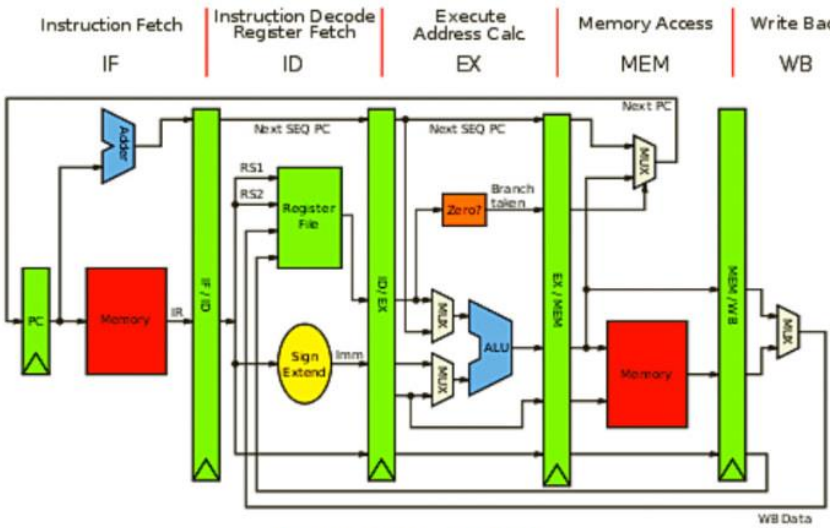
EC

ENTREGA 3

COMPUTER ARCHITECTURE



MIPS

Realizado por: Jaime Aznar Espinosa

Índice

1. Objetivos

El objetivo principal de la práctica 7 es conocer el funcionamiento de los vectores, desde el acceso a los diferentes elementos del vector dependiendo del tipo de dato y también como se representan las cadenas de caracteres.

Las prácticas 8 y 9 comprenden el temario que corresponde a los números en coma flotante, con el objetivo de conocer la unidad en coma flotante de MIPS es decir la FPU, y conocer los registros estándar de la coma flotante para el paso de parámetros y cálculo de operaciones.

2. Práctica 7

2.1 Cuestión 1

```
        ###################################
        #                                 #
        # Código ejemplo de la actividad 1 #
        # 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
```

- Modifica el código de la actividad 1 para que muestre por pantalla el mensaje "El número de caracteres de la cadena es: " y a continuación el resultado.

Simplemente añadimos un módulo a continuación que haga print del número:

```
imprim: la $a0, out
        li $v0, 4
        syscall
        move $a0, $s1
        li $v0, 1
        syscall
        j exit
```

Teniendo un *string* que acabe en carácter nulo, en este caso "out":

```
out: .asciiz "El número de caracteres es: "
```

Que tiene como resultado:

```
El número de caracteres es: 29
-- program is finished running --
```

2.2 Cuestión 5

```
        ###################################
        #                                 #
        # Código ejemplo de la actividad 2 #
        # 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                #Acaba el programa
  syscall
```

- Completa el programa de la actividad 2 para que se rellene el vector C con la suma de los elementos del vector A y del B (C[i]=A[i]+B[i] ∀i).

No veía mucho punto en sumar un vector vacío con uno que no lo estuviera, con lo que primero he rellenado el vector B, y luego he hecho lo que pedía el ejercicio:

```
.text
la $s0, A
la $s1, B
la $s2, C
li $s5, 5
```

```
rellena_B:
        addi $s3, $s3, 1 #contador
        add $t2, $s1, $t0 #desplazar el vector B tantas palabras como numeros llevemos introducidos
        la $a0, intro
        li $v0, 4
        syscall
        li $v0, 5
        syscall
        move $t1, $v0
        sw $t1, 0($t2) #introducir en el vector mediante el puntero
        sll $t0, $s3, 2 #desplazarnos 2 bits, es decir el tamaño de palabra
        bne $s3, $s5, rellena_B #volver a hacer el modulo si no hemos hecho 5
        j loop #saltar al bucle de suma
```

```
loop:
        #carga de la posicion en el vector
        add $t1, $s0, $t0
        add $t2, $s1, $t0
        add $t3, $s2, $t0

        addi $s4, $s4, 1 #incrementar el contador

        lw $t4, 0($t1) #cargar palabra de A
        lw $t5, 0($t2) #cargar la palabra de B

        add $t6, $t4, $t5 #suma de A+B
        sw $t6, 0($t3) #introduccion en C

        sll $t0, $s4, 2
        bne $s4, $s5, loop

li $v0, 10
syscall
```

La manera que he tenido de comprobar que funciona es posteriormente haciendo un print de todo el vector C.

En los registros aparecen desordenados, sin embargo, al recorrerlo están en orden. He adjuntado los archivos por si requiere revisión.

2.3 Cuestión 7

```
            ##########################################
            #                                        #
            #    Código de partida de la cuestión 7  #
            #                                        #
            ##########################################

.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
            ####################################
            #            Funciones             #
            ####################################

            sum: #Código a implementar
```

- Analiza el programa de la cuestión 7 i escribe 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 $v1 la suma de los negativos. Recuerda que en la función tienes que utilizar los registros t_j$.

La función suma tiene dos variantes, una si el numero es positivo y otra si es negativo, en las cuales tendremos que sumar en las variables $v0 y $v1 y volver a la etiqueta de suma, si el salto condicional no se produce entonces devolveríamos el control al programa que estaba en jal sum, donde se hará el print de los datos:

```
sum:
        add $t0, $a0, $t1
        addi $s1, $s1, 1
        lw $s2, 0($t0)
        #si es un numero positivo lo llevo a su modulo
        bgez $s2, sumarPos
        #sino sera negativo
        b sumarNeg

#aqui aumento la variable v0 y desplazo el vector
sumarPos:
        add $v0, $v0, $s2
        sll $t1, $s1, 2
        bne $s1, $a1, sum
        jr $ra

#lo mismo con la variable v1
sumarNeg:
        add $v1, $v1, $s2
        sll $t1, $s1, 2
        bne $s1, $a1, sum
        jr $ra
```

```
La suma de los valores positivos es = 10
La suma de los valores negativos es = -5
-- program is finished running --
```

3. Práctica 8

3.1 Cuestión 4

- Escribe el código que haba que el contenido de $f1 sea el valor 1 en coma flotante y el de $f2 el valor -2 en coma flotante utilizando las instrucciones de conversión de tipo.

Tenemos que mover los valores 1 y -2 al coprocesador 1 donde se encuentran los registros en coma flotante, después hacer la conversión de tipo con la operación cvt.s.w , por último hacer un print de comprobación en el registro $f12.

```
.text
li $t1, 1
li $t2, -2

#movemos ambos valores al coprocesador1
mtc1 $t1, $f0
mtc1 $t2, $f1
#los cambiamos de entero a float en simple precision
cvt.s.w $f1, $f1
cvt.s.w $f0, $f0

#los floats tienen el print en f12
mov.s $f12, $f0
li $v0, 2
syscall


li $a0, '\n'
li $v0, 11
syscall

mov.s $f12, $f1
li $v0, 2
syscall

li $v0, 10
syscall
```

```
1.0
-2.0
-- program is finished running --
```

3.2 Cuestión 5

- Completa el código de la actividad 6 para que muestre en consola un mensaje de error por desbordamiento. Comprueba que funciona correctamente.

Lo que se realiza en el código de la actividad 6, es primeramente extraer la máscara, y posteriormente extraer el exponente, desplazando 23 lugares a la derecha para que quede exactamente en el bit 0 el primer 1. Posteriormente se compara este número con el exponente (definido en el .data y correspondiente a todo 1s), si coinciden, es decir si es todo a unos, tendríamos un overflow, y tendríamos que saltar a la etiqueta exp_a_1.

```
desborde: .asciiz "Se ha desbordado"
```

```
beq $t2, $t3, exp_a_1 #con el desplazamiento de 23, en t2 se quedan 0x000000ff

li $v0, 10
syscall

exp_a_1: la $a0, desborde
        li $v0, 4
        syscall

        li $v0, 10
        syscall
```

3.3 Cuestión 8

```
            ####################################
            #                                  #
            #Código de partida de la cuestión 8#
            #                                  #
            ####################################

.data

 array: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
 long: .word 10
 suma :.word 0
```

- Haz el código que suma los elementos del vector y calcula el valor medio en coma flotante. Muestra el resultado por la consola.

```
.text

lw $s1, long
lw $s2, suma
la $s0, array

sumatorio:
        #movimiento dentro del vector
        add $t1, $s0, $t0
        addi $s3, $s3, 1
        lw $t2, 0($t1) #acceso a palabra
        add $s2, $s2, $t2
        #desplazamiento en x palabras
        sll $t0, $s3, 2
        bne $s3, $s1, sumatorio
        j media


media:
         mtc1 $s2, $f0 #mover al coprocesador 1 la suma
         mtc1 $s1, $f1 #mover al coprocesador 1 la longitud
         cvt.s.w $f0, $f0 #cambiamos a simple precision
         cvt.s.w $f1, $f1 #cambiamos a simple precision
         div.s $f12, $f0, $f1 #division
         li $v0, 2 #mostrado
         syscall

         li $v0, 10
         syscall
```

Tenemos que recorrer el vector de la misma manera que en la práctica 7, lo único que debemos tener en cuenta es que la media es un valor flotante, por lo que tendremos que pasar la longitud y la suma a registros flotantes y mostrar el resultado:

```
5.5
-- program is finished running --
```

4. Práctica 9

4.1 Cuestión 1

- ¿Cuál es la razón por la que el registro base de las instrucciones *lwc1* y *swc1* pertenecen al banco de registros de enteros y no de la FPU?

Porque al ser acceso a memoria, esta memoria se carga mediante la instrucción load address que acepta únicamente registros del banco de registros de enteros, con lo que tendremos que acceder a esa posición mediante dichos registros y moverlos al FPU mediante estas instrucciones.

4.2 Cuestión 4

```
            ####################################
            #                                  #
            #Código de partida de la cuestión 4#
            #                                  #
            ####################################

.data

 Array: .float 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
 long: .word 10
 Suma: .float 0
```

- Haz el código que suma los elementos del vector y calcula el valor medio. Muestra el resultado por la consola.

Se haría de la misma manera que en la práctica anterior, pero usando las operaciones para cargar valores flotantes.

```
la $s0, Array
lw $s2, long


bucle_suma:
        #movimiento dentro del vector
        add $t1, $s0, $t0
        addi $s3, $s3, 1
        lwc1 $f0, 0($t1) #acceso a float
        add.s $f1, $f1, $f0 #guardamos todo en el auxiliar
        #desplazamiento en x palabras
        sll $t0, $s3, 2
        bne $s3, $s2, bucle_suma
        j media

media:
        swc1 $f1, Suma #guardar resultado en la etiqueta
        mtc1 $s2, $f2 #mover la longitud del vector a float
        cvt.s.w $f2,$f2 #conversion de tipo
        div.s $f12, $f1, $f2 #division float
        la $a0, out
        li $v0, 4
        syscall
        li $v0, 2 #mostrar dato
        syscall

        li $v0, 10
        syscall
```

```
El resultado del calculo de la media del vector es: 5.5
-- program is finished running --
```

4.3 Cuestión 6

- Implementar la función *max* que nos devuelve el valor mayor de dos números en coma flotante. Los argumentos se pasan según convenio en $f12 y $f14 y el resultado se devuelve en $f0. Utilizad el siguiente código de partida:

```
#####################################
#                                   #
#Código de partida de la cuestión 6#
#                                   #
#####################################

.data
Xpide:      .asciiz "X = "
Ypide:      .asciiz "Y = "
MaxRes:     .asciiz "El mayor es "
.text

la $a0, Xpide
li $v0,4
syscall
li $v0,6
syscall
mov.s $f12,$f0

la $a0, Ypide
li $v0,4
syscall
li $v0,6
syscall
mov.s $f14,$f0

jal max

la $a0,MaxRes
li $v0,4
syscall
mov.s $f12,$f0
li $v0,2
syscall

li $v0,10
syscall
```

```
max:
        #mover a registros para trabajar con ellos
        mov.s $f1, $f12
        mov.s $f2, $f14
        c.le.s $f1, $f2
        bclt set_B
        bclf set_A

set_A: mov.s $f0, $f1
        jr $ra

set_B: mov.s $f0, $f2
        jr $ra
```

Lo único que tenemos que hacer es utilizar la instrucción de comparación para devolver un valor o el otro, y devolver el control al caller en jal.

5. Conclusiones

La práctica ha servido para conocer los accesos a memoria en vectores tanto en enteros como en flotantes, muestra también como operar con strings de .asciiz para ayudar con la claridad del programa, así como los tipos de desplazamiento en vectores y las operaciones en coma flotante, tanto condicionales como aritméticas.