

TEMA 2.3

Programación Máquina III: Procedimientos

■ Procedimientos

- Mecanismos
- Estructura de la pila
- Convenciones de llamada
 - Pasando el control
 - Pasando los datos
 - Gestionando datos locales
- Ejemplos ilustrativos de Recursividad

Función Recursiva

```
/* Recursive popcount */  
long pcount_r(unsigned long x) {  
    if (x == 0)  
        return 0;  
    else  
        return (x & 1)  
            + pcount_r(x >> 1);  
}
```

pcount_r:

```
    movl    $0, %eax  
    testq   %rdi, %rdi  
    je      .L6  
    pushq   %rbx  
    movq   %rdi, %rbx  
    andl   $1, %ebx  
    shrq   %rdi  
    call   pcount_r  
    addq   %rbx, %rax  
    popq   %rbx
```

.L6:

```
    rep; ret
```

Condición Terminación Función Recursiva

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

`pcount_r:`

<code>movl</code>	<code>\$0, %eax</code>
<code>testq</code>	<code>%rdi, %rdi</code>
<code>je</code>	<code>.L6</code>
<code>pushq</code>	<code>%rbx</code>
<code>movq</code>	<code>%rdi, %rbx</code>
<code>andl</code>	<code>\$1, %ebx</code>
<code>shrq</code>	<code>%rdi</code>
<code>call</code>	<code>pcount_r</code>
<code>addq</code>	<code>%rbx, %rax</code>
<code>popq</code>	<code>%rbx</code>

`.L6:`

`rep; ret`

Registro	Uso(s)	Tipo
<code>%rdi</code>	<code>x</code>	Salva-invocante 
<code>%rax</code>	Valor de retorno	Salva-invocante 

=> Registro que se usa para guardar datos de forma temporal y que no necesitan ser preservados entre llamadas a funciones.

Preservar Registro para Llamada Recursiva

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

Registro	Uso(s)	Tipo
%rbx	x & 1	Salva-invocado

↓

Registro que se usa para mantener datos que perduran en el tiempo, que deben ser procesados entre llamadas al sistema.
Se preserva el registro para llamada recursiva.

pcount_r:

movl	\$0, %eax
testq	%rdi, %rdi
je	.L6
pushq	%rbx
movq	%rdi, %rbx
andl	\$1, %ebx
shrq	%rdi
call	pcount_r
addq	%rbx, %rax
popq	%rbx

.L6:

rep; ret



Preparar Llamada Función Recursiva

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

`pcount_r:`

<code>movl</code>	<code>\$0, %eax</code>
<code>testq</code>	<code>%rdi, %rdi</code>
<code>je</code>	<code>.L6</code>
<code>pushq</code>	<code>%rbx</code>
<code>movq</code>	<code>%rdi, %rbx</code>
<code>andl</code>	<code>\$1, %ebx</code>
<code>shrq</code>	<code>%rdi</code>
<code>call</code>	<code>pcount_r</code>
<code>addq</code>	<code>%rbx, %rax</code>
<code>popq</code>	<code>%rbx</code>

`.L6:`

`rep; ret`

Registro	Uso(s)	Tipo
<code>%rbx</code>	<code>x & 1</code>	Salva-invocado 
<code>%rdi</code>	<code>x >> 1</code> Argumento recurs.	Salva-invocante 

Llamada Función Recursiva

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

pcount_r:

```
    movl    $0, %eax
    testq   %rdi, %rdi
    je      .L6
    pushq   %rbx
    movq   %rdi, %rbx
    andl   $1, %ebx
    shrq   %rdi
    call   pcount_r
    addq   %rbx, %rax
    popq   %rbx
```

.L6:

rep; ret

Registro	Uso(s)	Tipo
%rbx	x & 1	Salva-invocado 
%rax	Valor de retorno de llamada recursiva	Salva-invocante 

Resultado Función Recursiva

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

pcount_r:

```
    movl    $0, %eax
    testq   %rdi, %rdi
    je      .L6
    pushq   %rbx
    movq   %rdi, %rbx
    andl   $1, %ebx
    shrq   %rdi
    call   pcount_r
    addq   %rbx, %rax
    popq   %rbx
```

.L6:

```
    rep; ret
```

Registro	Uso(s)	Tipo
%rbx	x & 1	Salva-invocado 
%rax	Valor de retorno	Salva-invocante 

Terminar Función Recursiva

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1)
            + pcount_r(x >> 1);
}
```

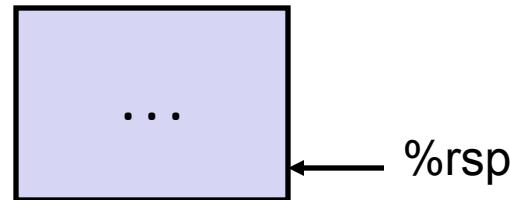
Registro	Uso(s)	Tipo
%rax	Valor de retorno	Salva-invocante 

pcount_r:

```
    movl    $0, %eax
    testq   %rdi, %rdi
    je     .L6
    pushq   %rbx
    movq    %rdi, %rbx
    andl    $1, %ebx
    shrq    %rdi
    call    pcount_r
    addq    %rbx, %rax
    popq    %rbx
```

.L6:

rep; ret



Observaciones Sobre la Recursividad

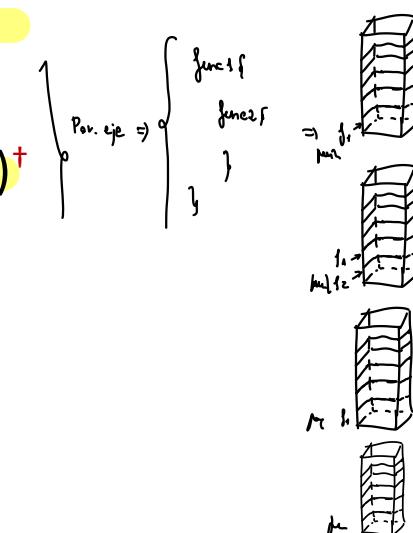
■ Manejada sin Especiales Consideraciones

- Marcos pila implican que cada llamada a función tiene almacenamiento privado
 - Variables locales y Registros preservados
 - Dirección de retorno salvada
- Convenciones preservación registros previenen que una llamada a función corrompa los datos de otra En decir, que las llamadas a función no escriben los datos de otras, a no ser que el propio código lo haga.
 - A menos que el código C explícitamente lo haga (p.ej. buffer overflow)
- Disciplina de pila sigue el patrón de llamadas / retornos
 - Si P llama a Q, entonces Q retorna antes que P
 - Primero en entrar, último en salir (Last-In, First-Out)[†]

■ También funciona con recursividad mutua[‡]

- P llama a Q; Q llama a P

 Marcos de pila \Rightarrow Cada llamada a función almacenamiento privado.



[‡] en general con reentrancia

[†] pila = lista LIFO

Resumen de Procedimientos en x86-64

■ Puntos Importantes

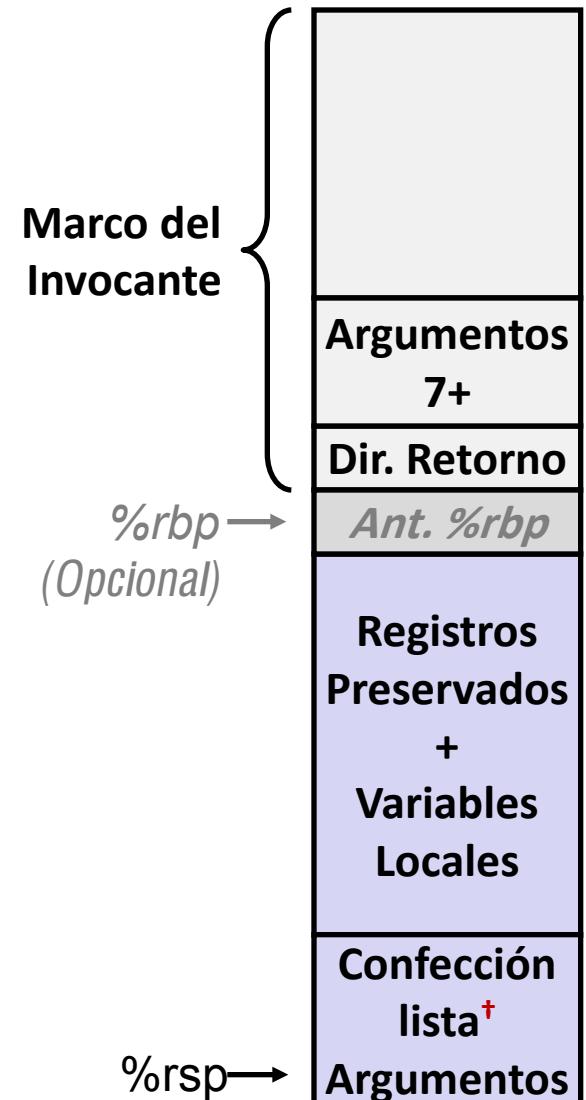
- Pila es la estructura de datos correcta para llamada / retorno procedimientos
 - P llama a Q, entonces Q retorna antes que P

■ Recursividad (y recursividad mutua) con mismas convenciones de llamada normales

- Se pueden almacenar valores tranquilamente en el marco de pila local y en registros salva-invocado
- Poner argumentos 7+ de la función en tope de pila
- Devolver resultado en %rax

■ Punteros son direcciones de valores

- Global o en pila



Para la llamada / entrada de procedimientos
la estructura de datos correcta es la Pila

En cuanto a recursividad, se siguen las mismas
convenciones de las llamadas normales.

Podemos almacenar valores en el marco de pila
local y en registros salva-llamado.

(cuando el número de argumentos que parametriza una
función excede de 7 (incluidos), estos se guardan
en la pila)

El valor de retorno de una función ; el resultado, se
almacena en %rax

TEMA 2.4

Programación Máquina IV: Datos

■ Arrays[†]

- Uni-dimensionales
- Multi-dimensionales (anidados)
- Multi-nivel

■ Estructuras

- Ubicación
- Acceso
- Alineamiento

■ Uniones

[†] Hay autores españoles que distinguen entre “vectores” (1D) y “matrices” (2D) 3

Tipos de Datos Básicos

■ Enteros

- Almacenados y manipulados en **registros (enteros)** propósito general
- Con/sin signo depende de las instrucciones usadas[†]

Intel	ASM [#]	Bytes	C
byte	b	1	[unsigned] char
word	w	2	[unsigned] short
double word	l	4	[unsigned] int
quad word	q	8	[unsigned] long int (x86-64)

■ Punto Flotante

- Almacenados y manipulados en registros punto flotante

Intel	ASM	Bytes	C
Single	s	4	float
Double	l	8	double
Extended	t	10/12/16	long double

[#] sufijos en sintaxis AT&T Linux
[†] y del tipo datos indicado en C, de los flags ó “condition codes” consultados en código ASM. 4

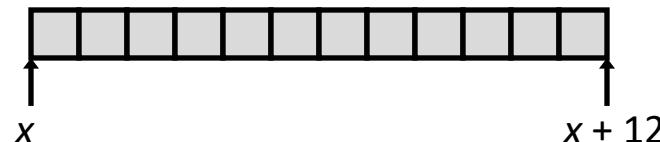
Ubicación[†] de Arrays

■ Principio Básico

T A [L];

- Array de tipo **T** y longitud **L**
- Reservada[†] región contigua en memoria de $L * \text{sizeof}(T)$ bytes

```
char string[12];
char = 1 byte  $\Rightarrow 12 \cdot 1 = 12$  pos. memoria
```

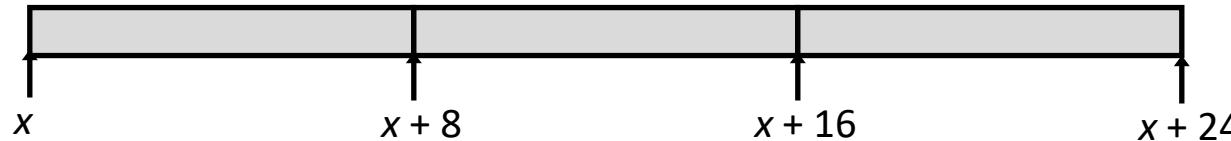


Reservamos tanto memoria como elementos tenga el array (longitud), multiplicado por el tamaño en memoria del tipo de dato del array.

```
int val[5];
int = 4 bytes  $\Rightarrow 5 \cdot 4 = 20$  pos. memoria
```



```
double a[3];
```



```
char *p[3];
```



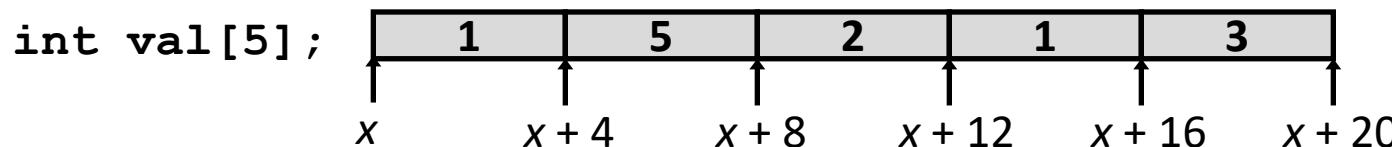
"(de)allocate" = reservar/liberar
+ "allocation" = ubicación 5

Acceso a Arrays

■ Principio Básico

T A [L] ;

- Array de tipo T y longitud L
 - El identificador **A** (Tipo T^*) puede usarse como puntero al elemento 0



■ Referencia [†]	Tipo	Valor
---------------------------	------	-------

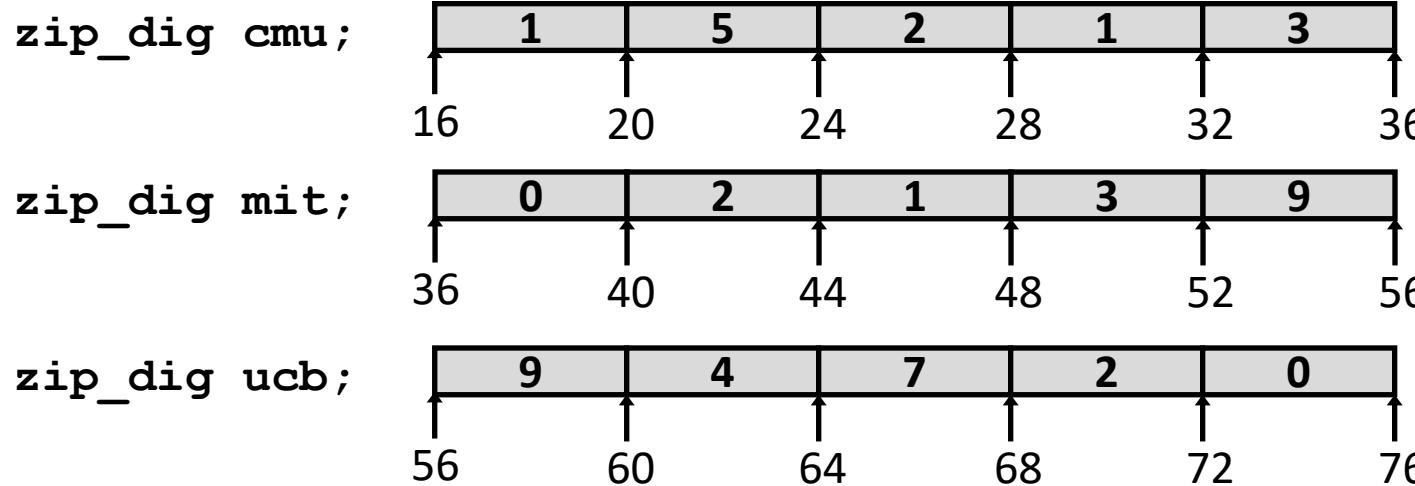
val[4]	int
val	int * \Rightarrow Dirección del elemento 0
val+1	int * \Rightarrow Dirección del elemento 1
&val[2]	int * \Rightarrow Dirección del elemento que ocupa la posición 2 en el array.
val[5]	int
*(val+1)	int
val + i	int *

[†] otros autores usan “(de)reference” en sentido mucho más estricto, para indicar el tipo puntero, o la operación de seguir el puntero. 6

Ejemplo de Arrays

```
#define ZLEN 5
typedef int zip_dig[ZLEN];

zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig mit = { 0, 2, 1, 3, 9 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```



Aquí se observa que los tres arrays están en bloques sucesivos de memoria, de 20 bytes cada uno ($5 \cdot 4 + 4 = 20$). Sin embargo, en general no sucede así, porque hay que ubicar en bloques de memoria no sucesivos.

- Declaración “`zip_dig cmu`” equivalente a “`int cmu[5]`”
- Los arrays del ejemplo fueron ubicados en bloques sucesivos de 20 bytes
 - En general no está garantizado que suceda

† ZIP = "Zone Improvement Plan"
especie de código postal en USA 7

Ejemplo de Acceso a Arrays

`zip_dig cmu;`



```
int get_digit
+ (zip_dig z, size_t digit)
{
    return z[digit];
}
```

```
get_digit:                                # z en %rdi,
+                                         # digit %rsi
    movl (%rdi,%rsi,4), %eax # z[digit]
    ret
```

- El registro **%rdi** contiene dirección inicio del array
- El registro **%rsi** contiene el índice al array
- El dígito deseado está en **4 * %rdi + %rsi**
- Usar referencia a memoria[‡] (**%rdi, %rsi, 4**)

[†] `size_t` es usualmente `unsigned long int` en `x86_64`

[‡] “memory reference” en sentido manuales
IA-32, “direcciónamiento a memoria” 8

Ejemplo de Acceso a Arrays

`zip_dig cmu;`



```
int get_digit
+ (zip_dig z, int digit)
{
    return z[digit];
}
```

Entiende que el segundo argumento es un entero, que ocupa $4 \text{ bytes} = 32 \text{ bits} \Rightarrow \%rsi$
Y muere, con extensión de signo, el contenido de $\%esi \rightarrow \%rsi$ (llenando lo que faltó con el bit de signo).

```
get_digit:           # z en %rdi,
+ movslq  %esi, %rsi      # digit=%rsi
    movl (%rdi,%rsi,4), %eax # z[digit]
    ret
```

- El registro **%rdi** contiene dirección inicio del array
- El registro **%rsi** contiene el índice al array
- El dígito deseado está en $4 * \%rdi + \%rsi$
- Usar referencia a memoria[#] (**%rdi, %rsi, 4**)

[#] "Move with Sign-extend Long to Quad", mnemotécnico MOVSX según Intel

[#] "memory reference" en sentido manuales

IA-32, "direcciónamiento a memoria" 9

Ejemplo de Bucle sobre Array

```
void zincr(zip_dig z) {
    size_t i;
    for (i = 0; i < ZLEN; i++)
        z[i]++;
}
```

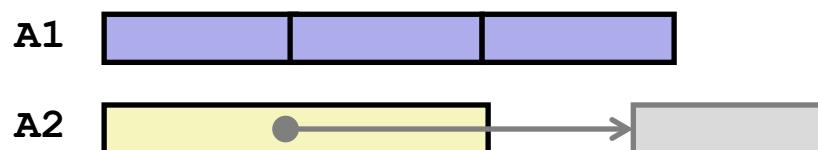
```
# %rdi = z
movl    $0, %eax
jmp     .L3
.L4:
    addl    $1, (%rdi,%rax,4)
    addq    $1, %rax
.L3:
    cmpq    $4, %rax
    jbe     .L4
rep; ret
```

{ El índice está en %eax (recordar que es un entero. El vector está en %rdi (primer argumento).
 Inicialización del índice i a cero.
 Saltar a L3: comparar el índice para ver si ha llegado al final. Si no, hacer (llegado saltar a L4:
 sumar 1 a z[i] uno; addl \$1, (%rdi,%rax,4)
 y actualización del índice : sumar 1 al rax}

Comprendiendo Arrays y Punteros #1

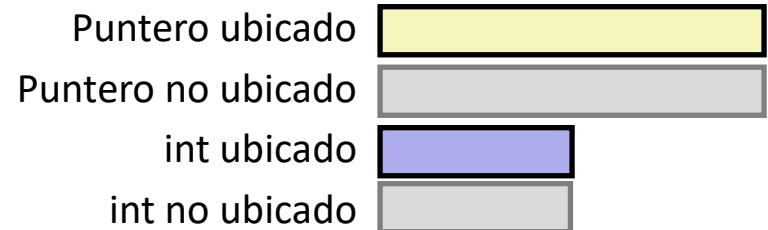
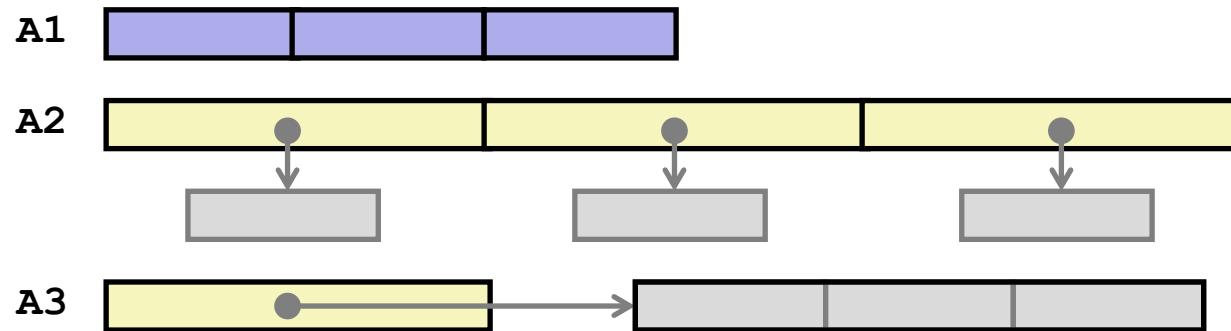
Decl	A1 , A2			*A1 , *A2		
	Comp	Ptr	Size	Comp	Ptr	Size
int A1[3]	.	.	Página tienen 3 elementos de 4 bytes cada uno : 32 - 42 12	.	.	El puntero es de tamaño de memoria y tiene dirección de memoria 4
int *A2	.	.	Un puntero es tipo que tiene dirección de memoria 8	.	S	El puntero apunta a un valor que tiene dirección 4

- Comp: Compila (S·/N)
- Ptr: Posible error referencia puntero (S/N·)
- Size: Valor devuelto por sizeof



Comprendiendo Arrays y Punteros #2

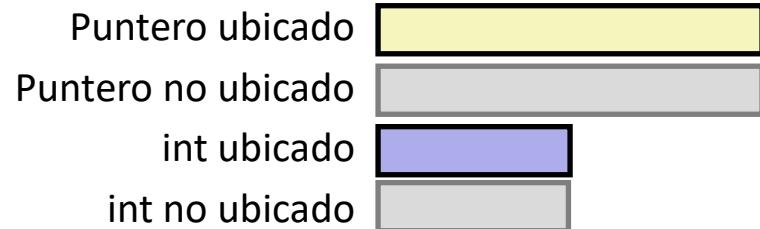
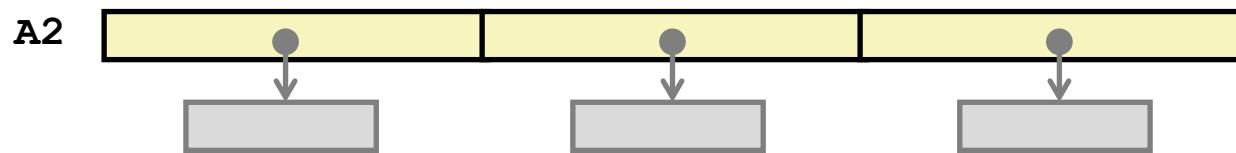
Decl	<i>An</i>			<i>*An</i>			<i>**An</i>		
	Cmp	Ptr	Size	Cmp	Ptr	Size	Cmp	Ptr	Size
<code>int A1[3]</code>									
<code>int *A2[3]</code>									
<code>int (*A3)[3]</code>					.†	12			



† gcc traduce void*v=A3;
igual que int*p=*&A3;
como movq A3(%rip), %rax,
movq %rax, p/v(%rip).

Comprendiendo Arrays y Punteros #2

Decl	A_n			$*A_n$			$**A_n$		
	Cmp	Ptr	Size	Cmp	Ptr	Size	Cmp	Ptr	Size
int A1[3]	.	.	12	.	.	4	N	-	-
int *A2[3]	.	.	24	.	.	8	.	S	4
int (*A3)[3]	.	.	8	.	.†	12	.	S	4



† gcc traduce void*v=A3;
igual que int*p=*A3;
como movq A3(%rip), %rax,
movq %rax, p/v(%rip).

Arrays Multidimensionales (Anidados)

↗ Matriz.

■ Declaración

$T \ A[R][C];$

- Array 2D de (elems. de) tipo T
- R filas (rows), C columnas
- Elems. tipo T requieren K bytes

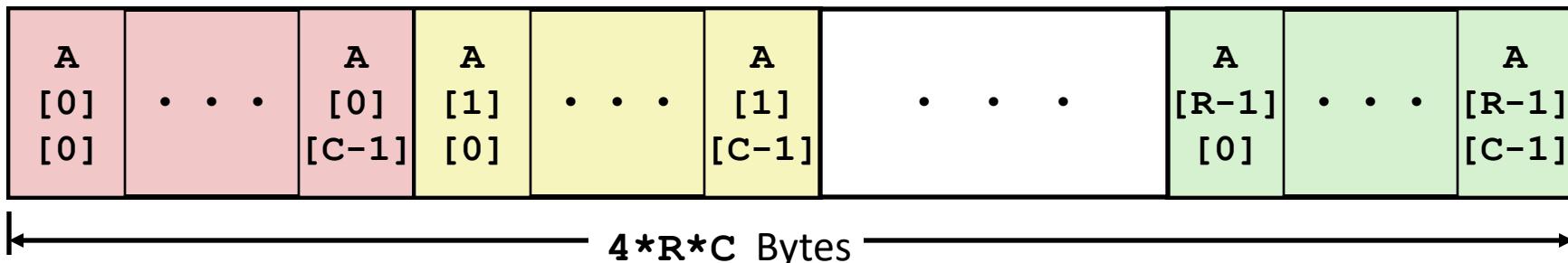
■ Tamaño Array

- $R * C * K$ bytes

■ Disposición

- Almacenamiento por filas (row-major-order)[†]

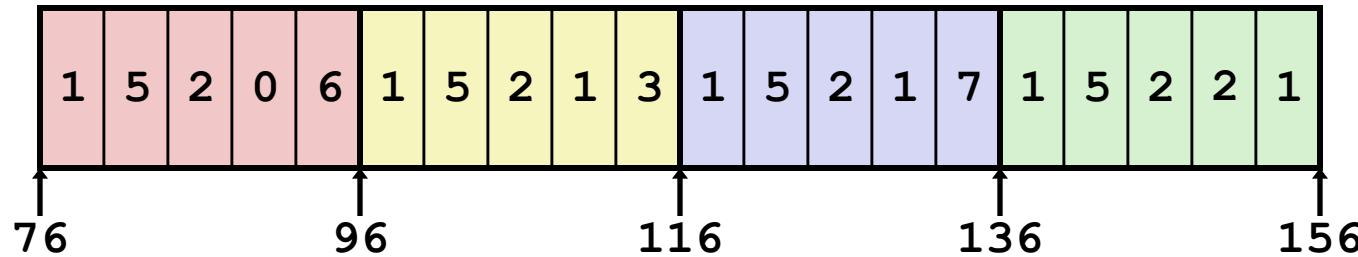
`int A[R][C];`



Ejemplo de Array Anidado

```
#define PCOUNT 4
zip_dig pgh[PCOUNT] =
{{1, 5, 2, 0, 6},
 {1, 5, 2, 1, 3 },
 {1, 5, 2, 1, 7 },
 {1, 5, 2, 2, 1 }};
```

zip_dig
pgh[4];



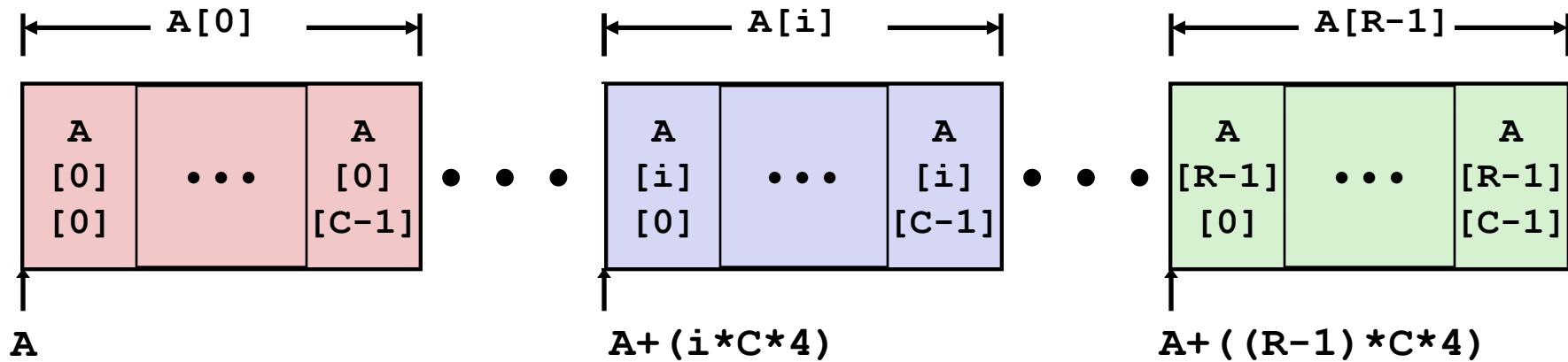
- “`zip_dig pgh[4]`” equivalente a “`int pgh[4][5]`”
 - Variable `pgh`: array de 4 elementos, ubicados contiguamente
 - Cada elemento es un array de 5 `int`'s, ubicados contiguamente
- Garantizado almacenamiento por filas (“row-major order”)

Acceso a Filas en Arrays Anidados

■ Vectores Fila

- $A[i]$ es un array de C elementos
- Cada elemento de tipo T requiere K bytes
- Dirección de comienzo $A + i * (C * K)$

```
int A[R][C];
```



Código Acceso Filas Arrays Anidados

```
int *get_pgh_zip
    (size_t index)
{
    return pgh[index];
}
```

```
#define PCOUNT 4
zip_dig pgh[PCOUNT] =
    {{1, 5, 2, 0, 6},
     {1, 5, 2, 1, 3},
     {1, 5, 2, 1, 7},
     {1, 5, 2, 2, 1}};
```

```
get_pgh_zip:
    leaq (%rdi,%rdi,4), %rax      # index en %rdi
                                         # 5 * index
+ leaq    pgh(,%rax,4), %rax      # pgh + (20 * index)
    ret
```

■ Vector Fila

- `pgh[index]` es array de 5 int's, comienza en `pgh+20*index`

■ Código x86-64

- Calcula `pgh + 4 * (index+4*index)`

Código Acceso Filas Arrays Anidados

```
int *get_pgh_zip
    (int index)
{
    return pgh[index];
}
```

```
#define PCOUNT 4
zip_dig pgh[PCOUNT] =
    {{1, 5, 2, 0, 6},
     {1, 5, 2, 1, 3},
     {1, 5, 2, 1, 7},
     {1, 5, 2, 2, 1}};
```

`get_pgh_zip:`

<code>movslq %edi, %rdi</code> <code>leaq (%rdi,%rdi,4), %rdx</code> [†] <code>leaq pgh(%rip), %rax</code> <code>leaq (%rax,%rdx,4), %rax</code> <code>ret</code>	<code># index = %edi</code> <code># 5 * index</code> <code># pgh</code> <code># pgh + (20 * index)</code>
--	--

■ Vector Fila

- `pgh[index]` es array de 5 `int`'s, comienza en `pgh+20*index`

■ Código x86-64

- Calcula `pgh + 4 * (index+4*index)`

[†] es direccionamiento relativo a Contador de Programa `%rip`, existe en x86-64, no en x86.

Activado por defecto desde Ubuntu 17.10

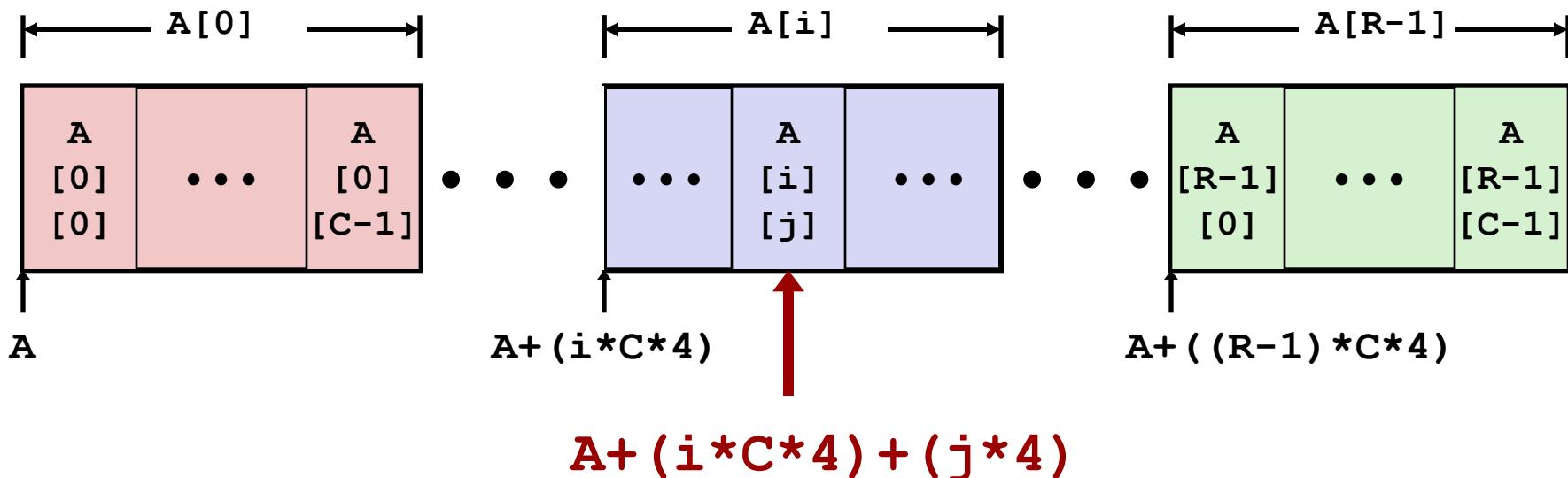
`#Position-independent_executables`

Acceso a Elementos en Arrays Anidados

■ Elementos del Array

- $A[i][j]$ es elemento de tipo T , que requiere K bytes
- Dirección $A + i * (C * K) + j * K = A + (i * C + j) * K$

```
int A[R][C];
```



$$A + (i * C * 4) + (j * 4)$$

Código Acceso Elementos Arrays Anidados

```
int get_pgh_digit
    (size_t index, size_t digit)
{
    return pgh[index][digit];
}
```

```
get_pgh_digit:
                    # digit en %rax
                    # index en %rdi
leaq   (%rdi,%rdi,4), %rax      # 5*index
addq   %rax, %rsi                # 5*index+digit
  
+movl  pgh(,%rsi,4), %eax      # pgh + 4 * (5*index+digit)
ret
```

■ Elementos del Array

- `pgh[index] [dig]` es `int`
- Dirección: $pgh + 20*idx + 4*dig = pgh + 4*(5*idx+dig)$
- Código x86_64 calcula la dirección $pgh + 4*((idx+4*idx)+dig)$

Código Acceso Elementos Arrays Anidados

```
int get_pgh_digit
    (int index, int digit)
{
    return pgh[index][digit];
}
```

```
get_pgh_digit:
    movslq %esi, %rax          # digit = %rax
    movslq %edi, %rdi          # index = %rdi
    leaq   (%rdi,%rdi,4), %rsi # 5*index
    addq   %rax, %rsi          # 5*index+digit
    leaq   pgh(%rip), %rax     # pgh
    movl   (%rax,%rsi,4), %eax # pgh + 4 * (5*index+digit)
    ret
```

■ Elementos del Array

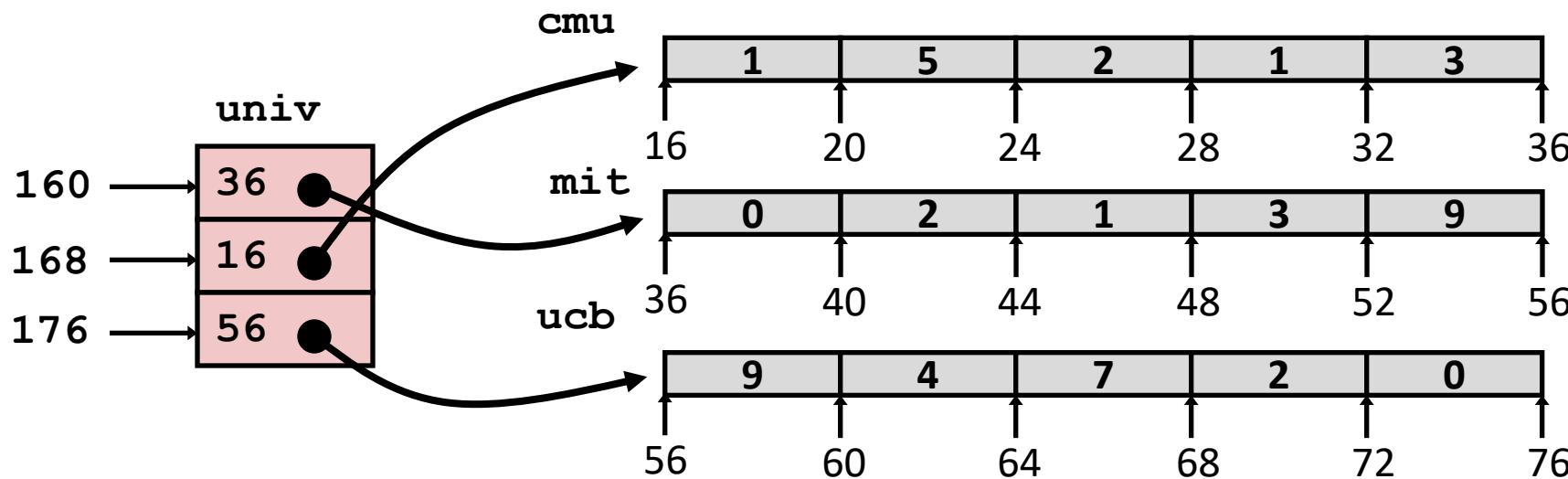
- `pgh[index] [dig]` es `int`
- Dirección: $pgh + 20*idx + 4*dig = pgh + 4*(5*idx+dig)$
- Código x86_64 calcula la dirección $pgh + 4*((idx+4*idx)+dig)$

Ejemplo de Array Multi-Nivel[†]

```
zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig mit = { 0, 2, 1, 3, 9 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```

```
#define UCOUNT 3
int *univ[UCOUNT] = {mit, cmu, ucb};
```

- Variable **univ** denota un array de 3 elementos
- Cada elemento un puntero
 - 8 bytes
- Cada puntero apunta a un array de int's



[†] no se suele decir "multinivel", es más usual "array de punteros". 23

Acceso a Elementos en Array Multi-Nivel

```
int get_univ_digit  
    (size_t index, size_t digit)  
{  
    return univ[index][digit];  
}
```

```
get_univ_digit:  
  
    movq    univ(,%rdi,8), %rax  # p = * (univ+8*index)  
    movl    (%rax,%rsi,4), %eax  # return * (p+4*digit)  
    ret
```

■ Cuentas

- Acceso a elemento **Mem[Mem[univ+8*index]+4*digit]**
- Debe hacer **dos lecturas de memoria**
 - Primero **obtener puntero al array[†] fila**
 - Entonces **acceder elemento dentro del array[†]**

[†] se suele decir “vector” (1D) como opuesto a “matriz” (2D) 24

Acceso a Elementos en Array Multi-Nivel

```
int get_univ_digit
    (size_t index, size_t digit)
{
    return univ[index][digit];
}
```

```
get_univ_digit:
    † salq    $2, %rsi          # 4*digit
    addq    univ(,%rdi,8), %rsi   # p = univ[index] + 4*digit
    movl    (%rsi), %eax        # return *p
    ret
```

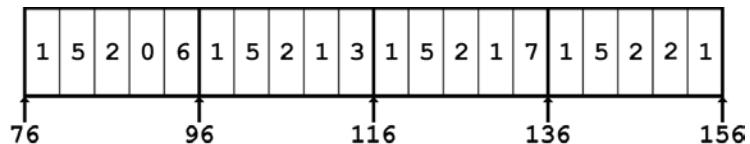
■ Cuentas

- Acceso a elemento **Mem[Mem[univ+8*index]+4*digit]**
- Debe hacer dos lecturas de memoria
 - Primero obtener puntero al array[†] fila
 - Entonces acceder elemento dentro del array[†]

Acceso a Elementos en Arrays

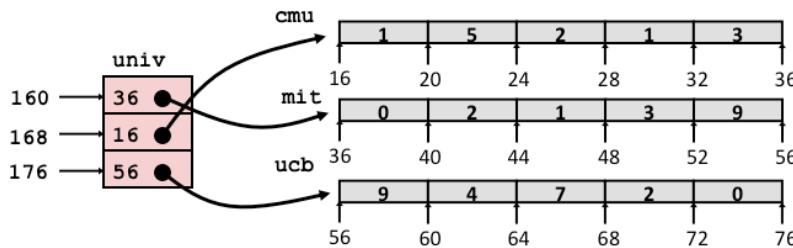
Array anidado

```
int get_pgh_digit
    (size_t index, size_t digit)
{
    return pgh[index] [digit];
}
```



Array Multi-nivel

```
int get_univ_digit
    (size_t index, size_t digit)
{
    return univ[index] [digit];
}
```



Accesos parecen similares en C, pero cuentas muy diferentes:

`Mem[pgh+20*index+4*digit]`

`Mem[Mem[univ+8*index]+4*digit]`

Código para Matriz N X N

■ Dimensiones fijas

- Se conoce valor de N en tiempo de compilación

```
#define N 16
typedef int fix_matrix[N][N];
/* Get element A[i][j] */
int fix_ele(fix_matrix A,
            size_t i, size_t j)
{
    return A[i][j];
}
```

■ Dimensiones variables, indexado explícito

- Forma tradicional de implementar arrays dinámicos

```
#define IDX(n, i, j) ((i)*(n)+(j))
/* Get element A[i][j] */
int vec_ele(size_t n, int *A,
            size_t i, size_t j)
{
    return A[IDX(n,i,j)];
}
```

■ Dimensiones variables, indexado implícito

- Soportado ahora^t por gcc

```
/* Get element A[i][j] */
int var_ele(size_t n, int A[n][n],
            size_t i, size_t j) {
    return A[i][j];
}
```

^t Ver p.ej.: <http://gcc.gnu.org/onlinedocs/gcc/Variable-Length.html>

ISO C99, gcc lo soporta en C90/C89 y C++ desde antes v-2.95 1999

Acceso a Matriz 16 X 16

■ Elementos del Array

- `int A[16][16];`
- Dirección $A + i * (C * K) + j * K$
- $C = 16, K = 4$

```
/* Get element A[i][j] */
int fix_ele(fix_matrix A, size_t i, size_t j)
{
    return a[i][j];
}
```

```
# A en %rdi, i en %rsi, j en %rdx
salq    $6, %rsi          # 64*i
addq    %rsi, %rdi         # A + 64*i
movl    (%rdi,%rdx,4), %eax # M[A + 64*i + 4*j]
ret
```

Acceso a Matriz n X n

■ Elementos del Array

- `int A[n][n];`
- Dirección `A + i * (C * K) + j * K`
- C = n, K = 4
- Hay que realizar multiplicación entera

```
/* Get element A[i][j] */
int var_ele(size_t n, int A[n][n], size_t i, size_t j)
{
    return A[i][j];
}
```

```
# n en %rdi, a en %rsi, i en %rdx, j en %rcx
imulq  %rdx, %rdi          # n*i
leaq   (%rsi,%rdi,4), %rax # A + 4*n*i
movl   (%rax,%rcx,4), %eax # A + 4*n*i + 4*j
ret
```

Ejemplo: Accesos a un Array

```
#include <stdio.h>
#define ZLEN 5
#define PCOUNT 4
typedef int zip_dig[ZLEN];

int main(int argc, char** argv) {
    zip_dig pgd = {{1, 5, 2, 0, 6},
                    {1, 5, 2, 1, 3 },
                    {1, 5, 2, 1, 7 },
                    {1, 5, 2, 2, 1 }};
    int *linear_zip = (int *) pgd;
    int *zip2 = (int *) pgd[2];
    int result =
        pgd[0][0] +
        linear_zip[7] +
        *(linear_zip + 8) +
        zip2[1];
    printf("result: %d\n", result);
    return 0;
}
```

```
linux> ./array
result: 9
```

Ejemplo: Accesos a un Array

```
#include <stdio.h>
#define ZLEN 5
#define PCOUNT 4
typedef int zip_dig[ZLEN];

int main(int argc, char** argv) {
    zip_dig pgm[PCOUNT] =
        {{1, 5, 2, 0, 6},
         {1, 5, 2, 1, 3 },
         {1, 5, 2, 1, 7 },
         {1, 5, 2, 2, 1 }};
    int *linear_zip = (int *) pgm;
    int *zip2 = (int *) pgm[2];
    int result =
        pgm[0][0] +
        linear_zip[7] +
        *(linear_zip + 8) +
        zip2[1];
    printf("result: %d\n", result);
    return 0;
}
```

```
linux> ./array
result: 9
```

Programación Máquina IV: Datos

■ Arrays

- Uni-dimensionales
- Multi-dimensionales (anidados)
- Multi-nivel

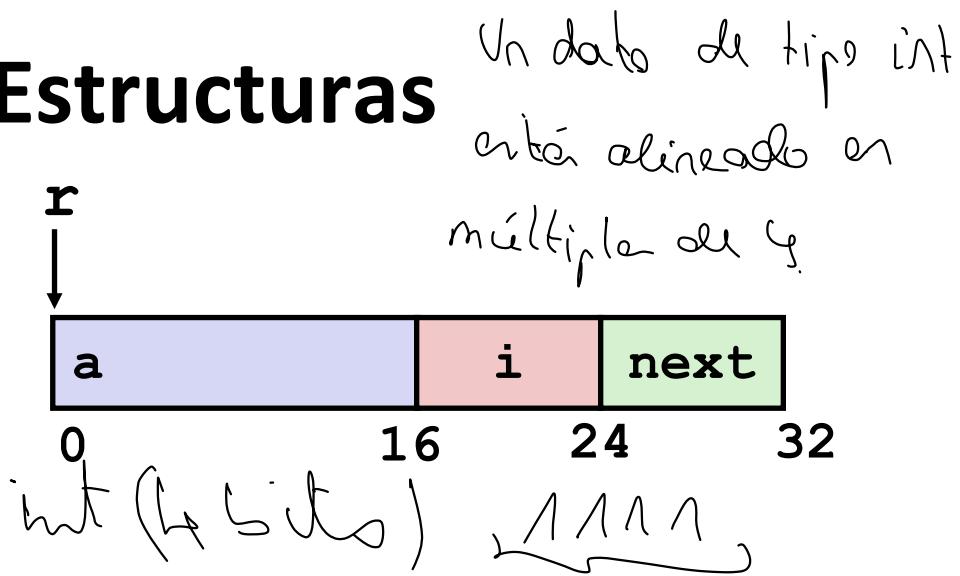
■ Estructuras

- Ubicación
- Acceso
- Alineamiento

■ Uniones

Representación de Estructuras

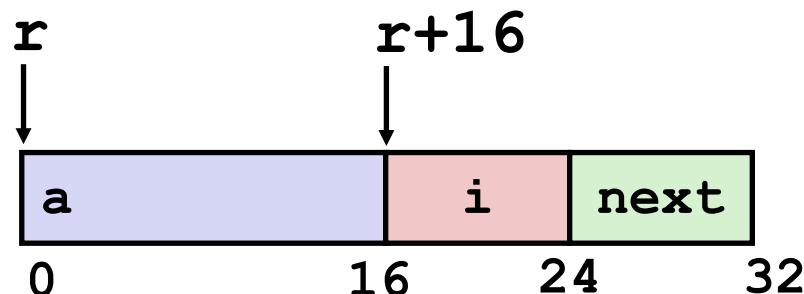
```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Estructuras representadas como un **bloque de memoria**
 - Suficientemente **grande** como para **contener** todos los **campos**
- Referencia a **campos** de la estructura mediante sus **nombres**
 - Sintaxis **struct.field**, **pointer->field**
- Campos **ordenados** según la **declaración**
 - Incluso si otro orden pudiera producir una representación más compacta
- Compilador determina **posición/tamaño** conjunto de los **campos**
 - El **programa a nivel máquina** no **entiende** las **estructuras** del **código fuente**

Acceso a Estructuras

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



■ Accediendo a un Miembro de la Estructura

- Puntero indica primer byte de la estructura[‡]
- Acceder a los elementos mediante sus desplazamientos[†]

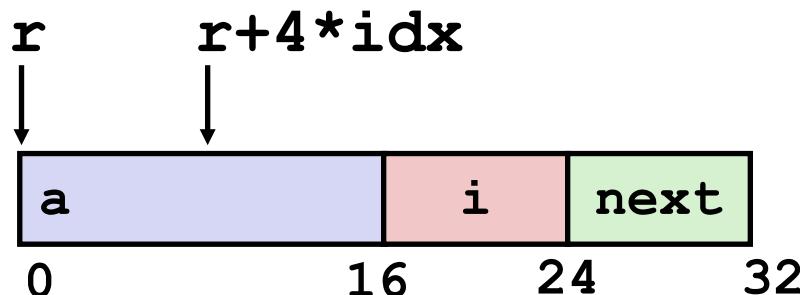
```
void set_i
    (struct rec *r,
     size_t val)
{
    ‡ r->i = val;
}
```

<pre>set_i:</pre>	<pre># r en %rdi, # val en %rsi movq %rsi, 16(%rdi) # Mem[r+16] = val ret</pre>
-------------------	--

[†] “offset”=compensación, “desplazamiento”
[‡] `ptr->fld` es abreviación para `(*ptr).fld`
Si se declara “struct rec R;”, entonces “R.a” es array, “R.a[0]” y “R.i” enteros, “R.n” puntero, y “R.n->a” otra vez array 34

Generando Puntero a Miembro Estructura

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



■ Generando Puntero a un Elemento del Array

- Desplaz.[†] de cada miembro struct queda determinado en tiempo compilación
- Se calcula `r + 4*idx`

```
int *get_ap
(struct rec *r, size_t idx)
{
# return &r->a[idx];
```

```
# r en %rdi, idx en %rsi
leaq (%rdi,%rsi,4), %rax
ret
```

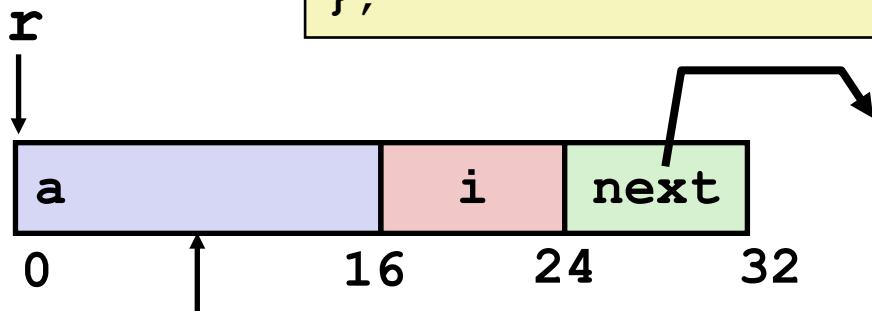
[†] “offset”=compensación, “desplazamiento”
[#] `ptr->fld` es abreviación para `(*ptr).fld`
 Si se declara “`struct rec R;`”, entonces
`“R.a”` es array, `“R.a[0]”` y `“R.i”` enteros,
`“R.n”` puntero, y `“R.n->a”` otra vez array

Siguiendo Lista Encadenada

■ Código C

```
void set_val
    (struct rec *r, int val)
{
    while (r) {
        int i = r->i;
        r->a[i] = val;
        r = r->next;
    }
}
```

```
struct rec {
    int a[3];
    int i;
    struct rec *next;
};
```

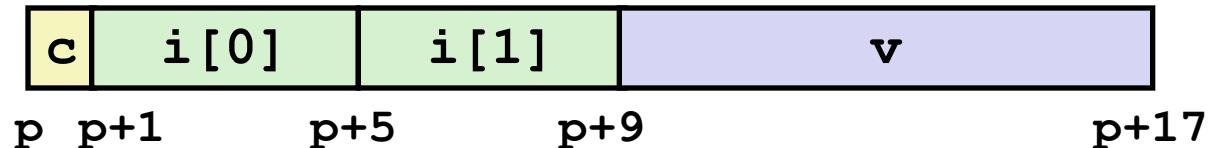


Registro	Valor
%rdi	r
%rsi	val

<pre>.L11:</pre> <pre> movslq 16(%rdi), %rax # i = M[r+16] movl %esi, (%rdi,%rax,4) # M[r+4*i] = val movq 24(%rdi), %rdi # r = M[r+24] testq %rdi, %rdi # Test r jne .L11 # if !=0 goto loop</pre>	<pre># loop: # i = M[r+16] # M[r+4*i] = val # r = M[r+24] # Test r # if !=0 goto loop</pre>
--	---

Estructuras y Alineamiento

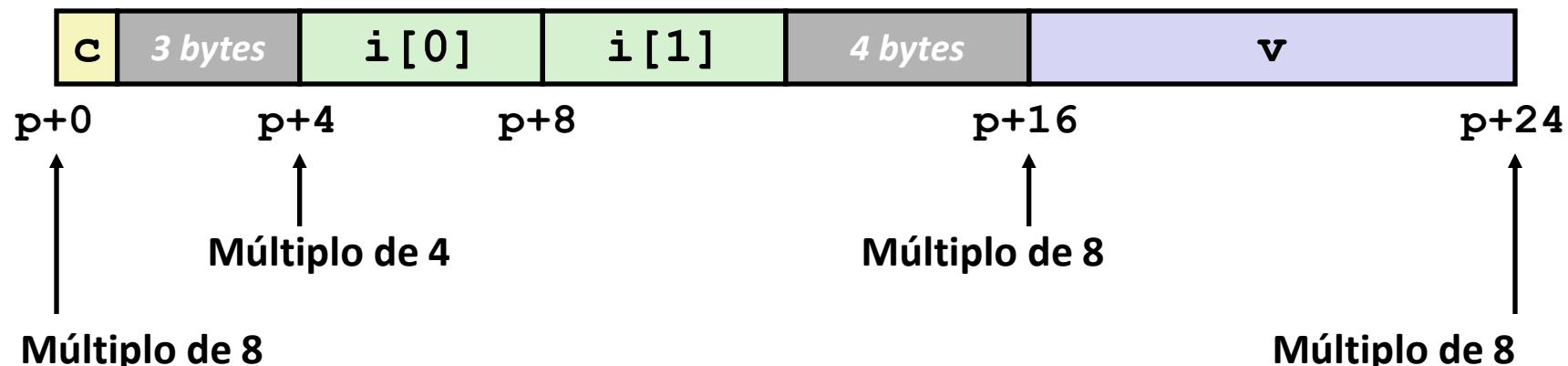
■ Datos Desalineados



```
struct S1 {
    char c;
    int i[2];
    double v;
} *p;
```

■ Datos Alineados

- El tipo de datos primitivo requiere K bytes
- La dirección debe ser múltiplo de K



Principios de Alineamiento

■ Datos Alineados

- El tipo de datos primitivo requiere K bytes
- La dirección debe ser múltiplo de K
- Requisito en algunas máquinas; recomendado en x86-64

■ Motivación para Alinear los Datos

- A la memoria se accede (físicamente) en trozos (alineados) de 4 ú 8 bytes (dependiendo del sistema)
 - Ineficiente cargar o almacenar dato que cruza frontera quad word
 - Mem. virtual muy delicada cuando un dato se extiende a 2 páginas

■ Compilador

- Inserta huecos en estructura para asegurar correcto alineamiento campos

Casos Concretos de Alineamiento

	<i>Linux x86</i>		<i>x86-64</i>		<i>Windows MinGW32</i>		<i>MinGW64</i>	
■ <i>Tipo de Datos C</i>	<i>tam.alin.</i>	<i>tam.alin.</i>	<i>tam.alin.</i>	<i>tam.alin.</i>	<i>tam.alin.</i>	<i>tam.alin.</i>	<i>tam.alin.</i>	<i>tam.alin.</i>
▪ char	1	1	1	1	1	1	1	1
▪ short	2	2	2	2	2	2	2	2
▪ int	4	4	4	4	4	4	4	4
▪ long	4	4	8	8	4	4	4	4
▪ long long	8	4	8	8	8	8	8	8
▪ float	4	4	4	4	4	4	4	4
▪ double	8	4	8	8	8	8	8	8
▪ long double	12	4	16	16	12	4	16	16
▪ void *	4	4	8	8	4	4	8	8
■ <i>Regla para memorizar</i>	4x		KB Kx		8x		KB Kx	
	ahorro M.		alin.estriicto		sin ahorro		sin long 8B	

estricto-12B #Typical alignment of C structs on x86

Casos Concretos de Alineamiento (x86-64)

- **1 byte: char, ...**
 - sin restricciones en la dirección
- **2 bytes: short, ...**
 - el LSb[†] (bit más bajo) de la dirección debe ser 0_2
- **4 bytes: int, float, ...** (y long en Windows!!!)
 - los 2 LSb's de la dirección deben ser 00_2
- **8 bytes: double, long , char *, ...**
 - los 3 LSb's de la dirección deben ser 000_2
- **16 bytes: long double (GCC en Linux x86-64)**
 - los 4 LSb's de la dirección deben ser 0000_2

Cumpliendo Alineamiento en Estructuras

■ Dentro de la estructura:

- Deben cumplirse requisitos alinm. de cada elemento

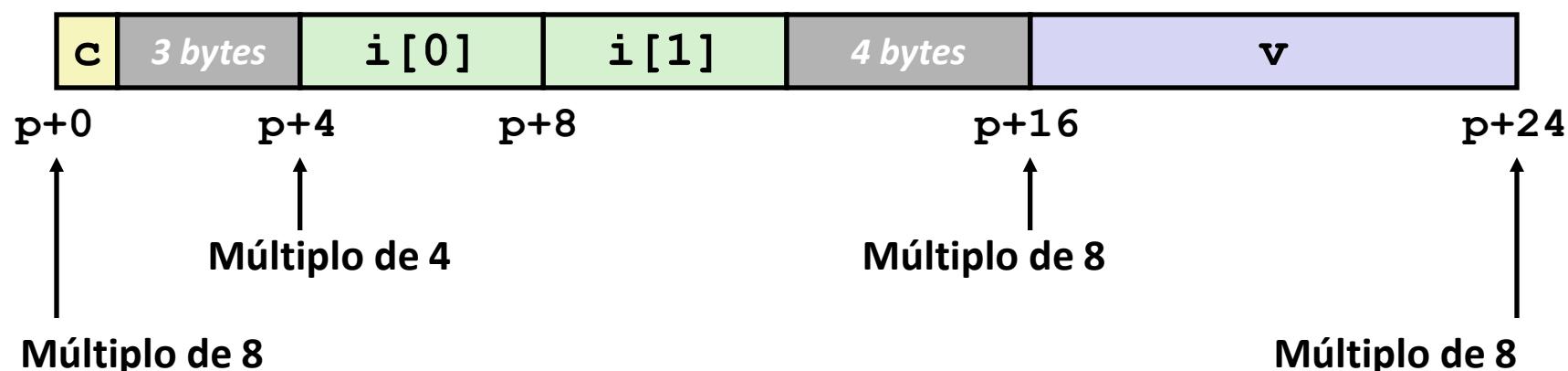
■ Colocación global de la estructura

- Cada estructura tiene un requisito de alineamiento K
 - K = Mayor alineamiento de cualquier elemento
- Dirección inicial y longitud estructura deben ser múltiplos de K

■ Ejemplo:

- K = 8, debido al elemento **double**

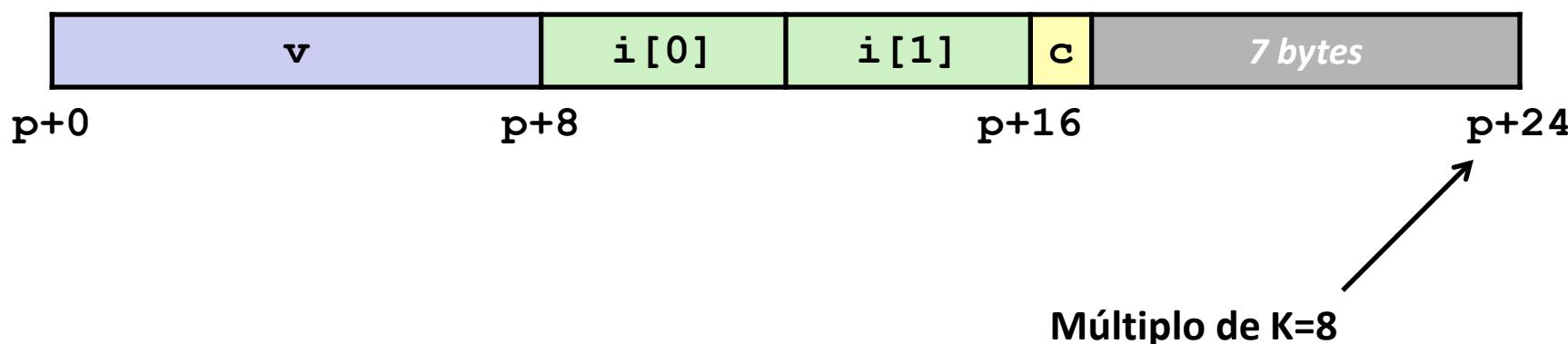
```
struct S1 {  
    char c;  
    int i[2];  
    double v;  
} *p;
```



Cumpliendo Requisito Alineamiento Global

- Si el requisito de alineamiento máximo es K
- La struct debe ocupar glob. múltiplo de K

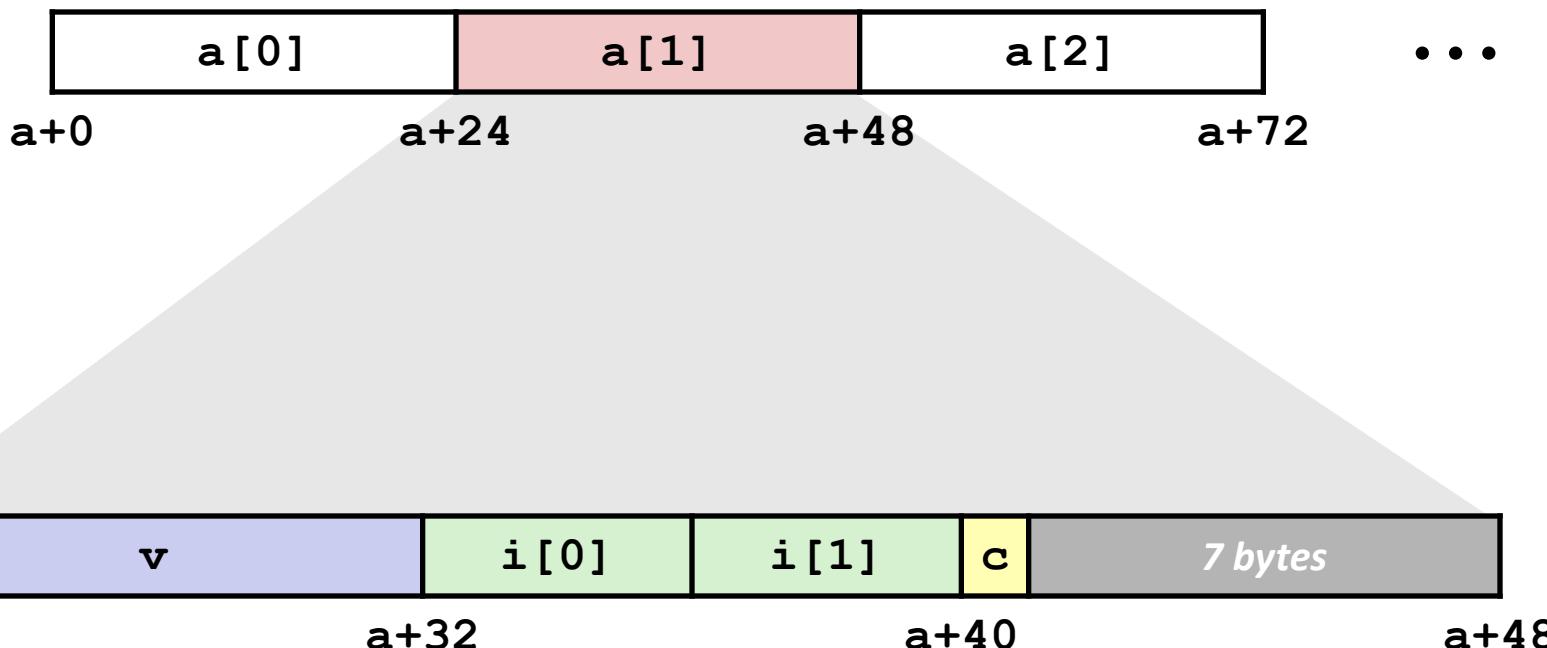
```
struct S2 {  
    double v;  
    int i[2];  
    char c;  
} *p;
```



Arrays de Estructuras

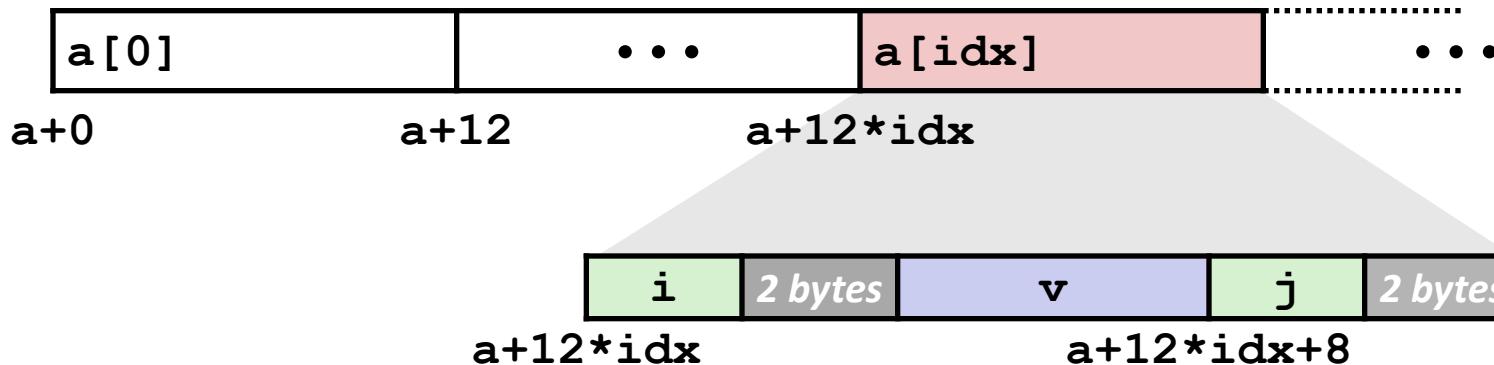
- Longitud global estructura[†] múltiplo de K
- Cumplir requisitos alnmto. de cada elemento

```
struct S2 {  
    double v;  
    int i[2];  
    char c;  
} a[10];
```



Acceso a Elementos del Array

- Calcular desplazamiento elem. array: $12i$
 - $\text{sizeof}(S3) * i$, incluyendo espaciadores alineamiento
- Elemento $j @$ despl. 8 dentro de estructura
- El ensamblador genera desplazamiento $a+8$
 - Resuelto durante enlazado (en tiempo de *link*)



```
short get_j(size_t idx)
{
    return a[idx].j;
}
```

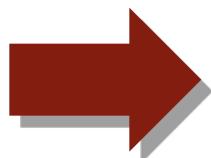
```
# idx en %rdi
leaq (%rdi,%rdi,2),%rax # 3*idx
+ movzwl a+8(%rax,4),%eax
```

[†] "Move with Zero-extend Word to Long", mnemotécnico MOVZX según Intel

Ahorro de Espacio

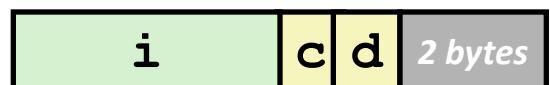
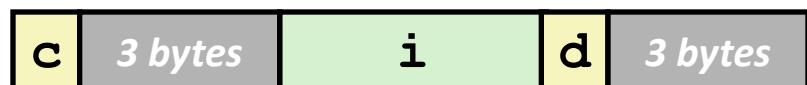
■ Poner primero los tipos de datos grandes

```
struct S4 {  
    char c;  
    int i;  
    char d;  
} *p;
```



```
struct S5 {  
    int i;  
    char c;  
    char d;  
} *p;
```

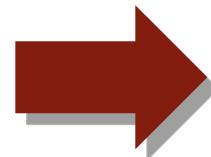
■ Efecto (K=4)



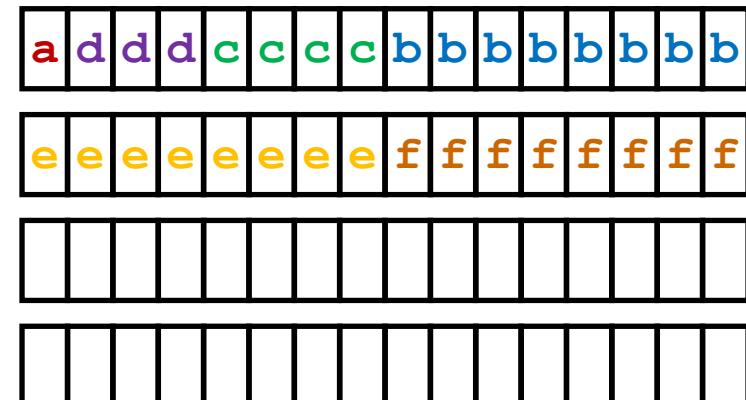
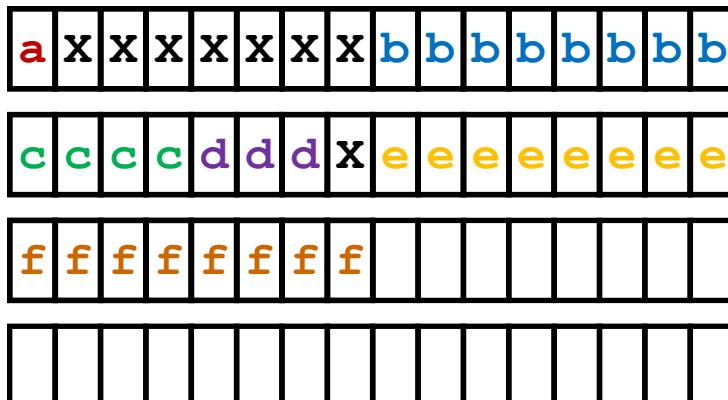
Ejemplo: Ejercicio sobre structs

- Indicar cómo se ubicaría en memoria la siguiente estructura (gcc Linux x86_64), marcando las posiciones ocupadas por cada campo con su nombre, y las de relleno (tanto interno como global) con una X. Repetir el ejercicio reordenando los campos para obtener el máximo ahorro de memoria posible. (ver ~ Ex.Probl.Sep'13)

```
struct foo {  
    char a;  
    long b;  
    float c;  
    char d[3];  
    int *e;  
    short *f;  
} mystruct1;
```



```
struct bar {  
    char a;  
    char d[3];  
    float c;  
    long b;  
    int *e;  
    short *f;  
} mystruct2;
```



Programación Máquina IV: Datos

■ Arrays

- Uni-dimensionales
- Multi-dimensionales (anidados)
- Multi-nivel

■ Estructuras

- Ubicación
- Acceso
- Alineamiento

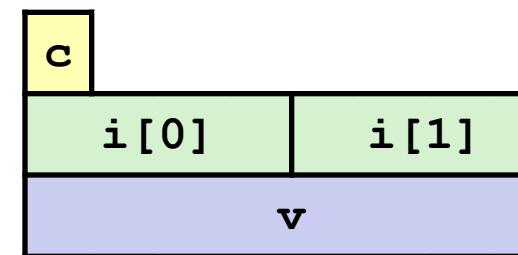
■ Uniones

Ubicación[†] de Uniones

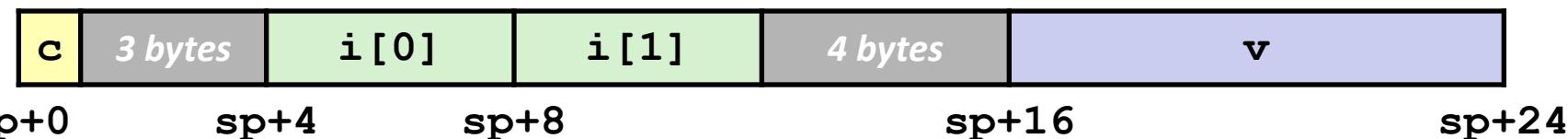
- Reservar[†] de acuerdo al elemento más grande
- Sólo puede usarse un campo a la vez

```
union U1 {  
    char c;  
    int i[2];  
    double v;  
} *up;
```

```
struct S1 {  
    char c;  
    int i[2];  
    double v;  
} *sp;
```



up+0 up+4 up+8



Uso de Uniones para Acceder Patrones Bit

```
typedef union {
    float f;
    unsigned u;
} bit_float_t;
```



```
float bit2float(unsigned u)
{
    bit_float_t arg;
    arg.u = u;
    return arg.f;
}
```

```
unsigned float2bit(float f)
{
    bit_float_t arg;
    arg.f = f;
    return arg.u;
}
```

¿Lo mismo que (float) u ?

¿Lo mismo que (unsigned) f ?

Ordenamiento de Bytes[†]: un Repaso

■ Idea

- Palabras short/long/quad, almacenadas en mem. como 2/4/8B consecutivos
- ¿Cuál es el byte más (menos) significativo?
- Puede causar problemas al intercambiar datos binarios entre máquinas

■ BigEndian[‡] (extremo mayor)

- El byte más significativo está en la dirección más baja (“viene primero”)
- Sparc

■ LittleEndian[‡] (extremo menor)

- El byte menos significativo está en la dirección más baja
- Intel x86, ARM con Android e IOS

■ BiEndian

- Se puede configurar de cualquiera de las dos formas
- ARM

[†] “byte ordering” en inglés, se refiere al orden de bytes en palabras, no a ordenar un array de bytes = “sorting”

[‡] “big/little endian” = “partidario extremo mayor/menor”, ver libro, §2.1.3, Aside: Origin of “endian” 50

Ejemplo de Ordenamiento de Bytes

```
union {
    unsigned char c[8];
    unsigned short s[4];
    unsigned int i[2];
    unsigned long l[1];
} dw;
```

32-bit

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]				
s[0]		s[1]		s[2]		s[3]					
i[0]				i[1]							
l[0]											

64-bit

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]				
s[0]		s[1]		s[2]		s[3]					
i[0]				i[1]							
l[0]											

Ejemplo de Ordenamiento de Bytes (Cont.).

```
int j;
for (j = 0; j < 8; j++)
    dw.c[j] = 0xf0 + j;

printf("Characters 0-7 ==
[0x%x,0x%x,0x%x,0x%x,0x%x,0x%x,0x%x,0x%x]\n",
       dw.c[0], dw.c[1], dw.c[2], dw.c[3],
       dw.c[4], dw.c[5], dw.c[6], dw.c[7]);

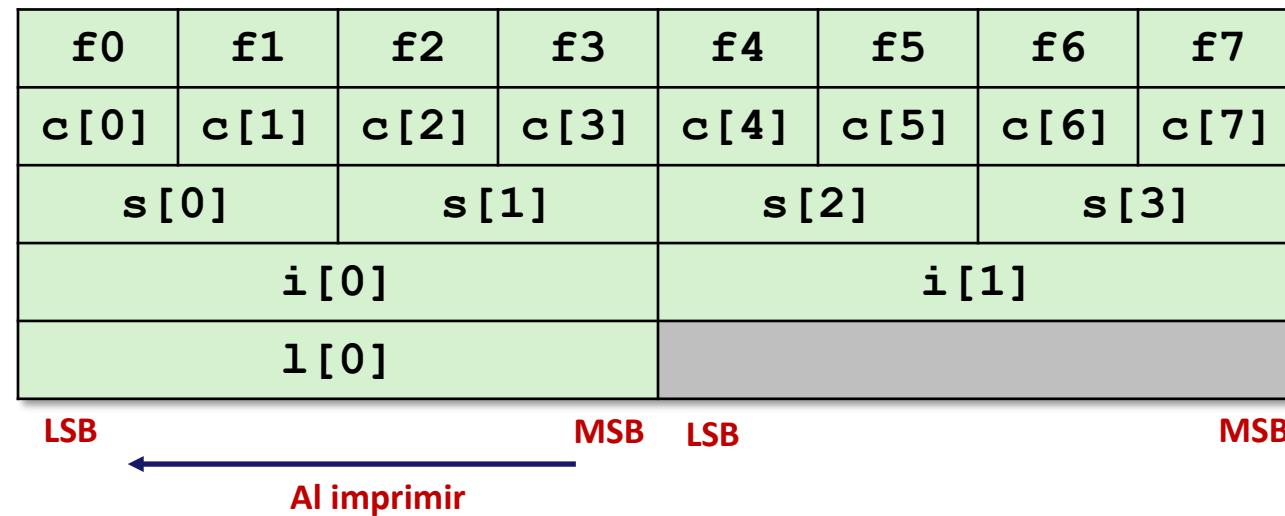
printf("Shorts 0-3 == [0x%x,0x%x,0x%x,0x%x]\n",
       dw.s[0], dw.s[1], dw.s[2], dw.s[3]);

printf("Ints 0-1 == [0x%x,0x%x]\n",
       dw.i[0], dw.i[1]);

printf("Long 0 == [0x%lx]\n",
       dw.l[0]);
```

Ordenamiento de Bytes en IA32

Little Endian



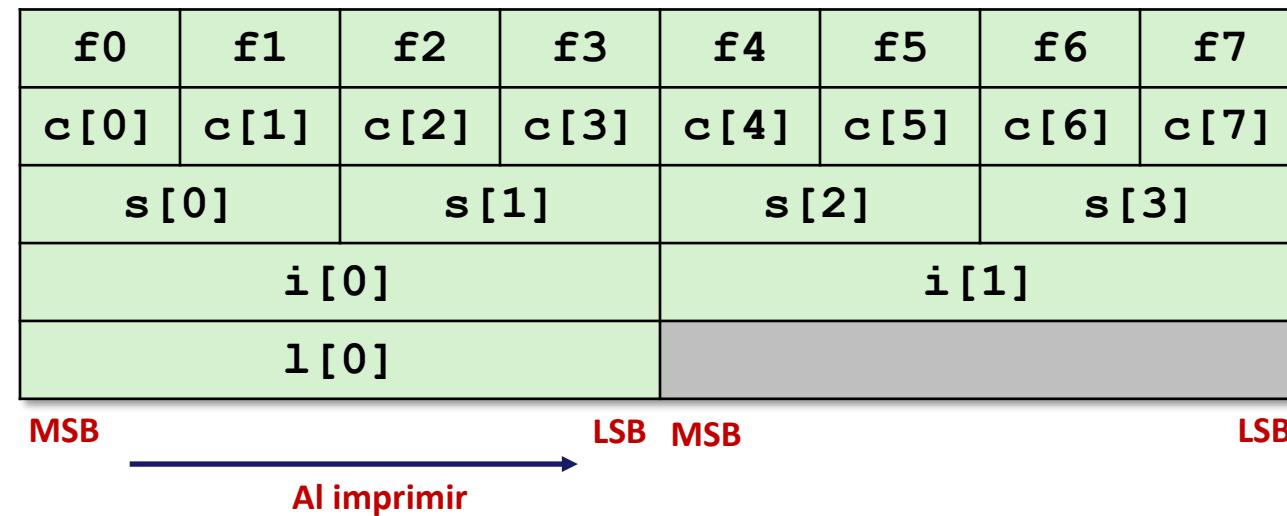
Salida :

```

Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts      0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints        0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long         0    == [0xf3f2f1f0]
  
```

Ordenamiento de Bytes en Sun

Big Endian

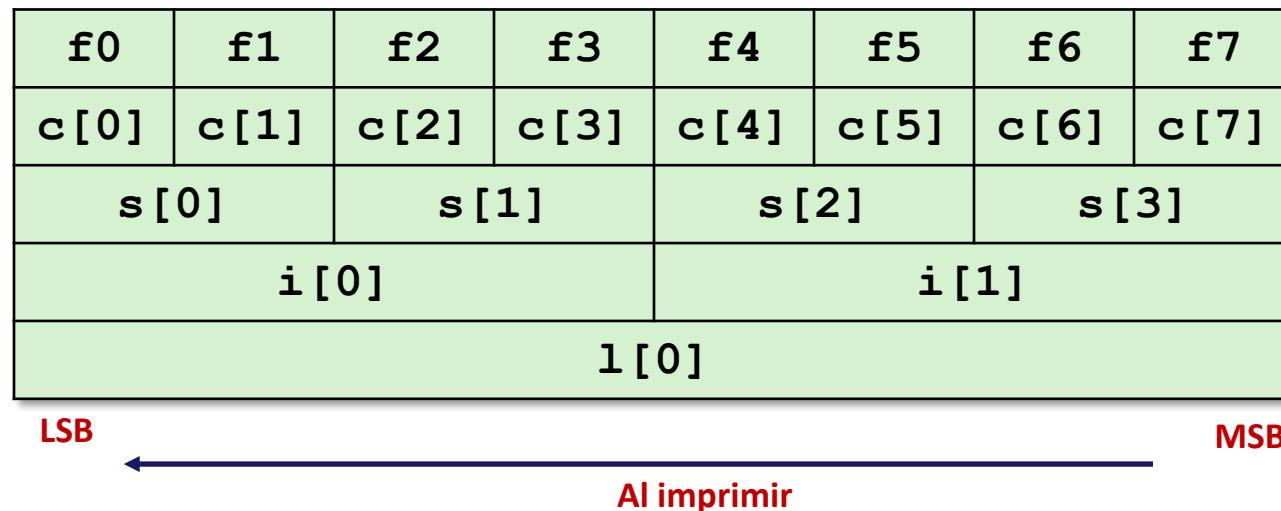


Salida en Sun:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf0f1,0xf2f3,0xf4f5,0xf6f7]
Ints       0-1 == [0xf0f1f2f3,0xf4f5f6f7]
Long        0    == [0xf0f1f2f3]
```

Ordenamiento de Bytes en x86-64

Little Endian



Salida en x86-64:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts    0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints      0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long      0    == [0xf7f6f5f4f3f2f1f0]
```

Resumen de Tipos Compuestos en C

■ Arrays

- Reserva de memoria contigua para almacenar elementos
- Se usa aritmética de indexación para localizar elementos individuales
- Puntero al primer elemento
- Sin chequeo de límites

■ Estructuras

- Reserva de una sola región de memoria, campos van en el orden declarado
- Se accede usando desplazamientos determinados por el compilador
- Puede requerir relleno interno y externo para cumplir con el alineamiento

■ Combinaciones

- Se pueden anidar representación estructura y array arbitrariamente
- Relleno externo estructuras garantiza alineamiento en arrays de structs

■ Uniones

- Declaraciones superpuestas
- Forma de soslayar el sistema de promoción de tipos en C

TEMA 3

Unidad de control

■ Camino de datos

- Unidad de procesamiento y unidad de control
- Unidad de procesamiento con un bus
- Unidad de procesamiento con múltiples buses

■ Unidades de control cableadas y microprogramadas

- Diseño de una UC cableada
- Ejemplo de UC cableada
- Concepto de UC microprogramada

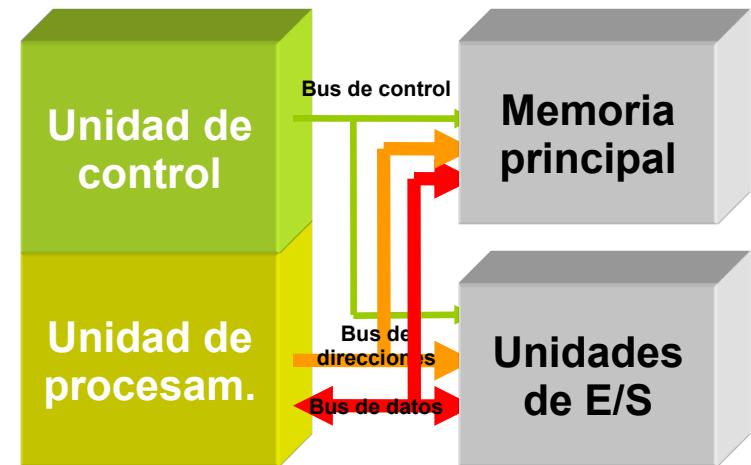
■ Control microprogramado

- Formato de las microinstrucciones
- Nanoprogramación
- Secuenciamiento de microinstrucciones
- Ejemplo de arquitectura microprogramada

Introducción

- Un computador con arquitectura Von Neumann consta de tres bloques fundamentales:

- CPU o procesador
 - Memoria principal
 - (datos + instrucciones)
 - Unidades de E/S
 - (y memoria masiva)
- } unidos mediante buses



- En este tema estudiaremos la CPU, que puede entenderse como una unidad constituida por:
 - Unidad de procesamiento o camino de datos ("datapath")
 - Unidad de control

Unidad de procesamiento

- La unidad de procesamiento comprende elementos hardware como:
 - unidades funcionales (ALU, desplazad., multiplic., etc.)
 - registros
 - registros de uso general (varios GPR)
 - registro de estado
 - contador de programa (PC)
 - puntero de pila (SP)
 - registro de instrucción (IR)
 - registro de dato de memoria (MDR / MBR)
 - registro de dirección de memoria (MAR)
 - multiplexores
 - buses internos
 - ...

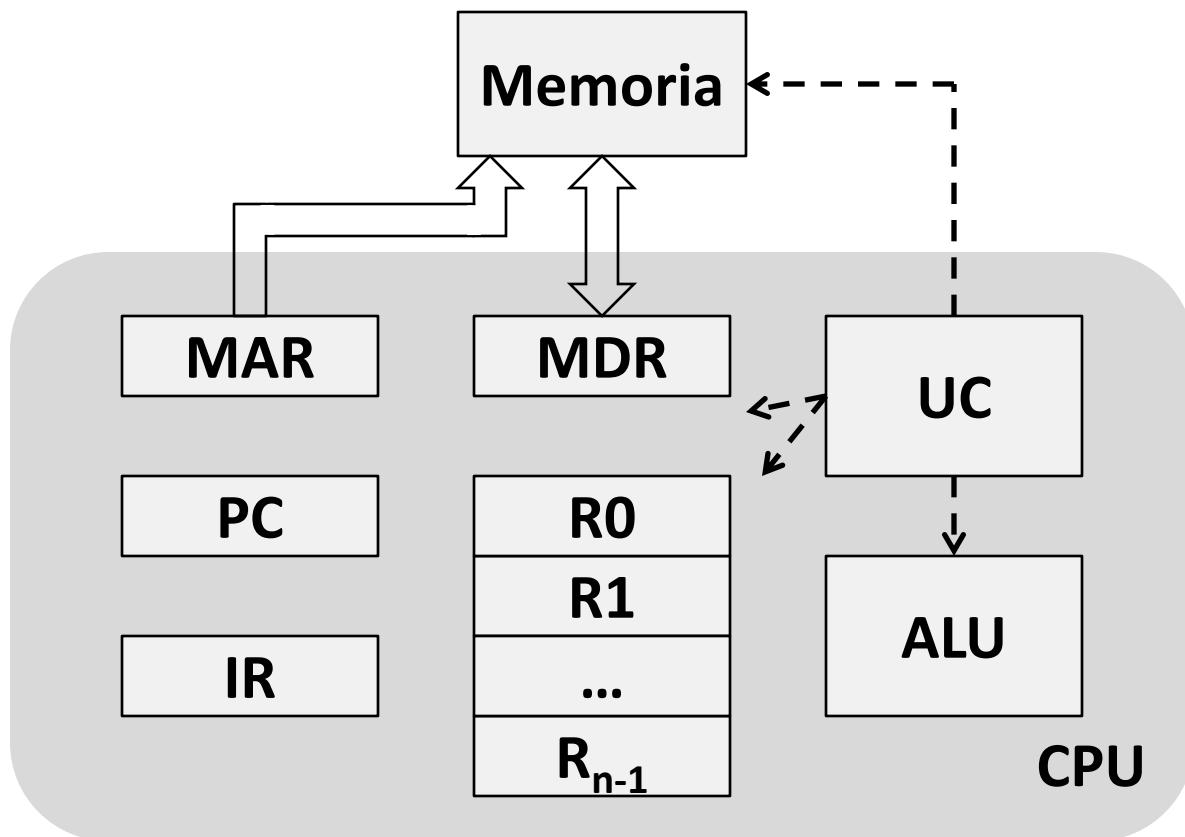
Unidad de control

- **La unidad de control interpreta y controla la ejecución de las instrucciones leídas de la memoria principal, en dos fases:**
 - secuenciamiento de las instrucciones
 - La UC lee de MP la instrucción apuntada por PC, $IR \leftarrow M[PC]^*$
 - determina la dirección de la instrucción siguiente y la carga en PC*
 - ejecución/interpretación de la instrucción en IR
 - La UC reconoce el tipo de instrucción,
 - manda las señales necesarias para tomar los operandos necesarios y dirigirlos a las unidades funcionales adecuadas de la unidad de proceso,
 - manda las señales necesarias para realizar la operación,
 - manda las señales necesarias para enviar los resultados a su destino.

Ej. de ejecución Add A, R0*

■ Pensar tareas realizadas por UC para ejecutar instrucción

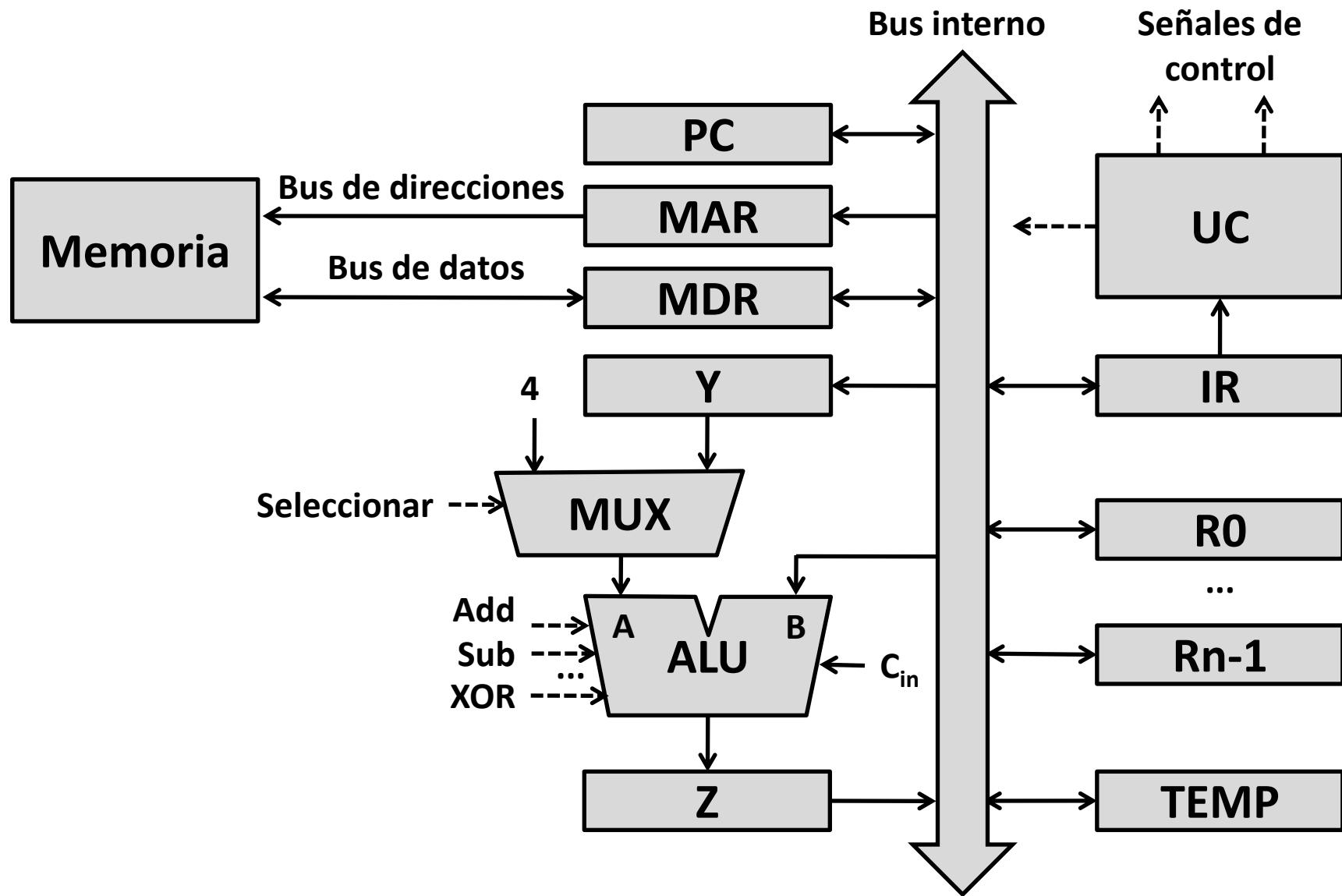
- Por ejemplo: Add A, R0
 - $M[A] + R0 \rightarrow R0$
- Detalles en [HAM03] Cap-1.3
- Ejercicios similares en TOC §2.1



Ej. de ejecución Add A, R0*

- $M[POS_A] + R0 \rightarrow R0$
 - Ensamblador traduce p.ej: $POS_A=100$
 - Valor anterior R0 perdido, el de POS_A se conserva
 - Arquitectura R/M
- **Pasos básicos de la UC**
 - PC apunta a posición donde se almacena instrucción
 - **Captación:** $MAR \leftarrow PC$, Read, $PC \leftarrow PC + 1$, $T_{acc} \leftarrow$ MDR \leftarrow bus, $IR \leftarrow MDR$
 - **Decodificación:** se separan campos instrucción
 - Codop: ADD $mem + reg \rightarrow reg$
 - Dato1: 100 direcciónamiento directo, habrá que leer $M[100]$
 - Dato2: 0 direcciónamiento registro, habrá que llevar R0 a ALU
 - CPUs con longitud instrucción variable – dirección (100) en siguiente palabra
 - **Operando:** $MAR \leftarrow 100$, Read, $T_{acc} \leftarrow$ MDR, $ALU_{in1} \leftarrow MDR$
 - **Ejecución:** $ALU_{in2} \leftarrow R0$, add, T_{alu}
 - **Almacenamiento:** $R0 \leftarrow ALU_{out}$

Unidad de procesamiento con un bus



Unidad de procesamiento con un bus

- Componentes interconectados mediante bus común
- MDR: dos entradas, dos salidas
- MAR: unidireccional (procesador → memoria)
- R0...Rn-1: GPR, SP, índices,...
- Y, Z, TEMP:
 - registros transparentes al programador
 - almácen temporal interno en la ejecución de una instrucción
- MUX: selecciona entrada A de la ALU
 - La constante 4 se usa para incrementar el contador de programa
- La UC genera señales internas y externas (memoria)

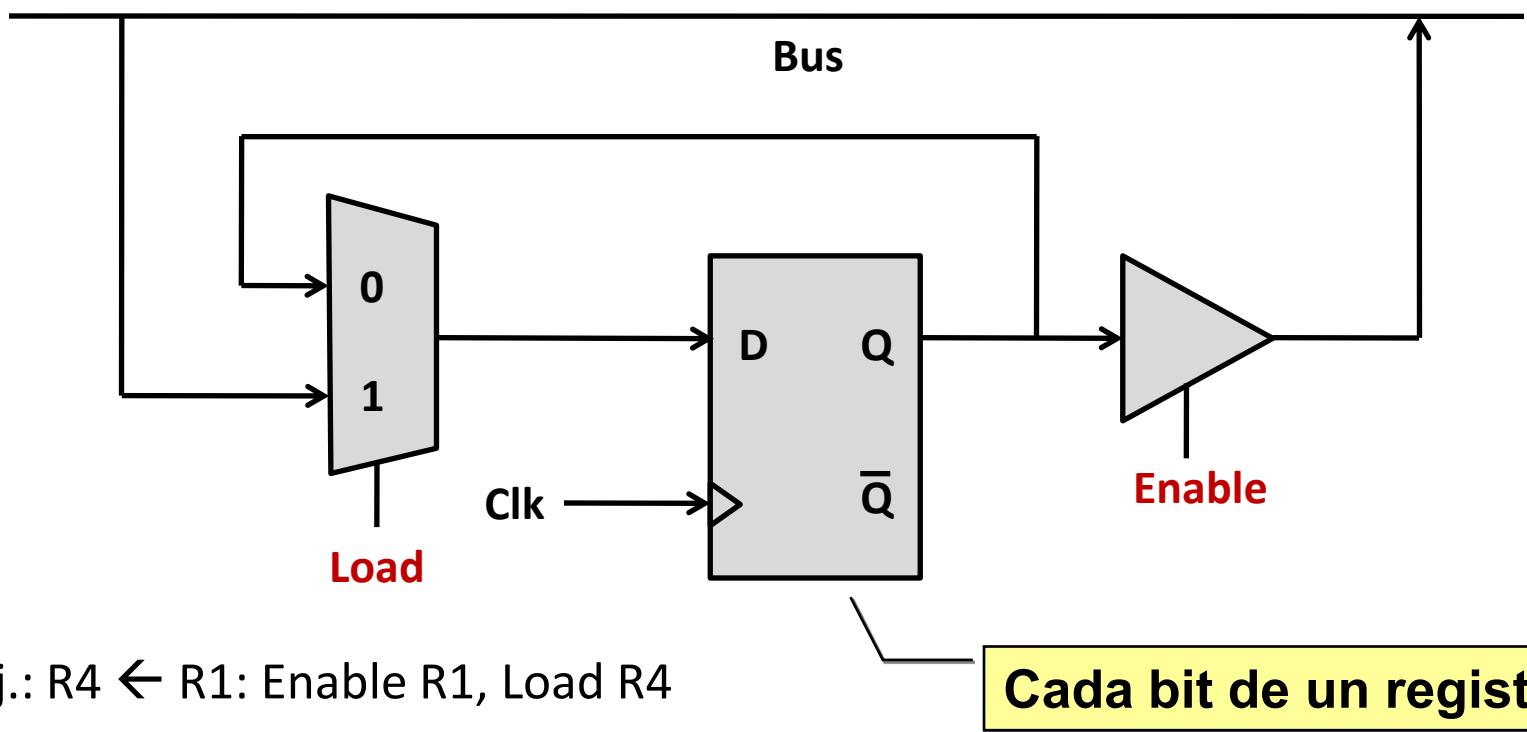
Unidad de procesamiento con un bus

- Una instrucción puede ser ejecutada mediante una o más de las siguientes operaciones:
 - Transferir de un registro a otro
 - Realizar operación aritmética o lógica y almacenar en registro
 - Cargar posición de memoria en registro
 - Almacenar registro en posición de memoria

Unidad de procesamiento con un bus

■ Transferir de un registro a otro

- Cada registro usa dos señales de control:
 - Load: Carga en paralelo
 - Enable: Habilitación de salida (buffer triestado)



Unidad de procesamiento con un bus

■ Realizar operación aritmética o lógica y almacenar en registro

- ALU: circuito combinacional sin memoria
- Resultado almacenado temporalmente en Z
- Ej.: $R3 \leftarrow R1 + R2$

1. Enable R1, Load Y
2. Enable R2, Select Y, Add, Load Z
3. Enable Z, Load R3

Cada línea: 1 ciclo de reloj

Esta transferencia no puede realizarse en el paso 2 ¿por qué?

- Las señales de control de la ALU podrían estar codificadas

Unidad de procesamiento con un bus

■ Cargar posición de memoria en registro

- Transferir dirección a MAR
- Activar lectura de memoria
- Almacenar dato leído en MDR (MDR dos entradas, dos salidas)
- La temporización interna debe coordinarse con la de la memoria
 - La lectura puede requerir varios ciclos de reloj
 - El procesador debe esperar la activación de señal de finalización de ciclo de memoria
- Ej.: **Load (R1) → R2**
 1. Enable R1, Load MAR
 2. Comenzar lectura
 3. Esperar fin de ciclo de memoria, Load MDR desde memoria
 4. Enable MDR hacia bus interno, Load R2

Unidad de procesamiento con un bus

■ Almacenar registro en posición de memoria

- Transferir dirección a MAR
- Transferir dato a escribir a MDR (MDR dos entradas, dos salidas)
- Activar escritura de memoria
- La temporización interna debe coordinarse con la de la memoria
 - La escritura puede requerir varios ciclos de reloj
 - El procesador debe esperar la activación de señal de finalización de ciclo de memoria
- Ej.: **Store R2 → (R1)**
 1. Enable R1, Load MAR
 2. Enable R2, Load MDR desde bus interno
 3. Comenzar escritura
 4. Esperar fin de ciclo de memoria

Unidad de procesamiento con un bus

■ Ejecución de una instrucción completa

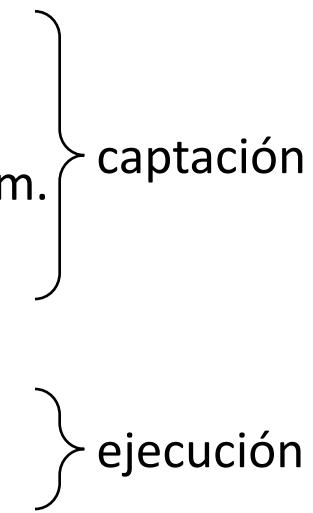
■ Ej.: **Add (R3) → R1**

1. Enable PC, Load MAR, Select 4, Sumar, Enable Z
 2. Comenzar lectura, Enable Z, Load PC, Load Y*
 3. Esperar fin de ciclo de memoria, Load MDR desde mem.
 4. Enable MDR hacia bus interno, Load IR
 5. Decodificar instrucción
 6. Enable R3, Load MAR
 7. Comenzar lectura, Enable R1, Load Y
 8. Esperar fin de ciclo de memoria, Load MDR desde mem.
 9. Enable MDR hacia bus interno, Select Y, Sumar, Load Z
 10. Enable Z, Load R1, Saltar a captación
-
- } captación
- } ejecución

Unidad de procesamiento con un bus

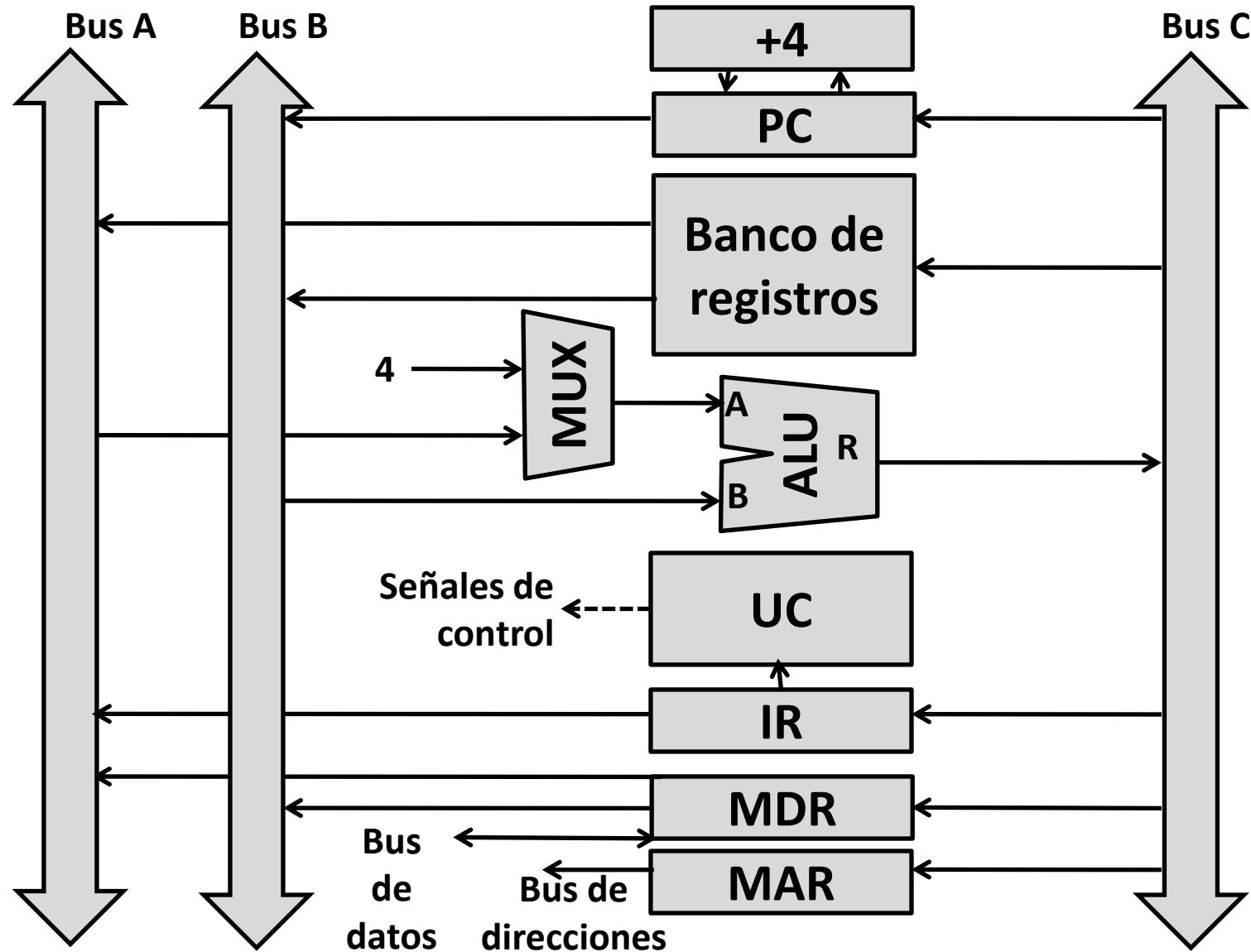
■ Ejecución de una instrucción de salto

■ Ej.: **Jmp desplazamiento**

1. Enable PC, Load MAR, Select 4, Sumar, Enable Z
 2. Comenzar lectura, Enable Z, Load PC, Load Y
 3. Esperar fin de ciclo de memoria, Load MDR desde mem.
 4. Enable MDR hacia bus interno, Load IR
 5. Decodificar instrucción
 6. Enable Campo desplazamiento en IR, Sumar, Load Z
 7. Enable Z, Load PC, Saltar a captación
- 
- captación
- ejecución

■ ¿Qué habría que añadir al paso 6 para **JS** (saltar si negativo)?

Unidad de procesam. buses múltiples



Unidad de procesam. buses múltiples

■ Banco de registros con tres puertos

- Dos registros pueden poner sus contenidos en los buses A y B
 - Un dato del bus C puede cargarse en un registro
- } En el mismo ciclo

■ ALU

- No necesita los registros Y y Z
- Puede pasar A o B directamente a R (bus C)

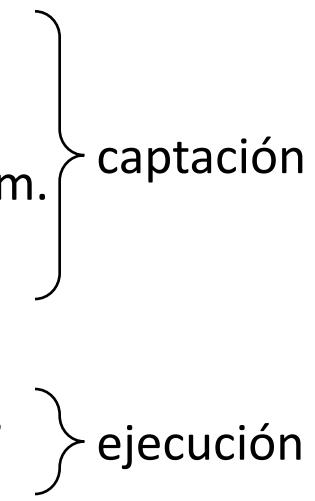
■ Unidad de incremento (+4)

- Pero la constante 4 como fuente de la ALU sigue siendo útil para incrementar otras direcciones en instrucciones de movimiento múltiple

Unidad de procesam. buses múltiples

■ Ejecución de una instrucción completa

■ Ej.: **R6 ← R4 + R5**

1. Enable PC, R=B, Load MAR
 2. Comenzar lectura, Incrementar PC
 3. Esperar fin de ciclo de memoria, Load MDR desde mem.
 4. Enable MDR hacia B, R=B, Load IR
 5. Decodificar instrucción
 6. Enable R4 hacia A, Enable R5 hacia B, Select A, Sumar,
Load R6, Saltar a captación
- 
- The diagram illustrates the execution of the instruction R6 ← R4 + R5. It shows a sequence of six steps. Steps 1 through 3 are grouped by a brace on the right labeled 'captación' (capture), indicating the time required to read from memory and load the MDR. Steps 4 through 6 are grouped by another brace on the right labeled 'ejecución' (execution), indicating the time required to perform the addition and store the result.

Unidad de control

■ Camino de datos

- Unidad de procesamiento y unidad de control
- Unidad de procesamiento con un bus
- Unidad de procesamiento con múltiples buses

■ Unidades de control cableadas y microprogramadas

- Diseño de una UC cableada
- Ejemplo de UC cableada
- Concepto de UC microprogramada

■ Control microprogramado

- Formato de las microinstrucciones
- Nanoprogramación
- Secuenciamiento de microinstrucciones
- Ejemplo de arquitectura microprogramada

Unidad de control

■ Señales de entrada a la UC:

- Señal de reloj
- Instrucción actual (codop, campos de direccionamiento,...)
- Estado de la unidad de proceso
- Señales externas (por ej. interrupciones)

■ Señales de salida de la UC:

- Señales que gobiernan la unidad de procesamiento:
 - Carga de registros
 - Incremento de registros
 - Desplazamiento de registros
 - Selección de entradas de multiplexores
 - Selección de operaciones de la ALU ...
- Señales externas
 - Por ej. lectura/escritura en memoria

Tipos de unidades de control

■ Existen dos formas de diseñar la UC:

- Control fijo o cableado (“hardwired”)
 - Se emplean métodos de diseño de circuitos digitales secuenciales a partir de diagramas de estados.
 - El circuito final se obtiene conectando componentes básicos como puertas y biestables, aunque más a menudo se usan PLA.
- Control microprogramado
 - Todas las señales que se pueden activar simultáneamente se agrupan para formar palabras de control, que se almacenan en una memoria de control (normalmente ROM).
 - Una instrucción de lenguaje máquina se transforma sistemáticamente en un programa (microprograma) almacenado en la memoria de control.
 - Mayor facilidad de diseño para instrucciones complejas
 - Método estándar en la mayoría de los CISC.

Diseño de una UC cableada

- **Se diseña mediante puertas lógicas y biestables siguiendo uno de los métodos clásicos de diseño de sistemas digitales secuenciales ya conocidos (TOC)**
 - El diseño es laborioso y difícil de modificar debido a la complejidad de los circuitos.
 - Suele ser más rápida que la misma UC microprogramada.
 - Se utilizan PLA (matrices lógicas programables) para llevar a cabo la implementación.

Debido a las modernas técnicas de diseño y a RISC ha tomado nuevo auge la realización de UC cableadas

Diseño de una UC cableada

■ Técnicas de diseño por computador (CAD) para circuitos VLSI (compiladores de silicio)

- Resuelven automáticamente la mayor parte de las dificultades de diseño de lógica cableada.
- Generan directamente las máscaras de fabricación de circuitos VLSI a partir de descripciones del comportamiento funcional del circuito en un lenguaje de alto nivel.



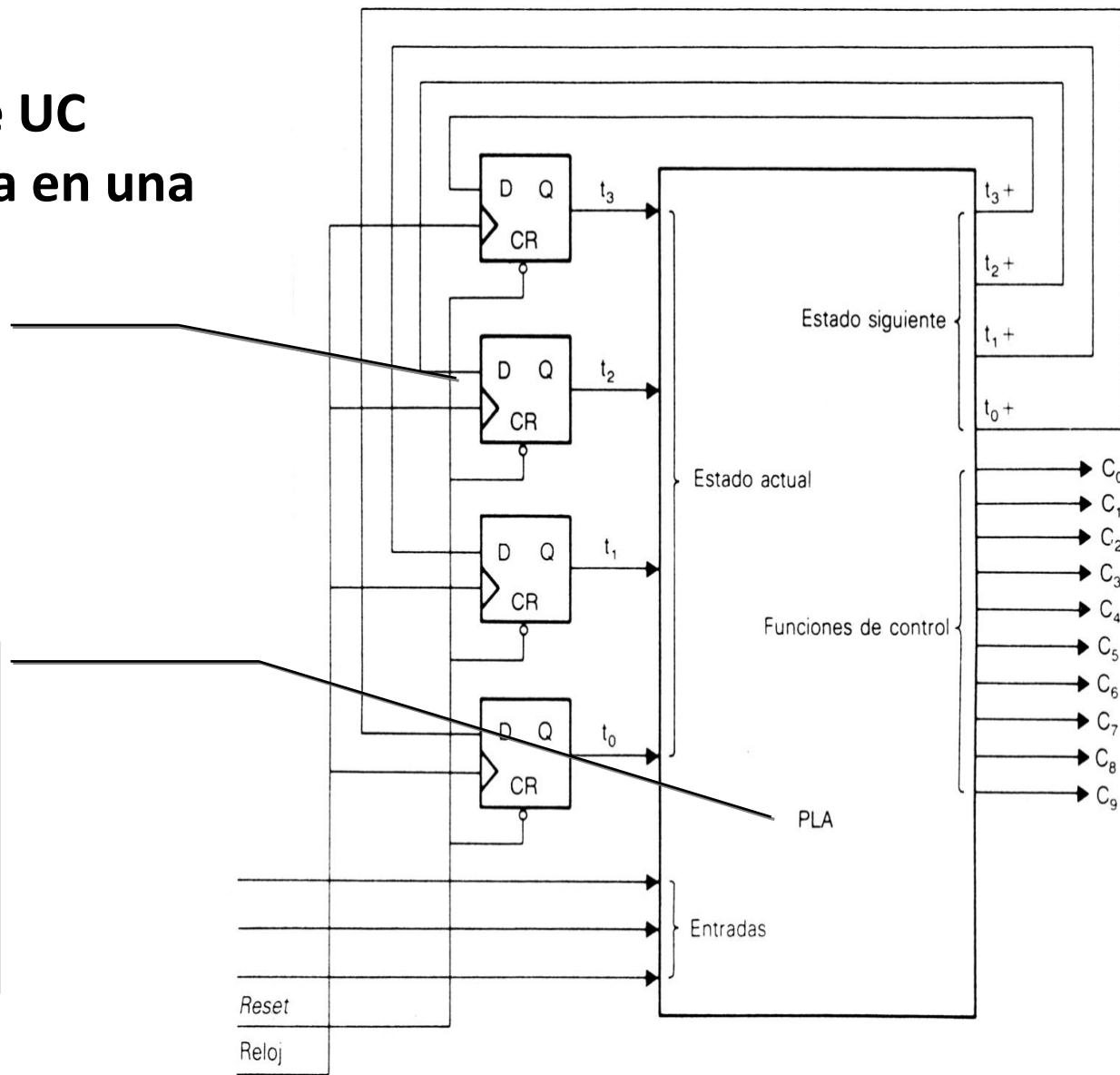
Debido a las modernas técnicas de diseño y a RISC ha tomado nuevo auge la realización de UC cableadas

Diseño de una UC cableada

- Organización de UC cableada basada en una PLA:

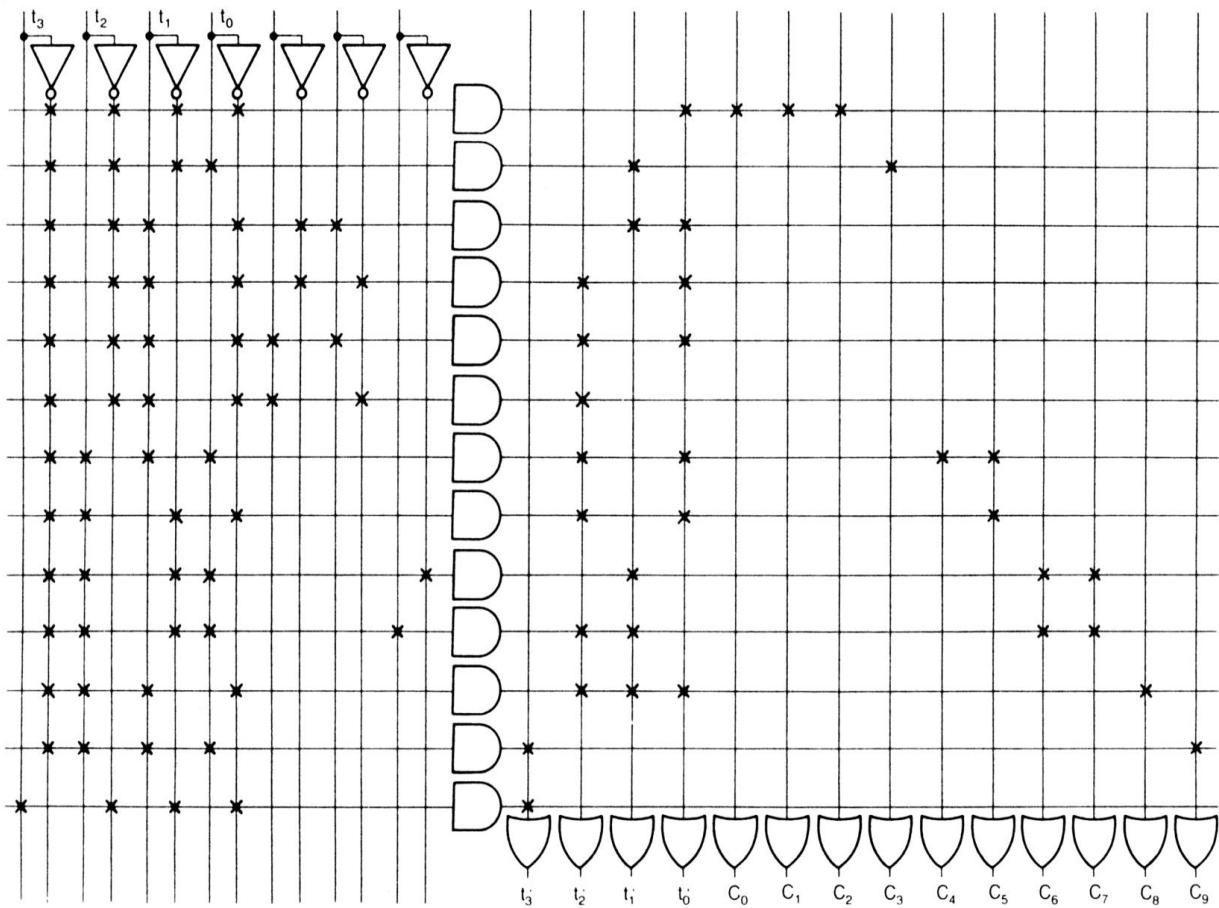
Los biestables contienen la información relativa al estado en que se encuentra el sistema

La PLA utiliza esta información de estado, junto con las entradas externas, para generar el siguiente estado



Diseño de una UC cableada

- Organización de UC cableada basada en una PLA:



Ventajas:

- ✓ Minimización del esfuerzo de diseño.
- ✓ Mayor flexibilidad y fiabilidad.
- ✓ Ahorro de espacio y potencia.

Ejemplo de UC cableada

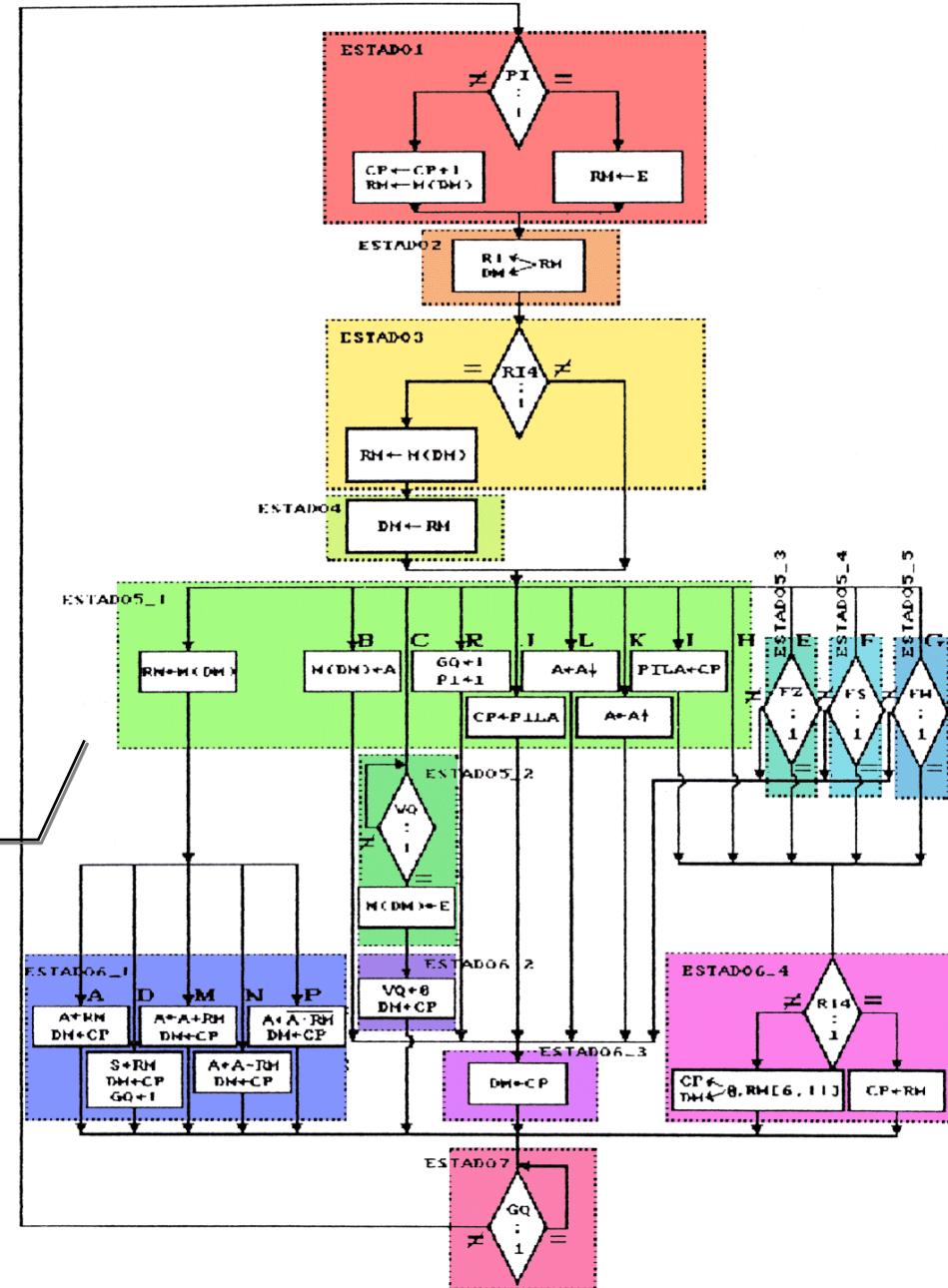
- **Implementación de una unidad de control cableada sencilla (ODE)**
- **Pasos a seguir para llegar al diseño físico:**
 1. Definir una máquina de estados finitos
 2. Describir dicha máquina en un lenguaje de alto nivel
 3. Generar la tabla de verdad para la PLA
 4. Minimizar la tabla de verdad
 5. Diseñar físicamente la PLA partiendo de la tabla de verdad

Ej. de UC cableada

■ 1. Definir una máquina de estados finitos

- Dado el diagrama de flujo de la UC de ODE, detallamos éste como un conjunto finito de estados y transiciones entre ellos.

Modelo Mealy: salidas dependen de entradas y estado presente



Ej. de UC cableada

■ 2. Describir dicha máquina en un lenguaje de alto nivel

- El lenguaje concreto depende del programa que utilicemos para “compilar” la descripción de la máquina.
- Estos lenguajes tienen sentencias para definir:
 - entradas y salidas
 - estados y transiciones condicionales e incondicionales entre estados

```

-- ***** UNIDAD DE CONTROL DE ODE *****
-- Definicion de líneas de entrada y de salida

INPUTS: RIO R11 R12 R13 R14 PI V0 G0 FS FW FZ V;
OUTPUTS: A B C D E F G H I J K L M N P R S Y T U R1 R2 R3 R4 R5 R6 INSTR_C INSTR_D INSTR_R;

-- Estado de reset

RESET ON V TO STATE ESTADO07 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=?);

-- Definicion de estados

ESTADO01: IF PI THEN ESTADO02 (A=? E=? F=? G=? I J=0 K L=? M=? N=? R=? S=? Y=? R1)
          ELSE ESTADO02 (A=? D E=? F=? G=? I J=1 K L=? M=0 N=0 R=? S=? Y=? R1);

ESTADO02: GOTO ESTADO03 (A=? E=? F=? G=0 H I J=? L=0 M=1 N=? R=? S=? Y=? U R2);

ESTADO03: IF R14 THEN ESTADO04 (A=? E=? F=? G=? I J=1 K L=? M=? N=? R=? S=? Y=? R3)
          ELSE ESTADO05_1 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? R3);

ESTADO04: GOTO ESTADOS_1 (A=? E=? F=? G=0 H I J=? L=1 M=1 N=? R=? S=? Y=? R4 INSTR_C=0 INSTR_D=0 INSTR_R=0);

ESTADOS_1: CASE (RIO R11 R12 R13)
    0 0 0 1 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=0 N=1 R=? S=? Y=? RS);
    0 0 1 0 => ESTADO05_2 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    1 1 0 1 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS INSTR_R);
    1 0 0 1 => ESTADO06_3 (A=? B C E=0 F=1 G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    1 0 1 1 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=1 S=0 Y=0 T RS);
    1 0 0 0 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 0 1 => ESTADO05_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 0 0 => ESTADO05_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 1 0 => ESTADO05_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 1 1 => ESTADO05_5 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 0 1 => ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    1 0 0 0 => ESTADO06_4 (A=? B E=1 F=0 G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    => ESTADO06_1 (A=? E=? F=? G=? I J=1 K L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_2: IF V0 THEN ESTADO06_2 (A=? E=? F=? G=? I J=? L=? M=0 N=0 R=? S=? Y=? RS)
             ELSE LOOP (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_3: IF FZ THEN ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS)
             ELSE ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_4: IF FS THEN ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS)
             ELSE ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_5: IF FW THEN ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS)
             ELSE ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO06_1: CASE (RIO R11 R12 R13)
    0 0 0 0 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=1 S=1 Y=0 T R6);
    0 0 1 1 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? P R=? S=? Y=? R6 INSTR_D);
    1 1 0 0 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=0 S=0 Y=1 T R6);
    1 1 0 1 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=0 S=1 Y=0 T R6);
    1 1 1 0 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=0 S=1 Y=1 T R6);
    ENCASE => ANY (A=? B=? C=? D=? E=? F=? G=? H=? I=? J=? K=? L=? M=? N=? P=? R=? S=? Y=? T=? U=? R1=? R2=? R3=? R4=? RS=? R6=?);

ESTADO06_2: GOTO ESTADO07 (A=? E=? F=? G=1 H I J=? L=? M=? N=? R=? S=? Y=? R6 INSTR_C);

ESTADO06_3: GOTO ESTADO07 (A=? E=? F=? G=0 H I J=? L=? M=? N=? R=? S=? Y=? R6);

ESTADO06_4: IF R14 THEN ESTADO7 (A=0 C E=? F=? G=0 I J=? L=1 M=1 N=? R=? S=? Y=? R6)
             ELSE ESTADO7 (A=0 C E=? F=? G=0 H I J=? L=0 M=1 N=? R=? S=? Y=? R6);

ESTADO07: IF GO THEN LOOP (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=?)
             ELSE ESTADO1 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=?);

```

Ej. de UC cableada

■ 3. Generar tabla de verdad necesaria para PLA

- Según la descripción que hayamos hecho de la máquina de estados...
 - podemos usar un programa que use el modelo **Mealy**
 - salidas dependen de entradas y de estado presente
 - o uno que use el modelo **Moore**
 - salidas dependen exclusivamente de estado actual

Ej. de UC cableada

■ 4. Minimizar la tabla de verdad

- Mediante un programa que utiliza algoritmos heurísticos rápidos.

41 términos producto (PLA más pequeña)



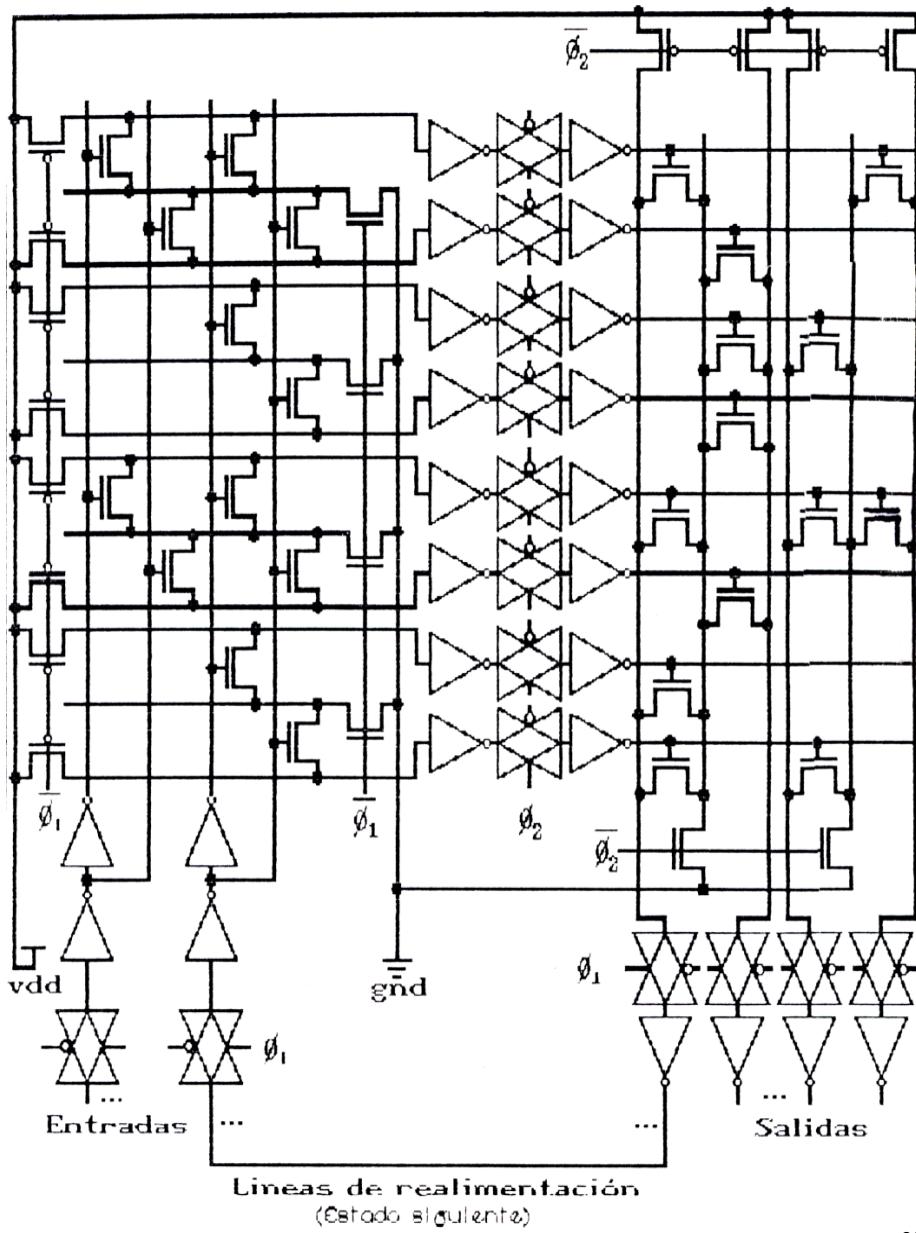
Ej. de UC cableada

- **5. Diseñar físicamente la PLA partiendo de la tabla de verdad minimizada**
 - Automáticamente:
 - Mediante un programa especial para diseño de layouts de PLA.
 - Semiautomáticamente:
 - Diseñando mediante un programa de CAD de circuitos VLSI cada una de las celdas que, repetidas convenientemente, forman la PLA.
 - Dando una especificación de cómo han de colocarse (tabla de verdad minimizada).

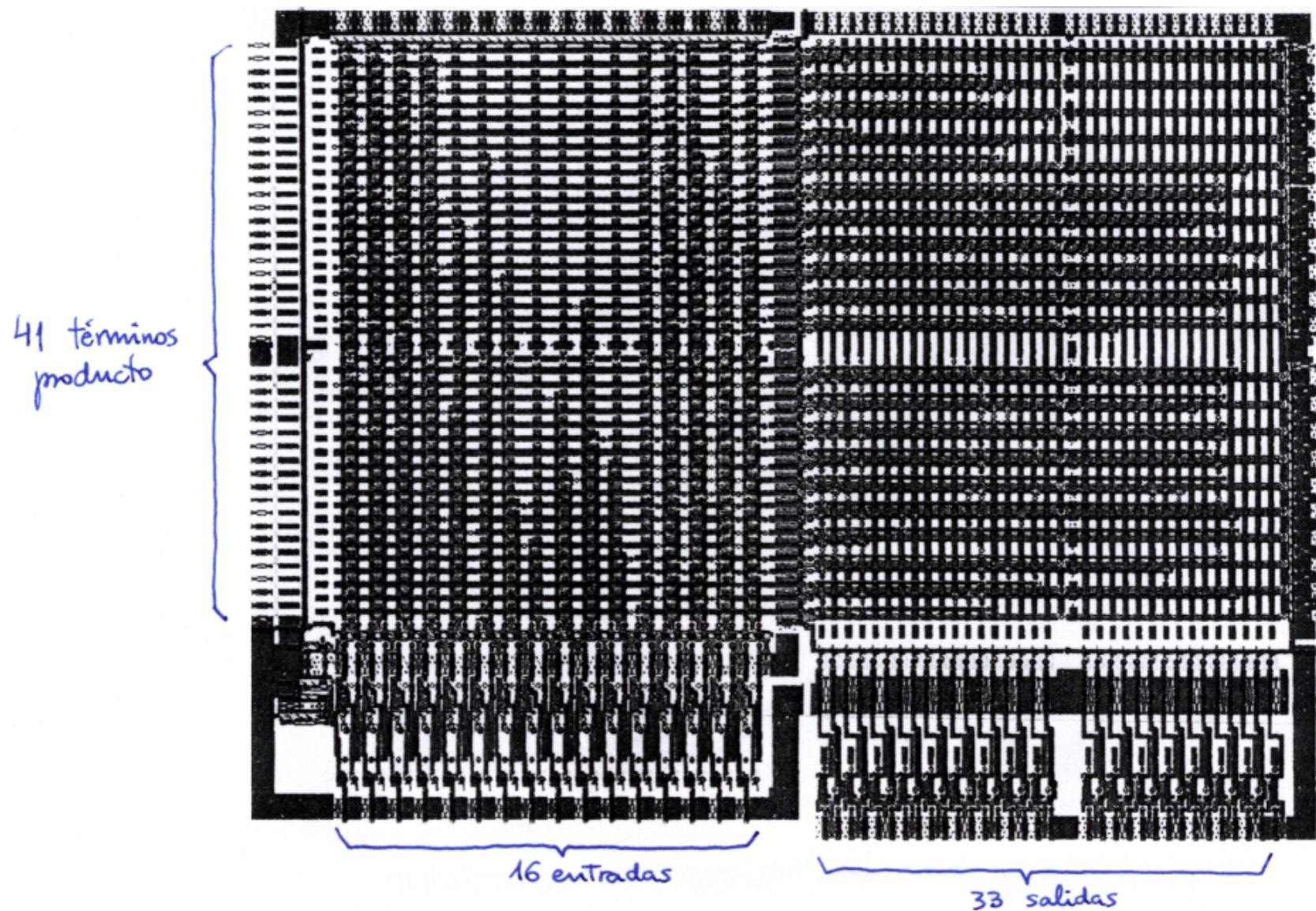
Ej. de UC cableada

- Esquema simplificado de la PLA usada para la UC de ODE

- CMOS de dos fases de reloj
 - No hacen falta biestables de estado siguiente
 - $\Phi_1=1$ se leen las entradas y se precarga el plano AND
 - $\Phi_2=1$ se evalúa el plano AND y se precarga el plano OR
 - $\Phi_2=0$ se evalúa el plano OR



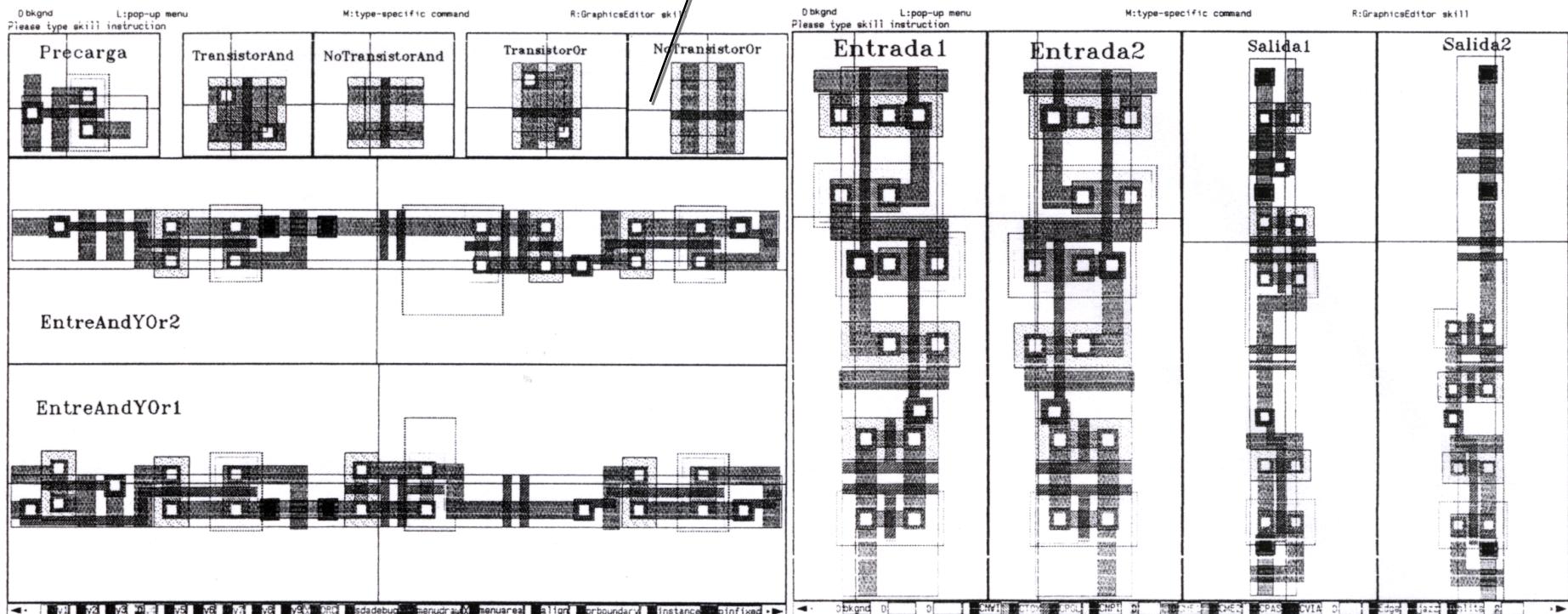
Ej. de UC cableada



Ej. de UC cableada

- PLA diseñada semiautomáticamente

Detalle de las celdas básicas que constituyen la PLA diseñadas con un programa de CAD.
Un programa puede unirlas de acuerdo con el archivo de la tabla de verdad minimizada.



Concepto de UC microprogramada

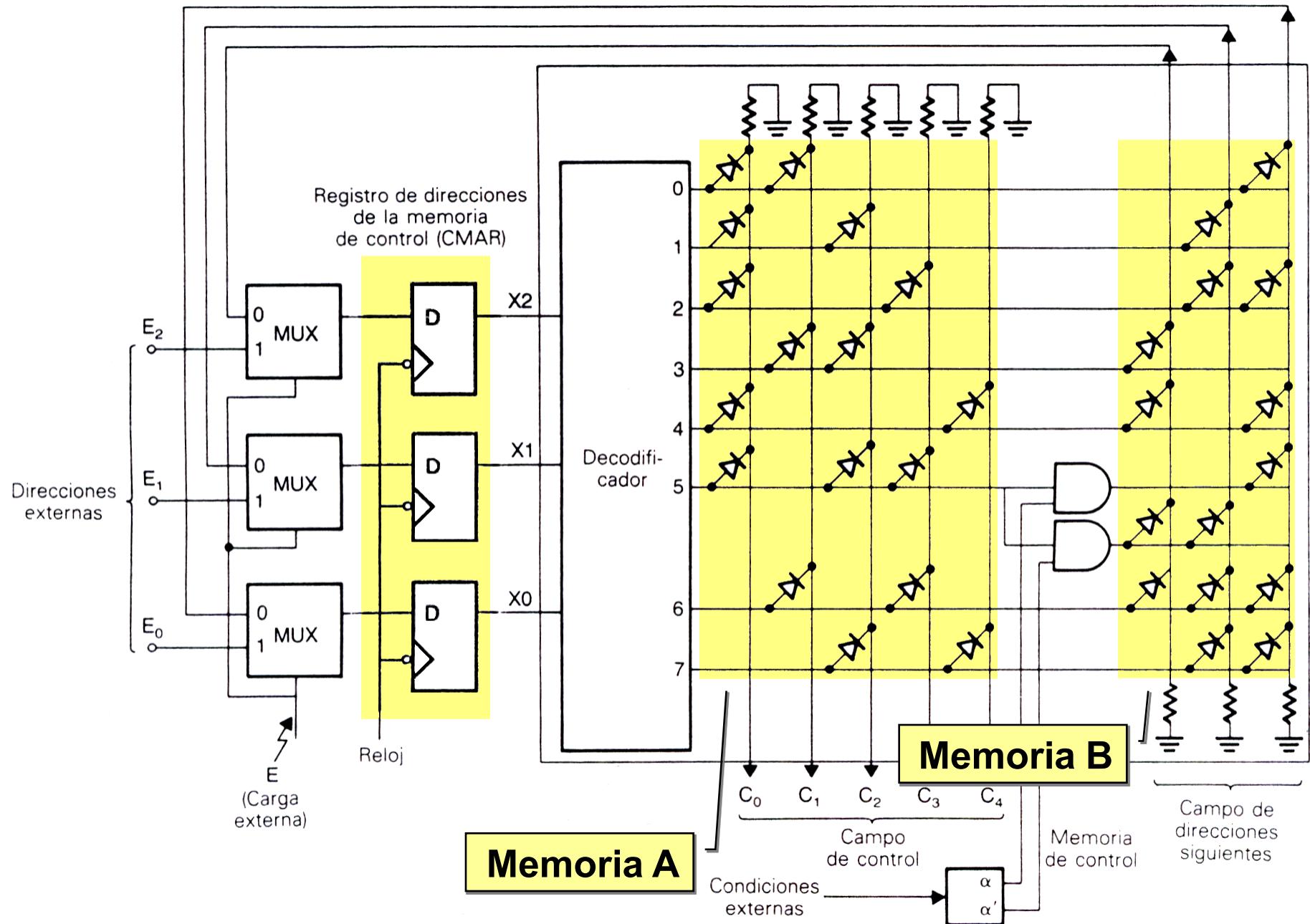
■ Idea básica:

- Emplear una memoria (de control) para almacenar las señales de control de los períodos de cada instrucción

■ Origen histórico

- Maurice V. Wilkes (1913-2010) en 1951-1953 propone el siguiente esquema:
 - Dos memorias A y B, construidas con matrices de diodos.
 - Las señales de control se encuentran almacenadas en la memoria A.
 - La memoria B contiene la dirección de la siguiente microinstrucción.
 - Se permiten microbifurcaciones condicionales, mediante un biestable y un decodificador que selecciona entre dos direcciones de la matriz B.
- Más info.: <http://www.cs.clemson.edu/~mark/uprog.html>





Concepto de UC microprogramada

■ Definiciones:

- **Microinstrucción**: cada palabra de la memoria de control
- **Microprograma**: conjunto ordenado de microinstrucciones cuyas señales de control constituyen el cronograma de una (macro)instrucción del lenguaje máquina.
- Ejecución de un microprograma: lectura en cada pulso de reloj de una de las microinstrucciones que lo forman, enviando las señales leídas a la unidad de proceso como señales de control.
- **Microcódigo**: conjunto de los microprogramas de una máquina.

Concepto de UC microprogramada

■ Ventajas de la microprogramación:

- Simplicidad conceptual.
 - La información de control reside en una memoria.
- Se pueden incluir, sin dificultades, instrucciones complejas, de muchos ciclos de duración.
 - El único límite es el tamaño de la memoria de control.
- Las correcciones, modificaciones y ampliaciones son mucho más fáciles de hacer que en una unidad de control cableada.
 - No hay que rediseñar toda la unidad, sino sólo cambiar el contenido de algunas posiciones de la memoria de control.
- Permite construir computadores con varios juegos de instrucciones, cambiando el contenido de la memoria de control (si es RAM permite emular otros computadores).

Concepto de UC microprogramada

- **Desventaja de la microprogramación:**
 - Lentitud frente a cableada, debido a una menor capacidad de expresar paralelismo de las microinstrucciones.

Unidad de control

■ Camino de datos

- Unidad de procesamiento y unidad de control
- Unidad de procesamiento con un bus
- Unidad de procesamiento con múltiples buses

■ Unidades de control cableadas y microprogramadas

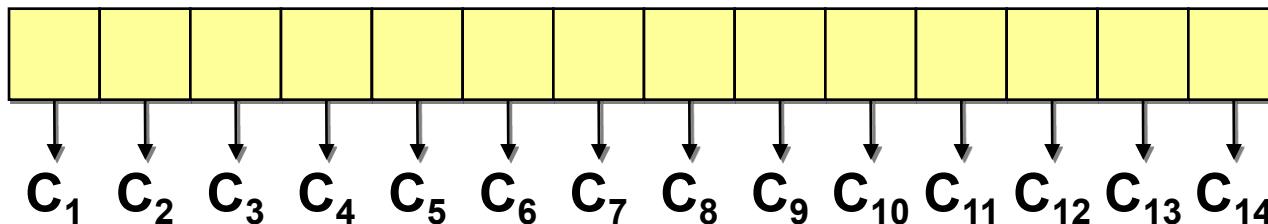
- Diseño de una UC cableada
- Ejemplo de UC cableada
- Concepto de UC microprogramada

■ Control microprogramado

- Formato de las microinstrucciones
- Nanoprogramación
- Secuenciamiento de microinstrucciones
- Ejemplo de arquitectura microprogramada

Formato de las microinstrucciones

- Las señales de control que gobiernan un mismo elemento del datapath se suelen agrupar en campos.
 - Ejemplos:
 - señales triestado que controlan el acceso a un bus
 - señales de operación de la ALU
 - señales de control de la memoria
- Formato no codificado:
 - Hay un bit para cada señal de control de un campo

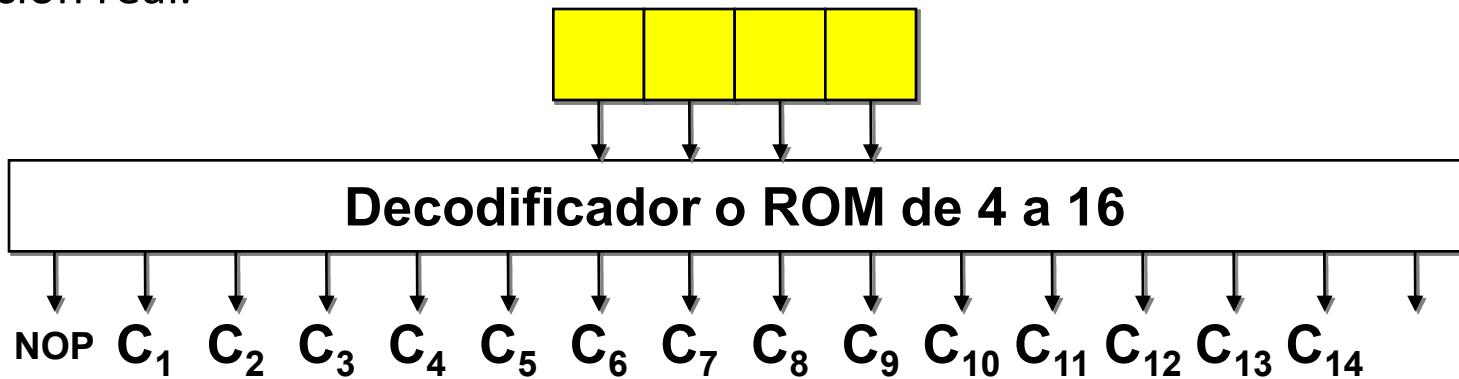


Formato de las microinstrucciones

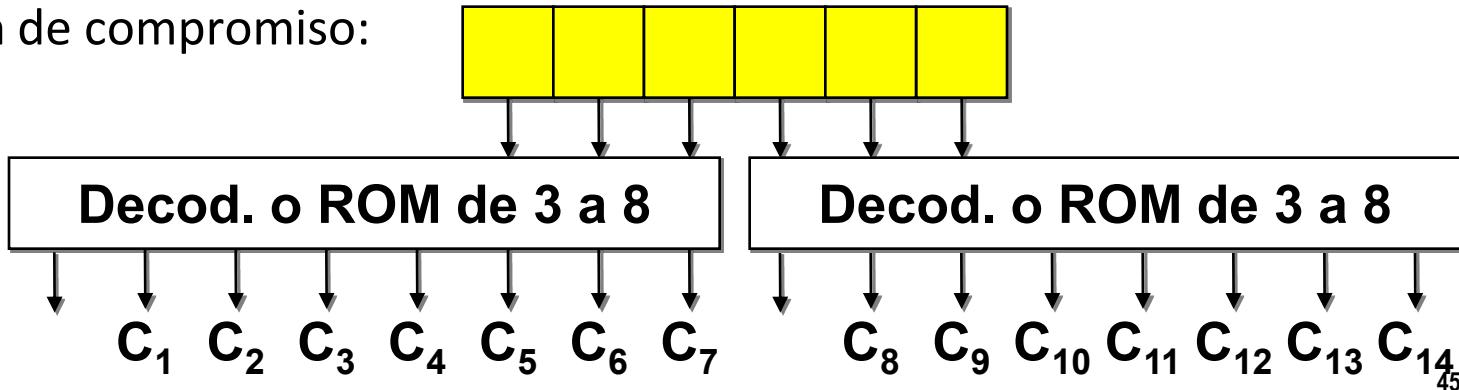
■ Formato codificado:

- Para acortar el tamaño de las microinstrucciones se codifican todos o alguno de sus campos.

Inconveniente: Hay que incluir decodificadores para extraer la información real.



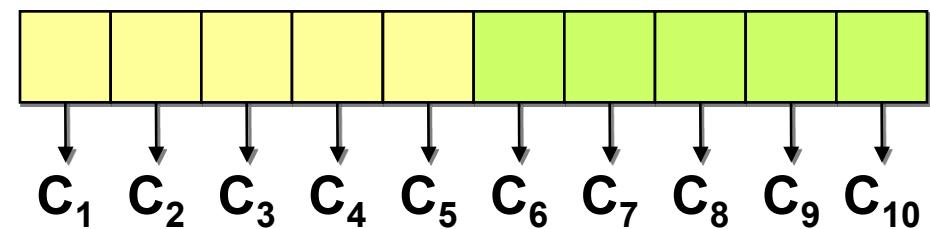
- Solución de compromiso:



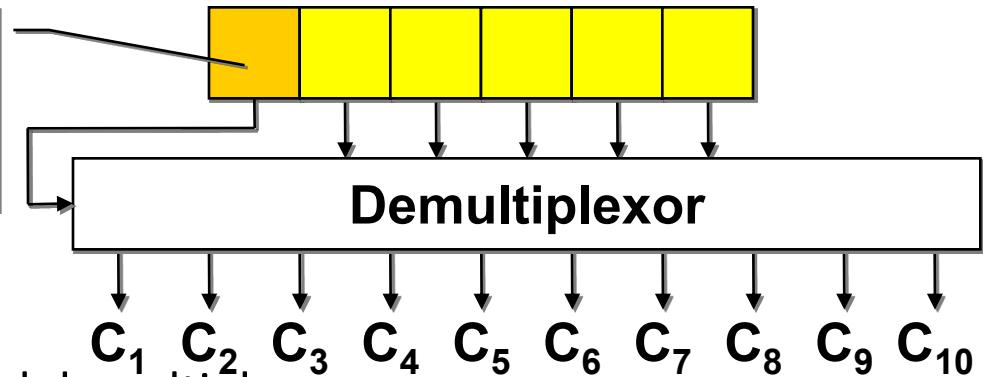
Formato de las microinstrucciones

■ Solapamiento de campos:

- Si sólo unas pocas señales de control están activas en cada ciclo, o existen con frecuencia señales excluyentes, que no se pueden activar simultáneamente...
- ...se puede acortar la longitud de las microinstrucciones solapando campos.



La señal adicional de control define si los bits corresponden al campo 1 o al campo 2



- Inconvenientes:
 - Retardo introducido por el demultiplexor.
 - Hace incompatibles las operaciones de los campos solapados.

Formato de las microinstrucciones

Micro-programación horizontal:

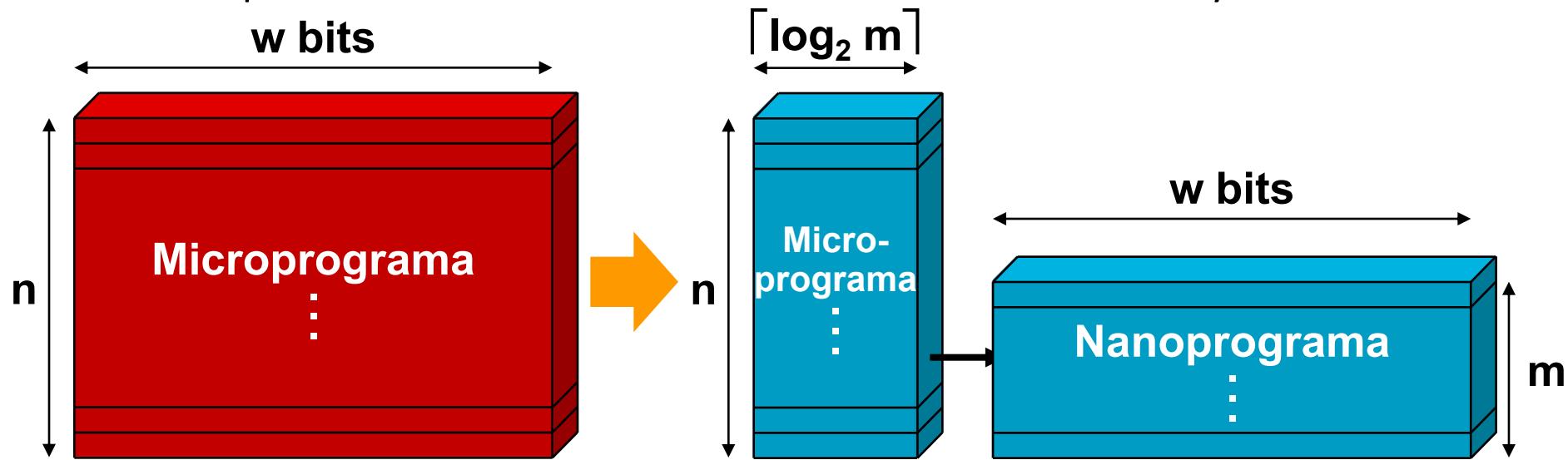
- Ninguna o escasa codificación
- ✓ Capacidad para expresar un alto grado de paralelismo en las microoperaciones a ejecutar (simultáneamente)
- ✗ Microinstrucciones largas

Micro-programación vertical:

- Mucha codificación
- ✓ Microinstrucciones cortas
- ✗ Escasa capacidad para expresar paralelismo (la longitud del programa se ve incrementada)

Nanoprogramación

- **Objetivo: reducir el tamaño de la memoria de control**
 - Implica una memoria a dos niveles: memoria de control y nanomemoria.



Microprograma original con n μinstrucciones de w bits. Tamaño = $n \cdot w$

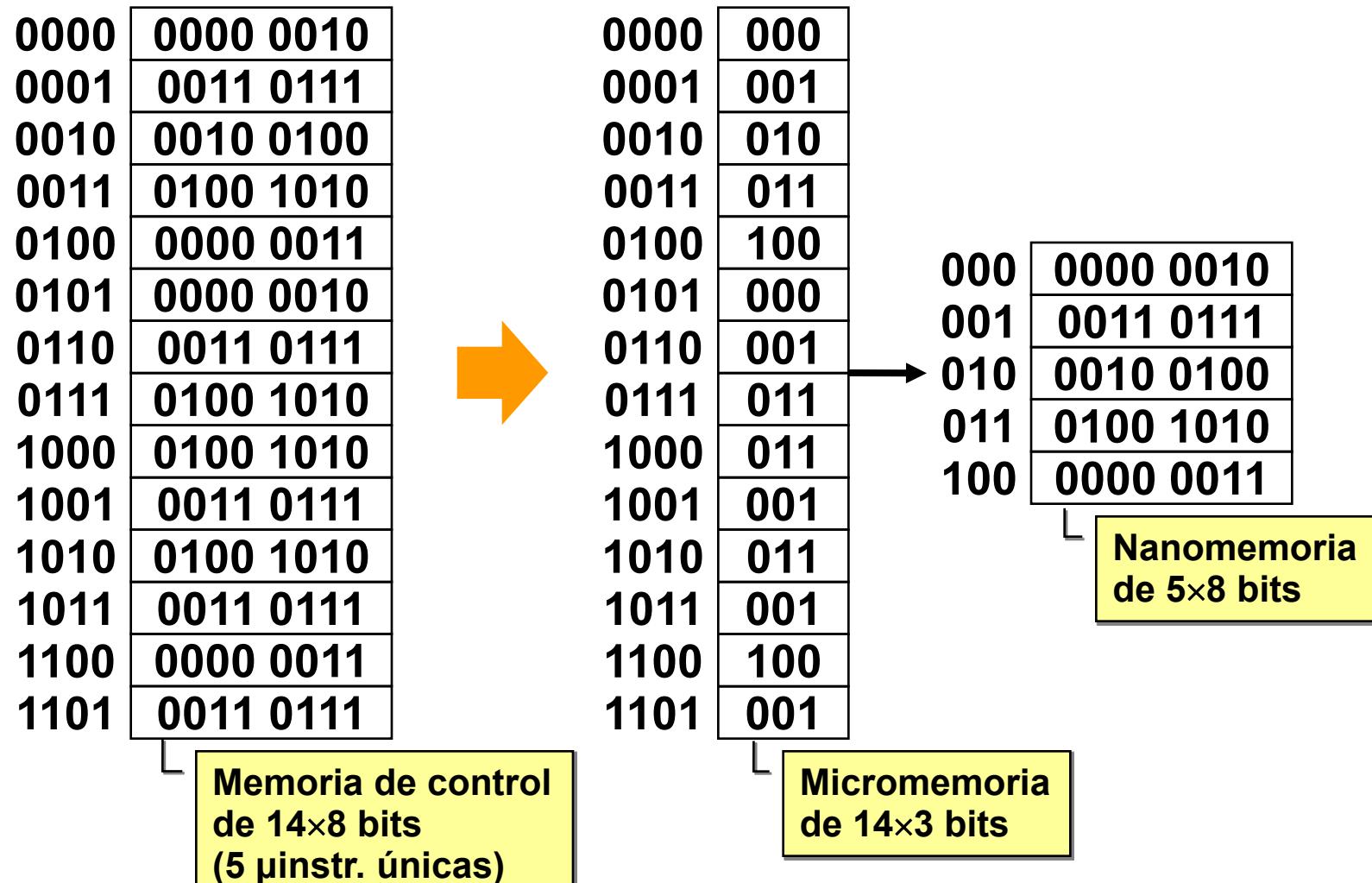
$m << n$ μinstrucciones únicas de 2^w posibles

**Se reemplaza cada μinstrucción por su dirección en la nanomemoria
Tamaño: $n \cdot \lceil \log_2 m \rceil$**

**Contiene las m μinstrucciones diferentes (cada una se incluye una sola vez).
Tamaño: $m \cdot w$**

Ahorro de memoria: $n \cdot w - (n \cdot \lceil \log_2 m \rceil + m \cdot w)$

Nanoprogramación. Ejemplo 1

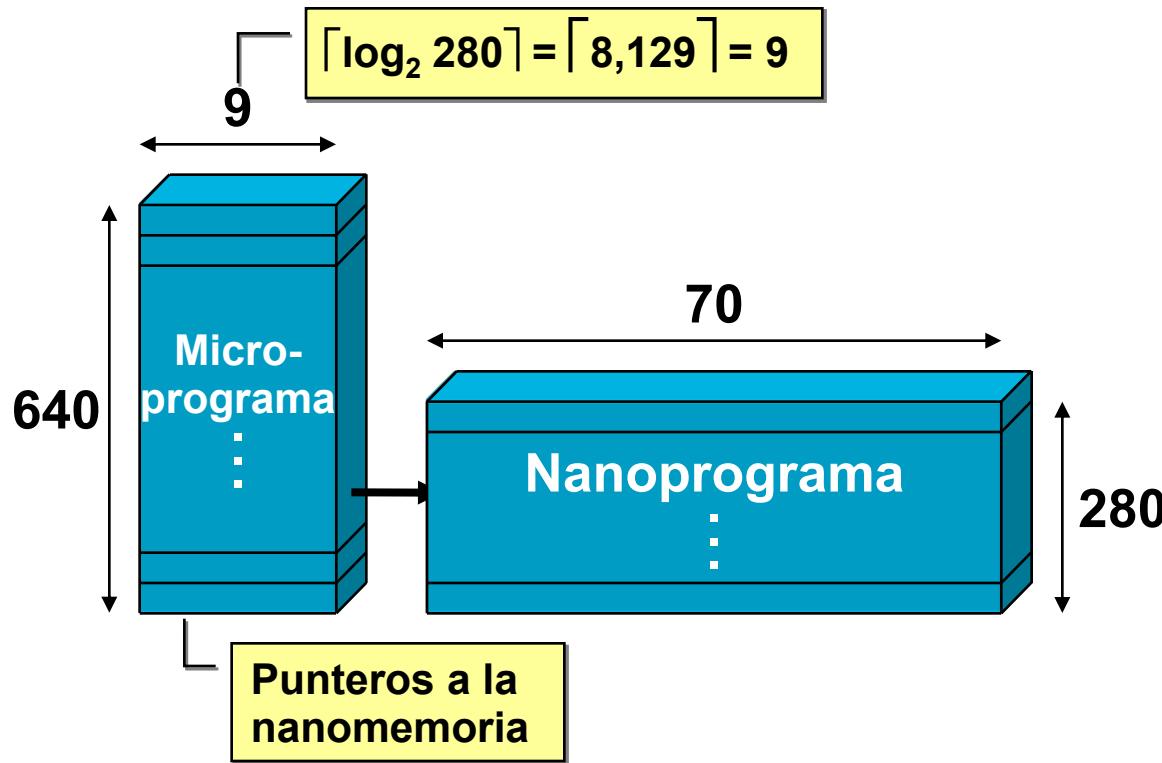


Ahorro de memoria: $14 \cdot 8 - (14 \lceil \log_2 5 \rceil + 5 \cdot 8) = 112 - 82 = 30$ bits (27%)

Nanoprogramación. Ejemplo 2

■ Estructura de la UC del Motorola 68000

- 640 microinstrucciones, de las cuales 280 son únicas



Ahorro de memoria: $640 \cdot 70 - (640 \cdot 9 + 280 \cdot 70) = 44800 - 25360 = 19440$ bits (43%)

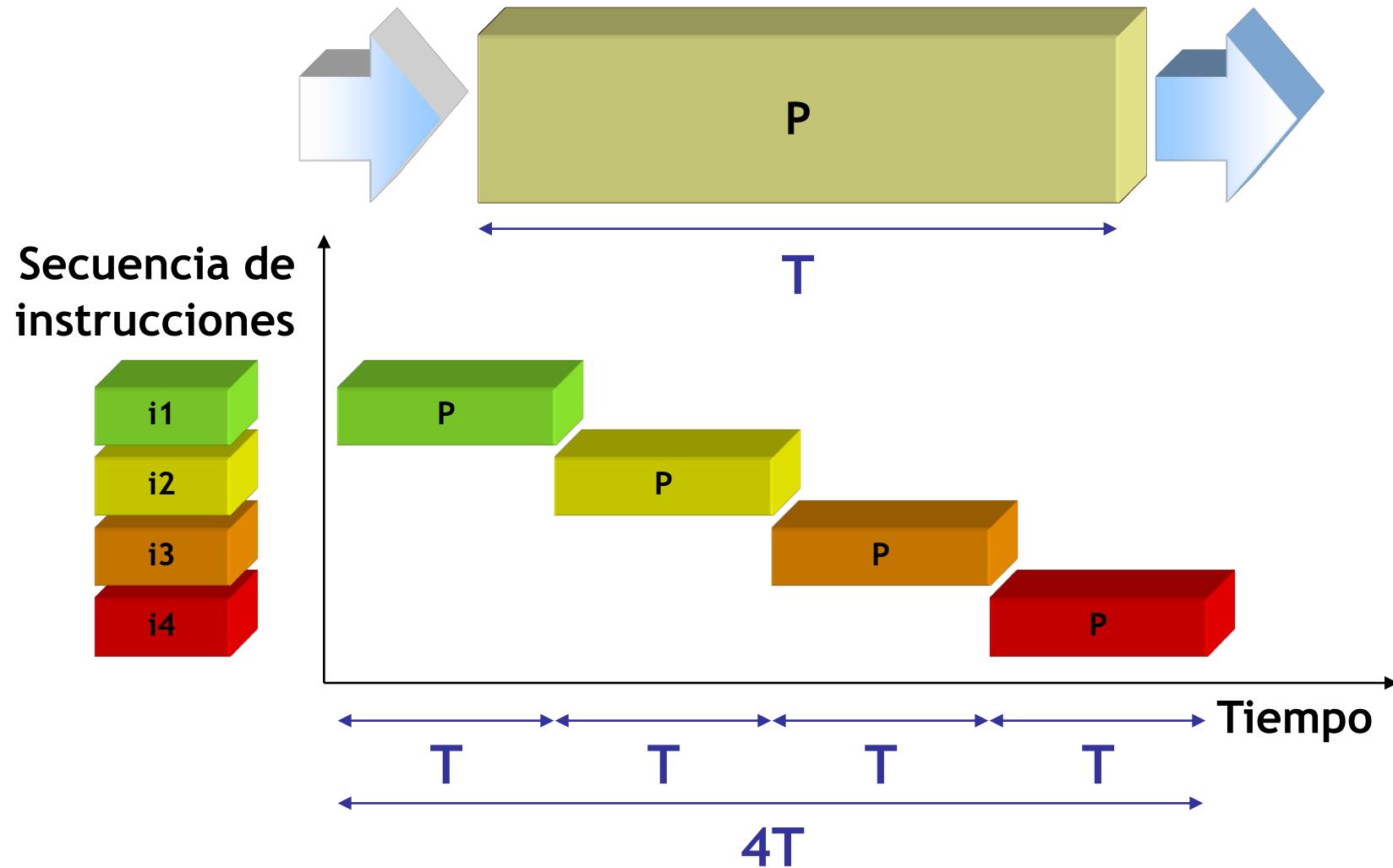
TEMA 4

Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

Concepto de segmentación

Procesador sin segmentar



Concepto de segmentación

¿Es posible incrementar la velocidad de procesamiento, al margen de mejoras tecnológicas, sin incrementar el número de procesadores?

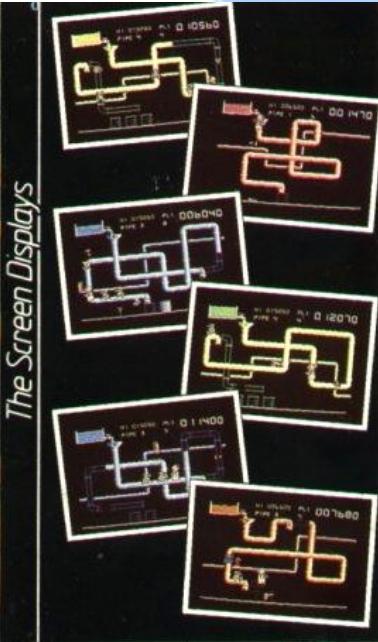
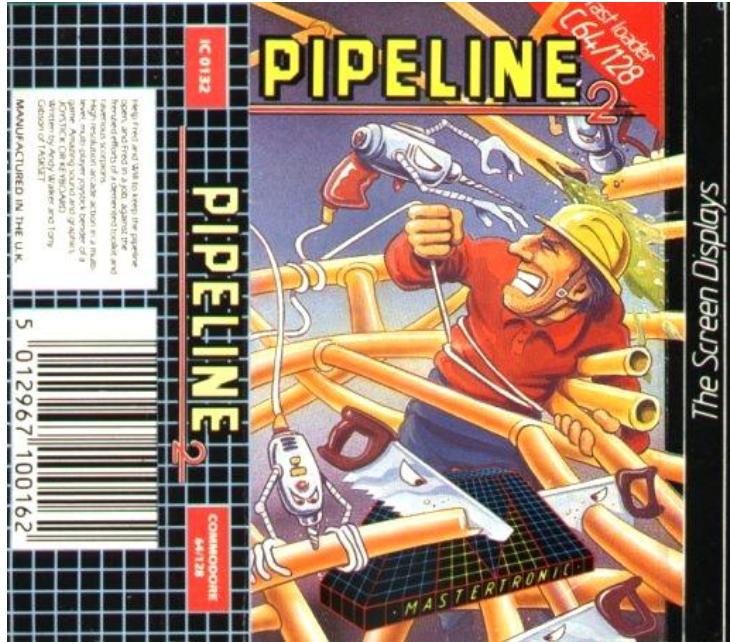


SOLUCIÓN



Segmentación de cauce
(*pipelining*)

Concepto de segmentación

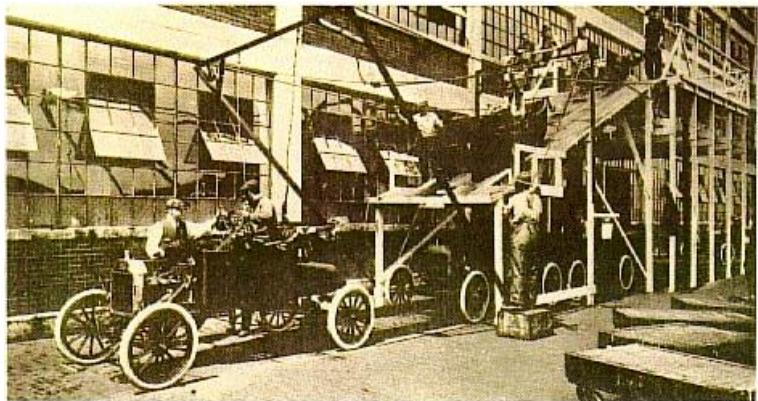


THE PIPELINE

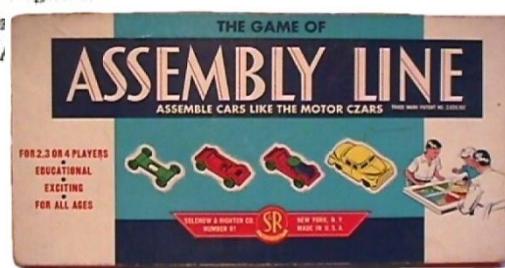
SOLUCIÓN

Segmentación de cauce
(*pipelining*)

Concepto de segmentación



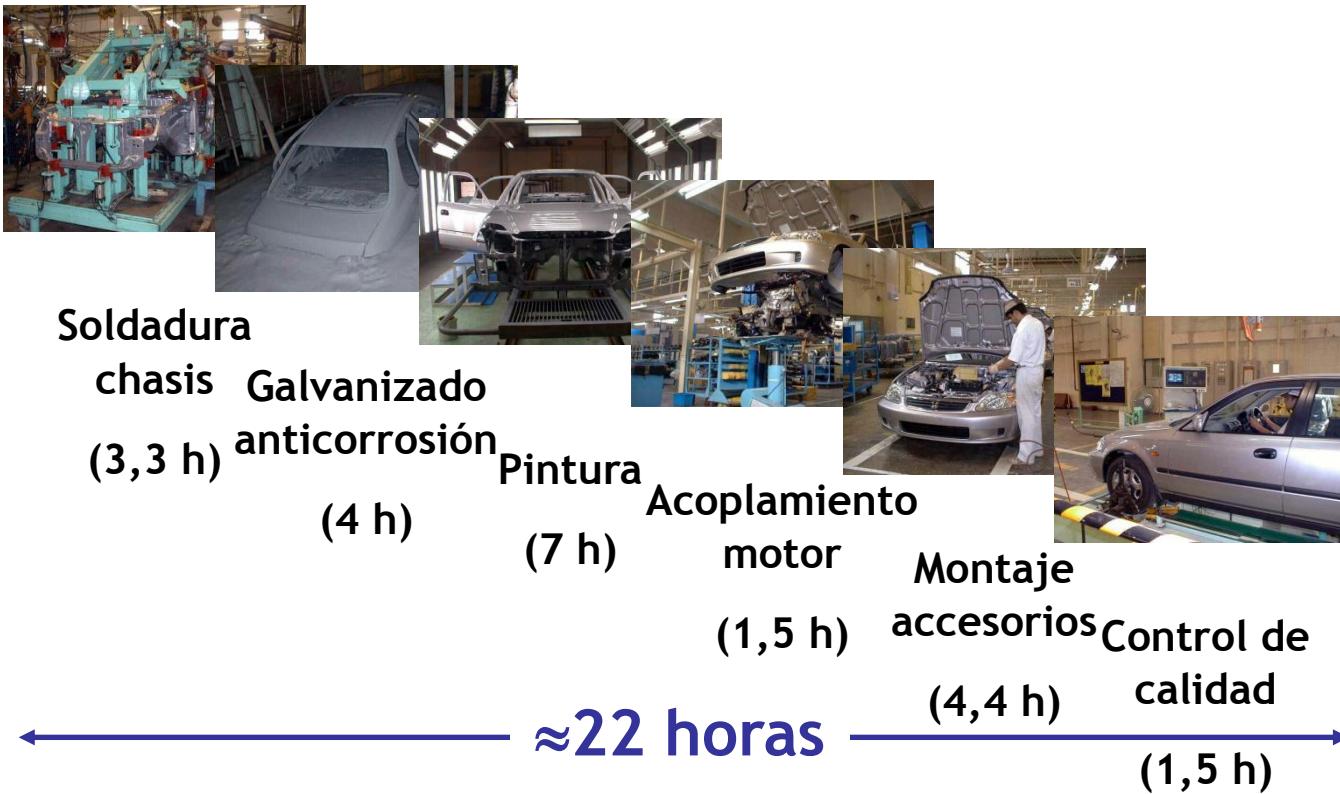
Montaje final del vehículo del modelo T en la fábrica de Highland Park (Detroit, EUA), propiedad de Ford. En ella se ensamblan grandes unidades constructivas (chasis y carrocería) que forman el vehículo.



SEGMENTACIÓN EN UNA FÁBRICA DE AUTOMÓVILES

Cadena de montaje

Concepto de segmentación

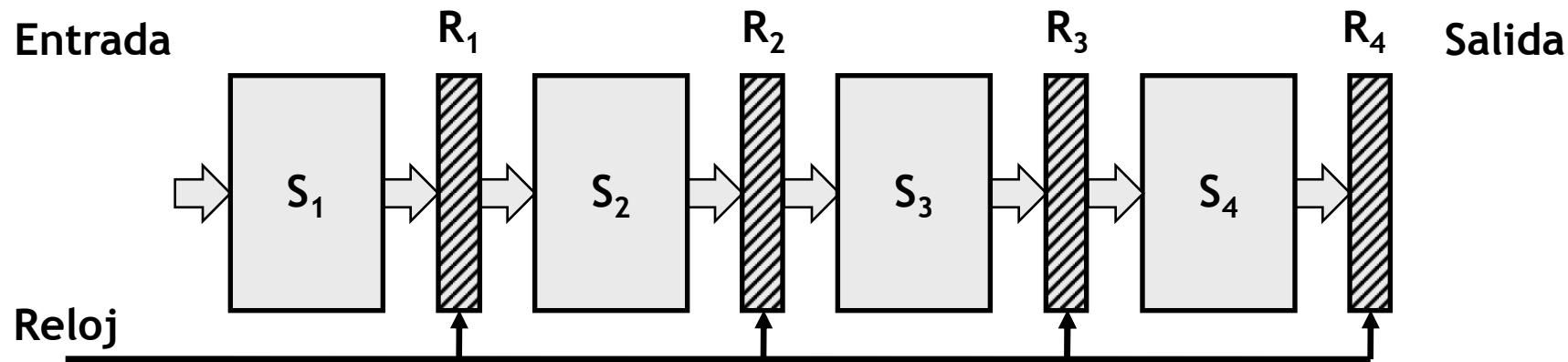


SEGMENTACIÓN EN UNA FÁBRICA DE AUTOMÓVILES
Subdividir el proceso en n etapas, permitiendo
el solapamiento en la fabricación de automóviles

Concepto de segmentación

S_i : etapa de segmentación i -ésima

R_i : registro de segmentación de la etapa i -ésima

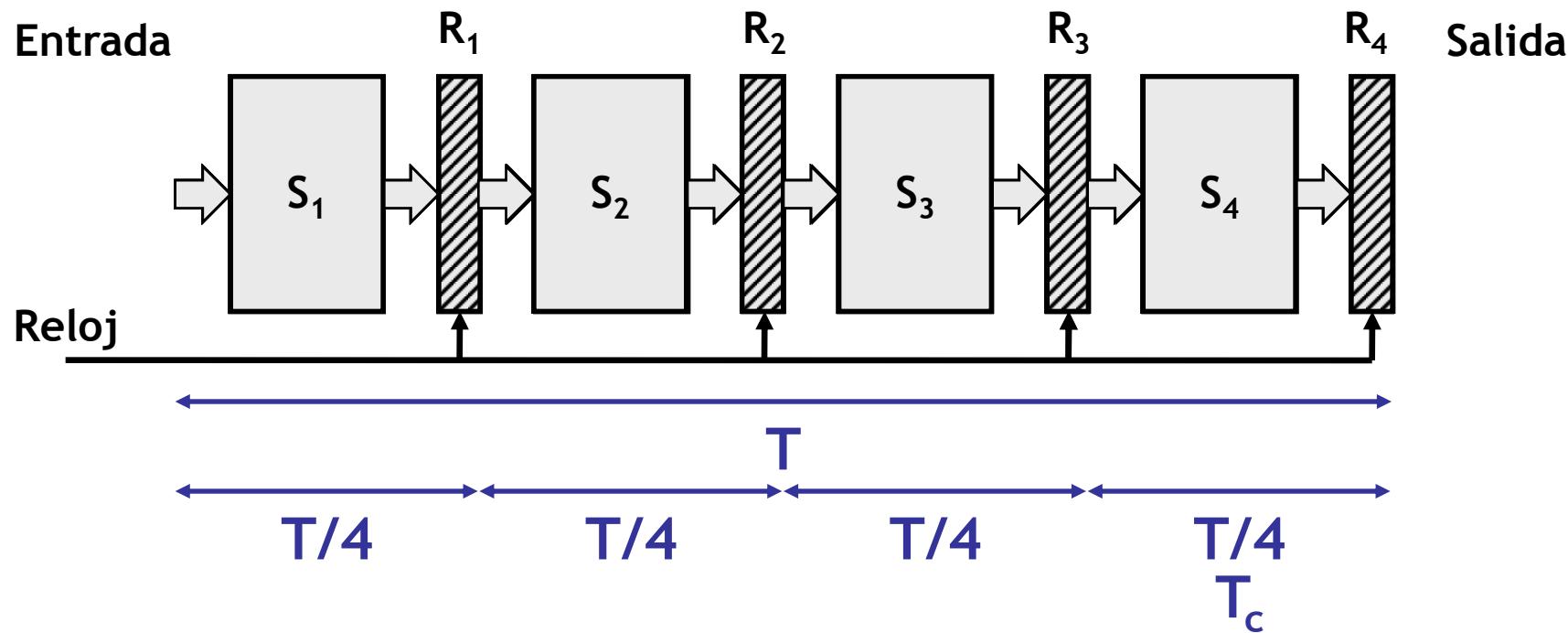


SEGMENTACIÓN EN UN PROCESADOR

Subdividir el procesador en n etapas, permitiendo el solapamiento en la ejecución de instrucciones

Concepto de segmentación

Las instrucciones entran por un extremo del cauce, son procesadas en distintas etapas y salen por el otro extremo.



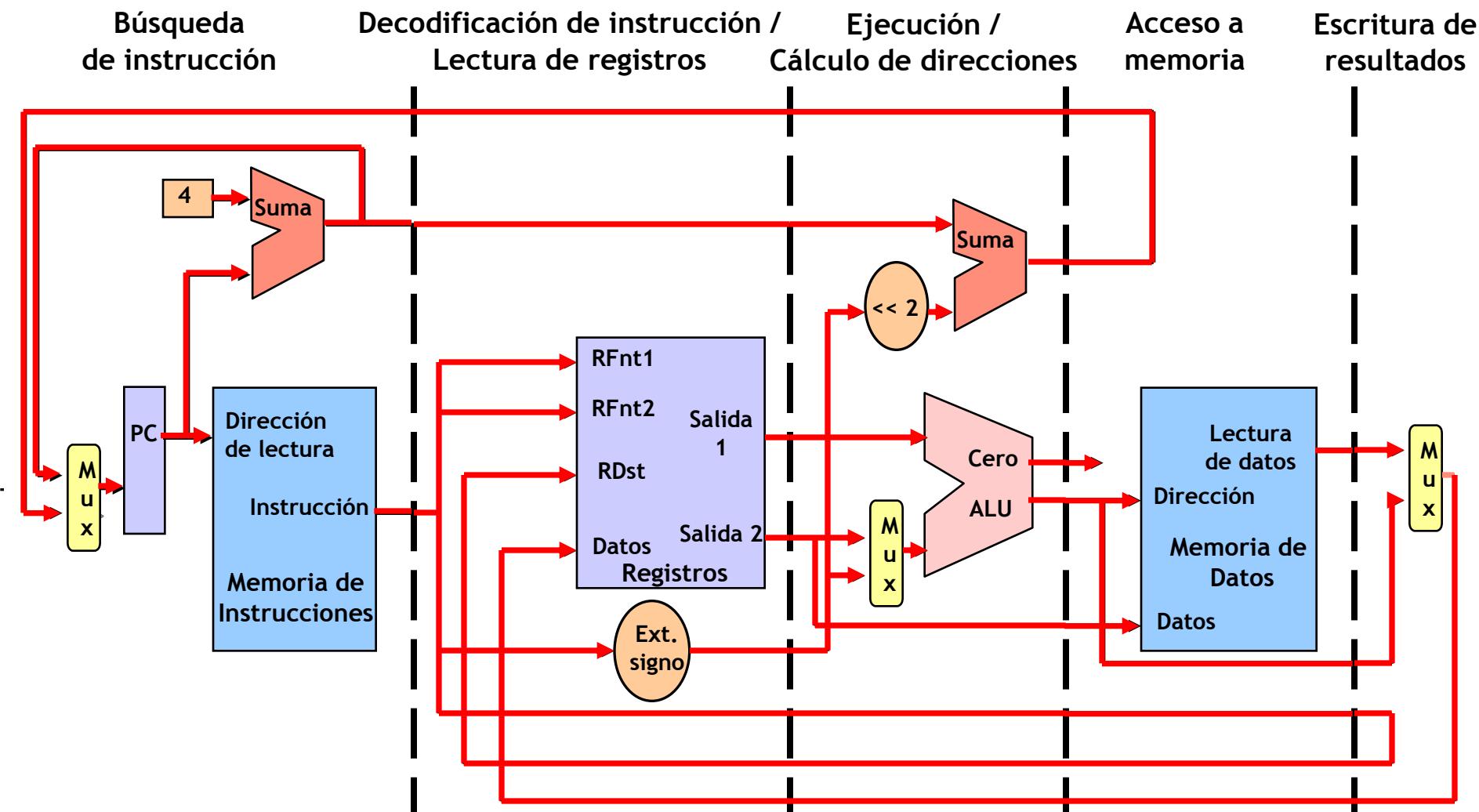
Cada instrucción individual se sigue ejecutando en un tiempo T ...

...pero hay varias instrucciones ejecutándose simultáneamente

Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

Ejemplo de segmentación



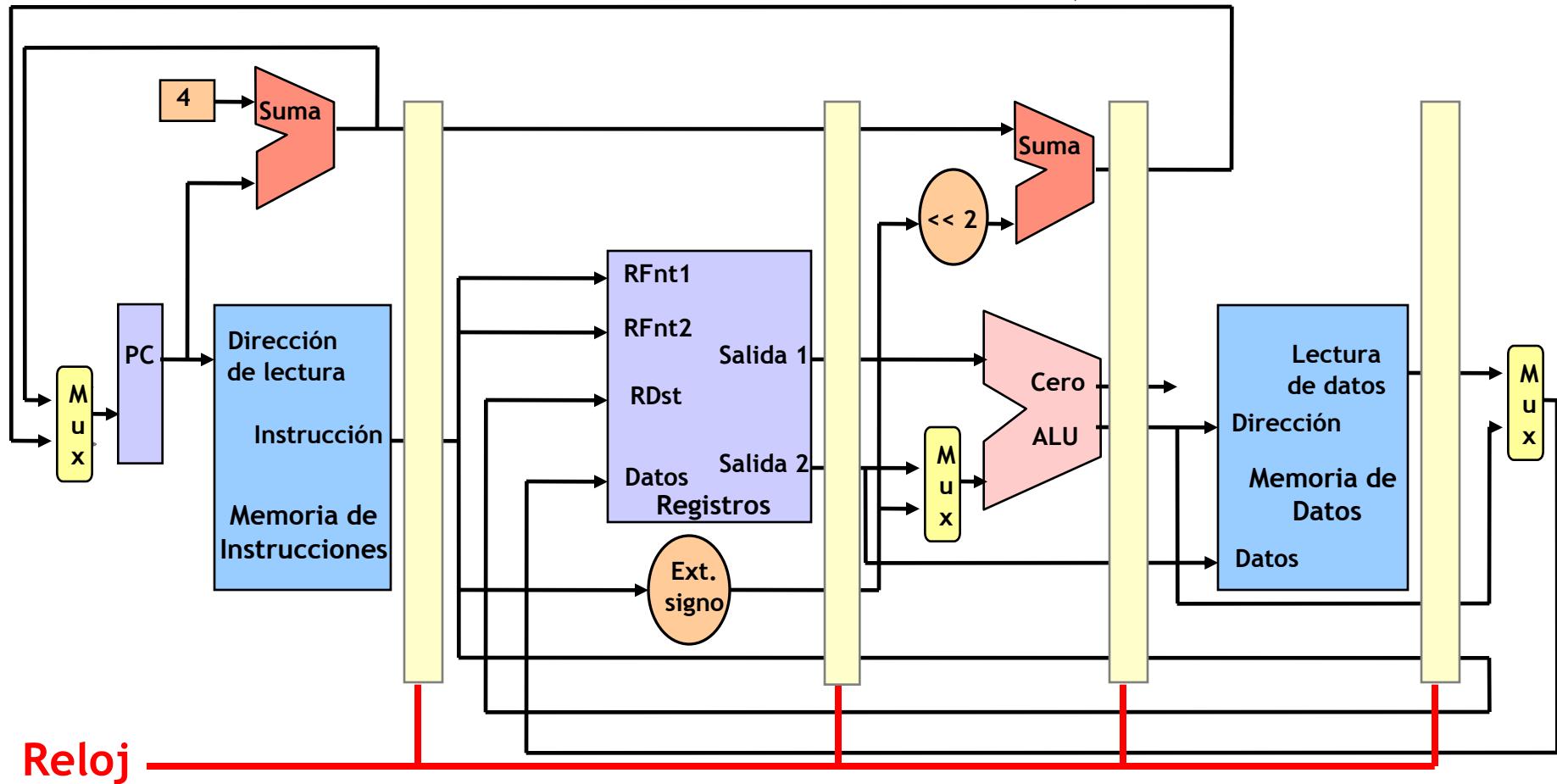
IF
(Instruction Fetch)

 ID
*(Instruction Decode
and register fetch)*

 EX
*(EXecute and
effective
address
calculation)*

 MEM
*(MEMory
access)*

 WB
*(Write
back)*



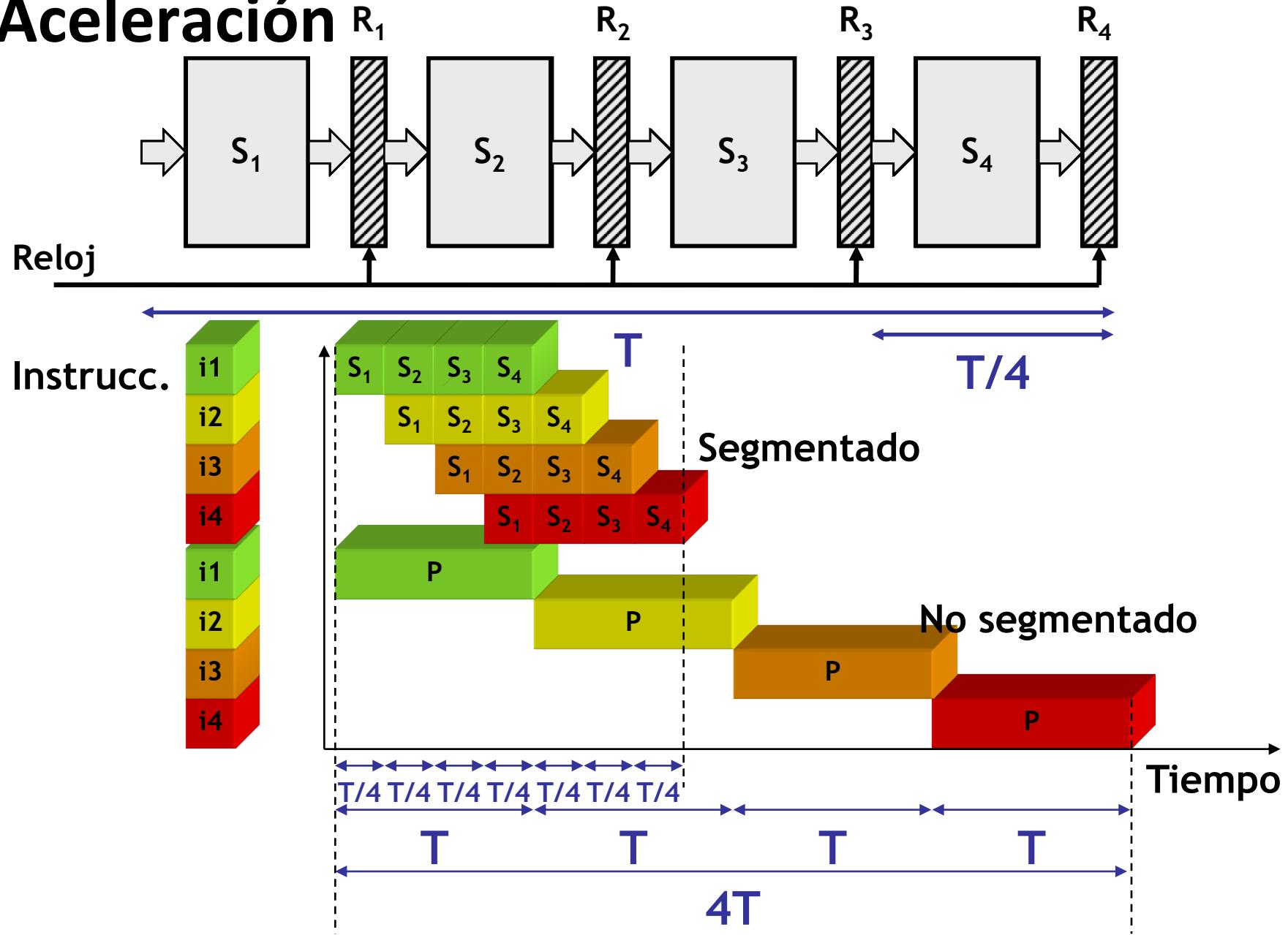
Ejemplo de segmentación

- **Cada etapa del cauce debe completarse en un ciclo de reloj**
- **Fases de captación y de memoria**
 - Si acceden a memoria principal, el acceso es varias veces más lento
 - La caché permite acceso en un único ciclo de reloj
- **El periodo de reloj se escoge de acuerdo a la etapa más larga del cauce**

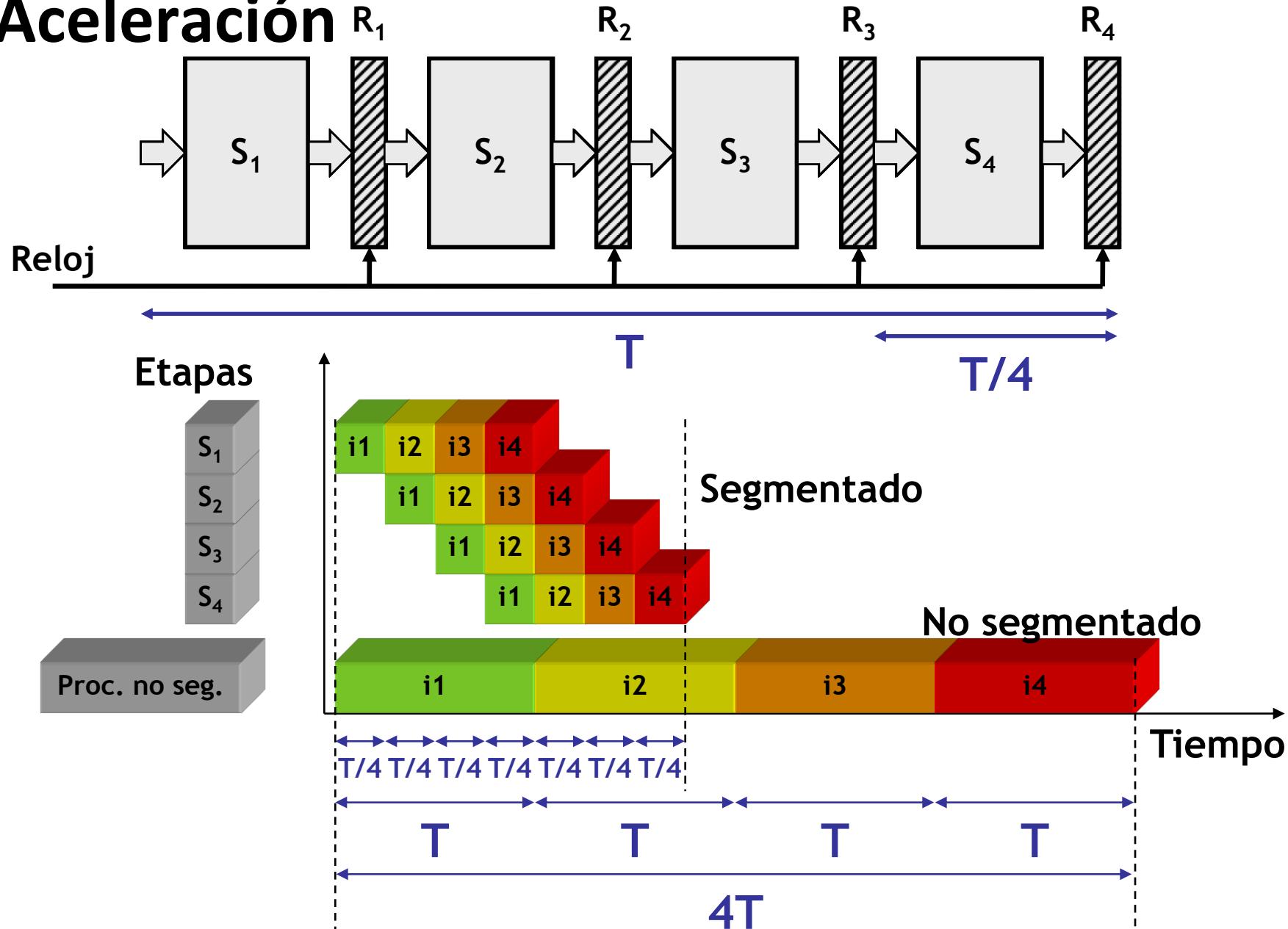
Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

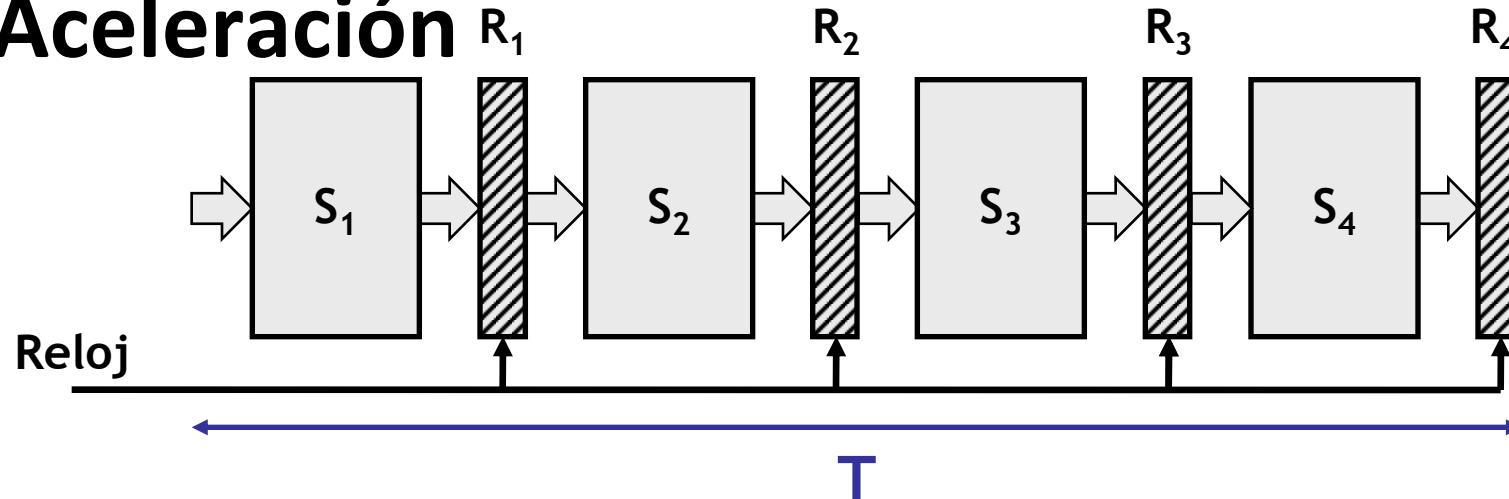
Aceleración



Aceleración



Aceleración



Aceleración en el ejemplo

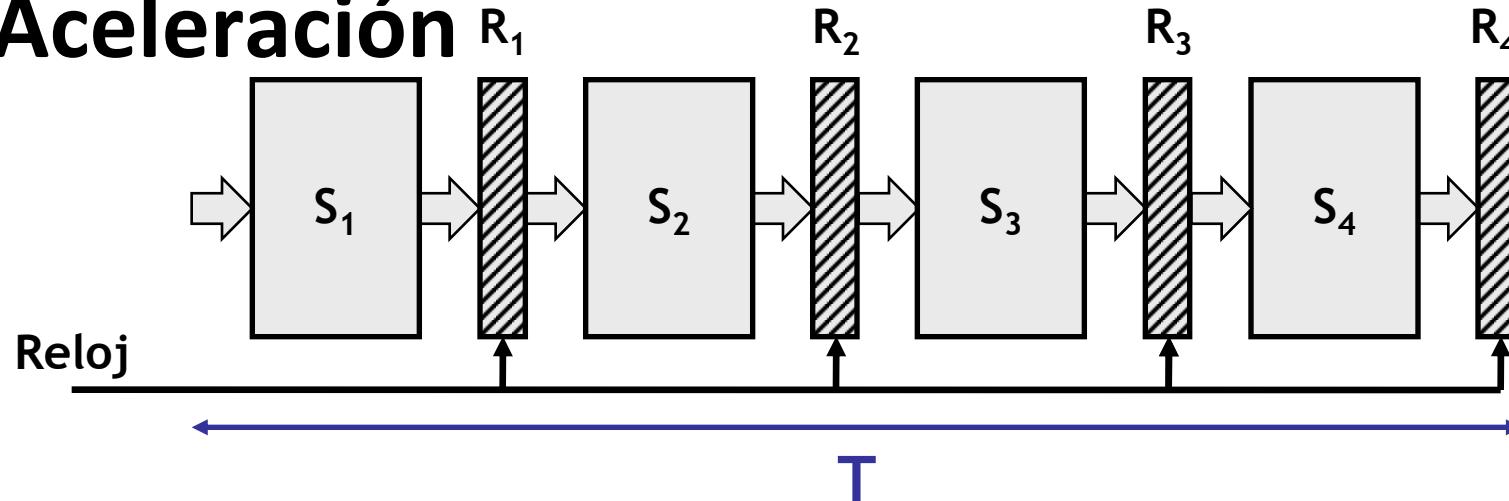
$$\text{Aceleración} = \frac{\text{Tiempo máquina no-segmentada}}{\text{Tiempo máquina segmentada}} = \frac{4T}{7T/4} = \frac{4}{7/4} = \frac{16}{7}$$

Aceleración ideal

$$\text{Aceleración ideal} = \frac{kT}{(n-1+k)T/n} = \frac{nkT}{(n+k-1)T} = n \quad k \rightarrow \infty$$

La aceleración ideal coincide con el número de etapas de segmentación

Aceleración



Causas que disminuyen la aceleración

- ✗ Coste de la segmentación
 - ✗ Duración del ciclo de reloj
impuesto por etapa más lenta
 - ✗ Riesgos (*hazards*) \Rightarrow Bloqueo del avance de instrucciones
- $\} \Rightarrow T_c > T/n$

Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

Riesgos

Riesgo

Situación que impide la ejecución de la siguiente instrucción del flujo del programa durante el ciclo de reloj designado



Obliga a modificar la forma en la que avanzan las instrucciones hacia las etapas siguientes



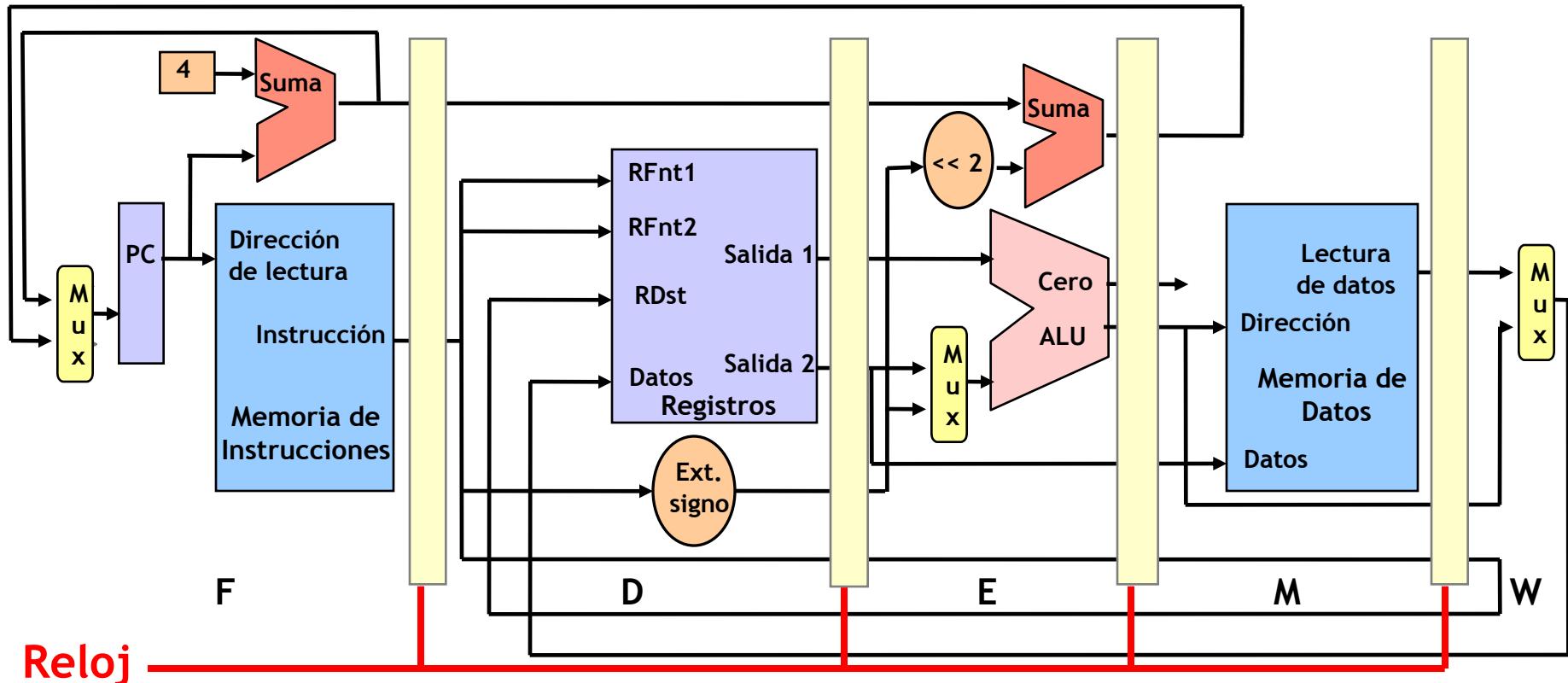
Reducción de las prestaciones logradas por la segmentación

Riesgos

■ Supongamos las siguientes etapas:

- **F**: búsqueda (**fetch**) de instrucción.
- **D**: **d**ecodificación de instrucción / lectura de registros.
- **E**: **e**jecución / cálculo de direcciones
- **M**: acceso a **m**emoria.
- **W**: escritura (**w**rite) de resultados.

Riesgos



Riesgos estructurales

- Conflicto por el **empleo de recursos**, dos instrucciones necesitan un mismo recurso.
- Ej. 1: lectura de dato + captación suponiendo **una única memoria** para datos e instrucciones.

lw	r4,20(r1)	F D E M W
and	r7,r2,r5	F D E M W
or	r8,r6,r2	F D E M W
add	r9,r2,r2	F D E M W
lw	r4,20(r1)	F D E M W
and	r7,r2,r5	F D E M W
or	r8,r6,r2	F D E M W
add	r9,r2,r2	- F D E M W

Riesgos estructurales

- Ej. 2: ejecución de una operación con más de un ciclo en E.

add	rx,rx,rx	F D E M W
mul	rd,rs,rs	F D E E E M W
add	rx,rx,rx	F D E M W
add	rx,rx,rx	F D E M W
add	rx,rx,rx	F D E M W

add	rx,rx,rx	F D E M W
mul	rd,rs,rs	F D E E E M W
add	rx,rx,rx	F D - - E M W
add	rx,rx,rx	F - - D E M W
add	rx,rx,rx	F D E M W

Riesgos estructurales

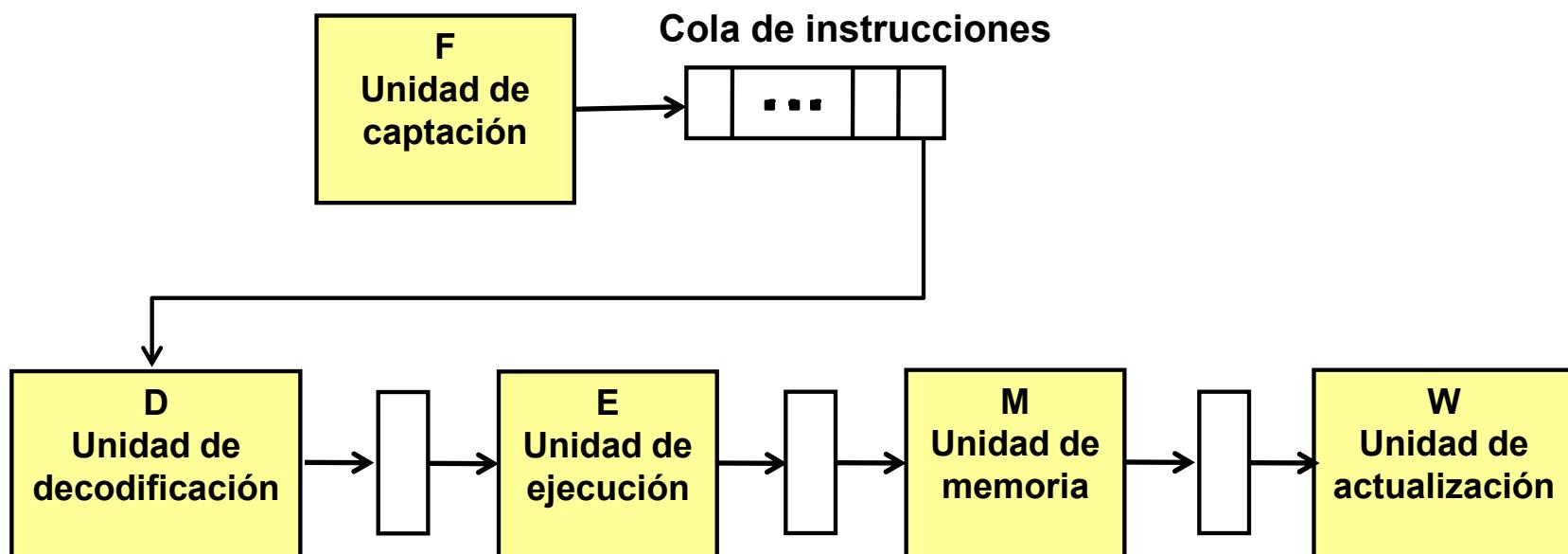
- Ej. 3: fallo de caché al captar una instrucción.

add	rx,rx,rx	F D E M W
sub	rx,rx,rx	F F F F D E M W
add	rx,rx,rx	F D E M W
add	rx,rx,rx	F D E M W
add	rx,rx,rx	F D E M W

add	rx,rx,rx	F D E M W
sub	rx,rx,rx	F F F F D E M W
add	rx,rx,rx	- - - F D E M W
add	rx,rx,rx	F D E M W
add	rx,rx,rx	F D E M W

Riesgos estructurales

- Para reducir el efecto de los fallos de caché se suelen captar instrucciones antes de que sean necesarias (precaptación) y se almacenan en una cola de instrucciones.



Riesgos (por dependencias) de datos

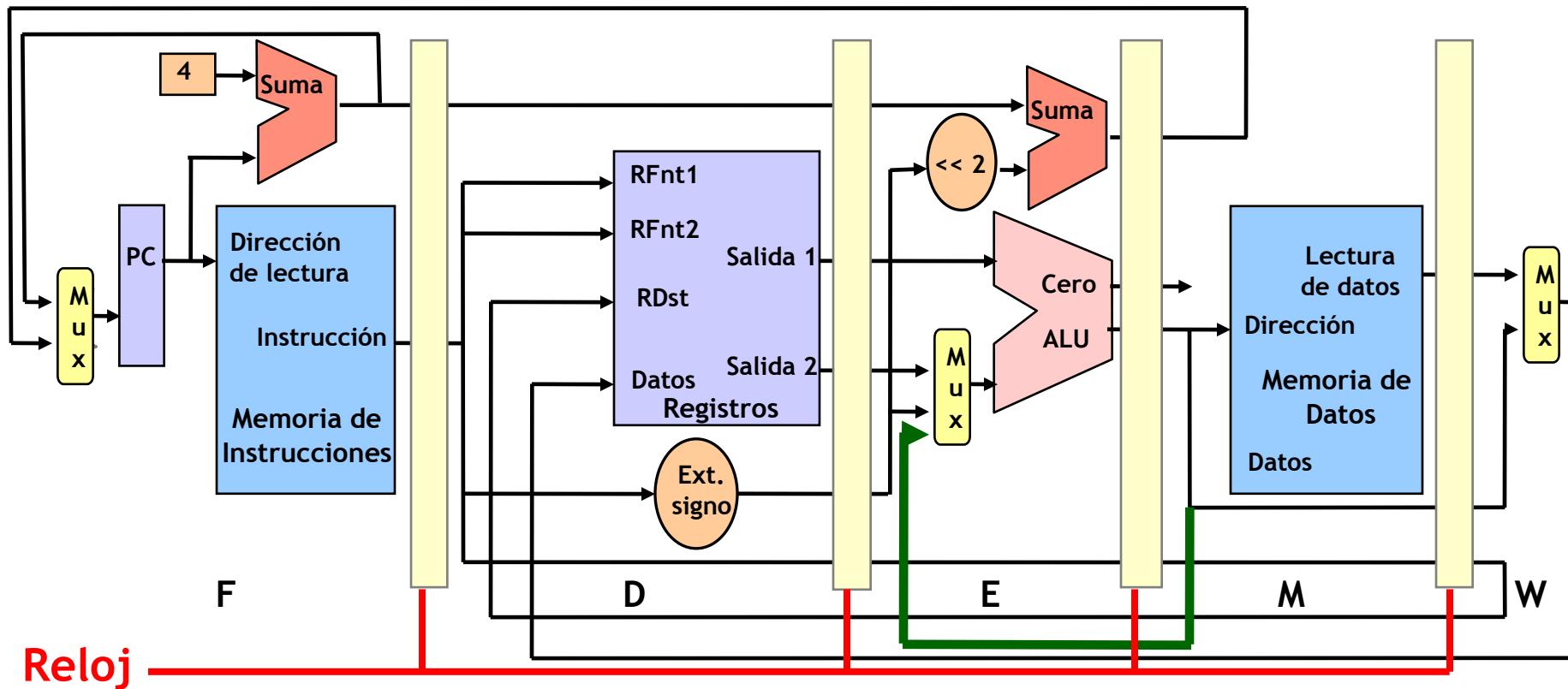
- Acceso a datos cuyo valor actualizado depende de la ejecución de instrucciones precedentes.

sub	r2,r1,r3	F D E M W
and	r7,r2,r5	F D E M W
or	r8,r6,r2	F D E M W
add	r9,r2,r2	F D W M W

sub	r2,r1,r3	F D E M W
and	r7,r2,r5	F D - - D E M W
or	r8,r6,r2	F - - - D E M W
add	r9,r2,r2	F D W M W

Riesgos (por dependencias) de datos

- El nuevo r2 está a la salida de la ALU al terminar E.
 - Si r2 se envía de nuevo a la ALU se elimina el retardo (*register forwarding*).



Riesgos (por dependencias) de datos

- Las dependencias de datos las descubre el hardware al decodificar las instrucciones.
- Alternativamente puede resolverlas el compilador:

sub	r2,r1,r3	F D E M W
nop		F D E M W
nop		F D E M W
nop		F D E M W
and	r7,r2,r5	F D E M W

- Ventajas:
 - Hardware más simple
 - Reorganizar instrucciones para hacer trabajo útil en lugar de NOP
- Inconveniente:
 - Aumenta el tamaño del código

Riesgos de control

- Consecuencia de la ejecución de instrucciones de salto.
- Salto **incondicional**:

br	L1	F D E M W
and	r2,r1,r4	F D E - -
sub	r5,r6,r7	F D - - -
or	r8,r1,r6	F - - - -
L1:add	r6,r1,r4	F D E M W

- Se pierden 3 ciclos (**huecos de retardo de salto***), ya que tras captar la instrucción br, hasta después del 4º ciclo (es decir, pasados otros 3 ciclos) no se conoce la dirección de salto.

Riesgos de control

- Importante averiguar la dirección de salto **lo antes posible**, por ej. en la etapa de decodificación:

br	L1	F D E M W
and	r2,r1,r4	F - - - -
sub	r5,r6,r7	
or	r8,r1,r6	
L1:add	r6,r1,r4	F D E M W

- Se pierde 1 ciclo, ya que tras captar la instrucción br, después del 2º ciclo ya se conoce la dirección de salto.

Riesgos de control

■ Salto condicional:

beq	r2,r3,L1	F D E M W
and	r2,r1,r4	F D E M W
sub	r5,r6,r7	F D E M W
or	r8,r1,r6	F D E M W
L1:add	r6,r1,r4	F D E M W

- Si se produce el salto se pierden 3 ciclos.
- Si no se produce el salto, no se pierden ciclos.

Riesgos de control

■ Degradación de prestaciones debida a los saltos.

- Supongamos algún mecanismo hardware que permita descartar la ejecución de las instrucciones siguientes si se produce el salto.
 - b : nº de ciclos desperdiciados cuando se produce el salto.
 - p_b : probabilidad de que se ejecute una instrucción de salto
 - (entre 0,15 y 0,30 normalmente)
 - p_t : probabilidad de que realmente se produzca el salto cuando se ejecuta una instrucción de salto
 - $p_e = p_b p_t$: probabilidad efectiva de que se produzca un salto
 - CPI: nº de ciclos de reloj por instrucción (suponer CPI = 1 sin saltos)

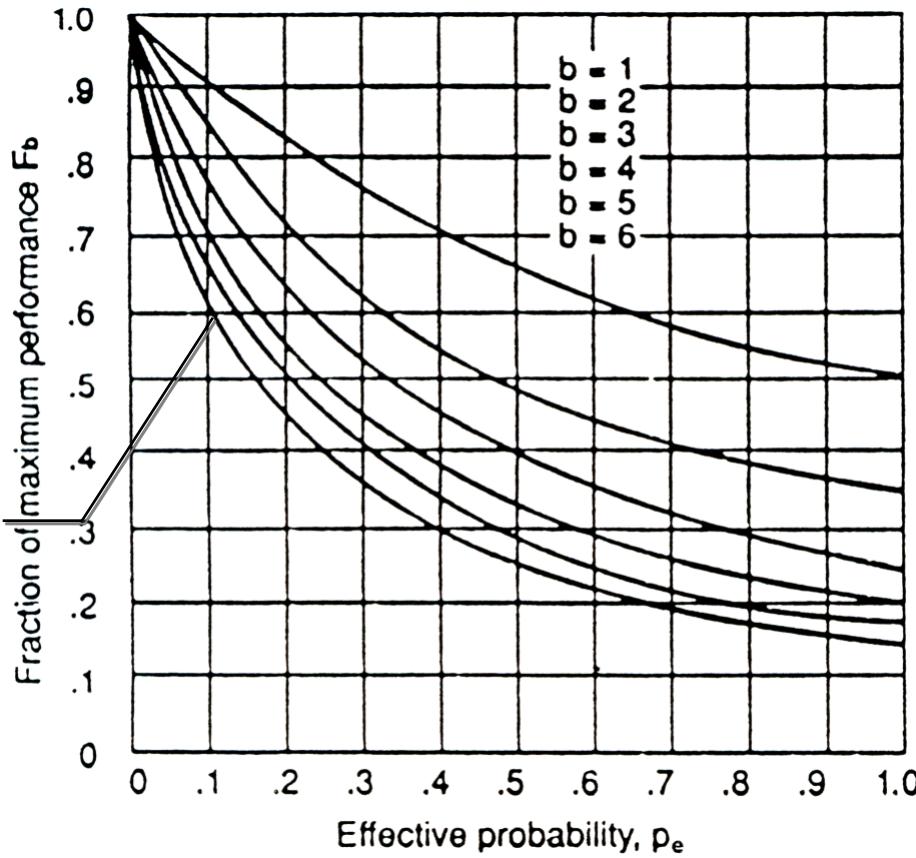
$$\text{CPI} = (1 - p_b) (1) + p_b [p_t (1 + b) + (1 - p_t) (1)] = 1 + p_b p_t b = 1 + p_e b$$

Riesgos de control

- F_b : fracción de máximas prestaciones (relación entre el nº de ciclos CPI si no hubiera saltos y el nº de ciclos CPI con saltos)

$$F_b = \frac{1}{1 + p_e b}$$

La degradación crece rápidamente al crecer p_e
(más rápidamente a medida que b es mayor)



Salto retardado (*delayed branch*)

- En lugar de desperdiciar las etapas posteriores a la de salto, una o más instrucciones parcialmente completadas se completarán antes de que el salto tenga efecto.
- El compilador busca instrucciones **anteriores** lógicamente al salto que pueda colocar tras el salto.
- Si el salto es condicional, las instrucciones colocadas detrás no deben afectar a la condición de salto.

Antes:

```
mov r1,#3  
jmp etiq  
nop
```

Después:

```
jmp etiq  
mov r1,#3
```

Salto retardado (*delayed branch*)

- Otra posibilidad es colocar la(s) instrucción(es) destino de un salto tras el salto.

Antes:

call subr

nop

...

subr:

mov r3, #100

Después:

call subr+4

mov r3, #100

...

subr:

mov r3, #100

- Esto no funcionaría para saltos condicionales
- No podemos “subir” una instrucción que sólo tiene que ejecutarse algunas veces (cuando el salto se produce) a una posición donde siempre se ejecuta.

Annulling branch

- Un salto de este tipo ejecuta la(s) instrucción(es) siguiente(s) sólo si el salto se produce, pero la(s) ignora si el salto no se produce.
- Con un salto de este tipo, el destino de un salto condicional sí puede colocarse tras el salto.

Antes:

```
bz else  
nop  
; código then  
...  
else:
```

```
    mov r3,#100
```

Después:

```
bz,a else+4  
mov r3,#100  
; código then  
...  
else:
```

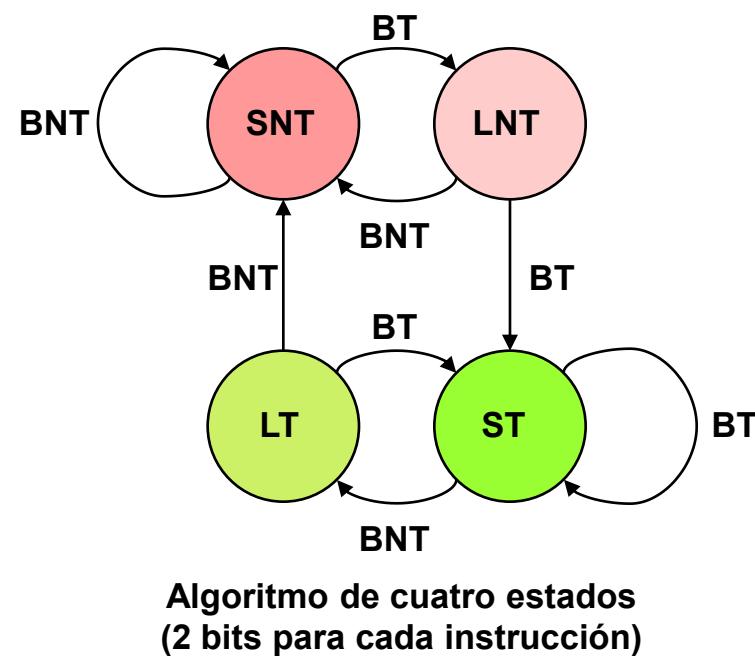
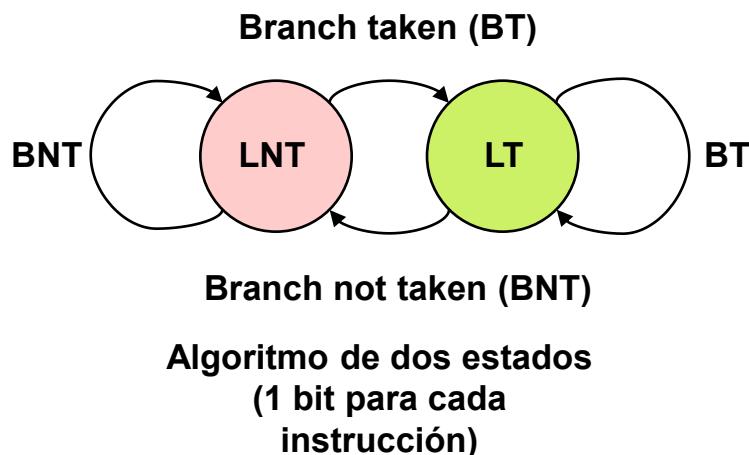
```
    mov r3,#100
```

Predicción de saltos

- Intentar predecir si una instrucción de salto concreta dará lugar a un salto (**branch taken**) o no (**branch not taken**).
 - Si los resultados de las instrucciones de salto condicional fueran aleatorios, comenzar a ejecutar las instrucciones siguientes al salto desperdiciaría ciclos en la mitad de las ocasiones.
 - Se pueden minimizar las pérdidas de ciclos inútiles si para cada instrucción de salto se puede predecir con un acierto > 50% si el salto se producirá o no.
 - Se pueden hacer predicciones distintas si el salto es hacia direcciones menores o mayores.
- Tipos de predicción:
 - **Estática**: se toma la misma decisión para cada tipo de instrucción
 - **Dinámica**: cambia según la historia de ejecución de un programa

Predicción dinámica de saltos

- ST: Muy probable saltar
- LT: Probable saltar
- LNT: Probable no saltar
- SNT: Muy probable no saltar



Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- **Influencia en el repertorio de instrucciones**
- Funcionamiento superescalar

Modos de direccionamiento

- Es deseable que un operando no necesite más de **un acceso a memoria**
 - Constante
 - Registro
 - Indirecto a través de registro
 - Indexado ($EA = \text{reg.} + \text{desp.}$, o $EA = \text{reg.} + \text{reg.}$)
- Es deseable que sólo puedan acceder a memoria las instrucciones de **carga y almacenamiento**

Modos de direccionamiento

■ Modo de direc. complejo, 2 accesos a memoria

lw	r4 , (20(r1))	F D E M M W
and	r7 , r2 , r5	F D E - M W

■ Modo de direc. más simple, 1 acceso a memoria

lw	r3 , 20(r1)	F D E M W
lw	r4 , (r3)	F D E M W
and	r7 , r2 , r5	F D E M W

- Idéntica duración, pero hardware más sencillo

Códigos de condición

- Las dependencias que introducen los bits de condición dificultan al compilador reordenar el código (deseable para evitar riesgos).
- En el ejemplo siguiente, la decisión de saltar se produce tras la etapa E de la instrucción cmp
- Es deseable elegir en cada instr. si afecta o no a los cód. de condición, para reordenar el código:

Antes:

```
add r1,r2  
cmp r3,r4  
jz etiq
```

Después:

```
cmp r3,r4  
add r1,r2  
jz etiq
```

Al reordenar el código, se puede tomar la decisión de salto un ciclo antes, y desperdiciar un ciclo menos tras el salto

Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

Funcionamiento superescalar

■ Procesamiento segmentado

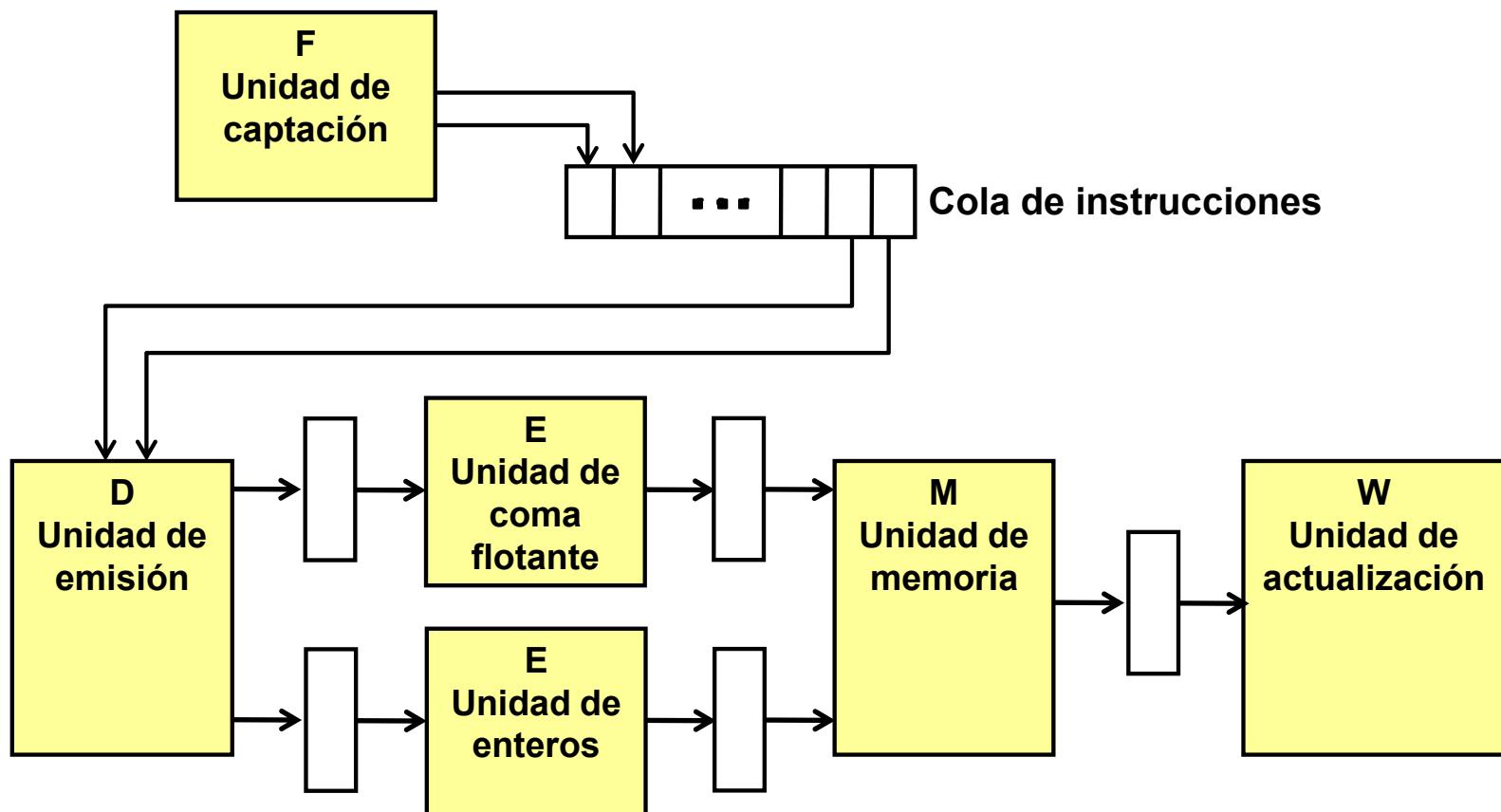
- Una instrucción tras otra
- Rendimiento ideal: una instrucción por ciclo

■ Procesamiento superescalar

- Varias instrucciones en **paralelo**
- Necesidad de **varias unidades funcionales**
- **Emisión múltiple**: se puede comenzar a ejecutar más de una instrucción por ciclo de reloj
- Rendimiento: **más de una instrucción por ciclo**
- Es fundamental poder captar instrucciones rápidamente: conexión ancha con caché + cola de instrucciones

Funcionamiento superescalar

- Ej.: Procesador con dos unidades de ejecución:



Funcionamiento superescalar

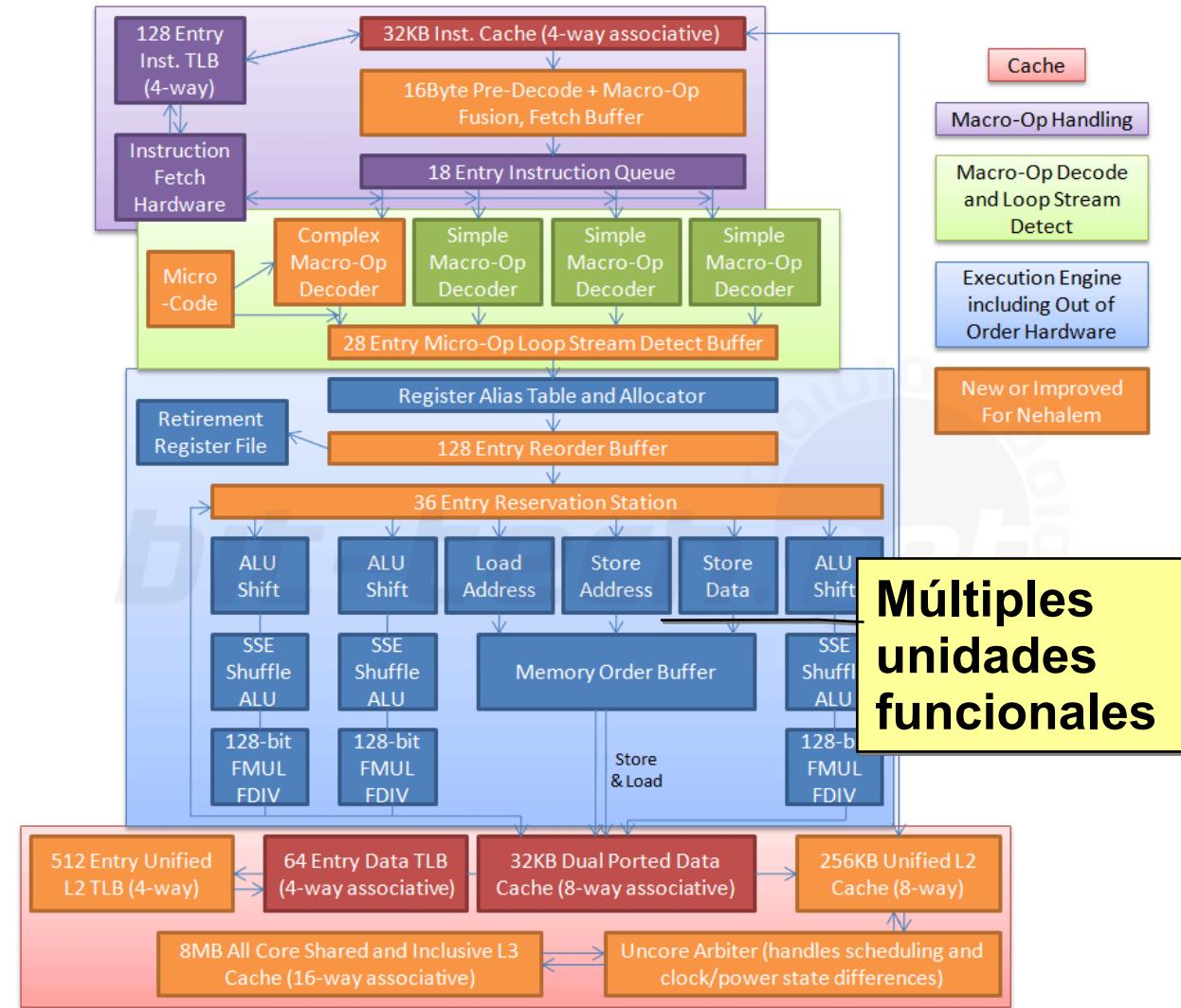
- El efecto negativo de los riesgos es más pronunciado.
- El compilador puede reordenar instrucciones para evitar riesgos.

fadd	rx, rx, rx	F D E ₁ E ₂ E ₃ M W
add	rx, rx, rx	F D E M W
fsub	rx, rx, rx	F D E ₁ E ₂ E ₃ M W
sub	rx, rx, rx	F D E M W

- Las instrucciones pueden emitirse en orden y finalizar de forma desordenada (ej. add finaliza antes que fadd)
 - Múltiples problemas y soluciones, que se verán en Arquitectura de Computadores

Core i7 (Nehalem)

- 4 cores/chip
- Segmentación:
 - 16 etapas
- Superescalar:
 - 4 instrucciones en paralelo
- Caches:
 - L1: 32KB I + 32KB D
 - L2: 256KB
 - L3: 8MB, compartida por los 4 cores



TEMA 6.1

Memoria I: Jerarquía de memoria

- **La abstracción de memoria (concepto de Lectura y Escritura)**
- **RAM: bloque constructivo de memoria principal**
- **Configuración y diseño de memorias utilizando varios chips**
- **Localidad de las referencias**
- **Jerarquía de memoria**
- **Tecnologías de almacenamiento, y tendencias**

Lectura y Escritura (de/a) Memoria

■ Escritura

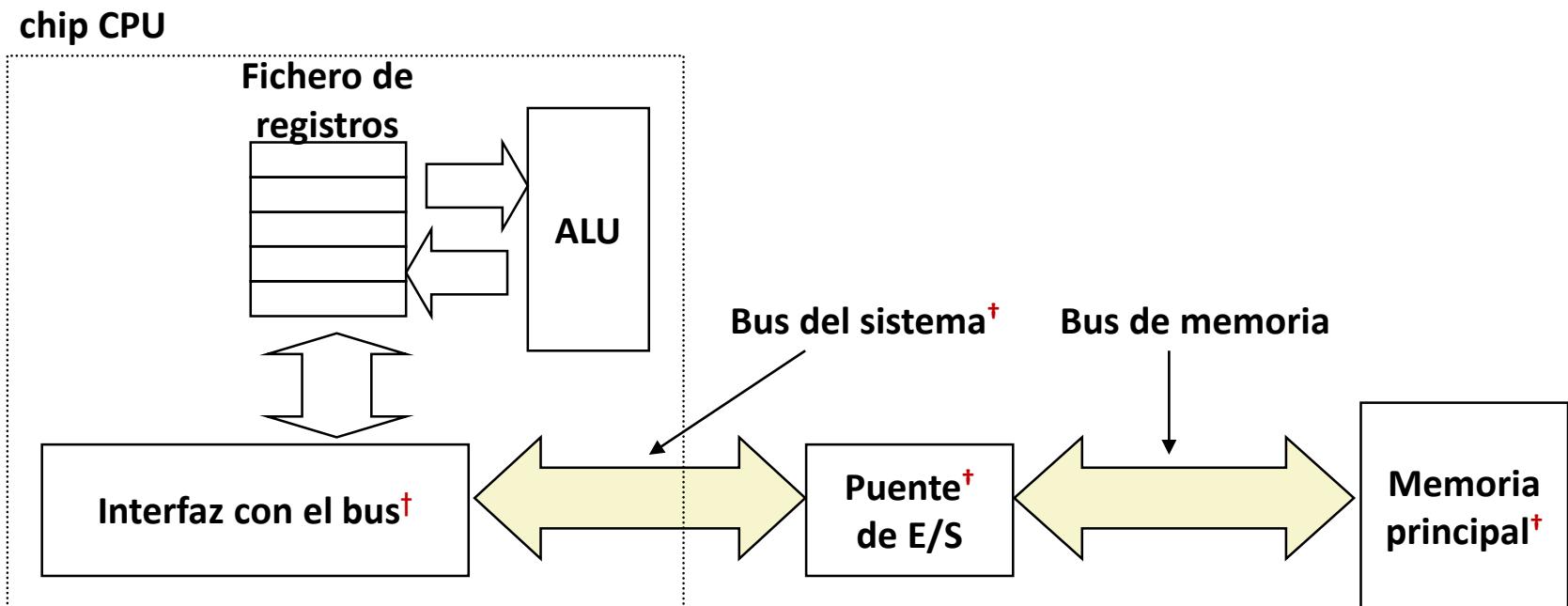
- Transferir datos de CPU a memoria
`movq %rax, 8(%rsp)`
- Operación “Store”

■ Lectura

- Transferir datos de memoria a CPU
`movq 8(%rsp), %rax`
- Operación “Load”

Estructura de buses CPU – Memoria (tradicional[#])

- Un **bus** es un conjunto de cables en paralelo que transportan direcciones, datos, y señales de control
- Típicamente a un bus se conectan varios dispositivos

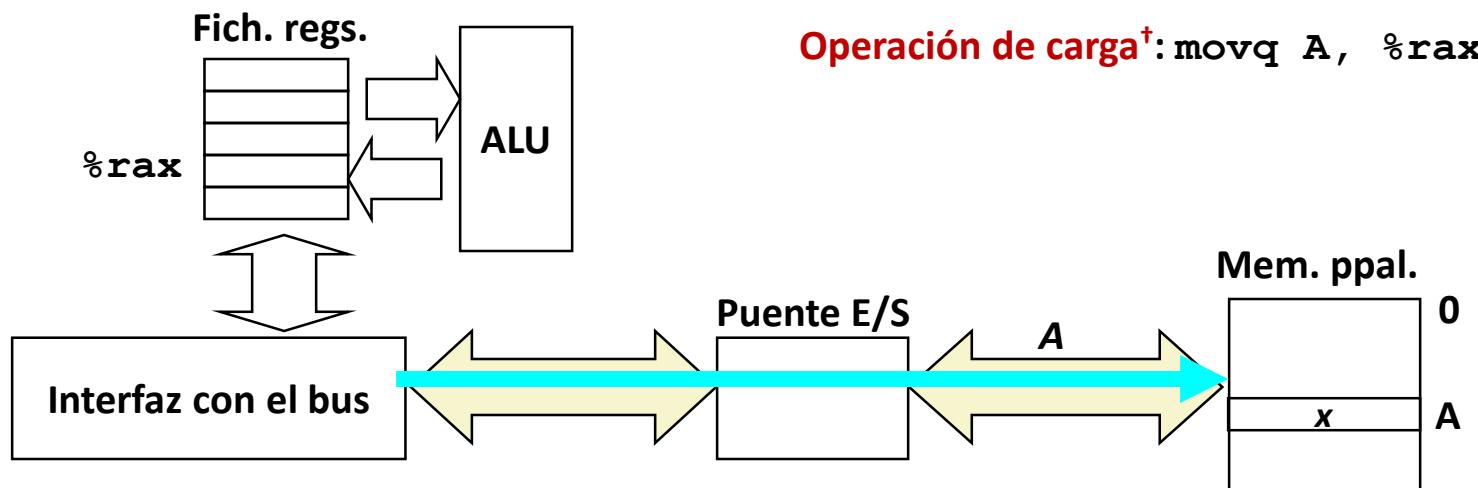


[#] "System bus", "Bus interface", "I/O bridge", "Main memory".

[#] North/SouthBridge (1990), Intel Hub Arch (IHA=MCH/ICH 1999), Platform Controller Arch=(IMC/PCH (2008), IOH/ICH (2008).

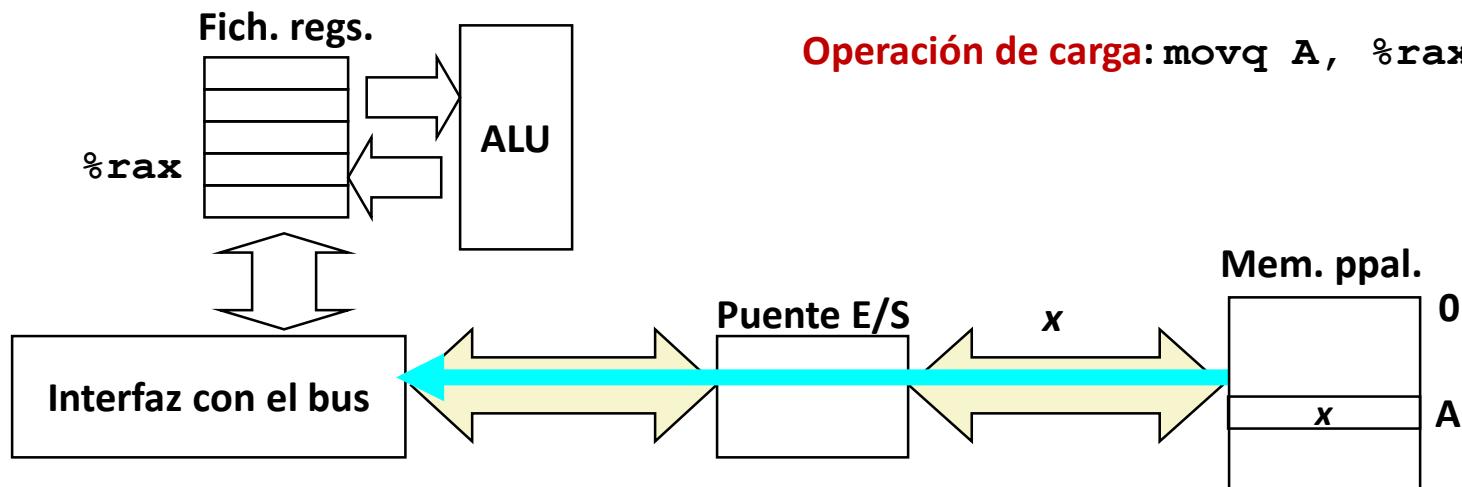
Transacción de Lectura de Memoria (1)

- La CPU pone la dirección A en el bus de memoria



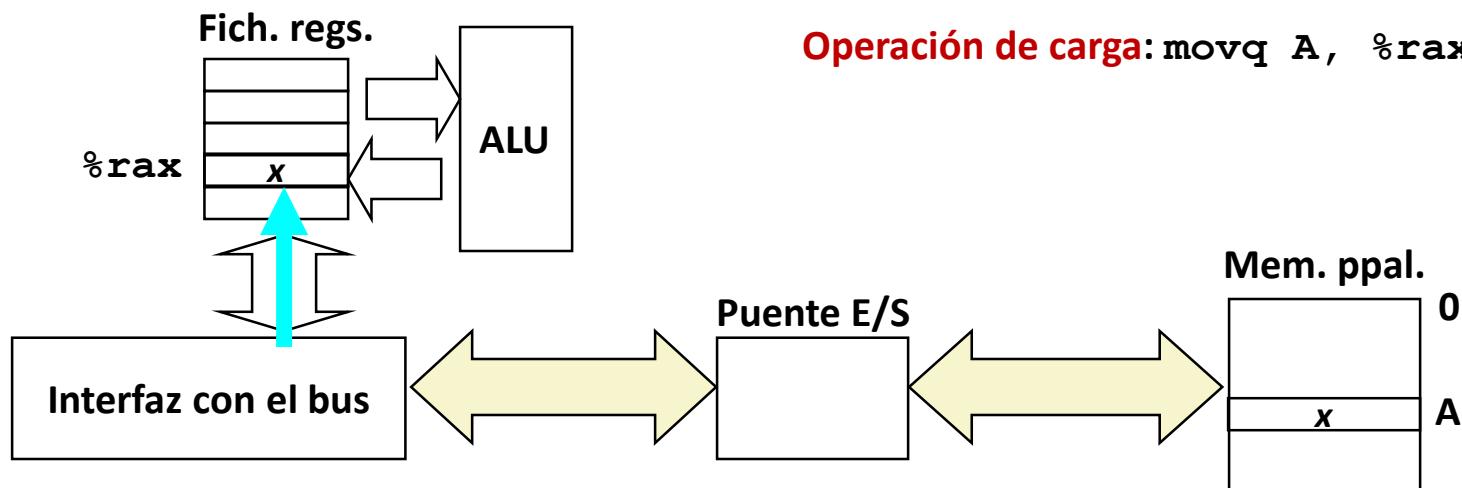
Transacción de Lectura de Memoria (2)

- La memoria principal recibe dirección A del bus de memoria, recupera la palabra x, y la pone en el bus



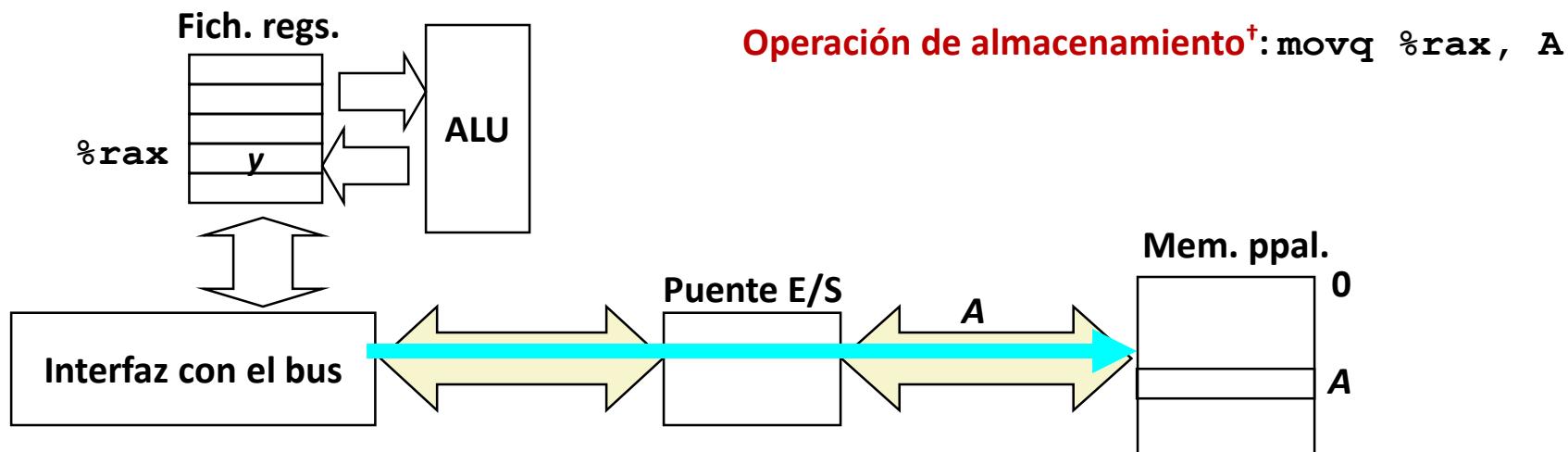
Transacción de Lectura de Memoria (3)

- La CPU lee palabra x del bus y la copia al registro $\%rax$



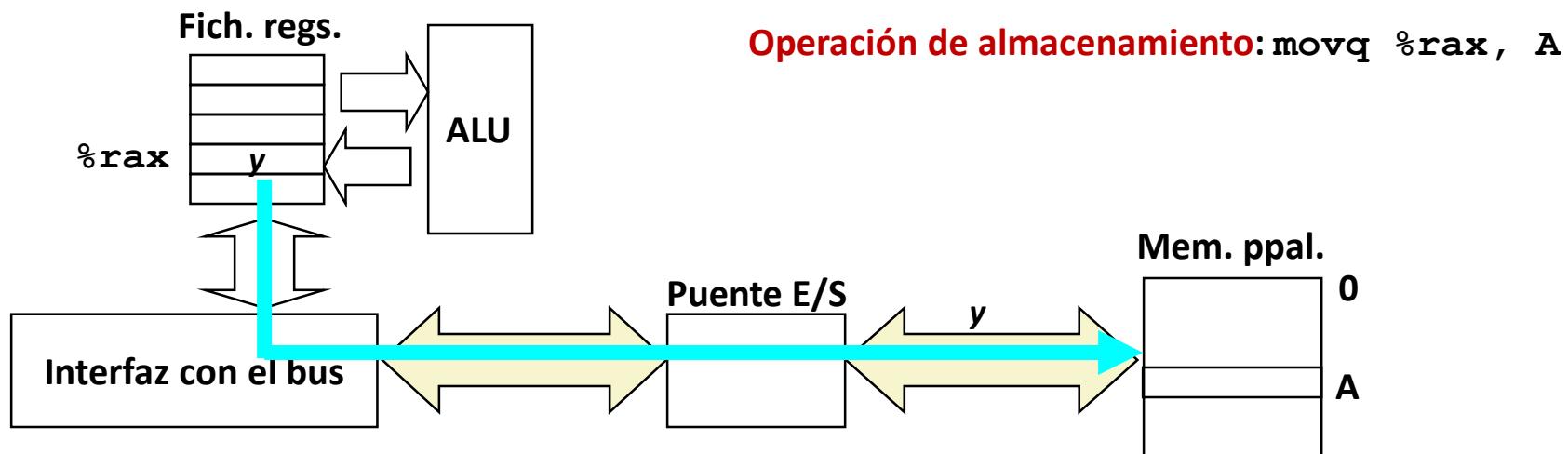
Transacción de Escritura a Memoria (1)

- La CPU pone dirección A en el bus. La mem. ppal. la recibe y espera a que llegue la palabra de datos correspondiente



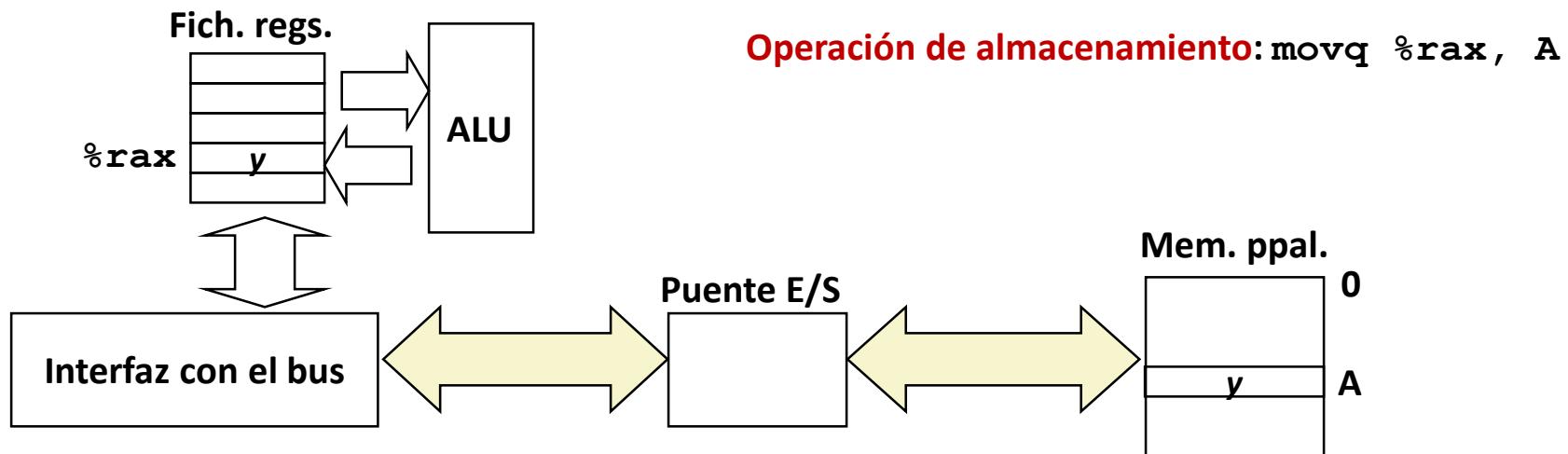
Transacción de Escritura a Memoria (2)

- La CPU pone la palabra de datos y en el bus



Transacción de Escritura a Memoria (3)

- La memoria principal recibe la palabra de datos y del bus y la almacena en la posición con dirección A



Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- **RAM: bloque constructivo de memoria principal**
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Algunas definiciones

- **Tiempo de acceso:**
 - tiempo que se requiere para leer (o escribir) un dato (palabra) en la memoria
 - medido en ciclos o en $(n \cdot \mu \cdot m)$ s
- **Ancho de banda:** (de la memoria de un computador)
 - Número de palabras a las que puede acceder el procesador (o que se pueden transferir entre el procesador y la memoria) por unidad de tiempo
 - medido en (K-M-G) B/s

Algunas definiciones

■ Métodos de acceso:

- **Aleatorio (RAM)**: tiempo de acceso independiente de posición a acceder
 - SRAM, ROM
- **Secuencial (SAM)**: t. acceso depende de posición de los datos a acceder
 - Cinta magnética
- **Directo** (semialeatorio, **DASD** – direct access storage device)
 - tiempo acceso tiene una componente aleatoria y otra secuencial
 - Discos giratorios (época en que lat. rotacional $>>$ t. búsqueda)
- actualmente muchos dispositivos tienen 2 o más componentes acceso
 - DRAM: CL-T_{RCD}-T_{RP}-T_{RAS}
 - (CAS latency, Row-Col delay, Row precharge, Row active)
 - No es lo mismo acceder a nueva fila, a otra columna de la misma fila, a ráfaga...

Memoria de Acceso Aleatorio (RAM)

■ Características principales

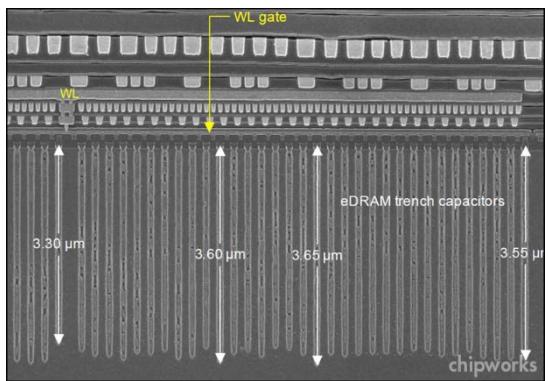
- La **RAM** tradicionalmente se empaqueta como un chip
 - o incluida[†] como parte de un chip procesador
- Unidad básica almacenamiento es normalmente una celda (1 bit/celda)
- Múltiples chips de RAM forman una memoria

■ La RAM tiene dos variedades:

- SRAM (RAM estática)
- DRAM (RAM Dinámica)

Tecnologías RAM

■ DRAM

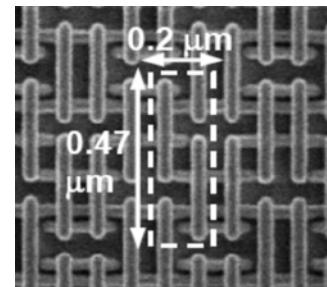


■ (1 Transistor + 1 condensador) / bit

- Condensador orientado verticalmente

■ Debe refrescar estado periódicamente

■ SRAM



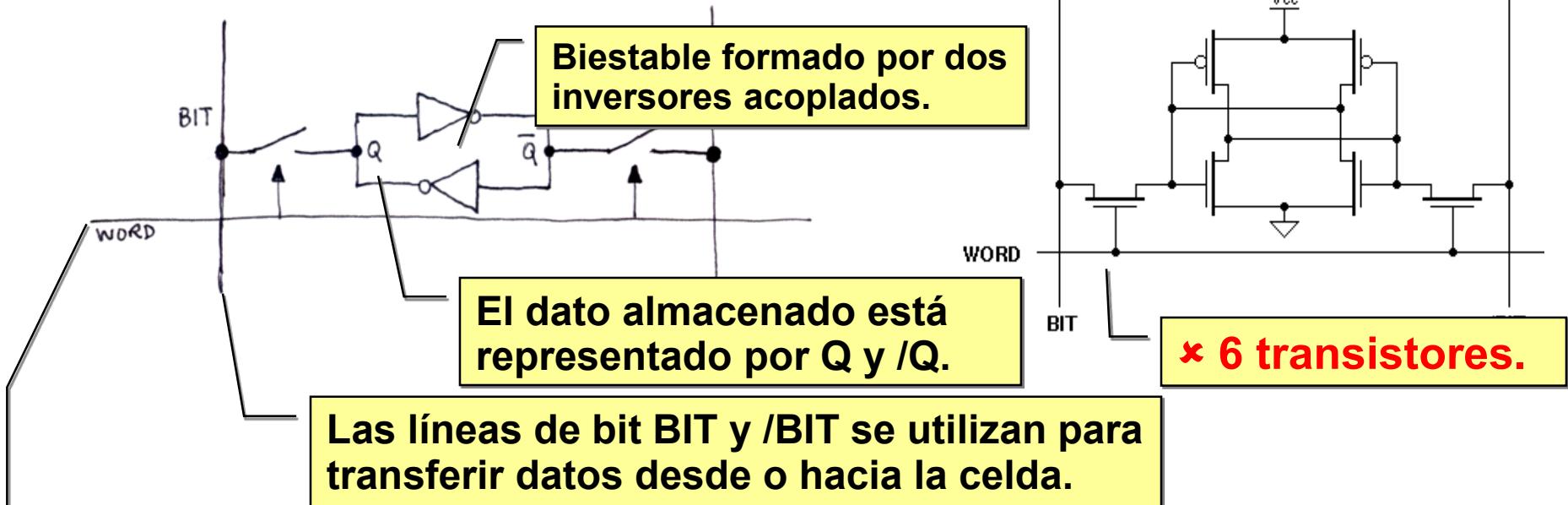
■ 6 transistores / bit

- 2 inversores (x 2 tr) + 2 puertas de paso
- Mantiene el estado indefinidamente

SRAM

■ Celda de memoria SRAM

- Estática ⇒ los datos almacenados se mantienen por un tiempo indefinido si hay alimentación.



- Cuando la línea WORD esté a 1 se selecciona la celda para lectura o escritura.
- Cuando la línea WORD esté a 0 los dos transistores asociados no conducen, desconectando la celda de las líneas de bit.

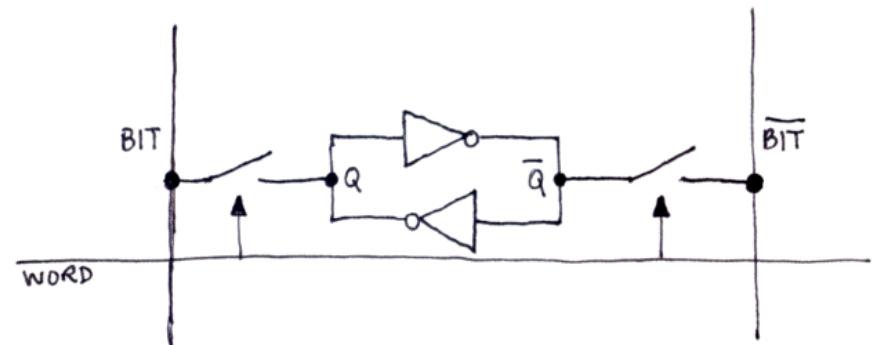
x 6 transistores.

SRAM

■ Celda de memoria SRAM

■ Operación de lectura:

- Se selecciona la celda poniendo WORD a 1 \Rightarrow Q se conecta a BIT y /Q a /BIT.
- Si Q = 1 ($/Q = 0$) \Rightarrow la línea /BIT se pone a 0.
 - BIT = 1 y $/BIT = 0 \Rightarrow$ se lee un 1.
- Si Q = 0 ($/Q = 1$) \Rightarrow la línea BIT se pone a 0.
 - BIT = 0 y $/BIT = 1 \Rightarrow$ se lee un 0.



✓ Lectura no destructiva.

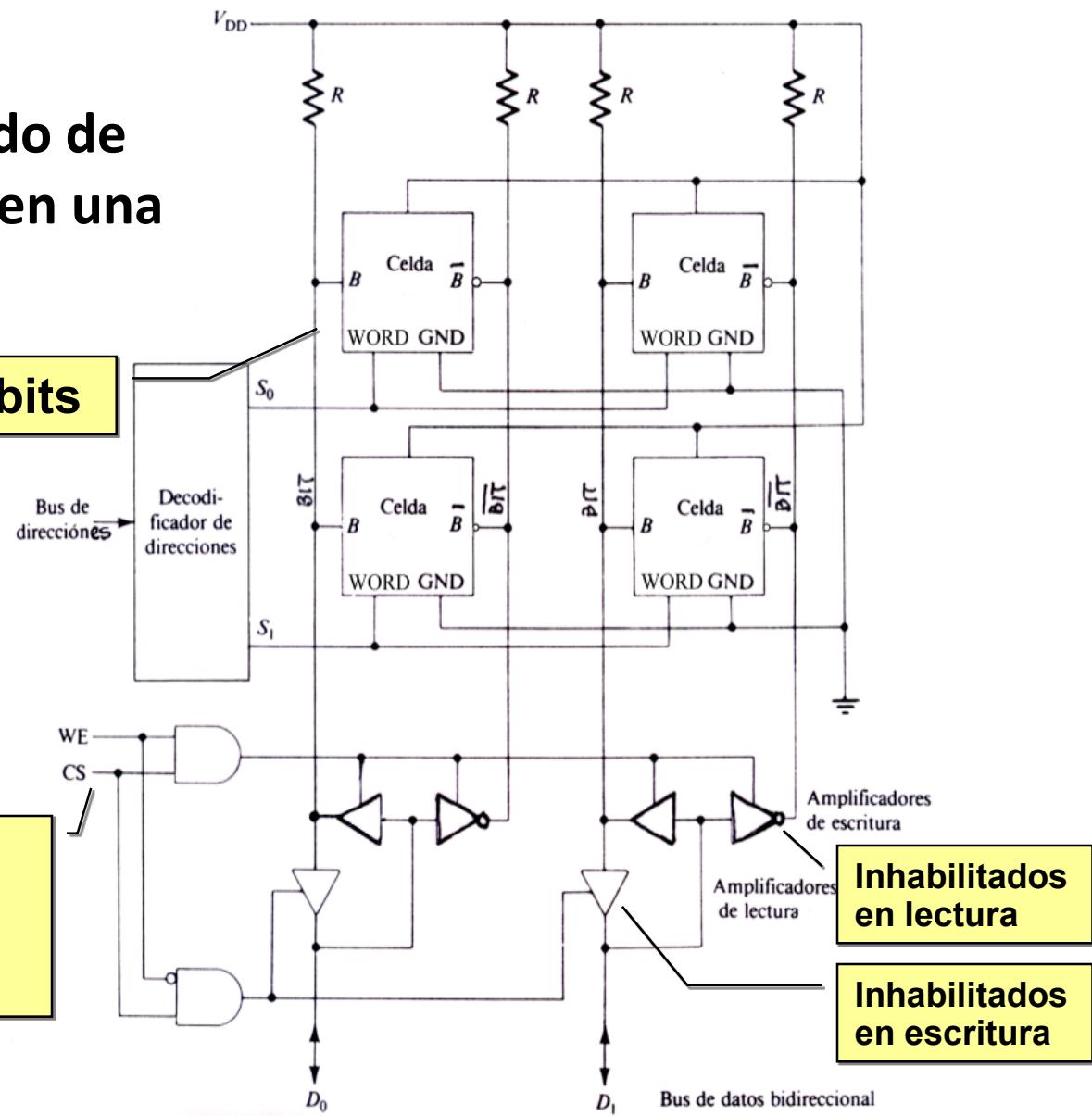
✓ Mayor velocidad que DRAM.

SRAM

- Esquema simplificado de conexión de celdas en una SRAM:

SRAM de 2 palabras de 2 bits

**Si CS = 1 \Rightarrow
WE = 1 \Rightarrow Escritura
WE = 0 \Rightarrow Lectura**



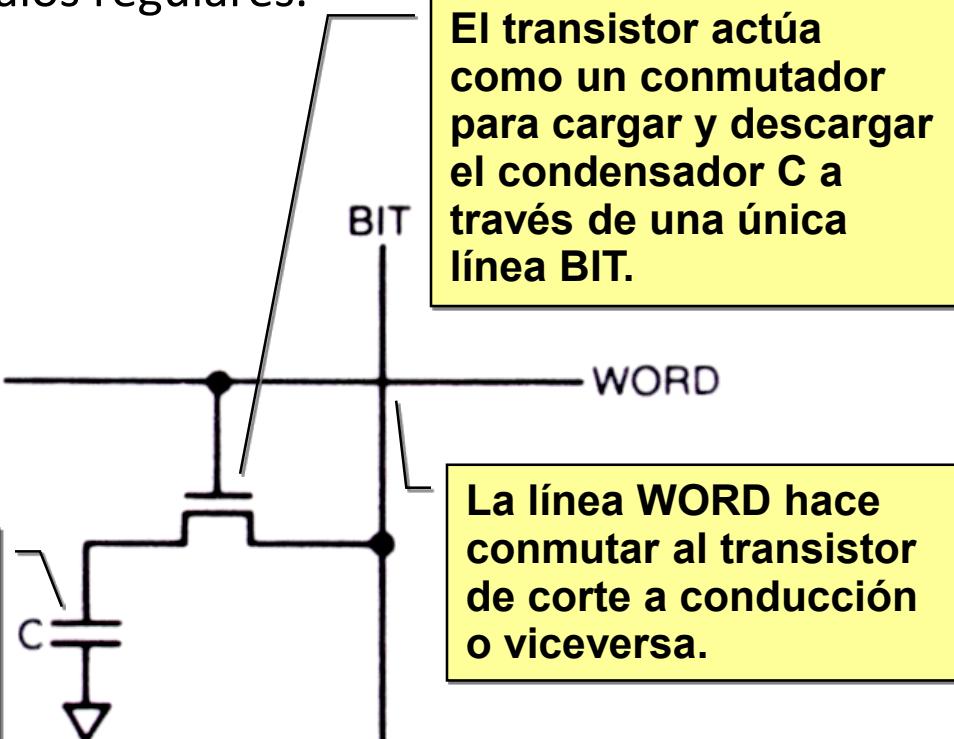
DRAM

■ Celda de memoria DRAM

- **Dinámica** ⇒ los datos almacenados decaen o se desvanecen y deben ser restaurados a intervalos regulares.

✓ **Sencillez de la celda de almacenamiento.**

La información se almacena en forma de carga eléctrica en un condensador C. La presencia (o ausencia) de carga indica que hay almacenado un 1 (o un 0).



El transistor actúa como un conmutador para cargar y descargar el condensador C a través de una única línea BIT.

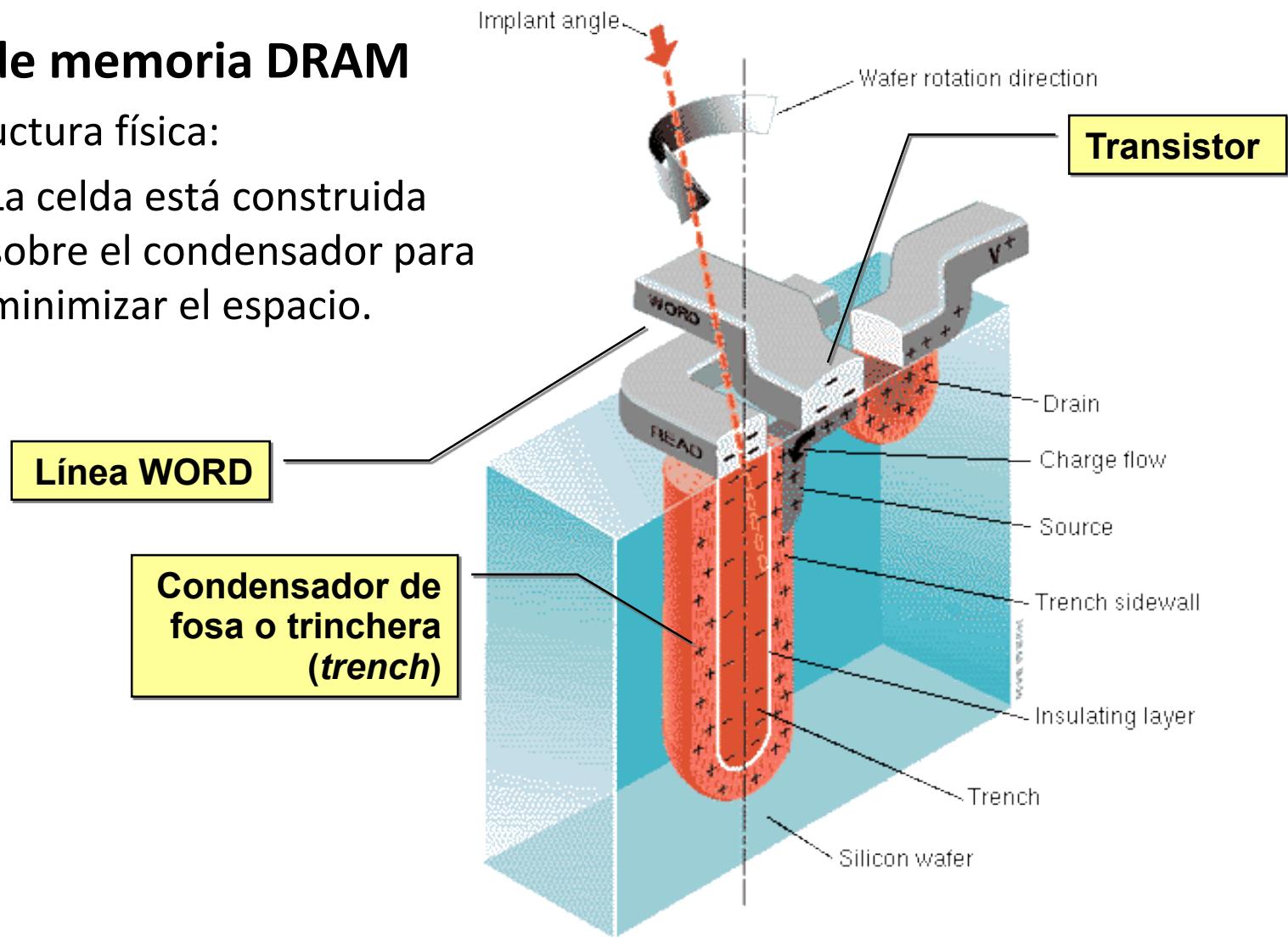
La línea WORD hace conmutar al transistor de corte a conducción o viceversa.

DRAM

■ Celda de memoria DRAM

■ Estructura física:

- La celda está construida sobre el condensador para minimizar el espacio.

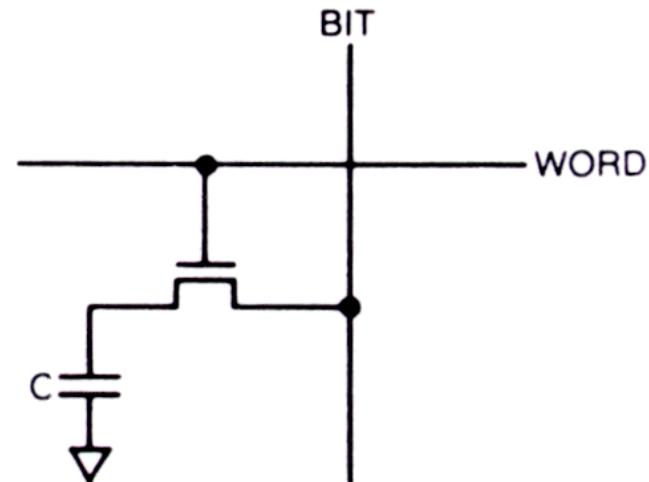


DRAM

■ Celda de memoria DRAM

■ Operación de lectura:

- La circuitería externa convierte a BIT en una línea de salida, seleccionándose la celda con WORD = 1.
- Si C está cargado (= 1) \Rightarrow se descarga a través de la línea BIT \Rightarrow se produce un pulso de corriente que es detectado por un amplificador de salida ("*sense amplifier*") \Rightarrow aparece un 1 en la línea de datos de salida.
- Si C está descargado (= 0) \Rightarrow no se produce pulso de corriente \Rightarrow aparece un 0 en la línea de datos de salida.



x Lectura destructiva.

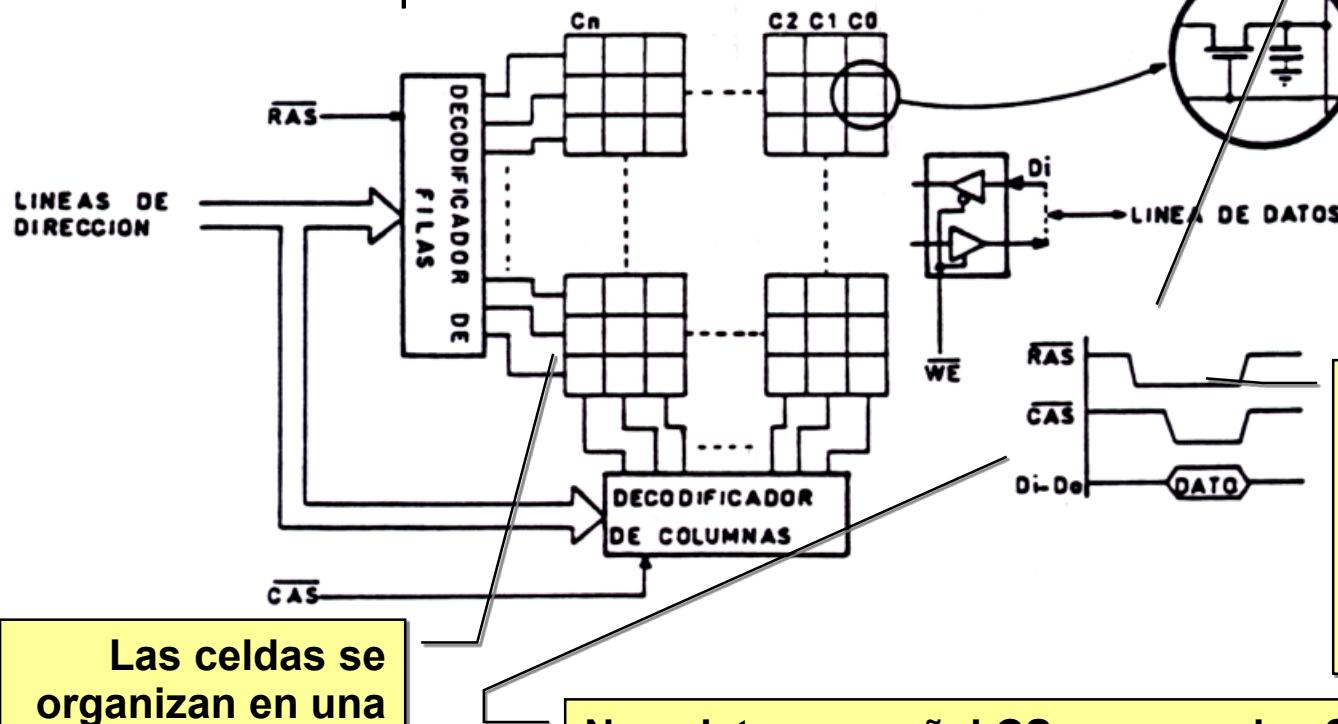
↓
Debe ir seguida de una escritura que restaure el estado original.

↓
Realizada automáticamente por los circuitos de control de la DRAM

DRAM

■ Circuitos de memoria DRAM

- Capacidad elevada de los chips de memoria DRAM ⇒ las direcciones han de proporcionarse multiplexadas en el tiempo.



Las celdas se organizan en una matriz lo más cuadrada posible.

No existe una señal CS como en las SRAM. Para dar por válido un acceso a memoria es necesaria la secuencia completa /RAS - /CAS junto con la señal /WE (Write Enable).

Los bits correspondientes a las filas de la matriz de memoria se proporcionan en primer lugar, validándose por la señal **/RAS (Row Access Strobe)**.

Después se proporcionan los bits que direccionan las columnas, validados por **/CAS (Column Access Strobe)**.

DRAM

- Tiempo de acceso.
- Tiempo de ciclo.

$t_{RAC} = t_a$
(tiempo de acceso):

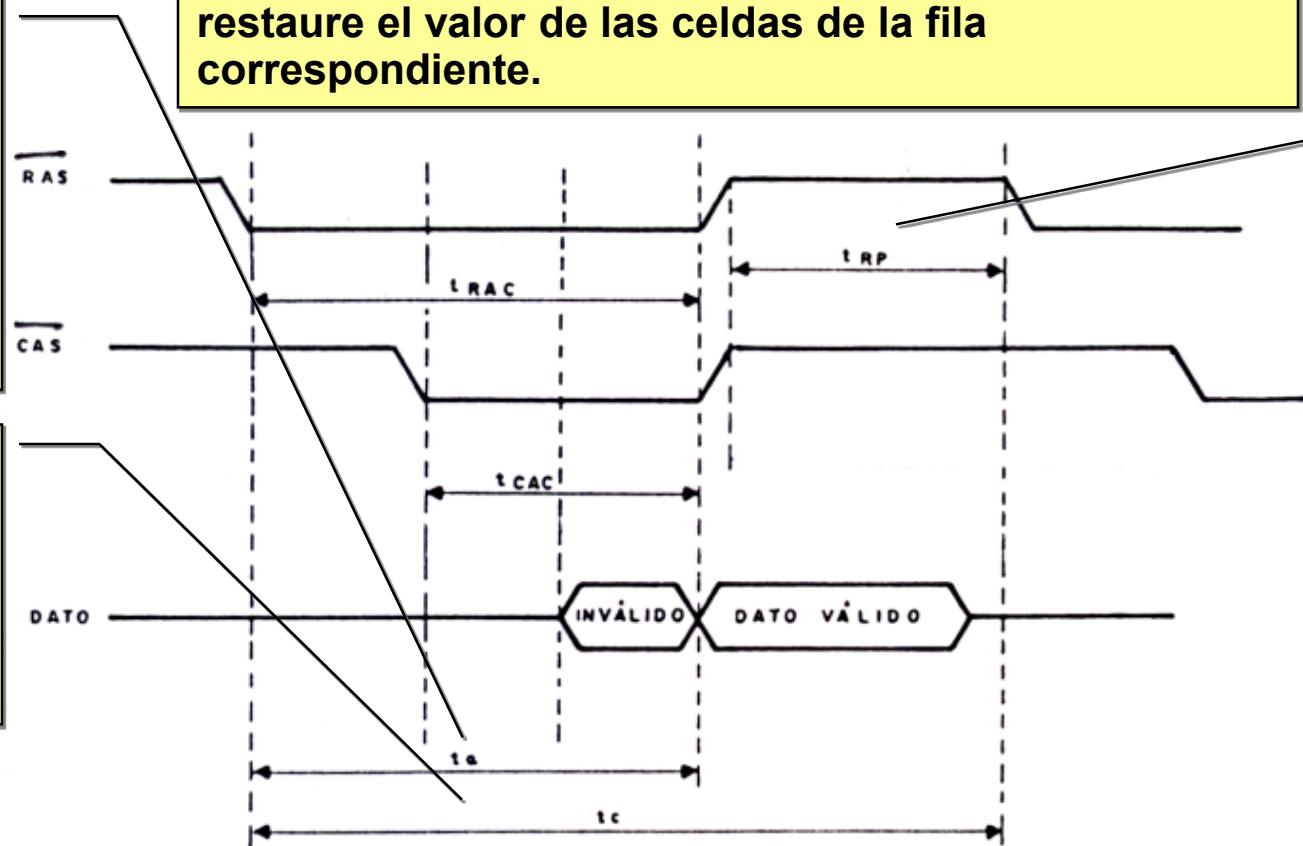
tiempo desde que se inicia una lectura ($/RAS \downarrow$) hasta que el dato está disponible en el bus de datos.

t_c (tiempo de ciclo):
 tiempo mínimo entre dos accesos consecutivos.

$$t_c \approx t_{RAC} + t_{RP}$$

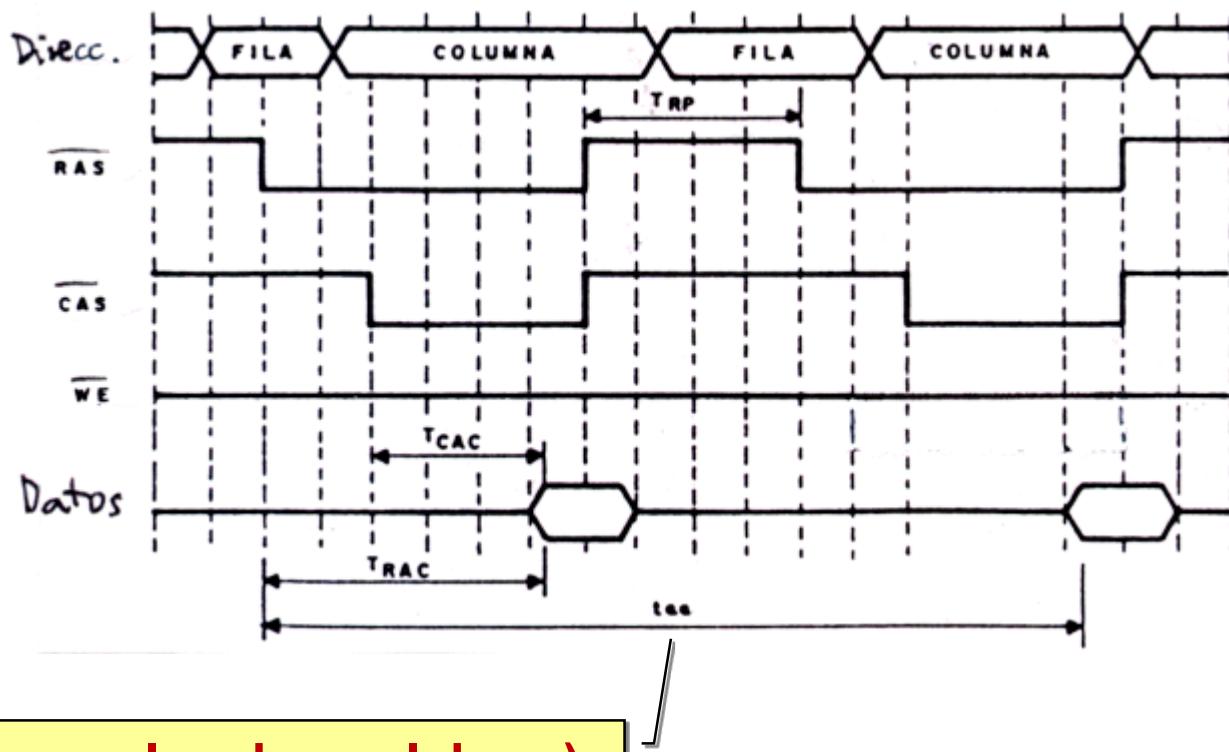
$$t_c > t_a$$

No se puede comenzar un segundo acceso tras t_a , dado que las lecturas son destructivas (el condensador de la celda de memoria se descarga) y es necesario esperar **t_{RP} (tiempo de precarga de fila)** para que la circuitería interna de la DRAM restaure el valor de las celdas de la fila correspondiente.



DRAM

- Acceso a dos palabras:



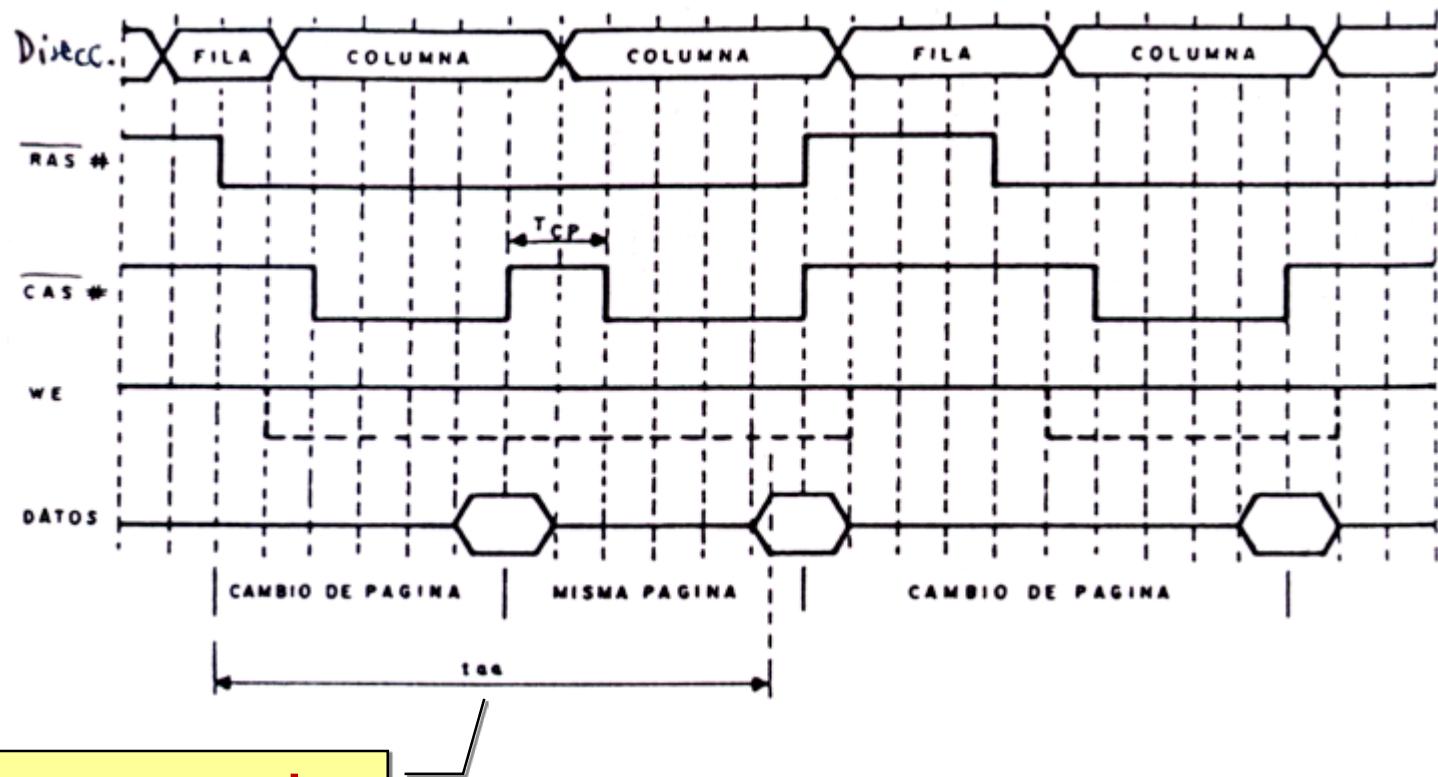
t_{aa} (tiempo de acceso a las dos palabras):

$$t_{aa} \approx t_{RAC} + t_{RP} + t_{RAC}$$

Ej.: $t_{aa} \approx 80 \text{ ns} + 60 \text{ ns} + 80 \text{ ns} = 220 \text{ ns}$

DRAM FPM

- Acceso en modo página rápido (*Fast Page Mode*):



t_{aa} (tiempo de acceso a dos palabras en modo página):

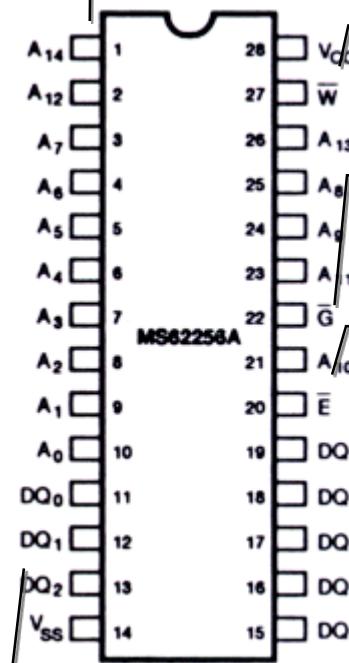
$$t_{aa} \approx t_{RAC} + t_{CP} + t_{CAC}$$

Ej.: $t_{aa} \approx 80 \text{ ns} + 10 \text{ ns} + 20 \text{ ns} = 110 \text{ ns}$

Ejemplo de SRAM

256 K bits (32 K palabras × 8 bits)

A₀-A₁₄: 15 señales de dirección para seleccionar una de 32768 palabras de 8 bits.



/W (Write Enable) ≡ /WE. Se utiliza para indicar lectura o escritura.

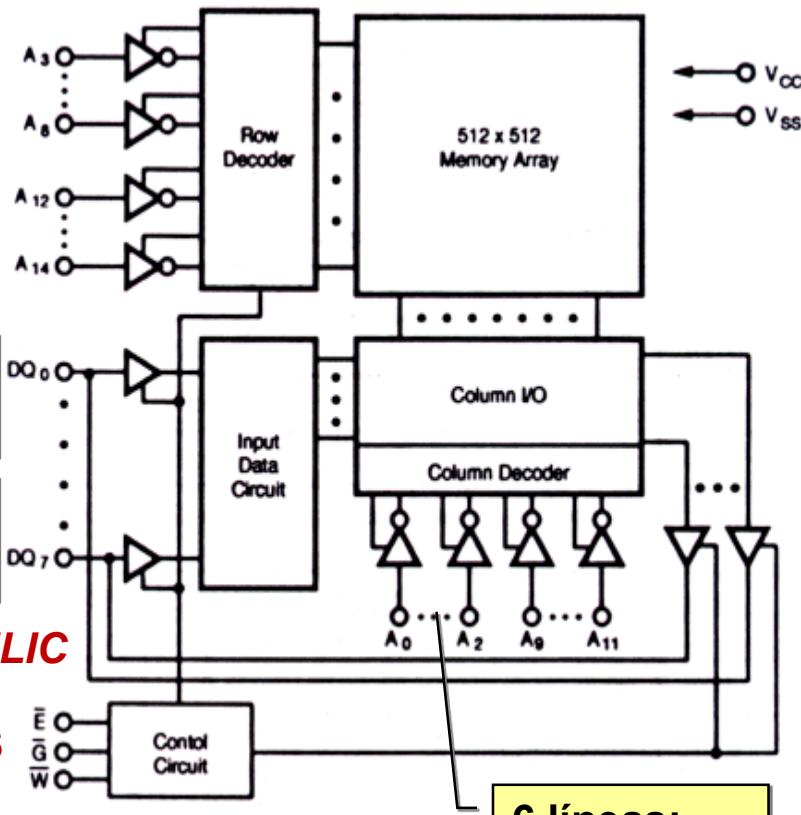
/G (Output Enable) ≡ /OE. Se selecciona (=0) para leer del chip.

/E (Chip Enable) ≡ /CS. Si no está activa ⇒ el chip no está seleccionado y permanece en “standby”, pasando su consumo de 900 mW a 50 mW.

**MOSEL-VITELIC
MS62256A
SRAM CMOS
32K × 8**

Mode	\bar{E}	\bar{G}	\bar{W}	I/O Operation
Standby	H	X	X	High Z
Read	L	L	H	D _{OUT}
Output Disabled	L	H	H	High Z
Write	L	X	L	D _{IN}

DQ₀-DQ₇ (Data Input/Output)



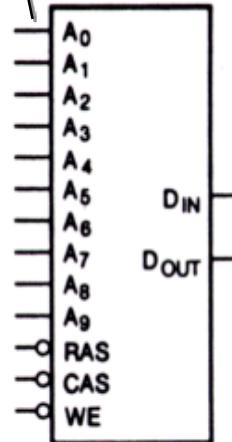
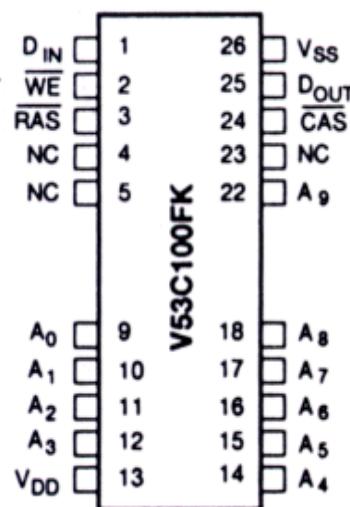
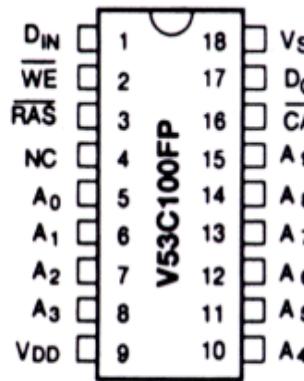
6 líneas:
 $9 - 6 = 3 \Rightarrow$
 se lee o escribe una palabra de $2^3 = 8$ bits

Ejemplo de DRAM

FPM 1 M bit (1 M palabras × 1 bit)

Una dirección (20 bits) se divide (multiplexada en el tiempo) en 10 bits para la fila y 10 para la columna.

Tiempo de acceso: 60 ns
 Tiempo de ciclo: 120 ns
 T. de ciclo en modo página: 40 ns



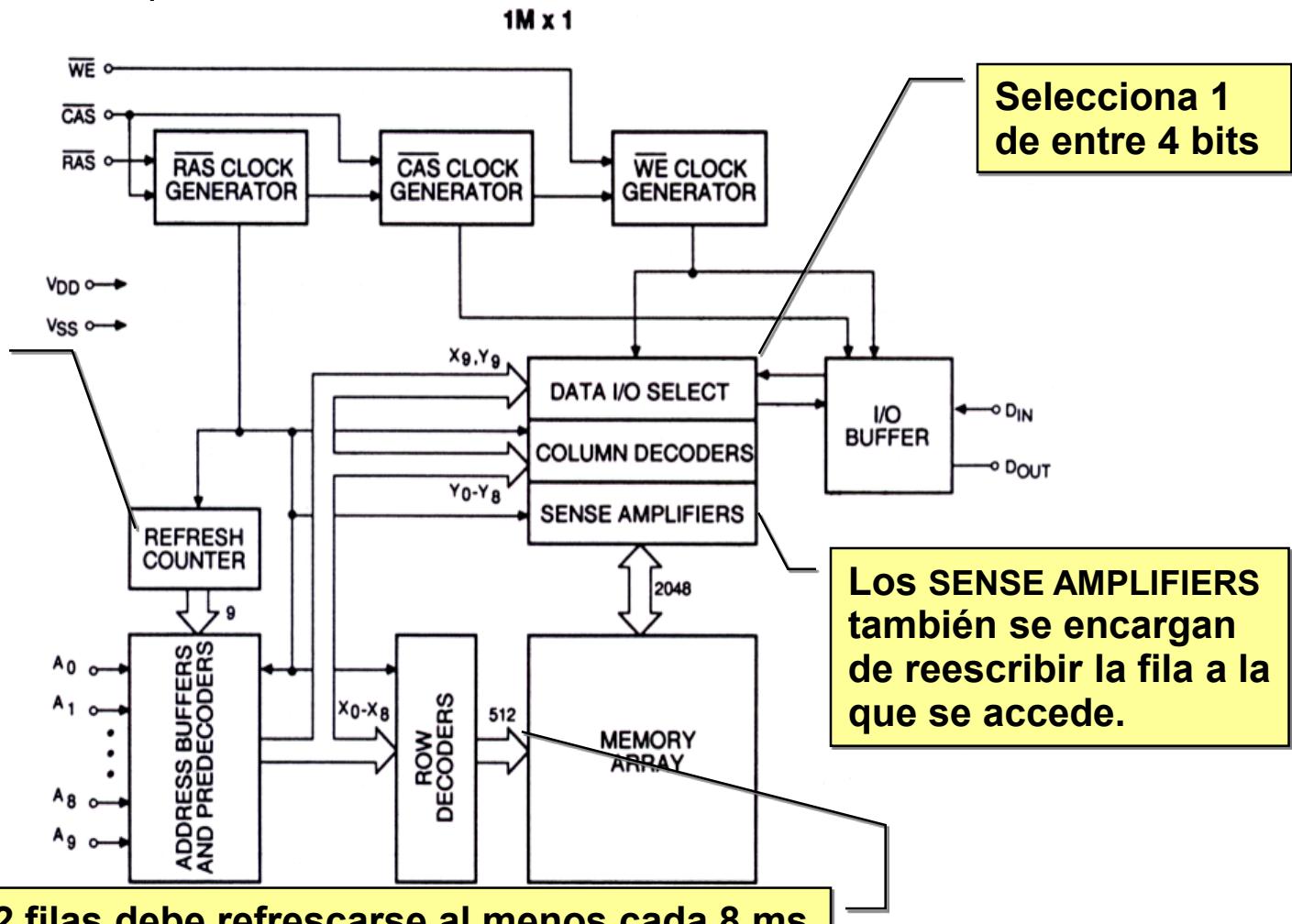
(Símbolo lógico)

A ₀ -A ₉	Address Inputs
RAS	Row Address Strobe
CAS	Column Address Strobe
WE	Write Enable
D _{IN}	Data Input
D _{OUT}	Data Output
V _{DD}	+5V Supply
V _{SS}	0V Supply
NC	No Connect

MOSEL-VITELIC
FAMILIA V53C100F
DRAM CMOS
FAST PAGE MODE
1M × 1 BIT

Ejemplo de DRAM

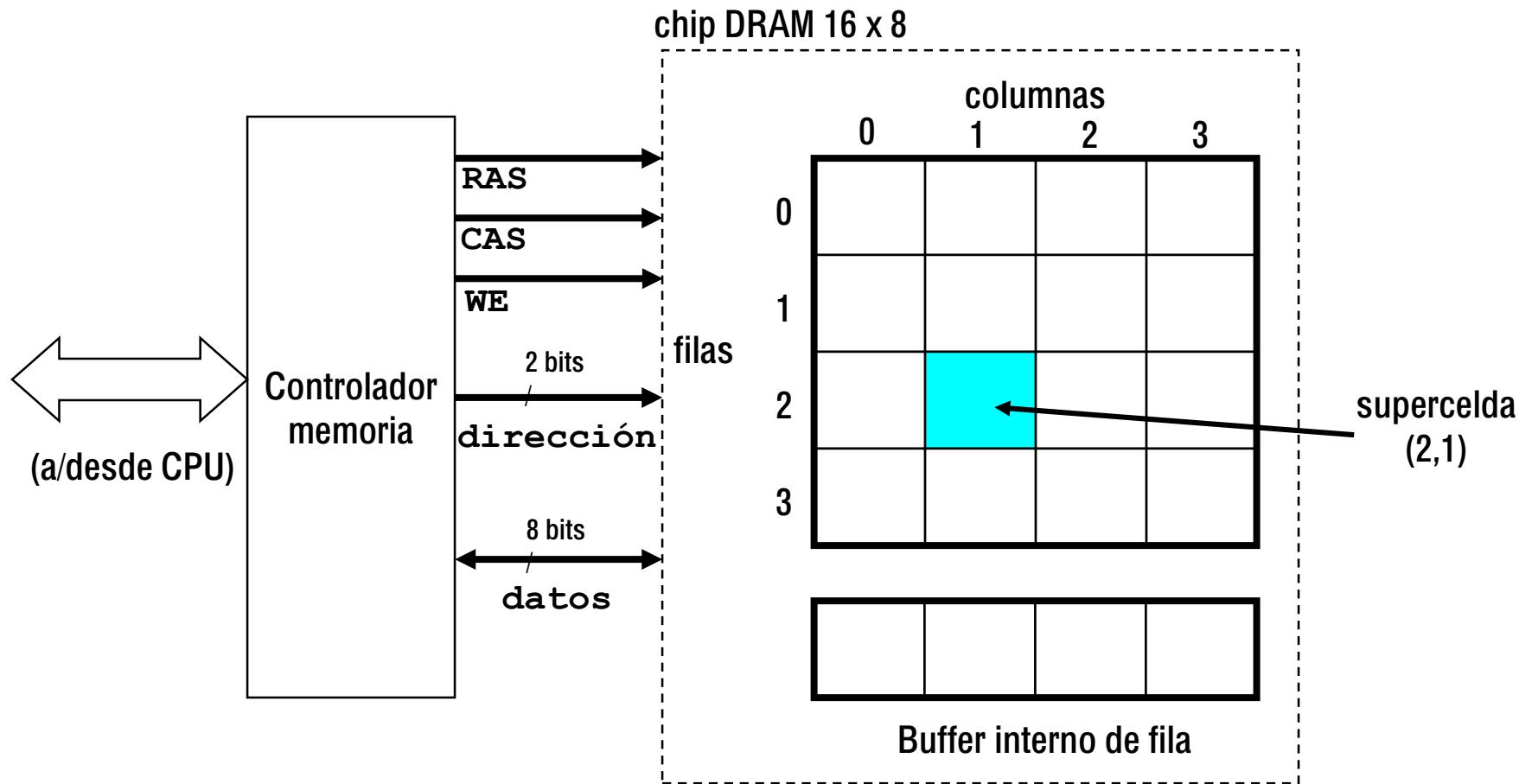
- Diagrama de bloques funcionales:



Organización DRAM convencional

■ DRAM $d \times w$:

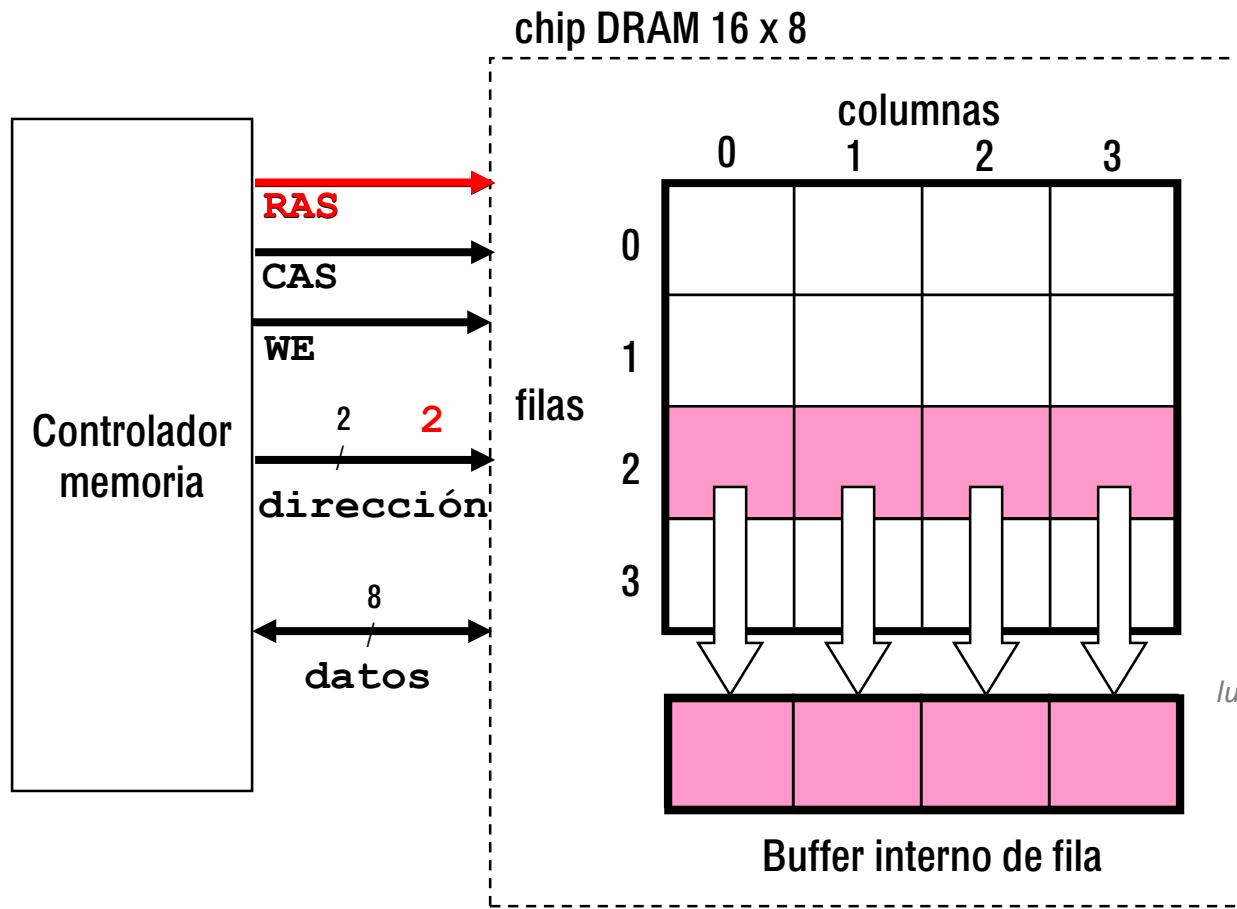
- $d \cdot w$ bits total organizados como d superceldas de tamaño w bits



Lectura de Supercelda DRAM (2,1)

Paso 1(a): **RAS** (Row address strobe[†], comando *Activate*) selecciona fila 2

Paso 1(b): Fila 2 copiada de array DRAM a buffer de fila



[†] strobe =
luz q. parpadea rápidamente
traducir como “señal
dirección fila/columna”
desde SDRAM, 3/4 líneas
para formar un comando
RAS, CAS, WE, A10

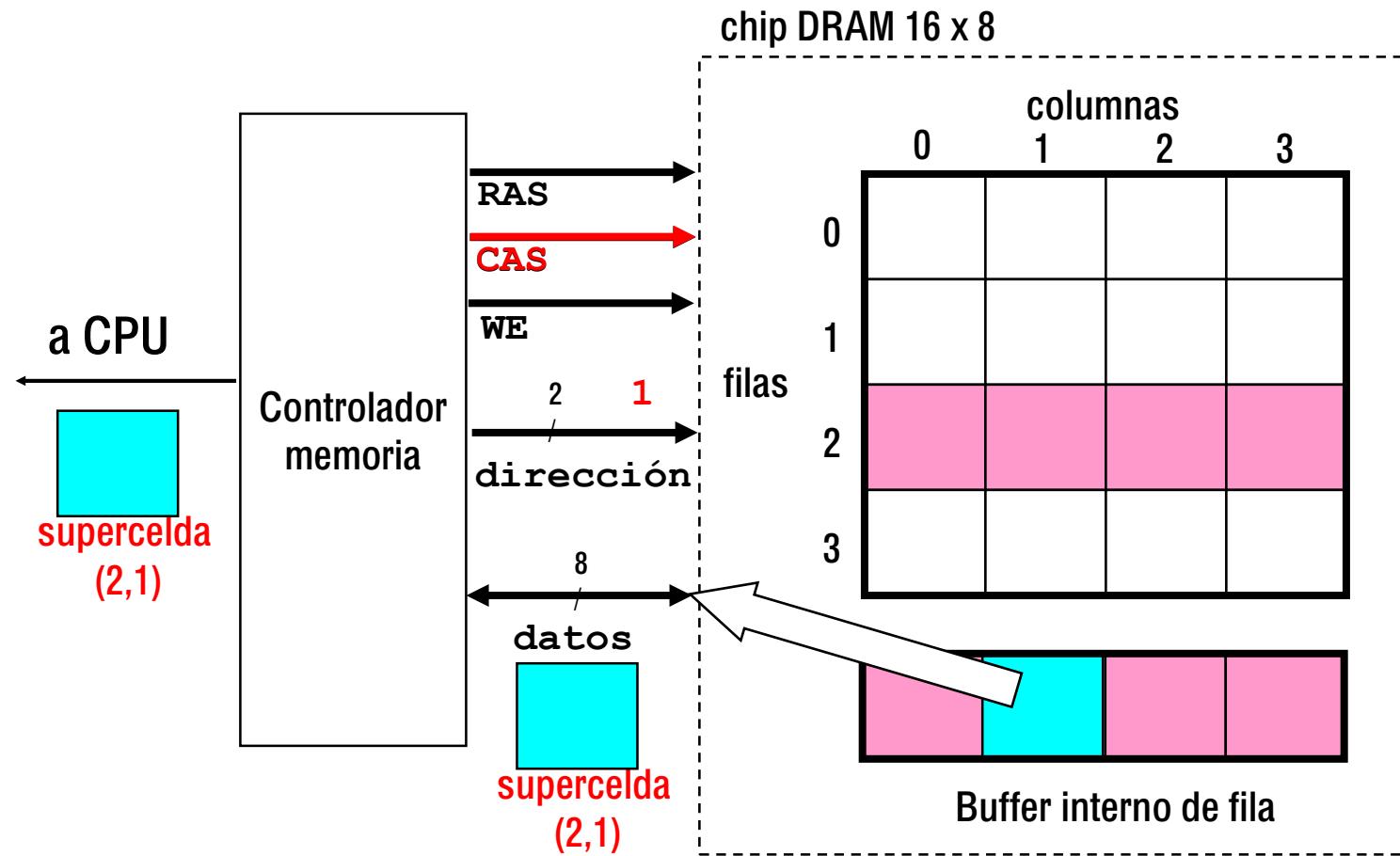
Ver https://en.wikipedia.org/wiki/Synchronous_dynamic_random-access_memory#Commands

Lectura de Supercelda DRAM (2,1)

Paso 2(a): **CAS** (Column address strobe, comando *Read*) selecciona columna 1

Paso 2(b): Supercelda (2,1) copiada de buffer a líneas bus datos → CPU.

Paso 3: Buffer copiado de vuelta a la fila para refreshar datos



Resumen SRAM vs DRAM

	Trans. por bit	tiempo acceso	necesita refresco?	EDC [#] ?	Coste	Aplicaciones
SRAM	6 ú 8	1x	No	Quizás	100x	memoria cache
DRAM	1	10x	Sí	Sí	1x	memoria principal, frame buffers [†]

EDC[#]: Error detection and correction

■ Tendencias

- La SRAM escala con la tecnología de semiconductores
 - está llegando a sus límites
- Escalado DRAM limitado por mínima capacidad necesaria C (condensador)
 - razón de aspecto limita cómo de profundo se puede hacer el C
 - también llegando a su límite

[#] Detección y corrección de errores

[†] buffer de fotogramas (vídeo)

DRAMs mejoradas

- **Funcionamiento celda DRAM no ha cambiado desde invención**
 - Comercializada por Intel en 1970
- **Núcleos DRAM con mejor lógica de interfaz y E/S más rápida:**
 - DRAM síncrona (**SDRAM**)
 - Usa una señal de reloj convencional en lugar de control asíncrono
 - puede aceptar un comando y transferir una palabra en cada ciclo reloj
 - puede hacer pipeline accediendo concurrentemente a distintos bancos
 - Reinterpreta señales RAS/CAS/WE/A10 como comando (Activate, Read...)†
 - Añade 1..3 bits selección banco (BA_{0-2}) (2,4,8 “Memory Arrays”)‡
 - DRAM síncrona a doble velocidad datos (double data-rate, **DDR SDRAM**)
 - temporización a doble flanco envía 2 bits por ciclo por pin
 - diferentes tipos según el tamaño de un pequeño buffer precaptación:
 - **DDR** (2 bits), **DDR2** (4 bits), **DDR3** (8 bits), **DDR4** (8[†] bits)
 - DDR4 cambia formato comandos (ACT, RAS/CAS/WE= A_{16-14} , BC/AP= $A_{12,10}$)
 - para 2010, estándar para mayoría sistemas sobremesa y servidor
 - Intel Core i7 admite SDRAM DDR3 y/o DDR4 (según modelo)

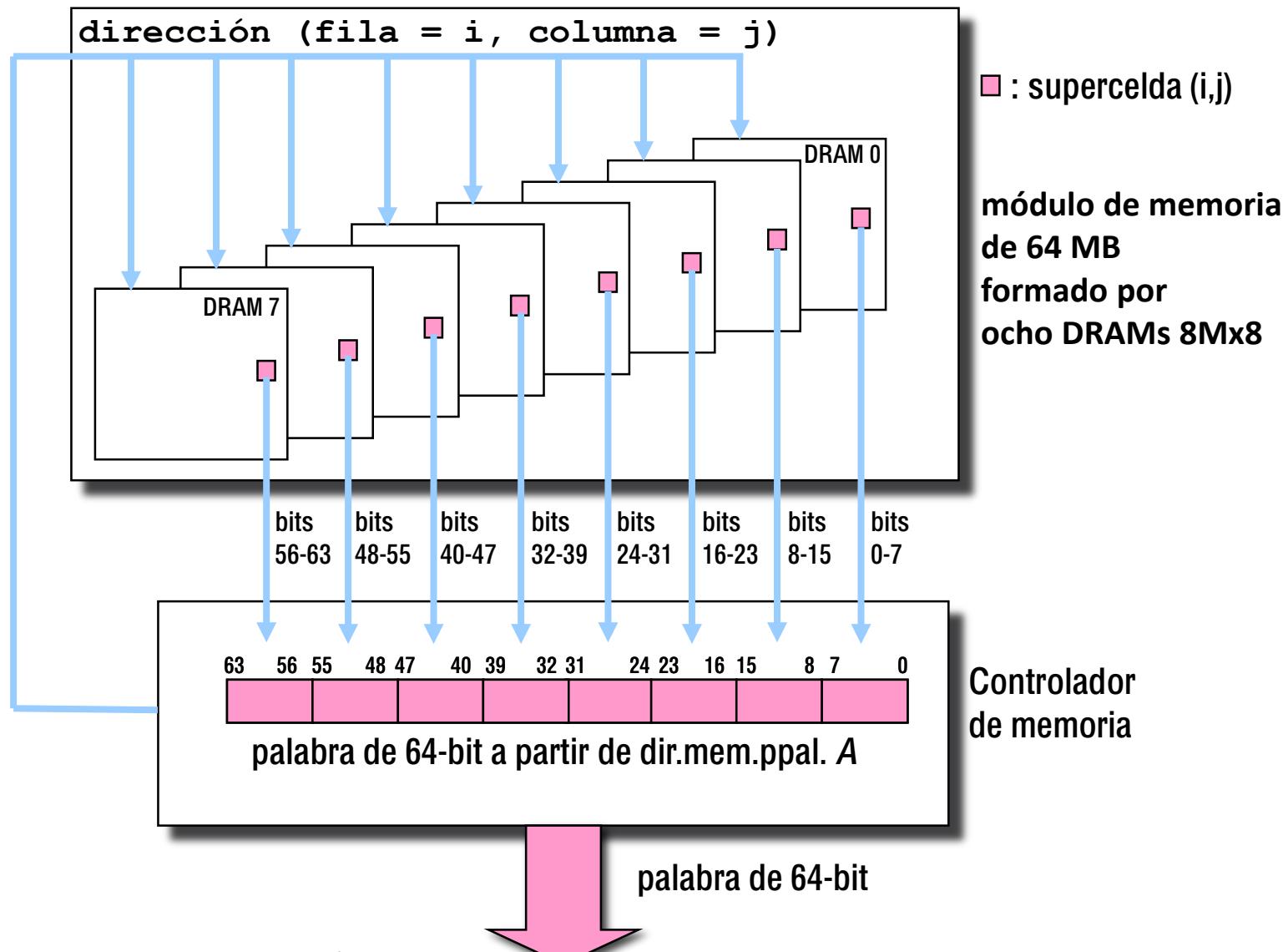
† Ver https://en.wikipedia.org/wiki/Synchronous_dynamic_random-access_memory#Command_signals

Los entusiastas pueden consultar la tesis enlazada en “See also”: http://drum.lib.umd.edu/bitstream/1903/11269/1/Gross_umd_0117N_11844.pdf

https://en.wikipedia.org/wiki/DDR4_SDRAM#Command_encoding † formato de los comandos

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition .../DDR4_SDRAM#Features † 8n / bank group interleave 34.

Módulos de Memoria



Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- **Configuración y diseño de memorias utilizando varios chips**
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Ampliación de memoria

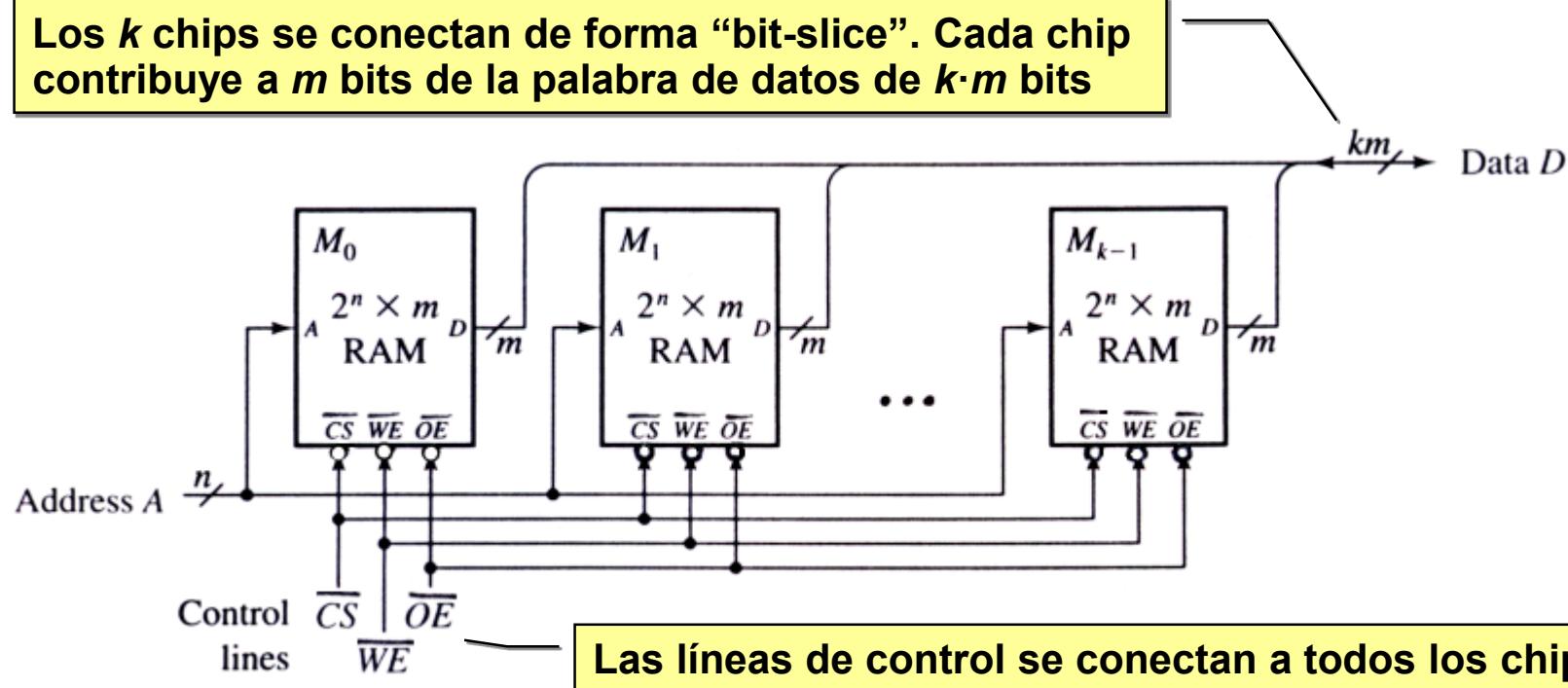
■ Problema:

- Construir una memoria de 2^N palabras de M bits a partir de chips de 2^n palabras de m bits.

① Incrementar el ancho de palabra de m a $k \cdot m = M$

- Necesitamos k circuitos de tamaño de palabra m :

Los k chips se conectan de forma “bit-slice”. Cada chip contribuye a m bits de la palabra de datos de $k \cdot m$ bits



Ampliación de memoria

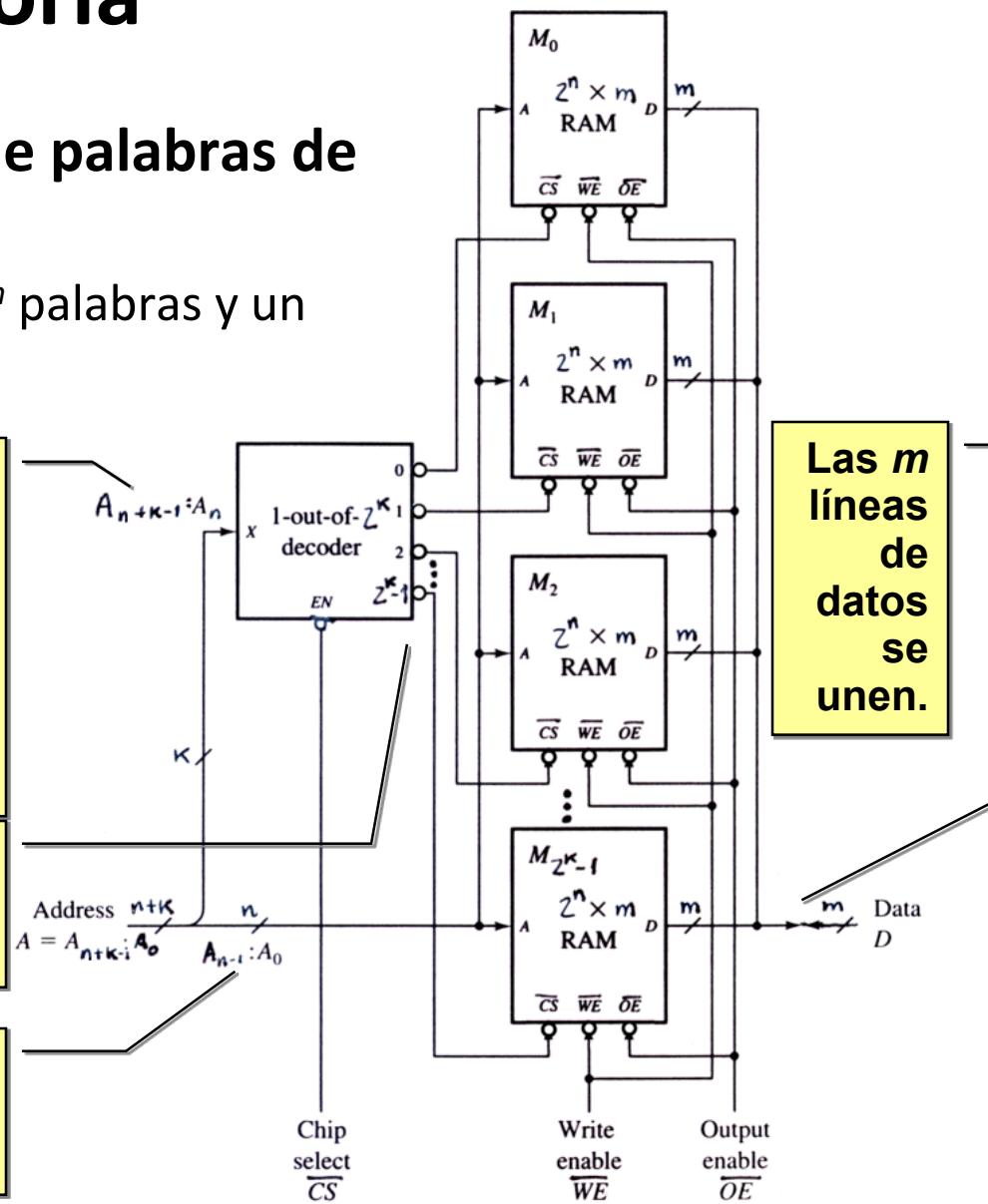
② Incrementar el número de palabras de 2^n a $2^{n+k}=2^N$

- Necesitamos 2^k circuitos de 2^n palabras y un decodificador de 1 entre 2^k .

Las k líneas de dirección de orden superior ($A_{n+k-1}:A_n$) se conectan a las entradas de un decodificador de k a 2^k
 \Rightarrow cada configuración de los $n+k$ bits hace que se seleccione solamente el circuito RAM indicado por los bits de dirección $A_{n+k-1}:A_n$.

Las 2^k líneas de salida del decodificador se conectan a las entradas de selección /CS de los 2^k circuitos.

Las n líneas de dirección de orden inferior se conectan en común a las líneas de dirección de los 2^k chips.



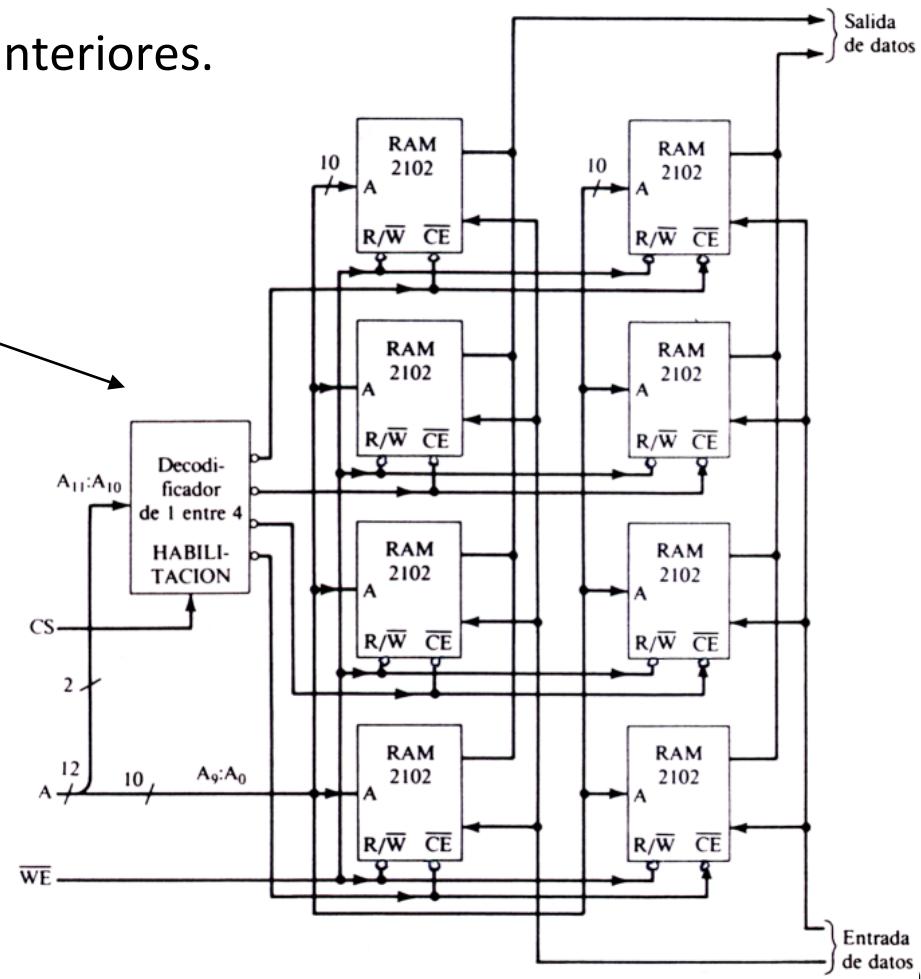
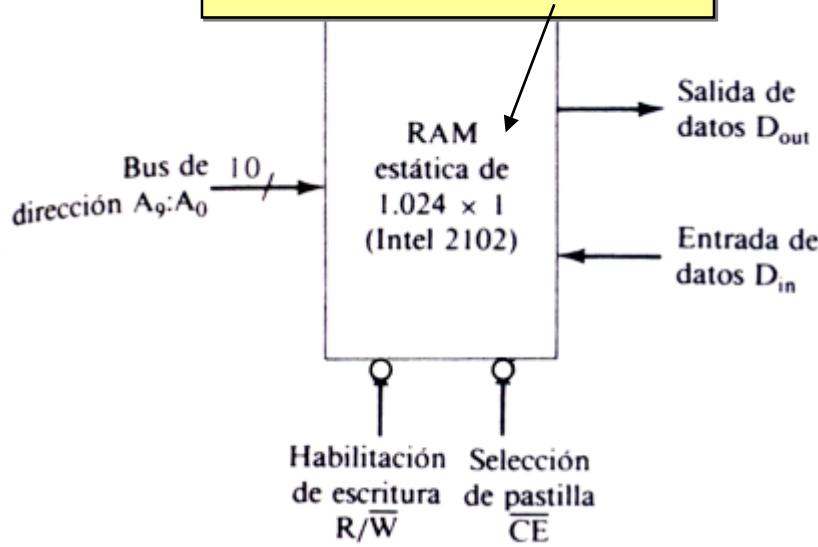
Ampliación de memoria

③ Incrementar simultáneamente el número de palabras y el ancho de palabra.

- Basta combinar las dos técnicas anteriores.

Ejemplo 1:

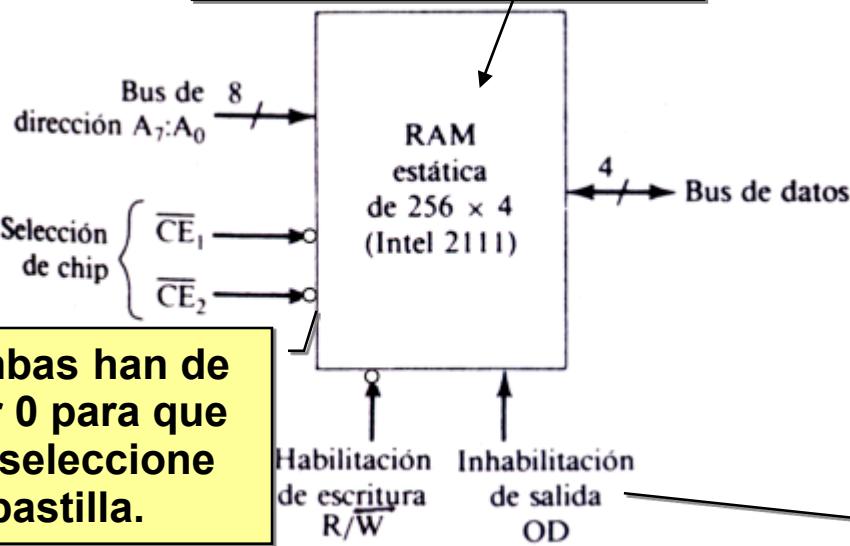
RAM de 4096×2 construida con 8 RAM de 1024×1 .



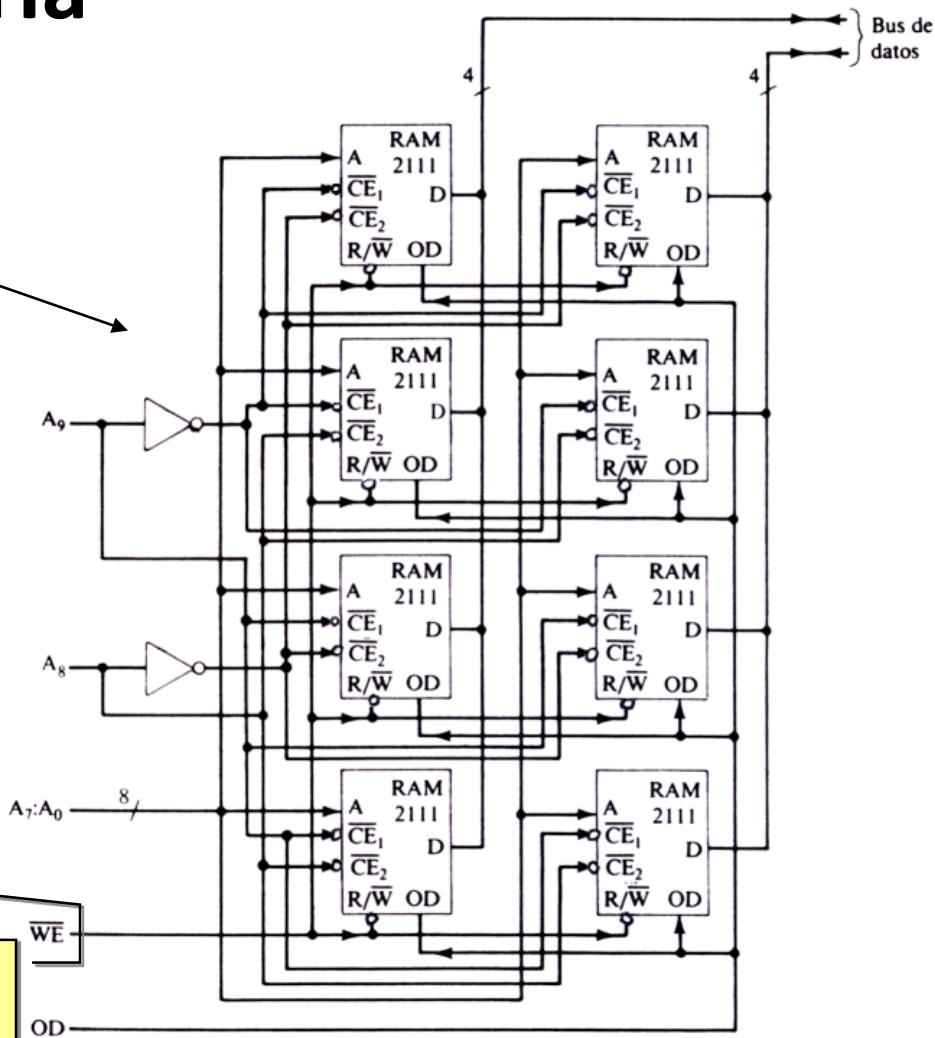
Ampliación de memoria

Ejemplo 2:

**RAM de $1\text{ K} \times 8$
construida con 8
RAM de 256×4 .**



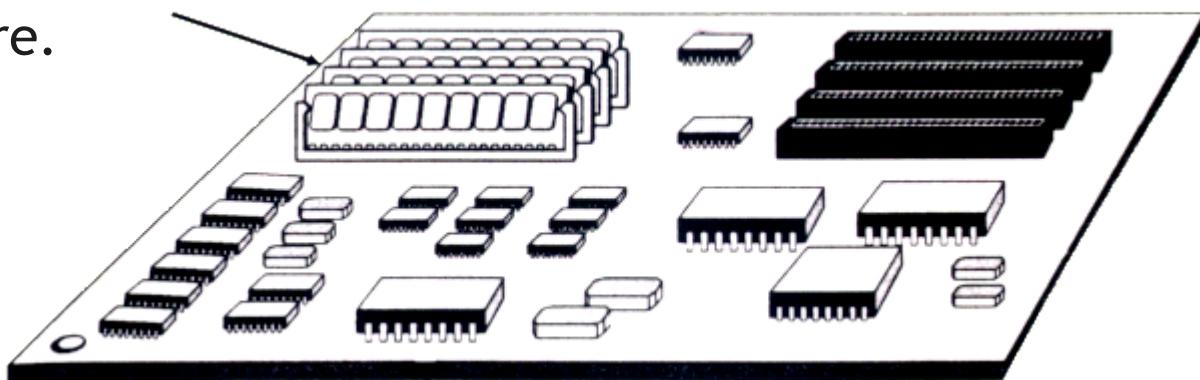
Como hay un solo bus de datos \Rightarrow la línea OD (*Output Disable*) se activa durante los ciclos de escritura para evitar que se lean datos internos desde el chip al bus mientras el procesador usa el bus para salida datos.



Módulos de memoria en línea

- En los 80, la memoria solía soldarse directamente en la placa madre del ordenador. Pero a medida que aumentaron los requisitos de memoria, esta técnica resultó poco factible.
- SIMM, DIMM, SODIMM:

- Varios chips DRAM en una pequeña placa de circuito impreso (PCB) que calza en un conector en la placa madre.



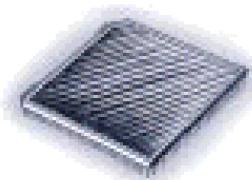
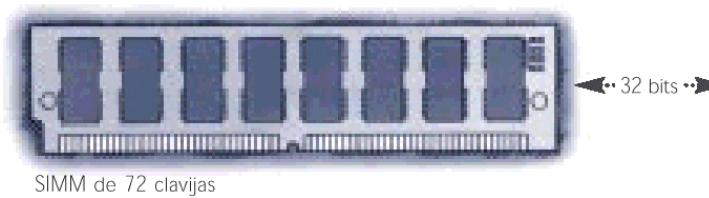
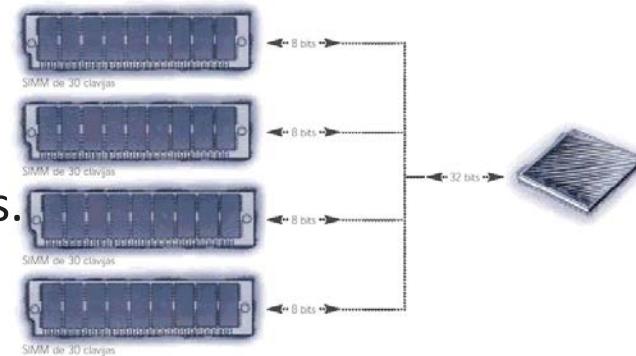
- ✓ método flexible para actualizar la memoria
- ✓ ocupa menos espacio en la placa madre.
- Normalmente sólo se ampliaba el bus de datos, no el de direcciones (no había necesidad de decodificador).

Módulos de memoria SIMM



■ **SIMM (Single In-line Memory Module)**

- En un SIMM, cada contacto de una cara está conectado con el alineado en la otra cara para formar un único contacto.
- 80's y 90's, FPM y EDO-RAM
- 30 contactos:
 - 8 bits de datos \Rightarrow 4 SIMM para bus 32 bits.
 - 12 pines dirección \Rightarrow 24 bits dir \Rightarrow 16MB máx.
- 72 contactos:
 - 32 bits de datos (24 bits dir) \Rightarrow 64MB máx.
 - Dos rangos[†] (con /RAS1, /RAS3) \Rightarrow 128MB máx.

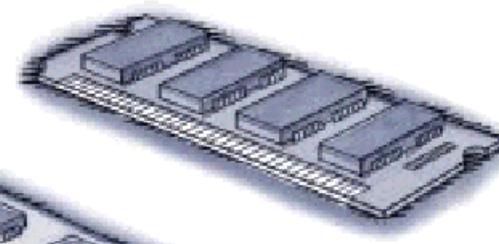


[†] Ver https://en.wikipedia.org/wiki/SIMM#72-pin_SIMMs
https://en.wikipedia.org/wiki/Memory_rank.
 se usaban señales RAS adicionales para duplicar tamaño sin aumentar pines dirección

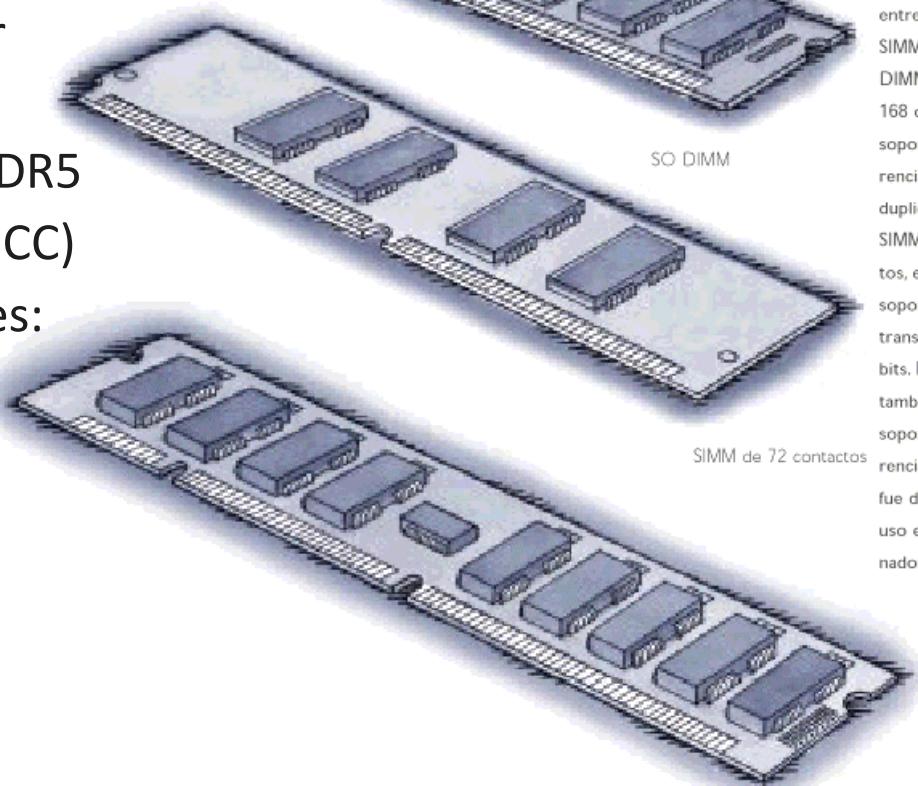
Módulos de memoria DIMM

■ **DIMM (Dual In-line Memory Module)**

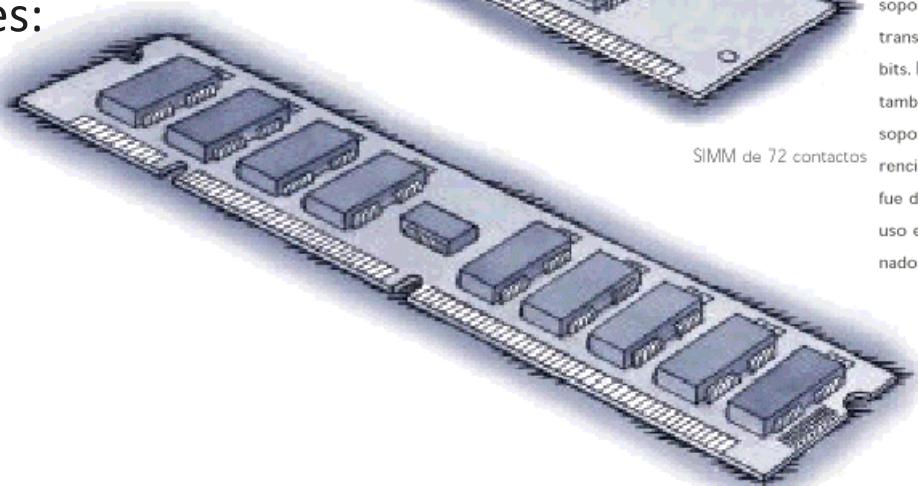
- En DIMM los contactos opuestos están aislados eléctricamente para formar **dos contactos separados**.
- 90's-hoy, SDRAM-DDR-...-DDR5
- 64 bits de datos (ó 72 con ECC)
- Incremento progresivo pines:
 - 168-pin: FPM, EDO y SDRAM
 - 184-pin: DDR
 - 240-pin: DDR2, DDR3
 - 288-pin: DDR4, DDR5



SO-DIMM



SIMM de 72 contactos



DIMM de 168 contactos

Los tres ejemplos ilustran las diferencias entre los productos SIMM, DIMM y SO DIMM. El DIMM de 168 contactos brinda soporte para transferencias de 64 bits, sin duplicar el tamaño del SIMM de 72 contactos, el cual brinda soporte sólo para transferencias de 32 bits. El SO DIMM también brinda soporte para transferencias de 32 bits y fue diseñado para su uso en los ordenadores portátiles.

Módulos de memoria SODIMM



■ SODIMM (*Small Outline DIMM*)

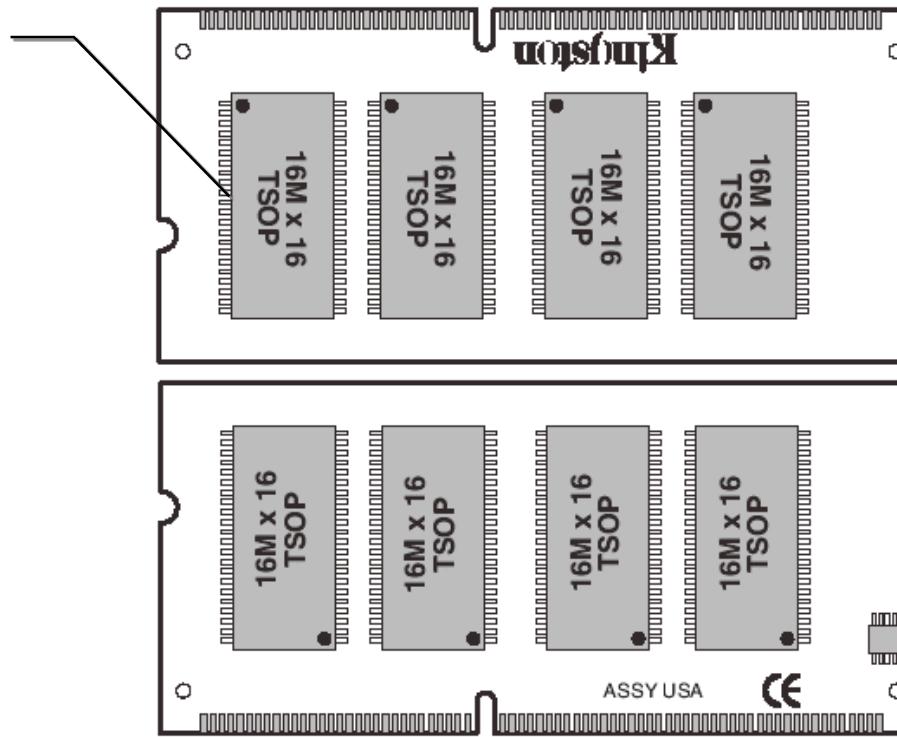
- Menos tamaño que un DIMM, menos contactos.
- Uso en portátiles. (100/144-pin SDR, 200-pin DDR-DDR2, 204-pin DDR3, 260-pin DDR4)
- Ejemplo: SODIMM SDR (144-pin) de 256 MB ($2 \times 16M \times 64$) a 133 MHz:

8 chips (4 en cada cara)
SDRAM a 133 MHz de
 $16M \times 16$

Cada chip tiene 4
bancos de $4M \times 16$

Bus de datos: 64 bits =
 $4 \text{ chips} \times 16 \text{ bits/chip}$

El módulo tiene 2 ranks
(¡en este caso sí se
amplía el número de
direcciones!)

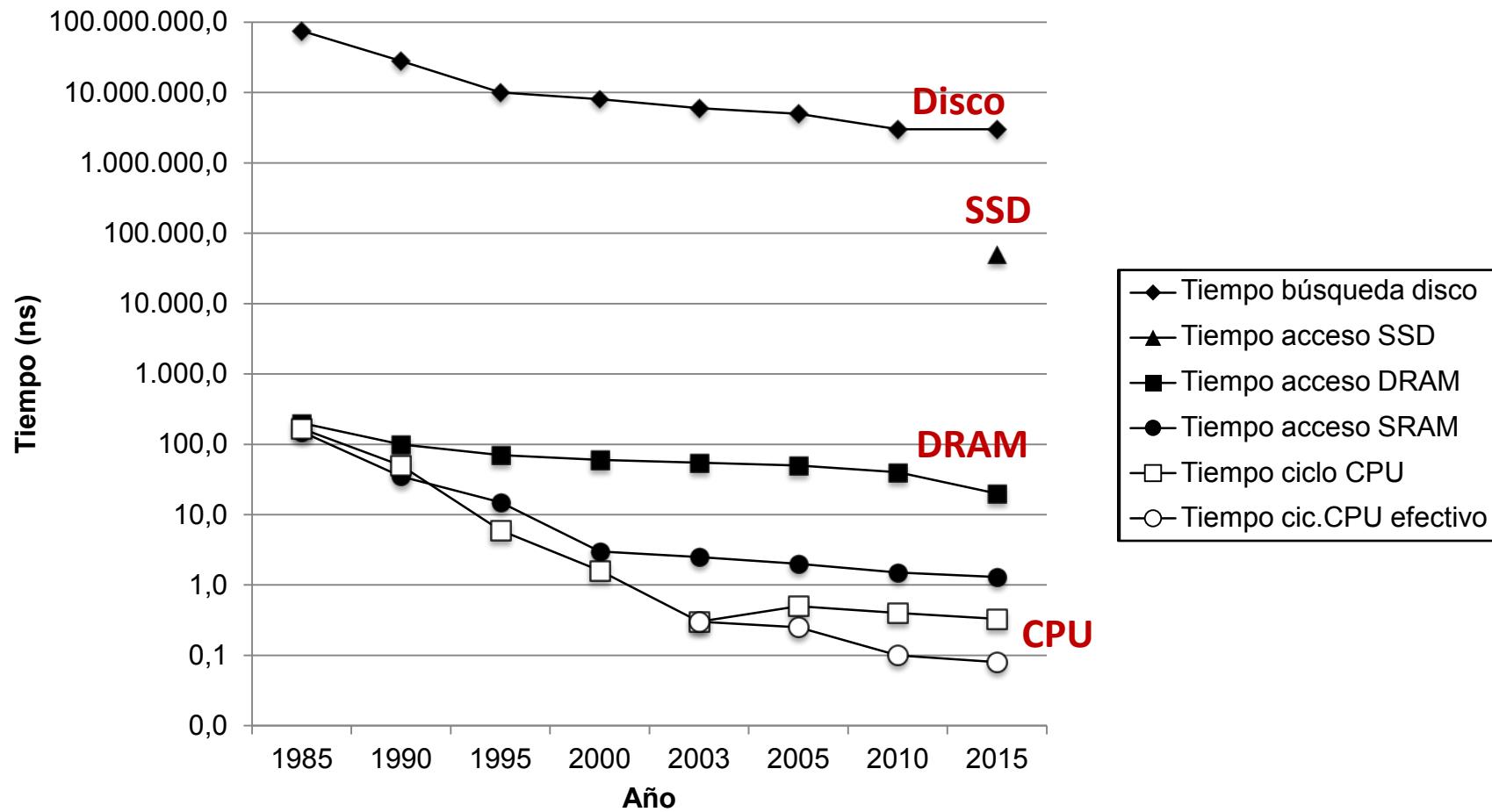


Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

La brecha[†] CPU-Memoria

La brecha entre velocidades de disco, DRAM y CPU se ensancha.

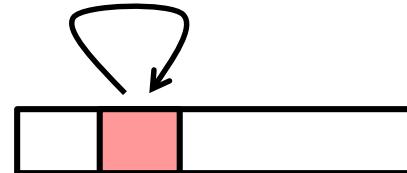


¡Localidad al rescate!

La clave para salvar esta brecha CPU-Memoria es una propiedad fundamental de los programas informáticos conocida como **localidad**

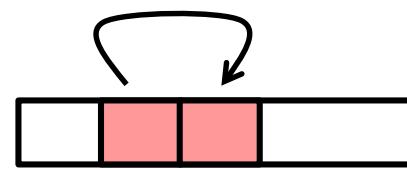
Localidad

- **Principio de localidad:** Los programas tienden a usar datos e instrucciones con direcciones iguales o cercanas a las que han usado recientemente



- **Localidad temporal:**

- Elementos referenciados recientemente probablemente serán referenciados de nuevo en un futuro próximo



- **Localidad espacial:**

- Elementos con direcciones cercanas tienden a ser referenciados muy juntos en el tiempo

Expresión matemática de localidad

- si en instante tiempo t se accede al dato/posición mem. $d(t) \dots$
- **Temporal**
 - $d(t + n) = d(t)$ con n pequeño
- **Espacial**
 - $d(t + n) = d(t) + k$ con n, k pequeños

Ejemplo de Localidad

```
sum = 0;  
for (i = 0; i < n; i++)  
    sum += a[i];  
return sum;
```

■ Referencias a datos

- Referenciar elementos array en sucesión
(patrón de referencias de paso-1[†])
- Referenciar variable **sum** cada iteración

Localidad

Espacial o Temporal?

espacial

temporal

■ Referencias a instrucciones

- Referenciar instrucciones en secuencia
- Iterar bucle repetidamente

espacial

temporal

[†] stride-1 reference pattern

stride = paso, zancada

50

Estimaciones cualitativas de localidad

- **Afirmación:** Ser capaz de mirar un código y sacar una idea cualitativa de su localidad es una habilidad clave para un programador profesional
- **Pregunta:** ¿Tiene esta función buena localidad respecto al array a?

Pista: alm. por filas
(row-major order)

Respuesta: sí

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

a [0] [0]	· · ·	a [0] [N-1]	a [1] [0]	· · ·	a [1] [N-1]	· · ·	a [M-1] [0]	· · ·	a [M-1] [N-1]
-----------------	-------	-------------------	-----------------	-------	-------------------	-------	-------------------	-------	---------------------

Ejemplo de localidad

- **Pregunta:** ¿Tiene esta función buena localidad respecto al array a?

```
int sum_array_cols(int a[M] [N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i] [j];
    return sum;
}
```

Respuesta: no, salvo si...

N es muy pequeño
...o M muy muy pequeño
y N no muy grande

a [0] [0]	· · ·	a [0] [N-1]	a [1] [0]	· · ·	a [1] [N-1]	· · ·	a [M-1] [0]	· · ·	a [M-1] [N-1]
-----------------	-------	-------------------	-----------------	-------	-------------------	-------	-------------------	-------	---------------------

Ejemplo de Localidad

- **Pregunta:** Se pueden permutar los bucles de forma que la función recorra el array 3-d a con patrón de referencias de paso-1 (y tenga por tanto buena localidad espacial)?

```
int sum_array_3d(int a[M] [N] [N])  
{  
    int i, j, k, sum = 0;  
  
    for (i = 0; i < N; i++)  
        for (j = 0; j < N; j++)  
            for (k = 0; k < M; k++)  
                sum += a[k] [i] [j];  
    return sum;  
}
```

Respuesta: sí: for k,i,j, bucles en mismo orden índices
(segunda opción, peor) for i,k,j, haría NxM bucles (de N elems, paso-1)

Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- **Jerarquía de memoria**
- Tecnologías de almacenamiento, y tendencias

Jerarquía de Memoria

- **Algunas propiedades fundamentales y perdurables del hardware y software:**
 - Las tecnologías de almacenamiento rápidas cuestan más por byte, tienen menor capacidad y requieren más potencia (¡ calor!)
 - La brecha velocidad CPU-memoria principal se está ampliando
 - Los programas bien escritos tienden a exhibir buena localidad
- **Estas propiedades fundamentales se complementan muy convenientemente**
- **Sugieren un enfoque para organizar sistemas de memoria y almacenamiento conocido como jerarquía de memoria**

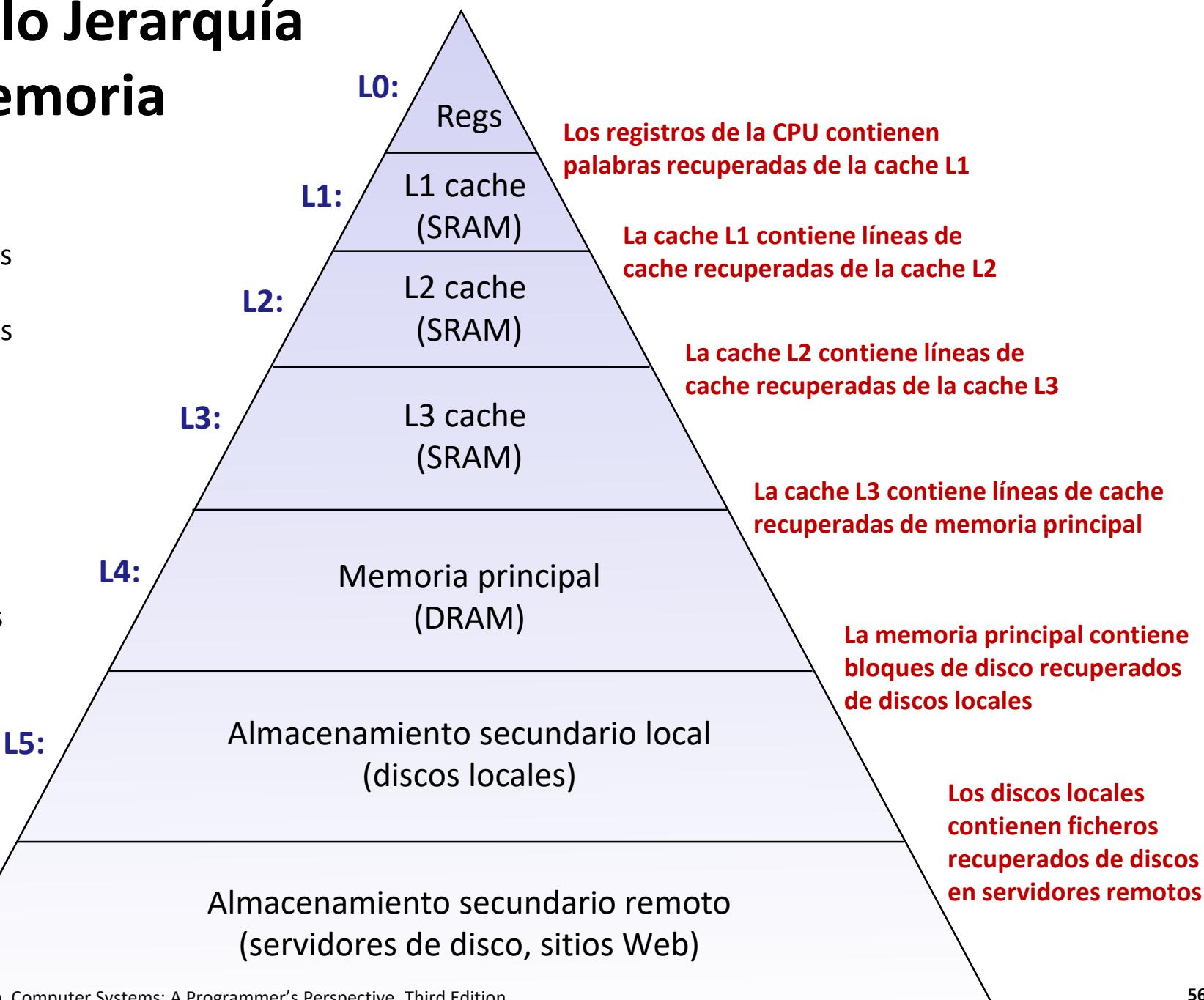
Ejemplo Jerarquía Memoria

dispositivos
almacnmtos.
más rápidos
más costosos
(por byte)
y + pequeños
(de menor
capacidad)

dispositivos
almacnmtos.
más lentos
más baratos
(por byte)
y + grandes
(de mayor
capacidad)

L6:

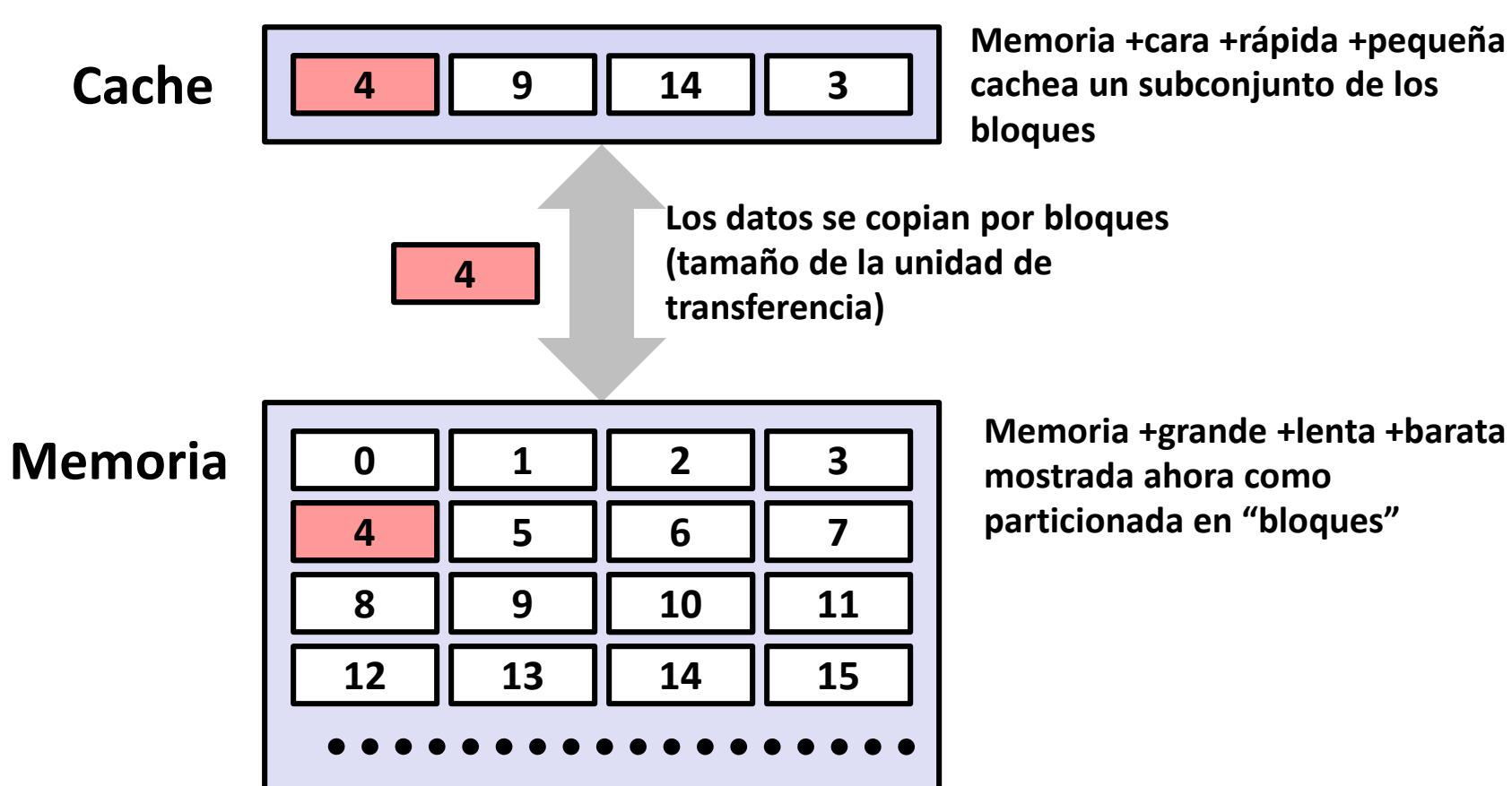
Almacenamiento secundario remoto
(servidores de disco, sitios Web)



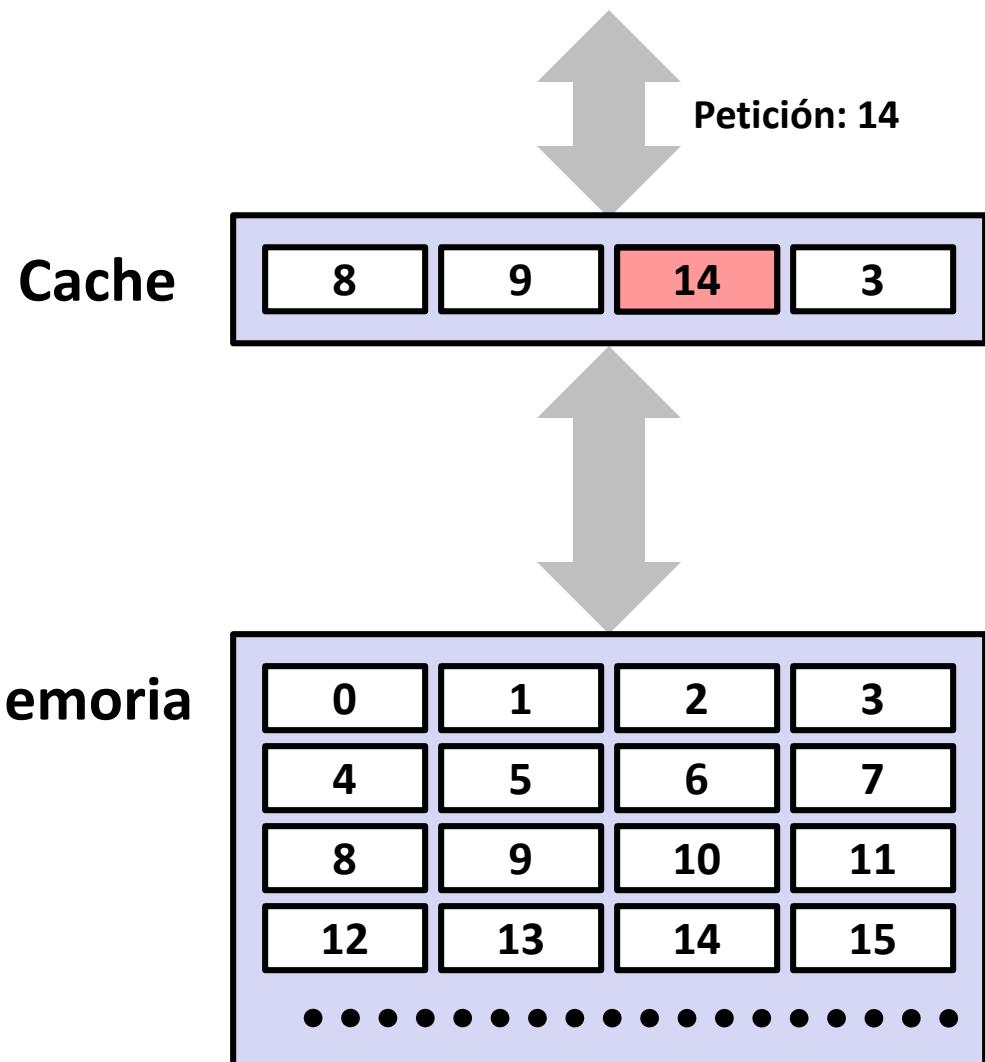
Caches

- ***Cache:*** Un dispositivo de almacenamiento más rápido y pequeño que funciona como zona de trabajo temporal para un subconjunto de los datos de otro dispositivo mayor y más lento
- Idea fundamental de una jerarquía de memoria:
 - $\forall k$, el dispositivo a nivel k (+rápido, +pequeño) sirve de cache para el dispositivo a nivel $k+1$ (+lento, +grande)
- ¿Por qué funcionan bien las jerarquías de memoria?
 - Debido a la localidad, los programas suelen acceder a los datos a nivel k más a menudo que a los datos a nivel $k+1$
 - Así, el almacenamiento a nivel $k+1$ puede ser más lento, y por tanto más barato (por bit) y más grande
- ***Idea Brillante (ideal):*** La jerarquía de memoria conforma un gran conjunto de almacenamiento que cuesta como el más barato pero que proporciona datos a los programas a la velocidad del más rápido

Conceptos Generales de Cache



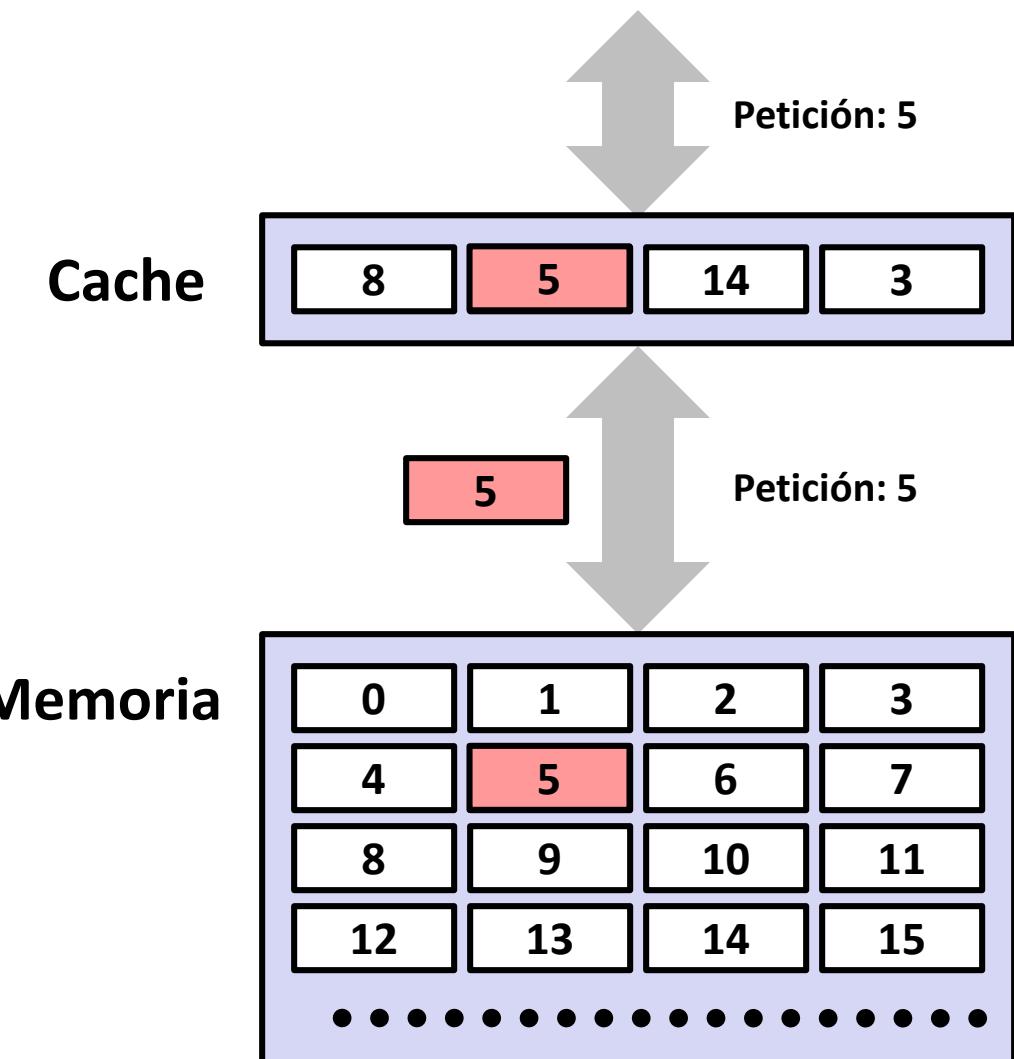
Conceptos Generales de Cache: Acierto[†]



*Se necesitan datos
del bloque b*

*El bloque b está en cache:
¡Acierto!*

Conceptos Generales de Cache: Fallo[†]



*Se necesitan datos
del bloque b*

*El bloque b no está en cache:
¡Fallo!*

*El bloque b se capta de
memoria*

*El bloque b se almacena
en cache*

- **Política de colocación[†]:** determina dónde va b
- **Política de reemplazo[†]:** determina qué bloque es desalojado[†] (víctima)

[†] cache miss

[re]placement policy,
evicted/victim block

Conceptos Generales de Cache:

3 Tipos de Fallo de Cache

■ Fallos en frío (obligados)

- Los fallos en frío ocurren porque la cache empieza vacía y esta es la primera referencia al bloque

■ Fallos por capacidad

- Ocurren cuando el conjunto de bloques activos (**conjunto de trabajo**) es más grande que la cache

■ Fallos por conflicto

- Mayoría caches limitan que los bloques a nivel $k+1$ puedan ir a pequeño subconjunto (a veces unitario) de las posiciones de bloque a nivel k
 - P.ej. Bloque i a nivel $k+1$ debe ir a bloque $(i \bmod 4)$ a nivel k (corr. directa)
- Fallos por conflicto ocurren cuando cache nivel k suficientemente grande pero a varios datos les corresponde ir al mismo bloque a nivel k
 - P.ej. Referenciar bloques 0, 8, 0, 8, 0, 8, ... fallaría continuamente (ejemplo anterior con correspondencia directa)

Ejemplos de cacheado en Jerarquía Memoria

Tipo de Cache	Qué se cachea?	Dónde se cachea?	Latencia (ciclos)	Gestionado por
Registros	Palabras de 4-8 B	CPU (on-chip)	0	Compilador
TLB [†]	Trad. de direcciones	TLB (on-chip)	0	MMU [†] (hardw)
cache L1	Bloques de 64 bytes	L1 (on-chip)	4	Hardware
cache L2	Bloques de 64 B	L2 (on-chip)	10	Hardware
cache L3	Bloques de 64 B	L3 (on-chip)	50	Hardware
Memoria Virtual	Páginas de 4 KB	Memoria principal	200	Hardware + SO
Buffer de disco	Partes de ficheros	Memoria principal	200	SO
cache de disco	Sectores de disco	Controladora de disco	100,000	Firmware disco
Buffer disco red	Partes de ficheros	Disco local	10,000,000	Cliente NFS [†]
cache Navegador	Páginas Web	Disco local	10,000,000	Navegador web
cache Web	Páginas Web	Discos de servidores remotos	1,000,000,000	Servidor web proxy [†]

[†] Translation Lookaside Buffer, Memory Management Unit,

Network File System, proxy=procurador, apoderado (intermediario) 62

¡Hora de juego!

Conectarse a:

<https://swad.ugr.es> > EC > Evaluación > Juegos

Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- **Tecnologías de almacenamiento, y tendencias**

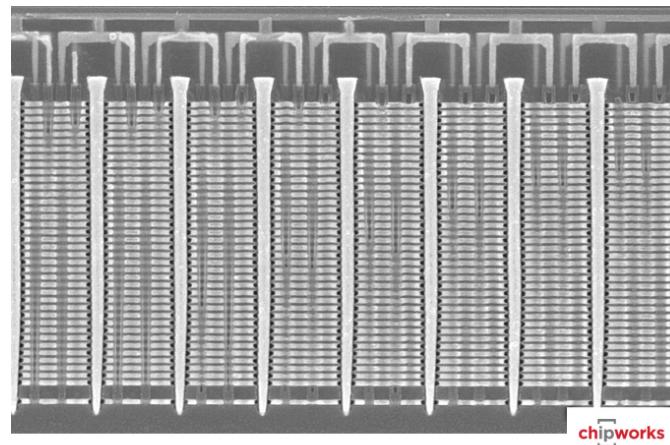
Tecnologías de almacenamiento

■ Discos Magnéticos



- Almacenamiento en medio magnético
- Acceso electromecánico

■ Memoria No-volátil (Flash)



- Almacenamiento como carga persistente
- Implementado con estructura 3-D[†]

- +100 niveles de celdas
- 3 bits de datos por celda

[†] Ver https://en.wikipedia.org/wiki/Flash_memory#Vertical_NAND

¿Qué hay dentro de una unidad de disco?

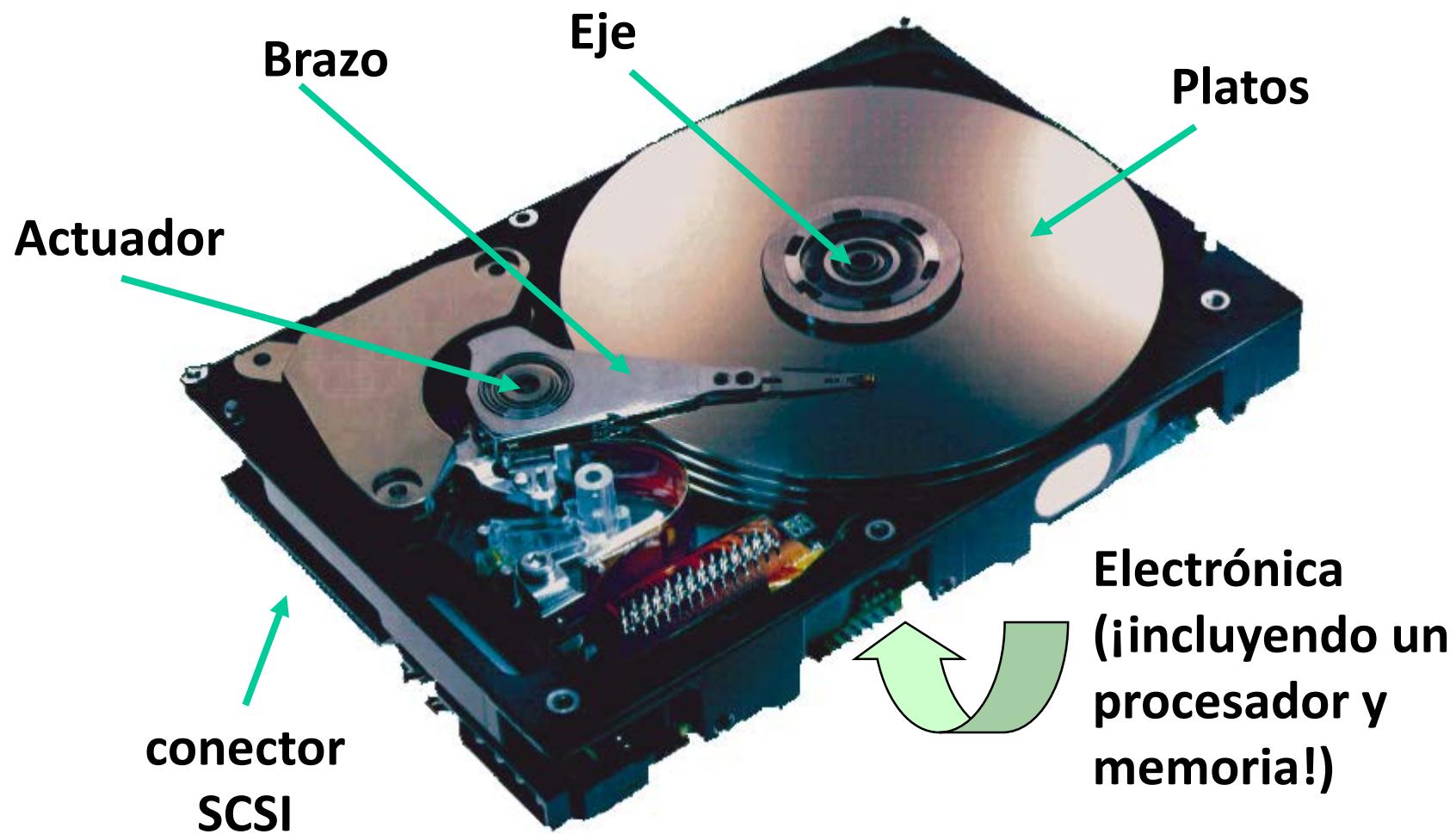
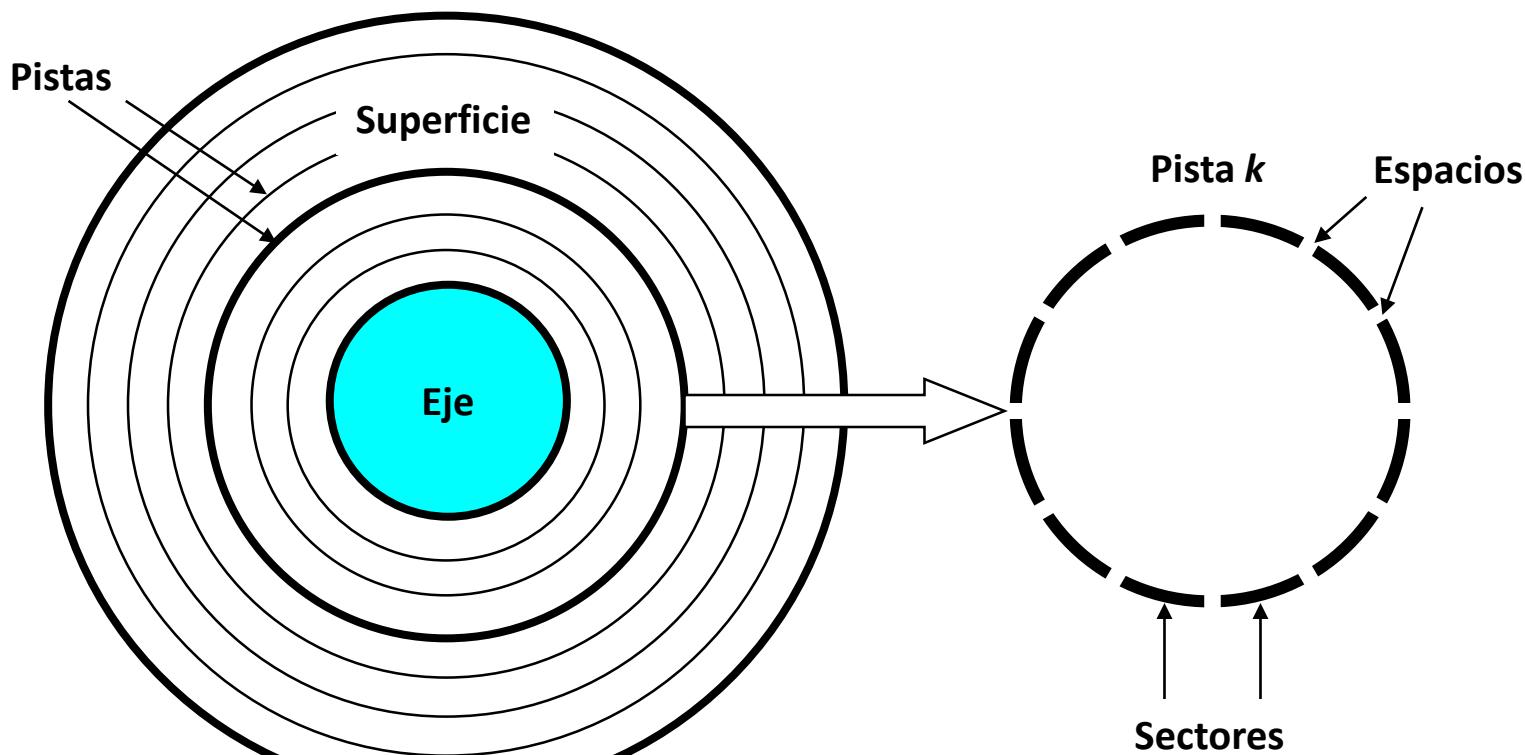


Imagen cortesía de Seagate Technology

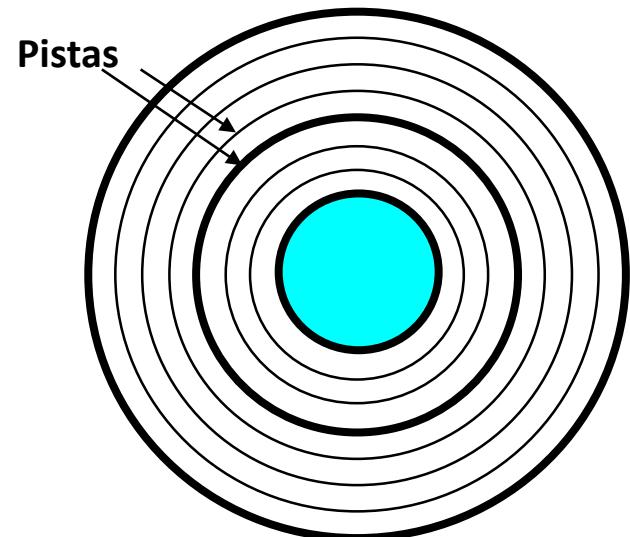
Geometría de un disco

- Los discos consisten en **platos** con dos **superficies (caras)**[†]
- Cada cara consiste en anillos concéntricos llamados **pistas**
- Cada pista consiste en **sectores** separados por **espacios**[†]



Capacidad de un disco

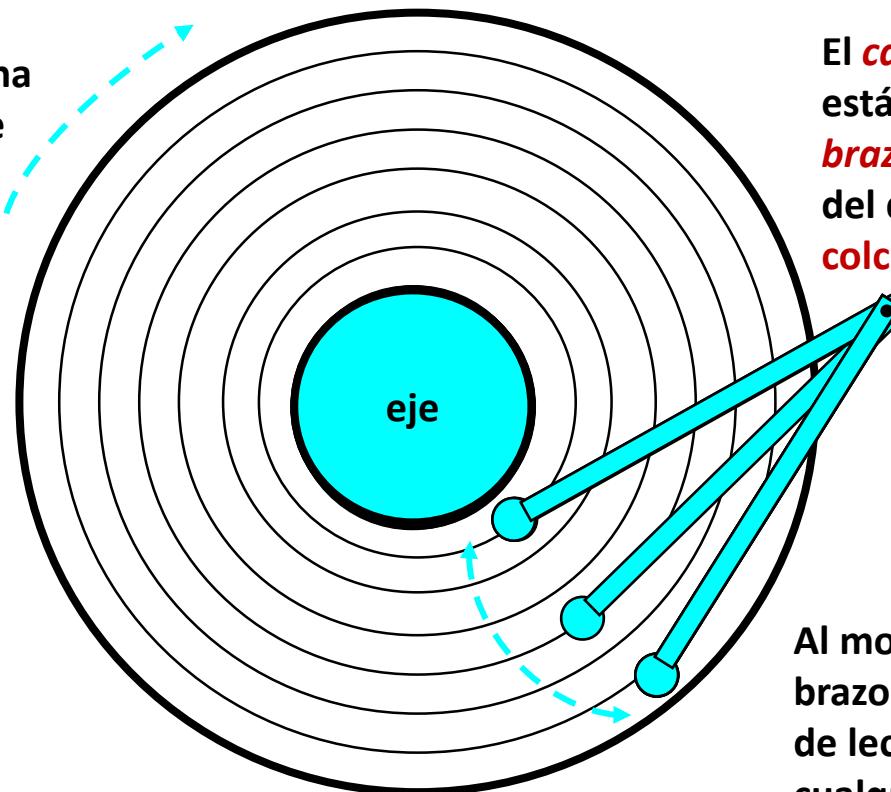
- **Capacidad:** máximo número de bits que se pueden almacenar
 - Los proveedores expresan la capacidad en unidades de gigabytes (GB) o terabytes (TB), donde $1\text{ GB} = 10^9\text{ Bytes}$ y $1\text{ TB} = 10^{12}\text{ Bytes}$
- **La capacidad queda determinada por estos factores tecnológicos:**
 - **Densidad de grabación** (bits/pulgada[†]): nº de bits que se pueden comprimir en un tramo de pista de 1 pulgada
 - **Densidad de pistas (radial)** (pistas/pulgada): nº de pistas que se pueden comprimir en un tramo radial de 1 pulgada
 - **Densidad superficial** (bits/pulg²): producto de ambas densidades (grabación y radial)



[†] bits/in, bits per inch 68

Funcionamiento de un disco (Visto en un solo plato)

La superficie del disco gira con una velocidad fija de rotación

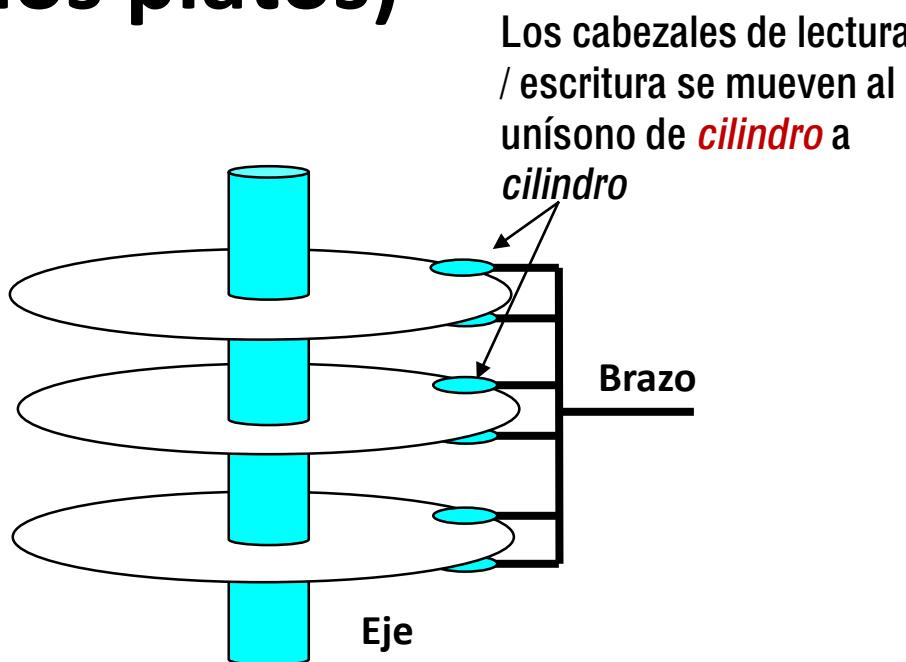


El **cabezal** de lectura / escritura está unido al extremo del **brazo** y vuela por la superficie del disco sobre un delgado **colchón de aire**

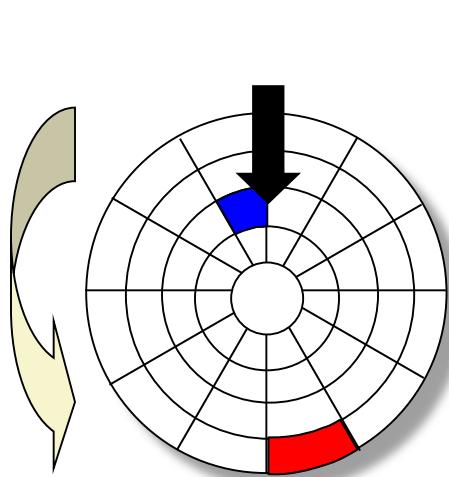
Al moverse radialmente, el brazo puede colocar el cabezal de lectura / escritura sobre cualquier pista.

Funcionamiento de un disco

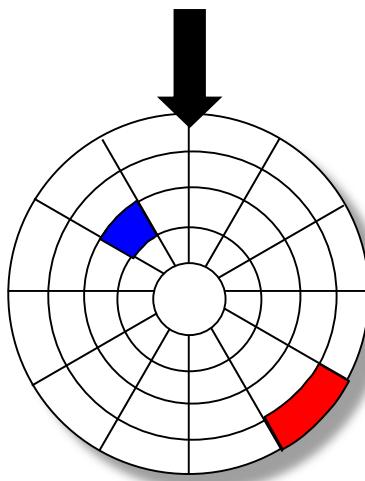
(Visto en varios platos)



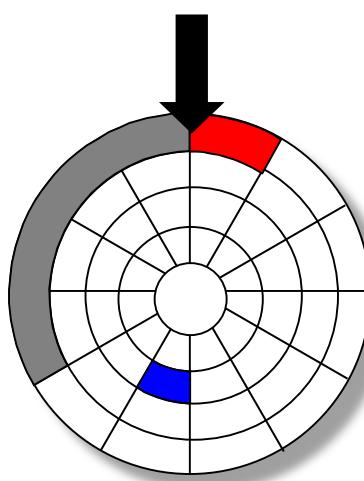
Acceso a disco: componentes del tiempo de servicio



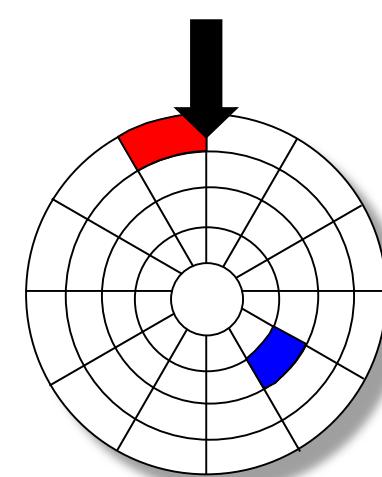
Tras leer AZUL



Buscar ROJO



Latencia rotacional



Tras leer ROJO

↑
Transferencia
de datos

↑
Búsqueda†

↑
Latencia
rotacional
↑
Transferencia
de datos

Tiempo de acceso a disco

■ Tiempo promedio acceso algún sector determinado, aprox:

$$\text{■ } T_{\text{acceso}} = T_{\text{prom b\u00fasqueda}} + T_{\text{prom rotaci\u00f3n}} + T_{\text{prom transferencia}}$$

■ Tiempo de b\u00fasqueda ($T_{\text{prom b\u00fasqueda}}$)

- Tiempo para colocar cabezales sobre el cilindro que contiene el sector
- Valores t\u00edpicos $T_{\text{prom b\u00fasqueda}} = 3-9 \text{ ms}$

■ Latencia rotacional ($T_{\text{prom rotaci\u00f3n}}$)

- Tiempo esperando a que pase bajo cabezales el primer bit del sector
- $T_{\text{prom rotaci\u00f3n}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ s}/1 \text{ min}$
- Velocidad rotacional t\u00edpica = 7,200 RPMs ($\Rightarrow 4.17 \text{ ms}$)

■ Tiempo de transferencia ($T_{\text{prom transferencia}}$)

- Tiempo para leer los bits del sector
- $T_{\text{prom transferencia}} = 1/\text{RPMs} \times 1/(\# \text{ sectores/pista prom}) \times 60 \text{ s}/1 \text{ min}$

Tiempo para una rotaci\u00f3n (minutos) fracci\u00f3n de rotaci\u00f3n a leer

Ejemplo de tiempo de acceso a disco

■ Dados:

- Velocidad rotacional = 7,200 RPM
- Tiempo búsqueda promedio = **9 ms**
- # sectores/pista promedio = 400

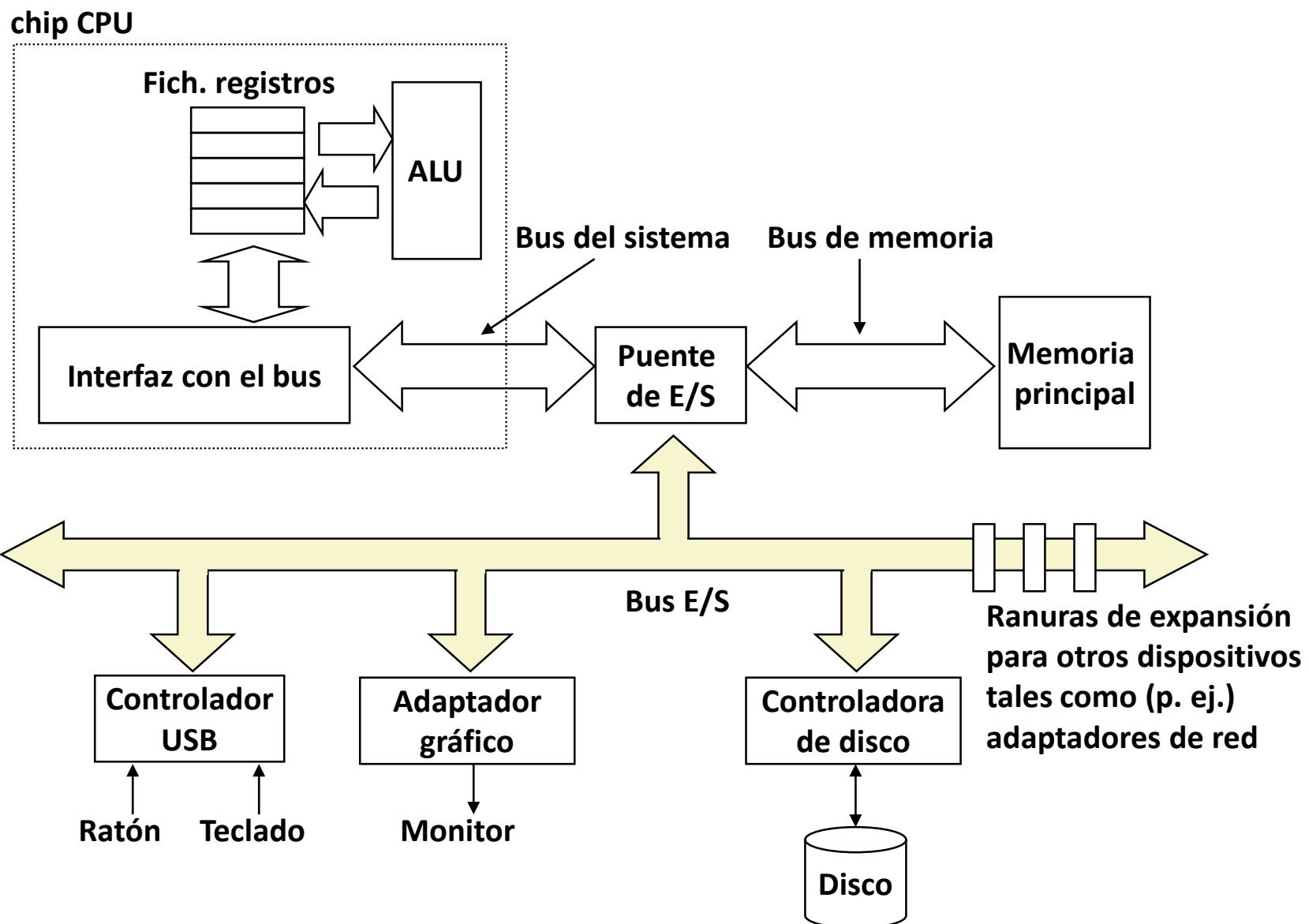
■ Calcular:

- $T_{\text{prom rotación}} = 1/2 \times (60 \text{ s}/7200 \text{ RPM}) \times 1000 \text{ ms/s} = 4.17 \text{ ms}$
- $T_{\text{prom transferencia}} = 60/7200 \times 1/400 \times 1000 \text{ ms/s} = 0.02 \text{ ms}$
- $T_{\text{acceso}} = 9 \text{ ms} + 4.17 \text{ ms} + 0.02 \text{ ms} = 13.19 \text{ ms}$

■ Puntos importantes:

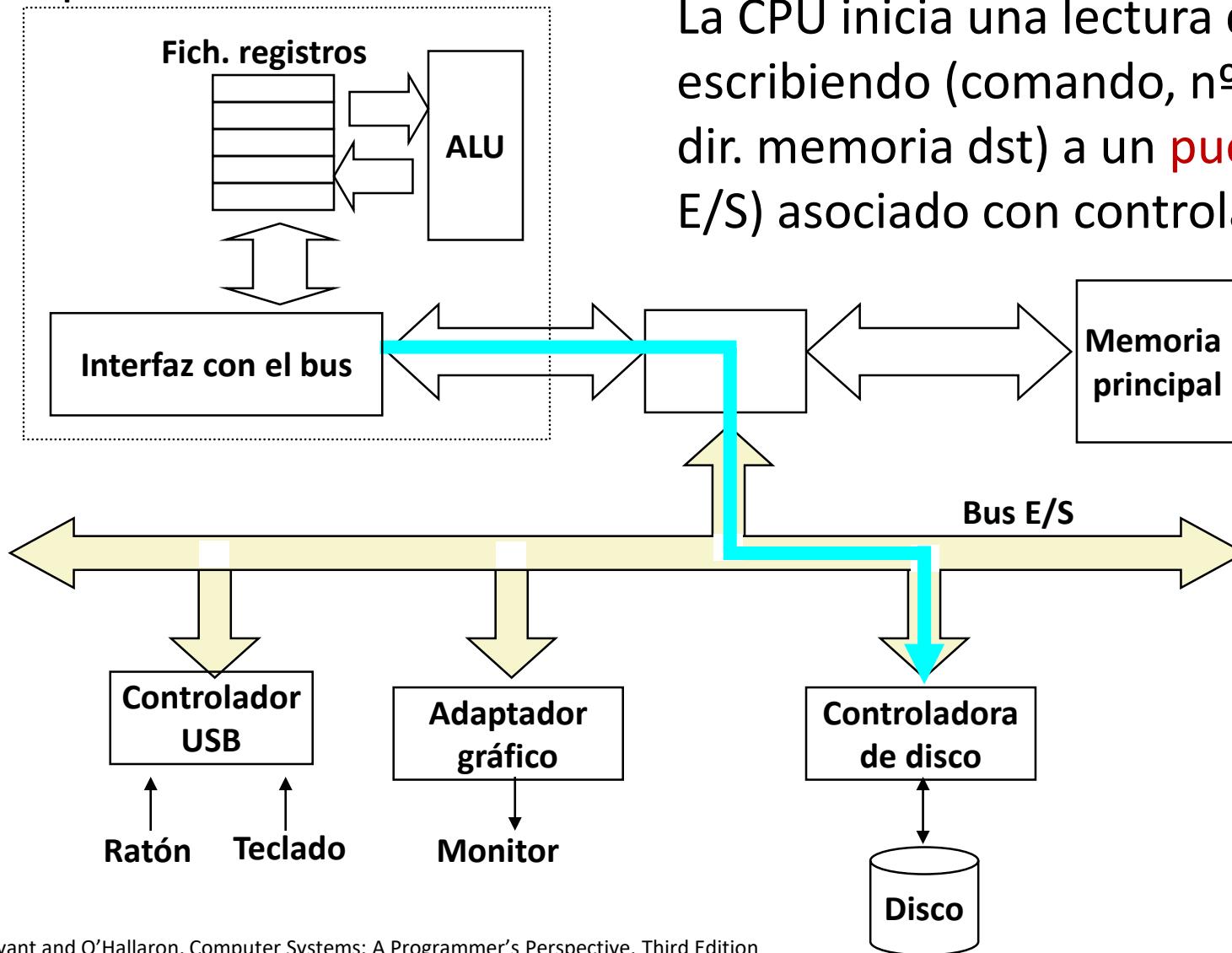
- Tiempo acceso dominado por tiempo búsqueda y latencia rotacional
- El primer bit en un sector es el más caro (lento), el resto sale gratis.
- *Tiempo acceso SRAM aprox. 4ns/palabra 64b, DRAM aprox. 60ns*
 - *Disco es aprox. 40.000 veces más lento que SRAM,*
 - *2.500 veces más lento que DRAM[†]*

Bus de E/S



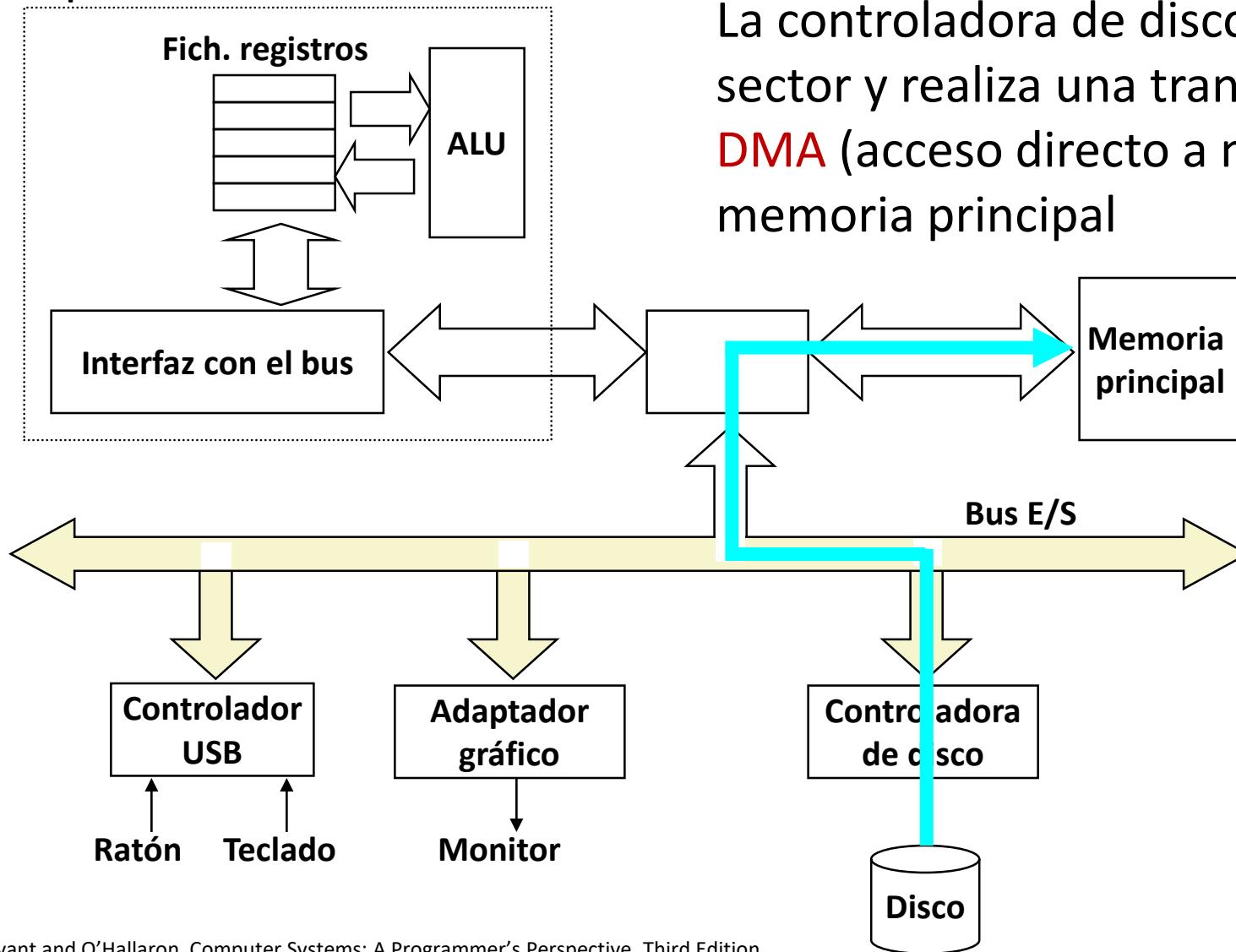
Lectura de un sector de disco (1)

chip CPU



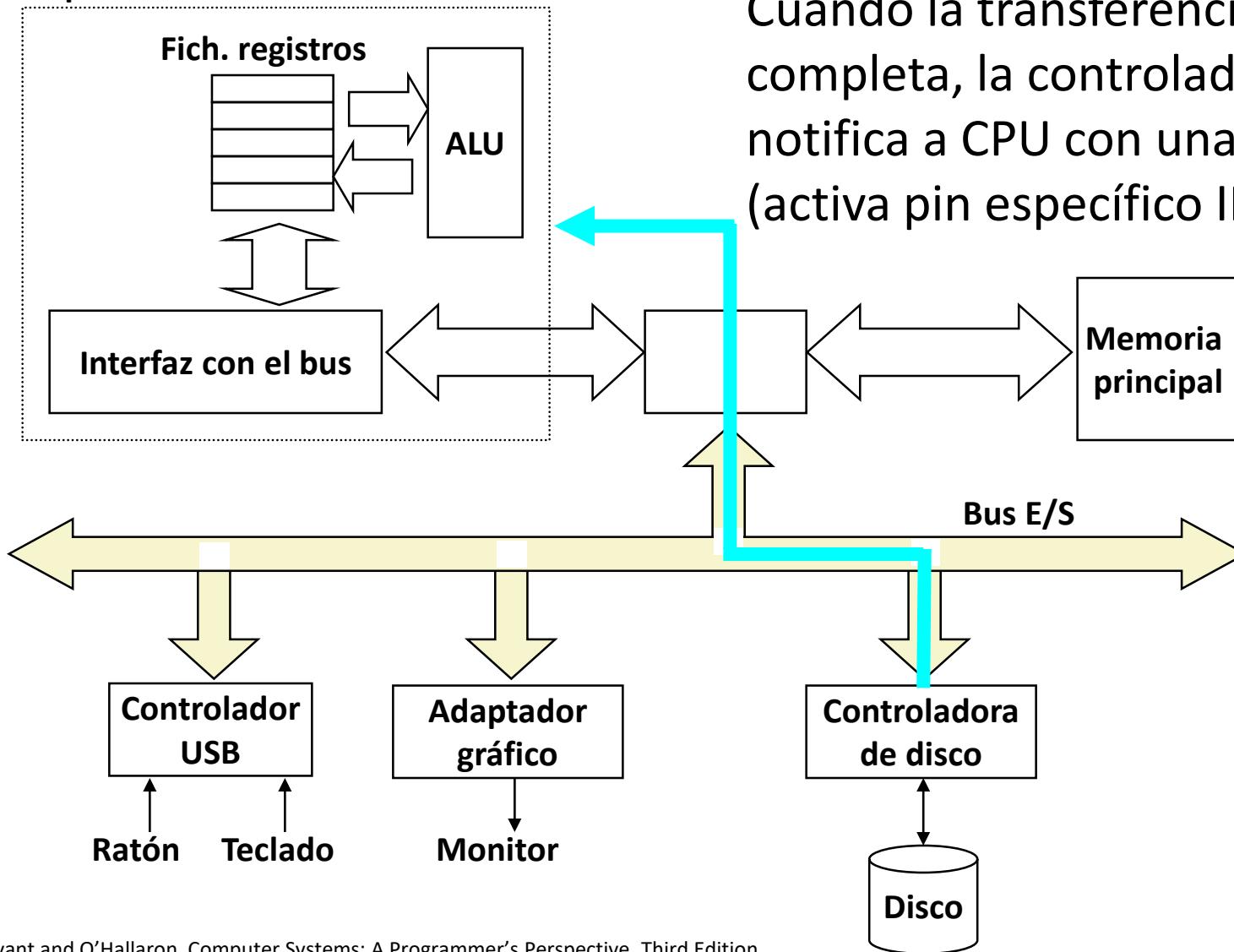
Lectura de un sector de disco (2)

chip CPU



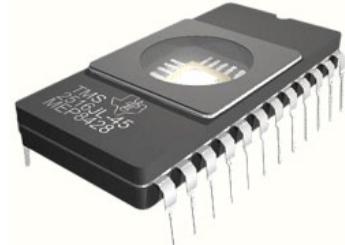
Lectura de un sector de disco (3)

chip CPU

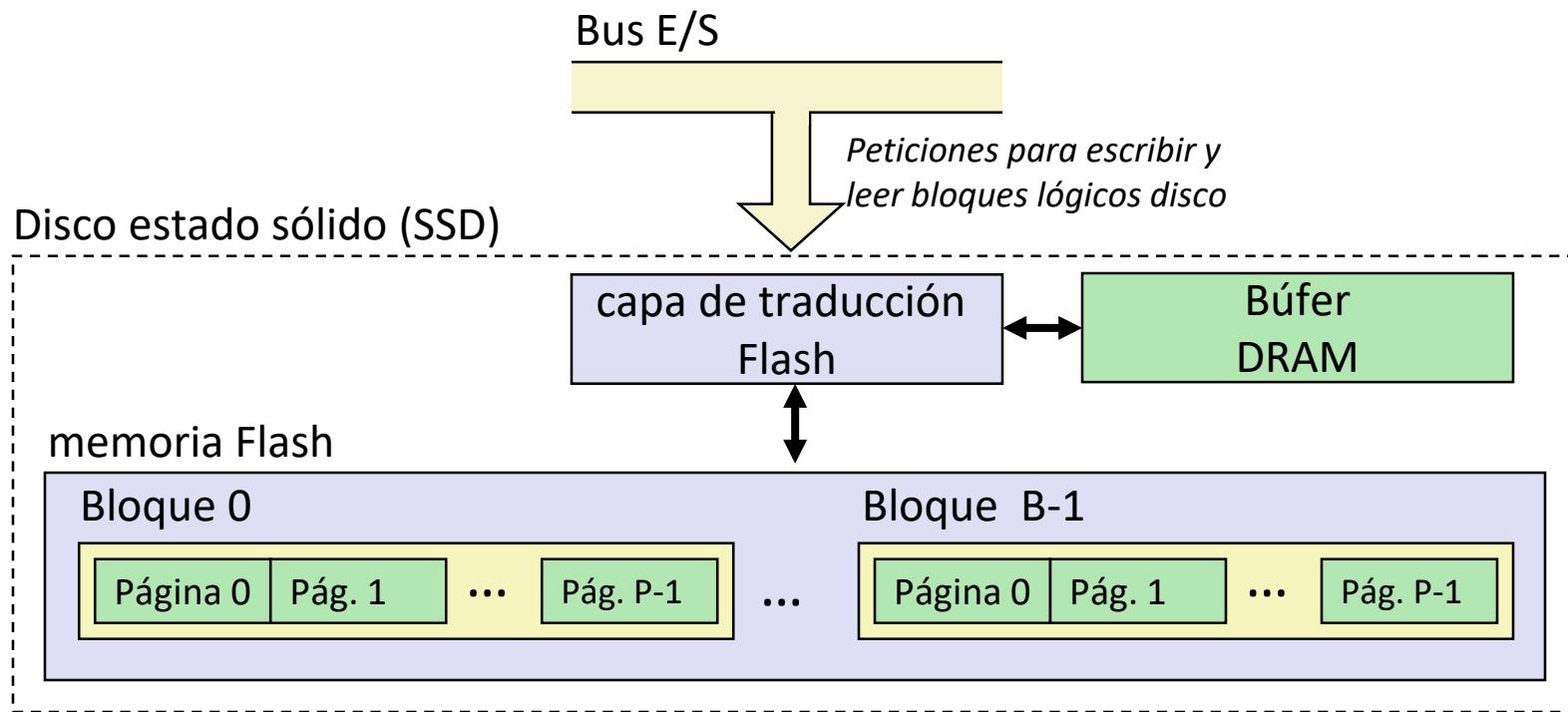


Cuando la transferencia DMA se completa, la controladora de disco notifica a CPU con una *interrupción* (activa pin específico IRQ de la CPU)

Memorias no volátiles

- **SRAM y DRAM son memorias volátiles**
 - Pierden información si se apagan
- **Las memorias no volátiles retienen valores incluso si se apagan**
 - Memoria de sólo lectura (**ROM**):
 - programada durante su fabricación
 - **PROM** progr. usr. irreversible, **EPROM** borrable (luz UV) → 
 - **EEPROM**: PROM borrable eléctricamente
 - Memorias **Flash**: EEPROMs con capacidad borrado parcial (por bloques)
 - se desgastan tras aprox. 100.000 borrados
 - 3D XPoint[†] (Intel Optane) & NVMs emergentes
 - nuevos materiales
- **Aplicaciones de las memorias no volátiles**
 - Programas firmware almacenados en una ROM (BIOS, controladoras de disco, tarjetas de red, aceleradores gráficos, subsistemas seguridad...)
 - Discos de estado sólido (reemplazando discos giratorios)
 - Caches de disco

Discos de estado sólido (SSDs)



- **Páginas: de 4KB a 512KB, Bloques: de 32 a 128 páginas**
- **Datos escritos/leídos en unidades de páginas**
- **Una página se puede escribir sólo tras borrar su bloque**
- **Un bloque se desgasta tras unas 100.000 escrituras**

Prestaciones características SSD

■ Benchmark[#] de un Samsung 970 EVO Plus

<https://ssd.userbenchmark.com/SpeedTest/711305/Samsung-SSD-970-EVO-Plus-250GB>

Rendimiento lectura secuencial 2.126 MB/s Rendmto. escritura sec. 1.880 MB/s

Rendimiento[†] lectura aleatoria 140 MB/s Rendmto. escritura aleat. 59 MB/s

■ Acceso secuencial mucho más rápido que aleatorio

- Tema omnipresente en jerarquías de memoria

■ Escrituras aleatorias son especialmente lentas

- Borrar un bloque lleva mucho tiempo (~1ms)
- Modificar una página de un bloque requiere que todas las otras se copien a un nuevo bloque
- La capa de traducción Flash permite acumular una serie de pequeñas escrituras antes de realizar una escritura de bloque

SSD frente a Discos giratorios

■ Ventajas

- No partes móviles → más rápidos, menor consumo, más resistentes

■ Desventajas

- Eventual desgaste
 - mitigado por “lógica nivelado desgaste” en capa traducción flash
 - p.ej. Samsung 970 EVO Plus garantiza que se pueda escribir 600x la capacidad del disco antes de desgastarse[†]
 - controladora migra datos para repartir/minimizar nivel desgaste
- En 2019, aprox. 4x más caro por byte (que giratorios)
 - y seguirá cayendo ese coste relativo

■ Aplicaciones

- reproductores MP3, smartphones, portátiles
- cada vez más común en servidores y PC sobremesa

[†] $600x << 100.000$ borrados bloque

ver folleto en <https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/970evoplus/>

disponible en 0.25, 0.5, 1, 2TB con TBW = 150, 300, 600, 1200 TB **81.**

Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Resumen

- La brecha de velocidad entre CPU, memoria y almacenamiento masivo continúa ampliándose.
- Los programas bien escritos exhiben una propiedad llamada localidad.
- Las jerarquías de memoria, basadas en cacheado, cierran la brecha al explotar la localidad.
- El progreso en memoria flash está sobre pasando a todas las demás tecnologías de memoria y almacenamiento (DRAM, SRAM, disco magnético)
 - Capaz de apilar celdas en tres dimensiones

Diapositivas suplementarias

Tendencias en almacenamiento

SRAM

Métrica	1985	1990	1995	2000	2005	2010	2015	2015:1985
\$/MB	2,900	320	256	100	75	60	25	116
acceso (ns)	150	35	15	3	2	1.5	1.3	115

DRAM

Métrica	1985	1990	1995	2000	2005	2010	2015	2015:1985
\$/MB	880	100	30	1	0.1	0.06	0.02	44,000
acceso (ns)	200	100	70	60	50	40	20	10
tam. típico (MB)	0.256	4	16	64	2,000	8,000	16.000	62,500

Disco

Métrica	1985	1990	1995	2000	2005	2010	2015	2015:1985
\$/GB	100,000	8,000	300	10	5	0.3	0.03	3,333,333
búsqueda (ms)	75	28	10	8	5	3	3	25
tam. típico (GB)	0.01	0.16	1	20	160	1,500	3,000	300,000

Frecuencia reloj CPU

Punto inflexión en historia computadores
al chocar diseñadores con “Muro Potencia”

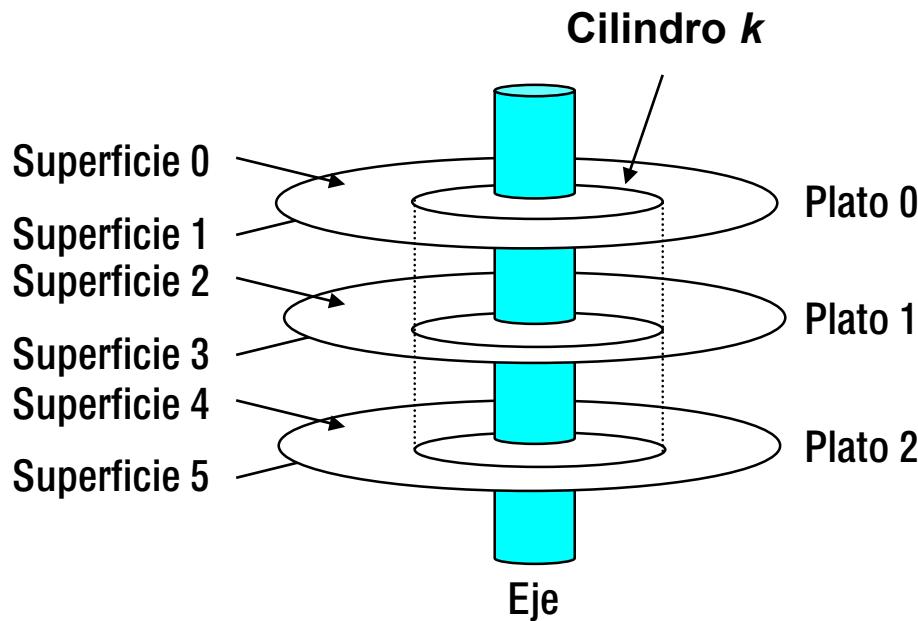


	1985	1990	1995	2003	2005	2010	2015	2015:1985
CPU	80286	80386	Pentium	P-4	Core 2	Core i7(n)	Core i7(h)	
Frecuencia reloj (MHz)	6	20	150	3,300	2,000	2,500	3,000	500
Tiempo ciclo (ns)	166	50	6	0.30	0.50	0.4	0.33	500
Cores	1	1	1	1	2	4	4	4
Tiempo efectivo ciclo (ns)	166	50	6	0.30	0.25	0.10	0.08	2,075

(n) procesador Nehalem
(h) procesador Haswell

Geometría de un disco (Vista en varios platos)

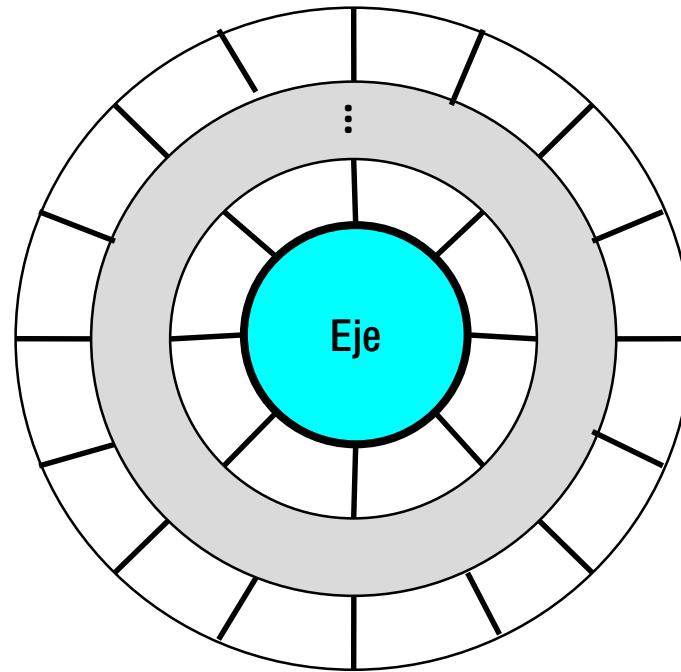
- Pistas alineadas forman un cilindro



Zonas de grabación

■ Los discos modernos
particionan las pistas en
subconjuntos disjuntos
llamados **zonas de grabación**

- Las pistas de una zona tienen el mismo número de sectores, determinado por la circunferencia de la pista más interna.
- Cada zona tiene una cantidad diferente de sectores/pista, las zonas externas tienen más que las zonas internas.
- Así que usamos el promedio de sectores/pista cuando calculamos la capacidad.



Cálculo de la capacidad de un disco

**Capacidad = (# bytes/sector) x (# sectores/pista prom.) x
(# pistas/superficie) x (# superficies/plato) x
(# platos/disco)**

Ejemplo:

- 512 bytes/sector
- 300 sectores/pista (en promedio)
- 20,000 pistas/superficie
- 2 superficies/plato
- 5 platos/disco

$$\begin{aligned}\text{Capacidad} &= 512 \times 300 \times 20000 \times 2 \times 5 \\ &= 30,720,000,000 \\ &= 30.72 \text{ GB}\end{aligned}$$

Bloques (de disco) Lógicos[†]

- Los discos “modernos” presentan una visión más simple y abstracta de la compleja geometría de sectores en disco:
 - El conjunto de sectores disponibles se modela como una secuencia $(0, 1, 2, \dots)$ de **bloques lógicos** de tamaño b
- Correspondencia entre **bloques lógicos** y **sectores reales (físicos)**
 - Mantenida en hardware/firmware (controladora de disco)
 - Convierte peticiones de bloques lógicos en tripletes (superficie, pista, sector).
- Permite al controlador reservar cilindros de repuesto para cada zona.
 - Contribuye a la diferencia entre "capacidad formateada" y "capacidad máxima"

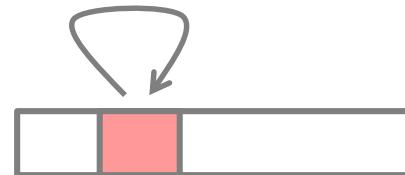
TEMA 6.2

Memoria II: Cache

- **Organización y Funcionamiento de la memoria cache**
- **Impacto de la cache en el rendimiento**
 - Modelo de evaluación
 - La montaña de memoria
- **Programación de código aprovechando la cache**

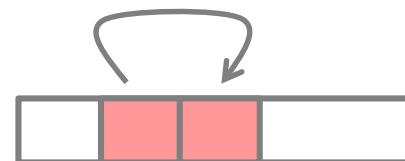
Localidad (Recordatorio)

- **Principio de localidad:** Los programas tienden a usar datos e instrucciones con direcciones iguales o cercanas a las que han usado recientemente



- **Localidad temporal:**

- Elementos referenciados recientemente probablemente serán referenciados de nuevo en un futuro próximo



- **Localidad espacial:**

- Elementos con direcciones cercanas tienden a ser referenciados muy juntos en el tiempo

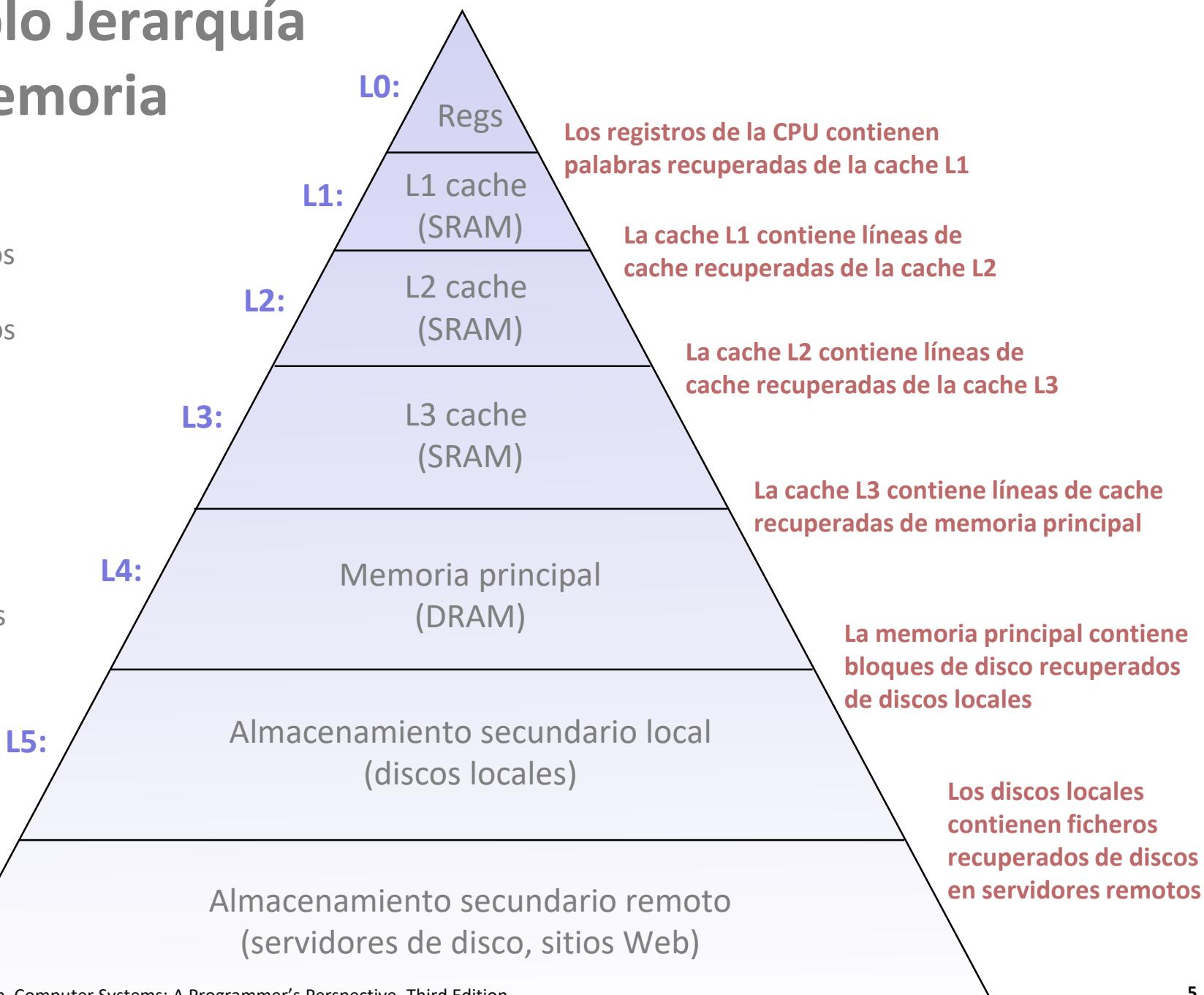
Ejemplo Jerarquía Memoria

dispositivos
almacnmtos.
más rápidos
más costosos
(por byte)
y + pequeños
(de menor
capacidad)

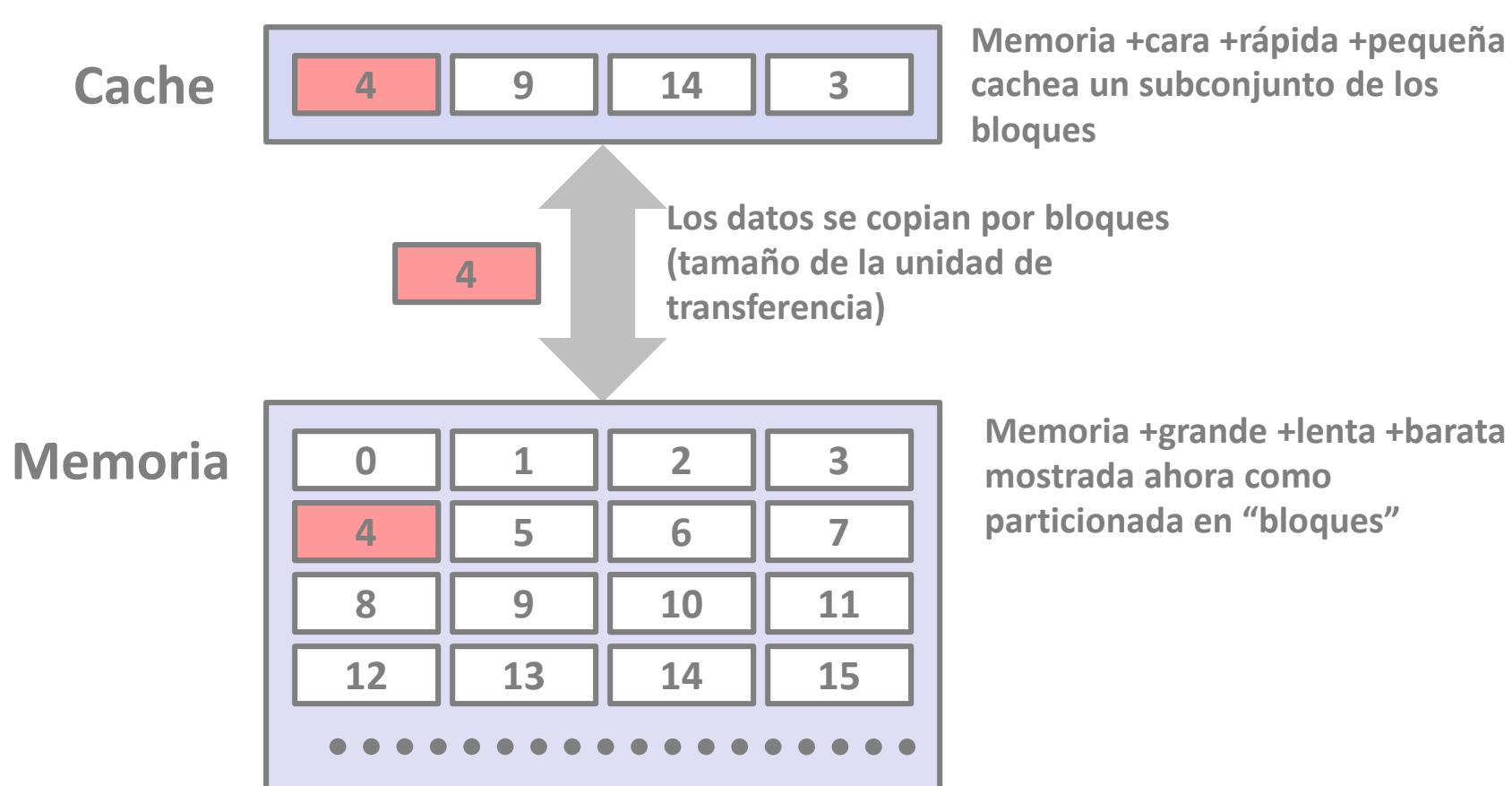
dispositivos
almacnmtos.
más lentos
más baratos
(por byte)
y + grandes
(de mayor
capacidad)

L6:

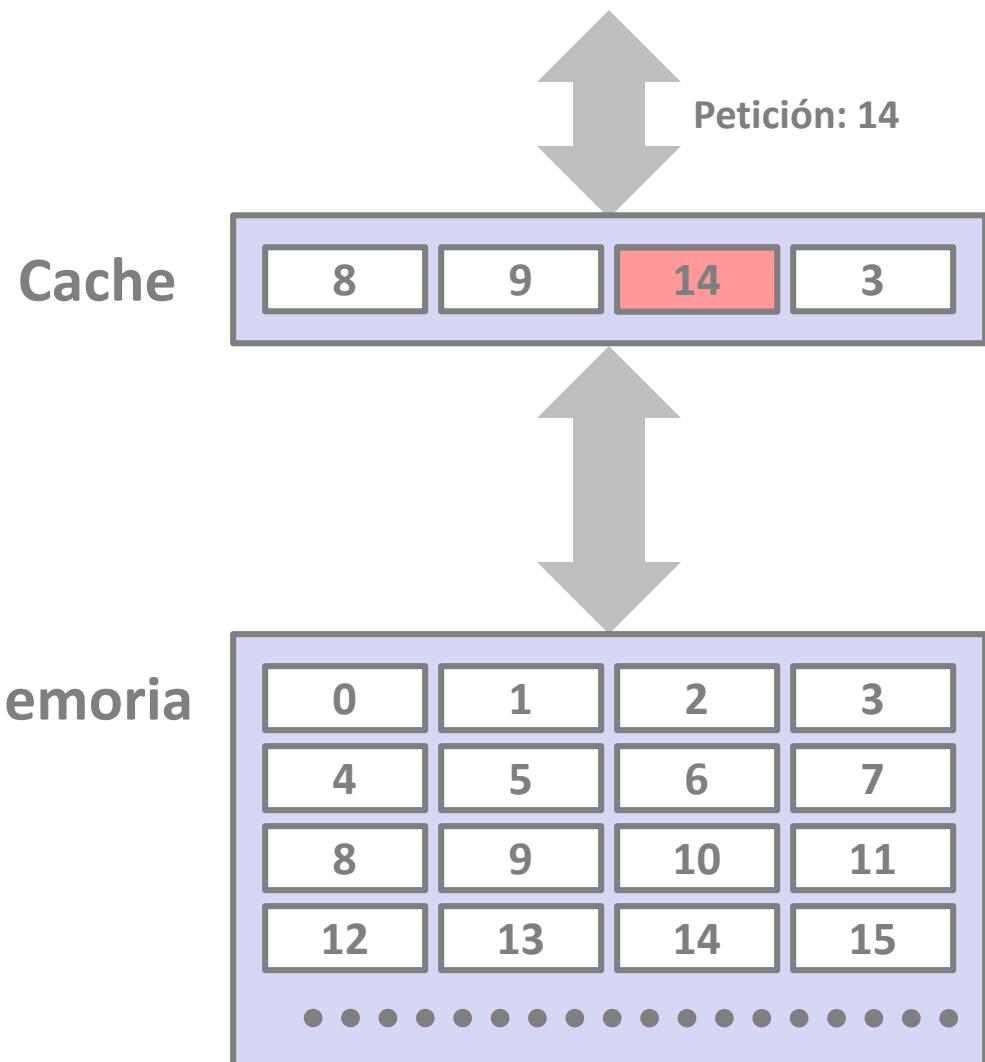
Almacenamiento secundario remoto
(servidores de disco, sitios Web)



Conceptos Generales de Cache



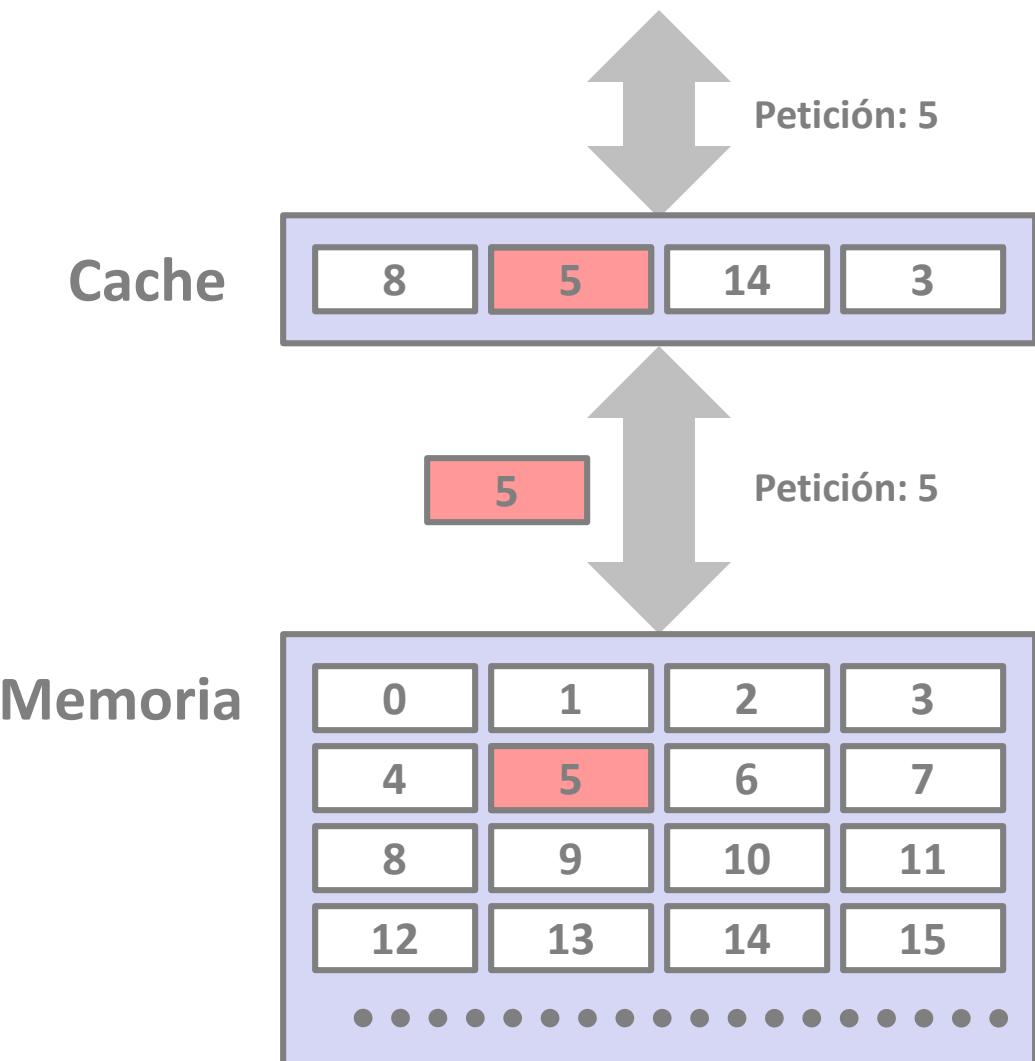
Conceptos Generales de Cache: Acierto[†]



*Se necesitan datos
del bloque b*

*El bloque b está en cache:
¡Acierto!*

Conceptos Generales de Cache: Fallo[†]



*Se necesitan datos
del bloque b*

*El bloque b no está en cache:
¡Fallo!*

*El bloque b se capta de
memoria*

*El bloque b se almacena
en cache*

- **Política de colocación[†]:** determina dónde va b
- **Política de reemplazo[†]:** determina qué bloque es desalojado[†] (víctima)

[†] cache miss

[re]placement policy,
evicted/victim block

Conceptos Generales de Cache:

3 Tipos de Fallo de Cache (**Recordatorio**)

■ Fallos en frío (obligados)

- Los fallos en frío ocurren porque la cache empieza vacía y esta es la primera referencia al bloque

■ Fallos por capacidad

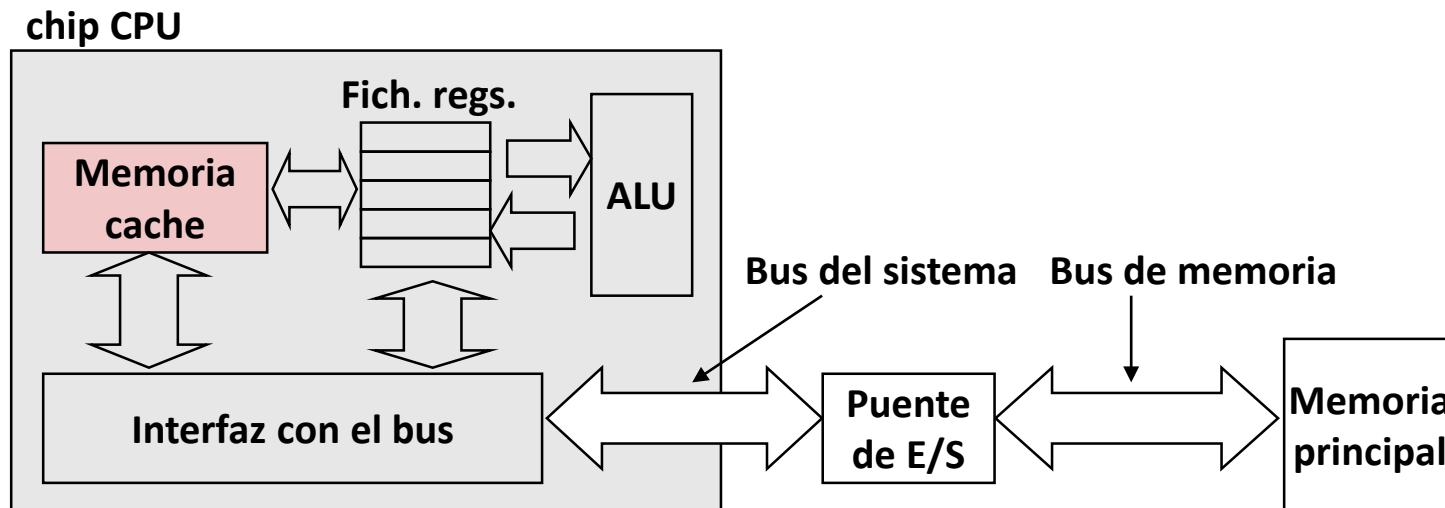
- Ocurren cuando el conjunto de bloques activos (**conjunto de trabajo**) es más grande que la cache

■ Fallos por conflicto

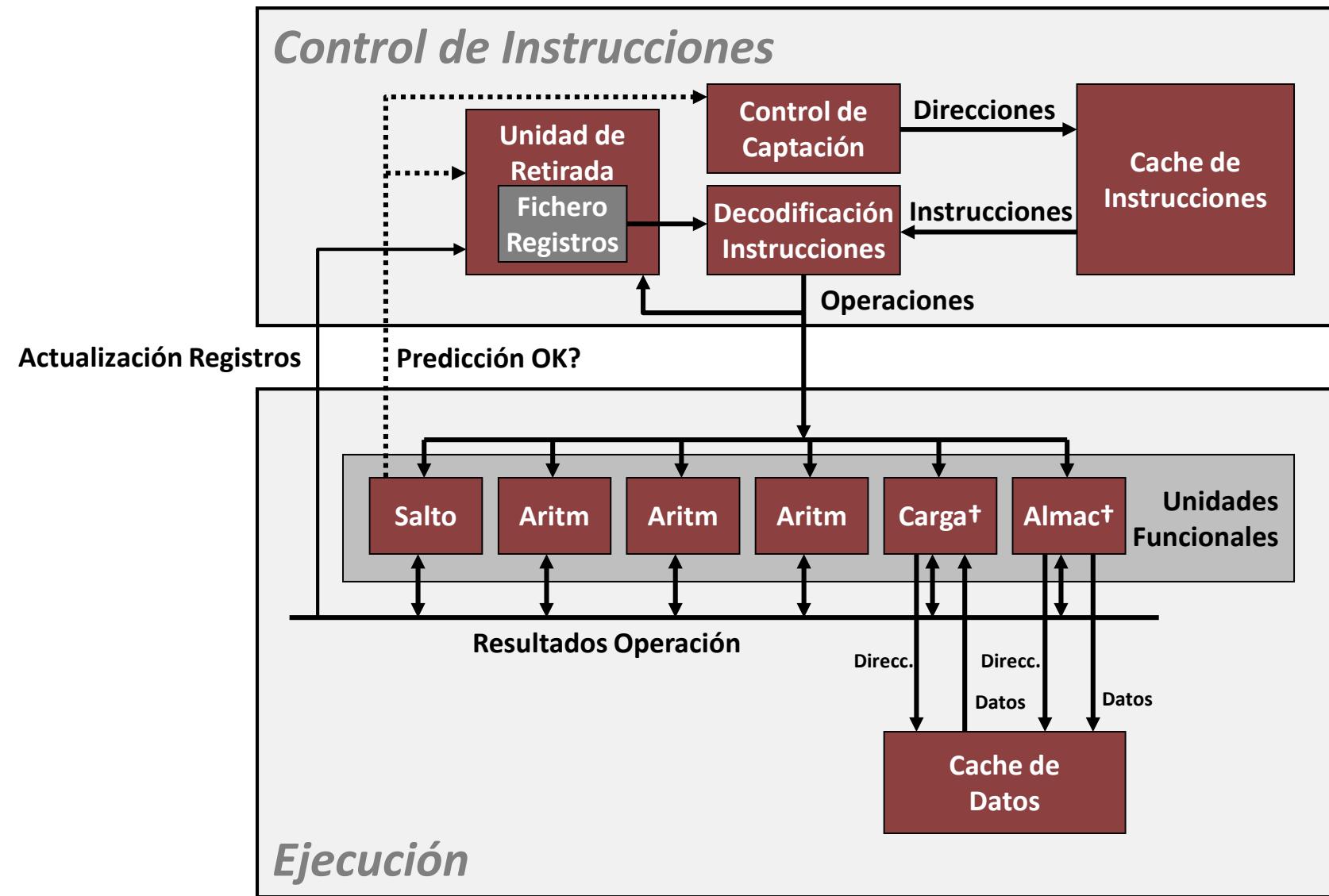
- Mayoría caches limitan que los bloques a nivel $k+1$ puedan ir a pequeño subconjunto (a veces unitario) de las posiciones de bloque a nivel k
 - P.ej. Bloque i a nivel $k+1$ debe ir a bloque $(i \bmod 4)$ a nivel k (corr. directa)
- Fallos por conflicto ocurren cuando cache nivel k suficientemente grande pero a varios datos les corresponde ir al mismo bloque a nivel k
 - P.ej. Referenciar bloques 0, 8, 0, 8, 0, 8, ... fallaría continuamente (ejemplo anterior con correspondencia directa)

Memorias Cache

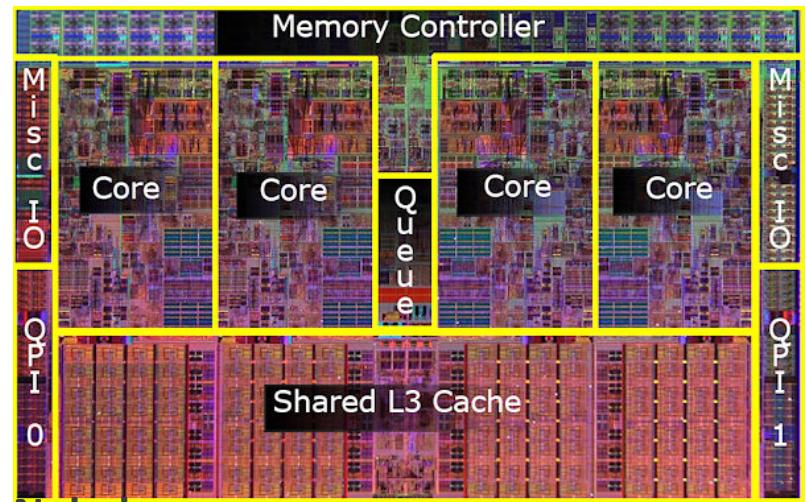
- Las **memorias cache** son memorias pequeñas y rápidas basadas en SRAM gestionadas automáticamente por hardware
 - Retiene bloques de memoria principal accedidos frecuentemente
- La CPU busca los datos primero en caché
- Estructura típica del sistema:



Diseño moderno de CPU

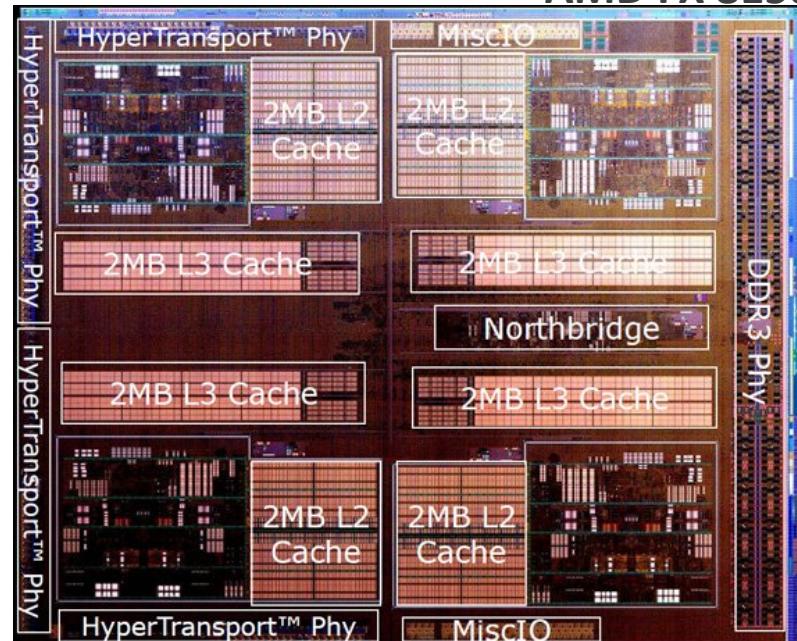


El aspecto que tiene realmente

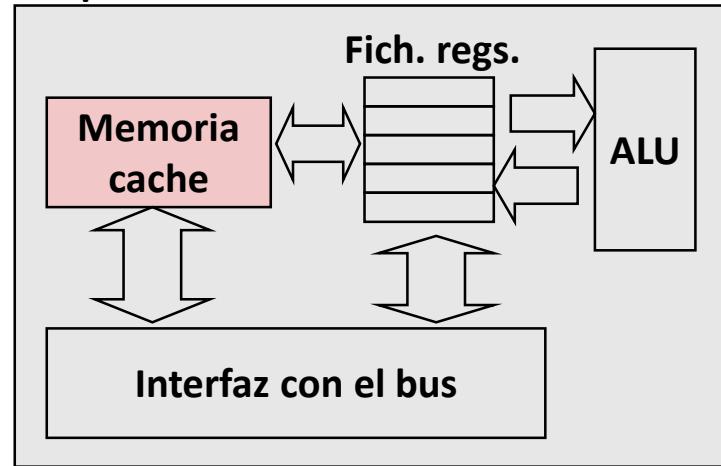


Nehalem

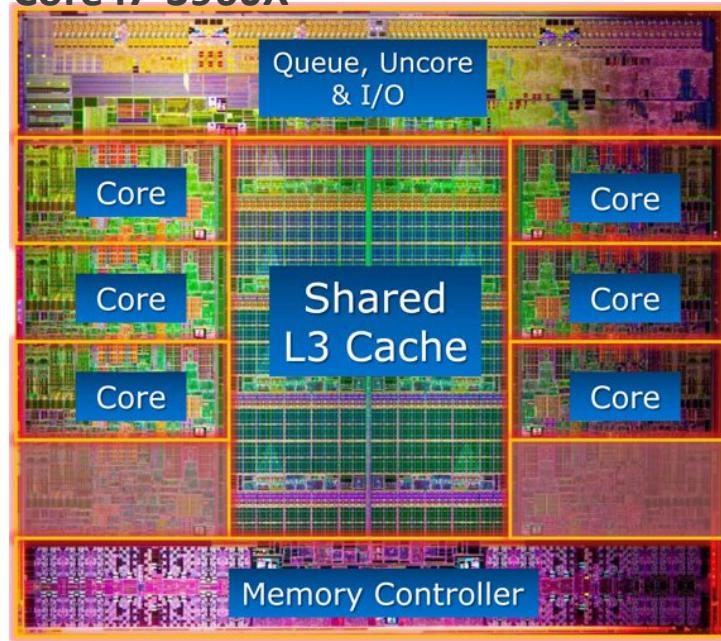
AMD FX 8150



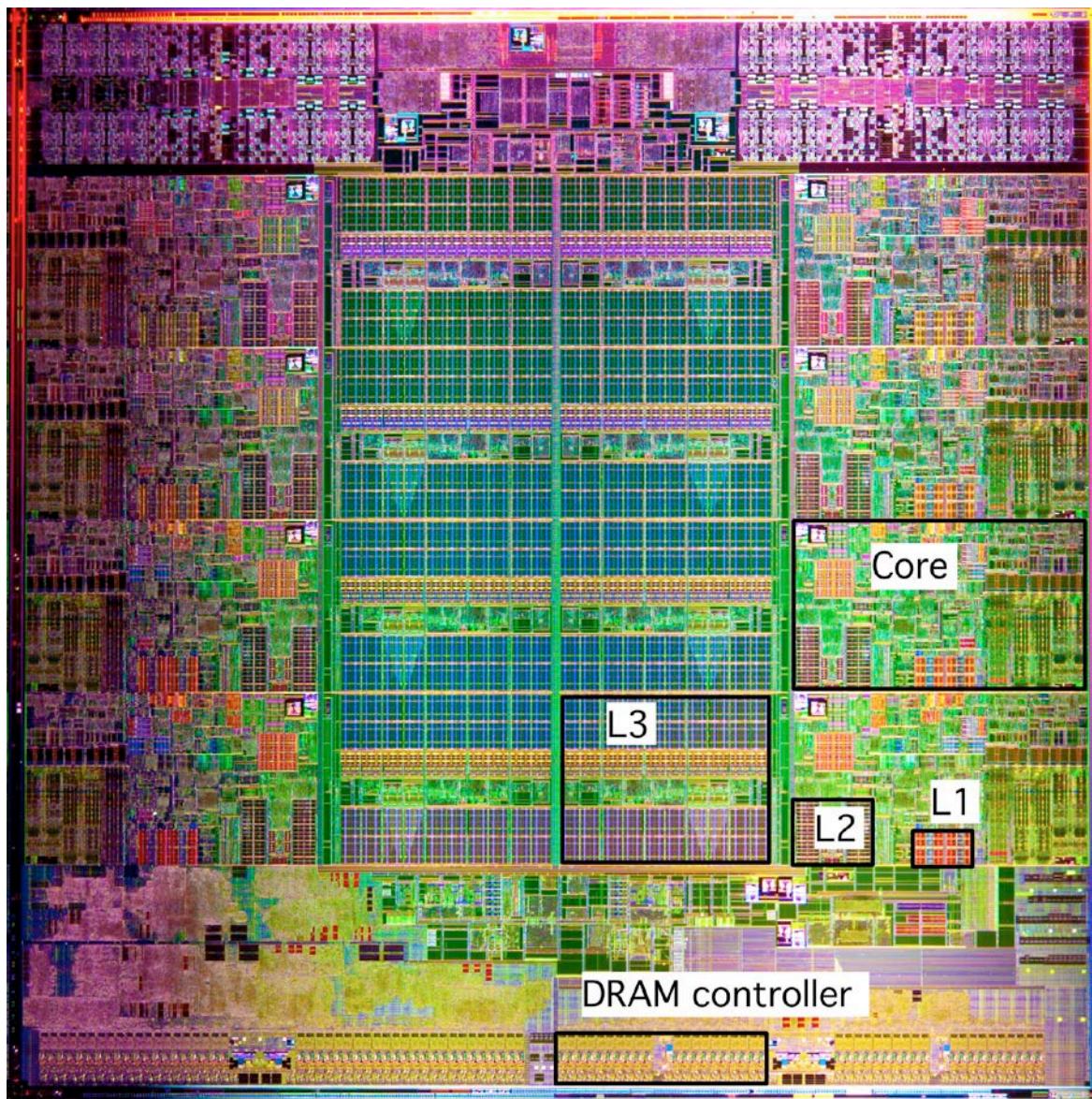
chip CPU



Core i7-3960X



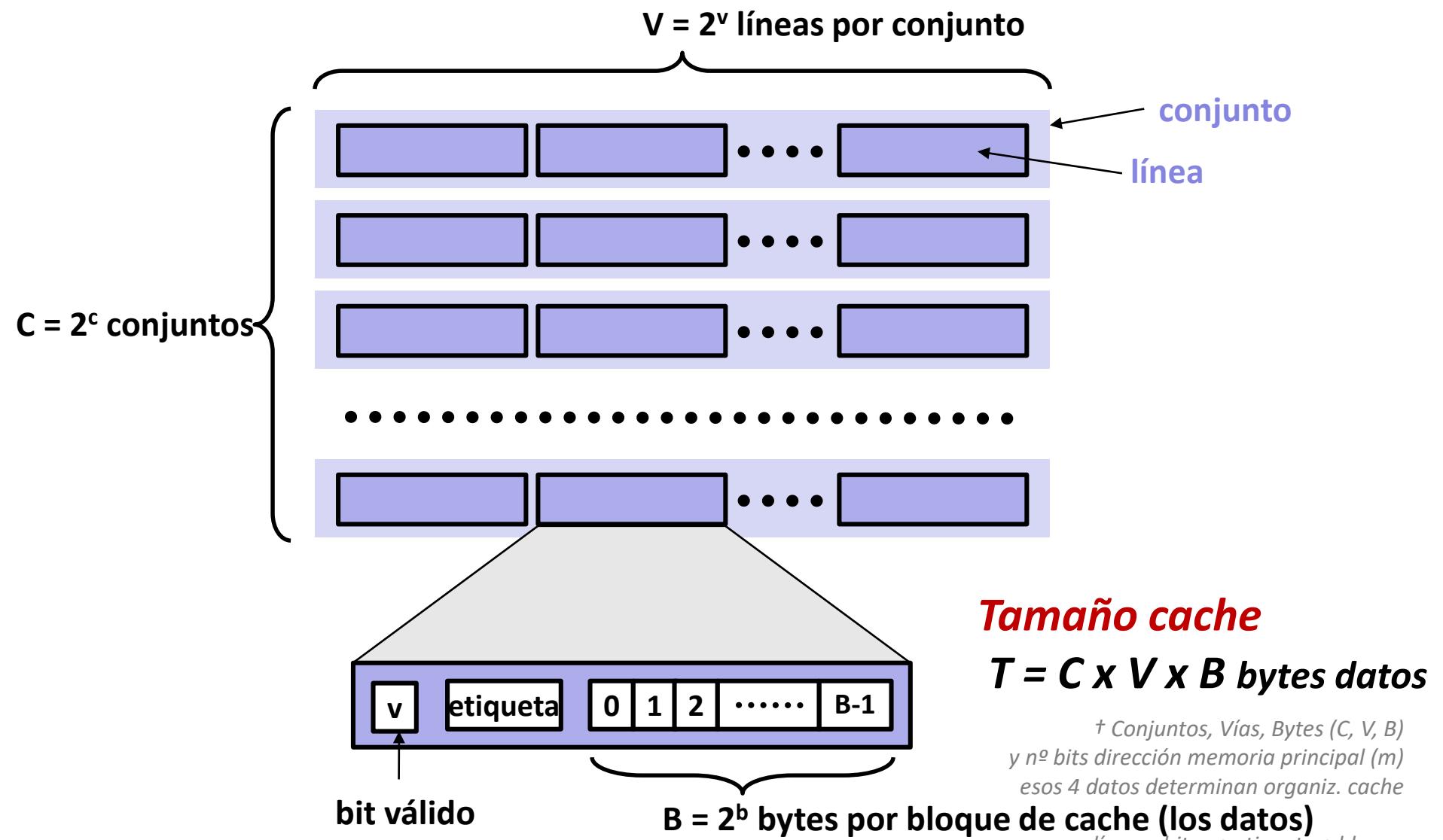
El aspecto que tiene realmente (Cont.)



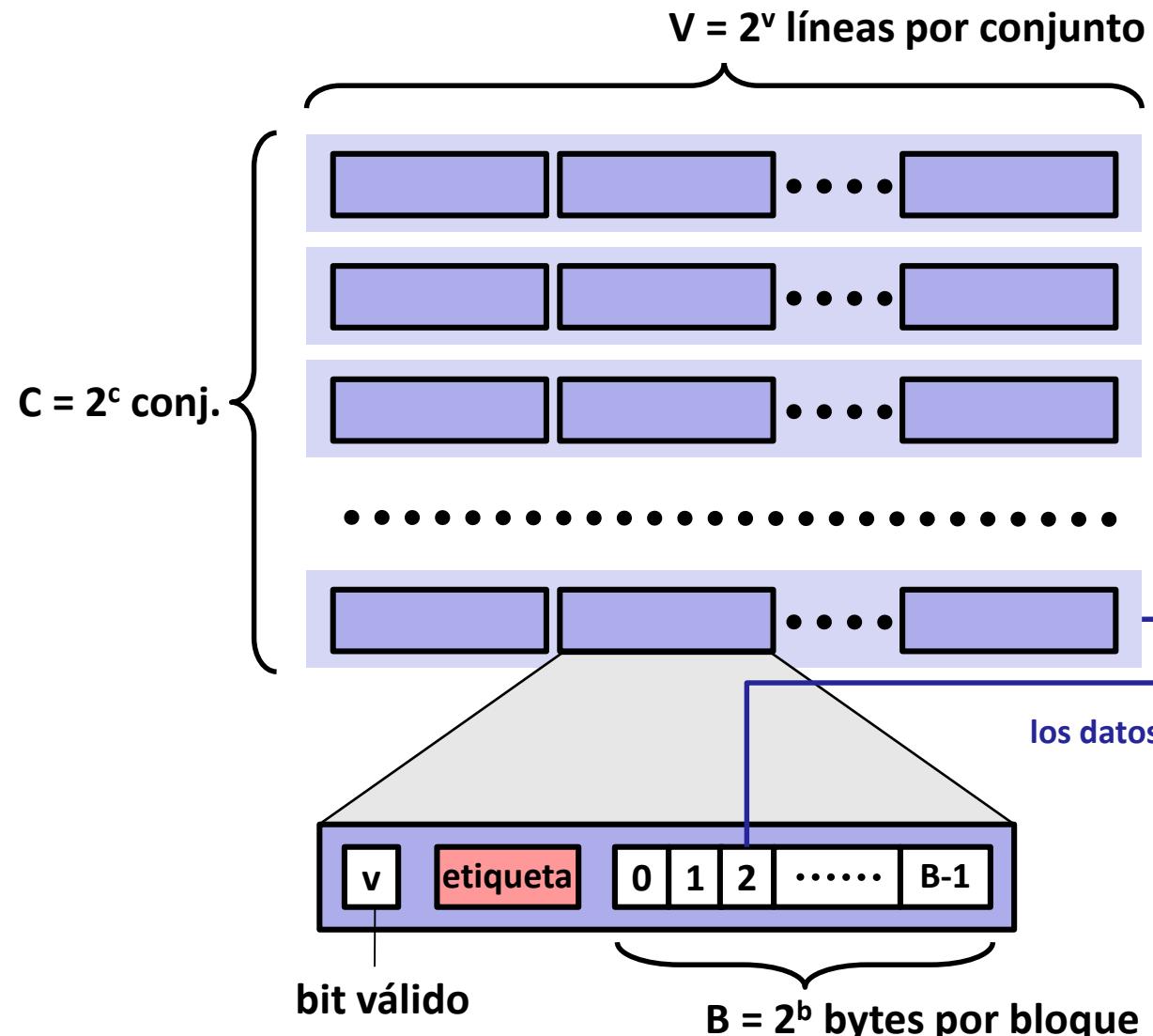
Dado+ Procesador
Intel Sandy Bridge

L1: 32KB Instrucciones + 32KB Datos
L2: 256KB
L3: 3–20MB

Organización General de Cache (C, V, B, m^+)

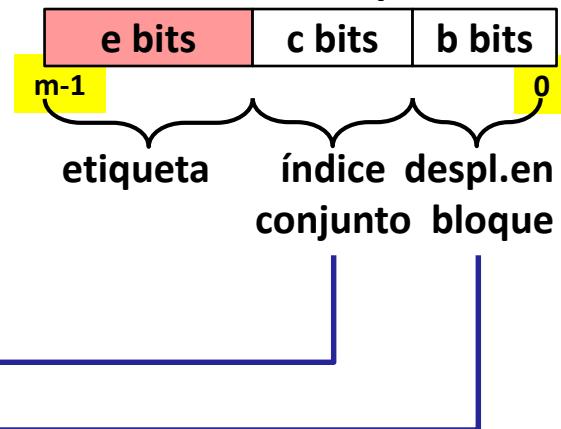


Lectura de Cache



- *Indexar conjunto*
 - *Si alguna línea tiene etiqueta coincidente y es válida: Acierto*
 - *Localizar datos a partir del desplazamiento*

Dirección inicio[†] palabra:



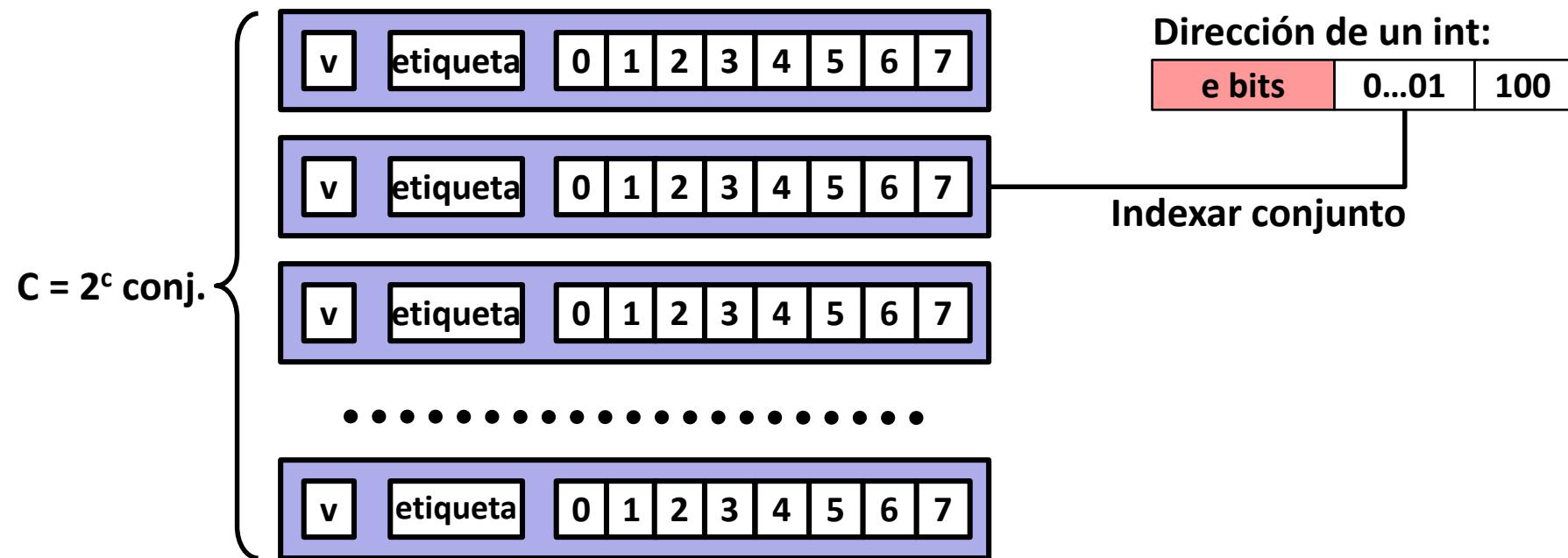
los datos empiezan en este desplazamiento

† Al haber dicho que 2^b son bytes, y al haber incluido los b bits en la dirección de memoria, se deduce que es memoria de bytes

Ej: Cache con Correspondencia Directa ($V = 1$)

Correspondencia directa: Una línea por conjunto

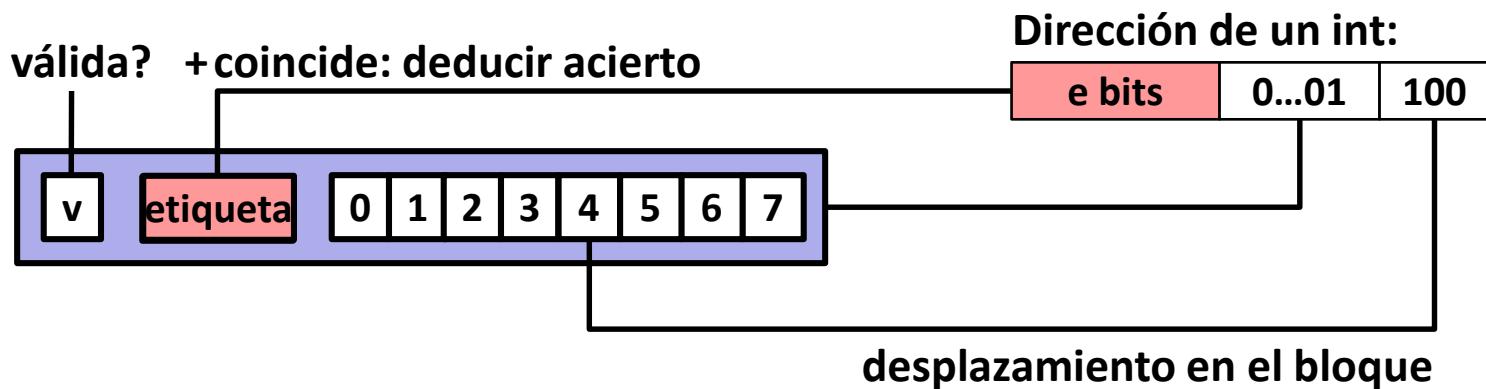
Suponer: tamaño bloque cache $B=8$ bytes



Ej: Cache con Correspondencia Directa ($V = 1$)

Correspondencia directa: Una línea por conjunto

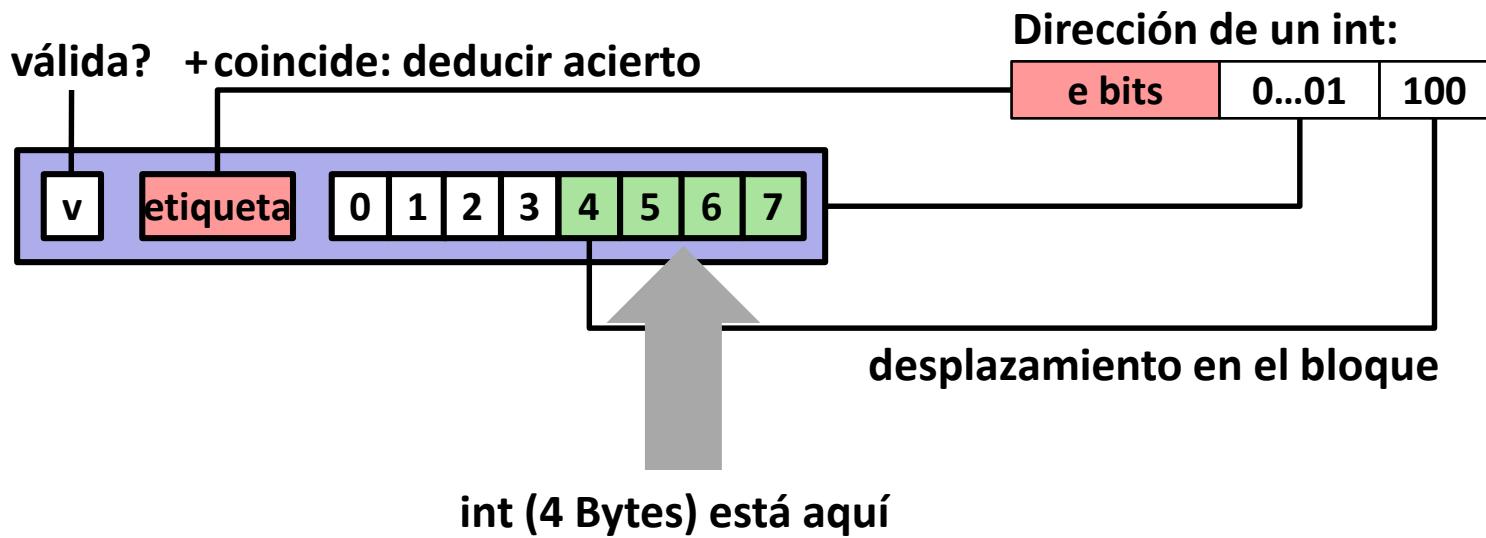
Suponer: tamaño bloque cache $B=8$ bytes



Ej: Cache con Correspondencia Directa ($V = 1$)

Correspondencia directa: Una línea por conjunto

Suponer: tamaño bloque cache $B=8$ bytes



Si etiqueta no coincide (= fallo): vieja línea desalojada y reemplazada
(nuevos datos y nueva etiqueta)

Simulación cache correspondencia directa

$e=1 \quad c=2 \quad b=1$

x	xx	x
---	----	---

direcciones 4 bits (espacio dirccnmt tam M=16 bytes)
 $C=4$ conjuntos, $V=1$ vía (bloq./conj.), $B=2$ bytes/bloque

Traza de direcciones (lecturas, un byte por lectura):

0	$[0\underline{000}_2]$,	fallo
1	$[0\underline{001}_2]$,	acuerdo
7	$[0\underline{111}_2]$,	fallo
8	$[1\underline{000}_2]$,	fallo
0	$[0\underline{000}_2]$	fallo

	v	Etiq.	Bloque
Conj. 0	1	0	M[0-1]
Conj. 1	0		
Conj. 2	0		
Conj. 3	1	0	M[6-7]

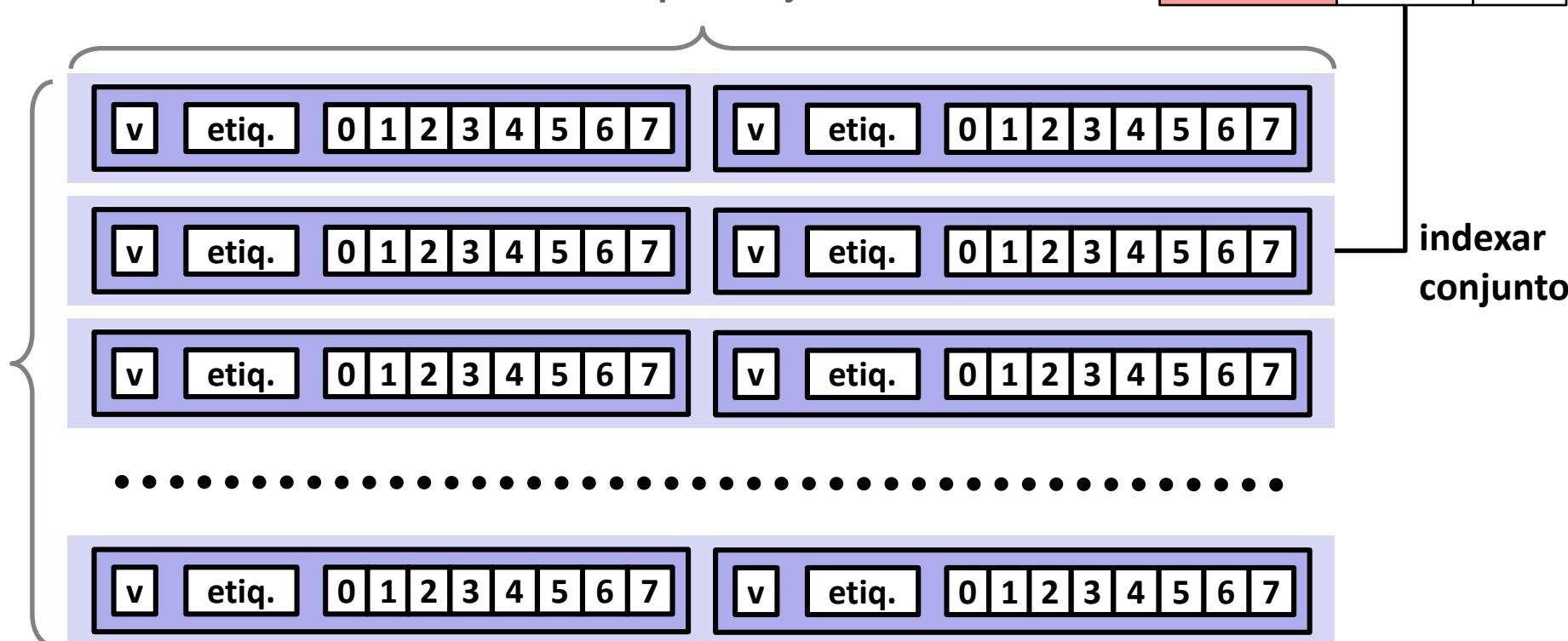
Cache Asociativa por Conjuntos de V vías (con $V=2$)

$V = 2$: Dos líneas por conjunto

Suponer: tamaño bloque cache $B=8$ bytes

2 líneas por conjunto

Dirección de un short int:



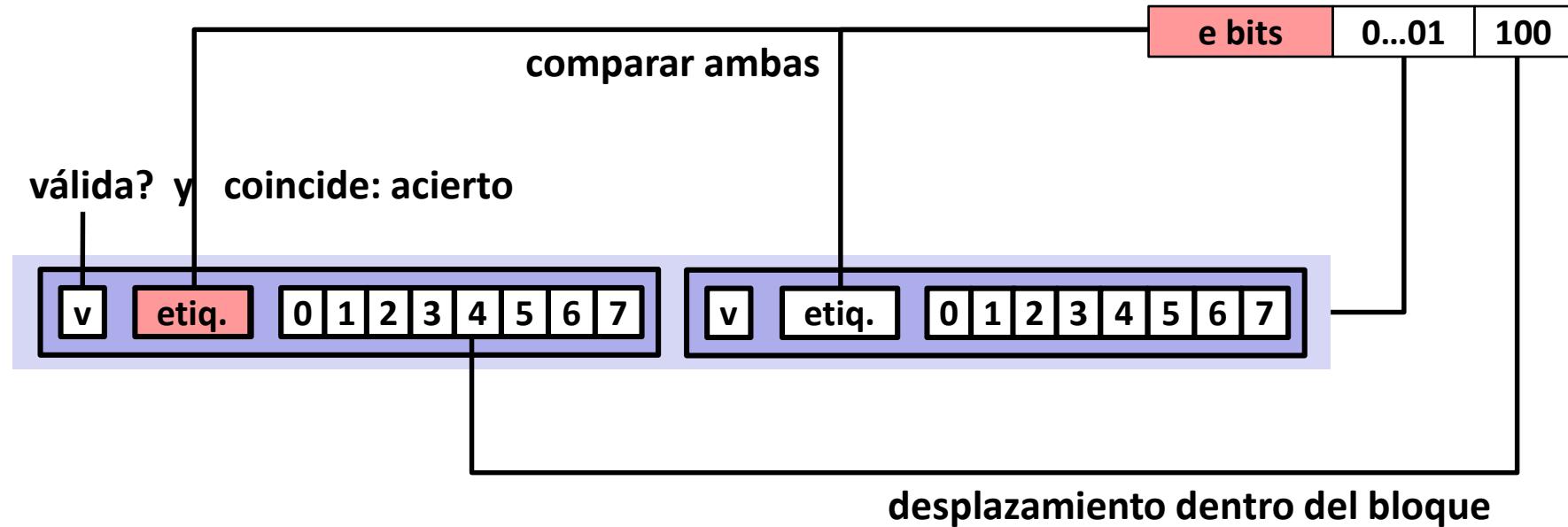
C conjuntos

Cache Asociativa por Conjuntos de V vías (aquí V=2)

V = 2: Dos líneas por conjunto

Suponer: tamaño bloque cache B=8 bytes

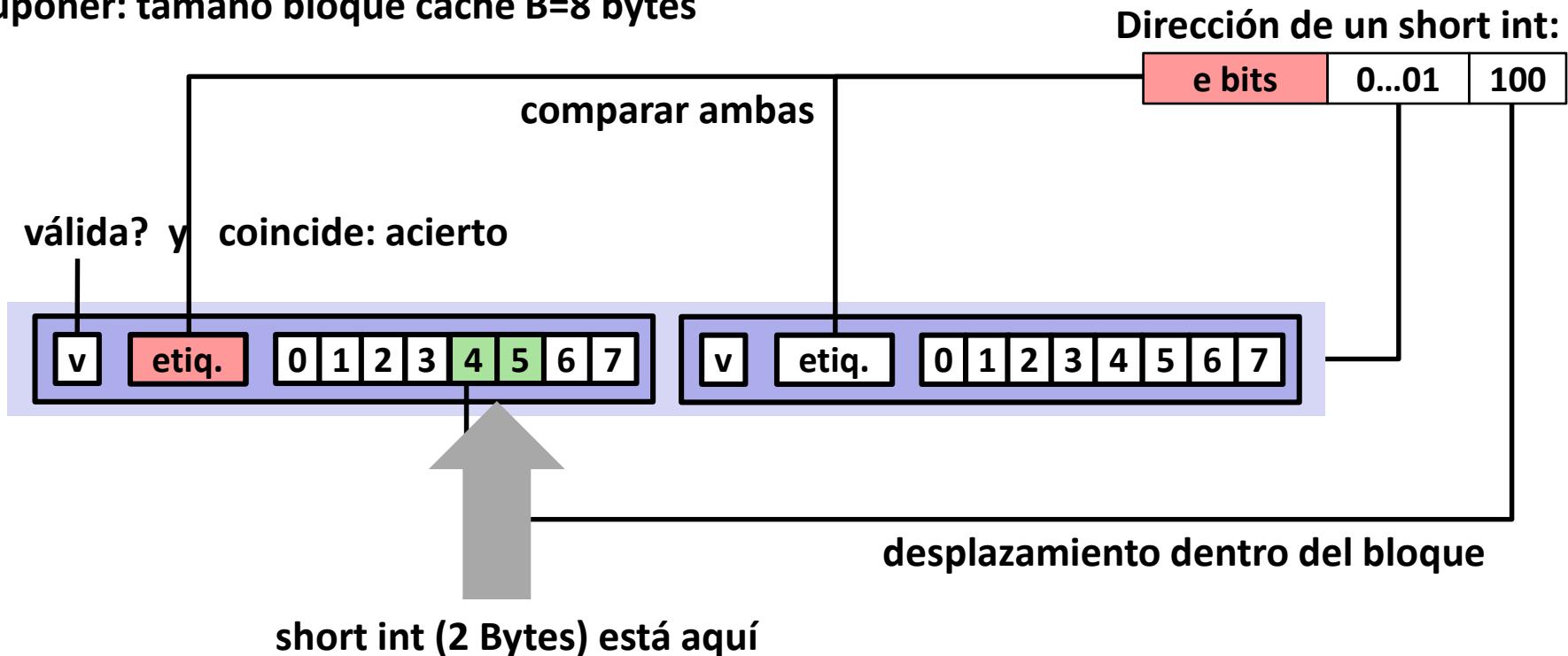
Dirección de un short int:



Cache Asociativa por Conjuntos de V vías (aquí V=2)

V = 2: Dos líneas por conjunto

Suponer: tamaño bloque cache B=8 bytes



Si ninguna coincide o no válida (= fallo):

- Se escoge una línea del conjunto para desalojo y reemplazo
- Políticas de reemplazamiento: aleatoria, menos rec.uso (LRU), ...

Simulación cache asociativa conjuntos 2-vías

e=2 c=1 b=1

XX	x	x
-----------	----------	----------

direcciones 4 bits ($M=16$ bytes)

C=2 conjuntos, V=2 vías (bloq./conj.), B=2 bytes/bloque

Traza de direcciones (lecturas, un byte por lectura):

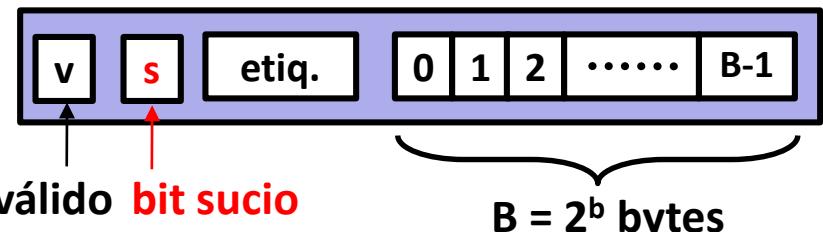
0	[<u>0000</u> ₂],	fallo
1	[<u>0001</u> ₂],	acierto
7	[<u>0111</u> ₂],	fallo
8	[<u>1000</u> ₂],	fallo
0	[<u>0000</u> ₂]	acierto

	v	Etiq.	Bloque
Conj. 0	1	00	M[0-1]
	1	10	M[8-9]
Conj. 1	1	01	M[6-7]
	0		

¿Qué hay de las escrituras?

■ Múltiples copias de los datos:

- L1, L2, L3, Mem. principal, Disco



■ ¿Qué hacer en acierto escritura?

- Write-through (escribir inmediatamente en la memoria) ([Escritura directa](#))
- Write-back (diferir escritura hasta reemplazo de la línea) ([Escritura diferida](#))
 - Cada línea caché necesita un **bit sucio** (=1 si línea difiere de bloque M)

■ ¿Qué hacer en fallo escritura?

- Write-allocate (cargar y actualizar línea en cache) ([Asignación en Escritura](#))
 - Conveniente si van a haber más escrituras a ese bloque
- No-write-allocate (escribir directo a memoria, sin cargar en cache)

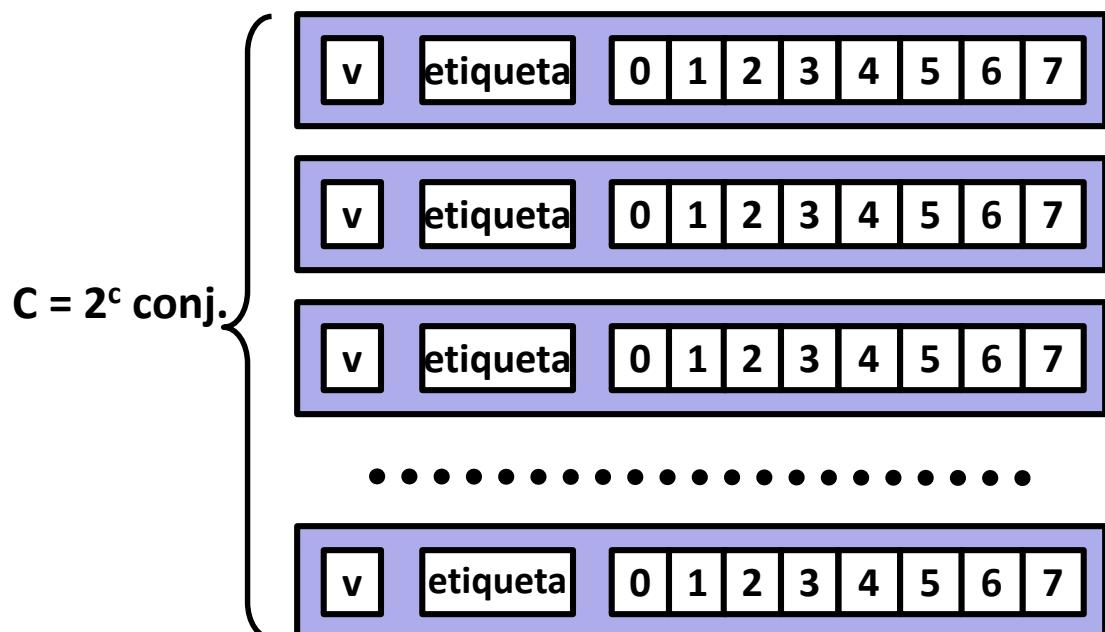
■ Usual

- Write-through + No-write-allocate (Escritura directa sin Asignación en Escritura)
- Write-back + Write-allocate (Post-Escritura con Asignación en Escritura)

¿Por qué indexar con los bits intermedios?

Correspondencia directa: Una línea por conjunto

Suponer: tamaño bloque cache $B=8$ bytes



Método estándar:
Indexar con bits intermedios

Dirección de un int:



localizar conjunto

Método alternativo (hipotético):
Indexar con bits superiores

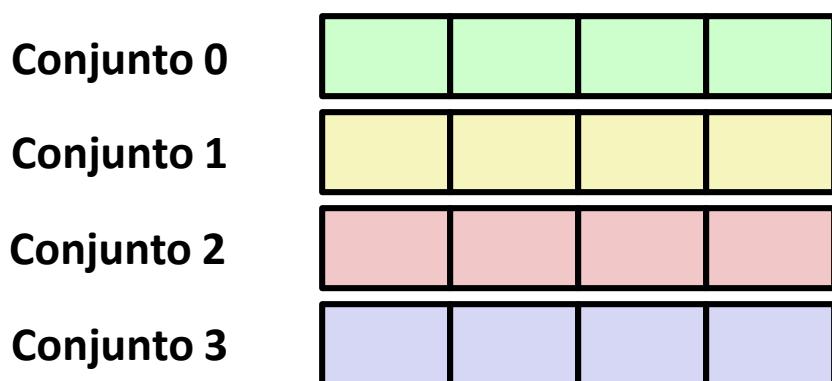
Dirección de un int:



loc. conj.

Ilustración de los métodos de indexación

- Memoria de 64 bytes
 - Direcciones de 6 bits
- Cache de 16B, corresp. directa
- T. bloque $B=4$ ($\Rightarrow C=4$ conj. ¿Por qué?)
- 2 bits etiq., 2 bits índ., 2 bits despl.



0000xx
0001xx
0010xx
0011xx
0100xx
0101xx
0110xx
0111xx
1000xx
1001xx
1010xx
1011xx
1100xx
1101xx
1110xx
1111xx

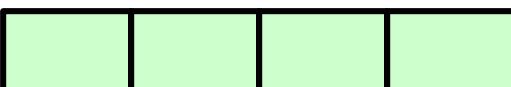
Indexado c/bits intermedios

- Direcciones de la forma **TTSSBB**

- **TT** bits etiqueta
- **SS** bits índice conjunto
- **BB** bits desplazamiento bloque

- Hace buen uso de localidad espacial

Conjunto 0



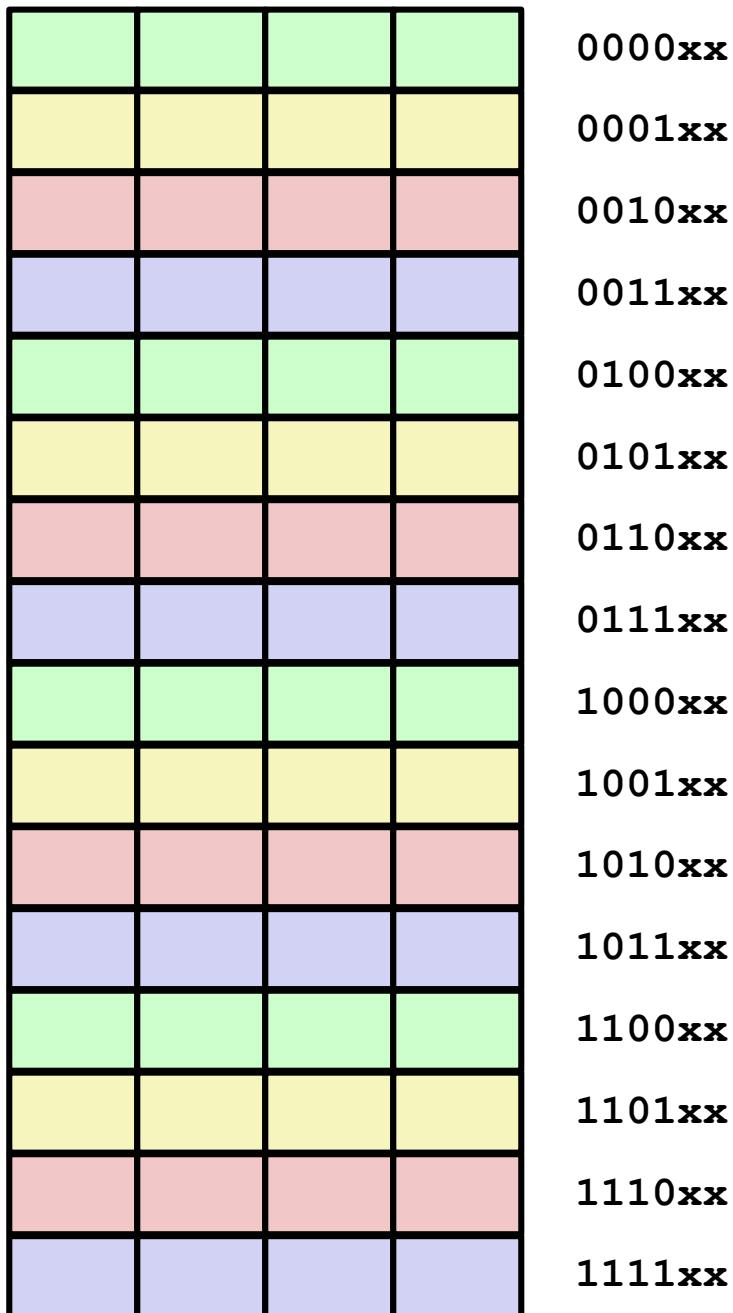
Conjunto 1



Conjunto 2

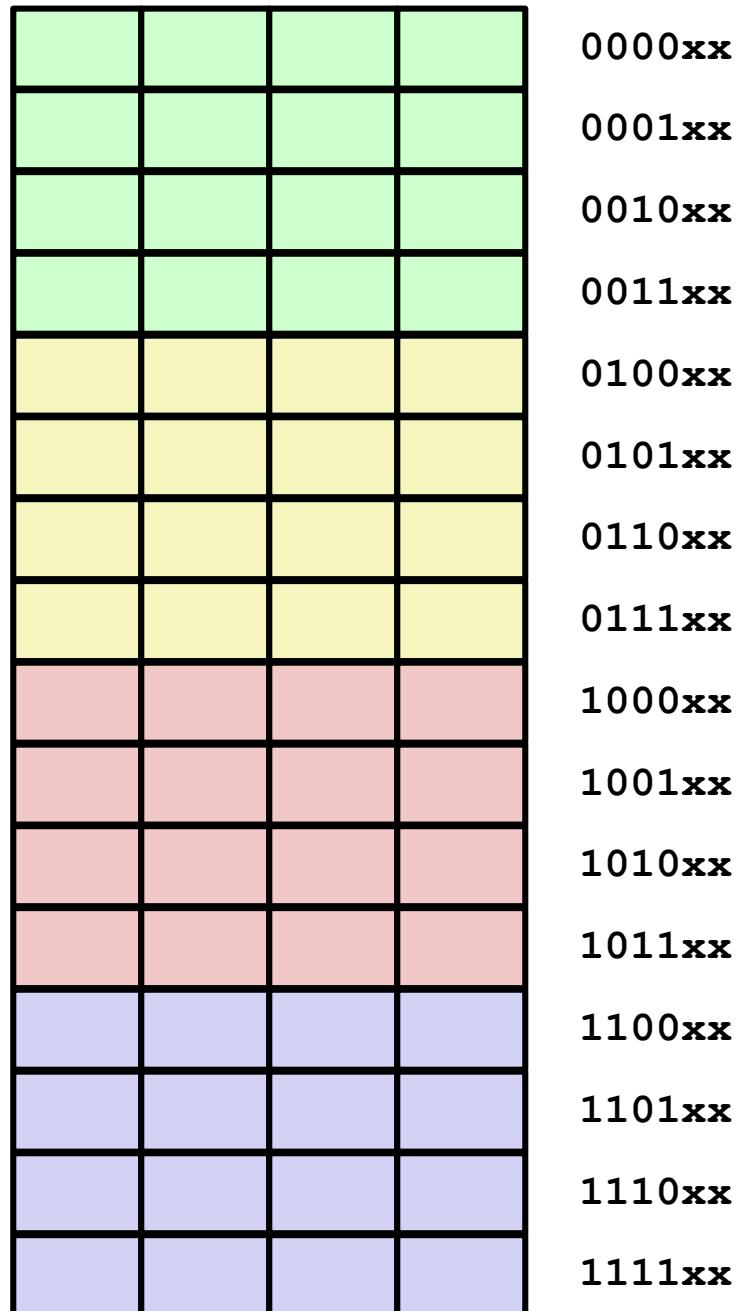
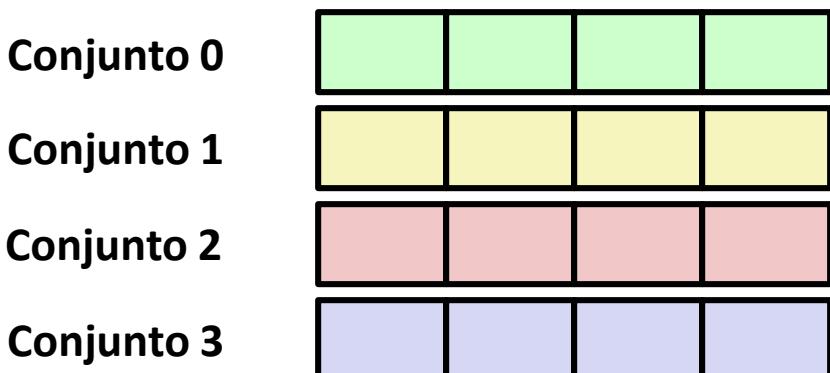


Conjunto 3



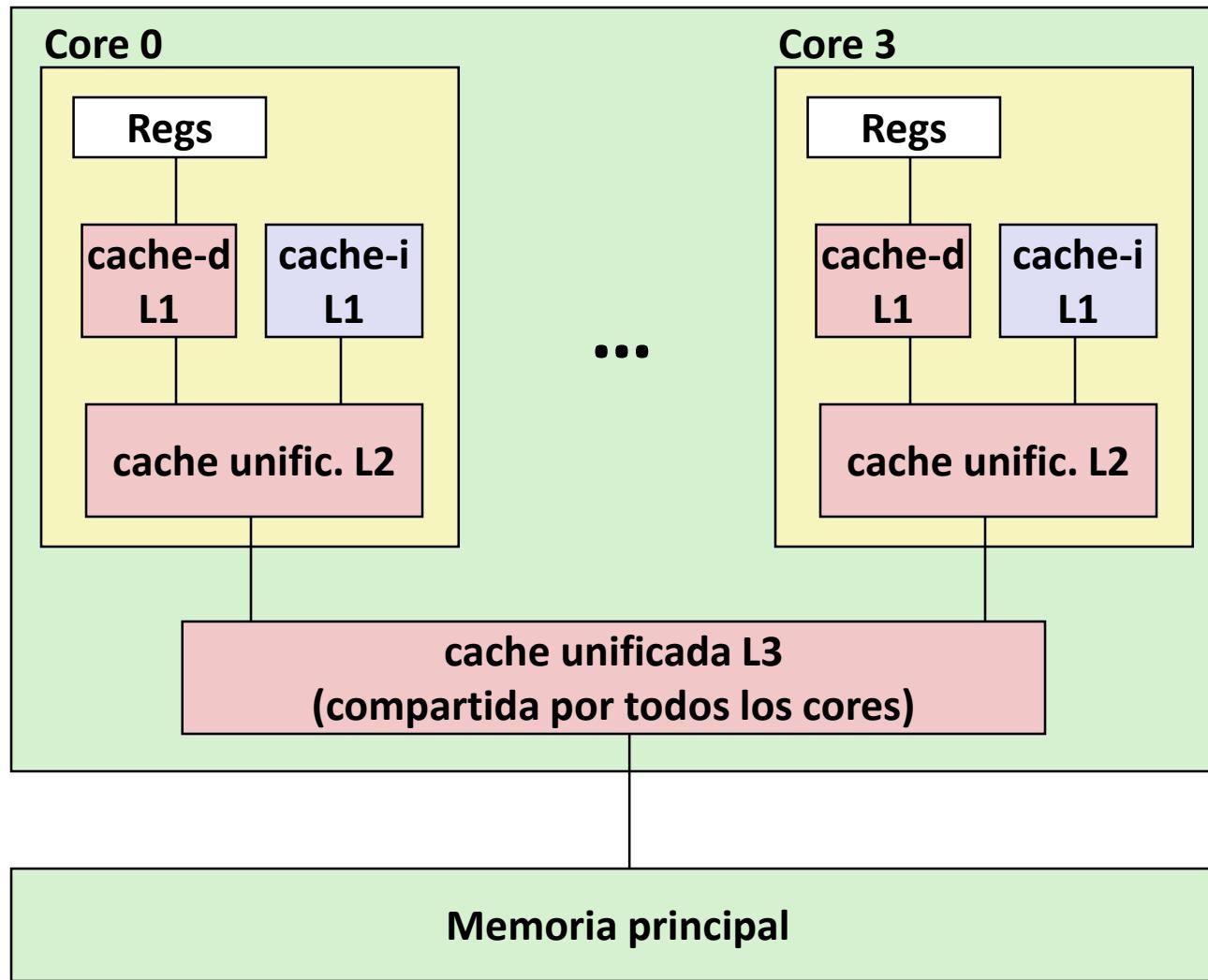
Indexado c/bits superiores

- **Direcciones de la forma SS_TT_BBB**
 - **SS** bits índice conjunto
 - **TT** bits etiqueta
 - **BB** bits desplazamiento bloque
 - **Programa con alta localidad espacial generaría muchos conflictos** (fallos por conflicto)



Jerarquía de caches del Intel Core i7

Paquete Procesador



cache-i y cache-d L1:

32 KB, 8-vías,
Acceso: 4 ciclos

cache unificada L2:

256 KB, 8-vías,
Acceso: 10 ciclos

cache unificada L3:

8 MB, 16-vías,
Acceso: 40-75 ciclos

Tamaño de bloque: 64 B
en todas las caches

...pero el propio Intel prefiere Paquete

† Processor package debería traducirse por Empaquetamiento 29

Ejemplo: cache de datos L1 del Core i7

32 KB 8-vías (asoc.conj.)

64 bytes/bloque

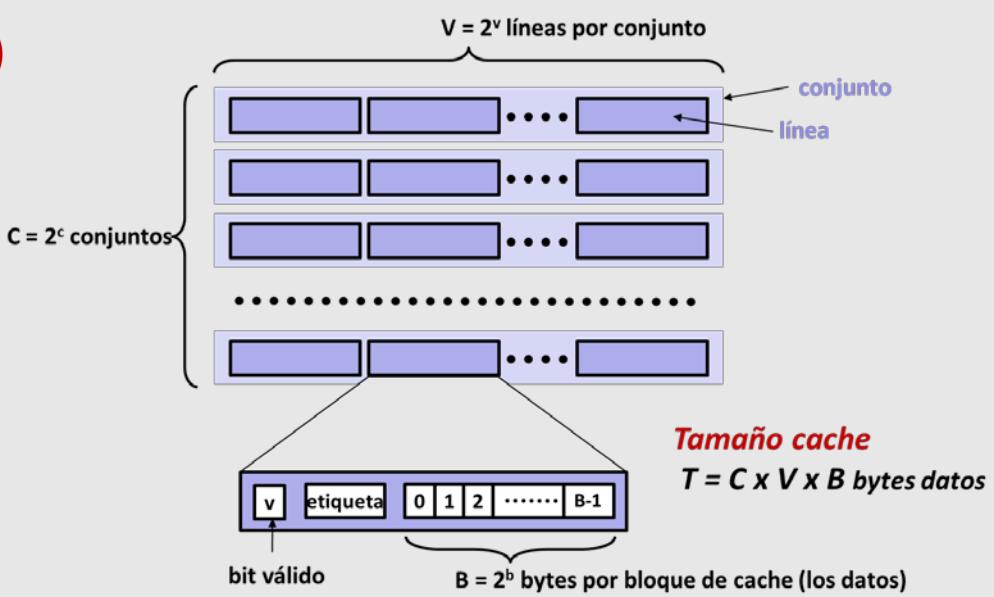
47 bit rango direcciones

B = , b =

V = , v =

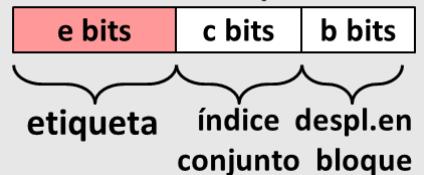
C = , c =

T =



	Hex	Decimal	Binary
0	0	0000	
1	1	0001	
2	2	0010	
3	3	0011	
4	4	0100	
5	5	0101	
6	6	0110	
7	7	0111	
8	8	1000	
9	9	1001	
A	10	1010	
B	11	1011	
C	12	1100	
D	13	1101	
E	14	1110	
F	15	1111	

Dirección inicio palabra:



despl. bloque: _ bits

índice conj.: _ bits

etiqueta: _ bits

Dirección de Pila:
0x00007f7262a1e010

despl. bloque: **0x??**
índice conj.: **0x??**
etiqueta: **0x??**

Ejemplo: cache de datos L1 del Core i7

32 KB 8-vías (asoc.conj.)

64 bytes/bloque

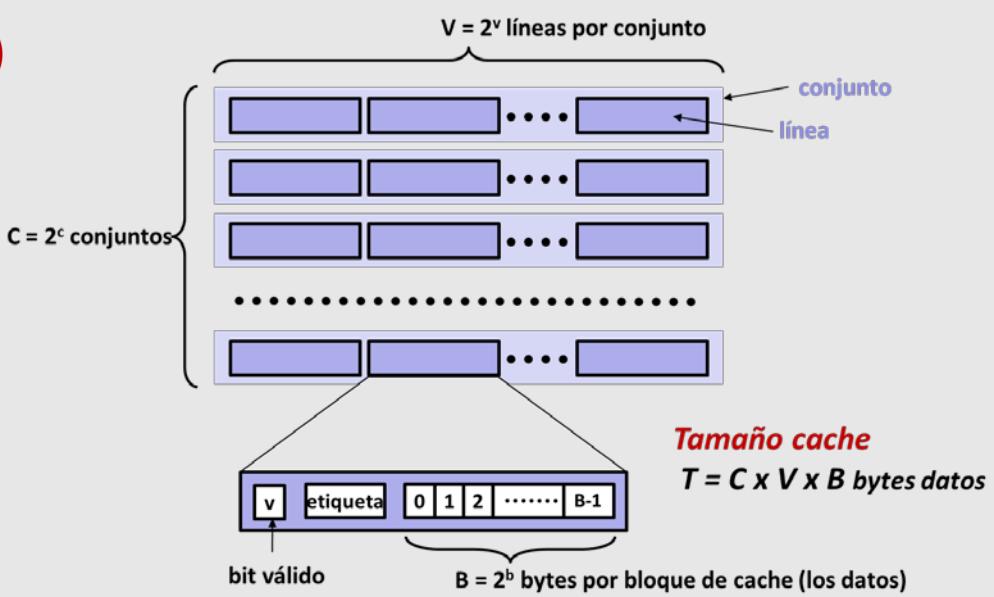
47 bit rango direcciones

$$B = 64, \quad b = 6$$

$$V = 8, \quad v = 3$$

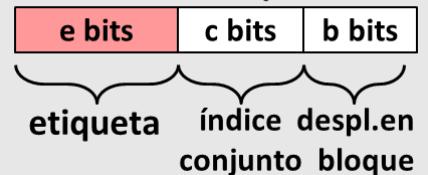
$$C = 64, \quad c = 6$$

$$T = 64 \times 64 \times 8 = 32,768$$



	Hex	Decimal	Binary
0	0	0000	00000000
1	1	0001	00000001
2	2	0010	00000010
3	3	0011	00000011
4	4	0100	00000100
5	5	0101	00000101
6	6	0110	00000110
7	7	0111	00000111
8	8	1000	00001000
9	9	1001	00001001
A	10	1010	00010010
B	11	1011	00010011
C	12	1100	00010100
D	13	1101	00010101
E	14	1110	00010110
F	15	1111	00010111

Dirección inicio palabra:



despl. bloque: 6 bits
índice conj.: 6 bits
etiqueta: 35 bits

Dirección de Pila:
 $0x00007f7262a1e010$

0000 0001 0000

despl. bloque: 0x10
índice conj.: 0x0
etiqueta: 0x7f7262a1e

Cache: resumen de políticas de colocación

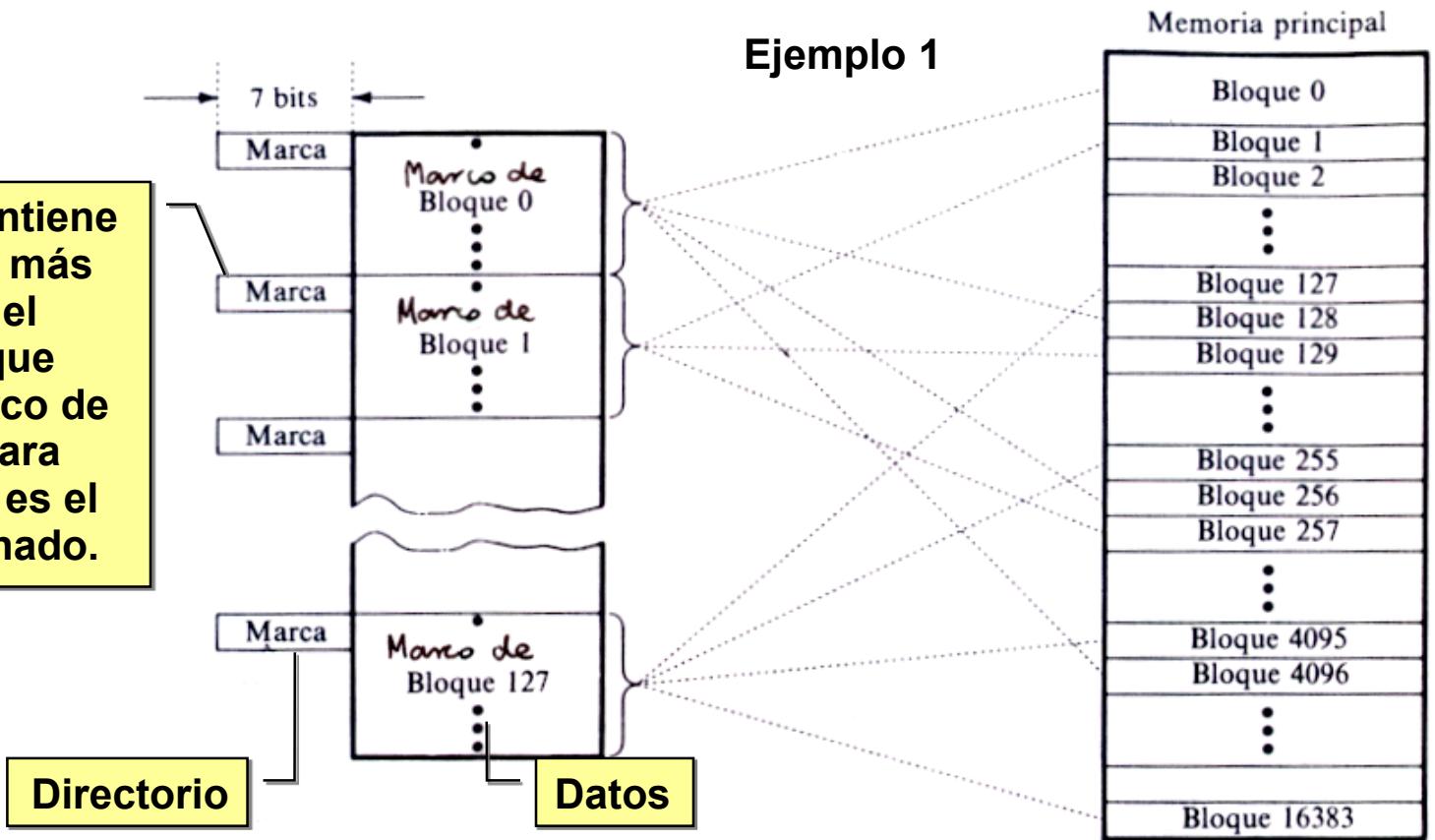
- **Organización (C,V,B,m)**
 - La caché tiene $2^c \times 2^v = 2^{c+v}$ líneas. Un bloque tiene 2^b bytes. Una dirección física tiene m bits. La MP tiene 2^m bytes (2^{m-b} bloques).
- **Correspondencia directa**
 - Bloque i de MP \Rightarrow línea $i \bmod 2^l$ de cache (L líneas= $2^l=2^{c+v}=2^c$ con $v=0$)
- **Correspondencia totalmente asociativa**
 - Bloque i de MP \Rightarrow cualquier línea de cache
- **Correspondencia asociativa por conjuntos**
 - Bloque i de MP \Rightarrow conjunto $i \bmod 2^c$ de cache (cualquier línea del conjunto)
- **Consideraremos el “Ejemplo 1”:**
 - Tamaño de caché: 2K bytes. $\Rightarrow c+v+b=11$
 - 16 bytes por bloque. $b=4 \Rightarrow c+v=7$, **128 líneas en cache**
 - Memoria principal máx: 256K bytes. $\Rightarrow m=18$, **16K bloques en MP**
 - $(CxV=128, B=16, m=18)$, $c+v=7, b=4, c+v+b=11, CxVxB=2K$

Cache: política de colocación

■ Correspondencia directa

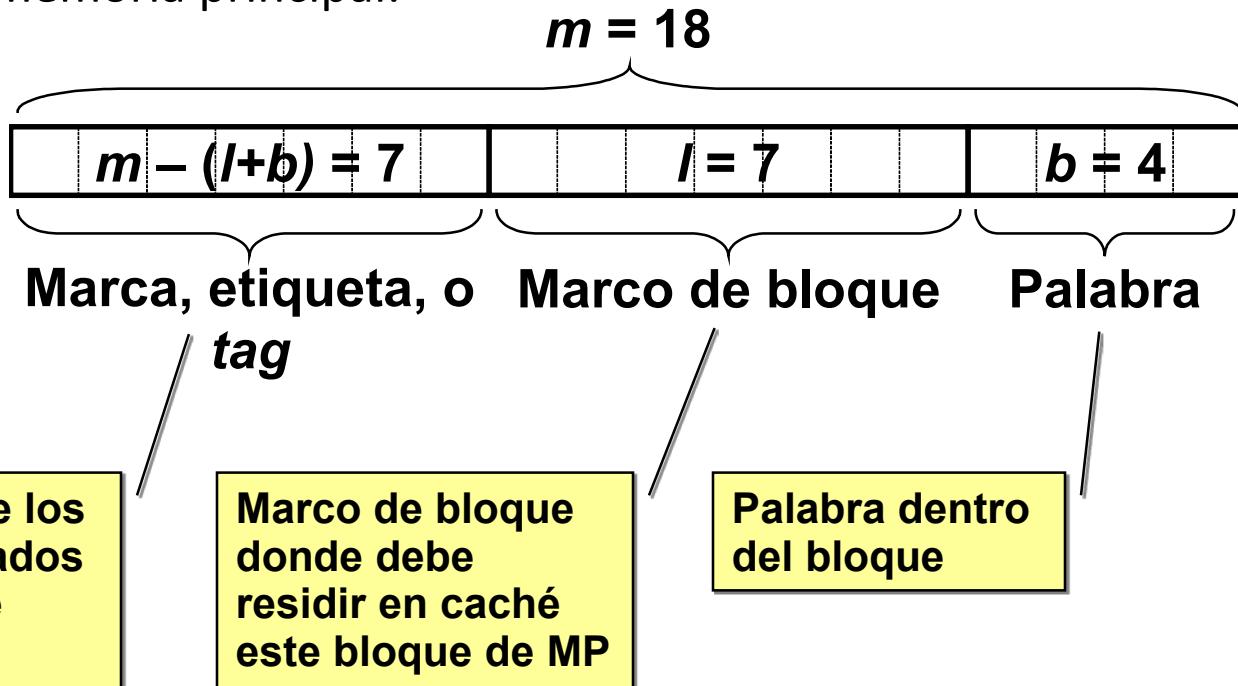
- Bloque i de MP \Rightarrow línea $i \bmod 2^l$ de caché.
- A cada línea le corresponde sólo un subconjunto de bloques de MP.

Cada marca contiene los $m-(l+b)$ bits más significativos del bloque de MP que hay en ese marco de bloque. Sirve para identificar cuál es el bloque almacenado.



Cache: política de colocación

- Dirección de memoria principal:



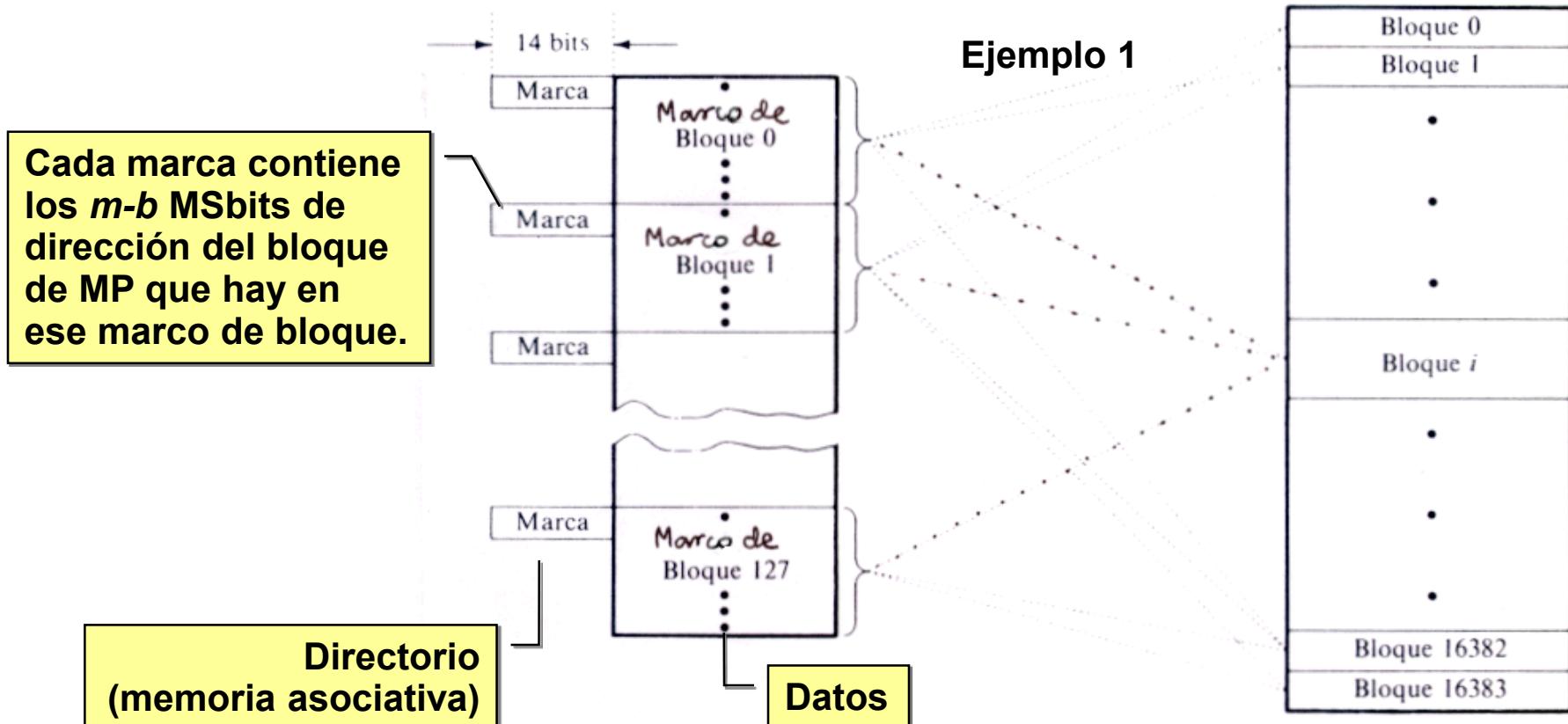
✓ Simplicidad y bajo coste.

✗ Si dos o más bloques, utilizados alternativamente, corresponden al mismo marco de bloque \Rightarrow el índice de aciertos se reduce drásticamente.

Cache: política de colocación

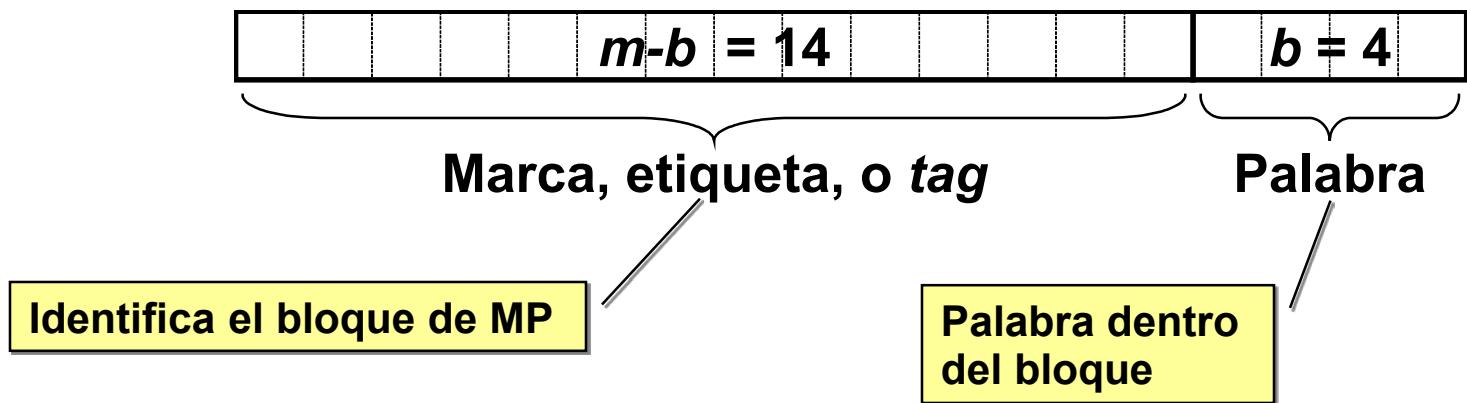
■ Correspondencia totalmente asociativa

- Un bloque de MP puede residir en cualquier marco de bloque de caché.
- Cuando se presenta una solicitud a la caché, todas las marcas se comparan simultáneamente para ver si el bloque está en la caché.



Cache: política de colocación

- Dirección de memoria principal:



✓ **Flexible.** Permite cualquier combinación de bloques de MP en la caché. Elimina en gran medida conflictos entre bloques.

✗ **Compleja y costosa de implementar (por la memoria asociativa).**

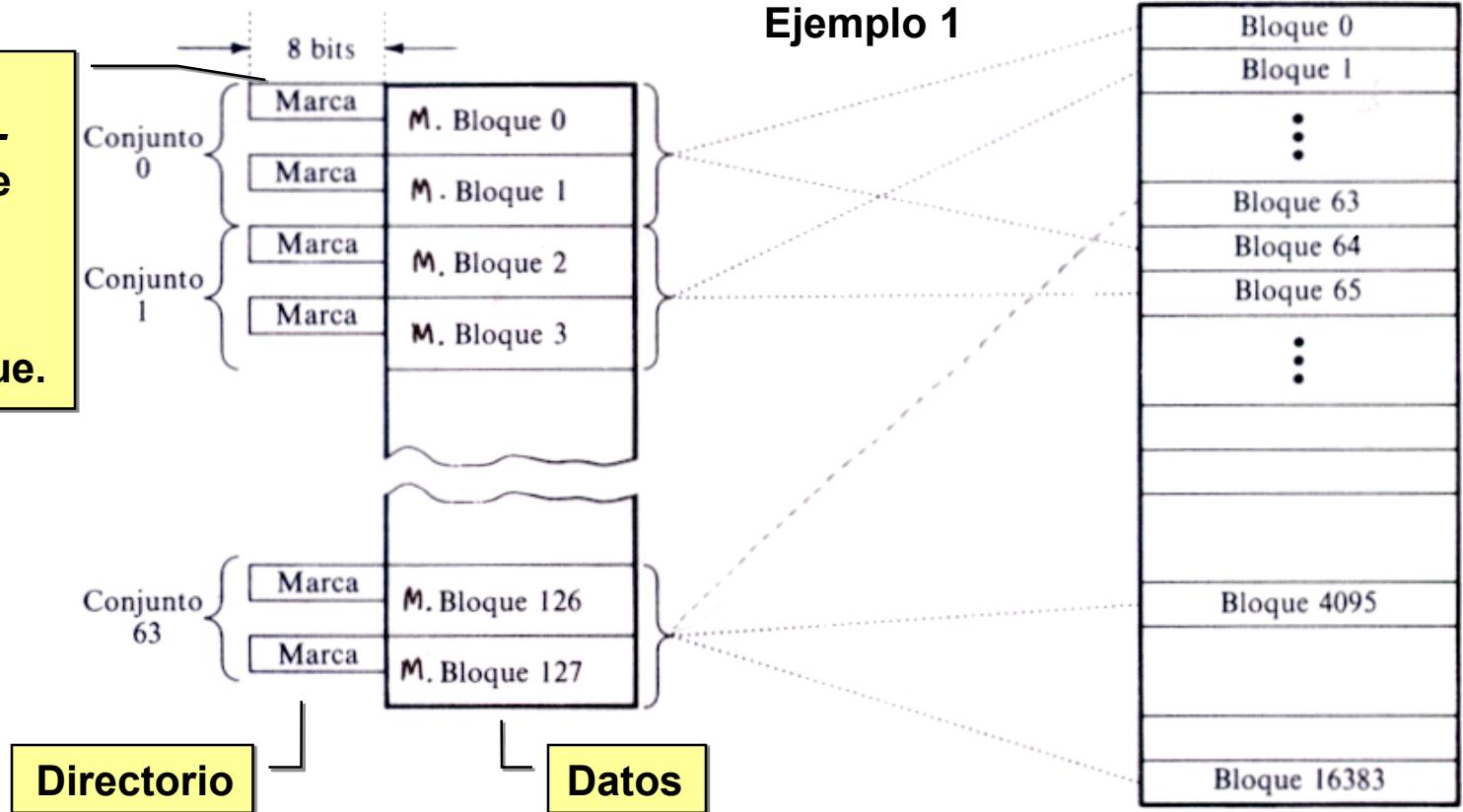
Cache: política de colocación

■ Correspondencia asociativa por conjuntos

- La caché se subdivide en 2^c conjuntos disjuntos
 - 2^v marcos de bloque / conjunto
- Bloque i de MP \Rightarrow conjunto $i \bmod 2^c$ de caché. Dentro de ese conjunto puede estar en cualquier marco de bloque.
- Hay dos fases en el acceso a caché:
 - Selección directa del conjunto donde puede estar ese bloque.
 - Búsqueda asociativa (dentro del conjunto) de la marca.
- A la correspondencia asociativa por conjuntos con 2^v marcos de bloque / conjunto también se le llama correspondencia asociativa de 2^v vías.
 - La vía i está formada por todos los marcos de bloque de la caché que ocupan el lugar i -ésimo dentro de su conjunto.
 - Completar enunciado del Ejemplo 1 fijando $v = 1$, $2^v = 2$ vías.

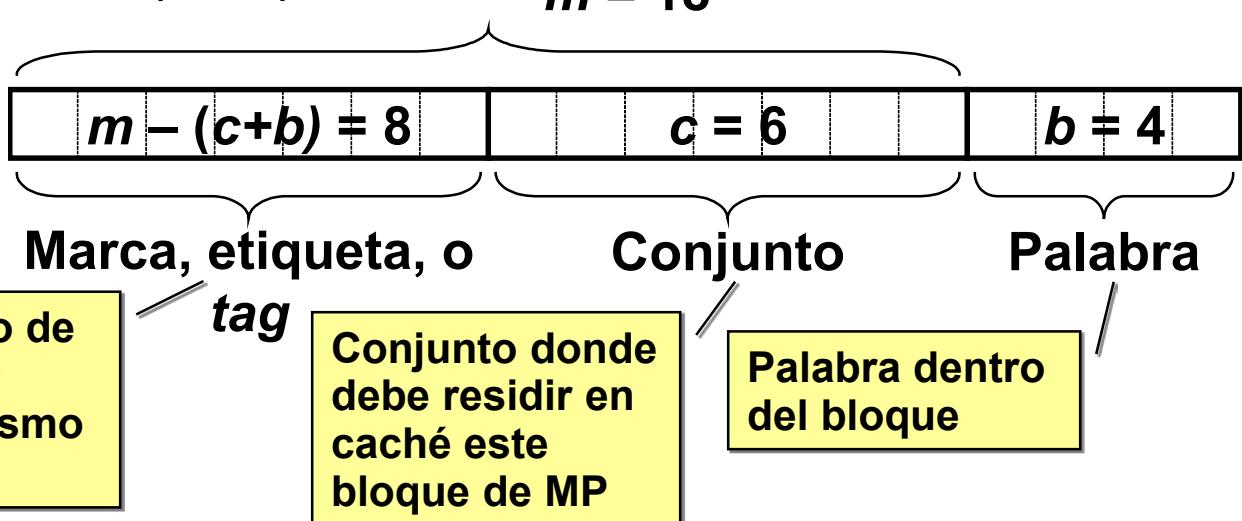
Cache: política de colocación

Cada marca contiene los $m - (c+b)$ MSbits de dirección del bloque de MP que hay en ese marco de bloque.



Cache: política de colocación

- Dirección de memoria principal:



$c = 0$ (1 conjunto) \Rightarrow corresp. totalmente asociativa.

$c = n$ (1 marco de bloque / conjunto) \Rightarrow corresp. directa.

$0 < c < n \Rightarrow$ se pretende reducir el **coste de la totalmente asociativa** manteniendo un **rendimiento similar** \Rightarrow es la técnica más utilizada.

✓ Resultados experimentales demuestran que un tamaño de conjunto de 2 a 16 marcos de bloque funciona casi tan bien como una corresp. totalmente asociativa con un incremento de coste pequeño respecto de la corresp. directa.