Lab Sheet 7 for **CS F342 Computer Architecture**

Semester 1 – 2017-18

Version 1.0

20th October 2017

Goals for the Lab: Build up on prior labs to further explore functions and also

1. Understand mapping of structures
2. Memory allocation using system calls (syscall 9)
3. Input and output characters (syscall 11, 12)

Background: We will be exploring system call 9 (sbrk) for allocating memory. We will also explore when to use temporary registers and when to save register values etc., using examples that may involve more than one return points from the function.

Exercise 1: Study the given code for finding factorial of an integer recursively (Also the solution to the Exercise 3 of the previous lab sheet)

```
Input: Single integer
4
Output: Single integer
24
```

Pseudo Code:

```
int factorial(int input)
{
   int output = input;
   if(input > 1)
      output = input *factorial(input-1);
   return output;
}

main()
{
   printf( "Enter a number to find factorial:");
   scanf("%d", &i);
   j = factorial(i);
   printf("The result of factorial for %d is %d\n", i, j);
   exit(0);
}
```

```
.data
        promptMessage: .asciiz "Enter a number to find it's factorial:"
        resultMessage: .ascii "\nThe factorial of the given number is:"

.text
    main:
        li $v0, 4
        la $a0, promptMessage
        syscall
        li $v0,5 # get the number from user
        syscall
        move  $a0, $v0
        jal findFactorial #call findFactorial function

        move $s0,$v0
        li $v0, 4
        la $a0, resultMessage
        syscall
        li $v0, 1 #display the result
        move $a0, $s0
        syscall
        li $v0,10 # exit from main
        syscall


    findFactorial:

        subu $sp,$sp,8 #adjust stack pointer
        sw $ra,0($sp)
        sw $s0,4($sp) # since the register s0 will be modified during recursion
                      # a0 is not saved, since its value is not used after return
        li $v0,1      # v0 is not saved, since its value is reset before return
        beq $a0,0,factDone #the base case (input = 0) – return 1

        move $s0,$a0 #find findfactorial(n-1)
        sub $a0,$a0,1
        jal findFactorial
        mul $v0,$s0,$v0


    factDone:

        lw $ra,0($sp)
        lw $s0,4($sp)
        addu $sp,$sp,8
        jr $ra
```

Take home assignment

Write a recursive MIPS assembly program to print the nth number of Fibonacci sequence

```
Input : Single Integer
6
Output : Single Integer
8
```

Pseudo Code :

```
int fib(int n)
{
if (n == 0)
return 0;
else if (n == 1)
return 1;
return fib(n - 1) + fib(n – 2);
}
void main()
{
int n;
printf("Please enter a non negative integer :");
scanf("%d",&n);
ans=fib(n);
printf("The %dth fibonacii number is %d.",n,ans);
exit(0);
}
```

<u>New concept:</u> To dynamically allocate memory in MIPS use syscall named **sbrk.**

Interestingly, sbrk behaves much more like its namesake (the UNIX sbrk system call) than like malloc– it extends the data segment by the number of bytes requested, and then returns the location of the previous end of the data segment (which is the start of the freshly allocated memory). The problem with sbrk is that it can only be used to allocate memory, never to give it back (release / free).
In this course we may use the term allocate, but keep in mind that its actual implementation is not same as alloc / malloc.

- To represent structures in MIPS

```
typedef struct node{
   int val;  //value of this node
   struct node * left;  //pointer to left child
   struct node* right;   //pointer to right child
 } nodeType;
```

MIPS assembly	C equivalent

Suppose $t1 points to 12 bytes of free memory
li $a0,12 *//bytes to be allocated*
li $v0,9
syscall *//now $v0 holds the address of first byte of 12 bytes of free memory*
sw $s0, 0($v0)
sw $s1, 4($v0)
sw $s2, 8($v0)
 	
lw $s0, 0($v0)
lw $s1, 4($v0)
lw $s2, 8($v0)
	a,b,c,ptr are analogous to values of $s0,$s1,$s2,$v0 respectively.
	
	node* ptr = (node*)malloc(sizeof(node));
	
	# ptr->val = a;
	# ptr->left = b;
	# ptr->right = c;
	
	# a = ptr->val;
	# b = ptr->left;
	# c = ptr->right;

Exercise 2: Complete the code given below to

1. Build an ordered binary tree T containing all the values to be sorted(Integer values)
2. Do an inorder traversal of T, printing out the values of each node.

```
.data
  space: .asciiz " "
 .text
 main:
  li $s0,0      #$s0 always points to the root node of binary tree, initially NULL
  get_input: #infinite loop for getting number to be inserted in tree, 0 terminates the loop

  li $v0,5
  syscall

  beq $v0,$zero,break_of_loop

  move $a0,$v0     #$a0 = number to be inserted
  move $a1,$s0     #$a1 = ptr to the root (holds address of root node), initially NULL
  jal insert_in_tree
  move $s0,$v0    #v0 adress to the root, storing it in $s0
  j get_input

break_of_loop:    #exit from loop above for entry value as 0
  move $a0,$s0
  jal inorder_traversal      #inorder traversal, $a0 is argument, holds address of the root

  li $v0,10 #exit from main
  syscall
```

```
insert_in_tree:
  bne $a1,$zero,not_base_case#check for base case if(pointer == NULL),
                             #    then dynamically allocate memory for the new node)
  move $t0,$a0                # for base case, you can avoid stack pointer and use
                             #    temporary registers for restoring purposes
  #In next few lines, do dynamic memory allocation, $a0 = size(in bytes), $v0 = pointer to
the new memory, insert the input value into the created structure, and set left and right
child pointers as NULL ($zero), restore value of $a0,then return.


  jr $ra                     #    return only for base case – not for rest

not_base_case:
#Taking hint from returnNonBase label, store the required values in stack
```

```
#compare number in current node with the number to be inserted, accordingly traverse left or right
left:
   addi $a1,$a1,4 #left node pointer at 4($a1) : after this it's at 0($a1)
   move $s0,$a1
   lw   $a1,($a1)  # now we have the pointer value – may be null
   jal insert_in_tree #return value of this function is in $v0, which holds the address of the newly created child (left or right acc. to value of $s0)
   j returnNonBase

right:
#traverse right, change value of $a1 accordingly
```

```
 returnNonBase:
   sw $v0,($s0) #NOTE – null value is being updated with the address returned
   lw $ra,($sp)
   lw $a1,4($sp)
   lw $s0,8($sp)

   addi $sp,$sp,12
   move $v0,$a1#NOTE
   jr $ra
```

#########END OF INSERT IN TREE FUNCTION#####

```
inorder_traversal: #a0 is argument, holds address of the root initially
   beq $a0,$zero,return_inorder

   addi $sp,$sp,-8
   sw $ra,($sp)
   sw $a0,4($sp)
    #traverse left, then print the middle element, then traverse right
```

```
 #restore the register values
   lw $a0,4($sp)
   lw $ra,($sp)
   addi $sp,$sp,8
   return_inorder:

   jr $ra
```

#########END OF INORDER TRAVERSAL FUNCTION#####

Exercise 3: Modify the above code to incorporate characters instead of integer values.

Hint:

- Conditions for branch instructions will change
- Size of the structure will change
- lw, sw will change to lb, sb
- refer syscall 11,12 for printing and reading chars

References:

http://stackoverflow.com/questions/22976456/mips-recursive-fibonacci-sequence

https://www.cs.ucsb.edu/~franklin/64/lectures/mipsassemblytutorial.pdf