# 计算机组成

1. Lec01 计算机体系结构
   1. 冯诺依曼结构
      1. 输入
      2. 输出
      3. 存储器
         1. 指令与数据合并存储
      4. ALU算术逻辑单元
      5. 控制单元
   2. 哈佛结构
      1. 指令与数据分开存储
2. Lec01 Microcomputer system
   1. 嵌入式系统
      1. 使用一个microcontroller或microprocessor来**一次做一个任务**
      2. Microcomputer 微型计算机
         1. Microcontroller 微控制器
            1. 有一个CPU和一些固定数量的RAM，ROM，IO port在一个chip上
            2. CPU: Microprocessor 微处理器

不包含RAM，ROM，I/O port在chip上

CPU

ALU

算数功能

逻辑功能

CU（Control Unit）在instructions下工作

instruction decoder：将指令译码并生成控制信号

program counter：指向下一条要执行的指令

Instruction Set 指令集

CISC: Complex Instruction Set Computers

指令长度可变

指令执行时间可变

更多指令格式

向上兼容：新的版本兼容老的版本

RISC: Reduced Instruction Set Computers

固定擦汗功能都

所有指令执行时间固定

**容易使用流水化**

更少的形式

Register

* + - 1. Memory
         1. 存储单位

Bit 位 0/1

Byte=8 Bits（微处理器的**最小可寻址单元**）

Nibble=4 Bits

**Word**=CPU一次可以处理的最大位数，依赖于CPU的**Register**和**Bus宽度**

* + - 1. Input/output ports
      2. Bus总线
         1. 种类

固定

多路

* + - * 1. Arbitration

中心化：由Bus Controller

分布式：每一个模块自行控制

* + - * 1. Timing

同步：由global events控制

异步：设备有自己的clock

* + - * 1. 四种bus体系

Single-bus

pros:：简单

cons：吞吐率差

CPU-Central Dual-Bus

pros：效率高

cons：memory和IO之间需要经过CPU使得CPU性能差

Memory-Central Dual-Bus

pros：CPU性能和数据性能都好

* + - * 1. 三种Bus种类

Data Bus

双向

宽度与CPU的Register宽度相同

决定了一个cycle可以传输多少数据（定义为word）

Address Bus

单向

决定了总的可寻址内存的大小

Control Bus

传输控制信号（模块、data bus、address bus控制）

* 1. I/O device（设备而非仅仅接口）
  2. 软件
     1. 系统软件
     2. 应用程序

1. Lec02 再论Memory和IO
   1. Memory
      1. 位置
         1. CPU中作为Register
         2. Internal：主存
         3. External：磁盘
      2. 传输单位
         1. Internal：word
         2. External：block
      3. 最小可寻址单元
         1. Internal：byte
         2. External：cluster
      4. 访问方式：
         1. Sequential 串行访问：tape。访问时间取决于位置
         2. Direct 直接访问：disk。独立的block由独立的地址，访问时间取决于位置和先前的位置
         3. Random 随机访问：独立的位置有独立的地址。访问时间**与数据位置与上一个数据的位置都无关**
         4. Associative 关联访问：数据存储基于它的内容比例而非地址
      5. 性能指标
         1. Access Time：从输出地址到获取有效数据
         2. Memory Cycle Time：Access+Recovery
         3. Transfer Time：数据移动的速率
      6. 物理类型
         1. 半导体：01二值，可写可读
            1. RAM-Random Access memory：可写可读，易失

DRAM：电容

SRAM：开关

* + - * 1. ROM-Read-Only memory
      1. 根据易失性分类
         1. 易失

DRAM

* + - * 1. 非易失

SRAM

ROM

* + 1. 内存地址译码
       1. 内存空间大小：前几位保证\# CS，后面可以从全0取到全F
          1. RAM：32K\times8
          2. ROM：64K\times8
       2. 物理地址：PA=SEGMENT\_VALUE<<4+OFFSET
       3. 三八译码器：选中的端口为CBA对应的值。
       4. 地址别名：同一个位置可以有多个地址，**因为不是所有的地址线都在译码电路中**
       5. 数据一致性
          1. 总和校验码Checksum Byte

将所bytes加起来并抛弃进最高位位，取2的补码（取反加一）得到checksum byte

将checksum byte与数据一起存储

**检验时**算一下数据的bytes和看是否等于checksum byte

* + - * 1. DRAM校验码 Parity Byte

even parity：当1的数量为奇数那么parity bit=1，反之为0（**保证data+parity的1的数量为偶数**）

**odd parity（8086）**：当1的数量为偶数那么parity bit=1，反之为0（**保证data+parity的1的数量为奇数**）

* + - * 1. 一整个包CRC
      1. 8086 **奇偶地址bank**
         1. 高地址：奇地址；低地址：偶地址
         2. A\_1~A\_{19}并行访问两个bank
         3. bank select 信号

A\_0：选中low bank

\bar BHE（bank high enable）：选中 high bank

* 1. IO
     1. IO寻址方式
        1. Memory-mapped IO: 一个地址空间，一套指令
        2. Isolated IO: 两个地址空间，两套指令
     2. IO与CPU的交互过程
        1. 1. CPU检查IO模块来获取设备状态
        2. 2. IO模块返回设备状态给CPU
        3. 3. 如果IO准备好了，CPU就请求数据传输
        4. 4. IO模块从设备获取单位数据
        5. 5. IO模块向CPU传输数据
     3. IO命令
        1. Control
        2. Test (status bit)
        3. Read/Write
     4. IO控制技术
        1. Programmed IO，程序控制IO（*需要CPU介入*）
           1. 从CPU角度看数据传输类似内存传输
           2. 每一个device给定一个**独一无二的identifier**
           3. 具体过程

CPU请求IO操作

IO执行操作，结束后改变**状态字status bit**

**同时CPU周期性检查状态字**

IO不告知CPU也不会终端CPU

CPU必须等待IO执行完成

* + - * 1. 问题：简单，但是CPU等待时间太长
        2. 适用情况

1. 设备较少

2. 设备速度快

* + - 1. Interrupt IO，中断控制IO（*需要CPU介入*）
         1. 不用CPU等待从而节省时间。当IO结束时通过**中断**Interrupt来告知CPU
         2. 具体过程

CPU请求IO操作，**然后干别的事情**

IO执行操作，**结束时发送中断告知CPU**

**CPU处理中断**

1. 保存上下文（context），即Register

硬件：保存最重要的信息：CS&IP

软件：在ISR中保存所有寄存器

2. 处理中断

3. 恢复保存的上下文

* + - * 1. 设计问题

CPU何以知道中断来源

一个设备一个接脚：**扩展性差**

Software Poll（软件依次查询）

效率差

扩展性差

Dasiy Chain（菊花链），即Hardware Poll（一个bus贯穿设备，INTA信号依次传入每个IO，如果不是当前设备发送中断则INTA原样传出）（**实际使用的方式**）

如何定位中断处理程序

使用一个**总的中断处理程序**，这个程序依次检查各个设备并获取对应的的处理程序的地址

使用**中断向量表（interrupt vector）**，IVT存储在一个固定位置，处理程序可以存放在任意位置

key为IO编号

value为对应**中断处理程序的入口地址**

如何处理多个同时到达的中断请求

给中断设置优先级

高优先级的中断可以抢占低优先级的中断

加入新设备设置中断的过程

1. 选择一个中断号N（代表IVT中的位置）

2. 将驱动ISR地址装入IVT

3. 向CPU发送中断请求（INTR）

4. CPU确认中断，发送（INTA）

5. IO在收到INTA后将INT type送上data bus

6. CPU保存上下文并处理中断

7. CPU更改PC并返回

* + - 1. Direct Memory Access（DMA），直接存储器访问（*不需要CPU介入*）
         1. 不需要CPU控制，而使用额外的模块**DMA controller**来控制。
         2. **数据必须先传到DMA controller ，然后由它传给Memory**
         3. 具体过程

CPU告知DMA controller的内容。然后继续其他操作

读写操作

设备地址

数据的**开始地址**

要传输**数据的大小**

DMA controller 执行数据传输，结束时发送中断

* + - * 1. Transfer Cycle Stealing：**指令周期**>**总线周期**

问题：可能由于DMA导致指令被挂起

方案：DMA一次只使用**一个处理器周期**来传输**一个word**

1. Lec02 芯片组织（Chip Organization）
   1. 减少地址介绍数量：
      1. 多路（multiplex）行列地址：将16Mbit组支撑2048×2048×4bit
      2. 增加一个地址接脚，地址空间×4
   2. 芯片扩展方式
      1. bit extension 位扩展：
         1. 目的：当模块需要**比给定芯片更大的传输单位**。如8个256k×1bit组成一个256k×8bit的芯片。内存空间大小不变，但是位数变多了。
         2. 做法：将一个地址传给多个芯片，一次取出多位。每个芯片有相同的地址空间
      2. word extension 字扩展
         1. 目的：需要比给定芯片更多的字。如将8个256k×8bit组支撑一个2M×8bit。
         2. 做法：保证一次取出的位不变增大容量。每个chip有不同的地址空间，因为地址中有几位用来选chip。
2. Lec 3 80X86 微处理器
   1. 8086 vs 8088
      1. 8086的字长
         1. 16-bit处理器：16bit为二进制**寄存器的宽度**，从某一个地址开始传16-bit
         2. 20-bit地址，内存为2^{20}=1MB
      2. 8088的两个字长
         1. internal：16-bit：寄存器16-bit
         2. external：8-bit：**计算机系统Computer System的字长**，对外接口的字长，为了适应8-bit的外设：
   2. 8086内部结构：BIU和EU **同时** 工作
      1. Bus interface unit：CPU与**内存**和**外设**之间数据传输
         1. 寄存器分为**通用寄存器（8个）**，**段寄存器（4个：CS，DS，ES，SS）**和**指令寄存器（1个：IP）**
         2. *IP 寄存器（Instruction Pointer）即 PC（Program Counter）*
         3. **流水化**：当EU在执行时，BUI取下一条指令并放入一个**6-byte的指令队列**
            1. 当指令队列中**有两个byte空出时**，BIU就会启动取指令
            2. EU从指令队列中找**预取**的指令
            3. 当jump指令发生时所有的预取的指令都无效
      2. Execution unit：**执行指令**
         1. Registers寄存器
            1. 4个16-bit **通用寄存器**：AX(Accumulator)，BX(Base)，CX(Count)，DX(Data)——可拆成两半
            2. 2个16-bit**Pointer Registers**：SP(Stack)，BP(Base Pointer)
            3. 2个16-bit**Index Registers**：SI(Source Index)，DI(Destination Index)
            4. 1个16-bit**Flag Register**：只有9个bit被使用
         2. ALU
   3. 8086/8088的**两种工作模式**
      1. Minimum mode：一个CPU负责所有控制信号的生成
      2. Maximum mode：有协处理器协同处理
   4. 高阻态的原因：pin从input状态向output状态转换需要时间
   5. **8086编程**：**Segment**
      1. **segments**
         1. code segment：**保存指令**
            1. 逻辑地址：CS：IP
            2. **当要取的指令超过代码段（64K）**：应修改CS让指令包含在代码段内。*但是****CS无法使用MOV****修改*
         2. data segment：**保存要被处理的数据**
            1. 逻辑地址：DS：offset（offset可以是**BX,SI,DI**）
            2. 物理地址可以是**8bit或者16bit**
            3. 若数据超出数据段就可以修改DS（可通过MOV直接修改）
            4. 超过一个byte的数据存储：**8086使用小端法**

小端法（little endian）数据的低字节存储在**低内存地址**位置

大端法（big endian）数据的低字节存储在**高内存地址**位置

* + - 1. stack segment
         1. CPU使用RAM的一部分来临时存储信息
         2. 逻辑地址 SS：SP（特别情况下SS：BP）
         3. stack segment向下增长（grow downward）

**从高地址开始存储**

目的：防止越界而导致崩溃，保证code segment和stack segment不会overlap

* + - * 1. **stack的依次push或pop必须处理16-bit**
        2. pop并不抹除数据，只不过改了SP
      1. extra segment
         1. 一个额外的段，一般是为了**字符串操作**
         2. 逻辑地址ES：offset（BX，SI，DI）
      2. *段最多64K=2^{16}，因为寄存器只有16bit。因此****段开始的地址一定要可以被16整除，如XXXX0H***
    1. Logic&Physical Address
       1. 物理地址：20-bit物理地址从00000H到FFFFH
       2. 逻辑地址：格式为**CS：IP**
          1. 16-bit段值segment value：段的开始位置
          2. 16-bit偏移量offset address：段内的相对位置
       3. 逻辑地址\rightarrow 物理地址映射
          1. 16bit地址生成20bit地址：**CS<<4+IP**
          2. logic\rightarrow physical为一一映射，但反之不然。
       4. **物理地址Warp-around（越界）**：当计算到的物理地址超过**FFFFFH**，就**抛弃溢出位**，剩下的等价于从00000H重新开始
       5. **段Overlapping（重合）**：在某些特殊的情况下也许有用
  1. BIOS（Basic input-output system）功能
     1. 在开机时检测连接的设备并检测、报告错误
     2. 将DOS加载进RAM
     3. 将PC的控制权交给DOS
  2. Flag Register：16-bit的status register，保存了处理器状态字（PSW：processor status word）、
     1. 6个条件flag
        1. CF（carry flag）：当最高位有进位（8bit的第7位，16bit的第15位）时设为1，用于检查**有无符号计算的错误**
        2. PF（parity flag）：**奇偶校验位**，当**低byte**有**偶数个1**是设为1
        3. AF（Auxiliary carry flag）：**辅助进位（BCD相关计算）**，当d3~d4位有进位时设为1.
        4. ZF（Zero flag）：当计算结果位0时设为1
        5. SF（Sign flag）：复制计算结果的sign bit（最高位）
        6. OF（Overflow Flag）：**是否溢出**
           1. 当有符号操作的结果太大时引起**符号位错误**，用于检查**有符号计算的错误**
           2. 当d6\rightarrowd7和d7\rightarrowd8**只有一个**进位发生代表溢出
     2. 3个控制flag
        1. DF（direction flag）：字符串和查表的方向
        2. IF（interrupt flag）：中断使能
        3. TF（Trap flag）：=1则在每一个时钟周期的末尾中断，来打印每一个寄存器的值（用于debug）
  3. **8086寻址方式（8086 Addressing Mode）**
     1. 寄存器寻址方式：**MOV BX,DX**
        1. 不需要访问内存
        2. **CS和IP不能用这种方式访问**
     2. 立即数寻址方式（Immediate Addressing Mode）： MOV AL,40H
        1. source 是一个constant常量
        2. **intermediate立即数无法被写入段寄存器**
        3. 数据嵌入指令中，**不需要访问内存**
     3. 直接寻址方式（Direct addressing mode）：MOV DL,[2400] MOV[3518],AL
        1. 数据存储在内存中，指令中给出地址，如[]中的2400即地址的offset，段值默认为DS
        2. 地址直接嵌入指令
        3. **需要访问内存**
     4. 寄存器间接寻址方式（Register Indirect Addressing Mode）：MOV AL,[BX]
        1. 地址offset在寄存器BX中在[]中传入，默认段值在DS
        2. 可以传地址的寄存器只有**BX，SI，DI**
        3. **需要访问内存来获取数据**
     5. Based Relative Addressing Mode：MOV CS,[BX]+10
        1. Based访问方式必须使用**Based Registers**
        2. 传入地址的寄存器只有**BX,SP**，其默认段值为**DS**和**SS**
        3. 实际算地址应**将+10加在地址上而非取出的值上**，即实际物理地址为**DS：BX+10**
     6. Indexed Relative Addressing Mode MOV DX,[SI]+5
        1. 必须使用Index寄存器**DI，SI**，默认段值寄存器为**DS**
     7. Based Indexed Relative Addressing Mode：MOV CL,[BX][DI]+8
        1. 使用一个**based**和一个**indexed**寄存器来表示地址
        2. 实际独立地址为DS：BX+DI+8
        3. 可以用的based寄存器为**BX,BP**，默认段值为**DS,SS**
     8. *8086允许重写默认段值寄存器，如BP的默认段值为SS*，MOV AX,CS:[BP] 将默认段值视为CS

1. Lec09 中断详述&8259
   1. 中断的种类
      1. 硬件中断
         1. NMI：non-maskable interrupt（不可屏蔽）
            1. 触发方式：

NMI pin 高

input-signal 高

在上升沿高电平维持两个周期

**不检查IF** ：因此不可屏蔽

* + - * 1. 种类：INT 02（**只有这一种**）
        2. 来源举例：parity bit检测错误
      1. INTR：interrupt（可屏蔽）
         1. 触发方式

INTR pin 高

input-signal 高

**当且仅当IF=0才可触发**

SIT 指令 设置IF=1

CLI 指令 设置IF=0

* + - * 1. 种类：非预定义种类
        2. 来源举例：IO发送的INTR
        3. **处理可屏蔽中断的过程**（*CPU视角*）

一、CPU响应INTR请求的过程

CPU在**每一个指令的最后一个周期**检查**INTR**和**IF**，如果都**high**则响应中断请求

CPU发送**两个低电平**信号给**\bar INTA**

第一个用于**clear INTR** pin。否在周期结束会再check INTR，再restart ISR——进入死循环

第二个用于让**IO将中断种类N 放在data bus上**

二、CPU执行中断处理程序（ISR）

CPU读取中断种类（中断号）N

save context

清除IF和TF

IF=0 意味着**CPU默认在处理中断时屏蔽其他中断**

TF=0 意味着**在处理中断时不更新寄存器的值**

保存CS、IP、**FR** 进堆栈

加载ISR的地址进入CS：IP

IRET：将CS，IP，FR从堆栈中弹出

CPU继续其他任务

* + 1. 软件中断
       1. INT 指令(INT N)，
          1. 直接传入中断种类并执行中断，**不检查IF**
          2. 可以理解为类似于指令CALL ISR。但是也**INT与CALL也有一些区别**

跳转范围

CALL FAR 可以在1M范围内任意跳转

INT 只能跳转到一个固定位置

执行时间

CALL指令在指令序列中（先验信息）

external 外部中断可能在任意时刻出现

可屏蔽性

CALL 不可屏蔽

INT 不可屏蔽，但是一般外部中断INTR可以屏蔽

上下文

CALL：CS：IP

INT：CS：IP+FR

返回指令

CALL：RETF

INT：IRET

* + - 1. 预定义（predefined）中断
  1. 中断向量表的构成
     1. N=0~5，共五个：predefined
     2. N=5~31：共27个：系统中断
     3. N=32~255，共224个：留给用户定义
  2. 中断优先级
     1. CPU 视角
        1. 优先级排序：INT>NMI>INTR
        2. INTR 内部的优先级
           1. Software polling：先来的优先级高
           2. Hardware polling（daisy chain）：\bar INTA依次传入IO，连接在前面的有优先级高
     2. 向量化中断控制器：8259
        1. 具有很强的灵活性
           1. 可以处理最多64个中断输入
           2. 中断可屏蔽
           3. 可以通过编程设置优先级

1. Lec08 计时器&8253
   1. 实现循环的两种方式
      1. **软件**方式：使用变量控制，但是相当不稳定
      2. **硬件**方式：使用8253
         1. 系统接口：**有三个独立的计数器（16-bit down-counter：一个cycle将counter-1，到0后生成信号并中断CPU）**
            1. Counter 接口

CLK\_0——输入：时钟频率

**上升沿**：检查Gate

**下降沿**：将counter-1

GATE\_0——输入：计数器的**使能**信号

OUT\_0——输出：有三种形式

Square\_wave：方波

One-shot

Square-wave with various duty cycles：不同占空比的方波

D\_0~D\_7：Counter，计数初值，1~65536（二进制），1~10000（BCD）。必须时**偶地址**

* + - * 1. 控制逻辑接口

~CS

~RD，~WR

写

写控制字到控制寄存器

加载**低位**或**高位**byte进控制寄存器

读

simple read

读两次，先读low-order byte再读high-order byte

问题：直接读的时候和读回的时候有延时，要读的内容已发生改变

counter latch command

先发一个信号锁存（latch）要读的内容

再用simple read的方法读两次

A1，A0：用与选择Counter，当等于1，1时选中**控制字寄存器Control word Register**

指定哪些counter要被使用

指定各个counter的mode

指定各个counter的读写操作。读写可以只读高byte/低byte或分两次操作一个word

* + - 1. 实际频率的计算：**CLK\div Counter**。
         1. CLK频率为8253内部的频率，只**每过一个周期将Counter-1**
         2. Counter内为计数初值.
      2. Mode 0~4
         1. Mode 0

Output初始为 0

当Counter到0时**输出高信号并保持，直到Counter被重新加载**

**可以用Gate信号来使能**

**当新的值被加载时，就从新的值最为Counter来进行计数**

当Counter有两个byte时

第一个byte使当前计数终止

第二个byte写的下一个周期重新开始计数

实际频率：CLK/(Counter+1)\approx CLK/Counter

**不会自动重复计数**

* + - * 1. Mode 1：硬件可再触发的One-shot

每当一个One-shot的Gate=1，则出现一段低电平，这段低电平的长度由Counter决定。

**不会自动重复计数**

**新的值被加载不会影响此次计数**，只会影响下一次one-shot

* + - * 1. Mode 2：

一直保持高电平，再计数达到0的前一个周期（1\rightarrow 0）产生一个周期低电平。

即**N-1个周期high+1个周期low**

当**计数结束时会reload Counter**，重复此过程

当**在那一个低电平周期中Gate=0，则Output被立刻设置为高**

**当前计数不会被新的Counter影响，新的Counter在下次计数开始**

* + - * 1. Mode 3：方波生成器

**初始时为High**

Counter为偶数，则N/2个周期High，N/2个周期Low（通过每次将Counter-2）

Counter为奇数，则（N+1）/2个周期High（通过先将Counter-1，再每次-2），（N-1）个周期Low（通过先将Counter-3，再每次-2）

Gate使能

当低电平时Gate=0，Output立刻被设置为1

当Gate重新被设置为1时，将初始的计数值加载进Counter并重复计数

加载一个新Counter时，不会影响当前**半周期**

* + - * 1. Mode 4：Software Triggered Strobe

**初始为High**

在Counter**到0以后的一个周期Low**，然后重新加载Counter，实际周期为**N+1**

**加载新Counter后，将会从新Counter开始重新计数**（*但是与Mode 0有区别*）

第一个Byte**不会影响**当前计数

写第二个Byte才reload Counter

**自动重复**

* + - * 1. Mode 5：Hardware Triggered Strobe、

**初始为High**，当One-shot的Gate=1后开始计数

在Counter**到0以后的一个周期Low**，然后重新加载Counter，实际周期为**N+1**

Retriggering：此次计数中的one-shot，会**使得计数重新开始**

**新的Counter不会影响此次计数，只会影响下一次Trigger（One-shot）**

1. Lec11 连续数据传输&8251
   1. **Parallel** data transfer·
      1. 特点
         1. 每一个bit使用一个独立的line
         2. 通常8个或更多lane被使用
         3. 有额外的line用于传输控制信号
         4. **快速、昂贵、短距离传输**
   2. **Serial** data transfer
      1. 特点
         1. 只用一个line，数据的bit在这一个line上一个个依次传输
         2. 没有固定的line用于传输控制信号
         3. **慢速、便宜、长距离传输**
         4. **从低位开始传输**
      2. Data transfer rate
         1. Symbol rate：单位时间放上的symbol数量，**单位**：波特（baud）。
            1. 一个Symbol可以代表**一个或多个bit**
         2. Bit rate：单位时间传的bit数，单位bits per second（bps）
      3. Synchronization methods
         1. 异步start bit和同步synch character的区别
            1. 异步时无数据时line 保持ide（high），所以只要一个周期low即可表示开始
            2. 同步一直在传输数据（line上一直是0或1），所以要约定一串bits来表示数据传输开始
         2. Asynchronous 异步的
            1. **一次传输一个Byte**
            2. 每一个byte的开始是异步的，使用**start bit**来进行sender和receiver之间通信
            3. 数据在start bit和end bit之间

idle state：start bit之前和end bit之后

start bit：一个低电平信号，代表数据的开始。

stop bits：stop bits一定是高电平。用来确定transmitter和receiver对于一个Byte的约定是相同的

* + - * 1. Delay distortion：Receiver对Transmitter的采样最好在中间开始采样

解决方案：**receiver的时钟是transmitter时钟的N倍**，receiver每隔N/2个周期采样一次。**N越大越好**

N又称作**波特率因子**

* + - 1. Synchronous 同步的
         1. **一次传输一个block**
         2. sender和receiver之间需要**synch characters**
         3. 保证data中不含synch character：在数据中插入值来避免出现synch character。receiver需要用对应的方式解码
         4. 无start bit，因为时钟对齐
    1. 通信方式 Communication modes
       1. Simplex：双方**总是单向**传输
       2. Half-duplex：**同时只能单向**传输
       3. Duplex：**双向传输**
    2. Error detection
       1. Parity bit
       2. CRC
    3. Modulation and Demodulation
       1. 信噪比
          1. 信噪比高，应该用**symbol少**的，传输速率慢
          2. 信噪比低，应该用**symbol多**的，传输速率快
  1. UART 8251
     1. 可以**同步也可以异步**，对应的baud rate和character不同
     2. Full duplex，double-buffered
     3. 错误检测电路
     4. 8251编程
        1. 1. 写 控制寄存器，同步异步，**baut factor**时钟是波特率的多少倍。

1. 信号总结
   1. DRAM
      1. \bar RAS：row address select
      2. \bar CAS：column address select
      3. \bar WE：write enable——允许写
      4. Isolated I/O信号
         1. \bar OE：output enable——当此信号无效（高电平），则全chip都无效
         2. \bar MREQ：类似于\bar OE
      5. ALE：Access latch enable
         1. =1 说明lock打开，input和output相同
         2. =0 说明lock锁住，左边input pin可复用
      6. \bar WR：写数据
      7. \bar RD：读数据
      8. M/\bar IO：访问memory还是IO，**8086是IO isolated**
      9. INTR：中断使能
      10. \bar INTA：中断认可
      11. NMI：不可屏蔽中断
      12. HOLD：DMA controller申请Bus的使用权
      13. READY：表明Data bus上的数据是否有效（外部设备是否ready）
      14. RESET：产生长时间的reset信号给CPU，重启
          1. 将IP、DS、SS、ES都clear（置为0）
          2. 将**CS置为FFFFH**
          3. **原因**：下一个执行的指令地址为*FFFF0H*，即BIOS
      15. \bar BHE：是否选奇地址
      16. Data transceiver 信号
          1. \bar DEN：数据传输器Data transceiver使能信号
          2. DT /\bar R：data transceiver 方向。
             1. =1代表 sending data
             2. =0代表 receiving data