Pràctica 7
Vectors

7.1 Objectius

- Conèixer com es representen les cadenes de caràcters en llenguatge assemblador.
- Conèixer com es representen els vectors de nombres enters en llenguatge assemblador.
- Conèixer les maneres d'accedir a cadascun dels elements d'un vector.

7.2 Material

Simulador MARS i un codi font de partida

7.3 Desenvolupament de la pràctica

7.3.1 Introducció

Un vector (*array* o *string*) és un element que conté un conjunt de valors del mateix tipus. En l'assemblador del MIPS, un vector s'implementa emmagatzemant un conjunt de valors en posicions contigües de la memòria i accedint a cada valor mitjançant un desplaçament de l'adreça de començament del vector.

Un vector es podria emmagatzemar en qualsevol segment de dades (estàtic, dinàmic o pila) de la memòria del MIPS. La diferència és que, en l'estàtic o en la pila, la grandària del vector es fixa quan es compila el programa i en les dinàmic pot variar durant l'execució del programa. De moment estudiarem els vectors en el segment de dades estàtic.

7.3.2 Vectors de caràcters

Les cadenes de caràcters són també vectors amb dades de grandària un byte. El MIPS ens permet definir cadenes de caràcters en memòria amb les directives *.asciiz* i *.ascii*. Amb la directiva *.asciiz* es defineix una cadena de caràcters acabada amb el caràcter *null*, útil quan es vol recórrer-la, ja que ens permet reconèixer el seu final. S'utilitza la directiva *.ascii* quan es vol definir una cadena que no acabe amb *null*. Per exemple `.ascii "Universitat d'Alacant"` reserva 21 bytes consecutius de la memòria de dades iniciades amb aquests caràcters. Si volguérem utilitzar la instrucció `syscall` per a imprimir una cadena en la pantalla, necessàriament hauria d'acabar amb el caràcter *null*. Podem utilitzar *.ascii* amb una cadena molt llarga en la qual dividim la seua definició en diverses línies del programa, com mostra l'exemple següent:

```
    ###################################
    #                                 #
    # Codi exemple de l'activitat 1   #
    # Comptar caràcters d'una cadena  #
    #                                 #
    ###################################

.data         #Definim segment de dades
 str:.ascii "Estructures de"
     .asciiz "Computadors"

 .text        #Comença el programa

 la $s0, str
 add $s1, $zero, $zero  #Iniciem comptador a 0
 loop:
     add $t0, $s0, $s1    #adreça byte a examinar
     lb $t1, 0( $t0 )
     beq $t1, $zero, exit    # eixim si caràcter
                             # llegit='\0'
     addi $s1, $s1, 1     # increment comptador
     j loop

 exit: li $v0, 10
 syscall
```

7.3.2.1 Activitat 1

- Analitzeu el codi *Comptar caràcters d'una cadena* per a esbrinar què fa. Assembleu i executeu el codi. Quants caràcters té la cadena?
- En quina adreça de memòria es troba el caràcter *null*?
- Com s'actualitza l'índex del vector?

- Funcionaria el programa si la cadena sols constara del caràcter *null*?

Qüestió 1

- Modifiqueu el codi de l'activitat 1 perquè mostre per pantalla el missatge "El nombre de caràcters de la cadena és: " i a continuació el resultat.

Qüestió 2

- Modifiqueu el codi de l'activitat 1 perquè calcule el nombre de vegades que es repeteix la vocal *u*.

7.3.3 Vectors d'enters

Imaginem-nos que volem definir i operar amb un vector A d'enters de *n* elements, per a accedir a l'element i-èsim del vector l'expressem com a A[i]. Com ho podem fer en MIPS? Suposem que l'adreça del primer element del vector la guardem en un registre, per exemple $t0. Aquesta serà l'adreça base del vector. Per a accedir a qualsevol element del vector ho farem mitjançant un desplaçament respecte de l'adreça base. El que cal tenir amb compte és que un enter ocupa una paraula en memòria (4 bytes) i l'accés a la memòria del MIPS és per byte; per tant, el desplaçament sempre serà un múltiple de 4. Podem accedir a un element del vector en la memòria per a llegir-lo o escriure'l, per exemple: `lw $s0, 4($t0)` llegirà el segon element del vector.

Considereu l'exemple següent de codi:

```
        ###################################
        #                                 #
        # Codi exemple de l'activitat 2   #
        #  Recórrer un vector d'enters    #
        #                                 #
        ###################################

.data                 #Definim segment de dades
 A: .word 2, 4, 6, 8, 10 #Vector A iniciat amb valors
 B: .word 0:4            #Vector B buit
 C: .space 50            #Altra definició vector buit

 .text          #Comença el programa
  la $s0, A        # Adreça base del vector A
  la $s1, B        # Adreça base del vector B
  li $s5, 5           # Grandària del vector

  loop:
       add $t1, $s0, $t0
       add $t2, $s1, $t0
       addi $s2, $s2, 1  # Índex del vector
       lw $t3, 0($t1)
```

```
        sw $t3, 0($t2)
        sll $t0, $s2, 2   # Índex del vector x4
        bne $s2, $s5, loop

    li $v0, 10    #Acaba el programa
    syscall
```

En el programa s'han definit tres vectors de 5 elements, la grandària de cadascun dels elements és d'una paraula (4 bytes). El primer vector A s'ha definit amb valors inicials definits. El segon vector B s'ha definit indicant exclusivament la grandària. Sols utilitzem aquests dos en el codi. El tercer vector C s'ha definit reservant 50 bytes de la memòria (5 paraules de 4 bytes).

El codi copia els valors del vector A en el vector B i ho fa recorrent els dos vectors, cada vegada que llig un element d'A el còpia en B. Per a recórrer els vectors multiplica per 4 l'índex del vector en cada iteració.

7.3.3.1 Activitat 2

- Assembleu i executeu el programa *Recórrer un vector d'enters*. Comproveu que el vector A es copia en el vector B. En quina adreça comença el vector B?

- Per què no es poden acabar els vectors amb el caràcter *null* igual que es fa amb les cadenes de caràcters?

- En el programa es recorren els vectors actualitzant l'índex amb la instrucció *sll*. De quina altra manera es podrien recórrer els vectors?

Qüestió 3

- En el programa de l'activitat 2 s'ha utilitzat un bucle del tipus *do-while*; modifiqueu el programa perquè el bucle siga de tipus *for-while*.

Qüestió 4

- Modifiqueu el programa de l'activitat 2 perquè el vector B s'òmpliga amb enters llegits del teclat. Prèviament s'ha de mostrar un missatge per consola que demane els elements a introduir.

Qüestió 5

- Completeu el programa l'activitat 2 perquè el vector C s'òmpliga amb la suma dels elements del vector A i del B, és a dir, C[i]=A[i]+B[i] ∀i.

7.3.4 Adreçament a la memòria

En MIPS l'accés a una posició de la memòria per a llegir o escriure es fa, com ja hem vist prèviament, obtenint l'adreça mitjançant la suma del contingut d'un registre base i un desplaçament de 16 bits. Per exemple, `lw $s1, 4($s2)` llig el contingut de la posició de memòria obtinguda amb la suma $s2+4 i el guarda en $s1. A aquesta manera d'accedir a la memòria se la coneix amb el nom de *adreçament relatiu a un registre base*. És l'utilitzat, per exemple, per accedir a variables estructurades en les quals un registre conté l'adreça de la variable i el desplaçament és el corresponent als camps als quals s'accedeix.

Un altre tipus d'adreçament que podem simular en MIPS és *l'adreçament indirecte*. En aquest cas, l'adreça de la dada en memòria es troba en un registre. Per a tenir aquest adreçament en MIPS, el desplaçament dels operadors d'accés a la memòria ha de ser zero. Per exemple, `lw $s1, 0($s2)` llig el contingut de la posició de memòria que es troba en $2. Un exemple d'utilització és quan el programador vol accedir a una adreça calculada per programa o per a seguir un punter (del qual vam parlar en la pràctica 6) o recorre variables estructurades.

Una altra manera d'accedir a una dada en memòria és utilitzant *l'adreçament absolut*, en el qual s'accedeix a una posició de memòria indicant directament l'adreça en la mateixa instrucció. El MIPS no l'incorpora directament perquè les adreces són de 32 bits i no caben en les instruccions del MIPS que són també de 32 bits, però podem simular-lo mitjançant les pseudoinstruccions `lw $rt, Etiq` i `sw $rt, Etiq`, en què *Etiq* és l'etiqueta que fa referència a la posició de memòria on s'ha definint prèviament una dada.

En l'exemple següent es mostra la utilització dels diferents modes d'adreçament:

```
##################################
#                                #
# Codi exemple de l'activitat 3  #
#      Modes d'adreçament        #
#                                #
##################################

.data          #Definim segment de dades
A:.word 6
B:.word 8
C:.space 4
```

```
        .text          #Comença el programa
        la $t0,A       # En $t0 l'adreça de A
        lw $t1,0($t0)  # Adreçament indirecte
                       # (adreça en $t0)
        lw $t2,4($t0)  # Adreçament relatiu
                       # (adreça=$t0+4)
        add $t3,$t1,$t2
        sw $t3,C       # Adreçament absolut (adreça=C)
```

7.3.4.1 Activitat 3

- Analitzeu el codi *Modes d'adreçament*. Quantes pseudo-instruccions conté el codi?.
- Assembleu el codi i observeu la traducció de les pseudoinstruccions en instruccions MIPS. Amb quines instruccions s'ha traduït `sw $t3, C`? Quin registre auxiliar s'ha utilitzat?

En l'exemple següent es fa ús dels diferents modes d'adreçament. El programa defineix una matriu quadrada, la llig per columnes i mostra la seua transposada en la pantalla.

```
        ###################################
        #                                 #
        # Codi exemple de l'activitat 4   #
        #    Imprimir matriu transposada  #
        #                                 #
        ###################################


    .data               #Definim segment de dades
                        #Definim matriu 3x3 de bytes

     matriu: .byte   1,  4,  7,
                     2,  5,  8
                     3,  6,  9

     columes: .word 3   #Nombre de columnes

  .text                 #Comença el programa
   la $t0, matriu
   lw $t3, columnes
   li $t2, 0         #iniciem índex de recorre matriu
   bucle:
        lb $t1, 0($t0)       #Fila 0
        move $a0, $t1
        jal imprim           #Crida la funció imprim
        lb $t1, 3($t0)       #Fila 1 (1*3elements)
        move $a0, $t1
        jal imprim           #Crida la funció imprim
        lb  $t1, 6($t0)      #Fila 2 (2*3elements)
        move $a0, $t1
        jal imprim           #Crida la funció imprim
        jal nova_lin         #Crida la funció nova_lin
        addi $t2, $t2, 1     #Increment índex
```

```
        addi $t0, $t0, 1     #nova columna
        bne $t2, $t3, bucle

li, $v0, 10          #Acaba el programa
syscall

        ##################################
        #          Funcions              #
        ##################################

imprim:           #Funció imprim
        li, $v0, 1
        syscall
        li $a0, '\t'
        li $v0, 11
        syscall
        jr $ra      #Retorn de la funció

nova_lin:         #Funció nova línia
        li $a0,'\n'
        li, $v0, 11
        syscall
        jr $ra      #Retorn de la funció
```

7.3.4.2 Activitat 4

- Analitzeu el codi *Imprimir matriu transposada*. Identifiqueu els diferents modes d'adreçament a la memòria utilitzats en el programa: adreçament absolut, adreçament indirecte i adreçament relatiu.

- Assembleu el programa i comproveu si el resultat és correcte.

Qüestió 6

- Quins canvis s'haurien de fer en el codi de l'activitat 4 *Imprimir matriu transposada* si els elements de la matriu es declararen com a paraules de 32 bits? Es a dir, si la definició de la matriu fóra la següent:

```
  matriu: .word   1,  4,  7,
                  2,  5,  8
                  3,  6,  9
```

- Introduïu els canvis en el programa i comproveu que funciona correctament.

Qüestió 7

Donat el codi següent:

```
      ###################################
      #                                 #
      # Codi de partida de la qüestió 7 #
      #                                 #
      ###################################

.data

vector: .word -4, 5, 8, -1
msg1: .asciiz "\n La suma dels valors positius és = "
msg2: .asciiz "\n La suma dels valors negatius és = "

.text

Principal:

      li $v0, 4 # Funció per imprimir string
      la $a0, msg1 # llig adreça de msg1
      syscall
      la $a0, vector # Adreça vector com a paràmetre
      li $a1, 4 # Longitud del vector com a paràmetre

      jal sum # Crida a la funció sum

      move $a0, $v0 # Resultat 1 de la funció
      li $v0, 1
      syscall # Imprimir suma positius
      li $v0, 4
      la $a0, msg2
      syscall
      li $v0, 1
      move $a0, $v1 # Resultat 2 de la funció
      syscall # imprimir suma negatius

      li $v0, 10 # Acabar programa
      syscall

      ###################################
      #          Funcions               #
      ###################################

      sum: #Codi a implementar
```

- Analitzeu el codi de la qüestió 7 i feu el codi de la funció ***sum*** que calcula la suma dels valors positius i negatius del vector l'adreça del qual es passa com a paràmetre en $a0 i la longitud en $a1. La funció torna en $v0 la suma dels valors positius, i en $v1, la suma dels negatius. Recordeu que en la funció heu d'utilitzar els registres t_i$.

Qüestió 8

- Feu el codi que calcula la suma dels elements de la diagonal principal d'una matriu 4x4 de valors enters introduïda per teclat. Mostreu la suma per la pantalla.

7.4 Resum

- Els elements de les cadenes de caràcters són de tipus byte.
- Podem accedir a la memòria utilitzant diferents modes d'adreçament.
- El MIPS sols implementa el mode d'adreçament relatiu a registre base, però se'n poden simular d'altres.