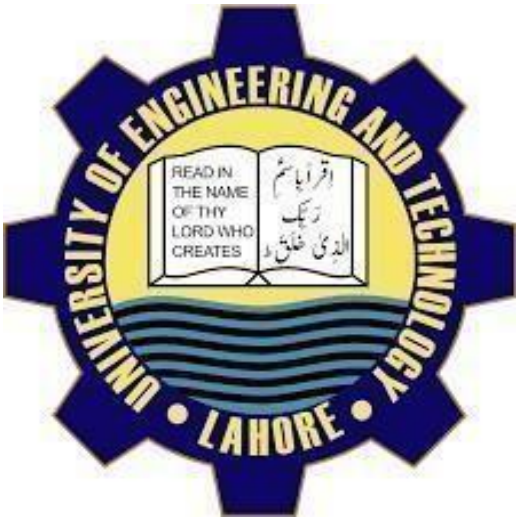
Computer Architecture

Final Report



Title:

3-stage Pipeline

Submitted to:

Sir. Afeed Obaid

Submitted By:

Hira Firdous [2020-CE-47]

Department of Computer Engineering

University of Engineering and Technology, Lahore

3-Stage Pipeline:

A 3-stage pipeline generally refers to a computing or processing architecture that divides the execution of instructions into three main stages. This approach is common in the design of processors and digital systems to improve performance and efficiency. The three stages typically include:

1. **Fetch Stage (IF - Instruction Fetch):**
    - Fetching the next instruction from memory.
    - Incrementing the program counter to point to the next instruction.
    - Loading the instruction into a register.
2. **Decode and Execute Stage (ID/EX - Instruction Decode/Execution):**
    - Decoding the instruction to determine the operation to be performed and the operands involved.
    - Fetching any required data from registers or memory.
    - Executing the operation (arithmetical or logical) on the data.
3. **Write-back Stage (MEM/WB - Memory/Write-back):**
    - If the operation involves memory access, this stage performs the read or write operation in memory.
    - Writing the result back to the appropriate register or memory location.

The concept of a pipeline is to overlap the execution of instructions, so while one instruction is in the execute stage, the next instruction can be in the decode stage, and the one after that can be in the fetch stage. This overlap helps in improving the throughput and overall efficiency of the processor.

Difference in 3 and 5 stage pipelines:

The complexity of a processor pipeline is not solely determined by the number of stages. The design complexity depends on various factors, including the specific tasks each stage performs, the interconnection between stages, and how hazards are handled. In general, more stages in a pipeline can provide greater opportunities for parallelism and optimization but also introduce challenges in terms of managing dependencies and hazards.

A 5-stage pipeline is often considered more complex than a 3-stage pipeline because it allows for more fine-grained control over the execution of instructions. The additional stages can include separate stages for instruction fetch, instruction decode, execute, memory access, and write-back. This finer division allows for more precise control over the various tasks associated with instruction execution.

However, with increased complexity comes the need for better handling of pipeline hazards such as data hazards, control hazards, and structural hazards. Advanced techniques like out-of-order execution and speculative execution may also be implemented to further enhance performance, but they add complexity to the design.

In summary, while a 5-stage pipeline is generally considered more complex than a 3-stage pipeline, the actual complexity and performance of a processor depend on various design choices and optimizations made by the architects.

Explanation of code:

Following is the detailed explanation of my code.

Inputs and Outputs:

- **Inputs:**

- **clk**: Clock signal.
- **rst**: Reset signal.

Internal Signals and Wires:

- **Control Signals:**
  - **rf_en_DE**, **rf_en_DM**: Register file enable signals for the Decode (DE) and Data Memory (DM) stages.
  - **sel_b**: Select signal for choosing the second operand in the ALU.
  - **stall_IF**: Stall signals for the Instruction Fetch (IF) stage.
  - **flush_DE**: Flush signal for the Decode (DE) stage.
- **Program Counter (PC) and Branching Signals:**
  - **pc_out_IF**, **pc_out_DE**: Program counter outputs for IF and DE stages.
  - **new_pc**: New program counter value.
  - **br_taken_IF**, **br_taken_DE**: Branch taken signals for IF and DE stages.
- **Control and Status Registers (CSR):**
  - **csr_epc_IF**, **csr_epc_WB**: Exception Program Counter for IF and Writeback (WB) stages.
  - **csr_pc**: Control and Status Register (CSR) holding the program counter.
  - **epc**: Exception Program Counter.
- **Instruction and Immediate Signals:**
  - **inst_IF**, **inst_DE**: Instruction signals for IF and DE stages.
  - **opcode**, **funct3**, **funct7**: Opcode and function fields of the instruction.
  - **imm**, **imm_val**: Immediate values.
- **Register and Operand Signals:**
  - **rd**, **rs1**, **rs2**: Register destination and source signals.
  - **rdata1**, **rdata2**: Register data values.
  - **opr_a**, **opr_b**: ALU operands.
  - **opr_res_IF**, **opr_res_DE**, **opr_res_DM**: ALU operation results.
- **Memory and Writeback Signals:**
  - **wdata_DE**, **wdata_MW**: Write data signals for DE and Memory Writeback (MW) stages.
  - **wb_sel_DE**, **wb_sel_DM**: Writeback select signals for DE and DM stages.

Modules and Components:

- **Mux Units:**
  - **mux_2x1**: 2-to-1 multiplexer used for PC selection and CSR PC selection.
  - **mux_3x1**: 3-to-1 multiplexer used for Writeback MUX.
- **Processor Components:**
  - **pc**: Program Counter module.
  - **inst_mem**: Instruction Memory module.
  - **inst_dec**: Instruction Decoder module.
  - **reg_file**: Register File module.
  - **imm_gen**: Immediate Generator module.
  - **alu**: Arithmetic Logic Unit module.
  - **controller**: Controller module.
- **Data Memory and CSR:**
  - **data_mem**: Data Memory module.
  - **CRS**: Control and Status Register module.

- **Hazard Unit:**
    - **hazard_unit**: Hazard Unit module used for handling data and control hazards.

Pipeline Stages:

1. **Instruction Fetch (IF) Stage:**
    - Fetches instructions from memory.
    - Handles branching and updates the program counter.
2. **Decode (DE) and Execute Stage:**
    - Decodes instructions.
    - Performs ALU operations.
3. **Memory and Writeback (MW) Stage:**
    - Handles memory operations.
    - Writes back results to registers.

Hazard Handling:

- **Hazard Unit (hazard_unit):**
    - Handles data hazards (forwarding and stalling).
    - Handles control hazards (flushing).

Sample code and diagram:

I have executed a simple code of:

```
add x3, x4, x2
```