MODOS DE DIRECCIONAMIENTO

Aunque la mayoría de las instrucciones de MIPS utilizan como operandos los registros, existen algunas capaces de acceder a memoria. Existen distintos modos de especificar la dirección o modos de direccionamiento.

Modo direccionamiento	Ejemplo	Dirección
Relativo a registro	**Lw $t1,($t2)**	Guarda en el registro t1 el contenido del registro t2.
Absoluto (con valor)	**Lw $t1,0x10010008**	Guarda en t1 el contenido de la dirección especificada en la propia instrucción.
Relativo a registro base (RB)	**Lw $t1,4($t2)**	Se guarda en t1 el contenido de una dirección de memoria especificada por un registro,$t2, más un desplazamiento, el 4
Absoluto (con etiqueta)	**lw $t1, etiqueta**	Se guarda en t1 el contenido de una dirección de memoria especificada por la ETIQUETA "etiqueta".
Etiqueta±desplazamiento	**lw $t1,etiqueta+8**	Se guarda en t1 el contenido de una dirección de memoria especificada por la ETIQUETA "etiqueta" + un desplazamiento (8)
Etiqueta + RB	**lw $t1,etiqueta($t3)**	Se guarda en t1 el contenido de una dirección de memoria especificada por la ETIQUETA "etiqueta" + un desplazamiento dado por un registro, en este caso t3.
Etiqueta±RB+desplazamiento	**lw $t1,etiqueta+4($t3)**	Se guarda en t1 el contenido de una dirección de memoria especificada por la ETIQUETA "etiqueta" + el contenido de un registro (t3) + un desplazamiento alineado 4.

VECTORES O ARRAYS DE NÚMERO ENTEROS

Los vectores son elementos que no pueden guardarse en los registros, así siempre serán guardados en memoria. Lo que si puede almacenarse en los registros es la dirección del primer elemento del vector y a continuación será posible acceder al resto de elementos usando el **modo de direccionamiento relativo a registro base**.

Podrán crearse vectores de enteros donde cada elemento sea de un byte de longitud, de media palabra o de una palabra entera.

```
.data
 array_byte:        .byte 60,61,62,63    # Hex → 3c,3d,3f,3e
 array_media:       .half 10,20,30,40    # Hex → 000a,0014, 001e,0028
 array_palabra:     .word -1,-2,-3,-4    # Hex (C a 2)→
                                         ffffffff,fffffffe,fffffffd,fffffffc
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x3f3e3d3c	0x0014000a	0x0028001e	0xffffffff	0xfffffffe	0xfffffffd	0xfffffffc	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

	3	2	1	0
1001 0000	3f	3e	3d	3c
1001 0004	00	14	00	0a
1001 0008	00	28	00	1e
1001 000c	ff	ff	ff	ff
1001 0010	ff	ff	ff	fe
1001 0014	ff	ff	ff	fd
1001 0018	ff	ff	ff	fc
.....				

```
.data
 array_byte:      .byte 60,61,62,63
 array_media:     .half 10,20,30,40
 array_palabra:   .word -1,-2,-3,-4
```

```
.text
.globl main
main:
 la $t1,array_byte      #Apunta a la dirección del 1° elemento de array_byte

 lb $s0,0($t1)          # Carga el 1° elemento del vector en s0
 lb $s1,1($t1)          # Carga el 2° elemento del vector en s1
 lb $s2,2($t1)          # Carga el 3° elemento del vector en s2
 lb $s3,3($t1)          # Carga el 4° elemento del vector en s3
```

Es importante darse cuenta, de que según el tipo de datos empleados, el incremento de la dirección de cada elemento del array se produce según múltiplos de 1, 2 o 4 para bytes, media palabra y palabras respectivamente. Existe una instrucción similar a "lw" que permite acceder a cada byte de una palabra de memoria y es "lb".

PROBLEMA 1

Modificar el código anterior para que se cargue en los registros s0, s1, s2, s3 y s4 el vector denominado ***array_media****.*

PROBLEMA2

Repetir para el tercer vector de enteros de palabras denominado ***array_palabra***

LLAMADAS AL SISTEMA

Las llamadas a funciones del sistema (SYSCALL) más habituales se pueden encontrar en la hoja de referencia de MIPS y también en la ayuda de MARS. Cada función se identifica por un número y además puede recibir parámetros y/o devolver resultados. Para usarlas seguiremos estos cuatro pasos:

1. Guardar parámetros -si la función los usa-, en los registros correspondientes ($a0 y $a1).
2. Guardar en $v0 el número de la función que se quiere llamar.
3. Ejecutar la instrucción SYSCALL
4. Recoger los resultados –si la función los devuelve- de los registros correspondientes ($a0 y $a1).

La llamada más habitual es siempre la que muestra resultados por pantalla. En el caso de MIPS, habrá que indicar donde se encuentra el valor a imprimir, p.ej. el elemento 0 de array_byte, sito en s0 tras ejecutar los códigos anteriores, y a continuación guardar en v0 el valor numérico asociado a la llamada deseada, 1 para la impresión de un entero por pantalla.

Añadiendo las siguientes líneas al código anterior, se visualizará el primer elemento del array_byte en pantalla.

```
move $a0,$s0      # Mueve el contenido del registro s0, primer elemento, al registro $a0
li $v0,0x00000001# La llamada asociada con imprimir por pantalla un entero es la 1
syscall           # Efectúa la llamada
```

PROBLEMA 3

Añadir en el código del problema 1, las líneas necesarias para mostrar por pantalla solo los elementos impares del array tratado.

PROBLEMA 4

Añadir en el código del problema 2, las líneas necesarias para mostrar por pantalla solo los elementos pares del array tratado, separados por una coma.

PROBLEMA 5

Escribe el código de un programa que sume dos valores almacenados en memoria. Estos dos valores reciben los nombres Numero1 y Numero2 y sus valores son respectivamente 100 y -10. Para el resultado realice una reserva de espacio en memoria a continuación de Numero1 y Numero2 (Utilice la directiva .space). Para la carga de memoria en registros use t0 para Numero1 y t1 para Numero2.

ACCESO A VECTORES MEDIANTE ÍNDICES

MEDIANTE SUMAS

Este programa calcula la suma de todos los elementos de un vector. Contiene un bucle en el que la variable "índice" almacenada en $t0 toma los valores 0, 1, 2, 3. Este índice se multiplica en cada iteración por el tamaño de los datos usados (.word = 4 bytes) para obtener el offset o desplazamiento correspondiente a cada dato del vector. El desplazamiento se suma a la dirección base del vector (indicada por la etiqueta "valor") para obtener la dirección efectiva de cada dato. Usando esta dirección se descarga cada dato y se acumula su valor en la variable "suma" almacenada en el registro $t1.

```
.data
valor: .word 0xa,0xb,0x01,0x02

.text
.globl main
 main:

 move $t0,$zero           # $t0<-- "indice" con valor inicial 0
 move $t1,$zero           # $t1<-- "suma" con valor inicial 0
 li   $t2,4               # $t2<-- constante
 la   $s0, valor          # $s0 ← puntero a 'valor'

bucle:
 add $t3,$t0,$t0    # t3 = indide +indice = 2 * indice
 add $t3,$t3,$t3    # t3 = 2*indide + 2*indice = 4 * indice
 add $t4,$s0,$t3    # Suma del desplazamiento a la dirección base, t1= base +
                      4*i

 lw $s1,0($t4)      # Carga del elemento referenciado por la dirección
                    guardada en t1, se carga en s1
 add $t1,$t1,$s1    # Suma el elemnto a la suma anterior

 add $t0,$t0,1      # Incremento del indice
 blt $t0,$t2,bucle  # Repite el bucle si no se ha llegado al ultimo elemento
                    (indice<4)
```

MEDIANTE MULTIPLICACIÓN. PROBLEMA 6

Observar cómo en el programa anterior se calcula el desplazamiento $t3 mediante sumas repetidas del índice. Se pide sustituir este cálculo por una instrucción de multiplicación. Es buena idea usar la referencia de MIPS o la ayuda de MARS para encontrar el nombre y la sintaxis de la instrucción que se va a utilizar.

MEDIANTE DESPLAZAMIENTO ARITMÉTICO A LA IZQUIERDA. PROBLEMA 7.

De la misma manera que en el ejercicio anterior, modificar esta vez el cálculo del desplazamiento para realizarlo mediante un desplazamiento de bits (equivalente a multiplicar por una potencia de 2).

PROBLEMA 8

En el código anterior, para acceder a cada valor del array se calcula su dirección efectiva (puntero) en $t4, como suma de la dirección base del array + desplazamiento variable.
En vez de eso, se pide ahora revisar los modos de direccionamiento (primera página de este documento) y encontrar cuál de ellos permite acceder directamente a cada valor, sabiendo que ya tenemos una etiqueta en la dirección base del array, y un desplazamiento calculado en el apartado anterior.

PROBLEMA 9

Modificar el código anterior para que al término de la suma del array el programa muestre por pantalla el mensaje "La suma es:" seguido del resultado numérico. Para esto será necesario primero declarar la cadena de caracteres en memoria de datos, y después añadir al programa las llamadas al sistema (SYSCALL) necesarias.

Más atrás en este mismo documento (sección "llamadas al sistema") se indican los pasos que seguir para

PROBLEMA 10

Modificar en el código anterior la declaración de datos para que el vector sea de medias palabras (.half) y corregir el programa para que funcione de acuerdo con este cambio. Observar que la modificación es fácil gracias a que el desplazamiento lo calculamos a partir de un índice que se incrementa de uno en uno. Si en vez de usar ese índice hubiéramos ido incrementando de 4 en 4 el desplazamiento, esta modificación habría implicado cambios más incómodos (especialmente la condición de salida del bucle).

PROBLEMA 11

Modificar el código anterior para que al término de la suma del array guarde en memoria el resultado. Considere que inicialmente habrá que efectuar una reserva de memoria para esta variable a la que denominaremos "SUMA".

PROBLEMA 12

Diseñe un código para almacenar en un array en memoria, denominado array1, los siguientes valores enteros de 32 bits: 5, 13,-7,-5, 17.

PROBLEMA 13

Copie el contenido de array1, de 5 elementos enteros y ubicado al principio de la memoria de datos, en array 2, situado dos posiciones más delante de array1.

Array1={2,34,12,-4,6}

PROBLEMA 14

Diseñe el código de un programa que determine el elemento de mayor valor dentro de un array de cuatro elementos. Debe mostrar por pantalla tanto el índice del elemento como el valor, precedidos ambos por los textos: "Indice" y "Valor".

PROBLEMA 15

Diseñe un código que pida por teclado dos valores enteros, que los sume y muestre el resultado por pantalla.

PROBLEMA 16

Mediante índices y bucles desarrolle un código que muestre por pantalla un array de bytes almacenado en memoria. La presentación por pantalla atenderá al siguiente formato: "Nombre del ARRAY"={VALOR 1, VALOR 2,…}.