EE309 : Microprocessors
Pipelined CPU Implementation

Gayatri Bhat - 23B1217
Pragyan Paramita Behera - 23B1278

May 3, 2025

Pipelined Datapath

A pipelined datapath divides instruction execution into overlapping stages to maximize throughput. Each stage operates concurrently on different instructions.

Pipeline Stages

Instruction Fetch (IF)

This stage retrieves the next instruction from memory using the **Program Counter (PC)**, which holds the address of the current instruction. After fetching, the PC is updated to point to the next instruction. The fetched instruction is temporarily stored in a pipeline register for the next stage.

Instruction Decode (ID)

The control unit decodes the instruction to determine the operation (opcode) and operands. This involves:

- Identifying registers containing input values.
- Extending immediate values (constants) to match processor word size.
- Preparing branch/jump targets by calculating offsets.

Decoded data is passed to the next stage via another pipeline register.

Execute (EX)

The Arithmetic Logic Unit (ALU) performs computations, such as:

- Arithmetic operations (e.g., addition, subtraction).
- Logical comparisons (e.g., AND, OR).
- Memory address calculations for load/store instructions.

Results are stored in a pipeline register for subsequent stages.

Memory Access (MEM)

This stage interacts with data memory:

- **Load instructions**: Retrieve data from a computed address.
- **Store instructions**: Write data to a computed address.

Results (either memory data or ALU outputs) are stored in a pipeline register.

Write Back (WB)

The final stage writes results back to the register file. Data sources include:

- ALU outputs from arithmetic or logic instructions.
- Data loaded from memory.

This completes the instruction's journey through the pipeline.

Key Features

- Uses **pipeline registers** (e.g., `IF/ID`, `ID/EX`) to isolate stages and forward intermediate data[1][4].
- Achieves near-1 Cycle Per Instruction (CPI) by overlapping stages, but requires hazard resolution (data/control)[1][4].
- Clock cycle time determined by the slowest stage[1][6].

Multicycle Datapath

A multicycle datapath executes instructions sequentially across multiple clock cycles, reusing hardware components for different stages.

Multicycle Stages

Instruction Fetch (IF)

- Fetches the instruction from memory using the **Program Counter (PC)**.
- Stores the fetched instruction in the **Instruction Register (IR)**.
- Prepares the next PC value (PC + 4) for sequential execution.

Instruction Decode (ID)

- Decodes the opcode to determine operation type.
- Reads source register values into temporary registers A and B.
- Sign-extends immediate values for arithmetic/logic operations.
- Computes potential branch/jump targets.

Execute (EX)

- Performs ALU operations using data from A, B, or immediates.
- Calculates memory addresses for load/store instructions.
- Evaluates branch conditions and computes target addresses.
- Stores ALU results in `ALUout` register.

Memory Access (MEM)

- Completes memory operations:
  - **Load**: Reads data into **Memory Data Register (MDR)**.
  - **Store**: Writes data from B to memory.
- For non-memory instructions, bypasses this stage.

Write Back (WB)

- Writes results to destination register:
  - ALU output (from `ALUout`) for arithmetic/logic operations.
  - Memory data (from MDR) for load instructions.
- Updates PC with jump/branch targets if required.

Key Features

- Uses **temporary registers** (e.g., IR, MDR, A/B) to hold intermediate values between cycles[2][7].
- CPI ranges from 4-5 cycles per instruction, with shorter clock cycles than single-cycle designs[2][7].
- No overlapping of instructions-each completes fully before the next begins[2][7].

Differences

Aspect	**Pipelined Datapath**	**Multicycle Datapath**
CPI	Near 1 (ideal)	4-5[1][2]
Overlap	Stages overlap across instructions	No overlap[1][2]
Registers	Pipeline registers (e.g., `IF/ID`)	Temporary registers (e.g., IR, MDR)[1][2]
Hazards	Requires handling data/control hazards	No hazards (sequential execution)[1][4]
Clock Cycle	Determined by slowest stage	Shorter, optimized for critical paths[1][2]

Structural Comparison

- Pipelined datapaths use dedicated hardware per stage (e.g., separate ALU for each stage), while multicycle designs reuse components like a single ALU[1][2].
- Control in pipelined designs is distributed across stages, whereas multicycle uses a centralized finite-state machine[2][7].

Supported Instructions

- **LOAD (0000 Ra):** Load from front of DMEM to register A
- **STORE (0001 Ra):** Store from register A to back of DMEM
- **ADD (0010 Ra Rb Rc):** A $\leftarrow$ B + C
- **SUB (0011 Ra Rb Rc):** A $\leftarrow$ B - C

- **MUL (0100 Ra Rb Rc):** A ← B × C (16-bit result)
- **ADDI (0101 Ra Rb Imm6):** A ← B + Imm
- **SLL (0110 Ra Rb Rc):** $A \leftarrow B \ll \mathrm{C}[3:0]$
- **JRI (0111 Ra Imm9):** PC ← A + 2×Imm

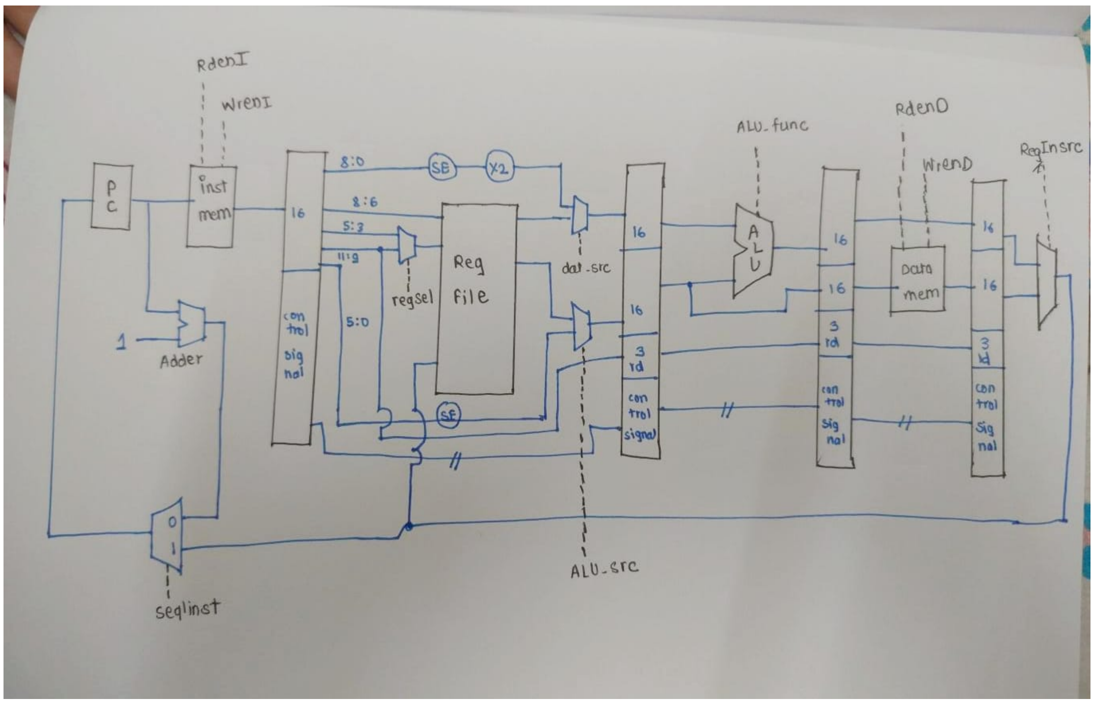


Figure 1: Datapath

Component Description

1. Fetch Stage (stage1)

- Reads instructions from instruction memory (IMEM).
- Updates the Program Counter (PC).
- Outputs: PC, `IR` (Instruction Register).

2. Decode Stage (stage2)

- Decodes the instruction.
- Selects operands from the register file.
- Outputs ALU operands `x`, `y`, and the destination register `dest0`.

3. Execute Stage (stage3)

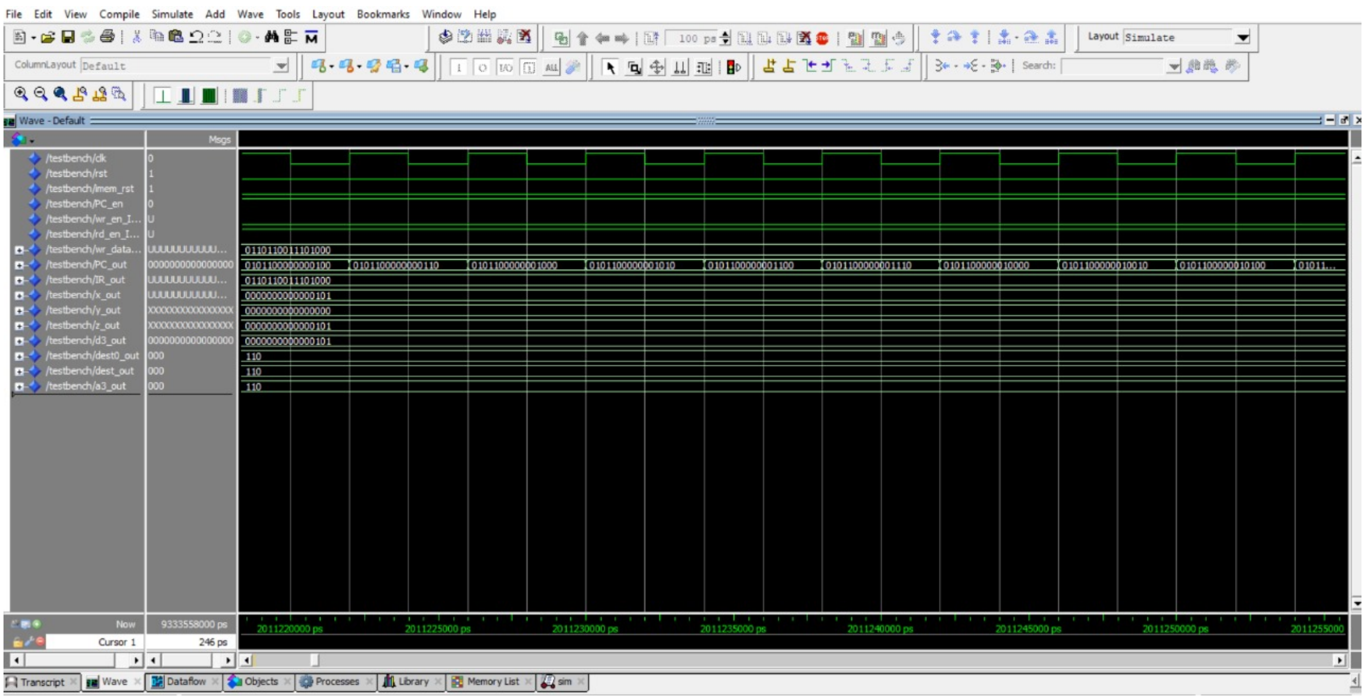
- Performs ALU operations based on the control signals.
- Inputs: `x`, `y`, control signals like ALU`Func`, `zSel`, etc.
- Outputs: result `z` and destination register `dest`.

4. Writeback Stage (stage5)

- Writes back the result `z` to the register file.
- Also outputs the write data `d3` and destination address `a3`.

5. Control Unit (control)

- Takes the opcode (`op`) from the instruction and generates all control signals:
  - `xSel`, `ySel`, `WrEnD`, `RdEnD`, `zSel`, `RegWrEn`



```
                        Msgs
/testbench/clk          0
/testbench/rst          1
/testbench/imem_rst     1
/testbench/PC_en        0
/testbench/wr_en_I...   U
/testbench/rd_en_I...   U
/testbench/wr_data...   UUUUUUUUUUU...     0110110011101000
/testbench/PC_out       0000000000000000   0101100000000100 |0101100000000110 |0101100000001000 |0101100000001010 |0101100000001100 |0101100000001110 |0101100000010000 |0101100000010010 |0101100000010100 |01011...
/testbench/IR_out       UUUUUUUUUUU...     0110110011101000
/testbench/x_out        UUUUUUUUUUU...     0000000000000101
/testbench/y_out        XXXXXXXXXXXXXXXX   0000000000000000
/testbench/z_out        XXXXXXXXXXXXXXXX   0000000000000101
/testbench/d3_out       0000000000000000   0000000000000101
/testbench/dest0_out    000                110
/testbench/dest_out     000                110
/testbench/a3_out       000                110

Now       9333558000 ps    2011220000 ps   2011225000 ps   2011230000 ps   2011235000 ps   2011240000 ps   2011245000 ps   2011250000 ps   2011255000
Cursor 1  246 ps
```

Figure 2: ModelSim Waveform

Thank You