0801CS221114
PURVI PORWAL

Lab Assignment - 03

Q1. Write a assembly program for Hello world and print it into console

```
.text
main:
li $v0,4
la $a0, greeting
syscall
li $v0, 10
syscall
.data
greeting: .asciiz "Hello World"
```

Q2. Write a program to read an integer number from user and print to the console.

```
.text
main:
# Prompt for the integer to enter
li $v0, 4
la $a0, prompt
syscall
li $v0, 5
syscall
move $s0, $v0
li $v0, 4
la $a0, output
syscall
li $v0, 1
move $a0, $s0
syscall
li $v0, 10
syscall
.data
prompt: .asciiz "Please enter an integer: "
```

```
output: .asciiz "\nYou typed the number "
```

Q3. Program to prompt and read a string from a user.

```
.data
input_buffer: .space 64
prompt: .asciiz "Enter a string: "
.text
.globl main
main:
 # Print the prompt
 li $v0, 4
 la $a0, prompt
 syscall
 # Read user input
 li $v0, 8
 la $a0, input_buffer
 li $a1, 64 # Maximum number of characters to read
 syscall
 # Display the entered string
 li $v0, 4
 la $a0, input_buffer
 syscall
 # Exit the program
 li $v0, 10
 syscall
```

Q4. Write a program to print out a random number from 1..100.

```
.data
newline: .asciiz "\n"
.text
.globl main
main:
 # Initialize random seed with the current time
 li $v0, 40 # syscall code for getting current time
 syscall
 move $s0, $v0 # store the current time in $s0
 # Seed the random number generator
 li $v0, 42 # syscall code for seeding random number generator
 move $a0, $s0 # load the seed (current time) into $a0
 syscall
 # Generate a random number between 1 and 100
 li $v0, 42 # syscall code for generating a random integer
 li $a0, 100 # set the upper bound (exclusive) for the random number
(101-1)
 syscall
 # Add 1 to the generated number to make it inclusive
 addi $v0, $v0, 1
 # Print the random number
 li $v0, 1 # syscall code for printing an integer
 move $a0, $v0 # load the random number to be printed into $a0
 syscall
 # Print a newline character
 li $v0, 4 # syscall code for printing a string
 la $a0, newline # load the newline character into $a0
 syscall
 # Exit the program
 li $v0, 10 # syscall code for program exit
 syscall
```

Q5. Write a MIPS assembly program called Problem2.asm. This program should have
two global variables, which stores numeric values. The program itself should sum those
two values and then print the result to the console.

```
.data
 # Define two global variables
 num1: .word 10 # Initialize num1 with a value of 10
 num2: .word 20 # Initialize num2 with a value of 20
 result_msg: .asciiz "The result is: "
```

```
.text
.globl main
main:
 # Load the values of num1 and num2 into registers
 lw $t0, num1 # Load num1 into $t0
 lw $t1, num2 # Load num2 into $t1
 # Add the values of num1 and num2
 add $t2, $t0, $t1 # $t2 = $t0 + $t1
 # Print the result message
 li $v0, 4 # syscall code for printing a string
 la $a0, result_msg # Load the address of the result message
 syscall
 # Print the result (in $t2)
 li $v0, 1 # syscall code for printing an integer
 move $a0, $t2 # Load the result (in $t2) into $a0
 syscall
 # Exit the program
 li $v0, 10 # syscall code for program exit
 syscall
```

Q6. The program should have two temporary registers , which stores numeric values.
The program must have multiply these two numbers and the result is store in the saved
value registers.

```
.text
.globl main
main:
 # Load values into temporary registers
 li $t0, 5 # Load a value into $t0 (you can change this value)
 li $t1, 7 # Load another value into $t1 (you can change this value)
 # Multiply the values in $t0 and $t1
 mul $s0, $t0, $t1 # $s0 = $t0 * $t1
 # Print the result (in $s0)
 li $v0, 1 # syscall code for printing an integer
 move $a0, $s0 # Load the result (in $s0) into $a0
 syscall
 # Exit the program
 li $v0, 10 # syscall code for program exit
 syscall
```

Q7. Write a program to find out the square of an number and print the result to the
console.

```
.text
.globl main
main:
 # Load a number into a temporary register $t0 (you can change this value)
 li $t0, 5 # Load the number 5 into $t0 (you can change this number)
 # Square the number ($t0 * $t0)
 mul $t0, $t0, $t0 # $t0 = $t0 * $t0
 # Print the result (the squared number in $t0)
 li $v0, 1 # syscall code for printing an integer
 move $a0, $t0 # Load the result (in $t0) into $a0
 syscall
 # Exit the program
 li $v0, 10 # syscall code for program exit
 syscall
```

Q8. Convert the following program

```
.text
main:
li $v0, 4
la $a0, result1
syscall
li $v0, 4
li $a0, 4
syscall
li $v0, 4
la $a0, result2
```

```
syscall
li $v0, 1
li $a0, 8
syscall
addi $v0, $zero, 10 #Exit program
syscall
.data
result1: .asciiz "\nfirst value = "
result2: .asciiz "\nsecond value = "
#include <stdio.h>
int main() {
 printf("\nfirst value = ");
 printf("%d", 4);
 printf("\nsecond value = ");
 printf("%d", 8);
 return 0;
}
```

Q9. Write a program to retrieve two numbers from a user, and swap those number
using only the XOR operation. You should not use a temporary variable to store the
numbers while swapping them. Your program should include a proper and useful
prompt for input, and print the results in a meaningful manner

```
.data
prompt1: .asciiz "Enter the first number: "
prompt2: .asciiz "Enter the second number: "
result1: .asciiz "After swapping, the first number is: "
result2: .asciiz "After swapping, the second number is: "
.text
.globl main
main:
 # Prompt for the first number
 li $v0, 4
```

```
la $a0, prompt1
syscall
# Read the first number
li $v0, 5
syscall
move $s0, $v0 # Save the first number in $s0
# Prompt for the second number
li $v0, 4
la $a0, prompt2
syscall
# Read the second number
li $v0, 5
syscall
move $s1, $v0 # Save the second number in $s1
# Swap the numbers using XOR
xor $s0, $s0, $s1 # $s0 = $s0 XOR $s1
xor $s1, $s0, $s1 # $s1 = $s0 XOR $s1
xor $s0, $s0, $s1 # $s0 = $s0 XOR $s1
# Print the swapped numbers
li $v0, 4
la $a0, result1
syscall
li $v0, 1
move $a0, $s0
syscall
li $v0, 4
la $a0, result2
syscall
li $v0, 1
move $a0, $s1
syscall
# Exit the program
li $v0, 10
syscall
```

Q10. Write an Assembly code which take an alphabet from user and print the next and
previous alphabets on the screen.
Sample input vs output:
Please enter an alphabet ? d
Previous alphabet in English grammar is : c
You have entered alphabet : d
Next alphabet in English grammar is : e

```
.data
prompt: .asciiz "Please enter an alphabet: "
prev_msg: .asciiz "Previous alphabet in English grammar is : "
next_msg: .asciiz "Next alphabet in English grammar is : "
.text
.globl main
main:
 # Prompt the user to enter an alphabet
 li $v0, 4
 la $a0, prompt
 syscall
 # Read a single character (alphabet) from the user
 li $v0, 12 # Read character (supports case)
 syscall
 move $s0, $v0 # Save the input character in $s0
 # Check if the entered character is lowercase or uppercase
 # If it's lowercase, convert it to uppercase
 li $t0, 'a' # Load 'a' into $t0
 li $t1, 'z' # Load 'z' into $t1
 bge $s0, $t0, check_upper
 addi $s0, $s0, 'A' # Convert to uppercase
 j print_result
check_upper:
 li $t0, 'A' # Load 'A' into $t0
 li $t1, 'Z' # Load 'Z' into $t1
 ble $s0, $t1, print_result
```

```
print_result:
 # Print the previous alphabet
 li $v0, 4
 la $a0, prev_msg
 syscall
 li $v0, 11 # Print a character
 move $a0, $s0 # Load the input character
 syscall
 # Print the entered alphabet
 li $v0, 4
 la $a0, prev_msg
 syscall
 # Calculate and print the next alphabet
 addi $s0, $s0, 1 # Increment the character (next alphabet)
 li $v0, 11 # Print a character
 move $a0, $s0 # Load the next alphabet
 syscall
 # Exit the program
 li $v0, 10
 syscall
```

Q11. Implementation of fibonacci series program In MIPS

```
.data
prompt: .asciiz "Enter the number of Fibonacci terms to generate: "
result_msg: .asciiz "Fibonacci Series:"
.text
.globl main
main:
 # Prompt the user to enter the number of terms
 li $v0, 4
 la $a0, prompt
 syscall
 # Read the number of terms from the user
 li $v0, 5
 syscall
 move $t0, $v0 # Store the input in $t0
 # Initialize the first two terms of the Fibonacci series
 li $t1, 0 # First term
 li $t2, 1 # Second term
 # Print the result message
 li $v0, 4
 la $a0, result_msg
 syscall
 # Print the first two terms (0 and 1)
 li $v0, 1
 move $a0, $t1
 syscall
 li $v0, 1
 move $a0, $t2
 syscall
 # Generate and print the remaining terms
 li $t3, 2 # Initialize loop counter to 2 (already printed first two terms)
generate_fibonacci:
 # Calculate the next term in the Fibonacci series
 add $t3, $t3, 1 # Increment the loop counter
 add $t4, $t1, $t2 # Calculate the next term
 # Print the next term
 li $v0, 1
 move $a0, $t4
 syscall
 # Update $t1 and $t2 for the next iteration
 move $t1, $t2
 move $t2, $t4
 # Check if we've generated the desired number of terms
 bne $t3, $t0, generate_fibonacci
 # Exit the program
 li $v0, 10
 syscall
```

Q12. Write a MIPS program that inputs two integer values. The program should output
equal if the two integers are equal. Otherwise, it should output not equal. Use the
branch instruction to check for equality.

```
.data
prompt1: .asciiz "Enter the first integer: "
prompt2: .asciiz "Enter the second integer: "
equal_msg: .asciiz "equal"
not_equal_msg: .asciiz "not equal"
.text
.globl main
main:
 # Prompt the user to enter the first integer
 li $v0, 4
 la $a0, prompt1
 syscall
 # Read the first integer from the user
 li $v0, 5
 syscall
 move $s0, $v0 # Store the first integer in $s0
 # Prompt the user to enter the second integer
 li $v0, 4
 la $a0, prompt2
 syscall
 # Read the second integer from the user
 li $v0, 5
 syscall
 move $s1, $v0 # Store the second integer in $s1
 # Compare the two integers for equality
 beq $s0, $s1, equal # Branch if $s0 equals $s1
 # If not equal, print "not equal" message
 li $v0, 4
 la $a0, not_equal_msg
 syscall
 j end
equal:
 # If equal, print "equal" message
 li $v0, 4
 la $a0, equal_msg
 syscall
end:
 # Exit the program
 li $v0, 10
 syscall
```

Q13. Variables are of type unsigned integers and word sized. Also the variables b, c, d
and e are initialized to values 10, 20, 30 and 40 respectively.

```
 a = (b + c) - (d + e)
.data
 b: .word 10
 c: .word 20
 d: .word 30
 e: .word 40
 a: .word 0 # Initialize a to 0, will store the result
.text
.globl main
main:
 # Load values of b, c, d, and e into registers
 lw $t0, b # Load b into $t0
 lw $t1, c # Load c into $t1
 lw $t2, d # Load d into $t2
 lw $t3, e # Load e into $t3
 # Perform the calculation: a = (b + c) - (d + e)
 add $t4, $t0, $t1 # $t4 = b + c
 add $t5, $t2, $t3 # $t5 = d + e
 sub $t6, $t4, $t5 # $t6 = a
 # Store the result in variable a
 sw $t6, a
 # Exit the program
 li $v0, 10
 syscall
```

Q14. Write MIPS Assembly Language Programs equivalent to the following C-code
fragments.

```
1. if(a<b)
{ //a and b are signed integers
 c = a - b;
}
2. if(a<b)
{ // a and b are unsigned integers
 c=a-b;
}
3. if ( a < -1234)
{ // a is a signed integer
 a = 4 * a ;
}
else
{
 a = a/4;
}
```

If statement for signed integers:

```
.data
a: .word 5 # Initialize a to a signed integer value
b: .word 8 # Initialize b to a signed integer value
c: .word 0 # Initialize c to store the result
.text
.globl main
main:
 # Load values of a and b into registers
 lw $t0, a # Load a into $t0
 lw $t1, b # Load b into $t1
 # Check if a < b
 slt $t2, $t0, $t1 # $t2 = 1 if a < b, else $t2 = 0
 # If a < b, calculate c = a - b
 beqz $t2, else_label # Branch to else_label if $t2 is 0
 sub $t3, $t0, $t1 # Calculate c = a - b
 sw $t3, c # Store the result in c
 j end_label
else_label:
 # Else, do nothing (c remains 0)
end_label:
 # Exit the program
 li $v0, 10
 syscall
```

If statement for unsigned integers:

```
.data
a: .word 5 # Initialize a to an unsigned integer value
b: .word 8 # Initialize b to an unsigned integer value
c: .word 0 # Initialize c to store the result
.text
.globl main
main:
 # Load values of a and b into registers
 lw $t0, a # Load a into $t0
 lw $t1, b # Load b into $t1
 # Check if a < b
 sltu $t2, $t0, $t1 # $t2 = 1 if a < b, else $t2 = 0
 # If a < b, calculate c = a - b
 beqz $t2, else_label # Branch to else_label if $t2 is 0
 sub $t3, $t0, $t1 # Calculate c = a - b
 sw $t3, c # Store the result in c
 j end_label
else_label:
 # Else, do nothing (c remains 0)
end_label:
 # Exit the program
 li $v0, 10
 syscall
```

If-else statement for a signed integer:

```
.data
a: .word -1500 # Initialize a to a signed integer value
.text
.globl main
main:
 # Load the value of a into a register
 lw $t0, a # Load a into $t0
 # Check if a < -1234
 li $t1, -1234 # Load -1234 into $t1
 slt $t2, $t0, $t1 # $t2 = 1 if a < -1234, else $t2 = 0
 # If a < -1234, calculate a = 4 * a
 beqz $t2, else_label # Branch to else_label if $t2 is 0
 sll $t0, $t0, 2 # Calculate a = 4 * a
 j end_label
else_label:
 # Else, calculate a = a / 4
 sra $t0, $t0, 2 # Calculate a = a / 4
end_label:
 # Exit the program
 li $v0, 10
 syscall
```

---

15. Write a MIPS assembly for the following C codes :

```
int a, b, result
int main(){
a = 0x12345;
b = 7;
result = a + b;
return 1;
}
```

```
.data
a: .word 0x12345 # Initialize a to 0x12345
b: .word 7 # Initialize b to 7
```

```
result: .word 0 # Initialize result to 0
.text
.globl main
main:
 # Load the value of a into $t0
 lw $t0, a
 # Load the value of b into $t1
 lw $t1, b
 # Calculate result = a + b
 add $t2, $t0, $t1
 # Store the result in the result variable
 sw $t2, result
 # Exit the program with return value 1
 li $v0, 10 # syscall code for program exit
 li $a0, 1 # return value 1
syscall
```