

Simulacion y optimizacion de un programa en un procesador escalar segmentado

David Bocanegra

Laura Barona

Alfonso Narvaez

Universidad Internacional de la Rioja

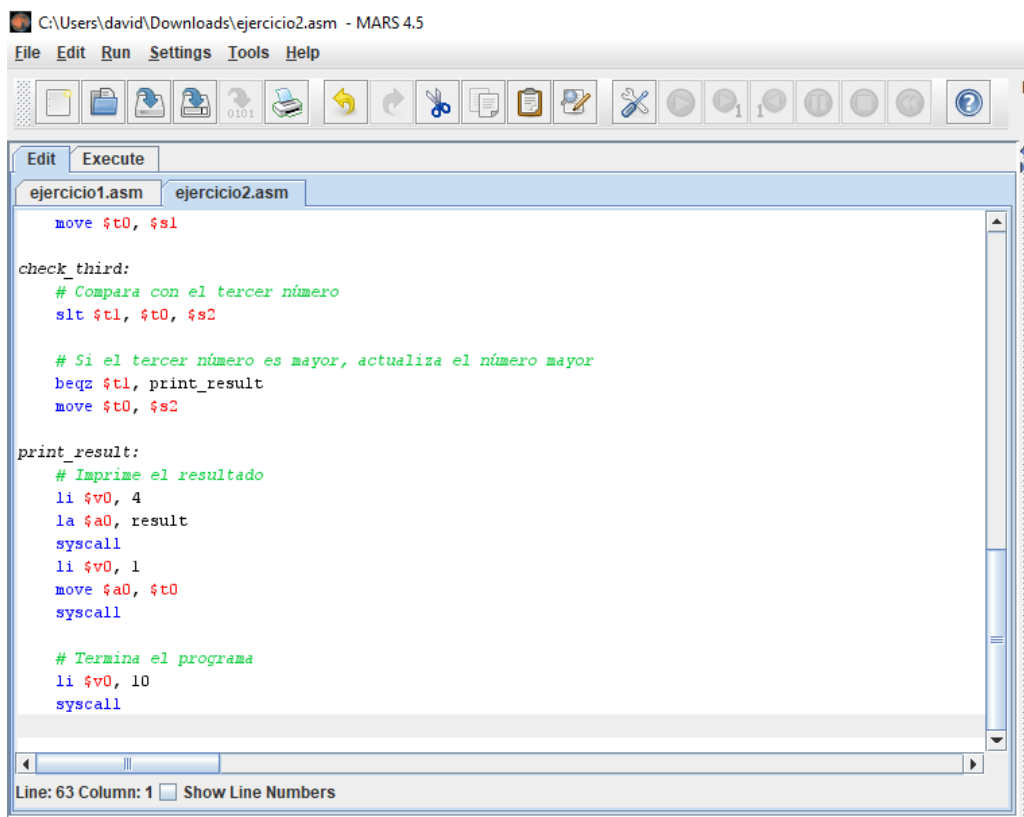
Javier Diaz Diaz

11/septiembre/2023

Documentació código assembler.

Ejercicio 1: Número mayor

En este código podremos encontrar el número mayor entre tres números ingresados por el usuario, el código se realizó en el lenguaje de assembler. En donde el programa solicita al usuario ingresar tres números y luego determina cuál de ellos es el mayor, en la siguiente imagen vamos a poder ver el código base, sin compilar



The screenshot shows the MARS 4.5 IDE with the file "ejercicio2.asm" open. The code is written in MIPS assembly and is as follows:

```
move $t0, $s1

check_third:
    # Compara con el tercer número
    slt $t1, $t0, $s2

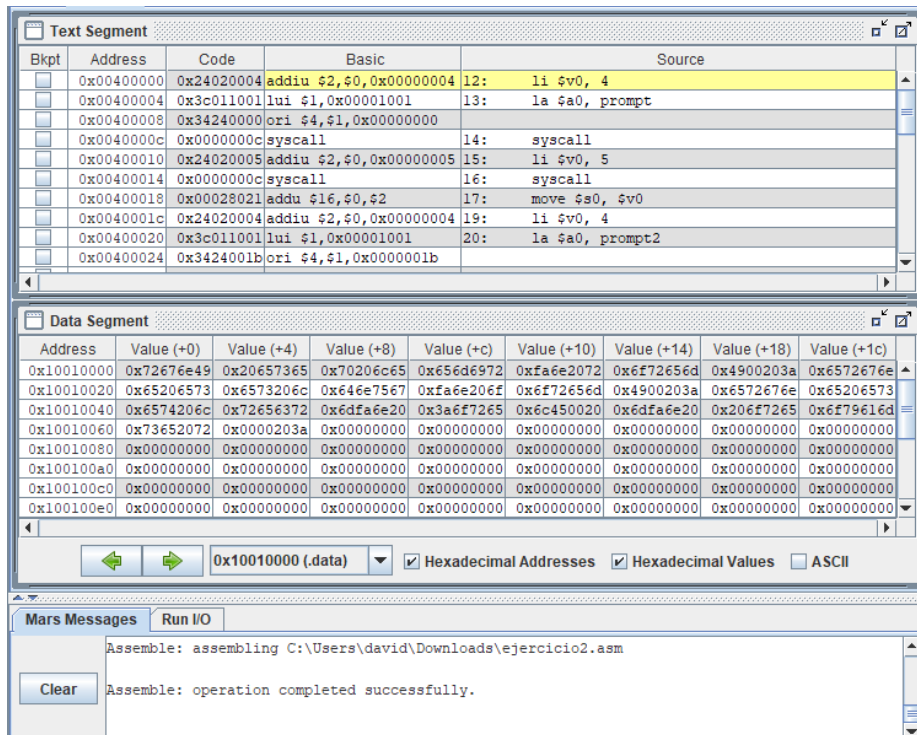
    # Si el tercer número es mayor, actualiza el número mayor
    beqz $t1, print_result
    move $t0, $s2

print_result:
    # Imprime el resultado
    li $v0, 4
    la $a0, result
    syscall
    li $v0, 1
    move $a0, $t0
    syscall

    # Termina el programa
    li $v0, 10
    syscall
```

The status bar at the bottom indicates "Line: 63 Column: 1" and has a checkbox for "Show Line Numbers".

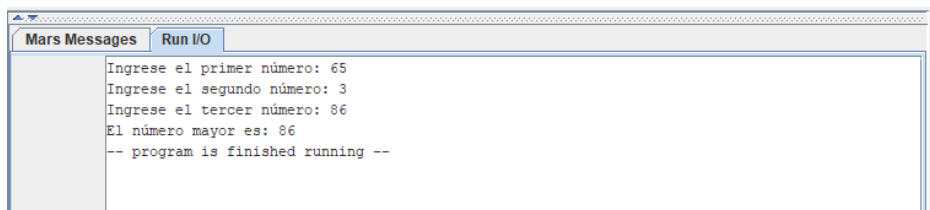
Ahora vamos a poder observar el código después de la compilación en donde nos indica que todo se encuentra en un estado successfully



En donde al indicar esto poder decir que lo que el código está haciendo es:

1. Solicitar al usuario ingresar tres números
2. Comparar esos tres número para determinar cuál es el mayor
3. Mostrar el número mayor pon pantalla

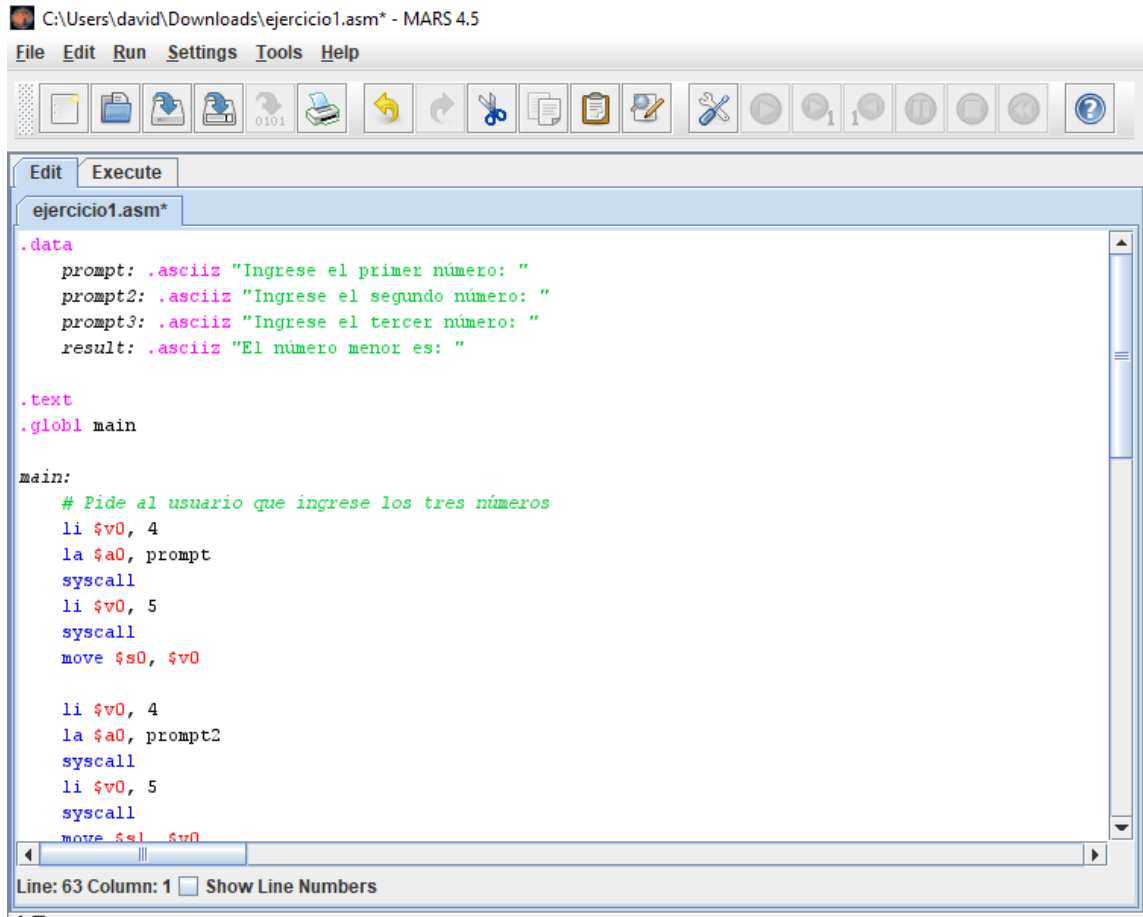
Lo cual nos daría como resultado lo siguiente



Ejercicio 2: Número menor

En este código podemos encontrar un ejemplo que demuestra cómo encontrar el número menor entre tres números ingresados por el usuario en lenguaje assembler.

La función del programa va a ser solicitar al usuario ingresar tres números y luego determina cuál de ellos es el número menor.



The screenshot shows the MARS 4.5 IDE with the file 'ejercicio1.asm' open. The code is as follows:

```
.data
prompt: .asciiz "Ingrese el primer número: "
prompt2: .asciiz "Ingrese el segundo número: "
prompt3: .asciiz "Ingrese el tercer número: "
result: .asciiz "El número menor es: "

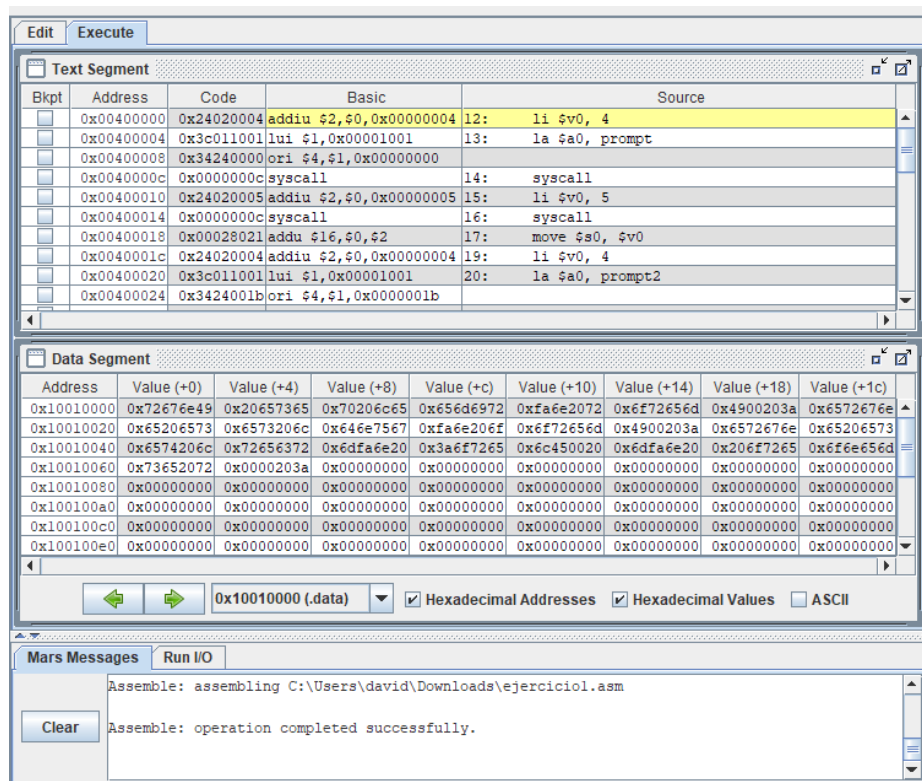
.text
.globl main

main:
    # Pide al usuario que ingrese los tres números
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    move $s0, $v0

    li $v0, 4
    la $a0, prompt2
    syscall
    li $v0, 5
    syscall
    move $s1, $v0
```

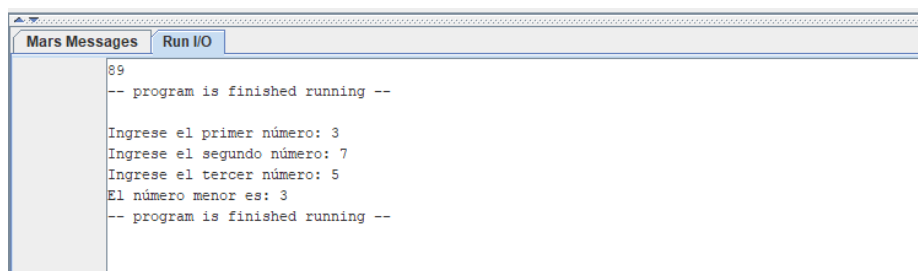
The status bar at the bottom indicates 'Line: 63 Column: 1' and has a checkbox for 'Show Line Numbers'.

Ahora vamos a poder observar el código después de la compilación en donde nos indica que todo se encuentra en un estado successfully.



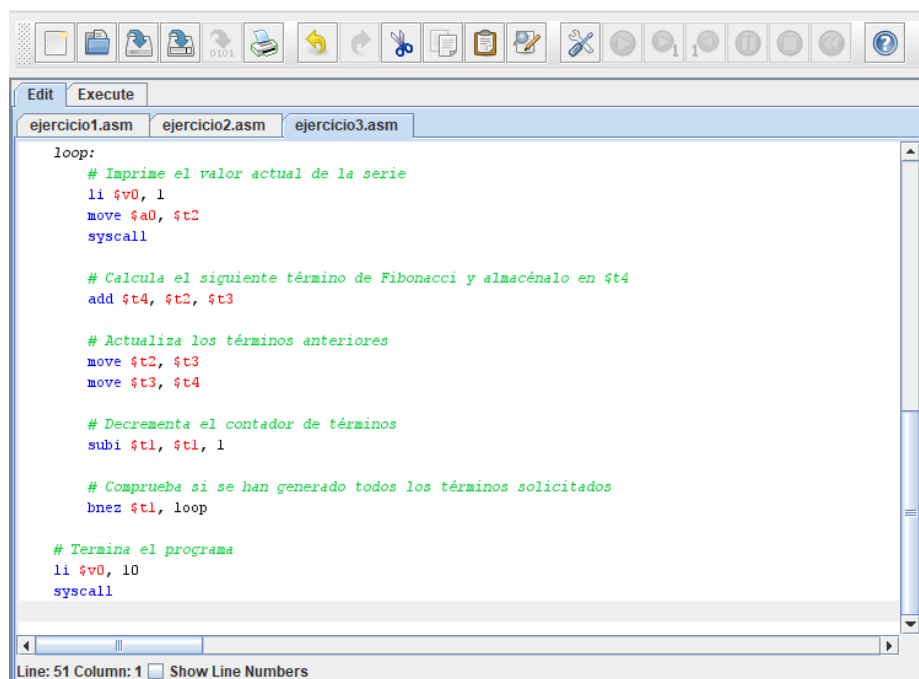
Al momento en que el código nos indica eso nos está informando que no tenemos ningún tipo de error y que además de esto los pasos que va a ejecutar son los siguientes:

- Solicita al usuario ingresar tres números.
- Comparar estos números para determinar cuál es el menor.
- Muestra el número menor en la pantalla



Ejercicio 3: Serie Fibonacci

En este código podemos encontrar un ejemplo que muestra cómo calcular y mostrar la serie de Fibonacci en lenguaje Assembler. En donde podemos decir que la serie de Fibonacci es una secuencia de números en la que cada número es la suma de los dos números anteriores, comenzando desde 0 y 1.



```
loop:
    # Imprime el valor actual de la serie
    li $v0, 1
    move $a0, $t2
    syscall

    # Calcula el siguiente término de Fibonacci y almacénalo en $t4
    add $t4, $t2, $t3

    # Actualiza los términos anteriores
    move $t2, $t3
    move $t3, $t4

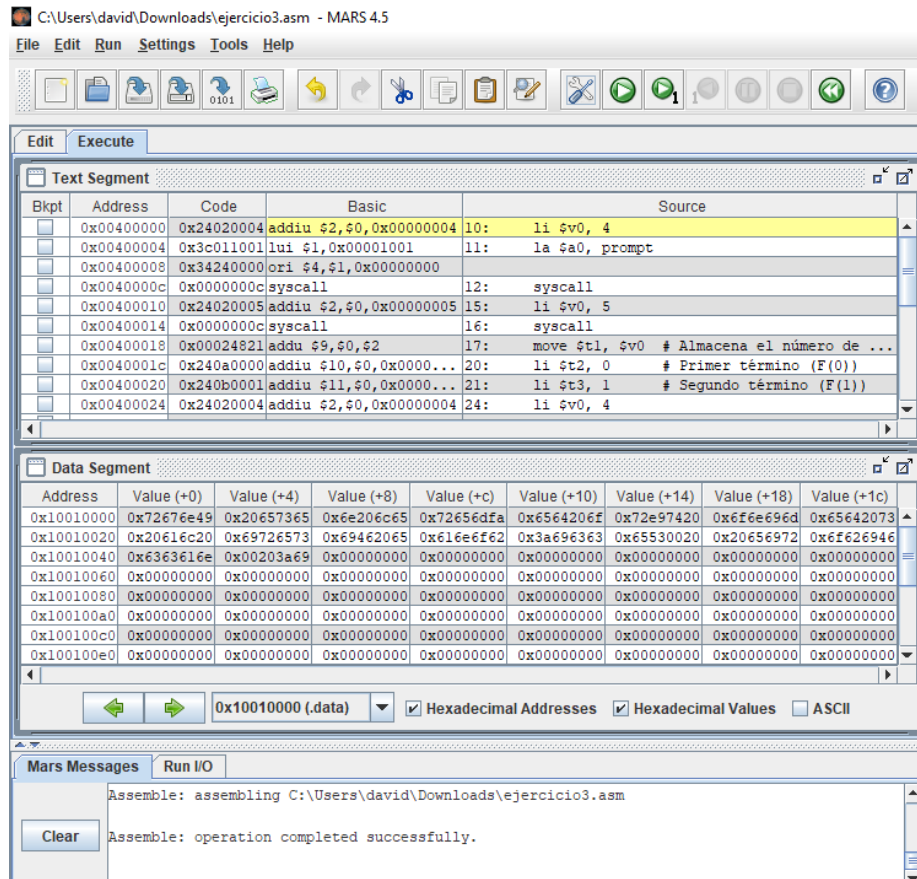
    # Decrementa el contador de términos
    subi $t1, $t1, 1

    # Comprueba si se han generado todos los términos solicitados
    bnez $t1, loop

# Termina el programa
li $v0, 10
syscall
```

Line: 51 Column: 1 ☐ Show Line Numbers

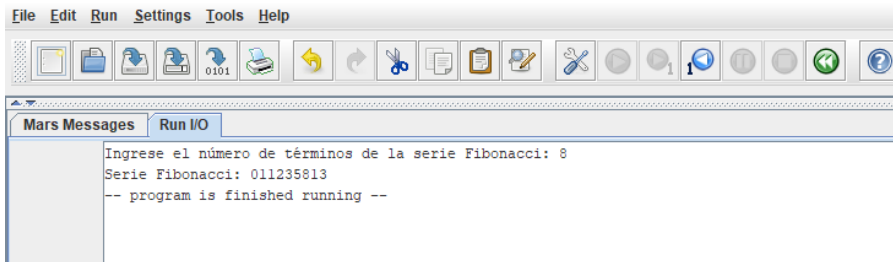
Ahora vamos a poder observar el código después de la compilación en donde nos indica que todo se encuentra en un estado successfully.



Y después de esto vamos a poder observar que el procedimiento que este código va a realizar es el siguiente:

- Solicita al usuario ingresar el número de términos de la serie Fibonacci que desea generar.
- Inicializa los primeros dos términos de la serie en 0 y 1.
- Utiliza un bucle para calcular y mostrar los términos de la serie Fibonacci uno por uno.

- Muestra cada término en la pantalla a medida que se calcula.
- El bucle continúa hasta que se hayan generado todos los términos solicitados.
- Finaliza el programa.



Repositorios GitHub

Para poder evidenciar el código completo de los 3 ejercicios se van a poder evidenciar en cualquiera de los siguientes repositorios GitHub ya que al ser un poco extensos se puede tornar un poco tedioso ver solo la imagen por este medio.

- <https://github.com/AlfonsoNarvaezRada/EstructuraDeComputadores.git>
- <https://github.com/LalaBarona45/unir.git>
- <https://github.com/oscardbocanegra/Universidad/tree/main/estructuraComputadores>

Conclusiones

- El código MIPS para calcular la serie de Fibonacci es un ejemplo ilustrativo de cómo se pueden utilizar las capacidades de programación en lenguaje ensamblador para generar una secuencia matemática conocida.
- El código MIPS para encontrar el número mayor entre tres números ingresados por el usuario es un ejemplo práctico de cómo se pueden realizar comparaciones condicionales en lenguaje ensamblador para tomar decisiones basadas en datos de entrada.
- El código MIPS para encontrar el número menor entre tres números ingresados por el usuario es otro ejemplo práctico de cómo se pueden realizar comparaciones condicionales en lenguaje ensamblador.