X86 基本指令简介

传送类指令

指令的主要类别

传送类指令

例如：从存储器到寄存器、从寄存器到输入输出接口

运算类指令

例如：加、减、乘、除、与、或、非等运算

控制类指令

例如：暂停处理器、清除标志位等

转移类指令

例如：无条件转移、条件转移、过程调用等

操作数的长度

64位 linux 编译器

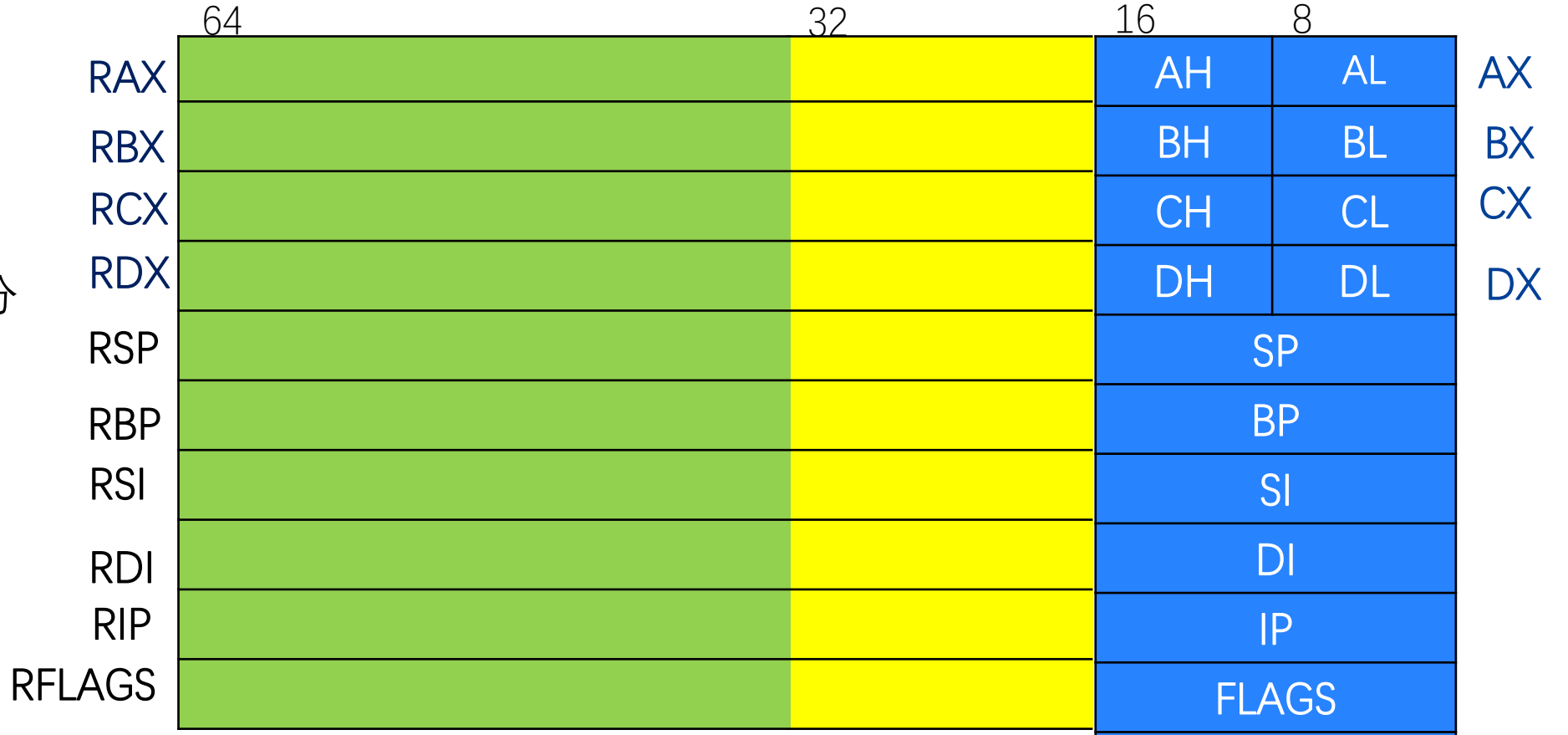
- “Integer” 类数据
  - 字符型 char、 unsigned char、signed char： 1 byte
  - 短整型 short 、unsigned short、signed short: 2 bytes
  - 整型 int: 4 bytes,
  - 长整型 long: 8 bytes （编译器不同，长度可能不同）
  - 指针类型（地址） 8 bytes （编译器不同，长度可能不同）
- “Floating point “ 浮点类数据
  - 单精度浮点型 float: 4bytes
  - 双精度浮点型 double: 8 bytes
  - 扩展精度浮点型 long double: 10 bytes， （编译器不同，长度可能不同）
- 复合类型，例如 数组（ array ）或 结构（structure ）
  - 在内存中连续地分配字节，并对齐存放

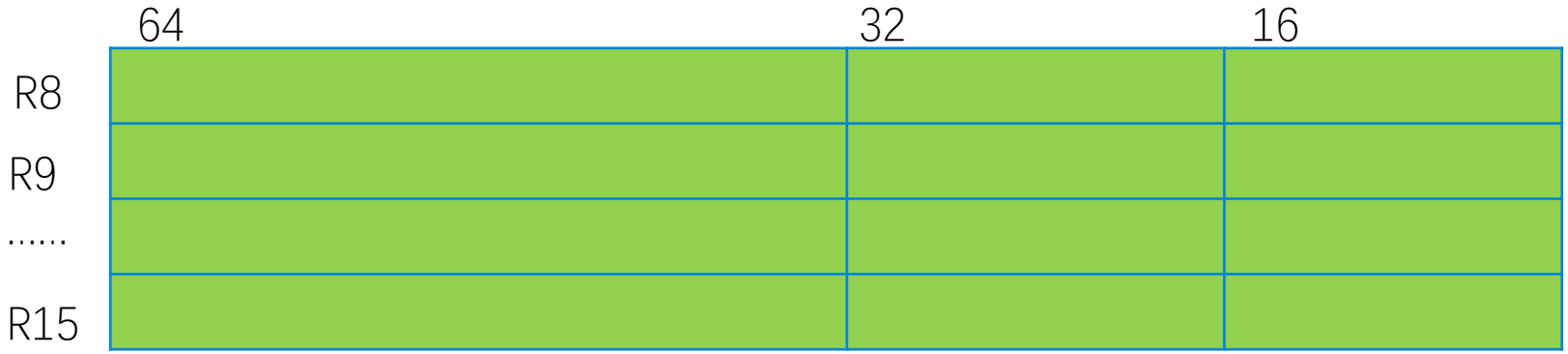
X86-64 寄存器

X86-64 寄存器新增部分

IA32 寄存器新增部分

8086寄存器模型





X86 传送指令

指令	助记符	功能	操作数类型
通用数据传送指令	MOV	传送	字节/字
	PUSH	压栈	字
	POP	出栈	字
	XCHG	交换	字节/字
地址传送指令	LEA	装入有效地址	字
	LDS	将指针装入寄存器和DS	4个字节
	LES	将指针装入寄存器和ES	4个字节
标志传送指令	LAHF	把标志装入AH	字节
	SAHF	把AH送标志寄存器	字节
	PUSHF	标志压栈	字
	POPF	标志出栈	字
累加器专用传送指令	XLAT	换码	字节
	IN	输入	字节/字
	OUT	输出	字节/字

本节主要介绍

- 数据传送指令 MOV
- 地址传送指令 LEA

数据传送指令 mov使用举例

- `movq $0x4,%rax`
- `movq %rax,%rdx`

- `long temp1,temp2;`
- `temp1 = 0x4;`
- `temp2 = temp1;`
- q: 操作数长度为64位

movq 与 movl 对比

- `movq $0x4,%rax`
- `movq %rax,%rdx`

- `long temp1,temp2;`
- `temp1 = 0x4;`
- `temp2 = temp1;`
- q：操作数长度为64位

- `movl $0x4,%eax`
- `movl %eax,%edx`

- `int temp1,temp2;`
- `temp1 = 0x4;`
- `temp2 = temp1;`
- l：操作数长度为32位

- 后缀：q: 64位，四字；l: 32位，双字；w: 16位；b: 8位

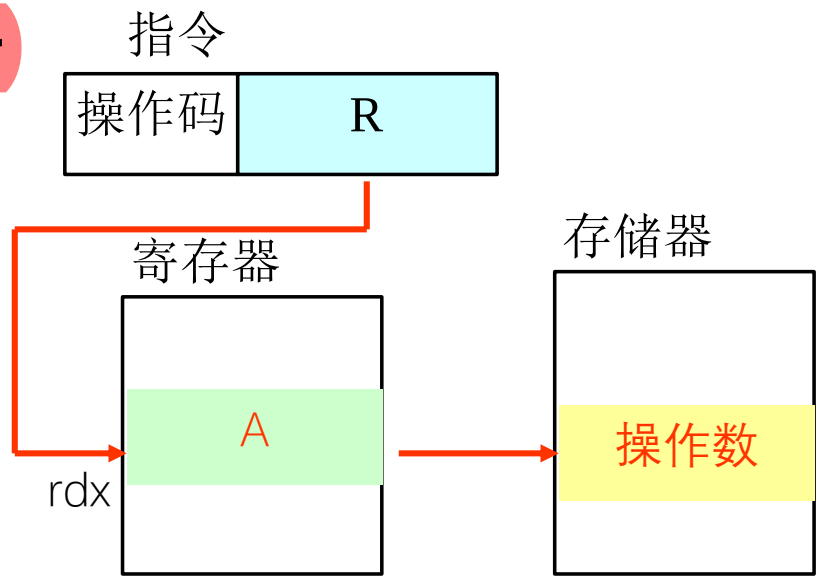
数据传送指令：mov 操作数的组合

源(source)	目(Dest)	Src, Dest	C语言代码
movq q:64位			
Imm (立即数)	*Reg*	`movq $0x4,%rax`	`temp = 0x4;`
	Mem	`movq $-147,(%rax)`	`*p = -147;`
Reg (寄存器)	*Reg*	`movq %rax,%rdx`	`temp2 = temp1;`
	Mem	`movq %rax,(%rdx)`	`*p = temp;`
存储器 (Mem)	*Reg*	`movq (%rax),%rdx`	`temp = *p;`

不能在一条指令中实现存储器单元到存储器单元之间的传送

存储操作数的常见寻址方式

- **寄存器间接寻址：(R)**
- 代表访问 Mem[Reg[R]]
  - **寄存器 R 中存放的是操作数的内存地址**
    - **可以表示 C语言中指针所指向的数据**



例如：

```
movq %rax,(%rdx)
```

表示： `*p = temp;`

操作数寻址方式举例

```
void swap
    (long *xp, long *yp)
{
  long t1 = *xp;
  long t2 = *yp;
  *xp = t2;
  *yp = t1;
}
```

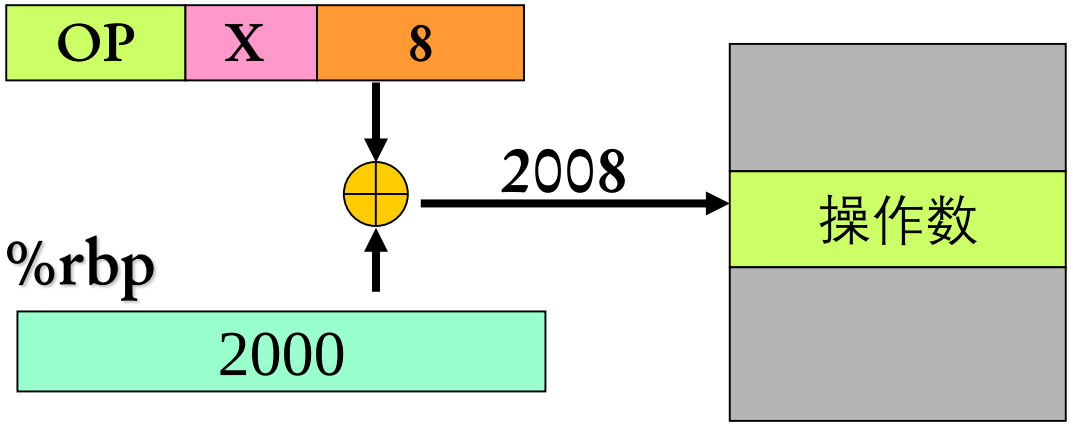
```
swap:
    movq    (%rdi), %rax
    movq    (%rsi), %rdx
    movq    %rdx, (%rdi)
    movq    %rax, (%rsi)
    ret
```

(%rdi), 寄存器间接寻址

(%rsi), 寄存器间接寻址

存储操作数的常见寻址方式

- 基址/变址寻址：D(R) 代表访问
  - 例如：Mem[Reg[R]+D]
  - 寄存器 R 表示某个存储区域的起始位置
  - 常数 D 表示偏移量（offset）




```
movq 8(%rbp),%rdx
```

存储器操作数的寻址方式：一般形式

```
movq $-147,(%rax)          *p = -147;
```

- 一般形式：D(Rb,Ri,S)　　表示：　Mem[Reg[Rb]+S*Reg[Ri]+ D]
  - D:　常量，表示“偏移量”，长度可以为1, 2, 或 4 bytes
  - Rb:　基址寄存器: 任意16个通用寄存器中的一个
  - Ri:　索引寄存器: 任意一个寄存器，除了 **%rsp**
  - S:　比例因子(Scale): 可以是1, 2, 4, 或 8 (为什么是这个比例？字对齐)

特殊情况：	(Rb,Ri)	Mem[Reg[Rb]+Reg[Ri]]
	D(Rb,Ri)	Mem[Reg[Rb]+Reg[Ri]+D]
	(Rb,Ri,S)	Mem[Reg[Rb]+S*Reg[Ri]]

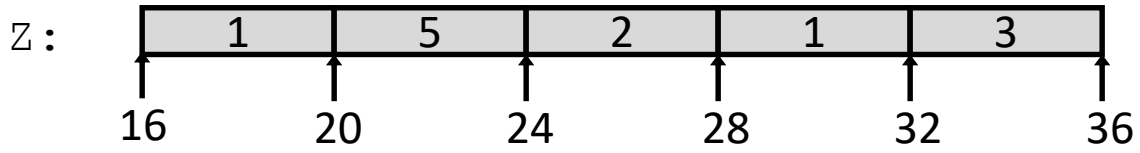
更多地址计算方式：举例

一般形式：D(Rb,Ri,S) 表示： Mem[Reg[Rb]+S*Reg[Ri]+ D]

%rdx	0xf000
%rcx	0x0100

表达式	计算	地址
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx,%rcx)	0xf000 + 0x100	0xf100
(%rdx,%rcx,4)	0xf000 + 4*0x100	0xf400
0x80(,%rdx,2)	2*0xf000 + 0x80	0x1e080

存储器操作数寻址方式举例：数组元素的访问




```
int z[5]= { 1, 5, 2, 1, 3 };,
int digit;

int x = z[digit];
```

```
   # %rdi = z
   # %rsi = digit
   # %eax = x      //x 是一个32位的int
movl (%rdi,%rsi,4), %eax  # z[digit]
```

- %rdi 存放数组的起始地址
- %rsi 存放数组下标（array index）
- 需要访问的元素所在的内存地址：
  %rdi + 4*%rsi
- 使用(%rdi,%rsi,4)访问该单元

不同长度操作数的传送

- 类型声明为：
  - src_type x;
  - dst_type *p;
- C语言赋值语句：
  - *p=(dst_type) x;
- x对应的寄存器
  - RAX或EAX
  - AX或AL
- p对应的寄存器
  - RDX

src_type	dst_type	机器代码
char	int	
int	char	
int	unsigned	
short	int	
unsigned char	unsigned	
char	unsigned	
int	int	
long	long	
long	int	

给定src_type 和 dst_type 的类型组合，赋值语句对应的机器级代码

不同长度操作数的传送

- 类型声明为：
  - src_type x;
  - dst_type *p;
- C语言赋值语句：
  - *p=(dst_type) x;
- x对应的寄存器
  - RAX或EAX
  - AX或AL
- p对应的寄存器
  - RDX (64位地址)

给定src_type 和 dst_type 的类型组合， 赋值语句对应的机器级代码

src_type	dst_type	机器代码
char	int	movsbl %al, (%rdx)
int	char	movb %al, (%rdx)
int	unsigned	movl %eax, (%rdx)
short	int	movswl %ax, (%rdx)
unsigned char	unsigned	movzbl %al, (%rdx)
char	unsigned	movzbl %al, (%rdx)
int	int	movl %eax, (%rdx)
long	long	movq %rax, (%rdx)
long	int	movl %eax, (%rdx)

- 后缀：代表操作数长度
  - q: 64位， 四字； l: 32位， 双字； w: 16位； b: 8位

X86-64 地址传送指令

- **leaq** *Src*, *Dst*
  - *Src* 可为地址表达式，　*Dst* 一般为寄存器
  - 仅计算地址，不访问地址所指向的内存单元
  - 将计算出来的地址传送给指定寄存器

```
int *p;
p = &x[i];
# %rbx = x
# %rcx= i
# %rdx = p
leaq (%rbx,%rcx,4), %rdx
```

```
int y;
y = x[i];
# %rbx = x
# %rcx= i
# %eax = y
movl (%rbx,%rcx,4), %eax
```

用leaq 来完成算术运算

计算数学表达式： x + k*y; k = 1, 2, 4, or 8

例如：

```
long m12(long x)
{
  return x*12;
}
```

编译后生成:

```
leaq (%rdi,%rdi,2), %rax  # t <- x+x*2
salq $2, %rax             # return t<<2
```

多维数组

T **A**[*R*][*C*];

- *R* 行, C列
- 类型 T的元素需要 K bytes

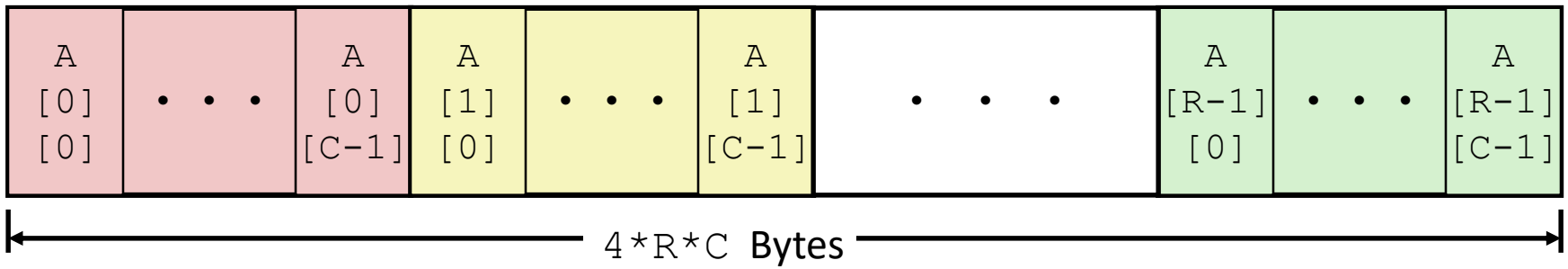
- 数组的大小
  - *R* * *C* * *K* bytes
- 内存中的存放：行主序

$$\begin{bmatrix} A[0][0] & \cdots & A[0][C-1] \\ \vdots & & \vdots \\ A[R-1][0] & \cdots & A[R-1][C-1] \end{bmatrix}$$

R 行

C列

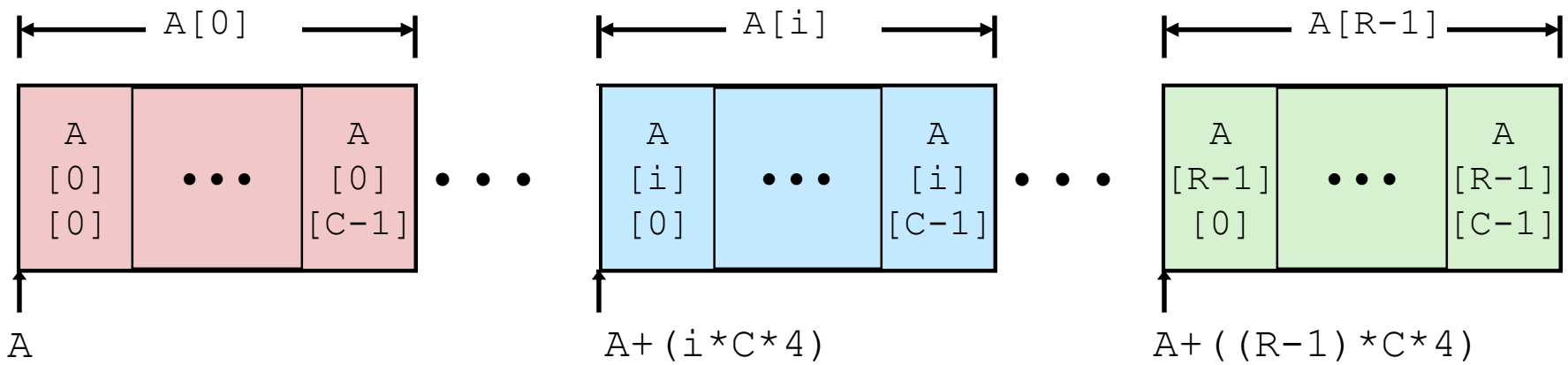
int A[R][C];



二维数组的行

- 行向量
  - **A[i] 是一个有C个元素的数组**
  - **每个元素属于类型T，占用 *K* bytes**
  - **A[i] 的内存地址：** ***A + i * (C * K)***

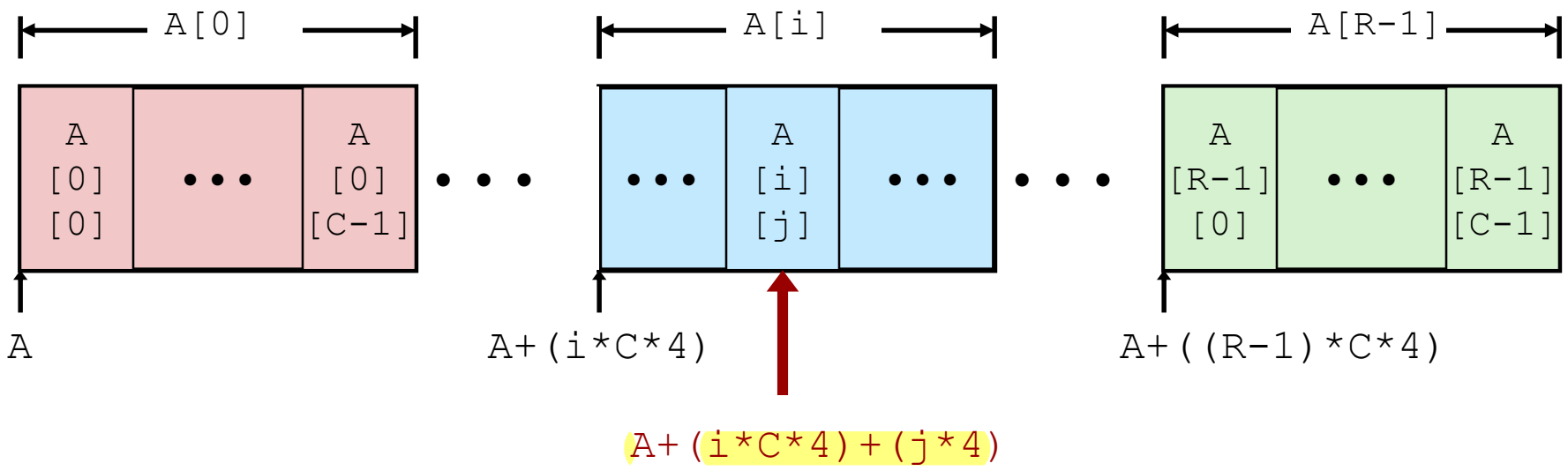
```
int A[R][C];
```



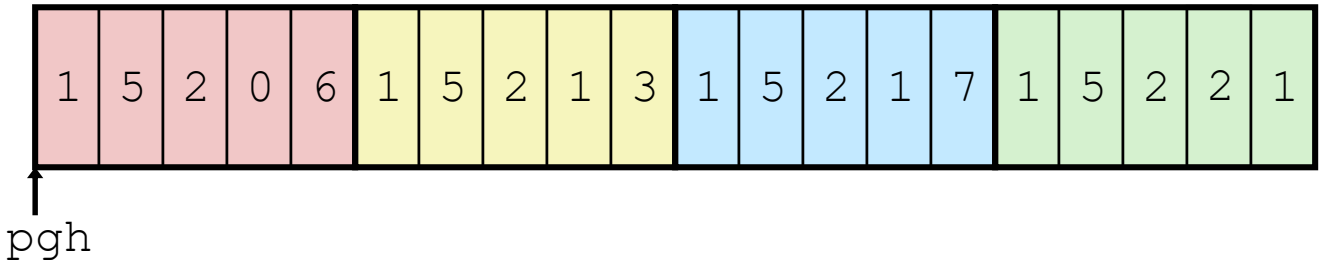
访问二维数组的某个元素

- Array Elements
  - **`A[i][j]`** is element of type *T,* which requires *K* bytes
  - Address **`A +`** $i * (C * K) + j * K = A + (i * C + j) * K$

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



访问二维数组的某个元素：代码




```
int pgh[4][5];
int index, int dig;
int x=pgh[index][dig];
}
```

- 数组元素的地址
  - **pgh[index][dig] 是一个 int**
  - 地址: **pgh + 20*index + 4*dig**
    - = **pgh + 4*(5*index + dig)**

```
leaq (%rdi,%rdi,4), %rax # 5*index
addl %rax, %rsi          # 5*index+dig
movl pgh(,%rsi,4), %eax  # M[pgh + 4*(5*index+dig)]
```

小结

- X86-64 数据传送指令
- X86-64 地址传送指令
- 使用 地址传送指令 完成 算术运算

谢谢！