5 stage Pipeline design for a 32-bit MIPS

The goal of this project is to better understand the fundamentals of pipelining and the MIPS pipeline's implementation.

Description

5 stage Pipeline design for a 32-bit MIPS. The code is simulated using the EDAplayground platform.

Objective

The CPU supports the following fundamental assembly instructions: LDUR, STUR, ADD, SUB, ORR, AND, CBZ, B, and NOP. LDUR and STUR support instantaneous values when executing specific operations to the registers module.

The processor is pipelined, enabling the execution of many instructions at once. Between each stage of the processor's operation, buffers and caches are added to make this possible. Each instruction takes less time to execute thanks to this optimization, which also guards against atomic reads that happen in between register-base instructions. The hazard detection unit and the forwarding unit are two extra modules in this implementation. Hazards are potential issues with a processor's instruction pipeline. These risks may cause errors in the computation's output. The processor implements a danger detecting unit and a forwarding unit to deal with loading hazards. The pipeline is stopped by the hazard detecting unit, which also inserts a bubble (NOPS) until the following instruction is read. When an instruction that requires the use of the loaded register is immediately followed by an LDUR instruction, the hazard detecting unit is frequently employed. The group also put into place a forwarding unit that, when an instruction in the EX stage requires a computed value from a previous instruction, sends that computed value from the MEM or WB stage of the pipeline to the EX stage. The most recent computed value can be used because multiplexers in the EX stage pass values from the MEM and WB stages.

Supported Instructions set

The examples below demonstrate the functionality for each instruction using the following 'variables':
- r#: Register # in the CPU (From 0 to 31)
- RAM: Random Access Memory (RAM)
- PC: Program Counter

LDUR: Load RAM into Registers
Example 1: LDUR r3, [r9]
- Sudo - C code: r3 = RAM[r9]

Explanation: Put the value from memory address r9 into register r3 after retrieving it.
Example 2: LDUR r7, [r6, #1]
Sudo-C code: r7 = RAM[r6 + 1]
Explanation: Take the value from memory located at address r6 + immediate (1) and enter it in register r7.

STUR: Store Registers into RAM
Example 1: STUR r2, [r10]
Sudo-C code: RAM[r10] = r2
Explanation: Place the value of r2 at location r10 in memory.

Example 2: STUR r5, [r8, #1]
Sudo-C code: RAM[r8 + 1] = r5
Explanation: Memory address r8 + immediate should be used to store the value of r5 (1).

ADD: Add Registers
Example: ADD r6, r4, r3
Sudo-C code: r6 = r4 + r3
Explanation: Input the result into r6 after adding the values of r4 and r3.

SUB: Subtract Registers
Example: SUB r3, r2, r1
Sudo-C code: r3 = r2 – r1
Explanation: Add the result into r3 after subtracting the values of r2 and r1.
ORR: Bit-wise OR Registers
Example: ORR r8, r4, r5
Sudo-C code: r8 = r4 | r5
Explanation: Put the result into r8 after bitwise OR the values of r4 and r5.
AND: Bit-wise AND Registers
Example: AND r8, r6, r4
Sudo-C code: r8 = r6 & r2
Explanation: Put the results of bit-wise AND the values of r6 and r4 into r8.

CBZ: Conditional Jump (when the value in Register is zero)
Example: CBZ r5, #2
Sudo-C code: if (r5 == 0) { PC = 2 } else { PC++ }
Explanation: Jump to instruction 2 if the value of r5 is zero; else, carry on with PC++.

B: Unconditional (arbitrary) Jump
Example: B #2
Sudo-C: PC = 2
Explanation: Jump to instruction 2

NOP: No Operation
Explanation: a command that causes the processor to pause for one clock cycle.

Simulation

To run this project, you will need to have an account on EDAplayground. Once you have an account, you can follow these steps:

1. Log in to EDAplayground.
2. Click on the "New" button in the top left corner of the screen.
3. Choose "Verilog" as the language and give your project a name.
4. Copy the contents of the "design.v" file into the editor window.
5. Copy the contents of the "testbench.v" file into the editor window.
6. In tools and simulations choose “Icarus Verilog 0.9.7” to run
7. Check “Open EPWave” after run to see simulation waveforms.
8. Click the "Run" button to simulate the design.

Contributors

1. Sabrina Davidson
2. Bhavin Goswami
3. Harshal Patil
4. Harsh Patel
5. Damin Shah