ECE-GY 6913
CSA Project Phase-1
Abdul Samad Zaheer Khan
ak9943@nyu.edu

Performance Modelling RISC-V Processor

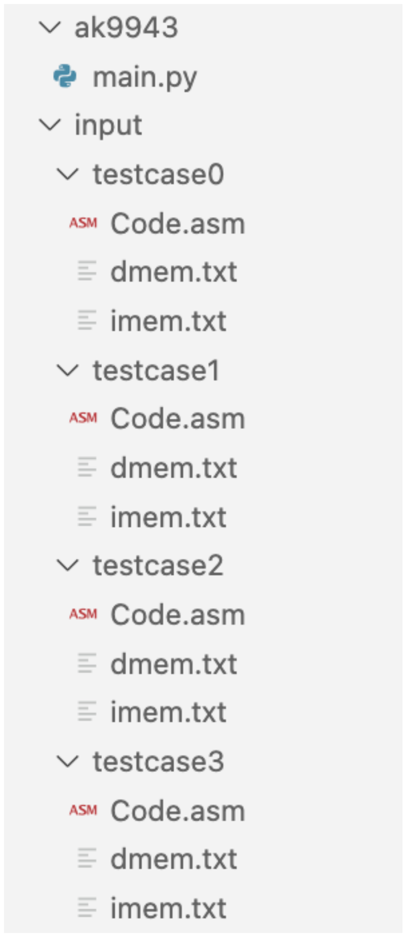
The phase-1 of the project simulates Single-step RISC-V processor and stores the necessary performance metrics.

The following folder structure is used:-

```
ak9943_csa_project
|
|__ak9943/
|       |
|       |__main.py
|
|__input/
|       |
|       |__testcase0/
|       |       |
|       |       |__dmem.txt
|       |       |__imem.txt
|       |
|       |__testcase1/
|       |       |
|       |       |__dmem.txt
|       |       |__imem.txt
|       |
|       |__testcase2/
|       |       |
|       |       |__dmem.txt
|       |       |__imem.txt
|       |
|       |__testcase3/
|               |
|               |__dmem.txt
|               |__imem.txt
|
|
|__output_ak9943/
        |
        |__PerformanceMetrics_Result.txt
        |__SS_DMEMResult.txt
        |__SS_RFResult.txt
        |__StateResult_SS.txt
        |__FS_DMEM_Result.txt
        |__StateResult_FS.txt
```

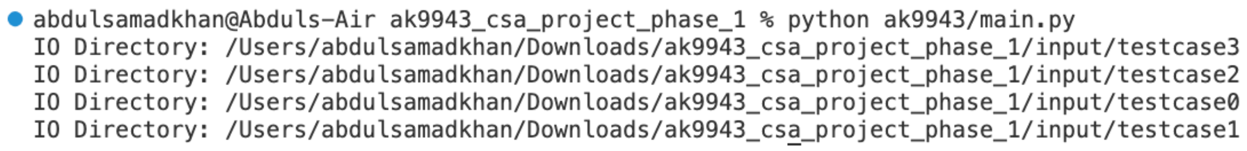
The folder **output_ak9943** is created after running the command **python ak9943/main.py** in the **ak9943_csa_project** folder.
The files **imem.txt** and **dmem.txt** are provided before runtime for every testcase.

The folder structure before running the command is:



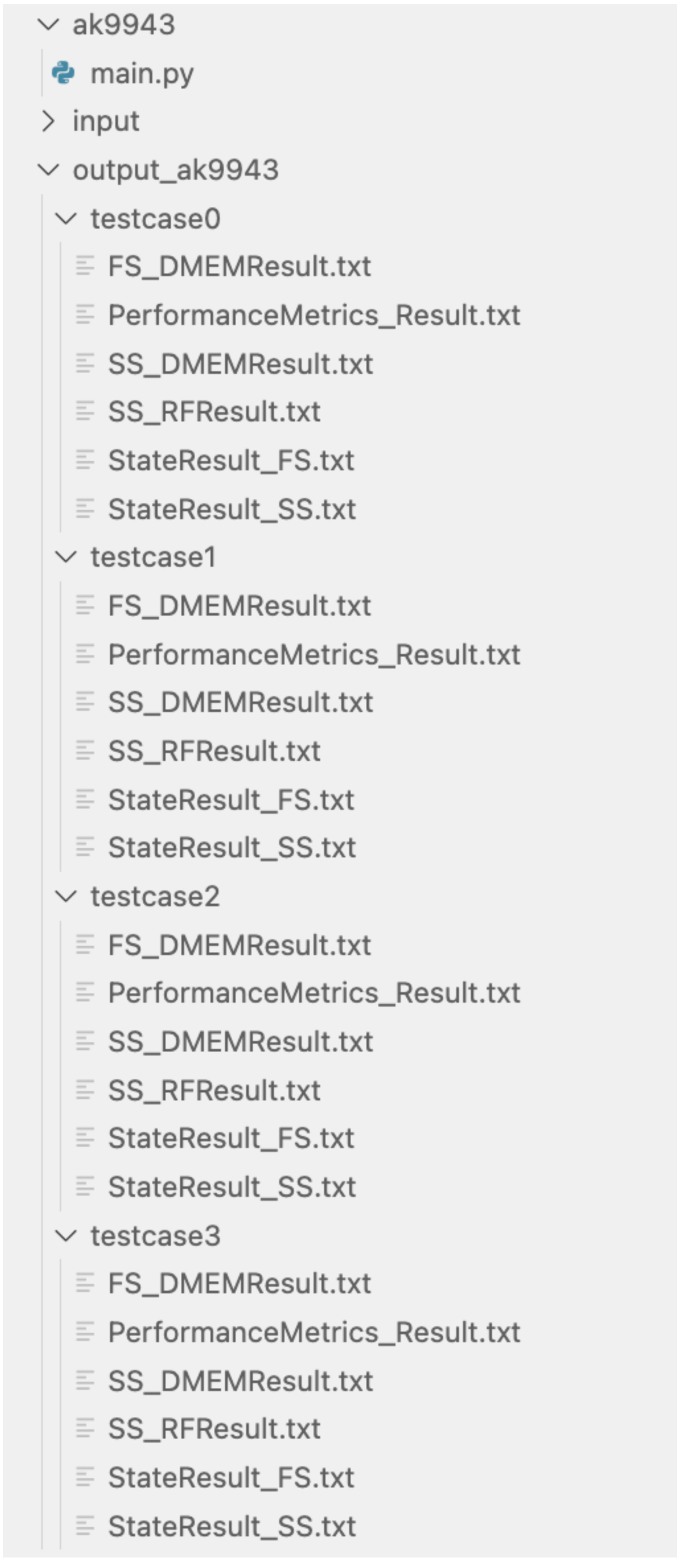
```
ak9943
  main.py
input
  testcase0
    Code.asm
    dmem.txt
    imem.txt
  testcase1
    Code.asm
    dmem.txt
    imem.txt
  testcase2
    Code.asm
    dmem.txt
    imem.txt
  testcase3
    Code.asm
    dmem.txt
    imem.txt
```

Terminal output after successfully running the python command to simulate the RISC-V ISA is:



```
abdulsamadkhan@Abduls-Air ak9943_csa_project_phase_1 % python ak9943/main.py
IO Directory: /Users/abdulsamadkhan/Downloads/ak9943_csa_project_phase_1/input/testcase3
IO Directory: /Users/abdulsamadkhan/Downloads/ak9943_csa_project_phase_1/input/testcase2
IO Directory: /Users/abdulsamadkhan/Downloads/ak9943_csa_project_phase_1/input/testcase0
IO Directory: /Users/abdulsamadkhan/Downloads/ak9943_csa_project_phase_1/input/testcase1
```

Folder structure after running the python command **python ak9943/main.py** in the **ak9943_csa_project** folder to generate performance metrics:

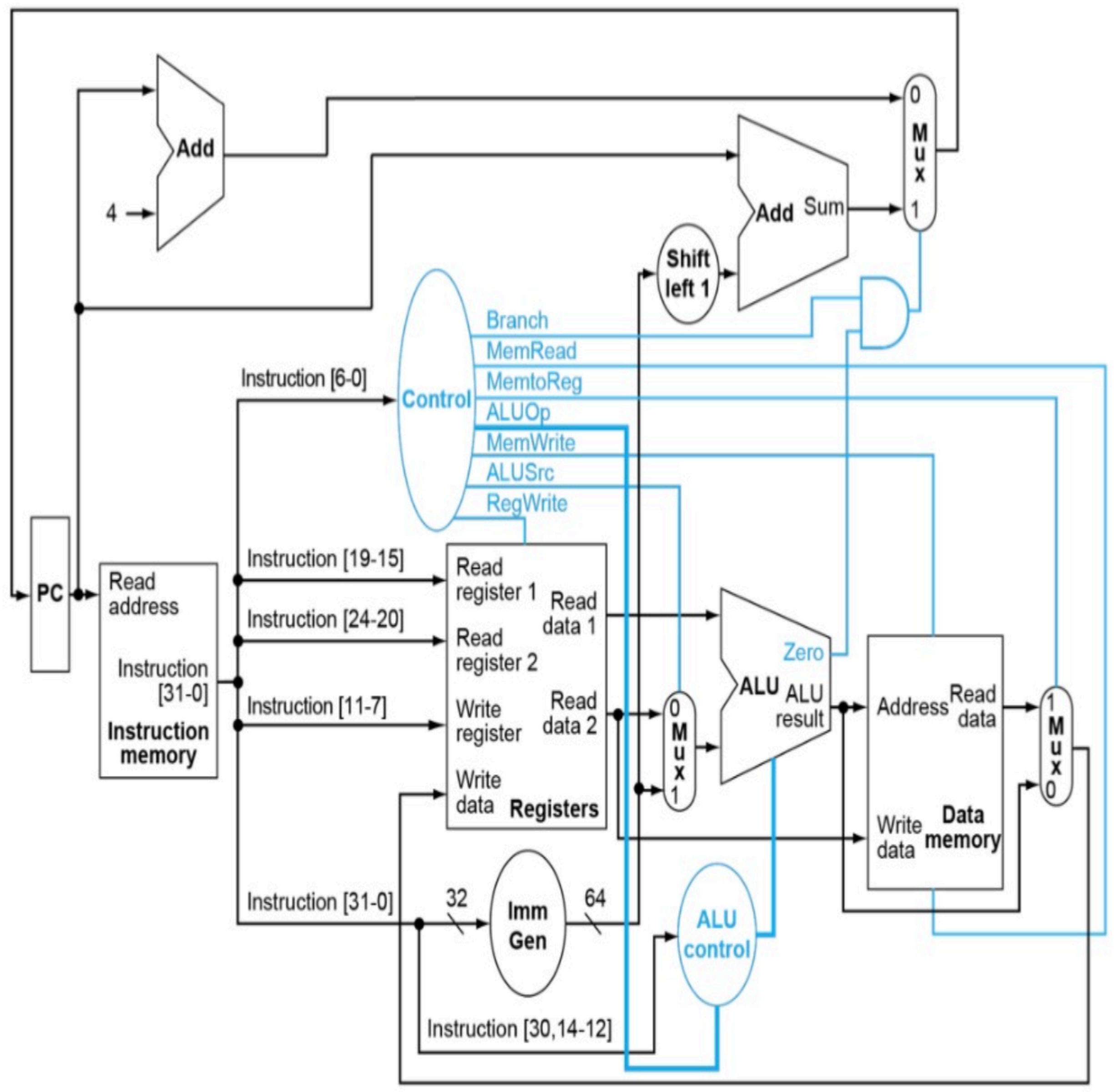


```
ak9943
    main.py
input
output_ak9943
    testcase0
        FS_DMEMResult.txt
        PerformanceMetrics_Result.txt
        SS_DMEMResult.txt
        SS_RFResult.txt
        StateResult_FS.txt
        StateResult_SS.txt
    testcase1
        FS_DMEMResult.txt
        PerformanceMetrics_Result.txt
        SS_DMEMResult.txt
        SS_RFResult.txt
        StateResult_FS.txt
        StateResult_SS.txt
    testcase2
        FS_DMEMResult.txt
        PerformanceMetrics_Result.txt
        SS_DMEMResult.txt
        SS_RFResult.txt
        StateResult_FS.txt
        StateResult_SS.txt
    testcase3
        FS_DMEMResult.txt
        PerformanceMetrics_Result.txt
        SS_DMEMResult.txt
        SS_RFResult.txt
        StateResult_FS.txt
        StateResult_SS.txt
```

Questions:

1. Draw the schematic for a single stage processor and fill in your code in the provided file to run the simulator.

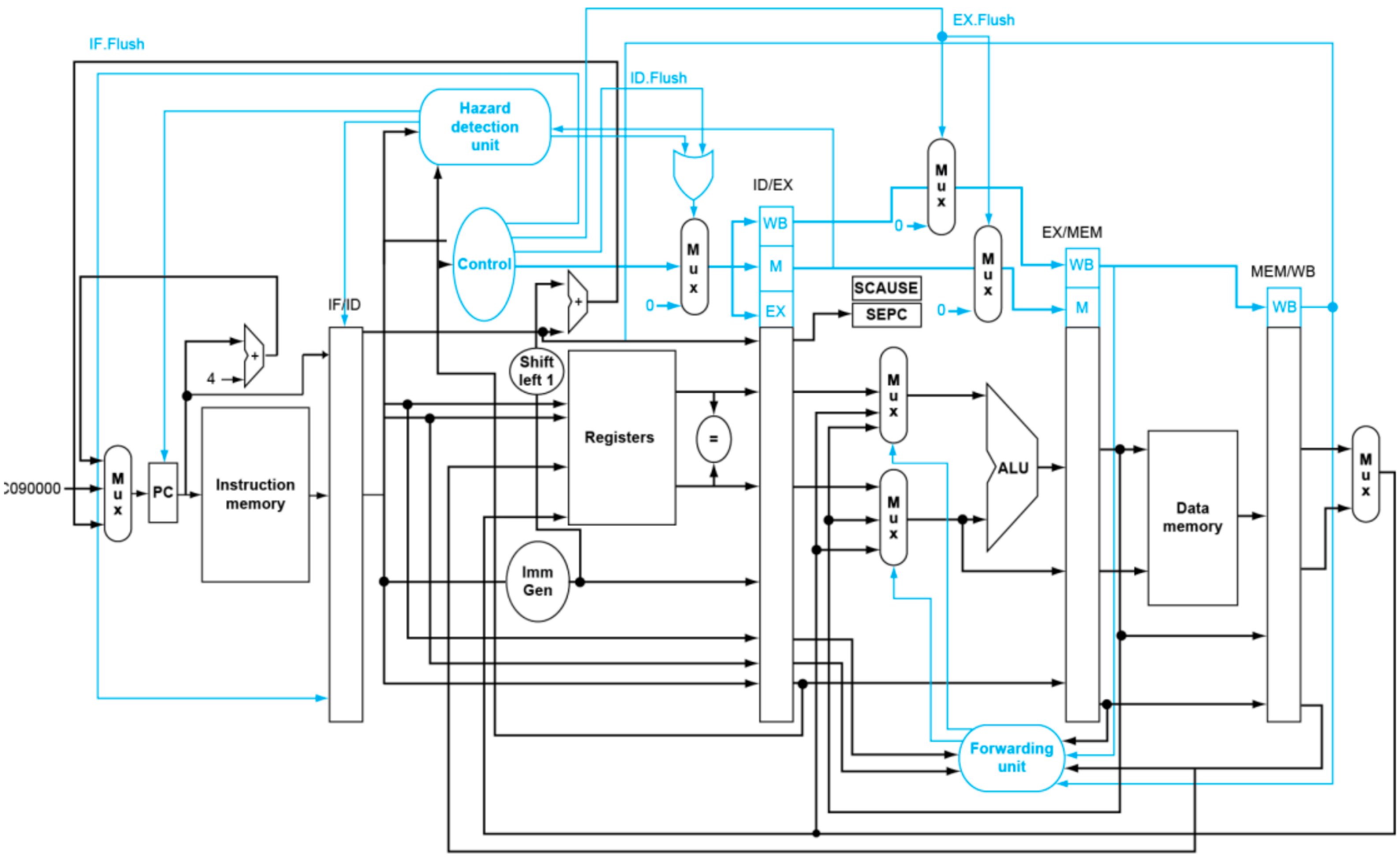
Solution:



***Sourced from lecture material**

2. Draw the schematic for a five stage processor and fill in your code in the provided file to run the simulator.

Solution:



***Sourced from lecture material**

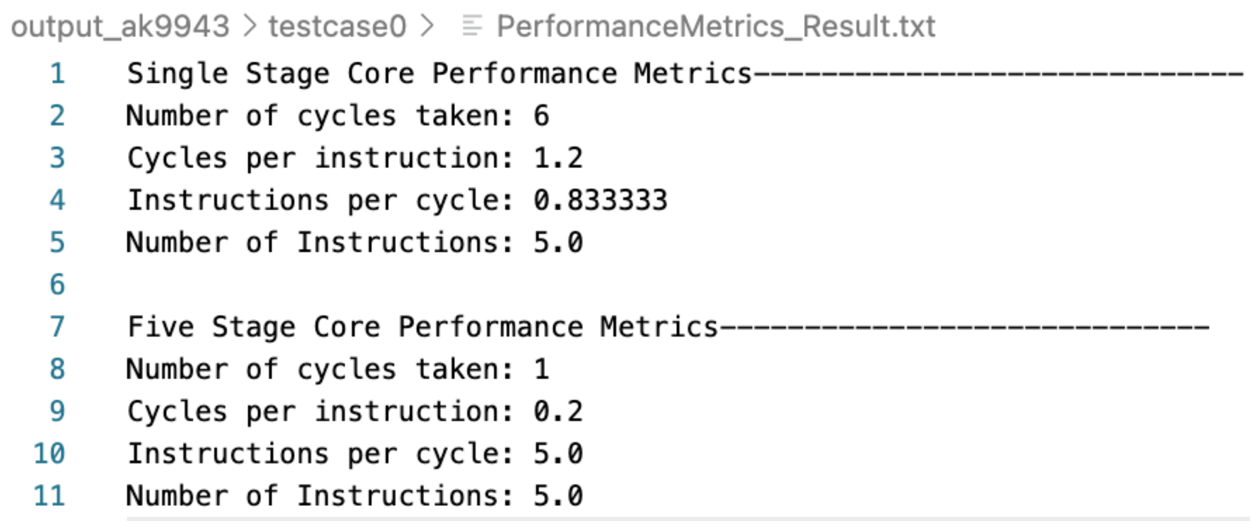
3. Measure and report average CPI, Total execution cycles, and Instructions per cycle by adding performance monitors to your code. Make sure you output these values to a file.

Solution:

Single Stage Performance Metrics

Testcase-0:
Single Stage Core Performance Metrics:-
Number of cycles taken: 6
Cycles per instruction: 1.2
Instructions per cycle: 0.833333
Number of Instructions: 5.0

Five Stage Core Performance Metrics:-
Number of cycles taken: 1
Cycles per instruction: 0.2
Instructions per cycle: 5.0
Number of Instructions: 5.0

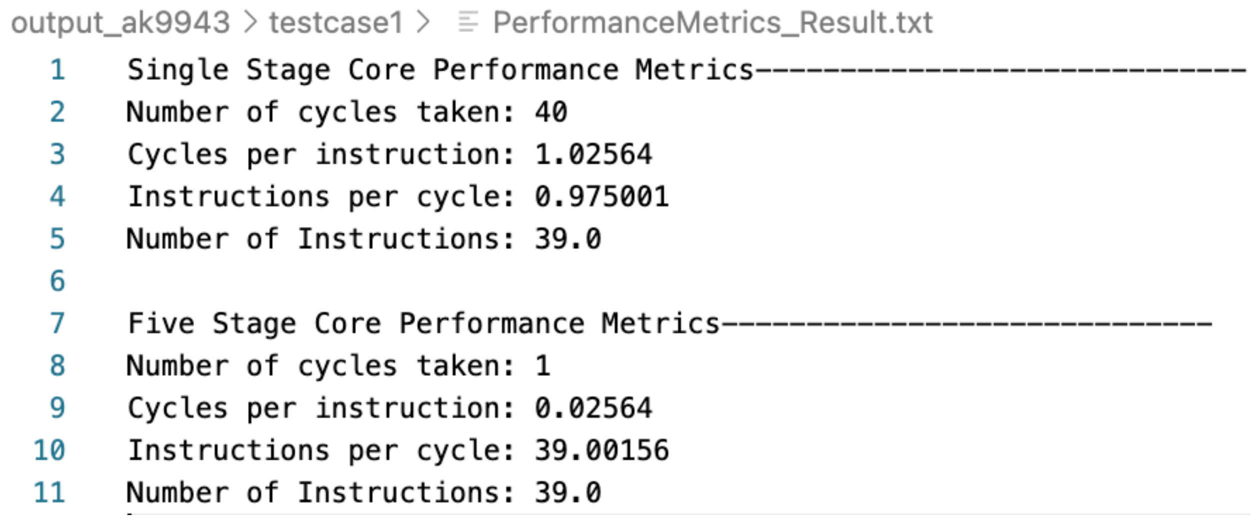

output_ak9943 > testcase0 > PerformanceMetrics_Result.txt

```
Single Stage Core Performance Metrics-----------------------------
Number of cycles taken: 6
Cycles per instruction: 1.2
Instructions per cycle: 0.833333
Number of Instructions: 5.0

Five Stage Core Performance Metrics-----------------------------
Number of cycles taken: 1
Cycles per instruction: 0.2
Instructions per cycle: 5.0
Number of Instructions: 5.0
```

Testcase-1:
Single Stage Core Performance Metrics:-
Number of cycles taken: 40
Cycles per instruction: 1.02564
Instructions per cycle: 0.975001
Number of Instructions: 39.0

Five Stage Core Performance Metrics:-
Number of cycles taken: 1
Cycles per instruction: 0.02564
Instructions per cycle: 39.00156
Number of Instructions: 39.0

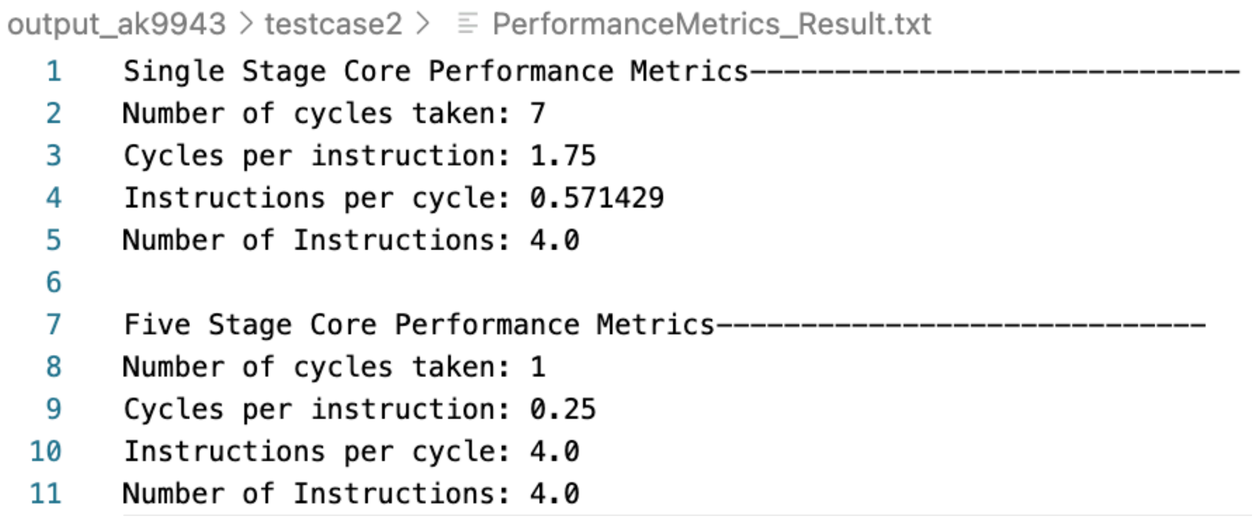


```
Single Stage Core Performance Metrics----------------------------
Number of cycles taken: 40
Cycles per instruction: 1.02564
Instructions per cycle: 0.975001
Number of Instructions: 39.0

Five Stage Core Performance Metrics----------------------------
Number of cycles taken: 1
Cycles per instruction: 0.02564
Instructions per cycle: 39.00156
Number of Instructions: 39.0
```

Testcase-2:
<u>Single Stage Core Performance Metrics</u>:-
Number of cycles taken: 7
Cycles per instruction: 1.75
Instructions per cycle: 0.571429
Number of Instructions: 4.0

<u>Five Stage Core Performance Metrics</u>:-
Number of cycles taken: 1
Cycles per instruction: 0.25
Instructions per cycle: 4.0
Number of Instructions: 4.0

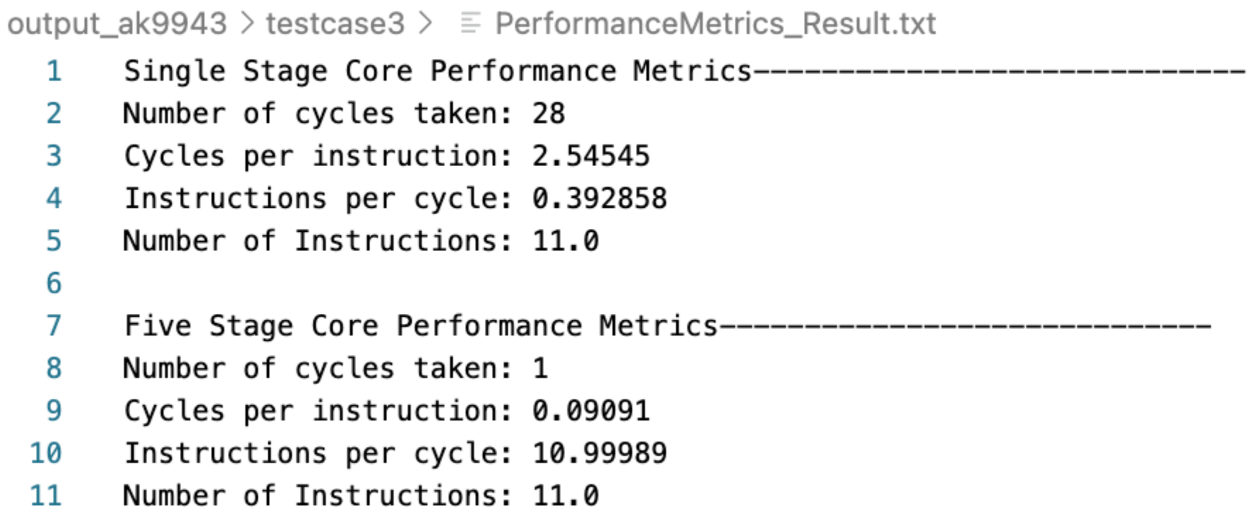

output_ak9943 > testcase2 > PerformanceMetrics_Result.txt

```
Single Stage Core Performance Metrics-----------------------------
Number of cycles taken: 7
Cycles per instruction: 1.75
Instructions per cycle: 0.571429
Number of Instructions: 4.0

Five Stage Core Performance Metrics-----------------------------
Number of cycles taken: 1
Cycles per instruction: 0.25
Instructions per cycle: 4.0
Number of Instructions: 4.0
```

Testcase-3:
<u>Single Stage Core Performance Metrics</u>:-
Number of cycles taken: 28
Cycles per instruction: 2.54545
Instructions per cycle: 0.392858
Number of Instructions: 11.0

<u>Five Stage Core Performance Metrics</u>:-
Number of cycles taken: 1
Cycles per instruction: 0.09091
Instructions per cycle: 10.99989
Number of Instructions: 11.0



```
Single Stage Core Performance Metrics-----------------------------
Number of cycles taken: 28
Cycles per instruction: 2.54545
Instructions per cycle: 0.392858
Number of Instructions: 11.0

Five Stage Core Performance Metrics-----------------------------
Number of cycles taken: 1
Cycles per instruction: 0.09091
Instructions per cycle: 10.99989
Number of Instructions: 11.0
```

4. What optimizations or features can be added to improve performance? (Extra credit)

Solution:

The following optimizations or features can be added to improve performance:

1. We can lower the overhead needed in between the steps to load and store required data.
2. Branch mis-prediction latency is a major issue affecting processor performance. To resolve this control flow hazard we can substitute arithmetic and lookup operations for the branch instructions.
3. At hardware level we can use small and fast caches to reduce memory access latency. This includes instruction caches and data caches to speed-up execution.
4. Another hardware level optimization is using simple static branch prediction techniques improve the flow of instruction execution, reducing the flow of control flow changes.
5. Software level optimization may include enhancing the compiler to generate more efficient code specifically designed for the RISC-V ISA.
6. Another software level optimization is to apply loop unrolling in software to reduce the overhead of loop control and increase the instruction throughput.
7. Optimizing the ALU for faster computation by using carry lookahead adders or other types of fast arithmetic circuits.
8. Using profiling to identify bottlenecks and optimizing critical sections of the code.
9. Optimize interrupt handling mechanisms to minimize latency and the overhead of context switching.
10. Improving bus architecture to allow faster data transfer rates between the processor and the peripherals.

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

Metric	**Single Stage Processor**	**Five Stage Pipelined Processor**
Hazards	No hazards can occur.	Data Hazard, Structural Hazard, Control Hazard, etc.. can occur.
Execution	1 instruction is executed per cycle.	At most 5 instructions can be executed parallelly in a single cycle.
Execution Time	Execution time is high.	Execution time is low.
Throughput	Lower throughput.	Higher throughput.
Cycles Per Instruction(CPI)	Low CPI.	High CPI.
Instructions Per Cycle(IPC)	High IPC.	Low IPC;

<u>Advantages of single-stage processor over a five-stage pipelined processor:</u>
1. In a single stage processor there is no need for overhead to store and access data in between stages of instructions.
2. A single-stage processor doesn't cause any structural, data and control hazards, because there is no forwarding, resource conflict or faulty memory access.
3. Single-stage processors are simpler in design compared to five-stage pipelined processor.

Advantages of five-stage pipelined processor over a single-stage processor:
1. Five-stage pipeline processor has a higher throughput compared to single-stage processor.
2. Five-stage pipeline processor make a better use of the hardware resources by concurrently processing different stages of different instructions.
3. Five-stage pipeline processor can easily achieve higher clock-rate compared to single stage processor.