**INDIAN INSTITUTE OF TECHNOLOGY**

**GOA**

**COMPUTER ARCHITECTURE LAB (CS 211)**

Faculty : Dr. Sharad Sinha

Teaching Assistant: Prachi Kashikar and Pavitra Bhade

**LAB 04**

In this course, you will study the working of stacks and subroutines in MIPS32 processor. You will learn about the various special purpose registers used for this purpose. You will be writing assembly codes in QtSpim, which will illustrate this concept.

This lab exercise will be split into two parts. In the first part you will be introduced to the concept of stacks and subroutines. In the second part, you will be writing assembly codes to understand the same.

**Part A:**

**Stacks:**

A stack is a way of organising data in the memory. To locate elements in stack, a special register, by the name ‘stack pointer’ is used. This stack pointer points to the topmost element in the stack i.e the stack pointer holds the address of the topmost element (most recently added element). Stack works by the technique of ‘Last in First out’, which means, the element that is added at last, is the first one to be moved of the stack. In Mips, each location of the stack can hold 32 bit word. The stack grows downwards as shown in Fig 1. This means it grows from higher address to lower address. Hence, whenever a new data element is placed in the stack, the stack pointer is decremented by 4 and then the element is pushed on to the stack. Same way when an element has to be popped from the stack, the element has to be copied to some other location first and then the stack pointer has to be incremented by 4. This will ensure that logically that particular element does no longer exist in the stack.

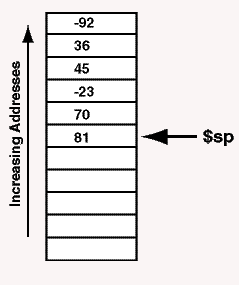


Fig 1.

**Subroutines:**

A subroutine is a logical division of the code that may be regarded as a self-contained operation. The j (jump) or the b (branch) instruction is used to unconditionally shift the control of the program to the separate location of the respective subroutine. When the main routine needs to start-up a subroutine, it can shift control to that subroutine using any of the jump instructions. At the end of the subroutine, the control must be returned using another jump instruction. Now this instruction can include a location pointed by a label. But if this subroutine is called multiple times from different locations, the return statement in the subroutine will return the control to a constant location which will lead to repetition. Hence, when the subroutine is called multiple times, the return location will change depending on the calling location. A special register named ‘ra’ stores the return address. This will ensure that the control is returned to the respective caller. The following instruction placed at the end of the subroutine:

jr $ra

A subroutine that might call another subroutine must push the return address present in $ra, onto the stack. When it returns to its caller, it pops the stack to get the return address. The subroutines can also be global. This means they can be accessed by some other file. When the called and calling files have been loaded in the simulator, you can see the two files linked together in memory, in the text and data section.

**Part B:**

Write the following MIPS assembly language programs and run them using the single step method. This will help you understand the stack push and pop operations and also the flow of program in case of the subroutines.

1. Evaluate the expression ‘ab-10a+20b+16’. Consider that only $t0 and $t1 are available to store temporary values. Store a=10 and b=20 in data section. Use stack for other memory requirements. Display the sum.
2. Find the maximum of the three expressions: x\*x ; x\*y ; y\*5. Take x and y as input from user. Write a global subroutine, in another file, to calculate values of these expressions. Write a subroutine to find maximum of two integers and use it to find the maximum of these three expressions. Display the result.