Bitwise operation
與七段顯示器對應技巧

Bitwise operation

符號	名稱	說明
&	And	以最低位元對齊做and運算
|	Or	以最低位元對齊做or運算
^	Xor	以最低位元對齊做xor運算
~	Not	將所有位元反轉
>>	右移	將所有位元右移，在空位元補0
<<	左移	將所有位元左移，在空位元補0

範例

AND: &

c = a & b :

```
  a = 0x00101001
& b = 0x10001100
  c = 0x00001000
```

OR: |

c = a | b :

```
  a = 0x00101001
| b = 0x10001100
  c = 0x10101101
```

XOR: ^

c = a ^ b :

```
  a = 0x00101001
^ b = 0x10001100
  c = 0x10100101
```

NOT: ~

c = ~a :

```
~ a = 0x00101001
  c = 0x11010110
```

Left shift: <<

c = a << 2:

```
  a = 0x00101001
  c = 0x10100100
```

Right shift: >>

Unsigned int: 0

c = a >> 2:

```
  a = 0x10010100
  c = 0x00100101
```

Right shift: >>

signed int: signed bit

c = a >> 2:

```
  a = 0x10010100
  c = 0x11100101
```

- 問題:偵測X中第P個位元是0或1
  - X = 01001101
  - P：所要處理的位元, 0 <= P <=7
- 方法:

```
M=0x1;
B=  X & ( M <<  P);
if( B == 0 ) Ans = 0;
else  Ans = 1;
```

  - 舉例:

偵測X中第3個位元是0或1
X = 0x01001101
M = 0x00000001
B = X & (M << 3)

Step1.M = (M << 3) = 0x00001000
Step2. B = X & M

```
  X = 0x01001101
& ( M << 3 ) = 0x00001000
  B = 0x00001000
```

- 問題:將X中第P個位元設定成0或1
  - X = 01001101
  - P：所要處理的位元, 0 <= P <=7
- 方法:

  M=0x1;

  Case1: B = X & ~(M << P); 設定成0

  Case2: B = X | (M << P);設定成1
  - 舉例:

將X的第三個bit變成0
X = 0x01001101
M = 0x00000001
B = X & ~(M << 3)

Step1 M = (M << 3) = 0x00001000

Step2 M = ~M = 0x11111011

Step3. B = X & M

```
    X = 0x01001101
& ~( M << 3 ) = 0x11111011
    B = 0x01001001
```

- 問題:將X中第P個位元0變1或1變0
  - X = 01001101
  - P: 所要處理的位元, 0 <= P <=7
- 方法:

  M=0x1;

  B = X ^ (M << P);
  - 舉例:

將X的第三個bit變成0

X = 0x01001101

M = 0x00000001

B = X ^ (M << 3)

Step1.M = (M << 3) = 0x00001000

Step2. B = X ^ M

```
  X = 0x01001101
^ (M << 3) = 0x00001000
-----------------------
  B = 0x01001001
```

七段顯示器對應技巧

- 可以額外宣告陣列儲存顯示數字的代碼

```
Byte pattern[10] = {0b00111111, //0
                    0b00000110, //1
                    0x01011001, //2
                    ……}
```

七段顯示器對應技巧

- 可以額外宣告陣列儲存現在顯示的數字，在更改顯示器上的數字時會更方便

```
int display_num[8];

//在更改數字時
send(pattern[ display_num[i] ]);
```