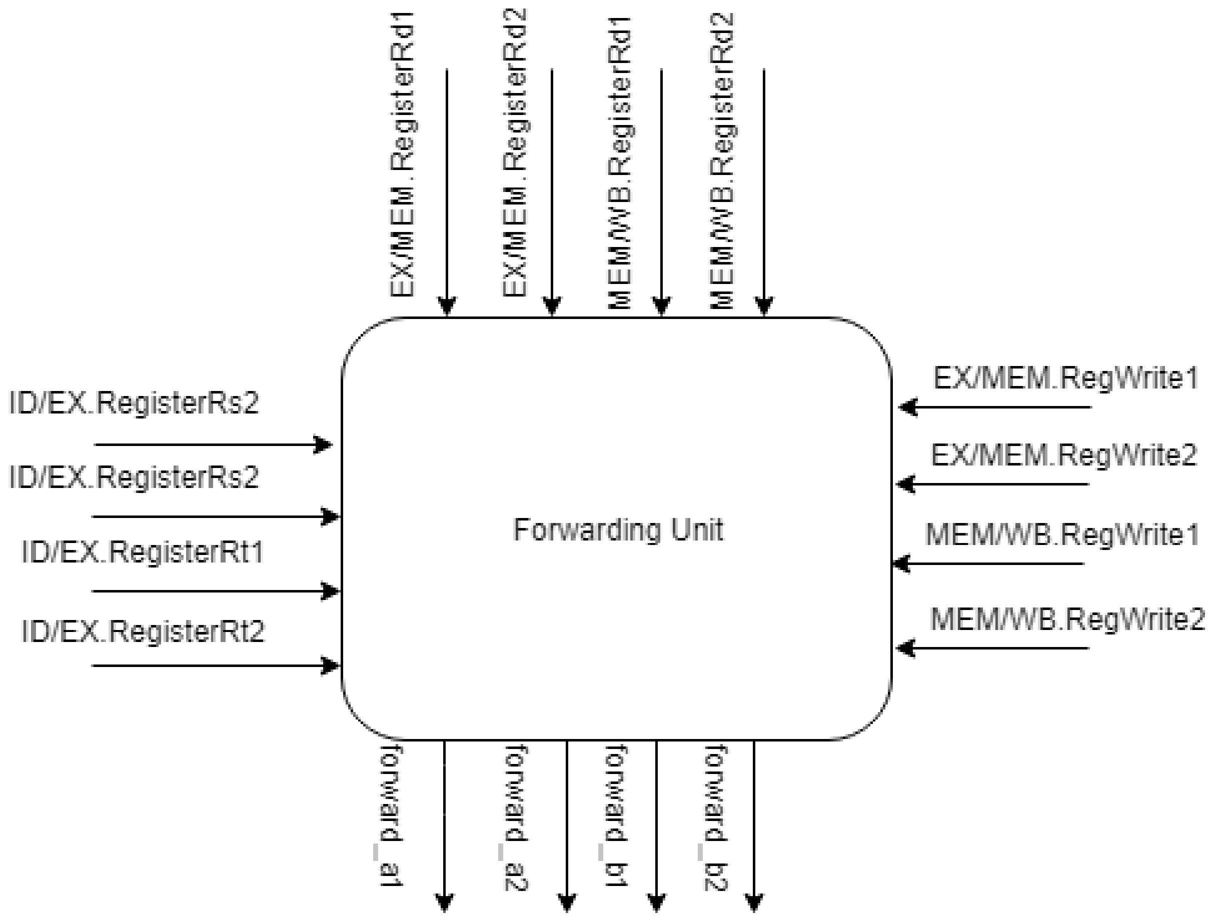
  -alu is connected directli to IN and OUT PORTS

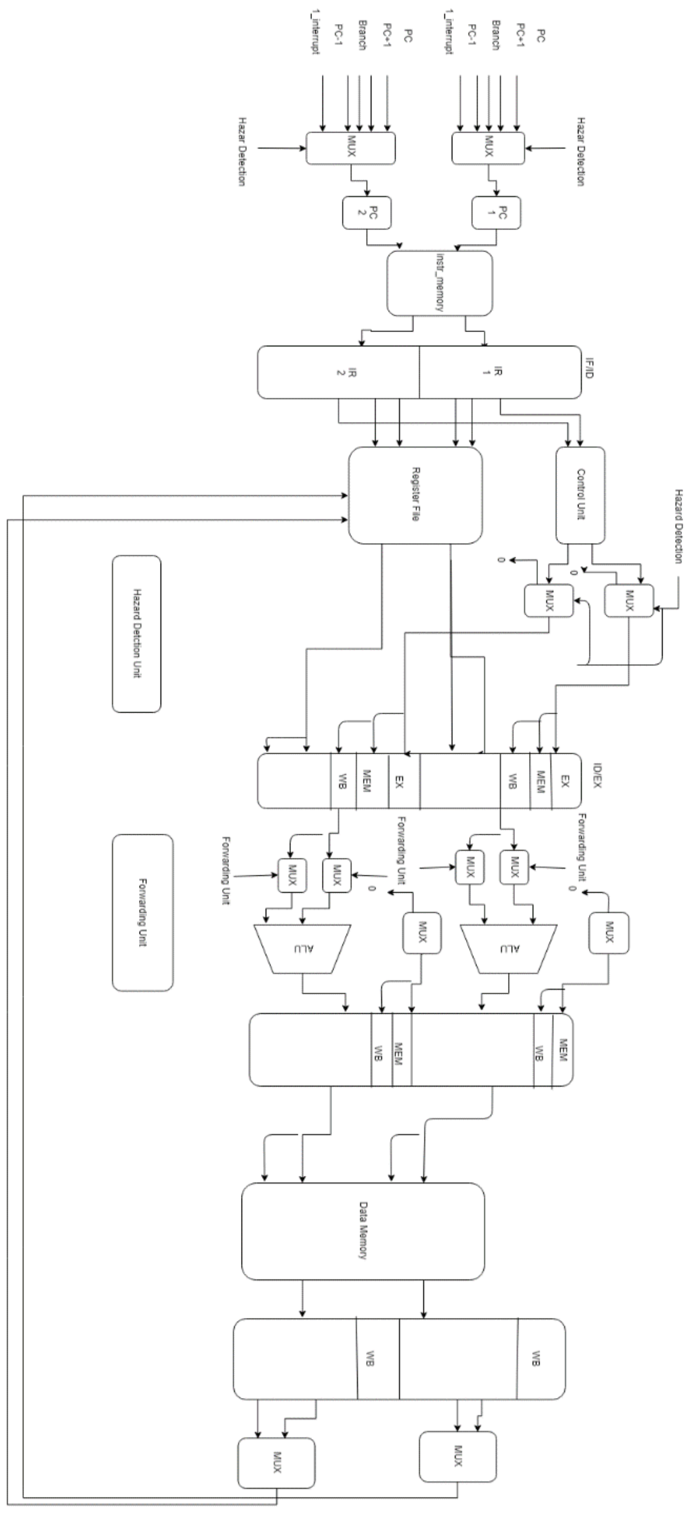


- Hazard Detection



- Forwarding unit




PC
PC+1
Branch
PC-1
1_interrupt
Hazard Detection
Hazar Detection
MUX
PC 1
PC 2
instr_memory
IF/ID
IR 1
IR 2
Control Unit
Register File
Hazard Detection
0
ID/EX
EX
MEM
WB
Forwarding Unit
ALU
Hazard Detection Unit
Forwarding Unit
Data Memory

Pipeline register details

- F/D Buffer (16 bit) 2

I R

- D/E Buffer (51 bit) x 2

Rdst (4 bit)
Rsrc (4 bit)
Branch taken (1 bit)
Load use (1 bit)
Rsrc data (16 bit)
Rdst data (16 bit)
Stall_long (1 bit)
WB (1 bit)
Memory read (1 bit)
Memory write (1 bit)
Alu op (2 bit)

- E/M (24 bit) x 2

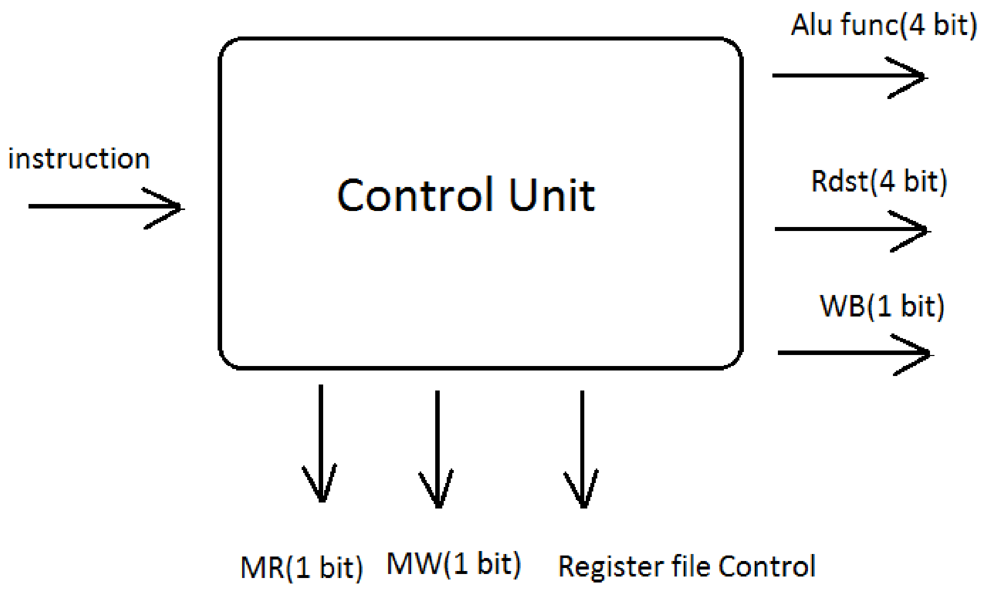
Rdst (4 bit)
Wb (1 bit)
Stall_long (1 bit)
Memory read (1 bit)
Memory write (1 bit)
Alu result (16 bit)

- M/W (39 bit) x 2

Rdst (4 bits)
Wb (1 bit)
Memory read (1 bit) mux select
Stall_long (1 bit)
Alu result (16 bit)
Memory result (16 bit)

Control Signals Details

- **Control unit**



Control Signals

- **Alu Control**

combination	**F =**
00000	No op
00001	Set carry (one op)
00010	Clear carry (one op)
00011	Not (one op)
00100	Increment (one op)
00101	Decrement (one op)
00110	Rdst
00111	Rsrc
01000	Add
01001	Sub
01010	And
01011	Or
01100	Shift left
01101	Shift right
01110	out
01111	in
10000	Inc src (sp)
10001	Dec src (sp)

- MR , MW , WB , Register file (which register to open according to IR)

One operand Instructions

(alu uses Rdst as the one operands)

- No op

  IR all zero (one operand – no op)

  WB, MR, MW zero

  ALU no op

- Set carry

  IR (one operand |set carry | rsrc = 0 | rdst =0)

  WB, MR, MW zero

  ALU set carry

- Clear carry

  IR (one operand | clear carry | rsrc = 0 | rdst =0)

  WB, MR, MW zero

  ALU set carry

- Not dst

  IR (one operand | not | rsrc =0 | rdst = dst)

  MW ,MR zero

  WB one

  ALU not

- Inc dst

  IR (one operand | inc | rsrc =0 | rdst = dst)

  MW,MR zero

  WB one

  ALU not

  - Dec dst

  IR (one operand | dec | rsrc =0 | rdst = dst)

  MW,MR zero

  WB one

  ALU dec

  - Out dst

  IR (one operand | dec | rsrc =0 | rdst = dst)

  MW,MR zero

  WB zero

  ALU out

  - In rdst

  IR (one operand | dec | rsrc =0 | rdst = dst)

  MW,MR zero

  WB 1

  ALU in

Two operand instructions

- Mov

  IR (Two operands | Mov | Rsrs=src | Rdst=dst)

  MR, MW= 0

  WB=1

  ALU F=Rsrc

- Add

  IR (Two operands | Add | Rsrs=src | Rdst=dst)

  MR, MW=0

  WB=1

  ALU F=Rdst+Rsrc

- Sub

  IR (Two operands | Sub | Rsrs=src | Rdst=dst)

  MR, MW=0

  WB=1

  ALU F=Rdst-Rsrc

- And

  IR (Two operands | And | Rsrs=src | Rdst=dst)

  MR, MW =0

  WB=1

  ALU F=Rdst And Rsrc

- OR

  IR (Two operands | OR | Rsrs=src | Rdst=dst)

  MR, MW=0

  WB=1

  ALU F=Rdst OR Rsrc

- Shift left

  IR (Two operands | Shift_left | Rsrs=0 | Rdst=dst)

  MR, MW=0

  WB=1

  From hazards when shift instr detected "immediate" shift amount value will be forwarded as Rsrc to ALU as it comes in the following line in the instr memory.

  ALU F=shift_left

- Shift right

  IR (Two operands | Shift_right | Rsrs=0 | Rdst=dst)

  MR, MW=0

  WB=1

  From hazards when shift instr detected "immediate" shift amount value will be forwarded as Rsrc to ALU as it comes in the following line in the instr memory.

  ALU F=shift_right

Memory Instructions

- Push dst

IR (mem | push | rsrc =sp | rdst = 0)

MW1

MR , WB zero

ALU dec sp

- Pop dst

IR (mem | pop | rsrc =0 | rdst = dst)

WB , MW1

MR 1

ALU inc

- LDM dst ,imm

IR (mem | ldm | rsrc =0 | rdst = dst)

WB 1

MR ,MW 0

ALU rsrc

- LDD rsrc ,rdst

IR (mem | ldd | rsrc =rsrc | rdst = dst)

WB , MR1

MW 0

ALU rsrc

- STD rsrc , rdst

IR (mem | std | rsrc =rsrc | rdst = dst)

WB , MR0

MW 1

ALU rdst

Branch instructions

-Branch offset and Branch evaluation both are in Decode phase

- jump if zero

  IR (Branch | JZ | Rsrs=0 | Rdst=dst)

  WB,MR, MW=0

  ALU F=no op

  PC <-- Branch offset (will be selected by the hazards detection unit)

- jump if negative

  IR (Branch | JN | Rsrs=0 | Rdst=dst)

  WB,MR, MW=0

  ALU F=no op

  PC <-- Branch offset (will be selected by the hazards detection unit)

- jump if carry

  IR (Branch | JC | Rsrs=0 | Rdst=dst)

  WB,MR, MW=0

  ALU F=no op

  PC <-- Branch offset (will be selected by the hazards detection unit)

- jump (unconditional)

  IR (Branch | JMP | Rsrs=0 | Rdst=dst)

  WB,MR, MW=0

  ALU F=no op

  PC <-- Branch offset (will be selected by the hazards detection unit)

- Call subroutine

  IR (Branch | CALL | Rsrc=SP | Rdst=dst)

  WB,MR=0

  MW=1

  ALU F=sp-1

  PC will be pushed to stack fisrt - PC -> memory data

  (alu src = sp

  mem[sp]=pc+1)

  PC <-- subroutine offset (will be selected by the hazards detection unit)

- RET from subroutine

  IR (Branch | RET | Rsrs=0 | Rdst=PC)

  MR=1

  MW=0

  WB=1

  ALU F=sp+1

  PC will be poped from stack fisrt - data read from stack will be saved back in PC

  (alu src = sp

)

  PC <-- retrieved pc+1

- Return from interrupt

  IR (Branch | RTI | Rsrs=SP | Rdst=PC)

  MR=1

  MW=0

  WB=1

  ALU F=SP+1

  PC will be poped from stack fisrt - data read from stack will be saved back in PC

  (alu src = sp)

  PC <-- retrieved pc+1

Signals

- Reset

  IR all zero (one operand – no op)

  WB, MR, MW zero

  ALU no op

  PC <-- MEM[0]

- Interrupt

  IR (Branch | CALL | Rsrc=SP | Rdst=0)

  WB,MR=0

  MW=1

  ALU F=sp-1

  PC will be pushed to stack fisrt - PC -> memory data

  (alu src = sp

  mem[sp]=pc+1)

  PC <-- MEM [1]

  Flags preserved --> flages are saved in a temp register

Hazard Detection

As we have mentioned that we will fetch packet of instructions and execute those using concept of parallelism. The packet consists of only two instructions. These instructions may depend on each other and this dependency will cause a problem called hazards.

Hazards Types

- Structural Hazard
- Control Hazard
- Data Hazard

Structural Hazard

Structural hazard occurs in the following situation: -

- Instruction Fetch is conflicting with data memory access.
    - **Solution** is to use two separated memory (Data and Instruction).
- Register File is accessed by the instruction in the **decode** stage (reading) and at the same time other instruction in the **write-back** stage.
    - **Solution** is to ensure that writing is done in the falling edge and reading is done in the rising edge.
- Packets is executed in parallel and no sufficient resources
    - **Solution** is to duplicate the resources. We will have two ALUs, two MAR, two MDR, Register File has two signals **WB** (WB1 for the first instruction in the packet and WB2 for the second instruction in the packet).

Data Hazards Types

- Data Inner Hazard
- Data Outer Hazard

Data Inner Hazard

This type of hazards occurs when the packet itself is dependent. For example, if I have a load in the first instruction that uses some registers and the second instruction is using the same register to do some logic in the program. Assume code has only three instructions (**Inst1 to Inst3**) and the first, second cause data hazard because of dependency.

Solution is to stall the pipeline until this instruction will be written back in the register file after doing the logic of first instruction then start fetch a new packet and this new packet will contains the following **Inst2**--**Inst3.**

How this solution actually works behind the scenes?!

The Hazard Detection Unit **(HDU)** takes inputs from the previous, current packet. The inputs are the **IR** of the first packet instructions (Two IR) and the **IR** of the second packet instructions (Two IR). **Write after write may also make conflicts in the register.**

check_one = (DEC_Rsrc2 = DEC_Rdst1)
check_waw = (DEC_Rdst1 = DEC_Rdst2)
stall_long = check_one or check_waw

stall_long is responsible for making the first instruction in the packet resume executing (only it) for 3 clock cycles using crawling **stall_long** in the pipeline and take the result of it in the last buffer. How! By disabling the (IR Register) which means that we add to the pipeline **NOP** operations until the desired instruction finished and disabling the **PC** to keep the next instruction but we need after that to decrement it by 1 to fetch the correct packet.

Rst_IR = !(EXE_stall_long OR MEM_stall_long OR WB_stall_long)

enable_PC = !(EXE_stall_long OR MEM_stall_long)

pc = pc – 1 when WB_stall_long.

Then our new fetch will be correct isa.

Outer Hazard

This type of hazards occurs when an instruction (or more) in the packet depend on previous packet (**Load-Use/POP**). For example we've 4 instruction **Inst1 to Inst4**. Assume Inst2 is LDD Rdst, Rsrc and inst3 need to use Rdst. We've 4 cases **(Inst3 and Inst2), (Inst3 and Inst1), (Inst4 and Inst1), (Inst4 and Inst2)**

Solution is to stall the pipeline one cycle (stall bit is added in the control word).

How to stall the pipeline?!

- Disabling The **IR** & **PC** (Don't take the new instruction that the **PC** stored its address and don't increment **PC**).
- Reset The Buffer between the decoder stage and execution stage (act as **NOP**).

Stall Logic

```
stall = EXE_Memory_Read AND
                     ( EXE_Rdst1 = DEC_Rsrc1 )
                 OR  ( EXE_Rdst1 = DEC_Rsrc2 )
                 OR  ( EXE_Rdst2 = DEC_Rsrc1 )
                 OR  ( EXE_Rdst2 = DEC_Rsrc2 )
```

Control Hazard

Control hazard occurs in the following situation: -

- **Branch Taken (Jump)**
  - Static prediction is not to take the branch and continue the program **(NOT-TAKEN).**
  - We know that the instruction is branch in the decode stage and we raise one bit called **branch_taken** if taken or not.
  - In the execution if the branch_taken is raised then we need to flush else continue the programs (no problem).
- **Load Immediate**
  - Some instructions take a load value from the next instruction so I've to take this value and put it in the control signal then flushing the instruction (actually the instruction is a value not an instruction).

Solution is to flush the new instruction or (value).

How to flush the instruction or value?!

- Raise signal **(flush)** (acts as **NOP**), we may need to RST the IR (explained in the branch when occupies the 1st place in the packet).

There are two cases

- The branch instruction is in the first instruction of the packet
- The branch instruction is in the second instruction of the packet

Each case need a different handler. For the case #1 assume we have this code

- JZ Rdst
- Add R1, R2

So in the above example we need to flush the 2nd instruction in the packet and the next packet once we knew that the branch is taken.

For case #2

- Add R1, R2
- JZ Rds

But in the above example we've to complete the 1st instruction in the packet till the end then we back to case #1.

stall_long = (DEC_Inst2 = Branch)

Rst_IR = !(EXE_stall_long OR MEM_stall_long OR WB_stall_long)

enable_PC = !(EXE_stall_long OR MEM_stall_long)

pc = pc – 1 when WB_stall_long.

Flush Logic

Flush = immediate_load OR branch_taken.
PC = effective_address when branch_taken.
Rst_IR = branch_taken (flush second packet)

Immediate Load

There are two cases

- The load instruction is in the first instruction of the packet
- The load instruction is in the second instruction of the packet

We always need the load instruction (**LDM**) to be the first in the packet, so if it occupies in the second instruction in the packet we will make the slight same logic of the **data-inner-hazard**.

Each case need a different handler. For the case #1 assume we have this code

- LDM R5, 10
- 10

In the above example we raise only the signal of flush that will make the mux pass zeros to the alu.

For the case #2 assume we have this code

- Add R1 , R2
- LDM R5, 10

stall_long = (DEC_INST2 = LDM OR DEC_INST2 = SHL OR DEC_INST2 = SHR)

Rst_IR = !(EXE_stall_long OR MEM_stall_long OR WB_stall_long)

enable_PC = !(EXE_stall_long OR MEM_stall_long)

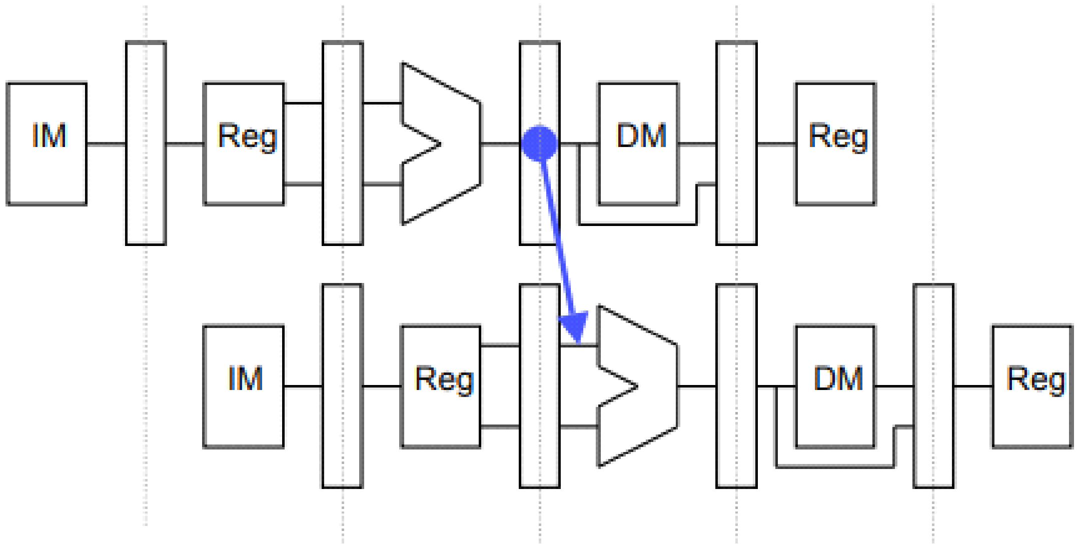
pc = pc – 1 when WB_stall_long.

Forwarding

Data Forwarding unit:

- A forwarding unit selects the correct ALU inputs for the EX stage to solve any data hazards.
    I. If there is no hazard, the ALU's operands will come from the register file (selectors = "000").
    II. If there is a hazard, the operands will come from either the EX/MEM or MEM/WB pipeline registers instead.
- The ALU sources will be selected by two multiplexers, with two selectors which controlled by the forwarding unit.
- There are two kinds of data hazards.
    I. EX/MEM data hazards.
    II. MEM/WB data hazards.

Detecting EX/MEM data hazard



An EX/MEM hazard occurs between one of the instructions of the issue currently in its EX stage and the previous two instructions in the previous issue if:

I. One of the previous instructions will write to the register file, and

II. One of the destination registers is one of the ALU source registers in the EX stage.

example on EX/MEM data hazard:

I1. add R1, R2

I2. sub <u>R6</u>, R7

I3. or R5, <u>R6</u>

I4. and R3, R4

here there is a data hazard between I3 instruction from the second packet of instructions and I2 instruction from the first one.

The ALU source comes from the pipeline register when necessary.

if (EX/MEM.RegWrite1 = 1 and EX/MEM.RegisterRd1 = ID/EX.RegisterRs1)

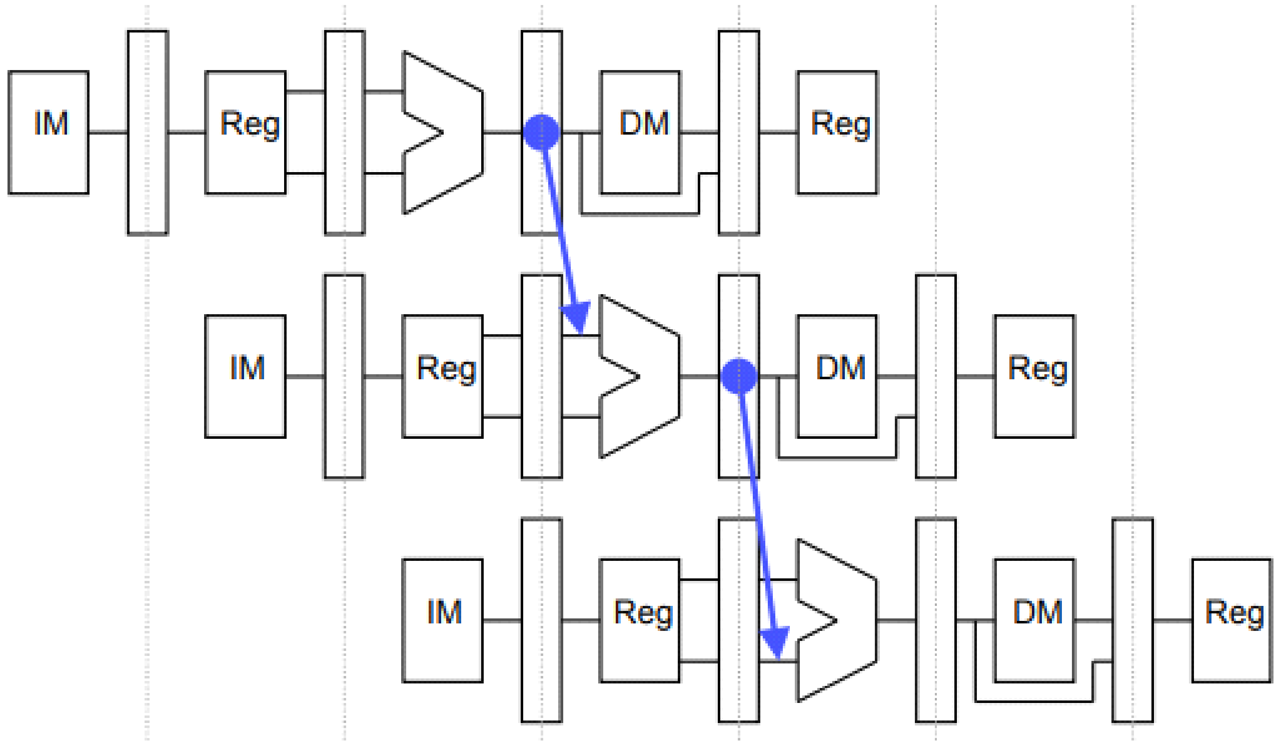
then forward_a1 = "001" //choose Rdst of the prev. alu of the first instruction in the previous packet .

if (EX/MEM.RegWrite2 = 1 and EX/MEM.RegisterRd2 = ID/EX.RegisterRs1)

then forward_a1 = "010" //choose Rdst of the prev. alu of the second instruction in the previous packet.

The same checks are done for the second operand of the first instruction of the packet, also for the second instruction in the packet.

<u>Detecting MEM/WB data hazards</u>



A MEM/WB hazard may occur between an instruction in the EX stage and

the instruction from two cycles ago.

For detecting and handling MEM/WB hazards for the first ALU source.

if (MEM/WB.RegWrite1 = 1 and MEM/WB.RegisterRd1 = ID/EX.RegisterRs1)

then forward_a1 = "011" //choose Rdst of the alu of the first instruction in the earlier packet.

if (MEM/WB.RegWrite1 = 2 and MEM/WB.RegisterRd2 = ID/EX.RegisterRs1)

then forward_a1 = "100" //choose Rdst of the alu of the second instruction in the earlier packet.

The same checks are done for the second operand of the first instruction of the packet, also for the second instruction in the packet.