ASSIGNMENT – 2

Group members:

1. Shashank Devarmani (IMT2022107)

2. Kandarp Dave (IMT2022115)

Explanation of programs:

1. IMT2022_107_115_fibonacci.asm

In this program, we ask the user to give 2 inputs:

i. Number of Fibonacci numbers to be calculated (say N)

ii. Memory address (in decimal) from where the numbers need to be stored.

Then, we run a simple loop to calculate the first N Fibonacci numbers and store each of them at a memory location.

2. IMT2022_107_115_factorial.asm

In this program, we take 1 input:

i. Memory address (in decimal) from where the factorials need to be stored.

Our aim is to calculate the factorials of numbers 1 to 10, and store them in memory. If we need to find n!, then we load (n – 1)! from the memory and multiply n to it. This is achieved through a loop.

Explanation of codes:

1. **IMT2022_107_115_nonpipeline.py**

   Program Selection:

   The user is presented with a menu to choose between two programs: calculating Fibonacci numbers or factorials of the first 10 numbers.

   The user's choice is read from the standard input.

   Program Initialization:

   Depending on the user's choice, the program initializes various parameters and loads machine code instructions from text files. The program's execution is determined by the choice made.

Dictionary Initialization:

Three dictionaries are initialized:

a. InstructionMemory: A mapping of memory addresses to machine code instructions.
b. RegisterFile: A mapping of register names to their initial values.
c. DataMemory: A mapping of memory addresses to data values.

Binary to Decimal Conversion:

A utility function, binaryToDecimal, is defined to convert binary strings to decimal numbers.

Execution Loop:

The program enters a loop where it simulates the execution of instructions.

It processes each instruction step by step, following the fetch-decode-execute cycle for each instruction.

Fetch Stage:

The instruction fetch stage retrieves the next instruction from memory and increments the program counter (PC).

The current instruction is printed, and the clock cycle count is incremented.

Decode Stage:

The decode stage deciphers the opcode and other relevant fields from the instruction.

It determines the type of instruction (R, I, or J format) and identifies the specific operation to perform.

ALU Stage:

The Arithmetic Logic Unit (ALU) stage performs the actual calculations based on the decoded instruction.

The result is stored in the ALUOutput variable.

Memory Access Stage:

In this stage, data memory is accessed for load and store operations (lw and sw).

Data is read from or written to the data memory, and the values are printed.

Register Writeback Stage:

The register writeback stage updates registers based on the results of the ALU stage.

Registers are updated with the calculated values, as needed.

Clock Cycle Count:

The clock cycle count is incremented after each instruction is processed, and its value is printed for each instruction.

Loop Continues:

The loop continues until the program counter (PC) reaches the maximum possible value (PCMax).

End of Execution:

After the loop concludes, the program prints the total number of clock cycles used during execution.

Register and Data Memory Values:

The program prints the final values of the registers and data memory.

1. Fibonacci program output

```
shashank@DESKTOP-S26019G:/mnt/c/Users/Admin/Programming files/CA Processor pipelining/CA assignment final$ python3 non_pipeline1.py
1. Fibonacci Numbers
2. Factorials of first 10 numbers
Enter appropriate number: 1
Enter starting address of data memory: 268501216
Enter number of fibonacci numbers to be calculated: 4
PCMax =  4194384
currInstruction =  00000000000000001000000000100000

instructionToPerform =  add
opcode =  000000
ALUOutput =  0
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  s0
dataToWriteToRegister =  0
clock cycles =  5
PC =  4194332
currInstruction =  00100000000100010000000000000001

instructionToPerform =  addi
opcode =  001000
ALUOutput =  1
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  s1
dataToWriteToRegister =  1
clock cycles =  10
PC =  4194336
currInstruction =  10101101010100000000000000000000

instructionToPerform =  sw
opcode =  101011
ALUOutput =  268501216
dataMemoryOutput =  Garbage value
```

```
instructionToPerform =  sw
opcode =  101011
ALUOutput =  268501220
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  1
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  20
PC =  4194344
currInstruction =  00100001010010110000000000001000

instructionToPerform =  addi
opcode =  001000
ALUOutput =  268501224
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t3
dataToWriteToRegister =  268501224
clock cycles =  25
PC =  4194348
currInstruction =  00100000000011000000000000000010

instructionToPerform =  addi
opcode =  001000
ALUOutput =  2
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t4
dataToWriteToRegister =  2
clock cycles =  30
PC =  4194352
currInstruction =  00010001100010010000000000000111
```

```
instructionToPerform =  beq
opcode =  000100
ALUOutput =  -2
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  35
PC =  4194356
currInstruction =  00000010000100011001000000100000

instructionToPerform =  add
opcode =  000000
ALUOutput =  1
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  s2
dataToWriteToRegister =  1
clock cycles =  40
PC =  4194360
currInstruction =  10101101011100100000000000000000

instructionToPerform =  sw
opcode =  101011
ALUOutput =  268501224
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  1
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  45
PC =  4194364
currInstruction =  00000010001000001000000000100000
```

```
currInstruction =  00100001011010110000000000000100

instructionToPerform =  addi
opcode =  001000
ALUOutput =  268501232
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t3
dataToWriteToRegister =  268501232
clock cycles =  105
PC =  4194380
currInstruction =  00001000000100000000000000001100
instructionToPerform =  jump
opcode =  000010
ALUOutput =  Garbage value
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  110
PC =  4194352
currInstruction =  00010001100010010000000000000111

instructionToPerform =  beq
opcode =  000100
ALUOutput =  0
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  115
PC =  4194384
Total number of clock cycles =  115
```

```
Register Values:
$s0: 1
$s1: 2
$s2: 2
$s3: Garbage value
$s4: Garbage value
$s5: Garbage value
$s6: Garbage value
$s7: Garbage value
$0: 0
$t0: Garbage value
$t1: 4
$t2: 268501216
$t3: 268501232
$t4: 4
$t5: Garbage value
$t6: Garbage value
$t7: Garbage value



Data Memory:
268501216: 0
268501220: 1
268501224: 1
268501228: 2
```

2. Factorial screenshots:

```
shashank@DESKTOP-S26019G:/mnt/c/Users/Admin/Programming files/CA Processor pipelining/CA assignment final$ python3 non_pipeline1.py
1. Fibonacci Numbers
2. Factorials of first 10 numbers
Enter appropriate number: 2
Enter starting address of data memory: 268501216
PCMax =  4194344
currInstruction =  00100000000001001000000000001010

instructionToPerform =  addi
opcode =  001000
ALUOutput =  10
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t1
dataToWriteToRegister =  10
clock cycles =  5
PC =  4194308
currInstruction =  00100000000001010000000000000001

instructionToPerform =  addi
opcode =  001000
ALUOutput =  1
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t2
dataToWriteToRegister =  1
clock cycles =  10
PC =  4194312
currInstruction =  10101101011010100000000000000000
```

```
instructionToPerform =  sw
opcode =  101011
ALUOutput =  268501216
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  1
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  15
PC =  4194316
currInstruction =  00010001010010010000000000000110

instructionToPerform =  beq
opcode =  000100
ALUOutput =  -9
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  20
PC =  4194320
currInstruction =  00100001010010100000000000000001

instructionToPerform =  addi
opcode =  001000
ALUOutput =  2
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t2
dataToWriteToRegister =  2
clock cycles =  25
PC =  4194324
currInstruction =  10001101011100000000000000000000

instructionToPerform =  lw
opcode =  100011
ALUOutput =  268501216
dataMemoryOutput =  1
dataToWriteToMemory =  Garbage value
writebackDestination =  s0
dataToWriteToRegister =  1
clock cycles =  30
PC =  4194328
currInstruction =  00100001011010110000000000000100
```

```
currInstruction =  00100001011010110000000000000100

instructionToPerform =  addi
opcode =  001000
ALUOutput =  268501220
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  t3
dataToWriteToRegister =  268501220
clock cycles =  35
PC =  4194332
currInstruction =  01110010000010101000100000000010

instructionToPerform =  mul
opcode =  011100
ALUOutput =  2
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =  s1
dataToWriteToRegister =  2
clock cycles =  40
PC =  4194336
currInstruction =  10101101011100010000000000000000

instructionToPerform =  sw
opcode =  101011
ALUOutput =  268501220
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  2
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  45
PC =  4194340
currInstruction =  00001000000100000000000000000011
instructionToPerform =  jump
opcode =  000010
ALUOutput =  Garbage value
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  50
PC =  4194316
currInstruction =  00010001010010010000000000000110
```

```
currInstruction =  00010001010010010000000000000110

instructionToPerform =  beq
opcode =  000100
ALUOutput =  0
dataMemoryOutput =  Garbage value
dataToWriteToMemory =  Garbage value
writebackDestination =
dataToWriteToRegister =  0
clock cycles =  335
PC =  4194344
Total number of clock cycles =  335



Register Values:
$s0: 362880
$s1: 3628800
$s2: Garbage value
$s3: Garbage value
$s4: Garbage value
$s5: Garbage value
$s6: Garbage value
$s7: Garbage value
$0: 0
$t0: Garbage value
$t1: 10
$t2: 10
$t3: 268501252
$t4: Garbage value
$t5: Garbage value
$t6: Garbage value
$t7: Garbage value
```

```
Data Memory:
$268501216: 1
$268501220: 2
$268501224: 6
$268501228: 24
$268501232: 120
$268501236: 720
$268501240: 5040
$268501244: 40320
$268501248: 362880
$268501252: 3628800
```

2. IMT2022_107_115_pipeline.py

This program simulates a pipelined MIPS processor. Forwarding along with stalls is used, without any re-ordering of instructions.

In a dictionary, we keep track of the stage that needs to be executed for a particular instruction. The key is PC for that instruction and value is the corresponding stage (IF, ID, EX, ME, WB, or None). None means that the instruction has finished execution and thus can be removed from the dictionary.

If any stage is repeated in the same clock cycle, then we do not execute it in order to avoid structural hazard. We keep track of this using 5 boolean variables, each corresponding to the respective stage.

At each clock cycle, we iterate through the dictionary. Then, based on the stage that needs to be executed, the corresponding steps are taken:

i. IF stage

Given the PC, we fetch the binary instruction from the instruction memory. The instruction memory is a dictionary and thus we access the instruction using instructionMemory[PC]. We also update the stage to ID for given PC.

ii. ID stage

In this stage, the instructionDecode function returns a list of the components that make up the instruction. For example, for an R format instruction, the following list is returned:

[opcode, rs, rs_value, rt, rt_value, rd, funct]

Additionally, we also implement fast branching for BEQ in the ID stage. If the values are already written back, then we can directly check for equality based on the value in registerToValue dictionary. Otherwise, we need to forward the values from previous clock cycles. These values are maintained in the dictionary executeValues. We return a tuple (True, decodedList) if the values are equal.

If we cannot forward the values, then we need to put a stall. This is achieved by returning None. The stage remains ID for the next clock cycle.

If we take the branch, then we need to flush the instructions after BEQ. Otherwise, the program flow is not disturbed.

We update the stage to EX if there is no stall.

iii. EX stage

In this stage, we perform the required ALU operation. Forwarding needs to be done in case the registers are not written back. To know whether forwarding is required or not, we find the clock cycles when the WB occurs and when the value of register is available. If forwarding is not possible at current clock cycle, then we return None.

For load word and store word instructions, the memory locations are calculated in the EX stage. We do not forward registers in this stage, as the forwarding is done in the MEM stage.

In case of a jump instruction, we update our PC to the jump target address and flush some instructions.

We also update the registerToClockCycle for given register to help in forwarding.

We update the next stage to ME if there is no stall.

iv. ME stage

Only SW and LW instructions access the data memory.

In case of SW instruction, we forward the required registers from previous clock cycles. Then, store the value at the desired memory location.

In case of LW instruction, we forward the register if required. We return the value at the given memory location. This value is stored in the memoryValues dictionary.

Update the next stage to be WB in case of no stalls.

v. WB stage

In the WB stage, we write the values calculated in EX and ME stage to the desired register.

These values are stored in the dictionaries executeValues and memoryValues. Also, we need to update the registserToWB dictionary for the destination register.

Update the next stage to be None, indicating that the instruction has finished execution.

After iterating through the dictionary, we update PC to JTA or PC + 4. Also, we need to set the stage as IF for new instruction, if any. We also remove the completed instructions from dictionary.

During the various stages, we keep track of the registers that are forwarded. Also, we maintain the pipeline diagram and update it according to the stage executed.

In the end, we print values of all registers and values stored in data memory. We display the total clock cycles, along with the pipeline diagram (with forwarding messages).

Fibonacci program output:

```
1. Fibonacci Numbers
2. Factorials of first 10 numbers
Enter appropriate number: 1
No. of fibonacci numbers to be calculated: 4
Enter starting address of numbers (in decimal format): 268501216

Register Values:
$s0: 1
$s1: 2
$s2: 2
$s3: Garbage value
$s4: Garbage value
$s5: Garbage value
$s6: Garbage value
$s7: Garbage value
$t0: Garbage value
$t1: 4
$t2: 268501216
$t3: 268501232
$t4: 4
$t5: Garbage value
$t6: Garbage value
$t7: Garbage value

Data Memory:
268501216: 0
268501220: 1
268501224: 1
268501228: 2
```

```
Total Clock Cycles: 32

     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  1. IF ID EX ME WB

  2.    IF ID EX ME WB

  3.       IF ID EX ME WB
Forward $s0 from clock cycle 3 to clock cycle 6

  4.          IF ID EX ME WB
Forward $s1 from clock cycle 4 to clock cycle 7

  5.             IF ID EX ME WB

  6.                IF ID EX ME WB

  7.                   IF __ ID EX ME WB
Forward $t4 from clock cycle 8 to clock cycle 9

  8.                        IF ID EX ME WB

  9.                           IF ID EX ME WB
Forward $s2 from clock cycle 11 to clock cycle 13

 10.                              IF ID EX ME WB

 11.                                 IF ID EX ME WB

 12.                                    IF ID EX ME WB

 13.                                       IF ID EX ME WB

 14.                                          IF ID EX ME WB

 15.                                                   IF ID EX ME WB

 16.                                                      IF ID EX ME WB

 17.                                                         IF ID EX ME WB
Forward $s2 from clock cycle 21 to clock cycle 23

 18.                                                            IF ID EX ME WB

 19.                                                               IF ID EX ME WB

 20.                                                                  IF ID EX ME WB

 21.                                                                     IF ID EX ME WB

 22.                                                                        IF ID EX ME WB

 23.                                                                                 IF ID EX ME WB

 24. Flushed due to BEQ
```

Factorial program output:

```
root@DESKTOP-Kandarp:/mnt/c/Users/LENOVO/Desktop/Computer Architecture/Assignments/MARS/IMT2022_107_115_assignment2# python3 pipeline.py
1. Fibonacci Numbers
2. Factorials of first 10 numbers
Enter appropriate number: 2
Enter starting address of factorials (in decimal format): 268501216

Register Values:
$s0: 362880
$s1: 3628800
$s2: Garbage value
$s3: Garbage value
$s4: Garbage value
$s5: Garbage value
$s6: Garbage value
$s7: Garbage value
$t0: Garbage value
$t1: 10
$t2: 10
$t3: 268501252
$t4: Garbage value
$t5: Garbage value
$t6: Garbage value
$t7: Garbage value

Data Memory:
268501216: 1
268501220: 2
268501224: 6
268501228: 24
268501232: 120
268501236: 720
268501240: 5040
268501244: 40320
268501248: 362880
268501252: 3628800

Total Clock Cycles: 89
```

```
     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
  1. IF ID EX ME WB

  2.    IF ID EX ME WB

  3.       IF ID EX ME WB
Forward $t2 from clock cycle 4 to clock cycle 6

  4.         IF ID EX ME WB
Forward $t2 from clock cycle 4 to clock cycle 5

  5.            IF ID EX ME WB

  6.               IF ID EX ME WB

  7.                  IF ID EX ME WB

  8.                     IF ID EX ME WB

  9.                        IF ID EX ME WB
Forward $t3 from clock cycle 9 to clock cycle 12
Forward $s1 from clock cycle 10 to clock cycle 12

 10.                           IF ID EX ME WB

 13.                                    IF ID EX ME WB

 14.                                       IF ID EX ME WB

 15.                                          IF ID EX ME WB

 16.                                             IF ID EX ME WB

 17.                                                IF ID EX ME WB

 18.                                                   IF ID EX ME WB
Forward $t3 from clock cycle 18 to clock cycle 21
```

For the factorial program, full pipeline diagram is not attached as it is too big (89 clock cycles) and hence not properly indented for later clock cycles.