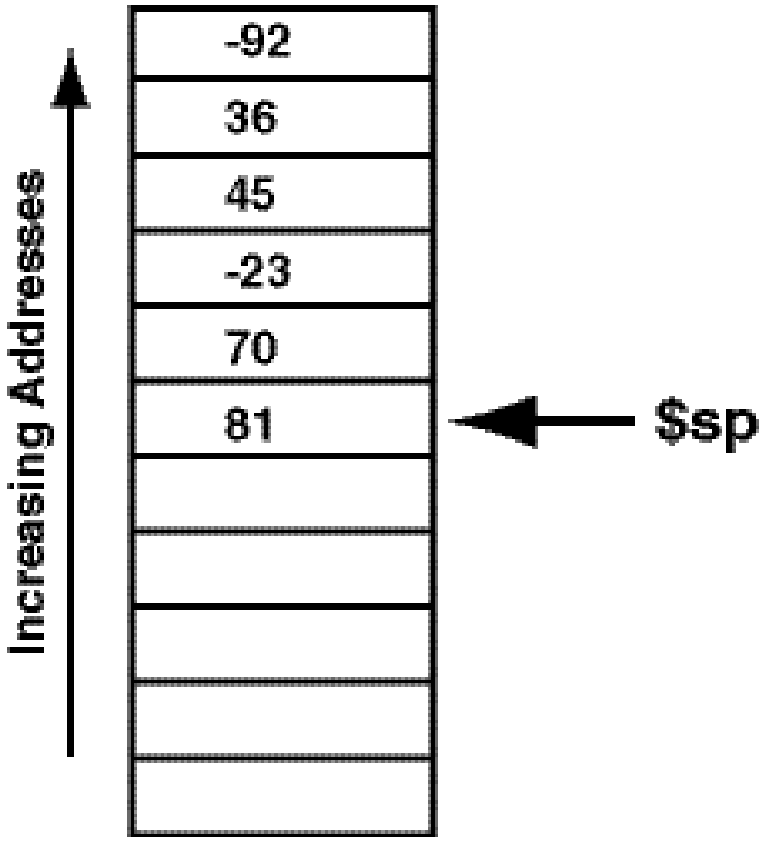
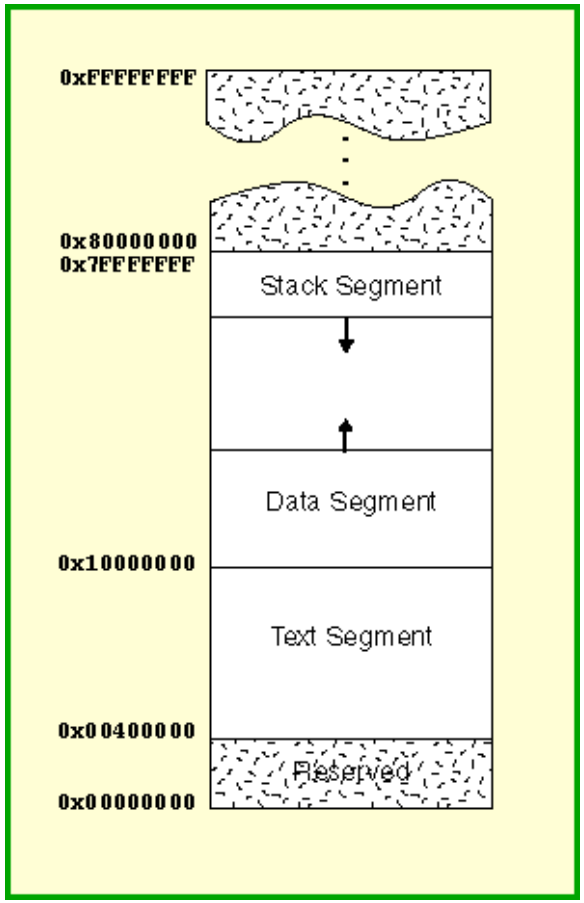
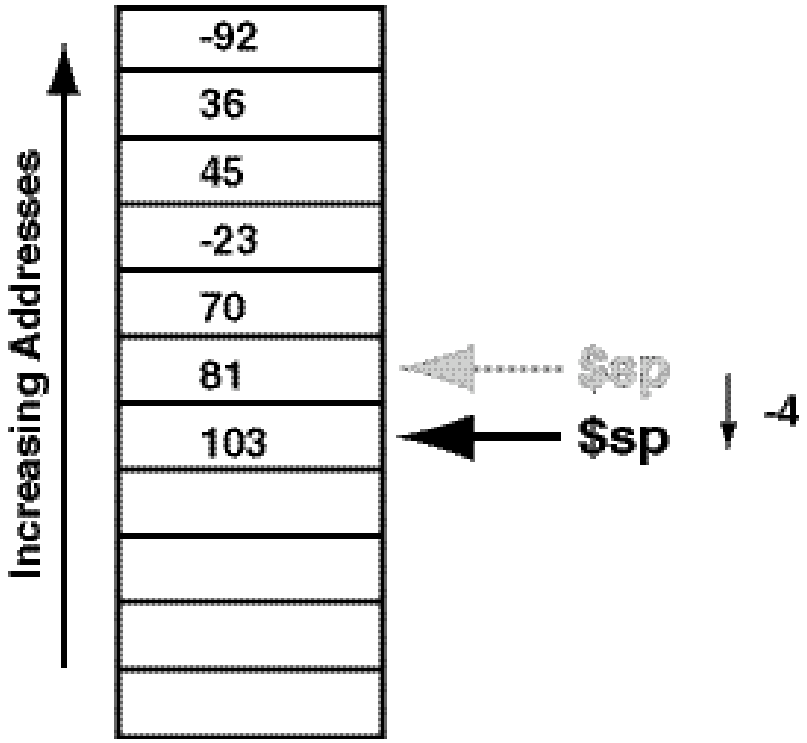
MIPS Stack

- Upside down Stack of words (LIFO)
- $sp($29): Stack Pointer register (top item)
- Initially $sp = 0x7FFFEFFC



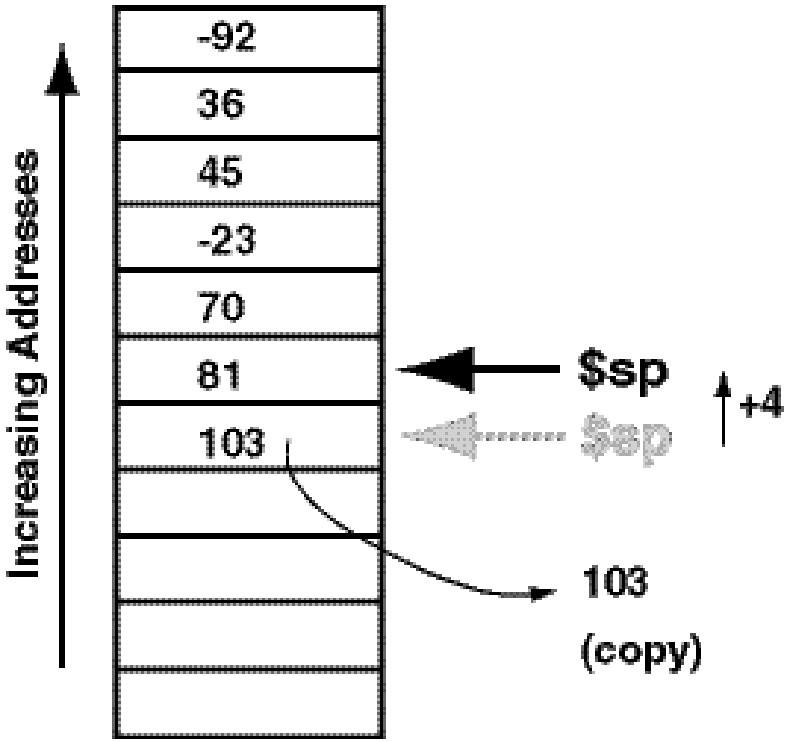


Push Operation



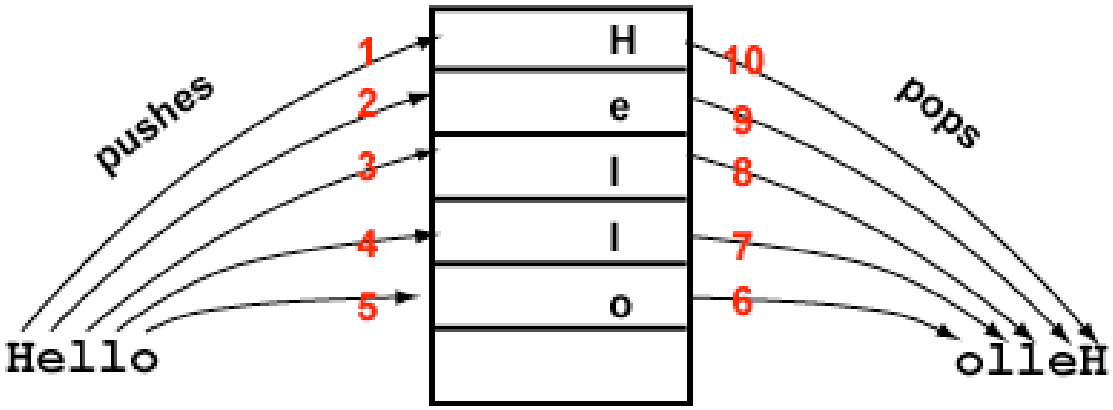

```
# PUSH the item in $t0
subu $sp,$sp,4 # point to the new place
sw   $t0,($sp) # store $t0 at the new top
```

Pop Operation




```
# POP the item into $t0
lw   $t0,($sp) # copy the top item into $t0
addu $sp,$sp,4 # point to the new place
```

Example : Reverse String



- Program Outline

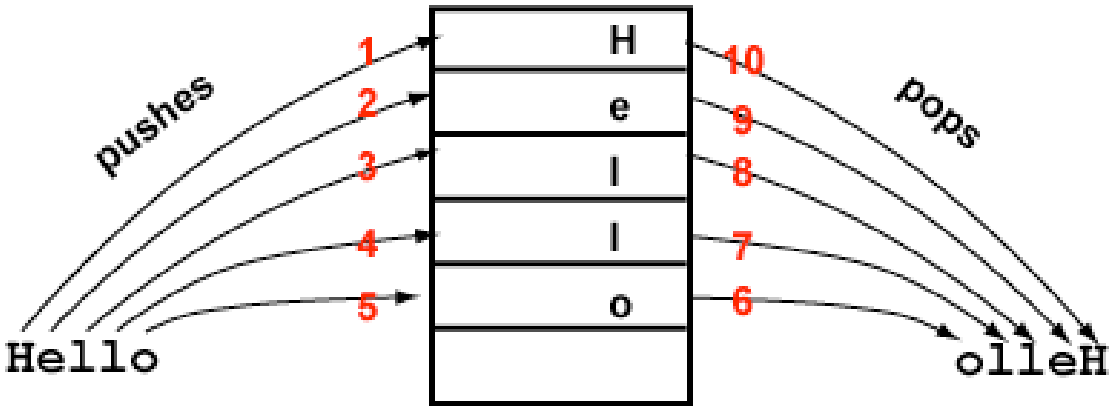
  Print the reversed string

  Push each character onto the stack

  Input the string into a buffer

  Pop chars from stack back into the buffer

Example : Reverse String



- Program Outline

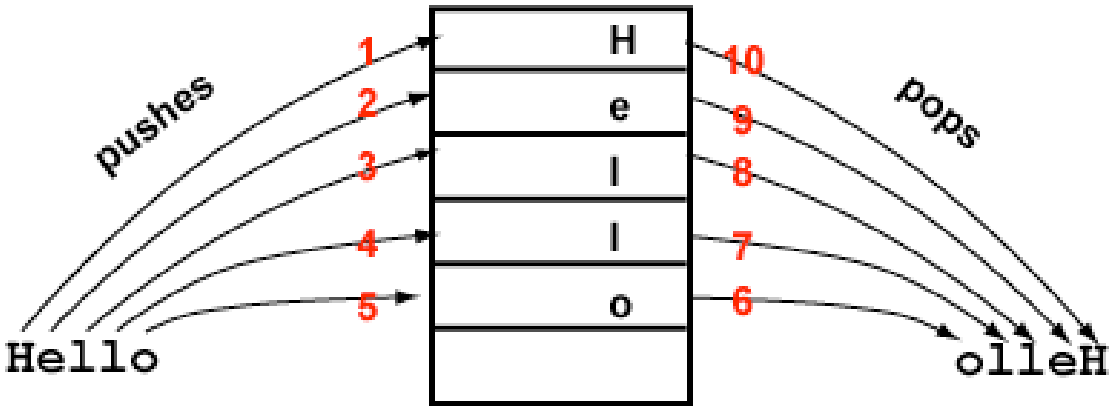
  Print the reversed string

  Push each character onto the stack

  Input the string into a buffer ----------(1)

  Pop chars from stack back into the buffer

Example : Reverse String



- Program Outline

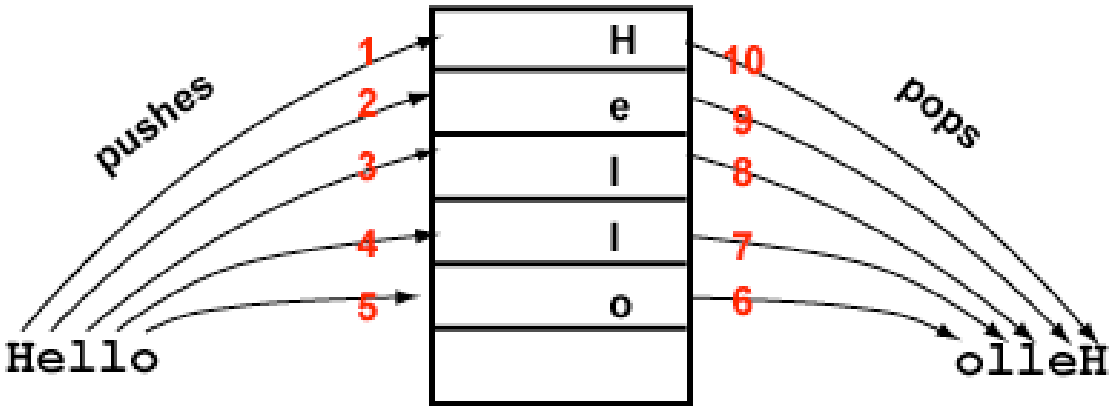
  Print the reversed string

  Push each character onto the stack ----------(2)

  Input the string into a buffer ----------(1)

  Pop chars from stack back into the buffer

Example : Reverse String



- Program Outline

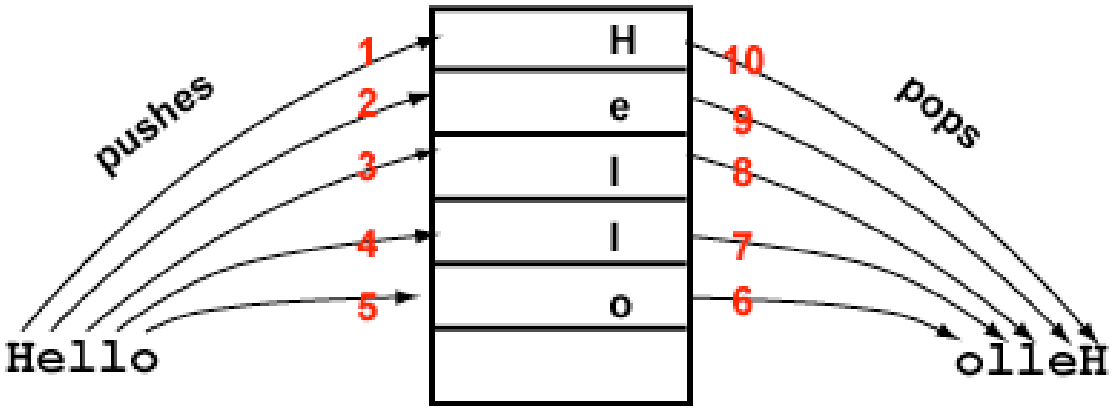
  Print the reversed string

  Push each character onto the stack ----------(2)

  Input the string into a buffer ----------(1)

  Pop chars from stack back into the buffer --------(3)

Example : Reverse String



- Program Outline

  Print the reversed string ----------(4)

  Push each character onto the stack ----------(2)

  Input the string into a buffer ----------(1)

  Pop chars from stack back into the buffer -------(3)

First section : Input

```
.data
str:   .space 128 # character buffer
.text
.globl main
main:
    #input the string
    li $v0,8        # service code
    la $a0,str      # address of buffer
    li $a1,128      # buffer length
    syscall
    li $t0,0        # push a null
    subu $sp,$sp,4  # onto the stack
    sw $t0,($sp)    # to signal its bottom
    li $t1,0        # index of first char in str buffer
```

Second section : Push

```
# push each character onto the stack
pushl:
    lbu     $t0,str($t1)    # get current char into
                            # a full word
    beqz    $t0,stend       # null byte: end of string

    subu    $sp,$sp,4       # push the full word
    sw      $t0,($sp)       # holding the char

    addu    $t1,1           # increment the index
    b       pushl           # loop
stend:
```

Third section : Pop

```
# pop chars from stack back into the buffer
stend:
    li      $t1,0  # index of first byte of str
    buffer
popl:
    lw      $t0,($sp)     # pop a char off the stack
    addu    $sp,$sp,4
    beqz    $t0,done      # null means empty stack

    sb      $t0,str($t1) # store at string[$t1]
    addu    $t1,1         # inc the index
    b       popl          # loop
done:
```

Fourth section : Output

```
# print the reversed string
done:
   li       $v0,4           # service code
   la       $a1,str         # address of string
   syscall

   li       $v0,10          # exit
   syscall
```

Registers Usage Conventions

Register Number	Mnemonic Name	Conventional Use
$0	zero	Permanently 0
$1	$at	Assembler Temporary (reserved)
$2, $3	$v0, $v1	Value returned by a subroutine
$4-$7	$a0-$a3	Arguments to a subroutine
$8-$15	$t0-$t7	Temporary (not preserved across a function call)
$16-$23	$s0-$s7	Saved registers (preserved across a function call)
$24, $25	$t8, $t9	Temporary
$26, $27	$k0, $k1	Kernel (reserved for OS)
$28	$gp	Global Pointer
$29	$sp	Stack Pointer
$30	$fp	Frame Pointer
$31	$ra	Return Address (Automatically used in some instructions)

System Calls

Service	Code in $v0	Arguments	Returned Value
print integer	1	$a0 == integer	
print float	2	$f12 == float	
print double	3	$f12 == double	
print string	4	$a0 == address of string	
read integer	5		$v0 <-- integer
read float	6		$f0 <-- float
read double	7		$f0 <-- double
read string	8	$a0 == buffer address $a1 == buffer length	
allocate memory	9	$a0 == number of bytes	$v0 <-- address
exit	10		