CS/EE 320 Computer Organization and Assembly Language (Spring 2025)

Electrical Engineering Department
SBASSE LUMS

Instructor: Dr. Shahid Masud
TA: Ibraheem

Note: QtSpim and VS Code are already installed in the lab systems. But if you want to download and install these software tools on your systems or laptops you can follow the following downloading and installation procedures.

Introduction:

QtSpim is MIPS32 Simulator that reads and executes MIPS32 Assembly Language programs and displays the processor's registers and memory. It cannot run the binary (compiled) code. QtSpim provides a simple debugger and small set of operating system services.

It can run on multiple platforms like Windows, Mac OS X, and Linux.

QtSpim does not have built-in text editor. Therefore, it is needed to download and install any suitable text editor. There are well known text editors available such as Notepad++, Atom, VS code, etc. We are using VS Code in this course. If you want to use your favorite text editor, then feel free to use it.

Downloading:

QtSpim can be easily downloaded from the following link.

https://sourceforge.net/projects/spimsimulator/

VS code can be downloaded from the following link.

https://code.visualstudio.com/download

Installation:

- Double click the downloaded file.
- Click “NEXT” when the installation window pops up.
- Agree the terms & conditions and select the location where you want to install QtSpim.
- After Installation is completed, click “FINISHED”.

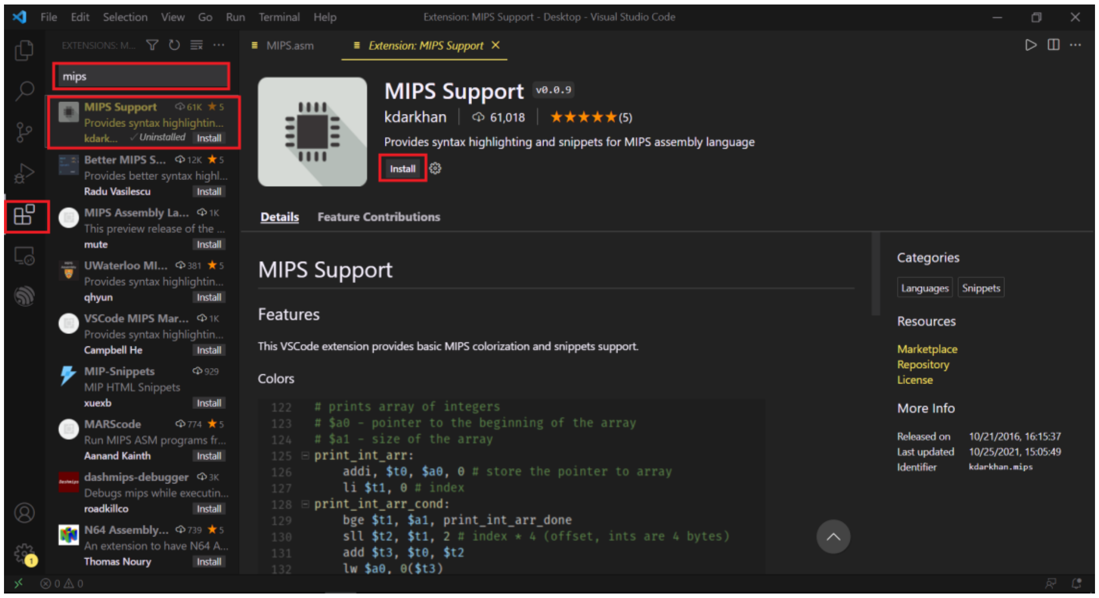
Similarly, you can install the VS code.

VS Code Extensions:

Different extensions are used in VS Code to facilitate the programmer to write programs with ease and provide better readability in terms of text colors, auto-completion etc.

We need to install MIPS Support extension in VS Code for this purpose.

- Click on the “EXTENSIONS SYMBOL” on left side tool bar.
- Type MIPS SUPPORT in the search box.
- From the search results list, click the “MIPS SUPPORT”.
- Click on install button.
- The extension will be installed.



MIPS Support

Provides syntax highlighting and snippets for MIPS assembly language

MIPS Support

Features

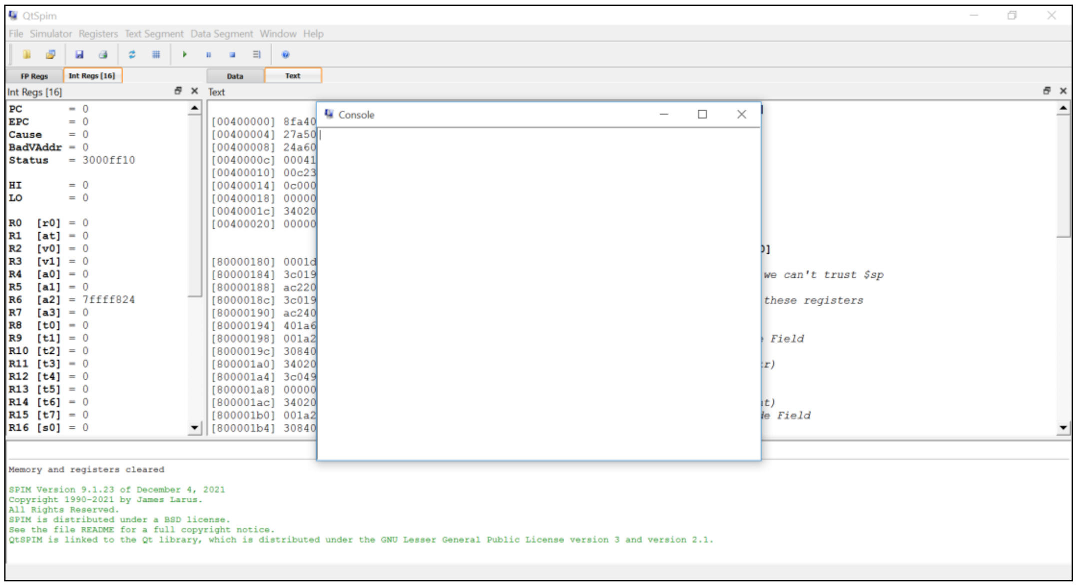
This VSCode extension provides basic MIPS colorization and snippets support.

Colors

```
# prints array of integers
# $a0 - pointer to the beginning of the array
# $a1 - size of the array
print_int_arr:
    addi, $t0, $a0, 0 # store the pointer to array
    li $t1, 0 # index
print_int_arr_cond:
    bge $t1, $a1, print_int_arr_done
    sll $t2, $t1, 2 # index * 4 (offset, ints are 4 bytes)
    add $t3, $t0, $t2
    lw $a0, 0($t3)
```

Overview of QtSpim:

When QtSpims starts, two windows will open as shown in figure below. One window is for console which will display output information when you run your program. Another window is QtSpim main window.


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
Text
Console

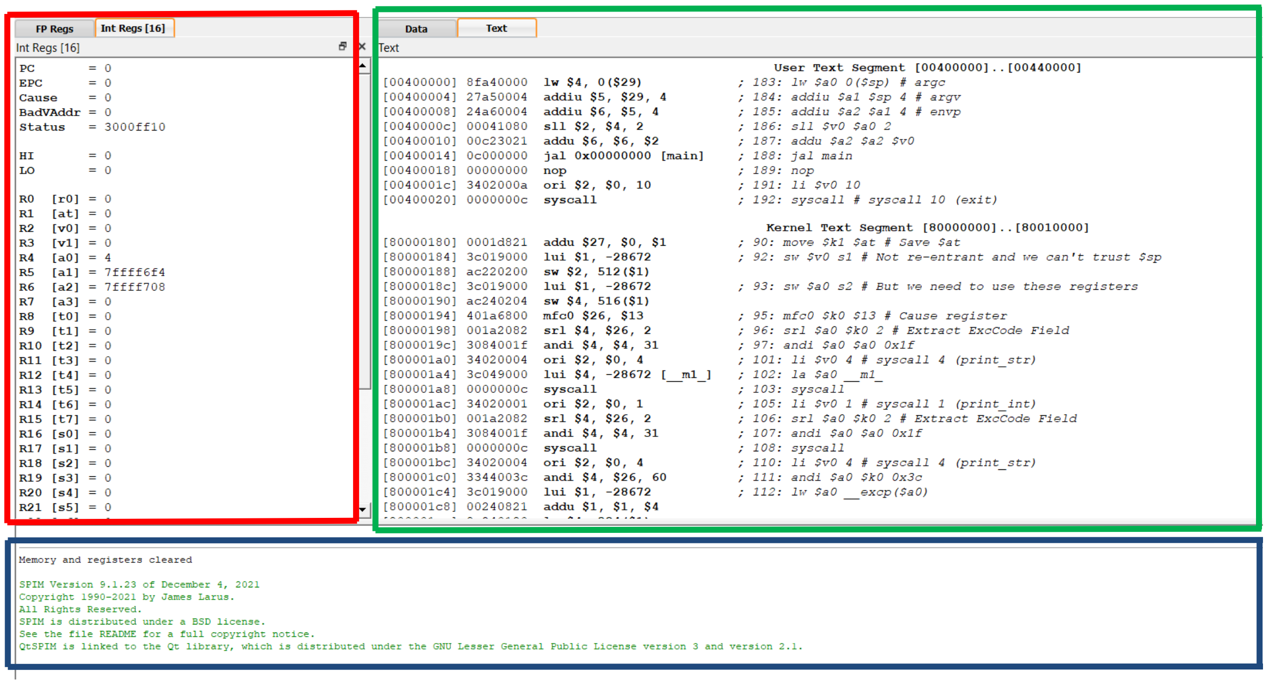
```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 0
R6  [a2] = 7ffff824
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
```

```
[00400000] 8fa40
[00400004] 27a50
[00400008] 24a60
[0040000c] 00041
[00400010] 00c23
[00400014] 0c000
[00400018] 00000
[0040001c] 34020
[00400020] 00000

[80000180] 0001d
[80000184] 3c019
[80000188] ac220
[8000018c] 3c019
[80000190] ac240
[80000194] 401a6
[80000198] 001a2
[8000019c] 30840
[800001a0] 34020
[800001a4] 3c049
[800001a8] 00000
[800001ac] 34020
[800001b0] 001a2
[800001b4] 30840
```

```
Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021
Copyright 1990-2021 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.
```

It is divided into three main parts as shown in figure below.

- The left pan which displays the registers and their values as the instruction gets executed. You can select either integer registers tab or floating point registers tab.
- The right big pan which displays the text segment (shows instructions) and data segment. You can select either text tab or data tab.
- The bottom pan which displays the QtSpim messages. If there is an error in your code or your code gets run successfully, it will show corresponding messages in this pan.



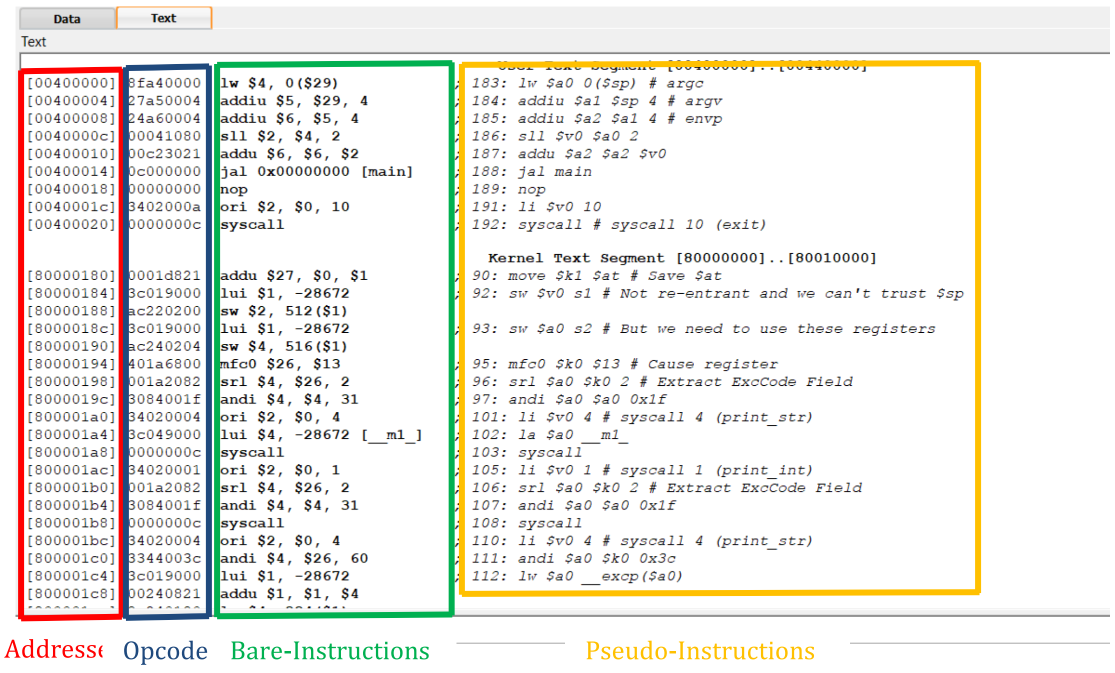
```
PC        = 0
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 0
R2   [v0] = 0
R3   [v1] = 0
R4   [a0] = 4
R5   [a1] = 7ffff6f4
R6   [a2] = 7ffff708
R7   [a3] = 0
R8   [t0] = 0
R9   [t1] = 0
R10  [t2] = 0
R11  [t3] = 0
R12  [t4] = 0
R13  [t5] = 0
R14  [t6] = 0
R15  [t7] = 0
R16  [s0] = 0
R17  [s1] = 0
R18  [s2] = 0
R19  [s3] = 0
R20  [s4] = 0
R21  [s5] = 0
```

```
                                                 User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscall 10 (exit)

                                                Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4              ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]     ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                    ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1              ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2             ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31            ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                    ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4              ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60           ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672             ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
```

```
Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021
Copyright 1990-2021 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.
```

The text segment displays the following information (as shown in figure below):

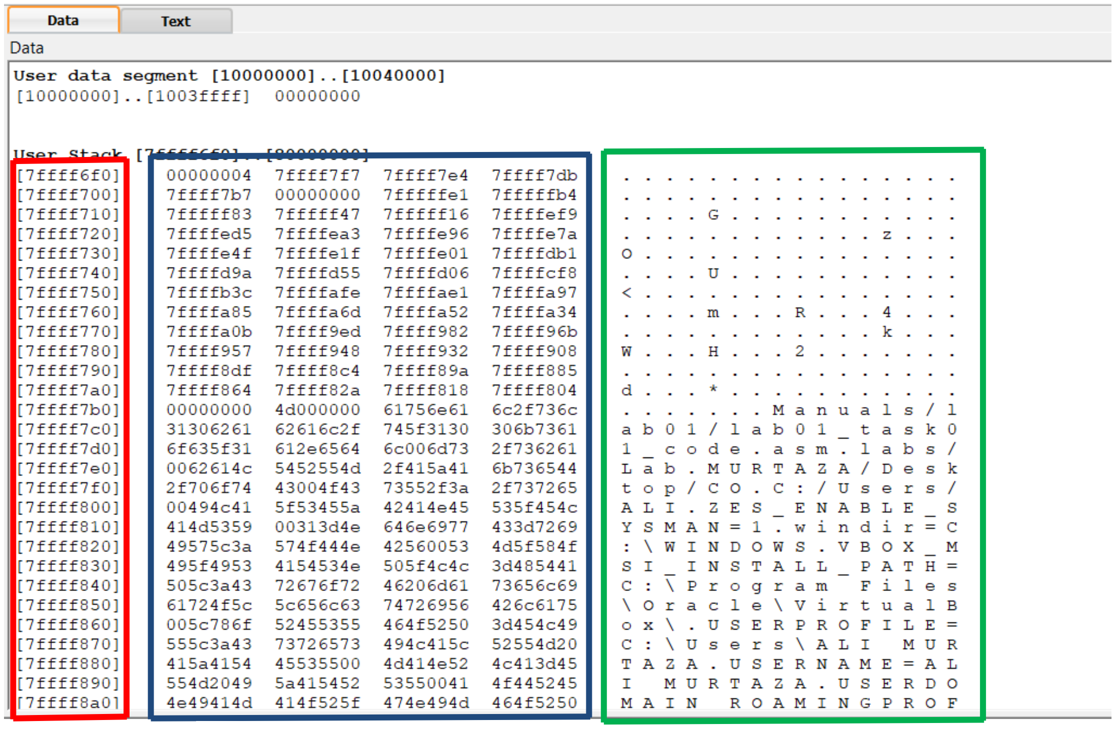
- Memory Addresses of the instructions.
- OpCode of the instructions.
- Bare-Instructions and
- Pseudo-Instructions



```
[00400000] 8fa40000  lw $4, 0($29)             ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4          ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4           ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2             ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2           ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]     ; 188: jal main
[00400018] 00000000  nop                       ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10            ; 191: li $v0 10
[00400020] 0000000c  syscall                   ; 192: syscall # syscall 10 (exit)

                                                 Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1          ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672            ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672            ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13             ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2            ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31           ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4             ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]    ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                   ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1             ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2            ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31           ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                   ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4             ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60          ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672            ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
```

The data segment displays the following information (as shown in figure below);

- Memory Addresses of the data
- Hex Representation of the data
- ASCII Representation of the data

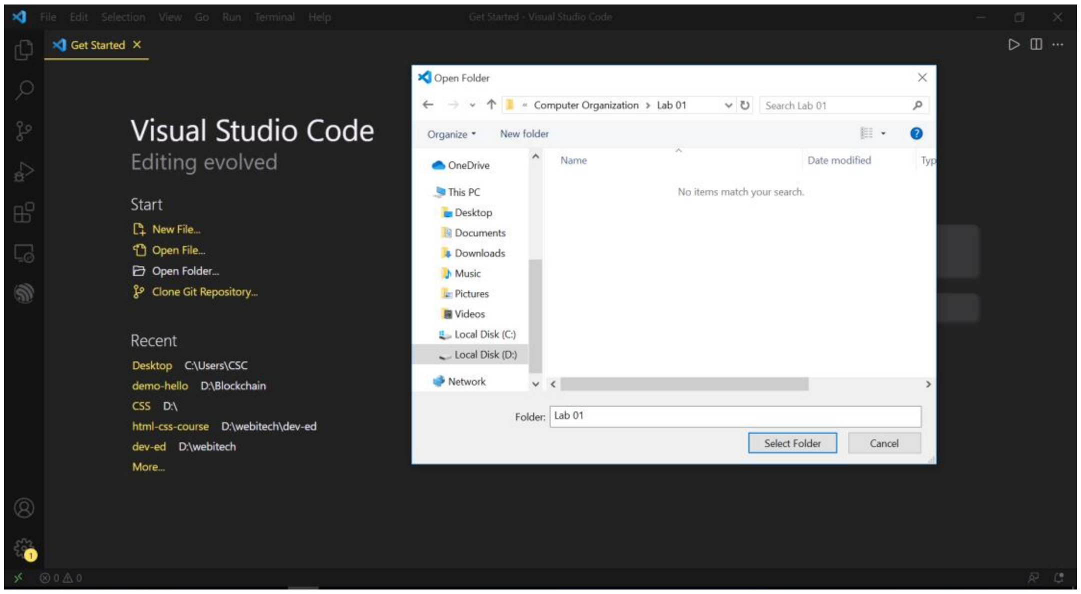


```
User data segment [10000000]..[10040000]
[10000000]..[1003ffff]  00000000


User Stack [7ffff6f0]..[80000000]
[7ffff6f0]  00000004  7ffff7f7  7ffff7e4  7ffff7db   . . . . . . . . . . . . . . . .
[7ffff700]  7ffff7b7  00000000  7fffffe1  7fffffb4   . . . . . . . . . . . . . . . .
[7ffff710]  7fffff83  7fffff47  7fffff16  7ffffef9   . . . . G . . . . . . . . . . .
[7ffff720]  7ffffed5  7ffffea3  7ffffe96  7ffffe7a   . . . . . . . . . . . . z . . .
[7ffff730]  7ffffe4f  7ffffe1f  7ffffe01  7ffffdb1   O . . . . . . . . . . . . . . .
[7ffff740]  7ffffd9a  7ffffd55  7ffffd06  7ffffcf8   . . . . U . . . . . . . . . . .
[7ffff750]  7ffffb3c  7ffffafe  7ffffae1  7ffffa97   < . . . . . . . . . . . . . . .
[7ffff760]  7ffffa85  7ffffa6d  7ffffa52  7ffffa34   . . . . m . . . R . . . 4 . . .
[7ffff770]  7ffffa0b  7ffff9ed  7ffff982  7ffff96b   . . . . . . . . . . . . k . . .
[7ffff780]  7ffff957  7ffff948  7ffff932  7ffff908   W . . . H . . . 2 . . . . . . .
[7ffff790]  7ffff8df  7ffff8c4  7ffff89a  7ffff885   . . . . . . . . . . . . . . . .
[7ffff7a0]  7ffff864  7ffff82a  7ffff818  7ffff804   d . . . * . . . . . . . . . . .
[7ffff7b0]  00000000  4d000000  61756e61  6c2f736c   . . . . . . . . M a n u a l s / l
[7ffff7c0]  31306261  62616c2f  745f3130  306b7361   a b 0 1 / l a b 0 1 _ t a s k 0
[7ffff7d0]  6f635f31  612e6564  6c006d73  2f736261   1 _ c o d e . a s m . l a b s /
[7ffff7e0]  0062614c  5452554d  2f415a41  6b736544   L a b . M U R T A Z A / D e s k
[7ffff7f0]  2f706f74  43004f43  73552f3a  2f737265   t o p / C O . C : / U s e r s /
[7ffff800]  00494c41  5f53455a  42414e45  535f454c   A L I . Z E S _ E N A B L E _ S
[7ffff810]  414d5359  00313d4e  646e6977  433d7269   Y S M A N = 1 . w i n d i r = C
[7ffff820]  49575c3a  574f444e  42560053  4d5f584f   : \ W I N D O W S . V B O X _ M
[7ffff830]  495f4953  4154534e  505f4c4c  3d485441   S I _ I N S T A L L _ P A T H =
[7ffff840]  505c3a43  72676f72  46206d61  73656c69   C : \ P r o g r a m   F i l e s
[7ffff850]  61724f5c  5c656c63  74726956  426c6175   \ O r a c l e \ V i r t u a l B
[7ffff860]  005c786f  52455355  464f5250  3d454c49   o x \ . U S E R P R O F I L E =
[7ffff870]  555c3a43  73726573  494c415c  52554d20   C : \ U s e r s \ A L I   M U R
[7ffff880]  415a4154  45535500  4d414e52  4c413d45   T A Z A . U S E R N A M E = A L
[7ffff890]  554d2049  5a415452  53550041  4f445245   I   M U R T A Z A . U S E R D O
[7ffff8a0]  4e49414d  414f525f  474e494d  464f5250   M A I N   R O A M I N G P R O F
```

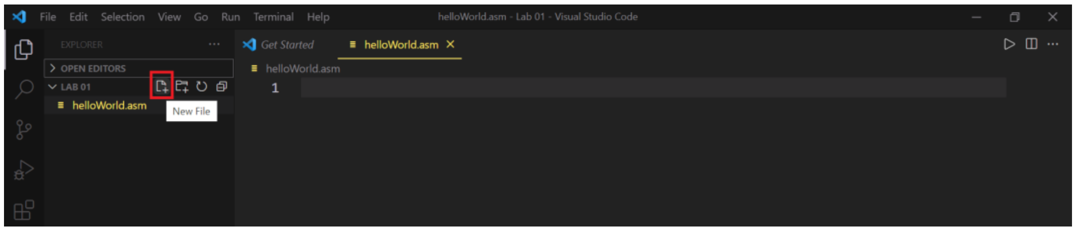
Addresses Data (Hex Representation) Data (ASCII Representation)

Hello World Program:

- Open the VS Code text editor and click on the "OPEN FOLDER".
- In the pop up window, create a new folder at a suitable location. And select that folder as shown in the figure below.



- Then click on the “NEW FILE ICON” on left side pan of VS Code as shown in figure below. Type the name of the file for example “HelloWorld.asm”.
- It is better to use “**asm**” file extension.
- Double click on that file to open it.

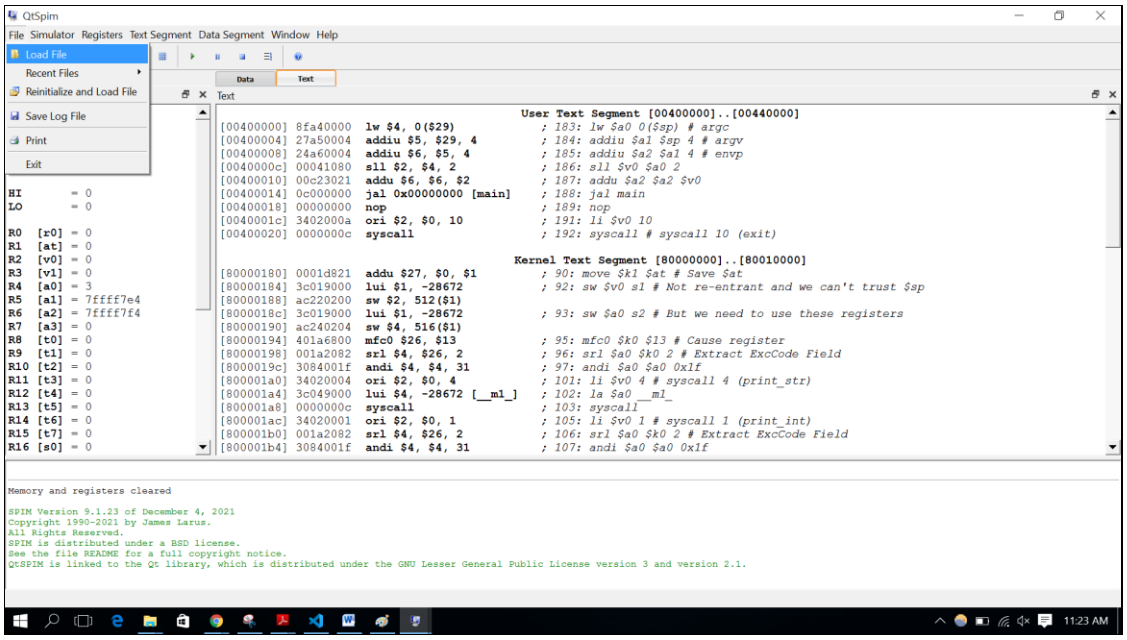


- Type the following code.
- You can also copy it and paste it to the VS Code.

```
.data
    message: .asciiz "Hello World!!\n"
.text
.globl main
.ent main
main:
    li $v0, 4
      la $a0, message
      syscall
    li $v0, 10
    syscall # all done!
```

```
.end main
```

- Now save the file.
- Open the QtSpim and go to “file -> load file” and select that file which you have created as shown in the figure below.

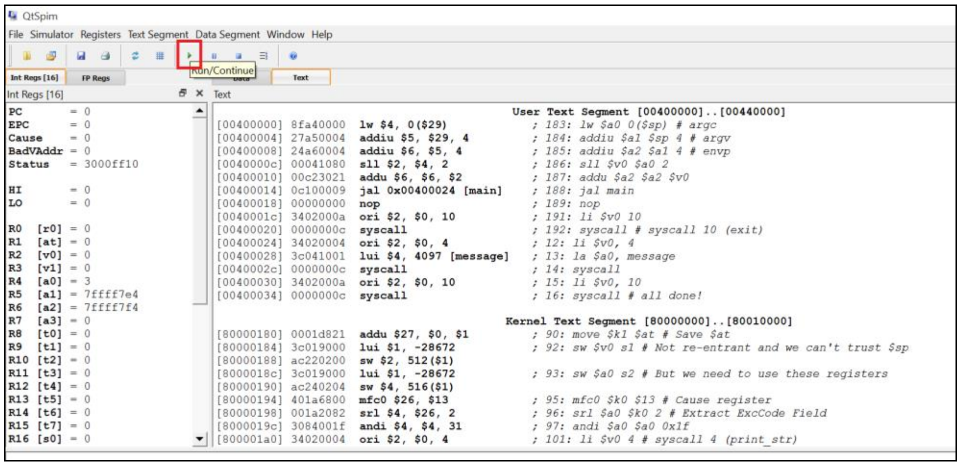

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Load File
Recent Files
Reinitialize and Load File
Save Log File
Print
Exit
Data
Text
Text

```
HI        = 0
LO        = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 3
R5  [a1] = 7ffff7e4
R6  [a2] = 7ffff7f4
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
```

```
                              User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscall 10 (exit)

                             Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4              ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]     ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                    ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1              ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2             ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31            ; 107: andi $a0 $a0 0x1f
```

```
Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021
Copyright 1990-2021 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.
```

- Run the code by clicking the run button as shown in the figure below.



```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 3
R5  [a1] = 7ffff7e4
R6  [a2] = 7ffff7f4
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
```

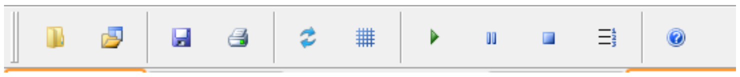
```
                                                  User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)                  ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4               ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4                ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2                  ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2                ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]          ; 188: jal main
[00400018] 00000000  nop                            ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10                 ; 191: li $v0 10
[00400020] 0000000c  syscall                        ; 192: syscall # syscall 10 (exit)
[00400024] 34020004  ori $2, $0, 4                  ; 12: li $v0, 4
[00400028] 3c041001  lui $4, 4097 [message]         ; 13: la $a0, message
[0040002c] 0000000c  syscall                        ; 14: syscall
[00400030] 3402000a  ori $2, $0, 10                 ; 15: li $v0, 10
[00400034] 0000000c  syscall                        ; 16: syscall # all done!

                                                Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1               ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672                 ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672                 ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13                  ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2                 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31                ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4                  ; 101: li $v0 4 # syscall 4 (print_str)
```

- “Hello World!!” will be shown in the console window as shown below.

`

Overview of Toolbar:

Each icon of the toolbar is explained below from left to the right:



- The first icon is used for loading the assembly program file, which needs to be executed by the QtSpim simulator.
- The second icon is used for reinitializing the simulator and loading the assembly program file.
- The third icon is used for saving the registers status (values), data segment, text segment and console output to the log file.
- The fourth icon is used to print the registers status (values), data segment, text segment and console.
- The fifth icon is used to clear the registers value.
- The sixth icon is used to reinitialize the simulator, which means you need to load the assembly program file again.
- The seventh icon (run/continue) is used to run/execute the whole program at once or at one click.
- The eighth icon (pause) is used to pause or halt the program execution. And you can continue the program execution by clicking the seventh icon (run/continue).

- The ninth icon (stop) is used to stop the program execution.
- The tenth icon (single step) is used to execute the code one line at one click. This way you can observe how the values of registers are changing by each instruction.

Task 01:

You are provided with an assembly program file named "task01_code.asm". Load this file to the QtSpim simulator and execute it. And see the results on console window.

Task 02:

Save the registers value and console output to the log file by using "save log file" icon.

Task 03:

Now clear the registers value by clicking the "clear registers" icon. Now execute the same program again by clicking the "single step" icon and observe the program execution flow and how the values of registers are changing. Explain it in your words.

Grading Scheme:

Task#	Marks	Submission
Task1	5	After running the code show the output to TA.
Task2	5	Show the saved log file to TA and upload it on dropbox.
Task3	10	Upload the explanation file on dropbox before leaving the lab.