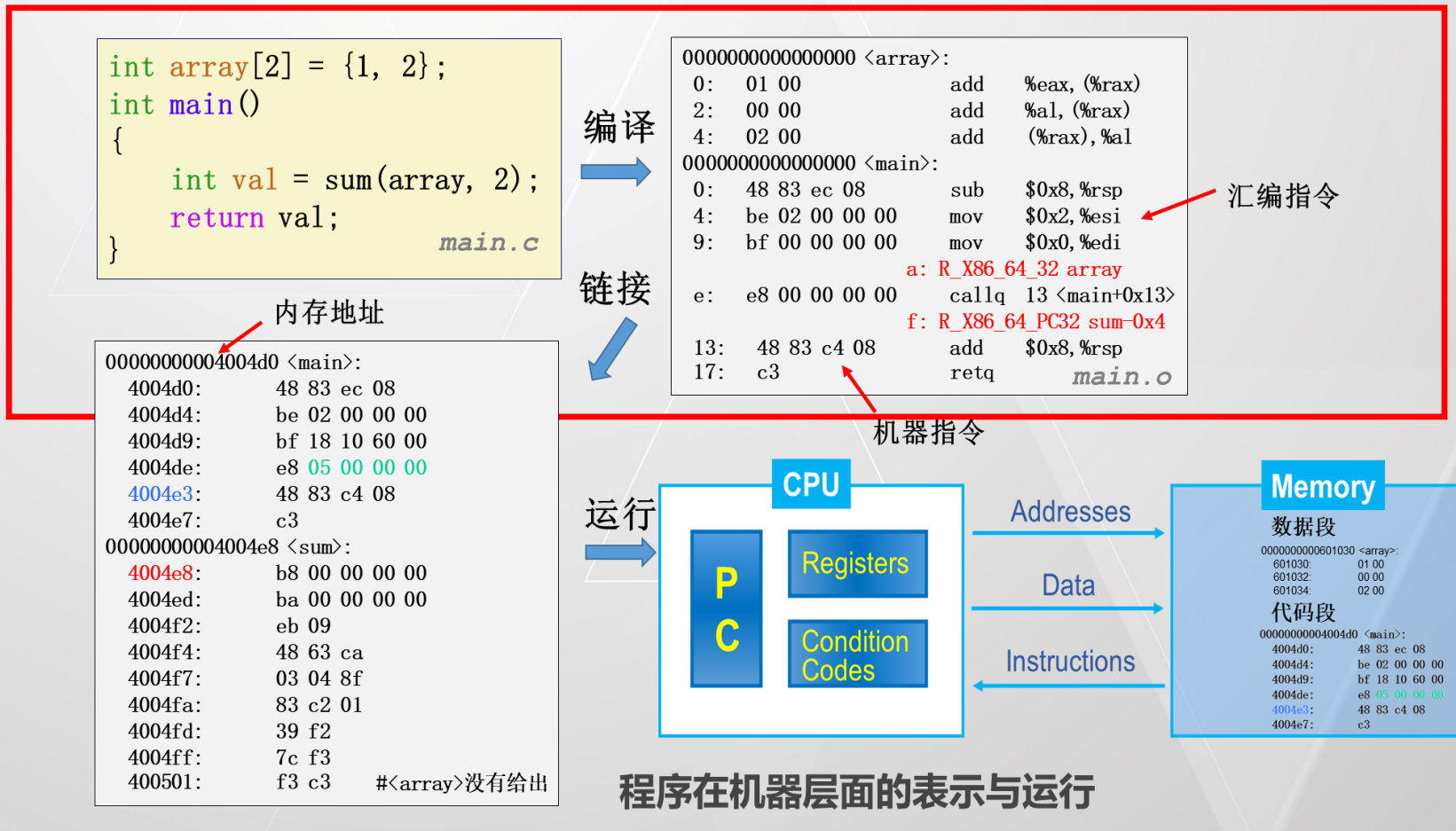
汇编语言程序设计

第2节　　整　　数

C程序在硬件层面的表示

- 数据
  - 整数（第二讲）
  - 浮点数（第三讲）
  - 数组、结构（第八讲）
- 代码
  - 基本概念/基本指令/寻址方式（第五讲）
  - 程序控制流与相关指令（第六讲）
  - 函数调用与相关指令（第七讲）



程序在机器层面的表示与运行

目录

预备知识

1K = 2^{10} = 1024（Kilo）
1M = 1024K = 2^{20} （Mega）
1G = 1024M = 2^{30} （Giga）
1T = 1024G = 2^{40} （Tera）
1P = 1024T = 2^{50} （Peta）
1E = 1024P = 2^{60} （Exa）

1个二进制位：bit （比特）
8个二进制位：Byte （字节） 1Byte = 8bit
2个字节：Word （字）
1Word = 2Byte = 16bit*

*X86架构下如此，MIPS或RISC-V的话1-word = 32-bit

数制

数　　制		基 数	数　　码
二进制	Binary	2	0, 1
八进制	Octal	8	0, 1, 2, 3, 4, 5, 6, 7
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

数制之间的转换

十六进制数：逢十六进一　借一当十六

$$\begin{array}{r} 05C3\ H \\ +\ 3_{1}D25\ H \\ \hline 42E8\ H \end{array} \qquad \begin{array}{r} 3\overset{-1}{D}25\ H \\ -\ 05C3\ H \\ \hline 3762\ H \end{array}$$

逻辑运算（按位操作）

“与”运算（AND）

A	B	A∧B （&）
0	0	0
0	1	0
1	0	0
1	1	1

“或”运算（OR）

A	B	A∨B （|）
0	0	0
0	1	1
1	0	1
1	1	1

“非”运算（NOT）

A	¬A （~）
0	1
1	0

“异或”运算（XOR）

A	B	A∀B （^）
0	0	0
0	1	1
1	0	1
1	1	0

逻辑运算（按位操作）

例：X = 00FFH　Y = 5555H，Z = X∀Y = ?

X = 0000 0000 1111 1111 B

∀ Y = 0101 0101 0101 0101 B

Z = 0101 0101 1010 1010 B

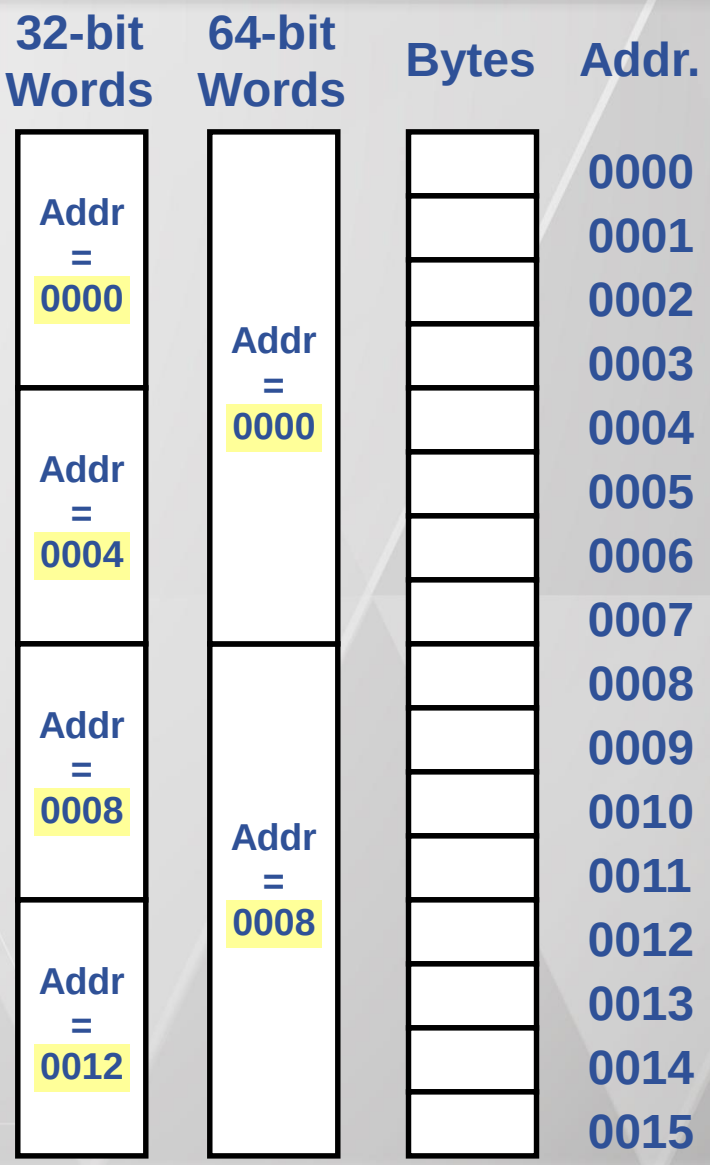
∴ Z = 55AAH

数的机器表示

- **机器字(machine word)长**
  - **一般指计算机进行一次整数运算所能处理的二进制数据的位数**
    - 通常也指数据地址长度
  - **32位字长**
    - 地址的表示空间是4GB
    - 对很多内存需求量大的应用而言，非常有限
  - **64位字长**
    - 地址的表示空间约是1.8 X 10^{19} bytes
    - 目前的X86-64 机型实际支持 48位宽的地址: 256 TB

机器字在内存中的组织

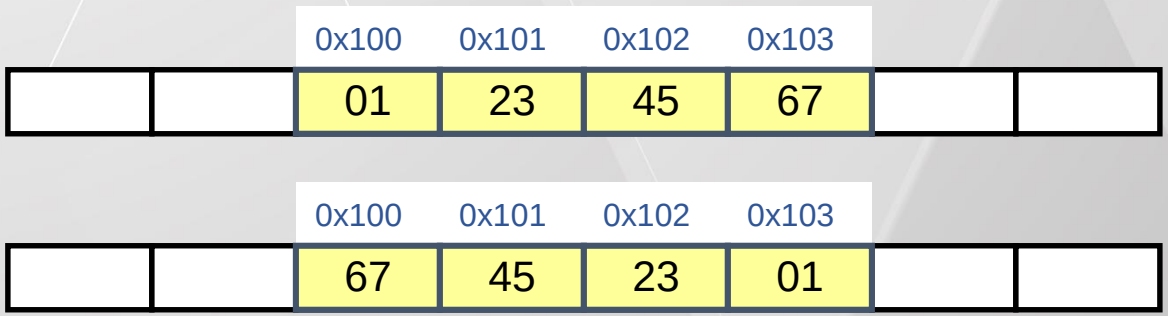
- **地址按照字节(byte)来定位**
  - 机器字中第一个字节的地址
  - 相邻机器字的地址相差4 (32-bit) 或者8 (64-bit)



字节序（Byte Ordering）

- 一个机器字内的各个字节如何排列?
  - **Big Endian: Sun, PowerPC, Internet, Java**
    - 低位字节(Least significant byte, LSB) 占据高地址
  - **Little Endian: x86**
    - 与LSB相反

数值是0x01234567，地址是0x100

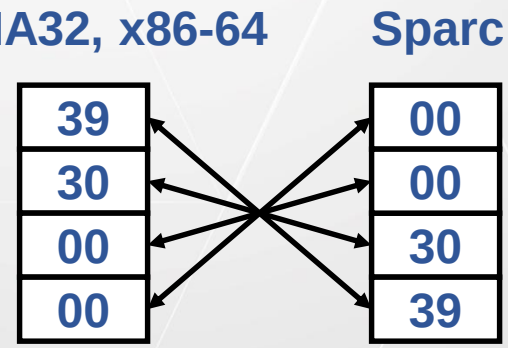


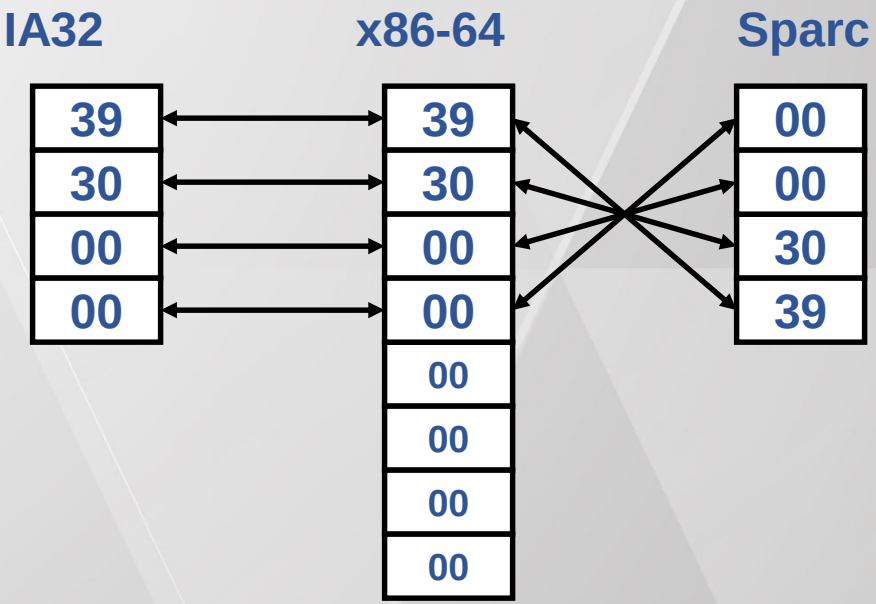
Gulliver's Travels

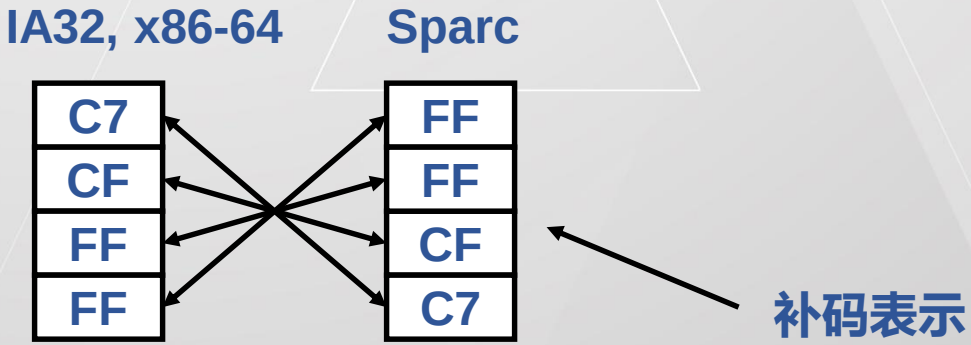
字节序（Byte Ordering）

- `int A = 12345;`
- `int B = -12345;`
- `long int C = 12345;`

Decimal:	12345
Binary:	0011 0000 0011 1001
Hex:	3 0 3 9







整数表示

- C语言中基本数据类型的大小 (in Bytes)

C Data Type	Typical 32-bit	x86-32	x86-64	
char	1	1	1	
short	2	2	2	
int	4	4	4	
long	4	4	8	(Linux)
long long	8	8	8	
float	4	4	4	
double	8	8	8	
long double	8	10/12	10/16	
char *	4	4	8	

  - Or any other pointer

计算机中整数的二进制编码方式（w表示字长）

无符号数（原码表示）

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

带符号数（补码，Two's Complement）

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

符号位

```
short int x =  12345;
short int y = -12345;
```

	Decimal	Hex	Binary
x	12345	30 39	00110000 00111001
y	-12345	CF C7	11001111 11000111

- 符号位（sign bit）
  - 对于补码表示, MSB （Most Significant Bit）表示整数的符号
    - 0 for nonnegative
    - 1 for negative

非负数：补码 = 原码　　负数：补码 = 反码 + 1 = 2^w + 该负数（反码是原码各位取反）

一个数的补码表示是它的相反数的补码表示的按位取反加一

取值范围

- 无符号数
  - *UMin* = 0
    000…0
  - *UMax* = $2^w - 1$
    111…1
- 带符号数（补码）
  - *TMin* = -2^{w-1}
    100…0
  - *TMax* = $2^{w-1} - 1$
    011…1
- Other Values
  - 负1 = 111…1

假设字长为16(w=16)

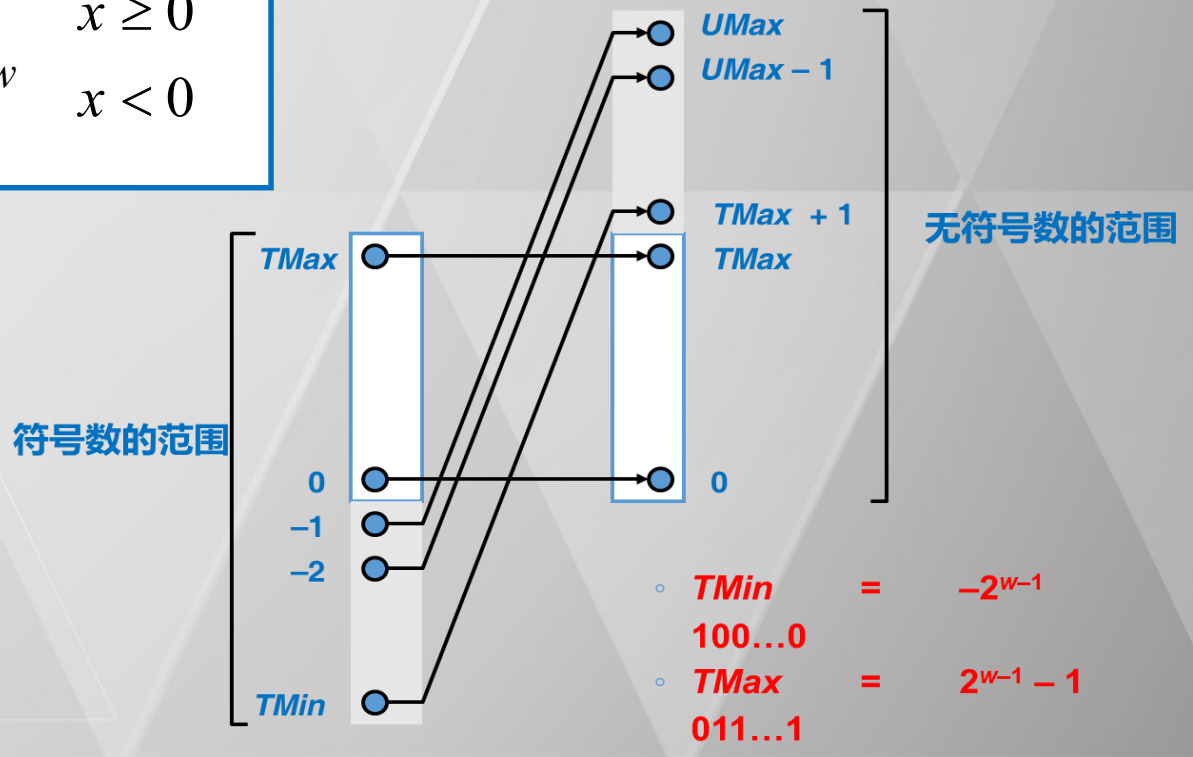
	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

无符号数与带符号数

X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

无符号数与带符号数之间的转换：
二进制串的表示是不变的。

$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$



补码加法公式*

$$[x]_{补}+[y]_{补}\equiv[x+y]_{补}\ (\mathrm{mod}\ 2^{w})$$

意义：负整数用补码表示后，可以和正整数一样来处理。这样，（处理器的）运算器里只需要一个加法器就可以

$$[y]_{补}=2^{w}+y\ (y<0)$$

证明：

(1) x>0;y>0 由于正数的补码和原码一致，x+y>0。所以我们可以发现在这种情况下$[x]_{补}+[y]_{补}=[x+y]_{补}$

(2) x>0;y<0且x+y>0 我们有如下的等式：$[x]_{补}=x$ $[y]_{补}=2^{w}+y$

所以，$[x]_{补}+[y]_{补}=x+y+2^{w}=x+y=[x+y]_{补}$:

(3) x>0;y<0且x+y<0，可以发现以下等式：$[x]_{补}=x$ $[y]_{补}=2^{w}+y$

所以，$[x]_{补}+[y]_{补}=x+y+2^{w}=[x+y]_{补}$:

……

C语言中的无符号数与带符号数

- 常数（Constants）
  - 默认是带符号数
    - 如果有“U”作为后缀则是无符号数，如 0U，4294967259U
- 如果无符号数与带符号数混合使用，则带符号数默认转换为无符号数
  - 包括比较操作符
  - 实例（w=32）

$Constant_1$	$Constant_2$	Relation
0	0U	
-1	0	
-1	0U	
2147483647	-2147483647-1	
2147483647U	-2147483647-1	
-1	-2	
(unsigned) -1	-2	
2147483647	2147483648U	
2147483647	(int) 2147483648U	

C语言中的无符号数与带符号数

Constant$_1$	Constant$_2$	Relation	Evaluation
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483647-1	>	signed
2147483647U	-2147483647-1	<	unsigned
-1	-2	>	signed
(unsigned) -1	-2	>	unsigned
2147483647	2147483648U	<	unsigned
2147483647	(int) 2147483648U	>	signed

何时采用无符号数

- 模运算
- 按位运算

- 建议：不能仅仅因为取值范围是非负而使用

示例一

```
unsigned i;
for (i = cnt-2; i >= 0; i--)
  a[i] += a[i+1];
```

示例二

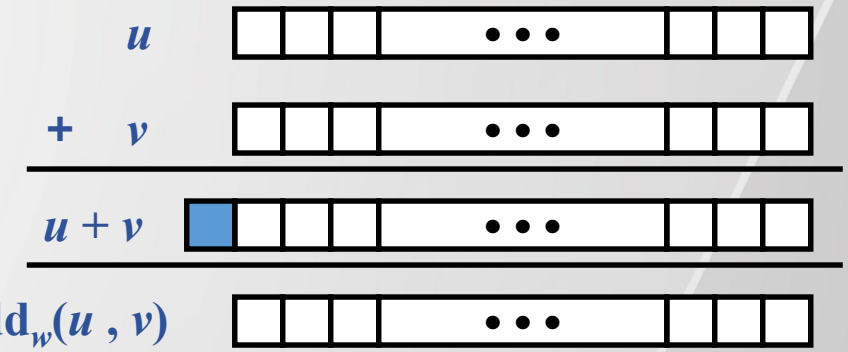
```
#define DELTA sizeof(int)
int i;
for (i = CNT; i-DELTA >= 0; i-= DELTA)
  . . .
```

无符号数加法

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



$$s = \mathrm{UAdd}_w(u, v) = (u + v) \bmod 2^w$$

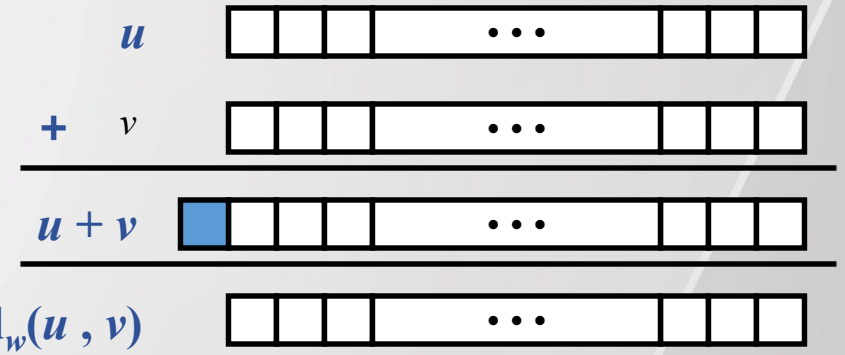
$$UAdd_w(u,v) = \begin{cases} u+v & u+v < 2^w \\ u+v-2^w & u+v \geq 2^w \end{cases}$$

补码加法

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits

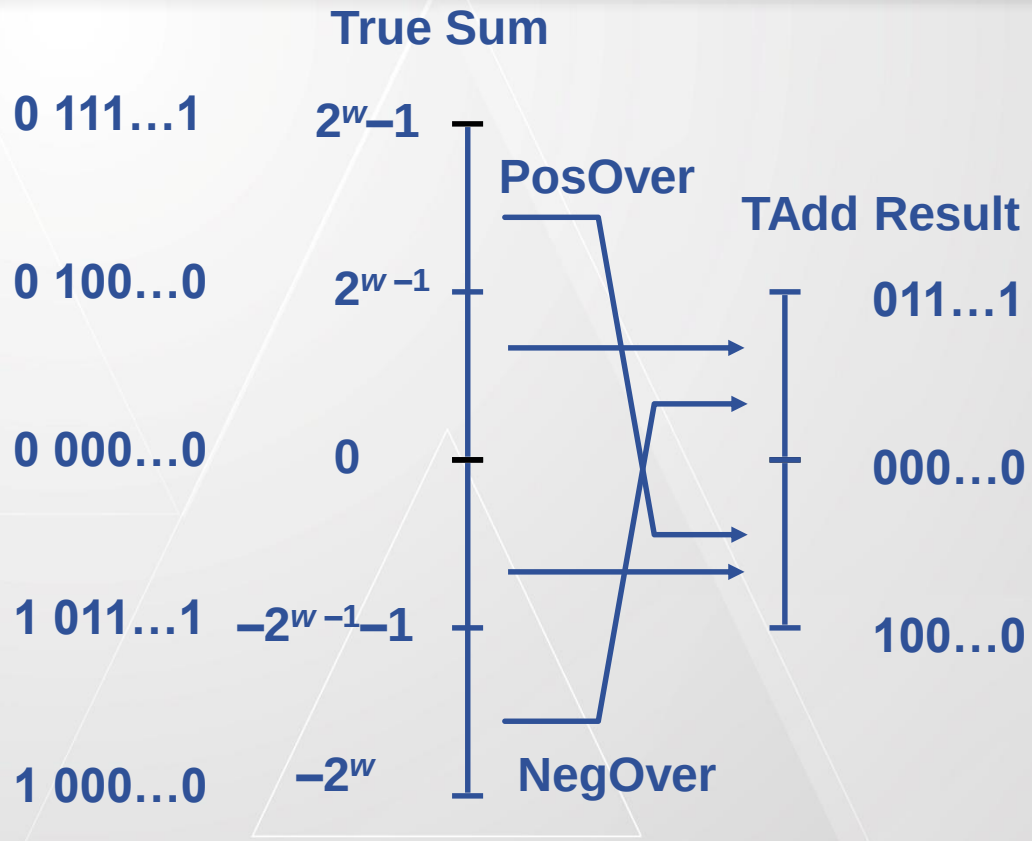


- 与无符号数的一致
  - Signed vs. unsigned addition in C:

```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v
```

  - `s == t`

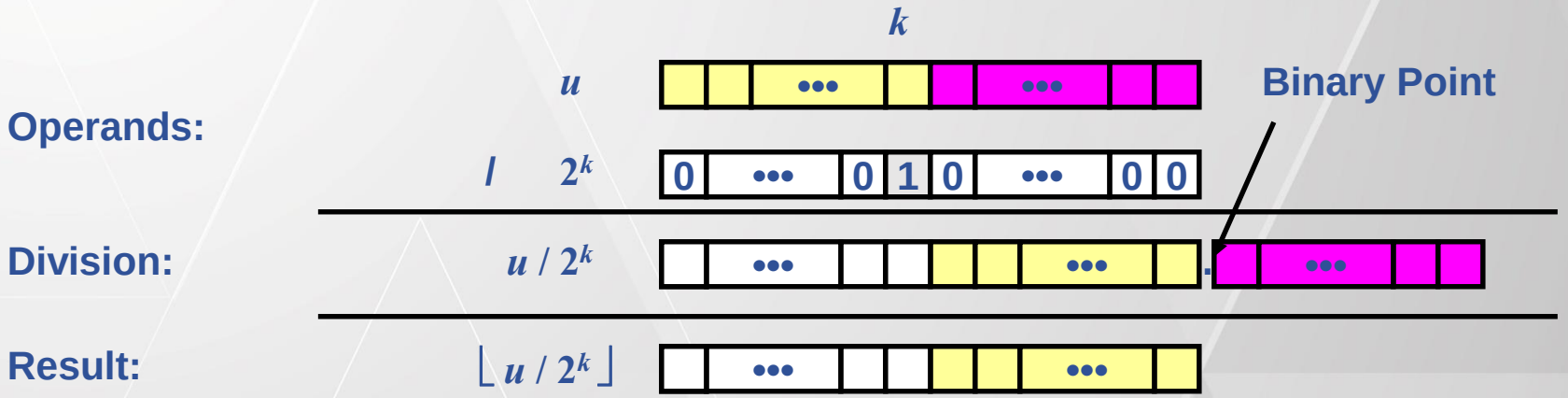
补码加法的溢出



$$TAdd_w(u,v) = \begin{cases} u+v+2^w & u+v < TMin_w \quad \text{(NegOver)} \\ u+v & TMin_w \le u+v \le TMax_w \\ u+v-2^w & TMax_w < u+v \quad \text{(PosOver)} \end{cases}$$

无符号整数除以2的k次幂

- u >> k gives $\lfloor u / 2^k \rfloor$
- 采用逻辑右移



	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	00011101 10110110
x >> 4	950.8125	950	03 B6	00000011 10110110
x >> 8	59.4257813	59	00 3B	00000000 00111011

无符号整数除以2的k次幂

C Function

```
unsigned udiv8(unsigned x)
{
  return x/8;
}
```

Compiled Arithmetic Operations

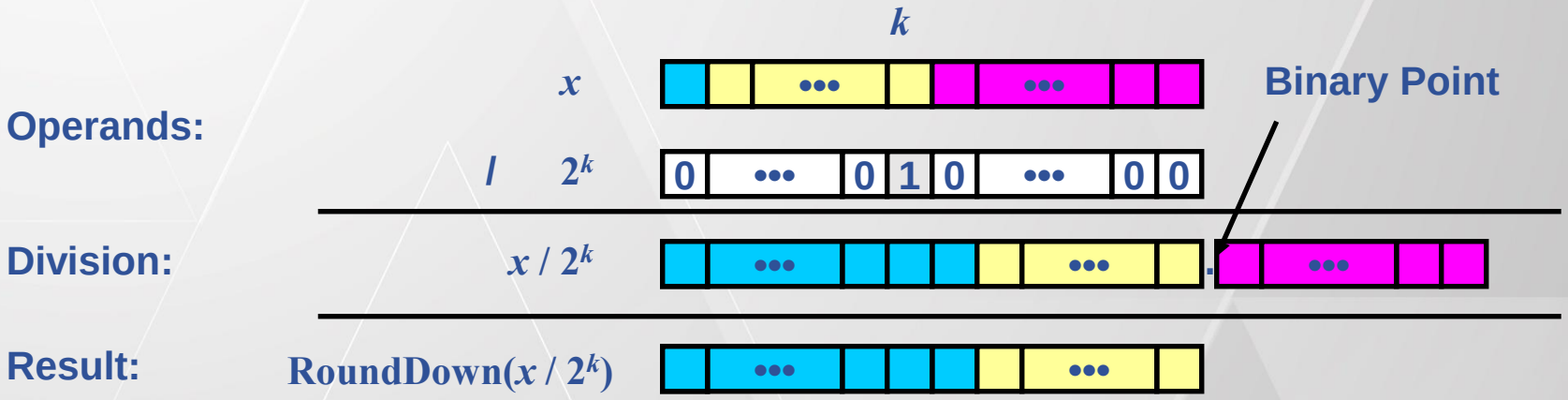
```
shrl      $3, %eax
```

Explanation

```
# Logical shift
   return x >> 3;
```

带符号整数除以2的幂

- x >> k gives $\lfloor x / 2^k \rfloor$
- 采用算术右移
  - 但是x < 0时，舍入错误

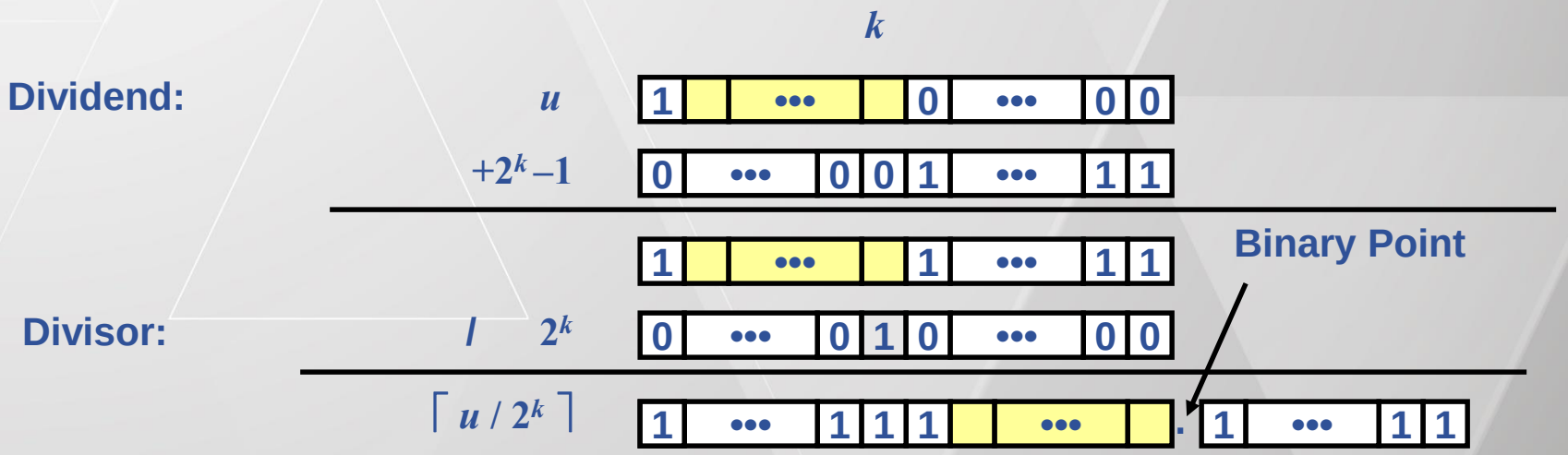


	Division	Computed	Hex	Binary
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606.5	-7607	E2 49	11100010 01001001
y >> 4	-950.8125	-951	FC 49	11111100 01001001
y >> 8	-59.4257813	-60	FF C4	11111111 11000100

带符号整数除以2的幂

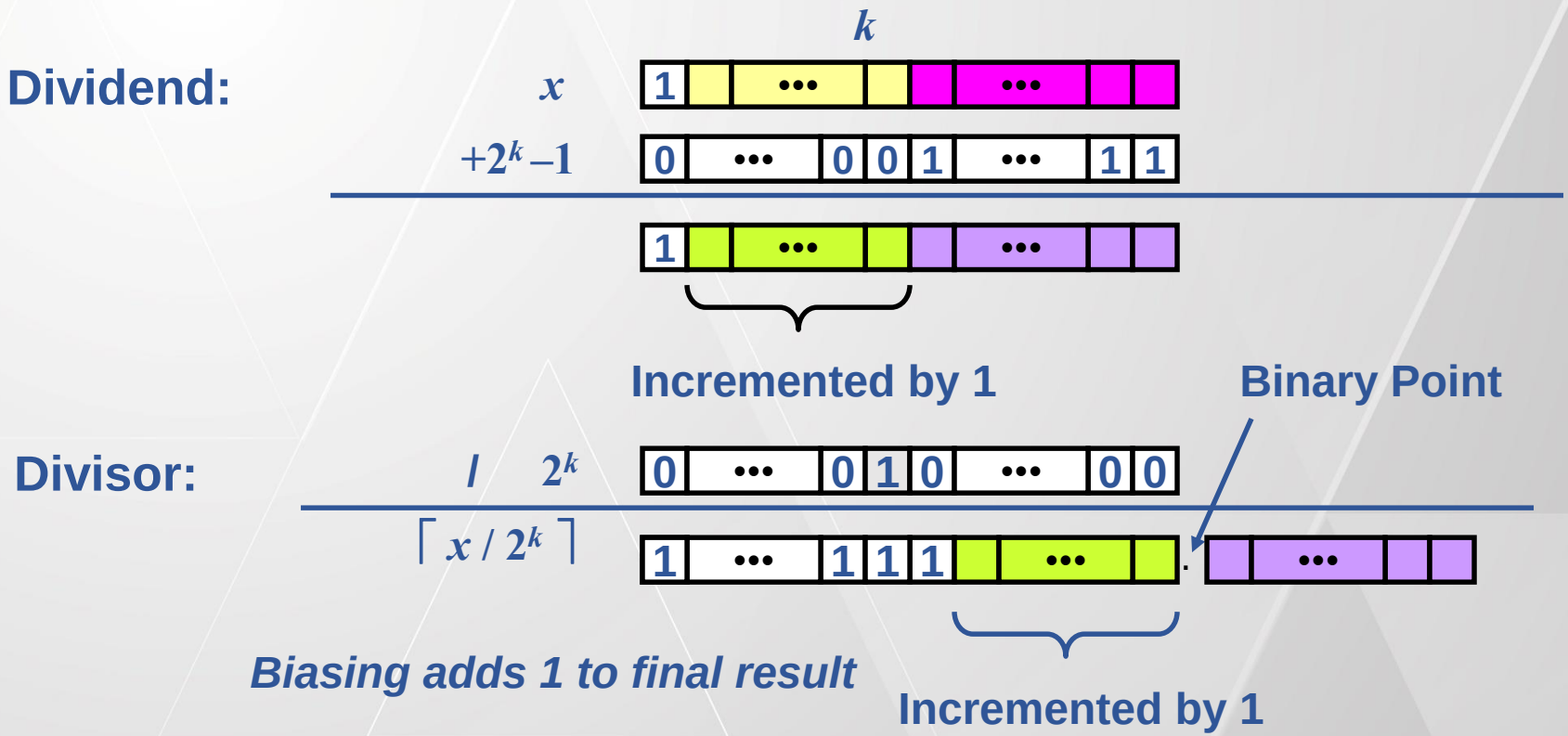
- Want ⌈ x / 2^k ⌉ (需要向0舍入，而不是向”下”舍入)
- Compute as ⌊ (x+2^k−1) / 2^k ⌋
  - In C: (x + (1<<k)−1) >> k
    - Biases dividend toward 0

Case 1: No rounding



Biasing has no effect

Case 2: Rounding



Case 2: Rounding

C Function

```
int idiv8(int x)
{
  return x/8;
}
```

Compiled Arithmetic Operations

```
testl %eax, %eax
   js       L4
L3:
   sarl     $3, %eax
   ret
L4:
   addl     $7, %eax
   jmp      L3
```

Explanation

```
if x < 0
        x += 7;
# Arithmetic shift
return x >> 3;
```

Integer C Puzzles

- 判断以下的推断或者等式是否成立(不成立则给出示例)
  - x, y 为32位带符号整数;

初始化

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

- x < 0 ⇒ ((x*2) < 0)
- ux >= 0
- x & 7 == 7 ⇒ (x<<30) < 0
- ux > -1
- x > y ⇒ -x < -y
- x > 0 && y > 0 ⇒ x + y > 0
- x >= 0 ⇒ -x <= 0
- x <= 0 ⇒ -x >= 0
- (x|-x) >> 31 == -1
- ux >> 3 == ux/8
- x >> 3 == x/8
- x & (x-1) != 0