Computer Organization and Design

The Instruction Set Architecture

中山大学

计算机学院

郭雪梅

Email:guoxuem@mail.sysu.edu.cn

Representing Instructions in the Computer

Computer Organization and Design

The Instruction Set Architecture

中山大学

计算机学院

郭雪梅

Email:guoxuem@mail.sysu.edu.cn

Representing Instructions in the Computer

Binary Code and Data (Hello World!)

- Programs consist of Code and Data
- Code and Data are Encoded in Bits

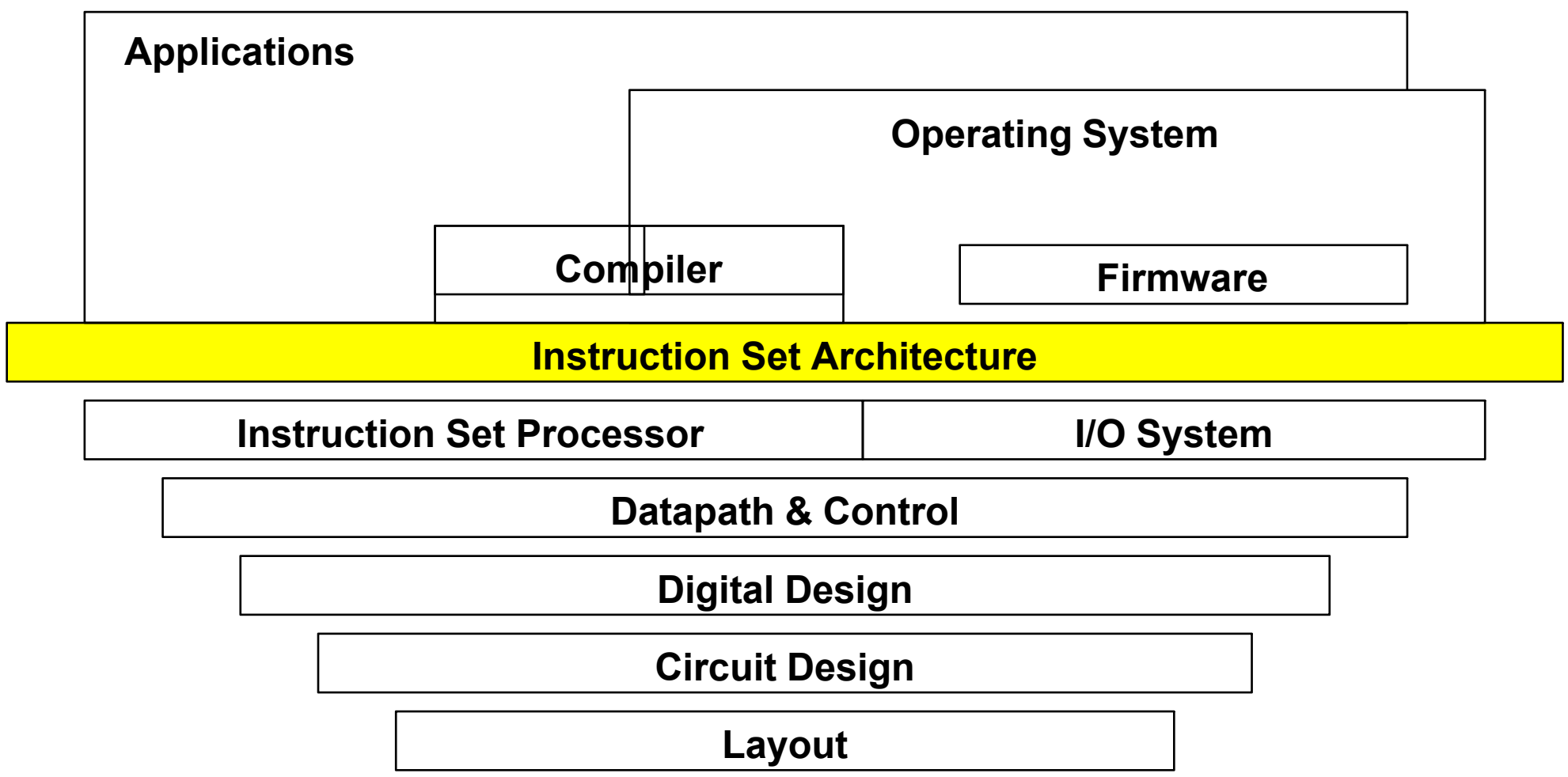
IA-64 Binary (objdump)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............

...

00000260: 5002 0000 0000 0000 006c 6962 632e 736f  P........libc.so
00000270: 2e36 2e31 0070 7269 6e74 6600 5f5f 6c69  .6.1.printf.__li
00000280: 6263 5f73 7461 7274 5f6d 6169 6e00 474c  bc_start_main.GL
00000290: 4942 435f 322e 3200 0000 0200 0200 0000  IBC_2.2.........

...

00000860: 4865 6c6c 6f20 576f 726c 6421 0d00 0000  Hello World!....

...

4000000000000690 <main>:
4000000000000690:       00 10 15 08 80 05       [MII]       alloc r34=ar.pfs,5,4,0
4000000000000696:       30 02 30 00 42 20                   mov r35=r12
400000000000069c:       04 00 c4 00                         mov r33=b0
40000000000006a0:       0a 20 81 03 00 24       [MMI]       addl r36=96,r1;;
40000000000006a6:       40 02 90 30 20 00                   ld8 r36=[r36]
40000000000006ac:       04 08 00 84                         mov r32=r1
40000000000006b0:       1d 00 00 00 01 00       [MFB]       nop.m 0x0
40000000000006b6:       00 00 00 02 00 00                   nop.f 0x0
40000000000006bc:       b8 fd ff 58                         br.call.sptk.many b0=4000000000000460;;
40000000000006c0:       00 08 00 40 00 21       [MII]       mov r1=r32
40000000000006c6:       80 00 00 00 42 00                   mov r8=r0
40000000000006cc:       20 02 aa 00                         mov.i ar.pfs=r34
40000000000006d0:       00 00 00 00 01 00       [MII]       nop.m 0x0
40000000000006d6:       00 08 05 80 03 80                   mov b0=r33
40000000000006dc:       01 18 01 84                         mov r12=r35
40000000000006e0:       1d 00 00 00 01 00       [MFB]       nop.m 0x0
40000000000006e6:       00 00 00 02 00 80                   nop.f 0x0
40000000000006ec:       08 00 84 00                         br.ret.sptk.many b0;;
```

Interfaces in Computer Systems

Software: Produce Bits Instructing Machine to Manipulate State or Produce I/O

软件：产生位指令让机器操作状态或生成输入输出



Hardware: Read and Obey Instruction Bits

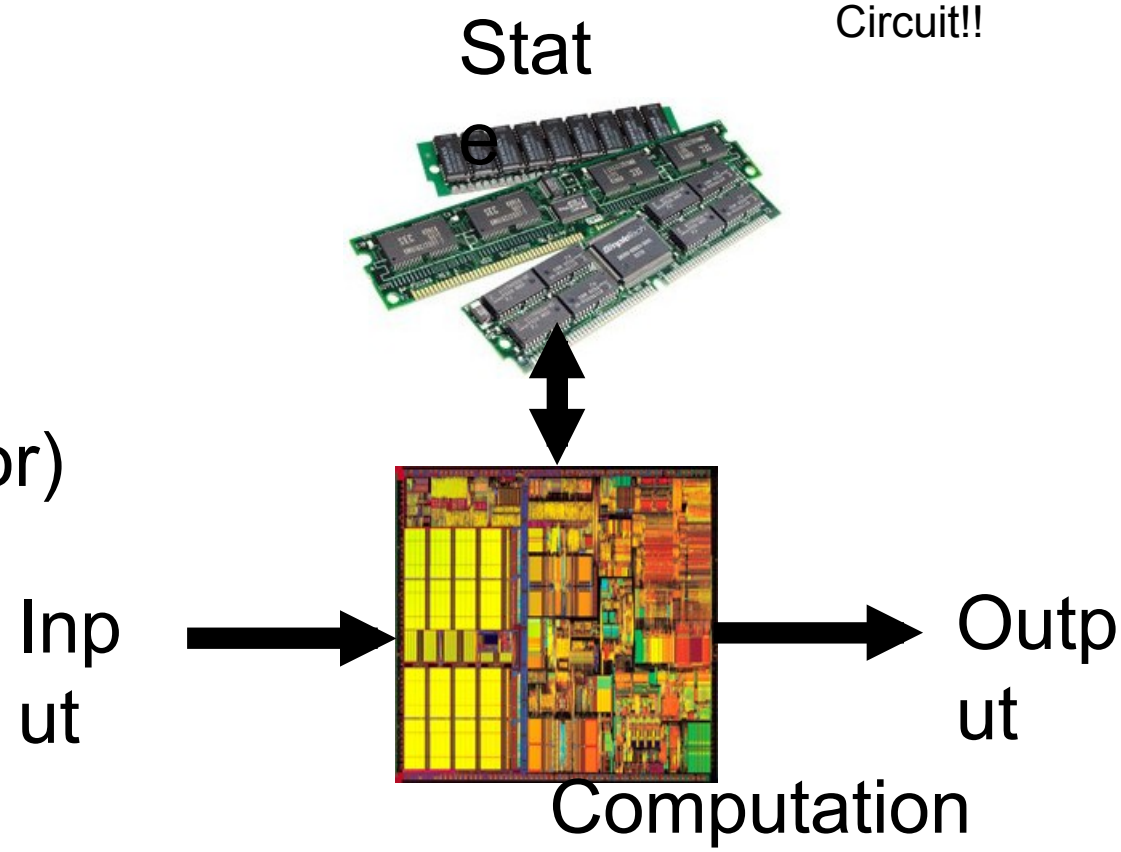
硬件：读指令位，按照指令执行

Instructions 指令

Computers process information

- Input/Output (I/O)
- State (memory)
- Computation (processor)

计算机处理信息，包括输入输出、状态存储、计算



Sequential Circuit!!

- Instructions instruct processor to manipulate state
- Instructions instruct processor to produce I/O in the same way

指令让处理器对状态进行操作、输入输出。

State – 状态

现代机器典型的架构状态存于:

- 内存– Big, Slow
- 寄存器 – Small, Fast (always on processor chip)
- 程序计数器– Address of executing instruction

架构 - 汇编程序员界面的一部分

PC
10

Registers							
0	1	2	3	4	5	6	7
0000	0788	B700	0010	0401	0002	0003	00A0
8	9	A	B	C	D	E	F
0000	0788	B700	0010	0401	0002	0003	00A0

Main Memory	
00:	0000 0000 0000 0000 0000 0000 0000 0000
08:	0000 0000 0000 0000 0000 0000 0000 0000
10:	9222 9120 1121 A120 1121 A121 7211 0000
18:	0000 0001 0002 0003 0004 0005 0006 0007
20:	0008 0009 000A 000B 000C 000D 000E 000F
28:	0000 0000 0000 FE10 FACE CAFE ACED CEDE
.	
.	
E8:	1234 5678 9ABC DEF0 0000 0000 F00D 0000
F0:	0000 0000 EEEE 1111 EEEE 1111 0000 0000
F8:	B1B2 F1F5 0000 0000 0000 0000 0000 0000

Instructions: Language of the Computer
讲计算机语言

- 命令计算机硬件工作，你必须讲它的

  语言 language.

* 机器语言的单词叫做 **指令 “Instructions”.**
* 词汇被称为 **指令集 “Instruction set”.**
* 机器语言更简单更容易，一旦你学会了一种，你就可以轻松地学会其他语言。

  所有的计算机硬件和操作原理都是类似的，基本操作也是类似的（加、乘、传送等），一旦你学会了一种，你就可以轻松地学会其他语言。

计算机中数的表示基本问题

- 基本约束：采用二进制，只有**1**和**0**；
- 数的表示要解决的问题
  - 数的符号：正数、负数、零
  - 数的形态：整数、小数、小数点的性质；

伟大的现实：无论你计划在计算机上存储什么，最终都必须表示为有限的比特集合。无论是整数、实数、字符、字符串、数据结构、指令、程序、图片、视频等，都是如此。这实际上意味着只有离散的量才能被精确地表示。非离散（连续）量必须近似。

为什么不采用10进制表示?

电子实现复杂

以电子方式对10个不同的值很难存储。

ENIAC（第一台电子计算机）使用了10个真空管/数字

它们很难传输。需要高精度在一根线上编码10个信号电平。

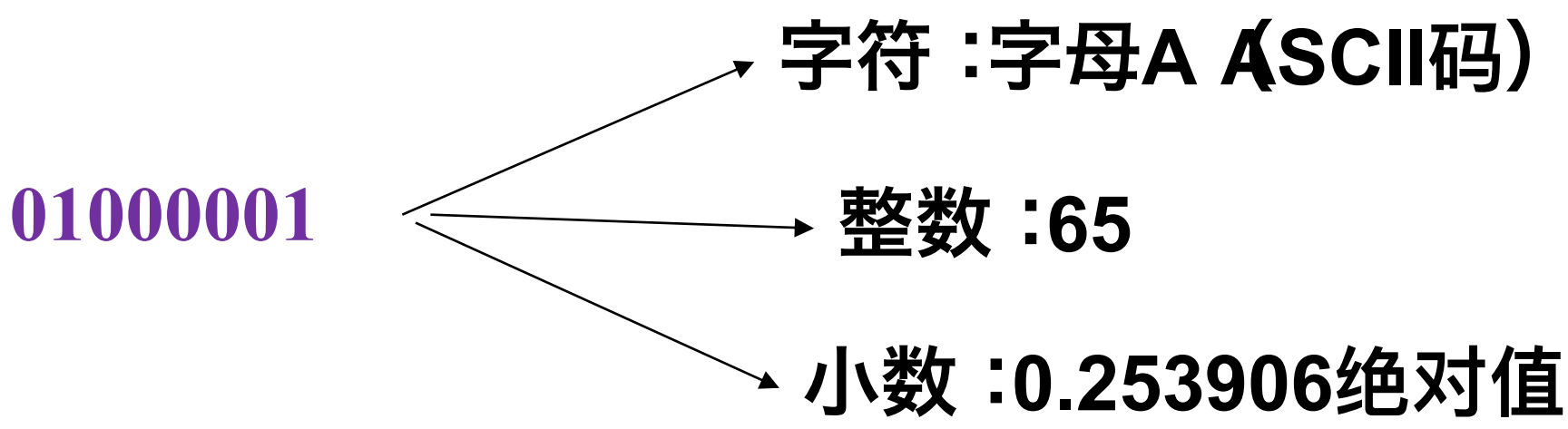
实现数字逻辑功能混乱：加法、乘法等。

Encoding（信息的表示）

- Encoding = assign representation to information

  编码 = 为信息分配表示

- Examples:
  - 假设你要用符号表示两种事情，对其编码
    - 比如手势☞ 和手势☜
    - 怎样表示? 用一位二进制0和1
  - 现在要对4个符号进行编码
    - 😄 (微笑), 😱 (尖叫), 😖 (混乱), 😪 (困倦)
    - 怎样表示? 两位二进制00、01、10、11



Fixed-Size Codes

Length of a fixed-length code （位数相同的编码）

如果选择的可能性相同，那么信息表示通常使用固定位数的代码。 要用足够位数表示信息内容。

ex. 10个十进制数字= **{0,1,2,3,4,5,6,7,8,9}**

4-bit BCD (二进制编码十进制)

$\log_2(10/1) = 3.322 < 4$ bits

BC D
0 - 0000
1 - 0001
2 - 0010
3 - 0011
4 - 0100
5 - 0101
6 - 0110
7 - 0111
8 - 1000
9 - 1001

ex. ~84个英文字符= {A-Z (26), a-z (26), 0-9 (10), punctuation (8), math (9),financial (5)}

7-bit ASCII (American Standard Code for Information Interchange)

$\log_2(84/1) = 6.392 < 7$ bits

ASCII

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
000	NUL	SOH	STX	ETX	EOT	ACK	ENQ	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- 字符大小写第六位编码不同，10XXXXX 大写, 11XXXXX 是小写
- 最高位 00 为特殊的控制字符

ex. cntl-g → bell（响铃），cntl-m → carriage return（回车）, cntl-[→ esc

- 7位**ASCII** + 1位可选奇偶校验为8位，这也是为什么字节定义为8位的原因之一.

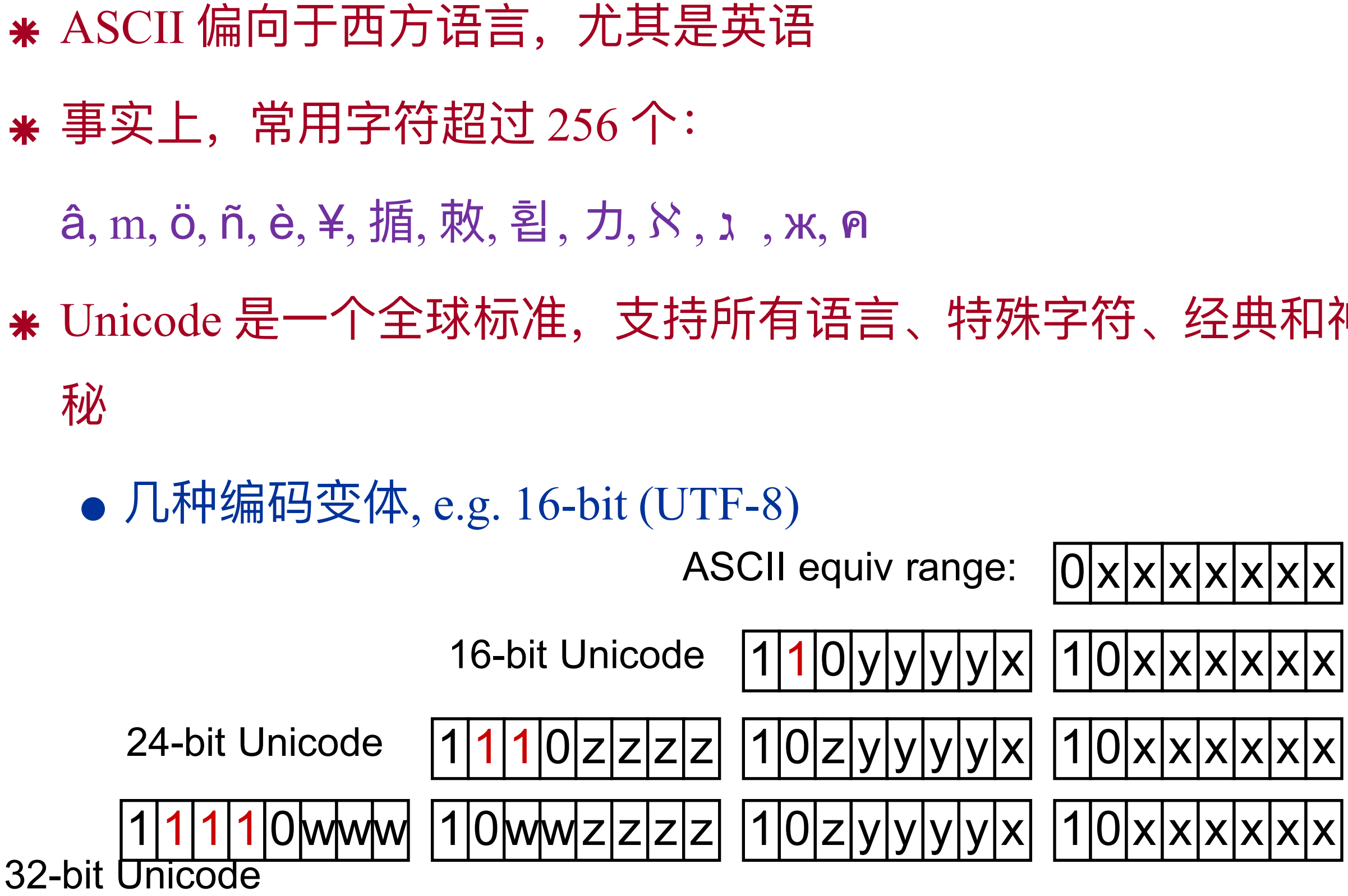
Unicode（统一码）

* ASCII 偏向于西方语言，尤其是英语
* 事实上，常用字符超过 256 个：

  â, m, ö, ñ, è, ¥, 揗, 敕, 횕, 力, ᛯ, ג, ж, ค

* Unicode 是一个全球标准，支持所有语言、特殊字符、经典和神秘

  - 几种编码变体, e.g. 16-bit (UTF-8)



Encoding Positive Integers

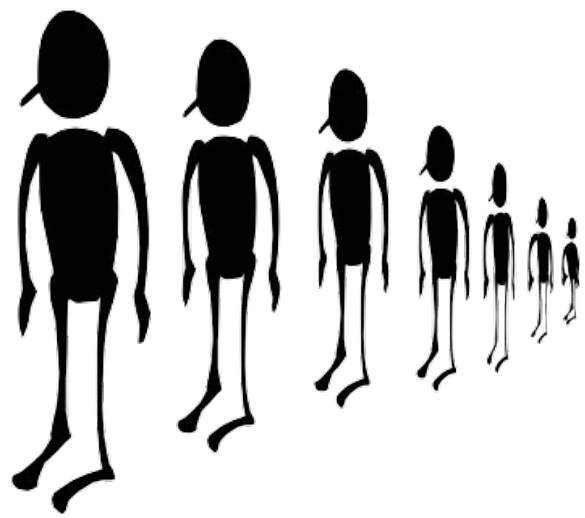
正整数编码

将正整数编码为位序列很简单，每个位都分配了一个权重。

从右到左排序，这些权重是 2 的递增幂。以这种方式编码的 n 位数的值由以下公式给出：

2^{15} 2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^{9} 2^{8} 2^{7} 2^{6} 2^{5} 2^{4} 2^{3} 2^{2} 2^{1} 2^{0}

0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	0



$$v = \sum_{i=0}^{n-1} 2^i b_i$$

2^1 = 2

\+ 2^5 =32

\+ 2^6 =64

\+ 2^7 =128

\+ 2^8 =256

\+ 2^9 =512

\+ 2^{10} =1024

2018

Some Bit Tricks（一些技巧）

* 习惯于以二进制工作
* 这里有一些有用的指南

1. Memorize the first 10 powers of 2

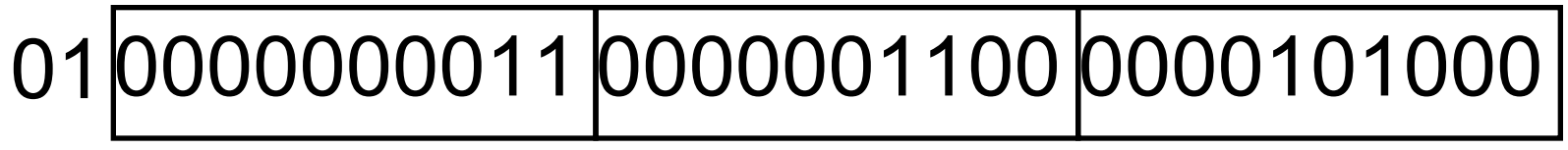
$2^0 = 1$	$2^5 = 32$
$2^1 = 2$	$2^6 = 64$
$2^2 = 4$	$2^7 = 128$
$2^3 = 8$	$2^8 = 256$
$2^4 = 16$	$2^9 = 512$

2. Memorize the prefixes for powers of 2 that are multiples of 10（记住2的10的倍数幂）

2^{10} = Kilo (1024)
2^{20} = Mega (1024*1024)
2^{30} = Giga (1024*1024*1024)
2^{40} = Tera (1024*1024*1024*1024)
2^{50} = Peta (1024*1024*1024 *1024*1024)
2^{60} = Exa (1024*1024*1024*1024*1024*1024)

Even More Tricks with Bits

* 习惯于以二进制工作
* 这里有一些有用的指南



3. 把二进制数转换为10进制, 先分成10位一组.
4. 先计算最左边剩余位的值(1) 确定前缀(GIGA)

   这里，2^{30}，1G
5. 计算每个剩余 10 位簇的值并相加

1*1024^3+3*1024*1024+12*1024+40

Binary Number Conversion
(二进制数转换)

Binary → Decimal

$1001010_{two} = ?_{ten}$

Binary Digit	Decimal Value
0	$0 \times 2^0 = 0$
1	$1 \times 2^1 = 2$
0	$0 \times 2^2 = 0$
1	$1 \times 2^3 = 8$
0	$0 \times 2^4 = 0$
0	$0 \times 2^5 = 0$
1	$1 \times 2^6 = 64$
	$\sum = 74_{ten}$

Decimal → Binary

$74_{ten} = ?two$

Decimal	Binary (odd?)
74	0
/2 = 37	1
/2 = 18	0
/2 = 9	1
/2 = 4	0
/2 = 2	0
/2 = 1	1
Collect →	1001010_{two}

如果十进制为奇数，则二进制数为 1，否则为 0

Hexadecimal（十六进制）

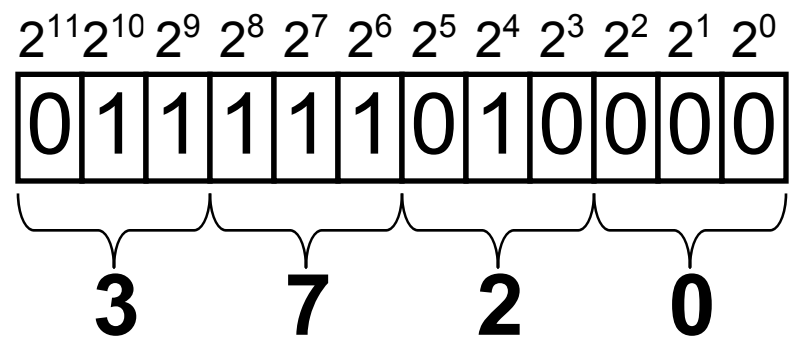
- Problem: 二进制位数太多
  - e.g. $7643_{ten} = 1110111011011_{two}$
- Solutions:
  - 分组: 1 1101 1101 1011$_{two}$
  - 十六进制: 1DDB$_{hex}$
  - 八进制Octal: 1 110 111 011 011$_{two}$ 16733$_{oct}$

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Other Helpful Clusterings(其他有用的聚类)

- 有时方便使用其他数字“基数”
  - 常常使用2的幂次作为基数: e.g., 8, 16
    - 将位数分成组
  - 基数 8 叫做8进制 octal → 3位二进制位为一组
    - 常规: 8进制最前面以**0 开头**

$$v = \sum_{i=0}^{n-1} 8^i d_i$$



= 2000_{10}

03720

Octal - base 8

000 - 0
001 - 1
010 - 2
011 - 3
100 - 4
101 - 5
110 - 6
111 - 7

$0*8^0 = 0$
$+ 2*8^1 = 16$
$+ 7*8^2 = 448$
$+ 3*8^3 = 1536$
2000_{10}

最后一个聚类

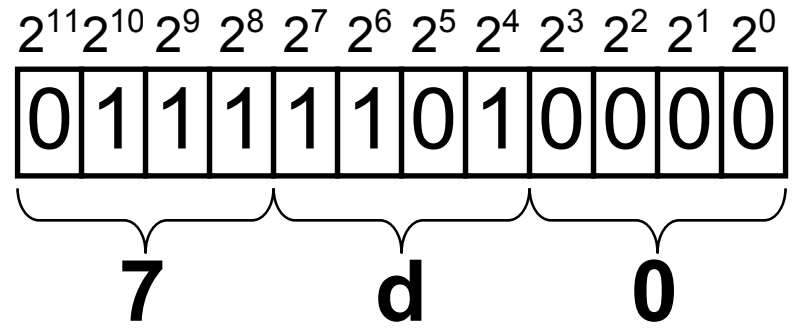
* 基数 16 最常用!
  - 叫做16进制 hexadecimal or hex ➔ 二进制4位一组
  - hex ‘digits’ (“hexits”): 0-9, and A-F
  - 16进制的每一位表示16的幂次
    - 16进制习惯表示: 以 **0x 开头**

$$v = \sum_{i=0}^{n-1} 16^i d_i$$

0x7d0

Hexadecimal - base 16

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f



$$\begin{aligned} 0*16^0 &= 0 \\ +\ 13*16^1 &= 208 \\ +\ 7*16^2 &= 1792 \\ \hline & 2000_{10} \end{aligned}$$

Signed-Number Representations
有符号数的表示

* 有符号数怎样表示?机器数表示
  - 使用额外一位编码表示符号

(数的正负问题：设符号位,“0”表示“正”，“1”表示“负”，固定为编码的最高位。)

➤ 约定：最高有效位（最左边）用于符号

➤ 叫做有符号数表示

$$v = (-1)^S \sum_{i=0}^{n-2} 2^i b_i$$

S	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	0	1	0	0	0	0

2000　　-2000

Signed Integer Representation
有符号整数的表示

符号 & 数值 (8-bit example):

加法规则，a + b:

- If (*a>0 and b>0):* add, sign 0
- If (*a>0 and b<0):* subtract, sign …
- … 有符号数加法怎样确定正负，有4种情况
- +0, -0 → 它们一样吗?比较器必须处理特殊情况!
- **真值0怎么办：正零，负零**

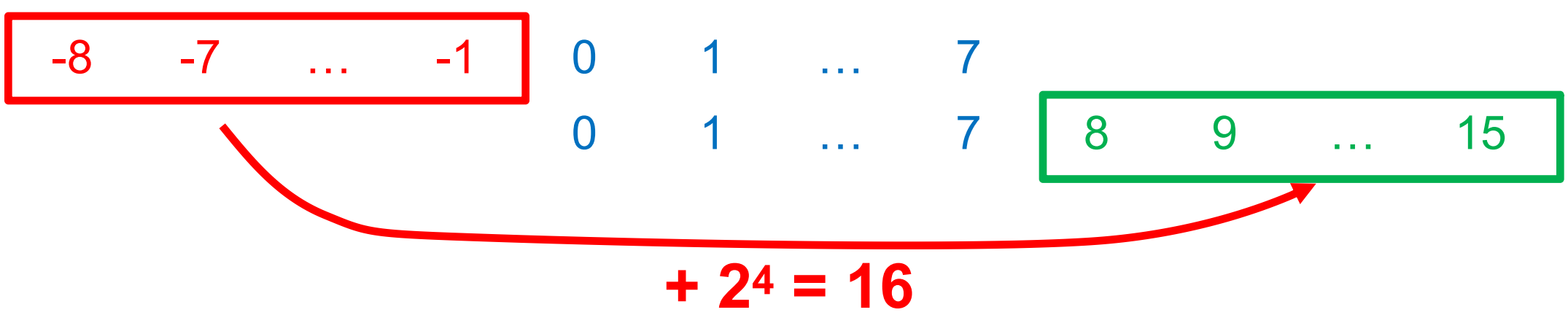
这种表示方法麻烦

- ”复杂” 化硬件: 降低速度/ 增加能耗
- ***有没有更好的表示方法?***

4-bit Example（用4位数的计算举例说明为什么使用补码表示） M=16

← Decimal →			← Binary →	
7	7	7	0111	
+ -3	+ -3 + 16	+ 13	+ 1101	
4	4 + 16	4 + 16	1 0100	0100 + 1 0000

- 映射负数→ 正数
  - Example for N=4-bit: -3 → $2^4 - 3 = 13$
  - “二进制补码表示”
  - 正负数相加没有特殊规则，一样处理



Two's Complement

(8-bit example)

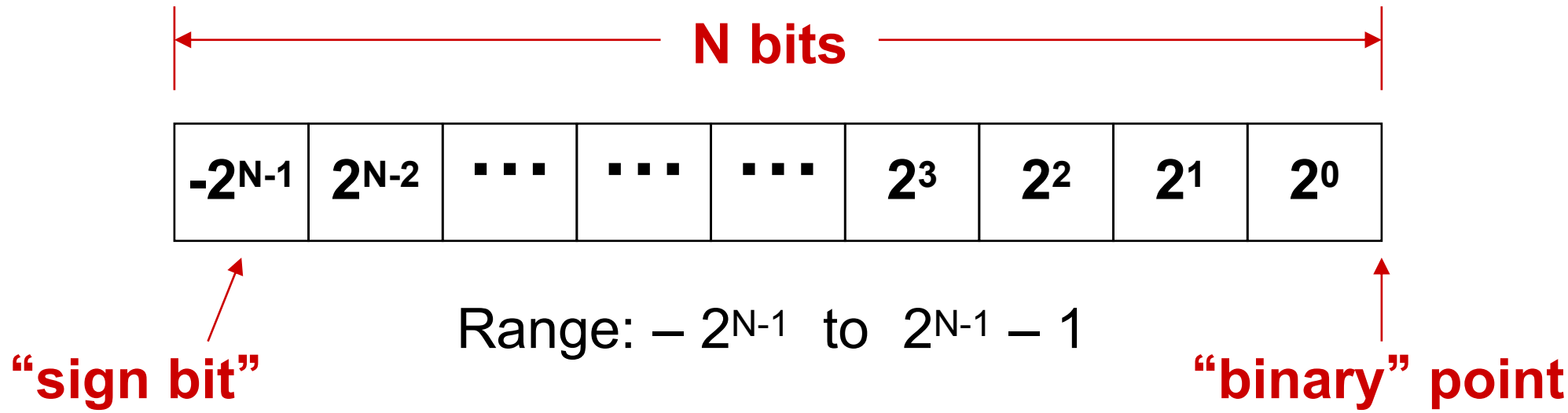
Signed Decimal	Unsigned Decimal	Binary Two's Complement
-128	128	1000 0000
-127	129	1000 0001
…	…	…
-2	254	1111 1110
-1	255	1111 1111
0	0	0000 0000
1	1	0000 0001
…	…	…
127	127	0111 1111

+256

+0

Note: Most significant bit (MSB) equals sign

二进制补码的其它表示



简单地修正了无符号数的表示，只是把符号位当做负数。

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

符号数

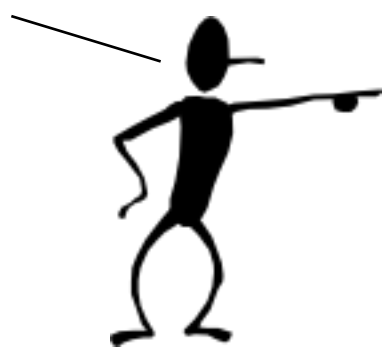
8-bit 2’s complement example:

$$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1$$
$$= -128 + 64 + 16 + 4 + 2 = -42$$

Why 2's Complement?

- Benefit: 模 2^n 的运算，用补码表示减法变成了加负数
  - 只有加法没有减法运算，不用管符号位!
  - 处理无符号数同样适用!

符号数表示法加一个负数意味着减一个正数。但是，补码表示（2's complement），加法就是加法，不用考虑符号。事实上，当你使用 2 的补码表示时，你永远不需要减法。



Example:

$$
\begin{array}{rrcl}
 & 55_{10} & = & 00110111_2 \\
+ & 10_{10} & = & 00001010_2 \\
\hline
 & 65_{10} & = & 01000001_2
\end{array}
$$

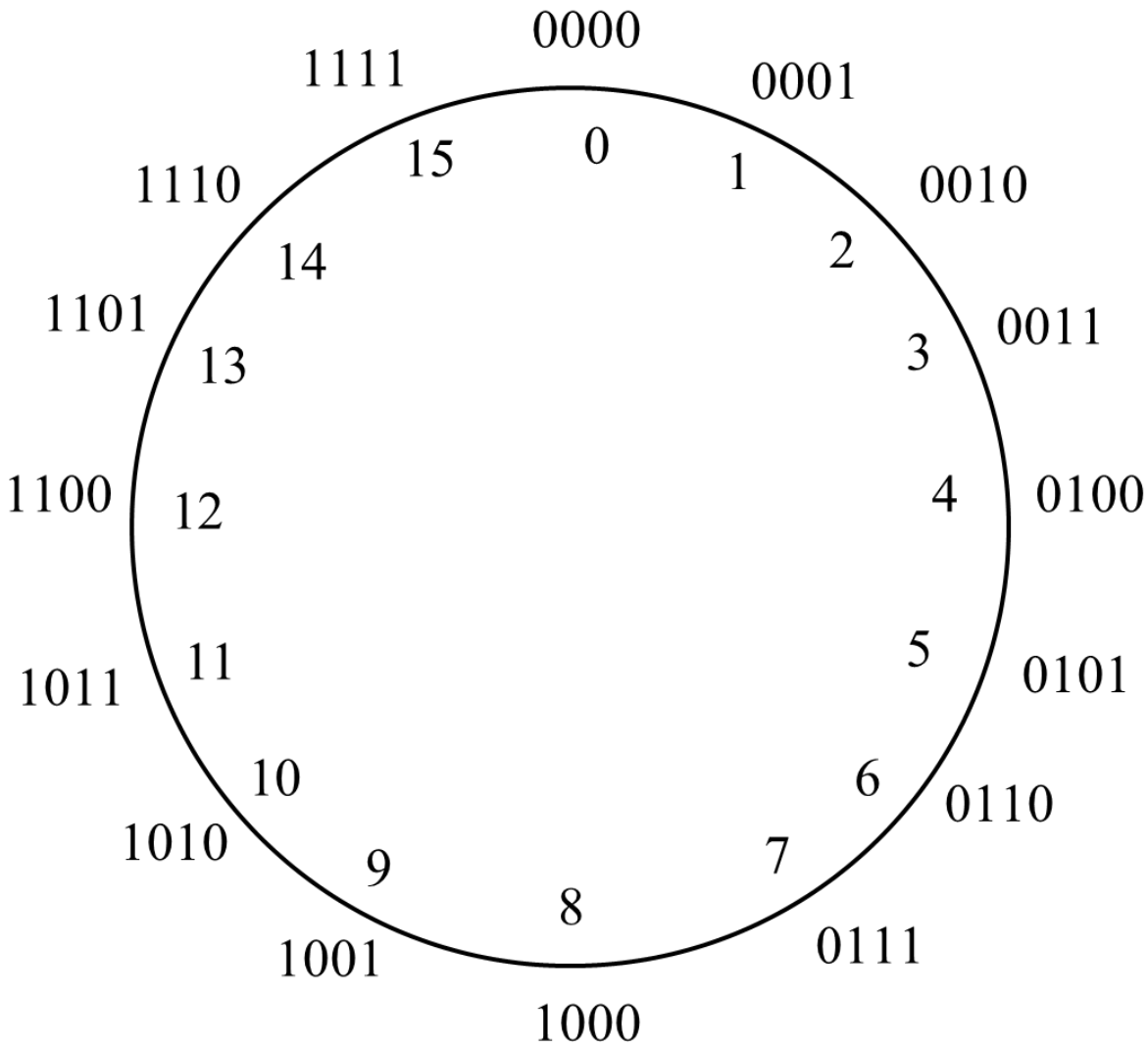
$$
\begin{array}{rrcl}
 & 55_{10} & = & 00110111_2 \\
+ & -10_{10} & = & 11110110_2 \\
\hline
 & 45_{10} & = & 100101101_2
\end{array}
$$

2's Complement

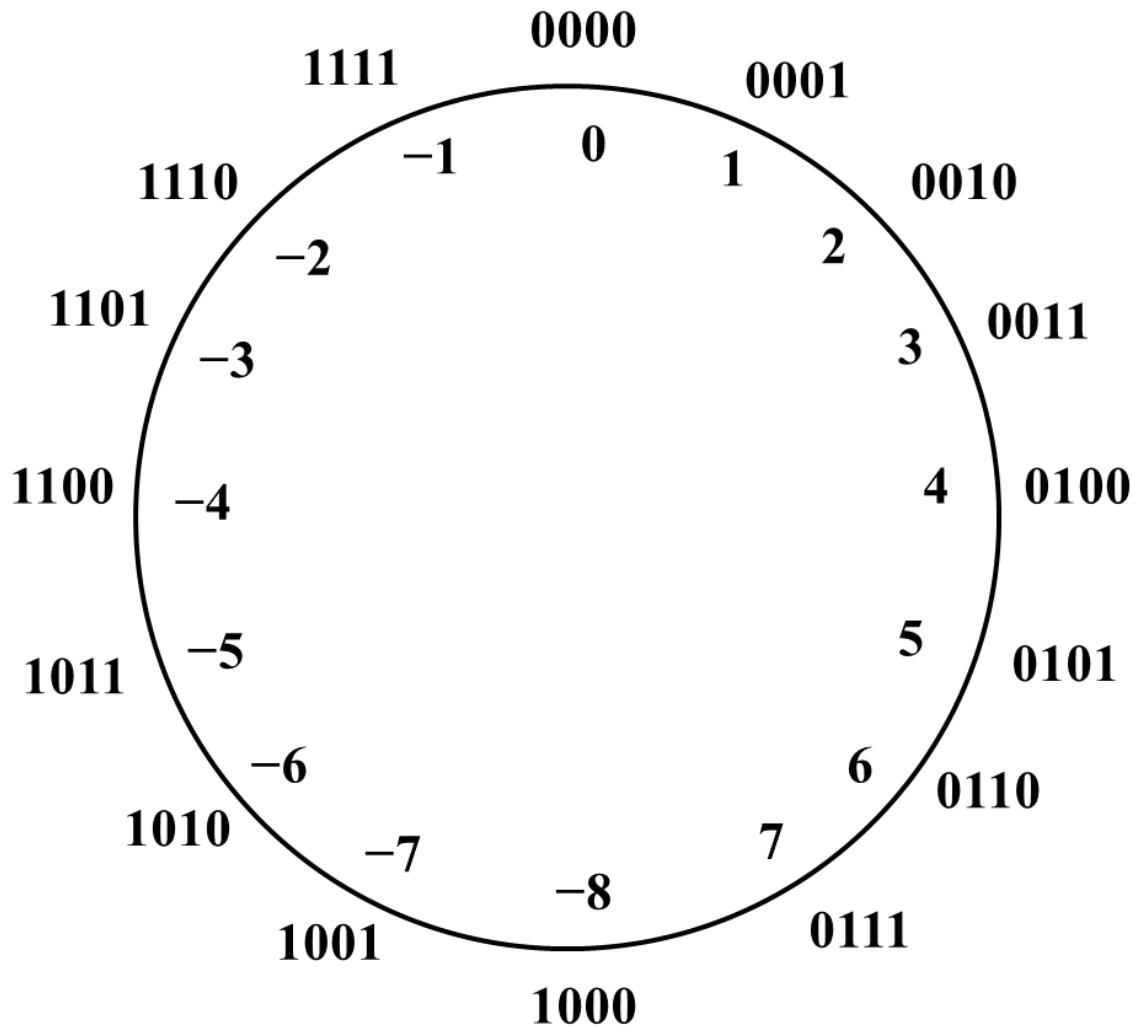
* 怎样求补?
  - 先按位求反(i.e. 1 → 0, 0 → 1), 然后加一按位求反加1
  - Example: 20 = 00010100, -20 = 11101011 + 1 = 11101100
* 符号扩展
  - 假设有一个需要"扩展"为 16 位的 8 位数字
    - 为什么? 因为有可能与一个16位数相加…

- Examples
  - 16-bit version of 42 = 0000 0000 0010 1010
  - 8-bit version of -2 = 1111 1111 1111 1110

Unsigned Integers: 4-bit System

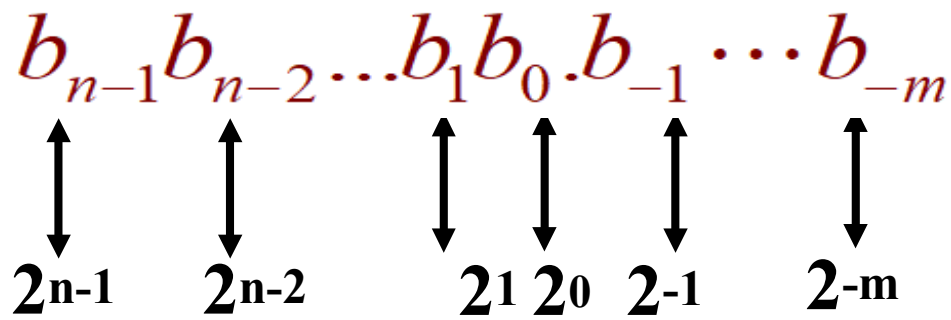


Signed Integers: 4-bit System



＊ 二进制转换10进制

对一n位整数和m位小数的二进制数为

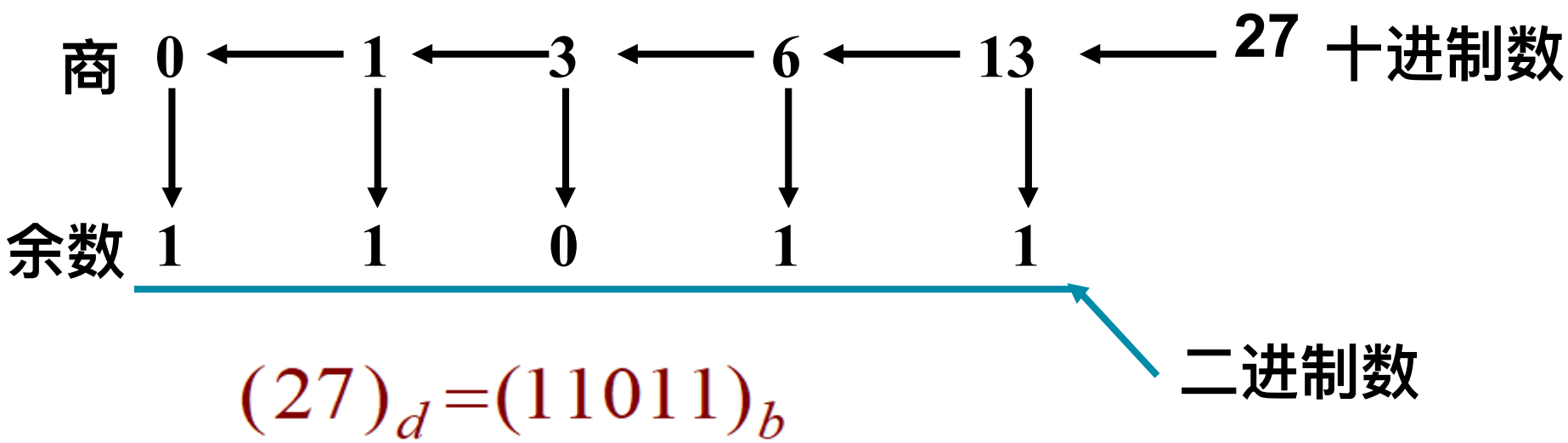


则其按权展开式为

$$v=\sum_{i=0}^{n-1}2^i b_i$$

＊ 10进制转换二进制

除基取余法：用目标数制的基数（R=2）去除十进制数，第一次相除所得余数为目的数的最低位 K_0，将所得商再除以基数，反复执行上述过程，直到商为“0”，所得余数为目的数的最高位K_{n-1}。

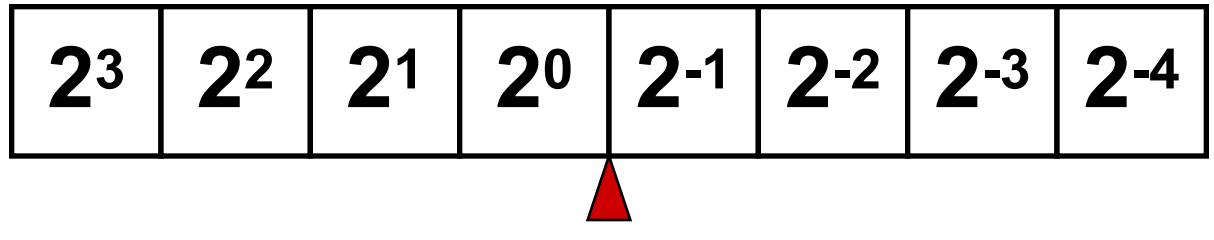


Non-Integral Numbers

* 非整数怎么表示?
  - examples
    - 1.234
    - -567.34
    - 0.00001
    - 0.000000000000012
  - fixed-point representation（定点表示）
  - floating-point representation（浮点表示）

Fixed-Point Representation（定点表示）

* 为“二进制”小数点设置一个确定的位置
  - 它左边都是数字的整数部分
  - 它右边都是数字的小数部分



* 定点左边表示整数部分，右边表示小数部分

$$1101.0110 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3}$$
$$= 8 + 4 + 1 + 0.25 + 0.125$$
$$= 13.375$$

Or

$$1101.0110 = 214 * 2^{-4} = 214/16 = 13.375$$

Fixed-Point Base Conversion

定点基转换

* **二进制小数转换10进制**
  - 将每位乘以其位置的2的幂次方
  - 只是 2 的幂现在是负数
  - 对于 m 位小数位

$$v = \frac{b_{-1}}{2} + \frac{b_{-2}}{4} + \frac{b_{-3}}{8} + \frac{b_{-4}}{16} + \ldots + \frac{b_{-m}}{2^m}$$

$$v = \sum_{i=-1}^{-m} 2^i b_i$$

$$v = 2^{-1}b_{-1} + 2^{-2}b_{-2} + 2^{-3}b_{-3} + 2^{-4}b_{-4} + \ldots + 2^{-m}b_{-m}$$

* Examples
  - $0.1_2 = ½ = 0.5_{10}$
  - $0.0011_2 = 1/8 + 1/16 = 0.1875_{10}$
  - $0.001100110011_2 = 1/8+1/16+1/128+1/256+1/2048+1/4096 = 0.19995117187_{10}$ (getting close to 0.2)
  - 0.0011_2 (repeats) $= 0.2_{10}$

Addition
4-bit Example

Unsigned

$$\begin{array}{r} 3_{ten} \\ +\ 4_{ten} \\ \hline 7_{ten} \end{array} \qquad \begin{array}{r} 0011_{two} \\ +\ 0100_{two} \\ \hline 0111_{two} \end{array}$$

Signed (Two's Complement)

$$\begin{array}{r} 3_{ten} \\ +\ 4_{ten} \\ \hline 7_{ten} \end{array} \qquad \begin{array}{r} 0011_{two} \\ +\ 0100_{two} \\ \hline 0111_{two} \end{array}$$

$$\begin{array}{r} 3_{ten} \\ +\ 11_{ten} \\ \hline 14_{ten} \end{array} \qquad \begin{array}{r} 0011_{two} \\ +\ 1011_{two} \\ \hline 1110_{two} \end{array}$$

$$\begin{array}{r} 3_{ten} \\ +\ -5_{ten} \\ \hline -2_{ten} \end{array} \qquad \begin{array}{r} 0011_{two} \\ +\ 1011_{two} \\ \hline 1110_{two} \end{array}$$

No special rules for two's complement signed addition

Overflow
4-bit Example

Unsigned

$$\begin{array}{rr} 13_{ten} & 1101_{two} \\ +\ 14_{ten} & +\ 1110_{two} \\ \hline 27_{ten} & 1\ 1011_{two} \end{array}$$

carry-out and overflow

$$\begin{array}{rr} 7_{ten} & 0111_{two} \\ +\ 1_{ten} & +\ 0001_{two} \\ \hline 8_{ten} & 0\ 1000_{two} \end{array}$$

no carry-out and no overflow

Carry-out → Overflow

Signed (Two's Complement)

$$\begin{array}{rr} -3_{ten} & 1101_{two} \\ +\ -2_{ten} & +\ 1110_{two} \\ \hline -5_{ten} & 1\ 1011_{two} \end{array}$$

carry-out but no overflow

$$\begin{array}{rr} 7_{ten} & 0111_{two} \\ +\ 1_{ten} & +\ 0001_{two} \\ \hline -8_{ten} & 0\ 1000_{two} \end{array}$$

no carry-out but overflow

Carry-out → Overflow

Overflow Detection

4-bit Example

Unsigned

- Carry-out indicates overflow

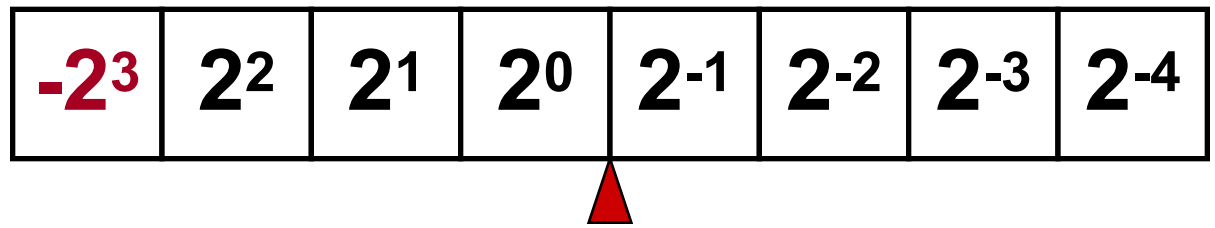
Signed (Two's Complement)

- Overflow if
  - Signs of operands are equal *AND*
  - Sign of result differs from sign of operands
- No overflow when signs of operands differ

Overflow rules depend on operands (signed vs unsigned)

Signed fixed-point numbers
（有符号数的定点表示）

- 怎样加入符号?
  - 有符号定点小数的表示
    - 最左边加一位存储符号
    - 跟符号数整数表示类似
  - 2's complement 二进制补码
    - 最左边的位有一个负系数



$1101.0110 = -2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3}$

$= -8 + 4 + 1 + 0.25 + 0.125 = -2.625$

- OR:
  - 首先忽略二进制点，使用2的补码，再把点放回去

$1101.0110 = (-128 + 64 + 16 + 4 + 2) * 2^{-4}$

$= -42/16 = -2.625$

Bias Notation（偏差表示）

* 思路：加一个大数，使所有数都变成正数!
  - 其原值必须从其表示中减去这个“偏差”
  - 这种表示称为“偏差表示法”

$$v = \sum_{i=0}^{n-1} 2^i b_i - Bias$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	1	1	0

Ex: (Bias = 127)

$$\begin{array}{rr} 6 * 1 = & 6 \\ 13 * 16 = & 208 \\ & -\ 127 \\ \hline & 87 \end{array}$$

Why? Monotonicity 单调性）

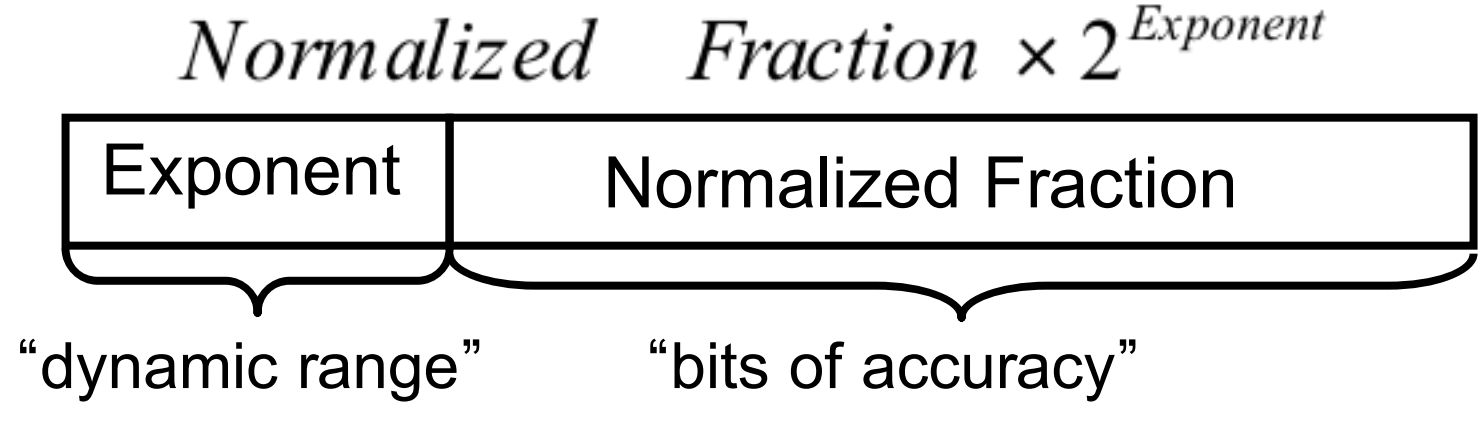
浮点表示

另一种表示数字的方法是使用类似于科学记数法的表示.

- 此格式可用于表示数字比如3.90×10^{-4}, 非常小的数1.60×10^{-19}, 非常大的数6.02×10^{23}.
- 此表示法使用两个字段来表示每个数字. 第一部分表示归一化分数（normalized fraction） (称为尾数（significand）), 第二部分表示指数部分 (i.e. 浮点表示的二进制数的小数点位置).

每个数用两部分表示，第一部分用来表示规格化分数，另一部分表示指数部分

$$Normalized\ \ Fraction \times 2^{Exponent}$$



浮点数表示（IEEE 754）

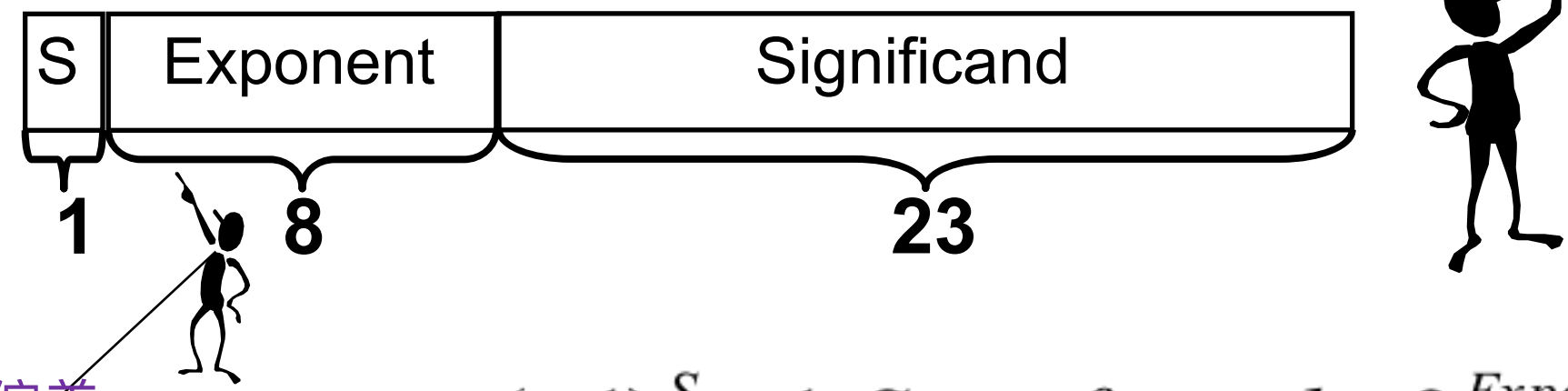
- **IEEE 754**：符号（**Sign**）、阶码（**Exponent**）和尾数（**Mantissa**）。
- **IEEE 754**标准：单精度浮点数**32**位，双精度浮点数**64**位，

- 数符**S**： **1**位，**0**表示正数，**1**表示负数
- 阶码**E**：用移码表示，**n** 位阶码偏移量为**2^(n-1)-1**。如**8**位阶码偏移量为**7FH**（即**127**），**11**位阶码偏移量**3FFH**（即**1023**
- 尾数**M**： 尾数必须规格化成小数点左侧一定为**1**，并且小数点前面这个**1**作为隐含位被省略。这样单精度浮点数尾数实际上为**24**位。**M=1.m**

IEEE 754 Floating-Point Formats

1 是隐藏的，因为它在数字“标准化”后不用提供任何信息

Single-precision format



指数以偏差 127 表示法表示。

$$v = (-1)^S \times 1.Significand \times 2^{Exponent-127}$$

$42.75 = 00101010.11000000_2$

Normalize: $001.01010110000_2 \times 2^5$

(127+5)

$0\ 10000100\ 01010110000000000000000_2$

$0100\ 0010\ 0010\ 1011\ 0000\ 0000\ 0000\ 0000_2$

$42.75 = 0x422B0000_{16}$

IEEE 754 Floating-Point Formats

单精度浮点表示范围

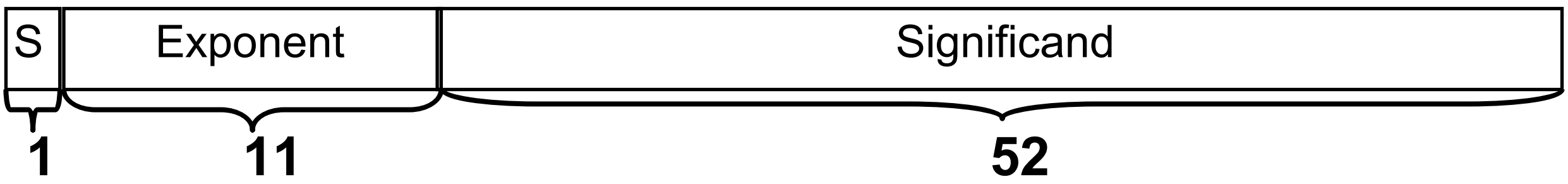
- 最小正数: ~1.18 x 10^{-38}

- 最大正数: ~3.4 x 10^{38}

多次单精度运算后变得不准确

MinNorm	MaxNorm
$(-1)^s \times (2)^{-126} \times 1.0$	$(-1)^s \times (2)^{127} \times (2-2^{-23})$
S 00000001 00000000000000000000000	S 11111110 11111111111111111111111

双精度浮点表示



$$v = (-1)^S \times 1.Significand \times 2^{Exponent-1023}$$

浮点数的表示有一定的范围，超出范围时会产生溢出（Flow），一般称大于绝对值最大的数据为上溢（Overflow），小于绝对值最小的数据为下溢（Underflow）。

1、浮点数的表示约定

单精度浮点数和双精度浮点数都是用IEEE 754标准定义的，其中有一些特殊约定，例如：

1、当指数E=0，尾数M=0时，表示0。

2、当指数E=255，尾数M=0时，表示无穷大，用符号位来确定是正无穷大还是负无穷大。

3、当指数E=255，M≠0时，表示NaN（Not a Number，不是一个数）

Special values in the IEEE 754 standard.

Sign	8-Bit Based Exponent	23-Bit Normalized Fraction
[31]	[30:23]	[22:0]

Ebais=2^(8−1)−1=127，偏移量取127
指数E的取值范围是 (−2^(8−1)+1)−2^(8−1), 即-127 -- 128
做为常规值， E的取值范围是-126 -- 127

最大值和最小值为

MinNorm	MaxNorm
$(-1)^s \times (2)^{-126} \times 1.0$	$(-1)^s \times (2)^{127} \times (2-2^{-23})$
S 00000001 00000000000000000000000	S 11111110 11111111111111111111111

Special Value	Exponent	Significand
+/- 0	0000 0000	0
Denormalized number	0000 0000	Nonzero
NaN	1111 1111	Nonzero
+/- infinity	1111 1111	0

2、非规范浮点数

当 两个绝对值极小的浮点数相减后，其差值的指数可能超出允许范围，最终只能近似为0。为了解决此类问题，IEEE标准中引入了非规范 （Denormalized）浮点数，规定当浮点数的指数为允许的最小指数值时，尾数不必是规范化（Normalized）的。有了非规范浮点数，去掉了隐含的尾数位的制约，可以保存绝对值更小的浮点数。而且，由于不再受到隐含尾数域的制约，上述关于极小差值的问题也不存在了，因为所有可以保存的浮点数之间的差值同样可以保存。

[例.1]： 单精度浮点数0x00280000（real*4）
转换成二进制
00000000001010000000000000000000（非规格化数Denormalized，指数为0，尾数不为0）
符号位 指数部分（8位） 尾数部分
0 00000000 01010000000000000000000
符号位=0；因指数部分=0，则：尾数部分M为m：
0.01010000000000000000000=0.3125
该浮点数的十进制为： (-1)^0*2^(-126)*0.3125
=3.673419846319648462402301678819 5e-39

[例2]：双精度浮点数0xC04E000000000000（real*8） 转换成二进制

1100000001001110000000000000000000000000000000000000000000000000

符号位 指数部分（11位）

1　　10000000100

尾数部分1110000000000000000000000000000000000000000000000000

符号位=1；指数=1028，因指数部分不为全'0'且不为全'1'，则：尾数部分M为1+m：

1.1110000000000000000000000000000000000000000000000000=1.875

该浮点数的十进制为：(-1)^1*2^(1028-1023)*1.875=-60

将十进制数转换成浮点格式（real*4）

[例3]：26.0，十进制26.0转换成二进制11010.0

规格化二进制数1.10100*2^4

计算指数4+127=131

符号位 指数部分 尾数部分

0 10000011 10100000000000000000000

以单精度（real*4）浮点格式存储该数

0100 0001 1101 0000 0000 0000 0000 0000　0x41D0 0000

[例4]：0.75，十进制0.75转换成二进制0.11

规格化二进制数1.1*2^-1　计算指数-1+127=126

符号位 指数部分 尾数部分

0 01111110 10000000000000000000000

以单精度（real*4）浮点格式存储该数

0011 1111 0100 0000 0000 0000 0000 0000　0x3F40 0000

[例5]：-2.5 十进制-2.5转换成二进制-10.1

规格化二进制数-1.01*2^1

计算指数1+127=128

符号位 指数部分 尾数部分

1 10000000 01000000000000000000000

以单精度（real*4）浮点格式存储该数

1100 0000 0010 0000 0000 0000 0000 0000

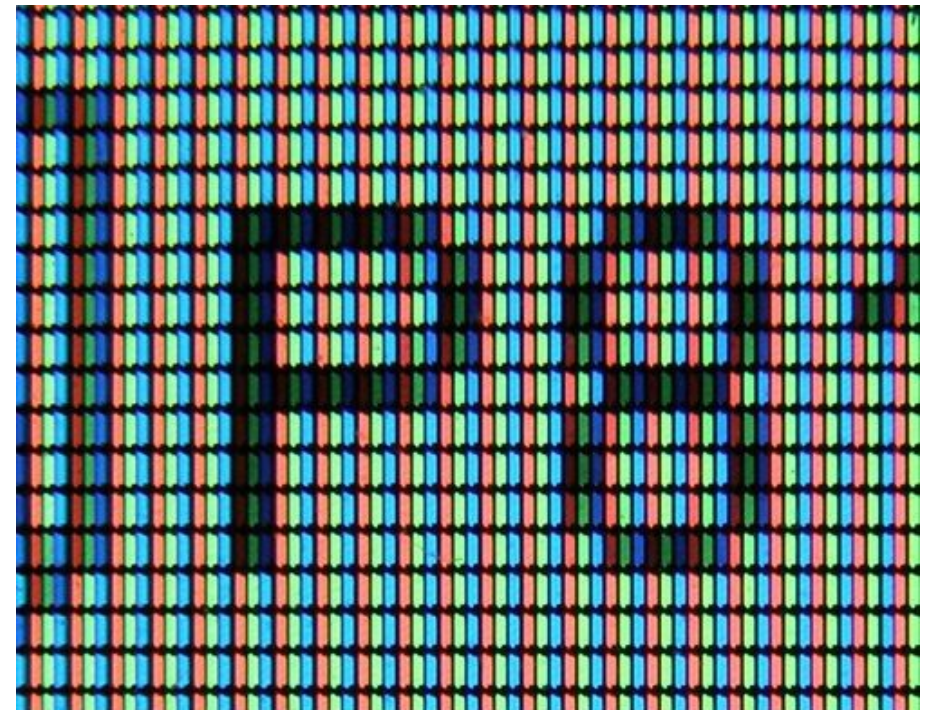
0xC020 0000

Bits You can See

视觉显示的最小元素称为“像素”。 像素具有生成大部分可感知颜色范围的三个独立颜色分量。

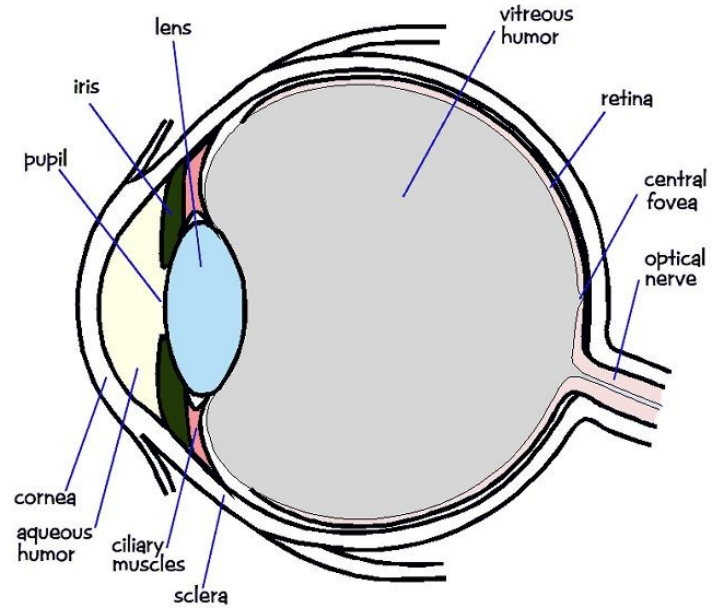
为什么是三个颜色分量，它们是什么？它们在计算机中是如何表示的?

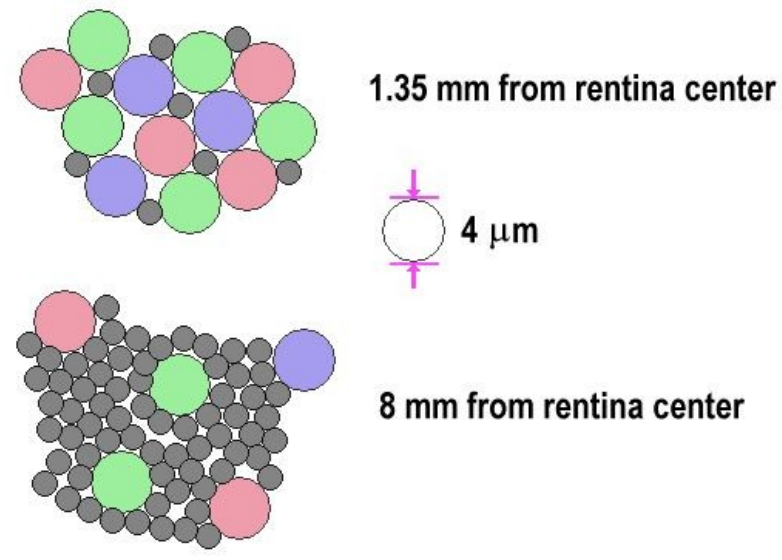
首先，让我们讨论一下可感知的概念



It starts with the Eye

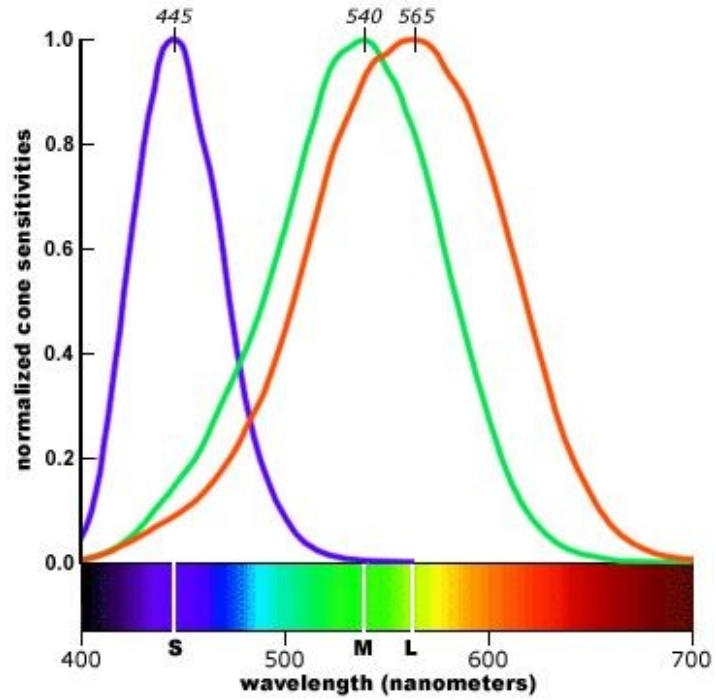
- 眼睛的感光部分称为视网膜。
- 视网膜主要由两种细胞类型组成，称为视杆细胞和视锥细胞。
- 视锥细胞负责颜色感知。视锥细胞在中央凹内最密集。共有三种类型的锥体，称为 S、M 和 L，其光谱灵敏度随波长而变化。

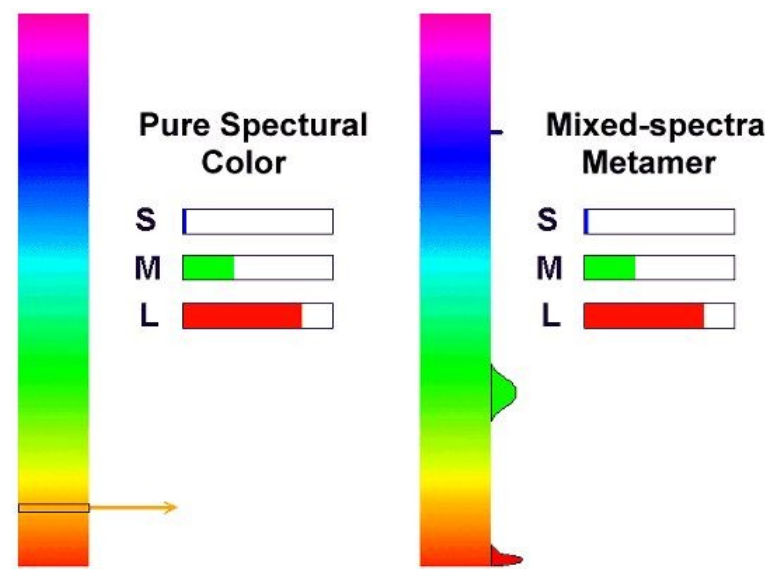




Why we see in color

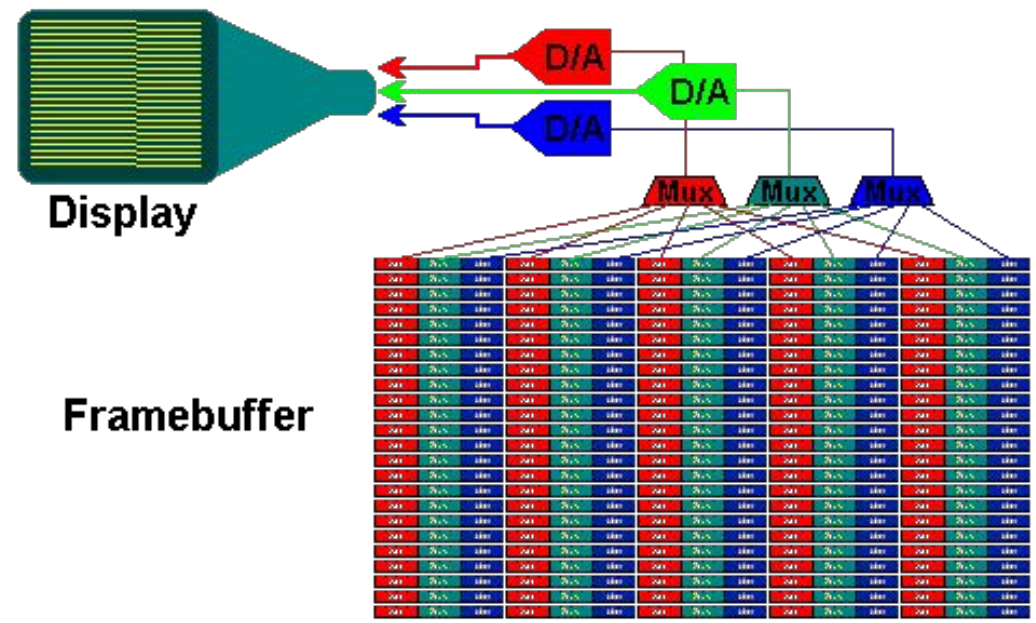
- 纯光谱颜色在某种程度上模拟了所有的视锥细胞。混合多种颜色可以刺激视锥细胞做出与纯色无异的反应。
- 感知相同的感觉被称为条件等色.
- 这允许我们只使用三种颜色来生成所有其他颜色。

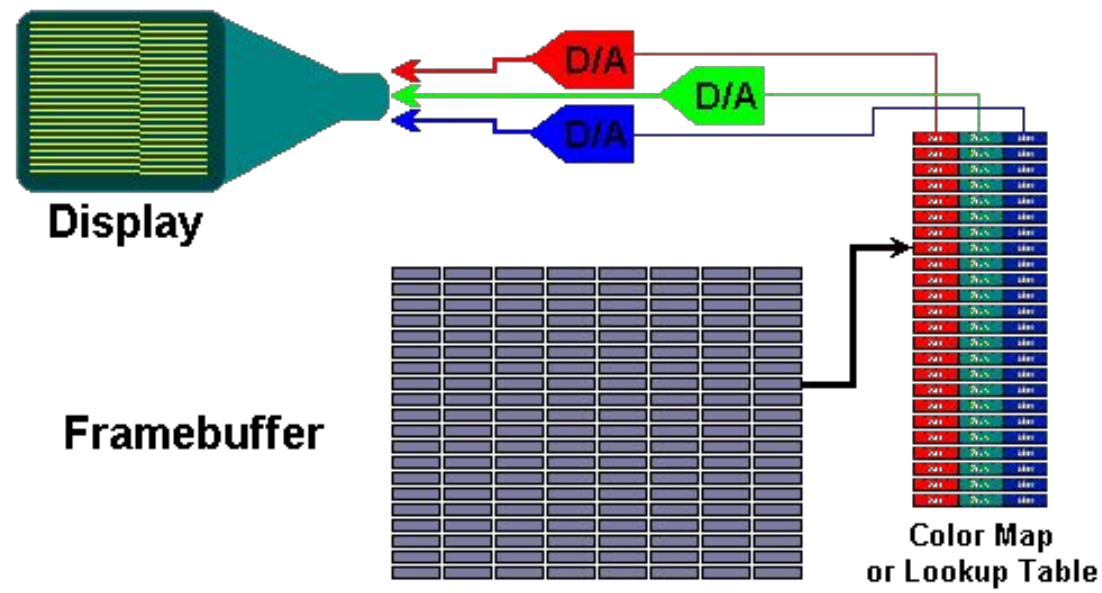


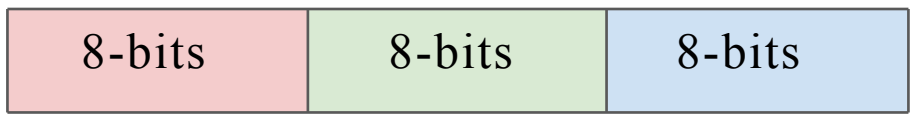


How colors Are Represented
颜色是如何表示的

- 每个像素存储为三个主要部分红色、绿色和蓝色，通常每个通道大约 8 位。
- 像素可以有单独的 R、G、B 分量或它们可以通过“查找表”间接存储







3 - 8-bit unsigned binary integers (0,255)
-OR-
3 - fixed point 8-bit values (0-1.0)

Color Specifications

Web colors:

Name	Hex	Decimal Integer	Fractional
Orange	#FFA500	(255, 165, 0)	(1.0, 0.65, 0.0)
Sky Blue	#87CEEB	(135, 206, 235)	(0.52, 0.80, 0.92)
Thistle	#D8BFD8	(216, 191, 216)	(0.84, 0.75, 0.84)

Colors are stored as binary too. You'll commonly see them in Hex, decimal, and fractional formats.

Summary

- 所有现代计算机都使用二进制补码表示有符号整数
- 二进制补码表示消除了对减法单元的需要
- 无符号数和补码数的加法相同
- 计算机上数字的有限表示可能会导致异常
- 浮点数有定点表示也有指数规格化表示