Computer Architecture Assignment-2

1.)Name – Mupparapu Koushik

Roll No. – IMT2022570

Email-ID – Koushik.Mupparapu@iiitb.ac.in

2.)Name – Komaragiri Sai Vishwanath Rohit

Roll No. – IMT2022576

Email-ID – Komaragiri.Sai@iiitb.ac.in

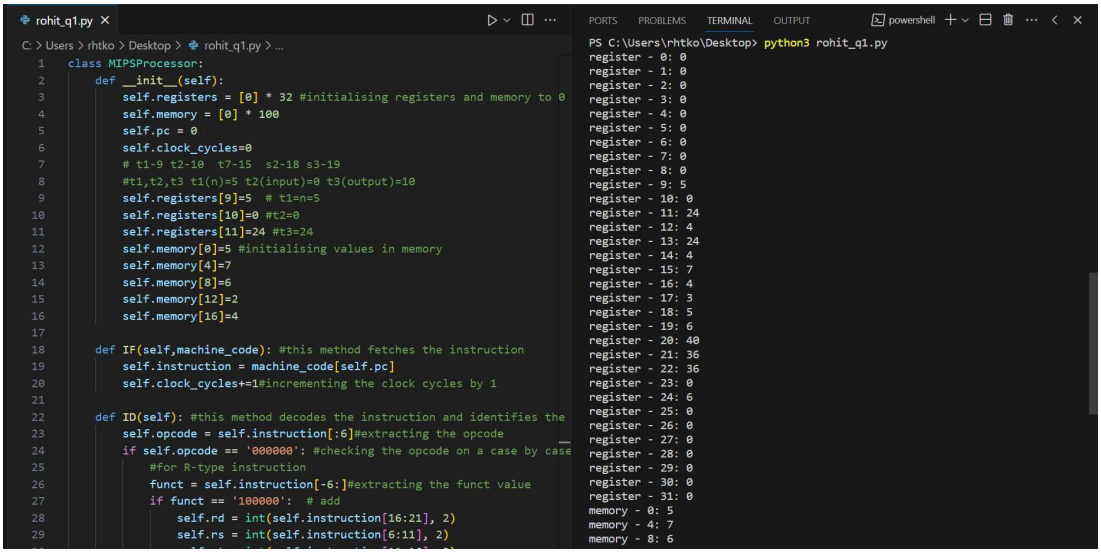
Question – 1:

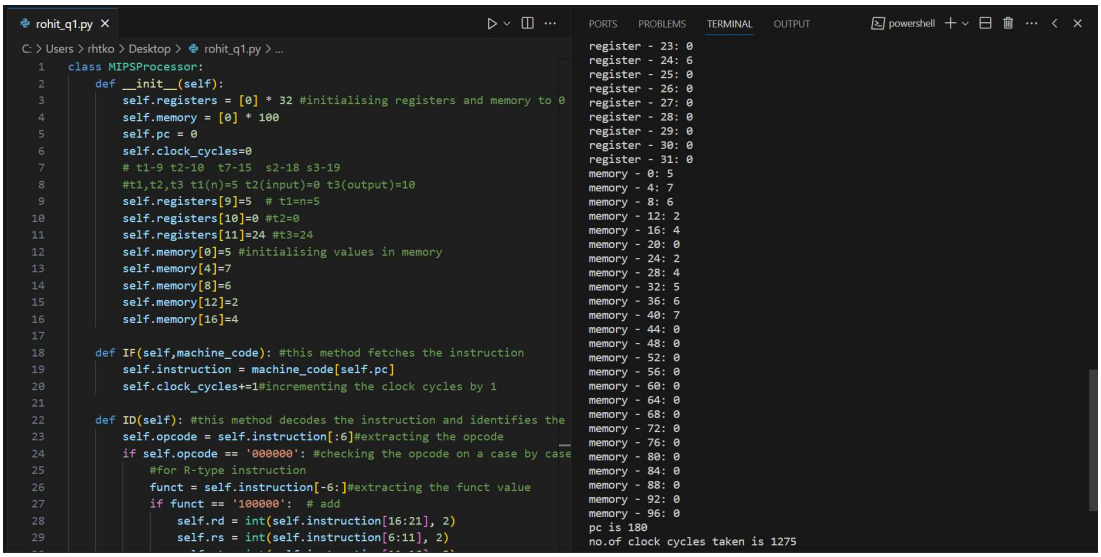
For the first question, we implemented (i.e. simulated) the Non-Pipelined MIPS Processor in a Five-stage manner. We used two lists – 'memory' and 'registers', which are initialised to 0, to simulate storing the values in memory and registers respectively. We take the input, which is machine code in the binary format, from an external file read it and store it in a list called 'machine_code'. The 5 stages of the Processor are:

- IF – In this stage, we fetch the corresponding instruction from the input file.
- ID – In this stage, we Decode the instruction which was fetched in the previous stage i.e. we extract the OpCode, Operand Registers/Immediate Value[s](in case of I-Type of Instructions)/Address(in case of J-Type Instructions) and Funct values(in case of R-Type Instructions).
- EX – In this stage, we perform the specific operations based on the decoded OpCode, the decoded registers, operands/immediate values/addresses like addition, subtraction, loading from memory and storing into memory, branching etc.
- MEM – In this stage, we load the data from the memory (LW instruction) or store it in the memory (SW instruction).
- WB – In this stage, we write the data into corresponding registers.

All these 5 stages are executed in order in a function called run_program in the MIPSProcessor class. The function calls them in the aforementioned order and increments the PC value after all 5 stages for a single instruction. The clock cycles are incremented by 1 every time it reaches the IF stage and in the output, it is multiplied by 5 as each instruction cycle has 5 stages. PC is incremented by 1 after each instruction and finally in the output it is multiplied by 4.

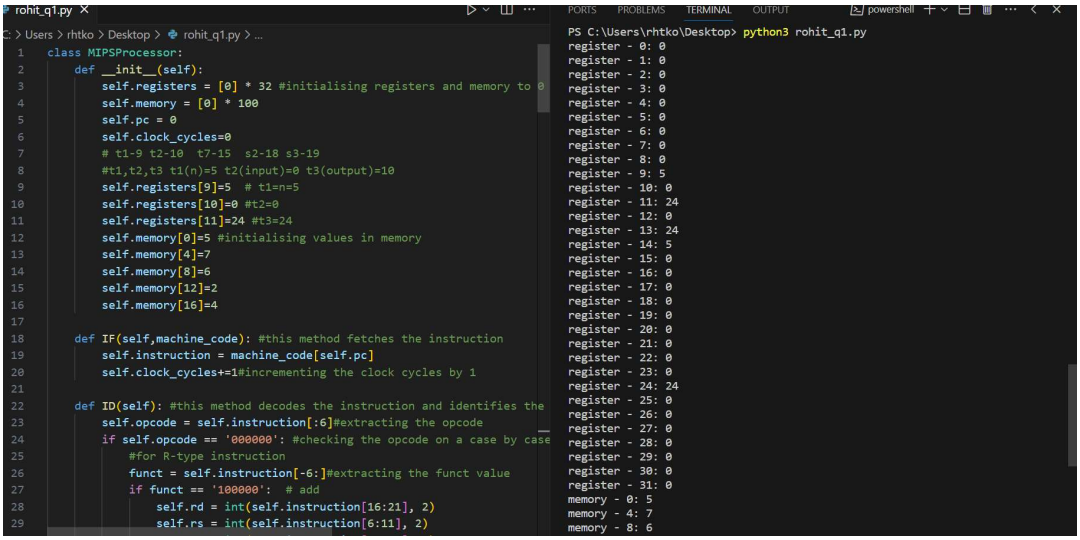
Firstly, we used the Processor to execute a Sorting Algorithm (Selection Sort). The input values that are to be sorted are hardcoded into the memory (i.e. memory list) and we also store the numbers of integers to be inputted, the starting address of input and the starting address of output in the registers as shown in the code. We get the output as shown below.

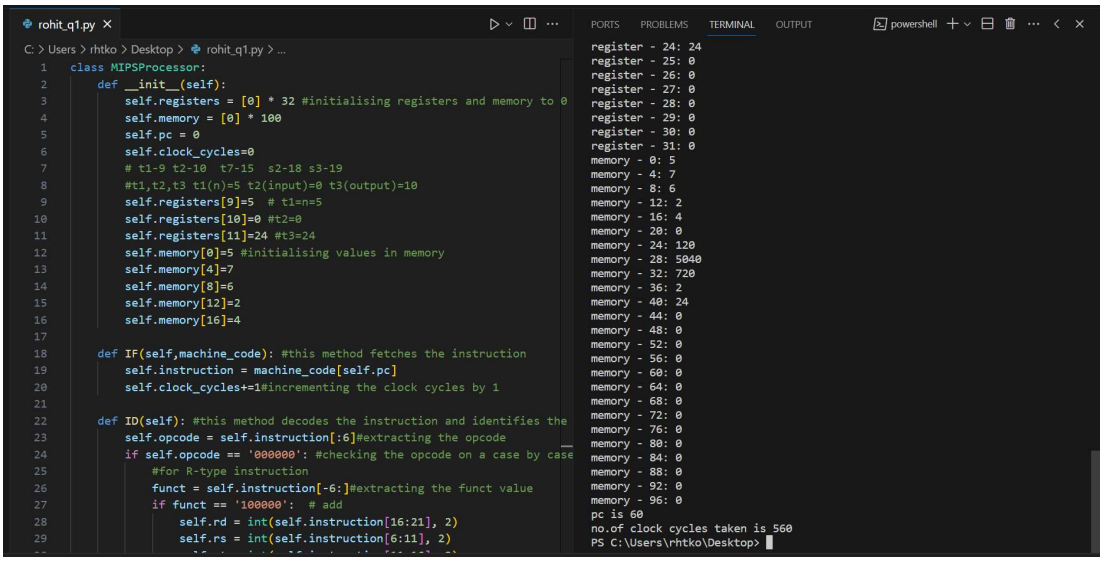




As shown in the above output pics, we first give the input as 5 numbers to be sorted from memory location 0 – 5,7,6,2,4. We then get the output from memory location 24 as 2,4,5,6,7. Since we have 45 instructions in the Sorting Machine Code we get the final PC value as 45*4 = 180. The number of clock cycles taken to execute the entire Selection Sorting Algorithm is 1275 as shown.

Secondly, we used the Processor to execute a Factorial Algorithm. The input value for which the factorial is to be calculated is hardcoded into the memory (i.e. memory list), the starting address of input and the starting address of output in the registers as shown in the code. We get the output as shown below.





As shown in the above output pics, we first give the input as 5 numbers for which factorial is to be calculated from memory location 0 – 5,7,6,2,4. We then get the output from memory location 24 as 120, 5040, 720, 2, and 24 (which are the factorials of the corresponding numbers). Since we have 15 instructions in the Factorial Machine Code, we get the final PC value as 15*4 = 60. The number of clock cycles taken to execute the entire Factorial Algorithm is 560 as shown.

Question – 2:

For the second question, we implemented (i.e. simulated) the Pipelined MIPS Processor in a Five-stage manner, with the help of 5 additional Pipelined Registers. We used two lists – 'memory' and 'registers', which are initialised to 0, to simulate storing the values in memory and registers respectively similar to the first question. We take the input, which is machine code in the binary format, from an external file read it and store it in a list called 'machine_code' just like we did in the first question. The 5 stages of the Processor are similar to the first question but we also use the pipelined registers in the stages as explained below:

- IF – In this stage, we fetch the corresponding instruction from the input file.
- ID – In this stage, we Decode the instruction which was fetched in the previous stage i.e. we extract the OpCode, Operand Registers/Immediate Value[s](in case of I-Type of Instructions)/Address(in case of J-Type Instructions) and Funct values(in case of R-Type Instructions) similar to the first question. We are implementing fast branching in the case of BEQ, BNE and J instructions, where PC value changes in the ID stage itself to resolve any control hazards. We don't need to flush the pipeline as we are changing the PC value before we encounter the IF of the next instruction. Hence forwarding from the EX and MEM stages of the previous instructions to the ID stage of the current instruction (BEQ or BNE instructions) occurs when we encounter a data hazard.
- EX – In this stage, we perform the specific operations based on the decoded OpCode, the decoded registers, operands/immediate values/addresses like addition, subtraction, loading from memory and storing into memory, branching etc. We forward the data from the WB or MEM stage of the previous instructions to the EX stage of the current instruction when we encounter a data hazard.
- MEM – In this stage, we load the data from the memory (LW instruction) or store it in the memory (SW instruction). We forward the data from the WB stage of the previous instructions to the MEM stage of the current instruction when we encounter a data hazard.
- WB – In this stage, we write the data into corresponding registers.

As mentioned earlier, we also use 5 Pipelined registers between the 5 stages as explained below:

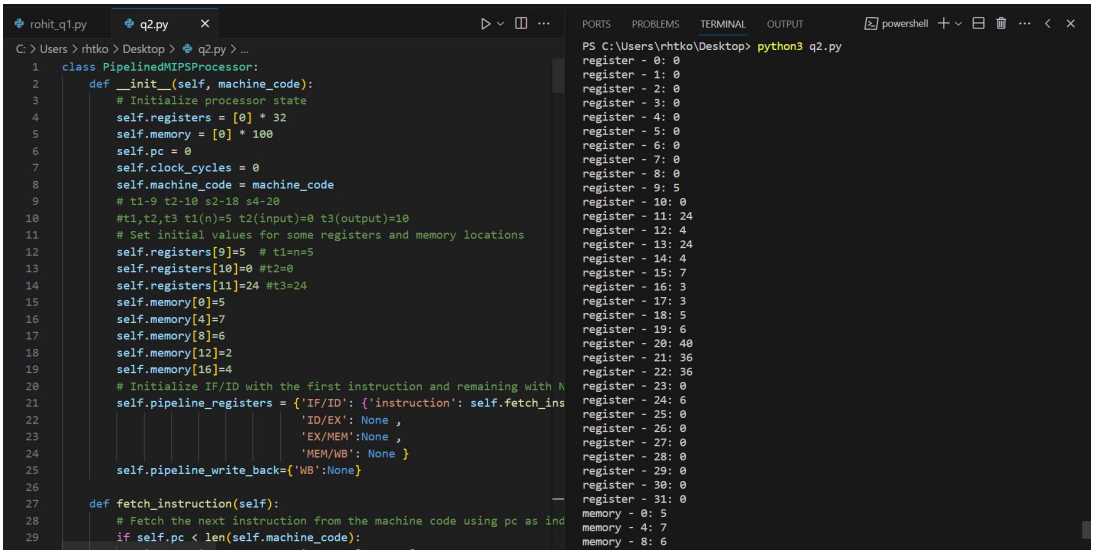
- IF/ID – Stores the 32-bit instruction and the PC value.
- ID/EX – Stores the OpCode, PC value, operand registers and the values stored in those respective operand registers, immediate values(in case of I-Type Instructions), addresses(in case of J-Type Instructions),funct values(in case of R-Type Instructions).
- EX/MEM – Stores the OpCode, PC value, destination register, address(in case of LW or SW instruction) and ALU Result.
- MEM/WB – Stores the OpCode, PC value, destination register, ALU Result and Value stored at the memory location stored in ALU Result(only for LW instruction).
- Writeback – Same as MEM/WB for each and every instruction.

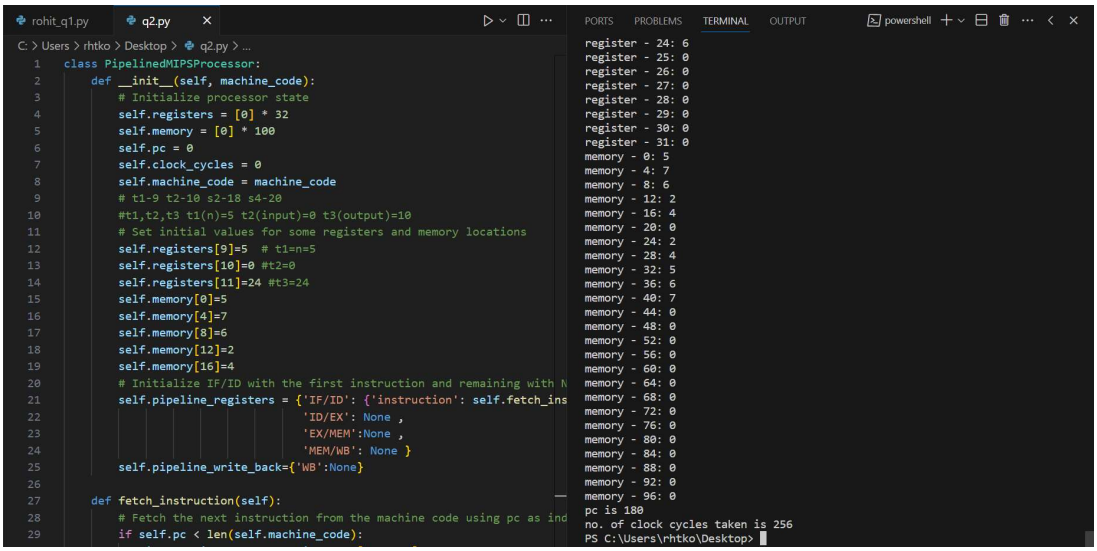
All these 5 stages are executed by a function called execute_stage in the PipelinedMIPSProcessor class. The function calls them and increments the PC value after the IF stage of a single instruction. The clock cycles are incremented by 1 every time we call the execute_instruction_set function.

The execute_instruction_set function executes a full clock_cycle set of instructions in the correct order.

We added the functionality of a stall which is necessary for a data hazard which may be encountered with LW instruction.

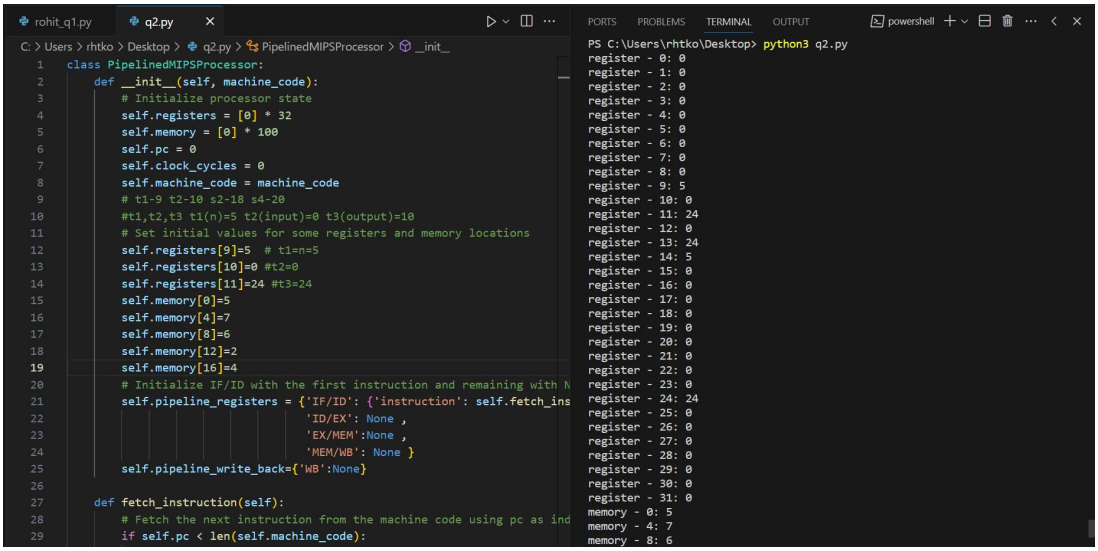
Firstly, we used the Processor to execute a Sorting Algorithm (Selection Sort) similar to the first question. The input values that are to be sorted are hardcoded into the memory (i.e. memory list) and we also store the numbers of integers to be inputted, the starting address of input and the starting address of output in the registers as shown in the code. We get the output as shown below.

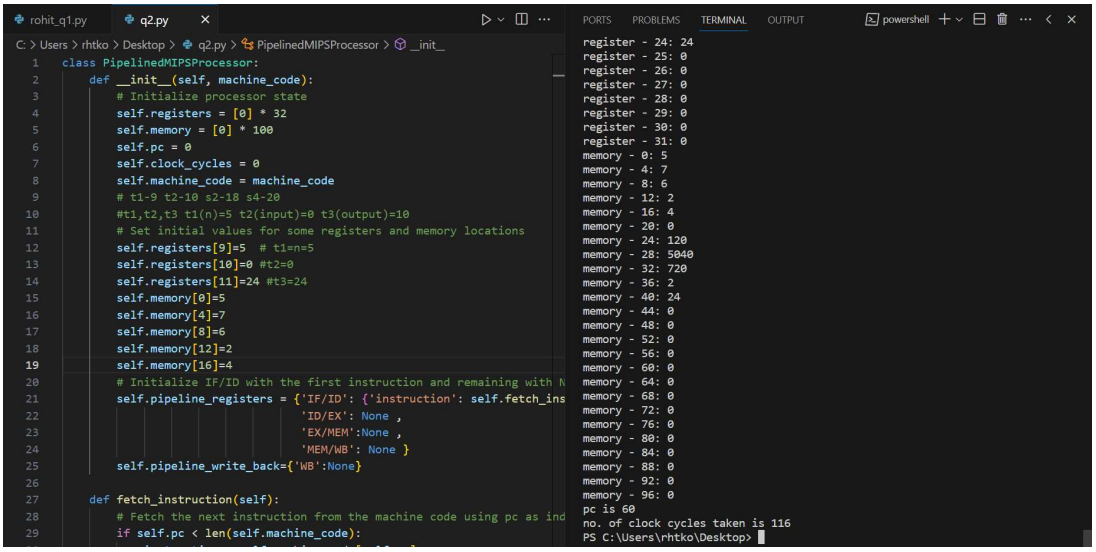




As shown in the above output pics, we give the same input as the first question. We then get the same output as the first question. Since we have 45 instructions in the Sorting Machine Code similar to the first question, we get the final PC value as 45*4 = 180. The number of clock cycles taken to execute the entire Selection Sorting Algorithm is 256 as shown. Comparatively, the Non-Pipelined code took 1275 clock cycles, which is almost 5 times the clock cycles for the same code taken by the Pipelined code. Hence, we can say that the Pipelined code is very fast when compared to the Non-Pipelined code.

Secondly, we used the Processor to execute a Factorial Algorithm similar to the first question. The input value for which the factorial is to be calculated is hardcoded into the memory (i.e. memory list), the starting address of input and the starting address of output in the registers as shown in the code. We get the output as shown below.





As shown in the above output pics, we first give the same input as the first question. We then get the same output as the first question. Since we have 15 instructions in the Factorial Machine Code we get the final PC value as 15*4 = 60. The number of clock cycles taken to execute the entire Selection Sorting Algorithm is 116 as shown. Comparatively, the Non-Pipelined code took 560 clock cycles, which is almost 5 times the clock cycles for the same code taken by the Pipelined code. Hence, we can say that the Pipelined code is very fast when compared to the Non-Pipelined code.