微机原理与系统设计笔记1 | 绪论与数制表示

- 打算整理汇编语言与接口微机这方面的学习记录。
- 参考资料
  - 西电《微机原理与系统设计》周佳社
  - 西交《微机原理与接口技术》
  - 课本《汇编语言与接口技术》王让定
  - 小甲鱼《汇编语言》

---

1. 介绍

2022年春学习了MIPS视角下的计算机体系结构与组成原理，同年夏自学了操作系统（科普级别的了解），汇编语言是学习操作系统的时候，基于学过的计组强行吃下来的，并没有单独学过。

秋天学学校的OS（比较深入），同时我也报了汇编语言与接口技术这门选修课。在这门选修课上我感觉汇编相关内容还是十分丰富的，并且在系统、底层、安全这些方面用处很多。

这个系列是对这方面知识的整理。打算在这个系列的最后进行课程内容的复习。

2. 概览

课程10章，前四章为汇编语言，后6章为微机硬件接口电路。

2.1 电子信息系统

- 有了CPU的电子信息系统功能更为丰富。
- 电子信息系统的**输入**是物理量
- 物理量转化为电量：**传感器**

  转化后的电量为 mv 级
- 电量放大并滤波：**前端调理电路**

  模电课程（本人不学），到伏特级，留调试点。
- 模拟信号到数字信号：**A/D转化**

  可以根据需求选择A/D芯片转化的精度和速度。
- 数字信号提交给CPU处理：**CPU**型号也是根据需求选择。
- CPU应当可以向A/D转化器件和前段调理电路发送控制信号，控制一些处理速度。CPU处理后的数字信号应有**存储器**存储。
- 需要输出的数字结果：**D/A转化**
- 与**前端调理**对称地，需要**后端调理电路**，其输出量来**控制外部器件/设备**；

  > 实验时这里后端调理结束后通过测试点引出到示波器了。
- 另一些应用场合，输出要**显示**（电视机）、打印（打印机）。

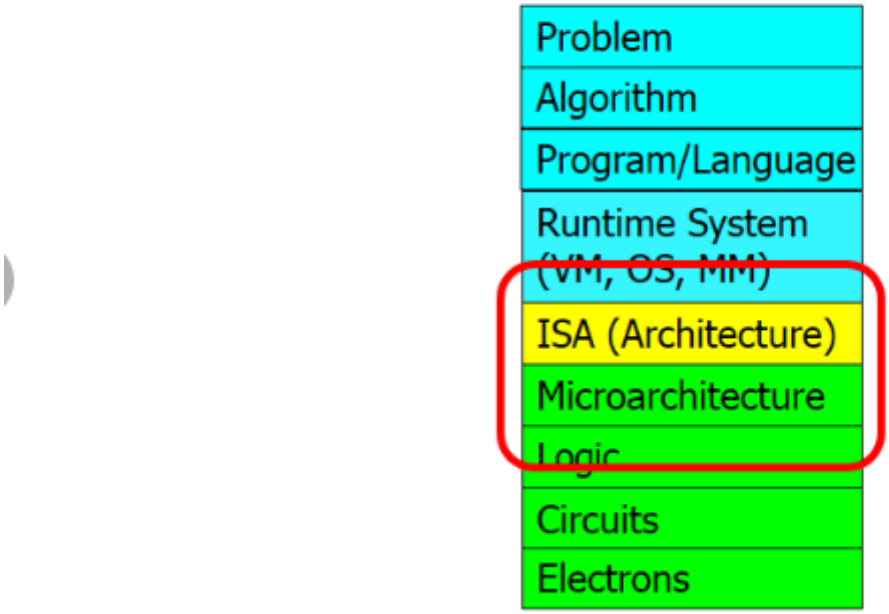
我们需要给CPU编程序（**汇编语言**），来决定整个系统怎么工作，这就是前4章的主要内容。不同的I/O设备的类型也有差异，设备接受信息的格式不同（如串行并行、模拟量数字量），所以CPU和设备之间要设计一段**接口电路**，这是后半部分硬件部分的内容（时序、存储器、I/O设备接口）。

2.2 微型计算机系统组成

A. 软件子系统

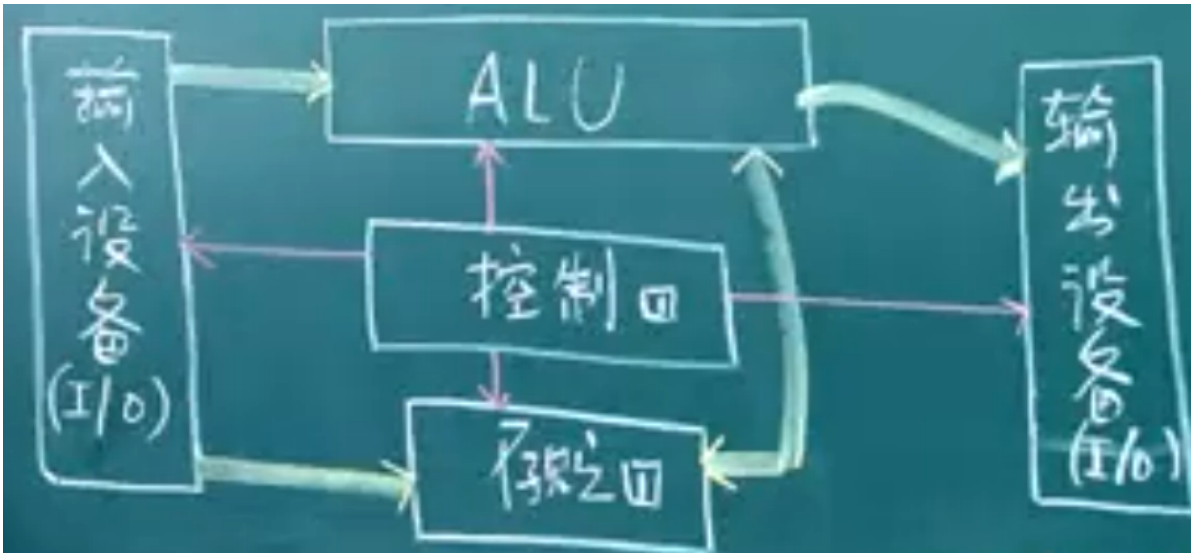
不是本课程的讲解内容。也可以简单再分为系统软件和应用软件。

"The purpose of computing is insight" (*Richard Hamming*)
We gain and generate insight by solving problems
How do we ensure problems are solved by electrons?



B. 硬件子系统

早期计算机硬件组成



> - 控制器控制ALU运算：ALU运算的数据在控制器控制下送往存储器；
> - 存储器中的数据被控制器控制器送到ALU；
> - 输入设备（IO）控制器控制输入设备输入数据到存储器或ALU。

- 随着集成电路发展，将ALU和控制器集成在一起，成为中央处理器**CPU**（也称微处理器）。
- 上述部件之间传递信息：
  - 数据信息、控制信息、地址信息

（这点很好理解）

- 对应的传送信息的线/通道：数据总线、控制总线、地址总线。

现代微型计算机硬件组成

- 以微处理器/CPU为核心来组成。CPU通过引脚来跟外部存储器、I/O实现地址、数据、控制信息交互。而8位CPU8个引脚，16位CPU16个引脚，是无法同时满足三条总线分开接入的。

> 8086CPU有16根数据总线，20条地址总线和16条控制总线。

- 因此实现了一种**分时复用**技术，**部分**引脚分时复用。

  因此需要设计一个电路（**系统总线形成电路**），鉴别复用的部分引脚此时到底输出/入的是什么信息，并输出输入。

> 要想设计这个电路需要了解CPU引脚的时序（第五章）。
>
> 引脚也称**微处理器级总线**。
>
> 转化后形成的三总线称为**系统总线**。外设就挂接在系统总线上。

- CPU具体如何与外设交互：
  - CPU要给存储器写数据，需要通过CPU的写控制引脚发送写控制命令，同时发送地址信息，写入对应地址，控制总线还要接收外设的状态信息。
- 硬件子系统中的存储子系统，由ROM和RAM组成。对于内部不含存储电路的CPU电子系统而言，必须用这两样ROM和RAM，在三总线上扩展设计存储器。

> 计组实验总线信息这里讲挺清楚的，汇编与接口这课详细讲了怎么设计存储器。
>
> ROM又称程序存储器，用专用设备写入，接入系统电路时是只读的。

2.3 汇编简介

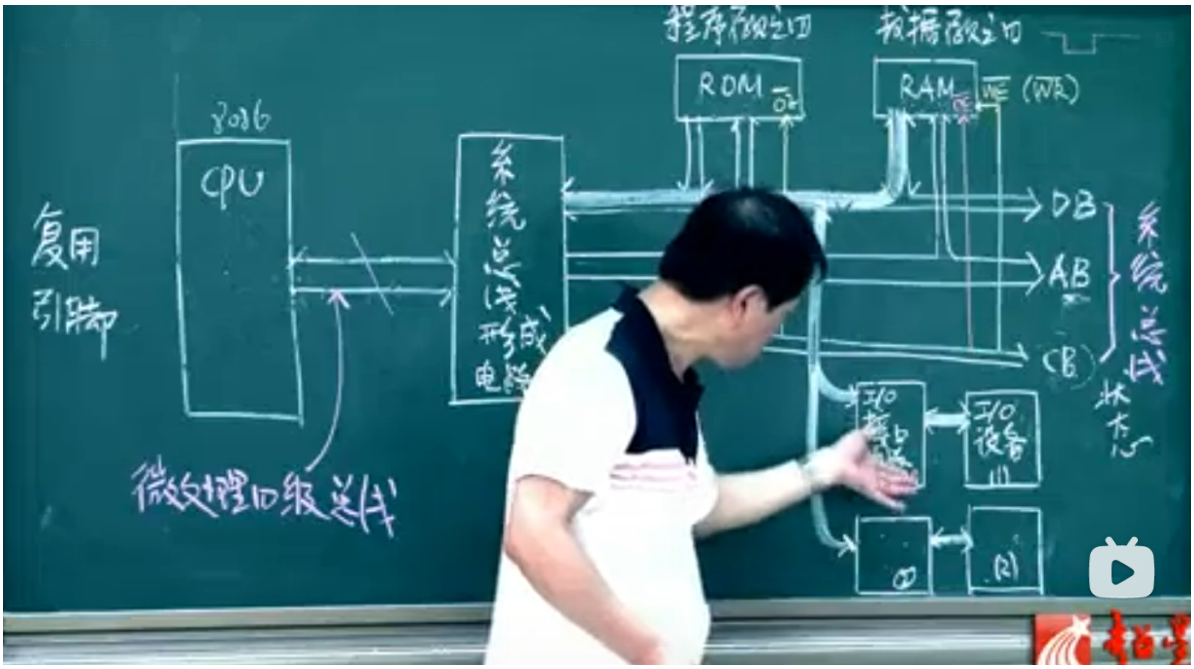
指令的概念

- 指令是一组二进制代码。用于明确指示上述系统中CPU的工作。汇编语言是二进制代码的符号化描述。

2.4 接口简介

前面提到了，I/O设备不会直接接到CPU上，要通过一段接口电路来完成标准化并与CPU连接。

> 接口电路也就是常说的接口。



I/O接口是连接CPU和I/O设备的控制电路。比如显示器与CPU之间通讯有显卡电路，音响与CPU之间有声卡。

当然，CPU要想向I/O设备读写数据，需要地址来引导。这就提出了**I/O接口地址**。这是为I/O设备对应的I/O接口电路分配的地址，这就完成了从接口电路进行读写操作到从接口地址进行读写操作的转化。

> 这一点跟存储器的转化是相通的。

这里有一个问题，如何对I/O接口编址？

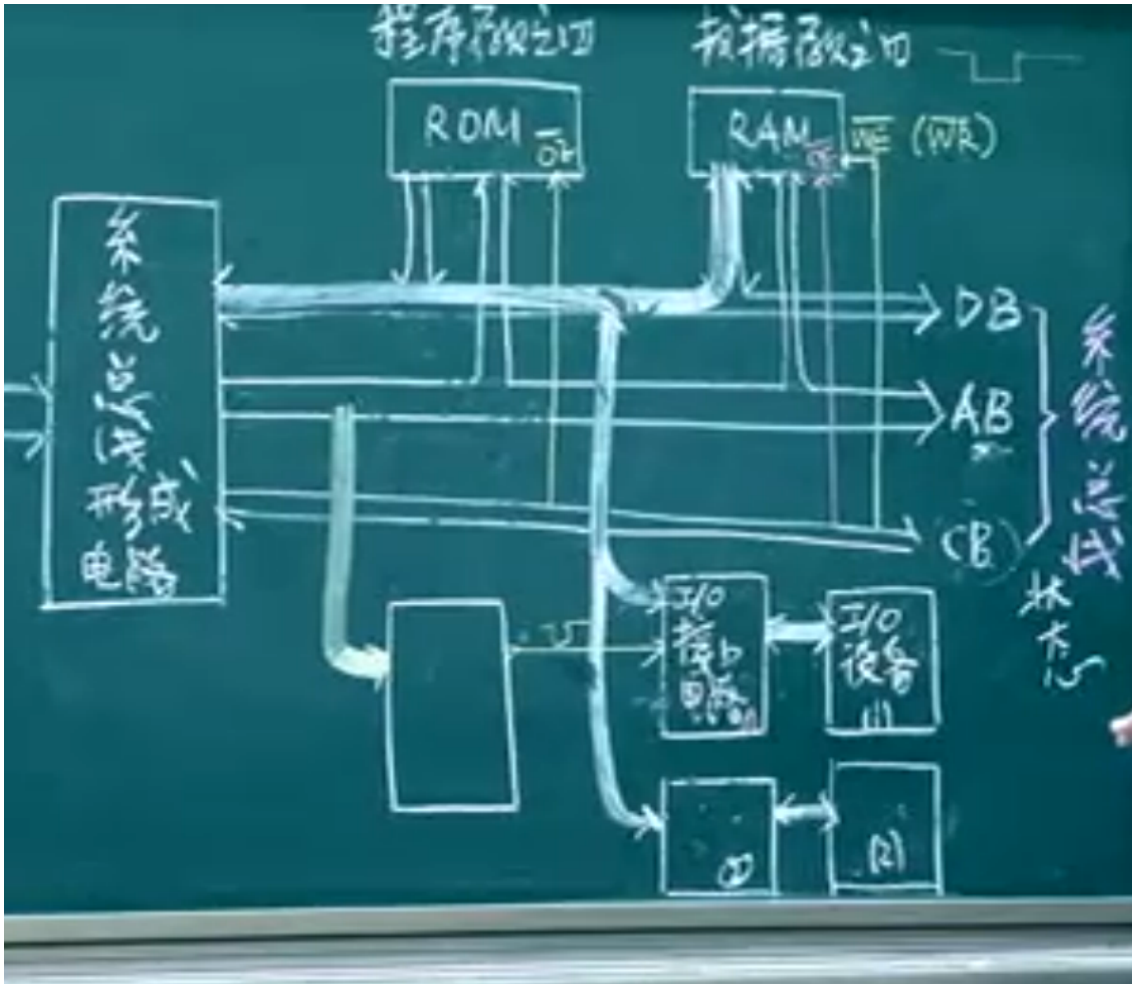
- 8086系统中，是通过地址线来统一编址。即既编址存储器地址，也编址I/O接口。这时读写存储器和I/O的指令相同。

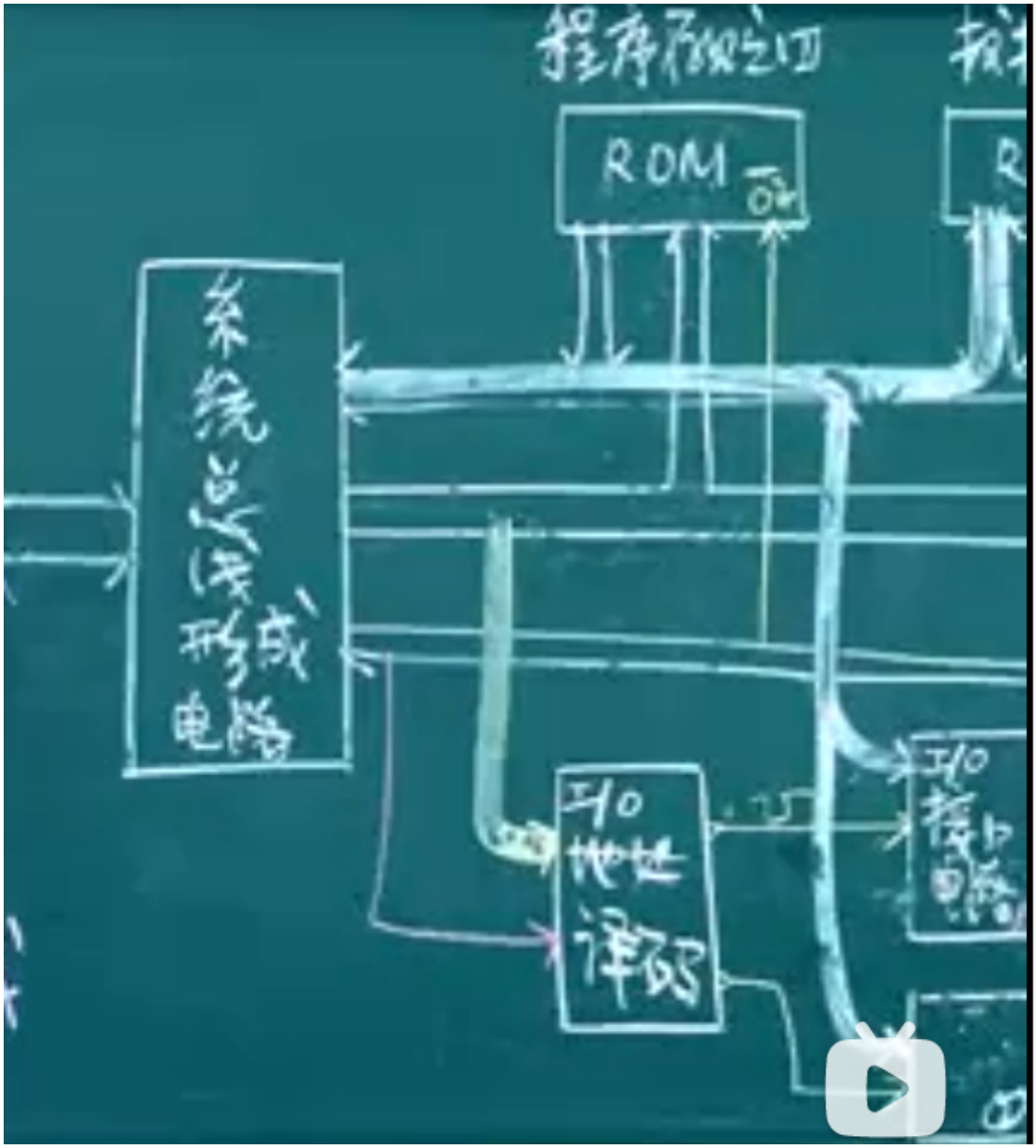
  > 当然，对于存储器地址空间和I/O地址空间，也有很多微处理器采用两个独立地址空间。这时存储单元地址和I/O端口的地址值可能相同，所以需要采用存储器读写信号和I/O读写信号来区分。这样CPU读写存储器和I/O的指令是不同的。

- 对于一个I/O接口来说，地址码通常只有一个，这里需要设计一个电路来完成从16根地址线（16位二进制）+控制线上的有效位 到一个控制命令（高低电平0/1）的转化。

  其实也就是一个I/O地址译码器。这是数电/数字逻辑课的内容。

  > 存储器相应也有类似的结构。


程序存储空间
数据存储空间
ROM
RAM
WE (WR)
系统总线形成电路
DB
AB
CB
系统总线
I/O接口电路
I/O设备
R0
R1



> 这部分课程后续会再详细介绍。

2.5 有关微机系统

上述硬件结构还不能工作，只是一个微型计算机，而不是微型计算机系统。需要配上I/O设备、电源、软件等，才能正常工作。

2.6 存储器简介

前面已经提到过很多次存储器，这里再简单整理下。

存储器有几个要素，地址单元大小，地址号码。一个地址单元内是一个字节8bit=1byte。而地址号则决定地址单元的逻辑位置，地址线编码了地址号，CPU通过地址线寻址存储器。所以大空间的存储器也要求着地址线条数要多。

> 地址线和数据线宽度的区别：
>
> - 数据线的宽度决定一次传送的数据规模。
> - 8086CPU数据线16位，所以一次可以取两个字节的内容。
>
>   因此，两个相邻的地址单元（字节），可以组成一个字单元（也就是16位）。字分为高八位和第八位。

> - **大端字节序和小端字节序**。而8086系统是高字节存放在高地址空间，低字节存放在低地址空间。
> - 读/写一个字时，读/写的地址是低字节的地址号。

设计存储器时我们需要对芯片进行相关组织而达到相关存储要求。

3. 数制码制快速复习

这部分各个课（C、数字逻辑、计组）都讲，实在重复。

3.1 数制

- 程序设计中的二进制->十进制的一种方法：
  - 除十取余法

    二进制除十的二进制，然后余数变成十进制，这样就是十以内的转化
  - 比较法

> 后面汇编部分会实现这里的简单转换算法

- 程序设计中的十进制->二进制：

  比如要从键盘输入95，CPU中9*10得到的结果就是二进制，再加5，得到的就是二进制95.

> 这个想法根本原因是：CPU只认识二进制。
>
> `mov AL, 96`
>
> `add AL, 89`
>
> 中的96、89就是二进制的形式被CPU识别的。

3.2 二进制算术运算

首先要搞清楚运算的对象是8位二进制（字节）还是16位二进制（字）。

- 在现在大多数情况的汇编语言中数字是十六进制的, **如果最开头的一位是A~F, 仍然要求在前面加零**.
- 处理器状态字寄存器/标志寄存器：

（2）微处理器状态字 PSW（processor state word）：它是一个 16 位的寄存器，一共设定了 9 个标志位，其中 6 个标志位用于反映 ALU 前一次操作的结果状态，另 3 个标志位用于控制 CPU 操作。具体位置如图 2.5 所示。

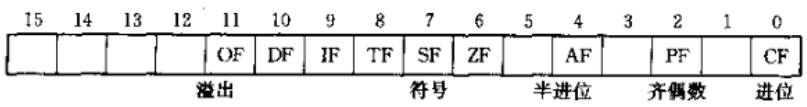


图 2.5 PSW 中的标志位

只有**运算**才能影响标志寄存器。包括下面的逻辑运算。

> 补充：
>
> - PF是奇偶校验，只看低八位，如果1个数为偶数则为1，奇数为0.可以用于数据串行通信过程中的奇偶校验的硬件实现。
> - AF是辅助进位，管理四位间的进位借位，有进位为1。

3.3 二进制逻辑运算

- 逻辑与 AND：
  - 可以用于清零某些位，更宽泛说可以规范化数据（如操作系统中查页表时的操作）
- 逻辑或 OR：
  - 同上，规范化数据，将某些位置为1而其他位不变，置1的位为1即可。
- 异或 XOR：
  - 这里我经常记错，两运算对象对应位不同则为1，相同为0.
  - 可以用于某些位取反而其他位不变，取反的位置为1即可。

3.4 表示范围和溢出问题

- 无符号数表示范围：
  - 8位：00H~FFH
  - 16位：0000H~FFFFH
- 有符号数表示范围：
  - 机器数表示法：0--正，1--负数
- 有符号数原码：
  - 符号位+正常二进制表示
  - 缺点，+0和-0的机器数表示并不同，这在计算机中是矛盾的
  - 表示范围：
    - 8位：+127~+0，-0~-127
    - 16位：+32767~+0,-0~-32767
- 有符号数反码：
  - 正数就是原码
  - 负数：符号数+原码其余位取反
- 有符号数补码：
  - 正数就是原码
  - 负数：反码+1==符号位不变，原码按位取反+1
  - 表示范围：
    - 8位，+127~-128
    - 16位：+32767~-32768
  - 计算机中用的就是补码

> 最近遇到了一个C程序上的问题，从端口中拿出的状态码是char类型的，但是用char接收会溢出，所以向上适用int类型就会正常。

3.5 补码的运算

公式1：[X+Y]补 = [X]补+ [Y]补

公式2：[X-Y]补 = [X]补- [Y]补

公式3：[X-Y]补 = [X]补+ [-Y]补

- 看结果时不看最高位溢出的（第九位），只看八位
- 公式三由公式2发展而来，思路更简单，并且节省了硬件电路。

由补码的运算结果（补码）求原码：对补码求补码（符号位不变，取反+1）。所以有公式4.

公式4：[X]=[X补]补

在上面公式3的情境中，需要实现一个操作：已知[Y]补，求[-Y]补。这时需要包括符号位在内全部取反，再加1。

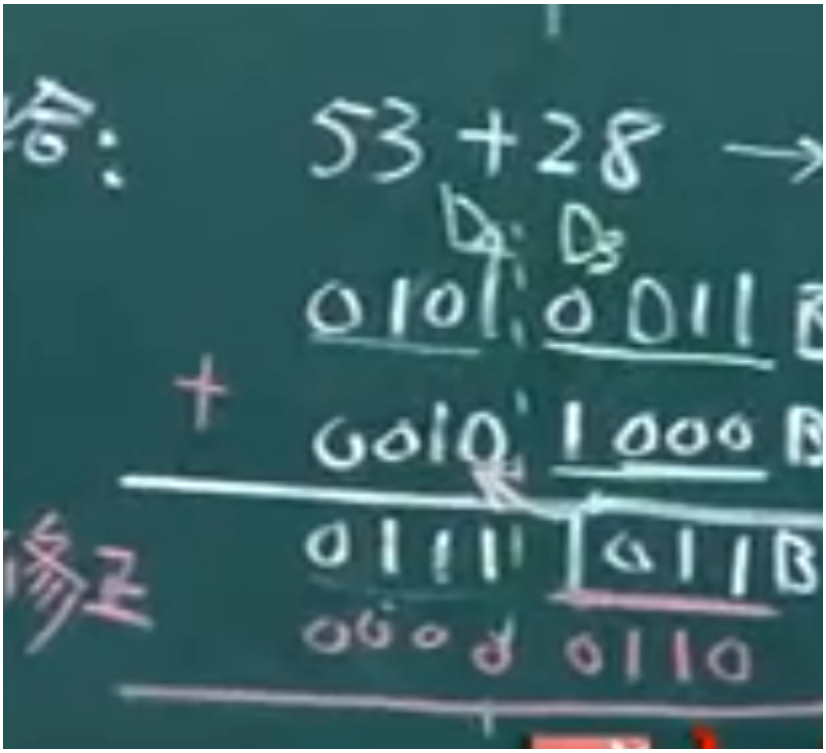
3.6 BCD码

用四位二进制描述0~9十个十进制数

- 对应关系
  - 0000---0
  - 0001---1
  - ……
  - 1001---9
- 也称8421码
- 更复杂的表示
  - 组合BCD数，两个BCD码放在存储器的一个字节中。
  - 分离BCD数，两个BCD码分别放在两个字节的低四位，高四位为0.
  - 加减法有以上两种表示，而乘除只有分离BCD数。
- BCD码的运算：主要是换算进位问题。

  如下图，53+28的运算中需要判断是否超过9，如果超过9而未达16，需要+6强制转化。

  > 这时就用到了AF。



程序实现中

```
mov AL,53
add AL,28
DAA;如果设计BCD计算，+6调整
;如果是分离BCD,需要AAA指令
```

3.7 字符表示 ASCII码

- '0'~'9'：30H~39H
    - 字符到数字转化：-30H
- 'A'~'F'：41H~46H
    - 字符到数字转化：-37H
- 'a'~'f'：61H~66H
    - 大小写转化：

```
;假设将AL中转换为大写
;大写和小写仅在第五位有区别
;;大写：0100_xxxx
;;小写：0110_xxxx
AND AL,0DFH
;;由于第七位必然为0
AND AL,5FH


;AL中转换为小写
OR AL,20H
```