CSCE 3301 Project 2 Report

Yara Abdelkader

Ahmed Bamadhaf

Github repo: https://github.com/Yara-5/Computer-Architecture-Project-2.git

Design and Implementation

The project consists of two primary source files: **main.cpp** and **ROB.cpp**.
Together, they implement a simplified Tomasulo-style CPU simulator with reservation stations, a reorder buffer, dynamic scheduling, load/store forwarding, and configurable execution parameters.

1. main.cpp (The Simulator)

This file contains the full pipeline simulation.
Its structure is divided into seven logical phases:

1.1 Main Object Initialization

- Creates the reservation stations, ROB, register file, memory array, and program memory.
- Initializes data structures that track:
    - Register renaming (Qi table)
    - Execution timers for each instruction
    - Store buffers and memory values
- Calls chooseVariables() which implements **Bonus Feature #1**:
    - User-configurable ROB size
    - Variable execution cycle counts for ALU, Load, and Store
    - Adjustable number of reservation stations

1.2 Instruction Issue

- Fetches instructions from program memory using pc.
- Allocates a free ROB entry.
- Allocates an appropriate reservation station based on the instruction type (ALU, Load, Store, Branch…).
- Performs register renaming by updating Qi table.

1.3 Execution Stage

- Checks each reservation station.
- Begins execution when operands are ready (Qj == -1, Qk == -1 or forwarded).
- Decrements execution cycle counters.
- Includes special handling for loads:
    - Load–store dependency checking
    - Forwarding values from earlier stores
    - Delaying loads if a conflicting store is still pending

1.4 Write-Back Stage

- When execution finishes, results are broadcast on the CDB.
- ROB entries are marked ready.
- Reservation stations dependent on this result update their operands.

1.5 Commit Stage

- Commits instructions from the ROB **in order**, ensuring:
    - Register writes occur only at commit time
    - Stores update memory only when they reach the head of the ROB
    - Branches and jumps can flush the pipeline if mispredicted

1.6 Logging

- Every cycle produces logs for each instruction issued, began/finished executing, written back or committed.

1.7 Simulator Loop

The main loop repeatedly performs:

1) Issue → Execute → Write Back → Commit → Increment Cycle

The loop terminates only when:

- All ROB entries are empty, **and**
- pc has exceeded the program size, **and**
- No reservation station is still busy (satisfied by empty ROB).

2. ROB.cpp (Reorder Buffer Implementation)

This file defines the **ROB class**, which encapsulates:

- The vector of ROB entries
- Methods to allocate, mark ready, and commit entries
- Accessors such as:
    - addEntry(...)
    - markReady(index)
    - canCommit(index)
    - getDestination(index)
    - getValue(index)
    - getInstructionId(index)

    - isEmpty()
- Helper logic for:
    - Load/store hazard resolution
    - Choosing the nearest previous store for forwarding

The ROB ensures **in-order commit**, precise state, and safe memory updates.

3. Bonus Features Implemented

3.1 User-Configurable Architecture Parameters

Integrated through chooseVariables():

- Change ROB size
- Change execution latency for each instruction type
- Adjust number of Reservation Stations per type

3.2 Label Support and Offset Calculation

Implemented in loadProgram():

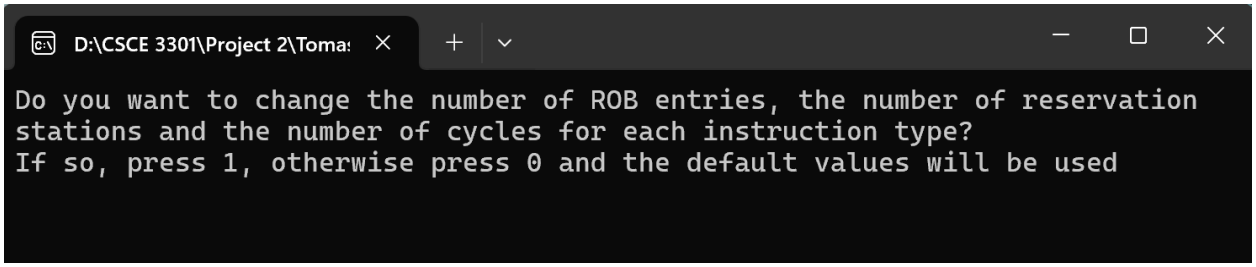
- Parses assembly with labels.
- Replaces label references with computed PC-relative offsets.
- Enables execution of programs with branches, loops, and structured control flow.

AI Use

- I conversed with AI on which data structure would be most suitable for the ROB, as it couldn't be a simple queue as we need to access data other than the head of the queue. I reached the conclusion of implementing it as a separate class which I implemented myself (not by AI).

User Guide

The user can run the main.cpp file in any IDE or use the executable version main.exe.
Once you run it, you'll get this in the console:



If 1 is pressed, you'll have to enter all this data:

```
Enter the number of ROB entries
6
Enter the number of cycles taken for memory read: 2
Enter the number of cycles taken for memory write: 3
Enter the number of reservation stations for load: 2
Enter the number of cycles taken for load: 2
Enter the number of reservation stations for store: 1
Enter the number of cycles taken for store: 2
Enter the number of reservation stations for beq: 1
Enter the number of cycles taken for beq: 3
Enter the number of reservation stations for call/ret: 1
Enter the number of cycles taken for call/ret: 2
Enter the number of reservation stations for add/sub: 3
Enter the number of cycles taken for add/sub: 2
Enter the number of reservation stations for nand: 1
Enter the number of cycles taken for nand: 2
Enter the number of reservation stations for mul: 3
Enter the number of cycles taken for mul: 10
```

However, if 0 is pressed, this section will be skipped.
After that, you are asked:

```
Choose program input method:
1) Enter instructions manually
2) Load instructions from a file
```

Enter 1 or 2 based on your preference, if 1 is pressed, then you are asked to enter the instructions and enter an empty line at the end:

```
Choose program input method:
1) Enter instructions manually
2) Load instructions from a file
1
Enter assembly program, one instruction per line.
End input with an empty line.
ADD R1, R0, R0
MUL R2, R1, R0
STORE R2, 0(R1)

At what address does your program start?
```

After entering the instructions as above, you are asked at what address the program starts, when entering it, you get asked:

```
100
Do you need to enter data into the data memory?
If so press 1, else press 0.
1
Enter the address you want to enter data to: 12
Enter the data you want to enter at address 12: 5

Do you want to enter more data?
If yes, press 1, else press 0
```

If you need to enter data for your program, you enter it as seen above, if you need to enter more data you press 1 for the question at the bottom then repeat.

When you're done entering the data, the simulator runs and provides you with the results in this format:

```
The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
100: 0  1  2  3  4
101: 1  3  12  13  14
102: 2  13  14  15  18

2. The total number of cycles the program took is: 18
3. The IPC is: 0.166667, the CPI is: 6
4. The branch misprediction percentage is: 0%
```

Test Cases

- Test case 1:

  LOAD R1, 0(R0)
  LOAD R2, 1(R0)
  ADD   R3, R1, R2
  SUB   R4, R3, R2
  MUL   R5, R3, R2
  NAND R6, R3, R4

```
Choose program input method:
1) Enter instructions manually
2) Load instructions from a file
2
Enter file path: D:\CSCE 3301\Project 2\Tomasulo Simulator\test1.txt
At what address does your program start?
100
Do you need to enter data into the data memory?
If so press 1, else press 0.
1
Enter the address you want to enter data to: 0
Enter the data you want to enter at address 0: 1

Do you want to enter more data?
 If yes, press 1, else press 0
1
Enter the address you want to enter data to: 1
Enter the data you want to enter at address 1: 2

Do you want to enter more data?
 If yes, press 1, else press 0
0

The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
100: 0  1  6  7  8
101: 1  2  7  8  9
102: 2  8  9  10  11
103: 3  10  11  12  13
104: 4  10  21  22  23
105: 5  12  12  13  24
2. The total number of cycles the program took is: 24
3. The IPC is: 0.25, the CPI is: 4
4. The branch misprediction percentage is: 0
```

This test case is very simple but it was testing the simulator's basic features, 2 loads can run concurrently as there are 2 reservation stages for it, and they each take 6 cycles. For the add instruction, it has to wait for the load instructions due to a RAW dependency, same goes for the sub at pc 103, it waits for R3 to be written by the add. As for mul and nand at the end, they get executed at the same time as there are no dependencies between them, notice that nand writes back before the mul instruction, but cannot commit before it.

- Test case 2:

  LOAD R1, 0(R0)
  ADD   R2, R1, R1
  STORE R2, 8(R0)
  LOAD R3, 8(R0)
  ADD   R4, R3, R1

```
Do you want to change the number of ROB entries, the number of reservation stations and
 the number of cycles for each instruction type?
If so, press 1, otherwise press 0 and the default values will be used
0
Choose program input method:
1) Enter instructions manually
2) Load instructions from a file
2
Enter file path: D:\CSCE 3301\Project 2\Tomasulo Simulator\test2.txt
At what address does your program start?
0
Do you need to enter data into the data memory?
If so press 1, else press 0.
1
Enter the address you want to enter data to: 0
Enter the data you want to enter at address 0: 1

Do you want to enter more data?
 If yes, press 1, else press 0
0

The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
0: 0  1  6  7  8
1: 1  7  8  9  10
2: 2  9  10  11  15
3: 3  4  15  16  17
4: 4  16  17  18  19
2. The total number of cycles the program took is: 19
3. The IPC is: 0.263158, the CPI is: 3.8
4. The branch misprediction percentage is: 0
```

This program focused on the interactions between load and stores. Notice that as we try to load a value that a store instruction is writing to, we do not need to access the memory and just get the data from the ROB, thus the load instruction takes only 2 cycles in execution.

- Test case 3:

```
LOAD  R1, 0(R0)
LOAD  R2, 1(R0)
BEQ   R1, R2, equal
ADD   R3, R0, R0
ADD   R4, R0, R0
equal: SUB   R3, R1, R2
BEQ   R3, R0, noteq
ADD   R4, R1, R1
noteq: ADD   R5, R1, R2
```

```
1) Enter instructions manually
2) Load instructions from a file
2
Enter file path: D:\CSCE 3301\Project 2\Tomasulo Simulator\test3.txt
At what address does your program start?
0
Do you need to enter data into the data memory?
If so press 1, else press 0.
1
Enter the address you want to enter data to: 0
Enter the data you want to enter at address 0: 5

Do you want to enter more data?
If yes, press 1, else press 0
1
Enter the address you want to enter data to: 1
Enter the data you want to enter at address 1: 5

Do you want to enter more data?
If yes, press 1, else press 0
0

The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
0: 0  1  6  7  8
1: 1  2  7  8  9
2: 2  8  8  9  10
3: 3  4  5  6  - 1
4: 4  5  6  - 1  - 1
5: 5  8  9  - 1  - 1
6: 6  - 1  - 1  - 1  - 1
7: 7  8  9  - 1  - 1
8: 8  9  - 1  - 1  - 1
5: 10  11  12  13  14
6: 11  13  13  14  15
7: 12  13  14  - 1  - 1
8: 13  14  - 1  - 1  - 1
8: 15  16  17  18  19

2. The total number of cycles the program took is: 19
3. The IPC is: 0.736842, the CPI is: 1.35714
4. The branch misprediction percentage is: 100%
```

This program tested taking branches and the consequent flushing of instructions.

Notice how much slower the same program runs when we reduce the number of ROB entries and reservation stages.

```
Enter file path: D:\CSCE 3301\Project 2\Tomasulo Simulator\test3.txt
At what address does your program start?
0
Do you need to enter data into the data memory?
If so press 1, else press 0.
1
Enter the address you want to enter data to: 0
Enter the data you want to enter at address 0: 5

Do you want to enter more data?
If yes, press 1, else press 0
1
Enter the address you want to enter data to: 1
Enter the data you want to enter at address 1: 5

Do you want to enter more data?
If yes, press 1, else press 0
0

The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
0: 0  1  6  7  8
1: 7  8  13  14  15
2: 8  14  16  17  18
3: 9  10  11  12  - 1
4: 12  13  14  15  - 1
5: 15  16  17  - 1  - 1
6: 17  - 1  - 1  - 1  - 1
5: 18  19  20  21  22
6: 19  21  23  24  25
7: 21  22  23  - 1  - 1
8: 25  26  27  28  29

2. The total number of cycles the program took is: 29
3. The IPC is: 0.37931, the CPI is: 2.63636
4. The branch misprediction percentage is: 100%
```

- Test case 4:

```
LOAD  R2, 0(R0)          # M[0] must be greater than 5, ex: 20
CALL  func
ADD   R4, R2, R3
ADD R1, R2, R0           # Used to prevent infinite loop caused by RET

func: ADD   R3, R2, R2
RET
```

```
Enter file path: D:\CSCE 3301\Project 2\Tomasulo Simulator\test4.txt
At what address does your program start?
0
Do you need to enter data into the data memory?
If so press 1, else press 0.
1
Enter the address you want to enter data to: 0
Enter the data you want to enter at address 0: 20

Do you want to enter more data?
If yes, press 1, else press 0
0

The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
0: 0  1  6  7  8
1: 1  2  2  3  9
2: 2  7  8  - 1  - 1
3: 3  7  8  - 1  - 1
4: 4  7  8  - 1  - 1
5: 5  - 1  - 1  - 1  - 1
4: 9  10  11  12  13
5: 10  11  11  13  14
2: 14  15  16  17  18
3: 15  16  17  18  19
4: 16  17  18  19  20
5: 17  18  18  20  21

2. The total number of cycles the program took is: 21
3. The IPC is: 0.571429, the CPI is: 1.75
4. The branch misprediction percentage is: 0%
```

This program tested function calls which use the same flush algorithm for branches which we can see is working properly. Notice as there is no instruction that stops the program, the function is executed again after the second ADD, to escape the RET instruction from sending us to the instruction after CALL, must manually change R1 to an address that exceeds the program size.

- Test Case 5:

```
LOAD R1, 100(R0)    # R1 <- M (Base)
LOAD R2, 101(R0)    # R2 <- N (Exponent/Counter)
LOAD R4, 1(R0)      # R4 <- 1 (Constant 1)
ADD R3, R4, R0
loop: BEQ R2, R0, done
MUL R3, R3, R1
SUB R2, R2, R4
BEQ R0, R0, loop
done: STORE R3, 102(R0)
```

```
Enter the address you want to enter data to: 100
Enter the data you want to enter at address 100: 2

Do you want to enter more data?
If yes, press 1, else press 0
1
Enter the address you want to enter data to: 101
Enter the data you want to enter at address 101: 3

Do you want to enter more data?
If yes, press 1, else press 0
0

The program will start running now.
pc:  issue time, execution start time, execution end time, write back time, commit time
0: 0  1  6  7  8
1: 1  2  7  8  9
2: 7  8  13  14  15
3: 8  14  15  16  17
4: 9  10  10  11  18
5: 10  16  27  28  29
6: 11  14  15  17  30
7: 12  13  13  15  31
8: 13  28  29  30  - 1
4: 31  32  32  33  34
5: 32  33  44  45  46
6: 33  34  35  36  47
7: 34  35  35  37  48
8: 35  45  46  47  - 1
4: 48  49  49  50  51
5: 49  50  61  62  63
6: 50  51  52  53  64
7: 51  52  52  54  65
8: 52  62  63  64  - 1
4: 65  66  66  67  68
5: 66  67  - 1  - 1  - 1
6: 67  - 1  - 1  - 1  - 1
8: 68  69  70  71  75

2. The total number of cycles the program took is: 75
3. The IPC is: 0.306667, the CPI is: 3.26087
4. The branch misprediction percentage is: 57%
```

This program features a loop that multiplies a number by itself a number of times to get its power, in this example we execute the loop 3 times to calculate 2^3. The program features both taken and not taken branches.