Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Organización y Estructura del Computador 2

LABORATORIO 01 ORGA 2

RISC-V y Venus

Docente:

Prof. Jaime

Alumno:

Pérez Touceda, Isaac CI - 31065844

Caracas 12 de mayo de 2025

Introducción

En este informe vamos a experimentar todas las opciones de Risc-V y su editor online Venus, que cuenta con un compilador y depurador del mismo. Documentaremos todo el proceso explicando paso a paso el proceso con las observaciones pertinentes.

Siguiendo el siguiente código en Risc-V:

ej1_hola.s

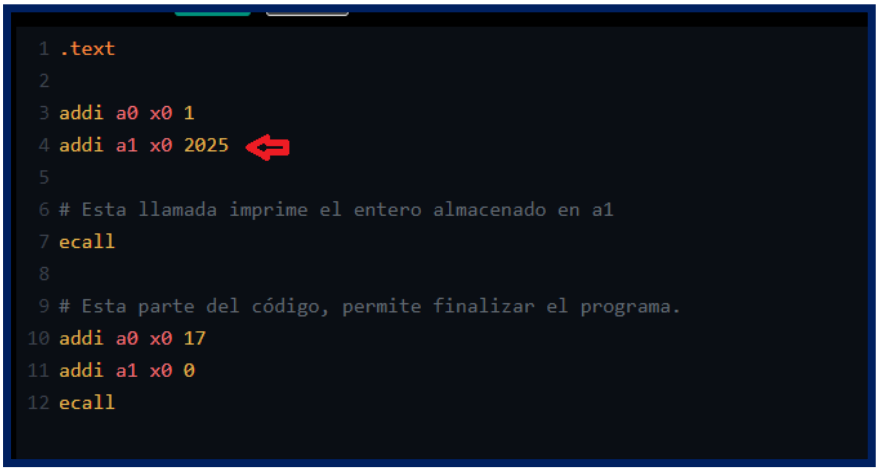
```
1 .text
2
3 addi a0 x0 1
4 addi a1 x0 1234
5
6 # Esta llamda imprime el entero almacenado en a1
7 ecall
8
9 # Esta parte del código, permite finalizar el programa.
10 addi a0 x0 17
11 addi a1 x0 0
12 ecall
```

Se pide que modifique dicho código para que la salida imprima el valor 2025.

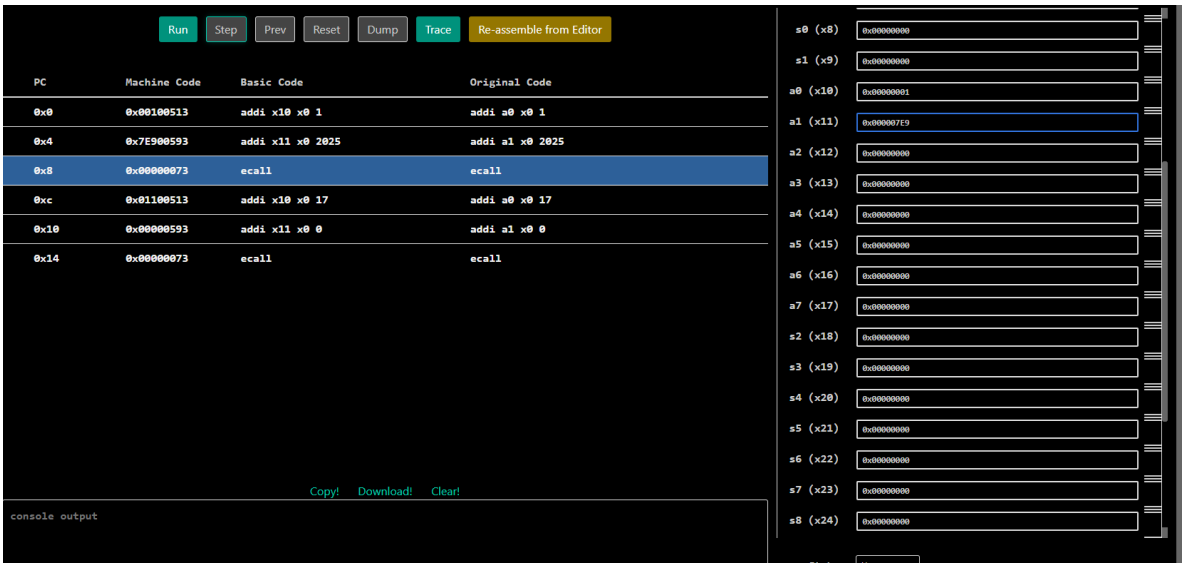
Observaciones:

Me llamo la atención el funcionamiento de ecall el cual hace que una llamada al sistema (syscall), permitiendo que un programa en modo usuario solicite servicios del sistema operativo (o del entorno de ejecución, como Venus). Es decir, permite que un programa interactúe con el entorno externo (como imprimir en consola, leer entrada, terminar la ejecución, etc.), y sí, puede usarse para "ver" el contenido de un registro (por ejemplo, imprimiéndolo en pantalla).

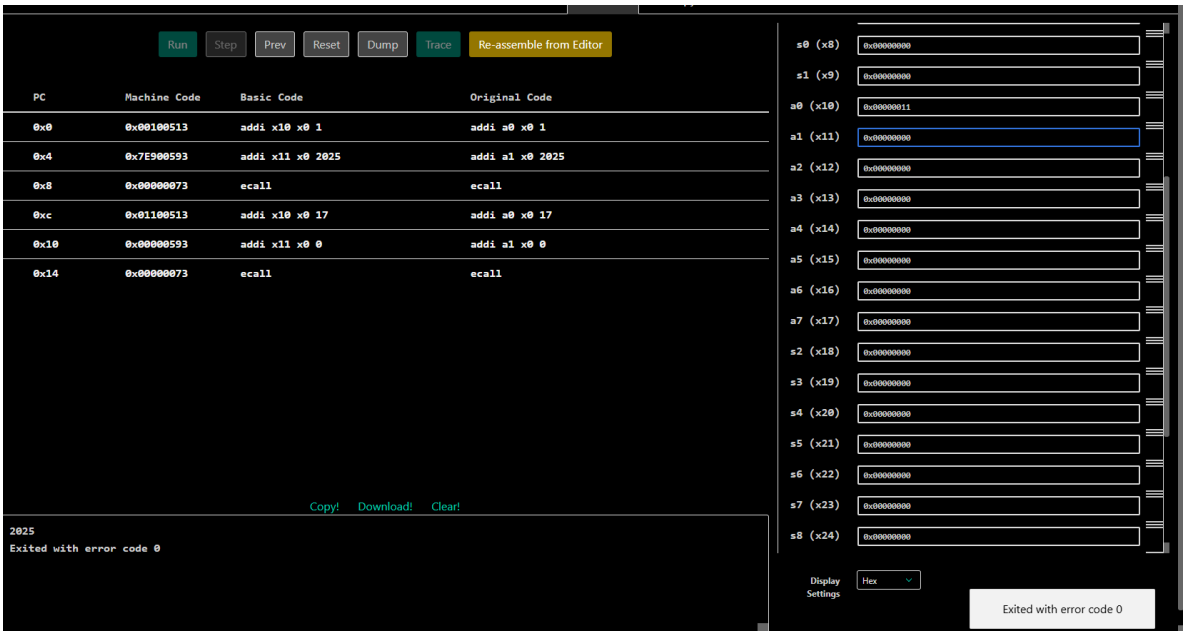
Por tanto, la modificación que se tiene que realizar es cambiar el inmediato que se carga en a1 por la instrucción “addi”:



Al realizar este cambio cuando se invoca el ecall se imprime el valor cargando en el registro a1 el cual es 2025. Cumpliendo así el objetivo del experimento.



Al ejecutar las 2 primeras instrucciones se observa como en el registro a1 se carga el valor de 2025.



La salida mostrada fue la solicitada en el enunciado, “2025”.

Posteriormente se realiza una secuencia de preguntas relacionadas al código del Fibonacci (fib.s). Dichas respuestas se encuentran en el txt referente. De todas maneras, agregare las preguntas por acá para completar el informe de la mejor manera:

Pregunta 1: ¿Cuál es el código máquina de la instrucción resaltada? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: En la instrucción resaltada con el PC indicando a la dirección 0x00 hay un código de máquina de 32 bits el cual es --> 0x000002B3

Pregunta 2: ¿Cuál es el código máquina de la instrucción en la dirección 0x34? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: En la instrucción 0x34 se encuentra el código de maquina --> 0x00000073

Pregunta 3: ¿Cuál es el valor del registro sp? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: El valor que se encuentra en el registro sp es --> 0x7FFFFFDC

Pregunta 4: ¿Cuál es el nuevo valor del registro t1? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: El valor que se encuentra en el registro t1 con el prefijo 0x es --> 0x00000001 (Esto sucede porque la instrucción ejecutada anteriormente carga en el registro t1 el valor inmediato de 1)

Pregunta 5: ¿Cuál es el código máquina de la instrucción actual? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: La instrucción actual a la que apunta el contador de programa tiene el código de máquina de 0x10000E17

Pregunta 6: ¿Cuál es el valor del registro t3? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: El valor del registro t3 en la dirección actual del PC 0x10 es de 0x10000000 posterior a ejecutar el la (load addres) de n al registro t3

Pregunta 7: ¿Cuál es el byte al que apunta t3? La respuesta debe ser un número hexadecimal de 8 bits (1 byte), con el prefijo 0x: t3 tiene la dirección de memoria de n. Por tanto, al buscar en memoria esta dirección de memoria encontramos un valor. Dicho valor es 0x0C

Pregunta 8: ¿Cuál es el valor del registro t0? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: El valor del registro t0 es de --> 0x00000001

Pregunta 9: ¿Cuál es el nuevo valor del registro t0? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: El valor del registro t0 es de --> 0x0000000D

Pregunta 10: ¿Cuál es el valor del registro t0 en decimal? La respuesta debe ser un número decimal sin prefijo: El valor decimal es 13

Pregunta 11: ¿Cuál es la salida del programa? La respuesta debe ser un número decimal sin prefijo: La salida del programa es el número 144. El cuál es el termino 12 de la sucesión de Fibonacci.

Parte A: Imprimir letras (‘A’–‘Z’)

Objetivo: crear un programa RISC-V que recorra los códigos ASCII de 65 a 90 e imprima cada valor como un carácter mayúsculo.

La salida de este código de Risc-V es → A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```
.data
space: .asciiz " "        # separador en blanco
   .text
   .globl _start
_start:
   # 1) Inicializar ASCII 'A'
   li   t0, 65           # t0 = 65 ('A')
loop:
   # 2) Imprimir carácter
   li   a0, 11           # ecall print_char ①
   mv   a1, t0           # carácter a imprimir
   ecall                 # invoca print_char
   # 3) Imprimir espacio
   li   a0, 4            # ecall print_str
   la   a1, space
   ecall                 # invoca print_str
   # 4) Incrementar y comprobar fin de bucle
   addi t0, t0, 1        # siguiente código ASCII
   li   t1, 91           # 'Z' + 1 = 91
   blt  t0, t1, loop     # mientras t0 < 91, repetir bucle
   # 5) Salida limpia
   li   a0, 10           # ecall exit
   ecall                 # invoca exit
```

Parte B: Imprimir códigos numéricos (65–90) con espacio

Objetivo: reutilizar la misma lógica (el registro t0 recorre 65–90) pero, en lugar de print_char, usar print_int para interpretar cada valor como un entero y luego separar por espacios.

```
.data
space: .asciiz " "        # separador en blanco
   .text
   .globl _start
_start:
   # 1) Inicializar valor 65
   li   t0, 65           # t0 = 65
loop:
   # 2) Imprimir entero
   li   a0, 1            # ecall print_int ①
   mv   a1, t0           # entero a imprimir
   ecall                 # invoca print_int
   # 3) Imprimir espacio
   li   a0, 4            # ecall print_str
   la   a1, space
   ecall                 # invoca print_str
   # 4) Incrementar y comprobar fin de bucle
   addi t0, t0, 1        # siguiente valor
   li   t1, 91           # 90 + 1 = 91
   blt  t0, t1, loop     # mientras t0 < 91, repetir bucle
   # 5) Salida limpia
   li   a0, 10           # ecall exit
   ecall                 # invoca exit
```

La salida de este código es de:

65 66 67 68 69 70 71 72 73 74
75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90

Reflexión

• **¿Qué observa en la misma secuencia de valores?:** el código en esencia es lo mismo e incluso el valor inmediato que se almacena en el registro t0 es el mismo en ambos códigos, el 65 y va incrementando de 1 en 1 hasta el 90. Lo que cambia es al usar el codigo especial para que el syscall lo interprete de una manera u otra.

• **¿Cómo cambia el resultado cuando usas print_char vs. print_int?:** cambia la interpretación del hexadecimal que se va a mostrar por consola gracias al código que lee el ecall interpretado por el syscall. Eso hace que el 65(en Hex) sea A o simplemente sea un 65 en entero.

• **¿Qué nos enseña esto sobre la abstracción de registros y llamadas al sistema en RISC-V?:** Los registros de 32 bits de Risc-V solo almacenan 1 y 0, pues almacenan un valor hexadecimal de 32 bits y lo que cambia son las llamadas al sistema que dependiendo el código que se le pase como argumento al ecall el sistema interpretara estos hexadecimales de la manera indicada.

Ejercicio 3: Uso de Memcheck: se respondieron numerosas preguntas sobre el ejercicio de MemCheck. Estas preguntas se encuentran en el archivo que está en esta misma carpeta, pero los resúmenes de las mismas para poner un contexto al informe son:

Pregunta 1: ¿Qué dirección intentó acceder el programa, pero causó el error? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: La dirección a la que intento acceder el programa, pero causo un error es --> 0x10008058

Pregunta 2: ¿Cuántos bytes intentaba acceder el programa? La respuesta debe ser un número decimal sin prefijo: Intento acceder a 4 bytes en la dirección previamente mencionada

Pregunta 3: ¿Qué dirección intentó acceder el programa, pero causó el error? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: Intento acceder a --> 0x10008080

Pregunta 4: ¿Cuántos bytes fueron asignados en el bloque relacionado con el error? La respuesta debe ser un número sin unidades: Un bloque de tamaño 40.

Pregunta 5: ¿Qué línea del archivo fuente causó este error? La respuesta debe ser un número: Línea 18

Pregunta 6: ¿Cuál es el valor de t1 según el mensaje de error de memcheck? La respuesta debe ser un número decimal: El valor de t1 según el mensaje de error en memcheck es de 10

Pregunta 7: ¿Cuántos bytes no fueron liberados cuando el programa terminó? La respuesta debe ser un número decimal sin unidades: 40 bytes no fueron liberados al terminar el programa

Pregunta 8: ¿Cuál es la dirección del bloque que no se liberó? La respuesta debe ser un número hexadecimal de 32 bits, con el prefijo 0x: La dirección del bloque que no se libero es --> 0x10008058

Además, al final se manda a corregir el problema que está teniendo el código de Risc-V. Por aquí anexo el código corregido

```
.import utils.s

.text
main:
    # Este programa llenarÃ¡ un arreglo de tamaÃ±o 10 con 0.

    # Asignar una matriz de tamaÃ±o 10
    li a0 40   # 10 ints, 4 bytes cada uno
    jal malloc # malloc esta definifo en utils.s
    mv t0 a0   # el apuntador se devuelve en a0

    # Llene la matriz con 0
    li t1 0  # t1 es el indice
    li t2 10 # t2 es el tamaÃ±o del arreglo

loop:
    # Almacenar 0 en el Ã•ndice actual
    sw x0 0(t0)
    # Incrementar el indice
    addi t1 t1 1
    # Incrementar el apuntador
    addi t0 t0 4
    # Comprueba si hemos terminado
    # Si no, ir a loop
    blt t1, t2 loop

    # Salir del programa
    jal free
    li a0 0
    jal exit
```

```
Exited with error code 0
```

Con el Memcheck activado. Por lo tanto, no existen fugas de memoria ni problemas de acceso.

EJERCICIO 4 → Realizando el reporte del ejercicio 4.

He sido capaz de realizar una función que al ser utilizada en ej4_discreta_fn_tester arroja el resultado que se requiere. Considera la función de valores discretos f definida en los enteros del conjunto {-3, -2, -1, 0, 1, 2, 3}. Aquí está la definición de la función:

```
f(-3) = 6
f(-2) = 61
f(-1) = 17
f(0) = -38
f(1) = 19
f(2) = 42
f(3) = 5
```

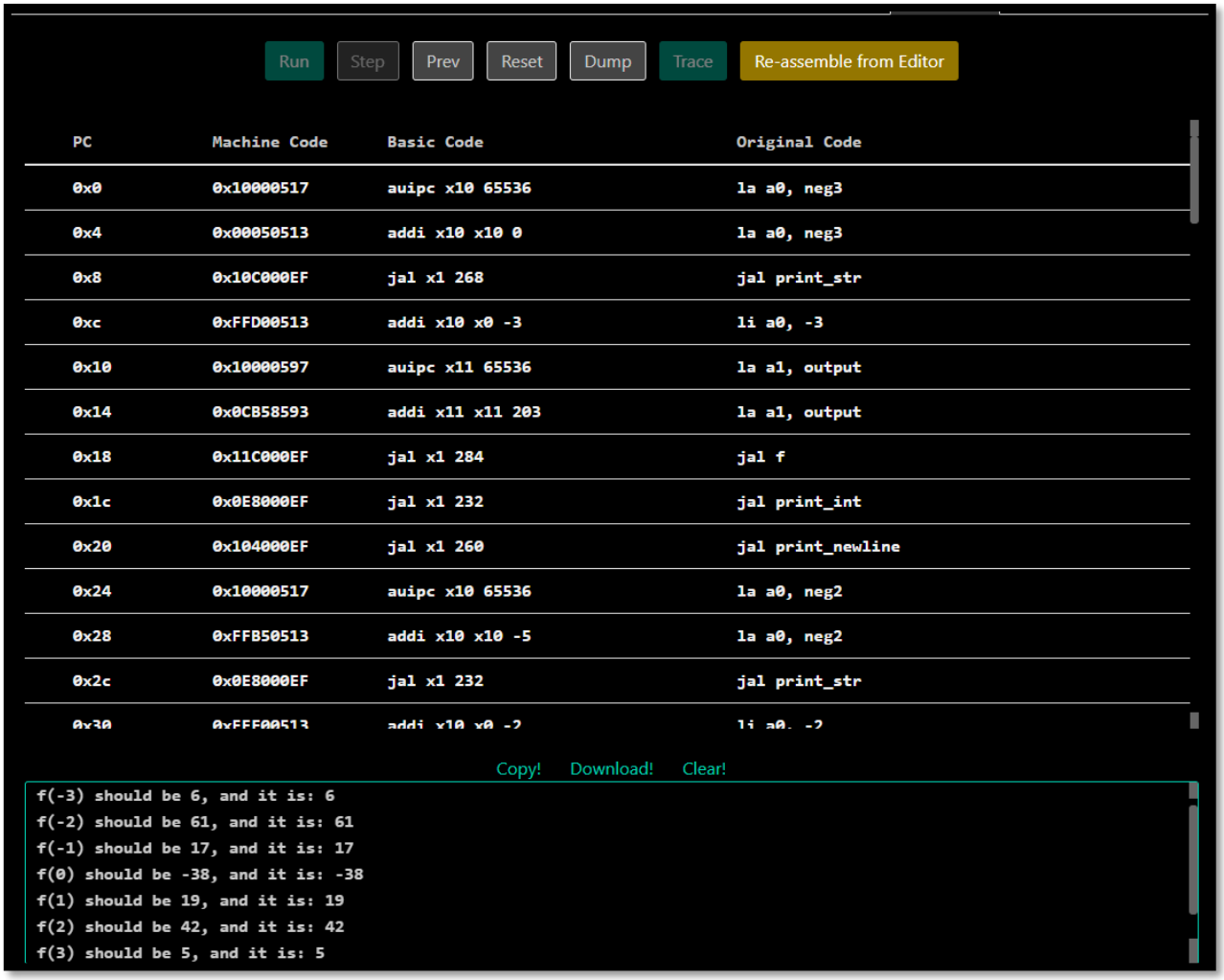
El código creado que permitió que esto fuese posible:

Active File: /ej4_discreta_fn.s Save Close

```
.globl f

.text
f: # Funcion que se llamara desde un codigo Risc-V
mv t0, a0    #Movemos el -3 que se pasa por argumento a t0
mv t1, a1    #Movemos el puntero a la direccion del arreglo a t1
addi t0, t0, 3         #Hacemos (Numero a evaluar) + 3(Max del arreglo) = indice
slli t0, t0, 2         #indice x 4 2(elevado a la 2) sera igual al desplazamiento para acceder a dicho indice
add t1, t1, t0         #Sumamos el desplazamiento necesario a la direccion base del arreglo
lw a0, 0(t1)           #Cargamos el elemento requerido de ese indice del arreglo
jr ra                  #Terminamos la funcion
```

Y la salida esperada fue resuelta con éxito →



Ejercicio 5: Factorial

Asegúrate de que memcheck esté deshabilitado para este ejercicio. En este ejercicio, implementarás la función factorial en RISC-V. ¡Esta función recibe un parámetro entero n y devuelve n! El esqueleto de esta función (archivo ej5_factorial. s) es el siguiente:

EJERCICIO 5 Consiste en realizar una función que calcule el factorial de cualquier numero N

Se hicieron y se realizaron todas las pruebas pertinentes respecto al código que realice y funciona perfectamente para cualquier numero N. Por lo tanto, el código es el siguiente:

```
.globl factorial
.data
 n: .word 8
 .text

 main:
    la t0, n
    lw a0, 0(t0)
    jal ra, factorial
    addi a1, a0, 0
    addi a0, x0, 1
    ecall # Imprimir Resultado
    addi a1, x0, '\n'
    addi a0, x0, 11
    ecall # Imprimir newline
    addi a0, x0, 10
    ecall # Exit
 # factorial toma un argumento:
 # a0 contiene el número del cual queremos calcular el factorial
 # El valor de retorno debe almacenarse en a0
 factorial:
    mv t0, a0 #Movemos n a t0
    addi t4, t4, 1 #Inicializamos el factorial = 0
    addi t1, t1, 1 #Iterador externo = 1
    loop:
    beq t1, t0, finalizar #Si iterator externo es igual a N, entonces terminamos
    addi t2, t1, 1 #Veces a multplicar el t4(t4 = factorial calculado de manera iterativa)
    addi t3, x0, 1 #Iterador del bucle interno = 1
    mv t5, t4 #Copiamos el valor que se va a sumar

    loop_mult:  #Bucle para multiplicar
    add t4, t4, t5  #Sumamos varias veces el mismo numero (multiplicar)
    addi t3, t3, 1  #Aumentamos el iterador en 1 (t3)
    blt t3, t2 loop_mult #Mientras t3(iterador) sea menor que t2 continuamos multiplicando
    addi t1, t1, 1 #Aumentamos el iterador del bucle externo
    j loop #Continuamos el bucle

    finalizar:
    mv a0, t4 #Pasamos el contenido de la operacion del factorial al registro de retorno a0
    jr ra  #retornamos
```

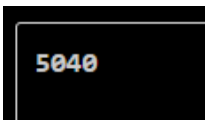
Las salidas esperadas para este código con algunos ejemplos son: 0! = 1, 3! = 6, 7! = 5040 y 8! = 40320.

Y efectivamente el código creado da estas salidas perfectamente:

Para n = 0!

Para n = 3!

Para n = 7!



Para n = 8!

Con el ejercicio número 5 concluimos el informe sobre el laboratorio número 1. En mi opinión fue un laboratorio fructífero y me siento entusiasmado de haber aprendido nuevos conocimientos sobre Risc-V y Venus en el proceso. La documentación y archivos adecuados de cada etapa se encuentran en el zip y además en el repositorio de mi github: https://github.com/IsaacTou/Lab01-Orga2 Sin nada más que agregar me despido.