Arquitectura del computador

Práctica 5 - 6

Aguilar Y. Bryan A., Abad L. Freddy L.

Escuela de Informática, Facultad de Ingeniería, Universidad de Cuenca, Ecuador

bryan.aguilar@ucuenca.edu.ec, freddy.abadl@ucuenca.edu.ec

Práctica 5: Modos de direccionamiento

Objetivos:

- Aprender los modos de direccionamiento de MIPS

Desarrollo:

- MIPS es una arquitectura de carga/almacenamiento. Sólo las instrucciones de carga y almacenamiento pueden acceder a memoria, según los cálculos efectuados sobre los valores en los registros.
- Casi todas las instrucciones de carga y almacenamiento operan sólo sobre datos alineados. La alineación de una cantidad significa que su dirección de memoria es un múltiplo de su tamaño en bytes. Por ejemplo, una palabra de 32 bits debe almacenarse en direcciones múltiplo de 4. Las instrucciones para alinear datos no alineados son: lwl, lwr, swl y swr.
- Hay varias formas de direccionar un dato en memoria (estudiar la transparencia nº 10 del tutorial). La forma en que se calcula esta dirección de memoria depende del modo de direccionamiento contemplado por la instrucción de acceso a memoria. A continuación se listan los modos de direccionamiento existentes:

Modos de direccionamiento

Formato	*Cálculo de la dirección*	*Ejemplo*
(registro)	Contenido del registro (cr)	`lw $t0,($t2)`
valor	Valor inmediato (vin)	`lw $t0,0x10010008`
valor (registro)	vin + cr	`lw $t0,0x10010000($t1)`
identificador	dirección del identificador (did)	`lw $t0,array`
identificador +/- valor	did +/- vin	`lw $t0,array+8`
identificador (registro)	did + cr	`lw $t0,array($t1)`
identificador +/- valor (registro)	did +/- vi + cr	`lw $t0,array+4($t1)`

- Hacer un programa que cargue el contenido de una posición de memoria en un registro, y luego imprima el valor del registro en consola. Repetir esta operación para cada uno de los 7 modos de direccionamiento listados anteriormente, y comprobar que en la consola aparece siempre el mismo valor (pues son modos distintos de acceder a un mismo dato).

```
.data
array: .word 2,4,0,5,1,8
salto: .asciiz "\n"
.text
.globl main
main:
        # Direccionamiento por registro
        la $t0,array
        lw $t0,0($t0)
        li $v0, 1 #Print
        add $a0 , $zero , $t0
        syscall

        li $v0 , 4
        la $a0 , salto
        syscall

        #Direccionamiento por valor
        lw $t0,268500992
        li $v0, 1 #Print
        add $a0 , $zero , $t0
        syscall

        li $v0 , 4
        la $a0 , salto
        syscall

        #Direccionamiento por valor (registro)
        la $t0,array
        lw $t0,8($t0)
        li $v0, 1 #Print
        add $a0 , $zero , $t0
        syscall

        li $v0 , 4
        la $a0 , salto
        syscall

        #Direccionamiento por identificador
        lw $t0,array
        li $v0, 1 #Print
        add $a0 , $zero , $t0
        syscall

        li $v0 , 4
        la $a0 , salto
        syscall
```

```
#Direccionamiento +/- valor
lw $t0,array+8
li $v0, 1 #Print
add $a0 , $zero , $t0
syscall

li $v0 , 4
la $a0 , salto
syscall

#Direccionamiento por identificador (registro)
la $t0,20
lw $t0,array($t0)
li $v0, 1 #Mando a Imprimir
add $a0 , $zero , $t0
syscall

li $v0 , 4
la $a0 , salto
syscall

#Direccionamiento por identificador +/- valor
#registro
la $t0,4
lw $t0,array+12($t0)
li $v0, 1 #Mando a Imprimir
add $a0 , $zero , $t0
syscall
```

Práctica 6: Condiciones if-then-else

Objetivos:

- Cómo implementar la condición if-then.
- Cómo implementar la condición if-then-else

Desarrollo:

A. Sea el siguiente código C:

```
#include <iostream.h>

main() {
  int a[] = {36, 20, 27, 15, 1, 62, 41};
  int n = 7;
  int max, i;

  for (max = i = 0; i<n; i++)
    if (a[i] > max)
      max = a[i];

  cout << max << endl;
}
```

Este programa calcula el máximo de los elementos del array a, almacenando en la variable max. Estudiar el código ensamblador equivalente:

```
# PRAC_6A.S
        .text
        .globl main
main:
        li $t0, 0                       # i en $t0, t0=0
        li $s0, 0                       # max en $s0,s0=0
        lw $s1, n                       # n en $s1, cargo en s1, el valor de n
m1:     bge $t0, $s1, m3                #salta a m3 si t0>=s1
        mul $t1, $t0, 4                 # Dá el valor adecuado para i
        lw $t2, a($t1)                  # Carga a[i] en $t2
        ble $t2, $s0, m2                # Salta el "then" si a[i] <= max
        move $s0, $t2                   # "then": max = a[i]
m2:     addi $t0, $t0, 1                # i++
        b m1                            # ejecuta m1
m3:     move $a0, $s0                   # Fin del bucle - copia s0 en a0
        li $v0, 1                       # v0=1
        syscall
        li $v0, 10                      #v0=10
        syscall
        .data
a:      .word   36, 20, 27, 15, 1, 62, 41
n:      .word   7
max:    .word   0
```

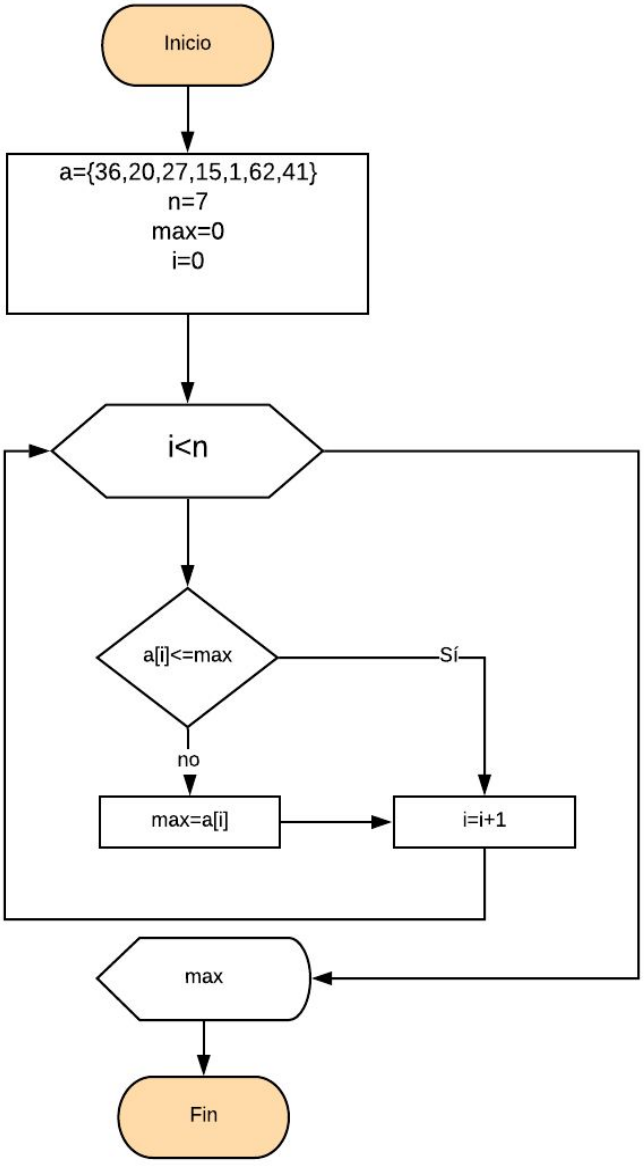
1. Estudiar en profundidad cómo trabaja el código. El resultado debe aparecer en consola: 62.

   El código representado en C y en formato Assembler, refiere a un simple proceso de elegir el máximo número de una lista, mediante una iteración simple a toda la lista, una por una se verifica cual es el valor máximo. El proceso se puede decir es el más básico y fundamental para encontrar maximos de una lista. Su proceso atraviesa por un bucle for y en el caso de mips, dos condicionales if, que verifiquen la posición en la lista y el número máximo.

2. Dibujar un organigrama o diagrama de flujo que responda a las operaciones descritas

Organizacion y Arquitectura de Computadoras
Práctica 4

Freddy L. Abad L. & Bryan A. Aguilar Y.



B. Sea el siguiente código C

```c
#include <iostream.h>

main() {
  int a[] = {-36, 20, -27, 15, 1, -62, -41};
  int n = 7;
  int i;
  int npos, nneg;

  for (i = npos = nneg = 0; i<n; i++)
    if (a[i] > 0)
      npos++;
    else
      nneg++;

  cout << "+: " << npos << "; -: " << nneg << endl;
}
```

Este código cuenta, en las variable npos y nneg, cuántos números positivos y negativos, respectivamente, aparecen en el array a:

ractica 5 - 6 Abad Aguilar ▸ Practica6B.asm

```
2           .text
3           .globl main
4   main:
5           li          $t0, 0              # i en $t0, t0=1
6           li          $t1, 0              # npos en $t1, t1=0
7           li          $t2, 0              # nneg en $t2, t2=0
8           lw          $s1, n              # n en $s1 , sq=n
9   m1:     bge         $t0, $s1, m4        # salta a t0 si s1<=m4
10          mul         $t3, $t0, 4         # da el valor adecuado para i
11          lw          $t4, a($t3)         # carga a[i] en $t4
12          bgez        $t4, m2             # if (a[i] > max)...
13          addi        $t2, $t2, 1         # "else": incrementa nneg
14          b m3                            # salta sobre "then"
15          m2: addi    $t1, $t1, 1         # "then": incrementa npos
16          m3: addi    $t0, $t0, 1         # i++
17          b m1                            #salta a m1
18  m4:     move        $a0, $t1            # imprime npos
19          li          $v0, 1              #carga en v0=1
20          syscall
21          la          $a0, endl           # carga en a0 salto de linea
22          li          $v0, 4              # v0=4
23          syscall
24          move        $a0, $t2            # imprime nneg
25          li          $v0, 1              # v0=1
26          syscall
27          li          $v0, 10             # v0=10
28          syscall
29          .data
30  a:      .word       -36, 20, -27, 15, 1, -62, -41
31  n:      .word       7
32  max:    .word       0
33  endl:   .asciiz     "\n"
```

1. Estudiar en profundidad cómo trabaja el código. Los resultados que deben aparecer en consola son npos=3 y nneg=4.

   Este código únicamente realiza un conteo de números positivos y negativos, lo realiza mediante un bucle for que itera hasta llegar al número equivalente al tamaño de la lista, en cada iteración realiza una comparación mediante un if, segun esto incrementa determinado contado (ya sea npos o nneg). Procediendo finalmente a imprimir estos contadores.

2. Dibujar un organigrama o diagrama de flujo que responda a las operaciones descritas.