Práctica 7
Vectores

Objetivos

- Conocer cómo se representan las cadenas de caracteres en el lenguaje ensamblador.
- Conocer cómo se representan los vectores de números enteros en lenguaje ensamblador
- Conocer las maneras de acceder a cada uno de los elementos de un vector.

Materiales

Simulador MARS y un código fuente de partida.

Desarrollo de la práctica

Introducción

Un vector (array o string) es un elemento que contiene un conjunto de valores del mismo tipo. En el ensamblador del MIPS un vector se implementa almacenando un conjunto de valores en posiciones contiguas de la memoria y accediendo a cada valor mediante un desplazamiento desde la dirección de comienzo del vector.
Un vector se podría almacenar en cualquier segmento de datos (estáticas, dinámicas o pila) de la memoria del MIPS, la diferencia está en que en las estáticas o en la pila el tamaño del vector se fija cuando se compila el programa y en las dinámicas puede variar durante la ejecución del programa. De momento estudiaremos los vectores en el segmento de datos estáticos.

Vectores de caracteres

Las cadenas de caracteres son también vectores con datos de tamaño un byte. El MIPS nos permite definir cadenas de caracteres en memoria con las directivas *.asciiz* y *.ascii*. Con la directiva *.asciiz* se define una cadena de caracteres acabada con el carácter null, útil cuando se la quiere recorrer, puesto que nos permite reconocer su final. Se utiliza la directiva *.ascii* cuando se quiere definir una cadena que no acaba con null. Por ejemplo, `.ascii “Universitat d'Alacant”` reserva 21 bytes consecutivos en la memoria de datos iniciados con esos caracteres. Si quisiésemos utilizar la instrucción syscall para imprimir una cadena en la pantalla, necesariamente tendría que acabar con el carácter null. Podemos utilizar *.ascii* con una cadena muy larga en la que dividimos su definición varías líneas del programa como muestra el ejemplo siguiente:

```
            #Contar caracteres de una cadena
.data
str:.ascii "Estructuras de los"
    .asciiz "Computadores"

.text

la $s0, str
add $s1, $zero, $zero           # Iniciamos contador a 0
loop:
     add $t0, $s0, $s1          # dirección del byte a examinar
     lb $t1, 0( $t0 )
     beq $t1, $zero, exit       # salimos si carácter leído='\0'
     addi $s1, $s1, 1           # incrementamos el contador
     j loop

exit: li $v0, 10
      syscall
```

- Ensambla y ejecuta el código. ¿Cuántos caracteres tiene la cadena?
- ¿En qué dirección de la memoria se encuentra el carácter *null*)?
- ¿Cómo se actualiza el índice del vector?
- ¿El programa funcionaría si la cadena solo constara del carácter *null*?
- Modifica el código para que muestre por pantalla el mensaje "El número de caracteres de la cadena es: " y a continuación el resultado.
- Modifica el código para que calcule el número de veces que se repite la vocal "u".

Vectores de enteros

Imaginemos que queremos definir y operar con un vector A de enteros de *n* elementos, para acceder al elemento i-ésimo del vector lo expresamos como A[i]. ¿Cómo lo podemos hacer en MIPS? Suponemos que la dirección del primer elemento del vector la guardamos en un registro, por ejemplo $t0. Esta será la dirección base del vector. Para acceder a cualquier elemento del vector lo haremos mediante un desplazamiento respecto de la dirección base. Lo que hay que tener con cuenta es que un entero ocupa una palabra en memoria (4 bytes) y el acceso a la memoria del MIPS es por byte, por lo tanto el desplazamiento siempre será un múltiplo de 4. Podemos acceder a un elemento del vector en la memoria para leerlo o escribirlo, por ejemplo: `lw $s0, 4($t0)` leerá el segundo elemento del vector.

```
            #Ejemplo: recorrer un vector de enteros

.data
 A: .word 2, 4, 6, 8, 10        # vector A iniciado con valores
 B: .word 0:4                   # Vector B vacío
 C: .space 50                   # Otra definición de vector
vacio

.text
la $s0, A                       # Dirección base del vector A
la $s1, B                       # Dirección base del vector B
li $s5, 5                       # Tamaño del vector

loop:       add $t1, $s0, $t0
            add $t2, $s1, $t0
```

```
            addi $s2, $s2, 1     # Índice del vector
            lw $t3, 0($t1)
            sw $t3, 0($t2)
            sll $t0, $s2, 2      # Índice del vector x4
            bne $s2, $s5, loop

li $v0, 10
syscall
```

Análisis del código:

Se han definido tres vectores de 5 elementos, el tamaño de cada uno de los elementos es de una palabra (4 bytes). El primer vector A se ha definido con valores iniciales definidos. El segundo vector B se ha definido indicando exclusivamente el tamaño. Sólo utilizamos estos dos en el código. El tercer vector C se ha definido reservando 50 bytes de la memoria (5 palabras de 4 bytes).
El código copia los valores del vector A al vector B y lo hace recorriendo los dos vectores, cada vez que lee un elemento de A lo copia en B. Para recorrer los vectores multiplicamos por 4 el índice del vector en cada iteración.

- Ensambla y ejecuta el programa. Comprueba que el vector A se copia en el vector B. ¿En qué dirección empieza el vector B?
- ¿Porque no se pueden acabar los vectores con el carácter *null* igual que se hace con las cadenas de caracteres?
- En el programa se recorren los vectores actualizando el índice con la instrucción *sll*. ¿De qué otra manera se podrían recorrer los vectores?
- Se ha utilizado un bucle del tipo do-while, modifica el programa por que el bucle sea de tipo for-while.
- Modifica el programa para que el vector B se rellene con enteros leídos del teclado. Previamente se tiene que mostrar un mensaje por consola que pida los elementos.
- Completa el programa para que se rellene el vector C con la suma de los elementos del vector A y del B (C[y]=A[y]+B[y]).

Direccionamiento en memoria

En MIPS el acceso a una posición de la memoria para leer o escribir se hace, como ya hemos visto previamente, obteniendo la dirección mediante la suma del contenido de un registro base y un desplazamiento de 16 bits. Por ejemplo, `lw $s1, 4($s2)` lee el contenido de la posición de memoria obtenida con la suma $s2+4 y lo guarda en $s1. A esta forma de acceder a la memoria se la conoce con el nombre de ***direccionamiento relativo a uno registre base***. Es el utilizado, por ejemplo, para acceder a variables estructuradas en las que un registro contiene la dirección de la variable y el desplazamiento es el correspondiente a los campos a los que se accede.
Otro tipo de direccionamiento que podemos simular en MIPS es el ***direccionamiento indirecto***. En este caso la dirección del dato en memoria se encuentra en un registro. Para tener este direccionamiento en MIPS el desplazamiento de los operadores de acceso a la memoria tiene que ser cero. Por ejemplo, `lw $s1, 0($s2)` lee el contenido de la posición de memoria que se encuentra en $s2. Un ejemplo de utilización es cuando el programador quiere acceder a una dirección calculada por programa o para seguir un puntero (del que hablamos en la práctica 6) o recorre variables estructuradas.

Otra manera de acceder a un dato en memoria es utilizando el ***direccionamiento absoluto***, en el que se accede a una posición de memoria indicando directamente la dirección en la propia instrucción. El MIPS no lo incorpora directamente porque las direcciones son de 32 bits y no caben en las instrucciones del MIPS que son también de 32 bits, pero podemos simularlo mediante las pseudoinstruccions `lw $rt, Etiq` y `sw $rt, Etiq`, donde *Etiq* es la etiqueta que hace referencia a la posición de memoria donde se ha definiendo previamente un dato.
En el siguiente ejemplo es muestra la utilización de los distintos modos de direccionamiento:

```
# Ejemplo de direccionamiento

            .data
            A:.word 6
            B:.word 8
            C:.space 4

            .text
            la $t0,A       #  En $t0 la dirección de A
            lw $t1,0($t0)  # Direccionamiento indirecto
(dirección en $t0)
            lw $t2,4($t0)  # Direccionamiento relativo (dirección
=$t0+4)
            add $t3,$t1,$t2
            sw $t3,C       # Direccionamiento absoluto
(dirección =C)
```

- ¿Cuántas pseudoinstrucciones contiene el código?
- Ensambla el código y observa la traducción de las pseudoinstrucciones en instrucciones del MIPS. ¿En qué instrucciones se ha traducido `sw $t3, C`? ¿Qué registro auxiliar se ha utilizado?

En el siguiente ejemplo se hace uso de los distintos modos de direccionamiento. El programa define una matriz cuadrada, la lee por columnas y muestra su transpuesta en la pantalla.

```
#Imprimir matriz transpuesta

.data
matriz: .byte   1,  4,  7,
                2,  5,  8
                3,  6,  9
columes: .word 3 # Numero de columnas

 .text
 la $t0, matriz
 lw $t3, columnas
 li $t2, 0      #iniciamos índice para recorrer matriz
 bucle:
           lb $t1, 0($t0)     #Fila 0
           move $a0, $t1
           jal imprim
           lb $t1, 3($t0)     #Fila 1 (1*3elementos)
           move $a0, $t1
           jal imprim
```

```
          lb  $t1, 6($t0)    #Fila 2 (2*3elementos)
          move $a0, $t1
          jal imprimir
          jal nuevalin
          addi $t2, $t2, 1 # incremento índice
          addi $t0, $t0, 1  #nueva columna
          bne $t2, $t3, bucle

li, $v0, 10
syscall

imprimir: li, $v0, 1
          syscall
          li $a0, '\t'
          li $v0, 11
          syscall
          jr $ra
nuevalin: li $a0,'\n'
          li, $v0, 11
          syscall
          jr $ra
```

- Identifica los distintos modos de direccionamiento en memoria utilizados: direccionamiento absoluto, direccionamiento indirecto y direccionamiento relativo.
- Ensambla el programa y comprueba si el resultado es correcto.
- ¿Qué cambios se tendrían que hacer si los elementos de la matriz se declarasen como palabras de 32 bits? Introdúcelos y comprueba su funcionamiento correcto.

Ejercicios a entregar

- Dado el siguiente código:

```
.data

vector: .word -4, 5, 8, -1
msg1: .asciiz "\n La suma de los valores positivos és = "
msg2: .asciiz "\n La suma de los valores negativos és = "

.text

Principal:

li $v0, 4 # Función para imprimir string
la $a0, msg1 # Leer la dirección de msg1
syscall
la $a0, vector # dirección del vector como parámetro
li $a1, 4 # Longitud del vector como parámetro

jal sum # Llamada a la función sum

move $a0, $v0 # Resultado 1 de la función
li $v0, 1
syscall # Imprimir suma positivos
li $v0, 4
la $a0, msg2
```

```
syscall
li $v0, 1
move $a0, $v1 # Resultado 2 de la función
syscall # imprimir suma negativos

li $v0, 10 # Acabar programa
syscall
```

Haz el código de la función ***sum*** que calcula la suma de los valores positivos y negativos del vector, dirección del cual se pasa como parámetro en $a0 y la longitud en $a1. La función devuelve en $v0 la suma de los valores positivos y en $v1 la suma de los negativos. Recuerda que en la función tienes que utilizar los registros t_j$.

- Haz el código que calcula la suma de los elementos de la diagonal principal de una matriz 4x4 de valores enteros introducida por teclado. Muestra la suma por pantalla.

Resumen

- Los elementos de las cadenas de caracteres son del tipo byte.
- Podemos acceder a la memoria utilizando diferentes modos de direccionamiento.
- El MIPS solo implementa el modo de direccionamiento relativo a registro base, pero se pueden simular los otros.