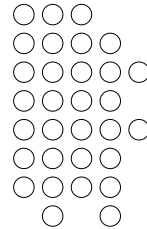


第十五章 位元處理

各種進位系統
位元運算子的使用方法
位元欄位結構的使用



數字系統概述

- 十進位的計數方式是由 0 到 9，遇到 10 即進位
 - 6935 可以用下式表達：

$$6935_{10} = 6 \times 10^3 + 9 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$$

- 一個星期有七天，因此可以看成是七進位系統

表 15.1.1 十進位、七進位與星期系統的對照

十進位	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
七進位	0	1	2	3	4	5	6	10	11	12	13	14	15	16	20	21	22
星期系統	日	一	二	三	四	五	六	日	一	二	三	四	五	六	日	一	二

位元與位元組

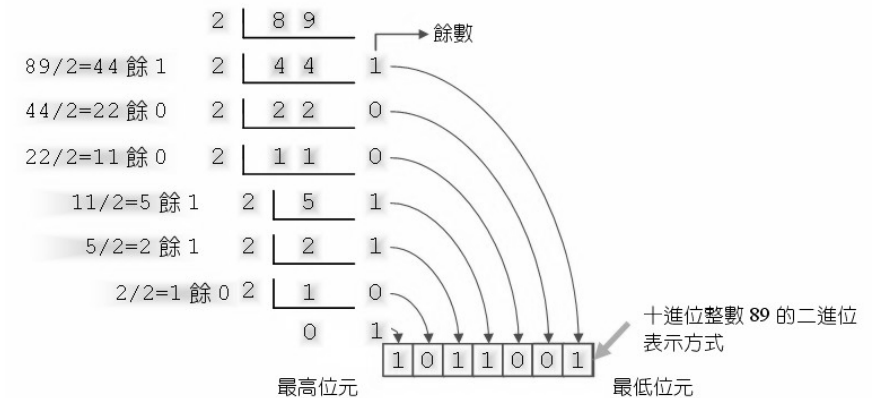
- 二進位以 0 與 1 表示數字，如此可以代表許多資訊：



- 最小的儲存單位是位元 (bit)
- 一個位元組 (bytes) 等於 8 個位元
- 用光、電和磁的訊號在儲存、傳輸和計算時以二進位制表示時容錯能力最高

二進位系統

- 將十進位整數 89 轉換成二進位的計算過程：



$$1011001_2 = 1 \times 2^6 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 89_{10}$$



數字系統轉換的範例 (1/2)

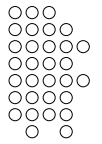
- 下面的show_binary() 可將十進位整數轉換成二進位：

```

01 /* prog15_1, 將十進位整數以二進位來表示 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 #define SIZE 8 /* 定義SIZE為8,代表以8個數字顯示二進位 */
05 void show_binary(int); /* 宣告 show_binary() 函數的原型 */
06 int main(void)
07 {
08     printf("89的二進位為: ");
09     show_binary(89); /* 顯示 89 的二進位*/
10
11     system("pause");
12     return 0;
13 }

```

5



數字系統轉換的範例 (2/2)

```

14 void show_binary(int num) /* 函數 show_binary() 的定義 */
15 {
16     int i,b[SIZE]={0}; /* 宣告陣列b,並設定元素的初值都是0 */
17     for(i=1;i<=SIZE;i++)
18     {
19         b[SIZE-i]=num%2; /* 將 num%2 的餘數設定給 b[SIZE-i] 存放 */
20         num=num/2; /* 將 num/2 的值設回給 num */
21     }
22     for(i=0;i<SIZE;i++)
23         printf("%d",b[i]);
24     printf("\n");
25 }

```

/* prog14_1 OUTPUT--

89的二進位為: 01011001

-----*/

6



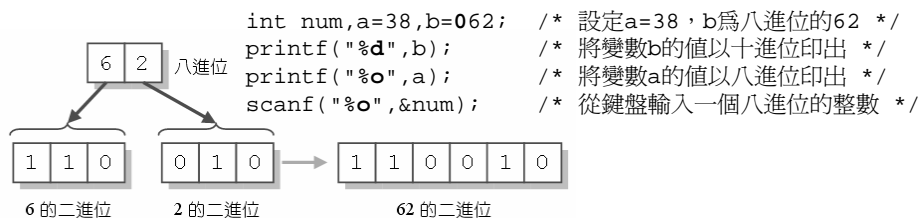
八進位系統

- 八進位數字系統: 以8為基底,由0~7等8個數字所組成

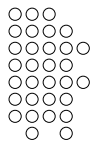
表 15.3.1 八進位和二進位的相對應值

八進位	0	1	2	3	4	5	6	7
二進位	000	001	010	011	100	101	110	111

- 八進位的每一個位數可看成是三個二進位數字的組合：



7



十六進位系統 (1/2)

- 十六進位數字系統就是以 16 為基底的數字系統

十進位	二進位	八進位	十六進位
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

8



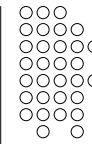
十六進位系統 (2/2)

- 十六進位的5D，可以寫成下面的式子

$$5D_{16} = 5 \times 16^1 + 13 \times 16^0 = 93_{10}$$

- 下面是一些十六進位整數在使用上的範例：

```
int num,a=25,b=0x6A; /* 設定a=25，b的值為十六進位的6A */
printf("%d",b);      /* 將變數b的值以十進位印出 */
printf("%x",a);     /* 將變數a的值以十六進位印出 */
scanf("%x",&num);   /* 從鍵盤輸入一個十六進位的整數 */
```



位元邏輯運算子

- 位元邏輯運算子可用來對每一個位元做邏輯運算
- 下表列出了位元邏輯運算子：

表 15.4.1 位元邏輯運算子

位元邏輯運算子	意義
~	位元 NOT 運算子
&	位元 AND 運算子
	位元 OR 運算子
^	位元 XOR 運算子

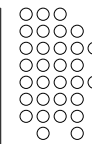


NOT運算子「~」

- 「~」運算子用來將位元顛倒
- 使位元值為 0 者變成 1，位元值 1 者變成 0：

表 15.4.2 NOT 運算子「~」之真值表

a	~a
0	1
1	0



AND運算子「&」

- AND運算子「&」
- 將運算子左、右兩邊運算元裡的每一個位元進行 AND 運算

表 15.4.3 AND 運算子「&」之真值表

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

- 設變數a=105，b=26，則 a&b 的結果為 8

```

01101001 十進位的值為 105
00011010 十進位的值為 26
a&b -----
00001000 十進位的值為 8

```

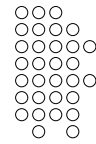
位元AND運算子「&」的範例

• AND運算子「&」的範例

```
01 /* prog14_2, 位元AND運算子「&」的範例說明 */
02 #include <stdio.h>
03 int main(void)
04 {
05     int a=105,b=26;
06     printf("%d&%d=%d\n",a,b,a&b); /* 計算 a&b 的值 */
07
08     system("pause");
09     return 0;
10 }

/* prog14_2 OUTPUT---
105&26=8
-----*/
```

13



OR運算子「|」

• OR運算子「|」

- 可將運算子左、右兩邊運算元裡的每一個位元進行 OR 運算：

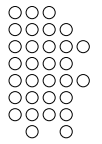
表 15.4.4 OR 運算子「|」之真值表

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

- 下圖為計算 a | b 的運算結果：

	0	1	1	0	1	0	0	0	1	十進位的值為 105
	0	0	0	1	1	0	1	0	0	十進位的值為 26
a b										
	0	1	1	1	1	0	1	1	1	十進位的值為 123

14



XOR運算子「^」

• XOR運算子「^」：

- 只有一個位元為 1 時其結果才會為 1，其餘皆為 0

表 15.4.5 XOR 運算子「^」之真值表

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

- 下圖為計算 a ^ b 的運算結果：

	0	1	1	0	1	0	0	0	1	十進位的值為 105
	0	0	0	1	1	0	1	0	0	十進位的值為 26
a ^ b										
	0	1	1	1	0	0	1	1	1	十進位的值為 115

15

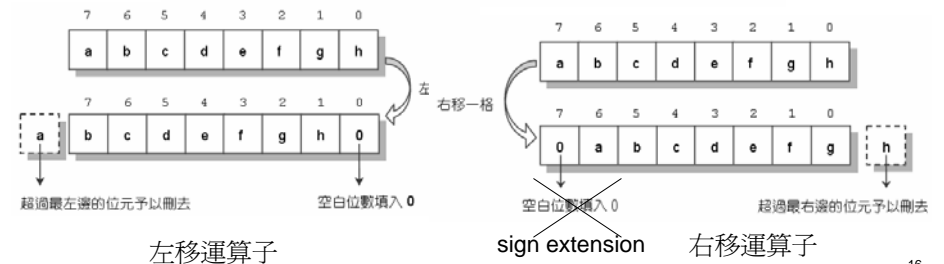


位元位移運算子

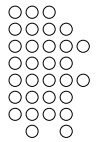
- 位元位移運算子可將位元左移或右移 n 個位元

表 15.4.6 位元位移運算子

位元位移運算子	意義
num << n	左移，將 num 的位元向左移 n 個位元
num >> n	右移，將 num 的位元向右移 n 個位元



16





位元左移運算子的範例

- 下面的程式可將整數89左移一個位元：

```
01  /* prog15_3, 左移運算子「<<」的使用範例 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  #define SIZE 8      /* 定義 SIZE 為 8, 代表以 8 個數字來顯示二進位 */
05  void show_binary(int); /* 宣告 show_binary() 函數的原型 */
06  int main(void)
07  {
08      int a;
09      a=(89<<1);      /* 將整數 89 往左移一個位元, 然後設定給變數 a 存放 */
10      printf("89 二進位的值為: ");
11      show_binary(89); /* 顯示數字 89 的二進位 */
12      printf("左移一個位元後: ");
13      show_binary(a); /* 顯示 89 左移一個位元後的二進位 */
14      printf("左移一個位元後的十進位值為: %d\n", a);
15      system("pause");
16      return 0;
17  }
```

17



位元欄位

低階的驅動程式常常使用

- 位元欄位：

- 一種特別的結構，能夠充分使用到結構變數中的每一個位元

位元欄位結構的宣告格式

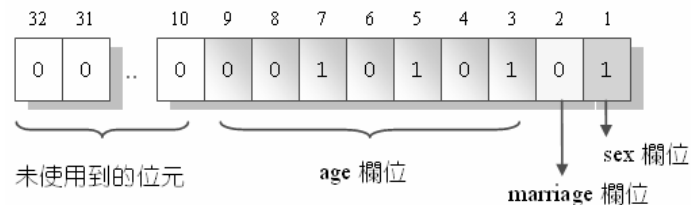
```
struct 位元欄位結構的名稱
{
    資料型態 欄位名稱1: 位元長度;
    資料型態 欄位名稱2: 位元長度;
    ...
    資料型態 欄位名稱n: 位元長度;
};
```

18

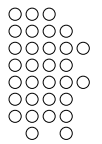


位元欄位結構宣告的範例

```
struct status /* 定義位元欄位結構 */
{
    unsigned sex:1; /* sex欄位, 佔1個位元 */
    unsigned marriage:1; /* marriage欄位, 佔1個位元 */
    unsigned age:7; /* age欄位, 佔7個位元 */
};
struct status tom; /* 宣告struct status的結構變數tom */
```



19



位元欄位結構的使用範例 (1/2)

```
01  /* prog15_4, 位元欄位結構的使用 */
02  #include <stdio.h>
03  int main(void)
04  {
05      struct status /* 定義位元欄位結構 */
06      {
07          unsigned sex:1;
08          unsigned marriage:1;
09          unsigned age:7;
10      };
11      struct status tom={1,0,21}; /* 宣告並設定結構變數的初值 */
12
13      if(tom.sex==0) /* 判別 sex 欄位的位元是否為 0 */
14          printf("性別:女,");
15      else
16          printf("性別:男,");
```

20



位元欄位結構的使用範例 (2/2)

```

17
18  if (tom.marriage==0)          /* 判別 marriage 欄位的位元是否為 0 */
19      printf("未婚,");
20  else
21      printf("已婚,");
22
23  printf("%d歲\n", tom.age);    /* 印出 age 欄位的值 */
24
25  printf("sizeof (tom)=%d\n", sizeof (tom)); /* 印出變數 tom 的大小 */
26
27  system("pause");
28  return 0;
29  }

/* prog15_4 OUTPUT--
性別:男,未婚,21歲
sizeof (tom)=4
-----*/

```

21

有趣的程式加速



```

int compare (const void * a, const void * b) { return ( *(int*)a - *(int*)b); }
...
int data[32768];
for (unsigned c = 0; c < 32768; ++c) data[c] = rand() % 256;
qsort(data, 32768, sizeof(int), compare);          11.78 sec without qsort
clock_t start = clock();                          2.35 sec with qsort
long long sum = 0;                                  Core i7 920 @ 3.5 GHz
for (unsigned i = 0; i < 100000; ++i)
    for (unsigned c = 0; c < arraySize; ++c)      Branch prediction
        if (data[c] >= 128) sum += data[c];
double elapsedTime = ((double)(clock() - start)) / CLOCKS_PER_SEC;

int t = (data[c] - 128) >> 31;                    2.56 sec without qsort
sum += ~t & data[c];                               2.59 sec with qsort

```

22