Pipelining Implementation

Team no : 19

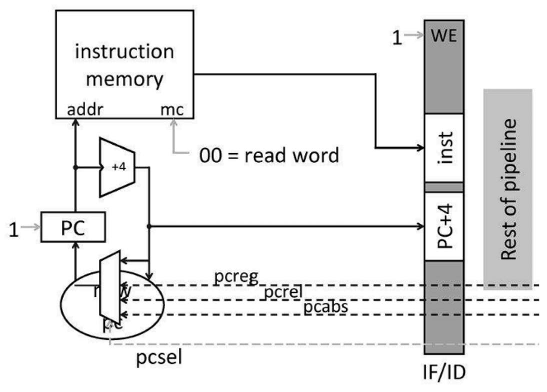
1. **Gangineni Yaswanth** - 2023102007
2. **Nethavath Praveen**-2023102013
3. **Kotha Phanindra Manoj Kumar** - 2023102008

Introduction

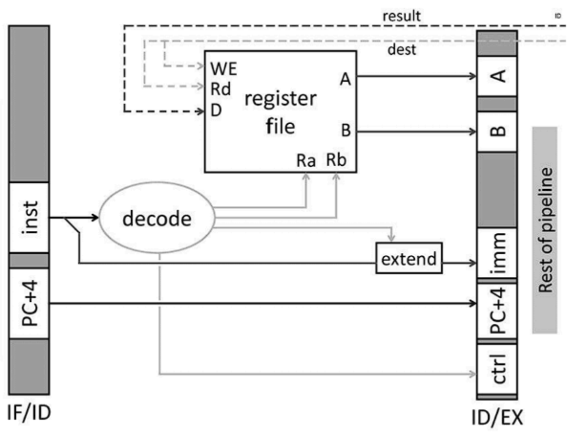
In the pipeline process implementation, the execution of instructions is broken down into several sequential stages, with each stage handling a different part of the instruction execution process. Instead of waiting for one instruction to complete all stages before starting the next, multiple instructions can be in different stages of execution simultaneously.

Stages in a Pipelined RISC-V Processor

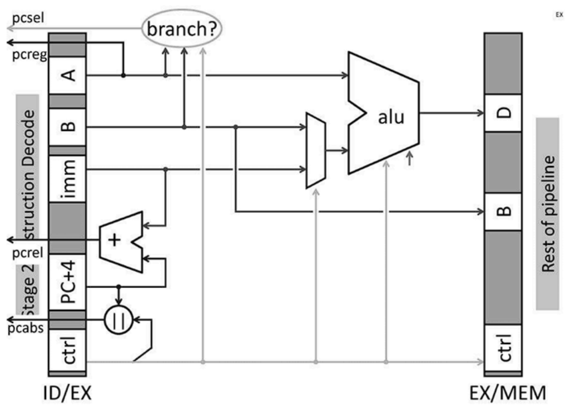
- **IF : Instruction Fetch from Memory**



- The processor retrieves the next instruction from memory using the current Program Counter (PC).
- The PC is then incremented to point to the next instruction.
- This stage may include simple branch prediction.
- ID : **Instruction Decode & Register Read**

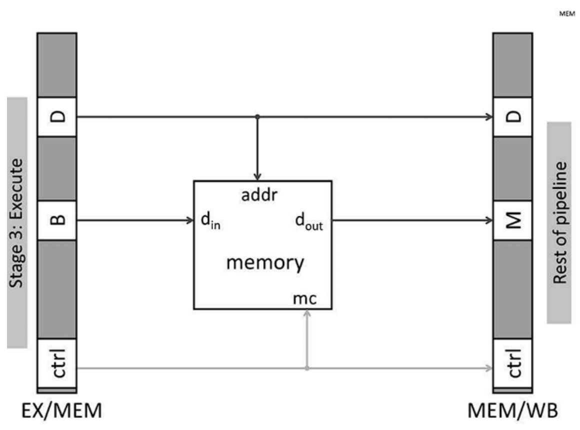


  - The instruction is decoded to determine its type and required operations.
  - Register values needed for execution are read from the register file.
  - Control signals are generated for subsequent stages.
- **EX: Execute Operation or Calculate Address**



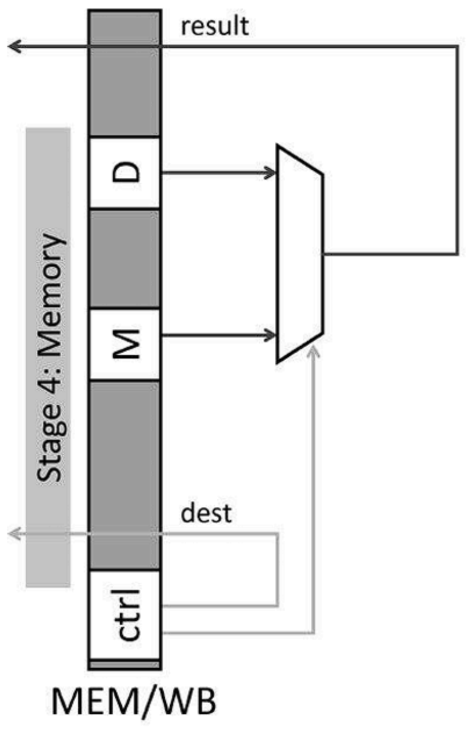
- For arithmetic/logic instructions: The ALU performs the specified operation.
- For memory instructions: The memory address is calculated.
- For branch instructions: The branch condition is evaluated and the target address is calculated.
- This stage implements the core computation of the instruction.

- **MEM: Access Memory Operand**



  - For load instructions: Data is read from memory at the calculated address.
  - For store instructions: Data is written to memory at the calculated address.
  - For non-memory instructions: This stage is effectively bypassed.

- **WB: Write Result Back to Register**



  - Results from ALU operations or memory loads are written back to the destination reg ister.
  - This completes the instruction execution.

Pipeline Registers in a 5-Stage RISC-V Pipeline

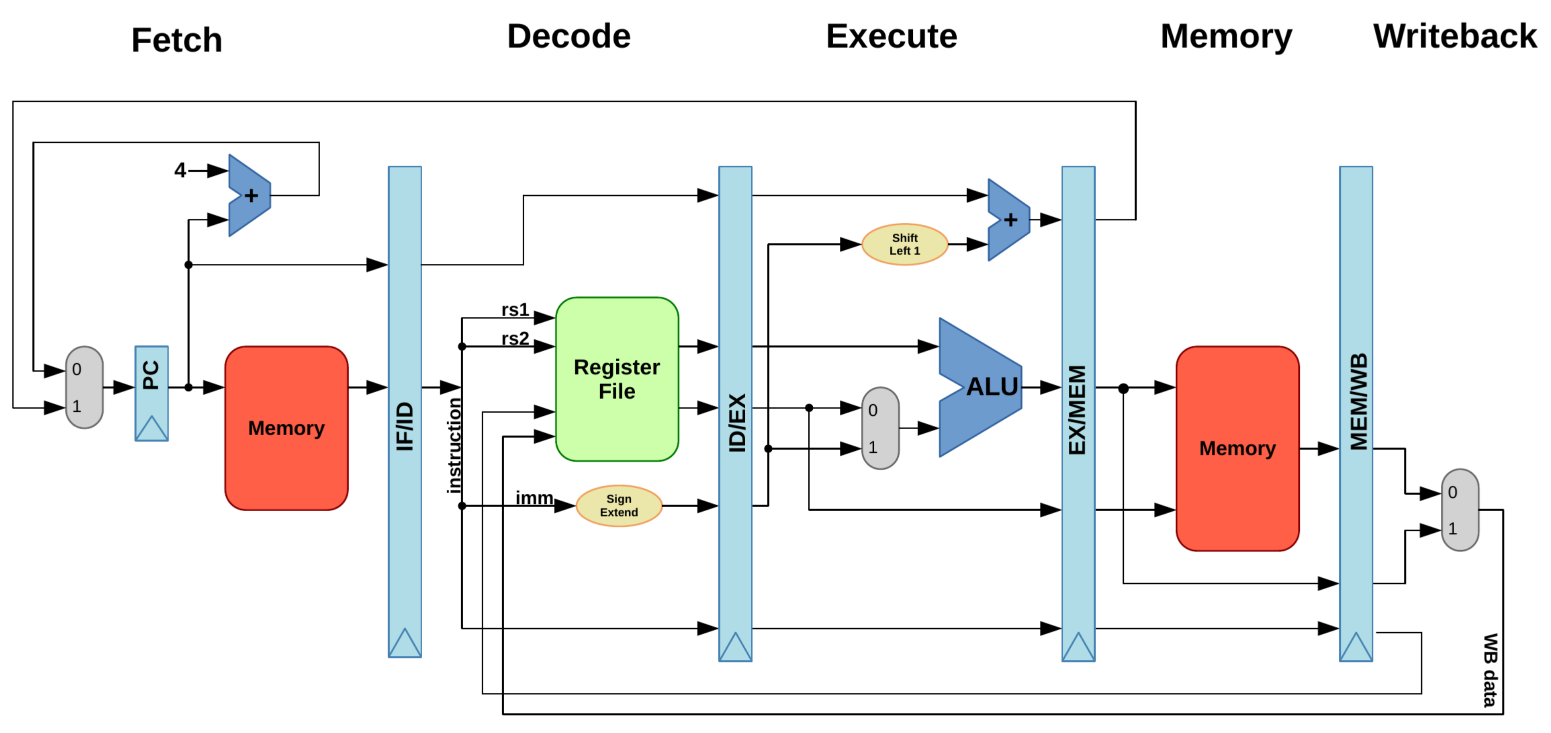
In a 5-stage RISC-V pipeline, there are four sets of pipeline registers:

- **IF/ID Register (Between Fetch and Decode)**
    - Stores the fetched instruction.
    - Stores the incremented PC value (PC+4).
    - May store the PC of the current instruction (for branch calculation).
- **ID/EX Register (Between Decode and Execute)**
    - Stores register values read from the register file.
    - Stores immediate values.
    - Stores control signals for Execute, Memory, and Writeback stages.
    - Stores register destination addresses.
    - Stores function codes for the ALU.
    - Stores the PC value (for branch/jump instructions).
- **EX/MEM Register (Between Execute and Memory)**
    - Stores ALU result/computed address.
    - Stores data to be written to memory (for store instructions).
    - Stores branch result (taken/not taken).
    - Stores control signals for Memory and Writeback stages.
    - Stores register destination address.

MEM/WB Register (Between Memory and Writeback)

- Stores data read from memory (for load instructions).
- Stores ALU result that bypassed memory.
- Stores control signals for the Writeback stage.
- Stores register destination address.

Registers in Pipelined RISC V Processor :



Example Content in Pipeline Registers :

For an ***add x3, x1, x2*** instruction flowing through the pipeline:

- IF/ID Register :

        - Instruction: ***add x3, x1, x2*** (32-bit binary).
        - ***PC+4*** : Address of the next instruction.
    - ID/EX Register :
        - Register values: Contents of x1 and x2.
        - Control signals: ALU should add, no memory operation, register write enabled.
        - Destination register: x3.
        - ALU function: ADD.
    - EX/MEM Register :
        - ALU result: Sum of x1 and x2.
        - Control signals: No memory operation, register write enabled.
        - Destination register: x3.
    - MEM/WB Register :
        - Write data: Sum of x1 and x2 (same as ALU result)
        - Control signals: Register write enabled.
        - Destination register: x3.

Importance of Registers in Handling Hazards

Pipeline registers are critical for implementing hazard detection and resolution:

- **Forwarding (Bypassing) :** The pipeline registers hold information needed to detect data dependencies. When a dependency is detected, forwarding logic can route data from one pipeline register directly to a functional unit, bypassing the register file.
- **Stalling :** Pipeline registers can be controlled to hold their values (not update) when a stall is required, effectively freezing parts of the pipeline.
- **Flushing : A**fter a branch misprediction, certain pipeline registers can be cleared/invalidated to flush incorrect speculative instructions.

Instruction Execution Flow

- **Clock Cycle 1: Instruction Fetch (IF) :**
    - Program Counter (PC) provides address of instruction.
    - Instruction memory is accessed.
    - Instruction is fetched.
    - PC is updated to point to the next instruction
- Clock Cycle 2: Instruction Decode (ID) :
    - Instruction is decoded.
    - Required registers are read from the register file.
    - Immediate values are extended.
    - Control signals are generated.
    - Hazard detection occurs.
- Clock Cycle 3: Execute (EX)
    - ALU performs operation based on opcode.
    - For R-type: Register-register operation (add, sub, etc.).
    - For I-type: Register-immediate operation or address calculation.

  - For branches: Condition evaluation and target address calculation.
  - For jumps: Target address calculation
- Clock Cycle 4: Memory Access (MEM)
  - For load instructions: Memory is read using address from EX stage.
  - For store instructions: Data is written to memory.
  - For other instructions: Stage is passed through
- Clock Cycle 5: Write Back (WB) :
  - Results from ALU or memory are written back to the destination register.
  - Instruction execution is now complete

Cycle	Instruction 1	Instruction 2	Instruction 3	Instruction 4	Instruction 5
1	IF				
2	ID	IF			
3	EX	ID	IF		
4	MEM	EX	ID	IF	
5	WB	MEM	EX	ID	IF
6		WB	MEM	EX	ID
7			WB	MEM	EX
8				WB	MEM

Explanation of each new Module implemented in Verilog

Register between Stages

IF/ID Pipeline Register :
This register captures the output of the Fetch stage to pass to the Decode stage.

Register Components
• IF ID Instruction (32 bits): Holds the fetched instruction.
• IF ID PC (64 bits): Holds the current program counter value.

Clock and Control Behavior

• Triggered on the negative edge of the clock (@(negedge Clk)).
• Reset behavior: Clears both registers to zero when reset is high.
• Flush behavior: When Flush signal is high (branch taken), both registers are cleared to zero.
• Stall behavior: When Stall signal is high (hazard detected), maintains previous values.

Inputs
• Instruction: Current instruction from instruction memory.
• PC: Current program counter value.
• Stall: Signal from hazard detection unit.
• Flush: Signal from control hazard unit.

Outputs
• IF ID Instruction: Used by Decode stage to extract register addresses and immediates.
• IF ID PC: Used for branch target calculation.

ID/EX Pipeline Register

This register captures the output of the Decode stage to pass to the Execute stage.

Register Components

Data Registers:

- ID EX Instruction (32 bits): Passed instruction.
- ID EX PC (64 bits): Program counter value.
- ID EX A (64 bits): Data from first source register (Rs1).
- ID EX B (64 bits): Data from second source register (Rs2).
- ID EX C (64 bits): Immediate value.
- ID EX rd (5 bits): Destination register address.

Control Signals:

- ID EX ALUOp (2 bits): ALU operation type.
- ID EX ALUSrc (1 bit): Selects between register or immediate for ALU input.
- ID EX MemRead (1 bit): Memory read enable.
- ID EX MemWrite (1 bit): Memory write enable.
- ID EX MemtoReg (1 bit): Selects between ALU result or memory data for writeback.
- ID EX RegWrite (1 bit): Register write enable.
- ID EX Branch (1 bit): Branch instruction indicator.

Clock and Control Behavior

- Triggered on the negative edge of the clock (@(negedge Clk)).
- Reset behavior: All registers cleared to zero.
- Flush behavior: All registers cleared to zero (effectively inserting a NOP).
- Stall behavior: Data registers maintain values, but control signals are disabled (set to 0).

Inputs

- IF ID Instruction: Instruction from IF/ID register.
- IF ID PC: Program counter from IF/ID register.
- A, B: Register file outputs (read data).
- C: Immediate value generated by decode.
- Control signals from control unit.

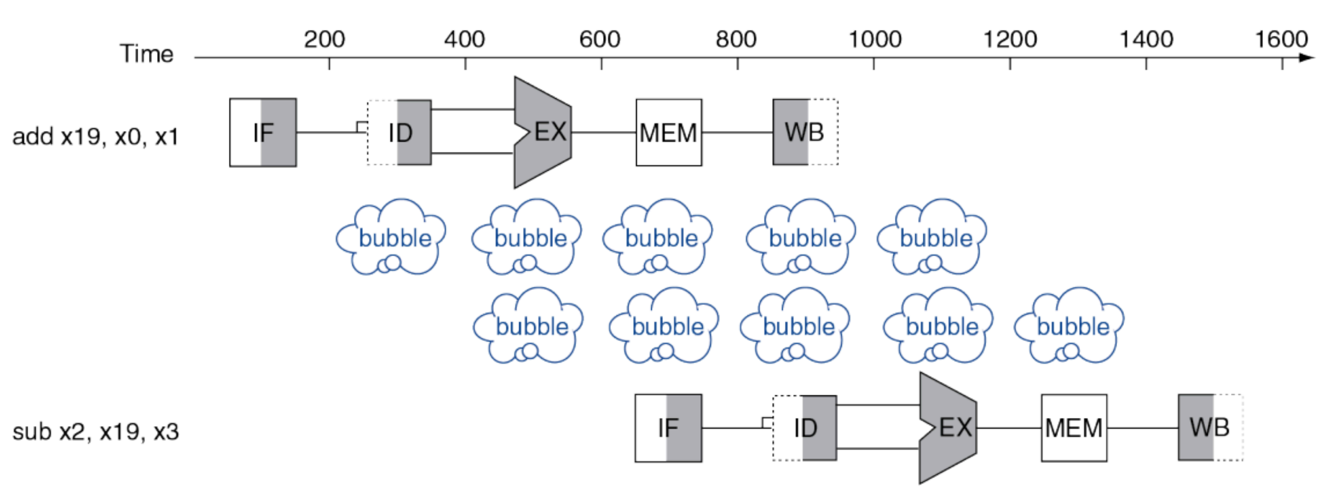
Outputs

- Used by Execute stage for ALU operations, branch calculations.
- Used by hazard detection unit (ID EX rd and ID EX MemRead)

Data Hazards

- **Cause:** An instruction depends on completion of data access by a previous instruction
- **Example:**

```
add  x19, x0, x1
sub  x2, x19, x3
```

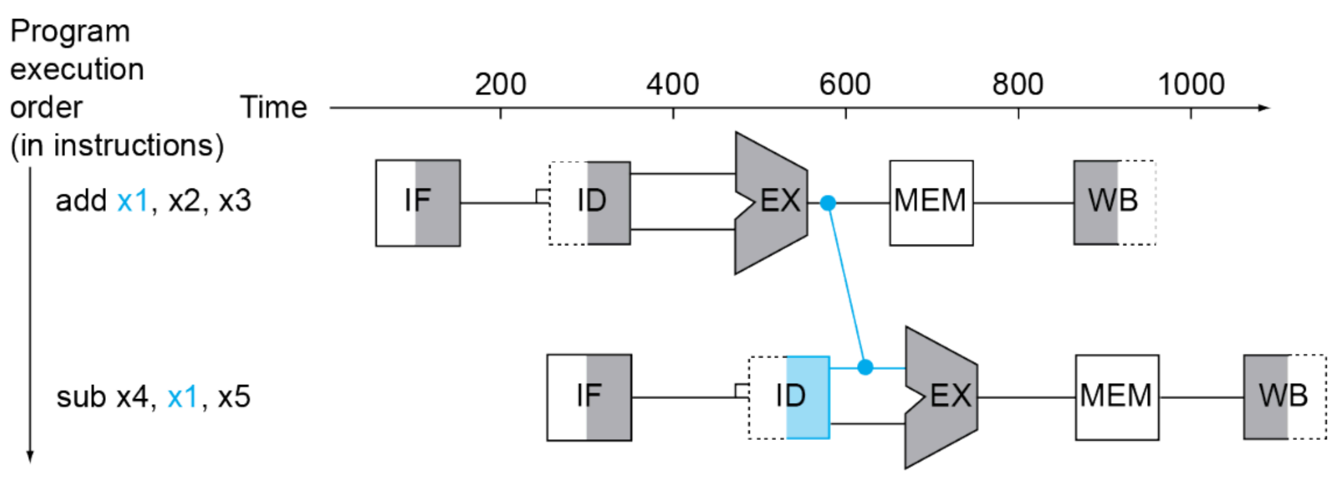


Solution:

Forwarding (Aka By Passing)

- Use result when it is computed
- Don’t wait for it to be stored in a register
- Requires extra connections in the data path

Stalling (Pipeline Stall)



- If forwarding is not possible, insert a **NOP (stall cycle)** to wait for the data.
- this case occurs when load use data hazard occurs.

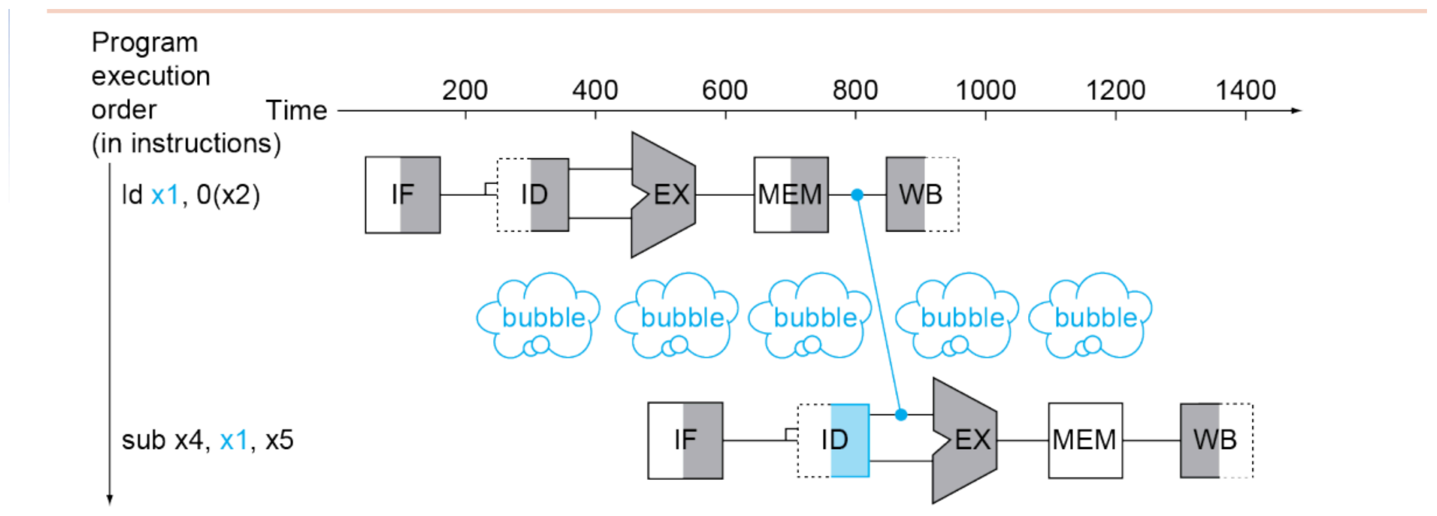
Load Use Data Hazard

occurs:

- Can’t always avoid stalls by forwarding
- If value not computed when needed
- Can’t forward backward in time!

Example:

```
ld x1,0(x2)
sub x4,x1,x5
```



- stall will occur we have to stop the the instruction for one cycle.

Control Hazards

- **Cause:** When the pipeline does not know **which instruction to fetch next**, usually caused by **branch instructions** (e.g., `beq`, `bne`, `jal`).
- **Example:**

```
BEQ x1, x2, LABEL  // Branch if x1 == x2
ADD x3, x4, x5     // Next instruction (may need to be flushed)
```

- If the **branch is taken**, the instruction already in the pipeline (`ADD x3, x4, x5`) **must be discarded**.
- This **wastes cycles** and reduces performance.

🔷 Solution:

Branch Prediction

- The processor predicts the **most likely path** and **speculatively executes instructions**.
- If the prediction is correct → No penalty.
- If the prediction is wrong → Flush the incorrect instructions and restart.

Forwarding Unit :

```
module ForwardingUnit
(
    input wire [4:0] rs1,
    input wire [4:0] rs2,
    input wire [4:0] rd3,
    input wire [4:0] rd4,
    input wire WB3,
    input wire WB4,
    output reg [1:0] ForwardA,
    output reg [1:0] ForwardB
);

always @(*)
begin
    //No forwarding condn
    ForwardA = 2'b00;
    ForwardB = 2'b00;

    // Check if rd3 is the destination and should be forwarded
    if (WB3 && (rd3 != 0)) begin
        if (rd3 == rs1) ForwardA = 2'b10;
        if (rd3 == rs2) ForwardB = 2'b10;
    end

    // Check if rd4 is the destination and should be forwarded
    if (WB4 && (rd4 != 0)) begin
      if (rd4 == rs1) ForwardA = 2'b01;
      if (rd4 == rs2) ForwardB = 2'b01;
    end
  end

endmodule
```

- **Inputs:**
  - `rs1`, `rs2`: Source registers 1 and 2 which are operands used in instructions.
  - `rd3`, `rd4`: Destination registers 3 and 4 which store results of executed instructions.
  - `WB3`, `WB4`: Signals indicating if there is a write-back (update) to registers rd3 and rd4.
- **Outputs:**
  - `ForwardA`, `ForwardB`: Control signals indicating if source registers rs1 and rs2 should be forwarded from rd3 or rd4.
- **Behavior:**
  - The `always @(*)` block continuously evaluates changes in inputs.
  - It initializes `ForwardA` and `ForwardB` to no forwarding (`2'b00`).
  - It checks if there's a write-back (`WB3`, `WB4`) and if the corresponding destination register (`rd3`, `rd4`) is not zero.
  - If true, it checks if the source registers (`rs1`, `rs2`) match the destination register (`rd3`, `rd4`) and updates `ForwardA` and `ForwardB` accordingly:
    - `2'b10` indicates forward from rd3,
    - `2'b01` indicates forward from rd4.

conflict :

In case both EX/MEM and MEM/WB can forward to the same register, prioritize EX/MEM (Taking Care of Double Data Hazard)

Example :

ADD X3, X1, X2

ADD X5 ,X3,X4

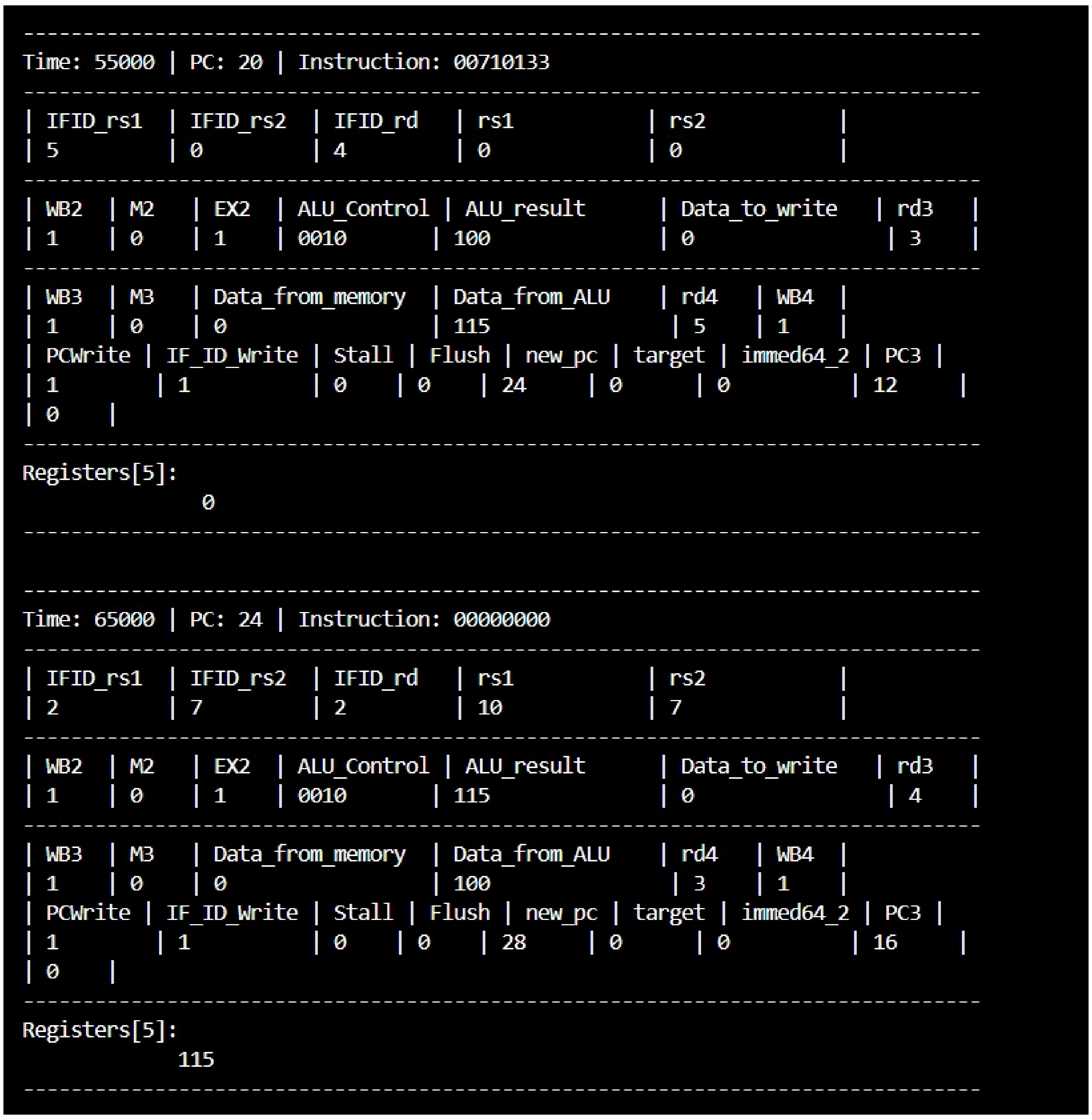
Initially

X1=5, X2=10 ,X4=100

Output :

X5=115

Terminal Output :



```
--------------------------------------------------------------------------------
Time: 55000 | PC: 20 | Instruction: 00710133
--------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1            | rs2            |
| 5         | 0         | 4         | 0              | 0              |
--------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result      | Data_to_write   | rd3   |
| 1    | 0    | 1    | 0010        | 100             | 0                | 3    |
--------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU     | rd4    | WB4  |
| 1    | 0    | 0                 | 115                | 5     | 1    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1          | 0     | 0     | 24     | 0      | 0          | 12     |
| 0    |
--------------------------------------------------------------------------------
Registers[5]:
               0
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
Time: 65000 | PC: 24 | Instruction: 00000000
--------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1            | rs2            |
| 2         | 7         | 2         | 10             | 7              |
--------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result      | Data_to_write   | rd3   |
| 1    | 0    | 1    | 0010        | 115             | 0                | 4    |
--------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU     | rd4    | WB4  |
| 1    | 0    | 0                 | 100                | 3     | 1    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1          | 0     | 0     | 28     | 0      | 0          | 16     |
| 0    |
--------------------------------------------------------------------------------
Registers[5]:
             115
--------------------------------------------------------------------------------
```

Hazard Detection Unit

The **Hazard Detection Unit (HDU)** is a key component in a **pipelined processor** that ensures correct instruction execution by handling two types of pipeline hazards:

1. **Load-Use Data Hazard**
2. **Control Hazard (Branch Hazard)**

The **Hazard Detection Unit (HDU)** prevents incorrect execution in a pipelined RISC-V processor by:

1. **Detecting Load-Use Hazards** – Stalls the pipeline if a load instruction is followed by an instruction needing its result.
2. **Handling Control Hazards** – Flushes the pipeline when a branch is taken to avoid executing the wrong instructions.

```
module HazardDetectionUnit
(
  input wire ID_EX_MemRead,         // Load instruction in EX stage
  input wire [4:0] ID_EX_RegisterRd, // Destination register in EX stage
  input wire [4:0] IF_ID_RegisterRs1, // Source register 1 in ID stage
  input wire [4:0] IF_ID_RegisterRs2, // Source register 2 in ID stage
  input wire BranchTaken,           // Combined signal: Branch & ALUZero
```

```
    output reg Stall,       // Stall signal for load-use hazard
    output reg PCWrite,     // Control signal to stop PC update
    output reg IF_ID_Write, // Control signal to stop instruction fetch
    output reg Flush        // Control signal to flush incorrect instruction
);

always @(*) begin
    // Default values (no stall or flush)
    Stall = 0;
    PCWrite = 1;
    IF_ID_Write = 1;
    Flush = 0;

    // Handle Load-Use Hazard
    if (ID_EX_MemRead && (ID_EX_RegisterRd == IF_ID_RegisterRs1 || ID_EX_RegisterRd == IF_ID_RegisterRs2)) be
        Stall = 1;
        PCWrite = 0;
        IF_ID_Write = 0;
    end

    // Handle Control Hazard (Branch)
    if (BranchTaken) begin // Branch Taken
        Flush = 1;   // Flush IF/ID pipeline register
        PCWrite = 1; // Allow PC to update to branch target
        IF_ID_Write = 1; // Continue fetching
    end
end

endmodule
```

- **Inputs:**
  - `ID_EX_MemRead`: Indicates if the current instruction in the EX stage is a **load** instruction (e.g., `ld`).
  - `ID_EX_RegisterRd`: Destination register of the EX stage instruction.
  - `IF_ID_RegisterRs1`, `IF_ID_RegisterRs2`: Source registers of the instruction currently in the ID stage.
  - `BranchTaken`: Signal that becomes high when a branch is taken (based on branch condition & ALU result).
- **Outputs:**
  - `Stall`: Stops instruction execution to prevent a load-use hazard.
  - `PCWrite`: Controls whether the program counter (PC) should update.
  - `IF_ID_Write`: Controls whether the instruction fetch (IF/ID pipeline register) should update.
  - `Flush`: Clears the IF/ID pipeline register to remove incorrect instructions after a branch.

*By default, the pipeline continues execution with **no stall or flush**.*

A
Load-Use hazard occurs when an instruction needs data from a **load (`ld`) instruction** that is still in the EX stage.

A
control hazard occurs when a **branch (`beq`, `bne`) is taken**, but the **next instruction was already fetched** incorrectly.

Hazard Type	Detection Condition	Solution	Effect on Pipeline
Load-Use Hazard	ID_EX_MemRead+ID_EX_RegisterRd==`IF_ID_RegisterRs1`, `IF_ID_RegisterRs2`	Stall (`Stall = 1`)	1-cycle stall

Branch Hazard	`BranchTaken` ==1	Flush (`Flush = 1`)	Removes incorrect instruction

- If the instruction in the **EX stage** is a **load instruction (** `MemRead` **is high)** and its destination register (`ID_EX_RegisterRd`) matches either of the **source registers** (`IF_ID_RegisterRs1` or `IF_ID_RegisterRs2`) of the instruction in the **ID stage**, a **stall** is required to avoid incorrect data being used before it is available.
- **Actions Taken:**
  - `Stall = 1` : Stop execution of the current instruction.
  - `PCWrite = 0` : Halt program counter updates.
  - `IF_ID_Write = 0` : Stop fetching new instructions.
- If a **branch is taken (** `BranchTaken` **is high)**, it means the previous instruction decided to change the program counter (PC).
- The next instruction fetched is **incorrect**, so we must **flush it** from the pipeline.
- **Actions Taken:**
  - `Flush = 1` : Remove incorrect instruction from IF/ID pipeline register.
  - `PCWrite = 1` : Allow PC to update to branch target.
  - `IF_ID_Write = 1` : Allow fetching new instructions after branch

Load Hazard :

example testcase :

LD X8 , 10(X7)

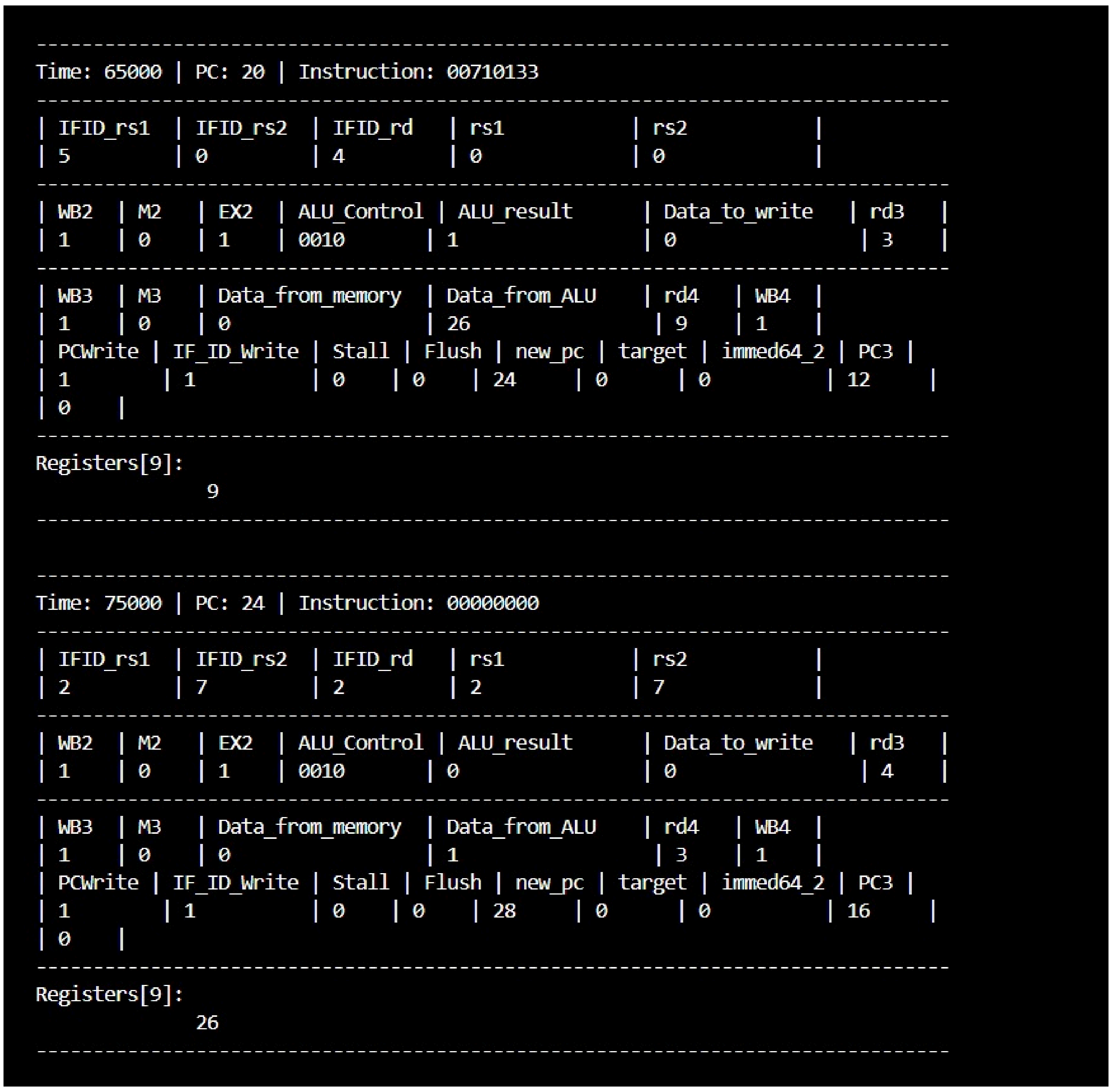
ADD X9,X8,X10

initially X7=7, X10 =10

Data_memory[17]=16

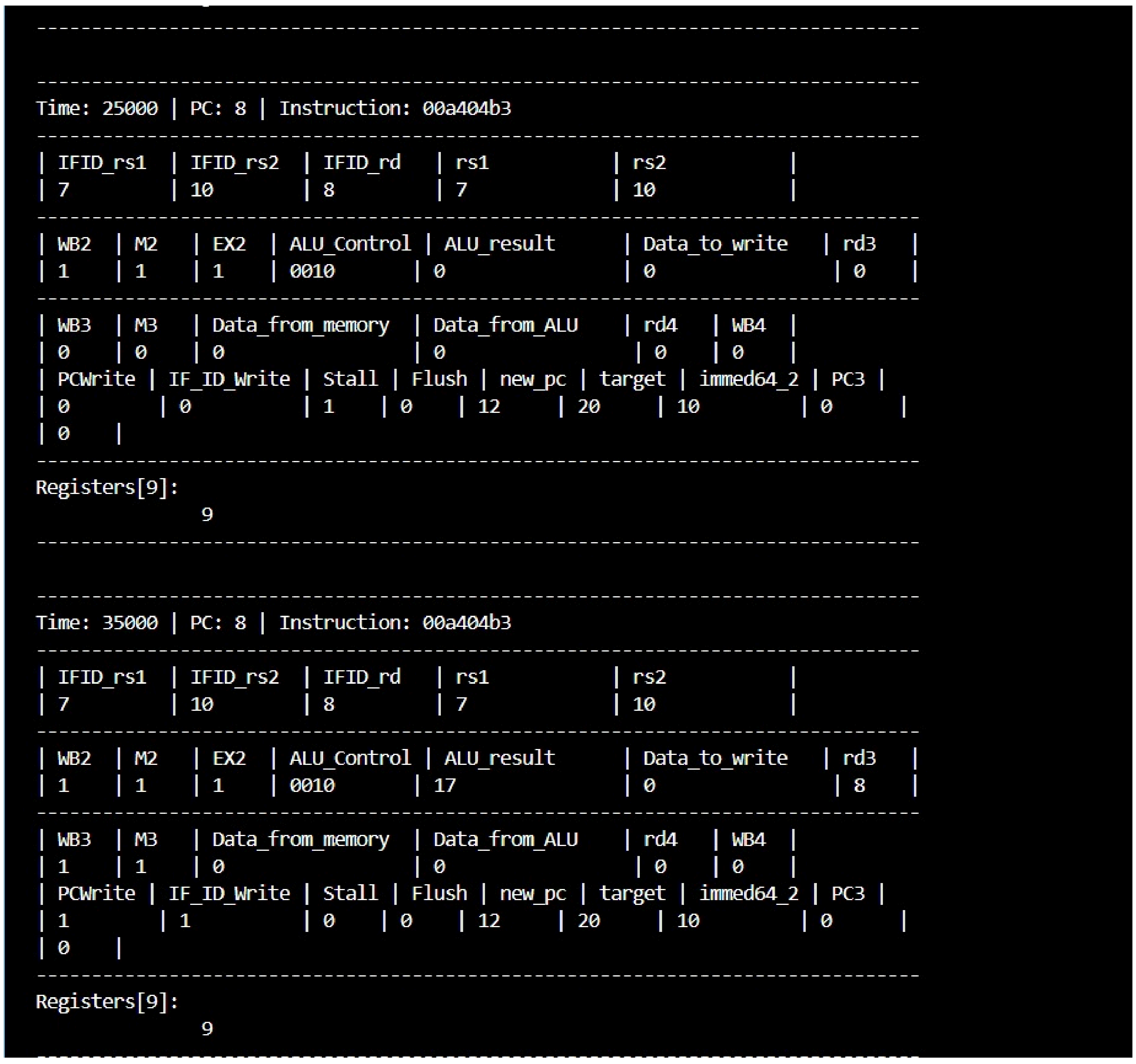
Output : X9 = 10+16 =26

OUT PUT :



```
-------------------------------------------------------------------------------
Time: 65000 | PC: 20 | Instruction: 00710133
-------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1             | rs2             |
| 5         | 0         | 4         | 0               | 0               |
-------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result      | Data_to_write   | rd3   |
| 1    | 0    | 1    | 0010        | 1               | 0                | 3    |
-------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU   | rd4    | WB4  |
| 1    | 0    | 0                 | 26               | 9     | 1    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1           | 0    | 0     | 24     | 0      | 0          | 12     |
| 0    |
-------------------------------------------------------------------------------
Registers[9]:
              9
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
Time: 75000 | PC: 24 | Instruction: 00000000
-------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1             | rs2             |
| 2         | 7         | 2         | 2               | 7               |
-------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result      | Data_to_write   | rd3   |
| 1    | 0    | 1    | 0010        | 0               | 0                | 4    |
-------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU   | rd4    | WB4  |
| 1    | 0    | 0                 | 1                | 3     | 1    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1           | 0    | 0     | 28     | 0      | 0          | 16     |
| 0    |
-------------------------------------------------------------------------------
Registers[9]:
             26
-------------------------------------------------------------------------------
```

PC is Constant for one clock Cycle which indicates a stall :

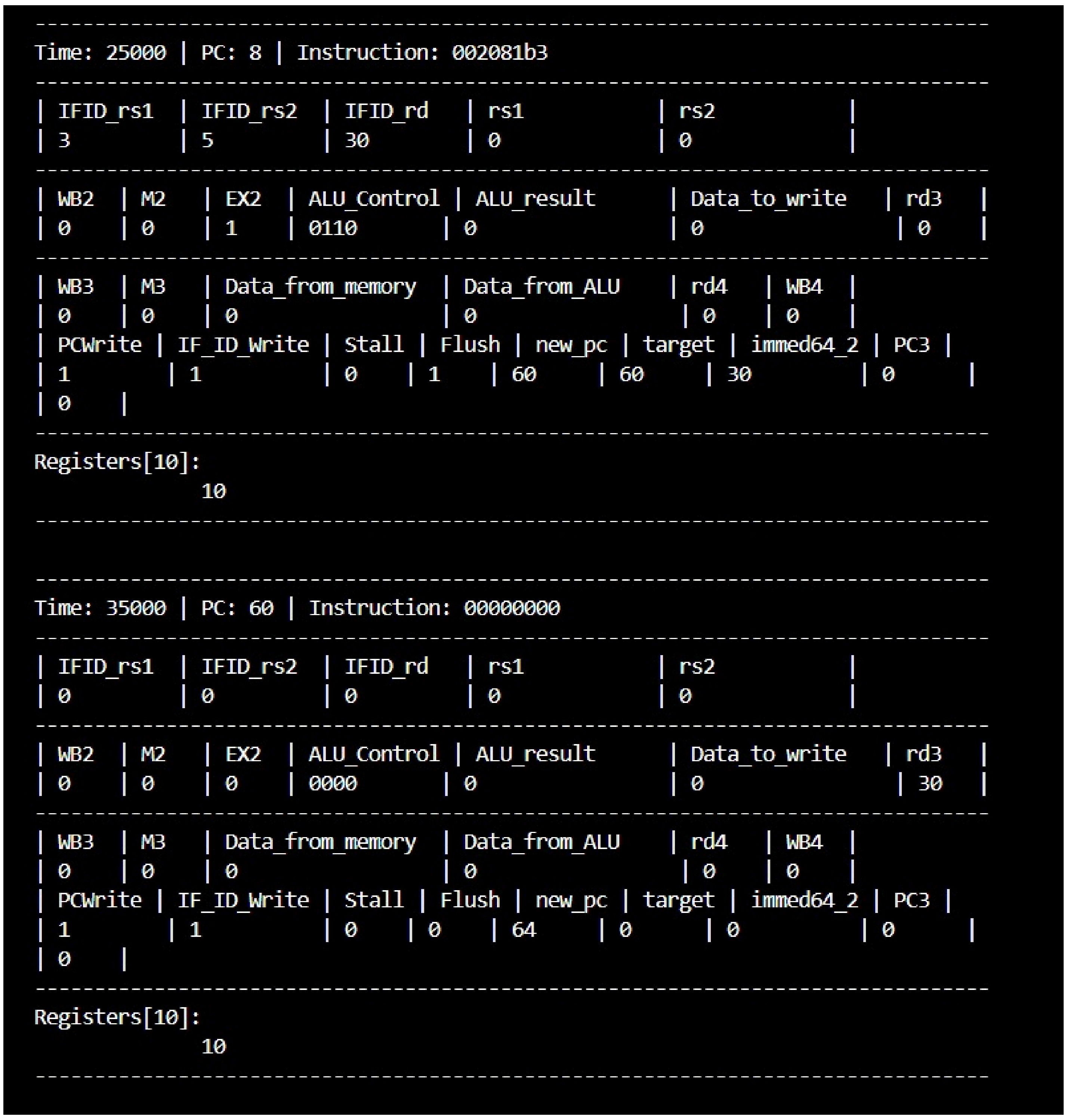


```
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
Time: 25000 | PC: 8 | Instruction: 00a404b3
-------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1             | rs2             |
| 7         | 10        | 8         | 7               | 10              |
-------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result      | Data_to_write   | rd3   |
| 1    | 1    | 1    | 0010        | 0               | 0                | 0    |
-------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU   | rd4    | WB4  |
| 0    | 0    | 0                 | 0                | 0     | 0    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 0        | 0           | 1    | 0     | 12     | 20     | 10         | 0      |
| 0    |
-------------------------------------------------------------------------------
Registers[9]:
              9
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
Time: 35000 | PC: 8 | Instruction: 00a404b3
-------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1             | rs2             |
| 7         | 10        | 8         | 7               | 10              |
-------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result      | Data_to_write   | rd3   |
| 1    | 1    | 1    | 0010        | 17              | 0                | 8    |
-------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU   | rd4    | WB4  |
| 1    | 1    | 0                 | 0                | 0     | 0    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1           | 0    | 0     | 12     | 20     | 10         | 0      |
| 0    |
-------------------------------------------------------------------------------
Registers[9]:
              9
-------------------------------------------------------------------------------
```

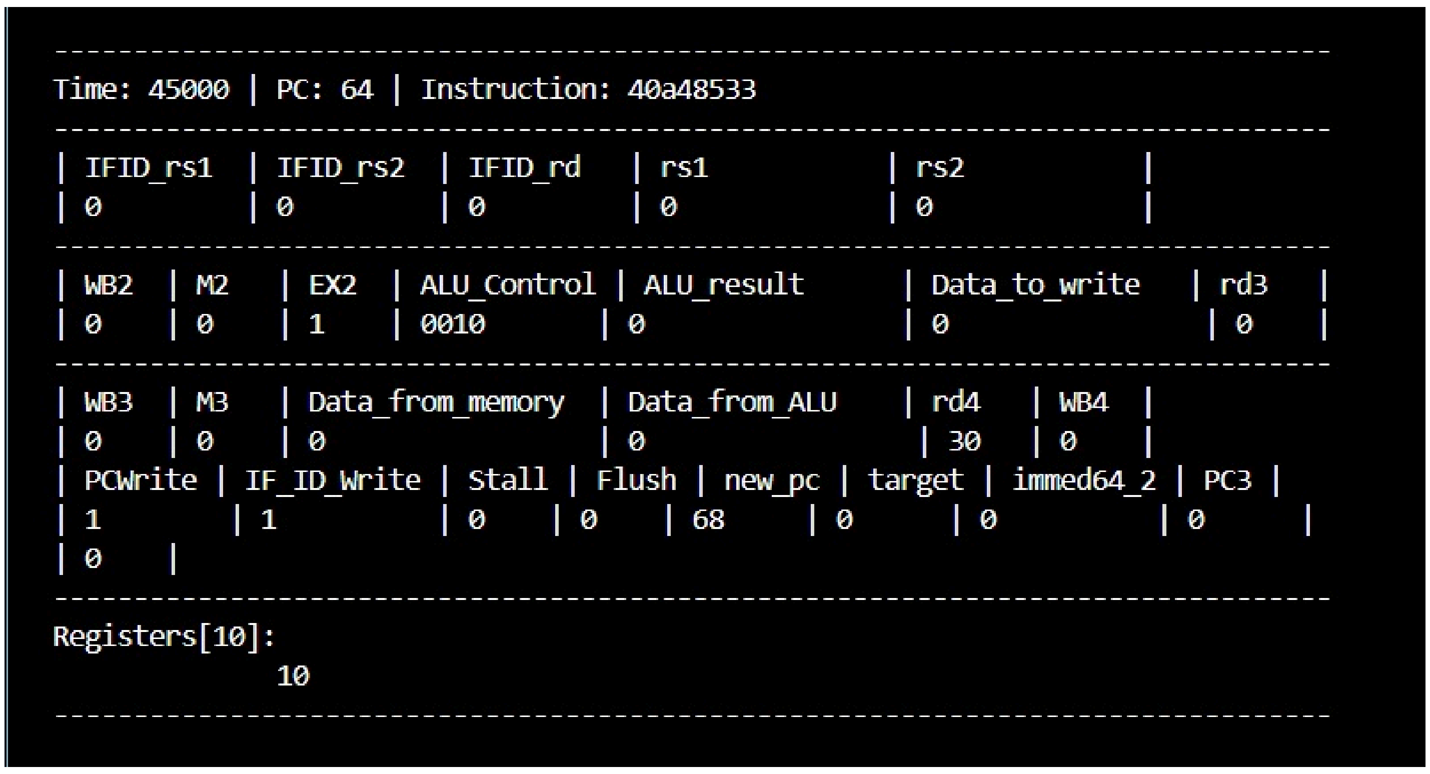
Control Hazard :

```
beq X3 , X5 , 30
ADD X3 , X2 , X1
ADD X5 , X3 , X4
EXIT:
AT 60th adress
SUB X10 ,X9 , X10
EXIT :
```

All the Reg Values are set 0 because Flush i s 1

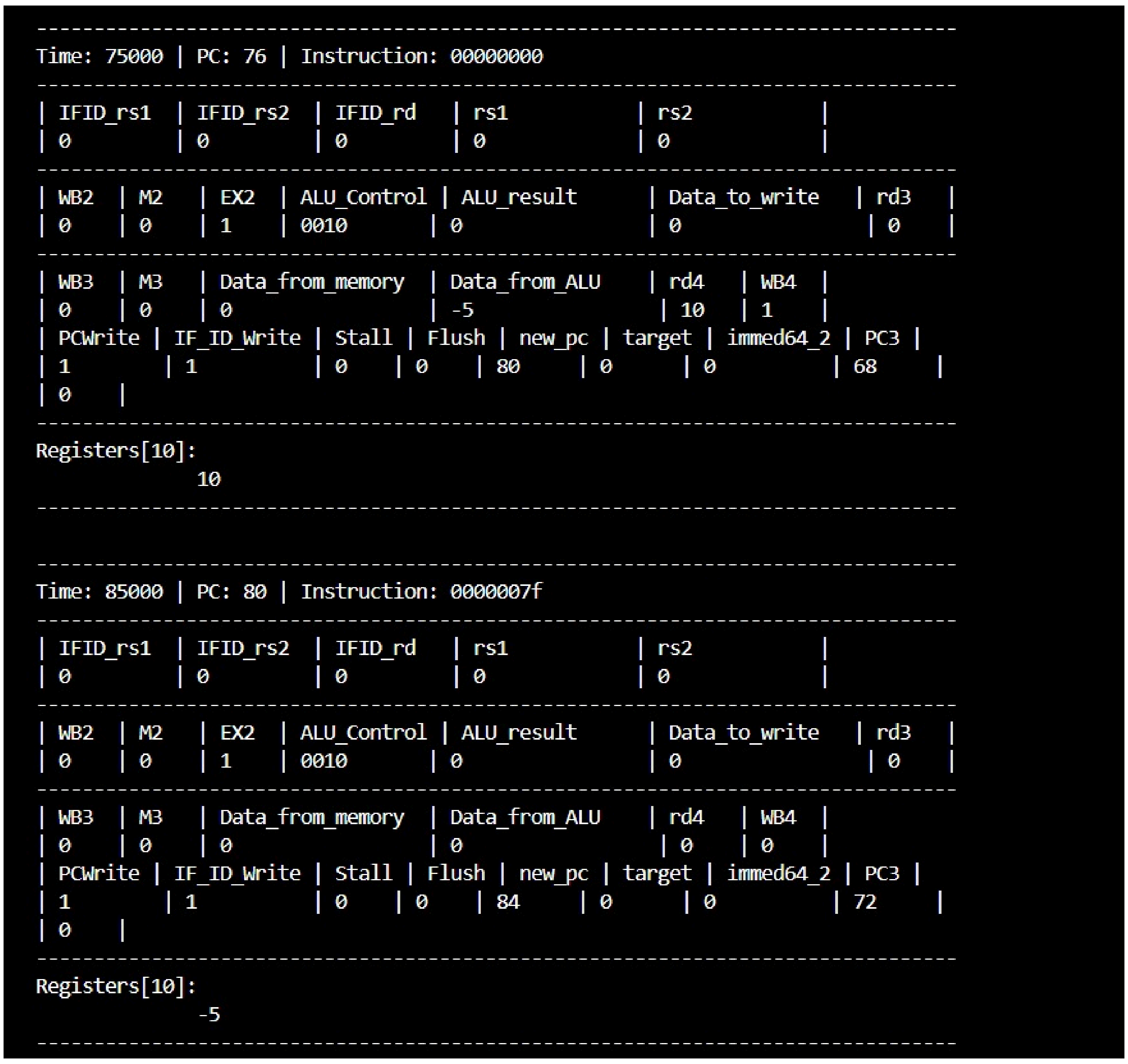


```
--------------------------------------------------------------------------------
Time: 25000 | PC: 8 | Instruction: 002081b3
--------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1             | rs2             |
| 3         | 5         | 30        | 0               | 0               |
--------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result        | Data_to_write   | rd3  |
| 0    | 0    | 1    | 0110        | 0                 | 0                 | 0    |
--------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU     | rd4    | WB4  |
| 0    | 0    | 0                 | 0                  | 0    | 0    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1           | 0    | 1    | 60      | 60      | 30          | 0      |
| 0    |
--------------------------------------------------------------------------------
Registers[10]:
              10
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
Time: 35000 | PC: 60 | Instruction: 00000000
--------------------------------------------------------------------------------
| IFID_rs1  | IFID_rs2  | IFID_rd   | rs1             | rs2             |
| 0         | 0         | 0         | 0               | 0               |
--------------------------------------------------------------------------------
| WB2  | M2   | EX2  | ALU_Control | ALU_result        | Data_to_write   | rd3  |
| 0    | 0    | 0    | 0000        | 0                 | 0                 | 30   |
--------------------------------------------------------------------------------
| WB3  | M3   | Data_from_memory  | Data_from_ALU     | rd4    | WB4  |
| 0    | 0    | 0                 | 0                  | 0    | 0    |
| PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
| 1        | 1           | 0    | 0    | 64      | 0       | 0           | 0      |
| 0    |
--------------------------------------------------------------------------------
Registers[10]:
              10
--------------------------------------------------------------------------------
```



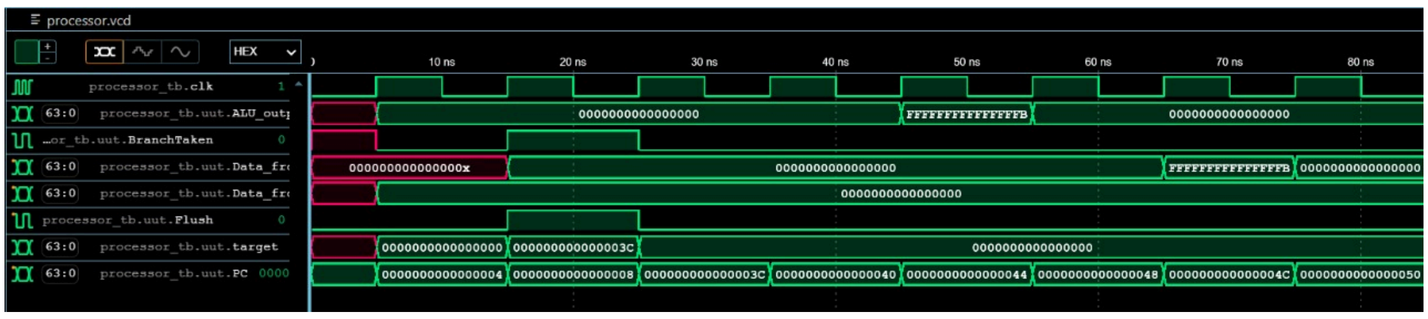
```
 ------------------------------------------------------------------------------
 Time: 45000 | PC: 64 | Instruction: 40a48533
 ------------------------------------------------------------------------------
 | IFID_rs1   | IFID_rs2   | IFID_rd    | rs1             | rs2             |
 | 0          | 0          | 0          | 0               | 0               |
 ------------------------------------------------------------------------------
 | WB2   | M2    | EX2   | ALU_Control | ALU_result       | Data_to_write    | rd3   |
 | 0     | 0     | 1     | 0010       | 0                | 0                 | 0     |
 ------------------------------------------------------------------------------
 | WB3   | M3    | Data_from_memory  | Data_from_ALU     | rd4    | WB4   |
 | 0     | 0     | 0                 | 0                  | 30     | 0     |
 | PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
 | 1        | 1          | 0     | 0     | 68     | 0      | 0          | 0      |
 | 0     |
 ------------------------------------------------------------------------------
 Registers[10]:
               10
 ------------------------------------------------------------------------------
```

OUTPUT :



```
 ------------------------------------------------------------------------
 Time: 75000 | PC: 76 | Instruction: 00000000
 ------------------------------------------------------------------------
 | IFID_rs1   | IFID_rs2   | IFID_rd    | rs1             | rs2             |
 | 0          | 0          | 0          | 0               | 0               |
 ------------------------------------------------------------------------
 | WB2   | M2    | EX2   | ALU_Control | ALU_result       | Data_to_write    | rd3   |
 | 0     | 0     | 1     | 0010       | 0                | 0                 | 0     |
 ------------------------------------------------------------------------
 | WB3   | M3    | Data_from_memory  | Data_from_ALU     | rd4    | WB4   |
 | 0     | 0     | 0                 | -5                 | 10     | 1     |
 | PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
 | 1        | 1          | 0     | 0     | 80     | 0      | 0          | 68     |
 | 0     |
 ------------------------------------------------------------------------
 Registers[10]:
              10
 ------------------------------------------------------------------------


 ------------------------------------------------------------------------
 Time: 85000 | PC: 80 | Instruction: 0000007f
 ------------------------------------------------------------------------
 | IFID_rs1   | IFID_rs2   | IFID_rd    | rs1             | rs2             |
 | 0          | 0          | 0          | 0               | 0               |
 ------------------------------------------------------------------------
 | WB2   | M2    | EX2   | ALU_Control | ALU_result       | Data_to_write    | rd3   |
 | 0     | 0     | 1     | 0010       | 0                | 0                 | 0     |
 ------------------------------------------------------------------------
 | WB3   | M3    | Data_from_memory  | Data_from_ALU     | rd4    | WB4   |
 | 0     | 0     | 0                 | 0                  | 0      | 0     |
 | PCWrite | IF_ID_Write | Stall | Flush | new_pc | target | immed64_2 | PC3 |
 | 1        | 1          | 0     | 0     | 84     | 0      | 0          | 72     |
 | 0     |
 ------------------------------------------------------------------------
 Registers[10]:
              -5
 ------------------------------------------------------------------------
```

GTKWAVE :



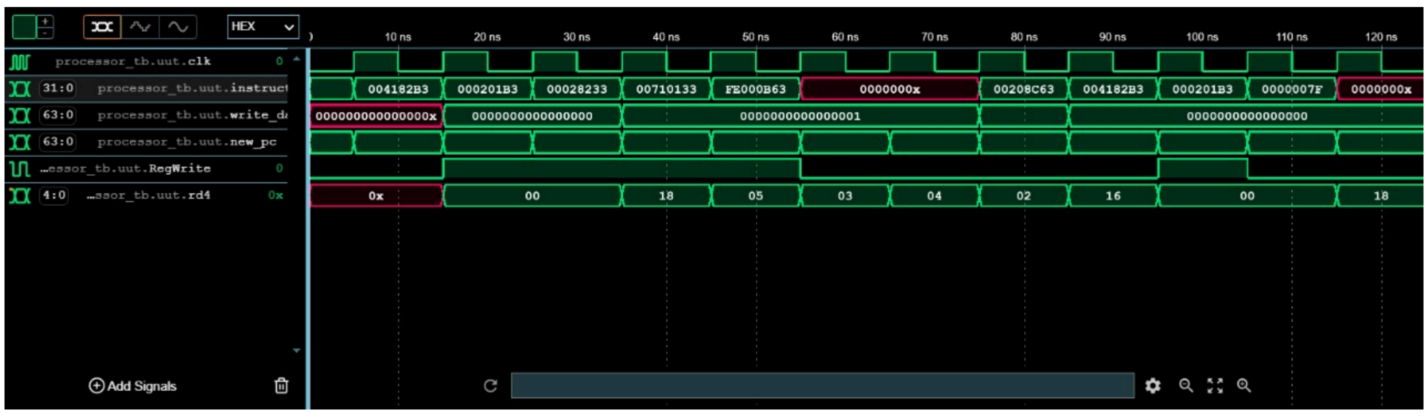
Overall Circuit :

example :

Fibonacci

input Registers :

```
registers[0]  = 64'd0;
registers[1]  = 64'd5;//n=5
registers[2]  = 64'd2; //i=2
registers[3]  = 64'd0;// fib(1)
registers[4]  = 64'd1;//fib(2)
registers[5]  = 64'd0;//result
registers[6]  = 64'd0;
registers[7]  = 64'd1;
registers[8]  = 64'd0;
registers[9]  = 64'd9;
registers[10] = 64'd10;
registers[11] = 64'd0;
registers[12] = 64'd0;
registers[13] = 64'd0;
registers[14] = 64'd0;
registers[15] = 64'd0;
registers[16] = 64'd0;
registers[17] = 64'd0;
registers[18] = 64'd0;
registers[19] = 64'd0;
registers[20] = 64'd0;
registers[21] = 64'd0;
registers[22] = 64'd0;
registers[23] = 64'd0;
registers[24] = 64'd0;
registers[25] = 64'd0;
registers[26] = 64'd0;
registers[27] = 64'd0;
registers[28] = 64'd0;
registers[29] = 64'd0;
registers[30] = 64'd0;
registers[31] = 64'd0;
```



OUTPUT :

The out 3 is stored in X5 .

```
registers[0]  = 64'd0;
registers[1]  = 64'd5;//n=5
registers[2]  = 64'd5; //i=5
```

```
    registers[3]  = 64'd1;// fib(1)
    registers[4]  = 64'd2;//fib(2)
    registers[5]  = 64'd3;//result
    registers[6]  = 64'd0;
    registers[7]  = 64'd1;
    registers[8]  = 64'd0;
    registers[9]  = 64'd9;
    registers[10] = 64'd10;
    registers[11] = 64'd0;
    registers[12] = 64'd0;
    registers[13] = 64'd0;
    registers[14] = 64'd0;
    registers[15] = 64'd0;
    registers[16] = 64'd0;
    registers[17] = 64'd0;
    registers[18] = 64'd0;
    registers[19] = 64'd0;
    registers[20] = 64'd0;
    registers[21] = 64'd0;
    registers[22] = 64'd0;
    registers[23] = 64'd0;
    registers[24] = 64'd0;
    registers[25] = 64'd0;
    registers[26] = 64'd0;
    registers[27] = 64'd0;
    registers[28] = 64'd0;
    registers[29] = 64'd0;
    registers[30] = 64'd0;
    registers[31] = 64'd0;
```

contributions :

- Gangineni Yaswanth
  - Sequential
    - Reg Block
    - MUX 2:1 , New PC
    - Wrapper
  - Pipeline
    - Load Use Hazard
    - Pipeline Wrapper
- Nethavath Praveen
  - Sequential
    - Alu , Alu Ctrl Block
    - Data Memory
    - Wrapper
  - Pipeline
    - Forwarding Unit

    - Pipeline Wrapper
- Kotha Phanindra Manoj Kumar
  - Sequential
    - Instruction Fetch
    - control Unit , Immediate
    - Wrapper
  - Pipeline
    - Branch Hazard
    - Pipeline Wrapper

Equal Contribution Towards Project Report , Testcases and Debuging .