Nang Thiri Wutyi

20113

7/22/2024

**Homework 4**

1. Implement the following subroutine function in the utils.asm file, properly documenting them, and include programs to test them.

a.Mult10- take an input parameter and return that parameter multiplie

byusing ONLY shift and add operations.

# utils.asm

# Mult10 - Multiplies an input parameter by 10 using only shift and add operations

.text

.globl Mult10

# Mult10: Multiplies an input parameter by 10

# Input: $a0 - integer to be multiplied by 10

# Output: $v0 - result of the multiplication

Mult10:

# Save the input parameter

move $t0, $a0 # $t0 = input

# Perform the multiplication by 10 using shift and add operations

sll $t1, $t0, 3 # $t1 = $t0 \* 8

sll $t2, $t0, 1 # $t2 = $t0 \* 2

add $v0, $t1, $t2 # $v0 = $t1 + $t2 = $t0 \* 10

jr $ra # Return to the caller

# test\_mult10.asm

# Program to test the Mult10 subroutine

.text

.globl main

main:

# Test case 1

li $a0, 5 # Input: 5

jal Mult10 # Call Mult10

move $t0, $v0 # Store result in $t0

# Expected output: 50

# Test case 2

li $a0, 7 # Input: 7

jal Mult10 # Call Mult10

move $t1, $v0 # Store result in $t1

# Expected output: 70

# Test case 3

li $a0, -3 # Input: -3

jal Mult10 # Call Mult10

move $t2, $v0 # Store result in $t2

# Expected output: -30

# End of program

li $v0, 10 # Exit

syscall

b. ToUpper- take a32-bits input which is3characters and a null, or a3-characters string. Conver the 3 characters to upper case if they are lower case or do nothing if they are already upper case.

# utils.asm

# ToUpper - Converts 3 characters to upper case if they are lower case

.text

.globl ToUpper

# ToUpper: Converts 3 characters to upper case if they are lower case

# Input: $a0 - 32-bit integer containing 3 characters and a null character

# Output: $v0 - 32-bit integer with characters converted to upper case

ToUpper:

# Save the input parameter

move $t0, $a0 # $t0 = input (3 chars and null)

# Extract each character

andi $t1, $t0, 0xFF # $t1 = first character (8 bits)

srl $t0, $t0, 8

andi $t2, $t0, 0xFF # $t2 = second character (8 bits)

srl $t0, $t0, 8

andi $t3, $t0, 0xFF # $t3 = third character (8 bits)

# Convert first character to upper case if necessary

blt $t1, 0x61, Skip1 # If char < 'a', skip

bgt $t1, 0x7A, Skip1 # If char > 'z', skip

subi $t1, $t1, 32 # Convert to upper case (subtract 32)

Skip1:

# Convert second character to upper case if necessary

blt $t2, 0x61, Skip2 # If char < 'a', skip

bgt $t2, 0x7A, Skip2 # If char > 'z', skip

subi $t2, $t2, 32 # Convert to upper case (subtract 32)

Skip2:

# Convert third character to upper case if necessary

blt $t3, 0x61, Skip3 # If char < 'a', skip

bgt $t3, 0x7A, Skip3 # If char > 'z', skip

subi $t3, $t3, 32 # Convert to upper case (subtract 32)

Skip3:

# Combine the characters back into a single 32-bit word

sll $t0, $t3, 16 # Move third character to its position

sll $t4, $t2, 8 # Move second character to its position

or $t0, $t0, $t4 # Combine third and second character

or $v0, $t0, $t1 # Combine with the first character

jr $ra # Return to the caller

# test\_toupper.asm

# Program to test the ToUpper subroutine

.text

.globl main

main:

# Test case 1

li $a0, 0x61626300 # Input: "abc\0"

jal ToUpper # Call ToUpper

move $t0, $v0 # Store result in $t0

# Expected output: "ABC\0"

# Test case 2

li $a0, 0x41424300 # Input: "ABC\0"

jal ToUpper # Call ToUpper

move $t1, $v0 # Store result in $t1

# Expected output: "ABC\0"

# Test case 3

li $a0, 0x61614100 # Input: "aA\0"

jal ToUpper # Call ToUpper

move $t2, $v0 # Store result in $t2

# Expected output: "AA\0"

# End of program

li $v0, 10 # Exit

syscall

c. ToLower- take a bits input which is 3 characters and a null, or a 3- characters string. Convert the3

characters to lower case if they are upper case or do nothing if they are already lower case.

# utils.asm

# ToLower - Converts 3 characters to lower case if they are upper case

.text

.globl ToLower

# ToLower: Converts 3 characters to lower case if they are upper case

# Input: $a0 - 32-bit integer containing 3 characters and a null character

# Output: $v0 - 32-bit integer with characters converted to lower case

ToLower:

# Save the input parameter

move $t0, $a0 # $t0 = input (3 chars and null)

# Extract each character

andi $t1, $t0, 0xFF # $t1 = first character (8 bits)

srl $t0, $t0, 8

andi $t2, $t0, 0xFF # $t2 = second character (8 bits)

srl $t0, $t0, 8

andi $t3, $t0, 0xFF # $t3 = third character (8 bits)

# Convert first character to lower case if necessary

blt $t1, 0x41, Skip1 # If char < 'A', skip

bgt $t1, 0x5A, Skip1 # If char > 'Z', skip

addi $t1, $t1, 32 # Convert to lower case (add 32)

Skip1:

# Convert second character to lower case if necessary

blt $t2, 0x41, Skip2 # If char < 'A', skip

bgt $t2, 0x5A, Skip2 # If char > 'Z', skip

addi $t2, $t2, 32 # Convert to lower case (add 32)

Skip2:

# Convert third character to lower case if necessary

blt $t3, 0x41, Skip3 # If char < 'A', skip

bgt $t3, 0x5A, Skip3 # If char > 'Z', skip

addi $t3, $t3, 32 # Convert to lower case (add 32)

Skip3:

# Combine the characters back into a single 32-bit word

sll $t0, $t3, 16 # Move third character to its position

sll $t4, $t2, 8 # Move second character to its position

or $t0, $t0, $t4 # Combine third and second character

or $v0, $t0, $t1 # Combine with the first character

jr $ra # Return to the caller

# test\_tolower.asm

# Program to test the ToLower subroutine

.text

.globl main

main:

# Test case 1

li $a0, 0x41424300 # Input: "ABC\0"

jal ToLower # Call ToLower

move $t0, $v0 # Store result in $t0

# Expected output: "abc\0"

# Test case 2

li $a0, 0x61626300 # Input: "abc\0"

jal ToLower # Call ToLower

move $t1, $v0 # Store result in $t1

# Expected output: "abc\0"

# Test case 3

li $a0, 0x41414100 # Input: "AAA\0"

jal ToLower # Call ToLower

move $t2, $v0 # Store result in $t2

# Expected output: "aaa\0"

# End of program

li $v0, 10 # Exit

syscall

2. Write a program to find prime numbers from 3 to n in a loop in MIPS assembly.

.data

primes: .space 4000 # Space to store prime numbers (adjust size as needed)

prompt: .asciiz "Enter the upper limit (n): "

newline: .asciiz "\n"

prime\_msg: .asciiz "Prime numbers are:\n"

.text

.globl main

main:

# Print the prompt

li $v0, 4 # syscall: print\_string

la $a0, prompt

syscall

# Read integer input from the user

li $v0, 5 # syscall: read\_int

syscall

move $t1, $v0 # $t1 = n

# Initialize array index

la $t2, primes # $t2 points to the start of primes array

# Initialize first prime number

li $t3, 2 # $t3 = 2

sw $t3, 0($t2) # store 2 in primes array

addi $t2, $t2, 4 # move to next array slot

# Initialize loop variable

li $t4, 3 # $t4 = 3 (starting number to check for prime)

check\_prime:

# Check if $t4 > n

bgt $t4, $t1, print\_primes # if $t4 > n, proceed to print primes

# Assume $t4 is prime

li $t5, 1 # $t5 = 1 (flag to indicate if $t4 is prime)

# Initialize divisor

li $t6, 2 # $t6 = 2

check\_divisor:

# If $t6 \* $t6 > $t4, we have confirmed $t4 is prime

mul $t7, $t6, $t6

bgt $t7, $t4, store\_prime

# Check if $t4 % $t6 == 0

div $t4, $t6

mfhi $t7

beq $t7, $zero, not\_prime

# Increment divisor

addi $t6, $t6, 1

j check\_divisor

not\_prime:

# $t4 is not prime, increment $t4 and check next number

li $t5, 0 # $t5 = 0 (indicating $t4 is not prime)

addi $t4, $t4, 1

j check\_prime

store\_prime:

# Check if $t5 is 1 (indicating $t4 is prime)

beq $t5, $zero, increment\_and\_continue

# $t4 is prime, store it in primes array

sw $t4, 0($t2)

addi $t2, $t2, 4

increment\_and\_continue:

# Increment $t4 and check next number

addi $t4, $t4, 1

j check\_prime

print\_primes:

# Print the prime numbers

li $v0, 4 # syscall: print\_string

la $a0, prime\_msg

syscall

la $t2, primes # $t2 points to the start of primes array

print\_loop:

lw $t3, 0($t2) # load prime number from primes array

beq $t3, $zero, end # if 0 is found, end the program

li $v0, 1 # syscall: print\_int

move $a0, $t3

syscall

li $v0, 4 # syscall: print\_string

la $a0, newline

syscall

addi $t2, $t2, 4 # move to next array slot

j print\_loop

end:

# End of program

li $v0, 10 # Exit program

syscall

3. Prompt the user for a number from 3...100 and determine the prime factors for that number. For example, 15 has prime factors 3 and 5. 60 has prime factors 2, 3, and 5. You ONLY have to print out the prime factors.

.data

prompt: .asciiz "Enter a number between 3 and 100: "

newline: .asciiz "\n"

invalid\_input: .asciiz "Invalid input. Please enter a number between 3 and 100.\n"

prime\_factors\_msg: .asciiz "Prime factors are:\n"

.text

.globl main

main:

# Print the prompt

li $v0, 4 # syscall: print\_string

la $a0, prompt

syscall

# Read integer input from the user

li $v0, 5 # syscall: read\_int

syscall

move $t1, $v0 # $t1 = input number

# Check if input is between 3 and 100

li $t2, 3 # lower bound

li $t3, 100 # upper bound

blt $t1, $t2, invalid # if input < 3, print invalid message

bgt $t1, $t3, invalid # if input > 100, print invalid message

# Print the prime factors message

li $v0, 4 # syscall: print\_string

la $a0, prime\_factors\_msg

syscall

# Initialize prime factor to check

li $t4, 2 # $t4 = 2 (first prime number)

find\_factors:

# Check if $t4 \* $t4 > $t1, if so, we are done

mul $t5, $t4, $t4

bgt $t5, $t1, print\_last\_factor

check\_divisibility:

# Check if $t1 % $t4 == 0

div $t1, $t4

mfhi $t6

beq $t6, $zero, print\_factor

# Increment prime factor

addi $t4, $t4, 1

j find\_factors

print\_factor:

# Print the current prime factor

li $v0, 1 # syscall: print\_int

move $a0, $t4

syscall

# Print a newline

li $v0, 4 # syscall: print\_string

la $a0, newline

syscall

# Divide $t1 by the prime factor

div $t1, $t4

mflo $t1

# Check if $t1 is still divisible by $t4

j find\_factors

print\_last\_factor:

# Print the last prime factor if $t1 > 1

li $t2, 1

bgt $t1, $t2, print\_last

# End of program

j end

print\_last:

# Print the last prime factor

li $v0, 1 # syscall: print\_int

move $a0, $t1

syscall

# Print a newline

li $v0, 4 # syscall: print\_string

la $a0, newline

syscall

# End of program

j end

invalid:

# Print the invalid input message

li $v0, 4 # syscall: print\_string

la $a0, invalid\_input

syscall

end:

# Exit program

li $v0, 10 # syscall: exit

syscall

4. Using only sll and srl, implement a program to check if a user input value is even or odd. The program should read a user input integer and print out "The number is even" if the number is even, or "The number is odd", if the number is odd.

.data

prompt: .asciiz "Enter an integer: "

even\_msg: .asciiz "The number is even\n"

odd\_msg: .asciiz "The number is odd\n"

.text

.globl main

main:

# Print the prompt

li $v0, 4 # syscall: print\_string

la $a0, prompt

syscall

# Read integer input from the user

li $v0, 5 # syscall: read\_int

syscall

move $t0, $v0 # $t0 = input number

# Check if the number is even or odd using srl and sll

srl $t1, $t0, 1 # Shift right to get the LSB into $t1

sll $t1, $t1, 1 # Shift left to restore the number minus the LSB

sub $t2, $t0, $t1 # Subtract to isolate the LSB

# Check the LSB

beq $t2, $zero, print\_even # If $t2 == 0, the number is even

j print\_odd # If $t2 != 0, the number is odd

print\_even:

# Print "The number is even"

li $v0, 4 # syscall: print\_string

la $a0, even\_msg

syscall

j end

print\_odd:

# Print "The number is odd"

li $v0, 4 # syscall: print\_string

la $a0, odd\_msg

syscall

end:

# Exit program

li $v0, 10 # syscall: exit

syscall

5. Prompt the user for a number n, 0 < n < 100. Print out the smallest number of coins (quarters, dimes, nickels, and pennies) which will produce n. For example, if the user enters "66", your program should print out " 2 quarters, 1 dime, 1 nickel, and 1penny".

.data

prompt: .asciiz "Enter a number between 1 and 99: "

invalid\_input: .asciiz "Invalid input. Please enter a number between 1 and 99.\n"

quarters\_msg: .asciiz " quarters, "

dimes\_msg: .asciiz " dimes, "

nickels\_msg: .asciiz " nickels, and "

pennies\_msg: .asciiz " pennies.\n"

.text

.globl main

main:

# Print the prompt

li $v0, 4 # syscall: print\_string

la $a0, prompt

syscall

# Read integer input from the user

li $v0, 5 # syscall: read\_int

syscall

move $t0, $v0 # $t0 = input number

# Check if input is between 1 and 99

li $t1, 1 # lower bound

li $t2, 99 # upper bound

blt $t0, $t1, invalid # if input < 1, print invalid message

bgt $t0, $t2, invalid # if input > 99, print invalid message

# Calculate the number of quarters

li $t3, 25 # $t3 = value of a quarter

div $t0, $t3 # $t0 / 25

mflo $t4 # $t4 = number of quarters

mfhi $t0 # $t0 = remaining cents

# Calculate the number of dimes

li $t3, 10 # $t3 = value of a dime

div $t0, $t3 # $t0 / 10

mflo $t5 # $t5 = number of dimes

mfhi $t0 # $t0 = remaining cents

# Calculate the number of nickels

li $t3, 5 # $t3 = value of a nickel

div $t0, $t3 # $t0 / 5

mflo $t6 # $t6 = number of nickels

mfhi $t0 # $t0 = remaining cents

# Remaining cents are the number of pennies

move $t7, $t0 # $t7 = number of pennies

# Print the result

li $v0, 1 # syscall: print\_int

move $a0, $t4 # print number of quarters

syscall

li $v0, 4 # syscall: print\_string

la $a0, quarters\_msg

syscall

li $v0, 1 # syscall: print\_int

move $a0, $t5 # print number of dimes

syscall

li $v0, 4 # syscall: print\_string

la $a0, dimes\_msg

syscall

li $v0, 1 # syscall: print\_int

move $a0, $t6 # print number of nickels

syscall

li $v0, 4 # syscall: print\_string

la $a0, nickels\_msg

syscall

li $v0, 1 # syscall: print\_int

move $a0, $t7 # print number of pennies

syscall

li $v0, 4 # syscall: print\_string

la $a0, pennies\_msg

syscall

# End of program

j end

invalid:

# Print the invalid input message

li $v0, 4 # syscall: print\_string

la $a0, invalid\_input

syscall

end:

# Exit program

li $v0, 10 # syscall: exit

syscall