ARQUITECTURAS AVANZADAS DE PROCESADORES

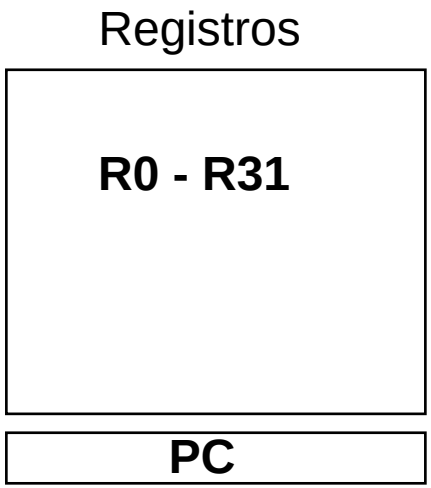
UNIDAD III. INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR PRINCIPIOS DE DISEÑO

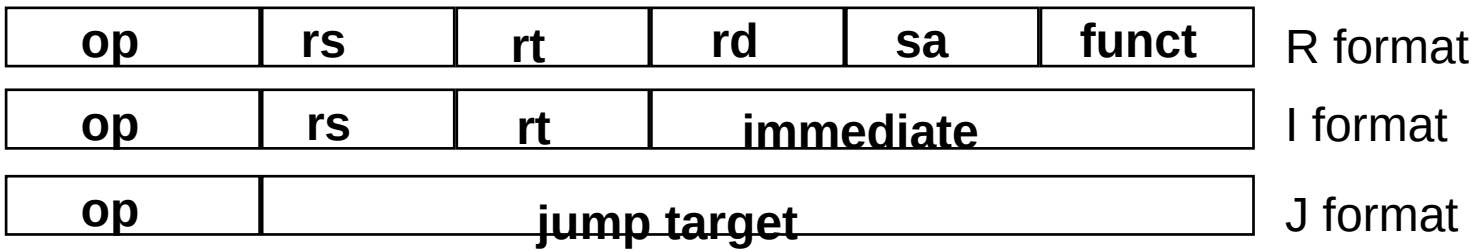
- Principio de diseño 1: *"La simplificidad favorece la regularidad"*
  - Lo demuestra la relación lenguaje alto nivel y lenguaje ensamblador
- Principio de diseño 2: *"Cuanto más pequeño, más rápido"*
  - Un número muy grande de registros puede aumentar la duración del ciclo de reloj debido a que aumenta el número de puertas para seleccionar los registros (decodificador mayor->mayor retardo de la señal de activación)
- Principio de diseño 3: *"Hacer rápido lo común"*
  - Todo lo que pueda hacer el hardware no debe ser realizado por software cuando se trate de manejar datos comunes
- Principio de diseño 4: *"El buen diseño exige buenos compromisos".*
  - Igual tamaño de longitud al elegir las instrucciones es un buen compromiso para diseñar una computador

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR ESTRUCTURA DEL PROCESADOR MIPS

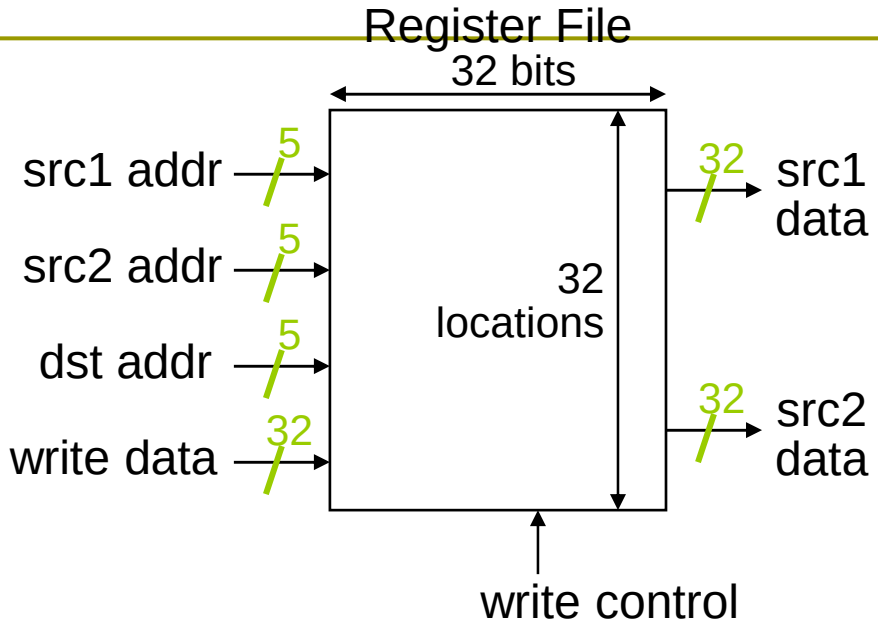
- Tipos de Instrucciones
  - Computacional (aritméticas)
  - Load/Store (carga/almacenamiento)
  - Jump and Branch
  - Punto Flotante
    - Coprocesador
  - Acceso a memoria
  - Especiales: relacionada con el Sistema Operativo
- 32 registros de un ancho de palabra de 32 bits
- Utilización de los registros frente a la memoria por ser más rápidos. Todo procesamiento de un dato antes hay que cargarlo en un registro

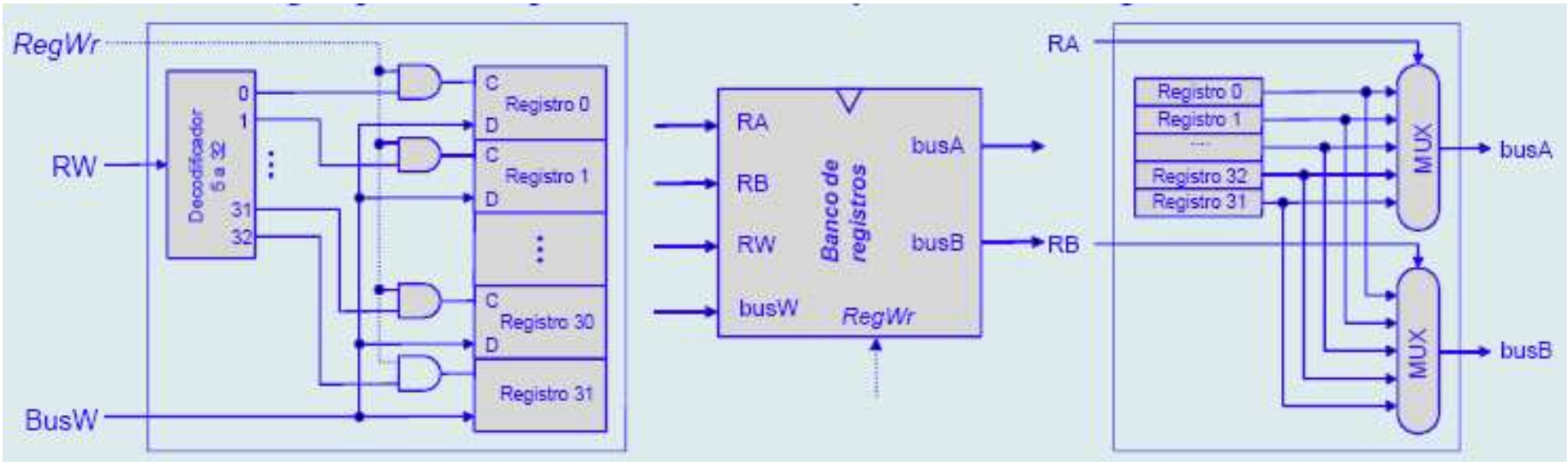


- **3 Tipos de formato de instrucciones de un ancho de palabra de 32 bits**

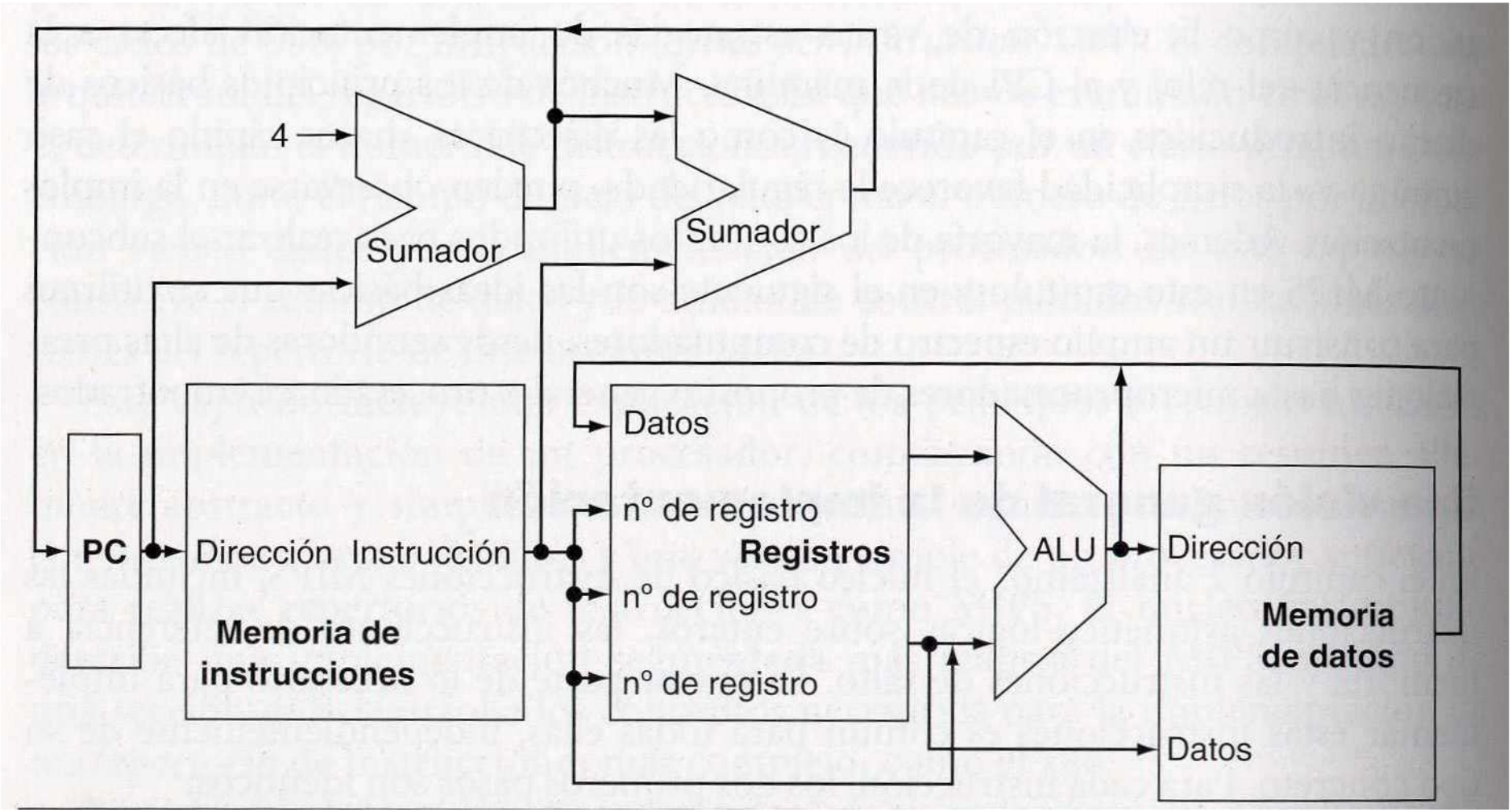


INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
ESTRUCTURA DEL PROCESADOR MIPS





INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
ESTRUCTURA DEL PROCESADOR MIPS

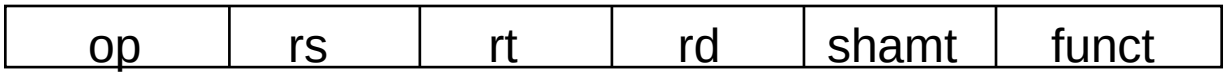


INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR ESTRUCTURA DEL PROCESADOR MIPS

Nombre	Número registro	Uso
$zero	0	Constante 0 (hardware)
$at	1	Reservado para ensamblador
$v0 - $v1	2-3	Retorno de valores en funciones
$a0 - $a3	4-7	Argumentos en funciones
$t0 - $t7	8-15	Temporales (no mantenido durante llamada)
$s0 - $s7	16-23	Temporales salvados (mantenido durante llamada)
$t8 - $t9	24-25	Temporales (no mantenido durante llamada)
$k0 - $k1	26-27	Reservado para el SO
$gp	28	Puntero de área global (global pointer)
$sp	29	Puntero de pila (stack pointer)
$fp	30	Puntero de bloque de activación (frame pointer)
$ra	31	Dirección de retorno (hardware)

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERACIONES DEL HARDWARE

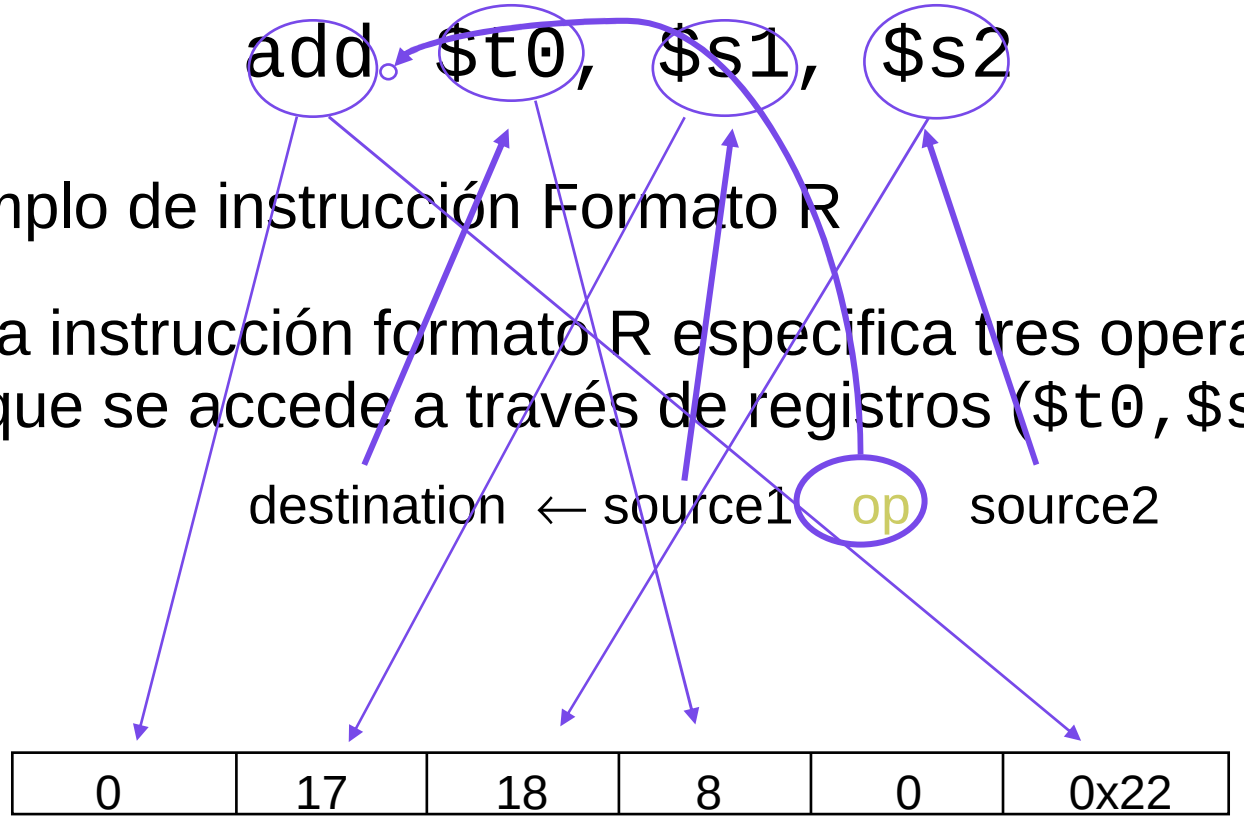
- Formato R. Nombres de los campos y su función



op	6-bits	op código t que especifica la operación
rs	5-bits	dirección del registro del primer operando source
rt	5-bits	dirección del registro del segundo operando source
rd	5-bits	dirección del registro del resultado destino
shamt	5-bits	shift amount (contador para instrucciones de desplazamiento)
funct	6-bits	function código argumento que acompaña al código op

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERACIONES DEL HARDWARE

- Operación sumar dos registros fuente y los guarda en un registro destino



- Ejemplo de instrucción Formato R
- Cada instrucción formato R especifica tres operandos a los que se accede a través de registros (`$t0,$s1,$s2`)

Lenguaje ensamblador MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Tres operandos; datos en registros
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Tres operandos; datos en registros
	add immediate	`addi $s1,$s2,100`	$s1 = $s2 + 100	Usado para sumar constantes
Transferencia de dato	load word	`lw $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Palabra de memoria a registro
	store word	`sw $s1,100($s2)`	Memory[$s2 + 100] = $s1	Palabra de registro a memória
	load half	`lh $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Media palabra de memoria a registro
	store half	`sh $s1,100($s2)`	Memory[$s2 + 100] = $s1	Media palabra de registro a memoria
	load byte	`lb $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Byte de memoria a registro
	store byte	`sb $s1,100($s2)`	Memory[$s2 + 100] = $s1	Byte de registro a memoria
	load upper immed.	`lui $s1,100`	$s1 = 100 * 2^{16}	Cargar constante en los 16 bits de mayor peso
Lógica	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Tres registros operandos; AND bit-a-bit
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Tres registros operandos; OR bit-a-bit
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 |$s3)	Tres registros operandos; NOR bit-a-bit
	and immediate	`andi $s1,$s2,100`	$s1 = $s2 & 100	AND Bit-a-bit registro con constante
	or immediate	`ori $s1,$s2,100`	$s1 = $s2 | 100	OR Bit-a-bit registro con constante
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Desplazamiento a la izquierda por constante
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Desplazamiento a la derecha por constante
Salto condicional	branch on equal	`beq $s1,$s2,L`	if ($s1 == $s2) go to L PC + 4 + 100	Comprueba igualdad y bifurca relativo al PC
	branch on not equal	`bne $s1,$s2,L`	if ($s1 != $s2) go to L PC + 4 + 100	Comprueba si no igual y bifurca relativo al PC
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compara si es menor que; usado con beq, bne
	set on less than immediate	`slt $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara si es menor que una constante
Salto incondicional	jump	`j 2500`	go to 10000	Salto a la dirección destino
	jump register	`jr $ra`	go to $ra	Para retorno de procedimiento
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	Para llamada a procedimiento

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERACIONES DEL HARDWARE

- A partir de ahora utilizaremos lenguaje C para crear un segmento de programa y a continuación veríamos su compilación a ensamblador MIPS. Utilizaremos para el ensamblador de MIPS variables genéricas que posteriormente se traducirán en operandos hardware (registros)

Lenguaje C	Ensamblador MIPS	Comentarios
a = b + c	add a, b, c	#la suma de b y c se coloca en a

a = b + c + d + e	add a, b, c
	add a, a, d
	add a, a, e
		#la suma de b y c se coloca en a
		#la suma de b, c y d se coloca en a
		#la suma de b, c, d, y e se coloca en a

a = b + c d = a – e	add a, b, c sub d, a, e	#la suma de b y c se coloca en a #la resta de a y e se coloca en d

f = (g + h) – (i + j)	add t0, g, h
	add t1, i, j
	sub f, t0, t1
		#la suma de g y h se coloca en una variable temporal t0
		#la suma de i y j se coloca en una variable temporal t1
		#la resta de t0 y t1 se coloca en f

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS DEL HARDWARE (REGISTROS)

- Veamos cómo se definen los operandos hardware (registros) en ensamblador MIPS

Lenguaje C	Ensamblador MIPS	Comentarios
a = b + c	add $s0, $s1, $s2	#la suma de b ($s1) y c ($s2) se coloca en a ($s0)

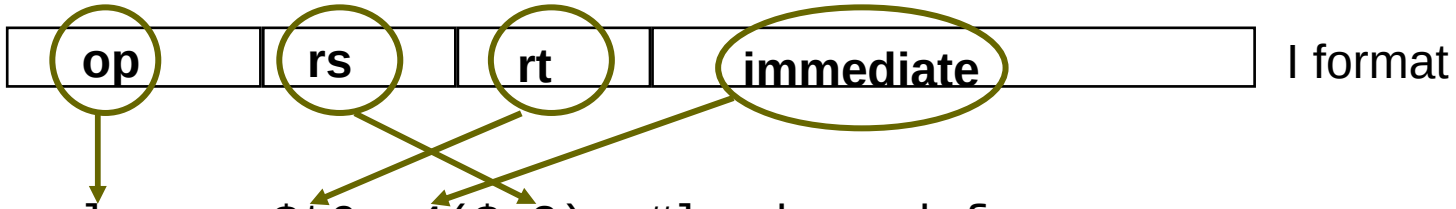
a = b + c + d + e	add $s0, $s1, $s2
	add $s0, $s0, $s3
	add $s0, $s1, $s4
		#la suma de b ($s1) y c ($s2) se coloca en a ($s0)
		#la suma de b ($s1), c ($s2) y d ($s3) se coloca en a ($s0)
		#la suma de b ($s1), c ($s2), d ($s3) y e ($s4) se coloca en a ($s0)

a = b + c d = a – e	add $s0, $s1, $s2 sub $s4, $s0, $s3	#la suma de b ($s1) y c ($s2) se coloca en a ($s0) #la resta de a ($s0) y e ($s3) se coloca en d ($s4)

f = (g + h) – (i + j)	add t0, $s1, $s2
	add t1, $s3, $s4
	sub $s0, t0, t1
		#la suma de g ($s1) y h ($s2) se coloca en una variable temporal t0
		#la suma de i ($s3) y j ($s4) se coloca en una variable temporal t1
		#la resta de t0 y t1 se coloca en f ($s0)

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS DEL HARDWARE (EN MEMORIA)

- Veamos cómo se definen los operandos hardware en memoria en ensamblador MIPS para poder acceder a grandes volúmenes de datos como tablas (arrays) o estructuas.
- MIPS tiene dos instrucciones básicas de transferencia para acceder a la memoria. Se definen en el formato I

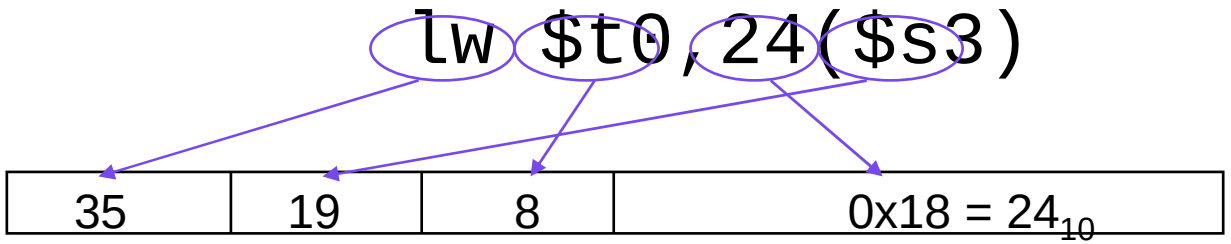



```
lw      $t0, 4($s3)  #load word from memory
sw      $t0, 8($s3)  #store word to memory
```

- El dato es cargado en (lw) o almacenado desde (sw) un registro para lo que hay que indicar la dirección de 5 bits del registro

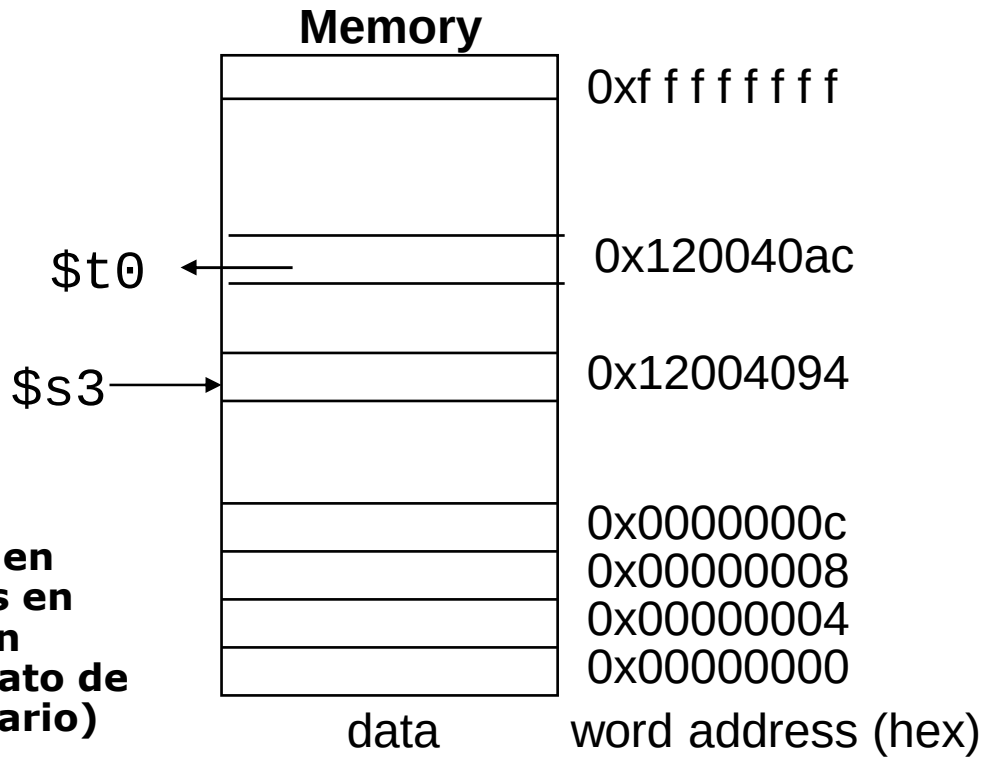
- La dirección de memoria (direcciones de 32 bits) está formada por la suma de la dirección base dado por el registro y el valor de offset
  - El campo del formato I para indicar el valor de offset es de 16 bits, lo que limita las posiciones de memoria a una región de $\pm 2^{13}$ o 8.192 words ($\pm 2^{15}$ or 32.768 bytes)

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS DEL HARDWARE (EN MEMORIA)



24_{10} + $s3 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac



- **Nota importante: no equivocarse en cómo se representan los números en ensamblador del MIPS (valores en decimal) con los valores del formato de la instrucción (hexadecimal o binario)**

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS DEL HARDWARE (EN MEMORIA)

- Ejemplos de instrucciones load y store

Lenguaje C	Ensamblador MIPS	Comentarios
g = h + A[8]	lw $t0, 32($s3) add $s1, $s2, $t0	#el registro temporal $t0 toma el valor de A[8] <=> 32($s3) #la suma de A[8] y h ($s2) se coloca en g ($s1)

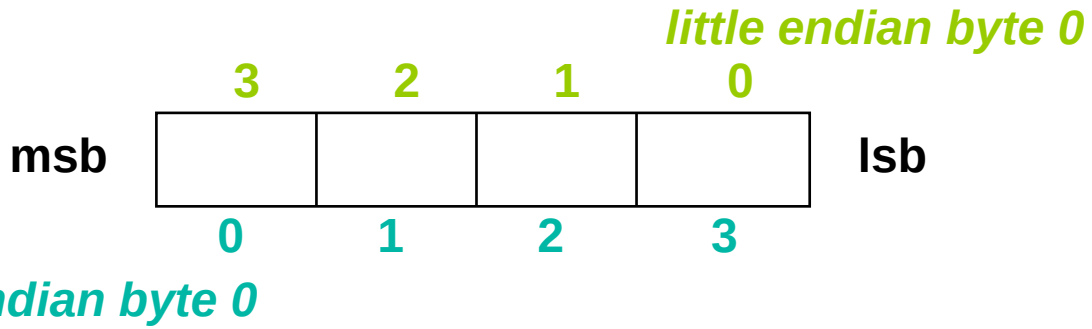
A[12] = h + A[8]	lw $t0, 32($s3)
	add $t0, $s2, $t0
	sw $t0, 48($s3)
		#el registro temporal $t0 toma el valor de A[8] <=> 32($s3)
		#la suma de A[8] y h ($s2) se coloca en $t0
		#el registro temporal $t0 se almacena en A[12] <=> 48($s3)

g = h + A[8]	lb $t0, 8($s3) add $s1, $s2, $t0	#el registro temporal $t0 toma el valor de A[8] <=> 8($s3) #la suma de A[8] y h ($s2) se coloca en g ($s1)
A[12] = h + A[8]	lb $t0, 8($s3) add $t0, $s2, $t0 sb $t0, 12($s3)	#el registro temporal $t0 toma el valor de A[8] <=> 8($s3) #la suma de A[8] y h ($s2) se coloca en $t0 #el registro temporal $t0 se almacena en A[12] <=> 12($s3)

- ¿Cómo se carga 1 byte en un registro de 32 bits?
- ¿Cómo se almacena 1 byte de un registro de 32 bits en la memoria?

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS DEL HARDWARE (EN MEMORIA)

- El uso frecuente de variable tipo bytes induce a que muchas arquitecturas realizan direccionamientos de bytes en la memoria
  - Restricciones de alineamiento – el direccionamiento de una palabra para cargar un registro en el MIPS supone cuatro direcciones de 1 byte, por lo que cuando se requiere cargar un registro con una palabra de 4 bytes, es necesario decidir la forma de cargarse esos 4 byte en el registro de 32 bit
- Big Endian: más a la izquerda es la palabra direccionada
  IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian: más a la derecha es la palabra direccionada
  Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

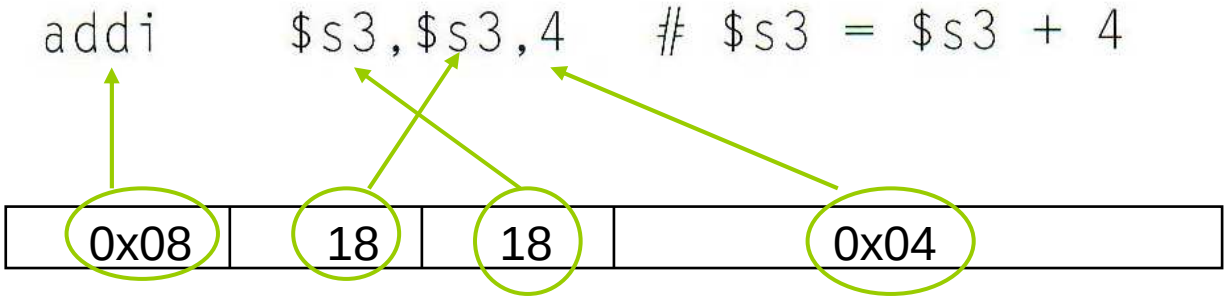


INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS CONSTANTES E INMEDIATOS

- Los operandos constantes son muy frecuentes, e incluir las constantes en las operaciones aritméticas hacen que la carga del registro se realice más rápidamente que si se accediese a memoria.

```
lw   $t0, AddrConstant4($s1)# $t0 = constante 4
add  $s3,$s3,$t0            # $s3 = $s3 + $t0 ($t0 == 4)
```

- Se diseña un tipo de suma en formato I denominada suma inmediata para cargar un valor constante en un registro



- Los valores constantes tienen un rango de $+2^{15}-1$ to -2^{15}

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR OPERANDOS CONSTANTES E INMEDIATOS

- El valor constante cero juega un papel muy importante, y definirlo en un registro específico $zero (valor cero constante fijado por hardware) simplifica el repertorio de instrucciones.

- La instrucción mover es simplemente una suma con el valor cero

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
OPERANDOS CONSTANTES E INMEDIATOS

- ¿Cómo podemos cargar una constante de 32 bits en un registro? Para ello necesitamos dos instrucciones
- Una nueva instrucción "load upper immediate"

```
lui $t0, 1010101010101010
```

16	0	8	1010101010101010_2

- Después debemos cargar los 16 bits de menor peso

```
ori $t0, $t0, 1010101010101010
```

1010101010101010	0000000000000000
0000000000000000	1010101010101010

1010101010101010	1010101010101010

Lenguaje ensamblador MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Tres operandos; datos en registros
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Tres operandos; datos en registros
	add immediate	`addi $s1,$s2,100`	$s1 = $s2 + 100	Usado para sumar constantes
Transferencia de dato	load word	`lw $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Palabra de memoria a registro
	store word	`sw $s1,100($s2)`	Memory[$s2 + 100] = $s1	Palabra de registro a memória
	load half	`lh $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Media palabra de memoria a registro
	store half	`sh $s1,100($s2)`	Memory[$s2 + 100] = $s1	Media palabra de registro a memoria
	load byte	`lb $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Byte de memoria a registro
	store byte	`sb $s1,100($s2)`	Memory[$s2 + 100] = $s1	Byte de registro a memoria
	load upper immed.	`lui $s1,100`	$s1 = 100 * 2^{16}	Cargar constante en los 16 bits de mayor peso
Lógica	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Tres registros operandos; AND bit-a-bit
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Tres registros operandos; OR bit-a-bit
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 |$s3)	Tres registros operandos; NOR bit-a-bit
	and immediate	`andi $s1,$s2,100`	$s1 = $s2 & 100	AND Bit-a-bit registro con constante
	or immediate	`ori $s1,$s2,100`	$s1 = $s2 | 100	OR Bit-a-bit registro con constante
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Desplazamiento a la izquierda por constante
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Desplazamiento a la derecha por constante
Salto condicional	branch on equal	`beq $s1,$s2,L`	if ($s1 == $s2) go to L PC + 4 + 100	Comprueba igualdad y bifurca relativo al PC
	branch on not equal	`bne $s1,$s2,L`	if ($s1 != $s2) go to L PC + 4 + 100	Comprueba si no igual y bifurca relativo al PC
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compara si es menor que; usado con beq, bne
	set on less than immediate	`slt $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara si es menor que una constante
Salto incondicional	jump	`j 2500`	go to 10000	Salto a la dirección destino
	jump register	`jr $ra`	go to $ra	Para retorno de procedimiento
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	Para llamada a procedimiento

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
OPERACIONES LÓGICAS

Operaciones lógicas	Operadores C	Operadores Java	Instrucciones MIPS
Shift left (desplazamiento a la izquierda)	<<	<<	sll
Shift right (desplazamiento a la derecha)	>>	>>>	srl
AND (bit-a-bit)	&	&	and, andi
OR (bit-a-bit)	|	|	or, ori
NOT (bit-a-bit)	~	~	nor

- Desplazamientos a izquierda y derecha. Formato R

```
sll $t2, $s0, 8    #$t2 = $s0 << 8 bits
srl $t2, $s0, 8    #$t2 = $s0 >> 8 bits
```

0		16	10	8	0x00

- Podemos realizar hasta $2^5 - 1$ desplazamientos

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
OPERACIONES LÓGICAS

- Operaciones lógicas bit a bit. Formato R:

```
and $t0, $t1, $t2 #$t0 = $t1 & $t2
or  $t0, $t1, $t2 #$t0 = $t1 | $t2
nor $t0, $t1, $t2 #$t0 = not($t1 | $t2)
```

- Instrucciones lógicas en formato I:

```
andi $t0, $t1, 0xFF00  #$t0 = $t1 & ff00
ori  $t0, $t1, 0xFF00  #$t0 = $t1 | ff00
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- Instrucciones para la toma de decisiones (salto condicionados). Ejemplo de sentencia *"if"*.
  Formato I:

```
bne $s0, $s1, Lbl #go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl #go to Lbl if $s0=$s1
```

  - Ejemplo:

```
if (i==j) h = i + j;

        bne $s0, $s1, Lbl1
        add $s3, $s0, $s1
Lbl1:   ...
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- Sentencia *"if-then-else". Para ello se introduce la instrucción de salto incondiconal. Formato J:*

```
j   label       #go to label
```

- Nos permite saltos de 2^{26} posiciones de memoria de código
  - Ejemplo:

```
        if (i==j) h = i + j; else h = i - j
        bne $s0, $s1, Else
        add $s3, $s0, $s1
        j Exit
Else:   sub $s3, $s0, $s1
Exit:
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- Las instrucciones de salto condicionadas del tipo "salto si menor que", "salto si mayor que", salto si menor o igual que" y "salto si mayor o igual que" necesitan ser construidas como **seudointrucciones** apoyándose de las siguientes instrucciones:

```
slt $t0, $s0, $s1       # if $s0 < $s1    then
                        # $t0 = 1         else
                        # $t0 = 0
```

- Otras versiones de `slt`

```
slti $t0, $s0, 25       # if $s0 < 25 then $t0=1 ...

sltu $t0, $s0, $s1      # if $s0 < $s1 then $t0=1 ...

sltiu $t0, $s0, 25      # if $s0 < 25 then $t0=1 ...
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- Construir las seudoinstrucciones de salto condicional:
  - blt ; salto si menor que
  - bgt ; salto si mayor que
  - ble ; salto si menor que o igual
  - bge ; salto si mayor que o igual
- Construir estas seudoinstrucciones como instrucciones puede hacer que el ciclo de reloj para ejecutarlas se alargara

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- Lazos:

```
while (guardar[i] == k)
    i += 1;
```

- Supongamos que:

```
i corresponde al registro $s3
k corresponde al registro $s5
guarda está en el registro $s6
```

- Código MIPS de un bucle while:

```
Loop: sll $t1, $s3, 2    #$t1 = 4 * i
      add $t1, $t1, $s6  #$t1 = dir. de guarda[i]
      lw St0, 0($t1)     #$t0 = guardar[i]
      bne $t0, $s5, Exit#ir a salida si guarda[i] ≠ k
      addi $s3, $s3, 1   #i = i + 1
      j Loop             #ir a loop (lazo)
Exit:
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- La sentencia Case/Switch:

```
switch(k){
    case 0: f = i + j; break;
    case 1: f = g + h; break;
    case 2: f = g - h; break;
    case 3: f = i - j; break;
}
```

- Supongamos que: `f a k se corresponden con los registros $s0 a $s5 respectivamente, y el registro $t2 contiene un 4, y el registro $t4 tiene la dirección de comienzo del primer Case, es decir, la dirección de case 0`

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INSTRUCCIONES PARA LA TOMA DE DECISIONES

- Código MIPS de una sentencia Case/Switch:

```
      blt $5, $zero, Fin #seudoinstrucción k < 0 ir a Fin
      bge $5, $t2, Fin   #seudoinstrucción k ≥ 4 ir a Fin

      sll $t1, $s5, 2        #registro $t1 = 4 * k
      add $t1, $t1, $t4      #$t1 = dirección del case
      lw $t0, 0($t1) #$t0 = dirección de salto al case
      jr $t0                 #salto al case del valor de k
L0:   add $s0, $s3, $s4
      j Fin
L1:   add $s0, $s1, $s2
      j Fin
L2:   sub $s0, $s1, $s2
      j Fin
L3:   sub $s0, $s3, $s4
Fin:
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
INICIALIZACIÓN DE REGISTROS

- Inicialización de registros:

```
a = 0;
for (i=0; i<5; ++i)
{
a = a+i;
}
```

```
#Inicialización de registros
move $s0, $zero # a = 0
# Operación
move $s1, $zero # i = 0
loop: add $s0, $s0, $s1
addi $s1, $s1, 1
slti $t0, $s1, 5
bne $t0, $zero, loop
```

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR FORMATO DE DATO. NÚMEROS CON SIGNO Y SIN SIGNO

- La palabra MIPS tiene una longitud de 32 bits de manera que podemos representar los números del 0 al 2^{32}-1.

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dos} = 0_{diez}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{dos} = 1_{diez}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dos} = 2_{diez}$$
$$\ldots \qquad\qquad \ldots$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dos} = 4\ 294\ 967\ 293_{diez}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dos} = 4\ 294\ 967\ 294_{diez}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dos} = 4\ 294\ 967\ 295_{diez}$$

- El resultado de las operaciones que no pueden ser representados con el formato de datos del microprocesador se denomina ***desbordamiento***. Es responsabilidad del programador y del sistema operativo qué hacer si se produce desbordamiento.

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR FORMATO DE DATO. NÚMEROS CON SIGNO Y SIN SIGNO

- Representación de números negativos. La representación de números negativos enteros (punto fijo) en los computadores es la de complemento a dos. Se desecha la representación en Signo_Magnitud por diversos motivos

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dos} = 0_{diez}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{dos} = 1_{diez}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dos} = 2_{diez}$
...
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dos} = 2\ 147\ 483\ 645_{diez}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dos} = 2\ 147\ 483\ 646_{diez}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dos} = 2\ 147\ 483\ 647_{diez}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dos} = -2\ 147\ 483\ 648_{diez}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{dos} = -2\ 147\ 483\ 647_{diez}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dos} = -2\ 147\ 483\ 646_{diez}$
...
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dos} = -3_{diez}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dos} = -2_{diez}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dos} = -1_{diez}$

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR FORMATO DE DATO. NÚMEROS CON SIGNO Y SIN SIGNO

- El desbordamiento en las operaciones con números con signo viene determinado por el signo incorrecto del resultado, siendo1 si el resultado es positivo, y 0 si el resultado es negativo.
- Definición de extensión de signo. Es fácil obtener un número 2n bits a partir de un número de n bits sin que su valor se modifique.
- Permite la notación sesgada a la hora de representar números en punto flotante en relación con el exponente.

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR REPRESENTACIÓN DE INSTRUCCIONES EN EL COMPUTADOR

- Para obtener el lenguaje máquina a partir del lenguaje ensamblador, hay que sustituir cada uno de los campos (depende de cada tipo de formato R, I, J) del formato de instrucción a su conversión binaria
- También se puede realizar el proceso contrario. Obtener el código ensamblador a partir del código máquina
- Ejemplo:

A[300] = h + A[300]	lw $t0, 1200($t1)
	add $t0, $s2, $t0
	sw $t0, 1200($t1)

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
REPRESENTACIÓN DE INSTRUCCIONES EN EL COMPUTADOR

- Ejemplo:

op	rs	rt	dirección		
			rd	shamt	funct
35	9	8	1200		
100011	01001	01000	0000 0100 1011 0000		
0	18	8	8	0	32
000000	10010	01000	01000	0000	100000
43	9	8	1200		
101011	01001	01000	0000 0100 1011 0000		

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR PROCEDIMIENTOS

- En la ejecución de un procedimiento, el programa debe seguir los siguientes seis pasos:
  1. Colocar los parámetros en un lugar donde el procedimiento pueda acceder a ellos
  2. Transferir el control al procedimiento
  3. Adquirir los recursos del almacenamiento necesarios para el procedimiento
  4. Realizar la tarea deseada
  5. Colocar el valor del resultado en un lugar donde el programa que lo ha llamado pueda tener acceso
  6. Retornar el control al punto del origen, puesto que un procedimiento puede ser llamado desde varios puntos de un programa
- Disponibilidad de registros para la llamada de procedimientos:
  - $a0 - $a3 : cuatro registros de argumentos para pasar parámetros
  - $v0 - $v1 : dos registros de valores para retornar valores
  - $ra : un registro con la dirección de retorno para volver al punto de origen

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR PROCEDIMIENTOS

- Instrucción saltar y enlazar (jump-and-link) se utiliza para invocar el procedimiento
  - jal ProcedureAddress | salta a una dirección y simultáneamente guarda la dirección de la instrucción siguiente (PC+4) en el registro $ra (dirección de retorno)
- Una vez que va a finalizar el procedimiento se utiliza la instrucción jr $ra para retornar a la dirección de retorno almacenada en $ra
- Si necesitamos utilizar más registros de los 4 de argumento y dos de retorno de valores, se utiliza la estructura de acceso a la memoria pila mediante instrucciones push (poner datos en la pila) y pop (sacar datos de la pila)

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR PROCEDIMIENTOS

- Ejemplo de procedimiento: suponer que g, h, i, j se corresponden con los registros $a0, $a1, $a2, $a3 respectivamente, y f se corresponde con el $s0

```
int ejemplo_hoja (int g, int h, int i, int j)
        {
                int f;
                f = (g + h) – (i + j);
                return f;
        }
```

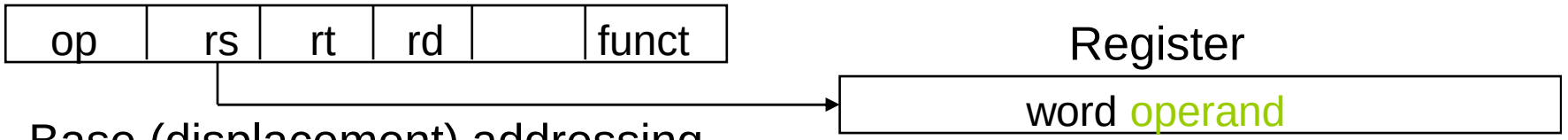
- Código MIPS:

```
ejemplo_hoja:   addi $sp, $sp, -12 #se guarda en pila tres
                sw $t1, 8($sp)     #registros que van a ser
                sw $t0, 4($sp)     #utilizados
                sw $s0, 0($sp)

                add $t0, $a0, $a1
                add $t1, $a2, $a3
                sub $s0, $t0, $t1  #operación f = (g + h) – (i + j)

                add $v0, $s0, $zero #retorno de f

                lw $s0, 0($sp)     #retorno de valores de pila
                lw $t0, 4($sp)     #guardados
                lw $t1, 8($sp)     #ajuste de la pila para
                addi $sp, $sp, 12  #eliminar tres posiciones

                jr $ra             #salto a dirección de retorno
```

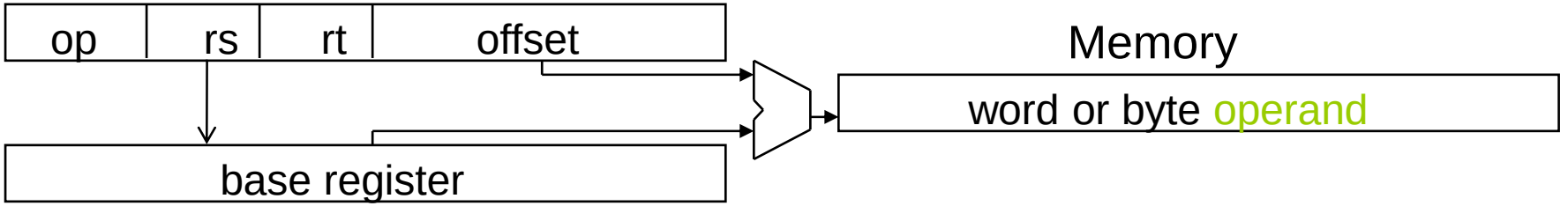
- Los valores de $t0 y $t1 no tienen por qué ser guardados su valores frente a la llamada de procedimiento
- El problema de los procedimientos anidados (ejemplo: factorial)

INSTRUCCIONES: EL LENGUAJE DEL COMPUTADOR
MODOS DE DIRECCIONAMIENTO DEL MIPS

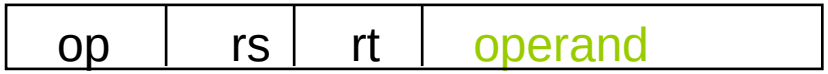
1. Register addressing



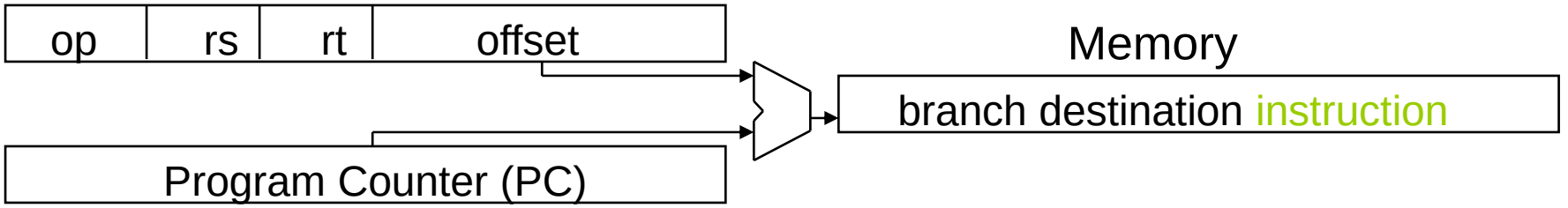
2. Base (displacement) addressing



3. Immediate addressing



4. PC-relative addressing



5. Pseudo-direct addressing