The American University in Cairo

Ali Yassine - 900204483

Aref Mamdouh - 900192254

Ibrahim Soliman - 900192478

Project Report

Professor Cherif Salama - Computer Architecture

Fall 2022

Abstract:

This project simulates Tomasulo's Algorithm in issuing, executing and writing back instructions.

It is done using c++ and supports the following instructions.

- ADD
- ADDI
- MUL
- RET
- JAL
- BEQ
- NEG
- NOR
- LOAD
- STORE

About the Implementation:

Components:

map<int,int> Memory

Struct ReservationStation

vector<ReservationStation> reservationStations

vector<Instruction>InstructionMemory

vector<Instruction>InstructionPool

Struct RegisterFile

Struct CDB

Functions:

Void readFile():

The readfile function parses the input text file that contains the instructions. It creates an object of the struct instruction, assigns to it the needed attributes for this specific instruction and pushes back all the instructions into a vector of structs **"Instruction Memory**".

Int getNextRs(string op):

This function compares the operation of the instruction with the operation of the reservation stations and when the latter match, if the matched station is not busy the station gets indexed and is assigned to this instruction.

Void startIssuing(Reservation station, stationindex,instruction):

```
    bool needsRs1 = false;
    bool needsRs2 = false;
    bool takesImm = false;
    bool hasRetrun = false;
    bool haveJump = false;
    bool foundBranch = false;
```

According to the operation of the instruction being issued, we assign these booleans to 0 or 1. Then according to the value of each boolean we set Qj,Qk,Vj,Vk and A of the reservation station.

Example:

```
if (needsRs1)
    {
        station.qj = rf.getProducingUnit(instruction.rs1);
        if (station.qj == 0)
            station.vj = rf.getValue(instruction.rs1);

    }
    if (needsRs2)
    {
        station.qk = rf.getProducingUnit(instruction.rs2);
        if (station.qk == 0)
            station.vk = rf.getValue(instruction.rs2);
    }
    if (takesImm)
```

Void Issue():

Calls getNextRs and startIssuing.

Void finishExecution(ResevationStation):

Here we set the boolean finished to true, we set the cycle that the instruction ended at and we do the computation of each instruction according to the operation. The computation result is saved in an attribute “result” of the reservation station.

Void execute():

Loop over reservation stations and check if the station is busy. If it is busy then we have several options. If the station is after a branch we continue to the next station. If the station has already started execution and has not finished yet, we decrement the needed cycles by 1.And when the needed cycles become 0 we call the function “finished execution”. The third option is if the station has not finished is not executing, thus it is its first cycle in execution.Here we set executing boolean to true, we decrement the needed cycles and set the time that the execution started.

Bool CheckWAW(ReservationStation):

Checks for Write after write hazard

Void flushUneededInstructions():

If a station came after a branch, where in our case the maximum number of stations after a branch is one since branching takes one cycle to execute, then this function resets this station with a function reset in the struct of the reservation station.

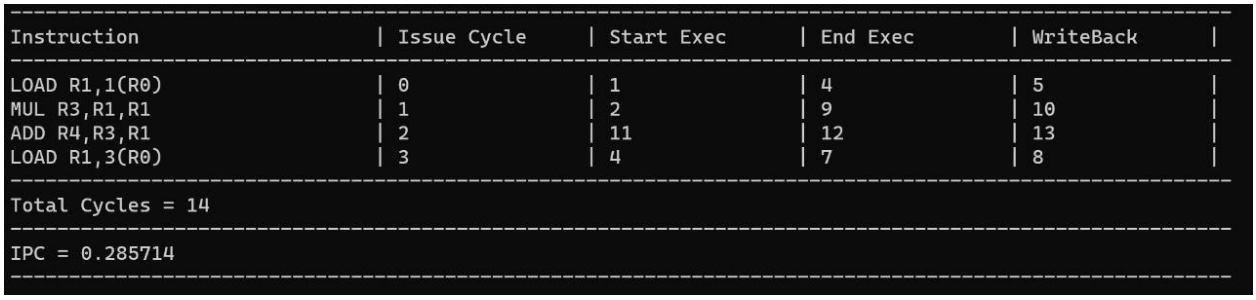
Void write(ReservationStation, stationindex):

Here we set the value of the result we computed into its register in the register file, flushes if needed and passes the data (result and source) to the data bus.

Void writeback():

We loop over the reservation stations and check if the station finished execution or not using the boolean “finished”. If a station did not finish we continue, else we check for WAW hazard, if there is not then we write using the function “write”. We then loop over stations to see if the station needs values from the common data bus.

Snapshots:



Instruction	Issue Cycle	Start Exec	End Exec	WriteBack
LOAD R1,1(R0)	0	1	4	5
MUL R3,R1,R1	1	2	9	10
ADD R4,R3,R1	2	11	12	13
LOAD R1,3(R0)	3	4	7	8

Total Cycles = 14

IPC = 0.285714

The above results make sense. We can see issuing is done in the first 4 cycles since reservation stations can issue all the instructions after each other since there are enough reservations stations available. Load then takes 4 execution cycles from the beginning of 1 till the ending of 4 (4 here means the ending of 4 and 5 in writeback mean the ending of 5).MUL is issued in the next cycle after issuing LOAD , starts executing in cycle 2 and takes 8 cycles indeed.

Since ADD uses R3 which is the destination register of MUL, the execution of ADD only starts after MUL writebacks as we can see.

Proof that ADDI is working:

Instruction	Issue Cycle	Start Exec	End Exec	WriteBack
LOAD R1,1(R0)	0	1	4	5
MUL R3,R1,R1	1	2	9	10
ADDI R4,R3,2	2	11	12	13
LOAD R1,3(R0)	3	4	7	8

Total Cycles = 14

IPC = 0.285714

In the snapshot below we can see that the second MUL waits for the first MUL to finish from the reservation station before it gets issued.

```
MUL R2,R2,R2
ADDI R2,R2,1
MUL  R3,R1,R1
```

Instruction	Issue Cycle	Start Exec	End Exec	WriteBack
MUL R2,R2,R2	0	1	8	9
ADDI R2,R2,1	1	10	11	12
MUL R3,R1,R1	10	11	18	19

Total Cycles = 20

IPC = 0.150000

Branch Mispredictions = 0.000000

This snapshot below is proof that NEG is working:

```
NEG  R3,R1
ADD R2,R2,R2
ADDI R2,R2,1
```

Instruction	Issue Cycle	Start Exec	End Exec	WriteBack
NEG R3,R1	0	1	1	2
ADD R2,R2,R2	1	2	3	4
ADDI R2,R2,1	2	5	6	7

Total Cycles = 8

IPC = 0.375000

Branch Mispredictions = 0.000000

Proof that NOR is working:

```
NOR R3,R1,R2
ADD R2,R2,R2
ADDI R2,R2,1
-------------------------------------------------------------------------------------------------------
Instruction                    | Issue Cycle     | Start Exec      | End Exec        | WriteBack      |
-------------------------------------------------------------------------------------------------------
NOR R3,R1,R2                   | 0               | 1               | 1               | 2              |
ADD R2,R2,R2                   | 1               | 2               | 3               | 4              |
ADDI R2,R2,1                   | 2               | 5               | 6               | 7              |
-------------------------------------------------------------------------------------------------------
Total Cycles = 8
-------------------------------------------------------------------------------------------------------
IPC = 0.375000
-------------------------------------------------------------------------------------------------------
Branch Mispredictions = 0.000000
-------------------------------------------------------------------------------------------------------
```

Proof that Store is working:

```
NOR R3,R1,R2
ADD R2,R2,R2
ADDI R2,R2,1
MUL R2,R2,R2
STORE R2,1(R1)
MUL  R3,R1,R1
-------------------------------------------------------------------------------------------------------
Instruction                    | Issue Cycle     | Start Exec      | End Exec        | WriteBack      |
-------------------------------------------------------------------------------------------------------
NOR R3,R1,R2                   | 0               | 1               | 1               | 2              |
ADD R2,R2,R2                   | 1               | 2               | 3               | 4              |
ADDI R2,R2,1                   | 2               | 5               | 6               | 7              |
MUL R2,R2,R2                   | 3               | 8               | 15              | 16             |
STORE R2,1(R1)                 | 4               | 17              | 20              | 21             |
MUL  R3,R1,R1                  | 17              | 18              | 25              | 26             |
-------------------------------------------------------------------------------------------------------
Total Cycles = 27
-------------------------------------------------------------------------------------------------------
IPC = 0.222222
-------------------------------------------------------------------------------------------------------
Branch Mispredictions = 0.000000
-------------------------------------------------------------------------------------------------------
```

General Example with 6 instructions:

```
NOR R3,R1,R2
ADD R2,R2,R2
ADDI R2,R2,1
MUL R2,R2,R2
ADDI R2,R2,1
MUL  R3,R1,R1
```

Instruction	Issue Cycle	Start Exec	End Exec	WriteBack
NOR R3,R1,R2	0	1	1	2
ADD R2,R2,R2	1	2	3	4
ADDI R2,R2,1	2	5	6	7
MUL R2,R2,R2	3	8	15	16
ADDI R2,R2,1	4	17	18	19
MUL R3,R1,R1	17	18	25	26

```
Total Cycles = 27
IPC = 0.222222
Branch Mispredictions = 0.000000
```

A problem faced with JAL and BEQ:

```
ADD R3,R1,R2
JAL R2,2
ADDI R2,R2,1
MUL R2,R2,R2
MUL  R3,R1,R1
Jumping from 2 to 0
```

Instruction	Issue Cycle	Start Exec	End Exec	WriteBack
ADD R3,R1,R2	0	1	2	3
JAL R2,2	1	2	3	4

```
Total Cycles = 5
IPC = 0.400000
Branch Mispredictions = 0.000000
```

Problem faced with RET.

```
ADD R3,R1,R2
ADDI R2,R2,1
MUL R2,R2,R2
MUL R3,R1,R1
RET
-----------------------------------------------------------------------------------------------------------
Instruction                    | Issue Cycle     | Start Exec      | End Exec        | WriteBack      |
-----------------------------------------------------------------------------------------------------------
ADD R3,R1,R2                   | 0               | 1               | 2               | 3              |
ADDI R2,R2,1                   | 1               | 2               | 3               | 4              |
MUL R2,R2,R2                   | 2               | 5               | 12              | 13             |
-----------------------------------------------------------------------------------------------------------
Total Cycles = 14
-----------------------------------------------------------------------------------------------------------
IPC = 0.214286
-----------------------------------------------------------------------------------------------------------
Branch Mispredictions = 0.000000
-----------------------------------------------------------------------------------------------------------
```