http://chortle.ccsu.edu/AssemblyTutorial/TutorialContents.html

Stack-like behavior is sometimes called "LIFO" for Last In First Out.

Push

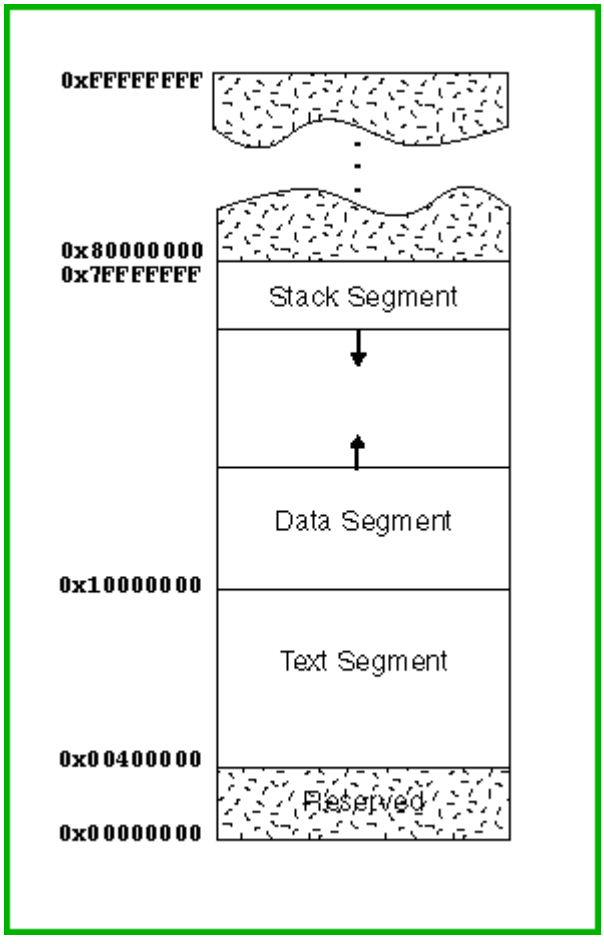
To push **an item onto the stack, first** subtract **4 from the stack pointer, then store the item at the address in the stack pointer.**

PUSH the item in $t0:

```
subu $sp,$sp,4       #   point to the place for the new item,
sw   $t0,($sp)       #   store the contents of $t0 as the new top.
```

Pop

```
# POP the item into $t0:
lw   $t0,($sp)       #   Copy top the item to $t0.
addu $sp,$sp,4       #   Point to the item beneath the old top.
```



The `jal` instruction does the following in the execute phase of the machine cycle:

```
jal sub     # $ra <- PC+4  $ra <- address 8 bytes away from the jal
            # PC  <- sub   load the PC with the subroutine entry point
```

By agreement between programmers (not by hardware) registers have been assigned different roles with subroutine linkage:

- **`$t0 - $t9` — The subroutine is free to change these registers.**
- **`$s0 - $s7` — The subroutine must not change these registers.**
- **`$a0 - $a3` — These registers contain arguments for the subroutine. The subroutine can change them.**
- **`$v0 - $v1` — These registers contain values returned from the subroutine.**

Here is an example program fragment. Subroutine `subB` calls `subC` which uses two "S" registers.

```
subB:
          sub     $sp,$sp,4     # push $ra
          sw      $ra,($sp)
          . . . .
          jal     subC          # call subC
          nop
          . . . .
          lw      $ra,($sp)     # pop return address
          add     $sp,$sp,4
          jr      $ra           # return to caller
          nop

# subC expects to use $s0 and $s1
# subC does not call another subroutine
#
subC:
          sub     $sp,$sp,4     # push $s0
          sw      $s0,($sp)
          sub     $sp,$sp,4     # push $s1
          sw      $s1,($sp)
          . . . .               # statements using $s0 and $s1
          lw      $___,($sp)    # pop _____
          add     $sp,$sp,4
          lw      $___,($sp)    # pop _____
          add     $sp,$sp,4

          jr      $ra           # return to caller
          nop

# subC expects to use $s0 and $s1
# subC does not call another subroutine
#
subC:
          sub     $sp,$sp,4     # push $s0
          sw      $s0,($sp)
          sub     $sp,$sp,4     # push $s1
          sw      $s1,($sp)
          . . . .               # statements using $s0 and $s1
          lw      $s1,($sp)     # pop s1
          add     $sp,$sp,4
          lw      $s0,($sp)     # pop s0
```

```
        add     $sp,$sp,4

        jr      $ra          # return to subB
        nop
```

!!!!!!! Step by step subroutine examples study !!!!!

―――――――――――――――――――――――――――――

Kosuke's First Example

―――――――――――――――――――――――――――――

```
# subroutine (procedure) call

.text
.globl main
main:
     jal  sbrtn  # ret addrs x is stored in $ra

x:   j x         # forever loop

sbrtn:
    jr   $ra    # jump using $ra ($ra content is address x)
```

―――――――――――――――――――――――――――――

Kosuke's Second Example

―――――――――――――――――――――――――――――

```
# subroutine (procedure) call
# single parameter passing

.text
.globl main

main:
     addi $t0, $0, 5     # Call to sbrtn(5);
     addi $sp, $sp, -4   # allocate a word on stack
     sw   $t0, 0($sp)    # push 5 on stack
     jal  sbrtn          # call x
     addi $sp, $sp, 4    # restore $sp

x:   j x         # forever loop


sbrtn:
    lw   $t0, 0($sp)     # retrieve parameter from stack
```

```
    jr   $ra             # jump using $ra
```

Kosuke's Third Example

```
# subroutine (procedure) call
# multiple parameter passing
# C convention: right to left parameter passing

.text
.globl main

main:
     addi $t0, $0, 5     # sbrtn(4, 5); // C convention
     addi $sp, $sp, -4   # allocate a word on stack
     sw   $t0, 0($sp)    # push 5 on stack
     addi $t0, $0, 4
     addi $sp, $sp, -4   # allocate a word on stack
     sw   $t0, 0($sp)    # push 4 on stack
     jal  sbrtn          # call x
     addi $sp, $sp, 8    # restore $sp. It is 8 bytes!

x:   j x         # forever loop


sbrtn:
    lw   $a0, 0($sp)     # retrieve parameter from stack
    lw   $a1, 4($sp)     # retrieve parameter from stack
    jr   $ra
```

Kosuke's Forth Example

```
# subroutine (procedure) call
# storing ra to call another subroutine

.text
.globl main

main:
     jal  sbrtn  # call x, $ra contains location x below.
x:   j x         # forever loop


sbrtn:
```

```
    addi $sp, $sp, -4

    sw $ra, 0($sp) # save the current return address

    jal sbrtn1;    # this call overwrites $ra
                   # $ra contains now this very location
    lw $ra, 0($sp) # restore caller's return address to $ra
    addi $sp, $sp, 4
    jr   $ra

sbrtn1:
    jr $ra
```

Kosuke's Fifth Example

```
# subroutine (procedure) call
# saving registers and creating local variables in
# subroutine

.text
.globl main

main:
     addi $t0, $0, 5
     addi $sp, $sp, -4   # allocate a word on stack
     sw   $t0, 0($sp)    # push 5 on stack
     addi $t0, $0, 4
     addi $sp, $sp, -4   # allocate a word on stack
     sw   $t0, 0($sp)    # push 4 on stack

     jal  sbrtn  # call x, $ra contains location x below.

     addi $sp, $sp, 8    # restore $sp. It is 8 bytes!
x:   j x         # forever loop


sbrtn:
    addi $sp, $sp, -4 # 0($sp) was parameter 1's location
                      # but, now 0($sp) 1s parameter 1's
                      # location after sp=sp-4. @@@@@@

    sw   $fp, 0($sp)  # save current frame pointer on stack
    add  $fp, $0, $sp # fp=sp, $sp is freed. And,
                      # 4(fp) is parameter 1's location
                  # sp at call time is $fp+4, (we did sp-4)
```

```
# Now how the stack frame looks like
#
#   ...
#  fp+8: param2
#  fp+4: param1
#  fp+0: old fp

# Now, we do sp=sp-16 to create 4 words slots on stack

#  fp-4: $a0   We will save $a0 on stack
#  fp-8: $a1   We willsave $a1 on stack
#  fp-12: local word0
#  fp-16: local word1

# use fp to reference local variables and parameters

   addi $sp, $sp, -16
   sw   $a0, -4($fp)   # preserve a0
   sw   $a1, -8($fp)   # preserve a1
   lw   $a0, 4($fp)    # retrieve parameter1
   lw   $a1, 8($fp)    # retrieve parameter2
   sw   $a0, -12($fp)  # store parameter1 to word0
   sw   $a1, -16($fp)  # store parameter2 to word1

  # bottom part of code to return

    add  $sp, $0, $fp   # restore sp saved in fp
    lw   $fp, 0($sp)    # restpre fp
    addi $sp, $sp, 4    # restore sp to at the entry point
                        # this is for @@@@@@ above
    jr   $ra
```