Github link

https://github.com/Hirasoh/Risc_V_Termproject

Single Cycle CPU:

In a single-cycle CPU, each instruction executes in a single clock cycle. This means that the CPU takes one clock cycle to fetch, decode, execute, and write back an instruction. All instructions, regardless of complexity, take the same amount of time to execute. While it simplifies the design, it might not be the most efficient in terms of performance due to the possibility of idle time during the cycle.

- Test Case for Single Cycle CPU:
- **Load immediate value into register**
- LI R1, 10 Load immediate value 10 into register R1
- **Add values from two registers**
- ADD R2, R1, R3 Add value in R1 to value in R3, store result in R2
- **Load values into registers**
- LI R4, 5 Load immediate value 5 into register R4
- LI R5, 8 Load immediate value 8 into register R5
- **Conditional branch based on values in registers**
- BLT R4, R5, label1 # Branch to label1 if R4 is less than R5
- # Instructions following the branch
- ...
- label1:Instructions after branch taken
- ...
- JUMP label2 # Jump to label2
- Instructions after jump
- ...
- label2:
- Instructions after jump destination
- ...

CSR (Control and Status Registers):

Control and Status Registers are specific registers in a computer architecture that store control and status information that control the operation of the CPU. These registers handle privileged

operations, such as controlling interrupts, managing processor modes, and maintaining certain system control information.

Test Case for CSR:

- Change the processor mode from user mode to supervisor mode.
- Trigger an interrupt and handle it by saving the current context into designated registers.
- Modify a control register to enable or disable a specific hardware feature.
- Set supervisor mode
- CSRRS R1, R0, 1 # Set bit 1 to 1 in R1 (assuming CSR register 1 controls processor mode)
- Trigger an interrupt
- EBREAK Generate a breakpoint exception
- Exception handler code to handle the interrupt
- # Modify a control register to enable a hardware feature (assuming CSR register 3 controls the feature)
- CSRRS R2, R0, 3 # Set bit 3 to 1 in R2 to enable the feature

3-Stage Pipelining:

In a 3-stage pipeline, the instruction execution is broken down into three stages: Instruction Fetch (IF), Instruction Decode (ID), and Execute (EX). Each stage operates concurrently for different instructions in the pipeline. This overlapping of instructions improves throughput by allowing multiple instructions to be processed simultaneously. However, if one stage takes longer than the others, it can lead to pipeline stalls.

Test Case for 3-Stage Pipelining:

- Execute a series of arithmetic operations (addition, subtraction) with different operands.
- Perform conditional branching where the branch outcome is both taken and not taken.
- Load data from memory into registers and perform operations on that data in subsequent instructions.

- Perform addition
- ADD R1, R2, R3 # Add value in R2 to value in R3, store result in R1
- Perform subtraction
- SUB R4, R5, R6 # Subtract value in R5 from value in R6, store result in R4
- # Conditional branch
- BLT R7, R8, label3 # Branch to label3 if R7 is less than R8
- # Instructions following the branch

- # ...
- label3:
- # Instructions after branch taken
- # ...
- # Load data from memory into register
- LOAD R9, 1000 # Load data from memory address 1000 into register R9
- # Perform operations on loaded data
- ADDI R9, R9, 5 # Add immediate value 5 to value in R9

Design

https://app.diagrams.net/#G1KAm9smdAkrW6bKbmb4oHVD4JPK5ge2GR