The Chinese University of Hong Kong, Shenzhen

CSC 3050

Computer Architecture

CPU Report

Author
Jiakun Fan

Student ID
120090316

April 30, 2022

Contents

1 Purpose of the Project

This project aims to implement a 5-stage pipeline CPU to execute MIPS instructions.

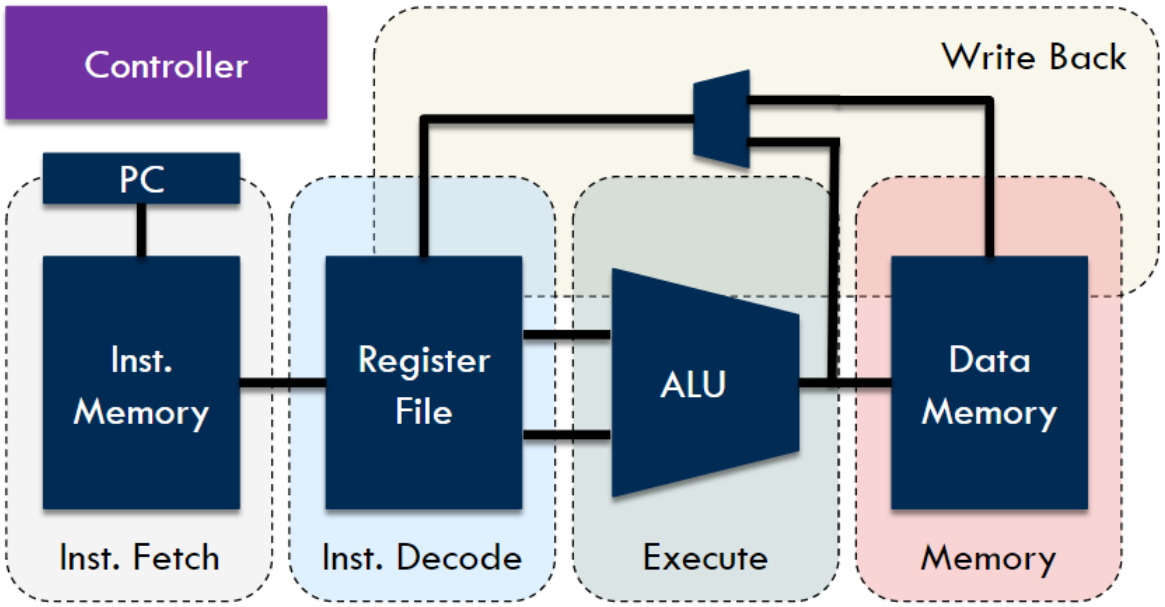

Figure 1: Five-stage Pipeline CPU

2 Design of the Project

2.1 High Level Logic

As shown in Figure 2, we first design the structure of CPU to implement five-stage pipeline and avoid hazard. Then we divide components of CPU into modules. We implement and test independent modules. After that, we assemble CPU with modules implemented before.

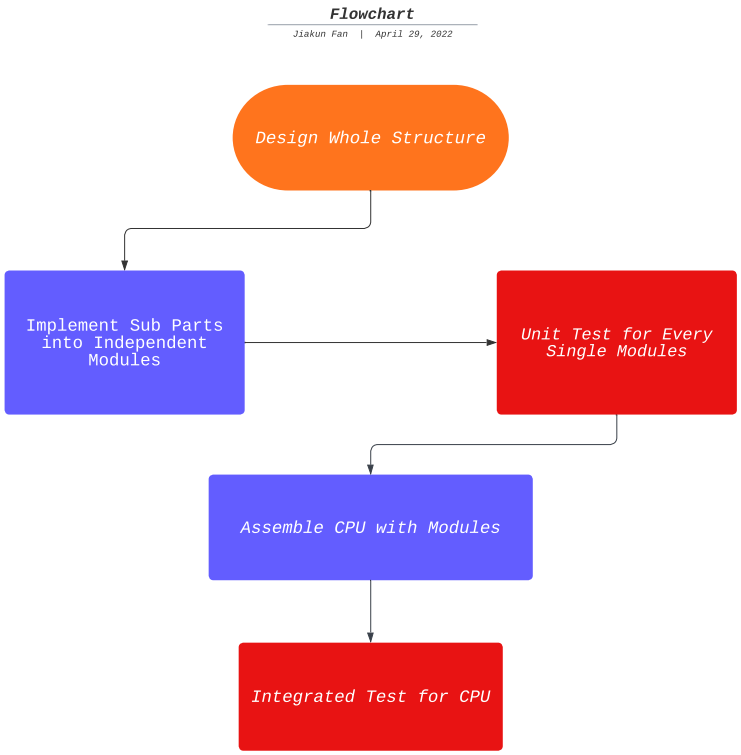

Figure 2: Data Flow Chart of CPU

2.2 Implementation Details

There are some drawbacks in the original structure shown in the requirements. We do some changes as shown below.

- **Shift Amount:** In the original structure, ALU cannot get shift amount which means **sll, srl, sra** are unable to execute. So we add one more signal **Shift** in Control Unit and store shift amount in the ID/EX buffer and then transfer to the ALU.
- **CPU Terminates:** When the IF/ID buffer get the Instruction **32'hffffffff**, it will start countdown using a counter with initial value 3. The counter will decrease one per clock. When the counter equals 1, main memory will be writen to file. When the counter equals 0, CPU terminates.
- **Hazard in Branch and Jump:** When branch and jump occurs, the instruction store in IF/ID, ID/EX buffers would be useless. To avoid those instructions being executed, we make the EX/MEM buffer not

handle the input in next three clocks when branch and jump occurs.

- **Forwarding**: We use a forwarding module to avoid hazard like RBW(read before write). We do not use any skills like staling to increase clock. So our whole cycles are the same as a normal pipeline CPU even faced with all kinds of hazards.

3 Test of the Project

3.1 Compile and Run the Program

We write a Makefile to test the CPU automatically.

```
cc = iverilog
flags = -o
test = test_CPU.v
main: $(test) Adder.v ALU.v ControlUnit.v CPU.v InstructionRAM.v MainMemory.v MUX.v PipeRegister.v
    Register.v Shift.v SignExtend.v HazardForwarding.v
  $(cc) $(flags) $@ $^
clean:
  rm -f main
```

You can execute this Makefile with inputing **make** in the terminal. After that, you should type **vvp main** in the terminal. Then you can check the main memory in the **MainMemory.txt**.