Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

Laboratorio Organización de computadores

Catalina Isidora Riquelme Zamora

Profesor: Carlos González
Ayudante: Ricardo Hasbun
Sección: L-4

Octubre 2021
Santiago - Chile

Contenido

Introducción

A lo largo del laboratorio se probarán las diversas predicciones realizadas en el Pre-Laboratorio, probando las hipótesis para los diversos programas presentados. Por otra parte, se realizarán diversos programas en MIPS con la finalidad de predecir el funcionamiento de un programa MIPS de manera correcta, además de lograr traducir diversos programas desde un alto nivel a lenguaje ensamblador, además de lograr escribir programas que puedan usar instrucciones aritméticas, salto y memoria, finalmente usar MARS para escribir, ensamblar y depurar programas en MIPS.

Parte 1: Probar las predicciones del Pre-Laboratorio en MARS

¿Los valores de los registros son como esperabas para el programa 1?

Se confirma lo planteado en el Pre-Laboratorio que los valores tomados por los registros son -8 y -4 para $t1 y $t0 respectivamente.

¿Los valores de los registros son como esperabas para el programa 2?

Al momento de ensamblarse existe un error debido que no se encuentra definida la subrutina A, en el supuesto que esta línea no existiera para cuando el valor del registro $t2 inicie en 2 esto no pasaran por la subbruta, de manera que el registro $t0, $t1 y $t2 respectivamente toman el valor de 2, 1 y 2. Por el contrario, suponiendo que existiera la subrutina para los casos en donde $t2 sea igual al valor de 0, se pasa por la subrutina el valor de $t0, dependiente de lo que suceda en A y $t1 será 1 ya que es definido como tal luego de pasar por la subrutina A.

¿Qué pasaría si se cambia la línea con beq a: “beq $t0, $t0, A”?

Al cambiar la línea a “beq $t0, $t0, A” provoca que se pase por la subrutina A y por lo tanto no deja ensamblar, pero suponiendo que se pudiese tanto $t0 como $t2 serían dependientes de esta subrutina y $t1 seria 1.

Parte 2: Escribir programas en MIPS

Programa 1

int x = 20; // usar $t0 para almacenar los valores de x

int y = x+5; //usar $t1 para almacenar los valores de y

int z = (x+4)-(y-9); //Usar $t2 para almacenar el valor de z

Para traducir el código en instrucciones MIPS, se almacenan en los registros $t0, $t1 y $t2 los valores de x, y y z respectivamente, por otra parte, s registros temporales $t3 y $t4 se utilizan para almacenar las operaciones (x+4) y (y-9) para luego ser trasladado al registro $t2.

```
1   .data
2   .text
3           #Se asigna 20 a $t0 (x) | 20
4           addi $t0, $zero, 20
5           #Se asigna 5 a $t1 (y) | x + 5
6           addi $t1, $t0, 5
7           #Se asigna 4 a $t3 | x +4
8           addi $t3, $t0, 4
9           #Se asigna -9 a $t4 | y-9
10          addi $t4, $t1, -9
11          #Se asignan los valores | (x+4)-(y-9)
12          sub $t2, $t3, $t4
```

Imagen 1: program1.s

Programa 2

int x = 1; // usar $t3 para almacenar valores de x

int y = -1; // usar $t4 para almacenar valores de y

int z = x+y // Seleccione un registro para guardar este valor

if (z == 0) {

    y++;

} else if (z == 1) {

    y--;

} else {

    y = 100;

}

En primera instancia se almacenan en $t1, $t4 y $t0, respectivamente, las variables x,y y z, para posteriormente definir los lebel If y Elseif, los cuales presentan la operación y++ e y—respectivamente.

```
.data
.text

    li $t3,-1 #Se asigna x a $t3
    li $t4,1 #Se asigna y a $t4
    add $t0,$t3,$t4 #Operación para calcular z

    #operacion condicional if
    li $t1,0
    li $t2,1
    beq $t0,$t1,if #If z == 0
    beq $t0,$t2,elseif #Elseif z == 1
    li $t4,100 #Else

    #Finaliza
    li $v0,10
    syscall

    #En el caso que se cumpla if
    if:
       addi $t4,$t4,1
       li $v0,10
       syscall

    #En el caso que se cumpla elseif
    elseif:
       sub $t4,$t4,1
       li $v0,10
       syscall
```

Imagen 2: program2.s

Programa 3

int[] a = new int[4];

// traduzca solo el código bajo esta línea.

// Asuma que el arreglo comienza en la dirección de memoria 0x10010000

a[0] = 3;

a[3] = a[0] - 1;

En primera instancia se declara un arreglo de 4 elementos, para posteriormente asignarle el valor de 0 a $t0 y a $t1 el valor 3, asignando el primer elemento del arreglo 3, para luego restarle 1 y guardar este resultado en la posición 3 del arreglo.

```
1  .data
2          #Se dereclarra un arreglo de 4 elementos
3          arr: .space 16
4  .text
5          #declaramos el indice
6          add $t0, $zero, $zero
7          #Se declara a[0]
8          addi $t1, $zero, 3
9          sw $t1, arr($t0)
10         #Se aplica operación
11         lw $t2, arr($t0)
12         addi $t2, $t2, -1
13
14         #El resultado se agrega a a[3]
15         addi $t0, $zero, 12
16         sw $t2, arr($t0)
17
```

Imagen 3: program3.s

Programa 4

int a = 2; // usar $t6 para almacenar valores de a

int b = 10; // usar $t7 para almacenar valores de b

int m = 0; // usar $t0 para almacenar valores de m

while (a > 0) {

m += b;

a -= 1; }

En primera instancia se asignan los valores de a, b y m a los registros $t6, $t7 y $t0, respectivamente, posteriormente se encuentran dos label While y Exit, donde el primero simula un ciclo iterativo a través de j, además por medio de blez se verifica que el valor analizado no sea superior a 0, en el caso contrario será dirigido al label Exit. Dentro de este ciclo While se realiza la operación m+=b y a-=1 a través de add y addi. Por otro lado, el label Exit asigna un valor a $v0 y ejecuta syscall terminando la ejecución del programa.

```
1  .data
2  .text
3          main:
4          #Se asigna a a $t6
5          addi $t6, $zero, 2
6          #Se asigna b a $t7
7          addi $t7, $zero, 10
8          #Se asigna m a $t0
9          addi $t0, $zero, 0
10         j While
11
12         #Label While
13         While:
14                 blez $t6, Exit
15                 add $t0, $t0 , $t7
16                 addi $t6, $t6, -1
17                 j While
18
19         #Label Exit
20         Exit:
21                 addi $v0, $zero, 10
22                 Syscall
23
```

Imagen 4: program4.s

Programa 5

int[] arr = {11, 22, 33, 44};

arrlen = arr.length; // traducción de lo de arriba está dada

// complete la traducción de lo siguiente...

int evensum = 0; // usar $t0 para valores de evensum

for (int i=0; i<arrlen; i++) {

    if (arr[i] & 1 == 0) { // ¿Qué significa esta condición?

        evensum += arr[i];

    }

}

Para realizar el programa se establece un arreglo de 4 elementos, los cuales ya se encuentran definidos. Posteriormente, se utiliza el valor de $s1 el cual fue asignado a $t3 como contador tanto para el label For como para el label Exit. Para el label For se confirma que el valor sea superior a 0, para posteriormente realizar la condición if (arr[i] & 1 == 0), la cual hace referencia a un comparativo bit a bit de números binarios provocando que el resultado de este sea un numero compuesto por los bits donde exista una coincidencia en 1, se compara el numero a analizar junto a 1 siendo esto igual al valor 0 de esta manera serán aceptados únicamente los pares. Para lograr esto se genera una división para revisar, mezclando una contante, junto con el valor 2 y el elemento del arreglo

para revisar el valor contenido en el registro hi, en este se almacena el resto de la división, en el label If se acumulan la suma de los números pares en $t1, para luego avanzar al ciclo For evalua si el arreglo se encuentra vacio de ser así saltará al label label Exit, el cual finaliza el programa.

```
.data
        arr: .word 10 22 15 40
        end:
.text
        main:
                addi $s2, $zero, 2 #Se establece una constante para divir
                la $s0, arr
                la $s1, end
                subu $s1, $s1, $s0
                srl $s1, $s1, 2
                add $t3, $zero, $s1
                j For

        For:
                blez $t3, Exit #En el caso que el contador llegue 0, finaliza
                lw $t0, 0($s0)
                div $t0, $s2
                mfhi $s4
                addi $t3, $t3, -1
                addi $s0, $s0, 4
                beqz $s4, If
                j For

        If:
                add $t1, $t1, $t0
                j For

        Exit:
                addi $v0, $zero, 1
                Syscall
```

Imagen 5: program5.s

Resultados

Programa 1

$t0	8	20
$t1	9	25
$t2	10	8
$t3	11	24
$t4	12	16

Imagen 6: registros program1.s

Problema 2

$t0	8	0
$t1	9	0
$t2	10	1
$t3	11	-1
$t4	12	2
$t5	13	0

Imagen 7: registros program2.s

Problema 3

$t0	8	12
$t1	9	3
$t2	10	2

Imagen 7: registros program3.s

Problema 4

$v0	2	10
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	20
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	10

Imagen 8: registros program4.s

Problema 5

$at	1	268500992
$v0	2	1
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	40
$t1	9	72
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	268501008
$s1	17	4
$s2	18	2
$s3	19	0

Imagen 9: registros program5.s

Conclusión

Tras la realización del laboratorio se puede concluir que tras la ejecución de los programas del pre-laboratorio las predicciones se encontraban acertadas, dando los resultados en los registros tal y como fueron planeadas. Por otro lado, se cumplieron los objetivos propuestos para esta experiencia logrando traducir diversos programas desde un alto nivel de programación a un lenguaje ensamblador, además de lograr escribir programas que puedan usar instrucciones aritméticas, salto y memoria, finalmente usar MARS para escribir, ensamblar y depurar programas en MIPS.