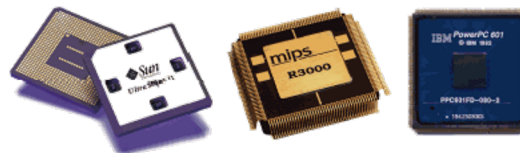


Tema 2A: Ensamblador MIPS

Fundamentos de Ordenadores y Sistemas Operativos

Mario Martínez Zarzuela
marmar@tel.uva.es



Preguntas a responder

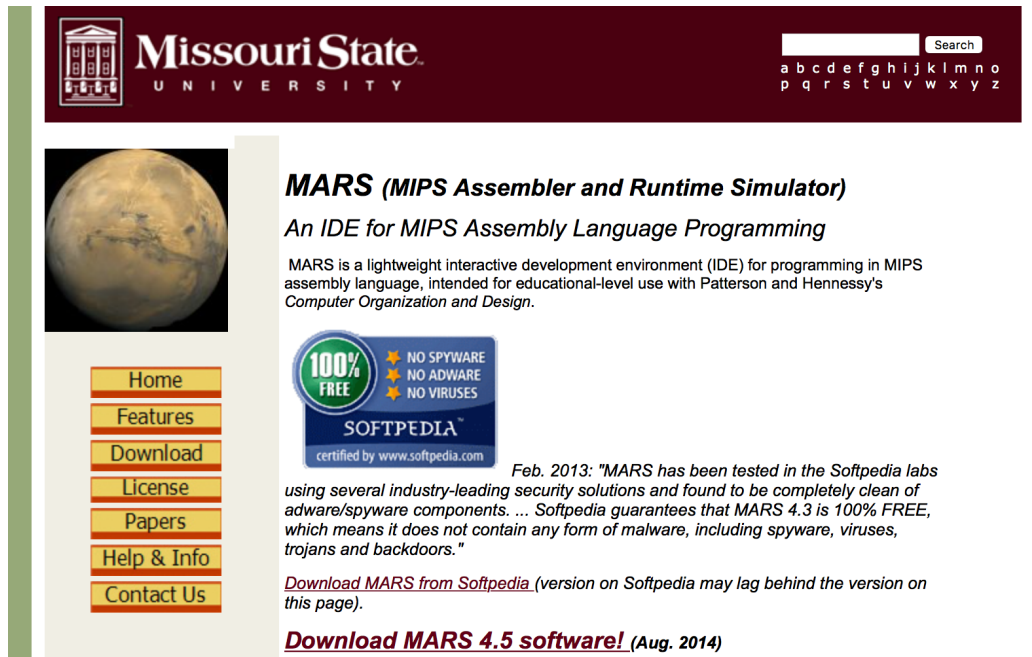
1. ¿**Qué necesito** para programar en MIPS en mi ordenador?
2. ¿Sería posible ejecutar ensamblador MIPS en mi ordenador personal **de forma nativa**?
¿Puede entender este código mi procesador?
3. ¿En qué se parece y en qué se **diferencia** de la programación en lenguajes de más alto nivel como C?

Contenidos

- Mi primer programa en ensamblador
- Memoria principal frente a fichero de registros
 - Ubicación y organización de la memoria principal
 - Ubicación y organización del fichero de registros
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- Accediendo a los elementos de un vector
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

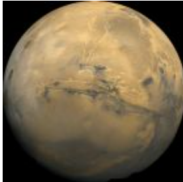
Descarga MARS en el ordenador

- <http://courses.missouristate.edu/KenVollmar/MARS/index.htm>
- Requiere tener instalado www.java.com



Missouri State
UNIVERSITY

Search
a b c d e f g h i j k l m n o
p q r s t u v w x y z



MARS (MIPS Assembler and Runtime Simulator)
An IDE for MIPS Assembly Language Programming

MARS is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use with Patterson and Hennessy's *Computer Organization and Design*.

100% FREE
NO SPYWARE
NO ADWARE
NO VIRUSES
SOFTPEDIA™
certified by www.softpedia.com

Feb. 2013: "MARS has been tested in the Softpedia labs using several industry-leading security solutions and found to be completely clean of adware/spyware components. ... Softpedia guarantees that MARS 4.3 is 100% FREE, which means it does not contain any form of malware, including spyware, viruses, trojans and backdoors."

[Download MARS from Softpedia](#) (version on Softpedia may lag behind the version on this page).

[Download MARS 4.5 software!](#) (Aug. 2014)

Mi primer programa en ensamblador

```
1 void main()  
2 {  
3     int var1=14;  
4     int var2=-6;  
5  
6     var1=var1+var2;  
7 }  
8
```

SEGMENTO DE DATOS

```
1  
2 var1: .word 14  
3 var2: .word -6  
4
```

```
5 .text  
6 main: lw $s1, var1($zero)  
7      lw $s2, var2($zero)  
8  
9      add $s1,$s1,$s2  
0      sw $s1, var1($zero)  
1  
2 end:  nop  
-
```

SEGMENTO DE INSTRUCCIONES

Mi primer programa en ensamblador

- Trata de intuir qué hace cada una de las líneas del código anterior en ensamblador
- Comenta con tus compañeros

Algunas instrucciones para comenzar

add	rd, rs, rt	# addition	$rd \leftarrow rs + rt$
sub	rd, rs, rt	# subtraction	$rd \leftarrow rs - rt$

lw	rt, imm(rs)	# load word from memory	
sw	rt, imm(rs)	# store word to memory	

addi	rt, rs, imm	# addition	$rd \leftarrow rs + imm$
subi	rt, rs, imm	# subtraction	$rd \leftarrow rs - imm$

Mi primer programa en ensamblador

- Escribe el programa en el simulador MARS
 - Realiza esta configuración y trata de averiguar qué conseguimos:

Settings → Memory Configuration → Compact, Data at Address 0

- Tras el ensamblado, comprueba el contenido de la memoria y las direcciones de los datos
- Ejecuta el programa línea a línea y comprueba la modificación de la memoria y de los registros

Mi primer programa en ensamblador

- ¿Puedes explicar a qué se deben estas diferencias?

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	12288	0x8c110000	lw \$17,0(\$0)	6: lw \$s1, var1(\$zero) #Est...
<input type="checkbox"/>	12292	0x8c120004	lw \$18,4(\$0)	7: A: lw \$s2, var2(\$zero)
<input type="checkbox"/>	12296	0x02328020	add \$16,\$17,\$18	9: suma: add \$s0,\$s1,\$s2
<input type="checkbox"/>	12300	0xac100000	sw \$16,0(\$0)	10: sw \$s0, var1(\$zero)
<input type="checkbox"/>	12304	0x00000000	nop	12: end: nop

Data Segment						
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)
0	14	-6	0	0	0	0
32	0	0	0	0	0	0
64	0	0	0	0	0	0
96	0	0	0	0	0	0
128	0	0	0	0	0	0

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,4097	6: main: lw \$s1, var1(\$zero)
<input type="checkbox"/>	0x00400004	0x00200821	addu \$1,\$1,\$0	
<input type="checkbox"/>	0x00400008	0x8c310000	lw \$17,0(\$1)	
<input type="checkbox"/>	0x0040000c	0x3c011001	lui \$1,4097	7: lw \$s2, var2(\$zero)
<input type="checkbox"/>	0x00400010	0x00200821	addu \$1,\$1,\$0	
<input type="checkbox"/>	0x00400014	0x8c320004	lw \$18,4(\$1)	
<input type="checkbox"/>	0x00400018	0x02328820	add \$17,\$17,\$18	9: add \$s1,\$s1,\$s2
<input type="checkbox"/>	0x0040001c	0x3c011001	lui \$1,4097	10: sw \$s1, var1(\$zero)
<input type="checkbox"/>	0x00400020	0x00200821	addu \$1,\$1,\$0	

Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	14	-6	0	0
0x10010020	0	0	0	0
0x10010040	0	0	0	0
0x10010060	0	0	0	0
0x10010080	0	0	0	0
0x100100a0	0	0	0	0

Contenidos

- Mi primer programa en ensamblador
- **Memoria principal frente a fichero de registros**
 - **Ubicación y organización de la memoria principal**
 - Ubicación y organización del fichero de registros
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- Accediendo a los elementos de un vector
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

Memoria Principal

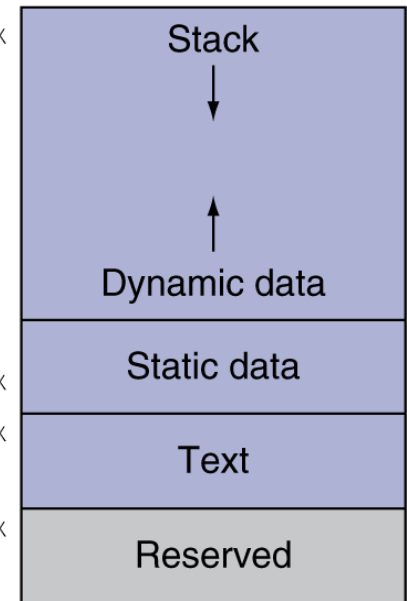
- Organización lógica:
 - **Text**: código de los programas
 - **Static data**: variables globales del programa
 - Pej: `int a; //Declarado fuera de main()`
 - **Dynamic data** (heap): espacio de memoria reservado dinámicamente
 - Pej: `malloc()`, `realloc()`
 - **Stack**: pila de almacenamiento para variables temporales
 - Pej: variables locales a una función

\$sp → 7fff fffc_{hex}

\$gp → 1000 8000_{hex}
1000 0000_{hex}

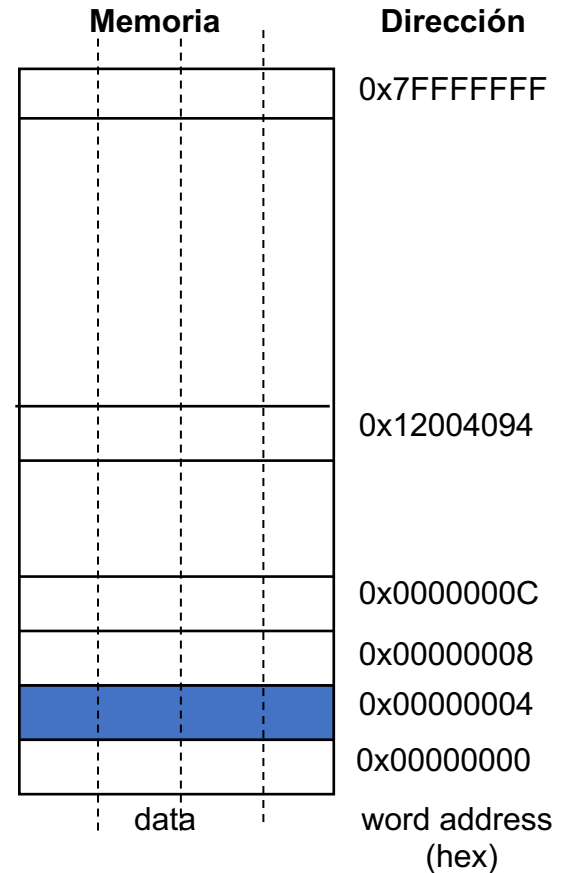
pc → 0040 0000_{hex}

0



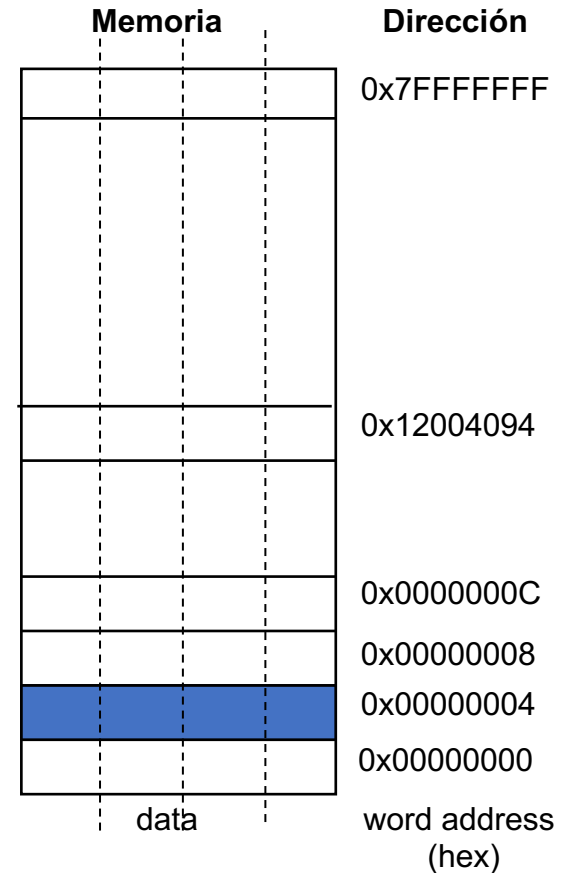
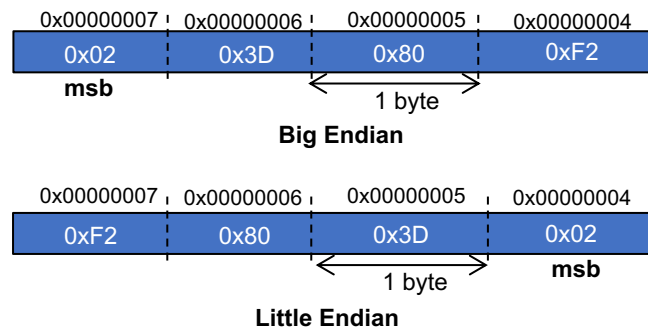
Memoria Principal

- Direcciones de memoria
 - Una dirección identifica un byte
 - Palabras alineadas en memoria: direcciones múltiplos de 4 bytes
- Datos en memoria
 - Un int o un float ocupan 4 bytes
 - Un char ocupa siempre 1 byte



Memoria Principal

- Datos en memoria
 - Representación **Big Endian** o **Little Endian**
 - Para almacenar datos de más de un byte en memoria
 - Orden según less/most significant bit
 - Ej. Almacenar valor 0x023D80F2



Contenidos

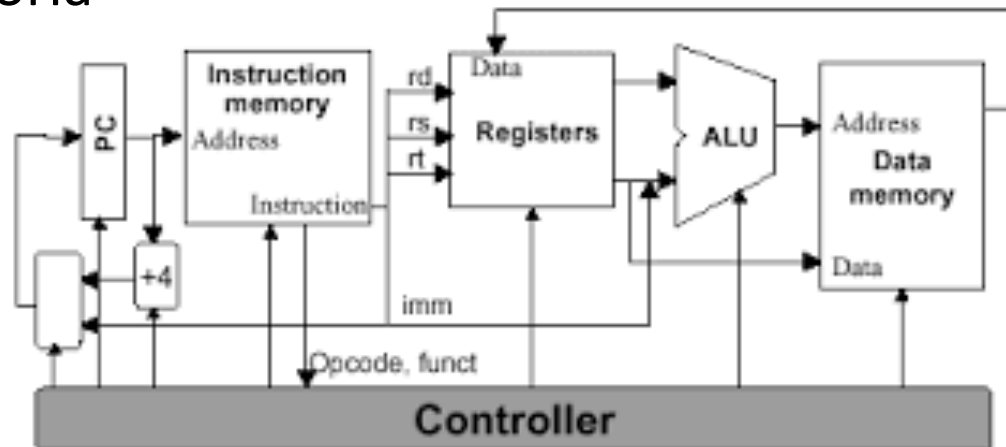
- Mi primer programa en ensamblador
- **Memoria principal frente a fichero de registros**
 - Ubicación y organización de la memoria principal
 - **Ubicación y organización del fichero de registros**
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- Accediendo a los elementos de un vector
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

Fichero de Registros

- Pequeño almacenamiento temporal **dentro** del procesador
 - En MIPS hay 32 registros en los que cabe una palabra (**word**) de 32 bits
- Acceso más rápido que la memoria principal
 - La **memoria principal** está fuera del procesador
 - Más adelante hablaremos de cachés, que sí que están dentro del procesador

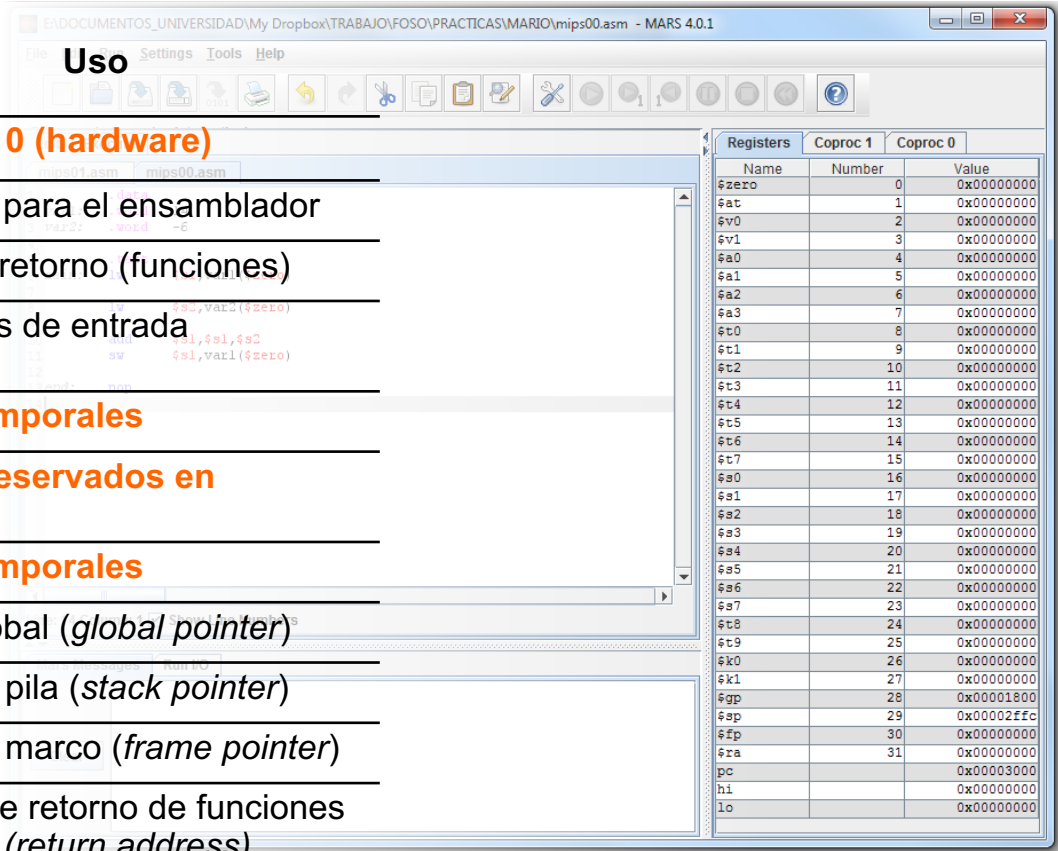
Fichero de Registros

- Necesarios para operar:
 - Cargar (**load**) en los registros operandos desde la memoria
 - Realizar la operación en la Unidad Aritmético Lógica (ALU)
 - Almacenar (**store**) datos de los registros hacia la memoria



Registros en MIPS

Nombre	Número	Uso
\$zero	0	Constante 0 (hardware)
\$at	1	Reservado para el ensamblador
\$v0 - \$v1	2-3	Valores de retorno (funciones)
\$a0 - \$a3	4-7	Argumentos de entrada (funciones)
\$t0 - \$t7	8-15	Valores temporales
\$s0 - \$s7	16-23	Valores preservados en funciones
\$t8 - \$t9	24-25	Valores temporales
\$gp	28	Puntero global (<i>global pointer</i>)
\$sp	29	Puntero de pila (<i>stack pointer</i>)
\$fp	30	Puntero de marco (<i>frame pointer</i>)
\$ra	31	Dirección de retorno de funciones (hardware) (<i>return address</i>)



Contenidos

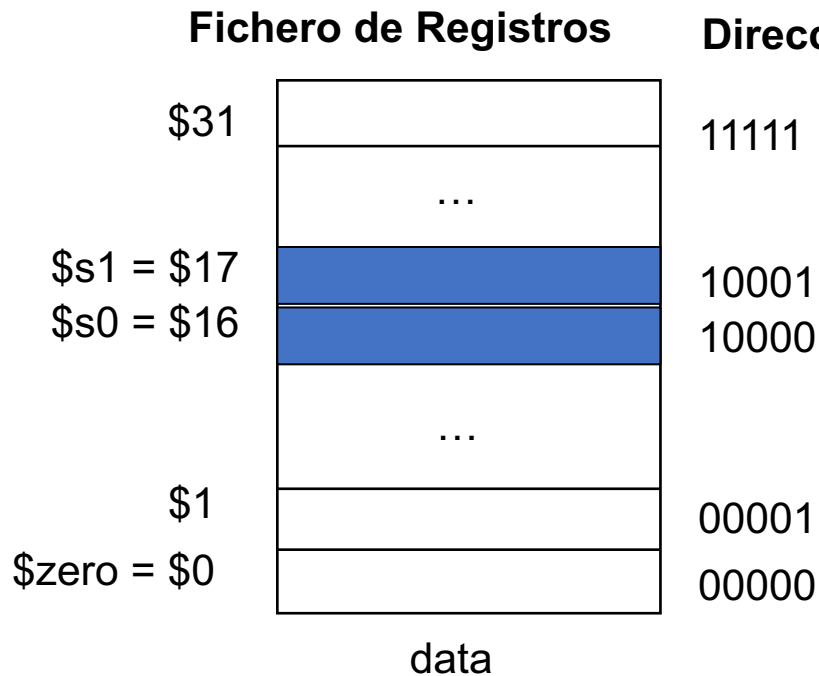
- Mi primer programa en ensamblador
- **Memoria principal frente a fichero de registros**
 - Ubicación y organización de la memoria principal
 - Ubicación y organización del fichero de registros
 - **Utilización combinada para realizar operaciones**
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- Accediendo a los elementos de un vector
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

Ubicación de los datos en registros

```
.data
f:  .word    5
g:  .word    7
...
.text
lw  $s0, f($zero)
lw  $s1, g($zero)
lw  $s2, h($zero)
lw  $s3, i($zero)
lw  $s4, j($zero)
```

- Dibuja el fichero de registros con sus direcciones, representando en qué posiciones se encuentran los valores de f y g.

Ubicación de los datos en registros



- Dibuja el fichero de registros con sus direcciones, representando en qué posiciones se encuentran los valores de f y g.

Laboratorio

- En MARS, inicializa una palabra del segmento de datos a un valor hexadecimal.
- Comprueba si el almacenamiento en memoria sigue la filosofía Big Endian o Little Endian.

Laboratorio

- Traduce los siguientes programas en C a ensamblador MIPS
 - Ejercicio 2.1

```
1 void main()
2 {
3     int f=0;
4     int g=1;
5     int h=4;
6     int i=3;
7     int j=5;
8
9     f=(g+h) - (i+j) ;
10
11 }
12
```

Contenidos

- Mi primer programa en ensamblador
- Memoria principal frente a fichero de registros
 - Ubicación y organización de la memoria principal
 - Ubicación y organización del fichero de registros
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- Accediendo a los elementos de un vector
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

Ejemplos de tipos de instrucciones

- Tipo R: todos los operandos en registros (**r**)

add	rd, rs, rt	# addition	$rd \leftarrow rs + rt$
sub	rd, rs, rt	# subtraction	$rd \leftarrow rs - rt$

- Tipo I: incluyen algún dato inmediato (**immediate**)

lw	rt, imm(rs)	# load word from memory
sw	rt, imm(rs)	# store word to memory

beq	rs, rt, imm	# branch if $rs == rt$
bne	rs, rt, imm	# branch if $rs \neq rt$

addi	rt, rs, imm	# addition	$rd \leftarrow rs + imm$
subi	rt, rs, imm	# subtraction	$rd \leftarrow rs - imm$

- Tipo J: para salto (**jump**)

j	addr	#jump
---	------	-------

Contenidos

- Mi primer programa en ensamblador
- Memoria principal frente a fichero de registros
 - Ubicación y organización de la memoria principal
 - Ubicación y organización del fichero de registros
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- **Programas con control de flujo**
 - Ejemplo de estructura if-else
- Accediendo a los elementos de un vector
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

Control de flujo: completa las instrucciones que faltan

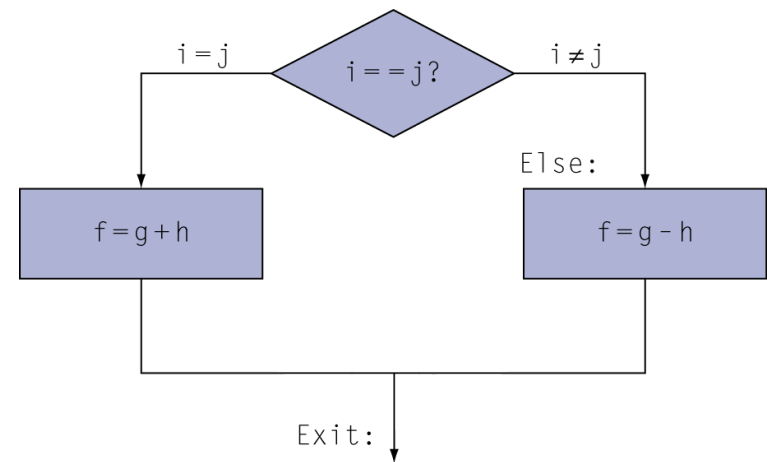
```
lw $s0, f($zero)
lw $s1, g($zero)
lw $s2, h($zero)
lw $s3, i($zero)
lw $s4, j($zero)
```

Etiquetas

```
add $s0, $s1, $s2
```

```
Else: sub $s0, $s1, $s2
Exit: nop
```

```
if (i==j)
    f = g+h;
else
    f = g-h;
```



Control de flujo: completa las instrucciones que faltan

```
lw $s0, f($zero)
lw $s1, g($zero)
lw $s2, h($zero)
lw $s3, i($zero)
lw $s4, j($zero)
```

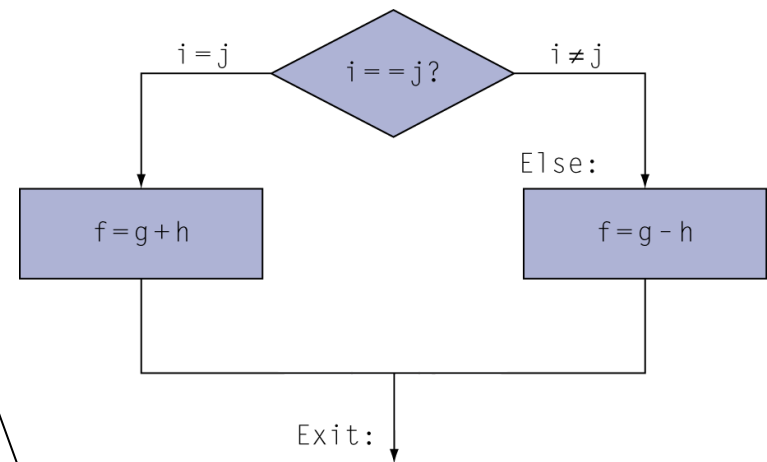
Etiquetas

```
Else: sub $s0, $s1, $s2
Exit: nop
```

```
add $s0, $s1, $s2
```

Compilación traduce etiquetas a direcciones de memoria virtuales

```
if (i==j)
    f = g+h;
else
    f = g-h;
```



Control de flujo: completa las instrucciones que faltan

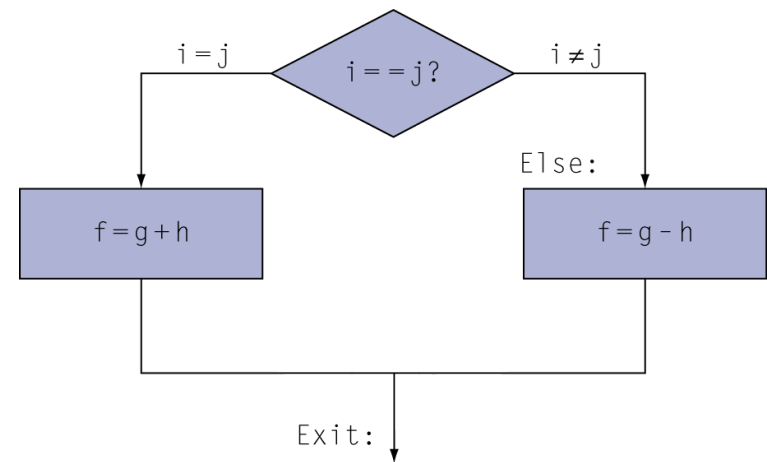
```
lw $s0, f($zero)
lw $s1, g($zero)
lw $s2, h($zero)
lw $s3, i($zero)
lw $s4, j($zero)
```

Etiquetas

```
Else: sub $s0, $s1, $s2
Exit: nop
```

Compilación traduce etiquetas a direcciones de memoria virtuales

```
if (i==j)
    f = g+h;
else
    f = g-h;
```



Laboratorio

- Traduce los siguientes programas en C a ensamblador MIPS
 - Ejercicio 2.2
 - Escribe en MARS el programa que ejecuta el código anterior
 - Acuérdate de inicializar el segmento de datos
 - Piensa en dos maneras alternativas de modificar el código para realizar el siguiente control de flujo

```
if (i!=j)
    f = g+h;
else
    f = g-h;
```

Contenidos

- Mi primer programa en ensamblador
- Memoria principal frente a fichero de registros
 - Ubicación y organización de la memoria principal
 - Ubicación y organización del fichero de registros
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- **Accediendo a los elementos de un vector**
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

Vectores

- Vectores de enteros: inicializar memoria separando con comas

`h: .word 7`

`A: .word 4,7,9,10`

- Situación en memoria

Label	Address ▲	Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
vector.asm		0	7	4	7	9	10
h	0	32	0	0	0	0	0
A	4	64	0	0	0	0	0

- ¿En qué posición de memoria se encuentra A[3]?
 - Es la dirección del 4º elemento del vector A
 - 4 bytes (comienzo de A) + 3 elementos x 4 bytes

Modos de funcionamiento de lw y sw

lw rt, imm(rs) # load word from memory
sw rt, imm(rs) # store word to memory

- Para expresar la dirección de memoria a acceder (lw) o a escribir (sw) hay dos opciones:

	imm	rs	Dirección accedida (suma)
Opción A)	dirección base	desplazamiento	
	4	12	16
Opción B)	desplazamiento	dirección base	
	12	4	16

Escribe en código MIPS

- Opción A) Ejemplo

$A[3] = h + A[3];$

```
h:      .data
        .word 7
A:      .word 4,7,9,10
        .text
        lw    $s2,h($zero)
        addi  $t0,$zero,12
        lw    $s0,A($t0)
        add   $s1,$s2,$s0
        sw    $s1,A($t0)
```

Memoria

.....	
A[3]	0x00000010
A[2]	0x0000000C
A[1]	0x00000008
A[0]	0x00000004
h	0x00000000

data

Fichero de Registros

\$31		11111
	...	
\$s3 = \$19		10011
\$s2 = \$18	7	10010
\$s1 = \$17	17	10001
\$s0 = \$17	10	10000
	...	
\$zero = \$0		00000

data

Escribe en código MIPS

- Opción B) Ejemplo

$A[3] = h + A[3];$

```
h:      .data
        .word 7
A:      .word 4,7,9,10
        .text
lw      $s2,h($zero)
la      $s3,A($zero)
lw      $s0,12($s3)
add     $s1,$s2,$s0
sw      $s1,12($s3)
```

Memoria

.....	
A[3]	0x00000010
A[2]	0x0000000C
A[1]	0x00000008
A[0]	0x00000004
h	0x00000000

data

la

Fichero de Registros

\$31		11111
	...	
\$s3 = \$19	4	10011
\$s2 = \$18	7	10010
\$s1 = \$17	17	10001
\$s0 = \$17	10	10000
	...	
\$zero = \$0		00000

data

Contenidos

- Mi primer programa en ensamblador
- Memoria principal frente a fichero de registros
 - Ubicación y organización de la memoria principal
 - Ubicación y organización del fichero de registros
 - Utilización combinada para realizar operaciones
- Tres tipos de instrucciones: R, I, J
- Programas con control de flujo
 - Ejemplo de estructura if-else
- **Accediendo a los elementos de un vector**
 - Dos modos de utilizar lw y sw
 - Ayudándonos de la multiplicación mediante sll

¿Qué ocurre si el acceso al elemento del array depende de otra variable?

```
i = 3;  
A[3] = h + A[i];
```

- El programa tiene que ser válido independientemente del valor de i
- ¿Cómo calculamos ahora el desplazamiento en memoria?
 - Piensa en la **alineación de los datos** en memoria
- Modifica el código de las transparencias 17 y 18 para utilizar ahora $A[i]$ en lugar de $A[3]$

Operaciones de multiplicación y división por p^n en números en base p

- Funcionan en cualquier base, siendo n el número de dígitos que desplazamos
- Hacia la izquierda \rightarrow multiplicamos por p^n
 - Rellenar con ceros los espacios
- Hacia la derecha \rightarrow dividimos por p^n

Base	Número	Izq $n=1$	Izq $n=2$	Der $n=1$	Der $n=2$
Decimal ($p=10$)	178	1780	17800	17	1
Binaria ($p=2$)	010011 (19)	0100110 (38)	01001100 (76)	01001 (9)	0100 (4)

Operaciones de desplazamiento de bits

- Tipo R: todos los operandos en registros (**r**)

`sll rd, rt, sh #shift left logical`

`srl rd, rt, sh #shift right logical`

- El valor almacenado en sh es el número de bits que se desplaza el dato en el registro rt
- Ejemplos:

`addi $s0, $zero, 6`

`sll $s1, $s0, 1 #6 x 2`

`sll $s1, $s0, 2 #6 x 4`

`sll $s1, $s0, 3 #6 x 8`

`srl $s1, $s0, 1 #6 / 2`

¿Qué ocurre si el acceso al elemento del array depende de otra variable?

```
i = 3;  
A[3] = h + A[i];
```

```
.data  
h:    .word 7  
A:    .word 4,7,9,10  
i:    .word 3  
.text  
lw     $s2,h($zero)  
lw     $t1,i($zero)  


---

addi   $t0,$zero,12    # 3 x 4 bytes = 12 bytes  
sll    $t0,$t1,¿?  
lw     $s0,A($t0)      # load word  
add    $s1,$s2,$s0     # h + A[3]  
sw     $s1,A($t0)      # store word
```

¿Qué ocurre si el acceso al elemento del array depende de otra variable?

```
i = 3;  
A[3] = h + A[i];
```

```
h: .data  
   .word 7  
A: .word 4,7,9,10  
i: .word 3
```

Número	Izq n=1	Izq n=2
000011 (3)	0000110 (6)	01001100 (12)

```
.text
```

```
lw $s2,h($zero)
```

```
lw $t1,i($zero)
```

```
addi $t0,$zero,12 # 3 x 4 bytes = 12 bytes
```

```
sll $t0,$t1,2
```

```
lw $s0,A($t0) # load word
```

```
add $s1,$s2,$s0 # h + A[3]
```

```
sw $s1,A($t0) # store word
```


Laboratorio

- Ensambla los siguientes códigos a MIPS
 - Utiliza instrucciones análogas a la diapositiva anterior, **calculando el desplazamiento en bytes a partir de i, j**
 - Ejercicio 2.3
Resultado: $A[i] = 24$
 - Ejercicio 2.4
Resultado: $A[j] = 24$

```
1 void main()
2 {
3     int h=4;
4     int i=3;
5     int j=5;
6     int A[10]={10,11,02,20,18,5,-3,6,452,0};
7
8     A[i]=h+A[i];
9 }
10
```

```
1 void main()
2 {
3     int h=4;
4     int i=3;
5     int j=5;
6     int A[10]={10,11,02,20,18,5,-3,6,452,0};
7
8     A[j] =h+A[i];
9 }
10
```