Computer Arithmetic

Basic building block of modern computers – bits

Information is stored as a collection of bits.

Question: Why not other representations.

More generally, Base-b representation of N is as follows.

$$N = \sum_{i \geq 0} \alpha_i b^i \quad \leftarrow \alpha_0 \text{ is the least significant digit.}$$

Byte: 8 bits.

Binary representation: 00000000_2 to 11111111_2.

subscript to denote the base of the representation.

Note that binary representation is verbose. Can we have a shorter representation?

↳ what about decimal / base-10 representation?

↳ Conversion between base-2 ⟷ base-10 can be tedious.

Hexadecimal representation (base-16)

Byte ranges from 00_{16} to FF_{16}

"Digits" in base-16

$\{0, \ldots, 9\} \cup \{A, B, \ldots, F\}$.

(we do not impose on the case of the alphabets. Be consistent)

Notation in C: 0x_ _ (0: zero; x: letter x.)

$0xFF \equiv FF_{16} \equiv ff_{16} \equiv fF_{16}$ } Equivalent.

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111
Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

Figure 2.2 **Hexadecimal notation.** Each Hex digit encodes one of 16 values.

Examples:

a. Conversion of Hexadecimal repr to Binary repr.

b. Conversion of binary repr to hexadecimal repr.

Words: Word size is a measure of size of integer and pointer data.

w-bit word size $\Rightarrow$ 0 to $2^w - 1$ range of addressing is possible.

Program has access to 2^w bytes.

Recall that 32-bit computers had a limit of 4GB RAM

Limitation imposed by the fact that $\leq 2^{32} \approx 4 \times 10^9$ bytes of virtual memory is available.

Data sizes:

Word size = 4 bytes → 8 bytes

C declaration	32-bit	64-bit
`char`	1	1
`short int`	2	2
`int`	4	4
`long int`	4	8
`long long int`	8	8
`char *`	4	8
`float`	4	4
`double`	8	8

Figure 2.3 **Sizes (in bytes) of C numeric data types.**

Exact numbers depend on the machines and compilers.

Endian:

$$\underbrace{x_{w-1}\; x_{w-2} \cdots x_{w-8}}_{\text{most significant byte}} \;\cdots\; \underbrace{x_7\; x_6 \cdots x_0}_{\text{least significant byte}}$$

MSB first
Big Endian
Ex: IBM, SUN machines etc.

LSB first
Little Endian
Ex: Intel machines

Big endian

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

Little endian

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

Representation of 0x01234567.

Strings: ASCII representation.

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	<sp>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

Figure 1.2 **The ASCII text representation of** `hello.c`**.**

Boolean operations

~	
0	1
1	0

&	0	1
0	0	0
1	0	1

<table>
<tr><th>|</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>

^	0	1
0	0	1
1	1	0

Figure 2.7 **Operations of Boolean algebra.** Binary values 1 and 0 encode logic values TRUE and FALSE, while operations ~, &, |, and ^ encode logical operations NOT, AND, OR, and EXCLUSIVE-OR, respectively.

Syntax from propositional logic: $\neg, \wedge, \vee, \oplus$

These operations can be extended to bit vectors.

$$\begin{array}{r} 0110 \\ \& \ 1100 \\ \hline 0100 \end{array} \qquad \begin{array}{r} 0110 \\ | \ 1100 \\ \hline 1110 \end{array} \qquad \begin{array}{r} 0110 \\ \hat{} \ 1100 \\ \hline 1010 \end{array} \qquad \begin{array}{r} \\ \sim \ 1100 \\ \hline 0011 \end{array}$$

(Related notions: ^, & and ~ can be thought of as +, × and negation over Boolean ring.).

Further a bit vector of size/len n can be used to represent subsets of n elems.

Ex: $S = \{1, 2, 3, 4, 5\}$

01101 represents the subset $\{2, 3, 5\}$.

→ | and & correspond to set union and set intersection.

Note that these can be applied to integral data types in C.

Integer representations

C data type	Minimum	Maximum
`char`	−127	127
`unsigned char`	0	255
`short [int]`	−32,767	32,767
`unsigned short [int]`	0	65,535
`int`	−32,767	32,767
`unsigned [int]`	0	65,535
`long [int]`	−2,147,483,647	2,147,483,647
`unsigned long [int]`	0	4,294,967,295
`long long [int]`	−9,223,372,036,854,775,807	9,223,372,036,854,775,807
`unsigned long long [int]`	0	18,446,744,073,709,551,615

Figure 2.10 **Guaranteed ranges for C integral data types.** Text in square brackets is optional. The C standards require that the data types have at least these ranges of values.

C data type	Minimum	Maximum
`char`	−128	127
`unsigned char`	0	255
`short [int]`	−32,768	32,767
`unsigned short [int]`	0	65,535
`int`	−2,147,483,648	2,147,483,647
`unsigned [int]`	0	4,294,967,295
`long [int]`	−2,147,483,648	2,147,483,647
`unsigned long [int]`	0	4,294,967,295
`long long [int]`	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
`unsigned long long [int]`	0	18,446,744,073,709,551,615

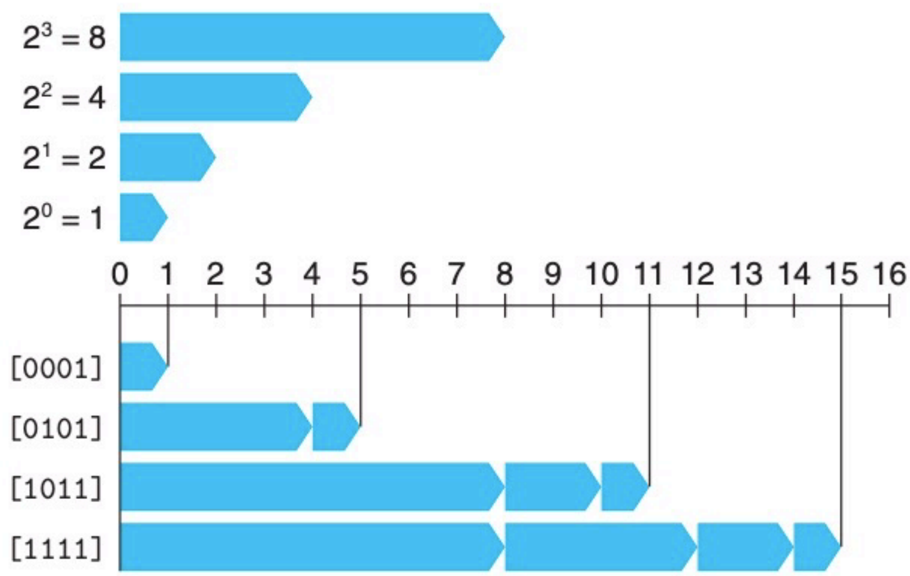
Figure 2.8 **Typical ranges for C integral data types on a 32-bit machine.** Text in square brackets is optional.

Unsigned encoding: $[x_{w-1}, x_{w-2}, \dots, x_0]$

$B2U(\vec{x}) = \sum_{i=0}^{w-1} x_i \cdot 2^i$. map from binary repr to non-negative integers.

(binary to unsigned)

Figure 2.11
Unsigned number examples for $w = 4$.
When bit i in the binary representation has value 1, it contributes 2^i to the value.



Formally,

$B2U_w : \{0,1\}^w \longrightarrow \{0, \dots, 2^w - 1\}$.

bijection

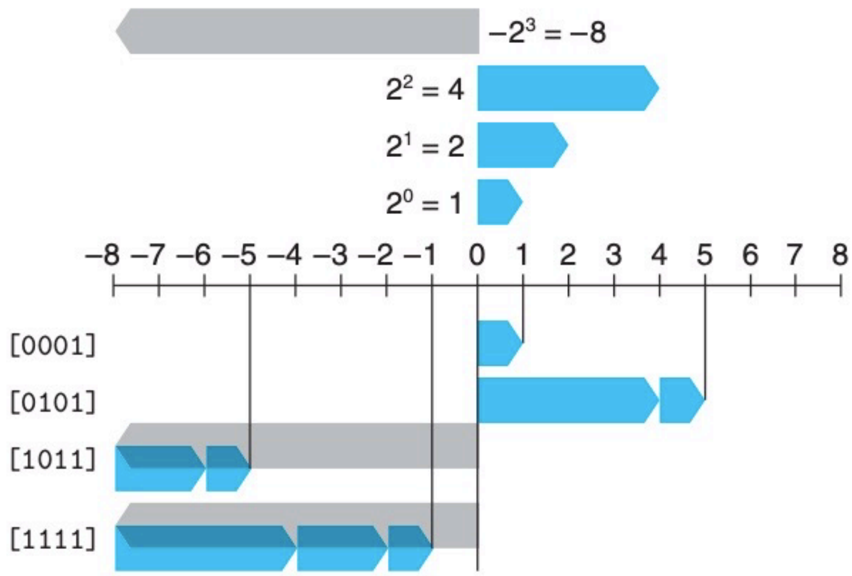
$w = 8$

$\{0,1\}^8 \longrightarrow \{0, \dots, 255\}$

Two's complement encoding:

$$B2T_w(\vec{x}) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i .$$

Figure 2.12 **Two's-complement number examples for $w = 4$.** Bit 3 serves as a sign bit, and so, when set to 1, it contributes $-2^3 = -8$ to the value. This weighting is shown as a leftward-pointing gray bar.



$w=8$ $\qquad \underbrace{\{-128, \cdots, 127\}}$.

Formally,

$$B2T_w : \{0,1\}^w \longrightarrow \{-2^{w-1}, \cdots, 0, \cdots, 2^{w-1}-1\}$$

Examples:

$$\begin{aligned}
B2T_4([0001]) &= -0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 &= 0+0+0+1 &= 1 \\
B2T_4([0101]) &= -0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 &= 0+4+0+1 &= 5 \\
B2T_4([1011]) &= -1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 &= -8+0+2+1 &= -5 \\
B2T_4([1111]) &= -1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 &= -8+4+2+1 &= -1
\end{aligned}$$

	Word size w			
Value	8	16	32	64
$UMax_w$	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	255	65,535	4,294,967,295	18,446,744,073,709,551,615
$TMin_w$	0x80	0x8000	0x80000000	0x8000000000000000
	−128	−32,768	−2,147,483,648	−9,223,372,036,854,775,808
$TMax_w$	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	127	32,767	2,147,483,647	9,223,372,036,854,775,807
−1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	0x00	0x0000	0x00000000	0x0000000000000000

Figure 2.13 **Important numbers.** Both numeric values and hexadecimal representations are shown.

There are two other standard representations for signed numbers:

Ones' Complement: This is the same as two's complement, except that the most significant bit has weight $-(2^{w-1} - 1)$ rather than -2^{w-1}:

$$B2O_w(\vec{x}) \doteq -x_{w-1}(2^{w-1} - 1) + \sum_{i=0}^{w-2} x_i 2^i$$

Sign-Magnitude: The most significant bit is a sign bit that determines whether the remaining bits should be given negative or positive weight:

$$B2S_w(\vec{x}) \doteq (-1)^{x_{w-1}} \cdot \left(\sum_{i=0}^{w-2} x_i 2^i \right)$$