



河南大學  
Henan University

# 汇编语言与接口技术

## ——第 2 章 80x86 微处理器

---

主讲教师：舒高峰

电子邮箱：gaofeng.shu@henu.edu.cn

联系电话：13161693313

# 目录

---

- 01 80x86 微处理器简介
- 02 存储器组织
- 03 寄存器组
- 04 工作模式

# 01 | 80x86简介-Intel微处理器

## 80x86

- 泛指基于英特尔架构 (Intel Architecture) 的各款 x86 微处理器
- 后期还有 Intel Xeon (至强)、Intel Core (酷睿)
- 最大特点: **保持与先前处理器的兼容**

## IA-32 系列处理器

- 1985年, Intel 推出 32 位微处理器 **80386**, 全面支持 **32 位数据类型**和 **32 位操作**
- 支持**实地址方式**和**保护方式**两种工作方式, 为进入 32 位时代做好了充分准备

x	时间	代表产品名称	CPU 字长
1	1978	Intel 8086	8
2	1982	Intel 80286	16
3	1985	Intel 80386	32
4	1989	Intel 80486	32
5	1993	Intel 80586 Pentium (奔腾)	32/64

# 01 | 80x86简介 - 8086内部结构1

## 功能分类

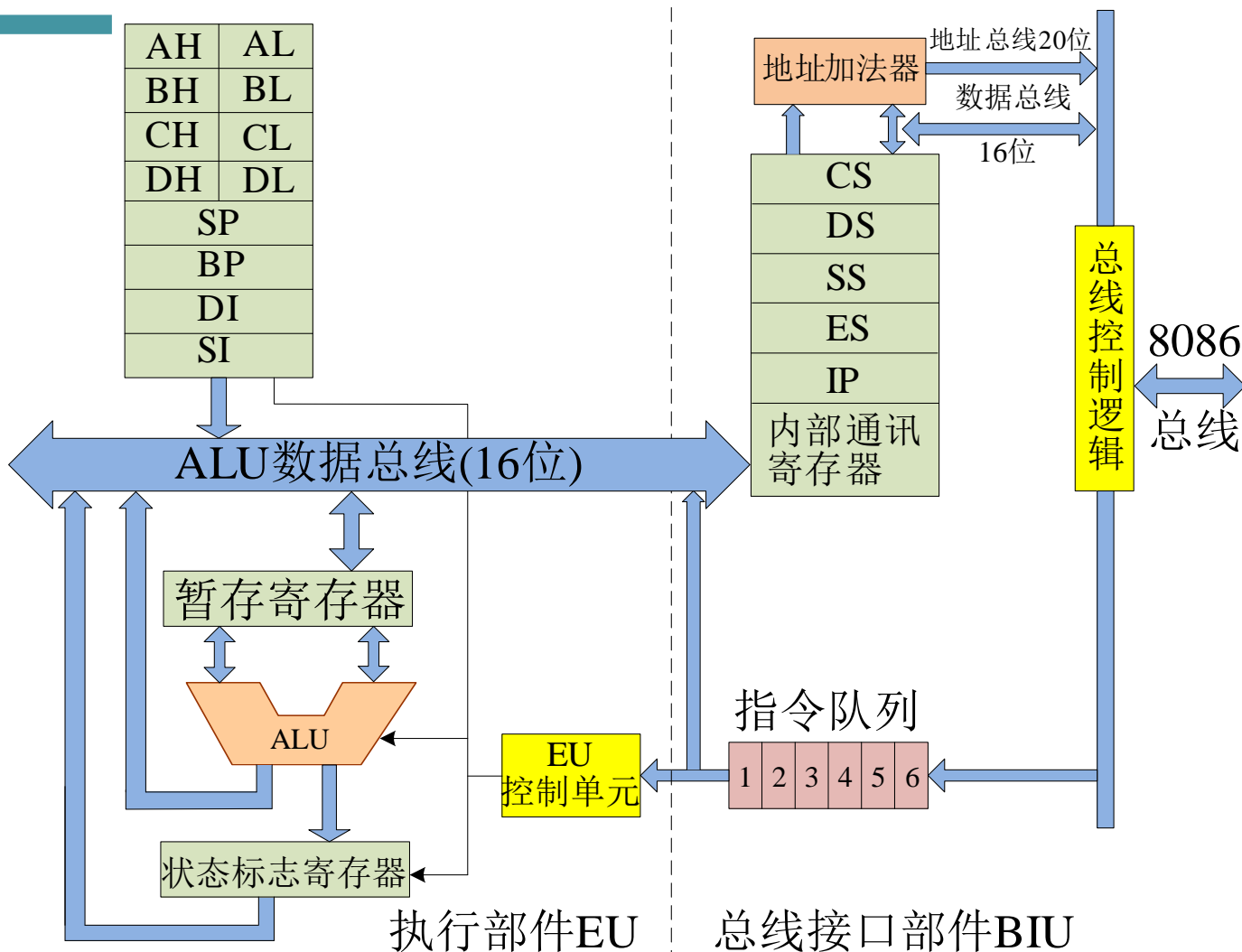
- **运算器**：负责所有的算术逻辑运算
- **控制器**：负责微机系统的所有控制功能

## 结构分类

- **执行单元** (Execution Unit, EU)
  - 部件：**指令译码部件**、**算术逻辑单元** (Arithmetic & Logical Unit, ALU) 和**通用寄存器组**
  - 功能：**负责指令译码和执行**
- **总线接口单元** (Bus Interface Unit, BIU)
  - 部件：**指令队列缓冲器**、**总线控制逻辑**、**专用的寄存器和地址产生器**
  - 功能：**负责 CPU 与外界的通信联络**

# 01 | 80x86简介 - 8086内部结构2

## 8086 内部结构



# 01 | 80x86简介 - 80386内部结构

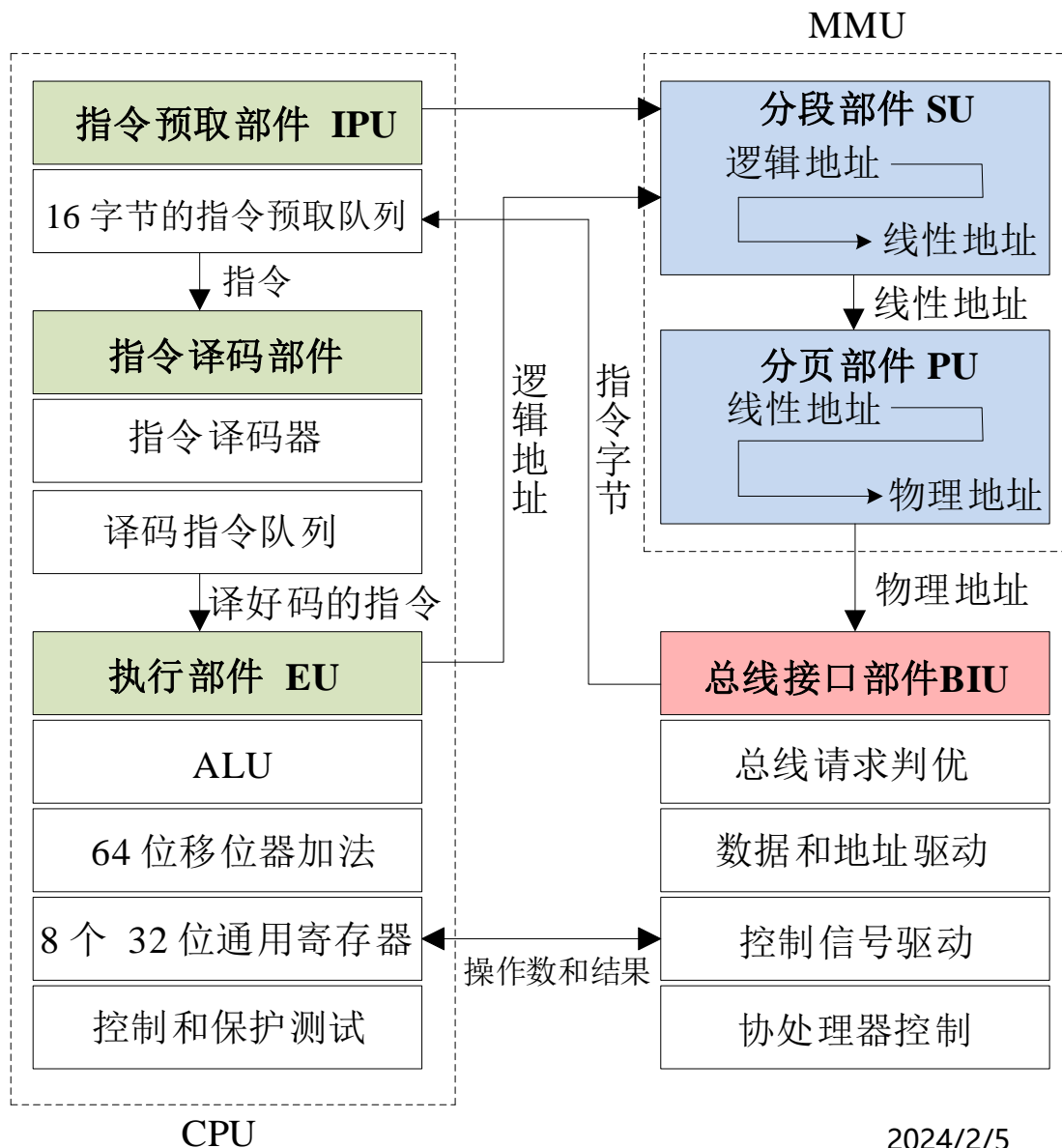
## 80386 内部结构

### ● 三大部件

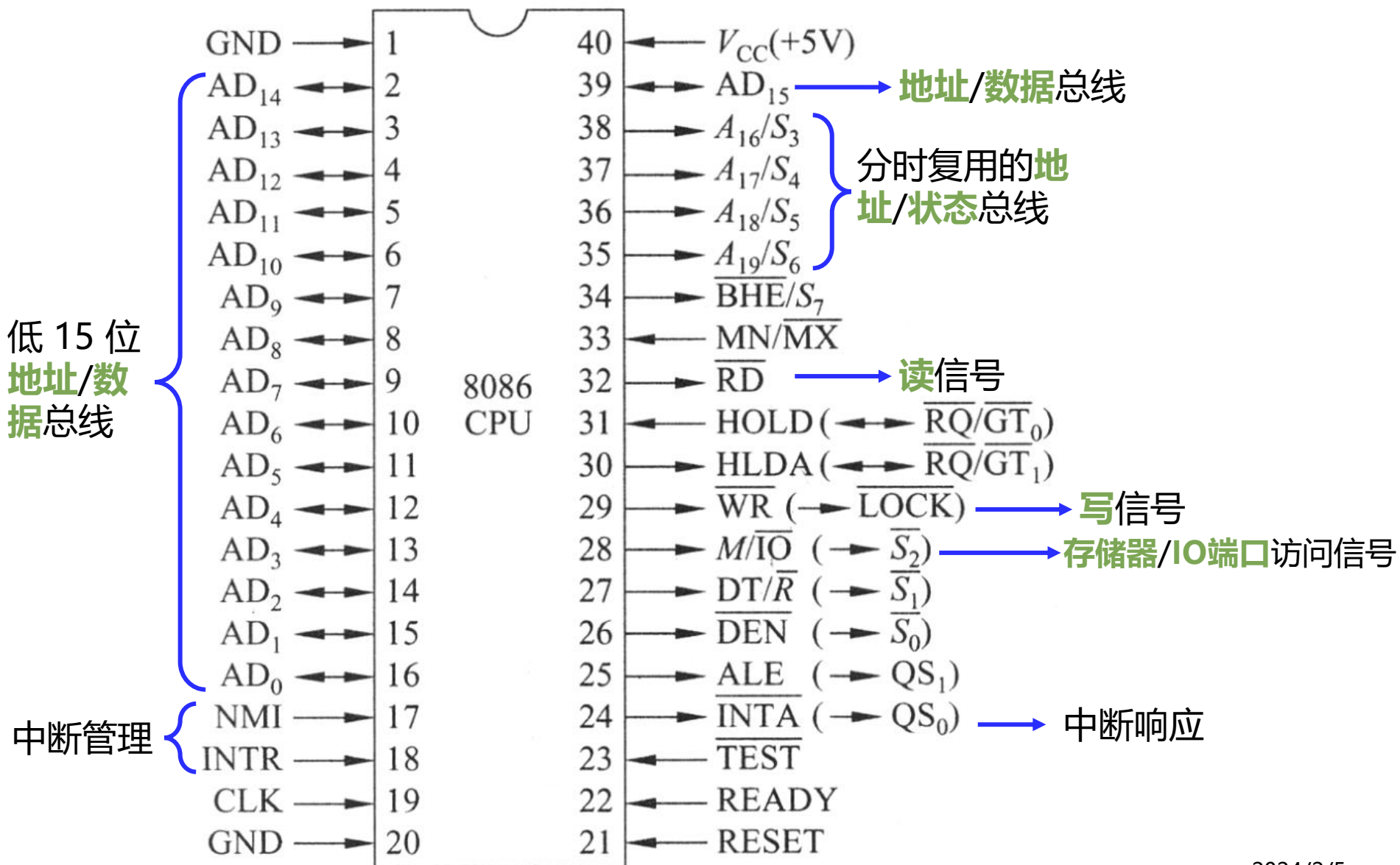
- 总线接口部件
- 中央处理部件
- 存储器管理部件

### ● 六个模块

- 总线接口部件
- 执行部件
- 指令预取部件
- 指令译码部件
- 分段部件
- 分页部件



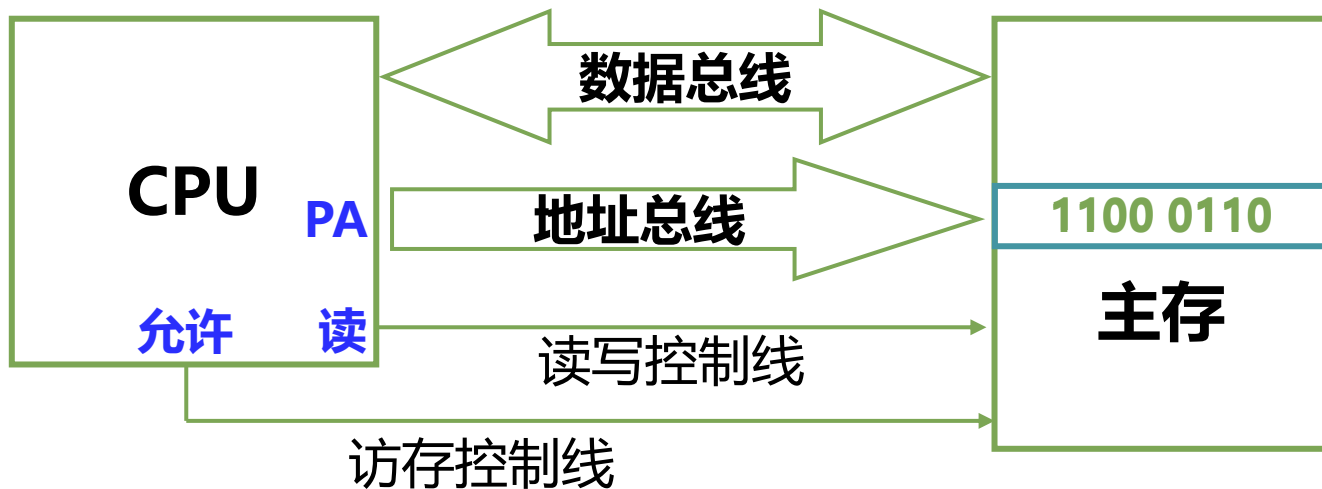
# 01 | 80x86简介-8086管脚



# 01 | 80x86简介-CPU访存过程

## 访存过程

1. CPU 通过**控制总线**，发出**访存信号**，通知主存准备数据读写
2. CPU 通过**地址总线**，发出**存储单元的地址**；
  - ✓ 主存储器接收到地址后，译码，寻址正确的存储单元
3. CPU 通过**控制总线**，发出**读写的命令**；
  - ✓ 主存储器将准备执行读写操作
4. CPU 通过**数据总线**，**读出或写入的数据**；



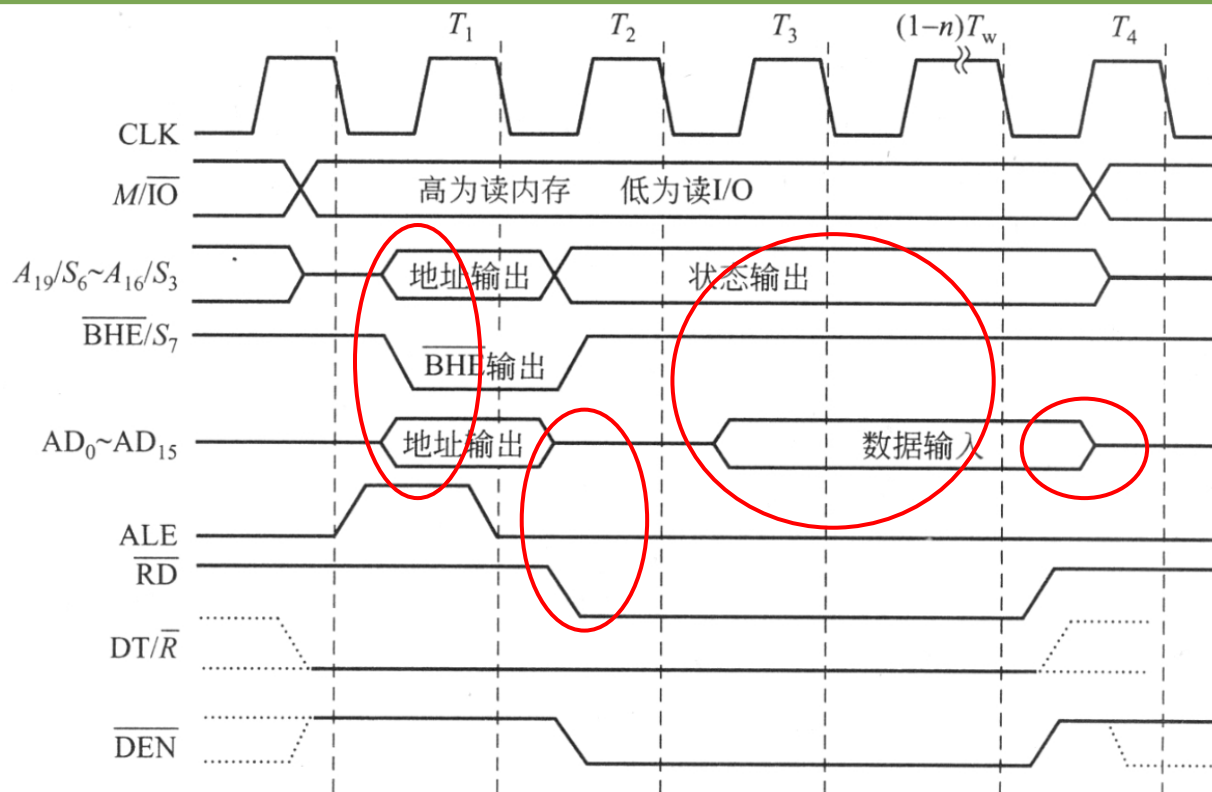
8086 需要 4 个时钟周期；

80486 只需要 1 个时钟周期



# 01 | 80x86简介-CPU读周期时序

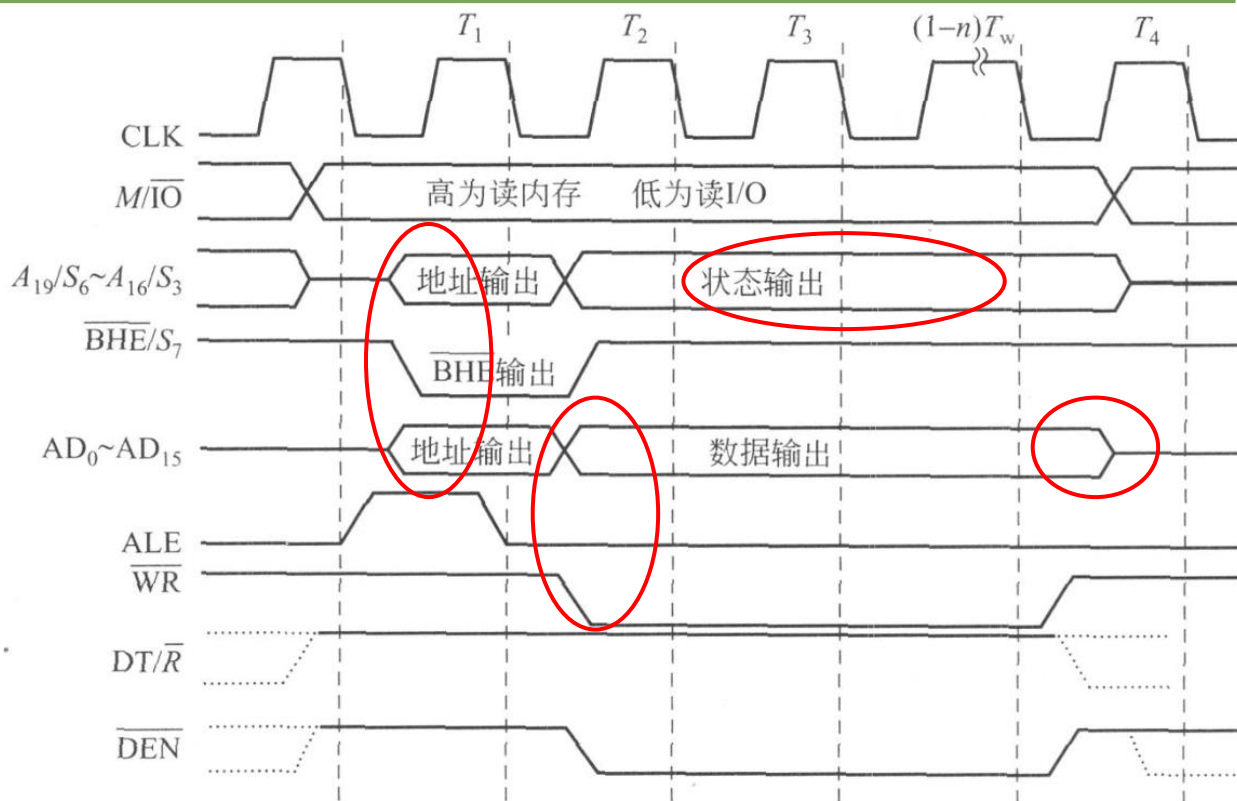
## 读周期时序



1.  $T_1$ : 地址总线输出地址
2.  $T_2$ : 状态总线发出读信号  $\overline{RD}$ , 地址总线改变方向
3.  $T_3$ : 状态总线查询状态, 数据总线得到数据
4.  $T_4$ : CPU 读取数据

# 01 | 80x86简介-CPU写周期时序

## 写周期时序



1.  $T_1$ : 地址总线输出地址
2.  $T_2$ : 状态总线发出写信号  $\overline{WR}$ , 数据总线得到数据
3.  $T_3$ : 状态总线查询状态
4.  $T_4$ : CPU 写入数据

# 目录

---

- 01 80x86 微处理器简介
- 02 存储器组织
- 03 寄存器组
- 04 工作模式

## 02 | 存储器组织-标准结构

### 基本存储单元

- 由 8 个连续的位构成，可用于存储一个字节的数据。所以，**基本存储单元**也被称为**字节存储单元**。

### 存储器

- 由一系列基本存储单元线性地组成，每一个基本存储单元有一个唯一的地址，称为**物理地址**

### 存储器功能

- **存储数据和程序**
- 地址总线对存储器进行寻址，数据总线对存储器内容进行读写

## 02 | 存储器组织-物理地址

### 物理地址 (Physical Address, PA)

- 整个存储器从第一单元到最后一个单元按顺序编号所得到的地址称为**物理地址**，也称**实际地址**
- 物理地址可以**唯一**地标识每一个存储单元
- CPU 访问主存时，必须通过**地址总线**输出所要访问存储单元的的物理地址
- **系统的最大主存容量**取决于**地址总线的位数**

主存储器

	0...00B
	0...01B
	0...10B
.....	.....
	1...11B

# 02 | 存储器组织-存储器分段的原因

## 原因

- 8086 有 20 位地址线，但 CPU 内部地址寄存器却都只有 16 位
- 地址总线
  - 可寻址主存空间为  $2^{20} = 1 \text{ MB}$
  - 物理地址区间  $0 \sim 0\text{FFFFFFH}$
- 数据总线
  - 运算的最大位数、指针等只有 16 位
  - 可直接寻址的空间为  $2^{16} = 64 \text{ KB}$
  - 直接使用的地址区间  $0 \sim 0\text{FFFFH}$

因此，采用分段方式管理和访问主存

# 02 | 存储器组织-存储器分段1

## 分段的思想

- 将存储器划分成**若干区间**，标记起始地址，区间内用较少位数的地址寻址 区间不固定，随机划分！
- 用**两个 16 位地址**合成的方法形成一个 **20 位的物理地址**
  - 段地址、段内偏移地址 (有效地址)

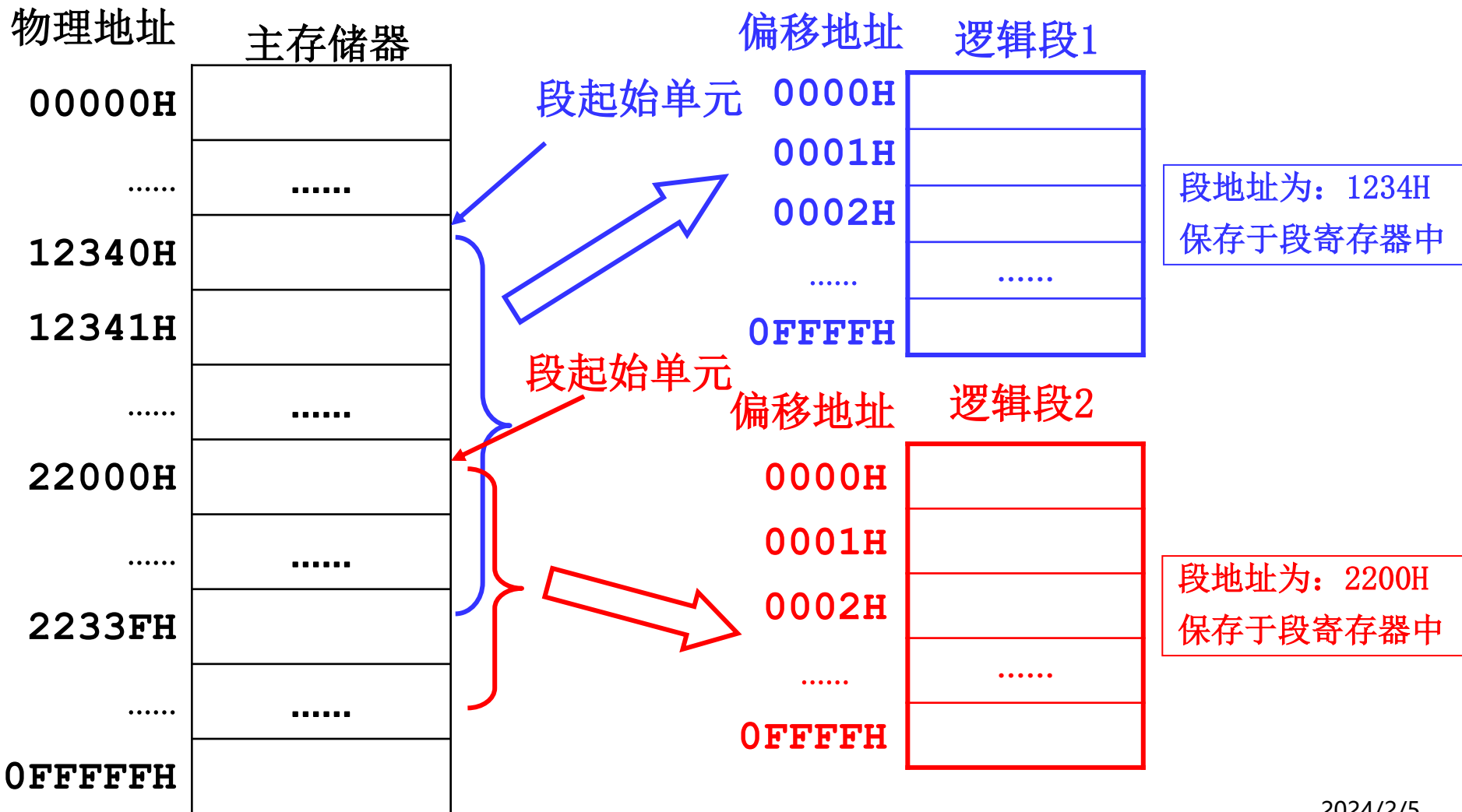
## 分段的规定

- **段的起始**：每个逻辑段的起始地址必须是 16 的倍数，即：**xxxx xxxx xxxx xxxx 0000B** 或：**xxxx0H**
- **段的容量**：每个逻辑段的最大容量可以达到 64 KB。
  - 注意：各逻辑段之间是可以重叠的

主存实际上并没有从物理上分段，段的划分只是来自于 **CPU** 的管理！

# 02 | 存储器组织-存储器分段2

## 分段的示意





# 02 | 存储器组织-地址类型

## 地址类型

某存储单元 A 的物理地址为 23000H

- 物理地址：每个存储单元在整个存储器中的唯一标识

23000H

- 段首地址：逻辑段首单元的物理地址的高16位的可能值

2000H

2100H

.....

- 有效地址：该存储单元相对于段首单元的偏移量

3000H

2000H

.....

- 逻辑地址：由段首地址和有效地址表示的存储单元地址形式

2000H:3000H

2100H:2000H

.....:.....

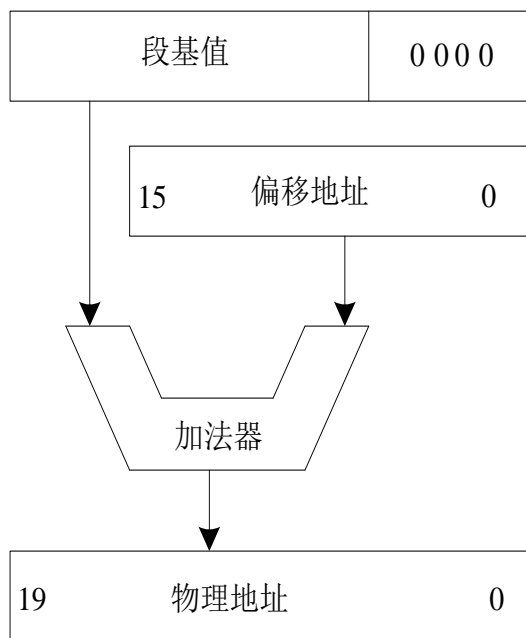
: 为段超越符或段超越前缀，用来改变默认段寻址。左侧给出段首址，右侧给出偏移地址

# 02 | 存储器组织 - 实际地址和逻辑地址

## 物理地址的形成

- 物理地址由两个逻辑地址产生：**段首地址**和**偏移地址**
- 8086 CPU 的实际地址计算公式

$$\text{物理地址} = \text{段首地址} \times 16 + \text{偏移地址}$$



将段首地址左移4个二进制位

一个存储单元只有**唯一**编码的物理地址；  
而由于分段的不同，一个实际地址可对应**多个**逻辑地址

若某数据段段首址为 2000H，偏移地址为 3000H，则该数据的物理地址为\_\_\_\_\_。

- ☐ A 2000H
- ☐ B 3000H
- ☒ C 2300H
- ☐ D 3200H

## 02 | 存储器组织-默认段地址

### 物理地址的形成

- 一般使用**段寄存器**与**指针寄存器**来共同表示逻辑地址；
- 在 CPU 内部由**地址加法器**完成运算，转换成物理地址输出访问主存储器

### 默认段地址

- 代码段段寄存器 **CS**，指针寄存器为 **IP**
- 数据段段寄存器 **DS**，指针寄存器一般用 **BX**、**SI**、**DI**
- 附加段段寄存器 **ES**，指针寄存器一般用 **DI**(字符串操作)。
- 堆栈段段寄存器 **SS**，指针寄存器 **SP** 指向栈顶，指针寄存器 **BP** 指向栈内任意位置

## 02 | 存储器组织-存储器数据的存取方法

### 高高低低原则

- 低地址单元存放低字节数据，高地址单元存放高字节数据

### 示例

- 将一个字节数据 12H 存于 12340H 单元中
- 将一个字数据 3456H 存于 12341H 单元中
- 读取 12342H 单元中的字数据为：1034H

### 注意

- 一般数据存储遵循“数据对齐规则”
  - 字数据存放在偶地址单元

	.....
12340H	12
12341H	56
12342H	34
12343H	10
12344H	20
	.....

# 目录

---

- 01 80x86 微处理器简介
- 02 存储器组织
- 03 寄存器组**
- 04 工作模式

# 03 | 寄存器组-寄存器

## 寄存器

- CPU 内部由若干触发器和逻辑电路组成，用来暂存二进制指令、数据和地址的部件

## 寄存器功能

- 可将寄存器内的数据**执行算术及逻辑运算**
- 寄存器内的数据可用来指向内存的某个位置，即**寻址**
- 可以用来**读写数据**到电脑的周边设备

## 寄存器特点

- 寄存器之间的**数据传送非常快**
- 使用灵活 (如暂存运算的中间数据)、控制方便 (如IP)

# 03 | 寄存器组-寄存器分类

## 寄存器的分类

- **通用寄存器**：传送和暂存数据；参与算数逻辑运算并保存结果
- **段寄存器**：保存段地址，用于寻址时构成物理地址
- **专用寄存器**
  - **指令指针寄存器**：保存将要取出的指令有效地址
  - **状态标志寄存器**：反映处理器的状态和运算结果的某些特征
- **控制寄存器**：用于控制和确定处理器的操作模式以及当前执行任务的特性
- **系统地址寄存器**：为段机制所用的重要表格数据
- **调试寄存器**：用于程序的调试
- **测试寄存器**：用于处理器自测或测试其他设备



# 03 寄存器组 - 80x86寄存器

## ● 8086 寄存器

AH	AL
BH	BL
CH	CL
DH	DL

AX累加器

BX基数

CX计数

DX数据

数据  
寄存器

通用  
寄存器

SP
BP
SI
DI

堆栈

基数

源变址

目的变址

指针  
寄存器

变址  
寄存器

CS
DS
SS
ES

代码段

数据段

堆栈段

附加数据段

段寄存器

IP
FLAGS

指令指针

状态标志

## ● 80386 寄存器

31	16	15	8	7	0	
			AH	AX	AL	EAX
			BH	BX	BL	EBX
			CH	CX	CL	ECX
			DH	DX	DL	EDX
				SP		ESP
				BP		EBP
				SI		ESI
				DI		EDI

通用寄存器组

15	0	
		CS
		DS
		SS
		ES
		FS
		GS

段寄存器组

31	16	15	0	
	VM	RF		标志寄存器 EFLAGS
			IP	EIP

专用寄存器

PG		ET	TS	EM	MP	PE	CR0
							由 Intel 公司指定 CR1
							页故障线性地址 CR2
							页目录基地址 0000 0000 0000 0000 CR3

控制寄存器组

31	15	0	
		界限	
			线性基地址 GDTR

31	15	0	
		界限	
			线性基地址 IDTR

		选择器	LDTR
		选择器	TR

系统地址寄存器组

31	0	
		断点0 DR0
		断点1 DR1
		断点2 DR2
		断点3 DR3
		由 Intel 公司指定 DR4
		由 Intel 公司指定 DR5
		调试状态寄存器 DR6
		调试状态寄存器 DR7

调试寄存器组

31	0	
		由 Intel 公司指定 TR0
		由 Intel 公司指定 TR1
		由 Intel 公司指定 TR2
		由 Intel 公司指定 TR3
		由 Intel 公司指定 TR4
		由 Intel 公司指定 TR5
		TLB 命令寄存器 TR6
		TLB 数据寄存器 TR7

测试寄存器组

# 03 | 寄存器组-通用寄存器1

## 通用寄存器 (General Purpose Registers)

- 通用寄存器不仅能**存储数据**，而且能**参与算术逻辑运算**，还能**给出存储单元的地址**
- IA-32 系列 CPU 通用寄存器名称分别是：EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP
  - 数据寄存器：EAX、EBX、ECX、EDX
  - 指针寄存器：**EBP**、**ESP** (堆栈**基址**和**栈顶**的偏移地址)
  - 变址寄存器：**ESI**、**EDI** (数据的**源变址寄存器**和**目的变址寄存器**)

AX	Accumulator	SI	Source Index
BX	Base	DI	Destination Index
CX	Counter	SP	Stack Pointer
DX	Data	BP	Base Pointer

# 03 | 寄存器组-通用寄存器2

## 通用寄存器 (General Purpose Registers)

	英文名	名称	作用
AX	Accumulator	累加器	常作隐含操作数，可通用
BX	Base	基地址寄存器	常作地址指针，可通用
CX	Counter	计数器	常存放计数值，可通用
DX	Data	数据寄存器	常与累加器配合，可通用
SI	Source Index	源变址寄存器	保存源操作数地址
DI	Destination Index	目的变址寄存器	保存目的操作数地址
SP	Stack Pointer	栈顶指针	只能保存堆栈栈顶地址
BP	Base Pointer	堆栈指针	可保存堆栈任意位置地址

# 03 | 寄存器组-通用寄存器3

## 通用寄存器 (General Purpose Registers)

- IA-32 系列 CPU 可以单独直接访问这些通用寄存器的低 16 位

	31	16	15	0
EAX				AX
EBX				BX
ECX				CX
EDX				DX
ESI				SI
EDI				DI
EBP				BP
ESP				SP

# 03 | 寄存器组-通用寄存器之数据寄存器1

## 数据寄存器

- 8086 CPU 也可以单独直接访问这些数据寄存器的低 8 位

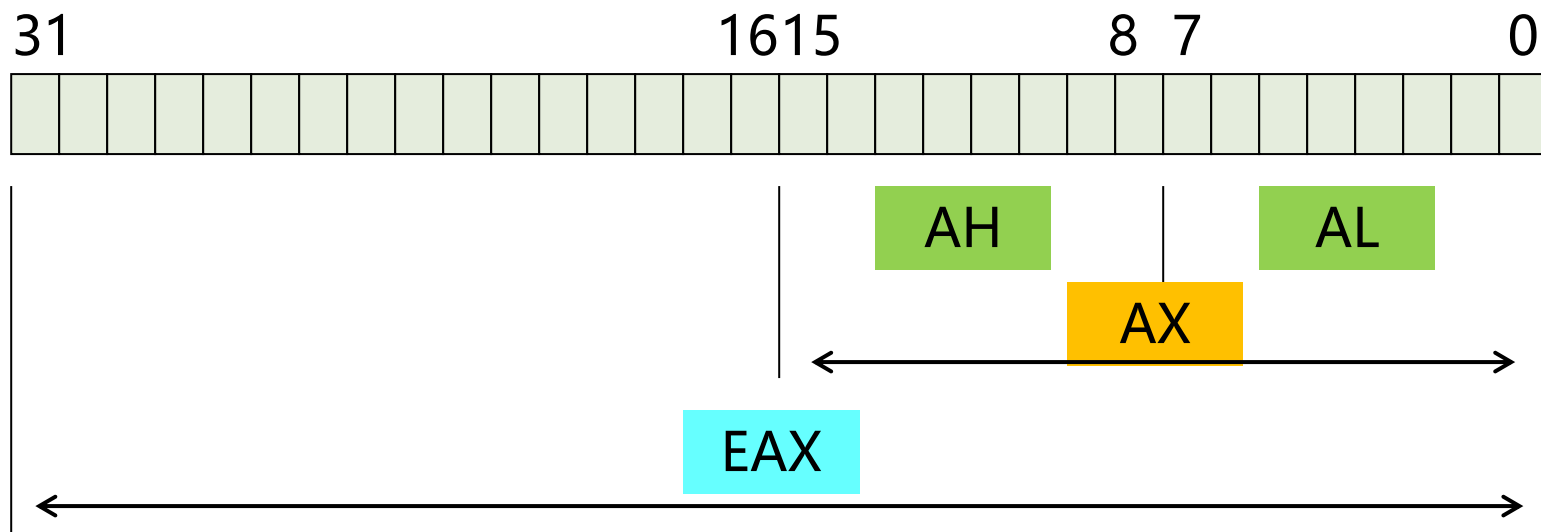
	31	16	15	8	7	0	
EAX			AH			AL	AX
EBX			BH			BL	BX
ECX			CH			CL	CX
EDX			DH			DL	DX
ESI						SI	
EDI						DI	
EBP						BP	
ESP						SP	

有名称的通用寄存器，  
可以独立访问；  
否则，不行

## 03 | 寄存器组-通用寄存器之数据寄存器2

### 数据寄存器命名规则

- 32 位通用寄存器的名称是在对应 16 位寄存器名称前加字母 **E**
- AH 是 AX 的高 (**High**) 字节; AL 是 AX 的低 (**Low**) 字节
- **EAX** 是 AX 的扩展 (64 位寄存器用 **RAX** 表示)



## 03 | 寄存器组-通用寄存器之变址寄存器

变址寄存器 SI (Source Index)、DI (Destination Index)

- 常作为指针，存放存储单元有效地址，也可暂存数据

### 特殊用法

- SI、DI 中保存的地址信息可以随着指令的执行而自动改变
- SI，串操作中存放源串地址，默认 DS 段
- DI，串操作中存放目的串地址，默认 ES 段
- 该特殊用法只在字符串操作中有效，其它场合下作一般的指针寄存器使用

# 03 | 寄存器组-通用寄存器之堆栈指针寄存器

## 堆栈 (Stack)

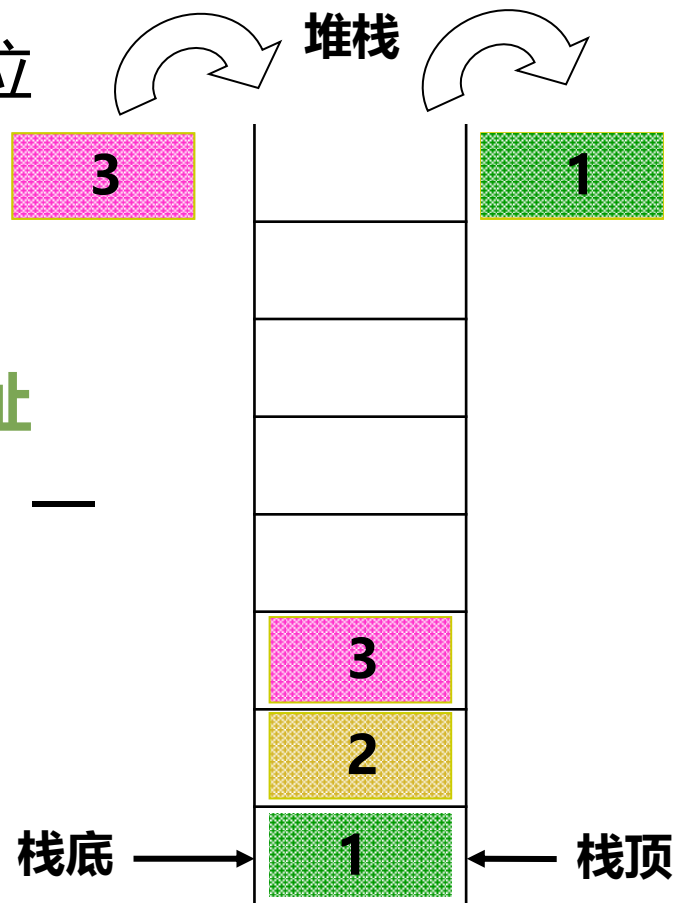
- 一个**先进后出**的数据结构，栈底位置不变

## 栈顶指针 SP (Stack Pointer)

- 其中始终存放**栈顶单元的有效地址**
- 其值是**由出入栈指令自动更改**的，一般不允许随意对该寄存器赋值

## 栈底指针 BP (Base Pointer)

- 其中数据一般作为地址进行访存
- **默认对应于 SS 段**，可寻址堆栈中的任何单元





# 03 | 寄存器组-通用寄存器示例1

## 示例

```
1. MOV EAX, 12345678H ;EAX=12345678H
2. MOV ESI, 11223344H ;ESI=11223344H
3. ADD EAX, ESI        ;EAX=235689BCH
4. MOV EBX, EAX        ;EBX=235689BCH
5. MOV ECX, [ESI]      ;ESI作为指针给出存储单元地址
6. MOV EDX, [EBX+8]    ;EBX作为计算存储单元地址的一部分
```

- 存储数据 (1, 2, 3, 4, 5, 6)
- 参与算术逻辑运算 (3)
- 给出存储单元的地址 (5, 6)

## 03 | 寄存器组-通用寄存器示例2

### 示例

```
1. MOV EAX, 11112222H      ;EAX=11112222H
2. MOV AX, 9999H           ;EAX=11119999H
3. MOV EDX, EAX            ;EDX=11119999H
4. MOV DX, 8765H           ;EDX=11118765H
5. ADD AX, DX              ;EAX=111120FEH
```

- 对 32 位寄存器低 16 位独立操作，不影响高 16 位

```
1. MOV EBX, 11112222H      ;EBX=11112222H
2. MOV BH, 77H             ;EBX=11117722H
3. MOV BL, 99H             ;EBX=11117799H
4. ADD BL, 82H             ;EBX=1111771BH
```

- 对 16 位寄存器低 8 位独立操作，不影响高 8 位

# 03 | 寄存器组 - 段寄存器

## 段寄存器 (Segment Registers)

- 用来存放**段首地址**或**段选择符**的 16 位寄存器
- 一段汇编语言程序**至少有一个逻辑段**——代码段，用于存放代码

类别	16 位	段寄存器名称	作用	
			8086	IA-32
段寄存器	CS	代码段	存放段首地址	存放段选择符
	DS	数据段	存放段首地址	存放段选择符
	SS	堆栈段	存放段首地址	存放段选择符
	ES	附加段	存放段首地址	存放段选择符
	FS	数据段	无	存放段选择符
	GS	数据段	无	存放段选择符

# 03 | 寄存器组 - 段寄存器之代码段

## 代码段 CS (Code Segment)

- 用来存放要执行的**指令序列**
- **段首地址**用代码段寄存器 CS 来保存
- **指令指针寄存器 IP** 指示本段中的地址
  - 将要执行的下条指令的有效地址
- CPU 利用 **CS:IP** 形成存储单元的物理地址，以获取下条要执行指令的代码。

: 为段超越符或段超越前缀，  
用来改变默认段寻址。左侧给  
出段首址，右侧给出偏移地址

16 位 CPU 在取指令时，需要用到的寄存器有\_\_\_\_和 \_\_\_\_;

☐ A AX

☐ B BP

☒ C CS

☒ D IP

## 03 | 寄存器组 - 段寄存器之数据段

### 数据段 DS (Data Segment)

- 用来存放程序运行所需要的数据
- 段首地址用数据段寄存器 DS 来保存
- CPU 利用 DS:EA 形成存储单元的物理地址，以获取数据段中的数据
  - EA 的形成方式详见第 3 章寻址方式的介绍

## 03 | 寄存器组 - 段寄存器之堆栈段

### 堆栈段 SS (Stack Segment)

- 用于存储程序运行中需要临时保护的数据
- 段首地址用堆栈段寄存器 SS 来保存
- 堆栈指针寄存器 SP 保存堆栈栈顶的有效地址
- CPU 利用 SS:SP 对堆栈栈顶单元进行操作  
利用 SS:BP 对堆栈中的任一单元进行操作

## 03 | 寄存器组 - 段寄存器之附加段

### 附加段 ES (Extra Segment)

- 即**附加的数据段**，保存程序运行所需要的数据
- **段首地址**用附加段寄存器 ES 来保存
- CPU 利用 **ES:EA** 形成存储单元的物理地址，以获取附加段中的数据
- **串操作指令**常将附加段 ES 作为目的操作数的存放区域



# 03 | 寄存器组 - 段寄存器之FS、GS

## 数据段 FS、GS

- IA-32 系列额外的数据段寄存器

## IA-32 CPU 段寄存器

- IA-32 CPU 段寄存器不直接给出段首址，而是作为**选择符 (Selector)**，选择对应**段描述符 (Descriptor)** 寄存器
- 段描述符寄存器由**系统地址寄存器**给出，包括段首址、段界限和其他特权属性等信息

段寄存器		描述符寄存器(自动装入)									
15	0	物理基地址	段界限	描述符的其他段属性							
选择符	CS										
选择符	SS										
选择符	DS										
选择符	ES										
选择符	FS										
选择符	GS										

## 03 | 寄存器组 - 段寄存器相关概念

### 段描述符 (Descriptor)

- 为了实现分段管理，把有关段的信息（段的**基址**、段的**界限**和段的**属性**）存放在一个 8 字节长的数据结构中，该数据结构称为**段描述符**

### 段描述符表

- 系统把有关的段描述符放在一起编成表，以便硬件的查找和识别

### 段选择符 (Selector)

- 也称**段选择子**。在保护方式下，段寄存器的内容不再直接表示为段基址，而是一个地址指针，它被称为 16 位的段选择子。

# 03 | 寄存器组 - 段寄存器之默认选择规定

## 默认段寄存器

- 为了简化指令系统，8086 的指令在形式上只给出了地址偏移值 (有效地址)，而隐含着对某段寄存器的操作
- 部分指令可以被段超越前缀改变

访存类型	默认段寄存器	段超越前缀的可用性
代码、指令	CS	不可用
PUSH、POP 类代码	SS	不可用
串操作的目标地址	ES	不可用
以 BP、SP 间址的指令	SS	可用 CS、DS、ES
其他	DS	可用 CS、SS、ES

## 03 | 寄存器组-专用寄存器

### 指令指针寄存器 EIP (Instruction Pointer)

- 保存将要执行指令的有效地址
- 该寄存器的内容是不允许人为更改的，通过指令的执行而自动改变。

### 状态标志寄存器 EFLAGS (Status Flags)

- 该寄存器是利用其中的每一位来反映当前 CPU 执行指令的结果或控制指令执行形式

16 位	32 位	名称	作用
IP	EIP	指令指针寄存器	保存将要取出的指令有效地址
FLAGS	EFLAGS	标志寄存器	保存 CPU 当前的状态标志信息

## 03 | 寄存器组-指令指针寄存器1

### 指令指针寄存器 EIP (Instruction Pointer)

- IA-32 系列 CPU 有一个 **32 位**的指令指针寄存器 **EIP**，它是由 8086 CPU 指令指针寄存器 **IP** 的扩展
- 由 **CS** 和 **EIP** 确定所取指令的存储单元地址
  - 段寄存器 CS 给出当前代码段的段号，指令指针寄存器 **EIP** 给出偏移
- 如果代码段起始地址是 0，则 **EIP** 给出的偏移，或者说有效地址，直接决定所取指令的存储单元地址
- 实方式下，段的最大范围是 64K，**EIP** 中高 16 位必须是 0，相当于只有低 16 位的 **IP** 起作用

# 03 | 寄存器组-指令指针寄存器2

## 顺序执行指令

- CPU 执行代码 (程序) 就是一条接一条地取机器指令、执行机器指令的过程
- 在取出一条指令后, 会根据所取指令的长度, **自动调整指令指针寄存器 EIP 的值**, 使其指向下一条指令。这样, 就实现了**顺序执行指令**

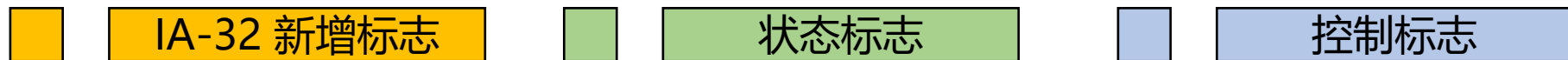
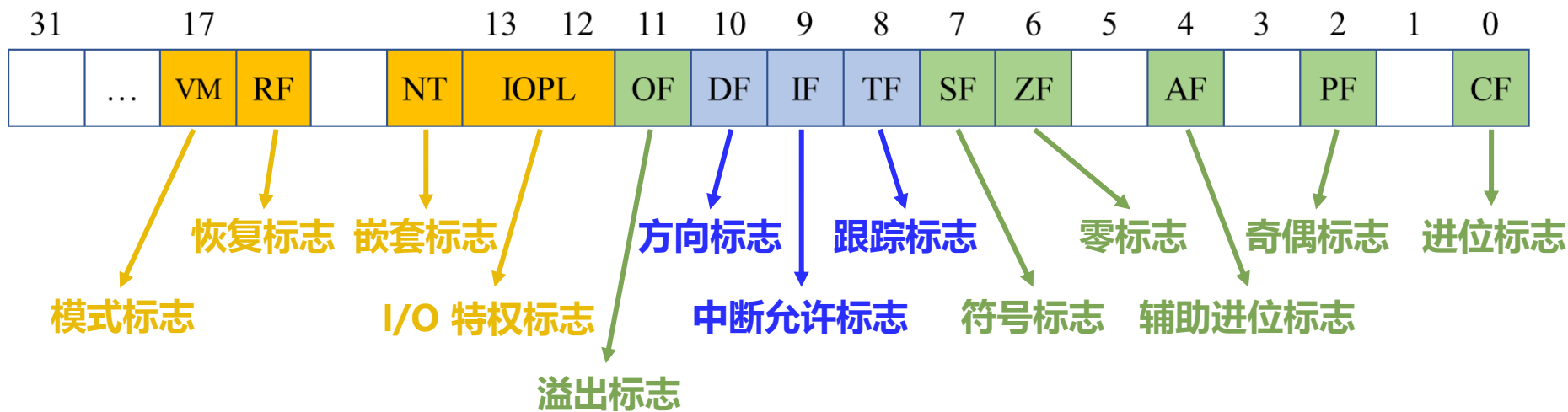
## 跳转执行指令

- 所谓**跳转**或**转移**指, **非自动顺序调整 EIP 内容**
- 各种**控制转移指令**用于根据不同的情形改变 EIP 内容, 从而实现**转移执行指令**

# 03 | 寄存器组-标志寄存器

## 状态标志寄存器 EFLAGS (Status Flags)

- 一个 32 位的寄存器，用于反映处理器的状态和运算结果的某些特征，低 16 位对应 8086 的 FLAGS 寄存器
- 可以暂时认为主要是，状态标志和控制标志



## 03 | 寄存器组 - 控制标志和新增标志

### 控制标志位

- DF (Direction Flag): 用于控制串操作指令的标志
- IF (Interrupt Flag): 用于控制可屏蔽中断的标志
- TF (Trap Flag): 跟踪标志, 又称单步标志

### IA-32 新增标志位

- IOPL (I/O Privilege Level): I/O 特权标志位 (2 位)
- NT (Nested Task Flag): 用于显示当前任务是否嵌套于另一任务
- RF (Resume Flag): 暂时禁用调试异常
- VM (Virtual 8086 Mode): 是否工作于虚拟 8086 模式



# 03 | 寄存器组-状态标志

## 状态标志位

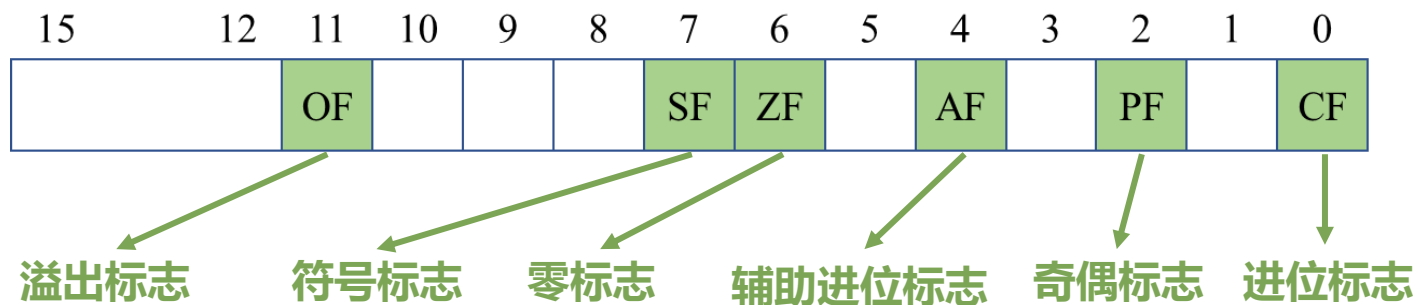


标志位=1



标志位=0

- CF (Carry Flag): 运算结果最高位**是否**产生进位或借位
- PF (Parity Flag): 运算结果低 8 位 1 的个数**是否**为偶数
- AF (Auxiliary Carry Flag): 运算结果低 4 位**是否**产生进位或借位
- ZF (Zero Flag): 当前结果**是否**为零
- SF (Sign Flag): 运算结果的最高位 (**正负**符号位)
- OF (Overflow Flag): 运算过程**是否**产生溢出



## 03 | 寄存器组-零标志


### 零标志 ZF (Zero Flag)

- 当运算结果为 0 时，设置  $ZF=1$ ；否则  $ZF=0$ 
  - ZF 与运算结果总有一个为 0
- 一般，零标志 ZF 总有效
  - 只关心运算结果的状态

### 举例

- $3AH + 7CH = 0B6H$ ,  $ZF = 0$
- $84H + 7CH = 00H$ ,  $ZF = 1$

$$\begin{array}{r} 0011\ 1010B \\ +\ 0111\ 1100B \\ \hline 1011\ 0110B \end{array}$$

$$\begin{array}{r} 1000\ 0100B \\ +\ 0111\ 1100B \\ \hline 10000\ 0000B \end{array}$$


$3AH + 7CH = \underline{\hspace{2cm}}$ ;  $84H + 7CH = \underline{\hspace{2cm}}$ ;  
计算上述两个式子计算结果对标志位 ZF 的影响

A 0 0

B 0 1

C 1 0

D 1 1

提交

## 03 | 寄存器组-符号标志


### 符号标志 SF (Sign Flag)

- 当运算结果为负数时，设置  $SF=1$ ；否则  $SF=0$ 
  - 运算结果的符号位 = 符号标志 SF 的状态
- 一般，符号标志 SF 总有效
  - 只关心运算结果的**正负**

### 举例

- $3AH + 7CH = 0B6H$ ,  $SF = 1$
- $84H + 7CH = 00H$ ,  $SF = 0$

$$\begin{array}{r} 0011\ 1010B \\ +\ 0111\ 1100B \\ \hline 1011\ 0110B \end{array}$$

$$\begin{array}{r} 1000\ 0100B \\ +\ 0111\ 1100B \\ \hline 10000\ 0000B \end{array}$$


## 03 | 寄存器组-奇偶标志


### 奇偶标志 PF (Parity Flag)

- 当运算结果低 8 位 1 的个数为偶数时，设置  $PF=1$ ；否则  $PF=0$

### 举例

- $3AH + 7CH = 0B6H, PF = 0$
- $84H + 7CH = 00H, PF = 1$

$$\begin{array}{r} 0011\ 1010B \\ +\ 0111\ 1100B \\ \hline 1011\ 0110B \end{array}$$

$$\begin{array}{r} 1000\ 0100B \\ +\ 0111\ 1100B \\ \hline 10000\ 0000B \end{array}$$


## 03 | 寄存器组-辅助进位标志

### 辅助进位标志 AF (Auxiliary Carry Flag)

- 加减运算时，运算结果的**低 4 位**有进位或借位，设置 **AF=1**；否则 **AF=0**
  - 加法时，产生辅助**进位**标志；减法时，产生辅助**借位**标志

### 举例

- $3AH + 7CH = 0B6H$ , **AF = 1**
- $82H + 7CH = 0FEH$ , **AF = 0**

$$\begin{array}{r} 0011\ 1010B \\ +\ 0111\ 1100B \\ \hline 1011\ 0110B \end{array}$$

$$\begin{array}{r} 1000\ 0010B \\ +\ 0111\ 1100B \\ \hline 1111\ 1110B \end{array}$$

## 03 | 寄存器组-进位标志


### 进位标志 CF (Carry Flag)

- 加减运算时，运算结果的**最高位**有进位或借位，设置  $CF=1$ ；否则  $CF=0$ 
  - 加法时，CF 称为**进位标志**；减法时，CF 称为**借位标志**

### 举例

- $3AH + 7CH = 0B6H, CF = 0$
- $0AAH + 7CH = 26H, CF = 1$

$$\begin{array}{r} 0011\ 1010B \\ +\ 0111\ 1100B \\ \hline 1011\ 0110B \end{array}$$

$$\begin{array}{r} 1010\ 1010B \\ +\ 0111\ 1100B \\ \hline 10010\ 0110B \end{array}$$


## 03 | 寄存器组-溢出标志1

### 溢出标志 OF (Overflow Flag)


- 运算结果超出了数据表示范围，设置 OF=1；否则 OF=0
  - 溢出的判断：正+正=负；负+负=正
- OF 主要针对补码表示的数据，即有符号数；无符号数可用 CF 兼作溢出标志

### 举例

- $3AH + 7CH = 0B6H$ , OF = 1 正+正=负
- $0AAH + 7CH = 26H$ , OF = 0 负+正=正

$$\begin{array}{r} 0011\ 1010B \\ +\ 0111\ 1100B \\ \hline 1011\ 0110B \end{array}$$

$$\begin{array}{r} 1010\ 1010B \\ +\ 0111\ 1100B \\ \hline 10010\ 0110B \end{array}$$





## 03 | 寄存器组-溢出标志2

### 机器数的范围

- 8 位表达的范围是:  $-128 \sim +127$  ( $-2^7 \sim +2^7-1$ )
- 16 位表达的范围是:  $-32768 \sim +32767$  ( $-2^{15} \sim +2^{15}-1$ )

### 溢出

- 如果运算结果超出这个范围，就产生了溢出，则表示该运算结果不正确！
- 简单的溢出的判断规则：**正+正=负**； **负+负=正**

## 03 | 寄存器组 - 溢出和进位的区别

### 进位标志 CF

- 其设置完全根据二进制数据的计算情况设置
- 表示无符号数的运算结果是否超出范围
- 无论 CF 为何值，无符号数的运算结果均正确

### 溢出标志 OF

- 其设置是把数据看作有符号数来判断的
- 表示有符号数运算结果是否超出范围
- 当 OF=1 时，有符号数的运算结果不正确

溢出标志 OF 和进位标志 CF 是两个意义不同的标志位

# 03 寄存器组-状态标志寄存器示例

## 示例

1.	MOV EAX, 77009966H	;EAX = 77009966	(十六进制)						
2.	MOV EDX, 55440000H	;EDX = 55440000		Z	S	P	A	C	O
3.	ADD EAX, EDX	;EAX = CC449966		0	1	1	0	0	1
4.	ADD EDX, EAX	;EDX = 21889966		0	0	1	0	1	0
5.	ADD AX, AX	;EAX = CC4432CC		0	0	1	0	1	1
6.	ADD AL, 6	;EAX = CC4432D2		0	1	1	1	0	0
7.	ADD AL, 52H	;EAX = CC443224		0	0	1	0	1	0
8.	ADD AX, 0CDDCH	;EAX = CC440000		1	0	1	1	1	0
9.	ADD EAX, 33BC0000H	;EAX = 00000000		1	0	1	0	1	0

- 除了 AF，其他标志位均根据**当前运算的结果**改变标志位的值
  - 若参与运算是 8 位，则根据 8 位结果改变标志
  - 若参与运算是 16 位，则根据 16 位结果改变标志
  - 若参与运算是 32 位，则根据 32 位结果改变标志

# 目录

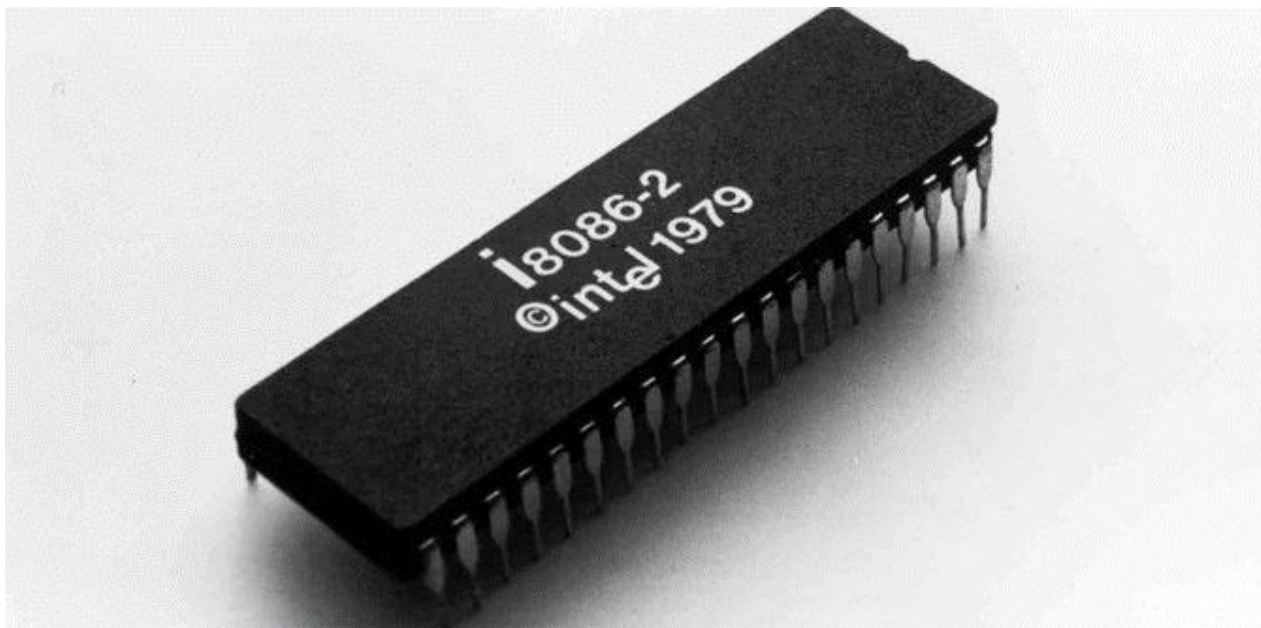
---

- 01 80x86 微处理器简介
- 02 存储器组织
- 03 寄存器组
- 04 工作模式

# 04 | 工作模式-8086工作模式

## 工作模式

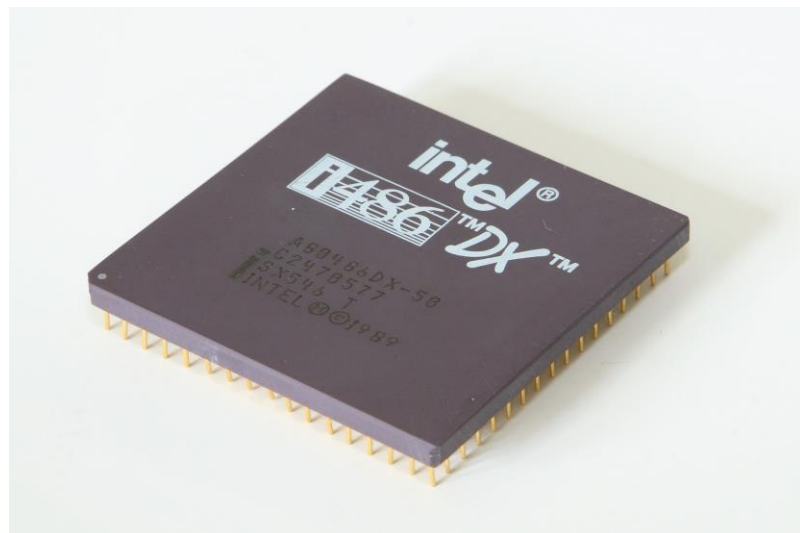
- **最小模式**：在系统中只有一个微处理器
- **最大模式**：两个或多个微处理器 (主处理器、协处理器)



# 04 | 工作模式 - 80486工作模式1

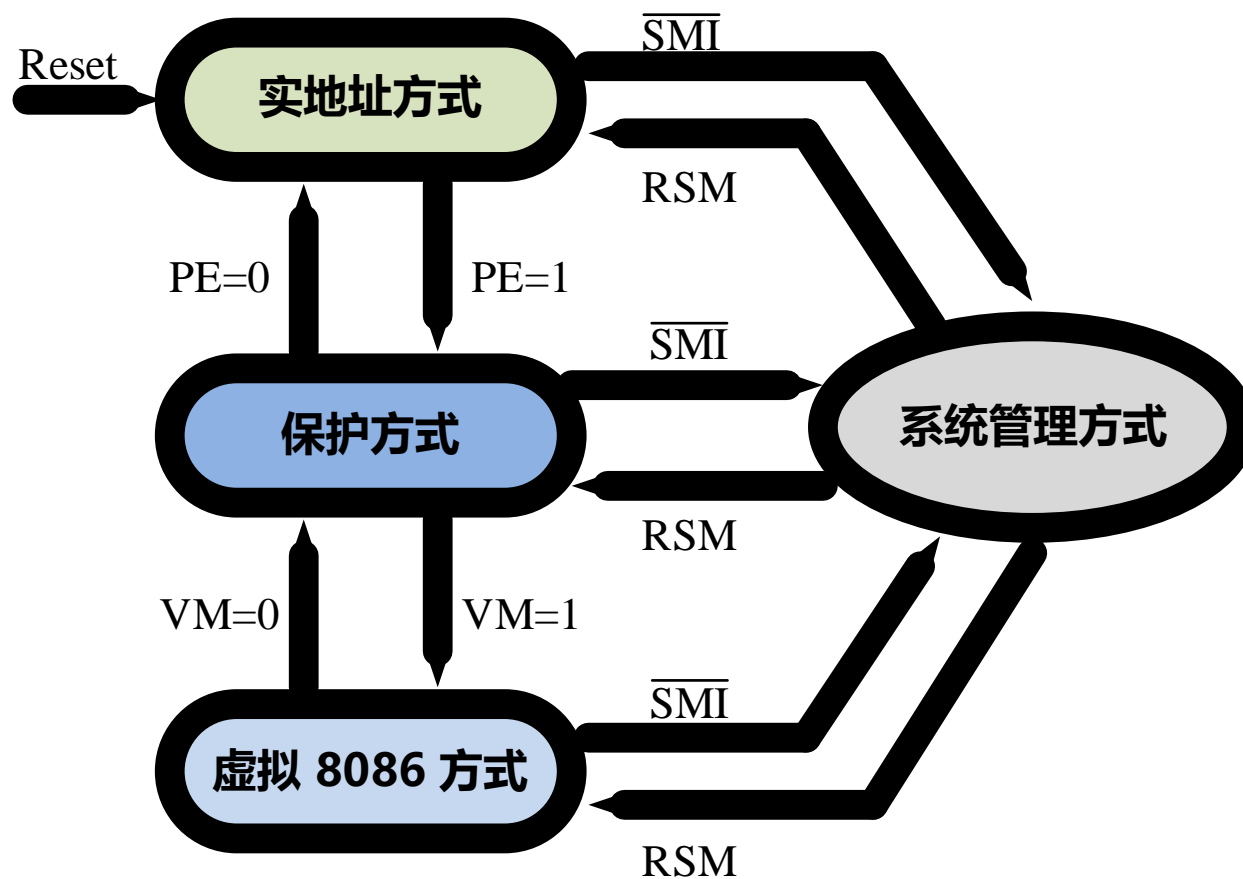
## 工作模式

- **实地址模式** (Real-address mode)
  - 工作方法相当于一个8086
- **保护模式** (Protected mode)
  - 提供支持多任务环境的工作环境，建立保护机制，是 **32位处理器的常态工作方式**
- **虚拟 8086 模式** (Virtual-8086 mode)
  - 保护模式下切换到 8086 的工作方式



# 04 | 工作模式 - 80486工作模式2

## 工作模式的切换



# 04 | 工作模式-实地址模式

## 实地址模式

- 实地址模式是 IA-32 系列处理器中**最初的处理器的工作方式**，用于初始化系统
- 采取了**分段寻址**的模式，16 位段基地址: 16 位偏移地址
- 只能访问最低端的 **1M 字节**的物理地址空间。地址空间的范围是 00000H 至 FFFFFH
- 只支持存储器的分段管理，而且每个存储段的大小限于 64K 字节
- 在实模式下，IA-32 系列处理器**不能发挥其全部性能**



# 04 | 工作模式-保护模式1

## 保护模式

- IA-32 系列处理器的常态工作模式
- 通过段选择符从相应的“段描述符表”中选择一个段描述符进行寻址
- 全部 32 根地址线有效，可寻址高达 4G 字节的物理地址空间
- 支持存储器分段管理和可选的存储器分页管理机制

# 04 | 工作模式-保护模式2

## 保护模式

- 支持**虚拟存储器**的实现，提供**完善的保护机制**，支持操作系统**实现多任务管理**
- **只有在保护模式下，IA-32 系列处理器才能够发挥出其全部性能和特点**
- Windows 操作系统和基于 IA-32 处理器的 Linux 操作系统都运行于保护模式

# 04 | 工作模式-虚拟8086模式

## 虚拟 8086 模式

- 虚拟 8086 模式是**保护模式**下的一种工作方式，也称为**V86 模式**
- 内存寻址方式、寻址空间，与实模式一样
- 既允许执行 8086/8088 程序，又能有效地利用保护功能
- 支持多任务和内存分页
- **既能够执行 8086 程序，又拥有保护模式下的多任务管理等特性**

# 本章小结

---

- 掌握 CPU 的内部结构，以及主要寄存器的基本特点
- 能够在以后的汇编语言程序设计中灵活使用寄存器
  - 注意段寄存器和标志寄存器的含义
- 理解 16 位系统中的存储器管理模式
- 清楚存储单元地址的含义和变换



河南大學  
Henan University



# Q&A

---

主讲教师：舒高峰

电子邮箱：gaofeng.shu@henu.edu.cn

联系电话：13161693313