Computer Architecture (Assignment-2)

Madhav Girdhar (IMT2022009)
Bhavya Kapadia (IMT2022095)

November 2023

NonPipeline and Pipeline Processors

Introduction

A simple Java processor which decodes your assembly instructions and do as instructed by the set of instructions provided by the user. In the following program, we are using a factorial and fibonacci sequence in which our code or processor gives you the factorial of one and multiple numbers at a time, and for fibonacci, it will give you the n^{th} fibo number similarly for single as well as multiple input.

Instructions for the running of the program

1. Select which program you want to run.
   (a) Press 0 for Factorial.
   (b) Press 1 for Fibonacci.
2. Give the no. of input for the program.
3. Enter the base address for input which is a multiple of four.
4. Give base address for output which is also a multiple of four and also ensure that it will greater than (base address of input + no. of input * 4) for the better result and also for avoid overwrite condition in memory.
5. Now Enter the integer one by one and wait for the result.

Explanation and Working of Code

1. NonPipeline Processor

1. We had made a different class based on their instruction type, whether it is a rtype, itype, or jtype instruction.
2. We had created a different instruction and data memory which store the instruction and data respectively in the multiple of four.

3. Now in the main we made a pc(program counter) which will increment by four after every cycle and help us to fetch instruction from instruction memory.

4. During instruction fetching it will detect that which type of instruction is this and make object accordingly so that we can start different phase of instruction like decode,execution and so on.

5. Now during decode it will take values of register from register file and send that data for execution.

6. Now in execution phase it will detect what to do with help of function field and start work accordingly.

7. Then there is Memory phase for load/Store type of instruction which load and store data from/in data memory.

8. Now in the write back phase it store the result in output registers and make change/update in register file.

9. Now the pc will increment by four for the fetching of next instruction and clock cycle increased by five because in non-pipeline the next instruction is fetched after every 5 stage of previous instruction.

2. Pipeline Processor

1. Here also the different types of classes are formed based on the type of instruction.

2. Here, instructions are fetched in every clock cycle unless there is a stall or any branch flushes.

3. As we are fetching instructions in every clock cycle, during the fetching of the second instruction, the decode of the first instruction is going on, and similarly, during the fetching of the third instruction, the execution of the first instruction is going on, and the decode of the second instruction is happening.

4. Here in this we made a Pipline register which store the register along with their value,available clock cycle as well as their write back clock cycle so that we can manage the forwarding or can check whether the forwarding needed or not as according to their write back clock cycles.

5. So for the above point's we made 4 different variable which will handle which instruction decode and which instruction execution will happen next same thing for memory and write back phase as well.

6. Here the pc will increment at the end of every clock cycle and clock cycle will increment after every fetch.

7. Basically here there different phases of different instruction are going on in a single clock cycle so it will take less time/clock cycle as compared to Nonpipline processor.

How we Handle Hazard

1. Structural Hazard

We manage this hazard by putting dummy instructions between two dependent instructions if there is a stall in forwarding. For example, in a load and add instruction, there is a dependency but also a stall between them.

IF $\rightarrow ID \rightarrow Ex \rightarrow Mem \rightarrow Wb$
"" $\rightarrow IF \rightarrow ID \rightarrow stall \rightarrow Ex \rightarrow Mem \rightarrow Wb$

As here we need a stall for forward data from Mem of 1st instruction to Ex of second instruction so we manage these type of stall which create structural hazard for next instruction by adding dummy instruction between them.

2. Data Hazard

We managed the data hazard by the help of forwarding the data from previous instruction to next instruction from pipeline register as they are storing the different register values,availability at which clock cycle,their write back clock cycle so that by the help of this we can check whether we can access this register value or not by if else condition in the different classes of our program.

3. Control Hazard

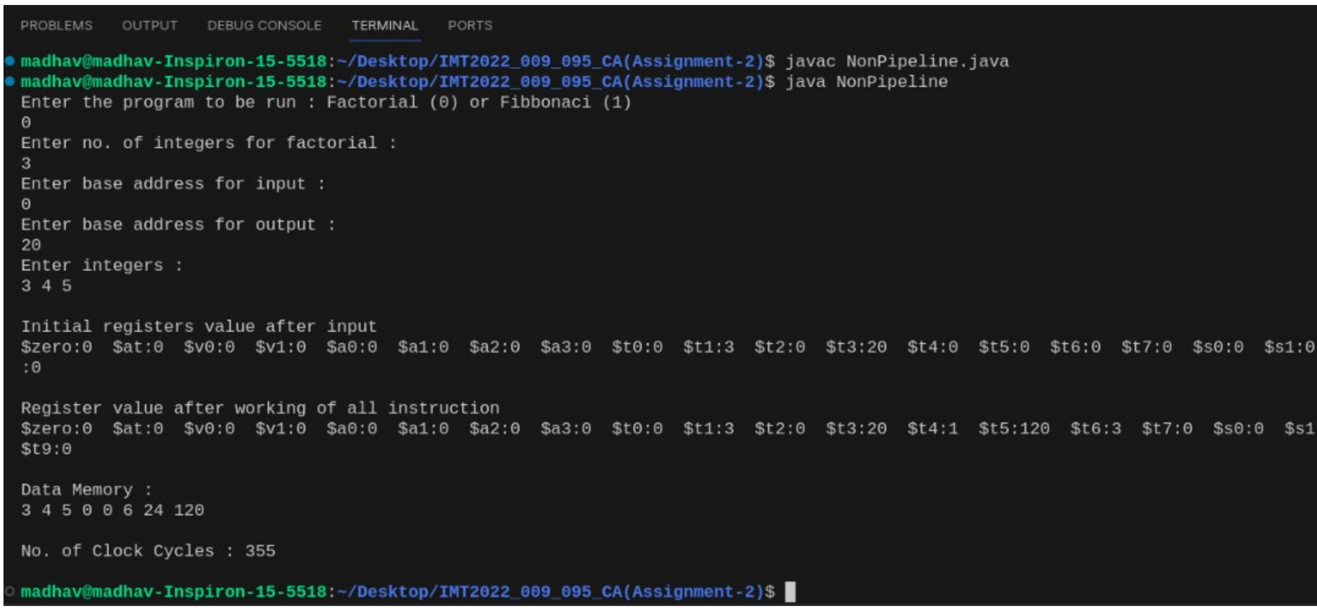
We managed this hazard which occur mainly during the flushing of pipeline when there is an branch instruction or jump instruction. So we managed the control hazard by checking the values of register in fetched stage itself so that our pc got it's values and no other instruction is fetched during that time so no need to flush the pipeline and also it increase the latency of the program.

Final Result's

Here are some examples of output for different code fibo/factorial for pipline as well as non-pipline processors in the images attached below. From this, you can learn how to compile and run the code properly.

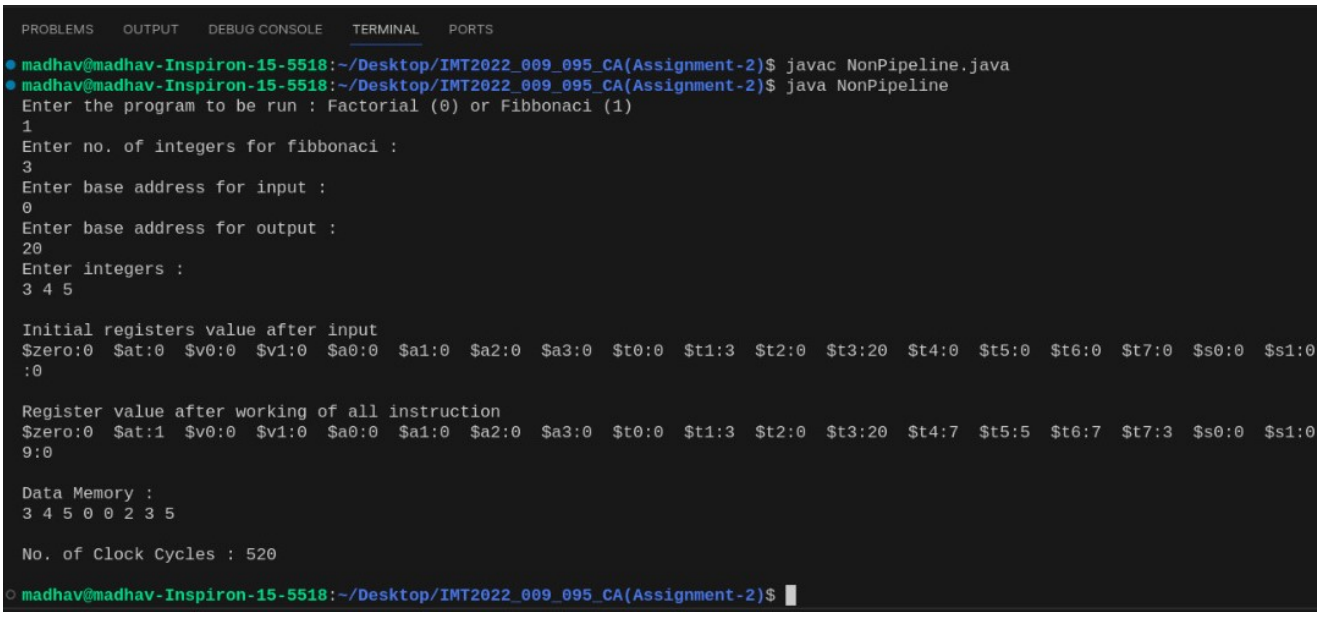
1. NonPipeline

(a) Factorial :



```
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ javac NonPipeline.java
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ java NonPipeline
Enter the program to be run : Factorial (0) or Fibbonaci (1)
0
Enter no. of integers for factorial :
3
Enter base address for input :
0
Enter base address for output :
20
Enter integers :
3 4 5

Initial registers value after input
$zero:0  $at:0  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:0  $t5:0  $t6:0  $t7:0  $s0:0  $s1:0
:0

Register value after working of all instruction
$zero:0  $at:0  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:1  $t5:120  $t6:3  $t7:0  $s0:0  $s1
$t9:0

Data Memory :
3 4 5 0 0 6 24 120

No. of Clock Cycles : 355

madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ █
```

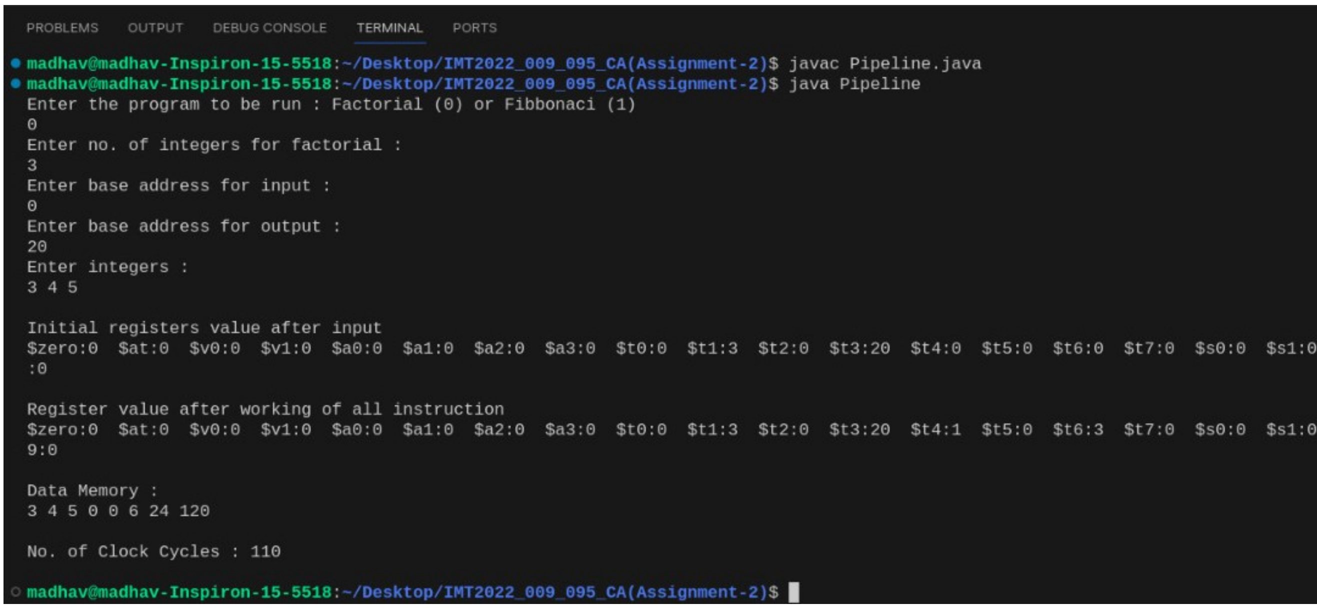
(b) Fibonacci :



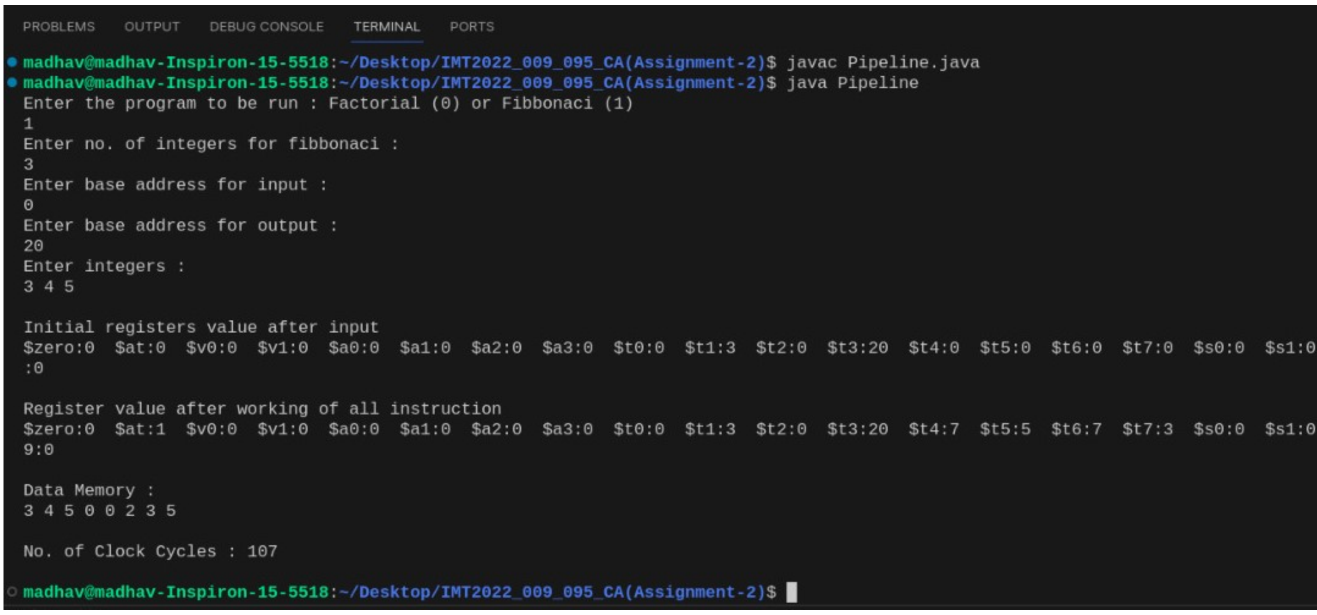
```
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ javac NonPipeline.java
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ java NonPipeline
Enter the program to be run : Factorial (0) or Fibbonaci (1)
1
Enter no. of integers for fibbonaci :
3
Enter base address for input :
0
Enter base address for output :
20
Enter integers :
3 4 5

Initial registers value after input
$zero:0  $at:0  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:0  $t5:0  $t6:0  $t7:0  $s0:0  $s1:0
:0

Register value after working of all instruction
$zero:0  $at:1  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:7  $t5:5  $t6:7  $t7:3  $s0:0  $s1:0
9:0

Data Memory :
3 4 5 0 0 2 3 5

No. of Clock Cycles : 520

madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ █
```

2. Pipeline

(a) Factorial :


```
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ javac Pipeline.java
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ java Pipeline
Enter the program to be run : Factorial (0) or Fibbonaci (1)
0
Enter no. of integers for factorial :
3
Enter base address for input :
0
Enter base address for output :
20
Enter integers :
3 4 5

Initial registers value after input
$zero:0  $at:0  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:0  $t5:0  $t6:0  $t7:0  $s0:0  $s1:0
:0

Register value after working of all instruction
$zero:0  $at:0  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:1  $t5:0  $t6:3  $t7:0  $s0:0  $s1:0
9:0

Data Memory :
3 4 5 0 0 6 24 120

No. of Clock Cycles : 110

madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$
```

(b) Fibonacci :


```
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ javac Pipeline.java
madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$ java Pipeline
Enter the program to be run : Factorial (0) or Fibbonaci (1)
1
Enter no. of integers for fibbonaci :
3
Enter base address for input :
0
Enter base address for output :
20
Enter integers :
3 4 5

Initial registers value after input
$zero:0  $at:0  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:0  $t5:0  $t6:0  $t7:0  $s0:0  $s1:0
:0

Register value after working of all instruction
$zero:0  $at:1  $v0:0  $v1:0  $a0:0  $a1:0  $a2:0  $a3:0  $t0:0  $t1:3  $t2:0  $t3:20  $t4:7  $t5:5  $t6:7  $t7:3  $s0:0  $s1:0
9:0

Data Memory :
3 4 5 0 0 2 3 5

No. of Clock Cycles : 107

madhav@madhav-Inspiron-15-5518:~/Desktop/IMT2022_009_095_CA(Assignment-2)$
```