Cairo University
Faculty of Engineering
Credit Hours System

Computer Architecture Project

Team 1

Group Members

- Aya Mohamed AbdelTawab Alsayed - 1210002
- Doha AbdelFattah Albeltagy- 1210146
- Mario Emad Saleh Fouad- 1210158
- Mohamed Alsayed Shaaban- 1210025

Computer Architecture Project Phase 1

Instruction format

Since there are 26 total instructions therefore an opcode of 5 bits would be sufficient to resemble each of these instructions.
Instructions would occupy one memory location (32 bits).
We use 5 bits for the Opcode (Bit 31-27), allowing for up to 32 distinct instructions.

Instructions Bit Details:

Instruction	**OpCode** bits[31:27]	bits[26:24]	bits[23:21]	bits[20:18]	bits[17:16]	bits[15:0]	Instruction type
NOP	00000						R-Type
HLT	00001						R-Type
SETC	00010						R-Type
NOT Rdst	00011			Rdst			R-Type
INC Rdst	00100			Rdst			R-Type
OUT Rdst	00101			Rdst			R-Type
IN Rdst	00110			Rdst			R-Type
MOV Rsrc, Rdst	00111	Rsrc		Rdst			R-Type
SWAP Rsrc, Rdst	01000	Rsrc		Rdst			R-Type
ADD Rdst, Rsrc1, Rsrc2	01001	Rsrc1	Rsrc2	Rdst			R-Type
SUB Rdst, Rsrc1, Rsrc2	01010	Rsrc1	Rsrc2	Rdst			R-Type
AND Rdst, Rsrc1, Rsrc2	01011	Rsrc1	Rsrc2	Rdst			R-Type
IADD Rdst, Rsrc	01100	Rsrc		Rdst		Imm	I-Type

,Imm							
PUSH Rdst	01101			Rdst			R-Type
POP Rdst	01110			Rdst			R-Type
LDM Rdst, Imm	01111			Rdst		Imm	I-Type
LDD Rdst, offset(Rsrc)	10000	Rsrc		Rdst		offset	I-Type
STD Rsrc1, offset(Rsrc2)	10001	Rsrc1	Rsrc2			offset	I-type
JZ Imm	10010					Imm	J-Type
JN Imm	10011					Imm	J-Type
JC Imm	10100					Imm	J-Type
JMP Imm	10101				11	Imm	J-Type
CALL Imm	10110				11	Imm	J-Type
RET	10111				10		R-Type
INT index	11000				00	index(0 or 1)--> bit 0	R-Type
RTI	11001				10		R-Type

Pipeline Registers' details:

1) IF/ID register (66 bits)
   Inputs: Instruction(32 bits), PC(32 bits)
   Control Signals:
   - EN (Enable / Write Enable) → 1 bit:
     - Function: Controls Stalling.
     - Active: Low (0).
     - Logic:
       - Normal: 0 (Update with new instruction).
       - Hazard (Load-Use): 1 (Freeze). The register keeps holding the *old* instruction. This forces the Decode stage to process the same instruction again (stall), while the Fetch stage also pauses.
   - CLR (Synchronous Clear / Flush) → 1 bit:
     - Function: Controls Flushing.
     - Active: high (1).
     - Logic:
       - Branch Taken / Interrupt: If the processor realizes the previous fetch was wrong (e.g., we jumped), CLR becomes 1.
       - Result: The output becomes 00...00 (which is the NOP opcode). The wrong instruction is effectively deleted from the pipeline.

2) ID/Ex register (158 bits)
   Inputs: src1(3 bits), src2(3bits), dest(3bits) ,immediate value after sign extension (32 bits), PC (32 bits), data1(R[src1]) (32 bits) & data2(R[src2]) (32 bits).
   Control Signals:
   - Control (WB) (3 bits)
     - Reg_Write (1 bit) Enables writing to the Register File.
     - Mem_to_Reg (2 bits) Selects write-back data (00=ALU, 01=Mem, 10=inPort).
   - Control (M) (6 bits)
     - Mem_Read (1 bit) Enables reading from Data Memory (LDD).
     - Mem_Write (1 bit) Enables writing to Data Memory (STD).
     - Port_Read (1 bit) Enables reading from Input Port (IN).
     - Port_Write (1 bit) Enables writing to Output Port (OUT).
     - Stack_Op (2 bits) Operation for Stack Pointer (00=None, 01=Push, 10=Pop).
   - Control (EX) (11)
     - ALU_Op (4 bits) Selector for ALU operation (Add, Sub, And, etc.).
     - ALU_Src (1 bit) Selects Operand B source (0=Reg, 1=Immediate).
     - Branch_Type (3 bits) Indicates branch type (JZ, JMP, CALL, RET, etc.).
     - RET / RTI (2 bits) Signals return from Call/Interrupt.
     - Interrupt (1 bit) Signals interrupt logic.
   - CLR (Synchronous Clear / Flush) (1)
     - 1 bit→**Function**: Inserts a "Bubble" (NOP) into the pipeline by resetting all control signals to 0.

       Trigger Sources:

   1. Hazard Detection Unit: Activates when a Load-Use hazard is detected (the instruction in Decode must wait, so a bubble is sent to Execute).
   2. Branch Control Unit: Activates when a control hazard (branch taken) is resolved in the Execute stage, necessitating the removal of the erroneously fetched instruction in the Decode stage.

3) Ex/Mem register (117 bits)
   Inputs: ALU result(32 bits), WriteData or Imm(32 bits), Rdst_address(3 bits), PC(32 bits), Flags(3 bits→ Z,N and C)
   Control Signals:
   - Control (WB) (3 bits)
     - Reg_Write (1 bit) Enables writing to the Register File.
     - Mem_to_Reg (2 bits) Selects write-back data (00=ALU, 01=Mem, 10=inPort).
   - Control (M) (9 bits)
     - Mem_Read (1 bit) Enables reading from Data Memory (LDD).
     - Mem_Write (1 bit) Enables writing to Data Memory (STD).
     - Port_Read (1 bit) Enables reading from Input Port (IN).
     - Port_Write (1 bit) Enables writing to Output Port (OUT).
     - Stack_Op (2 bits) Operation for Stack Pointer (00=None, 01=Push, 10=Pop).
   - Control (Sys) (3)
     - RET / RTI (2 bits) Signals return from Call/Interrupt.
     - Interrupt (1 bit) Signals interrupt logic.

4) Mem/WB register (102 bits)
   Inputs: MemOut (32 bits) , ALU result(32 bits), InPort (32 bits), Rdst_address (3 bits)
   Control Signals:
   - Control (WB) (3 bits)
     - Reg_Write (1 bit) Enables writing to the Register File.
     - Mem_to_Reg (2 bits) Selects write-back data (00=ALU, 01=Mem, 10=inPort).

Data Hazards

We have two main sources of hazards:

1. **Read after write**
   - **Problem:** Occurs when an instruction depends on the result of a previous instruction that has not yet been committed to the Register File (Write-Back stage).
   - **Solution:** Forwarding Unit→ if two operations are in the pipeline and one of them wants to read the value another would write to the memory just forward the value.
2. **Load use**
   - **Problem:** Occurs when an instruction tries to read a register that is currently being loaded from memory by the immediately preceding instruction.
   - **Solution**: Forwarding Unit→ Stall once (NOP) until data is read from memory. Forward ReadData from MEM/WB to EXEC stage

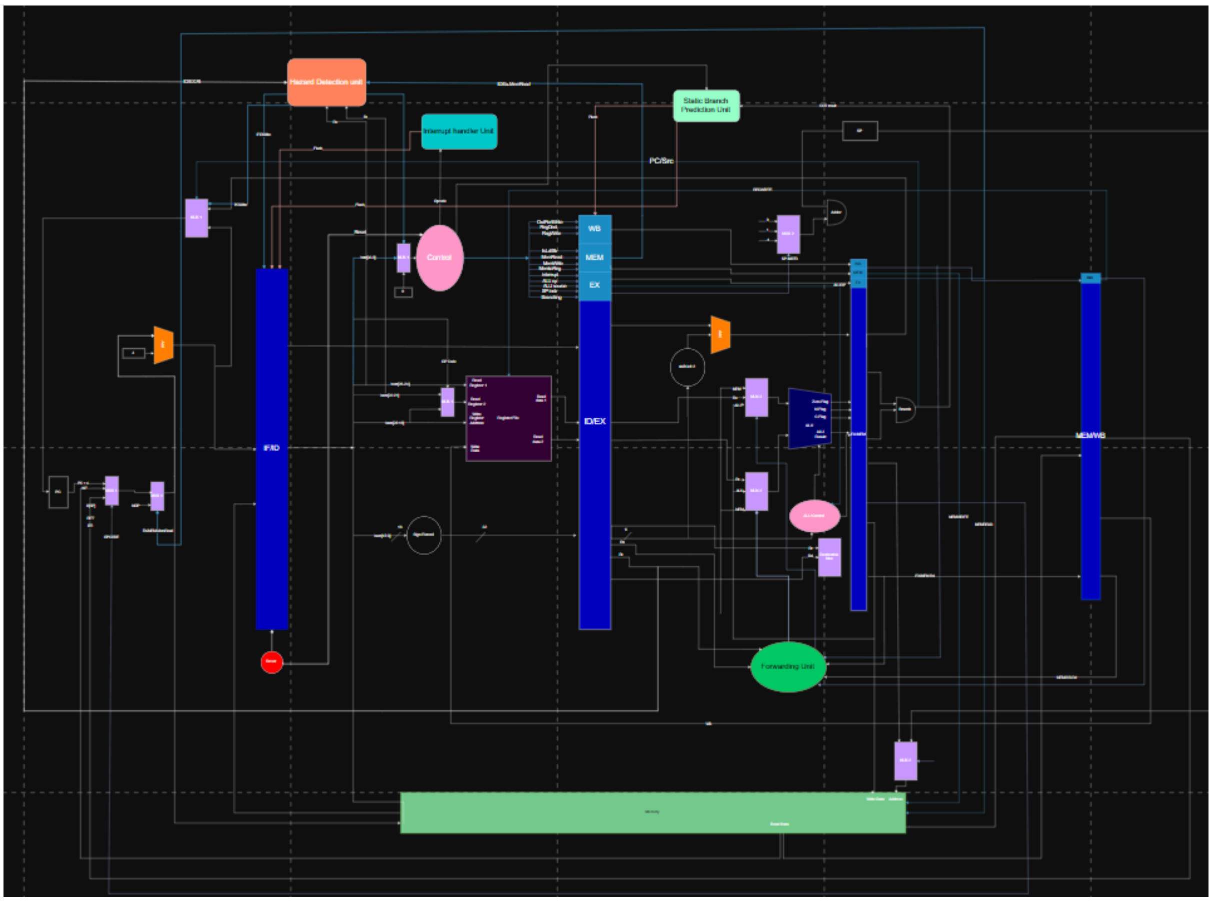
Structural Hazard

- **Problem:** When an instruction is fetching an instruction from the memory and another instruction is loading from memory.
- **Solution:** Add a MUX at the program counter so that when the EX/MEM.MemRead signal is 1, we insert a NOP so that we stall the Fetch for one cycle until the other instruction loads from memory.

Control Hazards

1. **Branching**
   - **Problem:** Occurs because the branch condition (Taken vs. Not Taken) and the Target Address are not calculated until the **Execute Stage**, but the processor must fetch instructions in the meantime.
   - **Solution**: Static Branch Detection Unit→ Predict Not Taken.
   - **Prediction:** The Fetch stage implicitly predicts "Not Taken" by always incrementing the PC (PC + 1) and fetching sequential instructions.
   - **Correction:** If the Branch Control Unit resolves the branch as **Taken**:
     1. **Flush:** It asserts the CLR signal to the **IF/ID Register**, turning the wrongly fetched instruction into a NOP.
     2. **Update:** It updates the PC Mux to load the calculated **Branch Target Address** instead of PC + 1.

Schematic Diagram:



Link: [Schematic diagram]()