

第一章 微机概述

● 地址引脚数：为 16 条，可以寻址 $2^{16} = 64\text{KB}$ 的存储单元。 $2^{10} = 1\text{K}$, $2^{20} = 1\text{M}$, $2^{30} = 1\text{G}$; **b** 表示位，**B** 表示字节。

● 补码：补码为原码取反加 1，最高位 1 表示负数，0 表示正数。例：[-89] 补 = 10100111。十六进制数后面要加 H，否则默认十进制。

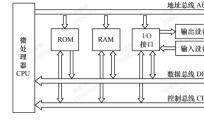
微处理器 微型计算机中用 CPU 表示，由一片或几片大规模集成电路组成，具有运算和控制器功能的中央处理器部件。

微型计算机 以微处理器为核心，配上存储器、输入输出接口电路及系统总线组成，又称主机。

微型计算机系统 以微型计算机为中心，配以外围设备、电源和辅助电路（硬件系统），以及指挥工作的软件系统。

软件系统 （操作系统、编译器、数据库等）和应用软件（Office、微信等）。

● 典型微机硬件系统结构：计算机的经典结构为冯·诺依曼结构（存储程序式计算机结构），特点包括：由运算器、控制器、存储器、输入设备和输出设备五个基本部分组成；程序和数据以二进制代码形式存放在存储器中，位置由地址指定，地址码也有二进制；控制器根据存储器中的指令序列（程序）工作，并由程序计数器（PC）控制指令执行，具有判断能力，可根据计算结果选择不同工作流程。微处理器是执行指令的核心部件，包含运算器和控制器。存储器用于存储当前正在使用的程序和数据。I/O 设备和接口实现微处理器外部设备的连接，如显示器接口、硬盘接口等。系统总线连接微处理器和其他部件，分为地址总线、数据总线和控制总线，分别传输地址、数据和控制信息。



数制与计算 • 有符号数的机器表示：真值指带正负号的实际数值（如 +5, -5），而机器数是计算机内部存储的编码形式。有符号数常用三种标准编码：

- 原码：[X]_原
- 反码：[X]_反
- 补码：[X]_补

● 正负数的编码规则：正数：三种表示法完全一致，即 [X]_原 = [X]_反 = [X]_补，符号位为 0，数值位直接表示绝对值。负数：三种表示法存在差异，所有区别仅体现在负数的编码规则上。

● 补码的主要用途：现代计算机均采用补码。补码便于硬件实现加法减法，成为实际工程标准。

● 补码加减法运算法则：核心思想：减法变加法，即 $X - Y$ 可转化为 $X + (-Y)$ ，简化硬件设计。

• 加法：[X + Y]_补 = [X]_补 + [Y]_补

• 减法：[X - Y]_补 = [X]_补 - [Y]_补 = [X]_补 + [-Y]_补

● 补码位数规定：补码加法时，符号位与数值位同等对待，全部参与二进制加法（逢二进一）。

● 补码求负（补规则）：[-X]_补 的求法：将 [X]_补 的所有位（包括符号位）按位取反，再加 1。即“全字取反加一”。注意：包括符号位，不区分符号和数值位。

第二章 微机结构



● 物理地址计算：PA = 段基址 $\times 10H$ + 偏移地址。段基址由段寄存器（16 位）左移 4 位提供，偏移地址由 IP 或 EU 计算。

指令队列 8086 为 6 字节，8088 为 4 字节。FIFO 原则。当队列空出 2 字节（8086）或 1 字节（8088）时，BIU 自动取指。执行跳转、调用/返回时请空。

通用寄存器 AX（累加器）、BX（基址）、CX（计数）、DX（数据），可拆分为 H/L 8 位使用。

指针与变址 SP（堆栈指针）、BP（基址指针）、SI（源变址）、DI（目的变址）。

第三章 80x86 系统指令

寻址方式 需要看清楚问的是源操作数还是目标操作数的寻址方式，如果是字操作数，要写两个单元的地址。

固定寻址 如 AAA
操作数直接包含在指令中。举例 MOV AL,15H || MOV AX,1234H
寄存器寻址 操作数在寄存器中，例：MOV AX,BX
存储器寻址 比较复杂，见下文详细说明。

● 存储器寻址：当执行单元 EU 需要读/写位于存储器的操作数时，应根据指令给出的寻址方式，由 EU 先计算出操作数的有效地址 EA（偏移地址），并将其送给 BIU；同时请求 BIU 执行一个总线周期，BIU 将某个段寄存器的内容左移 4 位，加上 EU 送来的有效地址 EA 形成 20 位的物理地址，然后执行总线周期，读/写指令所需的操作数。有效地址 EA（偏移地址）的值要根据指令所采用的寻址方式计算得出。计算 EA 的通式：

$$EA = \text{基址值} \left\{ \begin{array}{l} BX \\ BP \end{array} \right. + \text{变址值} \left\{ \begin{array}{l} SI \\ DI \end{array} \right. + \text{位移量} \left\{ \begin{array}{l} 0 \\ 8 \\ 16 \end{array} \right\}$$

直接寻址 操作数的 EA 由指令直接给出。段地址默认数据段 DS，其它数据段应在指令中用段前缀指出。这种寻址方式的指令执行速度较快，主要用于存取位于存储器中的简单变量。例：MOV AX,[1234H]

间接寻址 操作数在存储器中，存储单元的有效地址由寄存器指出。BX、SI、DI 一起认数据段 DS
BP—默认堆栈指针 SS。
注意：间接寻址的地址寄存器只能是 BX、BP、SI、DI，不能是其它寄存器。指令中地址寄存器要加方括号，如：[BX]。根据所采用的地址寄存器的不同，间接寻址方式又可分为以下 3 种：

基址寻址

操作数的有效地址由基址寄存器（BX 或 BP）的内容和指令中给出的地址位移量（0 位或 8 位或 16 位）之和来确定。EA = BX/BP + 0位/8位/16位移量。例：MOV AX,[BX]，假设 BX = 1122H, DS = 3000H，则 PA = 3000H + 1122H = 31122H, 30000H + 1123H = 31123H。假设 [31122H] = 34H, [31123H] = 56H，则指令执行后，AX = 5634H。例：假设 BETMA = 8, DS = 6000H, PA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘积的倍数 || MOV AL,[BX+SI+BETA] || MOV AL,[BX+BETA]。注：上面 4 条指令是等价的。EA = 5000H + 8 = 5008H, PA = DS × 16 + 5008H = 6000H + 5008H = 65008H。假设 [65008H] = 68H，执行后，AL = 68H。操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。EA（偏移地址）= SI/DI + 0位/8位/16位移量。例：MOV BETAD[DI], AX || MOV BX,[SI+BX]。[BX+SI] 是偏移地址位移量，不是乘

内存外存区别:

内存 存放当前运行的程序和数据。特点：**快、容量小、随机存取**，CPU 可直接通过系统总线访问。通常由半导体存储器 (RAM, ROM) 构成。
外存 存放非当前使用的程序和数据。特点：**慢、容量大、顺序/块存取**，CPU 不能直接访问，需通过I/O 接口电路 调入内存。如硬盘、U 盘、移动硬盘等。

RAM 类型特点:

SRAM 静态 RAM，利用**双稳态触发器**存储逻辑 0/1，**无需刷新**，只要不掉电信息不丢失。集成度低，外围控制电路简单，常用小容量存储 (如 Cache)。
DRAM 动态 RAM，利用**MOS 管栅极分布电容**存储电荷，**需定期刷新**。集成度高，外围控制电路复杂，常用大容量存储 (如 主存)。

存储容量 $N \times M$ (字数 \times 字长)
字数 N 单元总数，决定地址线数量 k ($2^k = N$)
字长 M 每单字位数，决定数据线位宽

常用存储芯片:

6264 (SRAM) $8K \times 8$ (8KB)，13 根地址线 ($2^{13} = 8K$)，8 根数据线
2114 (SRAM) $1K \times 4$ ，10 根地址线，4 根数据线 (常两片并联组成 8 位)
2764 (EPROM) $8K \times 8$ (8KB)，13 根地址线，8 根数据线，用于存储固件

● 地址译码：将 CPU 高位地址信号转换为芯片片选信号 CS#。常用 74LS138 (3-8 译码器) 实现。

Bit (位) 最小存储单位，对应硬件中的**双稳态触发器**状态。
Byte (字节) 基本处理单位，1 Byte = 8 bits。
Word (字) 基本处理单位，1 Word = 2 Bytes。
常用容量换算 $1KB = 2^{10} B$, $1MB = 2^{20} B$, $1GB = 2^{30} B$

基本性能指标 存取时间：从 CPU 发出地址信号到数据有效 (读) 或写入完毕 (写) 的时间。
存储周期：两次读或写所需的时间间隔。
存储周期 > 存取时间 (内部电路需要恢复时间)。
 $t_{cyc}(R)$: 连续两次读最小间隔； $t_{cyc}(W)$: 连续两次写最小间隔。

计算末尾地址的公式为：末尾地址 = 首地址 + 存储容量 - 1。注：减 1 是因为地址是从 0 开始计数的，且首地址本身占用了一个存储单元。

常见芯片

通用引脚特点 地址线 $A_0 \sim A_n$ 接地址总线 AB，输入信号，决定芯片容量 $N = 2^{n+1}$ 。
数据线 $D_0 \sim D_m$ 接数据总线 DB，双向 (RAM) 或输出 (ROM)，决定字长 M。
片选线 $CS/C/E$ 由**高位地址译码**产生，低电平有效，用于选中该芯片。
读写线 读允许 OE 接 CPU 的 RD；写允许 WE 接 CPU 的 WR。

6264 组织 地址线 $A_0 \sim A_{12}$ ($2^{13} = 8K$)；数据线 $D_0 \sim D_7$ 。
双片选机制 CS_1 (低有效) 和 CS_2 (高有效)。选中条件： $CS_1 = 0$ 且 $CS_2 = 1$ 。
读写控制 OE (输出允许，接系统 RD); WE (写允许，接系统 WR)。

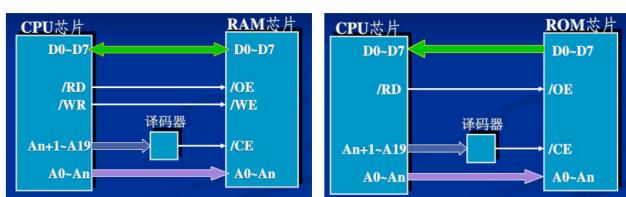
2114 容量与组织 $1K \times 4$ 位 (0.5 KB)；地址线 $A_0 \sim A_9$ (2^{10})，数据线 $D_1 \sim D_4$ 8 位系统需**两片并联**，分别负责高 4 位和低 4 位。

由于只有一个 WE 而没有 OE，因此 2114 的读写模式由 CS 和 WE 共同决定：

写模式 $CS = 0, WE = 0$ ，数据端口为输入 (DIN)
读模式 $CS = 0, WE = 1$ ，数据端口为输出 (DOUT)
待机模式 $CS = 1$ ，输出为高阻态 (Hi-Z)，实现总线隔离

2764 容量与组织 地址线 $A_0 \sim A_{12}$ ；数据线 $D_0 \sim D_7$ ，与 6264 引脚兼容。
CE# 片选使能，低电平有效。控制芯片激活及低功耗模式。
OE# 输出使能，低电平有效。读操作时打开输出缓冲器。
PGM# 编程脉冲，**低电平有效**。烧录时施加负脉冲，正常读取时置高。
Vpp 编程电压引脚，烧录时需施加高压 (12.5V 或 21V)。

在对芯片进行数据烧录 (编程) 时，需要在 PGM# 施加特定宽度 (如 50ms) 的负脉冲，配合 Vpp 引脚的高压 (通常 12.5V 或 21V)，将数据写入^{擦写晶体管 (EPROM)}。EPROM 没有 WE 引脚，因为它在正常工作时不可写。低位地址线用于片内寻址，高位地址线用于片选寻址：



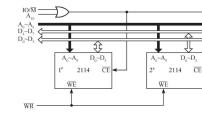
存储器扩展 • 存储器扩展：当单片芯片容量 (字数 N 或字长 M) 不足以满足系统需求时，需通过多片组合进行扩展。

位扩展 增加字长，字数不变。 $N \times M \rightarrow N \times (M \times k)$ 。各片地址线、读写线并联，数据线分别连接 CPU 数据总线的不同位。

字扩展 增加字数 (容量)，字长不变。 $N \times M \rightarrow (N \times k) \times M$ 。各片数据线、低位地址线并联，高位地址线经**译码器**产生片选信号 CS。

位和字节扩展 同时增加字长和字数。通常先通过位扩展组成满足字长要求的“存储组”，再通过字扩展 (译码片选) 连接各组。

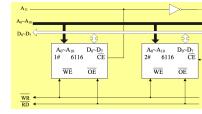
位扩展 当芯片字长小于 CPU 数据总线宽度时，通过多片并联增加**存储字长**。如两片 $1K \times 4$ 的 2114 并联可组成 $1K \times 8$ 存储器。



地址/片选/读写 全部并联，所有芯片接收相同地址，由同一片选信号同时选中，并行执行读/写操作。

数据线 (D_n) 分段连接，各片的数据端分别连接 CPU 数据总线的不同位段 (如一片接 $D_0 \sim D_3$ ，另一片接 $D_4 \sim D_7$)。

字节扩展 当芯片字长满足但容量不足时，增加**存储单元总数** (地址空间深度)。



地址线 (A_n) 低位地址线全部并联，用于片内寻址。
数据线 (D_n) 全部并联到系统数据总线 (与位扩展的区别)。

控制线 (CE) 读/写控制信号 (OE, WE) 全部并联。
片选线 (CS) 高位地址经译码器产生，各片独立连接，确保同一时刻仅一处有效。

● 核心逻辑：通过高位地址译码，将不同芯片映射到系统内存空间的不同地址段 (如 $0000H \sim 1FFFH$)。

位和字节扩展 当芯片字长和容量均不足时，需同时进行扩展。

● 芯片数计算：所需总芯片数 $Z = (M/L) \times (N/K)$ ，其中 M/L 为字扩展组数， N/K 为位扩展组数，L 为单片字长，K 为单片字数。

第一步：位扩展 将 N/K 片并联，地址/读写线并联，数据线分段连接，构成一个字长满足要求的存储组 (Bank)。

第二步：字扩展 将 M/L 个存储组级联，数据/低位地址线并联，高位地址经译码器产生各片的片选信号 CS。

地址译码电路设计 (片选) ● 片内寻址：根据芯片容量 N 确定最低位线数 k ($2^k = N$)，如 4KB 需 $A_{11} \sim A_0$ 。2. 片选逻辑：高位地址通过译码器 (如 74LS138) 或逻辑门产生 CS#。3. 范围固定：首地址为高位固定、低位全 0；末地址为高位固定、低位全 1。

全译码 CPU 全部地址线均被利用，低位用于片内寻址，剩余所有高位参与译码产生片选信号 CS#。特点：地址唯一性 (不重叠)，地址空间连续。
部分译码 仅利用部分高位地址参与译码。存在未使用的“悬空”地址线 (Don't Care)。特点：电路简单，但会导致**地址重叠** (镜像)，即一个物理单元对应多个逻辑地址。

● 地址重叠计算：若有 n 根高位地址线未参与译码，则一个物理单元对应 2^n 个重叠地址。

存储器与 CPU 连接

第七章 I/O 接口

概念 □ 端口：接口电路中用于缓存数据、状态、及控制信息的部件，分为：数据端口、状态端口、控制端口。

● 接口电路：计算机系统中包含多个不同功能的接口电路，每个接口电路又可能包含 1 个或多个端口。

● 寻址端口方法：先找到端口所在的接口电路芯片 (片选)，在该芯片上找到具体要访问的端口 (片内地址)。若接口中仅有 1 个端口，则找到芯片即找到端口；若接口中有多个端口，则找到芯片后还需找端口。每个端口号 = 片选地址 (高位地址) + 片内地址。

● 8086/8088 的 I/O 端口编址：采用 I/O 独立编址方式；I/O 操作只使用 20 根地址线中的 16 根 (A15~A0)；可寻址的 I/O 端口数为： $2^{16} = 64K$ (65536 个)；I/O 地址范围为：0~0FFFFH。

8259A • 8259A 特性 功能：8259A 是一个功能很强的中断扩充和多中断管理芯片，具有**中断扩展、自动提供中断类型码、中断优先级裁决**等中断管理功能。可编程：内部有多个寄存器以及功能部件都是可编程的，使用方便。级联扩展：单片可连接 8 个中断请求源，多片级联可扩展到 64 级中断。通过编程可设置**中断触发方式、中断类型码、中断屏蔽方式、中断优先级方式、中断结束方式**等。

● 8259A 内部结构：数据总线缓冲器：三态、双向、8 位寄存器。读写控制逻辑：接收 CPU 的读写控制信号。级联缓冲/比较器：支持单片或多片级联，主片/从片管理，最多可扩展到 64 级中断。控制逻辑：向片内各部件发送控制信号，向 CPU 发送中断请求信号 INT，接收 CPU 回送的 INTA* 信号，控制 8259A 进入中断管理状态。中断请求寄存器 (IRR)：8 位寄存器，记录外部中断请求，IRR 有请求时 IR_i 的相应位 DI 置 1，中断响应后清除。中断屏蔽寄存器 (IMR)：IMR 中 DI 位为 1 时禁止对应 IR_i 请求，为 0 时允许。优先权判决器：对 IR_i 中未屏蔽的中断进行优先级比较，选出当前优先级最高的中断申请。中断服务寄存器 (ISR)：记录 CPU 当前正在服务的中断标志，IRR 请求响应时 ISR 相应位置 1，复位由中断结束方式决定。

第八章 常用接口技术

8255A • 8255 功能介绍 8255 是一种可编程的并行通信接口芯片，可用于 CPU 和外设之间进行并行数据传输。内部有 8 个 8 位的数据端口，有三种工作方式。端口号的 A0A1 为 00、01、10 分别表示读写 A、B、C 口；11 表示只写控制寄存器。

● 并行通信：指多位数据同时进行传送的方式，其特点是**传输速度快**。

方式 0 基本的输入/输出方式，A/B/C 口均可用，C 口仅需设置方向。

方式 1 选通的输入/输出方式，A/B 口可用，支持中断方式传输，C 口部分引脚用于控制和中断信号。

方式 2 选通的双向传输方式，仅 A 口可用，支持双向传输和中断，C 口部分引脚用于控制和中断信号。

A 口 可工作在方式 0、1、2，方式 2 时支持双向传输。
B 口 可工作在方式 0、1，仅支持单向传输。
C 口 只能工作在方式 0，高四位和低四位可分开使用，方式 1/2 时部分引脚用于控制和中断。

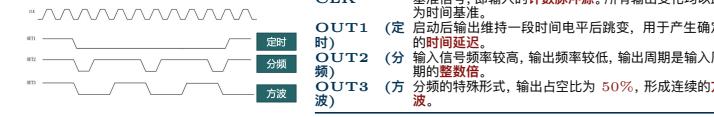
方式 1、2：选通输入输出方式，可以**中断方式传输**，且 C 口会固定的引脚用作**控制联络信号**和**中断请求信号**。

仅 A 口工作在方式 2 时，可以双向传输，A 口工作在方式 0、1 及 B、C 口只能单向传输。

● 控制字：控制字分为端口的方式选择控制字 (可使 8255 的 3 个数据端口工作在不同的方式) 和 C 口的按位置位和复位控制字 (可使 C 口的任意一位置位和复位)。控制字送入的端口为最后一个端口。



8253 • 概述 8253 是一种可编程的计数器/定时器接口芯片，最高计数频率为 2MHz，可用于产生各种定时波形，也可用于对外部事件计数。内部有三个独立的 16 位减一计数器 (互不干扰，支持二进制 (Binary) 或二-十进制 (BCD 码) 计数)，通过设置控制字，各计数器可以工作于 6 种工作方式。功能简述：



● 系统总线接口 (连接 CPU)：D₇ ~ D₀ 双向数据总线，用于 CPU 读写控制字或计数值。片选信号，低电平有效，由地址译码电路产生。读/写信号，低电平有效，控制数据传输方向。端口选择：00, 01, 10 分别对应计数器 0, 1, 2; 11 为控制寄存器。
● 计数通道信号 (连接外设)：CLK_{0~2} 时钟输入，计数脉冲源，下降沿触发减 1 操作。GATE_{0~2} 门控输入，用于启动、停止或暂停计数过程。OUT_{0~2} 输出信号，计数完成或到达条件时输出特定波形。

● 控制寄存器：只能进行**写操作**，不能读。CPU 通过向此寄存器写入“控制字”来设定各通道的工作方式。

● 计数通道结构：包含三个独立的计数通道 (0, 1, 2)，内部由以下三部分配合实现：预置 → 减计数 → 读出逻辑。

初值寄存器 (CR) 16 位。用于预置计数的起始值。
执行部件 (CE) 16 位溢出计数器。在 CLK 信号驱动下进行减 1 计数。
输出锁存器 (OL) 16 位。用于锁存当前计数值供 CPU 读取，不影响计数进行。

8253 没有状态寄存器，CPU 无法直接读取状态寄存器来获知当前工作状态或回读控制字。这与具有状态回读功能的 8254 不同。

● 端口地址分配：8253 占用 4 个 I/O 端口。CPU 通过 A1, A0 选择：

写 (OUT) 送入 初值寄存器 (CR)。
读 (IN) 来自 输出锁存器 (OL)。
这种“双缓冲”设计保证计数时读数稳定。

● 实例 1：连续地址 (8088)：

设定 地址范围：0380H ~ 0383H。地址连续 (步长为 1)。
分析 0380H(...000) → A₁A₀ = 00
0381H(...010) → A₁A₀ = 01
0382H(...100) → A₁A₀ = 10
0383H(...110) → A₁A₀ = 11。

连接 CPU 的 A₁, A₀ 直接连接到 8253 的 A₁, A₀。每一个逻辑地址都对应一个物理端口，无地址间隙。

● 读/写逻辑真值表：

片选前提 所有操作必须在 CS = 0 时有效。

写 (WR = 0) 根据 A₁, A₀ 写入通道 0/1/2 的计数初值寄存器，或写入控制字寄存器。

读 (RD = 0) 根据 A₁, A₀ 读取通道 0/1/2 的当前计数值。注意：控制字寄存器只写不读。

● 8 位总线与 16 位计数器接口：8253 内部的计数初值寄存器 (CR) 和输出锁存器 (OL) 都是 16 位的，但外部数据总线 (D₇ ~ D₀) 只有 8 位。必须通过两次 I/O 操作来完成一个 16 位数据的传输，由控制寄存器中的控制字来决定读写顺序 (如：先读/写低 8 位，再读/写高 8 位)。

控制字



SC₁, SC₀ 计数器选择，因共用端口 (3 个计数器共用一个控制端口)，需指定目标通道。

RL₁, RL₀ 读写格式。定义 CR/OL 的读写位宽及顺序。

OO: 锁存命令 (不停止计数读取)；01: 仅低 8 位；10: 仅高 8 位；11: 先高后低 (16 位常用)。

工作方式：设定方式 0 ~ 5。
计数制：0: 二进制 (FFFFFH); 1: BCD 码 (9999)。

- 实例 1：8 位读写模式：需求：计数器 0，方式 2，仅使用低 8 位，初值 100，二进制计数。地址：70H ~ 73H。

SC 00 (选择计数器 0)
RL 01 (只读/写低 8 位)
M 10 (方式 2, 分频器)
BCD 0 (二进制计数)
控制字 000010100B = 14H

汇编实现：
 • MOV AL, 14H
 • OUT 73H, AL ; 写控制字
 • MOV AX, 100 ; 初值 100
 • OUT 70H, AL ; 写低 8 位

● 锁存命令：解决在计数器运行过程中读取数值不稳定的问题。通过向控制寄存器写入 $RL_1 RL_0 = 00$ 的控制字，将当前数值复制到 **输出锁存器 (OL)** 中保持不变，而 **执行部件 (CE)** 继续计数。

锁存读出示例：

向控制口发送控制字，其中 $RL_1 RL_0 = 00$ 。8253 接收后锁存当前值，不影响内部计数。

实例分析 代码流程

注意 读出操作必须符合初始化时设定的 **RL** 格式（如 16 位模式需连读两次）。

● 相关名词解释：

CLK 脉冲 指 **CLK** 引脚上的信号单元。在计数过程中，每一个 **CLK** 脉冲的下降沿到来时，计数器减 1。
计数器时常数 与“计数通道”同义。
 指通过指令写入计数器的值，等同于 **初值**。输出波形的周期或延时由该值决定：时间 = 初值 $\times T_{CLK}$ 。

方式 0 ● 功能定义：

主要用于**定时中断**。给定时间 t_0 ，到达后输出信号通知 CPU。
GATE 是硬件门控信号，在方式 0 中充当“计数使能开关”。

初始化 写入方式控制字后，输出引脚 **OUT** 变为低电平。
 计数过程 写入初值后开始减 1 计数，期间 **OUT** 保持低 0 暂停计数，计数器保持当前值不变，忽略电平。
 计数结束 当计数值减到 0 时，**OUT** 立即跳变为高电平。1 产生上升沿信号通常连接 CPU 中断请求引脚。

● 时序分析 (基本过程)：重点在于 $N + 1$ 个时钟周期的由来。

写入控制字 WR 有效 \rightarrow OUT 变低 (起始状态)。
 写入初值 第二个 WR 将初值 N 写入 CR。
 截入时刻 写入后第 1 个 CLK 下降沿，数据 CR \rightarrow CE。
 计数过程 载入后开始减 1 ($N \rightarrow 0$)。总时长 $N + 1$ 个 CLK。
 终值输出 CE 减为 0 时，OUT 变高。

● GATE 控制 (暂停)：

GATE=0 暂停计数 (CE 保持)，OUT 保持低。
GATE=1 恢复计数 (从暂停值继续)。
 ● 结论：利用 GATE 可加长 OUT 低电平宽度。实际时间 = 预置 + 暂停。

● 重写初值 (重启)：

若在计数结束时写入新初值，8253 会放弃当前进程，在下一个 CLK 重新装入新初值并重新开始。同样实现了加长 OUT 低电平宽度的效果。
 ● 周期数：写入时常数为 N 时，OUT 低电平宽度为 $N + 1$ 个 CLK 周期。
 ● 计数完成后：CE 寄存器会变成 FFFFH，OUT (OL 锁存器) 一直输出高电平，但是 CE 持续在-1 计数。

● 方式 0 编程示例：

已知端口 40H ~ 43H，计数器 0，方式 0，初值 1500，二进制计数。

控制字 SC=00, RL=11, M=000, BCD=0 \rightarrow 30H
 时常数 1500 = 05DCH (低位 DCH, 高位 05H)

汇编实现：

```

MOV DX, 43H ; 指向控制口
MOV AL, 30H ; 写控制字
OUT DX, AL
MOV DX, 40H ; 指向计数器 0 数据口
MOV AX, 1500 ; AX = 05DCH
OUT DX, AL ; 写低 8 位 (DCH)
MOV AL, AH ; 取高 8 位 (05H)
OUT DX, AL ; 写高 8 位

```

● 核心逻辑：写初值 \rightarrow OUT 低 \rightarrow 计数 $\rightarrow 0 \rightarrow$ OUT 高。定时时长 = $(N + 1) \times T_{CLK}$ 。可通过 GATE 硬件信号暂停，或重写初值延长定时。

方式 1 ● 功能定义：

单脉冲形成。即产生一个宽度可控的负脉冲。

触发源 硬件触发。区别于方式 0 的软件写入触发，方式 1 必须由 **GATE** 引脚的上升沿 (由低电平变高电平) 来触发。
 脉冲宽度 输出负脉冲的宽度为 $N \times T_{CLK}$ ，其中 N 为预置的初值。

● 实例 2：16 位读写模式：需求：计数器 1，方式 1,16 位 (先低后高)，初值 1234，BCD 码，地址：70H ~ 73H。

SC 01 (选择计数器 1)
 RL 11 (先低 8 位后高 8 位)
 M 001 (方式 1, 单稳态)
 BCD 1 (BCD 码计数)
 控制字 01100011B = 73H

汇编实现：
 • MOV AL, 73H
 • OUT 73H, AL ; 写控制字
 • MOV AX, 1234H ; BCD 初值
 • OUT 71H, AL ; 写低 8 位
 • MOV AL, AH ; 取高 8 位
 • OUT 71H, AL ; 写高 8 位

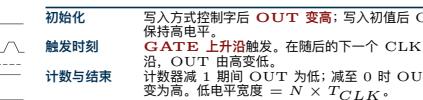
● 锁存命令：解决在计数器运行过程中读取数值不稳定的问题。通过向控制寄存器写入 $RL_1 RL_0 = 00$ 的控制字，将当前数值复制到 **输出锁存器 (OL)** 中保持不变，而 **执行部件 (CE)** 继续计数。

锁存读出示例：

向控制口发送控制字，其中 $RL_1 RL_0 = 00$ 。8253 接收后锁存当前值，不影响内部计数。

实例分析 代码流程

注意 读出操作必须符合初始化时设定的 **RL** 格式（如 16 位模式需连读两次）。



● 可重触发性：在脉冲未结束时，若 **GATE** 再次出现上升沿，8253 会在下一个 CLK 将初值寄存器 (CR) 的值重新装入执行单元 (CE)，使计数器重新开始，从而延长 OUT 低电平宽度。

● 修改初值 (方式 1)：在脉冲输出过程中，若 CPU 修改计数初值：

系统响应 新初值 N_{new} 仅存入 **初值寄存器 (CR)**。当前计数器 (CE) 不受影响，继续按旧值 N_{old} 减至 0，当前脉冲宽度不变。

生效时刻 仅在 **下一次 GATE 上升沿** 到来时，新值 N_{new} 才从 CR 装入 CE。

结论 写入新初值不会立即重启计数，而是预置给下一次触发使用。

| 方式 0 (软件触发) | 方式 1 (硬件触发) |
|----------------------|------------------|
| 写初值操作直接启动计数 | GATE 上升沿触发启动 |
| GATE 电平敏感：1 计数，0 停止 | GATE 边沿敏感：上升沿重触发 |
| 低电平宽度为 $N + 1$ 个 CLK | 低电平宽度为 N 个 CLK |

相同点

均为减 1 计数；工作时输出低电平；均具备定时功能。

● 实例 1-单一通道编程：
 需求：计数器 2，方式 1，初值 15，仅低 8 位。地址：控制口 COUNTD，计数器 2 COUNTC。

控制字 SC = 10, RL = 01, M = 001, BCD = 0 \rightarrow 92H
 汇编实现 MOV AL, 92H
 OUT COUNTD, AL ; 写控制字
 MOV AL, 15
 OUT COUNTC, AL ; 写初值

CNT0 控 SC = 00, RL = 01, M = 000 \rightarrow 10H
 CNT1 控 SC = 01, RL = 01, M = 001 \rightarrow 52H
 汇编实现 MOV AL, 10H ; OUT COUNTD, AL
 MOV AL, 52H ; OUT COUNTD, AL
 MOV AL, 7
 OUT COUNTA, AL ; CNT0 初值
 OUT COUNTB, AL ; CNT1 初值

● 方式 0 功能实现：

已知端口 40H ~ 43H，计数器 0，方式 0，初值 1500，二进制计数。

控制字 SC=00, RL=11, M=000, BCD=0 \rightarrow 30H
 时常数 1500 = 05DCH (低位 DCH, 高位 05H)

汇编实现：

MOV DX, 43H ; 指向控制口
 MOV AL, 30H ; 写控制字
 OUT DX, AL
 MOV DX, 40H ; 指向计数器 0 数据口
 MOV AX, 1500 ; AX = 05DCH
 OUT DX, AL ; 写低 8 位 (DCH)
 MOV AL, AH ; 取高 8 位 (05H)
 OUT DX, AL ; 写高 8 位

● 时序分析 (基本过程)：重点在于 $N + 1$ 个时钟周期的由来。

写入控制字 WR 有效 \rightarrow OUT 变低 (起始状态)。
 写入初值 第二个 WR 将初值 N 写入 CR。
 截入时刻 写入后第 1 个 CLK 下降沿，数据 CR \rightarrow CE。
 计数过程 载入后开始减 1 ($N \rightarrow 0$)。总时长 $N + 1$ 个 CLK。
 终值输出 CE 减为 0 时，OUT 变高。

● GATE 控制 (暂停)：

GATE=0 暂停计数 (CE 保持)，OUT 保持低。
GATE=1 恢复计数 (从暂停值继续)。

● 结论：利用 GATE 可加长 OUT 低电平宽度。实际时间 = 预置 + 暂停。

● 重写初值 (重启)：

若在计数结束时写入新初值，8253 会放弃当前进程，在下一个 CLK 重新装入新初值并重新开始。同样实现了加长 OUT 低电平宽度的效果。

● 周期数：写入时常数为 N 时，OUT 低电平宽度为 $N + 1$ 个 CLK 周期。

● 计数完成后：CE 寄存器会变成 FFFFH，OUT (OL 锁存器) 一直输出高电平，但是 CE 持续在-1 计数。

● 方式 0 编程示例：

已知端口 40H ~ 43H，计数器 0，方式 0，初值 1500，二进制计数。

控制字 SC=00, RL=11, M=000, BCD=0 \rightarrow 30H
 时常数 1500 = 05DCH (低位 DCH, 高位 05H)

汇编实现：

```

MOV DX, 43H ; 指向控制口
MOV AL, 30H ; 写控制字
OUT DX, AL
MOV DX, 40H ; 指向计数器 0 数据口
MOV AX, 1500 ; AX = 05DCH
OUT DX, AL ; 写低 8 位 (DCH)
MOV AL, AH ; 取高 8 位 (05H)
OUT DX, AL ; 写高 8 位

```

● 核心逻辑：写初值 \rightarrow OUT 低 \rightarrow 计数 $\rightarrow 0 \rightarrow$ OUT 高。定时时长 = $(N + 1) \times T_{CLK}$ 。可通过 GATE 硬件信号暂停，或重写初值延长定时。

● 方式 1 ● 功能定义：

单脉冲形成。即产生一个宽度可控的负脉冲。

触发源 硬件触发。区别于方式 0 的软件写入触发，方式 1 必须由 **GATE** 引脚的上升沿 (由低电平变高电平) 来触发。

脉冲宽度 输出负脉冲的宽度为 $N \times T_{CLK}$ ，其中 N 为预置的初值。