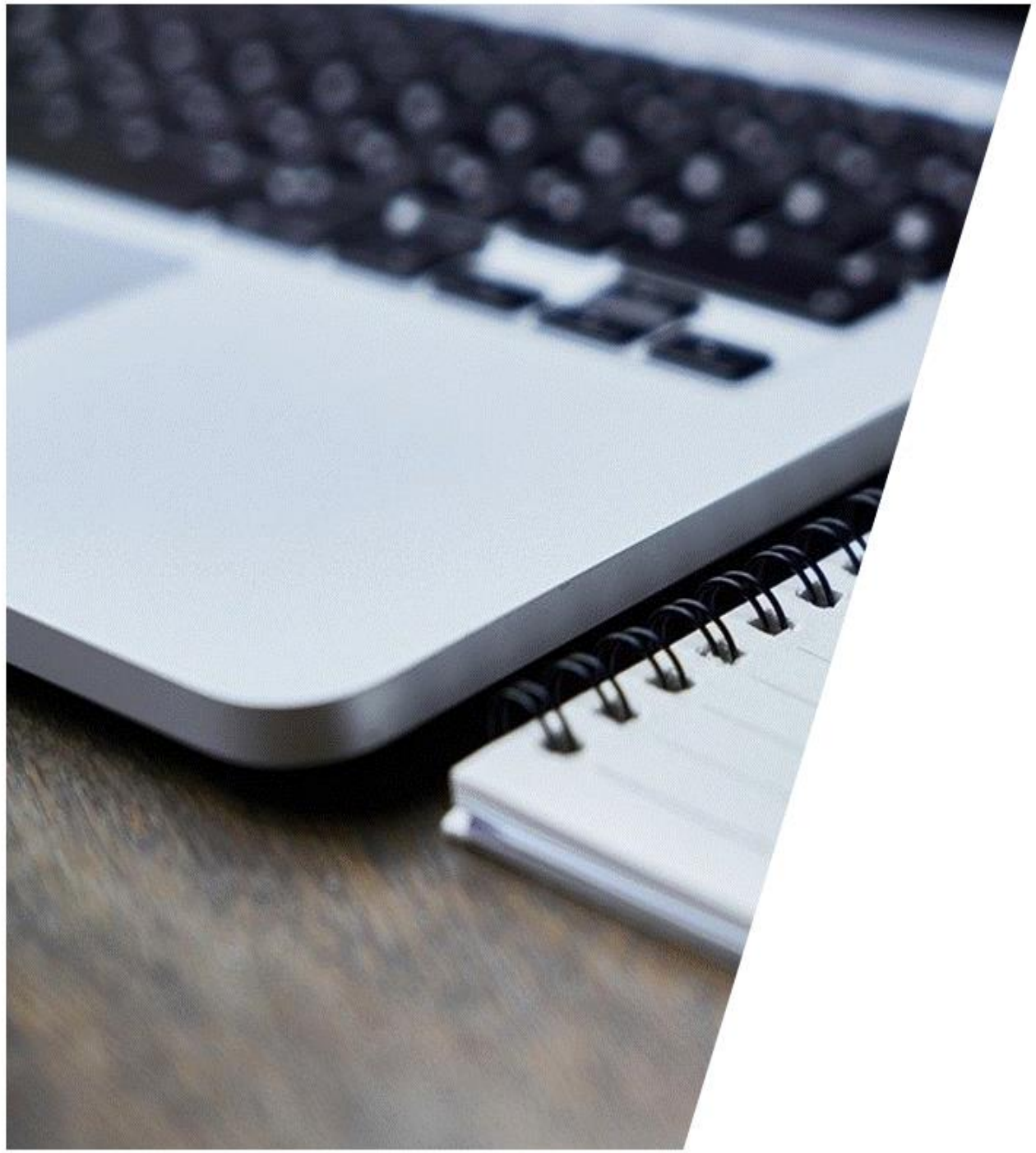


C 語言

位元運算子

背景知識

構成資料的基礎是位元(bit)

- 所有的資料在電腦的內部均是以一連串的位元(bit)來加以表示，每一個位元(bit)的值可為0或1
- 在大部分的系統裡，8個位元會構成一個位元組(byte)

延伸閱讀：來點科普

看一下英文字元與二進位的對照-ASCII

- 在大部分的系統裡，8個位元會構成一個位元組(byte)
- 1個byte是字元型別(char)變數的標準儲存單位，右圖是ASCII定義的文字資料的編碼表
  - 以文字A來說，A的二進位碼是01000001，以資料量來說是1個byte，也就是8個bit
- 其他的資料型態(整數或浮點數)則會存放較多的位元組

ASCII可顯示字元（共95個）

二進位	十進位	十六進位	圖形
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U

以上截圖自(https://zh.wikipedia.org/wiki/ASCII)

電腦儲存資料的單位：位元(bit)與位元組(byte)

- 位元：binary digit，簡稱bit
  - 電腦記憶體儲存的最小單位
  - 一個位元只能表達兩種狀態：0 與 1

- 位元組：byte
  - 由八個 bit 所組成，1 byte = 8bits
  - 1 byte 可以表達 2^8 = 256 種狀態

電腦內部如何儲存人類慣用的十進位的數字資料

- 所有十進位資料都必須轉換為二進位格式才能在電腦中儲存。
- 下圖是十進位與二進位的對照表，二進位是逢2進位的計算結果
- 以十進位的16為例，10000是其2進位的表示結果

十進位	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
二進位	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000

其他進位制的介紹

- 記數系統，表示數值的方式

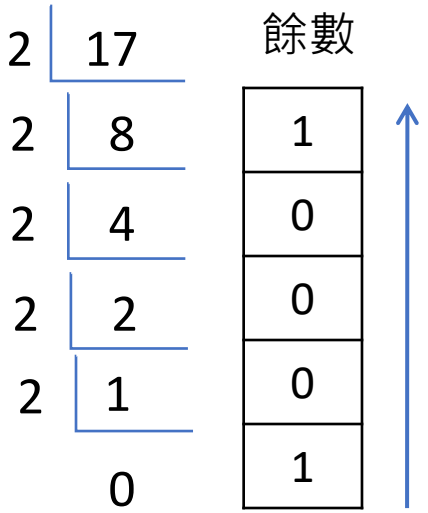
	底/基數	數字範圍	表示法
十進位 (最常使用的進制系統)	10	0 1 2 3 4 5 6 7 8 9	代表十進位數的 11：11_{10}
二進位 (電腦內部資料儲存方式)	2	0 1	代表二進位數的 11：11_{2}
八進位	8	0 1 2 3 4 5 6 7	代表八進位數的 11：11_{8}
十六進位	16	0 1 2 3 4 5 6 7 8 9 A B C D E	代表十六進位數的 11：11_{16}

進位制對照表

十進位	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
二進位	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
八進位	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20
十六進位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

進位制轉換的計算方法

將十進位制 17 轉成二進位制



$17_{10} = 10001_2$

將二進位制 101011 轉成十進位制

1	0	1	0	1	1

$101011_2 = 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

$= 32 + 8 + 2 + 1$

$= 43_{10}$

除2進位之外，為何特別提及8進位與16進位

- 提供一種方便的方式來表示二進位數字
- 八進位數字系統中，每個數字都可以表示為3位二進位數字，例如，二進位數字101可以表示為八進位數字5；十六進位每個數字都可以表示為4位二進位數字，例如，二進位數字1010可以表示為十六進位數字A。
- 過去，許多計算機硬體都是以8位元組為基礎的，因此，8進位方便操作，目前使用已經大幅減少
- 目前，16進位數字系統經常用於表示記憶體地址和數據。由於一個位元組（八位元）可以表示為兩個十六進位數字，因此十六進位數字常用於描述位元組序列（例如，檔案內容或網路傳輸數據）。此外，十六進位數字還常用於編程和調試，因為它們比二進位數字更易於讀取和比較。

運算子的種類

- 算術運算子：+ - * / %
- 位元運算子：
  - 一個整數資料型態被視為一個操作對象，會將十進位轉為二進位，然後對每一個位元(bit)進行操作
- 邏輯運算子：&& || !(後面章節會介紹)

十進位	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
二進位	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000

位元運算的操作

操作位元運算的方法有6個：位元運算子

運算子	運算	意義	運算過程說明
&	AND	且	對兩個二進位數字進行逐位比較： 只有當兩個位元都為1時結果為1，否則為0。
|	OR	或	對兩個二進位數字進行逐位比較： 只有當兩個位元都為0時結果為0，否則為1。
^	XOR	互斥	對兩個二進位數字進行逐位比較： 只有當兩個位元不同時結果為1，否則為0
~	NOT	反	將一個二進位數字： 每個位元都取反（即0變成1，1變成0）
>>	Right Shift	右移	將一個二進位數字： 向左移動指定的位數，右邊補0
<<	Left Shift	左移	將一個二進位數字： 向右移動指定的位數，左邊補符號位（對於帶符號數字），或者補0（對於無符號數字）

位元運算子 &（AND）

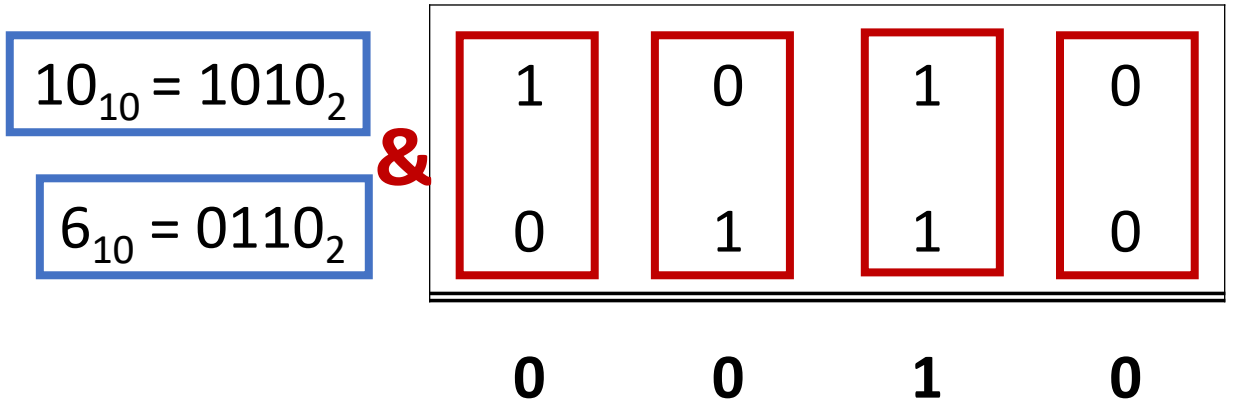
- 任一位元 n 與 0 的 & 運算等於 0。
- 任一位元n 與 1 的 & 運算等於 n。

- 當兩個 bit 都是 1 時，& 運算才會得到
- 利用與 0 的 & 運算可以將某位元設成 0。
- 可用來判斷位元是 1 還是 0 。

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

& (AND)

```c
#include <stdio.h>

int main(){
    int a = 10;
    int b = 6;

    printf("%d & %d = %d", a, b, a&b);
    return 0;
}
```



1010_2 & $0110_2 = 0010_2$

10_{10} & $6_{10} = 2_{10}$

位元運算子 |（OR）

- 任一位元n 與 0 的 | 運算等於 n。
- 任一位元n 與 1 的 | 運算等於 1。

- 當兩個 bit 都是 1 時，& 運算才會得到 1
- 利用與 1 的 | 運算可以將某位元設成 1。

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

| (OR)

```c
#include <stdio.h>

int main(){
    int a = 10;
    int b = 6;

    printf("%d | %d = %d", a, b, a|b);
    return 0;
}
```

$10_{10} = 1010_2$

$6_{10} = 0110_2$

|
1	0	1	0
0	1	1	0
1	**1**	**1**	**0**

1010_2 | $0110_2 = 1110_2$

10_{10} & $6_{10} = 14_{10}$

位元運算子 ^（XOR）

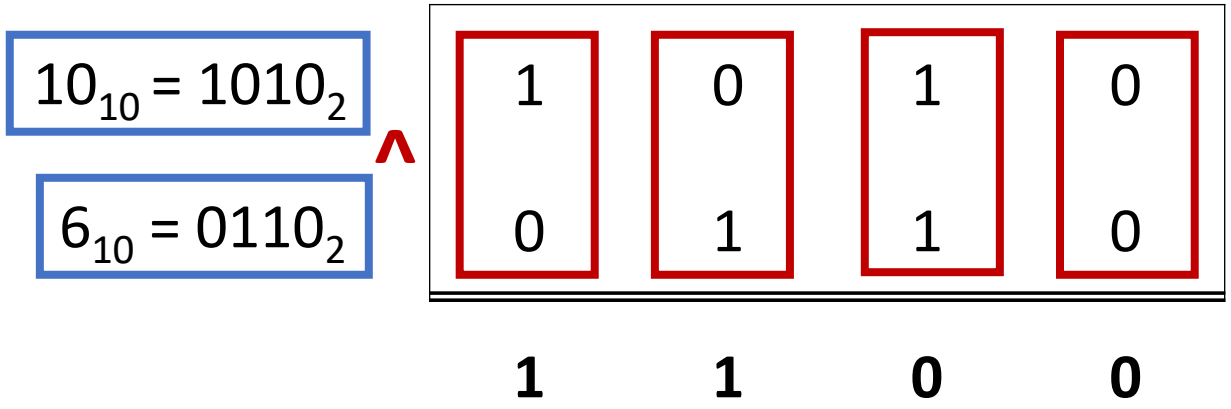
- 任一位元 n 與 0 的 ^ 運算等於 n。
- 任一位元n 與 1 的 ^ 運算等於 not n。

- 當兩個 bit 不一樣時，^ 運算才會得到 1 。
- 利用與 1 的 ^ 運算可以將某位元反相，0⇔1 。
- 可用來判斷兩個位元是否相等。

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

^ (XOR)

```c
#include <stdio.h>

int main(){
    int a = 10;
    int b = 6;

    printf("%d ^ %d = %d", a, b, a^b);
    return 0;
}
```



1010_2 ^ 0110_2 = 1100_2

10_{10} ^ 6_{10} = 12_{10}

位元運算子 ~（NOT）

- 將位元反相，0⇔1 。

a	~a
0	1
1	0

習題範例說明：
由於0的二進位碼所有位元都是0，取反後變為全1，因此結果是-1。而對於1進行NOT運算時，二進位碼為0001，取反後變為1110，這在補碼表示中是-2。

a	~a	結果
0000	1111	-1
0001	1110	-2

位元運算子 << (左移)

- 將變數內的所有位元向左移動 n 個位元，寫法：a << n。

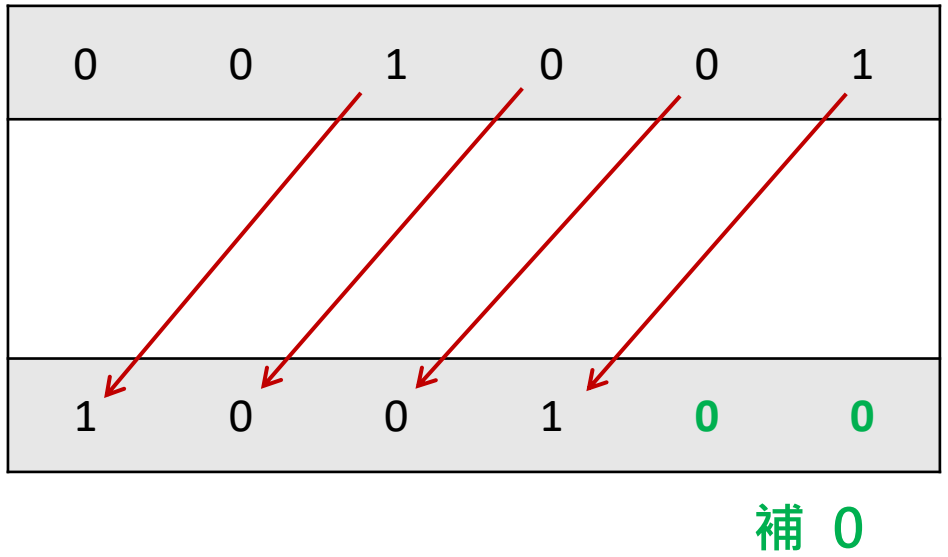
例如：

$9_{10} << 2$

$\Rightarrow 1001_2 << 2$

$\Rightarrow 100100_2$

$\Rightarrow 36_{10}$



位元運算子 << (左移)

- 移動後，若左邊位元超出變數占用的空間範圍時，超出的位元就會消失，低位元會補 0 。

例如：

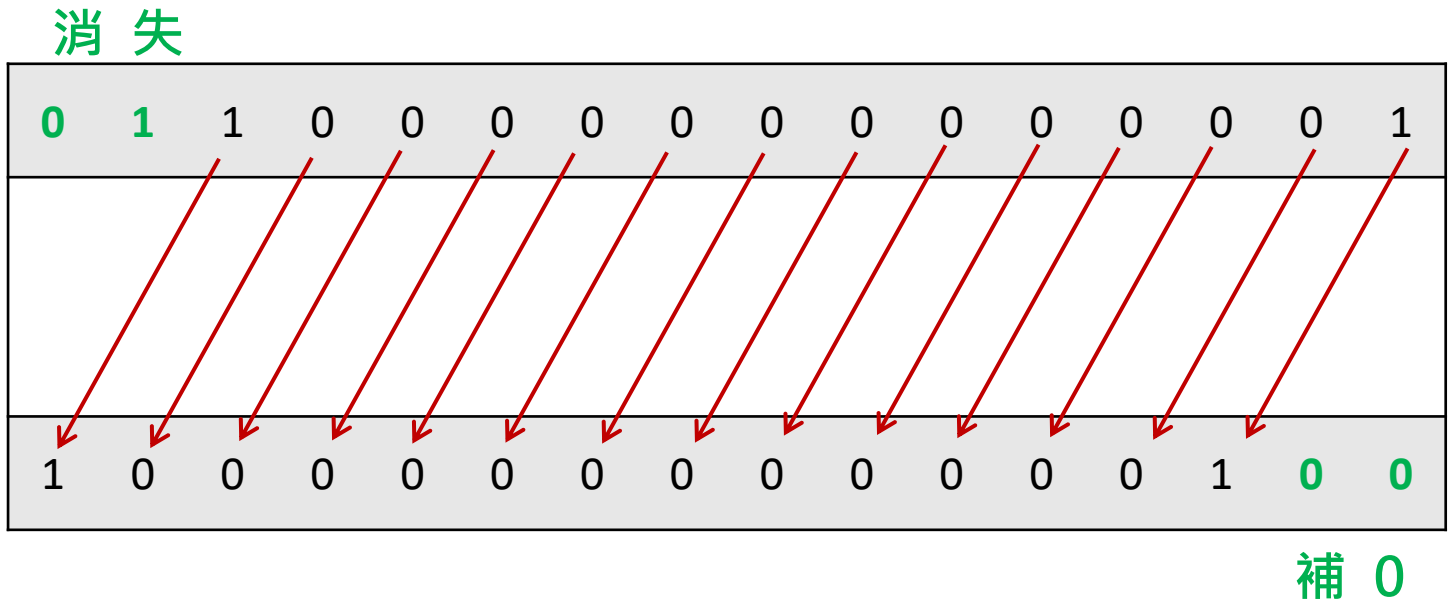
unsigned short a= 24577

$24577_{10} << 2$

$\Rightarrow 1100000000000001_2 << 2$

$\Rightarrow 1000000000000100_2$

$\Rightarrow 32772_{10}$



位元運算子 >> (右移)

- 將變數內的所有位元向右移動 n 個位元，寫法：a >> n。

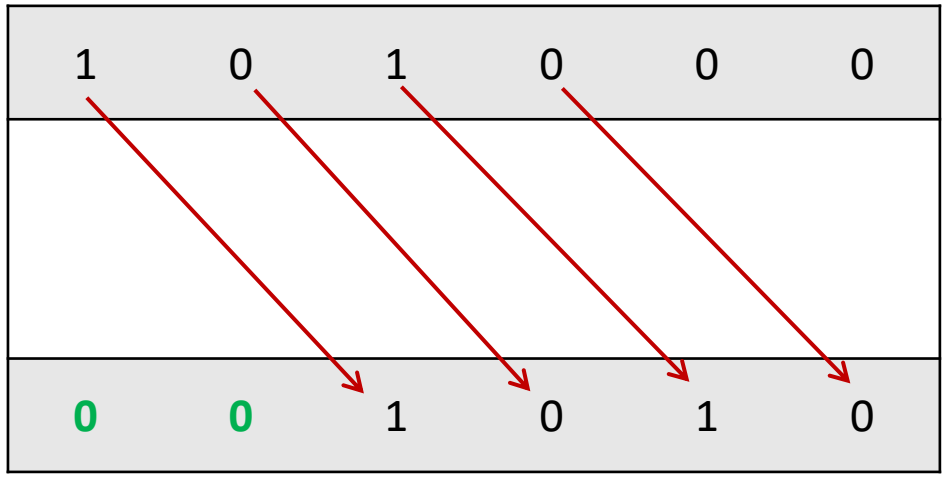
例如：

$40_{10} >> 2$

$\Rightarrow 10100_2 << 2$

$\Rightarrow 1010_2$

$\Rightarrow 10_{10}$



位元運算子 >>（右移）

- 移動後，低位元會消失，高位元會補 0 。

例如：

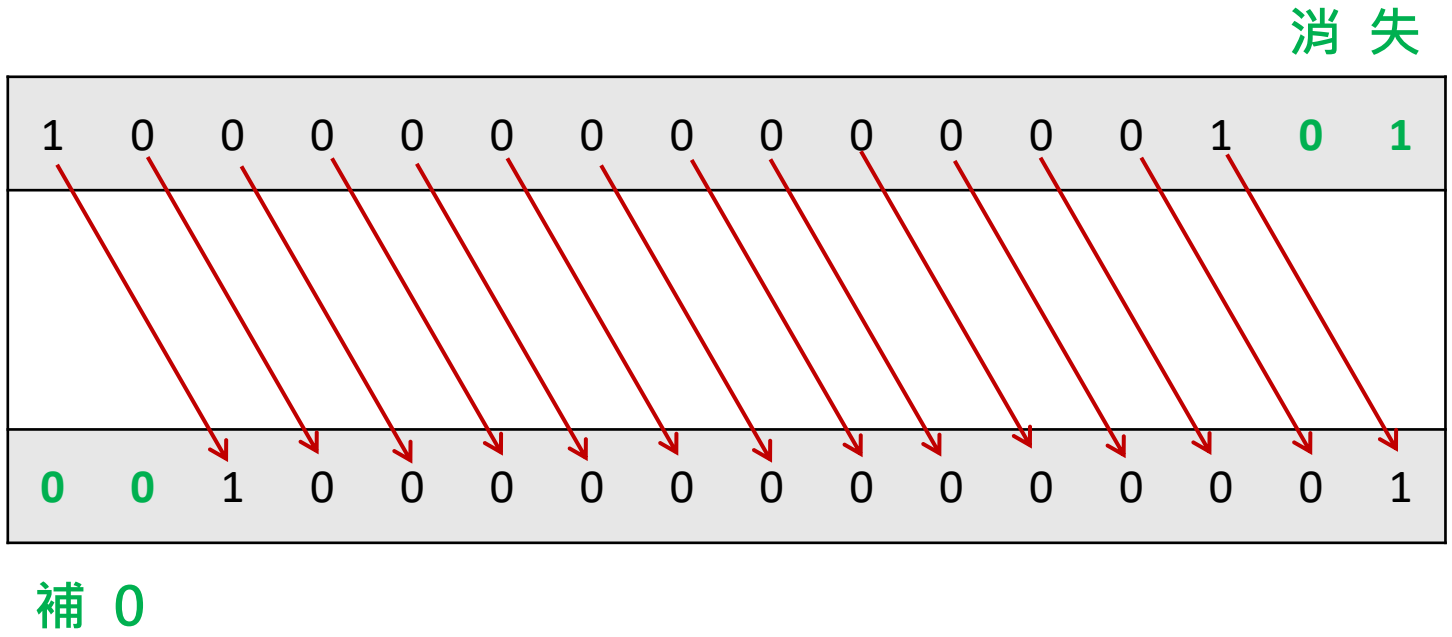
unsigned short a= 32773

$32773_{10} >> 2$

$\Rightarrow 1000000000000101_2 >> 2$

$\Rightarrow 1000000000001_2$

$\Rightarrow 8193_{10}$



註解unsigned short ：

- C語言中的資料類型之一，用於表示一個16位元無符號整數。它的取值範圍是0到65535
- “unsigned”表示這個類型的變量不包含符號，即它只能表示非負數。“short”表示這個類型的變量占用的內存空間比整型（int）變量少，通常為2個字元。

<< (左移) 與 >> (右移) 的練習題

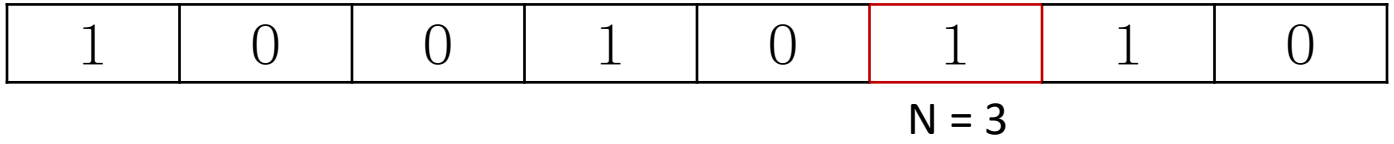
```c
#include <stdio.h>

int main(){
    int a = 10;
    int b = 20;

    printf("%d >> 2 = %d\n", a, a >> 2); // 10 >> 2 = 2
    printf("%d << 2 = %d\n", a, a << 2); // 10 << 2 = 40
    printf("%d >> 4 = %d\n", b, b >> 4); // 20 >> 4 = 1
    printf("%d << 4 = %d\n", b, b << 4); // 20 << 4 = 320
}
```

位元運算應用實例

問題1：二進制數值第 N 個位元值？

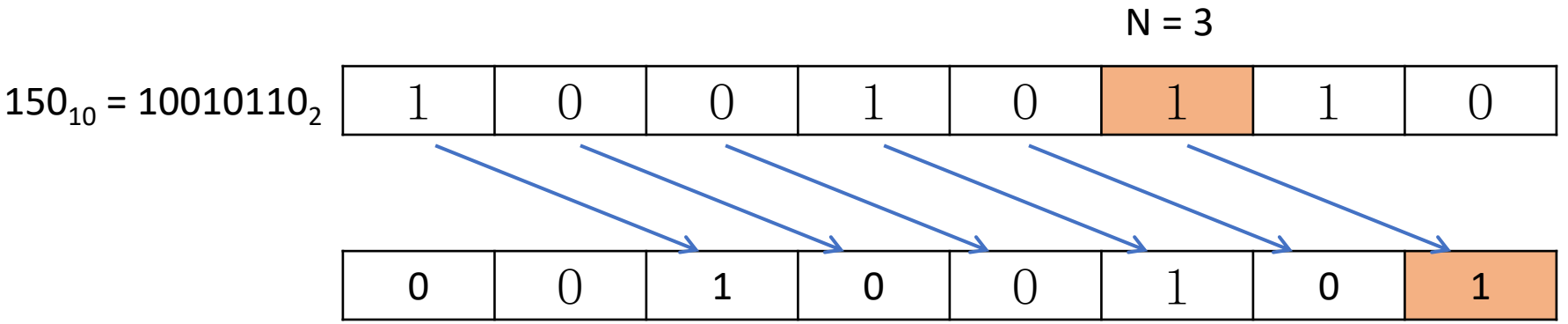
問題：150_{10} 的二進制數值右邊數來第 N 個位元是 1 還是 0？

$150_{10} = 10010110_2$



150_{10}的二進制第 N 個位元值？

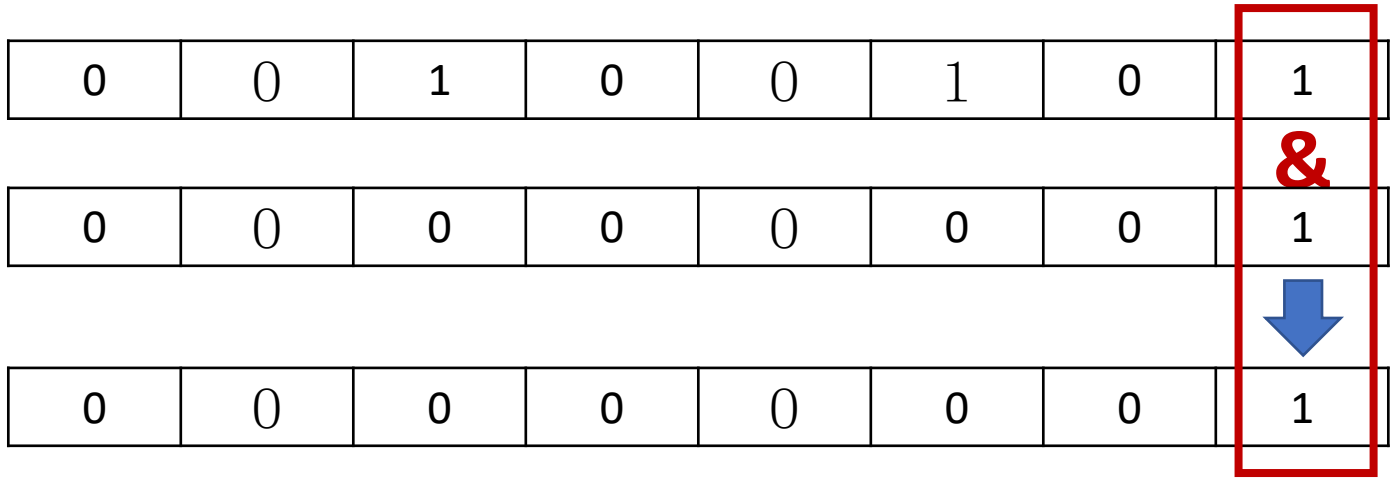
第一步：先右移 2 個位元




```
( a >> (3-1) )
```

150_{10}的二進制第 N 個位元值？

第二步：跟 1 進行 AND 運算，就可判斷位元是 1 還是 0 。




```
( a >> (N-1) ) & 1
```

解決：二進制數值第 N 個位元值？

```c
#include <stdio.h>

int main(){
    unsigned int a = 150;
    int i = 3;
    unsigned int x = ( a >> (i-1) ) & 1;
    // 取出 a 第 i 個位置的 bit 值

    printf("%u\n", x);
    return 0;
}
```

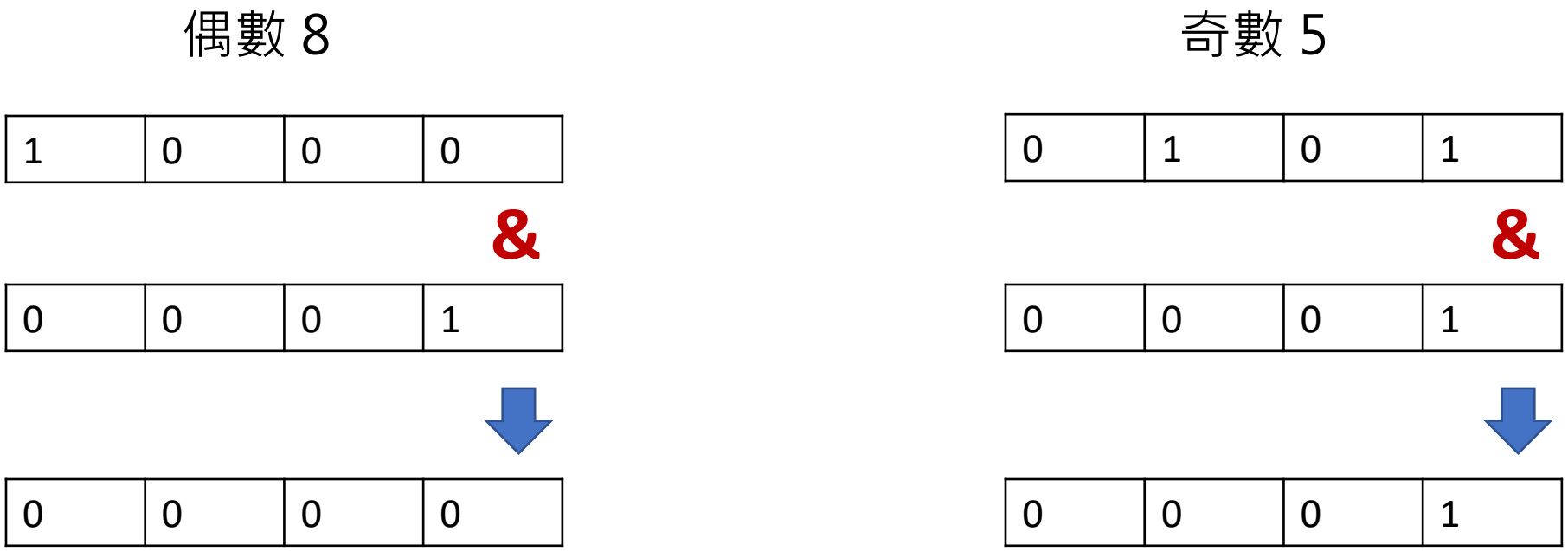
問題2：2 的 N 次方是多少？

2^0	0	0	0	0	0	0	1	1<<0
2^1	0	0	0	0	0	1	0	1<<1
2^2	0	0	0	0	1	0	0	1<<2
2^3	0	0	0	1	0	0	0	1<<3
2^4	0	0	1	0	0	0	0	1<<4

解决：2 的 N 次方是多少？

```c
#include <stdio.h>

int main(){
    int n ;

    scanf("%d", &n);
    printf("2 的 %d 次方 => %d\n", n, 2<<(n-1));
    return 0;
}
```

問題3：整數 N 是偶數嗎？



解決：整數 N 是偶數嗎？

```c
#include <stdio.h>

int main(){
    int n ;
    scanf("%d", &n);
    if (n & 1) {
        printf("%d 是奇數\n", n);
    } else {
        printf("%d 是偶數\n", n);
    }
    return 0;
}
```

位元運算的應用範圍

在計算機科學和電子工程等領域中，位元運算是一個非常重要的技術，位元遮罩、位元清除、位元翻轉、位元交換等等，可以用於許多應用，例如數據壓縮、加密解密、圖形處理、控制系統等。

自主學習

以上只討論正數，
負數在二進位怎麼表示?

對整數資料型態的操作

- 對位元（即0和1）進行運算的過程。
- 電腦以1個byte來放置字元資料型態
- 電腦對於整數資料型態空間的配置又是如何?
  - 除了常用的int，還有short 與 long等資料型別，分別有不同的位元組長度
  - int, short與long都是有符號整數，能表示正負數，也可以特別聲明unsigned(無符號整數)

整數資料型態

- 每個整數變數， 會保留最左邊的一個位元來記錄數值的正負狀態，1 表示負數，0表示正數。
- 指定 unsigned 的整數變數就不需要保留最左邊的一個位元，因此可表示的數值範圍會比較多。
- int / long 等占用的位元組長度會依照執行的系統環境而有差異。

	系統環境	位元組長度	數值範圍
int	16位元系統	2 bytes	-32768 to32767
int	32位元系統	4 bytes	-2,147,483,648 to 2,147,483,647
short		2 bytes	-32,768 to 32,767
long	32位元系統	4 bytes	-2,147,483,648 to 2,147,483,647
long	64位元系統	8 bytes	$-(2^{63})$ to $(2^{63})-1$

以unsigned short int來說明無正負符號的整數

- 宣告方式 unsigned short int x;
- unsigned short int 以16個位元來表示正整數

二進位	十進位
0000000000000000	0
0000000000000001	1
0000000000000010	2
0000000000000011	3
0000000000000100	4
…	…
1111111111111110	65534
1111111111111111	65535

以short int來說明無正負符號的整數

- 宣告方式 unsigned short int x;
- unsigned short int 以16個位元來表示正整數

二進位	十進位
0000000000000000	0
0000000000000001	1
0000000000000010	2
0000000000000011	3
0000000000000100	4
…	…
1111111111111110	65534
1111111111111111	65535

二進位該如何表示一負數?

最簡單的辦法-
數字的最左邊留用一個數字來表示正負數

- 因為以上的原則，演化出三種負數表示法
  - 符號表示法
  - 1’s 補數法
  - 2’s 補數法
- 現在的電腦全都是採用「2’s 補數法」來表示負數

以short int來說明有正負符號的整數

- 宣告方式 short int x;
- short int 的最左位元表示

二進位	十進位	二進位	十進位
0000000000000000	0	1000000000000000	-0
0000000000000001	1	1000000000000001	-1
0000000000000010	2	1000000000000010	-2
0000000000000011	3	1000000000000011	-3
0000000000000100	4	1000000000000100	-4
…	…	…	…
0111111111111110	32766	1111111111111110	-32766
0111111111111111	32767	1111111111111111	-32767

這樣表示的話，會發生什麼樣的問題呢？
「0」不就有「+ 0」和「- 0」 兩種表示了嗎？如果程式設計人員稍微粗心一點，就很容易發生錯誤
要解決這問題有個辦法，就是取「補數」。

負數表示法的發展-
1’s 補數法

- 以4個bit為例
- 有號1補數表示法在表示正數時，跟基本正數的二進位表示法一樣。
- 但在表示負數時，就是把基本正數二進位表示法整個取1補數，記得連第一位元的0都要變1。

1的補數， 0 如果是「0000」、取補數 -0 就會是「1111」，還是沒有解決 0 有 +0 和 -0 兩種結果的問題
為了改善，所以我們進一步再將數字加上 1 做修正。
這種對「1’ s 補數法」的進一步修正，我們稱為「**2’ s 補數法**」
這樣要表示 0 的話，就只有 000 一種寫法了；負數會從 -1 開始計算。

十進位	1的補數
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

負數表示法的發展-2's 補數法

- 以4個bit為例
- 為了改善，所以我們進一步再將數字加上 1 做修正。這種對「1' s 補數法」的進一步修正，我們稱為「2' s 補數法」

十進位	1的補數	2的補數
7	0111	0111
6	0110	0110
5	0101	0101
4	0100	0100
3	0011	0011
2	0010	0010
1	0001	0001
0	0000	0000
-0	1111	0000
-1	1110	1111
-2	1101	1101
-3	1100	1101
-4	1011	1100
-5	1010	1011
-6	1001	1010
-7	1000	1001

0

位元運算子 ~（NOT）

```c
#include <stdio.h>
#include <limits.h>

int main(){
    unsigned short int x = USHRT_MAX, y = 0;
    short int a = SHRT_MAX, b = SHRT_MIN;
    printf("NOT(%u) = %u\n", x, (unsigned short int )~x); // NOT(65535) = 0
    printf("NOT(%u) = %u\n", y, (unsigned short int )~y); // NOT(0) = 65535
    printf("NOT(%d) = %d\n", a, (short int)~a); // NOT(32767) = -32768
    printf("NOT(%d) = %d\n", b, (short int)~b); // NOT(-32768) = 32767
    return 0;
}
```