Computer Systems Architecture: Exercise Class Week 3

The purpose of this class is to start to become acquainted with MIPS assembly language, the MIPS simulator SPIM and to design a simple program in MIPS assembly language.

You can run the MIPS simulator SPIM by typing `H:\teach\ComputerSystemsArch\spim\pcspim.exe` at the 'Start'->'Run…' windows menu or at the command line. Doumentation for spim can be found at `http://www.informatics.sussex.ac.uk/users/michaelg/csa/SPIM_ps_docs.pdf` with a longer introduction found in Appendix A of the book 'Computer Organisation and Design' [`http://www.cs.wisc.edu/~larus/SPIM/cod-appa.pdf`]. Windows versions of SPIM can be found at `http://www.cs.wisc.edu/~larus/SPIM` should you wish to run it on your own system (or you could just copy the directory \spim* to your system).

SPIM simulates a very simple computer built with a MIPS processor. The input/output device is a teletype console for which a very simple set of **operating systems calls** provide a helpful interface to it (documentation section 1.5):

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		

How to use `syscall`

The first exercise gives you an example of how to use these operating system calls shown in the table above. The appropriate **System Call Code** needs to be contained in register **`$v0`**, this is done by the instruction

```
li $v0, <system call code as per above table>
```

then the **argument** registers need to be filled in (see Q1a below). Note that `li` refers to '**l**oad **i**mmediate' (i.e. load the specified small constant into the register) and `la` refers to '**l**oad **a**ddress' (i.e. load the memory address identified by the specified memory address label). **Only then**, once the register `$v0` **and** the argument registers have been set to the appropriate values should you then use the instruction `syscall` to call the operating system and have it provide the service you requested (see the SPIM documentation for details about how `syscall` works).

1a) The following program is the MIPS version of 'Hello World' (the assembler syntax is described in section 1.4 of the SPIM documentation, look there for explanations of '.data', '.text' etc) :

```
            .data
hi:         .asciiz "Hello World!\n"
            .text
            .globl main

main:       li $v0, 4       # syscall 4 print string
            la $a0, hi      # address of string
            syscall         # call the operating system
            li $v0, 10      # syscall 10 exit
            syscall         # call the operating system
```

first OS call

second OS call

Use a text editor to create a file 'hellow.s', load this into SPIM and get it to run. Read through the code making sure you fully understand it. What is `hi`, how is it used? Where is the 'hello world' string stored?

1b) Modify the above code to add another labelled string in memory which contains your name and then print ‘hello world’ and your name.

2) The SPIM manual explains what `.data`, `.asciiz`, `.text`, `.globl`, `.space` and `.word` are used for. **Look them up, you will need to use them in your programs.**

3) Write a program to input an integer number and then print it out.

4) Write a program to input two integers and print out the sum.

5) Write a program to input **n** numbers and print out their sum. (This needs to use conditional instructions)

MIPS registers and the convention governing their use

Register Name	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Expression evaluation and
v1	3	results of a function
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0	8	Temporary (not preserved across call)
t1	9	Temporary (not preserved across call)
t2	10	Temporary (not preserved across call)
t3	11	Temporary (not preserved across call)
t4	12	Temporary (not preserved across call)
t5	13	Temporary (not preserved across call)
t6	14	Temporary (not preserved across call)
t7	15	Temporary (not preserved across call)
s0	16	Saved temporary (preserved across call)
s1	17	Saved temporary (preserved across call)
s2	18	Saved temporary (preserved across call)
s3	19	Saved temporary (preserved across call)
s4	20	Saved temporary (preserved across call)
s5	21	Saved temporary (preserved across call)
s6	22	Saved temporary (preserved across call)
s7	23	Saved temporary (preserved across call)
t8	24	Temporary (not preserved across call)
t9	25	Temporary (not preserved across call)
k0	26	Reserved for OS kernel
k1	27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by function call)