ACCEP
Communication and Computer Engineering Program

كلية الهندسة بشبرا
FACULTY OF ENGINEERING AT SHOUBRA

ASSEMBLY ASSIGNMENT

COMPUTER ORGANIZATION
CCE307

Under the supervision of:
Dr. Lamia Alrefaai

By Farida Waheed Abd El Bary
221903168

Leaf Example

- **Code**

```
.data
    prompt_g: .asciiz "Enter the value of g: "
    prompt_h: .asciiz "Enter the value of h: "
    prompt_i: .asciiz "Enter the value of i: "
    prompt_j: .asciiz "Enter the value of j: "
    result_msg: .asciiz "The result f is: "
    g:  .word 0
    h:  .word 0
    i:  .word 0
    j:  .word 0
    f:  .word 0

.text
main:
    # Prompt the user to enter the value of g
    li  $v0, 4
    la  $a0, prompt_g
    syscall

    # Read the value of g
    li  $v0, 5
    syscall
    move $t0, $v0    # Store the value of g in $t0
    sw   $t0, g      # Store the value of g in memory

    # Prompt the user to enter the value of h
    li  $v0, 4
    la  $a0, prompt_h
    syscall

    # Read the value of h
    li  $v0, 5
    syscall
    move $t1, $v0    # Store the value of h in $t1
    sw   $t1, h      # Store the value of h in memory

    # Prompt the user to enter the value of i
    li  $v0, 4
    la  $a0, prompt_i
    syscall

    # Read the value of i
    li  $v0, 5
    syscall
    move $t2, $v0    # Store the value of i in $t2
    sw   $t2, i      # Store the value of i in memory
```

```
47
48      # Prompt the user to enter the value of j
49      li  $v0, 4
50      la  $a0, prompt_j
51      syscall
52
53      # Read the value of j
54      li  $v0, 5
55      syscall
56      move $t3, $v0    # Store the value of j in $t3
57      sw   $t3, j      # Store the value of j in memory
58
59      # Compute f = (g + h) - (i + j)
60      lw   $t0, g        # Load g from memory into $t0
61      lw   $t1, h        # Load h from memory into $t1
62      lw   $t2, i        # Load i from memory into $t2
63      lw   $t3, j        # Load j from memory into $t3
64
65      add  $t4, $t0, $t1    # $t4 = g + h
66      add  $t5, $t2, $t3    # $t5 = i + j
67      sub  $t6, $t4, $t5    # $t6 = (g + h) - (i + j)
68
69      # Store the result f in memory
70      sw   $t6, f
71
72      # Display the result f
73      li   $v0, 4
74      la   $a0, result_msg
75      syscall
76
77      li   $v0, 1
78      lw   $a0, f
79      syscall
80
81      # End program
82      li   $v0, 10
83      syscall
```

• Description of the program

Problem Statement: The program prompts the user to enter four integer values ***g***, ***h***, ***i***, and ***j***. It then computes the value of ***f*** using the formula ***f=(g+h)−(i+j)*** and displays the result.

Input: Suppose the user enters the following values: ***g***=40, ***h***=30, ***i***=20, ***j***=10.

Output: After processing the input values, the program displays the computed result ***f***. For the given input, the output would be "The result ***f*** is: 40".

• Updates Added

String Prompts: Added string prompts (***prompt_g***, ***prompt_h***, ***prompt_i***, ***prompt_j***) to ask the user to enter the values of ***g***, ***h***, ***i***, and ***j*** respectively. Also, added a string message (***result_msg***) to display the result ***f***.

Memory Allocation: Reserved memory space (***g***, ***h***, ***i***, ***j***, and ***f***) to store the input values and the result.

User Input Handling: Prompted the user to enter each value of ***g***, ***h***, ***i***, and ***j***, and stored them in their respective memory locations.

Computation: Calculated the value of ***f*** using the formula provided and stored the result in the memory location ***f***.

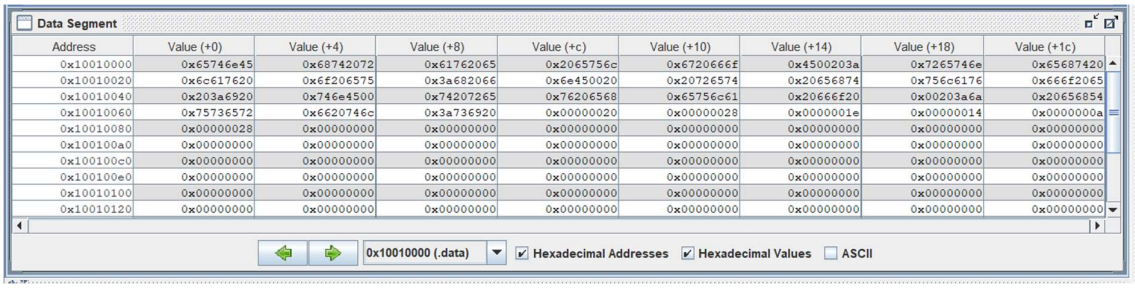
Output Display: Printed the result message (***result_msg***) along with the computed value of ***f*** to the console.

• Register Values

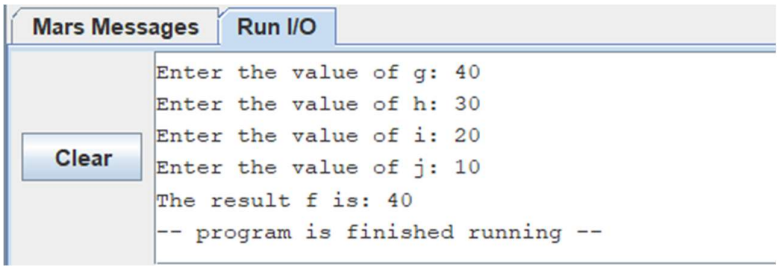
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000028
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000028
$t1	9	0x0000001e
$t2	10	0x00000014
$t3	11	0x0000000a
$t4	12	0x00000046
$t5	13	0x0000001e
$t6	14	0x00000028
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x004000ec
hi		0x00000000
lo		0x00000000

- **Data Segment**



- **Input & Output**



Factorial

- **Code**

```
.data
    prompt: .asciiz "Enter a number to find its factorial:"
    result: .asciiz "\nThe factorial of the number is :"
    theNumber: .word 0
    theAnswer: .word 0

.text
main:
    # Read the number from the user
    li $v0, 4               # Load immediate: syscall code for printing a string
    la $a0, prompt          # Load address: address of the prompt message
    syscall                 # System call to print the prompt message
    li $v0, 5               # Load immediate: syscall code for reading an integer
    syscall                 # System call to read the integer entered by the user
    sw $v0, theNumber       # Store the entered number in the memory location theNumber

    # Call the factorial function
    lw $a0, theNumber       # Load the number from memory into $a0
    jal findFactorial       # Jump and link to the findFactorial function
    sw $v0, theAnswer       # Store the result (factorial) in the memory location theAnswer

    # Display the results
    li $v0, 4               # Load immediate: syscall code for printing a string
    la $a0, result          # Load address: address of the result message
    syscall                 # System call to print the result message
    li $v0, 1               # Load immediate: syscall code for printing an integer
    lw $a0, theAnswer       # Load the factorial from memory into $a0
```

```
    syscall                 # System call to print the factorial

    # Tell the OS that this is the end of the program
    li $v0, 10              # Load immediate: syscall code for program termination
    syscall                 # System call to terminate the program

# findFactorial function
findFactorial:
    subu $sp, $sp, 8        # Adjust stack pointer to make space for local variables
    sw $ra, ($sp)           # Save the return address on the stack
    sw $s0, 4($sp)          # Save $s0 register on the stack

    # Base case: if the number is 0, return 1
    li $v0, 1                       # Load immediate: result set to 1
    beq $a0, $zero, factorialDone   # Branch to factorialDone if the number is 0

    # Recursive case: calculate factorial(n-1)
    move $s0, $a0           # Save the original number in $s0
    sub $a0, $a0, 1         # Decrement the number by 1
    jal findFactorial       # Recursive call to findFactorial with n-1

    # Multiply the result of factorial(n-1) by the original number
    mul $v0, $s0, $v0       # Multiply the original number by the result of factorial(n-1)

factorialDone:
    lw $ra, ($sp)           # Restore the return address from the stack
    lw $s0, 4($sp)          # Restore $s0 register from the stack
    addu $sp, $sp, 8        # Restore the stack pointer
    jr $ra                  # Jump back to the return address to return from the function
```

• Description of the program

Problem Statement: Write a MIPS assembly program that prompts the user to enter a number, calculates its factorial, and displays the result.

Input: Suppose the user enters the following value: ***theNumber***=5, “Enter a number to find its factorial: 5".

Output: After processing the input value, the program displays the computed result ***theAnswer***. For the given input, the output would be "The factorial of the number is: 120".

• Updates Added

String Prompts: Added string prompts prompt to ask the user to enter a number to find its factorial. Introduced a string message result to display the factorial result.

Memory Allocation: Reserved memory space for (***theNumber***) to store the entered number and (***theAnswer***) to store the factorial result.

User Input Handling: Modified the main function to prompt the user to enter a number. Utilized system calls to read the integer entered by the user and stored it in the memory location (***theNumber***).

Computation: Implemented the (***findFactoria***l) function to calculate the factorial recursively. Initialized stack pointer ***$sp*** to allocate space for local variables. Saved the return address and ***$so*** register on the stack.

Handled base case: If the input number is 0, set the result to 1 and branched to (***factorialDone***). For the recursive case, decremented the input number by 1 and made a recursive call to (***findFactorial***). Multiplied the result of (***findFactorial(n-1)***) by the original number to compute the factorial.

Stored Results in Memory: Stored the calculated factorial in the memory location (***theAnswer***).

Output Display: Printed the result message result along with the computed factorial stored in (***theAnswer***) using system calls.

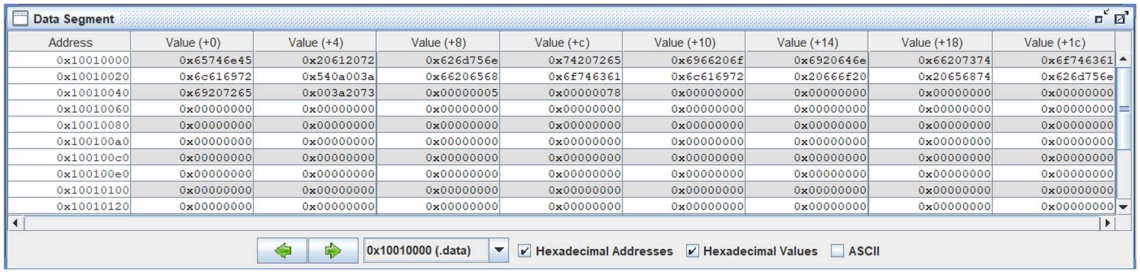
Program Termination: Included a system call to terminate the program after displaying the result.

- **Register Values**

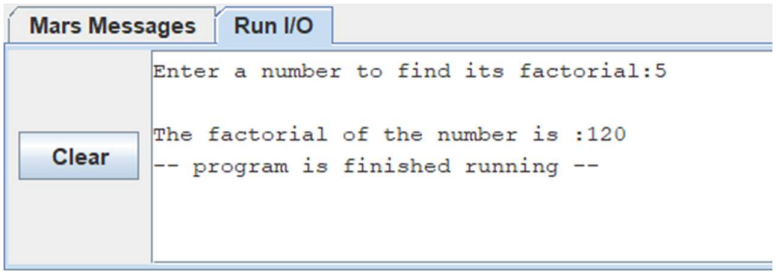
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000078
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x0040002c
pc		0x0040005c
hi		0x00000000
lo		0x00000078

- **Data Segment**



- **Input & Output**



Sort

- **Code**

```
.data
    array: .space 100              # Reserve space for 25 integers (each integer is 4 bytes)
    size: .asciiz "Enter size: "   # Prompt for entering the size of the array
    int: .asciiz "Enter int: "     # Prompt for entering an integer
    final: .asciiz "\nSorted: "    # Message to indicate the sorted array

.text
main:
    # Print "Enter size: " message and read the size of the array from the user
    li $v0, 4           # System call code for printing a string
    la $a0, size        # Load the address of the size prompt into $a0
    syscall             # Print the size prompt
    li $v0, 5           # System call code for reading an integer
    syscall             # Read the size of the array from the user
    move $s1, $v0       # Store the size of the array in $s1
    sub $s1, $s1, 1     # Decrement the size by 1 (since arrays start from index 0)

addint:
    # Loop to input integers into the array
    la $a0, int         # Load the address of the int prompt into $a0
    li $v0, 4           # System call code for printing a string
    syscall             # Print the int prompt
    li $v0, 5           # System call code for reading an integer
    syscall             # Read an integer from the user
    move $t3, $v0       # Store the integer in $t3
    add $t1, $zero, $zero   # Initialize loop counter to 0
    sll $t1, $t0, 2     # Calculate the offset of the next element in the array
```

```
    sw $t3, array($t1)  # Store the integer in the array at the calculated offset
    addi $t0, $t0, 1    # Increment the loop counter (index)
    slt $t1, $s1, $t0   # Check if all integers have been inputted
    beq $t1, $zero, addint  # If not, repeat the loop

    # Sort the array using sort
    la $a0, array       # Load the address of the array into $a0
    addi $a1, $s1, 1    # Add 1 to the size of the array (for comparison)
    jal sort            # Call sort to sort the array

    # Print "Sorted: " message
    la $a0, final       # Load the address of the final message into $a0
    li $v0, 4           # System call code for printing a string
    syscall             # Print the final message

    # Print the sorted array
    la $t0, array       # Load the base address of the array into $t0
    li $t1, 0           # Initialize loop counter to 0
print:
    lw $a0, 0($t0)      # Load the integer from the array
    li $v0, 1           # System call code for printing an integer
    syscall             # Print the integer
    addi $t0, $t0, 4    # Move to the next element in the array
    addi $t1, $t1, 1    # Increment loop counter
    slt $t2, $s1, $t1   # Check if all integers have been printed
    beq $t2, $zero, print  # If not, repeat the loop

    # End program
    li $v0, 10          # System call code for program termination
    syscall             # Terminate the program

sort:
    # Save registers $s0, $s1, $s2, $s3, $ra on the stack
    addi $sp, $sp, -20   # Allocate space on the stack for 5 registers
    sw $ra, 16($sp)      # Save $ra on the stack
    sw $s3, 12($sp)      # Save $s3 on the stack
    sw $s2, 8($sp)       # Save $s2 on the stack
    sw $s1, 4($sp)       # Save $s1 on the stack
    sw $s0, 0($sp)       # Save $s0 on the stack

    move $s2, $a0        # Save the base address of the array into $s2
    move $s3, $a1        # Save the size of the array into $s3
    li $s0, 0            # Initialize outer loop counter (i = 0)

for1tst:
    slt $t0, $s0, $s3     # Check if outer loop counter exceeds array size (i < n)
    beq $t0, $zero, exit1 # If i >= n, exit the outer loop
    addi $s1, $s0, -1     # Initialize inner loop counter (j = i - 1)

for2tst:
    slti $t0, $s1, 0      # Check if inner loop counter is negative (j < 0)
    bne $t0, $zero, exit2 # If j < 0, exit the inner loop
    sll $t1, $s1, 2       # Calculate offset of current element (j * 4)
    add $t2, $s2, $t1     # Calculate address of current element (base address + offset)
```

```
    lw $t3, 0($t2)         # Load word from current element (v[j])
    lw $t4, 4($t2)         # Load word from next element (v[j + 1])
    slt $t0, $t4, $t3      # Compare adjacent elements (v[j] < v[j + 1])
    beq $t0, $zero, exit2 # If v[j] >= v[j + 1], exit the inner loop

    # Swap v[j] and v[j + 1]
    sw $t4, 0($t2)       # Store v[j + 1] at v[j]
    sw $t3, 4($t2)       # Store v[j] at v[j + 1]
    addi $s1, $s1, -1    # Decrement inner loop counter (j -= 1)
    j for2tst            # Jump to the test condition of the inner loop

exit2:
    addi $s0, $s0, 1    # Increment outer loop counter (i += 1)
    j for1tst           # Jump to the test condition of the outer loop

exit1:
    # Restore saved registers from the stack
    lw $s0, 0($sp)       # Restore $s0 from the stack
    lw $s1, 4($sp)       # Restore $s1 from the stack
    lw $s2, 8($sp)       # Restore $s2 from the stack
    lw $s3, 12($sp)      # Restore $s3 from the stack
    lw $ra, 16($sp)      # Restore $ra from the stack
    addi $sp, $sp, 20    # Restore stack pointer
    jr $ra               # Return to the calling routine
```

• Description of the program

Problem Description: The program prompts the user to enter the size of the array and then the individual integers. After sorting, it displays the sorted array.

Input: Suppose the user enters the following values: ***size***=5, ***int***=3, ***int***=1, ***int***=5, ***int***=2, ***int***=4, “Enter size: 5”, “Enter int: 3”, “Enter int: 1”, “Enter int: 5”, “Enter int: 2”, “Enter int: 4”.

Output: After processing the input values, the program displays the computed result ***final***. For the given input, the output would be " Sorted: 1 2 3 4 5".

• Updates Added

String Prompts: Added string prompts (***size***) and (***int***) to ask the user to enter the size of the array and individual integers respectively. Introduced a string message (***final***) to indicate the sorted array.

Memory Allocation: Reserved memory space for the array using (***space***) directive.

User Input Handling: Modified the (***main***) function to prompt the user to enter the size of the array and individual integers. Utilized system calls to read the integers entered by the user and stored them in the array.

Sorting Algorithm Implementation: Implemented the bubble sort algorithm in the (***sort***) function. Saved and restored necessary registers on the stack. Used nested

loops to iterate through the array and perform comparisons and swaps. Sorted the array in ascending order.

Output Display: Printed the final message (***final***) along with the sorted array using system calls.

- **Register Values**

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000005
$a1	5	0x00000005
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x10010014
$t1	9	0x00000005
$t2	10	0x00000001
$t3	11	0x00000003
$t4	12	0x00000004
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000004
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400070
pc		0x004000b0
hi		0x00000000
lo		0x00000000

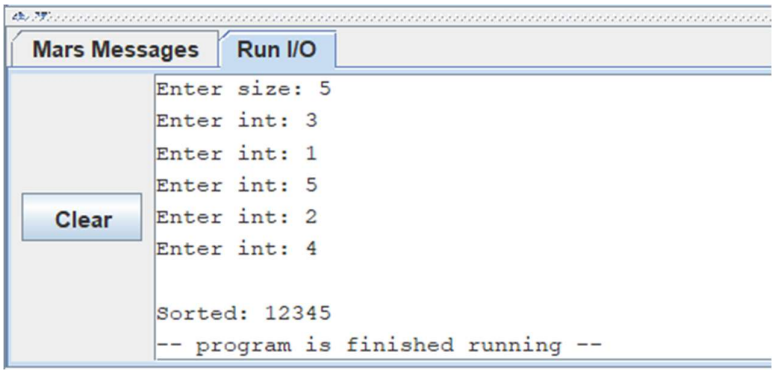
- **Data Segment**

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x65746e45	0x69732072	0x203a657a	0x746e4500	0x69207265	0x203a746e	0x6f530a00
0x10010080	0x64657472	0x0000203a	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

0x10010000 (.data) | Hexadecimal Addresses | Hexadecimal Values | ASCII

- **Input & Output**



String Copy

- **Code**

```
.data
    x: .space 128                       # Memory space reserved for string X
    y: .space 128                       # Memory space reserved for string Y
    prompt: .asciiz "Enter a string: "  # Prompt for entering the string

.text
main:
    # Print prompt to enter the string
    li $v0, 4                   # System call code for printing a string
    la $a0, prompt              # Load the address of the prompt
    syscall                     # Print the prompt

    # Read the string from the user
    li $v0, 8                   # System call code for reading a string
    la $a0, y                   # Load the address of the buffer for the string
    li $a1, 128                 # Maximum number of characters to read
    syscall                     # Read the string from the user

    la $a0, x                   # Load the address of the destination string (X[])
    la $a1, y                   # Load the address of the source string (Y[])
    jal strcpy                  # Call the strcpy function to copy Y[] to X[]

    # Print the copied string in X[]
    li $v0, 4                   # System call code for printing a string
    la $a0, x                   # Load the address of the copied string (X[])
    syscall                     # Print the copied string to the console

    # End the program
    li $v0, 10                   # System call code for program termination
    syscall                      # Terminate the program

strcpy:
    addi $sp, $sp, -4           # Adjust stack for 1 item
    sw $s0, 0($sp)              # Save $s0

```

```
36      add $s0, $zero, $zero      # Initialize loop counter (i = 0)
37
38  L1:
39      add $t1, $s0, $a1          # Calculate address of y[i] in $t1
40      lbu $t2, 0($t1)            # Load character y[i] into $t2
41
42      add $t3, $s0, $a0          # Calculate address of x[i] in $t3
43      sb $t2, 0($t3)             # Store character y[i] into x[i]
44
45      beq $t2, $zero, L2         # Exit loop if y[i] == 0 (reached end of string)
46      addi $s0, $s0, 1           # Increment loop counter (i = i + 1)
47      j L1                        # Jump back to L1 for next iteration
48
49  L2:
50      lw $s0, 0($sp)             # Restore saved $s0
51      addi $sp, $sp, 4           # Pop 1 item from stack
52      jr $ra                     # Return from function
53
```

• Description of the program

Problem Description: The program prompts the user to enter a string, copies it into another string, and then prints the copied string.

Input: Suppose the user enters the following sentence: ***y***= Hello, world!, “Enter a string: Hello, world!”.

Output: After processing the input sentence, the program displays the computed result ***x***. For the given input, the output would be " Hello, world!".

• Updates Added

String Input Handling: Modified the main function to prompt the user to enter a string. Utilized system calls to read the string entered by the user and stored it in the string ***y***.

String Copying Functionality: Implemented the (***strcpy***) function to copy the contents of string ***y*** to string ***x***. Saved and restored necessary registers on the stack. Used a loop to iterate through each character of string ***y*** and copy it to the corresponding position in string ***x***.

Output Display: Printed the copied string stored in string ***x*** using system calls.

- **Register Values**

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010000
$a1	5	0x10010080
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x1001008e
$t2	10	0x00000000
$t3	11	0x1001000e
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400038
pc		0x00400050
hi		0x00000000
lo		0x00000000

- **Data Segment**

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x6c6c6548	0x57202c6f	0x646c726f	0x00000a21	0x00000000	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	0x6c6c6548	0x57202c6f	0x646c726f	0x00000a21	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	0x65746e45	0x20612072	0x69727473	0x203a676e	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) | Hexadecimal Addresses | Hexadecimal Values | ASCII

- **Input & Output**

Mars Messages | Run I/O

```
Enter a string: Hello, World!
Hello, World!

-- program is finished running --
```

Clear

F To C

• Code

```
.data
    input: .asciiz "Enter Temperature in Fahrenheit: "   # Prompt for input
    output: .asciiz "Temperature in Celsius: "           # Message for output
    delta: .float 32.0                                   # Constant for Fahrenheit to Celsius conversion
    scalar: .float 5.0                                   # Constant for Fahrenheit to Celsius conversion
    another_scalar: .float 9.0                           # Constant for Fahrenheit to Celsius conversion

.text
li $v0, 4           # System call code for printing a string
la $a0, input       # Load address of the input prompt into $a0
syscall             # Print the input prompt

li $v0, 6           # System call code for reading a float (Fahrenheit temperature)
syscall             # Read the Fahrenheit temperature into $f0

l.s $f1, delta              # Load the constant 32.0 into $f1
l.s $f2, scalar             # Load the constant 5.0 into $f2
l.s $f3, another_scalar     # Load the constant 9.0 into $f3

sub.s $f0, $f0, $f1   # Subtract 32.0 (delta) from the Fahrenheit temperature
mul.s $f0, $f0, $f2   # Multiply the result by 5.0 (scalar)
div.s $f0, $f0, $f3   # Divide the result by 9.0 (another_scalar) to get Celsius temperature

li $v0, 4             # System call code for printing a string
la $a0, output        # Load address of the output message into $a0
syscall               # Print the output message

li $v0, 2           # System call code for printing a float
mov.s $f12, $f0     # Load the Celsius temperature into $f12
syscall             # Print the Celsius temperature

li $v0, 10          # System call code for program termination
syscall             # Terminate the program
```

• Description of the program

Problem Description: The program prompts the user to enter a temperature in Fahrenheit, performs the conversion, and then displays the result in Celsius.

Input: Suppose the user enters the following temperature: ***input***= 68.0, “Enter Temperature in Fahrenheit: 68.0”.

Output: After processing the input temperature, the program displays the computed result ***output***. For the given input, the output would be “Temperature in Celsius: 20.0”.

• Updates Added

Floating-Point Arithmetic Initialization: Modified the program to load three floating-point constants (***32.0***, ***5.0***, and ***9.0***) into floating-point registers (***$f1***, ***$f2***, and ***$f3*** respectively).

Temperature Conversion: Subtracted the constant ***32.0*** (***delta***) from the Fahrenheit temperature. Multiplied the result by the constant ***5.0*** (***scalar***). Divided the result by the constant ***9.0*** (***another_scalar***) to obtain the temperature in Celsius.

Output Display: Modified the program to print the calculated Celsius temperature using system calls.

- **Register Values**

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010022
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400060
hi		0x00000000
lo		0x00000000

- **Data Segment**

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x65746e45	0x65542072	0x7265706d	0x72757461	0x6e692065	0x68614620	0x686e6572	0x3a746965
0x10010020	0x65540020	0x7265706d	0x72757461	0x6e692065	0x6c654320	0x73756973	0x0000203a	0x42000000
0x10010040	0x40a00000	0x41100000	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

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

- **Input & Output**