Nang Thiri Wutyi

20113

08/03/2024

**Computer Architecture - Homework 5**

1. **Implement a subprogram that prompt the user for 3 numbers, finds the median (middle value) of the 3, and returns that value to the calling program.**

.data

prompt1: .asciiz "Enter the first number: "

prompt2: .asciiz "Enter the second number: "

prompt3: .asciiz "Enter the third number: "

median\_msg: .asciiz "The median value is: "

.text

.globl main

main:

# Prompt for the first number

li $v0, 4 # syscall for print\_string

la $a0, prompt1 # address of the prompt1 string

syscall

# Read the first number

li $v0, 5 # syscall for read\_int

syscall

move $t0, $v0 # store the first number in $t0

# Prompt for the second number

li $v0, 4 # syscall for print\_string

la $a0, prompt2 # address of the prompt2 string

syscall

# Read the second number

li $v0, 5 # syscall for read\_int

syscall

move $t1, $v0 # store the second number in $t1

# Prompt for the third number

li $v0, 4 # syscall for print\_string

la $a0, prompt3 # address of the prompt3 string

syscall

# Read the third number

li $v0, 5 # syscall for read\_int

syscall

move $t2, $v0 # store the third number in $t2

# Find the median

# We need to sort $t0, $t1, and $t2 and find the middle value

# Compare $t0 and $t1

ble $t0, $t1, next1

move $t3, $t0

move $t0, $t1

move $t1, $t3

next1:

# Compare $t1 and $t2

ble $t1, $t2, next2

move $t3, $t1

move $t1, $t2

move $t2, $t3

next2:

# Compare $t0 and $t1 again

ble $t0, $t1, median\_found

move $t3, $t0

move $t0, $t1

move $t1, $t3

median\_found:

# $t1 now holds the median value

# Print the median message

li $v0, 4 # syscall for print\_string

la $a0, median\_msg # address of the median\_msg string

syscall

# Print the median value

li $v0, 1 # syscall for print\_int

move $a0, $t1 # move the median value to $a0

syscall

# Exit the program

li $v0, 10 # syscall for exit

syscall

1. **Implement a recursive program that takes in a number and finds the square of that number through addition. For example, if the number 3 is entered, you would add 3+3+3=9. If 4 is entered, you would add 4+4+4+4=16. This program must be implemented using recursion to add the numbers together.**

.data

prompt: .asciiz "Enter a number: "

result: .asciiz "The square is: "

.text

.globl main

main:

# Print prompt

li $v0, 4

la $a0, prompt

syscall

# Read integer input

li $v0, 5

syscall

move $a0, $v0 # Move input to $a0 for the recursive function

# Call the recursive function to calculate the square

move $a1, $a0 # Copy input to $a1

jal square

move $t0, $v0 # Move result to $t0

# Print the result

li $v0, 4

la $a0, result

syscall

li $v0, 1

move $a0, $t0

syscall

# Exit program

li $v0, 10

syscall

# Recursive function to calculate square using addition

square:

addi $sp, $sp, -8 # Create space on stack

sw $ra, 4($sp) # Save return address

sw $a0, 0($sp) # Save argument

bnez $a0, recursive\_case # If $a0 != 0, go to recursive case

li $v0, 0 # Base case: $a0 == 0, result is 0

j end\_square

recursive\_case:

addi $a0, $a0, -1 # Decrement $a0

jal square # Recursive call

lw $a0, 0($sp) # Load original argument

add $v0, $v0, $a1 # Add original argument to result

end\_square:

lw $ra, 4($sp) # Restore return address

addi $sp, $sp, 8 # Restore stack pointer

jr $ra # Return from function

1. **Write a recursive program to calculate factorial numbers. Use the definition of factorial as F(n) = n \* F(n-1).**

.data

prompt: .asciiz "Enter a number: "

result: .asciiz "The factorial is: "

.text

.globl main

main:

# Print prompt

li $v0, 4

la $a0, prompt

syscall

# Read integer input

li $v0, 5

syscall

move $a0, $v0 # Move input to $a0 for the recursive function

# Call the recursive function to calculate the factorial

jal factorial

move $t0, $v0 # Move result to $t0

# Print the result

li $v0, 4

la $a0, result

syscall

li $v0, 1

move $a0, $t0

syscall

# Exit program

li $v0, 10

syscall

# Recursive function to calculate factorial

factorial:

addi $sp, $sp, -8 # Create space on stack

sw $ra, 4($sp) # Save return address

sw $a0, 0($sp) # Save argument

li $t1, 1 # Load 1 into $t1

beq $a0, $t1, base\_case # If $a0 == 1, go to base case

blez $a0, base\_case # If $a0 <= 0, go to base case

addi $a0, $a0, -1 # Decrement $a0

jal factorial # Recursive call

lw $a0, 0($sp) # Load original argument

mul $v0, $a0, $v0 # Multiply current number with result

j end\_factorial

base\_case:

li $v0, 1 # Base case: return 1

end\_factorial:

lw $ra, 4($sp) # Restore return address

addi $sp, $sp, 8 # Restore stack pointer

jr $ra # Return from function

1. **The following pseudo code converts an input value of a single decimal number from 1 ≤ 𝑛 ≥ 15 into a single hexadecimal digit. Translate this pseudo code into MIPS assembly.**

**main{**

**String a[16] a[0] = "0x0"**

**a[1] = "0x1"**

**a[2] = "0x2"**

**a[3] = "0x3"**

**a[4] = "0x4"**

**a[5] = "0x5"**

**a[6] = "0x6"**

**a[7] = "0x7"**

**a[8] = "0x8"**

**a[9] = "0x9"**

**a[10] = "0xa"**

**a[11] = "0xb"**

**a[12] = "0xc"**

**a[13] = "0xd"**

**a[14] = "0xe"**

**a[15] = "0xf"**

**int i = prompt("Enter a number from 0 to 15 ")**

**print("your number is " + a[i]) }**

.data

prompt: .asciiz "Enter a number from 1 to 15: "

result: .asciiz "Your number is: "

hex\_digits:

.asciiz "0x0"

.asciiz "0x1"

.asciiz "0x2"

.asciiz "0x3"

.asciiz "0x4"

.asciiz "0x5"

.asciiz "0x6"

.asciiz "0x7"

.asciiz "0x8"

.asciiz "0x9"

.asciiz "0xa"

.asciiz "0xb"

.asciiz "0xc"

.asciiz "0xd"

.asciiz "0xe"

.asciiz "0xf"

invalid\_msg: .asciiz "Invalid input. Please enter a number from 1 to 15.\n"

.text

.globl main

main:

# Print prompt

li $v0, 4

la $a0, prompt

syscall

# Read integer input

li $v0, 5

syscall

move $t0, $v0 # Move input to $t0

# Validate input: 1 <= t0 <= 15

li $t1, 1 # Lower bound

li $t2, 15 # Upper bound

blt $t0, $t1, invalid\_input

bgt $t0, $t2, invalid\_input

# Calculate the address of the hex\_digits array

# Each hex string is 4 bytes long

li $t3, 4 # Each hex digit string length

sub $t0, $t0, 1 # Adjust index to be 0-based (1 becomes 0, 15 becomes 14)

mul $t4, $t0, $t3 # Calculate the byte offset

la $t5, hex\_digits # Load the address of hex\_digits

add $t5, $t5, $t4 # Get the address of the desired string

# Print the result

li $v0, 4

la $a0, result

syscall

li $v0, 4

move $a0, $t5 # Move the address of the hex string to $a0

syscall

# Exit program

li $v0, 10

syscall

invalid\_input:

# Print invalid input message and exit

li $v0, 4

la $a0, invalid\_msg

syscall

li $v0, 10

syscall

1. **The following pseudo code program calculates the Fibonacci numbers from 1...n, and stores them in an array. Translate this pseudo code into MIPS assembly, and use the PrintIntArray subprogram to print the results.**

**main{**

**int size = PromptInt(“Enter a max Fibonacci number to calc: “)**

**int Fibonacci[size] Fibonacci[0] = 0**

**Fibonacci[1] = 1**

**for (int i = 2; i < size; i++){**

**Fibonacci[i] = Fibonacci[i-1] + Fibonacci[i-2]**

**}**

**PrintIntArray(Fibonacci, size)**

**}**

.data

prompt: .asciiz "Enter a max Fibonacci number to calc: "

space: .asciiz " "

newline: .asciiz "\n"

array: .word 0:100 # Assuming max size of 100, adjust if needed

.text

.globl main

main:

# Prompt for size

li $v0, 4

la $a0, prompt

syscall

# Read size

li $v0, 5

syscall

move $s0, $v0 # $s0 = size

# Initialize Fibonacci array

la $s1, array # $s1 = base address of array

sw $zero, 0($s1) # Fibonacci[0] = 0

li $t0, 1

sw $t0, 4($s1) # Fibonacci[1] = 1

# Calculate Fibonacci numbers

li $t0, 2 # i = 2

loop:

bge $t0, $s0, end\_loop

# Calculate Fibonacci[i] = Fibonacci[i-1] + Fibonacci[i-2]

sll $t1, $t0, 2 # $t1 = i \* 4 (offset)

add $t2, $s1, $t1 # Address of Fibonacci[i]

lw $t3, -4($t2) # Fibonacci[i-1]

lw $t4, -8($t2) # Fibonacci[i-2]

add $t5, $t3, $t4 # Fibonacci[i-1] + Fibonacci[i-2]

sw $t5, 0($t2) # Store result in Fibonacci[i]

addi $t0, $t0, 1 # i++

j loop

end\_loop:

# Call PrintIntArray

move $a0, $s1 # First argument: base address of array

move $a1, $s0 # Second argument: size

jal PrintIntArray

# Exit program

li $v0, 10

syscall

# PrintIntArray subprogram

PrintIntArray:

move $t0, $a0 # Base address of array

move $t1, $a1 # Size

li $t2, 0 # Counter

print\_loop:

bge $t2, $t1, print\_end

# Print integer

li $v0, 1

lw $a0, 0($t0)

syscall

# Print space

li $v0, 4

la $a0, space

syscall

addi $t0, $t0, 4 # Move to next element

addi $t2, $t2, 1 # Increment counter

j print\_loop

print\_end:

# Print newline

li $v0, 4

la $a0, newline

syscall

jr $ra # Return from subroutine