

Estructura y Tecnología de Computadores (ITIG)

12. Programación en ensamblador MIPS.

Ángel Serrano Sánchez de León
Luis Rincón Córcoles

Tema 12. Programación en ensamblador MIPS.

Programa

1. Introducción.
2. Registros.
3. Operaciones aritméticas básicas.
4. Accesos a memoria. Carga y almacenamiento.
5. Lenguaje máquina MIPS.
6. MIPS y las sentencias de control.
7. Llamadas a funciones.
8. Manejo de caracteres.
9. Aritmética para números con signo.
10. Operaciones lógicas.
11. Aritmética en coma flotante.
12. Bibliografía.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Conceptos básicos: registros de MIPS y convenio software de uso, operaciones básicas (aritméticas, lógicas, desplazamiento), operaciones de carga/almacenamiento, inmediato, formato de instrucciones en MIPS, sentencias de control (IF, FOR, WHILE, DO-WHILE, GOTO), subprograma o subrutina (función/procedimiento; hoja/anidada/recursiva), manejo de la pila, cadenas de caracteres, operaciones en coma flotante.

Instrucciones y pseudoinstrucciones de MIPS en este tema: add, add.d, add.s, addi, addiu, addu, and, andi, beqz, beq, bge, bgt, blt, bne, div, div.d, div.s, divu, j, jal, jr, lb, lbu, lh, lhu, li, li.d, li.s, lui, lw, lwcl, mfhi, mflo, move, mul, mul.d, mul.s, mult, multu, nor, or, ori, sb, sh, slt, slti, sltu, sub, sub.d, sub.s, subu, sw, swcl, xor, xori.

1. Introducción

- El lenguaje ensamblador depende directamente de la arquitectura del computador. Por tanto cada arquitectura tiene su propio lenguaje ensamblador.
- Los lenguajes ensambladores de dos arquitecturas diferentes son como primos cercanos, diferentes pero muy parecidos. Sin embargo, los lenguajes de alto nivel tienden a ser parientes lejanos, pues la sintaxis es más variable.
- En este tema veremos sólo los conceptos fundamentales de la programación en ensamblador para los procesadores MIPS.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

¡Hola Mundo!

```

.rdata                # Inicio de seccion de datos ROM
.align 2              # Direccion alineada a palabra (multiplo de 4)

texto: .ascii "Hola Mundo!" # Cadena identificada por etiqueta "texto"
.text                # Comienzo de seccion de codigo de usuario
.globl main           # La etiqueta "main" se hace conocida a
                     # nivel global
.ent main             # La etiqueta "main" marca un punto de
                     # entrada
main: la $4, texto    # Escribimos en el registro $4 la direccion
                     # de memoria asociada a la etiqueta "texto"
jal printf            # Llamada a funcion printf, que escribe
                     # cadenas de texto en la consola
j _exit               # Saltamos a la rutina de salida para
                     # terminar
.end main             # Final de la seccion "main"

```

5

2. Registros

- Recordemos que MIPS dispone, entre otros, de los siguientes registros:
 - 32 registros en la CPU, cada uno de 32 bits.
 - 32 registros en la unidad de coma flotante, cada uno de 32 bits.
 - Un contador de programa (PC) de 32 bits, que indica, al principio de cada ciclo, la dirección de memoria de la instrucción del programa que se va a ejecutar.
 - Dos registros de 32 bits para multiplicaciones y divisiones (HI y LO).
- Por convenio entre los programadores de MIPS, los registros de la CPU se usan utilizando unas "normas de buen uso":

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Registro	Número	Uso	¿Preservado?
\$zero	0	Constante con valor 0	No aplicable
\$at	1	Temporal para el ensamblador	No
\$v0 - \$v1	2 - 3	Valores devueltos en funciones	No
\$a0 - \$a3	4 - 7	Argumentos en funciones	No
\$t0 - \$t7	8 - 15	Temporales	No
\$s0 - \$s7	16 - 23	Temporales salvados	Sí
\$t8 - \$t9	24 - 25	Temporales	No
\$k0 - \$k1	26 - 27	Reservados para kernel	No
\$gp	28	Puntero global	Sí
\$sp	29	Puntero de pila	Sí
\$fp	30	Puntero de marco	Sí
\$ra	31	Dirección de retorno	Sí

7

3. Operaciones aritméticas básicas

- MIPS es una máquina de arquitectura carga-almacenamiento: para usar un dato almacenado en memoria, primero hay que pasarlo a un registro.
- Las operaciones aritméticas básicas en MIPS se caracterizan por:
 - Utilizar tres registros (2 para los operandos y 1 para el resultado).
 - Sintaxis: `operacion resultado op1 op2`
 - El último de los operandos puede ser una constante de 16 bits ("inmediato").
- Ejemplo: El mnemónico para la suma es `add`.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

- Si sólo podemos sumar dos registros en cada instrucción, ¿cómo hacemos operaciones más complicadas?
- Ejemplo: El mnemotécnico para la resta es `sub`.

⇒ C:

```
int a, b, c, d, e;
a = ( b + c ) - ( d + e );
```

⇒ MIPS:

```
# Suponiendo que los datos a, b, c, d, e
# están asignados a los registros
# $s0, $s1, $s2, $s3 y $s4, respectivamente:
add $t0, $s1, $s2 # $t0 = $s1 + $s2
add $t1, $s3, $s4 # $t1 = $s3 + $s4
sub $s0, $t0, $t1 # $s0 = $t0 - $t1
```

- Ejercicio: Modificar el programa MIPS anterior para evitar el uso de los registros temporales `$t0` y `$t1`.

9

Carga de inmediatos en registros

- Recordando que el registro `$zero` siempre vale 0, podemos utilizarlo para asignar valores a los otros registros mediante la instrucción `addi` ("sumar inmediato"). Ejemplo:

```
addi $s0, $zero, 100 # $s0 = $zero + 100 = 0 + 100 = 100
```

- Esta instrucción sólo nos permite utilizar inmediatos de 16 bits. Al igual que en otros casos que veremos más adelante, se produce **extensión de signo** del bit 15 del inmediato (bit de signo) del 31 al 16.
- Para utilizar los 32 bits de los registros, debemos usar la pseudoinstrucción `li`:

```
li $s0, 0xABCDEF01 # $s0 = ABCDEF0116
```
- Esta pseudoinstrucción equivale a la siguiente secuencia de instrucciones:

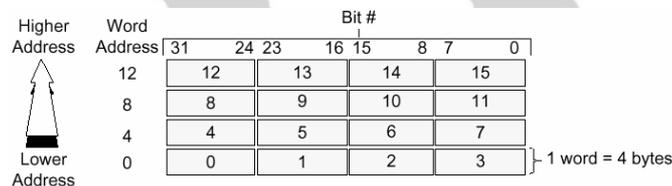
**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

4. Accesos a memoria

- El ancho del bus de datos (“palabra”) es 32 bits (4 bytes).
- De igual forma, el ancho del bus de direcciones es 32 bits. Por tanto, en la memoria hay capacidad para 2^{32} posiciones, a cada una de las cuales le corresponde 1 byte.
 - **Memoria total:** 2^{32} bytes = 2^{30} palabras = 4 Gigabytes
 - **“Big-Endian”:** dirección de palabra = dirección de byte más significativo.
 - **Restricción de alineamiento:** Las direcciones de palabra son obligatoriamente múltiplos de 4.



11

Carga y almacenamiento

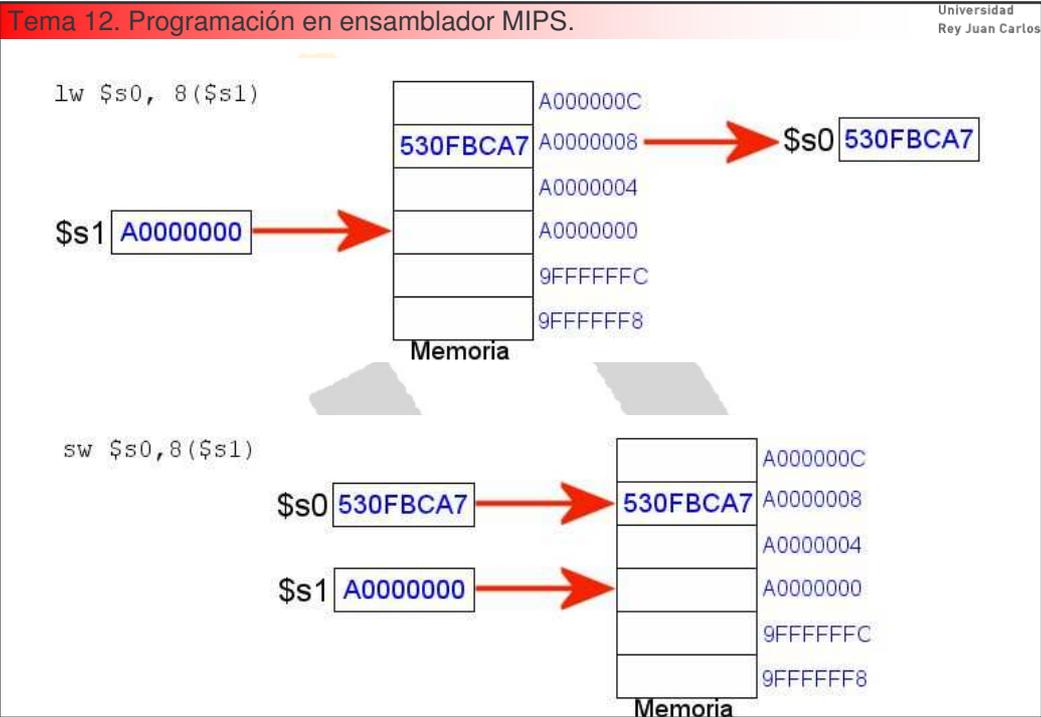
- Transferencia de datos entre memoria y registros:
 - **Carga (“load”):** Memoria → Registro.
 - **Almacenamiento (“store”):** Registro → Memoria.
- Los mnemotécnicos `lw` (load word) y `sw` (store word) permiten realizar transferencias entre memoria y registros de palabras enteras, utilizando la dirección de memoria almacenada en un registro base. Sintaxis:


```
lw registro_destino, desplazamiento(registro_origen)
sw registro_origen, desplazamiento(registro_destino)
```
- La posición de memoria exacta se indica mediante un desplazamiento en

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99



Tema 12. Programación en ensamblador MIPS. Universidad Rey Juan Carlos

Ejemplo

⇒ C:

```
int h, V[40];
V[12] = h + V[8];
```

⇒ MIPS:

```
# Suponiendo que h está asignado al registro $s0
# y la dirección del primer elemento del vector V,
# llamado V[0], está en $s1:
lw $t0, 32($s1)      # $t0 = V[8]
                    # La palabra 8 empieza en el byte 8*4=32
add $t0, $s0, $t0    # $t0 = $s0 + $t0
sw $t0, 48($s1)      # V[12] = $t0
                    # La palabra 12 empieza en el byte 12*4=48
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

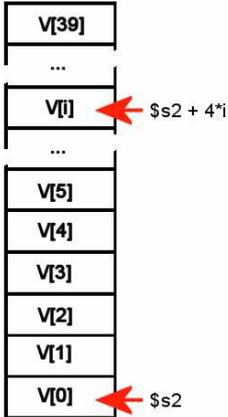
ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo

⇒ C:

```
int a, b, V[40], i;
a = b + V[i];
```



⇒ MIPS:

```
# Suponiendo que a, b, V[0], i están asignado a
# los registros $s0, $s1, $s2, $s3, respectivamente
# La palabra i hay que expresarla en bytes (x 4)
add $t0, $s3, $s3      # $t0 = i+i = 2*i
add $t0, $t0, $t0     # $t0 = $t0+$t0 = 4*i
add $t0, $s2, $t0     # $t0 = dirección de V[i]
lw $t1, 0($t0)       # $t1 = V[i]
add $s0, $s1, $t1    # $s0 = $s1 + $t1
```

5. Lenguaje máquina MIPS

- Recordemos los tres **formatos de instrucciones máquina** en MIPS:

Tipo R	Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino	Funct	
(shamt: <i>shift amount</i> en instrucciones de desplazamiento)	xxxxxx	rs	rt	rd	shamt	funct
	6	5	5	5	5	6
	31-26	25-21	20-16	15-11	10-6	5-0

Tipo I (carga o almacenamiento, ramificación condicional)	Cód. Op.	Registro base	Registro destino	Desplazamiento
	xxxxxx	rs	rt	Inmediato
	6	5	5	16
	31-26	25-21	20-16	15-0

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Tipo R
(shamt: *shift amount* en instrucciones de desplazamiento)

Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino	shamt	funct
xxxxxx	rs	rt	rd	shamt	funct
6	5	5	5	5	6
31-26	25-21	20-16	15-11	10-6	5-0

add \$t0, \$s1, \$s2

Operación → add \$t0, \$s1, \$s2
 Registro destino → \$t0
 Registro operando 1 → \$s1
 Registro operando 2 → \$s2

Código operación	Operando 1	Operando 2	Destino	Shamt	Función
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

Notación compacta hexadecimal: 02324020

Tipo I (carga o almacenamiento, ramificación condicional)

Cód. Op.	Registro base	Registro destino	Desplazamiento
xxxxxx	rs	rt	Inmediato
6	5	5	16
31-26	25-21	20-16	15-0

lw \$t0, 32(\$s2)

Operación → lw \$t0, 32(\$s2)
 Registro destino → \$t0
 Registro base → \$s2
 Desplazamiento → 32

Código	Base	Destino	Desplazamiento
--------	------	---------	----------------

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**



6. MIPS y las sentencias de control

- `beq r1, r2, etiqueta` (“branch if equal”)
 - Compara los valores contenidos en ambos registros.
 - Si son iguales, el flujo de programa salta a la instrucción que corresponde a la etiqueta.
 - Si no lo son, la instrucción que se ejecuta es la siguiente a ésta.
- `bne r1, r2, etiqueta` (“branch if not equal”)
 - En este caso, si los valores de ambos registros no son iguales, el programa salta a la instrucción que corresponde a la etiqueta.
 - Si son iguales, la instrucción que se ejecuta es la siguiente a ésta.
- `slt r1, r2, r3` (“set on less than”)
 - El registro r1 se cargará con el valor 0x00000001 si el valor contenido en r2 es menor que el de r3.
 - En caso contrario, r1 valdrá 0x00000000.

Ejemplo

⇒ C:

```
if (i==j) goto L1;
f = g + h;
L1: f = f - i;
```

f	\$s0
g	\$s1
h	\$s2
i	\$s3
j	\$s4

⇒ MIPS:

```
# Suponiendo que a las 5 variables de f a j les
# corresponden los registros de $s0 a $s4
beq $s3, $s4, L1 # if (i == j) goto L1
add $s0, $s1, $s2 # f = g + h
L1: sub $s0, $s0, $s3 # L1: f = f - i
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo

⇒ C:

```
if (i<j) goto L1;
f = g + h;
L1: f = f - i;
```

f	\$s0
g	\$s1
h	\$s2
i	\$s3
j	\$s4

⇒ MIPS:

```
# Suponiendo que a las 5 variables de f a j les
# corresponden los registros de $s0 a $s4
addi $s5, $zero, 1 # $s5 = 0 + 1 = 1
slt $t0, $s3, $s4 # if (i<j) then $t0=1
# else $t0=0
beq $t0, $s5, L1 # if ($t0 == 1) goto L1
add $s0, $s1, $s2 # f = g + h
L1: sub $s0, $s0, $s3 # L1: f = f - i
```

- Con sólo las instrucciones `bne`, `beq` y `slt` y con el registro `$zero` (que siempre vale 0) se pueden construir todas las condiciones de comparación (\leq , $<$, $=$, $>$, \geq).
- El ensamblador suele incorporar pseudoinstrucciones como `bgt`, `blt`, `begz`, etc.

21

⇒ Analizaremos las sentencias de control sencillas del lenguaje PASCAL trasladadas al lenguaje ensamblador de MIPS.

⇒ Se analizarán las siguientes sentencias de control:

- Bifurcación incondicional GOTO.
- Condiciones IF - THEN.
- Condiciones IF - THEN - ELSE.
- Bucle REPEAT - UNTIL.
- Bucle WHILE - DO.

• Bucle FOR - TO

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

6.1. Saltos incondicionales (“GOTO”)

Tipo J
(salto incondicional)



Cód. Op.	Dirección destino
xxxxxx	dirección
6	26
31-26	25-0

⇒ MIPS: “fin” es la etiqueta que identifica la instrucción almacenada en la palabra de memoria 2500 (hay que multiplicar por 4 para obtener su dirección en bytes=10000)

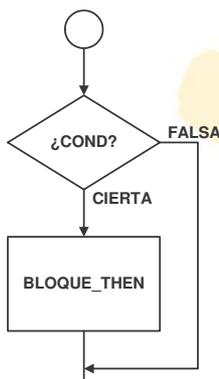
```

j fin
...
fin:
...
    
```

Notación compacta hexadecimal: 080009C4

Código operación	Destino
2	2500
000010	00000000000000100111000100

6.2. Condiciones (“IF – THEN”)



⇒ PASCAL:

```

IF x >= y
THEN
BEGIN
  x := x+2;
  y := y-2;
END;
    
```

⇒ MIPS:

```

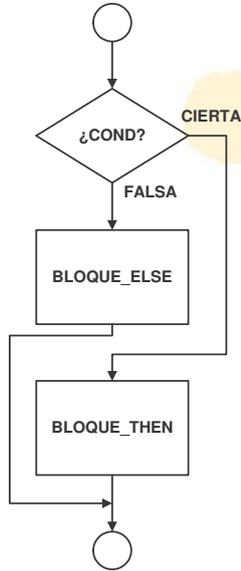
# $s0 = x
# $s1 = y
IF:
  blt  $s0,$s1,END
  addi $s0,$s0,2
  addi $s1,$s1,-2
END:
    
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Condiciones (“IF - THEN – ELSE”)



⇒ PASCAL:

```
IF x >= y
  THEN
    BEGIN
      x := x+2;
      y := y+2;
    END;
  ELSE
    BEGIN
      x := x-2;
      y := y-2;
    END;
END;
```

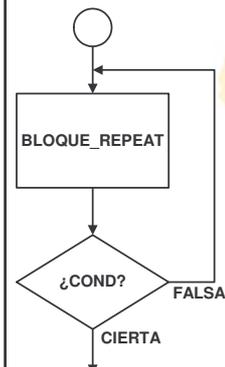
⇒ MIPS:

```
# $s0 = x
# $s1 = y
IF:
  bge    $s0, $s1, THEN
ELSE:   addi   $s0, $s0, -2
        addi   $s1, $s1, -2
        j      END
THEN:   addi   $s0, $s0, 2
        addi   $s1, $s1, 2
END:
```

bge = branch if greater or equal than (pseudoinstrucción que comprueba la condición $x \geq y$)

25

6.3 . Bucles (“REPEAT – UNTIL”)



⇒ PASCAL:

```
a := 81;
b := 18;
REPEAT
  mcd := b;
  resto := a MOD b;
  a := b;
  b := resto;
UNTIL resto = 0;
```

⇒ MIPS:

```
# $s0 = a
# $s1 = b
# $s2 = mcd
# $s3 = resto

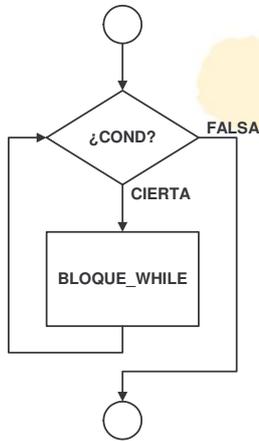
li    $s0, 81
li    $s1, 18
REPEAT:
  move $s2, $s1
  div  $s0, $s1
  mfhi $s3
  move $s0, $s1
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Bucles ("WHILE - DO")



⇒ PASCAL:

```
n := 5; fant := 1;
f := 1; i := 2;
WHILE i <= n DO
  BEGIN
    faux := f;
    f := f + fant;
    fant := faux;
    i := i+1;
  END;
```

f	f(n-1)
fant	f(n-2)
faux	Valor auxiliar

⇒ MIPS:

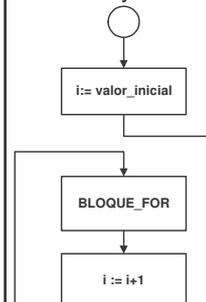
```
# $s0 = n
# $s1 = f
# $s2 = fant
# $s3 = i
# $s4 = faux
li $s0, 5
li $s2, 1
li $s1, 1
li $s3, 2
WHILE: bgt $s3, $s0, END
      move $s4, $s1
      add $s1, $s1, $s2
      move $s2, $s4
      addi $s3, $s3, 1
      j WHILE
END:
```

Algoritmo de cálculo del término n-ésimo de la serie de Fibonacci:
 $f(n) = f(n-1) + f(n-2)$, $n > 1$ entero, donde $f(0) = 1$ y $f(1) = 1$

27

Bucles ("FOR - TO")

- ⇒ En PASCAL estándar, el valor final del contador es indefinido, y además **el contador no debería ser modificado dentro del bucle** (¡puede haber resultados inesperados!).
- ⇒ El límite final no cambia a lo largo de la ejecución del bucle (aun cuando sea una variable y ésta se modifique en el bucle).



⇒ PASCAL:

```
n := 5; fant := 1; f := 1;
FOR i := 2 TO n DO
  BEGIN
    faux := f;
    f := f + fant;
    fant := faux;
  END;
```

⇒ MIPS:

```
# $s0 = n
# $s1 = f
# $s2 = fant
# $s3 = i
# $s4 = faux
li $s0, 5
li $s2, 1
li $s1, 1
FOR: move $t0, $s0
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo

⇒ C:

```
int a, V[100];
int i, j, k;
Bucle: a = a + V[i];
i = i + j;
if( i != k)
    goto Bucle;
```

⇒ MIPS:

```
# Suponiendo que a corresponde a $s0, V[0] a $s1,
# y las variables i, j, k a $s2 - $s4:
Bucle: add $t1, $s2, $s2      # t1 = 2*i
      add $t1, $t1, $t1      # t1 = 4*i
      add $t1, $t1, $s1      # t1 = dir. de V[i]
      lw $t0, 0($t1)         # t0 = V[i]
      add $s0, $s0, $t0      # a = a + V[i]
      add $s2, $s2, $s3      # i = i + j
      bne $s2, $s4, Bucle   # si i!=k, salta
```

a	\$s0
V[0]	\$s1
i	\$s2
j	\$s3
k	\$s4

29

Ejemplo

⇒ C:

```
int a, V[100];
int i, j, k;
while (V[i] == k)
{
    i = i + j;
}
```

⇒ MIPS:

```
# Suponiendo que a corresponde a $s0, V[0] a $s1,
# y las variables i, j, k a $s2 - $s4:
Bucle: add $t1, $s2, $s2      # t1 = 2*i
      add $t1, $t1, $t1      # t1 = 4*i
      add $t1, $t1, $s1      # t1 = dir. de V[i]
      lw $t0, 0($t1)         # t0 = V[i]
      bne $t0, $s4, Fin      # si i!=k salta
      add $s2, $s2, $s3      # i = i + j
      j Bucle
```

Fin:

a	\$s0
---	------

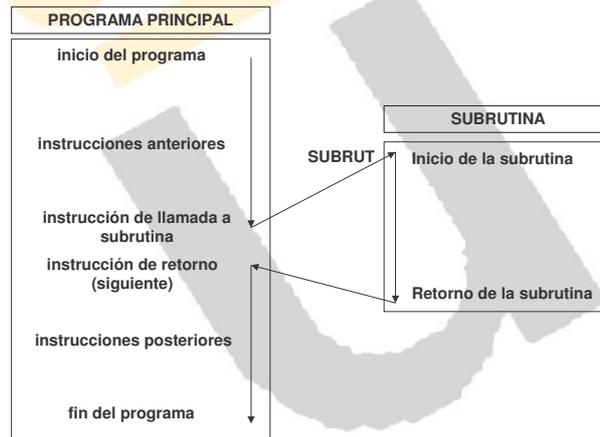
**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

7. Llamadas a subprogramas

⇒ Las subrutinas o subprogramas son fragmentos de código diseñados para realizar determinadas tareas y que pueden ser invocados desde diferentes puntos del programa principal o desde otras subrutinas.



31

- ⇒ Las subrutinas normalmente reciben un conjunto de parámetros (variables con cuyos valores realizan los cálculos u operaciones adecuadas).
 - Si es resultado de las operaciones es un valor, éste se devuelve al finalizar la subrutina → **Función**.
 - Si solamente se ejecuta una secuencia de instrucciones sin devolver ningún valor → **Procedimiento**.
- ⇒ En todo momento, las subrutinas tienen un punto de entrada y un punto de retorno a la función llamante y se ejecutan de manera independiente al programa principal.
- ⇒ Etapas:
 1. Situar los parámetros de la subrutina en los registros \$a0, \$a1, \$a2, \$a3 o en la pila.
 2. Llamar a la subrutina → jal etiqueta de subrutina (→ \$ra=PC+4)

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Convenio software para uso de registros

- ⇒ Si una subrutina usa los registros a su antojo, el invocador no puede confiar en sus contenidos.
 - Para evitar problemas, a veces es preciso salvaguardar ciertos registros en pila.
- ⇒ El invocado **SÍ** tiene la obligación de devolver intactos al invocador los siguientes registros (salvaguardándolos en pila si es preciso):
 - Los registros `$s0` ... `$s7`, ya que contienen variables de larga duración.
 - `$sp`: puntero de pila.
 - `$fp`: puntero a bloque de activación.
 - `$ra`: dirección de retorno.
 - `$gp`: puntero a zona de datos globales (la subrutina no debería modificarlo).
- ⇒ El invocado **NO** tiene la obligación de devolver intactos al invocador los siguientes registros (si el invocador quiere confiar en sus contenidos, debe salvaguardarlos en pila él mismo):
 - Los registros `$t0` ... `$t9`, ya que contienen datos de vida corta.
 - Los registros `$a0` ... `$a3`, que contienen los argumentos de entrada.
- ⇒ El invocado modifica los registros `$v0` y `$v1` poniendo su valor de retorno, por lo que evidentemente el invocador no los salvaguarda en pila.

33

Ejemplo: Función Hoja

⇒ C:

```
int funHoja(int g, int h, # Suponiendo que a los parámetros g - j les
int i, int j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

⇒ MIPS:

```
funHoja: add $t0, $a0, $a1 # $t0 = g + h
         add $t1, $a2, $a3 # $t1 = i + j
         sub $v0, $t0, $t1 # $v0 = (g+h) - (i+j)
         jr $ra           # vuelta a función
         # llamante
```

g	\$a0
h	\$a1

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo: Procedimiento Hoja

⇒ C:

```
swap (int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

⇒ MIPS:

```
# v[0] se recibe en el registro $a0 y k en $a1:
swap:  add $t1, $a1, $a1      # $t1 = 2*k
        add $t1, $t1, $t1    # $t1 = 4*k
        add $t1, $a0, $t1    # $t1 = dir. v[k]
        lw $t0, 0($t1)      # $t0 = v[k]
        lw $t2, 4($t1)      # $t2 = v[k+1]
        sw $t2, 0($t1)      # v[k] = v[k+1]
        sw $t0, 4($t1)      # v[k+1] = $t0
        jr $ra              # vuelta a función
        # llamante
```



• Una **procedimiento hoja** es aquel que no llama a otro procedimiento.

• Esta subrutina swap intercambia dos posiciones de memoria, la k y la k+1, dentro del vector de enteros v.

35

Subrutinas anidadas: tratamiento de la pila

- Una subrutina anidada es aquella que llama a otras.
- Para el correcto funcionamiento del programa, es necesario hacer uso de la pila, cuya dirección de inserción está marcada por el puntero de pila $\$sp$.
- Se trata de una sección de la memoria que crece hacia direcciones decrecientes, donde el programador puede insertar o extraer datos según sus necesidades.
- Es habitual en estos casos introducir en pila los registros siguientes: $\$ra$, argumentos $\$a0 - \$a3$, así como los registros $\$s$ que se quieran modificar en la subrutina.

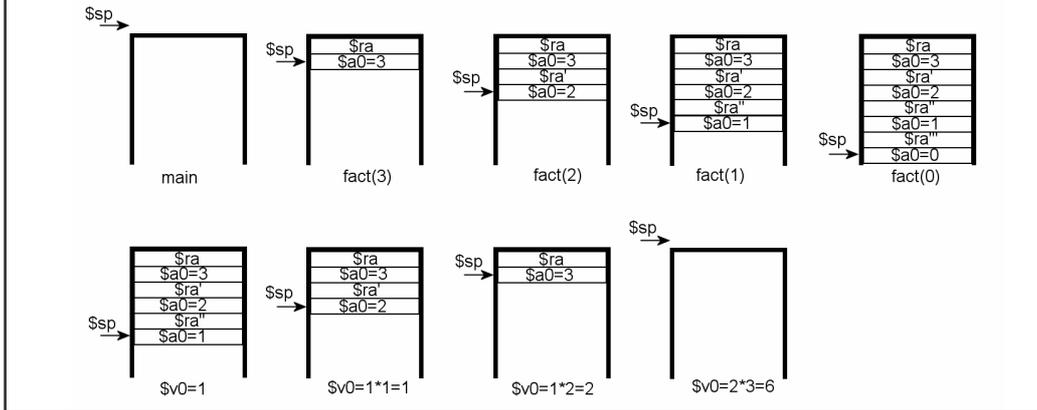
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Subrutinas anidadas: recursividad

- Un caso particular de este tipo de subrutinas son las **subrutinas recursivas**, que son las que se llaman a sí mismas.
- Ejemplo: la función factorial $n! = n(n-1)!$ Veamos el manejo de la pila en el caso del cálculo de $3!$



Ejemplo: Función Recursiva

⇒ C:

```
int fact(int n)
{
    if (n < 1)
    {
        return 1;
    }
    else
    {
        return n * fact(n-1);
    }
}
```

⇒ MIPS:

```
fact:  slti $t0, $a0, 1      # n<1 ?
      beq $t0, $zero, else  # Si n>=1, ir a else
      addi $v0, $zero, 1    # devuelve 1
      jr $ra
else:  addi $sp, $sp, -8     # Reserva 2 palabras
      # en pila (8 bytes)
      sw $ra, 4($sp)        # guarda ra y
      sw $a0, 0($sp)        # n en pila
      addi $a0, $a0, -1     # a0=n-1
      jal fact              # factorial de n-1
      lw $a0, 0($sp)        # restaura n
      lw $ra, 4($sp)        # y ra
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

8. Manejo de caracteres

- Los caracteres en MIPS son manejados por su código ASCII (1 byte).
- Las cadenas de caracteres son secuencias de caracteres almacenados consecutivamente en memoria (byte tras byte), terminadas por el carácter nulo.
- Equivalentemente a las instrucciones `lw` y `sw` para manejar palabras (4 bytes), existen:
 - `lb` (load byte):
 - Sintaxis: `lb registro_destino, desplazamiento(registro_origen)`
 - Lee un byte de la memoria y los almacena en los 8 bits menos significativos de un registro (con extensión de signo a 32 bits).
 - `sb` (store byte).
 - Sintaxis: `sb registro_origen, desplazamiento(registro_destino)`
 - Toma los 8 bits menos significativos de un registro y los almacena en memoria.

39

Ejemplo de cadena de caracteres

⇒ Ejemplo: tira de 12 caracteres llamada T5, con el contenido "Hola".

Dirección o etiqueta	Contenido de la memoria	
0	...	
T5	'H'	Comienzo de la tira
T5+1	'o'	
T5+2	'l'	
T5+3	'a'	
T5+4	0	Carácter nulo, fin de la parte legible
T5+5	¿?	Carácter desconocido
T5+6	¿?	Carácter desconocido
T5+7	¿?	Carácter desconocido

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

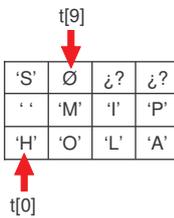
Cartagena99

Ejemplo

➤ PASCAL:

```
CONST N = 10;
VAR t: ARRAY [0..N-1] OF
  CHAR;
    lon: INTEGER;

BEGIN
lon := 0;
WHILE t[lon] <> CHR(0) DO
  lon := lon+1;
END
```



➤ MIPS:

```
# Suponemos que lon corresponde al registro $s0
n = 10 # Definición de Símbolo
t: .space n # Reservamos n bytes para t
    add $s0,$zero,$zero # $s0 = lon = 0

while:
    la $t0,t # $t0 = dir. de t[0]
    add $t0,$t0,$s0 # $t0 = dir. de t[lon]
    lb $t1,0($t0) # $t1 = byte en t[lon]
    beq $t1,$zero,fin # if ($t1==0) goto fin
    addi $s0,$s0,1 # $s0 = $s0 + 1
    j while # goto while

fin:
```

- Cálculo de la longitud de una cadena (compara byte a byte hasta llegar al carácter nulo, de código ASCII 0).

41

9. Aritmética para números con signo

- MIPS permite trabajar con datos en binario puro (datos sin signo) o en el sistema de representación de complemento a 2 (datos con signo).
- Por defecto las operaciones trabajan con datos en C2. En estos casos, cuando el dato es más pequeño que una palabra (32 bits) se produce una **extensión** de signo en 32 bits (tema 4).
- Cada instrucción de carga tiene otra equivalente que acaba en “u” (unsigned) para datos sin signo. Estas instrucciones no extienden el signo.

lb	lbu	Byte
lh	lhu	Media palabra

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Multiplicaciones y divisiones

- Instrucciones para multiplicar datos en MIPS (ninguna de las dos detecta desbordamientos): `mult` (con signo) y `multu` (sin signo).
 - El resultado de la multiplicación tendrá 64 bits, accesibles en los registros HI (32 bits más significativos) y LO (32 bits menos significativos).
- Instrucciones para dividir: `div` (con signo) y `divu` (sin signo). Ninguna de las dos detecta la división por cero.
 - LO almacena el cociente de la división y HI el resto.
- Transferencia del resultado a un registro:
 - `mfhi $reg` → transfiere los 32 bits de HI al registro indicado.
 - `mflo $reg` → transfiere los 32 bits de LO al registro indicado.

Ejemplo

⇒ C:

```
int func (int a0, int a1,
          int a2, int a3)
{
    return (a0+a1)*a2/a3;
}
```

⇒ MIPS:

```
func:  add $t0, $a0, $a1    # t0 = a0+a1
       mult $t0, $a2      # HI_LO=(a0+a1)*a2
       mflo $t0           # t0 = LO
       div $t0, $a3       # HI_LO=(a0+a1)*a2/a3
       mflo $v0           # v0 = LO (cociente)
       jr $ra             # fin de función
```

- En este programa tan simplificado realizamos la operación aritmética indicada.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

10. Operaciones lógicas (bit a bit)

- and destino, operando1, operando2
- andi destino, operando1, inmediato
- or destino, operando, operando2
- ori destino, operando, inmediato
- xor destino, operando1, operando2
- xori destino, operando, inmediato
- nor destino, operando1, operando2 → Se usa para la operación not (not dato = dato nor 0)

45

11. Aritmética en coma flotante

Operación	Prec. Simple (32 bits)	Prec. Doble (64 bits)
Suma	add.s	add.d
Resta	sub.s	sub.d
Multiplicación	mul.s	mul.d
División	div.s	div.d

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Ejemplo

⇒ C:

```
float f2c (float fahr)
{
    return
    ((5./9.)*(fahr-32.));
}
```

fahr	\$f12
resultado	\$f0

32
9
5

← \$sp

⇒ MIPS:

```
# Suponemos que el argumento fahr se pasa en el
# registro $f12 y que el resultado se devuelve
# en $f0 (que no vale 0 en la FPU)
# Suponemos que la función llamante ha insertado
# en la pila ($sp) las constantes 5, 9 y 32
f2c:  lwc1 $f16, 0($sp)      # $f16 = 5
      lwc1 $f18, 4($sp)      # $f18 = 9
      lwc1 $f20, 8($sp)      # $f20 = 32
      div.s $f16, $f16, $f18 # $f16 = 5./9.
      sub.s $f20, $f12, $f20 # $f20 = fahr - 32.
      mul.s $f0, $f16, $f20
      # $f0 = (5./9.)*(fahr-32.)
      jr $ra                  # fin de función
```

- Función que convierte una temperatura en grados Fahrenheit a grados Celsius.
- Las funciones que utilizan datos en coma flotante proporcionan los parámetros a través de los registros $\$f12 - \$f15$ y devuelven el resultado a través de $\$f0 - \$f3$.

47

12. Bibliografía

- ⇒ D.A. PATTERSON, J.L. HENNESSY. *Computer Organization and Design*. Morgan Kaufmann, 2005.
- ⇒ D.A. PATTERSON, J.L. HENNESSY. *Estructura y diseño de computadores*. Reverté, 2000.
- ⇒ D. SWEETMAN. *See MIPS Run*. Morgan Kaufmann, 2002.
- ⇒ E. FARQUHAR, P. BUNCE. *The MIPS Programmer's Handbook*. Morgan Kaufmann, 1994.
- ⇒ J. GOODMAN, K. MILLER. *A Programmer's View of Computer Architecture*. Saunders College Pub., 1993.
- ⇒ *MIPS32 Architecture For Programmers – Volume I: Introduction to the MIPS32 Architecture*. MIPS Technologies Inc., 2003.
- ⇒ *MIPS32 Architecture For Programmers – Volume II: The MIPS32 Instruction Set*. MIPS Technologies Inc., 2003.
- ⇒ *MIPS32 Architecture For Programmers – Volume III: The MIPS32 Privileged Resource Architecture*. MIPS Technologies Inc., 2003.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70