Low-Level Programming (1)

Program Design (II)

2022 Spring

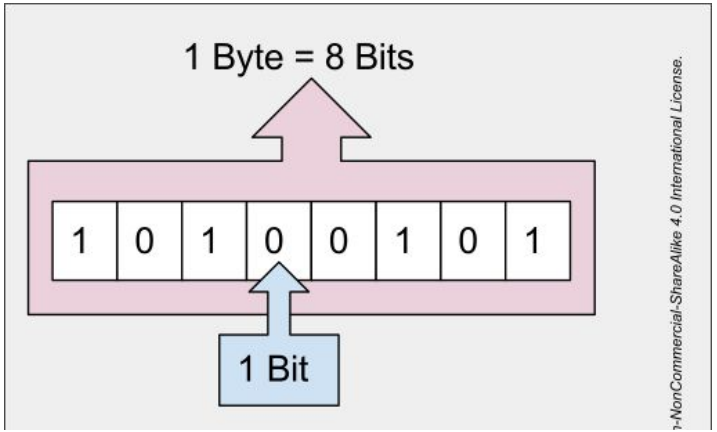
Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

Outline

- Introduction of Low-Level Programming
- Bitwise Operators

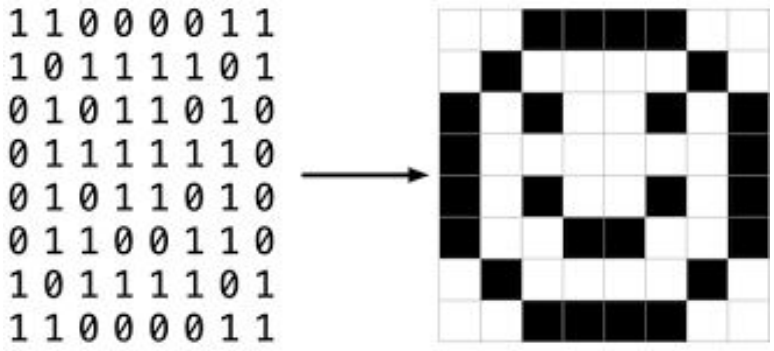
Introduction

- Previous chapters have described C's high-level, machine-independent features.
- However, some kinds of programs need to perform operations at the **bit** level
- What is **bit**?
  - **smallest unit** of **storage** in computer
  - https://web.stanford.edu/class/cs101/bits-bytes.html

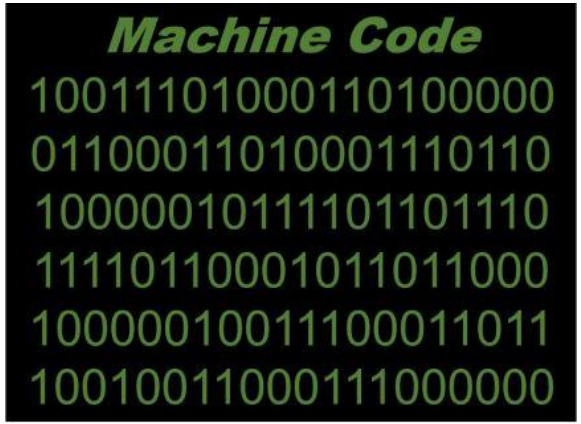


Introduction

- For example, …
- Systems programs (including compilers and operating systems)
- Encryption programs
- Graphics programs
- Programs for which **fast** execution and/or efficient use of space is critical

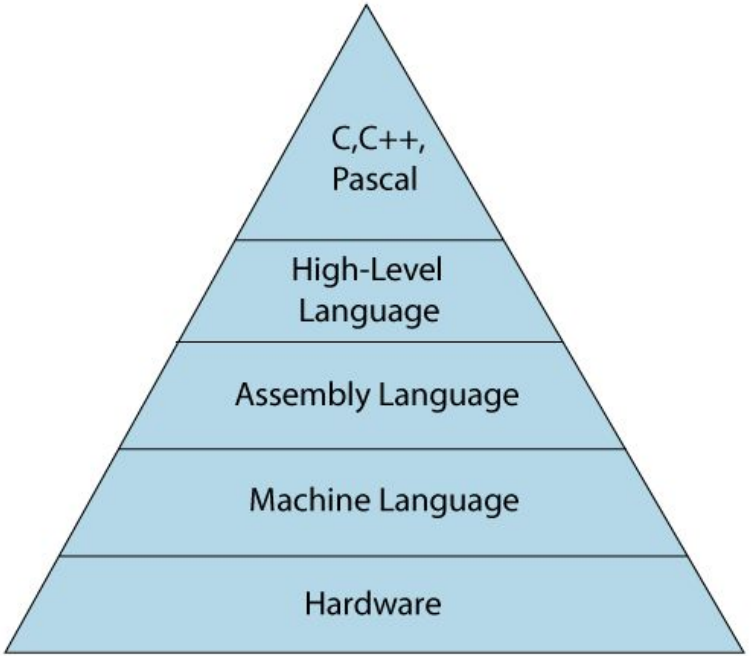


https://cs50.harvard.edu/college/2019/fall/psets/4/filter/less/



Introduction

- C is a high-level programming language, but it included some degree of access to low-level programming functions.
- Low-level language (e.g., machine-level language) consists of a set of instructions that are in the binary form 0 or 1.
- Therefore, we need to know how to use C to operate on bits (i.e., 0 and 1).



https://www.javatpoint.com/classification-of-programming-languages

Bitwise Operators

- C provides six ***bitwise operators*****,** which operate on **integer** data at the bit level.
- Two of these operators perform **shift** operations.
- The other four perform
  - bitwise complement
  - bitwise *and*
  - bitwise exclusive *or*
  - bitwise inclusive *or* operations.
- Let’s explore them one by one!

Bitwise Shift Operators

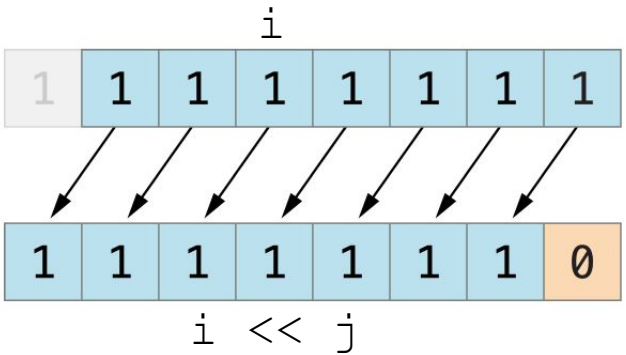
- The bitwise shift operators shift the bits in an integer to the left or right:

  << left shift
  >> right shift

- The operands for << and >> may be of any **integer** type (**including** `char`).
  - `char` in C is represented by integer value
- The integer promotions are performed on both operands; the result has the type of the left operand after promotion.

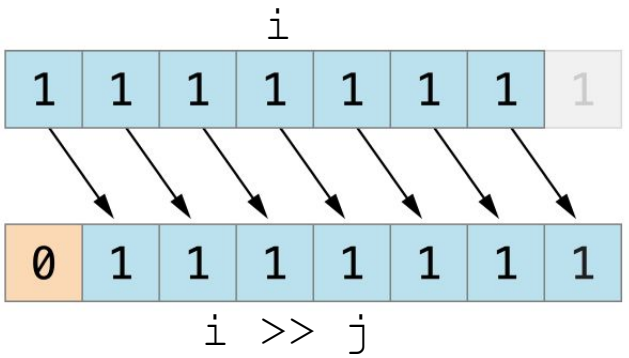
Bitwise Shift Operators

- The value of `i << j` is the result when the bits in `i` are shifted left by `j` places.
- For each bit that is “shifted off” the left end of `i`, a zero bit enters at the right.
- For example, when `j` is `1`



Bitwise Shift Operators

- The value of `i >> j` is the result when `i` is shifted right by `j` places.
- If `i` is of an **unsigned type** or if the value of i is **nonnegative**, zeros are added at the left as needed.
  - The **leftmost** bit of a signed integer (known as the **sign bit**) is **0** if the number is **positive** or zero, **1** if it's **negative**.



Bitwise Shift Operators

- The value of `i >> j` is the result when `i` is shifted right by `j` places.
- If `i` is of an **unsigned type** or if the value of i is **nonnegative**, zeros are added at the left as needed.
  - The **leftmost** bit of a signed integer (known as the **sign bit**) is **0** if the number is **positive** or zero, **1** if it's **negative**.
- If `i` is **negative**, the result is **implementation-defined**.
  - Different machine or complier will have different results
- It's best to perform shifts only on **unsiged** numbers

Bitwise Shift Operators

- Examples illustrating the effect of applying the shift operators to the number 13
- In this chapter, we will use short integers (16 bits) for simplicity.

```
unsigned short i, j;

i = 13;      /* i is now 13 (binary                    ) */
```

slido

What is 16-bits binary for 13?

ⓘ Start presenting to display the poll results on this slide.

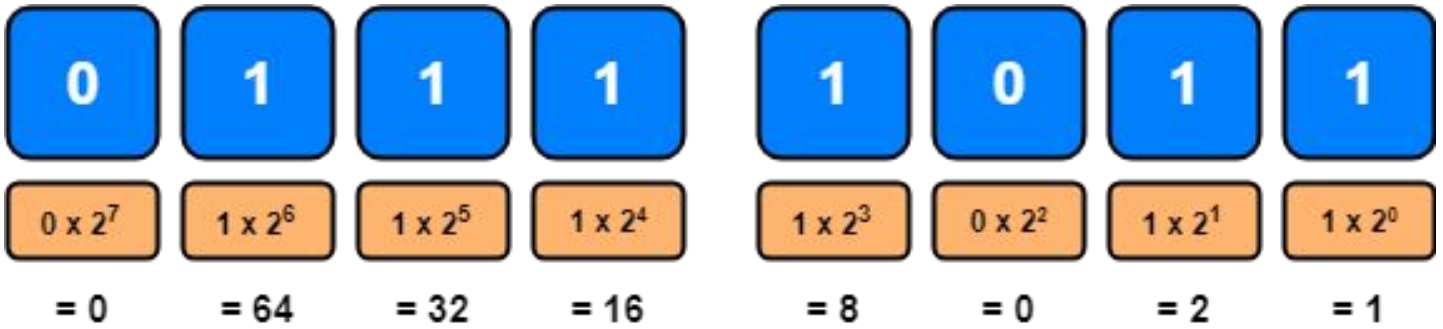
Bitwise Shift Operators

- Examples illustrating the effect of applying the shift operators to the number 13
- In this chapter, we will use short integers (16 bits) for simplicity.

```
unsigned short i, j;

i = 13;      /* i is now 13 (binary                ) */
j = i << 2; /* j is now  ? (binary        ?       ) */
j = i >> 2; /* j is now    (binary                ) */
```

Binary Computation



https://emre.me/computer-science/binary-computation-and-bitwise-operators/

Bitwise Shift Operators

- The bitwise shift operators have lower precedence than the arithmetic operators
- For example, `i << 2 + 1` means `i << (2 + 1)`, not `(i << 2) + 1`
- Use parentheses as much as possible to reduce confusion!

Bitwise Operators

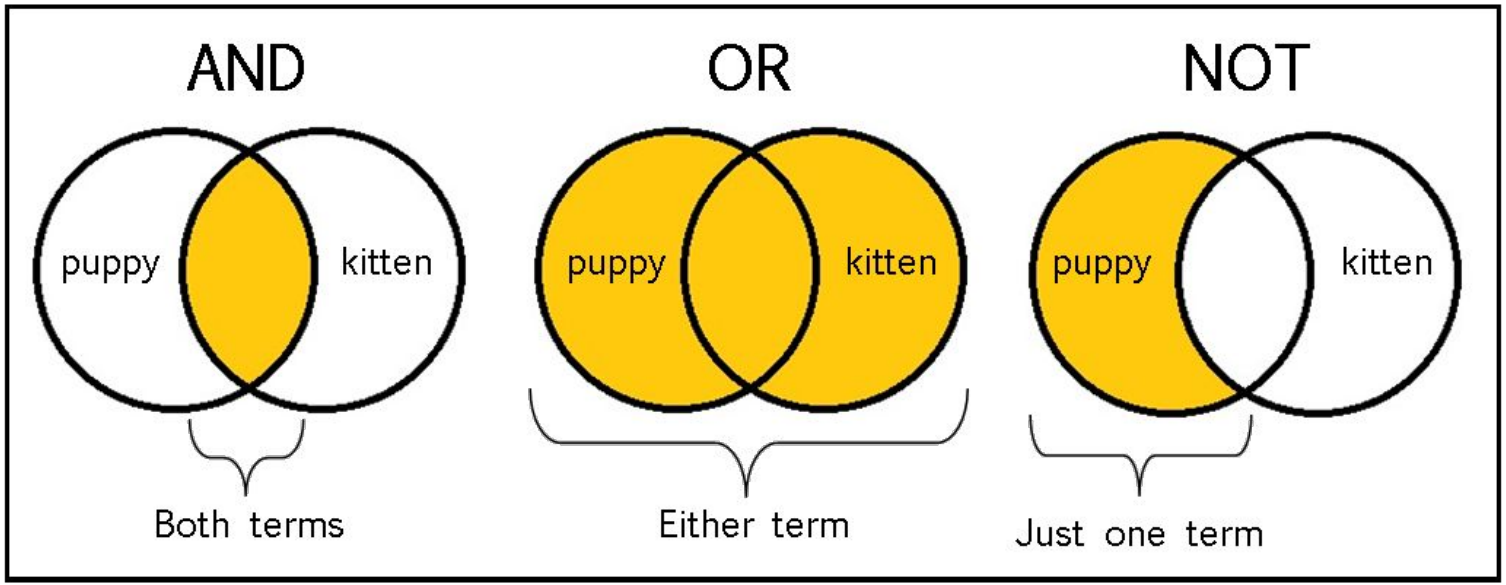
- Two of these operators perform **shift** operations.
  - << left shift
  - >> right shift
- **The other four perform**
  - bitwise complement
  - bitwise *and*
  - bitwise exclusive *or*
  - bitwise inclusive *or* operations.

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

Symbol	**Meaning**	**Example**	**Result of Example**
~	bitwise complement		
&	bitwise *and*		
^	bitwise exclusive *or*		
|	bitwise inclusive *or*		

- The ~ operator is unary
- The other operators are binary
- The `~`, `&`, `^`, and `|` operators perform **Boolean operations** on all bits in their operands.

Quick Review of Boolean operations



Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

Symbol	Meaning	Example	Result of Example
~	bitwise complement	`~(binary 001)`	`binary 110`
&	bitwise *and*		
^	bitwise exclusive *or*		
|	bitwise inclusive *or*		

- The ~ operator produces the complement of its operand (perform ***NOT*** on each bit)
- 0 replaced by 1 and 1 replaced by 0
- For example, ...

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

Symbol	**Meaning**	**Example**	**Result of Example**
~	bitwise complement	`~(001)`	`110`
&	bitwise *and*	`(011) & (110)`	`010`
^	bitwise exclusive *or*		
|	bitwise inclusive *or*		

- The `&` operator performs a Boolean ***AND*** operation on all corresponding bits in its two operands
- For example, ...

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

Symbol	Meaning	Example	Result of Example
~	bitwise complement	`~(001)`	`110`
&	bitwise *and*	`(011) & (110)`	`010`
^	bitwise exclusive *or*		
|	bitwise inclusive *or*		

- The ^ and | operators are similar.
- ^ bitwise exclusive *or:* ***XOR***
- | bitwise inclusive *or:* ***OR***

And, Exclusive Or, and Inclusive Or

AND

Inputs		Output
A	*B*	*Z*
0	0	0
0	1	0
1	0	0
1	1	1

Inclusive OR

Inputs		Output
A	*B*	*Z*
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive OR

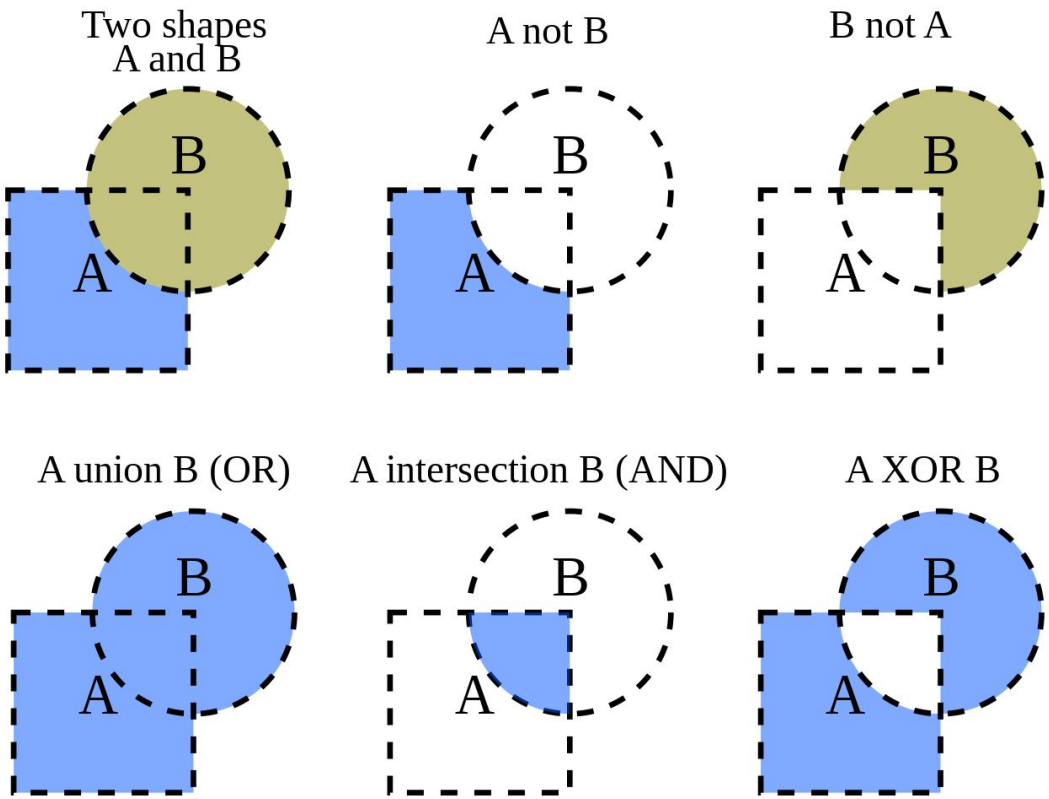
Inputs		Output
A	*B*	*Z*
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

Symbol	Meaning	Example	Result of Example
~	bitwise complement	`~(001)`	`110`
&	bitwise *and*	`(011) & (110)`	`0`**`1`**`0`
^	bitwise exclusive *or*	`(011) ^ (110)`	`101`
|	bitwise inclusive *or*	`(011) | (110)`	`111`

- So the examples of using ^ and | are …

Boolean operations on polygons



https://en.wikipedia.org/wiki/Boolean_operations_on_polygons#/media/File:Boolean_operations_on_shapes-en.svg

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

```
unsigned short i, j, k;
i = 21;      /* i is now    21 (binary 0000000000010101) */
j = 56;      /* j is now    56 (binary 0000000000111000) */
k = ~i;      /* k is now 65514 (binary 1111111111101010) */
```

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

```
unsigned short i, j, k;
i = 21;      /* i is now    21 (binary 0000000000010101) */
j = 56;      /* j is now    56 (binary 0000000000111000) */
k = i & j;   /* k is now    16 (binary 0000000000010000) */
```

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

```
unsigned short i, j, k;
i = 21;      /* i is now    21 (binary 0000000000010101) */
j = 56;      /* j is now    56 (binary 0000000000111000) */
k = i | j;   /* k is now    61 (binary 0000000000111101) */
```

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

```
unsigned short i, j, k;
i = 21;      /* i is now    21 (binary 0000000000010101) */
j = 56;      /* j is now    56 (binary 0000000000111000) */
k = i ^ j;   /* k is now    45 (binary 0000000000101101) */
```

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

- The ~ operator can be used to help make low-level programs more portable.
  - An integer whose bits are all 1: `~0`
  - An integer whose bits are all 1 except for the **last five**: `~0x1f`
- `0x1f` is a hexadecimal number by telling from `0x`
  - **Hexadecimal** constants contain digits between 0 and 9 and letters between a and f, and **always begin with 0x**: 0xf 0xff 0x7fff
- What is the decimal value of `0x1f?`

slido

What is the decimal value of 0x1f?

ⓘ Start presenting to display the poll results on this slide.

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

- The ~ operator can be used to help make low-level programs more portable.
  - An integer whose bits are all 1: `~0`
  - An integer whose bits are all 1 except for the **last five**: `~0x1f`
- `0x1f` in a 16-bits binary is `0000000000011111`
- So, `~0x1f` is `0000000000011111` in 16-bits binary
  - An integer whose bits are all 1 except for the **last five**

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

- Each of the `~`, `&`, `^`, and `|` operators has a different precedence:
  Highest to Lowest: `~  &  ^  |`
- Examples:
  `i & ~j | k` means `(i & (~j)) | k`
  `i ^ j & ~k` means `i ^ (j & (~k))`
- Using parentheses helps avoid confusion.

Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*

- The compound assignment operators `&=`, `^=`, and `|=` correspond to the bitwise operators `&`, `^`, and `|`:

```
unsigned short i, j;

i = 21;   /* i is now 21 (binary                    ) */
j = 56;   /* j is now 56 (binary                    ) */
i &= j;   /* i is now    (binary                    ) */
i ^= j;   /* i is now    (binary                    ) */
i |= j;   /* i is now    (binary                    ) */
```

Summary

- Introduction of Low-Level Programming
- Bitwise Operators
  - Bitwise Shift Operators
  - Bitwise Complement, *And*, Exclusive *Or*, and Inclusive *Or*
  - Binary and Hexadecimal Computation