

* → 3.1

OPEROANDO	VALUE
%eax	0x100
0x104	0xA8
\$0x108	0x108
(%eax)	0xFF
4(%eax)	0xAB
9(%eax, %edx)	0x11
26(%eax, %edx)	0x13
0xF4(%eax, 4)	0xFF
(%eax, %edx, 4)	0x11

* → 3.2

mov %eax, (%esp) → mov
 mov (%eax), %dx → movw
 mov \$0xFF, %SI → movb
 mov(%esp, %edx, 4), %dh → movb
 push \$0xFF → pushl
 mov %dx, (%eax) → movw
 pop %edi → popl

* → 3.3

* → 3.4

Línea	Explicación
1	No se puede usar %sl como registro de direcciones.
2	El sufijo de instrucción no es compatible con el 1B registro.
3	Destino y origen no pueden ser referencias de memoria.
4	No existe %sh
5	No puede tener valor inmediato
6	Tamaño incorrecto de destino
7	Igual que la 2.

src-t	dest-t	Instrucción
int	int	movl %eax, (%edx)
char	int	movssl %al, (%edx)
char	signed	movbsl %al, (%edx)
unsigned char	int	movzbl %al, (%edx)
int	char	movsb %al, (%edx)
unsigned	unsigned char	movzs %al, (%edx)
unsigned	char	movfl %eax, (%edx)

*→ 3.5

Void decadel(int *xp, int *yp, int *zp)

```

int x = *xp;
int y = *yp;
int z = *zp;

```

```
*xp = x;
```

```
*yp = y;
```

```
*zp = z;
```

{ }

→ 3.6

A) %eax se establece en la dirección de la instrucción popl.

B) Lo que ocurre es que no se trata de una llamada de procedimiento real, ya que el control sigue el mismo orden que las instrucciones y la dirección de retorno se extrae de la pila.

C) ~~obtener~~ Si propósito es el de obtener el valor del contador de programación en un registro de enteros.

→ 3.6

A) $(256 \rightarrow \text{factor } 16 \times 104) \cong 25 \text{ años} = \underline{\underline{2035}}$ B) $\left[(16 \text{ exabytes}) \wedge 1054 \times 109 \text{ sobre } 16.3 \text{ GB} \right] \cong 53 \text{ años} = \underline{\underline{2063}}$ C) Aumentar presupuesto factor (10 veces de 6 años) $\rightarrow \underline{\underline{2029 - 2057}}$

PÁGINA 2

TOTAL 8 + $\frac{3}{4}$ EJERCICIOS
+ $\frac{4}{7}$ AWALADOS

MARIO RODRIGUEZ DIAZ

20C-6IT
7473H95-X

* → 3.47

Src-t	dest-t	Instrucción	S	D
long	long	movq	%rdi	%rax
int	long	movslq	%edi	%rax
char	long	movsq	%dl	%rax
unsigned int	unsigned long	movl	%edi	%eax
" char	" long	movzbq	%dl	%rax
" "	" "	movzl	%dl	%eax
long	int	movsq	%edi	%rax
" "	" "	movl	%edi	%eax
unsigned long	unsigned	movl	%edi	%eax

* → 3.6

Instrucción	Resultados
leal 6(%eax,%edx)	6+x
leal (%eax,%ecx),%edx	x+y
leal (%ecx,%ecx,4),%edx	x+4y
leal 7(%eax,%eax,8),%edx	7+9x
leal 0xA(%ecx,4),%edx	10+4y
leal 9(%eax,%ecx,2),%edx	9+x+2y

Instrucción	Destinatario	Valor
add(%ecx,(%eax))	0x100	0x100
susb %edx,4(%eax)	0x104	0xA8
imul \$16,%eax,%edx,4	0x10C	0x110
incl 8(%eax)	0x108	0x14
dec %ecx	%ecx	0x0
susb %edx,%eax	%eax	0xFD

*→ 3.8

```

movl 8(%esp), %eax      # Taux x
sal $2, %eax             # x <= 2
movl 12(%esp), %ecx    # Taxun n
sarl %cl, %eax           # x >= 1

```

*→ 3.9

```

int arith(int x, int y, int z) {
    int t1 = x * y;
    int t2 = t1 >> 3;
    int t3 = ~t2;
    int t4 = t3 - z;
    return t4;
}

```

*→ 3.11

movl 8(%esp), %eax → Largur x en %eax → movl 8(%esp), %eax

cltd

→ Extender signo en %edx → movl \$0, %edx

idivl 12(%esp)

→ Dividir x(645) por y → divl 12(%esp)

movl %eax, 4(%esp)

→ Guardar x/y → movl %eax, 4(%esp)

movl %edx, (%esp)

→ Guardar x%y

→ movl %edx, (%esp)

*→ 3.16

void goto_cuid(int a, int *p)

```

{
    if (*p == 0)
        goto bne;
    if (a <= 0)
        goto done;
    *p += a;
done: return
}

```

PÁGINA 4

TOTAL:

+ 4 EJERCICIOS] 14
+ 10 ACUMULADOS]

* 3.17

```
int gotodiff_val(int x, int y)
{
    int result;
    if (x < y) goto true;
    result = x - y;
    goto done;
true:
    result = y - x;
done:
    return result;
}
```

* 3.18

```
int test(int x, int y)
{
    int val = x ^ y;
    if (x < -3) {
        if (y < x)
            val = y * x;
        else
            val = x + y;
    } else if (x > 2)
        val = x - y;
    return val;
}
```

* 3.22

```
int fun_a(unsigned x)
{
    int val = 0;
    while (x)
    {
        val ^= x;
        x >>= 1;
    }
    return val & ~0x1;
}
```

B) La función calcula la paridad del argumento "x".

* 3.23

A) int fun_b(unsigned x)

```

{
    int val = 0;
    int i;
    for (i = 0; i < 32; ++i)
    {
        val = (x & 0x1) | (val << 1);
        x >>= 1;
    }
    return val;
}
```

B) Lo que hace la función es invertir bits y guardarlos en "x".

PÁGINA S

TOTAL: 4 EJERCICIOS
+ 14 ACTIVIDADES 18

MARIO RODRIGUEZ DUR

2023 GII
7473749524

* d 3.10

- A) Esta instrucción lo que hace es poner el registro %edx a cero.
- B) La forma más directa sería con `mov $0,%edx`.
- C) La versión xorl requiere 2 bytes, mientras que la usual 5 bytes.

* d 3.12

- A) num-t es del tipo `unsigned long long`.
- B) La representación completa del producto sería de 96 bits de largo, pero solo se requieren los 64 bits de orden bajo. $y = y_64 \cdot 2^{32} \Leftrightarrow x \cdot y = x \cdot y_64 \cdot 2^{32} + x \cdot y_32$. El resultado final tiene como tl de orden bajo y s+th con orden alta.

* d 3.13

- A) "l" y los ID Registros indican operaciones de 32 bits. Comparación \rightarrow complemento a dos
data-t es int, por tanto.
- B) El sufijo "w" y los ID Registros indican op. de 16 bits. Comparación: $>=$
data-t es short
- C) El sufijo 'l' y los ID Regs. indican op. 8 bits. Comparación: ' $'$ sin signo
data-t es `unsigned char`
- D) El sufijo 'l' y los ID Regs. indican op. 32 bits. Comparación: ' $!=$ '
data-t puede ser `int`, `unsigned` o `ptrsize`.

* 3.14

- A) El sufijo 'l' y los 10 reg. indican op. de 32 bits. Comparación para ' $!=$ ' ^{con 8 bits}_{con 32 bits}.
data_t debe ser int, unsigned o punto.
- B) El sufijo 'w' y los 10 reg. indican 64 bits de op. Comparación para ' $==$ ' ^{con 32 bits}_{con 64 bits}.
data_t será short o unsigned short
- C) El sufijo 'l' y los 10 reg. indican op. de 8 bits. Comparación para complemento a dos de ' $>$ '.
data_t debe ser char
- D) El sufijo 'w' y los 10 reg. indican op. de 16 bits. Comparación para unsigned ' $>$ '.
data_t es unsigned short.

* 3.19

A) $n = 13 \rightarrow n! = 1.932.053.504$

B) Lo que sucede es que permite llegar hasta $n = 20$

* 3.24

A) El código entraría en un bucle infinito ya que la variable "entire" no permitiría que el índice o posición "i" se actualizara.

B) Haría que recuperar "entire" por "goto", saltando el cuerpo del bucle y llegando directamente a la actualización del índice.

* → 3.25

MARIO RODRÍGUEZ PINT

2016-24232854

A) $T_{MP} = (31 \text{ ciclos} - 16 \text{ ciclos}) \times 2 = \underline{\underline{30}}$ es la penalización.

B) Si el salto se predice de forma errónea a 16 ciclos predicciones + 30 penalizaciones
 $\frac{16 \text{ ciclos}}{46 \text{ ciclos}}$

* → 3.26

A) La operación es una división por una potencia de dos en desplazamiento a la derecha.

B)
 testl %edx, %eax # valor = $x+3$
 testl %eax, %edx # test de x
 cmovl %edx, %eax # si $>= 0$, valor = x
 srl \$1, %eax # Devuelve valor $>> 2$ ($= x/4$)

* → 3.27

```
int test(int x, int y)
{
    int val = x * 4;
    if(y > 0){
        if(x < y)
            val = x - y;
        else
            val = x * y;
    } else if(y < -2)
        val = x + y;
    return val;
}
```

* → 3.49

```
long fun_c(unsigned long x){
    long val = 0;
    int i;
    for(i=0; i<8; i++){
        val += x & 0x0000000000000001L;
        x >>= 1;
    }
    val += (val >> 32);
    val += (val >> 16);
    val += (val >> 8);
    return val & 0xFF;
```

PÁGINA 8

TOTAL: 4 EJERCICIOS
 $\frac{+24}{28}$ ACUMULADOS

* 3.28

- A) Las etiquetas de caso en el switch tienen los valores -2, 0, 1, 2, 3 y 4.
 B) El caso con el destino -16 tiene las etiquetas 2 y 3.

* 3.29

```
int switcher(int a, int b, int c)
{
    int answer;
    switch(a)
    {
        case 5:
            c = b ^ 15;
        case 0:
            answer = c / 12;
            break;
        case 2:
        case 7:
            answer = (c + b) << 2;
            break;
        case 4:
            answer = a;
            break;
        default:
            answer = b;
    }
    return answer;
}
```

* 3.31

El procedimiento debe guardar los registros en la pila antes de modificar sus valores y restaurarlos antes de volver.

* 3.32

Función prototipo:
`int fun(short c, char d, int *p, int x);`

* 3.34

- A) El registro %eax contiene el valor del parámetro "x", que puede utilizarse para calcular la expresión del resultado.

B) `int rfun(unsigned x)``if(x == 0)``return 0;``unsigned nx = x >> 1;``int rv = rfun(nx);``return (x & 0x1) + rv;`

- C) La función calcula la suma de bits en el argumento "x", lo hace recursivamente menos con el bit menos significativo.

TOTAL: 5 EXERCICIOS

+ 28 ACUMULADOS

* 3.50

Pueden haber cuatro prototipos válidos para incpros, dependiendo de si el parámetro "x" es long y si es signed o unsigned:

- void incpros_s (int x, long *q, int *t);
- void incpros_u (unsigned x, long *q, int *t);
- void incpros_s1 (long x, long *q, int *t);
- void incpros_u1 (unsigned long x, long *q, int *t);

* 3.51

%rsp	SP
0	Unused
-8	Unused
-16	a[3]
-24	a[2]
-32	a[1]
-40	a[0]

B) local_array:

```

    movq $2,-40(%rsp) # Guarda 2 en a[0]
    movq $3,-32(%rsp) # " " 3 en a[1]
    movq $5,-24(%rsp) # " " 5 en a[2]
    movq $7,-16(%rsp) # " " 7 en a[3]
    addl $3,%edi      # idx = i + 3
    movq -40(%rsp,%rdi,8),%rax # a[idx]
    ret                # Return

```

c) La función no cambia nunca el puntero de pila, sino que almacena todos sus valores locales en la región fuera de ella.

TOTAL: 2 EXERCICIOS
+ 33 ALUMNOS
35

PÁGINA 10

* 3.52

- A) El registro %rbx se utiliza para mantener el parámetro "x".
- B) El registro %rsi debe almacenarse en la pila. Este es el único uso de la pila en la función y por ello el código utiliza instrucciones push y pop para guardar y restaurar el registro.
- C)
- | | |
|-----------------------|-----------------------------|
| 1 - Guarda %rbx en | 7 - Llamada a rfact(xm1) |
| 2 - Copia x en %rsi | 8 - result = x * rfact(xm1) |
| 3 - result = 1 | 10 - done: |
| 4 - Test x | 11 - Restaura %rsi |
| 5 - Si x=0, goto done | 12 - return |
| 6 - xm1 = x-1 | |
- D) En lugar de utilizar el puntero de pila para incrementar y decrementar, el código puede utilizar pushq y popq para modificar y guardar o modificar el estado del registro.

* 3.35

Array	Tamaño el.	Tamaño total	Dirección inicial	Elemento i°
S	2	14	x _s	x _s + 2i
T	4	12	x _T	x _T + 4i
U	4	24	x _U	x _U + 4i
V	12	96	x _V	x _V + 12i
W	4	16	x _W	x _W + 4i

TOTAL: 2 EJERCICIOS
+ 35 ACUMULADOS
37

* 3.36

MARIO BORJA P. WJ12

202647437454

Expresión	Type	Valor	desplazamiento
$s+1$	short*	$x_5 + 2$	leal 2(%edx), %eax
$s[3]$	short	$M[x_5+6]$	movw 6(%edx), %ax
$\&s[i]$	short*	$x_5 + 2i$	leal (%edx,%ecx,2), %eax
$s[4*i+1]$	short	$M[x_5+8i+2]$	movw 2(%edx,%ecx,8), %ax
$s+i-s$	short*	$x_5+2i-10$	leal -10(%edx,%ecx,2), %eax
		0	-

* 3.37

Revisa como la referencia a la matriz $mat1$ está en el desplazamiento de bytes $4(x_i+j)$, mientras que la referencia a $mat2$ está en el desplazamiento $4(s_j+i)$. Es por ello que se pide determinar que $mat1$ dispone de 7 columnas y $mat2$ 5, siendo entonces

$$M = 5 \quad y \quad N = 7.$$

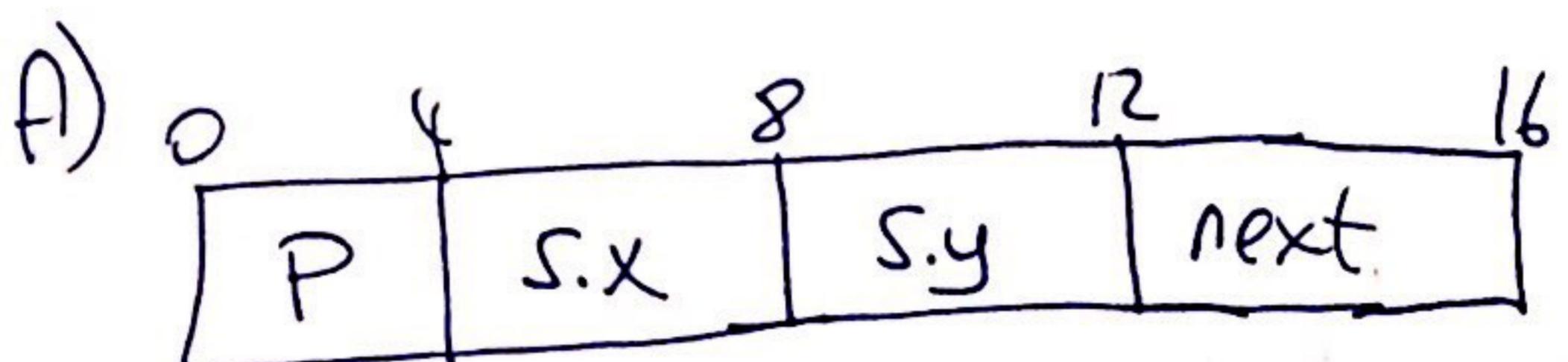
* 3.38

```
void fix_set_diag_sp(fix-matrix A, int val)
{
    int *Abase = &A[0][0];
    int index = 0;
    do {
        Abase[index] = val;
        index += (N+1);
    } while (index != (N+1)*N;
}
```

TOTAL : 3 EJERCICIOS
 $+ 3 + ACUMULADOS$
 $\frac{40}{40}$

PÁGINA 12

* 3.39



B) Utiliza 16 bytes

C) void sp_init(struct prob *sp){

sp->S.x = sp->S.y;

sp->p = &(sp->S.x);

sp->next = sp;

}

* 3.42

A)

	a	b	c	d	e	f	g	h
Tam.	4	2	8	1	4	1	8	4
Offset	0	4	8	16	20	24	32	40

B) La estructura tiene un total de 48 bytes

C) Con la nueva reorganización, 32 bytes.

	c	g	e	a	h	3	d	f
Tam.	8	8	4	4	4	2	1	1
Offset	0	8	16	20	24	28	30	31

TOTAL : 2 EJERCICIOS
+ 40 ASESINATOS
42

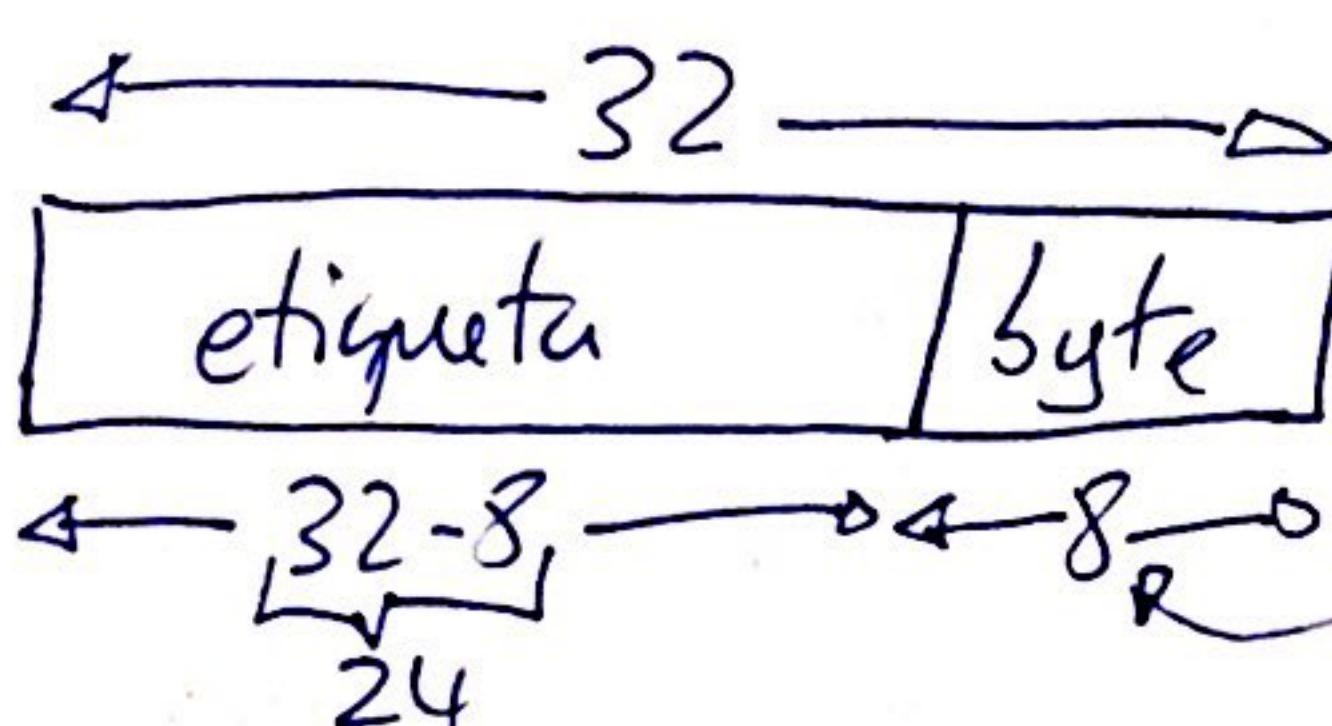
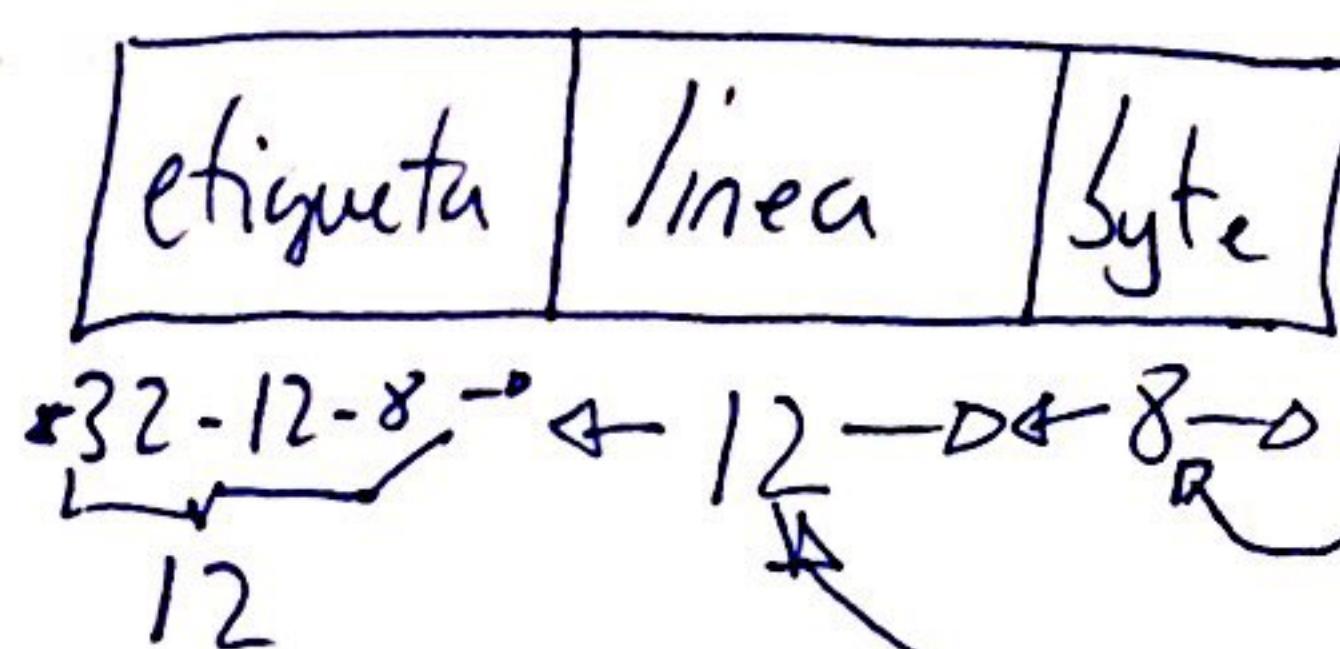
MEMORIA

21.

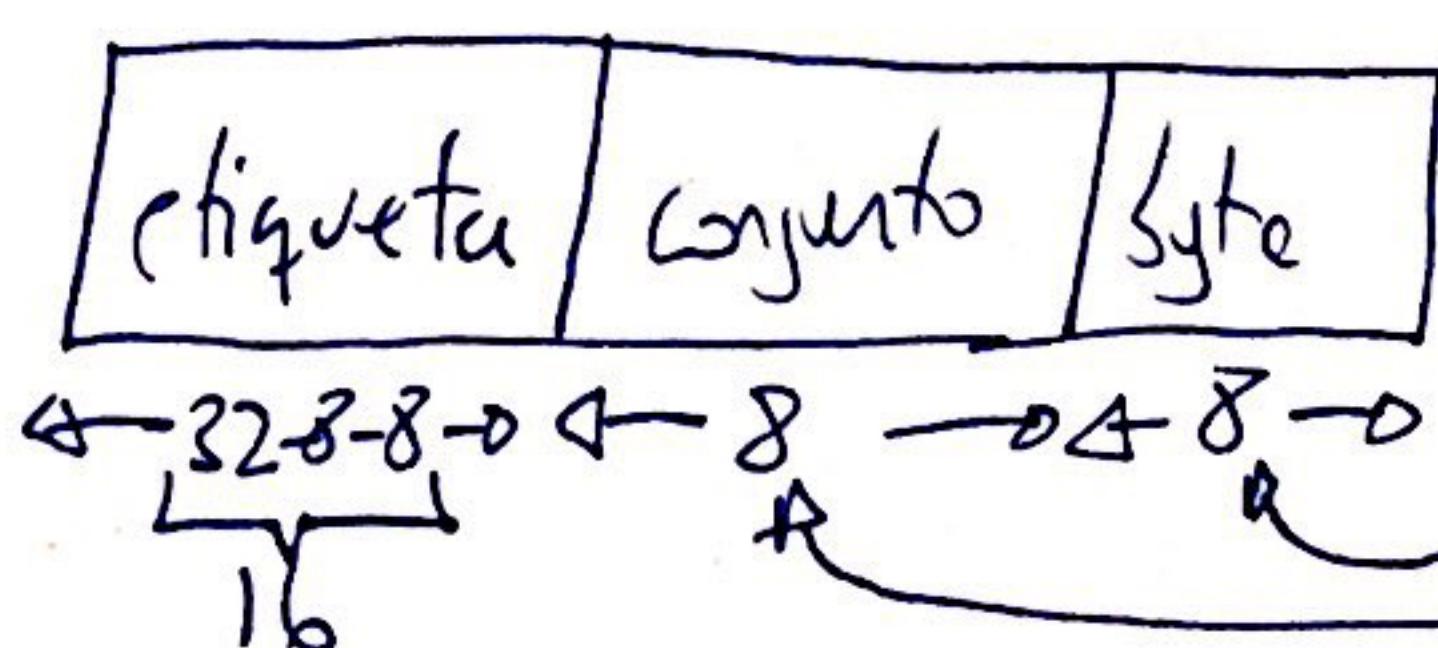
MP: $4\text{ GB} = 2^{32}\text{ B} \Rightarrow$ dirección memoria de 32 bits

MC: $1\text{ MB} = 2^{20}\text{ B}$

$256\text{ B/línea} = 2^8\text{ B/línea} \Rightarrow 8\text{ bits para determinar desplazamiento}$

a) TABACULANTE ASOCIATIVAb) DIRECTA

$$\frac{2^{20}\text{ B en cache}}{2^8\text{ B/línea}} = 2^{12} \text{ líneas en cache}$$

c) ASOCIATIVA POR CONJUNTOS \rightarrow 4 vías

$$\frac{2^{12}\text{ líneas cache}}{2^4\text{ líneas/conjunto}} = 2^8 \text{ conjuntos cache}$$

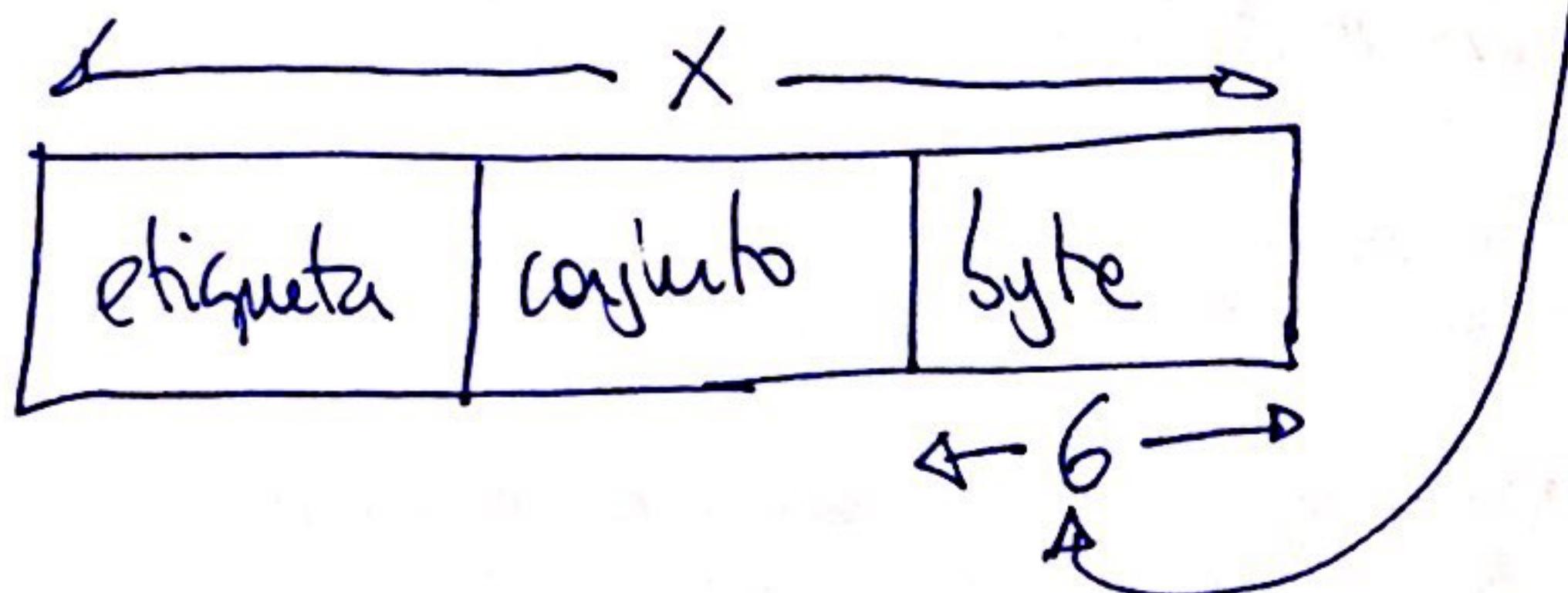
MEMORIA

G. SÓP16

a) ~~MAP: $32KB = 2^{15} B$~~

MC: $32KB = 2^{15} B$

$64B/\text{línea} = 2^6 B/\text{línea} \Rightarrow 6 \text{ bits para determinar desplazamiento}$

Asociativa por contenidos $\rightarrow 2$ vías

$2^{15} \text{ líneas en cache} = 2^9 \text{ líneas en cache}$
 $2^6 B/\text{línea}$

$2^9 \text{ líneas en cache} = 2^8 \text{ conjuntos}$
 $2 \text{ líneas/conjunto}$

b)
64 B/línea: $\frac{2^6}{2^2} = 2^4 \text{ ints/línea.}$

(ano se avanza de 4 en 4 ints, existe fallo cada $\frac{2^4}{2^2} = 4$ accesos.)

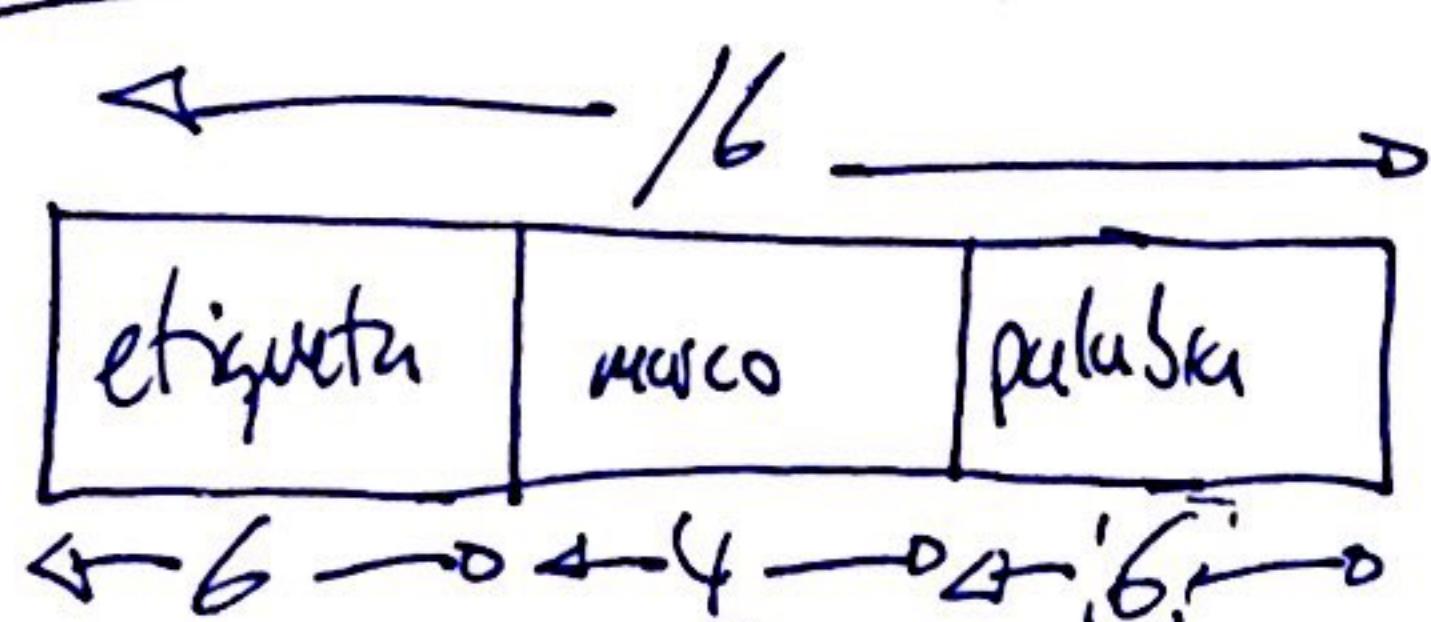
F---A---A---A---F---A...

PÁGINA 15

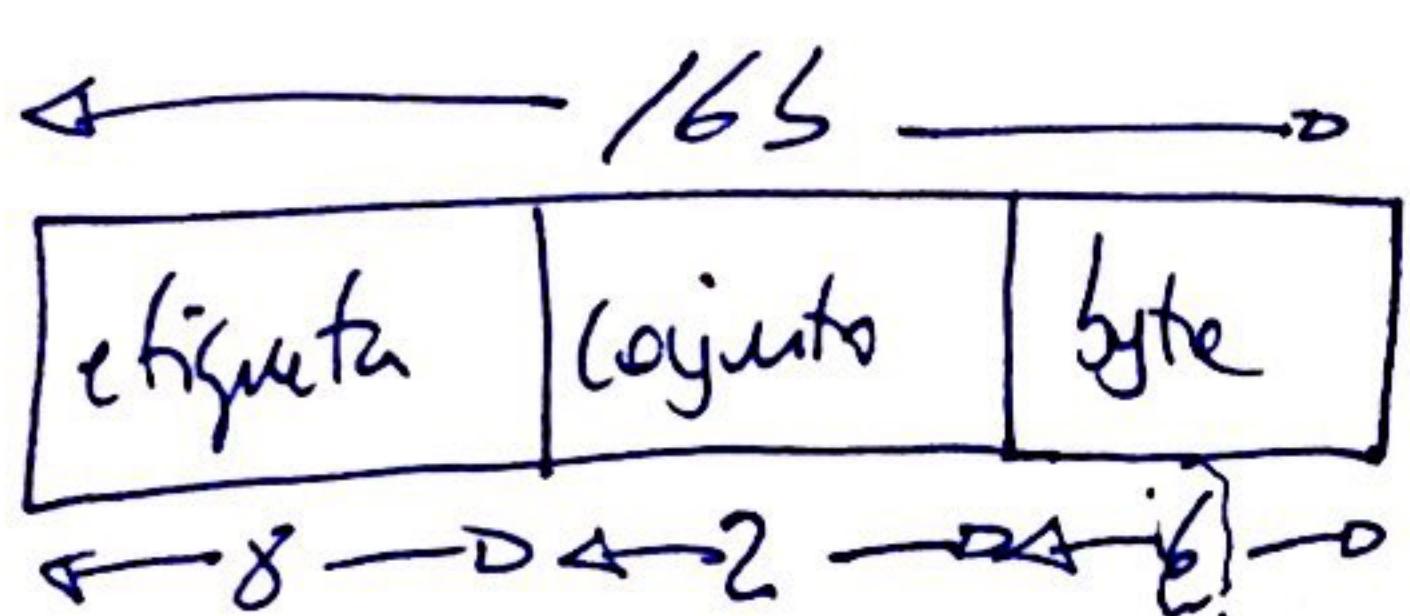
Total: 1 EXERCICIO
+ 43 ALUMNOS
44

MEMORIA

6-FB16

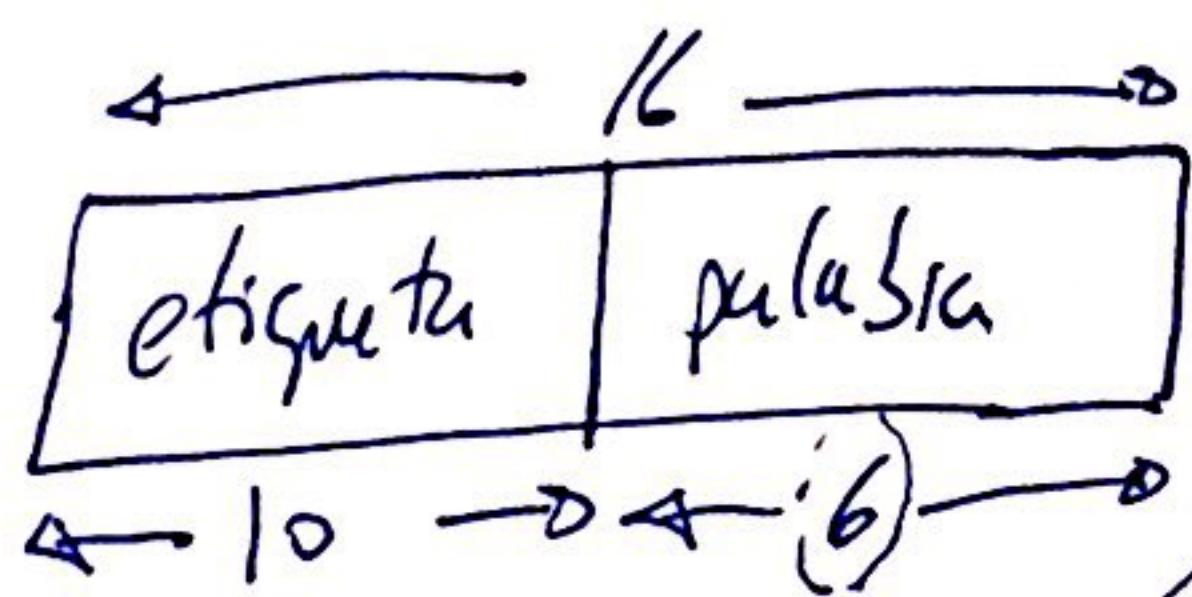
i) Directa

$$\text{línea} = 2^6 \text{ palabras} = 64 \text{ pal}$$

ii) ASOCIATIVA POR CONJUNTOS → 4 vías

$$\text{cache} = 2^{(4)} \text{ líneas} \times 2^6 \text{ pal/línea} = 1024 \text{ pal}$$

$$\text{dir MP: } 2^{(6+4+6)} = 2^{(8+2+6)} = 2^{(10+6)} = 2^{16} = 64 \text{ Kbytes}$$

iii) TOTALMENTE ASOCIATIVA

TOTAL: 1 EXERCICIO
+ 44 ACTIVIDADES
45

PÁGINA 16

Mario Rodríguez Ruiz

2026-11-24 23:45

MEMORIA

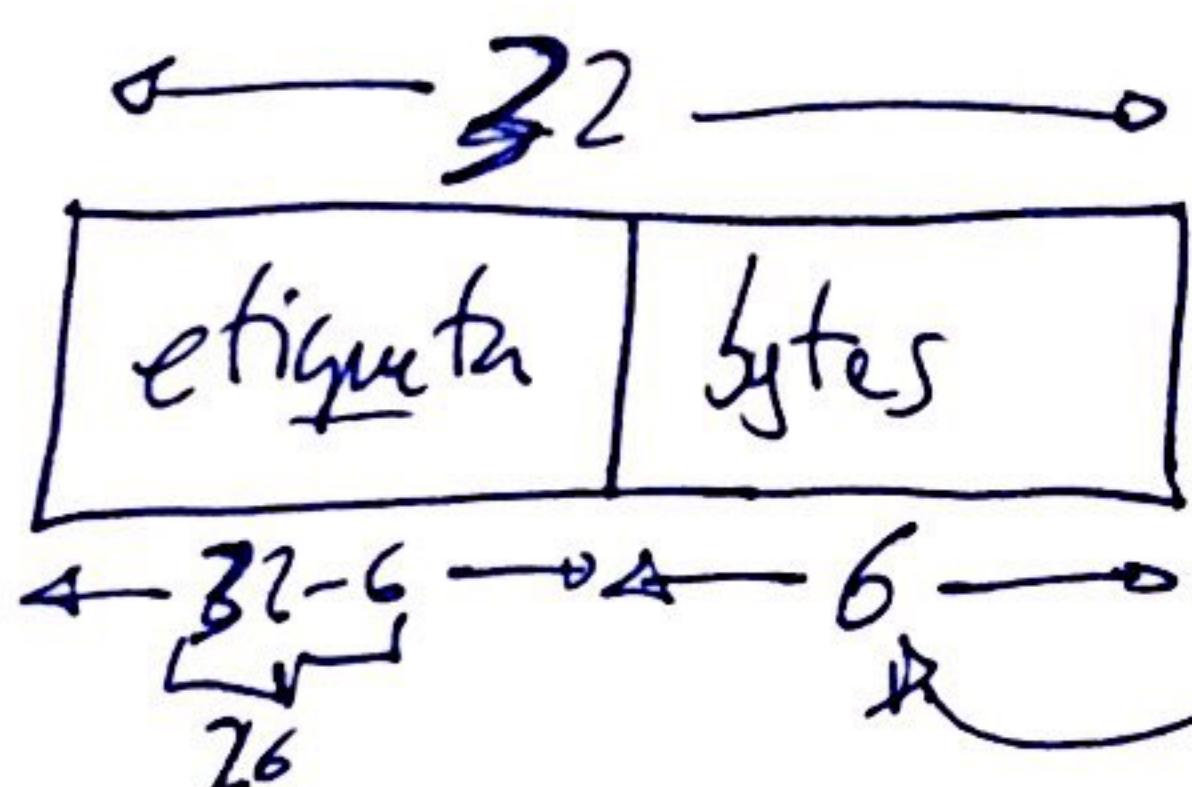
6-SEPLS

MP: $4GB = 2^{32}B \Rightarrow$ dirección de memoria de 32 bits

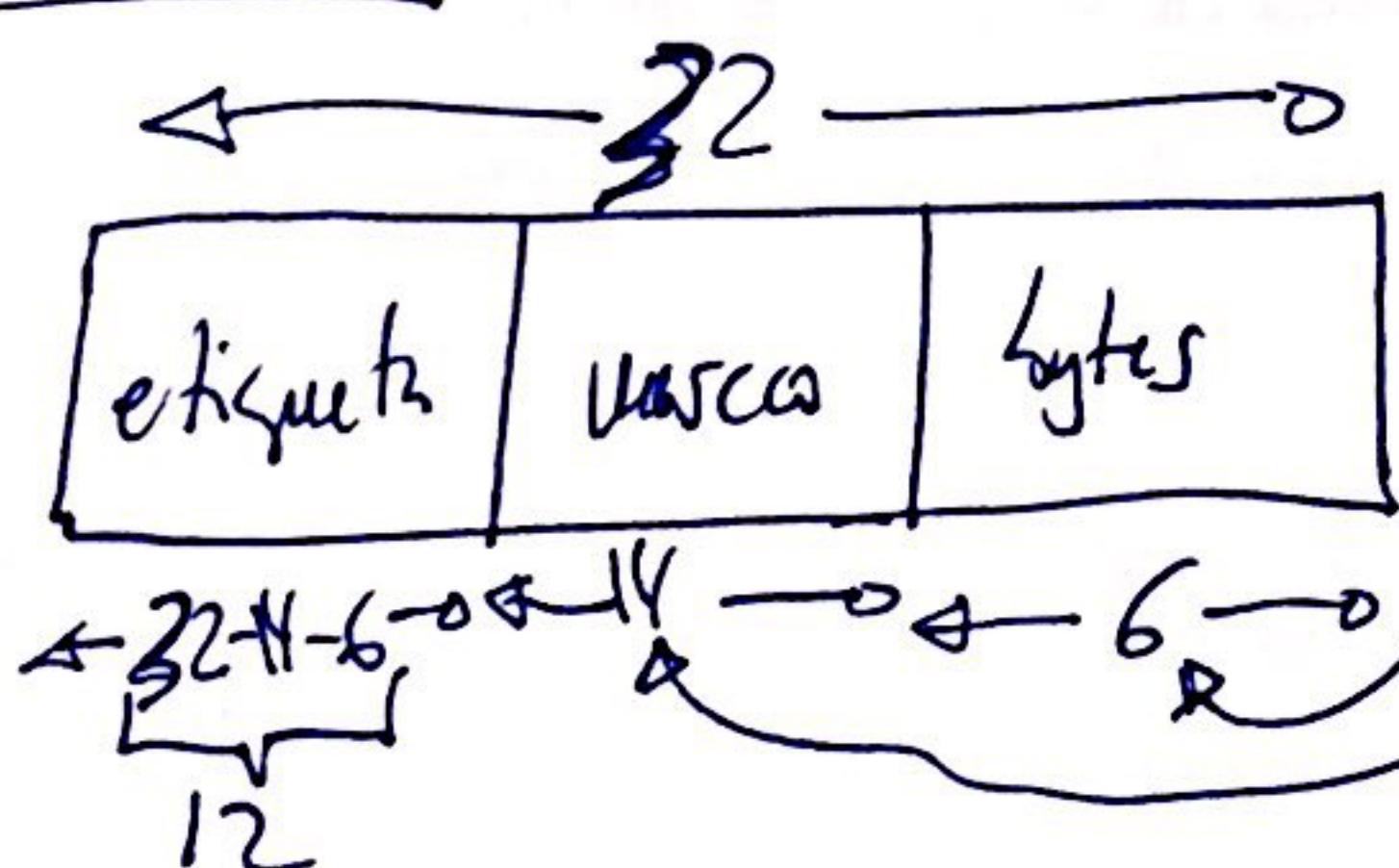
MC: $1MB = 2^{20}B$

$64B/\text{línea} = 2^6B/\text{línea} \Rightarrow 6$ bits para determinar desplazamiento

a) TRAMIENTO ASOCIATIVO

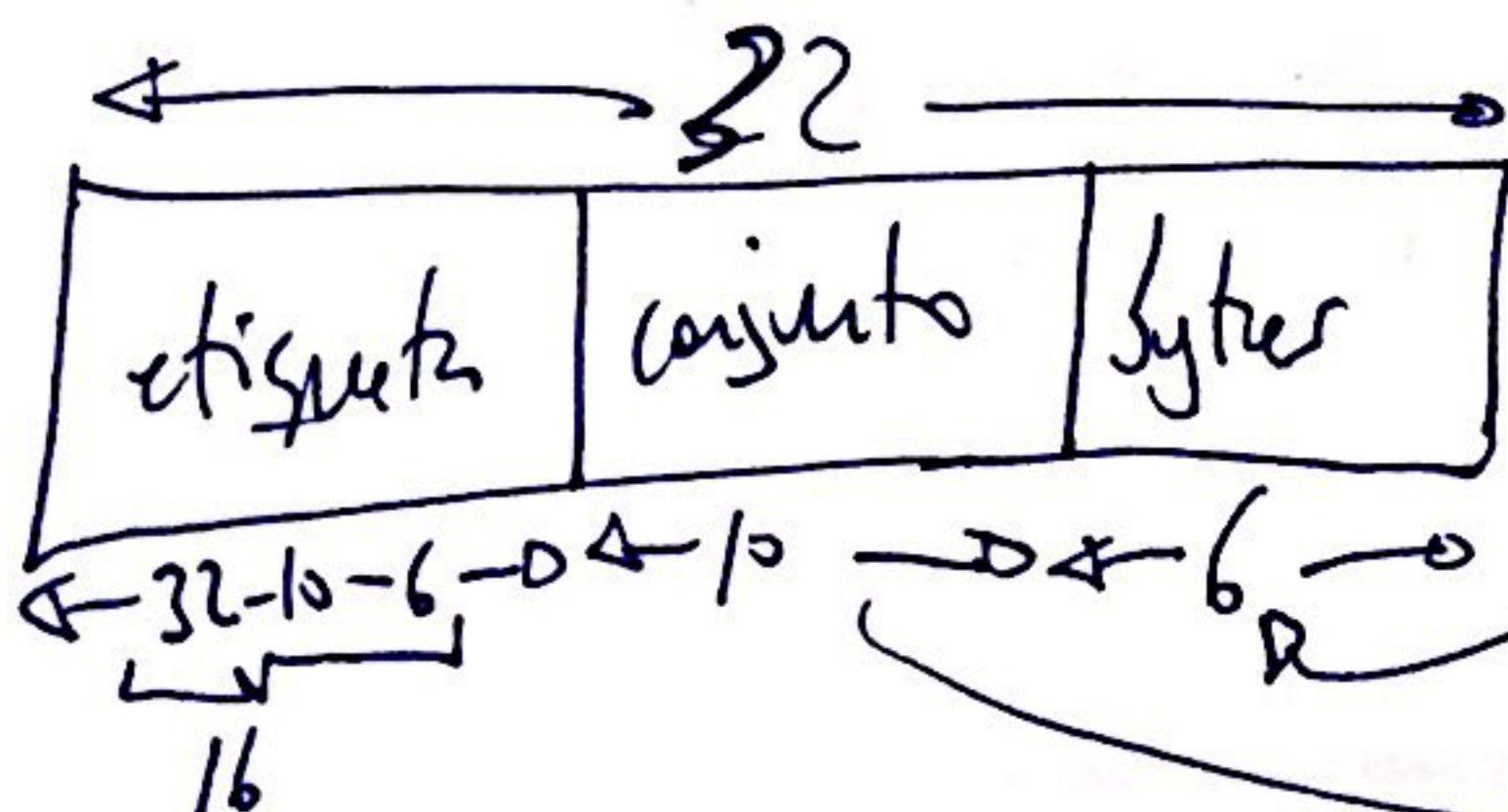


b) DIRECTA



$$\frac{2^{20} \text{ bytes en cache}}{2^6 B/\text{línea}} = 2^{14} \text{ líneas en cache}$$

c) ASOCIATIVA POR CONJUNTOS $\rightarrow 16$ vías



$$\frac{2^{14} \text{ líneas en cache}}{2^4 \text{ líneas/conjunto}} = 2^{10} \text{ conjuntos}$$

MARIO RODRÍGUEZ RUIZ

2º-GII-24737195-X

MEMORIA

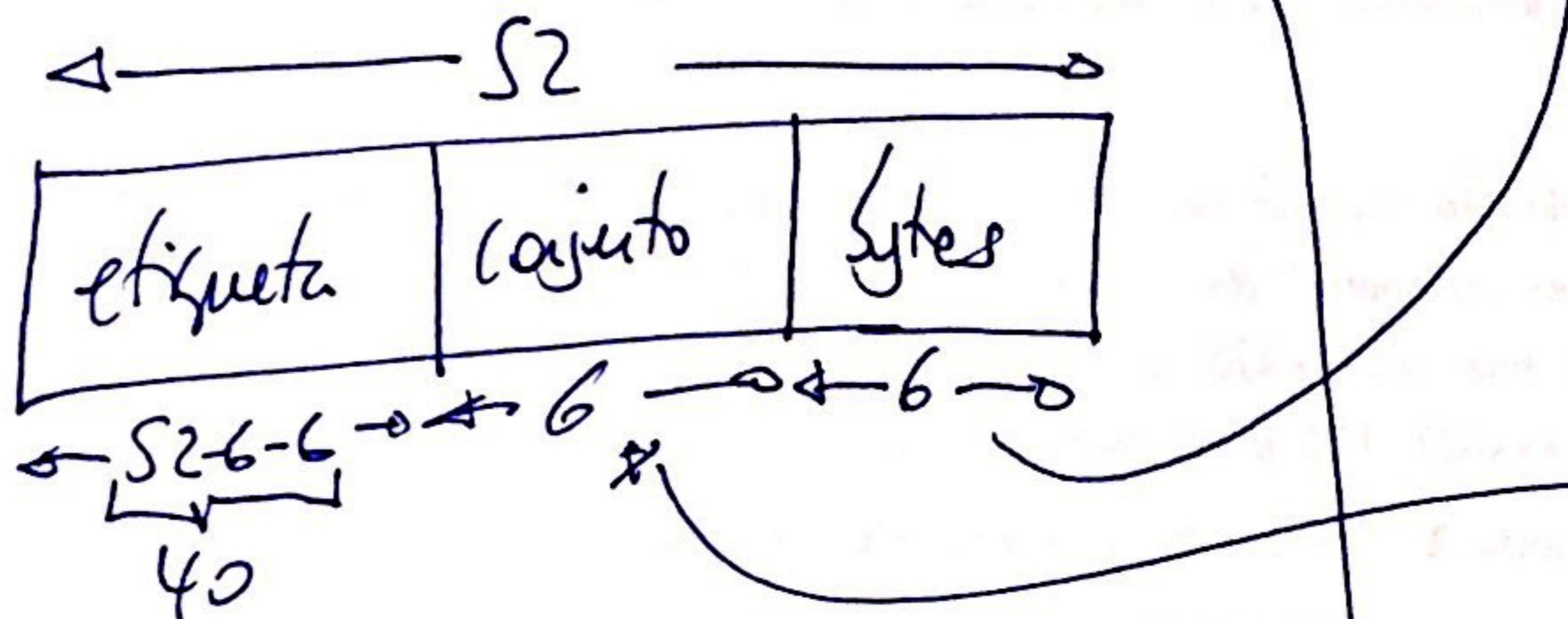
6-FEB15

MP \Rightarrow dirección de memoria de 52 bits

$$MC: 32 \text{ kB} = 2^5 \cdot 2^{10} \text{ B} = 2^{15} \text{ B}$$

64 B/línea = $2^6 \text{ B/línea} \Rightarrow 6$ bits para dirección de desplazamiento

ASOCIATIVA POR CONSULTAS - 8 vías



$$\frac{2^{15} \text{ B en cache}}{2^6 \text{ B/línea}} = 2^9 \text{ líneas en cache}$$

$$\frac{2^9 \text{ líneas en cache}}{2^3 \text{ líneas/página}} = 2^6 \text{ páginas en cache}$$

TOTAL: 1 EXERCICIO

+ 46 ALUMNOS

47

Memoria

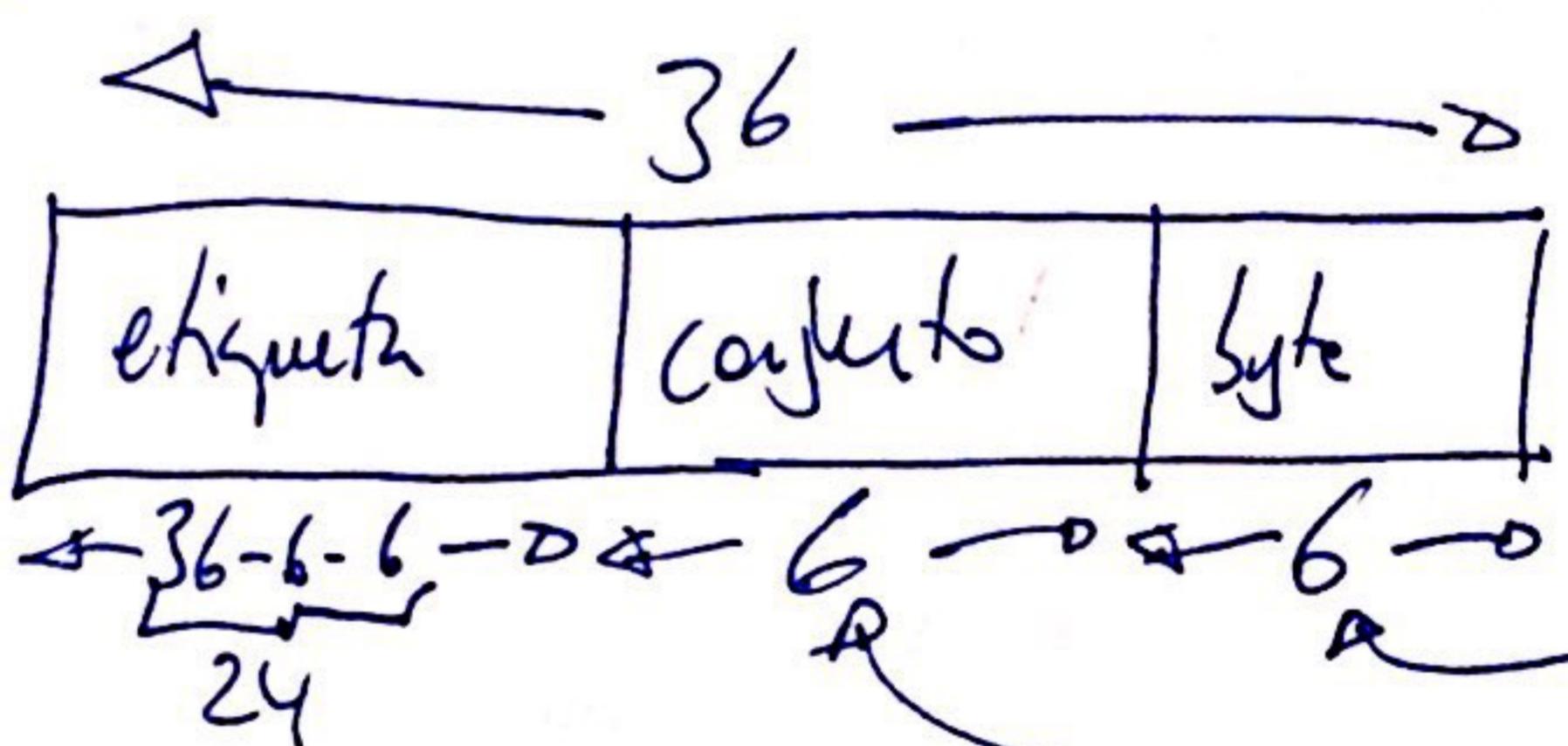
2-FEB-14

MP: $64\text{ GB} = 2^6 \cdot 2^{30}\text{ B} = 2^{36}\text{ B} \Rightarrow 36$ bits para dirección memoria

MC: $32\text{ KB} = 2^5 \cdot 2^{10}\text{ B} = 2^{15}\text{ B}$

Tam: $64\text{ B/líne} = 2^6\text{ B/líne} \Rightarrow 6$ bits para determinar desplazamiento

ASOCIACION POR CONJUNTOS - 8 vías



$$\frac{2^{15}\text{ B en cache}}{2^6\text{ B/líne}} = 2^9\text{ líneas en cache}$$

$$\frac{2^9\text{ líneas en cache}}{2^3\text{ libertad de página}} = 2^6\text{ páginas}$$

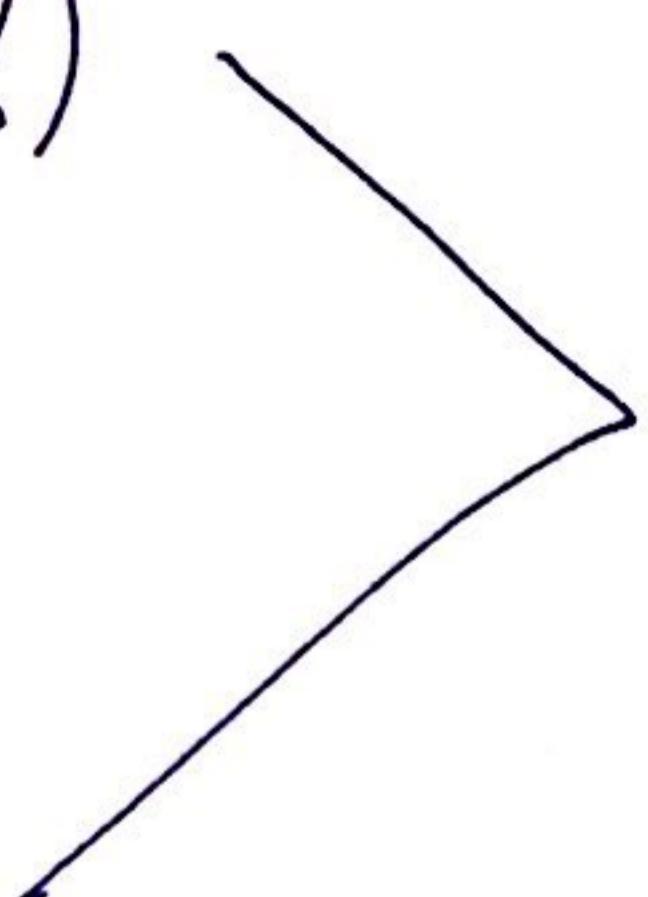
TOTAL: 1 EJERCICIO
+ 47 APROVADOS
48

A32

1-FEB14

```

int max (int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
  
```



max :

pushl %esp

movl %esp, %ebp

movl 8(%esp), %edx

movl 12(%esp), %eax

andl %edx, %eax

jge .cont

movl %edx, %eax

popl %esp

ret

a en edx

b en eax

si b < a

si falso ir a cont

a en edx

.cont :

popl %esp

ret

2-SEP14

```

int hex2bin(int c)
{
    if(c >= '0' & & c <= '9')
        return c - '0';
    if(c >= 'A' & & c <= 'F')
        return c + 10 - 'A';
    if(c >= 'a' & & c <= 'f')
        return c + 10 - 'a';
    return -1;
}
  
```

hex2bin :

push %esp

movl %esp, %ebp

cmpb \$48, 8(%esp) # compara '0' y c

jl .L2 # si c < 0, salta

cmpb \$57, 8(%esp) # compara '9' y c

jg .L2 # salta si: '9' > c

movl 8(%esp), %eax # mete c en %eax

subl \$48, %eax # resta '0' a c

.L2 :

cmpb \$65, 8(%esp)

jl .L4 # compara 'A' y c

cmpb \$70, 8(%esp)

jg .L4 # compara 'E' y c

movl 8(%esp), %eax

subl \$55, %eax # guarda c en eax

jmp .L3 # resta 10-'55' a c

.L3

popl %esp

PÁGINA 20

TOTAL : 2 EJERCICIOS

+ 48 ATRIBUICIONES

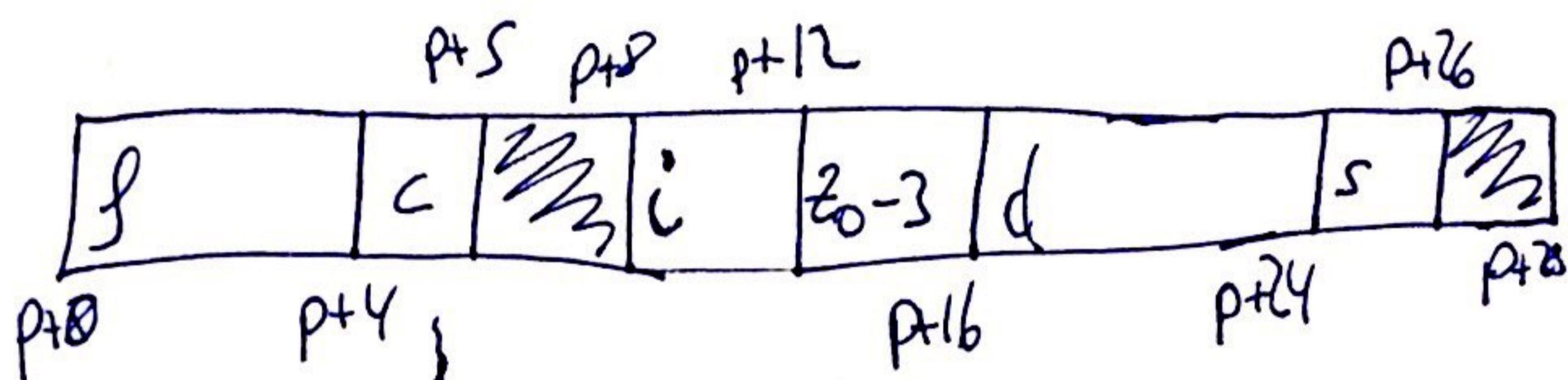
50

DISTRIBUCIÓN DE ESTRUCTURAS

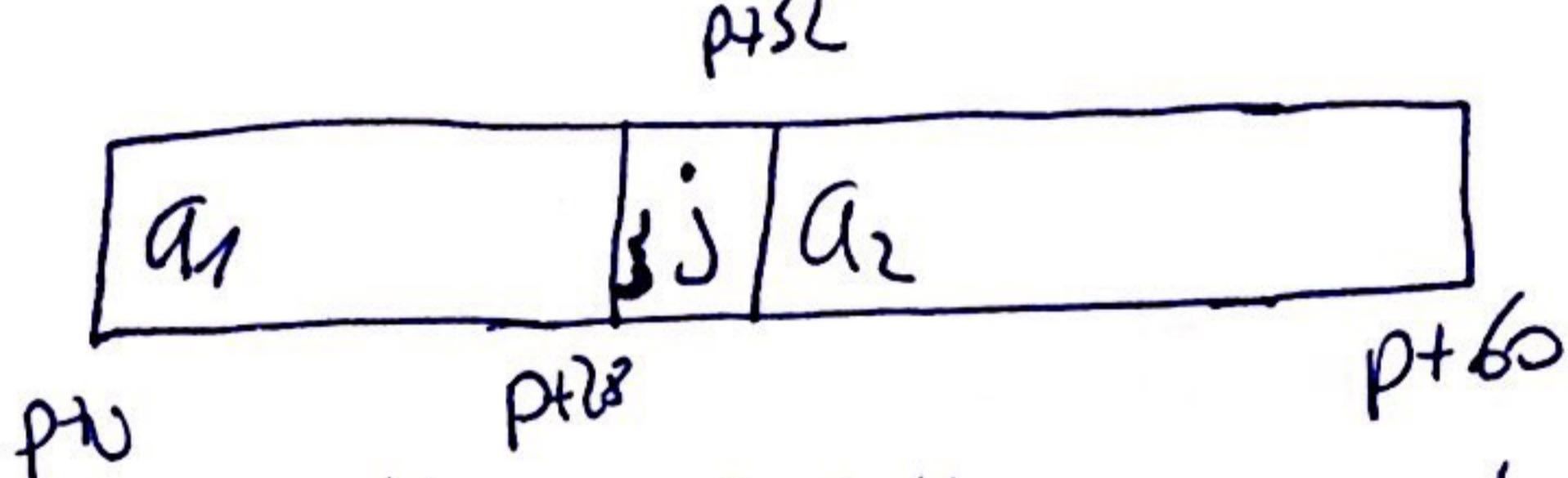
13-FEB16

a) IA32 $k=4$

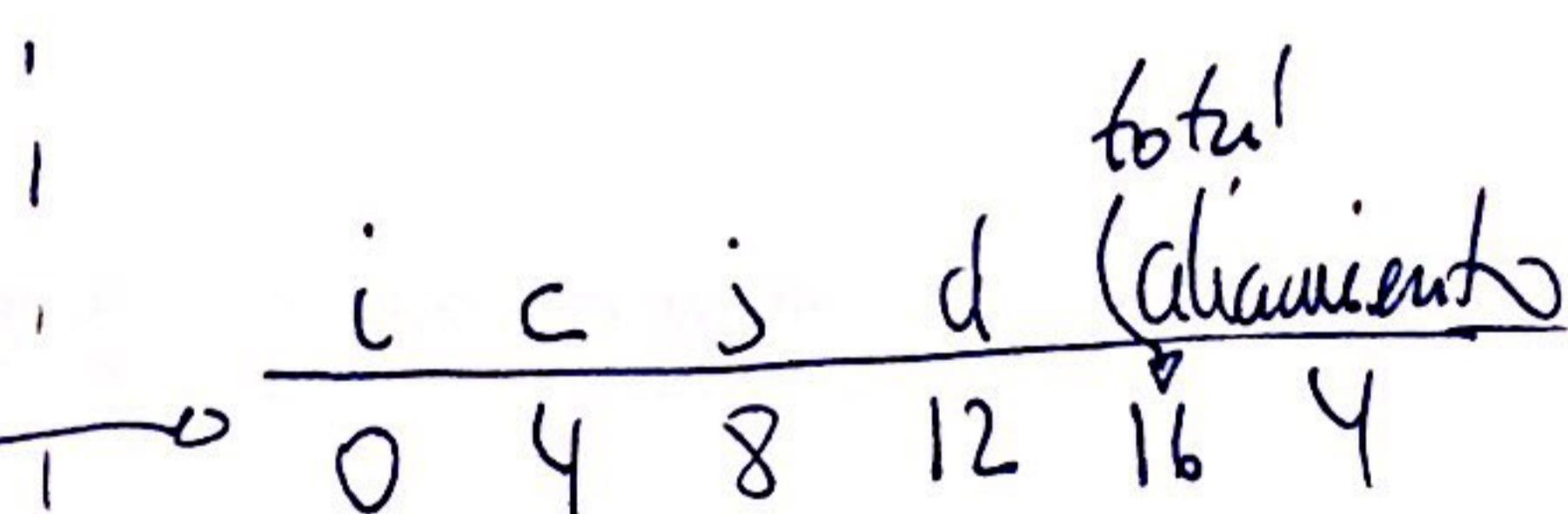
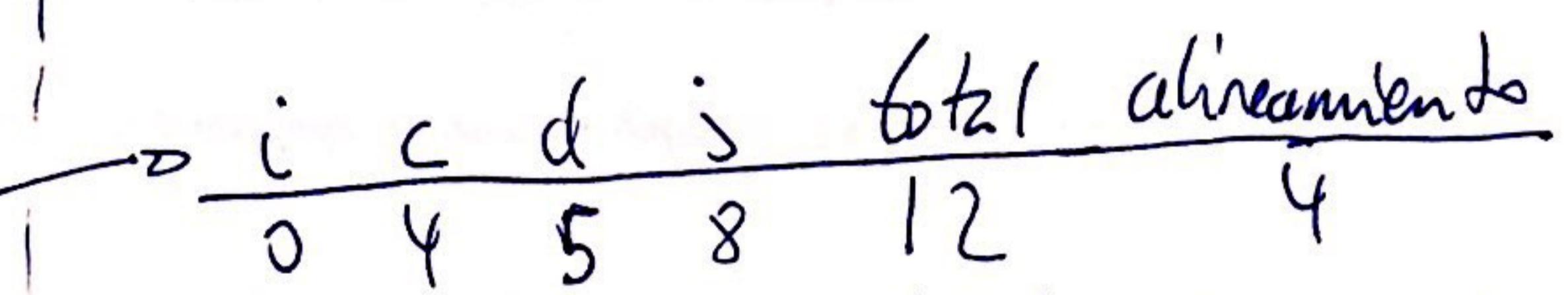
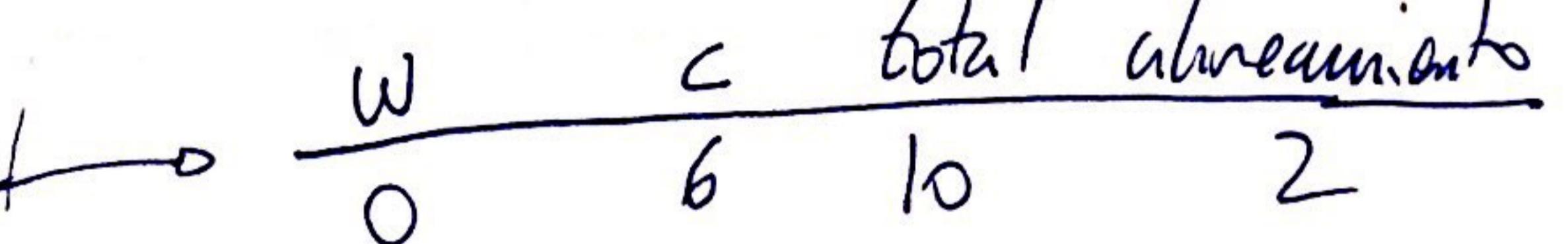
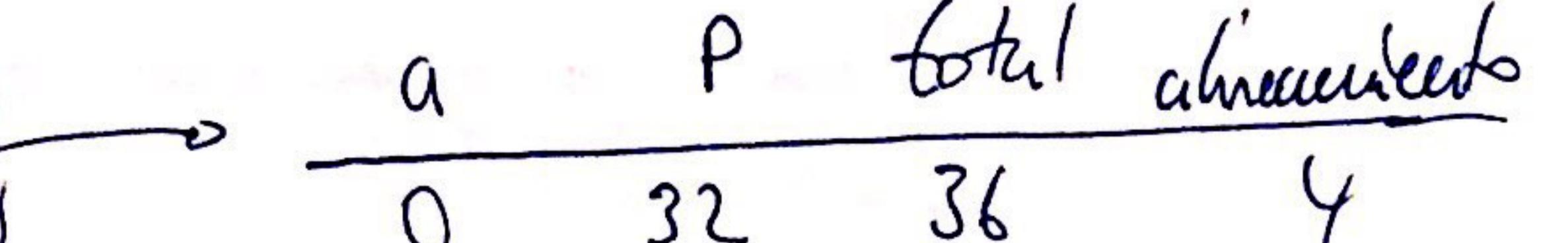
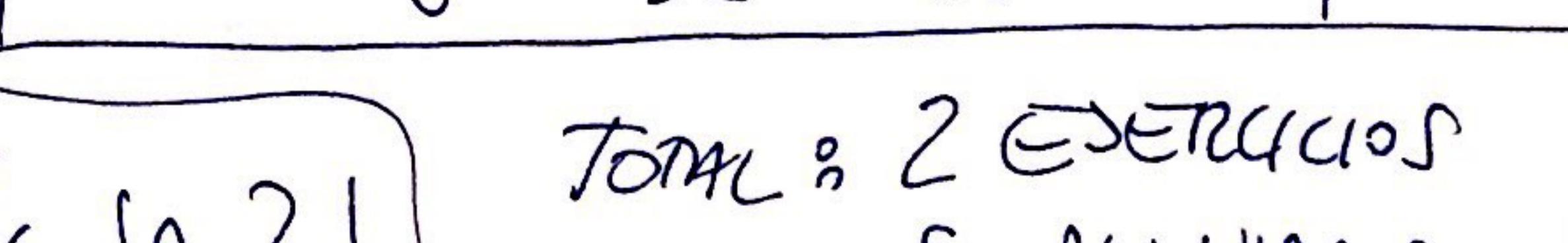
```
struct a {
    float f; → 4B
    char c; → 1B
    int i; → 4B
    char z[4]; → 4B
    double d; → 8B
    short s; → 2B
};
```

Por tanto, "struct a" usa 28 bytes

b)

"struct b" usa 60 bytes

12-FEB13

A. struct P1{int i; char c;
int j; char d;}; $k=4$ B. struct P2{int i; char c; char d;
int j;}; $k=4$ C. struct P3{short w[3]; $k=2$
char c[3];}; $k=2$ D. struct P5{short w[3]; $k=4$
char *c[3];}; $k=4$ E. struct P5{struct P1 a[2];
struct P2 *p;}; $k=4$ 

TOTAL: 2 EXERCICIOS

+ 50 AWALUACIONES

MARIO RODRIGUEZ PINT

20-GIF-1473745-8

Diseño de Memoria

5-FEB15

SRAM1:

$$A_0 - A_3 \Rightarrow 14 \text{ bits dir} \Rightarrow 2^{14} = 16 \text{ Kpal}$$

$$D_0 - 7 \Rightarrow \text{pal} = 8 \text{ bits} \Rightarrow 16 \text{ KB} = 16 \text{ K} \times 8$$

$$2^{15} = 32 \text{ K}$$

CS $\Rightarrow A_{15} - A_{14} = 10 \Rightarrow$ dirr CPU desde 1000 0000 0000 0000 hex 0x8000
hasta 1011 1111 1111 1111 hex 0xBFFF

SRAM2:

$$A_0 - A_3 = 16 \text{ Kpal}$$

$$D_0 - 7 = 16 \text{ KB} \Rightarrow 16 \text{ K} \times 8$$

CS $\Rightarrow A_{15} - A_{14} = 01 \Rightarrow$ dirr CPU desde 0000 0000 0000 0000 hex 0x4000
0111 1111 1111 1111 hex 0x7FFF

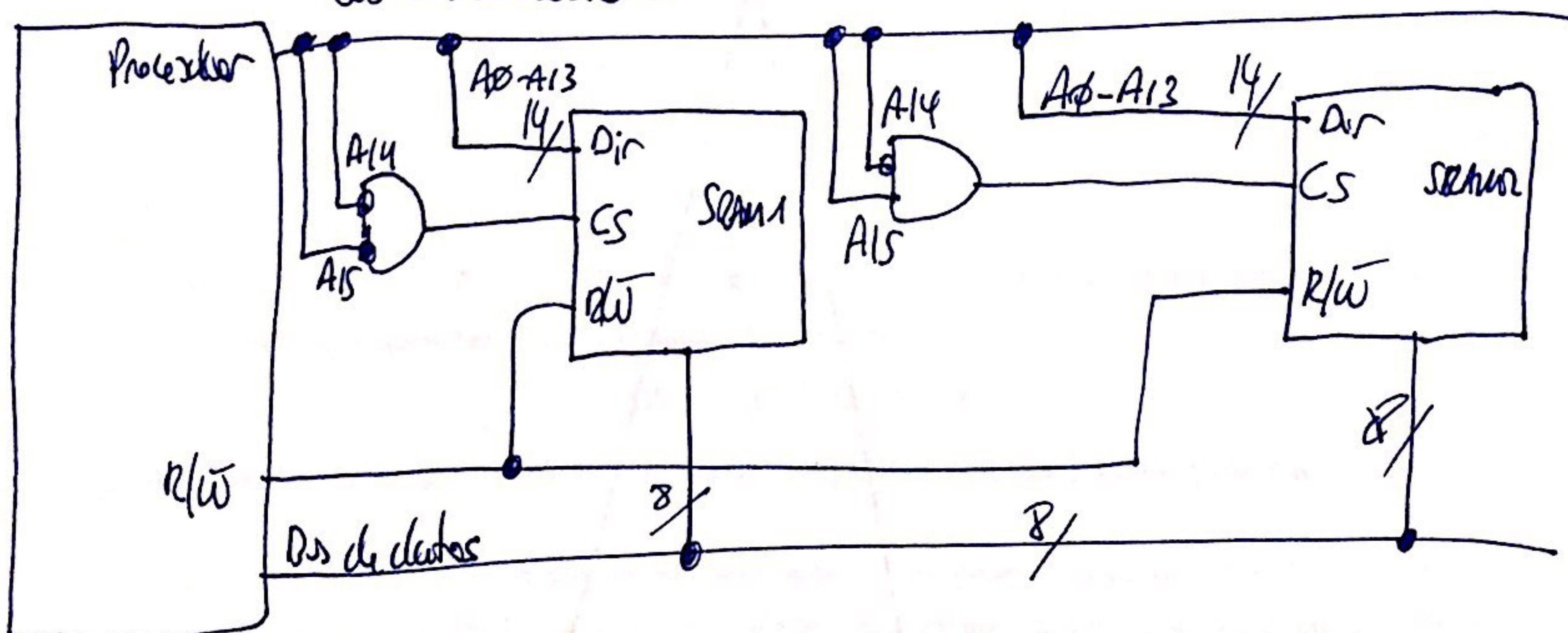
$$2^{15} + 2^{14} - 1 = 48 \text{ K} - 1$$

$$2^{14} = 16 \text{ K}$$

$$2^{15} - 1 = 32 \text{ K} - 1$$

5-SEP16

Bus de direcciones



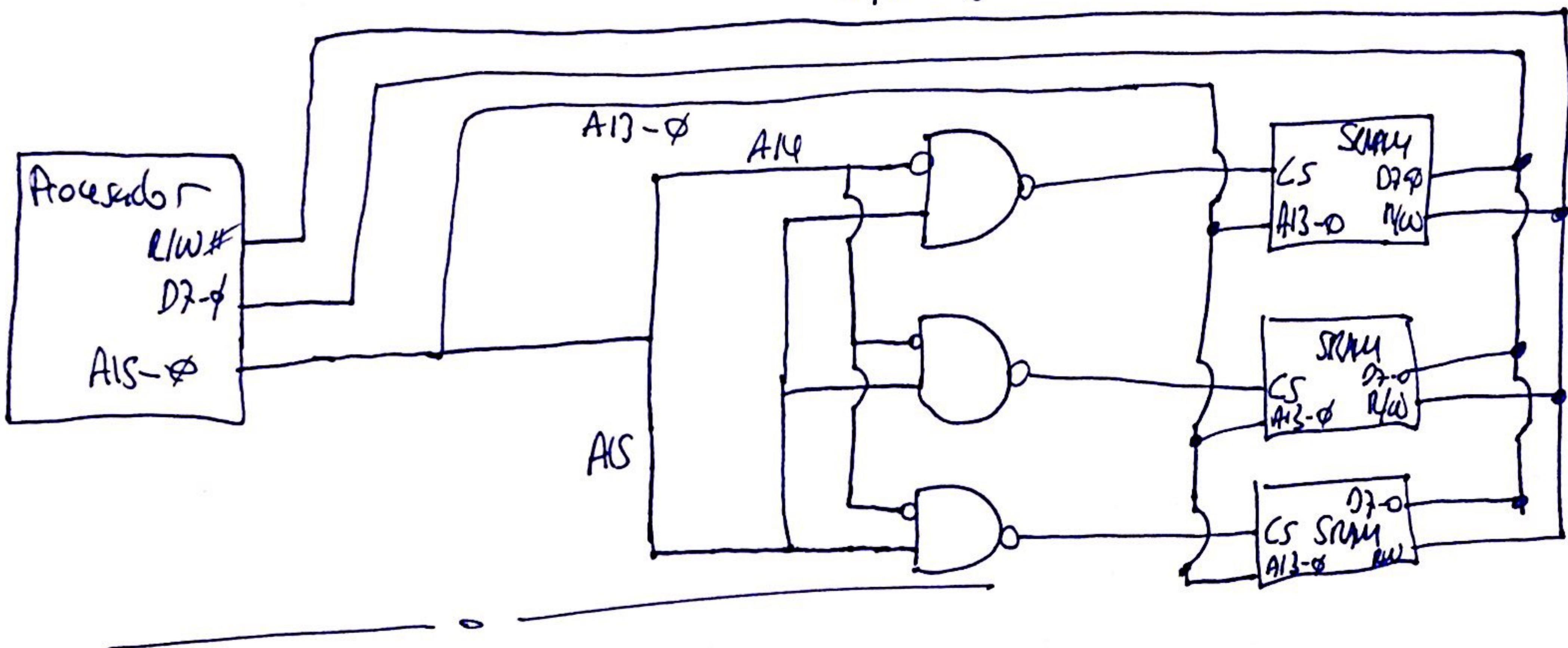
TOTAL: 2 EJERCICIOS
+ 52 ACUMULADOS
54

ENTRADA/SALIDA MEMORIA

14-FEB16

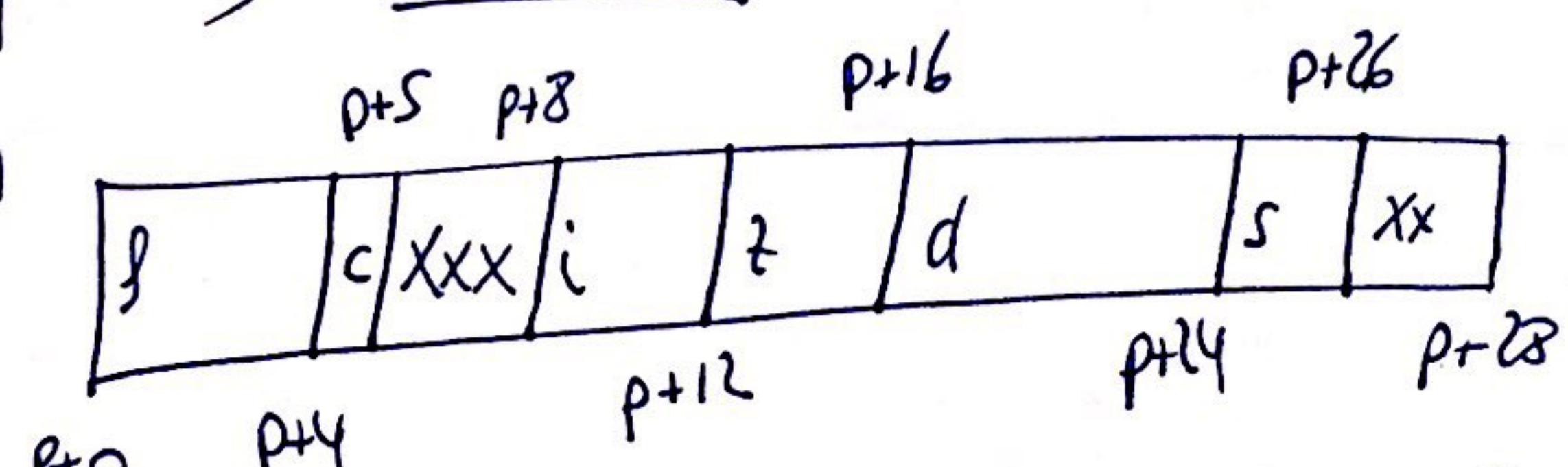
$$16K \times 8 = 16KB \rightarrow 16K \text{ pal} = 2^{14}$$

$\rightarrow \text{pal} = \text{byte}$

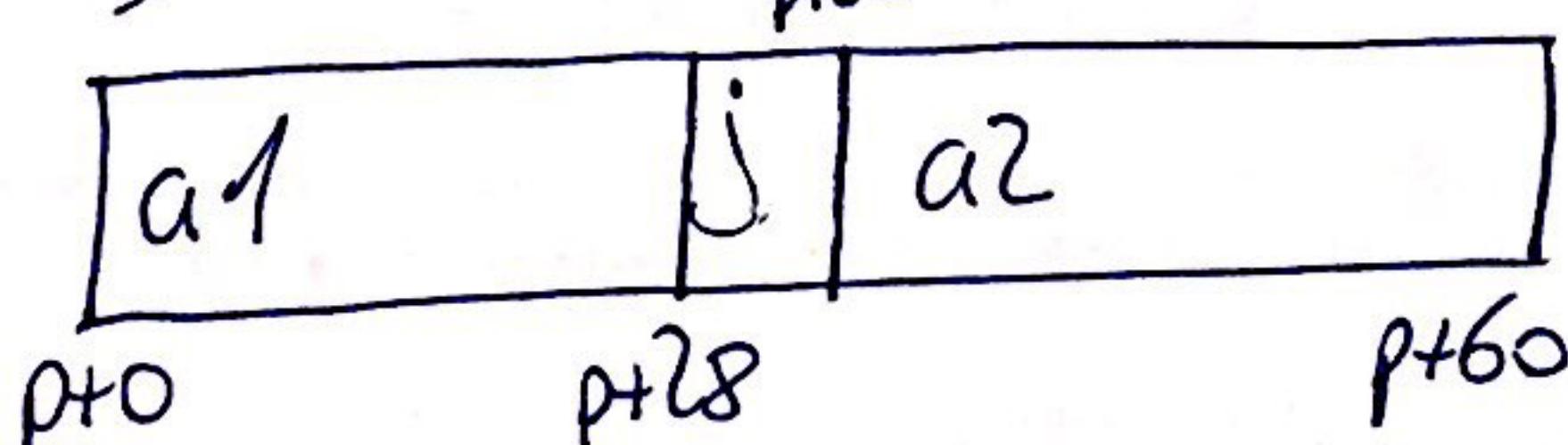
ESTRUCTURASIA32

```
struct a {
    float f;           → 4B
    char c;           → 1B
    int i;            → 4B
    char z[4];         → 4B
    double d;          → 8B
    short s;          → 2B
};
```

```
struct b {
    struct a a1;      → 28B
    int j;             → 4B
    struct a a2;      → 28B
};
```

a) struct a

Por tanto, struct a tiene 28 Bytes

b) struct b

Fácilmente, struct b tiene 60 bytes

BINL: 2 EJECUCIONES
+ 54 ACUMULADORES
S6

UNIDADE DE CONTROL5-F_{IG}

$\text{CALL} : z = SP - 4$
 $SP = z; MAR = z$
 $MBR = PC; Y = PC$
 $m[MAR] = MBR; z = PC + Y$
 $PC = z; \text{goto } FETCY$

$\text{RET} : MAR = SP; z = SP + 4$
 $MBR = m[MAR]; SP = z$
 $PC = MBR; \text{goto } FETCY$

4-S₁₆

$FETCY : MAR = PC; z = PC + 4$
 $PC = z; Y = z; MDR = m[PC + 4]$
 $IR = MDR$
 $\text{goto } f(IR);$

$\text{BRANCH} : z = Y + 1_R$
 $PC = z; \text{goto fetch}$

ENSAMBLEO IA32

```
int max(int a, int b){  
    if(a > b)  
        return a;  
    else  
        return b;  
}
```

```
void maxV(int *v1, int *v2,  
          int *v3, int N){
```

```
int i;  
for(i=0; i<N; i++)  
    v3[i] = max(v1[i], v2[i]);
```

```
call max  
add $8, %esp # recuperar pil max  
mov 16(%esp), %edx # EDX = V3  
mov %eax, (%edx, %esi, 4) # V3[EI]  
inc %esi # i++  
cmp 20(%esp), %esi # i < N?  
jne .loop # i < N - repita
```

```
max:  
    push %ebp  
    mov %esp, %ebp # ajuste unico  
    mov 8(%esp), %eax # eax = a  
    mov 12(%esp), %edx # edx = b  
    cmp %edx, %eax # a > b?  
    jg .finmax # a > b - nada  
    mov %edx, %eax # a <= b - return b  
    .finmax:  
    pop %ebp # Destruir marco  
    ret  
  
maxV:  
    # a: ajuste marco  
    push %esi  
    push %edi  
    push %ebx  
    mov 8(%esp), %esi # esi = V1  
    mov 12(%esp), %edi # edi = V2  
    cmp $0, 20(%esp) # N = 0?  
    jle .finmaxV # N <= 0 - acaba  
    mov $0, %esi # for i = 0, EBX = i  
  
.loop:  
    push(%edi, %edx, 4) # arg2º v2[i] Push M/M  
    push(%esi, %edx, 4) # arg1º v1[i] Push M/M
```

finmaxV:

```
pop %edx  
pop %edi  
pop %ebx  
ret
```

PÁGINA 24

TOTAL: 3 EJERCICIOS

+ 56 ACUMULADOS

S9

Diseño de memoria

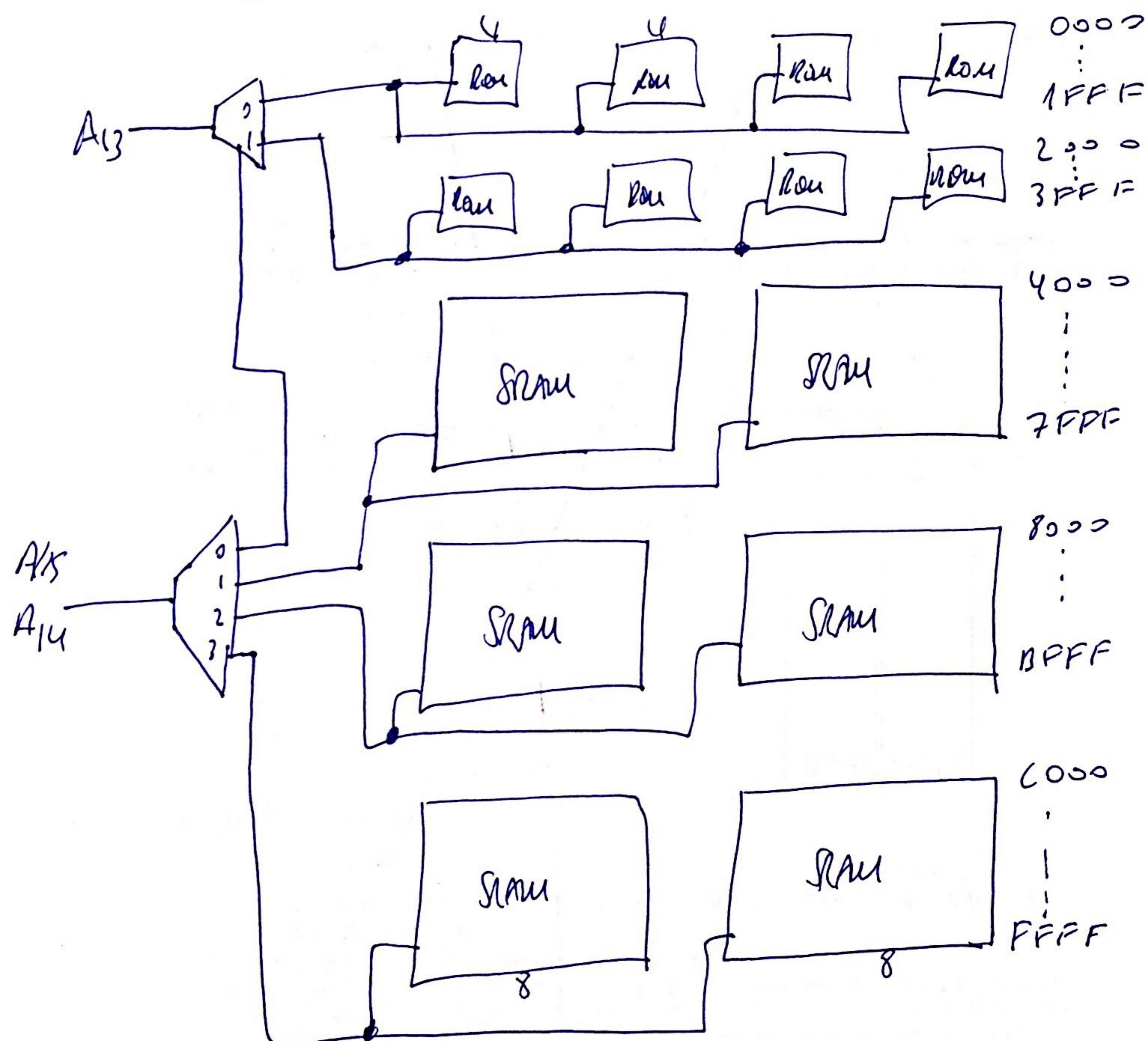
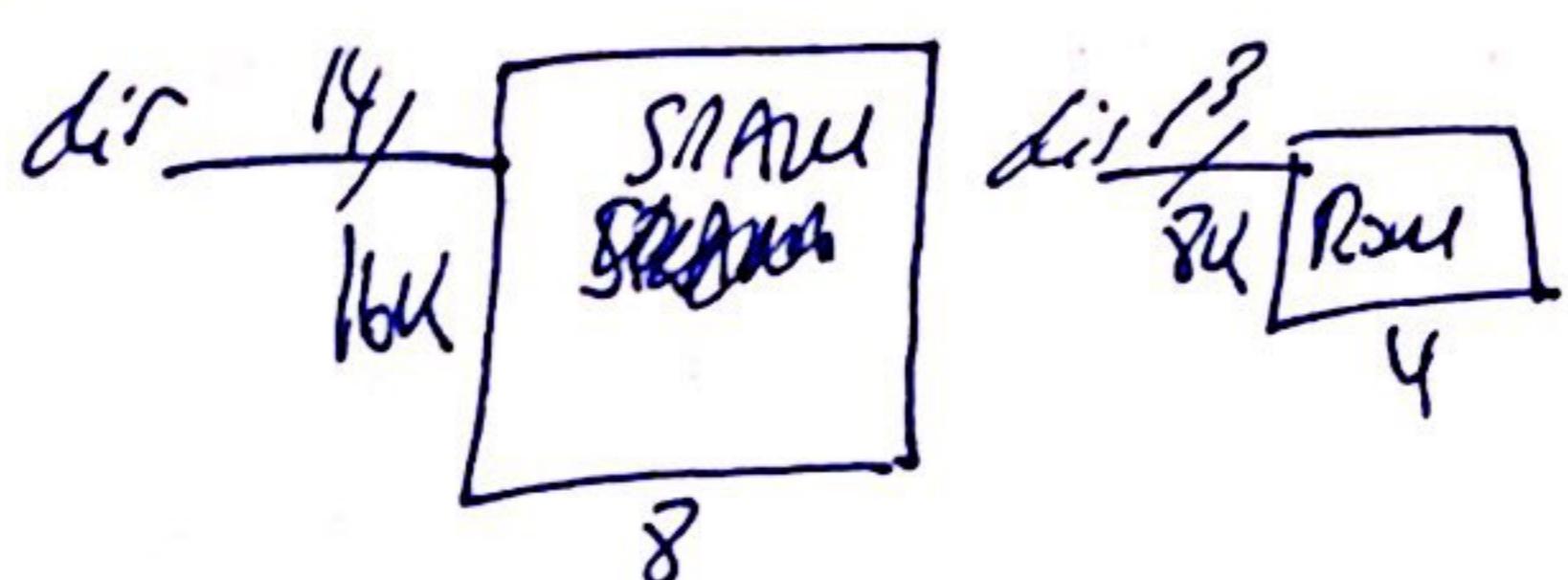
Mario Rodríguez Ruiz

2º C GIT-747324N-X

CPU → 16 bit latency address

SRAM de $16 \times 8 \rightarrow 0x4000-0xFFFF$

Rom de $8K \times 4 \rightarrow 0x0000-0x3FFF$



TOTAL: 1 EXERCICIO
+ SQ AWNLADS
— 60

(W0A) DE CONTROL

MARIO RODRIGUEZ DW17

2023-24-74737195-X

3-FE

FETC4: $\text{MAR} \leftarrow \text{PC}[7:0]$; $A \leftarrow \text{PC}$
 $\text{IR} \leftarrow M[\text{MAR}]$
 $\text{PC} \leftarrow A+1$; goto f(IR)

ADD: $A \leftarrow r$

$\text{MAR} \leftarrow \text{IR}[7:0]$
 $B \leftarrow M[\text{MAR}]$
 $A \leftarrow A+B$
 $M[\text{MAR}] \leftarrow A$; goto Fetch

6-FE

Captación y decodificación

Fetch: $\text{MAR} \leftarrow \text{PC}$; $Z \leftarrow \text{PC}+1$
 $\text{MBR} \leftarrow M[\text{MAR}]$; $\text{PC} \leftarrow Z$
 $\text{IR} \leftarrow \text{MAR}$
 goto f(IR);

7-G-Seq(3)

FETC4: $\text{MAR} \leftarrow \text{PC}$; $Z \leftarrow \text{PC}+4$
 Read; $\text{PC} \leftarrow Z$
 $\text{MBR} \leftarrow M[\text{IR}]$
 $\text{IR} \leftarrow \text{MBR}$
 goto f(IR);

ADDR: $Y \leftarrow \text{Rsrc}$
 $Z \leftarrow Y + \text{Rdst}$
 $\text{Rdst} \leftarrow Z$, goto FETC4;

8 Rel

FETC4: $\text{MAR} \leftarrow \text{PC}$; $Z \leftarrow \text{PC}+4$
 $\text{PC} \leftarrow Z$; if $Y \leftarrow Z$; $\text{MBR} \leftarrow M[\text{MAR}]$
 $\text{IR} \leftarrow \text{MBR}$
 goto f(IR);

BRANC4: $Z \leftarrow Y + \text{IR}$

$\text{PC} \leftarrow Z$
 goto FETC4;

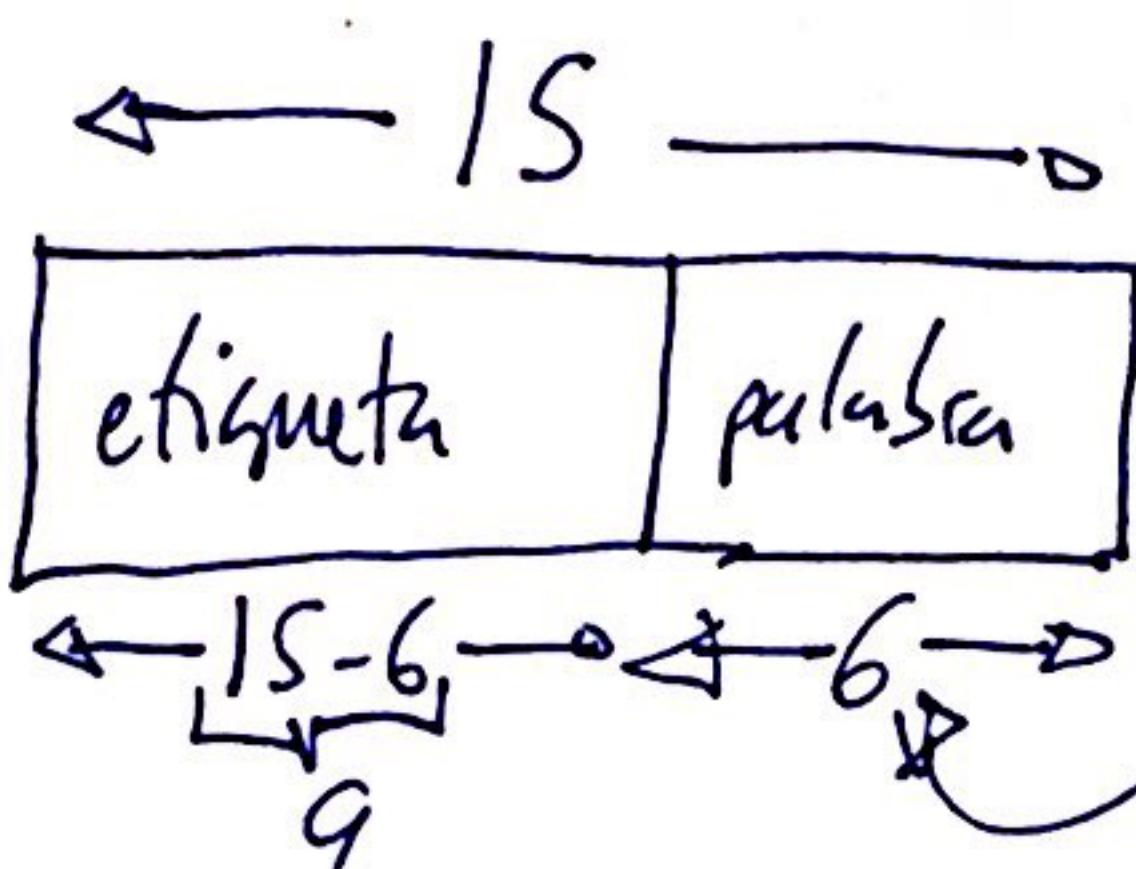
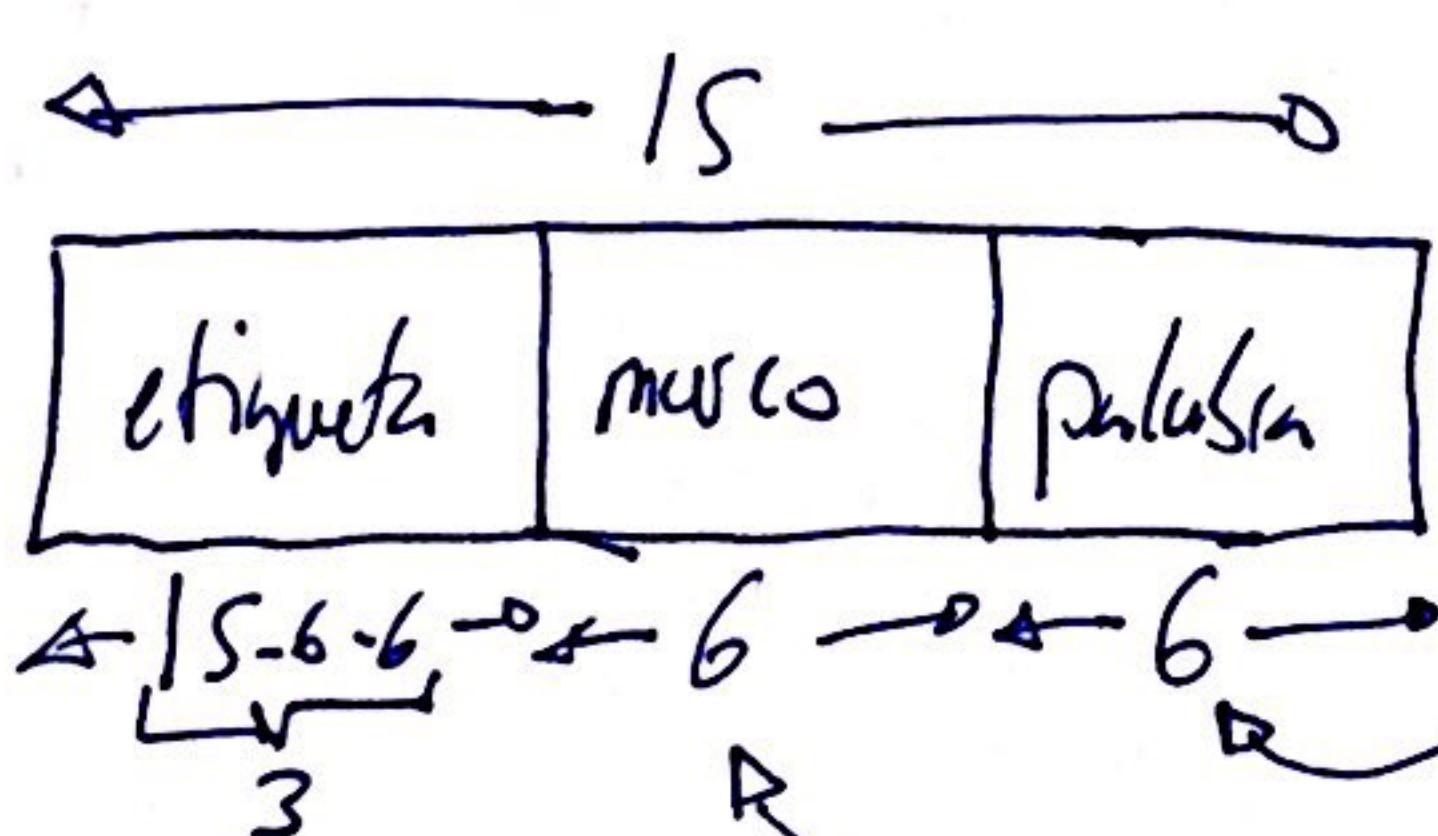
TOTAL: 3 EJERCICIOS
+ 60 ALGORITMOS
63

MEMORIA CACHE

MP: 32 K palabras $\Rightarrow 2^5 \cdot 2^{10} \text{ pal} = 2^{15} \Rightarrow 15$ bits para direcciones

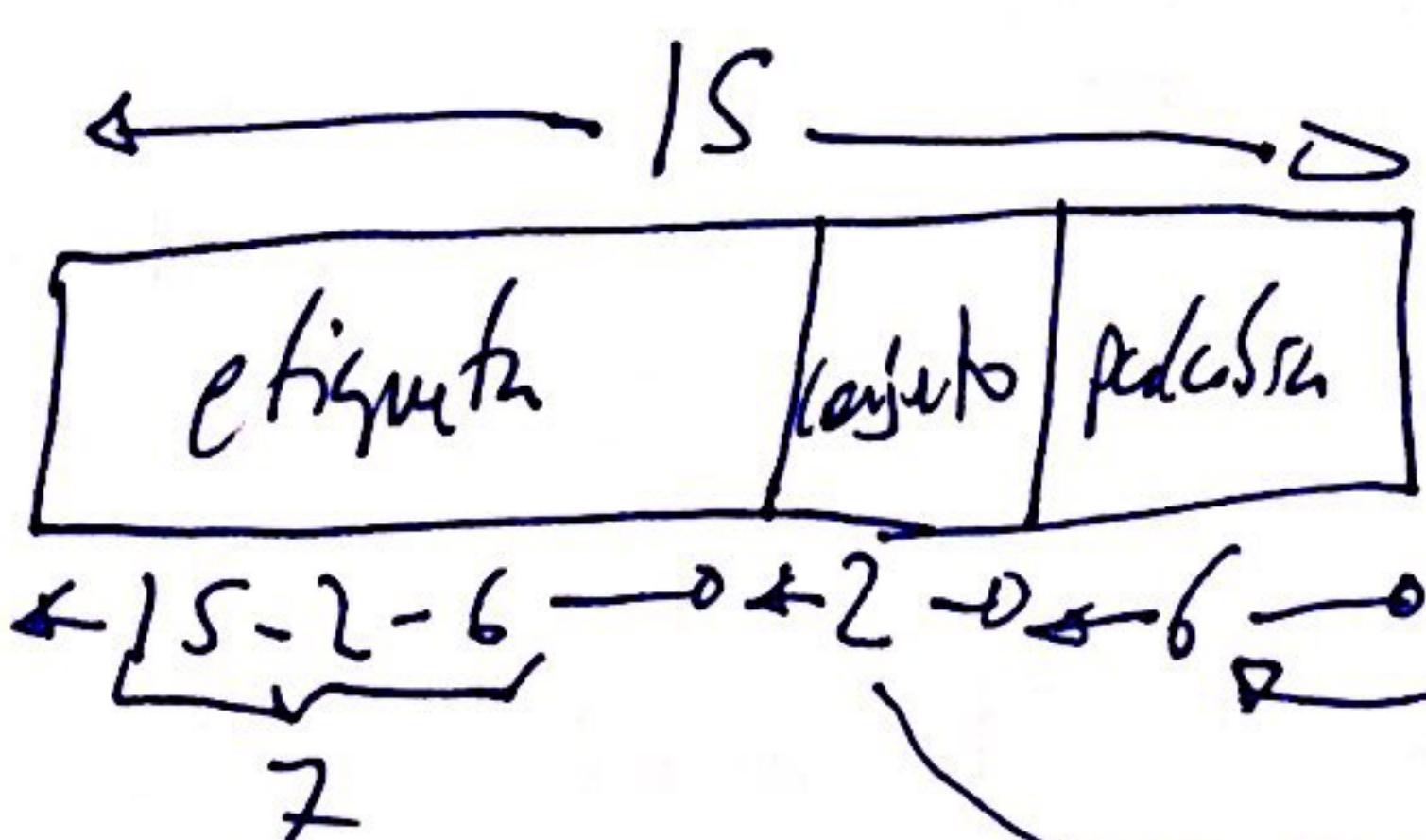
MC: 4 K palabras $\Rightarrow 2^2 \cdot 2^{10} \text{ pal} = 2^{12} \text{ pal}$

Tam. Bloque: 64 palabras $= 2^6 \text{ pal/línea}$

A. TOTALMENTE ASOCIATIVAB. CORRESPONDENCIA DIRECTA

$$\frac{2^{12} \text{ pal en Cache}}{2^6 \text{ pal/línea}} = 2^6 \text{ líneas}$$

(1) líneas cache

C. ASOCIATIVA POR CONJUNTOS $\rightarrow 16$ vías

$$\frac{2^6 \text{ líneas cache}}{2^4 \text{ líneas/conjunto}} = 2^2 \text{ conjuntos}$$

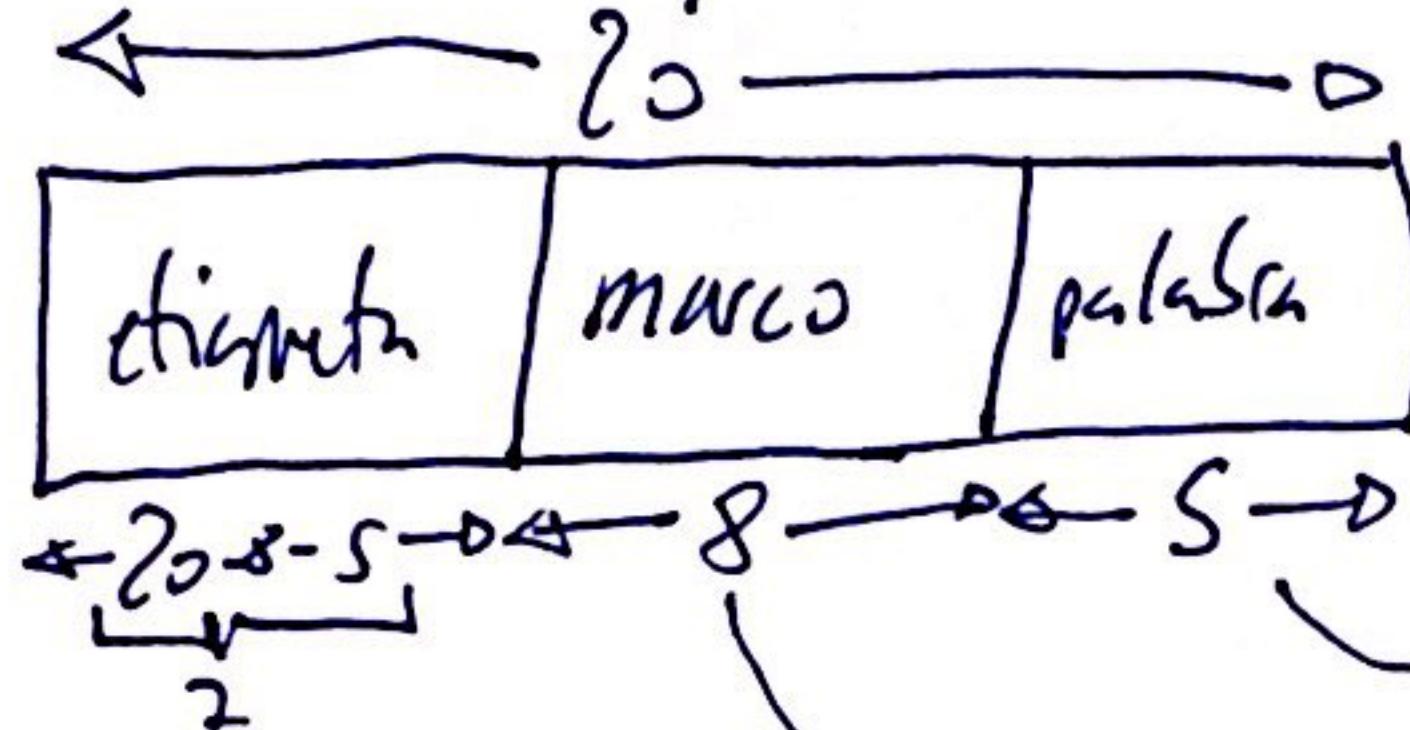
TOTAL: 1 EJERCICIO
+ 63 ACUMULADOS
64

Memoria Cache

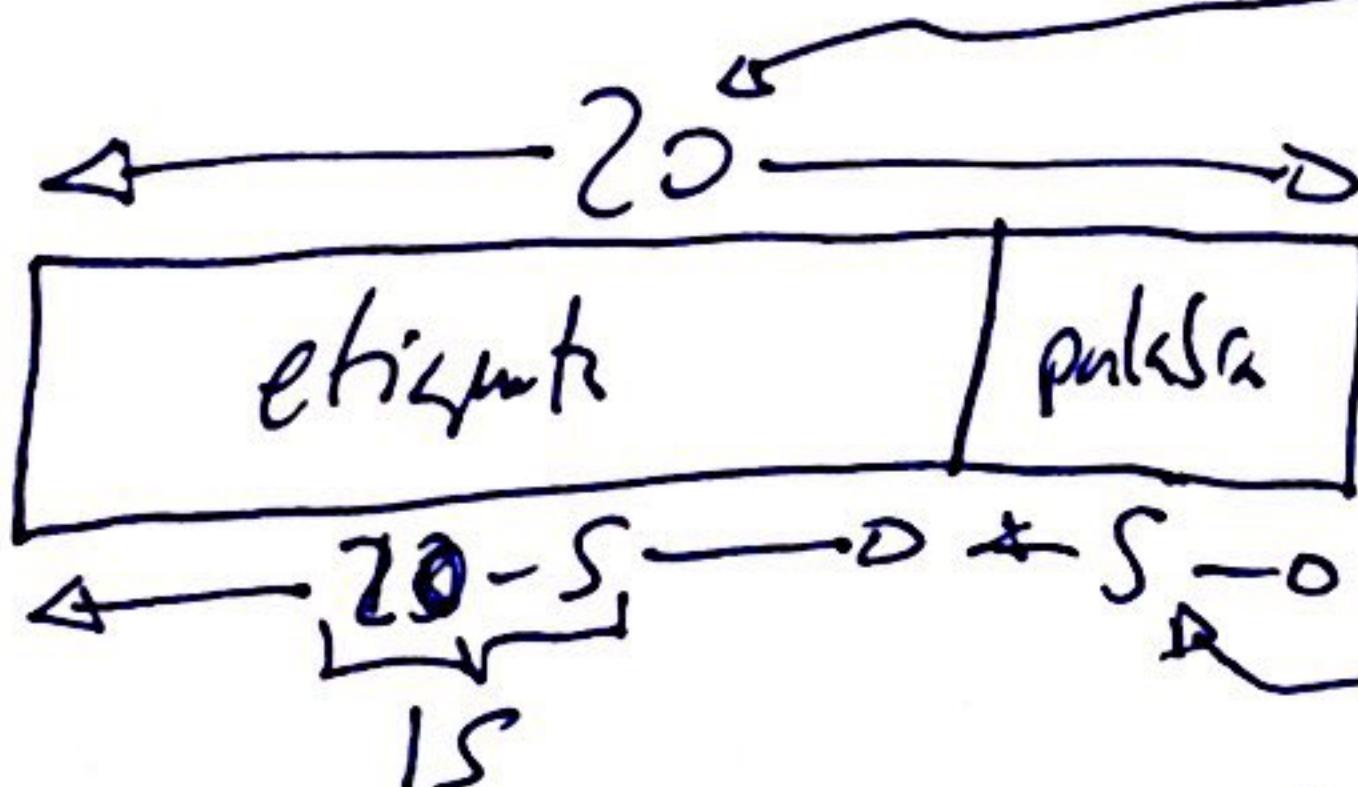
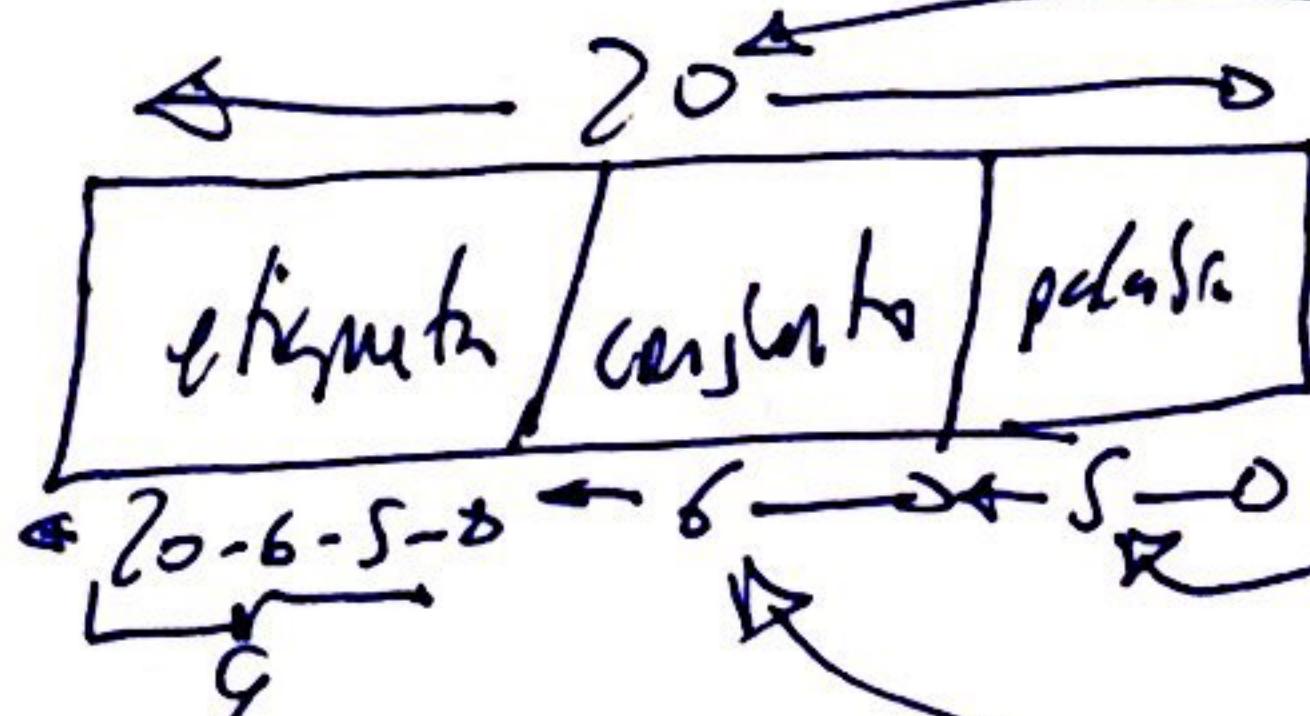
MP: 1M palabras = 2^{20} palabras = $2^{20} \cdot 8$ bits para direcciones

MC: 8K palabras = $2^3 \cdot 2^{10}$ palabras = 2^{13} pal

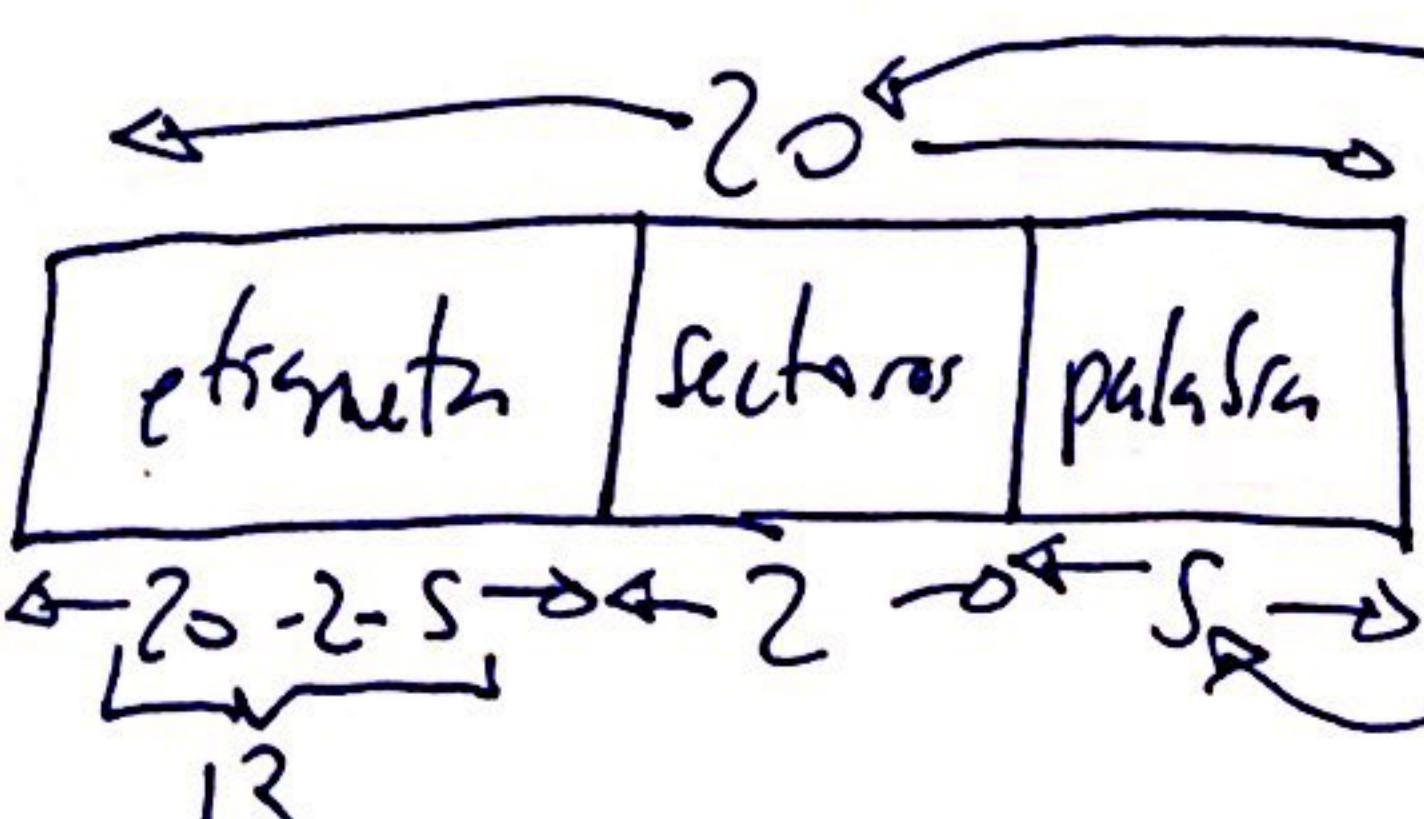
Tam. Bloque: 32 palabras = 2^5 pal/línea

A. Correspondencia Directa

$$\frac{2^{13} \text{ pal cache}}{2^5 \text{ pal/línea}} = 2^{18} \text{ líneas cache}$$

B. Complemento AscendenteC. Asignación por Contenidos → 4 vías

$$\frac{2^8 \text{ líneas cache}}{2^2 \text{ líneas/ajuste}} = 2^6 \text{ ajustes}$$

D. Por SECTORES → 4 Slices

1 EJERCICIO
TOTAL: $\frac{+64 \text{ sectores}}{65}$

Relación TEMA 6

Memoria

18

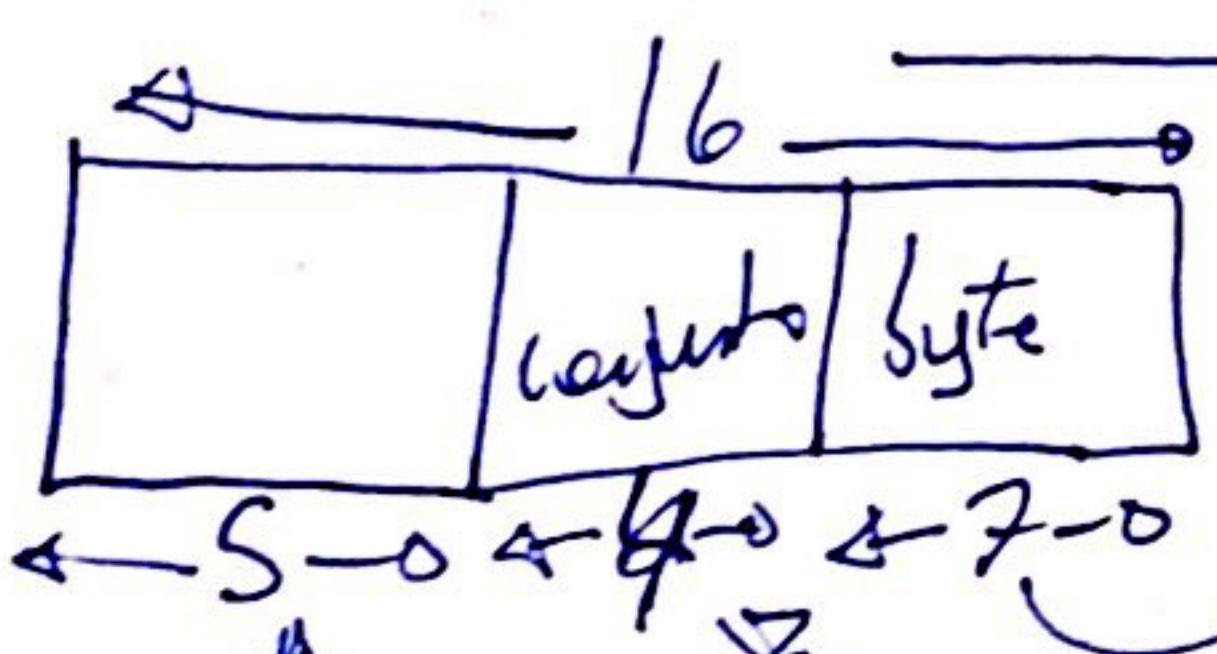
$$pul = 16 \text{ bits} = 2B$$

$$\text{MP: } 32K_{pul} = 32K_{pul} \cdot 16 \text{ bits/pul} = 64KB = 2^6 \cdot 2^{10}B = 2^{16}$$

$$\text{MC: } 4K_{pul} = 8KB = 2^3 \cdot 2^{10}B = 2^{13}B$$

$$\text{Tam. de byte: } 64pul = 128B = 2^7B / \text{lnea para determinar dir.}$$

a)

ASOCIATIVA POR CONSUMOS \rightarrow 4 vías

$$\frac{2^{13}B_{\text{cache}}}{2^7B/\text{lnea}} = 2^6 \text{ lneas cache}$$

$$\frac{2^6 \text{ lneas cache}}{2^2 \text{ bytes/bytes}} = 2^4$$

$$\text{etiquetas} = 16 - 4 - 7 = 5$$

b)

$$\bar{T} = Af_c + (1-A)(t_c + t_m)$$

$$E = \frac{t_c}{T}; \quad T_m = 10t_c \Rightarrow E = \frac{t_m}{t_c} = \frac{10t_c}{t_c} = 10$$

$$E = \frac{1}{1 + Y(1-A)} = \frac{1}{1 + 10(1 - 0.9)} = \frac{1}{1 + 1} = \frac{0.5}{2} = 0.5$$

$E = 0.5$

1 EXERCICIO
 TOTAL % + 65 ALUMNOS
 $\frac{65}{66}$

REL. TEMA 6

MEMORIA

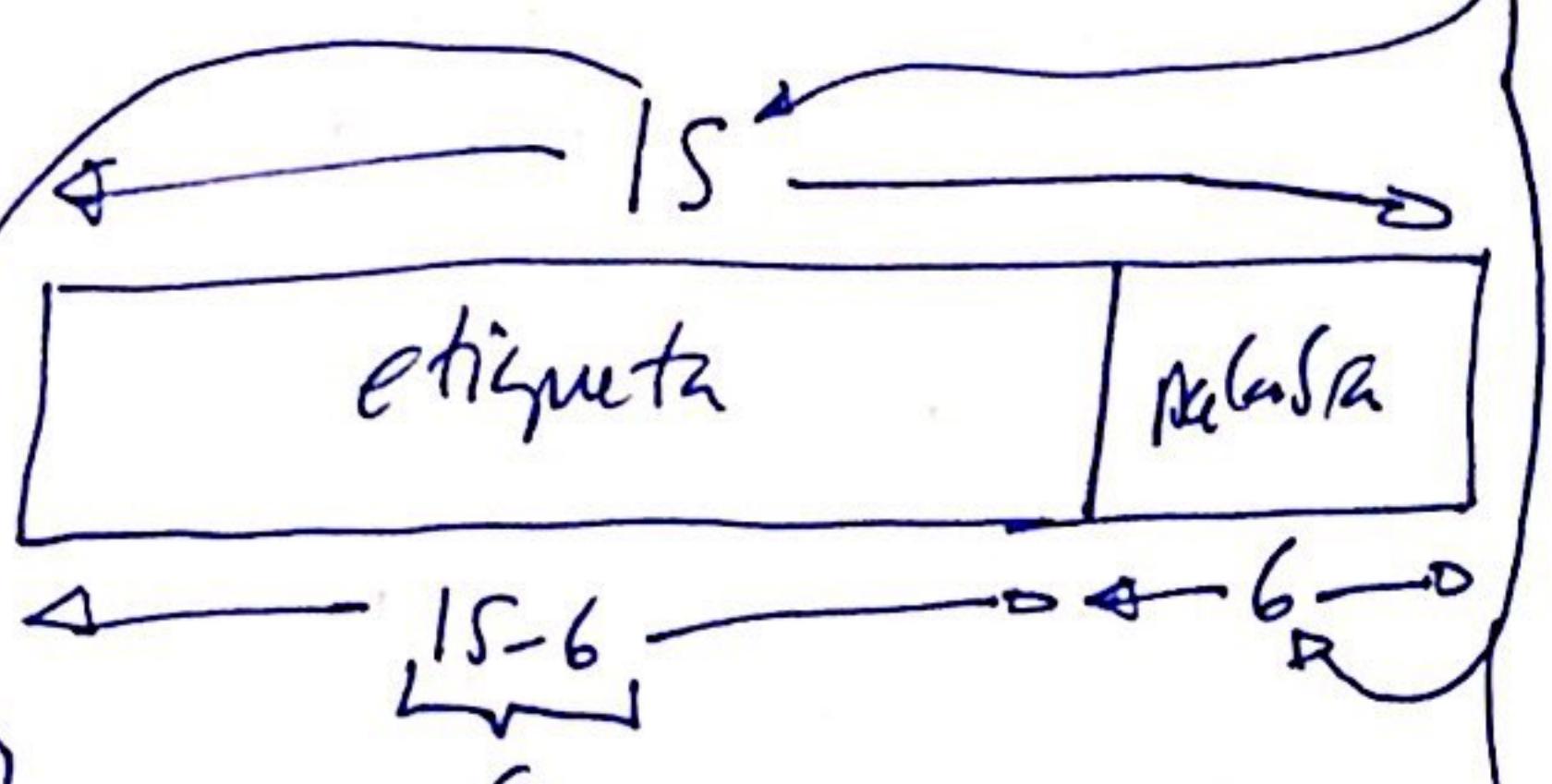
20

$$Mp: 32K\mu l = 2^5 \cdot 2^{10} \mu l = 2^9 \mu l$$

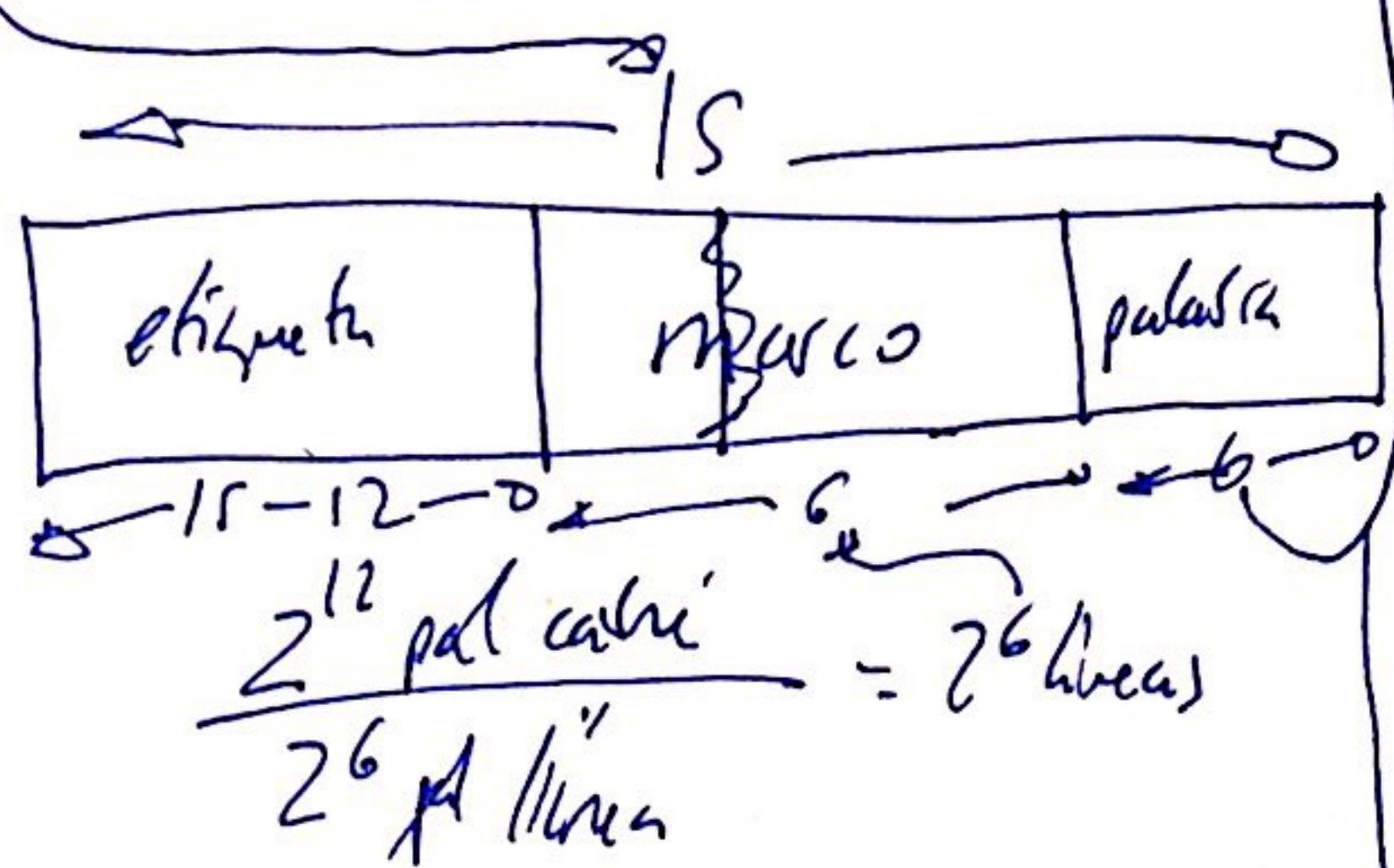
$$Mc: 4K\mu l = 2^2 \cdot 2^{10} \mu l = 2^{12} \mu l$$

$$\text{Tasa Bloque: } 64\mu l = 2^6 \mu l/\text{línea}$$

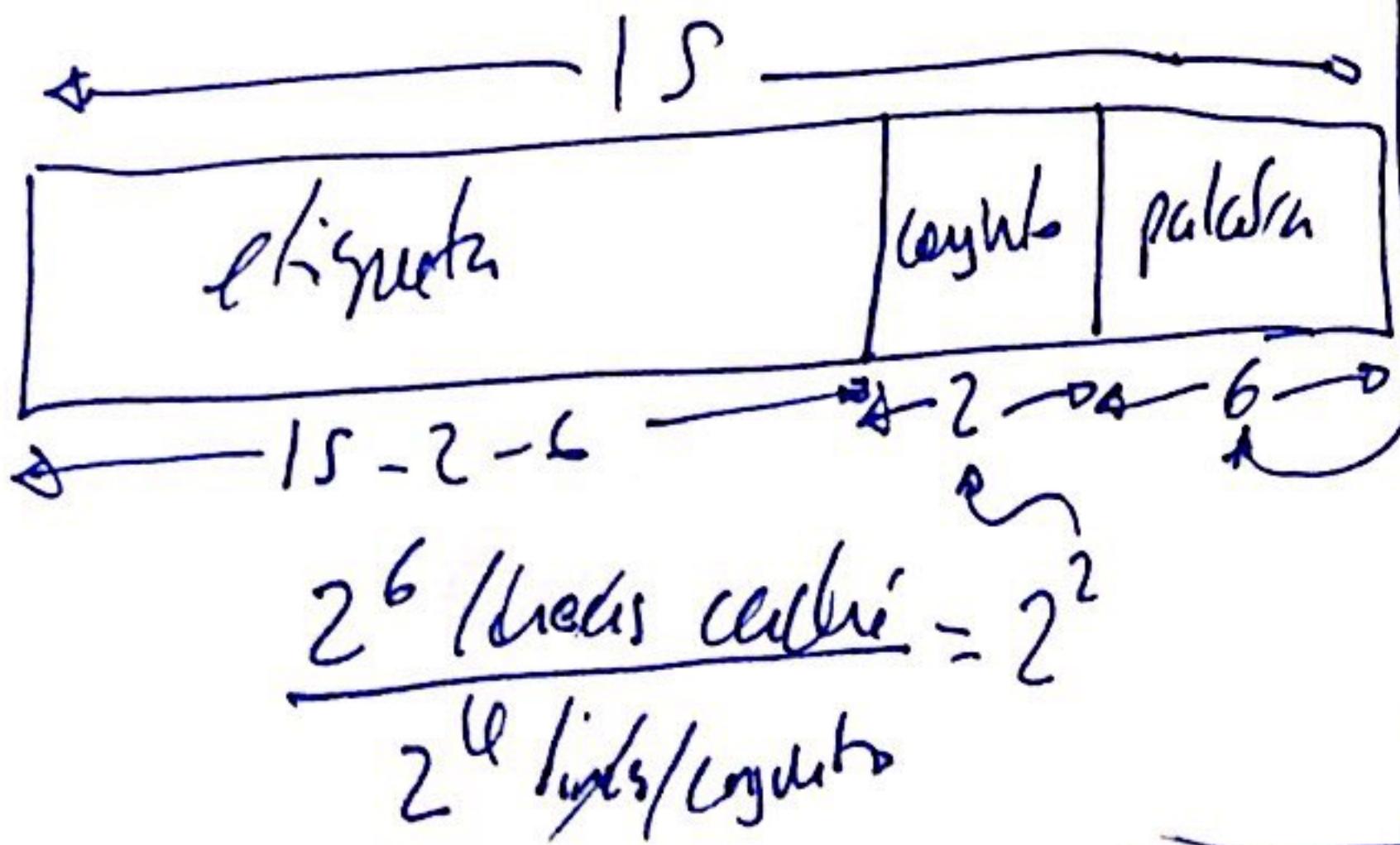
a) TORACENTE ASOCIATIVA



b) DIRECTA



c) ASOCIATIVA POR CONTENIDOS → 4 vías



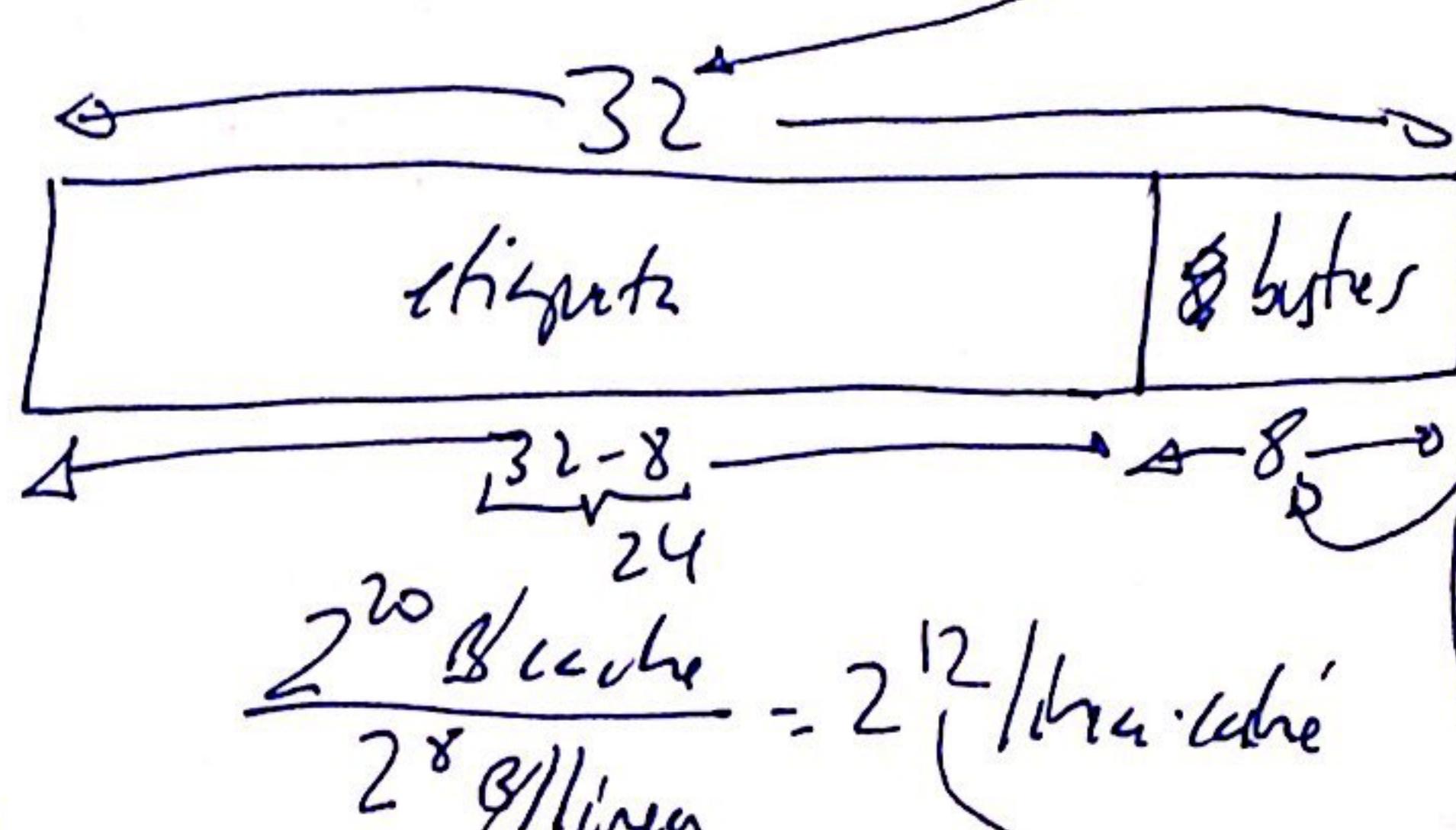
21

$$Mp: 460 = 2^2 \cdot 2^{30} \Rightarrow 32 \text{ bits dirección}$$

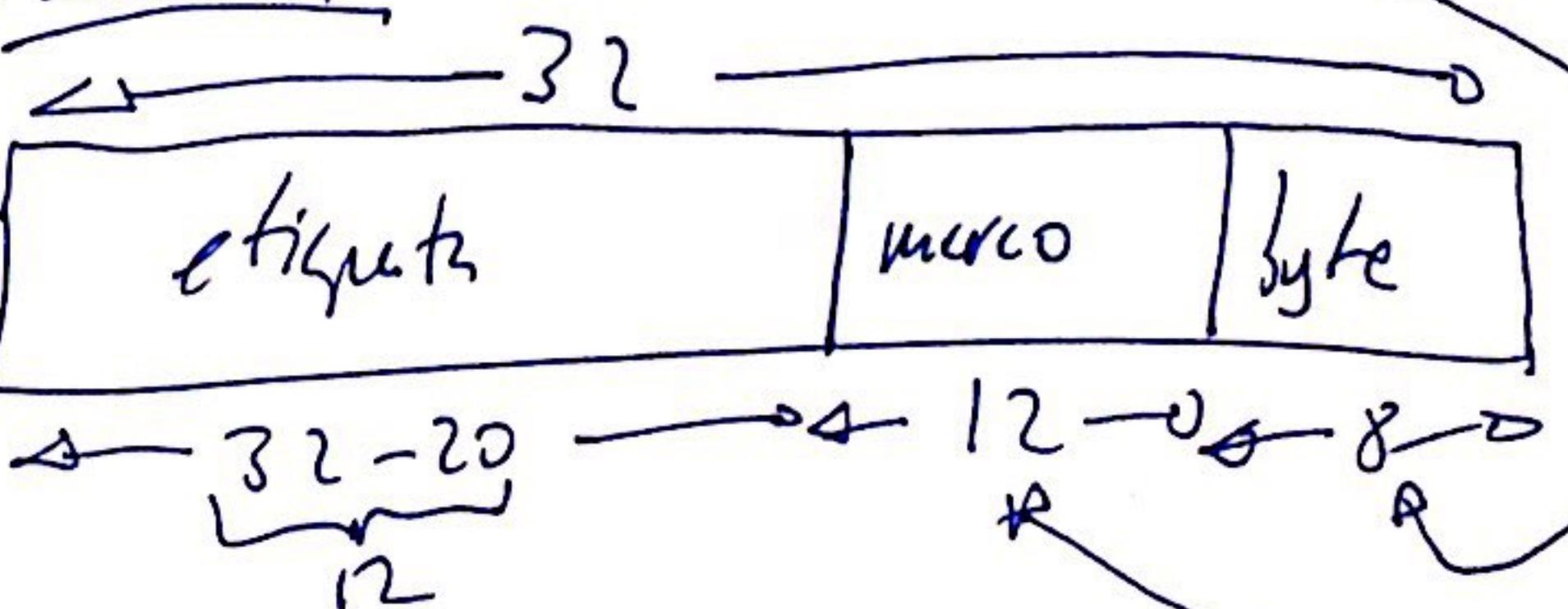
$$Mc: 1MB = 2^{20} B$$

$$\text{Tam bloq: } 256B = 2^8 B/\text{línea}$$

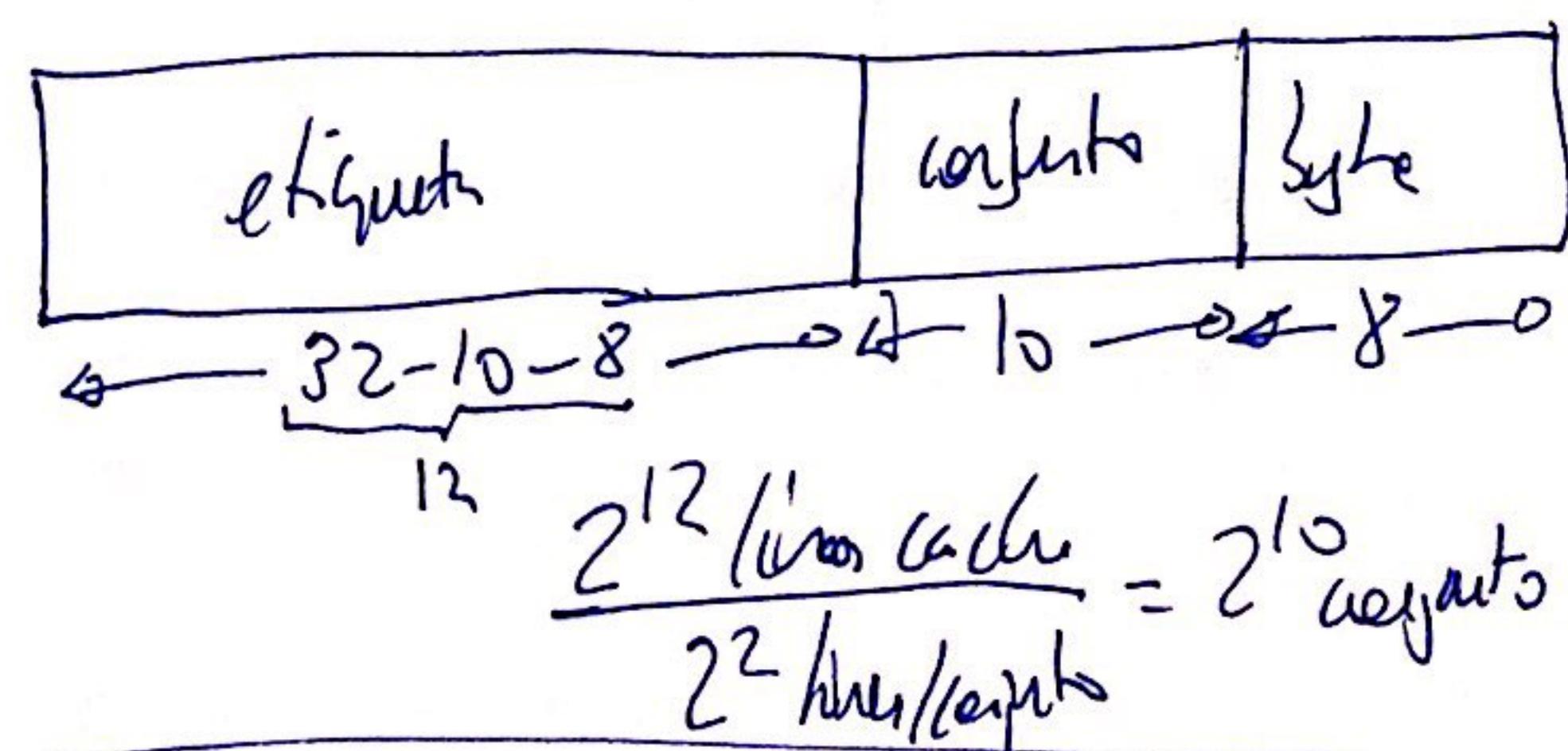
a) TORACENTE ASOCIATIVA



b) DIRECTA



c) ASOCIATIVA POR CONTENIDOS → 4 vías



TOTAL: 2 EJERCICIOS
+ 66 ALMACENAJES
68

TOTAL

EJERCICIOS 8

68