Computer Architecture Assignment 2

Submission Details:

The zip file contains the following files:

1. IMT2022527_Report.pdf: This pdf contains a brief explanation of the processor codes along with screenshots of the outputs.
2. "MIPS_PROGRAMS" folder which contains the following files
    a. IMT2022527_InsertionSort.asm: MIPS code for performing Insertion Sort.

    b. IMT2022527_Factorial.asm: MIPS code for performing factorial of a number in memory.

3. “PROCESSOR_AND_MACHINE_CODES” folder which contains the following files:
    a. IMT2022527_Non_Pipeline.cpp: This file contains the C++ code to simulate the working of a non-pipeline processor.
    b. IMT2022527_Pipeline.cpp: This file contains the C++ code to simulate the working of a pipeline processor.
    c. Factorial.txt: This file contains the machine code of the MIPS program for calculating the factorial of a number.
    d. Insertion_sort.txt: This file contains the machine code of the MIPS program for sorting the numbers in memory.

Assumptions:

->Separate memory for both instruction and data.

->Branching can be detected in the Execute phase itself.

->PC can be updated in the decoded phase itself on detecting a j, jal, jr instruction.

->The stages that are flushed are not shown in the output.

Non-Pipeline Processor (C++):

The C++ code has several pre-defined elements:

->Macros for defining the starting addresses of the instruction, data memory, register numbers and the total space of the data memory.

->Global control signals for convenient access.

->A ‘long long int’ declared globally to track the Program Counter (PC), along with a global counter for clock cycles.

->Various functions are implemented to choose between 2 values based off the control signals and a few functions to perform certain tasks (explained in the code).

->One unordered map (instructions_map) which maps the different types of opcodes of the instructions to the set of values of the control signals that these MIPS instructions have. The R-Type MIPS instructions have the same set of control signals but they do have different ALU control signals.

->Two unordered maps (R_control, I_J_control) to map the opcode to the ALU control. For R-Type instructions, the R_control map is used which maps the function field of the instruction to the ALU control value whereas the I_J_control map is used for the other instructions which map the opcode itself to the ALU control.

The code now takes input from user asking for which program to be executed (here, Factorial or Sorting). It then takes necessary input from user and reads from the appropriate ‘.txt’ based off the program chosen and fills the instruction memory and the data memory.

Now, the code starts to execute the non-pipeline processor using a while loop that continues until the PC points to the line after the last MIPS instruction.

In the first stage, INSTRUCTION FETCH(IF), the processor fetches the instruction from the instruction memory at the PC index (mapped down to indices 0,1,2…etc) only if the PC is valid. If not, the lops breaks, ending the program. The fetched instruction is stored in an appropriate variable.

The processor now moves on to the INSTRUCTION DECODE(ID) stage, where it extracts portions of the instruction, assigns the decimal equivalent to the read ports of the RegFile, generates signals using the instructions unordered map. It then fills the write port of the RegFile in addition to calculating the sign-exteneded immediate value. This marks the end of the ID stage.

Moving to the Execute (Ex) stage, the processor assigns ALU sources based on the RegFile read ports, control signals, and sign-extended value. It performs the necessary ALU operation using the ALU control signal(generated using the 2 unordered maps for ALU control), stores the result in the 'ALU_result' and 'Zero' variables, and updates the PC (based on the appropriate control signals) to fetch the next instruction after the completion of the present instruction. This marks the end of the Execute stage.

Now, the processor starts the MEMORY ACCESS (Mem) stage, where data is either written into memory or read from memory based off the appropriate signals MemWrite, MemRead respectively.

The final stage is Writeback(WB). In this stage, the appropriate value (based on the MemtoReg signal) is written back into the necessary register based on whether the RegWrite signal is high or not. This marks the end of the WB stage after which the processor again starts the IF stage and it goes on. This loop breaks only when the PC is pointing to a location out of range of the instruction memory.

When the processor finally breaks out of the loop, the appropriate output (based on the program executed) is printed. Throughout each stage, certain variables and the clock cycle are printed for better understanding of the processor’s operation.

Pipeline Processor (C++):

This C++ code also has several pre-defined elements:

->Macros for defining the starting addresses of the instruction, data memory, register numbers and the total space of the data memory.

->A ‘long long int’ declared globally to track the Program Counter (PC), along with a global counter for clock cycles.

->One unordered map (instructions_map) which maps the different types of opcodes of the instructions to the set of values of the control signals that these MIPS instructions have. The R-Type MIPS instructions have the same set of control signals but they do have different ALU control signals.

->Two unordered maps (R_control, I_J_control) to map the opcode to the ALU control. For R-Type instructions, the R_control map is used which maps the function field of the instruction to the ALU control value whereas the I_J_control map is used for the other instructions which map the opcode itself to the ALU control.

->Four different classes to represent the 4 different pipeline registers (IF/ID, ID/Ex, Ex/MEM, MEM/WB) used in a pipeline processor.

->A class to keep track of the state values of the 5 stages which determine whether that stage will be executed or not.

The code now takes user input in a similar format to the input taken in the non-pipeline processor C++ file, fills in the instruction memory, data memory as needed and calls the pipeline processor.

The processor starts off by initializing two copies of the 4 pipeline registers, one set of which are used to execute a certain stage and the other set which remains empty throughout the program. The program also initialises an instance of the class 'states' with the values of the five states being '1'.

Now, the code proceeds to execute the pipeline processor using a while loop that continues until the WB of the last instruction in the pipeline is finished. Inside the loop, the Wb,Ex,Mem,ID,IF stages are executed only if the state value associated to them are 5,4,3,2,1 respectively, else the stage is not executed and the state value of that particular stage is incremented by 1.

In the code, any stage (except for the IF stage) is executed only if the pipeline before it is not empty i.e., ID stage is executed only if the IF/ID pipeline is not empty, Ex stage is executed only if the ID/Ex pipeline is not empty and so on. In each stage, the information from the pipeline before it, is accessed and the necessary operations of that stage are performed after which the results of the operations are stored onto a temporary pipeline of the same type as that of the pipeline next to this stage(except for the WB stage) i.e., The Ex stage stores it's results into a temporary Ex/Mem pipeline register and so on.

In the Writeback stage, the information present in the Mem/Wb pipeline register is accessed and the writeback operations are done on them. Hence, the value of a register will be updated only after the WB stage. It is important to note that the writing into the RegFile happens in the first half of the clock cycle and reading from the RegFile happens in the second half of the clock cycle.

In the Memory Access stage, the information present in the Ex/Mem pipeline register is used to write into memory or read into memory. The code also forwards the required value in case of a data hazard, after which the necessary operations are done. The results of this stage are then pushed onto a temporary Mem/WB pipeline register.

In the Execute stage, the information present in the ID/Ex pipeline register is accessed to perform the necessary ALU operation of the instruction. Before performing the ALU operation, the code checks for any dependencies by comparing the index of the destination register of the Mem/Wb and the Ex/Mem pipeline registers (in the same order) with the indices of the source registers present in the

ID/Ex register. If there is a dependency (the indices match), the required value (ALU result or write data) is forwarded and used. But in cases where the value to be forwarded is produced only after the completion of the Memory access phase, the code implements a stall by not executing the Ex,ID,IF stages of the pipeline and updating only the Mem/Wb pipeline register in addition to making Ex/Mem pipeline register an empty pipeline register. Thus, in the next clock cycle, the required value can be obtained by simply forwarding. After taking care of the dependencies, the code simply performs the required ALU operations to be done using the unordered maps for generating the ALU control values and performs the necessary ALU operation. In the case that the Zero signal becomes high and it is a beq instruction or when the Zero signal is low and it is a bne instruction, then the code simply flushes the ID,IF instruction as they would have the instructions at a different address and also makes the IF/ID, ID/Ex pipeline registers to be empty. The value of PC is updated as needed before flushing the pipeline. This way the correct instruction is fetched starting from the next clock cycle. PC is simply incremented by 4 if it is not going to branch. Finally, the results of the Ex phase are pushed on to a temporary pipeline register Ex/MEM.

In the INSTRUCTION DECODE (ID) stage, the information present in the IF/ID pipeline register is accessed to decode the instruction and pass it on to the next pipeline register. In the case that the decoded instruction is either a J or Jal or Jr MIPS instruction, the code flushes the IF stage since the IF would fetch the instruction at PC+4 and not the correct target address. The code also updates PC to the correct address before moving on to the next clock cycle. Finally, the decoded instructions are passed onto a temporary ID/Ex pipeline register.

In the INSTRUCTION FETCH (IF) stage, the instruction at index PC (mapped down to indices 0,1,2…etc) is fetched from the instruction memory and is passed down to a temporary IF/ID pipeline register along with the value PC+4. The program simply doesn't fetch the instruction when the PC doesn't point to a valid index in the instruction memory.

Finally, after the execution of the five stages, the pipeline registers are updated to the temporary registers which hold the results of the execution of each stage. Then the code moves on to the next clock cycle repeating the same 5 five stages for the updated pipeline registers. This continues to happen until all the pipeline registers are empty after which the code breaks out of the loop.

The code then prints the appropriate output based on the executed program. Throughout the execution of the 5 stages, the code also prints certain variables for better understanding the processor's operations.

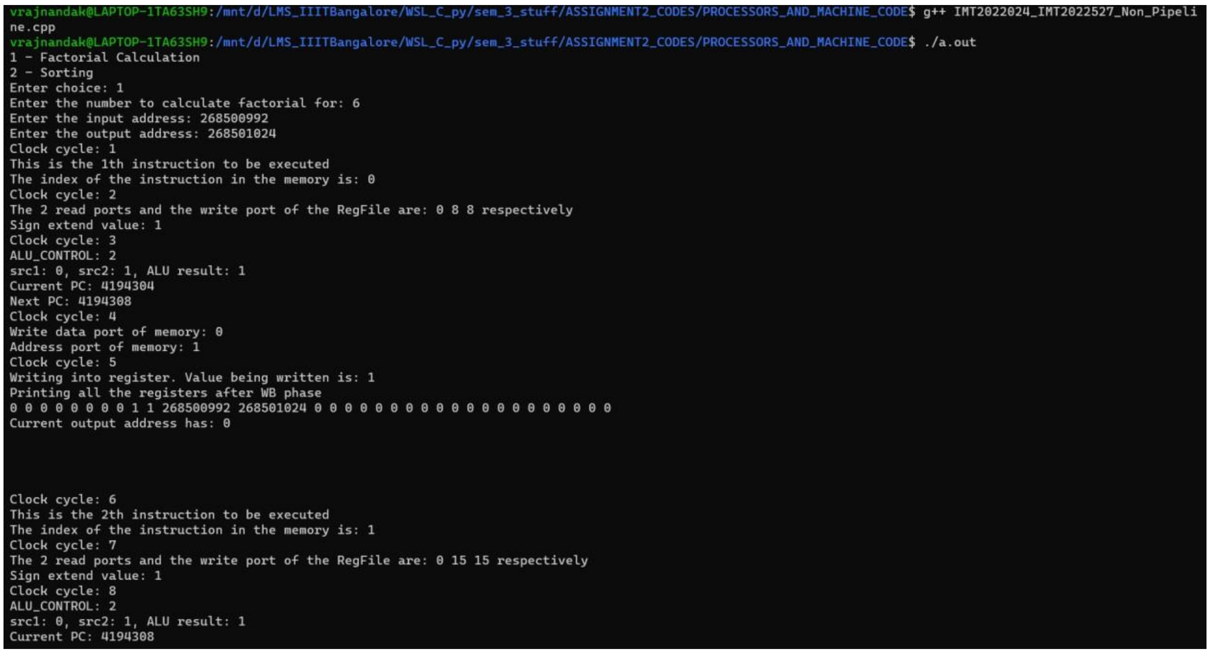
SCREENSHOTS OF OUTPUT:

Only part of the entire output is made visible for better understanding the output.

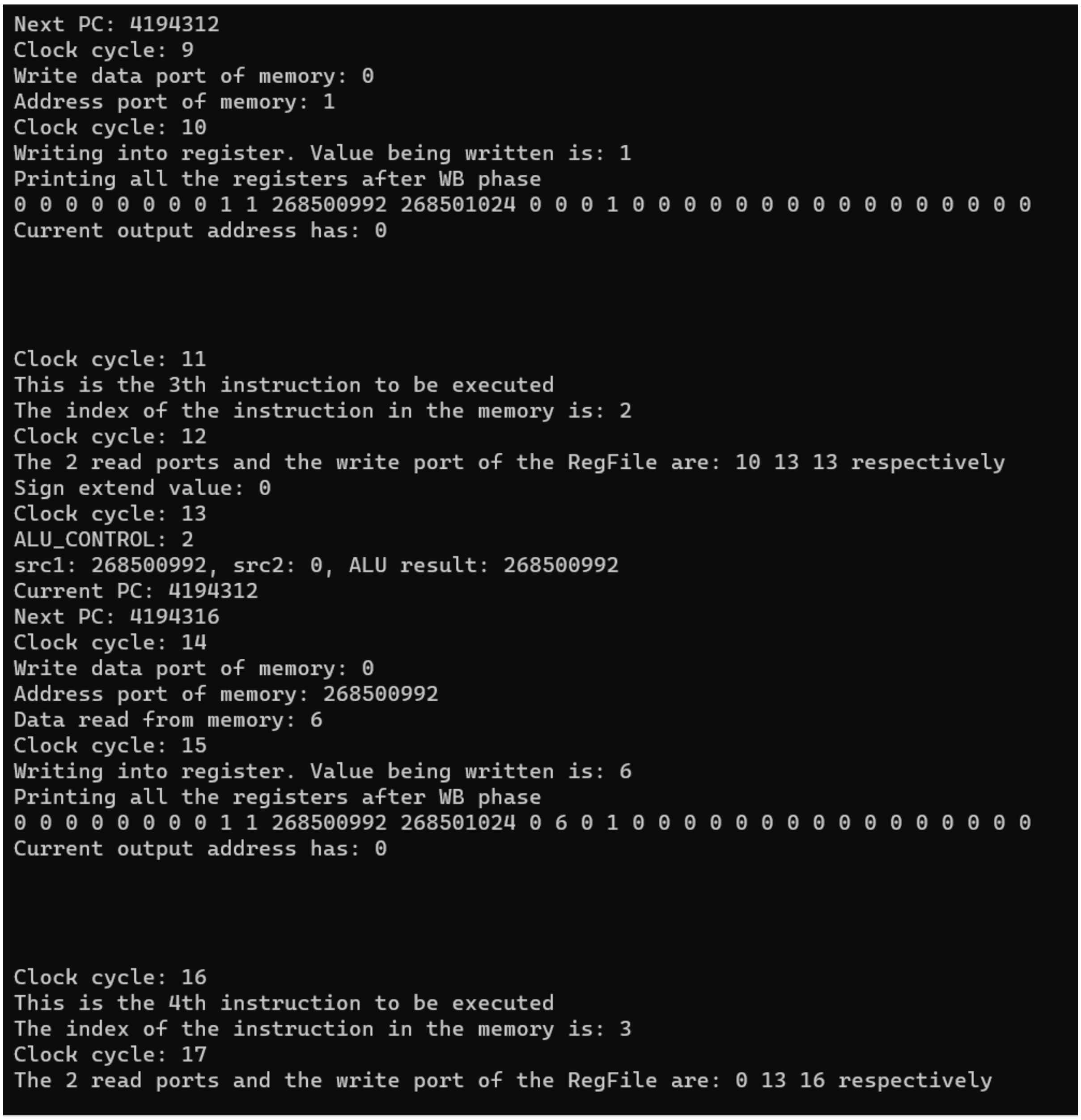
->Non-pipeline processor:

1. Factorial:

The first 2 screenshots contain the first few cycles of the output while the 3rd screenshot contains the output of the last few clock cycles.

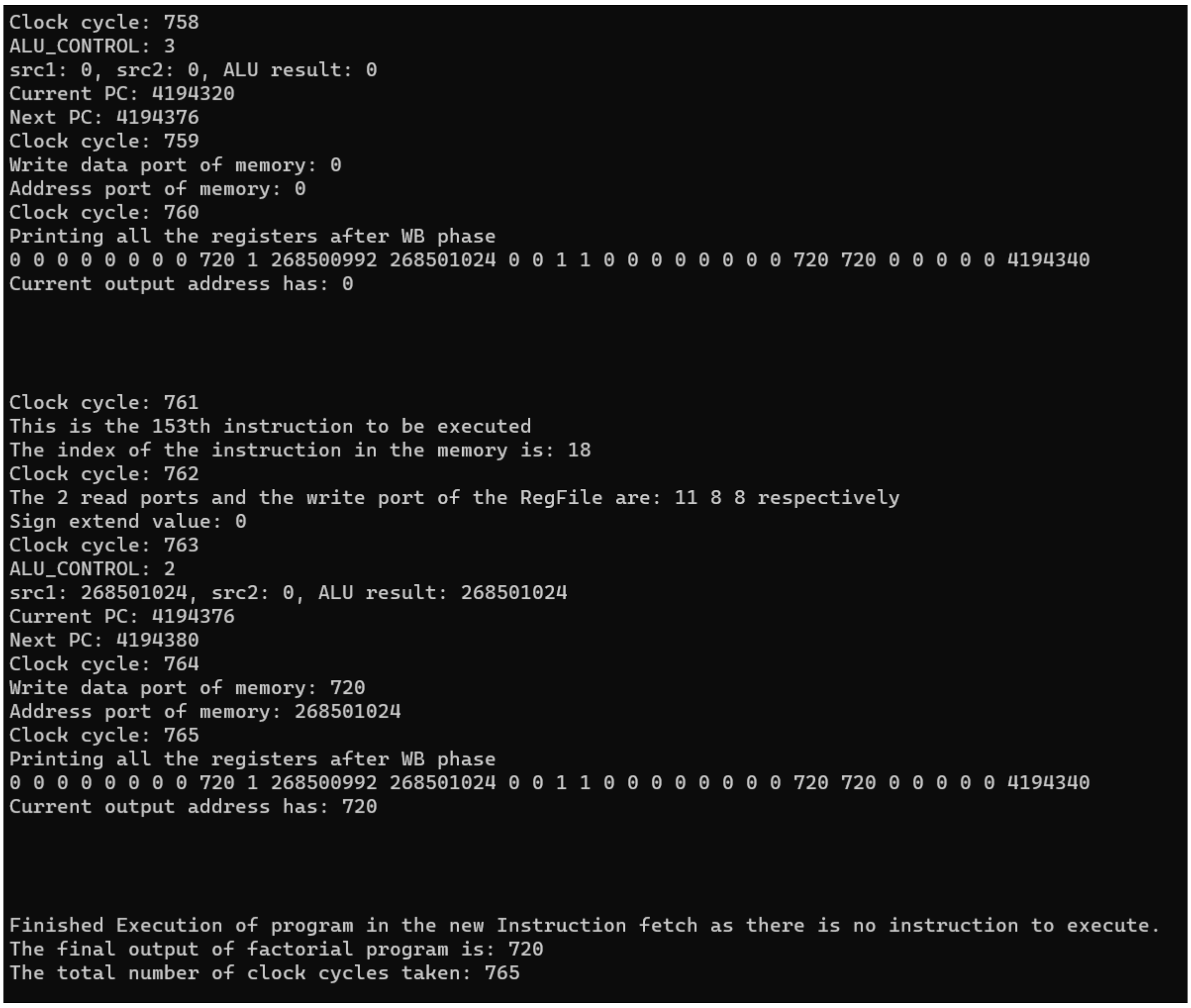


```
vrajnandak@LAPTOP-1TA63SH9:/mnt/d/LMS_IIITBangalore/WSL_C_py/sem_3_stuff/ASSIGNMENT2_CODES/PROCESSORS_AND_MACHINE_CODE$ g++ IMT2022024_IMT2022527_Non_Pipeli
ne.cpp
vrajnandak@LAPTOP-1TA63SH9:/mnt/d/LMS_IIITBangalore/WSL_C_py/sem_3_stuff/ASSIGNMENT2_CODES/PROCESSORS_AND_MACHINE_CODE$ ./a.out
1 - Factorial Calculation
2 - Sorting
Enter choice: 1
Enter the number to calculate factorial for: 6
Enter the input address: 268500992
Enter the output address: 268501024
Clock cycle: 1
This is the 1th instruction to be executed
The index of the instruction in the memory is: 0
Clock cycle: 2
The 2 read ports and the write port of the RegFile are: 0 8 8 respectively
Sign extend value: 1
Clock cycle: 3
ALU_CONTROL: 2
src1: 0, src2: 1, ALU result: 1
Current PC: 4194304
Next PC: 4194308
Clock cycle: 4
Write data port of memory: 0
Address port of memory: 1
Clock cycle: 5
Writing into register. Value being written is: 1
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 1 1 268500992 268501024 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Current output address has: 0




Clock cycle: 6
This is the 2th instruction to be executed
The index of the instruction in the memory is: 1
Clock cycle: 7
The 2 read ports and the write port of the RegFile are: 0 15 15 respectively
Sign extend value: 1
Clock cycle: 8
ALU_CONTROL: 2
src1: 0, src2: 1, ALU result: 1
Current PC: 4194308
```



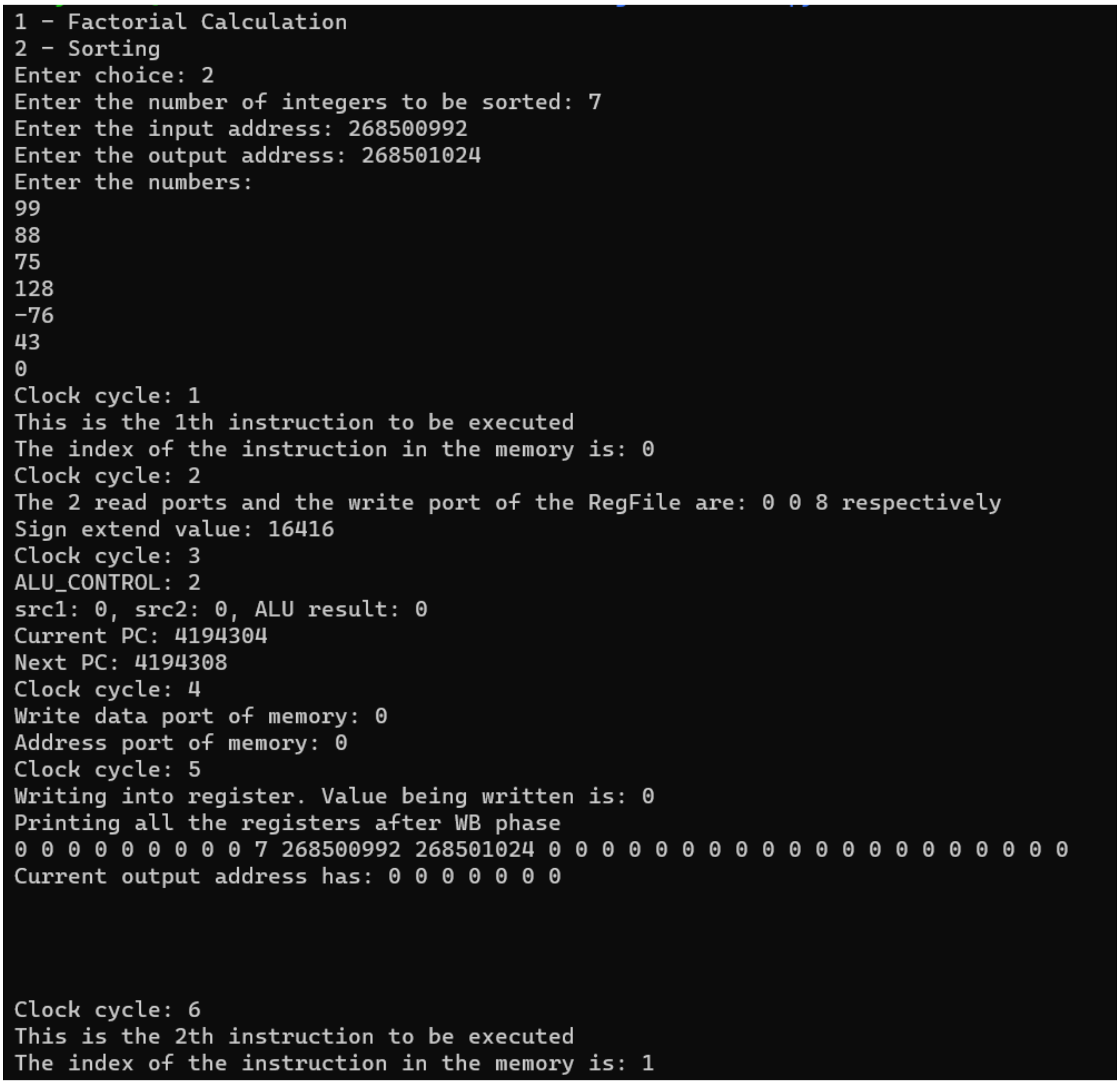
```
Next PC: 4194312
Clock cycle: 9
Write data port of memory: 0
Address port of memory: 1
Clock cycle: 10
Writing into register. Value being written is: 1
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 1 1 268500992 268501024 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Current output address has: 0




Clock cycle: 11
This is the 3th instruction to be executed
The index of the instruction in the memory is: 2
Clock cycle: 12
The 2 read ports and the write port of the RegFile are: 10 13 13 respectively
Sign extend value: 0
Clock cycle: 13
ALU_CONTROL: 2
src1: 268500992, src2: 0, ALU result: 268500992
Current PC: 4194312
Next PC: 4194316
Clock cycle: 14
Write data port of memory: 0
Address port of memory: 268500992
Data read from memory: 6
Clock cycle: 15
Writing into register. Value being written is: 6
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 1 1 268500992 268501024 0 6 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Current output address has: 0




Clock cycle: 16
This is the 4th instruction to be executed
The index of the instruction in the memory is: 3
Clock cycle: 17
The 2 read ports and the write port of the RegFile are: 0 13 16 respectively
```

The screenshot below contains the final output.

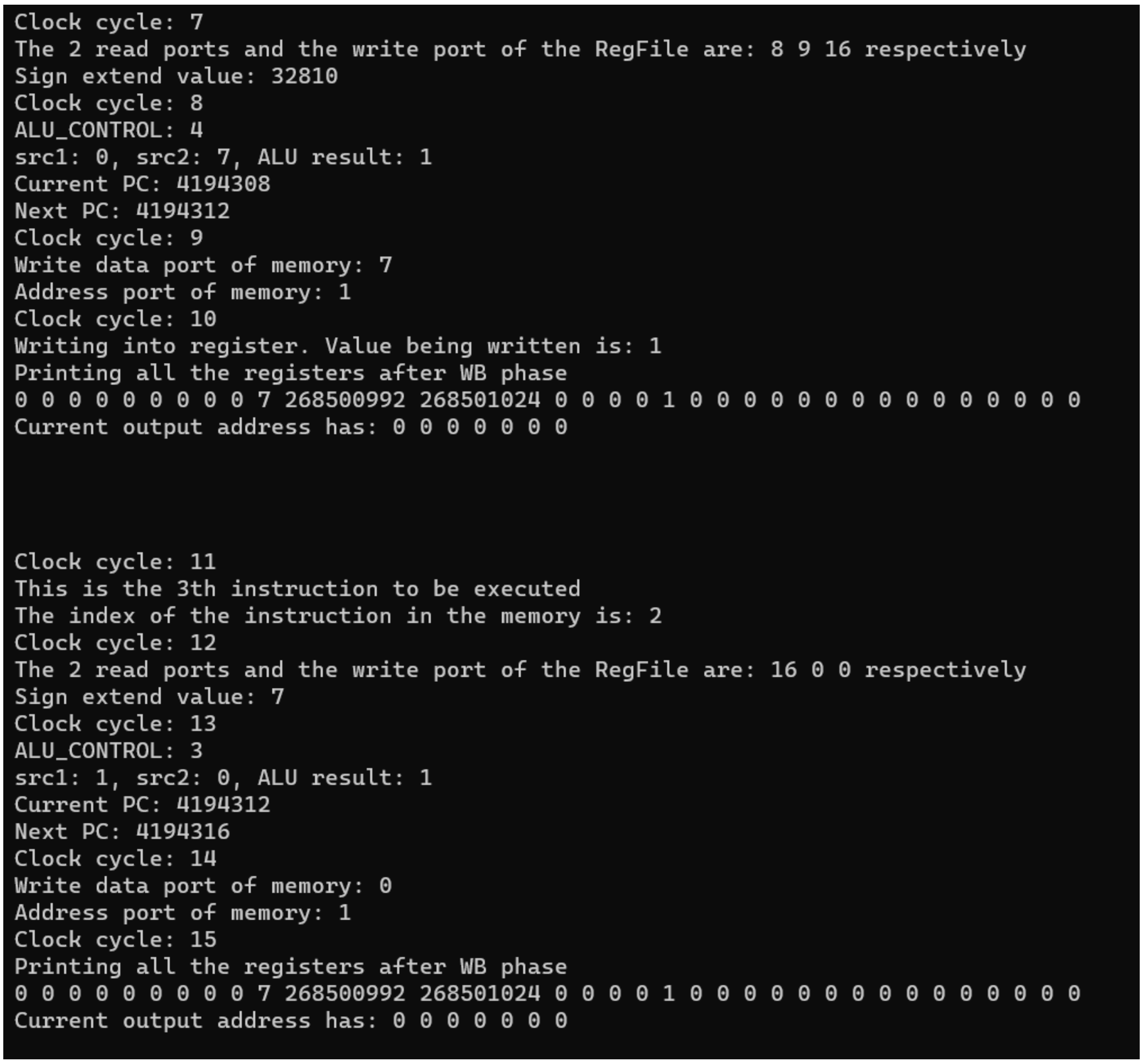

```
Clock cycle: 758
ALU_CONTROL: 3
src1: 0, src2: 0, ALU result: 0
Current PC: 4194320
Next PC: 4194376
Clock cycle: 759
Write data port of memory: 0
Address port of memory: 0
Clock cycle: 760
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 720 1 268500992 268501024 0 0 1 1 0 0 0 0 0 0 0 0 720 720 0 0 0 0 0 4194340
Current output address has: 0



Clock cycle: 761
This is the 153th instruction to be executed
The index of the instruction in the memory is: 18
Clock cycle: 762
The 2 read ports and the write port of the RegFile are: 11 8 8 respectively
Sign extend value: 0
Clock cycle: 763
ALU_CONTROL: 2
src1: 268501024, src2: 0, ALU result: 268501024
Current PC: 4194376
Next PC: 4194380
Clock cycle: 764
Write data port of memory: 720
Address port of memory: 268501024
Clock cycle: 765
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 720 1 268500992 268501024 0 0 1 1 0 0 0 0 0 0 0 0 720 720 0 0 0 0 0 4194340
Current output address has: 720



Finished Execution of program in the new Instruction fetch as there is no instruction to execute.
The final output of factorial program is: 720
The total number of clock cycles taken: 765
```

2. Sorting:

The first 2 screenshots contain the first few cycles of the output while the 3rd screenshot contains the output of the last few clock cycles.

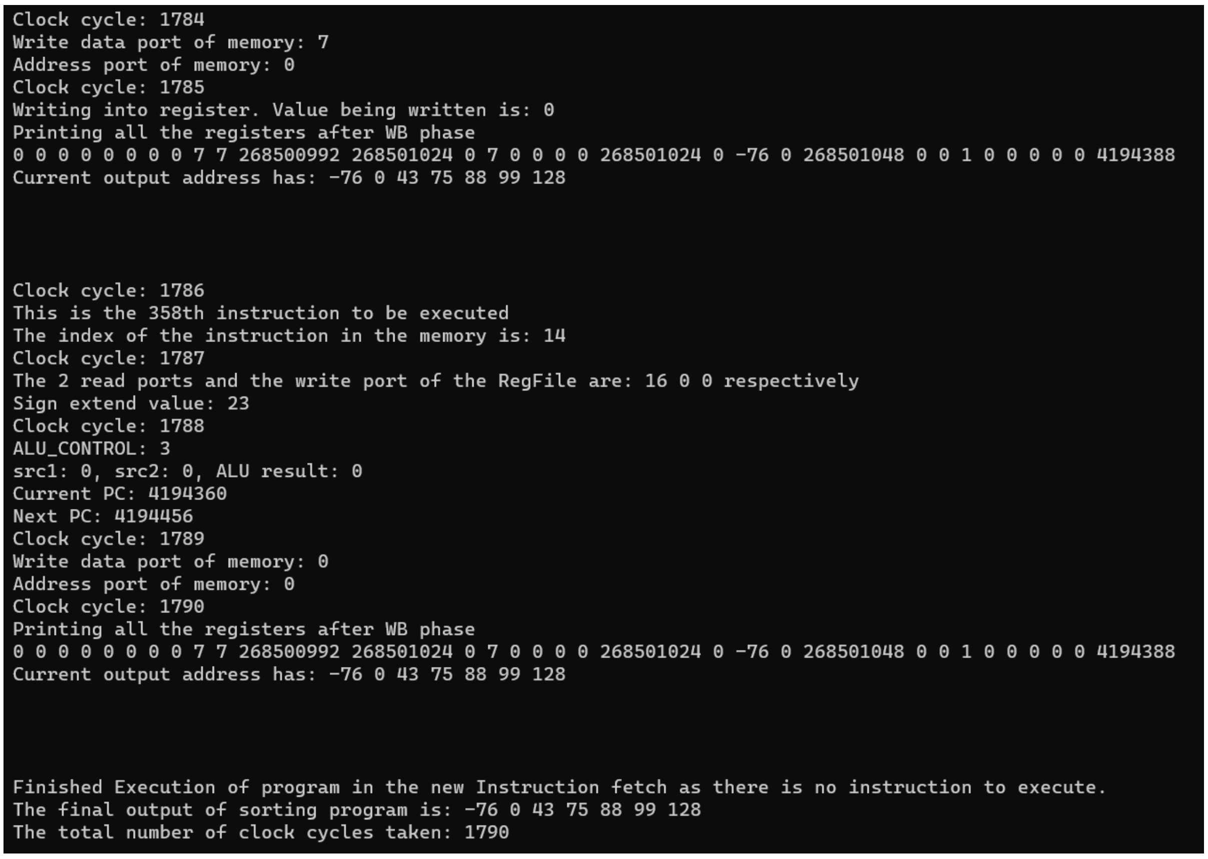


```
1 - Factorial Calculation
2 - Sorting
Enter choice: 2
Enter the number of integers to be sorted: 7
Enter the input address: 268500992
Enter the output address: 268501024
Enter the numbers:
99
88
75
128
-76
43
0
Clock cycle: 1
This is the 1th instruction to be executed
The index of the instruction in the memory is: 0
Clock cycle: 2
The 2 read ports and the write port of the RegFile are: 0 0 8 respectively
Sign extend value: 16416
Clock cycle: 3
ALU_CONTROL: 2
src1: 0, src2: 0, ALU result: 0
Current PC: 4194304
Next PC: 4194308
Clock cycle: 4
Write data port of memory: 0
Address port of memory: 0
Clock cycle: 5
Writing into register. Value being written is: 0
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 0 7 268500992 268501024 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Current output address has: 0 0 0 0 0 0 0




Clock cycle: 6
This is the 2th instruction to be executed
The index of the instruction in the memory is: 1
```



```
Clock cycle: 7
The 2 read ports and the write port of the RegFile are: 8 9 16 respectively
Sign extend value: 32810
Clock cycle: 8
ALU_CONTROL: 4
src1: 0, src2: 7, ALU result: 1
Current PC: 4194308
Next PC: 4194312
Clock cycle: 9
Write data port of memory: 7
Address port of memory: 1
Clock cycle: 10
Writing into register. Value being written is: 1
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 0 0 7 268500992 268501024 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Current output address has: 0 0 0 0 0 0 0




Clock cycle: 11
This is the 3th instruction to be executed
The index of the instruction in the memory is: 2
Clock cycle: 12
The 2 read ports and the write port of the RegFile are: 16 0 0 respectively
Sign extend value: 7
Clock cycle: 13
ALU_CONTROL: 3
src1: 1, src2: 0, ALU result: 1
Current PC: 4194312
Next PC: 4194316
Clock cycle: 14
Write data port of memory: 0
Address port of memory: 1
Clock cycle: 15
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 0 0 7 268500992 268501024 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Current output address has: 0 0 0 0 0 0 0
```

The screenshot below contains the final output.

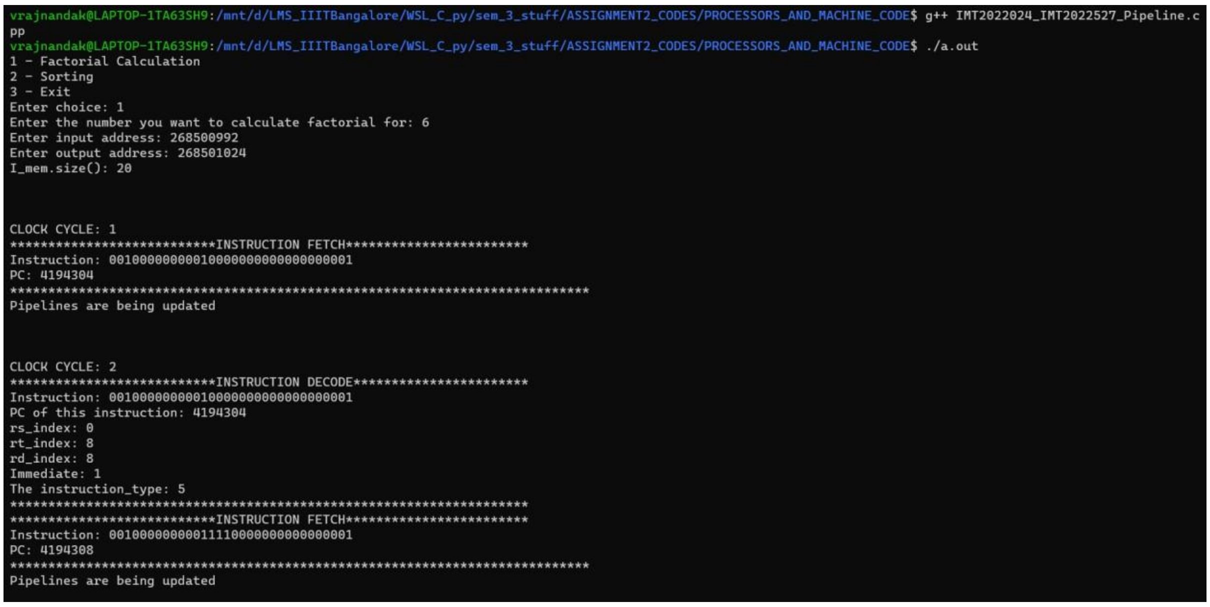


```
Clock cycle: 1784
Write data port of memory: 7
Address port of memory: 0
Clock cycle: 1785
Writing into register. Value being written is: 0
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 7 7 268500992 268501024 0 7 0 0 0 0 268501024 0 -76 0 268501048 0 0 1 0 0 0 0 0 4194388
Current output address has: -76 0 43 75 88 99 128




Clock cycle: 1786
This is the 358th instruction to be executed
The index of the instruction in the memory is: 14
Clock cycle: 1787
The 2 read ports and the write port of the RegFile are: 16 0 0 respectively
Sign extend value: 23
Clock cycle: 1788
ALU_CONTROL: 3
src1: 0, src2: 0, ALU result: 0
Current PC: 4194360
Next PC: 4194456
Clock cycle: 1789
Write data port of memory: 0
Address port of memory: 0
Clock cycle: 1790
Printing all the registers after WB phase
0 0 0 0 0 0 0 0 7 7 268500992 268501024 0 7 0 0 0 0 268501024 0 -76 0 268501048 0 0 1 0 0 0 0 0 4194388
Current output address has: -76 0 43 75 88 99 128




Finished Execution of program in the new Instruction fetch as there is no instruction to execute.
The final output of sorting program is: -76 0 43 75 88 99 128
The total number of clock cycles taken: 1790
```

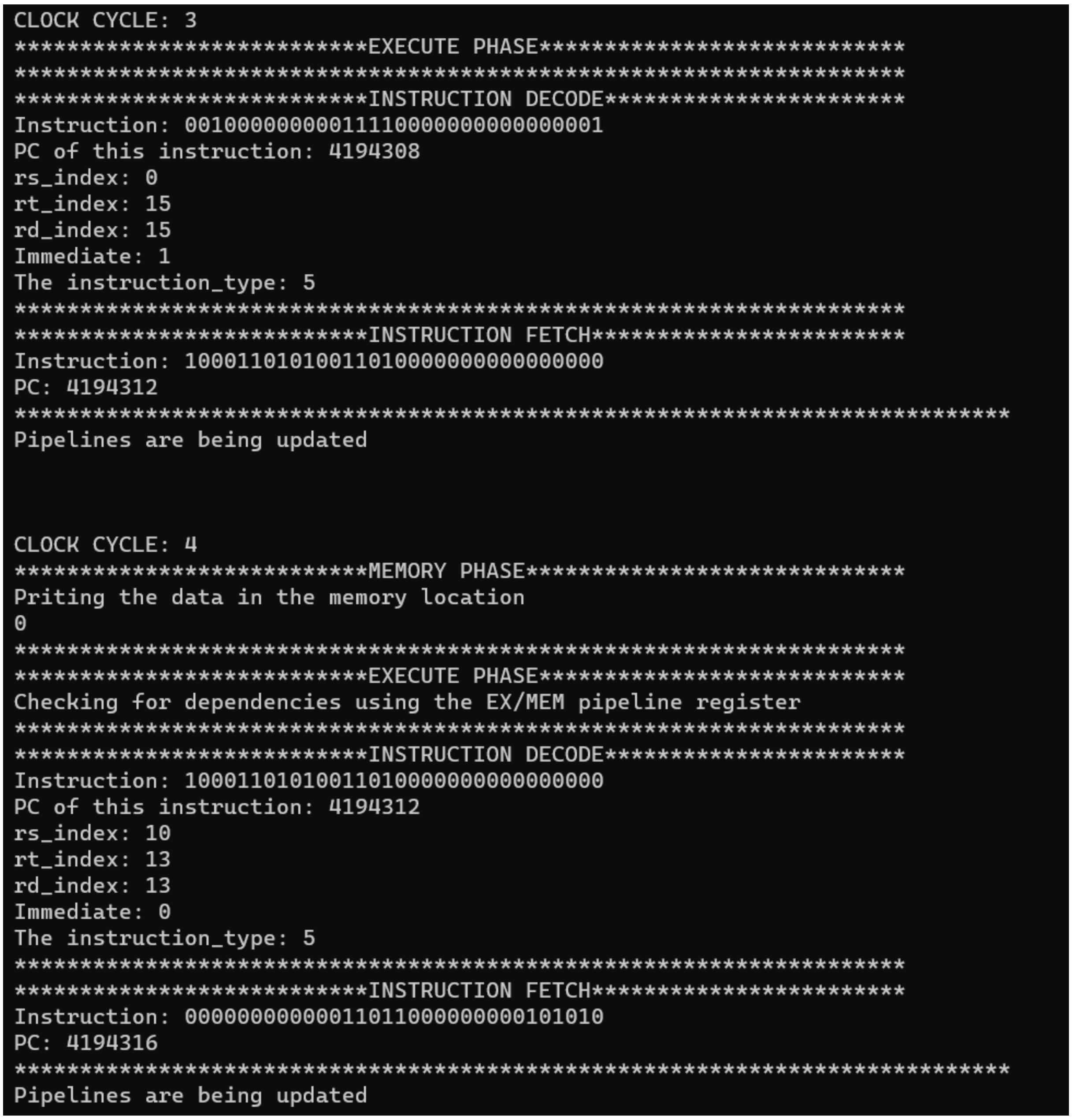
->Pipeline processor:

1. Factorial:

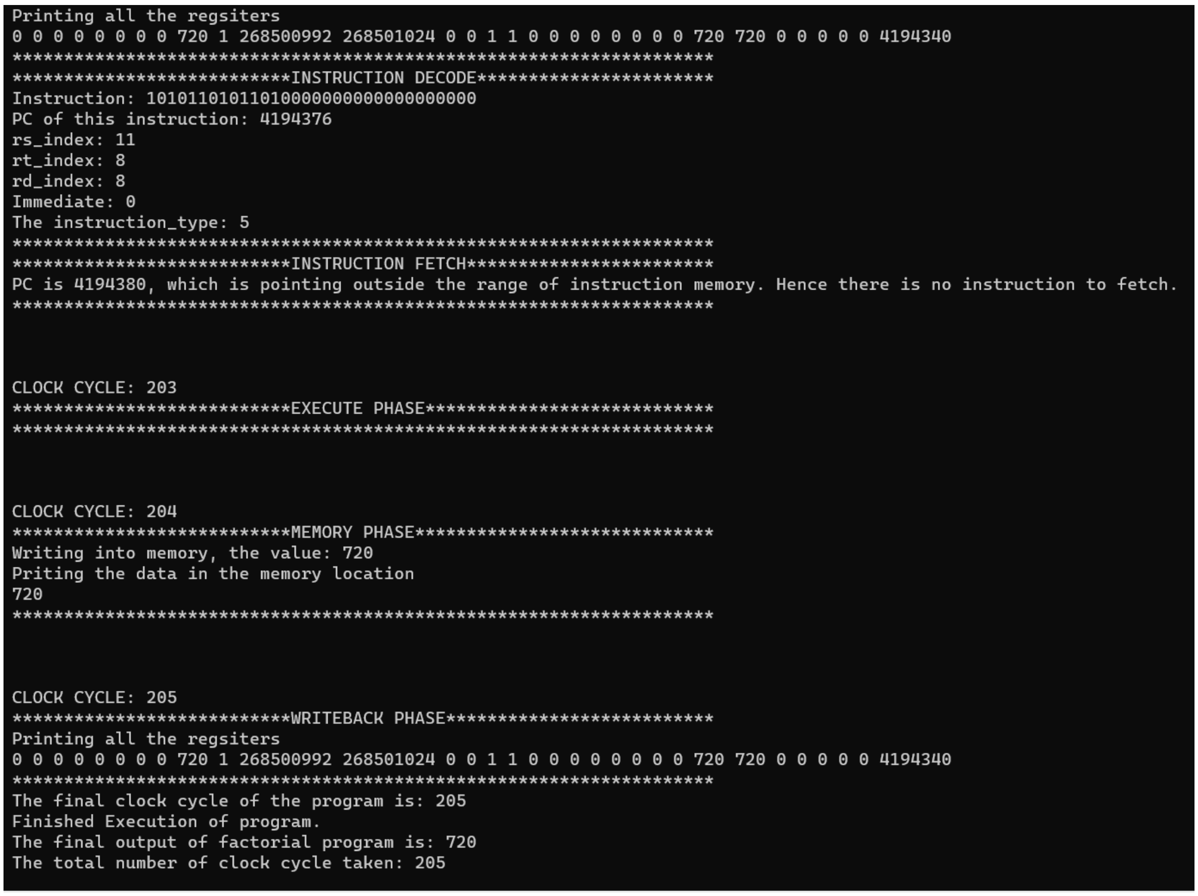
The first 2 screenshots contain the first few cycles of the output while the 3rd screenshot contains the output of the last few clock cycles.



```
vrajnandak@LAPTOP-1TA63SH9:/mnt/d/LMS_IIITBangalore/WSL_C_py/sem_3_stuff/ASSIGNMENT2_CODES/PROCESSORS_AND_MACHINE_CODE$ g++ IMT2022024_IMT2022527_Pipeline.c
pp
vrajnandak@LAPTOP-1TA63SH9:/mnt/d/LMS_IIITBangalore/WSL_C_py/sem_3_stuff/ASSIGNMENT2_CODES/PROCESSORS_AND_MACHINE_CODE$ ./a.out
1 - Factorial Calculation
2 - Sorting
3 - Exit
Enter choice: 1
Enter the number you want to calculate factorial for: 6
Enter input address: 268500992
Enter output address: 268501024
I_mem.size(): 20



CLOCK CYCLE: 1
***************************INSTRUCTION FETCH************************
Instruction: 00100000000010000000000000000001
PC: 4194304
****************************************************************************
Pipelines are being updated



CLOCK CYCLE: 2
***************************INSTRUCTION DECODE***********************
Instruction: 00100000000010000000000000000001
PC of this instruction: 4194304
rs_index: 0
rt_index: 8
rd_index: 8
Immediate: 1
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH************************
Instruction: 00100000000011110000000000000001
PC: 4194308
****************************************************************************
Pipelines are being updated
```


```
CLOCK CYCLE: 3
***************************EXECUTE PHASE****************************
********************************************************************
***************************INSTRUCTION DECODE***********************
Instruction: 00100000000011110000000000000001
PC of this instruction: 4194308
rs_index: 0
rt_index: 15
rd_index: 15
Immediate: 1
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH************************
Instruction: 10001101010011010000000000000000
PC: 4194312
****************************************************************************
Pipelines are being updated



CLOCK CYCLE: 4
***************************MEMORY PHASE*****************************
Priting the data in the memory location
0
********************************************************************
***************************EXECUTE PHASE****************************
Checking for dependencies using the EX/MEM pipeline register
********************************************************************
***************************INSTRUCTION DECODE***********************
Instruction: 10001101010011010000000000000000
PC of this instruction: 4194312
rs_index: 10
rt_index: 13
rd_index: 13
Immediate: 0
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH************************
Instruction: 00000000000011011000000000101010
PC: 4194316
****************************************************************************
Pipelines are being updated
```

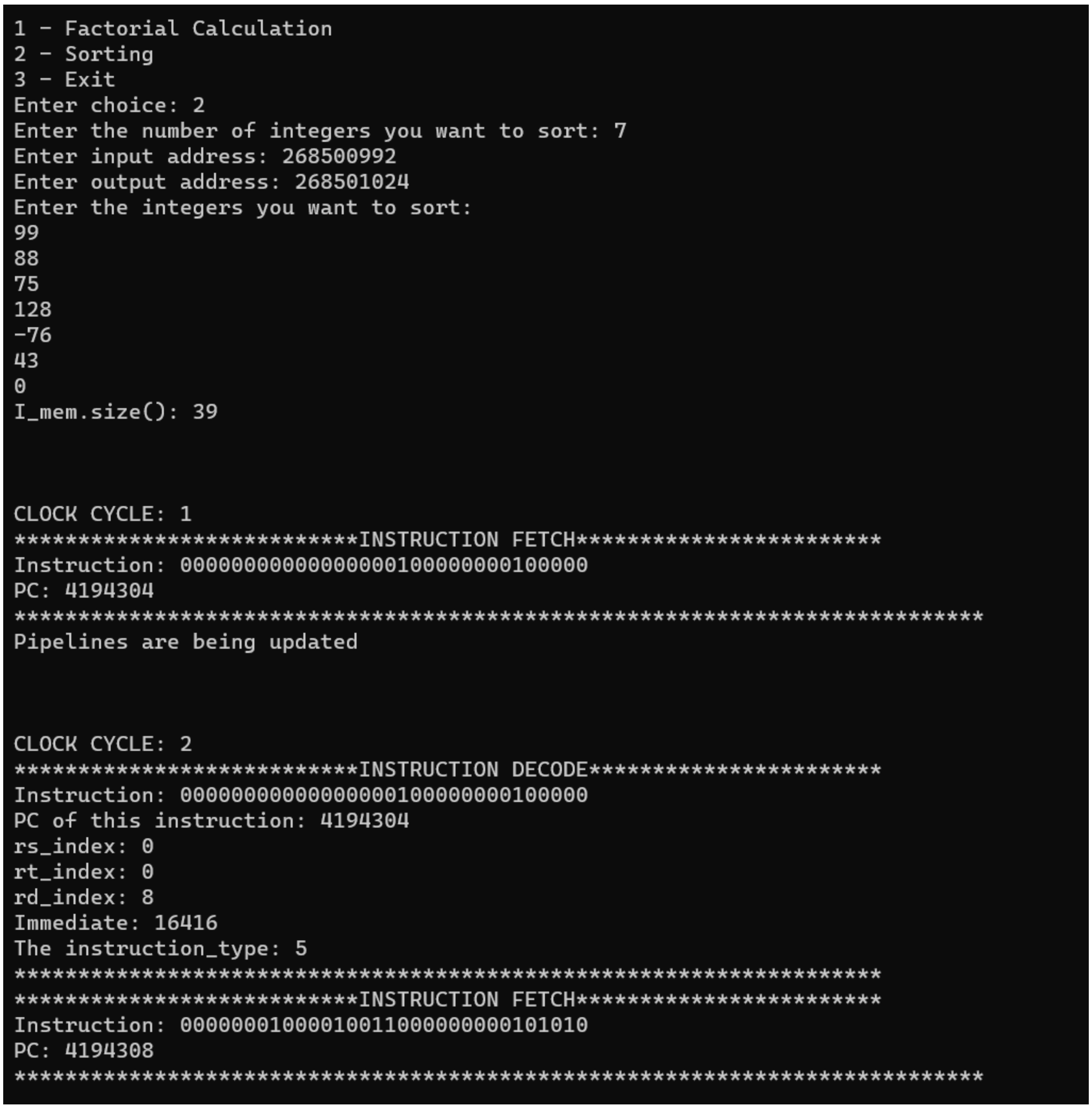
The below screenshot contains the output of the final few clock cycles.



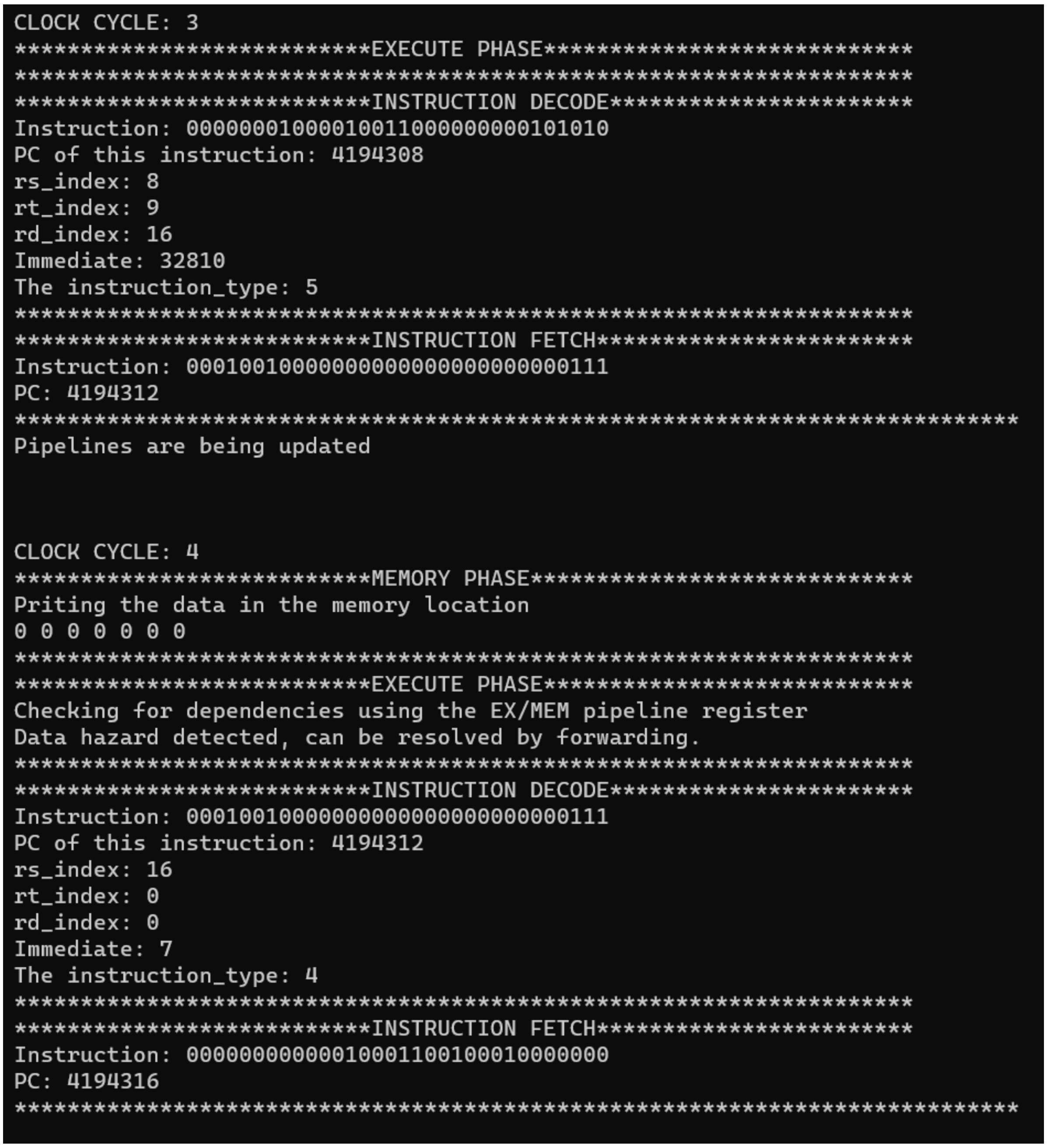
```
Printing all the regsiters
0 0 0 0 0 0 0 0 720 1 268500992 268501024 0 0 1 1 0 0 0 0 0 0 0 0 720 720 0 0 0 0 0 4194340
********************************************************************
***************************INSTRUCTION DECODE***********************
Instruction: 10101101011010000000000000000000
PC of this instruction: 4194376
rs_index: 11
rt_index: 8
rd_index: 8
Immediate: 0
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH************************
PC is 4194380, which is pointing outside the range of instruction memory. Hence there is no instruction to fetch.
********************************************************************



CLOCK CYCLE: 203
***************************EXECUTE PHASE****************************
********************************************************************



CLOCK CYCLE: 204
***************************MEMORY PHASE*****************************
Writing into memory, the value: 720
Priting the data in the memory location
720
********************************************************************



CLOCK CYCLE: 205
***************************WRITEBACK PHASE**************************
Printing all the regsiters
0 0 0 0 0 0 0 0 720 1 268500992 268501024 0 0 1 1 0 0 0 0 0 0 0 0 720 720 0 0 0 0 0 4194340
********************************************************************
The final clock cycle of the program is: 205
Finished Execution of program.
The final output of factorial program is: 720
The total number of clock cycle taken: 205
```

2. Sorting:

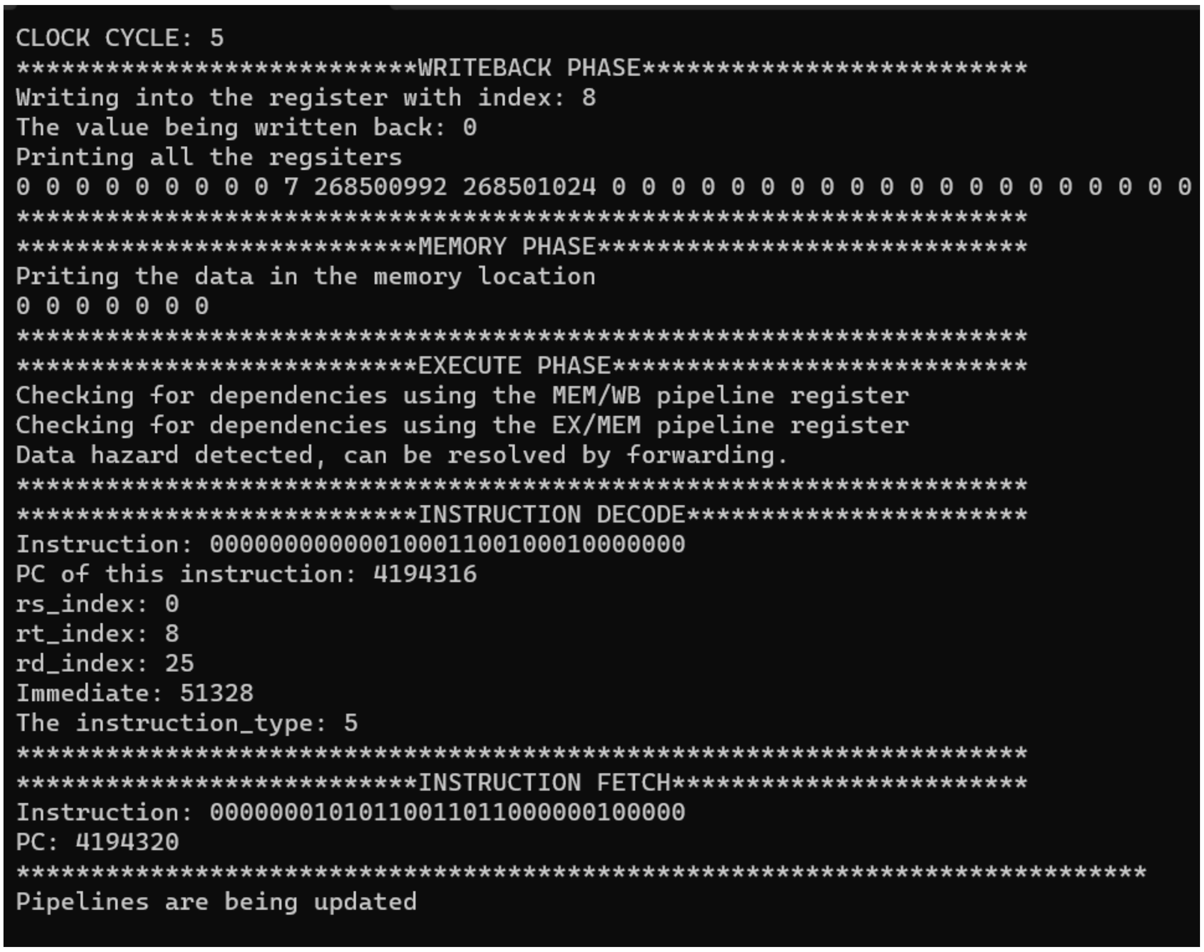
The first 3 screenshots contain the first few cycles of the output while the 4th screenshot contains the output of the last few clock cycles.



```
1 - Factorial Calculation
2 - Sorting
3 - Exit
Enter choice: 2
Enter the number of integers you want to sort: 7
Enter input address: 268500992
Enter output address: 268501024
Enter the integers you want to sort:
99
88
75
128
-76
43
0
I_mem.size(): 39



CLOCK CYCLE: 1
***************************INSTRUCTION FETCH***********************
Instruction: 00000000000000000100000000100000
PC: 4194304
****************************************************************************
Pipelines are being updated



CLOCK CYCLE: 2
***************************INSTRUCTION DECODE***********************
Instruction: 00000000000000000100000000100000
PC of this instruction: 4194304
rs_index: 0
rt_index: 0
rd_index: 8
Immediate: 16416
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH***********************
Instruction: 00000001000010011000000000101010
PC: 4194308
****************************************************************************
```

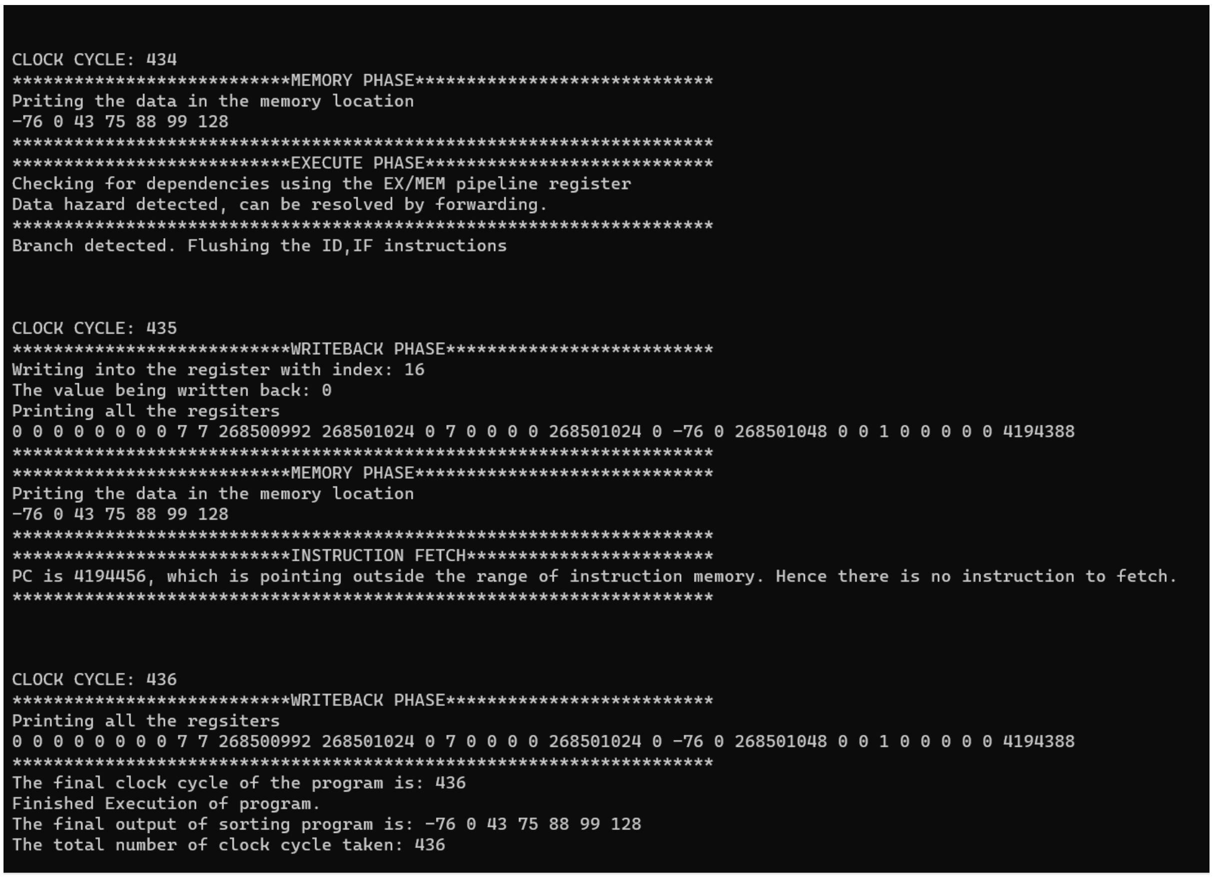


```
CLOCK CYCLE: 3
***************************EXECUTE PHASE****************************
********************************************************************
***************************INSTRUCTION DECODE***********************
Instruction: 00000001000010011000000000101010
PC of this instruction: 4194308
rs_index: 8
rt_index: 9
rd_index: 16
Immediate: 32810
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH************************
Instruction: 00010010000000000000000000000111
PC: 4194312
****************************************************************************
Pipelines are being updated



CLOCK CYCLE: 4
***************************MEMORY PHASE*****************************
Priting the data in the memory location
0 0 0 0 0 0 0
********************************************************************
***************************EXECUTE PHASE****************************
Checking for dependencies using the EX/MEM pipeline register
Data hazard detected, can be resolved by forwarding.
********************************************************************
***************************INSTRUCTION DECODE***********************
Instruction: 00010010000000000000000000000111
PC of this instruction: 4194312
rs_index: 16
rt_index: 0
rd_index: 0
Immediate: 7
The instruction_type: 4
********************************************************************
***************************INSTRUCTION FETCH************************
Instruction: 00000000000100011001000010000000
PC: 4194316
****************************************************************************
```



```
CLOCK CYCLE: 5
***************************WRITEBACK PHASE**************************
Writing into the register with index: 8
The value being written back: 0
Printing all the regsiters
0 0 0 0 0 0 0 0 0 7 268500992 268501024 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
********************************************************************
***************************MEMORY PHASE*****************************
Priting the data in the memory location
0 0 0 0 0 0 0
********************************************************************
***************************EXECUTE PHASE****************************
Checking for dependencies using the MEM/WB pipeline register
Checking for dependencies using the EX/MEM pipeline register
Data hazard detected, can be resolved by forwarding.
********************************************************************
***************************INSTRUCTION DECODE***********************
Instruction: 00000000000010001100100010000000
PC of this instruction: 4194316
rs_index: 0
rt_index: 8
rd_index: 25
Immediate: 51328
The instruction_type: 5
********************************************************************
***************************INSTRUCTION FETCH************************
Instruction: 00000001010110011011000000100000
PC: 4194320
****************************************************************************
Pipelines are being updated
```

The screenshot below contains the output of the final few cycles.



```
CLOCK CYCLE: 434
***************************MEMORY PHASE*****************************
Priting the data in the memory location
-76 0 43 75 88 99 128
********************************************************************
***************************EXECUTE PHASE****************************
Checking for dependencies using the EX/MEM pipeline register
Data hazard detected, can be resolved by forwarding.
********************************************************************
Branch detected. Flushing the ID,IF instructions


CLOCK CYCLE: 435
***************************WRITEBACK PHASE**************************
Writing into the register with index: 16
The value being written back: 0
Printing all the regsiters
0 0 0 0 0 0 0 0 7 7 268500992 268501024 0 7 0 0 0 0 268501024 0 -76 0 268501048 0 0 1 0 0 0 0 0 4194388
********************************************************************
***************************MEMORY PHASE*****************************
Priting the data in the memory location
-76 0 43 75 88 99 128
********************************************************************
***************************INSTRUCTION FETCH************************
PC is 4194456, which is pointing outside the range of instruction memory. Hence there is no instruction to fetch.
********************************************************************


CLOCK CYCLE: 436
***************************WRITEBACK PHASE**************************
Printing all the regsiters
0 0 0 0 0 0 0 0 7 7 268500992 268501024 0 7 0 0 0 0 268501024 0 -76 0 268501048 0 0 1 0 0 0 0 0 4194388
********************************************************************
The final clock cycle of the program is: 436
Finished Execution of program.
The final output of sorting program is: -76 0 43 75 88 99 128
The total number of clock cycle taken: 436
```

It is clear from the output of the programs for the non-pipeline processor and that of the pipeline processor that the total number of clock cycles in the execution of a program in a pipeline processor is lesser than that for a pipeline processor. From the screenshots it is clear that the pipeline processor has approximately 3 to 4 times lesser number of clock cycles (although this does depend on the number of stalls used, flushes performed) than that of the non-pipeline processor. Thus we can conclude that in general the pipeline processor is faster than a non-pipeline processor.