MIPS & SPIM Overview

TA in charge: Yongwook Lee
E-mail: wooks1998@kaist.ac.kr

I. MIPS

- ✓ **MIPS**: A reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies (formerly MIPS Computer Systems). Due to its relative simplicity and consistency, it is often favored for learning basics of assembly programing and ISA.
- ✓ **Assembly language**: A low-level programming language that is in between high-level language (like C/C++) and machine language (binary). While it consists of symbolic version of instructions, it can also contain pseudoinstructions and labels for the convenience of programming. Assembler converts assembly code into executable machine code by translating instructions, pseudoinstructions, and labels into binary.
- ✓ When you are writing a MIPS assembly program, please try to follow the MIPS register usage convention. (see SPIM quick reference in **IV. Notes**)

II. SPIM

- ✓ SPIM is a MIPS simulator which can read and execute assembly language programs written for **MIPS32 ISA**. It is named after the reversal of the letters "MIPS".
  *MIPS32 is a 32-bit ISA which was published in 1999. It is supported by most modern MIPS CPUs.
- ✓ Though it supports most of the instructions and pseudoinstructions of MIPS32 ISA, you can check the SPIM quick reference (see **IV. Notes**) to ensure if it supports a specific instruction.
- ✓ **Basic structure of MIPS assembly code for SPIM**: A MIPS assembly code runnable on SPIM must contain a 'main' label which designate the address that will be called by the loader of the simulator.
  In addition, it should specify an assembler directive (begins with '.') before each of the beginning of a data section (with *.data*) or a text (instruction) section (with *.text*). This directive tells the assembler (in this case, simulator) to allocate the following data/text in the data/text segment of virtual memory. (see the next page for an example.)

➢ **Example: sample_HelloWorld.s**

```
1  # PROGRAM: Hello, World!
2
3      .data        # Data declaration section
4
5      out_string: .asciiz "\nHello, World!\n"
6      list:
7          .word 1
8
9      .text        # Start of code section
10
11 main:
12
13     li $v0, 4
14     la $a0, out_string
15     syscall
16
17     li $a0, 3
18     li $a1, 4
19     move $s0, $ra
20     jal functions
21
22
23     move $ra, $s0
24     jr $ra
25
```

Data and Text section
in MIPS assembly code

Data | Text

Data

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    6c65480a  202c6f6c  6c726f57  000a2164    . H e l l o ,   W o r l d ! . .
[10010010]    00000001  00000000  00000000  00000000    . . . . . .
[10010020]..[1003ffff]  00000000
```

Data | Text

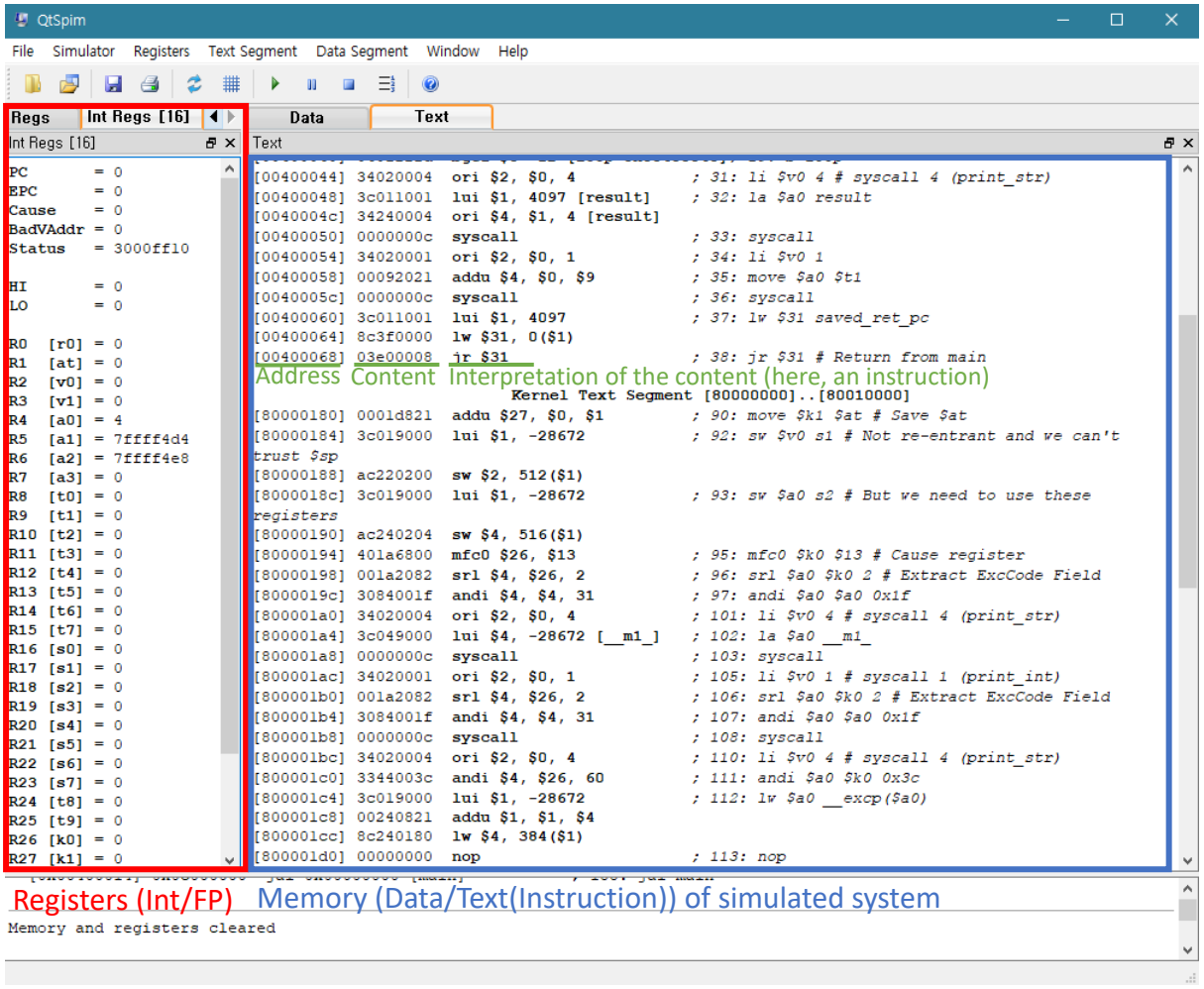
Text

```
                                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)
```

Data and Text segment in memory

III. QtSPIM

- ✓ QtSPIM is the latest, GUI version of SPIM based on Qt framework.
- ✓ You can download the latest version for Windows, Linux, and macOS from here.
- ✓ After installation, you could find a **manual** for this simulator from **[Menu bar]-[Help]-[View Help]**.
- ✓ When you launch QtSPIM, you will see a window similar to the one on the next page. The simulator displays current snapshot of registers (on the left pane) and the simulated virtual memory (on the right pane).
- ✓ Before loading an assembly program, what you will see in the text segment are loader (in User Text Segment) and exception handler (in Kernel Text Segment).



✓ You can load an assembly program by clicking **[Menu bar]-[File]-[Load File]** or the equivalent button ( ).
When you are trying to restart your simulation, you could click **[Reinitialize and Load File]**. This will reset both the registers and the memory to the *initial state* of the simulator (any data and instructions loaded from assembly program will also be cleared).

✓ If you've loaded a file, the data and the instructions within the file will be loaded to the data segment and the text segment of the memory.

✓ **[Menu bar]-[Simulator]-[Run/Continue]** will run a simulation until the termination (exit syscall) or a pause/stop or a breakpoint.
[Menu bar]-[Simulator]-[Single step] will run a simulation instruction-by-instruction.

✓ In the text segment pane, you can set a breakpoint on an instruction by right-clicking on the line of the instruction and selecting "Set Breakpoint".

✓ SPIM provides some operating-system-like services (print to/read from console, exit, file system calls, etc.) with `syscall` instruction. Before invoking a `syscall` instruction, you need to set a system call code into register $v0 and an argument to a specific register. (See sample_test.s for an example.)
Below is the table about the system services supported in SPIM.

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $v0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $v0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $v0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

✓ You can see a console window if you set a check on **[Menu bar]-[Window]-[Console]**. This console window is the interface for print or read by `syscall` instruction.

IV. Notes

✓ For more information, read textbook *Appendix A: Assemblers, Linkers, and the SPIM Simulator*.

✓ SPIM Quick Reference: https://minnie.tuhs.org/CompArch/Resources/spim_ref.html
This quick reference contains most of the knowledge you would need throughout homework 1 and 2.