

# Estructura de Computadores

Grado de Ingeniería Informática  
ETSINF

## Tema I: El procesador

DISCA

# Objetivos

- Contextualizar la asignatura
  - ✓ Conocer el contexto de la arquitectura MIPS32
  - ✓ Conocer los aspectos más generales de la arquitectura MIPS32
  - ✓ Aprender el ciclo de ejecución del procesador
- Conocer el diseño de la ruta de datos basada en multiplexores
- Conocer el diseño de la unidad de control del procesador

Introdutorio (Tema 0)

# Contenido y Bibliografía

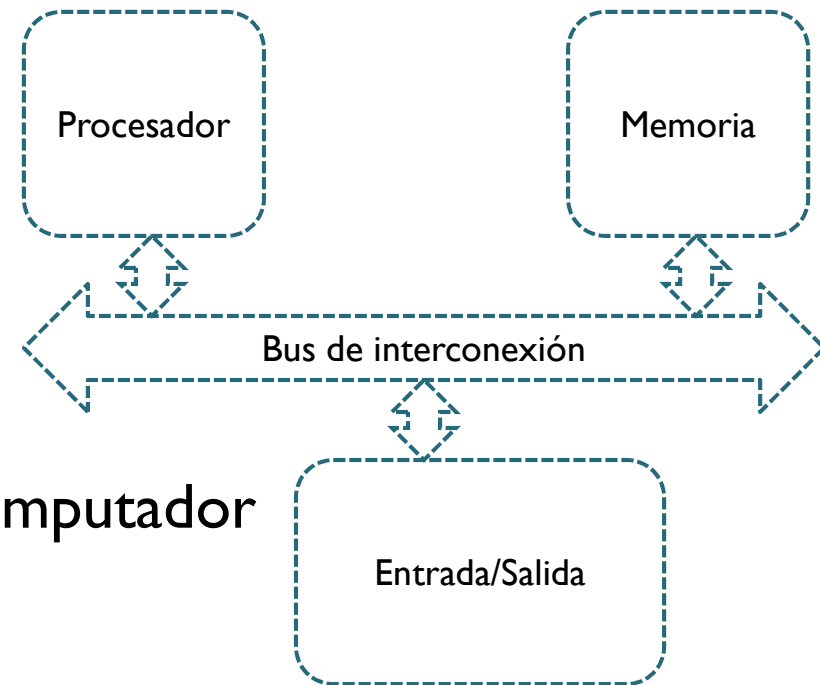
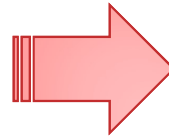
- Introducción
- 1 – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

Introductorio  
(Tema 0)

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

# Introducción



- Componentes básicos de un computador

- ✓ Procesador (UCP)
- ✓ Memoria
- ✓ Entrada/Salida
- ✓ Interconexión

Concepto de programa almacenado:

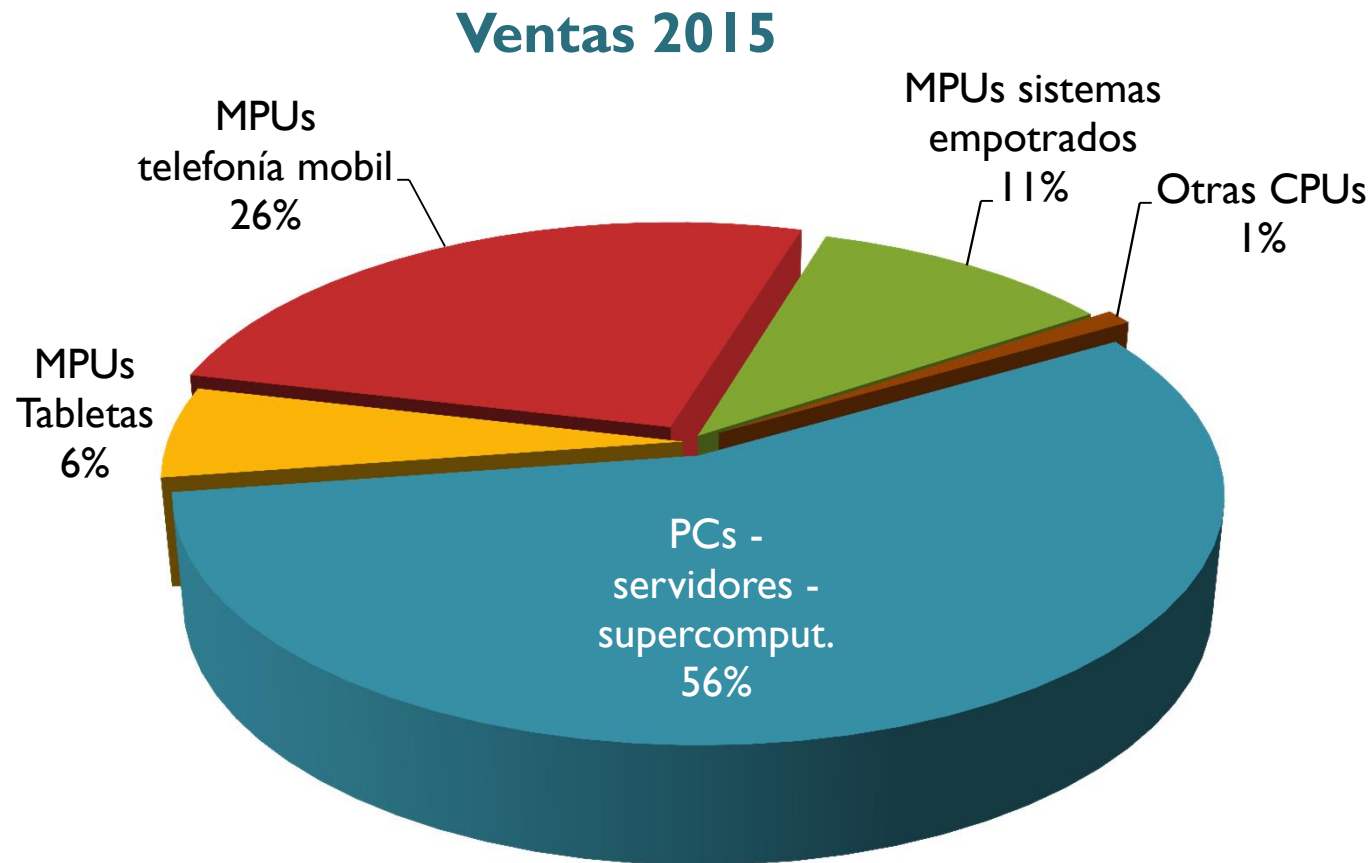
- Las instrucciones se representan como números
- Los programas son almacenados en memoria para ser leídos o escritos también como números
- Se interactúa con el mundo exterior mediante dispositivos de Entrada/Salida

# Introducción: Tipos de computadores

Tipo	Potencia	Ancho Palabra (bits)	Consumo	Aplicaciones
<b>Supercomputadores</b>	Muy alta potencia de cálculo.	64, 128	Extremadamente alto	Gestión en Corporaciones, Gobiernos. Aplicaciones computación masiva: centros de investigación.
<b>Estaciones de trabajo</b>	Alta	32, 64	Alto	Diseño, simulación, control: Empresas, Universidades
<b>Portátiles</b>	Alta-Media	32, 64	Medio Bajo	Informática en general Computadores personales
<b>Tabletas</b>	Media	32,64	Muy bajo	Informática personal
<b>Sistemas empotrados</b>	Media-Alta.	16, 32, 64	Medio Bajo	Sistemas empotrados para: Comunicaciones (routers, switches). Electrodomésticos, automoción, periféricos, sistemas industriales,...
<b>Teléfonos</b>	Media-Baja	16, 32	Extremadamente bajo	Teléfonos móviles (Smartphones)



# Introducción: El mercado de los procesadores



MPU: Microprocessor Unit

Todo procesador se define de acuerdo con una **ARQUITECTURA**

# Introducción: Arquitectura

- **Arquitectura (ISA: Instruction Set Architecture)**
  - Hace referencia al repertorio de instrucciones, registros, modelo de excepciones, manejo de la memoria virtual, mapa de direcciones físicas y otras características comunes de un procesador
  - Todo aquello que el programador debe saber acerca de la máquina
- **Ejemplos de arquitecturas:**

Arquitectura	Empresas
x86, x86-64, IA-32, INTEL®64	INTEL,AMD,..
MIPS-32, MIPS-64	Mips Technologies
DEC Alpha Architecture	Digital Equipment Corp.
Power , PowerPC	Apple, IBM, Motorola
ARM Architecture	Advanced RISC Machines Holdings)

- Las arquitecturas pueden evolucionar, dando lugar a versiones de 16, 32 o 64 bits, siempre compatibles entre sí.

# Introducción: Implementación

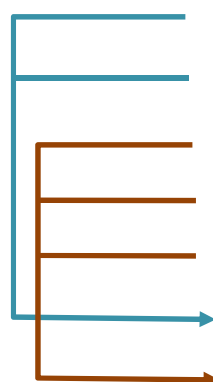
- **Implementación:** Hace referencia a las características concretas de los circuitos que conforman el diseño de un procesador concreto que ejecuta una arquitectura.
  - También se denominan **Microarquitecturas**

Arquitectura	Microarquitecturas	Modelos	Fabricantes
MIPS	MIPS32 ;MIPS64, Warrior, Aptiv	R2000, R3000, R4000,... P5600, P6600, MicroAptiv, InterAptiv, ProAptiv	MIPS Tech. Imagination Tech.
IA-32, Intel 64	P5, P6	Pentium II, III, Pro	INTEL
	NetBurst	Pentium 4, Pentium D, Celeron D	
	Pentium M	Core Duo, Core Solo, Pentium M, Celeron M	
	Intel Core	Core 2 Duo, Quad, Extreme, Xeon 5xxx, 7xxx, Celeron dual-core	
	Atom	Atom	
	Core i	i3, i5, i7	
X86, x86-64	K5,K6,K8	Athlon 64, Phenom, Turion, Sempron, Opteron	AMD
ALPHA	EV4,EV5,..EV9	21x64, 22x64, 23x64 (x = 0,1,2,3,4)	DEC



# Arquitectura MIPS

- MIPS: **M**icroprocerssor without **I**nterlocked **P**ipeline **S**tages)
  - ✓ Procesador RISC (**R**educed **I**ntruction **S**et) desarrollado en la Universidad de Stanford por John L. Hennessy que emplea la técnica de segmentación .
  - ✓ Comercializado por MIPS Technologies (Silicon Graphics International)
  - ✓ Arquitectura sencilla → Muy utilizada por las universidades para docencia
  - ✓ El diseño segmentado del MIPS es el precursor de la mayoría de los procesadores RISC posteriores



Versión juego de instrucciones (ISA)	Ancho palabra	Procesadores
MIPS I	32	R2000, R3000
MIPS II	32	R6000
MIPS III	64	R4000
MIPS IV	64	R5000, R10000
MIPS V	64	-
MIPS32	32	4K
MIPS64	64	5K

# Implementaciones del MIPS

- Los diferentes modelos del MIPS, R2000, R3000,... R10000 fueron utilizados como CPUs de computadores estándar por SGI, Olivetti, Siemens, etc.. No tuvo demasiado éxito, en un mercado dominado por Intel y AMD, excepto por el procesador Loongson (China)
- En la década de los 90 MIPS Technologies licenció el diseño del MIPS en forma de dos arquitecturas: MIPS32 y MIPS64. Estas fueron ampliamente usadas en microcomputadores y sistemas empuotrados diseñados y fabricados por terceras marcas.
- En 2012, Imagination Tech. adquirió MIPS Tech. Desde entonces los diseños MIPS se han distribuido ampliamente en forma de IP-cores (Warrior, Aptiv), siendo un 'peso-pesado' en el mercado de los procesadores para sistemas empuotrados y SoC: Cisco, Sony, NEC, Microchip, Toshiba,....

IP-core (Intellectual Property core): Bloque lógico o circuito que es propiedad intelectual de una compañía pero que puede ser licenciado a otras compañías que lo pueden usar en FPGAs o diseños ASIC (Applicacion Specific Integrated Circuit)

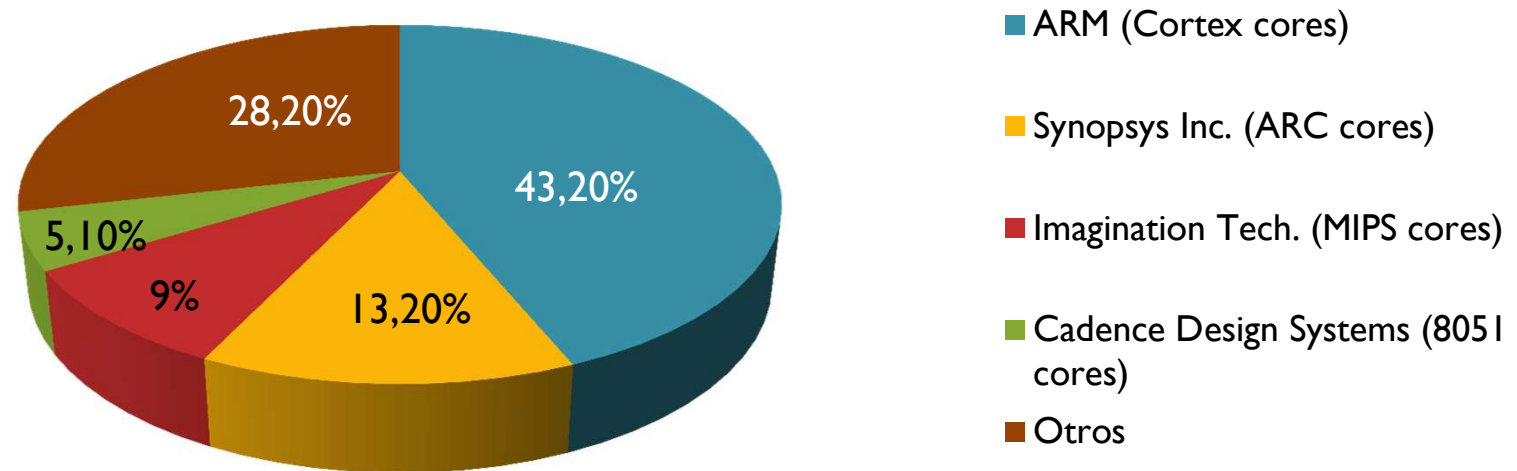
Embedded System: Sistema computador embebido o empuotrado en una máquina y diseñado para realizar unas funciones específicas, usualmente de tiempo real.

SoC (System On Chip): Circuito integrado que integra todos los componentes de un computador o sistema electrónico en un solo chip.

# Implementaciones del MIPS

- Los diferentes modelos (cores) del MIPS se pueden ver en:
  - [https://en.wikipedia.org/wiki/List\\_of\\_MIPS\\_microarchitectures](https://en.wikipedia.org/wiki/List_of_MIPS_microarchitectures)
  - <https://imgtec.com/mips/architectures>
- El mercado de los IP-cores en 2013 se distribuía como indica la figura:

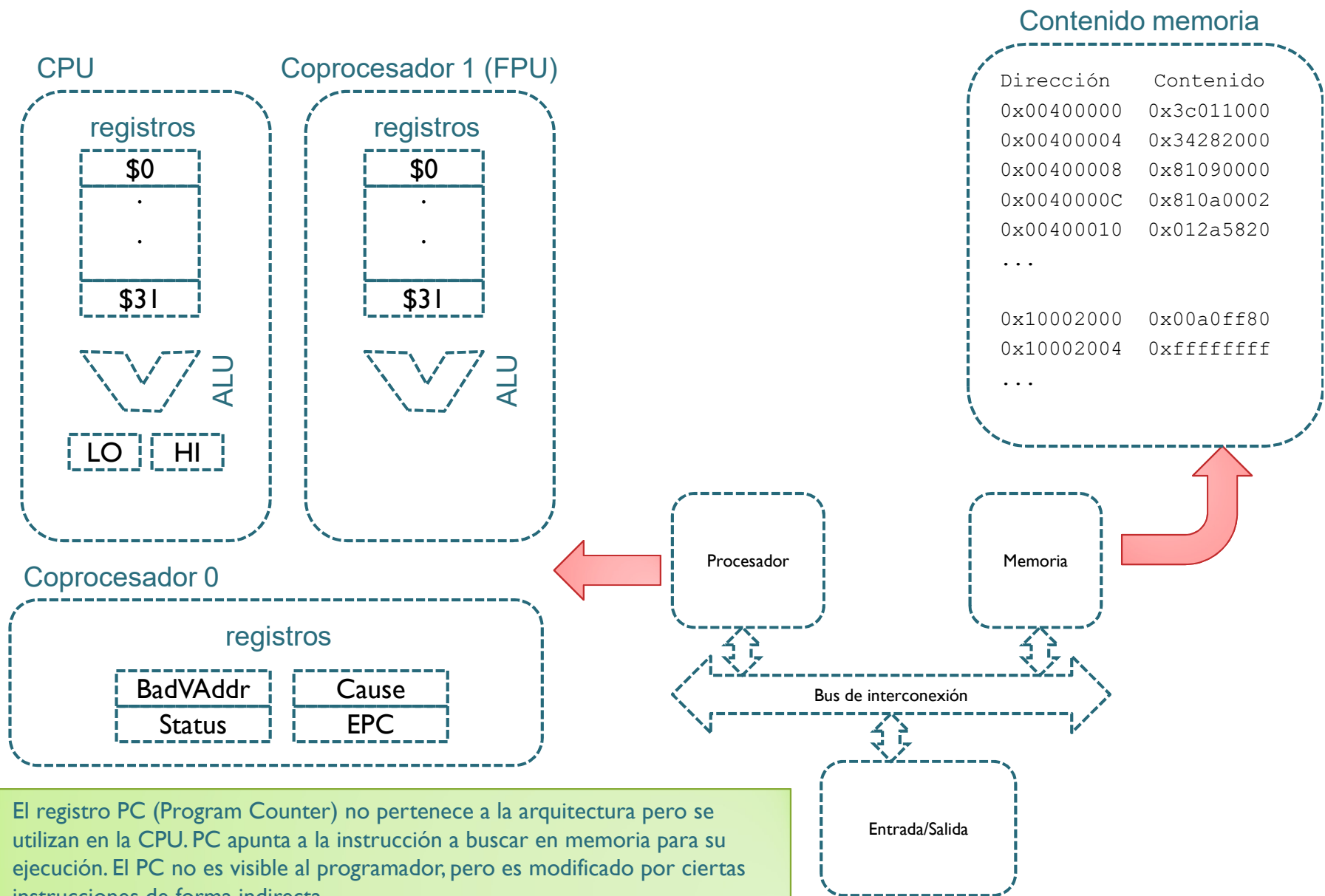
IP-cores market share 2013



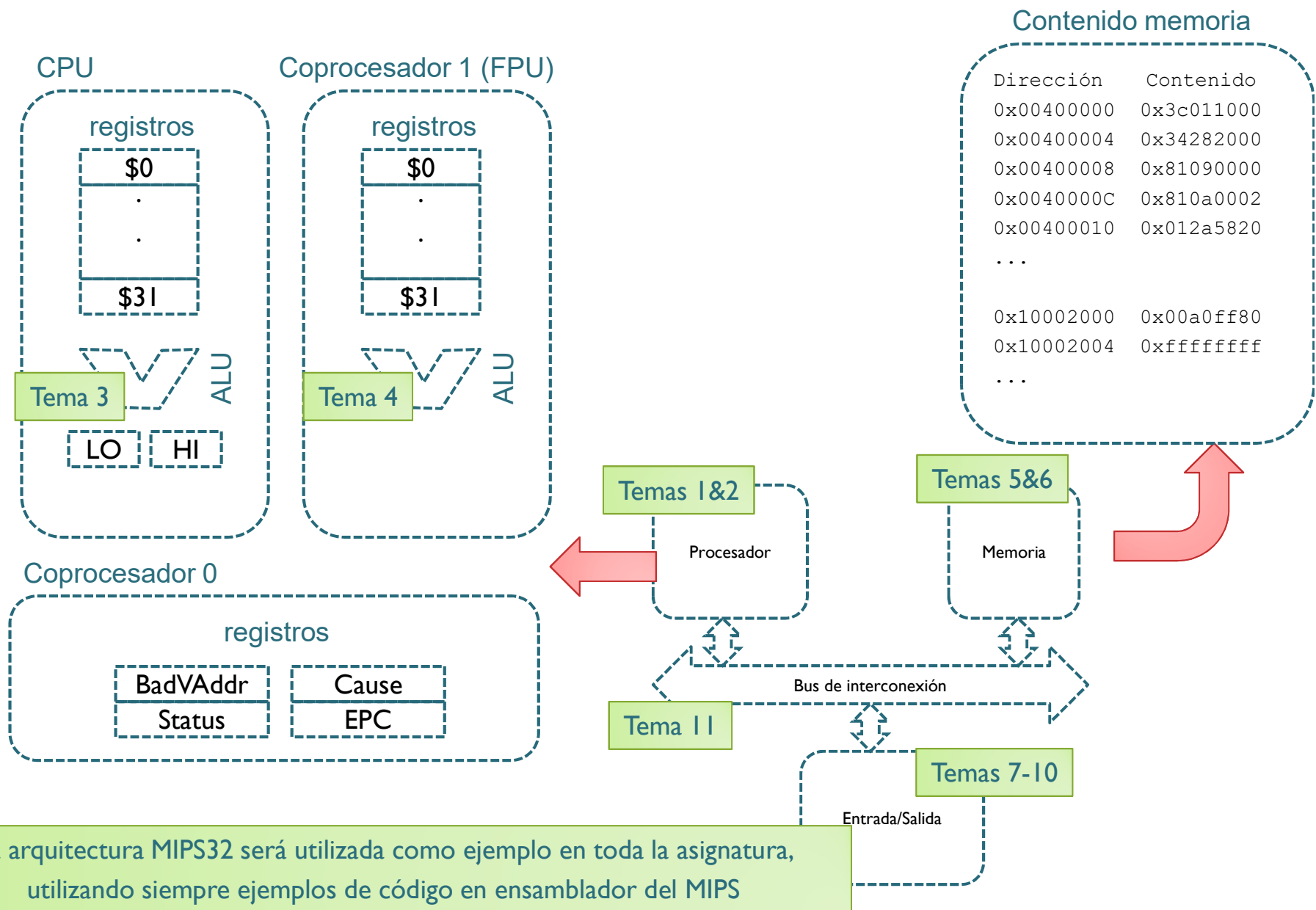
# Arquitectura MIPS32: Características Básicas

- Máquina RISC (Reduced Instruction Set Computer)
- Ancho de palabra y tamaño de buses de 32 bits
- Principales tamaño de datos en las instrucciones:
  - ✓ Byte (B), halfword (H), word (W)
- Arquitectura de carga/almacenamiento
  - ✓ Instrucciones específicas de lectura (carga) y escritura (almacenamiento) en memoria
  - ✓ Todos los operandos de una instrucción aritmética se cargan inicialmente en registros, o están en la propia instrucción (constantes)
  - ✓ Instrucciones aritméticas con 3 operandos de 32 bits en registros
- Modos de funcionamiento: usuario, núcleo (kernel) y depuración

# Arquitectura MIPS32: Características Básicas



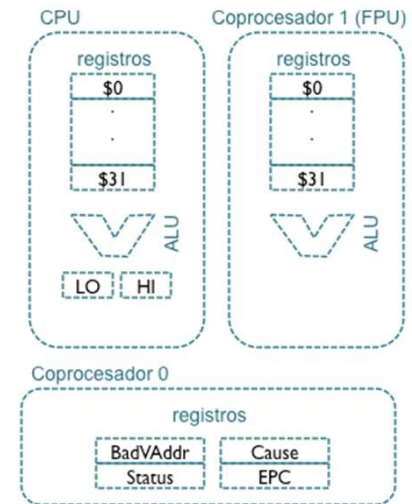
# Arquitectura MIPS32 en la asignatura





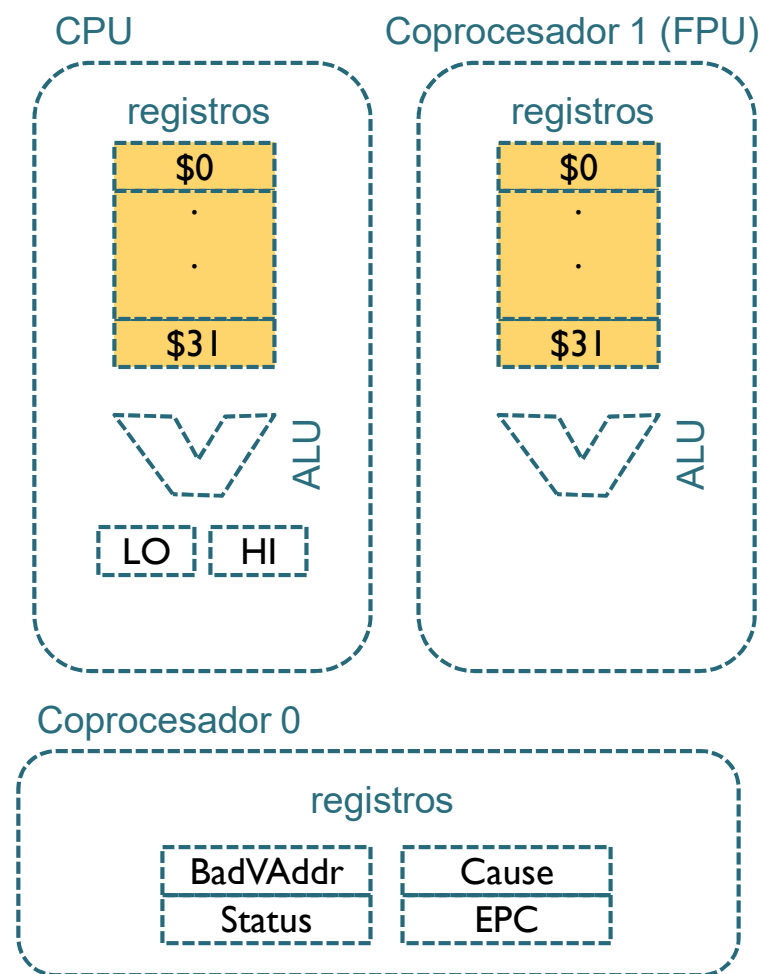
# Arquitectura MIPS32: Modelo de programación

- CPU
  - ✓ 32 registros de 32 bits (\$0..\$31)
  - ✓ Registros HI-LO
- Coprocesador 0
  - ✓ Control del sistema (jerarquía, excepciones, modos de ejecución, ...)
  - ✓ 4 registros específicos (hay más)
- FPU (Coprocesador 1, opcional)
  - ✓ 32 registros de 32 bits (\$0..\$31)
  - ✓ Aritmética coma flotante de simple precisión (32 registros)
  - ✓ Aritmética coma flotante de doble precisión (16 parejas de regs)

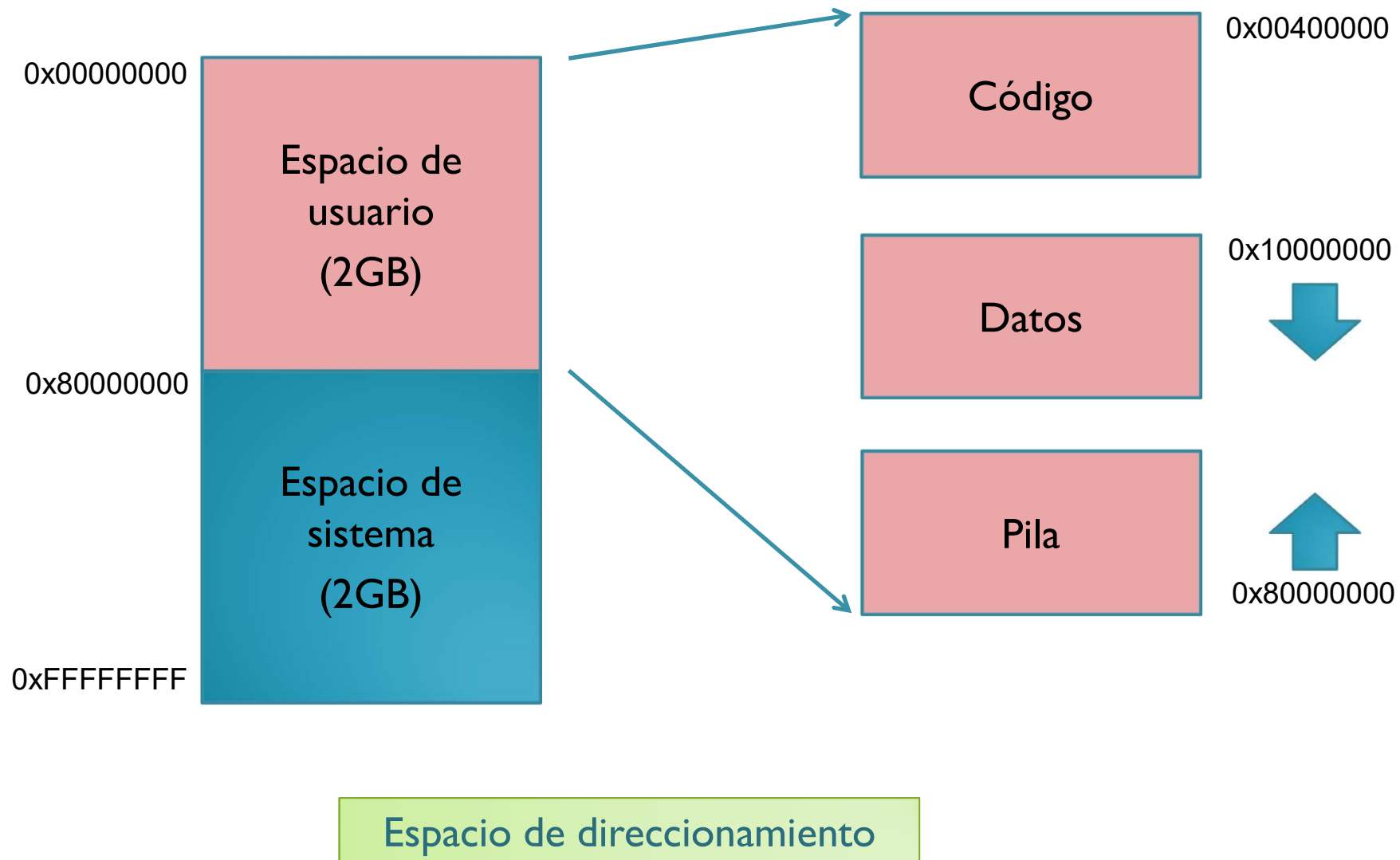


# Arquitectura MIPS32: Registros

Nombre	Registro	Uso
\$zero	\$0	Constante 0
\$at	\$1	Registro temporal del ensamblador
\$v0-\$v1	\$2-\$3	Retorno de funciones
\$a0-\$a3	\$4-\$7	Argumentos de funciones
\$t0-\$t7	\$8-\$15	Registros temporales
\$s0-\$s7	\$16-\$23	Temporales salvados
\$t8-\$t9	\$24-\$25	Registros temporales
\$k0-\$k1	\$26-\$27	Utilizados por el SO
\$gp	\$28	Puntero global
\$sp	\$29	Puntero de pila
\$fp	\$30	Puntero de marco (frame)
\$ra	\$31	Dirección de retorno
\$f0-\$f31	\$0..\$31	Registros de coma flotante



# Arquitectura MIPS32: Memoria



# Arquitectura MIPS32: Memoria

- Directivas de ensamblador

- ✓ Directivas de segmentos de memoria

- `.data [dirección]`
- `.text [dirección]`
- `.end`

- ✓ Directivas de reserva de memoria de datos

- `.space n`
- `.byte b1 [, b2] ...`
- `.half b1 [, b2] ...`
- `.word b1 [, b2] ...`
- `.ascii cadena1 [, cadena2] ...`
- `.asciiz cadena1 [, cadena2]...`

Las variables del programa son codificadas como etiquetas de datos

`A = 0x10000000`

`W = 0x10000004`

...

```

                                .data 0x10000000
A:                            .byte 2, 3, 4
W:                            .word 33
V:                            .space 100

                                .data 0x10004000
C1:                          .ascii "hola"
C2:                          .asciiz "hola"

                                .text 0x00400000
...
                                .end
  
```

# Arquitectura MIPS32: Memoria

- **Direccionamiento**
  - ✓ Direccionamiento a nivel de byte
  - ✓ Modos big-endian y little-endian soportados
- **Alineamiento**
  - ✓ Byte en cualquier dirección
  - ✓ Media palabra (half) en dirección par
  - ✓ Palabra (word) en dirección múltiplo de 4
  - ✓ Doble palabra (dword) en dirección múltiplo de 8
- **Directiva**
  - ✓ `.align N`
  - ✓ Alinea a partir de la siguiente instrucción múltiplo de  $2N$

# Arquitectura MIPS32: Instrucciones

- Grupos de instrucciones

- ✓ Transferencia entre registros
- ✓ Carga/almacenamiento
- ✓ Aritméticas
- ✓ Lógicas
- ✓ Comparación
- ✓ Salto
- ✓ Coma flotante
- ✓ Otras

Las instrucciones serán ejercitadas  
en las sesiones de laboratorio.  
El apéndice contiene ejemplos de uso.

## Instrucciones vs pseudo-instrucciones

Las instrucciones son soportadas por el hardware  
mientras que las pseudo-instrucciones son traducidas por  
el ensamblador en un conjunto de instrucciones equivalente

### EJEMPLO:

```
la $t1, 0x10002000 -> lui $1, 0x1000  
ori $t1, $1, 0x2000
```

- Sintaxis y codificación

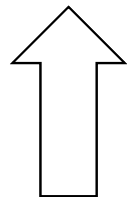
- ✓ <http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>
- ✓ [http://en.wikipedia.org/wiki/MIPS\\_architecture](http://en.wikipedia.org/wiki/MIPS_architecture)



# Arquitectura MIPS32: ejemplo de ejecución

```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
b: .word -1
```

```
.text 0x00400000
lui $1, 4096
ori $8, $1, 8192
lb $9, 0($8)
lb $10, 2($8)
add $11, $9, $10
sb $11, 3($8)
add $9, $0, $0
```



## Ensamblado:

- Pseudoinstrucciones
- Nombres de registros
- Cálculos de valores

Pseudo  
instrucción

```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1

.text 0x00400000
la $t0, a
lb $t1, 0($t0)
lb $t2, 2($t0)
add $t3, $t1, $t2
sb $t3, 3($t0)
add $t1, $0, $0
.end
```

Alias en nombre de  
los registros

## Contenido memoria

Dirección	Contenido
0x00400000	0x3c011000
0x00400004	0x34282000
0x00400008	0x81090000
0x0040000C	0x810a0002
0x00400010	0x012a5820
0x00400014	0xa10b0003
0x00400018	0x00004820
...	...
0x10002000	0x00a0...
0x10002004	0xffff...
...	...

0011 1100 0000 0001 0001 0000 0000 0000

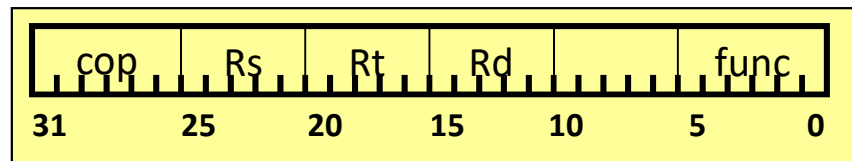
0000 0001 0010 1010 0101 1000 0010 0000

Instrucción con formato R (add \$11, \$9, \$10)  
 C.op: bits 31-26: 000000  
 Rs: bits 25-21: 01001 -> \$9  
 Rt: bits 20-16: 01010 -> \$10  
 Rd: bits 15:11: 01011 -> \$11  
 Func: bits 5-0: 100000 -> add

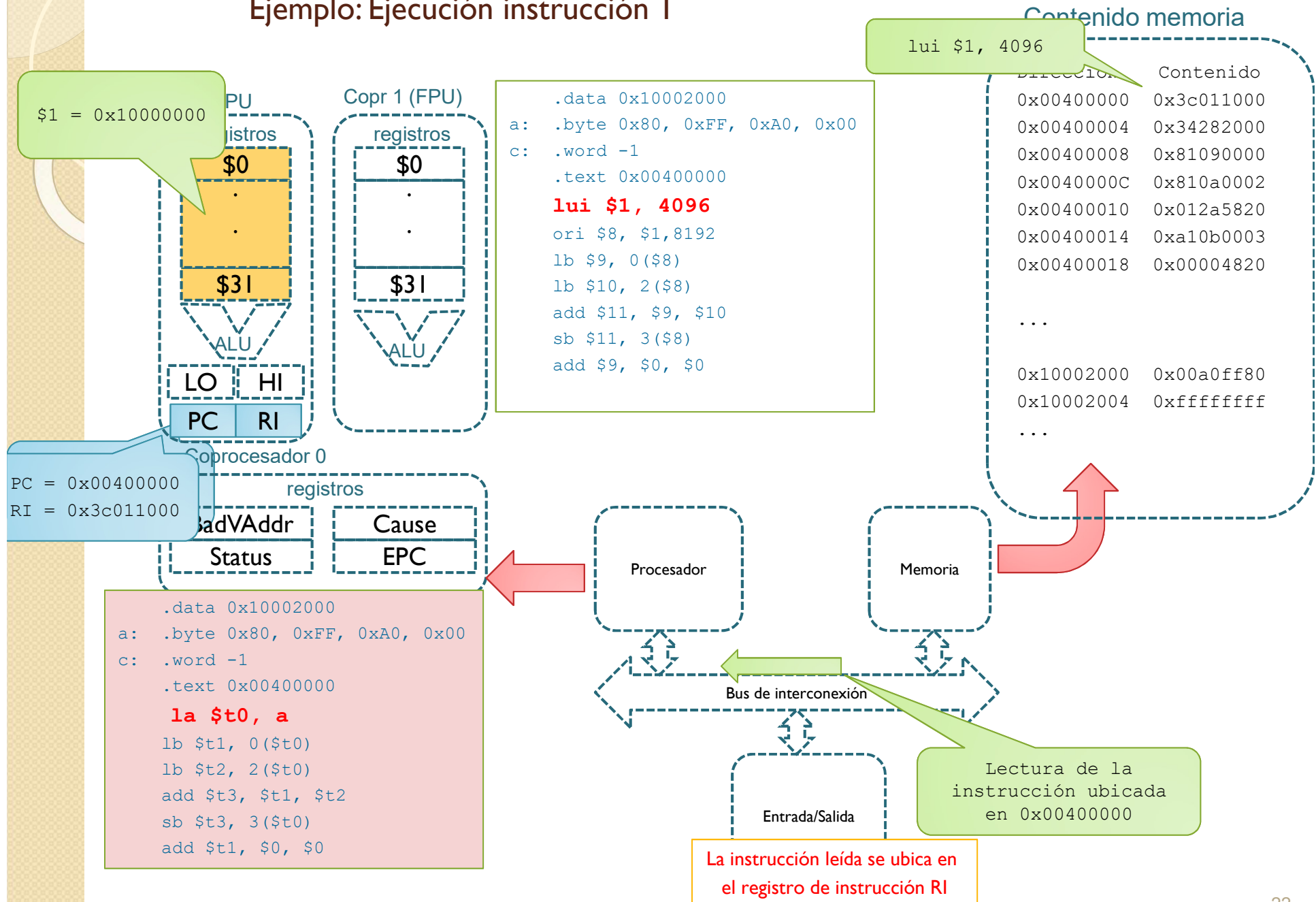
Obtenido con PCSpim

```
[00400000] 3c011000 lui $1, 4096 [a] ; 6: la $t0, a
[00400004] 34282000 ori $8, $1, 8192 [a]
[00400008] 81090000 lb $9, 0($8) ; 7: lb $t1, 0($t0)
[0040000c] 810a0002 lb $10, 2($8) ; 8: lb $t2, 2($t0)
[00400010] 012a5820 add $11, $9, $10 ; 9: add $t3, $t1, $t2
[00400014] a10b0003 sb $11, 3($8) ; 10: sb $t3, 3($t0)
[00400018] 00004820 add $9,$0,$0 ; 189: add $t1, $0,$0
```

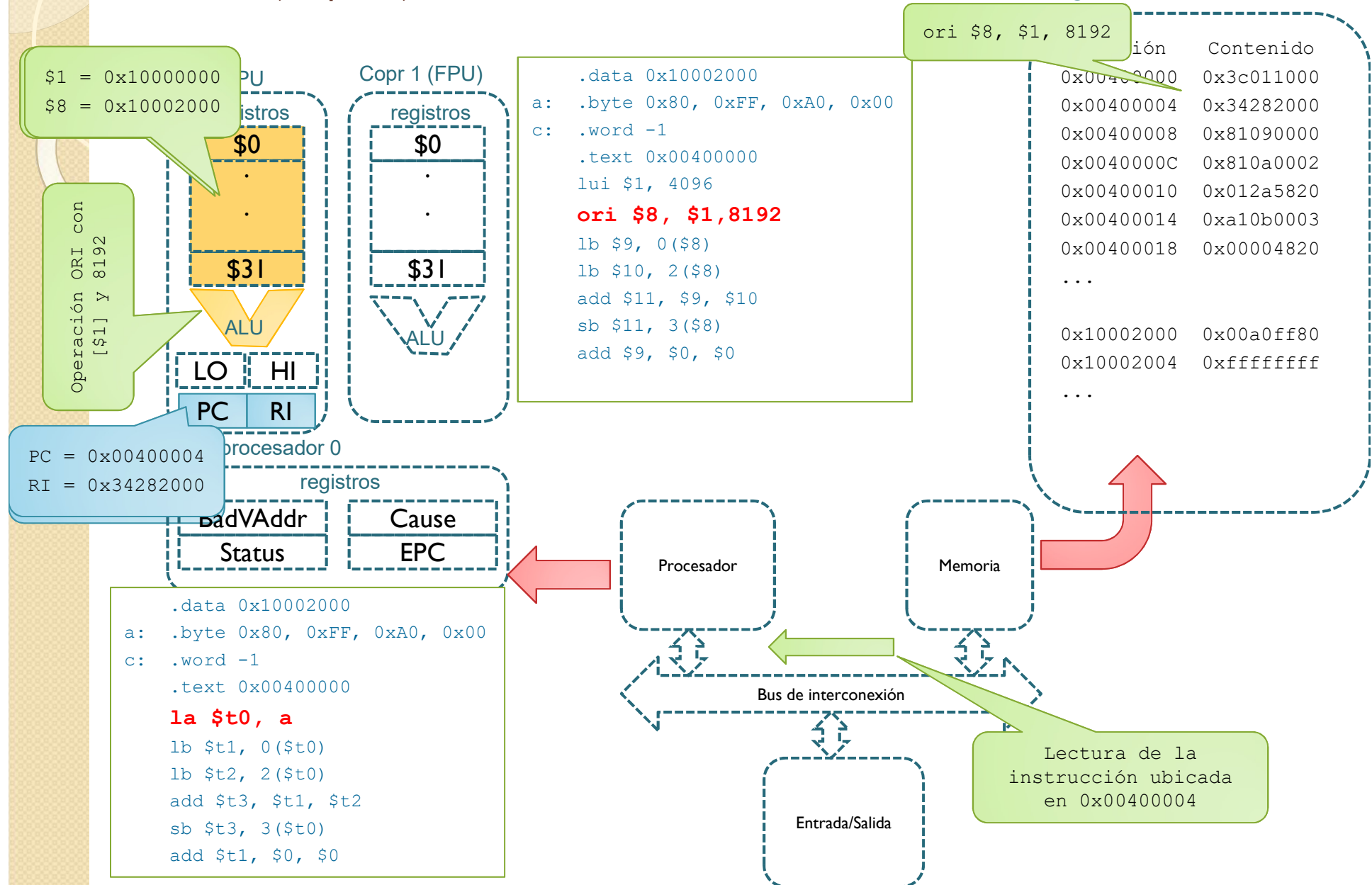
Codif.  
Formato R



## Ejemplo: Ejecución instrucción I



## Ejemplo: Ejecución instrucción 2



## Ejemplo: Ejecución instrucción 3

```
$1 = 0x10000000
$8 = 0x10002000
$9 = 0xFFFFF80
```

## Operación ADD con

```
PC = 0x00400008
RI = 0x81090000
```

```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
    .text 0x00400000
    la $t0, a
    lb $t1, 0($t0)
    lb $t2, 2($t0)
    lb $t3, 4($t0)
    lb $t4, 6($t0)
    lb $t5, 8($t0)
    lb $t6, 10($t0)
    lb $t7, 12($t0)
    lb $t8, 14($t0)
    lb $t9, 16($t0)
    lb $t10, 18($t0)
    lb $t11, 20($t0)
    lb $t12, 22($t0)
    lb $t13, 24($t0)
    lb $t14, 26($t0)
    lb $t15, 28($t0)
    lb $t16, 30($t0)
    lb $t17, 32($t0)
    lb $t18, 34($t0)
    lb $t19, 36($t0)
    lb $t20, 38($t0)
    lb $t21, 40($t0)
    lb $t22, 42($t0)
    lb $t23, 44($t0)
    lb $t24, 46($t0)
    lb $t25, 48($t0)
    lb $t26, 50($t0)
    lb $t27, 52($t0)
    lb $t28, 54($t0)
    lb $t29, 56($t0)
    lb $t30, 58($t0)
    lb $t31, 60($t0)
    lb $t32, 62($t0)
    lb $t33, 64($t0)
    lb $t34, 66($t0)
    lb $t35, 68($t0)
    lb $t36, 70($t0)
    lb $t37, 72($t0)
    lb $t38, 74($t0)
    lb $t39, 76($t0)
    lb $t40, 78($t0)
    lb $t41, 80($t0)
    lb $t42, 82($t0)
    lb $t43, 84($t0)
    lb $t44, 86($t0)
    lb $t45, 88($t0)
    lb $t46, 90($t0)
    lb $t47, 92($t0)
    lb $t48, 94($t0)
    lb $t49, 96($t0)
    lb $t50, 98($t0)
    lb $t51, 100($t0)
    lb $t52, 102($t0)
    lb $t53, 104($t0)
    lb $t54, 106($t0)
    lb $t55, 108($t0)
    lb $t56, 110($t0)
    lb $t57, 112($t0)
    lb $t58, 114($t0)
    lb $t59, 116($t0)
    lb $t60, 118($t0)
    lb $t61, 120($t0)
    lb $t62, 122($t0)
    lb $t63, 124($t0)
    lb $t64, 126($t0)
    lb $t65, 128($t0)
    lb $t66, 130($t0)
    lb $t67, 132($t0)
    lb $t68, 134($t0)
    lb $t69, 136($t0)
    lb $t70, 138($t0)
    lb $t71, 140($t0)
    lb $t72, 142($t0)
    lb $t73, 144($t0)
    lb $t74, 146($t0)
    lb $t75, 148($t0)
    lb $t76, 150($t0)
    lb $t77, 152($t0)
    lb $t78, 154($t0)
    lb $t79, 156($t0)
    lb $t80, 158($t0)
    lb $t81, 160($t0)
    lb $t82, 162($t0)
    lb $t83, 164($t0)
    lb $t84, 166($t0)
    lb $t85, 168($t0)
    lb $t86, 170($t0)
    lb $t87, 172($t0)
    lb $t88, 174($t0)
    lb $t89, 176($t0)
    lb $t90, 178($t0)
    lb $t91, 180($t0)
    lb $t92, 182($t0)
    lb $t93, 184($t0)
    lb $t94, 186($t0)
    lb $t95, 188($t0)
    lb $t96, 190($t0)
    lb $t97, 192($t0)
    lb $t98, 194($t0)
    lb $t99, 196($t0)
    lb $t100, 198($t0)
    lb $t101, 200($t0)
    lb $t102, 202($t0)
    lb $t103, 204($t0)
    lb $t104, 206($t0)
    lb $t105, 208($t0)
    lb $t106, 210($t0)
    lb $t107, 212($t0)
    lb $t108, 214($t0)
    lb $t109, 216($t0)
    lb $t110, 218($t0)
    lb $t111, 220($t0)
    lb $t112, 222($t0)
    lb $t113, 224($t0)
    lb $t114, 226($t0)
    lb $t115, 228($t0)
    lb $t116, 230($t0)
    lb $t117, 232($t0)
    lb $t118, 234($t0)
    lb $t119, 236($t0)
    lb $t120, 238($t0)
    lb $t121, 240($t0)
    lb $t122, 242($t0)
    lb $t123, 244($t0)
    lb $t124, 246($t0)
    lb $t125, 248($t0)
    lb $t126, 250($t0)
    lb $t127, 252($t0)
    lb $t128, 254($t0)
    lb $t129, 256($t0)
    lb $t130, 258($t0)
    lb $t131, 260($t0)
    lb $t132, 262($t0)
    lb $t133, 264($t0)
    lb $t134, 266($t0)
    lb $t135, 268($t0)
    lb $t136, 270($t0)
    lb $t137, 272($t0)
    lb $t138, 274($t0)
    lb $t139, 276($t0)
    lb $t140, 278($t0)
    lb $t141, 280($t0)
    lb $t142, 282($t0)
    lb $t143, 284($t0)
    lb $t144, 286($t0)
    lb $t145, 288($t0)
    lb $t146, 290($t0)
    lb $t147, 292($t0)
    lb $t148, 294($t0)
    lb $t149, 296($t0)
    lb $t150, 298($t0)
    lb $t151, 300($t0)
    lb $t152, 302($t0)
    lb $t153, 304($t0)
    lb $t154, 306($t0)
    lb $t155, 308($t0)
    lb $t156, 310($t0)
    lb $t157, 312($t0)
    lb $t158, 314($t0)
    lb $t159, 316($t0)
    lb $t160, 318($t0)
    lb $t161, 320($t0)
    lb $t162, 322($t0)
    lb $t163, 324($t0)
    lb $t164, 326($t0)
    lb $t165, 328($t0)
    lb $t166, 330($t0)
    lb $t167, 332($t0)
    lb $t168, 334($t0)
    lb $t169, 336($t0)
    lb $t170, 338($t0)
    lb $t171, 340($t0)
    lb $t172, 342($t0)
    lb $t173, 344($t0)
    lb $t174, 346($t0)
    lb $t175, 348($t0)
    lb $t176, 350($t0)
    lb $t177, 352($t0)
    lb $t178, 354($t0)
    lb $t179, 356($t0)
    lb $t180, 358($t0)
    lb $t181, 360($t0)
    lb $t182, 362($t0)
    lb $t183, 364($t0)
    lb $t184, 366($t0)
    lb $t185, 368($t0)
    lb $t186, 370($t0)
    lb $t187, 372($t0)
    lb $t188, 374($t0)
    lb $t189, 376($t0)
    lb $t190, 378($t0)
    lb $t191, 380($t0)
    lb $t192, 382($t0)
    lb $t193, 384($t0)
    lb $t194, 386($t0)
    lb $t195, 388($t0)
    lb $t196, 390($t0)
    lb $t197, 392($t0)
    lb $t198, 394($t0)
    lb $t199, 396($t0)
    lb $t200, 398($t0)
    lb $t201, 400($t0)
    lb $t202, 402($t0)
    lb $t203, 404($t0)
    lb $t204, 406($t0)
    lb $t205, 408($t0)
    lb $t206, 410($t0)
    lb $t207, 412($t0)
    lb $t208, 414($t0)
    lb $t209, 416($t0)
    lb $t210, 418($t0)
    lb $t211, 420($t0)
    lb $t212, 422($t0)
    lb $t213, 424($t0)
    lb $t214, 426($t0)
    lb $t215, 428($t0)
    lb $t216, 430($t0)
    lb $t217, 432($t0)
    lb $t218, 434($t0)
    lb $t219, 436($t0)
    lb $t220, 438($t0)
    lb $t221, 440($t0)
    lb $t222, 442($t0)
    lb $t223, 444($t0)
    lb $t224, 446($t0)
    lb $t225, 448($t0)
    lb $t226, 450($t0)
    lb $t227, 452($t0)
    lb $t228, 454($t0)
    lb $t229, 456($t0)
    lb $t230, 458($t0)
    lb $t231, 460($t0)
    lb $t232, 462($t0)
    lb $t233, 464($t0)
    lb $t234, 466($t0)
    lb $t235, 468($t0)
    lb $t236, 470($t0)
    lb $t237, 472($t0)
    lb $t238, 474($t0)
    lb $t239, 476($t0)
    lb $t240, 478($t0)
    lb $t241, 480($t0)
    lb $t242, 482($t0)
    lb $t243, 484($t0)
    lb $t244, 486($t0)
    lb $t245, 488($t0)
    lb $t246, 490($t0)
    lb $t247
```

Lectura del byte con dirección \$t0+0 y almacenamiento en \$t1

## Copr 1 (FPU)

registros

**\$0**

•

•

100%

**\$51**

A Venn diagram with two overlapping circles. The left circle is labeled 'All people' and the right circle is labeled 'All people who are not in the room'. The intersection of the two circles is shaded in light blue.

ALU

---

## Cause

# EPC

---

---

A0, 0:

---

 $\$+0+0$ 

---

```

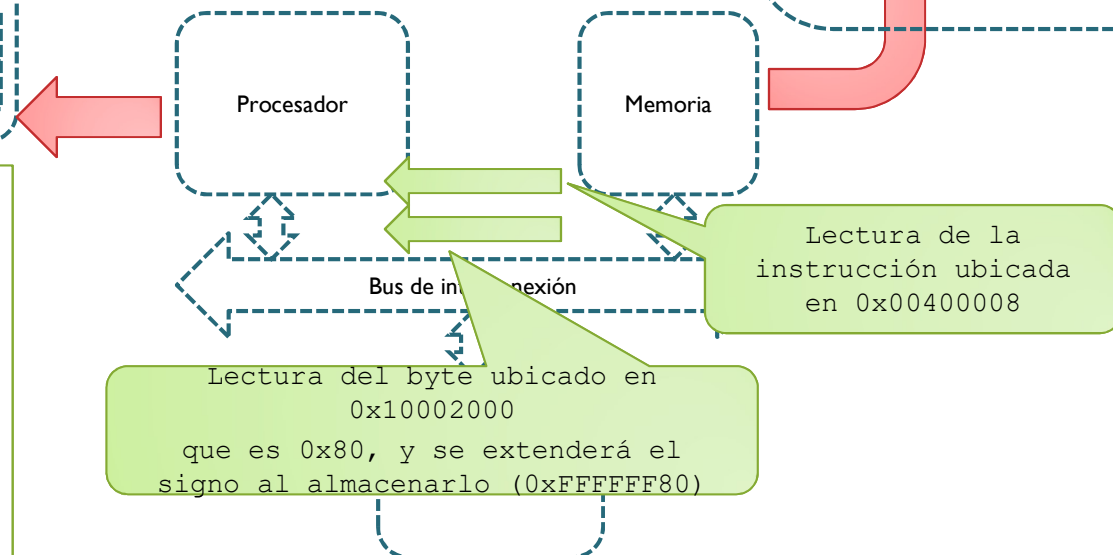
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
    .text 0x00400000
    lui $1, 4096
    ori $8, $1, 8192
lb $9, 0($8)
    lb $10, 2($8)
    add $11, $9, $10
    sb $11, 3($8)
    add $t1, $0, $0

```

1b \$9, 0 (\$8)

## Contenido memoria

Dirección	Contenido
\$8) 0x00000000	0x3c011000
0x00400004	0x34282000
0x00400008	0x81090000
0x0040000C	0x810a0002
0x00400010	0x012a5820
0x00400014	0xa10b0003
0x00400018	0x00004820
...	
0x10002000	0x00a0ff80
0x10002004	0xffffffff
...	



## Ejemplo: Ejecución instrucción 4

### Contenido memoria

Dirección	Contenido
0x00000000	0x3c011000
0x00000004	0x34282000
0x00400008	0x81090000
0x0040000C	0x810a0002
0x00400010	0x012a5820
0x00400014	0xa10b0003
0x00400018	0x00004820
...	
0x10002000	0x00a0ff80
0x10002004	0xffffffff
...	

\$1 = 0x10000000  
\$8 = 0x10002000  
\$9 = 0xffffffff80  
\$10 = 0xffffffffa0

Operación ADD con  
[\$8] y 2

Copr 1 (FPU)

registros

\$0

.

.

.

\$31

ALU

LO

HI

PC

RI

procesador 0

registros

BadVAddr

Cause

Status

EPC

PC = 0x0040000C  
RI = 0x810a0002

```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
.text 0x00400000
lui $1, 4096
ori $8, $1, 8192
lb $9, 0($8)
lb $10, 2($8)
add $11, $9, $10
sb $11, 3($8)
add $9, $0, $0
```

**lb \$10, 2(\$8)**

Procesador

Memoria

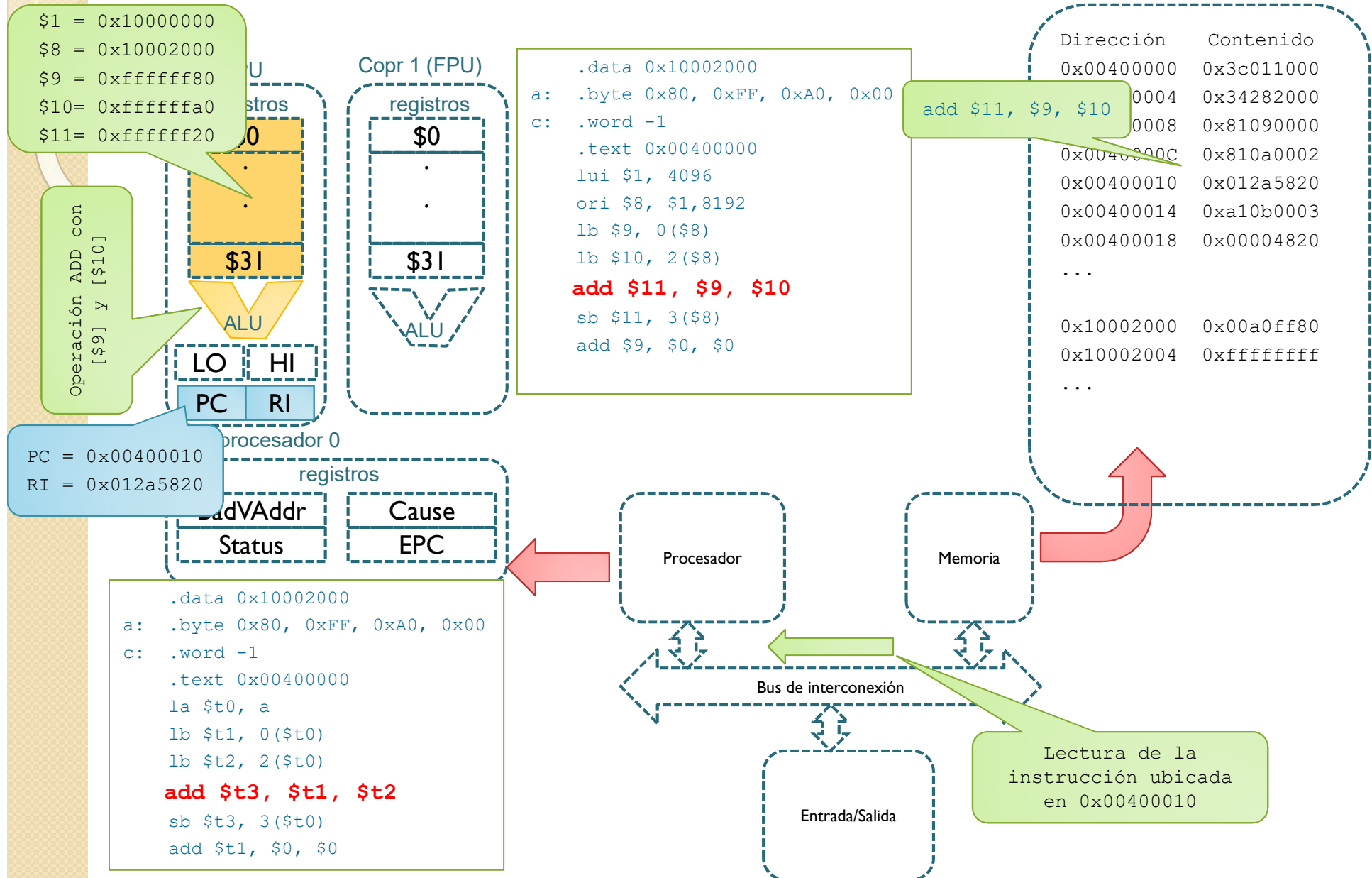
Bus de interconexión

Lectura de la  
instrucción ubicada  
en 0x0040000C

Lectura del byte ubicado en 0x10002002,  
que es 0xa0, y se extiende el signo al  
almacenarlo (0xFFFFFFFFA0)

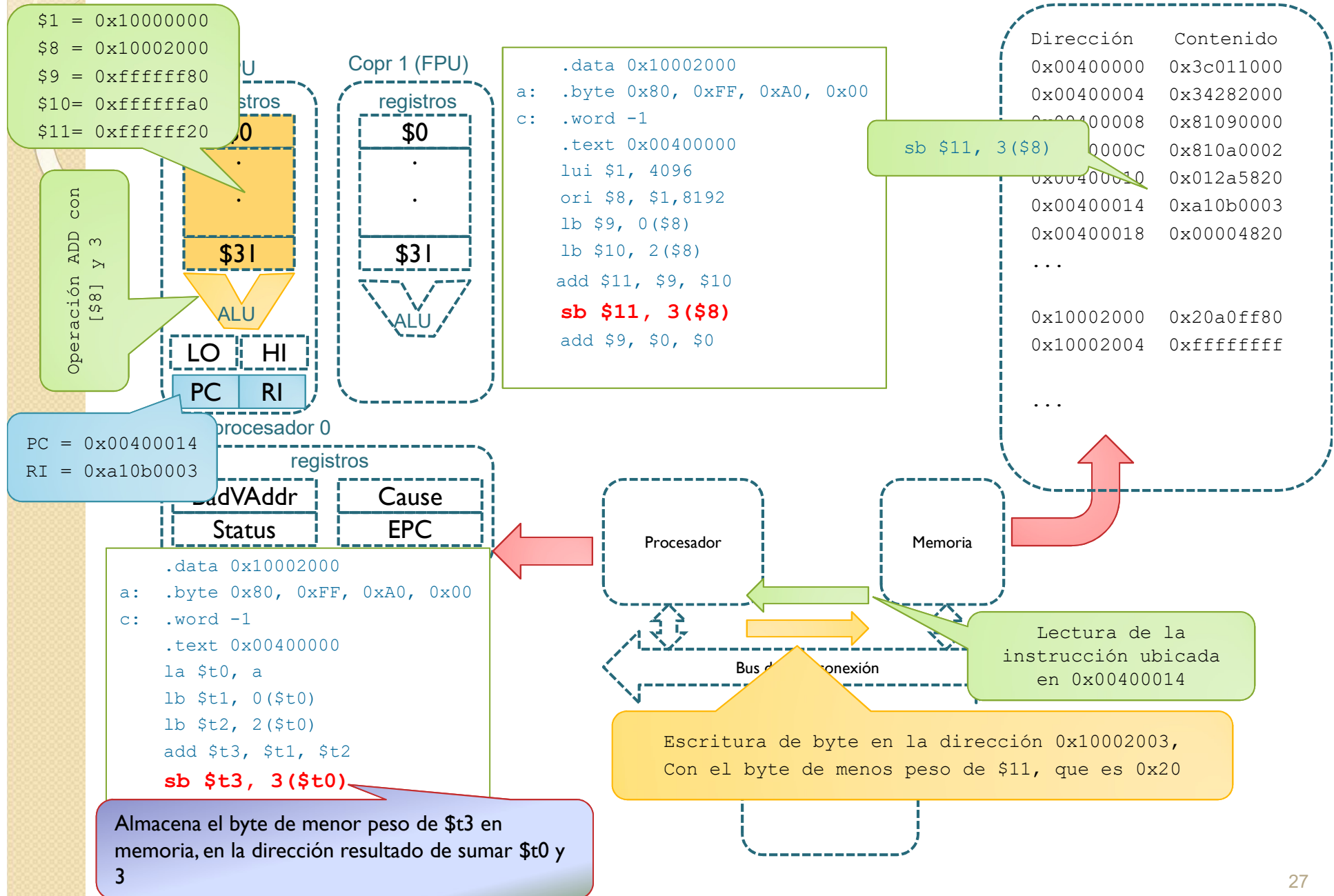
```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
.text 0x00400000
la $t0, a
lb $t1, 0($t0)
lb $t2, 2($t0)
add $t3, $t1, $t2
sb $t3, 3($t0)
add $t1, $0, $0
```

## Ejemplo: Ejecución instrucción 5

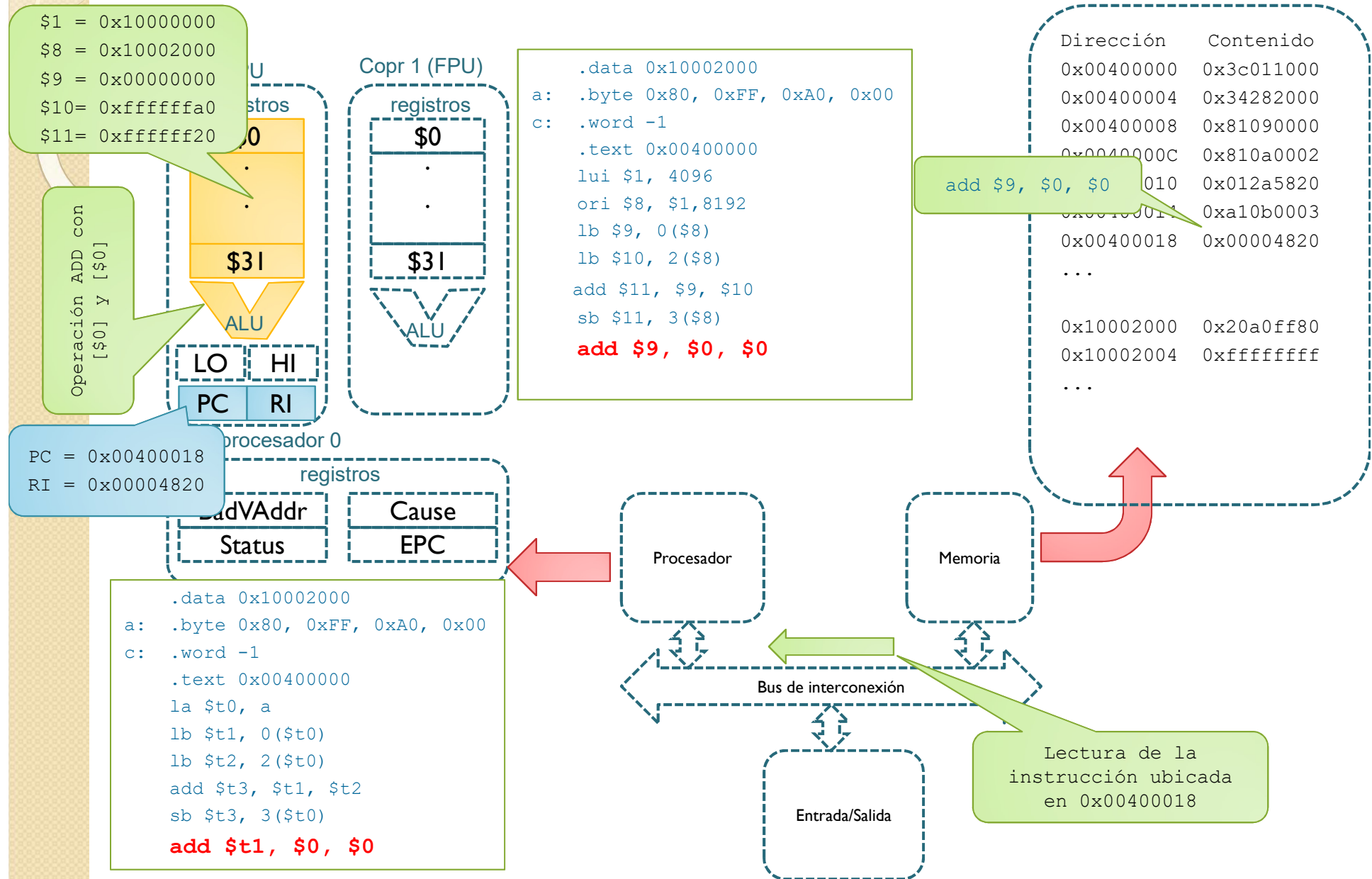




## Ejemplo: Ejecución instrucción 6



## Ejemplo: Ejecución instrucción 7



# Contenido y Bibliografía

- Introducción
- I – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

# La ruta de los datos y la unidad de control

- Diseño del procesador
- Elección del juego de instrucciones a ejecutar
  - ✓ Subconjunto del juego de instrucciones del MIPS
    - Aritméticas y comparación: `add`, `sub`, `and`, `or`, `slt`
    - Carga/almacenamiento: `lw`, `sw`
    - Salto: `beq`, `jump` (la instrucción `jump` será añadida posteriormente)
- Ciclo único de reloj

# La ruta de los datos y la unidad de control

- Diseño del procesador
  - ✓ Combinación de elementos lógicos sencillos
    - Circuitos combinacionales: puertas, decodificadores, multiplexores, ALU, ...
    - Circuitos secuenciales: biestables, registros, banco de registros, memoria, ...
  - ✓ Las **prestaciones** dependen del número de instrucciones que ejecuta por unidad de tiempo y de la duración del ciclo de reloj
    - **Objetivo inicial: Ejecutar una instrucción por ciclo y reducir el ciclo de reloj**

# La ruta de los datos y la unidad de control

- Etapas en el diseño
  - ✓ Diseñar la ruta de datos
    - Seleccionar los elementos necesarios para la ejecución de las instrucciones
    - Interconectar los elementos
  - ✓ Diseñar la unidad de control
    - Identificar las señales de control necesarias
    - Diseñar la lógica asociada a cada señal



# Contenido y Bibliografía

- Introducción
- I – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

# Etapas de búsqueda y decodificación

Todas las instrucciones comienzan con su búsqueda en la memoria. La dirección se encuentra en el PC.

$RI \leftarrow \text{Mem}[\text{PC}]$

Una vez la instrucción está en RI la unidad de control la decodifica para conocer cuál es y ejecutarla.

- Elementos funcionales necesarios



- Ruta de los datos



- Funcionamiento






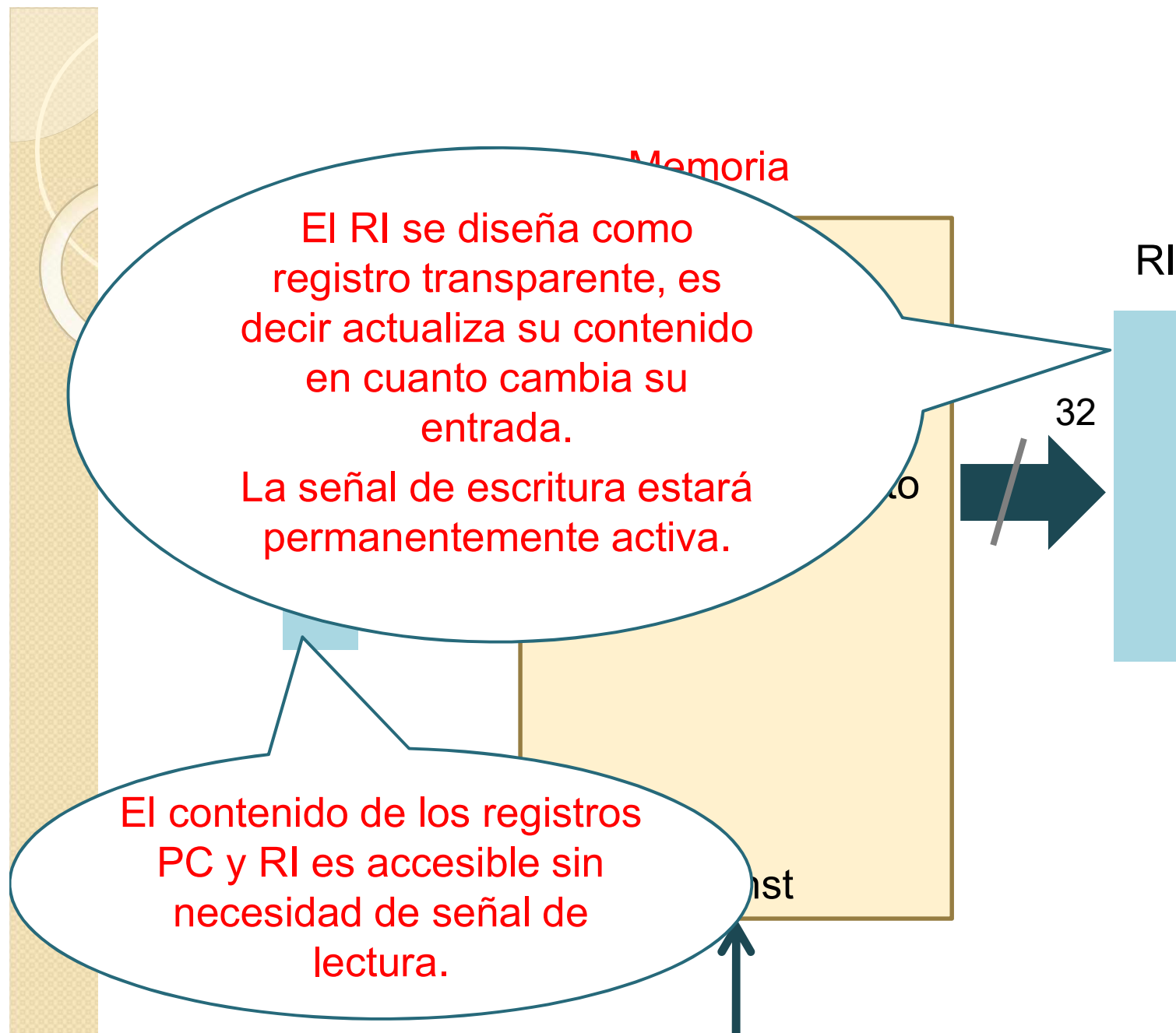
# Etapas de búsqueda y decodificación

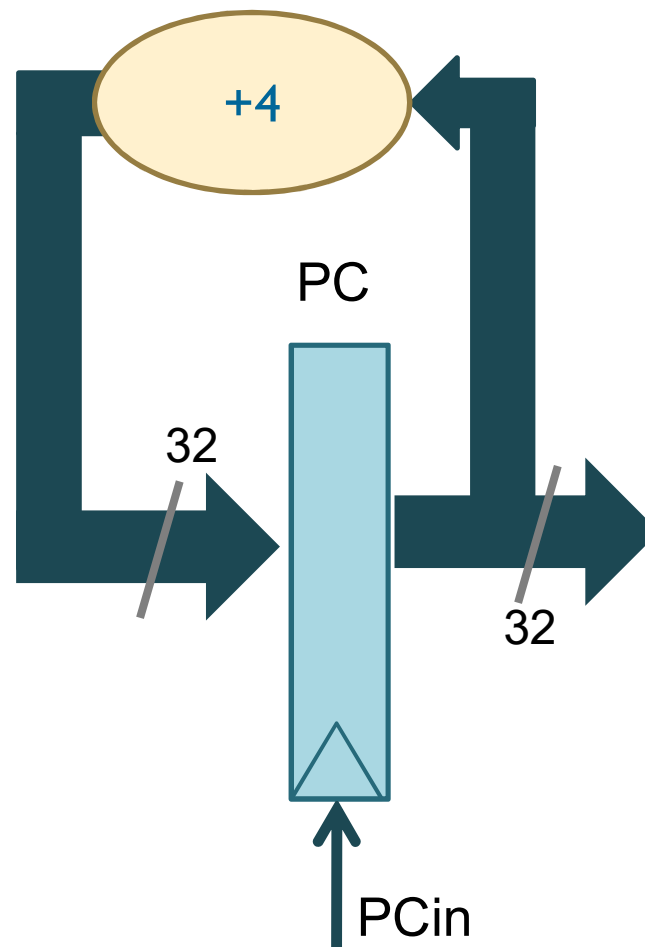
## OPERACIONES:

- Leer instrucción empleando PC y almacenar en RI
- Incrementar PC en 4
- Decodificar la instrucción en RI

## ELEMENTOS:

-  • Registro PC y RI y memoria de instrucciones.
-  • Unidad sumadora.
-  • Campos COP y Función del RI a la unidad de control





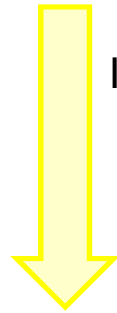
La señal PCin es la patilla de escritura del PC

✓ Será activa por flanco

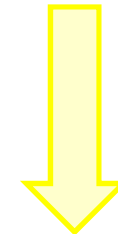
## Instrucción Aritmética tipo R

add rd, rs, rt

RI

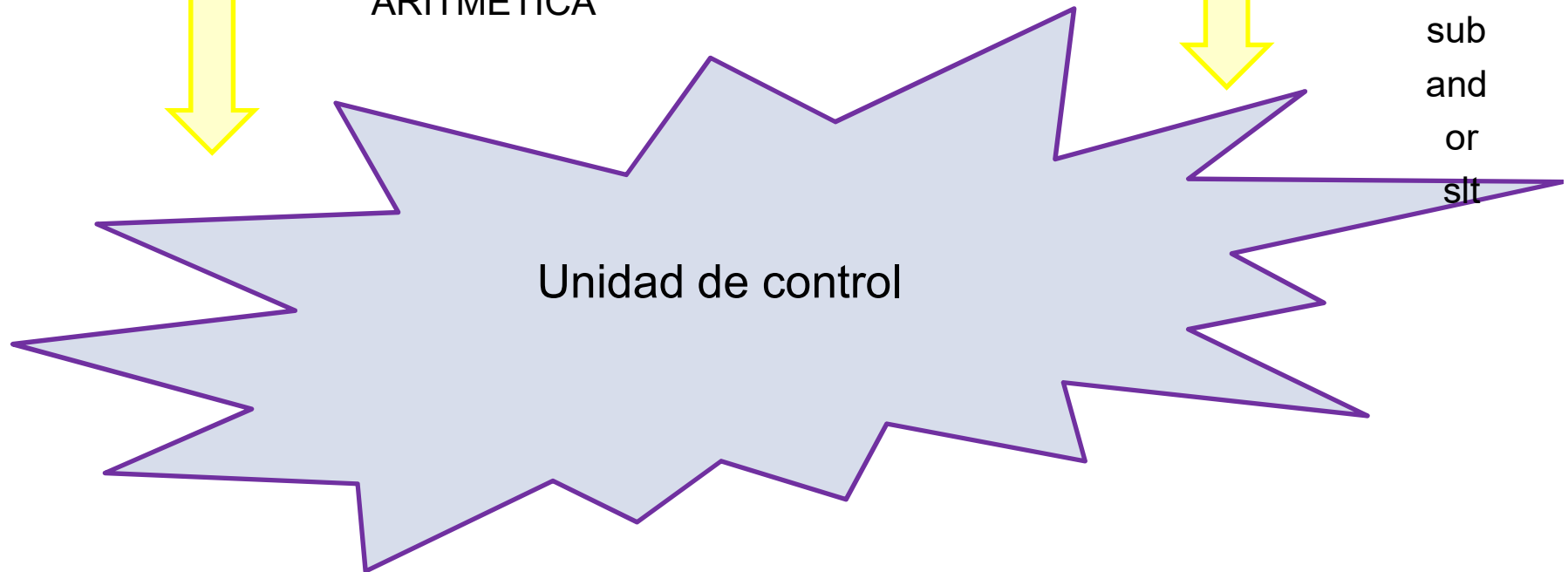


Indica que es una operación  
ARITMETICA



¿Cuál de ellas?

add  
sub  
and  
or  
slt



Unidad de control

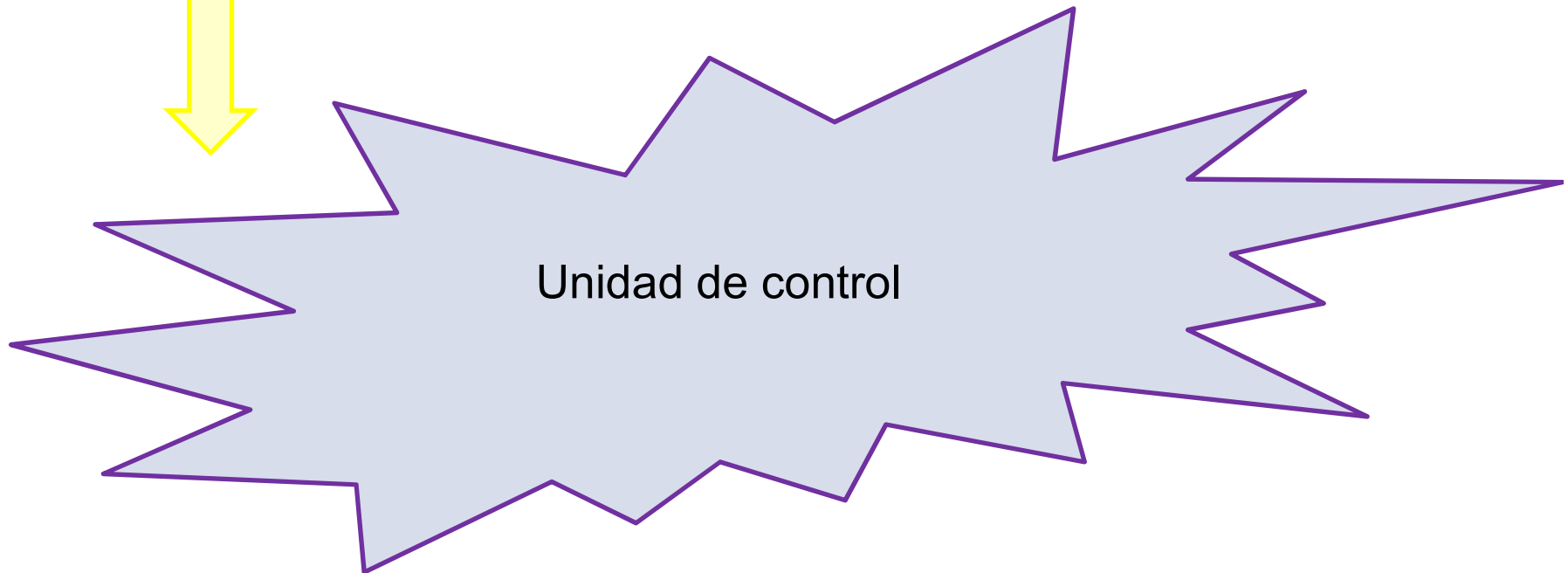
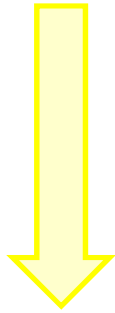
## Instrucciones tipo I

addi rt, rs, inmediato

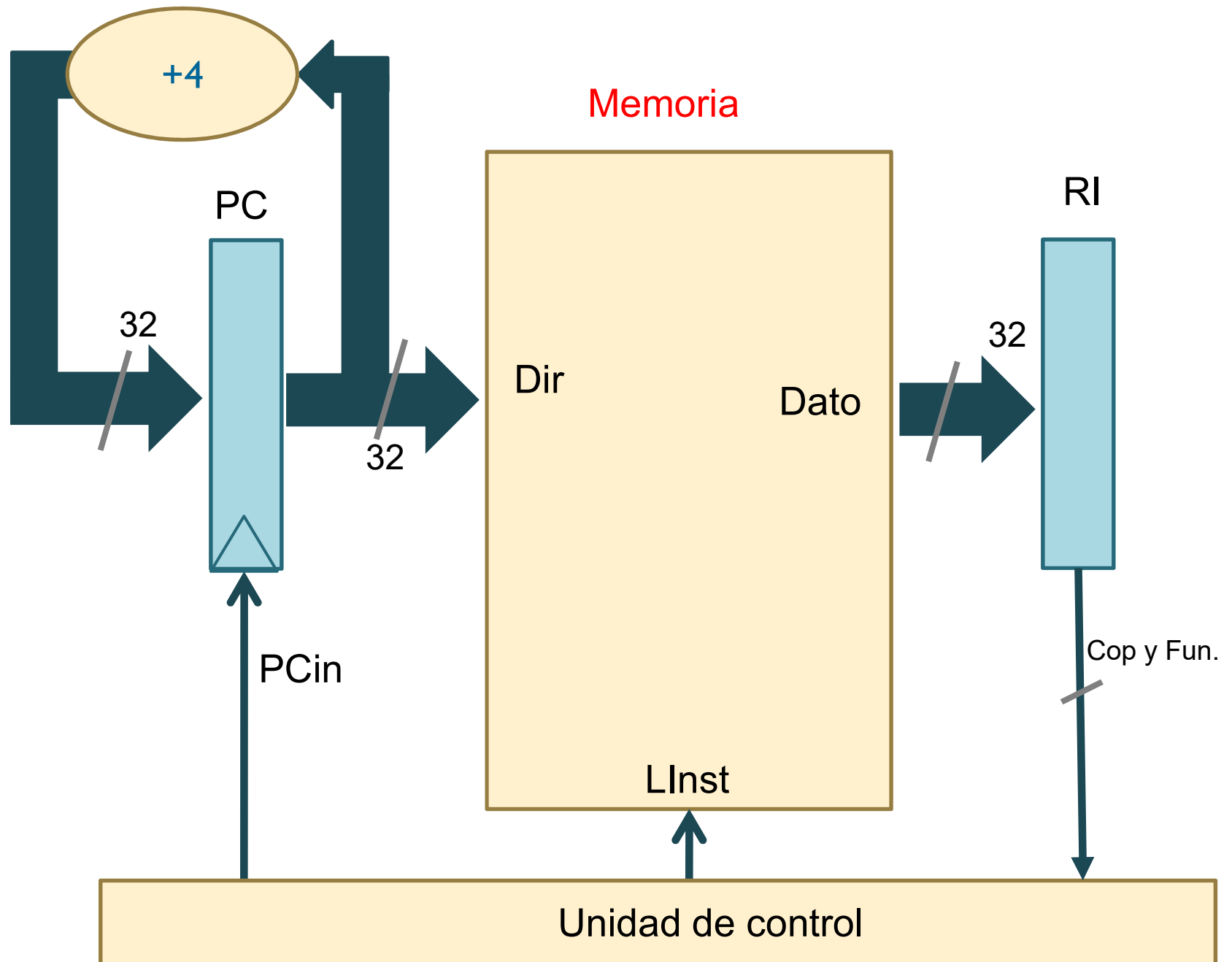
RI

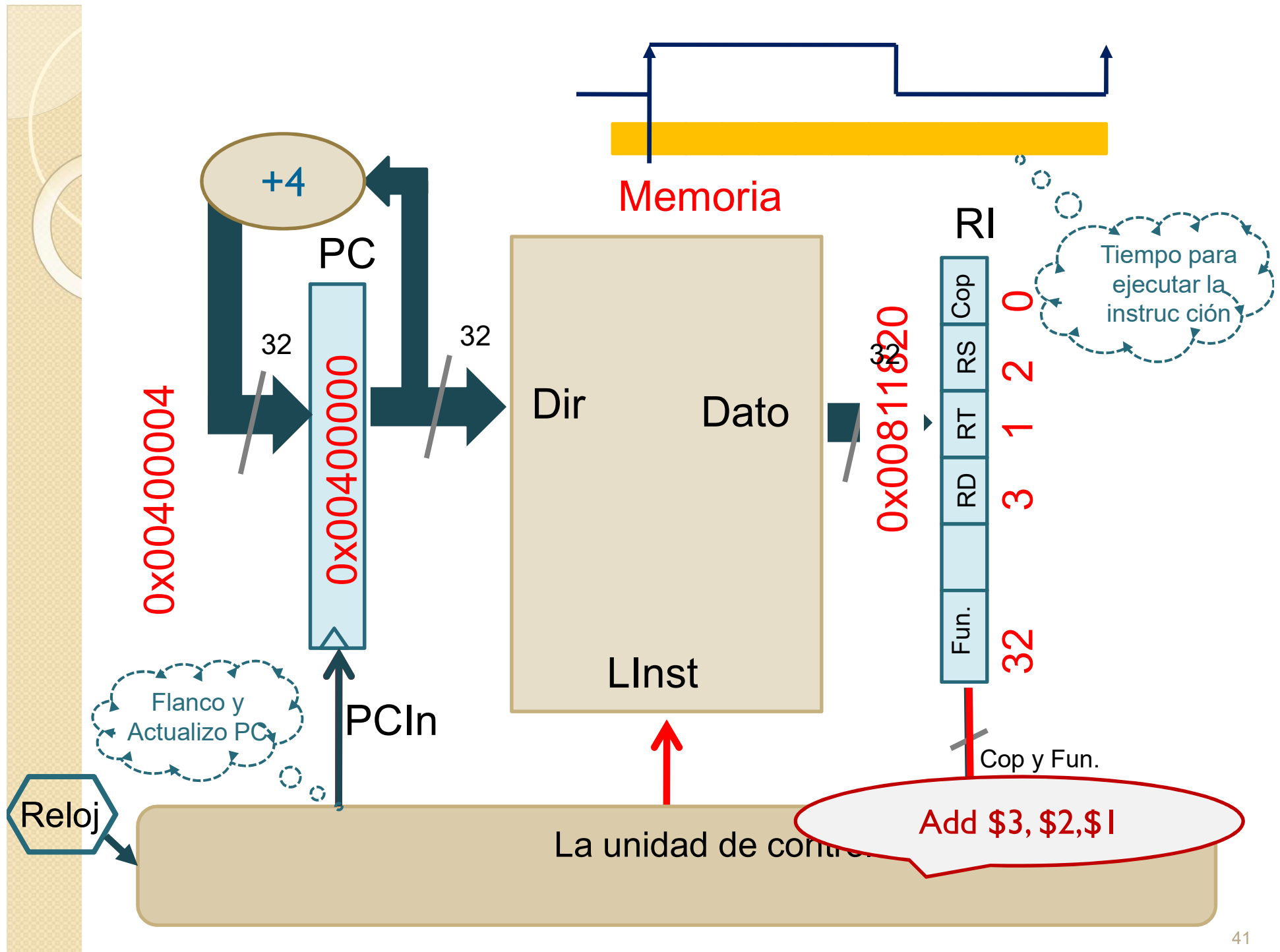


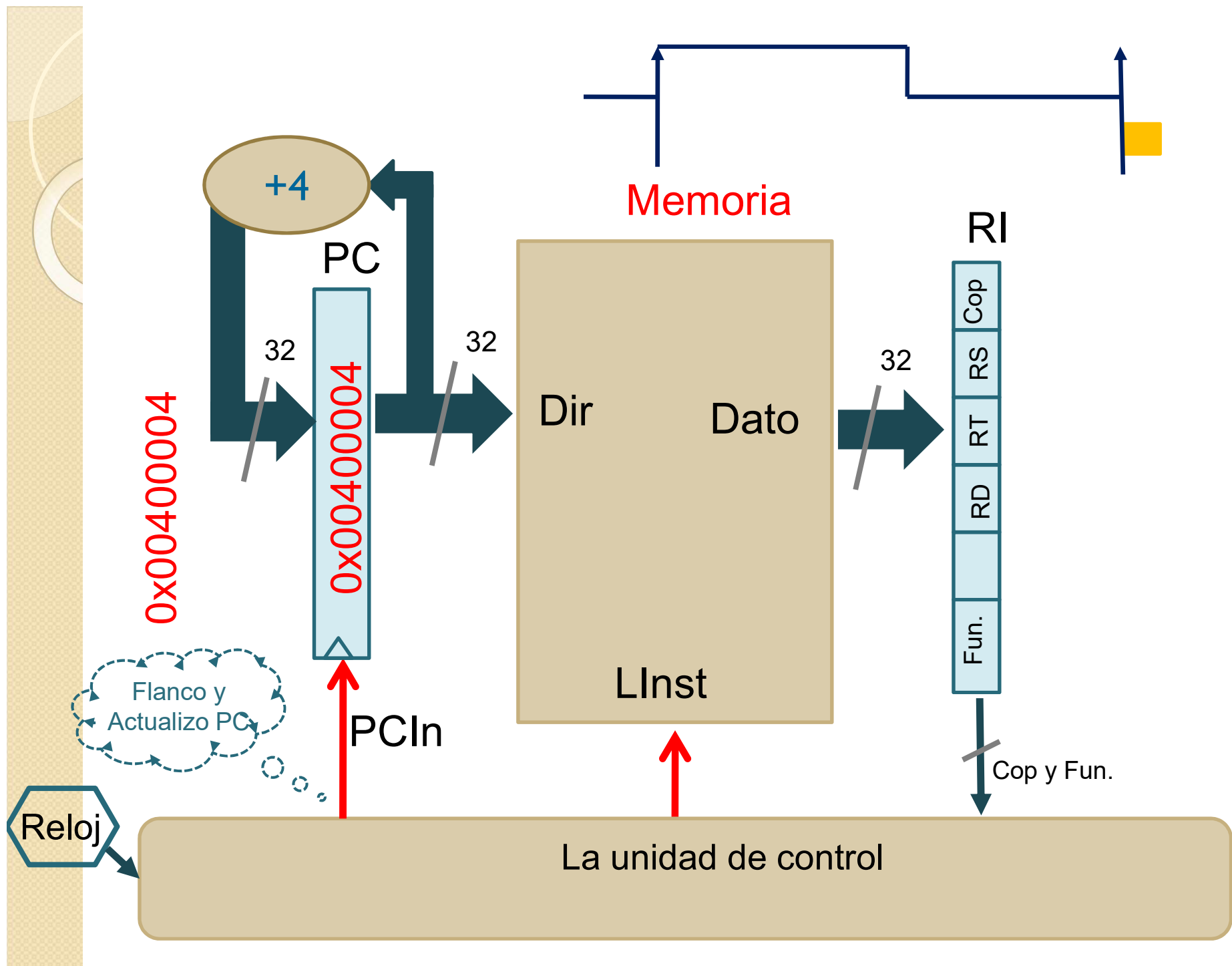
Identifica la instrucción











# Contenido y Bibliografía

- Introducción
- I – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

# La ruta de los datos aritmético/lógicas tipo R

add/sub/and/or/slt    \$3, \$1, \$2

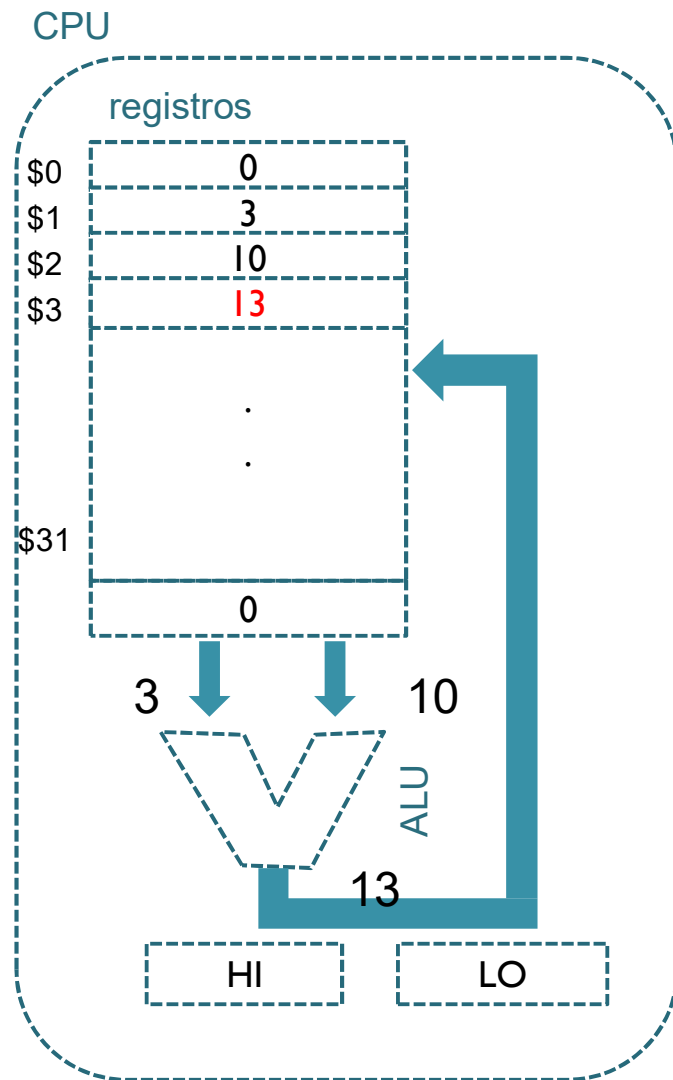
COP	RS	RT	RD		FUNC.
000000	00001	00010	00011	000000	tipo

- ¿Cómo se ejecutan estas instrucciones?

RD = RS operado RT

- Elementos funcionales necesarios
- Ruta de los datos
- Funcionamiento de la ruta
- Señales de control

# Arquitectura MIPS32. Op. aritmética tipo R



add \$3, \$1, \$2

Lectura registros \$1 y \$2

Suma




Escritura resultado en \$3

Add rd, rs, rt

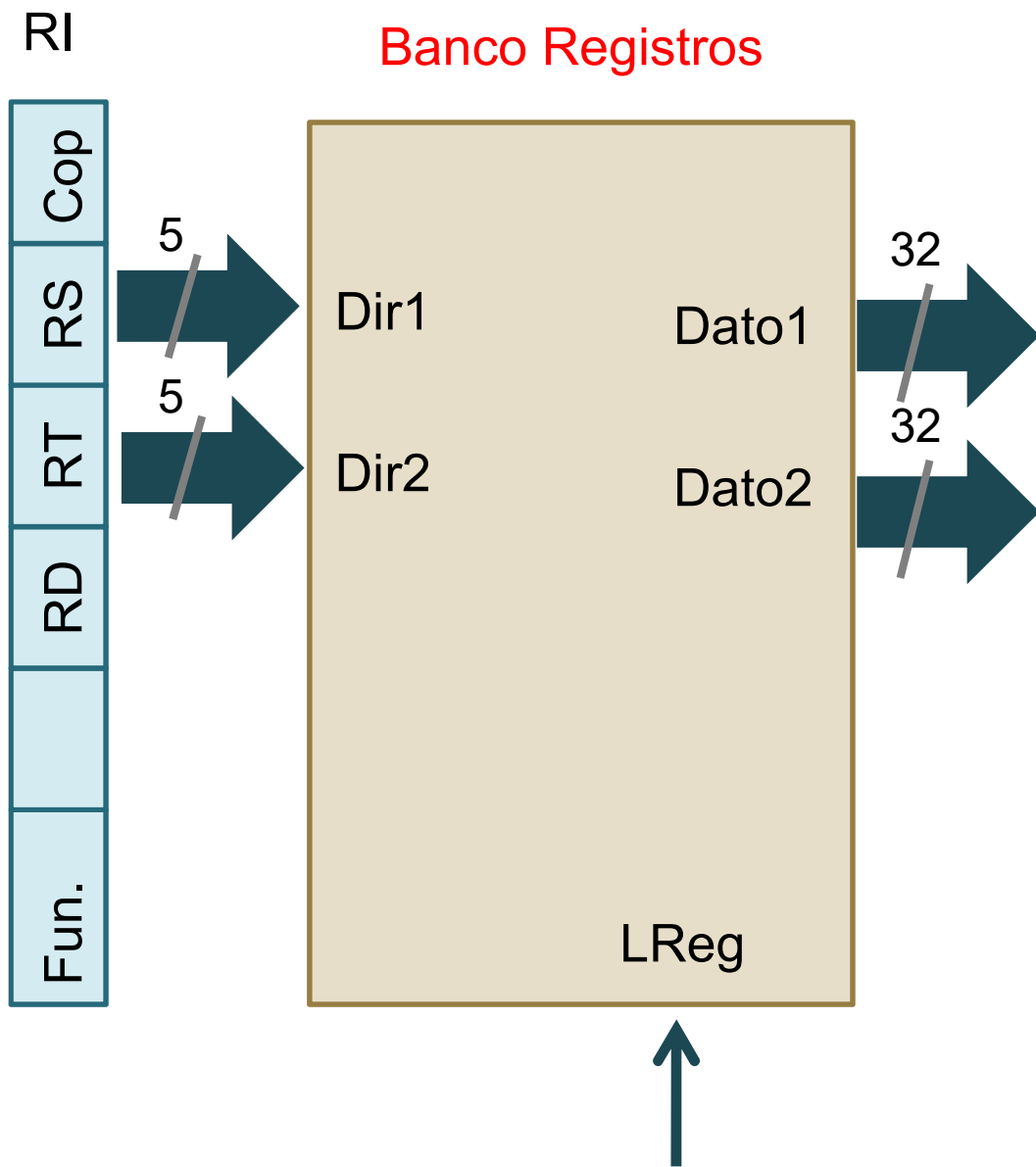
### OPERACIONES:

- Leer rs, rt
- Sumar rs y rt
- Guardar en rt dato sumado

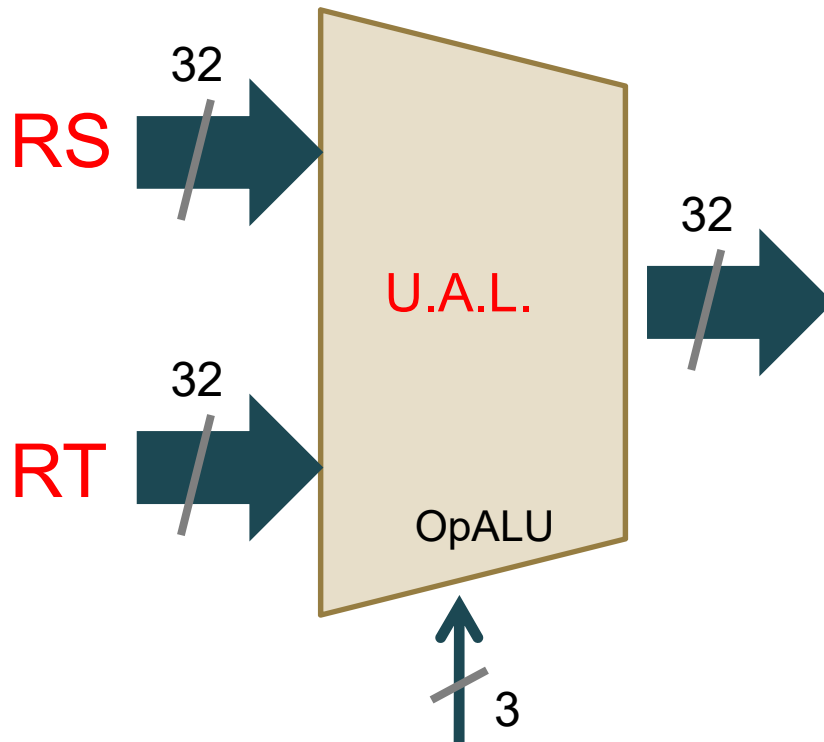
### ELEMENTOS:

-  • Banco de registros con dos direcciones de lectura
-  • UAL
-  • Banco de registros, con una tercera dirección de escritura





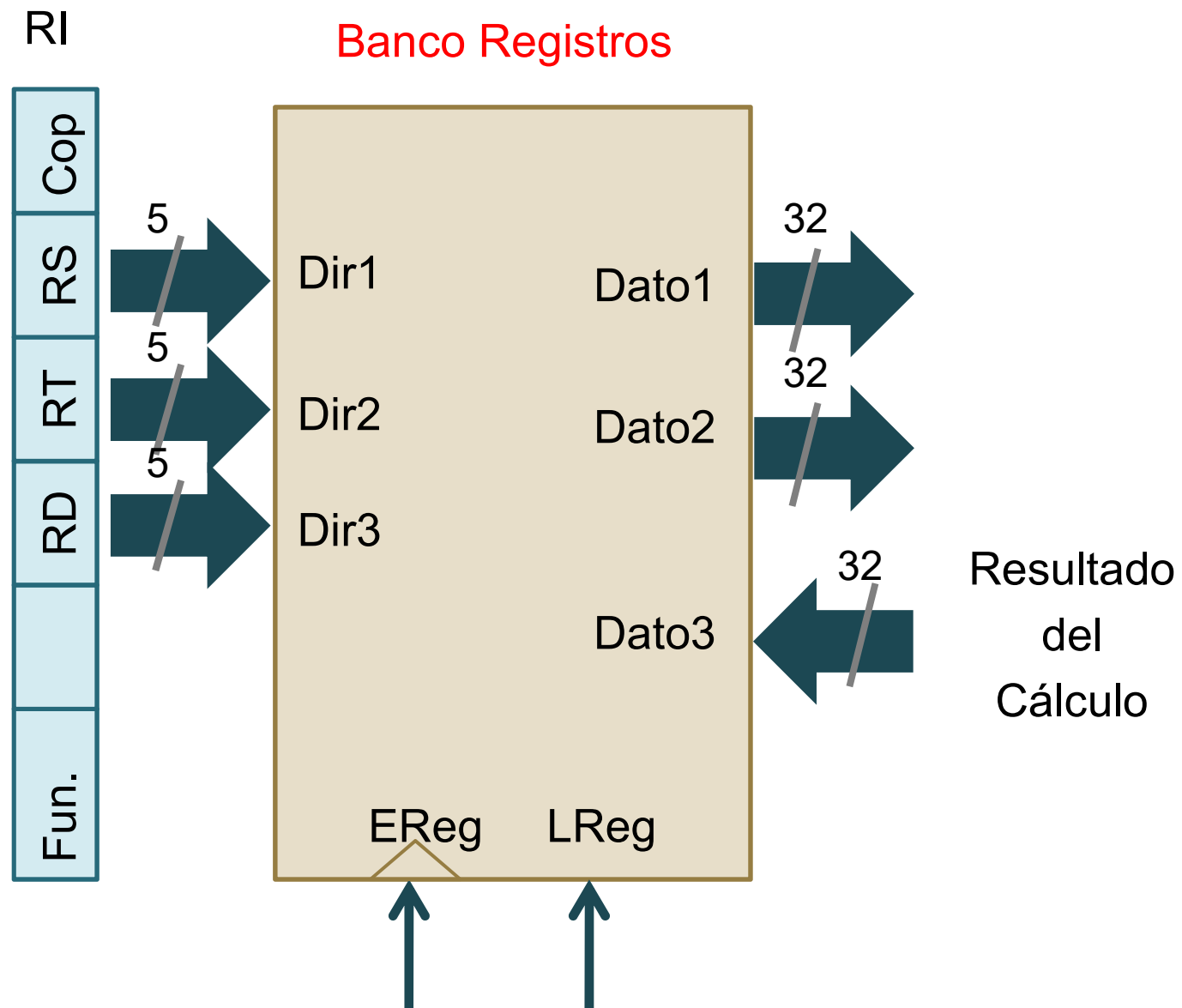
## Unidad aritmético y lógica

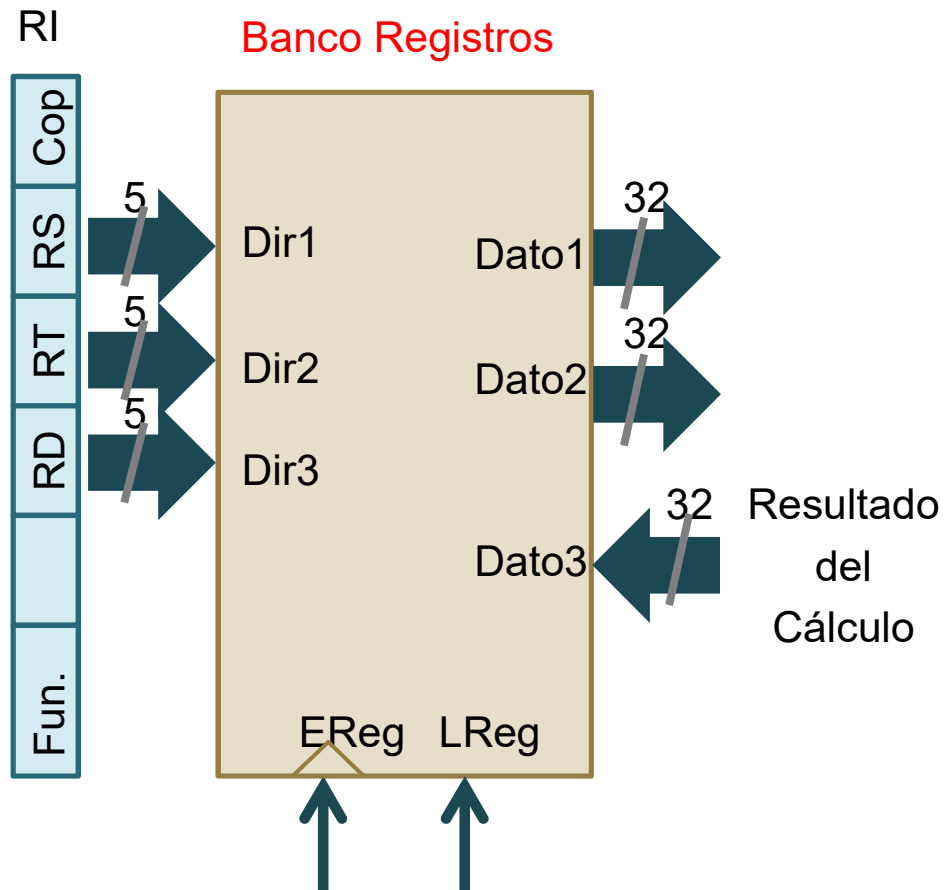


La UAL tendrá que tener las cinco operaciones: suma, resta, and, or y slt

La entrada OpALU tendrá tres bits, para poder codificar en binario las cinco operaciones

OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

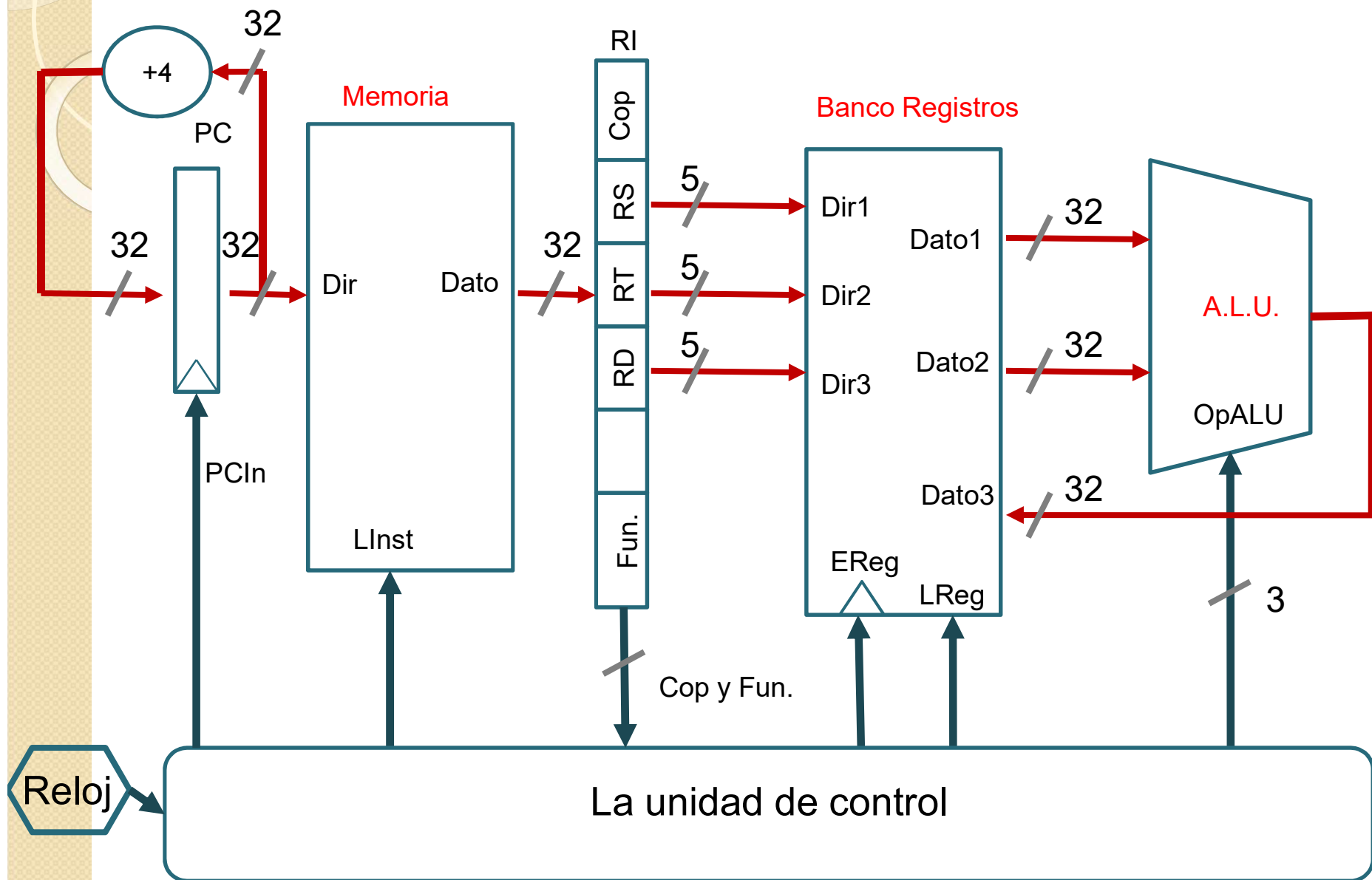




La señal **LReg**, por nivel: la lectura de los registros durará todo el ciclo

La señal **EReg**, por flanco: la escritura se realiza al final del ciclo, permitiendo así leer y escribir en el mismo ciclo en el mismo registro: ej. add \$3, \$3, \$2

# Ruta de los datos aritmético/lógicas tipo R


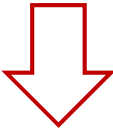




# ¿Cuál sería el T.Ciclo mínimo?

- Escritura en registros PC y RI sin coste.
- Lectura en memoria, 20ns.
- Lectura banco de registros, 15ns.
- Unidad de control 5ns en decodificar.
- UAL 5ns todas las operaciones.
- Escritura en registro, 20ns las señales estables a la entrada del banco de registros.

Todas las instrucciones del MIPS leen en el banco de registros, sin saber qué instrucción se está ejecutando, se puede acceder al mismo y ahorrar tiempo

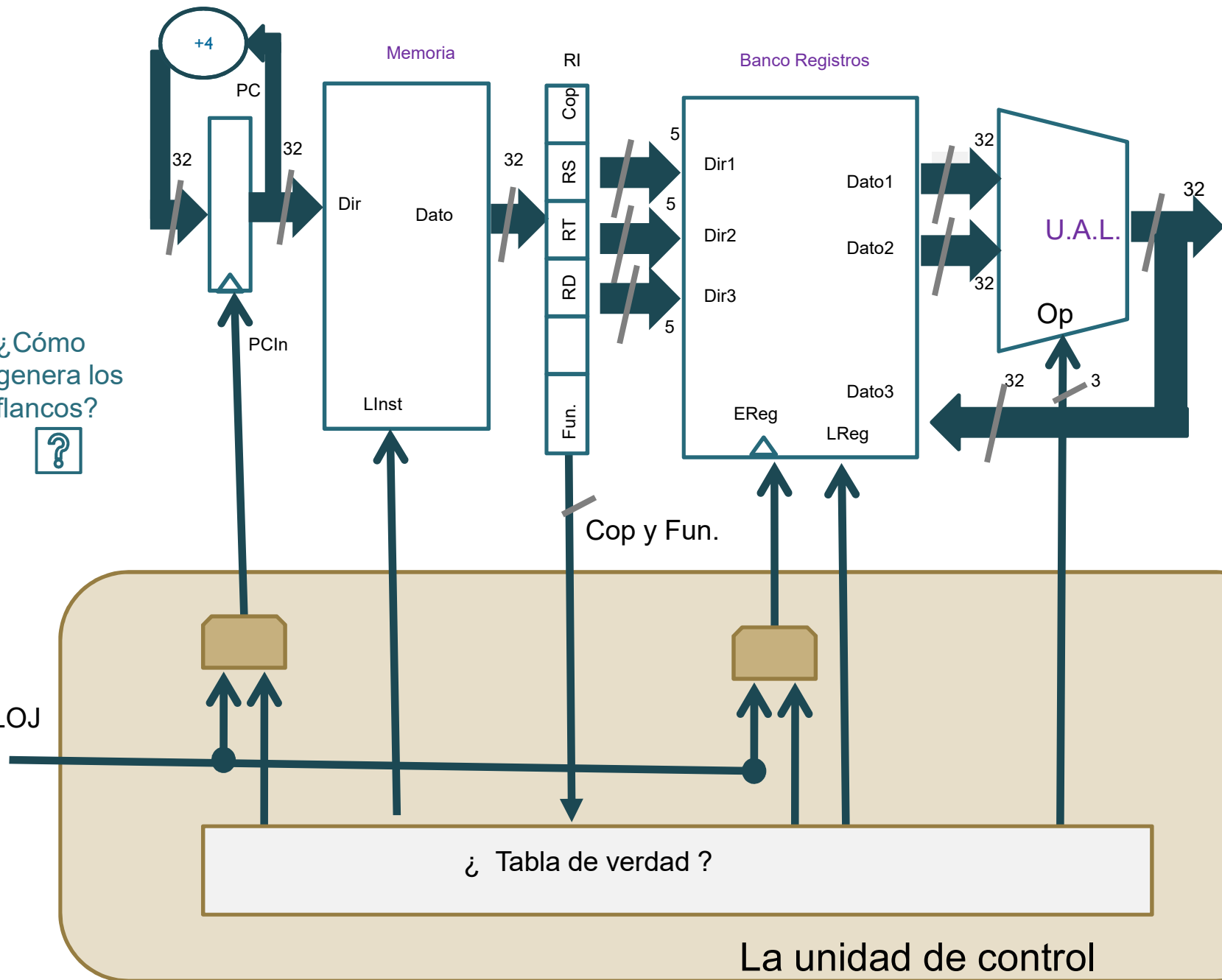
20	20
15	Max(15,5)
5	
5	5
20	20
	
65	60



¿Cómo genera los flancos?



RELOJ



## Aritmético/lógicas Tipo R: Señales de control

			↓	Mem. Incr. ↓	Banco Registros ↓		
Instrucción	Cop	Función	PCIn	LInst	LReg	EReg	OpALU
add rd, rs, rt	000000	100000	1	1	1	1	010
sub rd, rs, rt	000000	100010	1	1	1	1	110
and rd, rs, rt	000000	100100	1	1	1	1	000
or rd, rs, rt	000000	100101	1	1	1	1	001
slt rd, rs, rt)	000000	101010	1	1	1	1	111

Entradas

Salidas

OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

PCIn, LMem y LReg se  
pueden mantener siempre  
activas

# Contenido y Bibliografía

- Introducción
- I – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

# La ruta de los datos aritméticas tipo I

addi/slti    \$3, \$1, 8

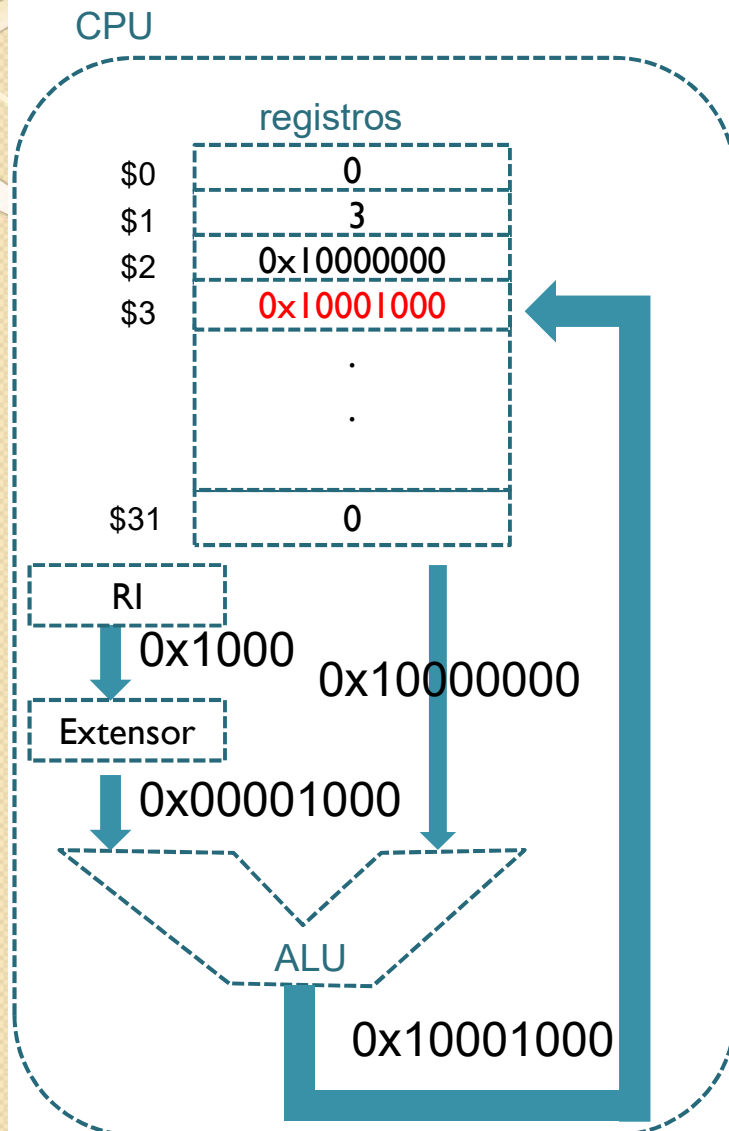
COP	RS	RT	Inmediato 16
tipo	00001	00011	00000000000001000

- ¿Cómo se ejecutan estas instrucciones?

RT = RS operado Inmediato extendido a 32b

- Elementos funcionales necesarios
- Ruta de los datos
- Funcionamiento de la ruta
- Señales de control

# Arquitectura MIPS32. Op. Carga tipo I



**addi \$3, \$2, 0x1000**

Lectura registro \$2 y dato

Suma




Escritura resultado en \$3

## Addi rt, rs, inmediato l6

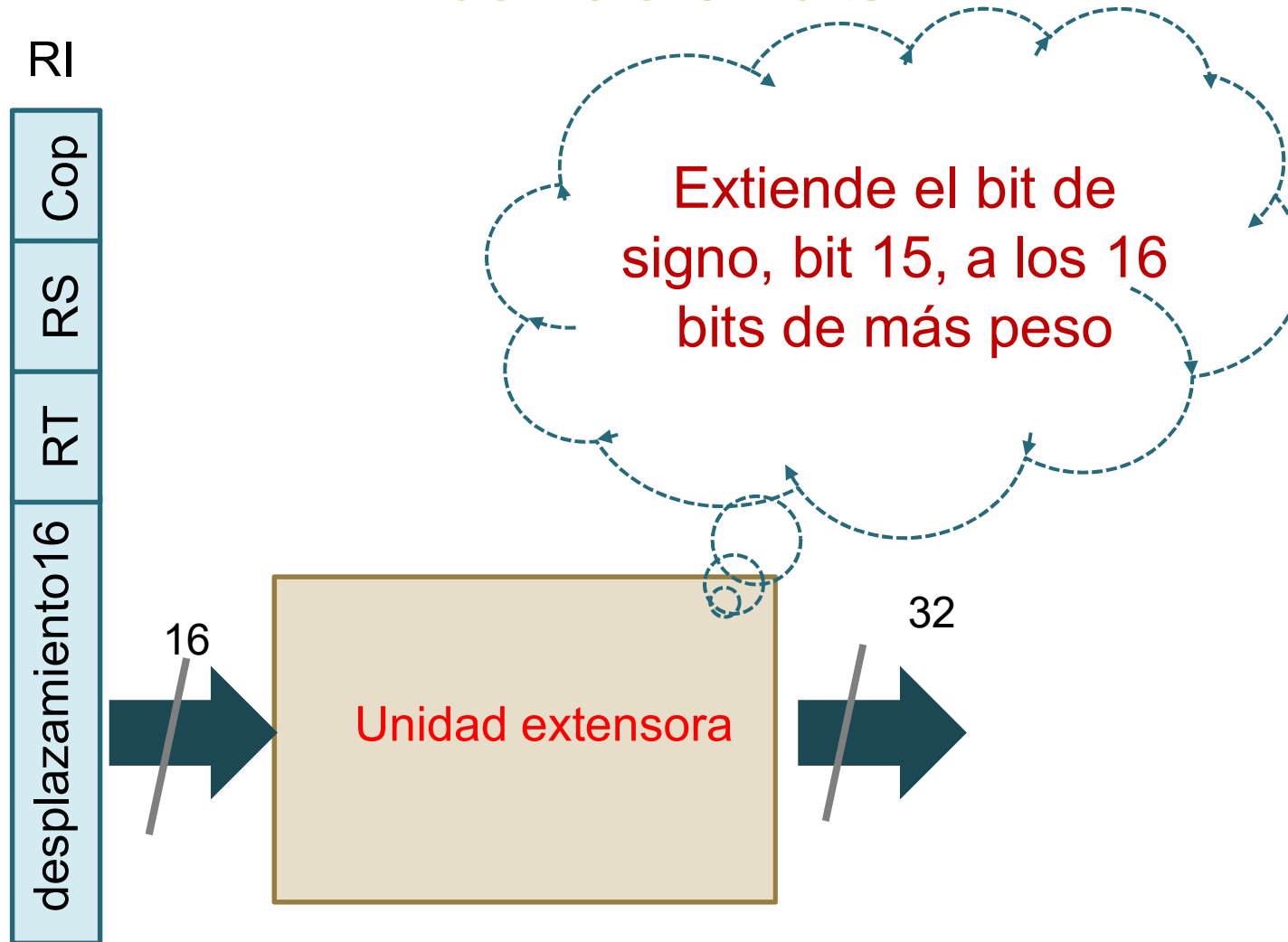
### OPERACIONES:

- Leer rs, inmediato l6
- Extender inmediato l6 a 32b
- Realizar la operación
- Guardar en rt dato sumado

### ELEMENTOS:

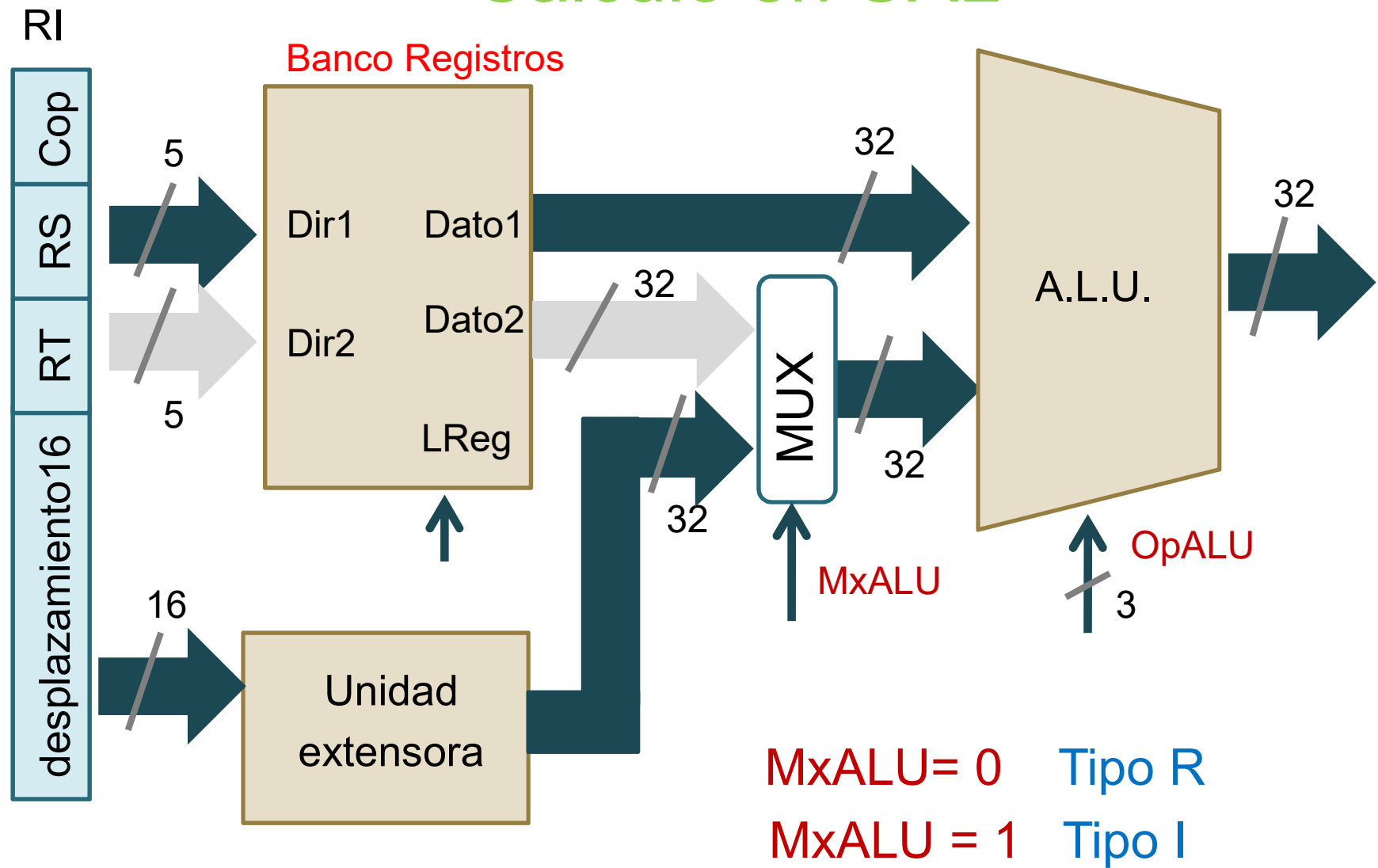
- Banco de registros (el mismo ruta tipo R)
-  • Unidad extensora del bit de signo.
-  • Multiplexor en una entrada de la UAL.
-  • Multiplexores en entradas del Banco de registros.

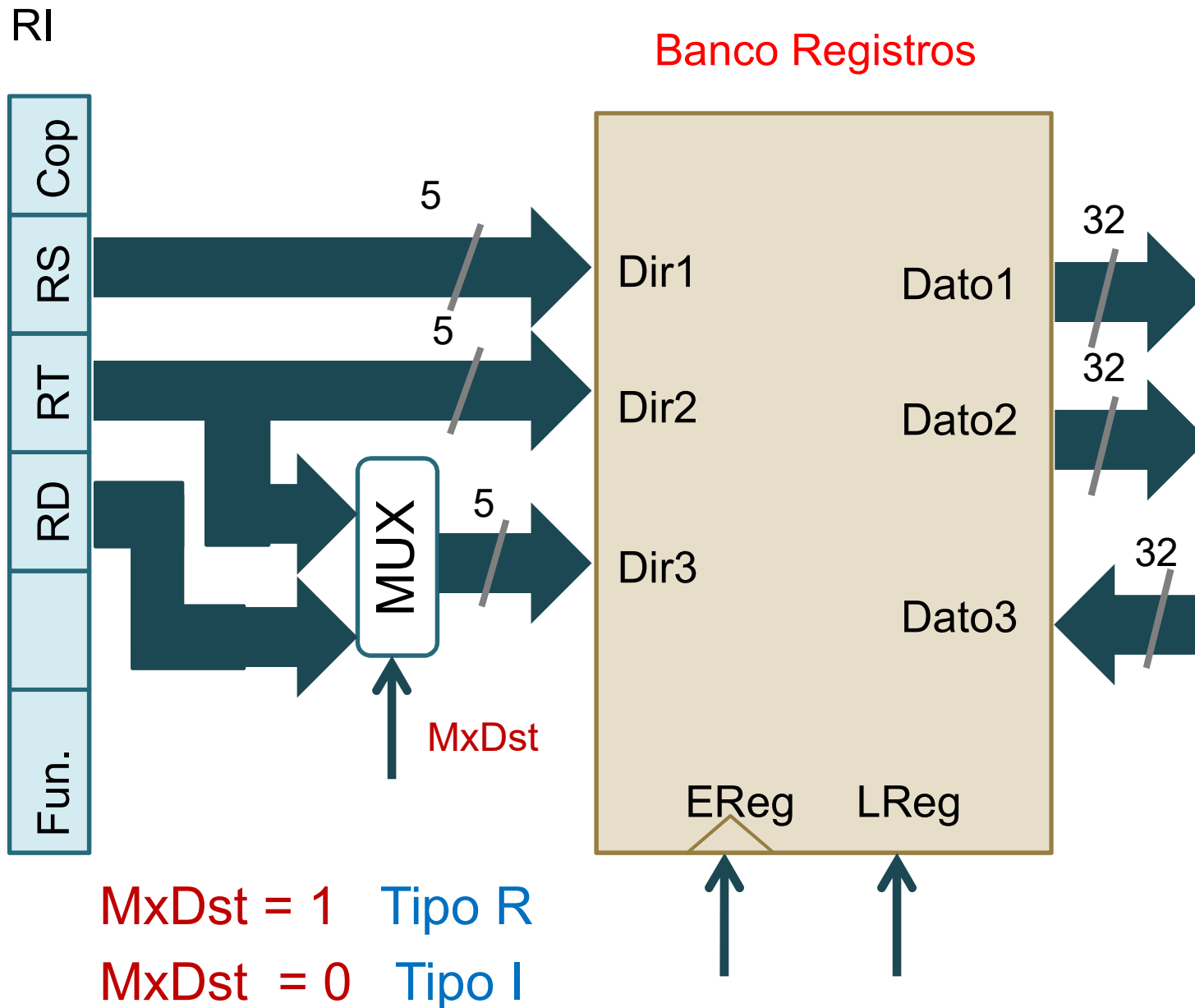
## Operación: extender desplazamiento de 16 a 32 bits



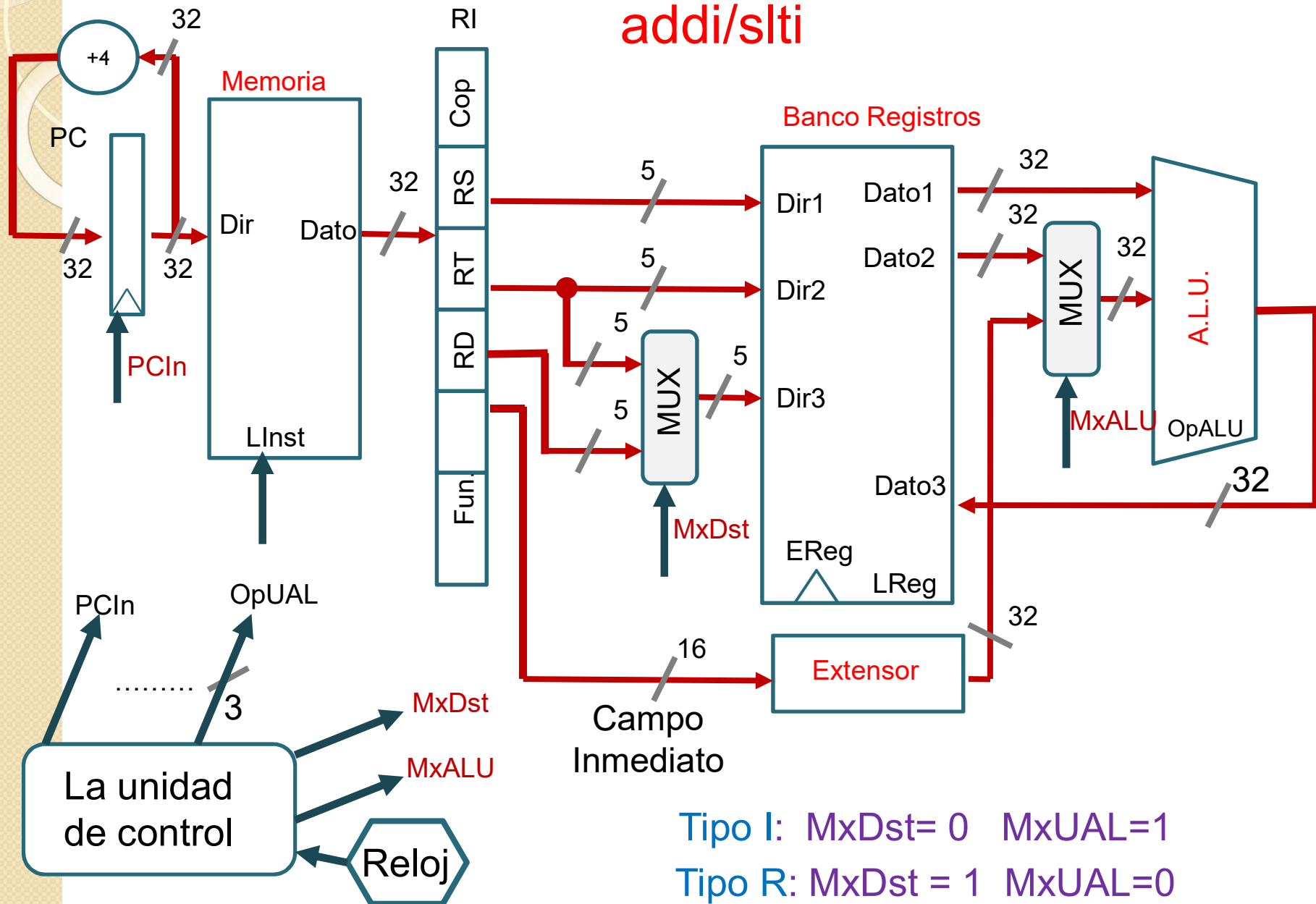


# Cálculo en UAL

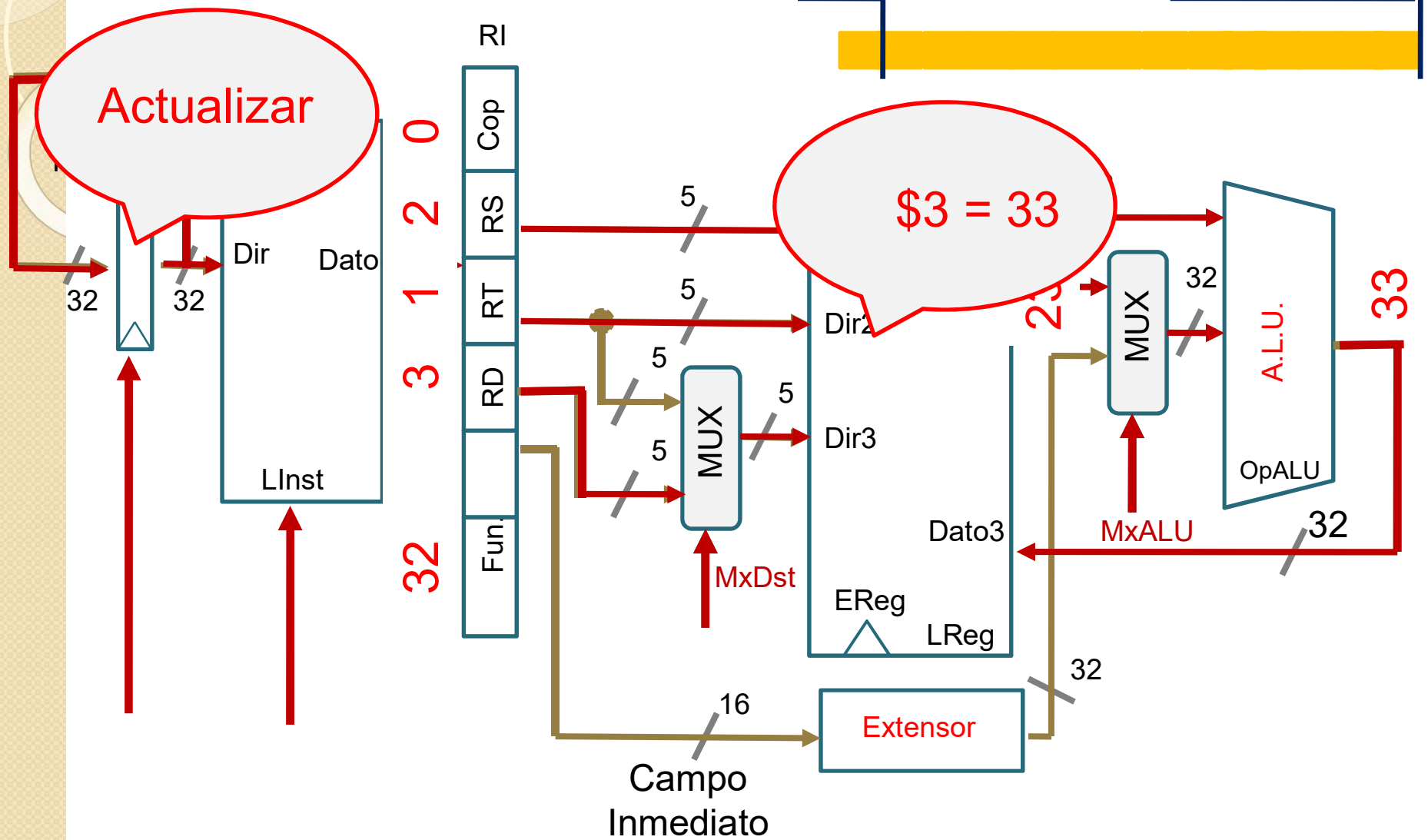




# Ruta de los datos aritmético/lógicas tipo R + addi/slti



Add \$3,\$2,\$1 # 0x00811820

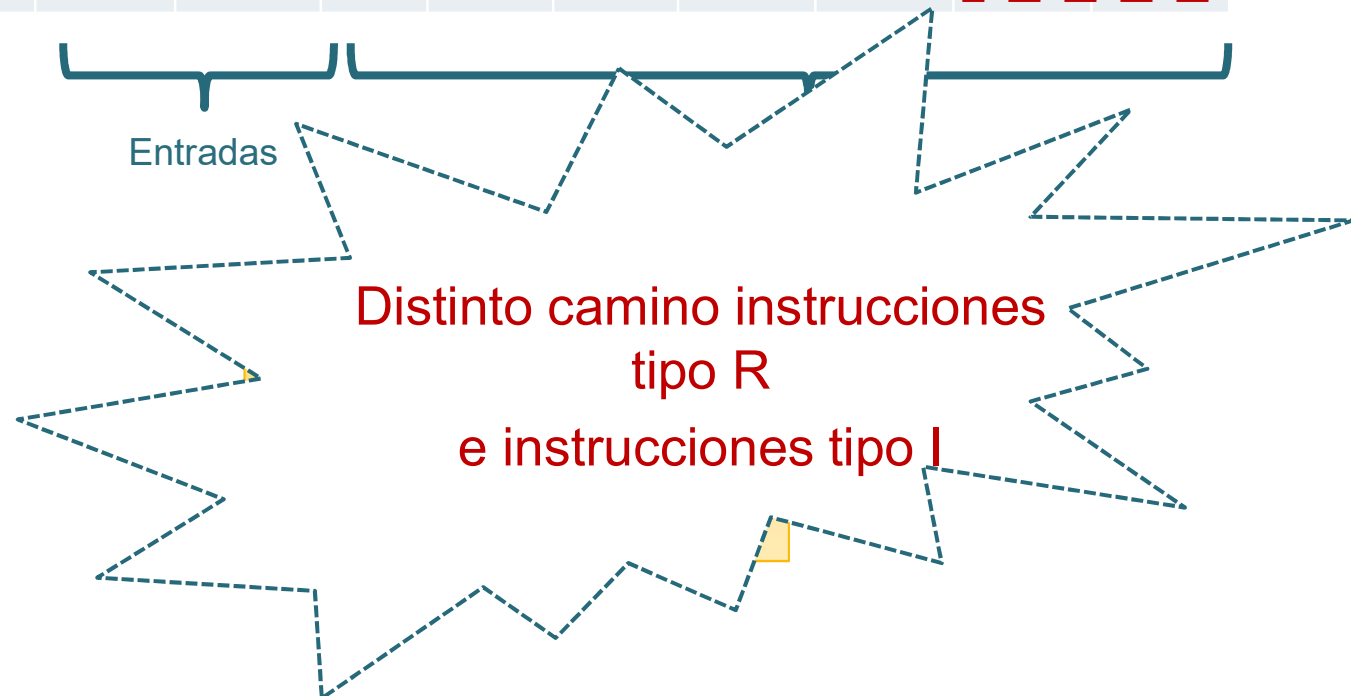






## Aritméticas Tipo I: Señales de control

			CP	Mem. Instr.	Banco Registros		ALU	Multiplexore	
Instrucción	Cop	Función	PCin	LMem	LReg	EReg	OpALU	MxALU	MxDst
add rd, rs, rt	000000	100000	1	1	1	1	010	0	1
addi rt, rs, imdto16	001000	-----	1	1	1	1	010	1	0
slti rt, rs, inmdto16	001010	-----	1	1	1	1	111	1	0



# Contenido y Bibliografía

- Introducción
- I – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

lw rt, rs, inmdto l6 # lw \$3, 0x100(\$2)

COP	RS	RT	desplazamiento
100011	rs	rt	Desplazamiento 16

? • ¿Cómo se ejecuta esta instrucción?

$R_t \leftarrow \text{Mem}[R_s + \text{desplazamiento l6 extendido}]$

? • Elementos funcionales necesarios

? • Ruta de los datos

? • Señales de control



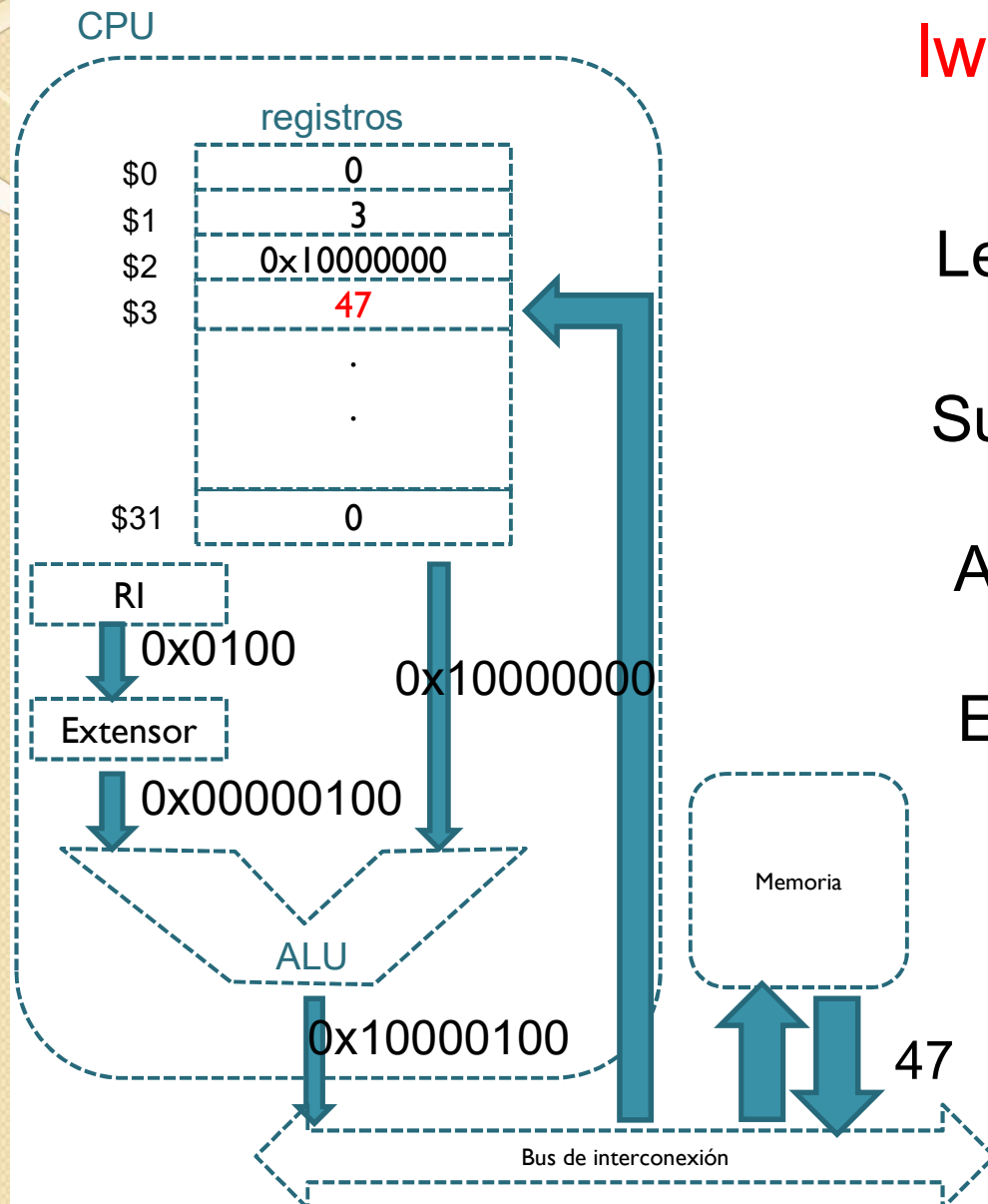
# Instrucción lw: Señales de control

				Reg. CP	Mem. Instr.	Banco Registros		ALU	Memori a DATOS	Multiplexores		
Instrucción	Form	Código Op.	Función	PCIn	LInst	LReg	EReg	OpALU	LMem	MxALU	MxDst	MxER
add rd, rs, rt	R	000000	100000	1	1	1	1	010	0	0	1	0
addi rt, rs, immdto16	I	001000		1	1	1	1	010	0	1	0	0
lw rt, desp(rs)	I	100011		1	1	1	1	010	1	1	0	1

Entradas
Salidas

OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

# Arquitectura MIPS32. Op. Carga tipo I



`lw $3, 0x100($2)`

Lectura registro \$2 y dato

Suma y obtención dirección

Acceso a Memoria

Escritura resultado en \$3

## Añadir a la ruta lw rt, desplazl 6(rs)

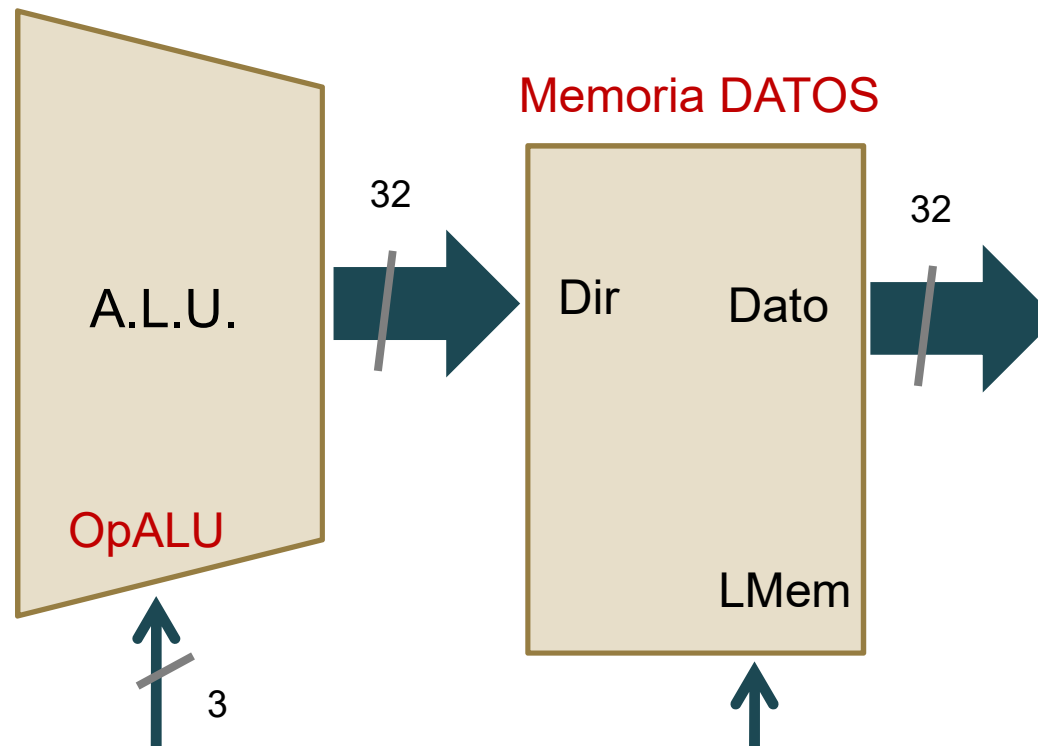
### OPERACIONES:

- Leer rs, desplazl 6
- Extender inmediato l 6 a 32b
- Calcular dirección
- Acceder a memoria
- Guardar en rt dato accedido

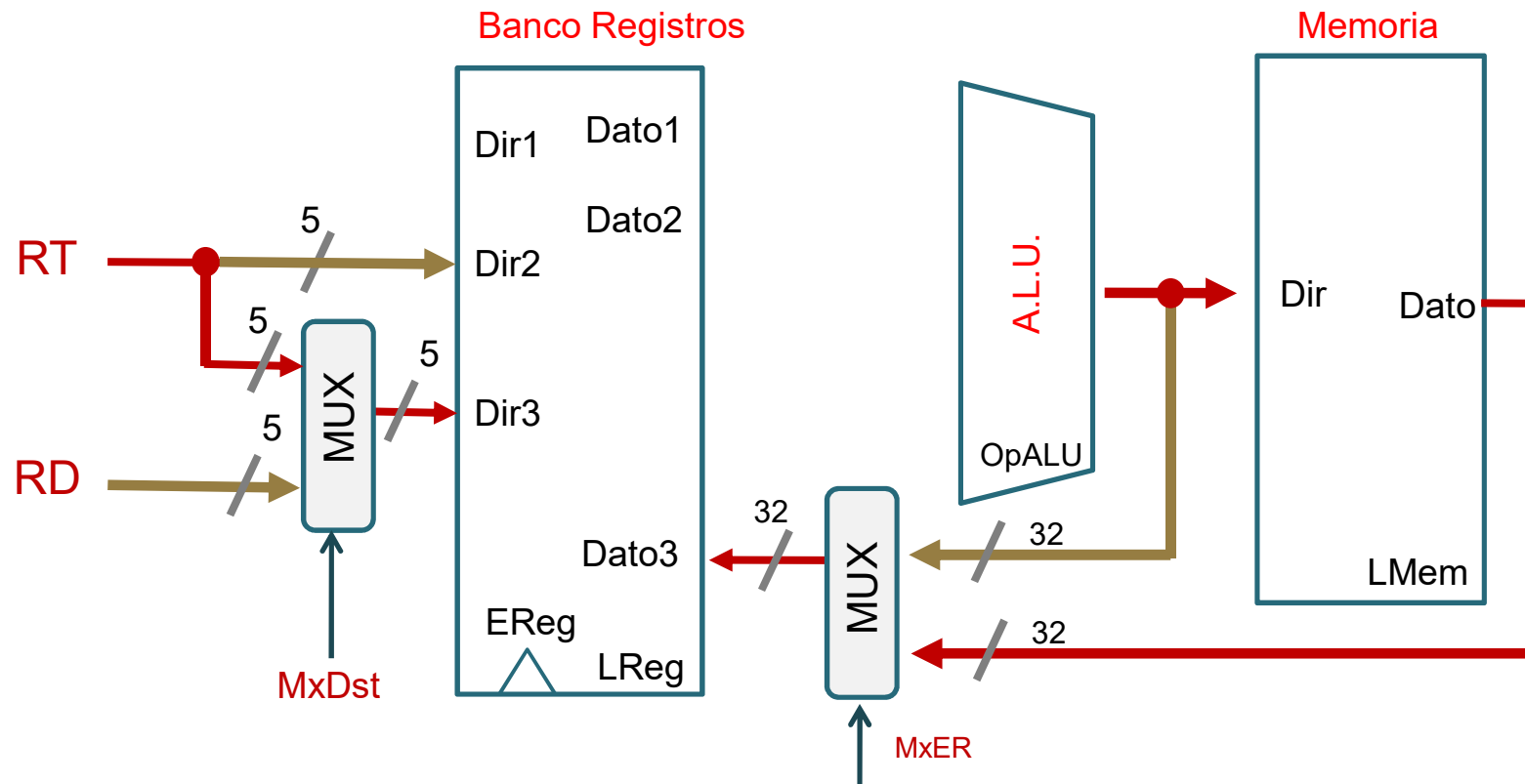
### ELEMENTOS:

- Banco de registros  
(el mismo ruta tipo R)
- Unidad extensora del bit de signo  
(la misma ruta tipo l)
- UAL empleando camino ruta tipo l
- Memoria datos
- Multiplexores en entrada del Banco de registros.

# Acceso a la memoria DATOS



# Escritura en Rt del Dato de memoria

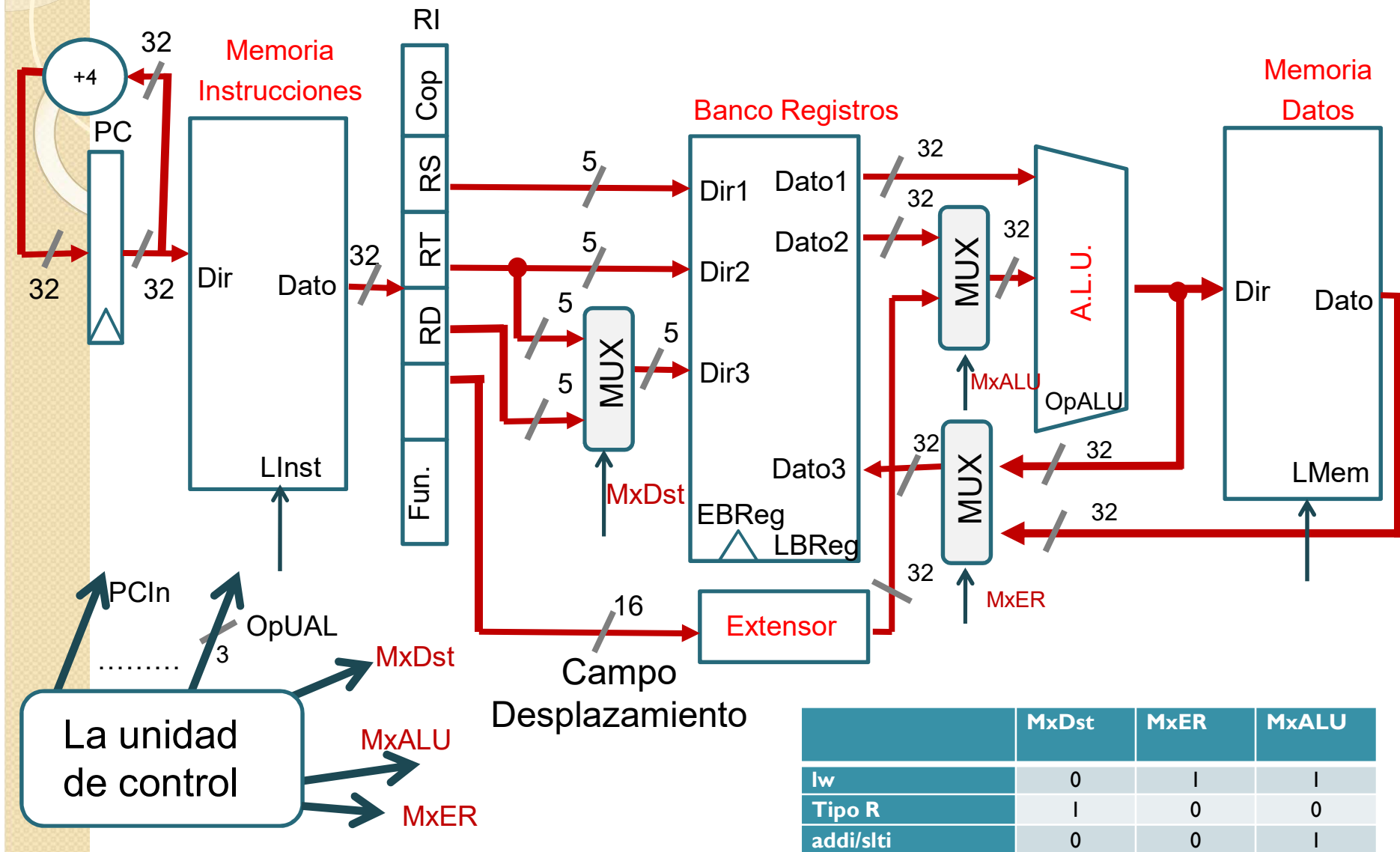


Tipo I (addi):  $MxDst = 0$   $MxER = 0$

Tipo I (lw):  $MxDst = 0$   $MxER = 1$

Tipo R:  $MxDst = 1$   $MxER = 0$

## Ruta de los datos arit./lóg. R, addi/slti y lw



sw rt, rs, inmdto l 6                      #    sw \$3, 0x100(\$2)

COP	RS	RT	desplazamiento
011011	rs	rt	Desplazamiento 16

? • ¿Cómo se ejecuta esta instrucción?

$R_t \rightarrow \text{Mem}[R_s + \text{desplazamiento l 6 extendido}]$

? • Elementos funcionales necesarios

? • Ruta de los datos

? • Señales de control

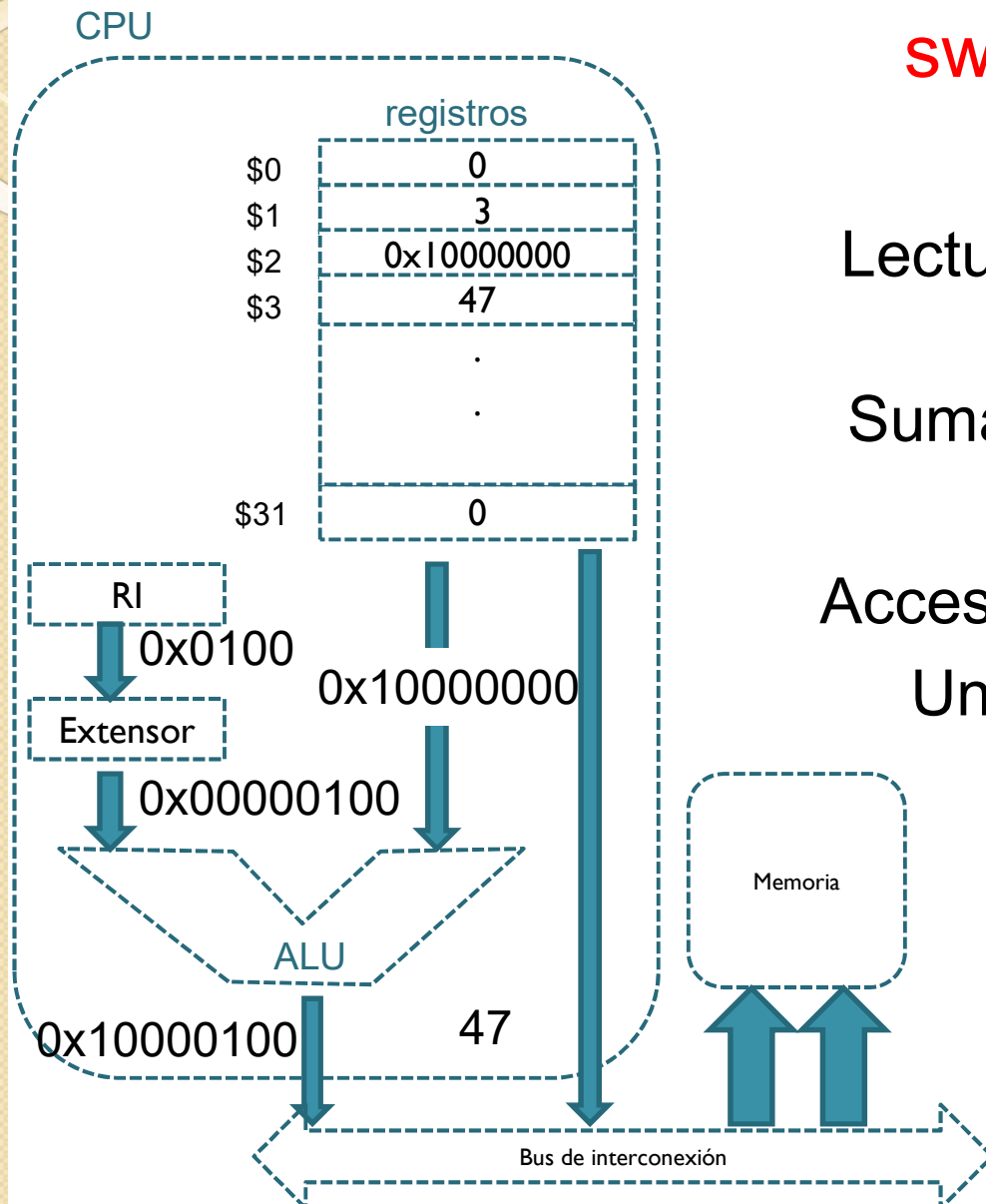
# Arquitectura MIPS32. Op.Almacenamiento

**sw \$3, 0x100(\$2)**

Lectura registro \$2,\$3 y dato

Suma y obtención dirección

Acceso a Memoria para escribir  
Un 47 en dir 0x10000100





## Añadir a la ruta sw rt, desplaz 6(rs)

### OPERACIONES:

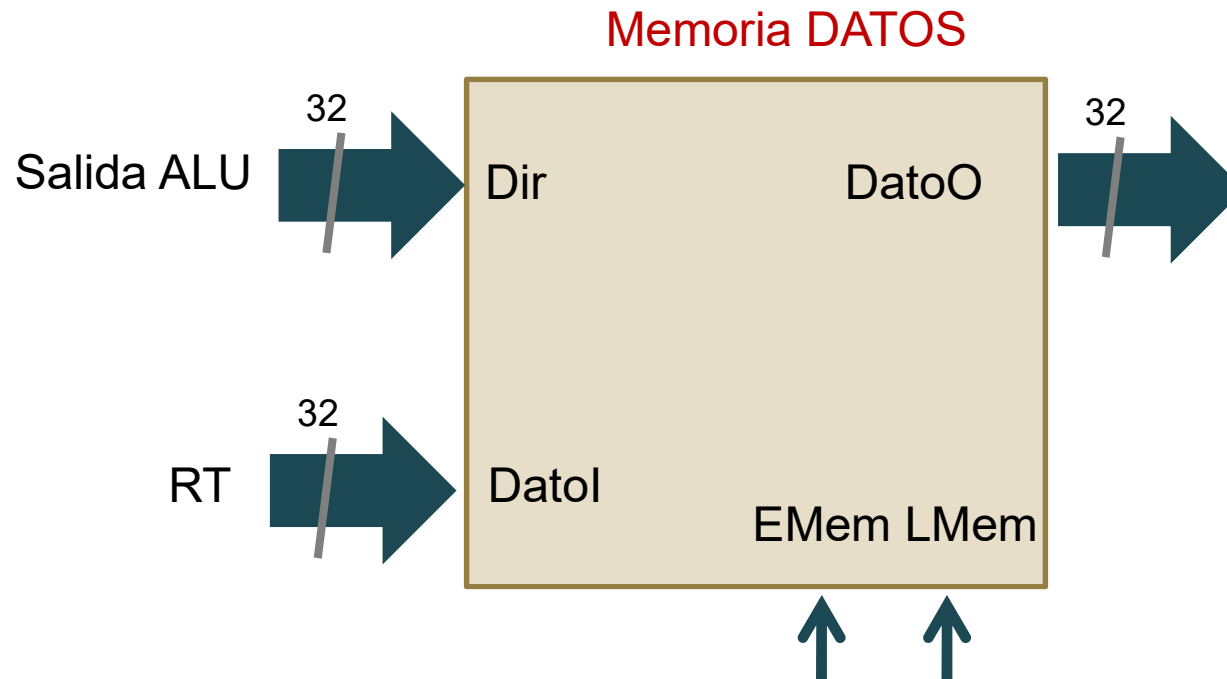
- Leer rs, rt y desplaz 6
- Extender inmediato 16 a 32b
- Calcular dirección
- Escribir en memoria el registro rt

### ELEMENTOS:

- Banco de registros  
(el mismo ruta tipo R)
- Unidad extensora del bit de signo  
(la misma ruta tipo I)
- UAL empleando camino ruta tipo I
- Memoria datos para escribir registro rt



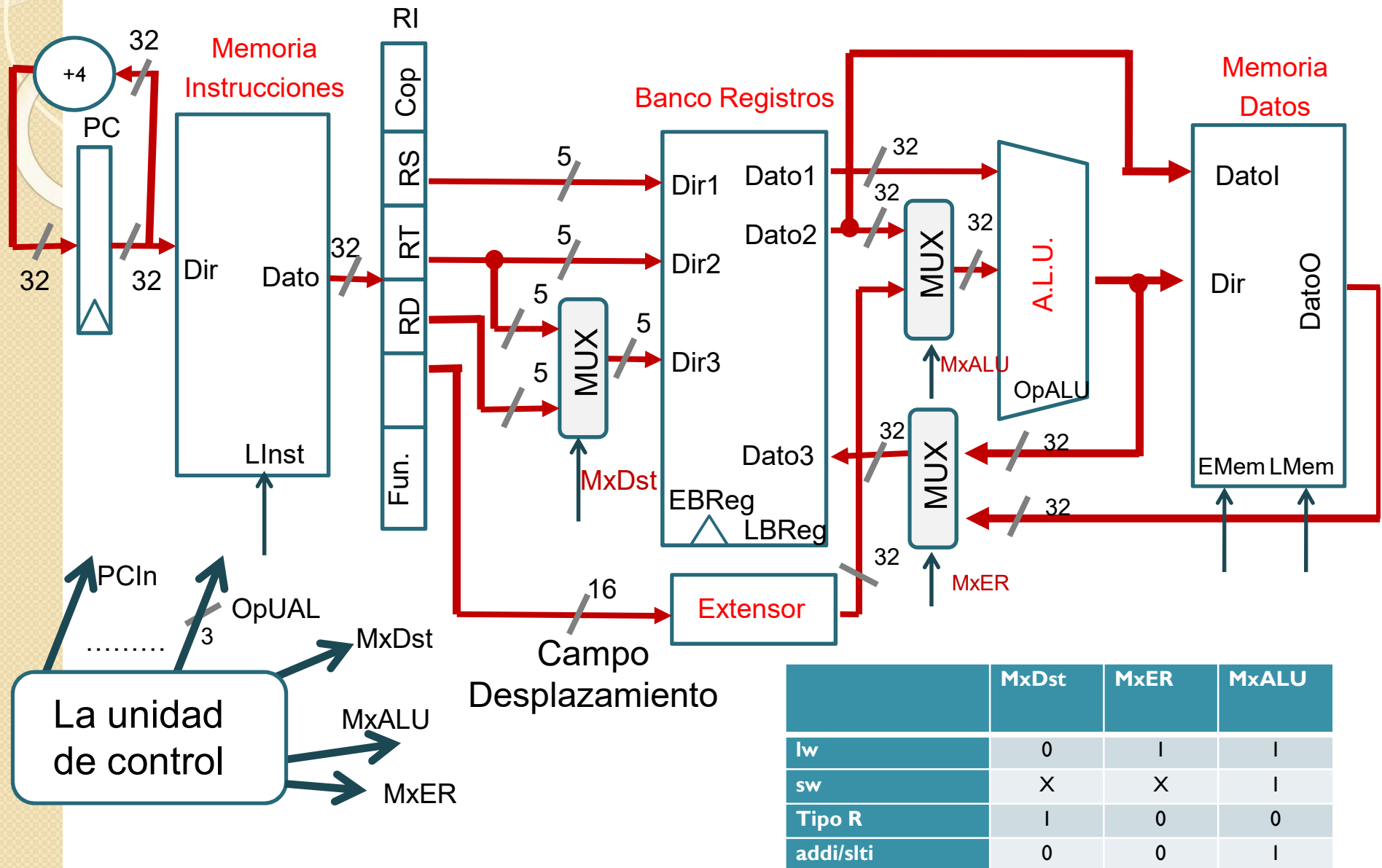
# Acceso a la memoria DATOS



**Leer:** Activar LMem y al cabo de un tiempo sale dato indicado en Dir por DatoO

**Escribir:** Activar EMem y al cabo de un tiempo se escribe el dato entrante por DatoI en la dirección indicada en Dir.

# Ruta de los datos aritm/lóg R, addi/slti, y lw/sw



# Instrucción sw: Señales de control

				Reg. CP	Mem. Instr.	Banco Registros		ALU	Memoria DATOS		Multiplexores		
Instrucción	Form	Código Op.	Función	PCin	LInst	LReg	EReg	OpALU	LMem	EMem	MxALU	MxDst	MxER
add rd, rs, rt	R	000000	100000	1	1	1	1	010	0	0	0	1	0
addi rt, rs, inmdto16	I	001000		1	1	1	1	010	0	0	1	0	0
lw rt, desp(rs)	I	100011		1	1	1	1	010	1	0	1	0	1
sw rt, desp(rs)	I	101011		1	1	1	0	010	0	1	1	x	x

Entradas

Salidas

OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

# Contenido y Bibliografía

- Introducción
- I – Arquitectura MIPS32
  - ✓ Características básicas
  - ✓ Ejemplo de ejecución
- 2 – La ruta de datos y la unidad de control
  - ✓ Etapas de búsqueda y decodificación
  - ✓ Diseño de la ruta para aritmético/lógicas de tipo R
  - ✓ Diseño de la ruta para aritmético/lógicas R y aritméticas tipo I
  - ✓ Diseño de la ruta para instrucciones lw/sw
  - ✓ Diseño de la ruta para instrucciones de salto beq/bne

---

**Bibliografía:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4ª edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

beq rt, rs, despl16      #      beq \$3, \$2,etiqueta

COP	RS	RT	desplazamiento
000100	00010	00011	Despl 16

? • ¿Cómo se ejecuta esta instrucción?

Si (rs = rt) Entonces

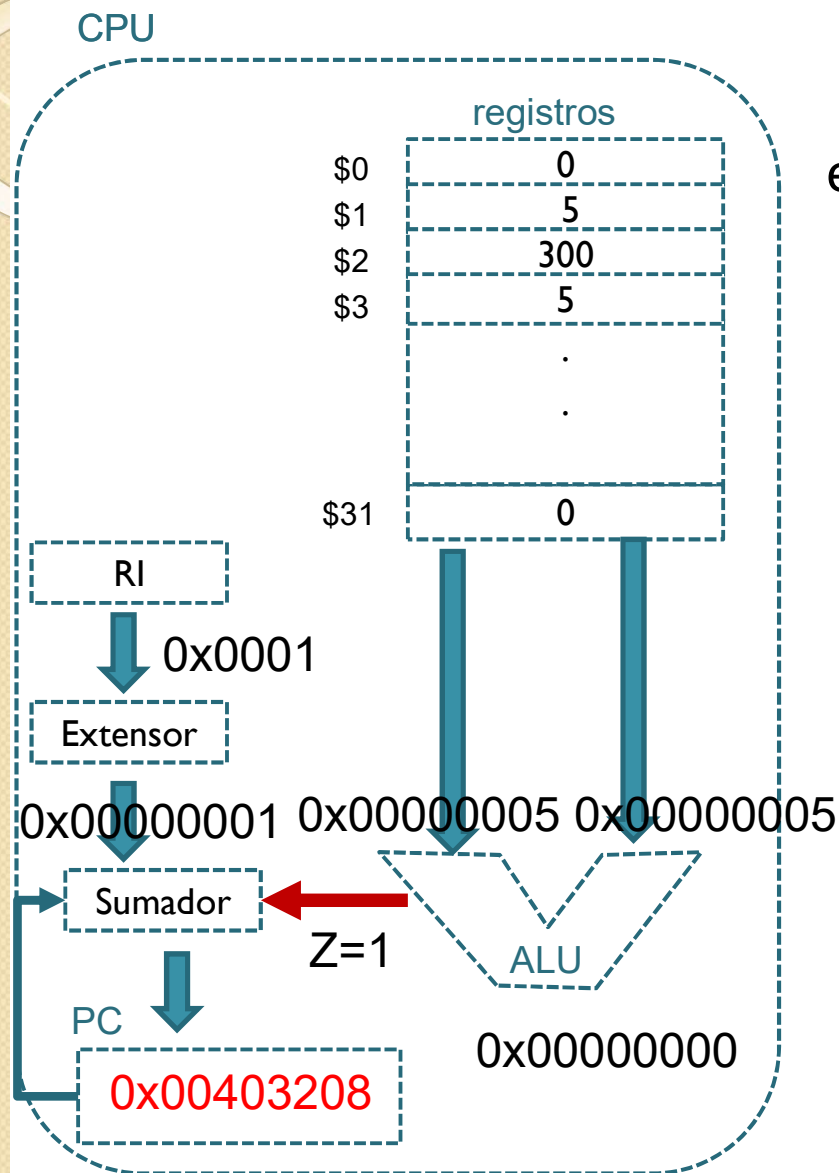
$$PC \leftarrow (Despl16 * 4) + (PC + 4)$$

? • Elementos funcionales necesarios

? • Ruta de los datos

? • Señales de control

# Arquitectura MIPS32. Op. aritmética tipo I



beq \$3, \$1, etiqueta

add \$2, \$1, \$3

etiqueta: slti \$4, \$3, 12

Lectura registros \$1, \$3 y dato

Comparación ¿\$3 = \$1 ?

Si resultado CIERTO:

$PC = (PC + 4) + (dato * 4) = \text{etiqueta}$

Sino no hacer nada

# Añadir a la ruta beq rs,rt, desplazl 6

## OPERACIONES:

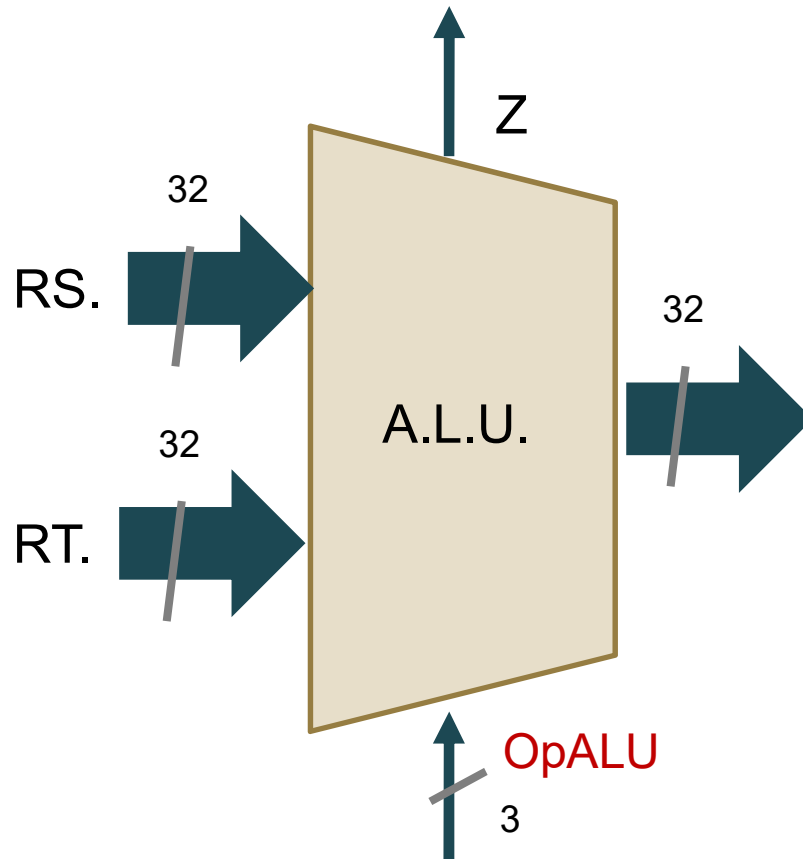
- Leer rs, rt y desplazl 6
- Extender inmediato l 6 a 32b
- Comparar Rs y Rt
- Calcular dirección
- Si Rs=Rt Entonces  
Modificar PC

## ELEMENTOS:

- Banco de registros  
(el mismo ruta tipo R)
- Unidad extensora del bit de signo  
(la misma ruta tipo l)
- UAL con bit Z  
(operación RESTA)
- Unidad para calcular
- Mux entrada de PC



# Comparación de Rs y Rt



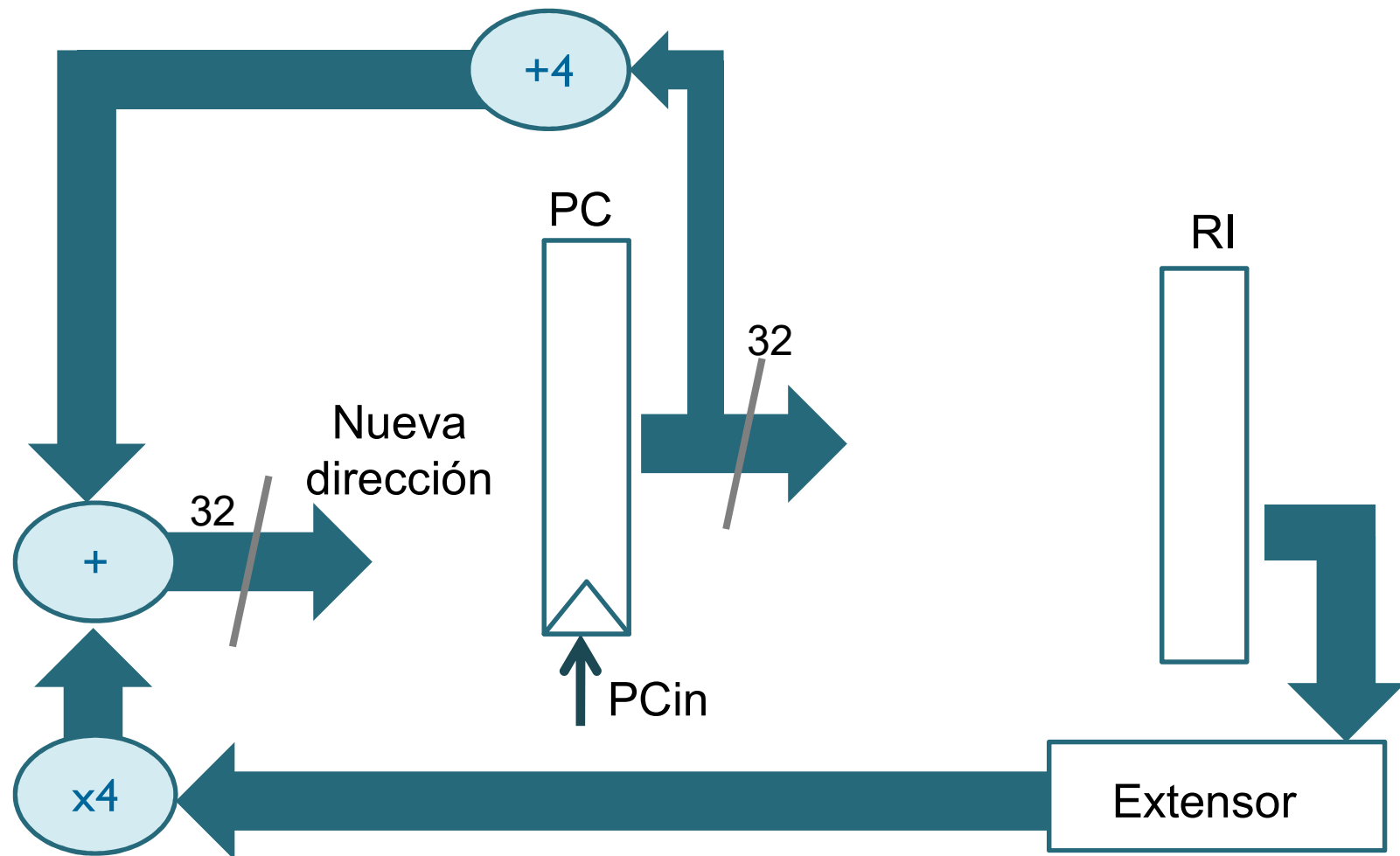
El bit Z es un bit indicador de resultado igual a cero.

Se activa a "1" cuando el resultado de la operación matemática es cero..

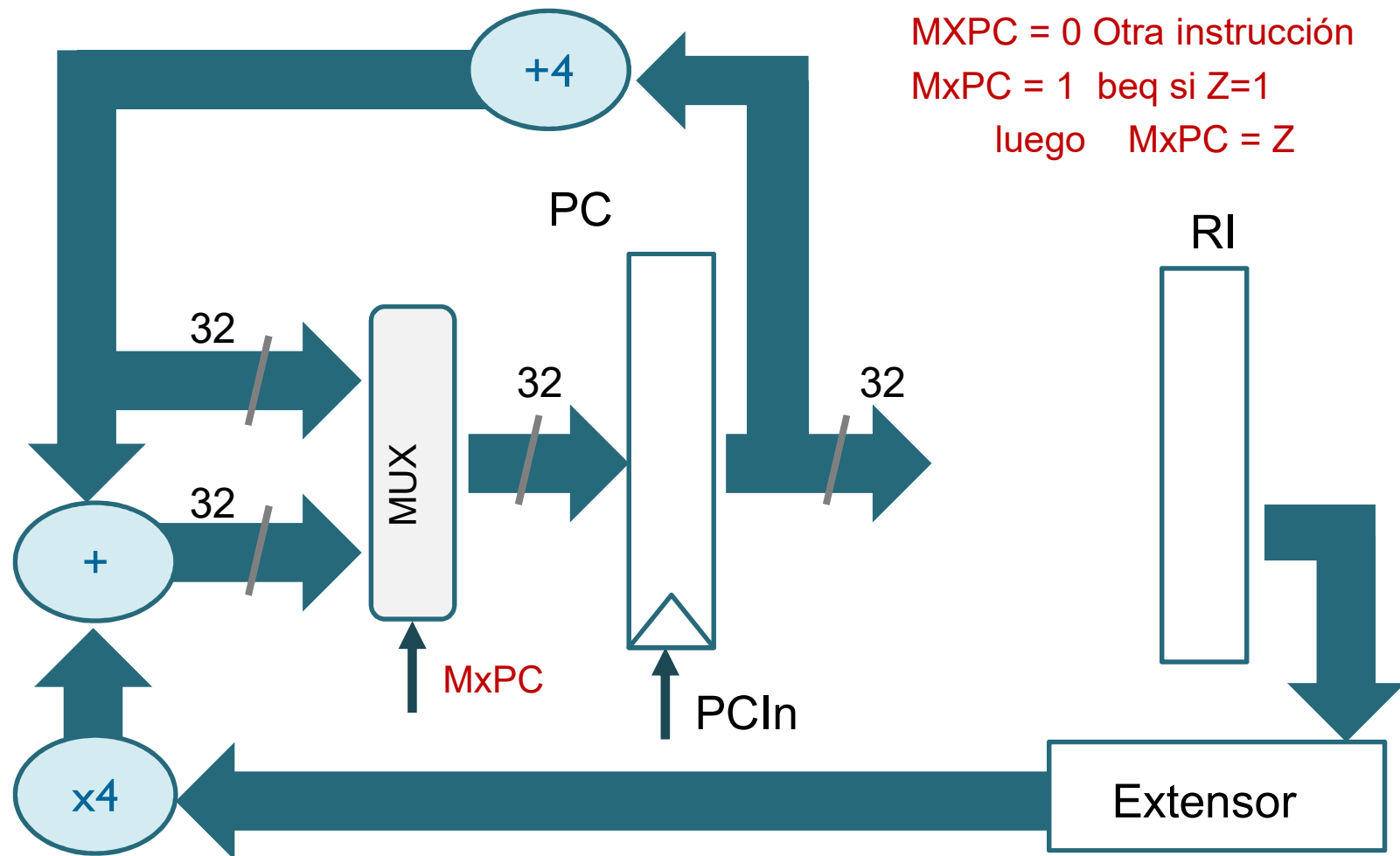
Comparar los valores de Rs y Rt consistirá en restar ambos y ver si Z se activa.

El bit Z será una nueva entrada de la unidad de control

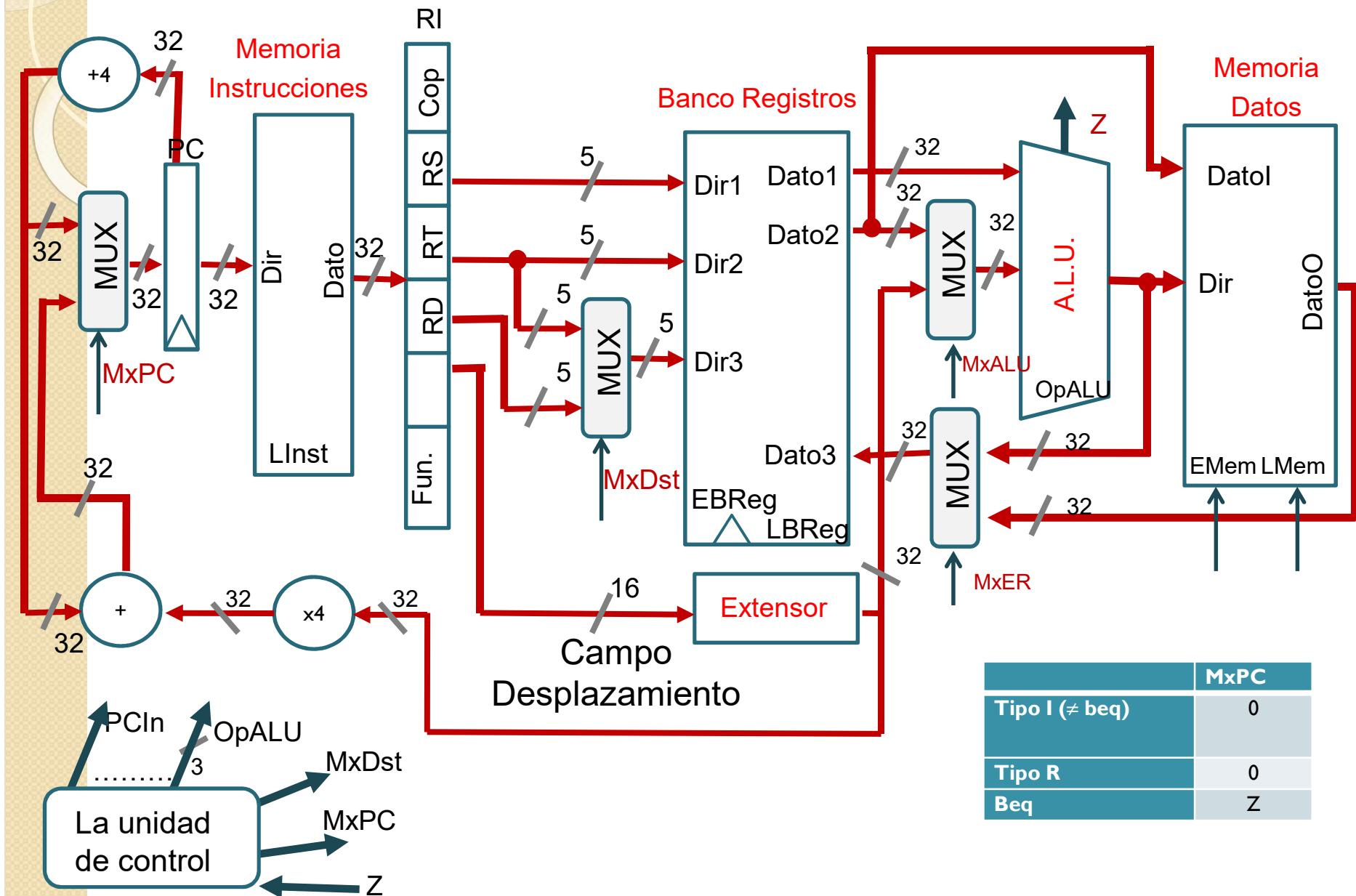
# Calcular dirección de salto



# Multiplexor de entrada



# Ruta de los datos arit./lóg. R, addi/slti, y lw/sw y beq



# Instrucción beq : Señales de control

				CP	Mem. Instr.	Banco Registros		ALU	Memoria DATOS		Multiplexores			
Instruc.	Form	Cop	Función	PCin	LInst	LReg	EReg	OpALU	LMem	EMem	MxALU	MxDst	MxER	MxPC
add	R	000000	100000	1	1	1	1	010	0	0	0	1	0	0
addi	I	001100		1	1	1	1	010	0	0	1	0	0	0
lw	I	100011		1	1	1	1	010	1	0	1	0	1	0
sw	I	101011		1	1	1	0	010	0	1	1	X	X	0
beq	I	000100		1	1	1	0	110	0	0	0	X	X	Z

Entradas

Salidas

OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

j displ26                      #                      j etiqueta

<small>COP</small>	<small>desplazamiento</small>
000010	Desplazamiento 26

- ¿Cómo se ejecuta esta instrucción?

$$PC_{27..0} \leftarrow (\text{Despl26} * 4)$$

- Elementos funcionales necesarios

- Ruta de los datos

- Señales de control

# Arquitectura MIPS32. Op. aritmética tipo J

j etiqueta

000010

01000101011001110000000000

Como PC tiene 32, se sustituyen los 28 bits de menos peso así:

$$PC = PC_{31..28} \parallel \text{dato} \parallel 00 = \text{etiqueta}$$


Por ejemplo si Etiqueta = 0x04567000=  
0000-0100-0101-0110-0111-0000-0000-00 00

## Añadir a la ruta j displ26

### OPERACIONES:

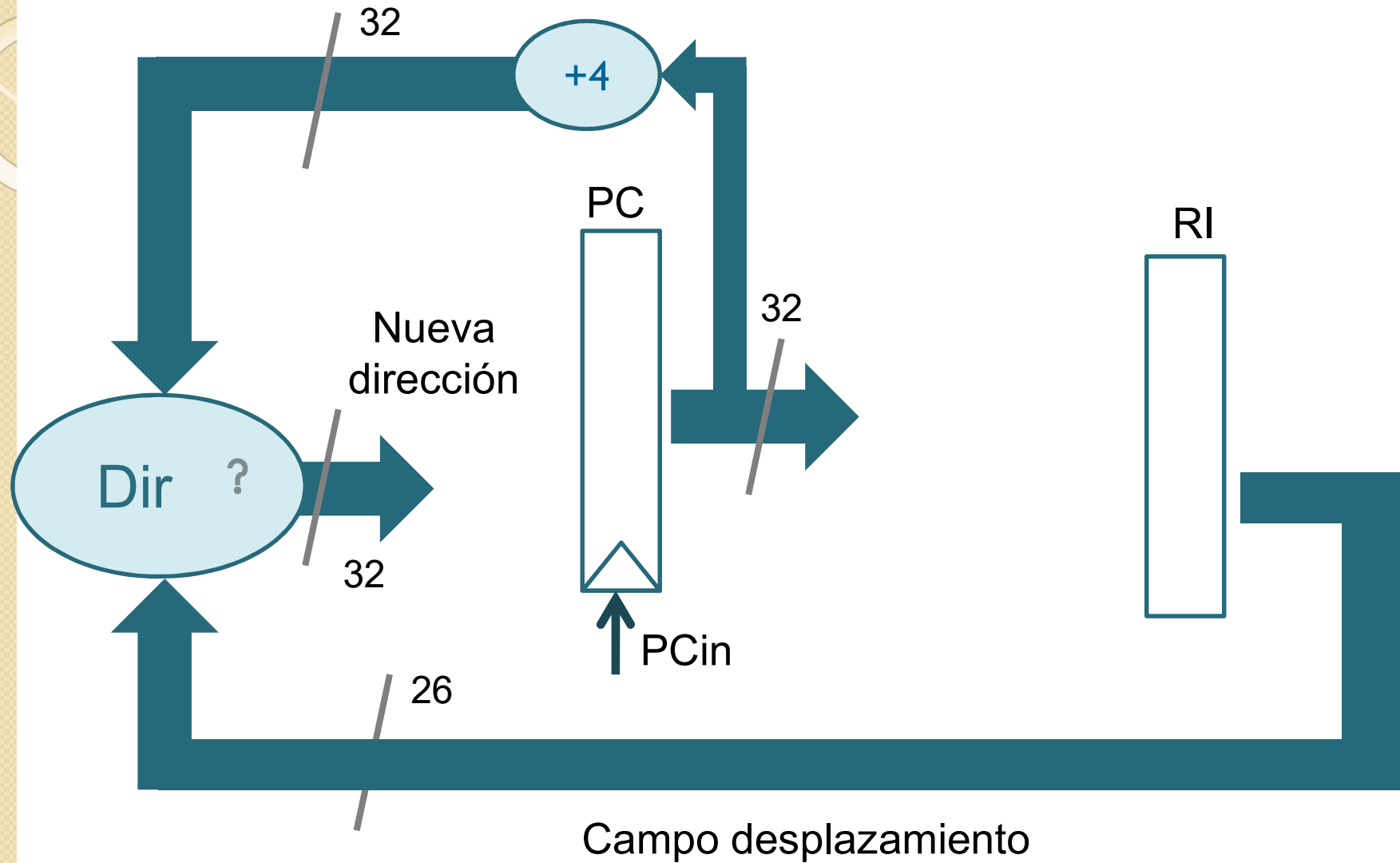
- Calcular dirección de salto
- Escribirla en PC

### ELEMENTOS:

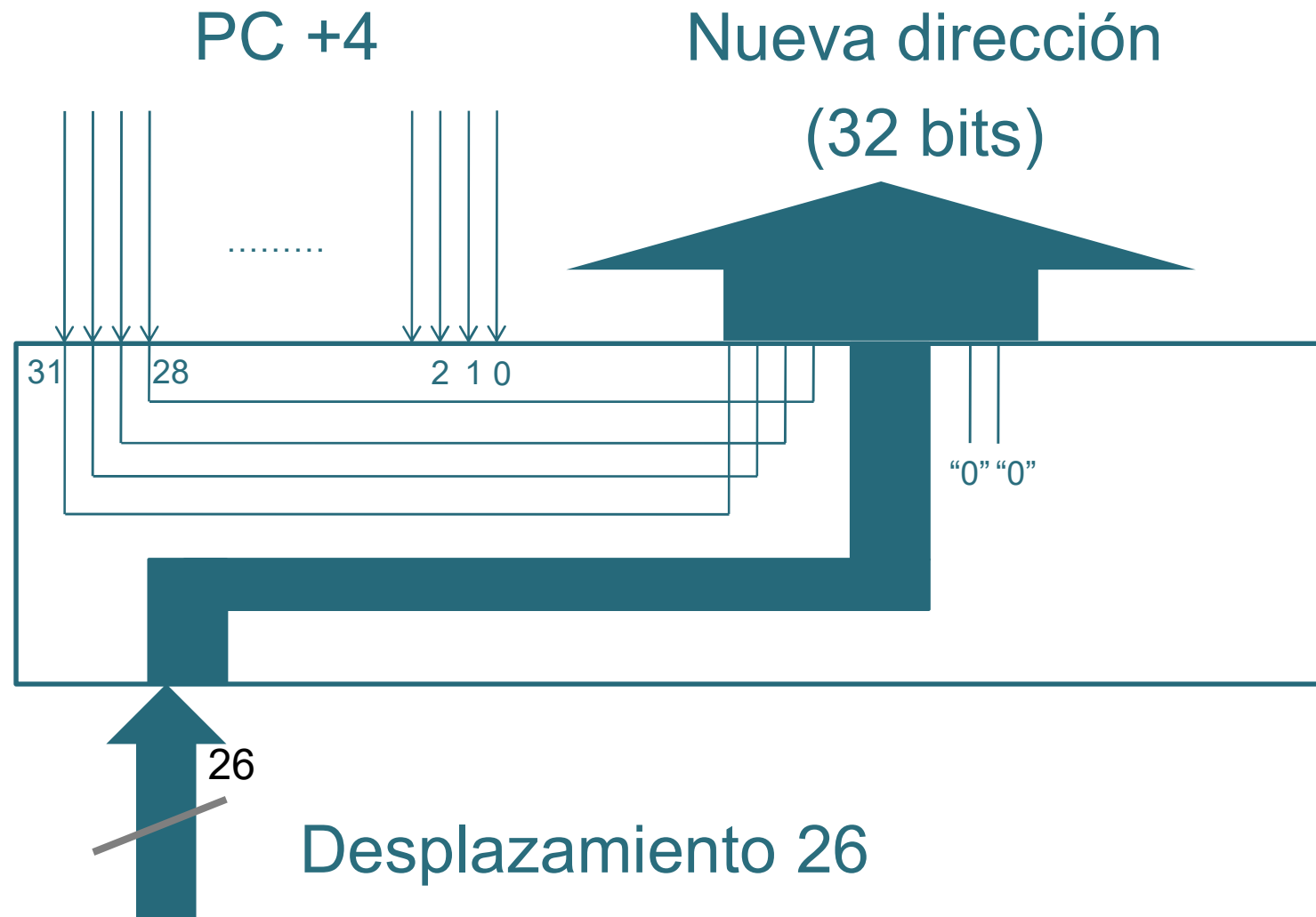
-  Unidad que componga la dirección empleando el RI y algunos bits de PC
- Modificar el Mux de PC



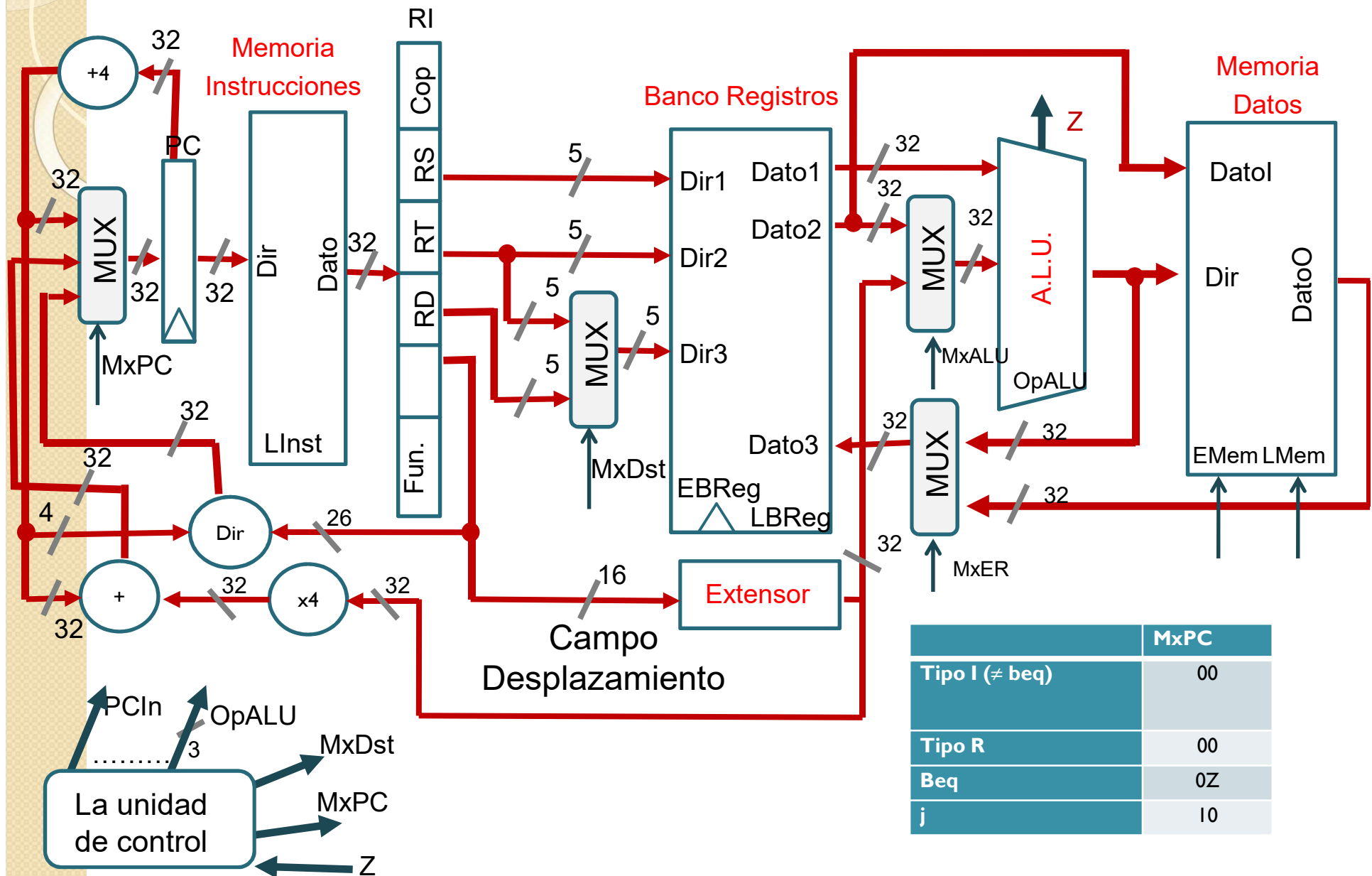
# Calcular dirección de salto



# Elemento que calcula dirección



# Ruta de los datos completa



# Instrucción j: Señales de control

				CP	Mem. Instr.	Banco Registros		ALU	Memoria DATOS		Multiplexores			
Instruc.	For m	Cop	Función	EPC	LInst	LReg	EReg	OpALU	LMem	EMem	MxALU	MxDst	MxER	MxPC
add	R	000000	100000	1	1	1	1	010	0	0	0	1	0	00
addi	I	001000		1	1	1	1	010	0	0	1	0	0	00
lw	I	100011		1	1	1	1	010	1	0	1	0	1	00
sw	I	101011		1	1	1	0	010	0	1	1	X	X	00
beq	I	000100		1	1	1	0	110	0	0	0	X	X	0Z
j	J	000010		1	1	X	0	XXX	0	0	X	X	X	10

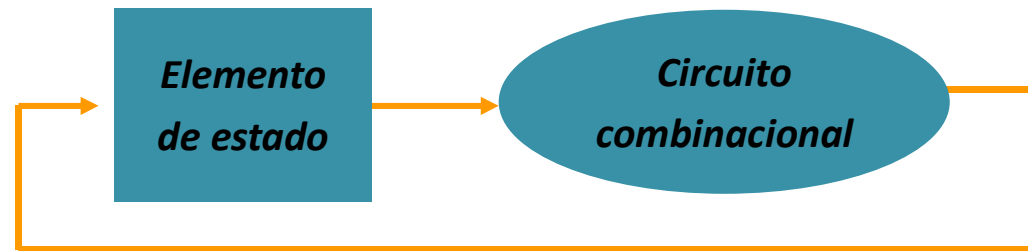
Entradas

Salidas

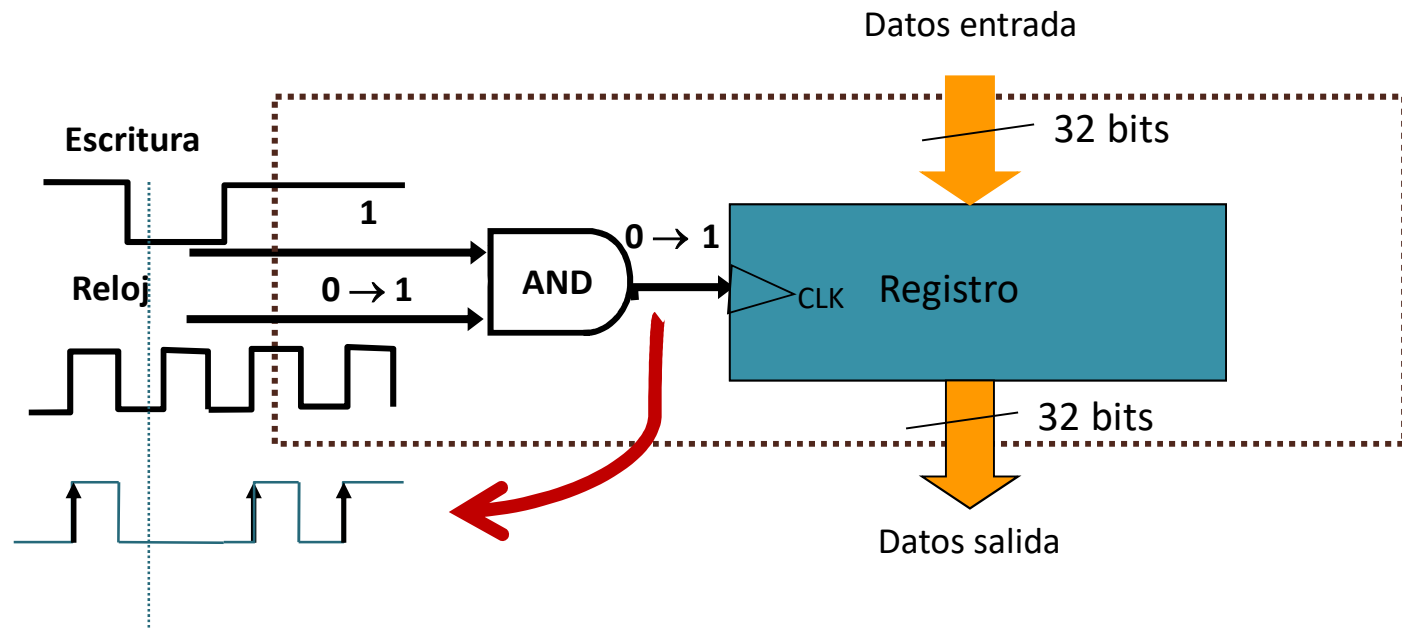
OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

# ¿Cómo generar flancos en la señal?

- Un elemento puede ser leído y escrito en el mismo ciclo de reloj



- Ejemplo de funcionamiento



# Unidad de Control: Señales de control

				Reg. CP	Mem. Instr.	Banco Registros		ALU	Mem. Datos		Multiplexores Configuración Ruta de Datos			
Instrucción	Form	Código Op.	Función	PCin	LInst	LReg	EReg	OpALU	LMem	EMem	MxPC	MxALU	MxDst	MxER
add rd, rs, rt	R	000000	100000	1	1	1	1	010	0	0	00	0	1	0
sub rd, rs, rt	R	000000	100010	1	1	1	1	110	0	0	00	0	1	0
and rd, rs, rt	R	000000	100100	1	1	1	1	000	0	0	00	0	1	0
or rd, rs, rt	R	000000	100101	1	1	1	1	001	0	0	00	0	1	0
lw rt, desp(rs)	I	100011		1	1	1	1	010	1	0	00	1	0	1
sw rt, desp(rs)	I	101011		1	1	1	0	010	0	1	00	1	X	X
beq rs, rs, etiq	I	000100		1	1	1	0	110	0	0	Z0	0	X	X
j	J	000010		1	1	X	0	XXX	0	0	01	X	X	X

Entradas



Constantes

Salidas

OpALU	Operación
000	$a \wedge b$ (and)
001	$a \vee b$ (or)
010	$a + b$ (suma aritmética)
110	$a - b$ (resta)
111	$a < b$ (menor que)

La implementación de esta tabla permite la correcta ejecución de las instrucciones en un ciclo de reloj



# Estructura de Computadores

Grado de Ingeniería Informática  
ETSINF

## Tema I: El procesador

DISCA