Faculty of Engineering & Technology

Department of Electrical and Computer Engineering

Computer Architecture
ENCS4370

Project #2

Design and Verification of a Simple Pipelined RISC Processor in Verilog

---

Prepared By:

Adnan Odeh ID: 1220175

Alaa Faraj ID: 1220808

Instructor: Prof. Ayman Hroub

Section: 1

Date: Jan 2026

Guidelines for the Course Project Report

Team Member Name	Team Member ID	Contributions
Adnan Odeh	1220175	*Block diagram of Processor's Pipelined Datapath *Report Contribution *Testing Contribution *Writing Main module Verilog code implementation
Alaa Faraj	1220808	*Block diagram of Processor's Pipelined Datapath *Report Contribution *Testing Contribution *Writing Main module Verilog code implementation

Processor Implementation (Tick One)		
Single Cycle	Yes	No
Multi Cycle	Yes	No
Pipelined	✓	

In case of pipeline implementation (Tick the correct answer)	Implemented and Verified and Correctly Worked?
Data hazards detection	✓
Control hazards detection	✓
Structural hazards detection	✓
Forwarding	✓
Stalling	✓

Tick the correct answer
My processor can execute test programing of one instruction only
My processor can execute complete programs (A simulation screenshot must be provided as evidence)
✓

Instruction	Did you implement this instruction in the RTL?	Did you write the verification code for this instruction?	Did the instruction Work perfectly when it has been tested?
OR Rd, Rs, Rt, Rp	✓	✓	✓
ADD Rd, Rs, Rt, Rp	✓	✓	✓
SUB Rd, Rs, Rt, Rp	✓	✓	✓
NOR Rd, Rs, Rt, Rp	✓	✓	✓
AND Rd, Rs, Rt, Rp	✓	✓	✓
ADDI Rd, Rs, Imm, Rp	✓	✓	✓
ORI Rd, Rs, Imm, Rp	✓	✓	✓
NORI Rd, Rs, Imm, Rp	✓	✓	✓
ANDI Rd, Rs, Imm, Rp	✓	✓	✓

Team Member Name	Team Member ID	Contributions		
LW Rd, Imm(Rs), Rp	✓	✓	✓	
SW Rd, Imm(Rs), Rp	✓	✓	✓	
J Label, Rp	✓	✓	✓	
CALL Label, Rp	✓	✓	✓	
JR Rs, Rp	✓	✓	✓	
Tick one of the following				
My processor can execute test programing of one instruction only				
My processor can execute complete programs (A simulation screenshot must be provided as evidence)				
✓				

Table 1: Report Table

Abstract:

This project involves designing and implementing a 32-bit pipelined RISC processor with predicated execution using Verilog. The processor follows a five-stage pipeline architecture, including fetch, decode, execute, memory access, and write-back stages, with integrated hazard detection and resolution mechanisms. It supports a predicated instruction set encompassing R-Type, I-Type, and J-Type formats for arithmetic, logical, memory, and control flow operations. The system features 32 general-purpose registers, with R0 hardwired to zero, R30 serving as the program counter, and R31 as the return address register. Predication is implemented through a predicate register (Rp) that conditionally enables instruction execution. The design incorporates data forwarding, pipeline stalling, and comprehensive hazard detection for data, control, and structural hazards. Control signals were systematically derived using truth tables, state diagrams, and Boolean equations. Functional verification was achieved through simulation-based testing using multiple assembly programs that demonstrate correct execution of all supported instructions and pipeline features.

Table of Contents

Table Of Figure

List Of Table

1. Datapath description, components, and implementation details:

The pipelined RISC processor Datapath is organized into five classic stages: Fetch (IF), Decode (ID), Execute (EX), Memory (MEM), and Write-Back (WB). The design integrates hazard handling, forwarding, and specialized control logic to support the ISA requirements. Below is a breakdown of the components and their integration:

1.1. Fetch Stage (IF):

1.1.1. Components:

- **Program Counter (PC):** 32-bit register holding the next instruction address.

  memory is **word-addressable** (not byte-addressable), each memory address points to one complete 32-bit word (one instruction). This means:

  PC increment = 1 (to move to the next instruction/word)

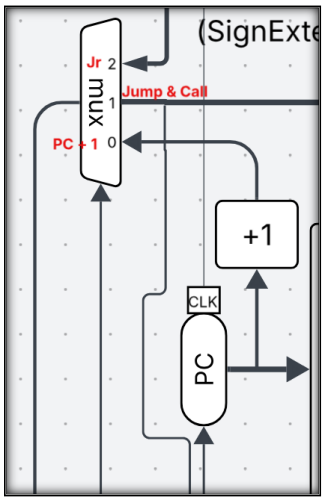


Figure 1-1: Program Counter module

- **Instruction Memory:** 256-word ROM storing instructions (word-addressable).

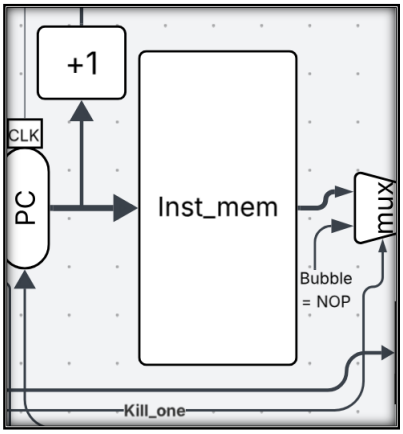


Figure 1-2: Instruction Memory module

- **PC Adder** - Calculates PC + 1 for sequential execution
- **PC Multiplexer (4:1)** - Selects next PC source:

  00: Sequential (PC + 1)

  01: Call/Jump target, Jump Target = PC + (SignExtend(Offset22) << 2)

  01: JR, Jump to the address in Register Rs
- **PC Control Unit** - Generates enable/select signals based on predicate evaluation
- **2x1 Mux**, that has an input of:

  00: Instruction Memory: In case no Control Hazard

  01: No Operation: In case Control Hazard Occurs with Jump Instructions. Which perform one stall cycle in the Fetch Stage.

- **This mux controlled by the PC Control module which test the existence for the Jump instruction and produce a control value for the mux, which will handle the control hazard by adding a stall cycle for the jump instruction in the first stage.**

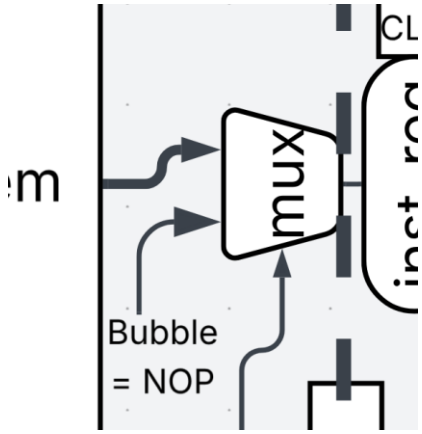


Figure 1-3: 2x1Mux Control Hazard Handling

PC Control Module: Works as an input for the PC Mux, it has two inputs and two outputs

- Input 1: **Opcode**, which defines the operations that will be done, and on what data sources will these operations be done.
- Input 2: **Predicate Register**, which is define whether the Instruction will execute or not.
    - 0'1b: Instruction will not execute.
    - 1'1b: Instruction will execute.
- Output: Is considered as an input to the mux that will find the next PC value to be executed on the CPU.

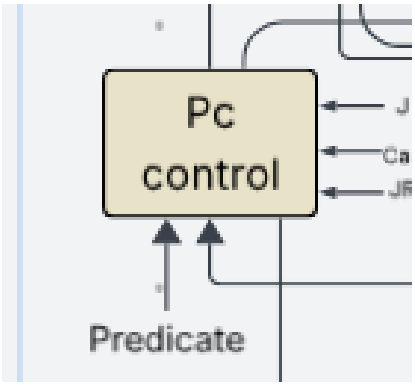


Figure 1-4: PC Control Unit

1.1.2. Assembly:

The stage Fetches the Instructions using PC value, and the PC Mux controls the flow for the Jump, CALL, JR Instruction. And PC Control Unit controls the Selection mux to manage Instructions and stalls for a proper input the next stage.

1.2. Decode Stage (ID):

1.2.1. Components:

Register File: 32 Registers each is 32 Bit.

R31: hardwired, Saves the address of the **current PC** before jumping into the address specified in the **CALL** Instruction.

R30: hardwired, Stores the value of the **PC**.

R0: Hardwired to **zero**.

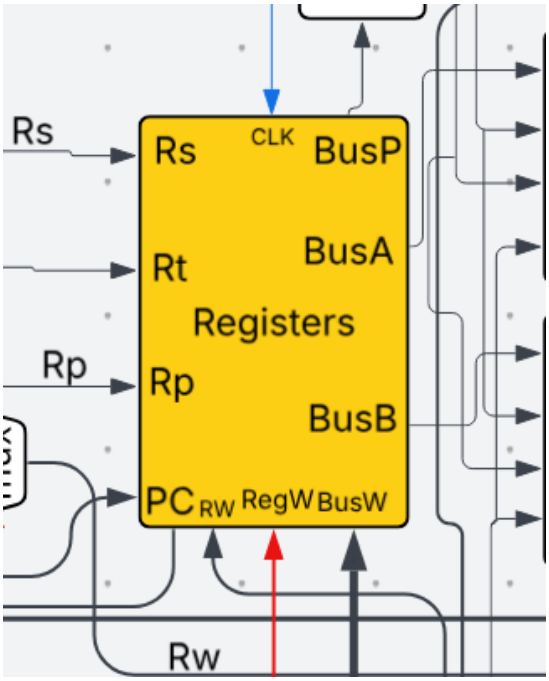


Figure 1-5: Register File

Predicate Register: Value that decides whether the instruction will execute or not. This can be done by finding whether the value of the Register Rp is zero or not. **Rp can be any general-purpose register**. If the content of Rp is zero, the instruction is not executed. However, if the content of Rp is non-zero, the instruction executes. To execute an instruction unconditionally, the programmer must set Rp to R0.

Practically: This done by ORing all bits in the value of register Rp, then give flag Predicate for this register.

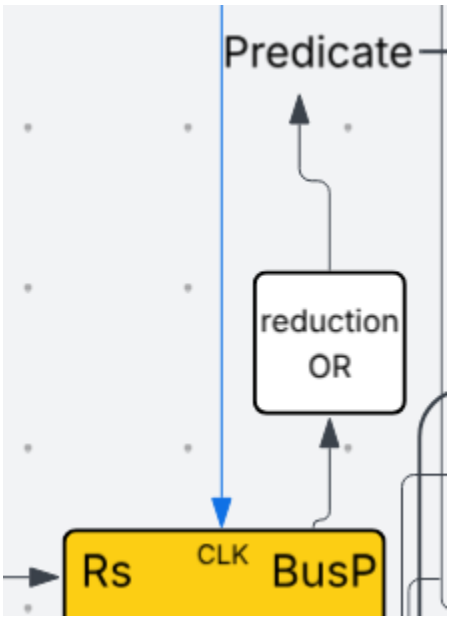


Figure 1-6: Predicate Register

Main and ALU Control: This Main control unit that take a decision for the input muxes to what values that will be taken into Register File. And it is a main input for a mux that will decide to give a stall cycle or not.

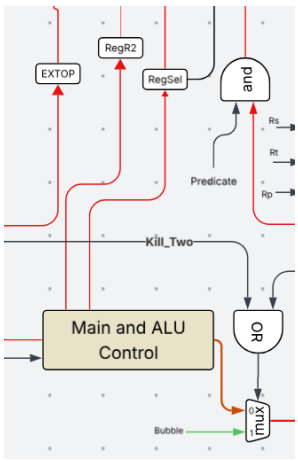


Figure 1-7: Main and ALU Control

Other Important Components:

Registers to move Pipelined Signals from stage to another:

- From Fetch to Decode, Registers are Instruction Code, PC, PC + 1.
- Then Splitting the Instruction code into Rs, Rp, Rd, Rt, Imm12.
- We used a mux with two inputs Rt, Rd and Selection line comes from the main ALU Control Unit with name RegR2, and output to the register file.
- Another mux with two inputs Rd, R31 and Selection line RegSel and output RW goes to the next stage.
- An Extender for the Immediate comes, and can be define according to the operation will pass through ALU, Sign Extend, Unsigned Extend.
- EXTOP, RegR2, RegSel comes from Main and ALU Control Unit.
- For RegW This comes from the Feedback in the last stage and addOpration with Predicate value.

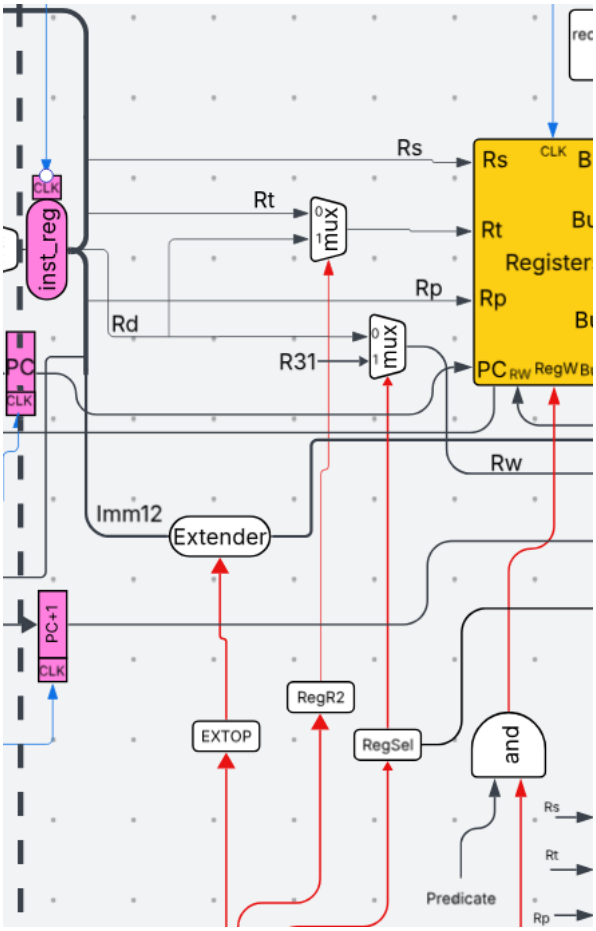


Figure 1-8: Components in Decode Stage

Hazard Handling:

Forwarding Unit → Resolves RAW hazards by selecting data from EX/MEM/WB stages via 2:1 muxes for Rs, Rt, Rd.

Stall Logic → Input to an **OR** Operation that That will define whether there is a stall cycle or not.

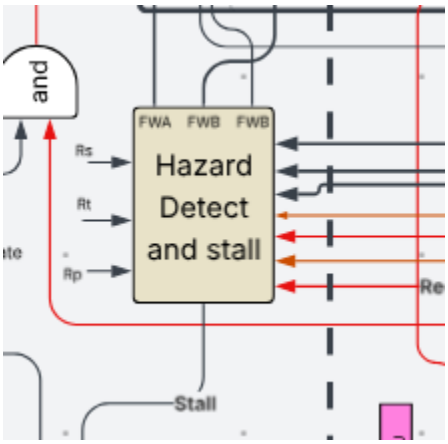


Figure 1-9: Hazard and Forwarding Unit

Forwarding Operation is down using these muxes with selection line muxes comes from Hazard Detection Unit.

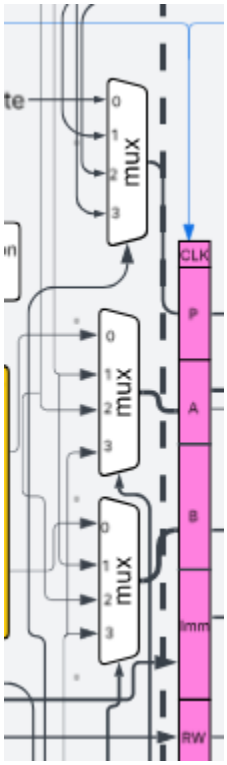


Figure 1-10: Forwarding Muxes

1.2.2. Assembly:

Instruction Parsing: 32-bit instruction split into fields opcode, Rp, Rd, Rs, Rt, immediate, offset.

Control Signals: Generated by Main_Control and ALU_control, suppressed during stalls. **Register File Access**: Three registers read (BusA, BusB, BusP), R0 = 0, PC held during stalls. Predicate Evaluation: Predicate = 1 if Rp = 0, else based on BusP, hazards resolved via forwarding (FWPMux).

PC Target Calculation: Jump target = PC + sign-extended offset, jump register target = BusA. **Immediate Processing**: Immediate either sign-extended (arithmetic/mem ops) or zero-extended (logical ops) via ExtOp.

Forwarding Resolution: Three 4-to-1 muxes (FWAMux, FWBMux, FWPMux) select between register outputs or forwarded EX/MEM/WB values.

Destination Register Selection: Two muxes choose between Rt/Rd and optionally R31 for CALL.

Pipeline Register Storage: Decoded values, operands, immediate, control signals, predicate, and PC+1 stored in D/EX, flushed with zeros for control signals while preserving data paths.

1.3. Execute Stage (EX):

1.3.1. Components:

ALU: Supports 000: ADD | 001: SUB | 010: OR | 011: NOR | 100: AND

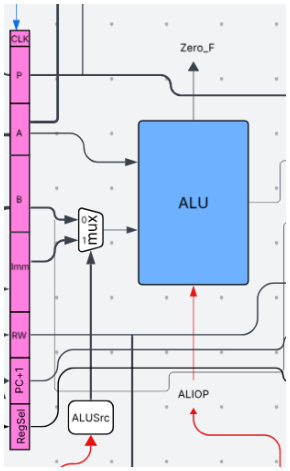


Figure 1-11: ALU Unit

- ALU Source Mux Chooses between Imm or BusB and this value selected from the ALUSrc.

1.3.2. Assembly:

ALU Source Selection: 2-to-1 mux chooses between immediate (Imm_E) or register (BUSB_E) for ALU operand based on ALUSrc.

ALU Operation: ALU performs operation (ADD, SUB, OR, NOR, AND) per ALUOP_E, zero flag set if result = 0.

Forwarding Detection: Hazard detection compares Decode stage sources (Rs, Rt, Rp) with EX/MEM/WB destinations, generates forwarding signals (FWA, FWB, FWP).

Load-Use Stall Detection: Stall inserted if EX-stage load writes to a register used by Decode-stage sources.

Pipeline Register Storage: ALU result, BUSB_E, destination register, predicate, control signals, and PC+1 stored in E/MEM register.

Hazard Handling: During stalls, D/EX control signals (RegWrite, MEMRd, MEMWr) set to zero, creating NOP without corrupting data.

1.4. Memory Stage (MEM):

1.4.1. Components:

Data Memory: 256-word RAM (word-addressable).

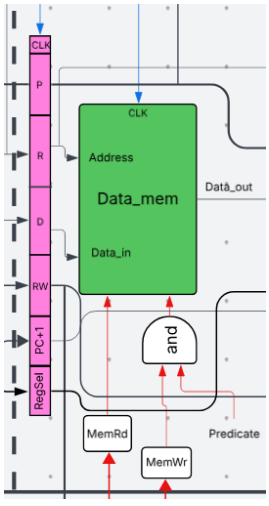


Figure 1-12: Memory Module

Memory Mux (4:1): Selects write_back_data from ALU result or memory read.

00: The address of the Memory

01: Data_out From Memory

10: Value of PC for the CALL Function

1.4.2. Assembly:

Memory Addressing: ALU result (Res_mem) is used as the memory address, only lower 5 bits are used (32-word memory).

Memory Write: Store occurs on clock edge when MEMWr_mem is asserted and Predicate_mem = 1. Memory Read: Load reads data asynchronously from memory when MEMRd_mem is asserted. Predicate-Controlled Stores: Predicate gates memory writes to support conditional execution.

Write-Back Data Selection: 4-to-1 mux selects ALU result, loaded data, PC+1, or zero based on WB_data_mem.

Pipeline Register Storage: BusW, destination register, predicate, and RegWrite stored in MEM/WB register for Write-Back stage.

1.5. Pipeline Registers:

1.5.1. Fetch to Decode Registers:

Buffer for the Instruction, Program Counter, and the Incremented Program counter which moved to final stage.

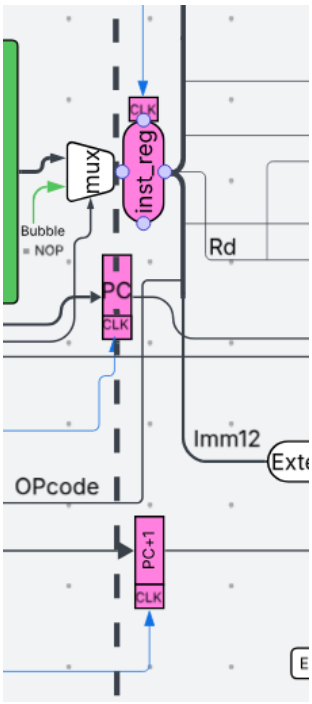


Figure 1-13: IF-ID Buffers

1.5.2. Decode to Execute Registers:

Buffers to move the values of BUSA, BUSB, Immediate Value, Register Write output from the mux, and Incremented Program Counter, Predicate Value, and the Register Selection for the mux used in memory stage.

Some Values are used in the Execution stage and other values used in the next stages.

Also, Some Register Flags from Main and ALU Control Unit moved to the next stages.

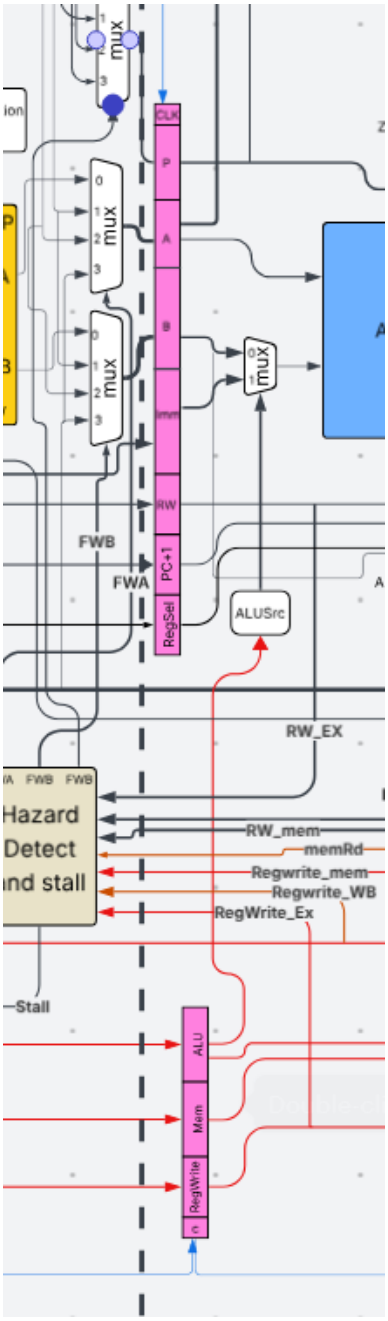


Figure 1-14: Buffer Register from ID – IE

1.5.3. Execute to Memory Buffers:

Buffers move the output of ALU, Predicate Value, Data in Value, Register Write Register, Incremented Program Counter.

Register Mux Selection, and other flags as an output of Main and ALU Control Unit.

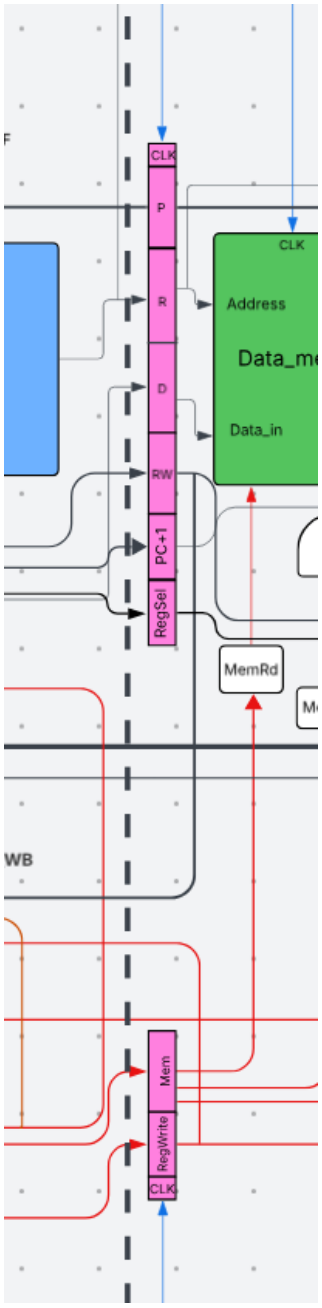


Figure 1-15: Buffers IE – IM

1.5.4. Memory to Write Back Buffers:

All of the Values in the Buffers included in the Figure Below are for Write Back Registers, which define the register that will be written and the flag to write or not in the register.

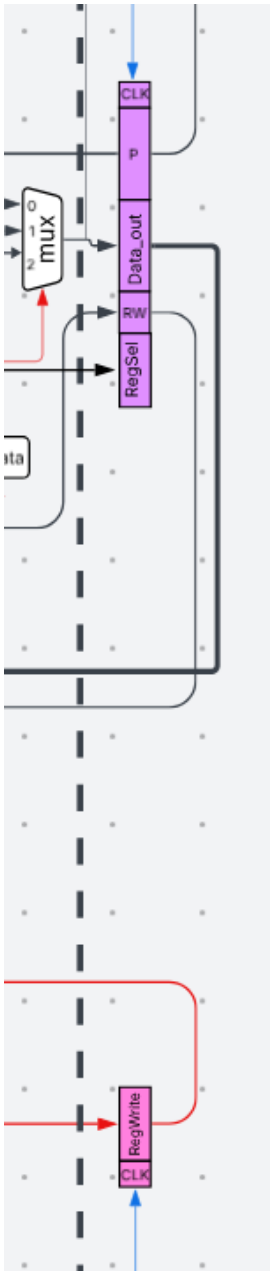


Figure 1-16: Memory to Write Back Registers

1.6. Implementation Details and Design Choices:

1.6.1. Hazard Resolution:

Forwarding: Implemented three parallel 4-to-1 forwarding multiplexers in the Decode stage for operands Rs, Rt, and predicate register Rp. Forwarding sources include:

- **00: Register file output (no hazard)**
- **01: EX stage result (EX-EX forwarding from RES)**
- **10: MEM stage result (MEM-EX forwarding from BusW)**
- **11: WB stage result (WB-EX forwarding from Data_WB)**

This three-level forwarding hierarchy eliminates most Read-After-Write (RAW) hazards without stalling. By forwarding from EX/MEM/WB stages back to the Decode stage, back-to-back dependent instructions can execute without bubbles. For example:

```
ADD R3, R1, R2, R3 written in cycle N

SUB R4, R3, R1, R3 forwarded from EX stage in cycle N+1
```

The forwarding unit (Hazard_Detect_And_Stall) compares destination registers (RW_EX, RW_mem, RW_WB) against source registers (Rs, Rt, Rp) and asserts the appropriate forwarding select signals (FWA, FWB, FWP).

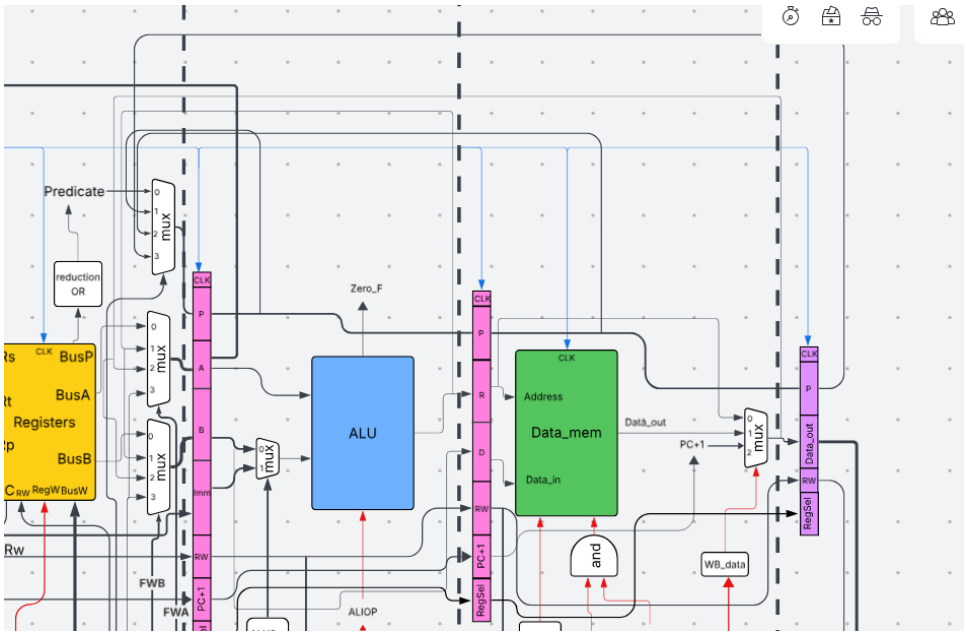


Figure 1-17: Forwarding Logic

Stalling: Choice: The hazard detection unit asserts a Stall signal when:

- **A load instruction (LW) is in the Execute stage (MEMRd_E == 1)**
- **AND its destination register matches any source register in Decode stage**
- **Condition: (Rs == RW_E || Rt == RW_E || Rp == RW_E) && (register != R0)**

When **Stall** is asserted:

PC is frozen: The PC register does not update (stall signal prevents PC write in REG_file)

F/D register holds: Fetch-to-Decode pipeline register does not update (due to stall input)

D/E register converts to NOP: The Decode-to-Execute register clears control signals (RegWrite_E, MEMRd_E, MEMWr_E set to 0) while preserving data paths, effectively injecting a pipeline bubble

Load instructions require two cycles to produce data (Execute computes address, Memory fetches data). Forwarding alone cannot resolve this timing gap, the data doesn't exist until the Memory stage completes. A mandatory 1-cycle stall ensures the dependent instruction waits until the loaded value is available for forwarding from the MEM stage.

Example:

LW R5, 0(R0), Cycle 1: EX, Cycle 2: MEM (data ready)

ADD R6, R5, R1, Stalls in Decode during Cycle 2, executes in Cycle 3 with forwarding

1.6.2. Centralized Control in Decode Stage

Choice: All control signals (ALUOP, RegWrite, ALUSrc, MemRd, MemWr, RegSel, WB) are generated in the Decode stage by the Main_Control and ALU_control modules. These signals propagate through pipeline registers (D_TO_E_reg, E_TO_MEM_reg, MEM_TO_WB_reg) alongside data.

Justification: This design separates control logic from execution logic, simplifying timing analysis. Each pipeline stage operates on locally buffered control signals from its input register, eliminating long combinational paths across stages. The modular approach also makes debugging easier, control signal errors are isolated to the Decode stage.

1.6.3. Stall-Aware Control Signal Suppression

Choice: When Stall is asserted, the Main_Control and ALU_control modules suppress write-enable signals:

```
assign RegWrite = (ADD | SUB | OR | ...) & ~Stall;

assign MemWr = SW & ~Stall;
```

Justification: During a stall, the instruction in Decode must not modify architectural state. By ANDing control signals with ~Stall, the pipeline automatically converts stalled instructions into NOPs without requiring explicit flush logic. This prevents erroneous writes during load-use hazards.

1.6.4. Pipeline Flushing for Control Hazards:

Choice: A flush signal is generated when control flow changes (Call, Jumps):

assign flush = (PC_control! = 2'b00);

```

When `flush` is asserted:

- **F/D register: ** Instruction is replaced with NOP (all zeros)

- **D/E register: ** All control signals (`RegWrite_E`, `MEMRd_E`, `MEMWr_E`) are cleared

**Justification: ** Call and Jumps redirect the PC, invalidating instructions already fetched into the pipeline. Flushing converts these "wrongly fetched" instructions into NOPs, preventing them from modifying state. For example:

```

J Label; Jump taken in Decode stage

ADD R1, R2, R3; Already fetched but must be flushed

Your design resolves branches in the Decode stage (using forwarded register values), resulting in a **1-cycle branch penalty**, only the instruction immediately after the branch is flushed.

1.6.5. Predicated Execution Handling:

Predicate Evaluation in Decode Stage

Choice: The predicate signal is computed combinationally:

```
assign Predicate = (Rp == 5'd0)? 1'b1: |BusP_outputmux_Forwarding;
```

- If Rp == R0, predicate is always true (unconditional execution)
- Otherwise, predicate is true if forwarded BusP is non-zero

Justification: Early predicate evaluation (in Decode rather than execute) allows the pipeline to suppress memory writes sooner. The predicate propagates through all pipeline stages and gates critical operations:

- **Memory Write:** wr = MEMWr_mem & Predicate_mem
- **Register Write:** if (RegW && (Rp == 5'd0 || RegPRes))

This ensures instructions with false predicates "execute" through the pipeline without side effects, maintaining pipeline flow without flushing.

Predicate Propagation Through Pipeline

Choice: The Predicate signal is stored in every pipeline register (Predicate_E, Predicate_mem, Predicate_WB) and used locally at each stage.

Justification: Different stages need predicate information at different times:

- **Memory stage:** Gates MemWr to prevent stores
- **Write-Back stage:** Gates RegWrite via RegPRes signal

By propagating the predicate through the pipeline, each stage makes independent decisions without backtracking to earlier stages.

1.6.6. Pipeline Register Design:

Stall-Aware Pipeline Registers

Choice: All pipeline registers include stall and flush inputs:

- **F_TO_D_reg:** Holds current instruction when stall=1, replaces with NOP when flush=1
- **D_TO_E_reg:** Clears control signals when stall=1 or flush=1, preserves data paths
- **E_TO_MEM_reg, MEM_TO_WB_reg:** Standard clocked registers (no stall/flush needed downstream)

Justification: This selective stalling approach minimizes hardware complexity:

- **Early stages (F/D, D/E):** Need stall/flush logic because hazards are detected in Decode
- **Late stages (E/M, M/WB):** Don't need stall logic because once an instruction reaches Execute, it will complete (hazards have been resolved)

The D/E register's behavior during stalls is critical:

```
if (stall) begin
    RegWrite_E <= 1'b0;
    MEMRd_E <= 1'b0;
    MEMWr_E <= 1'b0;
    // Data paths preserved but writes disabled
end
```

This converts the stalled instruction into a “NOP”, it passes through Execute, Memory, Write Back, without modifying state, effectively inserting a bubble without corrupting the pipeline.

PC Update Control

Choice: The PC register in the register file (`REG_file`) only updates when `!stall`:

```
else if (!stall) begin
    PC <= input_mux_pc;
    Register[30] <= input_mux_pc;
end
```

Justification: Freezing the PC during stalls ensures the same instruction is re-fetched in the next cycle, allowing the stalled instruction in Decode to advance once the hazard is resolved. This is simpler than buffering fetched instructions in a separate queue.

1.6.7. Register File Design Choices

Hardwired Zero Register (R0):

Choice: Register R0 is always forced to zero after every clock cycle:

```
Register[0] <= 32'b0;
```

Justification: This RISC convention simplifies many operations (e.g., loading immediates, no-op source operands) and is enforced in hardware to prevent programming errors. Even if software attempts to write to R0, the register file ignores it.

PC as Architectural Register (R30):

Choice: The PC is stored in register R30 and updated alongside the internal PC register:

```
Register[30] <= input_mux_pc;
```

Justification: Exposing the PC as a readable register enables PC-relative addressing and self-modifying code patterns. Writing to R30 is prevented by the write logic (Rd != 5'd30), ensuring the PC is only updated via control flow instructions.

Predicated Write Logic:

Choice: Register writes are gated by both RegWrite and the predicate result:

```
if (RegW && (Rp == 5'd0 || RegPRes) && (Rd != 5'd0) && (Rd != 5'd30))
    Register[Rd] <= BusW;
```

Justification: This implements predicated execution at the architectural level, instructions with false predicates complete through the pipeline but don't update registers. The check (Rp == 5'd0 || RegPRes) ensures unconditional instructions (where Rp=R0) always write, while conditional instructions only write if their predicate was true.

1.7. Control Signals:

1.7.1. Control Signals Truth Tables:

Table 2: Main and ALU Control Unit

Instruction	*Op*	*ALUOp*	*RegR2*	*ExtOp*	*RegWr*	*ALUSrc*	*MemRd*	*MemWr*	*WB*	*RegSel*
add	0	000	0	X	1	0	0	0	0	0
sub	1	001	0	X	1	0	0	0	0	0
or	2	010	0	X	1	0	0	0	0	0
nor	3	011	0	X	1	0	0	0	0	0
and	4	100	0	X	1	0	0	0	0	0
addi	5	000	X	1	1	1	0	0	0	0
ori	6	010	X	0	1	1	0	0	0	0
nori	7	011	X	0	1	1	0	0	0	0
andi	8	100	X	0	1	1	0	0	0	0
lw	9	000	X	1	1	1	1	0	1	0
sw	10	000	1	1	0	1	0	1	X	0
j	11	XXX	X	X	0	X	0	0	X	0
call	12	XXX	X	X	1	X	0	0	2	1
jr	13	XXX	X	X	0	X	0	0	X	0

Table 3: Signals Explanation

SIGNAL	BOOLEAN EXPRESSION	ACTIVE WHEN
REGR2	SW	Store Word instruction
EXTOP	ADDI + LW + SW	Sign-extension needed
REGWRITE	$S\overline{W} \cdot J\overline{R} \cdot \overline{J}$	All except SW, JR, J
ALUSRC	ADDI + ORI + NORI + ANDI + LW + SW	I-type instructions
MEMRD	LW	Load Word instruction
MEMWR	SW	Store Word instruction
REGSEL	CALL	Call instruction (write to R31)
ALUOP[2:0]	See truth table below	Varies by instruction

ALUOP Truth Table

Table 4: ALUOP Signals Logic

Instruction(s)	ALUOP[2:0]	Operation
ADD, ADDI, LW, SW	000	A + B
SUB	001	A - B
OR, ORI	010	A | B
NOR, NORI	011	~(A | B)
AND, ANDI	100	A & B

Complete Verilog Code

```
// From ALU_control.v
module ALU_control(
    input  [4:0] Opcode,
    input Stall,
    output reg [2:0] ALUOP
);
    wire ADD   = (Opcode == 5'd0);
    wire SUB   = (Opcode == 5'd1);
    wire OR    = (Opcode == 5'd2);
    wire NOR   = (Opcode == 5'd3);
    wire AND   = (Opcode == 5'd4);
    wire ADDI  = (Opcode == 5'd5);
    wire ORI   = (Opcode == 5'd6);
    wire NORI  = (Opcode == 5'd7);
    wire ANDI  = (Opcode == 5'd8);
    wire LW    = (Opcode == 5'd9);
    wire SW    = (Opcode == 5'd10);

    always @(*) begin
        if ((ADD | ADDI | LW | SW) && ~Stall)
            ALUOP = 3'b000;  // ADD operation
        else if (SUB && ~Stall)
            ALUOP = 3'b001;  // SUB operation
        else if ((OR | ORI) && ~Stall)
            ALUOP = 3'b010;  // OR operation
        else if ((NOR | NORI) && ~Stall)
            ALUOP = 3'b011;  // NOR operation
        else if ((AND | ANDI) && ~Stall)
            ALUOP = 3'b100;  // AND operation
        else
            ALUOP = 3'b000;  // Default: ADD
    end
endmodule
```

PC Source Control (PCSrc):

Table 5: PCSrc

Type	Instruction	OP	PCsrc
R-type	add	0	0
R-type	sub	1	0
R-type	or	2	0
R-type	nor	3	0
R-type	and	4	0
I-type	addi	5	0
I-type	ori	6	0
I-type	nori	7	0
I-type	andi	8	0
I-type	lw	9	0
I-type	sw	10	0
J-type	j	11	1 → OP=11 and predicate=1 else 0
J-type	call	12	1 → OP=12 and predicate=1 else 0
R-type	jr	13	2 → OP=13 and predicate=1 else 0

1.7.2. Control Signals Explanation and Choices:

RegR2 (Register Read 2 Selector)

Logic: `RegR2 = SW & ~Stall`

Purpose: Redirects the second register read port from Rt to Rd for Store Word instructions.

Justification: SW instructions use format `SW Rd, Imm(Rs)`, where Rd contains the data to be stored. RegR2=1 allows the register file to output Rd on BusB (normally used for Rt), making store data available for the Memory stage.

ExtOp (Extension Operation Selector)

Logic: ExtOp = (ADDI | LW | SW) & ~Stall

Purpose: Controls immediate field extension type:

- **ExtOp = 1:** Sign-extend (for ADDI, LW, SW) - preserves sign bit for two's complement arithmetic
- **ExtOp = 0:** Zero-extend (for ORI, ANDI, NORI) - preserves bit patterns for logical operations

RegWrite (Register Write Enable)

Logic: RegWrite = (ADD | SUB | OR | NOR | AND | ADDI | ORI | NORI | ANDI | LW | CALL) & ~Stall

Purpose: Enables writing to the register file in the Write-Back stage.

Justification: All instructions except SW, J, and JR produce results that must be stored. The ~Stall term prevents register writes during pipeline stalls, automatically converting stalled instructions to NOPs.

ALUSrc (ALU Source Selector)

Logic: ALUSrc = (ADDI | ORI | NORI | ANDI | LW | SW) & ~Stall

Purpose: Selects the second ALU operand:

- **ALUSrc = 0:** Register value (BusB) for R-type instructions
- **ALUSrc = 1:** Extended immediate for I-type instructions and memory addressing

MemRd (Memory Read Enable)

Logic: MemRd = LW & ~Stall

Purpose: Enables reading from data memory in the Memory stage. Only asserted for Load Word instructions.

MemWr (Memory Write Enable)

Logic: `MemWr = SW & ~Stall`

Purpose: Enables writing to data memory. Further gated by predicate signal: `wr = MEMWr_mem & Predicate_mem` to support conditional stores.

RegSel (Register R31 Selector)

Logic: `RegSel = CALL & ~Stall`

Purpose: Overrides destination registers to R31 for CALL instructions, establishing a calling convention where R31 always contains the return address.

Note: **Memory addressing operations (LW/SW) use ADD to compute effective address as base + offset.**

Stall Integration

All control signals are ANDed with `~Stall` to automatically convert stalled instructions into NOPs during load-use hazards. This approach eliminates the need for explicit NOP insertion:

- During stalls, `RegWrite`, `MemWr`, and `MemRd` are forced to 0
- Data paths remain intact, but write operations are disabled
- When the stall is released, the instruction is re-decoded with full control signals

2. Datapath:

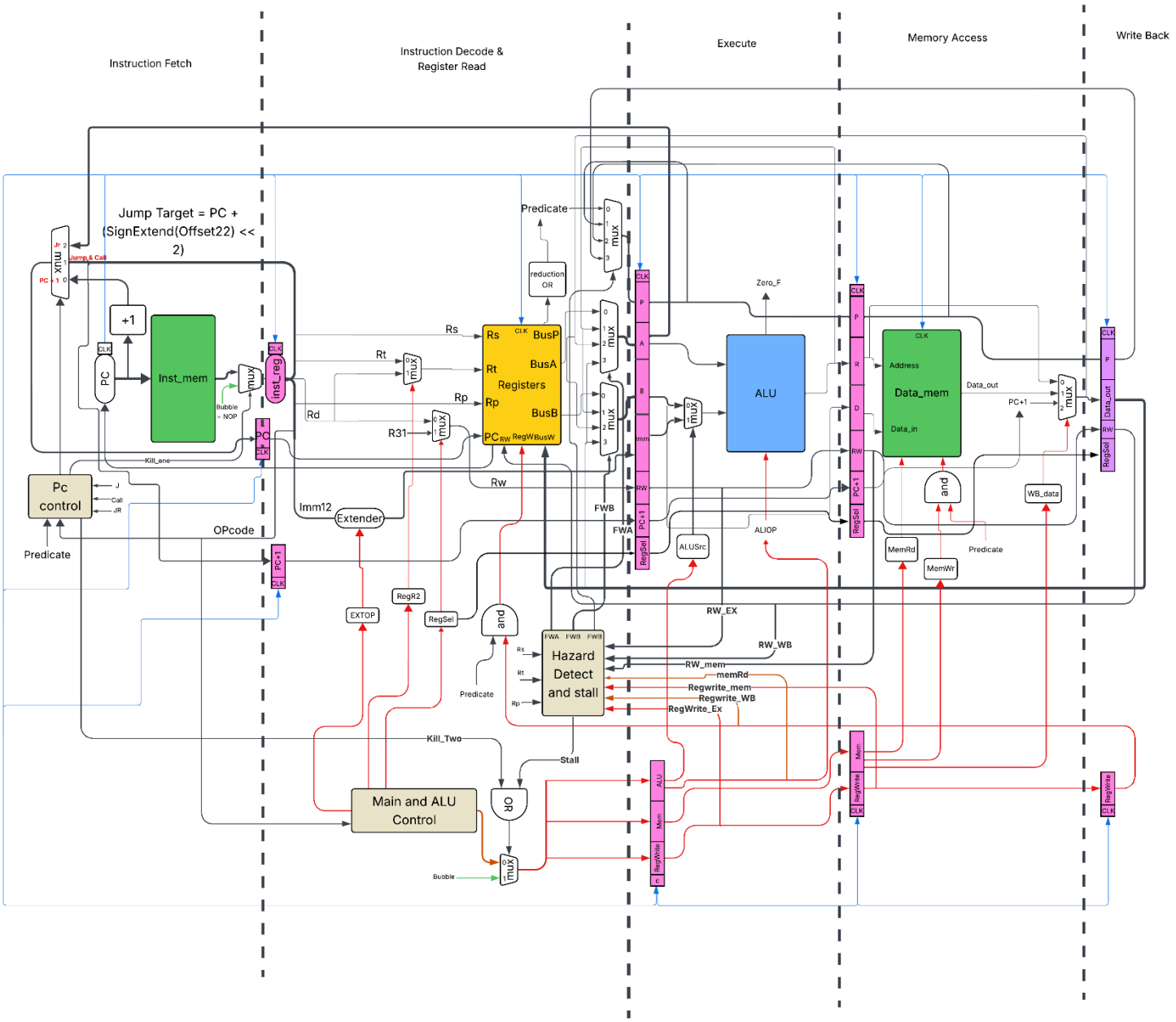


Figure 1-1: Datapath.

3. Testing

Here is the program where we tested all the instruction and the Hazards in the Pipelined processor.

I1) PC=0: ADD R7,R1,R2,R0 ; R7 = 100 + 150 = 250

I2) PC=1: SUB R8,R7,R1,R0 ; R8 = 250 - 100 = 150 [RAW: R7 from EX]

I3) PC=2: OR R9,R8,R2,R0 ; R9 = 150 | 150 = 150 [RAW: R8 from MEM]

I4) PC=3: AND R10,R9,R7,R0 ; R10 = 150 & 250 = 146 [RAW: R9 from WB]

I5) PC=4: ADDI R11,R1,50,R0; R11 = 100 + 50 = 150

I6) PC=5: ORI R12,R3,5,R0; R12 = 10 | 5 = 15

I7) PC=6: ANDI R13,R5,240,R0; R13 = 255 & 240 = 240

I8) PC=7: NORI R14,R3,0,R0; R14 = ~(10 | 0)

I9) PC=8: SW R1,10(R0),R0 ; MEM[10] = 100

I10) PC=9: LW R15,10(R0),R0 ; R15 = MEM[10] = 100

I11) PC=10: ADD R16,R15,R2,R0 ; R16 = 100 + 150 = 250 [STALL: Load-Use]

I12) PC=11: SUB R17,R16,R1,R0 ; R17 = 250 - 100 = 150

I13) PC=12: ADD R18,R1,R2,R20 ; R18 = 250 [Predicate R20=1, EXEC]

I14) PC=13: ADD R19,R1,R2,R21 ; R19 = 0 [Predicate R21=0, NOT EXEC]

I15) PC=14: SW R2,15(R0),R21 ; No write [Predicate R21=0, NOT EXEC]

I16) PC=15: J +5,R0 ; Jump to PC=20 [CONTROL HAZARD]

PC=16-19: [SKIPPED]

I17) PC=20: ADD R22,R3,R4,R0 ; R22 = 10 + 20 = 30

I18) PC=21: J +10,R21 ; NO Jump [Predicate R21=0]

I19) PC=22: ADD R23,R2,R3,R0 ; R23 = 150 + 10 = 160 [Jump not taken]

I20) PC=23: CALL +3,R0 ; Call to PC=26, R31=24

I21) PC=24: J +4,R0 ; Jump to PC=28 [After return from CALL]

I22) PC=25: [SKIPPED]

I23) PC=26: ADD R24,R1,R3,R0 ; R24 = 100 + 10 = 110 [In function]

I24) PC=27: JR R31,R0 ; Return to PC=24 [JR with R31]

I25) PC=28: ADD R25,R1,R4,R0 ; R25 = 100 + 20 = 120 [After return]

I26) PC=29: LW R26,0(R0),R0 ; R26 = MEM[0] = 100

I27) PC=30: LW R27,8(R0),R0 ; R27 = MEM[8] = 0xAA

I28) PC=31: ADD R28,R26,R27,R0; R28 = 100 + 170 = 270 [STALL: R27]

I29) PC=32: NOR R29,R3,R4,R0 ; R29 = ~(10 | 20)

in the first four instruction we tested the basic arithmetic instruction and the raw hazards: The results are from cycle 1 to 8 these instructions will be executed. If we look at instructions I2, we will see that the R7 is a source reg and in the previous instructions is a distribution reg, the same for I3,I4.

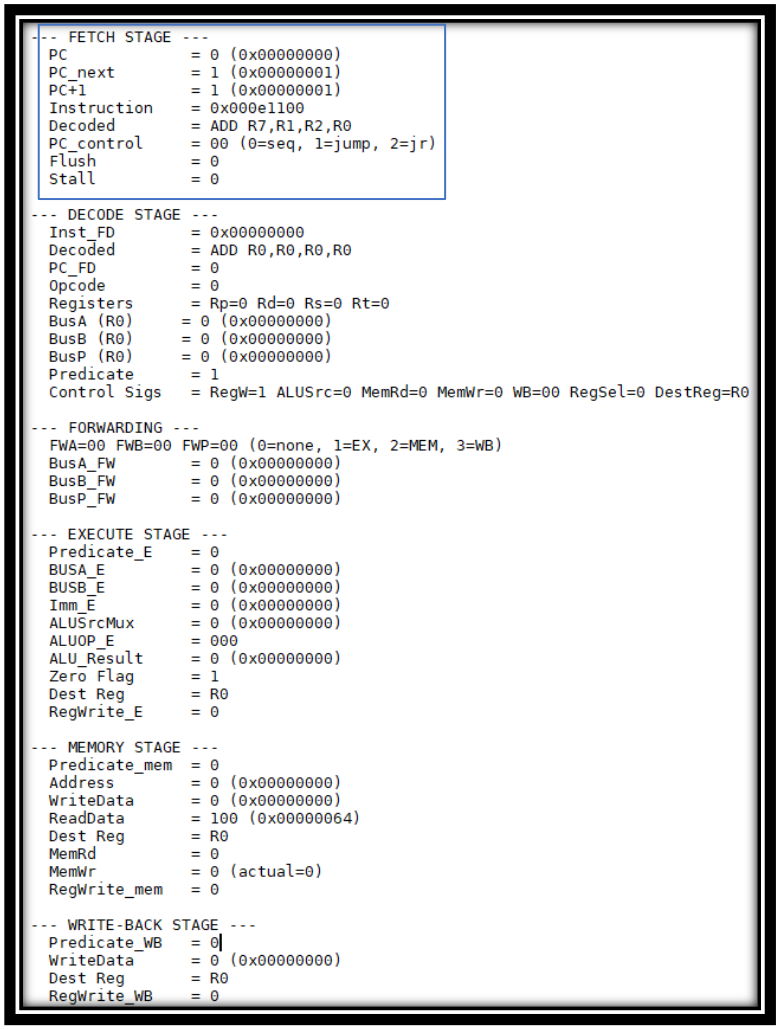


Figure 3-1:cycle #1.

In fig (2-1) we can see that I1 has been fetched, and all the other stages are empty.

```
--- FETCH STAGE ---
  PC              = 1 (0x00000001)
  PC_next         = 2 (0x00000002)
  PC+1            = 2 (0x00000002)
  Instruction     = 0x08107080
  Decoded         = SUB R8,R7,R1,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x000e1100
  Decoded         = ADD R7,R1,R2,R0
  PC_FD           = 0
  Opcode          = 0
  Registers       = Rp=0 Rd=7 Rs=1 Rt=2
  BusA (R1)      = 100 (0x00000064)
  BusB (R2)      = 150 (0x00000096)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R7

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 150 (0x00000096)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 0
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 0

--- WRITE-BACK STAGE ---
  Predicate_WB    = 0
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R30: 0 -> 1
```

Figure 3-2: cycle #2.

In the second cycle I2 was fetched and I1 moved to the decode stage as shown in the orange square.

```
--- FETCH STAGE ---
  PC              = 2 (0x00000002)
  PC_next         = 3 (0x00000003)
  PC+1            = 3 (0x00000003)
  Instruction     = 0x10128100
  Decoded         = OR R9,R8,R2,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x08107080
  Decoded         = SUB R8,R7,R1,R0
  PC_FD           = 1
  Opcode          = 1
  Registers       = Rp=0 Rd=8 Rs=7 Rt=1
  BusA (R7)      = 0 (0x00000000)
  BusB (R1)      = 100 (0x00000064)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R8

--- FORWARDING ---
  FWA=01 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 250 (0x000000fa)
  BusB_FW         = 100 (0x00000064)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 150 (0x00000096)
  Imm_E           = 256 (0x00000100)
  ALUSrcMux       = 150 (0x00000096)
  ALUOP_E         = 000
  ALU_Result      = 250 (0x000000fa)
  Zero Flag       = 0
  Dest Reg        = R7
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 0
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R30: 1 -> 2
```

Figure 3-3:cycle #3.

Here we fetched I3 and I2 moved to the decode stage but here we can see that I2 needs the value of R7, but Rr7 is the destination register for I1 and sources register for I2, so the CPU forward the value from the execution stage to the decoder stage.

```
--- FETCH STAGE ---
  PC              = 3 (0x00000003)
  PC_next         = 4 (0x00000004)
  PC+1            = 4 (0x00000004)
  Instruction     = 0x20149380
  Decoded         = AND R10,R9,R7,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x10128100
  Decoded         = OR R9,R8,R2,R0
  PC_FD           = 2
  Opcode          = 2
  Registers       = Rp=0 Rd=9 Rs=8 Rt=2
  BusA (R8)      = 0 (0x00000000)
  BusB (R2)      = 150 (0x00000096)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R9

--- FORWARDING ---
  FWA=01 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 150 (0x00000096)
  BusB_FW         = 150 (0x00000096)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 250 (0x000000fa)
  BUSB_E          = 100 (0x00000064)
  Imm_E           = 128 (0x00000080)
  ALUSrcMux       = 100 (0x00000064)
  ALUOP_E         = 001
  ALU_Result      = 150 (0x00000096)
  Zero Flag       = 0
  Dest Reg        = R8
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 250 (0x000000fa)
  WriteData       = 150 (0x00000096)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R7
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 1
  >>> WRITING R0 = 0 <<<

--- REGISTER CHANGES ---
  R30: 2 -> 3
```

Figure 3-4:cycle #4.

The same as before happening I3 needed the value from R8 so the CPU forwarded it to I3.

And in this cycle I4 was fetched and I3.

```
--- FETCH STAGE ---
  PC              = 4 (0x00000004)
  PC_next         = 5 (0x00000005)
  PC+1            = 5 (0x00000005)
  Instruction     = 0x28161032
  Decoded         = ADDI R11,R1,50,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x20149380
  Decoded         = AND R10,R9,R7,R0
  PC_FD           = 3
  Opcode          = 4
  Registers       = Rp=0 Rd=10 Rs=9 Rt=7
  BusA (R9)     = 0 (0x00000000)
  BusB (R7)     = 0 (0x00000000)
  BusP (R0)     = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R10

--- FORWARDING ---
  FWA=01 FWB=11 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 150 (0x00000096)
  BusB_FW         = 250 (0x000000fa)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 150 (0x00000096)
  BUSB_E          = 150 (0x00000096)
  Imm_E           = 256 (0x00000100)
  ALUSrcMux       = 150 (0x00000096)
  ALUOP_E         = 010
  ALU_Result      = 150 (0x00000096)
  Zero Flag       = 0
  Dest Reg        = R9
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 150 (0x00000096)
  WriteData       = 100 (0x00000064)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R8
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 250 (0x000000fa)
  Dest Reg        = R7
  RegWrite_WB     = 1
  >>> WRITING R7 = 250 <<<

--- REGISTER CHANGES ---
  R30: 3 -> 4
```

Figure 3-5:cycle #5

After the 5 cycle we see that the instruction in I1 is in the WB stage, instruction in I2 in Mem stage, instruction in I3 on the Execute stage, instruction in I4 in Decode stage, instruction in I5 in the fetch stage.

```
--- FETCH STAGE ---
  PC              = 5 (0x00000005)
  PC_next         = 6 (0x00000006)
  PC+1            = 6 (0x00000006)
  Instruction     = 0x30183005
  Decoded         = ORI R12,R3,5,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x28161032
  Decoded         = ADDI R11,R1,50,R0
  PC_FD           = 4
  Opcode          = 5
  Registers       = Rp=0 Rd=11 Rs=1 Rt=0
  BusA (R1)     = 100 (0x00000064)
  BusB (R0)     = 0 (0x00000000)
  BusP (R0)     = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=1 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R11

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 150 (0x00000096)
  BUSB_E          = 250 (0x000000fa)
  Imm_E           = 896 (0x00000380)
  ALUSrcMux       = 250 (0x000000fa)
  ALUOP_E         = 100
  ALU_Result      = 146 (0x00000092)
  Zero Flag       = 0
  Dest Reg        = R10
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 150 (0x00000096)
  WriteData       = 150 (0x00000096)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R9
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 150 (0x00000096)
  Dest Reg        = R8
  RegWrite_WB     = 1
  >>> WRITING R8 = 150 <<<

--- REGISTER CHANGES ---
  R7: 0 -> 250
  R30: 4 -> 5
```

Figure 3-6:cycle #6.

As we can see in fig (2-5) an immediate instruction was fetched, we can see how the CPU deals with the immediate value. These instructions will run from cycle 5 to 16.

```
--- FETCH STAGE ---
  PC              = 6 (0x00000006)
  PC_next         = 7 (0x00000007)
  PC+1            = 7 (0x00000007)
  Instruction     = 0x401a50f0
  Decoded         = ANDI R13,R5,240,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x30183005
  Decoded         = ORI R12,R3,5,R0
  PC_FD           = 5
  Opcode          = 6
  Registers       = Rp=0 Rd=12 Rs=3 Rt=0
  BusA (R3)      = 10 (0x0000000a)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=1 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R12

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 10 (0x0000000a)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 50 (0x00000032)
  ALUSrcMux       = 50 (0x00000032)
  ALUOP_E         = 000
  ALU_Result      = 150 (0x00000096)
  Zero Flag       = 0
  Dest Reg        = R11
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 146 (0x00000092)
  WriteData       = 250 (0x000000fa)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R10
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 150 (0x00000096)
  Dest Reg        = R9
  RegWrite_WB     = 1
  >>> WRITING R9 = 150 <<<

--- REGISTER CHANGES ---
  R8: 0 -> 150
  R30: 5 -> 6
```

Figure 3-7:cycle #7.

In fig (2-7), I7 has been fetched and the value if I2 from the previous test has been written on the register but it was on the WB in cycle 6 due to that the reg file works on the Clk in the next cycle it will write

```
--- FETCH STAGE ---
  PC              = 7 (0x00000007)
  PC_next         = 8 (0x00000008)
  PC+1            = 8 (0x00000008)
  Instruction     = 0x381c3000
  Decoded         = NORI R14,R3,0,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x401a50f0
  Decoded         = ANDI R13,R5,240,R0
  PC_FD           = 6
  Opcode          = 8
  Registers       = Rp=0 Rd=13 Rs=5 Rt=1
  BusA (R5)      = 255 (0x000000ff)
  BusB (R1)      = 100 (0x00000064)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=1 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R13

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 255 (0x000000ff)
  BusB_FW         = 100 (0x00000064)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 10 (0x0000000a)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 5 (0x00000005)
  ALUSrcMux       = 5 (0x00000005)
  ALUOP_E         = 010
  ALU_Result      = 15 (0x0000000f)
  Zero Flag       = 0
  Dest Reg        = R12
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 150 (0x00000096)
  WriteData       = 0 (0x00000000)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R11
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 146 (0x00000092)
  Dest Reg        = R10
  RegWrite_WB     = 1
  >>> WRITING R10 = 146 <<<

--- REGISTER CHANGES ---
  R9: 0 -> 150
  R30: 6 -> 7
```

Figure 3-8:cycle #8.

In fig (2-8) I8 has been fetched and I7 enters the Decode stage and we don't have any forwarding. The result of these instructions will be written from 9 to 12.

And we can see that the value of I3 has been written on the register file.

```
--- FETCH STAGE ---
  PC              = 8 (0x00000008)
  PC_next         = 9 (0x00000009)
  PC+1            = 9 (0x00000009)
  Instruction     = 0x5002000a
  Decoded         = SW R1,10(R0),R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x381c3000
  Decoded         = NORI R14,R3,0,R0
  PC_FD           = 7
  Opcode          = 7
  Registers       = Rp=0 Rd=14 Rs=3 Rt=0
  BusA (R3)      = 10 (0x0000000a)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=1 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R14

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 10 (0x0000000a)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 255 (0x000000ff)
  BUSB_E          = 100 (0x00000064)
  Imm_E           = 240 (0x000000f0)
  ALUSrcMux       = 240 (0x000000f0)
  ALUOP_E         = 100
  ALU_Result      = 240 (0x000000f0)
  Zero Flag       = 0
  Dest Reg        = R13
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 15 (0x0000000f)
  WriteData       = 0 (0x00000000)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R12
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 150 (0x00000096)
  Dest Reg        = R11
  RegWrite_WB     = 1
  >>> WRITING R11 = 150 <<<

--- REGISTER CHANGES ---
  R10: 0 -> 146
  R30: 7 -> 8
```

Figure 3-9:cycle #9

In fig (2-9), we fetched I9 (SW) and we can see that the I8 (NORI) is in the decode stage and I7 in the execute stage, I6 memory stage and I5 in the WB.

```
================================================================================
CYCLE 10 @ time=1250000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 9 (0x00000009)
  PC_next         = 10 (0x0000000a)
  PC+1            = 10 (0x0000000a)
  Instruction     = 0x481e000a
  Decoded         = LW R15,10(R0),R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x5002000a
  Decoded         = SW R1,10(R0),R0
  PC_FD           = 8
  Opcode          = 10
  Registers       = Rp=0 Rd=1 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 100 (0x00000064)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=0 ALUSrc=1 MemRd=0 MemWr=1 WB=00 RegSel=0 DestReg=R1

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 100 (0x00000064)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 10 (0x0000000a)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 011
  ALU_Result      = 4294967285 (0xfffffff5)
  Zero Flag       = 0
  Dest Reg        = R14
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 240 (0x000000f0)
  WriteData       = 100 (0x00000064)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R13
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 15 (0x0000000f)
  Dest Reg        = R12
  RegWrite_WB     = 1
  >>> WRITING R12 = 15 <<<

--- REGISTER CHANGES ---
  R11: 0 -> 150
  R30: 8 -> 9
```

Figure 3-10:cycle #10.

In fig (2-10) the load instruction is fetched (I10) and the SW moved to Decode stage and the result of I5 writes its value on the R11 and I6 moved to the WB stage, when the WS instruction moved to the execution stage we will find the address to store the value of R1 after that the LW instruction move to the execution stage we will calculate the address that we will read the value from it and store it in R15. But when the Load instruction (I10) was in the decode stage I11 was fetched (cycle 11) in the next cycle (12) I11 moved to the decode stage and I12 was fetched But in I9 still did not read the data from the data memory because of this the CPU made a Stall in the execution stage at cycle 13.

```
================================================================================
CYCLE 11 @ time=1350000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 10 (0x0000000a)
  PC_next         = 11 (0x0000000b)
  PC+1            = 11 (0x0000000b)
  Instruction     = 0x0020f100
  Decoded         = ADD R16,R15,R2,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x481e000a
  Decoded         = LW R15,10(R0),R0
  PC_FD           = 9
  Opcode          = 9
  Registers       = Rp=0 Rd=15 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=1 MemRd=1 MemWr=0 WB=01 RegSel=0 DestReg=R15

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 100 (0x00000064)
  Imm_E           = 10 (0x0000000a)
  ALUSrcMux       = 10 (0x0000000a)
  ALUOP_E         = 000
  ALU_Result      = 10 (0x0000000a)
  Zero Flag       = 0
  Dest Reg        = R1
  RegWrite_E      = 0

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 4294967285 (0xfffffff5)
  WriteData       = 0 (0x00000000)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R14
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 240 (0x000000f0)
  Dest Reg        = R13
  RegWrite_WB     = 1
  >>> WRITING R13 = 240 <<<

--- REGISTER CHANGES ---
  R12: 0 -> 15
  R30: 9 -> 10
```

Figure 3-11:cycle #11.

In this cycle, I11 has been fetched and I10 enter the decode stage. in this cycle we don't have stall or forwarding.

I9 in the execution stage to calculate the address to store the value of R1, I8 in the memory stage and I7 in the WB stage and the value from I7 is ready to be written on the register file, the changes that happened on the reg file made on R12 and R30

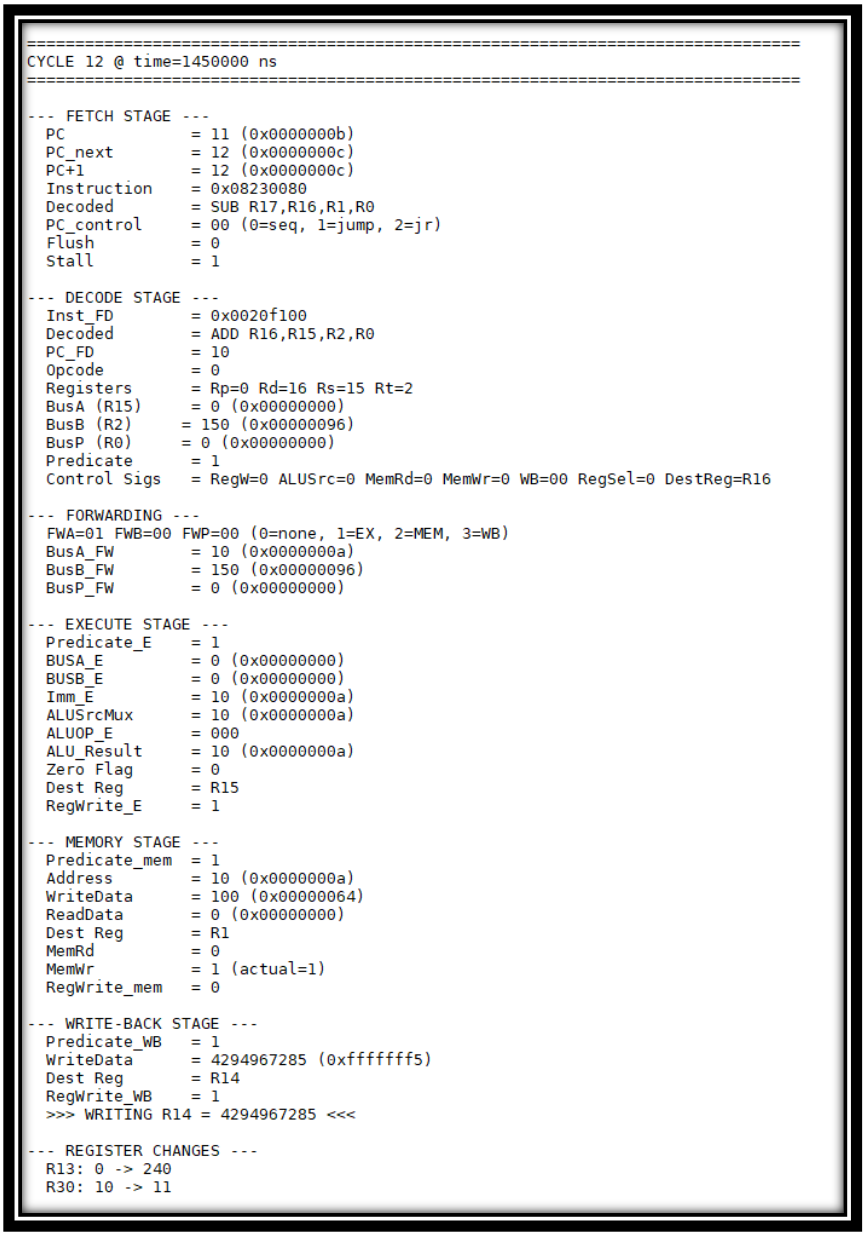


Figure 3-12:cycle #12.

In this cycle I12 has been fetched, and I11 entered the decode stage. but I11 need the value that I10 with write it on the R15 so here we need to forward the value from the execution stage to the decode stage but I10 is load instruction that will read the value from the memory and so in this case we will make a stall in the execution stage, see fig (14).

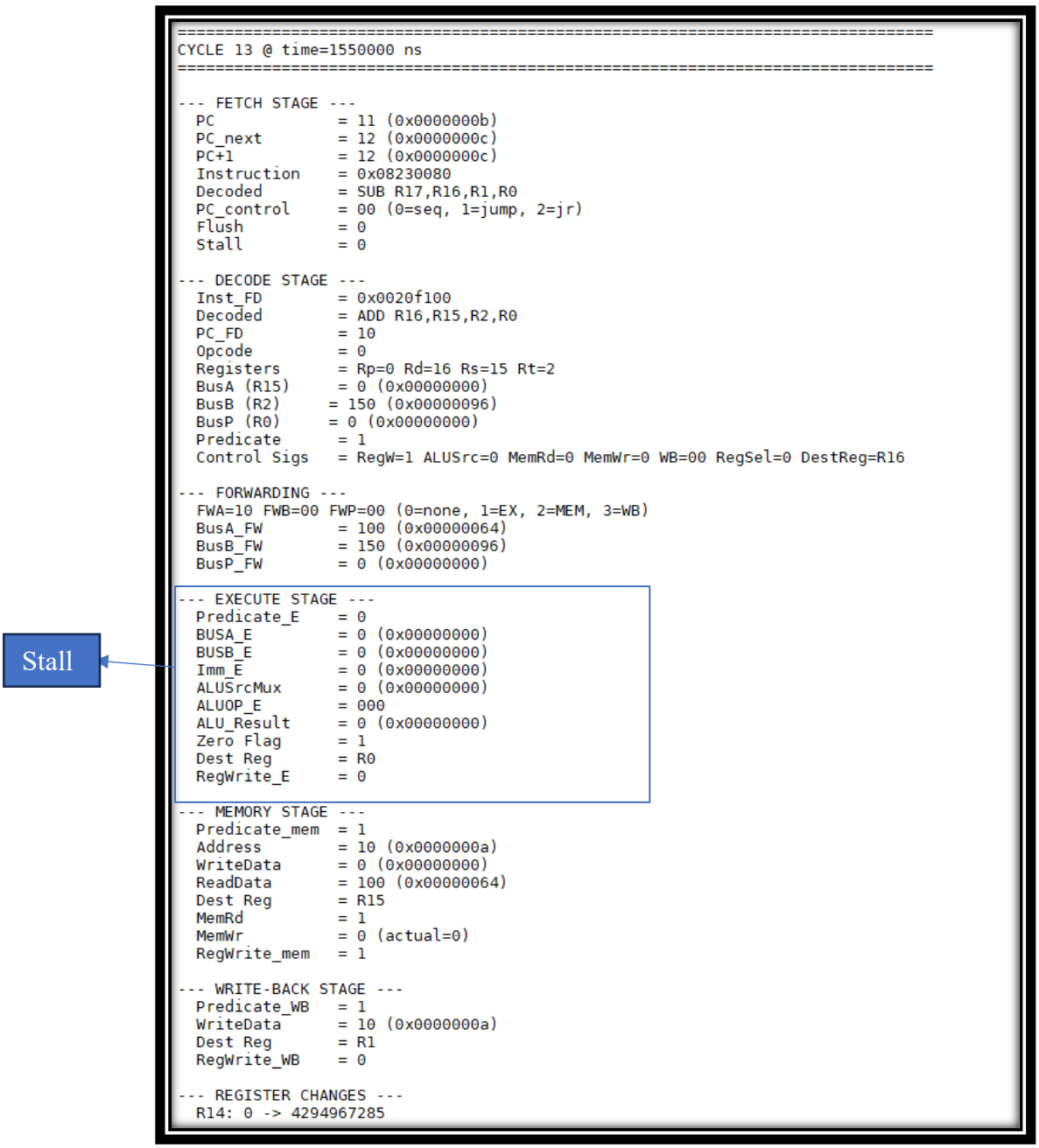


Figure 3-13:cycle #13.

in the fig (2-13) we can see that I13 has been fetched and I12 enter the decode stage but if we look at the execute stage, we can see that it contains zeros.

```
================================================================================
CYCLE 14 @ time=1650000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 12 (0x0000000c)
  PC_next         = 13 (0x0000000d)
  PC+1            = 13 (0x0000000d)
  Instruction     = 0x05241100
  Decoded         = ADD R18,R1,R2,R20
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x08230080
  Decoded         = SUB R17,R16,R1,R0
  PC_FD           = 11
  Opcode          = 1
  Registers       = Rp=0 Rd=17 Rs=16 Rt=1
  BusA (R16)      = 0 (0x00000000)
  BusB (R1)       = 100 (0x00000064)
  BusP (R0)       = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R17

--- FORWARDING ---
  FWA=01 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 250 (0x000000fa)
  BusB_FW         = 100 (0x00000064)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 150 (0x00000096)
  Imm_E           = 256 (0x00000100)
  ALUSrcMux       = 150 (0x00000096)
  ALUOP_E         = 000
  ALU_Result      = 250 (0x000000fa)
  Zero Flag       = 0
  Dest Reg        = R16
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 0
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 0

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 100 (0x00000064)
  Dest Reg        = R15
  RegWrite_WB     = 1
  >>> WRITING R15 = 100 <<<

--- REGISTER CHANGES ---
  R30: 11 -> 12
```

Figure 3-14:cycle #14.

Here is a normal execution.

```
CYCLE 15 @ time=1750000 ns

--- FETCH STAGE ---
  PC              = 13 (0x0000000d)
  PC_next         = 14 (0x0000000e)
  PC+1            = 14 (0x0000000e)
  Instruction     = 0x05661100
  Decoded         = ADD R19,R1,R2,R21
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x05241100
  Decoded         = ADD R18,R1,R2,R20
  PC_FD           = 12
  Opcode          = 0
  Registers       = Rp=20 Rd=18 Rs=1 Rt=2
  BusA (R1)     = 100 (0x00000064)
  BusB (R2)     = 150 (0x00000096)
  BusP (R20)      = 1 (0x00000001)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R18

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 150 (0x00000096)
  BusP_FW         = 1 (0x00000001)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 250 (0x000000fa)
  BUSB_E          = 100 (0x00000064)
  Imm_E           = 128 (0x00000080)
  ALUSrcMux       = 100 (0x00000064)
  ALUOP_E         = 001
  ALU_Result      = 150 (0x00000096)
  Zero Flag       = 0
  Dest Reg        = R17
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 250 (0x000000fa)
  WriteData       = 150 (0x00000096)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R16
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 0
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R15: 0 -> 100
  R30: 12 -> 13
```

I1

```
CYCLE 16 @ time=1850000 ns

--- FETCH STAGE ---
  PC              = 14 (0x0000000e)
  PC_next         = 15 (0x0000000f)
  PC+1            = 15 (0x0000000f)
  Instruction     = 0x5544000f
  Decoded         = SW R2,15(R0),R21
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x05661100
  Decoded         = ADD R19,R1,R2,R21
  PC_FD           = 13
  Opcode          = 0
  Registers       = Rp=21 Rd=19 Rs=1 Rt=2
  BusA (R1)     = 100 (0x00000064)
  BusB (R2)     = 150 (0x00000096)
  BusP (R21)      = 0 (0x00000000)
  Predicate       = 0
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R19

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 150 (0x00000096)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 150 (0x00000096)
  Imm_E           = 256 (0x00000100)
  ALUSrcMux       = 150 (0x00000096)
  ALUOP_E         = 000
  ALU_Result      = 250 (0x000000fa)
  Zero Flag       = 0
  Dest Reg        = R18
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 150 (0x00000096)
  WriteData       = 100 (0x00000064)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R17
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 250 (0x000000fa)
  Dest Reg        = R16
  RegWrite_WB     = 1
  >>> WRITING R16 = 250 <<<

--- REGISTER CHANGES ---
  R30: 13 -> 14
```

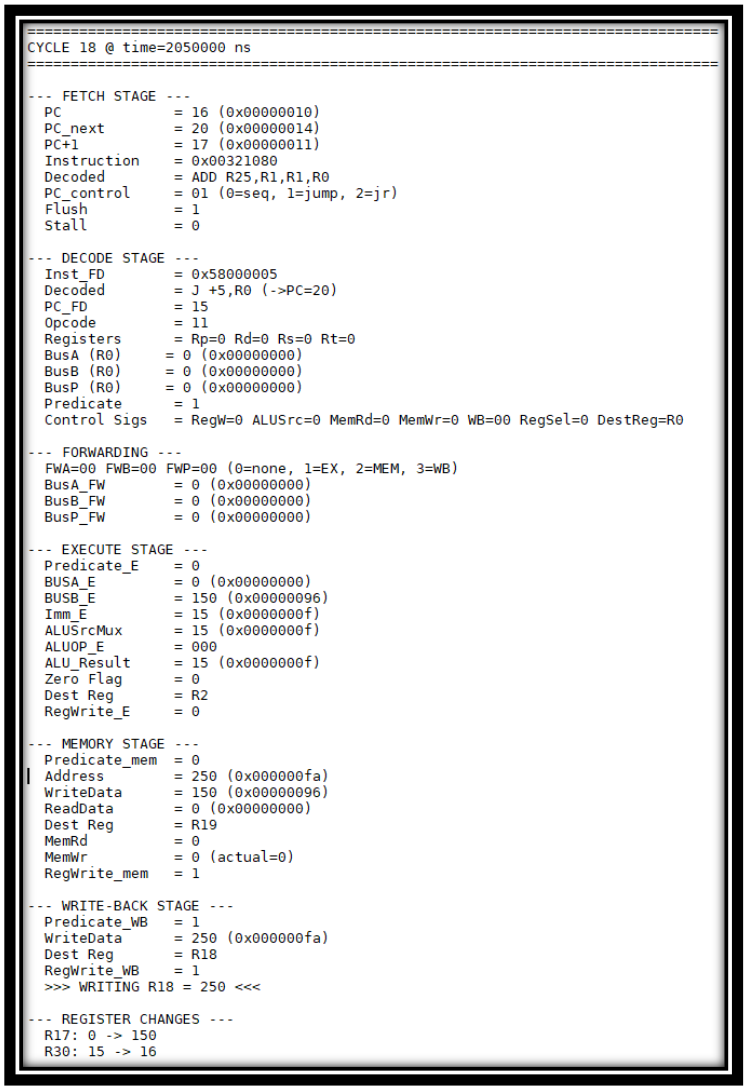
I15

```
CYCLE 17 @ time=1950000 ns

--- FETCH STAGE ---
  PC              = 15 (0x0000000f)
  PC_next         = 16 (0x00000010)
  PC+1            = 16 (0x00000010)
  Instruction     = 0x58000005
  Decoded         = J +5,R0 (->PC=20)
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x5544000f
  Decoded         = SW R2,15(R0),R21
  PC_FD           = 14
  Opcode          = 10
  Registers       = Rp=21 Rd=2 Rs=0 Rt=0
  BusA (R0)     = 0 (0x00000000)
  BusB (R0)     = 150 (0x00000096)
  BusP (R21)      = 0 (0x00000000)
  Predicate       = 0
  Control Sigs    = RegW=0 ALUSrc=1 MemRd=0 MemWr=1 WB=00 RegSel=0 DestReg=R2

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 150 (0x00000096)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 0
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 150 (0x00000096)
  Imm_E           = 256 (0x00000100)
  ALUSrcMux       = 150 (0x00000096)
  ALUOP_E         = 000
  ALU_Result      = 250 (0x000000fa)
  Zero Flag       = 0
  Dest Reg        = R19
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 250 (0x000000fa)
  WriteData       = 150 (0x00000096)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R18
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 150 (0x00000096)
  Dest Reg        = R17
  RegWrite_WB     = 1
  >>> WRITING R17 = 150 <<<

--- REGISTER CHANGES ---
  R16: 0 -> 250
  R30: 14 -> 15
```

Figure 3-15:cycle #15,16,17.

By looking at fig (2-15), we can see that the Predicate is zero for both I14 and I15 so the result for these two instructions won't be stored in the register file or the data memory.

At the cycle number 17 I16 was fetched and it was a jump instruction.



```
================================================================================
CYCLE 18 @ time=2050000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 16 (0x00000010)
  PC_next         = 20 (0x00000014)
  PC+1            = 17 (0x00000011)
  Instruction     = 0x00321080
  Decoded         = ADD R25,R1,R1,R0
  PC_control      = 01 (0=seq, 1=jump, 2=jr)
  Flush           = 1
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x58000005
  Decoded         = J +5,R0 (->PC=20)
  PC_FD           = 15
  Opcode          = 11
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)     = 0 (0x00000000)
  BusB (R0)     = 0 (0x00000000)
  BusP (R0)     = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=0 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 0
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 150 (0x00000096)
  Imm_E           = 15 (0x0000000f)
  ALUSrcMux       = 15 (0x0000000f)
  ALUOP_E         = 000
  ALU_Result      = 15 (0x0000000f)
  Zero Flag       = 0
  Dest Reg        = R2
  RegWrite_E      = 0

--- MEMORY STAGE ---
  Predicate_mem   = 0
  Address         = 250 (0x000000fa)
  WriteData       = 150 (0x00000096)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R19
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 250 (0x000000fa)
  Dest Reg        = R18
  RegWrite_WB     = 1
  >>> WRITING R18 = 250 <<<

--- REGISTER CHANGES ---
  R17: 0 -> 150
  R30: 15 -> 16
```

Figure 3-16:cycle #18.

If we look at fig (2-16) we can see that the next value for the PC is 20 and the instruction that is the fetch stage will be discarded and this will lead the CPU to have a stall cycle that means after cycle 18 a NOP instruction will enter the decode stage.

```
================================================================================
CYCLE 19 @ time=2150000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 20 (0x00000014)
  PC_next         = 21 (0x00000015)
  PC+1            = 21 (0x00000015)
  Instruction     = 0x002c3200
  Decoded         = ADD R22,R3,R4,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 0
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 5 (0x00000005)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 0

--- MEMORY STAGE ---
  Predicate_mem   = 0
  Address         = 15 (0x0000000f)
  WriteData       = 150 (0x00000096)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R2
  MemRd           = 0
  MemWr           = 1 (actual=0)
  RegWrite_mem    = 0

--- WRITE-BACK STAGE ---
  Predicate_WB    = 0
  WriteData       = 250 (0x000000fa)
  Dest Reg        = R19
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R18: 0 -> 250
  R20: 16 -> 20
```

Figure 3-17:cycle #19.

In this cycle we can see that the instruction that in the decode stage is NOP instruction and the instruction in the fetch stage is I17 that is in PC 20 and the instruction from PC 16 to 19 are skipped.

And if we look at the WB stage, we can see that I14 did not execute.

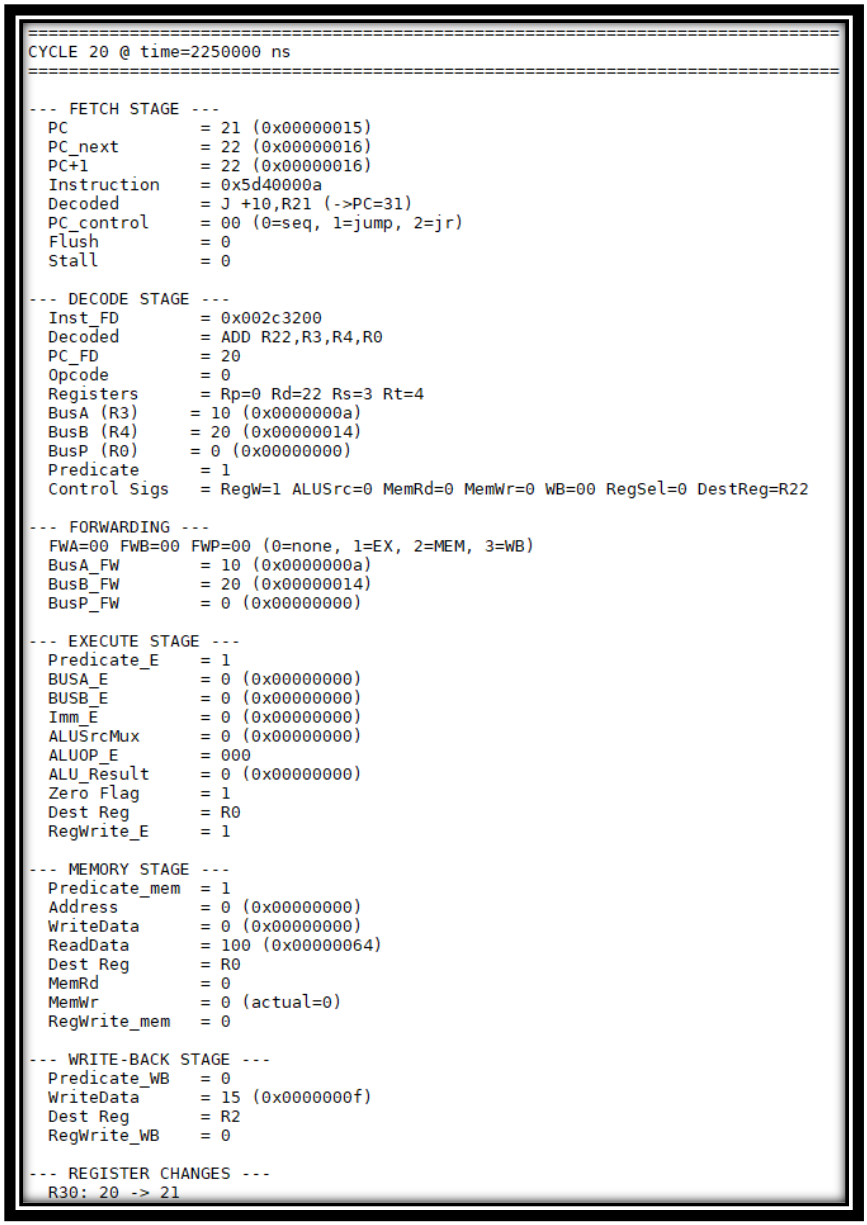


Figure 3-18:cycle #20.

In fig (2-18) we can see that another jump instruction was fetched and I15 moved to WB stage and it did not execute.

```
================================================================================
CYCLE 21 @ time=2350000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 22 (0x00000016)
  PC_next         = 23 (0x00000017)
  PC+1            = 23 (0x00000017)
  Instruction     = 0x002e2180
  Decoded         = ADD R23,R2,R3,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x5d40000a
  Decoded         = J +10,R21 (->PC=31)
  PC_FD           = 21
  Opcode          = 11
  Registers       = Rp=21 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R21)      = 0 (0x00000000)
  Predicate       = 0
  Control Sigs    = RegW=0 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 10 (0x0000000a)
  BUSB_E          = 20 (0x00000014)
  Imm_E           = 512 (0x00000200)
  ALUSrcMux       = 20 (0x00000014)
  ALUOP_E         = 000
  ALU_Result      = 30 (0x0000001e)
  Zero Flag       = 0
  Dest Reg        = R22
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R30: 21 -> 22
```

Figure 3-19:cycle #21.

We can see that the jump instruction did not execute because the Predicate for it was 0 and the next value of the PC will be 22 that been fetched.

```
================================================================================
CYCLE 22 @ time=2450000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 23 (0x00000017)
  PC_next         = 24 (0x00000018)
  PC+1            = 24 (0x00000018)
  Instruction     = 0x60000003
  Decoded         = CALL +3,R0 (->PC=26)
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x002e2180
  Decoded         = ADD R23,R2,R3,R0
  PC_FD           = 22
  Opcode          = 0
  Registers       = Rp=0 Rd=23 Rs=2 Rt=3
  BusA (R2)      = 150 (0x00000096)
  BusB (R3)      = 10 (0x0000000a)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R23

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 150 (0x00000096)
  BusB_FW         = 10 (0x0000000a)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 0
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 10 (0x0000000a)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 0

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 30 (0x0000001e)
  WriteData       = 20 (0x00000014)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R22
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 1
  >>> WRITING R0 = 0 <<<

--- REGISTER CHANGES ---
  R30: 22 -> 23
```

Figure 3-20:cycle #22.

In fig (2-20) we can see that a Call(I20) instruction was fetched.

```
================================================================================
CYCLE 23 @ time=2550000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 24 (0x00000018)
  PC_next         = 26 (0x0000001a)
  PC+1            = 25 (0x00000019)
  Instruction     = 0x58000004
  Decoded         = J +4,R0 (->PC=28)
  PC_control      = 01 (0=seq, 1=jump, 2=jr)
  Flush           = 1
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x60000003
  Decoded         = CALL +3,R0 (->PC=26)
  PC_FD           = 23
  Opcode          = 12
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=10 RegSel=1 DestReg=R31

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 150 (0x00000096)
  BUSB_E          = 10 (0x0000000a)
  Imm_E           = 384 (0x00000180)
  ALUSrcMux       = 10 (0x0000000a)
  ALUOP_E         = 000
  ALU_Result      = 160 (0x000000a0)
  Zero Flag       = 0
  Dest Reg        = R23
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 0
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 0

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 30 (0x0000001e)
  Dest Reg        = R22
  RegWrite_WB     = 1
  >>> WRITING R22 = 30 <<<

--- REGISTER CHANGES ---
  R30: 23 -> 24
```

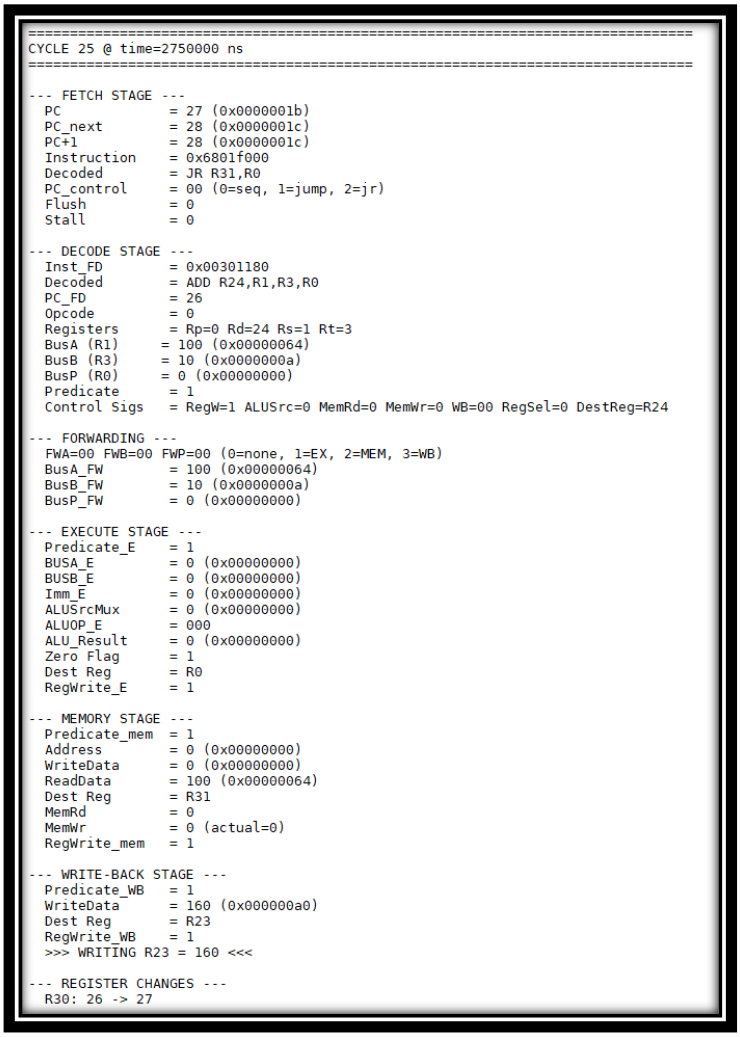
Figure 3-21:cycle #23.

In fig (2-21) we can see that the next PC is 26 not 24 so the instructions that will enter the decode stage after this cycle are NOP, and if we look at the detReg it is R31 that will contain the value of the PC.

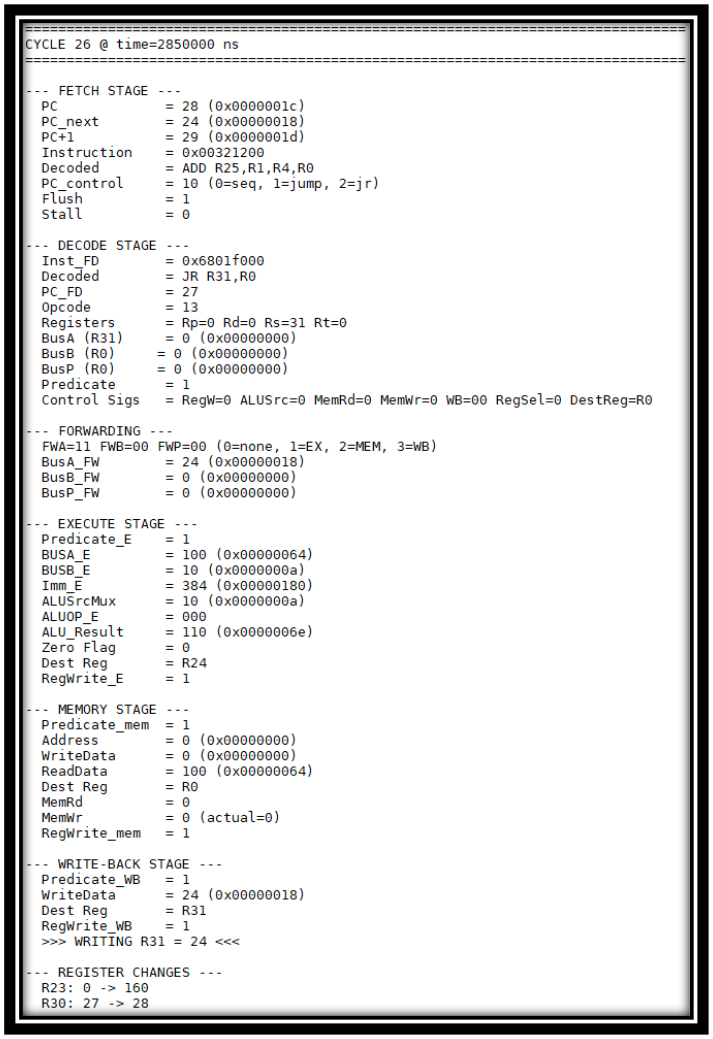
```
================================================================================
CYCLE 24 @ time=2650000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 26 (0x0000001a)
  PC_next         = 27 (0x0000001b)
  PC+1            = 27 (0x0000001b)
  Instruction     = 0x00301180
  Decoded         = ADD R24,R1,R3,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 0
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 3 (0x00000003)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R31
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 160 (0x000000a0)
  WriteData       = 10 (0x0000000a)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R23
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 0
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R22: 0 -> 30
  R30: 24 -> 26
```

Figure 3-22:cycle #24.

In fig (2-22) we can see that the instruction that fetched was I23 and the NOP instruction entered the decode stage.



```
================================================================================
CYCLE 25 @ time=2750000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 27 (0x0000001b)
  PC_next         = 28 (0x0000001c)
  PC+1            = 28 (0x0000001c)
  Instruction     = 0x6801f000
  Decoded         = JR R31,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00301180
  Decoded         = ADD R24,R1,R3,R0
  PC_FD           = 26
  Opcode          = 0
  Registers       = Rp=0 Rd=24 Rs=1 Rt=3
  BusA (R1)     = 100 (0x00000064)
  BusB (R3)     = 10 (0x0000000a)
  BusP (R0)     = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R24

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 10 (0x0000000a)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R31
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 160 (0x000000a0)
  Dest Reg        = R23
  RegWrite_WB     = 1
  >>> WRITING R23 = 160 <<<

--- REGISTER CHANGES ---
  R30: 26 -> 27
```



```
================================================================================
CYCLE 26 @ time=2850000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 28 (0x0000001c)
  PC_next         = 24 (0x00000018)
  PC+1            = 29 (0x0000001d)
  Instruction     = 0x00321200
  Decoded         = ADD R25,R1,R4,R0
  PC_control      = 10 (0=seq, 1=jump, 2=jr)
  Flush           = 1
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x6801f000
  Decoded         = JR R31,R0
  PC_FD           = 27
  Opcode          = 13
  Registers       = Rp=0 Rd=0 Rs=31 Rt=0
  BusA (R31)      = 0 (0x00000000)
  BusB (R0)     = 0 (0x00000000)
  BusP (R0)     = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=0 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=11 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 24 (0x00000018)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 10 (0x0000000a)
  Imm_E           = 384 (0x00000180)
  ALUSrcMux       = 10 (0x0000000a)
  ALUOP_E         = 000
  ALU_Result      = 110 (0x0000006e)
  Zero Flag       = 0
  Dest Reg        = R24
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 24 (0x00000018)
  Dest Reg        = R31
  RegWrite_WB     = 1
  >>> WRITING R31 = 24 <<<

--- REGISTER CHANGES ---
  R23: 0 -> 160
  R30: 27 -> 28
```

Figure 3-23:cycle #25,26.

If we look at fig (2-23) we can see the instruction that was fetched is I24 (JR, cycle 25) this instruction will jump to the value that in R31 and this value should be store after I20 is done execute and in cycle 26 the call instructions was in the WB stage and still did not write the value of it, so the value of R31 was forwarded to JR instruction in the decode stage.

And we can see that the next value of the PC is 24

```
================================================================================
CYCLE 27 @ time=2950000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 24 (0x00000018)
  PC_next         = 25 (0x00000019)
  PC+1            = 25 (0x00000019)
  Instruction     = 0x58000004
  Decoded         = J +4,R0 (->PC=28)
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 0
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 24 (0x00000018)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 24 (0x00000018)
  Zero Flag       = 0
  Dest Reg        = R0
  RegWrite_E      = 0

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 110 (0x0000006e)
  WriteData       = 10 (0x0000000a)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R24
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 1
  >>> WRITING R0 = 0 <<<

--- REGISTER CHANGES ---
  R30: 28 -> 24
  R31: 0 -> 24
```

Figure 3-24:cycle #27.

Here after we the JR instruction has done, we return to PC 24 and this PC we have a jump instruction, so we won't end up in infinite loop.

After this a normal execution happen to the end of the program.

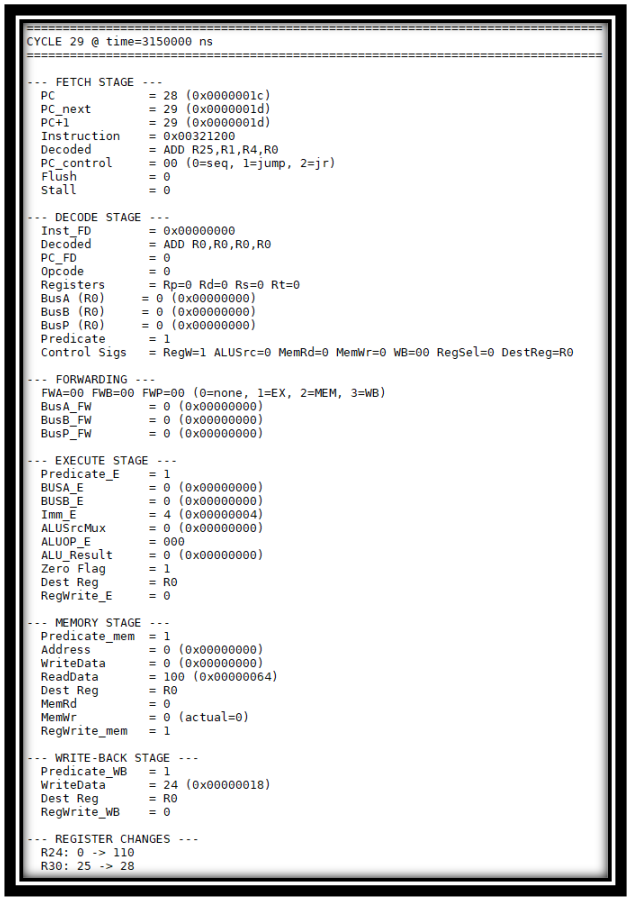
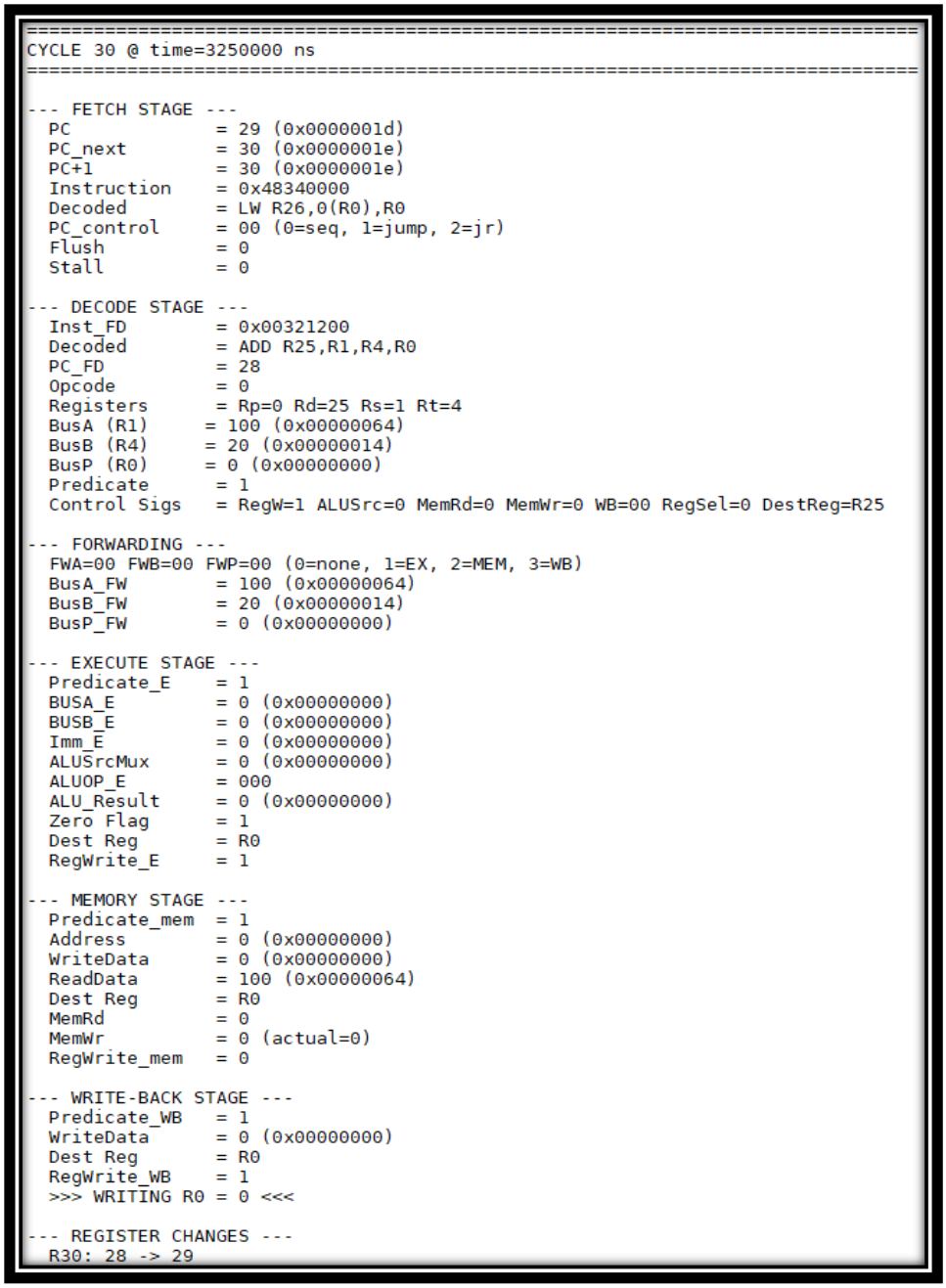


Figure 3-25:cycle #28,29.

here in cycle 28 I23 has entered the WB stage, and in the next cycle we moved to PC 28 and fetched I25 and I23 write its value on the register file.



```
================================================================================
CYCLE 30 @ time=3250000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 29 (0x0000001d)
  PC_next         = 30 (0x0000001e)
  PC+1            = 30 (0x0000001e)
  Instruction     = 0x48340000
  Decoded         = LW R26,0(R0),R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00321200
  Decoded         = ADD R25,R1,R4,R0
  PC_FD           = 28
  Opcode          = 0
  Registers       = Rp=0 Rd=25 Rs=1 Rt=4
  BusA (R1)      = 100 (0x00000064)
  BusB (R4)      = 20 (0x00000014)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R25

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 20 (0x00000014)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 0

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 1
  >>> WRITING R0 = 0 <<<

--- REGISTER CHANGES ---
  R30: 28 -> 29
```

Figure 3-26:cycle #30.

```
L: ================================================================================
L: CYCLE 31 @ time=3350000 ns
L: ================================================================================
L:
L: --- FETCH STAGE ---
L:   PC              = 30 (0x0000001e)
L:   PC_next         = 31 (0x0000001f)
L:   PC+1            = 31 (0x0000001f)
L:   Instruction     = 0x48360008
L:   Decoded         = LW R27,8(R0),R0
L:   PC_control      = 00 (0=seq, 1=jump, 2=jr)
L:   Flush           = 0
L:   Stall           = 0
L:
L: --- DECODE STAGE ---
L:   Inst_FD         = 0x48340000
L:   Decoded         = LW R26,0(R0),R0
L:   PC_FD           = 29
L:   Opcode          = 9
L:   Registers       = Rp=0 Rd=26 Rs=0 Rt=0
L:   BusA (R0)      = 0 (0x00000000)
L:   BusB (R0)      = 0 (0x00000000)
L:   BusP (R0)      = 0 (0x00000000)
L:   Predicate       = 1
L:   Control Sigs    = RegW=1 ALUSrc=1 MemRd=1 MemWr=0 WB=01 RegSel=0 DestReg=R26
L:
L: --- FORWARDING ---
L:   FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
L:   BusA_FW         = 0 (0x00000000)
L:   BusB_FW         = 0 (0x00000000)
L:   BusP_FW         = 0 (0x00000000)
L:
L: --- EXECUTE STAGE ---
L:   Predicate_E     = 1
L:   BUSA_E          = 100 (0x00000064)
L:   BUSB_E          = 20 (0x00000014)
L:   Imm_E           = 512 (0x00000200)
L:   ALUSrcMux       = 20 (0x00000014)
L:   ALUOP_E         = 000
L:   ALU_Result      = 120 (0x00000078)
L:   Zero Flag       = 0
L:   Dest Reg        = R25
L:   RegWrite_E      = 1
L:
L: --- MEMORY STAGE ---
L:   Predicate_mem   = 1
L:   Address         = 0 (0x00000000)
L:   WriteData       = 0 (0x00000000)
L:   ReadData        = 100 (0x00000064)
L:   Dest Reg        = R0
L:   MemRd           = 0
L:   MemWr           = 0 (actual=0)
L:   RegWrite_mem    = 1
L:
L: --- WRITE-BACK STAGE ---
L:   Predicate_WB    = 1
L:   WriteData       = 0 (0x00000000)
L:   Dest Reg        = R0
L:   RegWrite_WB     = 0
L:
L: --- REGISTER CHANGES ---
L:   R30: 29 -> 30
L:
```

Figure 3-27:cycle #31

```
================================================================================
CYCLE 32 @ time=3450000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 31 (0x0000001f)
  PC_next         = 32 (0x00000020)
  PC+1            = 32 (0x00000020)
  Instruction     = 0x0039ad80
  Decoded         = ADD R28,R26,R27,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x48360008
  Decoded         = LW R27,8(R0),R0
  PC_FD           = 30
  Opcode          = 9
  Registers       = Rp=0 Rd=27 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=1 MemRd=1 MemWr=0 WB=01 RegSel=0 DestReg=R27

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R26
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 120 (0x00000078)
  WriteData       = 20 (0x00000014)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R25
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 1
  >>> WRITING R0 = 0 <<<

--- REGISTER CHANGES ---
  R30: 30 -> 31
```

Figure 3-28:cycle # 32.

```
================================================================================
CYCLE 33 @ time=3550000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 32 (0x00000020)
  PC_next         = 33 (0x00000021)
  PC+1            = 33 (0x00000021)
  Instruction     = 0x183a3200
  Decoded         = NOR R29,R3,R4,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 1

--- DECODE STAGE ---
  Inst_FD         = 0x0039ad80
  Decoded         = ADD R28,R26,R27,R0
  PC_FD           = 31
  Opcode          = 0
  Registers       = Rp=0 Rd=28 Rs=26 Rt=27
  BusA (R26)      = 0 (0x00000000)
  BusB (R27)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=0 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R28

--- FORWARDING ---
  FWA=10 FWB=01 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 8 (0x00000008)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 8 (0x00000008)
  ALUSrcMux       = 8 (0x00000008)
  ALUOP_E         = 000
  ALU_Result      = 8 (0x00000008)
  Zero Flag       = 0
  Dest Reg        = R27
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R26
  MemRd           = 1
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 120 (0x00000078)
  Dest Reg        = R25
  RegWrite_WB     = 1
  >>> WRITING R25 = 120 <<<

--- REGISTER CHANGES ---
  R30: 31 -> 32
```

Figure 3-29:cycle #33.

```
================================================================================
CYCLE 33 @ time=3550000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 32 (0x00000020)
  PC_next         = 33 (0x00000021)
  PC+1            = 33 (0x00000021)
  Instruction     = 0x183a3200
  Decoded         = NOR R29,R3,R4,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 1

--- DECODE STAGE ---
  Inst_FD         = 0x0039ad80
  Decoded         = ADD R28,R26,R27,R0
  PC_FD           = 31
  Opcode          = 0
  Registers       = Rp=0 Rd=28 Rs=26 Rt=27
  BusA (R26)      = 0 (0x00000000)
  BusB (R27)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=0 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R28

--- FORWARDING ---
  FWA=10 FWB=01 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 8 (0x00000008)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 8 (0x00000008)
  ALUSrcMux       = 8 (0x00000008)
  ALUOP_E         = 000
  ALU_Result      = 8 (0x00000008)
  Zero Flag       = 0
  Dest Reg        = R27
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R26
  MemRd           = 1
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 120 (0x00000078)
  Dest Reg        = R25
  RegWrite_WB     = 1
  >>> WRITING R25 = 120 <<<

--- REGISTER CHANGES ---
  R30: 31 -> 32
```

Figure 3-30:cycle #34.

```
================================================================================
CYCLE 34 @ time=3650000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 32 (0x00000020)
  PC_next         = 33 (0x00000021)
  PC+1            = 33 (0x00000021)
  Instruction     = 0x183a3200
  Decoded         = NOR R29,R3,R4,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x0039ad80
  Decoded         = ADD R28,R26,R27,R0
  PC_FD           = 31
  Opcode          = 0
  Registers       = Rp=0 Rd=28 Rs=26 Rt=27
  BusA (R26)      = 0 (0x00000000)
  BusB (R27)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R28

--- FORWARDING ---
  FWA=11 FWB=10 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 100 (0x00000064)
  BusB_FW         = 170 (0x000000aa)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 0
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 0

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 8 (0x00000008)
  WriteData       = 0 (0x00000000)
  ReadData        = 170 (0x000000aa)
  Dest Reg        = R27
  MemRd           = 1
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 100 (0x00000064)
  Dest Reg        = R26
  RegWrite_WB     = 1
  >>> WRITING R26 = 100 <<<

--- REGISTER CHANGES ---
  R25: 0 -> 120
```

Figure 3-31:cycle #34

```
================================================================================
CYCLE 35 @ time=3750000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 33 (0x00000021)
  PC_next         = 34 (0x00000022)
  PC+1            = 34 (0x00000022)
  Instruction     = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x183a3200
  Decoded         = NOR R29,R3,R4,R0
  PC_FD           = 32
  Opcode          = 3
  Registers       = Rp=0 Rd=29 Rs=3 Rt=4
  BusA (R3)      = 10 (0x0000000a)
  BusB (R4)      = 20 (0x00000014)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R29

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 10 (0x0000000a)
  BusB_FW         = 20 (0x00000014)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 100 (0x00000064)
  BUSB_E          = 170 (0x000000aa)
  Imm_E           = 3456 (0x00000d80)
  ALUSrcMux       = 170 (0x000000aa)
  ALUOP_E         = 000
  ALU_Result      = 270 (0x0000010e)
  Zero Flag       = 0
  Dest Reg        = R28
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 0
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 0

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 170 (0x000000aa)
  Dest Reg        = R27
  RegWrite_WB     = 1
  >>> WRITING R27 = 170 <<<

--- REGISTER CHANGES ---
  R26: 0 -> 100
  R30: 32 -> 33
```

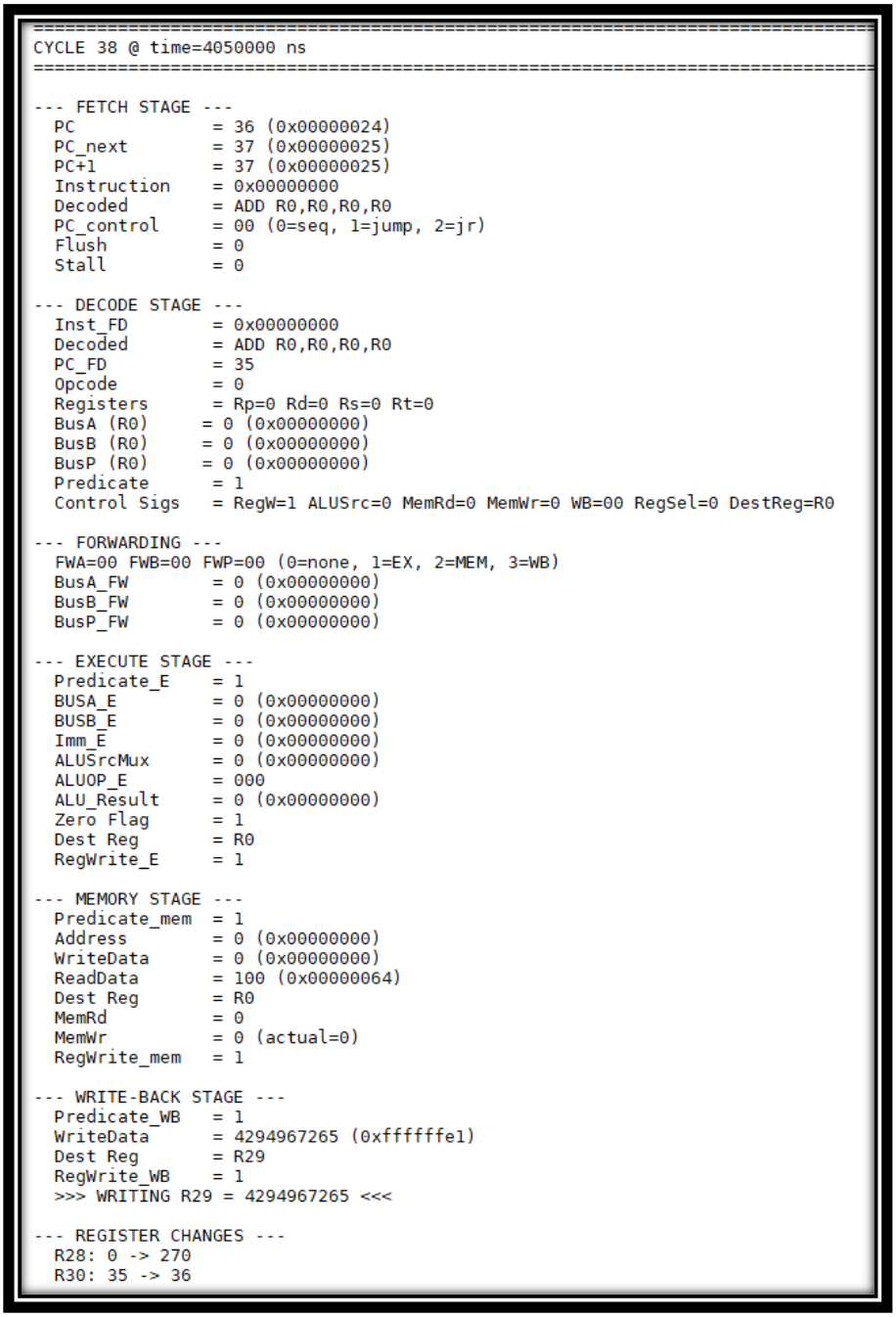
Figure 3-32:cycle #35

```
================================================================================
CYCLE 36 @ time=3850000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 34 (0x00000022)
  PC_next         = 35 (0x00000023)
  PC+1            = 35 (0x00000023)
  Instruction     = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 33
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 10 (0x0000000a)
  BUSB_E          = 20 (0x00000014)
  Imm_E           = 512 (0x00000200)
  ALUSrcMux       = 20 (0x00000014)
  ALUOP_E         = 011
  ALU_Result      = 4294967265 (0xfffffffe1)
  Zero Flag       = 0
  Dest Reg        = R29
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 270 (0x0000010e)
  WriteData       = 170 (0x000000aa)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R28
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 0
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 0

--- REGISTER CHANGES ---
  R27: 0 -> 170
  R30: 33 -> 34
```

Figure 3-33:cycle #36.

```
================================================================================
CYCLE 37 @ time=3950000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 35 (0x00000023)
  PC_next         = 36 (0x00000024)
  PC+1            = 36 (0x00000024)
  Instruction     = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 34
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 4294967265 (0xffffffe1)
  WriteData       = 20 (0x00000014)
  ReadData        = 0 (0x00000000)
  Dest Reg        = R29
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 270 (0x0000010e)
  Dest Reg        = R28
  RegWrite_WB     = 1
  >>> WRITING R28 = 270 <<<

--- REGISTER CHANGES ---
  R30: 34 -> 35
```

Figure 3-34:cycle #37.



```
================================================================================
CYCLE 38 @ time=4050000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 36 (0x00000024)
  PC_next         = 37 (0x00000025)
  PC+1            = 37 (0x00000025)
  Instruction     = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 35
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 4294967265 (0xffffffe1)
  Dest Reg        = R29
  RegWrite_WB     = 1
  >>> WRITING R29 = 4294967265 <<<

--- REGISTER CHANGES ---
  R28: 0 -> 270
  R30: 35 -> 36
```

Figure 3-35:cycle #38

```
================================================================================
CYCLE 39 @ time=4150000 ns
================================================================================

--- FETCH STAGE ---
  PC              = 37 (0x00000025)
  PC_next         = 38 (0x00000026)
  PC+1            = 38 (0x00000026)
  Instruction     = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_control      = 00 (0=seq, 1=jump, 2=jr)
  Flush           = 0
  Stall           = 0

--- DECODE STAGE ---
  Inst_FD         = 0x00000000
  Decoded         = ADD R0,R0,R0,R0
  PC_FD           = 36
  Opcode          = 0
  Registers       = Rp=0 Rd=0 Rs=0 Rt=0
  BusA (R0)      = 0 (0x00000000)
  BusB (R0)      = 0 (0x00000000)
  BusP (R0)      = 0 (0x00000000)
  Predicate       = 1
  Control Sigs    = RegW=1 ALUSrc=0 MemRd=0 MemWr=0 WB=00 RegSel=0 DestReg=R0

--- FORWARDING ---
  FWA=00 FWB=00 FWP=00 (0=none, 1=EX, 2=MEM, 3=WB)
  BusA_FW         = 0 (0x00000000)
  BusB_FW         = 0 (0x00000000)
  BusP_FW         = 0 (0x00000000)

--- EXECUTE STAGE ---
  Predicate_E     = 1
  BUSA_E          = 0 (0x00000000)
  BUSB_E          = 0 (0x00000000)
  Imm_E           = 0 (0x00000000)
  ALUSrcMux       = 0 (0x00000000)
  ALUOP_E         = 000
  ALU_Result      = 0 (0x00000000)
  Zero Flag       = 1
  Dest Reg        = R0
  RegWrite_E      = 1

--- MEMORY STAGE ---
  Predicate_mem   = 1
  Address         = 0 (0x00000000)
  WriteData       = 0 (0x00000000)
  ReadData        = 100 (0x00000064)
  Dest Reg        = R0
  MemRd           = 0
  MemWr           = 0 (actual=0)
  RegWrite_mem    = 1

--- WRITE-BACK STAGE ---
  Predicate_WB    = 1
  WriteData       = 0 (0x00000000)
  Dest Reg        = R0
  RegWrite_WB     = 1
  >>> WRITING R0 = 0 <<<

--- REGISTER CHANGES ---
  R29: 0 -> 4294967265
  R30: 36 -> 37
```

Figure 3-36:cycle #39.

Test summary:

```
================================================================================
                      VERIFICATION COMPLETE
================================================================================

TEST #1:                     ADD R7 (basic)
  R7: Expected=250, Actual=250 -> PASS

TEST #2:                  SUB R8 (RAW: EX)
  R8: Expected=150, Actual=150 -> PASS

TEST #3:                  OR R9 (RAW: MEM)
  R9: Expected=150, Actual=150 -> PASS

TEST #4:                AND R10 (RAW: WB)
  R10: Expected=146, Actual=146 -> PASS

TEST #5:                           ADDI R11
  R11: Expected=150, Actual=150 -> PASS

TEST #6:                            ORI R12
  R12: Expected=15, Actual=15 -> PASS

TEST #7:                           ANDI R13
  R13: Expected=240, Actual=240 -> PASS

TEST #8:                           NORI R14
  R14: Expected=4294967285, Actual=4294967285 -> PASS

TEST #9:                             LW R15
  R15: Expected=100, Actual=100 -> PASS

TEST #10:         ADD R16 (Load-Use Stall)
  R16: Expected=250, Actual=250 -> PASS

TEST #11:                            SUB R17
  R17: Expected=150, Actual=150 -> PASS

TEST #12:                  ADD R18 (Pred=1)
  R18: Expected=250, Actual=250 -> PASS

TEST #13:                  ADD R19 (Pred=0)
  R19: Expected=0, Actual=0 -> PASS

TEST #14:             ADD R22 (after Jump)
  R22: Expected=30, Actual=30 -> PASS

TEST #15:         ADD R23 (Jump not taken)
  R23: Expected=160, Actual=160 -> PASS

TEST #16:             R31 (Return address)
  R31: Expected=24, Actual=24 -> PASS

TEST #17:            ADD R24 (in function)
  R24: Expected=110, Actual=110 -> PASS

TEST #18:           ADD R25 (after return)
  R25: Expected=120, Actual=120 -> PASS

TEST #19:                             LW R26
  R26: Expected=100, Actual=100 -> PASS

TEST #20:                             LW R27
  R27: Expected=170, Actual=170 -> PASS

TEST #21:        ADD R28 (Multiple Stalls)
  R28: Expected=270, Actual=270 -> PASS

TEST #22:                           NOR R29
  R29: Expected=4294967265, Actual=4294967265 -> PASS

================================================================================
  Total Tests: 22 | Passed: 22 | Failed: 0
  STATUS: ALL TESTS PASSED
================================================================================
```

Figure 3-37:Test summary.

By looking at fig (2-37), we can see that all thar instruction (I1 to I29) passed correctly.

Now let’s see a wave from fetching an instruction:

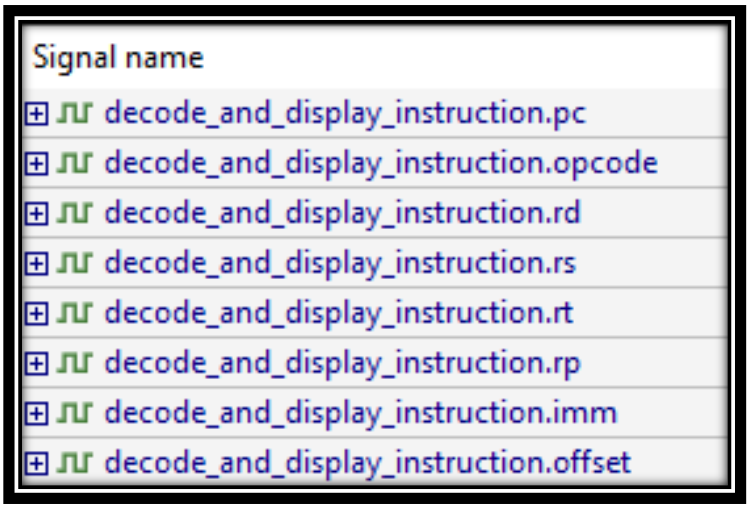


Figure 3-38: signal name at fetch stage.

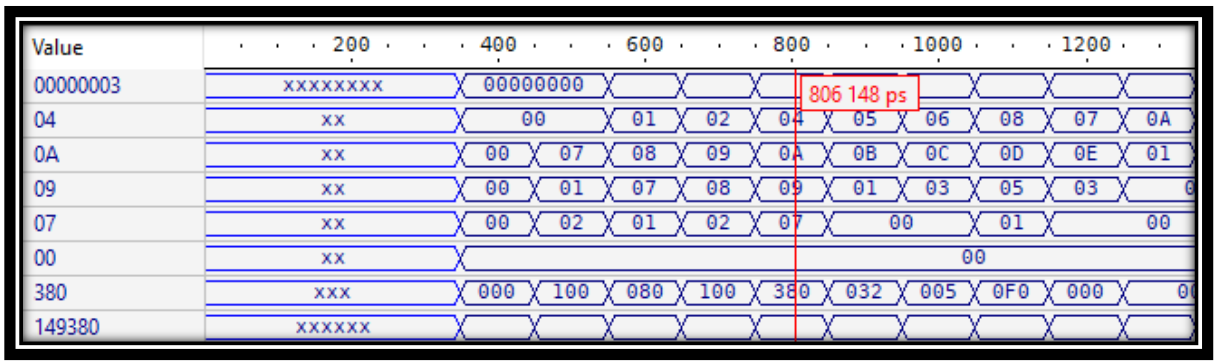


Figure 3-39: wave form test I4

In this wave form we can see the instructions that were fetched is R10,R9,R7,R0.

4. RTL Pipeline Design Description

4.1. R-type

Instruction format for R-type:

INSTRUCTION Rd, Rs, Rt, Rp

Bit Layout: Opcode[31:27] | Rp[26:22] | Rd[21:17] | Rs[16:12] | Rt[11:7] | Unused[6:0]

- General Pipeline Stages for All R-Type Instructions:
- **IF:**
  - IR ← IMEM[PC]
  - PC ← PC + 1
- **ID:**
  - A ← Reg[Rs]
  - B ← Reg[Rt]
  - Pred ← Reg[Rp]
  - Control signals set based on Opcode
  - Predicate_Valid ← (Pred ≠ 0)
- **EX:**
  - **ADD (Opcode: 0):** ALUOut ← A + B
  - **SUB (Opcode: 1):** ALUOut ← A - B
  - **OR (Opcode: 2):** ALUOut ← A | B
  - **NOR (Opcode: 3):** ALUOut ← ~(A | B)
  - **AND (Opcode: 4):** ALUOut ← A & B
  - **JR (Opcode: 13):**
    - Jump_Target ← A
    - if (Predicate_Valid) → PC ← Jump_Target (pipeline flush)
- **MEM:**
  - Pass-through (no memory operation for R-Type)
  - ALUOut propagates to WB stage
- **WB:**
  - if (Predicate_Valid AND Opcode ≠ 13) → Reg[Rd] ← ALUOut
  - else → No write
  - **Note:** JR does not write to any register

4.2. I-type

Instruction format for I-type:

INSTRUCTION Rd, Rs, Imm, Rp

Bit Layout: Opcode [31:27] | Rp [26:22] | Rd [21:17] | Rs [16:12] | Imm [11:0]

- **Arithmetic/Logical I-Type Instructions:**
- **General Pipeline for ADDI, ORI, NORI, ANDI (Opcodes: 5, 6, 7, 8)**
- **IF:**
  - **IR ← IMEM[PC]**
  - **PC ← PC + 1**
- **ID:**
  - **A ← Reg[Rs]**
  - **Pred ← Reg[Rp]**
  - **Imm_ext ←**
    - **Sign-extend(Imm[11:0]) for ADDI**
    - **Zero-extend(Imm[11:0]) for ORI, NORI, ANDI**
  - **Predicate_Valid ← (Pred ≠ 0)**
- **EX:**
  - **ADDI (Opcode: 5): ALUOut ← A + Imm_ext**
  - **ORI (Opcode: 6): ALUOut ← A | Imm_ext**
  - **NORI (Opcode: 7): ALUOut ← ~(A | Imm_ext)**
  - **ANDI (Opcode: 8): ALUOut ← A & Imm_ext**
- **MEM:**
  - **Pass-through**
- **WB:**
  - **if (Predicate_Valid) → Reg[Rd] ← ALUOut**
  - **else → No write**
- **Load/Store Instructions:**
- **LW Rd, Imm(Rs), Rp (Opcode: 9)**
- **IF:**
  - **IR ← IMEM[PC]**
  - **PC ← PC + 1**

- **ID:**
  - **A ← Reg[Rs]**
  - **Pred ← Reg[Rp]**
  - **Imm_ext ← sign-extend (Imm[11:0])**
  - **Predicate_Valid ← (Pred ≠ 0)**
- **EX:**
  - **ALUOut ← A + Imm_ext (compute memory address)**
- **MEM:**
  - **if (Predicate_Valid) → MDR ← DMEM[ALUOut]**
  - **else → MDR ← 0 (or don't care)**
- **WB:**
  - **if (Predicate_Valid) → Reg[Rd] ← MDR**
  - **else → No write**
- **SW Rd, Imm(Rs), Rp (Opcode: 10)**
- **IF:**
  - **IR ← IMEM[PC]**
  - **PC ← PC + 1**
- **ID:**
  - **A ← Reg[Rs]**
  - **B ← Reg[Rd]**
  - **Pred ← Reg[Rp]**
  - **Imm_ext ← sign-extend(Imm[11:0])**
  - **Predicate_Valid ← (Pred ≠ 0)**
- **EX:**
  - **ALUOut ← A + Imm_ext (compute memory address)**
- **MEM:**
  - **if (Predicate_Valid) → DMEM[ALUOut] ← B**
  - **else → No memory write**
- **WB:**
  - **No write-back**

4.3. J-type

Instruction format for J-type:

INSTRUCTION Label, Rp

Bit Layout: Opcode[31:27] | Rp[26:22] | Offset[21:0]

J Label, Rp (Opcode: 11)

- **IF:**
  - **IR ← IMEM[PC]**
  - **PC ← PC + 1**
- **ID:**
  - **Pred ← Reg[Rp]**
  - **Offset_ext ← sign-extend(Offset[21:0])**
  - **Jump_Target ← PC + Offset_ext**
  - **Predicate_Valid ← (Pred ≠ 0)**
  - **if (Predicate_Valid) → PC ← Jump_Target**
  - **else → PC remains unchanged**
- **EX/MEM/WB:**
  - **Flushed if jump taken**

CALL Label, Rp (Opcode: 12)

- **IF:**
  - **IR ← IMEM[PC]**
  - **PC ← PC + 1**
- **ID:**
  - **Pred ← Reg[Rp]**
  - **Offset_ext ← sign-extend(Offset[21:0])**
  - **Jump_Target ← PC + Offset_ext**
  - **Predicate_Valid ← (Pred ≠ 0)**
  - **if (Predicate_Valid):**
    - **Reg[R31] ← PC (store return address)**
    - **PC ← Jump_Target**
  - **else → PC remains unchanged, no R31 update**
- **EX/MEM/WB:**
  - **Flushed if call taken**

5. Conclusion

This project successfully designed and implemented a 32-bit pipelined RISC processor with predicated execution using Verilog HDL. The processor features a five-stage pipeline (Fetch, Decode, Execute, Memory, Write-Back) supporting 14 instructions across R-Type, I-Type, and J-Type formats. Comprehensive hazard detection and resolution mechanisms were integrated, including three-level data forwarding, automatic stall insertion for load-use dependencies, and pipeline flushing for control hazards with one-cycle branch penalties. The predication mechanism through the Rp register enables conditional execution without branching overhead, while special registers (R0, R30, R31) were successfully integrated with appropriate hardware protection. Extensive simulation testing validated correct execution of all instructions, hazard scenarios, and pipeline operations through waveform analysis. The modular design approach with centralized control and word-addressable memory effectively balanced complexity and performance, demonstrating fundamental computer architecture principles with potential for future enhancements including branch prediction, cache integration, and exception handling.

6. Future Improvements

While the processor meets all functional requirements, future work could explore:

- **Branch prediction mechanisms** (Branch Target Buffer, two-bit counters) to reduce control hazard penalties and improve pipeline efficiency.
- **Cache memory integration** for both instruction and data memory to decrease access latency and enhance overall throughput.
- **Exception and interrupt handling** to support runtime error detection (illegal instructions, arithmetic overflow) and enable operating system-level functionality.
- **Extended instruction set** including multiplication/division units, floating-point operations, and performance monitoring counters for benchmarking.