Daniela Jiménez (00322800)
Organización de Computadores
CMP 3004
17 / 05 / 2023

Tarea 3

1. **Use assembly to solve the following problems:**

Programa utilizado: Assembly Mars MIPS Simulator

Obtain the 50th Fibonacci number:

```
.data
fib50: .space 8  # variable para almacenar el 50mo número Fibonacci (64-bit)

.text
.globl main
main:
    # Inicializar los primeros 2 numeros de la secuencia Fibonacci
    li $t0, 0
    li $t1, 1

    # Calcular el 50mo numero de Fibonacci
    li $t2, 2
    li $t3, 50

fibonacci_loop:
    addu $t4, $t0, $t1   # Fib(n) = Fib(n-2) + Fib(n-1)
    move $t0, $t1        # Actualizar Fib(n-2)
    move $t1, $t4        # Actualizar Fib(n-1)

    addiu $t2, $t2, 1   # Counter para incremento

    blt $t2, $t3, fibonacci_loop  # Repetir el loop hasta que Counter < 50

    # Almacenar el 50mo numero de Fibonacci
    sd $t1, fib50

    # SALIDA DEL PROGRAMA:
    li $v0, 10
    syscall
```

Find out if a given year is leap:

```
# Entrada o input para el usuario:
li $v0, 4            # Imprimir el string
la $a0, prompt       # Cargar el string
syscall

# Lectura del input del usuario
li $v0, 5            # Leer el entero ingresado (año)
syscall
move $t0, $v0        # Guarda el año ingresado en $t0

# Verifica si el año es divisible para 4 (cada 4 años se da uno bisiesto)
li $t1, 4
rem $t2, $t0, $t1    # Residuo se almacena en $t2

# Verifica si el año es divisible para 100
li $t1, 100
rem $t3, $t0, $t1    # Residuo se almacena en $t3

# Verifica si el año es divisible para 400
li $t1, 400
rem $t4, $t0, $t1    # Residuo se almacena en $t4

# Se revisan todas las CONDICIONES de un año bisiesto
beqz $t2, divisible_by_4  # Si el residuo es 0, ir a divisible_by_4
beqz $t3, divisible_by_100 # Si el residuo es 0, ir a divisible_by_100
beqz $t4, divisible_by_400 # Si el residuo es 0, ir a divisible_by_400

# Si el año dado no es bisiesto:
li $v0, 4            # Imprimir string
la $a0, not_leap_year # Cargar string not_leap_year
syscall

j end_program        # Saltar a end_program

divisible_by_4:
# Año es divisible para 4 pero no para 100 o 400
li $v0, 4            # Imprimir string
la $a0, leap_year    # Se carga el string leap_year
syscall

j end_program        # Saltar a end_program
```

```
divisible_by_100:
# Año es divisible para 100 pero no para 400
li $v0, 4            # Impirmir string
la $a0, not_leap_year # Cargar string not_leap_year
syscall

j end_program        # Saltar a end_program

divisible_by_400:
# Año es divisible para 400
li $v0, 4            # Imprimir string
la $a0, leap_year    # Cargar string leap_year
syscall

end_program:
li $v0, 10           # SALIR DEL PROGRAMA
syscall

.data
prompt: .asciiz "Ingrese un año, por favor: "
leap_year: .asciiz "Su año ingresado SI es bisiesto."
not_leap_year: .asciiz " Su año ingresado NO es bisiesto."
```

Given an array of integers, calculate the average:

```
.data
array:     .word   85, 120, 145, 75, 60  # Arreglo predefinido de numeros enteros
size:      .word   5             # tamaño del arreglo predefinido antes ^
average:   .word   0             # Variable para almacenar el promedio del arreglo

.text
.globl main
main:
   la $t0, array         # Cargar la dirección del arreglo en $t0
   lw $t1, size          # Cargar el tamaño del arreglo en $t1
   la $t2, average       # Crargar la dirección de la variable del promedio en $t2

   li $t3, 0             # Inicializa la suma a 0
   li $t4, 0             # Inicializa el index a 0

loop:
   beq $t4, $t1, calculate_average   # Si index == tamaño, ir a calculate_average
   lw $t5, 0($t0)        # Cargar el elemento actual del arreglo en $t5
```

```
    add $t3, $t3, $t5      # Anadir el elemento actual a la suma
    addi $t4, $t4, 1       # Incrementar el index
    addi $t0, $t0, 4       # Incrementar el puntero del arreglo en 4 bytes
    j loop

calculate_average:
    div $t3, $t1           # Dividir la suma total para el tamaño del arreglo
    mflo $t3               # Mover el cociente (PROMEDIO) a $t3
    sw $t3, 0($t2)         # Se almacena el promedio en memoria

    # Imprimir el valor del promedio resultante
    li $v0, 1              # syscall para imprimir un entero
    lw $a0, average        # Cargar el valor del promedio para que sea impreso
    syscall

    li $v0, 10             # SALIR DEL PROGRAMA
    syscall
```

Write a program to transform from Celsius to Fahrenheit and vice versa:

```
.data
    celsius_input: .asciiz "Ingrese temperatura en CELCIUS: "
    fahrenheit_input: .asciiz "Ingrese temperatura en FARENHEIT: "
    temp_final: .asciiz "TEMPERATURA TRANSFORMADA = "
    newline: .asciiz "\n"

.text
.globl main

main:
    # Ingreso de temperatura Celsius del usuario
    li $v0, 4
    la $a0, celsius_input
    syscall

    # Leer temperatra en Celsius del usuario
    li $v0, 5
    syscall
    move $t0, $v0  # Almacenar temperatura en Celsius

    # CONVERSIÓN DE CELCIUS A FARENHEIT
    mul $t1, $t0, 9
    div $t1, $t1, 5
```

```
addi $t1, $t1, 32

# Imprimir el resultado de la conversión de temperatura
li $v0, 4
la $a0, temp_final
syscall

li $v0, 1
move $a0, $t1
syscall

li $v0, 4
la $a0, newline
syscall

# Ingreso de temperatura Farenheit del usuario
li $v0, 4
la $a0, fahrenheit_input
syscall

# leer temperatura en Farhenit del usuario
li $v0, 5
syscall
move $t0, $v0  # Almacenar temperatura en Fahrenheit

# CONVERSIÓN DE FARENHEIT A CELCIUS
subi $t0, $t0, 32
mul $t1, $t0, 5
div $t1, $t1, 9

# Imprimir el resultado de la conversión de temperatura:
li $v0, 4
la $a0, temp_final
syscall

li $v0, 1
move $a0, $t1
syscall

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

```
# SALIR DEL PROGRAMA
li $v0, 10
syscall
```

2. **For the following memory space, what would it look like after executing the assembly code below?:**

Address	Contents
10	1
11	4
12	5
13	112
14	7

```
LOAD 14
ADD (12)
STORE 12
```

- Con la instrucción de "**LOAD**" se carga el valor de la dirección de memoria (o address) 14, que es igual a 7 (en Contents).

- Con la instrucción de "**ADD**" se suma el valor de la dirección de memoria (o address) 12 (que es igual a 5) al valor que se cargó anteriormente (que era igual a 7).
  Por lo tanto: 7+5 = ***12***

- Con la instrucción de "**STORE**" se almacena el valor obtenido anteriormente (que es igual a 12) en la dirección de memoria (o address) 12.
  Debido a esto, después de ejecutar el código, el valor de ***Address 12*** pasa a ser ***"12"*** (en ***Contents***).

Address	Contents
10	1
11	4
12	12
13	112
14	7

3. **Implement a function named "abs_diff" that calculates the absolute value of the difference of two inputs a and b (i.e., |a-b|), and get the assembly code output.**

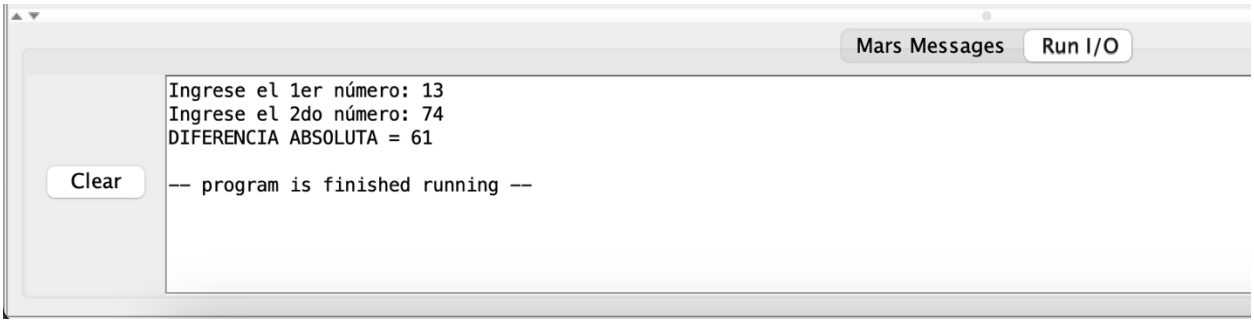
```
# Cálculo del valor absoluto de la diferencia entre 2 números "a" y "b" -> |a-b|

.data
```

```
prompt1: .asciiz "Ingrese el 1er número: "
prompt2: .asciiz "Ingrese el 2do número: "
result: .asciiz "DIFERENCIA ABSOLUTA = "
newline: .asciiz "\n"

.text
.globl main

main:
  # Ingresar el 1er número del usuario
  li $v0, 4
  la $a0, prompt1
  syscall

  # Leer el primer número
  li $v0, 5
  syscall
  move $s0, $v0 # Almacenar el primer número en $s0

  # Ingresar el 2do número del usuario
  li $v0, 4
  la $a0, prompt2
  syscall

  # Leer el 2do número
  li $v0, 5
  syscall
  move $s1, $v0 # Almacenar el 2do número en $s1

  # LLAMADA A LA FUNCIÓN "abs_diff" :
  move $a0, $s0
  move $a1, $s1
  jal abs_diff
  move $s2, $v0  # Almacena el resultado en $s2

  # Muestra el resultado
  li $v0, 4
  la $a0, result
  syscall

  move $a0, $s2
  li $v0, 1
  syscall
```

```
  # Se imprime una nueva línea
  li $v0, 4
  la $a0, newline
  syscall

  # Salida del programa
  li $v0, 10
  syscall

# FUNCIÓN "abs_diff" PARA CALCULAR LA DIFERENCIA ABSOLUTA
abs_diff:
  sub $t0, $a0, $a1
  bgez $t0, skip_negate
  neg $t0, $t0

skip_negate: # Omite negación de la diferencia si el valor en el registro $t0 es > ó = a 0
  move $v0, $t0
  jr $ra
```

Assembly Code Output:

EJEMPLO: Dado ***a = 13*** y ***b = 74***



4. **What are the differences among sequential access, direct access, and random access?**

Sequential Access

- La memoria está organizada en registros, que son unidades de data o información.
- El acceso debe realizarse en una secuencia lineal específica.
- La información de direccionamiento almacenada se utiliza para separar registros y también para ayudar con la recuperación de estos.
- Se utiliza un mecanismo compartido de lectura - escritura.

- El tiempo para acceder a un registro arbitrario es muy variable.
- *Ejemplo de donde se encuentra este tipo de acceso:* unidades de cinta (tape units).

Direct Access

- Similar al acceso secuencial, el acceso directo implica un mecanismo compartido de lectura - escritura.
- Los bloques o registros individuales tienen direcciones únicas basadas en la ubicación física.
- El acceso se logra llegando directamente a una vecindad general y luego realizando una búsqueda secuencial, contando, o esperando llegar a la ubicación final.
- El tiempo de acceso es variable.
- *Ejemplo de donde se encuentra este tipo de acceso:* unidades de disco (disk units).

Random Access

- Cada ubicación direccionable en la memoria tiene un mecanismo de direccionamiento único y físicamente conectado.
- El tiempo de acceso a una ubicación determinada es independiente de la secuencia de accesos anteriores y se mantiene constante.
- Se puede seleccionar cualquier ubicación al azar, dirigirse directamente y acceder a ella.
- *Ejemplos de donde se encuentra este tipo de acceso*: memoria principal (main memory) y ciertos sistemas de caché.

5. What common characteristics are shared by all RAID levels?

*Algunas características en común o que comparten todos los niveles de **RAID** son:*

• El sistema operativo ve al RAID en general como un conjunto de unidades de disco físico (physical disk drives), a su vez, el conjunto de unidades de disco físico (set of physical disk drive) se conoce como una unidad lógica.

• El esquema de eliminación (stripping scheme) se utiliza para distribuir los datos en un arreglo de unidades físicas.

• En todos los niveles de RAID, se utiliza un arreglo de disco (array of disk) que funciona de forma independiente.

• La información de paridad se almacena mediante capacidad de disco redundante.

• En todos los niveles de RAID, para recuperar los datos en caso de falla del disco, se usa información de paridad.

• Todos abordan la necesidad de redundancia de forma eficiente.

Bibliografía y fuentes de consulta:

Computer Organization and Architecture: Design for Performance, 9th Edition, William Stallings, Pearson Education, 2013.