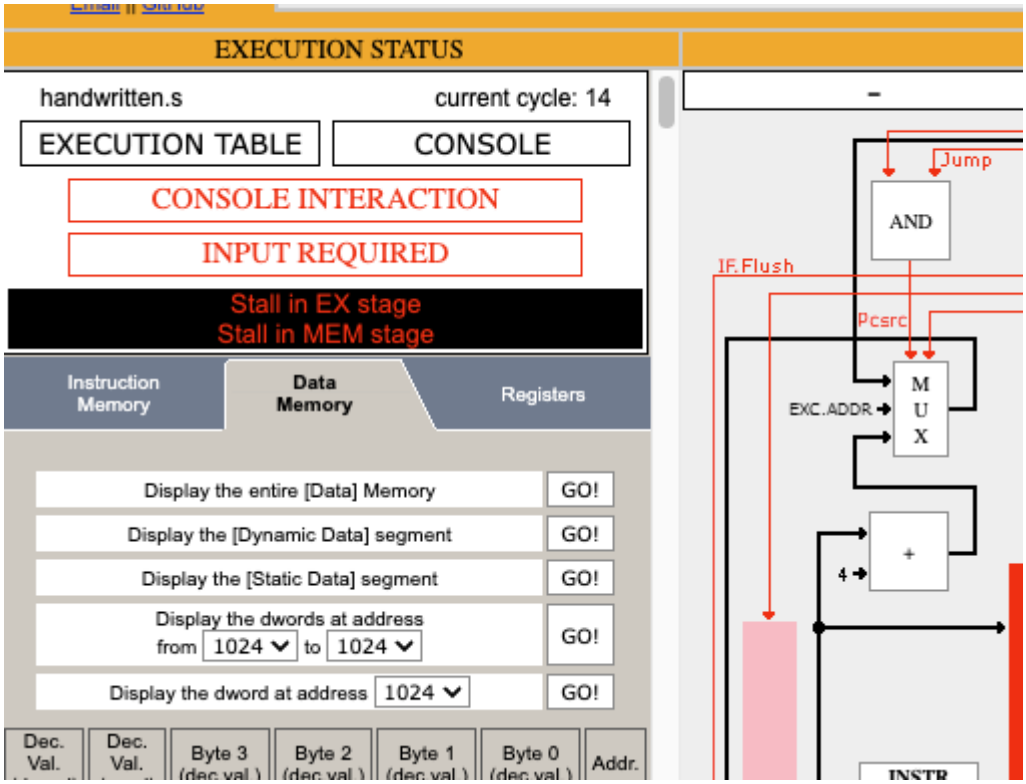
Practica L1

Ejercicio 1.1

¿Qué ocurre con el programa del ejemplo anterior si modificamos el atributo .word por .half?

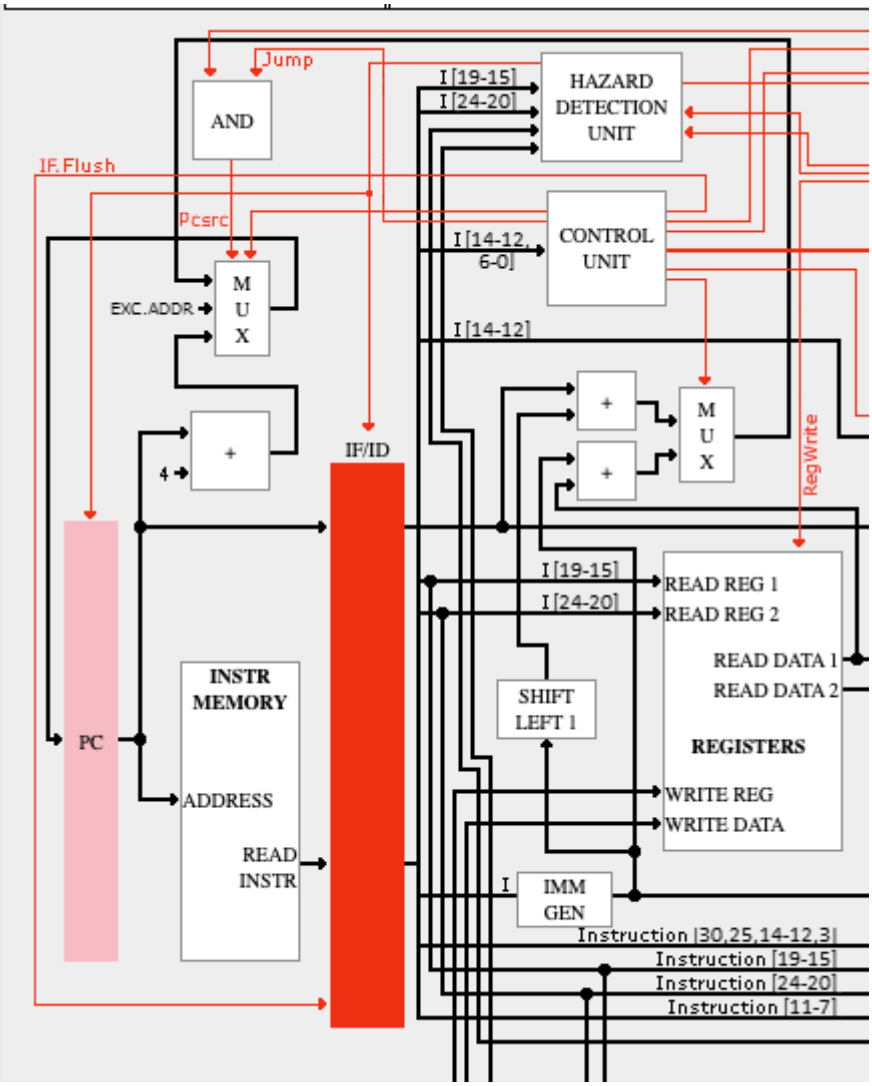
El programa dará un error al cargar el valor de la variable `number` en el registro `t1` ya que `lw` espera una dirección de memoria de 32 bits, pero la variable `number`, al haber sido declarada con el atributo `.half`, ocupa solo 16 bits. Por lo tanto, se deberá cambiar el atributo `.word` a `.half` tanto en la declaración de la variable `number` como en la instrucción `lw`. Además, al sumar el valor de `number` a `a0`, se deberá utilizar `addi` en lugar de `add` ya que los datos cargados con `lw` son de 16 bits y `add` solo funciona con datos de 32 bits.



1.2. ¿Qué componente se encarga de incrementar el PC?

El componente encargado de incrementar el PC (Contador de Programa) es la unidad de control (Control Unit) en el procesador. La unidad de control se encarga de interpretar las instrucciones almacenadas en la memoria y emitir señales de

control para los componentes del procesador, incluyendo el Contador de Programa, para ejecutar la instrucción actual y avanzar al siguiente en la secuencia. El Contador de Programa es un registro que mantiene la dirección de memoria de la siguiente instrucción que se debe ejecutar.



1.3. ¿Qué realiza la operación de control del Write Back marcada como “10”? ¿Qué realiza la operación de control de Memoria (MEM) marcada como “10100”?

La operación de control del Write Back marcada como "10" es responsable de escribir el resultado del cálculo en el registro designado. En otras palabras, esta operación determina si el resultado de la operación debe escribirse en un registro del procesador.

La operación de control de Memoria (MEM) marcada como "10100" es responsable de realizar operaciones de lectura y escritura en la memoria. Esta operación determina si se debe leer o escribir en la memoria y qué dirección de memoria se debe utilizar.

Ejercicio 2.

2.1. Carga el programa de ejemplo “Simple Calculator” y explica qué es lo que realiza (y dónde guarda el resultado) sin modificar de ninguna manera el programa inicial.

El programa "Simple Calculator" es un programa que permite realizar operaciones aritméticas simples. En este caso, se cargan cuatro números (1, 5, 10 y 1), y se utiliza el valor de la variable `s0` para determinar qué operación realizar. Si `s0` es igual a 1, se suma `s1` y `s2` y se almacena el resultado en `s3`. Si `s0` es igual a 2, se resta `s2` de `s1` y se almacena el resultado en `s3`. Si `s0` es igual a 3, se multiplica `s1` y `s2` y se almacena el resultado en `s3`. Si `s0` es igual a 4, se divide `s1` por `s2` y se almacena el resultado en `s3`. Si `s0` no es igual a ninguno de estos valores, se salta a la etiqueta `fine` y se termina el programa.

El resultado de la operación se almacena en la variable `s3`. Dependiendo de la operación realizada, este valor puede ser una suma, una resta, una multiplicación o una división.

2.2 Determina su CPI a partir de la tabla de ejecución proporcionada por el simulador e inserta una captura que lo demuestre.

EXECUTION TABLE

FULL LOOPS	CPU Cycles												
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13
addi s0, x0, 1	F	D	X	M	W								
addi s1, x0, 5		F	D	X	M	W							
addi s2, x0, 10			F	D	X	M	W						
addi t0, x0, 1				F	D	X	M	W					
beq s0, t0, 64					F	-	D	X	M	W			
addi t1, x0, 2							F						
add s3, s1, s2								F	D	X	M	W	
jal x0, 48									F	D	X	M	W
sub s3, s1, s2										F			

El CPI (Ciclos por Instrucción) se calcula sumando los ciclos de reloj totales y dividiéndolo por el número de instrucciones ejecutadas.

CPI = Ciclos de reloj totales / Número de instrucciones ejecutadas

13/9 = 1.44 CPI

2.3. ¿De qué manera se podría hacer que el programa multiplicara?

Para hacer que el programa multiplique, se debe cambiar la tercera instrucción `addi s2, x0, 10` por `addi s2, x0, 2`. Luego, se debe cambiar la cuarta instrucción `addi t0, x0, 1` por `addi t0, x0, 3`. Después, se debe cambiar la etiqueta de la quinta instrucción de `somma` a `molt`, y cambiar la instrucción de esta etiqueta de `add s3, s1, s2` por `mul s3, s1, s2`. De esta manera, la operación que se realizará será una multiplicación en lugar de una suma.

Ejercicio 3.

3.1 Realiza el seguimiento del programa “Simple Calculator” a través de todo el
pipeline. Durante el ciclo 5, explica lo que está sucediendo en cada una de las etapas del procesador.

Comenzando en el ciclo 1, el procesador está buscando la instrucción en la memoria y la decodificando en el ciclo 2. En el ciclo 3, se está ejecutando la

instrucción `addi s0, x0, 1`, que carga el valor 1 en el registro `s0`. En el ciclo 4, se está ejecutando la instrucción `addi s1, x0, 5`, que carga el valor 5 en el registro `s1`.

En el ciclo 5, se está ejecutando la instrucción `addi s2, x0, 10`, que carga el valor 10 en el registro `s2`. En el ciclo 6, se está ejecutando la instrucción `addi t0, x0, 1`, que carga el valor 1 en el registro `t0`. En el ciclo 7, se está comparando el registro `s0` con el registro `t0` para ver si son iguales. Como `s0` es igual a 1, se ejecuta la instrucción `add s3, s1, s2` en el ciclo 8. Esta instrucción suma los valores en los registros `s1` y `s2` y almacena el resultado en el registro `s3`.

3.2 En este mismo ciclo, específicamente en el componente ALU Control Unit, ¿qué está ocurriendo? ¿Qué realiza la operación de la ALU ALUop 10?

En el ciclo 5, el componente ALU Control Unit está decidiendo qué operación realizar en la unidad ALU. La operación que se va a realizar es una suma (`ALUop` 10 indica una suma). Luego, se utilizan los valores en los registros `s1` y `s2` como operandos para la suma y se almacena el resultado en el registro `s3`.

3.3 Realiza un seguimiento de la instrucción “beq s0, t0, 48” durante los ciclos 5
a 8 (ambos incluidos). ¿Por qué del ciclo 10 al 11 la instrucción “sub s3, s1, s2”
desaparece?

Durante los ciclos 5 a 8, se está comparando el valor en el registro `s0` con el valor en el registro `t0`. Como `s0` es igual a 1 y `t0` es igual a 1, la comparación devuelve verdadero y se salta a la etiqueta `fine` en la dirección de memoria 48. Por lo tanto, las instrucciones entre las direcciones de memoria 28 y 44 no se ejecutan.

Después de la instrucción `j fine` en el ciclo 9, el procesador comienza a buscar la siguiente instrucción en la dirección de memoria 48. En este caso, la siguiente instrucción es `addi s1, x0, 10`. Por lo tanto, las instrucciones entre las direcciones de memoria 28 y 44 no se ejecutan. La instrucción `sub s3, s1, s2` es la última instrucción antes de la etiqueta `fine`, por lo que no se ejecuta si se saltó a `fine`.

Ejercicio 4.

Modifique el programa “Simple Calculador” para que lea mediante una llamada al sistema (como hemos visto en el ejemplo) un número del 1 al 5 según la operación que se deseara realizar de manera que se comporte como una calculadora real. Para esto deberá de quedarse en un bucle infinito leyendo de la consola hasta que el número 5 sea leído y finalice la ejecución del programa. El resto de los valores serán 1 para realizar una suma, 2 para restar, 3 para multiplicar y 4 para dividir. Incluye el listado del programa

modificado en tu fichero PDF de respuestas. Además, incluye una captura de pantalla con una prueba de cada una de las opciones demostrando que funciona correctamente.

```
.data
msg1: .asciiz "Ingrese una operación (1 para suma, 2 para resta, 3 para multiplicació
n, 4 para división, 5 para salir): "
msg2: .asciiz "Ingrese el primer número: "
msg3: .asciiz "Ingrese el segundo número: "
res: .asciiz "El resultado es: "

.text
main:
loop:
    # Imprimir mensaje de operación
    li v0, 4
    la a0, msg1
    syscall

    # Leer operación
    li v0, 5
    syscall
    move s0, v0

    # Salir si la operación es 5
    beq s0, 5, end

    # Imprimir mensaje de primer número
    li v0, 4
    la a0, msg2
    syscall

    # Leer primer número
    li v0, 5
    syscall
    move s1, v0

    # Imprimir mensaje de segundo número
    li v0, 4
    la a0, msg3
    syscall

    # Leer segundo número
    li v0, 5
    syscall
    move s2, v0

    # Realizar operación
    addi t0, zero, 1
    beq s0, t0, sum
    addi t1, zero, 2
    beq s0, t1, sub
    addi t2, zero, 3
    beq s0, t2, mul
    addi t3, zero, 4
```

```
    beq s0, t3, div

    # Si la operación no es válida, volver al inicio del bucle
    j loop

sum:
    add s3, s1, s2
    j print

sub:
    sub s3, s1, s2
    j print

mul:
    mul s3, s1, s2
    j print

div:
    div s3, s1, s2
    j print

print:
    # Imprimir mensaje de resultado
    li v0, 4
    la a0, res
    syscall

    # Imprimir resultado
    move a0, s3
    li v0, 1
    syscall

    # Volver al inicio del bucle
    j loop

end:
    # Salir del programa
    li v0, 10
    syscall
```

Ejercicio 5. Carga el programa de ejemplo “Memory reference”

5.1. Explica detalladamente el funcionamiento de este programa

El programa "Memory reference" inicializa dos registros, `s0` y `s1`, respectivamente con los valores `-4` y `-44`. Luego se inicializa el registro `t0` con el valor `10` y el registro `t1` con el valor `1`. El registro `s2` se inicializa con el valor `0`. El programa ejecuta un ciclo que incrementa el valor de `t1` hasta que alcanza el valor de `t0`. Durante cada ciclo, el valor de `t1` se guarda en la memoria asignada en el stack, comenzando en la dirección de memoria inicial de `s2`. Al final del ciclo, el programa restablece `t1` al valor `1`.

Después del ciclo de inserción, el programa inicializa los registros `s2` y `s3` con los valores iniciales de `s0` y `s1`. Luego se ejecuta otro ciclo que copia los valores de la memoria en el stack en una nueva área de memoria en el stack. Los valores se copian comenzando en la dirección de memoria inicial de `s2` y terminando en la dirección de memoria inicial de `s3`. Al final del ciclo, el programa salta al final del programa.

En resumen, el programa asigna una región de memoria en el stack, inserta algunos valores en esa memoria y luego copia esos valores en otra región de memoria en el stack.

Funcionamiento del programa:

1. Inicializa los registros `s0` y `s1` con los valores `4` y `44`.
2. Inicializa los registros `t0` y `t1` con los valores `10` y `1`.
3. Inicializa el registro `s2` con el valor `0`.
4. Ejecuta un ciclo de inserción, incrementando el valor de `t1` y guardando el valor en la memoria asignada en el stack.
5. Restablece `t1` al valor `1`.
6. Inicializa los registros `s2` y `s3` con los valores iniciales de `s0` y `s1`.
7. Ejecuta un ciclo de copia, copiando los valores de la memoria en el stack en otra región de memoria en el stack.
8. Salta al final del programa.

El programa asigna una región de memoria en el stack, inserta algunos valores en esa memoria y luego copia esos valores en otra región de memoria en el stack.

5.2 Determina su CPI a partir de los datos del simulador e inserta una captura que lo demuestre.

El CPI (Ciclos por Instrucción) se calcula sumando los ciclos de reloj totales y dividiéndolo por el número de instrucciones ejecutadas.

CPI = Ciclos de reloj totales / Número de instrucciones ejecutadas

141/31 = 4.548 CPI

EXECUTION TABLE

FULL LOOPS

CPU Cycles

Instruction	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
jal x0, -48	W																									
beq t0, t1, 56	X	M	W																							
addi s2, s2, -4	D	X	M	W																						
lw t2, 0(s2)	F	D	X	M	W																					
addi s3, s3, -4		F	D	X	M	W																				
sw t2, 0(s3)			F	D	X	M	W																			
addi t1, t1, 1				F	D	X	M	W																		
jal x0, -48					F	D	X	M	W																	
beq t0, t1, 56							F	D	X	M	W															
addi s2, s2, -4								F	D	X	M	W														
lw t2, 0(s2)									F	D	X	M	W													
addi s3, s3, -4										F	D	X	M	W												
sw t2, 0(s3)											F	D	X	M	W											
addi t1, t1, 1												F	D	X	M	W										
jal x0, -48													F	D	X	M	W									
beq t0, t1, 56															F	D	X	M	W							
addi s2, s2, -4																F	D	X	M	W						
lw t2, 0(s2)																	F	D	X	M	W					
addi s3, s3, -4																		F	D	X	M	W				
sw t2, 0(s3)																			F	D	X	M	W			
addi t1, t1, 1																				F	D	X	M	W		
jal x0, -48																					F	D	X	M	W	
beq t0, t1, 56																							F	D	X	M
addi s2, s2, -4																								F	D	X
lw t2, 0(s2)																									F	D
addi s3, s3, -4																										F
sw t2, 0(s3)																										
addi t1, t1, 1																										
jal x0, -48																										
beq t0, t1, 56																										
addi s2, s2, -4																										
lw t2, 0(s2)																										
addi s3, s3, -4																										
sw t2, 0(s3)																										
addi t1, t1, 1																										
jal x0, -48																										
beq t0, t1, 56																										
addi s2, s2, -4																										
lw t2, 0(s2)																										
addi s3, s3, -4																										
sw t2, 0(s3)																										
addi t1, t1, 1																										
jal x0, -48																										
beq t0, t1, 56																										
addi s2, s2, -4																										

Instruction	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141
jal x0, -48																									
beq t0, t1, 56																									
addi s2, s2, -4																									
lw t2, 0(s2)																									
addi s3, s3, -4																									
sw t2, 0(s3)																									
addi t1, t1, 1																									
jal x0, -48																									
beq t0, t1, 56																									
addi s2, s2, -4																									
lw t2, 0(s2)																									
addi s3, s3, -4																									
sw t2, 0(s3)																									
addi t1, t1, 1																									
jal x0, -48																									
beq t0, t1, 56																									
addi s2, s2, -4																									
lw t2, 0(s2)																									
addi s3, s3, -4																									
sw t2, 0(s3)																									
addi t1, t1, 1																									
jal x0, -48																									
beq t0, t1, 56	W																								
addi s2, s2, -4	M	W																							
lw t2, 0(s2)	X	M	W																						
addi s3, s3, -4	D	X	M	W																					
sw t2, 0(s3)	F	D	X	M	W																				
addi t1, t1, 1		F	D	X	M	W																			
jal x0, -48			F	D	X	M	W																		
beq t0, t1, 56					F	D	X	M	W																
addi s2, s2, -4						F	D	X	M	W															
lw t2, 0(s2)							F	D	X	M	W														
addi s3, s3, -4								F	D	X	M	W													
sw t2, 0(s3)									F	D	X	M	W												
addi t1, t1, 1										F	D	X	M	W											
jal x0, -48											F	D	X	M	W										
beq t0, t1, 56													F	D	X	M	W								
addi s2, s2, -4														F	D	X	M	W							
lw t2, 0(s2)															F	D	X	M	W						
addi s3, s3, -4																F	D	X	M	W					
sw t2, 0(s3)																	F	D	X	M	W				
addi t1, t1, 1																		F	D	X	M	W			
jal x0, -48																			F	D	X	M	W		
beq t0, t1, 56																					F	D	X	M	W
addi s2, s2, -4																						F			

5.4 Haz un seguimiento a través del pipeline explicando etapa a etapa qué está sucediendo con la instrucción “lw t2, 0(s2)” desde el ciclo 67 al ciclo 71 y de la instrucción “sw t2, 0(s3)” desde el ciclo 69 al ciclo 73.

Seguimiento de la instrucción "lw t2, 0(s2)" (ciclos 67 a 71):

- **Ciclo 67 (IF):** El procesador busca la instrucción "lw t2, 0(s2)" en la memoria.
- **Ciclo 68 (ID):** El procesador decodifica la instrucción "lw t2, 0(s2)" y determina que debe cargar un valor de la memoria. La unidad de control emite señales de control para preparar la unidad de memoria para la lectura.
- **Ciclo 69 (EX):** La unidad ALU calcula la dirección de memoria que se debe leer (0 + s2 = s2) y la unidad de memoria accede a la memoria para obtener el valor.
- **Ciclo 70 (MEM):** La unidad de memoria lee el valor de la memoria y lo carga en el registro interno de la unidad de memoria.
- **Ciclo 71 (WB):** El valor leído se carga en el registro t2.

Seguimiento de la instrucción "sw t2, 0(s3)" (ciclos 69 a 73):

- **Ciclo 69 (EX):** La unidad ALU calcula la dirección de memoria donde se debe escribir el valor (0 + s3 = s3).

- **Ciclo 70 (MEM):** La unidad de memoria actualiza la memoria en la dirección de memoria s3 con el valor del registro t2.
- **Ciclo 71 (WB):** No hay operaciones para realizar en la etapa WB.
- **Ciclo 72 (IF):** El procesador busca la siguiente instrucción en la memoria.
- **Ciclo 73 (ID):** El procesador decodifica la siguiente instrucción y comienza el siguiente ciclo de ejecución.

5.5. Para estas dos instrucciones también revisar cómo tanto el fichero de registros como el mapa de memoria son modificados y mostrar las modificaciones mediante capturas, explicando lo que sucede.

memory.s current cycle: -

EXECUTION TABLE | CONSOLE

Empty IF stage
Empty ID stage
Empty EX stage
Empty MEM stage
Empty WB stage

Instruction Memory | Data Memory | **Registers**

R.No.	Reg.Id.	Dec.Val	Binary Value (64 bit)
0	x0	0	0000000000000000000000000000000000000000000000000000000000000000
1	ra	0	0000000000000000000000000000000000000000000000000000000000000000
2	sp	5120	0000000000000000000000000000000000000000000000000001010000000000
3	gp	1024	0000000000000000000000000000000000000000000000000000010000000000
4	tp	0	0000000000000000000000000000000000000000000000000000000000000000
5	t0	0	0000000000000000000000000000000000000000000000000000000000000000
6	t1	0	0000000000000000000000000000000000000000000000000000000000000000
7	t2	0	0000000000000000000000000000000000000000000000000000000000000000
8	s0/fp	5120	0000000000000000000000000000000000000000000000000001010000000000
9	s1	0	0000000000000000000000000000000000000000000000000000000000000000
10	a0	0	0000000000000000000000000000000000000000000000000000000000000000
11	a1	0	0000000000000000000000000000000000000000000000000000000000000000
12	a2	0	0000000000000000000000000000000000000000000000000000000000000000
13	a3	0	0000000000000000000000000000000000000000000000000000000000000000
14	a4	0	0000000000000000000000000000000000000000000000000000000000000000
15	a5	0	0000000000000000000000000000000000000000000000000000000000000000
16	a6	0	0000000000000000000000000000000000000000000000000000000000000000
17	a7	0	0000000000000000000000000000000000000000000000000000000000000000
18	s2	0	0000000000000000000000000000000000000000000000000000000000000000
19	s3	0	0000000000000000000000000000000000000000000000000000000000000000
20	s4	0	0000000000000000000000000000000000000000000000000000000000000000
21	s5	0	0000000000000000000000000000000000000000000000000000000000000000
22	s6	0	0000000000000000000000000000000000000000000000000000000000000000
23	s7	0	0000000000000000000000000000000000000000000000000000000000000000
24	s8	0	0000000000000000000000000000000000000000000000000000000000000000
25	s9	0	0000000000000000000000000000000000000000000000000000000000000000
26	s10	0	0000000000000000000000000000000000000000000000000000000000000000
27	s11	0	0000000000000000000000000000000000000000000000000000000000000000
28	t3	0	0000000000000000000000000000000000000000000000000000000000000000
29	t4	0	0000000000000000000000000000000000000000000000000000000000000000
30	t5	0	0000000000000000000000000000000000000000000000000000000000000000
31	t6	0	0000000000000000000000000000000000000000000000000000000000000000

memory.s

EXECUTION TABLE | CONSOLE

EXECUTION COMPLETED IN 141 CLOCK CYCLES

Empty IF stage
Empty ID stage
Empty EX stage
Empty MEM stage
Empty WB stage

Instruction Memory | Data Memory | **Registers**

R.No.	Reg.Id.	Dec.Val	Binary Value (64 bit)
0	x0	0	0000000000000000000000000000000000000000000000000000000000000000
1	ra	0	0000000000000000000000000000000000000000000000000000000000000000
2	sp	5120	0000000000000000000000000000000000000000000000000001010000000000
3	gp	1024	0000000000000000000000000000000000000000000000000000010000000000
4	tp	0	0000000000000000000000000000000000000000000000000000000000000000
5	t0	10	0000000000000000000000000000000000000000000000000000000000001010
6	t1	10	0000000000000000000000000000000000000000000000000000000000001010
7	t2	9	0000000000000000000000000000000000000000000000000000000000001001
8	s0/fp	5116	0000000000000000000000000000000000000000000000000001001111111100
9	s1	5076	0000000000000000000000000000000000000000000000000001001111010100
10	a0	0	0000000000000000000000000000000000000000000000000000000000000000
11	a1	0	0000000000000000000000000000000000000000000000000000000000000000
12	a2	0	0000000000000000000000000000000000000000000000000000000000000000
13	a3	0	0000000000000000000000000000000000000000000000000000000000000000
14	a4	0	0000000000000000000000000000000000000000000000000000000000000000
15	a5	0	0000000000000000000000000000000000000000000000000000000000000000
16	a6	0	0000000000000000000000000000000000000000000000000000000000000000
17	a7	0	0000000000000000000000000000000000000000000000000000000000000000
18	s2	5080	0000000000000000000000000000000000000000000000000001001111011000
19	s3	5040	0000000000000000000000000000000000000000000000000001001110110000
20	s4	0	0000000000000000000000000000000000000000000000000000000000000000
21	s5	0	0000000000000000000000000000000000000000000000000000000000000000
22	s6	0	0000000000000000000000000000000000000000000000000000000000000000
23	s7	0	0000000000000000000000000000000000000000000000000000000000000000
24	s8	0	0000000000000000000000000000000000000000000000000000000000000000
25	s9	0	0000000000000000000000000000000000000000000000000000000000000000
26	s10	0	0000000000000000000000000000000000000000000000000000000000000000
27	s11	0	0000000000000000000000000000000000000000000000000000000000000000
28	t3	0	0000000000000000000000000000000000000000000000000000000000000000
29	t4	0	0000000000000000000000000000000000000000000000000000000000000000
30	t5	0	0000000000000000000000000000000000000000000000000000000000000000
31	t6	0	0000000000000000000000000000000000000000000000000000000000000000

Ejercicio 6. Modificar el programa del ejercicio 4 para leer de consola los operandos, además del operador, de manera que estos se puedan modificar para
realizar cualquier tipo de operación, como en una calculadora real. Hacer una captura de pantalla con el resultado de cada una de las cuatro operaciones para
demostrar que funciona de manera satisfactoria (por ejemplo, del fichero de registros, indicando el significado de cada uno de ellos).

```
# Este programa sirve como una simple
# calculadora de 4 operaciones entre
# los dos operandos en s1 y s2
# la operación se elige en s0
# add     s0 = 1
# sub     s0 = 2
# mul     s0 = 3
# div     s0 = 4
# operando 1  = s1
# operando 2  = s2

loop:
    # Imprimir mensaje de operación
    li v0, 4
    la a0, msg1
    syscall

    # Leer operación
    li v0, 5
    syscall
    move s0, v0

    # Salir si la operación es 5
    beq s0, 5, end

    # Imprimir mensaje de primer número
    li v0, 4
    la a0, msg2
    syscall

    # Leer primer número
    li v0, 5
    syscall
    move s1, v0

    # Imprimir mensaje de segundo número
    li v0, 4
    la a0, msg3
    syscall
```

```
    # Leer segundo número
    li v0, 5
    syscall
    move s2, v0

    # Realizar operación
    addi t0, zero, 1
    beq s0, t0, sum
    addi t1, zero, 2
    beq s0, t1, sub
    addi t2, zero, 3
    beq s0, t2, mul
    addi t3, zero, 4
    beq s0, t3, div

    # Si la operación no es válida, volver al inicio del bucle
    j loop

sum:
    add s3, s1, s2
    j print

sub:
    sub s3, s1, s2
    j print

mul:
    mul s3, s1, s2
    j print

div:
    div s3, s1, s2
    j print

print:
    # Imprimir mensaje de resultado
    li v0, 4
    la a0, res
    syscall

    # Imprimir resultado
    move a0, s3
    li v0, 1
    syscall

    # Volver al inicio del bucle
    j loop

end:
    # Salir del programa
    li v0, 10
    syscall
```

Para que el programa lea los operandos desde la consola, se agregó un ciclo donde se leen los operandos y la operación deseada desde la consola. Luego, se realiza la

operación según la elección del usuario y se imprime el resultado en la consola.

Los registros utilizados son los siguientes:

- `s0`: Almacena la operación deseada (1 para suma, 2 para resta, 3 para multiplicación, 4 para división).
- `s1`: Almacena el primer operando.
- `s2`: Almacena el segundo operando.
- `s3`: Almacena el resultado de la operación.

Ejercicio 7. Cargar el programa de ejemplo “Factorial”

7.1. Entender el programa y hacer un seguimiento del mismo. Un programa cualquiera recursivo consta de tres fases, explica de manera general y breve con el programa factorial que hay de ejemplo qué está ocurriendo en cada una de ellas y aproximadamente en qué ciclos está cada una de ellas.

El programa `Factorial` es una función recursiva que calcula el factorial de un número `n`. El programa consta de tres fases principales:

1. **Condición de parada:** Si `n` es menor que 2, se devuelve 1. Esta es la condición de parada para la recursión. Este segmento del programa se ejecuta en los ciclos 15-18.
2. **Otra llamada recursiva:** Si `n` no es menor que 2, se llama a `factorialRec` con el argumento `n-1`. Este segmento del programa se ejecuta en los ciclos 19-22.
3. **Cálculo del resultado:** Una vez que se alcanza la condición de parada, se multiplica el valor actual de `n` con el resultado de la llamada recursiva anterior. Este segmento del programa se ejecuta en los ciclos 23-26.

En resumen, el programa `Factorial` es una función recursiva que calcula el factorial de un número `n`. Si `n` es menor que 2, se devuelve 1. De lo contrario, se llama a `factorialRec` con el argumento `n-1` y se multiplica el valor actual de `n` con el resultado de la llamada recursiva anterior.

El seguimiento del programa se puede dividir en los siguientes segmentos:

1. **Inicialización:** El registro `a2` se inicializa con el valor 5 y se salta a la etiqueta `main`. Esto sucede en el ciclo 5.
2. **Llamada principal:** El programa salta a la etiqueta `factorialRec` con el argumento `n=5`. Esto sucede en el ciclo 6.

3. **Condición de parada:** La función `factorialRec` comprueba si `n` es menor que 2. Como `n=5` no es menor que 2, se ejecuta el siguiente segmento del programa. Esto sucede en los ciclos 15-18.
4. **Otra llamada recursiva:** Se llama a `factorialRec` con el argumento `n-1=4`. Esto sucede en los ciclos 19-22.
5. **Condición de parada:** La función `factorialRec` comprueba si `n` es menor que 2. Como `n=4` no es menor que 2, se ejecuta el siguiente segmento del programa. Esto sucede en los ciclos 15-18.
6. **Otra llamada recursiva:** Se llama a `factorialRec` con el argumento `n-1=3`. Esto sucede en los ciclos 19-22.
7. **Condición de parada:** La función `factorialRec` comprueba si `n` es menor que 2. Como `n=3` no es menor que 2, se ejecuta el siguiente segmento del programa. Esto sucede en los ciclos 15-18.
8. **Otra llamada recursiva:** Se llama a `factorialRec` con el argumento `n-1=2`. Esto sucede en los ciclos 19-22.
9. **Condición de parada:** La función `factorialRec` comprueba si `n` es menor que 2. Como `n=2` es menor que 2, la función devuelve 1. Esto sucede en los ciclos 15-18.
10. **Cálculo del resultado:** El resultado de la llamada recursiva anterior se multiplica por el valor actual de `n=3`. Esto sucede en los ciclos 23-26.
11. **Cálculo del resultado:** El resultado de la llamada recursiva anterior se multiplica por el valor actual de `n=4`. Esto sucede en los ciclos 23-26

7.2. ¿Qué ocurre entre los ciclos 56 y 60? Revisar el valor del registro SP en ese instante y hacer una captura de pantalla en este punto del stack de memoria y explicar qué significan cada uno de los valores de esta.

Entre los ciclos 56 y 60, el programa ejecuta las instrucciones `addi sp, sp, -8`, `sw a2, 0(sp)` y `sw ra, 4(sp)`. Estas instrucciones modifican el valor del registro SP y guardan los valores de los registros `a2` y `ra` en el stack.

El registro SP indica la dirección de memoria actual en el stack. Cuando se guarda un valor en el stack, el registro SP se decrementa para apuntar a la siguiente posición de memoria disponible. En este punto del programa, el registro SP se ha decrementado en 8, lo que significa que se ha reservado espacio en el stack para dos valores.

La captura de pantalla muestra el contenido del stack de memoria en este punto del programa. Los valores `5` y `0` se han guardado en las dos posiciones de memoria más recientes del stack. Estos valores corresponden a los registros `a2` y `ra`, respectivamente. El registro `a2` contiene el valor `5`, que es el argumento pasado a la función `factorialRec`. El registro `ra` contiene la dirección de retorno para la función `factorialRec`.

7.3. Realiza un seguimiento de la instrucción “mul a0, a2, a0” entre los ciclos 72 y 76. ¿Qué está ocurriendo? ¿Y entre los ciclos 78 y 82? ¿Y finalmente, entre 84 y 88?

El programa es una función recursiva que calcula el factorial de un número `n`. La función `factorialRec` toma un argumento `n`, comprueba si `n` es menor que 2 y devuelve 1 si lo es. De lo contrario, llama a `factorialRec` con el argumento `n-1` y multiplica el resultado de la llamada recursiva por `n`.

El seguimiento de la instrucción "mul a0, a2, a0" (ciclos 72-76) se puede dividir en los siguientes segmentos:

1. **Ciclo 72 (EX):** La unidad ALU multiplica los valores de `a2` y `a0`.
2. **Ciclo 73 (MEM):** No hay operaciones para realizar en la etapa MEM.
3. **Ciclo 74 (WB):** El resultado de la multiplicación se carga en el registro `a0`.

El resultado de la multiplicación se almacena en el registro `a0`.

El seguimiento de la instrucción "mul a0, a2, a0" (ciclos 78-82) se puede dividir en los siguientes segmentos:

1. **Ciclo 78 (EX):** La unidad ALU multiplica los valores de `a2` y `a0`.
2. **Ciclo 79 (MEM):** No hay operaciones para realizar en la etapa MEM.
3. **Ciclo 80 (WB):** El resultado de la multiplicación se carga en el registro `a0`.

El resultado de la multiplicación se almacena en el registro `a0`.

El seguimiento de la instrucción "mul a0, a2, a0" (ciclos 84-88) se puede dividir en los siguientes segmentos:

1. **Ciclo 84 (EX):** La unidad ALU multiplica los valores de `a2` y `a0`.
2. **Ciclo 85 (MEM):** No hay operaciones para realizar en la etapa MEM.
3. **Ciclo 86 (WB):** El resultado de la multiplicación se carga en el registro `a0`.

El resultado de la multiplicación se almacena en el registro `a0`.

7.4. Las llamadas recursivas a la función se producen entre las líneas 28 ->11, en “bucle”, pero las instrucciones que calculan numéricamente el factorial están en las líneas 31 y 32, es decir fuera del bucle ¿Explica cómo es posible que se calcule el factorial?

El cálculo del factorial se realiza mediante una función recursiva llamada `factorialRec`. Esta función toma un argumento `n` y comprueba si `n` es menor que 2. Si `n` es menor que 2, la función devuelve 1. De lo contrario, la función llama a `factorialRec` con el argumento `n-1` y multiplica el resultado de la llamada recursiva por `n`.

El bucle `loop` en el programa principal solo se utiliza para leer los operandos y la operación deseada de la consola. Una vez que se han leído los operandos y la operación, se llama a la función correspondiente (suma, resta, multiplicación o división) y se imprime el resultado. La función correspondiente utiliza los operandos y realiza la operación deseada.

En el programa `Factorial`, la llamada recursiva a la función se realiza en el segmento del programa que comprueba si `n` es menor que 2. Si `n` es menor que 2, la función devuelve 1. De lo contrario, la función llama a `factorialRec` con el argumento `n-1` y multiplica el resultado de la llamada recursiva por `n`. La multiplicación se realiza después de la llamada recursiva, pero el resultado se utiliza en la llamada recursiva anterior, lo que permite el cálculo recursivo del factorial.

En resumen, el cálculo recursivo del factorial se realiza mediante una función que comprueba si `n` es menor que 2. Si lo es, devuelve 1. De lo contrario, llama a `factorialRec` con el argumento `n-1` y multiplica el resultado de la llamada recursiva por `n`. La multiplicación se realiza después de la llamada recursiva, pero el resultado se utiliza en la llamada recursiva anterior, lo que permite el cálculo recursivo del factorial.