汇编语言程序设计

新型计算机研究所

董小社

西一楼A段413室

Email: xsdong@xjtu.edu.cn

群名称：汇编语言2025年春
群　号：935095373

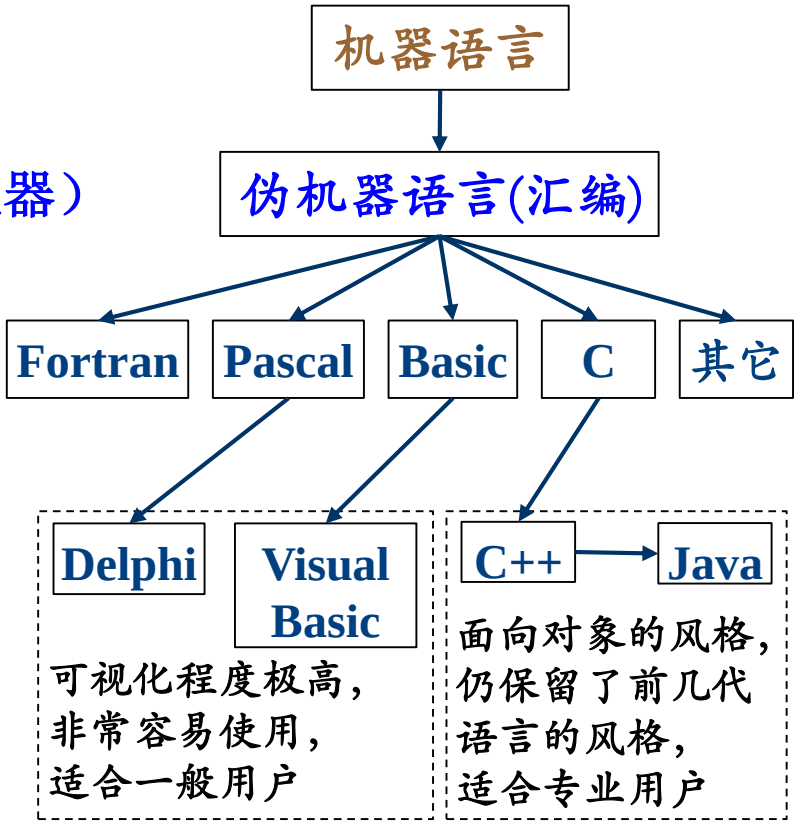
课程QQ群：汇编语言2025年春（935095373）

思源学堂：http://syxt.xjtu.edu.cn/

请将昵称修改为：姓名-学号

计算机语言的发展

- 第一代语言：机器语言（二进制编码）
- 第二代语言：汇编语言（符号式，面向机器）
- 第三代语言：高级语言、算法语言（接近自然语言，面向过程）
- 第四代语言：非过程化语言（面向目标、面向对象）
- 第五代语言：智能性语言（具有一定的智能，抽象问题求解）



易用：接近人类自然语言、图形化、底层细节透明、面向对象和问题描述等

开发高效：易编程，开发程序效率高等，复杂应用实现容易等，多线程并行有效利用硬件资源

什么是汇编语言？

- 机器语言：机器指令的集合。以二进制形式的指令组成的指令集合，是计算机唯一能够直接识别和处理的语言

  例如：10001001 11011000; 将寄存器BX的内容送到寄存器AX。

  缺点：程序编写、阅读、改错很不方便

  - 机器语言描述的程序称为目标程序（可执行程序），只有目标程序CPU才能直接执行

- 汇编语言：机器语言的符号化表示。面向机器的语言，用简单且容易记忆的符号（助记符号）来代替机器语言中“0”、“1”的一种程序设计语言

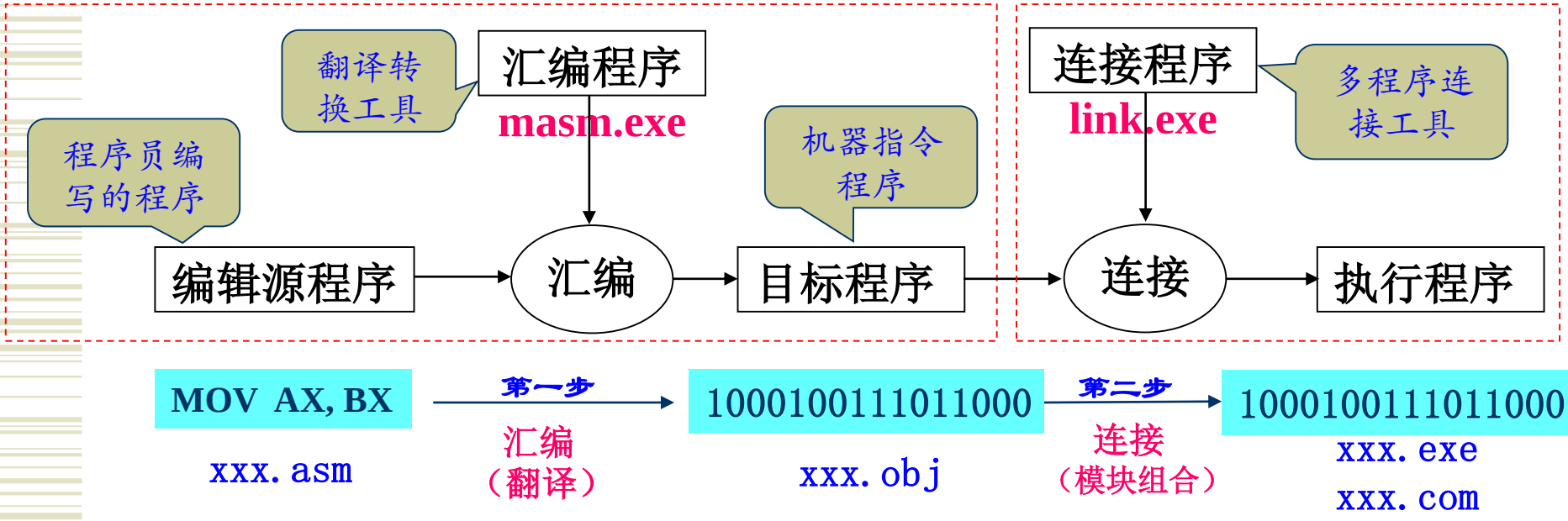
例如：	机器语言指令	汇编语言指令
	10001001 11011000	MOV AX, BX

  汇编语言的执行语句与机器语言的指令是一对一的关系

  - 现代计算机编程中使用汇编语言主要是为了某些情况下直接控制计算机硬件，以获得高级语言无法实现的功能和性能

用汇编语言编写程序、执行过程

- 由于计算机不能够直接识别符号语言，所以汇编语言编写的源程序也必须用汇编程序汇编（直接翻译）成机器语言后才能执行



为什么要学习汇编语言？

- 程序设计语言体系中一类重要语言
- 理解计算机的基本系统结构
- 能够学习到处理器是如何工作的
- 探究数据和指令的内部表述
- 能够创建小巧高效的程序
- 允许程序员绕过高层语言的限制编程
- 有些工作必须用汇编语言完成

当需要将资源效能发挥到极致并实现特殊优化需求时，开发者就不得不求助于汇编语言。例如：DeepSeek在训练DeepSeek-V3时，使用GPU的汇编语言PTX实现最大性能

- ✓ 编写嵌入式程序、实时应用、驱动程序等
- ✓ 通过类调用绕过高级语言限制直接使用低层硬件

汇编语言程序设计与后续课程关系

- **硬件课程：**
  - 计算机组成原理：如何用硬件实现机器指令集，即汇编语言指令功能
  - 微机原理与接口技术：使用汇编语言编写硬件驱动控制程序等
  - 计算机系统结构（重要课程）：CPU内部结构、机器指令、存储系统和IO系统优化设计、评价方法
- **软件课程：**
  - 编译原理：如何将 高级语言程序模块 通过编译优化转换成 机器指令程序模块（C/F/J/P→汇编语言→机器语言）
  - 操作系统：基于汇编语言设计实现对计算机系统资源管理、任务管理
  - 高级语言程序设计：使用汇编语言程序对低层的直接控制（课本第13章模块化设计）

使用教材及其他

- 教材、主要参考书籍（http://jiaocai.lib.xjtu.edu.cn/）
  - 教材：《80x86汇编语言程序设计》，沈美明，温冬婵 编著，清华大学出版社
  - 主要参考书籍：
    - 《汇编语言》，王爽著，清华大学出版社
    - 《Intel汇编语言程序设计》，Kip R.Irvine 著，温玉杰 等译，电子工业出版社
    - 《IBM PC Assembly Language and Programming》, Peter Abel, (影印版，清华大学出版社)
- 答疑
  - 时间：线上班级QQ群，线下办公室（上班时间)
  - 地点：西一楼A段413房间（董小社）
    西一楼B段811房间（张兴军）
    西一楼A段415房间（陈衡）

本课程学习方法

1. 熟练掌握计算机硬件组织结构、功能和工作原理
   - 处理器：寄存器、运算器、PSW等工作原理
   - 存储器（内存）：数据存储和访问方式
   - 输入/输出：工作原理和控制方式
2. 熟记常用指令、数据表示、DOS功能中断调用
3. 掌握非常用指令、数据表示方式
4. 学习汇编语言基本原理、程序设计方法
5. 在实践中总结、创新编程优化技巧

注：汇编语言程序设计课程要求掌握如何使用硬件资源；
计算机组成原理课程要求掌握如何设计实现硬件；
计算机系统结构课程要求掌握优化设计和评价方法

第1章　基础知识

1.1 进位计数制与不同基数的数之间转换

1.2 二进制数和十六进制数运算

1.3 计算机中数和字符的表示

1.4 几种基本的逻辑运算

1.1 计算机中的数

1. 汇编程序中常用的数制表示

二进制数：计算机硬件唯一识别和使用的数制

以2为基的数制表示法，数由2个数字构成（**0**、**1**），二进制数后缀为B。　　如10110111B　　IO的控制、状态检测时常用

十进制数：人类自然语言中常用的数制，编程阅读最方便

以10为基的数制表示法，数由10个数字构成（**0~9**），十进制数后缀为D。　　如1945D　　运算的数值时常用

十六进制数：程序设计中便于和机器数对照、转换，编程阅读方便的数制，是汇编语言编程使用的主要数制

以16为基的数制表示法，数由16个数字构成[**0~9**、A（10）、B（11）、C（12）、D（13）、E（14）、F（15）]，十六进制数后缀为H。　　如18ADH　　内存地址时常用

十进制数的特点

特点：

(1) 基数为10：有10个不同的数字符号

0、1、2、3、4、5、6、7、8、9

(2) 逢10进位，借1当10

- 十进制一般表达式

$$D = D_{n-1} \cdot 10^{n-1} + D_{n-2} \cdot 10^{n-2} + \cdots + D_1 \cdot 10^1 + D_0 \cdot 10^0 + D_{-1} \cdot 10^{-1} + D_{-2} \cdot 10^{-2} + \cdots + D_{-m} \cdot 10^{-m}$$

$$= \sum_{i=-m}^{n-1} D_i \cdot 10^i$$

二进制数的特点

- 特点：

  (1)基数是2：具有两个不同的基本符号0、1

  (2)逢2进1，借1当2

- 二进制一般表达式：

$$B_{n-1}\cdot 2^{n-1}+B_{n-2}\cdot 2^{n-2}+\cdots+B_1\cdot 2^1+B_0\cdot 2^0$$
$$+B_{-1}\cdot 2^{-1}+B_{-2}\cdot 2^{-2}+\cdots+B_{-m}\cdot 2^{-m}$$

- 二进制与十进制关系

$$D=\sum_{i=-m}^{n-1}B_i\cdot 2^i$$

二进制计量单位

- ***比特***：bit，或称位元，简称位，0或1；以“b”表示；最小单位
- ***字节***：byte，位组，8个bit；以“B”表示，一个字符用一个字节表示
- ***字***：word，2个字节，与计算机中的字长是两个概念
- $2^8=256$, $2^{10}=1024=1K$, $2^{16}=64K$, $2^{20}=1M$, $2^{24}=16M$, $2^{30}=1G$, $2^{32}=4G$, …
- *1KB=1024B*, *1MB=1024KB*, *1GB=1024MB*; *1TB=1024GB*

为什么需要十六进制数

① 便于编程与阅读：二进制的阅读、书写和记忆不方便

- 计算机用十进制数？
  - 如果不转换，用BCD码(Binary Code Decimal，二进制编码的十进制数）表数效率太低，处理困难
    - 0000～1001分别表示8421 BCD码的0～9
    - 1010～1111没有用，浪费6个编码，即37.5%
- 面向机器硬件编程时用十进制数？
  - 十进制与二进制无直接对应关系，转换困难、不易理解
  - 2^n作为基数的数制与二进制数转换方便、处理方便，
    - 八进制(000～111)、十六进制(0000～1111)

BCD:15D=0001 0101B

需要8位

② 十六进制数与计算机中数据存储、处理的单位长度适用

- 基本单位：一个二进制位（bit）
- 常用字符单位：8位二进制数组成的一个字节（byte）
- 计算机字长：字节的整数倍，8位、16位（字，Word）、32位、64位二进制数
- 1位十六进制数对应4位二进制数
- 1位八进制数对应3位二进制数

F=1111 仅需要4位,且转换方便

几种常用进位记数制的基数和数码

数制	基数	各位数码表示
二进制 Binary	**2**	**0, 1**
十进制 Decimal	**10**	**0, 1, 2, 3, 4, 5, 6, 7, 8, 9**
十六进制 Hexadecimal	**16**	**0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**

汇编程序中更易理解和表示的数值，一般使用方式/用途：

- 按位读写数据或控制：二进制
- 数据：十进制/十六进制，　　字符/字符串：十六进制
- 存储单元地址/内容：十六进制

十进制数 、二进制数、八进制数和十六进制数之间的对应关系

十进制(D)	二进制(B)	八进制(O)	十六进制(H)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

2. 不同进位计数制之间数据转换

- **CPU处理的是二进制数**
  - 写程序时，需要把十进制、二进制数转换为十六进制数
  - 阅读程序时，需要将二进制、十六进制转换为十进制数

- **数据转换：** 把一种进制数转换为另一种进制的数，其实质是进行基数的转换。基数转换是依据两个有理数相等，其整数部分与小数部分分别相等的原则。

- **转换方法：** 转换时，其整数部分与小数部分应分别进行转换，将转换后的结果合并，整数部分与小数部分之间用小数点隔开，就得到相应的转换结果。

(1) 二进制数转换为十进制数的转换规则

“按权值相加”。也就是说，只要把二进制数中数位是“1”的那些位的权值相加，其和就是等效的十进制数。

$$D=\sum_{i=-m}^{n-1} B_i \cdot 2^i$$

1101B = 1×2³ + 1×2² + 0×2¹ + 1×2⁰ = 13D

(2) 十六进制转换为十进制数的转换规则

$$D=\sum_{i=-m}^{n-1} h_i \cdot 16^i$$

(2) 十进制数转换为二进制数

整数和小数部分分别进行转换，转换结束后将整数转换结果写在左边，小数转换结果写在右边，中间点上小数点。

- 两种基本方法：降幂法、除法

十进制数转换为二进制数

方法1：降幂法

$$D = B_{n-1}\cdot 2^{n-1} + B_{n-2}\cdot 2^{n-2} + \cdots + B_1\cdot 2^1 + B_0\cdot 2^0 + B_{-1}\cdot 2^{-1} + B_{-2}\cdot 2^{-2} + \cdots + B_{-m}\cdot 2^{-m}$$

整数部分转换规则：

- 写出要转换的十进制数
- 写出所有小于此数的二进制各位权值
- 十进制数减去二进制权值，权值由大到小
  - 如够减，相应二进制位记1；如不够减，相应二进制位记0
- 不断反复，直到该数为0

小数部分的转换规则：

同整数部分

例： 27D = ? B

27	11	3	3	1
−↓	−↓	−↓	−↓	−↓
16	8	4	2	1
1	1	0	1	1

∴ 27D = 11011B

十进制数转换为二进制数

方法2：除法

$$D = B_{n-1}\cdot 2^{n-1} + B_{n-2}\cdot 2^{n-2} + \cdots + B_1\cdot 2^1 + B_0\cdot 2^0 + B_{-1}\cdot 2^{-1} + B_{-2}\cdot 2^{-2} + \cdots + B_{-m}\cdot 2^{-m}$$

$$D = \left[\left(B_{n-1}\cdot 2^{n-2} + B_{n-2}\cdot 2^{n-3} + \cdots + B_1\cdot 2^0\right)\cdot 2 + B_0\right] + \left[\left(B_{-1} + B_{-2}\cdot 2^{-1} + \cdots + B_{-m}\cdot 2^{-m+1}\right)\cdot 2^{-1}\right]$$

整数部分转换规则：

- 将十进制整数用基数2连续去除，直到商为0为止；
- 将每次除得的余数反向排列，就可得到十进制数整数部分的转换结果。
- 反向排列：最后得到的余数排在前边，作为结果的最高位，最先得到的余数排在后边，作为结果的最低位

小数部分的转换规则：

- 将十进制数的小数部分用基数2连续去乘，直到小数部分为0或达到精度为止；
- 将每次所得的乘积的整数部分正向排列，就可得到十进制小数的转换结果。
- 正向排列：最先得到的整数为结果的最高位，最后得到的整数为结果的最低位

例 N = 117.8125D

$$D = B_{n-1}\cdot 2^{n-1} + B_{n-2}\cdot 2^{n-2} + \cdots + B_1\cdot 2^1 + B_0\cdot 2^0 + B_{-1}\cdot 2^{-1} + B_{-2}\cdot 2^{-2} + \cdots + B_{-m}\cdot 2^{-m}$$

$$D = \left[\left(B_{n-1}\cdot 2^{n-2} + B_{n-2}\cdot 2^{n-3} + \cdots + B_1\cdot 2^0\right)\cdot 2 + B_0\right] + \left[\left(B_{-1} + B_{-2}\cdot 2^{-1} + \cdots + B_{-m}\cdot 2^{-m+1}\right)\cdot 2^{-1}\right]$$

- 整数部分：117D

117/2=58　　a_0=1

58/2=29　　a_1=0

29/2=14　　a_2=1

14/2=7　　a_3=0

7/2=3　　a_4=1

3/2=1　　a_5=1

1/2=0　　a_6=1

117D = 1110101B

- 小数部分：0.8125D

0.8125×2=1.625　　b_0=1

0.625×2=1.25　　b_1=1

0.25×2=0.5　　b_2=0

0.5×2=1.0　　b_3=1

0.8125D = 0.1101B

N=117.8125D=1110101.1101B

十进制数转换为十六进制数?

(3)二进制数转换为（八进制）十六进制数

- 将二进制数以小数点为界，向左、向右分别按（3位）4位一组划分，不足（3位）4位的部分用“0”补足（整数部分左补0，小数部分右补0），将每一组数写成一位对应的（八进制）十六进制数，就可得到转换结果。

例如：1110101.11B=？H

0111 0101 . 1100

7 5 . C

∴ 1110101.11B=75.CH

十进制(D)	二进制(B)	八进制(O)	十六进制(H)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

(4)（八进制）十六进制数转换为二进制数

- 将十六进制数以小数点为界，向左、向右分别展开为（3位）4位二进制数，就可得到转换结果。

例如：EA.11H=？B

E	A	.	1	1
1110	1010		0001	0001

∴ EA.11H=11101010.00010001B

十进制(D)	二进制(B)	八进制(O)	十六进制(H)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.2 二进制数和十六进制数运算

- 二进制数加法/减法规则

二进制加法规则：

0 + 0 = 0

1 + 0 = 1

0 + 1 = 1

1 + 1 = 0——向高位进位为1（逢2进1）

二进制减法规则：

0 – 0 = 0

1 – 0 = 1

0 – 1 = 1 ——向高位借1（借1当2）

1 – 1 = 0

- 十六进制数
  - 自学

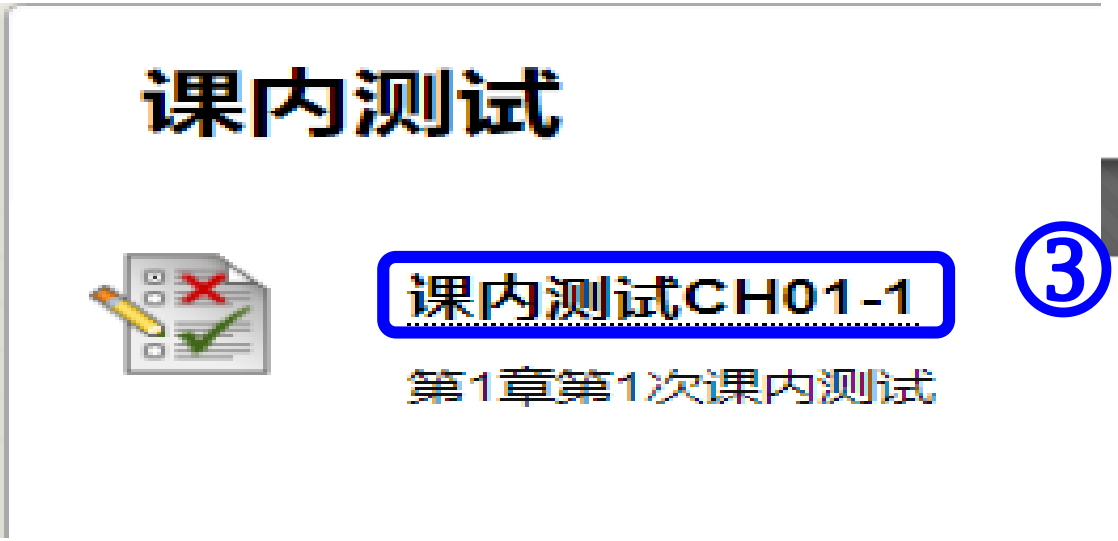
```
  1 0 1 1 0 1 1 1.1 1
+ 0 0 0 0 0 0 0 1.0 0
-------------------
  1 0 1 1 1 0 0 0.1 1
```

课内测试使用说明

1、登录思源学堂， http://syxt.xjtu.edu.cn

2、点击左侧栏目的 **课内测试**

3、点击 **课内测试CH01-1**



4、点击 开始，启动答题


开始： 课内测试CH01-1
说明
描述
第1章第1次课内测试
强制完成
本测试可保存并可稍后继续。
多次尝试
此测试允许进行多次尝试。
单击开始以开始: 课内测试CH01-1。单击取消返回。
④
选择"开始"可开始。选择"取消"可退出。
取消
开始

课内测试CH01-1

1、请填写正确答案“ASM” （10分）；

2、将十进制数转换为二进制数常用的方法是[填空]和[填空]。 （10分）

请进入思源学堂答题

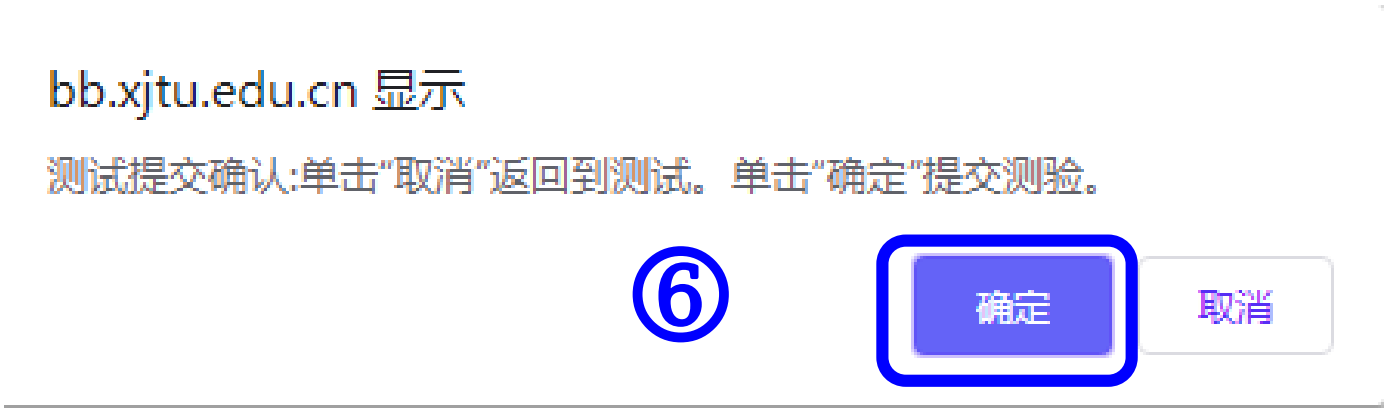
课内测试CH01-1

1、请填写正确答案“ASM” （10分）；

2、将十进制数转换为二进制数常用的方法是[填空]和[填空]。 （10分）

5、完成后，点击 保存并提交

6、提交前最后确认，点击 **确定**



7、提交后信息显示，点击 **确定**可查询得分情况及正确答案

1.3 计算机中数和字符的表示

计算机中数、字符和地址都是用二进制编码表示，但如何表示？

1.3.1 数的补码表示

1.3.2 补码的加法和减法

} 运算数据的表示方法

1.3.3 无符号整数

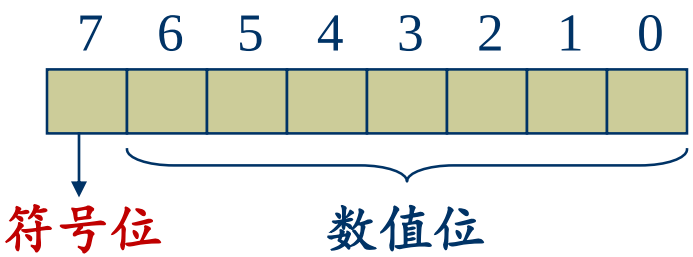
} 地址的表示方法

1.3.4 字符表示法

} 字符串的表示方法

数（机器数）的表示

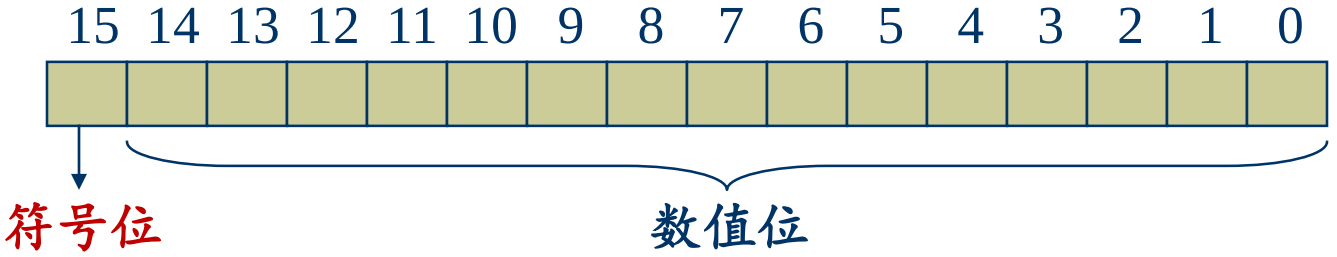
- 计算机中的数用二进制表示，数的符号也用二进制表示。
  - 假设机器字长n为8位



符号位=0 表示正数

符号位=1 表示负数

  - 假设机器字长n为16位



- 机器字长：指参与运算的数的基本位数(二进制)，标志着计算精度，一般是字节的整数倍（8位、16位、32位、64位等）。

机器数常用表示法 —— 原码、 反码、 补码

(Sign-Magnitude, Ones' complement, Two's complement)

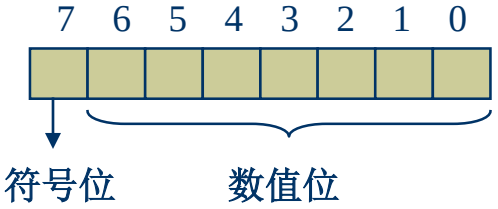
1.3.1 数的补码表示法

1. 原码表示法

（1）原码表示法： 将数的真值形式中的正（负）号，用代码0(1)来表示，数值部分用二进制来表示。符号 + 绝对值

正数：符号位为0，后面的n-1位为数值部分

负数：符号位为1，后面的n-1位为数值部分



（2）原码的特点

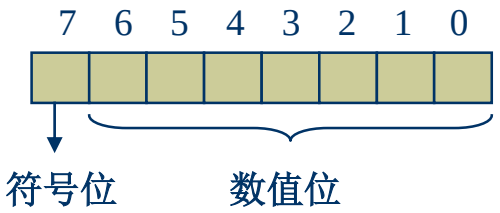
- “0”的原码有两种表示法

  $[+0]_{原}=00000000B$, $[-0]_{原}=10000000B$

  0的表示不唯一，编码效率低、处理复杂

- n位二进制原码所能表示的数值范围为： $-(2^{n-1}-1)\sim(2^{n-1}-1)$
- 最高位为符号位

2. 反码表示法



- 正数: 反码同原码
- 负数: 反码数值位与原码相反
- 例： n=8bit

$[+5]_{反码}$ = 0 0000101 = 05H

$[-5]_{反码}$ = 1 1111010 = FAH　　$[-5]_{原码}$ = 1 00000101 = 85H

$[+0]_{反码}$ = 0 0000000 = 00H

$[-0]_{反码}$ = 1 1111111 = FFH

0的表示不唯一，编码效率低、处理复杂

3. 补 码

（1）补码表示法

$[X]_{补}=M+X$ （$M=2^n$ ，当$n=8$ 时$M=256$）

结果超过n位，扔掉

$[-1]_{原}=10000001B$

$[-1]_{补}=256+(-1)=11111111B$

$[+1]_{原}=00000001B$

$[+1]_{补}=256+(+1)=00000001B$

$[-127]_{原}=11111111B$

$[-127]_{补}=256+(-127)=10000001B$

$[+127]_{原}=01111111B$

$[+127]_{补}=256+(+127)=01111111B$

$2^8 = 256D = 100000000B$

$- \quad 00000001B$

$11111111B$

$2^8 = 256D = 100000000B$

$+ \quad 00000001B$

$100000001B$

$[正数]_{补}$ 与 $[正数]_{原}$ 相同

（2）补码表示规则：

- 正数的补码：符号+绝对值（与正数的原码相同）

$[+1]_{补码}$ = 0000 0001 = 01H

$[+127]_{补码}$ = 0111 1111 = 7FH

$[+0]_{补码}$ = 0000 0000 = 00H

- 负数的补码：负数 X 用 $2^n-|X|$ 表示

$[-1]_{补码}$ = 2^8-1 = 1111 1111 = FFH

$[-127]_{补码}$ = 2^8-127 = 1000 0001 = 81H

一种简单转换方法：

（1）写出与该负数相对应的正数的补码

（2）按位求反

（3）末位加1

$[-1]_{补}$=?

$[+1]_{补}$=0000 0001

1111 1110

$[-1]_{补}$=1111 1111

例： 机器字长8位，$[-46]_{补码}$ = ？

$[+46]_{补码}$ = 0010 1110

按位求反

1101 0001

末位加1

$[-46]_{补码}$ = 1101 0010 = D2H

补码的符号扩展问题

- 指一个数从位数较少扩展到位数较多时应该注意的问题
  - 8位扩展到16位，16位扩展到32位，等
- 补码表示的扩展规则
  - 正数：前边补0
  - 负数：前边补1

  } **扩展符号位，操作简单**

机器字长8位，$[-46]_{补码}$ = 1101 0010B = D2H

机器字长16位，$[-46]_{补码}$ = 1111 1111 1101 0010B = FFD2H

- 原码表示的扩展规则？

（2）补码的特点

补码效率高；处理规则简单，硬件实现简单

计算机低成本、高性能

- “0”的补码表示唯一

$[+0]_{补}=[-0]_{补}=00000000B$

- 补码运算时符号位无需单独处理
- 采用补码运算时，减法可用加法来实现

$[13-10]_{补}=[13]_{补}+[-10]_{补}=00001101+11110110$

$=00000011B=[+3]_{补}$

（有进位，自动丢失，符号位为0结果为正）

$[10-13]_{补}=[10]_{补}+[-13]_{补}=00001010+11110011$

$=11111101B=[-3]_{补}$

（无进位，符号位为1，结果为负）

- 符号扩展问题简单

补码的表数范围

n位补码的表数范围： $-2^{n-1} \le N \le 2^{n-1}-1$

n=8　　$-128 \le N \le 127$　　2^8个数

n=16　$-32768 \le N \le 32767$　　2^{16}个数

比原码和反码多表示一个数，表数效率也高一点

1.3.2 补码的加法和减法

求补运算：对一个二进制数按位求反后在末位加1的运算

注意：求补运算仅指对一个数的求补运算

补码表示的数具有以下特点：

$$[X]_{补} \xrightarrow{求补} [-X]_{补} \xrightarrow{求补} [X]_{补}$$

$$[117]_{补} \xrightarrow{求补} [-117]_{补} \xrightarrow{求补} [117]_{补}$$

补码加法和减法的规则

$$[x + y]_{补} = [x]_{补} + [y]_{补}$$

$$[x - y]_{补} = [x]_{补} - [y]_{补}$$

$$= [x]_{补} + [-y]_{补}$$

- 补码减法可转换为补码加法
- 不必判断数的正负，符号位一起参加运算，能自动得到正确结果

运算器设计简单，不需要减法运算器

举例说明补码运算

已知：**X= -27D**

Y= +29D

求：$[X + Y]_{补} = ?$

$[X - Y]_{补} = ?$

解：运算步骤

求$[X + Y]_{补}$：

X= -27D
Y= +29D

1）求出$[X]_{补}$，$[Y]_{补}$

$[X]_{补}$ = 11100101B ， $[Y]_{补}$ = 00011101B

2）求出 $[X + Y]_{补}$

$[X + Y]_{补} = [X]_{补} + [Y]_{补}$

= 11100101 + 00011101

= 00000010B

```
  11100101B
+ 00011101B
-----------
 100000010B
```

求$[X - Y]_{补}$：

$[X]_{补}$ = 11100101B ，

$[Y]_{补}$ = 00011101B，[-Y]补 = 11100011B

1）$[X - Y]_{补} = [X]_{补} - [Y]_{补}$

= 11100101 - 00011101

= 11001000B

```
  11100101B
- 00011101B
------------
  11001000B
```

2）$[X - Y]_{补} = [X]_{补} + [-Y]_{补}$

= 11100101 + 11100011

= 11001000B

```
  11100101B
+ 11100011B
------------
 111001000B
```

1.3.3 无符号整数

- 当程序处理的数全是正数时，保留符号位就没有意义了，此时应该采用无符号数表示
- 表数范围

  16位无符号数的表示范围

  $0<=N<=2^{16}-1=65535$

  8位无符号数的表示范围

  $0<=N<=2^{8}-1=255$

- 一般用途：表示地址的数；双精度数的低位字

如果机器字长32位,那么双精度数是64位

表数范围

表数范围：给出n位二进制整数的表数范围

(1）无符号整数的范围

n位二进制数N能够表示的无符号数的范围

$0 \sim (2^n-1)$； 2^n个数

8位二进制数所能表示的无符号整数的范围是

$0 \sim 255$； $2^8=256$个数

16位二进制数所能表示的无符号整数的范围是

$0 \sim 65535$； $2^{16}=65536$个数

表数范围

（2）采用补码表示的带符号整数的表数范围

8位二进制数所能表示的带符号整数的范围是

-128 ~ 127；

16位二进制数所能表示的带符号整数的范围是

-32768 ~ +32767；

n位二进制数N能够表示的带符号数的范围

$$-2^{n-1} \sim 2^{n-1}-1$$

+127	0111 1111
+126	0111 1110
...	...
+2	0000 0010
+1	0000 0001
0	0000 0000
−1	1111 1111
−2	1111 1110
...	...
−126	1000 0010
−127	1000 0001
−128	1000 0000

n位二进制补码的表数范围

十进制	二进制	十六进制	十进制	十六进制
	n=8		n=16	
+127	0111 1111	7F	+32767	7FFF
+126	0111 1110	7E	+32766	7FFE
…	…	…	…	…
+2	0000 0010	02	+2	0002
+1	0000 0001	01	+1	0001
0	0000 0000	00	0	0000
-1	1111 1111	FF	-1	FFFF
-2	1111 1110	FE	-2	FFFE
…	…	…	…	…
-126	1000 0010	82	-32766	8002
-127	1000 0001	81	-32767	8001
-128	1000 0000	80	-32768	8000

- 指令中数据用补码表示，超出表数范围，汇编时出错
  - MOV AH, -129（超出表数范围）
- 运算结果超出表数范围， CPU会产生溢出错误中断
  - 假设AH=127、AL=2，ADD AH, AL（**结果溢出**）

十进制数在机器中表示

十进制数在机器中通常采用BCD码存储、处理

- BCD码是一种用二进制编码的十进制数，即用4位二进制形式（0000B-1001B）来表示一位十进制数（0-9），但每4位二进制数（1位十进制数）之间的进位又是十进制的形式

57D = 0101 0111 BCD

BCD码
- 压缩的BCD码（用4位二进制表示）
  0101 0111
- 非压缩的BCD码（用8位二进制表示，前面4位为0000）
  00000101 00000111

计算机中基本存储单位是字节

1.3.4 字符表示法

字符及字符串通常用ASCII码存储、处理

- 计算机处理的信息并不全是数据,有时需要处理字符或字符串。但机器中只能存储二进制数，所以字符在机器里必须用二进制数来表示。
- 一般采用目前最常用的美国信息交换标准代码（ASCII：American Standard Code for Information Interchange）来表示
- ASCII码：用一个字节来表示一个字符，其中低7位为字符的ASCII码值，最高位一般用作校验位

为什么不用其它表示法替代ASCII码？

常用字符的ASCII码

字符	ASCII码
0 ~ 9	30H ~ 39H
A ~ Z	41H ~ 5AH
a ~ z	61H ~ 7AH
回车CR	0DH
换行LF	0AH
$	24H
空格(SPACE)	20H

请熟记！

目前常用输入输出设备（显示器、打印机、键盘等）均采用ASCII码

1.4 几种基本的逻辑运算

1.4.1 AND “与”运算

1.4.2 OR “或”运算

1.4.3 NOT “非”运算

1.4.4 XOR “异或”运算

逻辑变量: 只能有0或1两种取值

所有数字设备中的控制、运算等处理电路都基于这几种基本逻辑运算器件实现

1.4.1 AND “与”运算

A	B	F = A∧B
0	0	0
0	1	0
1	0	0
1	1	1

1.4.2 OR “或”运算

A	B	F = A∨B
0	0	0
0	1	1
1	0	1
1	1	1

1.4.2 NOT “非”运算

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

1.4.4 XOR “异或”运算

A	B	F = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

谢谢！