INGENIERÍATÉCNICA EN INFORMÁTICA DE GESTIÓN. CURSO 2001-2002

LABORATORIO DE ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES

Sesión 11
Implementación de sentencias condicionales

INSTRUCCIONES DE COMPARACIÓN Y SALTOS

El lenguaje ensamblador no dispone de estructuras de control de flujo de programa definidas, que permitan decidir entre dos (o varios) caminos de ejecución de instrucciones distintos (por ejemplo, la sentencia *if* de otros lenguajes de programación como PASCAL, C, etc.). Normalmente para implementar cualquier estructura de este tipo es necesario evaluar previamente una condición, simple o compuesta. El camino que seguirá la ejecución del programa dependerá del resultado de esta evaluación.

En esta práctica se realiza, en primer lugar, un breve repaso al conjunto de instrucciones y pseudoinstrucciones que tiene el MIPS R2000 para realizar comparaciones y control del flujo de programa. Y, a continuación, se describe cómo implementar a partir de ellas estructuras condicionales como *Si-entonces* y *Si-entonces-sino*, típicas de lenguajes de alto nivel.

Instrucciones de salto condicional del MIPS R2000

El ensamblador del MIPS incluye dos instrucciones básicas de toma de decisiones `beq` (del ingés "branch if equal") y `bne` (del ingés "branch if not equal"), que permiten implementar estructuras de control muy sencillas. La sintaxis de estas instrucciones es la siguiente:

```
beq rs,rt,etiqueta

bne rs,rt,etiqueta
```

Ambas comparan el contenido de los registros `rs` y `rt` y, según el resultado de esta comparación (cierta o falsa), saltan o no a la dirección de la instrucción que referencia `etiqueta`. El resultado de la evaluación es cierto si el contenido del registro `rs` es igual al del registro `rt` (instrucción `beq`), o falso en caso contrario (instrucción `bne`).

El MIPS R2000 también dispone de instrucciones de salto condicional para realizar comparaciones con cero. Estas instrucciones son: `bgez` ("branch if greater or equal to zero"), `bgtz` ("branch if greater than zero"), `blez` ("branch if less or equal to zero"), `bltz` ("branch if less than zero"), y tienen la siguiente sintaxis:

```
bgez rs,etiqueta

bgtz rs,etiqueta

blez rs,etiqueta

bltz rs,etiqueta
```

Todas ellas comparan el contenido del registro `rs` con 0 y saltan a la dirección de la instrucción referenciada por `etiqueta` si `rs`≥0 (`bgez`), `rs`>0 (`bgtz`), `rs`≤0 (`blez`) o `rs`<0 (`bltz`).

Pseudoinstrucciones de salto condicional

Para facilitar la programación el lenguaje ensamblador del MIPS R2000 aporta un conjunto de pseudoinstrucciones de salto condicional que permiten comparar dos variables almacenadas en registros (a nivel de mayor, mayor o igual, menor, menor o igual) y, según el resultado de esa comparación, saltan o no, a la instrucción que referencia una etiqueta. Estas pseudoinstrucciones son: `bge` ("branch if greater or equal", saltar si mayor o igual), `bgt` ("branch if greater than", saltar si mayor que), `ble` ("branch if less or equal", saltar si menor o igual), `blt` ("branch if less than", saltar si menor que). El formato es el mismo para todas ellas:

```
bxx rs,rt,etiqueta
```

El salto a la instrucción referenciada por `etiqueta` se efectúa si el resultado de la comparación entre las variables contenidas en los registros `rs` y `rt` es: mayot o igual (`xx=ge`), mayor estricto (`xx=gt`), menor o igual (`xx=le`) o menor estricto (`xx=lt`).

Instrucciones de salto incondicional

La instrucción `j` ("jump", saltar) permite romper la secuencia de ejecución del programa de forma incondicional y desviarla hacia la instrucción referenciada mediante una etiqueta. Esta instrucción presenta el siguiente formato:

```
j etiqueta
```

Instrucciones de comparación

El lenguaje máquina y ensamblador del MIPS dispone de una instrucción que compara dos registros y carga un 1 ó un 0 en un tercer registro dependiendo del resultado de la comparación. Si el contenido del primer registro es menor estricto que el segundo carga un 1 y, en caso contrario, carga un 0. Ésta es la instrucción `slt` ("set if less than", poner 1 si menor que) y tiene la siguiente sintaxis:

```
slt rd,rs,rt
```

La nomenclatura que se va a seguir a lo largo de este capítulo para describir el resultado que proporciona la ejecución de una instrucción de este tipo (evaluación de una condición) es el siguiente:

```
rd(1) ← (rs < rt),
```

indicando que el registro `rd` se pondrá a 1 si el contenido del registro `rs` es menor que `rt` y se pondrá a 0 en caso contrario.

Pseudoinstrucciones de comparación

Para complementar la anterior instrucción de comparación (sin salto) el ensamblador del MIPS permite utilizar un conjunto de pseudoinstrucciones que facilitan la evaluación de otras condiciones habituales. Estas pseudoinstrucciones realizan comparaciones de dos variables a otros niveles (como mayor, mayor o igual, menor o igual, igual, distinto) contenidas en sendos registros y almacenan el resultado de la comparación en un tercer registro (poniendo un 1 si la condición es cierta, y un 0 en caso contrario). Estas pseudoinstrucciones tienen la misma sintaxis que la instrucción `slt` descrita y son las siguientes: `sge` ("set if greater or equal", poner 1 si mayor o igual), `sgt` ("set if greater than", poner 1 si mayor), `sle` ("set if less or equal", poner 1 si menor o igual), `sne` ("set if not equal", poner a 1 si distinto), `seq` ("set if equal", poner a 1 si igual).

La siguiente tabla resume el conjunto de instrucciones y pseudoinstrucciones de comparación y/o salto vistas hasta el momento:

INST. SALTO CONDICIONAL	SIGNIFICADO
beq rs,rt,etiqueta	Salta a etiqueta si rs = rt
bne rs,rt,etiqueta	Salta a etiqueta si rs ≠ rt
bgez rs,etiqueta	Salta a etiqueta si rs ≥ 0
bgtz rs,etiqueta	Salta a etiqueta si rs **>** 0
blez rs,etiqueta	Salta a etiqueta si rs ≤ 0
bltz rs,etiqueta	Salta a etiqueta si rs **<** 0
PSEUDO-INST. SALTO CONDICIONAL	**SIGNIFICADO**
bge rs,rt,etiqueta	Salta a etiqueta si rs ≥ rt
bgt rs,rt,etiqueta	Salta a etiqueta si rs **>** rt
ble rs,rt,etiqueta	Salta a etiqueta si rs ≤ rt
blt rs,rt,etiqueta	Salta a etiqueta si rs **<** rt
INSTRUCCIONES SALTO INCONDICIONAL	**SIGNIFICADO**
j etiqueta	Salta a etiqueta
INSTRUCCIONES COMPARACIÓN	**SIGNIFICADO**
slt rd,rs,rt	rd(1) ← (rs **<** rt)
PSEUDOINSTRUCCIONES COMPARACIÓN	**SIGNIFICADO**
sle rd,rs,rt	rd(1) ← (rs ≤ rt)
sgt rd,rs,rt	rd(1) ← (rs **>** rt)
sge rd,rs,rt	rd(1) ← (rs ≥ rt)
seq rd,rs,rt	rd(1) ← (rs **=** rt)
sneq rd,rs,rt	rd(1) ← (rs ≠ rt)

ESTRUCTURAS DE CONTROL CONDICIONAL

Una vez introducidas en la sección anterior el conjunto de instrucciones y pseudoinstrucciones que permiten implementar cualquier estructura de control de flujo de programa, se va a describir cómo se implementan las estructuras condicionales más típicas de un lenguaje de alto nivel, como son *Si-entonces* y *Si-entonces-sino,* si se habla en lenguaje algorítmico, o las sentencias *if-then* e *if-then-else* del lenguaje Pascal. Estas estructuras dependen, implícita o explícitamente, de la verificación de una o varias condiciones para determinar el camino que seguirá la ejecución del código en curso. La evaluación de esta condición (o condiciones) vendrá asociada a una o varias instrucciones de salto condicional e incondicional o pseudoinstrucciones.

Estructura de control *Si-entonces* con condición simple

Crea un fichero con el siguiente fragmento de código que implementa una estructura de control condicional *Si-entonces*:

```
            .data
dato1:      .word  40
dato2:      .word  30
res:        .space 4
            .text
main:       lw     $t0,dato1($0)   # cargar dato1 en $t0
            lw     $t1,dato2($0)   # cargar dato2 en $t1
            and    $t2,$t2,$0      # t2=0
Si:         beq    $t1,$0,finsi    # si $t1 = 0 finsi
entonces:   div    $t0,$t1         # $t0/$t1
            mflo   $t2             # almacenar LO en $t2
finsi:      add    $t3,$t0,$t1     # $t3=$t0+$t1
            add    $t2,$t3,$t2     # $t2=$t3+$t2
            sw     $t2,res($0)     # almacenar en res $t2
```

A continuación se muestra la descripción algorítmica de este programa en ensamblador. Ésta es una transcripción casi directa del programa en ensamblador, donde se tienen que utilizar registros del procesador para almacenar temporalmente las variables en memoria ya que se trata de un procesador basado en una arquitectura de carga y almacenamiento.

```
PROGRAMA SI-ENTONCES-SIMPLE-1
VARIABLES
      ENTEROS: dato1=40; dato2=30; res;
INICIO
      $t0=dato1;
      $t1=dato2;
      $t2=0;
      Si ($t1!=0) entonces
            $t2=$t0/$t1;
      FinSi
      $t3=$t0+$t1;
      $t2=$t2+$t3;
      res=$t2;
FIN
```

Una descripción algorítmica de más alto nivel pasaría por no utilizar los registros y trabajar directamente sobre las variables almacenadas en memoria:

```
PROGRAMA SI-ENTONCES-SIMPLE-2
VARIABLES
      ENTERO: dato1=40; dato2=30; res;
INICIO
      Si (dato2!=0) entonces
            res=dato1/dato2;
      FinSi
      res=res+dato1+dato2;
FIN
```

Una vez comprendido el funcionamiento del programa anterior, borra los valores de la memoria y carga el fichero que contiene el programa en el simulador.

Cuestión 1: Identifica la instrucción que evalúa la condición y controla el flujo de programa. Compárala con la condición del programa descrito en lenguaje algorítmico.

Cuestión 2: Identifica el conjunto de instrucciones que implementan la estructura condicional *Si-entonces*. Dibuja el diagrama de flujo asociado a la estructura de control implementada en el fragmento de código anterior.

Cuestión 3: ¿Qué valor se almacena en la variable `res` después de ejecutar el programa?

Cuestión 4: Si `dato2` es igual 0 ¿Qué valor se almacena en la variable `res` después de ejecutar el programa?

Cuestión 5: Implementar el siguiente programa descrito en lenguaje algorítmico:

```
PROGRAMA SI-ENTONCES-SIMPLE-3
VARIABLES
      ENTERO: dato1=40; dato2=30; res;
INICIO
      Si (dato2>0) entonces
            res=dato1/dato2;
      FinSi
      res=res+dato1+dato2;
FIN
```

Estructura de control *Si-entonces* con condición compuesta

Crea un fichero con el siguiente fragmento de código que implementa una estructura de control condicional *Si-entonces*:

```
            .data
dato1:      .word    40
dato2:      .word    30
res:        .space   4
            .text
main:       lw       $t0,dato1($0)     # cargar dato1 en t0
            lw       $t1,dato2($0)     # cargar dato2 en $t1
            and      $t2,$t2,$0        # $t2=0
Si:         beq      $t1,$0,finsi      # si $t1=0 saltar finsi
            beq      $t0,$0,finsi      # si $t0 =0 saltar finsi
entonces:   div      $t0,$t1           # $t0/$t1
            mflo     $t2               # almacenar LO en t2
finsi:      add      $t3,$t0,$t1       # $t3=t0+$t1
            add      $t2,$t3,$t2       # $t2=t3+$t2
            sw       $t2,res($0)       # almacenar en res $t2
```

La descripción algorítmica, utilizando registros para almacenar temporalmente las variables que están en memoria de este código en ensamblador se muestra a continuación:

```
PROGRAMA SI-ENTONCES-COMPUESTA-1
VARIABLES
      ENTERO: dato1=40; dato2=30; res;
INICIO
      $t0=dato1;
      $t1=dato2;
      Si ((t0!=0) and (t1!=0)) entonces
            $t2=t0/t1;
      FinSi
      $t2=$t2+$t0+$t1;
      res=$t2;
FIN
```

Borra los valores de la memoria y carga el fichero en el simulador.

Cuestión 6: Describe en lenguaje algorítmico, sin utilizar registros para almacenar temporalmente las variables que están en memoria, el programa ensamblador anterior.

Cuestión 7: Identifica la (las) instrucción(es) que evalúa(n) la condición y controla(n) el flujo de programa y compárala(s) con la condición del programa descrito en lenguaje algorítmico.

Cuestión 8: Identifica el conjunto de instrucciones que implementan la estructura condicional *Si-entonces*. Dibuja el diagrama de flujo asociado a esta estructura de control.

Cuestión 9: ¿Qué se almacena en la variable `res` al ajecutar el programa?

Cuestión 10: Si `dato1=0`, ¿qué valor se almacena en la variable `res` después de ejecutar el programa? Si `dato2=0`, ¿qué valor se almacena en la variable `res`?

Cuestión 11: Implementa el siguiente programa descrito en lenguaje algorítmico:

```
PROGRAMA SI-ENTONCES-COMPUESTA-2
VARIABLES
      ENTERO dato1=40; dato2=30; res;
INICIO
      Si ((dato1>0) and (dato2>=0)) entonces
            res=dato1/dato2;
      FinSi
      res=res+dato1+dato2;
FIN
```

Estructura de control *Si-entonces-sino* con condición simple.

Crea un fichero con el siguiente fragmento de código que implementa una estructura de control *Si-entonces-sino*.

```
            .data
dato1:      .word   30
dato2:      .word   40
res:        .space  4
            .text
main:       lw      $t0,dato1($0)     # cargar dato1 en $t0
            lw      $t1,dato2($0)     # cargar dato2 en $t1
Si:         bge     $t0,$t1, sino     # si $t0>=$t1 ir a sino
entonces:   sw      $t0,res($0)       # almacenar $t0 en res
            j       finsi             # ir a finsi
sino:       sw      $t1,res($0)       # almacenar $t1 en res
finsi:
```

Borra los valores de la memoria, carga el fichero en el simulador y ejecútalo.

Cuestión 12: Describe en lenguaje algorítmico el equivalente a este programa en ensamblador.

Cuestión 13: ¿Qué valor se almacena en `res` después de ejecutar el programa? Si `dato1=45`, ¿qué valor se almacena en `res` después de ejecutar el programa?

Cuestión 14: Identifica en el lenguaje máquina generado por el simulador el conjunto de instrucciones que implementan la pseudoinstrucción `bge`.

Cuestión 15: Implementa en ensamblador el siguiente programa descrito en lenguaje algorítmico:

```
PROGRAMA SI-ENTONCES-SINO-SIMPLE-1
VARIABLES
      ENTERO: dato1=30; dato2=40; res;
INICIO
      Si ((dato1>=dato2)) entonces
            res=dato1-dato2;
      Sino
            res=dato2-dato1;
      FinSi
FIN
```

Estructura de control *Si-entonces-sino* con condición compuesta

Crea un fichero con el siguiente fragmento de código que implementa una estructura de control *Si-entonces-sino*.

```
            .data
dato1:      .word   30
dato2:      .word   40
dato3:      .word   -1
res:        .space  4
            .text
main:       lw      $t1,dato1($0)       # cargar dato1 en $t1
            lw      $t2,dato2($0)       # cargar dato2 en $t2
            lw      $t3,dato3($0)       # cargar dato3 en $t3
Si:         blt     $t3,$t1, entonces   # si $t3<$t1 ir entonces
            ble     $t3,$t2, sino       # si $t3<=$t2 ir a sino
entonces:   addi    $t4,$0,1            # $t4=1
            j       finsi               # ir a finsi
sino:       and     $t4,$0,$0           # $t4=0
finsi:      sw      $t4,res($0)         # almacenar res
```

Borra los valores de la memoria, carga el fichero en el simulador.

Cuestión 16: Describe en lenguaje algorítmico el equivalente a este programa en ensamblador.

Cuestión 17: ¿Qué valor se almacena en `res` después de ejecutar el programa? Si `dato1=40` y `dato2=30`, ¿qué valor se almacena en `res` después de ejecutar el programa?

Cuestión 18: Implementa en ensamblador el siguiente programa descrito en lenguaje algorítmico:

```
PROGRAMA SI-ENTONCES-SINO-COMPUESTA-1
VARIABLES
      ENTERO: dato1=30; dato2=40; dato3=-1; res;
INICIO
      Si ((dato3>=dato1) AND (dato3<=dato2)) entonces
            res=1;
      Sino
            res=0;
      FinSi
FIN
```

Problemas propuestos

1. Implementa un programa en ensamblador del R2000 que defina la cadena de caracteres "`Probando`" y que almacene en una variable entera denominada `res` el número de veces que aparece en ella un determinado carácter almacenado en la variable `car`.

2. Implementa un programa en ensamblador del R2000 que defina un vector de enteros, `V`, inicializado a los siguientes valores `V`=[1, -4, -5, 2], y obtenga como resultado una variable booleana `res` que será 1 si todos los elementos de este vector son menores que cero.

3. Implementa un programa en ensamblador del R2000 que defina un vector de enteros, `V`, inicializado a los siguientes valores `V`=[2,0,0], y almacene en una variable entera denominada `res1` el número de componentes nulas del vector y en la variable entera `res2`, el número de componentes no nulas.

4. Implementa un programa en ensamblador del R2000 que almacene en memoria los 5 enteros siguientes (`dato1`=2, `dato2`=10, `dato3`=50, `dato4`=70, `dato5`=34) y que reserve 1 palabra para almacenar el resultado (variable `res`). El programa almacenará en la variable `res` un 1 si `dato5` está en alguno de los intervalos formados por [`dato1` ,`dato2`] o [`dato3`,`dato4`]. En caso contrario se almacenará un cero.