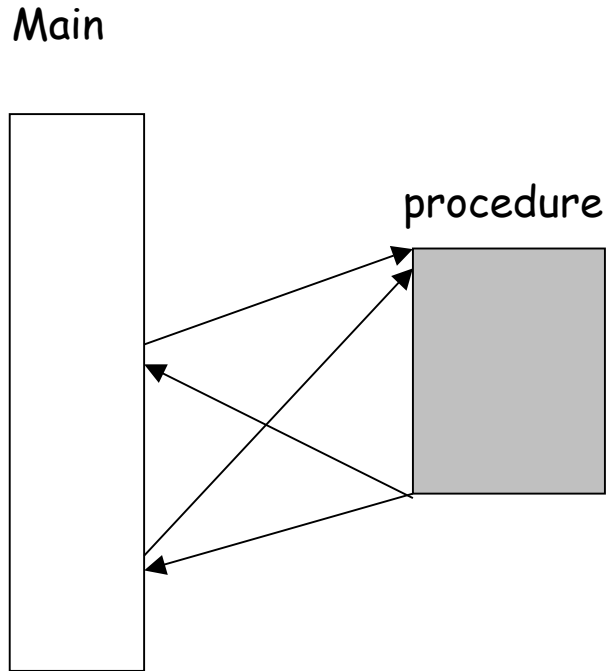


Procedure Call

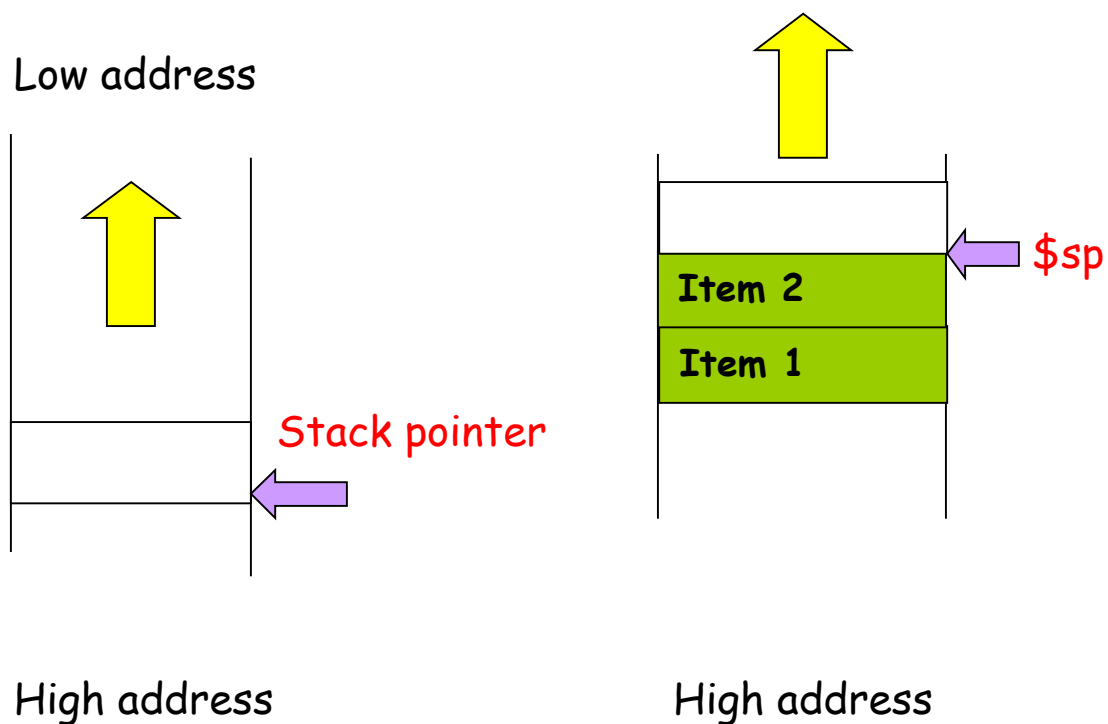


Typically procedure call uses a **stack**. What is a stack?

Question. Can't we use a **jump** instruction to implement a procedure call?

The stack

Occupies **a part of the main memory**. In MIPS, it grows from high address to low address as you push data on the stack. Consequently, the content of the stack pointer ($\$sp$) decreases.



Use of the stack in procedure call

Before the subroutine executes, save registers (why?).

Jump to the subroutine using **jump-and-link** (jal address)

(jal address means **ra** \leftarrow **PC+4**; **PC** \leftarrow **address**) For

MIPS, (ra=r31)

After the subroutine executes, restore the registers.

Return from the subroutine using **jr** (jump register)

(jr ra means **PC** \leftarrow **(ra)**)

Example of a function call

```
int leaf (int g, int h, int i, int j)
```

```
{
```

```
    int f;
```

```
    f = (g + h) - (i + j);
```

```
    return f;
```

```
}
```

The arguments g, h, i, j are put in **\$a0-\$a3**.

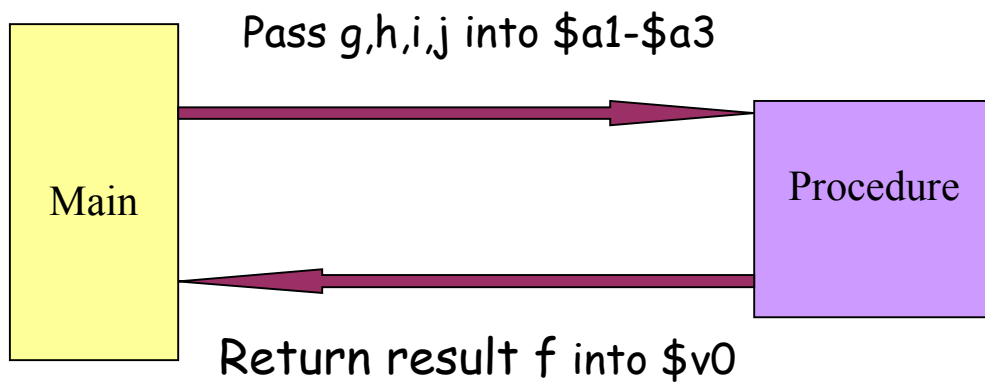
The result f will be put into **\$s0**, and returned to **\$v0**.

The structure of the procedure

```
Leaf:   subi $sp, $sp, 12   # $sp = $sp-12, make room
        sw $t1, 8($sp)     # save $t1 on stack
        sw $t0, 4($sp)     # save $t0 on stack
        sw $s0, 0($sp)     # save $s0 on stack
```

The contents of $\$t1$, $\$t0$, $\$s0$ in the main program will not be overwritten. Now we can use them in the body of the function.

```
add $t0, $a0, $a1   # $t0 = g + h
add $t1, $a2, $a3   # $t1 = i + j
sub $s0, $t0, $t1   # $s0 = (g + h) - (i + j)
```



Return the result into the register \$v0

```
add $v0, $s0, $zero    # returns f = (g+h)-(i+j) to $v0
```

Now restore the old values of the registers by popping the stack.

```
lw $s0, 0($sp)        # restore $s0
```

```
lw $t0, 4($sp)        # restore $t0
```

```
lw $t1, 8($sp)        # restore $t1
```

```
addi $sp, $sp, 12     # adjust $sp
```

Finally, return to the main program.

```
jr $ra                # return to caller.
```

