

《计算机系统》 ——位、字节、信息存储

湖南大学

《计算机系统》课程教学组



内容提要

用位来表示信息

01



02



位级操作

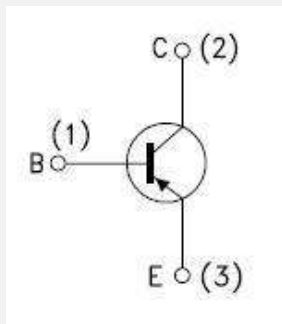
内存、指针与字符串

03



为什么是二进制？

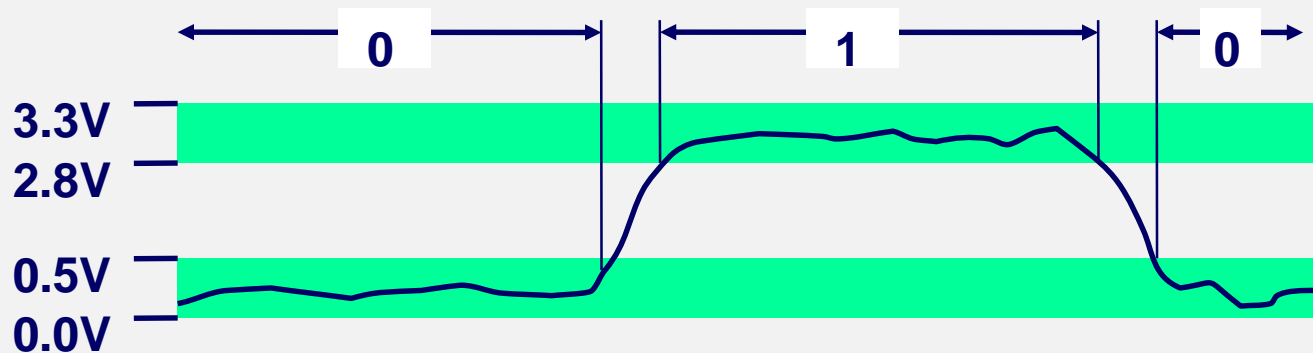
- 计算机是由**仅具有开关两个状态**的逻辑电路组成的；



- 二进制的**计算模式**比十进制的**简单**许多；
 - 加法与减法: XOR (异或)
 - 乘法与除法: AND (与)

为什么是二进制？

- 二进制电路具有较强的抗干扰性;



- 二进制便于布尔代数的运算

- &&

- ||

- !

二进制

- 使用一个选中的数**R**作为“**底**”，并用它的**幂**来作为“**权**”，这样就形成了用R来表示所有数的形式。

$$[x_{n-1} \dots x_0] = \sum_{k=0}^{n-1} x_k R^k$$

例如： $3582_{10} = 3 \times 10^3 + 5 \times 10^2 + 8 \times 10^1 + 2 \times 10^0$

所以二进制就是以2为底，2的幂次为权来表示数的一种形式

$$1110 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

- B → D

- 使用 “底-权公式”

$$[x_{n-1} \dots x_0] = \sum_{k=0}^{n-1} x_k R^k$$

e.g.

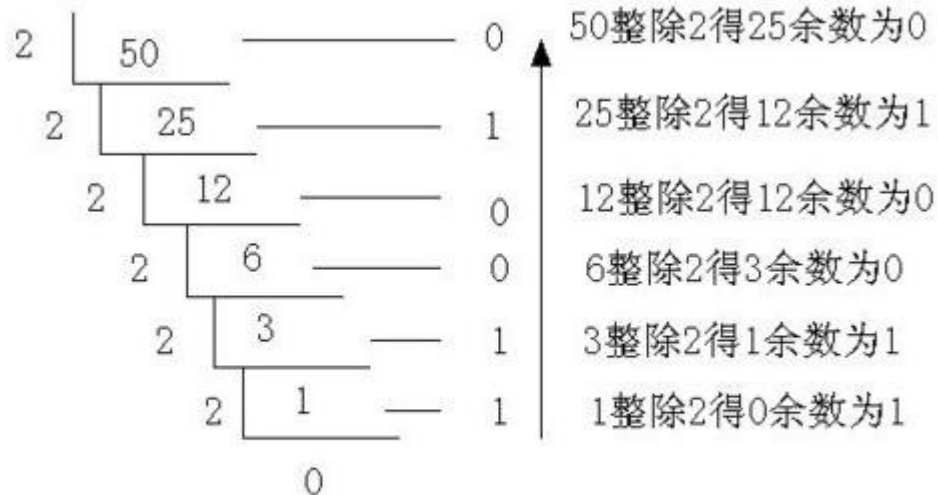
$$1110 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 14$$

熟记 2 的幂次会极大地方便二进制的计算！

.....8192, 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1

进制转换

● D → B



从下往上写出每个余数110010就是50的二进制表示。

● 也可以通过“2的幂次拼接”来快速完成

$$50 = 32 + 16 + 2 = 2^5 + 2^4 + 2^1$$

1	1	0	0	1	0
2^5	2^4	2^3	2^2	2^1	2^0

课堂习题1

$$78_{10} = (\quad)_2$$

$$01001010_2 = (\quad)_{10}$$

A. 01001101_2 138_{10}

B. 01001110_2 74_{10}

C. 01010011_2 40_{10}

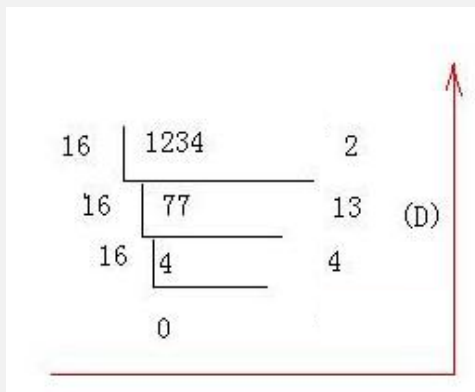
D. 01001110_2 138_{10}

十六进制

- 为了避免二进制表达式的不便之处，人们使用十六进制来获得更友好直观的数据的表达形式。
- $R=16$ ，计数符号为：
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- C语言中, 使用 0x***** 来标识十六进制数.

十六进制

- D→H



- 也可以通过“16的幂次拼接”来快速完成

$$552 = 2 \times 256(16^2) + 2 \times 16 + 8 \times 1(16^0)$$

2	2	8
16 ²	16 ¹	16 ⁰

- H → D

- 使用 “底-权公式”

$$[x_{n-1} \dots x_0] = \sum_{k=0}^{n-1} x_k R^k$$

例如：

$$0x7AF = 7 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 1967$$

熟记 16 的幂次会极大地方便十六进制的计算！

.....65536, 4096, 256, 16, 1

课堂习题2

$$260_{10} = (\quad)_{16}$$

$$1A5_{16} = (\quad)_{10}$$

A. 118_{16} 437_{10}

B. 150_{16} 448_{10}

C. 104_{16} 421_{10}

D. 128_{16} 406_{10}

进制转换

- **H → B**

将十六进制数的每一位转换为二进制的表达形式:

4	A	7	D	hex
0100	1010	0111	1101	binary

- **B → H**

从二进制数的最低位 (LSB) 往最高位 (MSB) 按照每四位一组来分组，每一组转换为一个十六进制数.

1010		1110		0110		1001	binary
A		E		6		9	hex

进制关系

- 1 Byte = 8 bits

- 二进制 00000000_2 to 11111111_2
- 十进制 0_{10} to 255_{10}
- 十六进制 00_{16} to FF_{16}
 - 16个字符
 - '0' to '9' and 'A' to 'F'
 - 在C语言中 $FA1D37B_{16}$ 可表示为 $0xFA1D37B$ 或 $0xfa1d37b$

最低有效位 (the Least Significant Bit, LSB) 是指一个二进制数字中的第0位 (即最低位), 具有权值为 2^0

最高有效位 (the Most Significant Bit, MSB) 是指一个n位二进制数字中的n-1位, 具有最高的权值 2^{n-1}

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

数据类型

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8

内容提要

用位来表示信息

01



02



位级操作

内存、指针与字符串

03



布尔代数

- 乔治布尔于19世纪开创了布尔代数
 - 属于逻辑的代数表达形式
 - 定义 “真” 的值为1, “假” 的值为0

And (与)

- $A \& B = 1$ when both $A=1$ and $B=1$

$\&$	0	1
0	0	0
1	0	1

Not (非)

- $\sim A = 1$ when $A=0$

\sim	
0	1
1	0

Or (或)

- $A \mid B = 1$ when either $A=1$ or $B=1$

\mid	0	1
0	0	1
1	1	1

Exclusive-Or (Xor , 异或)

- $A \wedge B = 1$ when either $A=1$ or $B=1$, but not both

\wedge	0	1
0	0	1
1	1	0

位运算

- 位向量的运算

- 按“位”进行逻辑运算

01101001	01101001
& 01010101	01010101
<u> </u>	<u> </u>
01000001	01111101

01101001	~ 01010101
^ 01010101	
<u> </u>	<u> </u>
00111100	10101010

- 布尔代数的各种逻辑完全适用

And.....Or.....Xor		
0.....0.....0	0.....0.....0	0.....0.....0
1.....0.....0	0.....1.....1	1.....1.....1
0.....1.....0	0.....1.....1	1.....1.....1
1.....1.....1	1.....1.....1	0.....0.....0

位运算

- **C语言中有所有位运算操作符：&, |, ~, ^**

- 适用于所有“整型”数据：long, int, short, char, unsigned
- 将参数视为位向量
- 参数是按位来参与运算

- **举例 (Char 数据)**

- $\sim 0x41 \rightarrow 0xBE$

$$\sim 01000001_2 \rightarrow 10111110_2$$

- $\sim 0x00 \rightarrow 0xFF$

$$\sim 00000000_2 \rightarrow 11111111_2$$

- $0x69 \& 0x55 \rightarrow 0x41$

$$01101001_2 \& 01010101_2 \rightarrow 01000001_2$$

- $0x69 | 0x55 \rightarrow 0x7D$

$$01101001_2 | 01010101_2 \rightarrow 01111101_2$$

- **逻辑运算符**

- $\&\&$, $\|$, $!$
- 定义 '0' 为假,所有非0值为真
- 返回值非0即1

- **举例 (Char 数据)**

- $!0x41 \rightarrow 0x00$
- $!0x00 \rightarrow 0x01$
- $!!0x41 \rightarrow 0x01$
- $0x69 \&\& 0x55 \rightarrow 0x01$
- $0x69 \| 0x55 \rightarrow 0x01$

移位运算

- **左移:** $x \ll y$

- 将位向量 x 左移 y 位

左边多余位均舍弃

右侧用 0 补齐

- **右移:** $x \gg y$

- 将位向量 x 右移 y 位

右边多余位均舍弃

- 逻辑右移

左边以 0 补齐

- 算术右移

左侧用最高位补齐

- **未定义运算**

移位值 < 0 或 \geq 字长

Argument x	01100010
$\ll 3$	00010000
Log. $\gg 2$	00011000
Arith. $\gg 2$	00011000

Argument x	10100010
$\ll 3$	00010000
Log. $\gg 2$	00101000
Arith. $\gg 2$	11101000

内容提要

用位来表示信息



01



02



03

内存、指针与字符串

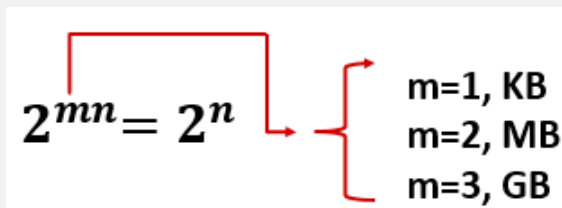
位级操作

基于字节的 内存模式

- 已知一条地址总线的位数，如何计算其所能表达的地址空间大小？

$$2^{10} = 1,024 = 1K, \quad 2^{20} = 1,048,576 = 1M, \quad 2^{30} = 1.09 \times 10^{12} = 1G$$

如果该总线有“mn”位(线), 那么,

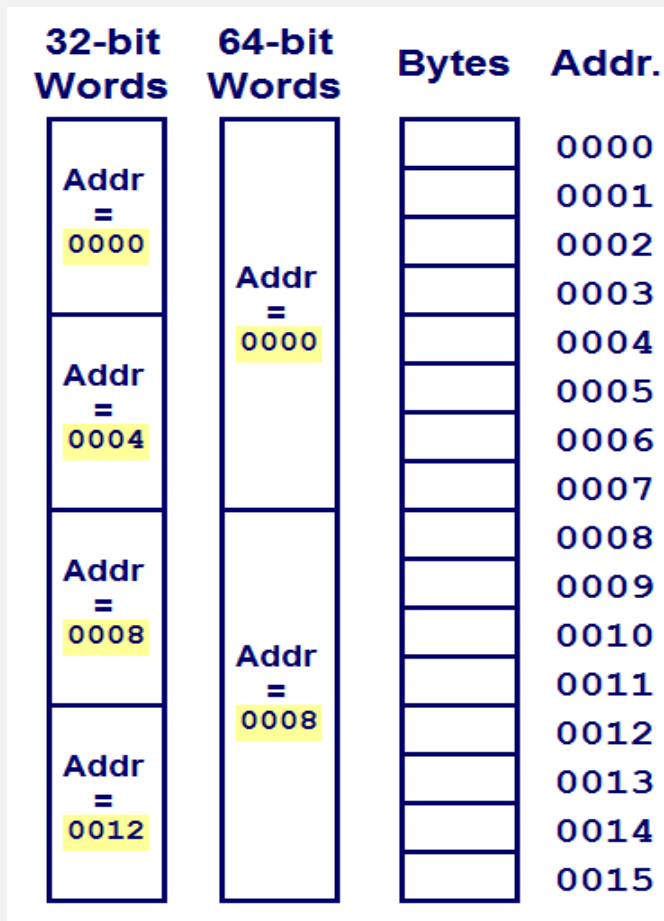

$$2^{mn} = 2^n$$

m=1, KB
m=2, MB
m=3, GB

例如：一条地址总线有13根地址线，则它表达的地址空间为：

$$2^{13} \text{Byte} = 2^3 \text{KB} = 8\text{KB}$$

- 一个内存地址存放的是一个字节 (8个bit) !
- 地址指明的是某个字节 (8bit) 的存放位置
 - 即某数据第一个字节所在的位置
 - 因此连续存放的两个字之间相隔4个地址 (32位字长) 或者8个地址 (64位字长)



字节存放顺序

- 一个多字节数据是怎样在内存中存放的？
- 2大传统模式
 - 大端法: Sun, PPC Mac, Internet : **高位字节占据低地址**
 - 小端法: x86 : **低位字节占据低地址**
- **举例** : 4字节变量 x 的值是 : 0x01234567, 存放在 0x100

大端法

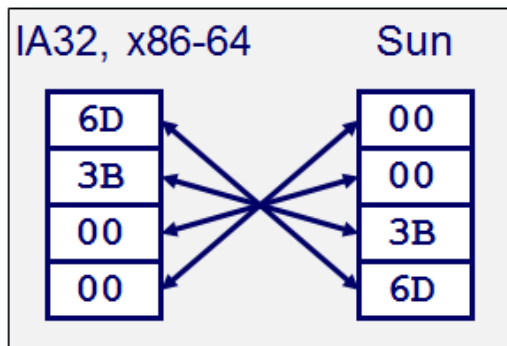
0x100		0x101	0x102	0x103		
		01	23	45	67	

小端法

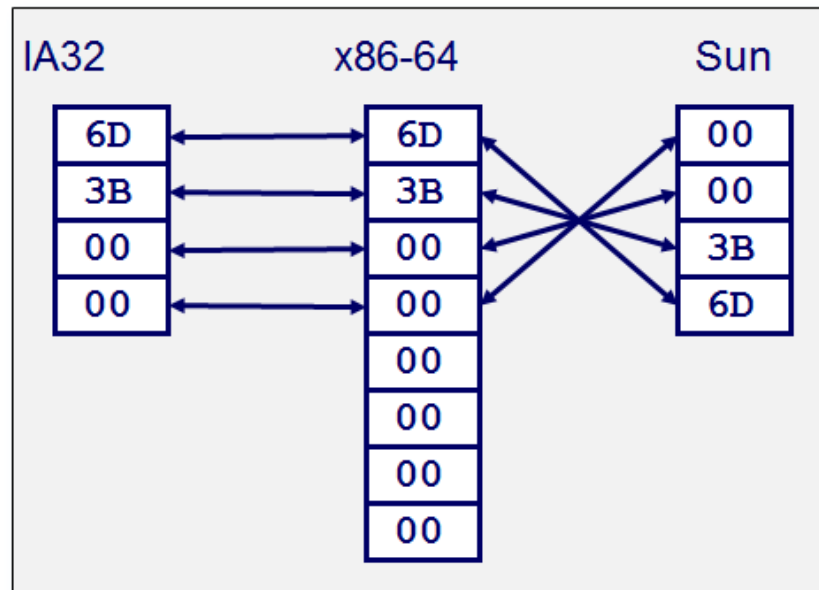
0x100		0x101	0x102	0x103		
		67	45	23	01	

大小端法举例2

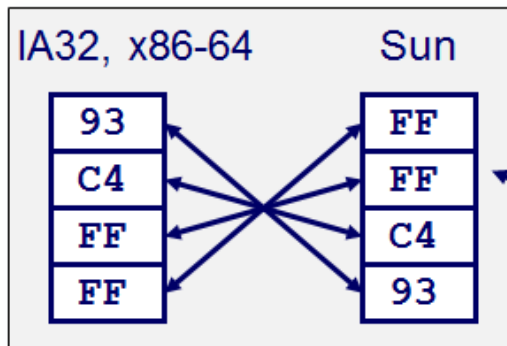
int A = 15213;



long int C = 15213;



int B = -15213;



补码表示

Decimal: 15213

Binary: 0011 1011 0110 1101

Hex: 3 B 6 D

指针表示

```
int B = -15213;  
int *P = &B;
```

Sun

EF
FF
FB
2C

IA32

D4
F8
FF
BF

x86-64

0C
89
EC
FF
FF
7F
00
00

不同的编译器和计算机为程序及数据分配不同的存储位置

字符串表示

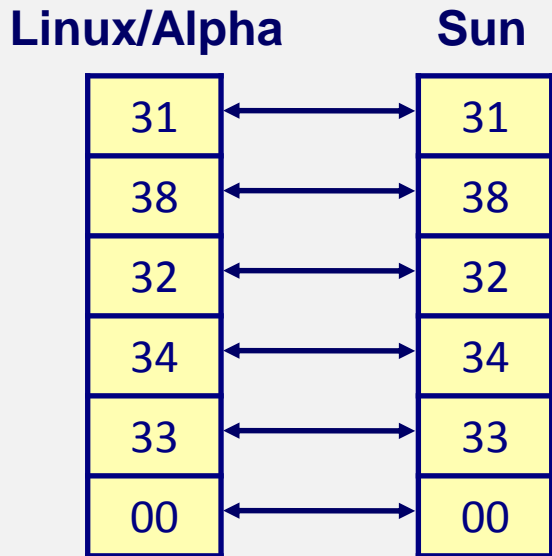
● C语言的字符串

- 由字符的数组表示
- 每一个字符由 ASCII 码来表示
 - 标准的7位字符集编码
 - 字符 '0' 的ASCII码为 0x30
 - 数 i 的ASCII码为 $0x30+i$
- 字符串必须以一个空字符表示结尾
 - 也就是说最后一个字符 = 0

● 兼容性

- 大端或小端存储不会影响字符串存储排列

```
char S[6] = "18243";
```



ASCII表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位 低四位		ASCII控制字符										ASCII打印字符															
		0000					0001					0010		0011		0100		0101		0110		0111					
		0					1					2		3		4		5		6		7					
十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl	
0000	0	0		^@	NUL	\0	空字符	16	►	^P	DLE	数据链路转义	32		48	0	64	@	80	P	96	`	112	p			
0001	1	1	☺	^A	SOH		标题开始	17	◄	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q			
0010	2	2	☹	^B	STX		正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r			
0011	3	3	♥	^C	ETX		正文结束	19	!!	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s			
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4	设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t			
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK	否定应答	37	%	53	5	69	E	85	U	101	e	117	u			
0110	6	6	♠	^F	ACK		肯定应答	22	—	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v			
0111	7	7	•	^G	BEL	la	响铃	23	↕	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w			
1000	8	8	▢	^H	BS	lb	退格	24	↑	^X	CAN	取消	40	(56	8	72	H	88	X	104	h	120	x			
1001	9	9	○	^I	HT	lt	横向制表	25	↓	^Y	EM	介质结束	41)	57	9	73	I	89	Y	105	i	121	y			
1010	A	10	◉	^J	LF	ln	换行	26	→	^Z	SUB	替代	42	*	58	:	74	J	90	Z	106	j	122	z			
1011	B	11	♂	^K	VT	lv	纵向制表	27	←	^[ESC	le	溢出	43	+	59	;	75	K	91	[107	k	123	{		
1100	C	12	♀	^L	FF	lf	换页	28	└	^_	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124				
1101	D	13	♪	^M	CR	lr	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93]	109	m	125	}			
1110	E	14	🎵	^N	SO		移出	30	▲	^^	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~			
1111	F	15	🎵	^O	SI		移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	△		*Backspace 代码: DEL	

下一节：整数

湖南大学

《计算机系统》课程教学组

