PORTLAND STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 586 Computer Architecture

Spring 2020

Project: MIPS-lite 5-stage in-order Pipeline Simulator

Guidance: Prof. Zeshan Chishti

Wanyu Zhang (zhangwan@pdx.edu)
Aniruddha Shahapurkar (anirud@pdx.edu)

Date: 06/11/2020

❖ OBJECTIVE:

- The objective of the project is to model a simplified version of the MIPS ISA called MIPS-lite and the in-order 5-stage pipeline.
- We need to design a simulator which will implement the following two important features:
  1. A functional simulator which simulates the MIPS-lite ISA and captures the impact of instruction execution on machine state,
  2. A timing simulator which models the timing details for the 5-stage pipeline.

❖ INSTRUCTION SET & INSTRUCTION FORMAT:

- In this project, we dealt with following instruction sets.
  1. Arithmetic Instructions
  2. Logical Instructions
  3. Memory Access Instructions
  4. Control Flow Instructions

- The instructions mentioned above are specified in one of the following two instruction formats:
  1. R-type format
  2. I-type format

❖ INPUT:

- The simulator will take a memory image as its input.
- This memory image represents the initial state of the simulated program.
- The image will be provided as a text file which contains the initial contents of both the code segment (for instructions) and the data segment (for data) in a shared address space.
- Each line in the text file represents one-word (4 bytes) of memory shown in the hexadecimal (base-16) format.
- The addresses start at “0”, from the first line in the image and then increase by “4” at every next line.
- The data in the trace file is represented in the “Big Endian” format.

❖ SIMULATOR IMPLEMENTATION AND RESULTS:

- We implemented three different versions of the simulator:
  1. Functional simulator only
  2. Functional simulator + Timing simulator assuming no pipeline forwarding
  3. Functional simulator + Timing simulator with pipeline forwarding

➢ Functional Simulator:

- We need to develop a functional simulator which captures the effect of running the simulated program on the simulated machine state. The machine state includes the following three components:
  (i) Program counter (PC),
  (ii) General purpose registers (R1 to R31),
  (iii) Memory.
- The initial state of the memory is contained in the memory image.
- It is assumed that the PC and all the general-purpose registers are initialized to "0".
- Each instruction will update the PC, which will in turn cause a new instruction to be fetched and simulated.
- We need to continue the simulation until a "HALT" instruction is encountered.

➢ Timing Simulator:

- We need to develop a pipeline timing model which captures the flow of instructions through the 5-stage pipeline.
- Our model should be capable of detecting pipeline hazards and then accounting for any resultant stall cycles or flushing of instructions from the pipeline.
- The throughput of instructions through the pipeline should be 1 instruction per clock cycle, except when any of the following events occur:
  (i) RAW hazard,
  (ii) Taken branch.

➢ Results Obtained from Simulation:

[1]. Total number of instructions and a breakdown of instruction frequencies for the following instruction types: Arithmetic, Logical, Memory Access, Control Transfer:

- After running the simulator, we obtained the following results for Instruction counts:

Instruction Type	Count
Total number of instructions	**1361**
Arithmetic instructions	**558**
Logical instructions	**91**
Memory Access instructions	**450**
Control Transfer instructions	**262**

Fig. Results in Tabular form

[2]. Final state of program counter, general purpose registers and memory:

- After running the simulator, we obtained the following results for PC, Register array and memory array.

(i) Program Counter and General-Purpose Register state:

Program Counter	**112**

General-Purpose Register	Contents
R11	**1164**
R12	**1956**
R13	**2760**
R14	**85**
R15	**-188**
R16	**273**
R17	**119**
R18	**3560**
R19	**-1**
R20	**-2**
R21	**-1**
R22	**76**
R23	**3**
R24	**-1**
R25	**101**

Fig. Results in Tabular form

(ii) Final Memory State:

Address: Contents	Address: Contents	Address: Contents	Address: Contents
2400: 2	2468: 36	2536: 70	2604: 4
2404: 4	2472: 38	2540: 72	2608: 6
2408: 6	2476: 59	2544: 74	2612: 8
2412: 8	2480: 42	2548: 76	2616: 10
2416: 10	2484: 44	2552: 78	2620: 12
2420: 12	2488: 46	2556: 119	2624: 14
2424: 14	2492: 48	2560: 82	2628: 16
2428: 16	2496: 50	2564: 84	2632: 18
2432: 18	2500: 52	2568: 86	2636: 29
2436: 29	2504: 54	2572: 88	2640: 22
2440: 22	2508: 56	2576: 90	2644: 24
2444: 24	2512: 58	2580: 92	2648: 26
2448: 26	2516: 89	2584: 94	2652: 28
2452: 28	2520: 62	2588: 96	2656: 30
2456: 30	2524: 64	2592: 98	2660: 32
2460: 32	2528: 66	2596: 149	2664: 34
2464: 34	2532: 68	2600: 2	2668: 36

Address: Contents	Address: Contents	Address: Contents	Address: Contents
2672: 38	2696: 50	2720: 62	2744: 74
2676: 59	2700: 52	2724: 64	2748: 76
2680: 42	2704: 54	2728: 66	2752: 78
2684: 44	2708: 56	2732: 68	2756: 119
2688: 46	2712: 58	2736: 70	
2692: 48	2716: 89	2740: 72	

Fig. Results in Tabular form

[3]. Stall conditions in both the “no forwarding” and “forwarding” cases:

- In MIPS, there is no possibility of WAR or WAW hazards.
- We need to stall the pipeline in case of RAW hazards only.
- Therefore, we need to reduce the stalls by using techniques like forwarding, so that it will in turn reduce the execution time.

Let’s discuss the two different scenarios when we implement pipeline without and with forwarding.

(i) Scenario 1: No Forwarding

(a) When there is a gap of two or more than two instruction between the producer and the dependent consumer instruction:

- In this scenario we do not need to stall the cycles as by the time the consumer instruction will need its operands, they will be already available in the respective memory/register file

(b) When there is a gap of one instruction between the producer and the dependent consumer instruction:

- In this case, we need to introduce a stall of one cycle, so that by that time, the required contents will be available in the respective memory/register file.

(c) When the dependent consumer instruction is immediately followed after the producer instruction:

- In this case, we need to introduce a stall of two cycles, so that by that time, the required contents will be available in the respective memory/register file.

(ii) Scenario 2: Forwarding

(d) When there is a gap of two or more than two instruction between the producer and the dependent consumer instruction:

- In this scenario we do not need to stall the cycles as by the time the consumer instruction will need its operands, they will be already available in the respective memory/register file

(e) When there is a gap of one instruction between the producer and the dependent consumer instruction:

- In this case, we do not need to introduce any stall cycle, as we can use forwarding technique.

(f) When the dependent consumer instruction is immediately followed after the producer instruction:

- In this case, we need to introduce a stall of only one cycle, as we use the forwarding technique.

[4]. In the case of "no forwarding", the number of data hazards and average stall penalty per hazard:

Total number of stalls	**830**
Total number of data hazards	**460**
Average Stall Penalty	$\mathbf{\frac{830}{460} = 1.804}$

[5]. In the case of "forwarding", the number of data hazards which could not be fully eliminated by forwarding:

- After implementation of forwarding technique, we observed **"90"** hazards which were still present and could not be eliminated fully by forwarding.

[6]. Execution time in terms of number of clock cycles for the "no forwarding" and the "forwarding" scenarios:

- After running the simulator, we obtained the following results for number of clock cycles for the "no forwarding" and the "forwarding" scenarios:

Scenario	**Total clock cycles**
No Forwarding	**2553**
Forwarding	**1813**

- We can clearly see that, because of forwarding technique, the simulator execution requires less number of clock cycles when compared with no forwarding scenario.

[7]. Speedup achieved by “forwarding” as compared to “no forwarding”:

- We know that,

Speedup = $\frac{Execution\ Time\ (Non{-}Forwarding)}{Execution\ Time\ (Forwarding)}$

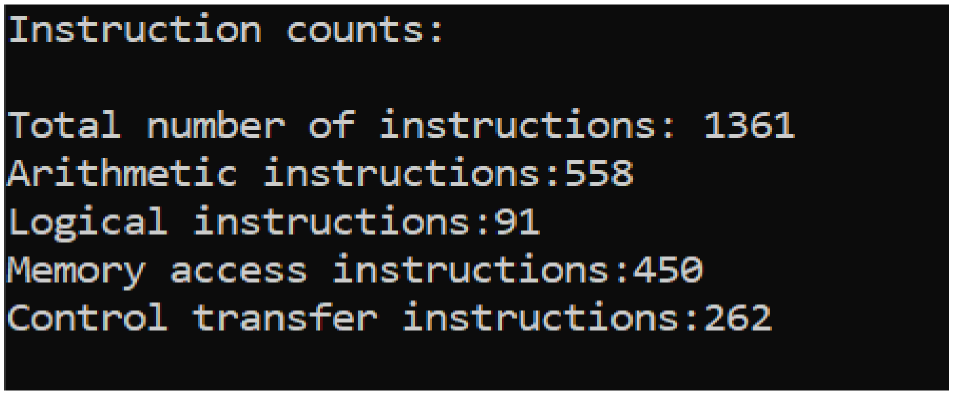
In our case,
Clock cycles for non-forwarding case scenario: 2553 and
clock cycles for forwarding case scenario: 1813

Therefore,

Speedup = $\frac{2553}{1813}$

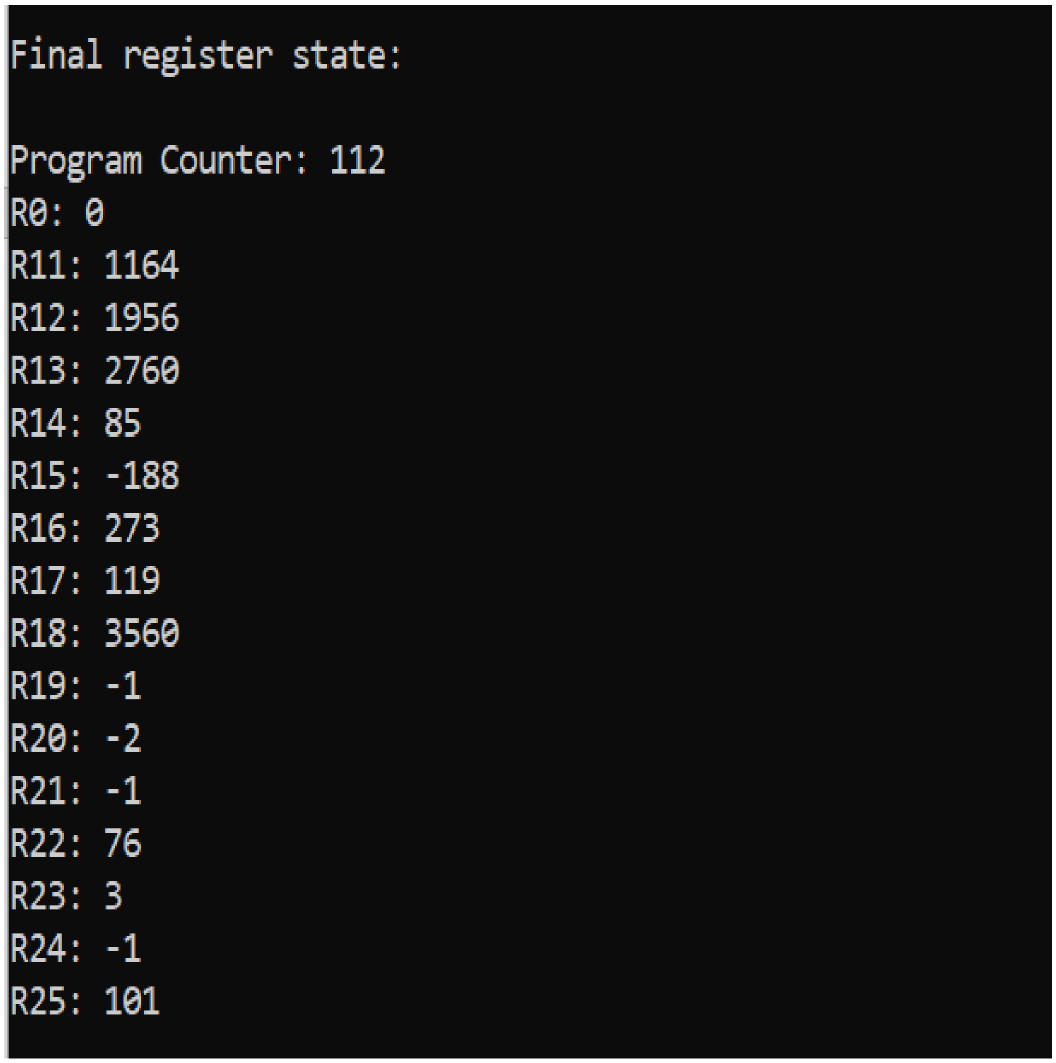
Speedup = **1.408**

➢ **Snapshots of the results obtained from Simulation:**



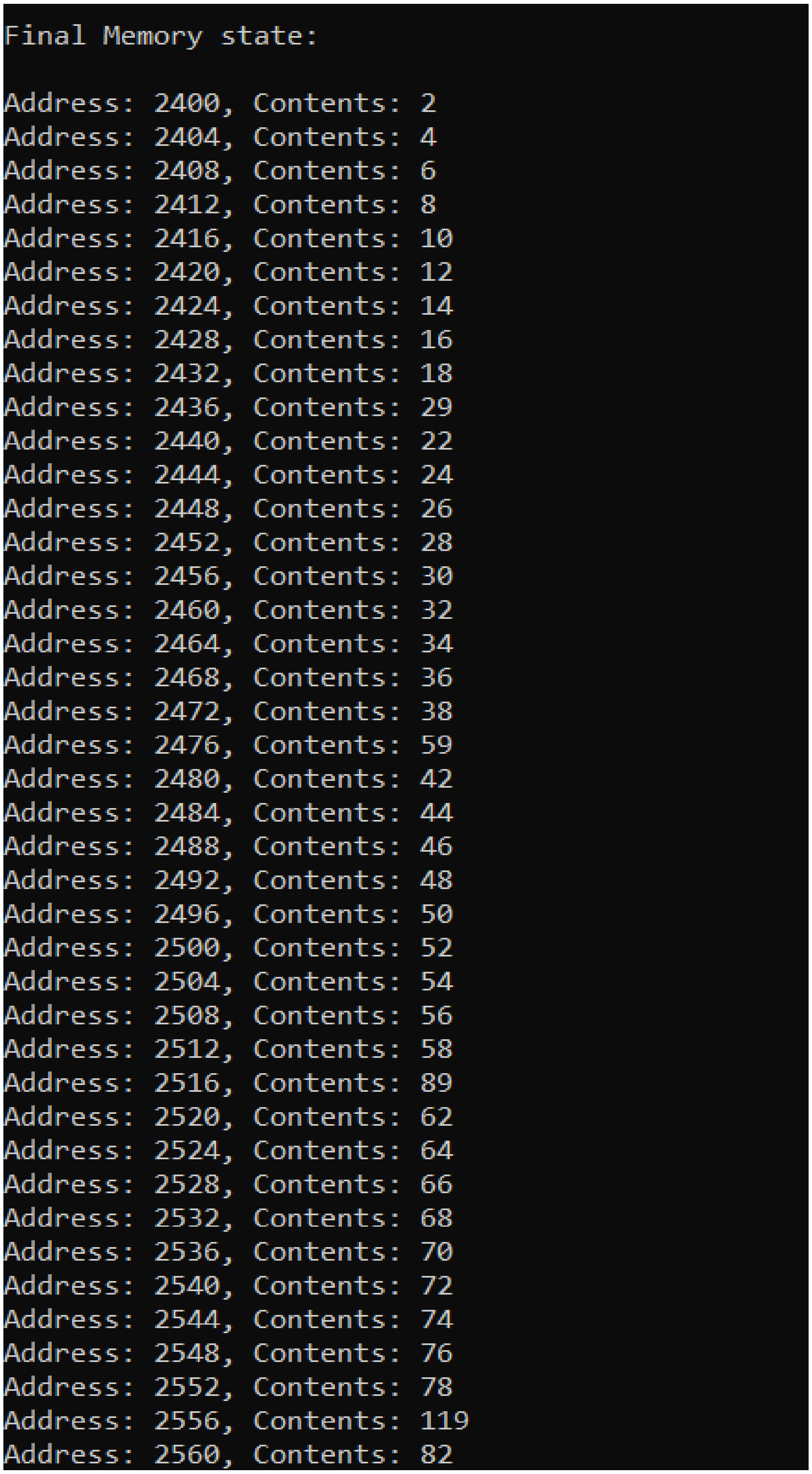
```
Instruction counts:

Total number of instructions: 1361
Arithmetic instructions:558
Logical instructions:91
Memory access instructions:450
Control transfer instructions:262
```



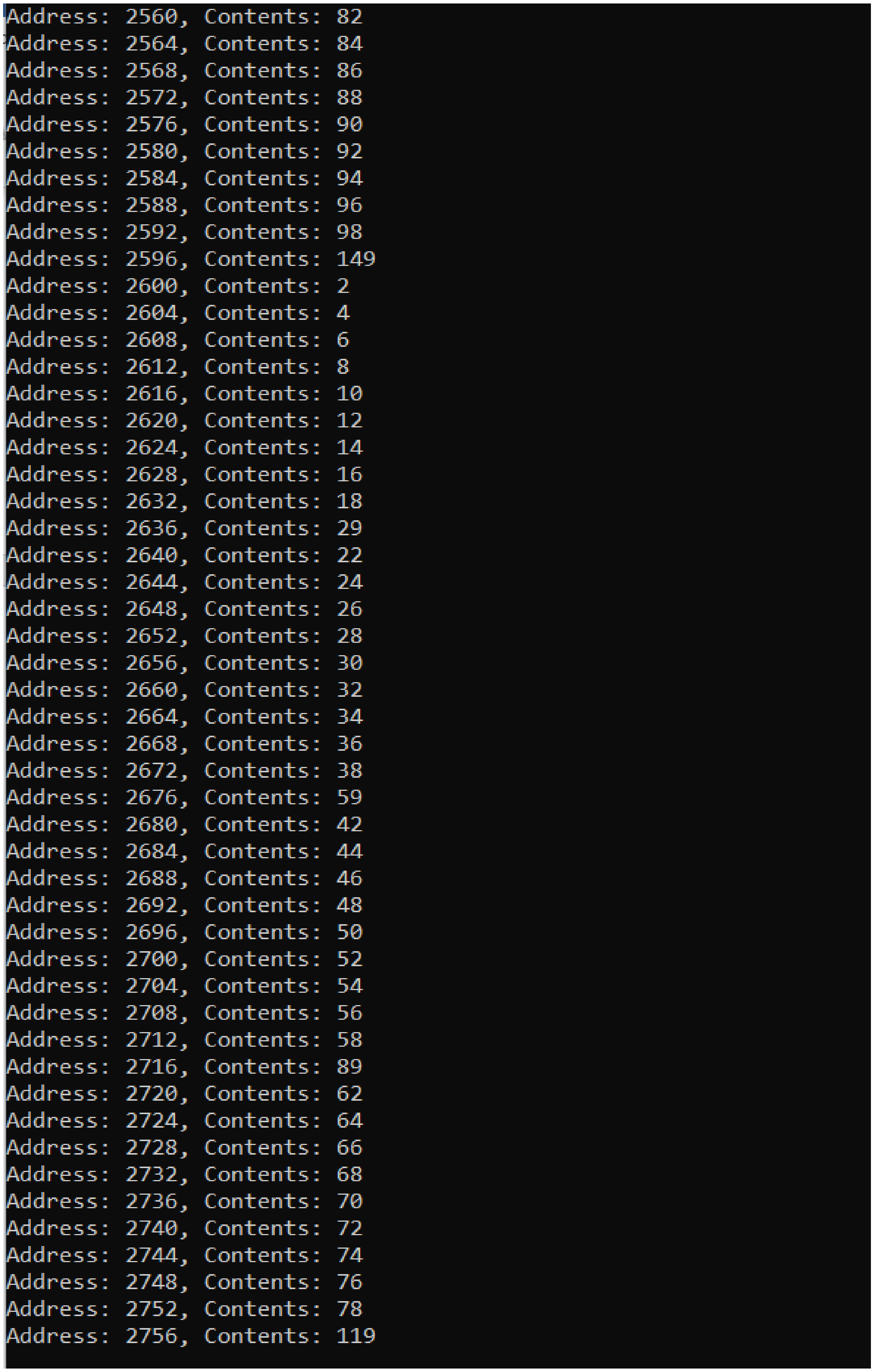
```
Final register state:

Program Counter: 112
R0: 0
R11: 1164
R12: 1956
R13: 2760
R14: 85
R15: -188
R16: 273
R17: 119
R18: 3560
R19: -1
R20: -2
R21: -1
R22: 76
R23: 3
R24: -1
R25: 101
```

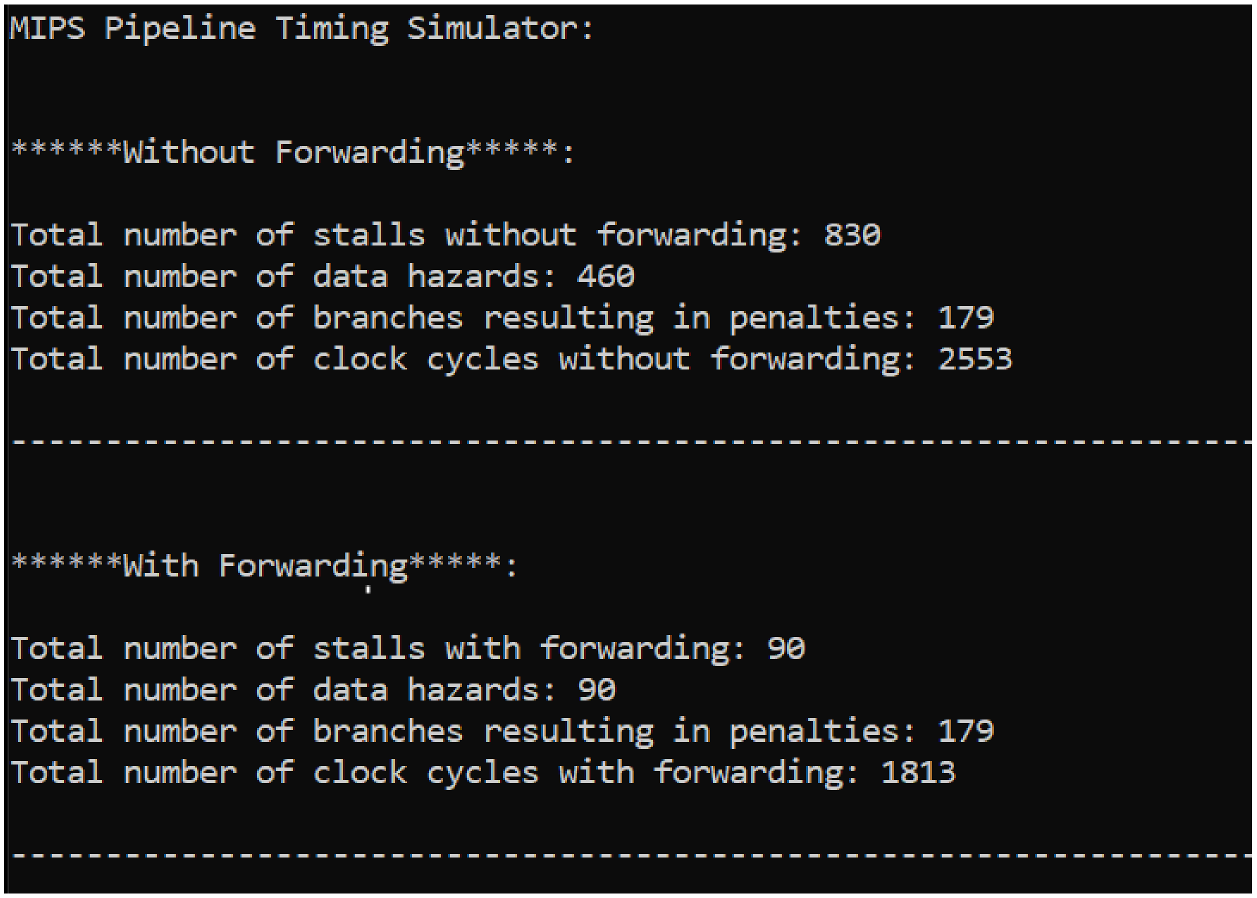


```
Final Memory state:

Address: 2400, Contents: 2
Address: 2404, Contents: 4
Address: 2408, Contents: 6
Address: 2412, Contents: 8
Address: 2416, Contents: 10
Address: 2420, Contents: 12
Address: 2424, Contents: 14
Address: 2428, Contents: 16
Address: 2432, Contents: 18
Address: 2436, Contents: 29
Address: 2440, Contents: 22
Address: 2444, Contents: 24
Address: 2448, Contents: 26
Address: 2452, Contents: 28
Address: 2456, Contents: 30
Address: 2460, Contents: 32
Address: 2464, Contents: 34
Address: 2468, Contents: 36
Address: 2472, Contents: 38
Address: 2476, Contents: 59
Address: 2480, Contents: 42
Address: 2484, Contents: 44
Address: 2488, Contents: 46
Address: 2492, Contents: 48
Address: 2496, Contents: 50
Address: 2500, Contents: 52
Address: 2504, Contents: 54
Address: 2508, Contents: 56
Address: 2512, Contents: 58
Address: 2516, Contents: 89
Address: 2520, Contents: 62
Address: 2524, Contents: 64
Address: 2528, Contents: 66
Address: 2532, Contents: 68
Address: 2536, Contents: 70
Address: 2540, Contents: 72
Address: 2544, Contents: 74
Address: 2548, Contents: 76
Address: 2552, Contents: 78
Address: 2556, Contents: 119
Address: 2560, Contents: 82
```



```
Address: 2560, Contents: 82
Address: 2564, Contents: 84
Address: 2568, Contents: 86
Address: 2572, Contents: 88
Address: 2576, Contents: 90
Address: 2580, Contents: 92
Address: 2584, Contents: 94
Address: 2588, Contents: 96
Address: 2592, Contents: 98
Address: 2596, Contents: 149
Address: 2600, Contents: 2
Address: 2604, Contents: 4
Address: 2608, Contents: 6
Address: 2612, Contents: 8
Address: 2616, Contents: 10
Address: 2620, Contents: 12
Address: 2624, Contents: 14
Address: 2628, Contents: 16
Address: 2632, Contents: 18
Address: 2636, Contents: 29
Address: 2640, Contents: 22
Address: 2644, Contents: 24
Address: 2648, Contents: 26
Address: 2652, Contents: 28
Address: 2656, Contents: 30
Address: 2660, Contents: 32
Address: 2664, Contents: 34
Address: 2668, Contents: 36
Address: 2672, Contents: 38
Address: 2676, Contents: 59
Address: 2680, Contents: 42
Address: 2684, Contents: 44
Address: 2688, Contents: 46
Address: 2692, Contents: 48
Address: 2696, Contents: 50
Address: 2700, Contents: 52
Address: 2704, Contents: 54
Address: 2708, Contents: 56
Address: 2712, Contents: 58
Address: 2716, Contents: 89
Address: 2720, Contents: 62
Address: 2724, Contents: 64
Address: 2728, Contents: 66
Address: 2732, Contents: 68
Address: 2736, Contents: 70
Address: 2740, Contents: 72
Address: 2744, Contents: 74
Address: 2748, Contents: 76
Address: 2752, Contents: 78
Address: 2756, Contents: 119
```



```
MIPS Pipeline Timing Simulator:


******Without Forwarding*****:

Total number of stalls without forwarding: 830
Total number of data hazards: 460
Total number of branches resulting in penalties: 179
Total number of clock cycles without forwarding: 2553

-----------------------------------------------------------------



******With Forwarding*****:

Total number of stalls with forwarding: 90
Total number of data hazards: 90
Total number of branches resulting in penalties: 179
Total number of clock cycles with forwarding: 1813

-----------------------------------------------------------------
```

Fig. Snapshots of Actual output obtained from the simulator

❖ Conclusion:

Thus, in this project, we implemented the MIPS-lite in-order 5-stage pipeline using C++ scripting and generated results for its functional and timing simulation. We achieved a speedup of 1.40 with the help of forwarding technique.