

# **Tomasulo scheduling**

Seif Hossam 52-0350

Abdelrahman Ahmed 52-3505

Micheal Mokhles 52-2717

Paula Bassem 52-0416

Youssef Khodeir 52-4833

## Project overview

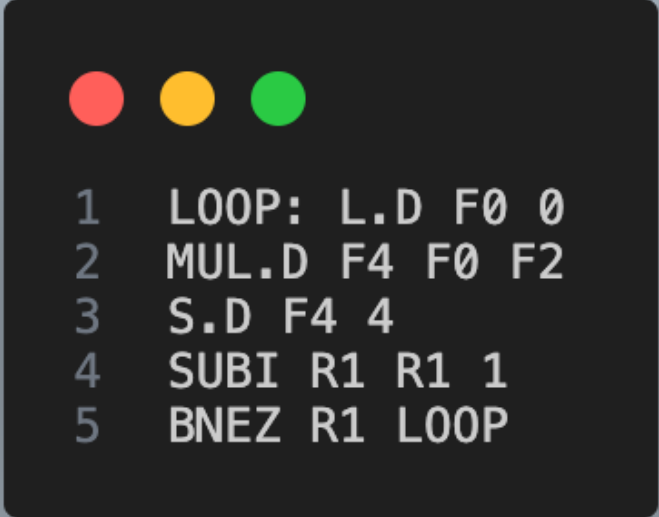
The project is made using Object Oriented Programming , So we will start with explaining each class first and then we will explain our main.java file content which is our main concern. The files to be explained are:

- Config.txt
- Register.java
- FileofRegisters.java
- Instruction.java
- LoadBuffer.java
- Memory.java
- ReservationArea.java
- StoreBuffer.java
- Main.java

---

### Config.txt

Config file is the text file where we read instructions from , each register is separated by a space



```
1  LOOP: L.D F0 0
2  MUL.D F4 F0 F2
3  S.D F4 4
4  SUBI R1 R1 1
5  BNEZ R1 LOOP
```

## Register.java

Java class that is made in order to define a Register with a name , value and Q

```
1  public class Register {
2      private String name;
3      private float value;
4      private String queue;
5
6      public Register(String name, float value, String queue) {
7          super();
8          this.name = name;
9          this.value = value;
10         this.queue = queue;
11     }
12 }
```

```
1  public String getName() {
2      return name;
3  }
4
5  public float getValue() {
6      return value;
7  }
8
9  public void setValue(float value) {
10     if (!this.name.equals("R0"))
11         this.value = value;
12 }
13
14 public String getQueue() {
15     return queue;
16 }
17
18 public void setQueue(String queue) {
19     this.queue = queue;
20 }
21
22 @Override
23 public String toString() {
24     return "Register [name=" + name + ", value=" + value + "];"
25 }
26 }
```

## FileOfRegisters.java

Considering that we are working with Mips 64 there are 32 Integer registers and 32 Floating point registers. In the first snippet of the code we are creating adding new registers which are the Program Counter (PC) , F0 (The first floating Point register) and R0 (first Integer register)  
N.B: all the 64 registers are initialized yet not displayed in this snippet , there are also functions that get the value of the register , sets the value and sets the Q

```
1 public class FileOfRegisters extends ArrayList<Register> {
2     public FileOfRegisters() {
3         this.add(new Register("R0", 0, "0"));
4         this.add(new Register("F0", 1, "0"));
5         this.add(new Register("PC", 0, "0"));
6     }
7 }
```

```
1 public float getValueRegister(String registerName){
2     for(int i=0;i<this.size();i++){
3         if(this.get(i).getName().equals(registerName)){
4             return this.get(i).getValue();
5         }
6     }
7     return -1;
8 }
9
10 public void setValue(int index, int value) {
11     this.get(index).setValue(value);
12 }
13
14 public void setQueue(int index, String queue) {
15     this.get(index).setQueue(queue);
16 }
17 }
```

## Instruction.java

A instruction class that takes as an input an instruction ID , reservation name, opcode , destination , source 1 , source 2 , issue (issue time) , startExec (start time for execution), endExec(end time for execution) ,writeResultClock(write time) , Result(Value after execution), Inexecution (Boolean flag)

```
1 public class Instruction {
2     private static int id = 1;
3     int instructionId;
4     String reservationName;
5     String opcode;
6     String dest;
7     String src1;
8     String src2;
9     int issue;
10    int startExec;
11    int endExec;
12    int writeResultClock;
13    float result;
14    boolean inExecution;
15
16    public Instruction(String opcode, String dest, String src1, String src2) {
17        this.instructionId = id++;
18        this.opcode = opcode;
19        this.dest = dest;
20        this.src1 = src1;
21        this.src2 = src2;
22        this.issue = -1;
23        this.startExec = -1;
24        this.endExec = -1;
25        this.writeResultClock = -1;
26        this.result = 0f;
27        this.inExecution = false;
28        this.reservationName = "";
29    }
30
31    public String toString() {
32        return "Instruction: " + this.instructionId + " Opcode: " + this.opcode + " Dest: " + this.dest + " Src1: "
33            + this.src1 + " Src2: "
34            + this.src2 + " Issue: "
35            + this.issue + " StartExec: " + startExec + " End Exec: "
36            + endExec + " Write Result Clock: " + writeResultClock;
37    }
38 }
39 }
```

## Load and Store buffers

```
1 public class LoadBuffer {
2     private int instructionId;
3     private String reservationAreaId;
4     private String Address;
5     private int busy;
6
7     public LoadBuffer(int instructionId,String reservationAreaId,int busy, String Address) {
8         this.instructionId = instructionId;
9         this.reservationAreaId = reservationAreaId;
10        this.Address = Address;
11        this.busy = busy;
12    }
13
14    public int getInstructionId() {
15        return instructionId;
16    }
17
18    public String getReservationAreaId() {
19        return reservationAreaId;
20    }
21
22    public String getAddress() {
23        return Address;
24    }
25
26    public int getBusy() {
27        return busy;
28    }
29
30    public void setAddress(String Address) {
31        this.Address = Address;
32    }
33
34    public void setBusy(int busy) {
35        this.busy = busy;
36    }
37
38    public void setInstructionId(int instructionId) {
39        this.instructionId = instructionId;
40    }
41
42    public void setReservationAreaId(String reservationAreaId) {
43        this.reservationAreaId = reservationAreaId;
44    }
45
46 }
47
```

```
1 public class StoreBuffer {
2     private int instructionId;
3     private String reservationAreaId;
4     private String Address;
5     private float value;
6     private String queue;
7     private int busy;
8
9     public StoreBuffer(int instructionId,String reservationAreaId, int busy, String Address, float value, String queue) {
10        this.instructionId = instructionId;
11        this.reservationAreaId = reservationAreaId;
12        this.Address = Address;
13        this.value = value;
14        this.queue = queue;
15        this.busy = busy;
16    }
17
18    public int getInstructionId() {
19        return instructionId;
20    }
21
22    public String getReservationAreaId() {
23        return reservationAreaId;
24    }
25
26    public String getAddress() {
27        return Address;
28    }
29
30    public float getValue() {
31        return value;
32    }
33
34    public String getQueue() {
35        return queue;
36    }
37
38    public int getBusy() {
39        return busy;
40    }
41
42    public void setAddress(String Address) {
43        this.Address = Address;
44    }
45
46    public void setInstructionId(int instructionId) {
47        this.instructionId = instructionId;
48    }
49
50    public void setValue(float value) {
51        this.value = value;
52    }
53
54    public void setQueue(String queue) {
55        this.queue = queue;
56    }
57
58    public void setBusy(int busy) {
59        this.busy = busy;
60    }
61
62    public void setReservationAreaId(String reservationAreaId) {
63        this.reservationAreaId = reservationAreaId;
64    }
65
66 }
```

Initialization for new Load buffer and store buffer Classes that take inputs (InstructionId , ReservationAreaId , Address , Busy ) for the Load buffer and (InstructionId , ReservationAreaId , Address , Value, Q ,Busy ) for the Store Buffers

---

## Memory

An array containing 2048 strings that denotes free spaces in the memory , it also has a counter that acts as pointer

## Reservation Area

A general class called reservation area that's called in the main class in order to create Add and multiply Reservation areas

```

1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  public class Memory {
7      String[] memory;
8      int counter = 0;
9      // ArrayList<potentialHazard> hazards = new ArrayList<potentialHazard>();
10     // ArrayList<jumpCondition> jumps = new ArrayList<jumpCondition>();
11
12     public Memory() {
13         memory = new String[2048];
14         for(int i=1024;i<1024+64;i++){
15             memory[i] = "0";
16             if(i==1024+0)
17                 memory[i] = "1";
18             if(i==1024+4)
19                 memory[i] = "0";
20         }
21         try {
22             readassembltfile();
23         } catch (FileNotFoundException e) {
24             e.printStackTrace();
25         }
26     }
27
28     public void readassembltfile() throws FileNotFoundException {
29         BufferedReader reader;
30         int counter = 0;
31
32         try {
33             reader = new BufferedReader(new FileReader("config.txt"));
34             String line = reader.readLine();
35
36             while (line != null) {
37                 if (counter == 1024) {
38                     break;
39                 }
40                 // String[] parts = line.split(" ");
41                 memory[counter] = line;
42
43                 line = reader.readLine();
44                 counter++;
45             }
46             reader.close();
47             this.counter = counter;
48         } catch (IOException e) {
49             e.printStackTrace();
50             System.out.println("File does not exist.");
51         }
52     }

```

```

1  public class ReservationArea {
2      int instructionId;
3      String reservationAreaId;
4      String opcode;
5      float value_j;
6      float value_k;
7      String queue_j;
8      String queue_k;
9      int busy;
10

```

## Main

The main code can be divided into 4 parts : Fetch , Issue, execute and Write . We will explain these parts thoroughly

### Fetch

Initially we check the memory pointer if it's less than the memory counter which infers the number of instructions and the stall state then we create a new instruction and add it to the table , set and queue of instructions

```
1  public static void fetch() {
2
3      if (pointerCache < memory.counter && !stall) {
4          String part = memory.memory[pointerCache];
5          String[] parts = part.split(" ");
6          Instruction instruction;
7
8          int isLoop = 0;
9
10         if (!(parts[0].equals("ADDI") || parts[0].equals("SUBI") || parts[0].equals("BNEZ")
11             || parts[0].equals("L.D")
12             || parts[0].equals("S.D") || parts[0].equals("MUL.D") || parts[0].equals("DIV.D")
13             || parts[0].equals("ADD.D") || parts[0].equals("SUB.D")
14             || parts[0].equals("DADD")))
15             isLoop = 1;
16
17         if (parts.length == 3 && isLoop == 0)
18             instruction = new Instruction(parts[0], parts[1], parts[2], "");
19         else if (parts.length == 4 && isLoop == 1)
20             instruction = new Instruction(parts[isLoop + 0], parts[isLoop + 1], parts[isLoop + 2], "");
21         else if (parts.length == 4 && isLoop == 0)
22             instruction = new Instruction(parts[0], parts[1], parts[2], parts[3]);
23         else
24             instruction = new Instruction(parts[isLoop + 0], parts[isLoop + 1], parts[isLoop + 2],
25                 parts[isLoop + 3]);
26
27         tableOfInstructions.add(instruction);
28         setOfInstructions.add(instruction);
29         queueInstructions.add(instruction);
30         pointerCache++;
31     }
32 }
```

## Issue

In the issue part we check if the peek of instructions queue is equal to null meaning that there are no more instructions to be executed else we check the opcode of the instruction and then check if the reservation station is full or not and if there's at least one empty reservation station you insert into one empty reservation station. We can see a snippet for the case if the opcode is mul.d or div.d and in the second image we insert this newly issued instruction into the registers file in order to set the Q with the value of the issued instruction which means that it's waiting for the execution and then remove it from the queue of instructions.

```

1  else if (instruction.opcode.equals("MUL.D") || instruction.opcode.equals("DIV.D")) {
2      for (int i = 0; i < MultiplyStation.length; i++) {
3          if (MultiplyStation[i].busy == 0) {
4              MultiplyStation[i].busy = 1;
5              MultiplyStation[i].instructionId = instruction.instructionId;
6              MultiplyStation[i].opcode = instruction.opcode;
7              for (int j = 0; j < fileOfRegisters.size(); j++) {
8                  if (instruction.src1.equals(fileOfRegisters.get(j).getName())) {
9                      if (fileOfRegisters.get(j).getQueue().equals("")) {
10                         MultiplyStation[i].value_j = fileOfRegisters.get(j).getValue();
11                         MultiplyStation[i].queue_j = "0";
12                     } else {
13                         MultiplyStation[i].queue_j = fileOfRegisters.get(j).getQueue();
14                     }
15                 }
16                 if (instruction.src2.equals(fileOfRegisters.get(j).getName())) {
17                     if (fileOfRegisters.get(j).getQueue().equals("")) {
18                         MultiplyStation[i].value_k = fileOfRegisters.get(j).getValue();
19                         MultiplyStation[i].queue_k = "0";
20                     } else {
21                         MultiplyStation[i].queue_k = fileOfRegisters.get(j).getQueue();
22                     }
23                 }
24             }
25             indexReservation = i;
26             isIssued = true;
27             break;
28         }
29     }
30 }

```

```

1  if (isIssued) {
2      instruction.issue = Main.clk;
3      for (int i = 0; i < fileOfRegisters.size(); i++) {
4          if (fileOfRegisters.get(i).getName().equals(instruction.dest)) {
5              String reservationString = "";
6              if (instruction.opcode.equals("S.D"))
7                  reservationString = "S" + indexReservation;
8              else if (instruction.opcode.charAt(0) == 'S' || instruction.opcode.charAt(0) == 'A'
9                      || instruction.opcode.charAt(0) == 'B')
10                 reservationString = "A" + indexReservation;
11              else if (instruction.opcode.charAt(0) == 'L')
12                 reservationString = "L" + indexReservation;
13              else if (instruction.opcode.charAt(0) == 'M' || instruction.opcode.charAt(0) == 'D')
14                 reservationString = "M" + indexReservation;
15
16              instruction.reservationName = reservationString.charAt(0) + "";
17              fileOfRegisters.get(i).setQueue(reservationString);
18              break;
19          }
20      }
21      queueInstructions.remove();
22      if (instruction.opcode.equals("BNEZ"))
23          stall = true;
24  }

```



## Execute

We check every reservation station and check the set of instructions, we check if there's no dependency meaning that both  $Q_i$  and  $Q_j$  are equal to 0, and in this snippet we check afterwards the opcode and if it's not in the execution stage already and if it hasn't just been issued we then initialize the values for start exec and end exec with the latency based on each operation and set the `inExecution` flag to true and after checking all the reservation stations we start execution itself by checking if the start execution time is not equal to -1 meaning that it actually started execution and then execute based on the respective opcode

```

1  for (int i = 0; i < AddStation.length; i++) {
2      if (AddStation[i].busy == 1) {
3          if (AddStation[i].getQueue_j().equals("0") &&
4              AddStation[i].getQueue_k().equals("0")) {
5              if (AddStation[i].opcode.equals("ADD.D") || AddStation[i].opcode.equals("SUB.D")) {
6                  // float result = 0;
7                  // result = AddStation[i].value_j + AddStation[i].value_k;
8                  for (int j = 0; j < setOfInstructions.size(); j++) {
9                      if (setOfInstructions.get(j).instructionId == AddStation[i].getInstructionId()
10                         && !setOfInstructions.get(j).inExecution
11                         && setOfInstructions.get(j).issue != Main.clk
12                         && setOfInstructions.get(j).endExec == -1) {
13                          // setOfInstructions.get(j).result = result;
14                          setOfInstructions.get(j).startExec = Main.clk;
15                          setOfInstructions.get(j).endExec = Main.clk + latencyAddD - 1;
16                          setOfInstructions.get(j).inExecution = true;
17                          break;
18                      }
19                  }
20              }
21          }
22      }
23  }

```

```

1  for (int i = 0; i < setOfInstructions.size(); i++) {
2      if (setOfInstructions.get(i).endExec == Main.clk && setOfInstructions.get(i).writeResultClock == -1
3          && setOfInstructions.get(i).startExec != -1) {
4
5          for (int j = 0; j < fileOfRegisters.size(); j++) {
6              if (setOfInstructions.get(i).dest.equals(fileOfRegisters.get(j).getName())) {
7                  if (setOfInstructions.get(i).opcode.equals("ADD.D")
8                      || setOfInstructions.get(i).opcode.equals("DADD")) {
9                      setOfInstructions.get(i).result = fileOfRegisters
10                         .getValueRegister(setOfInstructions.get(i).src1)
11                         + fileOfRegisters.getValueRegister(setOfInstructions.get(i).src2);
12                  }
13              }
14          }
15      }
16  }

```


## Write

In the write stage we check if the writeResultClock is equal to -1 meaning that it didn't write yet and if it took all the cycles to execute, we will get that instruction and then check all the reservation stations, store buffers if there's any dependency and remove it i.e clear every Q that has the same name to be equal to zero and set V\_i or V\_j to be equal to the value of the dependency that the instruction was waiting for and change the file of registers also

```
1 public static void write() {
2     Instruction myInstruction;
3     for (int i = 0; i < setOfInstructions.size(); i++) {
4         if (setOfInstructions.get(i).endExec < Main.clk && setOfInstructions.get(i).writeResultClock == -1
5             && !setOfInstructions.get(i).inExecution && setOfInstructions.get(i).startExec != -1) {
6             setOfInstructions.get(i).writeResultClock = Main.clk;
7             myInstruction = setOfInstructions.get(i);
8
9             for (int j = 0; j < fileOfRegisters.size(); j++) {
10                if (setOfInstructions.get(i).dest.equals(fileOfRegisters.get(j).getName())) {
11
12                    if (!setOfInstructions.get(i).opcode.equals("S.D")) {
13                        fileOfRegisters.get(j).setValue(setOfInstructions.get(i).result);
14                        fileOfRegisters.get(j).setQueue("0");
15                        break;
16                    }
17                }
18            }
19        }
20    }
```

## Test cases

### Test case 1:



```

1  LOOP: L.D F0 0
2  MUL.D F4 F0 F2
3  S.D F4 4
4  SUBI R1 R1 1
5  BNEZ R1 LOOP
  
```


### Test Case 3:

### Test case 2:



```

1  L.D F6 2
2  L.D F2 5
3  MUL.D F0 F2 F4
4  SUB.D F8 F2 F6
5  MUL.D F10 F0 F6
6  ADD.D F6 F8 F2
  
```



```

1  LOOP: L.D F00
2  MUL.D F4 F0 F2
3  S.D F4 4
4  SUBI R1 R1 1
5  BNEZ R1 LOOP
  
```