CSC3050 Computer Architecture

Project 3 report

Name: Xiao Nan

Student ID: 119010344

Date: 2021/5/14

The Chinese University of Hong Kong, Shen Zhen

0. Introduction:

    In project 3 I implemented a simple CPU module consisting of Datapath, Control, Pipeline and Hazard Detection. In the Datapath part, instructions are stored, fetched, and data in the memory and in the registers are exchanged. In the Control part, control signals specify how those data are transferred or modified. In the Pipeline part, pipeline registers synchronize data transfer through following the clock signal. In the Hazard Detection part, I detected hazards by control signals and register addresses and solved them using stalling and data forwarding. Also, I implemented a testbench for the convenience of testing the correctness of the CPU.

1. Big picture:

    In general, the execution of an instruction in CPU can be divided into 5 pipelined stages: In the first instruction fetch stage, the processor fetches an instruction from the instruction memory and passes it to the IF/ID pipeline register. In the second instruction decode stage, the information in the instruction such as the register addresses and the immediate number would be decoded and passed to the ID/EX pipeline register. Simultaneously, the control unit would decode control signals from the instruction for controlling data transfer. In the third execution stage, the ALU would perform operations on its selected inputs (such as the values in the registers and the immediate number) based on the ALU control and the result would be passed to the EX/MEM pipeline register. In the fourth memory reading or writing stage, where controlled by signals like MemRead and MemWrite, the data memory is read or written based on the address and data provided. Then both the values

read from the memory and the value calculated by ALU would be sent to the MEM/WB pipeline register. In the last writing back stage, controlled by the RegWrite signal, the register file would be read or written based on the value selected from ALU output and data from the memory, together with the write register address.

2. Data flow chart:

The data flow chart is shown in figure 2 below:

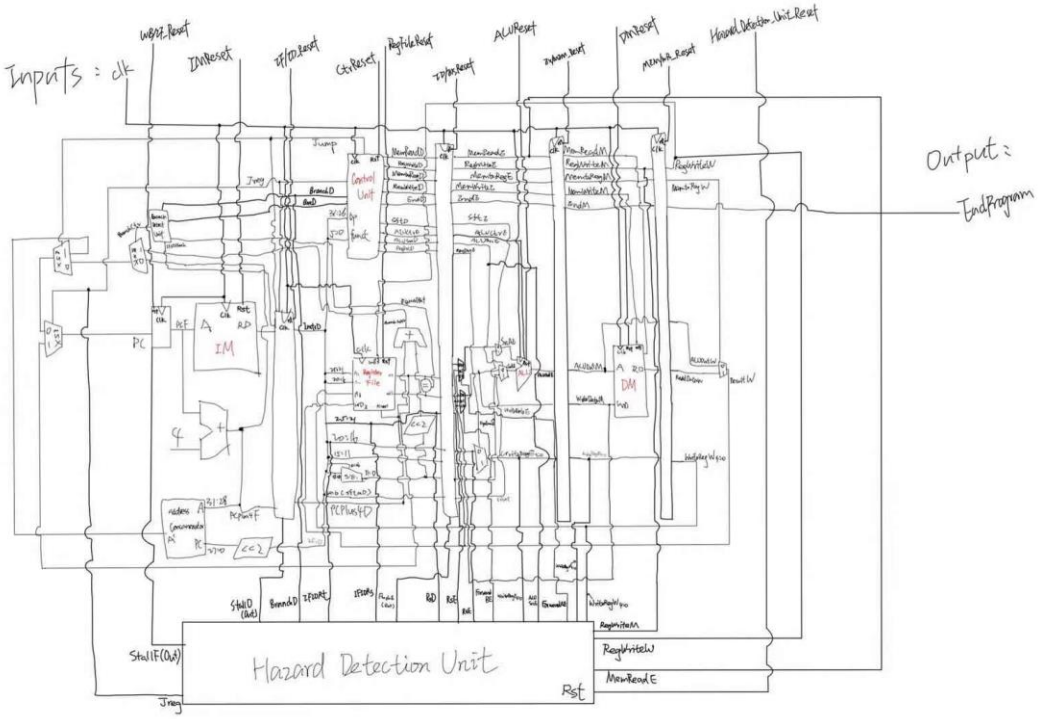

Figure 2: Data flow chart of the CPU

3. Implementation ideas:

I break down the problem into four main parts: Datapath construction, Control Unit construction, pipeline construction and hazard solving.

The Datapath construction part is further divided into four parts. In the first part, an instruction RAM is constructed which can fetch instructions. In the second part, a register file storing

the register values in a two-dimensional array is constructed, which can read or write registers according to their addresses and the control signals. In the third part, an ALU performing calculations based on the control signals and the inputs selected by multiplexers is constructed. In the fourth part, a data RAM with the size of 512 words is constructed. It can read or write values in the memory based on the address and the control signals. In addition, the operations of every component are synchronized with a clock signal in its negative edge. Each component has a reset signal for initialization.

The Control Unit is constructed taking the op code and the funct code of the instruction as inputs. It outputs the values for different control signals based on them using "case" syntax in Verilog. It is also negative-edge triggered by the clock signal.

The pipeline units consist of 5 pipeline registers, each passing data from stages IF to ID, ID to EX, EX to MEM, MEM to WB and WB to IF, correspondingly, controlled by the positive edge of the clock signal.

The hazard detection units consist of data hazard detection unit and branch hazard detection unit, which detect the possible occurring data hazards and branch hazards using the register addresses and the control signals from different stages. It also solves those hazards by providing control signals to components such as pipeline registers and multiplexers for data forwarding or stalling.

4. Implementation details:

In order to run the testbench, an "instructions.bin" file should be prepared under the same directory as the project (in

the submitted zip, the file is already given with the content of machine_code_1.txt of the provided cpu_test). You can compile with command “make”, and then run with command “./test.out” or “vvp test.out” under Linux. The outputs (the final state of the Data Memory) would be written in a file “output.txt” under the same directory, which shows the data memory after the execution of all instructions in the file. Also, the cycle length would be displayed in the terminal, as shown in figure 4.1.

```
E:\year2_term2\CSC3050\proj3_final\jj3rl>vvp test.out
VCD info: dumpfile test_CPU.vcd opened for output.
WARNING: InstructionRAM.v:23: $readmemb(instructions.bin): Not enough words in the file for the requested range [0:511].
cycle period: 2ns
1ns delay given by the testbench to ensure the complete printing of DM
test_CPU.v:56: $finish called at 113000000 (1fs)
```

Figure 4.1: Terminal results

The results after running the eight testbenches are shown in figure 4.2. Each cycle period is 2ns and all tests are passed.

Test	Cycles	Result Correctness
machine_code_1	56	✓
machine_code_2	15	✓
machine_code_3	18	✓
machine_code_4	17	✓
machine_code_5	179	✓
machine_code_6	54	✓
machine_code_7	50	✓
machine_code_8	33	✓

Figure 4.2: Testing results

5. Conclusion:

In this project I implemented a simplified CPU containing the function of 5 pipelined stages: instruction fetching, instruction decoding, basic ALU functions, data memory reading and writing, and register writing. I built the Datapath and constructed a control unit to control data operations. I detected possible hazards and solved them using stalling and data forwarding implemented in hazard detection units. Also, I implemented a testbench to test the functionality of the CPU.