Práctica 5
Estructuras de control

Objetivos

- Entender las necesidades de que existan instrucciones para romper la ejecución secuencial de los programas.
- Conocer la traducción del lenguaje ensamblador de las estructuras de control básicas de la programación estructurada.

Materiales

Simulador MARS y un código fuente de partida

Desarrollo de la práctica

1. Introducción

En un programa MIPS la orden de ejecución de las instrucciones es secuencial, el registro Contador de Programa (PC) contiene la dirección donde se encuentra la siguiente instrucción a ejecutar. Este valor se incrementa automáticamente con 4 después de cada ciclo de instrucción cómo habéis podido comprobar en las prácticas precedentes.
Sin embargo, hay veces que puede interesar saltar y ejecutar instrucciones que no se encuentran en la secuencia normal del programa. Habéis visto ya que el MIPS proporciona instrucciones para poder hacerlo como la jal o la j, son lo que se denominan saltos incondicionales, es decir, instrucciones que siempre que se ejecutan realizarán un salto modificando el contenido del PC. A parte de estas instrucciones, el MIPS nos proporciona otras que nos permiten romper la secuencia normal del programa dependiendo de si se cumple o no cierta condición. Son las llamadas instrucciones de salto condicional. Combinando los dos tipos de instrucciones de salto (condicional e incondicional) podemos programar estructuras de control complejas, lo que permite a los compiladores traducir sentencias de la programación en alto nivel como if-then-else, while, for-loop.

La estructura if-then-else es probablemente la más utilizada por los programadores. Veamos un ejemplo de su traducción en ensamblador. Suponed el siguiente fragmento de código en C:

```
if (a != b)
      f = g + h;
```

Se quiere ejecutar la suma si *a* es distinto de *b*, o lo que es lo mismo, si *a* es igual a *b* no se quiere hacer la suma. Esta sentencia se puede escribir en el ensamblador del MIPS utilizando la instrucción *beq* (*branch if equal*) de la siguiente forma:

```
            beq $t1, $t2, Final         # si a=b salta a Final
            add $t3, $t4, $t5           # f = g + h
 Final:     # Otras instrucciones
```

La instrucción *beq* es un ejemplo de instrucción de salto condicional. Para que se pueda hacer el salto se ha de satisfacer la condición de igualdad entre los registros indicados, en otro caso el programa continua ejecutándose en el orden normal. En el ejemplo, si no se satisface la condición (`$t1 = $t2`) se ejecuta la siguiente instrucción en secuencia que es el *add*.

2. Como hacer uso de las instrucciones de salto condicionales

A continuación estudiaremos como traducir a ensamblador condiciones lógicas simples utilizadas en la programación a alto nivel.
Vamos a hacer un programa que lea un entero del teclado y escriba en la consola su valor absoluto. Para obtener el valor absoluto se utilizará una estructura de control del tipo if-then-else. La estructura que tendrá el programa será:

```
            Lee el valor (Digamos A)
            Si (A ≥ 0) ir a etiq
            Hacer A = –A
   etiq:    Imprimir A
```

Para la condición $A \geq 0$ utilizaremos la instrucción de salto condicional *bgez* (*branch if greater than or equal to zero*). El programa quedaría:

```
            # Cálculo del valor absoluto de un entero

            .text

            li $a0, '>'
            li $v0,11           #Indicación de escribir un valor
            syscall
            li $v0,5
            syscall             #Leer el entero A

            bgez $v0, else      # Si (A ≥ 0) salta a else
            sub $a0, $zero, $v0 # En $a0 el negativo de A
            j exit              #Acaba parte if-then

else:       move $a0, $v0       # En $a0 el valor A

exit:       li $v0, 1           #Imprimir lo que hay en $a0
            syscall
            li $v0, 10          #Acaba el programa
            syscall
```

- Escribid, ensamblad y comprobad el funcionamiento del código.
- Observad en la ventana *registers* como varía el contenido del registro PC.
- La instrucción *bltz* (*branch if less than zero*) salta si el valor es menor que cero y es similar a *bgez* ya que también compara un registro con 0 pero siendo contraria la condición de salto. Cambiad la instrucción *bgez* por *bltz*, ¿Qué modificaciones tendríais que hacer en el código?
- Comprobad que el nuevo código con *bltz* opera correctamente.

En el repertorio de instrucciones del MIPS hay seis condiciones para los saltos condicionales con los cuales se pueden hacer tres pares de condiciones (= y ≠, > y ≤, < y ≥). Con las dos primeras condiciones (= y ≠) se han de especificar dos registros (por ejemplo: `beq rs, rt, etiqueta`), son las instrucciones *beq* y *bne*. Para el resto de las condiciones sólo se ha de especificar un registro en la instrucción ya que la comparación se hace con el registro $cero que siempre contiene un 0 y no hay que indicarlo (por ejemplo: `bltz rs, etiqueta`), son las instrucciones *bgtz*, *blez*, *bltz*, *bgez*.

¿Qué ocurre si queremos hacer un salto con una condición entre dos registros distinta de la igualdad o desigualdad? El MIPS no nos proporciona las instrucciones de salto que nos permiten hacerlo. Lo que sí nos proporciona son instrucciones de comparación que podemos utilizar combinándolas con las instrucciones de salto mencionadas para nuestros propósitos. Por ejemplo, la instrucción *slt* (*set on less than*) tiene la siguiente forma: `slt rd,rs,rt`. Esta instrucción compara dos registros (*rs* y *rt*) y pone un 1 en el registro *rd* si *rs* se menor que *rt*, en caso contrario introduce en *rd* un 0.
Imaginad que queremos traducir en el ensamblador la siguiente sentencia en pseudocódigo:

```
if $s2 ≥ $s1 then
      $s3 = 0
```

Lo haremos utilizando las instrucciones *slt* y *beq*:

```
            slt $t0,$s1,$s2            #$t0=1 si $s1<$s2
            beq $t0,$zero,salta        #Salta si falla condición
            add $s3,$zero,$zero        $s3=0

salta:
```

Ejercicios

- Escribid el programa que lea dos enteros del teclado y escriba en la consola el mayor. La estructura será:

```
      Leer el primer valor (A)
      Leer el segundo valor (B)
      Si (A<B) ir a eti
      Imprimir A
      Ir a acabar:
eti:  Imprimir B
acabar:
```

- Ensambla y prueba el programa con distintos valores de A y B.

3. Codificación en lenguaje máquina

Las instrucciones de salto condicional se codifican en binario siguiendo el formado tipo I para almacenarlas en la memoria. A diferencia de las de salto incondicional, no señalan la dirección a la que se saltará sino que indican el desplazamiento en bytes hasta donde se quiere llegar; este desplazamiento se añadirá al valor que haya almacenado en el PC para conseguir la dirección correcta.

- Observa el programa ensamblado que acabas de escribir y rellena los distintos campos en binario de la instrucción *beq* según el formato tipo I, escribe primero la instrucción que vas a codificar.

  Instrucción:

Codigo Operació (6 bits)	Rs (5 bits)	Rt (5 bits)	Desplazamiento

Las instrucciones de comparación son instrucciones aritméticas que implican a tres registros, siguen por lo tanto el formato R.

- Observa de nuevo el programa ensamblado que acabas de escribir y rellena los distintos campos en binario de la instrucción *slt* según el formato R. Escribe primero la instrucción a codificar.

  Instrucción:

Cod op (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shamt(5 bits)	Funció (6 bits)

4. Ayudas a la programación, pseudoinstrucciones.

Combinando instrucciones de comparación con instrucciones de salto condicional el MIPS nos proporciona una serie de pseudoinstrucciones de salto condicional con las que nos podemos apoyar para hacer los programas más legibles. El hecho de que haya pocas instrucciones básicas de salto se debe a la decisión tomada por los diseñadores del MIPS de no utilizar códigos de operación del formato tipo I innecesariamente sinó reservarlos para instrucciones más importante.
Así, por ejemplo, el MIPS dispone de la pseudoinstrucción `bgt Rs, Rt, etiqueta` la cual se traduce por la siguiente pareja de instrucciones.

```
slt $at,Rt,Rs
bne $at,$zero,Etiqueta
```

Hay que tener en cuenta que el ensamblador al hacer la sustitución de este tipo de pseudoinstrucciones utiliza el registro *$at* para guardar el resultado de la comparación.

Id con cuidado y no utilicéis el registro *$at* cuando programéis ya que el ensamblador podría destruir el valor que habéis introducido en ese registro.

En la tabla siguiente se han agrupado las instrucciones de salto condicional junto con algunas pseudoinstrucciones:

Instrucción	Ejemplo	Significado	Comentarios
Branch if equal	beq Rs, Rt, etiqueta	Si (*Rs* = *Rt*) ⇨ PC ← etiqueta	Salta a etiqueta si *Rs* es igual a *Rt*
Branch if Greater Than or Equal to Zero:	bgez Rs, etiqueta	Si (*Rs* ≥0) ⇨ PC ← etiqueta	Salta a etiqueta si *Rs* es mayor o igual que 0.
Branch if Greater Than Zero:	bgtz Rs , etiqueta	Si (*Rs* >0) ⇨ PC ← etiqueta	Salta a etiqueta si *Rs* es mayor que 0.
Branch if Less Than or Equal to Zero:	blez Rs ,etiqueta	Si (*Rs* ≤0) ⇨ PC ← etiqueta	Salta a etiqueta si *Rs* es menor o igual que 0.
Branch if Less Than Zero:	bltz Rs , etiqueta	Si (*Rs* <0) ⇨ PC ← etiqueta	Salta si *Rs* menor que 0.
Branch if Not Equal:	bne Rs, Rt , etiqueta	Si (*Rs* ≠ *Rt*) ⇨ PC ← etiqueta	Pseudoinstrucción. Salta a etiqueta si Rs no es igual a *Rt*
Branch if Greater Than or Equal	bge Rs, Rt , etiqueta	Si (*Rs* ≥ *Rt*) ⇨ PC ← etiqueta	Pseudoinstrucción. Salta a etiqueta si *Rs* es mayor o igual que *Rt*.
Branch if Greater Than:	bgt Rs, Rt , etiqueta	Si (*Rs* > *Rt*) ⇨ PC ← etiqueta	Pseudoinstrucción. Salta a etiqueta si *Rs* es mayor que *Rt* .
Branch if Less Than or Equal:	ble Rs , Rt , etiqueta	Si (*Rs* ≤ *Rt*) ⇨ PC ← etiqueta	Pseudoinstrucción. Salta a etiqueta si *Rs* es mayor o igual que *Rt* .
Branch if Less Than:	blt Rs , Rt , etiqueta	Si (*Rs* < *Rt*) ⇨ PC ← etiqueta	Pseudoinstrucción. Salta a etiqueta si *Rs* es menor que *Rt* .

En la siguiente tabla se muestran las instrucciones de comparación con algunas pseudoinstrucciones:

Instrucción	Ejemplo	Significado	Comentarios
Set on Less Than	slt Rd, Rs, Rt	*Rd*← 1 si *Rs* < *Rt* si no *Rd*← 0	Compara menor que y pone Rd a 1 si se cumple; complemento a 2
Set on Less Than Immediate:	slti Rd, Rs ,k	*Rd*← 1 si *Rs* < *k* si no *Rd*← 0	Compara menor que y pone Rd a 1 si se comple; complemento a 2
Set on Less Than unnsigned	sltu Rd, Rs, Rt	*Rd*← 1 si *Rs* < *Rt* si no *Rd*← 0	Compara menor que y pone Rd a 1 si se cumple; considera valores sin signo
Set if Equal:	seq Rd, Rs, Rt	*Rd*← 1 si *Rs* = *Rt* si no *Rd*← 0	Pseudoinstrucción Compara si igual y pone Rd a 1 si se cumple; complemento a 2
Set if Greater Than or Equal:	sge Rd, Rs, Rt	*Rd*← 1 si *Rs* ≥ *Rt* si no *Rd*← 0	Pseudoinstrucción Compara si mayor o igual, pone Rd a 1 si se cumple; complemento a 2.
Set if Greater Than:	sgt Rd, Rs, Rt	*Rd*← 1 si *Rs* > *Rt* si no *Rd*← 0	Pseudoinstrucción. Compara mayor que y pone Rd a 1 si se cumple ; complemento a 2
Set if Less Than or Equal:	sle Rd, Rs, Rt	*Rd*← 1 si *Rs* ≤ *Rt* si no *Rd*← 0	Pseudoinstrucción. Compara menor o igual y pone Rd a 1 si se cumple ; complemento a 2 .
Set if Not Equal:	sne Rd, Rs, Rt	*Rd*← 1 si *Rs* ≠ *Rt* si no *Rd*← 0	Pseudoinstrucción. Compara si no igual que y pone Rd a 1 si se cumple; complemento a 2 .

- Escribe un código ejemplo para ver como traduce el ensamblador las siguientes pseudoinstrucciones: *sgt*, *sge*, *ble*.
- Piensa en un caso en el que *slt* y *sltu* den resultados diferentes para los mismos contenidos de los registros. Compruébalo.

4. Bucles

Hay dos estructures de bucles simples en la mayoría de los lenguajes de programación, son las conocidas como *while-loop* y *for-loop*.
Un bucle while se caracteriza por tener una sentencia que controla si el bucle se ejecutará o no, mirad un ejemplo.
Suponed la siguiente sentencia en pseudocódigo:

```
while (s1 ≤ s2)
      s1 = s1 + s5;
```

queremos salir del bucle si falla la condición, es decir, si *s1>s2*, o escrito de otra forma, si *s2<s1*. Los pasos que se han de seguir para traducirlo a ensamblador serán siempre los mismos:

1. La sentencia que comprueba si se entra en el bucle se pone al inicio.
2. Habrá una etiqueta para comenzar el bucle, de esta manera la última instrucción del bucle será un salto incondicional a ella para comenzar de nuevo.
3. Habrá una segunda etiqueta para acabar el bucle.
4. Se saltará a ella cuando la condición de comprobación de **no** entrada al bucle se cumpla.

El código escrito en samblador del MIPS quedaría de la siguiente manera:

```
BucleWhile:     slt $t0, $s2, $s1
                bne $t0, $zero, SalirBucle
                add $s1, $s1, $s5
                j BucleWhile # Volver al inicio del bucle

SalirBucle:                  #Salida del bucle
```

Un bucle del tipo for-loop es aquel que se considera que va a ejecutarse un número determinado de veces. La forma normal de expresarlo es estableciendo un valor de inicio para el contador e incrementarlo en cada iteración del bucle. La condición de fin será cuando el contador llega a un valor predeterminado.
Un ejemplo es el siguiente código que suma los valores de 1 a 10:

```
n=11          # n-1, condición de finalización
total = 0;    #  Inicio del valor total de la suma
for (i = 1; i < n; i++)
{
      total = total + i
}
```

El bucle contiene 3 partes, la primera es la inicialización del contador que se tiene que hacer antes del bucle (*i=1*). Después encontramos la condición de continuar dentro del bucle (*i<n*). Y la tercera es la condición que nos dice como incrementar el contador (*i++*). Al traducir la estructura a ensamblador tenemos que incluir las tres partes.

```
                li $s0, 1           #Iniciamos contador
                li $s1, 11          #Condición de finalización
                li $s2,0            #Contador
inicio_for:     sle $t1, $s0, $s1
                beqz $t1, final_for
                add $s2, $s2, $s0   #cuerpo del bucle
                addi $s0, $s0, 1    #incremento del contador
                j inicio_for
final_for:
```

- En ejemplos como el anterior en el que sabemos que al menos el bucle se ejecuta una vez, es más eficiente poner la condición de salida del bucle al final del mismo. Reescribe el código anterior con la condición de salida al final del bucle (bucle del tipo *do-while*).
- Añade al código anterior la posibilidad de leer un valor *n* por teclado y escribe el resultado de la suma por consola. Haz que haya un bucle infinito que lea por teclado excepto en el caso de que se escriba 0, en el cual se saldrá del programa.

5. Ejercicios a entregar

- Haz el código que lea dos enteros de la consola y escriba la suma y vuelva a comenzar si el resultado es distinto de 0. Es pseudocódigo sería:

```
(bucle do-while)
seguir:      Leer el primer valor (A)
             Leer el segundo valor (B)
             Imprimir A+B
             Si (A+B) == 0 ir a acabar
             ir a seguir
acabar:
```

- Haz el código que lee de teclado dos valores positivos A y B en los que A<B. El programa tiene que escribir por consola los valores comprendidos entre ambos, incluyéndolos a ellos mismos. Es decir, si A=3 y B=6, escribe en la consola 3 4 5 6 (puedes escribir, por ejemplo, un salto de línea después de cada uno de los valores a mostrar).

Resumen

- Las instrucciones de salto nos permiten romper la ejecución secuencial de los programas.
- Las estructuras de control de alto nivel se pueden traducir de forma relativamente sencilla a ensamblador combinando las instrucciones de salto condicional e incondicional.
- Las pseudoinstruccions de salto combinan instrucciones aritméticas con instrucciones de salto condicional.