CSA Project Report

Jinrui Zhang

April 30, 2024

1 In General

Completed code is named as *main.py* in folder *Code*. programming language used in this project is Python. And there are no other extra libraries or packages that need to be installed.

For running the simulator, the following code need to implemented in prompt or terminal:

```
cd path/to/main.py folder    #go to the objective folder
python main.py - -ioidr path/to/testcase/folder    #run the program
```

After run the command above, all result files will be stored under *testcase* folder, which is the same folder contains *imem.txt* and *dmem.txt*

2 Project Tasks

2.1 Draw the schematic for a single-stage processor and fill in your code to run the simulator.

The figure 1 shows the schematic for a single-stage processor, which is sourced from class slides. The specific code is in the *main.py*.

The diagram depicts a simplified view of the datapath of a processor, detailing the critical components involved in instruction processing. The Arithmetic Logic Unit (ALU) is at the heart of the architecture, responsible for performing arithmetic and logical operations. Instructions are read from the instruction memory, directing the control unit to generate appropriate signals for directing processor operations. The Program Counter (PC) ensures that instructions are executed in the correct order by storing the address of the current instruction and preparing the address for the next one. Multiplexers (MUX) are critical components of conditional decision-making and data flow control. Registers are used to store immediate values and intermediate results, whereas data memory is used for data handling read and write operations. To facilitate decoding and execution, the instruction's various components, such as the operation code, source register identifiers, and immediate values, are routed to their respective units.

2.2 Draw the schematic for a five stage pipelined processor and fill in your code to run the simulator.

The figure 2 shows the schematic for a five-stage processor, which is sourced from class slides. The specific code is in the *main.py*.

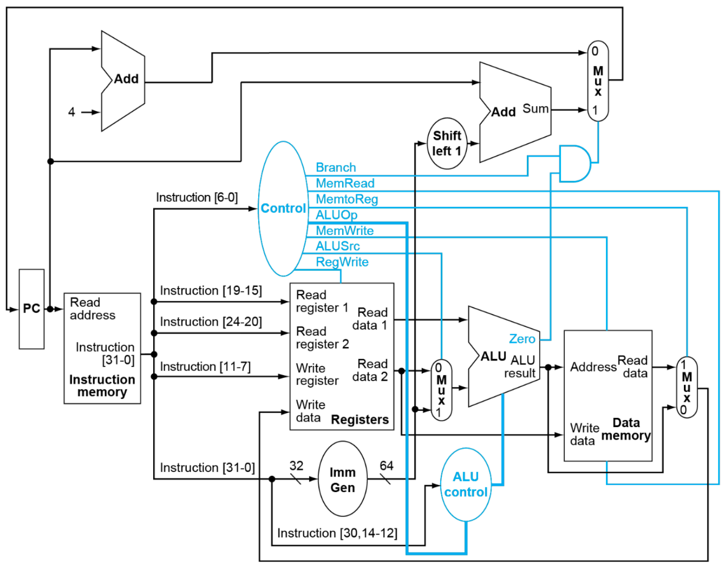

Figure 1: Single-stage schematic

***source: Class slides**

The diagram depicts a processor with advanced pipeline management, including hazard detection and resolution mechanisms. The hazard detection unit detects potential instruction execution conflicts that could disrupt the computational flow. To mitigate these risks, a forwarding unit is used, which allows data to bypass certain pipeline stages while maintaining instruction processing continuity. Flush controls, labeled IF.Flush, ID.Flush, and EX.Flush, are used to clear pipeline segments during branch mispredictions or instruction cancellations, ensuring program execution accuracy. SCAUSE and SEPC are specialized program counters that indicate the processor's ability to handle exceptions and interrupts, which is critical for responding to unexpected events. These components work together to optimize instruction throughput while also ensuring the integrity of operations within the processor's architecture.

2.3 Measure and report average CPI, Total execution cycles, and Instructions per cycle for both these cores by adding performance monitors to your code.

This function is implemented in the program. After running the program, there will be a file named *PerformanceMetrics_Result.txt* in the *output_jz6578* folder. Here I present some samples for test cases.

- For test case1:
  Single Stage Core Performance Metrics——————————
  Number of cycles: 6
  Cycles per instruction: 1.2
  Instructions per cycle: 0.833333

  Five Stage Core Performance Metrics——————————
  Number of cycles: 10
  Cycles per instruction: 2.0

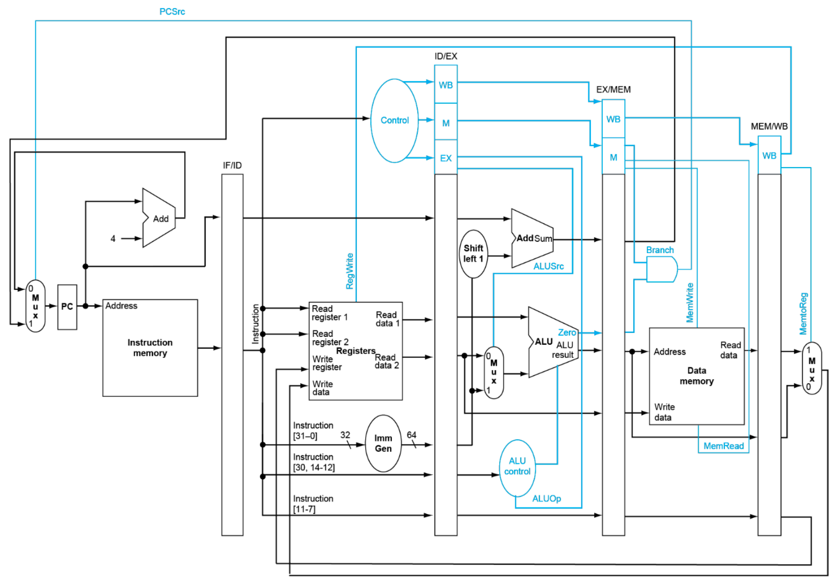

Figure 2: Five-stage schematic

***source: Class slides**

Instructions per cycle: 0.5

- For test case2:
  Single Stage Core Performance Metrics——————————
  Number of cycles: 40
  Cycles per instruction: 1.02564
  Instructions per cycle: 0.975001

  Five Stage Core Performance Metrics——————————
  Number of cycles: 46
  Cycles per instruction: 1.17949
  Instructions per cycle: 0.847824

- For test case3:
  Single Stage Core Performance Metrics——————————
  Number of cycles: 7
  Cycles per instruction: 1.75
  Instructions per cycle: 0.571429

  Five Stage Core Performance Metrics——————————
  Number of cycles: 10
  Cycles per instruction: 2.5
  Instructions per cycle: 0.4

2.4 Compare the results from both the single stage and the five stage pipelined processor implementations and explain why one is better than the other.

Metric	Single-Stage	Five-stage
CPI	Low	Hige
Instruction per cycle	High	Low
Execution time	High	Low
Hazards	No hazards	Control, Structure and data hazards
Style	One instruction each cycle	Several instructions are executed each cycle

To be specific, single-stage processors will never occur hazards since there is no resource conflict or inaccurate memory access. And there is no need for data storage or access between stages. For the five-stage processor, it has a faster clock rate and higher processor throughput than a single-stage processor since it reduces the amount of logic processed.

The cycle time of a single-stage processor is determined by the total amount of time required to process each instruction sequentially. However, the cycle time of a five-stage processor is influenced by the slowest stage. So a five-stage processor is much more likely to be faster than a single-stage processor.

2.5 What optimizations or features can be added to improve performance?

- Processor performance can be improved by increasing the number of pipeline steps or the depth of pipelining. As a result, each stage's logic processing will be reduced, improving clock rate and throughput.
- Reducing the amount of overhead required between stages for the storage and retrieval of critical data.
- Obtaining nearly balance among the five stages in order to alleviate time constraints caused by the slowest phase.