

TASK 3 : PIPELINE PROCESSOR DESIGN

BASIC INFORMATION

A 4-stage pipelined processor is designed to execute instructions in multiple stages, allowing different instructions to be processed simultaneously, thereby improving throughput and performance. The processor follows these four basic stages in its pipeline:

1. Instruction Fetch (IF):

- The processor fetches the instruction from memory.
- The instruction is stored in the instruction register (IR).

2. Instruction Decode (ID):

- The processor decodes the fetched instruction.
- The registers specified by the instruction are read, and control signals are generated.

3. Execution (EX):

- The processor performs the operation specified by the instruction (e.g., ADD, SUB).
- For arithmetic instructions, the ALU (Arithmetic Logic Unit) performs the operation.
- For load instructions, the memory address is calculated.

4. Memory Access (MEM) / Write-back (WB):

- **For Load instructions:** The processor accesses memory to load the data into a register.
- **For Arithmetic instructions:** The result is written back to the register file.

Key Instruction Types:

- **ADD** and **SUB**: Arithmetic operations performed by the ALU.
- **AND**: Logical operation performed by the ALU.
- **LOAD**: Memory access instruction to load data into registers.

PYHTON CODE:

```
instructions = ["ADD R1, R2, R3", "SUB R4, R5, R6", "LOAD R7, 100(R8)"]
```

```
stages = ["IF", "ID", "EX", "MEM/WB"]
```

```
for i in range(len(instructions)):
```

```
    print(f"Instruction {i+1}: {instructions[i]}")
```


```
    for stage in stages:
```

```
        print(f"Cycle {i+1}: {stage}")
```

OUTPUT:]

Programiz
Python Online Compiler

Turn your knowledge
into new customers



Programiz PRO >

main.py

Run

Share

Settings

Fullscreen

JS

Go

Python

Java

C++

C

PHP

Perl

Ruby

Swift

Kotlin

Scala

Go

Python

Java

C++

C

PHP

Perl

Ruby

Swift

Kotlin

Scala

Go

```
1 # Simple Simulation of Pipeline Stages
2 # Assuming an ideal pipeline with no hazards
3 instructions = ["ADD R1, R2, R3", "SUB R4, R5, R6", "LOAD R7, 100
  (R8)"]
4 # Initialize pipeline stages
5 stages = ["IF", "ID", "EX", "MEM/WB"]
6 # Start simulating the pipeline
7 for i in range(len(instructions)):
8     print(f"Instruction {i+1}: {instructions[i]}")
9     for stage in stages:
10         print(f"Cycle {i+1}: {stage}")
11
```

Output

Clear

```
Instruction 1: ADD R1, R2, R3
Cycle 1: IF
Cycle 1: ID
Cycle 1: EX
Cycle 1: MEM/WB
Instruction 2: SUB R4, R5, R6
Cycle 2: IF
Cycle 2: ID
Cycle 2: EX
Cycle 2: MEM/WB
Instruction 3: LOAD R7, 100(R8)
Cycle 3: IF
Cycle 3: ID
Cycle 3: EX
Cycle 3: MEM/WB

=== Code Execution Successful ===
```