# 第5章 8086微机系统原理和结构

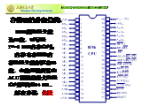
# 5.1 8086 CPU结构与功能

## 8086 CPU概述

1977年，Intel率先推出了16位微处理器8086，能并行处理16位数据，它需要16位的存储器， 16位DB，16位外设。1979年Intel研制了8088，称为准16位机。

- 引脚功能复用
- 单总线、累加器结构
- 可控三态电路
- 总线分时复用

# 一、 结构特点

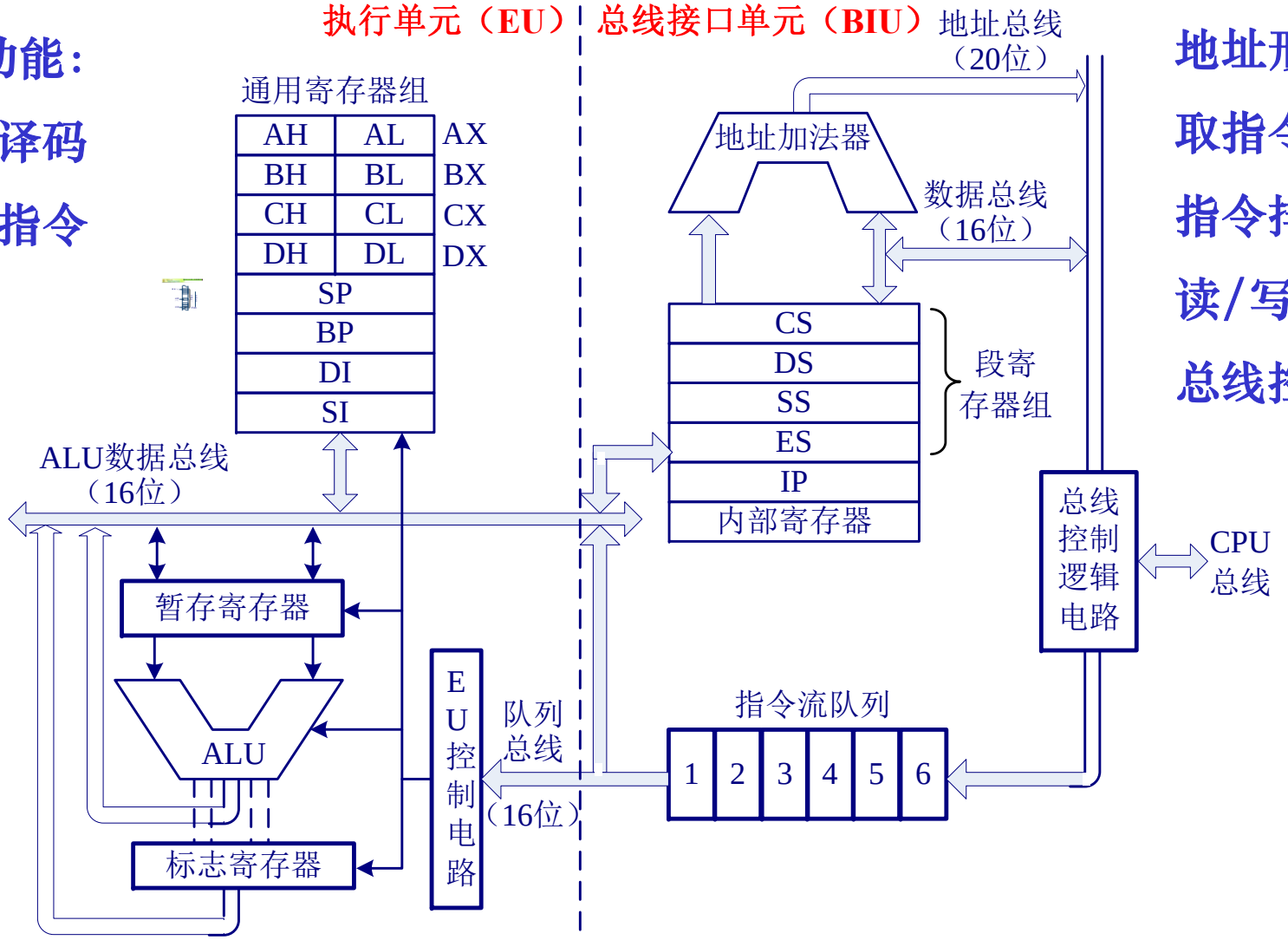
- 指令流水线 
- 存储器的分段结构 
- 支持用于浮点运算的协处理器及多微处理器系统
- 指令方面和结构设计支持使用该微处理器构成一个共享总线的多微处理器系统

# 二、 编程结构（功能结构）

**EU功能：**

**指令译码**

**执行指令**




**BIU功能：**

**地址形成**

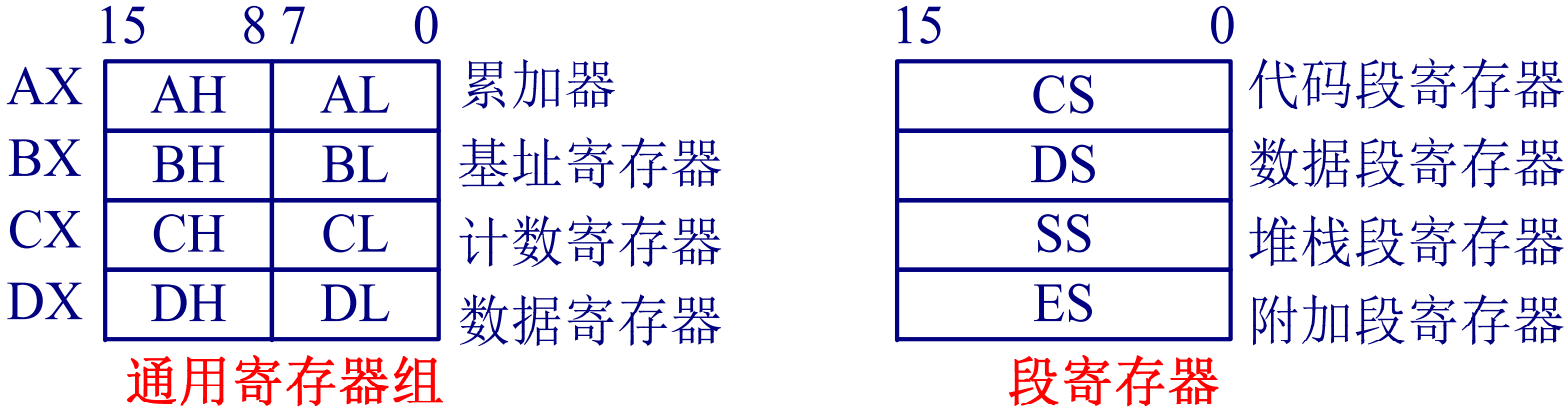
**取指令**

**指令排队**

**读/写操作数**

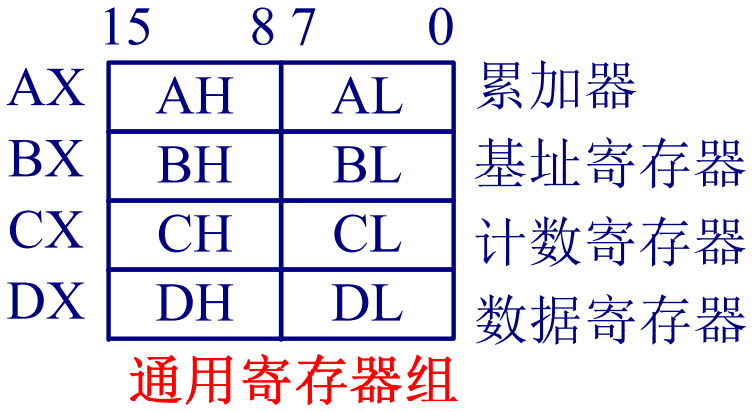
**总线控制**

# 三、 寄存器结构


15 8 7 0
AX AH AL 累加器
BX BH BL 基址寄存器
CX CH CL 计数寄存器
DX DH DL 数据寄存器
通用寄存器组
15 0
CS 代码段寄存器
DS 数据段寄存器
SS 堆栈段寄存器
ES 附加段寄存器
段寄存器
15 0
SP 堆栈指针寄存器
BP 基址指针寄存器
SI 源指针寄存器
DI 目的指针寄存器
指针和变址寄存器
15 0
IP 指令指针寄存器
PSW 标志寄存器
指令指针和
标志寄存器

## 通用寄存器：

每一个数据寄存器都是16位寄存器，但又可将高8位和低8位分别作为两个独立的8位寄存器使用。




① AX：常用于存放算术逻辑运算中的操作数。所有的I/O指令都使用累加器与外设接口传送信息。

② BX：常用来存放访问内存时的基地址。

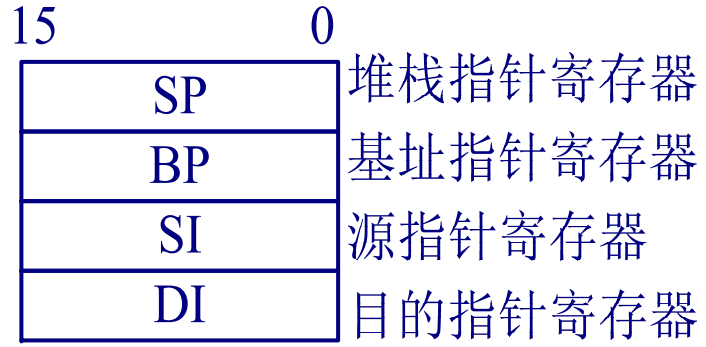
③ CX：在循环和串操作指令中用做计数器。

④ DX：在寄存器间接寻址的I/O指令中存放I/O端口的地址。

⑤ 在做双字长乘、除法运算时，DX与AX合起来存放一个双字长数(32位)，其中DX存放高16位，AX存放低16位。

**指针和变址寄存器:**

**SP：在堆栈操作中用来存放栈顶的偏移地址，永远指向堆栈的栈顶。**




指针和变址寄存器

**BP：基地址指针寄存器。一般也常用来存放访问内存时的基地址。但它通常是与SS寄存器配对使用(BX通常是与DS寄存器配对使用)。**

**SI、DI：它们常常在变址寻址方式中作为索引指针。在字符串操作指令中，要求用SI作为源变址寄存器，存放源操作数的偏移地址；DI作为目标变址寄存器，存放目标操作数的偏移地址。**

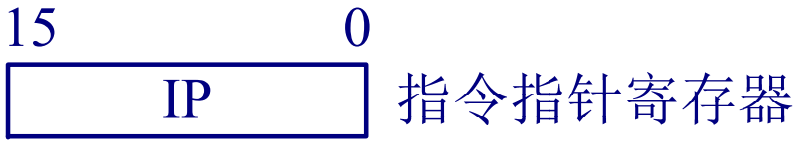
段寄存器：

CS：代码段存放的是当前执行程序的指令代码。CS的内容是代码段的段基地址，它和指令指针IP一起决定下一条所要执行指令的物理存储地址。


DS：数据段通常用来存放数据和字符。DS存放当前数据段的段基地址。

ES：附加段是一个附加数据段，主要用在字符串操作时作为目标地址使用。ES的内容就是加段的段基地址。

SS：堆栈是在存储器中开辟的一个特殊存储区，用于存放当前暂时不用但又需要保存的数据和地址。如在子程序调用或响应中断时需要保存返回主程序的地址和进入子程序后将要改变其值的寄存器的内容。




指令指针寄存器

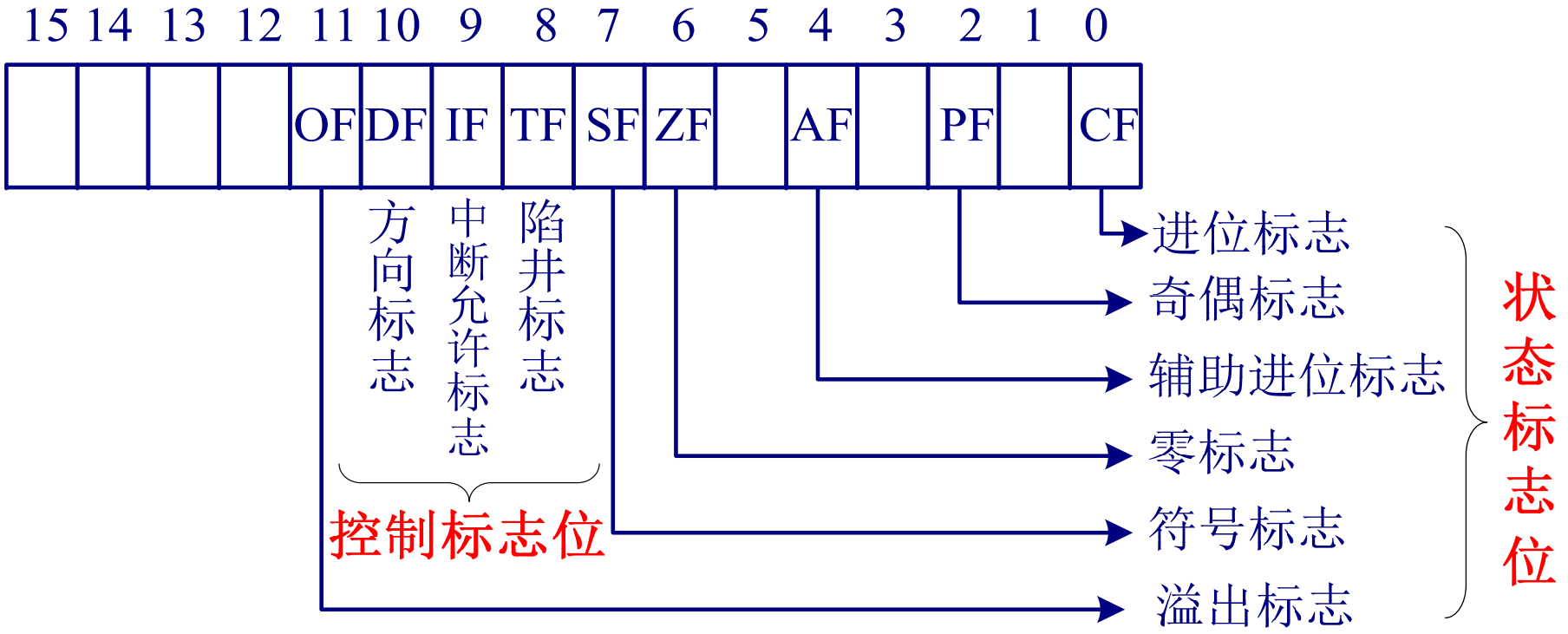
**指令指针寄存器:**

**IP用来存放下一条要执行指令的偏移地址。CPU取指令时总是以CS的内容为段基地址，以IP为段内偏移地址。当CPU从CS段偏移地址为(IP)的内存单元中取出指令代码的一个字节后，IP自动加1，指向指令代码的下一个字节。**

**遇到过程调用、转移及返回等指令时，系统将根据程序确定新的IP的内容，使其不再加1。**

**用户程序不能直接访问IP(指令的操作数不能是IP)。**

**标志寄存器：也称程序状态字(PSW)，是一个16位寄存器，但只使用了其中的9位，包括6个状态标志位和3个控制标志位。**



| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | OF | DF | IF | TF | SF | ZF | | AF | | PF | | CF |

## (1) 条件标志(6个)：反映指令执行后运算结果特征.

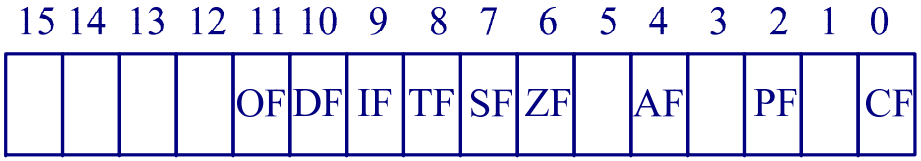
- CF(进位标志): CF=$D_{7CY}$或$D_{15CY}$

  执行算术运算指令后，结果的最高位(字节时为$D_{7CY}$或字为$D_{15CY}$)向更高位产生进位/借位，则CF=1，否则CF=0。该标志主要用于多字节加、减运算。

- 例:

```
    3FH+0B4H                    0BFH+0B4H
    0011 1111                   1011 1111
+   1011 0100               +   1011 0100
-------------               -------------
    1111 0011;CF=0              1 0111 0011;CF=1
```

注:对CF操作有三条专用指令: STC→CF=1;　　CLC→CF=0;　　CMC→CF=~CF




- PF(奇偶校验标志): $PF=\overline{D_7 \oplus \cdots \oplus D_0}$

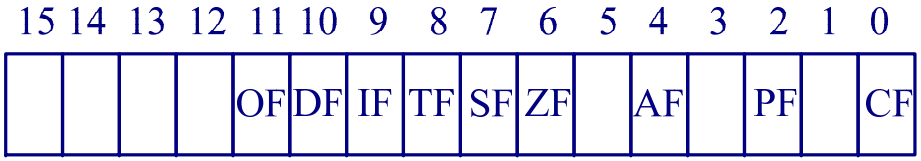
运算结果的低8位中“1”的个数为偶数，则PF=1，否则PF=0。该标志主要用于检测数据通信中是否发生错误。

- AF(辅助进位标志): $AF=D_{3CY}$

字节运算中，低4位向高4位有进位或借位时,则AF=1,否则AF=0。该标志主要用于BCD码运算的调整指令中。

例: 38H+49H

```
  0011 1000
+ 0100 1001
-----------
  1000 0001 ;AF=1;若视为BCD运算，则应调整。
```




- ZF(零标志): ZF=$D_7$+…+$D_0$或$D_{15}$+…+$D_0$

  运算结果为0，则ZF=1，否则ZF=0。

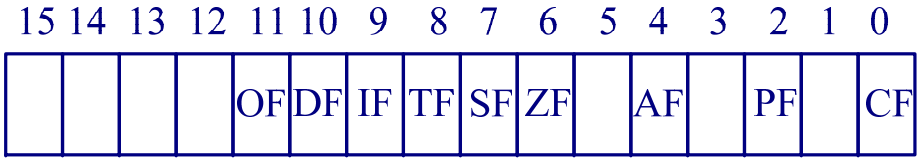
  结果非0，则ZF=0。

- SF(符号标志) SF=$D_7$或$D_{15}$

  运算结果为正数，则SF=0；为负数,则SF=1。

  如:3FH+0B4H=0F3H的SF=1

  而:0BFH+B4H=173H的SF=0




- OF(溢出标志): $OF=D_{7CY}\oplus D_{6CY}$或$D_{15CY}\oplus D_{14CY}$

当有符号数运算结果超出了机器所能表示的范围时，则OF=1，否则OF=0。

如:3FH+0B4H=0F3H中OF=0

而:0BFH+0B4H=173H中OF=1

- 注意：实际上机器把所有数都当无符号数运算，把结果都当有符号数来设置标志。
- 以上6个标志为指令执行后的结果标志，可作为控制转移的条件。

例： 假设执行一条加法指令，计算5439H+476AH后各状态标志位的状态为何?

解：

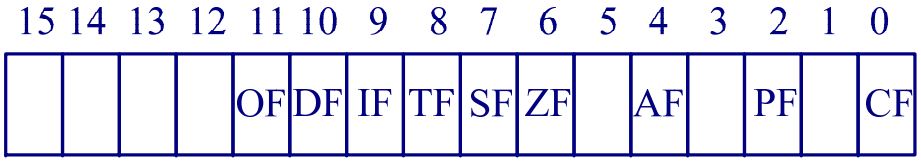
```
  0 1 0 1  0 1 0 0  0 0 1 1  1 0 0 1
+ 0 1 0 0  0 1 1 1  0 1 1 0  1 0 1 0
-------------------------------------
  1 0 0 1  1 0 1 1  1 0 1 0  0 0 1 1
```

则执行这条加法指令后标志寄存器的状态为：CF=0，PF=1，AF=1，ZF=0，SF=1，OF=1。

## CF和OF的进一步讨论：

- CF表示无符号数运算结果是否超出范围；OF表示有符号数运算结果是否超出范围
- 处理器对两个操作数进行运算时，按照无符号数求得结果，并相应设置进位标志CF；同时，根据是否超出有符号数的范围设置溢出标志OF
- 程序员决定标志位的使用。如果将参加运算的操作数认为是无符号数：CF；认为是有符号数：OF
- 判断运算结果是否溢出：当两个相同符号数相加，而运算结果的符号与原数据符号相反时，产生溢出。其他情况下，则不会产生溢出。

## (2) 控制标志----控制CPU的状态。




- DF(方向标志): 控制字符串操作中地址的步进方向。
  DF=0，地址增址；DF=1，地址减址。
  专门用于DF的指令: CLD→DF=0； STD→DF=1

- IF(中断允许标志): 控制CPU是否开中断。IF=1，允许CPU响应外部可屏蔽中断。IF=0，禁止CPU响应外部可屏蔽中断。
  两条关于IF的专用指令:CLI→IF=0； STI→IF=1

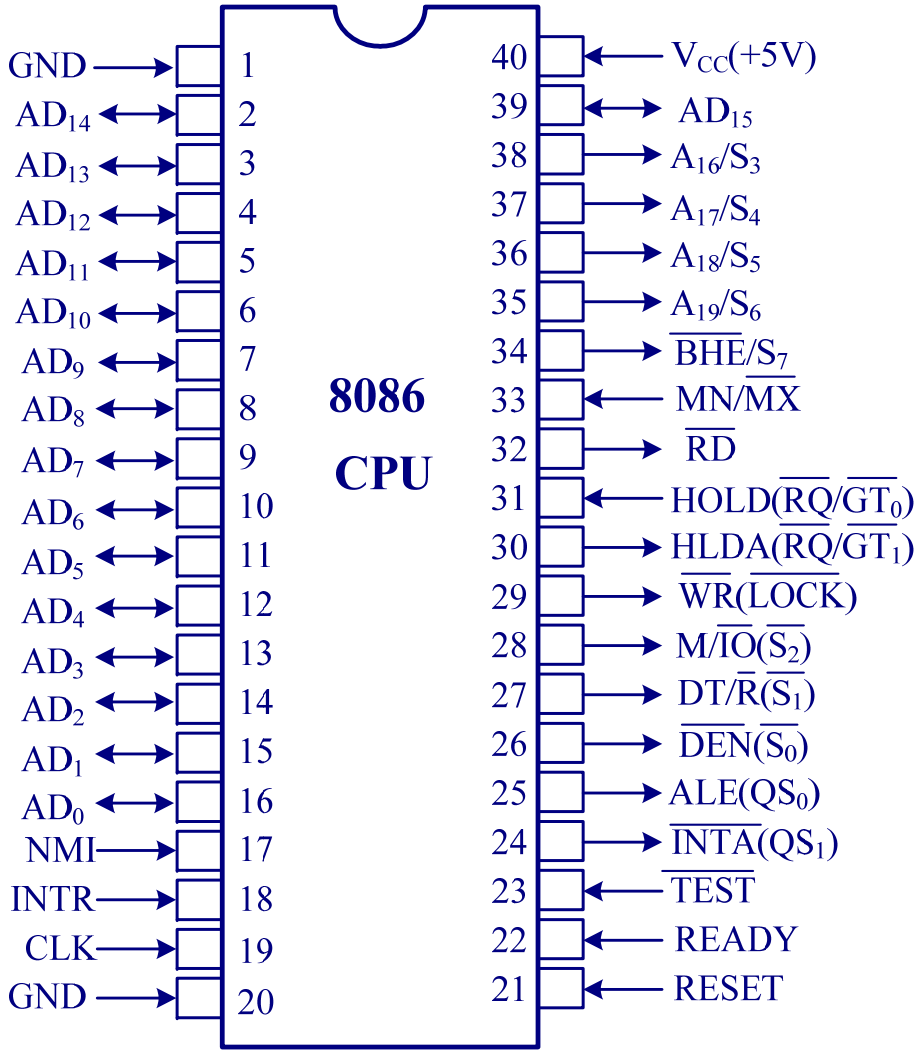
- TF(跟踪标志): TF=1，CPU处于单步工作方式，即CPU每执行一条指令就自动地发生一个内部中断，CPU转去执行一个中断程序，常用于程序调试，又称为陷井标志。TF=0，CPU正常执行程序。

# 5.2 8086 CPU的引脚及其总线结构

## 一、 8086 CPU的引脚

CPU的引脚：

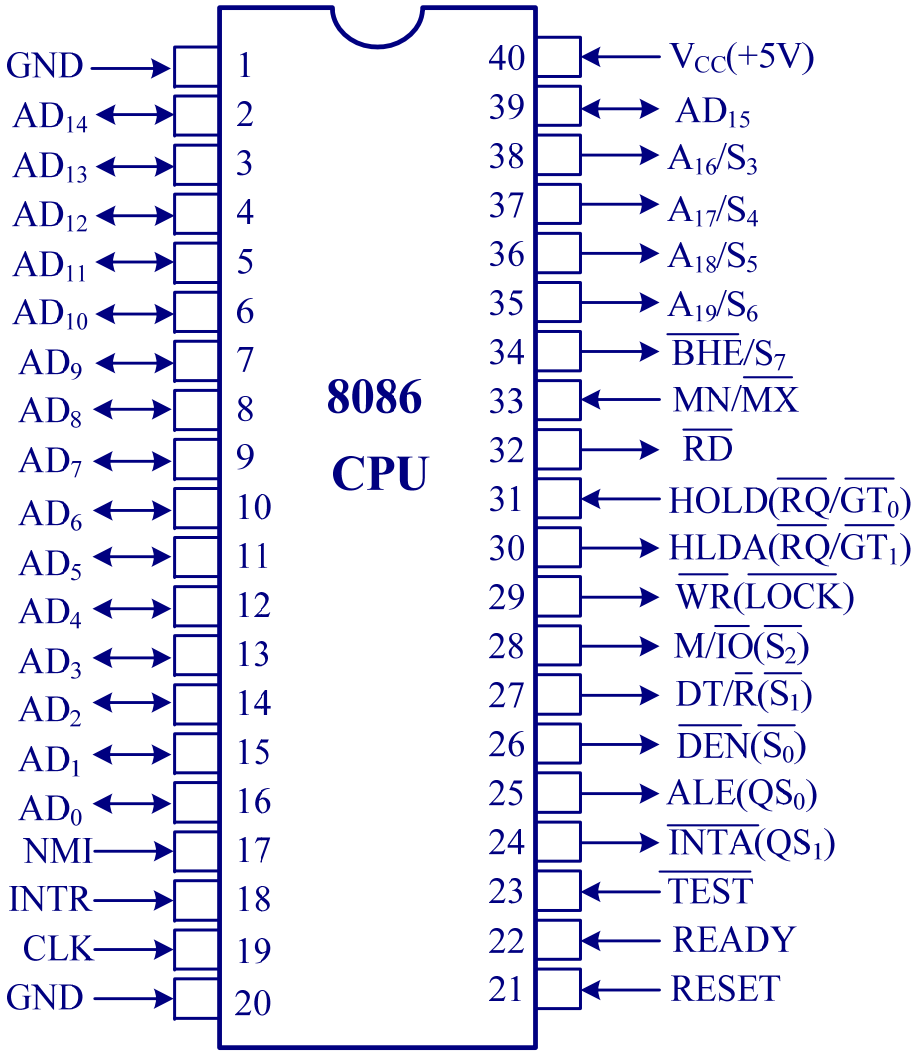
地址线与数据线，
控制与状态线，
电源与定时线。



# 8086的两种工作模式

- 最小模式：单微处理器模式，CPU仅支持由少量设备组成的单微处理器系统，小系统所需要的全部控制信号都由CPU直接提供。
- 最大模式：多微处理机模式，系统中除了有8086 CPU之外，还可以接另外的处理器构成多微处理器系统。

当MN/~MX =1时，8086工作在最小模式；当MN/~MX =0时，8086工作在最大模式。两种工作模式下的部分引脚具有不同的功能。



# 8086在两种工作模式下的公用引脚：

1)地址／数据总线：8086有20位地址线，16位数据线，采用分时复用方式，共同占用20根引脚。

$AD_0$~$AD_{15}$ 地址、数据分时复用双向信号线，三态。当ALE=1时，这些引脚上传输的是地址信号；~DEN=0时，这些引脚上传输的数据信号。

$A_{16}$~$A_{19}$/$S_3$~$S_6$ 分时复用的地址状态信号线，三态输出。在8086访问存储器时，读/写总线周期的第一个机器周期$T_1$，从这4个引脚上送出最高4位地址$A_{16}$~$A_{19}$。而在总线周期的其他机器周期，这4个引脚送出状态信号$S_3$~$S_6$。

| $S_4$ | $S_3$ | 当前正在使用的段寄存器 |
|---|---|---|
| 0 | 0 | ES |
| 0 | 1 | SS |
| 1 | 0 | CS或未使用任何段寄存器 |
| 1 | 1 | DS |

S5：取中断允许标志的状态
S6：0，表明8086在总线上；否则为1

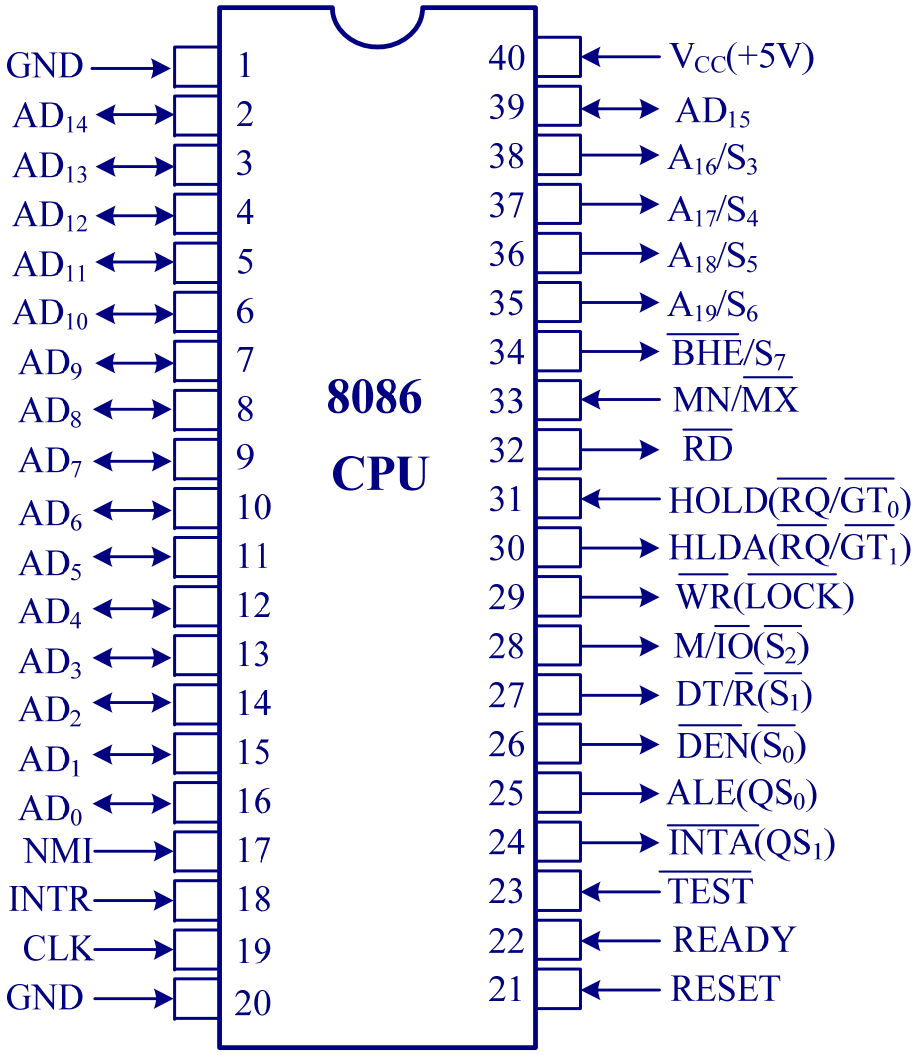
2) 控制总线：共有16根，其中两种工作模式共用的有8根引脚。

MN/~MX：工作方式控制输入。为高电平时，CPU工作在最小模式；为低电平时，CPU处于最大模式。

~RD：读选通信号，三态，低电平有效。有效时，表示CPU正在对存储器或I/O接口进行读操作。

READY：“准备好”信号输入引脚，高电平有效。有效时，表示存储器或I/O设备已准备好，CPU可以进行数据传送。

INTR：可屏蔽中断请求输入信号，高电平有效。CPU在每条指令的最后一个周期采样该信号，以决定是否进入中断响应周期。可用软件屏蔽。

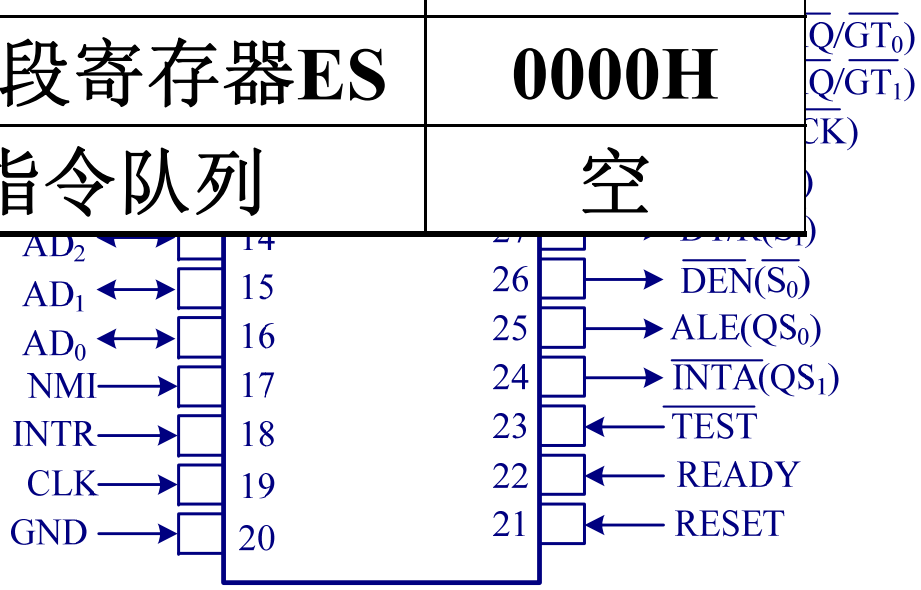


~TEST：测试信号输入引脚，
平有效。当CPU执行WAI
时，每隔5个时钟周期对此
进行一次测试。若为高电
CPU则继续处于空转状态进
待，直到引脚变为低电平
CPU才结束等待状态，继续
下一条指令。

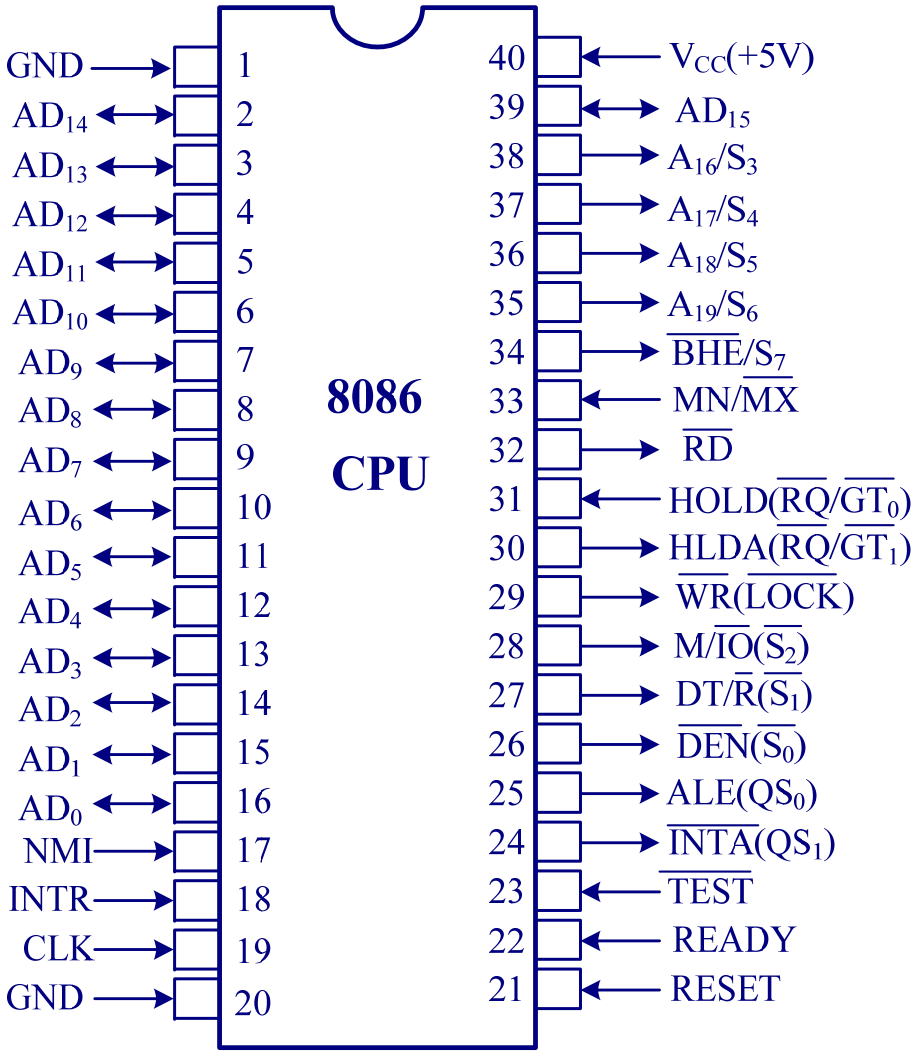
NMI：非屏蔽中断请求输入信
上升沿触发。这个引脚上的
请求信号不能用软件屏蔽，
在当前指令执行结束后就进
断过程。

RESET：系统复位输入信号，高电平有效。为使CPU完成内部复位过程，该信号至少要在4个时钟周期内保持有效。当RESET返回低电平时，CPU将重新启动。

| | |
|---|---|
| 标志寄存器FR | 0000H |
| 指令寄存器IP | 0000H |
| 代码段寄存器CS | FFFFH |
| 数据段寄存器DS | 0000H |
| 堆栈段寄存器SS | 0000H |
| 附加段寄存器ES | 0000H |
| 指令队列 | 空 |



~BHE/$S_7$：分时复用的控制／状态信号线，三态输出。在总线周期的第一个时钟周期输出信号，其他时钟周期输出状态信号$S_7$($S_7$的意义目前没有定义)。信号~BHE的意义是：当~BHE=0时，表示可使用高8位数据线$AD_8$~$AD_{15}$；否则只使用低8位数据线$AD_0$~$AD_7$。



最小模式下引脚24~31的功能定义：

~INTA：中断响应输出端。

ALE：地址锁存允许信号，三态输出，高电平有效。

~DEN：数据允许信号，三态，低电平有效。

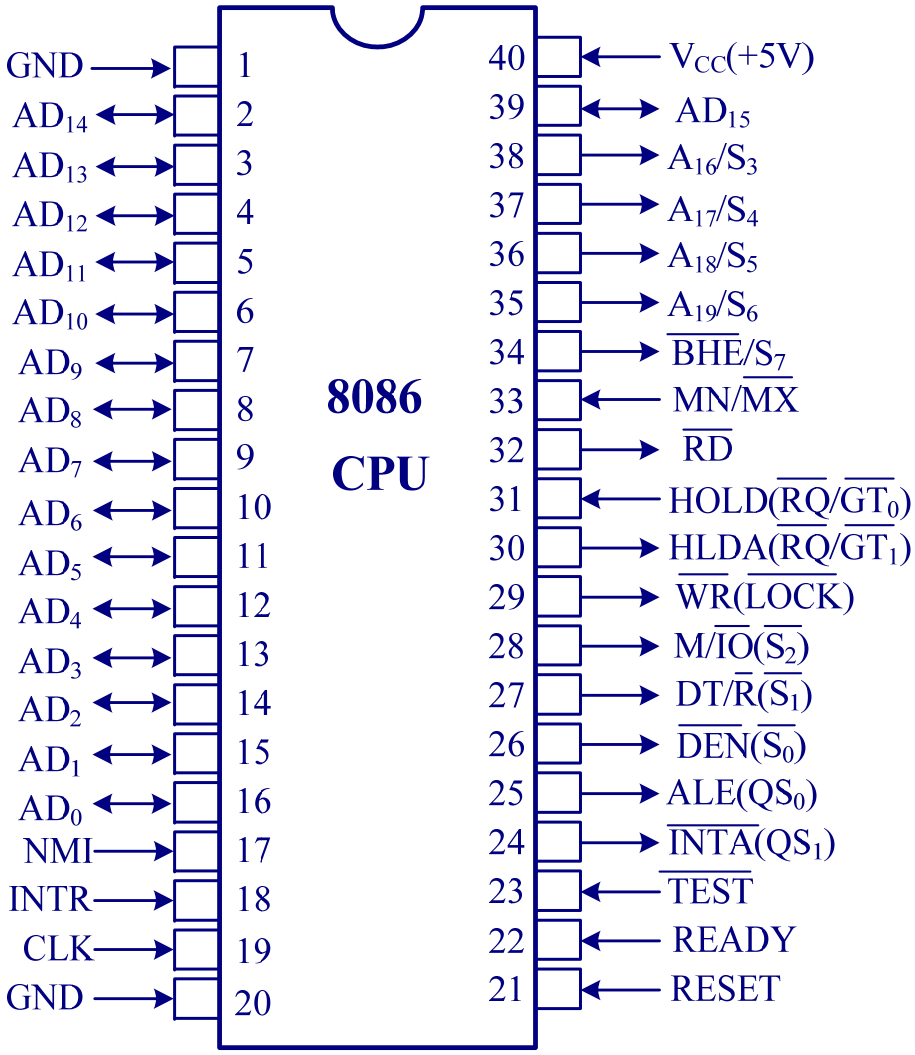
DT/~R：数据传送方向控制信号，三态。

M/~IO：输入／输出／存储器控制信号，三态。

~WR：写信号输出，三态。

HOLD：总线保持请求(其他部件共享总线)信号输入，高电平有效。

HLDA：总线保持响应信号输出，高电平有效。

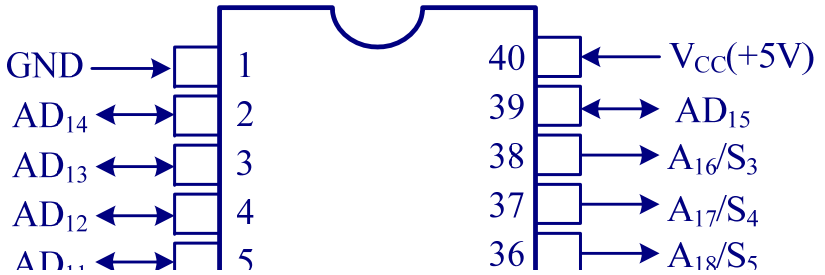


最大模式下引脚24~31的功能定义为：

QSl、QS0： 指令流队列状态输出。

提供指令流队列的状态，以便外部逻辑跟踪CPU内部的指令流队列。

~$S_2$、 ~$S_1$ 、 ~$S_0$ ： 总线周期状态信号输出，低电平有效，三态。这3




| $QS_1$ | $QS_0$ | |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | 队列中 |
| 1 | 0 | |
| 1 | 1 | 队列中 |

| $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$ | 对应的操作 |
|---|---|
| 0 0 0 | 发中断响应信号 |
| 0 0 1 | 读I / O端口 |
| 0 1 0 | 写I / O端口 |
| 0 1 1 | 暂停 |
| 1 0 0 | 取指令 |
| 1 0 1 | 读存储器 |
| 1 1 0 | 写存储器 |
| 1 1 1 | 无作用 |

号连

可产

~LC

有效

~RC

／总

既是总线请求输入，也是总线响应

但 ~RQ/~$GT_0$比~RQ/~$GT_1$具有

的优先权。

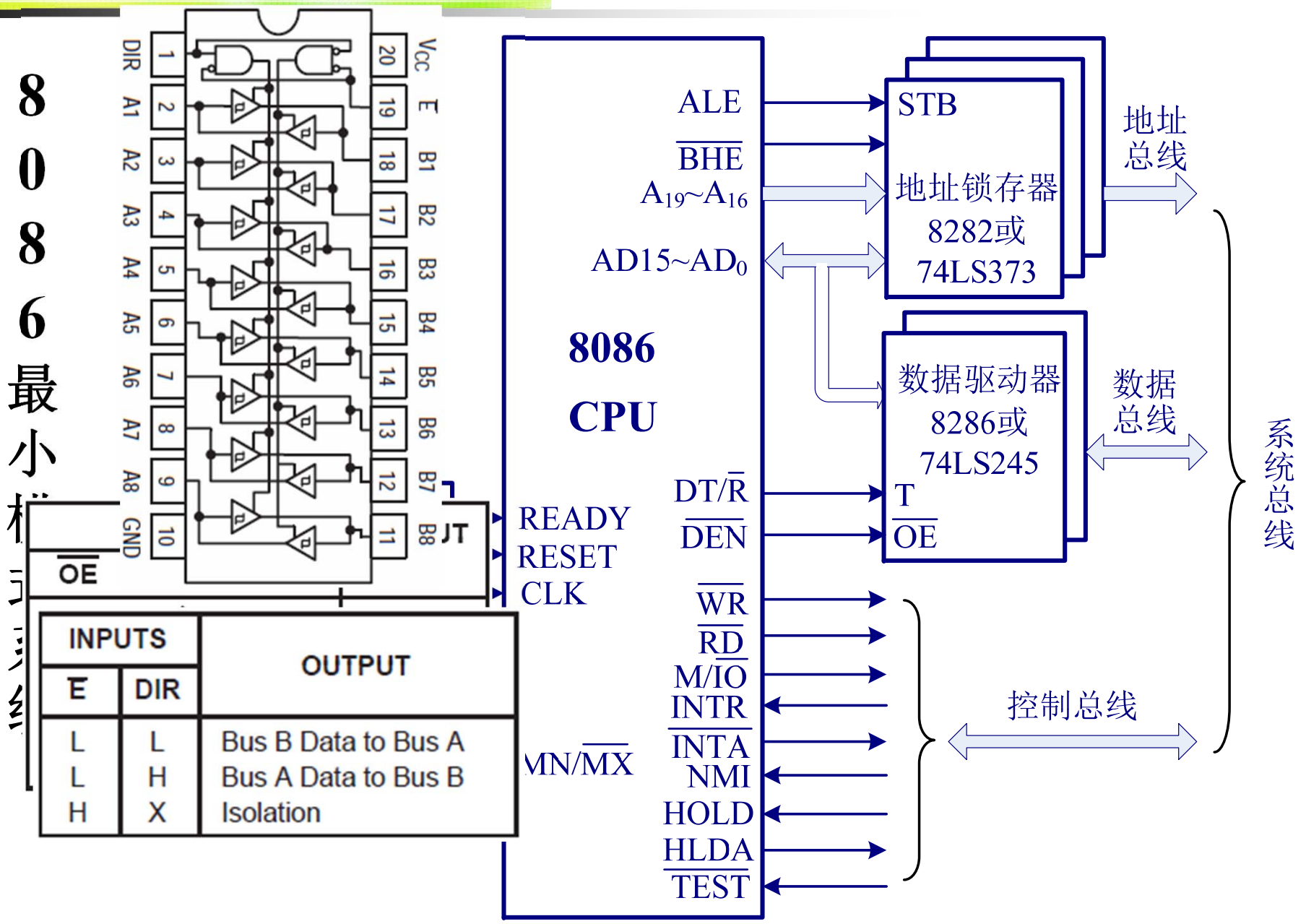
# 二、 8086系统配置

8086具有两种工作模式

最小模式：CPU仅支持由少量设备组成的单微处理器系统而不支持多处理器结构，小系统所需要的全部控制信号都由CPU直接提供。

最大模式：系统中除了有8086 CPU之外，还可以接另外的处理器(如8087数学协处理器)，构成多微处理器系统。此时CPU不直接提供读/写命令等控制信号，而是将当前要执行的传送操作类型编码成3个状态位输出，由总线控制器译码后产生相应控制信号。其他的控制引脚则直接提供所需要的控制信号。

# 8086最小模式系统


DIR
A1
A2
A3
A4
A5
A6
A7
A8
GND
VCC
B1
B2
B3
B4
B5
B6
B7
B8
OE
JT
INPUTS
OUTPUT
L
H
X
Bus B Data to Bus A
Bus A Data to Bus B
Isolation
ALE
BHE
A19~A16
AD15~AD0
8086
CPU
READY
RESET
CLK
MN/MX
DT/R
DEN
WR
RD
M/IO
INTR
INTA
NMI
HOLD
HLDA
TEST
STB
地址锁存器
8282或
74LS373
地址
总线
数据驱动器
8286或
74LS245
数据
总线
控制总线
系统总线

## 1.地址锁存器

- 将CPU发出的动态地址锁存，即锁存器。
- 地址和数据（0～15）与状态（16～19）分时复用，先输出地址，后输出数据/状态，然后利用这些稳定的地址，选择某个存储单元或I/O口来读/写。
- Intel 8282锁存器：8位锁存器（8个D锁存器），三态输出。
- 74LS373：8D锁存器，三态输出。

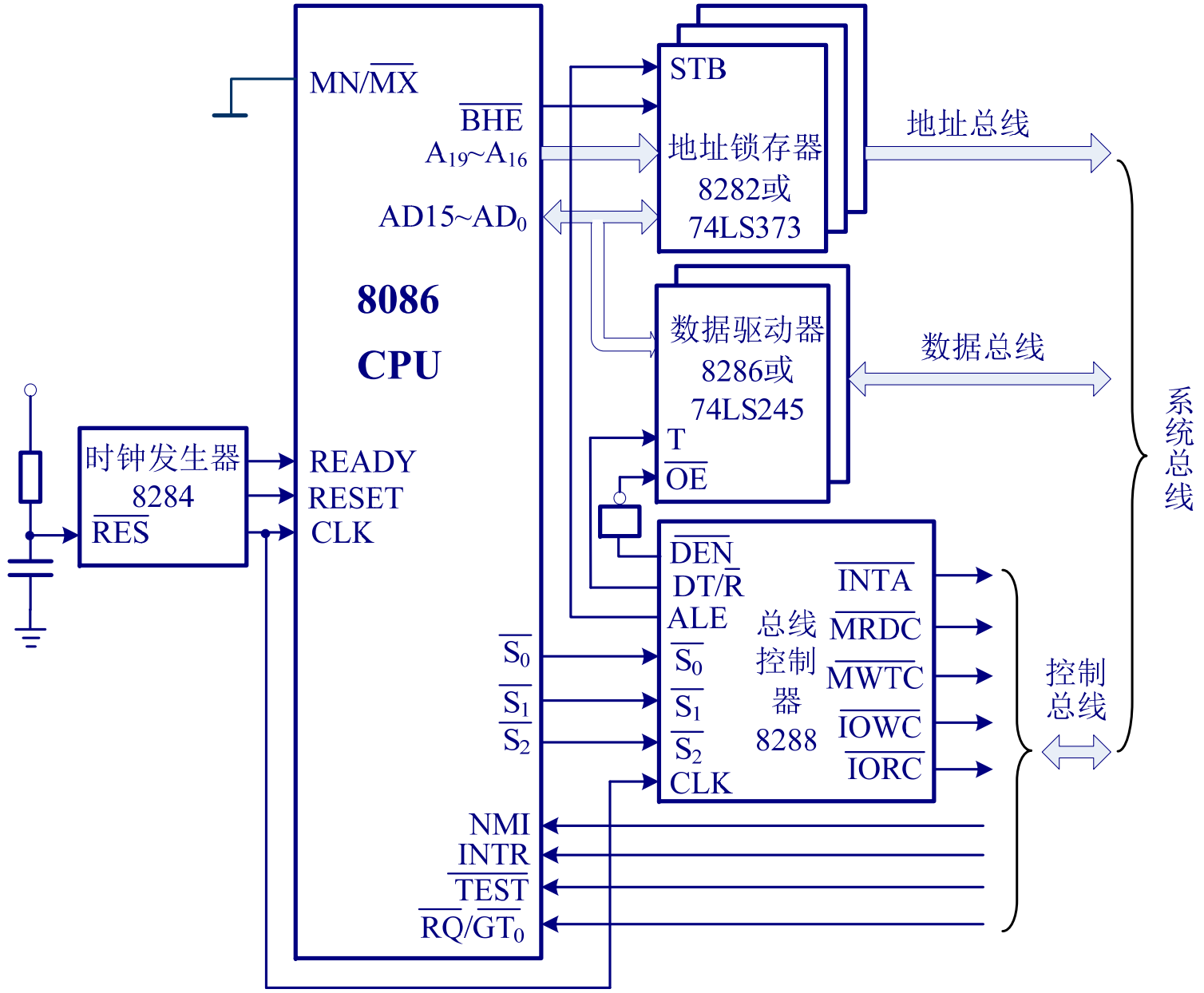
## 2. 双向总线驱动器（数据缓冲器）

- 增加8086的输出数据的驱动能力，隔离系统数据总线与CPU数据线（DMA期间需要隔离），实现双向收发。
- Intel 8286：收发器（8位总线收发器）；
- 74LS245： 8总线传送器，非反相三态门。

# 3. 时钟信号发生器

- 向系统提供符合定时要求的时钟信号CLK，准备好信号READY，复位信号RESET
- Intel 8284：时钟产生、复位、准备就绪电路
- 以PC为例：8284内部一晶体震荡器，外接石英晶体，便产生和晶体相同振荡频率的时钟OSC，经三分频成CLK时钟信号，再二分 频成PCLK，作为某些外设时钟（主要是8253计数器）
- 加电或按CTRL-ALT-DEL键时，开关电源产生电源的~RES信号送8284，内部复位逻辑便产生系统复位信号RESET。
  当等待状态逻辑电路产生的准备就绪RDY及地址允许信号~AEN有效时，使8284和时钟同步产生准备就绪READY信号。

# 8086最大模式系统


MN/MX
BHE
A19~A16
AD15~AD0
8086
CPU
READY
RESET
CLK
时钟发生器
8284
RES
STB
地址锁存器
8282或
74LS373
地址总线
数据驱动器
8286或
74LS245
T
OE
数据总线
系统总线
DEN
DT/R
ALE
S0
S1
S2
CLK
总线
控制
器
8288
INTA
MRDC
MWTC
IOWC
IORC
控制
总线
NMI
INTR
TEST
RQ/GT0

## 总线控制器8288

- 最大模式时，总线控制信号（如ALE、存储器读/写、I/O读写等）不能由8086直接提供，它只提供状态信号$S_0$～$S_2$，8086对此译码转换为总线控制信号。
- 电路组成：

  状态译码：对$S_0$～$S_2$译码；

  命令信号发生器：产生命令信号；

  控制信号产生器：产生总线控制信号；

  控制逻辑：控制8288工作方式。

# 5.3 8086存储器组织

## 一、 物理存储系统

- 位(Bit)：存储器的最小最基本单位，存放一个二进制数0或1。整个存储器由许许多多存储位构成。
- 字节(Byte)：8个Bit组成一个字节，存放相邻的8位二制数。字节的长度是固定的。
- 字(Word)：计算机内部进行数据处理的基本单位，通常与计算机内部的寄存器、ALU宽度一致。计算机的每一个字所包含的二进制位称为字长。如:Z80微机为8位机；8086，80286微机为16位机；386，486，Pentium微机为32位机。

- 存储容量：表示存放二进制代码的个数，用包含多少个存储单元，而每个单元又包含多少位来表示。微机中常以:字(节)数*字(节)的位数来表示，如:1024*8→1k*8→1KB
- PC系列微机存储器结构----以字节编址，即8位为一个单元。每个单元有一个唯一的地址代号。如PC微机的物理地址:00000H～0FFFFFH。

注:目前为了表示数据的方便，常把2个字节称为一个字，双字即为32位。

# ■数据存放规律

■字节数据：一个数存放一单元，
如:11H→00010H单元

■字数据：用二个连续单元存放，规定由2个单元中地址较小的一个确定。
如:2233H→00011H

“低对低,高对高”的存放规律

如:-4→00013H

■机器指令(机器码)：按字节顺序存放。
如:MOV BX，AX

→89C3H→00015H

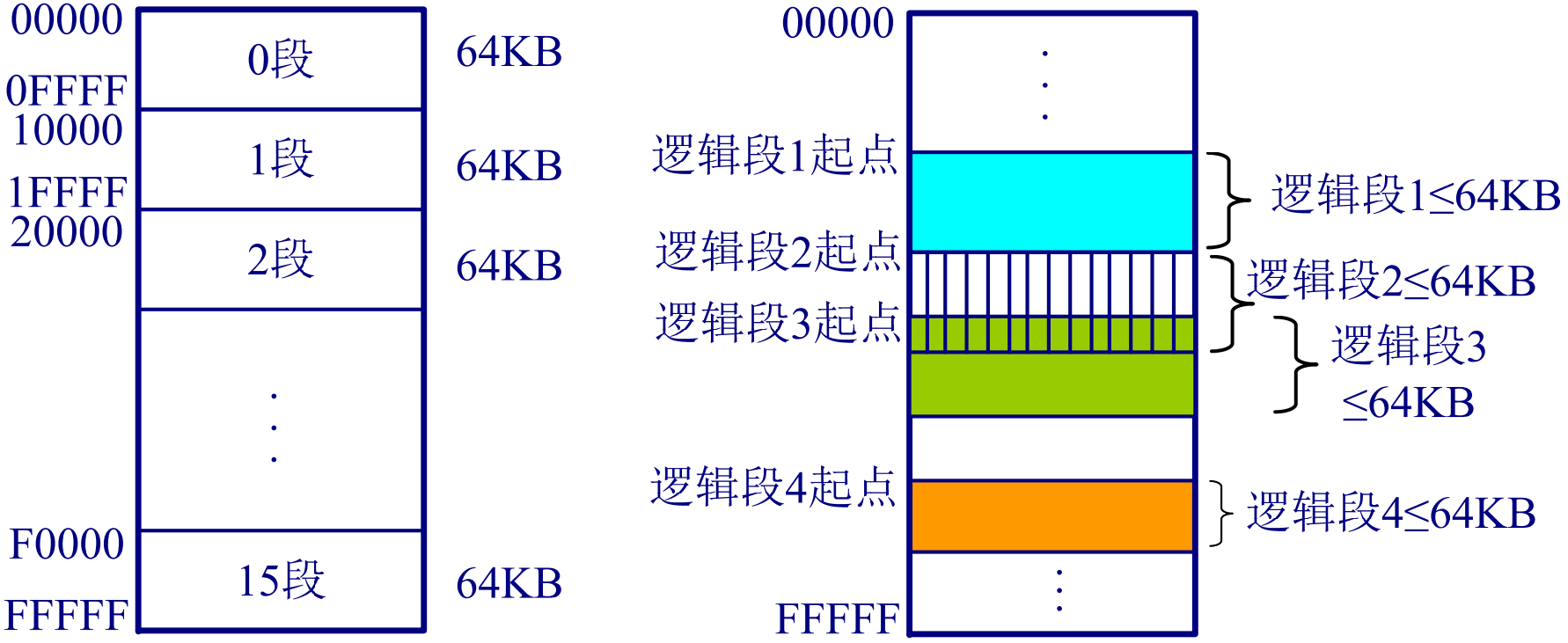
■字符串：从低地址开始，以ASCII码顺序存放，如:‘ABC’→00017H

| | |
|---|---|
| 00010H | 11 |
| 00011H | 33 |
| 00012H | 22 |
| 00013H | FC |
| 14H | FF |
| 15H | 89 |
| 16H | C3 |
| 17H | 41 |
| 18H | 42 |
| 19H | 43 |

# 二、存储器的分段结构

- 内部寄存器都是16位，不能直接输出20位地址，所以分段管理。
- 把1M分成许多个段，每一段最多可寻址$2^{16}$=64K个单元。
- 规定每个段地址的低4位为0，即能被16整除。段地址和偏移地址都是16位无符号数，所以分段并不是唯一的，可以相互重迭。
- CPU内部仅有四个段寄存器，所以在某个特定时刻仅能访问四个段。

# 8086/8088存储器的分段结构：逻辑段长$2^{16}$=64k字节



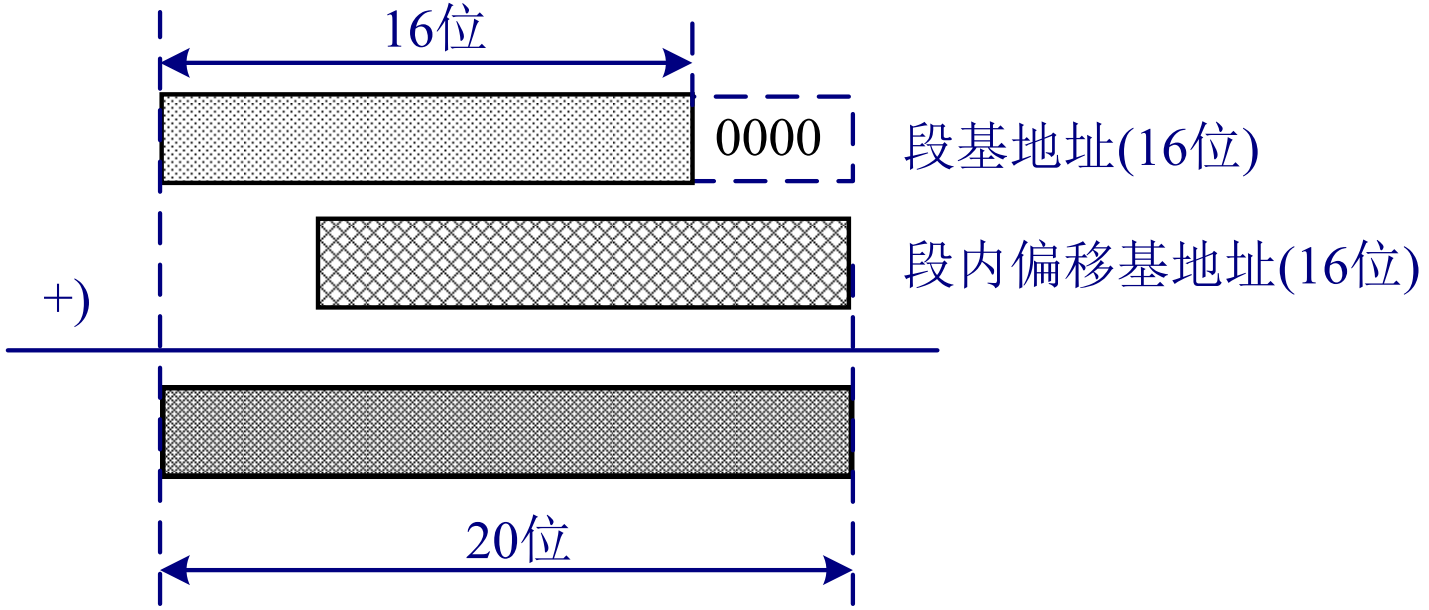
# 三、逻辑地址、物理地址及物理地址的形成

1.每个存储单元都可用地址的两种形式来表示

- **物理地址**：用唯一的20位二进制数表示，CPU访问时用物理地址。
- **逻辑地址**：段地址:偏移地址，程序中使用逻辑地址。如:03C0:0010

段基址（即段起始地址）=段地址<<4

20位物理地址=段地址<<4+偏移地址




例：若逻辑地址为3A00H:12FBH，对应的物理地址是3B2FBH；若逻辑地址是8000H:1200H，则对应的物理地址为81200H。

一个物理地址所对应的逻辑地址不是惟一的。 物理地址3B2FBH的逻辑地址可以是3A00H:12FBH，3000H:B2FBH，3B00H:02FBH等。

## 2. 8086对段地址与偏移地址的指定：

| 序号 | 内存访问类型 | 默认段寄存器 | 可指定段寄存器 | 段内偏移地址来源 |
|---|---|---|---|---|
| 1 | 取指令 | CS | 无 | IP |
| 2 | 堆栈操作 | SS | 无 | SP |
| 3 | 源串 | DS | CS、ES、SS | SI |
| 4 | 目的串 | ES | 无 | DI |
| 5 | BP用作基址寻址 | SS | CS、ES、DS | 按寻址方式计算得到的有效地址 |
| 6 | 一般数据存取 | DS | CS、ES、SS | 按寻址方式计算得到的有效地址 |

例:某可执行程序为2KB，已知CS=1063H，IP=0000H，求该程序的末地址(物理地址)。

末地址＝长度-1
＝2K-1=$2^{11}$-1
=0800H-1=07FFH

解 程序存放:

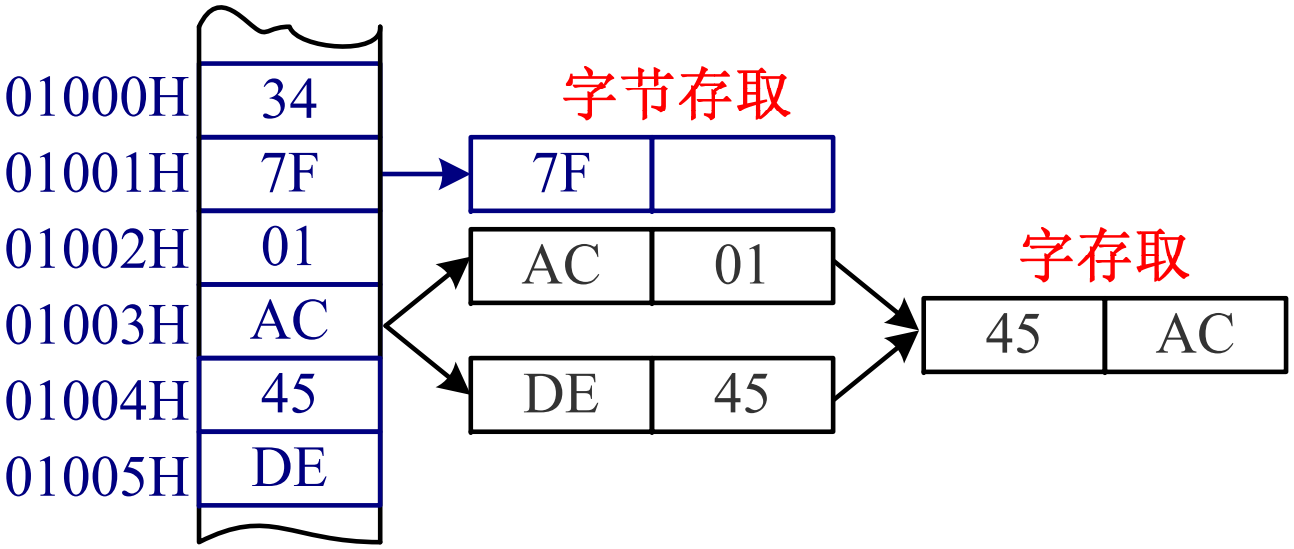
1063:0000
:
1063:07FF

所以末地址为:10630H+07FFH=10E2FH

# 四、数据的存取

## 1. 数据对准：

当CPU读/写一个字时，若字单元地址从偶地址开始，只需访问一次存储器；若字单元地址从奇地址开始，则需访问两次存储器。



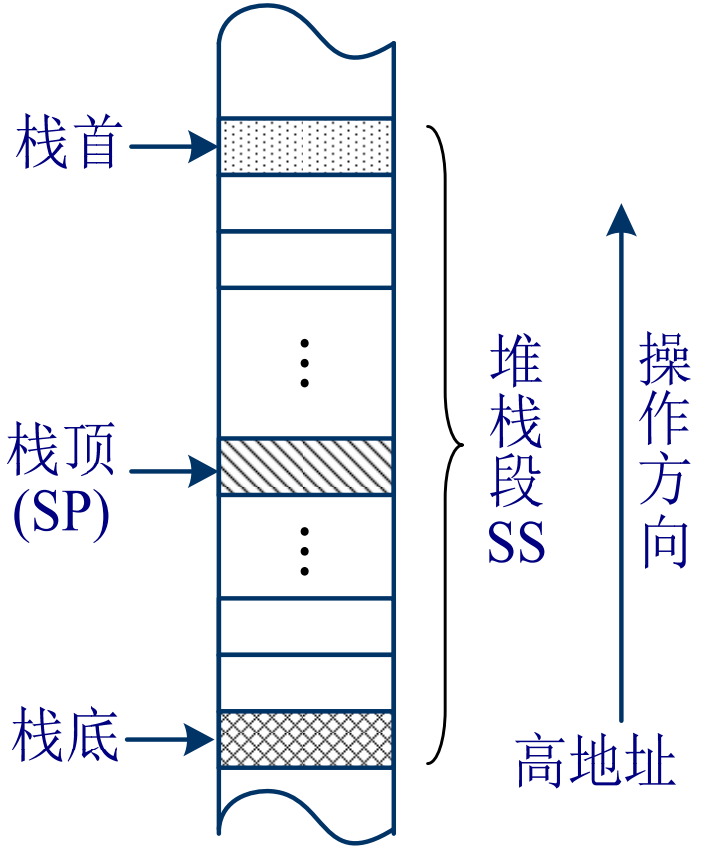
## 2. 堆栈的概念：

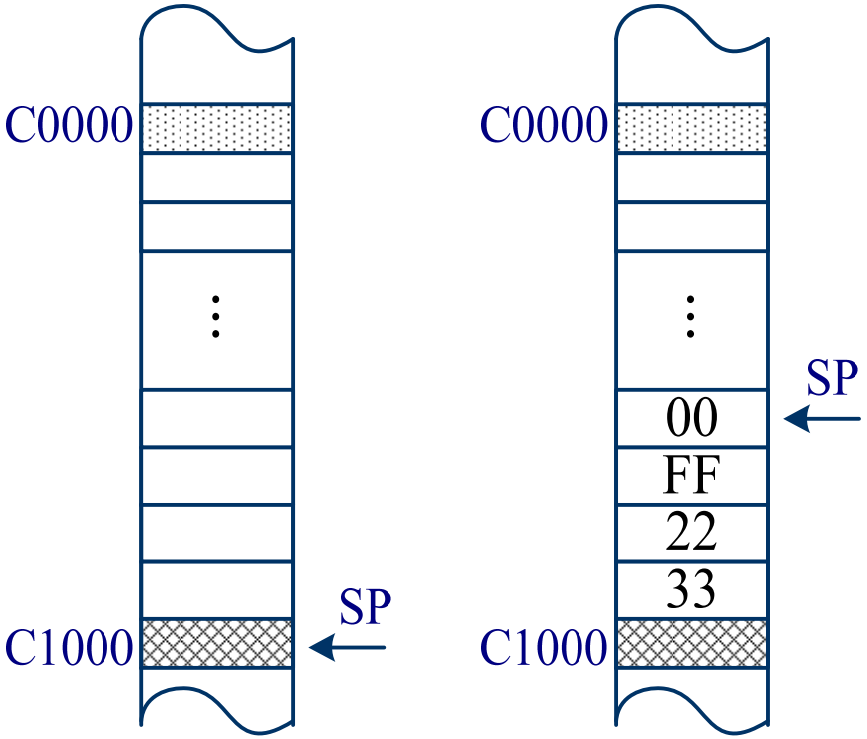
堆栈(STACK)是由若干个连续存储单元组成的、操作时遵循先进后出原则的一个存储器区，主要用于暂存中断和子程序调用时的现场数据及返回地址。

堆栈操作具有如下特点：

①操作只能在栈顶进行，入栈和出栈都必须是双字节数据(16位)。进行一次入栈操作，SP减2；进行一次出栈操作，SP加2；

②操作遵循“先进后出(FILO)”的原则。最后压入堆栈的数据会最先被弹出。



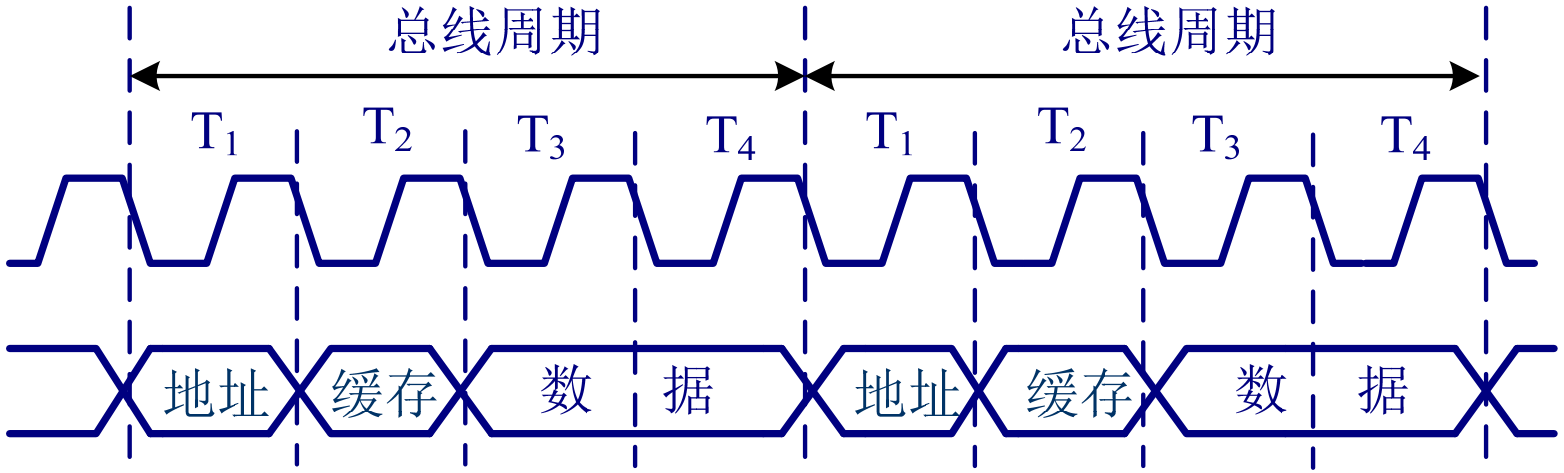



```
MOV AX, 3322H
MOV BX, 0FF00H
PUSH AX
PUSH BX
POP CX
POP DX
```

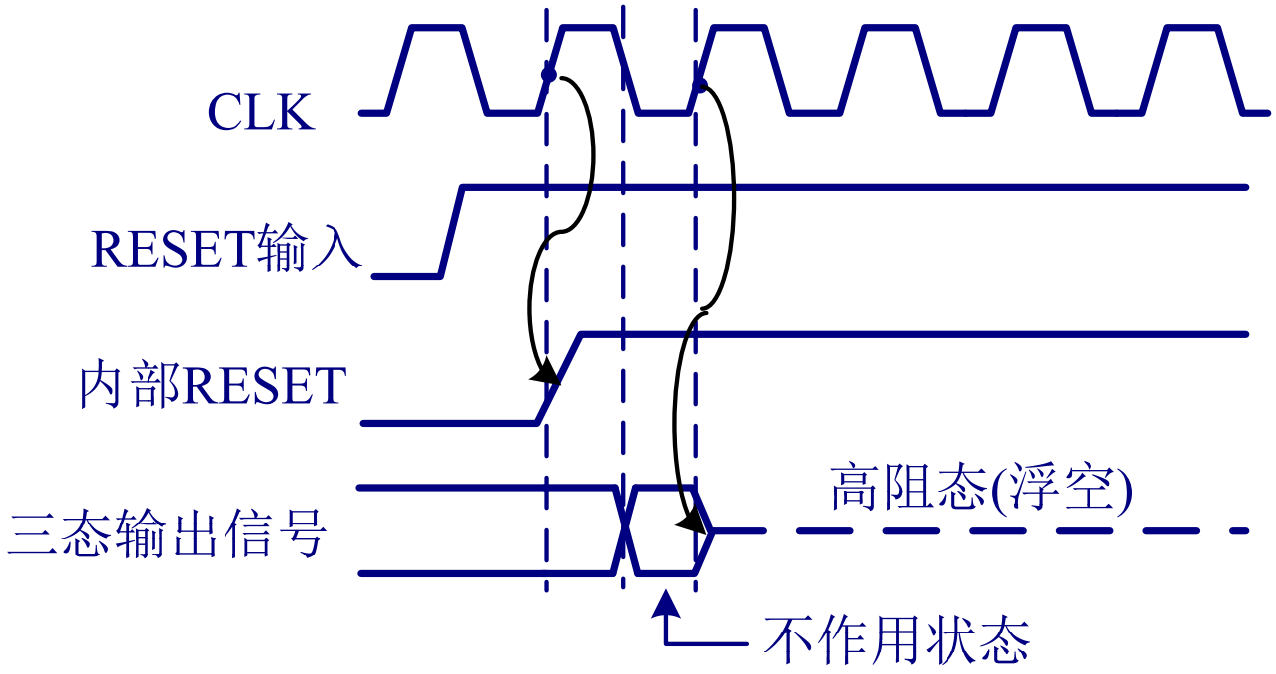
# 5.4 8086CPU时序

- 微处理器是在统一的时钟信号CLK控制下，按照一定的时序来工作的。
- 8086时钟频率为5 MHz，一个时钟周期等于200 ns。
- CPU的时序分为两种：时钟周期和总线周期。
- 8086CPU与内存或接口间的通信都是通过总线来进行的。通过总线对存储器或I／O接口进行一次访问所需的时间叫做一个总线周期，一个总线周期包括多个时钟周期。
- CPU每执行一条指令至少要访问一次存储器(取指令)，即至少要进行一次读存储器操作，占用一个读总线周期。

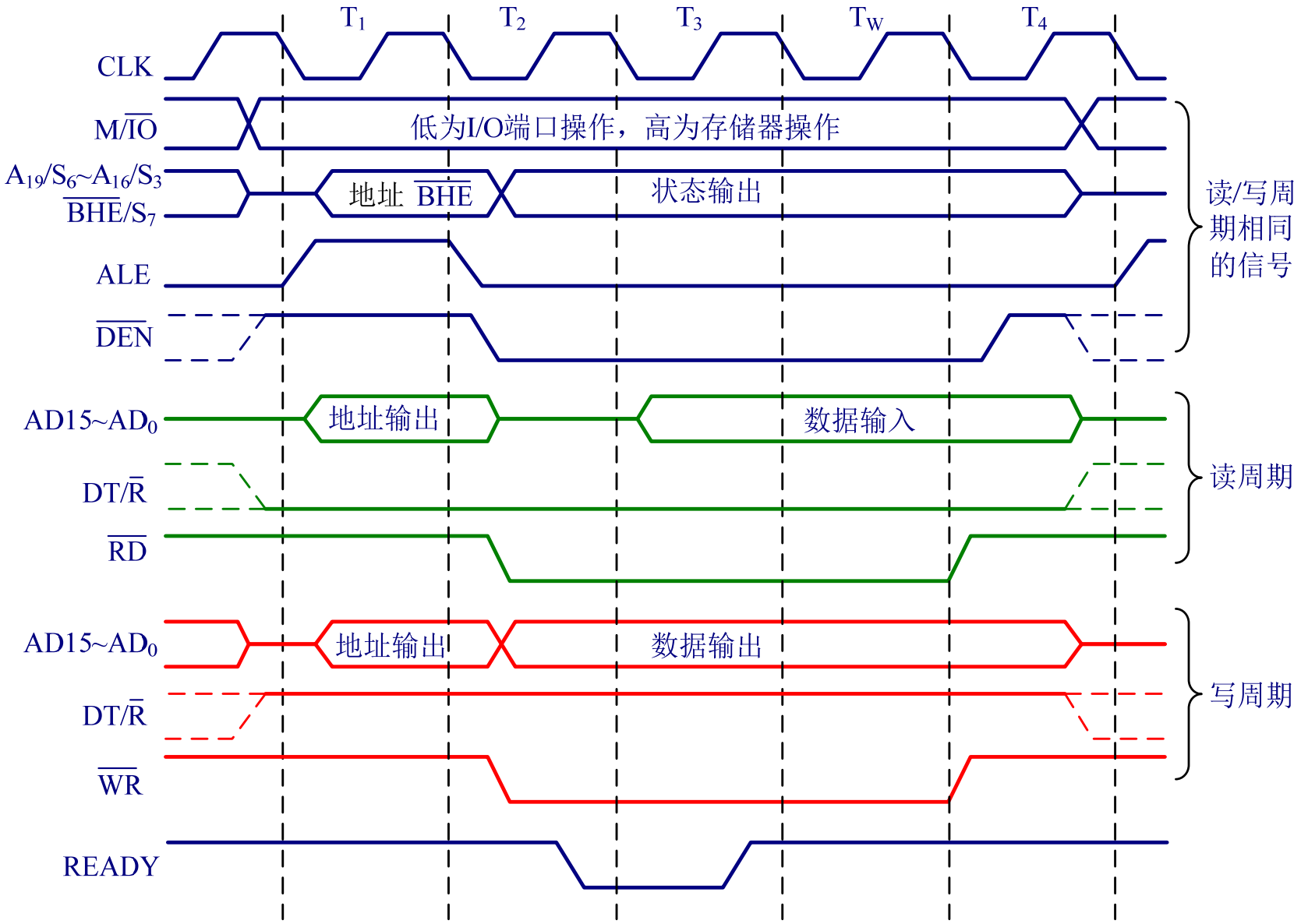
- 一条指令的执行需要若干个总线周期才能完成。而一个总线周期又由若干个时钟周期构成。
- 每个时钟脉冲的持续时间就称为一个时钟周期。8086 CPU的一个读(或写)总线周期至少包括4个时钟周期。
- 时钟周期越短，CPU执行的速度就越快。



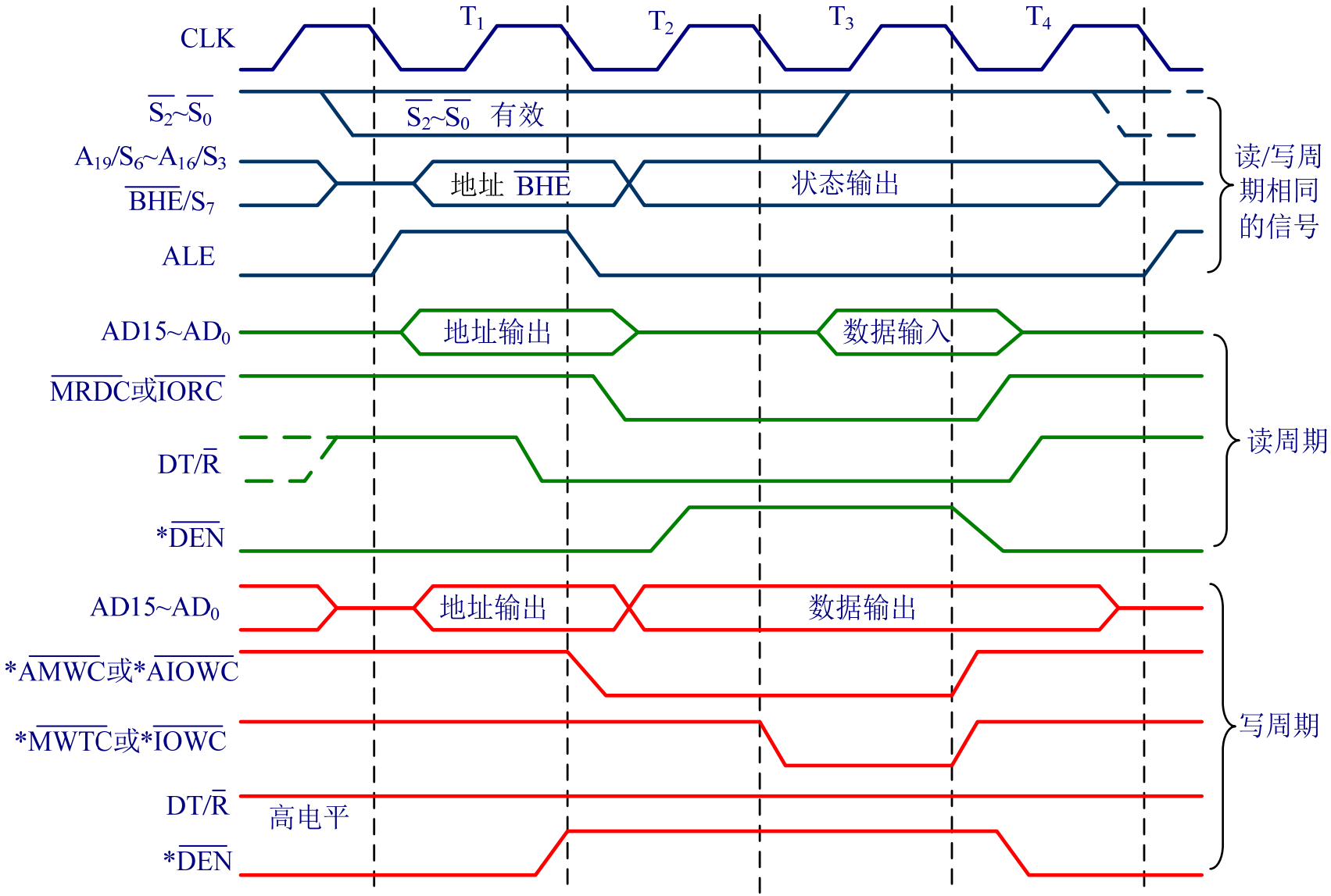
# 系统的复位和启动操作时序



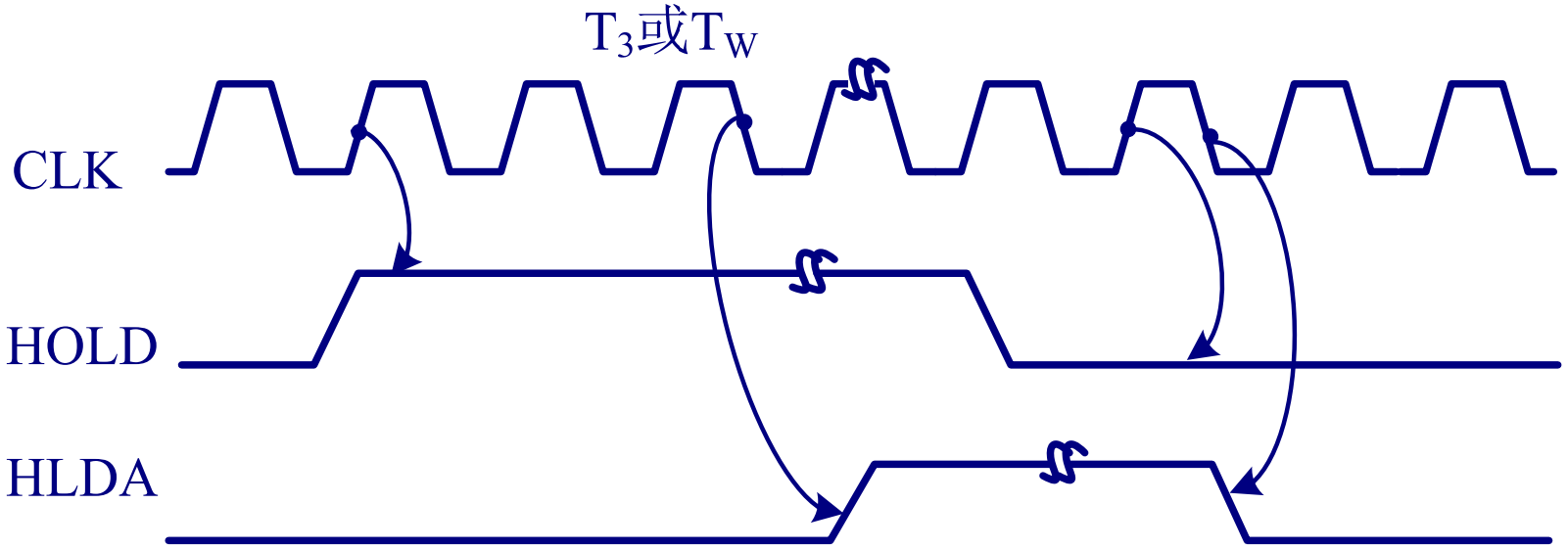
# 最小模式下的工作时序


T1
T2
T3
TW
T4
CLK
M/IO
低为I/O端口操作，高为存储器操作
A19/S6~A16/S3
BHE/S7
地址 BHE
状态输出
ALE
DEN
读/写周期相同的信号
AD15~AD0
地址输出
数据输入
DT/R
RD
读周期
AD15~AD0
地址输出
数据输出
DT/R
WR
写周期
READY

# 最大模式下的工作时序


T1
T2
T3
T4
CLK
S2~S0
S2~S0 有效
A19/S6~A16/S3
BHE/S7
地址 BHE
状态输出
ALE
读/写周期相同的信号
AD15~AD0
地址输出
数据输入
MRDC或IORC
DT/R
*DEN
读周期
AD15~AD0
地址输出
数据输出
*AMWC或*AIOWC
*MWTC或*IOWC
DT/R
高电平
*DEN
写周期

# 最小模式下的总线保持



# 5.5 8086CPU寻址方式和指令系统

## 一、 汇编语言概述

编程语言分:

机器语言：计算机能直接识别并执行某种操作的二进制代码串

汇编语言：用指令助记符、符号地址和标号等书写程序的语言

高级语言

汇编语言程序优点:

（1）占用空间小

（2）运行速度快

（3）有直接控制硬件的能力

开发环境：

DOS环境

开发工具：

MICROSOFT： MASM（宏汇编）

INTEL： ASM （小汇编）

BORLAND： TASM

需要的最基本的软件：

文本编辑器（EDIT.EXE） 编辑成*.ASM

ML.EXE： 汇编成*.OBJ

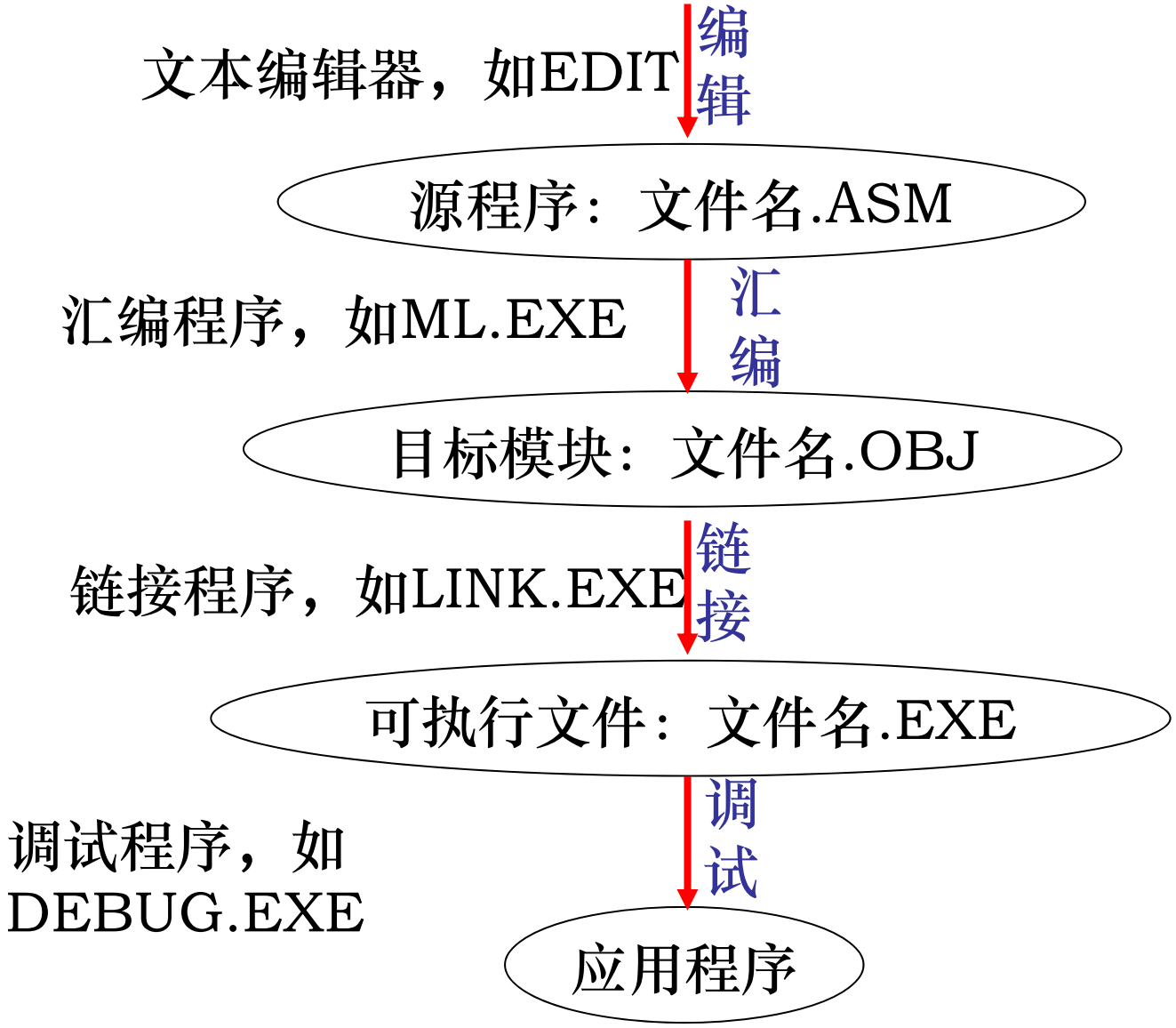
LINK.EXE： 连接成*.EXE

DEBUG.EXE： 调试

开发过程:

| | |
|---|---|
| （1）编辑源程序生成*.ASM | EDIT *.ASM |
| （2）汇编生成目标程序*.OBJ: | ML *.ASM; |
| （3）连接生成可执行程序*.EXE: | LINK *; |
| （4）调试: | DEBUG *.EXE |
| （5）执行: | 可执行文件名 |

或者（1）-（5）全在debug中完成，只能编写小程序，数据、代码、堆栈全在同一段内。


文本编辑器，如EDIT
编辑
源程序：文件名.ASM
汇编程序，如ML.EXE
汇编
目标模块：文件名.OBJ
链接程序，如LINK.EXE
链接
可执行文件：文件名.EXE
调试程序，如DEBUG.EXE
调试
应用程序

# 汇编语言程序举例：

语句格式：
[标号] 助记符 [操作数1,[操作数2]];注释

```
DATA SEGMENT ;数据段
DA1  DB 'Welcome to this course!',0DH,0AH
     DB 'This is a sample program.','$'
DATA ENDS ;数据段结束
STACK SEGMENT ;堆栈段
ST1  DB 100 DUP(?)
STACK   ENDS ;堆栈段结束
CODE SEGMENT ;代码段
  ASSUME CS:CODE,DS:DATA,SS:STACK
  MAIN PROC  FAR
  START: MOV AX,STACK ;送堆栈段段地址
       MOV SS,AX
       PUSH DS ;返回DOS用
       MOV AX,0
       PUSH AX
       MOV AX,DATA ;送数据段段地址
       MOV DS,AX
       MOV AH,9  ;DOS 9号功能调用，显示字符串
       MOV DX,OFFSET DA1
       INT 21H
       RET
  MAIN   ENDP
CODE ENDS ;代码段结束
END START ;程序结束
```

# 二、 操作数寻址方式

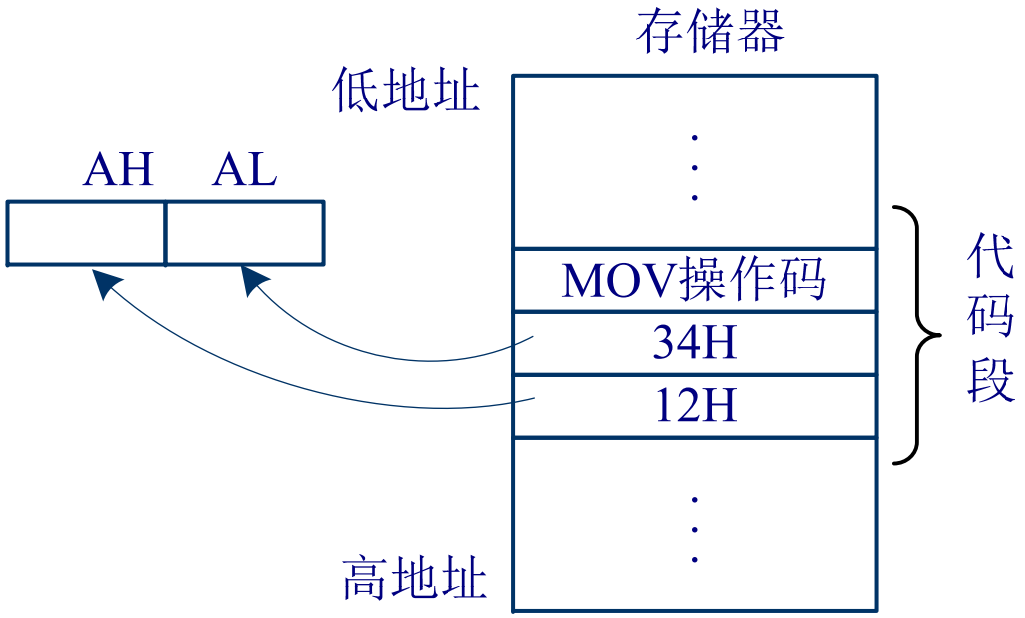
寻址方式：指令中给出操作数的方法。

| | |
|---|---|
| 立即数操作数： | 立即寻址 |
| 寄存器操作数： | 寄存器寻址 |
| 存储器操作数： | 直接寻址<br>寄存器间接寻址<br>寄存器相对寻址<br>基址变址寻址<br>相对基址变址寻址 |

# 1．立即寻址

(1) 格式：操作数以常数的形式直接表示在指令中。

```
MOV  AL，5                    ；AL←5
MOV  AX，1234H                ；AX←1234h
                              ；AH=12H，AL=34H
```



(2) 讨论：

- 适用于给寄存器或存储单元赋初值。立即数在内存的代码段内，指令执行时不需再存取存储器
- 不能直接对段寄存器用立即数赋值
- 立即数不能作目的操作数
- 给存储器操作数(默认段寄存器为DS)用立即数赋值时，要指明WORD/BYTE PTR

```
MOV  BYTE PTR [300H]，5FH
```

- 以A~F打头的数字出现在指令中时，前面要加数字0，以免与变量名等符号混淆

```
MOV       BX,0F77H
```

## 2．寄存器寻址

(1) 格式：操作数是通用寄存器或段寄存器中的内容。

寄存器可以是

通用寄存器：AX，BX，CX，DX，SP，BP，SI，DI，AH，AL，BH，BL，CH，CL，DH，DL

段地址寄存器：CS，SS，DS，ES

标志寄存器

```
INC   CX          ; CX←CX+1
MOV   SS, AX      ; SS←AX
ADD   CL, BH      ; CL←CL+BH
```

(2) 讨论：

- 操作数在CPU内的寄存器中，执行时间短；只访问寄存器，不访问内存
- CS一般不要求用户赋值。否则语法无错，执行错
- 两个操作数的长度必须一致

# 存储器寻址：

操作数在内存中除代码段以外的存储区中。

物理地址=段寄存器<<4bit+EA（有效地址或偏移地址）

确定物理地址的关键：从指令中找出EA（由若干部分合成），
再确定用哪个段寄存器。

## 3. 直接寻址

(1) 格式：操作数表示成[立即数]、[变量名]或变量名
（变量名称符号地址，DEBUG中不能用变量名）

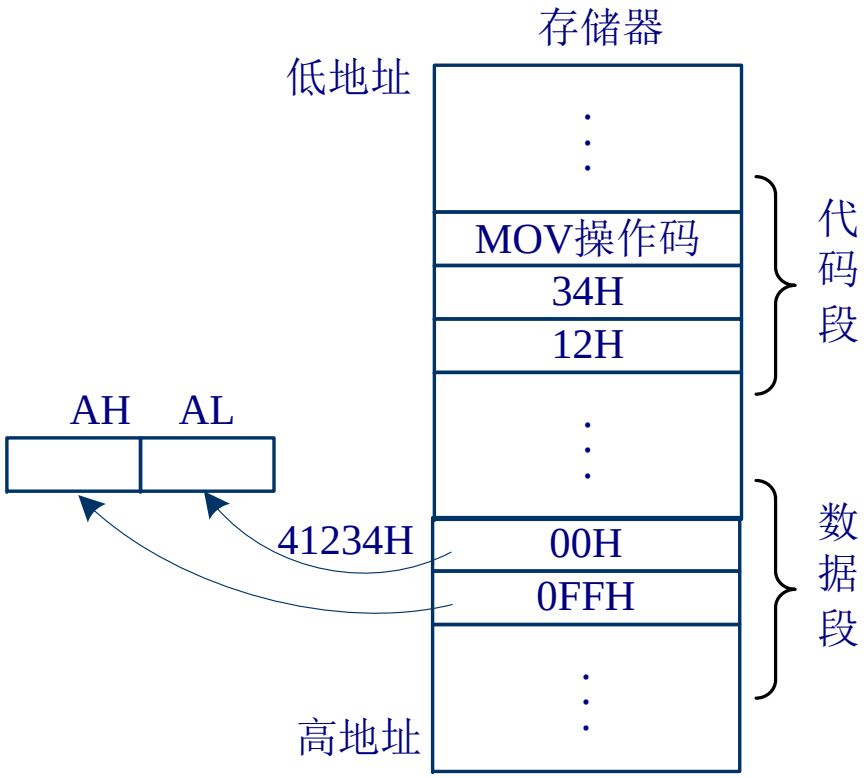
```
MOV AX，[3000H]
MOV AX，VALUE        或MOV AX，[VALUE]
MOV AX，[VALUE+2]  或MOV AX，[VALUE]+2
```

也可指明段寄存器（称段超越）：

```
MOV AX，ES：[2000H] ；“：”称修改属性运算符
MOV AX，CS：VALUE
```

(2) 物理地址：
PA=DS（或段超越指定的段寄存器）<<4+立即数或变量的偏移；

MOV AX，[1234H]；DS=4000H




(3) 讨论：不能两个操作数都用存储器寻址（对各种寻址方式）。

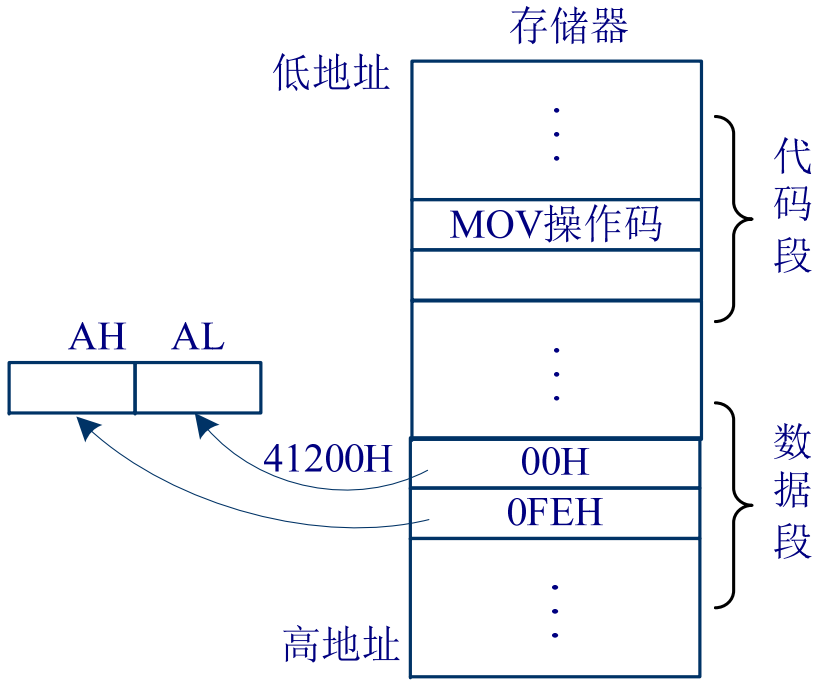
# 4. 寄存器间接寻址

（1）格式：操作数表示成[BX]、[BP]、[SI]或[DI]，
寄存器内容是操作数的有效地址

```
MOV  BX，[DI]
MOV  [SI]，DS
MOV  [BP]，AX
MOV  ES：[DI]，AX
MOV  DX，DS：[BP]
```

（2）物理地址：（缺省的情况）
PA=DS<<4bit+BX（或SI，DI）
或　PA=SS<<4+BP

MOV AX，[BX]；DS=4000H, BX=1200H




（3）讨论：

可用于：不知道内存单元的确切地址；存取一维数组或表格中的元素；只有4个寄存器可用。

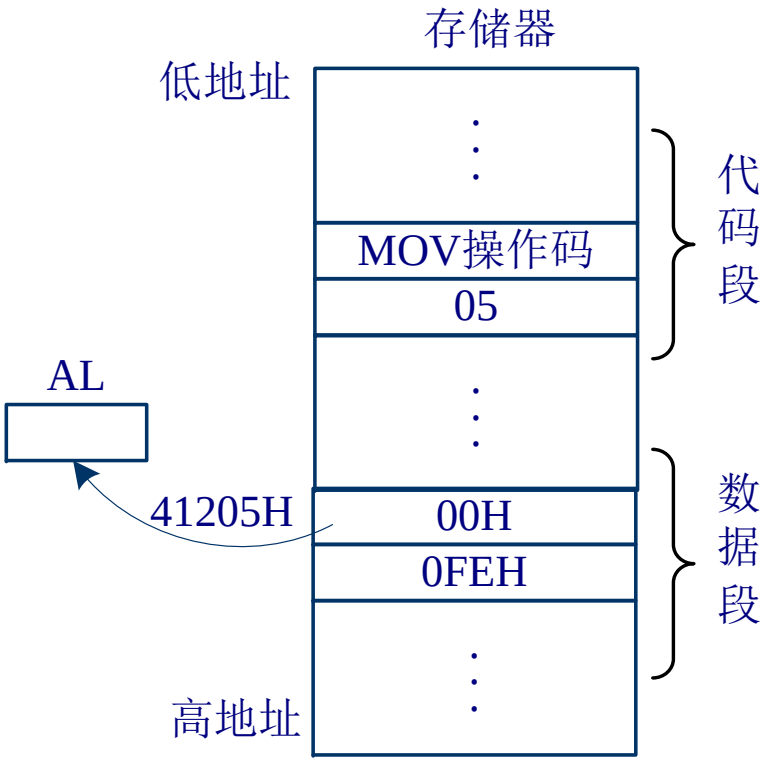
## 5. 寄存器相对寻址

（1）格式：操作数表示成[BX/BP/SI/DI+立即数/变量名]，
（变量的值是偏移地址）
是直接寻址和寄存器间接寻址的组合。

```
MOV  BX，[SI+1003H]
MOV  AL，[DI+TABLE]
MOV  AL，TABLE[BX]
MOV  AL，[BP]+TABLE
MOV  BX，ES：[SI+1003H]
MOV  TABLE[BP]，AX
```

（2）物理地址：
PA=DS<<4+（BX/SI/DI）+立即数或变量的偏移
PA=SS<<4+（BP）+立即数或变量的偏移

MOV AL，[BX+05]；DS=4000H, BX=1200H




**（3）讨论：存取一维数组或表格中的元素。**

# 6. 基址变址寻址

（1）格式：操作数表示成[BX/BP+SI/DI]

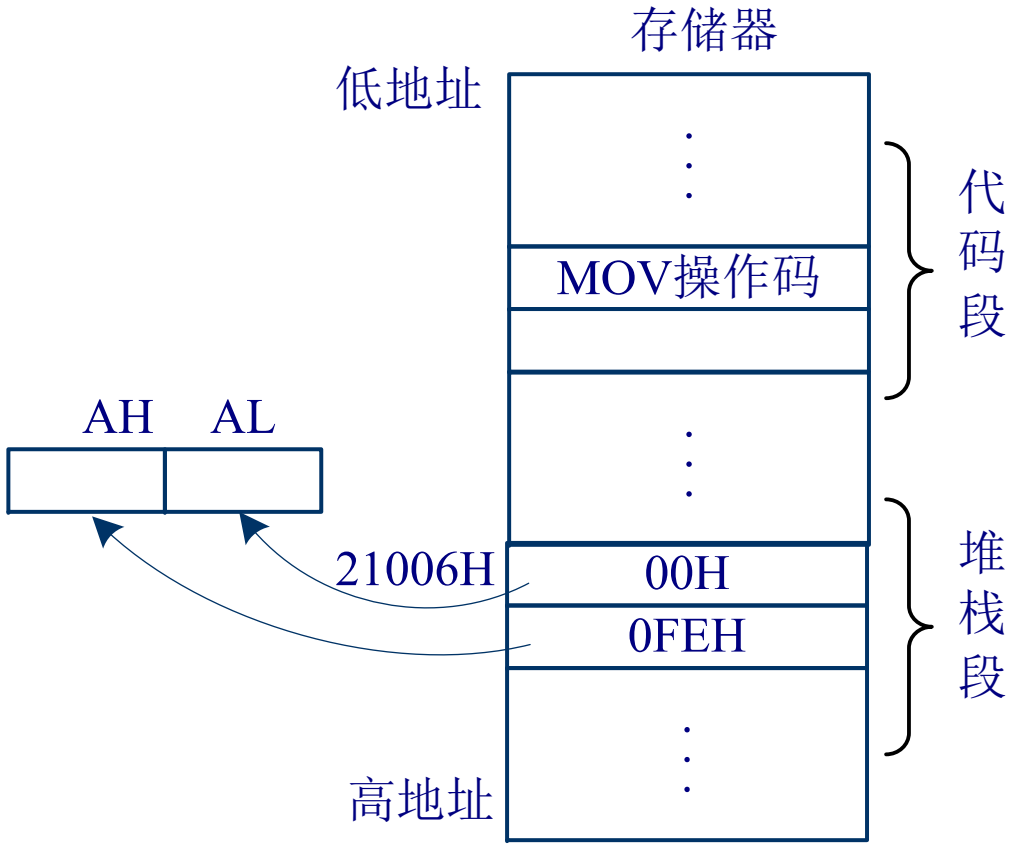
```
MOV  AX，[BX][DI]
MOV  CL，[BP+DI]
MOV  ES：[BX][SI]，AH ；或 MOV  [SI][BX]，AH
```

（2）物理地址：（段寄存器缺省的情况）

PA=DS<<4+BX+SI/DI

PA=SS<<4+BP+SI/DI

MOV AX, [BP+SI] ；SS=2000H，BP=1000H，SI=0006H




(3) 讨论：适用一维或二维数组或表格查找；
不允许将两个基址寄存器或两个变址寄存器组合在一起寻址

# 7. 相对基址变址寻址

（1） 格式：操作数表示成[BX/BP+SI/DI+立即数/变量名]

```
MOV  AX，COUNT[BX][SI]
MOV  AX，COUNT[SI][BX]
MOV  AX，[BX+COUNT][SI]
MOV  AX，[BX+SI+COUNT]
MOV  AX，[BX]COUNT[SI]
ADD  VALUE[BX][DI]，DX;
```

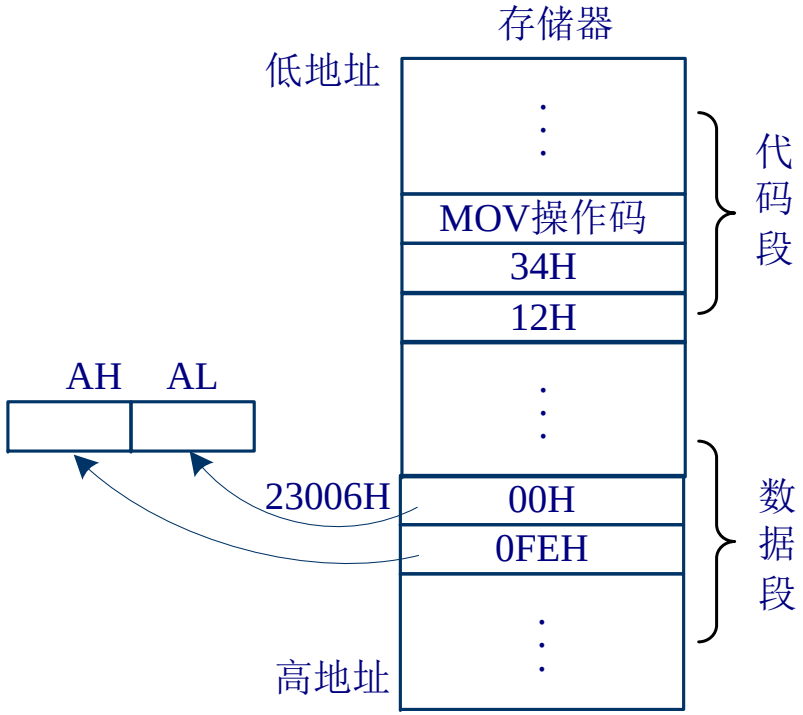
（2）物理地址：（缺省情况）

PA=DS<<4+BX+SI/DI+立即数或变量的偏移

PA=SS<<4+BP+SI/DI +立即数或变量的偏移

**MOV AX, [BX+DI+1234H]**

**；DS=2000H，BX=1000H，DI=2000H**




**(3) 讨论：适用堆栈处理，访问二维数组中某个元素；**
**不允许将两个基址寄存器或两个变址寄存器组合在一起寻址**

# 9．I/O端口寻址

(1) 直接寻址：指令中直接给出8位端口地址，只能寻址256个端口

```
IN   AL/AX，8位立即数(PORT)
OUT      8位立即数(PORT)，AL/AX
```

(2) 间接寻址：将16位端口地址放入DX

```
MOV      DX，16位立即数
IN AL/AX，DX
OUT      DX，AL/AX

IN AL,    20H   ; [20H]→AL

MOV      DX,   210H
OUT      DX,   AL     ;AL→[210H]
```

## 10．隐含寻址

在有些指令的操作码中，不仅包含了操作的性质，还隐含了部分操作数的地址。如乘法指令MUL，在这条指令中只需指明乘数的地址，而被乘数以及乘积的地址是隐含且固定的。这种将一个操作数隐含在指令码中的寻址方式就称为隐含寻址。

例： 指令“MUL BL”的功能是把AL中的内容与BL中的内容相乘，乘积送到AX寄存器。即(AL)×(BL)→AX。这条指令隐含了被乘数AL及存放结果的累加器AX。

例：假设DS=2000H，ES=2100H，SS=1500H，SI=00A0H，DI=0，BX=0100H，BP=0010H，数据变量VAL在数据段中的偏移地址为0050H，指出下列指示的源操作数是什么寻址方式，写出它的物理地址。（5-10自己练习）

(1) mov al,0abh 立即，无
(2) mov ax,val 直接，20050H
(3) mov ax,val[DI] 寄相，20050H
(4) mov ax,val[BX][SI] 相基变，201F0H
(5) mov ax,bx
(6) mov ax,[100h]
(7) mov ax,[bx]
(8) mov ax,ES:[BX]
(9) mov ax,[BP]
(10) mov ax,[SI+10H]

例：（DS）=3000H，（BX）=1100H，（CS）=0062H，
（SI）=0002H，（31100H）=52H，（31101H）=8FH，
（31162H）=6BH，（31163H）=99H，（31103H）=F6H，
（32200H）=AAH，（32201H）=B6H，（32800H）=55H，
（32801H）=77H，写出下列指令执行后AX的内容

[（3—6）自己练习]

（1）MOV　AX，1100H[BX]

AX=（DS：1100+BX）=（32200H）=B6AAH

（2）MOV　AX，[2800H]

AX=（DS：2800）=（32800H）=7755H

（3）MOV　AX，BX

（4）MOV　AX，[BX]

（5）MOV　AX，[1160H+SI]

（6）MOV　AX，4200H

# 例：找错

1. MOV 3,SI
2. MOV CH,1234H
3. MOV [BX],33H
4. MOV AX,CL
5. MOV AX,[DX]
6. MOV X,[100H]
7. MOV [100H],[DI]
8. MOV DS,1000H
9. MOV CS,AX
10. MOV [AX],BX

# 三、 转移地址寻址

1. 段内直接转移
2. 段内间接转移
3. 段间直接转移
4. 段间间接转移

用于控制程序流程的指令：转移(JMP,JZ等）、循环(LOOP)、过程调用(CALL)、中断(INT)

1．段内直接（相对）转移寻址： 短转移：8位偏移
近转移：16位偏移

（1）格式：操作码 [SHORT 或NEAR PTR] 近标号。

```
JMP   NEXT          ；NEXT是一个短标号或近标号
CALL SUB            ；SUB是一个段内子过程名
```

在DEBUG中不能用标号和过程名：

```
JMP   110H          ；执行CS:110处的指令
```

（2）转移的目标地址：

CS不变；IP=当前IP+指令中的8/16位偏移量

（3）注意：

条件转移只能段内直接短转移

2．段内间接转移寻址：

（1）格式：操作码　16位通用寄存器操作数/字类型存储器操作数

```
JMP     BX
JMP     AX
JMP     SI
JMP     TABLE[BX]           ；TABLE是字变量
JMP     WORD PTR[BP][DI]
JMP     [BX]                ；缺省表示是字
```

（2）转移的目标地址：

CS不变；IP=寄存器或连续2字节存储单元的内容

3．段间直接转移寻址：

(1) 格式：操作码　　[FAR PTR] 远标号

　　JMP　　LABEL　　；LABEL为远标号

(2) 转移的目标地址: CS=标号的段地址；IP=标号的偏移地址

4．段间间接转移寻址：

(1) 格式：操作码　FAR [PTR]或DWORD PTR　存储器操作数

　　JMP　　DWORD PTR [BP][SI]

　　JMP　　VAR　　；VAR是双字变量

（2）转移的目标地址：

CS=高2字节存储单元的内容；IP=低2字节存储单元的内容。

| 序号 | 内存访问类型 | 默认段寄存器 | 可指定段寄存器 | 段内偏移地址来源 |
|---|---|---|---|---|
| 1 | 取指令 | CS | 无 | IP |
| 2 | 堆栈操作 | SS | 无 | SP |
| 3 | 源串 | DS | CS、ES、SS | SI |
| 4 | 目的串 | ES | 无 | DI |
| 5 | BP用作基址寻址 | SS | CS、ES、DS | 按寻址方式计算得到的有效地址 |
| 6 | 一般数据存取 | DS | CS、ES、SS | 按寻址方式计算得到的有效地址 |

# 四、 指令的格式与编码

## 1. 指令的格式

| | | | | | |
|---|---|---|---|---|---|
| OP | | | | | |
| OP | Mod 字节 | | | | |
| OP | Mod 字节 | Disp/Data | | | |
| OP | Mod 字节 | Disp/Data(低) | Disp/Data(高) | | |
| OP | Mod 字节 | Disp(低) | Disp(高) | Data | |
| OP | Mod 字节 | Disp(低) | Disp(高) | Data(低) | Data(高) |

**零操作数指令：只有操作码OP而没有操作数。这种指令一般有两种：一是不需要任何操作数。如空操作指令NOP、暂停指令HLT等；二是操作数是隐含的，即指令有默认的操作数。如字符串操作指令MOVSB，该指令表示将源地址中的字节数传送到目标地址中去。**

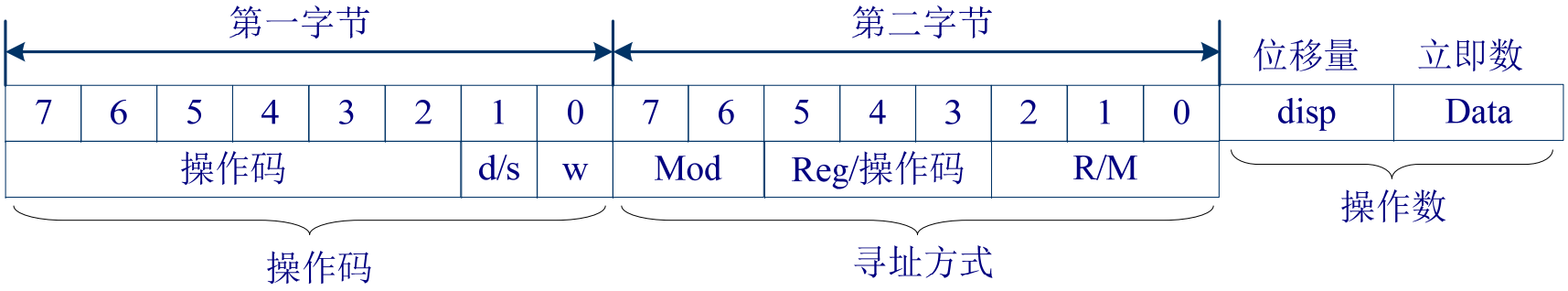
单操作数指令：操作数只有一个。

- 操作数既表示运算数据的地址，也表示存放运算结果的地址。如加1指令：INC AX，该指令执行的结果是将累加器AX中的内容加1然后再送回AX；
- 操作数是源操作数，而目标操作数被隐含了（双操作数指令），运算的结果也默认存放在隐含的操作数地址中。如乘法指令：MUL BL，该指令的执行是将累加器AL的内容与BL内容相乘，结果送AX。

双操作数指令：操作数有两个。

- 最常见的指令格式，微处理器的基本指令系统中多数指令都是双操作数指令。
- 例如：加法指令：ADD AX，BX。指令中的源操作数是BX，表示参加运算的操作数之一是BX的内容，AX为目标操作数，它既表示参加运算的另一数据是AX的内容，同时也表示运算的结果存放在AX中。该指令执行的结果是将AX的内容加上BX的内容，结果送AX中。

# 2. 指令的编码

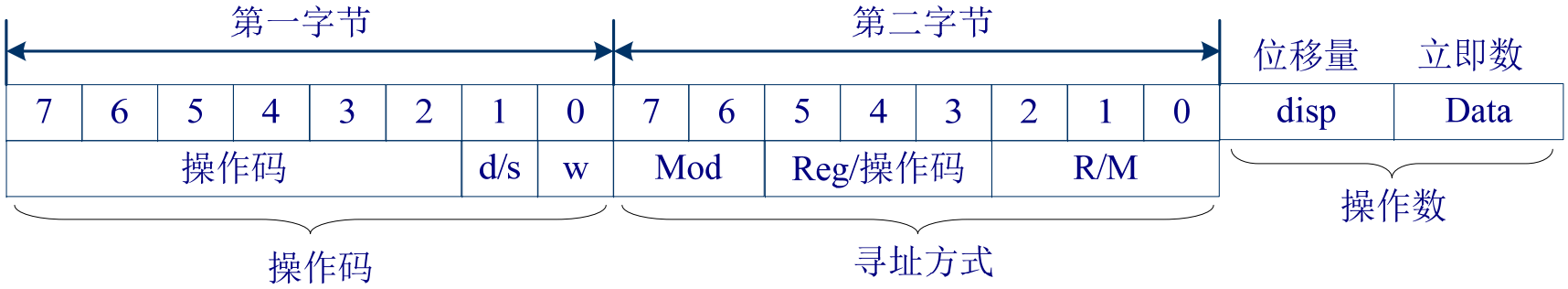



d：进行寄存器寻址的标志，双操作数指令有效。d=0，源操作数为寄存器；d=1，目标操作数为寄存器。

w：字标志操作位。w=0，表示当前指令进行字节操作；w=1，表示当前指令进行字操作。

s：符号扩展位。s=0，对操作数不进行符号扩展；当s=1，w=1时，对8位立即数操作数进行符号扩展，得到16位操作数。

Mod：方式字段。指示操作数是在存储器中还是在寄存器中。




| Mod<br>R/M | 00 | 01 | 10 | 11 | |
|---|---|---|---|---|---|
| | | | | w=0 | w=1 |
| 000 | DS:[BX+SI] | DS:[BX+SI+disp8] | DS:[BX+SI+disp16] | AL | AX |
| 001 | DS:[BX+DI] | DS:[BX+DI+disp8] | DS:[BX+DI+disp16] | CL | CX |
| 010 | SS:[BP+SI] | SS:[BP+SI+disp8] | SS:[BP+SI+disp16] | DL | DX |
| 011 | SS:[BP+DI] | SS:[BP+DI+disp8] | SS:[BP+DI+disp16] | BL | BX |
| 100 | DS:[SI] | DS:[SI+disp8] | DS:[SI+disp16] | AH | SP |
| 101 | DS:[DI] | DS:[DI+disp8] | DS:[DI+disp16] | CH | BP |
| 110 | DS:disp16 | DS:[BP+disp8] | DS:[BP+disp16] | DH | SI |
| 111 | DS:[BX] | DS:[BX+disp8] | DS:[BX+disp16] | BH | DI |

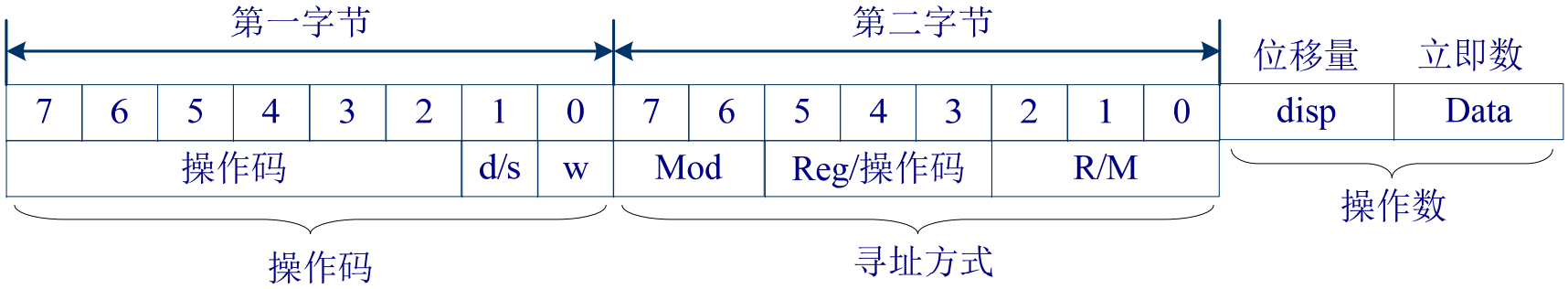
**立即数操作数：** **寄存器操作数：** **存储器操作数：**

**立即寻址** **寄存器寻址** **直接寻址** **寄存器间接寻址**

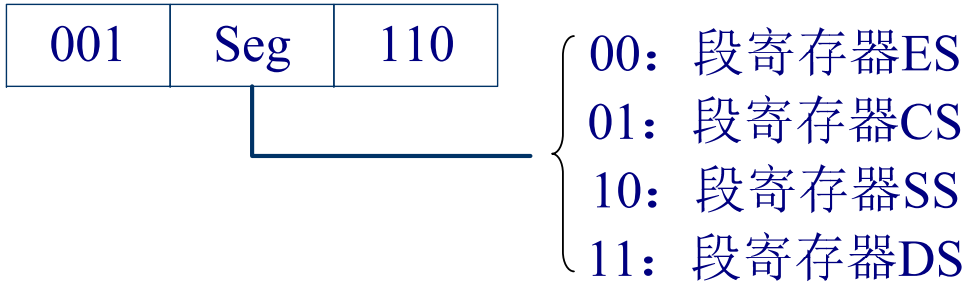
**寄存器相对寻址** **基址变址寻址**

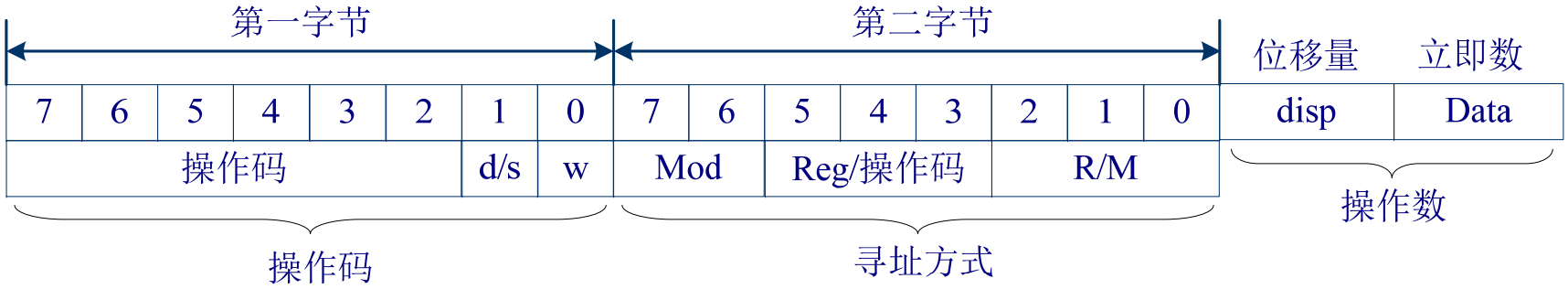
**相对基址变址寻址**

| Mod R/M | 00 | 01 | 10 | 11 | |
|---|---|---|---|---|---|
| | | | | w=0 | w=1 |
| 000 | DS:[BX+SI] | DS:[BX+SI+disp8] | DS:[BX+SI+disp16] | AL | AX |
| 001 | DS:[BX+DI] | DS:[BX+DI+disp8] | DS:[BX+DI+disp16] | CL | CX |
| 010 | SS:[BP+SI] | SS:[BP+SI+disp8] | SS:[BP+SI+disp16] | DL | DX |
| 011 | SS:[BP+DI] | SS:[BP+DI+disp8] | SS:[BP+DI+disp16] | BL | BX |
| 100 | DS:[SI] | DS:[SI+disp8] | DS:[SI+disp16] | AH | SP |
| 101 | DS:[DI] | DS:[DI+disp8] | DS:[DI+disp16] | CH | BP |
| 110 | DS:disp16 | DS:[BP+disp8] | DS:[BP+disp16] | DH | SI |
| 111 | DS:[BX] | DS:[BX+disp8] | DS:[BX+disp16] | BH | DI |

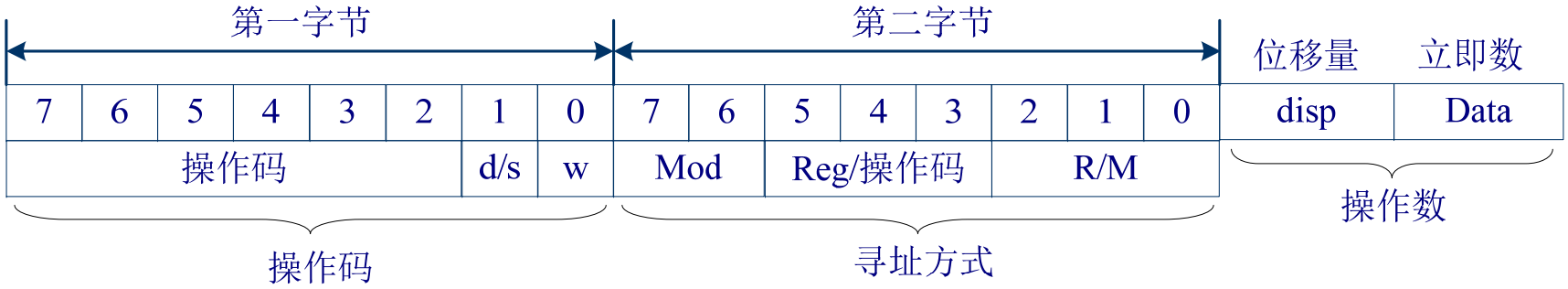



**表中的段寄存器是指无段超越的情况下所使用的隐含的段寄存器。如果指令中指定段超越前缀，则在机器语言中使用放在指令之前的一个字节来表示，其格式为：**

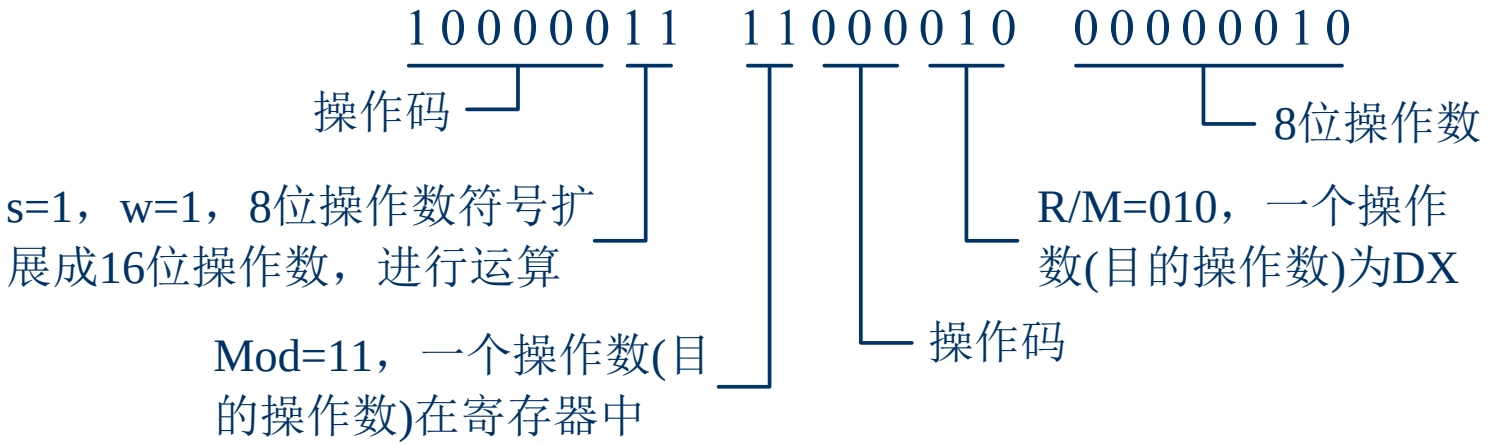


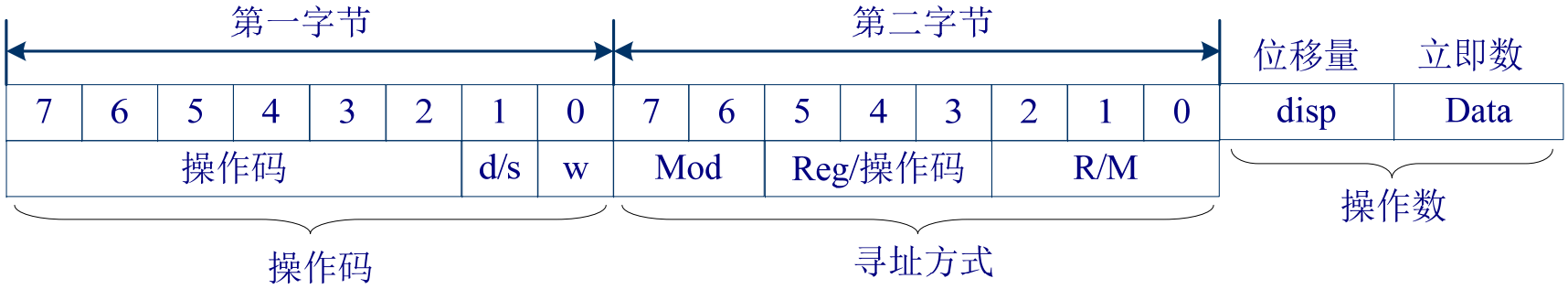



| Reg | w=0（字节操作） | w=1（字操作） |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

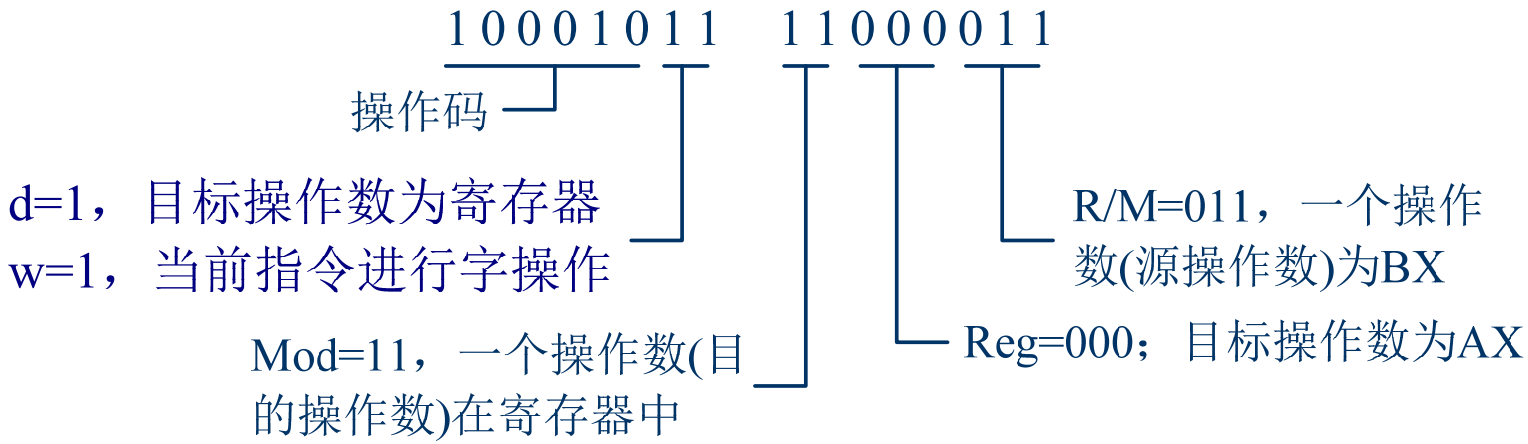

第一字节
第二字节
位移量
立即数
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
disp
Data
操作码
d/s
w
Mod
Reg/操作码
R/M
操作数
操作码
寻址方式

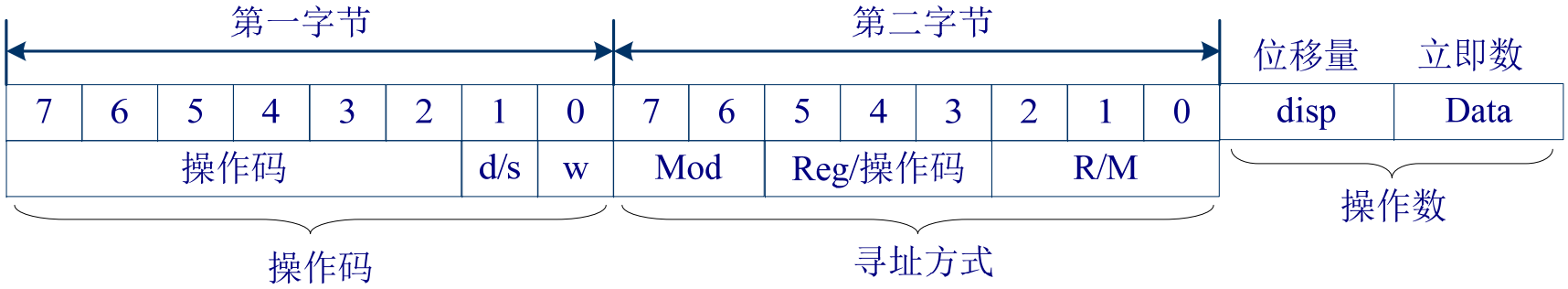
# 例：ADD DX, 02H


10000011 11000010 00000010
操作码
8位操作数
s=1，w=1，8位操作数符号扩展成16位操作数，进行运算
R/M=010，一个操作数(目的操作数)为DX
Mod=11，一个操作数(目的操作数)在寄存器中
操作码


第一字节
第二字节
位移量
立即数
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
disp
Data
操作码
d/s
w
Mod
Reg/操作码
R/M
操作数
操作码
寻址方式

# 例：MOV AX, BX


1 0 0 0 1 0 1 1
1 1 0 0 0 0 1 1
操作码
d=1，目标操作数为寄存器
w=1，当前指令进行字操作
Mod=11，一个操作数(目的操作数)在寄存器中
Reg=000；目标操作数为AX
R/M=011，一个操作数(源操作数)为BX




## 例：MOV ES:[BX+SI+1200H], 3456H

00100110 11000111 10000000 00000000 00010010 01010110 00110100

- 00100110：Seg=00，段超越：ES
- 11000111：w=1，当前指令进行字操作
- 10000000：Mod=11，R/M=000，寻址方式为[BX+SI+disp16]
- 00000000：disp低字节
- 00010010：disp高字节
- 01010110：Data低字节
- 00110100：Data高字节

## 例：HLT

11110100

# 五、 指令介绍

1、数据传送

2、算术运算

3、逻辑运算与移位

4、字符串处理

5、控制转移

6、处理器控制

| 指令类型 | | 助记符 |
|---|---|---|
| 数据传送 | 一般数据传送 | MOV，PUSH，POP，XCHG，XLAT，CBW，CWD |
| | 输入/输出 | IN，OUT |
| | 地址传送 | LEA，LDS，LES |
| | 标志传送 | LAFH，SAFH，PUSHF，POPF |
| 算术运算 | 加法 | ADD，ADC，INC |
| | 减法 | SUB，SBB，DEC，NEG，CMP |
| | 乘法 | MUL，IMUL |
| | 除法 | DIV，IDIV |
| | 十进制调整 | DAA，AAA，DAS，AAS，AAM，AAD |
| 逻辑运算和移位 | | AND，OR，NOT，XOR，TEST，SHL，SAL，SHR，SAR，ROL，ROR，RCL，RCR |
| 串操作 | | MOVS，CMPS，SCAS，LODS，STOS |
| 控制转移 | | JMP，CALL，RET，LOOP，LOOPE，LOOPNE，INT，INTO，IRET，各类条件转移指令 |
| 处理器控制 | | CLC，STC，CMC，CLD，STD，CLI，STI，HLT，WAIT，ESC，LOCK，NOP |

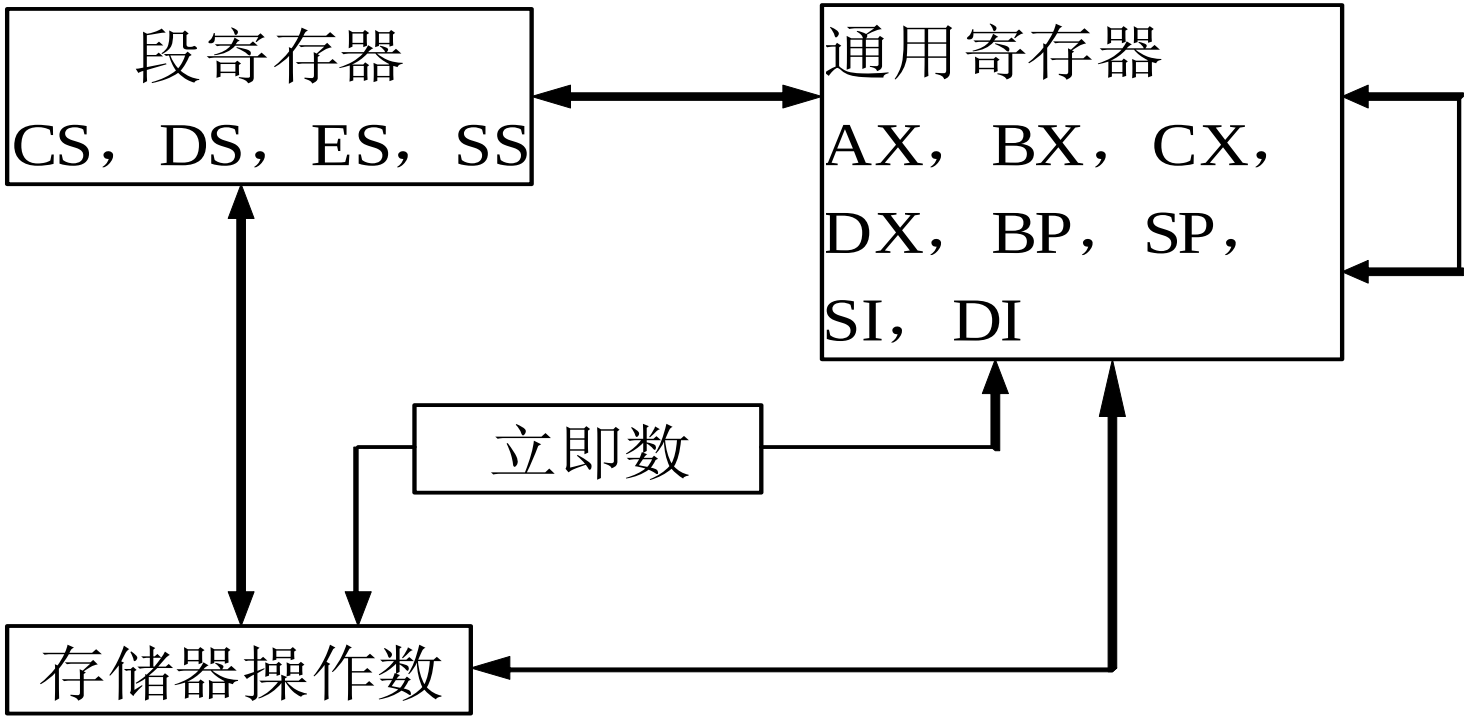
# 1、数据传送指令

通用传送　　MOV PUSH POP XCHG
累加器专用传送　IN OUT　XLAT
地址传送　　LEA LDS　　LES
标志寄存器传送　LAHF SAHF PUSHF POPF

对标志位有影响

## 1）、MOV

（1）格式：MOV　DST（目的操作数），SRC（源操作数）
（2）执行：DST←SRC
（3）操作数类型：立即数，寄存器操作数和存储器操作数



（4）注意：

（A）两个操作数不能全为存储器

（B）两个操作数不能全为段寄存器

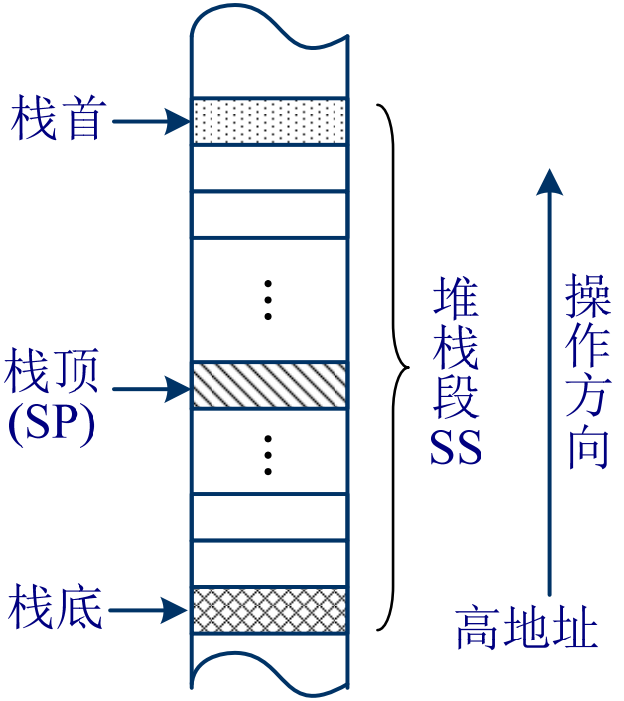
（C）DST不能是立即数

（D）不能将立即数直接传送给段寄存器，段地址必须通过通用寄存器赋值。

（E）源和目的类型必须一致

（5）不影响标志位

## 2）、PUSH和POP




（1）格式： PUSH SRC（字）
POP DST（字）

（2）执行：

PUSH： SP-2→SP
SRC→（SS：SP）

POP： （SS：SP）→DST
SP+2 →SP

（3）操作数类型：16位寄存器操作数或16位存储器操作数
（除立即数、字节寄存器、CS以外的各种寻址方式）

PUSH DS
POP VAL[SI]

讨论：

（1）堆栈是一个存储器区域，存取数据是“先进后出，后进先出”，只有一个出入口。

（2）新的数据进栈时，栈中原存的信息不受破坏，而只改变栈顶位置；数据从堆栈中弹出时，弹出的是栈顶位置的数据，弹出后自动调整栈顶位置。

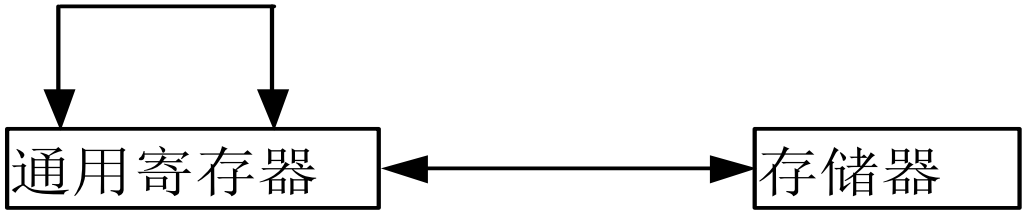
（3）每次传送的数据为16位，16位数据在堆栈中的存放顺序按照“高高低低”原则。

（4）用于保护数据：PUSH和POP指令保护和恢复寄存器和内存内容；CALL和RET或INT和IRET指令自动保护和恢复主程序断点处的段和偏移地址

（5）不影响标志位。

## 3）、XCHG

（1）格式：XCHG OPR1,OPR2（字节/字）
（2）执行：OPR1←→OPR2
（3）操作数类型：




XCHG　AX, [BX+DI]

（4）注意：

不能两个都是存储器；不能有段寄存器

（5）不影响标志位。

## 4）、XLAT（换码或查表指令）

（1）格式：XLAT [OPRD]
（2）执行：（（BX）+（AL））→（AL）
（3）操作数：由隐含的BX和AL寻址的内存字节单元和AL
（4）不影响标志位

```
ASCII  DB  30H,31H,,32H,33H,34H ;ASCII表格
       DB  35H,36H,37H,38H,39H
       MOV AL,5                 ;取“5”的偏移量
       MOV BX,OFFSET ASCII      ;取表首地址
       XLAT                     ;查表转换
```

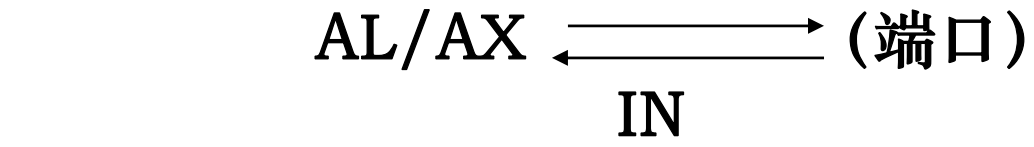
## 5）、IN和OUT

（1）格式： IN AX/AL，端口地址（8位直接/DX间接）
OUT 端口地址（8位直接/DX间接），AX/AL

（2）端口操作数类型：立即数（0-255），DX（0-65535）

```
MOV  DX,210H
IN   AL,DX
OUT  20H,AX
```

（3）执行：




（4）注意： 寄存器只能采用AX/AL，DX

（5）不影响标志位

## 6）、 LEA

（1）格式： LEA　REG，SRC

（2）执行：SRC的EA→REG（取内存的偏移地址）

（3）操作数类型：SRC必须是内存操作数；
REG必须是字通用寄存器。

（4）不影响标志位

```
LEA   SI, VAL
LEA   BX,[100H]          ;BX=100H
LEA   AX,[BX+SI+62H]     ;AX=BX+SI+62H
LEA   BX,ASCII           ;等价于下一条指令
MOV BX,OFFSET ASCII
```

## 7）、LDS和LES

（1）格式： LDS REG，SRC
LES REG，SRC

（2）操作数类型：SRC必须是双字内存操作数；
REG必须是字通用寄存器

（3）执行：
(SRC) →REG， (SRC+2) →DS
(SRC) →REG， (SRC+2) →ES

LDS SI，[10H]

执行前 (DS) =C000H，(C0010H) =0180H，
(C0012H) =2000H

执行后 (SI) = (C000H+10H) =180H，(DS=2000H)

（4）不影响标志位

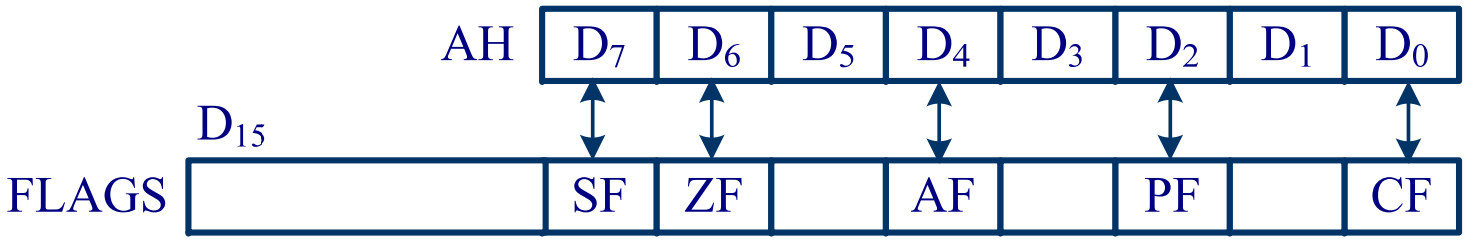
## 8）、LAHF和SAHF（字节）

（1）格式： LAHF
SAHF

（2）执行： 标志寄存器的低字节 $\underset{\text{SAHF}}{\overset{\text{LAHF}}{\rightleftharpoons}}$ AH

（3）操作数：FLAG和AH（隐含）

（4）SAHF影响标志位的SF,ZF,AF,PF,CF



## 9）、PUSHF和POPF（字）

（1）格式： PUSHF
POPF

（2）执行： 标志寄存器 $\xrightarrow{\text{PUSHF}}$ 堆栈 $\xrightarrow{\text{POPF}}$ 标志寄存器

（3）操作数：FLAG和堆栈隐含

（4）POPF影响标志位

## 2、算术运算指令

加法 ADD ADC INC DAA AAA
减法 SUB SUBB DEC NEG CMP DAS AAS
乘法 MUL IMUL AAM
除法 DIV IDIV CBW CWD AAD

上述指令基本上都影响标志位

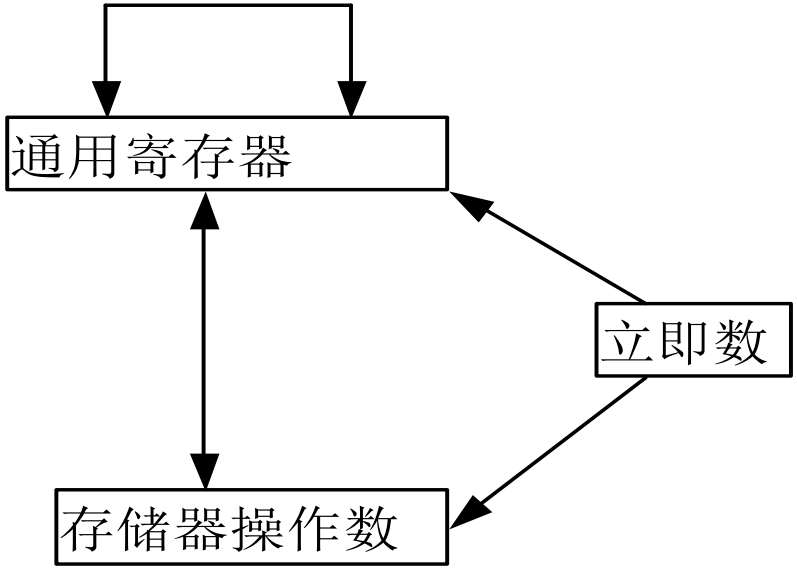
## 1）、ADD和ADC

（1）格式：ADD/ADC　DST，SRC（字节/字）

（2）执行：ADD：DST=DST+SRC

ADC：DST=DST+SRC+CF

（3）操作数类型：




```
ADD  AX,[30H]
ADD  BP,20H
ADD  VAL,DX
ADD  BYTE PTR [SI+10H],5
```

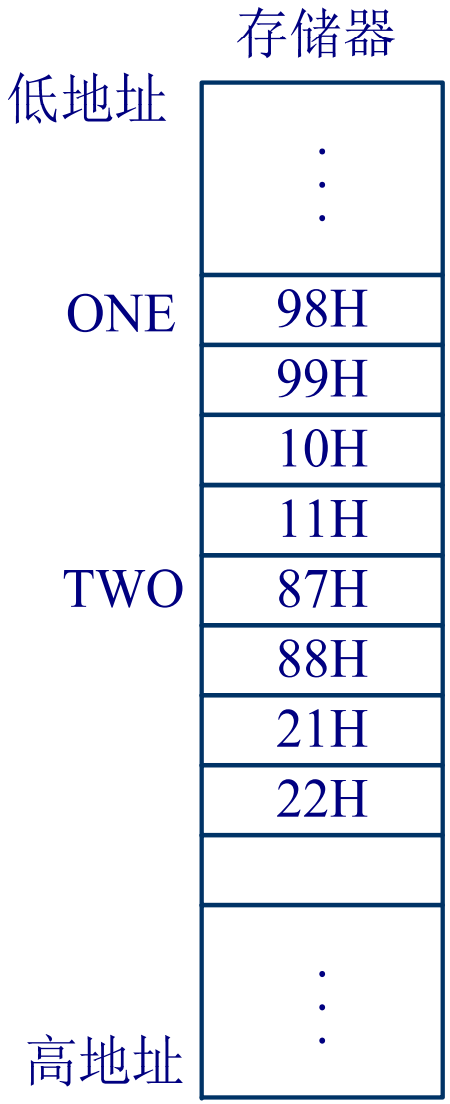
（4）对标志位C，O，P，S，Z和A有影响。

32位加法：

11109998H+22218887H=3332221FH

```
DSEG    SEGMENT 'DATA'
  ONE DW 9998H,1110H
  TWO DW 8887H,2221H
DSEG    ENDS

CSEG    SEGMENT 'CODE'
    MOV     AX, DSEG
    MOV     DS, AX
    MOV AX,[ONE]
    ADD AX,[TWO]
    MOV DX,[ONE+2]
    ADC DX,[TWO+2]
CSEG    ENDS
```



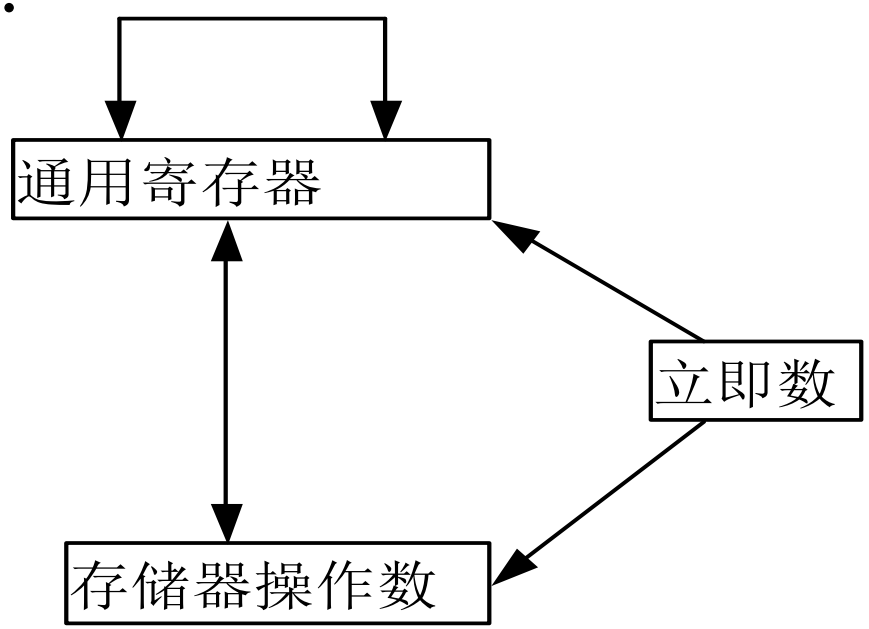
## 2）、SUB和SBB

（1）格式： SUB/SBB DST，SRC

（2）执行： SUB：DST=DST-SRC

SBB： DST=DST-SRC-CF

（3）操作数：



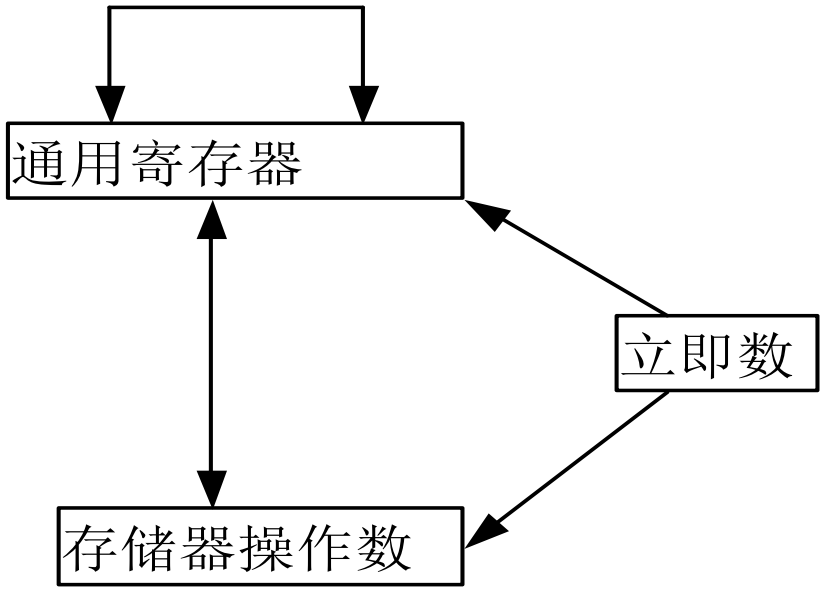

（4）影响标志A,C,O,P,S,Z。

## 3）、CMP

（1）格式： **CMP OPR1，OPR2**

（2）执行： **OPR1-OPR2→** 影响标志位**ACOPSZ**，不改变操作数

（3）操作数：同**SUB**




（4）影响标志**A,C,O,P,S,Z**。

## 4）、INC和DEC

（1）格式：INC/DEC　　OPRD

（2）执行：INC：OPRD+1→OPRD

　　DEC：OPRD-1→OPRD

比ADD/SUB的机器码字节少，执行的时钟周期少。

（3）操作数类型：通用寄存器、存储器

```
INC   SI
INC   VAL
DEC   BX
```

（4）影响标志位A,O,P,S和Z，但不影响C。

（5）主要用于在循环程序中修改地址指针和循环次数计数等。

## 5）、NEG（取负，符号取反）

（1）格式：NEG　OPR

（2）执行：OPR=0-OPR=10000H（100H）-OPR

或OPR= ~OPR+1

加负号

正数 ⟷ 负数

取绝对值

（3）操作数：通用寄存器、存储器

```
MOV  AX,93A4H
NEG  AX          ;AX=6C5CH
NEG  AX          ;AX=93A4H
```

（4）影响　A，C，O，P，S，Z。

只有对-128或-32768求补（结果128或32768）时OF=1。

只有对0求补时CF=0。

## 6）、DAA（加法的BCD码调整）

（1）格式：DAA（紧跟在ADD/ADC/INC后）

（2）执行：（AL）→ （AL）

转成压缩BCD码

表示上一条加法指令是当成十进制运算。

具体调整过程：if ((AL&0FH)>9 )|| (AF=1), then

AL+6→AL; 1→AF

if ((AL&0F0H )>90H)|| (CF=1), then

AL+60H→AL; 1→CF

（3）操作数隐含为AL

```
MOV  AL，28H
ADD  AL，68H    ；  (AL) =90H
DAA             ；  (AL) =96H，C=0，A=1，S=1
```

（4）影响A，C，P，S，Z，对O未定义

## 7）、AAA（加法的ASCII码调整）

（1）格式： AAA

（2）执行： （AL）→（AL） 转成非压缩BCD码
（AH）+CF→（AH）

（3）对标志寄存器的AF=CF有影响，其他无定义。
可用于将一位16进制数转换成非压缩的ASCII码

（4）例1：
```
MOV AX，6
ADD AX，7          ；AX=0DH
AAA                ；AX=0103H
```
例2：
```
MOV AX，A535H
ADD AL，39H        ；AX=A56EH
AAA                ；AX=A604H，AF=CF=1
```

## 8）、DAS（减法的十进制调整）

（1）格式： DAS

（2）执行： （AL）→ （AL）

转成压缩BCD码

（3）操作数隐含为AL

（4）影响A，C，P，S，Z，对O未定义

（5）例：
```
MOV  AL，86H
SUB  AL，7      ；  (AL) =7FH，C=0，A=1
DAS             ；  (AL) =79H，C=0，A=1
```

## 9）、AAS（减法的ASCII码调整）

例：

```
MOV     AX，A535H
SUB     AL，39H     ；AX=A5FCH
AAS                 ；AX=A406H
```

## 10）、MUL（无符号乘）和 IMUL（有符号乘）

（1）格式： MUL/IMUL SRC（字/字节）

（2）执行： （AL）*SRC→(AX)

（AX）*SRC→（DX，AX)

（3）操作数：SRC可以是通用寄存器，存储器操作数（不能立即数）；另一乘数是累加器。

```
MOV  AL,0B4H
MOV  BL,11H
MUL  BL      ;AX=0BF4H,CF=OF=1
```

如果换成 IMUL BL ;AX=FAF4H,CF=OF=1

（4）影响CF，OF，其他未定义

MUL结果的高半！=0，则CF=OF=1，可判断结果的范围

IMUL结果的高半是符号扩展（全0或1），则CF=OF=0。

（5）注意：如果用存储器操作数必须指明类型

例：MUL BYTE PTR[DI]

## 11）、DIV（无符号除）和IDIV（有符号除）

（1）格式： DIV/IDIV　SRC（字/字节）

（2）执行：　（AX）/SRC→（AL，AH）

（DX，AX）/SRC→（AX，DX）

AL/AX为商，AH/DX为余数

IDIV：4个均有符号，余数与被除数符号相同。

（3）操作数： SRC可以是通用寄存器，存储器操作数；被除数缺省为AX和DX。

```
MOV  DX,1234H
MOV  AX,5678H
MOV  BX,3456H
DIV   BX              ;AX=590BH,DX=30C6H
```

（4）标志位未定义

（5）注意：商超过寄存器表示的范围或除以0，则溢出，自动执行0#中断，显示：DIVIDED OVERFLOW

## 12）、CBW和CWD

（1）格式： CBW
CWD

（2）执行： CBW： AL ——→ AX
符号扩展
CWD： AX ——→ （DX，AX）

（3）操作数隐含为AL，AX，DX

```
MOV     AL,4
CBW     ;AX=0004
MOV     AL,84H
CBW     ;AX=FF84H
```

（4）不影响标志

（5）用途：有符号加法的溢出保护；有符号除法之前被除数的符号扩展或溢出保护

## 13）、AAM（乘法的ASCII码调整）

（1）格式：AAM

（2）执行：AL/10→AH（商）和AL（余数）

```
例：    MOV  AL，6
        MOV  AH，7
        MUL  AH              ；AX=02AH
        AAM                  ；AX=0402H
```

（3）可单独用于将16进制数转换成ASCII码（未压缩的BCD码）

```
例：    MOV  AX，3EH
        AAM  ；AX=0602H
```

## 14）、AAD（除法的ASCII码调整）

（1） 格式：AAD（除法指令前）

（2） 执行：AH*10+AL→AL，0→AH，60个时钟周期

AX —未压缩BCD码转换成十六进制数→ AL

（3） 例：MOV AX，0602H
MOV BL，8
AAD ；AX=3EH
IDIV BL ；AX=0607H，62/8=7…6

（4）可单独用于将ASCII码（未压缩的BCD码）转换成16进制数

## 3、逻辑操作指令

| | | |
|---|---|---|
| 逻辑运算 | | AND OR XOR NOT TEST |
| 移位 | 算术移位 | SAL SAR |
| | 逻辑移位 | SHL SHR |
| | 循环移位 | ROL ROR |
| | 带借位的循环移位 | RCL RCR |

## 1）、AND、OR和XOR

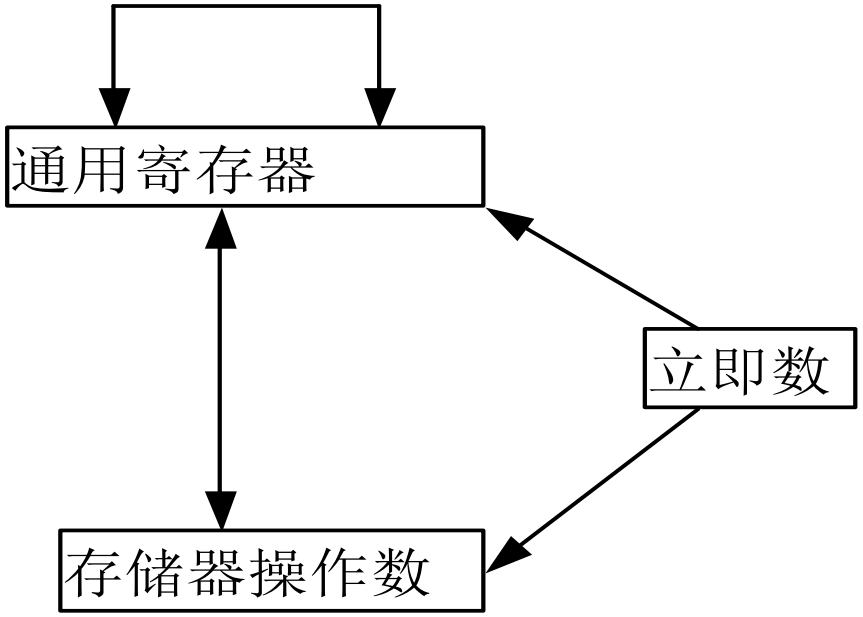
（1）格式： AND/OR/XOR DST，SRC

（2）执行： DST=DST & SRC

DST=DST | SRC

DST=DST ^ SRC

（3）操作数




（4）影响S、Z、P，O=C=0，A未定义。

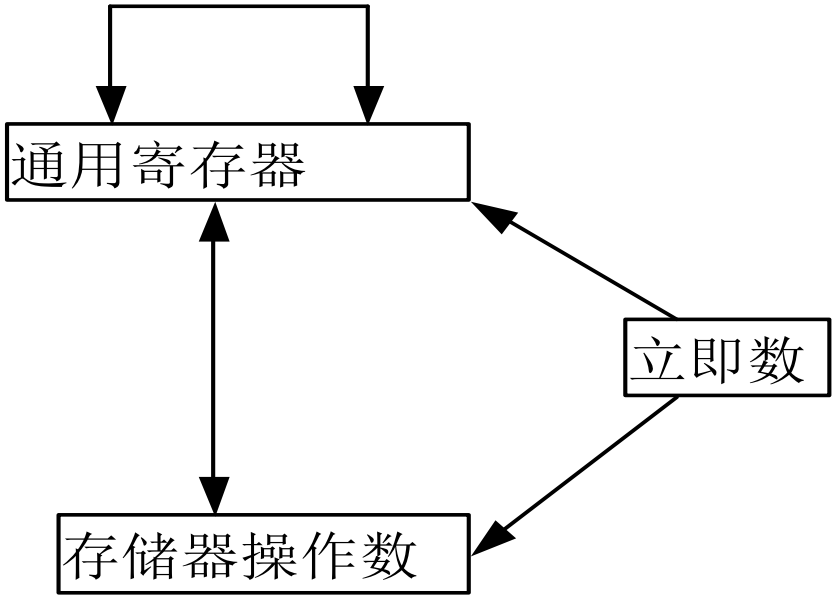
```
例1：AND屏蔽位   AND  AL，11111101B  ；BIT1=0
例2：OR置位      OR   AL，00000010B  ；BIT1=1
例3：XOR位取反   XOR  AL，00000010B  ；BIT1=！BIT1
例4：XOR清0      XOR  AL，AL       ；AL=0
例5：AND，ASCII→二进制
                 AND  AX，0FH
例6：OR，二进制→ASCII
                 OR   AX，30H
例7：XOR比较相等
                 XOR  AX，42EH  ；等效于CMP
                 JZ   MATCH
例8：AND，OR，设置奇校验位，例：奇校验
                 AND  AL，7FH
                 JNP  NEXT
                 OR   AL，  80H
            NEXT:  ……
```

## 2）、TEST

（1）格式： TEST OPR1，OPR2

（2）执行： OPR1 & OPR2，结果仅影响标志

（3）操作数：




```
TEST AL,00000010B
JZ    ZERO1
```

（4）影响S、Z、P，O=C=0，A未定义。

## 3）、NOT

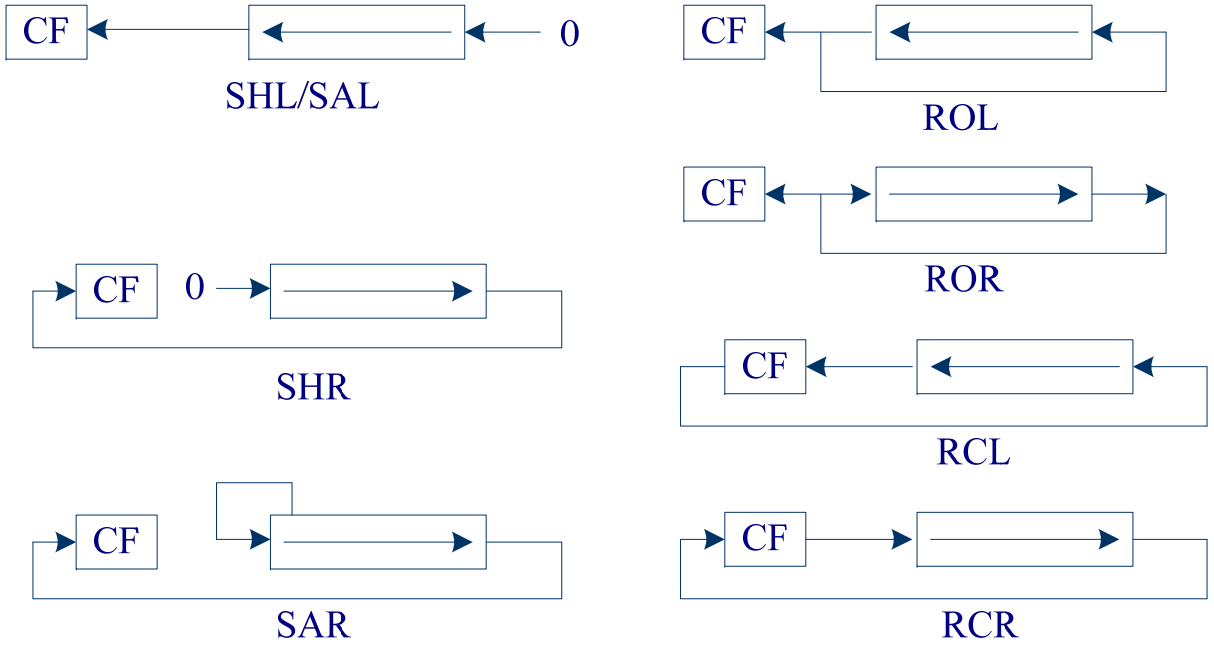
（1）格式： NOT OPR
（2）执行： OPR=～OPR
（3）操作数：通用寄存器，存储器
（4）不影响标志

## 4）、SAL，SAR，SHL，SHR，ROL，ROR，RCL和RCR

（1）格式： 助记符 OPR（字/字节），1/CL

（2）操作数：8/16位通用寄存器，存储器

（3）执行：




（4）前4个影响C、O（最高位变化，O=1）、S、Z、P；
后4个影响C、O（最高位变化，O=1）

## 4、串操作指令

MOVS （MOVSB MOVSW）、 CMPS（CMPSB CMPSW）
LODS（LODSB LODSW）、STOS（ STOSB STOS W）、
SCAS（ SCASB、 SCASW）
前缀 REP REPZ/REPE REPNZ/REPNE

（1） SRC的缺省地址：DS：SI
（2） DST的缺省地址：ES：DI
（3） SI和DI自动增（若DF=0）或减（若DF=1）
1（若字节操作）或2（若字操作）
（4） 可在前面加重复操作前缀

传送或比较前准备工作：
（1）源串首址→DS和SI，或源内容→AL/AX
（2）目的串首址→ES，DI
（3）串长→CX
（4）设置DF（CLD或STD指令）

## 1）、MOVS

（1）格式：

| | | | |
|---|---|---|---|
| | MOVS | DST，SRC | （字/字节） |
| [REP] | MOVSB | （字节） | |
| | MOVSW | （字） | |

（2）执行：

[IF CX！=0]
（ES：DI）=（DS：SI）
DI=DI±2/1
SI=SI±2/1
[CX=CX-1]

（3）不影响标志

例：字节串STR1在数据段，STR2也在数据段，100字节，
传送STR1→ STR2

```
MOV  AX, DS
MOV  ES,  AX
LEA   SI，STR1
LEA   DI，STR2
MOV  CX，100
CLD
REP   MOVSB      ；REP MOVS STR2，STR1
```

## 2）、CMPS

（1）格式：

[REPZ/REPE/REPNZ/REPNE] CMPS SRC，DST
CMPSB
CMPSW

（2）执行：

[IF CX！=0且ZF=1/0]
（DS：SI）-（ES：DI）→标志
DI=DI±2/1
SI=SI±2/1
[CX=CX-1]

第一次循环不检测ZF，所以无须初始化ZF

（3） 影响标志

例：比较数据段的串STR1和串STR2，100字节，相同则BX=FFFFH，否则不同处的偏移地址→BX

```
        MOV  BX，0FFFFH          ；用初始化的方式减少分支
        CLD
        MOV  AX，DS
        MOV  ES，AX
        LEA   SI，STR1
        LEA   DI，STR2
        MOV  CX，100
        REPZ CMPSB
        JZ     SAME          ；或JCX      SAME
ERROR: DEC      SI
        MOV  BX，SI
SAME: …..
```

## 3）、LODS

（1）格式： LODS SRC （字/字节）
LODSB （字节）
LODSW （字）

（2）执行：
AX/AL（字/字节）=（DS：SI）
SI=SI±2/1

（3）不影响标志

## 4）、STOS

（1）格式：

| | | |
|---|---|---|
| | STOS DST | （字/字节） |
| [REP] | STOSB | （字节） |
| | STOSW | （字） |

（2）执行：

[IF CX！=0]
（ES：DI）=AX/AL（字/字节）
DI=DI±2/1
[CX=CX-1]

（3）不影响标志

（4）用于初始化内存区

# 5）、SCAS

（1）格式：

[REPZ/REPE/REPNZ/REPNE] SCAS DST
SCASB
SCASW

（2）执行：

[IF CX！=0且ZF=1/0]
AX/AL-（ES：DI）→标志
DI=DI±2/1
[CX=CX-1]

（3）影响标志

（4）用于从串中查找需要的字符或字

例：从0000：2000H开始的100字节内查找‘A’，偏移地址→DX

```
    CLD
    MOV     DI，0
    MOV     ES，DI   ;赋附加段基址
    MOV     DI，2000H  ;DI=偏移地址
    MOV     CX，100   ;CX=字符串长度
    MOV     AL，‘A’    ;
    REPNZ   SCASB    ;CX≠0且ZF＝0重复
    JNZ     NOFOUND    ;
    MOV     DX，DI     ;
    DEC     DX    ;
NOFOUND：….
```

## 5、控制转移指令

| | |
|---|---|
| 无条件跳转 | JMP |
| 条件跳转 | JZ/JE JNZ/JNE JS JNS JO JNO<br>JP/JPE JNP/JPO |
| (无符号数比较) | JC/JB/JNAE JNC/JNB/JAE<br>JBE/JNA JNBE/JA |
| (有符号数比较) | JL/JNGE JNL/JGE JLE/JNG<br>JNLE/JG JCXZ |
| 循环 | LOOP |
| 过程调用和返回 | CALL RET |
| 软件中断和返回 | INT IRET |

## 1）、JMP

（1）格式：

JMP 短标号/近标号/远标号/字寄存器/字存储器/双字存储器

（2）不影响标志

例：JMP SHORT LABEL ;段内直接转移

JMP 2100H; 段内直接转移，IP←IP+1200H

JMP BX ； 段内间接转移 ，IP←BX

JMP [BX]； 段内间接转移 ，IP← (BX)

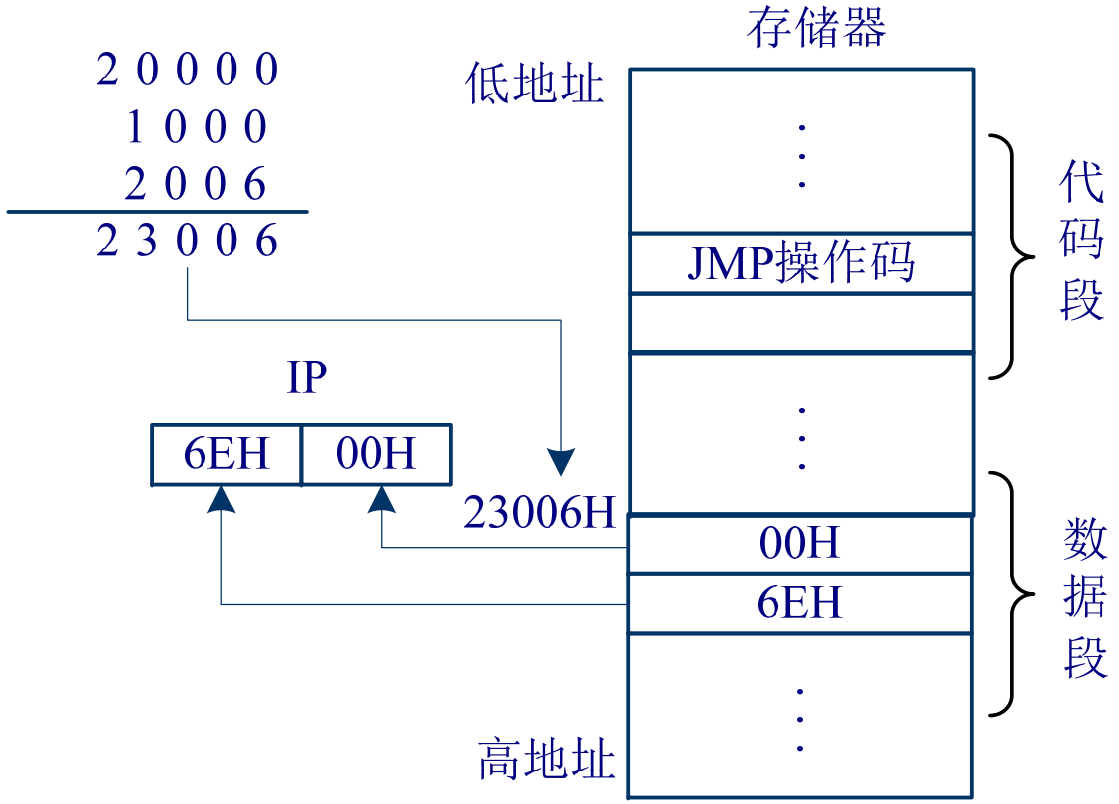
JMP FAR PTR LABEL ;段间间接转移

JMP DWORD PTR [BX] ;段间间接转移

JMP WORD PTR [BX+DI]

；(DS)=2000H，(BX)=1000H，(DI)=2006H，

；(23006H)=6E00H，则指令执行后，(IP)=6E00H，



JMP DWORD PTR[BX]

; (DS)=3000H

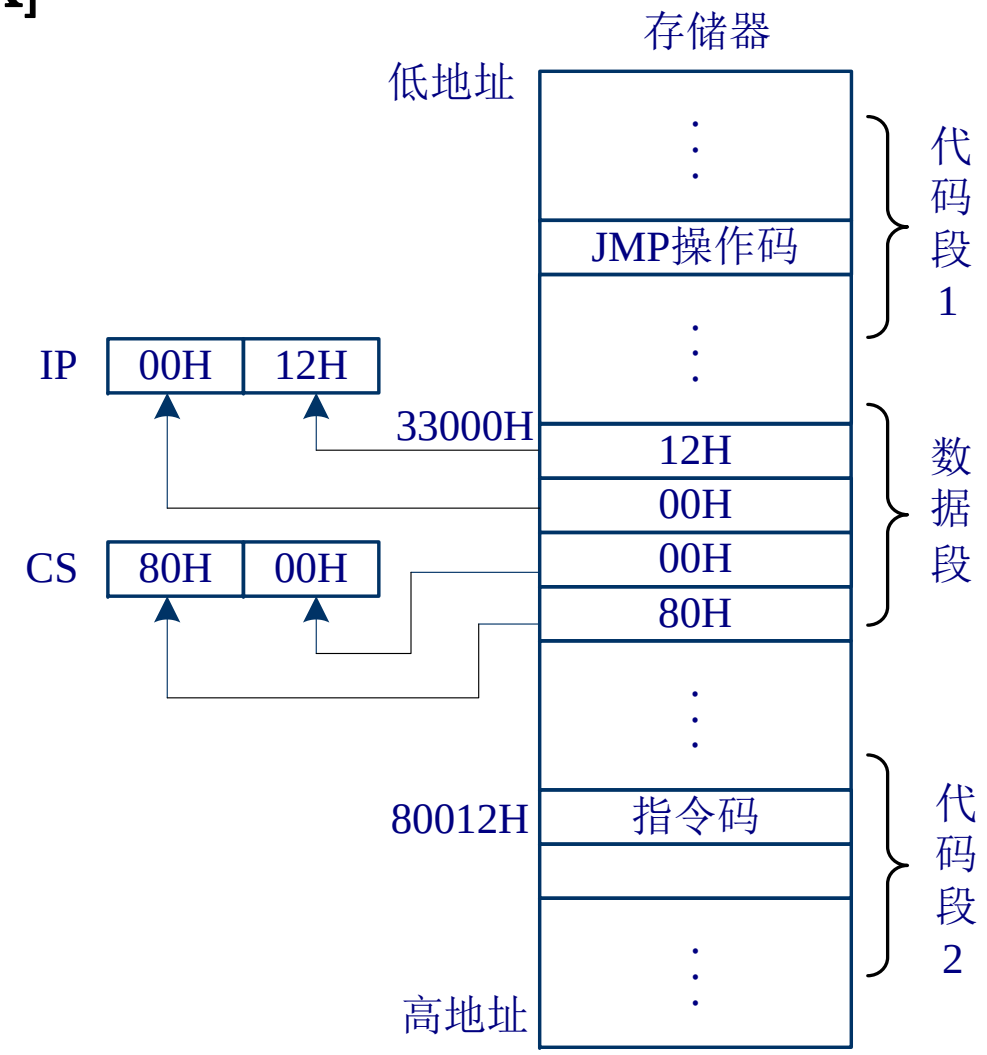
; (BX)=3000H

; (33000H)=12H;

; (33001H)=00H

; (33002H)=00H

; (33003H)=80H



## 2）、 条件转移

（1）格式： 助记符 [SHORT] 短标号（8位相对距离）

（2）

| 助记符 | 测试条件（标志位） | 用于数的比较 | |
|---|---|---|---|
| JZ/JE | Z=1 | A=B | |
| JNZ/JNE | Z=0 | A！=B | |
| JS | S=1 | | |
| JNS | S=0 | | |
| JO | O=1 | | |
| JNO | O=0 | | |
| JP/JPE | P=1 | | |
| JNP/JPO | P=0 | | |
| JC/JB/JNAE | C=1 | A<B | 无符号数 |
| JNC/JNB/JAE | C=0 | A≥B | |
| JBE/JNA | C\|Z=1 | A≤B | |
| JNBE/JA | C\|Z=0 | A>B | |
| JL/JNGE | S⊕O=1 | A<B | 有符号数 |
| JNL/JGE | S⊕O=0 | A≥B | |
| JLE/JNG | （S⊕O）\| Z=1 | A≤B | |
| JNLE/JG | （S⊕O）\| Z=0 | A>B | |
| JCXZ | CX=0 | | |

A：ABOVE B：BELOW
G：GREAT L：LESS

E：EQUAL NE：NOT EQUAL

# 无符号数相减的判断

| | ZF | CF |
|---|---|---|
| A>B | 0 | 0 |
| A=B | 1 | 0 |
| A<B | 0 | 1 |

| 大小关系 | 判断条件 |
|---|---|
| A=B | ZF=1 |
| A<B | CF=1 |
| A>B | ZF\|CF=0 |
| A≤B | ZF\|CF=1（ZF⊕CF=1） |
| A≥B | CF=0 |

# 有符号数相减的判断

| A | B | 大小关系 | SF | ZF | OF |
|---|---|---|---|---|---|
| A>0 | B≥0 | A>B | 0 | 0 | 0 |
| A≥0 | B>0 | A<B | 1 | 0 | 0 |
| A≥0 | B<0 | A>B | 0 | 0 | 0（无溢出） |
| | | | 1 | 0 | 1（溢出） |
| A<0 | B≥0 | A<B | 1 | 0 | 0（无溢出） |
| | | | 0 | 0 | 1（溢出） |
| A<0 | B≤0 | A<B | 1 | 0 | 0 |
| A≤0 | B<0 | A>B | 0 | 0 | 0 |
| | | A=B | 0 | 1 | 0 |

| 大小关系 | 判断条件 |
|---|---|
| A=B | ZF=1 |
| A<B | SF⊕OF=1 |
| A>B | (SF⊕OF)\|ZF=0 |
| A≤B | (SF⊕OF)\|ZF=1 |
| A≥B | SF⊕OF=0 |

例： 比较2个双字有符号数(DX，AX)和(BX，CX)大小，大于跳到标号X，小于等于跳到标号Y

```
        CMP   DX，BX
        JG    X
        JL    Y
        CMP   AX，CX
        JA    X
Y:      ….                          ；A≤B
        JMP   EXIT
X:      …..                         ；A>B
EXIT:   ……
```

## 3）、LOOP，LOOPZ/LOOPE，LOOPNZ/LOOPNE

（1）格式： 助记符 段内短标号

（2）执行：

CX=CX-1

IF CX！=0 [&& ZF=1/0]，重复

如果初始化CX=0，循环65536次;至少循环一次。

（3）CX的减不影响标志

**emu8086：例3.92**

## 4）、CALL

（1）格式：

CALL 段内子过程名/段间子过程名/字寄存器/字/双字存储器

（2）执行：

[SP=SP-2， CS入栈]

SP=SP-2， IP入栈

[CS=子过程的段地址/双字内存的高字内容]

IP=子过程的偏移地址/字寄存器内容/字内存内容/双字内存的低字内容

（3）不影响标志

## 5）、RET

（1）格式：

RET （段内/段间直接返回）

RET 常数表达式 （段内/段间带立即数返回）

（2）执行：

IP出栈，SP=SP+2

[CS出栈，SP=SP+2]

[SP=SP+常数]

（3） 带参数返回用于堆栈传递参数的情况

## 6）、INT

（1）格式： INT TYPE（0~255）

（2）执行：

SP=SP-2，FLAG进栈
SP=SP-2 ，CS进栈
SP=SP-2 ，IP进栈
IP=（TYPE*4）
CS=（TYPE*4+2）
类似段间间接调用

（3） 主要用于调用BIOS，DOS功能调用

（4）中断概念：

计算机在执行正常程序的过程中，由于某些事件发生，需要暂时中止当前程序的运行，转到中断服务程序去为临时发生的事件服务，中断服务程序执行完毕后，又返回正常程序继续运行，该过程称中断。

引起中断的原因（称中断源）有：

- 外部中断（硬件中断），从8086的NMI（不可屏蔽中断引脚）或INTR（可屏蔽中断引脚）引入。
- 内部中断（软件中断）：CPU运算过程中发生的意外情况或由INT指令产生。

响应中断的过程：

- 将CS、IP和FLAG送到堆栈保护；
- 根据中断接口电路送入的或INT指令中指定的中断类型号，查中断向量表，找到中断服务程序入口地址，转到相应的中断服务程序；
- 服务程序结束后通过执行IRET，从堆栈中恢复原标志和断点，返回正常程序继续执行。

中断向量表：

- 内存00000—003FFH的1K大小的空间
- 每个入口地址4字节，一共可以放256个中断向量
- 各中断向量在表中排序为中断类型号

专用中断：

除法错中断（0）

单步中断（1）

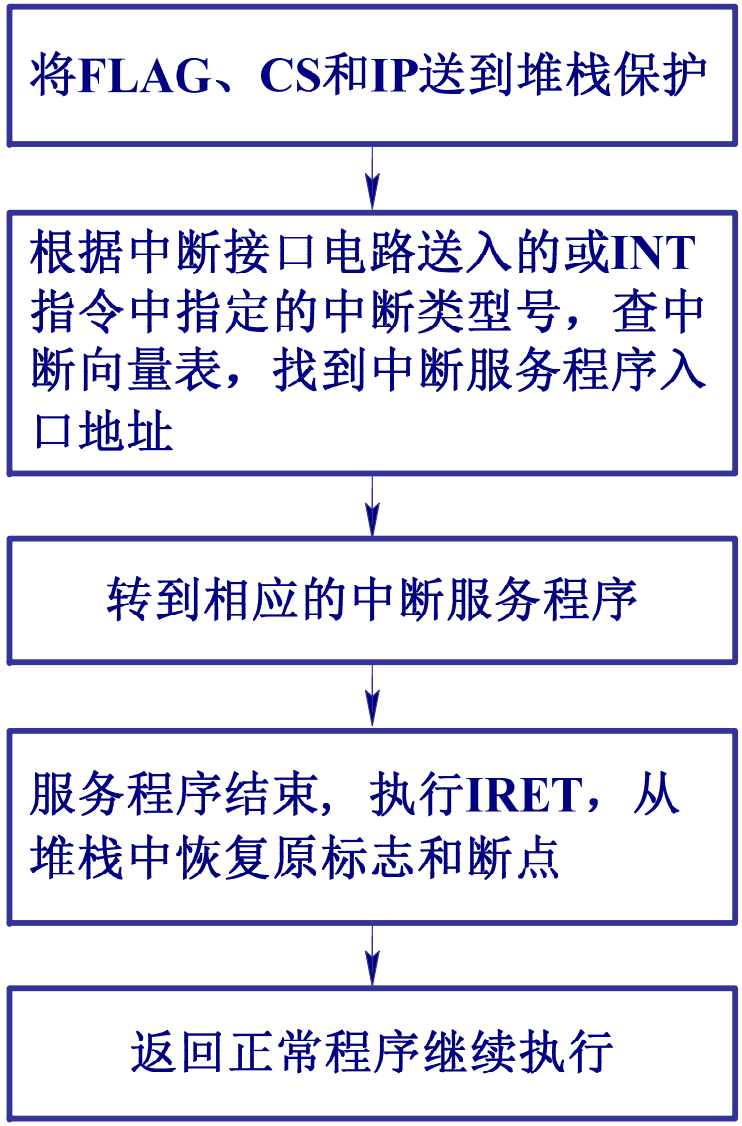
不可屏蔽中断（2）

断点中断（3）

溢出中断（4）

| |
|---|
| 0# 中断程序入口偏移地址的低字节 |
| 0# 中断 程序入口偏移地址的高字节 |
| 0# 中断程序入口段地址的低字节 |
| 0# 中断 程序入口段地址的高字节 |
| . . . |
| 255# 中断 程序入口偏移地址的低字节 |
| 255#中断 程序入口偏移地址的高字节 |
| 255# 中断 程序入口段地址的低字节 |
| 255# 中断程序入口段地址的高字节 |

# 中断响应过程


将FLAG、CS和IP送到堆栈保护
根据中断接口电路送入的或INT指令中指定的中断类型号，查中断向量表，找到中断服务程序入口地址
转到相应的中断服务程序
服务程序结束，执行IRET，从堆栈中恢复原标志和断点
返回正常程序继续执行

## 7）、IRET

（1）格式：

IRET

（2）执行：

SP=SP+2，IP出栈
SP=SP+2，CS出栈
SP=SP+2，FLAG出栈

（3）影响标志

# 6、处理器控制指令

标志位操作　CLC　STC　CMC CLD　STD　CLI　STI
空操作　NOP
外同步操作　HLT　ESC　WAIT　LOCK

| | |
|---|---|
| CLC | CF=0 |
| STC | CF=1 |
| CMC | CF=~CF |
| CLD | DF=0 |
| STD | DF=1 |
| CLI | IF=0 |
| STI | IF=1 |

## 7、 指令周期

8086指令的执行速率是由晶振控制产生的时钟决定的，每条指令执行都需要几个时钟周期。
8086CPU的工作频率5MHZ，一个时钟周期
T=1/5MHZ=0.2μs

指令执行的时钟周期举例：
MOV 寄存器，寄存器 2T
MOV 寄存器，立即数 4T
MOV 内存，寄存器 9T+EA
ADD 内存，立即数 17T+EA
MUL 16位寄存器 118T-133T
SHL 寄存器，CL 8T+4T/位

其中EA是计算EA的时间：
直接寻址：6T
寄存器间接寻址：5T
寄存器相对寻址：9T
基址变址寻址：7T-8T
相对基址变址寻址：11T-12T

# 5.6 8086CPU的存储器扩展

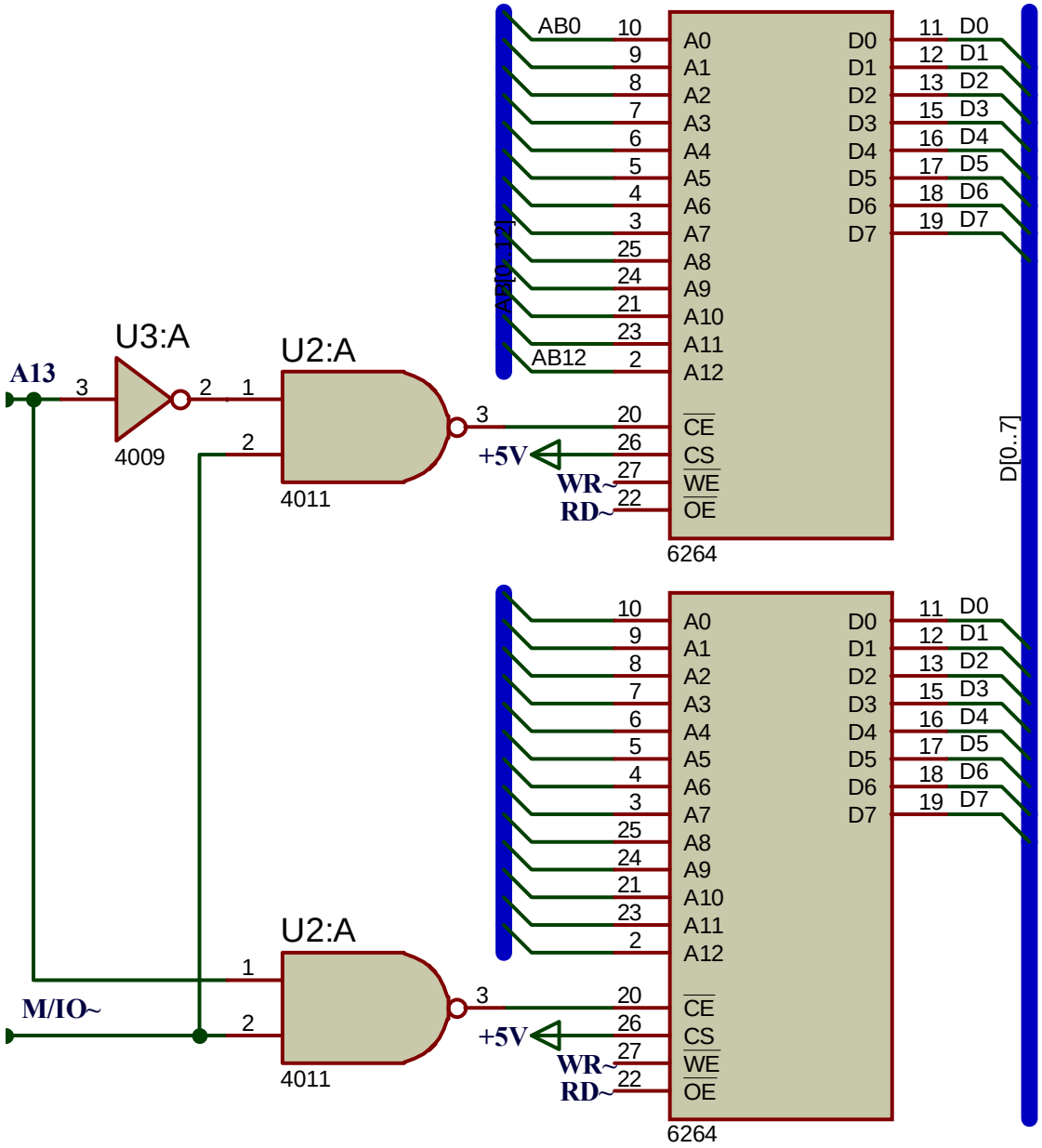
- 连接部分主要由三个部分组成:

  1.地址线 2.数据线 3.控制线

- 连接中需要考虑的问题:
  - CPU总线的负载能力
  - CPU的时序和存储器存取速度之间的配合
  - 存储器的地址分配和片选
  - 控制信号的连接

# 1 线性选择方式

- 只有存储芯片的片选信号CS有效，才能对该芯片进行操作
- 连接方式（书例5.1）：
  - 将CPU地址总线低13位与存储芯片地址线相连
  - CS端与某一位高位地址线（A13）相连
  - 1#芯片地址：0000～1FFFH、4000～5FFFH、……
  - 2#芯片地址：2000～3FFFH、6000～7FFFH、……

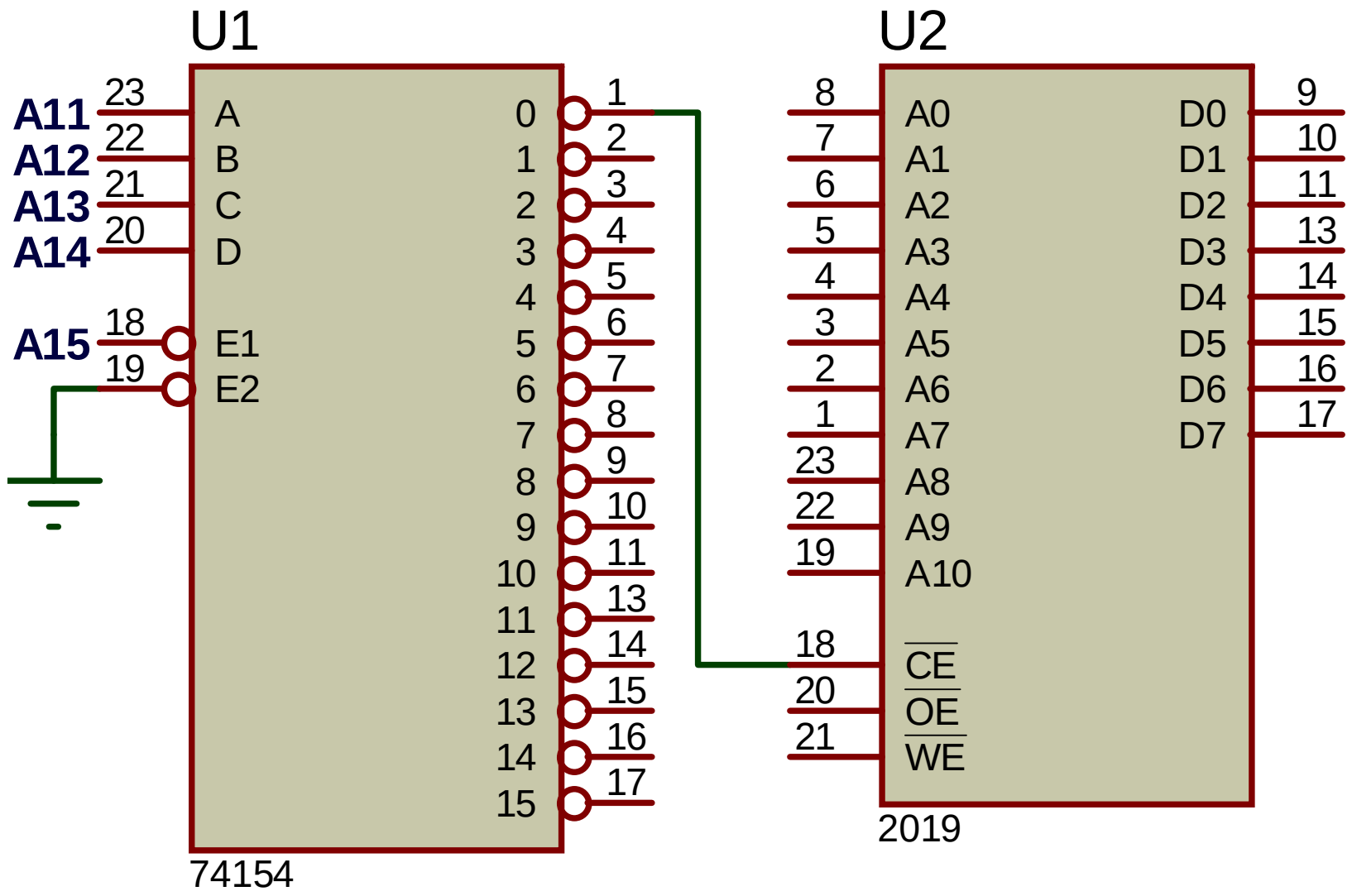

AB0
AB[0..12]
AB12
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
D0
D1
D2
D3
D4
D5
D6
D7
D[0..7]
CE
CS
WE
OE
6264
A13
U3:A
4009
U2:A
4011
+5V
WR~
RD~
M/IO~

## 2 全译码

- 所有的系统地址线均参与对存储单元的译码寻址
- 包括低位地址线对芯片内各存储单元的译码寻址（片内译码），高位地址线对存储芯片的译码寻址（片选译码）
- 采用全译码，每个存储单元的地址都是唯一的，不存在地址重复
- 译码电路可能比较复杂、连线也较多

# 译码和译码器

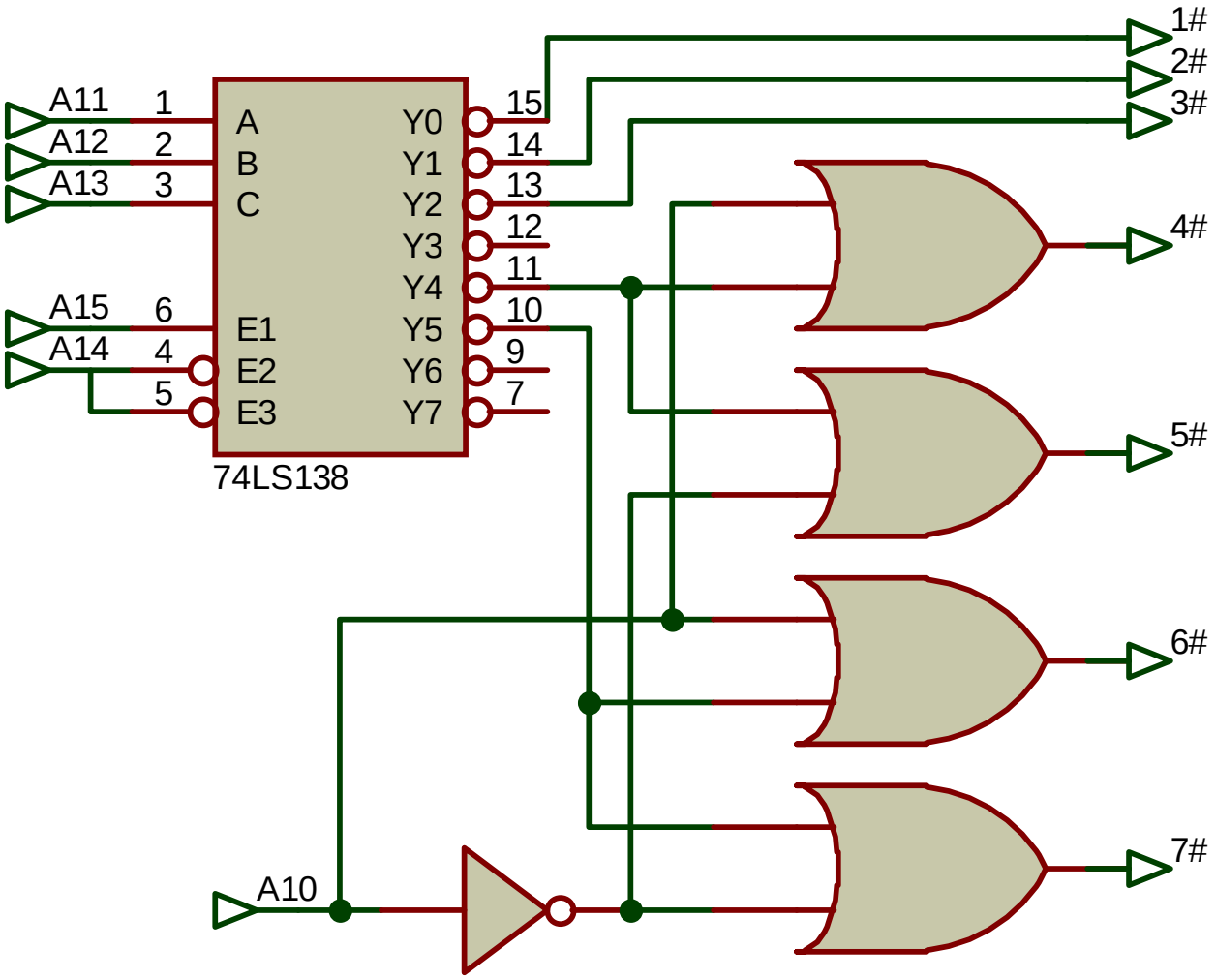
- 译码：将某个特定的“编码输入”翻译为唯一“有效输出”的过程
- 译码电路可以使用门电路组合逻辑
- 译码电路更多的是采用集成译码器
- 常用的**2-4**译码器**74LS139**
- 常用的**3-8**译码器**74LS138**
- 常用的**4-16**译码器**74LS154**


U1
A11
A12
A13
A14
A15
23
22
21
20
18
19
A
B
C
D
E1
E2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
1
2
3
4
5
6
7
8
9
10
11
13
14
15
16
17
74154
U2
8
7
6
5
4
3
2
1
23
22
19
18
20
21
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
CE
OE
WE
D0
D1
D2
D3
D4
D5
D6
D7
9
10
11
13
14
15
16
17
2019

# 部分译码

- 只有部分（高位）地址线参与对存储芯片的译码
- 每个存储单元将对应多个地址（地址重复），需要选取一个可用地址
- 可简化译码电路的设计，但系统的部分地址空间将被浪费

# 例：1#~7#接存储器，试确定寻址范围。




8000H~87FFH

8800H~8FFFH

9000H~97FFH

A000H~A3FFH

A400H~A7FFH

A800H~ABFFH

AC00H~AFFFH

# 片选端译码小结

- 存储芯片的片选控制端可以被看作高位地址线单独选用或经译码而得
- 在系统中，与地址相关的有：地址空间的选择（接系统的**M/IO**～信号）和高位地址的译码选择（与系统的高位地址线相关联）
- 对一些存储芯片通过片选无效可关闭内部的输出驱动机制，起到降低功耗的作用

# 存储芯片数据线的处理

- 若芯片的数据线正好16根：
  - 一次可从芯片中访问到16位数据
  - 全部数据线与系统的16位数据总线相连
- 若芯片的数据线不足16根（例5.4）：
  - 一次不能从一个芯片中访问到16位数据
  - 利用多个芯片扩充数据位
  - 这种扩充方式简称为位扩充

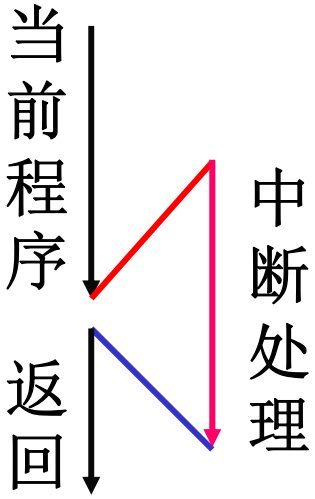
# 存储芯片的读写控制

- 芯片~OE与系统的读命令线相连
  - 当芯片被选中、且读命令有效时，存储芯片将开放并驱动数据到总线
- 芯片~WE与系统的写命令线相连
  - 当芯片被选中、且写命令有效时，允许总线数据写入存储芯片

# 5.7 8086CPU的中断系统

## 一、中断的概念与分类

- 中断的概念
  - CUP在正常执行程序的过程中，由于某种原因，使CPU暂停当前程序的执行，
    转去处理临时发生的事件，
    处理完毕再返回继续执行暂停的程序。
    ——该过程称中断

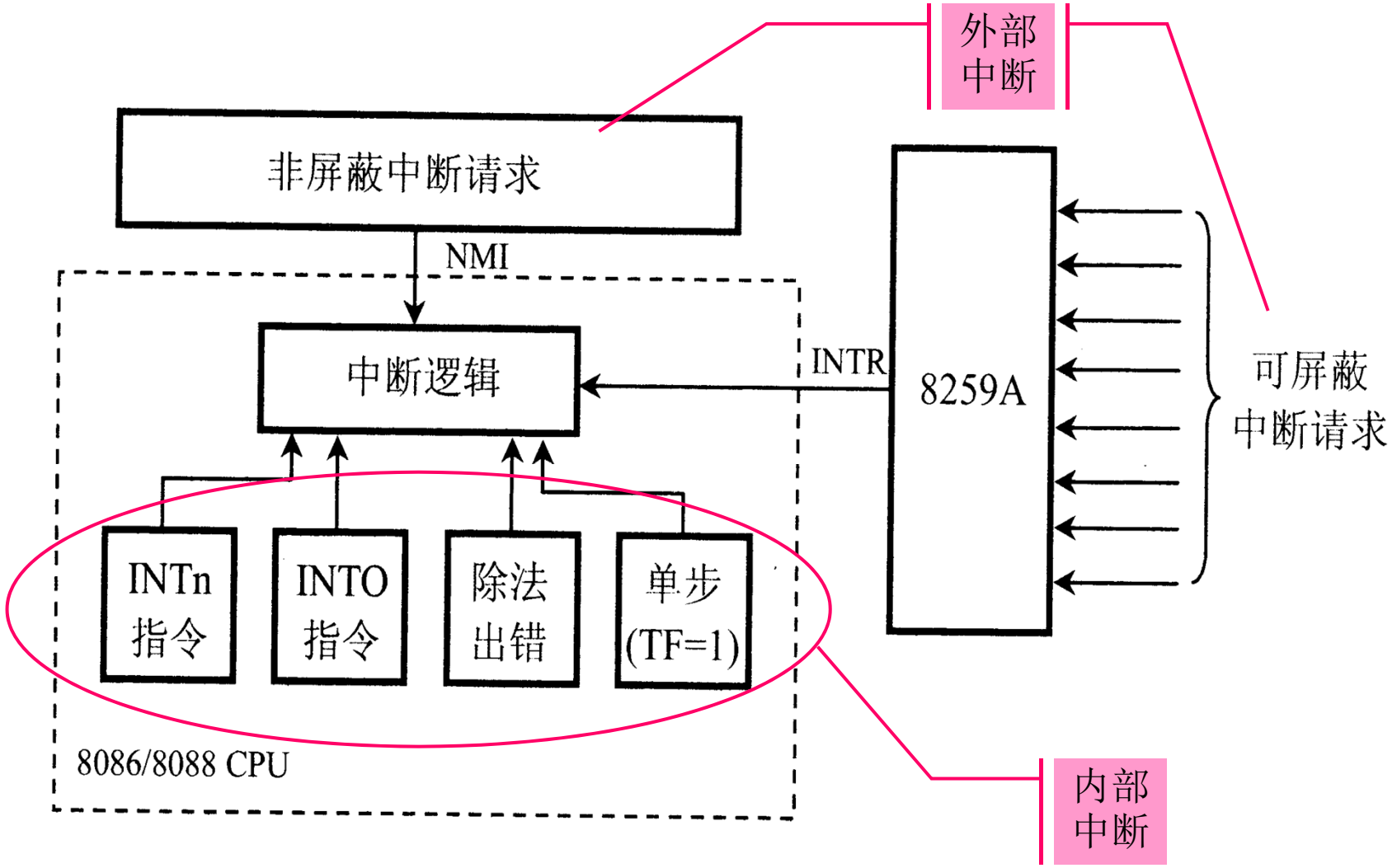


- 中断源
  - 引起程序中断的事件
    - 外部中断
    - 内部中断
- 中断响应
  - **CPU**在每条指令的最后一个周期检测中断信号引脚，当条件满足时，**CPU**响应中断，向外设发中断响应信号，并保护断点，转向中断服务程序

- 中断向量表
  - 中断服务程序的入口地址存放处
- 中断优先级
  - 为每个中断源分配一个优先级，**CPU**总是优先响应优先级高的中断
- 中断屏蔽
  - 通过软件设置，使**CPU**不能响应中断源的申请

# 中断的分类

- 中断类型号
  - **8086/8088**可以处理**256**种中断，每一种中断都规定一个唯一的中断类型号**N**，即中断向量
- **256**种中断分为两类
  - 外部中断——由外部硬件的请求产生的中断，又称硬件中断
  - 内部中断——是由指令的执行所引起的中断，又称软件中断


外部
中断
非屏蔽中断请求
NMI
中断逻辑
INTR
8259A
可屏蔽
中断请求
INTn
指令
INTO
指令
除法
出错
单步
(TF=1)
8086/8088 CPU
内部
中断

# 外部中断

- 非屏蔽中断请求
  - 由引脚**NMI**引入，边沿触发，上升沿之后维持两个时钟周期高电平有效，中断类型号**N＝2**
  - 不受中断标志位**IF**影响
  - 引起原因
    - **RAM**奇偶校验错误
    - **I/O**通道扩展板奇偶校验错误
    - 协处理器**8087**中断请求

# 外部中断（续）

- 可屏蔽中断请求
  - 引脚**INTR**引入，电平触发，高电平有效
  - 中断标志位**IF＝1**时允许中断；**IF＝0**时禁止中断
    - 可用**STI**指令置位**IF**状态（开中断），**CLI**指令复位（关中断）
  - 引起原因
    - 外部设备的中断请求

# 内部中断

- INT n 指令中断
  - CPU执行INT n 指令后，产生中断类型号N＝n 的中断
  - 中断向量表地址＝4×n
  - 例如：INT 21H，产生中断类型号为21H的中断，并从中断向量表的4*21H（即0:84H）单元取出中断服务程序的入口地址，转去执行

# 内部中断（续）

- 除法错中断
  - 除数为0或商超出寄存器范围。中断类型号N=0
- 溢出中断指令INTO
  - 在算术运算指令之后紧跟INTO指令，可检查溢出标志OF 。中断类型号N＝4
  - 例如：测试加法的溢出

```
MOV  AX,0009H
ADD AX,0080H
INTO
:
```

❖ 无溢出,不中断,顺序

```
MOV AX,9000H
ADD AX,8000H
INTO
:
```

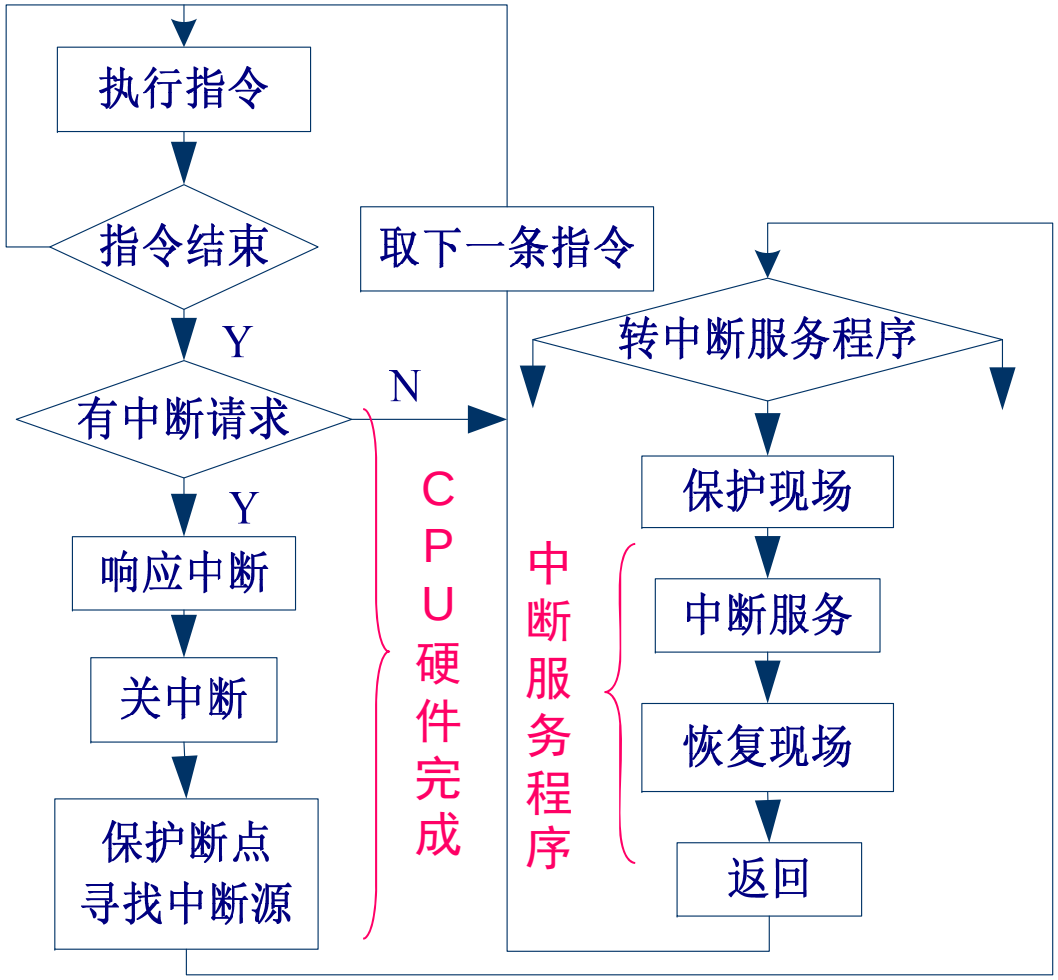
❖ 溢出,中断,转移

# 内部中断（续）

- 单步中断
  - 当标志位**TF＝1**时，每执行一条指令，**CPU**便产生中断类型号**N＝1**的单步中断。单步中断用于 **Debug**调试程序
- 断点中断
  - 当程序设置了断点时，**CPU**执行到断点处便产生中断类型号**N＝3**的断点中断，并显示寄存器及单元内容，供**Debug**调试程序使用

# 二、中断的处理过程

可屏蔽中断处理流程




**CPU响应可屏蔽中断的条件：**

**1** 外设提出中断申请

**2** 本中断位未被屏蔽

**3** 中断允许

- 可屏蔽中断请求INTR的响应

1. CPU响应可屏蔽中断
2. CPU转入中断服务过程

## 1. CPU响应可屏蔽中断

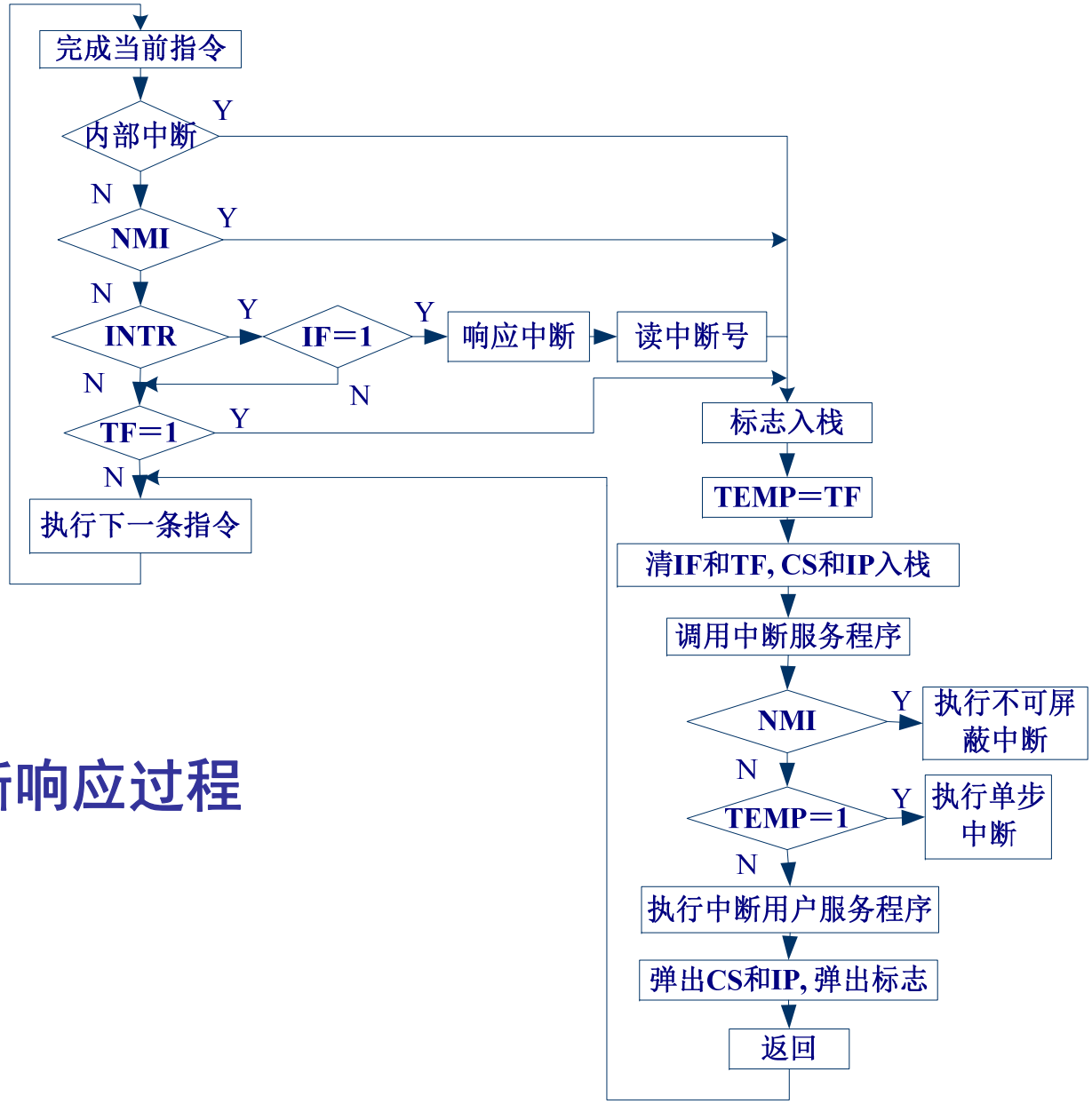
- 当中断屏蔽触发器未被屏蔽时，外设发出中断请求信号
- CPU在每条指令的最后一个机器周期的最后一个T状态采样中断请求INTR引脚，若有中断请求信号且CPU内部中断允许触发器是开放的（IF＝1），则CPU响应中断
- CPU向外设接口发两个中断响应信号$\overline{\text{INTA}}$
- 外设收到第二个$\overline{\text{INTA}}$，往数据线送中断类型号

## 2. CPU转入中断服务过程

- 从数据总线读取由外设输入的中断类型号
- 标志寄存器PSW的值入栈
- PSW中的中断允许标志IF和单步标志TF清0
- 保护断点，下一条指令的段地址CS和指令指针IP的值入栈
- 取中断向量表的中断入口地址，转入中断服务子程序
- 中断处理程序结束后，从堆栈依次弹出IP、CS和PSW，返回主程序断点处继续执行

- 非屏蔽中断请求NMI
  - CPU检测有NMI，不必判断IF标志，内部自动产生中断类型号N＝2，并转入相应中断服务过程
- 软件（内部）中断INT n
  - 由软件设定，不受IF标志影响，CPU内部形成中断类型号N＝n，并转入相应中断服务过程

# 8086/8088中断响应过程



# 中断向量表

- 中断向量表
  - 也称中断服务程序入口地址表
  - 中断向量表安排在内存的前**1KB**，即**00000H~003FFH**
  - 每个服务程序入口地址**CS:IP**占用**4**个字节（**256*4＝1KB**），高字节存放段地址**CS**，低字节存放段内偏移**IP**，按中断类型号顺序存放

# 8086/8088中断向量表


- 256种中断类型的分配
  - ❖ 0～4为专用中断
  - ❖ 5～31为系统使用中断
    - □ 08H～0FH：8259A中断向量
    - □ 10H～1FH：BIOS中断向量
  - ❖ 32～255由用户定义中断
    - □ 20H～3FH：DOS中断调用
    - □ 40H～FFH：用户使用

# 中断操作过程示例

- 指令INT 68H执行过程

① 取中断类型号68H

② 计算中断向量地址

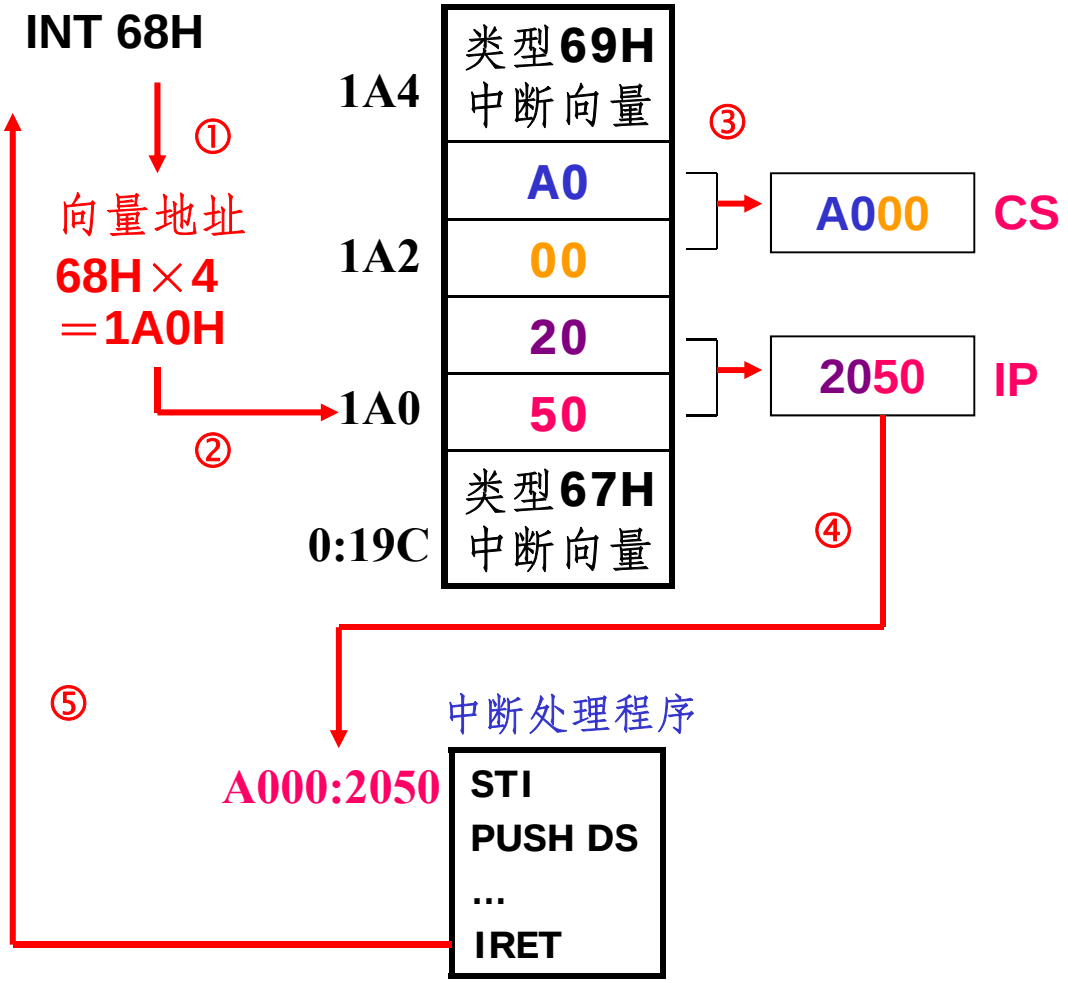
③ 取中断入口地址

低地址⇒IP

高地址⇒CS

④ 转中断服务程序

⑤ 中断返回到INT 68H的下一指令



# 中断类型号的获取

- 除数出错，单步断点，NMI，断点中断和溢出中断
  - ⇒中断类型号0～4
- 用户自己确定的软件中断INT n
  - ⇒中断类型号由n决定
- 外部可屏蔽中断INTR
  - ⇒中断类型号由硬件电路设计产生
- 外部可屏蔽中断INTR
  - ⇒中断类型号由可编程控制器8259A获得

# 主程序的编程

- 主程序中的初始化
  1. 设置中断向量
  2. 设置8259A的中断屏蔽寄存器的中断屏蔽位
  3. 设置CPU中断允许位标志IF

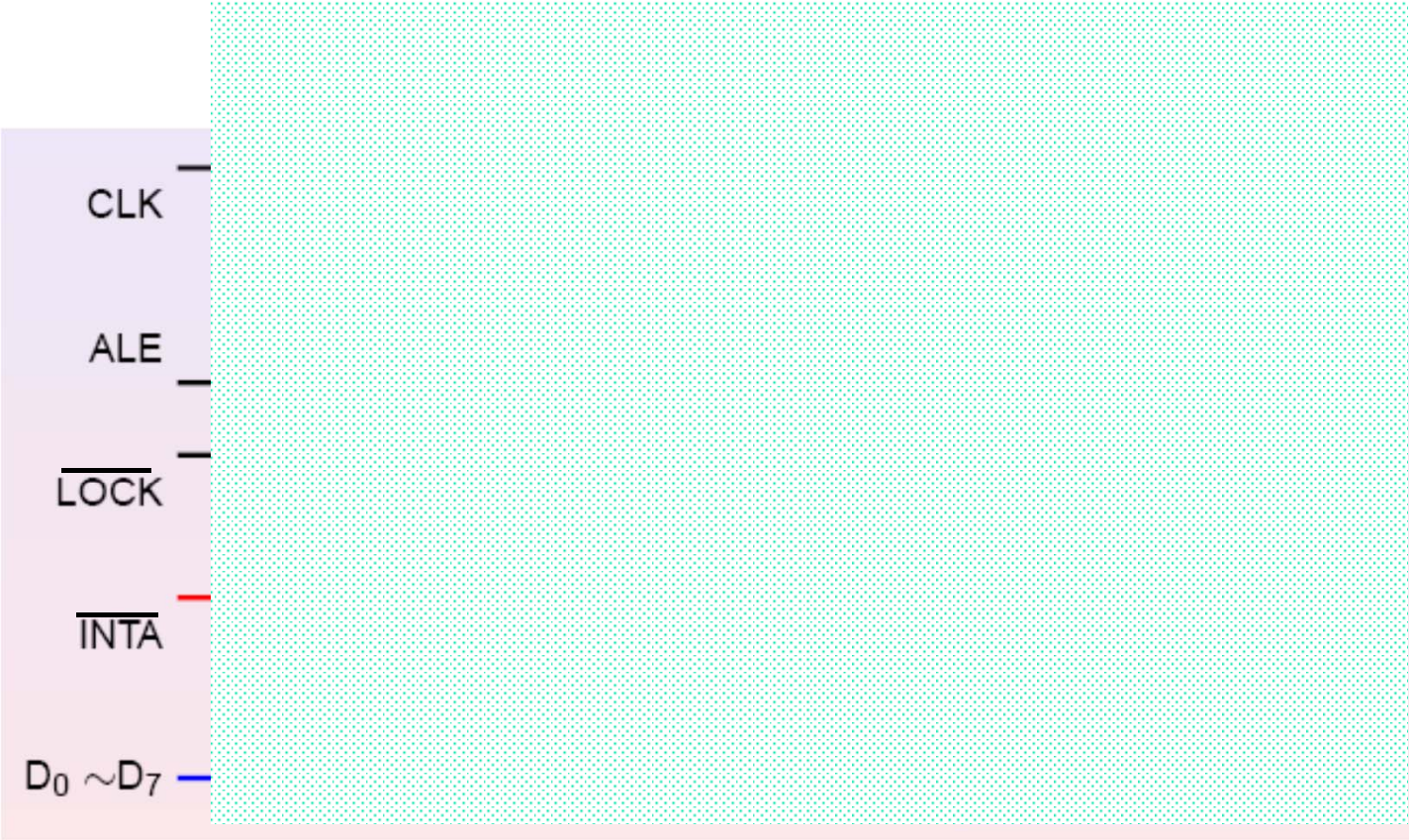
- 硬件（外设接口）和CPU自动完成
  1. 外设向CPU INTR端发中断请求
  2. 执行完当前指令CPU发两个中断响应信号 $\overline{\text{INTA}}$
  3. CPU取中断类型号n
  4. CPU将当前PSW、CS、IP入栈保护
  5. 清IF、TF，禁止外部中断和单步中断
  6. 从中断向量表取[4n]→IP，[4n＋2]→CS
  7. 转向中断服务子程序

# 中断服务子程序

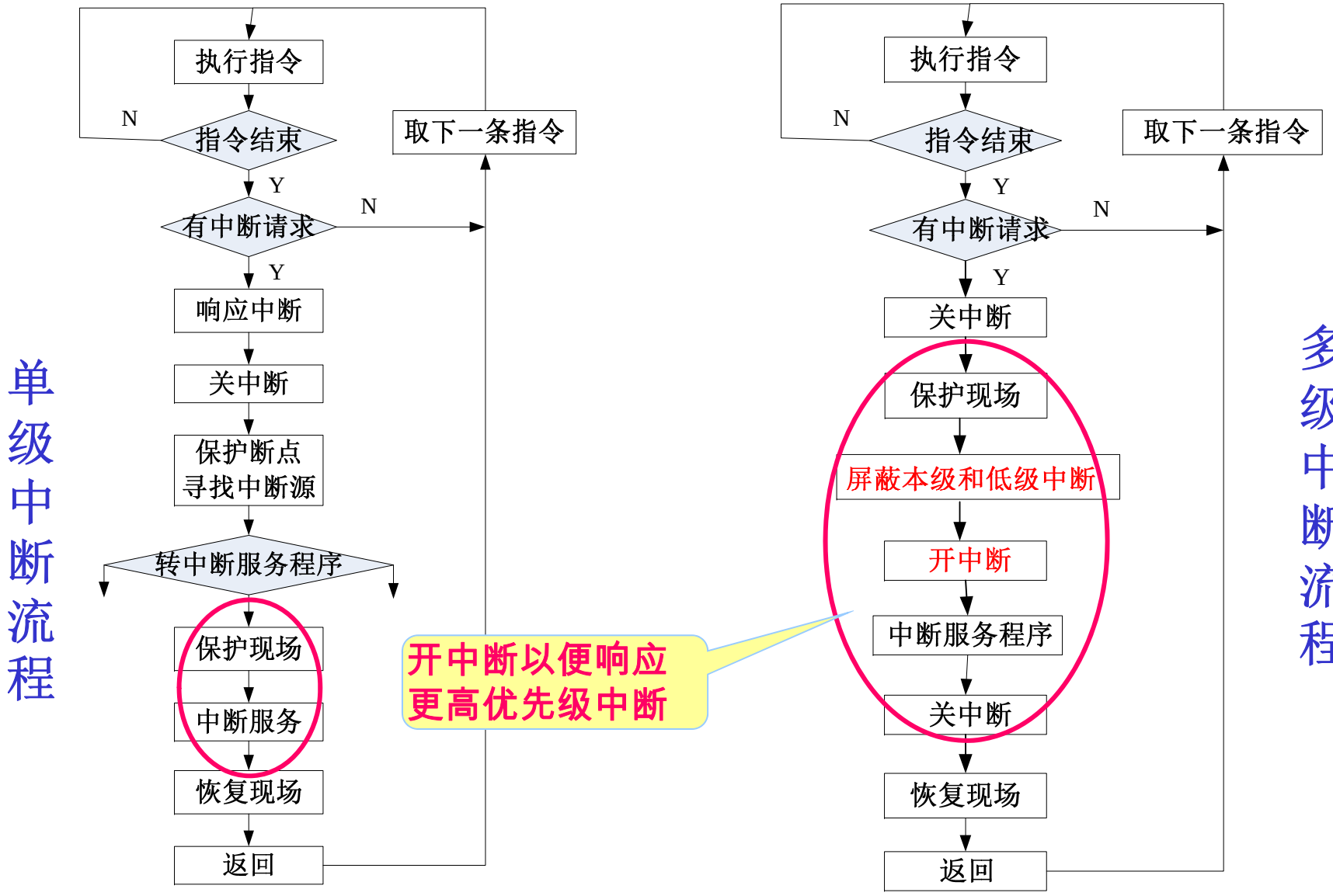
- 中断服务子程序的编写
  - 保护现场，用**PUSH**指令将各寄存器值入栈
  - 若允许中断嵌套，则用**STI**开中断（置**IF＝1**）
  - 执行中断处理程序
  - 用**CLI**关中断（清**IF＝0**）
  - 给中断命令寄存器送中断结束命令**EOI**
  - 恢复现场，用**POP**指令将各寄存器值退栈恢复
  - 用中断返回指令**IRET**返回主程序

# 中断响应时序

- 8086/8088中断响应总线周期时序图



# 三、中断优先级和中断嵌套



# 1. 中断优先级

- 中断优先级

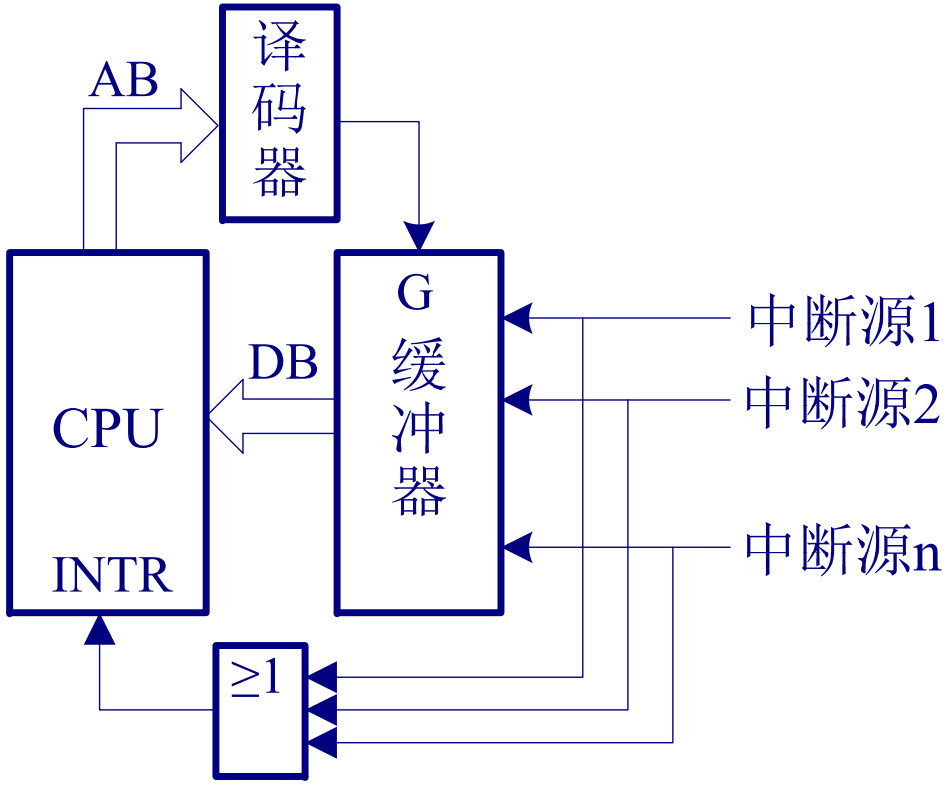
高 → 低

内中断（除法错、INTO、INT n）
非屏蔽中断（NMI）
可屏蔽中断（INTR）
单步中断

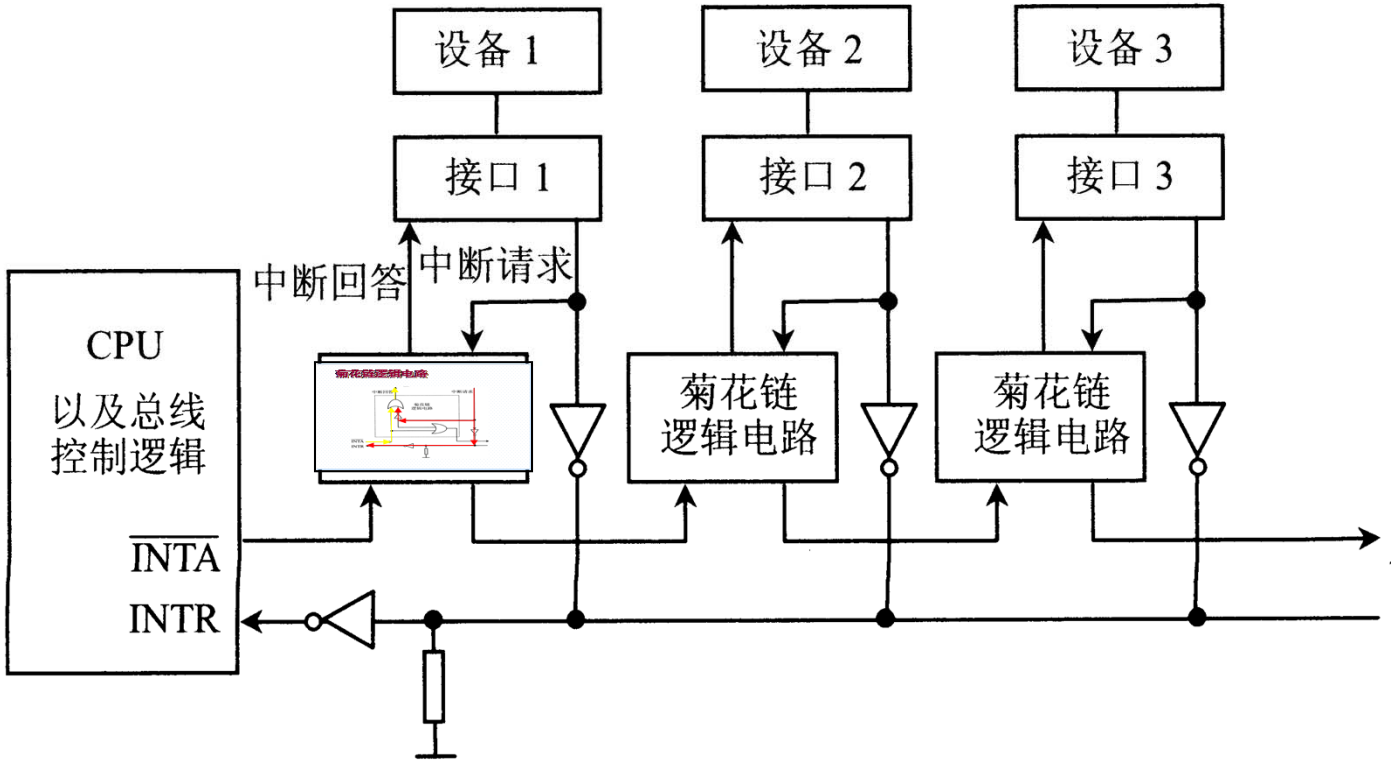
高 ↓ 低

# 可屏蔽中断的优先级设定

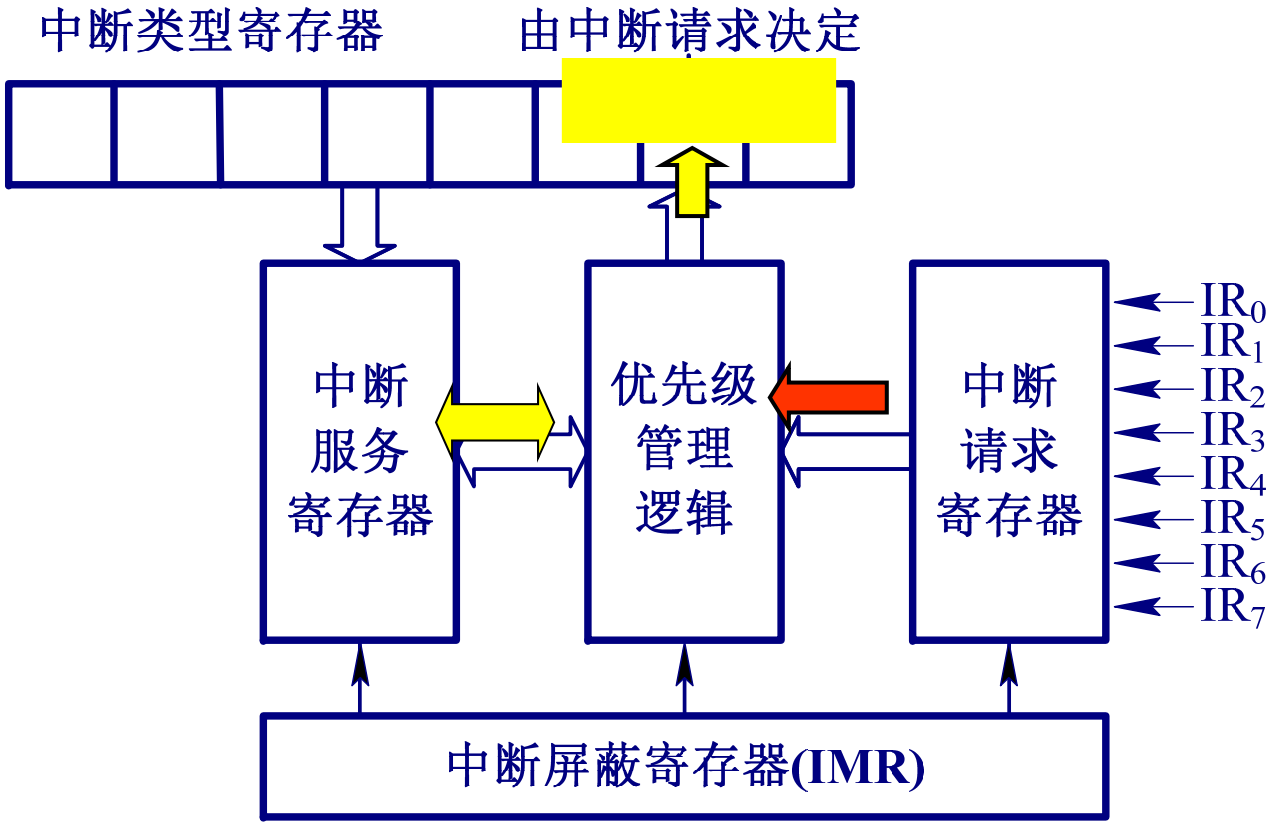
- 软件查询中断优先级
  - 软件查询状态位的次序决定外设中断优先权



- 硬件查询优先方式——菊花链法
  - 越靠近CPU的外设，优先级越高

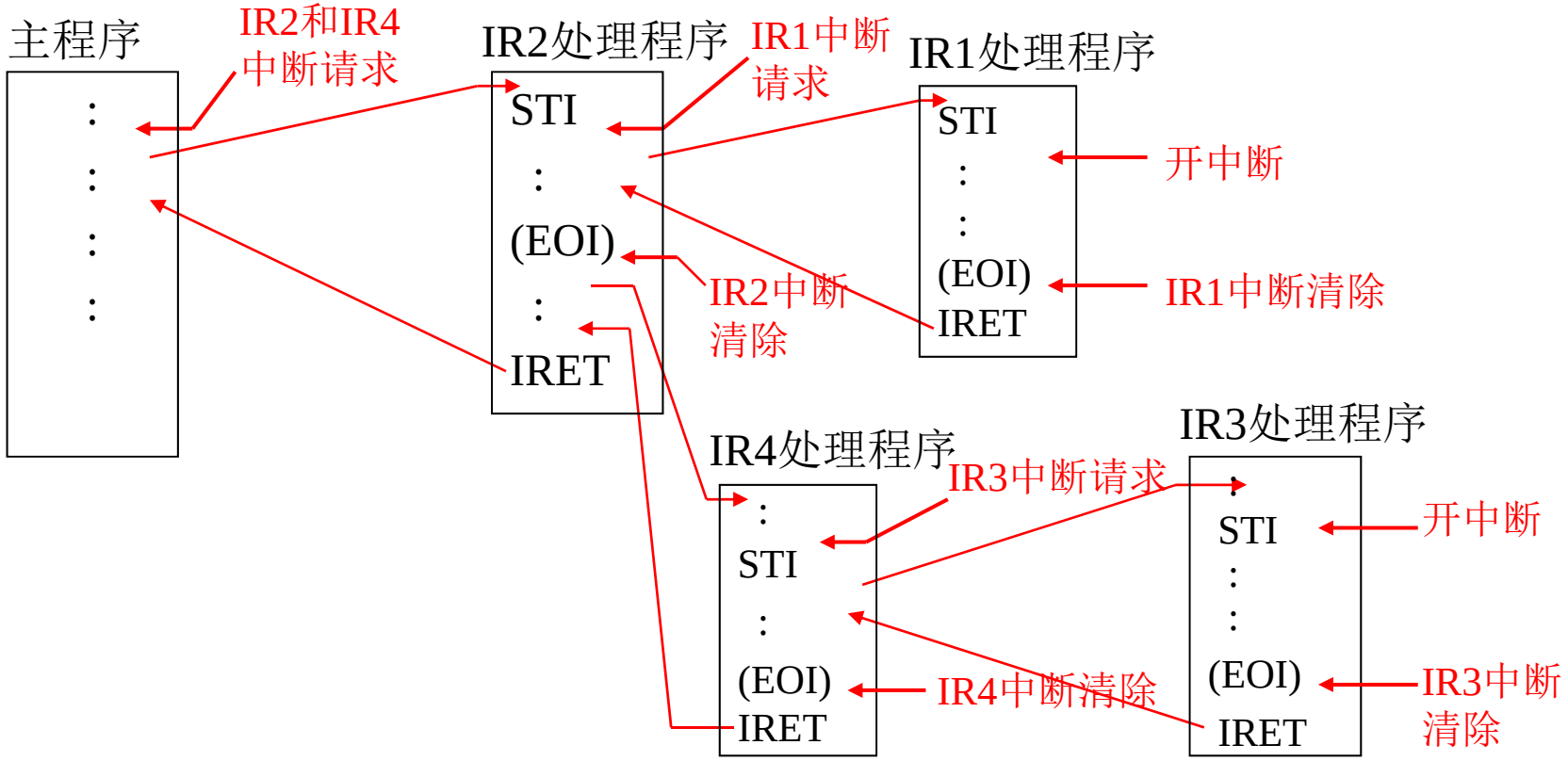


- 矢量中断优先级
  - 由优先级管理逻辑判别最高优先级中断请求



# 2. 中断嵌套

- 高优先级的中断源能中断低优先级的中断处理



# 习题：

p55：1，5，9，11，12，14，16，18，20