数据表示及检错纠错

2021年秋

6 Great Ideas in Computer Architecture

- 1. Abstraction(Layers of Representation/Interpretation)
- 2. Moore’s Law
- 3. Principle of Locality/Memory Hierarchy
- 4. Parallelism (Pipeline)
- 5. Performance Measurement & Improvement
- 6. Dependability via Redundancy

- 计算机组成，体系结构会经常随着底层技术和上层应用的变化而变化，新出来的概念繁多，需要从中总结出规律性的东西
- 学习组成原理的时候，需要经常思考上面的想法是如何落实到实际的计算机组成中的
- 在课程的不同部分会体现上面的某一点或者几点，比如在本讲座中会看到通过冗余来获得更高的可靠性

内容概要

- 数据表示的需求
- 逻辑型数据表示
- 字符的表示
- 整数与浮点数
- 检错纠错码

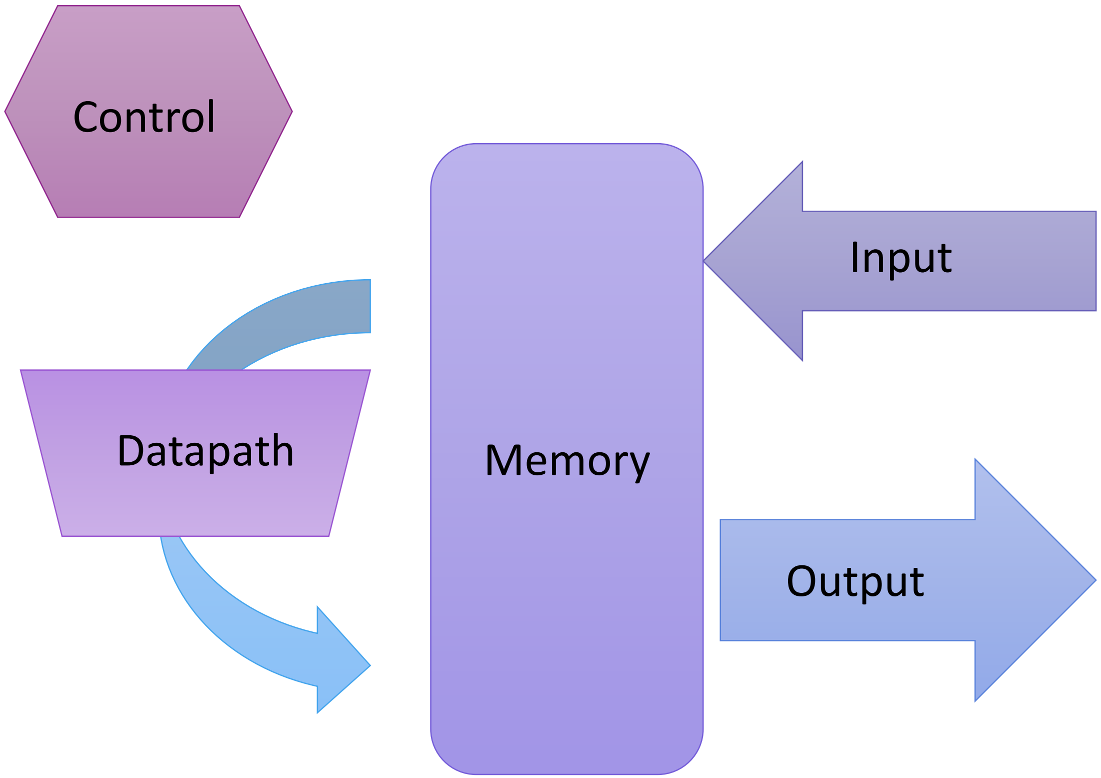
计算机是什么

- 一种高速运行的电子设备
- 用于进行数据的计算
- 可接受输入信息
- 根据用户要求对信息进行加工
- 输出结果
- A calculating machine, esp. an automatic electronic device for performing mathematic or logical operations; freq. with defining word prefixed, as analogue, digital, electronic computer.
- –Oxford English Dictionary

计算机程序

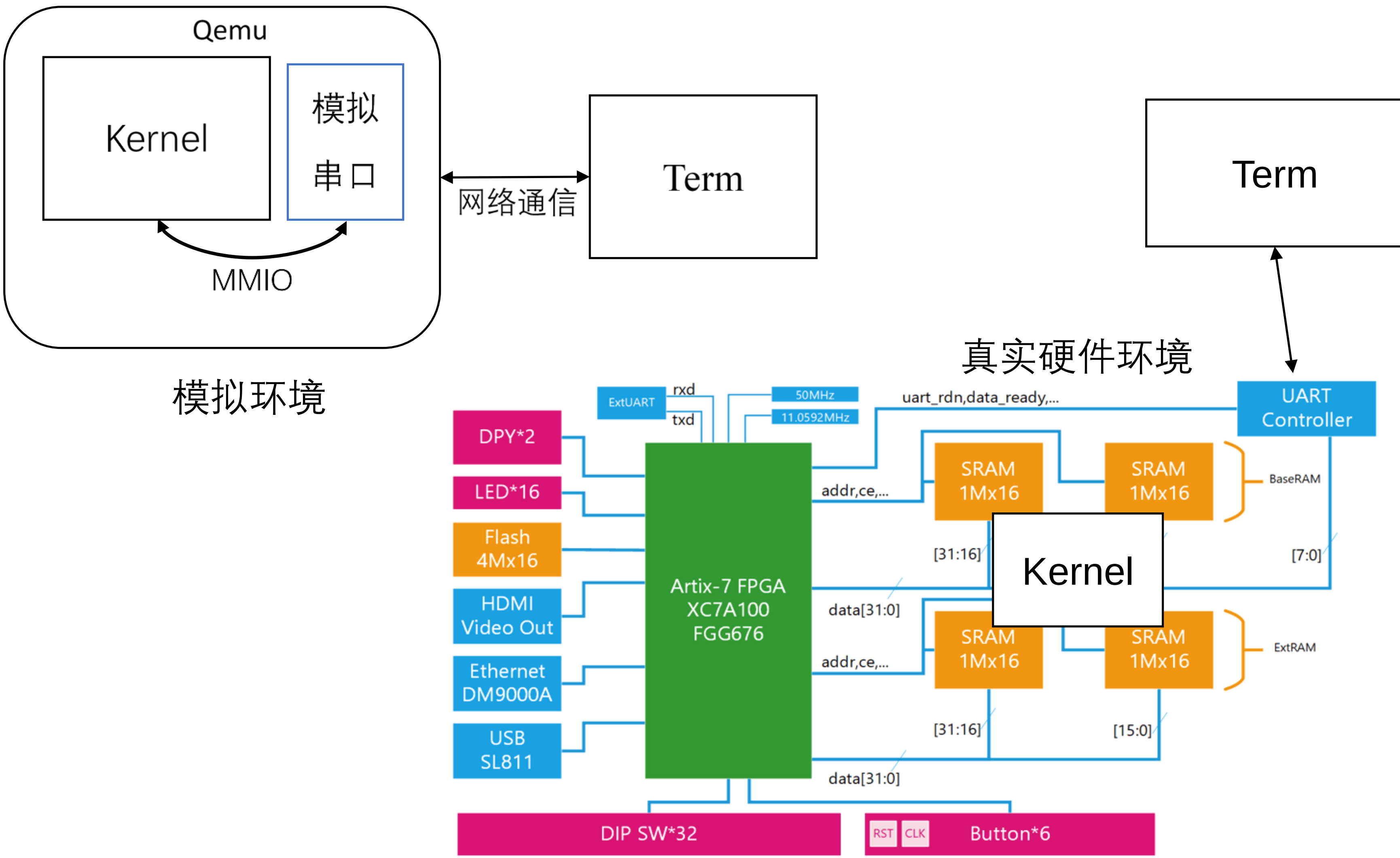
- **Computer programs**(also **software programs**, or just **programs**) are instructions for a computer. A computer requires programs to function, and a computer program does nothing unless its instructions are executed by a central processor. Computer programs are either executable programs or the source code from which executable programs are derived (e.g., compiled).
- 程序员和计算机进行交互的语言
- 计算机程序分类
  - 高级语言
  - 汇编语言
  - 机器语言

计算机运行机制



- Datapath: 完成算术和逻辑运算，通常包括其中的寄存器。
- Control: CPU的组成部分，它根据程序指令来指挥datapath，memory以及I/O运行，共同完成程序功能。
- Memory: 存放运行时程序及其所需要的数据的场所。
- Input: 信息进入计算机的设备，如键盘、鼠标等。
- Output: 将计算结果展示给用户的设备，如显示器、磁盘、打印机、喇叭等。

kernel, term, qemu的关系



监控程序的地址空间划分

虚地址区间	说明
0x80000000-0x800FFFFF	Kernel代码空间
0x80100000-0x803FFFFF	用户代码空间
0x80400000-0x807EFFFF	用户数据空间
0x807F0000-0x807FFFFF	Kernel数据空间
0x10000000-0x10000008	串口数据及状态

串口寄存器位定义

地址	位	说明
0x10000000（数据寄存器）	[7:0]	串口数据，读、写地址分别表示串口接收、发送一个字节
0x10000005（状态寄存器）	[5]	状态位，只读，为1时表示串口空闲，可发送数据
0x10000005（状态寄存器）	[0]	状态位，只读，为1时表示串口收到数据

程序设计举例（写入串口）

```
WRITE_SERIAL:                            // 写串口：将a0的低八位写入串口
    li t0, COM1
.TESTW:
    lb t1, %lo(COM_LSR_OFFSET)(t0)      // 查看串口状态
    andi t1, t1, COM_LSR_THRE           // 截取写状态位
    bne t1, zero, .WSERIAL              // 状态位非零可写进入写
    j .TESTW                            // 检测验证，忙等待
.WSERIAL:
    sb a0, %lo(COM_THR_OFFSET)(t0)      // 写入寄存器a0中的值
    jr ra
```

程序设计举例（从串口读出）

```
READ_SERIAL:                                    // 读串口：将读到的数据写入a0低八位
    li t0, COM1
.TESTR:
    lb t1, %lo(COM_LSR_OFFSET)(t0)
    andi t1, t1, COM_LSR_DR                     // 截取读状态位
    bne t1, zero, .RSERIAL                      // 状态位非零可读进入读
    j .TESTR                                    // 检测验证
.RSERIAL:
    lb a0, %lo(COM_RBR_OFFSET)(t0)
    jr ra
```

数据的编码与表示

- 需要在计算机中表示的对象
  - 程序、整数、浮点数、字符（串）、逻辑值
  - 通过编码表示
- 表示方式
  - 用数字电路的两个状态表示，存放在机器字中
  - 由上一层的抽象计算机来识别不同的内容
- 编码原则
  - 少量简单的基本符号
  - 一定的规则
  - 表示大量复杂的信息
  - 计算性能/存储空间

编码表示

- 基本元素
  - 0、1两个基本符号
- 字符
  - 26字符➔5位
  - 大/小写+其它符号➔7bits
  - 世界上其它语言的文字➔16bits（Unicode）
- 无符号整数（0，1，…，2^{n-1}）
- 有符号整数
- 浮点数
- 逻辑值
  - 0→false, 1→true
- 颜色（RGB）
- 位置/地址/指令
  - 但是：n位只能代表2^n个不同的对象

理解对象表示以及在对象上的允许的操作

逻辑型数据

- 逻辑性数据
  - True，真
  - False，假
- 数据表示
  - 1
  - 0
- 数据运算
  - 与运算
  - 或运算
  - 非运算
  - 异或运算

X	Y	X与Y	X或Y	X的非
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

字符型数据

- 重要的人机界面
  - 由符号组成
  - 为每个符号进行编码、由输入/输出设备进行转换
  - 一般以字符串的形式在计算机存储器中存放
- 字符集编码标准
  - 主机和设备、主机之间进行信息交换的基础
    - ASCII
    - UNICODE
    - UTF-8

ASCII字符编码

- American Standard Code for Information Interchange
- 采用7位二进制编码，占用一个字节
- 表示128个西文字符

ASCII码字符集

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

UNICODE字符集

- Universal Multiple-Octet Coded Character Set，简称为UCS
- 标准ISO10646
- 它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。1990年开始研发，1994年正式公布。
- 若使用16位表示一个字符（UCS-2编码, 2字节），可以表示65536个字符
  - UNICODE兼容ASCII，如在编码前插入一个0x0转UCS-2

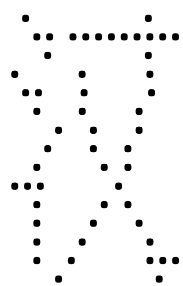
UTF-8编码

字符位数	字节1	字节2	字节3	字节4	字节5	字节6
7	0ddddddd					
11	110ddddd	10dddddd				
16	1110dddd	10dddddd	10dddddd			
21	11110ddd	10dddddd	10dddddd	10dddddd		
26	111110dd	10dddddd	10dddddd	10dddddd	10dddddd	
31	1111110d	10dddddd	10dddddd	10dddddd	10dddddd	10dddddd

- 变长字符编码，提高存储空间利用率
- 字符长度由首字节确定
- 字符首字节外，均以“10”开始，可自同步
- 可扩展性强
- 成为互联网上占统治地位的字符集

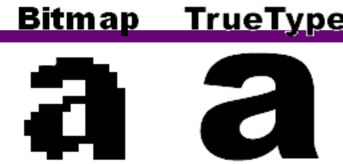
点阵字体

- 点阵本质是单色位图
- 如果为0，对应位置没有点，为1显示点
- 文字编码→查找字体文件→找到点阵→显示



	1											1			
		1	1		1	1	1	1	1	1	1	1	1		
			1									1			
1						1						1			
	1	1				1						1			
		1				1					1				
				1			1				1				
			1				1			1					
		1						1		1					
1	1	1							1						
		1						1		1					
		1					1				1				
		1				1						1			
		1			1							1	1	1	
				1									1		

矢量字体（TrueType）



- 一个字可以用多条曲线来表示，每条曲线保存其关键点
- 显示字的时候，取出这些关键点，采用平滑的曲线将这些关键点连接起来，并填充闭合空间以显示
- 需要放大或者缩小的时候，按照比例改变关键点的相对位置即可

数值型数据表示

- 定点数
  - 小数点位置固定
  - 整数
  - 定点小数

- 浮点数
  - 小数点位置浮动

数值范围和数据精度

- 数值范围
  - 数值范围是指一种类型的数据所能表示的最大值和最小值
- 数据精度
  - 通常指实数所能给出的有效数字位数；对浮点数来说，精度不够会造成误差，误差大量积累会出问题
- 机内处理
  - 数值范围和数据精度概念不同。在计算机中，它们的值与用多少个二进制位表示某种类型的数据，以及怎么对这些位进行编码有关

整数

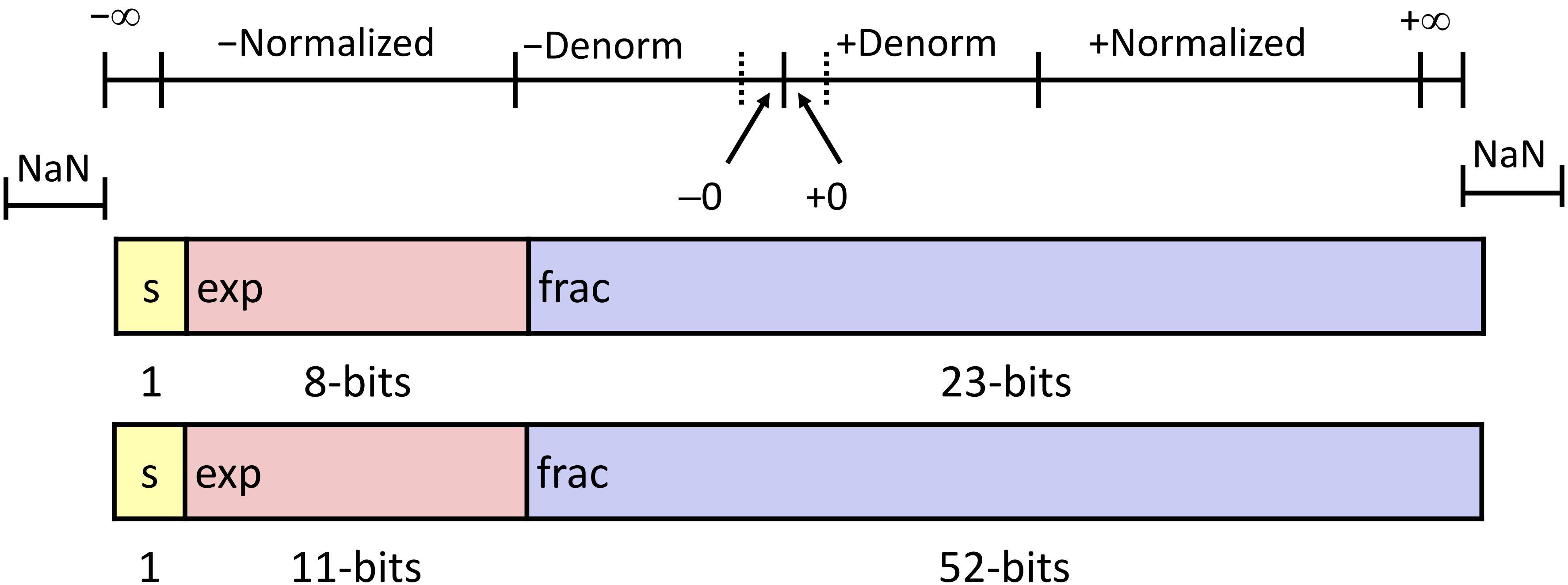
- 原码，反码，补码
- 符号扩展
- 大端，小端
- 加法，减法，乘法，除法

原码，反码，补码表示小结

- 正数的原码、反码、补码表示均相同，
  - 符号位为**0**，数值位同数的真值。
- 零的原码和反码均有**2**个编码，补码只**1**个码
- 负数的原码、反码、补码表示均不同
  - 符号位为**1**，数值位：原码为数的绝对值
  - 反码为每一位均取反码
  - 补码为反码再在最低位**+1**
  - 只有一个负数的原码与补码是相同的：1100 0000 0000 0000 0000 0000 0000 0000 （想想为什么）
- 由$[X]_{补}$求$[-X]_{补}$：每一位取反后再在最低位**+1**

浮点数

$(-1)^s\ \boldsymbol{M}\ 2^{\boldsymbol{E}}$ $Bias = 2^{k-1} - 1$



- 单精度，双精度
- 规格化数(exp≠0，exp≠111…1)，非规格化数（exp=0），0，无穷(exp=111…1, frac=000…0)，NaN(exp=111…1, frac≠000…0)

$v = (-1)^s\ \boldsymbol{M}\ 2^{\boldsymbol{E}}$
$\boldsymbol{E = exp - Bias}$
$M = 1.frac$

$v = (-1)^s\ \boldsymbol{M}\ 2^{\boldsymbol{E}}$
$\boldsymbol{E = 1 - Bias}$
$\boldsymbol{M = 0.frac}$

单精度:
k=8, Bias=127
双精度:
k=11,Bias=1023

浮点数算数运算

- 浮点数加减法运算
- $X=M_x \times 2^{Ex}$， $Y=M_Y \times 2^{Ey}$
- （1）对阶操作，求阶差：$\Delta E=E_X - E_Y$，
  - 使阶码小的数的尾数右移$|\Delta E|$位
  - 其阶码取大的阶码值；
- （2）尾数加减
- （3）规格化处理
- （4）舍入操作，可能带来又一次规格化
- （5）判结果的正确性，即检查阶码上下溢出

浮点数加运算举例

- $X=2^{+010}\times 0.1101100$，$Y=2^{+100}\times(-0.1010110)$
- 写出X、Y的正确的浮点数表示：
  - 阶码用4 位移码　尾数用8 位原码
  - （含符号位）　（含符号位）
  - $[X]_{浮}$= 0 1 010 1101100
  - $[Y]_{浮}$= 1 1 100 1010110
- 为运算方便，尾数的符号位写在数值位之前：
  - $[X]_{浮}$= 1 010 0 1101100
  - $[Y]_{浮}$= 1 100 1 1010110

浮点数加运算举例

- X=2^{+010}×0.1101100，Y=2^{+100}×(-0.1010110)
- （1）计算阶差（移码计算）：
  - $\Delta E=E_X - E_Y = E_X + (-E_Y)$= 1 010 + 0 100 = 0 110
  - 注意：阶码计算结果的符号位在此变了一次反，为-2 的移码，是X的阶码值小，使其取Y 的阶码值1100 (即+4); 因此，相应地修改$[M_X]_{原}$=0 001101100（即右移2 位）（右移出的00被保存到保护位中）
- （2）尾数求和：此处是原码加法，符号不相同，绝对值大的减小的，结果符号取决于绝对值大的数

$$
\begin{array}{r}
1\ 1010110 \\
-\ 0\ 0011011\ 00 \\
\hline
1\ 0111011\ 00
\end{array}
$$

浮点数加运算举例

- （3）规格化处理：
  - 相加结果,数值的最高位为0,应执行1 次左移操作,
  - 故得$[M_{X+Y}]_{原}$= 1 1110110, 阶码减1得1 011 (为+3)
- （4）舍入处理: 舍入位是0，按0舍1入规则，得到最终结果：1 1110110
- （5）检查溢出否：和的阶码为1011，不溢出
  - 计算后的$[X+Y]_{浮}$= 1 1011 1110110
  - 即数的实际值为：2^3×(-0.1110110)

浮点数乘除法

- 算法：
  - 阶码加减：乘法：Ex+Ey，除法Ex-Ey
  - 对尾数进行乘除法，求得结果
  - 规格化
  - 舍入，可能再次进行规格化
  - 进行溢出检查（阶码）

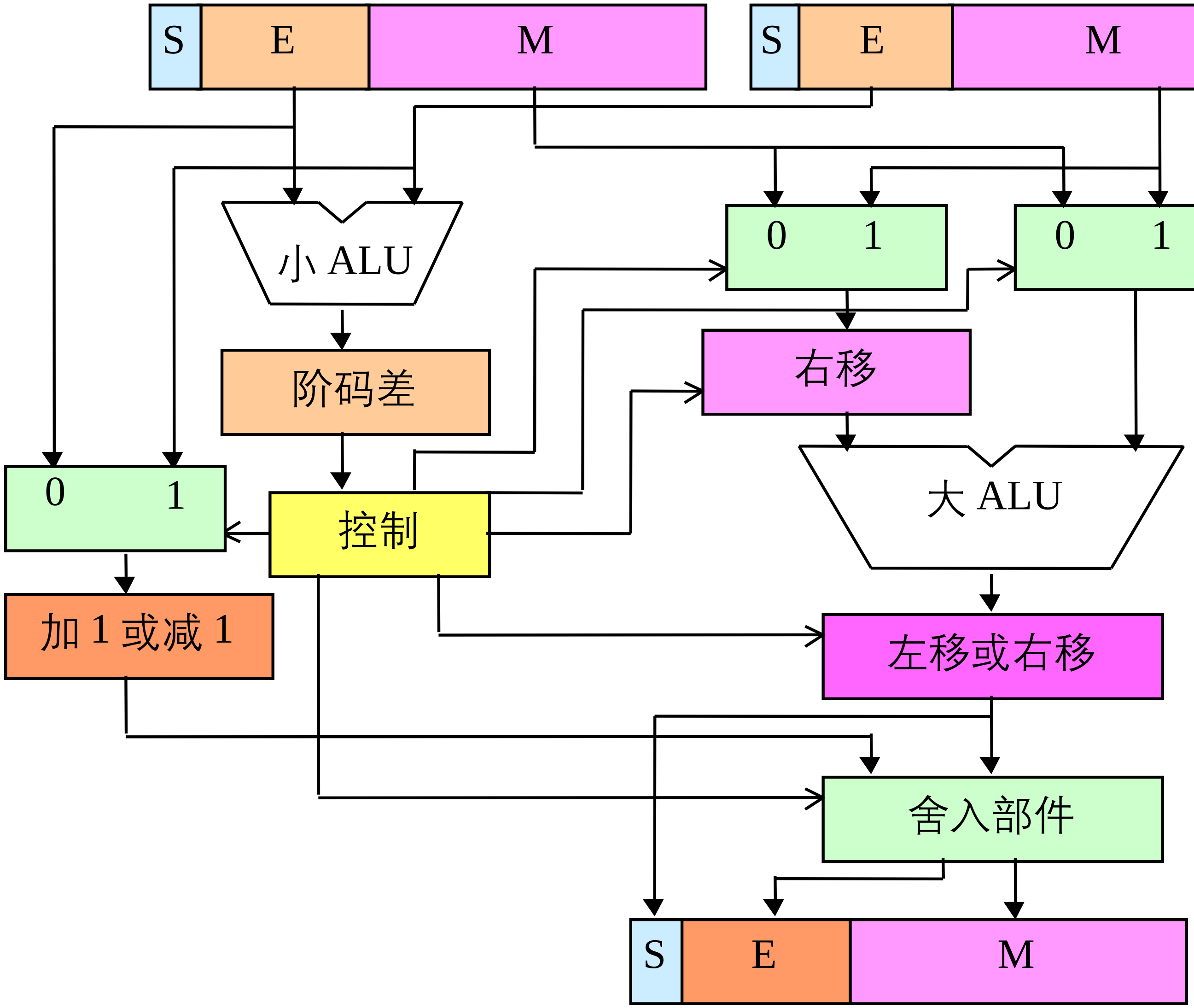
浮点数运算

- 浮点数的加减法
  - 移码的减法运算
  - 无符号数运算
- 浮点数的乘除法
  - 移码的加减运算（注意溢出）
- 浮点数尾数运算
  - 原码运算

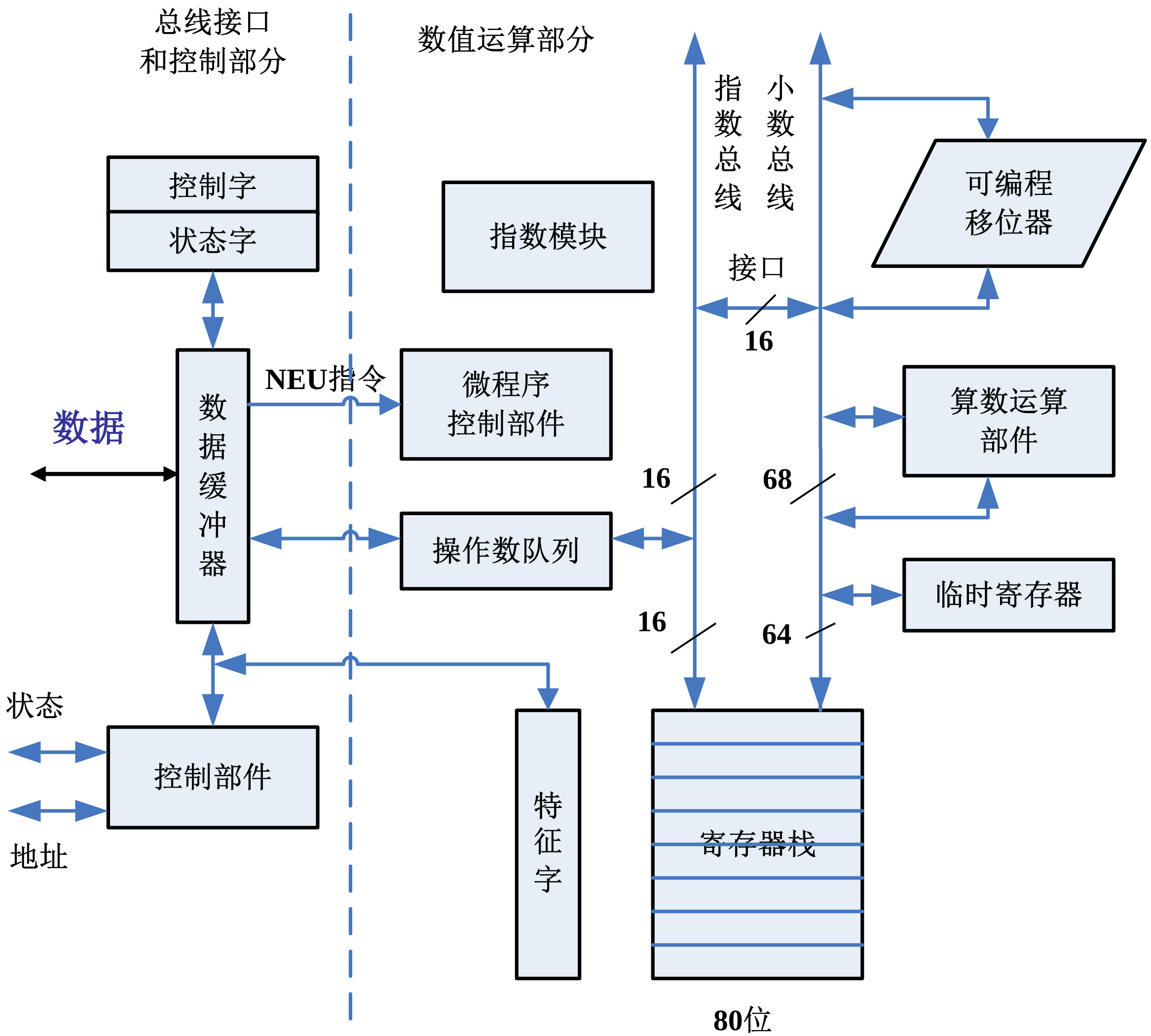
浮点运算的特点

- 浮点数的加法不满足结合律
  - $x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, and $z = 1.0$
  - $x + (y + z) = -1.5\text{x}10^{38} + (1.5\text{x}10^{38} + 1.0) = -1.5\text{x}10^{38} + (1.5\text{x}10^{38}) = 0.0$
  - $(x + y) + z = (-1.5\text{x}10^{38} + 1.5\text{x}10^{38}) + 1.0 = (0.0) + 1.0 = 1.0$
- 浮点数加法不可结合
- 浮点数的相等比较：小心！只是近似
- for(i=0;i!=10;i+=0.1)

浮点运算部件



浮点运算器举例-Intel 80287



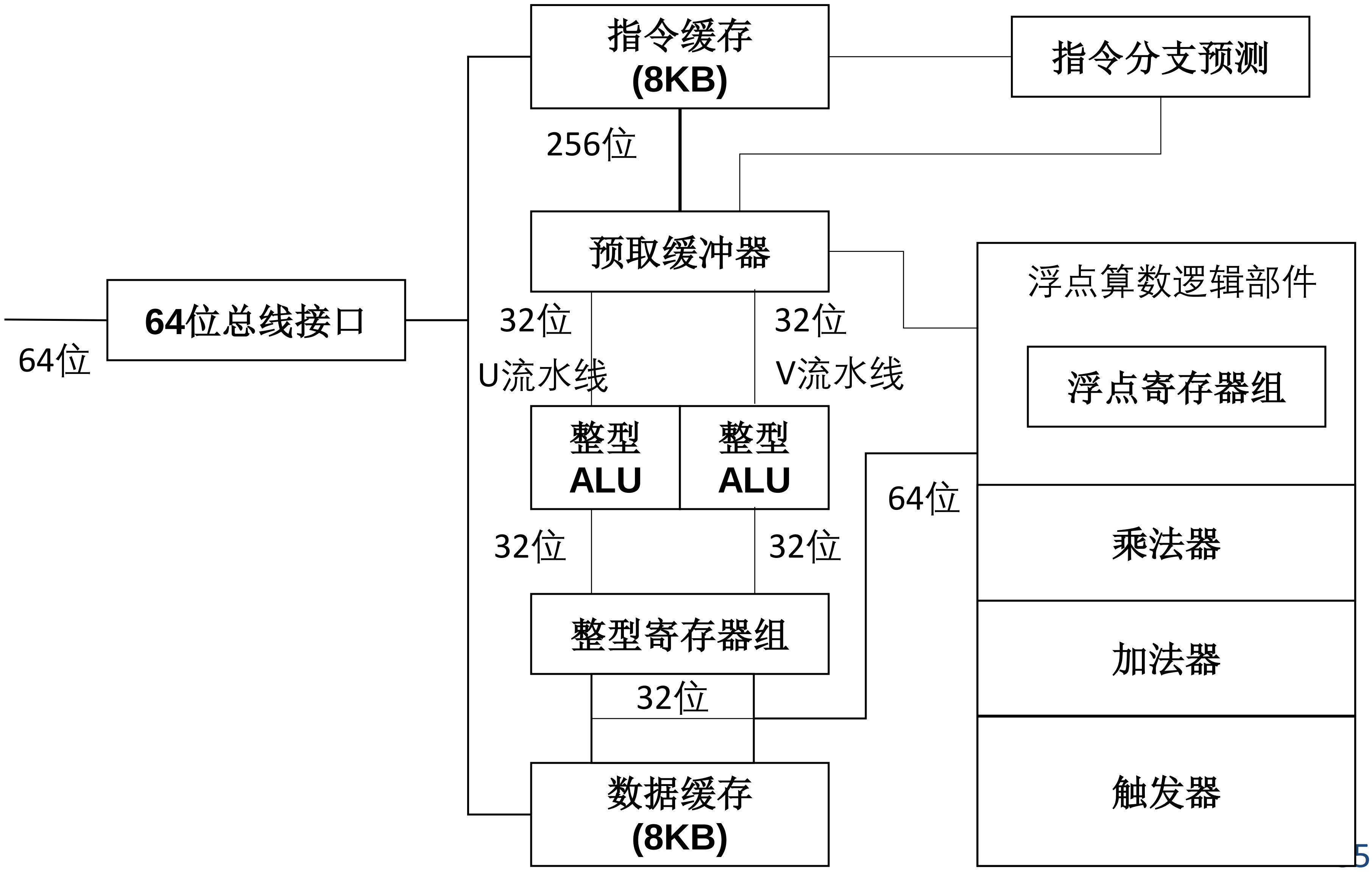
浮点运算部件以协处理器方式和CPU 连接，有独立的控制逻辑；

8个 80位 浮点数寄存器，精度更高，采用堆栈结构并进行了扩展；

支持 3大类共 7 种数据，支持约 60 条指令；

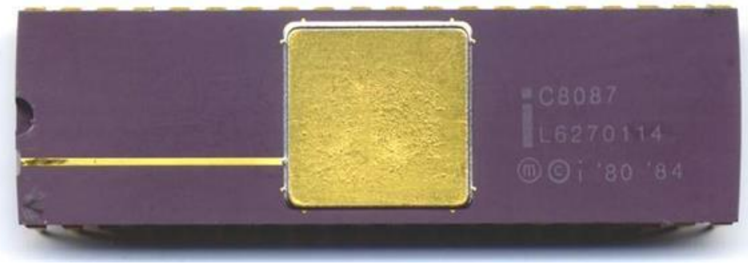
在后来的奔腾机中有重大改进。

Pentium结构简图



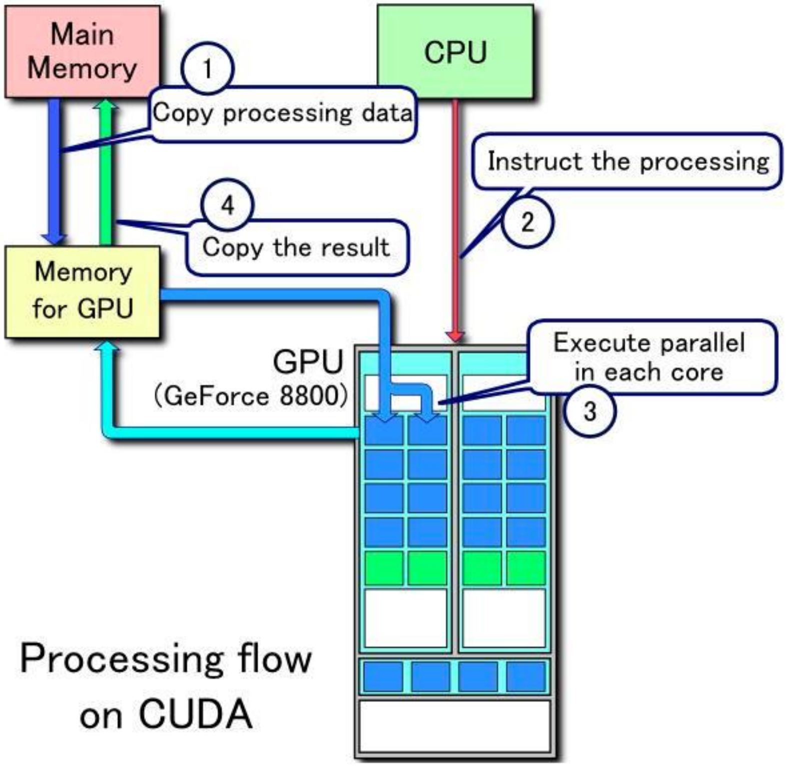
现代处理器对浮点数运算的支持

- Intel设计了独立于8086和8088处理器外的8087数学辅助处理器
- Intel在80486DX处理器核心内首次集成了浮点运算单元
- 在现代处理器中，浮点计算功能会通过SIMD（Single Instruction Multiple Data，单指令多数据流）的技术实现并行计算能力
- MMX，SSE1~4，AVX1~2~512，FMA



现代的高速浮点计算单元

- GPGPU, General Purpose Graphic Processing Unit
- CUDA
- OpenCL
- CPU中的浮点运算单元是为了更高精度浮点运算设计的。Intel-AVX指令集处理512位扩展数据
- GPU中的处理器都是为高度并行计算而设计的，在Nvidia以及AMD图形处理器上支持的数据精度大多是单精度和双精度浮点计算（FP32和FP64），甚至随着机器学习，深度学习，神经网络的流行，最新的图形处理器甚至支持了半精度浮点运算（FP16）。

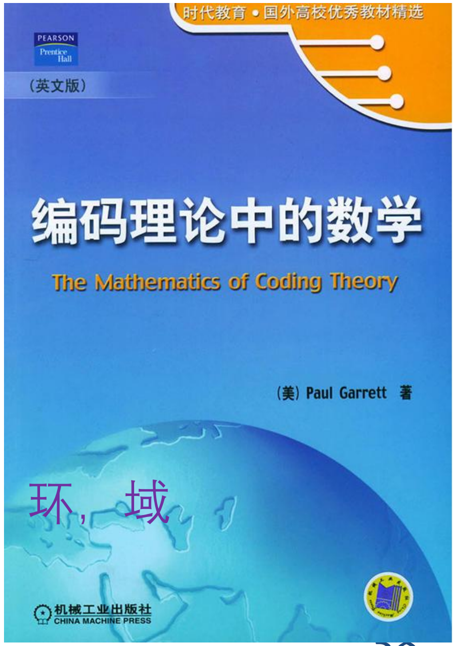


检错纠错码

- 数据或编码在存储、传输等过程中可能出错（甚至可能丢失）
- 如何判断已经出错
  - 比较：与所有正确的编码进行比较
  - 特征：检验是否存在某些特征（预先放置到编码中）
- 发现错误之后能否自动纠正？

- 计算机种的数据如何进行检错？
- 纠错？
- 纠删码（Erasure Code）
  - 丢失之后恢复（存储）

深入了解
基础:
- 代数结构：群，环，域
- 线性代数



检错纠错码

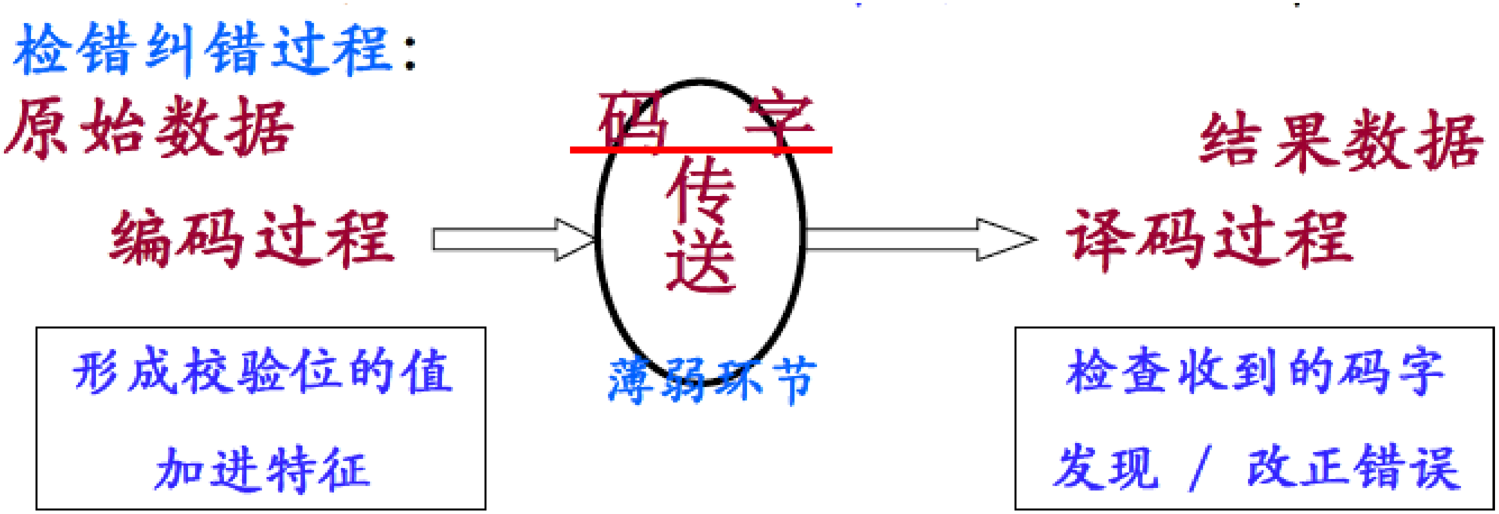
- 使编码具有某种特征，通过检查这种特征是否存在来判断编码是否正确
- 出错时，如果还能指出是哪位出错，则可以纠正错误
  - 编码
  - 检查
  - 出错后纠正

码距

- 码距（最小码距）的概念：是指任意两个合法码之间至少有几个二进制位不相同。
  - 仅有一位不同的编码是无纠错能力的。例如用4位二进制表示16种状态，则16种编码都用到了，此时码距为1。任意一个编码状态的四位码中的一位或者几位出错，都会变成另外一个合法码。这种编码无检错能力。
  - 若用4个二进制位表示8种合法的状态，就可以只使用其中的8个编码来表示，另外8个为非法编码。此时可以使合法码的码距为2。任何一位出错后都会成为非法码，有发现一位出错的能力
- 合理增大码距，能提高发现错误的能力，但表示一定数量的合法码所使用的二进制位数要变多，增加了电子线路的复杂性和数据存储、数据传送的数量。

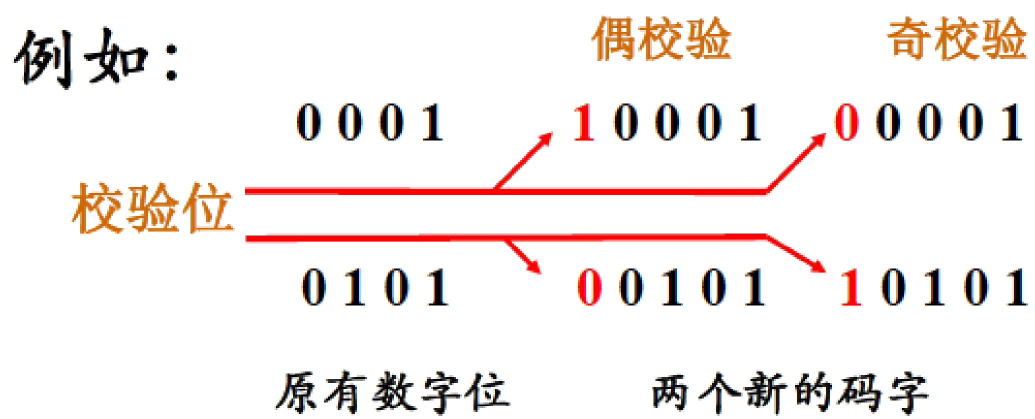
常用检错纠错码

- 数据编码➔ 数据传输 ➔ 数据译码
- 三种常用的检错纠错码：
- 奇偶校验码：并行数据传输
- 海明校验码：并行数据传输
- 循环冗余校验码：串行数据传输

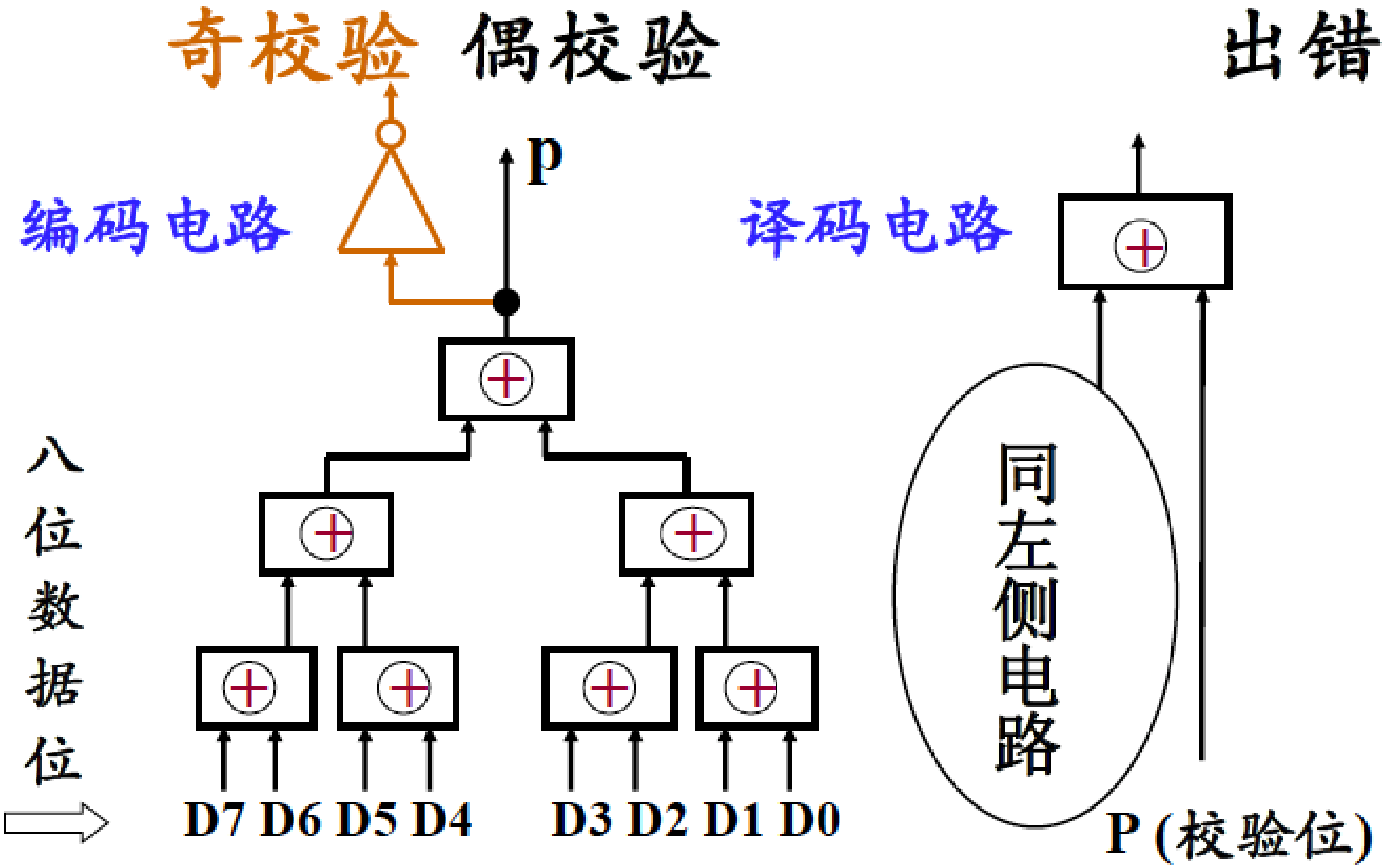


奇偶校验码

- 奇校验，偶校验
- 用于并行码检错
- 原理：在k 位数据码之外增加1 位校验位，
- 使K+1 位码字中取值为1 的位数总保持为偶数（偶校验）或奇数（奇校验）。
- 有效数字：0001
- 有效数字：0101



奇偶校验的实现电路



海明校验码

- 用于多位并行数据检错纠错处理
- 实现：为k个数据位设立r个校验位，使k+r位的码字同时具有这样两个特性
  - 能发现并改正k+r位中任何一位出错
  - 能发现k+r位中任何二位同时出错，但已无法改正

- k与r之间应该满足什么样的关系？

海明码的编码方法

- 合理的使用k位数据形成r个校验位的值，保证用k个数据中不同的数据位组合来形成每个校验位的值，使任何一个数据位出错时，将影响r个校验位中不同的校验位组合起变化。这样一来，就可以通过检查哪种校验位组合起了变化，来推断是那个数据位错误造成的，对该位求反则实现纠错
- 有的时候两位出错与某种情况的一位出错对校验位组合的影响相同，必须加以区分与解决
- 位数r和k的关系：$2^r \geq k+r+1$，即用2^r个编码分别表示k个数据位，r个校验位中哪一位出错，都不错
- $2^{r-1} \geq k+r$，用r-1位校验码为出错位编码，再单独设一位用以区分1位还是2位同时出错，更实用

海明码的实现

例如: k =3, r =4

D3	D2	D1	P4	P3	P2	P1
1	1	1	1	1	1	1
1	1	0	0	1	0	0
1	0	1	0	0	1	0
0	1	1	0	0	0	1

$\oplus$ 表示异或

$P1 = D2 \oplus D1$

$P2 = D3 \oplus D1$

$P3 = D3 \oplus D2$

编码方案 $P4 = P3 \oplus P2 \oplus P1 \oplus D3 \oplus D2 \oplus D1$

译码方案

$S1 = P1 \oplus D2 \oplus D1$

$S2 = P2 \oplus D3 \oplus D1$

$S3 = P3 \oplus D3 \oplus D2$

$S4 = P4 \oplus P3 \oplus P2 \oplus P1 \oplus D3 \oplus D2 \oplus D1$

海明码的实现方案

- 如何分配不同的数据位组合来形成每个校验位的值
- P1 P2 D1 P3 D2 D3 P4
- 1 2 3 4 5 6 编码方案

- （一）准备工作：
- (1) 从1~6按次序排列数据位、校验位，
- (2) 将校验位P1、P2、P3依次安排在2的幂次方位。
- (3) P4为总校验位，暂不考虑。

海明码的实现方案

- 如何分配不同的数据位组合来形成每个校验位的值
- **P1 P2 D1 P3 D2 D3 P4**
- **1 2 3 4 5 6** 编码方案

- （二）为各校验位分配数据位组合：
- (1) 看数据位的编号分别为3、5、6，它们是校验位编号的组合：
- 3=1+2、5= 1+4、6= 2+4
- (2) 1出现在3和5中，则P1负责对D1和D2进行校验。
- (3) 2出现在3和6中，则P2负责对D1和D3进行校验。
- (4) 4出现在5和6中，则P3负责对D2和D3进行校验。

海明码的实现方案

- 如何分配不同的数据位组合来形成每个校验位的值
- **P1 P2 D1 P3 D2 D3 P4**
- **1 2 3 4 5 6** 编码方案

- （三）写出各校验位的编码逻辑表达式：
- (1) 结果是:
- $P1 = D2 \oplus D1$；$P2 = D3 \oplus D1$；$P3 = D3 \oplus D2$
- (2) 用其他各校验位及各数据位进行异或运算求校验位 P4 的值，用于区分无错、奇数位错、偶数位错3 种情况
- 总校验位$P4 = P3 \oplus P2 \oplus P1 \oplus D3 \oplus D2 \oplus D1$

海明码的译码方案

- 译码方案是：
  - 对接收到数据位再次编码，用得到的结果和传送过来的校验位的值相比较，二者相同表明无错，不同是有**1** 位错了。或者将校验位与对应数据位进行异或，获得**S4~S1**值
- 排查是哪一位错了，看**S4~S1** 这**4** 位的编码值

海明码的应用实例

- 如已有数据为110，编码为:P1P2 1 P3 10 P4则有：
- P1=0，P2=1 P3=1，P4=0

请看如下 3 种情况：

无错，	单独 1 位错，	2 位同时错
若无错，则	若仅 D1 错，则	若P2 D1错，则
S4 S3 S2 S1=0000	S4 S3 S2 S1=1011	S4 S3 S2 S1=0001
4 位 S 全为 0	S3S2S1 不为 000 其中 S4 必为 1	其中 S4 必为 0, S3S2S1 不为 000

更多的海明码

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		**p1**	**p2**	**d1**	**p4**	**d2**	**d3**	**d4**	**p8**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**p16**	**d12**	**d13**	**d14**	**d15**	
Parity bit coverage	**p1**	X		X		X		X		X		X		X		X		X		X		
	p2		X	X			X	X			X	X			X	X			X	X		...
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

检错纠错码小结

- （1）k位码有2^K个编码状态，全用于表示合法码，则任何一位出错，均会变成另一个合法码，不具有检错能力
- （2）从一个合法码变成另外一个合法码，至少改变几位码的值，称为最小码距（码距），码距和编码方案将决定其检错纠错能力。
  - 奇偶校验码的码距为2
  - 海明码的码距为4

检错纠错能力

- （3）k+1位码，只用其2^k个状态，可以使码距为2，如果一个合法码中的一位错了，就称为非法码，通过检查码字的合法性，就得到检错能力，这就是奇偶校验码，只能发现1位错，不具备纠错能力
- （4）对于k位数据位，当给出r位校验位时，要发现并改正一位错，须满足如下关系：
  - $2^r \geq k+r+1$
- 要发现并改正一位错，也能发现两位错，则应：
  - $2^{r-1} \geq k+r$

小结

- 数据表示
  - 通过二进制编码表示数据
  - 逻辑型
  - 字符型
  - 整数
- 检错和纠错
  - 通过冗余的编码，使之满足某些规则，来检查编码在传输中是否发生错误，并进行改正
  - 检错纠错能力

阅读及思考

- 阅读
- 思考
  - 原，反，补码的定义及实现算数运算难易比较
  - 试证明补码的性质
  - 试推导海明码校验位r和数据位k的关系
  - 补码算术运算如何使用硬件来实现？

谢谢

backup

有关整数的相关内容

整数的二进制表示

- 二进制的两个状态0和1
- n位可得到2^n种组合，可表示2^n个整数
- 那么，如何来表示呢？

机内表示	真值
0	0
01	1
10	2
11	3
100	4
101	5
…	…
1{n}	2^n-1

评价标准

- 无符号数
  - 表示范围
  - 直观
  - 便于算术运算的实现
- 有符号数
  - 表示范围
  - 直观
  - 正、负数平衡
  - 便于算术运算的实现

进位计数法

- 进位计数法

$$N = \sum_{i=m}^{-k} D_i r^i$$

  - N表示某个数值
  - r是这个数制的基
  - i表示这些符号排列的位号
  - Di是位号为i的位上的一个符号
  - r^i是位号为i的位上的一个1代表的值
- 常用的进制
  - 十进制、二进制、八进制、十六进制

数制与进位记数法

- 二进制
  - r=2，基本符号：0 1
- 八进制
  - r=8，基本符号：0 1 2 3 4 5 6 7
- 十进制
  - r=10，基本符号：0 1 2 3 4 5 6 7 8 9
- 十六进制
  - r=16，基本符号：0 1 2 3 4 5 6 7 8 9 A B C D E F
- 计算机采用二进制

数制转换

把二进制数转换为十进制数，

累加二进制数中全部数值为 1 的那些位的位权

$$(1101.1100)_2=(1\times2^3+1\times2^2+0\times2^1+1\times2^0)_{10}$$

$$+(1\times2^{-1}+1\times2^{-2}+0\times2^{-3}+0\times2^{-4})_{10}=(13.75)_{10}$$

把二进制数转换成八或十六进制数时，从小数点向左和向右把每 3 或者 4 个二进制位分成一组，直接写出每一组所代表的数值，小数点后不足位数补0。

$(1101.1001)_2=(D.9)_{16}=(15.44)_8$，而不是$(15.41)_8$

数制转换

二进位数和十进制数之间的转换方法

二进制：$r=2$，　基本符号：0 1

十进制：$r=10$，基本符号：0 1 2 3 4 5 6 7 8 9

求二进制数所对应的十进制数值，可通过进位记数公式来计算，即把取值为 1 的数位的位权累加。

把十进制数转换为二进制，对整数部分通过除 2 取余数来完成，对小数部分通过乘 2 取整数来完成。

2	13--------1	低位	
2	6--------0		
2	3--------1		
2	1--------1	高位	
	0		

$(13)_{10}=(1101)_2$

	0.76×2	
1	0.52×2	高位
1	0.04×2	
0	0.08×2	
0	0.16	低位

$(0.76)_{10}=(0.1100)_2$

整数的二进制表示

- 无符号的整数
  - 无符号整数每一个1有位权，所有非零位权的和即为最终的数值

- 有符号的整数
  - 需要一种机制来表达负数

无符号整数

- 字节
- 字
- 无符号整数
- 无符号长整数

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

- unsigned char, unsigned short, unsigned int, unsigned long

- sizeof

无符号整数表示范围

- sizeof(unsigned int) = 4
- sizeof(unsigned long) = 8

- 最小值：0
  - #define UCHAR_MAX   255   /* max value for an unsigned char */
  - #define USHRT_MAX   65535   /* max value for an unsigned short */
  - #define UINT_MAX   0xffffffff /* max value for an unsigned int */
  - #define ULONG_MAX   0xffffffffffffffffUL   /* max unsigned long */

有符号整数

- 需要有1位来表示符号
- 最高位
- 0表示正数、1表示负数
- 其他位表示数据

有符号整数

- 主要是负数如何表达
  - 使用原码表示：最高位是符号位
  - 使用反码表示：绝对值的每一位取反为负数
  - 使用补码表示：

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

    - 补码快速计算：绝对值各位取反加1，（如何证明？）
  - 原码和反码中有两个0
  - 补码中的加法

  - $[x+y]_补=[x]_补+[y]_补$
  - $[x-y]_补=[x+(-y)]_补= [x]_补+[-y]_补$

原码，反码，补码

编码	原码	反码	补码
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	-0	-3	-4
101	-1	-2	-3
110	-2	-1	-2
111	-3	-0	-1

- 负数表示形式：
- 原码（Sign Magnitude）：符号位||数的绝对值
- 反码（One’s Complement）：符号位||数值按位求反
- 补码（Two’s Complement）：反码的最低位+1

整数编码的定义

x 为真值　　n 为整数的位数

$$[x]_{原} = \begin{cases} x & 2^n > x \geq 0 \\ 2^n - x & 0 \geq x > -2^n \end{cases}$$

$$[x]_{补} = \begin{cases} x & 2^n > x \geq 0 \\ 2^{n+1} + x & 0 \geq x \geq -2^n \pmod{2^{n+1}} \end{cases}$$

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

补码的性质

- 补码与真值的对应
  - 补码求真值

$$N = -b_{n-1} * 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

  - 真值求补码
    - 正数的补码是绝对值原码
    - 负数的补码是绝对值原码按位求反后，再在最低位加1
- 补码的加法的运算
  - 加法运算：符号位和数据位同样计算
  - $[x+y]_{补}=[x]_{补}+[y]_{补}$

补码的性质

- $[x]_补$与$[-x]_补$
  - $[x]_补$连同符号位在内，逐位求反，再在最低位加1，即可得$[-x]_补$
  - 当X>=0时，…
  - 当X<0时，…
- 补码减法
  - $[x-y]_补= [x+(-y)]_补= [x]_补+[-y]_补$
- 补码的乘除法

补码表示中的符号位扩展

- int y = -100; long x = (long) y; 符号位扩展到高位

由 $[X]_{补}$ 求 $[X/2]_{补}$ 的方法

原符号位不变，且符号位与数值位均右移一位

例如，

$[X]_{补}=\underline{10010}$ 则 $[X/2]_{补}=1\underline{10010}$

不同位数的整数补码相加减时，要进行符号扩展

位数少的补码数的符号位向左扩展，

一直扩展到与另一数的符号位对齐。

$$\begin{array}{r} 0101010111000011 \\ +\ 11111111\ 10011100 \\ \hline 0101010101011111 \end{array} \qquad \begin{array}{r} 0101010111000011 \\ +\ 00000000\ 00011100 \\ \hline 0101010111011111 \end{array}$$

大端机与小端机

大端存储

- 数据的低位保存在内存的高地址字节中
- 数据的高位保存在内存的低地址字节中
- 例如：32位整数“12345678”保存在内存4000起始地址

内存地址	4000	4001	4002	4003
存放数据	12	34	56	78

小端存储

- 数据的高位保存在内存的高地址字节中
- 数据的低位保存在内存的低地址字节中
- 例如：32位整数“12345678”保存在内存4000起始地址

内存地址	4000	4001	4002	4003
存放数据	78	56	34	12

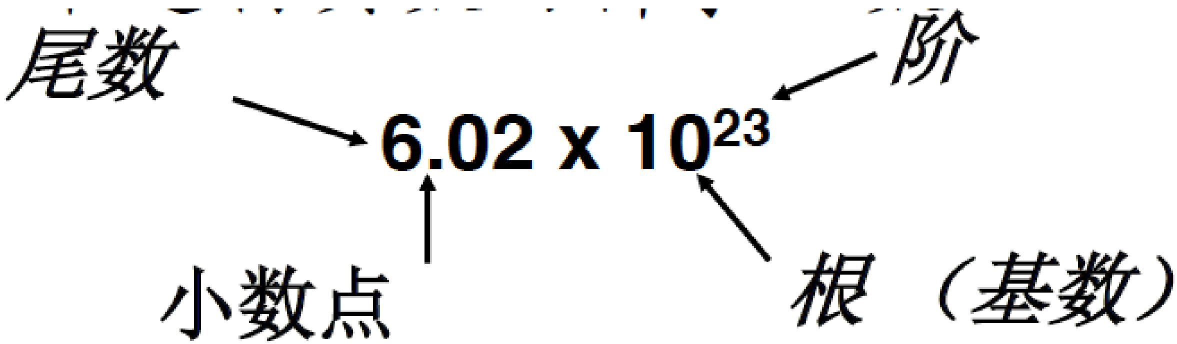
原码，反码，补码表示小结

- 正数的原码、反码、补码表示均相同，
- 符号位为**0**，数值位同数的真值。
- 零的原码和反码均有**2**个编码，补码只**1**个码
- 负数的原码、反码、补码表示均不同，
  - 符号位为**1**，数值位：原码为数的绝对值
  - 反码为每一位均取反码
  - 补码为反码再在最低位**+1**
- 由$[X]_{补}$求$[-X]_{补}$：每一位取反后再在最低位**+1**

有关浮点数的相关内容

实数的表示

- 实数
  - 无限，连续，表示不唯一
- 十进制实数的科学计数法



- 固定小数点的位置，表示方法唯一
- 主要包括3个部分
  - 尾数，阶，根

二进制浮点数科学记数法

尾数　　　　　　　　　　阶

$$1.0_2 \times 2^{-1}$$

“二进制小数点”　根（基数）

- 需要计算机内表示的部分
  - 尾数
  - 阶

浮点数表示

- 浮点数是数学中实数的子集合，由一个纯小数乘上一个指数值两部分组成。在计算机内，其纯小数部分被称为浮点数的尾数，对非0 值的浮点数，要求尾数的绝对值必须≥1，称满足这种表示要求的浮点数为规格化表示；
- 把不满足这一表示要求的尾数，变成满足这一要求的尾数的操作过程，叫作浮点数的规格化处理，通过移位尾数和修改阶码实现。
- 尾数包括有：符号位+尾数的绝对值
- 阶码：符号位+阶码的绝对值

浮点数的机器表示

- 尾数：定点小数
- 阶码：整数
- 如何表示？
  - $X = M_S E_S E_m \ldots E_2 E_1 M_{-1} M_{-2} \ldots M_{-n}$
  - (n+1)位尾数
  - (m+1)位阶码
  - 表示范围和表示精度？

IEEE浮点数标准754

- 浮点数: $X = M_S E_S E_m \ldots E_2 E_1 M_{-1} M_{-2} \ldots M_{-n}$
- IEEE 标准：阶码用移码, 对规格化数阶码用移127方案
- 尾数用原码,对规格化数的尾数用隐藏位技术
- 支持正负无穷大的浮点数和非规格化的浮点数
- 设置一个非法浮点数编码供编程人员用于排错

	符号位数	阶码位数	尾数位数	总位数
短浮点数:	1	8	23	32
长浮点数:	1	11	52	64

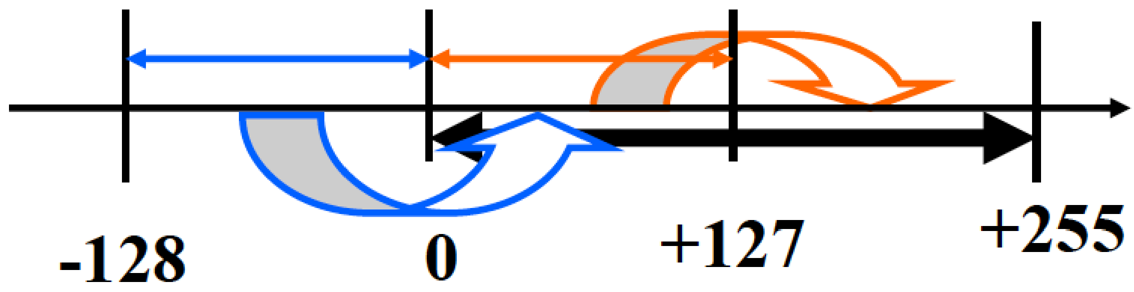
浮点数的尾数部分

- 浮点数: X =$M_S E_S E_m \ldots E_2 E_1 M_{-1} M_{-2} \ldots M_{-n}$
- IEEE 标准：阶码用移码，基为2；

- 尾数用原码表示
- 按**IEEE**的浮点数标准，尾数用规格化原码表示**,**即符号位**Ms**用**0** 表示正，**1** 表示负，且非**0** 值尾数数值的最高位必定为**1**；既然这位必定为**1**，则在保存浮点数到内存前，通过尾数左移**,** 强行把该位去掉**,** 则用同样多的尾数位就能多存一位二进制数，有利于提高数据表示精度，把这种处理方案称作为隐藏位技术。当然，在取回这样的浮点数到运算器执行运算时**,**必须先恢复该隐藏位。

阶码的移码表示法

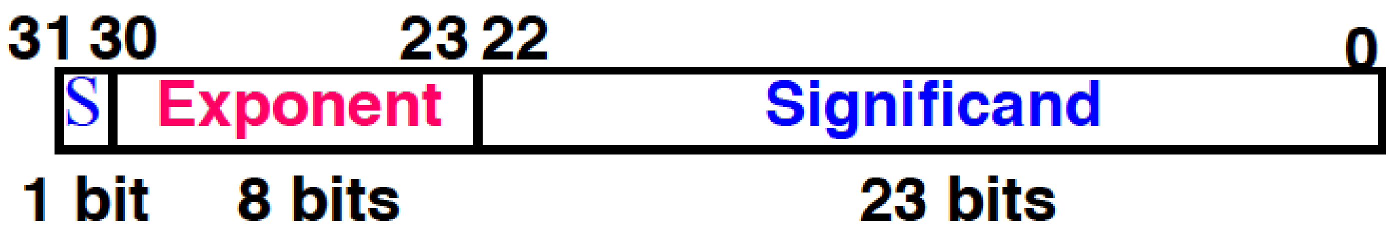
- 移码：整数补码+偏移值
  - $[E]_{移}$=E+OFFset　　　　$-2^n<=E<2^n$
  - 使浮点数0的机器表示为全0



- 对移127 的方案，8 位移码表示的机器数是数的真值在数轴上向右平移了127 个位置。

IEEE 754浮点数标准

- 规格化形式：$+1.xxxxxxxxxx_2 * 2^{yyyy_2}$
- 单精度浮点数(32 bits)



- S表示符号位
- Exponent 表示y，即阶，移127
- Significand表示x，即尾数的后部分
- 可表示的范围：2.0×10^{-38}至2.0×10^{38}

IEEE 754浮点数标准

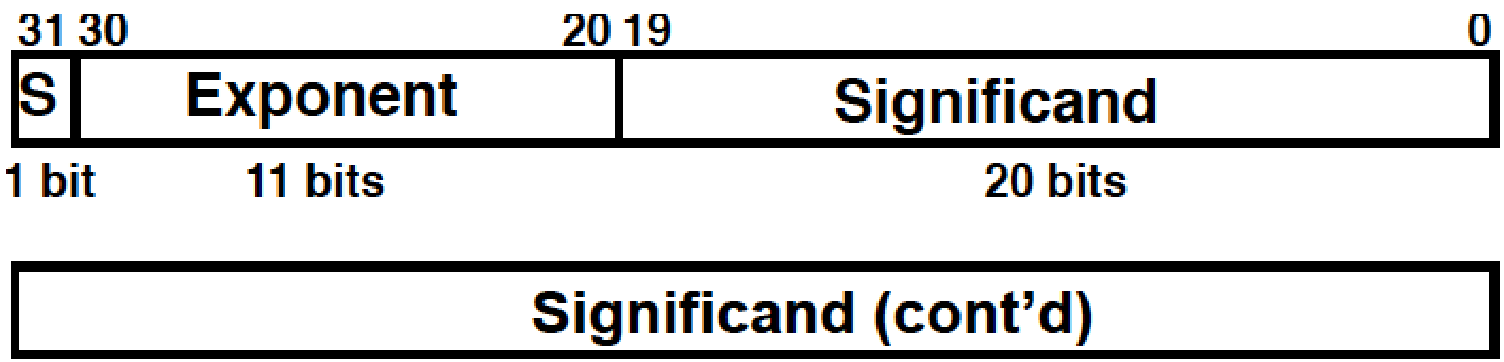
- 规定对长、短浮点数的尾数使用隐藏位技术，即把规格化非0 值尾数的最高位上的1 经过左移操作后强行去掉，则原来不能表示的更低一位进到最低一位。对单精度浮点数采用隐藏位之后，就使23 位的规格化尾数数值位能给出24 位的精度。
- 短浮点数采用移127的方案, 阶码值范围: 00000001 ~11111110, 表示-126~+127。还有2 个特定的阶码值:
- 00000000跟23位的非0 尾数表示非规格化浮点数(隐藏位必定为0)，00000000 跟全0尾数是浮点数0。
- 11111111跟全0 尾数表示无穷大的浮点数，可正可负，由符号位决定。11111111 跟非全0尾数时属于非法数值。

IEEE 754浮点数标准

S(1位)	E(8位)	M(23位)	X(共32位)
符号位	0	0	0
符号位	0	不等于0	$(-1)^S \cdot 2^{-126} \cdot (0.M)$ 非规格化
符号位	1到254之间	不等于0	$(-1)^S \cdot 2^{E-127} \cdot (1.M)$ 规格化
符号位	255	0	无穷大
符号位	255	不等于0	NaN(非数值)

- 鉴于IEEE754标准对计算机界的重要贡献，发挥关键作用的数学家Kahan 于1989年被授予图灵奖。

双精度浮点数



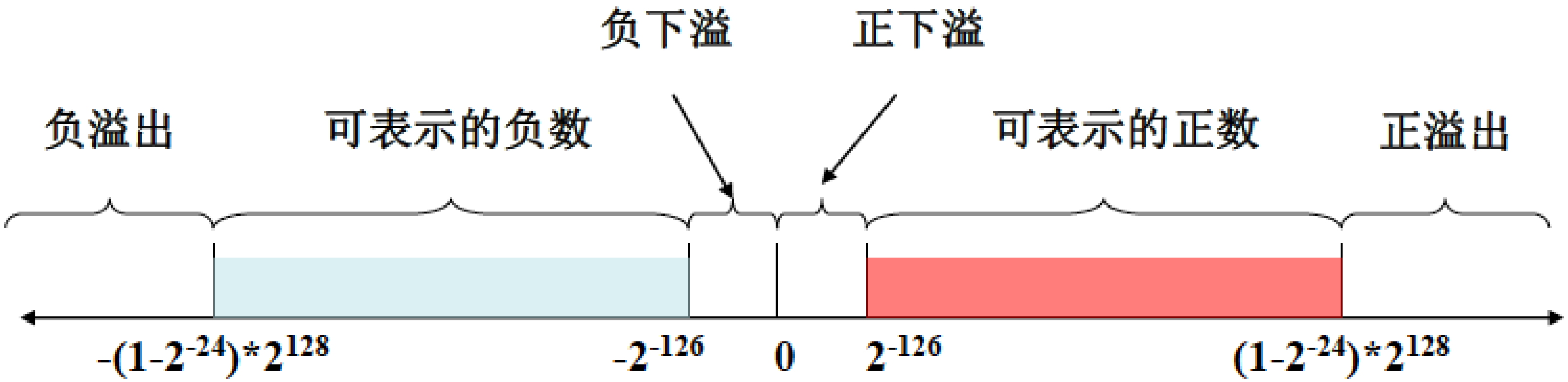
- C 语言中的double 类型
  - 尾数：原码
  - 阶：移1023
- 十进制的范围扩展到2.0 x 10^{-308}至2.0 x 10^{308}
- 最主要的好处是精度得到了扩展(52 位)

IEEE 754浮点数标准（总结）

- 被几乎所有计算机采纳**(**自**1985**年起**)**
- 符号位：1表示负数，0表示正数
- 有效位：
  - 使用原码表示
  - 规格化小数中，隐含最高位**1**
  - 单精度为：**23** 位，双精度为**52** 位
  - **0 <** 有效数**< 1**
- 阶：单精度：移**127**，双精度：移**1023**
- 全**0**用来表示**0**值
- 在阶码中保留**0**

$$(-1)^S * (1 + \text{Significand}) * 2^{\text{Exp}}$$

特殊的浮点数值



特殊值	阶	有效数
+/- 0	0000 0000	0
非规格化数	0000 0000	非0
NaN	1111 1111	非0
+/- ∞	1111 1111	0

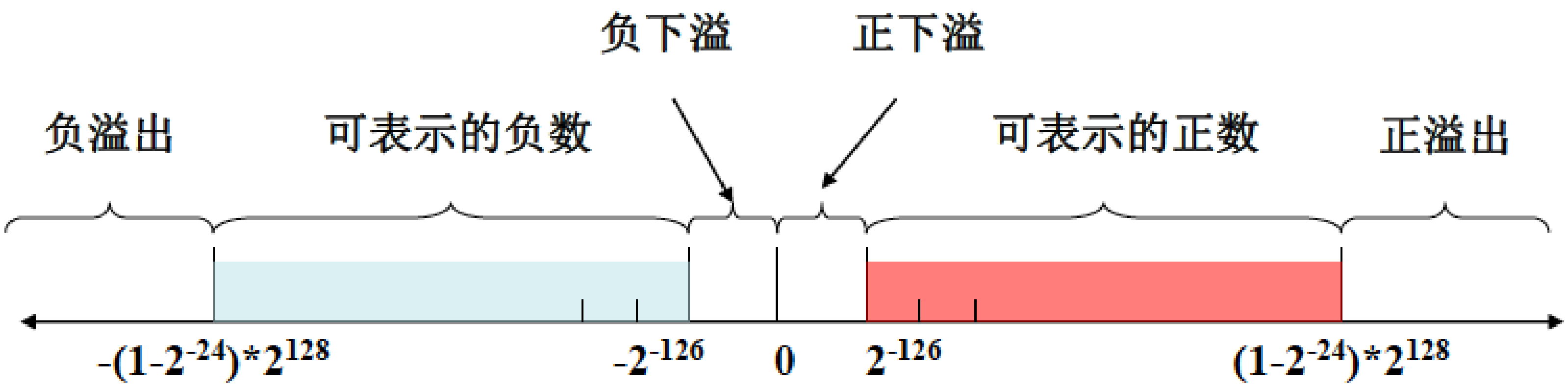
Not A Number

- 下列结果是什么: sqrt(-4.0)or 0/0?
  - 如果无穷大不是错误的话，那以上也不算
  - 称其为Nota Number (NaN)
  - 阶= 255, 有效位非0
- 应用
  - NaN可帮助排错
  - 自包含：op(NaN, X) = NaN

上溢和下溢

- 上溢
  - 数的绝对值太大（> 2.0x10^{38}）
  - 阶的值超出8位能表示的范围
- 下溢
  - 数的绝对值太小
  - >0, < 2.0x10^{-38}
  - 阶码超出了8位二进制位能表示的范围
- 如何减少上溢和下溢?
- 如何提高数据表示的精度?

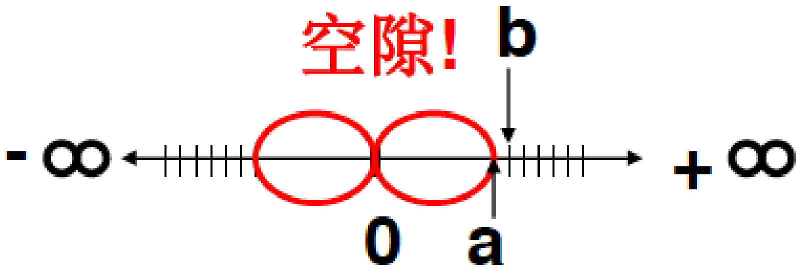
浮点数溢出



- 可表示的大于0的最小规格化数（隐藏位的值为1）：
- 正数：0 00000001 00000000000000000000000
- 可表示的大于0的最小非规格化数（不使用隐藏位技术）：
- 正数：0 00000000 00000000000000000000001

非规格化数

- 问题：在0周围还有一些空隙没有用来表示浮点数
  - 最小的正数：$a=1.00\ldots00_2 * 2^{-126} = 2^{-126}$
  - 次小的正数：$b=1.00\ldots01_2 * 2^{-126} = 2^{-126} + 2^{-149}$
  - $a - 0 = 2^{-126}$
  - $b - a = 2^{-149}$



- 解决办法：
  - 使用非规格化数：没有隐含的前导1
  - 最小的正数：$a = 2^{-149}$
  - 次小的正数：$b = 2^{-148}$

舍入

- 浮点数的算术运算=> 舍入
- 类型转换时也需要舍入
  - **Double ←→ single Precision ←→ integer**
- 向上舍入
  - **2.001 => 3; -2.001 => -2**
- 向下舍入
  - **1.999 => 1; -1.999 => -2**
- 截断
  - 丢弃最后的位（向**0**舍入）

浮点数的二—十进制转换

0	0110 1000	101 0101 0100 0011 0100 0010

- 符号位: 0 => 正数
- 阶:
  - $0110\ 1000_2 = 104_{10}$
  - 移码校正: 104 - 127 = -23
- 有效数:
  - $1 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} + 0\times 2^{-4} + 1\times 2^{-5} + \ldots$

    $=1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$

    $= 1.0 + 0.666115$

- 十进制值: $1.666115*2^{-23} \sim 1.986*10^{-7}$

浮点数的十——二进制转换

- 简单情况：如果除数是2 的整倍数，则比较简单
- 例如：-0.75的浮点数
- -0.75=3/4
- $-11/100_2 = -0.11_2$
- 规格化为：$-1.1_2 * 2^{-1}$
- $(-1)^S * (1+\text{Significand})*2^{(\text{Exponent} -127)}$
- $(-1)^1 * (1+0.10000000\ldots000)*2^{(126 -127)}$

1	0111 1110	100 0000 0000 0000 0000 0000

浮点数的十——二进制转换

- 除数不是**2**的整数倍
  - 该数无法精确表示
  - 可能需要多位有效位来保证精度
- 难点：如何得到有效位？
  - 循环小数有一个循环体
- 转换
  - 求出足够多的有效位.
  - 根据精度要求（单、双）截断多余的位。
  - 按标准要求给出符号位、阶和有效位。

转换举例

$- 3.333333\ldots$

$$\begin{array}{r} 0.33333333 \\ \times\ 2 \\ \hline 0.66666666 \end{array} \qquad \begin{array}{r} 0.66666666 \\ \times\ 2 \\ \hline 1.33333332 \end{array} \qquad \begin{array}{r} 0.33333332 \\ \times\ 2 \\ \hline 0.66666664 \end{array}$$

$-11.0101010\ldots \Rightarrow -1.1010101.. \times 2^1$

- 有效位: **101 0101 0101 0101 0101 0101**
- •符号位: 负=> **1**
- •阶: **$1+127=128_{10}=1000\ 0000_2$**

1	1000 0000	101 0101 0101 0101 0101 0101

浮点数算数运算

- 浮点数加减法运算
- $X=M_x \times 2^{Ex}$ ，$Y=M_Y \times 2^{Ey}$
- （1）对阶操作，求阶差：$\Delta E=E_X - E_Y$，
- 使阶码小的数的尾数右移$|\Delta E|$位
- 其阶码取大的阶码值；
- （2）尾数加减
- （3）规格化处理
- （4）舍入操作，可能带来又一次规格化
- （5）判结果的正确性，即检查阶码上下溢出

浮点数加运算举例

- $X=2^{+010}\times 0.1101111$，$Y=2^{+100}\times(-0.1010110)$
- 写出X、Y的正确的浮点数表示：
- 阶码用4 位移码　　尾数用8 位原码
- （含符号位）　　　（含符号位）
- $[X]_{浮}$= 0 1 010 1101111
- $[Y]_{浮}$= 1 1 100 1010110
- 为运算方便，尾数的符号位写在数值位之前：
- $[X]_{浮}$= 1 010 0 1101111
- $[Y]_{浮}$= 1 100 1 1010110

浮点数加运算举例

- $X=2^{+010}\times 0.1101100$，$Y=2^{+100}\times(-0.1010110)$
- （1）计算阶差（移码计算）：
- $\Delta E=E_X - E_Y = E_X + (-E_Y) = 1\ 010 + 0\ 100 = 0\ 110$
- 注意：阶码计算结果的符号位在此变了一次反，为-2 的移码，是X的阶码值小，使其取Y 的阶码值1100 (即+4);因此，相应地修改$[M_X]_{原}$=0 0011011**00**（即右移2 位）（右移出的00被保存到保护位中）
- （2）尾数求和：此处是原码加法，符号不相同，绝对值大的减小的，结果符号取决

$$
\begin{array}{rl}
 & 1\ 1010110 \\
- & 0\ 0011011\ \ 00 \\
\hline
 & 1\ 0111011\ \ 00
\end{array}
$$

浮点数加运算举例

- （3）规格化处理：
- 相加结果,数值的最高位为0,应执行1 次左移操作,
- 故得$[M_{X+Y}]_{原}$= 1 1110110, 阶码减1得1 011 (为+3)
- （4）舍入处理: 舍入位是0，按0舍1入规则，得到最终结果：1 1110110
- （5）检查溢出否：和的阶码为1011，不溢出
- 计算后的$[X+Y]_{浮}$= 1 1011 1110110
- 即数的实际值为：$2^3 \times (-0.1110110)$

浮点数乘除法

- 算法：
- 阶码加减：乘法：Ex+Ey，除法Ex-Ey
- 对尾数进行乘除法，求得结果
- 规格化
- 舍入，可能再次进行规格化
- 进行溢出检查（阶码）

浮点数运算

- 浮点数的加减法
  - 移码的减法运算
  - 无符号数运算
- 浮点数的乘除法
  - 移码的加减运算（注意溢出）
- 浮点数尾数运算
  - 原码运算

浮点运算的特点

- 浮点数的加法不满足结合律
  - $x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, and $z = 1.0$
  - $x + (y + z) = -1.5x10^{38} + (1.5x10^{38} + 1.0) = -1.5x10^{38} + (1.5x10^{38}) = 0.0$
  - $(x + y) + z = (-1.5x10^{38} + 1.5x10^{38}) + 1.0 = (0.0) + 1.0 = 1.0$
- 浮点数加法不可结合
- 浮点数的相等比较：小心！只是近似
- for(i=0;i!=10;i+=0.1)