《微机原理与接口》

第2章 微机原理(8088)

教师：苏曙光

华中科技大学软件学院

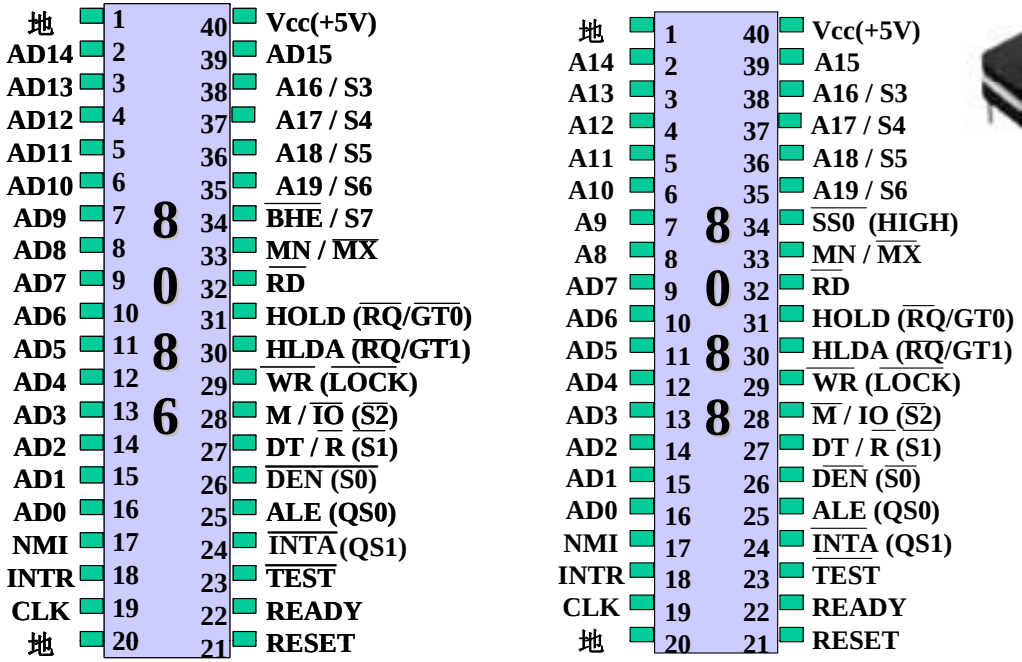
- 第二章 微机原理
  - 第1节 8088CPU的基本原理
  - 第2节 数字电路、常用门和IC芯片
  - 第3节 8088CPU外部结构
  - 第4节 8088CPU总线时序

- 本章重点
  - 8088内部结构
  - 8088外部引脚；
  - 8088内部寄存器；
  - 8088的存储器组织；
  - 8088的工作时序
  - 数字电路回顾：常用门和IC芯片

第1节 8088CPU的基本原理

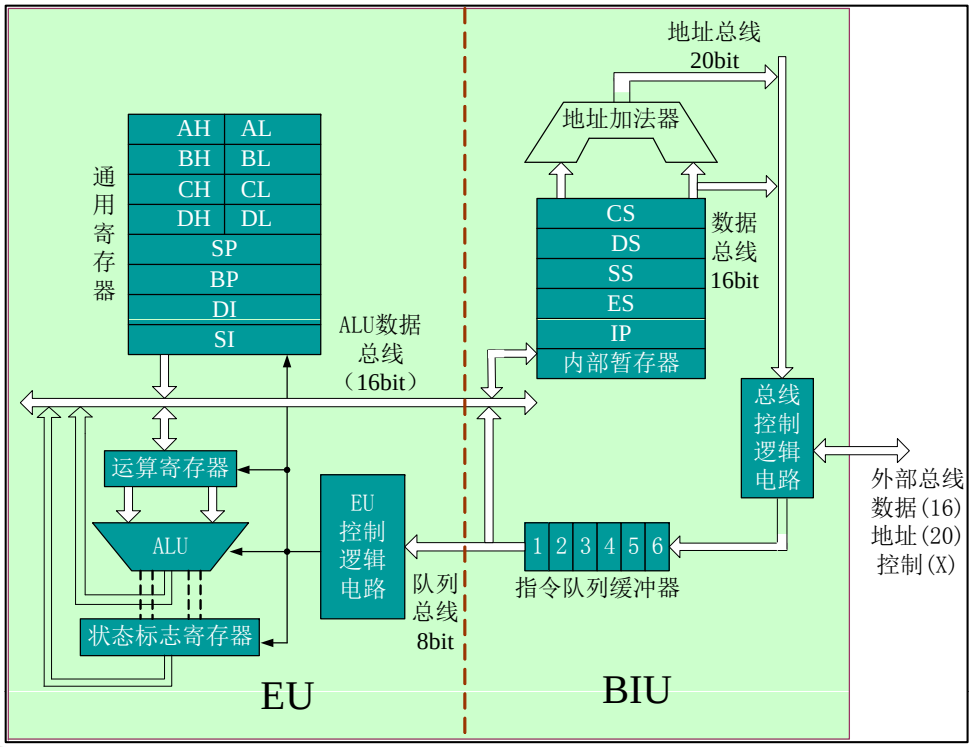
- 8086/8088 MPU
  - 16位CPU：内部寄存器16位；
  - 20根地址线：1MB内存；
  - 8086和8088差异：数据总线，指令长度
    - 8086：内外16根
    - 8088：内部16根，外部8根 ：准16位机；
    - 8086指令队列：6字节
    - 8088指令队列：4字节

● 8086/8088 MPU管脚



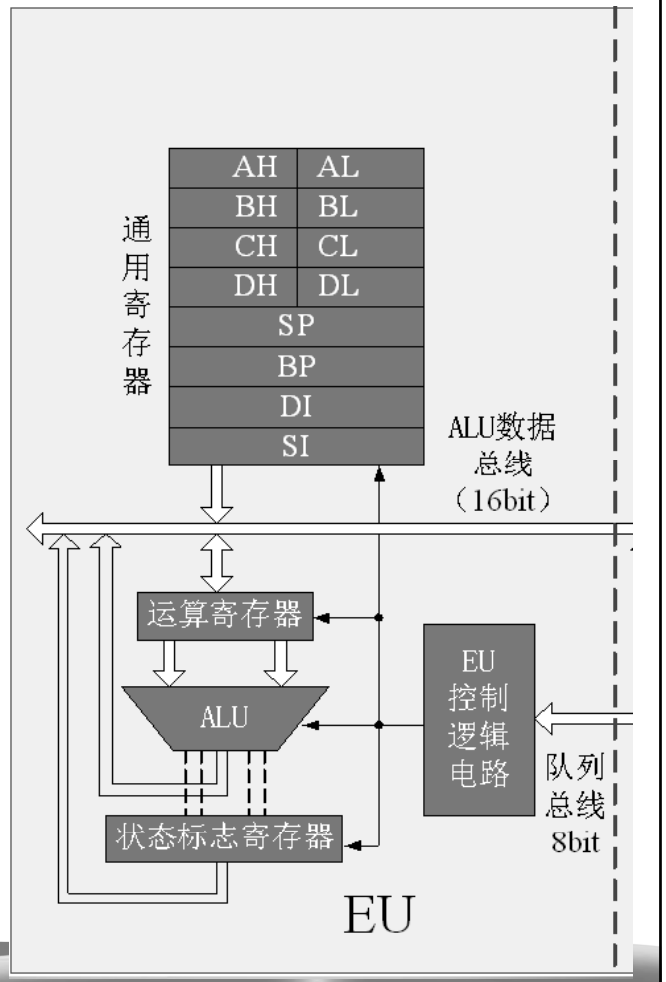
8088 内部结构：EU和BIU

- **EU（Execute Unit，执行单元）：负责执行指令或运算**
- **BIU (Bus Interface Unit，总线接口单元）：负责读指令或数据**



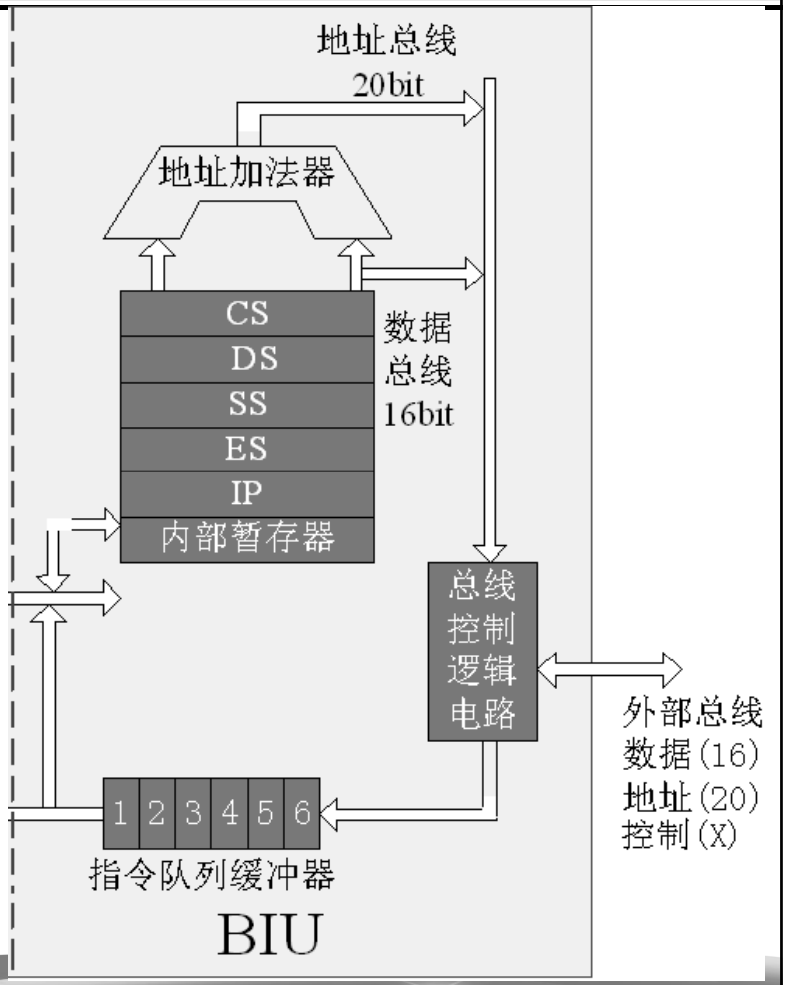
EU功能和内部构成

- 功能:负责执行指令或运算
  - 从指令队列中取指令代码，译码，在**ALU**中完成数据的运算，结果的特征保存在标志寄存器中。
- 内部构成
  - **1）ALU**：执行基本运算和处理.
  - **2）**一组通用寄存器 **+** 标志寄存器
  - **3）EU**控制系统：队列控制和时序控制

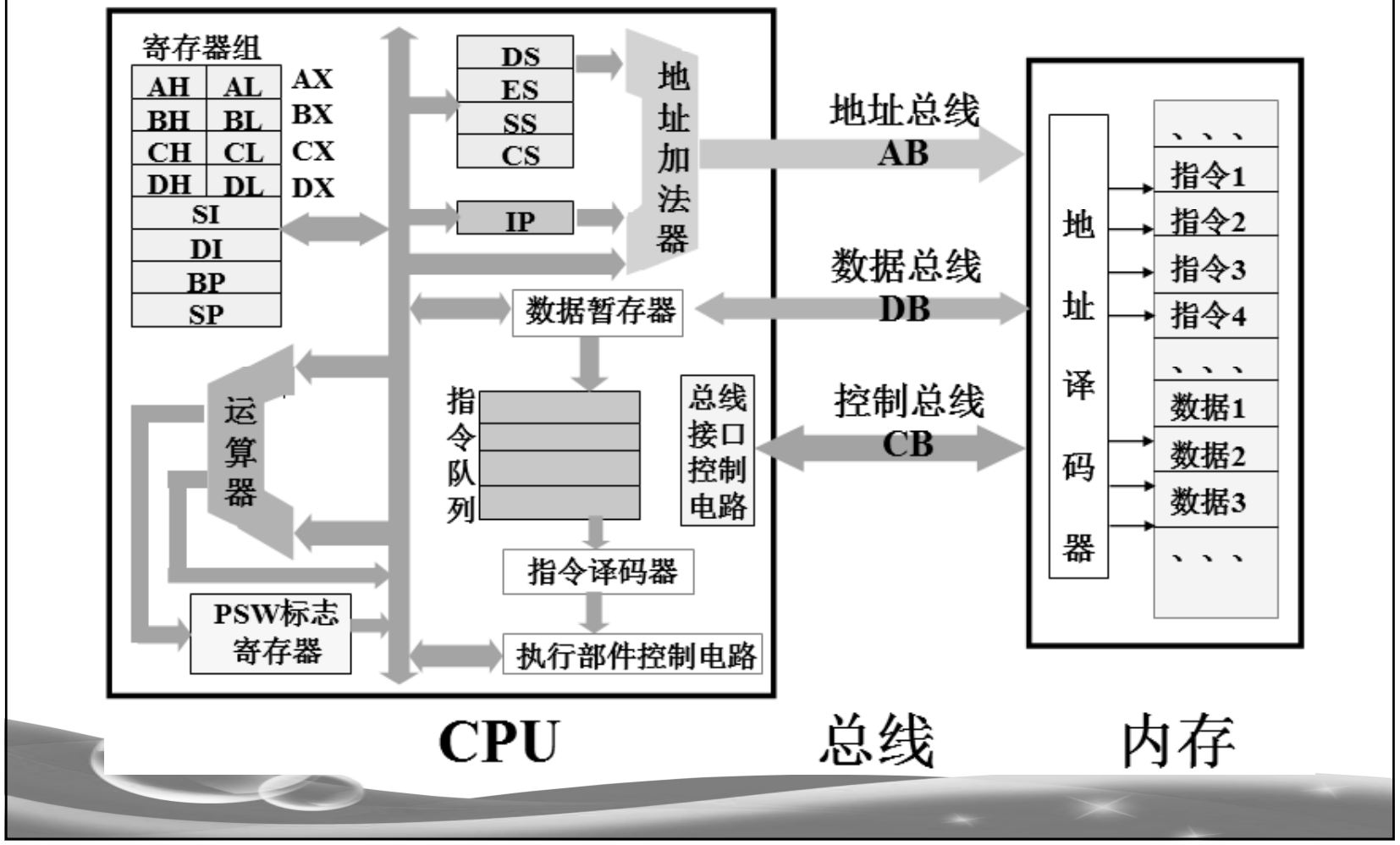


BIU功能和内部构成

- 功能
  - 具有预取指令的功能
    - ◆执行指令的同时从内存取下一条或几条指令放在队列中。
  - 指令队列：6字节或4字节。
  - 指令执行顺序
    - ◆顺序指令执行：指令队列存放紧接在执行指令的下一条指令。
    - ◆执行转移指令后： 清除队列。从新地址取指令，并立即送往执行单元。
- 构成
  - 1)一组段寄存器+ 指令指针IP
  - 2)地址加法器：将段地址和偏移地址相加，形成20位物理地址
  - 3)指令队列缓冲器：寄存指令。
  - 4)总线控制逻辑：内外总线接口。

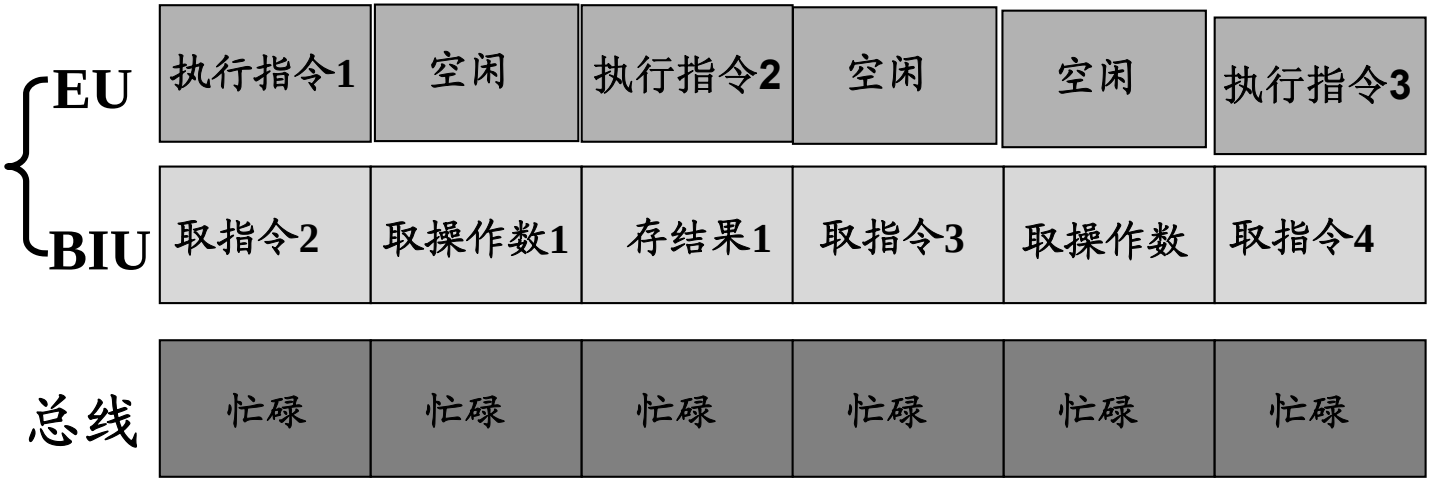


8088工作原理：取指令，执行指令



8088并行工作方式：流水线

- 指令预取队列的存在使**EU**和**BIU**可同时工作
- **2**级流水线



8086/8088 CPU的特点

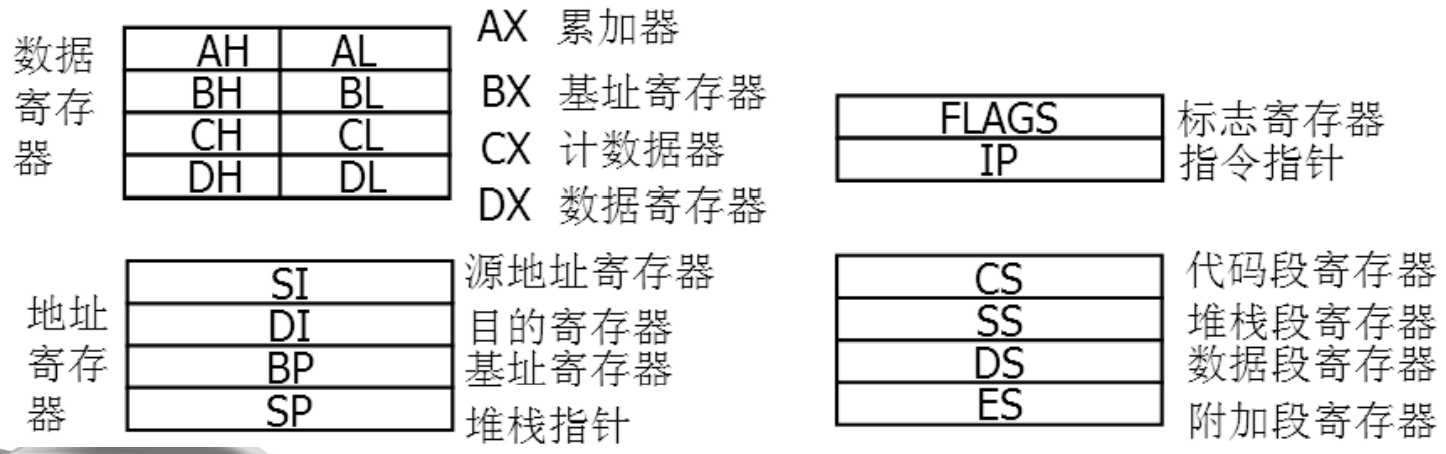
- 采用并行流水线工作方式
- 支持多处理器系统
- 片内无浮点运算部件，浮点运算由数学协处理器**8087**支持（也可用软件模拟）
  - 注：**80486DX**以后的**CPU**均将数学协处理器作为标准部件集成到**CPU**内部
- 对内存空间实行分段管理

第2节 8088的运行（执行）环境

- 8088的运行（执行）环境
  - 寄存器
  - 内存空间
  - 堆栈（Stack）
  - I/O端口

14个基本寄存器

- **8个通用寄存器(General Registers)**
- **1个标志寄存器(F Flags Register)；**
- **1个指令指针寄存器 (IP: Instruction Point Register)**
- **4个段寄存器(Segment Registers)。**
  - **CS,DS,SS,ES**



4个数据寄存器：AX,BX,CX和DX

- 常用来存放参与运算的操作数或运算结果
- 16位数据寄存器，分为8个8位寄存器
  - AX： AH，AL
  - BX： BH，BL
  - CX： CH，CL
  - DX： DH，DL
- 8位可以单独操作，不影响另外8位

- 习惯使用
  - AX：累加器。多用于存放中间运算结果。使用频　度最高，用于算术、逻辑运算以及与外设传送信息等；
  - BX：基址寄存器。常用于存放内存地址；
  - CX：计数寄存器。用于在循环或串操作指令中存放循环次数或重复次数；
  - DX：数据寄存器。在32位乘除法运算存放高16位数；

- 寄存器的特殊使用
  - **AX—**操作数和结果数据的累加器；
  - **BX—**在**DS**段中数据的指针；
  - **CX—**串和循环操作的计数器；
  - **DX—I/O**指针(端口地址)；

4个段寄存器：CS,DS,ES和SS

- 用于存放逻辑段的段基地址
  - **CS**：代码段寄存器 **Code Segment**
    - 代码段用于存放指令代码
  - **DS**：数据段寄存器 **Data Segment**
  - **ES**：附加段寄存器 **Extended Segment**
    - 数据段和附加段用来存放操作数
  - **SS**：堆栈段寄存器 **Stack Segment**
    - 堆栈段用于存放返回地址，
    - 保存寄存器内容，传递参数

2个指针寄存器： SP和BP

- **SP,BP**
  - 指针寄存器，用于寻址堆栈内的数据
  - 与段寄存器**SS**联合使用，确定堆栈中的单元地址
- **SP**
  - 堆栈指针寄存器，其内容为栈顶的偏移地址；
- **BP**
  - 基址指针寄存器，表示数据在堆栈中的基地址。

- **BX**与**BP**在应用上的区别
  - 作为通用寄存器，二者均可用于存放数据；
  - **BX**通常用于寻址数据段**DS**或扩展**ES**段
  - **BP**则通常用于寻址堆栈段**SS**。

2个变址寄存器：SI和DI

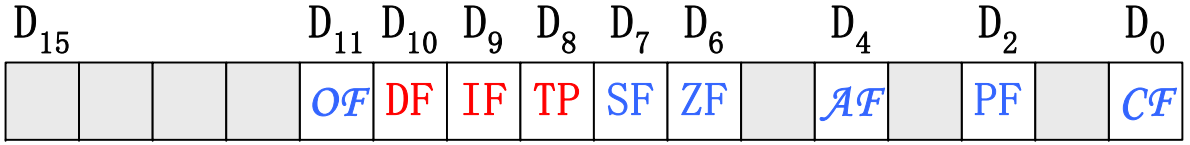
- **SI,DI**
  - 常用于指令的间接寻址或变址寻址。
- **SI**：源变址寄存器
  - 指向**DS**段中的数据指针、串操作的源指针；
- **DI**：目的变址寄存器
  - 指向**ES**段中的数据指针、串操作的目标指针；

1个指令指针寄存器：IP

- 存储**CPU**将要执行的下一条指令的偏移地址；
- **CPU**在执行完一条指令之后，会自动将下一条指令的偏移地址存入到**IP**中。

1个状态标志寄存器： F

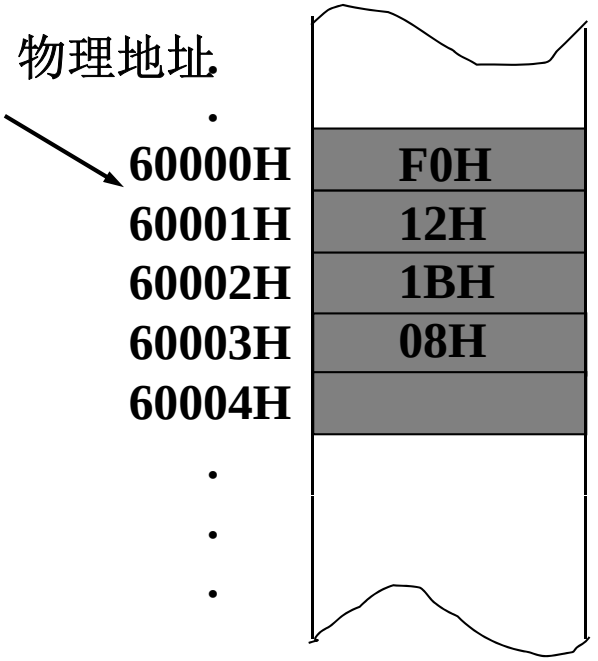
- 16位，包含9个标志位(6个状态位，3个控制位)



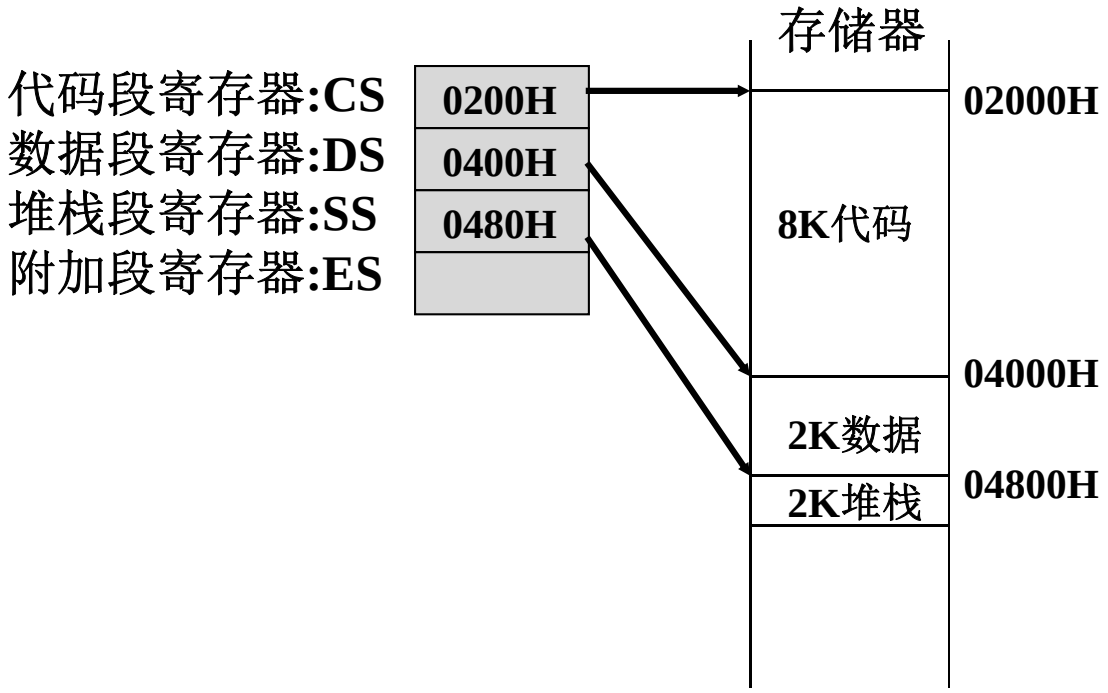
  - ■状态位：标示算术、逻辑运算的结果状态
  - ■控制位：控制CPU操作
  - ■状态位的例子
    - ◆OF(Overflow Flag)，溢出标志位
      - □功能：标示符号数的运算结果是否溢出
  - ■控制位的例子
    - ◆DF(Direction Flag)，方向位
      - □功能：用于控制字符串操作的地址步进方向

8088的存储器结构

- 存储空间
  - **20**根地址线：**1M**
- 问题：
  - 寄存器**16**位，如何生成**20**位地址？
  - 解决：存储器分段



- 分段管理
  - 段起始地址(20位)的高十六位称为该段的段地址
    - 段地址放在DS、SS、CS、ES中


存储器
代码段寄存器:CS
数据段寄存器:DS
堆栈段寄存器:SS
附加段寄存器:ES
0200H
0400H
0480H
02000H
8K代码
04000H
2K数据
04800H
2K堆栈

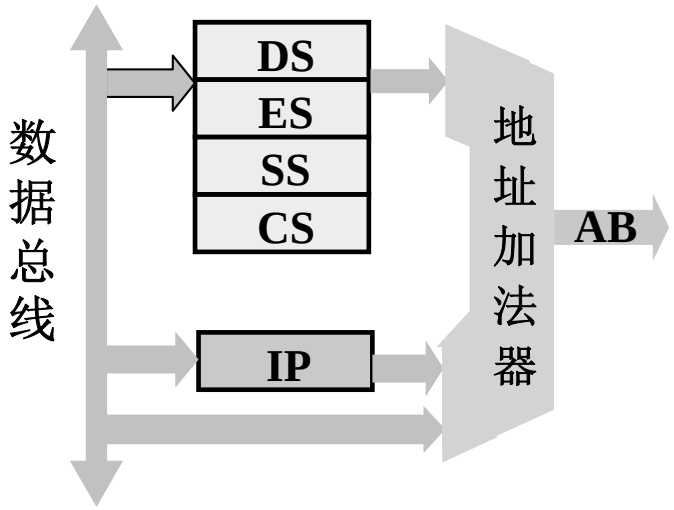
- 分段管理
  - 段起始地址(20位)的高十六位称为该段的段地址
    - 段地址放在DS、SS、CS、ES中
  - 段内某个单元到段首的偏移称为偏移地址；
    - 偏移地址放在BX、SI、DI、 BP 、SP、IP中。
    - 最大段：64K = 2^{16}B
  - 例如

    DS:BX， DS:SI (数据区)

    SS:SP (堆栈区)

    CS:IP (代码区)
  - 物理地址 =段地址（16位）* 10H +偏移（16位）

- 物理地址 =段地址（16位）* 10H +偏移（16位）



数据存放规律

- 字节数据
  - ■ 一单元存放一个数

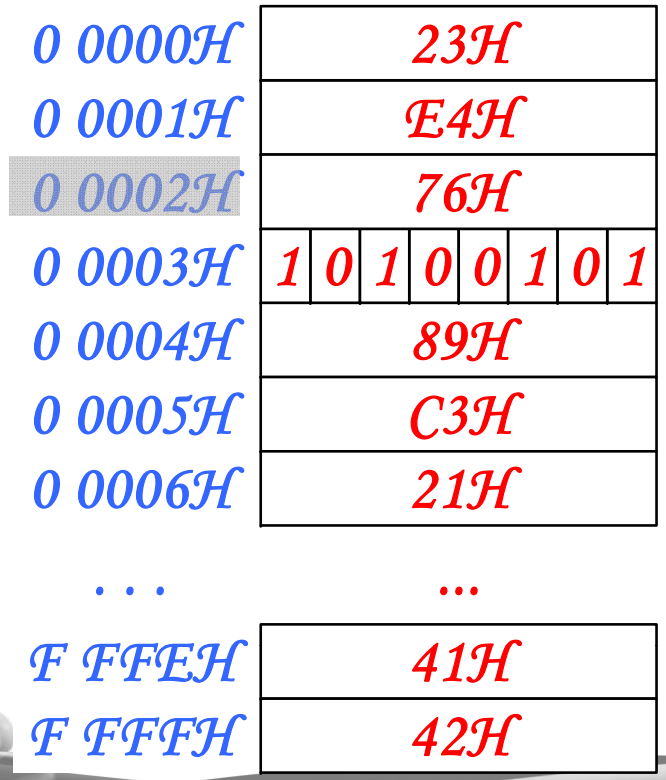
    例子：**E4H** 放在 **00001H**单元；
- 字数据
  - ■ **2**单元：“低对低，高对高”
  - ■ 字的地址：**2** 个单元中的低地址

    例子：**76E4H**放在**00001H**地址中
- 机器指令(机器码)：
  - ■ 按字节顺序地址递增存放

    如：**MOV BX，AX ；89C3H，**放在**00004H**单元
- 字符串：按地址递增存放，同机器指令。

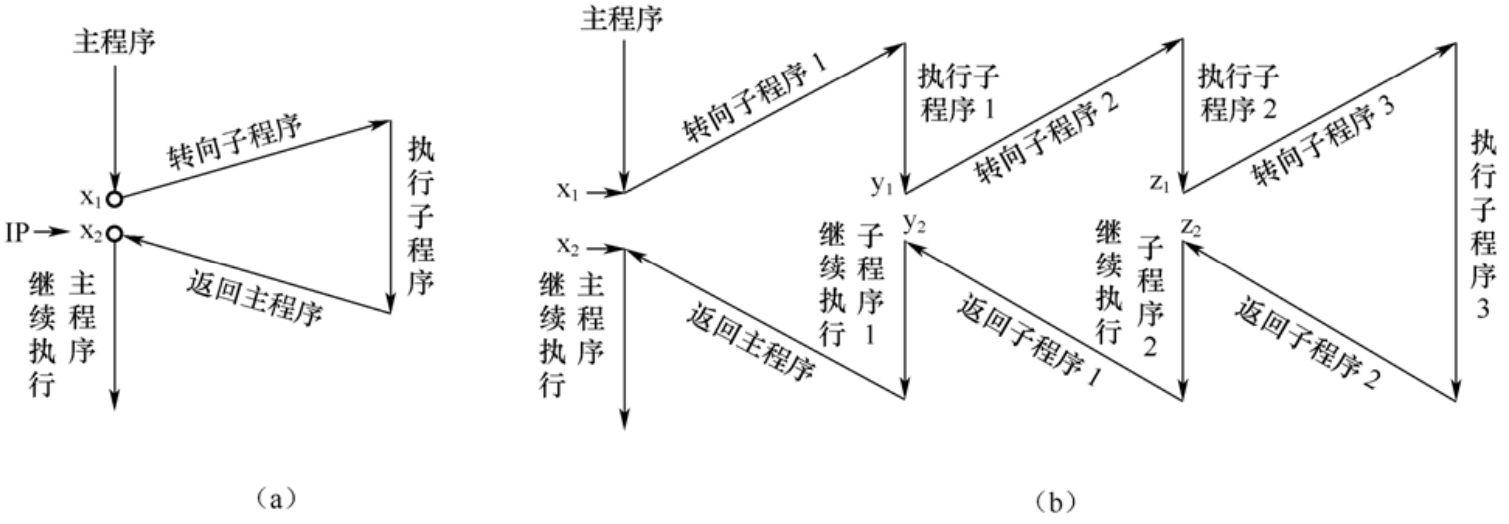
0 0000H	23H
0 0001H	E4H
0 0002H	76H
0 0003H	1 0 1 0 0 1 0 1
0 0004H	89H
0 0005H	C3H
0 0006H	21H
. . .	...
F FFEH	41H
F FFFH	42H

- 练习题
  - **00002H单元存放的字节/ 字/ 双字节指令为多少？**



堆栈

- 例子：子程序调用的过程



- 调用发生后，主程序在**CPU**中的运行环境被破坏。
- 调用返回时，必须恢复主程序之前的运行环境

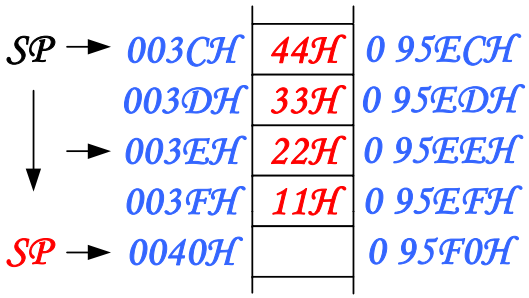
- 堆栈（**STACK**）
  - 功能一：在子程序调用和中断服务时存储现场数据；
  - 特殊内存
    - “后进先出”**(LIFO)**存储
    - 堆栈一端固定**(**栈底**)**，另一端活动**(**栈顶**)**，数据只允许从栈顶存取（进或出）
    - 栈指针：指示栈顶位置（**Stack Poniter**，**SP**）
  - 堆栈的伸展方向
    - 栈底的地址大，栈顶的地址小
  - 栈的操作（**PC**）
    - 入栈：将一个数存入栈顶，并改变**SP** （变小）
    - 出栈：从栈顶读出一个数据,并改变**SP** （变大）

- 入栈操作
  - **PUSH SRC; SRC** 代表寄存器或存储单元地址
  - 功能：将寄存器或存储单元中的一个字压入堆栈
  - 操作：
    - “先减后入”：**SP-1 → SP，** 字高位 → **[SP]**
      SP-1 → SP， 字低位 → **[SP]**
  - 结果：**SP-2,** 数据高对高，低对低存放。
  - 例：**AX=1122H，BX=3344H**
    SS=095BH，SP=0040H
    执行：**PUSH AX ;SP=003EH**
    PUSH BX ;SP=003CH

- 出栈操作
  - **POP DST ；DST** 代表寄存器或存储单元地址
  - 功能：将栈顶一个字传送到寄存器或存储单元中
  - 操作
    - “先出后加”：**[SP]** → 字低位， **SP+1 → SP**
      [SP] → 字高位，**SP+1 → SP**
  - 结果：**SP+2**, 数据低对低,高对高存放
  - 例：上述前一例子中再执行：

    POP CX ；CX=3344H, SP=003EH

    POP DX ；DX=1122H, SP=0040H



PUSHA

- （Push All）将所有16位通用寄存器存入堆栈。
  - Temp ←（SP）；
  - Push（AX）；
  - Push（CX）；
  - Push（DX）；
  - Push（BX）；
  - Push（Temp）；//SP
  - Push（BP）；
  - Push（SI）；
  - Push（DI）；

POPA

- **Pop All，自堆栈弹出至相应的16位通用寄存器。**
  - **DI←Pop( );**
  - **SI←Pop( );**
  - **BP←Pop( );**
  - **SP增量2（跳过堆栈的下2个字节）**
  - **BX←Pop( );**
  - **DX←Pop( );**
  - **CX←Pop( );**
  - **AX←Pop( );**

- **Flag寄存器出/入栈**
  - ■命令格式

    PUSHF; F入栈， SP-2 →SP

    POPF ; F出栈， SP+2 → SP
  - ■功能： 保护和恢复状态标志寄存器**Flag**
- 注意：
  - ■栈操均以字为单位，下列指令均错：

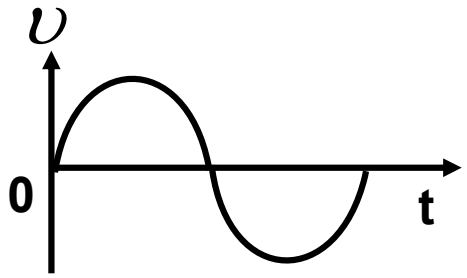
    PUSH AL

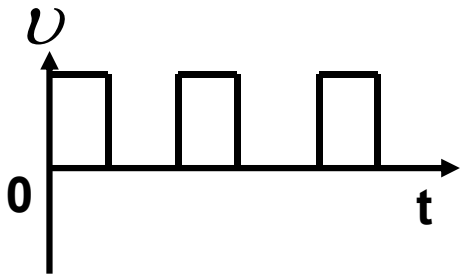
    POP DH
  - ■**PUSH**与**POP**成对，避免堆栈溢出或程序出错；
  - ■堆栈实为内存区，还可按数据区的方法对其操作。

第3节 数字电路、常用门和IC芯片

- 本节基本内容
  - 数字电路基本概念
  - 常用门电路
  - 常用IC芯片

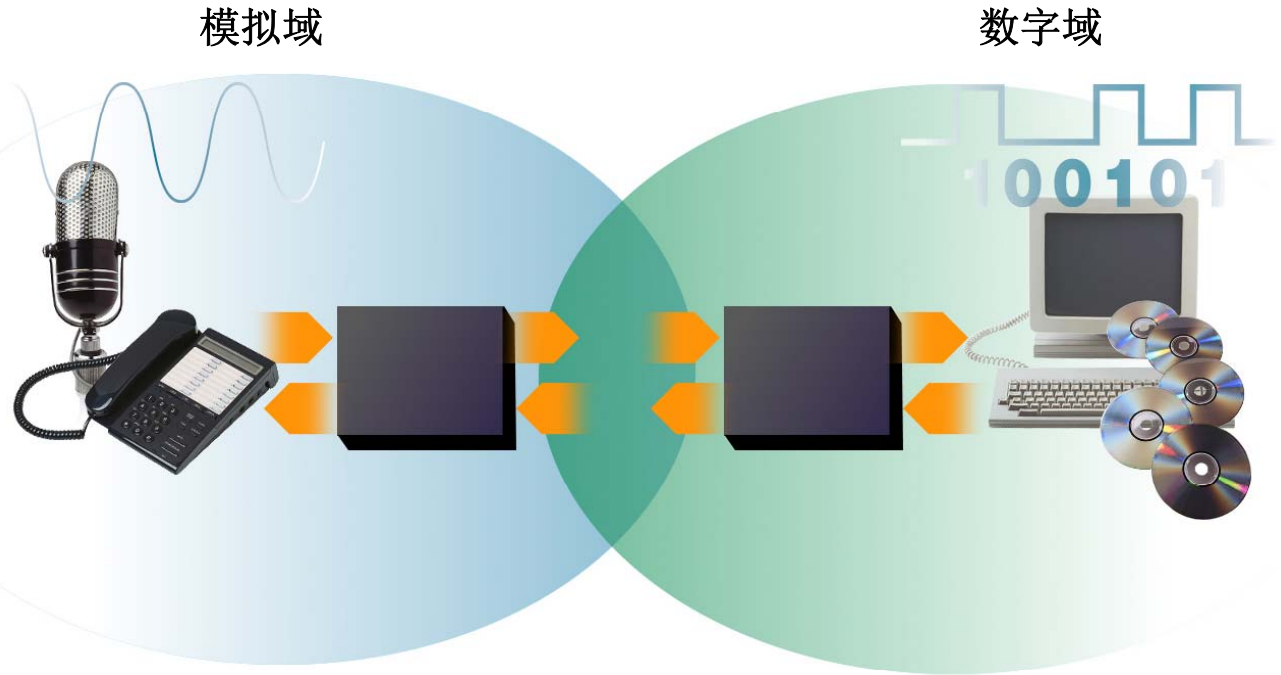
- 电路中两类信号
  - 模拟信号：在时间上和幅值上均连续的信号
  - 数字信号：在时间上和幅值上均离散的信号





- 两类电路
  - 模拟电路：处理模拟信号的电路
  - 数字电路：处理数字信号的电路

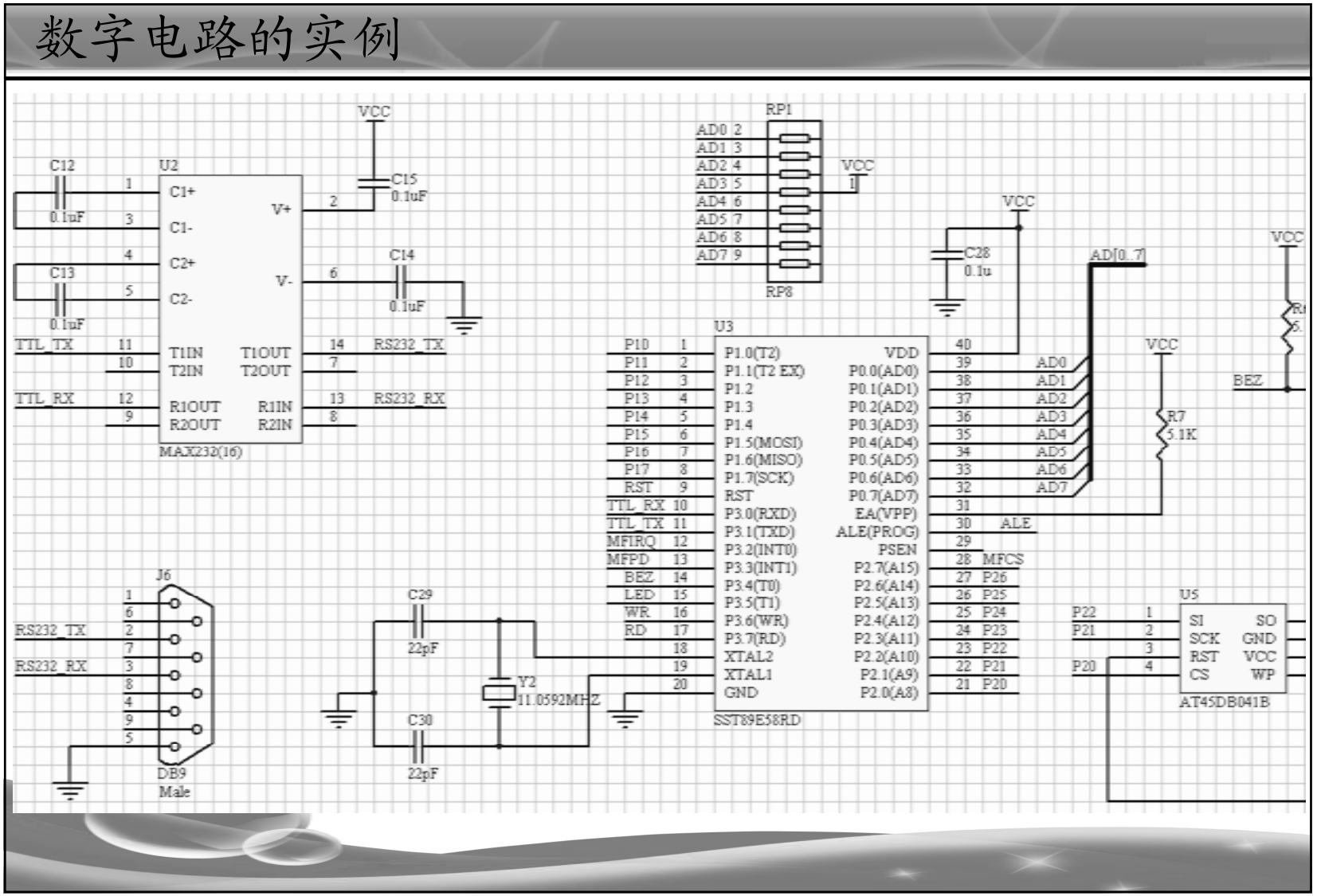
- 数字电路与模拟电路的联系



- **0和1：两个数字，高低两种电平，两种逻辑，…**
- 电路中半导体管工作在开关状态
  - 二极管工作在导通态和截止态
  - 三极管工作在饱和态和截止态
- 基本逻辑运算
  - 与、或、非。
  - 任何复杂逻辑运算通过三种基本运算来实现。
- 使用标准化，积木式的元件/芯片构建电路系统。
  - 硬件设计“软件化”
  - **CPLD/FPGA**

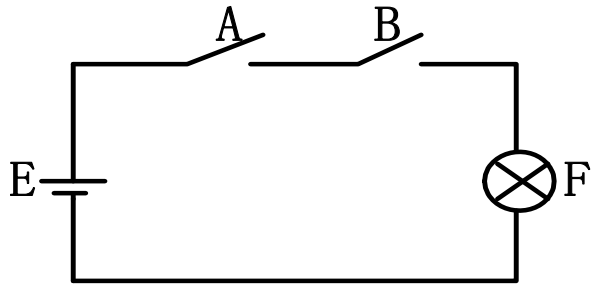
- 数字电路的应用
  - 通信、计算机、网络、家电、数码产品、雷达、自动控制（数控等）、仪器仪表、电子测量等。
  - 《数字电子技术》是一门极其重要的技术基础课。

数字电路的实例


VCC
RP1
AD0 2
AD1 3
AD2 4
AD3 5
AD4 6
AD5 7
AD6 8
AD7 9
RP8
C12
0.1uF
U2
C1+
C1-
C2+
C2-
V+
V-
C15
0.1uF
C14
0.1uF
C13
0.1uF
TTL_TX
TTL_RX
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
RS232_TX
RS232_RX
MAX232(16)
C28
0.1u
AD[0..7]
R7
5.1K
BEZ
U3
P1.0(T2)
P1.1(T2 EX)
P1.2
P1.3
P1.4
P1.5(MOSI)
P1.6(MISO)
P1.7(SCK)
RST
P3.0(RXD)
P3.1(TXD)
P3.2(INT0)
P3.3(INT1)
P3.4(T0)
P3.5(T1)
P3.6(WR)
P3.7(RD)
XTAL2
XTAL1
GND
VDD
P0.0(AD0)
P0.1(AD1)
P0.2(AD2)
P0.3(AD3)
P0.4(AD4)
P0.5(AD5)
P0.6(AD6)
P0.7(AD7)
EA(VPP)
ALE(PROG)
PSEN
P2.7(A15)
P2.6(A14)
P2.5(A13)
P2.4(A12)
P2.3(A11)
P2.2(A10)
P2.1(A9)
P2.0(A8)
SST89E58RD
ALE
MFCS
J6
DB9
Male
C29
22pF
C30
22pF
Y2
11.0592MHZ
U5
SI
SCK
RST
CS
SO
GND
VCC
WP
AT45DB041B

逻辑代数（续）

- “与”运算
  - 只有决定一事件的全部条件都具备时，这件事才成立；如果有一个或一个以上条件不具备，则这件事就不成立。这样的因果关系称为“与”逻辑关系



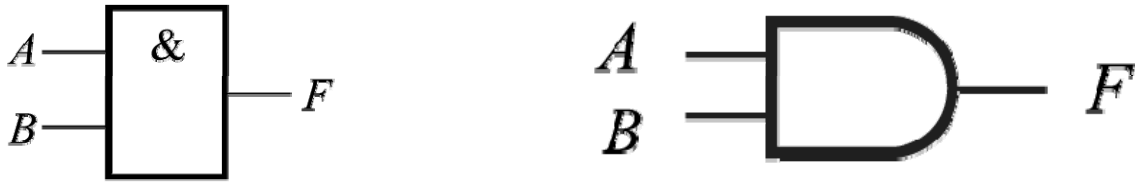
与逻辑电路

与逻辑真值表

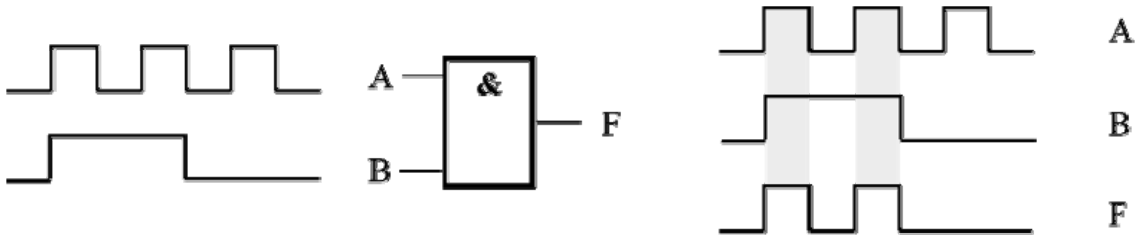
A	B	F=A · B
0	0	0
0	1	0
1	0	0
1	1	1

与的逻辑功能概括：
1）有“0”出“0”；
2）全“1”出“1”。

- “与”运算的符号

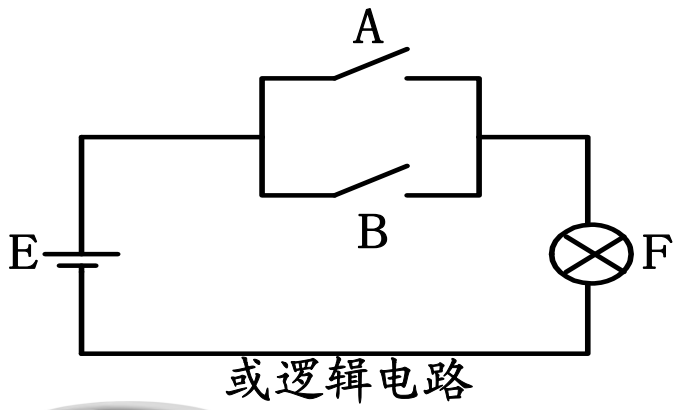


- 例：已知与门的输入波形，求其输出波形**F**。



逻辑代数（续）

- “或”运算
  - 在决定一事件的各种条件中，只要有一个或一个以上条件具备时，这件事就成立；只有所有条件都不具备时，这件事就不成立。这样的因果关系称为“或”逻辑关系。



或逻辑电路

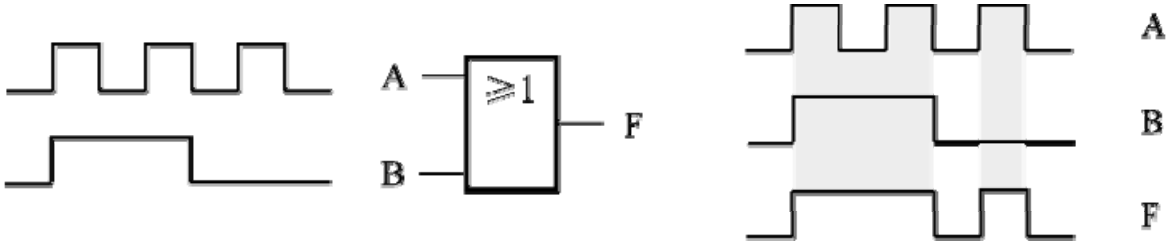
或逻辑真值表

A	B	F=A+ B
0	0	0
0	1	1
1	0	1
1	1	1

或门的逻辑功能概括:
1）有“1”出“1”；
2）全“0”出“0”.

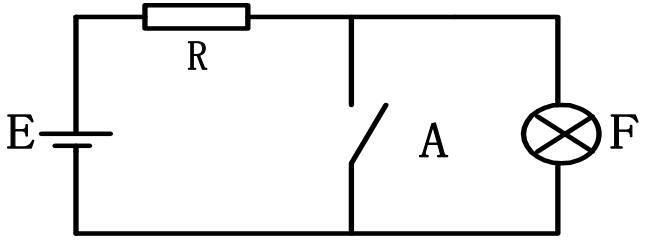
- “或”运算的符号

- 例：已知或门的输入波形，求其输出波形**F**。



逻辑代数（续）

- “非”运算
  - 假定事件F成立与否同条件A的具备与否有关；若A具备，则F不成立；若A不具备，则F成立。F和A之间这种因果关系称为“非”逻辑关系
  - 逻辑表达式
    - $F=\bar{A}$



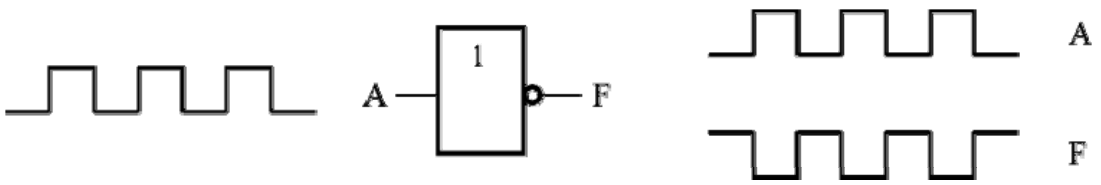
非逻辑电路

非逻辑真值表

A	$F=\bar{A}$
0	1
1	0

- “非”运算的符号

- 例：已知非门的输入波形，求其输出波形**F**。



复合逻辑门

- 基本逻辑运算的复合叫做复合逻辑运算。而实现复合逻辑运算的电路叫复合逻辑门。
  - 与非门
  - 或非门
  - 异或门
  - 同或门

与非门

- “与”运算后再进行“非”运算的复合运算称为“与非”运算。
- 实现“与非”运算的逻辑电路称为与非门。
- 与非门的逻辑关系表达式为：$F = \overline{A \cdot B}$

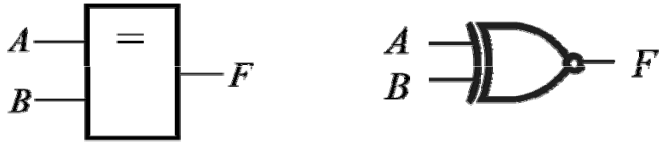
- 与非门的逻辑符号

- “或”运算后再进行“非”运算的复合运算称为“或非”运算，实现“或非”运算的逻辑电路称为或非门。
- 或非门的逻辑关系表达式为：$F=\overline{A+B}$

- 或非门的逻辑符号：

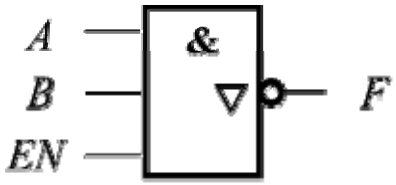
- 异或门的逻辑符号 $F = A \oplus B = \overline{A}B + A\overline{B}$

- 同或门的逻辑符号：$F = A \odot B = \overline{A \oplus B} = \overline{A}\,\overline{B} + AB$

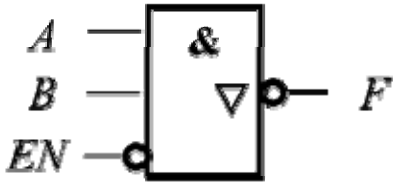


其它逻辑门：三态门

- 三态门（简称**TS**门，有倒三角符号）有三种逻辑状态，即**0**、**1**、**Z**。第三种状态为高阻状态**(Z)**，或禁止状态。
- 在普通门的基础上增加一个使能端（**EN**），使门原来的输出增加了一个Z态。
  - 当**EN**有效时，门按原逻辑工作，输出**0**或**1**；
  - 当**EN**无效时，门输出高阻态（**Z**）。
- 例：三态门的例子



EN =1 ： 使能
EN = 0 ： 失能，输出Z态



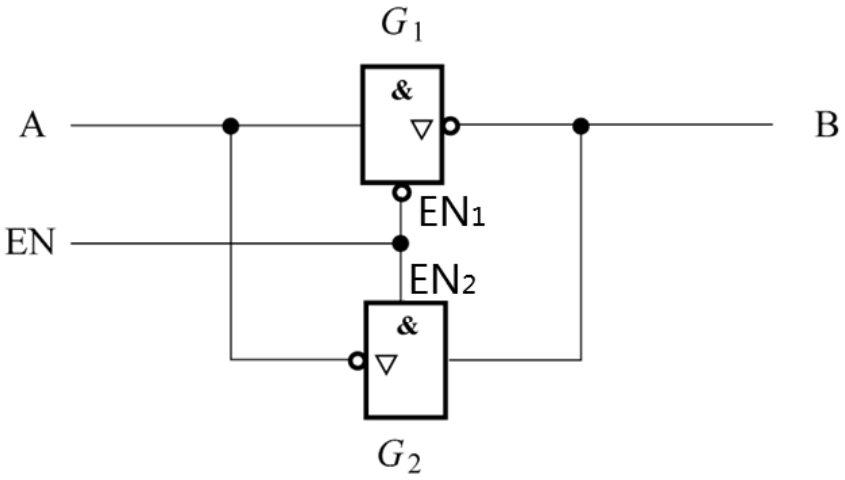
EN =0 ： 使能
EN = 1 ： 失能，输出Z态

三态门的典型应用

- 数据传输方向控制
- 总线存取控制
- 模拟开关

数据传输方向控制

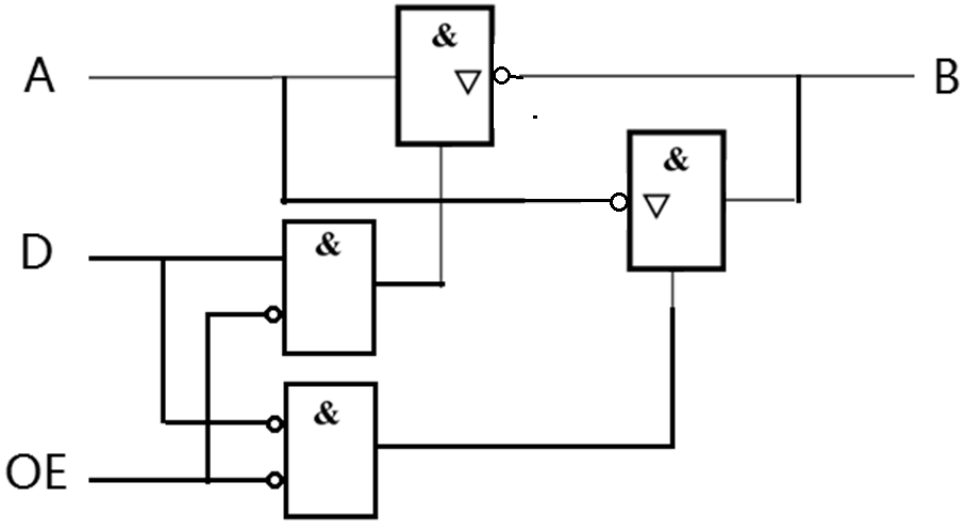
- **A和B之间进行数据传输……, EN控制传输方向**



- 思考：
  - **EN=1**时，数据能从 B 端传到 A 端；
  - **EN=0**时，数据能从 A 端传到 B 端；

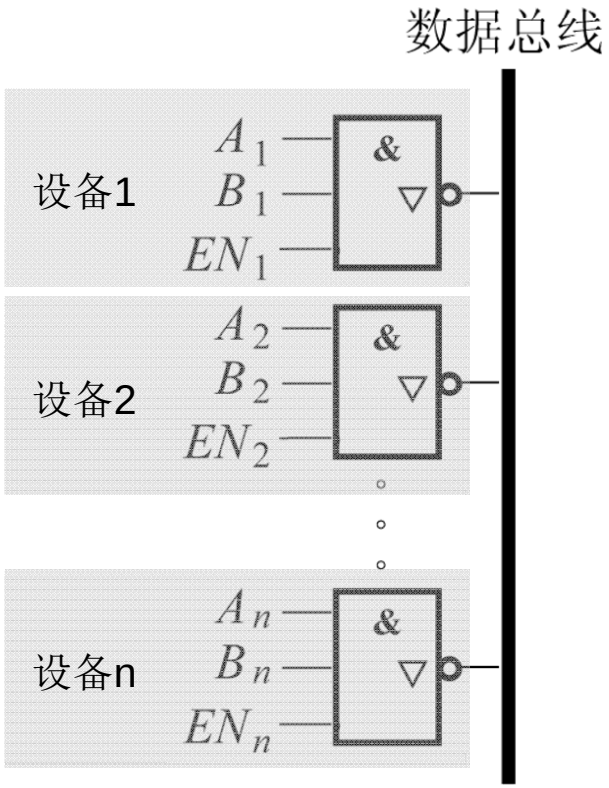
数据传输方向控制

- **A和B之间进行数据传输…，D控制方向,OE是使能端。**



- 思考：
  - **OE =** 1 **时, A和B之间高阻**
  - **OE =** 0 **时， A和B之间连通**
    - **D = 0** 时，数据能从 B 端传到 A 端；
    - **D = 1** 时，数据能从 A 端传到 B 端；

总线存取控制



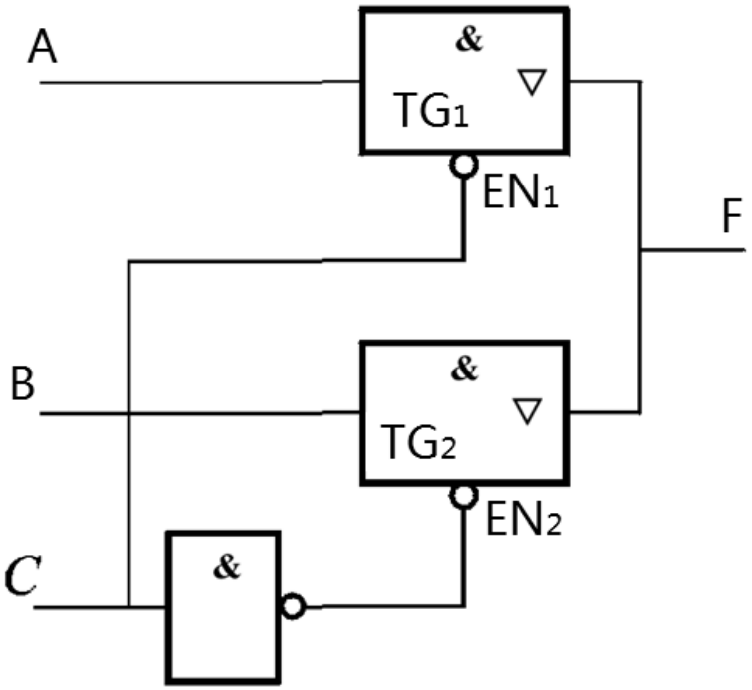
多个设备物理上连在数据总线上。必须控制存取权限，只能让最多1个设备逻辑连接（占用）总线

ENi = 0： 设备i脱离总线（Z态）

ENi = 1： 设备i逻辑连上总线

模拟开关

- 实现单刀双掷开关
- 思考
  - C=0，F = A还是=B?
  - C=1，F = A还是=B?
- 答案
  - C = 0
    - TG_1通，$F = A$；
  - C = 1
    - TG_2通，F = B。

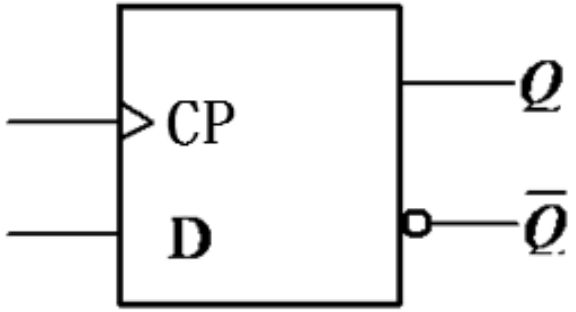


触发器——D触发器

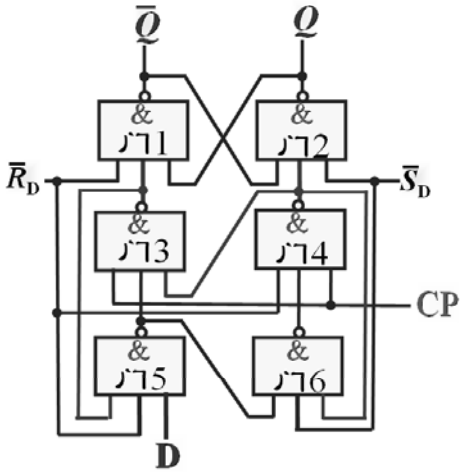
- **触发器的功能和特点**
  - 能储存一位二进制信息的单元电路。
  - 用于信号保持
  - 用做导通开关
  - 特点：0-1双稳态电路
- **触发器与门的联系**
  - 联系：触发器是在门电路的基础上引入反馈构成的。
  - 区别：门是组合电路，触发器是时序电路。
- **触发器的种类**
  - 基本RS触发器、同步RS触发器、主从型JK触发器、**维持阻塞型D触发器**、T和T'触发器等。

结构和原理

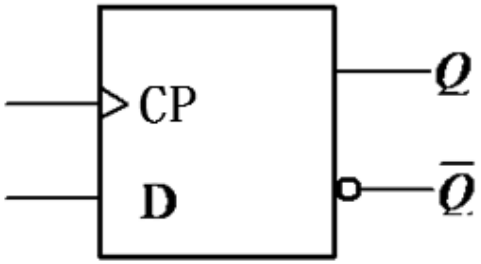
- 外部引脚
  - D：信号输入端
  - CP：时钟输入端
  - Q, $\overline{Q}$：输出端（反相）

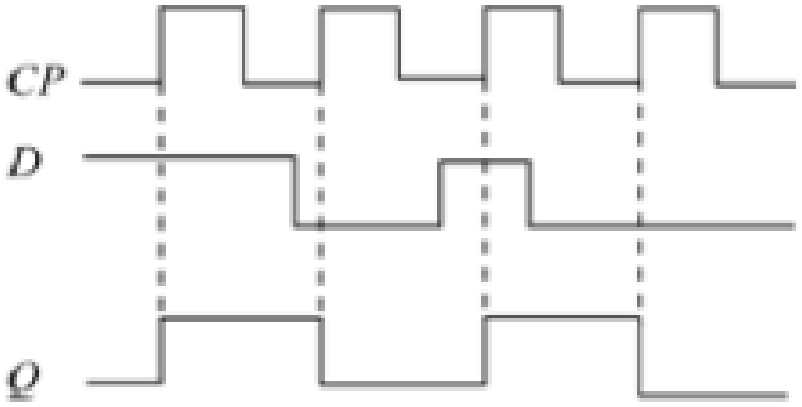


- 内部结构

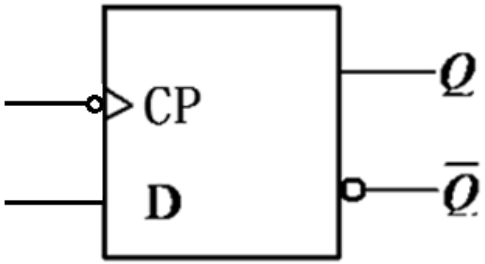


- 工作方式
  - CP上升沿锁存D (阻塞D)，上升沿后Q端保持不变。
  - 触发器在CP脉冲的上升沿产生状态变化:Q=D。而在上升沿后，D端信号变化对触发器输出状态没有影响。触发器的次态取决于CP脉冲上升沿时的D信号，

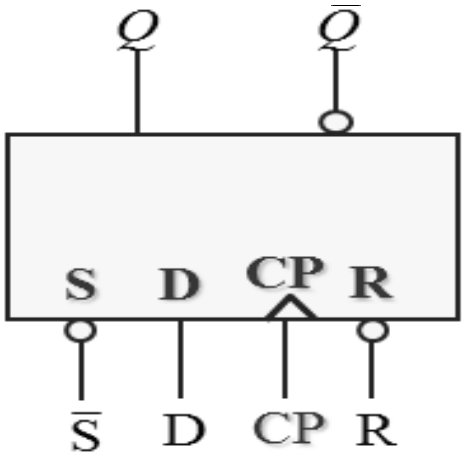




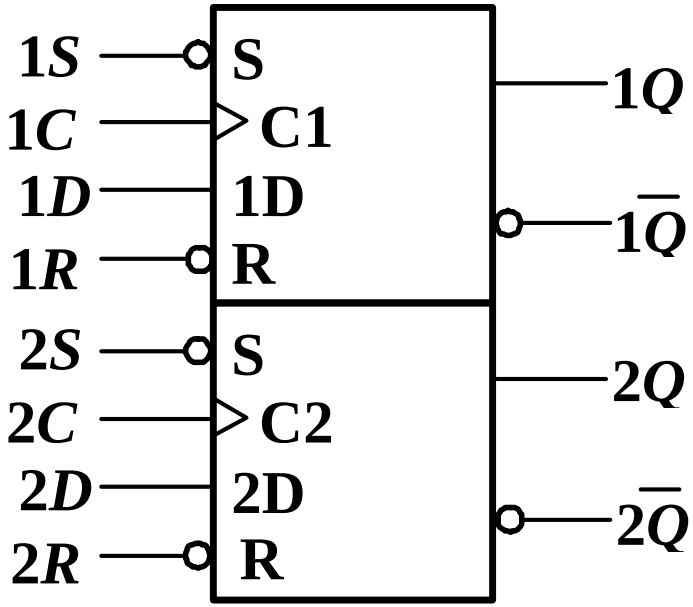
- **CP下降沿触发的触发器**



- 带清零和置1端的D触发器
  - R：置0端
  - S：置1端

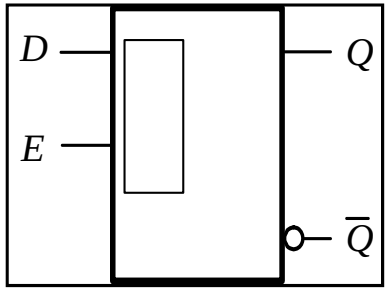


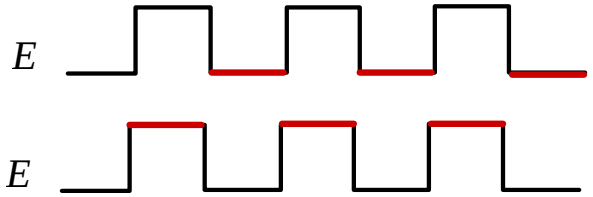
- D触发器集成电路
  - 74HC74
  - 2D触发器



锁存器——D锁存器

- 和触发器有类似的功能
  - 具有0和1两个稳状，能自行保持。
  - 能存储一位二进制码。
- 区别
  - 锁存器对**电平敏感**，触发器对**边沿敏感**





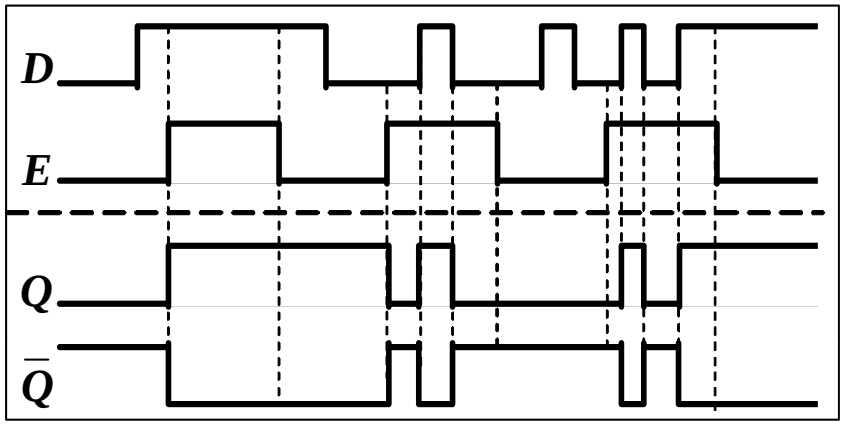
- *E*=1时 *Q* = *D*
- *E*=0时 *Q*不变

D锁存器的功能表

E	D	Q	$\overline{Q}$	功能
0	×	不变	不变	保持
1	0	0	1	置0
1	1	1	0	置1

E=1时 $Q = D$

E=0时 Q 不变



IC的基本概念

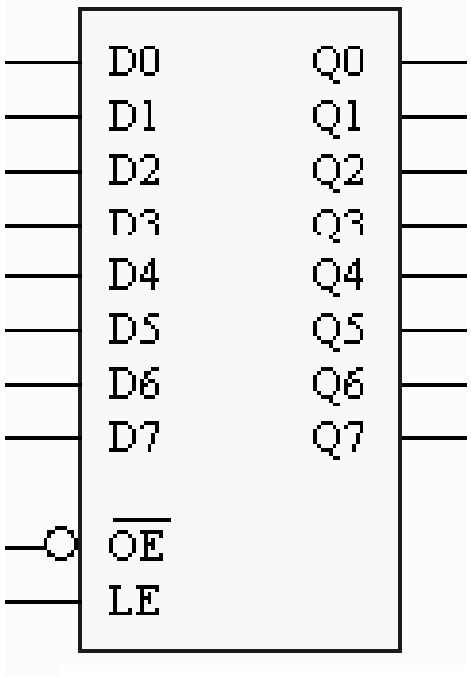
- **IC**
  - 把若干个有源器件或无源器件及其连线，按照一定的功能要求，制作在一块半导体基片上，这样的产品叫集成电路,即**IC**
  - 最简单的数字集成电路是集成逻辑门。
    - 常用集成门电路(TTL系列）

常用集成门电路(TTL系列）

序号	型号	名称	主要功能
1	74LS00	四2输入与非门	
2	74LS02	四2输入或非门	
3	74LS04	六反相器	
4	74LS05	六反相器	OC门
5	74LS08	四2输入与门	
6	74LS10	三3输入与非门	
7	74LS14	六反相器	施密特触发
8	74LS20	双4输入与非门	
9	74LS21	双4输入与门	
10	74LS30	8输入与非门	
11	74LS32	四2输入或门	
12	74LS64	4-2-3-2输入与或非门	
13	74LS86	四2输入异或门	
14	74LS125	四总线缓冲器	三态输出

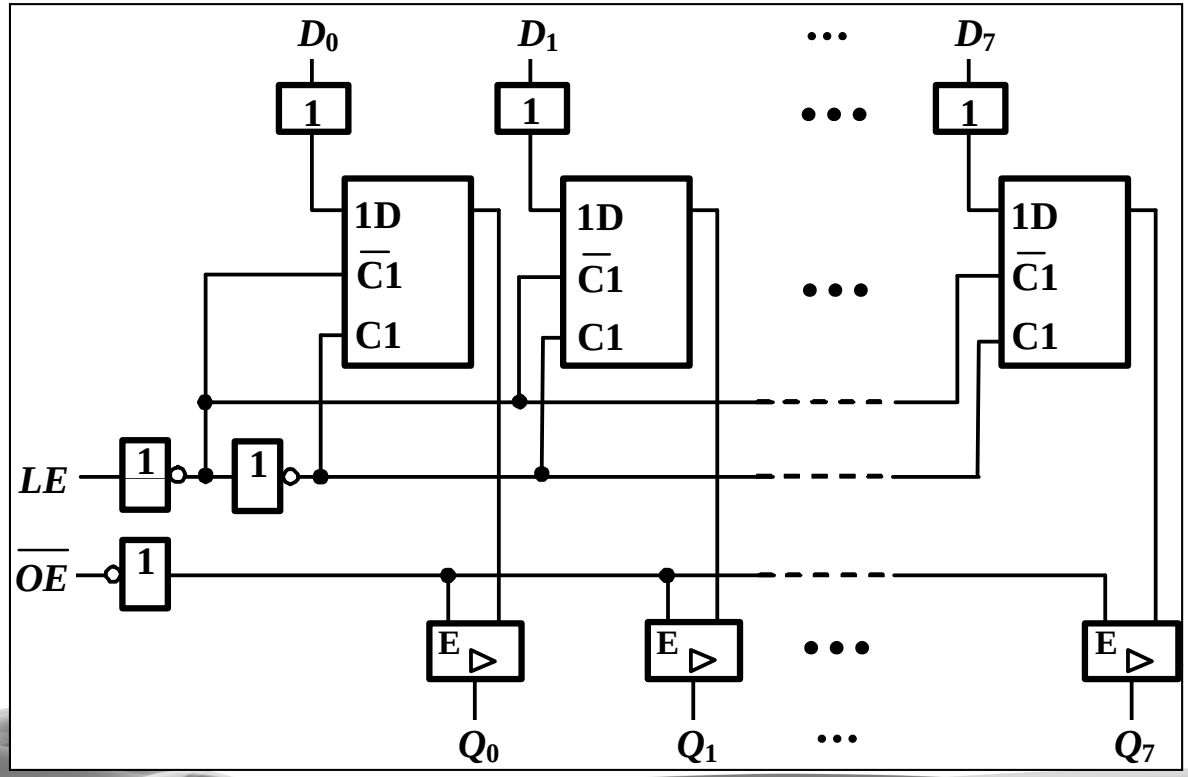
锁存器74HC373：8D三态锁存器

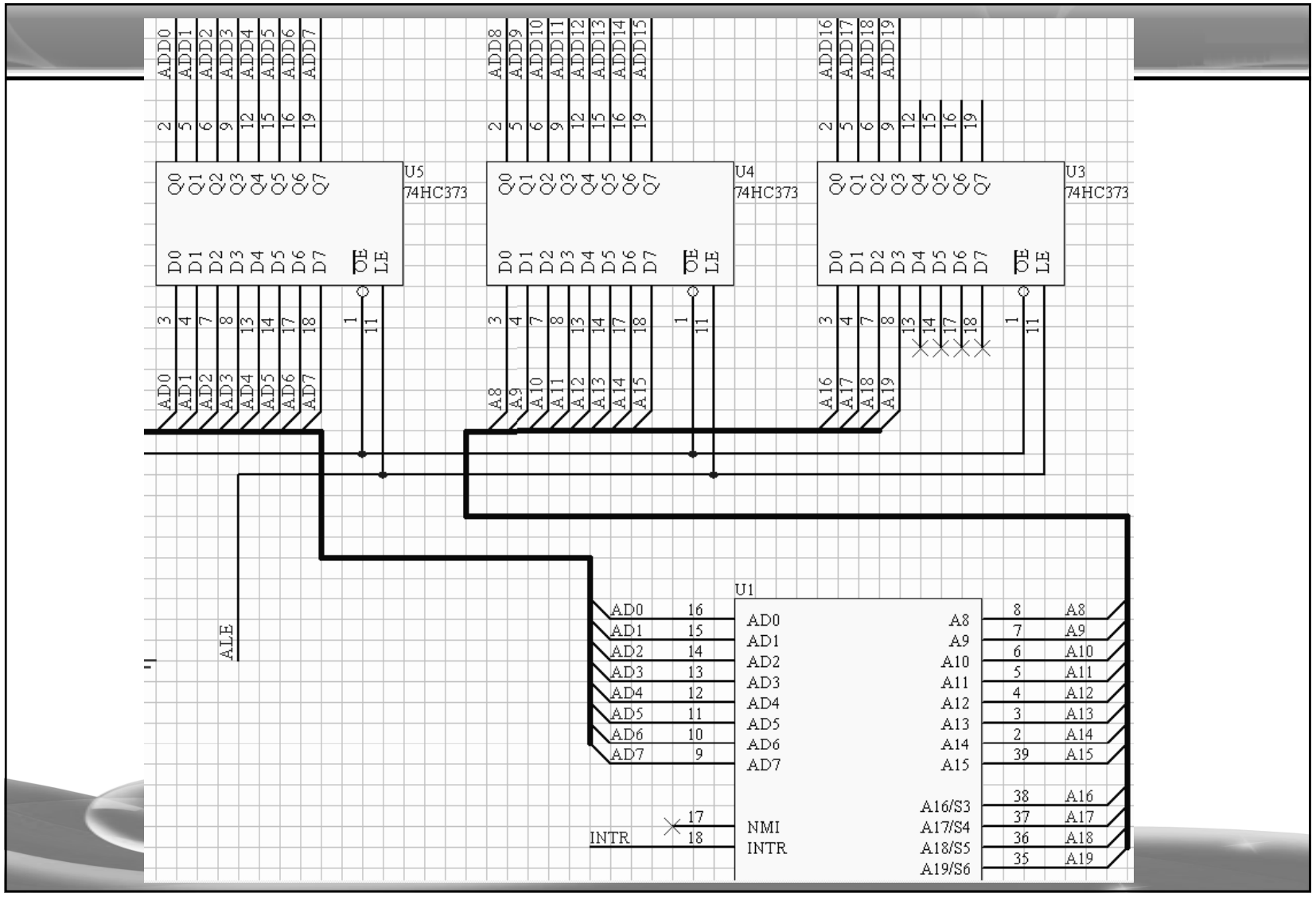
- ***LE*（*Lock Enable)*** *LE=1*：Q = D；*LE=0*：Q不变
- $\overline{\mathbf{OE}}$（三态输出使能）：**0**输出使能，**1**输出失能



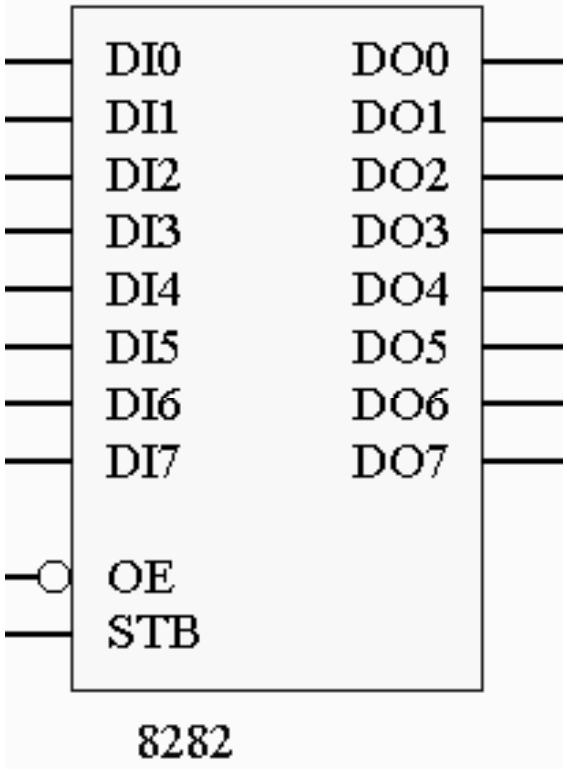
锁存器74HC373：8D三态锁存器

- ***LE: Lock Enable***
- **OE:** 三态输出使能




U5
74HC373
U4
74HC373
U3
74HC373
U1
ALE
INTR
NMI
INTR
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
A8
A9
A10
A11
A12
A13
A14
A15
A16/S3
A17/S4
A18/S5
A19/S6
A16
A17
A18
A19

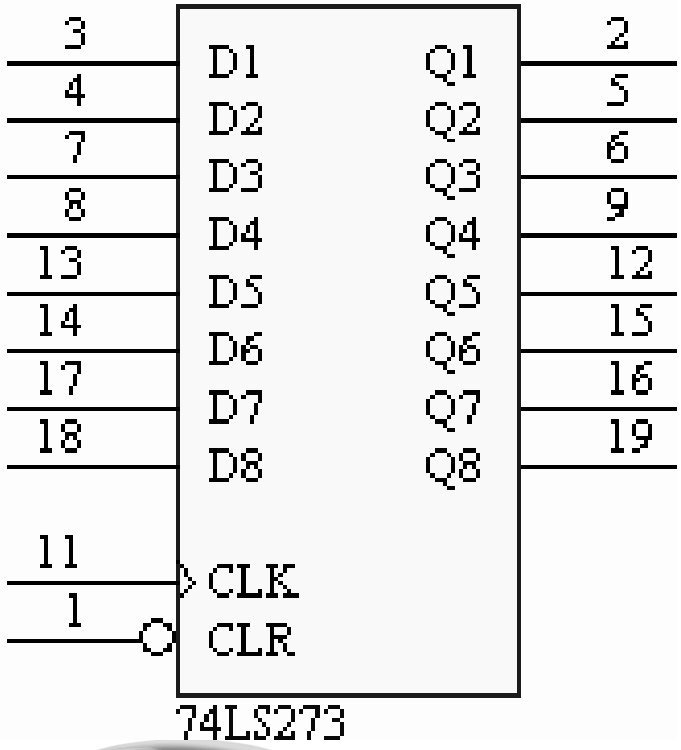
Intel 8282：三态锁存器



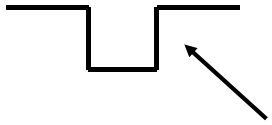
DI：输入；
DO：输出
STB：选通控制(等同373的LE)
OE ：输出使能

异步清零的锁存器：74LS273

D：输入 ； Q：输出　 CLK：控制　CLR: Clear

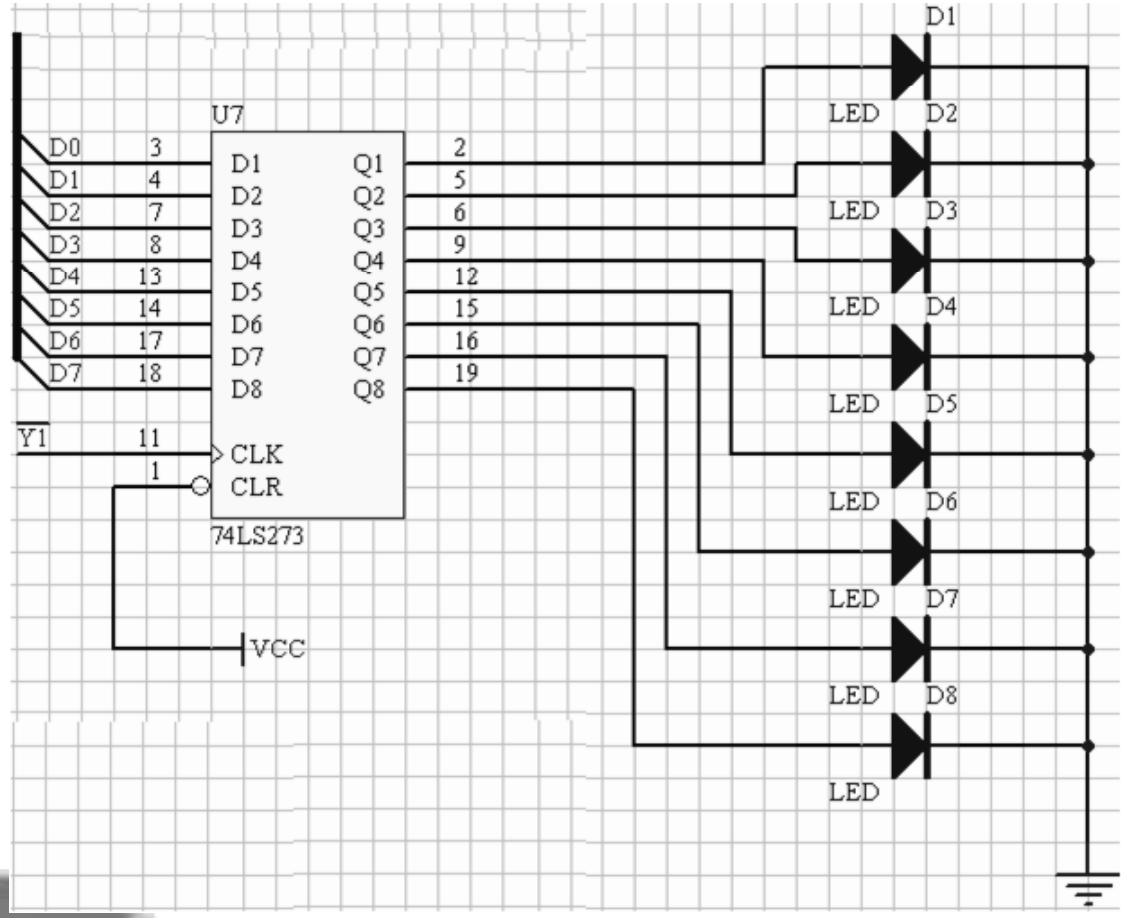


具有异步清零的
上升沿锁存器

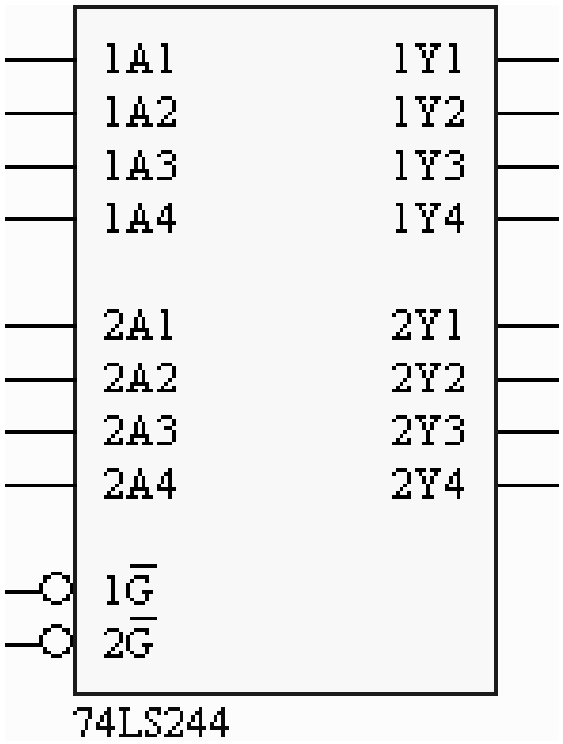


负脉冲的上升沿

74LS273


U7
D0
D1
D2
D3
D4
D5
D6
D7
3
4
7
8
13
14
17
18
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
2
5
6
9
12
15
16
19
Y1
11
1
CLK
CLR
74LS273
VCC
LED
D1
D2
D3
D4
D5
D6
D7
D8

74LS244 ——缓冲器(实质是三态开关)



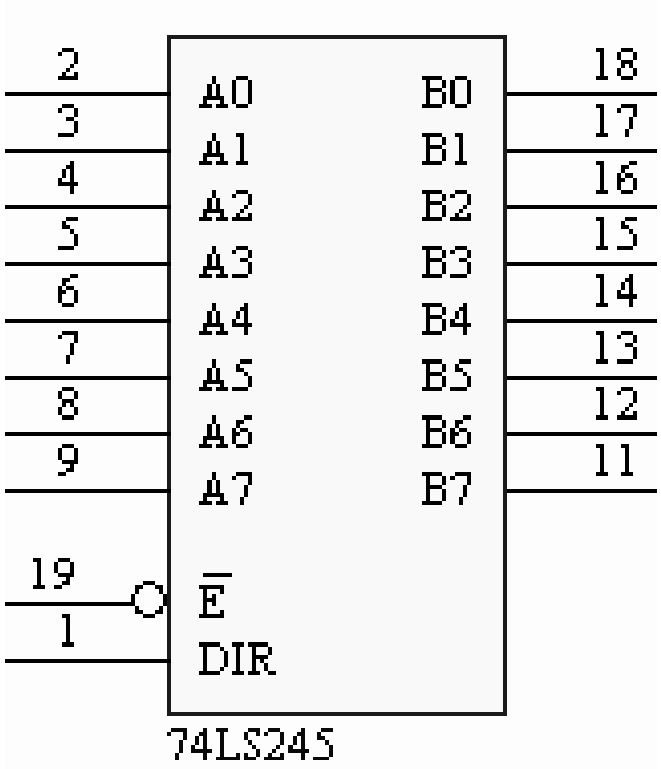
1$\overline{G}$ = 0: 1Y = 1A
1$\overline{G}$ = 1: 1Y = Z

2$\overline{G}$ = 0: 2Y = 2A
2$\overline{G}$ = 1: 2Y = Z

特点：

8位单向缓冲器
双**4**位分两组
输出与输入同相

74LS245 ——缓冲器(实质是三态开关)



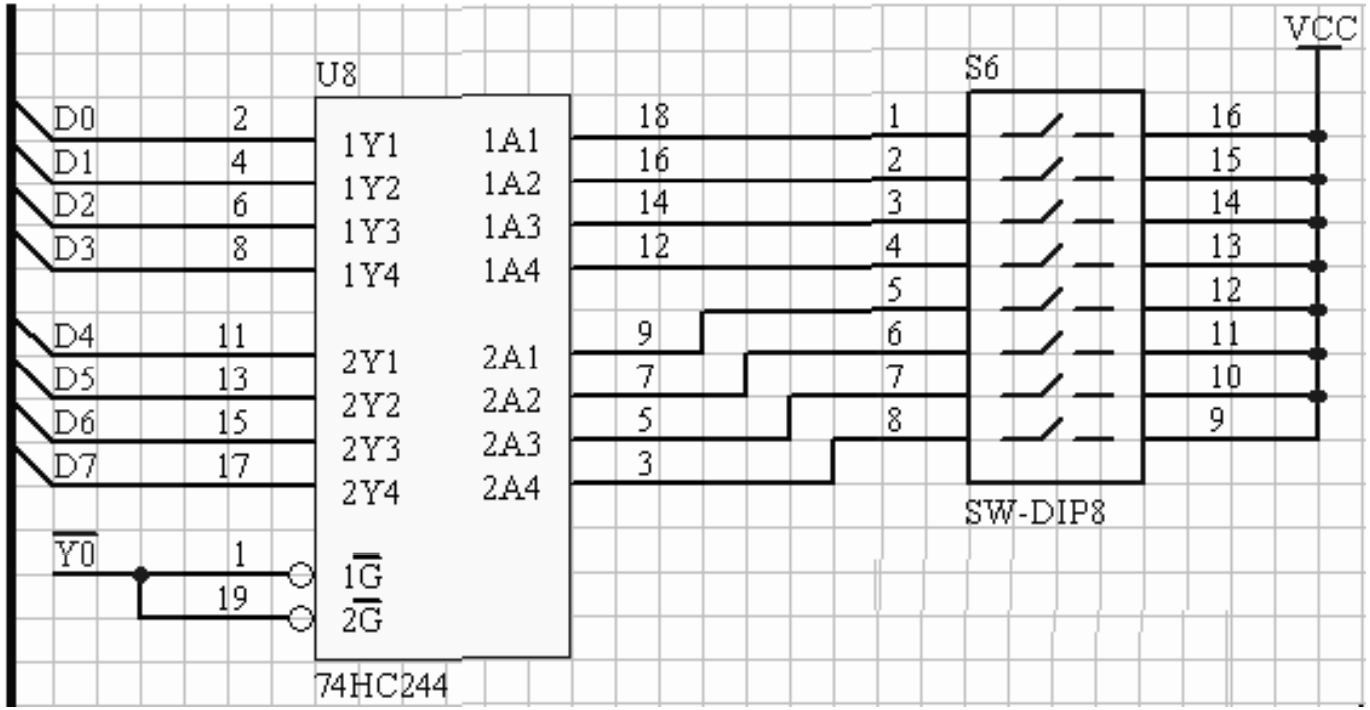
8位双向缓冲器
控制端**E**低电平有效
输出与输入同相

E：低电平导通
DIR：决定传输方向

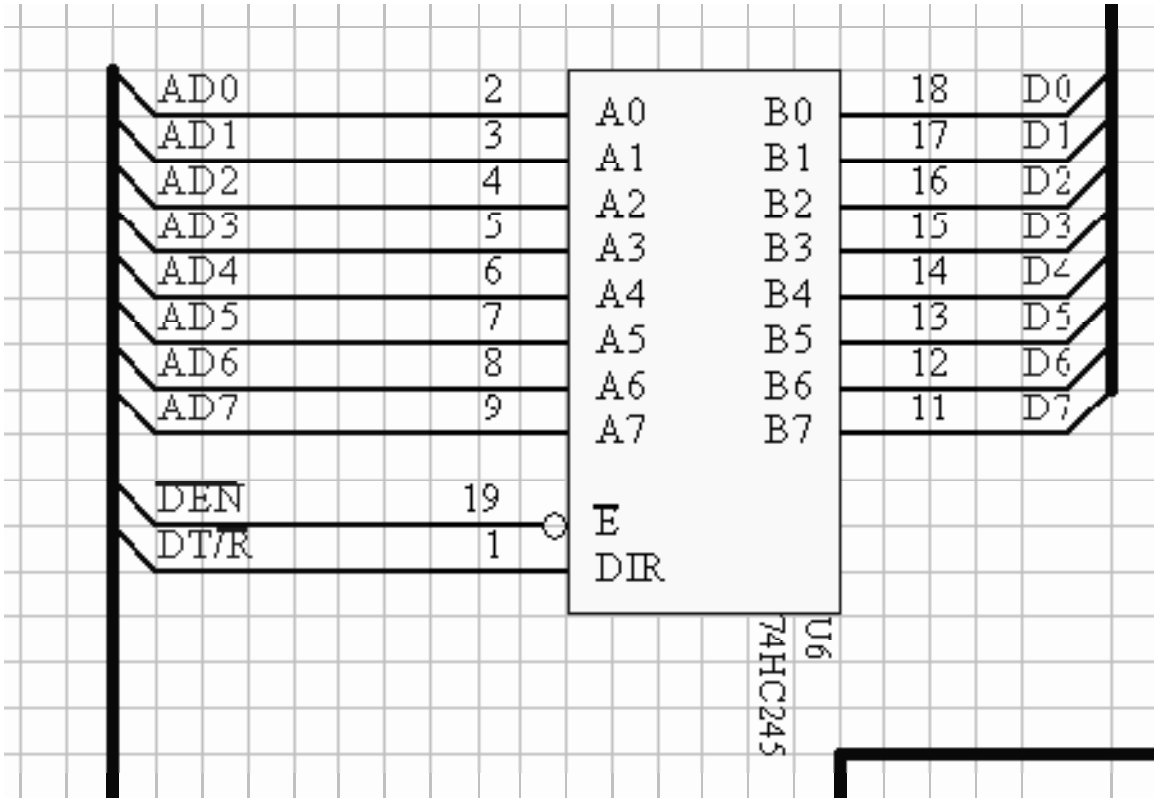
DIR=1：**A → B**

DIR=0：**B → A**

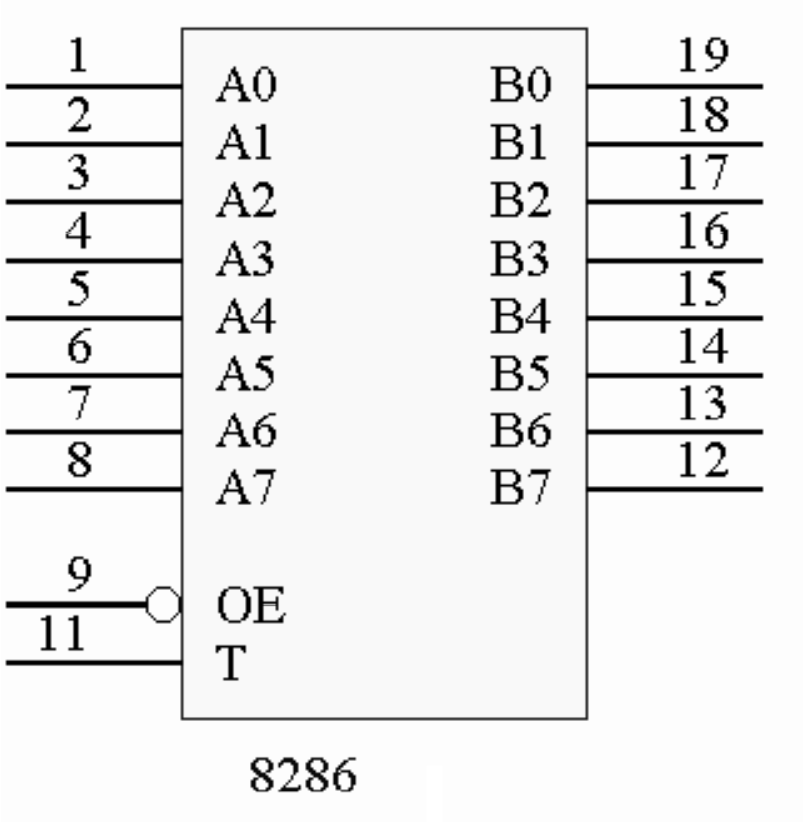
74LS244


U8
D0
D1
D2
D3
D4
D5
D6
D7
2
4
6
8
11
13
15
17
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
1A1
1A2
1A3
1A4
2A1
2A2
2A3
2A4
18
16
14
12
9
7
5
3
S6
1
2
3
4
5
6
7
8
16
15
14
13
12
11
10
9
VCC
SW-DIP8
Y0
1
19
1G
2G
74HC244

74LS245


AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
DEN
DT/R
2
3
4
5
6
7
8
9
19
1
A0
A1
A2
A3
A4
A5
A6
A7
E
DIR
B0
B1
B2
B3
B4
B5
B6
B7
18
17
16
15
14
13
12
11
D0
D1
D2
D3
D4
D5
D6
D7
U6
74HC245

缓冲器—— Intel 8286 (实质是三态开关)



8位双向缓冲器
控制端**OE**低电平有效
输出与输入同相

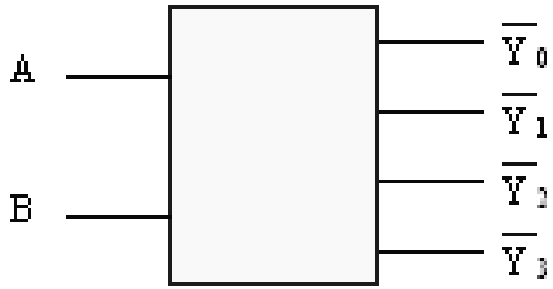
E：低电平导通
T：决定传输方向

DIR = 1：**A → B**
DIR = 0：**B → A**

数据收发器

基本芯片--译码器

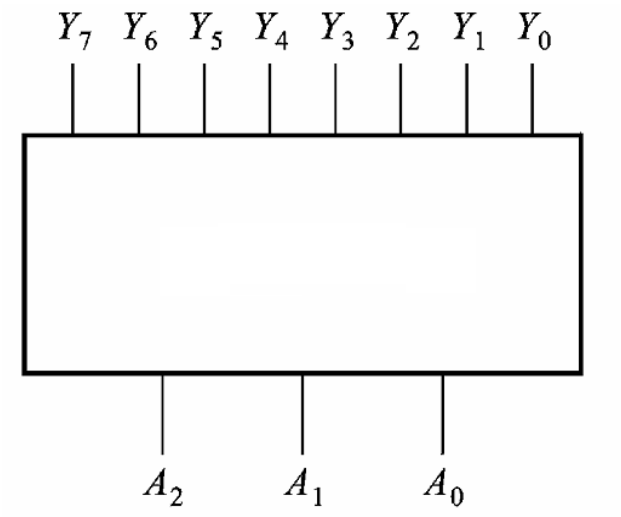
- 译码器
  - 功能：将某个二进制数据的含义“翻译”出来，指示唯一的某1个事件有效。
  - 结构：n位输入脚，2^n个输出脚（每脚对应1个事件）



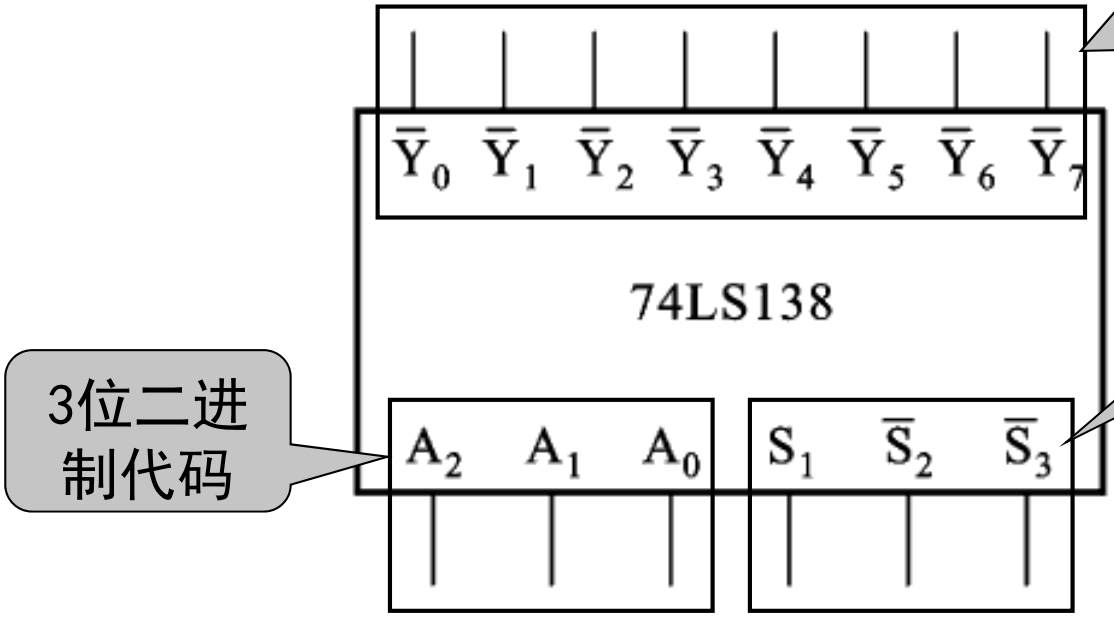
  - 特点：每次译码，仅唯一的1个输出引脚为有效电平
    - 输入：AB = [00, 01, 10, 11]
    - 输出：Y0Y1Y2Y3= 0111，1011，1101，1110

3-8译码器

- 结构：3个输入引脚，8个（ $=2^3$ ）输出引脚
- 功能：输入3位二进制代码$A_2A_1A_0$（000~111），在唯一的某1个输出引脚上出现有效电平。



74LS138 ——3-8译码器



74LS138的逻辑符号

$$S = S_1 \cdot \overline{S_2} \cdot \overline{S_3}$$

- **S=1** 工作
- **S=0** 禁止（ 输出全**1**）

74LS138的功能表

输入				输出							
S_1	$\overline{S}_2+\overline{S}_3$	A_2	A_1	A_0	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
×	1	×	×	×	1	1	1	1	1	1	1	1
0	×	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

禁止译码

译码工作

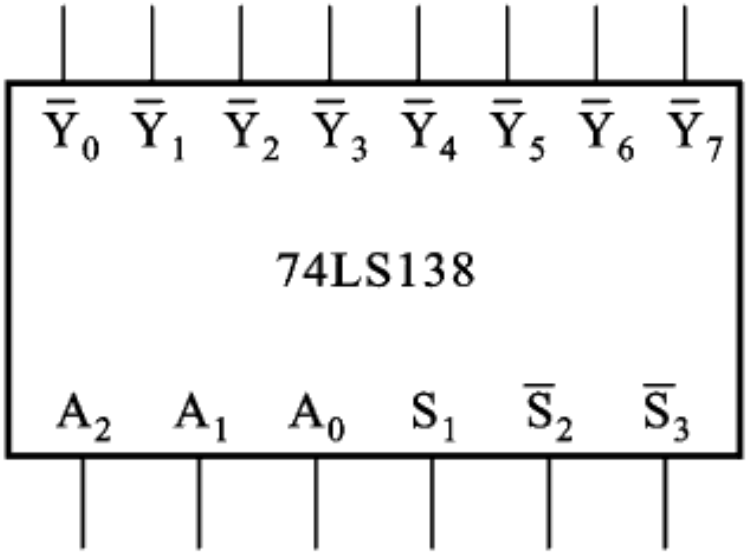
译中为0

高电平有效

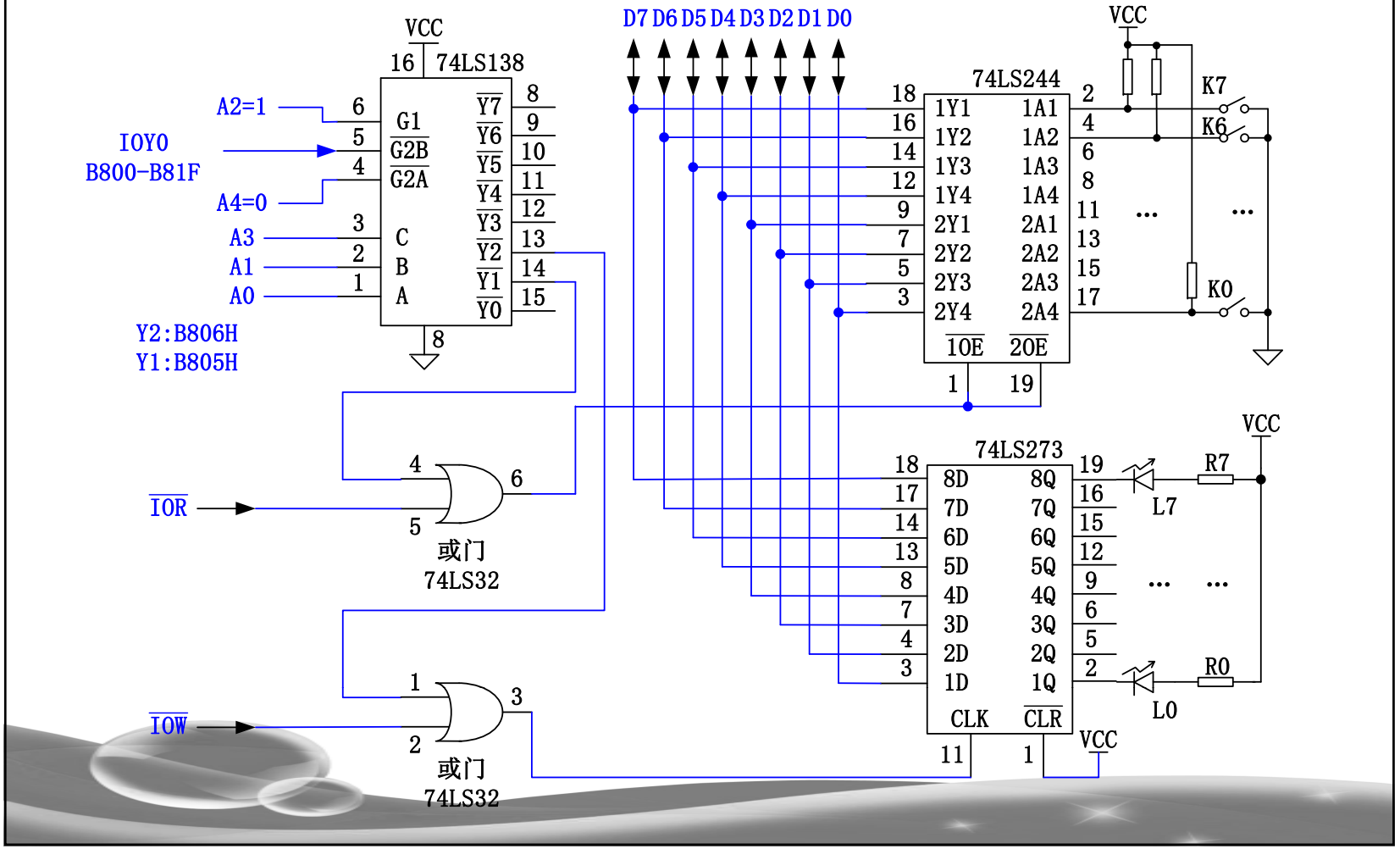
低电平有效

74LS138

- 译码工作时
- **A2 A1 A0**和输出
  - **000 →Y0**低电平
  - **001 →Y1**低电平
  - **010 →Y2**低电平
  - **011 →Y3**低电平
  - **100 →Y4**低电平
  - **101 →Y5**低电平
  - **110 →Y6**低电平
  - **111 →Y7**低电平



门和IC在CPU典型接口电路中的应用



第4节 8088微处理的外部结构

8088的两种工作模式

- 最小模式
  - 构成小规模的应用系统——单处理器系统
  - 8088本身提供所有的系统总线信号
- 最大模式
  - 构成较大规模的应用系统——多处理器系统，例如可以接入数值协处理器8087
  - 控制信号较多，8088和总线控制器8288共同形成系统总线信号

8088的两种工作模式

- 两种模式利用MN/MX*引脚区别
  - MN/MX*接高电平为最小模式
  - MN/MX*接低电平为最大模式
  - 硬件决定工作方式
- 两种模式内部操作没有区别
  - 本书以最小模式展开基本原理
  - IBM PC/XT采用最大模式

8088的外部引脚——电气特性

- 电气特性
  - ■电源VCC
    - ◆ 5V ± 10% 的条件下能够工作；
  - ■输入特性：

    低电平　0.8 V (0)

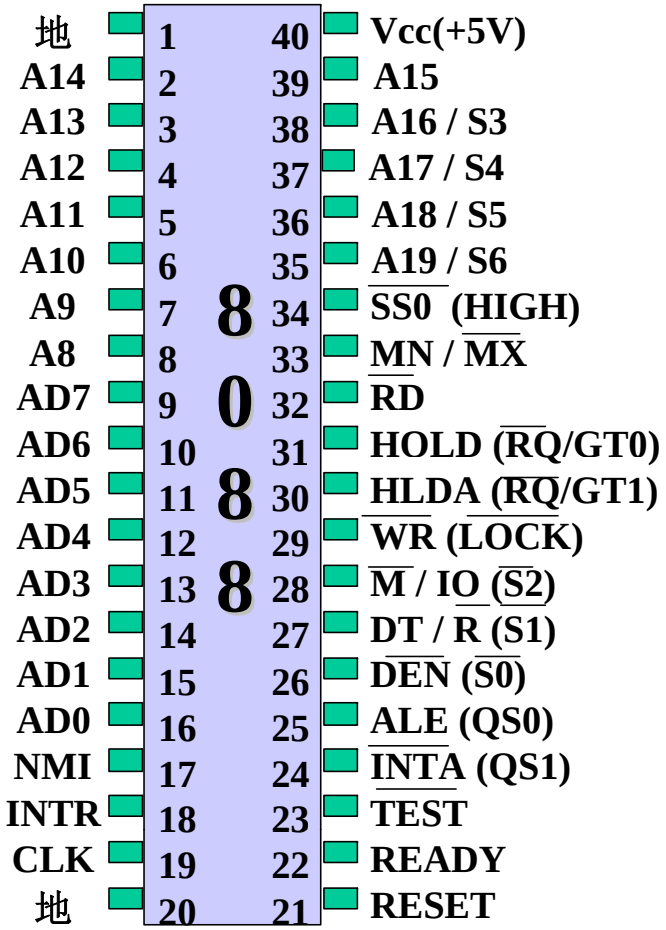
    高电平　2.0 V (1)
  - ■输出特性：

    低电平　0.4 V (0)

    高电平　2.4 V (1)

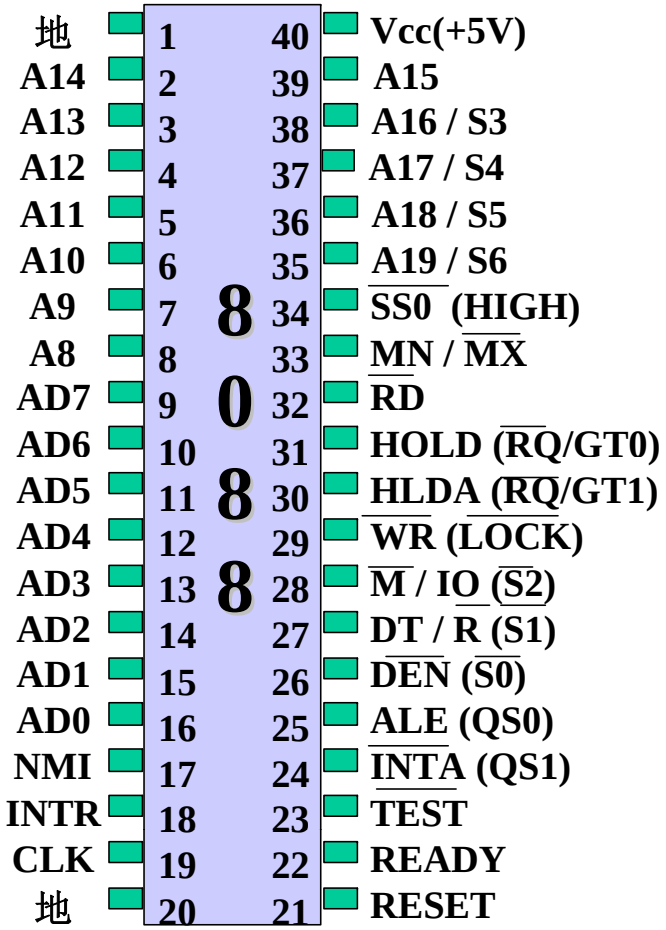
8088的外部引脚

- 外部特性
  - 引脚的功能
  - 是否复用
  - 信号的流向
  - 有效电平
  - 三态能力



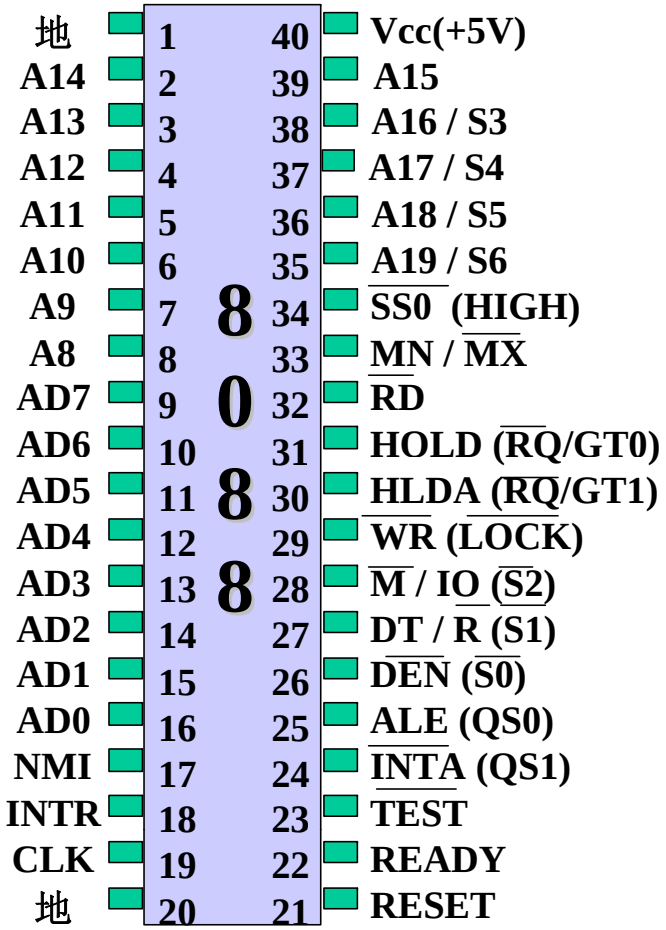
8088的外部引脚——电源，时钟，复位

- 电源管脚
  - VCC(40)、GND(1/20)
  - 5V±10%
- 时钟管脚
  - CLK(19)
  - 外接4.77MHz外部时钟
- 复位管脚
  - RESET(21)
  - 正脉冲复位(4T)
    - ◆标志清0：Flag=0000H；
    - ◆DS,SS,ES复位为0H;
    - ◆CS8088FH ,IP复位为0H；
      - FFFF0H放第一条指令。



8088的外部引脚

- 主要功能引脚
  - 读、写信号$\overline{RD}$、$\overline{WR}$
  - 端口/存储器选择信号$\overline{M}$/IO
  - 数据传送/接收信号DT/$\overline{R}$
  - 数据线AD7～AD0
  - 数据允许$\overline{DEN}$
  - 地址线A19～A0
  - 地址锁存ALE
  - TEST
  - READY
  - INTR
  - $\overline{INTA}$
  - NMI



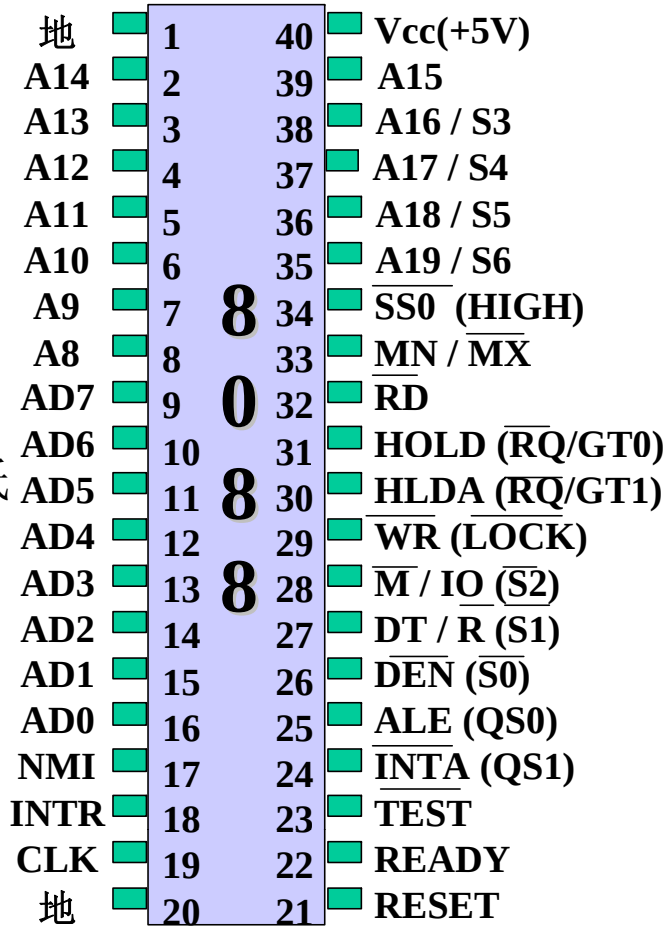
8088的外部引脚

- $\overline{RD}$(32) ——读(输出，三态)
  - 读I/O端口或存储器时，$\overline{RD}$输出低电平；否则为高阻态。

```
IN      AL，25H      ；读I/O
MOV   AX，[BX]       ；读MEM
```

- $\overline{WR}$(29) ——写(输出，三态)
  - 写I/O端口或存储器时，$\overline{WR}$输出低电平；否则为高阻态。

```
OUT  25H，AX         ；  写I/O
MOV [BX]，AX         ；写MEM
```



8088的外部引脚

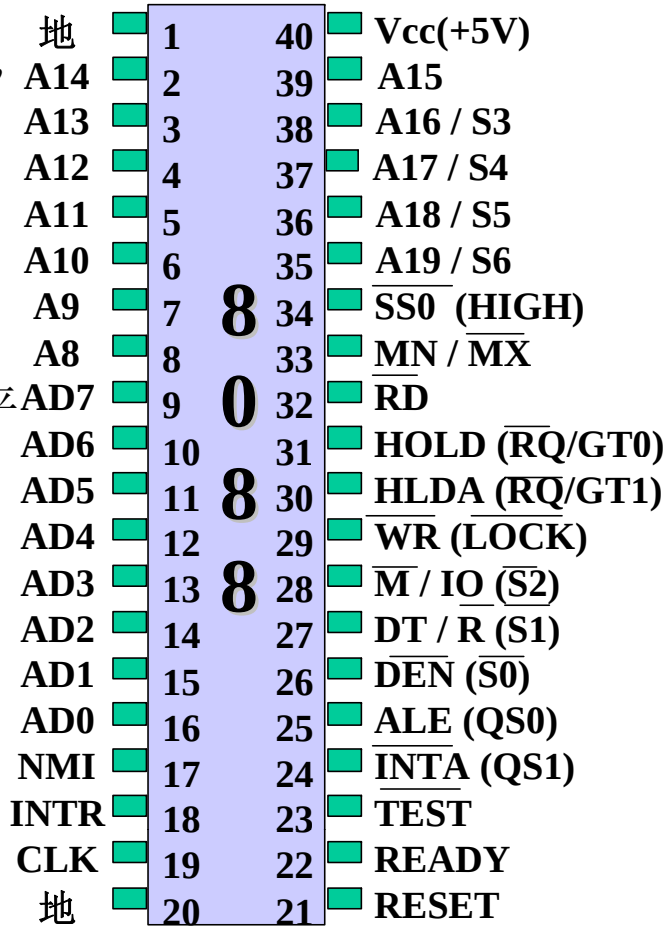
- IO/$\overline{M}$(28) ——端口/存储器选择(输出，三态)
  - 对端口读或写， IO/$\overline{M}$输出高电平

    IN　　AL，25H　；读I/O

    OUT　52H，AL　　；写I/O
  - 对存储器读或写，IO/$\overline{M}$输出低电平

    MOV　AX，[BX]　　；读MEM

    MOV　[BX]，AX　　；写MEM
  - 其它情况处于高阻状态



读写信号和IO/$\overline{M}$信号的组合

- 由IO/$\overline{M}$、$\overline{WR}$和$\overline{RD}$三个信号生成存储器读、存储器读写、I/O读、I/O写等4种信号（低电平有效）

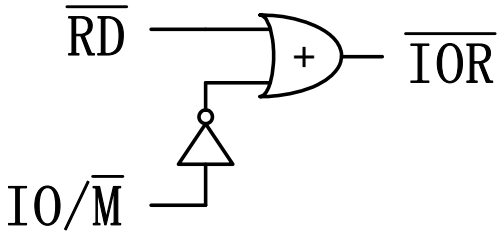
信号名称	IO/$\overline{M}$	$\overline{WR}$	$\overline{RD}$
存储器读(MEMR)	低	高	低
存储器写(MEMW)	低	低	高
I/O读(IOR)	高	高	低
I/O写(IOW)	高	低	高

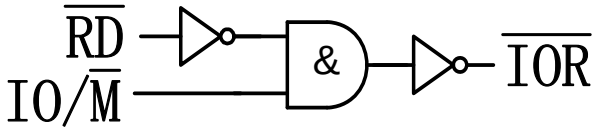
读写信号和IO/$\overline{M}$信号的组合——实现IO读信号

- 通过RD、WR、IO/M得到IO读信号(IOR)

IO/$\overline{M}$	$\overline{RD}$	$\overline{IOR}$
1	0	0
0	X	1
X	1	1

$$\overline{IOR} = \overline{IO/\overline{M}+\overline{RD}} = \overline{IO/\overline{M}\cdot\overline{\overline{RD}}}$$



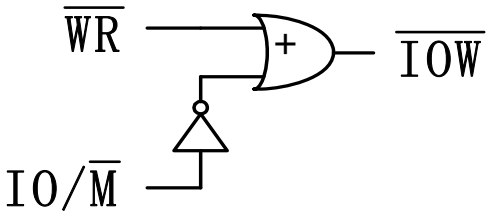


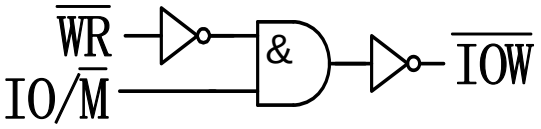
读写信号和IO/$\overline{M}$信号的组合——实现IO写信号

- 通过RD、WR、IO/M得到IO写信号(IOW)

IO/$\overline{M}$	$\overline{WR}$	$\overline{IOW}$
1	0	0
0	X	1
X	1	1

$$\overline{IOW}=\overline{IO/\overline{M}+\overline{WR}}=\overline{IO/\overline{M}\cdot\overline{WR}}$$





课堂练习——实现存储器读/写信号

- 通过RD、WR、IO/M得到存储器读/写信号(MEMR/MEMW)

$IO/\overline{M}$	$\overline{RD}$	$\overline{MEMR}$
0	0	0
1	X	1
X	1	1

$IO/\overline{M}$	$\overline{WR}$	$\overline{MEMW}$
0	0	0
1	X	1
X	1	1

$\overline{MEMR}=IO/\overline{M}+\overline{RD}=\overline{\overline{IO/\overline{M}}\cdot\overline{RD}}$

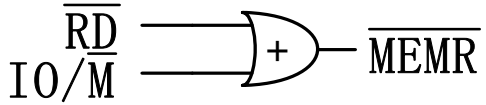
$\overline{MEMW}=IO/\overline{M}+\overline{WR}=\overline{\overline{IO/\overline{M}}\cdot\overline{WR}}$

- 画出它们各自或和与两种形式的电路图

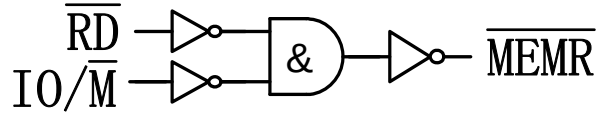
读写信号和IO/$\overline{M}$信号的组合——实现存储器读信号

- 通过RD、WR、IO/M得到存储器读信号

IO/$\overline{M}$	$\overline{RD}$	$\overline{MEMR}$
0	0	0
1	X	1
X	1	1



$$\overline{MEMR}=IO/\overline{M}+\overline{RD}=\overline{IO/\overline{M}\cdot\overline{RD}}$$

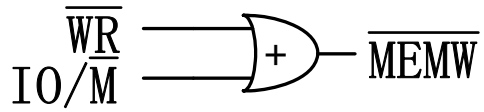


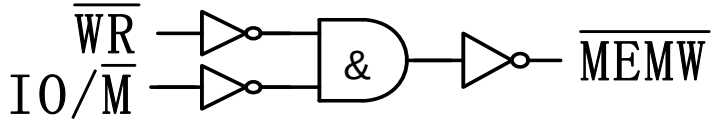
读写信号和IO/$\overline{M}$信号的组合——实现存储器写信号

- 通过RD、WR、IO/M得到信号存储器写信号

IO/$\overline{M}$	$\overline{WR}$	$\overline{MEMW}$
0	0	0
1	X	1
X	1	1

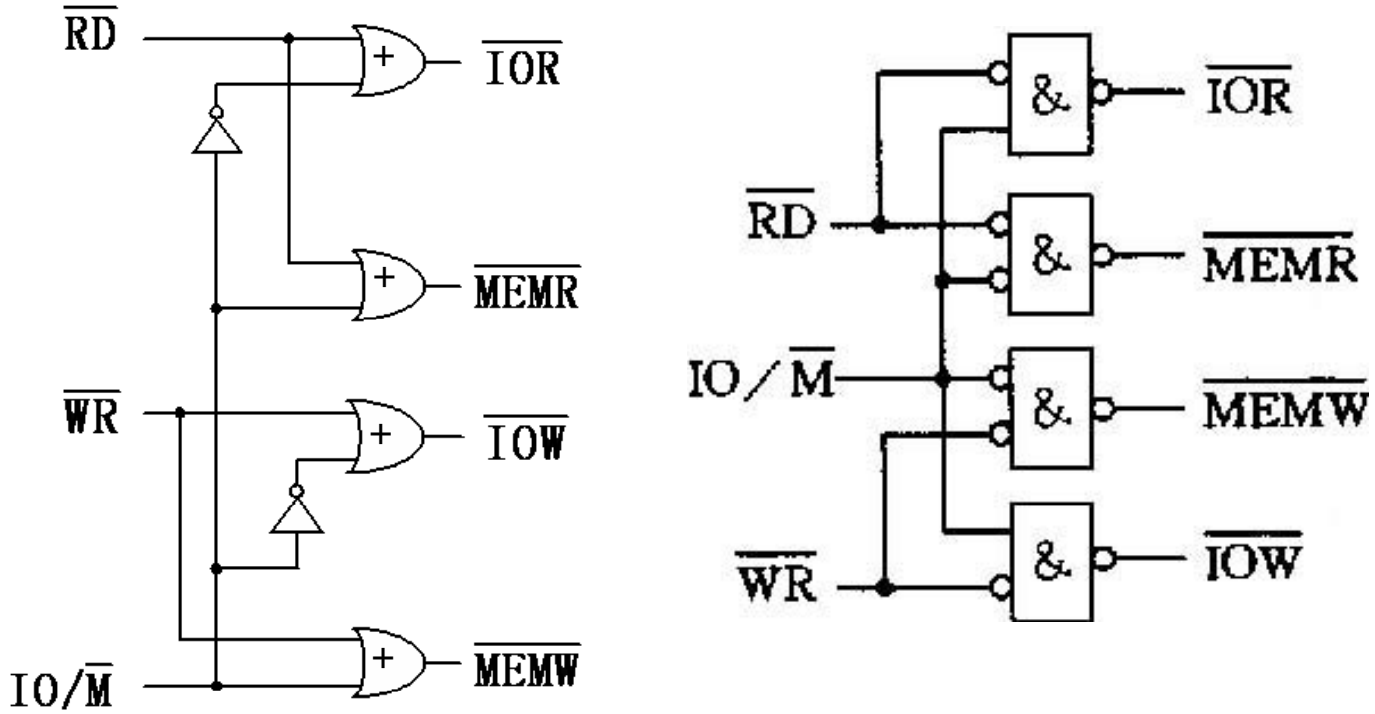
$$\overline{MEMW}=IO/\overline{M}+\overline{WR}=\overline{\overline{IO/\overline{M}}\cdot\overline{\overline{WR}}}$$





读写信号和IO/$\overline{M}$信号的组合——综合实现存储器和IO读写信号

- 通过单一电路同时实现存储器和I/O的读写4个信号



8088的外部引脚

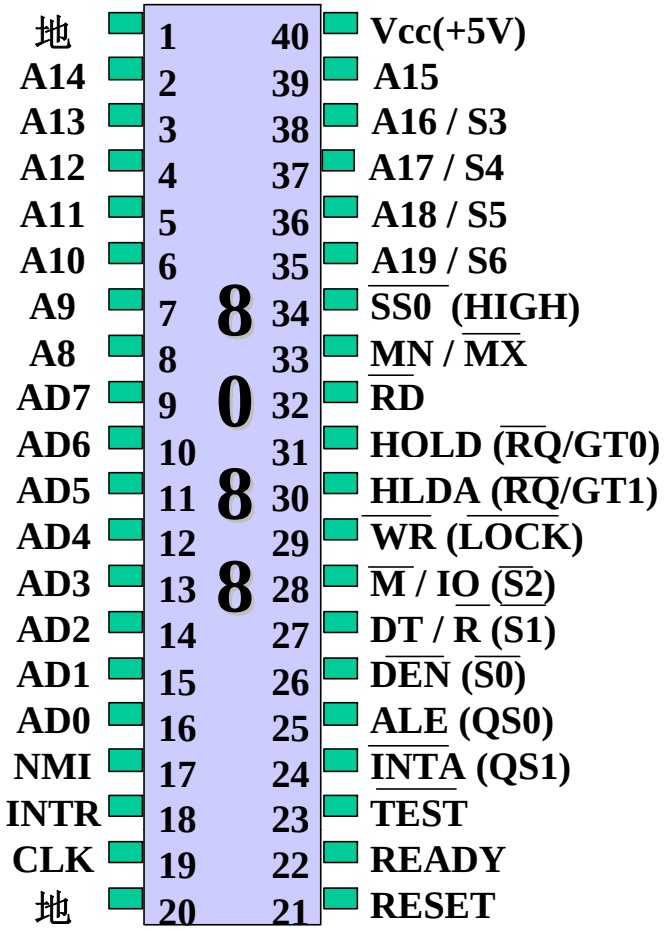
- DT/$\overline{R}$(27)
  - 数据传送/接收信号 (出，三态)
  - 向I/O或存储器传送数据，输出高

    OUT 25H， AL ；写I/O

    MOV [BX]，AX ；写MEM
  - 从I/O或存储器接收数据，输出低

    IN AL，25H； 读I/O

    MOV AX，[BX]； 读MEM
  - 其它情况下处于高阻状态

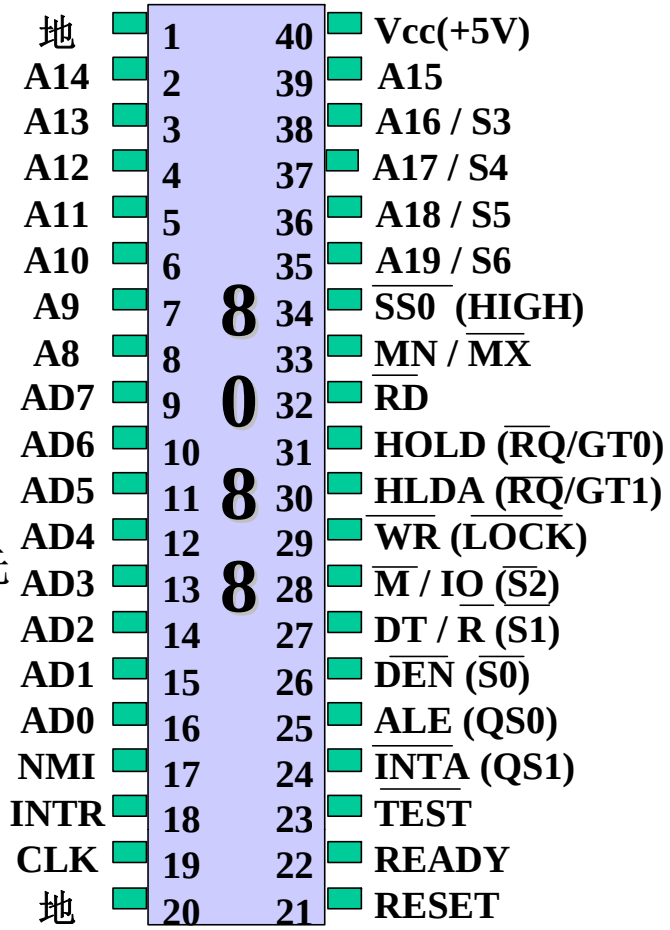


8088的外部引脚

- D7～D0(9～16)数据线(双，三态)
  - 读写I/O或存储器时出现相应数据

```
IN    AL，25H          ；读I/O
OUT 25H，AL            ；写I/O
MOV  BX，0000H
MOV  AL，[BX]          ；读MEM
MOV  [BX]， 32H        ；写MEM
```

  - 注意：D7～D0和A7～A0复用，先传送地址，后传送数据。
- $\overline{DEN}$(26)数据允许 (出，三态)
  - 低电平：$AD_{7\sim0}$传送的是数据。



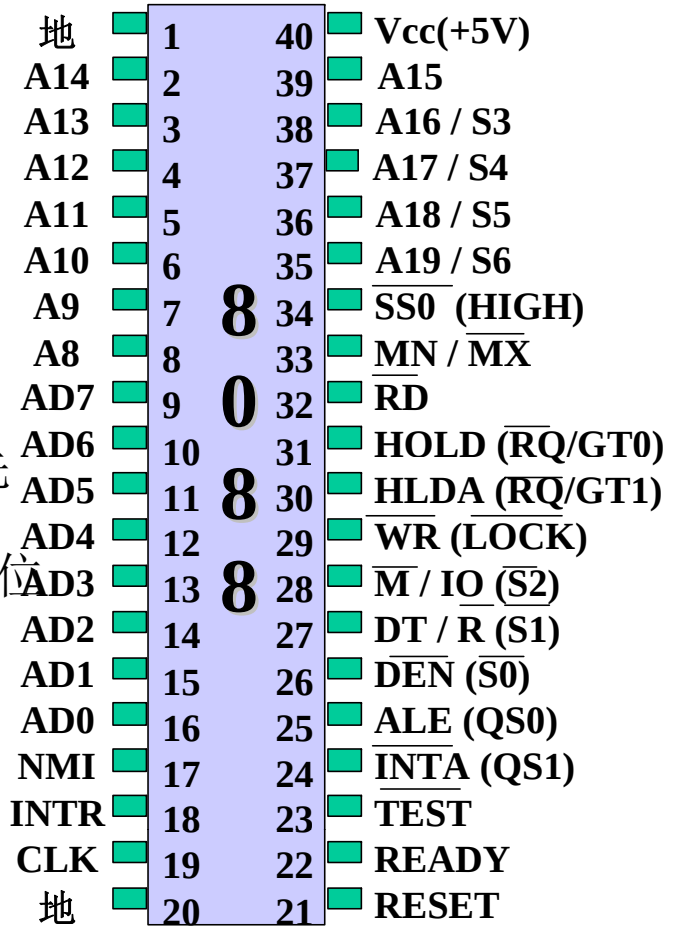
8088的外部引脚

- A19～A0地址线(出，三态)
  - 读写I/O或存储器，出现地址信息

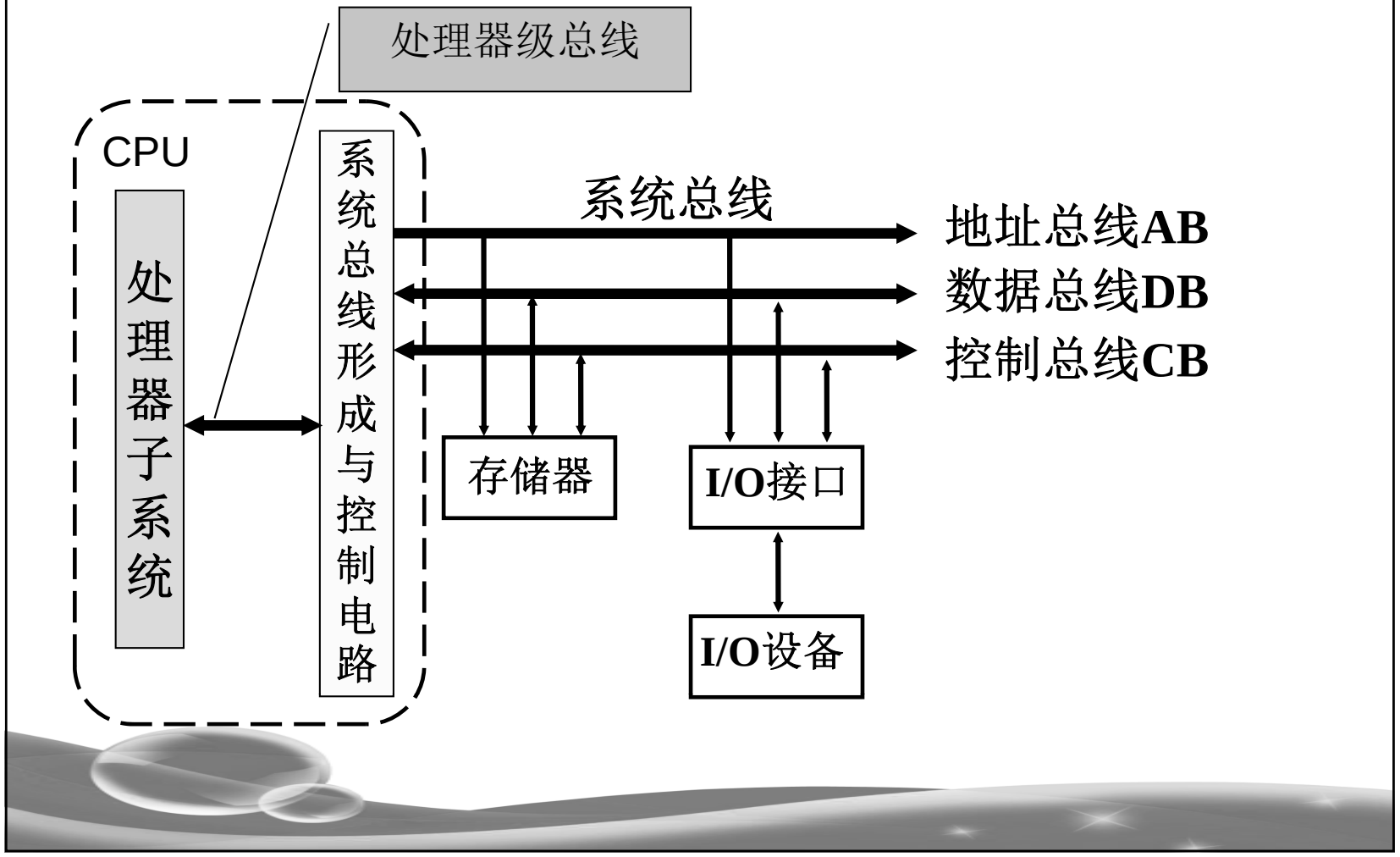
```
IN    AL，25H      ；读I/O
OUT   25H，AL      ；写I/O
MOV   BX，0000H
MOV   AL，[BX]  ；读MEM
MOV  [BX]，32H  ；写MEM
```

  - 注意：D7～D0和A7～A0复用，先传送地址，后传送数据。
    - ◆逻辑地址由地址加法器转为20位物理地址。

- ALE(25)地址锁存信号(出，三态)
  - 当$A_{19\sim0}$出现地址，产生正脉冲。
  - 利用该脉冲锁存地址



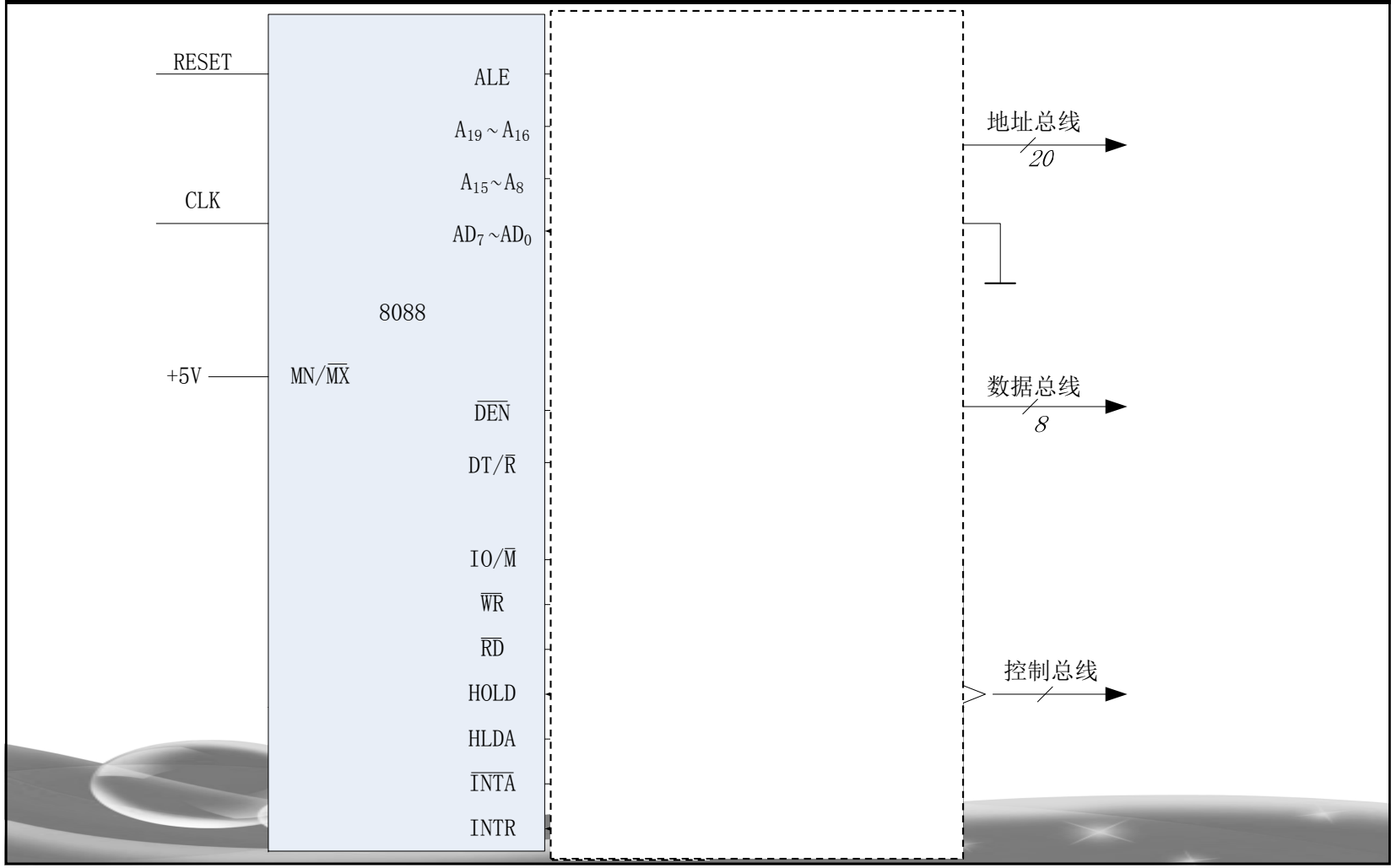
8088系统总线的形成



8088系统总线的形成（最小系统）

- 主要解决
  - 实现地址总线，数据总线和控制总线
    - 地址与数据的分离
    - 地址锁存

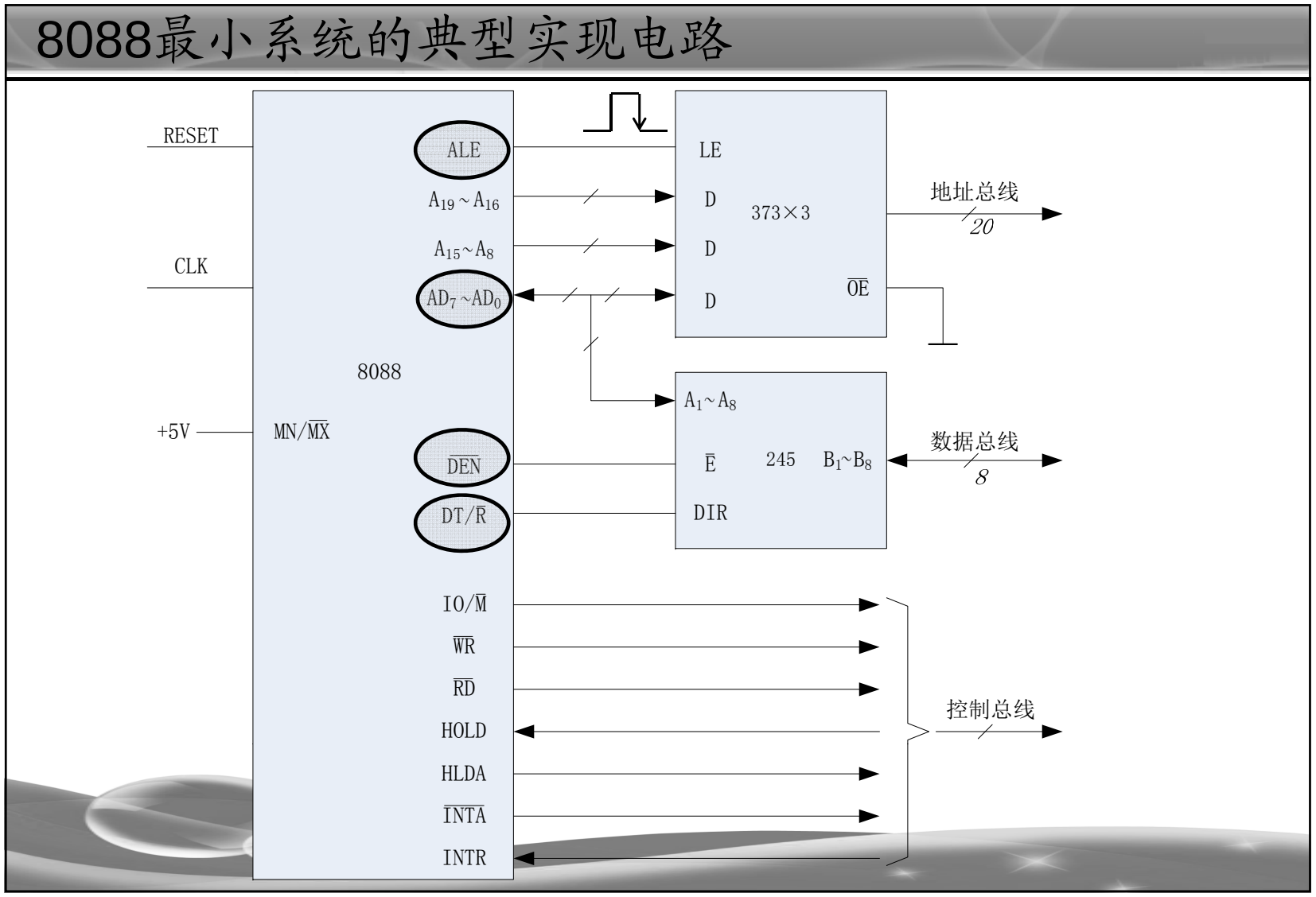
8088最小系统的典型实现电路



8088系统总线的形成（最小系统）

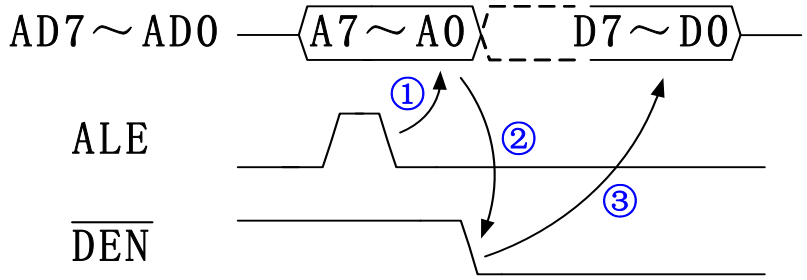
- 电路实现方案
  - 用3片8位锁存器（例74LS373）实现地址锁存。ALE为锁存控制信号，$\overline{OE}$=0输出地址；
  - 用1片双向缓冲器（三态门,例如74LS245）用作数据总线隔离，DT/$\overline{R}$控制方向，$\overline{DEN}$作为使能信号；
  - 控制信号由8088直接产生。

8088最小系统的典型实现电路



8088最小系统的典型实现电路

- $AD_{7\sim0}$解复用和地址锁存
  - ■先传送地址，后传送数据



①总线出现地址后，在ALE下降沿，地址被373锁存；

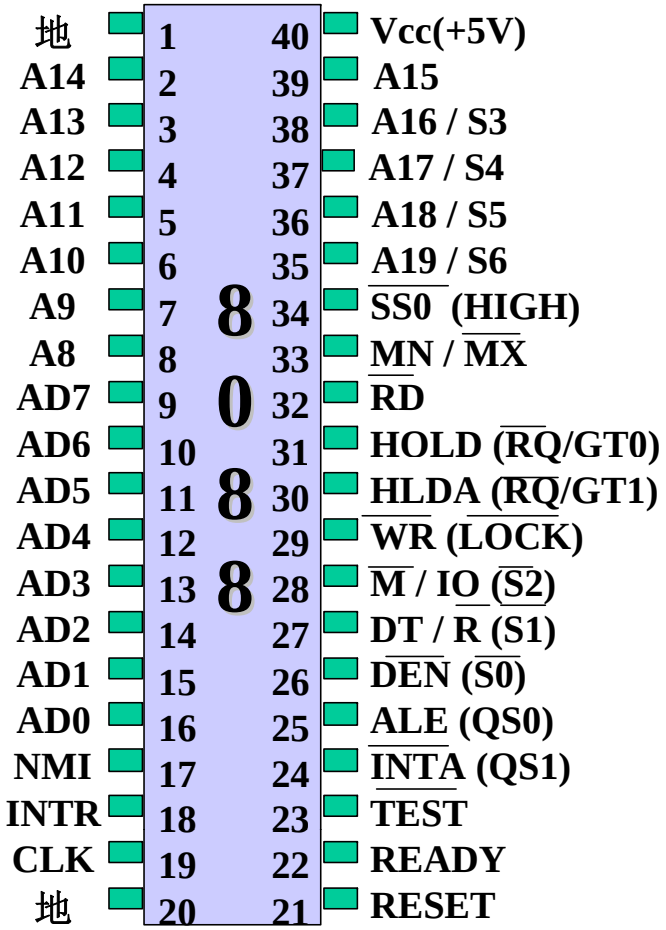
②总线出现地址时，$\overline{DEN}$高电平，245失能，数据总线被禁止。

③总线出现数据时，$\overline{DEN}$变低，245使能，数据总线开通。

- 数据传送方向的确定
  - ■数据输出时DT/$\bar{R}$高电平，245的DIR高电平：A ⟶ B
  - ■数据输出时DT/$\bar{R}$高电平，245的DIR高电平：B ⟵ A

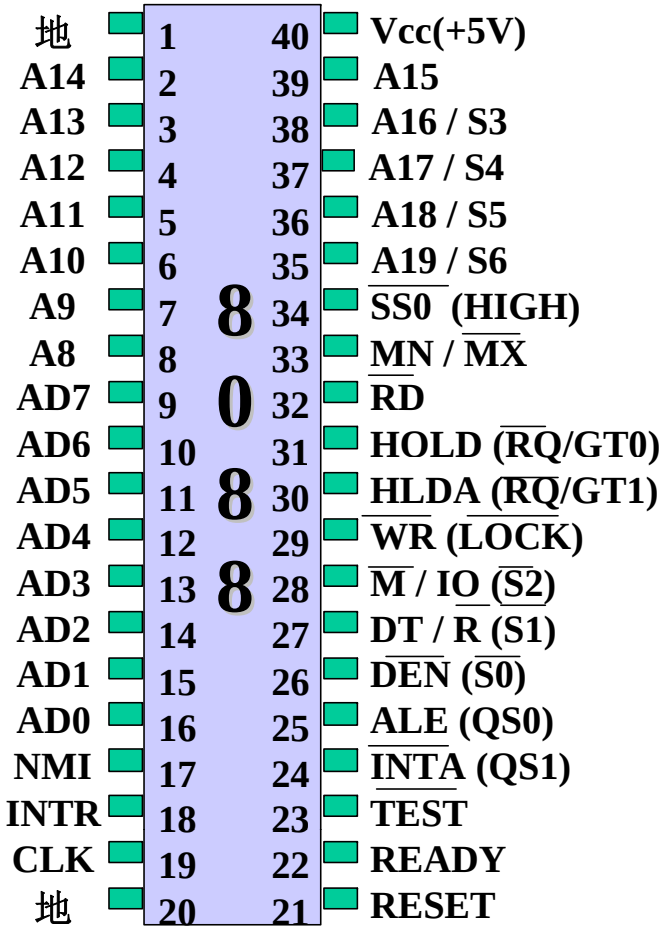
8088的外部引脚

- INTR(18,中断请求信号,入)
  - 高电平有效
  - 外设发来的可屏蔽中断请求信号。
- $\overline{\text{INTA}}$(24,中断请应答信号,出)
  - 低电平有效
  - 向外设回应的中断应答信号。
  - 连续2个负脉冲
- NMI(17,非屏蔽中断请求信号,入)
  - 非屏蔽中断请求信号
  - 高电平有效



8088的外部引脚

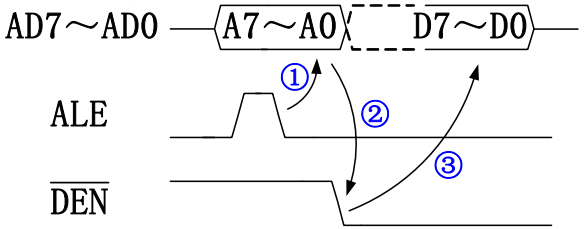
- READY(22,准备就绪信号,入)
  - 高电平有效
  - 存储器或I/O口准备好时将该信号置为高电平。
- TEST(23, 测试信号,入)
  - 低电平有效
  - 执行WAIT指令时CPU监视TEST端，为低电平则执行WAIT后面的指令；为高时CPU空转。
  - 用来与外设同步。



第5节 8088处理器的时序

8088处理器的时序

- 时序概念（Timing）
  - 时序是指引脚信号的时间顺序关系。



  - 时序描述CPU如何通过总线对外完成各种操作（总线操作）：
    - 存储器读操作
    - I/O读操作
    - 存储器写操作
    - I/O写操作
    - 中断响应操作
    - 总线请求及响应操作

和时序相关的几个概念

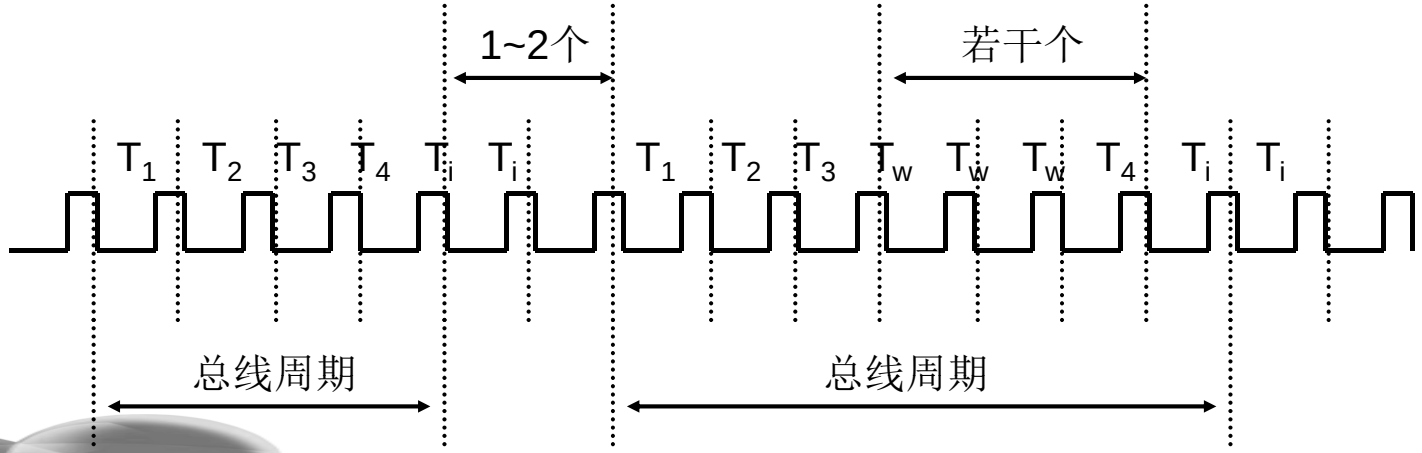
- 周期
  - 时钟周期 | 总线周期 | 指令周期
- 时钟周期（**Clock Cycle**）
  - 时钟的周期，时钟频率的倒数。 8088：4.77MHz
- 总线周期（ **Bus Cycle** ）
  - **CPU**通过总线与外部进行基本操作（一次数据交换）的过程
  - **I/O**读或写总线周期，存储器读或写总线周期,…
- 指令周期（ **Instruction Cycle** ）
  - 指令经取指、译码、读写操作数到执行完成的过程

指令周期 > 总线周期 > 时钟周期

- 总线周期（即总线操作）的产生
  - 指令取指阶段：存储器读总线周期(读取指令代码)
  - 源操作数是存储单元的指令：存储器读总线周期
  - 目的操作数是存储单元的指令：存储器写总线周期
  - 执行IN指令：I/O读总线周期
  - 执行OUT指令：I/O写总线周期
  - CPU响应可屏蔽中断：中断响应总线周期
- 空闲总线周期
  - CPU不执行任何存储单元或I/O操作，则执行空闲周期T_i（Idle）

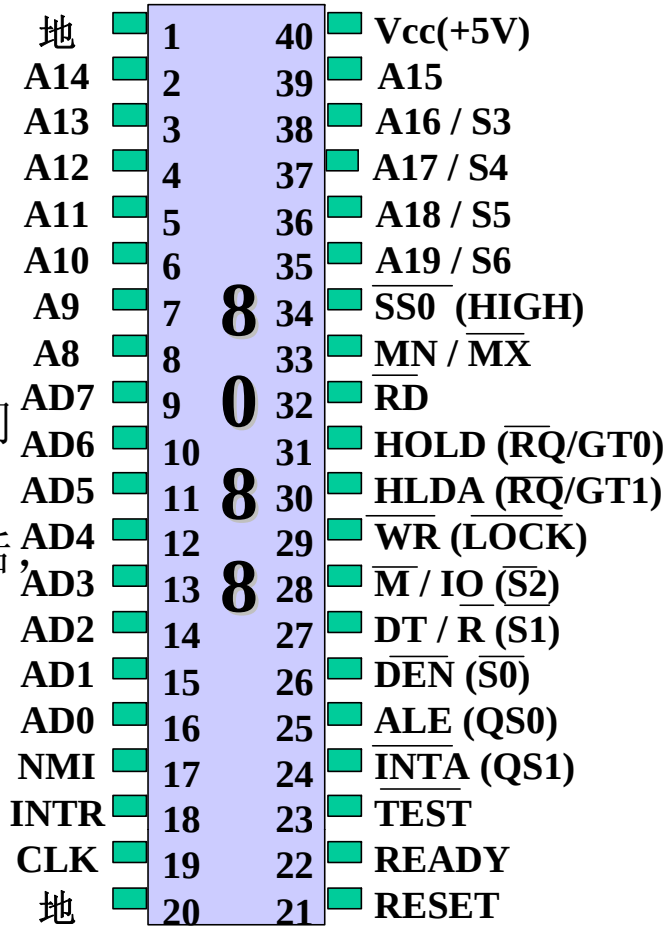
总线周期的构成

- (基本)总线周期需要4个时钟周期：T1、T2、T3和T4
  - 时钟周期也被称作“T状态” (T State)
  - 空闲时钟周期T_i，在两个总线周期之间插入。
  - 当要延长总线周期时要插入等待状态Tw（Wait）
    - 在T3和T4之间插入Tw

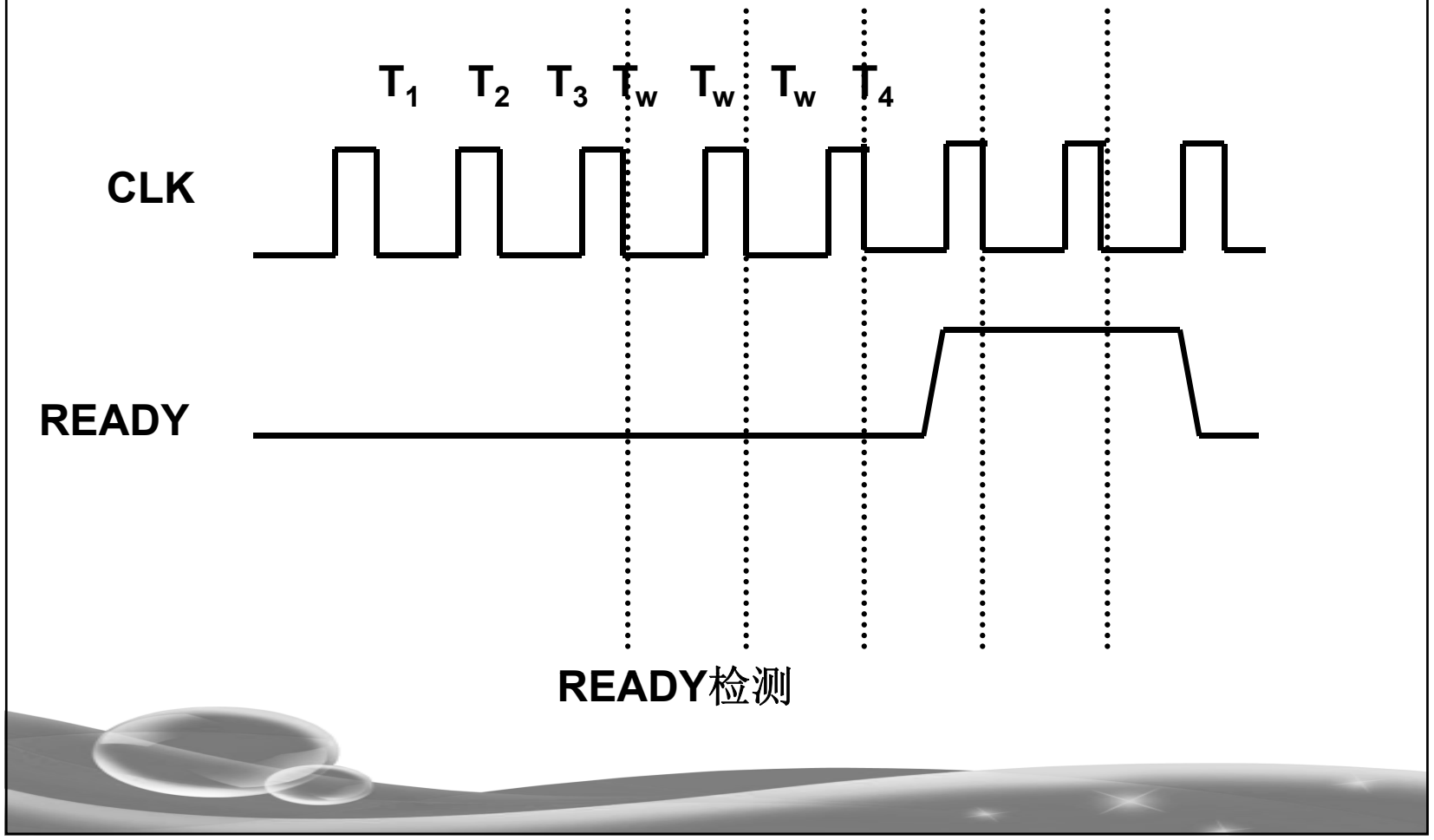


CPU和外设操作的同步：READY,等待状态Tw

- READY(22,准备就绪信号,入)
  - 高电平有效
  - MEM或I/O准备好时将该信号置高
  - CPU在T3期间采样READY信号
    - ◆1.T3期间检测READY是否有效？
    - ◆2.如果READY无效，在T3和T4间插一个等效T3的Tw状态，转1
    - ◆3.如果READY有效，执行完T3后，进入T4。

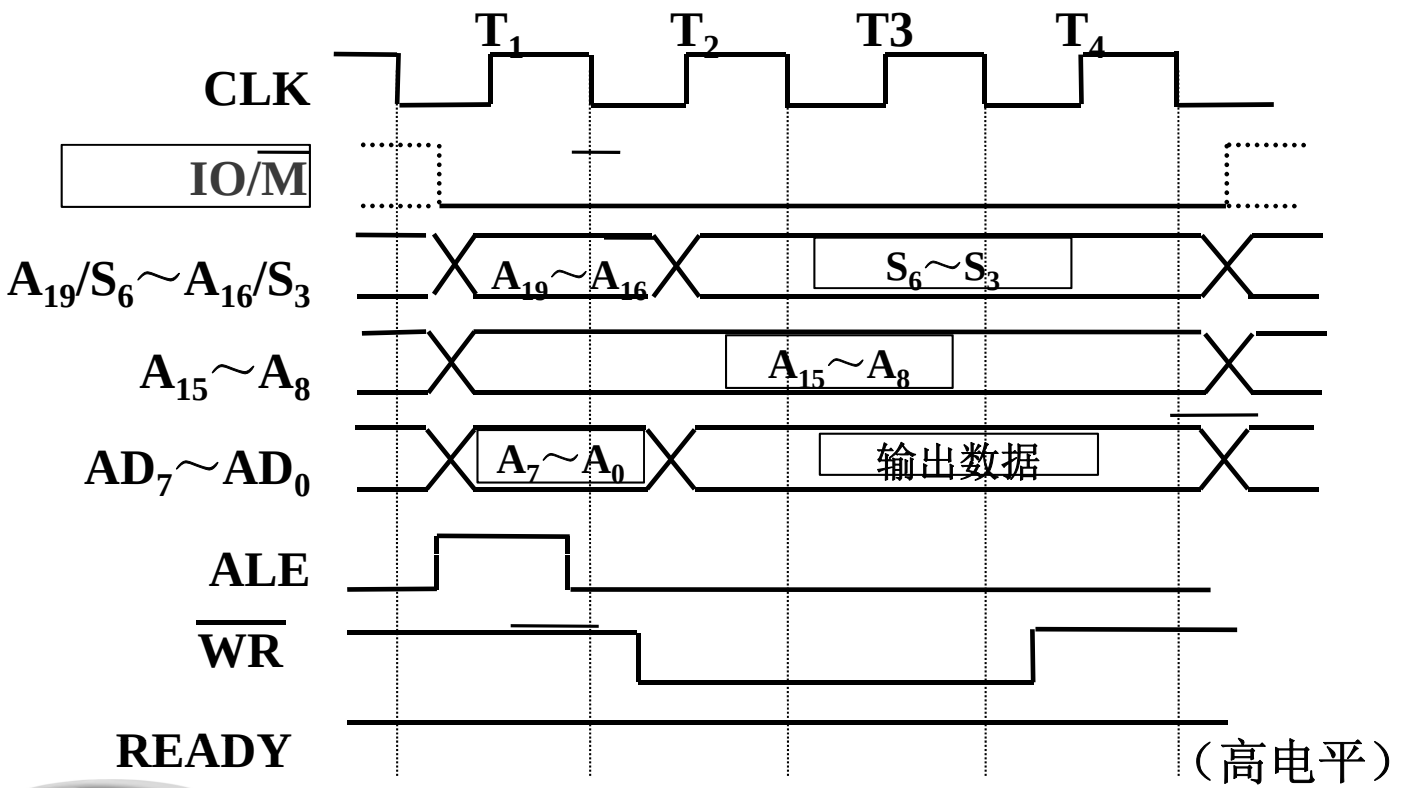


同步：T3和T4间插入等待状态Tw

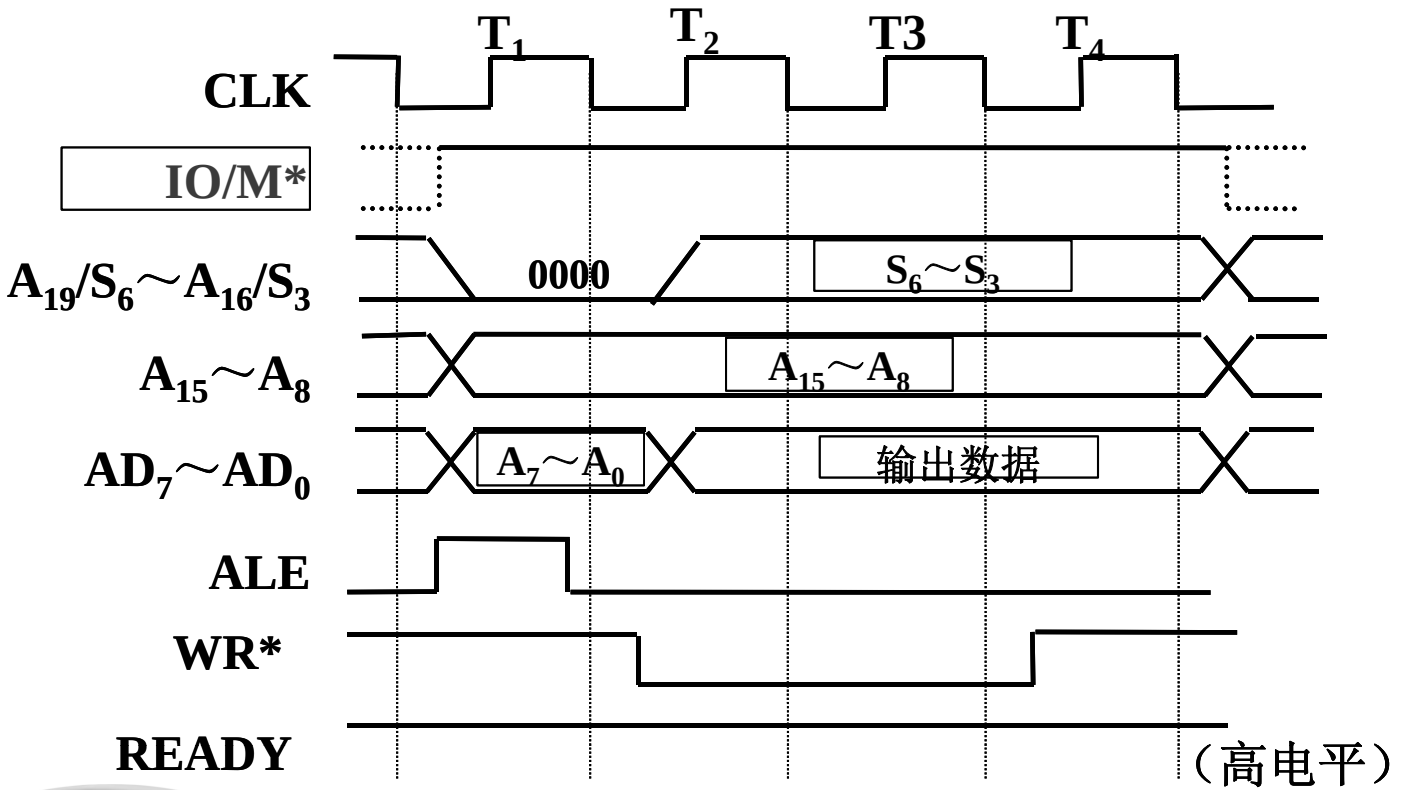


- 基本的总线周期
  - 存储器读
  - 存储器写
  - I/O端口读
  - I/O端口写
  - 中断响应

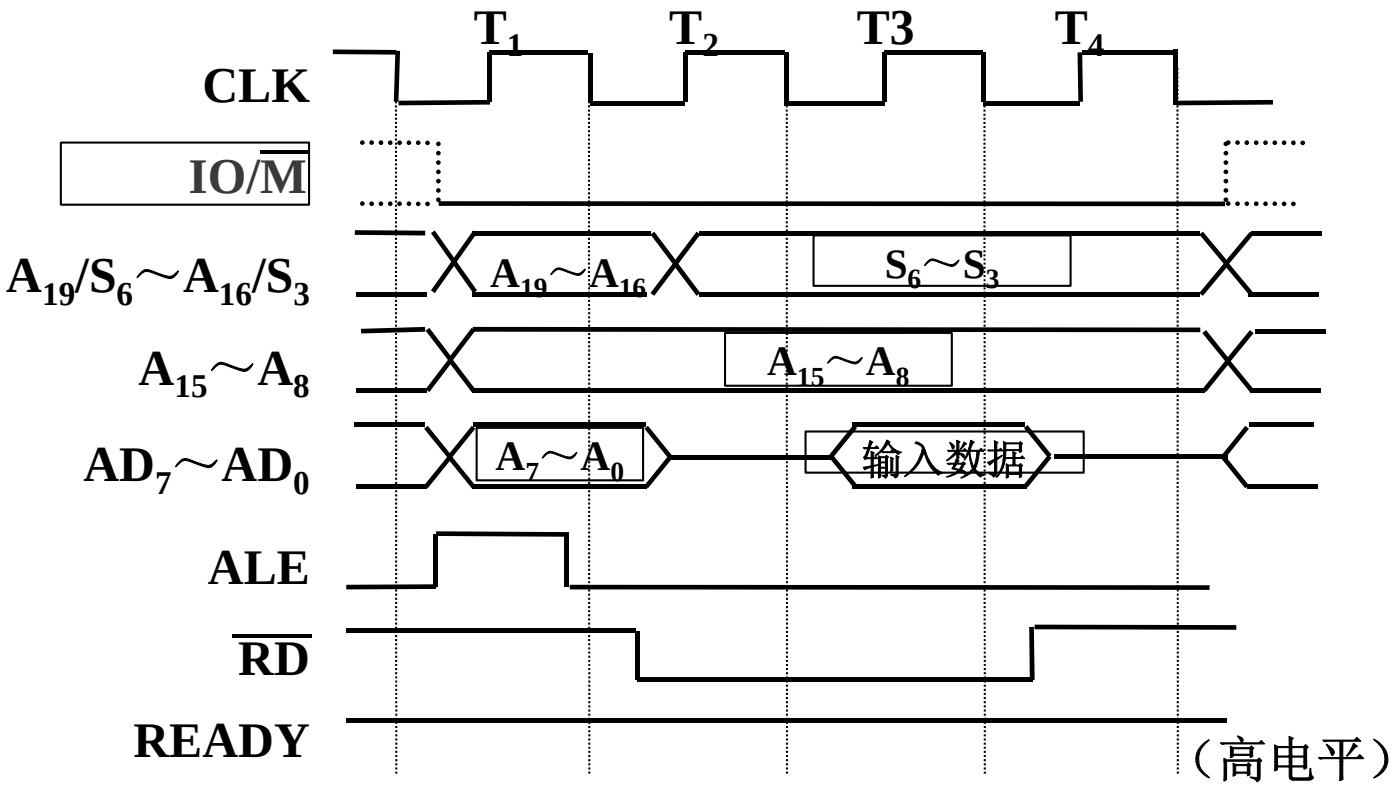
存储器写总线周期


T1
T2
T3
T4
CLK
IO/M
A19/S6～A16/S3
A19～A16
S6～S3
A15～A8
A15～A8
AD7～AD0
A7～A0
输出数据
ALE
WR
READY
（高电平）

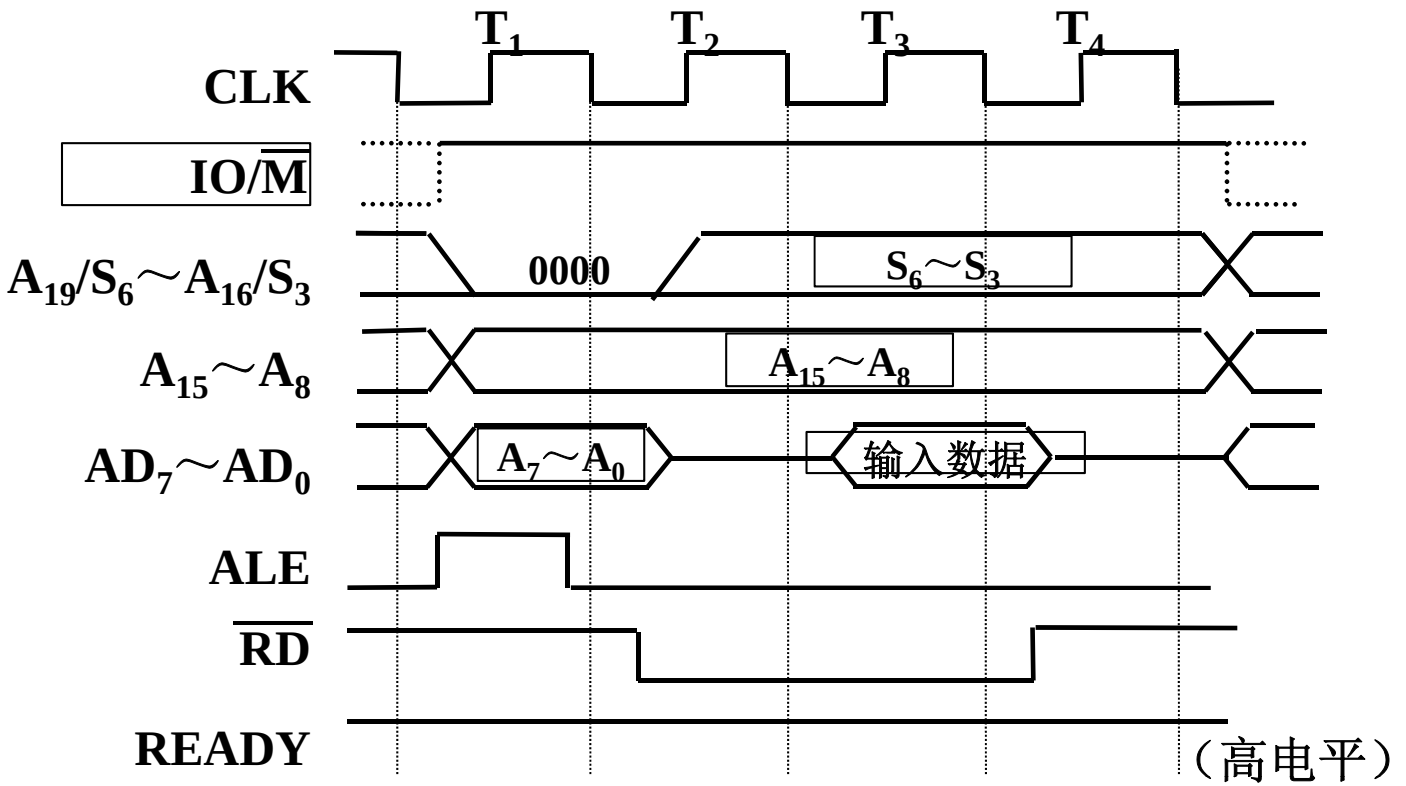
I/O写总线周期


T1
T2
T3
T4
CLK
IO/M*
A19/S6～A16/S3
0000
S6～S3
A15～A8
A15～A8
AD7～AD0
A7～A0
输出数据
ALE
WR*
READY
（高电平）

存储器读总线周期


T1
T2
T3
T4
CLK
IO/M
A19/S6～A16/S3
A19～A16
S6～S3
A15～A8
A15～A8
AD7～AD0
A7～A0
输入数据
ALE
RD
READY
（高电平）

I/O读总线周期


T1
T2
T3
T4
CLK
IO/M
A19/S6～A16/S3
0000
S6～S3
A15～A8
A15～A8
AD7～AD0
A7～A0
输入数据
ALE
RD
READY
（高电平）