Lenguaje ensamblador : Conceptos

OBJETIVOS:

- **Conocer las directivas de definición de datos y reserva de espacio en memoria**
- **Estudiar los conceptos fundamentales para la realización de programas estructurados en lenguaje ensamblador del procesador MIPS R2000**
- **Conocer las Pseudoinstrucciones que ofrece el ensamblador para facilitar la programación**

Índice de Contenidos

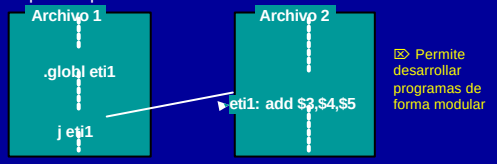
1. Directivas del Ensamblador

- Las directivas del ensamblador sirven para ubicar los datos y las instrucciones en la memoria del procesador
- Sintaxis:
  - Sus nombres empiezan con un punto y se emplean corchetes para indicar que uno o varios argumentos son opcionales
- Se agrupan en tres tipos:
  - Directivas de propósito variado
  - Directivas para reserva de posiciones de memoria
  - Directivas para indicar el inicio del área de datos y de instrucciones

1.1. Directivas de propósito variado

Directiva .globl

- Sintaxis: .globl etiqueta
- Descripción:
  - La función de esta directiva es declarar nombres de etiquetas o de funciones de forma global .
  - De esta manera desde un archivo se podrá acceder a etiquetas que se encuentran definidas en otros archivos.



Directivas de propósito variado

Directiva .end

- Descripción:
  - Esta directiva indica el final de un programa codificado en ensamblador
  - Ejemplo:

```
__start:   addi $2, $3, 12
           addi $4, $5, 12
           add  $6, $2, $4
           .end
```

Este programa realiza varias operaciones aritméticas y finaliza con la directiva .end

La etiqueta __start indica el punto de entrada al programa

1.2. Directivas para reserva de memoria

Directiva .space

- Sintaxis: .space n
- Descripción:
  - Se utiliza para reservar un número de *n* posiciones de memoria de un tamaño de 8 bits (1 byte) inicializándolas a cero.
  - Ejemplos:

```
.space 10    # Reserva 10 bytes en memoria.
.space1      # Reserva 1 byte.
```

Directivas para reserva de memoria

- **Directiva .ascii y asciiz**
  - Sintaxis: .ascii cadena [, cadena2, cadena3, …, cadenan]
    .asciiz cadena [, cadena2, cadena3, …, cadenan]
  - Descripción:
    - Se utiliza para almacenar una cadena de caracteres en memoria, (la cadena debe estar entre ")
    - La cadena finaliza con un caracter nulo si se utiliza la directiva .asciiz
    - Ejemplos:

```
.ascii "hola"   #Almacena en memoria la cadena "hola"
.asciiz "hola"  #Almacena en memoria la cadena "hola"
                #finalizándola con el caracter ascii 0
```


Directivas para reserva de memoria

- **Directiva .byte**
  - Sintaxis: .byte b1 [, b2, b3, …, bn]
  - Descripción:
    - Inicializa posiciones consecutivas de memoria con enteros en complemento a 2 de 8 bits (1 byte)
    - Ejemplos:

```
.byte 32          # Reserva una posicion de 8 bits en memoria y
                  # almacena el número 32.

.byte 63, 20      # Reserva en memoria dos posiciones
                  # consecutivas de 8 bits cada una cuyos
                  # contenidos serán los valores
                  # 63 y 20 respectivamente
```


Directivas para reserva de memoria

- **Directiva .half**
  - Sintaxis: .half h1 [, h2, h3, ..., hn]
  - Descripción:
    - Inicializa posiciones consecutivas de memoria con enteros en complemento a 2 de 16 bits (2 bytes)
    - Ejemplos:

```
.half 300, 2340   # Reserva dos posiciones consecutivas de 16
                  # bits en memoria cuyos contenidos serán
                  # los valores 300 y el 2340 respectivamente
```


Directivas para reserva de memoria

- **Directiva .word**
  - Sintaxis: .word w0 [, w1, w2, …, wn]
  - Descripción:
    - Inicializa posiciones consecutivas de memoria con enteros en complemento a 2 de 32 bits (4 bytes)
    - Ejemplos:

```
.word 6           # Reserva una posicion de 32 bits en memoria
                  # y almacena en ella el número 6

.word 1,2,3,7     # Reserva 4 posiciones de 32 bits en memoria
                  # e inicializa sus valores a 1,2, 3 y 7
                  # respectivamente
```


Directivas para reserva de memoria

- **Directivas .float y .double**
  - Sintaxis: .float f0 [, f0, f1 , ..., f n]
    .double d0 [, d0, d1 , …, dn]
  - Descripción:
    - Inicializa posiciones consecutivas de memoria de 32 bits (.float) y 64 bits (.double) con números reales representados en formato IEEE 754 de simple y doble precisión respectivamente
    - Los reales deben expresarse en decimal
    - Ejemplos:

```
.float 6.5    # Reserva una posicion de 32 bits en memoria
              # y almacena en ella el número real 6.5
.double 100.25  # Reserva una posicion de 64 bits en memoria
                # y almacena en ella el número real 100.25
```


1.3. Directivas para inicio de datos e instrucciones.

- **Directiva .data**
  - Sintaxis: .data [dir]
  - Descripción:
    - Sirve para ubicar datos a partir de la dirección de memoria indicada por **dir**
    - Si no se indica una dirección, por defecto se empieza en la dirección 0x10000000
    - Ejemplo:

```
.data 0x10000010
.word 8
.word 0x32
```

      - En memoria quedaría:

Dirección de Memoria	Contenido en Hexadecimal y en decimal	
0x10000010	0x00000008	8
0x10000014	0x00000032	50

Directivas para inicio de datos e instrucciones. (II)

- **Directiva .text**
  - Sintaxis: .text [dir]
  - Descripción:
    - Sirve para ubicar instrucciones a partir de la dirección de memoria indicada por **dir**
    - Si no se indica una dirección, por defecto se empieza en la dirección 0x00400000
    - Ejemplo:

```
        .text 0x400020
__start: and $8,$0,$0
         or $9,$0,$0
```

      - En memoria quedaría:

Dirección de Memoria	Contenido en Hexadecimal	Instrucción
0x400020	0x00004024	and $8,$0,$0
0x400024	0x00004825	or $9,$0,$0

2. Alineación de datos en memoria

- **Alineación:**
  - **.space, .byte, . ascii y .asciiz reservan memoria a partir de cualquier dirección**
  - **.half reserva memoria a partir de direcciones múltiplos de 2 (pares)**
  - **.word y .float reservan memoria a partir de direcciones múltiplos de 4**
  - **.double reserva memoria a partir de direcciones múltiplos de 8**
  - **El simulador del MIPS almacena los datos en formato Little Endian**
- **Ejemplo 1:**

```
.data 0x10000020
.word 0x20000001
.space 1
.word 0x10
.byte 0x1, 0x2, 0x3
.double 1.0  #0x3FF0000000000000 IEEE doble precisión
```

Contenido en hexadecimal (31 ..... 24 23 ..... 16 15 ..... 8 7 ..... 0)	Dirección mem.
0x20000001	0x10000020
0x00	0x10000024
0x00000010	0x10000028
0x03 | 0x02 | 0x01	0x1000002C
0x00000000	0x10000030
0x3FF00000	0x10000034

2. Alineación de datos en memoria

- Ejemplo 2:

```
.data 0x10000020
.word 0x20000001
.space 1
.byte 0x1, 0x2, 0x3
.word 0x10
.double 1.5
.space 1
.half 0xFF86
.space 2
.asciiz "HOLA"
```

Contenido en hexadecimal (31 ..... 24 23 ..... 16 15 ..... 8 7 ..... 0)	Dirección mem.
0x20000001	0x10000020
0x03 | 0x02 | 0x01 | 0x00	0x10000024
0x00000010	0x10000028
	0x1000002C
Ejercicio	0x10000030
Ejercicio	0x10000034
0xFF86 | | 0x00	0x10000038
O | H | 0x00 | 0x00	0x1000003C
| NULL | A | L	0x10000040

3. Pseudoinstrucciones

- **Son un conjunto de instrucciones que el lenguaje ensamblador incorpora además de las instrucciones máquina del procesador**
- **El programador puede usar las pseudoinstrucciones como si se trataran de instrucciones máquina soportadas por el procesador**
- **Es el programa ensamblador quien se encarga de sustituir dichas pseudoinstrucciones por las instrucciones máquina correspondientes (de una a cuatro por cada pseudoinstrucción)**

Pseudoinstrucciones

- **Pseudoinstrucción li (load inmediate)**
  - **Sintaxis: li Rdest, inm**
  - **Descripción:**
    - **Carga en el registro indicado por Rdest el entero con signo en complemento a 2 de 16 bits inm**
    - **Ejemplo:**

```
li $5,32        # carga en el registro 5 el número decimal 32
li $4,0x32      # carga en el registro 4 el valor decimal 50
```

    - **Código equivalente:**

	Instrucciones máquina equivalentes
li $5, 32	ori $5, $0, 32
li $4, 0x32	ori $4, $0, 0x32

Pseudoinstrucciones

- **Pseudoinstrucción la (load address)**
  - **Sintaxis: la R, etiqueta**
  - **Descripción:**
    - **Carga en el registro R la dirección de memoria a la que hace referencia etiqueta**
    - **Ejemplo:**

La etiqueta Vect hace referencia a la dirección del primer elemento del vector, ubicado en la dirección 0x10000008

```
        .data 0x10000008
Vect    .word 0x1
        .word 0x2
        .text 0x400000
__start la $3,Vect
        end
```

Código Equivalente

```
lui $1,0x1000
ori $3,$1,0x8
```

Pseudoinstrucciones

- **Pseudoinstrucciones seq, sge, sgt, sle y sne**
  - **Sintaxis: sxx rd, rs1, rs2**
  - **Descripción:**
    - **Si se cumple la relación de equivalencia, indicada por xx (eq, ge, gt, le y ne), entre los registros rs1 y rs2, se almacena un 1 en el registro rd, si no se cumple, se almacena un 0 en el registro rd**
    - **Ejemplos:**

	Pseudoinst.	Comentario	Código máquina equivalente
__start:	seq $3,$4,$5	# si [$4] = [$5] entonces # [$3]=1 # si no [$3]=0	ori $3,$0,1 beq $5,$4, +2 ori $3,$0,0

Pseudoinstrucciones

Pseudoinst.	Comentario	Código máquina equivalente
sge $3,$4,$5	# si [$4] ≥ [$5] entonces # [$3]=1 # si no [$3]=0	slt $3,$5,$4 bne $5,$4,+2 ori $3,$0,1
sgt $3,$4,$5	# si [$4] > [$5] entonces # [$3]=1 # si no [$3]=0	slt $3,$5,$4
sle $3,$4,$5	# si [$4] ≤ [$5] entonces # [$3]=1 # si no [$3]=0	slt $3,$4,$5 bne $5,$4,+2 ori $3,$0,1
sne $3,$4,$5	# si [$4] ≠ [$5] entonces # [$3]=1 # si no [$3]=0	ori $3,$0,1 bne $5,$4,+2 ori $3,$0,0

Pseudoinstrucciones

- **Pseudoinstrucciones beqz y bnez**
  - **Sintaxis: beqz rs, etiqueta**
    bnez rs, etiqueta
  - **Descripción:**
    - **Si rs es igual a cero (beqz) o distinto de cero (bnez) el programa sigue su ejecución a partir de la etiqueta**
    - **Ejemplos:**

	Pseudoinst.	Comentario	Código máquina equivalente
	.text		
start:	ori $2, $0, 1		
	beqz $5, ig	# si [$5] = 0 entonces cp = ig	beq $5, $0, ig
di:	sub $5, $5, $2		
	bnez $5, di	# si [$5] ≠ 0 entonces cp = di	bne $5, $0, di
ig:	.end		

Pseudoinstrucciones

- **Pseudoinstrucciones bge, bgt y ble**
  - **Sintaxis: bxx rs1, rs2, etiqueta**
  - **Descripción:**
    - **Si se cumple la relación de equivalencia, indicada por xx (ge, gt y le), entre los registros rs1 y rs2, se salta a la dirección a la que hace referencia la etiqueta**
    - **Ejemplos:**

Pseudoinst.	Comentario	Código máquina equivalente
bge $3, $4, et1	# si [$3] ≥ [$4] entonces cp = et1	slt $1, $3, $4 beq $1, $0, et1
bgt $3, $4, et1	# si [$3] > [$4] entonces cp = et1	slt $1, $4, $3 bne $1, $0, et1
ble $3, $4, et1	# si [$3] ≤ [$4] entonces cp = et1	slt $1, $4, $3 beq $1, $0, et1

Pseudoinstrucciones

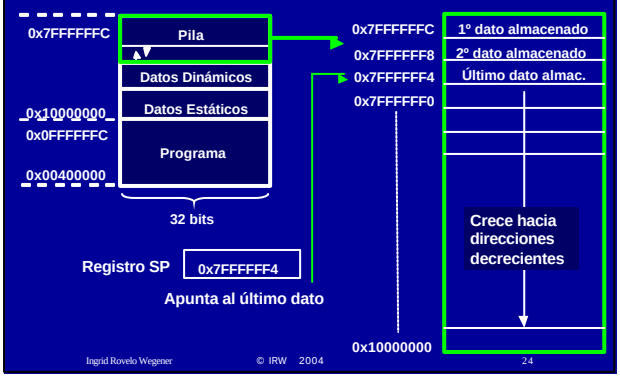
- **Resumen:**

Pseudoinstrucción	Posible secuencia de instrucciones máquina equivalentes
li rd, inm	ori
la rd, dirección	lui, ori
seq rd, rs1, src2	ori, beq, ori
sge rd, rs1, rs2	slt, bne, ori
sgt rd, rs1, rs2	slt
sle rd, rs1, rs2	slt, bne, ori
sne rd, rs1, rs2	ori, bne, ori
beqz rs, etiqueta	beq
bnez rs, etiqueta	bne
bge rs1, rs2, etiqueta	slt, beq
bgt rs1, rs2, etiqueta	slt, bne
ble rs1, rs2, etiqueta	slt, beq

4. Gestión de la pila

- Pila o Segmento de Pila
  - Zona de memoria donde se almacenan datos siguiendo una estructura LIFO (del inglés *last in first out*), el último dato en almacenarse es el primero en sacarse. Cada dato ocupa 4 bytes en la pila.
  - Para gestionar la pila se utiliza un registro especial llamado puntero de pila (SP: *stack pointer*)
  - El SP contiene, en cualquier instante, la dirección de memoria correspondiente a la posición que ocupa en la pila el último dato almacenado. Se dice que el registro SP siempre apunta al último dato almacenado
  - El registro SP se corresponde con el registro $29 del banco de registros y se puede referenciar mediante $29 o **$sp**
  - Existen dos operaciones básicas sobre la pila
    - Operación de inserción en la pila (PUSH)
    - Operación de extracción desde la pila (POP)
  - La pila crece hacia direcciones decrecientes de memoria

Gestión de la pila (II)



4. Gestión de la pila

- **Operación de Inserción (Apilado o PUSH)**
  - **Consiste en almacenar un dato en la pila**
  - **El dato a almacenar será el contenido de un registro de 32 bits por lo que se ocuparán 4 bytes en memoria (1 palabra)**
  - **Pasos a realizar:**
    - **Decrementar el puntero de pila en 4 bytes. De esta manera se actualiza su contenido con la nueva posición donde se almacenará el dato**
    - **Almacenar el dato en la nueva posición a la que apunta el puntero de pila**
  - **Ejemplo. Si se desea almacenar en la pila el contenido del registro $10 se deben ejecutar las dos instrucciones siguientes:**

```
addi $29, $29, -4
sw $10, 0($29)
```

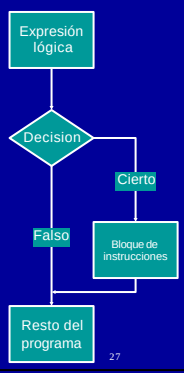

4. Gestión de la pila

- **Operación de Extracción (Desapilado o POP)**
  - Consiste en cargar un dato desde la pila hacia un registro
  - Pasos a realizar:
    - Cargar el dato actual apuntado por el registro SP en un registro de propósito general
    - Incrementar el puntero de pila en 4
  - Ejemplo. Si se desea recoger de la pila un dato y guardarlo en el registro $10 las instrucciones a ejecutar son:

```
lw $10, 0($29)
addi $29, $29, +4
```


Estructuras de control de flujo

- **Estructura If then**
  - **Forma más sencilla de bifurcación condicional**
  - **Esta sentencia lo primero que hace es evaluar una expresión**
  - **Si la condición evaluada es cierta se ejecuta un bloque de instrucciones**



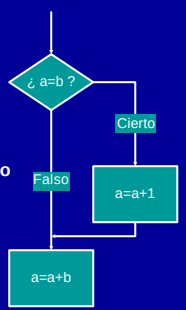
Estructuras de control de flujo

- **Ejemplo**
  - **Se dispone del siguiente fragmento de programa escrito en Pascal:**

```
if a=b then a:=a+1;
a:=a+b;
```

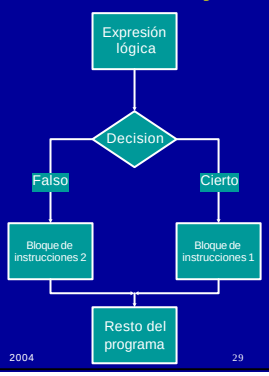
  - **Diseñar el diagrama de flujo y escribir el programa en ensamblador equivalente suponiendo que la variable a está contenida en el registro $2 y la variable b está contenida en el registro $3**
  - **Solución:**

```
        bne $2,$3,salida
        addi $2,$2,1
salida: add $2,$2,$3
```



Estructuras de control de flujo

- **Estructura if-then-else**
  - Evalúa una condición y si el resultado de la evaluación es cierto la ejecución del programa toma un camino, en caso contrario el programa ejecuta otro camino alternativo

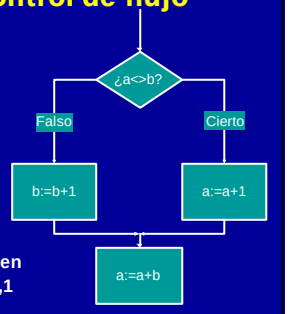


Estructuras de control de flujo

- Ejemplo
  - Se dispone del siguiente fragmento de programa escrito en Pascal:

```
if a<>b then a:=a+1
   else b:=b+1;
      a:=a+b;
```

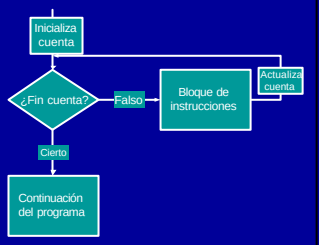
  - Diseñar el diagrama de flujo y escribir el programa en ensamblador equivalente suponiendo que las variables a y b están contenidas en los registros $2 y $3 respectivamente
  - Solución:




```
        bne $2,$3,then
else:   addi $3,$3,1
        j cont
then:   addi $2,$2,1
cont:   add $2,$2,$3
```


Estructuras de control de flujo

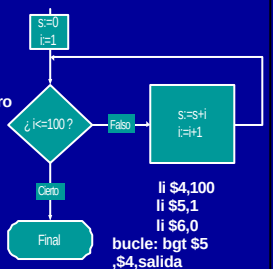
- Estructura for
  - Permite al programador la ejecución repetitiva de un bloque de instrucciones
  - Esta estructura lleva asociado un contador que indica el número de veces que ha de ejecutarse el bloque de instrucciones



Estructuras de control de flujo

- Ejemplo
  - Obtener el diagrama de flujo del algoritmo que realiza la suma de los números desde 1 hasta 100
  - Implementar el programa ensamblador correspondiente asumiendo que el registro $6 al final contendrá la suma s, que el registro $5 es el contador que llevará la cuenta de 1 a 100 y que el registro $4 contendrá el número máximo (100) a sumar
  - Solución:

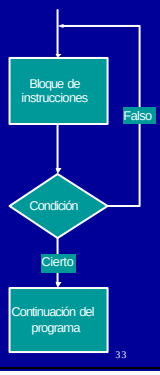
$$s = \sum_{i=1}^{100} i$$




```
        li $4,100
        li $5,1
        li $6,0
bucle:  bgt $5,$4,salida
        add $6,$6,$5
        addi $5,$5,1
        j bucle
salida: .end
```

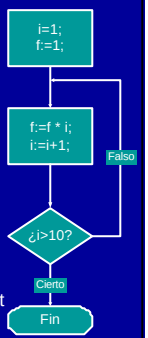

Estructuras de control de flujo

- Estructura Repeat-Until
  - Permite, al igual que for, procesar un bloque de instrucciones de forma repetitiva
  - A diferencia de for, ejecuta el bloque hasta que se cumple una condición de salida del bucle
  - La ejecución del bloque de instrucciones se realiza al menos una vez



Estructuras de control de flujo

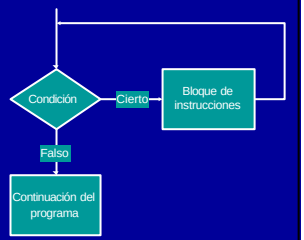
- Ejemplo
  - Diséñese el diagrama de flujo y la correspondiente implementación en ensamblador de un algoritmo que calcule el factorial de un número N (mayor que cero). Como ejemplo N=10
  - Solución:




```
        li $2,1          # $2=i=1
        li $3,1          # $3=f=1
        li $4,10         # $4=10
repeat: mult $2,$3       # f*i
        mflo $3          # $3=f*i
        addi $2,$2,1     # incrementar el contador i
        ble $2,$4, repeat  # Si  i<=10  saltar a repeat
        .end
```


Estructuras de control de flujo

- Estructura While
  - Permite procesar un bloque de instrucciones de forma repetitiva mientras se cumpla una condición
  - A diferencia de repeat-until, se comprueba la condición antes de que el bloque de instrucciones se ejecute
  - Puede suceder que el bloque de instrucciones nunca se ejecute



Estructuras de control de flujo

Ejemplo

- Obtener el diagrama de flujo y el programa en ensamblador del algoritmo que calcula el factorial de N (mayor que cero). En este caso, para N=10, utilizar la estructura while

- Solución:

```
        li $2,1          # i=1
        li $3,1          # f=1
        li $4,10         # $4 contendrá el 10
while:  beq $2,$4,fin    # Si i=10 salta a fin
        addi $2,$2,1     # inc. el contador i
        mult $2,$3       # f*i
        mflo $3          # $3=f*i
        j while          # Saltar al bucle
        while
fin:    .end
```