微机原理与接口技术

唐彦，吕鑫

tangyan@hhu.edu.cn

河海大学计算机与信息学院
QQ：1795305539

课程目标

- 了解微型计算机的基本工作原理
- 熟练掌握汇编语言程序设计方法
- 掌握可编程接口芯片的应用

教材及主要参考书

- 《微机原理与接口技术》
  冯博琴主编，清华大学出版社，2016
  第四版

主要参考书:

- 《x86 PC汇编语言、设计与接口》（第五版），（美）马兹迪、考西著，电子工业出版社，2011
- Zen of Assembly Language
- Programming Black Book – Quake3 FPS

课程内容

- 8章节 / 8 周时间
- 第一周: 第一，二章
- 第二、三周: 第三章 (动手)
- 第四周: 第四章

课程内容

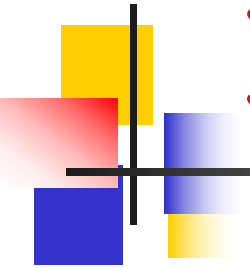
第5-7 周： 5-8章节，吕鑫老师

- 第8 周：课程小结，复习和答疑

- 答疑时间:
- 唐彦： 周二：16:00 - 17:30，勤4105

课程考核

- 平时成绩: 30%
  - 作业 + 考勤（1 次）: 10%
  - 课程设计（1次）20% 4人以上一组
    - 编写课设：扫雷 / 五子棋 / 机器学习算法;
    - 每组4人以上
    - 研究前沿分析报告，题目自选：智能芯片、深度学习、人机交互（智能设备 + 人）。
- 期末考试 70%： 12月底-1月初



第一章

微型计算机基础概论

1.1 微型计算机系统

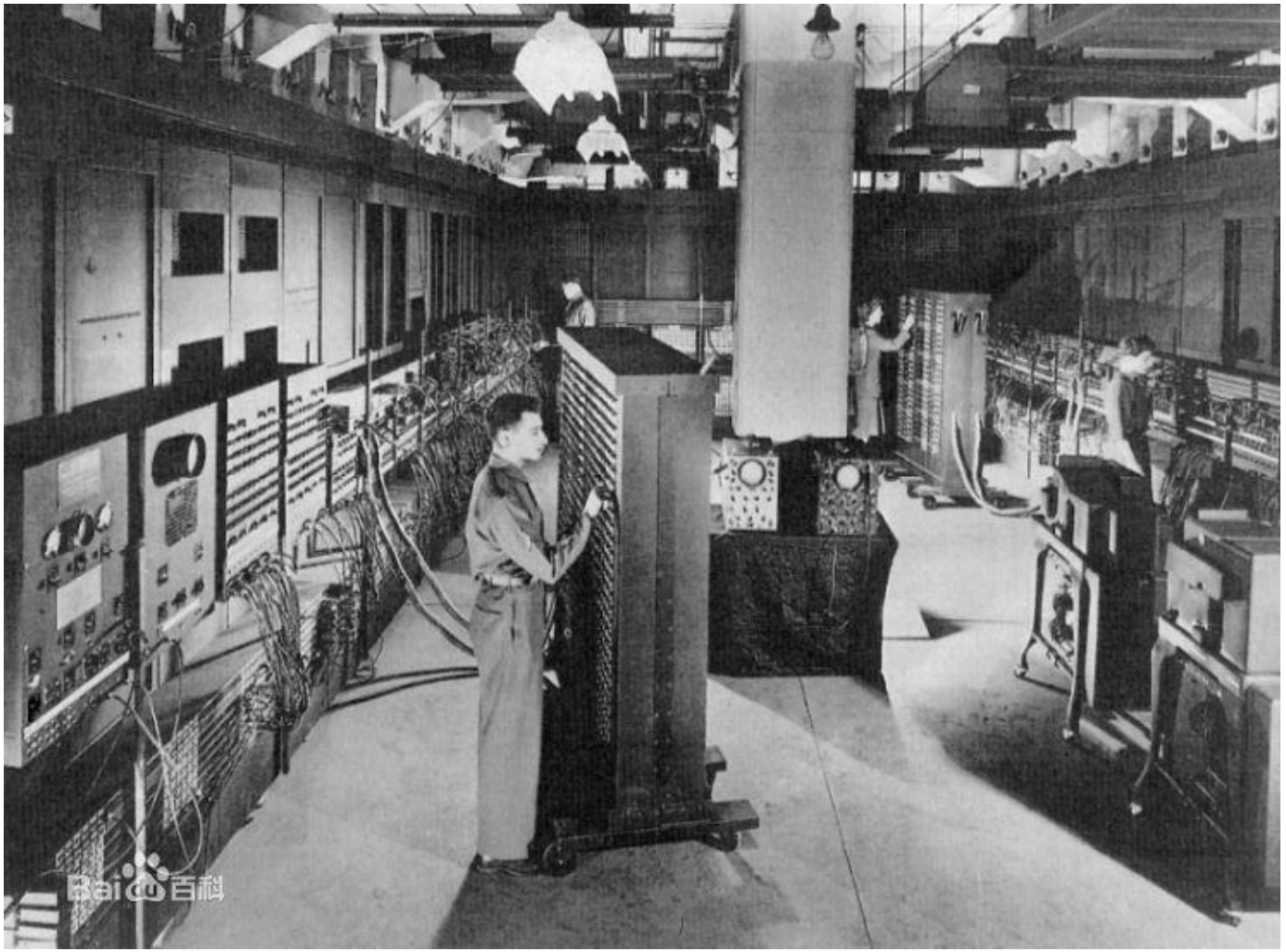
1.1.1 微型计算机的发展

- 微型计算机的发展是以微处理器的更新换代为标志。

- 自1946年第一台计算机产生(Eniac) 计算机经历了电子管、晶体管、集成电路、大规模及超大规模计算机的发展更替。

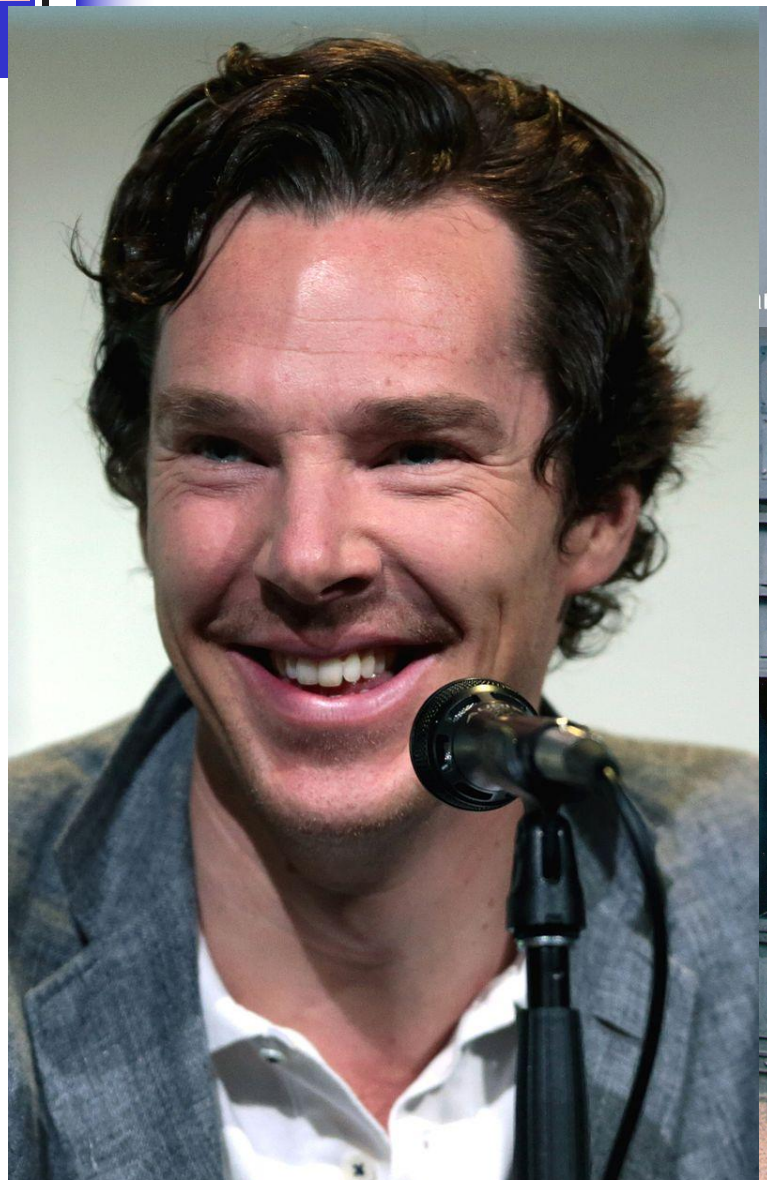
- 按性能、价格和体积等指标，计算机可分为：巨型机、大型机、中型机、小型机和微型机五大类。

Eniac

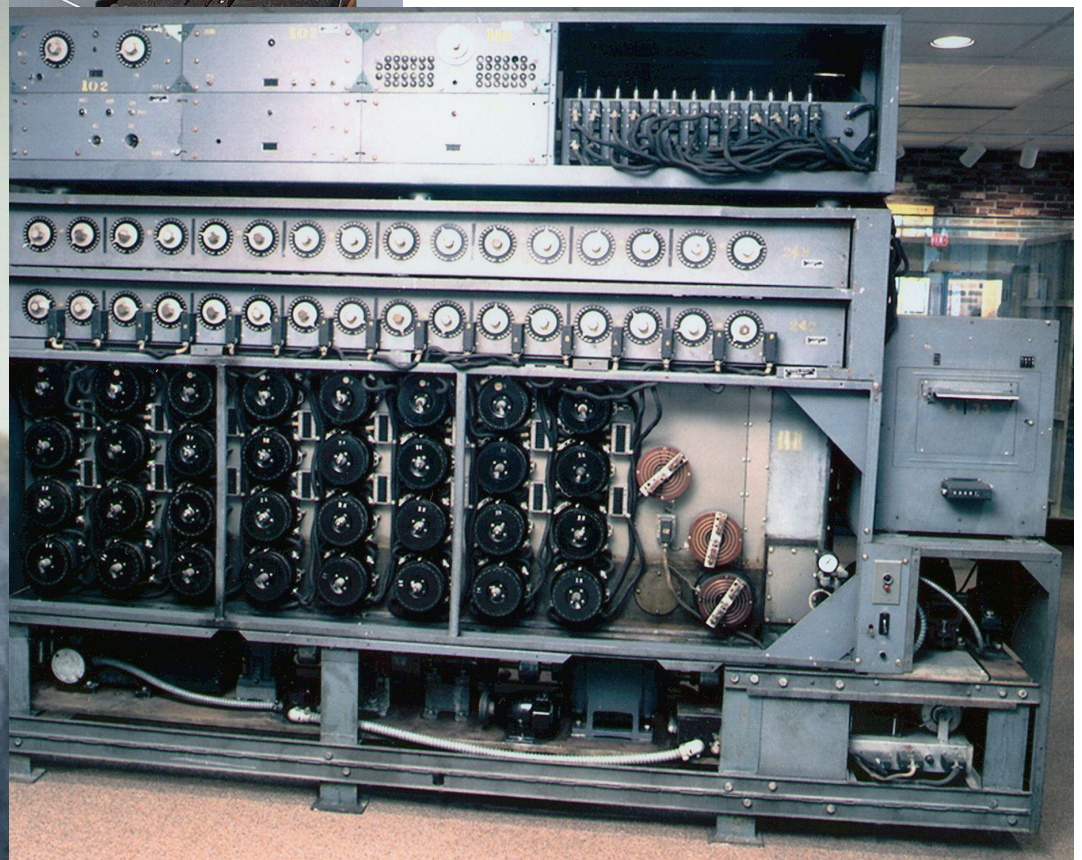


T.L 25Y

The Imitation Game

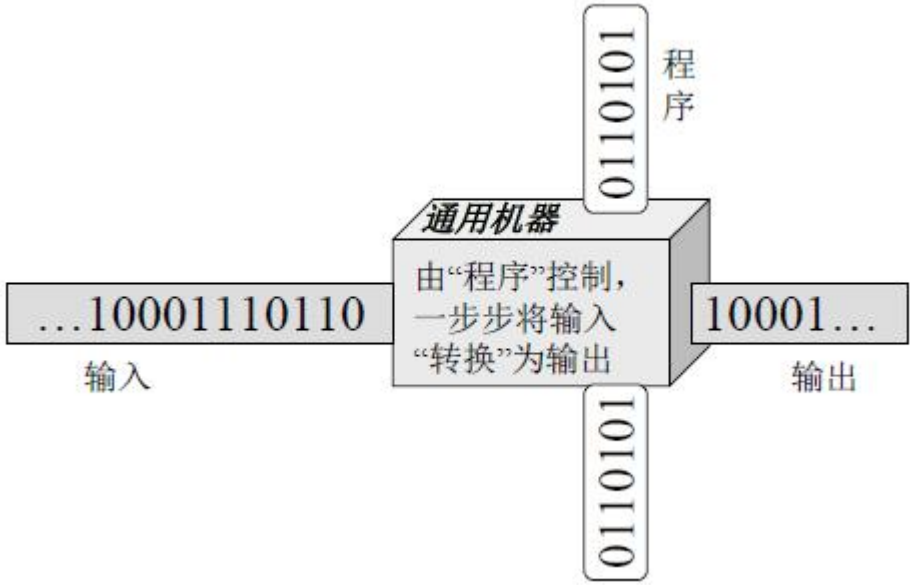


1942



图灵机

- 图灵机就是指一个抽象的机器，它有一条无限长的纸带，纸带分成了一个一个的小方格，每个方格有不同的颜色。
- 有一个机器头在纸带上移来移去。机器头有一组内部状态，还有一些固定的程序。在每个时刻，机器头都要从当前纸带上读入一个方格信息，然后结合自己的内部状态查找程序表，根据程序输出信息到纸带方格上，并转换自己的内部状态，然后进行移动。



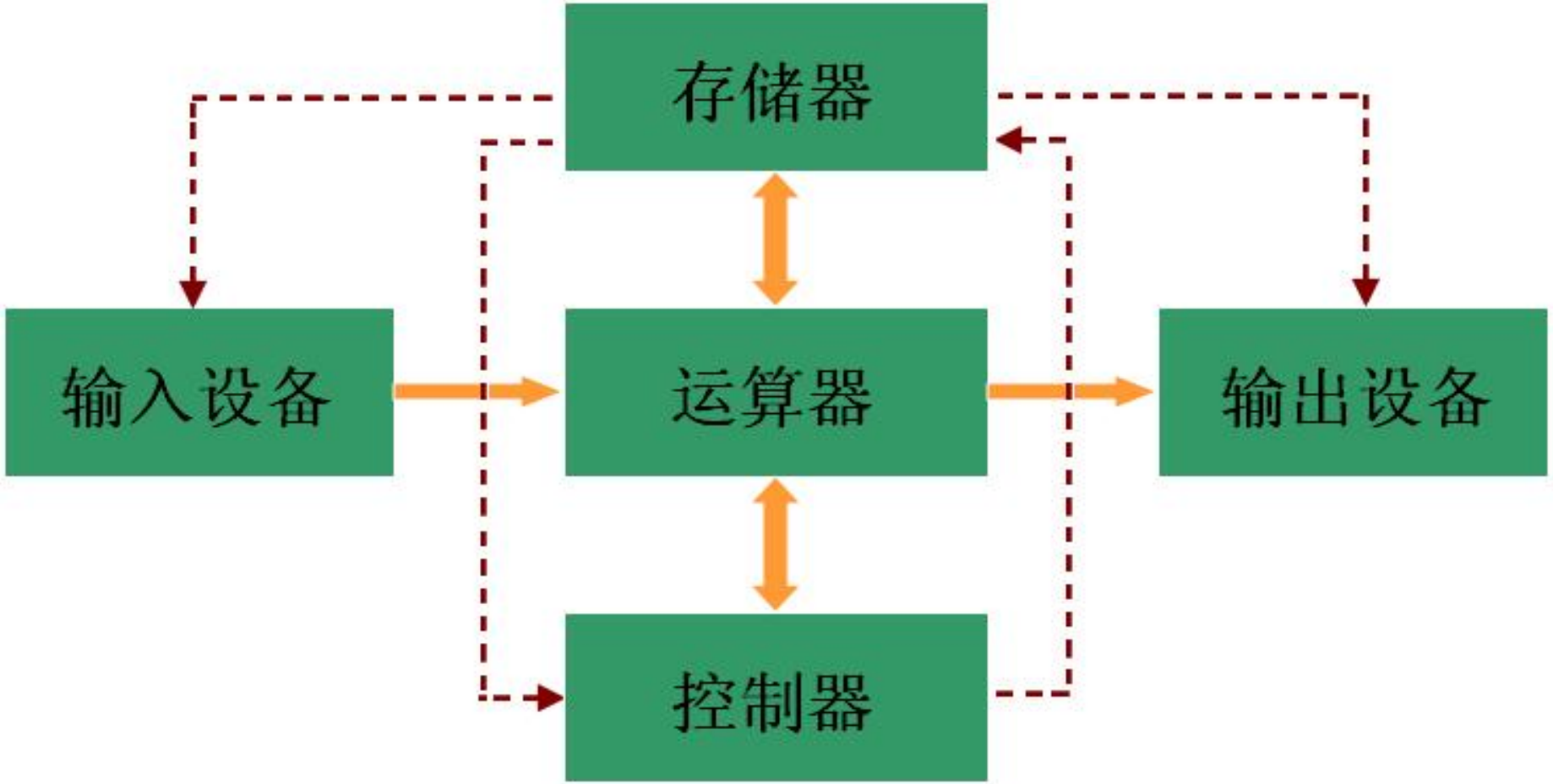
1.1.2 微型计算机的工作过程

冯·诺依曼计算机的原理

每台计算机都有各种类型的机器指令，这些指令按照一定的规则存放在存储器中，在中央控制系统的统一控制下，按一定顺序依次取出执行，这就是冯诺依曼的核心原理，即存储程序的工作原理。

计算机的工作过程就是执行程序的过程，而程序是指令序列的集合。

冯·诺依曼计算机结构



程序放在哪里？ -- 内存

把程序和数据送到具有记忆能力的存储器中保存起来，计算机工作时只要给出程序中第一条指令的地址，控制器就可依据存储程序的指令顺序地、周而复始地取出指令、分析指令、执行指令，直到执行完全部指令为止。

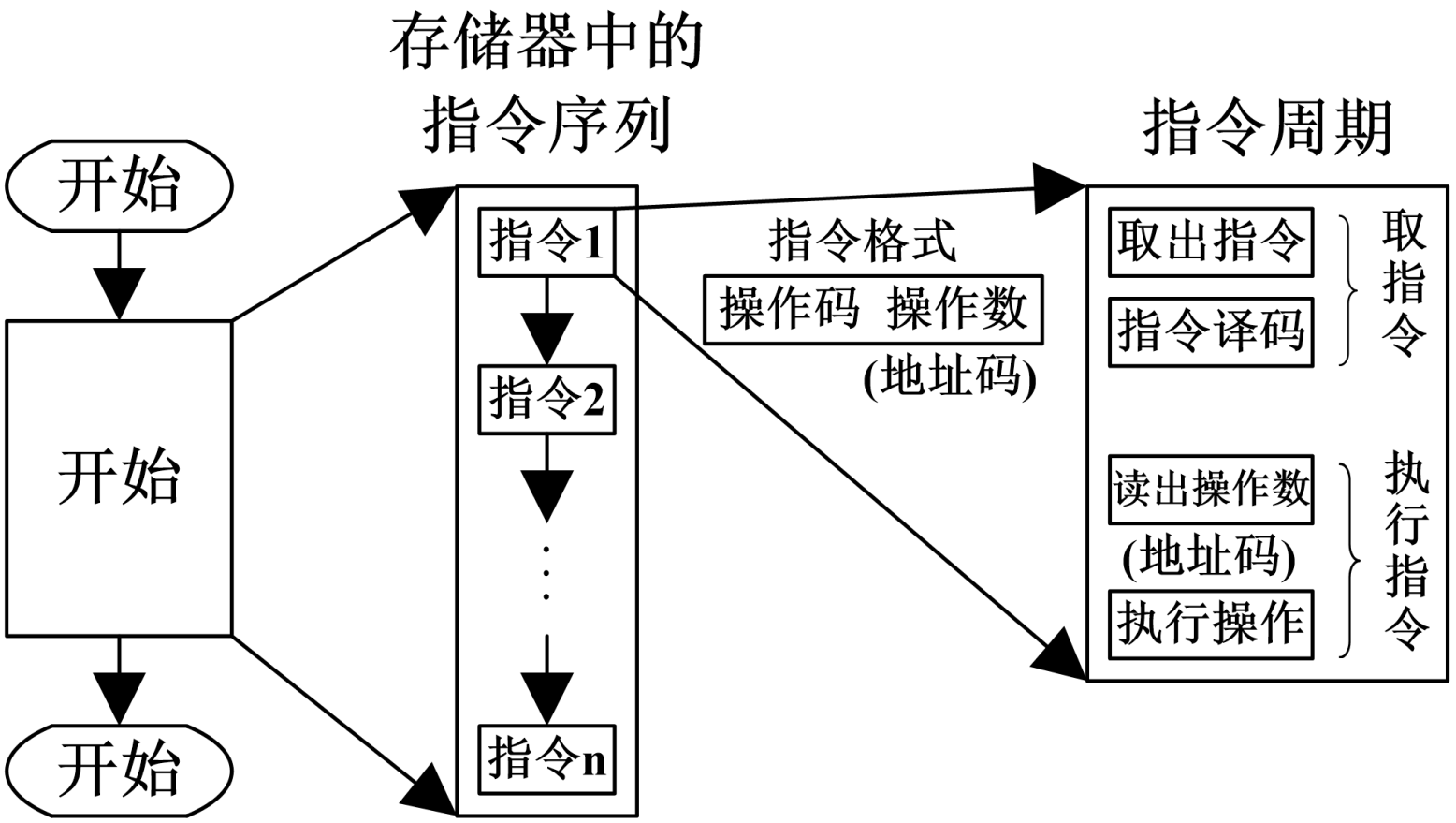
- **指令**：一条基本操作命令称为一条机器指令。指令是对计算机发出的一条条工作命令，命令它执行规定的操作。机器指令必须满足两个条件：
  a、机器指令的形式必须是计算机能够理解的，必须使用二进制数字编码形式表示。
  b、机器指令规定的操作必须是计算机能够执行的。必须有硬件支持。
- **指令系统**：应用于某种CPU的机器指令及其使用规则的集合。指令系统决定了计算机的能力，也影响着计算机的结构。
- **程序**：是实现某种任务的指令序列。

冯·诺依曼机的特点

- 将计算过程描述为由许多条指令按一定顺序组成的程序，并放在存储器保存。
- 程序中的指令和数据必须采用**二进制编码**，且能够被执行该程序的计算机所识别。
- 指令按其在存储器中存放的顺序执行，存储器的字长固定并按顺序线性编址。

- 由**控制器**控制整个程序和数据的存取以及程序的执行。
- 以**运算器**为核心，所有的执行都经过运算器

微型计算机的工作过程

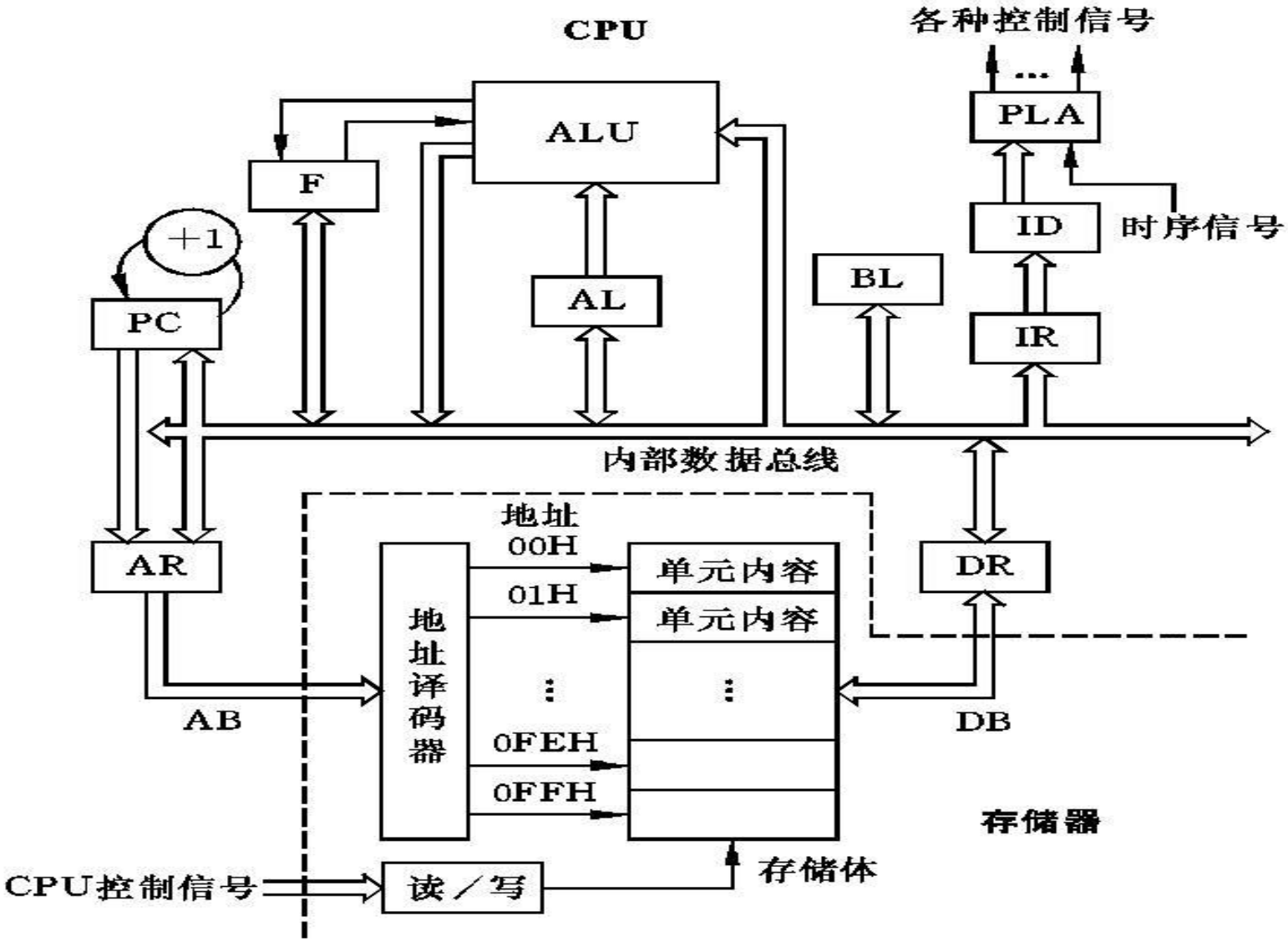

存储器中的
指令序列
指令周期
开始
开始
开始
指令1
指令2
指令n
指令格式
操作码 操作数
(地址码)
取出指令
指令译码
取指令
读出操作数
(地址码)
执行操作
执行指令

微型计算机的工作过程

- 1. 首先将第一条指令由内存中取出;
- 2. 将取出的指令送指令译码器译码，以确定要进行的操作;
- 3. 读取相应的操作数(即执行的对象);
- 4. 执行指令;
- 5. 存放执行结果;
- 6. 一条指令执行完后，转入了下一条指令的取指令阶端，如此终而复始地循环，直到程序中遇到暂停指令才结束。

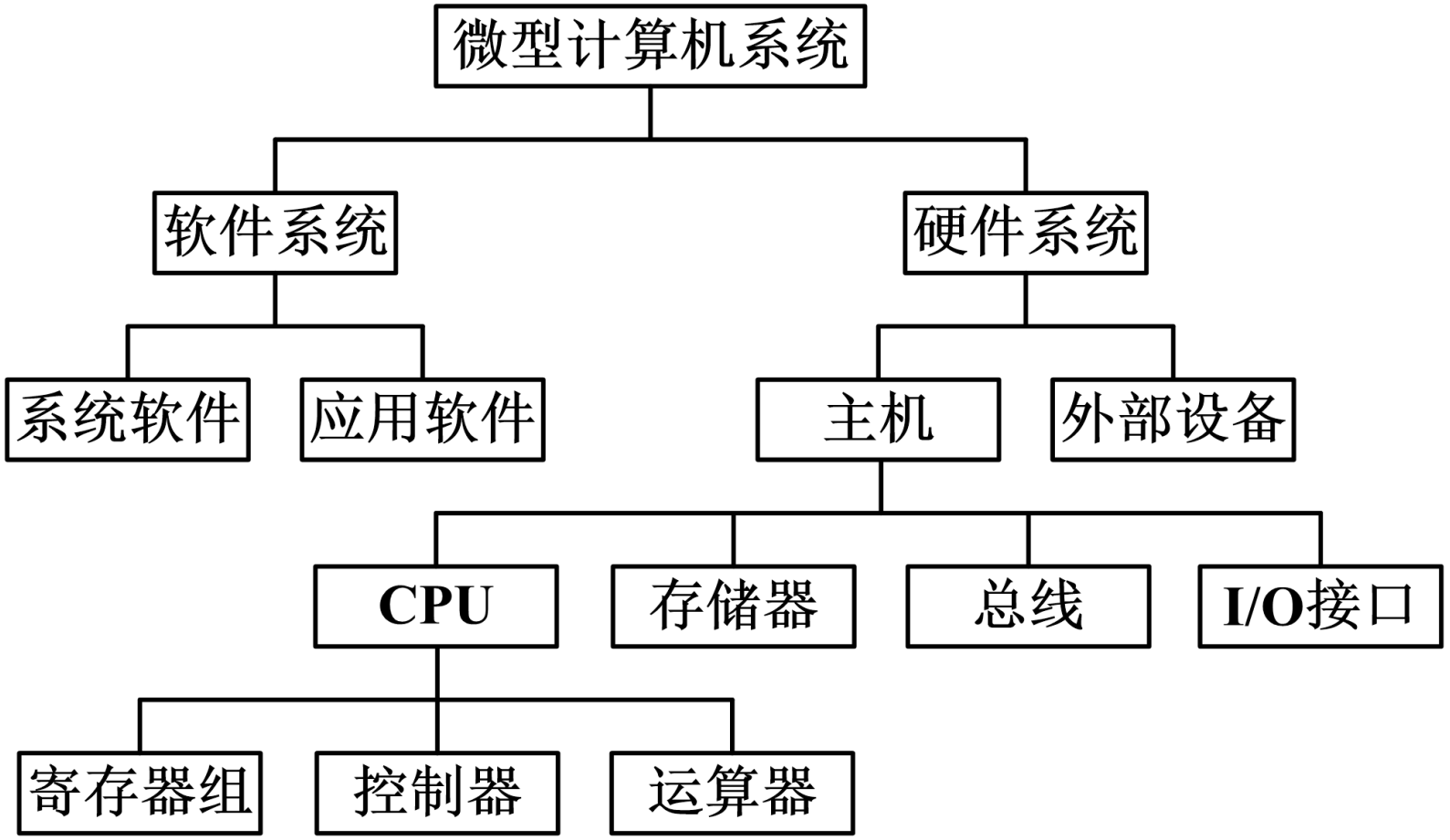
微型计算机的工作过程

- 计算机的工作过程就是执行程序的过程，即逐条执行指令序列的过程
  - 程序是指令序列的集合。
- 指令执行的两个基本阶段
  (1)取指令阶段: 由一系列相同的操作组成。
  取指令阶段的时间总是相等的。
  (2)执行指令阶段: 由不同的事件顺序组成，
  它取决于被执行指令的类型。
- 微机工作过程就是不断地取指令和执行指令的过程。


CPU
各种控制信号
...
PLA
ALU
F
ID
时序信号
+1
PC
AL
BL
IR
内部数据总线
地址
00H
01H
0FEH
0FFH
AR
地址译码器
单元内容
单元内容
DR
AB
DB
存储器
CPU控制信号
读／写
存储体

计算机执行指令的过程

1.1.3 微机系统的组成


微型计算机系统
软件系统
硬件系统
系统软件
应用软件
主机
外部设备
CPU
存储器
总线
I/O接口
寄存器组
控制器
运算器

1)微处理器

简称CPU，是计算机的核心

主要包括:

- 运算器
- 控制器
- 寄存器组

□运算器

算术逻辑单元ALU:

加法器：加、减、乘、除

逻辑运算功能部件：与、或、非、异或

通用或专用寄存器组:

提供操作数和暂存中间运算结果及结果特征

内部总线:

数据传输通道

控制器

组成:

程序计数器、指令寄存器、指令译码器、时序控制部件，微操作控制部件

功能:

1、指令控制

2、时序控制

3、操作控制

4、处理对异常情况及某些外部请求

寄存器组

CPU内部的若干个存储单元;
分为专用寄存器和通用寄存器;
专用寄存器：其作用是固定的,如SP、FLAGS 。
通用寄存器：如AX、BX等由程序员规定其用途。

2)存储器

- 定义：存储器又叫内存或主存,是微型计算机的存储和记忆部件。

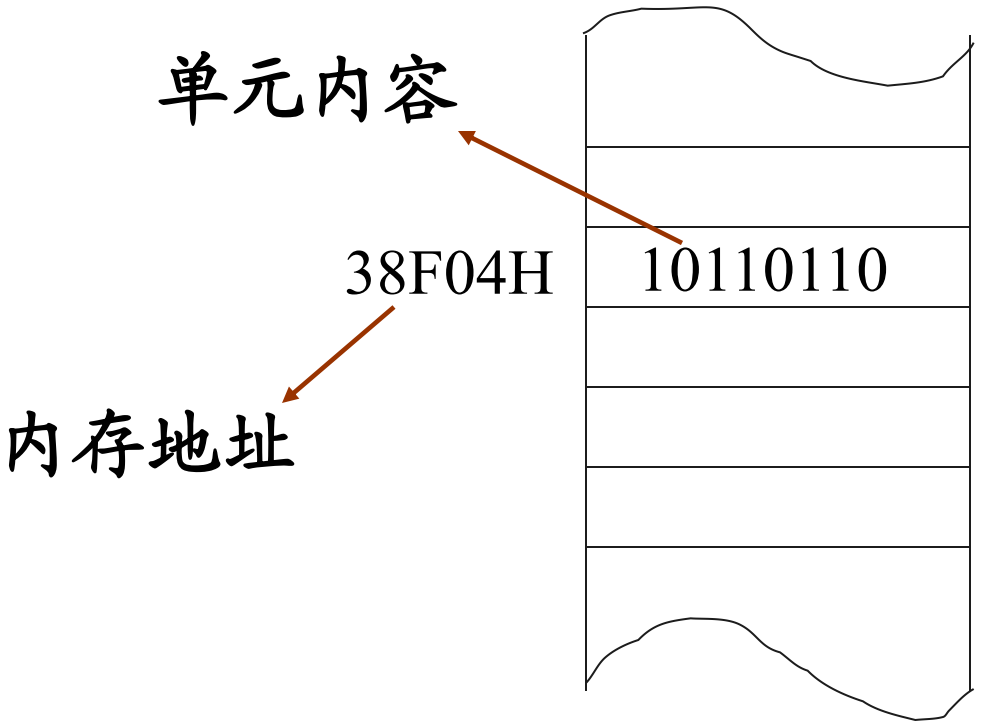
- 作用：存放计算机工作过程中需要操作的数据和当前执行的程序。

寄存器与存储器的比较

寄存器	存储器
在CPU内部	在CPU外部
访问速度快	访问速度慢
容量小，成本高	容量大，成本低
用名字表示	用地址表示
没有地址	地址可用各种方式形成

■注意区分：内存单元的地址和内容

- 地址:每个单元都对应一个编号，以实现对单元内容的寻址
- 内存单元的内容:内存单元中存放的信息



☆ 指标：内存容量

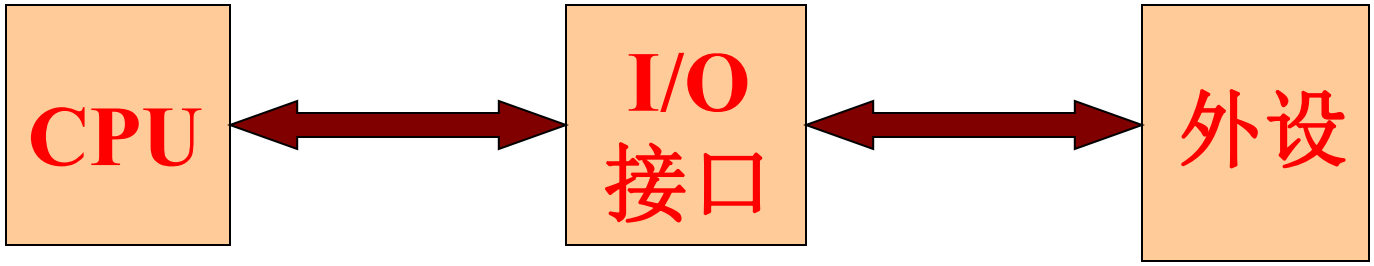
- 内存所含存储单元的个数，以字节为单位
- 内存容量的大小依CPU的寻址范围而定（即CPU地址信号线的位数）

☆ 内存操作

- 读：将内存单元的内容取入CPU，原单元内容不改变
- 写：CPU将信息放入内存单元，单元中原来的内容被覆盖

3）输入/输出接口

- 输入/输出接口是微型计算机的重要组成部分
- 接口是CPU与外部设备间的桥梁



☆接口的分类：

- 串行接口
- 并行接口

- 输入接口
- 输出接口

☆ 接口的功能

- 数据缓冲寄存
- 信号电平或类型的转换
- 实现主机与外设间的运行匹配

4）总线

总线：是一组信号线的集合，是在计算机系统各部件之间传输地址、数据和控制信息公共通路。

内部总线：位于芯片内部的总线。

系统总线：连接微处理器与存储器、输入输出接口，用以构成完整的微型计算机的总线 (外部总线) 。

分类：数据总线、地址总线和控制总线。

数据总线：用于传送数据信息，实现微处理器、存储器和I/O接口之间的数据交换。数据总线是双向总线，数据可在两个方向上传输。

地址总线：用于发送内存地址和I/O接口的地址，是单项总线。

控制总线：则传送各种控制信号和状态信号，使微型计算机各部件协调工作。每一根是单向的，但整体是双向的。

程序设计语言

机器语言：计算机直接执行的二进制形式的程序。

汇编语言：助记符语言表示的程序。

高级语言：不依赖于具体机型的程序设计语言。

汇编语言

- 汇编语言通常被应用在底层，**硬件操作**和**高要求的程序优化**的场合。**驱动程序**、**嵌入式操作系统**和**实时运行程序**都需要汇编语言。

- 学习汇编语言，有助于深入了解计算机的运行原理，机器的逻辑功能。

- Linux内核的关键地方使用了汇编代码，可以用于软件的加解密、计算机病毒的分析和防治，以及程序的调试和错误分析等各个方面（CIH99）

- 机器语言程序
- 汇编语言程序
- 高级语言程序

高级语言程序和汇编语言程序必须先翻译成机器语言程序才能执行。这个翻译过程，对汇编语言程序叫**汇编（Assemble）**；对高级语言程序有的叫**解释（Interpretation）**，有的叫**编译（Compilation）**。

通常又将翻译前的程序叫源程序，而将翻译后的机器语言程序叫目标程序。

完成汇编、解释、编译的程序则分别叫作**汇编程序（Assembler）**、**解释程序（Interpreter）**、**编译程序（Compiler）**,它们作为工具软件事先存放在计算机中。

1.2 计算机中的数制及编码

◆ 1.2.1常用数制:

二进制、八进制、十进制、 十六进制

■十进制数: 以十为底，逢十进一。共有0～9十个数字符号。用D表示.

$$D=D_{n-1}\times10^{n-1}+D_{n-2}\times10^{n-2}+\cdots+D_0\times10^{0}+D_{-1}\times10^{-1}+\cdots+D_{-m}\times10^{-m}$$

$$=\sum_{i=-m}^{n-1}D_i\times10^{i}$$

[例] $(3256.87)_{10}$

2.二进制数：以2为底，逢2进位；只有0和1两个符号。用B表示。

$$(B)_2 = B_{n-1} \times 2^{n-1} + B_{n-2} \times 2^{n-2} + \cdots + B_0 \times 2^0 + B_{-1} \times 2^{-1} + \cdots + B_{-m} \times 2^{-m}$$

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

[例] $(1010)_2$

3.十六进制数：有0--9及A--F共16个数字符号，逢16进位。用H表示。

$$(H)_{16} = H_{n-1} \times 16^{n-1} + H_{n-2} \times 16^{n-2} + \cdots + H_0 \times 16^0 + H_{-1} \times 16^{-1} + \cdots + H_{-m} \times 16^{-m}$$

$$= \sum_{i=-m}^{n-1} H_i \times 16^i$$

[例] Color Code $(2AEF)_{16}$

4. K进制数：有0-(K-1)共K个数字符号，逢K进位。

$$(S)_K = S_{n-1} \times \mathrm{K}^{n-1} + S_{n-2} \times \mathrm{K}^{n-2} + \cdots + S_0 \times \mathrm{K}^0 + S_{-1} \times \mathrm{K}^{-1} + \cdots + S_{-m} \times \mathrm{K}^{-m}$$

$$= \sum_{i=-m}^{n-1} S_i \times \mathrm{K}^i$$

[例] $(876.45)_{\mathrm{K}}$

1.2.2 各种进制数间的转换

1. 非十进制数转换为十进制数

方法：将非十进制数按相应的权表达式展开，再求和。

[例] $(1101)_2=(13)_{10}$

[例] $(ff)_{16}=(256)_{10}$

$$(S)_K = S_{n-1}\times K^{n-1} + S_{n-2}\times K^{n-2} + \cdots + S_0 \times K^0$$

$$= (\cdots((S_{n-1}\times K + S_{n-2})\times K + S_{n-3})\times K + \cdots S_1)\times K + S_0$$

2. 十进制数转换为非十进制数

(1) 十进制数转换为二进制数

对整数：除2取余；
对小数：乘2取整。

[例] $(112.25)_{10}=(1110000.01)_{2}$

3. 二进制数与十六进制数之间的转换

[例] $(110100110.101101)_2$

$=(1A6.B4)_{16}$

[例] $(2A8F.6D)_{16}$

$=(001010101000111.01101101)_2$

1.2.4 二进制编码

1. 二进制编码的十进制数

BCD码: 用二进制编码表示的十进制数，简称BCD码(Binary Coded Decimal)

(1) 8421 BCD码

用4位二进制编码表示的1位十进制数，其4位二进制编码的每1位都有特定的位权，从左到右分别为$2^3=8$，$2^2=4$，$2^1=2$，$2^0=1$。

十进制数：0000 ~ 1001 十种状态；

1010 ~ 1111 六种状态非法。

(2) BCD码与十进制数、二进制数的转换

- **BCD码与十进制数的转换**

  对十进制数的每一位按对应关系单独进行

  转换即可。

  [例]: $(234.15)_{10}$

  $=(0010\ 0011\ 0100.0001\ 0101)_{BCD}$

- **BCD码与二进制数的转换**

  先转换为十进制数，再转换二进制数；反

  之同样。

  [例]: $(0001\ 0001.0010\ 0101)_{BCD}$

- 计算机中BCD码的存储方式
  - ▲ 压缩BCD码

    用4位二进制码表示一位十进制数

    [例]: $(10010010)_{BCD}=92$
  - ▲ 扩展BCD码

    用8位二进制码表示一位十进制数

    [例]: $(00000010)_{BCD}=2$

2. 字符的编码--ASCII码

ASCII码: American Standard Code for Information Interchange, 美国国家标准信息交换码

字符的编码用7位二进制码表示。在需要时可在D7位加校验位。

1.3 无符号二进制数的算术运算和逻辑运算

1.3.1 二进制数的算术运算

1. 加法运算

运算法则: 0+0=0　0+1=1　1+0=1　1+1=0

(有进位)

2. 减法运算

运算法则: 0-0=0　1-0=1　1-1=0　0-1=1

(有借位)

1.3.2 无符号二进制数的表示范围

☆ 机器数：符号数值化了的数；用一位表示符号的二进制数。

☆ 机器数的真值：原来的数值(包括+、-号)。（包括+、-号）

1. 无符号二进制数的表示范围

一个n位无符号二进制数X，它可表示的数的范围为：

$0 \leqq X \leqq 2^n-1$，若运算结果超出这个范围，则产生溢出。

对无符号数：运算时，当最高位向更高位有进位(或借位)时则产生溢出。

[例] 10110111B+0100110B=(1) 00000100B

↑

溢出

8位（1字节）表示数的范围：0～255

2. 无符号二进制数的溢出判断

判断：最高有效位D_i向更高位有进位(或相减有借位)，则产生溢出。

对错：无符号数的溢出不一定是错的，可能是向更高位的进位或借位

1.3.3 二进制数的逻辑运算

1. “与” 运算

“与” 运算的规则：按位相“与”：

1∧1=1　1∧0=0　0∧1=0　0∧0=0

[例] 10110110B∧10010011B

2. “或” 运算

“或” 运算的规则：按位相“或”：

0∨0=0　0∨1=1　1∨0=1　1∨1=1

[例] 11011001B∨10010110B

3. “非” 运算

“非” 运算的规则：按位取反：

$\overline{1}=0$　$\overline{0}=1$

[例] 11011001 B=00100110B

4. “异或” 运算

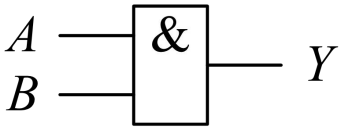
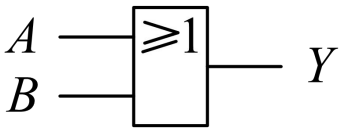
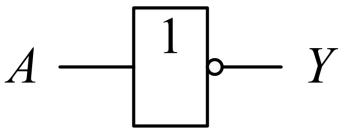
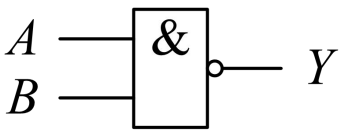
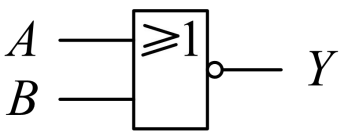
“异或” 运算的规则：

按位操作，不同为1，相同为0

0⊕ 0=0　1⊕ 1=0　0⊕ 1=1　1⊕ 0=1

[例] 11010011B⊕ 10100110B

1.3.4 基本逻辑门及常用逻辑部件

基本逻辑门	表示	逻辑符号
1. 与门	$Y=A\wedge B$	
2. 或门	$Y=A\vee B$	
3. 非门	$Y=\overline{A}$	
4. 与非门	$Y=\overline{A\wedge B}$	
5. 或非门	$Y=\overline{A\vee B}$	

真值表

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1
0	0	0
0	1	1
1	0	1
1	1	1
0		1
1		0
0	0	1
0	1	1
1	0	1
1	1	0
0	0	1
0	1	0
1	0	0
1	1	0

1. 与门
2. 或门
3. 非门
4. 与非门
5. 或非门

实际使用的器件及符号:

与门： 双与门74LS08、三与门74LS11、四与门74LS21
或门：双或门74LS32
非门：反相器74LS04及74HC04
与非门：双与非门74LS00、三与非门74LS10
或非门：三或非门74LS27

6. 译码器

译码器的作用是将一组输入信号转换为在确定的输出信号。译码器是多输入，多输出的组合逻辑电路。

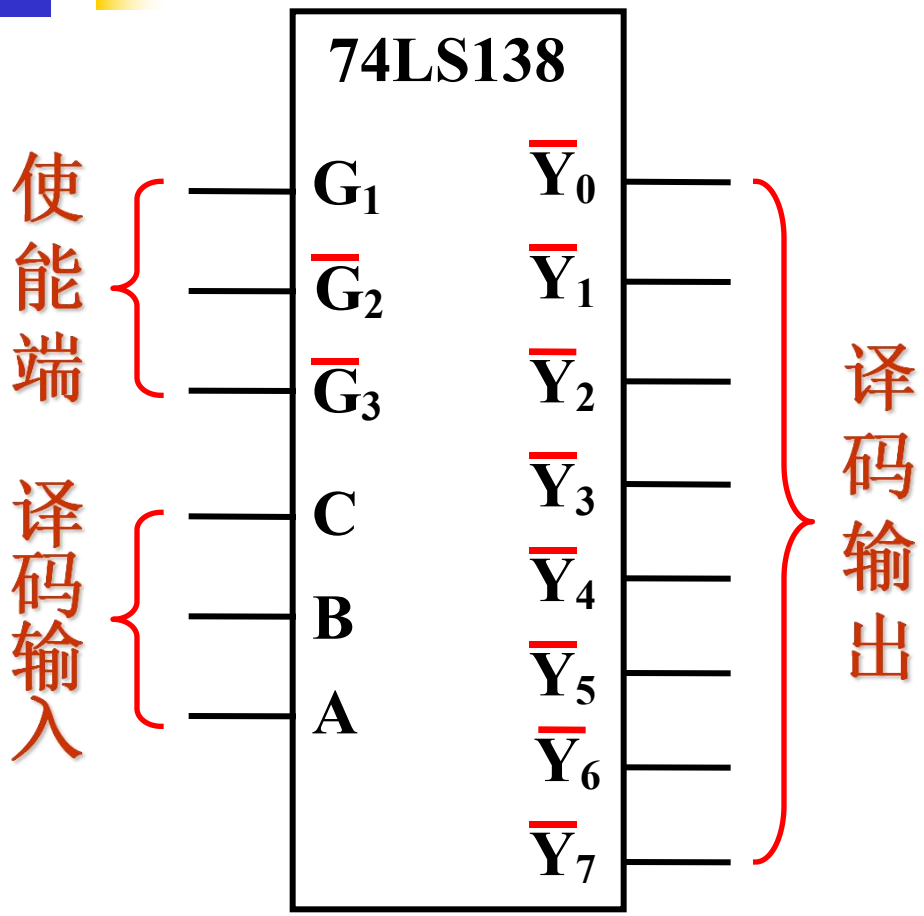
常用的译码器有:

(1) 2-4 译码器：74LS139

(2) 3-8 译码器：74LS138

(3) 4-16 译码器：CD4514

3-8 译码器：74LS138



当译码输出G_1=1，G_2=G_3=0时，译码器处于使能状态，即译码器当前被允许工作。

三位二进制译码器的状态表

输入			控制端			输出							
A	B	C	$G1$	$\overline{G}_{2A}$	$\overline{G}_{2B}$	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
0	0	0	1	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	0	1	0	1	1	1	1	1	1
0	1	0	1	0	0	1	1	0	1	1	1	1	1
0	1	1	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1	0	1	1
1	1	0	1	0	0	1	1	1	1	1	1	0	1
1	1	1	1	0	0	1	1	1	1	1	1	1	0

1.4 有符号二进制数的表示及运算

☆ 常规: “0” --- 正

“1” --- 负

☆ 机器数: 符号数值化了的数;

用一位表示符号的二进制数。

☆ 机器数的真值: 原来的数值。

(包括+、-号)

1.4.1 有符号二进制数的表示方法

1. 原码 $[X]_{原}$

最高位为符号位，用“0”表示正，用“1”表示负；

其余为真值部分

[例] 已知X=+42，Y=-42，求$[X]_{原}$，$[Y]_{原}$

$[X]_{原}$=0 0101010

$[Y]_{原}$=1 0101010

优点：真值和其原码表示之间的对应关系简单，容易理解

缺点：计算机中用原码进行加减运算比较困难，0的表示不惟一

数0的原码

8位数0的原码：$[+0]_{原}=0\ 0000000$

$[-0]_{原}=1\ 0000000$

即：数0的原码不唯一

原码的定义：若二进制数$X=X_{n-1}X_{n-2}\cdots X_1X_0$

$$[X]_{原}=\begin{cases} X & 2^{n-1} > X \geqq 0 \\ 2^{n-1}-X = 2^{n-1}+|X| & 0 \geqq X > -2^{n-1} \end{cases}$$

2. 反码 $[X]_{反}$

对一个机器数X:

若X≥0，则 $[X]_{反}=[X]_{原}$

若X<0，则 $[X]_{反}$= 对应原码的符号位不变，数值部分按位求反

[例] 已知X=+42，Y=-42，求$[X]_{反}$反，$[Y]_{反}$反

$[X]_{反}$=0 0101010

$[Y]_{反}$=1 1010101

0的反码:

$[+0]_{反}$ = 00000000

$[-0]_{反}$ = 11111111

即：数0的反码不唯一

补充：负数反码的计算方法小结

1. 负数的反码的数值部分为真值的各位按位取反;
2. 负数的反码等于其对应正数的原码按位取反再加1;
3. 反码的定义公式：

$$[X]_{反}=\begin{cases} X & 2^{n-1}>X\geqq 0 \\ (2^{n}-1)+X & 0\geqq X>-2^{n-1} \end{cases}$$

3. 补码 $[X]_{补}$

$X = X + 2^n \ (\mathrm{mod}\ 2n)$

对一个机器数X：

若X>0， 则$[X]_{补}= [X]_{反}= [X]_{原}$

若X<0， 则$[X]_{补}= [X]_{反}+1$

[例] 已知X=+42，Y=-42，求$[X]_{补}$，$[Y]_{补}$

$[X]_{补}$=0 0101010

$[Y]_{补}=[Y]_{反}+1=1$1010101$+1=11010110$

0的补码:

$[+0]_{补} = 00000000$

$[-0]_{补} = 11111111+1=00000000(\mathrm{mod}\ 2^8)$

即：数0的补码唯一,$[0]_{补} = 00000000$

补码的定义

若二进制数$X=X_{n-1}X_{n-2}\cdots X_1X_0$

$$[X]_{补} = \begin{cases} X & 2^{n-1} > X \geqq 0 \\ 2^n+X= 2^n-|X| & 0 > X \geqq -2^{n-1} \end{cases}$$

补充：负数补码的计算方法小结

1. 负数的补码等于其符号位不变，数值部分的各位按位取反再加1;
2. 负数的补码等于其对应正数的补码包括符号位一起按位取反再加1，即$[Y]_{补}=[Y]_{反}+1$
3. $[Y]_{补}=2^n+Y(\mathrm{mod}\ 2^n)$, $Y<0$ 时
4. 补码的定义公式：

$$[X]_{补}=\begin{cases} X & 2^{n-1}>X\geqq 0 \\ 2^n+X=2^n-|X| & 0>X\geqq -2^{n-1} \end{cases}$$ (方法3)

1.4.2 补码数与十进制数之间的转换

补码数 ⟶ 十进制数的步骤:
1)求出真值; 2)进行转换

1. 正数补码的转换
除符号位以外的数值部分就是该数的真值。

1.4.2 补码数与十进制数之间的转换

负数补码的转换

$$[X]_{补} \xrightarrow{\text{按位取反加1}} [-X]_{补} \xrightarrow{\text{按位取反加1}} [X]_{补}$$

当*X*为正数时，对其补码按位取反加1，结果是-*X*的补码；

当*X*为负数时，对其补码按位取反加1，结果是+*X*的补码；

所以，对负数补码再求补的结果就是该负数的真值。

$X=[[X]_{补}]_{补}$ ；$X<0$

例：将一个用补码表示的二进制数转换为十进制数

(1) $[X]_{补}$=0 0101110B, 求X的真值。

(2) $[X]_{补}$=1 1010010B, 求X的真值。

解：(1) $[X]_{补}$=0 0101110B 为正数

所以：X=+46

(2) $[X]_{补}$=1 1010010B 为负数

所以：$X=[[X]_{补}]_{补}=[1\ 1010010]_{补}$

$= 1\ 0101101 + 1$

$= 1\ 0101110$(符号位变为-)

$= -\ 0101110$

$= -\ 46$

1.4.3 补码的运算

☆ 通过引进补码，可将减法运算转换为加法运算

☆ 补码运算规则：

(1) $[X+Y]_{补}=[X]_{补}+[Y]_{补}$

(2) $[X-Y]_{补}=[X]_{补}-[Y]_{补}$

(3) $[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补}$

$[-Y]_{补}$称为对补码数$[Y]_{补}$求变补，变补的规则是：

(1) 对$[Y]_{补}$的每一位(包括符号位)按位取反加1，则结果就是$[-Y]_{补}$

(2) 直接对-Y求补码

[例]:

X=+66，Y=+51，求$[X-Y]_{补}$=?

$[X]_{补}$ =01000010

$[-Y]_{原}$=10110011

$[-Y]_{补}$= $[-Y]_{反}$+1=11001100+1=11001101

所以：$[X-Y]_{补}$= $[X]_{补}$+ $[-Y]_{补}$

= 01000010

+ 11001101

= 1 00001111

↑

自然丢失

1.4.4 符号数运算中的溢出问题

1. 带符号数的表示范围

☆8位二进制符号数，原、反、补码所能表示的范围：

- 原码：11111111B ~ 01111111B　-127 ~ +127
- 反码：10000000B ~ 01111111B　-127 ~ +127
- 补码：10000000B ~ 01111111B　-128 ~ +127

☆16位二进制符号数，原、反、补码所能表示的范围：

- 原码：FFFFH ~ 7FFFH　-32767 ~ +32767
- 反码：8000H ~ 7FFFH　-32767 ~ +32767
- 补码：8000H ~ 7FFFH　-32768 ~ +32767

带符号数运算时的溢出判断

两个带符号二进制数相加或相减时，若

$$C_{最高位} \oplus C_{次高位} = 1$$

则结果产生溢出。（包含2种情况）

次高位向最高位有进位，而最高位向前无进位

最高位向前有进位，而次高位向前无进位

[例]：X=+72，Y=+98，用补码计算X+Y=?

$[X]_{补}$ = 01001000

$[Y]_{补}$ = 01100010

$[X]_{补}$+ $[Y]_{补}$= 01001000 +72

\+ 01100010 +98

10101010 -86

即：次高位向最高位有进位，而最高位向前无进位，产生溢出

(事实上，两正数相加得出负数，结果出错)

C_7=0，C_6=1； $C_7 \oplus C_6$=1

72+98=170=-86+256（mod 256）

[例]: X=-83，Y=-80，用补码计算X+Y=?

$[X]_{补}$ = 10101101

$[Y]_{补}$ = 10110000

$[X]_{补}$+ $[Y]_{补}$ = 10101101 -83

\+ 10110000 -80

1 01011101 +93

即: 最高位向前有进位，而次高位向前无进位，产生溢出。(事实上，两正数相加得出负数，结果出错)

$C_7 \oplus C_6 = 1$

-83+(-80)=-163=-163+256（mod 256）=93

无溢出条件: C_7=1，C_6=1; C_7=0，C_6=0。

用自陷中断处理溢出。

本章重点:

1. 冯-诺依曼计算机结构

2. 微型计算机的工作过程

4. 计算机中的数制及编码， 原、反、补