

Nội dung

- **Mảng - Array**

- Mảng 1 chiều
- Mảng 2 chiều (nested)
- Nhiều chiều

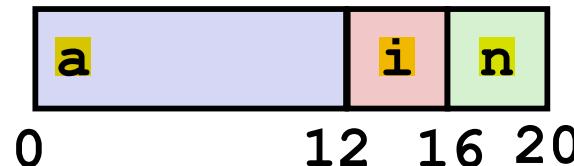
- **Cấu trúc – Structure**

- Cấp phát
- Truy xuất
- Alignment (căn chỉnh)

Cấp phát Structure

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

Memory Layout in IA32



■ Ý tưởng

- Là một **vùng nhớ** được **cấp phát liên tục**
- **Tham chiếu** đến các **thành phần** trong **structure** bằng **tên**
- Các **thành phần** có thể **khác** **kiểu dữ liệu**

■ Các trường được sắp xếp dựa trên định nghĩa

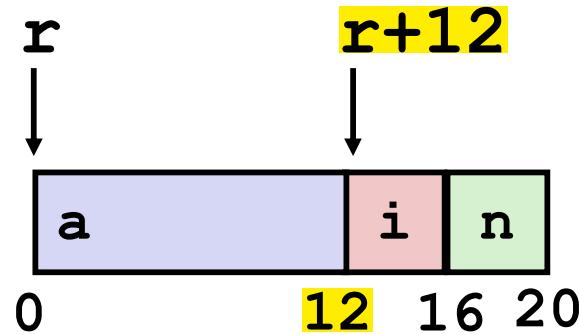
- Ngay cả khi cách sắp xếp khác có thể biểu diễn gọn hơn

■ Compiler quyết định kích thước tổng + vị trí các trường

- Các chương trình mức máy tính không biết về cấu trúc trong source code

Truy xuất structure

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



■ Truy xuất các thành phần trong structure dựa trên:

- Con trỏ xác định vị trí **bắt đầu** của structure
- **Offset** hay **khoảng cách** từ vị trí bắt đầu structure đến vị trí của từng thành phần

khi build hệ thống tự thêm tham số

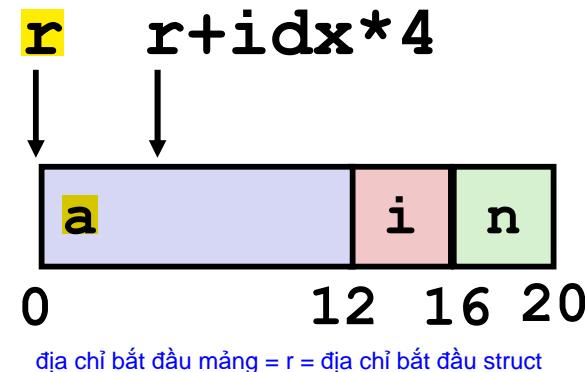
```
void  
set_i(struct rec *r,  
      int val)  
{  
    r->i = val;  
}
```

IA32 Assembly

```
# %edx = val  
# %eax = r  
movl %edx, 12(%eax) # Mem[r+12] = val
```

Con trỏ đến thành phần Structure - 1

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



địa chỉ bắt đầu mảng = r = địa chỉ bắt đầu struct

Tạo con trỏ đến thành phần trong structure

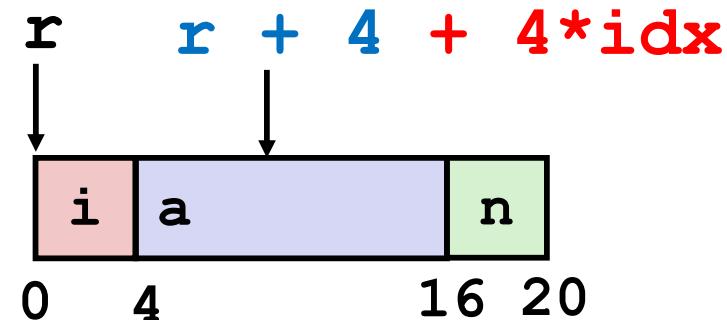
- Offset của mỗi thành phần structure được xác định lúc biên dịch
- Ví dụ: con trỏ đến **a[idx]** trong structure: **r + 4*idx**

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

```
movl 12(%ebp), %eax # Get idx  
sall $2, %eax         # idx*4  
addl 8(%ebp), %eax # r+idx*4
```

Con trỏ đến phần Structure - 2

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



Tạo con trỏ đến phần trong structure

- Offset của mỗi thành phần structure được xác định lúc biên dịch
- Ví dụ: con trỏ đến **a[idx]** trong structure: **r + 4 + 4*idx**

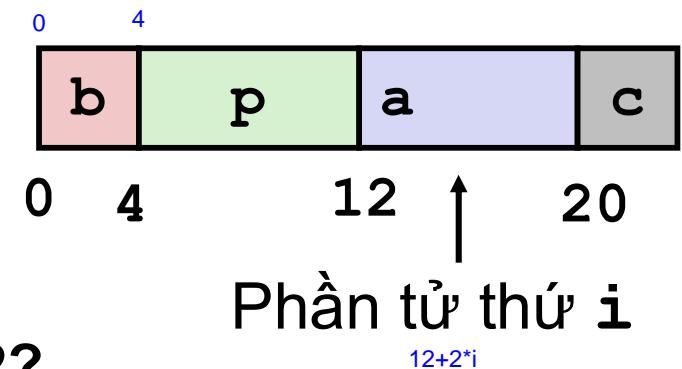
```
int *get_ap(struct rec *r, int idx)  
{  
    return &r->a[idx];  
}
```

```
movl 8(%ebp), %ebx      # Get r  
leal 4(%ebx), %esi      # r+4  
movl 12(%ebp), %edi      # Get idx  
sall $2, %edi            # 4*idx  
leal (%edi,%esi), %eax  # r+4+4*idx
```

Ví dụ: Truy xuất structure

■ C Code

```
struct example{  
    int b;  
    double p;  
    short a[4];  
    char* c;  
};
```



■ Xác định offset của các trường trong IA32?

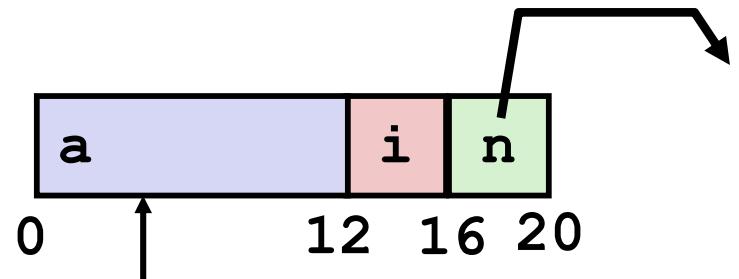
Trường	Offset
b	0
p	4
c	20
a[i]	$12 + 2 \cdot i$

Ví dụ: Danh sách liên kết

C Code

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

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



Phần tử thứ i

Register	Value
%edx	r
%ecx	val

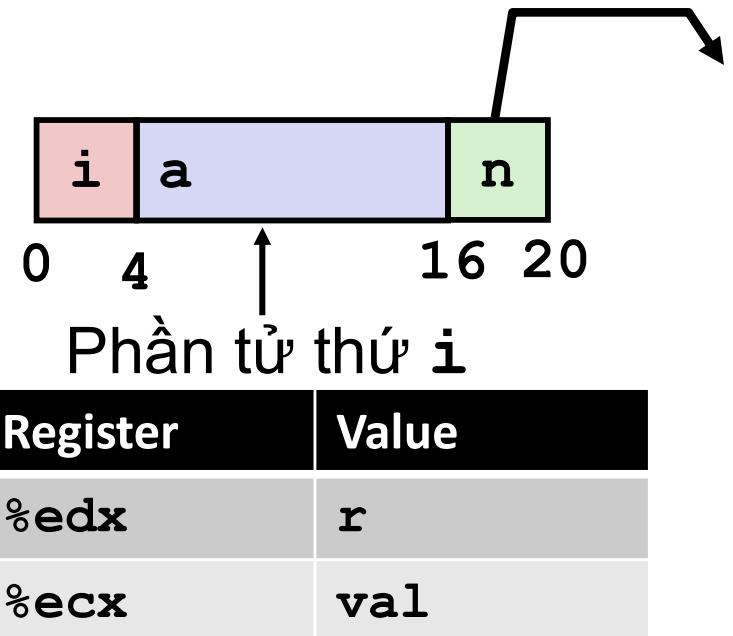
```
.L17:                                # loop:
    movl 12(%edx), %eax             # r->i
    movl %ecx, (%edx,%eax,4)       # r->a[i] = val
    movl 16(%edx), %edx            # r = r->n
    testl %edx, %edx               # Test r
    jne .L17                      # If != 0 goto loop
```

Ví dụ: Danh sách liên kết

C Code

```
void set_val(struct rec *r, int val)
{
    while (r) {
        int i = r->i;      (edx)  eax = i
        r->a[i] = val;   4(edx,eax,4)
        r = r->n;
    }
}
```

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



```
.L17:
    movl (%edx), %eax
    movl %ecx, 4(%edx,%eax,4)
    movl 16(%edx), %edx
    testl %edx, %edx
    jne .L17
```

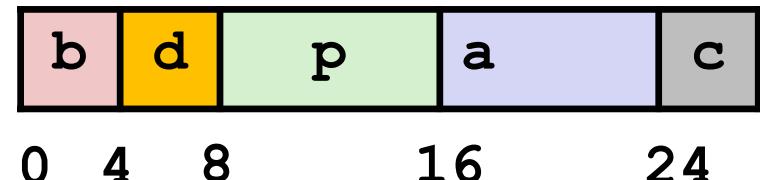
```
# loop:
# r->i
# r->a[i] = val
# r = r->n
# Test r
# If != 0 goto loop
```

Dự đoán tổng kích thước struct?

■ C Code

```
struct example{  
    int b;  
    float d;  
    double p;  
    short a[4];  
    char c;  
};
```

Kích thước struct example = ?



■ Kích thước từng trường?

Trường	Offset	Kích thước
b	0	4
d	4	4
p	8	8
c	24	1
a	16	$4 * 2 = 8$

Structure & Alignment (căn chỉnh)

■ Dữ liệu được căn chỉnh

- Kiểu dữ liệu yêu cầu K bytes
- Địa chỉ phải là bội số của K
- Bắt buộc ở một số hệ thống; được khuyến cáo ở x86_64

■ Vì sao?

- Truy xuất bộ nhớ hiệu quả hơn khi alignment
- Bộ nhớ được truy xuất bằng các khối 4 hoặc 8 byte (tuỳ hệ thống)
 - Load hoặc lưu các dữ liệu lớn hơn 8 bytes không hiệu quả
 - Bộ nhớ ảo phức tạp hơn khi dữ liệu lớn hơn 2 pages

■ Compiler

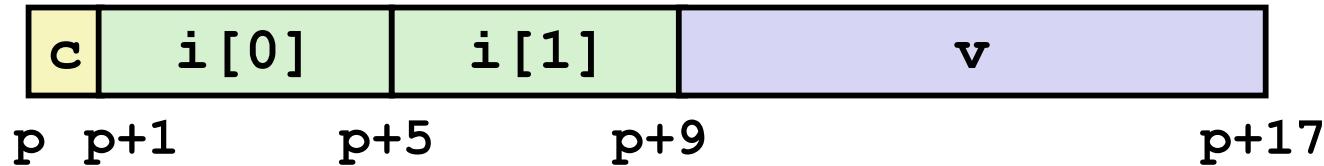
tim địa chỉ là bội số K

- Thêm những khoảng trống vào structure để đảm bảo căn chỉnh đúng cho các trường dữ liệu

mặc định hệ thống tự căn chỉnh

Structure & Alignment (căn chỉnh)

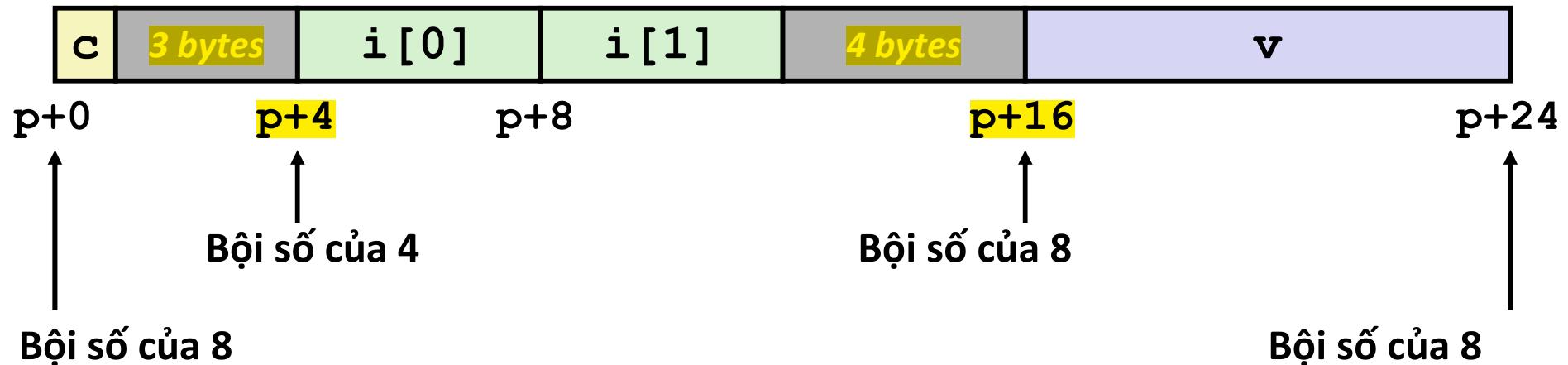
■ Dữ liệu không căn chỉnh



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

■ Dữ liệu được căn chỉnh

- Kiểu dữ liệu yêu cầu **K bytes**
- Địa chỉ phải là bội số của **K**



Yêu cầu căn chỉnh (IA32)

- **1 byte: char, ...**
 - Không có ràng buộc về địa chỉ
- **2 bytes: short, ...**
 - Bit thấp nhất của địa chỉ phải bằng 0_2
- **4 bytes: int, float, char * ...**
 - 2 bit thấp nhất của địa chỉ phải bằng 00_2
- **8 bytes: double, ...**
 - Windows (và hầu hết các OS và instruction set khác):
 - 3 bits thấp nhất của địa chỉ phải bằng 000_2
 - Linux:
 - 2 bit thấp nhất của địa chỉ phải bằng 00_2 (xem như xử lý kiểu dữ liệu 4 byte)
- **12 bytes: long double**
 - Windows, Linux:
 - 2 bit thấp nhất của địa chỉ phải bằng 00_2 (xem như xử lý kiểu dữ liệu 4 byte)

Yêu cầu căn chỉnh (x86_64)

- **1 byte: char, ...**
 - Không có ràng buộc về địa chỉ
- **2 bytes: short, ...**
 - Bit thấp nhất của địa chỉ phải bằng 0_2
- **4 bytes: int, float, ...**
 - 2 bit thấp nhất của địa chỉ phải bằng 00_2
- **8 bytes: double, long, char *, ...**
 - 3 bit thấp nhất của địa chỉ phải bằng 000_2
- **16 bytes: long double (GCC on Linux)**
 - 4 bit thấp nhất của địa chỉ phải bằng 0000_2

Đảm bảo căn chỉnh với Structure

■ Trong structure

- Phải đảm bảo yêu cầu căn chỉnh của mỗi thành phần

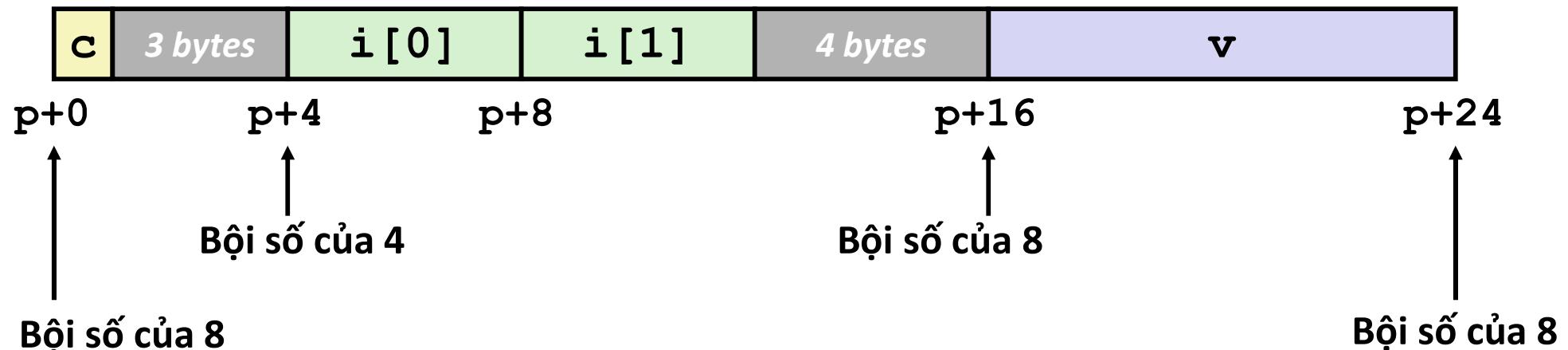
■ Vị trí chung của structure

- Mỗi structure có yêu cầu căn chỉnh K
 - K = Yêu cầu căn chỉnh lớn nhất của các thành phần
 - Địa chỉ bắt đầu & kích thước structure phải là bội số của K

■ Ví dụ trong hệ thống 64 bit:

- K = 8, do thành phần có K lớn nhất là kiểu double

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

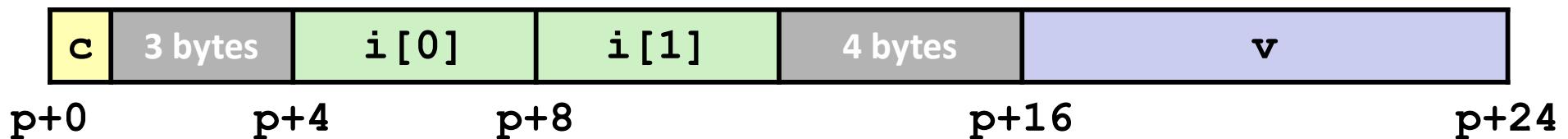


Ví dụ: Các quy ước căn chỉnh khác nhau

■ x86-64 hoặc IA32 Windows:

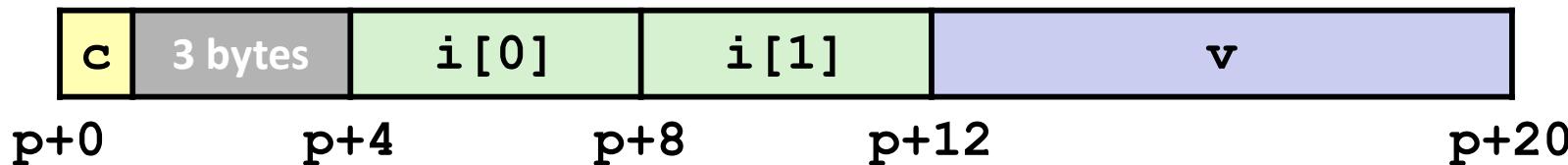
- K = 8, do thành phần lớn nhất có kiểu **double**

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



■ IA32 Linux

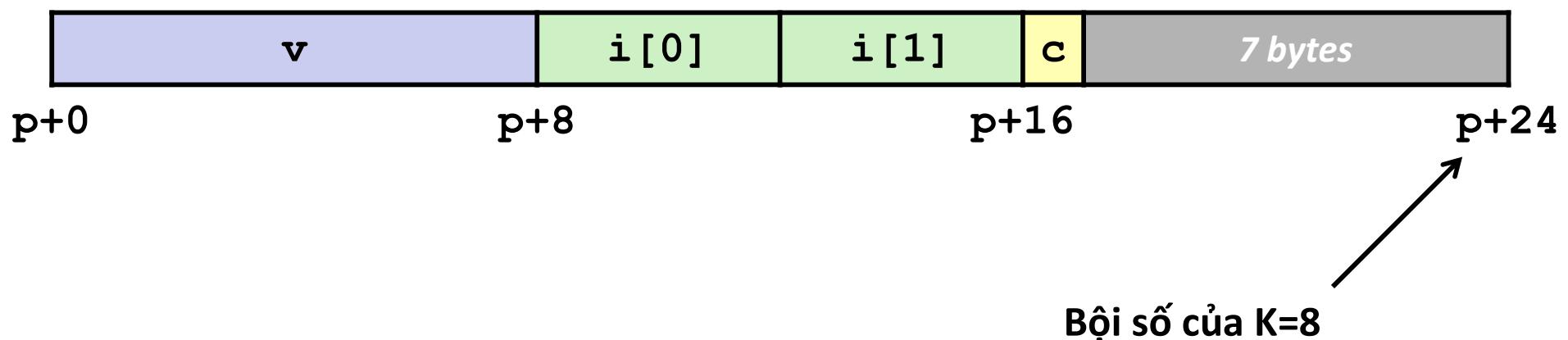
- K = 4; kiểu **double** vẫn được xử lý như kiểu dữ liệu 4-byte



Đảm bảo yêu cầu căn chỉnh chung

- Với yêu cầu căn chỉnh lớn nhất **K**
- Structure phải có kích thước là bội số của **K**

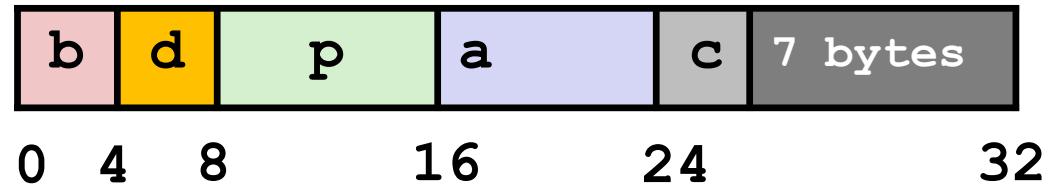
```
struct s2 {  
    double v;  
    int i[2];  
    char c;  
} *p;
```



Tổng kích thước struct (có căn chỉnh) – 64 bit

■ C Code

```
struct example{  
    int b;  
    float d;  
    double p;  
    short a[4];  
    char c;  
};
```



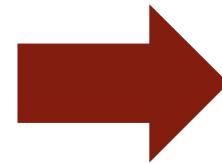
Structure phải có kích thước là bội số của K(double)

Kích thước struct example = 32

Tiết kiệm không gian lưu trữ

■ Khai báo các kiểu dữ liệu lớn trước

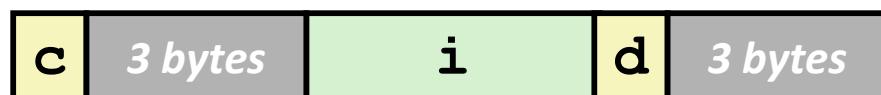
```
struct S4 {  
    char c;  
    int i;  
    char d;  
} *p;
```



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

■ Tác dụng (K=4)

S4



S5



Tổng kích thước struct

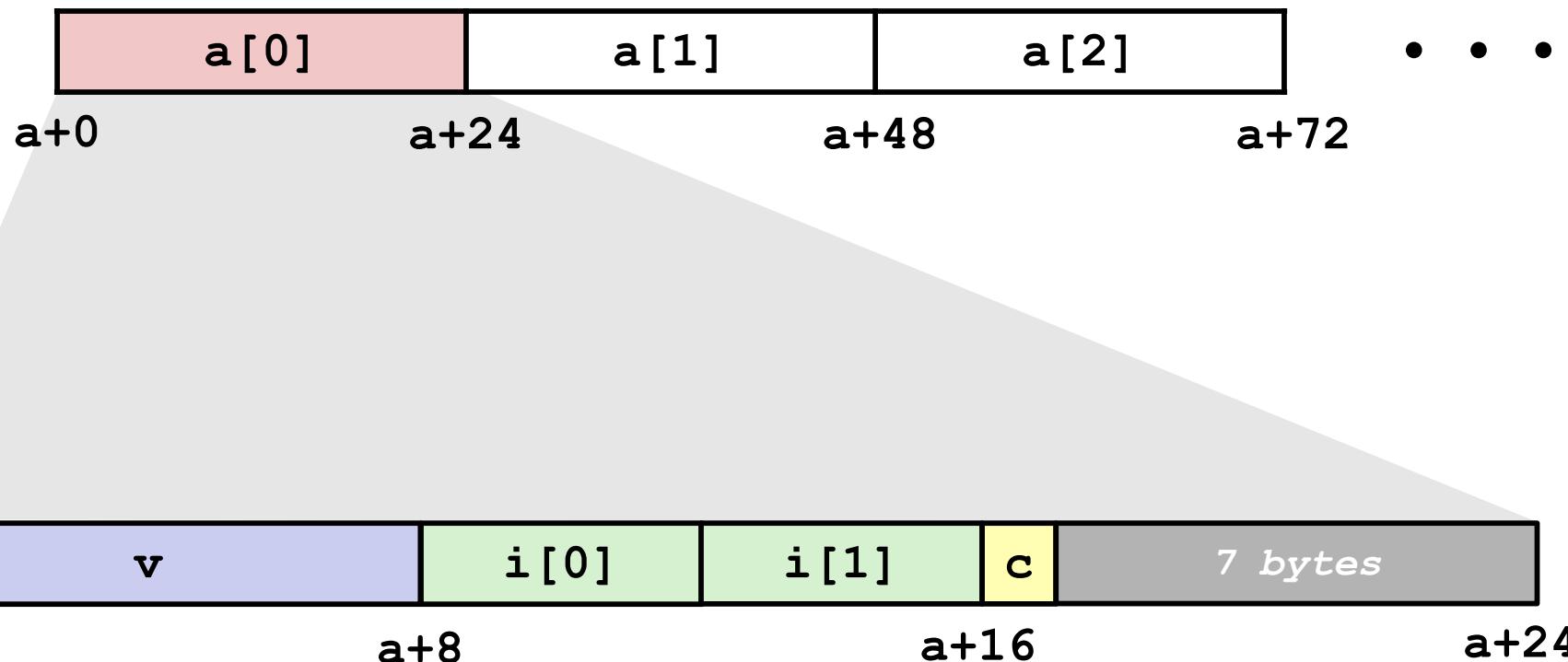
```
1 #include <stdio.h>
2
3 struct example{
4     int b;
5     float d;
6     double p;
7     short a[4];
8     char c;
9 };
10
11
12 int main()
13 {
14     struct example ex1;
15     printf("Total size of ex1 is %d\n", sizeof(ex1));
16     return 0;
17 }
```

```
$gcc -o main *.c
$main
Total size of ex1 is 32
```

Thêm: Căn chỉnh trong mảng Structures

- Kích thước structure là bội số của K
- Đảm bảo yêu cầu căn chỉnh cho tất cả thành phần
- Structure S2 trong x86_64: K = 8

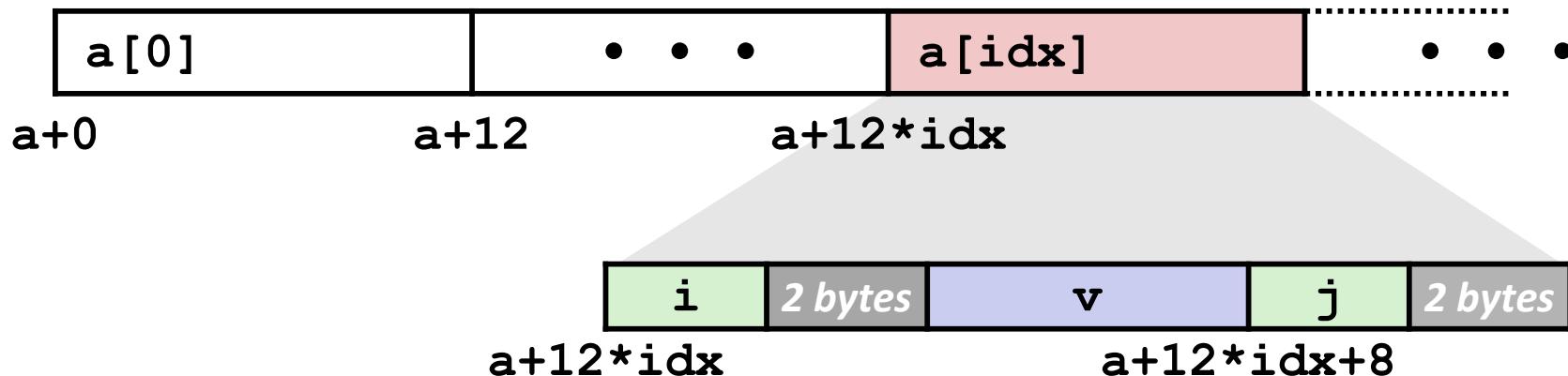
```
struct S2 {  
    double v;  
    int i[2];  
    char c;  
} a[10];
```



Thêm: Truy xuất phần tử mảng structure

- Offset để truy xuất phần tử mảng a: $12 * \text{idx}$
 - `sizeof(S3)=12` gồm cả khoảng trống để căn chỉnh
- Thành phần j nằm ở offset 8 trong structure
- Assembler cung cấp sẵn offset a+8
 - Qua quá trình linking

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



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

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

Structure & Alignment: Bài tập 1 (*)

Cho định nghĩa cấu trúc như sau được biên dịch trên 1 máy Windows 32-bit và có yêu cầu alignment.

```
struct {
    char *a; 0->4
    short b; 4->6
    double c; 8->16
    char d; 17
    float e; 20
    void *f; 24->28
} foo; 28->32
```

```
{
char a[10] -> 10 => 0-10
int b -> 4 => 12-16
double c -> 4 => 16-24
short d -> 2 => 24-26
float *e -> 4 => 28-32
} foo
```

```
union
{
char a[10] ->10 | ali:1
int b ->4| ali 4
double c ->8|ali 8
short d -> 2|ali 2
float *e -> 4|ali4
} foo
=> tổng 16 byte
```

a. Xác định vị trí (offset) của từng trường trong structure này? Vẽ hình minh họa việc cấp phát structure?

b. Tổng kích thước của structure?

32

c. Sắp xếp lại vị trí các trường để hạn chế tối thiểu không gian trống? Offset của các trường và tổng kích thước sau khi sắp xếp lại?

```
c->a->e->f->b->d
double -> char* ->float->void*>short -> char
0->8->12->16->20->22->23->24
24 byte
```