3.1.1 数据存储的基本形式

关于变量的定义：

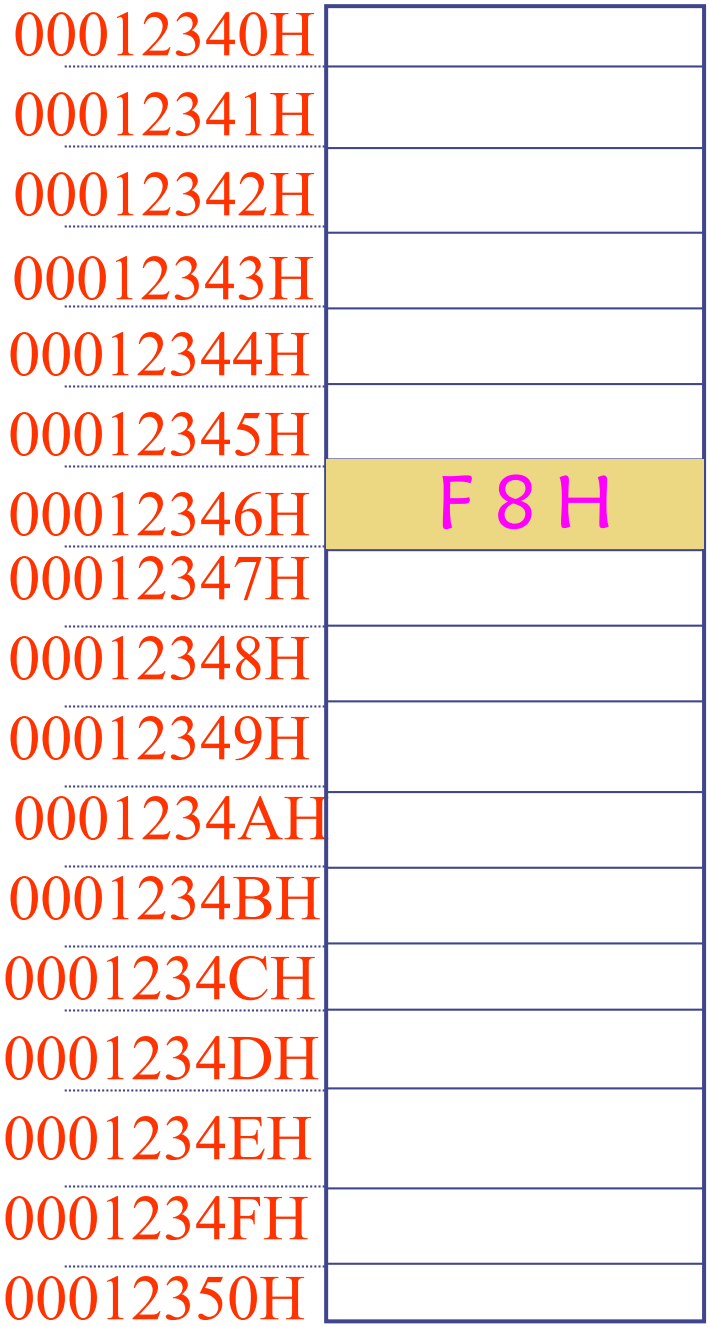
C 语 言：	char、	short、	int、	double
对应长度：	1个、	2个、	4个、	8个字节
	字节	字	双字	四字
汇编语言：	byte	word	dword	qword

内存条：用来存放程序和数据的装置

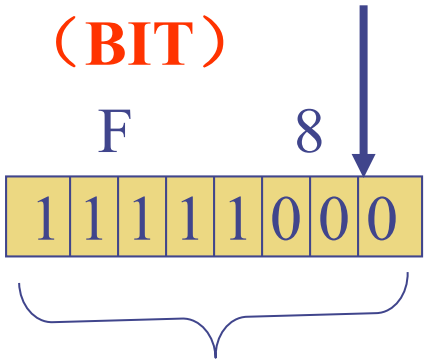
字节是最小的寻址单位

每一个字节都有一个地址

物理地址 (Physical Address, PA) 是唯一的



主存的基本存储单位是位（BIT）



8个位组成一个字节 BYTE

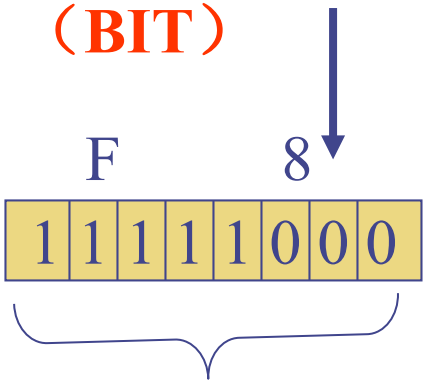
Q: 1M字节内存，地址编码需要多少二进制位？

Q:32位地址对应的内存大小可达到多大？

字节是最小的寻址单位

00012340H	
00012341H	
00012342H	
00012343H	
00012344H	
00012345H	
00012346H	F8H
00012347H	04H
00012348H	56H
00012349H	12H
0001234AH	
0001234BH	
0001234CH	
0001234DH	
0001234EH	
0001234FH	
00012350H	

主存的基本存储单位是位（BIT）



8个位组成一个字节 **BYTE**

字地址是这2个字节中低字节的地址

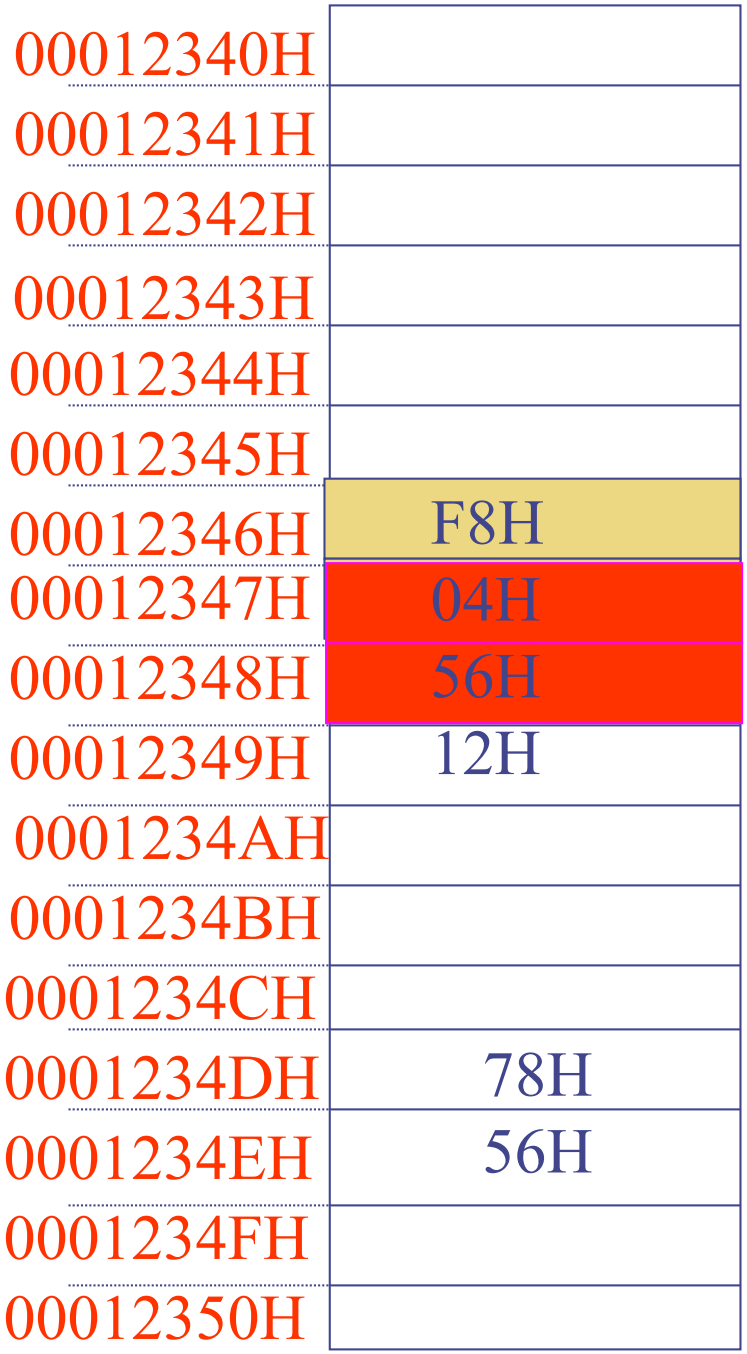
两个相邻的字节组成一个字 **WORD**

Q1:两个黄色的字节组成的字的地址是多少？字中的内容又是多少？

PA:00012346H
DATA: 04F8H

Q2:红色的呢？

PA:00012347H
DATA: 5604H



字数据的存放形式:

低8位在低字节；高8位在相邻的高字节中。

Q3:将5678H存放到地址为1234D的字单元中。

地址为12346H的双字是：
44434241H
即(00012346H) = 44434241H

地址为12346H的字是：
4241H
即(00012346H) = 4241H

地址为12346H的字节是：
41H
即(00012346H) = 41H

双字
四个连续的字节组成

其地址为四个字节中的最低字节的地址。

3.1.1 数据存储的基本形式

数据存储方法有两种

- 小端存储（Little Endian）
- 大端存储（Big Endian）

➢ Intel x86 系列采用小端存储方式

➢ 在小端存储方式中，最低地址字节中存放数据的最低字节，最高地址字节中存放数据的最高字节。按照数据由低字节到高字节的顺序依次存放在从低地址到高地址的单元中。

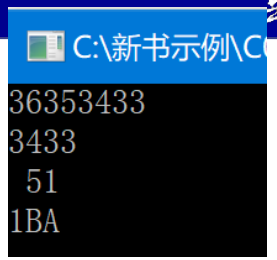
➢ 大端存储方式与小端存储方式相反。

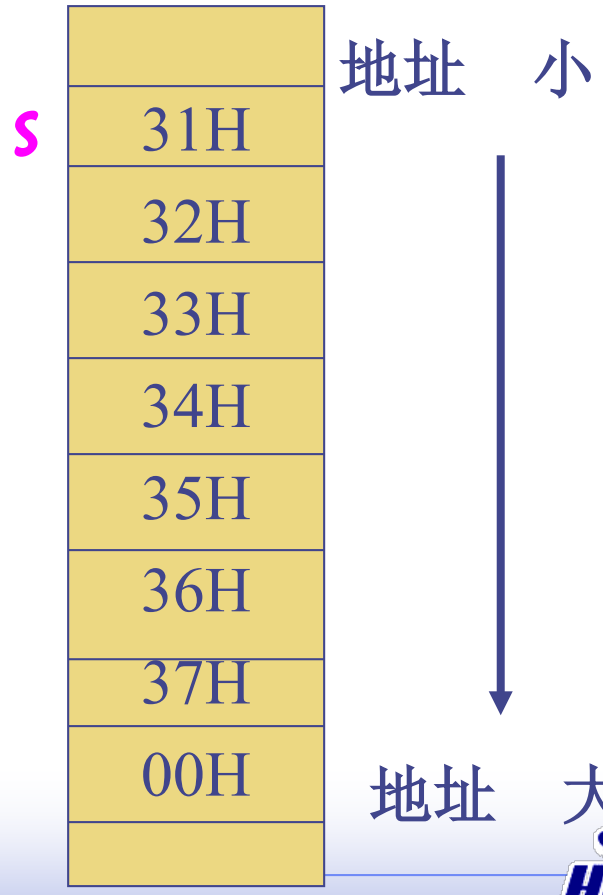
3.1.2 数据地址的类型及转换

華中科技大學

程序运行结果是什么？为什么？

```
char   s[10];
strcpy(s,"1234567");

printf("%x \n",  *(long *)(s+2));

printf("%x \n",  *(short *)(s+2));

printf("%d \n",  *(char *)(s+2));

*(int *)(s+1)=16706;
printf("%s \n",s);
```



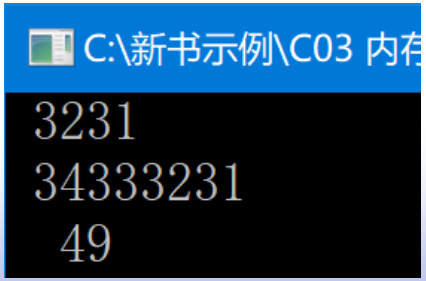


关键词：地址类型转换

工程 type_convert

3.1.2 数据地址的类型及转换


```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
union test {
    char s[10];
    char c;
    short x;
    int  y;
}temp;
int main()
{   strcpy(temp.s, "1234567");
    printf(" % x \n", temp.x);
    printf(" % x \n", temp.y);
    printf(" % d \n", temp.c);
    return 0;
}
```



工程：union_type

3.1.2 数据地址的类型及转换


```
union test {
    char s[10];
    char c;
    short x;
    int   y;
}temp;
strcpy(temp.s, "1234567");
```

工程：union_type

监视 1

搜索(Ctrl+E)　搜索深度: 3

名称	值	类型
&temp.x	0x0092a138 {union_type.exe!test temp} {12849}	short *
&temp.y	0x0092a138 {union_type.exe!test temp} {875770417}	int *
&temp.c	0x0092a138 "1234567"	char *
temp.s	0x0092a138 "1234567"	char[10]

3.3 字符数据在机内的表示形式

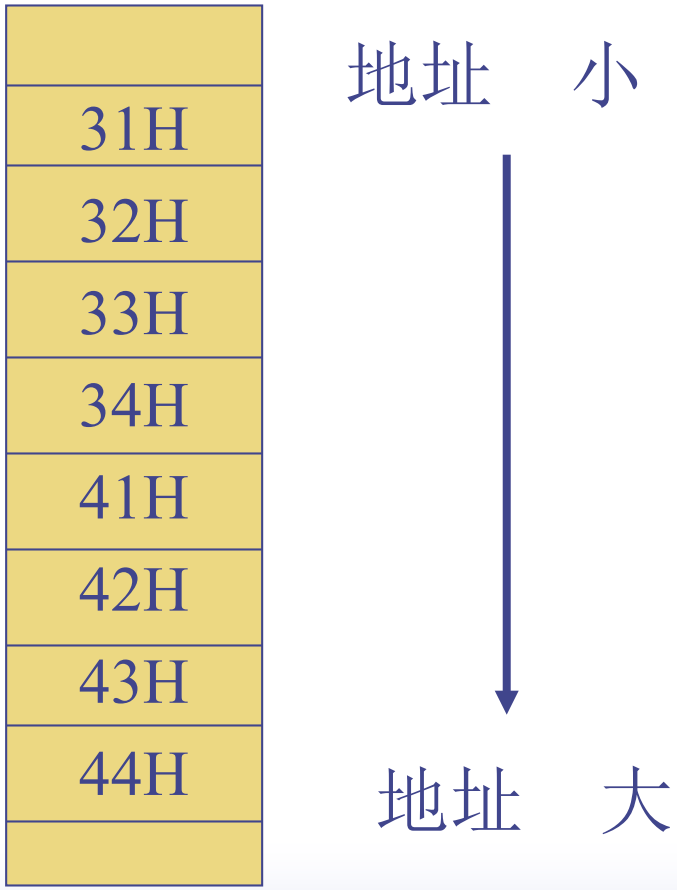
ASCII码字符表

		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
↓		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0				**0**		**P**		**p**								
1	1				**1**	**A**	**Q**	**a**	**q**								
2	2				**2**	**B**	**R**	**b**	**r**								
3	3				**3**	**C**	**S**	**c**	**s**								
4	4			**$**	**4**	**D**	**T**	**d**	**t**								
5	5				**5**	**E**	**U**	**e**	**u**								
6	6				**6**	**F**	**V**	**f**	**v**								
7	7				**7**	**G**	**W**	**g**	**w**								
8	8				**8**	**H**	**X**	**h**	**x**								
9	9				**9**	**I**	**Y**	**i**	**y**								
10	A	换行				**J**	**Z**	**j**	**z**								
11	B					**K**		**k**									
12	C					**L**		**l**									
13	D	回车				**M**		**m**									
14	E					**N**		**n**									
15	F					**O**		**o**									

American Standard Code for Information Interchange

3.3 字符数据在机内的表示形式

字符串
‘1234ABCD’
的表示结果：



3.3 字符数据在机内的表示形式

Q: 在键盘上输入123。计算机中得到是什么呢？
若要用其作数值运算，如何办呢？

‘123’的字符串形式　　123的数值形式

3.4.1 数据定义伪指令

[变量名]　数据定义伪指令　表达式[，…]

db、byte、sbyte	字节类型
dw、word、sword	字类型
dd、dword、sdword	双字类型
dq、qword、sqword	四字类型
dt、tbyte	10个字节类型
real4	单精度浮点数类型
real8	双精度浮点数类型

表达式有5种形式

3.4.2 表达式

[变量名]　数据定义伪指令　表达式[，…]

数据定义伪指令：DB，DW，DD,DQ，SBYTE ……

表达式的4种形式

- 数值表达式
- 字符串
- 地址表达式
- 重复子句 n DUP（表达式[，…]）

3.4 数据段定义

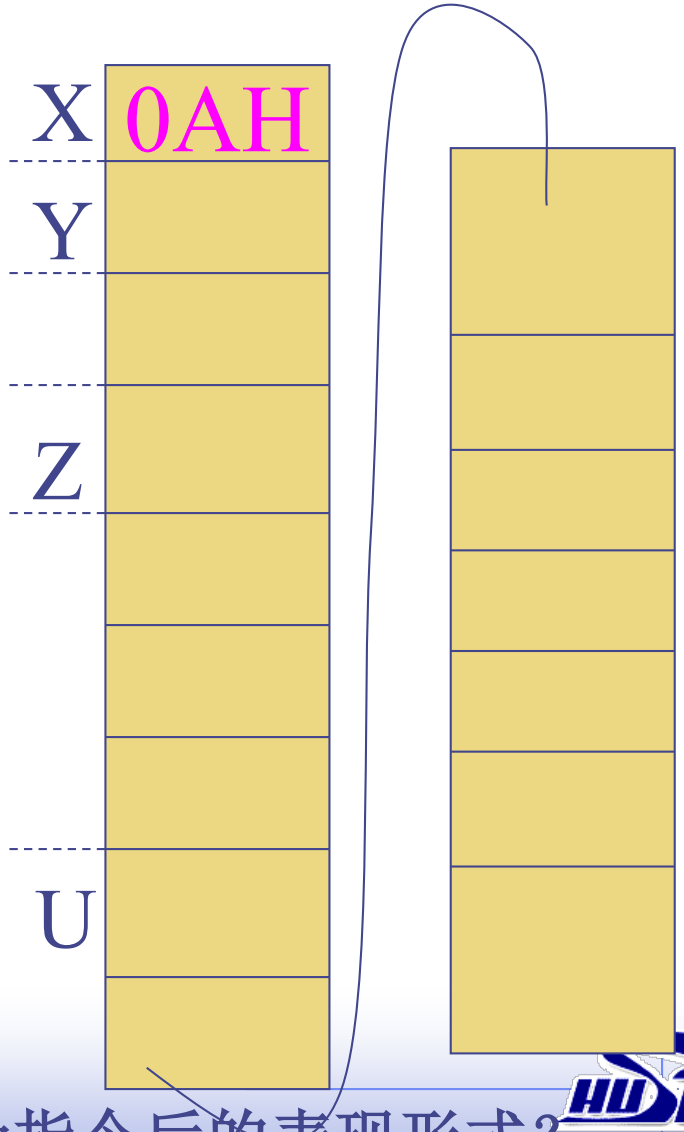
华中科技大学

数据定义伪指令——数值表达式

```
X   DB    10
Y   DW    10
Z    DD    12345678H
U    DQ    12345678H
V    DT    12345678H
```

如下定义正确吗？

~~K DB 1234H~~



Q：同一个数值表达式在不同数据定义伪指令后的表现形式？

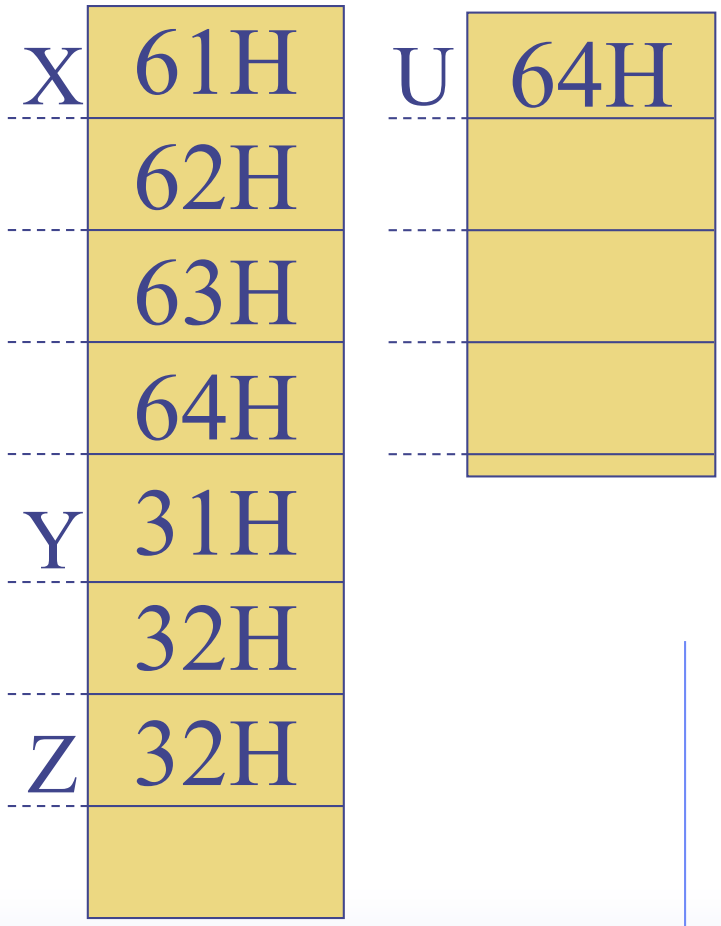
3.4 数据段定义

- 数据定义伪指令——字符串

```
X  DB  ‘abcd’
Y  DB  ’12’
Z  DW  ’12’
U  DD  ‘abcd’
```

如下定义正确吗？

~~K DW ‘1234’~~



- 数据定义伪指令——地址表达式

地址表达式：由变量、标号、常量、[R] 和运算符组成的有意义的式子。

在数据定义语句中，不能出现带寄存器符号的地址表达式。

3.4 数据段定义

■数据定义伪指令　地址表达式

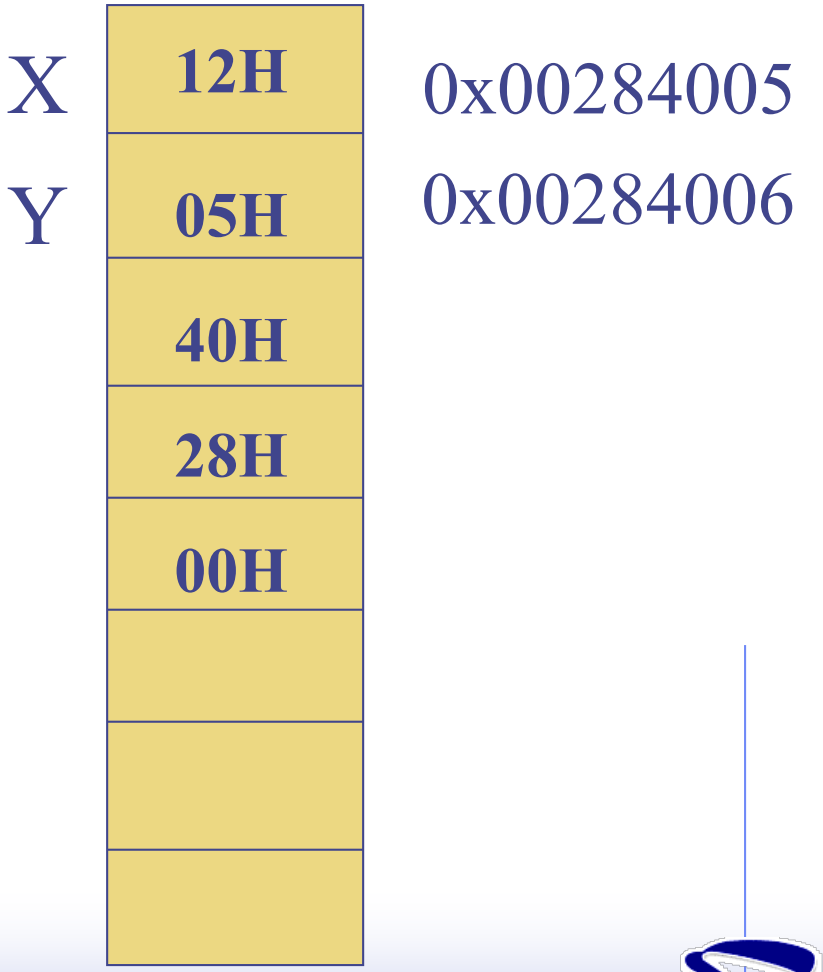
➢ **DD 变量或标号**

变量定义在32位段中，偏移地址是32位的，用变量的偏移地址来初始化相应双字单元。

3.4 数据段定义

地址表达式的数据存放

```
.DATA
X   DB    12H
Y    DD    X
```



地址表达式出现在 DD后, 其意义是什么？

3.4 数据段定义

- 数据定义伪指令　重复子句

  N DUP (表达式[，表达式]…)

  例1：　X DB 3 DUP (2)

  　　X DB 2, 2, 2

3.4 数据段定义

■数据定义伪指令　重复子句　重复子句的嵌套和展开

例2：　Y　DB　3 DUP (1,2)

Y　DB　1, 2, 1, 2, 1, 2

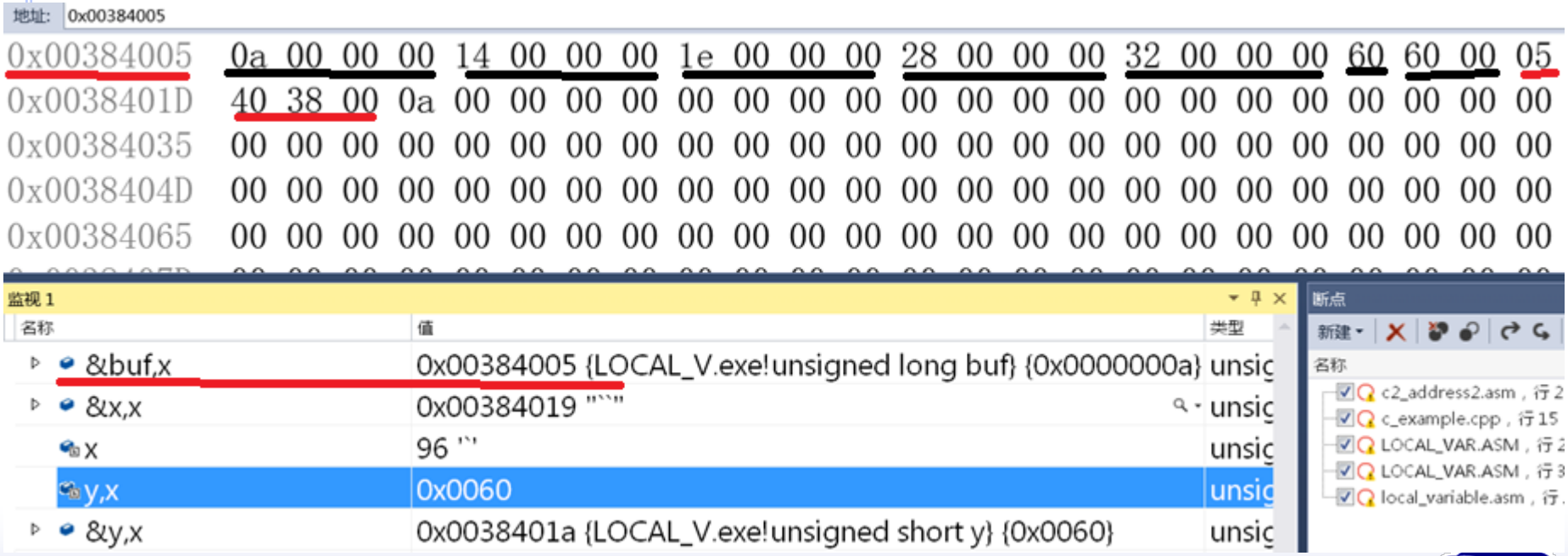
例3：Z　DB　3 DUP (1, 2 DUP (2))

Z　DB　3 DUP (1, 2 ,2　)

Z　DB 1, 2, 2, 1, 2, 2, 1, 2, 2

3.4 数据段定义


```
buf     dd   10, 20, 30, 40, 50
  x     db   60H
  y     dw   60H
  z     dd   BUF
```

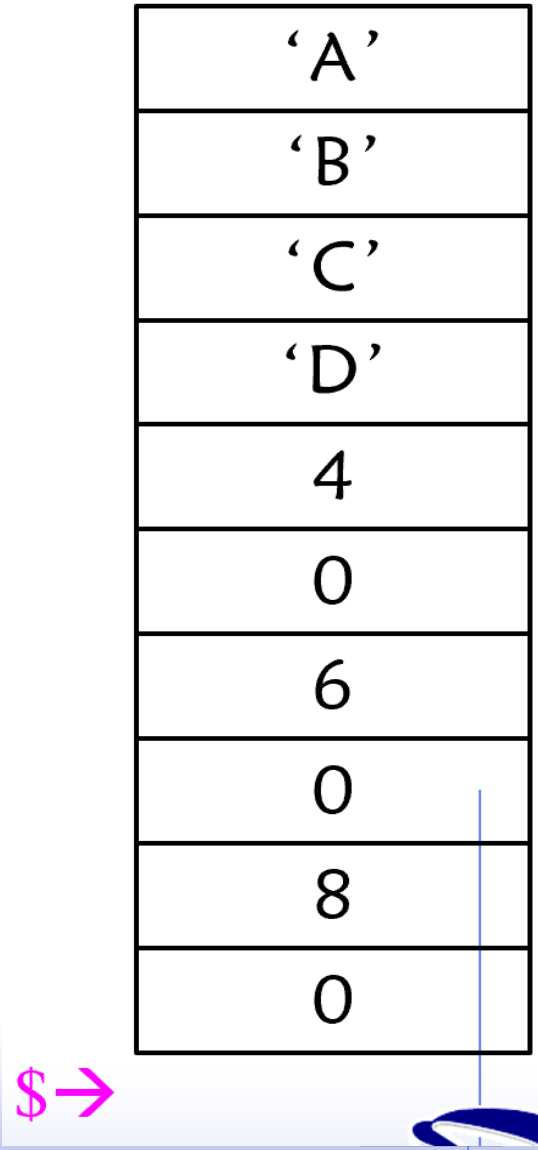


3.4 数据段定义

3.4.3 汇编地址计数器

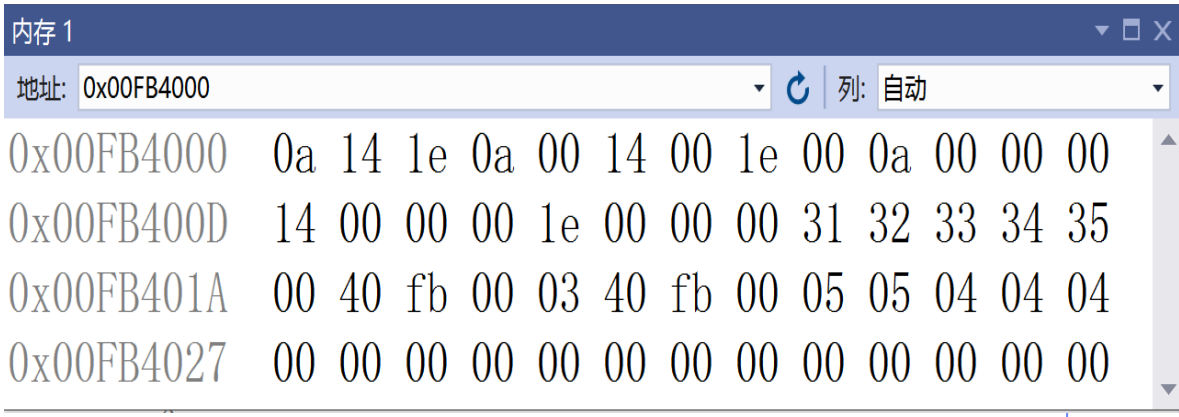
用来记录当前拟分配的存储单元的地址

```
x  db  'ABCD'
y  dw  $-x              ;  4
z  dw  $-x,  $-x        ;  6,  8

buf   dw  20,  30,   40
len = ($-buf)/2    ; 该值为buf中字数据的个数。
```



3.4 数据段定义

```
.686P
.model flat, stdcall
  ExitProcess proto  :dword
  includelib  kernel32.lib
.data
  x  db  10, 20, 30
  y  dw  10, 20, 30
  z  dd  10, 20, 30
  u  db  '12345'
  u_len  = 5
  p  dd  x,  y
  q  db  2 dup (5), 3 dup (4)
.stack 200
.code
 start:
    invoke ExitProcess, 0
end start
```



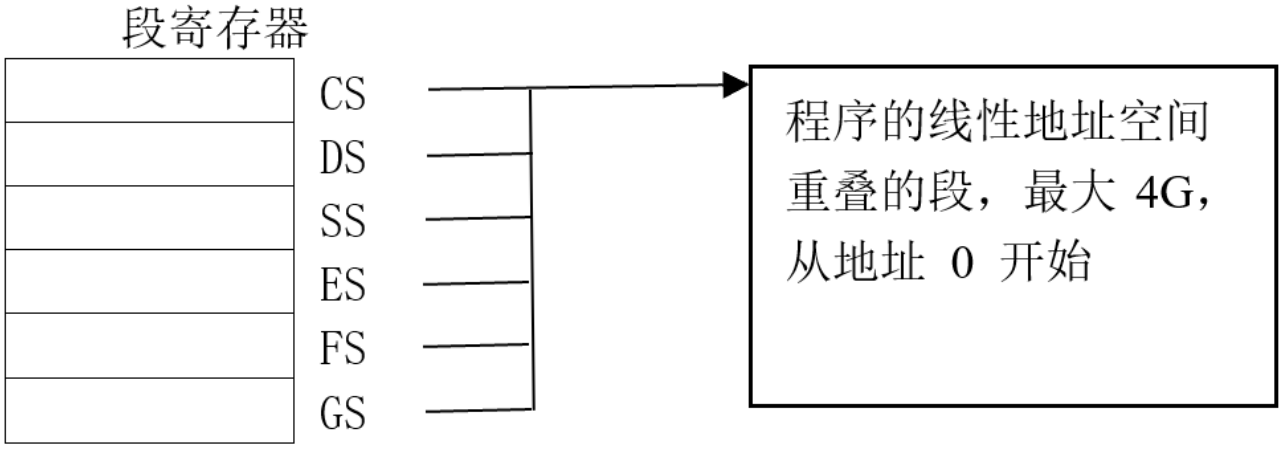
3.4 数据段定义

x	0aH	00FB4000H
	14H	
	1eH	
y	0aH	00FB4003H
	00H	
	14H	
	00H	
	1EH	
	00H	
z	0aH	00FB4009H
	00H	
	00H	
	00H	
	14H	00FB400DH
	00H	
	00H	
	00H	

	0eH	00FB4011H
	00H	
	00H	
	00H	
u	31H	00FB4015H
	32H	
	33H	
	34H	
	35H	
p	00H	00FB401AH
	40H	
	0fbH	
	00H	
	03H	00FB401EH
	40H	
	0fbH	
	00H	

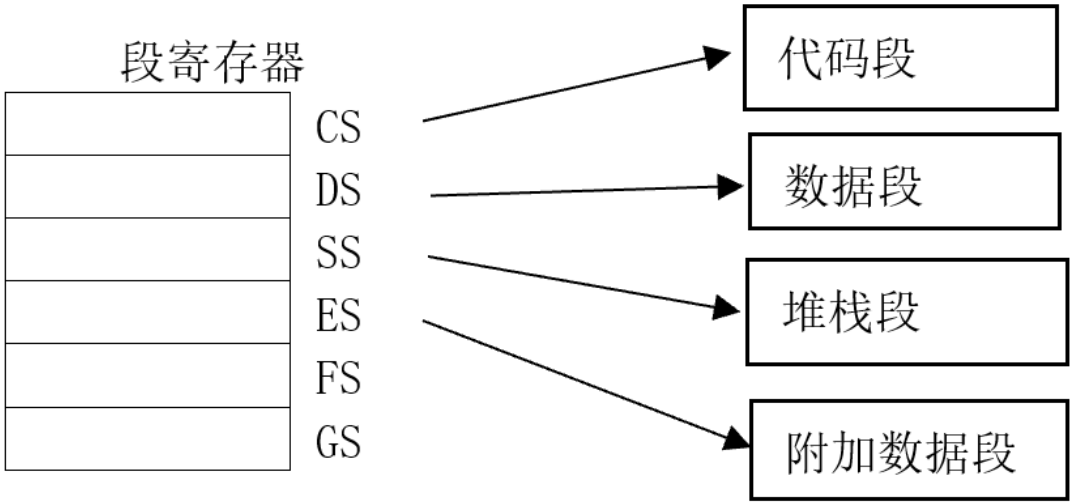
3.5 主存储器分段管理

- 内存管理有两种模型：扁平内存模型和分段内存模型。
- 在扁平内存模型中，代码、数据、堆栈等全部放在同一个4GB的空间中。
- 虽然代码、数据、堆栈放在同一个段中，但是在段的不同位置，仍然是分离的。



3.5 主存储器分段管理

- 内存管理有两种模型：扁平内存模型和分段内存模型。
- 在分段内存模型中，每个分段通常加载不同的分段选择器，以便每个段寄存器指向线性地址空间内的不同段。



3.6 主存储器物理地址的形成

3.6.1 8086和x86-32实方式下物理地址的形成

3.6.2 保护方式下物理地址的形成

3.6.1 8086和x86-32实方式下物理地址的形成

内存1M, 20位物理地址，CPU中是16位的寄存器。

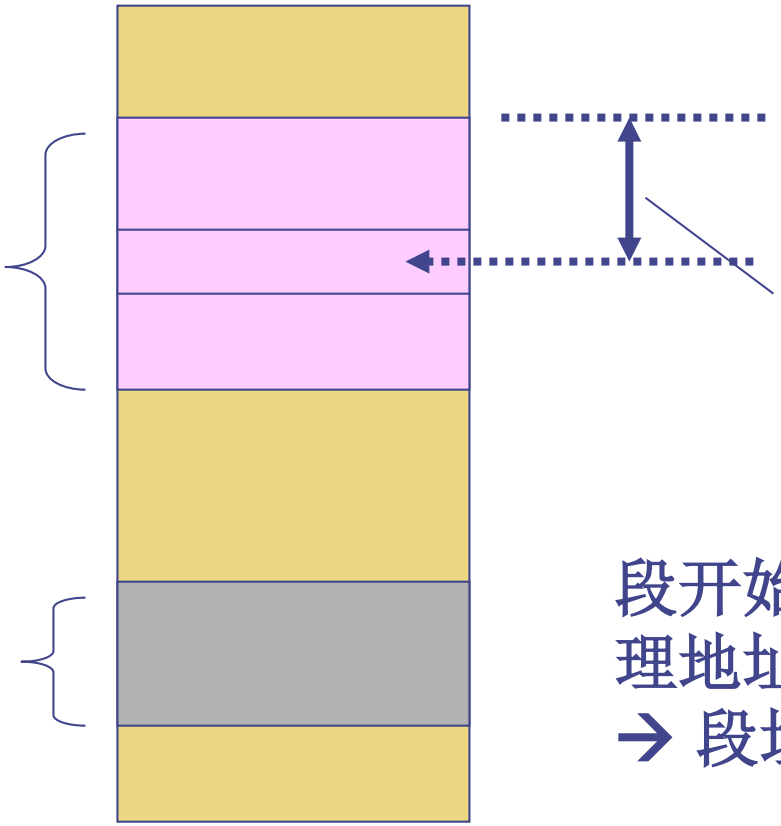
16位的寄存器如何与20位的物理地址建立对应关系？

分段

偏移地址

3.6.1 8086和x86-32实方式下物理地址的形成

分段和段内偏移



段的开始地址要能被16整除。

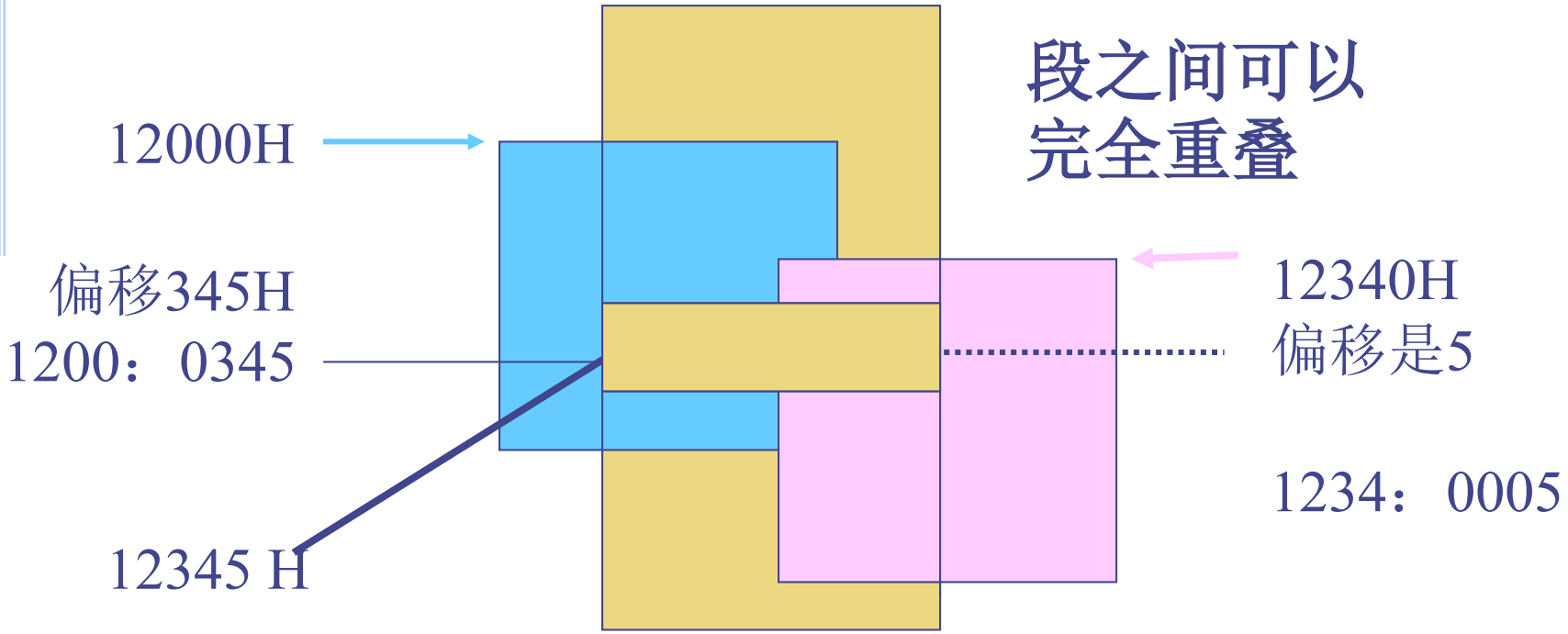
16 = 10H
= 10000B

段址 * 16 + 偏移地址 = 物理地址

3.6.1 8086和x86-32实方式下物理地址的形成

关于二维逻辑地址　　段寄存器：偏移地址

段址 * 16 + 偏移地址 = 物理地址

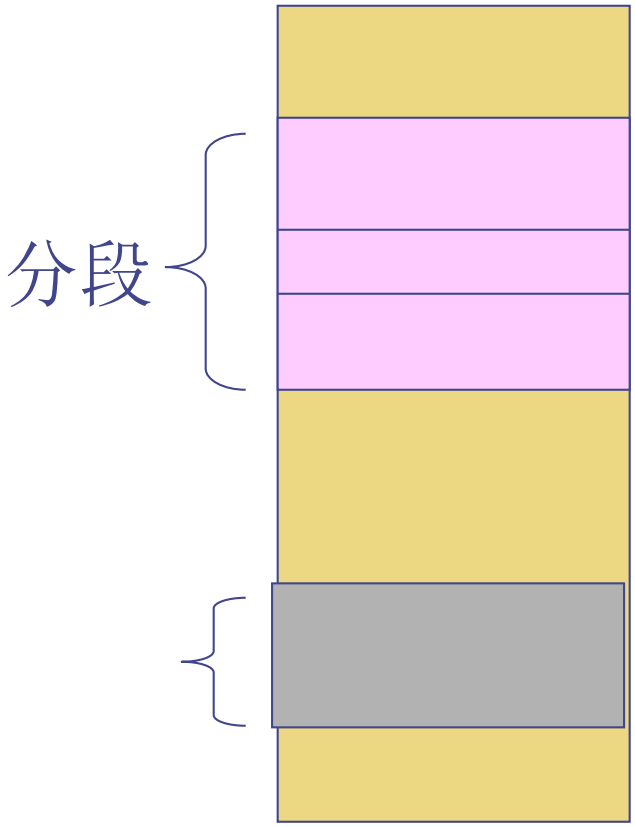


段中某一存储单元的地址是用两部分来表示的，“段首地址：偏移地址”，称它为二维的逻辑地址。

3.6.1 8086和x86-32实方式下物理地址的形成

关于分段

一个段最大为64KB. (2^{16})



1M内存最少有16个段

同时访问4个段, 段寄存器CS,DS,ES,SS

3.6.1 8086和x86-32实方式下物理地址的形成

总结：8086中，只有4个段寄存器CS,DS, ES,SS

在代码段中取指令时：

指令物理地址PA=（CS）左移四位+（IP）

注意，使用的是IP, 而不是EIP

在数据段中读/写数据时：

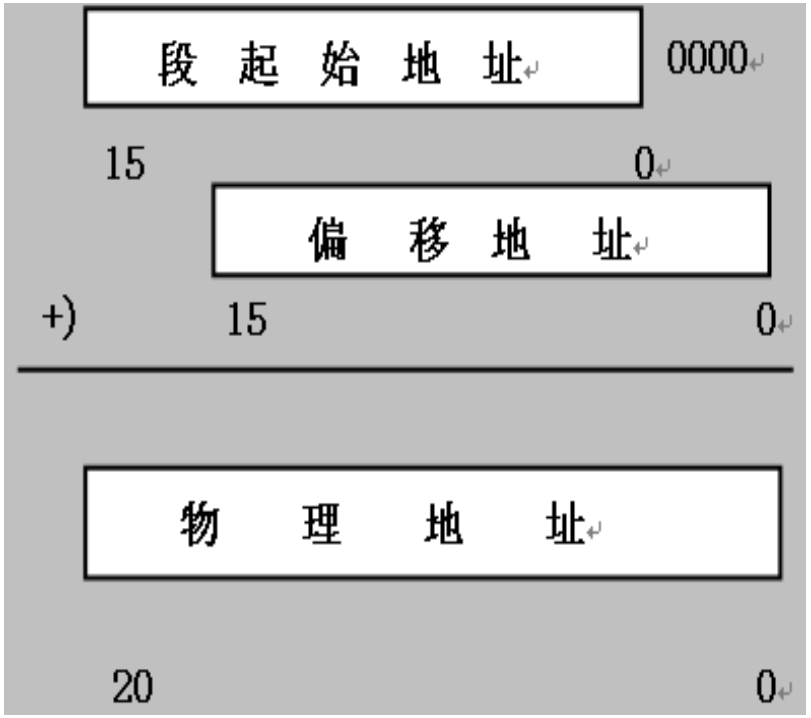
数据的物理地址PA=（DS或ES）左移四位 +16位偏移地址 （偏移地址由寻址方式确定）

在堆栈操作时：

栈顶的物理地址PA=（SS）左移四位+(SP)

1. 实方式物理地址的形成

- ✓32位CPU与8086一样
- ✓只能寻址1M物理存储空间
- ✓可以访问6个段
- ✓ CS, DS, SS, ES, FS, GS
- ✓ 每个段至多64K



物理地址 ＝ （段寄存器）左移4位 ＋偏移地址

3.6.1 8086和x86-32实方式下物理地址的形成

Q: 程序中能够直接使用物理地址吗？
有必要使用物理地址吗？

程序中单元（如变量等）的相对位置，逻辑地址

3.6.1 8086和x86-32实方式下物理地址的形成

- C语言程序中，变量的定义和指令写在一起
- C语言程序中无分段的概念
- 机器语言层次上，是要分段的
- 在C程序编译时，将变量的空间分配和指令分开，分别放在不同段中。

思考题：

为什么机器指令和数据存放要分开呢？

例：
```
MOV   EAX, 0
DB    'GOOD'
MOV   EBX,  10
```

3.6.2 保护方式下物理地址的形成

80386中寄存器32位，地址线32根。

- 在多任务环境下，系统中有多个程序在运行。
- 程序之间要隔离！

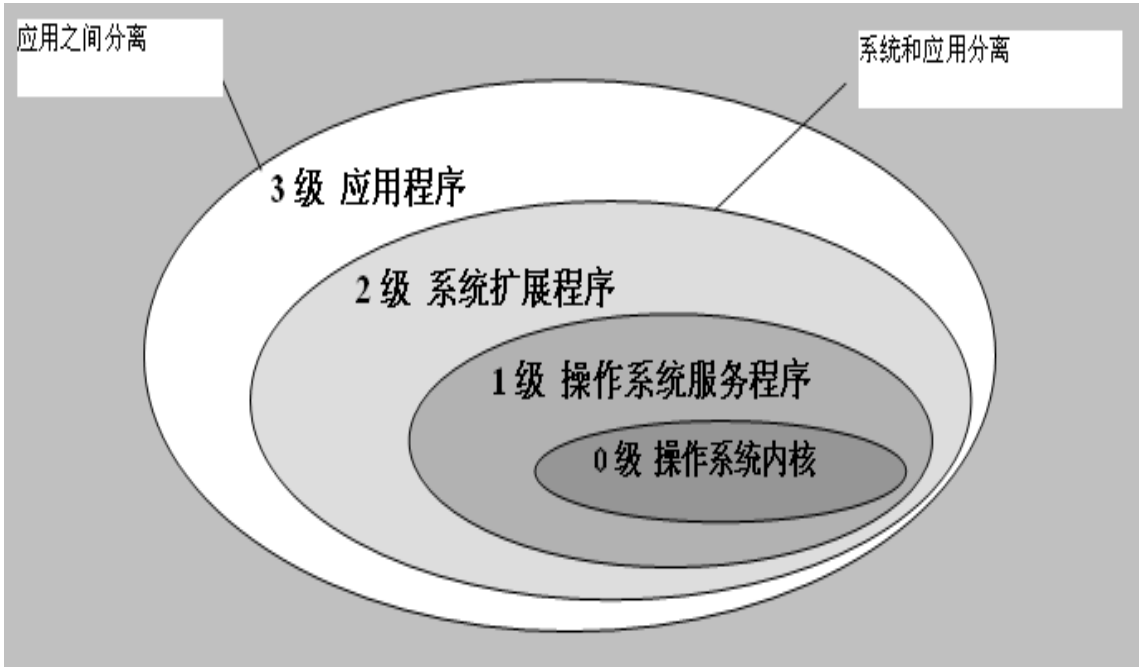
Q:如何隔离？

- 分段是存储管理的一种方式，为保护提供基础；
- 不同程序在不同段中；
- 一个程序可以包含多个段；
- 段用于封闭具有共同属性的存储区域；
- 二维逻辑地址的概念——段寄存器：偏移地址

3.6.2 保护方式下物理地址的形成

(1) 特权级

CPU 总是在一个特权级上运行，称为当前特权级。



被访问段的特权级应等于或低于当前特权级

3.6.2 保护方式下物理地址的形成

Q：要保护一个段，应该提供哪些信息？
这些信息又存放在何处？

(2) 描述符 (description)

段的起始位置（段基地址）

段的大小（段界限）

段的特权级DPL

段的属性TYPE：是代码段，数据段，还是堆栈段？）
（数据段是否可写？代码段是否可读出？）

段的位置P（在内存还是在磁盘？）

段的类型S（在系统段还是用户段？）

段的使用A（段被访问过，还是没有？）

3.6.2 保护方式下物理地址的形成

(2) 描述符

段的有关信息的描述

段基地址：B31 ~ B0, 共32位；

段界限：L19 ~ L0, 共20位

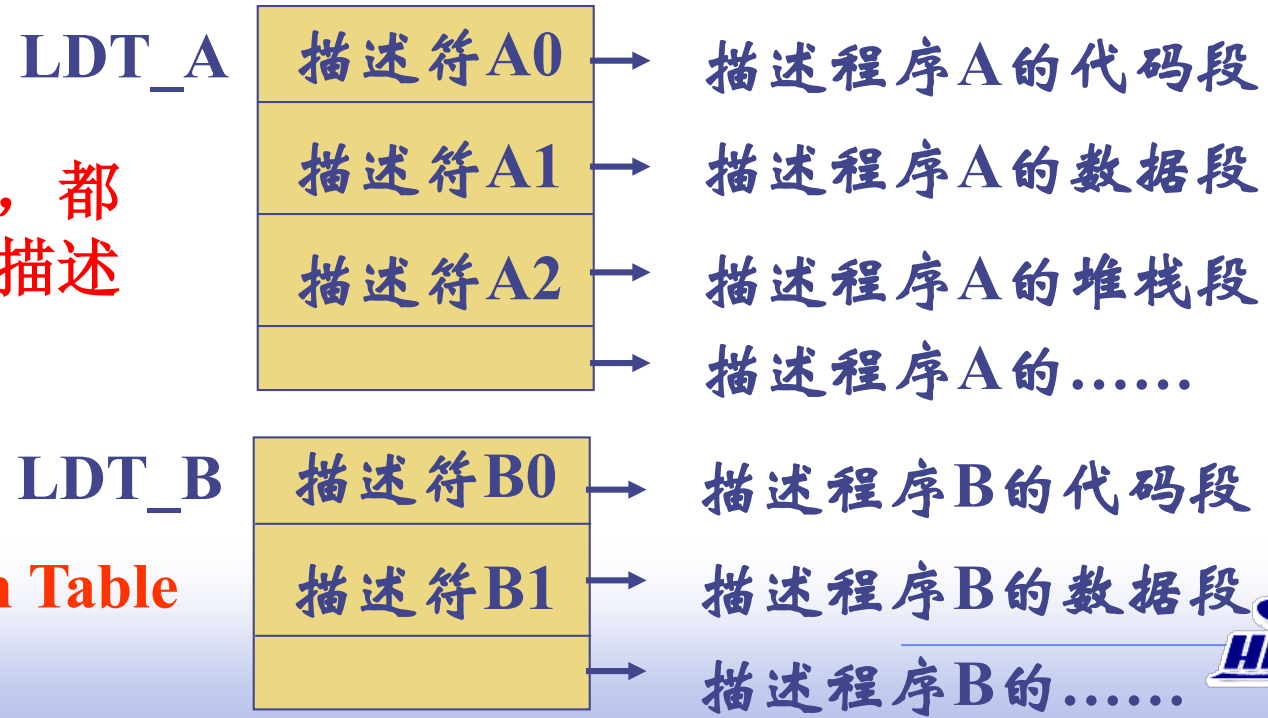
3.6.2 保护方式下物理地址的形成

(3) 描述符表　　描述符的集合 －－ 描述符表

局部描述符表：

一个LDT，是一个系统段，最大可为64KB,最多可存放8192个描述符。（64KByte / 8Byte per Descriptor)

对每一个程序，都建立一个局部描述符表（LDT)。



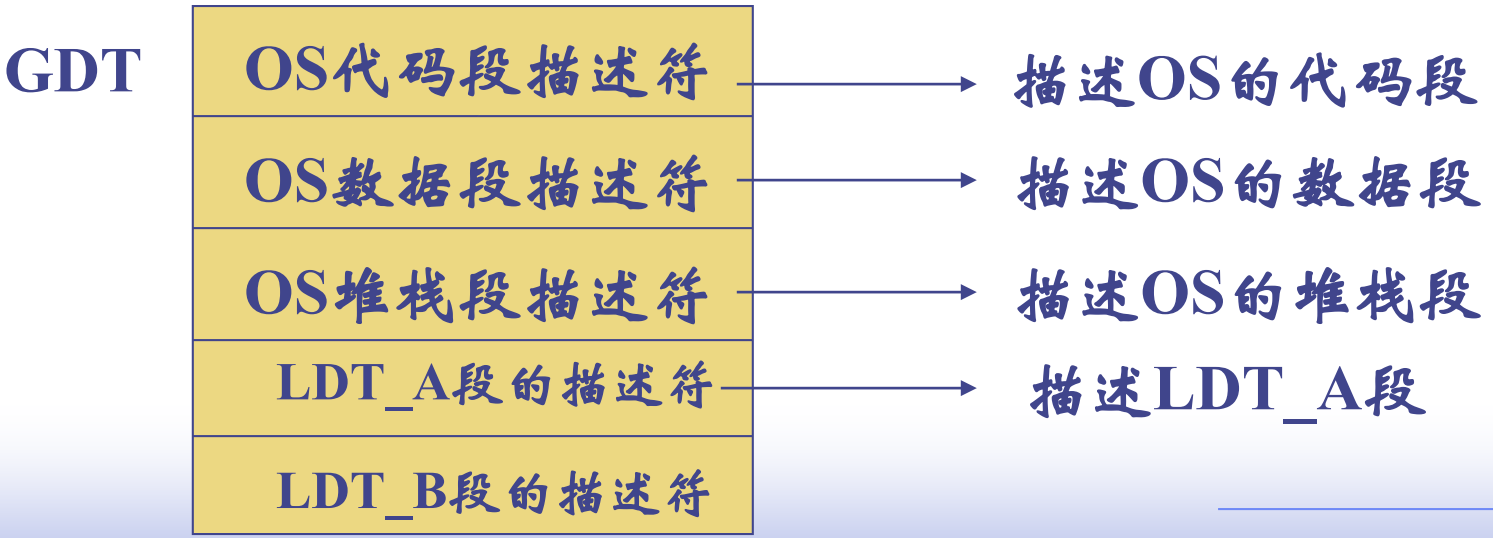
Local Description Table

3.6.2 保护方式下物理地址的形成

(3) 描述符表

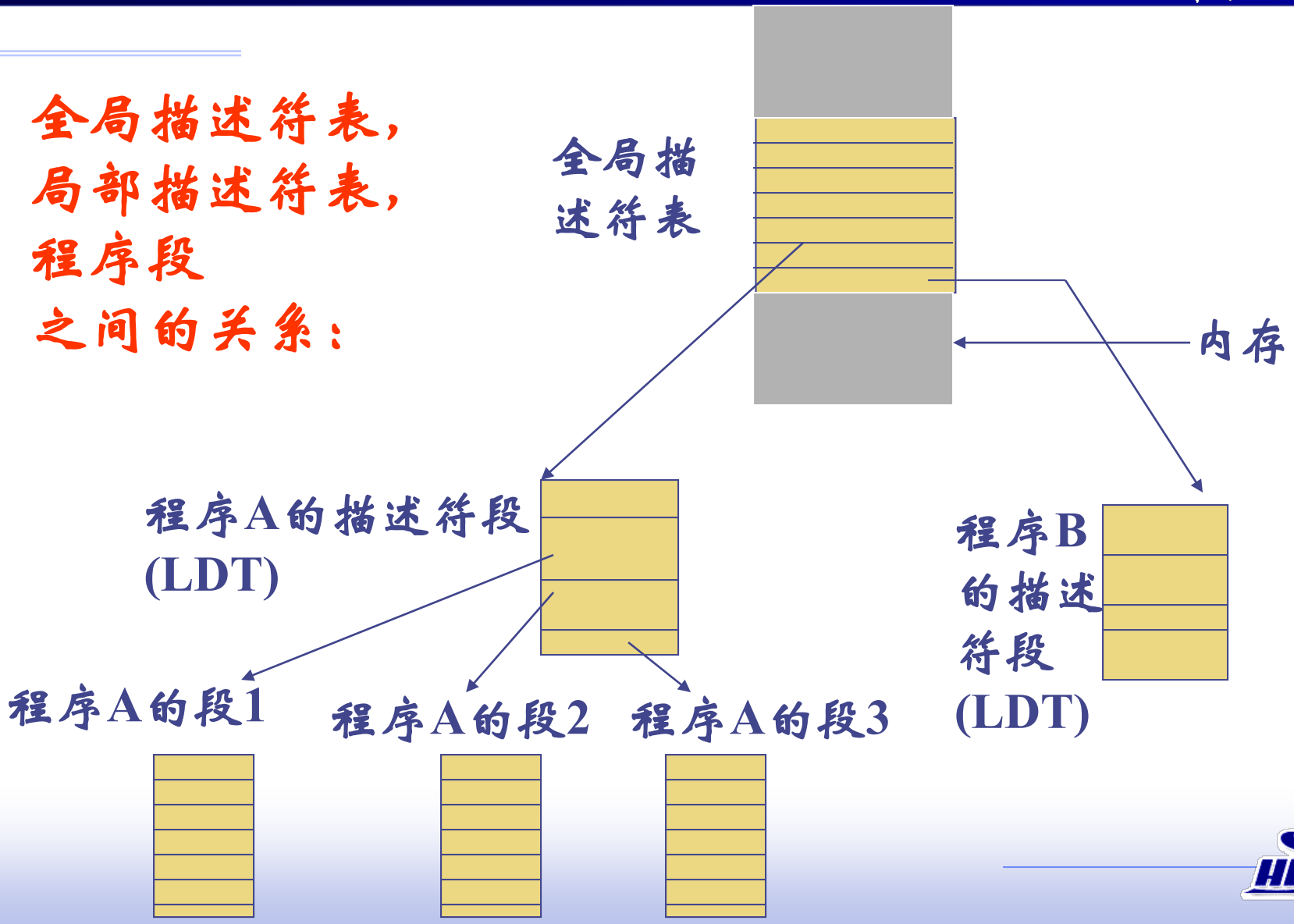
全局描述符表：只有一个。GDT最大可为64KB，存放8192个描述符。包括：

- 操作系统所使用的段的描述符；
- 各个LDT段的描述符



3.6.2 保护方式下物理地址的形成

全局描述符表，
局部描述符表，
程序段
之间的关系：



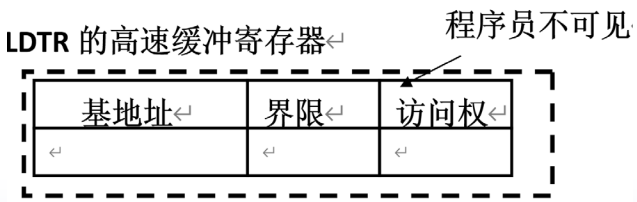
3.6.2 保护方式下物理地址的形成

LDTR

15 3 2 0

索引值n

13位的索引值，可表示8192个索引项



内存示意图

GDTR

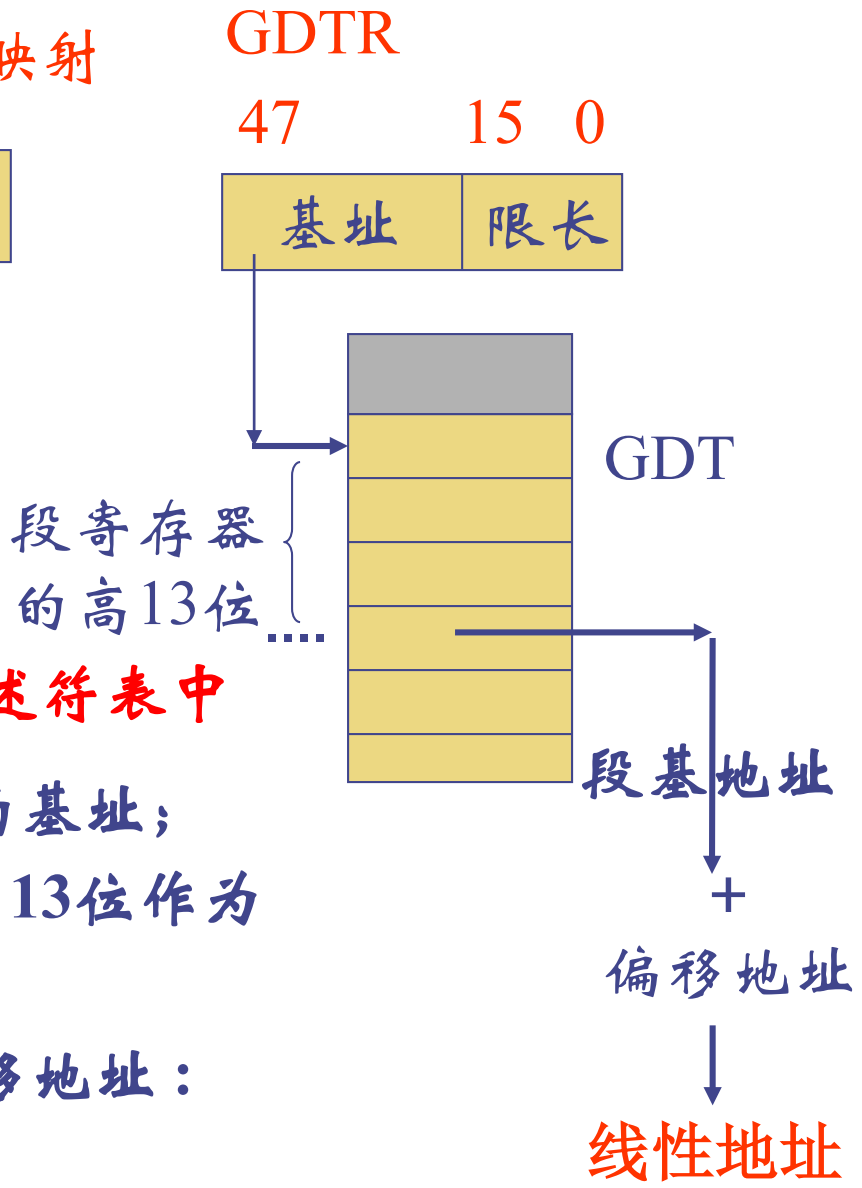
47 15 0

基址 限长

32位的基址，使得GDT可以定位在线性地址空间的任何位置

段寄存器：偏移地址到线性地址的映射

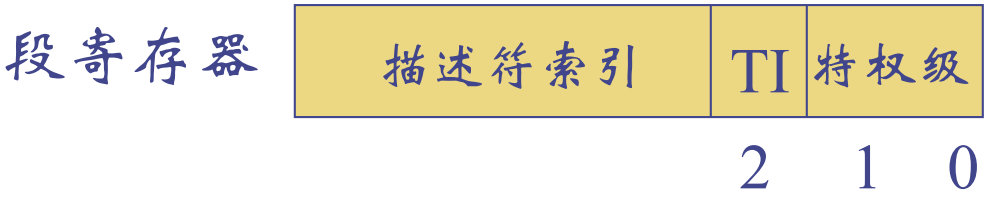
段寄存器的内容不是段的开始地址，而是指出找相应段描述符的方式。称为**段选择符**。



(1) TI 位为 0----描述符在全局描述符表中

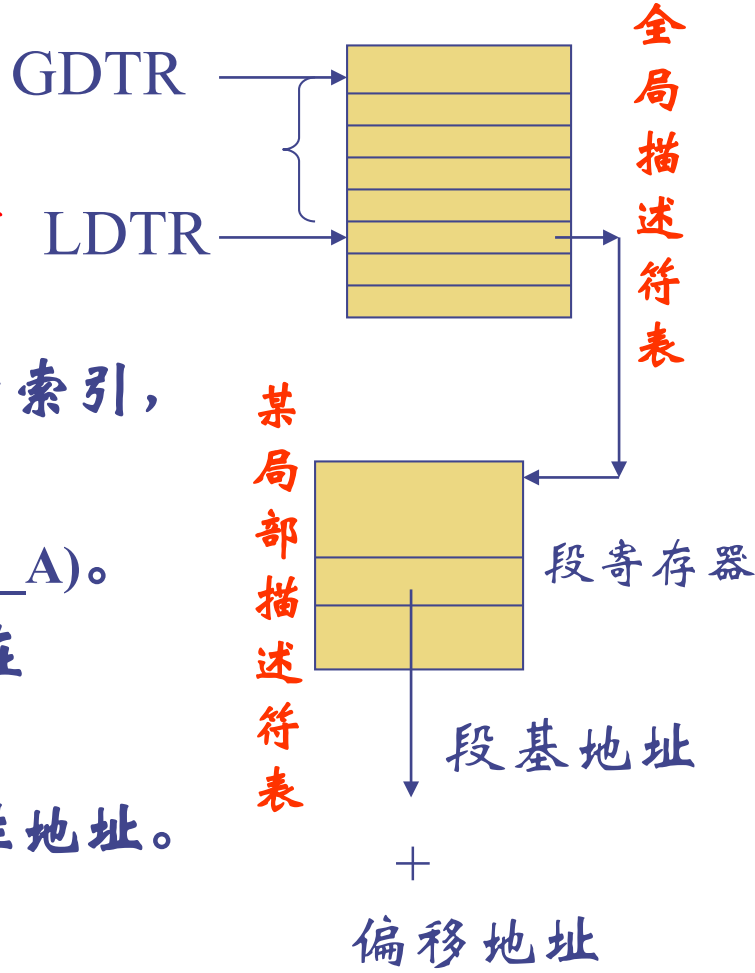
- 从GDTR寄存器中获取GDT的基址；
- 在GDT表中，以段寄存器的高13位作为索引，取出一个描述符A；
- 描述符A中的段基地址 + 偏移地址：为要访问单元的线性地址。

段寄存器：偏移地址到线性地址的映射

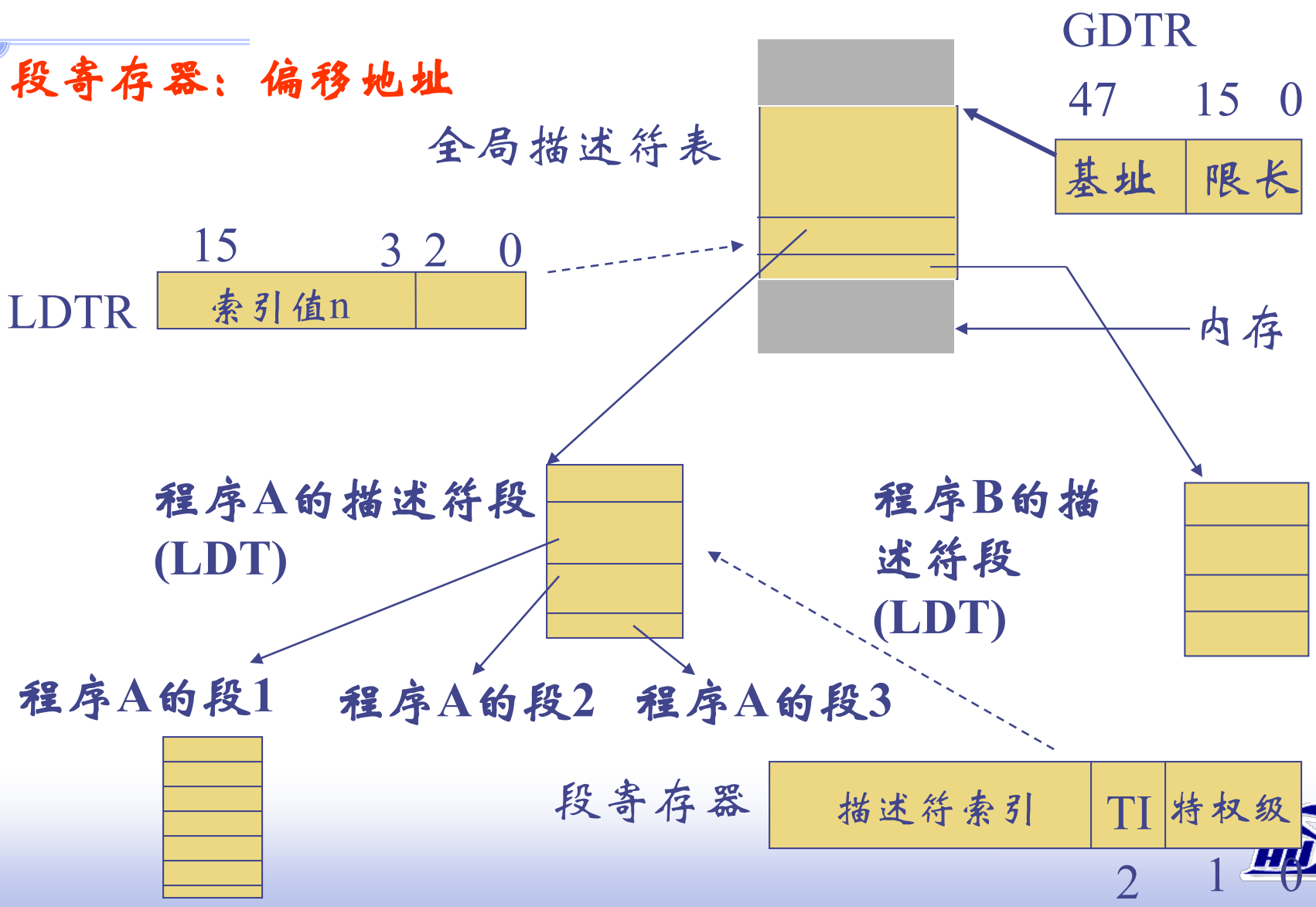


(2) 若TI 位为 1----描述符在局部描述符表

- 从GDTR寄存器中获取GDT的基址；
- 在GDT表中，以LDTR的高13位作为索引，取出一个描述符A；
- 描述符A描述的段为一个LDT段(LDT_A)。
- 用 段寄存器的高13位，作为索引，在LDT_A段中找到描述符P_A。
- P_A描述段的基址＋偏移地址 为线性地址。



3.6.2 保护方式下物理地址的形成



3.6.2 保护方式下物理地址的形成

- 总结:

保护方式下，用户既不用处理多任务，也不用关心程序映射到哪一片物理存储区，物理地址的计算是由CPU自动完成的。实方式和保护方式的逻辑地址表达是类似的。

段寄存器：偏移地址

作业

P51 3.13、3.14、3.15
3.16、3.17

实验准备：

汇编答疑群下载：MASM60以及虚拟机安装环境DOSBOX，并安装好。MASM以及TD环境使用可以阅读老书第七章（可在群里下载）