

# Lab 4

---

## Lab 4

---

### Pseudo Code:

```
# TODO
# allocate a byte array of 128 bytes on stack to save result
# store the return address in the stack.

# call remove_spaces on the return address

# print the resultant string (a1)

# retrieve the old return address
# put stack pointer back

# identify the registers that need to be preserved, but changed
# save/restore registers

# jump back to the return address to execute following lines
ra
```

### Actual Code:

```

# function
print_ns:
    # TODO
    # allocate a byte array of 128 bytes on stack to save result
    # call remove_spaces
    # print the result string
    # identify the registers that need to be preserved, but changed
    # save/restore registers

    # Make stack
    # addi sp, sp, -4 # Drop the stack pointer down by 4
    # sw a0, 0(sp) # Save a0 via sp
    addi sp, sp, -128 #Shift pointer sp down by 128
    add a1, zero, sp # Save a1 in sp

    addi sp, sp, -4 # Drop the stack pointer down by 4
    sw ra, 0(sp) # save ra via sp

    # Call Function
    # Extend stack for chars for into
    jal ra, remove_spaces # jump to remove_spaces

    #print a1
    li a7, 4 # Service 4 is print string
    add a0, a1, zero # Load desired value into a0 for the call
    ecall # Make the Call

    #retrieve the old return address
    lw ra, 0(sp)

    #put the stack pointer back
    addi sp, sp, 132

    #return
    jr ra

```

```

remove_spaces:
    # TODO
    # copy your code from lab 3 here
    # it should work if it uses only temporary and argument registers
    # make necessary changes if needed

    addi t4, x0, 0 # i = 0
    addi t5, x0, 0 # j = 0
    addi a2, x0, 32 # len till stop
loop:
    add t1, a0, t4 # go to str[0] + i
    lb t6, 0(t1) # c = str[i] read byte at i in the string
    bne t6, a2, if # if c != 32, goto if
    beq x0, x0, ifexit # Else, Exitif
if:
    add t1, a1, t5 # a1 + j
    sb t6, 0(t1) #res[j] = c
    addi t5, t5, 1 # j += 1
ifexit:
    addi t4, t4, 1 #i += 1
    beq t6, zero, exitloop # if str[i] = 0, exit
    beq x0, x0, loop # Else, go to the loop again

exitloop:

```

```

#li a7, 4 # Load print statment
#add a0, a1, zero #stage a0
#ecall          # make the call

jr      ra

```

## Return Address

This is the return address being retrieved from memory.

<input type="checkbox"/>	0x0040004c	0x00000073	ecall	63:	ecall	# Make the Call
<input checked="" type="checkbox"/>	0x00400050	0x00012083	lw x1, 0(x2)	66:	lw ra, 0(sp)	
<input type="checkbox"/>	0x00400054	0x08410113	addi x2, x2, 0x00000084	69:	addi sp, sp, 132	

Here is the data being stored as text over the array of memory.

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x676e6962	0x6d207375	0x69642063	0x7375676e	0x0000000a	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

## Test Runs

Messages

Run I/O

Reset: reset completed.  
  
\*\*\*\* user input : This is a stirng with spaces  
Thisisastirngwithspaces  
\*\*\*\* user input : bing bong  
bingbong  
\*\*\*\* user input : whoopty doopty dooo  
whoptydooptydooo  
\*\*\*\* user input : ok so this is it  
oksothisisit  
\*\*\*\* user input : this is the thing  
thisisthething  
\*\*\*\* user input : it totally works  
ittotallyworks  
\*\*\*\* user input : and is fine  
andisfine  
\*\*\*\* user input : w o w t h a t s t e r r i b b l e  
wowthatsterrible

Clear

Messages

Run I/O

Reset: reset completed.  
  
\*\*\*\* user input : dubs kis 101  
dubskisl01  
\*\*\*\* user input : t a k e m e t o y o u r l e a d e r  
takemetoyourleader  
\*\*\*\* user input : beep boop  
beepboop

Clear

Maximum string size is the same as in lab 3, capped at 32 characters.

### Analysis:

My code seems to work, the calling functions and saving the text stripped across memory is pretty neat, and you could see that my return address was saved in memory as 0x00400050. Pretty neat.