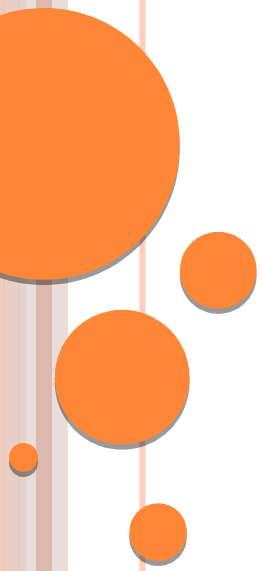

PROGRAMANDO EN MIPS

Principio de Computadores

Sesión académicamente dirigida nº 2

INTRODUCCIÓN: PROGRAMANDO EN ENSAMBLADOR

- Cuando uno desea desarrollar un algoritmo normalmente lo hace en un lenguaje de alto nivel como C o Java, así que el primer paso es hacer el pseudocódigo correspondiente.
- En esta sesión se explicarán algunos de los elementos básicos que son necesarios para poder realizar los algoritmos de nuestras prácticas. El uso de los registros y las estructuras de control.

LOS REGISTROS

- Nuestros programas de alto nivel usan variables para almacenar la información (masa, radio, etc.).
- Estas variables no son más que identificadores que hacen referencia a una posición de memoria donde se almacena el dato correspondiente.
- Ya se ha visto en clase que en ensamblador del MIPS declaramos nuestra variables bajo la directiva .data
- Sin embargo, cuando tratamos los datos en MIPS no lo hacemos directamente accediendo a memoria, ya que esto es muy ineficiente. Es necesario pues hacer uso de los registros.

USO DE LOS REGISTROS

- Si tu algoritmo usa pocas variables puedes mantener una tabla de referencia con los registros que estás usando en el mismo y a qué variable de tu algoritmo en lenguaje de alto nivel hace referencia.
- MIPS hace uso de 32 registros enteros $0-$31. Para hacer más legible por el humano, en nuestros programas de ensamblador podemos usar su nombre simbólico. $t0, $t1, … $s0, $sp, …
- Como ya se ha explicado en las clases de teoría, el programador de ensamblador es responsable de hacer un uso adecuado de los registros. Existe un convenio que ha de respetarse si queremos que nuestros programas puedan ser usados de forma correcta con otros desarrollados por otros programadores.

LOS REGISTROS

Num	Nombre	Descripción
0	$zero	Constante valor cero
1	$at	Temporal reservado para el ensamblador
2-3	$v0-$v1	Valores resultantes de funciones y evaluaciones
4-7	$a0-$a3	Argumentos para subrutinas. No se preservan a través de llamadas a subrutinas.
8-15	$t0-$t7	Temporales. No se preservan a través de llamadas de subrutinas, por lo que la rutina que llama debe salvarlos si los quiere conservar.
16-23	$s0-$s7	Valores salvados. Una subrutina que los use debe salvar sus valores antes de usarlos, y restaurarlos al salir.
24-25	$t8-$t9	Continuación a los $t0-t7. Temporales. No se preservan a través de llamadas de subrutinas, por lo que la rutina que llama debe salvarlos si los quiere conservar
26-27	$k0-$k1	Reservados para el kernel (manejo de interrupciones)
28	$gp	Puntero global. Apunta al medio del bloque de 64K en el segmento de datos estáticos
29	$sp	Puntero de pila (stack pointer)
30	$s8/$fp	Valor salvado. Puntero de marco. Se preserva a través de llamadas.
31	$ra	Dirección de retorno (return address)

REGISTROS: RESUMEN CONVENIO USO

- Aquellos que empiezan por la letra “s” significa que son “salvados” (saved) y por lo tanto su valor debe ser mantenido aunque se haga uso de una subrutina externa.
- Los registros que empiecen por “t” son “temporales”, haciendo notar que en cualquier parte del código podrán usarse. Esto quiere decir principalmente que si estamos programando una subrutina o función no es responsabilidad de la subrutina restaurar el valor de los mismos a la salida. Será en todo caso responsabilidad del módulo llamador salvar su valor en la pila en el momento de llamar a otra función para restaurarlo a su regreso.
- Los registros $a0-$a3 se usan como argumentos a las subrutinas o funciones.
- Los registros $v0-$v1 se usan para valores de retorno a las funciones.

ESTRUCTURAS DE CONTROL

- Para poder dirigir la secuencia de sentencias a ejecutar en nuestros algoritmos usamos las estructuras de control. Las más habituales son:
  - Condicional if-then-else
  - Bucles: While-do, do-while, for.

- El fundamento de estas estructuras de control, se basa en realizar saltos a zonas de programas en función del valor de evaluación de una condición.
- Las instrucciones de salto que hemos visto en teoría son:
- Saltos incondicionales:
  - `j etiqueta  # salta a la parte del código que empiece con etiqueta:`
  - `b etiqueta  # salta a la parte del código que empiece con etiqueta:`
  - `jr $t3  # salta a la dirección de programa contenida en $t3`

- Saltos condicionales:
  - `beq $t0,$t1,etiqueta # si $t0=$t1 entonces salta a etiqueta:`
  - `blt $t0,$t1,etiqueta # si $t0<$t1 entonces salta a etiqueta:`
  - `ble $t0,$t1,etiqueta # si $t0<=$t1 entonces salta a etiqueta:`
  - `bgt $t0,$t1,etiqueta # si $t0>$t1 entonces salta a etiqueta:`
  - `bge $t0,$t1,etiqueta # si $t0>=$t1 entonces salta a etiqueta:`
  - `bne $t0,$t1,etiqueta # si $t0<>$t1 entonces salta a etiqueta:`

PARA MUESTRA UN BOTÓN … IF

Cálculo del máximo de dos variables. Suposiciones:

$s0 primer número. $s1 segundo número. $s2 máximo($s0,$s1)

Nuestro algoritmo sería:

if ($s0 < $s1) {$s2 = $s1} else {$s2 = $s0}

En ensamblador:

```
# Ejemplo IF. Calculo del máximo
     .data
num1:     .word     7
num2:     .word     12
max:      .word     0

     .text
main:
     lw   $s0,num1
     lw   $s1,num2
if:  bge  $s0,$s1,el
     move $s2,$s1
     j    fi
el:  move $s2,$s0
fi:  sw   $s2,max
     li   $v0,10
     syscall
```

PARA MUESTRA UN BOTÓN ….
WHILE

El siguiente algoritmo acumula números enteros introducidos por teclado hasta que introduzcamos uno que tiene valor negativo.

Nuestro algoritmo:

```
cin >> numero;
suma = 0;
while (numero >=0) {
   suma += numero;
   cin >> numero;
}
cout << suma;
```

```
# ejemplo While. Acumula enteros hasta que
# metamos uno negativo
        .data
suma:   .word 0
        .text
main:
        # $s0 lo usamos para acumular
        li   $s0,0
        # leo un entero por consola
        li   $v0,5
        syscall # $v0 contiene el entero leido

while:  blt  $v0,0,endw
        add  $s0,$s0,$v0
        # leo siguiente entero por consola
        li   $v0,5
        syscall # $v0 contiene el entero leido
        b    while
endw:   sw $s0,suma
        li $v0,10
        syscall
```

PARA MUESTRA UN BOTÓN … FOR

Ahora una de las estructuras de bucle más comunes. Cuando sabemos con exactitud el número de iteraciones. Ejemplo: la suma de los 100 primeros números naturales.

```
sum a = 0;
for(i= 0;i<= 100;i++)
   {sum a += i;}
```

```
# ejemplo FOR. Suma los 100 primeros
# numeros naturales
          .data
suma:     .word 0
          .text
main:
          # $s0 lo usamos para acumular como suma
          li   $s0,0
          # $s1 sera la variable indice del bucle for
          li   $s1,0


for:      bgt  $s1,100,endf
          add  $s0,$s0,$s1
          addi $s1,$s1,1
          b    for
endf:     sw $s0,suma
          li $v0,10
          syscall
```

PARA QUE TRATES DE HACERLO TÚ …

El condicional múltiple … switch y el bucle do-while

```
switch (expresión)
{
case cte_1: sentencia_11;
sentencia_12;
...
case cte_2: sentencia_21;
sentencia_22;
...
...
...
...
case cte_n: sentencia_n;
sentencia_n;
...
}
```

```
do {

        Sentencia1;

        Sentencia2;

        Sentencia3;

        …

    } while (condición);
```

CONDICIONES MÁS COMPLEJAS

- No siempre las expresiones lógicas de los condicionales y los bucles son tan fáciles como comparar dos números. Si la expresión es más compleja las siguientes instrucciones te pueden ayudar:

```
seq $t1,$t2,$t3 Set EQual : if $t2 equal to $t3 then set $t1 to 1 else 0
sge $t1,$t2,$t3 Set Greater or Equal : if $t2 greater or equal to $t3 then set $t1 to 1 else 0
sgt $t1,$t2,$t3 Set Greater Than : if $t2 greater than $t3 then set $t1 to 1 else 0
sle $t1,$t2,$t3 Set Less or Equal : if $t2 less or equal to $t3 then set $t1 to 1 else 0
sne $t1,$t2,$t3 Set Not Equal : if $t2 not equal to $t3 then set $t1 to 1 else 0
```

```
and $t1,$t2,$t3 Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3
not $t1,$t2 Bitwise NOT (bit inversion)
or $t1,$t2,$t3 Bitwise OR : Set $t1 to bitwise OR of $t2 and $t3
nor $t1,$t2,$t3 Bitwise NOR : Set $t1 to bitwise NOR of $t2 and $t3
xor $t1,$t2,$t3 Bitwise XOR (exclusive OR) : Set $t1 to bitwise XOR of $t2 and $t3
```