微机原理与接口技术

微型计算机基础

微处理器（CPU）：由运算器、控制器、寄存器等部件构成的大规模集成电路芯片；
位：二进制位 0，1，计算机中存储信息最小单位；
字节：8 个二进制位为 1 字节，计算机中存储单元常按字节设置；
字：16 个二进制位；
主频：CPU 的时钟频率；
外频：外部总线频率，系统总线传输数据的频率；
缓存：位于 CPU 与内存间的临时存储器；

权值：某一数制表示的数中每一位所处的级别；

$$x=\sum_{i=-m}^{n} k_i * b^i$$

逻辑运算：

AND：对指定位置 0；
OR：对指定位置 1；
XOR：对指定位取反；

补码：

$$[x]_C=\begin{cases} x, 0\le x<2^{n-1} \\ x+2^n, -2^{n-1}\le x<0 \end{cases}$$

原码求补码

$$\left[[x]_C\right]_C=x$$

相反数求补码

$$[x]_C=1+\overline{[-x]_C}$$

有符号数溢出：

加法：数值最高位进位，符号位不进位/数值最高位无进位，符号位进位，结果溢出；
减法：数值最高位不需借位，符号位需借位/数值最高位需借位，符号位不需借位，结果溢出；

BCD 编码：用 4 位二进制数表示一位 10 进制数的编码方式；
组合 BCD 数：用一字节的高低位分别表示一个 BCD 数；
分离 BCD 数：用一字节的低位表示一个 BCD 数；
加法修正：
两 BCD 码位相加有进位/结果>9：该位+6 修正；
两 BCD 码位相加无进位，且结果≤9：该位不需修正；
低 BCD 码位修正结果使高 BCD 码位>9：高位+6 修正；

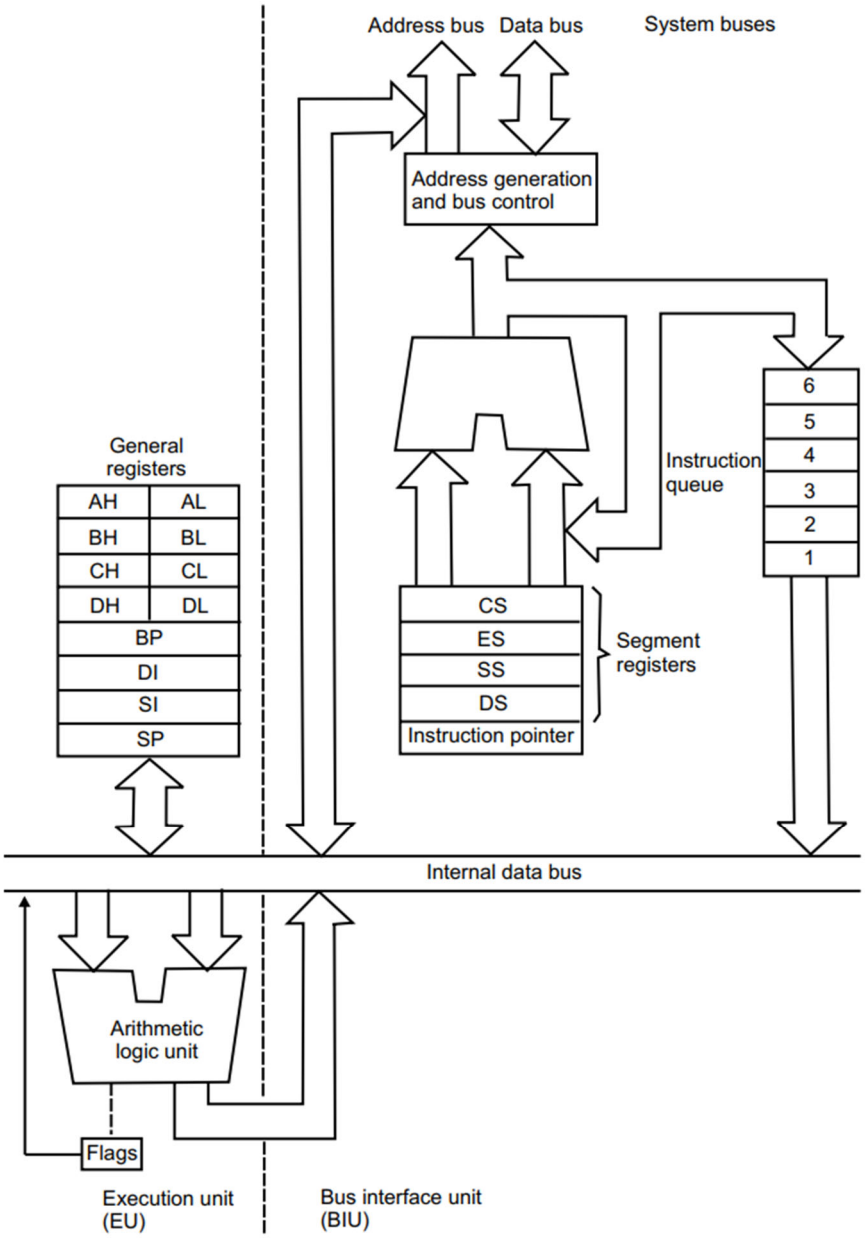
ASCII 码：
’0’：48；’A’：65；’a’：97；

8086CPU 结构与功能

微处理器级总线：CPU 引脚信号；

总线：用于连接 CPU 与其他部件的一组连线；

I/O 接口：连接 CPU 与 I/O 设备的控制电路；



CPU 内部结构：

1.算术逻辑单元：完成所有运算操作；

2.工作寄存器：残存寻址信息与计算中间结果；

3.控制器：完成指令的读入、寄存、译码，产生控制信号序列；

4.I/O 控制逻辑：处理 I/O 操作；

CPU 功能结构：

1.执行单元（Execution Unit）：执行指令规定操作；

2.总线接口单元（Bus Interface Unit）：完成取指令、存取数据的操作；

指令队列中无指令时，EU 处于等待状态；EU 执行指令需访问存储器时，BIU 尽快完成存取数据操作；

8086CPU 的寄存器组织：

1.通用寄存器

数据寄存器：

AX（Accumulator）：累加器；

BX（Base Register）：基址寄存器；

CX（Count Register）：计数寄存器；

DX（Data Register）：数据寄存器；

地址寄存器：

SI（Source Index）：变址寄存器，源操作数；

DI（Destination Index）：变址寄存器，目的操作数；

SP（Stack Pointer）：堆栈指针；

BP（Base Pointer）：基址指针；

2.段寄存器

CS（Code Segment）：代码段寄存器，存放当前执行程序段地址；

DS（Data Segment）：数据段寄存器，存放当前数据段地址；

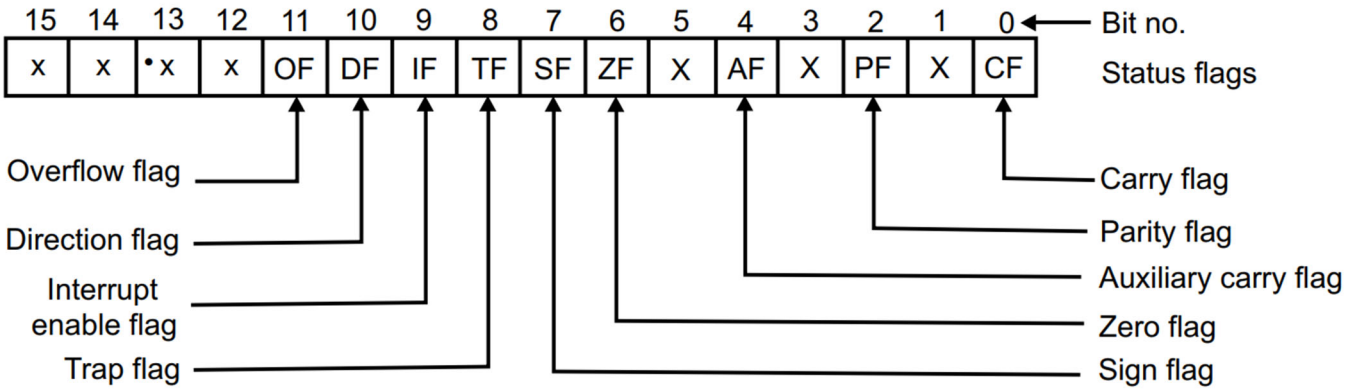
ES（Extra Segment）：附加段寄存器，存放附加数据段段地址；

SS（Stack Segment）：堆栈段寄存器，存放堆栈段段地址；

3.控制寄存器

IP（Instruction Pointer）：指令指针，保存下一条即将要执行指令的段内偏移地址；

PSW（Processor State Word）：微处理器状态字，共设定 9 个标志位；



反映 ALU 前一次操作结果状态：

CF（Carry Flag）：进位标志，加减运算时，有进/借位，**CF=1**；

PF（Parity Flag）：奇偶标志，操作结果低 8 位中，有偶数个“1“时，**PF=1**；

AF（Auxiliary Carry Flag）：辅助进位标志，加减运算时，D3 位有进/借位时为 1；

ZF（Zero Flag）：零标志，运算结果为 0 时，**ZF=1**；

SF（Sign Flag）：符号标志，运算结果为负时，**SF=1**；

OF（Overflow Flag）：溢出标志，有符号数运算有溢出时，**OF=1**；

控制 CPU 的标志位：

DF（Direction Flag）：方向标志，**DF=0** 时，变址寄存器内容自动递增；

IF（Interrupt enable Flag）：中断允许标志，**IF=0** 时，CPU 不能响应中断；

TF（Trap Flag）：陷阱标志，**TF=1** 时，CPU 处于单步执行方式；

8086CPU 的存储器与 I/O 组织：

对准：一个字从偶地址开始存储；

物理地址=段地址×10H+偏移地址；

8086CPU 采用地址总线低 16 位对 I/O 端口进行编址；

I/O 端口与存储器地址空间不同，需采用不同指令访问；
I/O 端口不需段地址；

汇编语言基础

语句类型：

1.指令：汇编后形成对应的机器语言指令；

2.伪指令：告诉汇编程序如何进行汇编，程序汇编时执行；

3.宏指令：用户自己定义的指令，程序汇编时，宏展开为指令与伪指令；

汇编语言语句：

名称 | 空格/:| 操作助记符 | 空格 | 操作数 |;注释

名称：字母开头，字母、数字、特殊符组成；

表达式：由操作数与操作符组成；

标号：指令语句定义的标识符；

属性：段地址、偏移地址、类型；

变量：

变量名 D_ 表达式

D_：W 字变量，D 双字变量，Q 长字变量，T10 字节变量；

？：表示只分配存储空间；

$：表示当前汇编语句的偏移地址；

DUP：重复表达式多次；

num DUP (expr)

属性：段地址、偏移地址、类型（B，W，DW）、长度（**num**）、大小（变量占用总字节数）；

属性操作符：获取标号/变量属性的操作符；

SEG：取出段地址；

OFFSET：取出偏移地址；

TYPE：取出类型；

LENGTH：取出变量重复次数；

SIZE：取出变量大小；

PTR：改变变量或标号的类型；

type PTR expr

立即数：可直接在指令中给出的 CPU 指令所需要的数据；

寻址方式：

数据寻址方式：

1.立即寻址：将数据直接放在指令后；

2.寄存器寻址：指令操作数存放在寄存器中；

3.直接寻址：操作数保存在存储单元，偏移地址直接在指令中给出（变量名、变量名与位移、地址）；

4.寄存器间接寻址：操作数保存在存储单元，有效地址存放于寄存器（**BX，SI，DI**）

5.寄存器相对寻址：操作数保存在存储单元，有效地址

$$EA=(BX/BP/SI/DI)+disp8/16$$

6.基址变址寻址：操作数保存在存储单元，有效地址

$$EA=(BX/BP)+(SI/DI)$$

7.基址变址且相对寻址：操作数保存在存储单元，有效地址

$$EA=(BX/BP)+(SI/DI)+disp8/16$$

8.隐含寻址：操作码本身隐含指明操作数地址；

转移地址寻址方式：

1.段内直接寻址：直接在指令中给出转移目的地址，转移在同一段内完成；

2.段内间接寻址：转移目的地址保存在寄存器/存储单元，转移在同一段内完成；

3.段间直接寻址：直接在指令中给出转移目的地址，转移在不同段间完成；

4.段间间接寻址：转移目的地址保存在寄存器/存储单元，转移在不同段间完成；

汇编语言程序结构：

段定义伪指令：

seg_name SEGMENT (seg_addr) (combination_type) (type)

…

seg_name ENDS

ASSUME 伪指令：

ASSUME seg_reg: name, …

ASSUME CS:CODE, DS:DATA, ES:DATA

告诉 MASM 各个段寄存器存放哪个段的段地址；

END 伪指令：

END expr

表示源程序的结束

EQU 伪指令：

name EQU expr

为表达式赋一个名称，仅可定义一次；

ORG 伪指令：

ORG expr

为后续指令指定段内偏移地址

源程序的汇编、链接、调试：

$$.asm \xrightarrow{MASM} .obj, .lst \xrightarrow{LINK} .exe$$

汇编语言指令与程序设计

NOP：空操作，占用 3 个时钟周期；

HLT：暂停，退出暂停状态条件：

1.**RESET** 信号有效；

2.**NMI** 信号有效；

3.**INTR** 信号有效，且 **IF=1**；

WAIT：等待，对 $\overline{TEST}$ 每隔 5 个时钟周期进行测试，低电平时退出等待；

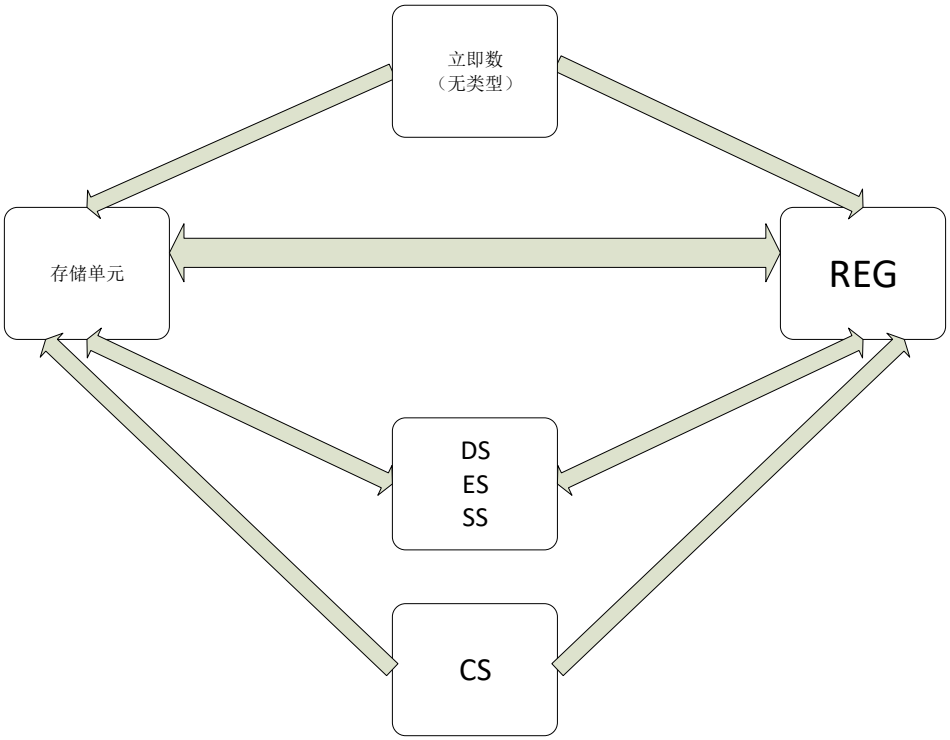
LOCK：总线锁定，保持总线使用权；

ESC：换码指令，完成多处理器间指令、数据的交换；

数据传送指令：

通用传送指令：

MOV　DST，SRC；(DST) ← (SRC)



两存储器单元间不能直接操作；

由立即数至存储单元，类型需指明；

取有效地址：

LEA　REG16，　MEM；(REG16) ← (MEM 偏移地址)

取地址指针：

LDS　REG16，　MEM；(DS) ← (MEM+2)，　(REG16) ← (MEM)

LES　REG16，　MEM；(ES) ← (MEM+2)，　(REG16) ← (MEM)

将双字变量 **MEM** 中高 16 位送到 **DS/ES**，低 16 位送到 **REG16**（不允许为段寄存器）；

标志传送：

LAHF；(AH) ← PSW 低 8 位

SAHF；PSW 低 8 位 ← (AH)

数据交换：

XCHG　DST，SRC；DST←→SRC

字节转换：

XLAT；(AL) ← ((BX)+(AL))

将有效地址为**(BX)+(AL)**的存储单元的内容送入 **AL**；

堆栈：

PUSH　SRC；将 SRC 压入堆栈，（SP）←（SP）- 2，（SP）←（SRC）

SRC 需为字型，不能为立即数；

PUSHF；将 PSW 压入堆栈，（SP）←（SP）- 2，（SP）←（PSW）

POP　DST；从堆栈中弹出 DST，（DST）←（SP），（SP）←（SP）+ 2

POPF；从堆栈中弹出 PSW，（PSW）←（SP），（SP）←（SP）+ 2

数据运算指令：

加法指令：

ADD　DST，SRC；(DST) ← (SRC) + (DST)

ADC　DST，SRC；(DST) ← (SRC) + (DST) + (CF)

减法指令：

SUB　DST，SRC；(DST) ← (DST) - (SRC)

SBB　DST，SRC；(DST) ← (DST) - (SRC) - (CF)

取负指令：

NEG　DST；(DST) ← 0 – (DST)

比较指令：

CMP　DST，SRC；(DST) - (SRC), 置标志位；

增量减量指令；

INC　DST；(DST) ← (DST) + 1

DEC　DST；(DST) ← (DST) - 1

乘法指令：

MUL　SRC；无符号数相乘，源操作数隐含为 **AL/AX**，目的操作数隐含于 **AX/DX**；

IMUL　SRC；有符号数相乘，源操作数隐含为 **AL/AX**，目的操作数隐含于 **AX/DX**；

除法指令：

DIV　SRC；无符号数除法，被除数隐含于 **AX/DX**，目的操作数商、余数含于 **AX/DX**；

IDIV　SRC；有符号数除法，被除数隐含于 **AX/DX**，目的操作数商、余数含于 **AX/DX**；

符号扩展指令：

CBW；将 **AL** 扩展为 **AX**；

CWD；将 **AX** 扩展为 **DX：AX**；

BCD 数运算调整指令：

加法调整：

AAA；分离 BCD；

DAA；组合 BCD；

减法调整：

AAS；分离 BCD；

DAS；组合 BCD；

乘法分离 BCD 调整：

AAM；

除法分离 BCD 调整：

AAD；

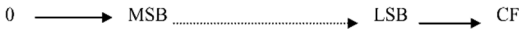
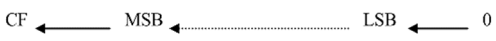
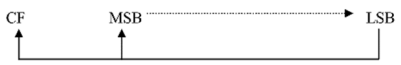
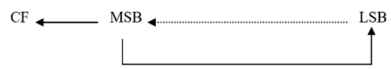
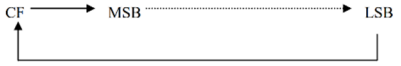
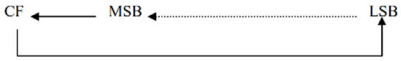
数据位操作指令：

逻辑运算指令：

AND　DST，SRC；(DST) ← (DST) ∧ (SRC), 设置 PSW；

TEST　DST，SRC；　(DST) ∧ (SRC), 设置 PSW；
OR　DST，SRC；
XOR　DST，SRC；
NOT　DST；

移位类指令：

SHR DST, CNT	逻辑右移	
SAR DST, CNT	算数右移	
SHL DST, CNT	逻辑左移	
SAL DST, CNT	算数左移	
ROR DST, CNT	循环右移	
ROL DST, CNT	循环左移	
RCR DST, CNT	带进位循环右移	
RCL DST, CNT	带进位循环左移	

标志位操作指令：

CLC；(CF) ← 0, clear carry flag;
STC；(CF) ← 1, set carry flag;

CMC； $(CF) \leftarrow (\overline{CF})$ **, complement carry flag;**

CLD；(DF) ← 0, clear direction flag;
STD；(DF) ← 1, set direction flag;
CLI；(IF) ← 0, clear interrupt enable flag;
STI；(IF) ← 1, set interrupt enable flag;

分支程序设计：

无条件转移指令：

JMP　LABEL/REG16/MEM；跳转到指定地址执行程序；

有条件转移指令：

（单个标志位）

JC　LABEL；(CF=1)　　**JNC　LABEL；(CF=0)**
JE/JZ　LABEL；(ZF=1)　　**JNE/JNZ　LABEL；(ZF=0)**
JS　LABEL；(SF=1)　　**JNS　LABEL；(SF=0)**
JO　LABEL；(OF=1)　　**JNO　LABEL；(OF=0)**
JP/JPE　LABEL；(PF=1)　　**JNP/JPO　LABEL；(PF=0)**

（无符号数）

JA/JNBE　LABEL；(CF&ZF=0)　　**JAE/JNB　LABEL；(CF=0)**
JB/JNAE　LABEL；(CF=1)　　**JBE/JNA　LABEL；(CF=1 or ZF=1)**

（有符号数）

JG/JNLE　LABEL； $((SF \forall OF) \vee ZF) = 0$　　**JGE/JNL　LABEL；** $SF \forall OF = 0$

JL/JNGE　LABEL; $SF \forall OF = 1$　　**JLE/JNG　LABEL;** $((SF \forall OF) \vee ZF) = 1$

循环程序设计：

循环控制指令：

LOOP　LABEL; (CX) ← (CX) – 1, (CX)=0 时退出；

LOOPZ/LOOPE　LABEL; (CX) ≠ 0 & ZF=1 转移；

LOOPNZ/LOOPNE　LABEL; (CX) ≠ 0 & ZF=0 转移；

JCXZ; CX=0 转移；

字符串操作指令：

MOVSB; (ES:DI) ← (DS:SI), (SI) ← (SI)±1, (DI) ← (DI)±1;

MOVSW; (ES:DI) ← (DS:SI), (SI) ← (SI)±2, (DI) ← (DI)±2;

MOVS　DST, SRC; (ES:DI) ← (DS:SI), (SI) ← (SI)±1/2, (DI) ← (DI)±1/2;

REP; (CX) ← (CX) – 1, (CX) = 0 退出；

REPZ; (CX) ≠ 0 & ZF=1 时重复执行；

REPNZ/REPNE; (CX) ≠ 0 & ZF=1 时重复执行；

CMPSB; (DS:SI) - (ES:DI), (SI) ← (SI)±1, (DI) ← (DI)±1;

CMPSW; (DS:SI) - (ES:DI), (SI) ← (SI)±2, (DI) ← (DI)±2;

CMPS　DST, SRC; (DS:SI) - (ES:DI), (SI) ← (SI)±1/2, (DI) ← (DI)±1/2;

SCASB; (AL) - (ES:DI), (DI) ← (DI)±1;

SCASW; (AX) - (ES:DI), (DI) ← (DI)±2;

SCAS　DST, SRC; (AL/AX) - (ES:DI), (DI) ← (DI)±1/2;

LODSB; (AL) ← (DS:SI), (SI)±1;

LODSW; (AX) ← (DS:SI), (SI)±2;

LODS　DST, SRC; (AL/AX) ← (DS:SI), (SI)±1/2;

STOSB; (ES:DI) ← (AL), (DI)±1;

STOSW; (ES:DI) ← (AX), (DI)±2;

STOS　DST, SRC; (ES:DI) ← (AL/AX), (DI)±1/2;

子程序设计：

CALL LABEL/OPR; 入口地址标号为 LABEL/入口地址操作为 OPR；

RET; 段内子程序返回，(IP) ← (SP)；

RETF; 段间子程序返回，IP，CS 出栈；

过程定义：

name PROC type

…

name ENDP

中断程序设计：

中断调用：

INT n; 调用第 n 号中断

1.PSW，CS，IP 入栈；

2.清除 IF，TF 标志；

3.中断向量表取出中断向量；

4.转到中断服务子程序执行；

5.中断返回；

中断返回：

IRET; IP，CS，PSW 出栈；

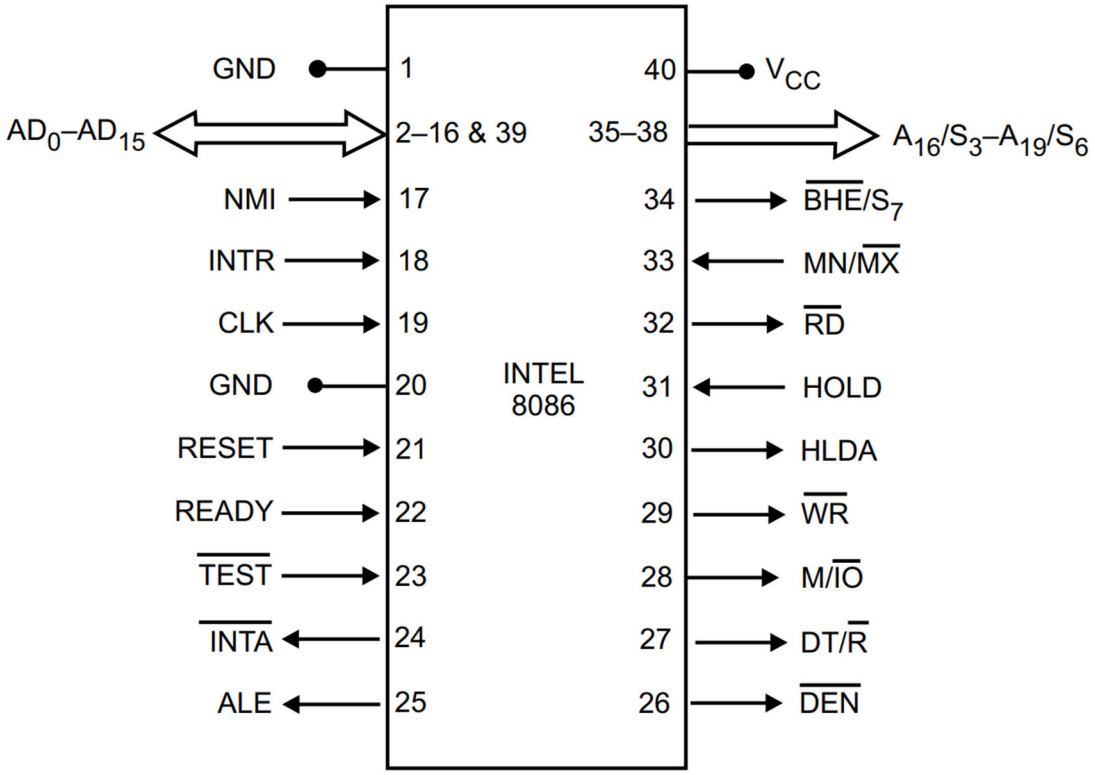
中断向量表：

计算机中用于存储中断向量的最低的 1024 个地址单元，n 号向量为 4n；

总线及其形成

总线：计算机系统中模块/子系统间传输数据、地址、控制信息的公共通道；

最小方式：系统仅有一个微处理器，控制信号全部由 CPU 直接产生；



MN / $\overline{MX}$：工作方式控制线，高电平为最小方式，低电平为最大方式；

CLK：时钟信号输入端；

总线周期：CPU 通过总线对外部进行一次访问所需时间；

RESET：系统复位信号，高电平有效，复位信号上升沿与 CLK 下降沿同步；

$S_3 \sim S_6$：S_6 始终为低电平；

S_5 为 PSW 的 IF 位当前状态；

S_4, S_3：当前正在使用的段寄存器；

S_3	S_4	段寄存器
0	0	ES
0	1	SS
1	0	CS / I/O
1	1	DS

ALE：地址锁存允许信号，高电平有效，表示地址线上信息有效；

$\overline{DEN}$：数据允许信号，表示 CPU 准备好接收、发送数据；

DT / $\overline{R}$：数据收/发信号，表示 CPU 接受或发送数据的状态；

$\overline{WR}$：写信号，表示 CPU 正在执行输出操作；

$\overline{RD}$：读信号，表示 CPU 正在进行输入操作；

$M/\overline{IO}$：区分访问存储器与 I/O 端口；

READY：来自存储器或 I/O 的应答信号；

$\overline{TEST}$：测试信号，每隔 5 个时钟周期对输入段进行测试；

$\overline{BHE}/S_7$：地址信号，在总线周期 T1 状态输处 $\overline{BHE}$，表示使用高 8 位数据线；

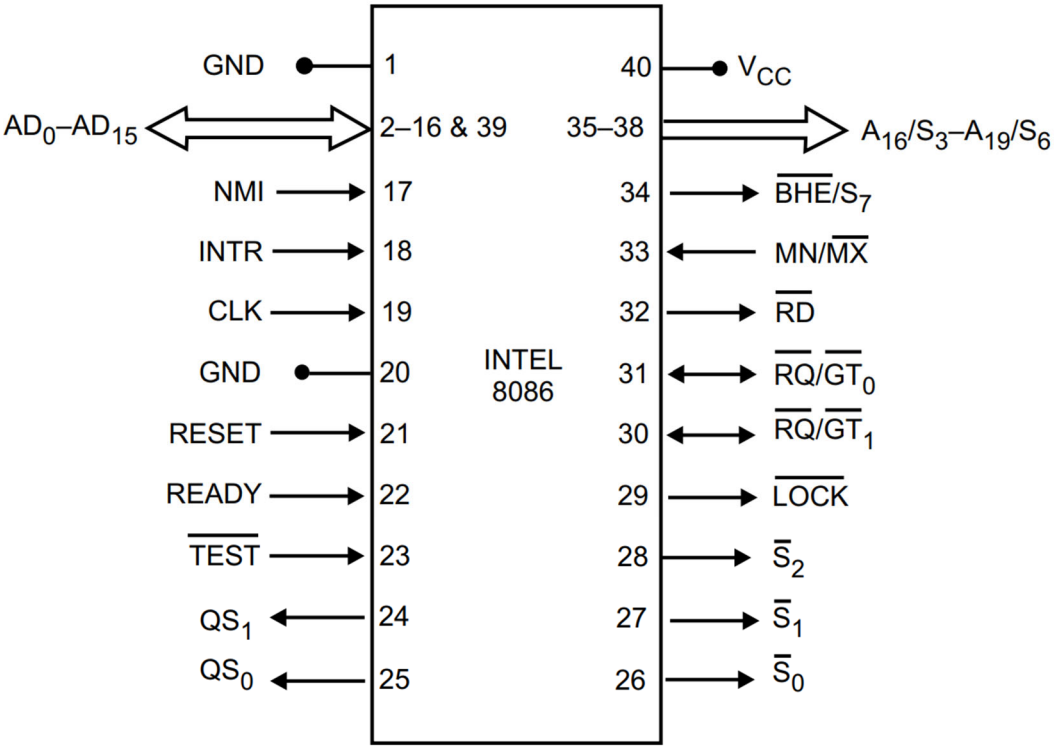
$\overline{BHE}$	A_0	Word/byte access
0	0	Both banks active, 16-bit word transfer on AD_{15} – AD_0
0	1	Only high bank active, upper byte from/to odd address on AD_{15} – AD_8
1	0	Only low bank active, lower byte from/to even address AD_7–AD_0
1	1	No bank active

NMI：外部非可屏蔽中断请求输入信号，上升沿有效；
INTR：外部可屏蔽中断请求输入信号，高电平有效；

$\overline{INTA}$：处理器向中断控制器发出的中断响应信号；

HOLD：系统中其他总线主控设备向 CPU 请求总线使用权的总线申请信号，高电平有效；
HLDA：CPU 对系统中其他总线主控设备请求总线使用权的应答信号，高电平有效；

最大方式：系统有多个微处理器，控制信号由总线控制器间接产生；



QS_0, QS_1：指令队列状态输出线；

QS_1	QS_0	指令队列状态
0	0	无操作
0	1	队列中取出当前指令第一字节
1	0	队列空
1	1	队列中取出指令后续字节

$\overline{S_2}, \overline{S_1}, \overline{S_0}$：状态信号输出线；

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CPU cycles
0	0	0	Interrupt acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	HALT
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

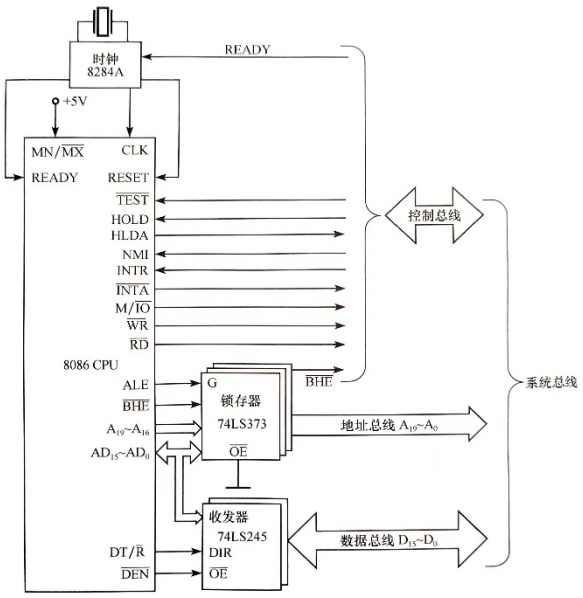
$\overline{LOCK}$：总线锁定信号，不允许其他主控设备占用总线；

$\overline{RQ}/\overline{GT_1}$：输入总线请求信号，低电平有效；

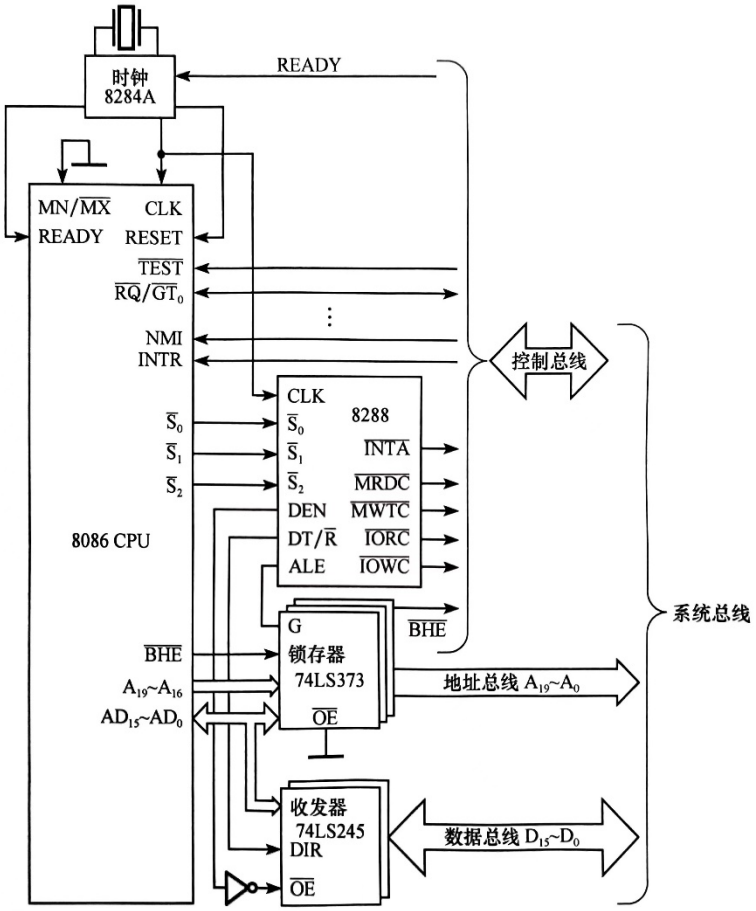
$\overline{RQ}/\overline{GT_0}$：输入总线授权信号，低电平有效，优先级较高；

系统总线形成：

最小方式：



最大方式：



8086 与 8088 的差异：

CPU 内部：

8086CPU 指令队列寄存器为 6 字节，8088 为 4 字节；

CPU 外部：

8086 的 $\boldsymbol{AD_{15} \sim AD_8}$ 在 8088 中为单一地址总线；

最小方式下，8088 中引脚 $\boldsymbol{IO / \overline{M}}$ 与 8086 中对应引脚极性相反；

8086 的 $\boldsymbol{\overline{BHE} / S_7}$ 引脚在 8088 中为 $\boldsymbol{\overline{SS_0}}$ ；

存储器设计

内存储器：

1.随机存取存储器 RAM；

2.只读存储器 ROM；

扩展存储器设计：

位扩展：存储器芯片字长不足 8 位；

地址总线相同，数据总线不同；

字节扩展：增加存储器字节容量；

地址总线不同，数据总线相同；

字节扩展和位扩展；

存储器地址译码：

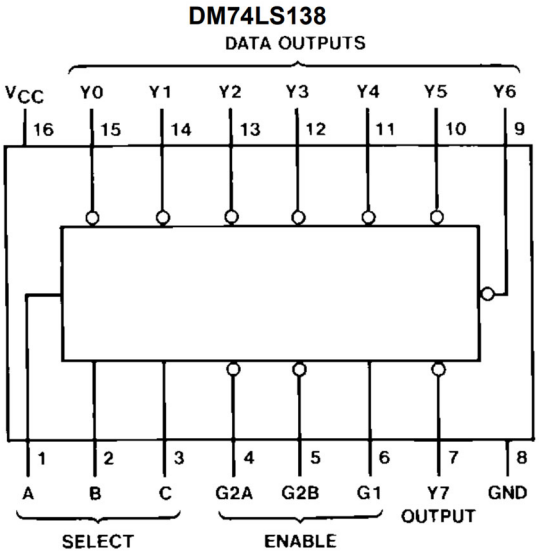
1.全地址译码：除用于存储器芯片片内地址外，微处理器总线的其他地址总线均参加芯片的片选地址译码；

2.部分地址译码：某些高位地址线被省略，简化译码电路，但地址空间有重叠；

3.线选译码：选择除存储器芯片片内寻址外高位地址线某一条，作存储器芯片片选信号；

译码电路：

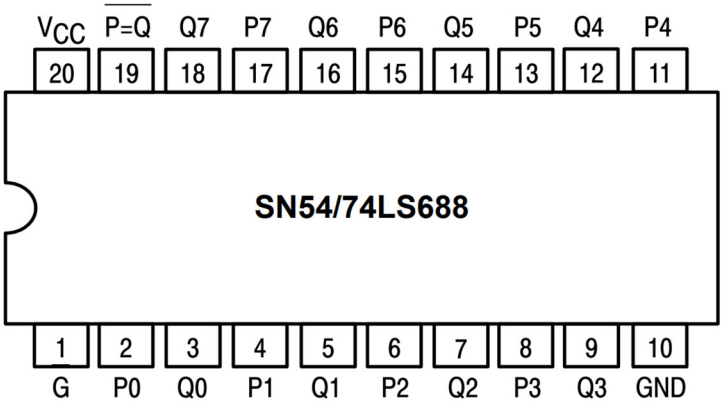
1.74LS138



$G_1 = 1, \overline{G_{2A}} = \overline{G_{2B}} = 0$ 时，74LS138 正常译码；

A 为最低位，C 为最高位；

2.74LS688



INPUTS			OUTPUTS	
DATA	ENABLES			
P, Q	$\overline{G}$, $\overline{GT}$	$\overline{G2}$	$\overline{P = Q}$	$\overline{P > Q}$
P = Q	L	L	L	H
P > Q	L	L	H	L
P < Q	L	L	H	H
X	H	H	H	H

总线竞争：在同一总线上，同一时刻有两个或以上的器件输出状态；

8088 存储器组成：与一般 8 位微机系统中存储器相同；

8086 存储器组成：

对应存储空间有两个存储体，A_0 作片选信号，奇地址与偶地址存储单元属于不同存储体；

$\overline{BHE}$	A_0	Word/byte access
0	0	Both banks active, 16-bit word transfer on AD_{15} – AD_0
0	1	Only high bank active, upper byte from/to odd address on AD_{15} – AD_8
1	0	Only low bank active, lower byte from/to even address AD_7–AD_0
1	1	No bank active

常用芯片的接口技术

I/O 端口：I/O 接口内部可由 CPU 进行 I/O 操作的各种寄存器；

编址方式：

1.独立编址：I/O 端口与存储器有相互独立的地址空间；

2.统一编址：I/O 端口与存储器共享同一个地址空间；

I/O 指令：

IN　DST, SRC; 端口输入指令，DST 只能取 AL/AX；

OUT　DST, SRC; 端口输出指令；

I/O 基本方式：

1.无条件传送方式：

微处理器直接执行预先编制的 I/O 程序实现 I/O 操作；

2.程序查询方式：

微处理器在进行 I/O 操作前不断查询 I/O 设备的状态；

3.I/O 中断方式：

I/O 设备准备就绪并提出中断请求后微处理器予以响应；

4.DMA 方式：

直接在主存储器与 I/O 接口间进行数据传送；

中断系统与可编程中断控制器 8259A

中断：CPU 暂停当前执行程序，转去执行处理该事件的中断服务程序，再返回当前执行程序的过程；
中断断点：中断发生时 CPU 正在执行指令的下一条指令的地址；

中断过程：

1.中断源请求中断：

外部中断源：可屏蔽、不可屏蔽；

内部中断源；

2.中断响应：

可屏蔽中断响应条件：

（1）微处理器 IF=1；

（2）无不可屏蔽中断请求和总线请求；

（3）当前指令执行结束；

不可屏蔽中断响应条件：

（1）无总线请求；

（2）当前指令执行结束；

内部中断响应条件：

（1）当前指令执行结束；

中断响应过程：

1.识别中断源，获得中断类型号；

2.将 PSW，CS，IP 依次压入堆栈；

3.清除 TF 与 IF；

4.获得相应中断服务入口地址，转入中断服务程序；

3.中断服务：

1.保护现场：保护中断服务中要使用的寄存器的内容；

2.开中断：中断服务程序中设 IF=1，保证对更高级别中断请求的相应；

3.中断处理：执行 I/O 操作或处理非常事件；

4.关中断；

5.恢复现场：将堆栈中内容弹出；

6.中断返回：将 IP，CS，PSW 内容弹出，程序回到被中断的地址；

可编程中断控制器 8259A：

功能： 将优先权最高的中断源的中断类型号送到 MPU；

实现中断的嵌套；

引脚：

A_0：地址线，输入，选择内部端口；

$\overline{CS}$：片选信号；

INT：中断请求信号，输出（→INTR）；

$\overline{INTA}$：中断响应信号，输入（←$\overline{INTA}$）；

$CAS_2 \sim CAS_0$：双向的级联线；

$IR_7 \sim IR_0$：外设向 8259A 发送的中断请求信号；

$\overline{SP}$ / $\overline{EN}$：主从设备设定/缓冲器读写控制；

中断优先权管理：

1.固定优先级：各个中断源优先级由引脚决定；

全嵌套方式：0~7 优先级递减；

特殊嵌套方式：级联下，主片>从片；

2.循环优先级：各个中断申请优先级相同；

优先权自动循环：0 首先为最高优先级；

优先权特殊循环：指定最高优先级；

中断屏蔽方式：

1.普通屏蔽方式：将中断屏蔽字写入 IMR 实现；

2.特殊屏蔽方式：允许低级别中断；

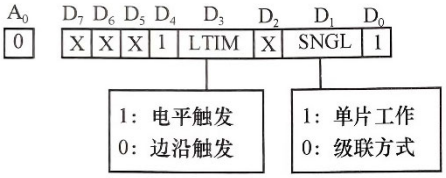
中断结束方式：

1.中断自动结束（AEOI）

2.中断正常结束（EOI）

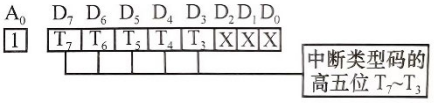
8259A 编程：

1.初始化命令字：



初始化 ICW_1：

$D_4 = 1$，D_3 设定中断请求信号触发方式，D_1 设定单片/级联；

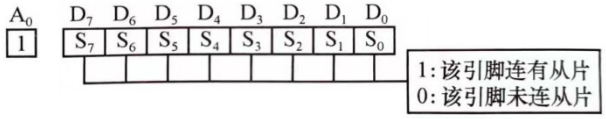


初始化 ICW_2：

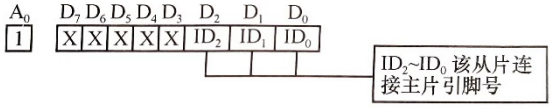
紧跟在初始化 1 后，$D_2 \sim D_0$ 确定中断类型号基值（IR_0）；

初始化 ICW_3：

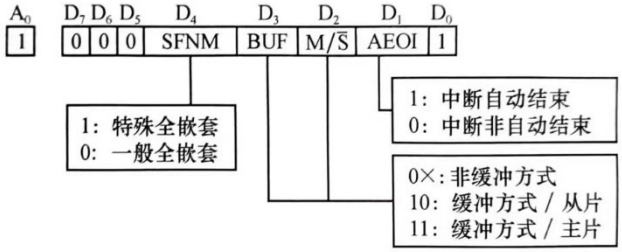
专用于级联方式，主片：



从片：

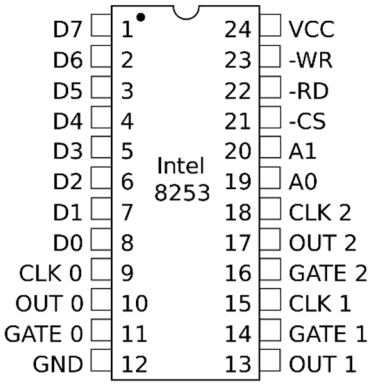


初始化 $\boldsymbol{ICW_4}$：

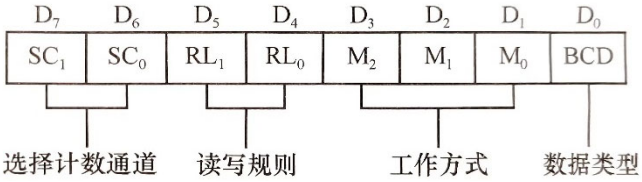


设定中断结束方式、缓冲方式、中断嵌套方式；

定时/计数器 8253 应用设计



8253 控制字与工作方式：



SC_1	SC_0	含义
0	0	计数通道 0
0	1	计数通道 1
1	0	计数通道 2
1	1	保留

RL_1	RL_0	含义
0	0	计数通道锁存命令
0	1	只读写 CR/OL 的低 8 位
1	0	只读写 CR/OL 的高 8 位
1	1	先读写 CR/OL 的低 8 位，再读写高 8 位

M_2	M_1	M_0	含义
0	0	0	方式 0
0	0	1	方式 1
×	1	0	方式 2
×	1	1	方式 3
1	0	0	方式 4
1	0	1	方式 5

8253 的工作方式：

方式 0：计数达到终值时中断

向 8253 置方式字或置时常数时，OUT 输出变成低电平；

置入时常数后，下个 CLK 脉冲，使 CR 内容置入计数单元；

GATE=1 时允许计数，为 0 时暂停计数；

计数至 0 时，OUT 由低变高，继续计数；

GATE=1，置时常数 N 后，要经过 N+1 个时钟周期才可 OUT 输出高电平；

方式 1：硬件触发的单脉冲形成

置方式 1 的控制字或置时常数时，OUT 输出高电平；

GATE 端输入有效触发信号，经过一个 CLK，OUT 变为低电平，开始减 1 计数；

CE 计数到 0 时，OUT 变为高电平；

计数通道时常数为 N 时，硬件触发产生的低电平宽度为 N 个 CLK；

方式 2：分频脉冲形成

置方式 2 的控制字后，OUT 变为高电平；

置时常数后的 CLK 期间，CR 读入 CE，开始减 1 计数；

CE 计数到 01 时，OUT 端输出负脉冲，重新读入时常数计数；

计数通道时常数为 N 时，OUT 产生的信号为计数时钟的 N 分频；

方式 3：方波信号形成

置方式 3 的控制字后，OUT 变为高电平；

置偶时常数后，OUT 变为高电平，开始减 2 计数，CE 为 0 时，OUT 变为低电平，重新读入时常数计数，再次为 0 时，OUT 变为高电平；

计数通道时常数为 N 时，OUT 产生的信号为计数时钟的 N 分频方波信号；

方式 4：软件触发产生选通信号

置方式 4 的控制字或置入时常数后，OUT 变为高电平；

软件触发到产生有效低电平间隔 N+1 个 CLK；

减 1 计数至 0 后，OUT 变为低电平；

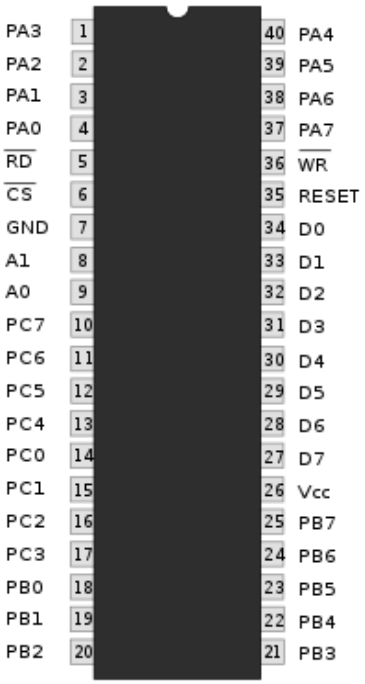
方式 5：硬件触发产生选通信号

置方式 5 的控制字或置入时常数后，OUT 变为高电平；

GATE 端每个上升沿都将在 OUT 端产生选通信号，硬件触发到产生有效低电平间隔 N+1 个 CLK；

计数器的计数操作不受 GATE 高低电平控制；

并行接口芯片 8255A 应用设计



PA0~PA7：8 位数据输出锁存/缓冲器，8 位数据输入锁存器；（端口 A）

PB0~PB7：8 位数据 I/O 锁存/缓冲器，8 位数据输入锁存器；（端口 B）

PC0~PC7：8 位输出锁存/缓冲器，8 位数据输入缓冲器；（端口 C）

A 组：**PC7~PC4, PA7~PA0；**（可设为方式 0~2）

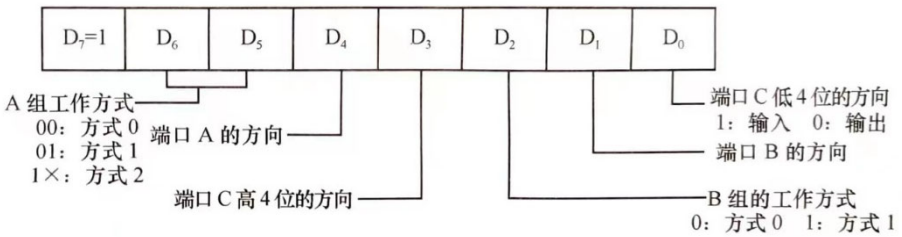
B 组：**PC3~PC0, PB7~PB0；**（可设为方式 0，1）

端口寄存器：

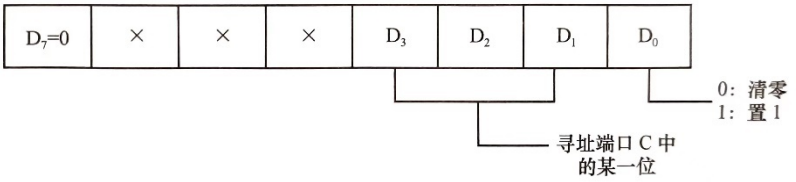
A_1	A_0	含义
0	0	访问端口 A
0	1	访问端口 B
1	0	访问端口 C
1	1	访问控制寄存器

8255A 控制字：

方式控制字：



置位控制字：



8255A 工作方式：

方式 0：基本 I/O

方式 1：有联络信号的 I/O

方式 2：双向传送