High Performance Computer Architecture

Assignment 1

The goal of this assignment is to write a simulator for Tomasulo's algorithm. You can implement the simulator using C/C++/Java/Python, whichever you prefer. **This assignment has to be completed individually by each student.**

Processor Configuration

The processor only supports four types of instructions: integer addition (`ADD`), integer subtraction (`SUB`), integer multiplication (`MUL`), and integer division (`DIV`). The processor has two units: *integer addition unit* and *integer multiplication unit*. The addition unit can perform both **addition** and **subtraction** operations, which take 2 cycles each to execute. The addition unit has **three reservation stations** (`RS0`, `RS1`, and `RS2`). The multiplication unit can perform both **multiplication** and **division** operations, which take 10 cycles and 40 cycles to execute respectively. The multiplication unit has **two reservation stations** (`RS3` and `RS4`). The processor has **eight registers** (`R0-R7`). The processor does not allow same-cycle issue and dispatch. The processor also does not allow same-cycle capture and dispatch. If a reservation station is freed in a certain cycle, it cannot be allocated again in the same cycle. If both the addition unit and the multiplication unit try to broadcast in the same cycle, the multiplication unit will have precedence. If two or more instructions are ready to be dispatched in the same cycle in the addition unit, the instruction in `RS0` will have the highest priority, followed by `RS1`, followed by `RS2`. Similarly, if two instructions are ready to be dispatched in the same cycle in the multiplication unit, the instruction in `RS3` will have priority over `RS4`. The processor has an instruction queue that can hold up to a **maximum of 10 instructions**. If any additional necessary configurations are required for the implementation, other than the description mentioned above, you can consider those with a proper explanation for the additional configuration in a README file while submitting the assignment. *Unnecessary assumptions will be penalised.*

Data Structure

In order to implement the simulator, you will need the following data structures. Instructions can be stored in the queue as instruction records. Each instruction record will have four fields:

OPCODE	DST OPERAND	SRC OPERAND 1	SRC OPERAND 2

A 10-entry array of instruction records can be used as the instruction queue. In each cycle, the instruction queue array should be updated as a queue. An 8-entry array of integers can be used as the register file. An 8-entry array of integers can also be used for the `RAT`. The `RAT` array will store the tag. For example, `RS0` can be recorded as the value 0. A special value can be stored to identify the situation where a register is currently not tagged with a reservation station. Reservation stations can also be represented as records with several fields.

Input

The input to the simulator will be a text file that has the following format - the first line indicates the number of instructions. The second line indicates the number of cycles of simulation. The third line onward are the encoded instructions. For example, 0 2 4 6 is the instruction `R2=R4+R6`. The first number indicates the instruction opcode (0: add; 1: sub; 2: multiply; 3: divide). The second number indicates the destination register. The third and the fourth numbers indicate the source registers. After the list of encoded instructions, the input file should have the initial values to the register file. A sample input file is shown below.

```
2
7
0 2 4 6
2 4 3 5
3
4
6
21
3
4
9
0
```

Output

The output of the simulator should display the state of the reservation stations, register file, `RAT` and instruction queue after simulating the processor for n cycles, where n is obtained from the second line of the input file. A sample output is given below:

```
      Busy  Op  Vj  Vk  Qj  Qk  Disp
RS0   1     Add       3  RS1        1
RS1   0
RS2   0
RS3   0
RS4   0
-----------------------------------
   RF   RAT
0: 4    RS3
1: 29
2:      RS1
3:
4: 3
5:
6:
7:
-----------------------------------
Instruction Queue
Add R2, R4, R7
Sub R6, R1, R1
```

Submission

Submit the source files (and supplementary materials if required) in a zipped file with the name in the format: `<roll_no>_<type>.zip`, where `<type>` is C for C/C++, J for Java, and P for Python. **Please comment abundantly to make your code understandable** (otherwise appropriate credits will be deducted).