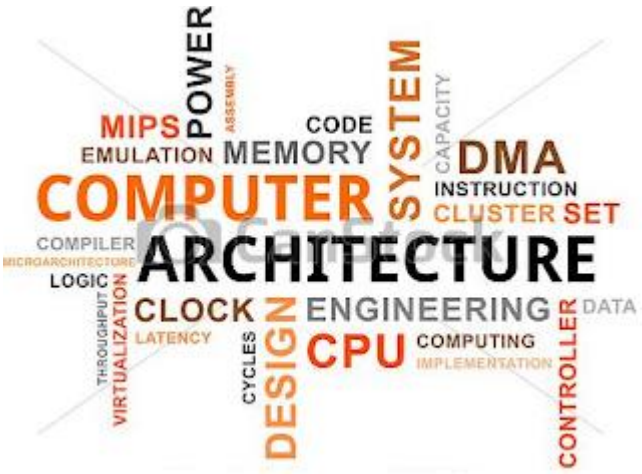
Date:7/22/24

EE488 - Computer Architecture
HW Assignment 4



NAME : S A SABBIRUL MOHOSIN NAIM

ID : 20176

ANSWER TO THE QUESTION NO 1:

(a)

Mult10 - take an input parameter and return that parameter multiplied by 10 using ONLY shift and add operations.

```
# Mult10 subroutine
```

```
# Input: $a0 - the input parameter
# Output: $v0 - the result (input * 10)
# Uses: $t0 for temporary calculation

.text
.globl main

main:
    # Test case 1: Mult10(5)
    li $a0, 5
    jal Mult10

    # Print result
    move $a0, $v0
    li $v0, 1
    syscall

    # Print newline
    li $a0, 10  # ASCII code for newline
    li $v0, 11
    syscall

    # Test case 2: Mult10(12)
```

```
    li $a0, 12
    jal Mult10

    # Print result
    move $a0, $v0
    li $v0, 1
    syscall

    # Exit program
    li $v0, 10
    syscall

Mult10:
    # Multiply by 8
    sll $t0, $a0, 3
    # Multiply by 2
    sll $v0, $a0, 1
    # Add the results (8x + 2x = 10x)
    add $v0, $v0, $t0
    jr $ra
```

(b)

ToUpper - take a 32-bits input which is 3 characters and a null, or a 3-characters string. Convert the 3 characters to upper case if they are lower case or do nothing if they are already upper case.

```
# ToUpper subroutine

# Input: $a0 - address of the 32-bit input (3 characters and a null)

# Output: None (modifies the input string in place)

# Uses: $t0 for address, $t1 for character, $t2 for loop counter


.data

test_str1: .asciiz "abc"

test_str2: .asciiz "XYZ"

test_str3: .asciiz "AbC"

newline: .asciiz "\n"


.text

.globl main


main:

   # Test case 1: "abc"

   la $a0, test_str1

   jal ToUpper

   la $a0, test_str1
```

```
    jal print_string

    # Test case 2: "XYZ"
    la $a0, test_str2
    jal ToUpper
    la $a0, test_str2
    jal print_string

    # Test case 3: "AbC"
    la $a0, test_str3
    jal ToUpper
    la $a0, test_str3
    jal print_string

    # Exit program
    li $v0, 10
    syscall

ToUpper:
    move $t0, $a0      # Copy string address to $t0
    li $t2, 0          # Initialize loop counter

    loop:
```

```
        lb $t1, ($t0)  # Load byte (character)
        beqz $t1, done # If null terminator, exit loop

        # Check if character is lowercase (between 'a' and 'z')
        blt $t1, 'a', next_char
        bgt $t1, 'z', next_char

        # Convert to uppercase by subtracting 32
        subi $t1, $t1, 32
        sb $t1, ($t0)  # Store converted character back

    next_char:
        addi $t0, $t0, 1  # Move to next character
        addi $t2, $t2, 1  # Increment counter
        blt $t2, 3, loop  # If counter < 3, continue loop

    done:
        jr $ra

print_string:
    li $v0, 4
    syscall
    la $a0, newline
```

```
    syscall
    jr $ra
```

(c)

ToLower - take a 32-bits input which is 3 characters and a null, or a 3-characters string. Convert the 3 characters to lower case if they are upper case or do nothing if they are already lower case.

```
# ToLower subroutine
# Input: $a0 - address of the 32-bit input (3 characters and a null)
# Output: None (modifies the input string in place)
# Uses: $t0 for address, $t1 for character, $t2 for loop counter

.data
test_str1: .asciiz "ABC"
test_str2: .asciiz "XYz"
test_str3: .asciiz "AbC"
newline: .asciiz "\n"

.text
.globl main

main:
```

```
    # Test case 1: "ABC"
    la $a0, test_str1
    jal ToLower
    la $a0, test_str1
    jal print_string

    # Test case 2: "XYz"
    la $a0, test_str2
    jal ToLower
    la $a0, test_str2
    jal print_string

    # Test case 3: "AbC"
    la $a0, test_str3
    jal ToLower
    la $a0, test_str3
    jal print_string

    # Exit program
    li $v0, 10
    syscall

ToLower:
```

```
move $t0, $a0      # Copy string address to $t0
li $t2, 0          # Initialize loop counter

loop:
    lb $t1, ($t0)  # Load byte (character)
    beqz $t1, done # If null terminator, exit loop

    # Check if character is uppercase (between 'A' and 'Z')
    blt $t1, 'A', next_char
    bgt $t1, 'Z', next_char

    # Convert to lowercase by adding 32
    addi $t1, $t1, 32
    sb $t1, ($t0)  # Store converted character back

next_char:
    addi $t0, $t0, 1  # Move to next character
    addi $t2, $t2, 1  # Increment counter
    blt $t2, 3, loop  # If counter < 3, continue loop

done:
    jr $ra
```

```
print_string:
   li $v0, 4
   syscall
   la $a0, newline
   syscall
   jr $ra
```

ANSWER TO THE QUESTION NO 2:

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

```
.data
n:     .word 20       # You can change this value to set the range limit
newline: .asciiz "\n"

.text
.globl main

main:
   li $v0, 4          # Print "Prime numbers between 3 and n:"
   la $a0, prompt
   syscall

   li $v0, 5          # Read integer for n
   syscall
```

```
    move $t0, $v0       # Store n in $t0

    li $t1, 3           # Initialize $t1 to 3 (starting number)
loop_check:
    bgt $t1, $t0, end   # If $t1 > n, end the loop

    move $t2, $t1       # Store current number in $t2
    li $t3, 2           # Start divisor at 2
    li $t4, 1           # Assume number is prime (1 = prime, 0 = not prime)

inner_loop:
    mul $t5, $t3, $t3   # Compute $t3 * $t3
    bgt $t5, $t2, print_prime  # If divisor^2 > current number, it's prime

    rem $t6, $t2, $t3   # Compute remainder of $t2 / $t3
    beq $t6, $zero, not_prime  # If remainder is 0, not prime

    addi $t3, $t3, 1    # Increment divisor
    j inner_loop

not_prime:
    li $t4, 0           # Set $t4 to 0 (not prime)
    j continue_loop
```

```
print_prime:
    beq $t4, 0, continue_loop # If not prime, continue loop

    li $v0, 1          # Print prime number
    move $a0, $t1
    syscall

    li $v0, 4          # Print newline
    la $a0, newline
    syscall

continue_loop:
    addi $t1, $t1, 1    # Increment number
    j loop_check

end:
    li $v0, 10         # Exit program
    syscall
.data
prompt: .asciiz "Prime numbers between 3 and n are:\n"
```

ANSWER TO THE QUESTION NO 03

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"
primestr: .asciiz "Prime factors: "

.text
.globl main

main:
   li $v0, 4            # Print the prompt
   la $a0, prompt
   syscall

   li $v0, 5            # Read an integer from the user
   syscall
   move $t0, $v0        # Store the input number in $t0

   li $v0, 4            # Print "Prime factors: "
   la $a0, primestr
   syscall
```

```
    li $t1, 2             # Initialize divisor to 2
    move $t2, $t0         # Store the input number in $t2 for factoring

find_factors:
    beqz $t2, end         # If $t2 is 0, all factors are found
    div $t2, $t1          # Divide $t2 by $t1
    mfhi $t3              # Move remainder to $t3
    mflo $t4              # Move quotient to $t4

    bnez $t3, not_factor  # If remainder is not zero, not a factor
    li $v0, 1             # Print the factor
    move $a0, $t1
    syscall

    li $v0, 4             # Print newline
    la $a0, newline
    syscall

    move $t2, $t4         # Update $t2 to the quotient
    j find_factors        # Continue finding factors

not_factor:
```

```
    addi $t1, $t1, 1       # Increment the divisor
    j find_factors         # Continue finding factors

end:
    li $v0, 10             # Exit program
    syscall
```

ANSWER TO THE QUESTION NO 04

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"
    odd_msg:    .asciiz "The number is odd"

.text
main:
    # Prompt user for input
    li $v0, 4
    la $a0, prompt
    syscall
```

```
    # Read integer
    li $v0, 5
    syscall
    move $t0, $v0  # Store input in $t0

    # Check if even or odd using only srl and sll
    srl $t1, $t0, 1   # Shift right logical by 1 (divide by 2)
    sll $t1, $t1, 1   # Shift left logical by 1 (multiply by 2)

    # If $t1 equals $t0, the number is even
    beq $t0, $t1, even

    # Otherwise, it's odd
    la $a0, odd_msg
    j print_result

even:
    la $a0, even_msg

print_result:
    # Print result message
    li $v0, 4
```

```
syscall


# Exit program
li $v0, 10
syscall
```

ANSWER TO THE QUESTION NO 05

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 1 penny".

```
.data
prompt: .asciiz "Enter a number between 1 and 99: "
quarters_msg: .asciiz " quarters, "
dimes_msg: .asciiz " dimes, "
nickels_msg: .asciiz " nickels, and "
pennies_msg: .asciiz " pennies\n"
result_msg: .asciiz "You need "

.text
.globl main

main:
```

```
# Prompt user for input
li $v0, 4               # Print string syscall
la $a0, prompt          # Load address of prompt
syscall                 # Print prompt

# Read integer input
li $v0, 5               # Read integer syscall
syscall                 # Read integer into $v0
move $t0, $v0           # Move input to $t0

# Calculate number of quarters
li $t1, 25              # Load value of a quarter
div $t0, $t1            # Divide input by 25
mflo $t2                # Get quotient (number of quarters) in $t2
mfhi $t0                # Get remainder in $t0

# Calculate number of dimes
li $t1, 10              # Load value of a dime
div $t0, $t1            # Divide remaining by 10
mflo $t3                # Get quotient (number of dimes) in $t3
mfhi $t0                # Get remainder in $t0

# Calculate number of nickels
```

```
li $t1, 5               # Load value of a nickel
div $t0, $t1            # Divide remaining by 5
mflo $t4                # Get quotient (number of nickels) in $t4
mfhi $t0                # Get remainder in $t0

# Remaining is the number of pennies
move $t5, $t0           # Move remainder to $t5 (number of pennies)

# Print the result message
li $v0, 4               # Print string syscall
la $a0, result_msg      # Load address of result message
syscall                 # Print result message

# Print number of quarters
move $a0, $t2           # Move number of quarters to $a0
li $v0, 1               # Print integer syscall
syscall

# Print quarters message
li $v0, 4               # Print string syscall
la $a0, quarters_msg    # Load address of quarters message
syscall                 # Print quarters message
```

```
# Print number of dimes
move $a0, $t3          # Move number of dimes to $a0
li $v0, 1              # Print integer syscall
syscall

# Print dimes message
li $v0, 4              # Print string syscall
la $a0, dimes_msg      # Load address of dimes message
syscall                # Print dimes message

# Print number of nickels
move $a0, $t4          # Move number of nickels to $a0
li $v0, 1              # Print integer syscall
syscall

# Print nickels message
li $v0, 4              # Print string syscall
la $a0, nickels_msg    # Load address of nickels message
syscall                # Print nickels message

# Print number of pennies
move $a0, $t5          # Move number of pennies to $a0
li $v0, 1              # Print integer syscall
```

```
syscall


# Print pennies message
li $v0, 4               # Print string syscall
la $a0, pennies_msg     # Load address of pennies message
syscall                 # Print pennies message


# Exit program
li $v0, 10              # Exit syscall
syscall
```