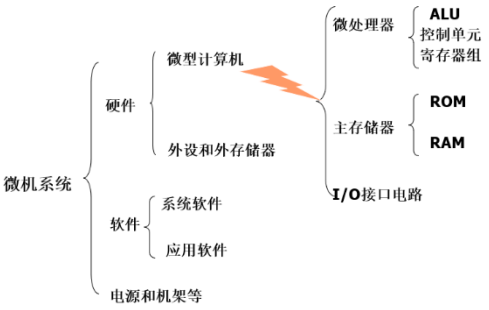
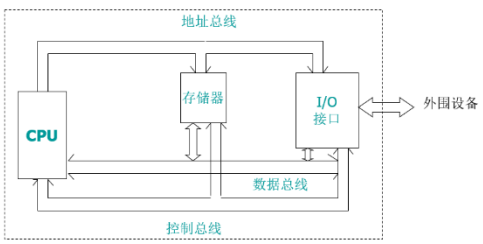
一、 第一章

1. **微型计算机：**以微处理器为核心，配上由大规模集成电路制作的只读存储器（ROM）、读写存储器（RAM）、输入 / 输出接口电路以及相应的辅助电路等所组成的计算机，称为微型计算机（Microcomputer）。
2. **单片机：**将这些组成部分集成在一片超大规模集成电路芯片上，称为单片微型计算机， 称单片机。(单片机体积小、功耗低，在智能化仪器仪表以及控制领域应用极广)
3. **微型计算机系统：**以微型计算机为中心，配以相应的外围设备（如键盘、鼠标、显示器、打印机等）和其他专用电器、电源、面板、机架以及控制微型计算机工作的**软件（系统软件和应用软件）**，就构成了完整的微型计算机系统（Microcomputer System）
   - 微型计算机如果不配有软件，通常称为**裸机**。
4. **单板机：**将微型计算机与简单的外设集成在一块电路板上，称为单板机。



5. **字长：**指微处理器一次可以直接处理的二进制数码的位数
   - 它通常取决于微处理器内部通用寄存器的位数和数据总线的宽度
   - 微处理器的字长有 4 位、8 位、16 位、32 位和 64 位等。
   - 8086 称为 16 位微处理器，而 80386DX 称为 32 位微处理器，安腾（Itanuim）为 64 位微处理器。 8088 称为准 16 位微处理器，而 80386SX 称为准 32 位微处理器。
6. **主频：**也称为时钟频率，用来表示微处理器的运行速度，主频越高表明微处理器运行越快，主频的单位是 MHz 或 GHz。
   - 早期微处理器的主频与外部总线的频率相同。
   - 从 80486DX2 开始，**主频=外部总线频率×倍频系数**
   - 外部总线频率频率通常简称为外频，也即为主板的工作频率，它的单位也是 MHz。外频越高说明微处理器与系统内存数据交换的速度越快，因而微型计算机的运行速度也越快。
   - 倍频系数是微处理器的主频与外频之间的相对比例系数。
   - 通过提高外频或倍频系数，可以使微处理器工作在比标称主频更高的时钟频率上，这就是所谓的超频。超频往往以改变外频为主
7. **微处理器的生产工艺：**指在硅材料上生产微处理器时内部各元器件间连接线的宽度(线宽)，一般以μm、nm 为单位，数值越小，生产工艺越先进，微处理器的功耗和发热量越小。目前微处理器的生产工艺已经达到 90nm，65nm，45nm。
8. **微处理器的集成度：**指微处理器芯片上集成的晶体管的密度。最早 Intel 4004 的集成度为 2250 个晶体管，而 Pentium III 的集成度已经达到 750 万个晶体管以上，集成度提高了 3000 多倍。Pentium IV 集成了 4200 万个晶体管。安腾四核处理器 Tukwilla：20.5 亿个晶体管，65nm
9. **微处理器的功能：**算术逻辑运算，指令译码、执行，数据暂存，与 MEM、I/O 交换数据，提供整个系统所需的定时和控制，响应中断请求
10. **微型计算机的基本结构：**CPU+存储器+I/O 接口+系统总线 + 外设



（此处的总线称为"片总线"，是微处理器的引脚信号，它是连接微处理器同内存储器、I/O 接口电路之间的连接纽带。）

11. **总线分类：**
    - 片总线：又称元件级总线；

- 内总线：又称系统总线、微机总线、板级总线（微型计算机系统中各插件之间的信息传输通路。）
- 外总线：又称通信总线。(微型计算机系统之间，或是微型计算机系统与其它系统之间的信息传输通路。)

12. **主板：**CPU 插槽[插座]，内存插槽，连接硬盘机、光盘驱动器的外设接口插座（IDE 接口），PCI 插槽、ISA 插槽、 AGP 插槽等接插各种接口卡所用的扩展插槽，主板 BIOS 芯片、CMOS 芯片（记录了系统的一些重要信息，如硬盘的设置以及系统日期和时间等，电脑每次启动时都要先读取里面的信息。）等，控制芯片组，外设接口（鼠标、键盘、串/并口、USB)、电源插座等，跳线和开关、电池、电容、电阻，等等
13. **主板控制芯片组：**主板控制芯片组（Chipset）是控制局部总线、内存和各种扩展卡的，是整块主板的灵魂所在。CPU 对其它设备的控制都是通过它们来完成的。
14. **芯片组的功能：**CPU 是 PC 机能完成信息处理功能的核心器件，但是 CPU 要完成 PC 机所需要的信息处理功能，还必须有一系列的“支持电路”和“接口电路”。
    - CPU 要能向外部设备输入或输出信息，必须要有并行接口电路和串行接口电路等。
    - CPU 要能向内存芯片进行数据传送．必须要有内存控制电路。
    - CPU 要能具有中断功能，必须要有“中断控制电路”。
    - CPU 要能支持功能 DMA，必须要有“DMA 控制电路”。
    - 要把 CPU 的芯片总线转换成系统中各模块间传输信息的公共通路——系统总线，必须要有“总线控制电路”。
    - 要向 CPU 及系统中其他部件提供时钟信号，那么“时钟发生电路”也是必不对少的，

通过 VLSI 技术，将主板上众多的接口电路和支持电路按不同功能分别集成到一块或几块集成芯片之中，这几片 VLSI 芯片的组合称为“控制芯片组”，简称“芯片组”。

二、 第二章

1. **64 位模式下的程序设计模型**
    - 64 位寄存器 R8~R15 是可以按照字节、字、双字或者四字的方式寻址的。按字节寻址部分是最低 8 位。
    - 不支持把其中的第 8~15 位作为 1 字节来直接寻址。
    - 高字节寄存器（AH、BH、CH、DH）不能与由 R8~R15 所表示的字节在同一指令中寻址。
2. **RIP 寄存器：**在 64 位模式中，目前包括 40 位地址，指向要执行的下一条指令的偏移值，该偏移值相对于指令所在代码段的基地址（段基址)。
    - EIP：32 位指令指针，用于 32 位微处理器中。
    - IP：在 8086 和 80286 中，指令指针为 16 位寄存器。
    - 程序员不能对 RIP/EIP/IP 进行直接存取操作。**程序中的转移指令、返回指令以及中断处理能对 RIP/EIP /IP 进行操作。**
    -
3. **IOPL（Input/Output Privilege Level）：I/O 特权级标志**
    - 用于保护方式，指示 I/O 设备的特权级，IOPL 的 2 位代码决定 4 级特权级。
    - 如果当前特权级（Current Privilege Level，CPL）高于 IOPL，则 I/O 指令能顺利执行；否则产生中断（异常 13 故障），使任务挂起。
4. **NT（Nested Task）：嵌套任务标志**
    - 用于保护方式。

- 指示当前执行的任务是否嵌套在另一任务中，该位的置位和复位通过向其它任务的控制转移来实现。
- IRET 指令会检测 NT 的值。若 NT=0，则执行中断的正常返回；若 NT = 1，则执行任务切换操作。

5. **RF（Resume）：恢复标志**
   - 该标志和调试寄存器配合使用，用于控制调试失败后强制程序恢复，返回断点继续执行。
   - 断点处理前在指令边界上检查该位。
   - 若 RF=1，则下条指令的任何调试故障都被忽略，不产生异常中断。
   - 若 RF=0，调试故障被接受，指令断点可以产生调试异常。
   - 每条指令成功执行完毕，如无故障出现，则 RF 自动被清 0。但 IRET、POPF 以及引起任务切换的 JMP、CALL 和 INT 指令除外，这些指令将 RF 置成存储器映象指定的值。
6. **关于六个标志位 CF，PF，AF，ZF，SF，OF：**
   - 在执行**算术和逻辑指令**后会改变；而对于任何数据传送指令或程序控制操作，这些标志都不改变。
   - 注意：任务切换。
   - **无符号整数，大于，CF∪ZF=0。（注：CF=0, ZF=0）**
   - **有符号整数，小于，OF⊕SF=1。（注：OF=1，SF=0）**
   - Eg. -2147483648（0x8000 0000）<2147483647，结果为 FALSE
     - CPU 对有符号数和无符号数采用同一套电路来处理。对程序员而言，做有符号数运算时，需关注 OF；做无符号数运算时，需关注 CF。
     - 对于此例，做减法时，实际上是对-2147483647 取补码，然后相加。但是，此时，相加会使得 CF=1？实际上，做减法时，CPU 要对 CF 标志是再次取反。
7. **6 个段寄存器：CS, DS, SS, ES, FS, GS**
   - 用来保存标志现行可寻址存储段的**段基址或段选择子**（Selector，又称选择符）。
   - 16 位微处理器： CS, DS, SS, ES，保存段基址。
   - 32 位和 64 位微处理器：CS, DS, SS, ES, FS, GS，保存段选择子。
8. **CS：Code Segment，代码段寄存器**
   - 定义代码段存储区的起始地址（及有关属性）。
   - 代码段是微处理器用来存放代码（程序，包括过程）的一段存储区。
   - 在实地址方式下，定义了一段 64KB 存储区的起始地址。
   - 在保护方式下，用来选择一个描述符（又称为描述子），该描述符用来描述一个代码段的若干特性——起始地址、段限以及访问权等。
   - 最大段限在 8086 及 80286 中为 64KB，而在 80386 及其以上的微处理器中则为 4GB。64 位模式下，代码段寄存器仍用于平展模式，但用法不同。
9. **DS：Data Segment，数据段寄存器**
   - 定义了数据段存储区的起始地址（及有关属性）。
   - 数据段是存放供程序使用的数据的一段存储区。
   - 数据段中的数据按其给定的偏移地址值 offset（或称有效地址 EA, Effective Address）来访问。
   - 数据段的长度规定同代码段。
10. **SS：Stack Segment，堆栈段寄存器**
    - 定义了堆栈段存储区的起始地址（及有关属性）。
    - 堆栈段是一段用作堆栈的存储区。

- 堆栈段现行的入口地址由堆栈指针 RSP（或 ESP、SP）决定。
- RBP（或 EBP、BP）也可寻址堆栈段中的数据。

11. **ES：Extra Segment，附加段寄存器**
    - 定义了附加段存储区的起始地址（及有关属性）。
    - 附加段是一段附加的数据段，通常供串操作类指令用于存放目的串数据。
12. **FS、GS**
    - 80386 以上的微处理器（含 80386）有两个新增的附加的段存储区。
    - FS 和 GS 用来定义这两个附加的数据段存储区的起始地址（及有关属性）。
    - DS、ES、FS 和 GS 的选择子用来指出现行的数据段。

13. 实模式存储器寻址：
- 8086：只能工作于实模式。
- 80286 及其以上：可以工作在实模式或者保护模式。

14. 实模式的特点：
a) 只允许微处理器寻址起始的 1MB 存储器空间，即使是 Pentium 4 和 Core 2 微处理器。
b) 微处理器每次加电或复位后，以实模式开始工作。
c) Windows 操作系统不使用实模式。
d) 如果 Pentium 4 或 Core 2 处于 64 位模式中，则不能执行实模式应用程序，从而 DOS 应用程序不存在于 64 位模式。
   i. 除非为 64 位模式编写虚拟 DOS。

15. 实模式段和偏移：
- 实模式下，用段地址和偏移地址的组合来访问存储单元。所有实模式存储单元的地址都有段地址加偏移地址组成。
- 实模式段的长度总是 64KB。
- 段地址（Segment Address）装在段寄存器中，确定 64KB 存储段的起始地址。
- 偏移地址（Offset Address）用于在 64KB 存储段内选择任一单元。

16. **实模式下的段式地址管理：**
    - 逻辑地址：程序设计时，使用的是逻辑地址。逻辑地址由“段基址”和“偏移量”构成（均为 16 位）。
      - **表示为“段基址:偏移量”**
    - 物理地址： CPU 访问存储器时，在地址总线上实际送出的地址。
      - **物理地址（20 位）= 段基值×16 + 偏移量。**
      - 每个存储单元有唯一的物理地址，但它却可由不同的“段基址”和“偏移量”组成。
      - 有时，寻址方式将多个寄存器内容与一个位移量结合，形成偏移地址。此时，如果这些值超过 FFFFH，则将进位舍去
    - 在 80286（有专门外部电路）及 80386~Pentium 4 中，当段地址是 FFFFH，在 80286（有专门外部电路）及 80386~Pentium 4 中，当段地址是 FFFFH，而且系统中安装了用于 DOS 的驱动程序 HIMEM.SYS 时，可以寻址 64KB 减 16 字节的附加存储器区域。
      - 这个可寻址的存储器区域（0FFFF0H~10FFEFH）称为高端存储器。
      - 用段地址 FFFFH 生成地址时，地址 A20 引脚被置位（如果支持）。
        - **例如，段地址为 FFFFH，偏移地址为 4000H。**
        - **如果支持 A20，则寻址 FFFF0H+4000H，即 103FF0H.**
        - **如果不支持 A20，则寻址 03FF0H，即 A20 为 0。**

- 段的划分：定长，可连续、可离散、可覆盖、可重叠
- 默认的“16 位段+偏移”寻址组合：

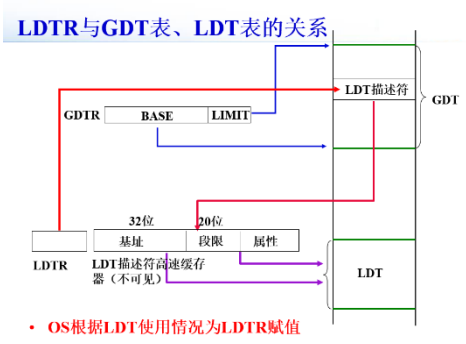
Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string instructions	String destination address

- 默认的“32 位段+偏移”寻址组合：

Segment	Offset	Special Purpose
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

17. 段和偏移寻址：
    - **可重定位程序（relocatable program）：**可以放入存储器的任何区域，且不需修改仍能执行的程序。
    - **可重定位数据（relocatable data）：**可以放入存储器的任何区域，且不需修改就可以被程序引用的数据。
    - **“段+偏移”寻址机制允许程序和数据的重定位。**
18. **保护模式存储器寻址：**
    - 形式上，依然是“段基址+偏移量”。因此，很多保护模式指令和实模式指令是相同的，但访问存储器时的解释方法不同。
    - **不同：**
    - 段寄存器中不是直接存放“段基址”，而是存放着一个“**选择子**”，用于选择描述符表中的一个描述符。描述符（Descriptor）包含了存储段的位置、长度和访问权限。
    - 在 32 位微处理器中，偏移地址是 32 位。
    - 为 32 位保护模式编写的程序也可以在 Pentium 4 的 64 位模式下运行。
19. **选择子：**
    - 段选择子寄存器
        - 保护模式下 CS、DS、SS、ES、FS、GS 寄存器称为段选择子寄存器，其值不再是段基址而是选择子，它从描述符表中选择一个定义存储器段大小和属性的描述符。
    - **全局描述符表（Global Descriptor Table，GDT）：**定义了能被系统中所有任务公用的存储分段，可以避免对同一系统服务程序的不必要的重复定义与存储。
        - 一个系统只能有一个全局描述符表（GDT）。
        - 全局描述符表：定义了能被系统中所有任务公用的存储分段，可以避免对同一系统服务程序的不必要的重复定义与存储。
        - 通常 GDT 中包含了操作系统使用的代码段、数据段、任务状态段以及系统中各个 LDT 所在段的描述符。
        - 注意：中断服务程序所在段的段描述符在 IDT 中。
        - **GDT 本身不是一个段；它是一个在线性地址空间的数据结构。**
        - Eg. (GDTR)=0010 0000 0FFFH，求 GDT 在存储器中的起始地址，结束地址，表的大小，表中可以存放多少个描述符？
            - GDT 的起始地址为 0010 0000H
            - GDT 的结束地址为 0010 0000H+0FFFH=0010 0FFFH
            - 表的大小为 0FFFH+1=4096 字节
            - 表中可以存放 4096/8=512 个描述符

- 通常 GDT 中包含了操作系统使用的**代码段、数据段、任务状态段以及系统中各个 LDT 所在段的描述符**。
  - 注意：中断服务程序所在段的段描述符在 IDT 中。
- **局部描述符表（Local Descriptor Table，LDT）**包含了与某个任务相关联的段描述符。
  - 在设计操作系统时，通常一个任务有一个独立的 LDT。
  - 包括 16 位段选择子，不可编程的 64 位段描述符寄存器（又称为段描述符 Cache）。
  - 在 64 位段描述符寄存器中，有 32 位 LDT 的线性基地址、20 位段限及 12 位的段属性。
  - 一个任务一个 LDT。因此，系统中有多个 LDT。为了访问局部描述符表，需在 LDTR 中装入一个选择子，用该选择子访问全局描述符表，并把局部描述符表的段基址、段限和段属性装入 LDTR 的 Cache 区。
  - LDT 提供了将一任务的代码段、数据段与操作系统的其余部分相隔离的机制。
  -



- **中断描述符表（Interrupt Descriptor Table，IDT）**最多包含 256 个中断服务程序的位置的描述符。
  - 系统所使用的每种类型的中断在 IDT 中必须有一个描述符表项。
  - IDT 的表项通过中断指令、外部中断和异常事件来访问。
  - 为容纳 Intel 保留的 32 个中断描述符，IDT 的长度至少应有 256Byte（32×8byte）。
  - 与 GDT 相似，IDT 也不是一个段。
-
- 每个描述符表包含 8192 个描述符，所以任何时刻应用程序最多可以有 16384 (2^{14}) 个描述符。
- 一个描述符对应一个存储段，一个存储段最大可达 4GB，则应用程序能够访问 4G*16384B=64TB。
- **描述符的 TYPE 段：TYPE 为 4 个字节，共有 16 种类型。其中：**
  - 2，LDT；9，TSS，非忙；B，TSS，忙；5，任务门；C，调用门；E，中断门；F，陷阱门

20. 程序不可见寄存器：

- 存储系统中有全局描述符表和局部描述符表。为了访问和指定这些表的地址，微处理器中包含一些程序不可见寄存器。
- 程序不可见寄存器不直接被软件访问，但其中一些寄存器可以被系统软件访问。
- **段寄存器及其高速缓冲存储器（Cache）**
  - 段寄存器：程序员可见。
  - 段描述符高速缓冲存储器：对程序员透明，又称段描述符寄存器。
- 每当一个新的“段号”放入段寄存器时，微处理器就访问一个描述符表，并把描述符装入该段寄存器对应的高速缓冲存储器区域内。
- 该描述符一直保存在高速缓冲存储器区域内，并在访问内存段时使用，直到“段号”发生变化。这就允许微处理器重复访问一个内存段时，不必每次去查询描述符表。
- **GDTR（Global Descriptor Table Register），**全局描述符表寄存器，共有 48 位。
  - 高 32 位保存全局描述符表的线性基地址。

    - 低 16 位是表限字段，即表的最大长度仅 64KB。
- **IDTR（Interrupt Descriptor Table Register），**中断描述符表寄存器，共有 48 位
    - 高 32 位用于保存中断描述符表 IDT 的 32 位线性基地址。
    - 低 16 位是表限字段，表的最大长度也是 64KB。
- **TR（Task State Segment Register），**任务状态段寄存器。
    - 包括 16 位段选择子，64 位描述符寄存器（其中包括 32 位任务状态段的线性基地址，20 位的段限及 12 位段属性）。
    - 每一个任务都有一个任务状态段 TSS。
    - 现行的 TSS 由 TR 来标识。
    - 根据 TR 中的选择子，从 GDT 中索引 TSS 描述符，微处理器自动将 TSS 描述符装入 TSS Cache 中。

21. 内存分页机制：

- 80386 及更高档微处理器。
    - 为任何线性地址分配任何物理存储器地址。
- **逻辑地址**，即虚拟地址，这是应用程序设计人员进行编程时要用到的地址。
    - “段选择子：偏移量”：32 位微机中，由一个 16 位的段选择子和 32 位偏移量两部分组成。段选择子放在段寄存器中；偏移量也称为偏移地址、有效地址。
- **线性地址**：沟通逻辑地址与物理地址的桥梁。
    - 32 位微处理器芯片内的分段部件将逻辑地址空间转换成 32 位的线性地址。
- **物理地址**：指内存芯片阵列中每个阵列对应的唯一的地址。32 位地址线可直接寻址 4GB 内存单元。

22. 分页有关寄存器：

- **CR0** 最左边的一位 PG：
    - PG=1，允许分页，通过分页部件将线性地址转换成物理地址；
    - PG=0，线性地址就是物理地址。
- **CR2**：页故障线性地址
    - 用于保存页故障线性地址（this linear address that caused a page fault），32 位。
    - 操作系统中的页异常处理程序可以通过检查 CR2 的内容，得知 32 位的线性地址。
- **CR3** 页目录基址寄存器。
    - 高 20 位存放页目录表的物理基地址。由于页目录表是按页对齐的（4K），因此只需保存高 20 位。
    - 低 12 位，有 PCD 和 PWT 等 7 位已定义。
    - PWT：Page write-through,指示是页面通写还是回写。
        - PWT=1,外部 Cache 对页目录进行通写；
        - PWT=0,进行回写。
    - PCD：Page Cache disable，页面 Cache 工作情况。(是否允许页面放在 Cache 中。)
        - PCD=1，禁止片内的页面 Cache；
        - PCD=0，允许片内的页面 Cache。

23. **分页：**

- 页目录表的长度为 4KB，最多可寻址 1024 个页表。
- 页表的长度为 4KB，最多可寻址 1024 个页。
- 对于 Pentium~Core2，可以按 4KB、2MB、4MB 长度分页。
- 按 2MB、4MB 长度分页时，只有页目录表和内存分页，没有页表。

24. **平展模式内存模型：**

- 采用 64 位扩展的 Pentium 4 或 Core 2 的内存系统为平展模式内存系统。
- 平展模式内存系统是不存在分段的系统（40 位地址）。
- 平展模式不使用段寄存器进行寻址。
  - 平展模式不使用描述符中的基址和界限来选择段的内存地址。
  - 段是重叠的（Overlapping），都从地址 0 开始。
  - 64 位模式下的偏移地址即为有效地址 Effective address。
  - CS 段寄存器用来从只定义代码段访问权限的描述符表中选择描述符。
  - 段寄存器仍然负责选择软件的优先级别。
- 在 64 位模式下，如果把地址设置为 IA32 兼容的（描述符中 L 位为 0），那么地址是 64 位的，但是由于地址中只有 40 位被引出到地址线，任何超过 40 位的地址都会截断。
  - 使用偏移地址的指令只能使用 32 位偏移，即允许从当前指令开始的±2GB 的地址范围。这种寻址方式被称为 RIP 相对寻址（在第 3 章解释）。
  - 直接传送指令（move immediate instruction）允许完全 64 位寻址和对任意平展模式内存地址的访问。
  - 其他指令不允许对 4GB 以上的地址空间进行访问，因为其偏移地址仍为 32 位。
  - 如果 Pentium 工作在完全 64 位模式下（描述符中 L 位为 1），其地址可以为 64 位或 32 位（在第 4 章解释）。

三、 第三章

四、 第四章

1. **机器语言：**作为指令由微处理器理解和使用的二进制代码，用来控制微处理器自身的运行。
   - 8086~Core 2 的机器语言指令长度从 1 字节到 13 字节。
   - 尽管机器语言好像很复杂，但这些微处理器的机器语言也很规则。
   - 共有 100,000 多种变化形式的机器语言指令。
   - 几乎不能列一个完整的指令表来包含这多么变形。
   - 在机器语言指令中，某些二进制位是已给定的，其余二进制位则由每条指令的变化形式来确定。
2. **16 位指令模式：**
   - 8086~80286 的指令是 16 位指令模式。
   - 16 位指令模式与 80386 及更高型号微处理器工作在 16 位指令模式时是兼容的。
   - 80386 及更高型号微处理器工作在实模式时，假定所有指令都是 16 位模式。
   - 在保护模式中，描述符的高端字节包含选择 16 位模式或 32 位模式指令的 D 位。
3. **32 位指令格式：**
   - 32 位指令格式的头 2 个字节，因为不常出现，故称为超越前缀（Override prefix）。
   - 第 1 个字节用来修改指令操作数的长度，称为地址长度前缀。
   - 第 2 个字节修改寄存器的长度，称为寄存器长度前缀。
   - 32 位指令格式的寄存器长度前缀：
   - 如果 80386~Pentium 4 按 16 位指令模式的机制操作（实模式或保护模式），而使用 32 位寄存器，则指令的前面出现寄存器长度前缀 66H。
   - 如果微处理器按 32 位指令模式操作（只在保护模式），而且使用 32 位寄存器，则不存在寄存器长度前缀。
   - 如果在 32 位指令模式中出现 16 位寄存器，则要用寄存器长度前缀选择 16 位寄存器。
   - mov ax，1 与 mov eax,1 的机器指令是相同的。具体 CPU 会执行哪中操作要视

当前代码段是 16 位还是 32 位而定。如果要在 32 位码段中执行 mov ax，1 这样的操作，则必须靠加上特殊的指令前缀来实现。

- **16 位指令模式用 8 位及 16 位寄存器和寻址方式；而 32 位指令模式使用 8 位及 32 位寄存器和寻址方式，这是默认的用法。**
- **前缀可以超越这些默认，使得 32 位寄存器可以用于 16 位模式，而 16 位寄存器可以用于 32 位模式。**

4. **指令格式：**
   - **操作码：**选择微处理器执行的操作（加、减、传送）等。多数机器语言指令的操作码长度为 1 或 2 个字节。
     - D 位：指示数据流的方向（REG→R/M 或 R/M→REG）。
       - D=1，数据从第 2 字节的 R/M 字段流向 REG 字段。
       - D=0，数据从 REG 字段流向 R/M 字段。
     - W 位：指令模式位（W=0: 8 位；W=1: 16 或 32 位）。
       - W=0，表示数据长度为字节。
       - W=1，表示数据的长度是字或双字（由寄存器长度前缀确定）。
   - **MOD 字段：**规定指令的寻址方式，选择寻址类型及所选类型是否有位移量。
     - 如果 MOD=11，选择寄存器寻址方式，用 R/M 字段指定寄存器而不是存储单元。
     - 如果 MOD=00，01，或 10，R/M 字段选择一种数据存储器寻址方式。
     - 00 表示不带位移量，例，MOV AL, [DI]
     - 01 表示包含 8 位有符号扩展的位移量。例，MOV AL, [DI+2]
     - 10 表示包含 16 位有符号扩展的位移量。例，MOV AL, [DI+1000H]
     - **当微处理器执行指令时，将 8 位的位移量符号扩展为 16 位的位移量。例如，80H 符号扩展（sign-extended）后成为 FF80H。**
     - **变化：**
       - 80386~Core 2 微处理器中，对于 16 位指令模式，MOD 字段的含义没有变化。
       - 80386~Core 2 微处理器中，MOD 字段的含义受地址长度超越前缀影响。
         - 当 MOD=10 时，16 位指令模式下，16 位位移量变成 32 位位移量。
         - 在 80386 以上微处理器中，工作在 32 位指令模式下时，如果不用地址长度超越前缀，则只允许用 8 位或 32 位位移量。
         - 当然，32 位指令模式下，8 位/16 位的位移量将被符号扩展至 32 位。

五、第五章

1. **加法：**

a) 加法

加法	格式	ADD REG/MEM, REG/MEM/IMM
	功能	源操作数、目的操作数相加，结果存入目的操作数
	标志	所有状态标志（ZF、CF、PF、AF、SF 及 OF）都受影响

b) **带进位加：**

带进位加	格式	ADC REG/MEM, REG/MEM/IMM
	功能	源操作数、目的操作数以及进位标志CF相加，结果存入目的操作数
	标志	所有状态标志（ZF、CF、PF、AF、SF 及 OF）都受影响

c) **+1**

加1	格式	**INC REG/MEM**
	功能	目的操作数加**1**
	标志	除**CF**标志位，其余状态标志都受影响

d) **交换相加：**

交换并相加	格式	**XADD REG/MEM, REG**
	功能	（**80486**以上）源操作数和目的操作数相交换，并将两者之和存入目的操作数
	标志	所有状态标志都受影响，根据加法结果设置

e) **加法指令注意事项：**

i. 源操作数和目的操作数不能同时为内存单元（MEM）。

ii. 不允许与段寄存器（SREG）相关的加法。

iii. XADD 指令的源操作数在寄存器（REG）中。

iv. 标志寄存器中状态位随运算结果而变化，但 INC 指令不影响 CF 标志。

v. 指令中操作数是带符号数还是无符号数由程序员解释。

vi. 注意：第 4 章的**数据传送指令不改变状态标志**。

vii. Eg. **试用加法指令对两个 8 位 16 进制数 5EH 和 3CH 求和，并分析加法运算指令执行后对标志位的影响。解：**

1. MOV AL, 5EH ;AL=5EH（94）
2. MOV BL, 3CH ;BL=3CH（50）
3. ADD AL, BL ;结果 AL=9AH
4. 运算后标志：ZF=0, AF=1, CF=0, SF=1, PF=1, OF=1。
5.
6. 若程序员认为两个加数是无符号数，则运算结果位 9AH，即 154。此时，SF 标志和 OF 标志没有意义。
7. 若程序员认为两个加数是有符号数，则运算溢出，结果无效。此时，CF 标志没有意义。

viii. **当加减运算结果的最高有效位有进位（加法）或借位（减法）时，CF 标志置 1，即 CF = 1；否则 CF = 0。**

1. 针对无符号整数，判断加减结果是否超出表达范围。（CF 是 1，超出）
2. **N 个二进制位表达无符号整数的范围：**
   a) **$0 \sim 2^N-1$**
3. Eg.
   a) $(00111010 + 01111100)_B=(3A + 7C)_H=(58 + 124)_D=182<255$ CF=0
   b) 10101010 + 01111100=AA + 7C=170 + 124=294>255 CF=1

ix. **有符号数加减结果有溢出，则 OF = 1；否则 OF = 0。**

1. 针对有符号整数，判断加减结果是否超出表达范围。（OF 是 1，超出）
2. **N 个二进制位表达有符号整数的范围：**
   a) **$-2^{N-1} \sim 2^{N-1}-1$**
3. Eg.
   a) $(00111010 + 01111100)_B=(3A + 7C)_H=(58 + 124)_D=182>127$ OF=1
   b) 10101010 + 01111100=AA + 7C=-86+124=38 OF=0

x. 进位和溢出的区别：

1. 进位标志反映无符号整数运算结果是否超出范围。有进位，加上进位或借

位后运算结果仍然正确。

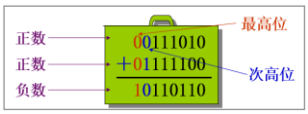
2. 溢出标志反映有符号整数运算结果是否超出范围。有溢出，运算结果已经不正确。

xi. **处理器按照无符号整数求得结果，在设置进位标志 CF 的同时，根据是否超出有符号整数的范围设置溢出标志 OF。**应该利用哪个标志，由程序员决定!

1. 操作数是无符号数，关心进位
2. 操作数是有符号数，注意溢出

xii. **溢出标志的判断：**

1. 最高位和次高位同时有进位或同时无进位，无溢出；最高位和次高位进位状态不同，有溢出



2. 只有当两个相同符号数相加（含两个不同符号数相减），而运算结果的符号与原数据符号相反时，产生溢出；其他情况下，不会产生溢出

2. **奇偶标志 PF（Parity Flag）**
   a) 当运算结果最低 8 位中“1”的个数为零或偶数时，PF = 1；否则 PF = 0
   b) 8 位二进制数相加：00111010 + 01111100 = 10110110 “1”的个数为 5 个：PF = 0
   c) 8 位二进制数相加：10000100 + 01111100 = [1]00000000 “1”的个数为 0 个：PF = 1
3. **零标志 ZF（Zero Flag）**
   a) 运算结果为 0，则 ZF = 1，否则 ZF = 0
   b) 8 位二进制数相加：00111010 + 01111100 = 10110110 结果不是 0，ZF = 0
   c) 8 位二进制数相加：10000100 + 01111100 = [1]00000000 结果是 0，ZF = 1
4. **符号标志 SF（Sign Flag）**
   a) 运算结果最高位为 1，则 SF = 1；否则 SF = 0。
   b) 8 位二进制数相加：00111010 + 01111100 = 10110110 最高位 = 1：SF = 1
   c) 8 位二进制数相加：10000100 + 01111100 = [1]00000000 最高位 = 0：SF = 0
5. **辅助进位标志（Auxiliary Carry）辅助进位标志。**
   a) 用于标志 D3 向 D4 位之间的进位（加法运算）或借位（减法运算）的状态。
   b) AF 标志供 DAA 和 DAS 指令使用，以便在 BCD 码的加法或减法之后对 AL 中的结果值进行十进制调整。
6. **减法：**

a) 减法：

减法	格式	SUB REG/MEM, REG/MEM/IMM
	功能	目的操作数—源操作数，结果存入目的操作数
	标志	所有状态标志（ZF、CF、PF、AF、SF 及 OF）都受影响

b) 带借位减：

带借位减	格式	SBB REG/MEM, REG/MEM/IMM
	功能	目的操作数—源操作数—进位标志CF，结果存入目的操作数
	标志	所有状态标志都受影响

c) -1：

减1	格式	DEC REG/MEM
	功能	目的操作数减1
	标志	除CF标志位，其余状态标志都受影响

d) 减法指令注意事项与加法指令类似：

i. 源操作数和目的操作数不能同时为内存单元（MEM）。

ii. 不允许与段寄存器（SREG）相关的加法。

iii. 标志寄存器中状态位随运算结果而变化，但 DEC 指令不影响 CF 标志。

iv. 指令中操作数是带符号数还是无符号数由程序员解释。

7. 比较指令：

a) 比较：

比较	格式	CMP REG/MEM, REG/MEM/IMM
	功能	目的操作数减去源操作数 源操作数、目的操作数相减不能同时为内存单元。
	标志	影响 ZF、CF、PF、AF、SF 及 OF 。

b) 比较交换：

比较交换	格式	CMPXCHG REG/MEM, REG
	功能	(80486以上) 比较目的操作数与AL、AX 、 EAX 或RAX寄存器中的值。如果两个值相等，则将源操作数加载到目的操作数。否则，将目标操作数加载到 AL、AX、 EAX 或RAX。
	标志	如果目标操作数与 AL、AX 、 EAX 或RAX中的值相等，则置 ZF 标志，否则清除此标志。CF、PF、AF、SF 及 OF 标志根据比较操作的结果设置。

c) 比较交换 8 字节：

比较并交换8字节	格式	CMPXCHG8B MEM64
	功能	（Pentium以上）比较目的操作数和EDX:EAX 中的 64 位值。如果这两个值相等，则将 ECX:EBX 中的 64位值存储到目的操作数。否则，将目标操作数的值加载到 EDX:EAX。目标操作数是 8 字节内存位置。EDX 与 ECX 包含 64 位值的 32 个高位，EAX 与 EBX 包含 32 个低位。
	标志	如果两值相等， ZF=1，否则ZF=0；CF、PF、AF、SF 及 OF 标志不受影响。

i. **CMPXCHG8B 指令有个毛病，可能会引起系统崩溃。**

d) 比较交换 16 字节：

比较并交换16字节	格式	CMPXCHG16B MEM128
	功能	与CMPXCHG8B类似，但寄存器为RDX:RAX，RCX:RBX。但是，要求MEM128是16-byte对齐。
	标志	与CMPXCHG16B一样。

8. 乘法：

a) 无符号乘法：

无符号乘法	格式	MUL REG/MEM
	功能	8位：源操作数与AL相乘，乘积在AX 16位：源操作数与AX相乘，乘积在DX:AX 32位：源操作数与EAX相乘，乘积在EDX:EAX 64位：源操作数与RAX相乘，乘积在RDX:RAX
	标志	影响 OF及 CF，其余状态标志ZF、PF、AF、SF不确定。 当结果（乘积）的高半部分=0时，CF←0，OF←0，表示高半部分是无效数字；否则CF←1，OF←1。
	注意	SRC可以是寄存器或存储单元，但不能是立即数。

b) 有符号乘法（1 个操作数）：

有符号乘法	格式	1个操作数：IMUL REG/MEM
	功能	8位：源操作数与AL相乘，乘积在AX 16位：源操作数与AX相乘，乘积在DX:AX 32位：源操作数与EAX相乘，乘积在EDX:EAX 64位：源操作数与RAX相乘，乘积在RDX:RAX
	标志	影响 OF及 CF，其余状态标志ZF、PF、AF、SF不确定。 如果乘积的高半部分不是低半部分的符号扩展（不是全0或全1），则视高半部分为有效位，置CF=1，OF=1； 如果结果的高半部分是全0或全1，表明它仅包含符号位，置CF=0，OF=0；

c) 有符号乘法（2 个操作数）：

有符号乘法	格式	2个操作数：IMUL REG，REG/MEM/IMM
	功能	目的操作数和源操作数相乘，乘积放在目的操作数。
	标志	• 影响 OF及 CF，其余状态标志ZF、PF、AF、SF不确定。 • 当结果必须截断以放在目的操作数时，OF和CF为1，否则为0。
	注意	• 8086不支持 • 目的操作数不能是8位寄存器。 • 立即数不能是64位。 • 源操作数和目的操作数长度相同。当IMM的长度不足时，进行符号扩展。

d) 有符号乘法（3 个操作数）：

有符号乘法	格式	3个操作数：IMUL REG, REG/MEM, IMM
	功能	第2个操作数与IMM相乘，乘积在第1个操作数中。
	标志	• 影响 OF及 CF，其余状态标志ZF、PF、AF、SF不确定。 • 当结果必须截断以放在目的操作数时，OF和CF为1，否则为0。
	注意	• 8086不支持 • 第1个操作数不能是8位寄存器。 • 立即数不能是64位。 • 三个操作数长度相同。当IMM的长度不足时，进行符号扩展。

e) Eg.

i. 例 1、设 AL=55H，BL=14H，计算它们的乘积。

1. MUL BL
2. 结果：AX=06A4H。由于 AH=06H，不为 0，则 CF=1，OF=1。

ii. 例 2、AL=－28H，BL=59H，计算它们的乘积。

1. IMUL BL
2. 结果：AX=0F98CH，CF=1，OF=1。

iii. 不能用 MUL 做有符号乘法：

1. 例、尝试用 MUL 计算 FFH×FFH。用二进制进行计算，可表示为：
2. 1111 1111
3. × 1111 1111
4. 1111 1110 0000 0001
5. 若为无符号数，则相当于 255×255 = 65025 的运算，结果正确。
6. 若为有符号数，则上面的结果表示为（－1）×（－1）＝－511，结果不正确

iv. **IMUL 指令采用什么算法来实现其功能？有多种方案。**

1. 可以直接采用补码相乘。
2. 也可以先将参加运算的操作数恢复成原码，数位当成符号号数相乘，然后给乘积赋予正确的符号。
3. **这些工作由微处理器自动完成。**

9. 除法：

a) 无符号除法：

无符号乘法	格式	DIV REG/MEM
	功能	• 8位：AX除以源操作数，商在AL，余数在AH • 16位：DX:AX除以源操作数，商在AX，余数在DX • 32位：EDX:EAX除以源操作数，商在EAX，余数在EDX • 64位：RDX:RAX除以源操作数，商在RAX，余数在RDX
	标志	所有状态 标志OF、CF、ZF、PF、AF、SF均不确定。

b) 有符号除法：

有符号除法	格式	IDIV REG/MEM
	功能	• 8位：AX除以源操作数，商在AL，余数在AH • 16位：DX:AX除以源操作数，商在AX，余数在DX • 32位：EDX:EAX除以源操作数，商在EAX，余数在EDX • 64位：RDX:RAX除以源操作数，商在RAX，余数在RDX
	标志	所有状态 标志OF、CF、ZF、PF、AF、SF均不确定。
	注意	商的符号符合一般代数符号规则，余数的符号与被除数相同。

c) 除法指令可能发生两种错误

i. 除数为 0（Windows 平台，除数为 0 时，执行时会弹出异常。）

ii. 除法溢出在被除数很大，而除数很小时，会发生除法溢出。例如，AX=3000，除数 BL=2，此时商在 AL=1500 使得除法溢出。（V86 模式下，产生除法出错中断时，显示“Divide Overflow”，且程序退出执行。）

1. Eg.给定无符号数 7A86H 和 04H，求 7A86H÷04H = ？ 。若用 DIV 指令进行计算，即

a) MOV AX, 7A86H

b) MOV BL, 04H

c) DIV BL ； 7A86H÷04H 的商为 1EA1H > FFH

d) 由于 BL 中的除数 04H 为字节，被除数为字，商 1EA1H 大于 AL 中能存放的最大无符号数 FFH，结果将产生除法出错中断。

d) 这两种错误都会使得微处理器产生中断。此时所得的商和余数都不确定。

e) 在任何微处理器中，都不存在立即数除法指令。

10. 乘除法和标志位：

a)

标志：	O	D	I	T	S	Z	A	P	C
IMUL	×	—	—	—	U	U	U	U	X
MUL	×	—	—	—	U	U	U	U	X
IDIV	U	—	—	—	U	U	U	U	U
DIV	U	—	—	—	U	U	U	U	U

b) **X**：根据结果设置。当结果（乘积）的高半部分=0 时，CF←0，OF←0，表示高半部分无有效数字；否则， CF←1，OF←1。**U**：无定义。**–**：不影响。

11. **符号扩展指令**（除法指令中，被除数常常需要进行符号扩展或零扩展。）

a) 字节扩展成字：字扩展成双字：双字扩展成四字：

b)

将字节扩展成字	格式	CBW
	功能	将AL的符号位扩展到AH
将字扩展成双字	格式	CWD
	功能	将AX的符号位扩展到DX
将双字扩展成四字	格式	CDQ
	功能	（386以上）EAX符号位扩展到EDX

c) Eg. 设 AX=379AH。

i. 若执行 CBW 指令，则 AX=FF9AH；

ii. 若执行的是 CWD 指令，则 DX=0000H，AX=379AH。

d) **注意：80386 以上 CPU 还有 MOVSX 指令和 MOVZX 指令。**

12. **Eg.** 二进制四则混合算术运算程序段，试计算：

a) AX = (V – (X*Y + Z-540)) / X 之商，DX = 余数，（其中，X，Y，Z，V 均为字变量、有符号数。）

```
; ( V- (X*Y+Z-540) ) / X
MOV AX, X;
IMUL Y;          X*Y,结果在DX:AX中
MOV CX, AX;
MOV BX, DX;      将乘积存在BX:CX中
MOV AX, Z;
CWD;             将符号扩展后的Z加到BX:CX中的乘积上去
ADD CX, AX;
ADC BX, DX;
SUB  CX, 540;
SBB  BX, 0;      从BX:CX中减去540
MOV AX, V;
CWD;
SUB AX, CX;      从符号扩展后的V中减去(BX:CX)并
SBB DX, BX;      除以X,商在AX中，余数在DX中。
IDIV X;
```

13. 余数：

a) 可以根据实际应用的需求来处理余数。

i. 四舍五入

ii. 截断

iii. 如果是无符号数除法，采取四舍五入方式时，可将余数与除数的一半进行比较，以决定余数是加入到商，还是舍去：

AX 除以 BL，无符号数，结果四舍五入：

```
        DIV BL
        ADD AH, AH
        CMP AH, BL
        JB NEXT
        INC AL
NEXT:
```

14. **BCD 算术运算指令**

a) BCD 算术运算指令不能用于 64 位模式。

b) **BCD 数：**二进制编码的十进制数（Binary Coded Decimal）

i. 用 4 位二进制码表示一位十进制数；

ii. 0000 ~ 1001 是合法 BCD 码；1010 ~ 1111 是非法 BCD 码。

c) **压缩 BCD 数：**用一个字节表示 2 位 BCD 数。例：37=0011 0111

d) **非压缩 BCD 数：**用一个字节的低 4 位表示一位 BCD 数，高 4 位为 0。

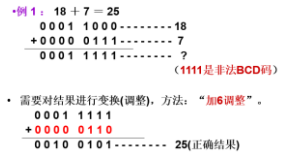
例：37=0000 0011 0000 0111

e) **压缩 BCD 数十进制调整原理**

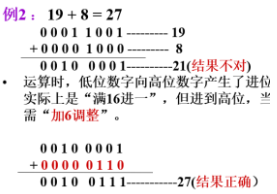
1. 运算时，低位数字向高位数字产生了进位(AF=1 或 CF=1)，实际上是“满 16 进一”，但进到高位，当成了 10，“少 6”，需“加 6 调整”。加法器实际上是按二进制运算“满 16 进一”。但对于 BCD 数，应当按 10 进制算 “满 10 进一”，“进 1 当 10”。

2. 在 BCD 码结果中：

- 若某一位 BCD 数字所对应的二进制码超过 9（1010~1111），应“加上 6”，产生进位，进行调整。



- 若低位数字向高位数字产生了进位(AF=1 或 CF=1)，应“加上 6”，补上少加的“6”，进行调整。这可由软件(调整指令)来完成。



3. **压缩 BCD 数加法十进制调整规则：**如果两个 BCD 数字相加的结果是一个在 1010 ~ 1111 之间的二进制数，或者有向高一位数字的进位（AF=1 或 CF=1），则应在现行数字上加 6（0110B）调整。

4. **压缩 BCD 数减法十进制调整规则：(1)**AF=1，或运算结果的低位是一个在 1010 ~ 1111 之间的二进制数，则在低位上要进行“－6”调整。**(2)** CF=1，或运算结果的高位是一个在 1010 ~ 1111 之间的二进制数，则在高位上要进行“－6”调整。

f) DAA，加法的十进制修正 Decimal Adjust AL after Addition

i. 格式： DAA （必须紧跟在 ADD、ADC 指令后）

ii. 操作：AL←AL 中的和数调整到压缩 BCD 格式

iii. 修正规律：

1. AL 的低 4 位>9 或 AF=1，则 AL←AL+06H，AF←1

2. AL 的高 4 位>9 或 CF=1，则 AL←AL+60H，CF←1

DAA——Decimal Adjust for Addition

标志：O D I T S Z A P C

U ———× ×× ××

iv. 注意：状态位 OF 不确定，其余状态位随运算结果而变。

g) DAS，减法的十进制修正 Decimal Adjust AL after Subtraction

i. 格式： DAS （必须紧跟在 SUB、SBB 指令后）

ii. 操作：AL ← AL 中的差数调整到压缩 BCD 格式。

iii. 修正规律：

1. AL 的低 4 位>9 或 AF=1，则 AL←AL－06H，AF←1
2. AL 的高 4 位>9 或 CF=1，则 AL←AL－60H，CF←1

DAS——Decimal Adjust for Subtraction

标志：**O D I T S Z A P C**

U ———× ××××

iv. 注意事项：状态位 OF 不确定，其余状态位随运算结果而变。

h) Eg. 设 BCD1，BCD2, BCD3 定义为字变量，可分别存放 4 位数字的组合 BCD 数。假定字变量 BCD1 的值为 1834 ,字变量 BCD2 的值为 2789。要求计算 BCD3 = BCD1+BCD2，并指出执行每条指令的操作及执行指令后 AL, AF, CF 的内容。

指令	操作	AL	CF	AF
MOV AL, BYTE PTR BCD1	AL ← 34	34	–	–
ADD AL, BYTE PTR BCD2	AL ← 34+89	BDH	0	0
DAA	调整	23_{BCD}	1	1
MOV BYTE PTR BCD3, AL	(BCD3) ← 23	23_{BCD}	1	1
MOV AL, Byte Ptr BCD1+1	AL←18	18	1	1
ADC AL, Byte Ptr BCD2+1	AL←18+27+CF	40H	0	1
DAA	调整	46_{BCD}	0	1
MOV Byte Ptr BCD3+1, AL	(BCD3+1)←46	46_{BCD}	0	1

15. **压缩 BCD 数的乘除法：**

a) 没有压缩 BCD 数的乘法和除法调整指令。主要原因是相应的调整算法比较复杂，所以不支持压缩 BCD 数的乘除法运算。

b) 如果要处理压缩 BCD 数的乘除法问题，可以把操作数（压缩 BCD 数）变换成相等的二进制数，然后用二进制算法进行运算，运算完成后再将结果转换成 BCD 数。

16. **ASCII 算术运算指令**

a) AAA，加法的 ASCII 修正

i. 格式：AAA（必须紧跟在 ADD，ADC 指令后）

ii. 功能：对 AL 中的非压缩 BCD 数（或十进制的 ASCII 码）的加法结果进行修正。

iii. 修正规律：

1. AL 的低 4 位>9 或 AF=1, 则 AL←AL + 06H, AF←1, AH←AH + 1, AL←AL∧0FH, CF←AF，其中 AH=AH+1 用来实现低位 BCD 数向高位的进位。
2. AL 的低 4 位<9 且 AF=0，AL←AL∧0FH，CF←AF。

iv. 注意：状态位 AF、CF 随操作数结果变化；其余状态位都是不确定的。

b) AAS，减法的 ASCII 修正

i. 格式：AAS（必须紧跟在 SUB，SBB 指令后）

ii. 功能：对 AL 中的非压缩 BCD 数（或十进制的 ASCII 码）的减法结果进行修正。

iii. 修正规律：

1. AL 的低 4 位>9 或 AF=1, 则 AL←AL－06H, AF←1, AH←AH－1, AL←AL∧0FH, CF←AF，其中 AH=AH－1 用来实现低位 BCD 数向高位的借位。
2. AL 的低 4 位<9 且 AF=0，AL←AL∧0FH，CF←AF。

iv. 注意：状态位 AF、CF 随操作数结果变化；其余状态位都是不确定的。

c) AAM，乘法的 ASCII 修正

i. 格式：AAM（必须紧跟在 MUL 指令后）

ii. 功能：操作数为累加器 AX，对 AL 中的非压缩 BCD 数的乘法结果进行修正。

iii. 修正规律：AH = AL/10 的商（高位非压缩 BCD 数），AL = AL/10 的余数（低位非压缩 BCD 数）。

iv. 注意：状态位 SF、ZF、PF 随操作结果变化；其余状态位都是不确定的。

d) AAD，除法的 ASCII 修正

i. 格式：AAD（必须紧跟在 DIV 指令**前**）

ii. 功能：操作数为累加器 AX，AX 的内容为两位非压缩的 BCD 数；在做除法前，对

AX 中的非压缩 BCD 数进行修正。

iii. 修正规律：AL = AH×10 + AL，AH = 0。

iv. 本质：把 BCD 码转换成二进制数。

v. 注意：状态位 SF、ZF、PF 随操作结果变化；其余状态位都是不确定的。

17. 逻辑运算指令

a) DST 可以是 reg，mem；

b) SRC 可以是 reg，mem，或 imm。

c) **AND DST, SRC，OR DST, DST 和 XOR　DST, SRC**

i. DST 和 SRC 不能同时为 mem。

ii. 状态位 SF、ZF 和 PF 随运算结果而变化，CF←0，OF←0，而 AF 不确定。

d) **NOT DST**

i. 注意：NOT 不影响标志位。

e) **TEST DST, SRC**

i. 执行 DST∧ SRC 操作后，两个操作数内容不变。

ii. DST 和 SRC 不能同时为 mem。

iii. 状态 SF、ZF 和 PF 随运算结果而变化，CF←0，OF←0，而 AF 不确定。

f) 别的测试指令：

位测试	格式	**BT REG/MEM, REG/IMM8**
	功能	测试目的操作数中的某一位，测试结果放入**CF**标志
位测试并取反	格式	**BTC REG/MEM, REG/IMM8**
	功能	测试目的操作数中的某一位，测试结果放入**CF**标志，并将测试位取反
位测试并清零	格式	**BTR REG/MEM, REG/IMM8**
	功能	测试目的操作数中的某一位，测试结果放入**CF**标志，并将测试位清零
位测试并置位	格式	**BTS REG/MEM, REG/IMM8**
	功能	测试目的操作数中的某一位，测试结果放入**CF**标志，并将测试位置位

BT AX, 4；如果第 4 位为 1，则 CF=1，否则 CF=0

18. 位移指令

a) 逻辑移位：把操作数作为无符号数进行移位。

i. 右移时，最高位补 0；逻辑右移　SHR REG/MEM, CL/IMM8

ii. 左移时，最低位补 0。逻辑左移　SHL REG/MEM, CL/IMM8

b) 算术移位：把操作数作为有符号数进行移位。

i. 右移时，最高位保持不变；算术右移　SAR REG/MEM, CL/IMM8（算术右移时，符号位保持不变。）

ii. 左移时，最低位补 0。算术左移　SAL REG/MEM, CL/IMM8（SHL 和 SAL 功能一样。）

c) 8086CPU 中，IMM8 只能是 1。

d) 8086CPU 不对移位次数取模。32 位 CPU 对移位次数取 32 的模。对于 64 位目的操作数，取 64 的模。

e) 状态位 SF、ZF 和 PF 随运算结果变化。AF 不确定。

f) 状态位 CF：

i. 最后一次移入到 CF 中的值。

ii. 对于 SHL 和 SHR，当移位次数超过目的操作数长度时，CF 值不确定。

g) 状态位 OF:

i. 左移 1 位时，结果的最高位(即符号位)与 CF 一致，则 OF=0，否则为 1。SAR 右移，OF=0；SHR 右移，OF=源操作数的最高有效位。

h) 左移/右移多位时，OF 值不确定。

19. **双精度移位指令**

a)

双精度左移	格式	SHLD REG/MEM, REG, CL/IMM8
	功能	第一操作数向左移，高位依次移入CF，其“空出”的低位由第二操作数的高位来填补，但第二操作数自己不移动、不改变。
	标志	CF、OF、PF、SF和ZF受影响，AF无定义。

双精度右移	格式	SHRD REG/MEM, REG, CL/IMM8
	功能	第一操作数向右移，低位依次移入CF，其“空出”的高位由第二操作数的低位来填补，但第二操作数自己也不移动、不改变。
	标志	CF、OF、PF、SF和ZF受影响，AF无定义。

20. **循环移位指令**

a) 不带进位的循环左移：ROL REG/MEM, CL/IMM8

b) 不带进位的循环右移：ROR REG/MEM, CL/IMM8

c) 带进位的循环左移：RCL REG/MEM, CL/IMM8

d) 带进位的循环右移：RCR REG/MEM, CL/IMM8

e) 状态位 SF、ZF、、AF 不受影响。

f) 状态位 CF：最后一次移入到 CF 中的值。

g) 状态位 OF:

i. 左移 1 位时，结果的最高位(即符号位)与 CF 一致，则 OF=0，否则为 1。

ii. 右移 1 位时，结果的最高 2 位的异或值。

iii. 左移/右移多位时，OF 值不确定。

21. **位扫描指令**

a) **BSF：Bit scan forward，向前位扫描指令**

向前位扫描	格式	BSF REG, REG/MEM
	功能	在源操作数中搜索值为1的最低位。如果找到，则将位索引存储到目标操作数。位索引是从0 算起的无符号偏移量。如果源操作数的内容为 0，则目标操作数的内容未定义。
	标志	如果源操作数的所有位都是 0，则 ZF=1；否则 ZF=0。CF、OF、SF、AF 及 PF 标志未定义。

b) **BSR：Bit scan reverse，向后位扫描指令**

向后位扫描	格式	BSR REG, REG/MEM
	功能	在源操作数中搜索值为1的最高位。如果找到，则将位索引存储到目标操作数。位索引是从0 算起的无符号偏移量。如果源操作数的内容为 0，则目标操作数的内容未定义。
	标志	如果源操作数的所有位都是 0，则 ZF=1；否则 ZF=0。CF、OF、SF、AF 及 PF 标志未定义。

c) BSF 和 BSR 的用途：为位操作指令寻址值为 1 的位。

d) 例，设 EAX=60000000H。

e) 如果执行 BSF EBX, EAX，则 EBX=29，ZF=0。

f) 如果执行 BSR EBX, EAX，则 EBX=30，ZF=0.

22. **串比较指令**

a) **SCAS：String Scan，串扫描**

串扫描	格式	**SCAS MEM 或 SCASB / SCASW / SCASD / SCASQ**
	功能	比较**ES:DI**或**ES:EDI**或**RDI**指定的字节、字或双字与 **AL**、**AX** 或 **EAX**或**RAX**中的值，并根据结果设置状态标志。比较之后，根据 **DF** 标志，**DI**、**EDI**或**RDI**自动递增或递减**1**、**2**、**4**或**8**。 **ES** 段不能使用跨段前缀覆盖。
	标志	**OF**、**SF**、**ZF**、**AF PF** 及 **CF** 标志根据比较的临时结果设置。

i. 通过 REP 前缀，SCAS、SCASB、SCASW 及 SCASD 指令可用于整块比较 CX 个字节、字或双字或四字。

ii. 假定从 BLOCK 开始的存储区域长为 100 字节，要求测试该存储区域，查看哪个单元有 00H。

iii. MOV DI, OFFSET BLOCK

iv. CLD

v. MOV CX, 100

vi. XOR AL, AL

vii. REPNE SCASB

b) **CMPS：String Compare，串比较**

串比较	格式	**CMPS MEM, MEM 或 CMPSB / CMPSW / CMPSD / CMPSQ**
	功能	比较**DS:SI**与**ES:DI**（或**DS:ESI**与**ES:EDI**，**RSI**与**RDI**）指定的字节、字、双字或四字的值，并根据结果设置状态标志。比较之后，根据 **DF** 标志，**DI**、**EDI**或**RDI**自动递增或递减**1**、**2**、**4**或**8**。 **ES** 段不能使用跨段前缀覆盖。
	标志	**OF**、**SF**、**ZF**、**AF PF** 及 **CF** 标志根据比较的临时结果设置。

i. 在 CMPS、CMPSB、CMPSW、CMPSD 及 CMPSQ 前面增加 REP 前缀，可整块比较 CX 个字节、字或双字或四字。

ii. 假定 LINE 和 TABLE 分别指向两段存储区域，要求检查它们的内容是否相同。

iii. MOV SI, OFFSET LINE

iv. MOV DI, OFFSET TABLE

v. CLD

vi. MOV CX, 10

vii. REPE CMPSB

23. REP 前缀

a) REP 前缀指令

i. 常用格式：REP MOVS/LODS/STOS

ii. 若 CX≠0，重复执行，每执行一次 CX=CX-1

iii. 若 CX=0，则退出重复，结束串操作。

iv. **REP MOVSB 指令：先检查 CX 是否等于 0，然后再执行 MOVSB 。若 CX=0，则一次都不执行 MOVSB ，也不会执行 CX=CX-1 的操作。**

b) REPE/REPZ 前缀指令

i. 常用格式：REPE/REPZ CMPS/SCAS

ii. CX≠0 且 ZF=1，重复执行，每执行一次 CX=CX-1

iii. CX=0 或 ZF=0，则停止重复执行。

c) REPNE/REPNZ 前缀指令

i. 常用格式：REPNE/REPNZ CMPS/SCAS

ii. CX≠0 且 ZF=0 重复执行，每执行一次 CX=CX-1

iii. CX=0 或 ZF=1，则停止重复执行。

d) 前缀指令本身不影响状态位。

六、第六章

1. 转移类型与寻址方式

a) 段内转移：同一个段，**只改变 IP/EIP/RIP**

i. near 类型：16 位，或 32 位，或 64 位偏移量（64 位模式，实际是 40 位）

ii. short 类型：8 位（是 near 类型的一个特例）

b) 段间转移：不同段，**改变 CS: IP/EIP/RIP**

i. far 类型

2. 无条件转移指令 JMP

a) 无条件将程序转移到指令指定的目的操作数。

b) 不记录返回地址信息。

c) JMP 指令可以实现**段内转移**和**段间转移**。

d) JMP 指令的操作数可以是**立即数**、**通用寄存器**、**存储器地址**。

e) 段内转移：

i.

寻址方式	操作数类型		操作数的使用方式	指令实例
直接	标号	1字节立即数	加入 IP/EIP/RIP	JMP SHORT START
		2字节立即数		JMP START1
		4字节立即数		JMP START2
	$	立即数		JMP $+2
间接	寄存器操作数		送入 IP/EIP/RIP	JMP BX 或 JMP EBX JMP RBX
	存储器操作数			JMP JTABLE[BX]

ii. START 和 START1、START2 是转移目的标号（符号地址， NEAR 类型）

iii. START 指示的目的地址与当前地址间的转移范围在-128~+127 个字节内。

iv. JTABLE 是变量，类型为 WORD（实模式），DWORD（保护模式）、QWORD（保护模式）。

f) 段间转移：

i.

寻址方式	操作数类型		操作数的使用方式	指令实例
直接	标号	4字节立即数	送入CS和 IP/EIP/RIP	JMP START3
		6字节立即数		
		10字节立即数		
间接	存储器操作数		送入CS和 IP/EIP/RIP	JMP JTABLE1[BX]

ii. START3 是标号，类型是 FAR。

iii. JTABLE1 是变量，类型为 DWORD（实模式），FWORD（保护模式）、TWORD（64 模式） 。（换行）对于位移量为 8 位的短转移，在标号前可以加说明符 SHORT，也可以省略不写。

g) 对于位移量位 16 位的近转移，在标号前可以加说明符 NEAR PTR，也可以省略不写。

h) 默认情况下，代码标号（标号后跟**单个冒号**）有一个局部域，对其所在过程内的语句可见，这阻止了跳转或循环语句转移到当前过程之外的标号。

i) 少数情况下，如果必须将控制转移到当前过程之外的标号处，标号必须被声明为全局的。声明全局标号，要在标号后跟**两个冒号**。

例，全局标号和局部标号的使用。

```
MAIN PROC
      JMP L2 ;错误!
 L1:: ..... ;全局标号
      .....
      RET
MAIN ENDP

SUB PROC
 L2: ..... ;局部标号
     JMP L1 ;正确
      RET
SUB ENDP
```

3. **条件转移指令：**
   a) 条件转移指令共计 **21 条**，这些指令根据上一条指令执行后处理器的状态标志，确定程序的执行方向。
   b) 转移范围：
      i. 对于 16 位微机，均为短转移：目的地址必须在当前段内，且与下一条指令的第一个字节的距离在 – 128 ~ 127 内。
      ii. 对于 80386 以上微处理器，为近转移（±32KB 范围）。
      iii. 在 Pentium4 的 64 位模式下，为近转移（±2GB 范围）
   c) 均为**直接转移**：使用标号地址，机器码中为相对位移量 disp。
   d) 条件转移指令不影响状态位。
   e) **直接标志转移：**这类指令在助记符中直接给出标志状态的测试条件，如 jc、jnc、jz、jnz。
   f) **间接标志转移：**这类指令在助记符中不直接给出标志状态的测试条件，但仍以某一个或某几个标志的状态作为测试条件。
      i. 无符号数：JA：高于/不低于等于，JB：低于/不高于等于，…
      ii. 有符号数：JG：大于/不小于等于，JL：小于/不大于等于，…
4. **条件设置指令**
   a) 条件设置指令的功能：根据对条件进行测试的结果，或者把一个字节设置为 01H，或者把该字节清除为 00H。
   b) 有近 20 条条件设置指令，格式类似。
   c) 以 SETC 为例：
      i. 格式：SETC REG8/MEM8
      ii. 功能：如果进位标志位 1，则 REG8/MEM8 置为 1，否则为 0。
   d) **条件转移指令要测试的条件可以由条件设置指令来建立。**
5. **循环控制指令**

格式	**LOOP DEST**	
功能	**8086~80286**	**CX ← CX-1，CX**不为**0**，则转移到**DEST**，否则顺序执行。
	80386~Core2	循环计数用**CX**（**16**位指令模式）或**ECX**（**32**位指令模式）； **LOOPW**使用**CX**； **LOOPD**使用**ECX**。
	64位模式	循环计数用**RCX**
标志	不影响状态位。状态位并不受**LOOP**指令中的“**CX－1**”的影响。因此，**ZF=1**时，**CX**未必为**0**。	

   a)
   b) **条件循环指令：**
      i. **为零(相等)循环** ： LOOPE/LOOPZ DEST
         1. CX≠0 且 ZF=1 时，转到 DST 所指指令。
      ii. **非零(不相等)循环** ： LOOPNE/LOOPNZ DEST
         1. CX≠0 且 ZF=0 时，转到 DST 所指指令。
   c) 8086~80286，使用 CX；80386~Core2，16 位指令模式使用 CX；32 位指令模式，使用 ECX；64 位模式，使用 RCX。
   d) 类似于 LOOP，也有 LOOPEW、LOOPED、LOOPNEW、LOOPNED 指令。
6. **.IF 语句**

a) 格式

i. .IF 表达式 1

（汇编语言语句组 1）

.ELSEIF 表达式 2

（汇编语言语句组 2）

.ELSEIF 表达式 3

（汇编语言语句组 3）

……

.ELSE

（汇编语言语句组 n）

.ENDIF

7. 过程

a) 近过程，段内调用

b) 远过程，段间调用

8. **CALL 指令**

a) 近 CALL 调用

i. 段内调用，将下一条指令的偏移地址（IP/EIP/RIP）压入堆栈。

ii. 例，CALL SORT；//设 SORT 是近过程

b) 远 CALL 调用

i. 段间调用，将将下一条指令的段基址（CS）和偏移地址（ IP/EIP/RIP ）压入堆栈。

ii. 例，CALL COS；//设 COS 是远过程

c) CALL 指令也可以使用寄存器操作数

i. 例，CALL BX，其功能是将 IP 入栈，并跳转到当前代码段以 BX 内容为偏移地址的地方继续执行。

d) CALL 指令也可以使用间接存储器寻址的操作数

i. 例，CALL TABLE [4*EBX]，从数据段 EBX 寻址的存储单元得到的数据，作为过程的起始地址。

9. RET 指令

a) 近返回：从栈顶取出偏移地址。

b) 远返回：从栈顶取出段基址和偏移地址。

c) 带参数的 RET 指令

i. 格式：RET n

ii. 功能：从栈顶弹出返回地址后，将堆栈指针（SP）的内容加上一个数值 n。

iii. 用途：调用过程前先把参数压入堆栈，如果返回时要丢弃这些参数，可以采用这种形式。非常适用于那些用 C/C++或 PASCAL 调用规则的系统。

10. **中断概述**

a) 中断的目的：在与低速 I/O 设备进行数据传输时，中断特别有用。

b) 中断引脚

i. 整个 Intel 系列微处理器的中断包括：

1. 2 个申请中断的硬件引脚：INTR 和 NMI

2. 1 个相应 INTR 中断申请的硬件引脚：INTA

ii. 注意：除了这些引脚外，微处理器还有：

1. 软件中断指令 INT、INTO、INT3 和 BOUND；

2. 标志位 IF（Interrupt Flag）和 TF（Trap Flag）；
3. 中断返回指令 IRET（或在 80386~Pentium4 中的 IRETD）。

c) 中断的产生

i. **硬件产生**（Hardware-generated），外部中断
   1. NMI 引脚
   2. INTR 引脚（可屏蔽中断）

ii. **软件产生**（Software-generated），内部中断
   1. 用于解决 CPU 在运行过程中发生的一些意外情况。
   2. 例如，除零或商溢出。

iii. 通常，**内部中断称为异常**。

iv. 任何类型的中断都是通过调用**中断服务程序**(ISP, Interrupt Service Procedure)来使当前程序暂停执行。

d) 中断及中断返回

i. CPU 每响应一次中断：
   1. 不但要像过程调用指令那样，把 CS 和 IP（或 EIP/RIP）寄存器的值（即断点）送入堆栈保存，而且还要将标志寄存器的值入栈保护，以便在中断服务程序执行完后，能够正确恢复 CPU 的状态。
   2. 根据中断类型号（0～255），找到中断服务程序的入口地址，转相应的中断服务程序。
   3. 中断服务程序结束后，通过中断返回指令 IRET，从堆栈中恢复中断前 CPU 的状态和断点，返回原来的程序继续执行。

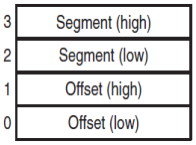
e) 中断向量

i. **中断向量表（IVT，Interrupt Vector Table）**为于存储器的最低 1024 字节，地址为 0000H~03FFH。

ii. 包含 256 个不同的 4 字节中断向量。

iii. 每个中断向量保护一个中断服务程序的入口地址（段基址和偏移量）。

iv. 微处理器按实模式操作时，中断向量（Interrupt Vector）是 4 个字节的数据，存放在存储器的第一个 1024 单元。



v. 在保护模式下，用中断描述符表代替向量表，每个中断用 8 个字节的中断描述符说明。

vi. 前 32 个中断向量是 Intel 保留的，其余的中断向量是用户可用。

vii. 前 5 个中断向量在所有 Intel 系列微处理器中都是相同的。其他中断向量存在于 80286 及向上兼容的 80386~ Core2 中，但不向下兼容 8086 或 8088。

viii. 类型 0～类型 4 中断：
   1. 类型 0：除法错中断（除数为 0 或商超过了寄存器能容纳的范围，自动产生）
   2. 类型 1：单步中断
   3. 类型 2：不可屏蔽中断
   4. 类型 3：断点中断（断点可以设置在程序中的任何地方，设置方法是插入一条 INT 3 指令）
   5. 类型 4：溢出中断（若溢出标志 OF 置 1，可由 INTO 指令产生类型为 4 的中断）

f) 中断与中断返回指令

i. INT 指令
   1. INT n （n 为中断类型码）

2. n 为中断类型号，可以为 0~255。INT n 可以在编程时安排在程序中的任何位置上。
3. 原则上讲，用 INT n 指令可以调用所有 256 个中断，尽管其中有些中断是硬件触发的。
4. 程序中需要调用某一类型的中断服务程序时，可通过插入 INT n 指令来实现；也可利用 INT n 指令来调试各种中断服务程序。
5. 例如，可用 INT 0 让 CPU 执行除法出错中断服务程序，而不必运行除法程序；可用 INT 2 指令执行 NMI 中断服务程序，从而不必在 NMI 引脚上加外部信号，就可对 NMI 子程序进行调试。
6. 功能调用：
   a) INT 16：BIOS 服务
   b) INT 21：DOS 服务

ii. INT3 指令
1. INT3 指令
2. 格式：INT3
3. 功能：同“INT 3”

iii. INTO 指令
1. 格式：INTO
2. 功能：同“INT 4”
3. 当带符号数进行算术运算时，如果 OF = 1，可由 INTO 产生溢出中断处理；若 OF=0，则 INTO 指令不产生中断。如果程序中无 INTO，溢出异常被忽略。
4. 因此，有符号数加减运算后，必须使用 INTO，一旦溢出就能及时向 CPU 提出中断请求，如显示出错信息。溢出中断处理完后，CPU 将不返回原程序继续执行，而是把控制权交给操作系统。

iv. IRET、IRETD、IRETQ 指令
1. IRET：实模式中断返回
   a) 总是被安排在中断服务程序的出口处。
   b) 当 IRET 执行后，首先从堆栈中依次弹出程序断点（送入 IP 和 CS），接着弹出标志寄存器；然后按 CS:IP 的值使 CPU 返回断点继续执行
   c) **IRET 相当于：先 RET，再 POPF。**
2. 保护模式：IRETD
3. 64 位模式：IRETQ

g) 中断控制

i. 硬件产生的外部中断有两个来源：
1. NMI 引脚
2. INTR 引脚（可屏蔽中断）

ii. **控制 INTR 引脚的指令有两条：STI、CLI。**

iii. STI 指令
1. 格式：STI
2. 功能：设置中断允许标志，将 IF 置 1，允许 INTR 输入。

iv. CLI 指令
1. 格式：CLI
2. 功能：清除中断允许标志，将 IF 清零，禁止 INTR 输入。

h) PC 机的中断

i. 早期的 PC 机是基于 8086/8088 的系统，Intel 保留的中断只包含 0~4 号中断。
ii. 早期的 16 位微机使用中断向量表。
iii. Windows 平台上访问保护模式中断结构，要通过 Microsoft 提供的内核调用功能而不能直接寻址。
iv. 保护模式中断使用中断描述符表。

i) 实模式中断操作

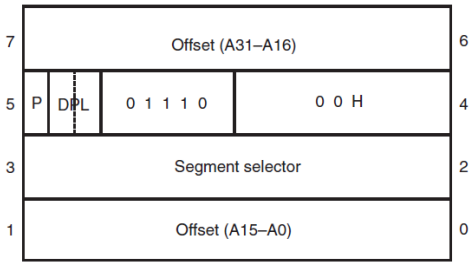
i. 当微处理器执行完当前指令后，按下列顺序**判断是否有中断发生**：
   1. 指令执行情况；
   2. 单步中断；
   3. NMI；
   4. 协处理器段超限；
   5. INTR；
   6. INT 指令。

ii. **如果有中断发生，则按下列顺序处理：**
   1. 将标志寄存器入栈；
   2. 清除 IF、TF 标志；
   3. CS 入栈；
   4. 指令指针 IP 入栈；
   5. 取出中断向量内容，送入 IP 和 CS。

iii. **关于返回地址：**
   1. 有时，返回地址为程序中的下一条指令。
   2. 有时指向程序中发生中断的地方。
   3. 中断类型 0、5、6、7、8、10、11、12 和 13 压入堆栈的返回地址是指向错误指令，而不是下一条指令。
   4. 使得中断服务程序在某些错误情况下有可能重新执行该指令。
   5. 一些保护模式中断（类型 8、10、11、12 和 13）将错误代码紧跟返回地址压入堆栈。错误代码识别引起中的选择符（Selector）。如果不包括选择符，则错误代码为 0。

j) 保护模式中断操作

i. 保护模式下的中断与实模式几乎完全相同，但中断向量表不同。
ii. 保护模式使用一组存储在中断描述符表（IDT）中的 256 个中断描述符取得中断向量。每个描述符占 8 个字节。
iii. 中断描述符表占 256×8=2KB 字节。
iv. 中断服务程序的地址：Segment Selector 和 Offset
v. 描述符是在内存中：P
vi. 描述符特权级：DPL



k) 实模式的中断向量可以转换成保护模式中断。

i. 复制中断向量表中的中断服务程序地址，并将其转换成存储于中断描述符中的 32 位偏移地址。
ii. 全局描述符表将存储器的前 1MB 标识为中断段，相应的选择符和段描述符可放在全局描述符表中。

l) **中断标志位**

例，置位TF的一个中断服务程序。

```
TRON PROC FAR USES AX BP
     MOV BP, SP
     MOV AX, [BP+8]
     OR  AH, 1
     MOV [BP+8], AX
     IRET
TRON EDNP
```

例，清除TF的一个中断服务程序。

```
TRON PROC FAR USES AX BP
     MOV BP, SP
     MOV AX, [BP+8]
     AND AH, 0FEH
     MOV [BP+8], AX
     IRET
TRON EDNP
```

i. **IF：中断允许标志**

1. STI、CLI 指令

ii. **TF：陷阱标志**

1. 当 TF=1，它在每条指令执行之后产生一个陷阱中断（类型 1）。这也是我们常成陷阱中断为单步中断的原因。
2. 当 TF=0，程序正常执行。
3. 没有特殊的指令来置位和清除陷阱标志。
4. PUSHF/PUSHFD，POPF/POPFD

m) **硬件中断**

i. NMI：一旦激活 NMI 输入，就发生类型 2 中断。

ii. INTR：INTR 的输入必须外部译码，以选择一个中断向量。

1. INTR 引脚可以选择任何中断向量，但通常只使用 20H~FFH 之间的中断向量。
2. Intel 保留 00H~1FH 之间的中断。

iii. INTA#：用于响应 INTR 输入的一个输出引脚，将向量类型号加载到数据总线 D7~D0 上。

iv. NMI：边沿触发，在上升沿申请中断。

1. 在上升沿之后，NMI 引脚必须保持逻辑 1，直到微处理器识别它。
2. 在上升沿被识别之前，NMI 引脚必须保持逻辑 1 至少 2 个时钟周期。
3. NMI 常用于奇偶校验错误和其他主要系统故障（如掉电）。
4. 响应这种类型的中断时，微处理器将所有内部寄存器存于使用电池的备份存储器或 EEPROM 中。

11. **机器控制及其他指令**

a) **控制进位标志**

i. STC：将 CF 置 1

ii. CLC：将 CF 清 0

iii. CMC：将 CF 取反

b) **WAIT 指令**

i. 监控 8086 上的硬件引脚 TEST#、286 和 386 上的硬件引脚 BUSY#。80486~Core2 没有相应引脚。

ii. 如果 WAIT 指令执行时，BUSY#=1，则继续执行下一条指令；如果 BUSY#=0，则微处理器要等待，直到 BUSY#=1。

c) **HLT 指令**

i. CPU 暂停，直到有复位（Reset）信号或外部中断请求时退出暂停状态。

ii. 在 RESET 上加复位信号。

iii. 在 NMI 引脚上出现中断请求。在允许中断的情况下，在 INTR 上出现中断请求信号。

iv. 出现 DMA 操作。

v. 在程序中，通常用 HLT 指令来等待中断的出现。

vi. 因为 DOS 和 Windows 大量使用中断，HLT 并不停机。

d) **NOP 指令**

i. 这是一条单字节指令，执行时需耗费 3 个时钟周期的时间，但不完成任何操作。

e) **LOCK 前缀**

i. 封锁总线指令，禁止其他主控设备使用总线。
ii. 是一种前缀，可加在任何指令的前端，用来维持总线封锁引脚 LOCK#有效。
iii. 例，LOCK: MOV AL, [SI]

f) **ESC 指令**
i. 转义指令，从微处理器向浮点协处理器传递指令。
ii. 协处理器从 ESC 指令获得其操作码，并开始执行协处理器指令。
iii. ESC 从来不在程序中出现。当协处理器指令出现时，汇编程序把它们看做是协处理器的 ESC。

g) **BOUND 指令**
i. 格式：BOUND reg, src
ii. （80186 以上 CPU）检查数组边界，reg 是 16 位或 32 为寄存器，src 为内存中的两个字或双字，是被检查数组的上限和下限。该指令将 reg 中的值与 src 中的值进行比较，若 reg 的值在 src 的上下限之间，则继续执行下一条指令，否则产生 5 号中断。
iii. 注意：该中断的返回地址是 BOUND 指令的地址。
iv. 例，BOUND SI, DATA
v. 下界在存储单元 DATA，上界在存储单元 DATA+2。

h) **ENTER 和 LEAVE 指令**
i. 80186 以上支持。
ii. ENTER 指令（使用时）通常是过程中的第一条指令，用于为过程建立新的堆栈帧。
iii. 在过程的末尾（就在 RET 指令的前面），使用 LEAVE 指令释放堆栈帧，恢复 SP 和 BP。

i) **ENTER 指令格式：ENTER data16, data8**
i. ENTER 指令通常是进入过程时要执行的第一条指令，为过程创建堆栈帧。第 1 个操作数（大小操作数）指定堆栈帧的大小（即堆栈上给过程分配的动态存储空间字节数）。第 2 个操作数（嵌套层数操作数）给出过程的词法嵌套层级（0~31）。这两个操作数都是立即数。
ii. 嵌套层级确定要从前面的帧复制到新堆栈帧“显示区”的堆栈帧指针数。若嵌套层级为 0，则处理器将帧指针 BP /EBP 压入堆栈，将当前堆栈指针 ESP 复制到 BP/EBP，并将当前堆栈指针值减去大小操作数中的值之后的结果加载到 SP/ESP。如果嵌套层级大于或等于 1，则处理器在调整堆栈指针之前，先将其它帧指针压入堆栈。这些额外的帧指针为被调用过程访问堆栈上的其它嵌套帧提供访问点。

七、 第八章

1. PULBIC 和 EXTRN

a) **PUBLIC：**将指令标号、变量名称、段名声明为对其他程序模块可用。
b) **EXTRN：**将模块中使用的一些标号声明为外部的。
c) 没有 PUBLIC、EXTRN，各模块程序就不能链接在一起从而创建一个程序。或许它们会产生链接，但模块之间是不通信的。

2. 宏

a) **宏的定义：**使用 MACRO 和 ENDM 伪指令。
b) **宏的调用：**宏名和参数。
c) 宏可以包含作为局部变量的标号（最多 35 个），用 LOCAL 伪指令声明。一定要用

LOCAL 伪指令说明为局部标号，以免多次调用宏时，发生标号重复定义错误。

d) 将宏定义放入模块中，用 INCLUDE 语句。

e) 例，如果文件 MACRO1.MAC 包含一组宏，将其放入程序文件时：

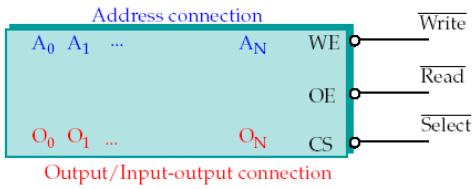
i. INCLUDE C:\ASSM\MACRO.MAC

f) 宏序列常用 INC 或 MAC 作为扩展名。

八、 第十章

1. 存储器器件

a) 引脚



i.

ii. **地址线：地址引脚数量**与存储单元个数有关，一般有 1M-64GB 个位置，因此有 20-36 个引脚。

iii. **数据线：数据引脚数量**与一个地址的大小有关

iv. **控制线：**CS（chip select）：片选，CE（chip enable）：片使能

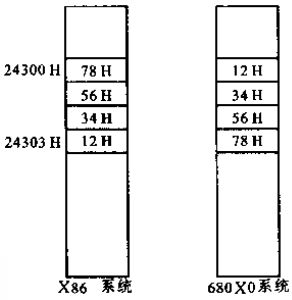
2. 存储器的数据组织

a) **存储字**：计算机系统中，作为一个整体一次存放和取出内存储器的数据称为“存储字”。

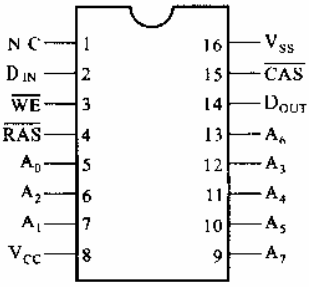
b) **字节编址**：一个存储地址对应一个 8 位存储单元。

c) Intel x86：低地址，低字节

d) Motorola 680X0：低地址，高字节



3. DRAM 芯片



地址总线：A_0~A_7

行地址,列地址选择：RAS#，CAS#

读写控制：WE#

数据输入/输出：D_{IN}，D_{OUT}

V_{CC}，V_{SS}

NC

a)

b) 容量：64K×1 位

c) 存取时间：150ns/200ns

d) 每 2ms 需刷新一遍，每次刷新 512 个单元。

九、 第十一章

1. I/O 指令

a) I/O 设备与累加器（AL、AX、EAX）之间传送数据：IN，OUT 指令。

b) I/O 设备与存储器之间传送数据：

i. INS，INSB，INSW，INSD 指令。

ii. OUTS，OUTSB，OUTSW，OUTSD 指令。

iii. INS 和 OUTS 指令前可以加 REP 前缀。

iv. 除 8086/8088 外，Intel 微处理器都支持 INS/OUTS。

c) Pentium4 和 Core2 的 64 位模式下，并没有 64 位的 I/O 指令。

d) **I/O 端口的位宽是 8 位。**

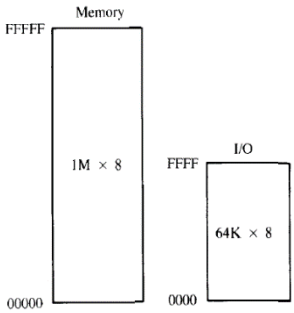
e) 任意两个 8 位的连续编址端口是一个 16 位的端口。

f) 任意四个 8 位的连续编址端口是一个 32 位的端口。

g) 例，以字的方式访问端口 100H，实际上就是访问了 100H 和 101H 两个端口。端口 100H 包含数据的低 8 位，端口 101H 包含数据的高 8 位。

h) 与存储器类似，16 位端口地址应当对齐到偶地址（0，2，4，……），32 位端口地址应当对齐到 4 的倍数（0，4，8，……）。

i) CPU 支持非对齐端口的读写，但要增加额外的总线周期。



2. **连接 I/O 与微处理器**

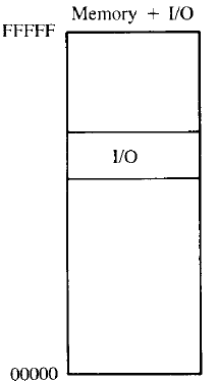
a) **独立编址 I/O：**I/O 端口和存储器的地址空间相隔离。

i. 优点：

1. 存储器同 I/O 端口的操作指令不同，程序比较清晰；
2. 存储器和 I/O 端口的控制结构相互独立，可以分别设计。

ii. 缺点：

1. I/O 与微处理器之间传送的数据必须用 IN、OUT、INS、OUTS 指令存取。



b) 存储器映像 I/O：存储器映像 I/O 设备被视为存储器映像中的一个存储单元。不 IN、OUT、INS、OUTS 使用指令。

i. 优点：

1. 任何存储器传送指令都可用来访问 I/O 设备。
2. IORC#和 IOWC#信号在存储器映像 I/O 系统中不起作用，可减少译码所需电路的数量。

ii. 缺点：

1. 一部分存储器被用作 I/O 映像，减少了可用存储器的数量。

c) **PC 机 I/O 映像**

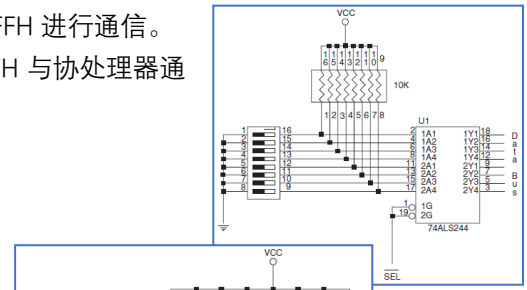
i. PC 机的部分 I/O 映像用于专用功能。

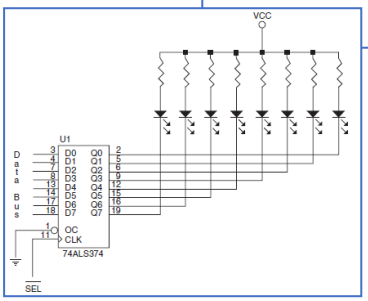
ii. 端口 0000H~03FFH 之间的 I/O 空间通常留给计算机系统和 ISA 总线。

iii. 端口 0400H~FFFFH 之间的 I/O 空间通常给用户应用、主板功能及 PCI 总线。

iv. 80287 算术运算协处理器使用 I/O 地址 00F8H~00FFH 进行通信。

v. 80386~Core2 使用 I/O 端口 800000F8H~800000FFH 与协处理器通信





3. **基本输入输出接口**

a) **基本输入设备：一组三态缓冲器。(上面那个)**

i. 该基本输入电路并不是可有可无的。只要输入数据接到微处理器上，就必须有此电路。

ii. 有时，它作为独立的电路出现，有时被包含在一个可编程 I/O 设备中。

b) **基本输出设备：一组数据锁存器。(下面这个)**

i. 通常，必须为某个外部设备保持数据。

ii. 有时，它作为独立的电路出现，有时被包含在 I/O 设备内部。

4. 握手

a) 许多 I/O 设备接收或发送信息的速度比 CPU 慢很多。

b) 此时，常用的 I/O 控制方法为握手（handshaking）或查询（polling），以使得 I/O 设备与 CPU 同步。

c) 例，打印机与微处理器之间的数据通信。

d) 数据通过数据线（D7~D0）传送；BUSY 指示打印机忙；STB#是时钟脉冲，用于发送数据给打印机打印。

e) **例，打印机与微处理器之间的数据通信。一个打印 BL 中 ASCII 内容的汇编语言程序。**

```
i.    PRINT PROC NEAR
ii.       .REPEAT
iii.          IN AL, BUSY；测试忙标志
iv.           TEST AL, BUSY_BIT
v.        .UNTIL ZERO
vi.       MOV AL, BL
vii.      OUT PRINTER, AL；把数据送到打印机
viii.     RET
ix.   PRINT ENDP
```

5. **关于接口电路的注释**

a) 输入设备：电平转换

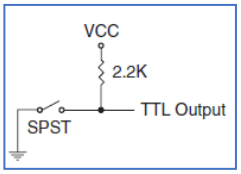
i. 如果输入设备是 TTL 或与 TTL 兼容的电路，可与微处理器及其接口部件相连。

ii. 不少输入设备是基于开关的设备，它们不是 TTL 电平。

iii. TTL 电平的逻辑 0 为 0.0~0.8V，逻辑 1 为 2.0~5.0V。

iv. 一个开关型设备要用作 TTL 兼容的输入设备，需做一些调整。

v. 例，将一个单刀单掷开关作为 TLL 设备连接，上拉电阻的标准范围通常在 1KΩ~10KΩ。



b) 输入设备：消除抖动

i. 机械开关闭合时，其触点会自然反跳（抖动）。为防止抖动，需要增加去抖动电路。

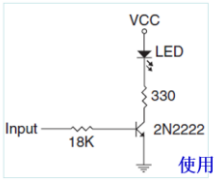
c) **输出设备：在连接任何输出设备之前，必须了解来自微处理器或 TTL 接口部件的电压和电流是多少。**

i. 来自微处理器或接口元件的电压是 TTL 兼容的。

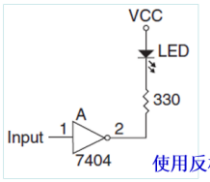
ii. 来自微处理器或许多接口部件的电流小于标准 TTL 部件的电流。

iii. **例，将一个简单的 LED 与微处理器外围引脚相连。**

iv.



使用晶体管



使用反相器

6. **I/O 端口地址译码**

a) I/O 端口地址译码与存储器地址译码非常相似。

i. 如果是存储器映像的 I/O 设备，则其地址译码方式与存储器的译码方式完全相同。

ii. 是否使用存储器映像 I/O，取决于存储系统的容量和系统中 I/O 设备的布局。

b) 独立编址 I/O 译码与存储器译码的主要区别：

i. 地址引脚数目不同。

ii. 使用 IORC#和 IOWC#激活 I/O 设备执行一次读写操作。早期 CPU 中，使用 IO/M#=1 与 RD#或 WR#，目前的 CPU 使用 M/IO=0 与 W/R#。

7. **Pentium~Core2 的 I/O 体**

a) 对于 Pentium~Core2，I/O 端口出现在不同的 I/O 体中，具体有 I/O 端口地址确定。
b) 例，8 位端口 0034H 出现在 Pentium 的 I/O 体 5 中。16 位端口 0034H~0035H 出现在 Pentium 的 I/O 体 5 和 I/O 体 6 中。最宽的 I/O 传送是 32 位，现在还没有 64 位 I/O 指令支持 64 位传送。
c) 当 I/O 端口扩展到 64 位时，应当尽量避免 I/O 端口的宽度跨越 64 位边界。
d) 例，16 位端口的地址 2007H 和 2008H，端口地址 2007H 在 I/O 体 7，端口地址 2008H 在 I/O 体 0，两者的被译码的地址是不同的。

8. **并行数据传输方式**
   a) 以计算机的字长（通常是 8 位、16 位、32 位或 64 位）为传输单位，一次传送一个字长的数据。
   b) 微机系统中最基本的信息交换方法。
      i. 例如，系统板上各部件之间，接口电路板上各部件之间。
   c) 适合于外部设备与微机之间进行近距离、大量和快速的信息交换。
      i. 例如，微机与并行接口打印机、磁盘驱动器。

9. 82C55
   a) 82C55 有 24 个引脚可用于 I/O，每组 12 个引脚可进行编程，以 3 种不同的操作方式工作。
   b) 82C55 可将任一 TTL 兼容设备与微处理器相连。
   c) 如果使用 82C55（CMOS 型）与高于 8MHz 时钟的微处理器一起工作，则需要插入等待状态。
   d) 82C55 可以为每个输出提供至少 2.5mA 的灌电流（logic 0），最大为 4mA。
   e) 82C55 在 Pentium 4 系统中仍有使用。尽管不是单独的 82C55 芯片，但编程是兼容的
   f) **8255A 由四个部分组成。**
      i. 数据总线缓冲器：双向、三态
      ii. 读/写控制逻辑
      iii. 三个八位数据端口：PA、PB、PC
      iv. A 组和 B 组的控制电路
         1. 根据 CPU 送来的编程命令控制 8255A 工作的电路
         2. 内部有控制寄存器，用来接收 CPU 送来的命令字
         3. A 组控制部件用来控制 PA 口和 PC 口的高 4 位
         4. B 组控制部件用来控制 PB 口和 PC 口的低 4 位
   g) **端口 A：PA0～PA7**
      i. 包含一个 8 位的数据输入锁存器，一个 8 位的数据输出锁存器/缓冲器。
      ii. 因此，A 端口作输入和输出时数据均能锁存。
      iii. A 组，支持工作方式 0、1、2。
      iv. 常作数据端口，功能最强大。
   h) **端口 B：PB0～PB7**
      i. 包含一个 8 位的数据输入缓冲器，一个 8 位的数据输入/输出锁存器/缓冲器。
      ii. B 组，支持工作方式 0、1。
      iii. 常作数据端口。
   i) **端口 C：PC0～PC7**
      i. 包含一个 8 位的数据输入缓冲器，一个 8 位的数据输出锁存器/缓冲器，无输入锁存功能。
      ii. 仅支持工作方式 0。

iii. 可作数据、状态和控制端口。
iv. 分为两个 4 位，每位可独立操作。
1. A 组控制高 4 位 PC4 ~ PC7。
2. B 组控制低 4 位 PC0 ~ PC3。

j) 读写控制逻辑

i. 用来管理数据信息、控制字和状态字的传送，它接收来自 CPU 地址总线的 A1、A0 和控制总线的有关信号，向 8255A 的 A、B 两组控制部件发送命令。
ii. RESET：复位信号
iii. CS#：片选
iv. RD#、WR#：读信号、写信号
v. A1、A0：端口选择信号。
vi. 8255A 内部有 3 个数据端口（PA、PB、PC）和一个控制字寄存器端口。

k) **方式 0 操作：基本输入输出方式**

i. 每一个端口都可以作为基本的输入/输出口
ii. A 口、B 口、C 口的高四位和低四位可以独立地设置为输入口或输出口。
iii. CPU 可以采用无条件读/写方式与 8255A 交换数据。
iv. 如果把 C 口的两个部分分别用作控制和查询口，与外设的控制和状态端相连，CPU 也可以通过对 C 口的读写实现对 A 口和 B 口的查询方式工作。
v. 规定：输出的数据被锁存，输入数据不锁存。

十、 算术协处理器、MMX 和 SIMD 技术

1. 概述

a) Intel 系列的算术协处理器包括 8087、80187、80287、80387SX、80387DX、80487SX。
b) 80486DX~Core2 微处理器均有内置的算术协处理器。
c) 但是，某些兼容的 80486 CPU（由 IBM 和 Cyrix 生产）内部并不包含算术协处理器。
d) 对于各种协处理器，指令系统和编程几乎完全相同，主要区别是每种协处理器被设计成与 Intel 不同型号的微处理器共同工作。
e) **80X87 协处理器**可以实现乘法、除法、加法、减法、求平方根、部分正切、部分反正切和对数运算。
f) 数据类型包括：
i. 16 位、32 位和 64 为带符号整数；
ii. 18 位 BCD 数据；
iii. 32 位、64 位和 80 位浮点数。
iv. 应用 80X87 执行的操作，通常比使用微处理器常用指令系统写出的最有效程序来执行同等的操作快许多倍。
v. **注意：针对协处理器进行汇编语言编程常常局限于修改诸如 C/C++高级语言生成的代码。**
g) 使用改进的 Pentium 协处理器，其运算速度比同等时钟频率下 80486 微处理器执行速度快 5 倍。
h) Pentium 微处理器常常可以同时执行一条协处理器指令和两条整数指令。
i) Pentium Pro ~ Pentium 4 协处理器与 Pentium 协处理器的操作类似，但增加了两条新的指令，即 FMOV 和 FCOMI。
j) **多媒体扩展（MMX）**与算术协处理器共享寄存器。
k) MMX 扩展是一种特殊的内部处理器，设计用于为外部多媒体设备高速执行指令。

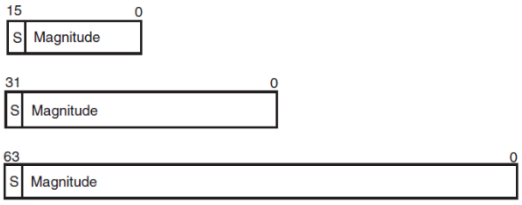
l) **SSE（Streaming SIMD Extensions）**：“与 MMX 指令类似，但作用于浮点数（而不是整数）；SSE 并不使用协处理器的寄存器空间”。

m) SSE 系列是 MMX 的超集，直到 SSE2 才跟 MMX 有本质的区别，添加了对 64 位双精度浮点数的支持，以及对整型数据的支持，也就是说这个指令集中所有的 MMX 指令都是多余的了，同时也避免了占用浮点数寄存器。

2. **算术协处理器的数据格式**

a) 算术协处理器支持的数据类型包括：

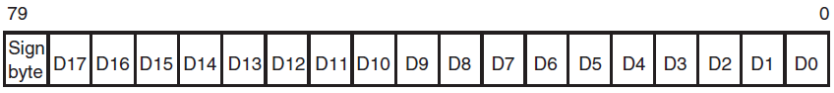
i. **带符号整数**



1. 以补码形式存储。
2. 字 (16 位)、双字 (32 位)、四字 (64 位)。
3. 数据定义：采用汇编伪指令 DD，DW 和 DQ。例：DATA1　DW　2
4.

ii. **BCD 数**

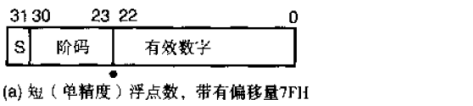
1. 以原码形成存储，而不是以 10 的补码形式存储的。
2. 一个 BCD 数需要 80 位的内存，占用 10 个字节。
3. 每个 BCD 数有 18 个数位，以压缩整数形式存储，每个字节有 2 个数位，共占用 9 个字节。
4. 第 10 个字节只包含带符号的 18 位 BCD 数的符号位
5.



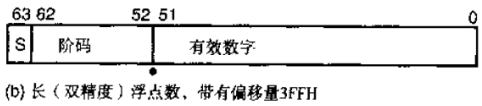
6. 数据定义：使用汇编伪指令 DT。
7. 例：
   a) 0000 00000000000000000200 DATA1 DT 200
   b) 000A 80000000000000000010 DATA2 DT -10
   c) 0014 00000000000000010020 DATA3 DT 10020
   d) 这种格式很少用，因为它唯一用于 Intel 协处理器。

iii. **浮点数**

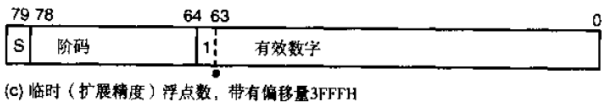
1. 浮点数通常称为实数，因为它们支持带符号整数、分数和混合数。
2. 一个浮点数包括 3 个部分：符号位、阶码、有效数字。
3. 浮点数通过科学二进制计数法来表示的。
4. Intel 系列算术协处理器支持三种类型的浮点数：
   a) 短浮点数（32 位），即单精度浮点数
   b) 长浮点数（64 位），即双精度浮点数
   c) 临时浮点数（80 位），即扩展精度浮点数



(a) 短（单精度）浮点数，带有偏移量7FH



(b) 长（双精度）浮点数，带有偏移量3FFH



(c) 临时（扩展精度）浮点数，带有偏移量3FFFH

5. 浮点数格式以及算术协处理器对它们的操作都遵循 IEEE-754 标准。
6. 有效数字是带有隐含位 1（整数部分）的数字。以扩展精度存储时，整数部分的 1 是可见的。
7. 表示规则：
   a) **零**：指数部分为 0，小数部分为 0。

b) **无穷大/无穷小**：指数部分为 2^e-1，小数部分为 0。

c) **NaN**：指数部分为 2^e-1，小数部分非零。

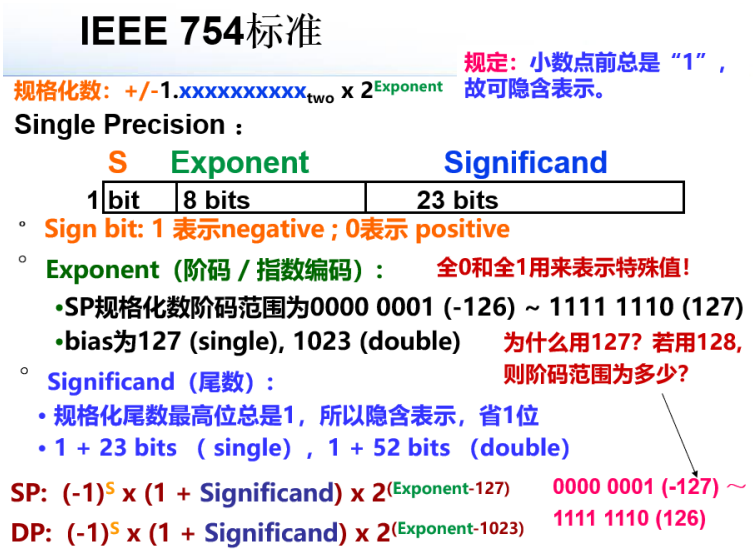
d) **规约形式**：指数部分 1~2^e-2，小数部分为任意值。

e) **非规约形式**：指数部分为 0，小数部分非 0。

f) **在非规约形式**下整数部份默认为 0，其他情况下一律默认为 1。

g) IEEE 754 标准规定：非规约形式的浮点数的指数偏移值比规约形式的浮点数的指数偏移值大 1。例如，最小的规约形式的单精度浮点数的指数部分编码值为 1，指数的实际值为-126；而非规约的单精度浮点数的指数域编码值为 0，对应的指数实际值也是-126 而不是-127。实际上非规约形式的浮点数仍然是有效可以使用的，只是它们的绝对值已经小于所有的规约浮点数的绝对值；即所有的非规约浮点数比规约浮点数更接近 0。规约浮点数的尾数大于等于 1 且小于 2，而非规约浮点数的尾数小于 1 且大于 0。NaN：例如，给负数-4 开平方, sqrt(-4)。

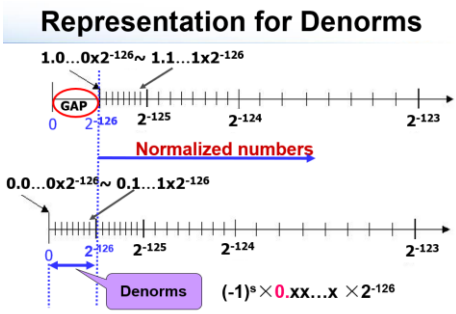
h)



i)

Exponent	Significand	Object
0	0	+/- 0
0	nonzero	Denorms
1-254	Anything (implicit leading 1)	Norms
255	0	+/- infinity
255	nonzero	NaN

Denorms 是不规范数



j) 不规范数是比 2 的-126 次方还要小的数

k) 在 Visual C++2008 或 Express 版中，采用了 float，double，Decimal 三种数据类型。

i. float：32 位；

ii. double：64 位；

iii. Decimal

iv. Visual Studio 的特殊类型。用于要求浮点数非常精确的领域，如银行。decimal 变量是 Visual Studio 2005 之后才有的。

1. Base 为 10，而不是 2。

2. 背景： double x= 0.1，但实际上，$x \neq 0.1$。

3. **转换步骤：**

a) 将十进制数转换为二进制数。

i. 规格化二进制数。

ii. 计算出阶码。

iii. 以浮点数格式存储该数。

1. 例，将十进制数 100.25 转换为单精度浮点数。

a) 步骤 1：100.25 => 1100100.01

b) 步骤 2：1100100.01 => 1.10010001×2^6

c) 步骤 3：000000110 + 01111111 => 10000101

d) 步骤 4：符号位= 0

e) 指数=10000101

f) 有效数字=10010001000000000000000

b) 将一个浮点数转换为十进制数形式的步骤：

i. 分离符号位、阶码和有效数字。

ii. 通过减去偏移量，将阶码转换为真正的指数。

iii. 将此数写为规格化的二进制数。

iv. 将规格化的二进制数转换为非规格化二进制数。

v. 将非规格化二进制数转换为十进制数。

1. 例，将浮点数转换为十进制数。

a) 步骤 1：Sign => 1

b) Exponent => 10000011

c) Significand => 10010010000000000000000

d) 步骤 2：100 = 10000011 - 01111111

e) 步骤 3：1.1001001×2^4

f) 步骤 4：11001.001

g) 步骤 5：-25.125

c) 使用汇编语言存储浮点数时：

i. 用 DD 伪指令存储单精度浮点数；

ii. 用 DQ 伪指令存储双精度浮点数；

iii. 用 DT 伪指令存储扩展精度浮点数。

iv. 例，浮点数的定义。

	内存中数据	变量名	伪指令	浮点数
2.	C377999A	DATA7	DD	-247.6
3.	40000000	DATA8	DD	2.0
4.	486F4200	DATA9	REAL 4	2,45E+5
5.	4059100000000000	DATAA	DQ	100.25
6.	3F543BF727136A40	DATAB	REAL 8	0.001235
7.	400487F34D6A161E4F76	DATAC	REAL10	33.9876

v.

4. 80X87 的结构

a) 80X87 与微处理器协同工作。

b) 80486DX~Core2 包含内置的、与 80387 完全兼容的协处理器。

c) 对于其它的 Intel 系列微处理器，协处理器是并联在微处理器上的外部集成电路。

d) 80X87 可以执行超过 68 条不同的指令。

e) 算术协处理器是一种特殊用途的微处理器，专门为有效地执行算术或超越函数的运算而设计的。

f) 微处理器执行所有的常规指令，80X87 只执行算术协处理器指令。

g) 微处理器截取和执行常规指令系统中的指令，而协处理器只截取和执行协处理器指令。

h) 协处理器指令实际上是换码（ESC）指令。微处理器使用这些指令为协处理器产生一个内存地址，使得协处理器可以执行协处理器指令。

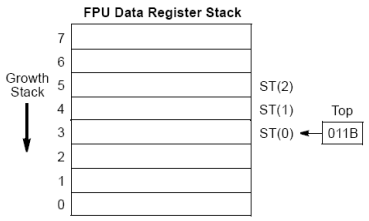
i) 微处理器和协处理器可以同时或并发地执行各自的指令。

5. **数据传送指令**

a) 浮点数传：

i. **FLD：将内存浮点数据装入由 ST 指向的内部栈顶。**

1. 该指令先将堆栈指针减 1，然后将数据存储在栈顶。
2. 装入栈顶的数据来自于任何存储单元，或来自于协处理器的另一个寄存器。
3. 传送数据的长度自动由汇编程序通过伪指令决定，如 DD 或 REAL4 表示单精度数据，DQ 或 REAL8 表示双精度数据，而 DT 或 REAL10 表示扩展精度数据。
4. **当协处理器复位或初始化时，栈顶为寄存器 0。**
5. 例
   a) FLD ST(2) ;将寄存器 ST(2)中的内容复制到栈顶。
   b) FLD DATA7 ;将 DATA7 的内容复制到栈顶。



ii. **FST：将栈顶的内容复制到存储单元或由操作数指示的协处理器寄存器中。**

1. 在存储过程中，内部的扩展精度浮点数舍入成为由控制器指定的浮点数长度。

iii. FSTP：将栈顶内容复制到内存或协处理器寄存器中，然后从栈顶弹出该数据。

1. FST 是复制指令，FSTP 是移动指令。

iv. FXCH：交换由操作数指定的寄存器或栈顶中的内容。

1. 例，FXCH ST(20)：交换栈顶数和 ST(2)中的数据。

v. **FILD：装入整数。**

vi. **FIST：存储整数。**

vii. **FISTP：存储并弹出整数**。

1. 这三条指令的功能与 FLD、FST 和 FSTP 一样，只不过传送的数据类型为整数，而不是浮点数。

viii. **协处理器自动将内部的扩展精度浮点数转换为整数。**

ix. **数据长度由汇编语言中用 DW、DD 和 DQ 定义标识的方法来决定。**

b) 带符号整数传：FILD, FIST, FISTP

c) BCD 数据传：FBLD, FBSTP

d) 数据只有在内存中才以带符号整数形式或 BCD 数形式出现。

e) 在协处理器内部，数据总是以 80 位扩展精度浮点数形式出现。