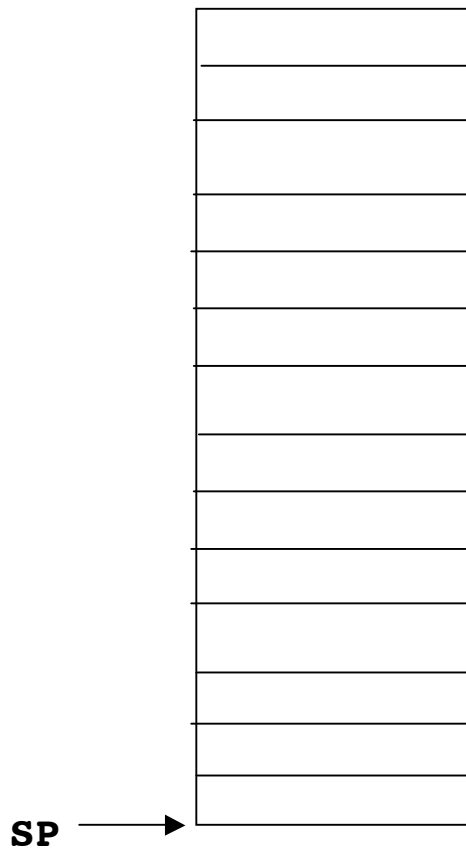


## Laboratory 11 Stacks and Procedures

*Computer Science 240*

The stack is just a chunk of memory used for temporary storage of values.

In MIPS, the bottom of this chunk of memory is pointed to by a register called the stack pointer ( $\$sp$ ), and is a large address. As you move up the stack, the addresses get lower. In this model, each row in the stack is word-size (4 bytes):



The stack can be used as a first-in-first-out queue, to store data.

The stack grows 'up'; pushing a value on the stack means decrementing SP by the size of the value, and then writing to the location pointed to by the new value of the SP.

Popping a value from the stack means reading the value pointed to by the SP, and then incrementing the SP.

Typically, the stack is used for storing parameters and/or saving registers during procedure calls.

The instructions which MIPS uses to call and return from procedures are **jal** (jump-and-link) and **jr** (jump register).

**jal subroutine** ;puts the address of the instruction following the subroutine call in \$ra, and puts the starting address of the subroutine in \$pc

\$ra<-pc + 4; pc <- address\_of\_subroutine

**jr \$ra** ;puts the return address from \$ra into the PC – always the last statement executed in a procedure!

\$pc <- \$ra (which contains return addr.)

As you have learned in lecture, MIPS provides a number of registers specifically for procedure calls. These include:

**\$a0-\$a3** argument registers to pass parameters

**\$v0-\$v1** value registers to return values

**\$ra** return address register to return to point of origin

If more information or parameters than will fit in these registers is needed to call or return from a subroutine, the stack can be used to pass the extra parameters.

Also, if the procedure modifies registers whose values are needed by the main or invoking program, the stack can be used to save the original values of the registers (which can then be restored before the return to the calling program).

The convention is that if registers \$s0 - \$s7 are modified by a procedure, the procedure should save the original value of the registers on the stack, and restore them before the return from the procedure.

On the next page is an example of the procedure call and return, and saving/restoring all 8 \$s registers on the stack:

```

# main program
main:    la    $a0,param0 #store parameters in $a0 - $a3
        la    $a1,param1
        la    $a2,param2
        la    $a3,param3
        jal  myproc     #call procedure
        move $t0,$v0    #get returned values from $v0,$v1
        move $t1,$v1

        . . .
        #rest of the main program instructions
        . . .

        li $v0,10      #exit main program
        syscall

#procedure definition
myproc:  addi $sp,$sp,-32 #allocate 8 words on stack
        sw   $s0,0($sp)  #store 8 registers on stack
        sw   $s1,4($sp)
        sw   $s2,8($sp)
        sw   $s3,12($sp)
        sw   $s4,16($sp)
        sw   $s5,20($sp)
        sw   $s6,24($sp)
        sw   $s7,28($sp)

        . . .
        #instructions which use parameters and registers
        #for some #purpose
        . . .

        lw   $v0,($a0) #put return values in $v0 and $v1
        lw   $v1,($a1)

        lw   $s7,28($sp) #restore 8 registers from stack
        lw   $s6,24($sp)
        lw   $s5,20($sp)
        lw   $s4,16($sp)
        lw   $s3,12($sp)
        lw   $s2,8($sp)
        lw   $s1,4($sp)
        lw   $s0,0($sp)
        addi $sp,$sp,32  #deallocate the stack

        jr   $ra        #return from procedure to main
                        #program

```

For recursive and nested procedures, there may be conflicts over the use of the shared registers, so the convention is more restrictive.

The calling program should save \$a0 - \$a3 and \$t0 - \$t9 if their values need to be preserved across a procedure call. The procedure should save and restore \$ra and any of the \$s0 - \$s7 registers modified by the procedure.

The example on the next page shows how to save and restore the registers for a program with nested procedures:

```

main:    lw    $t0,old_result
        lw    $a2,op1
        lw    $a3,op2

        addi $sp,-12        #store $t0,$a2,$a3 on stack
        sw    $t0,0($sp)
        sw    $a2,4($sp)
        sw    $a3,8($sp)

        jal   proc1        #call procedure

        lw    $a3,8($sp)    #restore registers from stack
        lw    $a2,4($sp)
        lw    $t0,0($sp)

        beq   $t0,$v0,done  #use return value from $v0

        . . .

done:    li    $v0,10        #exit main program
        syscall

# procedure 1 definition
proc1:   addi $sp,$sp,-8    #allocate 2 words on stack
        sw    $s0,0($sp)    #store $s0 and $ra on stack
        sw    $ra,4($sp)

        #assume some use of $s0 and $t0 which modifies
        #their values

        move  $a2,$s0
        move  $a3,$t0
        jal   proc2

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

#procedure 2 definition
proc2:   addi $sp,$sp,-4
        sw    $s0,0($sp)

        #assume some use of $s0, and set $v0=return value

        lw    $s0,0($sp)
        addi $sp,$sp,4
        jr    $ra

```