Programación a Nivel-Máquina V: Alineamiento y Uniones

Estructura de Computadores
Semana 6

Bibliografía:

[BRY11] Cap.3 Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
Signatura ESIIT/C.1 BRY com

Transparencias del libro CS:APP, Cap.3
Introduction to Computer Systems: a Programmer's Perspective
Autores: Randal E. Bryant y David R. O'Hallaron

Guía de trabajo autónomo (4h/s)

- **Lectura:** del Cap.3 CS:APP (Bryant/O'Hallaron)
  - Unions, Data Alignment.
    - 3.9.2 – 3.9.3 pp.278-285

- **Ejercicios:** del Cap.3 CS:APP (Bryant/O'Hallaron)
  - Probl. 3.41 - 3.42 pp.285

Bibliografía:

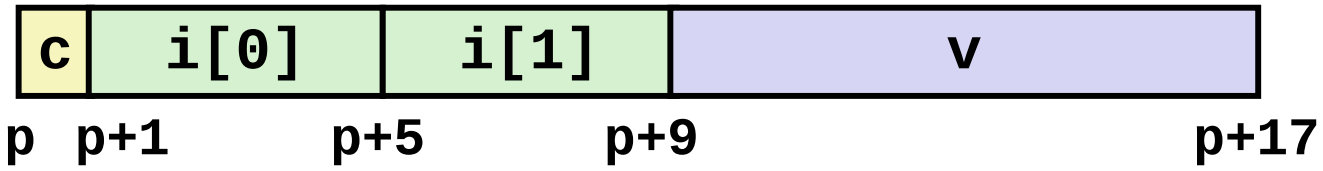
[BRY11] Cap.3 Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
Signatura ESIIT/C.1 BRY com

Progr. Máquina IV: Alineamiento & Uniones

- **Estructuras**
  - Alineamiento
- **Uniones**

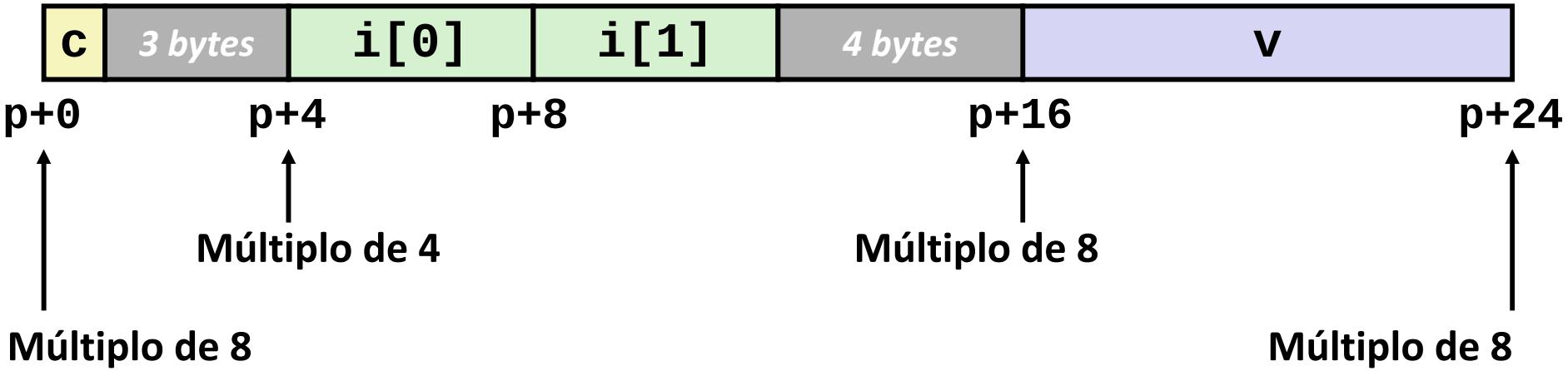
Estructuras & Alineamiento

- **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 IA32
    - ¡tratado diferentemente en IA32 Linux, x86-64 Linux, y Windows!*
- **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

** depende de la ABI del compilador, incluso de la optimización. Ver p.ej. http://gcc.gnu.org/onlinedocs/gcc/Compatibility.html*

Casos Concretos de Alineamiento (IA32)

- **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, char *, …**
  - los 2 LSB's de la dirección deben ser 00_2
- **8 bytes: double, …**
  - Windows (y la mayoría de otros SO's & repertorios):
    - los 3 LSB's de la dirección deben ser 000_2
  - Linux:
    - los 2 LSB's de la dirección deben ser 00_2
    - es decir, se tratan lo mismo que un tipo de datos primitivo de 4-byte
- **12 bytes: long double**
  - Windows, Linux:
    - los 2 LSB's de la dirección deben ser 00_2
    - es decir, se tratan lo mismo que un tipo de datos primitivo de 4-byte

** en inglés suelen decir "lowest 1,2,3 bits" y "least 1,2,3 significant bits" intercambiablemente. En español solemos decir "menos significativos"*

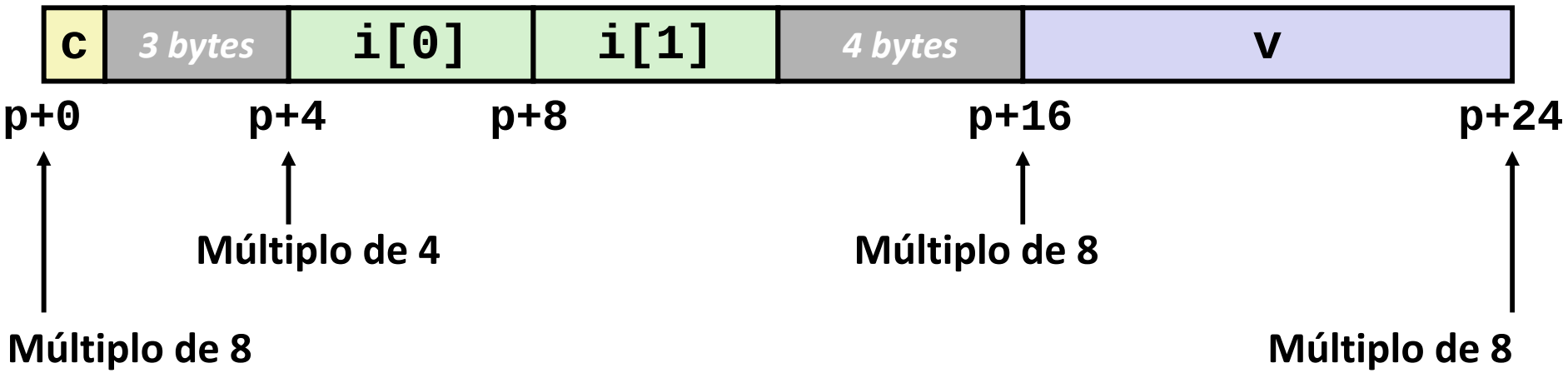
Casos Concretos de Alineamiento (x86-64)

- **1 byte: char, …**
  - sin restricciones en la dirección
- **2 bytes: short, …**
  - el LSB de la dirección debe ser 0_2
- **4 bytes: int, float, …**
  - los 2 LSB's de la dirección deben ser 00_2
- **8 bytes: double, char *, …**
  - Windows & Linux:
    - los 3 LSB's de la dirección deben ser 000_2
- **16 bytes: long double**
  - Linux:
    - los 3 LSB's de la dirección deben ser 000_2
    - es decir, se tratan lo mismo que un tipo de datos primitivo de 8-byte

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 & longitud estructura deben ser múltiplos de **K**
- **Ejemplo (bajo Windows ó x86-64):**
  - K = 8, debido al elemento **`double`**

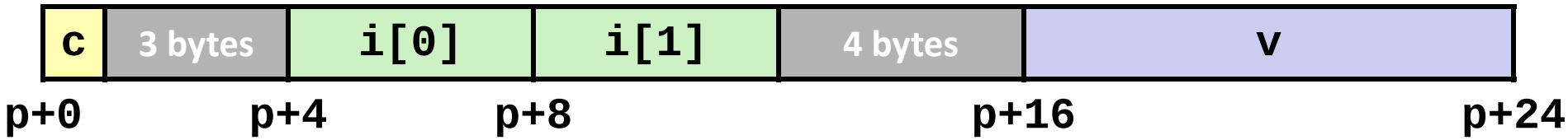
```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```



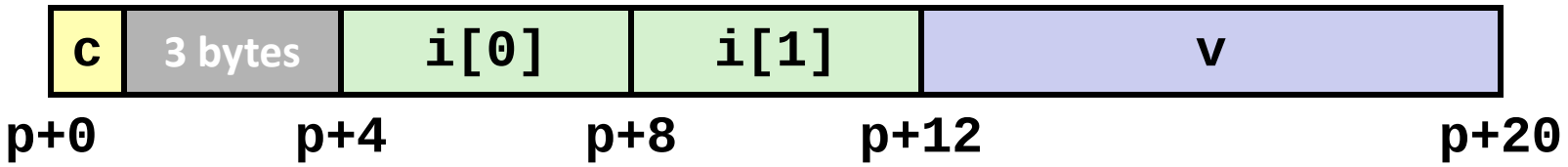
Convenciones de Alineamiento Distintas

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

- **x86-64 ó IA32 Windows:**
  - K = 8, debido al elemento **double**



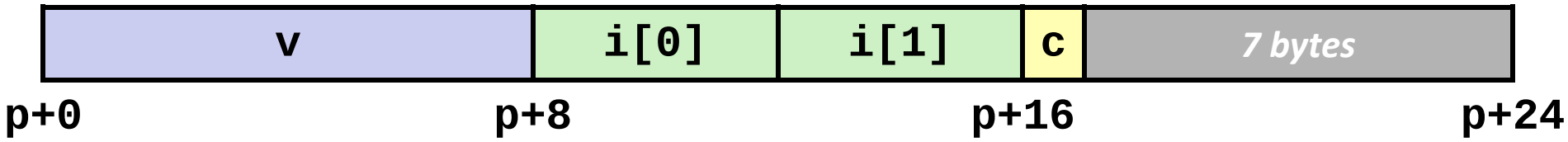
- **IA32 Linux**
  - K = 4; double tratado como un tipo de datos de 4-byte



Cumpliendo Requisito Alineamiento Global

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

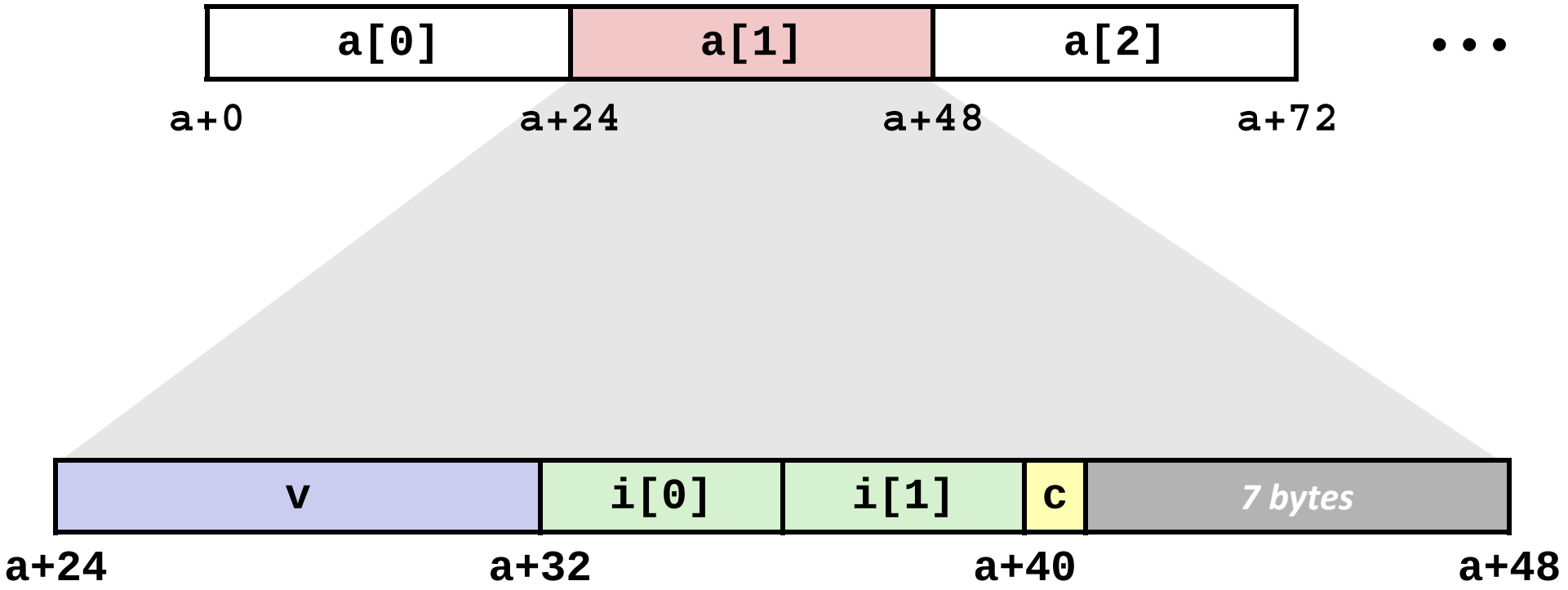
```
struct S2 {
  double v;
  int i[2];
  char c;
} *p;
```



Arrays de Estructuras

- Longitud global estructura* múltiplo de K
- Cumplir requisitos alineamiento de cada elemento

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

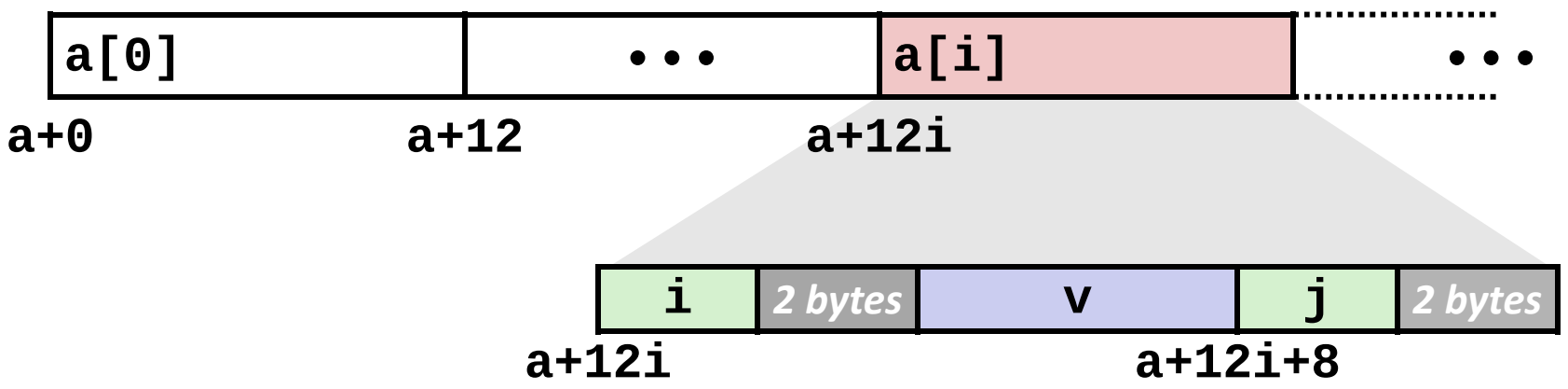


* …y por consiguiente del array, puesto que son estructuras concatenadas

Acceso a Elementos del Array

```
struct S3 {
  short i;
  float v;
  short j;
} a[10];
```

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




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

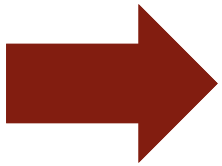
```
                          # %eax = idx
 leal (%eax,%eax,2),%eax # 3*idx
movswl a+8(,%eax,4),%eax
```

"movswl" = "Move Signed, Word to Long" ya que j es short 2B

Ahorro de Espacio

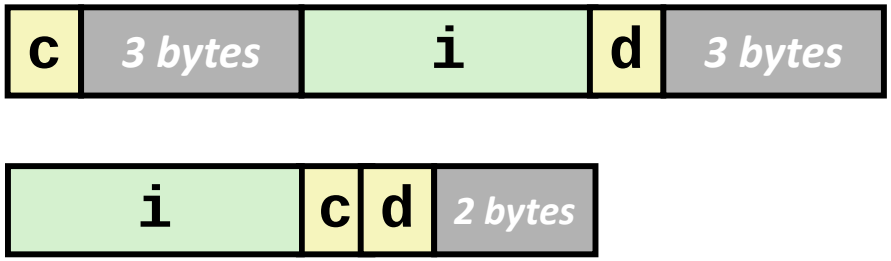
- Poner primero los tipos de datos grandes

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



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

- Efecto (K=4)



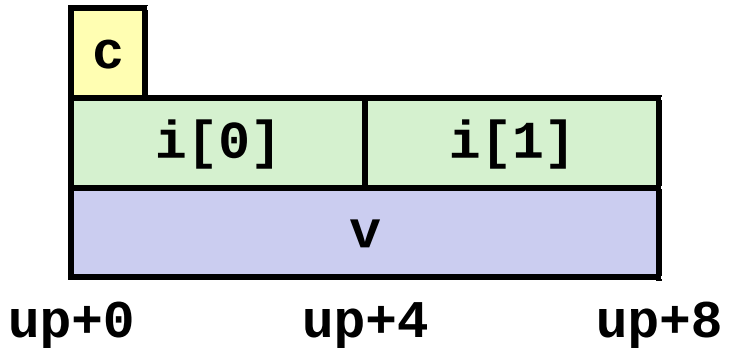
Progr. Máquina IV: Alineamiento & Uniones

- Estructuras
  - Alineamiento
- **Uniones**

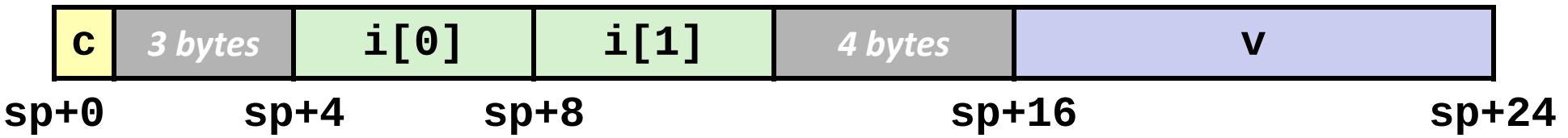
Ubicación† de Uniones

- 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;
```



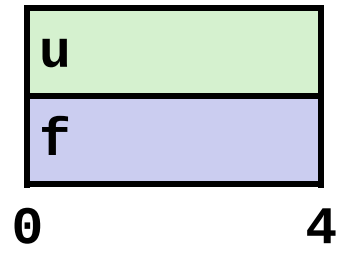

```
struct S1 {
  char c;
  int i[2];
  double v;
} *sp;
```



† *"alloc(ation)/(ate)" en inglés. Ver lección 2.4 pág.17*

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;
}
```

¿Lo mismo que (float) u ?

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

¿Lo mismo que (unsigned) f ?

Orden de Bytes†: un Repaso

- **Idea**
  - Palabras short/long/quad, almacenadas en mem. como 2/4/8B consecutivos
  - ¿Cuál es el más (menos) significativo?
  - Puede causar problemas al intercambiar datos binarios entre máquinas
- **Big Endian (extremo mayor)**
  - El byte más significativo está en la dirección más baja (“viene primero”)
  - Sparc
- **Little Endian (extremo menor)**
  - El byte menos significativo está en la dirección más baja
  - Intel x86

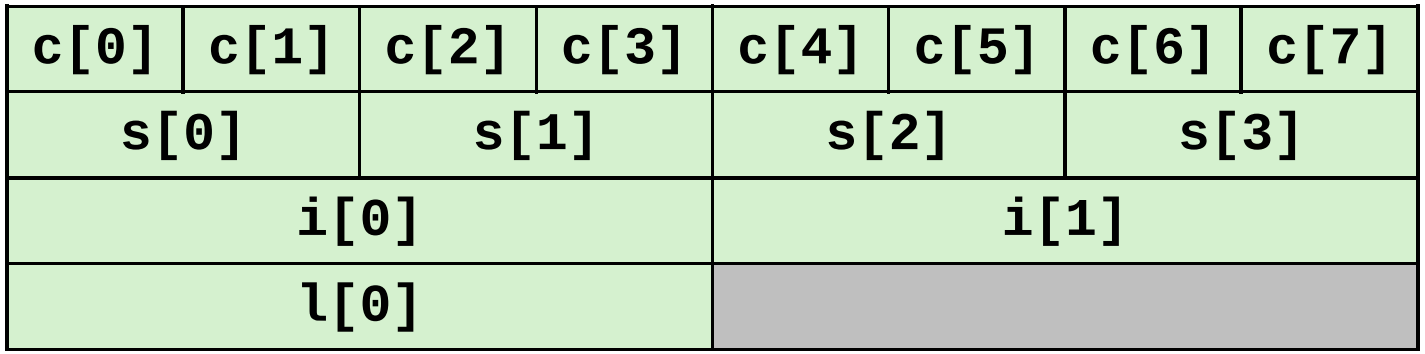
† “byte ordering” se refiere a ordenamiento de bytes en palabras, no a ordenar un array de bytes = “sorting”

** “big/little endian” = “partidario del extremo mayor/menor”, ver explicación en el libro, §2.1.4, Aside: Origin of endian*

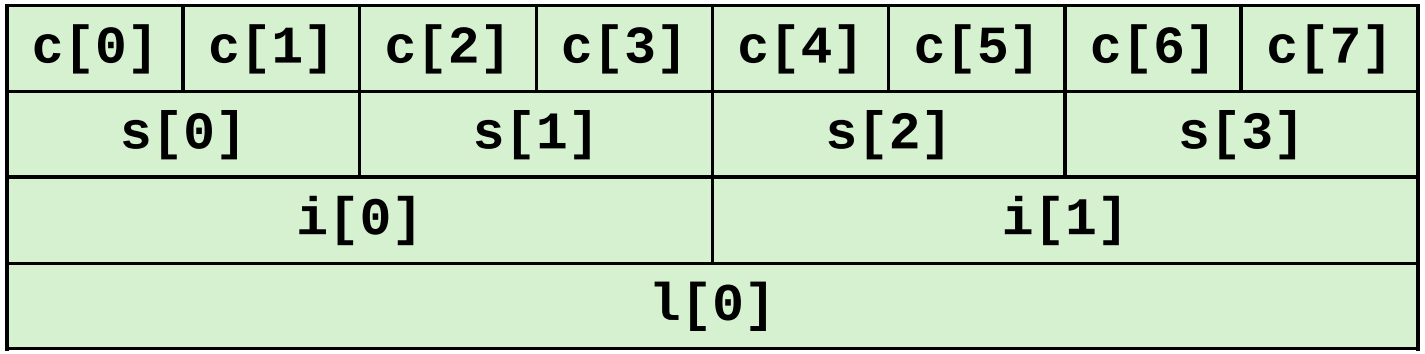
Ejemplo de Orden de Bytes

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

32-bit



64-bit

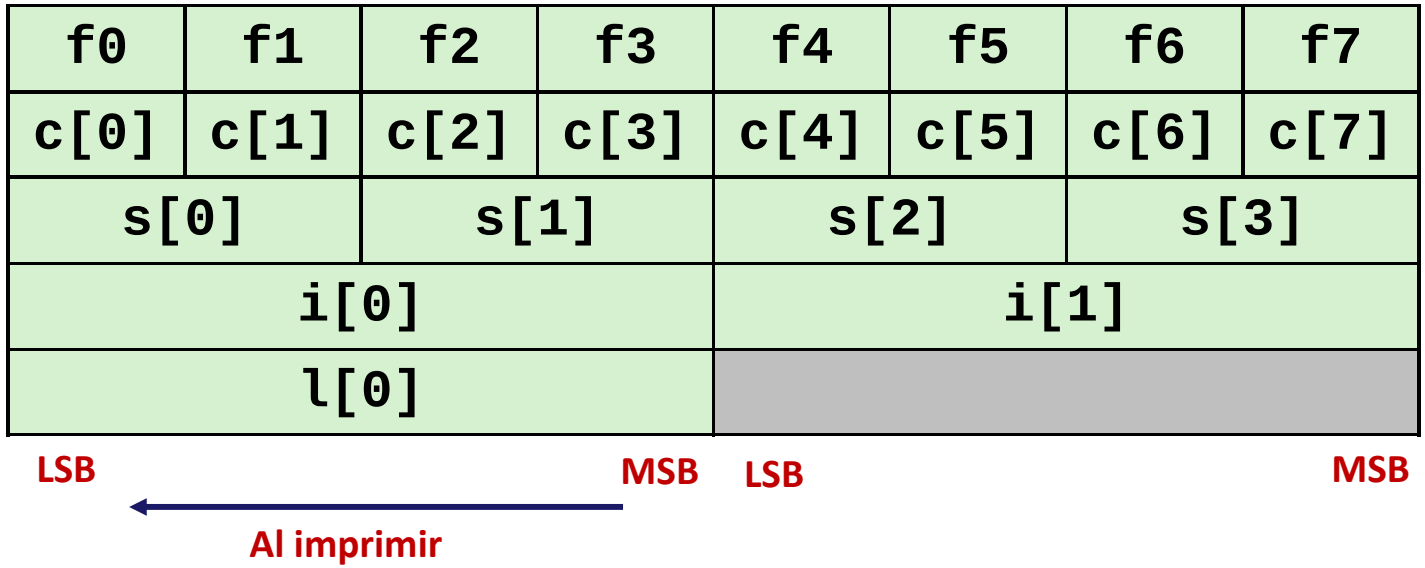


Ejemplo de Orden de Bytes (Cont).

```
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]);
```

Orden de Bytes en IA32

Little Endian

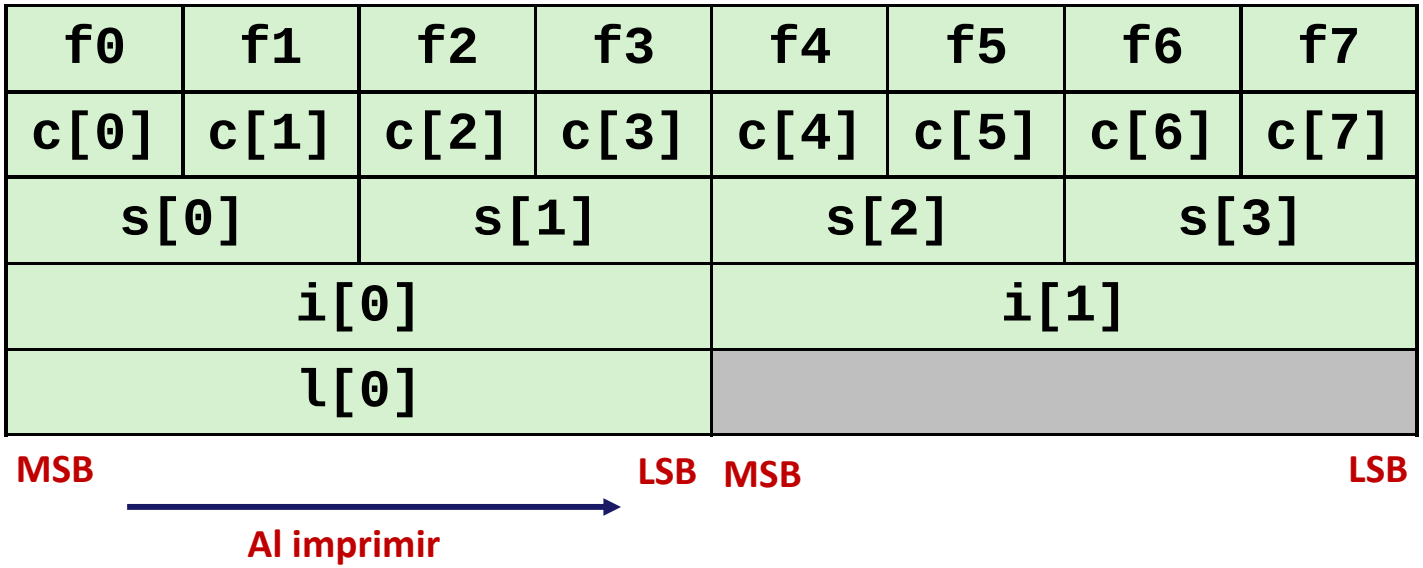


Salida:

```
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]
```

Orden de Bytes en Sun

Big Endian

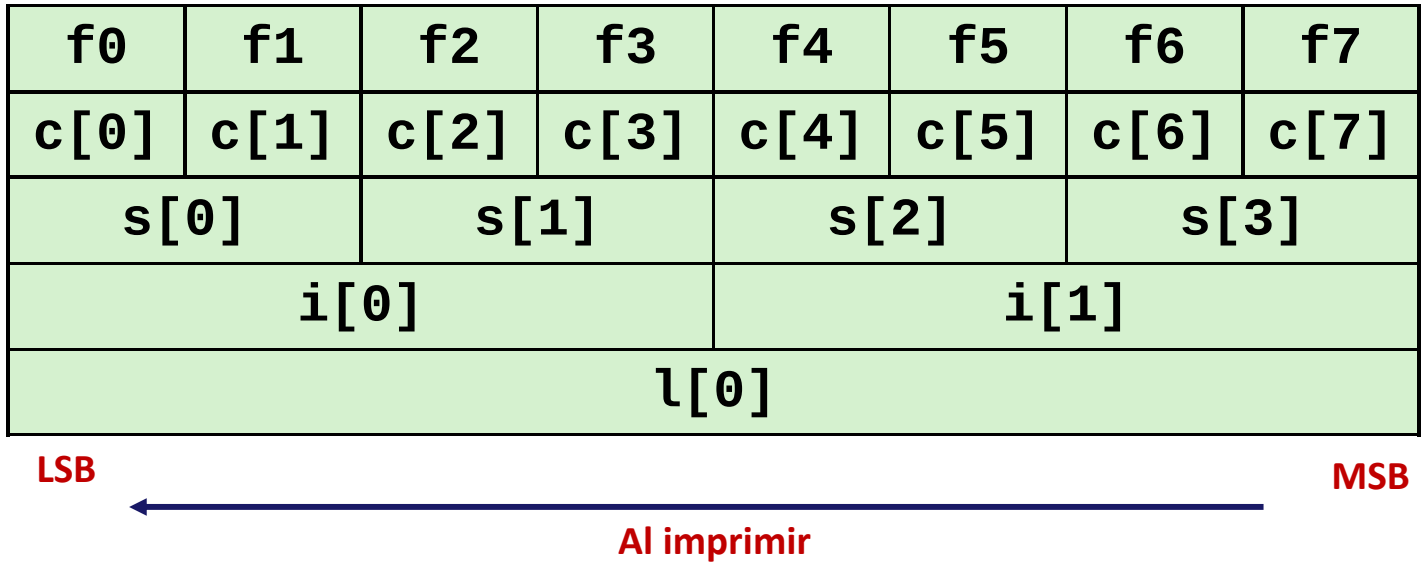


Salida en Sun:

```
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]
```

Orden 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

- **Arrays en C**
  - Reserva de memoria contigua
  - Alineados hasta cumplir con requisitos alineamiento de cada elemento
  - Puntero al primer elemento
  - Sin chequeo de límites
- **Estructuras**
  - Reservar bytes en el orden declarado
  - Rellenos en medio y al final para cumplir con alineamiento
- **Uniones**
  - Declaraciones superpuestas
  - Una forma de soslayar el sistema de tipos de datos

Guía de trabajo autónomo (4h/s)

- **Estudio:** del Cap.3 CS:APP (Bryant/O'Hallaron)
  - Unions, Data Alignment.
    - 3.9.2 – 3.9.3 pp.278-285
    - Probl. 3.41 - 3.42 pp.285

Bibliografía:

[BRY11] Cap.3 Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
Signatura ESIIT/C.1 BRY com