



CSCE3301

Computer Architecture

Tomasulo's Algorithm Simulator

Spring 2022

May 19, 2022

By:

Bishoy Sabry - 900183106

Omar Elayat - 900182568

Contents

Introduction.....	3
A brief description of the implementation	4
including bonus features.....	4
Full simulation with a step-by-step user guide	5
Results & discussion.	8

Introduction

This project implements an architectural simulator capable of assessing the performance of a simplified superscalar out-of-order 16-bit dual issue RISC processor that uses Tomasulo's algorithm with speculation.

The simulator assumes a simplified RISC ISA inspired by the ISA of the Ridiculously Simple Computer (RiSC-16) proposed by Bruce Jacob. As implied by its name, the word size of this computer is 16-bit. The processor has 8 general-purpose registers R0 to R7 (16-bit each). The register R0 always contains the value 0 and cannot be changed. Memory is word addressable and uses a 16-bit address. The instruction format itself is not very important to the simulation and therefore is not described here.

The simulator supports the following instructions:

- 1) Arithmetic operation: Addition, Adding with immediate, subtraction, NANDing, and multiplication.
- 2) Load operation.
- 3) Store operation.
- 4) Unconditional branch (Jumping to a specific address).
- 5) Conditional branch.
- 6) Jump and link operation
- 7) Return operation

This report details the instruction set to be supported, the inputs to the simulator, the implementation of the simulator (with its bonus features), the assumptions made, step-by-step simulation and the expected outputs.

A brief description of the implementation

including bonus features

The RISC processor simulator is a C++ program that reads the memory file and instructions file and then implements Tomassolo's algorithm to assess its' performance parameters. The program is decomposed into stages:

- 1- The program reads the memory and instructions files and then save the memory variables and the instructions in the code data structures (Maps/Vectors). Recognizing labels also in this stage.
- 2- The program starts to read instruction by instruction and parse the instruction from the operand. Function "parse" do this task.
- 3- The instruction is then parsed and classified according to its type (R,S,B,U,I). The Function "compute" do this task. The operands are also parsed according to the type of instruction. Each instruction, reservation station, Reorder buffer is modeled as a struct that contains variables for all its needed values, and all the flags they need for proper execution.
- 4- Function "Process" is then called to do the whole simulation of execution. Where it runs an infinite loop representing an infinite amount of clock cycles available for running. Inside, a nested loop where each iteration represents an instruction is executed in each clock cycle. The nested loop calls four functions within each iteration, which are: Commit, wb (write back), execute, and issue. Each function of the four implements its respective part in the algorithm description while updating the execution flags and variables of instructions, reservation stations, Reorder buffer entries structs.
- 5- The function "print" is then called to print on the terminal, which is useful for debugging.
- 6- Then the function "write" is called to write all the expected outputs in a single ".csv" file, it also writes the new memory contents in a ".txt" called "MemoryOutput.txt", and the state of the reservation stations and the value of the registers after execution in another ".txt" called "RegisterOutput.txt".

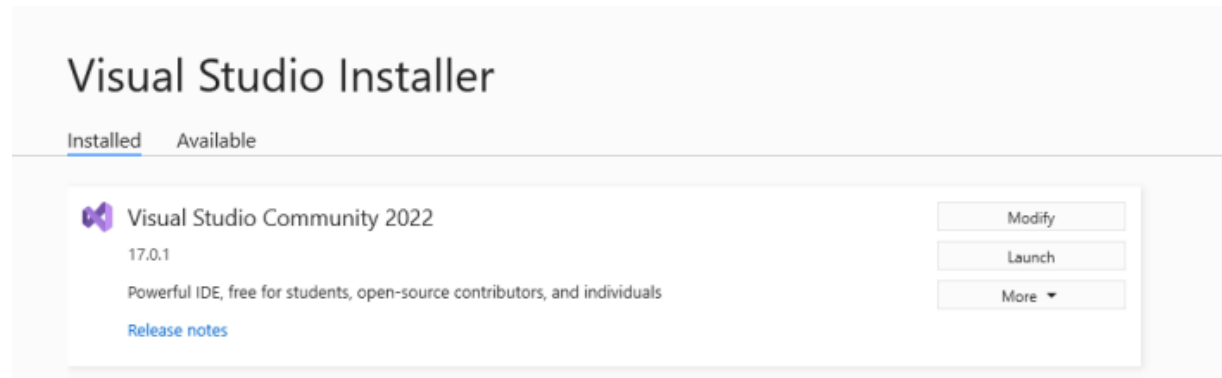
We implemented two bonus features in the project, which are:

- 1- We implemented a parsing function that gave the user the option to enter his/her assembly program in a text file. The file is read, parsed, executed, and the results are outputted in ".csv" file. (Bonus Feature 5).
- 2- The simulator is provided with a testing folder that contains 12 benchmark programs that test divergent aspects of the algorithm. The folder also contains the benchmark programs respective outputs as ".csv" files. (Bonus Feature 6)

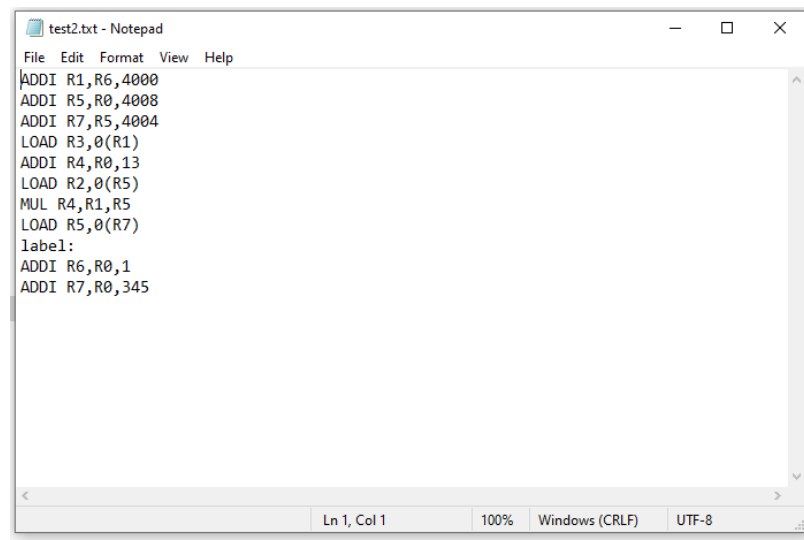
Full simulation with a step-by-step user guide

To run the program:

- 1- Install visual studio 17.0.1 (No additions needed)



- 2- Download and extract the folder “project2”
- 3- Put the RISC code you want to run in “test2.txt”. In this step-by-step guide, the RISC code used is in a file named “test_case6.txt”. It can be found in “Tests” folder inside the same “.zip” file.

The image shows a Notepad window titled "test2.txt - Notepad". The window contains the following RISC code:

```
File Edit Format View Help
ADDI R1,R6,4000
ADDI R5,R0,4008
ADDI R7,R5,4004
LOAD R3,0(R1)
ADDI R4,R0,13
LOAD R2,0(R5)
MUL R4,R1,R5
LOAD R5,0(R7)
label:
ADDI R6,R0,1
ADDI R7,R0,345
```

The status bar at the bottom indicates "Ln 1, Col 1", "100%", "Windows (CRLF)", and "UTF-8".

- 4- Put the memory addresses and values (0001 must be proceeded with the program starting address) in “mem.txt” file


```

Debug - x64 - Local Windows Debugger - Microsoft Visual Studio Debug Console
Tomasolo.cpp
Issued inst 7
empty_index8
iss_flag 1
issued inst 8
clock number: 10
committed 3
committed 4
.txt*);
.txt*);
I'm in wb stage of 5
empty_index9
iss_flag 1
issued inst 9
clock number: 11
committed 5
clock number: 12
instruction 8executed
clock number: 13
instruction 7executed
I'm in wb stage of 8
instruction 9executed
clock number: 14
I'm in wb stage of 7
I'm in wb stage of 9
clock number: 15
clock number: 16
clock number: 17
instruction 6executed
clock number: 18
clock number: 19
I'm in wb stage of 6
code 0 (0x0). clock number: 19
Loaded 'C:\Windocommitted 6
Loaded 'C:\Windocommitted 7
Loaded 'C:\Windocommitted 7
code 0 (0x0).clock number: 20
code 0 (0x0). committed 8
code 0 (0x0). committed 9
Tomasolo.exe' has clock number: 21

```

7- Observe the output in the files “MemoryOutput.txt”, “RegisterOutput.txt”, and “output_clock.csv”.

```

RegisterOutput.txt - Notepad
File Edit Format View Help
.....
R0 Decimal: 0
R1 Decimal: 4000
R2 Decimal: -11
R3 Decimal: 152
R4 Decimal: 16032000
R5 Decimal: 0
R6 Decimal: 1
R7 Decimal: 345
station num0
busy0
instruction typeLoad
vj8012
vk0
qj-1
qk-1
dest8
address8012
address8012
.....
station num1
busy0
instruction typeLoad
vj4008
vk0
qj-1
qk-1
dest6
address4008
address4008
.....
station num2
busy0
instruction type
vj0
vk0
<

```

Instruction index								
A	B	C	D	E	F	G	H	I
1	instruction	issued	started ex	finished e	write back	committed		
2	0 ADDI	1	2	4	5	6		
3	1 ADDI	1	2	4	5	6		
4	2 ADDI	2	5	7	8	9		
5	3 Load	2	5	8	9	10		
6	4 ADDI	5	6	8	9	10		
7	5 Load	5	6	9	10	11		
8	6 MUL	6	7	17	18	19		
9	7 Load	9	10	13	14	19		
10	8 ADDI	9	10	12	13	20		
11	9 ADDI	10	11	13	14	20		
12	total exec	20 cycles						
13	IPC	0.5						
14	branch mi	0						
15								
16								
17								
18								
19								
20								
21								

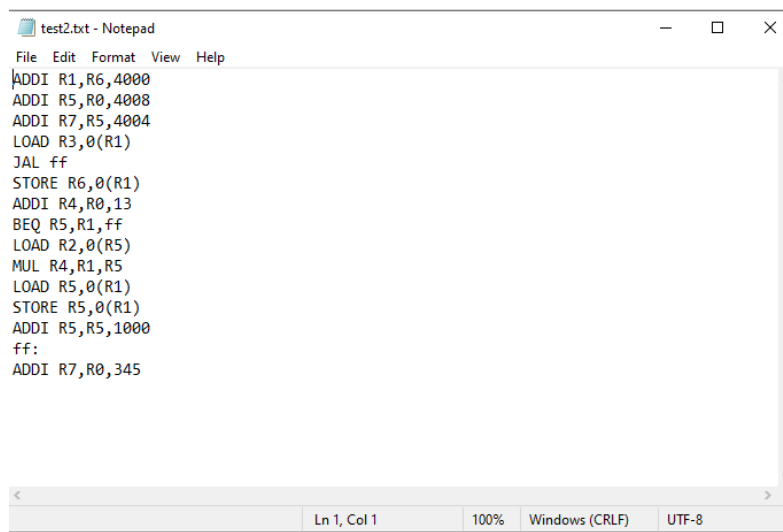
Results & discussion.

General Note:

- All the programs are in a folder called “Tests”.
- In order to run them, move the memory file and the instruction file of the program you want to run to the same location as the source.cpp file, and write its name in the main in the as.read_file (“”) & as.read_memory (“”) lines
- You can find the outputs of all the simulated programs inside the same folder.

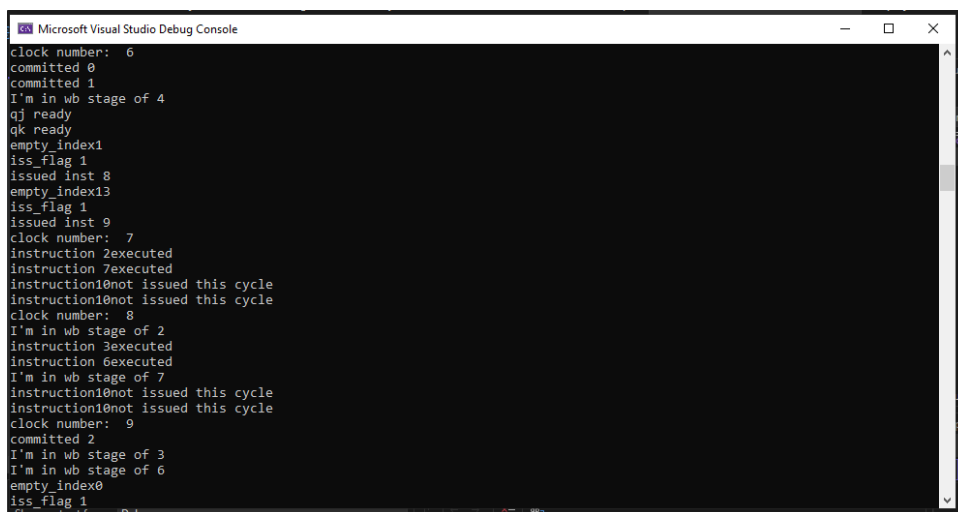
This section demonstrates only three test programs of the twelve provided.

Program 1 (Test case 8)



```
test2.txt - Notepad
File Edit Format View Help
ADDI R1,R6,4000
ADDI R5,R0,4008
ADDI R7,R5,4004
LOAD R3,0(R1)
JAL ff
STORE R6,0(R1)
ADDI R4,R0,13
BEQ R5,R1,ff
LOAD R2,0(R5)
MUL R4,R1,R5
LOAD R5,0(R1)
STORE R5,0(R1)
ADDI R5,R5,1000
ff:
ADDI R7,R0,345
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

This Program tests that the program is working in general. It also tests various instructions, including: Add with immediate, Add, Load, and JAL.



```
Microsoft Visual Studio Debug Console
clock number: 6
committed 0
committed 1
I'm in wb stage of 4
qj ready
qk ready
empty_index1
iss_flag 1
issued inst 8
empty_index13
iss_flag 1
issued inst 9
clock number: 7
instruction 2executed
instruction 7executed
instruction10not issued this cycle
instruction10not issued this cycle
clock number: 8
I'm in wb stage of 2
instruction 3executed
instruction 6executed
I'm in wb stage of 7
instruction10not issued this cycle
instruction10not issued this cycle
clock number: 9
committed 2
I'm in wb stage of 3
I'm in wb stage of 6
empty_index0
iss_flag 1
Show output from: Debug
```


Results:

```

RegisterOutput.txt - Notepad
File Edit Format View Help
.....
R0 Decimal: 0
R1 Decimal: 5
R2 Decimal: 0
R3 Decimal: 152
R4 Decimal: 0
R5 Decimal: 4008
R6 Decimal: 0
R7 Decimal: 345
station num0
busy0
instruction typeLoad
vj13
vk0
qj-1
qk-1
dest3
address0
address0
.....
station num1
busy0
instruction typeLoad
vj4008
vk0
qj-1
qk-1
dest1
address4008
address4008
.....
station num2
busy0
instruction typeSTORE
vj13
vk0
<

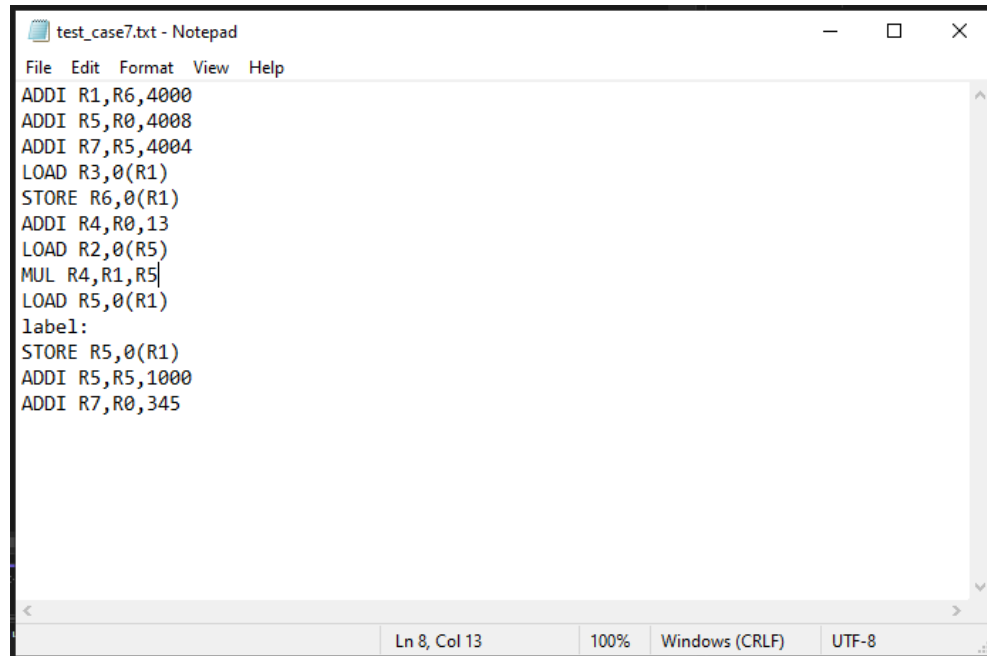
```

	A	B	C	D	E	F	G	H	I
1	instruction	issued		started ex	finished e	write back	committed		
2	0	ADDI	1	2	4	5	6		
3	1	ADDI	1	2	4	5	6		
4	2	ADDI	2	5	7	8	9		
5	3	Load	2	5	8	9	10		
6	4	JAL	3	4	5	6	10		
7	5	STORE	3	8	-8.4E+08	-8.4E+08	-8.4E+08		
8	6	ADDI	5	6	8	9	-8.4E+08		
9	7	BEQ	5	6	7	8	-8.4E+08		
10	8	Load	6	7	-8.4E+08	-8.4E+08	-8.4E+08		
11	9	MUL	6	7	-8.4E+08	-8.4E+08	-8.4E+08		
12	10	Load	9	-8.4E+08	-8.4E+08	-8.4E+08	-8.4E+08		
13	11	STORE	-8.4E+08	-8.4E+08	-8.4E+08	-8.4E+08	-8.4E+08		
14	12	ADDI	-8.4E+08	-8.4E+08	-8.4E+08	-8.4E+08	-8.4E+08		
15	13	ADDI	10	11	13	14	15		
16	total exec	15 cycles							
17	IPC	0.933333							
18	branch mi	0							
19									
20									
21									
22									
23									

Output_clock

As shown in the results, the issuing and committing is done in order. The execution order seems logical and the write back is done in an approximate, yet logical, order. The simulation successfully models the performance of processing the given text file.

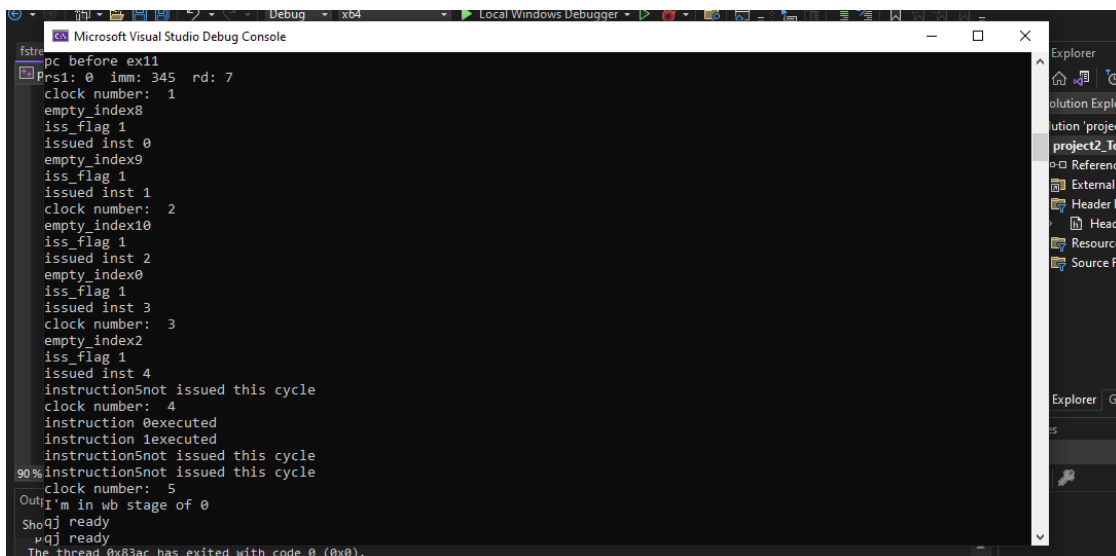
Program 2 (Test case 7)



```
test_case7.txt - Notepad
File Edit Format View Help
ADDI R1,R6,4000
ADDI R5,R0,4008
ADDI R7,R5,4004
LOAD R3,0(R1)
STORE R6,0(R1)
ADDI R4,R0,13
LOAD R2,0(R5)
MUL R4,R1,R5
LOAD R5,0(R1)
label:
STORE R5,0(R1)
ADDI R5,R5,1000
ADDI R7,R0,345
Ln 8, Col 13 100% Windows (CRLF) UTF-8
```

This test program is designed to test the load and store hazards where queue of loads and stores are constructed. If you want to change the elements of the registers, just modify the values in the memory text file.

Results:



```
Microsoft Visual Studio Debug Console
pc before ex11
Prs1: 0 imm: 345 rd: 7
clock number: 1
empty_index8
iss_flag 1
issued inst 0
empty_index9
iss_flag 1
issued inst 1
clock number: 2
empty_index10
iss_flag 1
issued inst 2
empty_index0
iss_flag 1
issued inst 3
clock number: 3
empty_index2
iss_flag 1
issued inst 4
instruction5not issued this cycle
clock number: 4
instruction 0executed
instruction 1executed
instruction5not issued this cycle
90%instruction5not issued this cycle
clock number: 5
Out! I'm in wb stage of 0
Shoqj ready
wqj ready
The thread 0x83ac has exited with code 0 (0x0).
```

```

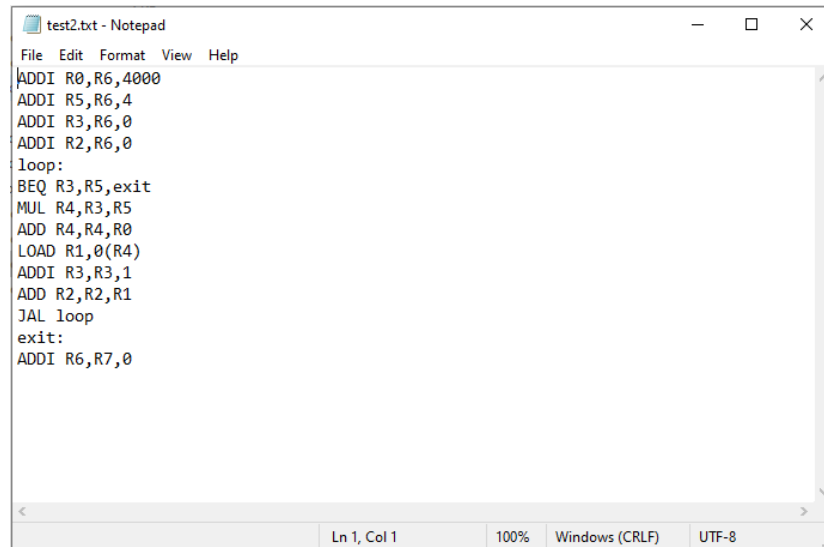
RegisterOutput.txt - Notepad
File Edit Format View Help
.....
R0 Decimal: 0
R1 Decimal: 4000
R2 Decimal: -11
R3 Decimal: 152
R4 Decimal: 16032000
R5 Decimal: 1152
R6 Decimal: 0
R7 Decimal: 345
station num0
busy0
instruction typeLoad
vj4000
vk0
qj-1
qk-1
dest1
address4000
address4000
.....
station num1
busy0
instruction typeLoad
vj4008
vk0
qj-1
qk-1
dest7
address4008
address4008
.....
station num2
busy0
instruction typeSTORE
vj4000
vk0
<

```

	A	B	C	D	E	F	G	H	I	J
1	instruction	instruction	issued	started ex	finished e	write back	committed			
2	0	ADDI	1	2	4	5	6			
3	1	ADDI	1	2	4	5	6			
4	2	ADDI	2	5	7	8	9			
5	3	Load	2	5	8	9	10			
6	4	STORE	3	8	11	12	13			
7	5	ADDI	5	6	8	9	13			
8	6	Load	5	6	9	10	14			
9	7	MUL	6	7	17	18	19			
10	8	Load	9	10	13	14	19			
11	9	STORE	9	13	16	17	20			
12	10	ADDI	10	14	16	17	20			
13	11	ADDI	10	11	13	14	21			
14	total exec	21	cycles							
15	IPC	0.571429								
16	branch mi	0								

As shown in the results, the issuing and committing is done in order. The execution order seems logical and the write back is done in an approximate, yet logical, order. The simulation successfully models the performance of processing the given text file.

Program 3 (Test case 9)



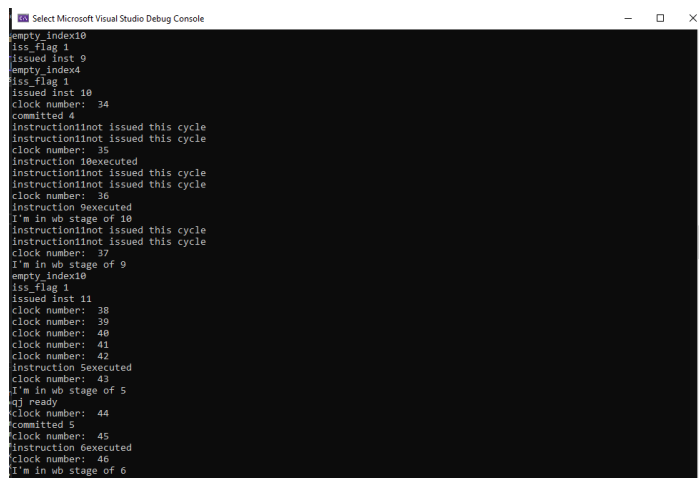
```
test2.txt - Notepad
File Edit Format View Help
ADDI R0,R6,4000
ADDI R5,R6,4
ADDI R3,R6,0
ADDI R2,R6,0
loop:
BEQ R3,R5,exit
MUL R4,R3,R5
ADD R4,R4,R0
LOAD R1,0(R4)
ADDI R3,R3,1
ADD R2,R2,R1
JAL loop
exit:
ADDI R6,R7,0
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

This program adds all the elements in a given array with a given size and saves the output in the memory. If the user wants to change the elements of the array, just modify them in the memory text file and modify the size of the array in the program text file

Note:

- The first element listed in the memory file is the starting address of the program.
- The starting address of the array in the instructions text must equal the one in the memory text

Results:



```
Select Microsoft Visual Studio Debug Console
empty_index0
iss_flag 1
issued inst 9
empty_index4
iss_flag 1
issued inst 10
clock number: 34
committed 4
instruction11not issued this cycle
instruction1not issued this cycle
clock number: 35
instruction10executed
instruction11not issued this cycle
instruction1not issued this cycle
clock number: 36
instruction 9executed
I'm in wb stage of 10
instruction11not issued this cycle
instruction1not issued this cycle
clock number: 37
I'm in wb stage of 9
empty_index0
iss_flag 1
issued inst 11
clock number: 38
clock number: 39
clock number: 40
clock number: 41
clock number: 42
instruction 5executed
clock number: 43
I'm in wb stage of 5
rdj ready
clock number: 44
committed 5
clock number: 45
instruction 6executed
clock number: 46
I'm in wb stage of 6
```

```

RegisterOutput.txt - Notepad
File Edit Format View Help
|.....
R0 Decimal: 4000
R1 Decimal: 11
R2 Decimal: -90
R3 Decimal: 4
R4 Decimal: 4012
R5 Decimal: 4
R6 Decimal: 0
R7 Decimal: 0
station num0
busy0
instruction typeLoad
vj4012
vk0
qj-1
qk-1
dest4
address0
address0
.....
station num1
busy0
instruction type
vj0
vk0
qj-1
qk-1
dest0
address0
address0
.....
station num2
busy0
instruction type
vj0
vk0
<

```

instruction index								
	A	B	C	D	E	F	G	H
1	instruction	issued	started ex	finished e	write back	committed		
2	0 ADDI	1	2	4	5	6		
3	1 ADDI	1	2	4	5	6		
4	2 ADDI	2	3	5	6	7		
5	3 ADDI	5	6	8	9	10		
6	4 BEQ	101	102	103	104	81		
7	5 MUL	102	103	89	90	91		
8	6 ADD	102	90	92	93	94		
9	7 Load	103	93	96	97	98		
10	8 ADDI	103	104	82	83	98		
11	9 ADD	104	97	99	100	101		
12	10 JAL	104	81	82	83	101		
13	11 ADDI	105	106	108	109	110		
14	total exec	110 cycles						
15	IPC	0.109091						
16	branch mi	1						
17								
18								
19	Note: This test case illustrates how the simulator work with the loops							
20								
21								
22								
23								

Note: Instruction 4- 11 is not issued at clock cycle 101 as shown. The code is a loop, so 101 is the cycle where the last BEQ in the loop was issued. Loop unrolling is not supported by our simulator.

As shown in the results, the issuing and committing is done in order. The execution order seems logical and the write back is done in an approximate, yet logical, order. The simulation successfully models the performance of processing the given text file.