COMPUTER ORGANIZATION (CDA – 3103)

SPRING 2005

Lab # 6: Introduction to SPIM

1 SIMULATOR AND DOCUMENTS

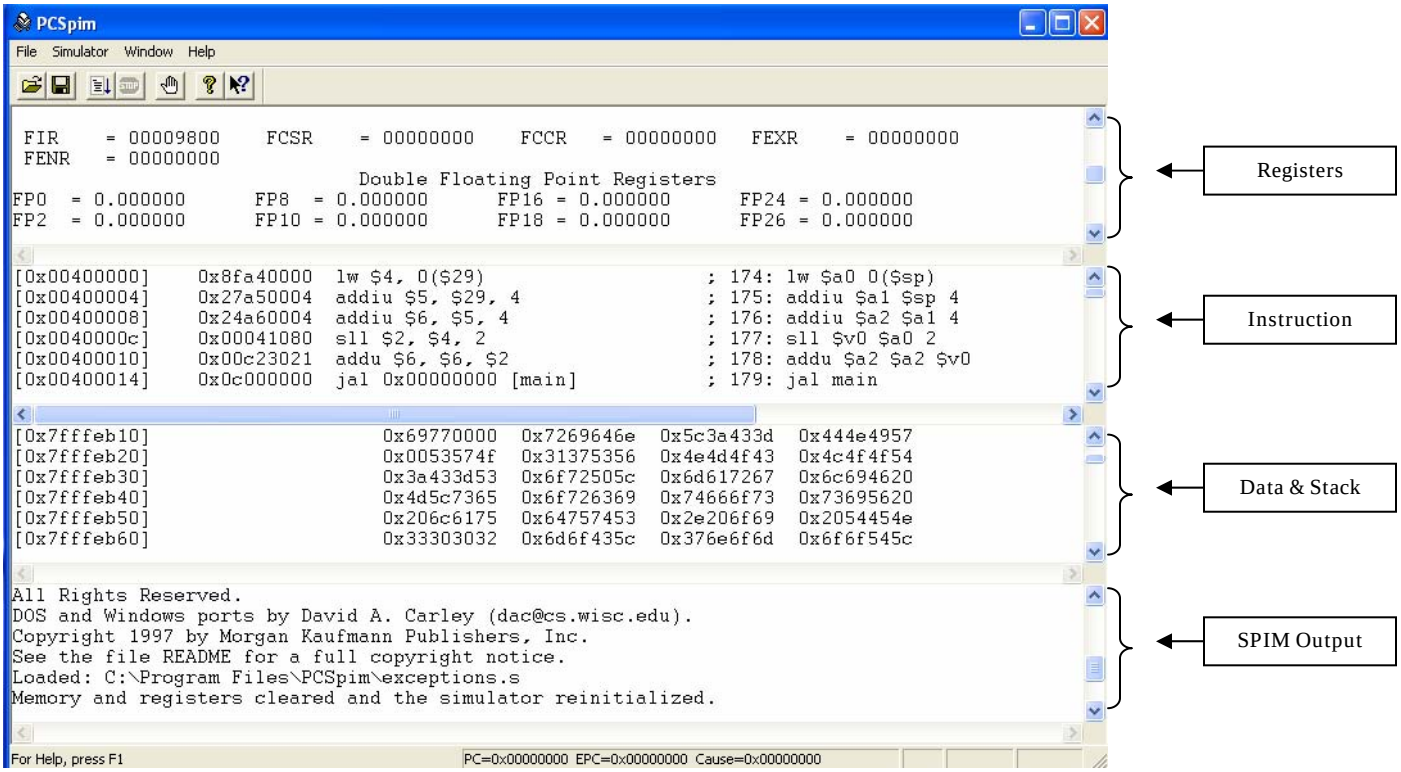
Kindly go on ***http://www.cs.wisc.edu/~larus/spim.html*** and download following files.

1. Section "Downloading SPIM"
    a. Windows based SPIM simulator *(http://www.cs.wisc.edu/~larus/SPIM/pcspim.zip)*
2. Section "Resource":
    a. Appendix A: Assemblers, Linkers, and the SPIM Simulator *(http://www.cs.wisc.edu/~larus/HP_AppA.pdf)*. This is appendix A of "Computer Organization & Design" (the text book of this course). This Appendix was available in 2nd edition; but has been removed in 3rd edition and made available online. So if you don't have 2nd edition, then download it.
    b. Getting Started with spim *(http://www.cs.wisc.edu/~larus/spim.pdf)*
    c. Getting Starting with PCSpim *(http://www.cs.wisc.edu/~larus/PCSpim.pdf)*
    d. SPIM Command-Line Options *(http://www.cs.wisc.edu/~larus/SPIM_command-line.pdf)*

2 INTRODUCTION

- SPIM is a software simulator that runs assembly language programs written for processors that implement the MIPS32 architecture.
- It contains a debugger and provides a few operating system–like services.
- MIPS processors can operate with either *big-endian* or *little-endian* byte order. So SPIM supports both of them.
- It has multiple versions:
    - Spim : Command line version of SPIM Simulator
    - PCSpim : Windows base version of SPIM Simulator *(preferred for these labs)*
    - XSpim : Linux Based Version of SPIM Simulator

3 PC SPIM



3.1 *System Calls*

SPIM provides a small set of operating-system-like services through the system call (***syscall***) instruction. To request a service, a program loads the system call code (see table given below) into register $v0 and arguments into registers $a0–$a3 (or $f12 for floating-point values).

System calls that return values put their results in register $v0 (or $f0 for floating-point results).

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 $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

4 *Arithmetical & Logical Operations*

Assembly instructions/memory layout of some of the arithmetical and logical operations is given below. For complete detail, see the document “Appendix A: Assemblers, Linkers, and the SPIM Simulator”.

Absolute Operation

- Absolute Instruction

```
abs rdest, rsrc
```

Addition Operations

- Addition Instruction (with overflow)

```
add rd, rs, rt
```

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

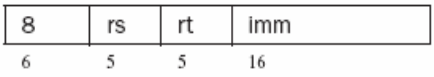
- Addition Instruction (without overflow)

```
addu rd, rs, rt
```

0	rs	rt	rd	0	0x21
6	5	5	5	5	6

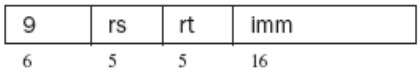
- Addition Immediate (with overflow)

```
addi rt, rs, imm
```



- Addition Immediate (without overflow)

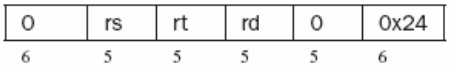
```
addiu rt, rs, imm
```



AND Operations

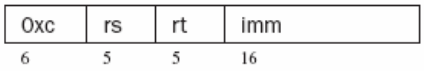
- AND Instruction

```
and rd, rs, rt
```



- AND Immediate Instruction

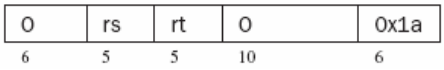
```
andi rt, rs, imm
```



Divide Operations

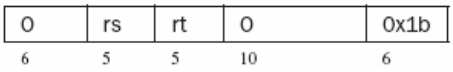
- Divide Instruction (with overflow)

```
div rs, rt
```



- Divide Instruction (without overflow)

```
divu rs, rt
```



- Divide Instruction (with overflow)

```
div rdest, rsrc1, src2
```

- Divide Instruction (without overflow)

```
divu rdest, rsrc1, src2
```

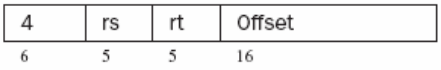
Branch Operations

- Branch Label

```
b label
```

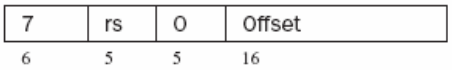
- Branch on Equal

```
beq rs, rt, label
```



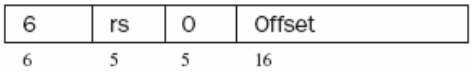
- Branch on greater than zero

```
bgtz rs, label
```



- Branch on less than equal to zero

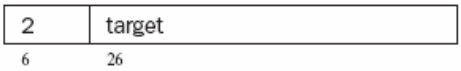
```
blez rs, label
```



Jump Operations

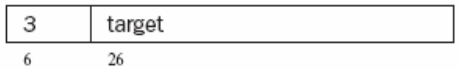
- Jump Instruction

```
j target
```



- Jump & Link

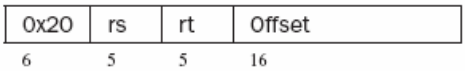
```
jal target
```



Load Instructions

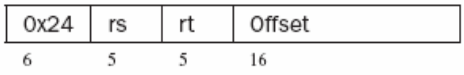
- Load byte

```
lb rt, address
```



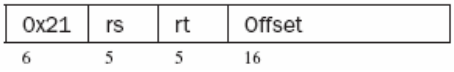
- Load unsigned byte

```
lbu rt, address
```



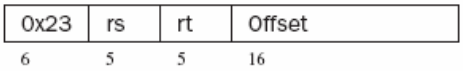
- Load halfword

```
lh rt, address
```



- Load word

```
lw rt, address
```



- Load double word

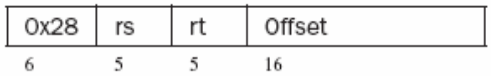
```
ld rdest, address
```

Loads the 64-bit quantity at *address* into registers *rdest* and *rdest + 1*.

Store Instructions

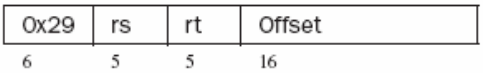
- Store Byte

```
sb rt, address
```



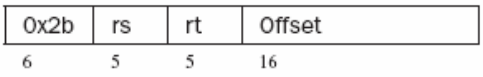
- Store Half-word

```
sh rt, address
```



- Store Word

```
sw rt, address
```



Data Movement Instructions

- Move

```
move rdest, rsrc
```

5 EXAMPLE PROGRAMS

5.1 PROBLEM 1

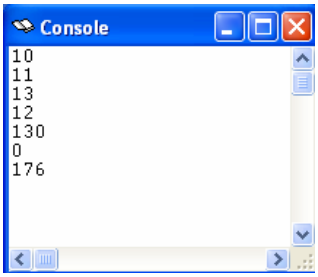
Question: Problem 6 of Appendix A: Using SPIM, write and test an adding machine program that repeatedly reads in integers and adds them into a running sum. The program should stop when it gets an input that is 0, printing out the sum at that point.

Solution:

1. Open Notepad.
2. Write following program

```
main:
        li $a0, 0
loop:
        li $v0, 5
        syscall                 # read next number from console
        add $a0, $a0, $v0
        bne $v0, $zero, loop    # loop back if current number was not ZERO
        li $v0, 1
        syscall                 # output the addition value
        li $v0, 10
        syscall                 # exit
```

3. Save as *Addition.asm* .
4. Open PCSpim
5. Load file *Addition.asm.*
6. Start simulation by pressing **F5** or by clicking menu Simulator → Go
7. It will prompt for "Run Parameters". Keep default values and press OK.
8. Now it will open a console (e.g. shown below). Write any 5 numbers (6th should be zero). The program will output a number which is the addition of first five numbers.



5.2 PROBLEM 2

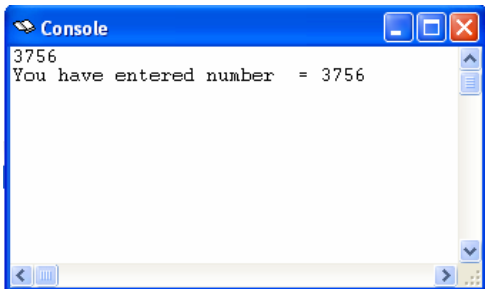
Question : Write a program in SPIM that takes a number as input. As a output it prints the string "You have entered number =" and then prints the input number.

Solution:

Write the program given below in notepad and run using steps given in Question 1.

main:

```
        .data
str:
        .asciiz " You have entered number  = "
        .text

        li $v0, 5
        syscall                         # read next number from console

        move $a1 , $v0                  # here a1 is acting as temporary variable to store input value

        li $v0, 4                       # system call code for print_str
        la $a0, str                     # address of string to print
        syscall                         # print the string

        li $v0, 1                       # system call code for print_int
        move $a0 , $a1
        syscall                         # print it

        li $v0, 10
        syscall                         # exit
```



6 PRACTICE ASSIGNMENT

(Problem 7 of Appendix A)

Its non-credit, optional assignment and doesn't carry any marks.

Using SPIM, write and test a program that reads in three integers and prints out the sum of the largest two of the three. Use the SPIM system calls described on pages A-43 and A-45. You can break ties arbitrarily.