ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.1. Un programa escrito para MIPS invoca a la subrutina func, a la que transfiere tres argumentos a, b y c. La subrutina tiene dos variables locales m y n. Mostrar la posición del puntero de pila y los contenidos de los registros relevantes de la pila, inicialmente vacía, cuando se invoca la subrutina en las líneas del programa que se indican:
a) Justo antes de la llamada en la línea rotulada como # Apartado a), antes de la instrucción **jal.**
b) Cuando se completa la zona de pila para **func.**
c) Justo antes de ejecutar la instrucción **lw** en la línea comentad con # Apartado c).

```
# Programa para llevar a la pila los argumentos a, b y c
        jal     func            # Apartado a)
        lw      $s1, 0($t2)     # Apartado c)
        …       …
func:                           # Comienzo de la subrutina
        …       …
        jr      $ra
```

SOLUCIÓN:

a) Si la pila está vacía, el puntero señala inicialmente al fondo de la pila, es decir a la primera posición que ya no es pila ($sp = 0x4000). Antes de transferir el control a la subrutina la zona de pila cambia, guardando las variables de paso.

	ANTES		DESPUES
		$sp → 0x3FF4	c
		0x3FF8	b
		0x3FFC	a
$sp → 0x4000	XXXXXXXX	0x4000	XXXXXXXX

b) Al comienzo de la subrutina, esta guarda el valor del registro de retorno $ra (o si se sabe que no hay llamada anidada, no es necesario hacerlo. Además, el orden de inserción de $ra, m y n puede ser el contrario) y crea espacio para las variables locales m y n. A partir de este momento la subrutina utiliza esta zona de memoria no como pila sino como una memoria local, utilizando el puntero de pila como registro de direcciones. Esta zona de memoria reservada para la subrutina desaparece una vez que esta termina.

	b)		c)
0x3FE4		0x3FE4	
$sp → 0x3FE8	n	0x3FE8	n
0x3FEC	m	0x3FEC	m
0x3FF0	$ra	0x3FF0	$ra
0x3FF4	c	0x3FF4	c
0x3FF8	b	0x3FF8	b
0x3FFC	a	0x3FFC	a
0x4000	XXXXXXXX	$sp → 0x4000	XXXXXXXX

a) Como la primera instrucción tras jal, no lee de una posición de memoria asociada al registro $sp (puntero de pila) implica que la subrutina no transfiere datos a la rutina principal. En este caso, antes de devolver el control a la rutina principal, libera el espacio de memoria de la pila decrementando en puntero tantas veces como lo haya incrementado en la subrutina mas el número de variables que le han sido transferidas. La subrutina decrementa en 24 posiciones el puntero (20 si no hubiera añadido $ra en la pila), señalando se nuevo a la posición inicial. Los datos guardados no cambian, pero se borraran al escribir encima nuevos valores.

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.2. En el proceso de enlace, un determinado código simbólico genera una tabla de símbolos que asocia a cada símbolo la posición de memoria en donde se ubicará tras el proceso de carga. Se pide escribir la tabla de símbolos que corresponde al código siguiente. Utilizar "U" para los símbolos indefinidos.

```
        .text   0x0100
        j       main
        .text   0x0200
main:   or      $s2, $s1, 0x1000
        srl     $s2, $s2, 10
lab_4:  sw      $s2, K($0)
        addi    $s1, $s1, -1
foo:    sw      $s1, K ($0)
        and     $s1, $s1, $0
        beq     $s1, $s2, lab_5
        .data    0x3000
cons:   .space  12
K:              0x4000
```

SOLUCION:

Tabla de Símbolos

Símbolo	Valor (hexa)
main	0x0200
lab_4	0x0208
K	0x300C
foo	0x0210
lab_5	U
cons	0x3000

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.3. Un desensamblador es un programa que lee un módulo objeto y genera un lenguaje simbólico. Dado el siguiente código objeto se pide el correspondiente lenguaje desensamblado para MIPS. Dado que el código objeto no contiene información para poder determinar los nombres de los símbolos, se les asignará ordenadamente las letras del abecedario a medida que sean necesarias.

0010 0010 0011 0001 0000 0000 0000 0001
0000 0010 1100 0000 1010 1000 0010 0101
0001 0010 0011 0101 0000 0000 0000 0010
0000 0000 0001 0110 1011 0010 1000 0010
0000 1100 0000 0000 0000 0000 0000 0000
0000 0011 1110 0000 0000 0000 0000 1000

SOLUCION:

op	rs	rt	rd	shamt	funct	imm	Código
							.text 0x0000
001000	10001	10001	--	--	--	0000000000000001	A: addi $s1, $s1, 1
000000	10110	00000	10101	00000	100101	--	or $s5, $s6, $0
000100	10001	10101	--	--	--	0000000000000010	beq $s1, $s5, B
000000	00000	10110	10110	01010	000010	--	srl $s6, $s6, 10
000011	--	--	--	--	--	0000....26 bits....0000	jal A
000000	11111	00000	00000	00000	001000	--	B: jr $ra

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.4. Escribir dos funciones o subrutinas para MIPS llamadas **push** y **pop** que sirvan, la primera a partir de la posición 0x0100 de memoria para escribir dos registros en la pila del sistema y la segunda a partir de la posición 0x0200 para leer dos registros que estén guardados en la pila del sistema. Utilice como argumentos los registros de MIPS definidos para el paso de parámetros.

SOLUCION:

```
                #rutina push
                .text  0x0100
push:           addi $sp, $sp, - 4
                sw $a0, 0($sp)
                addi $sp, $sp, - 4
                sw $a1, 0($sp)
                jr $ra

                #rutina pop
                .text  0x0200
pop:            lw $v0, 0($sp)
                addi $sp, $sp, + 4
                lw $v1, 0($sp)
                addi $sp, $sp, + 4
                jr $ra
```

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.5. El programa escrito para MIPS que se muestra invoca a una subrutina **sub1**, a la que transfieren dos argumentos A y B. La subrutina **sub1**, utiliza dos variables locales M y N y devuelve una variable resultado Z. Por su parte **sub1** invoca a la función **func1** a la que transfiere un único argumento P y de la que recibe también una variable S. La función **func1** necesita utilizar una variable local. Si el paso de variables se realiza por medio de la pila y el valor del puntero de pila antes del comienzo del programa es 0x0004000, se pide, utilizando la estructura facilitada para justificar cada respuesta, el contenido del registro $sp y los valores guardados en la pila en los siguientes hitos de la ejecución.

a) Justo antes de ejecutar la instrucción rotulada a.
b) Justo antes de ejecutar la instrucción rotulada d.
c) Justo antes de ejecutar la instrucción rotulada func1.
d) Justo antes de ejecutar la instrucción rotulada f.
e) Justo antes de ejecutar la instrucción rotulada e.
f) Escribir las instrucciones que se deben escribir en la posición rotulada con las letras b y c.

```
        # Programa para resolver

a:      jal     sub1
b:      # Instrucción nueva 1
c:      # Instrucción nueva 2
        # ---------------------------------------
sub1: # Comienza la subrutina sub1
        # ----
d:      jal     func1
        # ----
e:      jr      $ra     # retorno de sub1
        # ---------------------------------------
func1:  # Comienza la función func1
        # ----
        # ----
f:      jr      $ra     # retorno de func 1
```

SOLUCION:

a) Justo antes de ejecutar la instrucción rotulada a. El valor de $sp es: 0x3FF8
b) Justo antes de ejecutar la instrucción rotulada d. El valor de $sp es: 0x3FE8
c) Justo antes de ejecutar la instrucción rotulada func1. El valor de $sp es: 0x3FE8
d) Justo antes de ejecutar la instrucción rotulada f. El valor de $sp es: 0x3FE8
e) Justo antes de ejecutar la instrucción rotulada e. El valor de $sp es: 0x3FFC

ESTADO INICIAL		a) Antes de a	
		%sp→0x3FF8	B
0x3FFC		0x3FFC	A
$sp→ 0x4000	XXXXXXXX	0x4000	XXXXXXXX

b) Antes de d		c) Antes de func1	
$sp→ 0x3FE8	P	$sp→ 0x3FE8	P
0x3FEC	N*	0x3FEC	N*
0x3FF0	M*	0x3FF0	M*
0x3FF4	$ra*	0x3FF4	$ra*
0x3FF8	B	0x3FF8	B
0x3FFC	A	0x3FFC	A
0x4000	XXXXXXXX	0x4000	XXXXXXXX

d) Antes de f		e) Antes de e	
$sp→ 0x3FE8	S		S
0x3FEC	N*		N*
0x3FF0	M*		M*
0x3FF4	$ra*		$ra*
0x3FF8	B		B
0x3FFC	A	$sp→ 0x3FFC	Z
0x4000	XXXXXXXX	0x4000	XXXXXXXX

* $ra, M y N podrían estar en orden inverso.

f) Son las instrucciones necesarias para recuperar el valor de la pila y restaurarla a su valor inicial, es decir:

```
b:      lw  $sx, 0($sp)          # $sx <= Z; $sx, representa cualquier registro interno.

c:      addi $sp, $sp, 4
```

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.6. Dado el siguiente programa escrito para MIPS, en donde se mezclan instrucciones en ensamblador y en código máquina (hexadecimal). Se pide, **a)** completar el código ensamblador y **b)** Indicar la función que ejecuta el código y escribir el valor de la posición de memoria señalada por la etiqueta F.

# Problema para calcular	**SOLUCION:**
.text 0x0000	.text 0x0000
0x8C112000	lw $s1, N($0)
add $s3, $s1, $0	add $s3, $s1, $0
0x2232FFFF	addi $s2, $s1, -1
beq $s1, $0, fin	beq $s1, $0, fin
L1: **0x0C00000A**	L1: jal func
addi $s2, $s2, -1	addi $s2, $s2, -1
beq $s2, $0, fin	beq $s2, $0, fin
j L1	j L1
fin: sw $s3, F($0)	fin: sw $s3, F($0)
parar: **0x08000009**	parar: j parar
func: and $s6, $s6, $0	func: and $s6, $s6, $0
add $s7, $s2, $0	add $s7, $s2, $0
L2: add $s6, $s3, $s6	L2: add $s6, $s3, $s6
addi $s7, $s7, -1	addi $s7, $s7, -1
beq $s7, $0, L3	beq $s7, $0, L3
0x0800000C	j L2
L3: add $s3, $s6, $s0	L3: add $s3, $s6, $s0
0x03E00008	jr $ra
.data 0x2000	.data 0x2000
N: 0x0003	N: 0x0003
F: 0x0000	F: 0x0000

b) El programa calcula el factorial del número escrito en N. Por tanto, el valor de la posición de memoria F será: 3! = 6

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.7. Dado el siguiente programa escrito para MIPS, en donde se mezclan instrucciones en ensamblador y en código máquina (hexadecimal). Se pide, **a)** completar el código ensamblador y **b)** Indicar la función que ejecuta el código y escribir el valor del registro $s7 al final del proceso completo.

# Programa para calcular	**SOLUCION**:
.text 0x0000	.text 0x0000
lw $s1, X($0)	lw $s1, X($0)
0x8C132004	lw $s3, Y($0)
nor $s2, $s1, $s1	nor $s2, $s1, $s1
nor $s4, $s3, $s3	nor $s4, $s3, $s3
and $s5, $s4, $s1	and $s5, $s4, $s1
and $s6, $s3, $s2	and $s6, $s3, $s2
or $s7, $s6, $s5	or $s7, $s6, $s5
0x08000007	fin: j fin
.data 0x2000	.data 0x2000
X: 0x0055	X: 0x0055
Y: 0x00AA	Y: 0x00AA

b) La función es X ⊕ Y. El valor del registro $s7 será 0x000000FF

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.8. Escribir en código ensamblador para MIPS, una función llamada **mover**, que sirva para trasladar una tabla de datos con tamaño de celda de 4 bytes desde una ubicación inicial a otra final. Los argumentos que se le pasarán a la función por medio de los registros internos $a0, $a1y $a2 serán, longitud de la tabla (en bytes), dirección origen (primera posición de memoria origen) y dirección destino (primera posición de memoria destino) respectivamente. La longitud, deberá ser siempre un múltiplo de 4 y mayor que cero.
Nota: escribir junto a cada instrucción un comentario explicativo.

SOLUCIÓN:

```
mover:  # Comienzo de la función
        add   $t1, $a0, $0      # Guarda en el registro $t1 la longitud de la tabla
        add   $t2, $a1, $0      # Guarda en el registro $t2 la dirección origen
        add   $t3, $a2, $0      # Guarda en el registro $t3 la dirección destino
        and   $t4, $t4, $0      # Inicializa un contador que se guarda en $t4
mov:    lw    $t5,  0($t2)      # Lee de la memoria origen
        sw    $t5, 0($t3)       # Escribe en la memoria destino
        addi  $t4, $t4, 4       # Incrementa el contador
        addi  $t2, $t2, 4       # Incrementa dirección origen
        addi  $t3, $t3, 4       # Incrementa dirección destino
        beq   $t1, $t4, fin     # Termina si ha completado la tabla
        j     mov               # Si no ha terminado continúa
fin:    jr    $ra               # Vuelve a la rutina llamante
```

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.9. Se pide completar el programa dado para MIPS que calcula el producto de dos números P = M*N, donde N se puede escribir como $N = 2^X - 2^Y$. En memoria figuran inicialmente M, X e Y, siendo X e Y positivos entre 0 y 31.
Nota: Señalar un breve comentario de cada una de las instrucciones que se escriban.

# Programa para calcular	**Comentario**
.text 0x0000	
lw $t1, M($0)	#Lee M en $t1 (*)
lw $t2, X($0)	#Lee X en $t2
lw $t3, Y($0)	#Lee Y en $t3
sllv $t4, $t1, $t2	# $M*2^X$
sllv $t5, $t1, $t3	# $M*2^Y$
sub $s1, $t4, $t5	# $M*2^X - M*2^Y$
sw $s1, P($0)	# escribe el resultado
fin: j fin	
.data 0x2000	
M:	# Contiene multiplicando
X:	# Contiene el multiplicador $N = 2^X - 2^Y$
Y:	
P:	# Guarda el producto

(*) Es equivalente poner lw $t1, M($0) que lw $t1, M. En ambos casos el ensamblador tendrá que traducir la etiqueta X a un dato inmediato que sumará al registro $0 para componer la dirección.

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.10. Utilizando el lenguaje ensamblador de MIPS, escribir una función denominada **swap** que sirva para intercambiar el contenido de dos registros cualesquiera del banco de registros (por ejemplo entre $s0 y $s1), sin modificar ningún otro registro de propósito general.
a) Utilice la pila para la transferencia de parámetros entre el programa principal y la rutina.
b) Realice la misma función, pero en este utilice los registros específicos de MIPS para el paso de parámetros a procedimientos ($a0-$a3) y para el retorno de resultados ($v0-$v1). Además de la función **swap**, complete el código de llamada a la función y el de retorno después de la misma.
Nota: Se valorará que el programa funcione y que utilice el menor número de instrucciones posible.

SOLUCIÓN:

a) # Código para intercambiar dos registros utilizando la pila para el paso de parámetros

Instrucción		**Comentario**
main:	addi $sp, $sp, - 8	# Decrementa el puntero de pila en 8 unidades
	sw $s0, 4($sp)	# Guarda un registro ($s0) en la pila
	sw $s1, 0($sp)	# Guarda un registro ($s1) en la pila
	jal swap	# Llama a la función swap
	lw $s1, 0($sp)	# Lee de la pila el primer parámetro $s0 y lo guarda en $s1
	lw $s0, 4($sp)	# Lee de la pila el primer parámetro $s1 y lo guarda en $s0
	addi $sp, $sp, + 8	# Restaura la pilar a su valor inicial sumando 8 unidades
	…….	
swap:	lw $t0, 0($sp)	# Lee de la pila el primer parámetro $s1 y lo guarda en $t0
	lw $t1, 4($sp)	# Lee de la pila el segundo parámetro $s0 y lo guarda en $t1
	sw $t1, 0($sp)	# Escribe en la pila el primer parámetro de vuelta ($s0)
	sw $t0, 4($sp)	# Escribe en la pila el segundo parámetro de vuelta ($s1)
	jr $ra	# Retorna a la rutina llamante

b) # Utilizando una función que utilice los registros específicos para el paso de parámetros

Instrucción		**Comentario**
main:	add $a0, $s0, $0	# Se escribe en $a0 uno de los registros a intercambiar
	add $a1, $s1, $0	# Se escribe en $a1 el otro de los registro a intercambiar
	jal swap	# Llama a la función swap
	add $s0, $v0, $0	# Escribe el segundo parámetro devuelto en el primer registro
	add $s1, $v1, $0	# Escribe el primer parámetro devuelto en el segundo registro
	…….	
swap:		# la función recibe los registros a intercambiar en $a0 y $a1
	add $v0, $a1, $0	# $v0 <= $a1
	add $v1, $a0, $0	# $v1 <= $a0, la función retorna los registros ya intercambiados por $v0 y $v1
	jr $ra	# retorna a la rutina llamante

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.11. El tamaño de todas las instrucciones de un determinado procesador es de 24 bits. El procesador dispone de un banco de registros de propósito general con 64 registros. En el set de instrucciones se distinguen tres tipos diferentes. El número de instrucciones para cada uno de los tres tipos es: 14 con un formato que utiliza tres registros operando (dos fuentes y uno destino), 47 que utilizan dos (uno fuente y uno destino) y 4 que utilizan un solo registro operando (destino).

Nota: Considerar que el código de operación (OP) no tiene que ser del mismo tamaño para todas las instrucciones.

Indicar, justificando la respuesta:

a. Los campos necesarios para el formato de cada uno de los tipos de instrucciones señaladas. Es decir cuántos bits se utilizan para indicar el tipo de instrucción, cuántos para el código de operación, cuántos para la dirección de registros y cuántos para los otros campos considerados.

b. ¿A cuántas direcciones de memoria diferentes se puede acceder en instrucciones cuyo formato utiliza un único registro operando? Señalar un posible ejemplo para este tipo de instrucciones y cómo sería la operación que realiza.

SOLUCIÓN:

a) Al tener un banco de registros de 64, cada registro operando necesita un campo de 6 bits, por tanto,

al existir tres tipos de instrucciones, se necesitan 2 bits para su identificación (OP1).

Formato con tres registros: hay 14 instrucciones con tres operandos registro ($14 \leq 2^4$) se necesita un mínimo de 4 bits para el campo OP2.

OP1	OP2	R_FUENTE	R_FUENTE	R_DESTINO
2 bits	4 bits	6 bits	6 bits	6 bits

Formato con dos registros: hay 47 instrucciones con dos operandos registro ($47 \leq 2^6$) se necesita un mínimo de 6 bits para el campo OP2.

OP1	OP2	Sin USAR	R_FUENTE	R_DESTINO
2 bits	6 bits	4 bits	6 bits	6 bits

Formato con un registro: hay 4 instrucciones con un solo operando registro ($4 \leq 2^2$) se necesita un mínimo de 2 bits para el campo OP2.

OP1	OP2	OTROS USOS (Direcciones)	R_DESTINO
2 bits	2 bits	14 bits	6

b) Se podrá acceder hasta 2^{14} direcciones.

Nota: Una solución menos homogénea, poco habitual en RISC, permitiría aumentar en un bit el campo para direcciones en memoria para el tercero de los formatos. La solución implica usar 2 bits en el campo OP1 para los dos primeros, por ejemplo 00 y 01, lo cual permite identificar con un solo bit de valor '1' en el msb, el formato del tercer tipo. En este caso habría 15 bits para otros usos, en consecuencia se podría acceder hasta 2^{15} direcciones. Esa mejora supone un decodificador más complejo, por lo que no es recomendable en una arquitectura tipo RISC.

Una instrucción para llevar un dato desde memoria a un registro (LOAD), puede tener este formato.

El dato inmediato sirve para señalar la dirección de la memoria y el campo R_DESTINO señala dónde se guarda el dato leído.

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.12. Sea una arquitectura de 16 bits (tamaño de palabra y de registro) con 6 registros y un espacio de direccionamiento de 64 kBytes. La ALU es capaz de realizar 20 operaciones distintas (sumar, restar, and, or, etc…). En la resolución del problema se debe tener en cuenta que esta arquitectura no es MIPS y que:

1. Los códigos de operación tienen todos la misma longitud.
2. No todas las instrucciones tienen necesariamente el mismo tamaño.
3. Se puede leer/escribir de memoria un solo byte.

Se pide, **justificando necesariamente la respuesta**, diseñar el formato de instrucción que permita construir un juego de instrucciones con los siguientes tipos (un formato para cada tipo):

Tipo1: Instrucciones entre registros en la ALU, al menos 40 instrucciones.

Tipo2: Instrucciones para leer/escribir en memoria con direccionamiento absoluto a cualquier posición de memoria (la dirección está contenida en la instrucción). Al menos 5 instrucciones.

Tipo3: Instrucciones con memoria con direccionamiento relativo a un registro, con un desplazamiento máximo de 128 bytes (la dirección se obtiene sumando un dato inmediato al contenido de un registro). Al menos 5 instrucciones.

Tipo4: Instrucciones de salto condicional, con la posibilidad de saltar a cualquier otra instrucción guardada en memoria. Al menos 4 instrucciones.

SOLUCION:

Según el enunciado, el código de operación (OP) de todas las instrucciones tiene el mismo tamaño, como hay un mínimo de (40+5+5+4 = 54) instrucciones se necesitan 6 bits como mínimo para el código de operación. Respuestas con OP de mayor tamaño son válidas, si se mantiene el tamaño para todos los formatos.

Al tener 6 registros de propósito general, serán necesarios 3 bits para identificar cada operando de este tipo.

Al tener una capacidad de memoria direccionada por bytes de 64 kbytes = 2^{16} kbytes, se necesitan 16 bits para acceder a cualquier posición de la memoria (Tipo2) o a cualquier instrucción (Tipo4).

Para un desplazamiento relativo a registro de 128 bytes, si es siempre hacia delante suponen 8 bits en el campo desplazamiento [0 : +255]. Pero si también pueden ser hacia atrás se necesitan 9 bits [-256 : +255]. Ambas respuestas se consideran válidas.

Según lo anterior, una solución para los formatos solicitados puede ser:

Tipo1: Necesita el campo OP y tres campos para los registros destino (1) y fuente (x2). Como la palabra es de 16 bits, este tipo de instrucciones sólo necesita 1 palabra.

OP	$R_{DESTINO}$	R_{FUENTE}	R_{FUENTE}	Sin uso
6bits	3 bits	3 bits	3 bits	1 bit

Tipo2: Además del campo OP, este tipo de instrucciones necesita un campo para indicar el registro cuyo contenido se escribe/lee en/desde memoria ($R_{F/D}$) y otro campo para señalar la dirección absoluta. Como la palabra es de 16 bits, este tipo de instrucciones necesita 2 palabras.

OP	$R_{F/D}$	Sin uso
6bits	3 bits	7 bit

Dirección Absoluta
16 bits

Tipo3: Además del campo OP, para este tipo de instrucciones, se necesitan un campo para indicar el registro cuyo contenido se escribe/lee en/desde memoria ($R_{F/D}$), otro campo registro para generar la dirección efectiva (R_{FUENTE}) y por último otro campo para señalar el desplazamiento máximo de 128 bytes. Como la palabra es de 16 bits, este tipo de instrucciones necesita 2 palabras.

OP	$R_{F/D}$	R_{FUENTE}	Sin uso
6bits	3 bits	3 bits	4 bit

Sin uso	Dato Inmediato
8 ó 7 bits	8 ó 9 bits

Tipo4: Suponiendo que se comparan 2 registros (al igual que en MIPS), se necesitan referenciar a los dos registros, el campo OP y se necesita un campo para señalar la dirección de salto. Como la memoria es de 64 kbytes, este campo debe ser de al menos 16 bits. Como la palabra es de 16 bits, este tipo de instrucciones necesita 2 palabras.

OP	R_{FUENTE}	R_{FUENTE}	Sin uso
6bits	3 bits	3 bits	4 bit

Dirección de salto
16 bits

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.13. Se quiere escribir un programa para guardar en la posición de memoria señalada por el literal **resul**, la suma de los números en hexadecimal 0xFF00 y 0x00FF. Se facilita parte del programa, señalando con una línea discontinua la posición que deben ocupar las instrucciones o valores que faltan y que el estudiante debe completar.

```
    .text 0x0800
main:  ①----------------------------------# Escribir la instrucción que falta
       ②----------------------------------# Escribir la instrucción que falta
       add $s1, $s2, $s3
       ③----------------------------------# Escribir la instrucción que falta

    .data 0x2000
val1:          0xFF00
resul: ④----------------------------------# Escribir el resultado del programa
```

a) Se pide construir la tabla de variables del programa dado.

Literal	Valor
main	0x0800
resul	0x2004
val1	0x2000

b) Con la ayuda de la tabla de códigos adjunta, escribir en hexadecimal el código máquina para la instrucción: add $s1, $s2, $s3

0x02538820

c) Completar las líneas de código y los valores señalados en el programa.

① main: lw $s2, val1($0)

② addi $s3, $0, 0x00FF

③ sw $s1, resul($0)

④ resul: 0x0000FFFF # Hay que considerar la extensión de signo

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.14. Sea un sistema procesador basado en MIPS como el estudiado en clase. Con la ayuda de las tablas de código facilitadas, se pide:

a) Codificar en lenguaje máquina la instrucción: **beq $s1,$t1, fin.**

Indique el resultado en binario y en hexadecimal.

b) Encontrar la codificación de la instrucción escrita en lenguaje máquina del código dado. Con los operandos obtenidos en la codificación, señale la función que ejecuta esta instrucción.

a) beq $s1, $t1, fin

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op						rs					rt					imm																
0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

En hexadecimal: **0x1229000E**

b) 0x3C111CAA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op						rs					rt					imm																
0	0	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	0

op: 001111 => lui
rs: 000000
rt: 10001 => $s1
imm: 0x1CAA

La instrucción es: lui $s1, 0x1CAA

Esta instrucción coloca en los 16 bits de más peso del registro $s1 el valor 0x1CAA, poniendo 16 ceros en la parte baja del citado registro.

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.15. Se muestra el código MIPS para un cierto programa principal, main y dos rutinas sub1 y sub2 a las que la rutina principal llama durante su ejecución. Para el paso de parámetros entre rutinas, se utiliza la pila del sistema controlada por el puntero de la pila que utiliza el registro sp, que siempre señala a la última posición ocupada de la pila. Las tres rutinas están desarrolladas por personas diferentes por lo que hay que respetar los protocolos para la protección de registros que afectan a las rutinas escritas para MIPS.

```
        .text 0x0000
main:   addi $s3, $0, 0x000A
        lw $s5, A($0)
        addi $sp, $sp, -4
        sw $s5, 0($sp)
        beq $s3, $s5, salto1
salto2: jal sub1
        lw $s3, 0($sp)
        addi $sp, $sp, 4
        ori $s1, $s3, 0xFF00
        sw $s1, R($0)
        j fin
salto1: jal sub2
        lw $s4, 0($sp)
        addi $sp, $sp, 4
        andi $s1, $s4, 0xFF00
        sw $s1, R($0)
fin :   j fin

.data 0x2000
A: 0x0001
R:
```

```
        .text 0x0100
sub1:   lw $s1, 0($sp)
        addi $t2, $0, 0x0004
        sllv $t3, $s1, $t2
        sw $t3, 0($sp)
        jr $ra
```

```
        .text 0x0200
sub2:   lw $t3, 0($sp)
        addi $t5, $0, 0x0002
        sllv $t1, $t3, $t5
        sw $t1, 0($sp)
        jr $ra
```

Se pide,

a) Analizar el código dado y señalar:

a1. En hexadecimal el valor del símbolo R => 0x00002004

a2. En hexadecimal el valor que contiene la posición de memoria R => 0x0000FF10

b) **Justificando brevemente la respuesta**, señalar el número de parámetros de entrada y salida entre la rutina llamante y cada una de las rutinas llamadas.

La rutina main pasa por medio de la pila un parámetro ($s5) a ambas subrutinas, también las subrutinas devuelven un único parámetro por la pila ($t3) la subrutina sub1 y ($t1) la subrutina sub2.

c) **Justificando brevemente la respuesta**, aunque no afecte necesariamente al resultado final, señalar el error o errores cometidos por el programador de la rutina sub1.

Aunque no afecta al resultado, en la rutina sub1 se utiliza (escribe) sin preservar previamente en la pila el registro $s1 reservado para las rutinas llamantes.

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.16. Se sabe que el siguiente código realiza la media aritmética (redondeando hacia abajo) de los valores de las posiciones de memoria *A, B, C* y *D*, dejando el resultado en *R*. Para ello la función principal, *main*, llama a otra función, *med4*, que es la que realiza la media aritmética de sus cuatro operandos de entrada. El intercambio de información con dicha función se realiza mediante los registros reservados para tal fin en MIPS.
Se pide completar la función *med4* con las tres instrucciones que faltan (cada hueco corresponde a una y sólo una instrucción), así como dar el valor final del registro *$ra,* de la posición de memoria *R* y de las etiquetas *A*, *med4* y *fin*.

.text 0
lw $a0, A
lw $a1, B
lw $a2, C
lw $a3, D
jal med4
sw $v0, R
fin: j fin
	.text 0x0100
	med4: add $t0, $a0, $a1
	add $t1, $a2, $a3
	add $t0, $t0, $t1
	sra $v0, $t0, 2
	jr $ra
		.data 0x2000
		A: 5
		B: 6
		C: 7
		D: 8
		R: 0

Contenido final de:

$ra: 0x0014

R: 0x0006

Etiquetas:

A: 0x2000

med4: 0x0100

fin: 0x0018

ESTRUCTURA DE COMPUTADORES

SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.17. Se adjunta un programa en C para MIPS que implementa varias funciones y utiliza la pila para el paso de argumentos y el uso de variables locales. Se pide definir el estado de la pila en determinados instantes de ejecución del código. Considerar que el puntero a pila ($sp) empieza en la posición x4000 y que el compilador no realiza ninguna optimización de código. Indique, para cada caso, el valor real escrito en pila cuando sea posible saberlo. Todas las rutinas, la principal y las secundarias, utilizan la pila para guardar sus variables locales.

void main(){ *int aux = 5;* *aux=fun1(3, 7);* **(a)** *fun2(aux);* **(d)** *}* **(f)**	*int fun1(int a, int b){* *int k;* **(b)** *k = a+b;* *return k+1;* **(c)** *}*	*void fun2(a){* *printf("El número es\n",a);* *return;* **(e)** *}*

(a) en el "jal" en MIPS

Valor $sp	Contenido pila
x3FF4	7
	3
	5 (aux)

x4000 →

(b) en etiqueta b

Valor $sp	Contenido pila
	k
x3FF0	7
	3
	5 (aux)

x4000 →

(c) en el "jr" en MIPS

Valor $sp	Contenido pila
x3FF8	~~k~~
	~~7~~
	~~3~~ 11
	5 (aux)

x4000 →

(d) en el "jal" en MIPS

Valor $sp	Contenido pila
x3FF8	~~k~~
	~~7~~
	11
	11 (aux)

x4000 →

(e) en el "jr" de MIPS

Valor $sp	Contenido pila
x3FFC	
	~~7~~
	~~11~~
	11 (aux)

x4000 →

(f) terminado *main*

Valor $sp	Contenido pila
X4000	~~k~~
	~~7~~
	~~11~~
	~~11 (aux)~~

x4000 →

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.18. Se tienen dos números con signo guardados en dos posiciones de memoria cualesquiera etiquetadas como A y B y se desea escribir en las mismas posiciones la suma y el valor medio de ambos números respectivamente. Complete el programa para MIPS adjunto, que permita ejecutar esta tarea, y en el que existe una función denominada **SumMed** que recibe como parámetros de entrada los números y devuelve la suma y el valor medio de sus contenidos.

a. Utilice la pila para la transferencia de parámetros entre el programa principal y la rutina llamada. Complete el código de llamada a la función y el de retorno después de la misma.

b. Utilice los registros específicos de MIPS para el paso de parámetros a procedimientos ($a0-$a3) y para el retorno de resultados ($v0-$v1). Complete el código de llamada a la función y el de retorno después de la misma.

Nota: Se valorará que el programa funcione y que utilice el menor número de instrucciones posible, así como la adecuada inclusión de comentarios en la respuesta. Deje sin completar en la tabla las filas que no necesite.

SOLUCIÓN

a) Instrucción		**Comentario**
main:	**lw $s0, A($0)**	**# Lee de memoria el primer número A y lo guarda en $s0**
	lw $s1, B($0)	**# Lee de memoria el segundo número B y lo guarda en $s1**
	addi $sp, $sp, - 8	# Decrementa el puntero de pila en 8 unidades *
	sw $s0, 4 ($sp)	# Guarda un registro ($s0) en la pila
	sw $s1, 0($sp)	# Guarda un registro ($s1) en la pila
	jal SumMed	# Llama a la función SumMed
	lw $s1, 0($sp)	# Lee de la pila el valor de la suma y lo guarda en $s1
	lw $s0, 4($sp)	# Lee de la pila el valor del promedio y lo guarda en $s0
	addi $sp, $sp, +8	# Restaura la pila a su valor inicial sumando 8 unidades
	sw $s1, A($0)	**# Escribe en la posición de memoria A la suma.**
	sw $s0, B($0)	**# Escribe en la posición de memoria B el valor medio**
	…….	
SumMed:	lw $t1, 0($sp)	# Lee de la pila el número B y lo guarda en $t1
	lw $t0, 4($sp)	# Lee de la pila el número A y lo guarda en $t0
	add $t1, $t0, $t1	# Escribe en $t1 el resultado de la suma
	sra $t0, $t1, 1	# Escribe en $t0 el valor medio aritmético, conserva el signo.
	sw $t1, 0($sp)	# Guarda la suma (A +B) en la pila
	sw $t0, 4($sp)	# Guarda el valor medio la suma (A +B) / 2 en la pila
	jr $ra	# Retorna a la rutina llamante

b) Instrucción		**Comentario**
main:	lw $a0, A($0)	# Lee de memoria el primer número A y lo guarda en $a0
	lw $a1, B($0)	# Lee de memoria el segundo número B y lo guarda en $a1
	jal SumMed	# Llama a la función SumMed
	sw $v1, A($0)	# Escribe en la posición de memoria A la suma.
	sw $v0, B($0)	# Escribe en la posición de memoria B el valor medio
	…….	
SumMed:	add $v1, $a0, $a1	# Escribe en $v1 el resultado de la suma
	sra $v0, $v1, 1	# Escribe en $t0 el valor medio
	jr $ra	# Retorna a la rutina llamante

ESTRUCTURA DE COMPUTADORES
SOLUCIONES A LOS PROBLEMAS DE LA UNIDAD-3.- EL PROCESADOR I: EL LENGUAJE MAQUINA

3.19. Un programa escrito para MIPS, llama a una subrutina utilizando los registros específicos para el paso de parámetros. Sabiendo que la subrutina llamada recibe dos parámetros y devuelve uno,

a) Señale la instrucción o instrucciones que debe escribir en la rutina principal para guardar el parámetro recibido en la posición de memoria 0x2008, así como su código máquina en hexadecimal.

El registro que contiene el parámetro de salida de la subrutina es $v0 = $2, la instrucción para escribir en memoria es sw, por tanto la instrucción será:

```
sw $v0, 0x2008($0)
```

y su equivalente código máquina será: 101011 00000 00010 0010000000001000 = **0xAC022008**

b) Escriba un programa para MIPS para escribir en el registro $t4 la constante 0x12345678. Se valora la corrección del código y el uso del menor número de instrucciones posible. Comente brevemente la solución propuesta.

Una constante de más de 32 bits como es el caso, se debe guardar en un registro por medio de dos instrucciones:

```
lui $t4, 0x1234          # Carga los 16 msb en la parte alta del registro.
ori $t4, $t4, 0x5678     # Con la or lógica se incorporan los 16 lsb restantes de la constante.
```

c) Escriba las instrucciones necesarias dentro de un programa para MIPS para leer de memoria el valor de un número guardado en la posición de memoria identificada por la etiqueta o símbolo A y escribir su complemento a dos en la posición siguiente. Se valora la corrección del código y el uso del menor número de instrucciones posible. Comente brevemente la solución propuesta.

```
lw $s1, A($0)        # Lee de memoria el contenido de la posición A y lo guarda en $s1
sub $s1, $0, $s1     # Restando a cero el número se calcula su valor negativo, es decir su
                     # complemento a 2.
addi $t0, $0, 4      # Guarda un 4 en un registro auxiliar, tamaño en bytes del dato
sw $s1, A($t0)       # Escribe el complemento a 2 en la posición siguiente de A (4 bytes adelante).
```