Assignment-2

Dikshant Mahawar(IMT2022549) Krish Patel(IMT2022097)

November 11, 2023

1 Non Pipelining:

1.1 Fetch Instruction:

This function takes the instruction memory and program counter (pc) as parameters and returns the instruction located at the current program counter (pc) in the instruction memory.

1.2 Binary Neg Decimal Function:

This function converts a binary value to a decimal value, handling negative cases by checking the most significant bit (MSB) and converting accordingly.

1.3 Decode Instruction:

This function decodes an instruction based on its opcode. It supports various instruction types such as R-type, lw (load word), sw (store word), addi (add immediate), bne (branch not equal), beq (branch equal), mul (multiply), li (load immediate), jal (jump and link), and j (jump).

1.4 Execute Instruction:

This function executes the decoded instruction (l). It determines the control lines for write-back (wb), write memory (wm), and read memory (rm). It also calculates the result (rd1 for r-type instructions) and (rt1 for i-type instructions) based on the operation.

1.5 Memory Access Function:

This function handles memory access based on the control lines for write memory (wm) and read memory (rm). It either reads from or writes to data memory.

1.6 Write Back Function:

This function updates the register memory (reg memory) based on the result (rd1) and the destination register (rd).

1.7 Output:

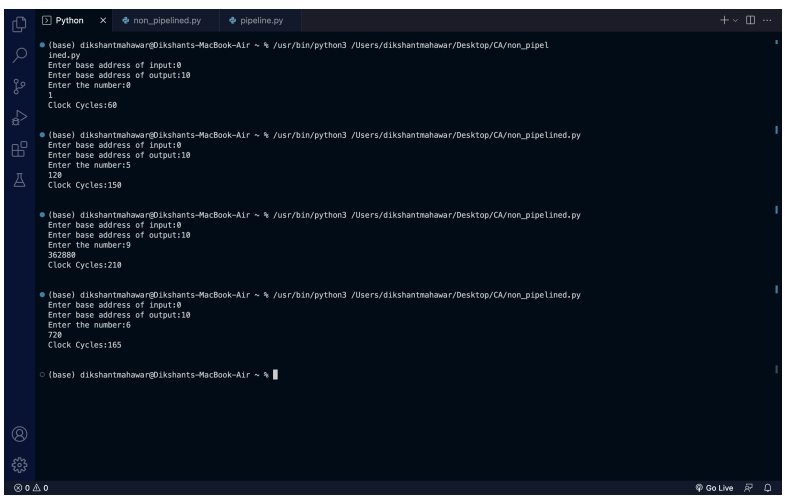

Figure 1: Factorial

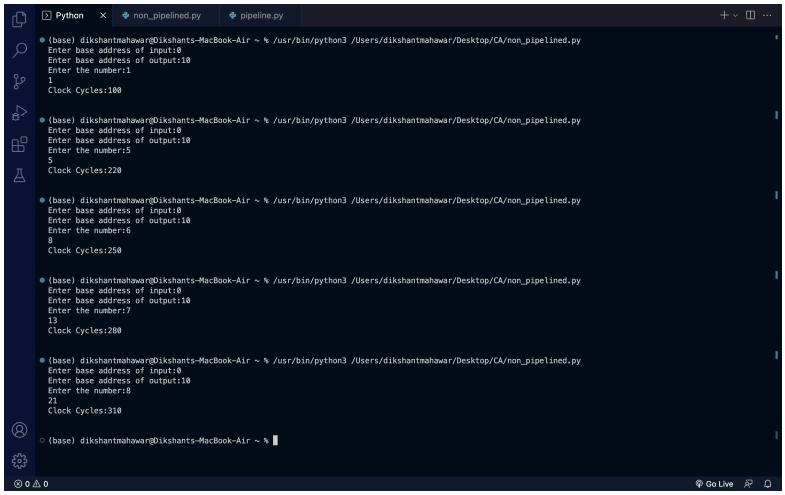

Figure 2: Fibonacci Series

2 Pipelining:

2.1 Dependency Function:

For finding dependency, we check if the current instruction has any dependencies on the previous 2 instructions. To check this, we see if the destination registers of the previous two instruction matches with any of the source register of the current instruction. We then create a dependency array which has true for the instructions which have any dependencies, else 0.

2.2 Adding Stalls:

For adding stalls, we use multiple variables which act as indexes for the instruction which are supposed to be fetches, decoded, executed and so on. Whenever we have any dependency in an instruction, we add 2 stalls and the other instructions follow the same. Suppose in decode stage we find that there is an dependency, we do not send the instruction to execute stage by not increasing the value of dec-idx. We decode the same instruction for 2 more cycles until the instruction above are written back. And for the following instruction, we make the fetch stage wait for 2 cycles until the one with dependency moves on to the execute stage. We do all this by playing with the values of fetch-idx, dec-idx, exe-idx, mem-idx, wb-idx

2.3 Handling Beq and Bne:

As we are allowed to check for bne and beq in any stage, we do this in the fetch stage. If we find that there is any jump instruction, we now change the value of fetch-idx to the jump address. We also ensure that no other instruction after the jump instruction is executed before jumping to the required address.

2.4 Output:

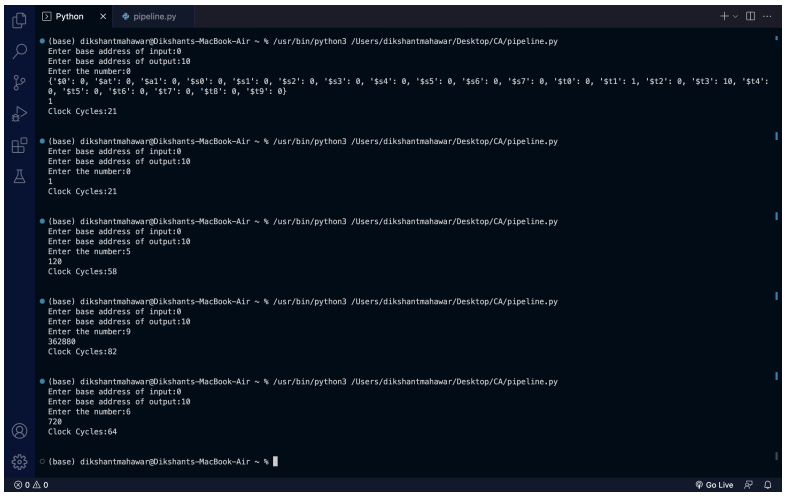

Figure 3: Factorial

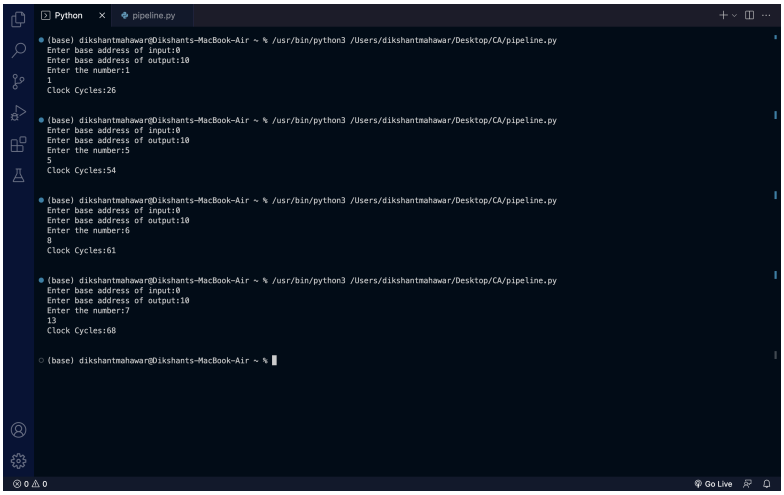

Figure 4: Fibonacci Series