CSC 411

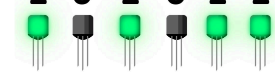
Computer Organization (Spring 2024)
Lecture 3: Bitwise Operations

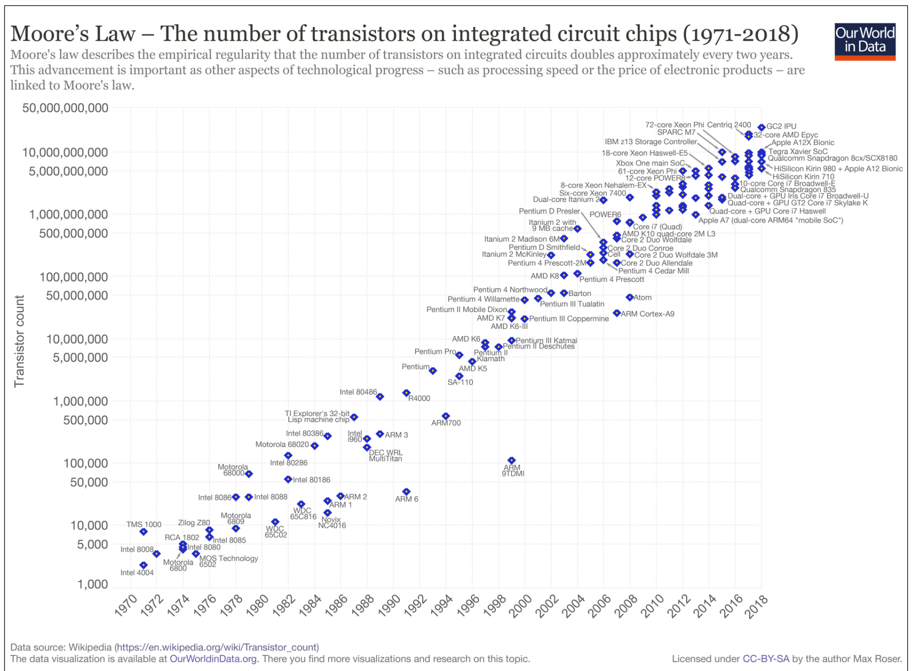
Prof. Marco Alvarez, University of Rhode Island

Bits, bytes, and computers

- A bit is the smallest unit of data in computing, representing a binary digit
  - can have a value of 0 or 1, easy to embed in digital devices
- Binary System
  - computers use the binary system to represent data
  - bits are the building blocks, forming the foundation for all digital information
- Bit Representation
  - bits are often represented by electrical voltages, where high voltage corresponds to 1 and low voltage to 0
  - this binary representation is the basis for encoding information in digital devices

1 0 1 0 1 1





Bits, bytes, and computers

- A group of 8 bits is called a byte
  - bytes are commonly used to represent characters, numbers, and other data in computer systems

1	0	0	0	1	0	0	1
MSB							LSB
most significant bit							least significant bit

- How many different values can be stored in 1 byte?
- How many different values can be stored in n bits?

Typical data types in C

The C language does not mandate this, it depends on the compiler used

More in this later …

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

Boolean algebra

- Developed by George Boole in the 19th century
  - branch of mathematics that deals with binary variables and logic operations
  - binary values represent logical states true or false
- Logic operations
  - defines three fundamental logic operations:
    - AND: output is 1 only if both inputs are 1 — conjunction
    - OR: output is 1 if at least one input is 1 — disjunction
    - NOT output is the opposite of the input — negation
- Boolean expressions
  - formed by combining variables and logic operations

Logical operation	Operator	Notation	Alternative notations	Definition
Conjunction	**AND**	$x \wedge y$	x AND y, Kxy	$x \wedge y = 1$ if $x = y = 1$, $x \wedge y = 0$ otherwise
Disjunction	**OR**	$x \vee y$	x OR y, Axy	$x \vee y = 0$ if $x = y = 0$, $x \vee y = 1$ otherwise
Negation	**NOT**	$\neg x$	NOT x, Nx, $\bar{x}$, x', $!x$	$\neg x = 0$ if $x = 1$, $\neg x = 1$ if $x = 0$

x	y	$x \wedge y$	$x \vee y$
0	**0**	0	0
1	**0**	0	1
0	**1**	0	1
1	**1**	1	1

x	$\neg x$
0	1
1	0

Bit vectors

- Bit vectors are just sequences of bits
  - boolean algebra can be extended to operate on bit vectors
- Applications in Computer Science
  - representing sets of elements efficiently
  - implementing data structures
  - performing bitwise manipulation in low-level programming
- Understanding boolean algebra with bit vectors is essential for working with binary data in computer science and digital design

Bitwise operations in C

- Used with “integer” data types
  - long, int, short, chart, unsigned
  - arguments are treated as bit vectors
  - corresponding binary logic operators are applied bitwise to operands
- Bitwise operators are commonly used to manipulate sets and masks

~	bitwise NOT	~a	the bitwise NOT of **a**
&	bitwise AND	a & b	the bitwise AND of **a** and **b**
|	bitwise OR	a | b	the bitwise OR of **a** and **b**
^	bitwise XOR	a ^ b	the bitwise XOR of **a** and **b**
<<	bitwise left shift	a << b	**a** left shifted by **b**
>>	bitwise right shift	a >> b	**a** right shifted by **b**

Bitwise operations in C

bit a	bit b	a & b (a AND b)
0	0	0
0	1	0
1	0	0
1	1	1

bit a	bit b	a | b (a OR b)
0	0	0
0	1	1
1	0	1
1	1	1

bit a	bit b	a ^ b (a XOR b)
0	0	0
0	1	1
1	0	1
1	1	0

~a (NOT a) is trivial

Examples

```
  1 0 1 1 1 0 0 1
& 0 1 1 1 0 1 1 0
-----------------
  0 0 1 1 0 0 0 0
```

```
  1 0 1 1 1 0 0 1
| 0 1 1 1 0 1 1 0
-----------------
  1 1 1 1 1 1 1 1
```

```
  1 0 1 1 1 0 0 1
^ 0 1 1 1 0 1 1 0
-----------------
  1 1 0 0 1 1 1 1
```

```
~ 0 1 1 1 0 1 1 0
-----------------
  1 0 0 0 1 0 0 1
```

Examples

```
~0x102

0xABC & 0x411
```

Examples

```
0xABC | 0x411


0x102030 & 0x00FF00
```

Shift operations

- Left shift (x << y)
  - shifts each bit in x to the left by the number of positions indicated by y
    - throw away y bits on left
  - blank spaces on right are filled up by zeroes

```
1 0 1 1 1 0 0 1   << 2
-----------------
1 1 1 0 0 1 0 0
```

Shift operations

- Right shift (x >> y)
  - shifts each bit in x to the right by the number of positions indicated by y
    - throw away y bits on right
  - logical shift — fill blank spaces on left with zeroes
  - arithmetic shift — fill blank spaces by replicating original MSB on left (most compilers implement this)

```
                      MSB
1 0 1 1 1 0 0 1 >> 2  1 0 1 1 1 0 0 1 >> 2
----------------      ----------------
0 0 1 0 1 1 1 0       1 1 1 0 1 1 1 0
    logical              arithmetic
```

Examples

```
0xF3 << 2


0x9A >> 3    (logical)


0x9A >> 3    (arithmetic)
```

Example: bit masking

- Assume an integer j that stores the value 0x1A35B127
  - define a mask to extract the most significant byte
  - write C code to store the extracted value in another variable (unsigned int)

Example: bit masking

- Assume an integer j that stores the value 0x1A35B127
  - write C code to set the least significant byte of j to all ones leaving all other bytes unchanged