CS F342
Computer Architecture

Semester 1 – 2022 – 23

Lab Sheet 8 & 9

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 4 of the previous lab sheet 6)

```
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);
}
```

MIPS Code:

```
.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.**

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
After Syscall $v0 points to 12 bytes of free memory (newly allocated) 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*	a,b,c,ptr are analogous to values of $s0,$s1,$s2,$v0 respectively. node* ptr = (node*)malloc(sizeof(node));

sw $s0, 0($v0)
sw $s1, 4($v0)
sw $s2, 8($v0)
	# ptr->val = a; // $s0 has the value
	# ptr->left = b; // $s1 has left pointer
	# ptr->right = c;// $s2 has right pointer

lw $s0, 0($v0)
lw $s1, 4($v0)
lw $s2, 8($v0)
	# a = ptr->val; # b = ptr->left;
	# c = ptr->right;

<u>Excersise 1:</u> Write a MIPS code to dynamically create an array of size N, and then find the sum of the array elements

<u>Excersise 2:</u> Write a MIPS code to create Structure to store Name, Roll no, CGPA of Students and display the details of students on console

<u>Exercise 3</u>: 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
root: .word 0 0 0                   # predefining root node as NULL
input: .asciiz "Enter numbers to insert into binary tree (0 to stop): \n"
output: .asciiz "Inorder traversal of the binary search tree: "
```

```
.text
main:
    la $a0, input
    li $v0, 4
    syscall
    li $v0, 5
    syscall                          # enter first number
    beqz $v0, end_of_loop1
    la $a0, root                     # load root address into $a0 for inserting values into BST
    sw $v0, ($a0)
    loop1:
        li $v0, 5                    # enter subsequent numbers
        syscall
        beqz $v0, end_of_loop1       # jump out of current loop if 0 is entered
        jal insert                   # call subroutine to insert into BST
        j loop1
    end_of_loop1:
        la $a0, output
        li $v0, 4
        syscall
        la $a0, root                 # load address of root node for inorder function, $a0 will always contain address of tree to call inorder traversal
        jal inorder                  # call inorder function
        li $v0, 10
        syscall

insert:
    move $t0, $v0
    li $a0, 12
    li $v0, 9
    syscall                          # allocate space for 12/4 = 3 integers (one for value, one for left pointer address, one for right pointer address)
    sw $t0, ($v0)                    # store input value into newly created node
    sw $0, 4($v0)                    # set left pointer of node to NULL
    sw $0, 8($v0)                    # set right pointer of node to NULL
    la $a0, root
    # write code to insert newly created node into the BST

    jr $ra

inorder:
    beqz $a0, end_of_inorder         # check if NULL node or not
    # write code to push values onto stack

    # write code to restore values from stack

    # write code to print integer

    # write code to push values onto stack

    # write code to restore values from stack

    end_of_inorder:
        jr $ra
```

<u>Exercise 4:</u> Modify the above code to incorporate characters instead of integer values.

<u>Hint:</u>

- 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