COL216 Assignment-3

Dhruv Kumar Gupta 2019CS10346
Swaraj Gawande 2019CS10406

March 2021

1 Design Overview

1.1 Input and Output

The Program accepts a command line input for the file name and path which contains the MIPS program.

If the filename is not provided, the program uses a default filename as "code.txt" .

Program then loads the instructions in the file, raises syntax errors if any, and if no syntax error is present, executes the instructions.

After each execution, 32 register values are printed, and at the end, as per the specifications, number of clock cycles and the number of times each instruction is executed is printed.

Example screen :

```
D:\IITD\COL216\Assignment1\Assignment3>.\Assignment3.exe code.txt
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x0 | $r2 : 0x0 | $r3 : 0x0 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0 | $re : 0x0
| $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0 | $r1c : 0x0 | $r1d : 0x
0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x0 | $r2 : 0x0 | $r3 : 0x10 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0 | $re : 0x0
 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0 | $r1c : 0x0 | $r1d : 0
x0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x0 | $r2 : 0x10 | $r3 : 0x10 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0 | $re : 0x
0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0 | $r1c : 0x0 | $r1d :
0x0 | $sp : 0x3ffff
Number of clock cycles is 3
Instruction 3 is run 1 times
Instruction 1 is run 1 times
Instruction 2 is run 1 times

D:\IITD\COL216\Assignment1\Assignment3>.\Assignment3.exe code2.txt
invalid instruction at line 6

D:\IITD\COL216\Assignment1\Assignment3>.\Assignment3.exe code3.txt
Syntax Error : Invalid instruction at line 1
```

1.2 Error Handling

Each instruction is checked for syntax errors, which if found are printed along with line number.

The program also checks for undeclared or repeated labels and handles them accordingly.

There are some runtime errors like out of bound addresses and modification of zero register which are handled during the run of the program.

2 Explanation of approach and code

The program consists of 3 classes :

- **RegisterFile** : This class contains the 32 registers where the first register \$0 is zero register, the last register \$31 is the stack pointer register pointing towards end of memory and all other registers are normal integer register named \$r0 to \$r29. The stack pointer has been used since, using the given operations, stacks can be used.
- **Memory** : This class represents the Memory of the simulator, where address differ by **1** unlike QtSpim[1]. The address ranges from 0 to $2^{18} - 1$.
- **Simulator** : This class represents the actual simulator which combines the above two classes and supports functions like load, run and clear.

A structure **Instruction** is used to clearly specify instructions with the help of enumeration **InstructionType** and other data fields.

The load function reads each line of the input file and removes comments if present. Then, the line is processed by a private function parser which takes a single line at a time as input from the file and then reads it, taking tokens which are seperated by a space or a tab. This tokenisation helps in recognising operators and the operands. Before this tokenisation, labels declarations are searched, which if present are stored in a Hash Table of the Simulator, and then the line is processed normally.

The operator recognised, is then matched with an already declared enumeration containing valid operators. According to the operator recognised, the operands are parsed accordingly with commas and spaces and brackets as delimiters. If the instruction has syntax errors

After the loading of instructions, a traversal through the instructions is done to ensure that the jump instructions referring to forward label declarations are correct.
The MIPS document provided, allows immediate values as second third operand in add, sub, mul, slt. Keeping that in mind, the program does allow such expressions. Since, the clarification was provided on the deadline, we decided to not change the code as this is more general than not allowing immediate values.

The simulator maintains a RegisterFile, a memory block, program counter, hashmap of labels, a hashmap of number of times an instruction is executed and a boolean value to choose printing base of registers, as base 16 or base 10 with default being base 16.

If each instruction is in valid format, the text file is executed. The program counter starts with pointing towards instruction 0 and instructions are executed until the counter gets outside the instruction memory. After each Instruction is executed successfully the state of all the registers is printed. And after execution for all the instruction is finished the program prints our how many times each of the instructions is executed.

The execution checks for the modification of \$zero register since it is not allowed. Since there is no division, division by 0 need not be checked, and C++ handles overflow itself.

```
D:\IITD\COL216\Assignment1\Assignment3>.\Assignment3.exe code7.txt
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x3 | $r2 : 0x0 | $r3 : 0x0 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0
| $re : 0x0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0
| $r1c : 0x0 | $r1d : 0x0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x3 | $r2 : 0xa | $r3 : 0x0 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0
| $re : 0x0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0
| $r1c : 0x0 | $r1d : 0x0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x3 | $r2 : 0xa | $r3 : 0x0 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0
| $re : 0x0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0
| $r1c : 0x0 | $r1d : 0x0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x3 | $r2 : 0xa | $r3 : 0x0 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0
| $re : 0x0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0
| $r1c : 0x0 | $r1d : 0x0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x3 | $r2 : 0xa | $r3 : 0x64 | $r4 : 0x0 | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0
 | $re : 0x0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0
 | $r1c : 0x0 | $r1d : 0x0 | $sp : 0x3ffff
$zero : 0x0 | $r0 : 0x0 | $r1 : 0x3 | $r2 : 0xa | $r3 : 0x64 | $r4 : 0xa | $r5 : 0x0 | $r6 : 0x0 | $r7 : 0x0 | $r8 : 0x0 | $r9 : 0x0 | $ra : 0x0 | $rb : 0x0 | $rc : 0x0 | $rd : 0x0
 | $re : 0x0 | $rf : 0x0 | $r10 : 0x0 | $r11 : 0x0 | $r12 : 0x0 | $r13 : 0x0 | $r14 : 0x0 | $r15 : 0x0 | $r16 : 0x0 | $r17 : 0x0 | $r18 : 0x0 | $r19 : 0x0 | $r1a : 0x0 | $r1b : 0x0
 | $r1c : 0x0 | $r1d : 0x0 | $sp : 0x3ffff
Number of clock cycles is 6
Instruction 7 is run 1 times
Instruction 2 is run 1 times
Instruction 1 is run 1 times
Instruction 6 is run 1 times
Instruction 3 is run 1 times
Instruction 5 is run 1 times
```

[1]Here memory is word addressable

3 Testing

Firstly testing was done for all combinations of white spaces and comments possible, tabs and newlines were also used.

The parse function is then tested firstly with all blanck or commented lines and then for all the instructions given in specifications for this assignment. Parsing is done in groups and thus tests for parsing also go in groups. **add, mul, sub, slt** are tested with incorrect or empty operands and invalid spaces in the description also as the third operand can also be immediate value that was tested too. **bne, beq** are tested with correct and incorrect labels declared before or after the instruction. In case of incorrect label an appropriate error message saying Illegal jump is displayed along with tests for register names and immediate values as in the group discussed above. **j** jump instruction only has the label as the operand and thus tested for the labels. **sw, lw** they take two registers and offset in a specific format all the valid and invalid options of having spaces in the format were tested. In doing so some parsing error were encountered and fixed. Lastly testing **addi** was simple and done as in first group.
Then some asm code with combination of these instructions are then used for testing some of which are written below. Infinite loops are also tested.

Then the run function is tested that if a given instruction does what it is expected to. Also having $zero as destination also should print an error message and stop the execution. In case of **sw** and **lw** it is tested for cases when the address is out of range of memory and an appropriate error message is printed.
In case of correct input the corresponding state of registers after each instruction executed is then checked to make sure that they are same as manually calculated values.

3.1 Some testcases

1.
```
begin: add $r2 , $r3 , $r4 #Hi
add $r3,$r4,16 # Hi
# comment
add $3, $4, $zero
```

2.
```
begin: add $r2 , $r3 , $r4 #Hi
add $r3,$r3,1 # Hi
j begin
# comment
add $3, $4, $zero
j end
end:
```

3.
```
begin: add $r2 , $r3 , $r4 #Hi
add     $r3,     $r3,1 # Hi
# comment
add $3, $4, $zero
j end
sw $r5, $2
end:
```

4.
```
jstart
start:
addi $r2,$s1,10

#label          hello
```

5.
```
begin: add $r2 , $r3 ,#Hi add $r3,$r3,1 # Hi
j begin
# comment
add $3, $4, $zero
j end
end:
```

6.
```
j label
# comment
label:
add $r1,$r2,-20
end:
bne $zero, $r1 ,label
```

7.
```
sub $2,$zero, -3
addi $r2, $r1, 7
beq $r2,$r2,jp
sw $r1,($zero)
jp:sw $r2,90($r2)
mul $r3,$r2,$r2
lw $r4, ($r3)
```

8.
```
addi $r1, $zero ,10
la:
mul $r1,$r1,10
sw $r1, ($r1)
lw $r2, ($r2)
j la
```