MIPS and QtSPIM Overview #1

TA in charge: Haejin Nam
E-mail: haejinnam@kaist.ac.kr

I. MIPS

- ✓ **MIPS**: a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies (formerly MIPS Computer Systems).
- ✓ **Assembly language**: a low-level programming language that has a very strong correspondence between the program's statements and the architecture's machine code instructions. Assembly language code is soon converted into executable machine code to be executed.
- ✓ **Basic Structure of MIPS Assembly code**: A MIPS assembly code consists of two parts: data section and text section. The **.data directive** tells the assembler to store the string in the program's data segment, and the **.text directive** tells the assembler to store the instructions in its text segment.
  - ➢ **Example: sample_HelloWorld.s**

```
# PROGRAM: Hello, World!

.data    # Data declaration section

out_string: .asciiz "\nHello, World!\n"
list:
  .word 1

.text

main: # Start of code section

  li $v0, 4
  la $a0, out_string
  syscall

  li $a0, 3
  li $a1, 4
  move $s0, $ra
  jal functions


  move $ra, $s0
  jr $ra
```

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

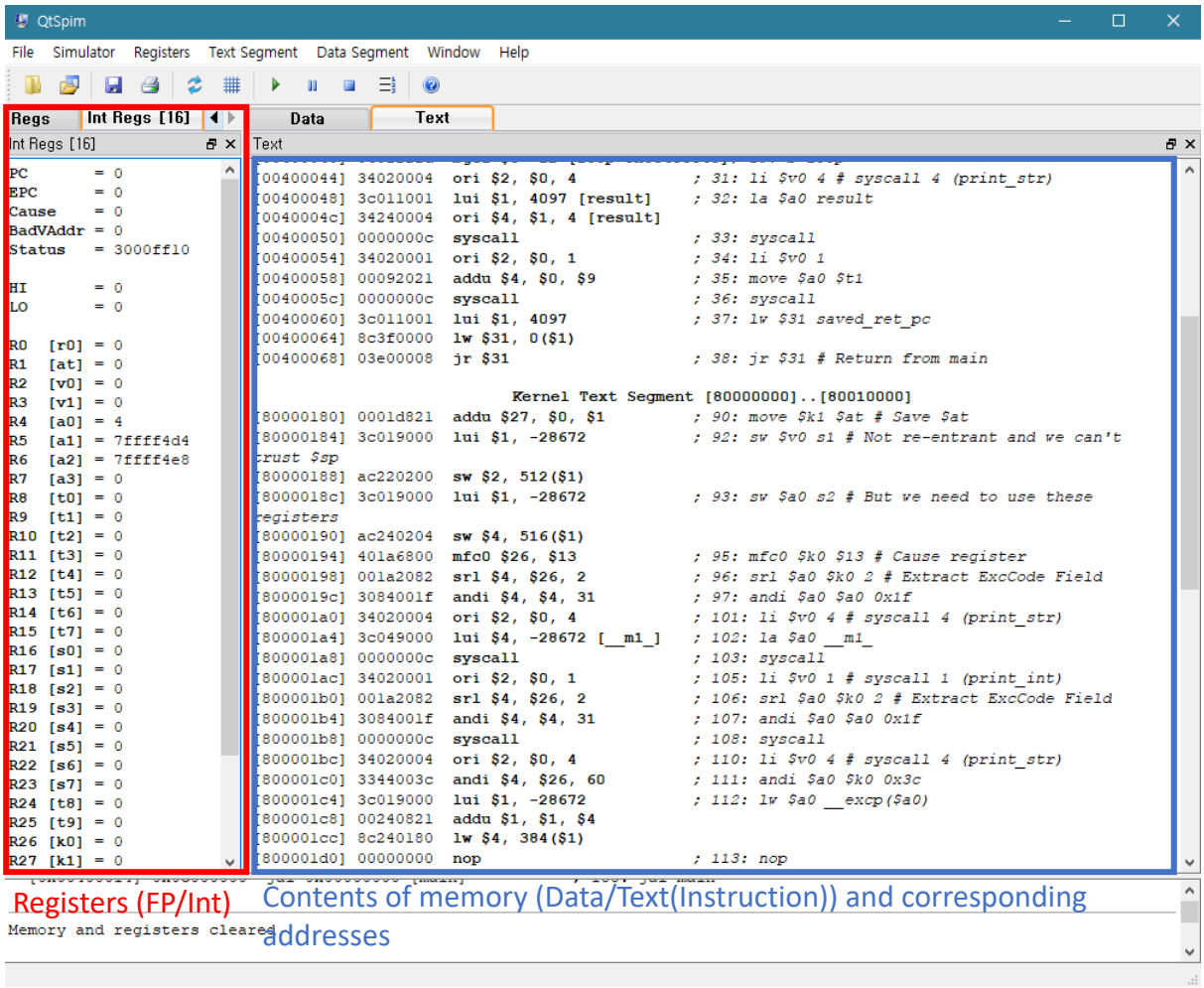
Text

```
                              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

II. QtSPIM

- ✓ Here we will cover the very basics of the simulator.
- ✓ **More detailed manual for the simulator is available at [Menu bar]-[Help]-[View Help]**.
- ✓ When you start the simulator, you will see a window as below.



- ✓ You can load an assembly program by [Menu bar]-[File]-[Load File] or  ; for convenience, you may select [Reinitialize and Load File].
- ✓ If you've loaded a file, the data and the instructions within the file will be added to the data segment and the text segment of the simulator.
- ✓ 'Reinitialize' here means that it resets both the registers and the memory to the *initial state* of the simulator (data and instructions loaded from any files will also be cleared).
- ✓ [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.
- ✓ At the text segment pane, you can set a breakpoint by right clicking on the line of an

instruction and selecting “Set Breakpoint”.

- ✓ You can see the console window if you set a check on [Menu bar]-[Window]-[Console].
- ✓ SPIM provides some simple 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. 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	