

LẬP TRÌNH HỆ THỐNG

ThS. Đỗ Thị Thu Hiền
(hiendtt@uit.edu.vn)



TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN - ĐHQG-HCM
KHOA MẠNG MÁY TÍNH & TRUYỀN THÔNG
FACULTY OF COMPUTER NETWORK AND COMMUNICATIONS

Tầng 8 - Tòa nhà E, trường ĐH Công nghệ Thông tin, ĐHQG-HCM
Điện thoại: (08)3 725 1993 (122)

Mảng - Cấu trúc



Nhắc lại về các kiểu dữ liệu cơ bản

Đơn vị: bytes

Kiểu dữ liệu	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	—	—	10/16
pointer	4	8	8

Kích thước các kiểu dữ liệu (x86-64)

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <float.h>
4
5
6 int main(){
7     char c;
8     short s;
9     int i;
10    long l;
11    float f;
12    double d;
13    long double ld;
14    char *p;
15
16    // sizeof
17    printf("Size of char is %d\n", sizeof c);
18    printf("Size of short is %d\n", sizeof s);
19    printf("Size of int is %d\n", sizeof i);
20    printf("Size of long is %d\n", sizeof l);
21    printf("Size of float is %d\n", sizeof f);
22    printf("Size of double is %d\n", sizeof d);
23    printf("Size of long double is %d\n", sizeof ld);
24    printf("Size of pointer is %d\n", sizeof p);
25 }
```

```
$gcc -o main *.c
$main
Size of char is 1
Size of short is 2
Size of int is 4
Size of long is 8
Size of float is 4
Size of double is 8
Size of long double is 16
Size of pointer is 8
```

Đơn vị: bytes

Nội dung

- **Mảng - Array**
 - Mảng 1 chiều
 - Mảng 2 chiều (nested)
 - Mảng nhiều cấp
- **Cấu trúc – Structure**
 - Cấp phát
 - Truy xuất
 - Alignment (căn chỉnh)

Cấp phát mảng

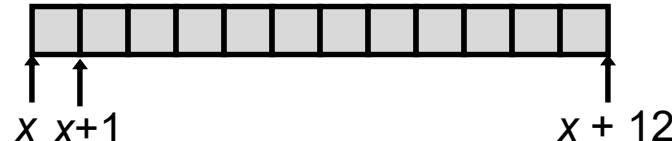
■ Nguyên tắc cơ bản

$T A[L];$

- Mảng của kiểu dữ liệu T và có độ dài L
- Mảng gồm L phần tử có cùng kiểu dữ liệu T
- Vùng nhớ được cấp phát liên tục với $L * \text{sizeof}(T)$ bytes trong bộ nhớ

`char string[12];`

1 byte



`int val[5];`

4 byte



`double a[3];`

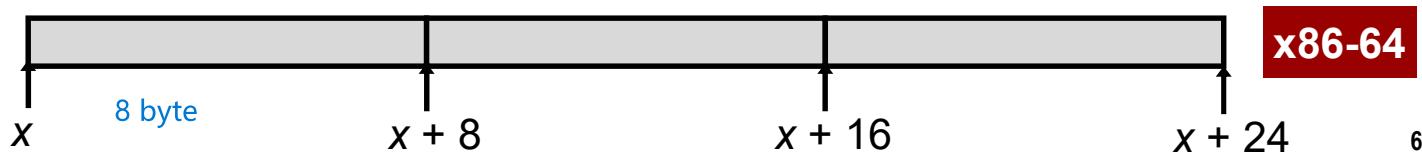
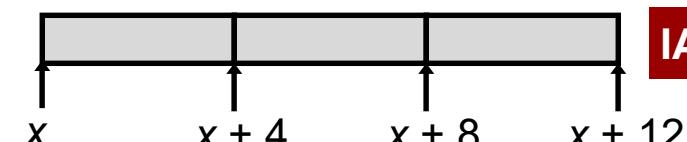
8 byte



`char *p[3];`

4 byte

IA32

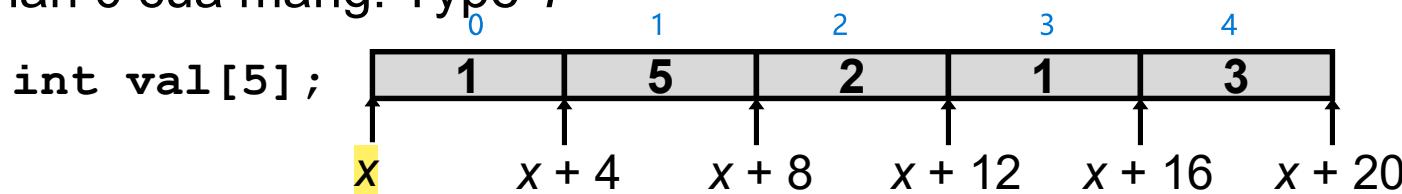


Truy xuất mảng

■ Nguyên tắc cơ bản

$T \in \mathbf{A}[L]$;

- Mảng của kiểu dữ liệu T và có độ dài L
 - Định danh A có thể dùng như là con trỏ trỏ đến mảng hoặc thành phần 0 của mảng: Type T^*



■ Tham chiếu Kiểu dữ liệu Giá trị

val[4]	int	3	
val	int *	x	địa chỉ bắt đầu của mảng
val+1	int *	x + 4	địa chỉ phần tử thứ 2 của mảng &val[1]
&val[2]	int *	x + 8	= val+2
			địa chỉ phần tử thứ 3 của mảng
val[5]	int	??	
* (val+1)	int	5	= val[1] lấy giá trị tại địa chỉ val+1
val + i	int *	x + 4 i	= &val[i]

Truy xuất mảng: Ví dụ

- Cho các mảng sau trong IA32:

```
char A[12];
```

```
char *B[8];
```

$x+n*i$ (size:n)

```
double C[7];
```

```
double *D[5];
```

```
char **E[5];
```

Điền vào bảng dưới đây các thông tin còn thiếu:

Mảng	Kích thước phần tử (byte)	Tổng kích thước (byte)	Địa chỉ bắt đầu	Địa chỉ phần tử thứ i
A	1	12	x_A	x_A+i
B	4	32	x_B	x_B+4i
C	8	56	x_C	x_C+8i
D	4	20	x_D	x_D+4i
E	4	20	x_E	x_E+4i

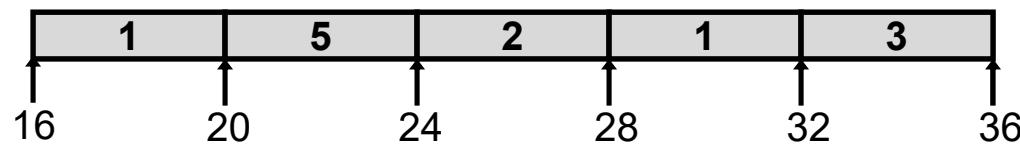
Mảng: Ví dụ

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

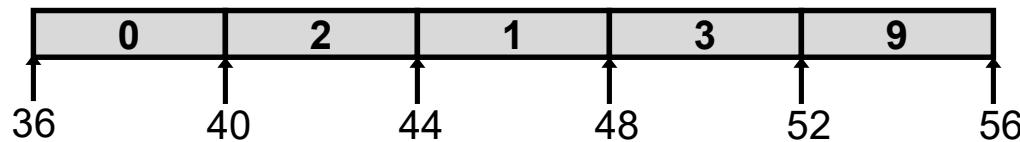
zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig mit = { 0, 2, 1, 3, 9 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```

cmu[3] = 1
cmu[6] = 2
mit[4] = 9
mit[7] = 7

zip_dig cmu;



zip_dig mit;



zip_dig ucb;

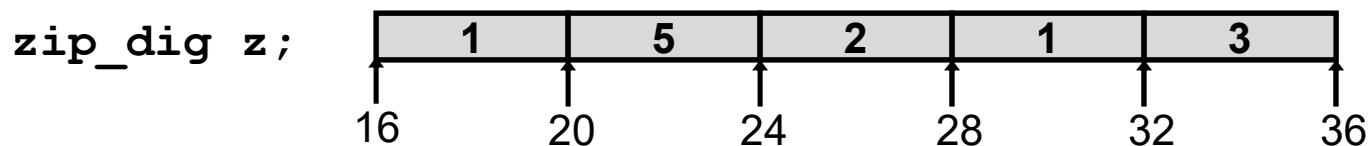


- Khai báo “`zip_dig cmu`” tương đương với “`int cmu[5]`”
- Các mảng ví dụ được cấp phát trong các block 20 bytes liên tiếp nhau
 - Không phải lúc nào cũng đảm bảo như vậy

Truy xuất mảng: Ví dụ

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

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



IA32

```
# %edx = z
# %eax = i
movl (%edx,%eax,4),%eax # z[i]
```

vị trí ($z + 4*i$)

- Thanh ghi `%edx` chứa địa chỉ bắt đầu của mảng
- Thanh ghi `%eax` chứa chỉ số index
- Vị trí của giá trị muốn lấy là $4 * %eax + %edx$
- Sử dụng tham chiếu ô nhớ `(%edx,%eax,4)`

Vòng lặp trong mảng: Ví dụ (IA32)

```
#define ZLEN 5
typedef int zip_dig[ZLEN];
void zincr(zip_dig z) {
    int i;
    for (i = 0; i < ZLEN; i++)
        z[i]++;
}
```

- Dùng chỉ số i để tính toán địa chỉ của phần tử $z[i]$.
- Chỉ số i tăng dần qua từng vòng lặp

1. # edx = z	# Địa chỉ của mảng z
2. movl \$0, %eax	# %eax = i
3. jmp .L3	# goto middle
4. .L4: z[i] (z+4*i)++	# loop:
5. addl \$1, (%edx,%eax,4)	# z[i]++
6. addl \$1, %eax i++	# i++
7. .L3:	# i:5
8. cmpl \$5, %eax	# if !=, goto loop
9. jne .L4	

? Nếu sửa thành short zip_dig[5], đoạn code assembly trên sẽ khác gì?

Dòng code 5 sẽ sửa thành: **addw \$1, (%edx, %eax, 2)**

Vòng lặp với con trỏ: Ví dụ (IA32)

```
void zincr_p(zip_dig z) {  
    int *zend = z+ZLEN;  
    do {  
        (*z)++;  
        z++;  
    } while (z != zend);  
}
```

```
void zincr_v(zip_dig z) {  
    void *vz = z;  
    int dt = 0;  
    do {  
        (*((int *) (vz+dt)))++;  
        dt += ISIZE;  
    } while (dt != ISIZE*ZLEN);  
}
```

- Dùng khoảng cách từ phần tử $z[i]$ đến $z[0]$
- Khoảng cách tăng dần qua từng vòng lặp

1.	# edx = z = vz	# Địa chỉ của z
2.	movl \$0, %eax	# dt = 0
3.	.L8:	# loop:
4.	addl \$1, (%edx,%eax)	# Increment vz+dt
5.	addl \$4, %eax	# dt += 4
6.	cmpl \$20, %eax <small>5*2->10</small>	# Compare dt:20
7.	jne .L8	# if !=, goto loop

? Nếu sửa thành short zip_dig[5], đoạn code assembly trên sẽ khác gì?

Dòng 4: addw \$1, (%edx, %eax)

Dòng 5: addl \$2, %eax

Dòng 6: cmpl \$10, %eax

Vòng lặp trong mảng: Ví dụ (x86_64)

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

```
# %rdi = z
movl    $0, %eax          #   i = 0
jmp     .L3                #   goto middle
.L4:                 # loop:
    addl    $1, (%rdi,%rax,4) #   z[i]++
    addq    $1, %rax          #   i++
.L3:                 # middle
    cmpq    $4, %rax          #   i:4
    jbe     .L4                #   if <=, goto loop
rep; ret
```

Câu hỏi: Mảng 1 chiều

Trong hệ thống 32bit, cho mảng 1 chiều **T A[N]** với **T** là 1 kiểu dữ liệu cơ bản, **N** là số phần tử và đều chưa biết. Biết tổng kích thước của **A** là **20 bytes**.

Tìm **T** và **N**?

$$N \cdot T = 20?$$

4*5 -> INT, * [5],, LONG, FLOAT

2*10 -> SHORT [10]

1*20 -> char A[20]

Mảng 2 chiều (Nested array)

■ Định nghĩa

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

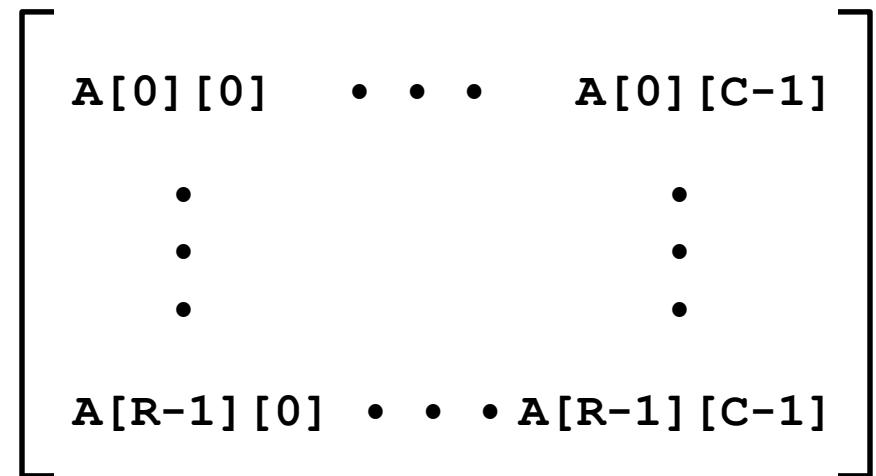
- Mảng 2 chiều của kiểu dữ liệu T
- R dòng, C cột
- Phần tử kiểu T cần K bytes

■ Kích thước mảng

- $R * C * K$ bytes

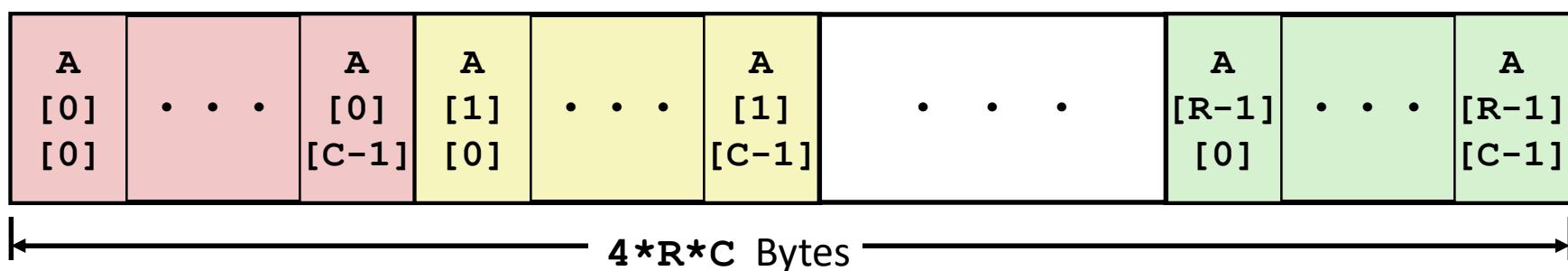
■ Sắp xếp

- Thứ tự Row-Major



`int A[R][C];`

1 MẢNG 1 CHIỀU -> MỖI PHẦN TỬ LÀ 1 MẢNG THEO THỨ TỰ DÒNG



Mảng 2 chiều: Ví dụ

```
#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];
```

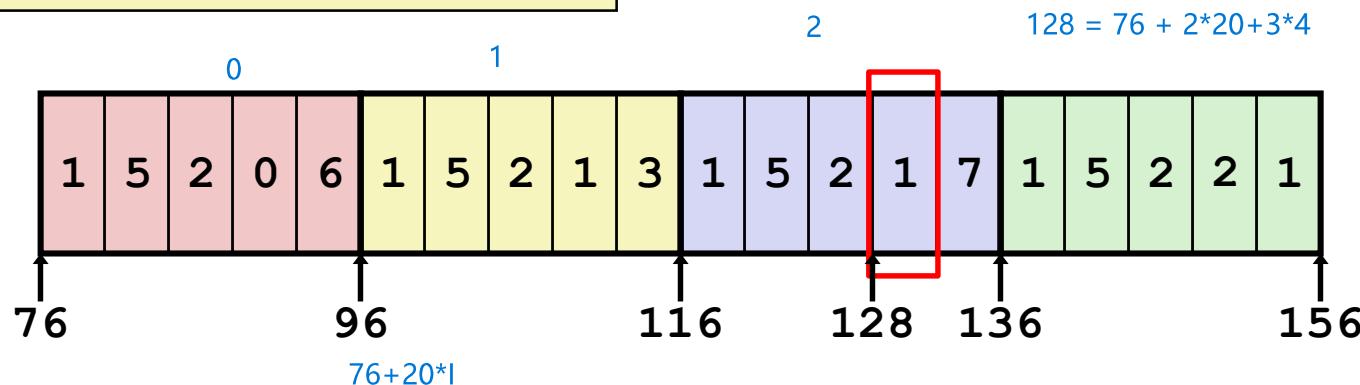
Yêu cầu lấy giá trị 1 trên hình?

Cú pháp trong C?

Xác định địa chỉ?

PGH+20*I+4*J
//I LÀ CHỈ SỐ PHẦN TỬ (CHỈ SỐ HÀNG)
//J CHỈ SỐ CỘT - CHỈ SỐ TRONG MẢNG CON HÀNG

Địa chỉ của pgh[i][i]?



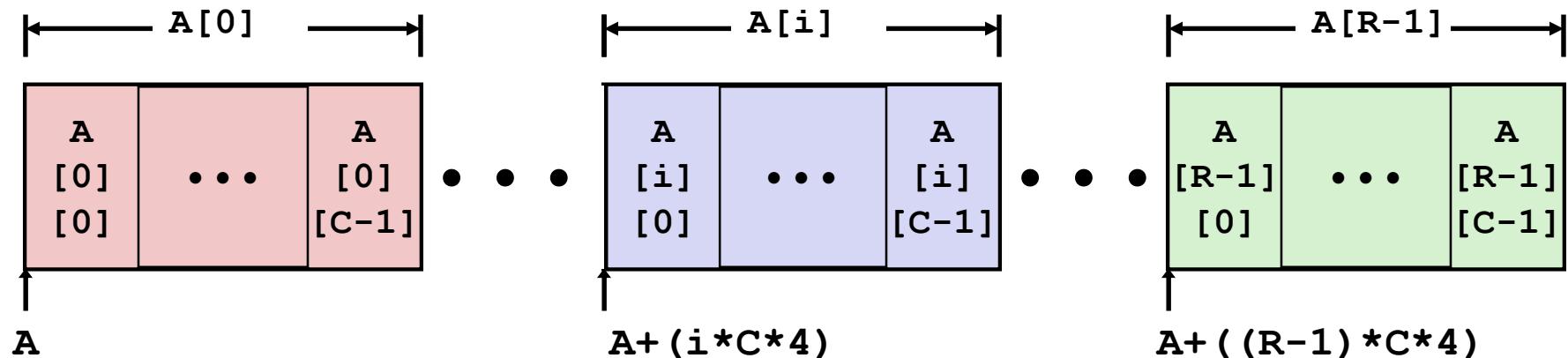
- “zip_dig pgh[4]” tương đương với “int pgh[4][5]”
 - Biến pgh: Mảng 4 phần tử, được cấp phát liên tục
 - Mỗi phần tử là một mảng 5 số int, được cấp phát liên tục
 - **Sắp xếp tất cả các phần tử đảm bảo theo “Row-Major”**

Truy xuất 1 dòng trong mảng 2 chiều

■ Các dòng trong mảng 2 chiều

- $A[i]$ là 1 mảng gồm C phần tử kiểu T
- Mỗi phần tử kiểu T chiếm K bytes.
- Kích thước của 1 mảng nhỏ $A[i]$: $C * K$
- Địa chỉ bắt đầu $A + i * (C * K)$

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

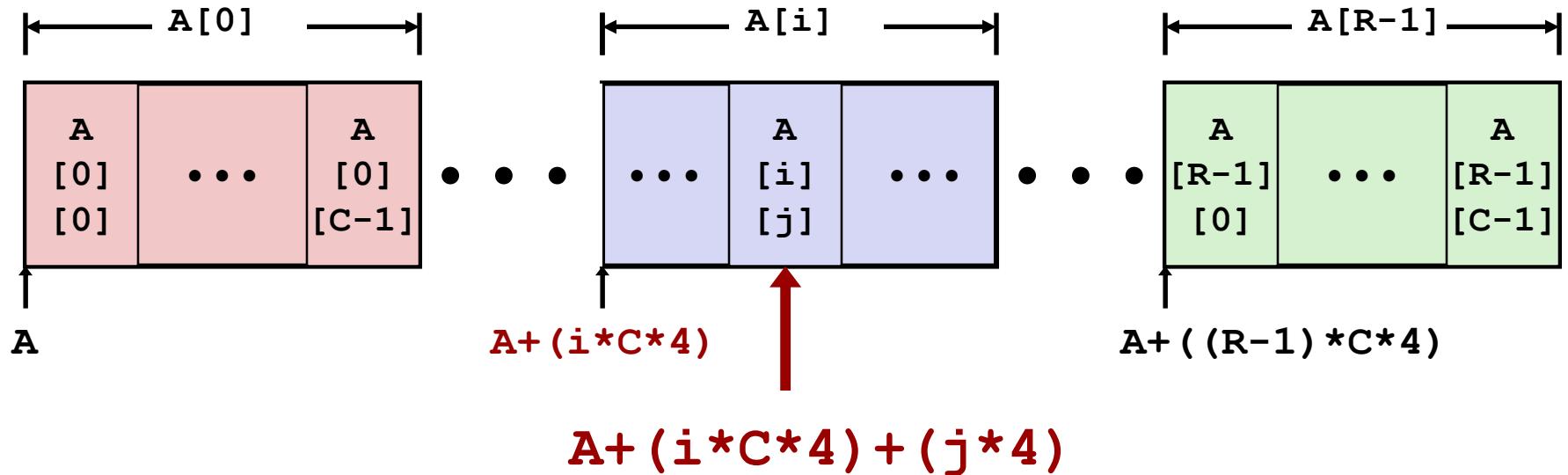


Truy xuất phần tử trong mảng 2 chiều

Các phần tử của mảng

- $A[i][j]$ là phần tử có kiểu T , cần K bytes
- Địa chỉ: $\underbrace{A + i * (C * K) + j * K}_{\text{địa chỉ của } A[i]} = A + (i * C + j) * K$

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



Truy xuất mảng 2 chiều: Ví dụ

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

```
movl 8(%ebp), %eax          # index  
leal (%eax,%eax,4), %eax   # 5*index  
addl 12(%ebp), %eax         # 5*index+dig  # pgh[index]  
movl pgh(,%eax,4), %eax     # offset 4*(5*index+dig)
```

■ Các phần tử của mảng

- `pgh[index] [dig]` có kiểu dữ liệu `int`
- Địa chỉ: $pgh + 20*index + 4*dig$
 $= pgh + 4*(5*index + dig)$

■ IA32 Code

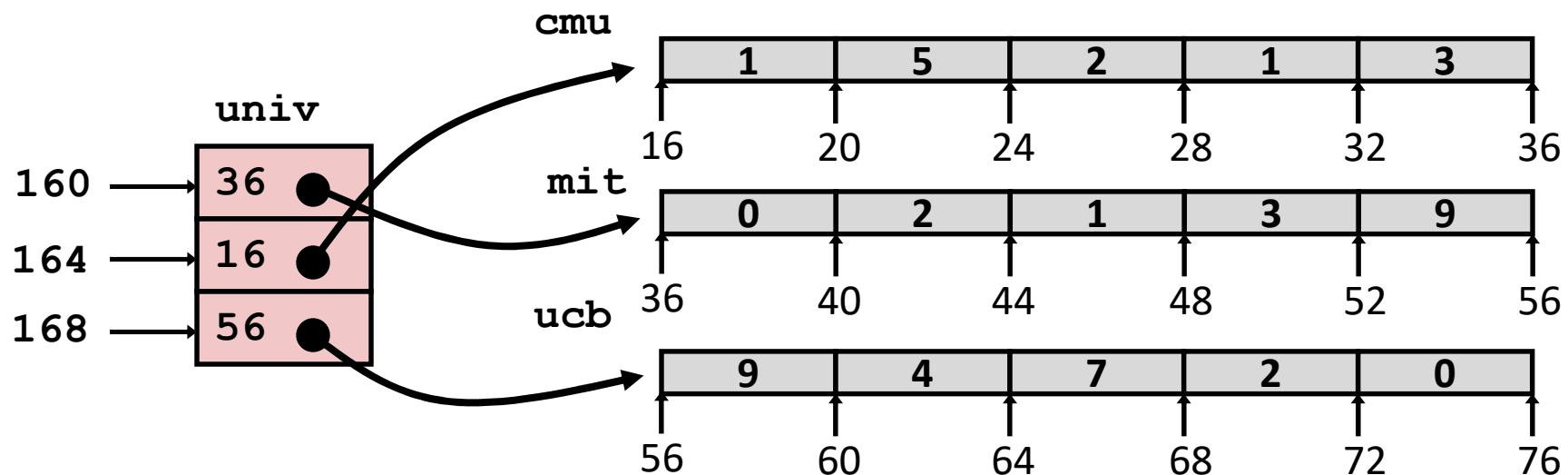
- Tính toán địa chỉ $pgh + 4*((index+4*index)+dig)$

Mảng nhiều cấp (Multi-Level array)

```
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};
```

- Biến `univ` là mảng có 3 phần tử
- Mỗi phần tử là 1 con trỏ
 - 4 (hoặc 8) bytes
- Mỗi con trỏ trỏ đến một mảng số `int`



Truy xuất phần tử trong Multi-Level Array

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

```
movl 8(%ebp), %eax          # index  
movl univ(,%eax,4), %edx    # p = univ[index]  
movl 12(%ebp), %eax          # digit  
movl (%edx,%eax,4), %eax    # p[digit]
```

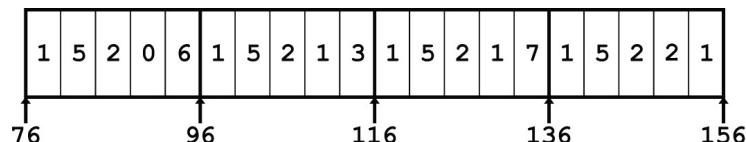
■ Tính toán

- Truy xuất phần tử: **Mem[Mem[univ+4*index]+4*digit]**
- Cần phải đọc bộ nhớ 2 lần
 - Lần 1 để lấy con trỏ trả về đến mảng chứa phần tử
 - Lần 2 truy xuất phần tử trong mảng

Truy xuất phần tử mảng

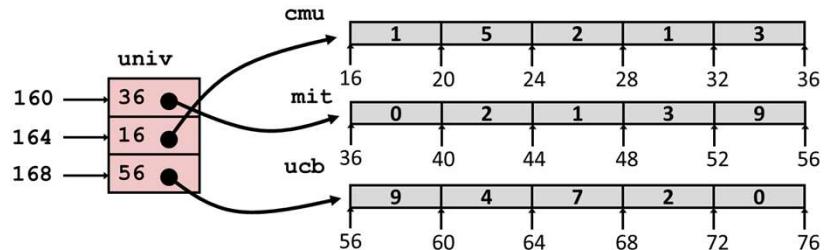
Nested array

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



Multi-level array

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



Truy xuất giống nhau trong C, nhưng cách tính toán địa chỉ khác nhau. Ví dụ trong IA32:

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

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

Ma trận NxN

■ Số chiều cố định

- Số chiều N đã biết khi biên dịch

```
#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];
}
```

■ Số chiều biến đổi, đánh chỉ số tường minh

- Cách truyền thông để hiện thực mảng động

```
#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)];
}
```

■ Số chiều biến đổi, đánh chỉ số ngầm

- Hiện được hỗ trợ trong 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];
}
```

Ví dụ: Truy xuất ma trận 16 X 16

■ Các phần tử của mảng

- Địa chỉ $A + i * (C * K) + j * K$
- $C = 16, K = 4$

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

```
movl 12(%ebp), %edx      # i  
sall $6, %edx            # i*64  
movl 16(%ebp), %eax      # j  
sall $2, %eax            # j*4  
addl 8(%ebp), %eax       # a + j*4  
movl (%eax,%edx), %eax  # *(a + j*4 + i*64)
```

Truy xuất ma trận N x N

■ Các phần tử của mảng

- Địa chỉ $A + i * (C * K) + j * K$
- C = n, K = 4

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

```
movl 8(%ebp), %eax      # n  
sall $2, %eax          # n*4  
movl %eax, %edx        # n*4  
imull 16(%ebp), %edx   # i*n*4  
movl 20(%ebp), %eax    # j  
sall $2, %eax          # j*4  
addl 12(%ebp), %eax    # a + j*4  
movl (%eax,%edx), %eax # *(a + j*4 + i*n*4)
```

Mảng: Bài tập 1

Cho 1 mảng 2 chiều **int array[M][N]** với **M** và **N** chưa biết.

Đoạn mã assembly bên dưới thực hiện **truy xuất phần tử array[i][j]**.

```
1. // %ecx = i, %edx = j
2. movl $array, %eax
3. sall $2, %edx
4. leal (%edx, %eax), %eax
5. movl (%eax, %ecx, 16), %eax      # eax = array[i][j]
```

Chọn nhận định **đúng**?

A. M = 16

B. N = 16

C. N = 4

D. M = 4

Mảng: Bài tập 2

Cho 2 mảng 2 chiều mat1, mat2 với các hằng số M, N và đoạn mã assembly bên dưới của hàm sum_element.

Thử phân tích mã assembly và tìm 2 giá trị cụ thể của M, N?

```
int mat1 [M] [N]
int mat2 [N] [M]

int sum_element(int i, int j) {
    return mat1[i][j] + mat2[j][i];
}
```

```
1.  movl  8(%ebp), %ecx
2.  movl  12(%ebp), %edx
3.  leal  0(,%ecx,8), %eax
4.  subl  %ecx, %eax
5.  addl  %edx, %eax
6.  leal  (%edx,%edx,4), %edx
7.  addl  %ecx, %edx
8.  movl  mat1(,%eax,4), %eax
9.  addl  mat2(,%edx,4), %eax
```

- Tính toán địa chỉ và truy xuất phần tử mat1[i][j]:

- Dòng 3, 4, 5 và 8
- Địa chỉ tính được:

$$\text{mat1} + 4 * (7 * i + j)$$

→ mat1 [M] [7]

→ N = 7

- Tính toán địa chỉ và truy xuất phần tử mat2[j][i]:

- Dòng 6, 7 và 9
- Địa chỉ tính được:

$$\text{mat2} + 4 * (5 * j + i)$$

→ mat2 [7] [5]

→ M = 5

Mảng: Bài tập 3

Trong hệ thống 32 bit, cho một ma trận $T A[N][N]$ với T là kiểu dữ liệu và N là hằng số chưa biết.

Cho địa chỉ của A là $0x1000$, địa chỉ lưu của phần tử $A[2][2]$ là $0x101C$.

Tìm T và N ?

$$\begin{aligned} 0x101c &= 0x1000 + (2*C=2)*K \\ \text{hoặc} \\ 0x101c &= 0x1000 + (2*C*N) + 2 \\ \rightarrow 28(10) &= 2*N*(C+1) \\ \rightarrow 14 &= N*(c+1) \\ \rightarrow 2,6 & \\ \rightarrow 1,13 & \end{aligned}$$