Lab Sheet 5 for **CS F342 Computer Architecture**

Semester 1 – 2017-18 (Week 6)

Version 1.0

11th September 2017

Goals for the Lab: We build up on prior labs and explore loops, arrays and shift instructions and attempts sorting etc.

Data Declaration (Recap):

Format for variable name (label) declarations in .data segment:

name-or-label: storage_type value(s)

- create storage for variable of specified type with given name and specified value
- value(s) usually gives initial value(s).
- for storage type ".space", gives number of bytes to be allocated.
- Note: Name (or labels) always followed by colon (:).
- Some Examples

  var1: .word 3 *#create a single integer variable with initial value 3*

  list: .word 17 5 92 87 41 30 23 55 -72 36 *#an array of 10 integers*

  array1: .byte 'a','b' *# 2- element char array- values of a and b (decimal ascii 97, 98)*

  Array2: .space 40 *# 40 consecutive bytes,* ***not initialized****; could be used as a 40 element char array, or a 10-element integer array; comment should be used to clarify*

Load / Store Instructions

- RAM access only allowed with load and store instructions
- all other instructions use register operands

LOAD_EXAMPLES:

format:

```
lw register_dest, RAM_source #copy word (4 bytes) at source RAM location to destination register.
lb register_dest, RAM_source #copy byte at source RAM location to low order byte of destination register
```

code snippet:

```
lwc1 $f0, 4($t4) # $f0 = Mem[ $t4 + 4 ] : Loads word(4-bytes) into coprocessor (CP)
ldc1 $f0, 0($t4) # $f0 = Mem[ $t4 + 0 ];  $f1 = Mem[ $t4 + 4 ]; Loads double(8 bytes) in to CP 1
```

STORE_EXAMPLES:
format:

```
sw register_source, RAM_destination #store word in src register into RAM dest.
sb register_source, RAM_destination #store byte (low order)in src reg into RAM dest.
```

code snippet:

```
sw $t2, ($t0) #store word in register $t2 into RAM at address contained in $t0
sw $t2, 12($t0) #store word in register $t2 into RAM at address ($t0 12)
swc1 $f0, 4($t4) # Mem[ $t4 + 4 ] = $f0; Store word(into RAM) from coprocessor 1.
sdc1 $f0, 0($t4) # Mem[ $t4 + 0 ] = $f0; Mem[ $t4 + 4 ] = $f1 ; Store double(into RAM) from CP 1.
```

SHIFT_INSTRUCTION_EXAMPLES
format:

```
sll $rd, $rt, shift_amt #Shift left logical value in $rt by a constant number of bits(shift_amt) and store the result in $rd, same as multiplying the number by 2^(shift_amt)
srl $rd, $rt, shift_amt #Shift right logical the value in $rt by a constant number of bit(shift_amt) and store the result in $rd, same as dividing the number by 2^(shift _amt)
```

Note: Similar to shift we have rotate instructions (`rol` and `ror`).

code snippet:

```
li   $t1, 6
sll  $t2, $t1, 2 #value of $t2 = 24
srl  $t2. $t1. 1 #value of $t2 = 3
```

Exercise A: Explore disassembly for the new instructions

1. 814c0000
2. c08a0000
3. a08a0000
4. e08a0000
5. e48a0000
6. f48a0000
7. 4604103e

Example Program: (Exercise:1)

A program to take a string from user and check whether it is a palindrome or not.

```
.data
        theStr : .space 6 #declare a space of 6 bytes
        isPal : .asciiz "Its is a Palindrome"
        notPal : .asciiz "Not a Palindrome"
        newLine : .asciiz "\n"
.text
main :
        lb $t4, newLine
        li $v0, 8 #8=> read string; $a0 is buffer; $a1 is length
        la $a0, theStr #load the base address of theStr
        li $a1, 6 #load the length of string(max length of string+1 for '\0' )
        syscall
        add $t2,$a0,$zero #load base address in $t2; find input string length

slen_0 : # loop label to find the last char
        lb $t3, ($t2) # load current byte
        addi $t2, $t2, 1 # increment for next iteration
        beq $t3, $t4, next #if current byte is '\n'
        bne $t3,$zero, slen_0 # if current byte isn't '\0', repeat

next :  # label to exit the above loop
        add $t1,$a0,$zero #load base address
        addi $t2,$t2,-2 # -2 because moved beyond '\0' or '\n'; need char before
test_loop :
        bge $t1, $t2, is_palin # if lower pointer >= upper pointer, yes
        lb $t3, 0($t1) # grab the char at lower ptr
        lb $t4, 0($t2) # grab the char at upper ptr
        bne $t3, $t4, not_palin # if different, it's not
        addi $t1, $t1, 1 # advance lower ptr
        addi $t2, $t2, -1 # advance upper ptr
        j test_loop # repeat the loop
is_palin :
        li $v0, 4
        la $a0, isPal
        syscall
        j exit
not_palin :
        li $v0, 4
        la $a0, notPal
        syscall
exit :
        li $v0,10
        syscall
```

Exercise 2:
Write a program to take string of length 5 as input from user and store its reverse string in different array and then print both the strings. Observe the values in data segment by stepping through the code. Do we need to worry about '\0' termination? Why / why not?

```
la $a0, my_arr2
sb $t2,4($a0)  #store byte at $t2 into array cell my_arr2 [4] 5th value
```

Exercise 3:
Write a program to implement C bubblesort program given below in MIPS.

C program code:

```
int main()
{
	int Sz = 10;
	int List[10] = {17, 5, 92, 87,41, 10, 23, 55, 72, 36} ;
	int Stop, // $s3: upper limit for pass
	Curr, // $s0: index of current value in comparison
	Next, // $s1: index of successor to current value
	Temp; // $s2: temp storage for swap
	for (Stop = Sz 1; Stop > 0; Stop)
	{
		for (Curr = 0; Curr < Stop; Curr++)
		{
			Next = Curr + 1;
			if ( List[Curr] > List[Next] )
			{
				Temp = List[Curr];
				List[Curr] = List[Next];
				List[Next] = Temp;
			}
		}
	}
	printf("Sorted list in ascending order:\n");
	for (Curr = 0; Curr < Stop; Curr++)
	printf("%d\n", List[Curr]);
}
```

Hint *::* To convert Curr to offset you can use sll $t4, $t2, 2 or similar where $t2 is Curr, $t4 is offset from starting address of buffer abd shift of 2 implies multiplying by 4.

Partial assembly code: (Highlighted part is complete)

```
.data
list: .word 17, 5, 92, 87,41, 10, 23, 55, 72, 36
space: .asciiz " "
.text
main:

li $s7,10 #size of the list(sz)
addi $s3,$s7,-1    # $s3 = Stop = sz-1

#Write the loop, swap code here

exit:      #print the array

la $t0,list
li $t2,0   #as a counter while printing the list

print:

lw $a0,($t0)        #load current word in $a0

li $v0,1
syscall             #print the current word
la $a0,space
li $v0,4
syscall             #print space in b/w words
addi $t0,$t0,4      #point to next word
addi $t2,$t2,1      #counter++
blt $t2,$s7,print


li $v0,10
syscall
```

Exercise 4: TBD (take home)
Write a program to implement above program but store floating point numbers instead of integer.

Hint: Use commands swc1, lwc1, c.le.s, bc1f, bc1t
Comparison of FP values sets a code in a special register and Branch instructions jump depending on the value of the code:

```
c.le.s $f2, $f4 #if $f2 <= $f4 then code = 1 else code = 0

bc1f  label #if code == 0 then jump to label

bc1t label # if code == 1 then jump to label
```

References: Green Sheet and text book appendix.

http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm
http://tfinley.net/notes/cps104/mips.html
https://www.doc.ic.ac.uk/lab/secondyear/spim/node20.html
https://people.cs.pitt.edu/~childers/CS0447/lectures/SlidesLab92Up.pdf
http://www.math.unipd.it/~sperduti/ARCHITETTURE1/mips32.pdf