Computer Architecture Homework #1
Due: September 7, 2007 (F)
(4 PM in ITT 305 mailbox or under my office door, ITT 313)

Type of Instruction	**Assembly Language**	**Register Transfer Language Description**
Memory Access (Load and Store)	lw $4, Mem	$4← [Mem]
	sw $4, Mem	Mem←$4
	lw $4, 16($3)	$4← [Mem at address in $3 + 16]
	sw $4, Mem	[Mem at address in $3 + 16]← $4
Move	move $4, $2	$4← $2
	li $4, 100	$4← 100
Load Address	la $5, mem	$4← load address of mem
Arithmetic Instruction (reg. operands only)	add $4, $2, $3	$4← $2 + $3
	mul $10, $12, $8	$10← $12 * $8 (32-bit product)
	sub $4, $2, $3	$4← $2 - $3
Arithmetic with Immediates (last operand must be an integer)	addi $4, $2, 100	$4← $2 + 100
	mul $4, $2, 100	$4← $2 * 100 (32-bit product)
Conditional Branch	bgt $4, $2, LABEL	Branch to LABEL if $4 > $2
Unconditional Branch	j LABEL	Always Branch to LABEL

```
sumPos = 0;
sumNeg = 0;
for i = 0 to length-1 do
    if numbers[i] < 0 then
        sumNeg = sumNeg + numbers[i]
    else
        sumPos = sumPos + numbers[i]
    end if
end for
```

1. Write MIPS Assembly Language code for the above algorithm that sums the array's elements.

```
            .data
numbers:    .word 2, 3, -1, 10, -5, -6, 3, 5, 1, -5
length:     .word 10
sumPos:     .word 0
sumNeg:     .word 0

            .text
             .globl main
main:


##### Your code goes here ########


    li $v0, 10          # system call for exiting the program
    syscall
```

General Directions: (This assume you are using PCSpim, but xspim is the labs under Linux too)

1) Log on the Wright lab using your AD-ITS (same account you use for the Library, Union, Dorm, and various hallway computers on campus. If you need your password reset, you must go to ITT 36 for ITS User Support. CNS Support CANNOT reset passwords for student accounts.)
2) Write your assembly language program on paper first! I will not help anyone debug their program without your handwritten program.
3) Type in your program using WordPad and save it on a USB flash memory stick. Remember to use quotes around the file name "hw1.s" (Details below)
4) Debug your MIPS assembly language program.
5) When it is correct, **run it to completion** and copy to the Window's clipboard a snapshot of the PCSpim window by using the <Alt> and <Print Screen> keys together.
6) Open up new Word document and set its page layout to Landscape by File | Page Setup | Paper Size and then select Landscape. (Details below)
7) Paste the snapshot of the PCSpim Debugger window into the Word document. Resize the snapshot to the margins and print a copy to turn in.
8) Print a copy of the assembly language program to turn in too.
9) Hand in a copy of your assembly language program **and** the snapshot of the PCSpim window showing the resulting sorted memory.

Entering the Program using WordPad:

1) Start | Programs | Accessories | WordPad
2) Type in the program in a new file
3) File | Save As
Save in file name: "hw1.s" (USE DOUBLE-QUOTES AROUND FILE NAME)
4) After debugging, print a copy of the program to hand in

Running the Program using PCSpim:

1) Start | Programs | Programming | PCSpim
2) Maximize the window by clicking on the 2nd icon in the upper right of the window
3) Load the program by File | Open
4) Observe the *initial* DATA values in memory (hexadecimal) before the program runs:

```
      DATA
[0x10000000]...[0x10010000]      0x00000000
[0x10010000]                     0x00000002  0x00000003  0xffffffff  0x0000000a
[0x10010010]                     0xfffffffb  0xfffffffa  0x00000003  0x00000005
[0x10010020]                     0x00000001  0xfffffffb  0x0000000a  0x00000000
[0x10010030]...[0x10040000]      0x00000000
```

5) Observe the *initial* register are all 0's before the program runs
6) Run the program by Simulator | Go, then Click "OK" in the Run Parameters window
7) Observe the *resulting* DATA values for "sumPos" and "sumNeg" in memory (hex) after the program runs:

```
      DATA
[0x10000000]...[0x10010000]      0x00000000
[0x10010000]                     0x00000002  0x00000003  0xffffffff  0x0000000a
[0x10010010]                     0xfffffffb  0xfffffffa  0x00000003  0x00000005
[0x10010020]                     0x00000001  0xfffffffb  0x0000000a  0x00000018
[0x10010030]                     0xffffffef  0x00000000  0x00000000  0x00000000
[0x10010040]...[0x10040000]      0x00000000
```

8) Observe the *resulting* register values after the program runs
9) Copy the PCSpim window to the Window's Clipboard using <Alt><Print Screen> keys

Printing the PCSpim window using Word:

1) Start | Programs | Microsoft Office | Microsoft Office Word 2003
2) Type your name at the top of the blank page and hit <enter>
3) Paste in the PCSpim window from the Clipboard by Edit | Paste
4) Change to landscape by File | Page Setup and then click "Lanscape" button
5) Resize the frame containing the PCSpim picture by clicking on the frame drag the lower-right corner to make the picture bigger
6) Print this picture of the PCSpim window to hand in

Downloading PCSpim at Home:

To run PCspim on a PC under Microsoft Windows, download the file http://www.cs.wisc.edu/~larus/SPIM/pcspim.zip, unzip it, and run setup.exe.

To run spim or xspim on a Unix or Linux system, copy either the compressed tar file (http://www.cs.wisc.edu/~larus/SPIM/spim.tar.Z) or the gzip'ed tar file (http://www.cs.wisc.edu/~larus/SPIM/spim.tar.gz). Both files contains source code and must be compiled on a particular system.

Sample program to perform sequential search of an array for a target value. HLL algorithm:

```
int numbers[10] = {25, 5, 40, 50, 30, 8, -1, -5, 20, 9};
int length = 10;
int target = 30;          // in numbers at index 4
int foundIndex = 0;     // gets the index of the target if found or -1 if target is not in the array

i = 0;
while (i < length) and (numbers[i] != target) do
    i = i + 1
end while
if (i == length) then
    i = -1
end if
foundIndex = i
```

Coresponding MIPS assembly language program:

```
              .data
numbers:      .word 25, 5, 40, 50, 30, 8, -1, -5, 20, 9
length:       .word 10
target:       .word 30
foundIndex:   .word 0

               .text
               .globl main
main:

      li    $8, 0             # Register $8 holds i
      la    $9, numbers       # Register $9 holds address of numbers[i] element
      lw    $10, length       # Register $10 holds value of length
      lw    $11, target       # Register $11 holds value of target
while:
      bge   $8, $10, end_while
# and
      lw    $12, 0($9)        # Register $12 holds value of numbers[i] element value
      beq   $12, $11, end_while
      addi  $8, $8, 1         # increment i
      addi  $9, $9, 4         # move $9 to point to next numbers[i] element
                              # (words in MIPS are 4 bytes)
                              # Alternatively we could have calculated the address of
                              # numbers[i] by the (starting addr. of numbers) + i*4, i.e.,
                              # la  $9, numbers
                              # mul $13, $8, 4
                              # add $9, $9, $13
      j     while
end_while:

if:
      bne   $8, $10, end_if
      li    $8, -1
end_if:

      sw    $8, foundIndex

     li $v0, 10           # system call for exiting the program
     syscall
```

2. Compare zero-, one-, two-, three-address, and the load & store machines by writing programs to compute

X = (A + B) * (C - D);
Y = X / (A * D);

for each of the five machines. The instructions available for use are as follows:

3 Address	2 Address	1 Address (Accumulator machine)	0 Address (Stack machine)
MOVE (X ← Y)	MOVE (X ← Y)	LOAD M	PUSH M
		STORE M	POP M
ADD (X ← Y + Z)	ADD (X ← X + Y)	ADD M	ADD
SUB (X ← Y - Z)	SUB (X ← X - Y)	SUB M	SUB
MUL (X ← Y * Z)	MUL (X ← X * Y)	MUL M	MUL
DIV (X ← Y / Z)	DIV (X ← X / Y)	DIV M	DIV
		Notes: "SUB M" performs AC = AC - M "DIV M" performs AC = AC / M	Notes: "SUB" performs POP T POP T2 T3 = T2 - T PUSH T3 "DIV" performs POP T POP T2 T3 = T2 / T PUSH T3

Load/Store Architecture - operands for arithmetic operations must be from/to registers. For example, to perform the high-level statement "Z = X - Y" we need the code:

```
LOAD  R2, X
LOAD  R3, Y
SUB  R4, R2, R3
STORE  R4, Z
```

3. Assume 8-bit opcodes, 32-bit absolute addressing, 5-bit register numbers, and 32-bit operands. Compute the number of bits needed in programs from question 1 by completing the following table.

	3 Address	2 Address	1 Address	0 Address	Load & Store
Number of bits needed to store the program					
Number of bits of data transferred to and from memory					
Total number of bits read and written while the program executes					