第2章 微处理器指令系统

第2章 微处理器指令系统

- 8088/8086的编程结构
- 8088/8086的存储器组织
- 8088/8086的寻址方式
- 8088/8086的基本指令
  - 数据传送
  - 算术运算
  - 逻辑运算、移位
  - 控制转移、功能调用
- 汇编语言程序段

8086与8088简介

- Intel 8086 是美国Intel公司于1978年发布的一款16-bit 的微处理器(microprocessor) 芯片。
- Intel 8088 发布于1979年, 它采用16-bit的内部数据总线和8-bit的外部数据总线, 被称为准16-bit的微处理器。该处理器被应用于最早的IBM PC的设计中, 在IBM PC XT中也被采用。
- 8086导致了x86架构的产生，并最终成为英特尔最成功的处理器系列。

为什么从8086学起

- 微处理器是嵌入式系统硬件的核心
- 汇编语言直接与特定的微处理器系列相联系
- 接口技术需结合一款微处理器来学习
- 实验条件成熟

2.1.2 8088/8086的功能结构

- 8088的内部结构从功能上分成两个单元

  1.总线接口单元BIU

  管理8088与系统总线的接口

  负责CPU对存储器和外设进行访问

  2.执行单元EU

  负责指令的译码、执行和数据的运算

- BIU和EU这两个单元相互独立，分别完成各自的操作，还可以并行执行，实现指令读取和执行的流水线操作。

2.1.3 8088/8086的寄存器结构

2.1.3 8088/8086的寄存器结构

- 8088/8086的寄存器组有
  - 8个通用寄存器
  - 4个段寄存器
  - 1个标志寄存器
  - 1个指令指针寄存器

14个寄存器均为16位！

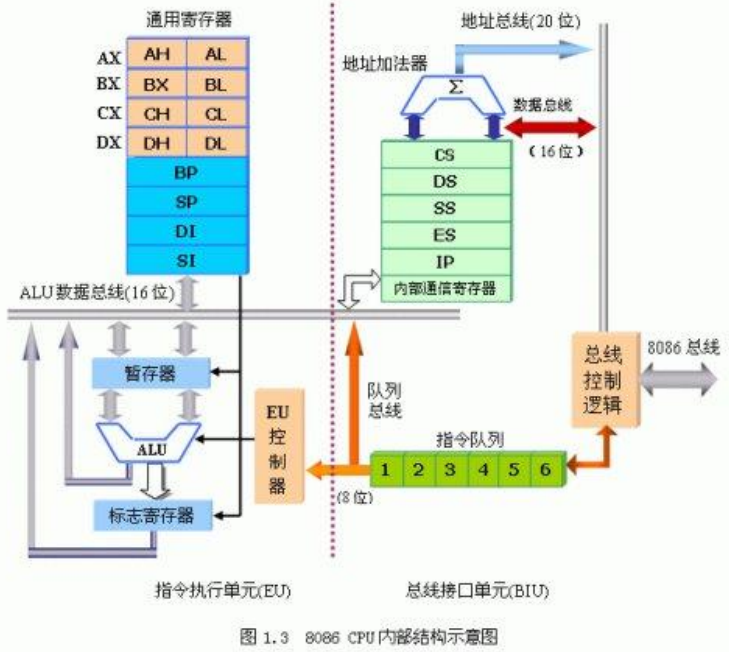
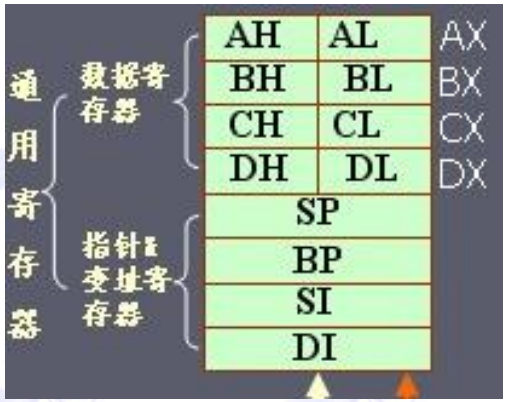


图 1.3 8086 CPU 内部结构示意图

汇编语言程序员看到的处理器，就是寄存器
所以，一定要熟悉这些寄存器的名称和作用

1. 通用寄存器

- 8088有8个16位的通用寄存器
  - （1）数据寄存器：**AX BX CX DX**
  - （2）变址寄存器：**SI DI**
  - （3）指针寄存器：**BP SP**



- 4个数据寄存器还可以分成高8位和低8位两个独立的寄存器，这样又形成8个通用的8位寄存器

AX：AH AL　　BX：BH BL

CX：CH CL　　DX：DH DL

（1）数据寄存器

- AX称为累加器（Accumulator）

  使用频度最高，用于算术、逻辑运算以及与外设传送信息等。

- BX称为基址寄存器（Base address Register）

  常用做存放存储器地址。

- CX称为计数器（Counter）

  作为循环和串操作等指令中的隐含计数器。

- DX称为数据寄存器（Data register）

  常用来存放双字长数据的高16位，或存放外设端口地址。

（2）变址寄存器

- 16位变址寄存器SI和DI
- 常用于存储器变址寻址方式时提供地址
  - SI是源地址寄存器（Source Index）
  - DI是目的地址寄存器（Destination Index）
- 在串操作类指令中，SI、DI还有较特殊的用法

现在不必完全理解，以后会详细展开

（3）指针寄存器

- 指针寄存器用于寻址内存堆栈内的数据
  - SP为堆栈指针寄存器（Stack Pointer），指示堆栈栈顶的位置（偏移地址）
  - BP为基址指针寄存器（Base Pointer），表示数据在堆栈段中的基地址
- SP和BP寄存器与SS段寄存器联合使用以确定堆栈段中的存储单元地址

2. 指令指针寄存器

- IP（Instruction Pointer）为指令指针寄存器，指示指令在主存储器中的位置
- 随着指令的执行，IP将自动修改以指示下一条指令所在的存储器位置
- IP寄存器是一个专用寄存器
- IP寄存器与CS段寄存器联合使用以确定下一条指令所在的存储单元地址（CS:IP即CPU中的PC）

3. 标志寄存器

- 标志（Flag）用于反映指令执行的结果或控制指令的执行方式。
- 8088处理器用一个16位的寄存器存放各种标志，称为标志寄存器（FLAGS Register），一些机器中也称作程序状态字寄存器（PSW）。

程序设计需要利用标志的状态

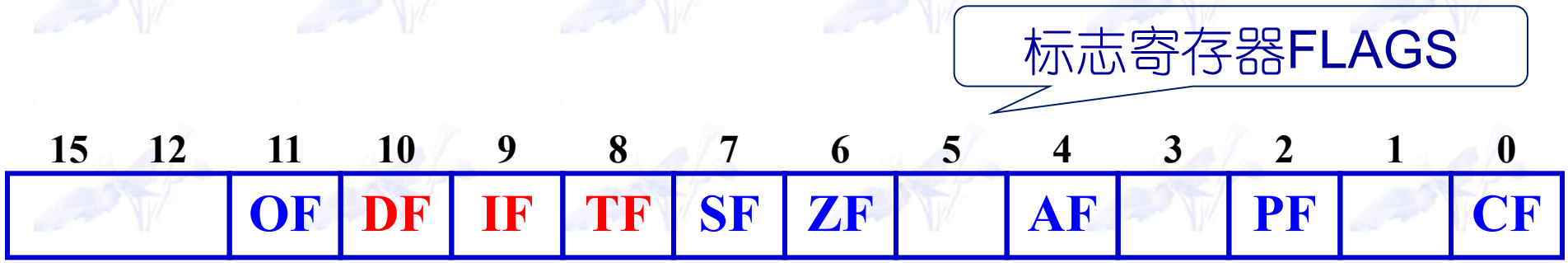
3. 标志寄存器 —— 标志分类

- 状态标志:用来记录程序运行结果的状态信息，许多指令的执行都将相应地设置它们

  CF ZF SF PF OF AF

- 控制标志:可由程序根据需要用指令设置，用于控制处理器执行指令的方式

  DF IF TF



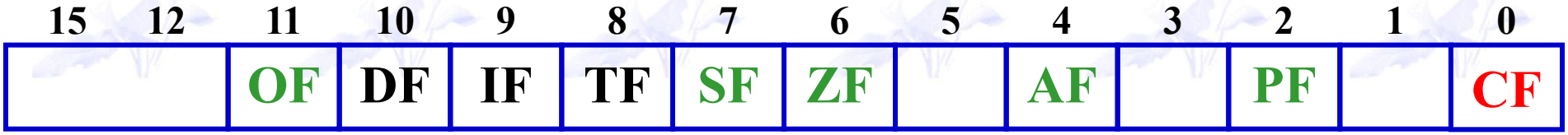
（1）进位标志CF（Carry Flag）

- 当运算结果的最高有效位有进位（加法）或借位（减法）时，进位标志置1，即CF＝1；否则CF＝0

例：

3AH + 7CH＝B6H，没有进位：CF = 0

AAH + 7CH＝（1）26H，有进位：CF = 1



（2）零标志ZF（Zero Flag）

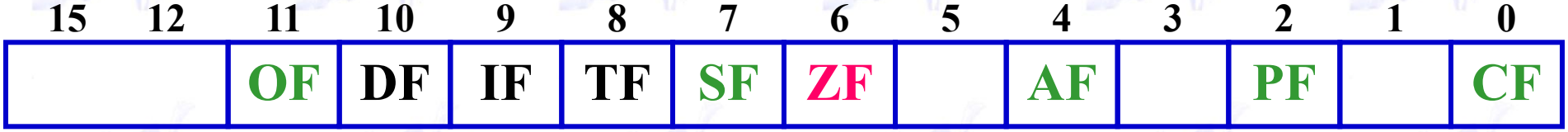
- 若运算结果为0，则ZF＝1；否则ZF＝0。

例：

3AH＋7CH＝B6H，结果不是零：ZF＝0

84H＋7CH＝（1）00H，结果是零：ZF＝1

注意：ZF为1表示结果是0



（3）符号标志SF（Sign Flag）

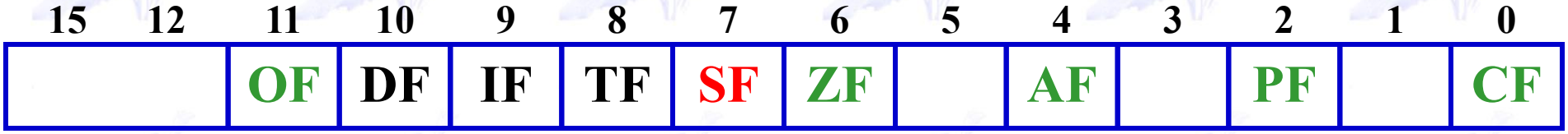
- 运算结果最高位为1，则SF＝1；　否则SF＝0。

例：

3AH＋7CH＝B6H，最高位D_7＝1，则SF＝1

84H＋7CH＝（1）00H，最高位D_7＝0，则SF＝0

有符号数据用最高有效位表示数据的符号，所以，最高有效位就是符号标志的状态



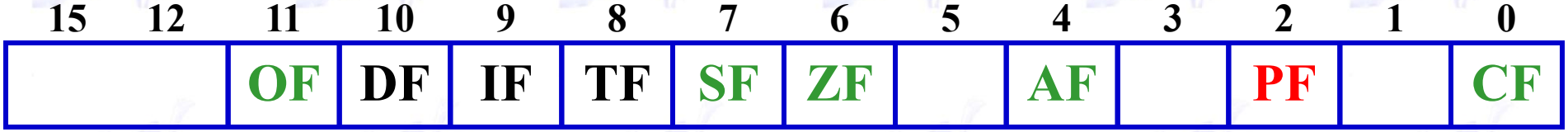
（4）奇偶标志PF（Parity Flag）

- 当运算结果最低字节中“1”的个数为零或偶数时，PF＝1；否则PF＝0。

例：3AH＋7CH＝B6H＝10110110B

结果中有5个“1”，是奇数，则PF＝0

PF标志仅反映最低8位中“1”的个数是偶或奇，即使是进行16位字操作也是如此



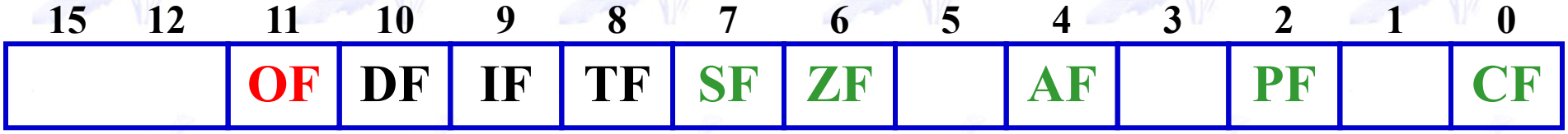
（5）溢出标志OF（Overflow Flag）

- 若有符号数运算发生溢出，则OF＝1；否则OF＝0。

例：

3AH + 7CH＝B6H，产生溢出，则OF＝1

AAH + 7CH＝（1）26H，没有溢出，则OF＝0



第2章：什么是溢出

- 处理器内部以补码表示有符号数

- B6H＝10110110B，最高位为1，作为有符号数是负数
- 对B6H求反加1等于：01001001B＋1＝01001010B＝4AH＝74
- 所以，B6H表达有符号数的真值为－74

3
已经超出－128～＋127范围，产生溢出，故OF＝1；
补码B6H表达真值是－74，显然运算结果也不正确

第2章：溢出和进位的区别

- 溢出标志OF和进位标志CF是两个意义不同的标志
- 进位标志表示无符号数运算结果是否超出范围，运算结果仍然正确
- 溢出标志表示有符号数运算结果是否超出范围，运算结果已经不正确

第2章： 溢出和进位的对比

例1：3AH＋7CH＝B6H

无符号数运算： 58＋124＝182

范围内，无进位

有符号数运算： 58＋124＝182

范围外，有溢出

例2：AAH＋7CH＝（1）26H

无符号数运算： 170＋124＝294

范围外，有进位

有符号数运算： －86＋124＝28

范围内，无溢出

第2章：溢出和进位的应用场合

- 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
- 应该利用哪个标志，则由程序员来决定。也就是说，如果将参加运算的操作数认为是无符号数，就应该关心进位；认为是有符号数，则要注意是否溢出

第2章：溢出的判断

- 判断运算结果是否溢出有一个简单的规则：
- 只有当两个相同符号数相加（包括不同符号数相减），而运算结果的符号与原数据符号相反时，产生溢出；因为，此时的运算结果显然不正确
- 其他情况下，则不会产生溢出

例1：3AH＋7CH＝B6H　溢出

例2：AAH＋7CH　无溢出

例3：3AH－7CH　无溢出

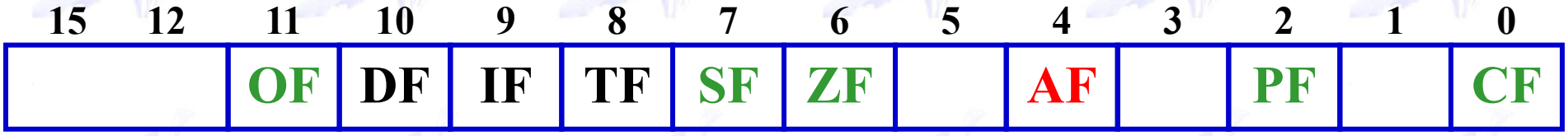
例4：AAH－7CH＝2DH　溢出

（6）辅助进位标志AF（Auxiliary Carry Flag）

运算中D3位向高位有进位或借位时，AF＝1；否则AF＝0。

例：3AH＋7CH＝B6H，D3有进位：AF＝1

这个标志主要由处理器内部使用，用于十进制算术运算调整指令中，用户一般不必关心。

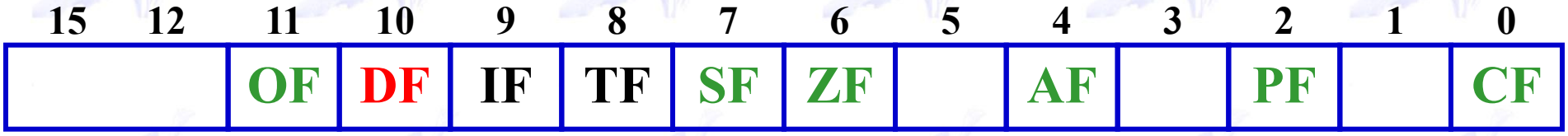


（7）方向标志DF（Direction Flag）

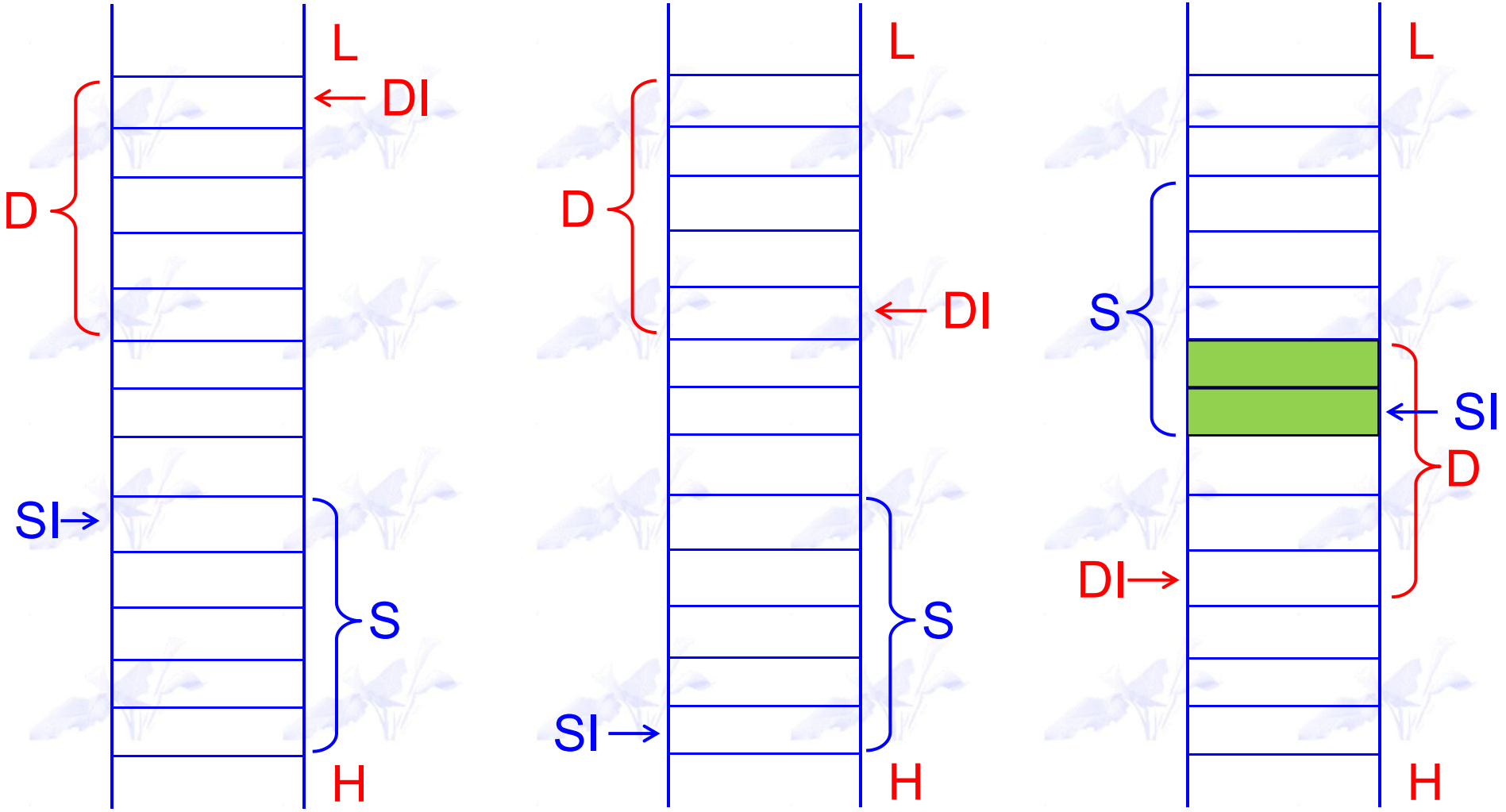
- 用于串操作指令中，控制地址的变化方向
  - 设置DF＝0，存储器地址自动增加；
  - 设置DF＝1，存储器地址自动减少

DF标志设置指令

- CLD指令复位方向标志：DF＝0
- STD指令置位方向标志：DF＝1



（7）方向标志DF（Direction Flag）



DF＝0，地址增加

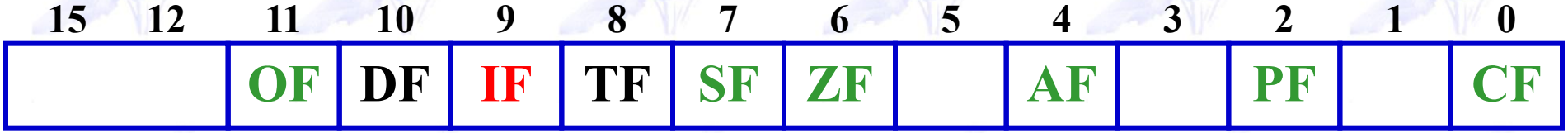
DF＝1，地址减小

（8）中断允许标志IF（Interrupt-enable Flag）

- 控制可屏蔽中断是否可以被处理器响应
  - 设置IF＝1，则允许中断
  - 设置IF＝0，则禁止中断

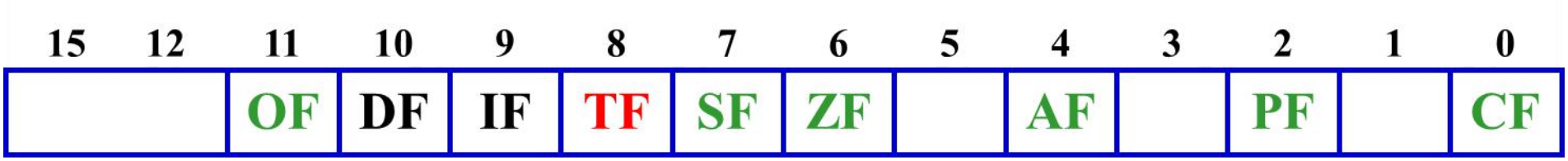
IF标志位控制指令

- CLI指令复位中断标志：IF＝0
- STI指令置位中断标志：IF＝1



（9）陷阱标志TF（Trap Flag）

- 用于控制处理器进入单步操作方式
  - 设置TF＝0，处理器正常工作
  - 设置TF＝1，处理器单步执行指令
  - 所谓单步执行指令，即处理器在每条指令执行结束时产生一个编号为1的内部中断，称为单步中断
  - 因此TF也称为单步标志
  - 利用单步中断可对程序进行逐条指令的调试，称为单步调试



2.1.4 8088/8086的存储器结构

2.1.4 8088/8086的存储器结构

主存储器是计算机中存储程序和数据的地方。掌握数据的存储格式及存储器的分段管理是对8086汇编语言程序员的基本要求。

1. 数据的存储格式

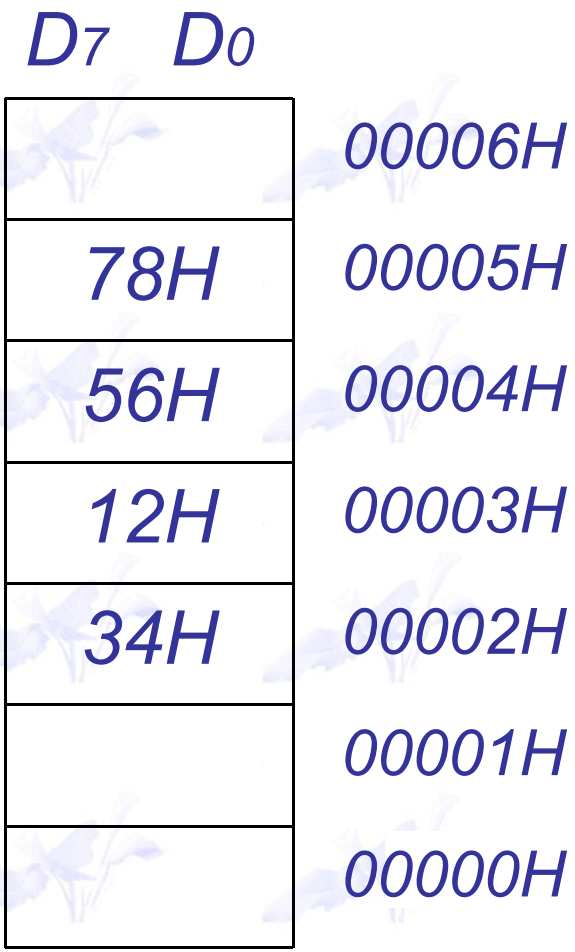
- 计算机中信息的单位
  - 二进制位Bit：存储一位二进制数，0或1
  - 字节Byte：8个二进制位，D_7～D_0
  - 字Word：16位，2个字节，D_{15}～D_0
  - 双字DWord：32位，4个字节，D_{31}～D_0
- 最低有效位LSB：数据的最低位，D_0位
- 最高有效位MSB：数据的最高位，对应字节、字、双字分别指D_7、D_{15}、D_{31}位

图示

- LSB： Least Significant Bit
- MSB： Most Significant Bit

（1）存储单元及其存储内容

- 每个存储单元存放一个字节的内容。
- 每个存储单元都有一个编号，称为存储器地址。



左图中地址为0002H的单元中存放了数据34H，描述方式如下：

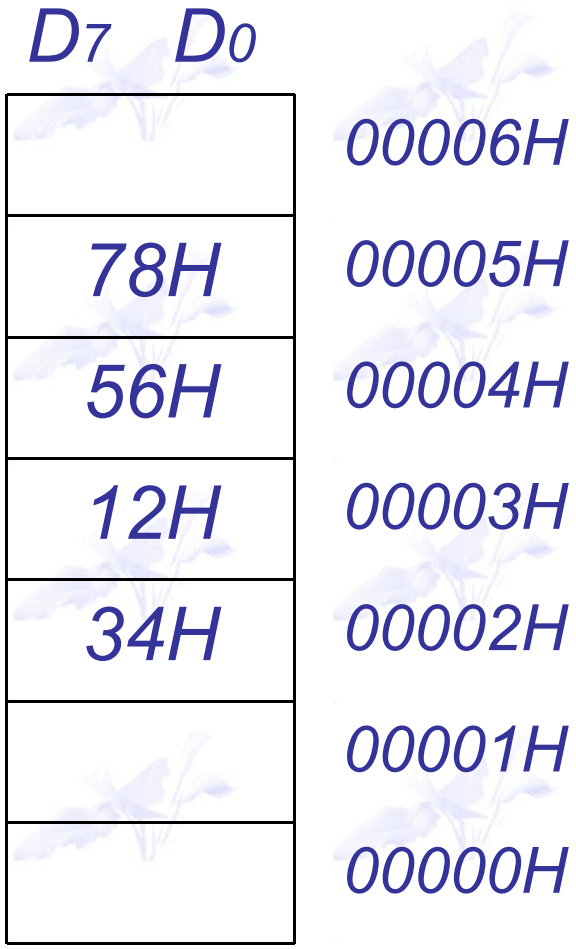
[0002H]＝34H

（2）多字节数据存放方式

- 多字节数据在存储器中需占用连续的多个存储单元
  - 存放时，数据的低位存入较低地址的单元，数据的高位存入较高地址的单元；
  - 访问时，用该数据占用的多个存储单元的最低地址值来表示该多字节数据的内存地址。

例：图中地址为0002H的“字”单元的内容为 [0002H] = 1234H

地址为0002H的“双字”单元的内容为 [0002H] = 78561234H



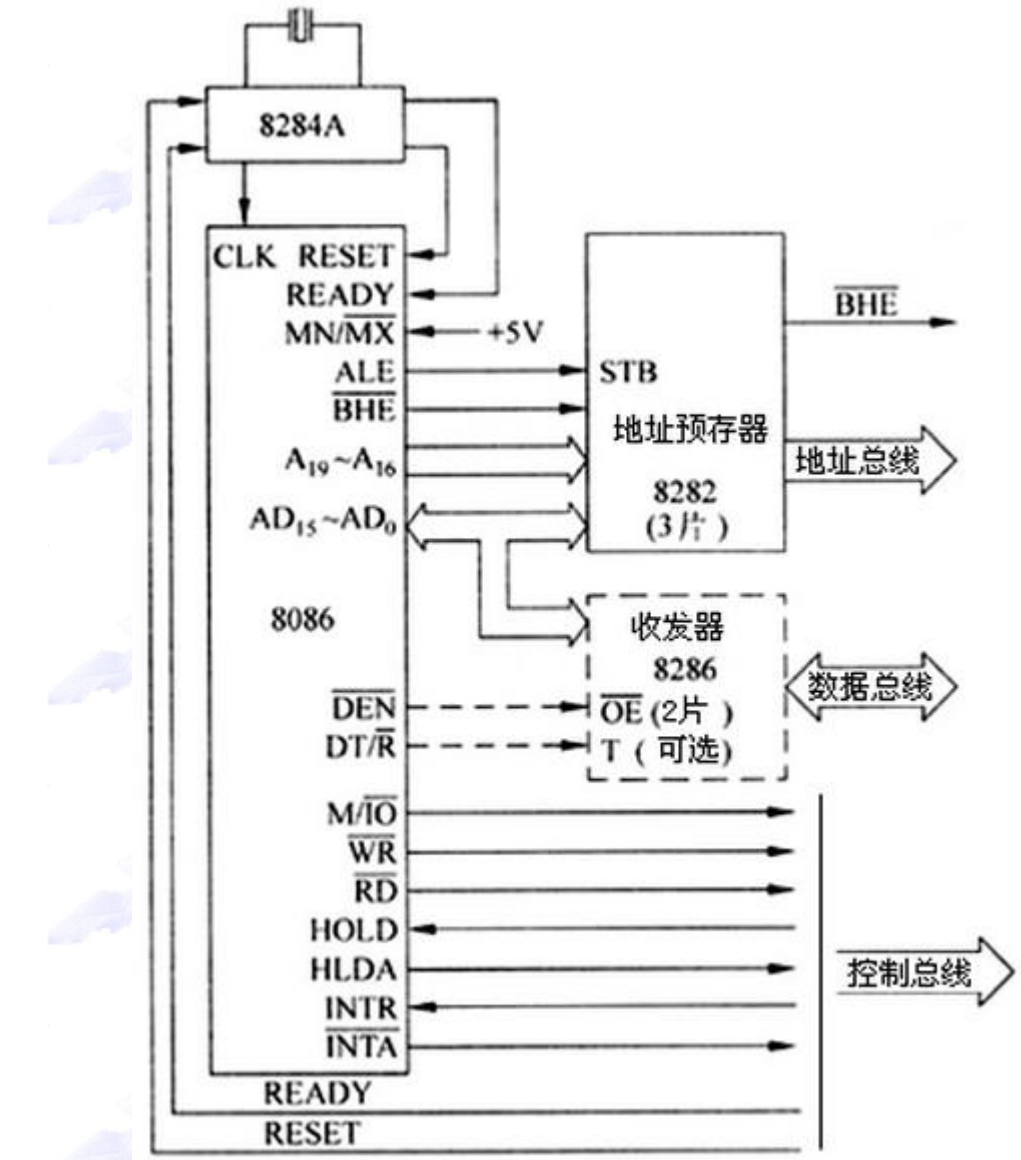
（2）多字节数据存放方式

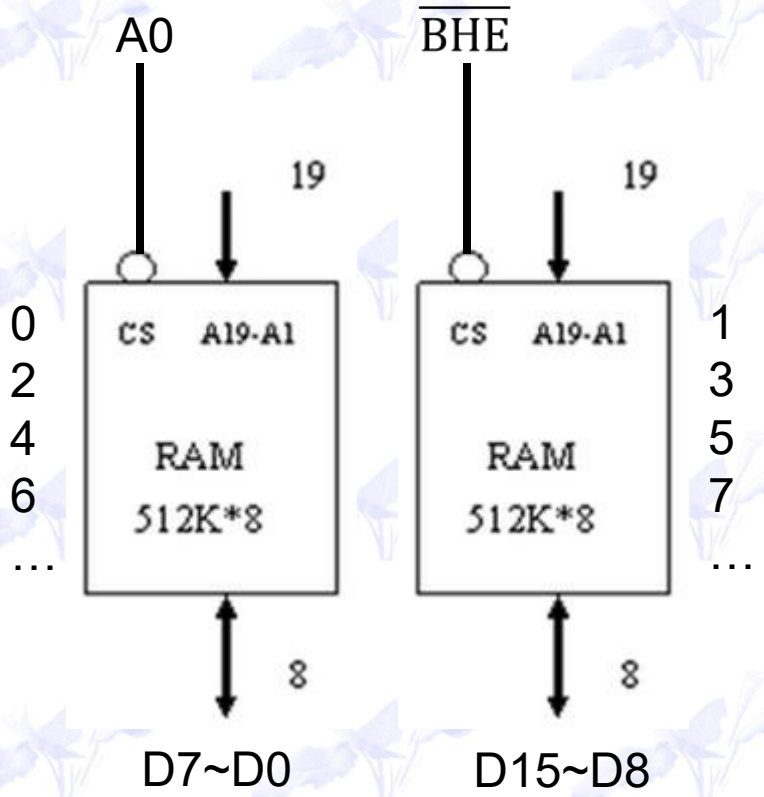
- 80x86处理器采用“低对低、高对高”的存储形式，被称为“小端方式Little Endian”。
- 与此规则相反的存储形式称为“大端方式Big Endian”。
- X86系列采用Little Endian方式，PowerPC 采用Big Endian方式，ARM处理器则可根据需要选择上述两种方式中的一种。

（3）数据的地址对齐

- 同一个存储器地址可以是一个字节单元地址，也可以是一个字单元地址，或者是一个双字单元地址（视具体情况来确定）
- 字单元安排在偶地址（xxx0B）、双字单元安排在模4地址（xx00B）等，被称为“地址对齐（Align）”
- 对于不对齐地址的数据，处理器访问时，需要额外的访问存储器时间，因此数据存放时应该将数据的地址对齐，以取得较高的存取速度

图示

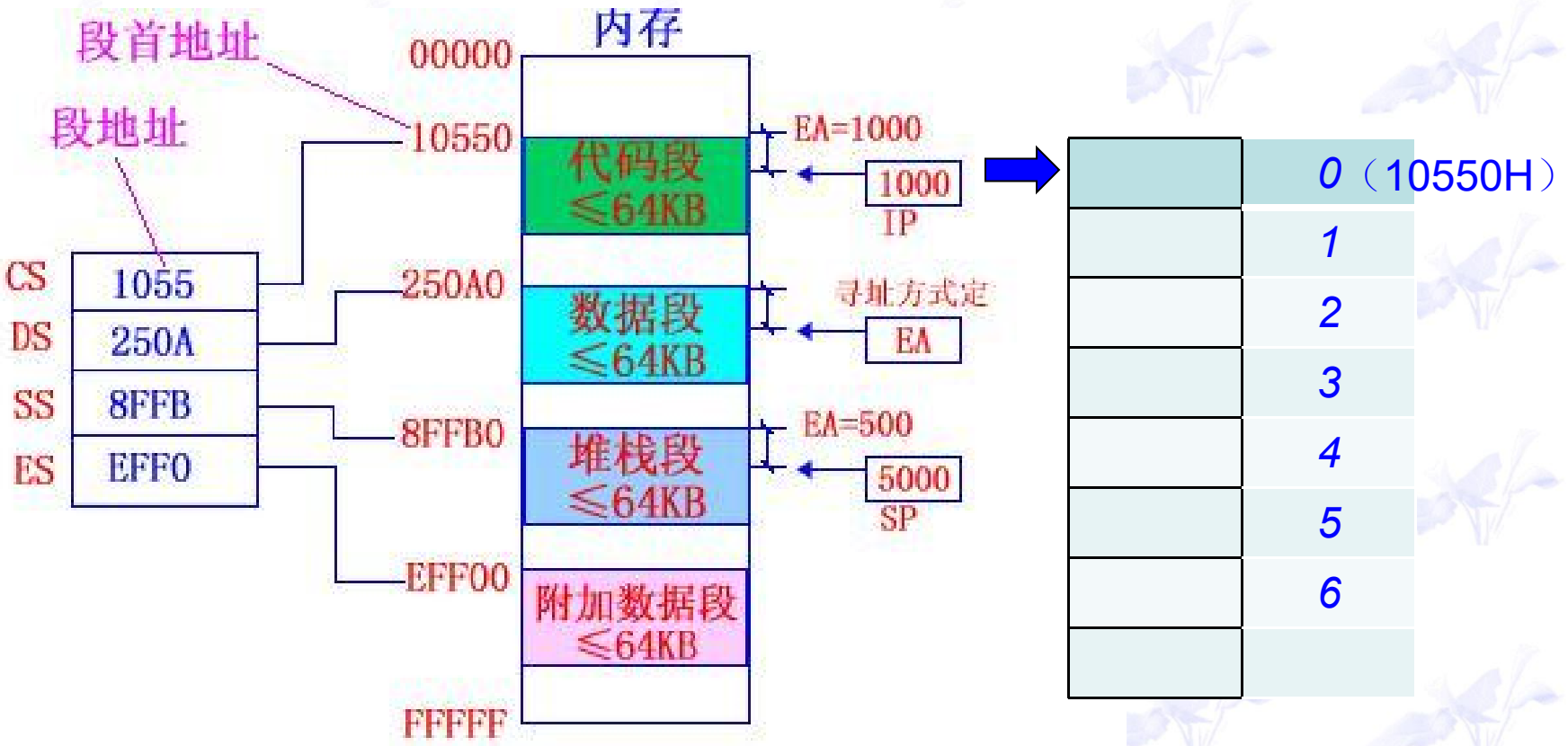

8284A
CLK RESET
READY
MN/MX
+5V
ALE
BHE
A19~A16
AD15~AD0
8086
DEN
DT/R
M/IO
WR
RD
HOLD
HLDA
INTR
INTA
READY
RESET
STB
地址预存器
8282
(3片)
BHE
地址总线
收发器
8286
OE (2片)
T (可选)
数据总线
控制总线


A0
BHE
19
19
CS
A19-A1
CS
A19-A1
RAM
512K*8
RAM
512K*8
0
2
4
6
…
1
3
5
7
…
8
8
D7~D0
D15~D8

2. 存储器的分段管理

- 8088CPU有20条地址线
  - 最大可寻址的内存空间为2^{20}＝1MB
  - 物理地址范围从00000H～FFFFFH
- 8088CPU将1MB空间分成许多逻辑段（Segment）
  - 每个段最大限制为64KB
  - 段地址的低4位为0000B

2. 存储器的分段管理



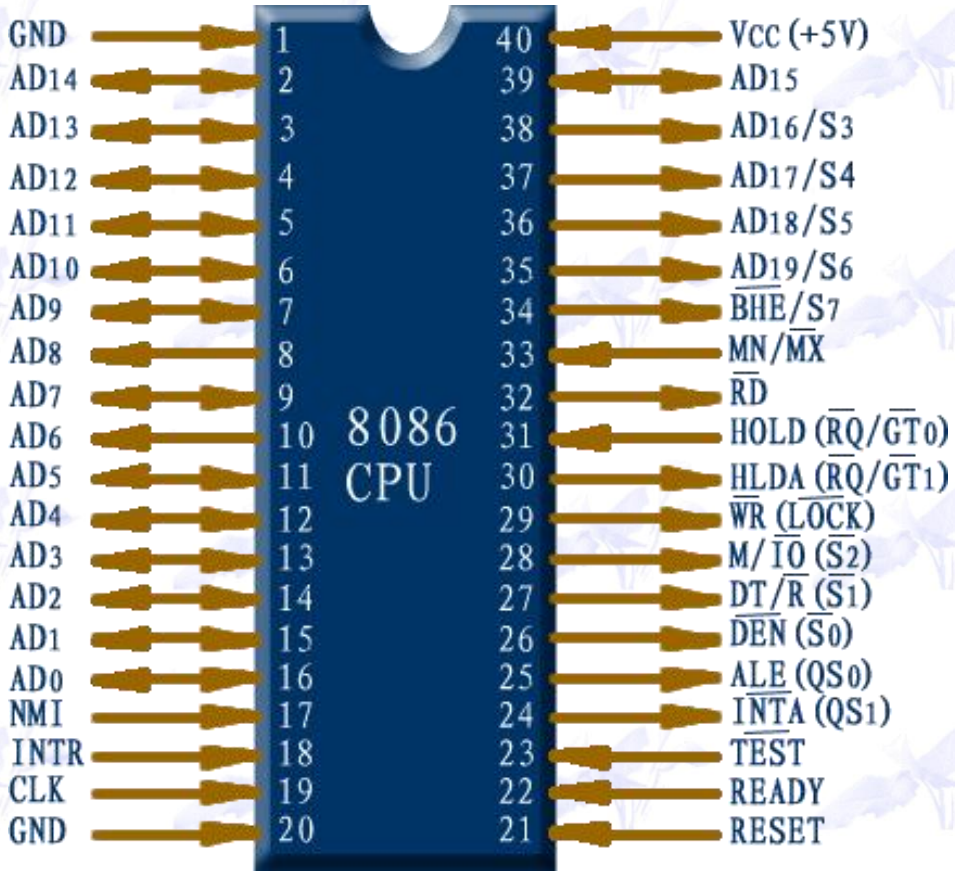
代码段、数据段、堆栈段、附加段在内存中的分配示意图

（1）物理地址和逻辑地址

- 8088CPU的存储系统中，每个物理存储单元有一个唯一的20位编号，称为该单元的物理地址，范围从00000H ~ FFFFFH
- 在用户编程时，采用段基地址:段内偏移地址的形式描述指定的内存单元，称为单元的逻辑地址。

物理地址　14700H
逻辑地址　1460H:100H

物理地址


8086
CPU
GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
40 Vcc (+5V)
39 AD15
38 AD16/S3
37 AD17/S4
36 AD18/S5
35 AD19/S6
34 BHE/S7
33 MN/MX
32 RD
31 HOLD (RQ/GT0)
30 HLDA (RQ/GT1)
29 WR (LOCK)
28 M/IO (S2)
27 DT/R (S1)
26 DEN (S0)
25 ALE (QS0)
24 INTA (QS1)
23 TEST
22 READY
21 RESET

逻辑地址

- 段地址说明逻辑段在主存中的起始位置。
- 8088规定段地址必须是模16地址：xxxx0H
- 省略低4位默认的0000B，段地址就可以用16位二进制数表示，也就能用16位段寄存器存储段地址。
- 偏移地址说明主存单元距离段起始位置的偏移量。
- 每段不超过64KB，偏移地址也可用16位二进制数表示。

（2）物理地址和逻辑地址的转换

- 将逻辑地址中的段地址左移4位，加上偏移地址就得到20位物理地址。
- 一个物理地址可以对应多个逻辑地址。

逻辑地址	1460:100、	1380:F00
物理地址	14700H	14700H

段地址左移4位 →	14600H	13800H
加上偏移地址 →	＋ 100H	＋ F00H
得到物理地址 →	14700H	14700H

3. 段寄存器

- 8088有4个16位的段寄存器，每个段寄存器用来确定一个逻辑段的起始地址
  - CS (Code Segment)指明代码段的起始地址
  - SS (Stack Segment)指明堆栈段的起始地址
  - DS (Data Segment)指明数据段的起始地址
  - ES (Extra Segment)指明附加段的起始地址

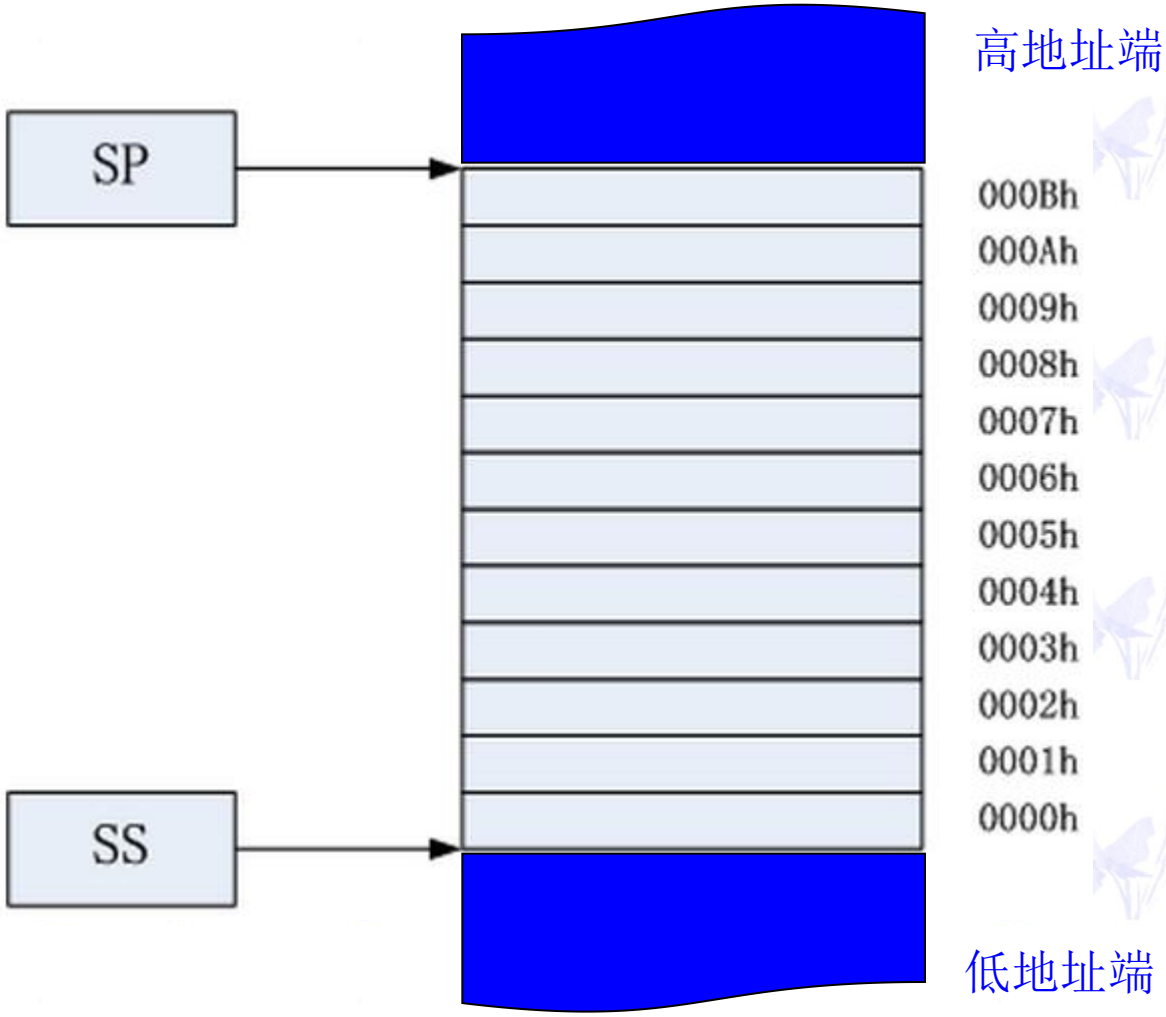
（1）代码段寄存器CS（Code Segment）

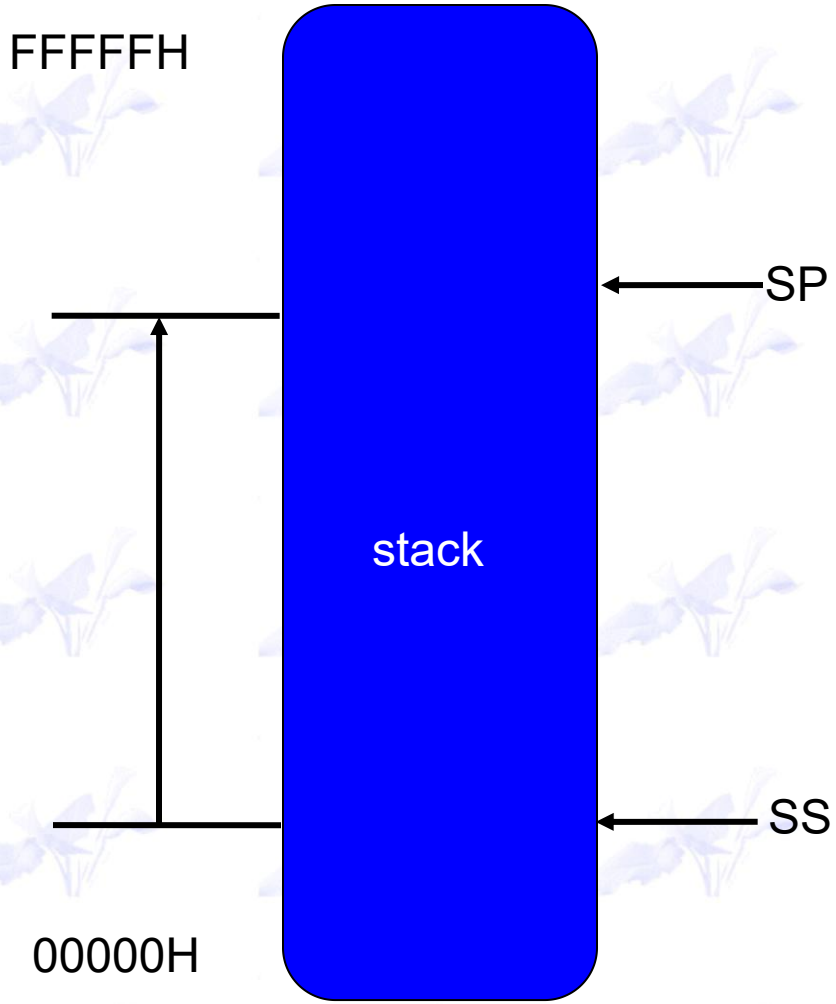
- 代码段用来存放程序的指令序列
  - 代码段寄存器CS存放代码段的段地址
  - 指令指针寄存器IP指示下条指令的偏移地址
- 处理器利用CS:IP取得下一条要执行的指令的地址

（2）堆栈段寄存器SS（Stack Segment）

- 堆栈段是堆栈所在的主存区域
  - 堆栈段寄存器SS存放堆栈段的段地址
  - 堆栈指针寄存器SP指示堆栈栈顶的偏移地址
- 处理器利用SS:SP操作栈顶的数据

8086堆栈操作


高地址端
SP
000Bh
000Ah
0009h
0008h
0007h
0006h
0005h
0004h
0003h
0002h
0001h
0000h
SS
低地址端



SS定位到堆栈段的起始地址（基地址，低地址），栈底位于堆栈段的有效地址的最末端（高地址）。SP初始化为堆栈段的大小，SS:SP永远指向堆栈的栈顶。在初始化时，SS:SP指向堆栈段的最高地址（此时，栈底和栈顶都指向这一内存地址）。随着压入元素，SP不断变小，进而SS:SP代表的栈顶地址变小，不再等于栈底地址，而是逐渐靠近堆栈段的起始地址，当SP为0时，SS:SP代表的栈顶地址与SS:0000代表的堆栈段的起始地址相等，进而确定栈满。

（3）数据段寄存器DS（Data Segment）

- 数据段存放运行程序所用的数据
  - 数据段寄存器DS存放数据段的段地址
  - 操作数的偏移地址（有效地址EA）依采用的寻址方式得到
- 程序中利用DS:EA形式的逻辑地址存取数据段中的数据

（4）附加段寄存器ES（Extra Segment）

- 附加段是附加的数据段，也保存数据
  - 附加段寄存器ES存放附加段的段地址
  - 操作数的偏移地址（有效地址EA）根据采用的寻址方式得到
- 处理器利用ES:EA存取附加段中的数据
- 串操作指令将附加段作为其目的操作数的存放区域

4. 如何分配各个逻辑段

- 程序的指令序列必须安排在代码段
- 程序使用的堆栈一定在堆栈段
- 程序中的数据默认是安排在数据段，也经常安排在附加段，尤其是串操作的目的区必须是附加段
- 数据的存放比较灵活，实际上可以存放在任何一种逻辑段中

演示

5. 段超越前缀指令

- 指令中没有指明时，默认的数据访问在DS段；当使用BP访问主存时，则默认访问SS段
- 当要访问的数据不在默认的段中时，需要使用段超越前缀指明。8088指令系统中有4个段超越前缀：
  - CS: 代码段超越，使用代码段的数据
  - SS: 堆栈段超越，使用堆栈段的数据
  - DS: 数据段超越，使用数据段的数据
  - ES: 附加段超越，使用附加段的数据

示例

第2章：段超越的示例

- 没有段超越的指令实例：

```
MOV AX,[2000H]   ;AX←DS:[2000H]
；从默认的DS数据段取出数据
```

- 采用段超越前缀的指令实例：

```
MOV AX,ES:[2000H]  ;AX←ES:[2000H]
；从指定的ES附加段取出数据
```

总结

第2章 段寄存器的使用规定

访问存储器的方式	默认	是否可超越	偏移地址
取指令	CS	否	IP
堆栈操作	SS	否	SP
一般数据访问	DS	CS ES SS	有效地址EA
BP基址的寻址方式	SS	CS ES DS	有效地址EA
串操作的源操作数	DS	CS ES SS	SI
串操作的目的操作数	ES	否	DI

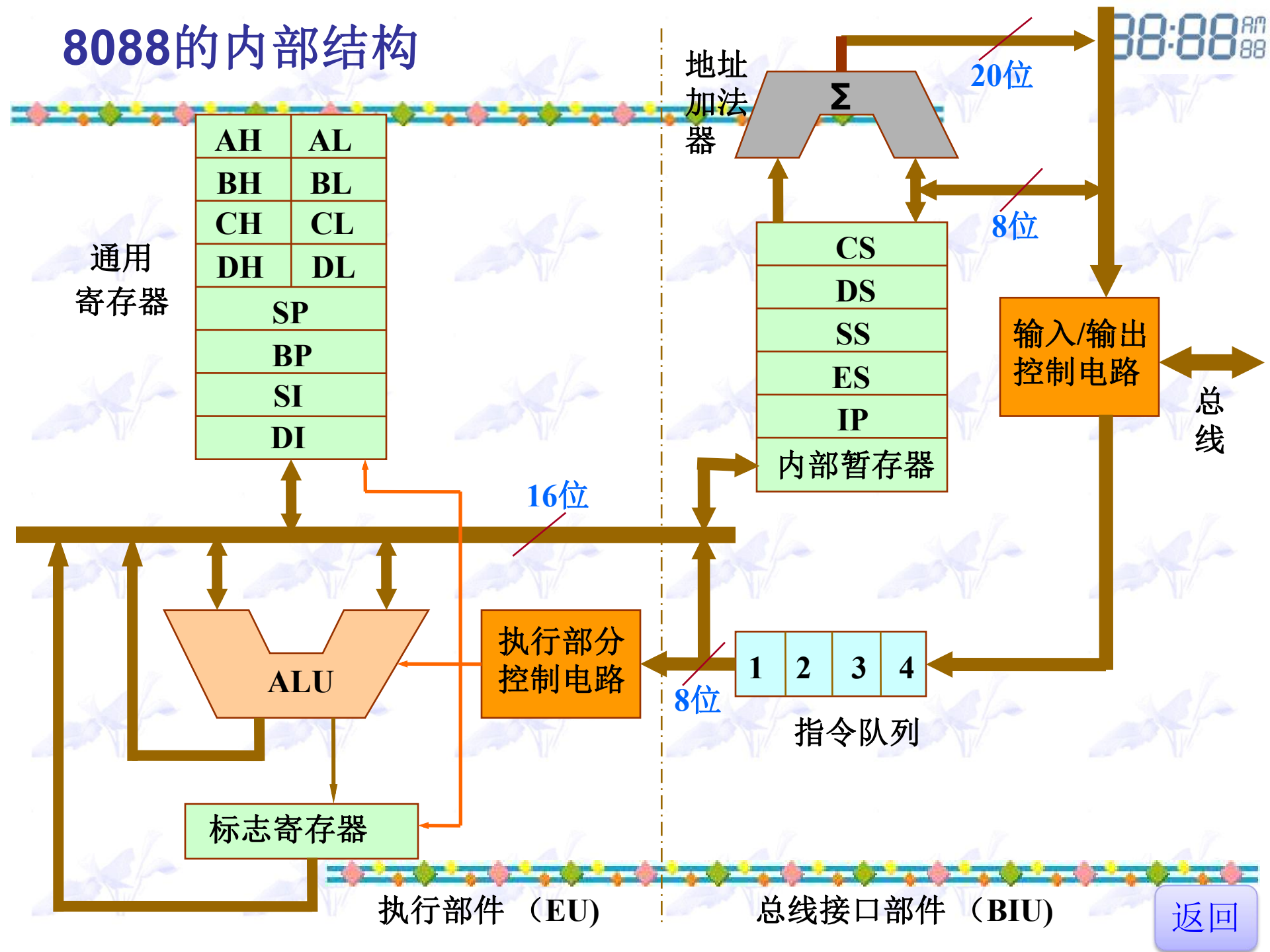
第2章小结

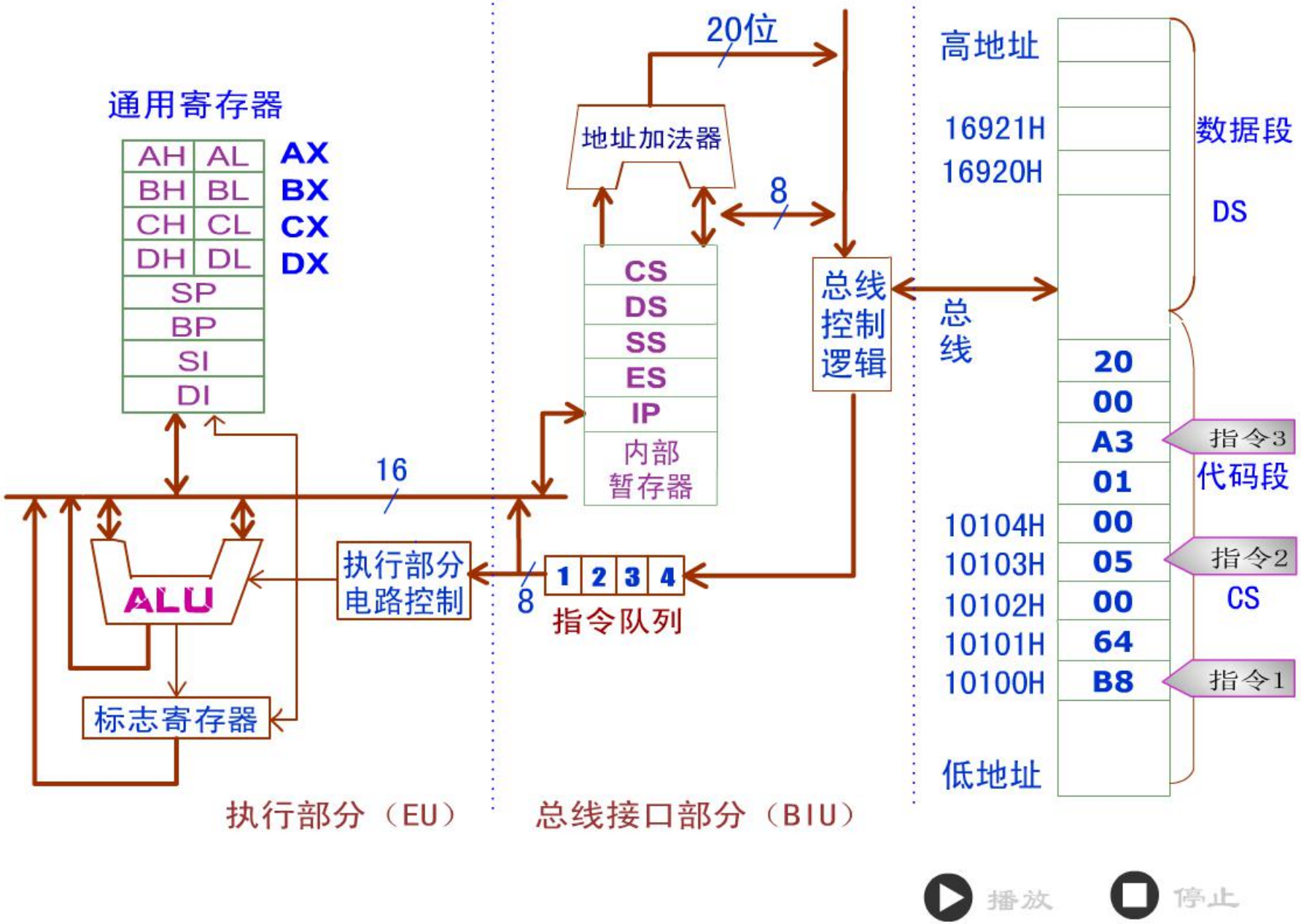
- 8088有8个8位通用寄存器、8个16位通用寄存器
- 8088有6个状态标志和3个控制标志
- 8088将1MB存储空间分段管理，有4个段寄存器，对应4种逻辑段
- 8088有4个段超越前缀指令，用于明确指定数据所在的逻辑段

熟悉上述内容后，就可以进入下节

参考资料

8088的内部结构


地址
加法
器
Σ
20位
8位
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
通用
寄存器
CS
DS
SS
ES
IP
内部暂存器
输入/输出
控制电路
总
线
16位
ALU
执行部分
控制电路
1
2
3
4
8位
指令队列
标志寄存器
执行部件 （EU)
总线接口部件 （BIU)
返回



8088的指令执行过程

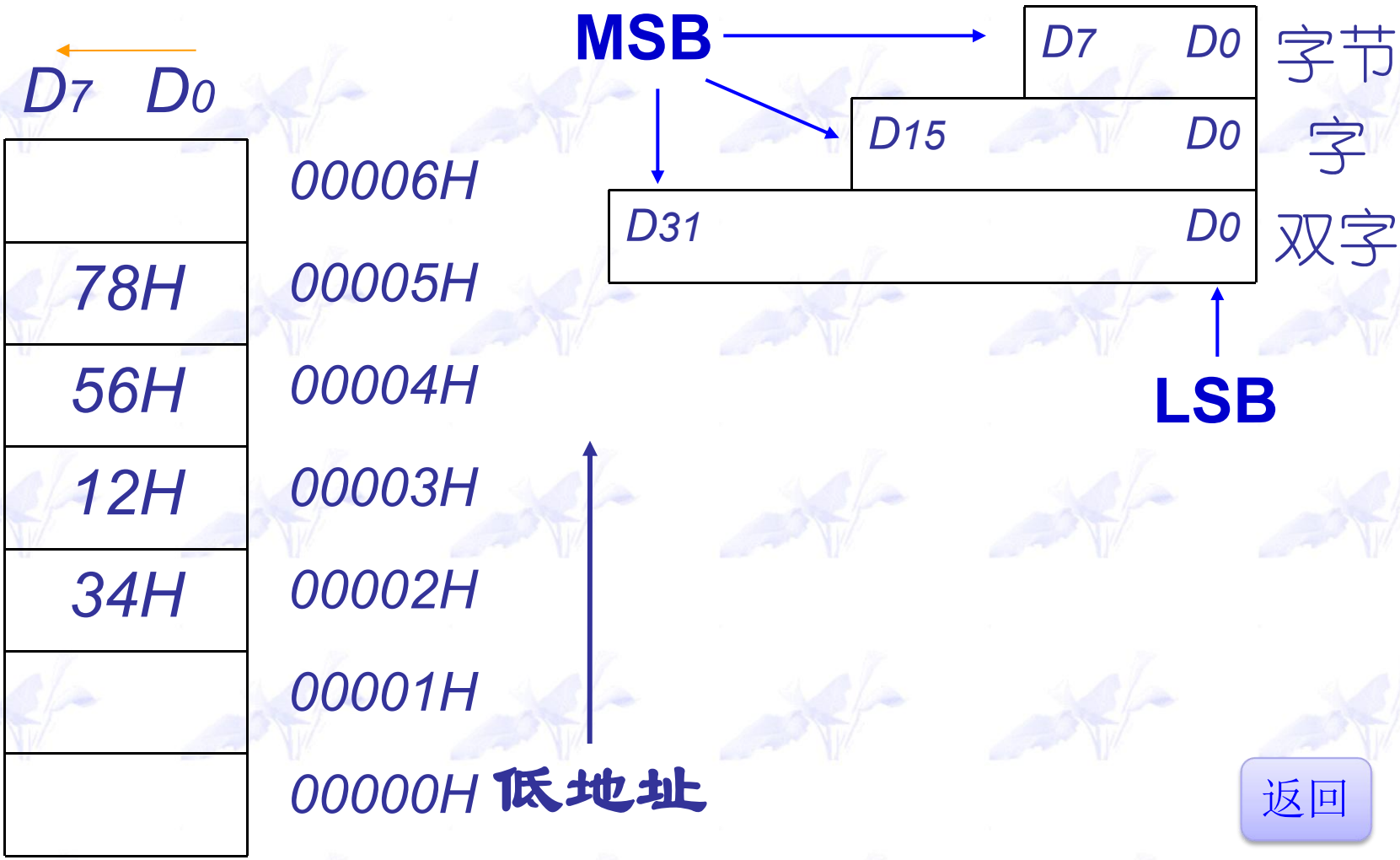
返回

第2章：寄存器、存储器和外存的区别

- 寄存器是微处理器（CPU）内部暂存数据的存储单元，以名称表示，例如：AX，BX……等
- 存储器也就是平时所说的主存，也叫内存，可直接与CPU进行数据交换。主存利用地址区别
- 外存主要指用来长久保存数据的外部存储介质，常见的有硬盘、光盘、磁带、U盘等。外存的数据只能通过主存间接地与CPU交换数据
- 程序及其数据可以长久存放在外存，在运行需要时才进入主存

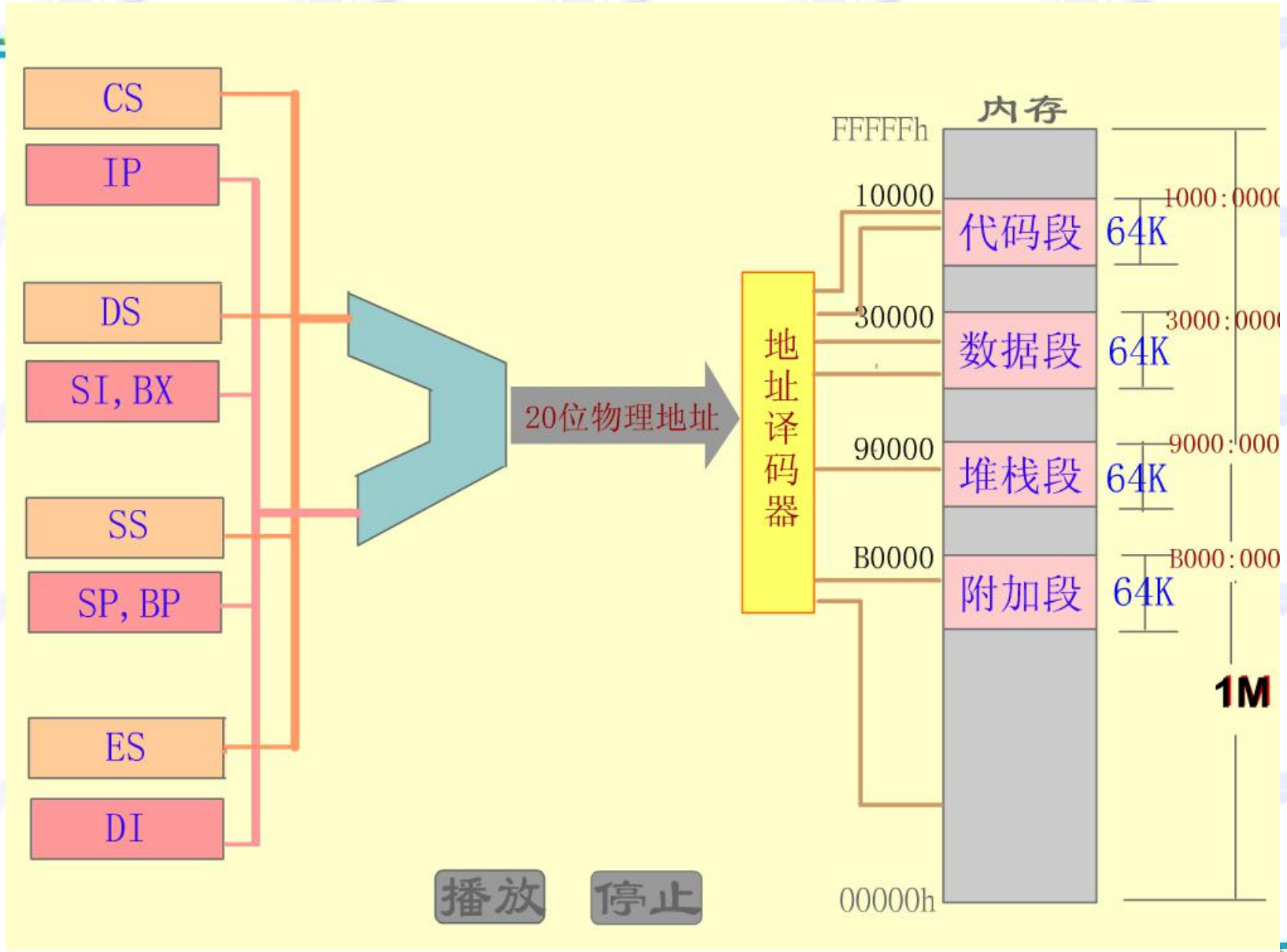
返回

图2-5 8088的存储格式



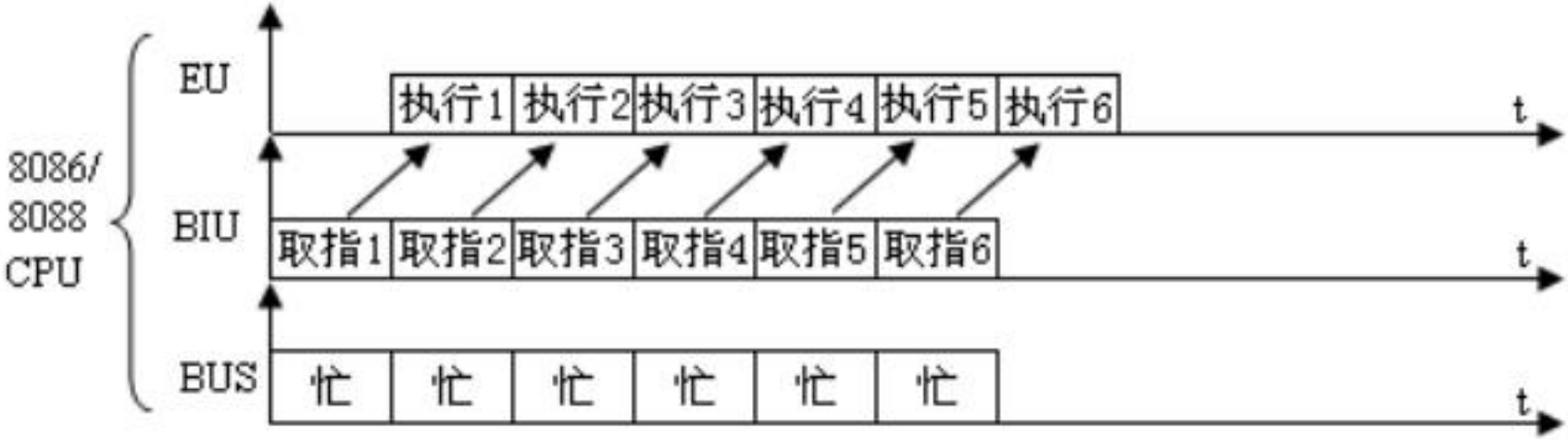
返回

逻辑段的分配


CS
IP
DS
SI, BX
SS
SP, BP
ES
DI
20位物理地址
地址译码器
内存
FFFFFh
10000
30000
90000
B0000
00000h
代码段
数据段
堆栈段
附加段
64K
64K
64K
64K
1000:0000
3000:0000
9000:0000
B000:0000
1M
播放
停止

返回

EU与BIU并行工作的情形



返回