Microprocessors project report

Tomasulo architecture

German University in Cairo

الجامعة الألمانية بالقاهرة

Team name:

Stress Overflow

Team members:

1- Ganna Mohamed 49-17219

2- Aya Mahmoud 49-17973

3- Nourhan Ahmed 49-2023

4- Malak ElDardeery 49-3377

5- Mohamed Saeed 52-14850

1-Approach

The main method starts by reading a .txt file from the src folder and converting it into instructions in the instruction queue. Then, inputs are read from the user to dictate the sizes of buffers and latencies of instructions. The engine starts at cycle 1. Each cycle, the processor tries to find an instruction to write to the bus, then tries to find instructions to execute, if possible, then tries to issue an instruction. This order is maintained to prevent an instruction from being issued and then directly executing in the same cycle, and to prevent an instruction from finishing execution and writing to the bus in the same cycle.

The write result step is performed by looping on all reservation stations and load buffers to check if a busy buffer/station is done executing and is waiting to write its result to the bus, and make sure that no other instruction is currently writing to the bus. If conditions are met, write the result of this instruction along with the buffer/station tag to the bus, empty the buffer/station, and set a flag to indicate that an instruction is writing now to the bus to prevent two instructions from writing to the bus at the same time. Also, a variable is set in the instruction queue to indicate that this instruction wrote its result in this particular clock cycle.

The execute step is performed by looping on all reservation stations and buffers to check if a busy buffer/station has all operands available so it’s ready to execute and has time left in its execution cycles. For all these stations/buffers, decrease the time left in execution cycles by 1. If it’s the first cycle of execution, set the startExec time for this instruction in the instruction queue. If it’s the last cycle of execution, set the endExec time for this instruction in the instruction queue, and perform the operation (either an ALU operation, reading from memory, or storing in memory) and save the result.

The issue step is performed by fetching the instruction at the head of the instruction queue and finding its type. According to its type, check if the corresponding stations/buffers have a slot available. If there is no slot available, the instruction cannot be issued and no other instructions after it can be issued. If a slot is available, the instruction is issued in that station/buffer and it’s set to busy. The issue field of that instruction in the instruction queue is updated to be the current clock cycle. The operation of the station/buffer is set according to the instruction. In case of an ALU operation, the processor checks if the first and second operands are available in the register file. If the register is available, its value is read from the register file and written in the Vj/Vk fields. If the register isn’t available, its Q tag is written in the Qj/Qk fields. In case of loads and stores, the address is read and directly stored in the buffer. The time left of execution cycles for the particular station/slot is then set to be equal to the latency of that instruction. Then, the tag of this station/buffer is written in the Q tag of the destination register of the instruction. The id of this instruction in the instruction queue is then stored in the slot so we can edit its start/end/write times.

Before the end of each cycle, the register file and all the reservation stations and the store buffer read from the bus. This is done in the second half of the cycle, while writing to the bus is done in the first half of the cycle so that reservation stations and registers don't miss values they need. This is done by looping on all reservation stations and store buffers and registers in the register file. If there's data on the bus and the reservation station or store buffer is busy and has a Q tag that equals the tag on the bus, this reservation station/store buffer/register reads from the bus. Then, the value is set to whatever value that was on the bus, and the Q field is set to be empty, indicating that this value is available.

At the end of each cycle, the flag indicating that an instruction is writing to the bus is set to false, and the bus and its tag are nullified. In addition, the instruction queue with times for each instruction is printed, along with all the reservation stations, buffers, and the register file. Finally, the clock cycle is incremented.

The clock keeps running until all instructions have written their results on the bus. Then, the memory values are printed once.

2-Code structure

The code is divided into two parts: objects, and engine.

Objects we used are:

A- instructions, each with attributes instruction (to indicate the operation), i1 (to indicate the first register), i2 (to indicate the second register or the address in case of load and store), i3 (to indicate the third register), issue (to indicate the clock cycle when this instruction was issued), execStart (to indicate the clock cycle when this instruction started execution), execEnd (to indicate the clock cycle when this instruction finished execution), writeResult (to indicate when this instruction wrote its result on the bus). This object is used to form the instruction queue

B- reservation stations, each with attributes busy, op, Vj, Vk, Qj, Qk, time, result (to read from it when writing to the bus), id in instruction queue (to edit the times for issue, start, end, and write result)

C- load buffers, each with attributes busy, address, time, result (to read from it when writing to the bus), id in instruction queue (to edit the times for issue, start, end, and write result)

D- store buffers, each with attributes busy, address, V, Q, time, id in instruction queue (to edit the times for issue, start, end, and write result)

E- registers, each with attributes name, value, Q.

F- register file, which is an array of 32 registers. This object has functions to check if a register's value is available, to get a register value that's available, to set a value for a register, to get the Q tag of a register, and to set the Q tag for a register.

G- memory, which is an array of 128 integers with addresses 0 through 127

The engine has:

A- an instruction queue that we pull instructions from to issue

B- an array of add/sub reservation stations which represents the first group of reservation stations, whose number is taken as an input from the user.

C- an array of mul/div reservation stations which represents the second group of reservation stations, whose number is taken as an input from the user.

D- an array of load buffers, whose number is taken as an input from the user.

E- an array of store buffers, whose number is taken as an input from the user.

G- a register file

H- a memory of size 128

I- add latency, subtract latency, multiply latency, divide latency, load latency, store latency, which are all inputs from the user.

J- a clock that starts at cycle 1 to indicate the current cycle.

K- a bus that can hold only one number/result at a time and instructions write their results to it.

L- a bus tag that holds the tag of the reservation station/buffer where the instruction currently writing on the bus is in.

M- a pointer to the next instruction to be fetched from the instruction queue, and it starts at 0.

N- a boolean isWritingNow to prevent two instructions from writing to the bus in the same cycle.

3-Test cases

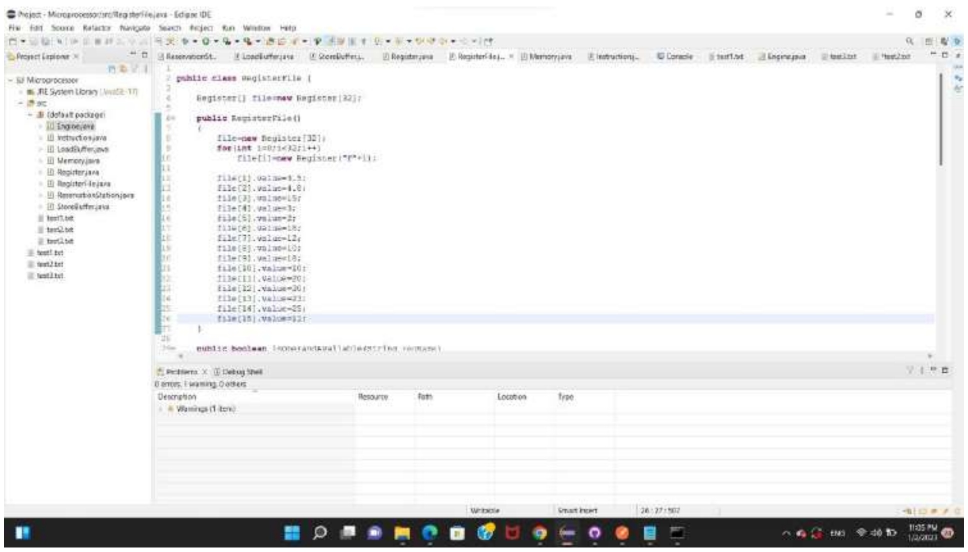
We assumed the memory size for our program is 128 addresses that run from 0 to 127.

The instructions the program can handle are written in the format of “ADD.D” “SUB.D” “MUL.D” “DIV.D” “L.D” “S.D”, and it’s assumed the effective address of the load and store is directly written in the instruction as L.D F2 50

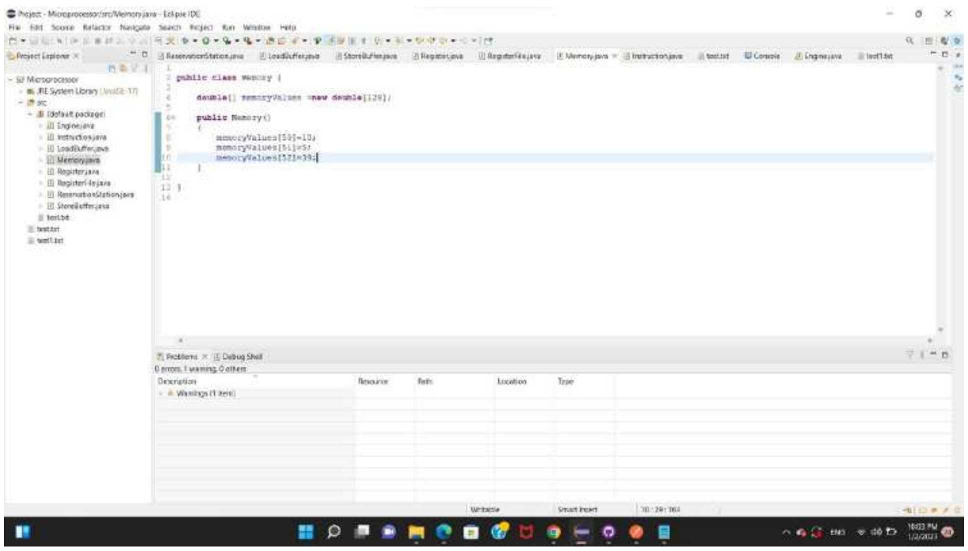
The instruction has no commas between arguments, only a single space character

(ex: ADD.D F5 F4 F3)

To test our program, we initialized some of the values in the register file and in the memory as follows (where the rest are 0 by default):



```java
public class RegisterFile {

    Register[] file=new Register[32];

    public RegisterFile()
    {
        file=new Register[32];
        for(int i=0;i<32;i++)
            file[i]=new Register("F"+i);

        file[1].value=5.5;
        file[2].value=4.8;
        file[3].value=15;
        file[4].value=3;
        file[5].value=2;
        file[6].value=18;
        file[7].value=12;
        file[8].value=10;
        file[9].value=18;
        file[10].value=10;
        file[11].value=20;
        file[12].value=30;
        file[13].value=23;
        file[14].value=25;
        file[15].value=12;
    }

    public boolean isOperandAvailable(String regName)
```



```java
public class Memory {

    double[] memoryValues =new double[128];

    public Memory()
    {
        memoryValues[50]=10;
        memoryValues[51]=5;
        memoryValues[52]=39;
    }

}
```

a) First case is that all instructions are independent and will find a place in the buffers. In addition, no two instructions finish execution in the same cycle and try to write to the bus at the same time.

(this uses the program test1.txt)

L.D F1 50

L.D F2 51

MUL.D F5 F3 F4

SUB.D F8 F6 F7

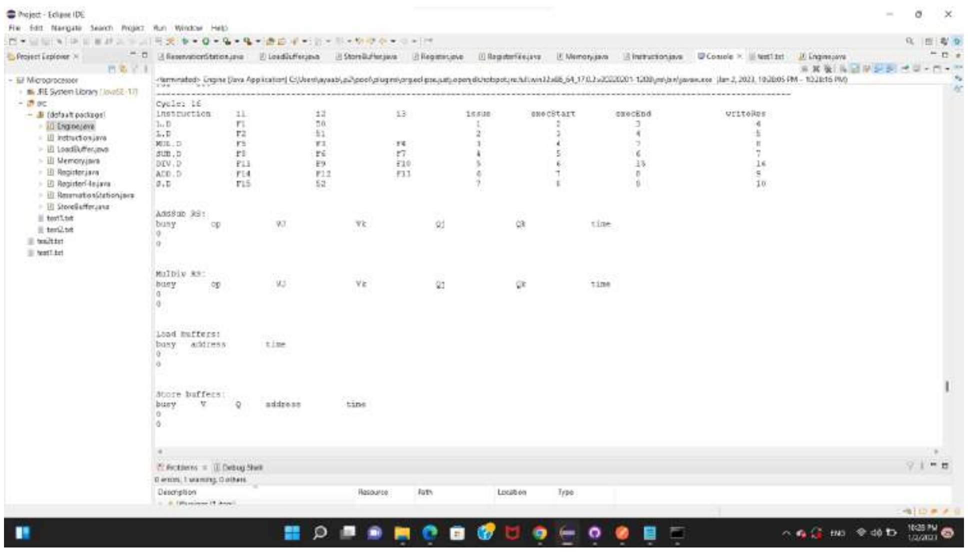
DIV.D F11 F9 F10

ADD.D F14 F12 F13

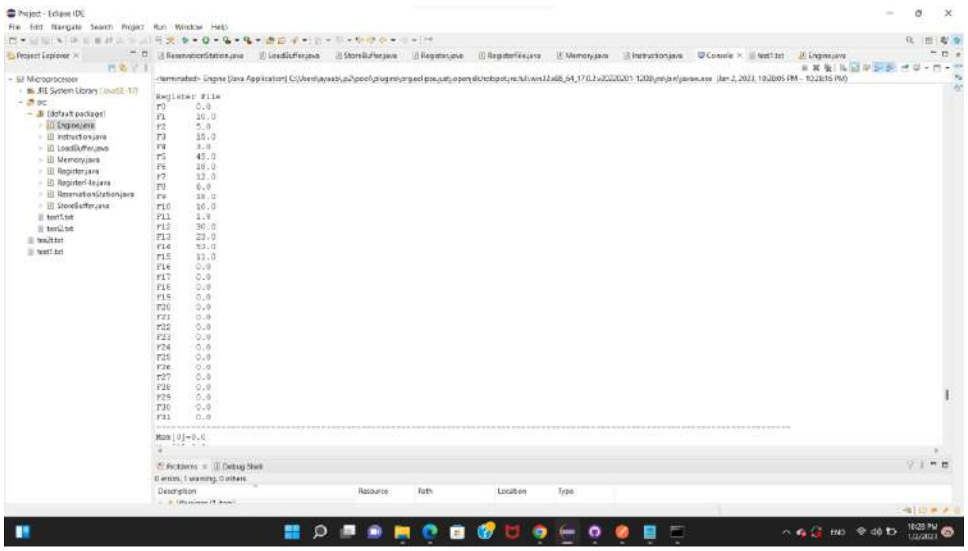
S.D F15 52

We assume here 2 add/sub reservations stations, 2 mul/div reservation stations, 2 load buffers, 2 store buffers, addition latency of 2, subtraction latency of 2, multiplication latency of 4, division latency of 10, load latency of 2, and store latency of 2.

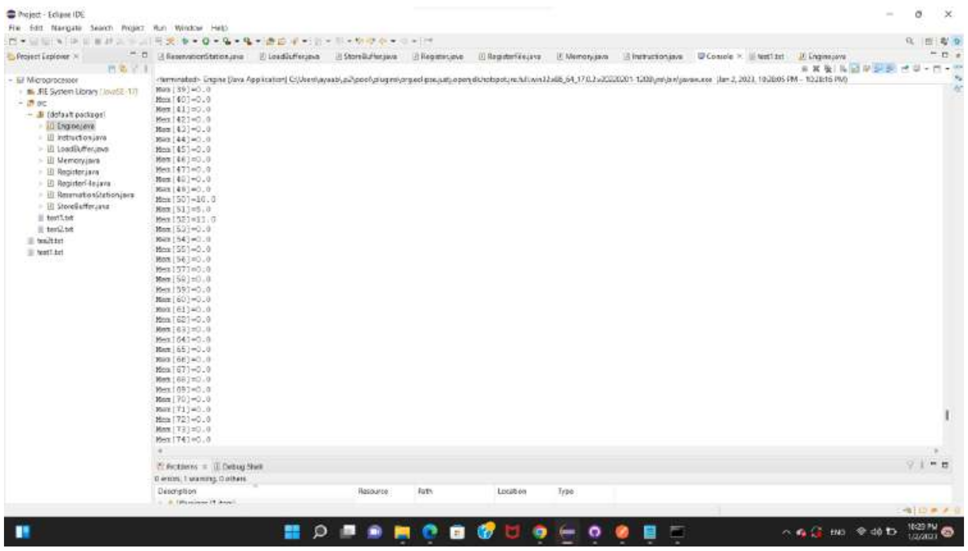
From eclipse:



```
Cycle: 16
Instruction    I1         I2         I3         issue     execStart     execEnd          writeRes
L.D            F1         50                    1         2             3                4
L.D            F2         51                    2         3             4                5
MUL.D          F5         F3         F4         3         4             7                8
SUB.D          F8         F6         F7         4         5             6                7
DIV.D          F11        F9         F10        5         6             15               16
ADD.D          F14        F12        F13        6         7             8                9
S.D            F15        52                    7         8             9                10

AddSub RS:
busy      op        Vj        Vk        Qj        Qk        time
0
0


MulDiv RS:
busy      op        Vj        Vk        Qj        Qk        time
0
0


Load buffers:
busy   address        time
0
0


Store buffers:
busy     V      Q     address        time
0
0
```



```
Register File
F0      0.0
F1      10.0
F2      5.0
F3      15.0
F4      3.0
F5      45.0
F6      18.0
F7      12.0
F8      6.0
F9      18.0
F10     10.0
F11     1.8
F12     30.0
F13     23.0
F14     53.0
F15     11.0
F16     0.0
F17     0.0
F18     0.0
F19     0.0
F20     0.0
F21     0.0
F22     0.0
F23     0.0
F24     0.0
F25     0.0
F26     0.0
F27     0.0
F28     0.0
F29     0.0
F30     0.0
F31     0.0
-----------------------------------------------------------
Mem[0]=0.0
```



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 2, 2023, 10:28:05 PM – 10:28:15 PM)
Mem[39]=0.0
Mem[40]=0.0
Mem[41]=0.0
Mem[42]=0.0
Mem[43]=0.0
Mem[44]=0.0
Mem[45]=0.0
Mem[46]=0.0
Mem[47]=0.0
Mem[48]=0.0
Mem[49]=0.0
Mem[50]=10.0
Mem[51]=5.0
Mem[52]=11.0
Mem[53]=0.0
Mem[54]=0.0
Mem[55]=0.0
Mem[56]=0.0
Mem[57]=0.0
Mem[58]=0.0
Mem[59]=0.0
Mem[60]=0.0
Mem[61]=0.0
Mem[62]=0.0
Mem[63]=0.0
Mem[64]=0.0
Mem[65]=0.0
Mem[66]=0.0
Mem[67]=0.0
Mem[68]=0.0
Mem[69]=0.0
Mem[70]=0.0
Mem[71]=0.0
Mem[72]=0.0
Mem[73]=0.0
Mem[74]=0.0
```

b) Second case is that all instructions are independent and will find a place in the buffers. However, two instructions finish execution in the same cycle and try to write to the bus at the same time. Here, we programmed the architecture to give priority in FIFO manner (I.e. the instruction that was issued earlier has the priority to write on the bus first, in case of a conflict)

(this uses the program test1.txt)

L.D F2 51

MUL.D F5 F3 F4

SUB.D F8 F6 F7

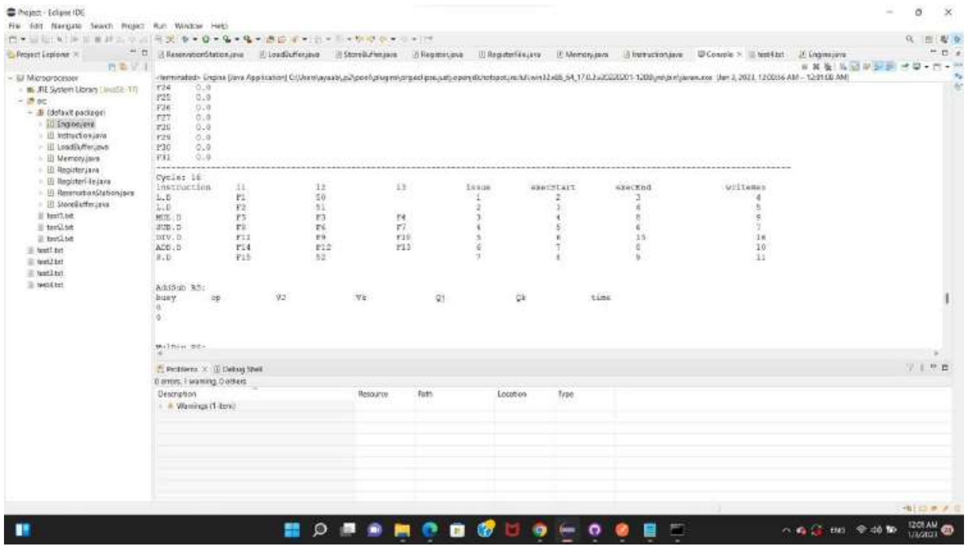
DIV.D F11 F9 F10

ADD.D F14 F12 F13

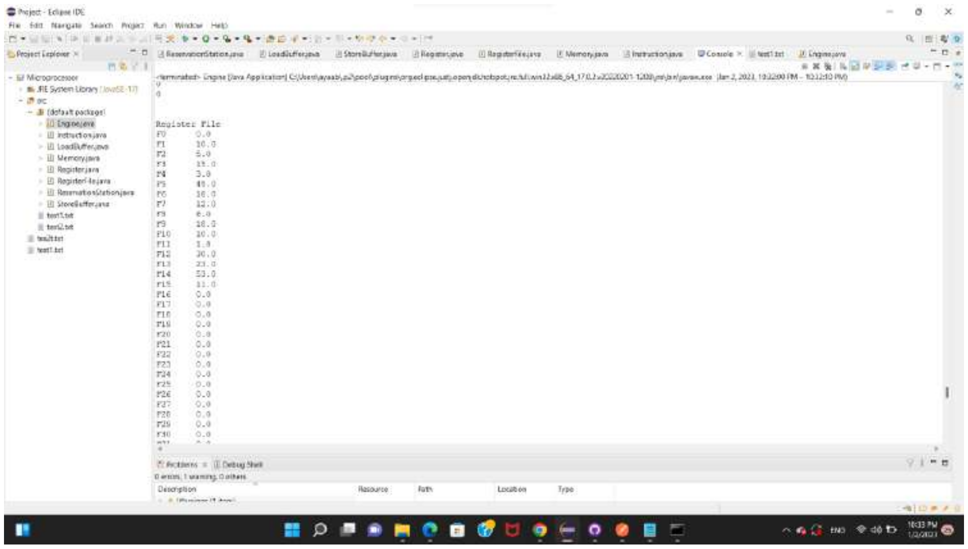
S.D F15 52

We assume here 2 add/sub reservations stations, 2 mul/div reservation stations, 2 load buffers, 2 store buffers, addition latency of 2, subtraction latency of 2, multiplication latency of 5, division latency of 10, load latency of 2, and store latency of 2.

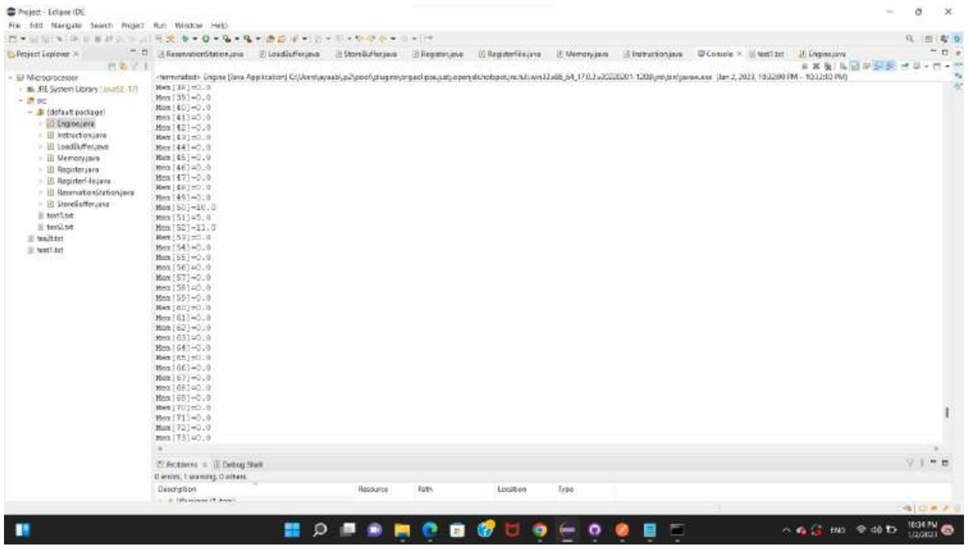
From eclipse:



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 3, 2023, 12:00:56 AM – 12:01:00 AM)
F24     0.0
F25     0.0
F26     0.0
F27     0.0
F28     0.0
F29     0.0
F30     0.0
F31     0.0
-------------------------------------------------------------------------------------------------------------------
Cycles: 16
Instruction    i1          i2          i3          Issue      execStart     execEnd         writeRes
L.D            F1          50                        1            2             3                 4
L.D            F2          51                        2            3             4                 5
MUL.D          F5          F3          F4            3            4             8                 9
SUB.D          F8          F6          F7            4            5             6                 7
DIV.D          F11         F9          F10           5            6             15                16
ADD.D          F14         F12         F13           6            7             8                 10
S.D            F15         52                        7            8             9                 11


AddSub RS:
busy      op          Vj          Vk          Qj          Qk          time
0
0


MulDiv RS:
```



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 2, 2023, 10:32:00 PM – 10:32:10 PM)
0
0


Register File
F0      0.0
F1      10.0
F2      5.0
F3      15.0
F4      3.0
F5      45.0
F6      18.0
F7      12.0
F8      6.0
F9      18.0
F10     10.0
F11     1.8
F12     30.0
F13     23.0
F14     53.0
F15     11.0
F16     0.0
F17     0.0
F18     0.0
F19     0.0
F20     0.0
F21     0.0
F22     0.0
F23     0.0
F24     0.0
F25     0.0
F26     0.0
F27     0.0
F28     0.0
F29     0.0
F30     0.0
```



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 2, 2023, 10:32:00 PM – 10:32:10 PM)
Mem[38]=0.0
Mem[39]=0.0
Mem[40]=0.0
Mem[41]=0.0
Mem[42]=0.0
Mem[43]=0.0
Mem[44]=0.0
Mem[45]=0.0
Mem[46]=0.0
Mem[47]=0.0
Mem[48]=0.0
Mem[49]=0.0
Mem[50]=10.0
Mem[51]=5.0
Mem[52]=11.0
Mem[53]=0.0
Mem[54]=0.0
Mem[55]=0.0
Mem[56]=0.0
Mem[57]=0.0
Mem[58]=0.0
Mem[59]=0.0
Mem[60]=0.0
Mem[61]=0.0
Mem[62]=0.0
Mem[63]=0.0
Mem[64]=0.0
Mem[65]=0.0
Mem[66]=0.0
Mem[67]=0.0
Mem[68]=0.0
Mem[69]=0.0
Mem[70]=0.0
Mem[71]=0.0
Mem[72]=0.0
Mem[73]=0.0
```

c) Third case is that all instructions are independent but won't necessarily find an empty reservation station to issue.

(this uses the program test2.txt)

L.D F1 50

L.D F2 51

L.D F3 52

ADD.D F8 F6 F7

SUB.D F11 F9 F10

ADD.D F14 F12 F13

MUL.D F18 F6 F7

DIV.D F19 F9 F10

MUL.D F20 F12 F13

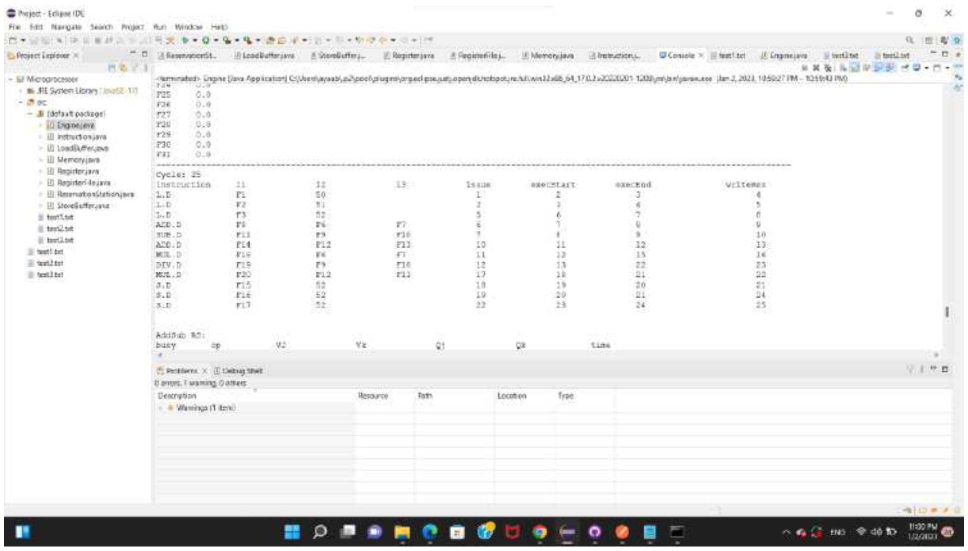
S.D F15 52

S.D F16 52

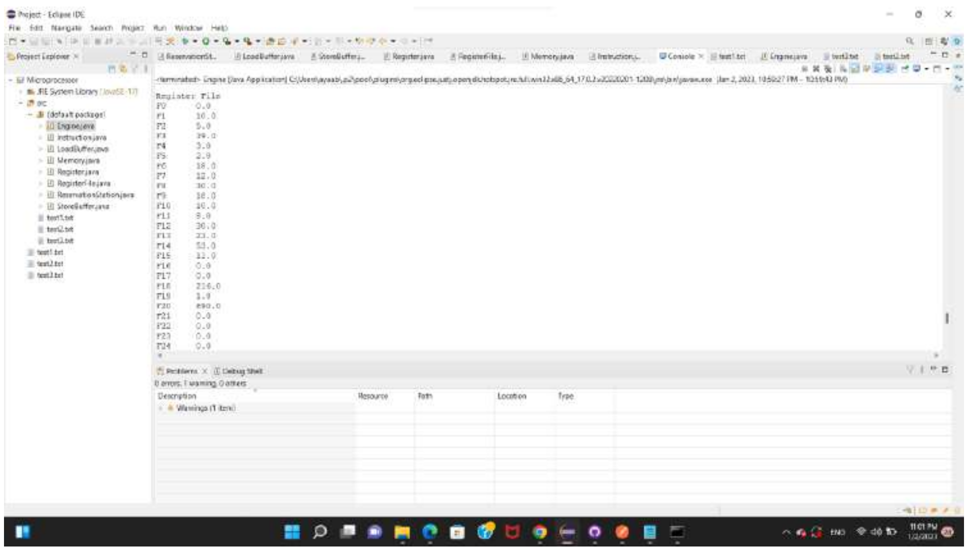
S.D F17 52

We assume here 2 add/sub reservations stations, 2 mul/div reservation stations, 2 load buffers, 2 store buffers, addition latency of 2, subtraction latency of 2, multiplication latency of 4, division latency of 10, load latency of 2, and store latency of 2.

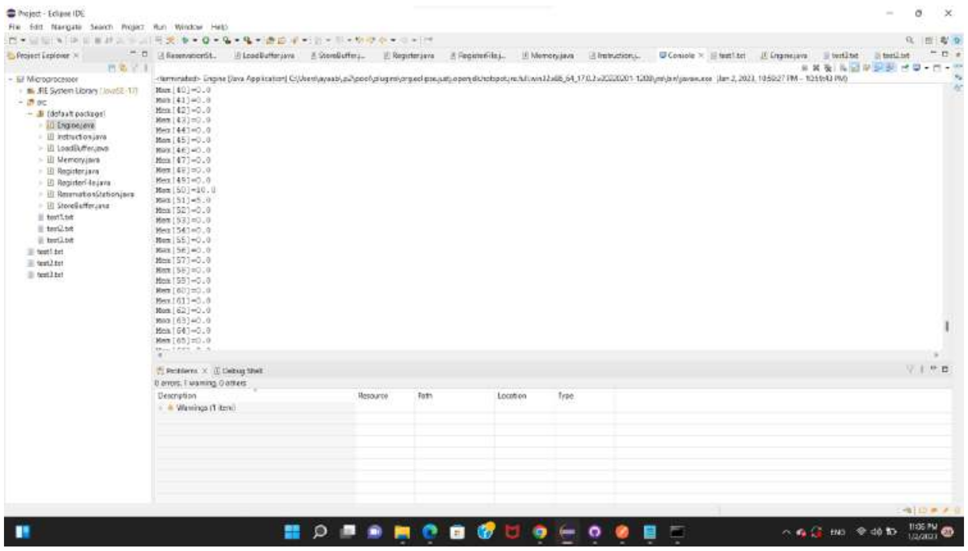
From eclipse:



```
F25     0.0
F26     0.0
F27     0.0
F28     0.0
F29     0.0
F30     0.0
F31     0.0
--------------------------------------------------------------------------------
Cycle: 25
Instruction   i1         i2         i3         Issue     execStart   execEnd      writeres
L.D           F1         50                    1         2           3            4
L.D           F2         51                    2         3           4            5
L.D           F3         52                    5         6           7            8
ADD.D         F8         F6         F7         6         7           8            9
SUB.D         F11        F9         F10        7         8           9            10
ADD.D         F14        F12        F13        10        11          12           13
MUL.D         F18        F6         F7         11        12          15           16
DIV.D         F19        F9         F10        12        13          22           23
MUL.D         F20        F12        F13        17        18          21           22
S.D           F15        52                    18        19          20           21
S.D           F16        52                    19        20          21           24
S.D           F17        52                    22        23          24           25

AddSub RS:
busy     op       Vj       Vk       Q1       Q2       time
```



```
Register File
F0      0.0
F1      10.0
F2      5.0
F3      39.0
F4      3.0
F5      2.0
F6      18.0
F7      12.0
F8      30.0
F9      18.0
F10     10.0
F11     9.0
F12     30.0
F13     23.0
F14     53.0
F15     12.0
F16     0.0
F17     0.0
F18     216.0
F19     1.0
F20     890.0
F21     0.0
F22     0.0
F23     0.0
F24     0.0
```



```
Mem[40]=0.0
Mem[41]=0.0
Mem[42]=0.0
Mem[43]=0.0
Mem[44]=0.0
Mem[45]=0.0
Mem[46]=0.0
Mem[47]=0.0
Mem[48]=0.0
Mem[49]=0.0
Mem[50]=10.0
Mem[51]=5.0
Mem[52]=0.0
Mem[53]=0.0
Mem[54]=0.0
Mem[55]=0.0
Mem[56]=0.0
Mem[57]=0.0
Mem[58]=0.0
Mem[59]=0.0
Mem[60]=0.0
Mem[61]=0.0
Mem[62]=0.0
Mem[63]=0.0
Mem[64]=0.0
Mem[65]=0.0
```

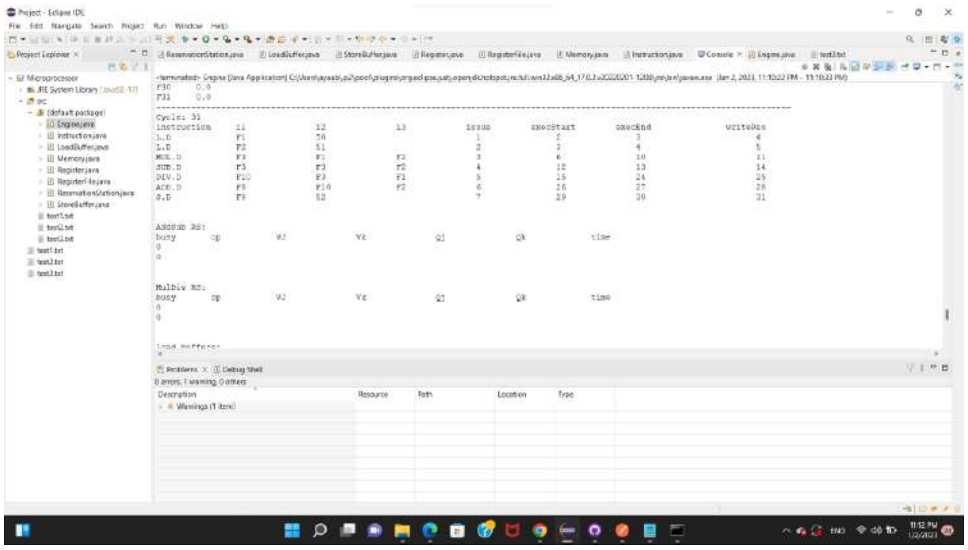
d) Fourth case is that instructions are dependent on each other (WAW, RAW)

(this uses the program test3.txt)

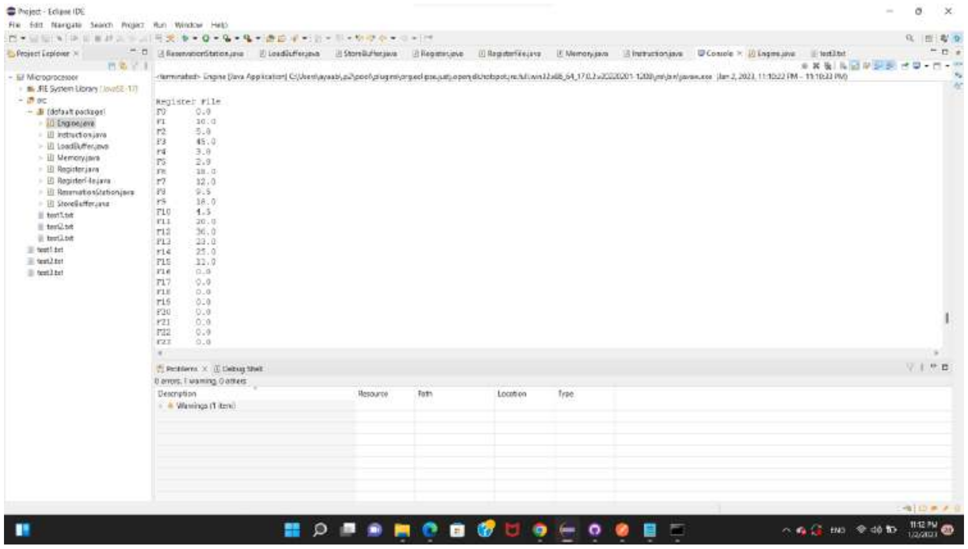
```
L.D F1 50

L.D F2 51

MUL.D F3 F1 F2

SUB.D F3 F3 F2

DIV.D F10 F3 F1

ADD.D F8 F10 F2

S.D F8 52
```

We assume here 2 add/sub reservations stations, 2 mul/div reservation stations, 2 load buffers, 2 store buffers, addition latency of 2, subtraction latency of 2, multiplication latency of 5, division latency of 10, load latency of 2, and store latency of 2.

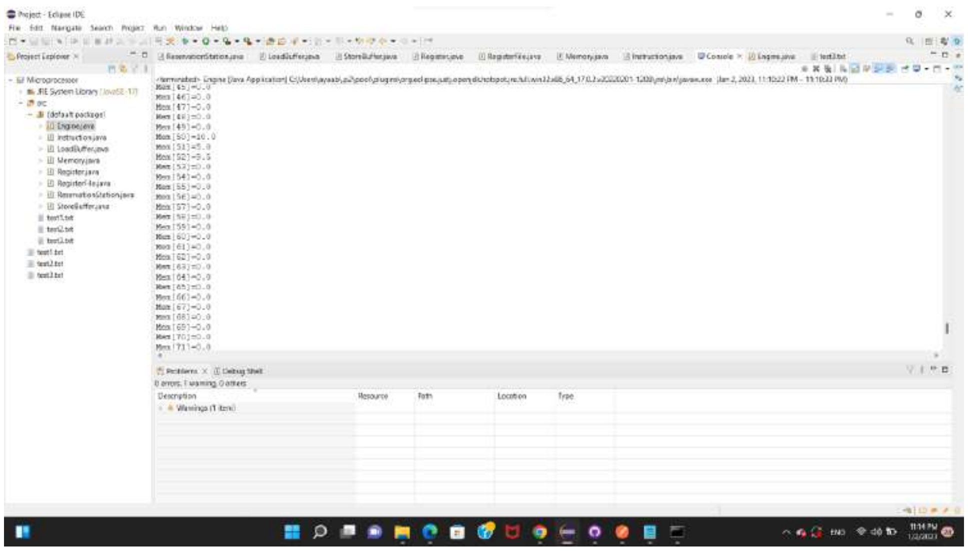
From eclipse:



```
F30     0.0
F31     0.0
------------------------------------------------------------------------------------------------
Cycle: 31
Instruction     i1          i2          i3          issue     execStart     execEnd       writeDes
L.D             F1          50                        1           2            3              4
L.D             F2          51                        2           3            4              5
MUL.D           F3          F1          F2            3           6            10             11
SUB.D           F3          F3          F2            4           12           13             14
DIV.D           F10         F3          F1            5           15           24             25
ADD.D           F8          F10         F2            6           26           27             28
S.D             F8          52                        7           29           30             31

AddSub RS:
busy      op        Vj        Vk        Qj        Qk        time
0
0


MulDiv RS:
busy      op        Vj        Vk        Qj        Qk        time
0
0


Load Buffers:
```



```
Register File
F0      0.0
F1      10.0
F2      5.0
F3      45.0
F4      3.0
F5      2.0
F6      18.0
F7      12.0
F8      9.5
F9      18.0
F10     4.5
F11     20.0
F12     36.0
F13     23.0
F14     25.0
F15     11.0
F16     0.0
F17     0.0
F18     0.0
F19     0.0
F20     0.0
F21     0.0
F22     0.0
F23     0.0
```



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 2, 2023, 11:10:22 PM – 11:10:33 PM)
Mem[45]=0.0
Mem[46]=0.0
Mem[47]=0.0
Mem[48]=0.0
Mem[49]=0.0
Mem[50]=10.0
Mem[51]=5.0
Mem[52]=9.5
Mem[53]=0.0
Mem[54]=0.0
Mem[55]=0.0
Mem[56]=0.0
Mem[57]=0.0
Mem[58]=0.0
Mem[59]=0.0
Mem[60]=0.0
Mem[61]=0.0
Mem[62]=0.0
Mem[63]=0.0
Mem[64]=0.0
Mem[65]=0.0
Mem[66]=0.0
Mem[67]=0.0
Mem[68]=0.0
Mem[69]=0.0
Mem[70]=0.0
Mem[71]=0.0
```

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				

e) Fifth case is that instructions are dependent on each other, while also not necessarily finding an empty reservation station directly.

(this uses the program test4.txt)

MUL.D F3 F1 F2

ADD.D F5 F3 F4

ADD.D F7 F2 F6

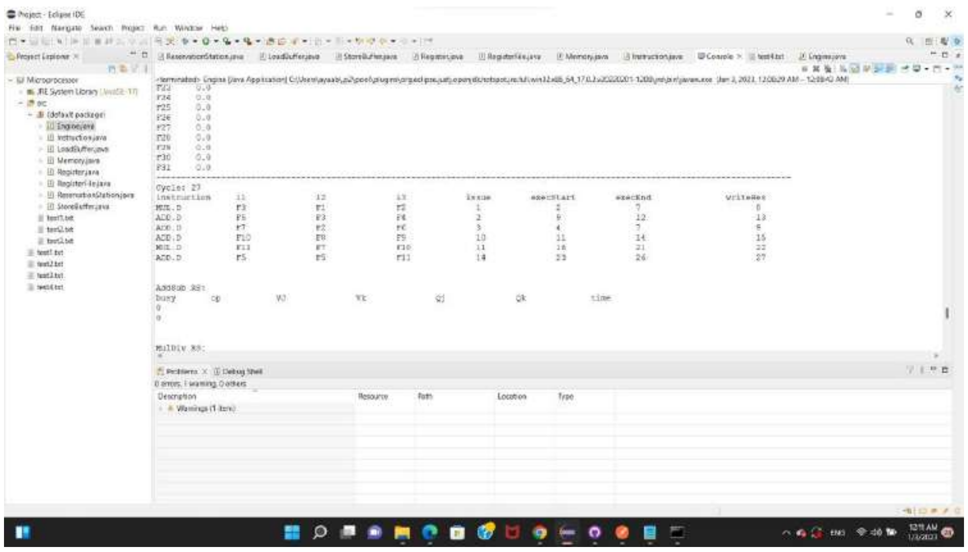
ADD.D F10 F8 F9

MUL.D F11 F7 F10

ADD.D F5 F5 F11

We assume here 2 add/sub reservations stations, 2 mul/div reservation stations, 2 load buffers, 2 store buffers, addition latency of 4, subtraction latency of 4, multiplication latency of 6, division latency of 10, load latency of 2, and store latency of 2.

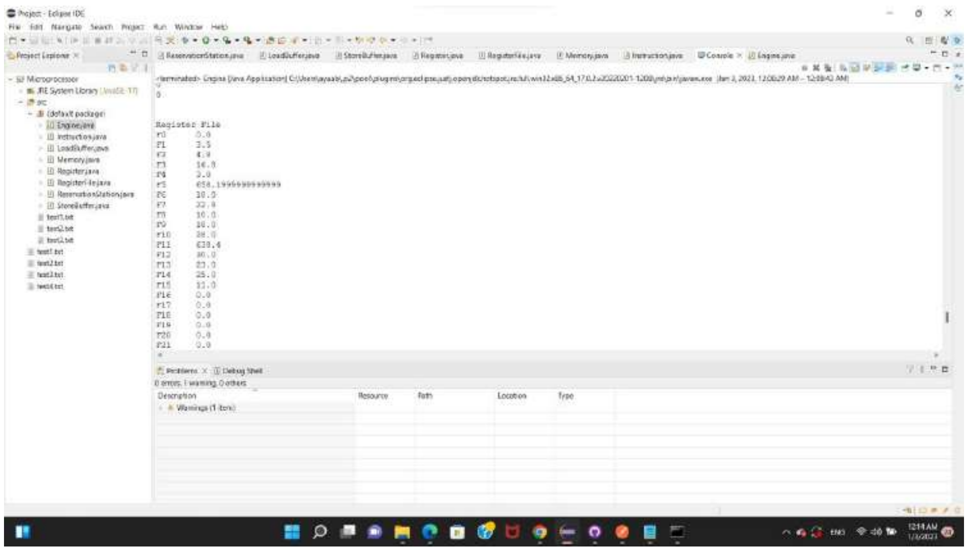
From eclipse:



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 3, 2023, 12:00:29 AM – 12:08:42 AM)
F23     0.0
F24     0.0
F25     0.0
F26     0.0
F27     0.0
F28     0.0
F29     0.0
F30     0.0
F31     0.0
-------------------------------------------------------------------------------------------------------------------
Cycle: 27
instruction     i1          i2          i3          issue       execStart     execEnd        writeRes
MUL.D           F3          F1          F2          1           2             7              8
ADD.D           F5          F3          F4          2           9             12             13
ADD.D           F7          F2          F6          3           4             7              8
ADD.D           F10         F8          F9          10          11            14             15
MUL.D           F11         F7          F10         11          16            21             22
ADD.D           F5          F5          F11         14          23            26             27


AddSub RS:
busy       op          Vj          Vk          Qj          Qk          time
0
0


MulDiv RS:
```

Problems × Debug Shell
0 errors, 1 warning, 0 others

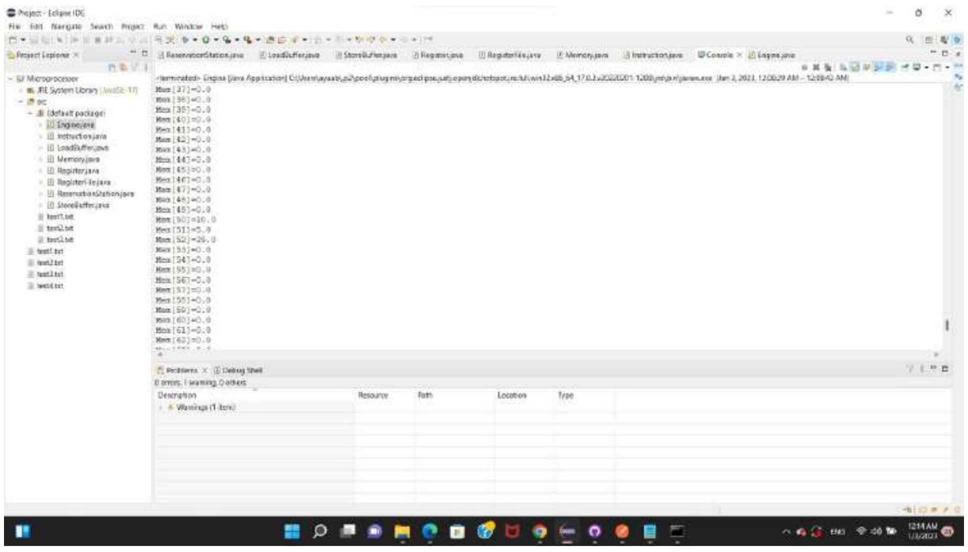
Description	Resource	Path	Location	Type
Warnings (1 item)				



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 3, 2023, 12:00:29 AM – 12:08:42 AM)
0


Register File
F0      0.0
F1      3.5
F2      4.9
F3      16.8
F4      3.0
F5      658.1999999999999
F6      10.0
F7      32.8
F8      10.0
F9      18.0
F10     28.0
F11     628.4
F12     30.0
F13     23.0
F14     25.0
F15     11.0
F16     0.0
F17     0.0
F18     0.0
F19     0.0
F20     0.0
F21     0.0
```

Problems × Debug Shell
0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				



```
<terminated> Engine [Java Application] C:\Users\ayaab\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Jan 3, 2023, 12:00:29 AM – 12:08:42 AM)
Mem[37]=0.0
Mem[38]=0.0
Mem[39]=0.0
Mem[40]=0.0
Mem[41]=0.0
Mem[42]=0.0
Mem[43]=0.0
Mem[44]=0.0
Mem[45]=0.0
Mem[46]=0.0
Mem[47]=0.0
Mem[48]=0.0
Mem[49]=0.0
Mem[50]=10.0
Mem[51]=5.0
Mem[52]=26.0
Mem[53]=0.0
Mem[54]=0.0
Mem[55]=0.0
Mem[56]=0.0
Mem[57]=0.0
Mem[58]=0.0
Mem[59]=0.0
Mem[60]=0.0
Mem[61]=0.0
Mem[62]=0.0
```

Problems × Debug Shell
0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				