TICKLAB

Documentary

MIPS RUNNER

Developer:

Nguyen Thanh Toan

Vu Nguyen Minh Huy

Instructor:

Huynh Hoang Kha

December 15th 2019

Content

1. Idea

1.1 Initital idea:

- Assembly is a low-level programming language that affects registers directly. The execution of an assembly program will affect the operating status of the computer. This leads to the need for a virtual machine to simulate assembly program behavior. The popular virtual machine is the Java virtual machine.
- MIPS-Setting low-level machine language on c ++ that can translate and show users running each command

1.2 Main idea:

- Write a compile program to check for code syntax errors
- Identify and save labels to perform address jump commands
- Simulate components with 32 bit memory cells, store values
- After compile, put the register data register on the screen and each command (10 sentences) for the user to manipulate to run each command line

2. Researching and doing process

2.1 Knowledge for research:

a) *Document*:

- Computer organization and design 5th edition
- Mr Dat's report
- Web documents
  + `http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html`
  + `http://www.cs.uwm.edu/classes/cs315/Bacon/Lecture/HTML/ch05s04.html`
  + `https://people.cs.pitt.edu/ childers/CS0447/lectures/SlidesLab92Up.pdf`

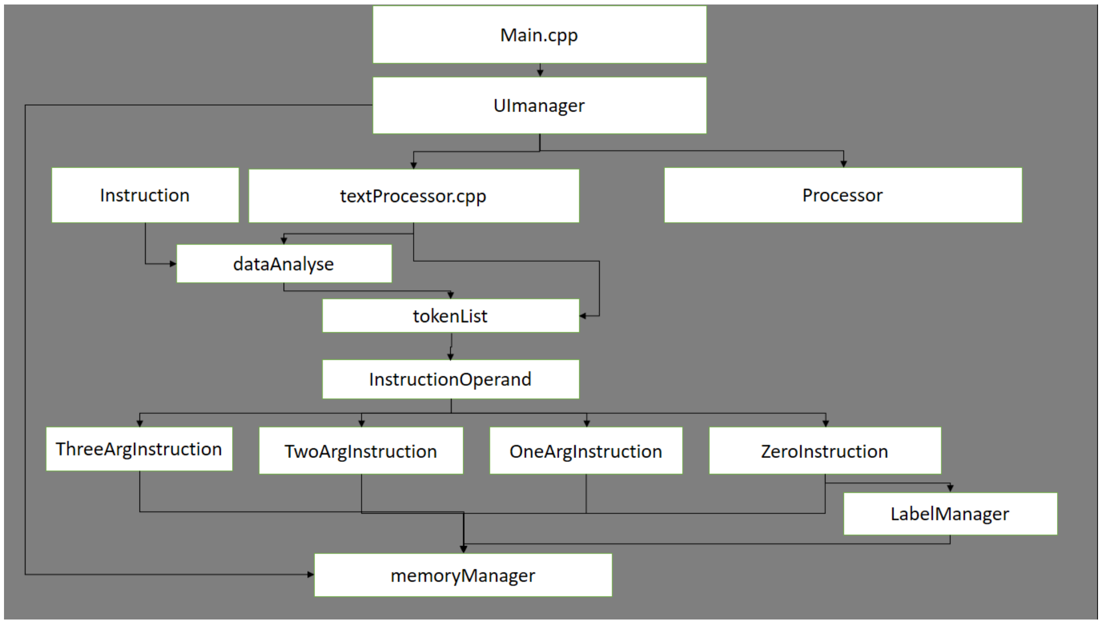
b) *Instructor:*

Huynh Hoang Kha

c) *The process of inquiry:*

- Some noticeable things:
  + Register
  + Instruction
  + The way computer executes variables
    Initialize variable - load value into memory -Store in register - executes.
  + How the computer processes loops by saving the address of the statement with a label

2.2. Figure:



2.3. **Processing:**

a) *First outline:*

- Read the code file from a text file
- Split into lines in the textProcessor file
- Split each line into tokens in the tokenList file
- Identify the tokens (Instruction, label) in the InstructionOperand file separately the data envelope (.data) and the commandtext area: will be processed separately to store variable labels
- Instruction is divided into 4 types
  + ThreeArgInstruction(three argument instruction)
  + TwoArgInstruction(two argument instruction)
  + OneArgInstruction(one argument instruction)
  + ZeroArgInstruction(zero argument instruction)
- Each type of instruction has corresponding instructions which execute those instructions.
- Register:
  + Registers are stored in the memoryManager file
  + Simulate registers into two array functions of 35 elements and 31 elements (floating-point register) to store values
  + Particularly register value pc, is the order of lines commands are processed in the textProcessor file

* Difficulty:

- To recognize token, when handling each line, tokens that do not recognize it is a immediate value or register values
- The idea is only suitable when the labels are declared before the jump or jal command to the labels
- Before processing the line must have a compile

b) *Final idea:*

- Right from the textProcessor file, identify the tokens as Immediate value, register, label, variable label
- Run the program before once to save the labels
- Add compile feature

3. Product

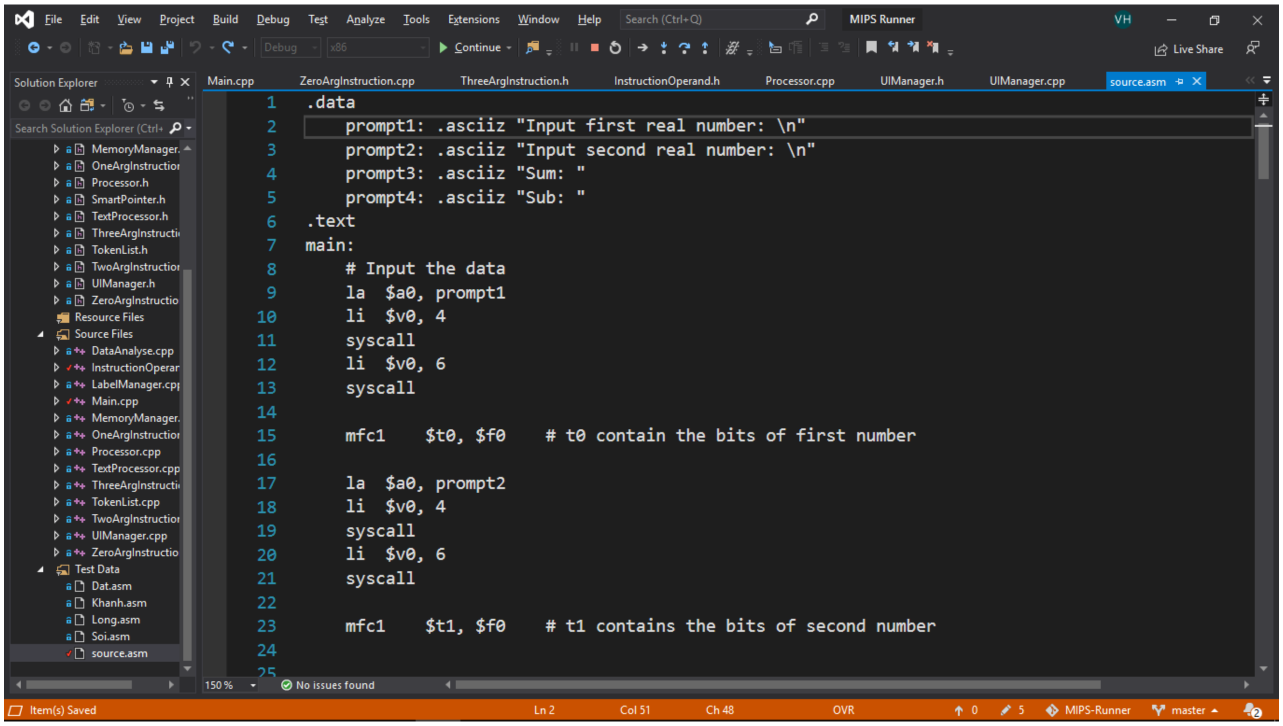
3.1. Priciple of operation:

- Read a text file (source.asm) from textProcessor class Then separate these sourecode line by line, save these lines to src (char **) to save each char * corresponding to 1 statement
- Parse sourceCode into Instruction:
  + Partition to declare data (.data) and write code (.text)
  + Identify the label
  + Identify Instruction:
    . Use the register pc to save the order of the current statement in the char** array
    . Use tokenList to split elements in the line into tokens, saved in token* array
    . Counts the number of elements in tokenList * to identify which Instruction type (Three, Two, One or Zero instruction) is
    . Compile:If the number of arguments in tokenList is more than 3, compile error
- Processing Instruction: e.g ThreeArgInstruction:
  + Simulate 3 variables rd, rs, rt in class InstructionOperand.
  + This Instruction Operand class is responsible for assigning identification 7 symbols:
    . Variable label
    . Label
    . Register
    . Floating-point register
    . Integer
    . Float
    . Address
  + Identify the Instruction in ThreeArgInstruction:
    . Use the variables rs, rt, rd to access memory if it is a register (include memoryManager into InstructionOperand, use the pointer to access array to store Register data)
    . Compile: If in that Instruction, the identifiers of rs, rt cannot be exceeded (e.g. addi requires rt as immediate value = integer), an error will be displayed
    . If it is immediate value, simulate a new variable different from the register memory cell
  + Identify the Instruction in OneArgInstruction:

. The labels have been saved the original address, when jumping, just change pc (saving the current location)

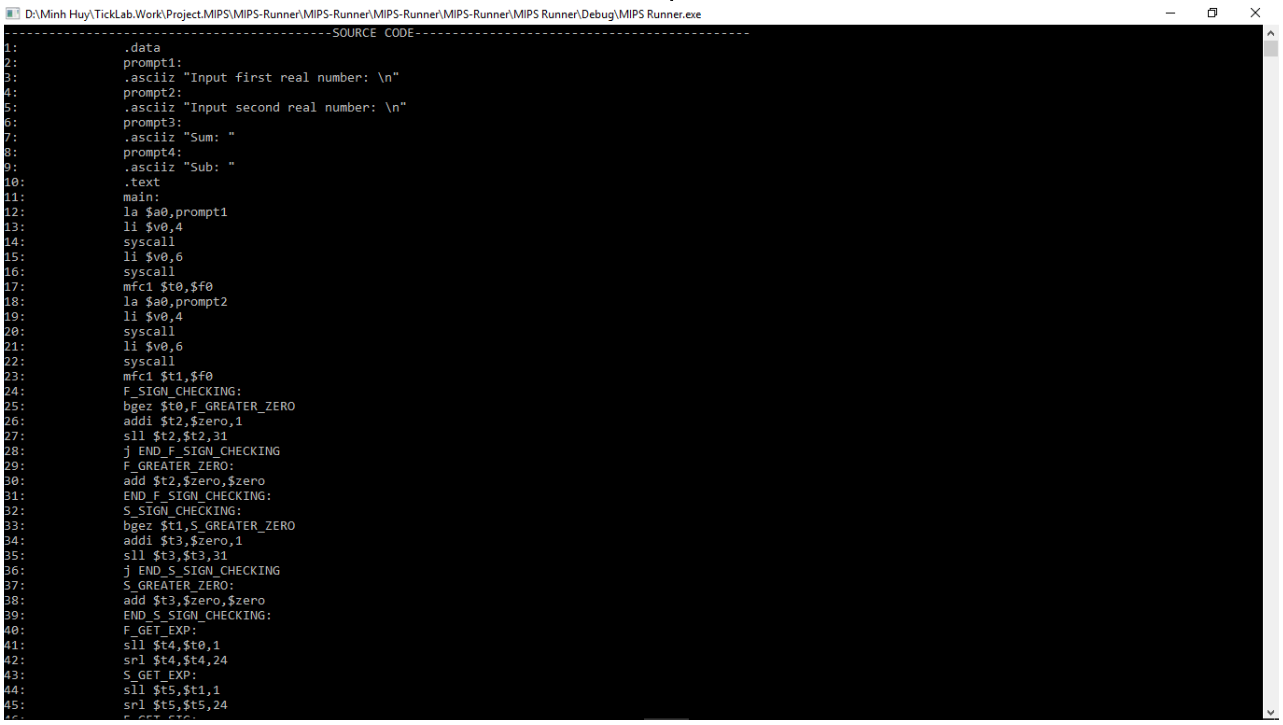
- Export values to the screen using UImanager class.

3.2 Demonstration:



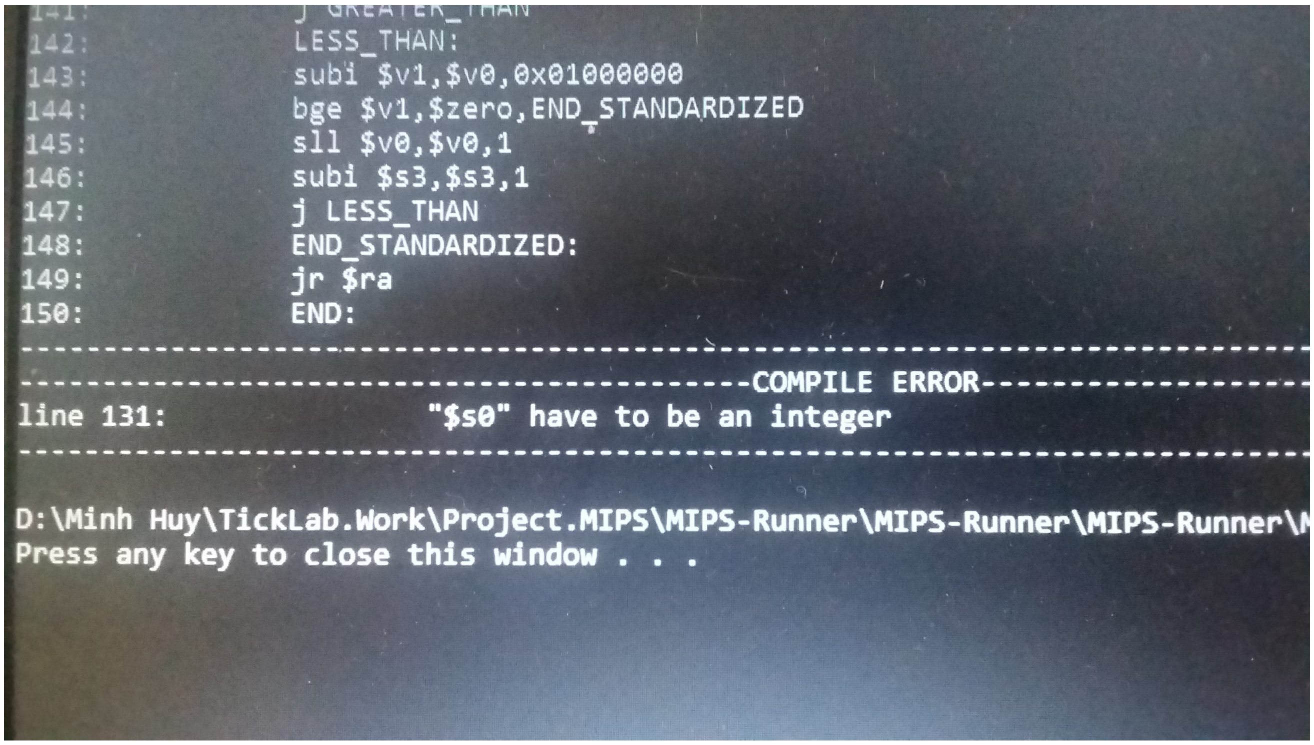
```
.data
    prompt1: .asciiz "Input first real number: \n"
    prompt2: .asciiz "Input second real number: \n"
    prompt3: .asciiz "Sum: "
    prompt4: .asciiz "Sub: "
.text
main:
    # Input the data
    la  $a0, prompt1
    li  $v0, 4
    syscall
    li  $v0, 6
    syscall

    mfc1    $t0, $f0    # t0 contain the bits of first number

    la  $a0, prompt2
    li  $v0, 4
    syscall
    li  $v0, 6
    syscall

    mfc1    $t1, $f0    # t1 contains the bits of second number

```

Assembly



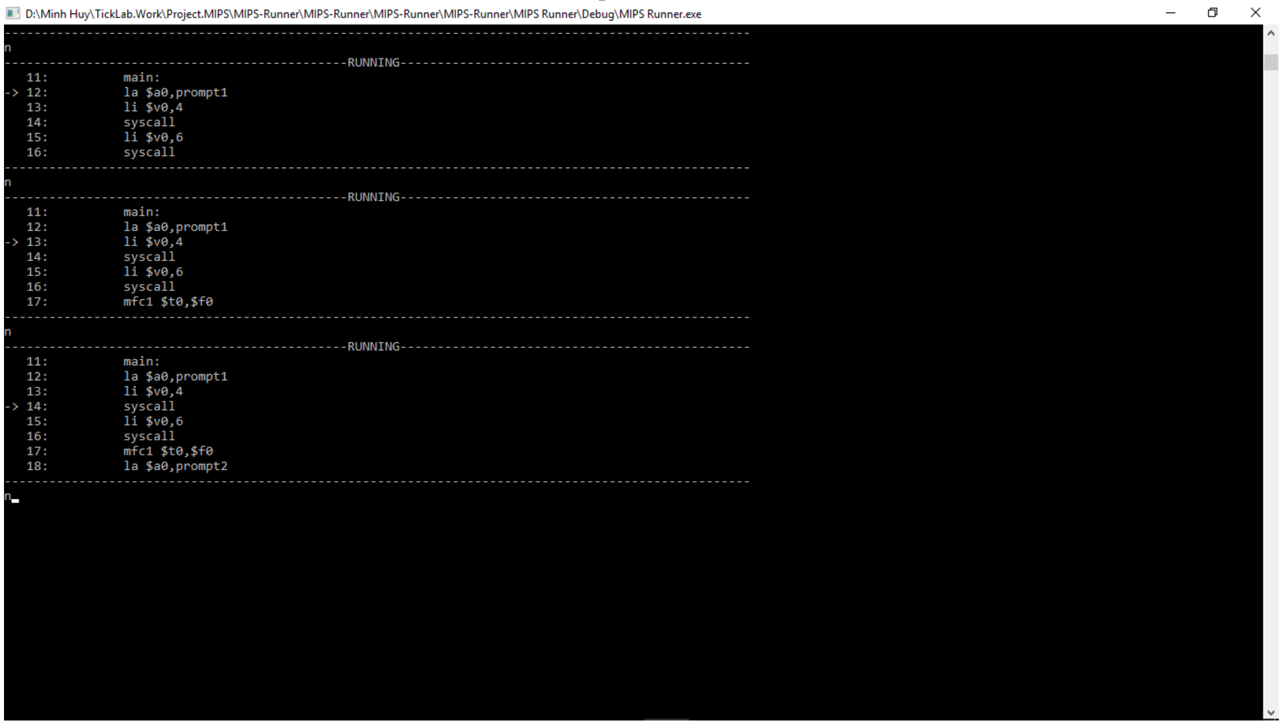
```
D:\Minh Huy\TickLab.Work\Project.MIPS\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS Runner\Debug\MIPS Runner.exe
--------------------------------------------SOURCE CODE--------------------------------------------
1:               .data
2:               prompt1:
3:               .asciiz "Input first real number: \n"
4:               prompt2:
5:               .asciiz "Input second real number: \n"
6:               prompt3:
7:               .asciiz "Sum: "
8:               prompt4:
9:               .asciiz "Sub: "
10:              .text
11:              main:
12:              la $a0,prompt1
13:              li $v0,4
14:              syscall
15:              li $v0,6
16:              syscall
17:              mfc1 $t0,$f0
18:              la $a0,prompt2
19:              li $v0,4
20:              syscall
21:              li $v0,6
22:              syscall
23:              mfc1 $t1,$f0
24:              F_SIGN_CHECKING:
25:              bgez $t0,F_GREATER_ZERO
26:              addi $t2,$zero,1
27:              sll $t2,$t2,31
28:              j END_F_SIGN_CHECKING
29:              F_GREATER_ZERO:
30:              add $t2,$zero,$zero
31:              END_F_SIGN_CHECKING:
32:              S_SIGN_CHECKING:
33:              bgez $t1,S_GREATER_ZERO
34:              addi $t3,$zero,1
35:              sll $t3,$t3,31
36:              j END_S_SIGN_CHECKING
37:              S_GREATER_ZERO:
38:              add $t3,$zero,$zero
39:              END_S_SIGN_CHECKING:
40:              F_GET_EXP:
41:              sll $t4,$t0,1
42:              srl $t4,$t4,24
43:              S_GET_EXP:
44:              sll $t5,$t1,1
45:              srl $t5,$t5,24
```

put Assembly on display



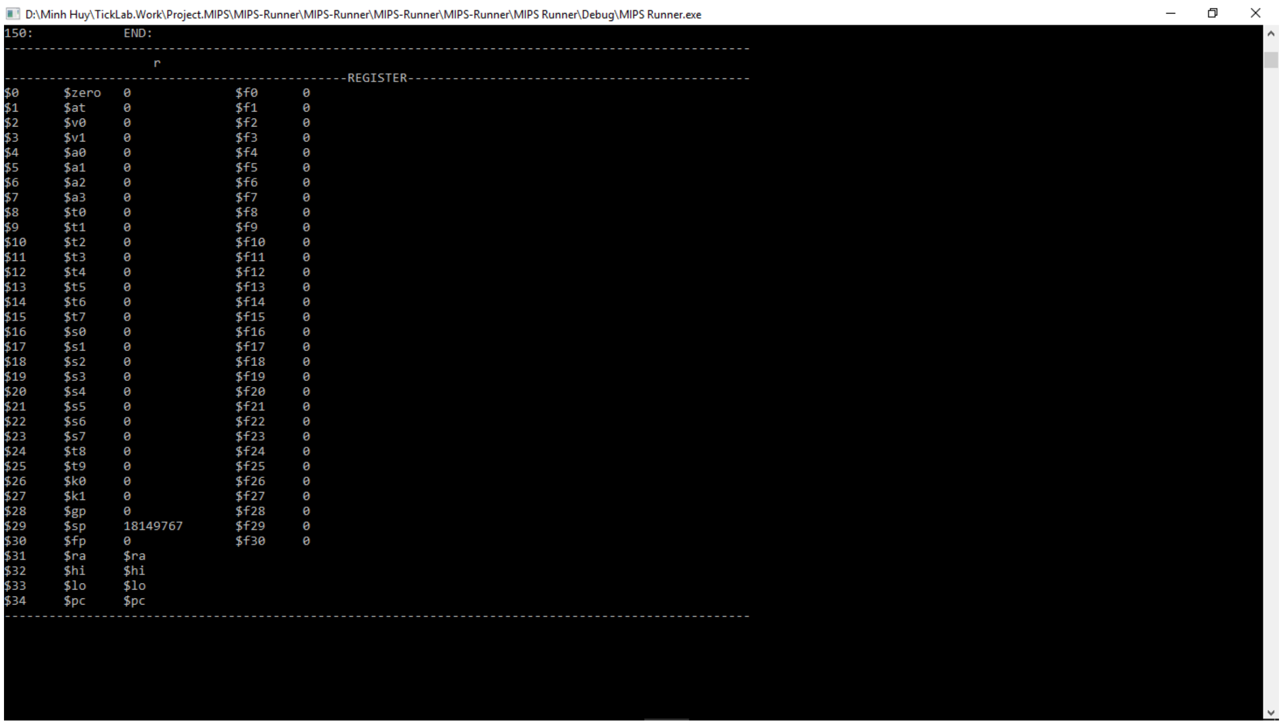
```
141:            j GREATER_THAN
142:            LESS_THAN:
143:            subi $v1,$v0,0x01000000
144:            bge $v1,$zero,END_STANDARDIZED
145:            sll $v0,$v0,1
146:            subi $s3,$s3,1
147:            j LESS_THAN
148:            END_STANDARDIZED:
149:            jr $ra
150:            END:
--------------------------------------------------------------------------------
---------------------------------------COMPILE ERROR-------------------
line 131:               "$s0" have to be an integer
--------------------------------------------------------------------------------

D:\Minh Huy\TickLab.Work\Project.MIPS\MIPS-Runner\MIPS-Runner\MIPS-Runner\M
Press any key to close this window . . .
```

compile error



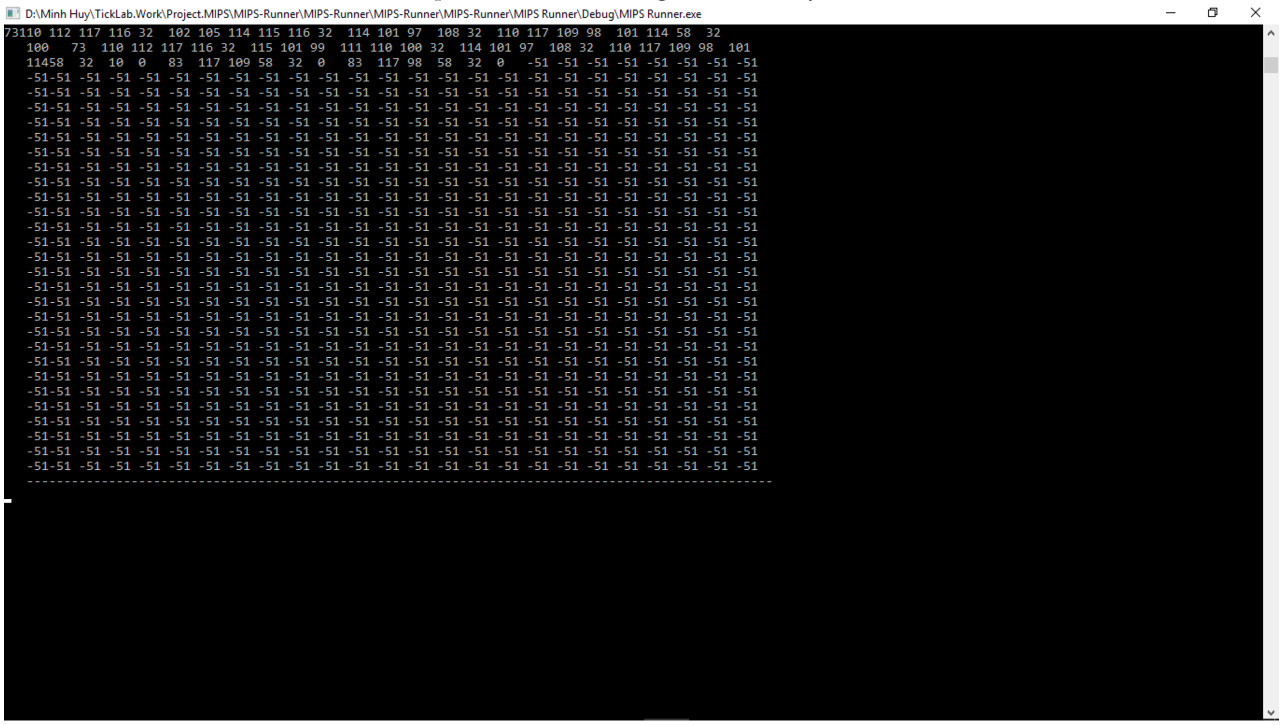
```
D:\Minh Huy\TickLab.Work\Project.MIPS\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS Runner\Debug\MIPS Runner.exe
--------------------------------------------------------------------------------
n
----------------------------------------------RUNNING---------------------------
   11:          main:
-> 12:          la $a0,prompt1
   13:          li $v0,4
   14:          syscall
   15:          li $v0,6
   16:          syscall
--------------------------------------------------------------------------------
n
----------------------------------------------RUNNING---------------------------
   11:          main:
   12:          la $a0,prompt1
-> 13:          li $v0,4
   14:          syscall
   15:          li $v0,6
   16:          syscall
   17:          mfc1 $t0,$f0
--------------------------------------------------------------------------------
n
----------------------------------------------RUNNING---------------------------
   11:          main:
   12:          la $a0,prompt1
   13:          li $v0,4
-> 14:          syscall
   15:          li $v0,6
   16:          syscall
   17:          mfc1 $t0,$f0
   18:          la $a0,prompt2
--------------------------------------------------------------------------------
n
```

press 'n' to execute next linepress 'u' to show line are executing (not executing)



```
D:\Minh Huy\TickLab.Work\Project.MIPS\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS Runner\Debug\MIPS Runner.exe
150:            END:
----------------------------------------------------------------------------------------------------
                    r
---------------------------------------------REGISTER-----------------------------------------------
$0      $zero   0               $f0     0
$1      $at     0               $f1     0
$2      $v0     0               $f2     0
$3      $v1     0               $f3     0
$4      $a0     0               $f4     0
$5      $a1     0               $f5     0
$6      $a2     0               $f6     0
$7      $a3     0               $f7     0
$8      $t0     0               $f8     0
$9      $t1     0               $f9     0
$10     $t2     0               $f10    0
$11     $t3     0               $f11    0
$12     $t4     0               $f12    0
$13     $t5     0               $f13    0
$14     $t6     0               $f14    0
$15     $t7     0               $f15    0
$16     $s0     0               $f16    0
$17     $s1     0               $f17    0
$18     $s2     0               $f18    0
$19     $s3     0               $f19    0
$20     $s4     0               $f20    0
$21     $s5     0               $f21    0
$22     $s6     0               $f22    0
$23     $s7     0               $f23    0
$24     $t8     0               $f24    0
$25     $t9     0               $f25    0
$26     $k0     0               $f26    0
$27     $k1     0               $f27    0
$28     $gp     0               $f28    0
$29     $sp     18149767        $f29    0
$30     $fp     0               $f30    0
$31     $ra     $ra
$32     $hi     $hi
$33     $lo     $lo
$34     $pc     $pc
----------------------------------------------------------------------------------------------------
```

press 'r' to show register memory



```
D:\Minh Huy\TickLab.Work\Project.MIPS\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS-Runner\MIPS Runner\Debug\MIPS Runner.exe
73110 112 117 116 32  102 105 114 115 116 32  114 101 97  108 32  110 117 109 98  101 114 58  32
   100   73  110 112 117 116 32  115 101 99  111 110 100 32  114 101 97  108 32  110 117 109 98  101
   11458  32  10  0   83  117 109 58  32  0   83  117 98  58  32  0   -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   -51-51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51 -51
   ----------------------------------------------------------------------------------------------------
```

press 'v' to show variable memory

4. Conclusion

4.1. Summary:

- Completed most of comments of assembly
- Comprehended how to write an assembly

- Understand how a computer operated, oraganised and designed

4.2. **Evaluating working processing:**

- Applying Git for operating project
- To aware the function of pare coding
- Frequent Interaction for planning idea,evaluating progress
- Comprehended the worthy value of using class,enhanced ability of applying object oriented programming

4.3. **Lesson after project:**

- Drawing an sufficient picture clearly before starting to code
- Practicing enhancing ability of algorithm for dealing with problems fast,adaptive for fixing after.

This is the end of the report.