Micro Processor & Controller

Bit Wise & Shift Operations

Bit Wise Operations

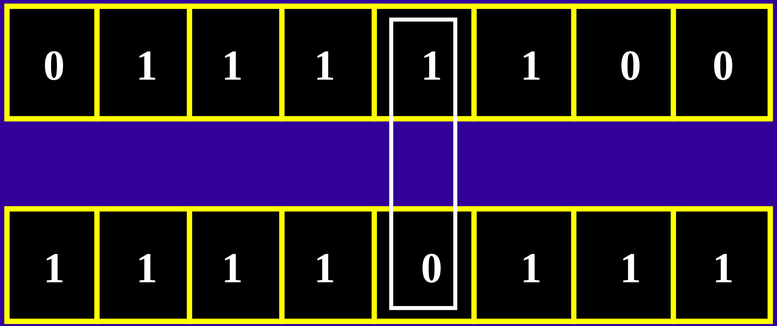
- C language has 4 bit wise operator for variable and constant:

  | (or)   &(and)   ^(xor)   ~(not).

- The bit wise operations are different from logic operations

  || (or)   &&(and)   !(not).

- The bit wise operations are operate on the object bit by bit, and not on the full object as Boolean value.



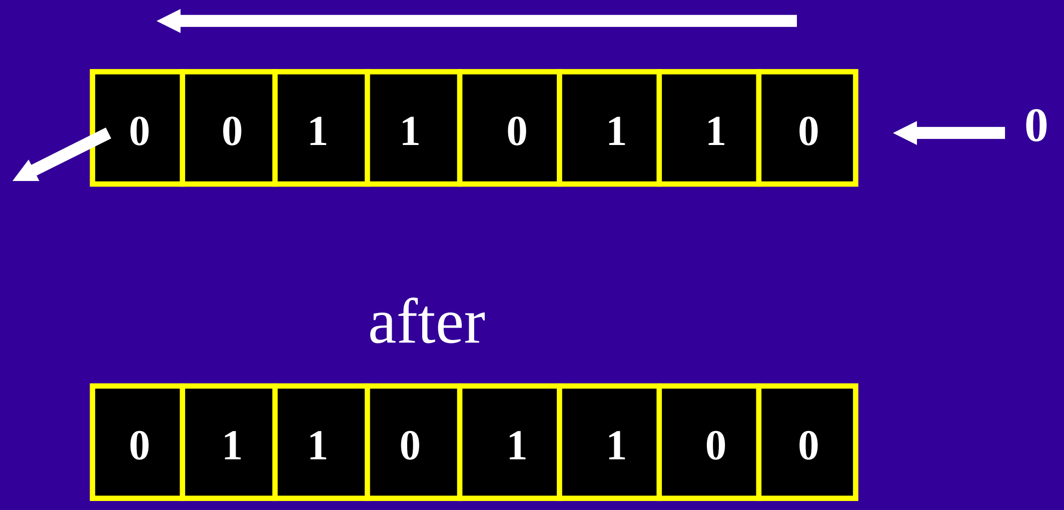
0	1	1	1	1	1	0	0
1	1	1	1	0	1	1	1

Shift Operations

- C language has two shift operations for variable and constant:

  >> (shift right) << (shift left).

- Shift left operation always insert ‘0’ to the LSB



0x36=54

0	0	1	1	0	1	1	0

0

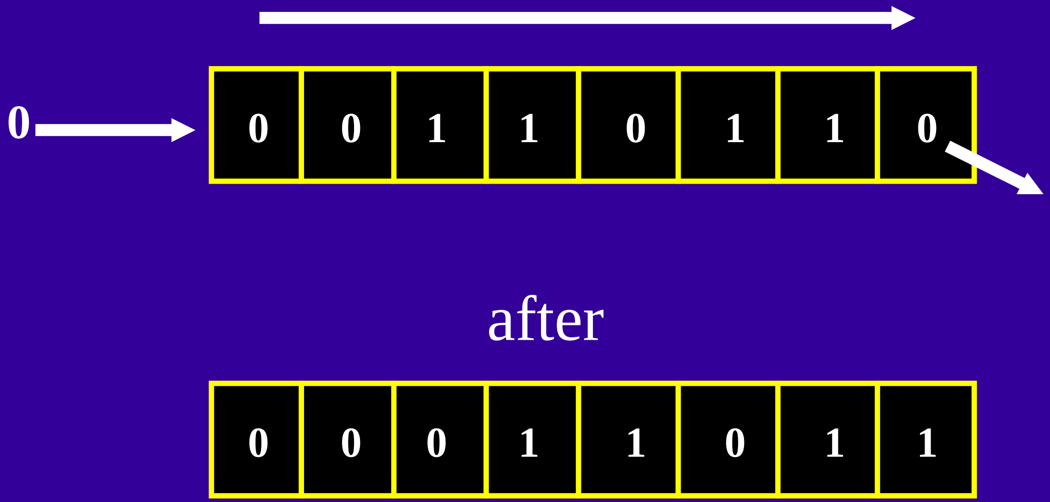
after

0x6C=108

0	1	1	0	1	1	0	0

Shift Operations

- Shift right operation insert, to the MSB, value that depended in the type of the variable or the constant.
- If the type is unsigned, ‘0’ is insert to the MSB



0

0	0	1	1	0	1	1	0

0x36=54

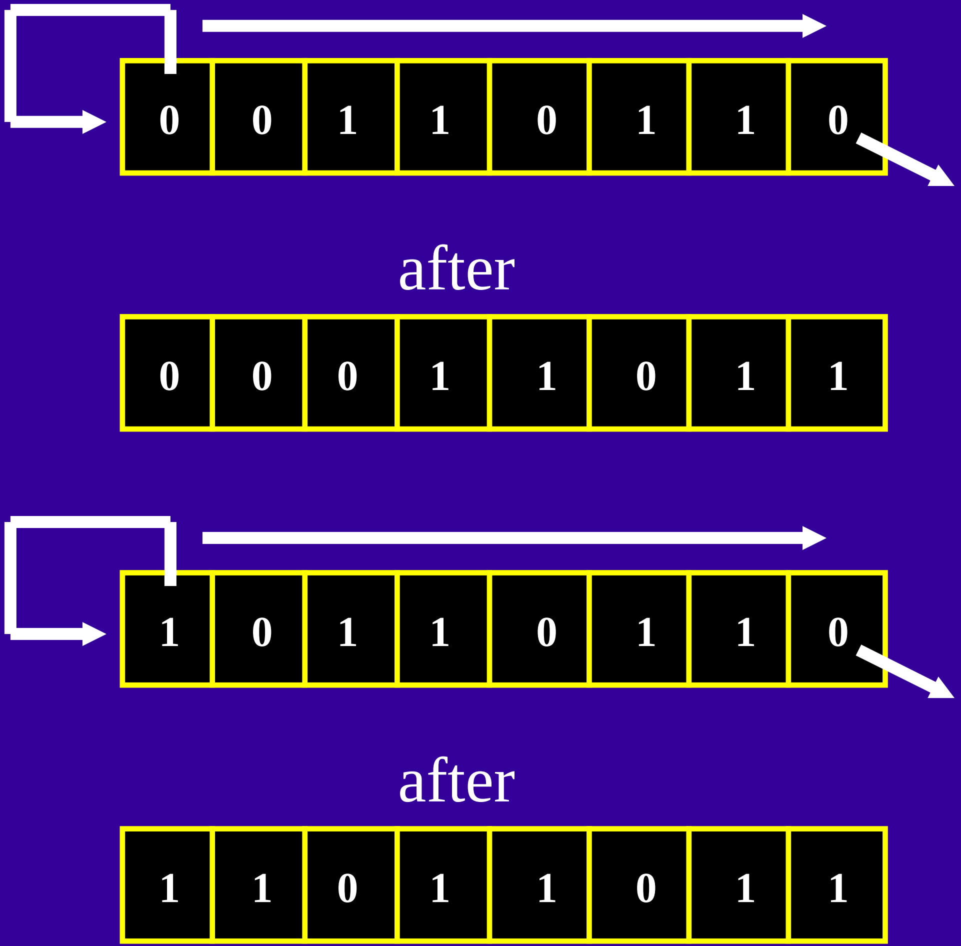
after

0	0	0	1	1	0	1	1

0x1B=27

Shift Operations

- If the type is sign, the MSB is duplicated (sign extended).



0 0 1 1 0 1 1 0 0x36=54

after

0 0 0 1 1 0 1 1 0x1B=27

1 0 1 1 0 1 1 0 0xB6=-74

after

1 1 0 1 1 0 1 1 0xDB=-37

Shift Operations Example (C)

```
int main ()
        {
        unsigned char x, y;
        char a, b;
        x = 1;              // x = 00000001
        y = x << 3;         // y = 8
        x = 255;            // x = 11111111
        y = x >> 3;         // y = 31

        a = 1;              // a = 00000001
        b = a << 3;         // b = 8
        a = -1;             // a = 11111111
        b = a >> 3;         // b = -1
        }
```

Constant Shift Operations

- The shift operations work with a constant too.
- For example:
  - 1 << 3 = 8

0	0	0	0	1	0	0	0

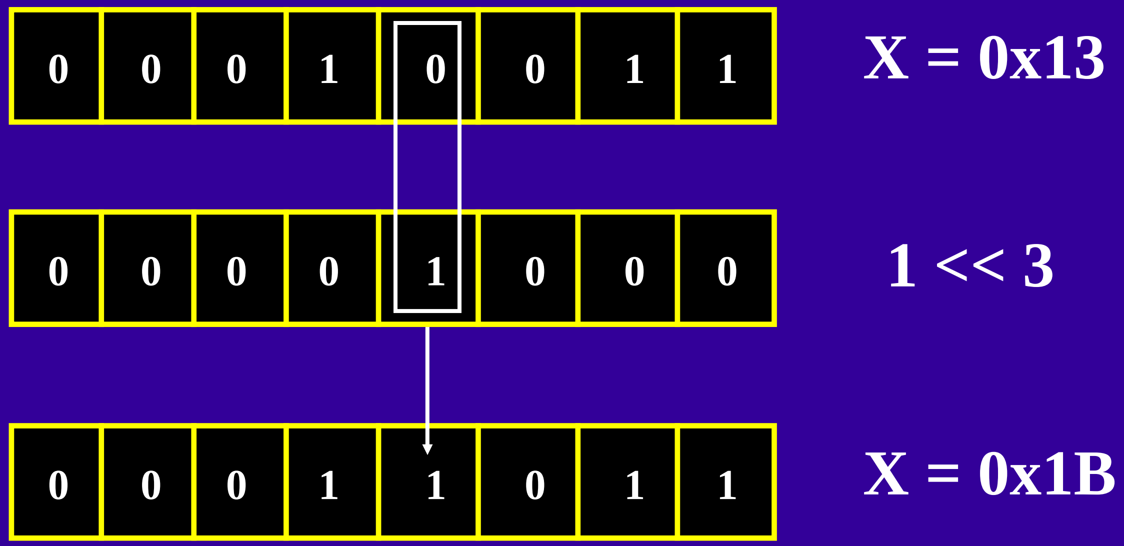
  - 48 >> 2 = 12

0	0	0	0	1	1	0	0

- Constant integer, is by default 32 bit signed.
- Unsigned constant is defined by U.

Bit Manipulations

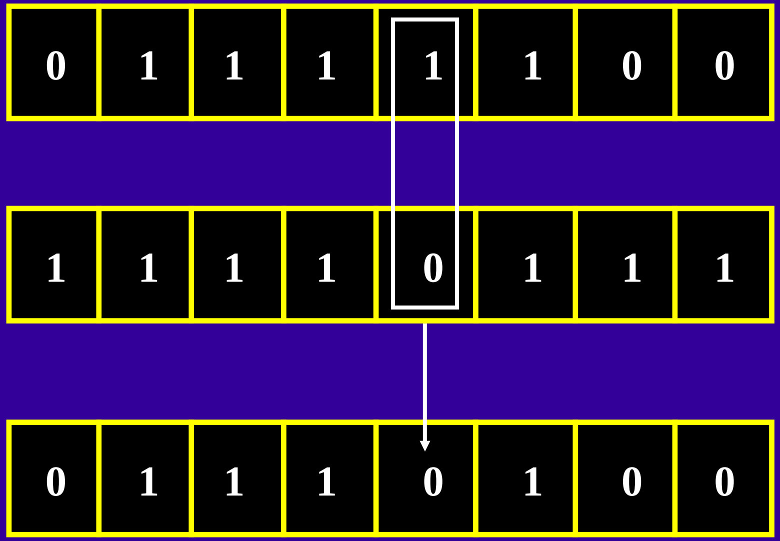
- How to set especial bit of the variable ?
- Set the k bit:
  - **x |= (1 << k)**
  - For example set bit 3 **X |= (1 << 3)**:



0	0	0	1	0	0	1	1	X = 0x13
0	0	0	0	1	0	0	0	1 << 3
0	0	0	1	1	0	1	1	X = 0x1B

Bit Manipulations

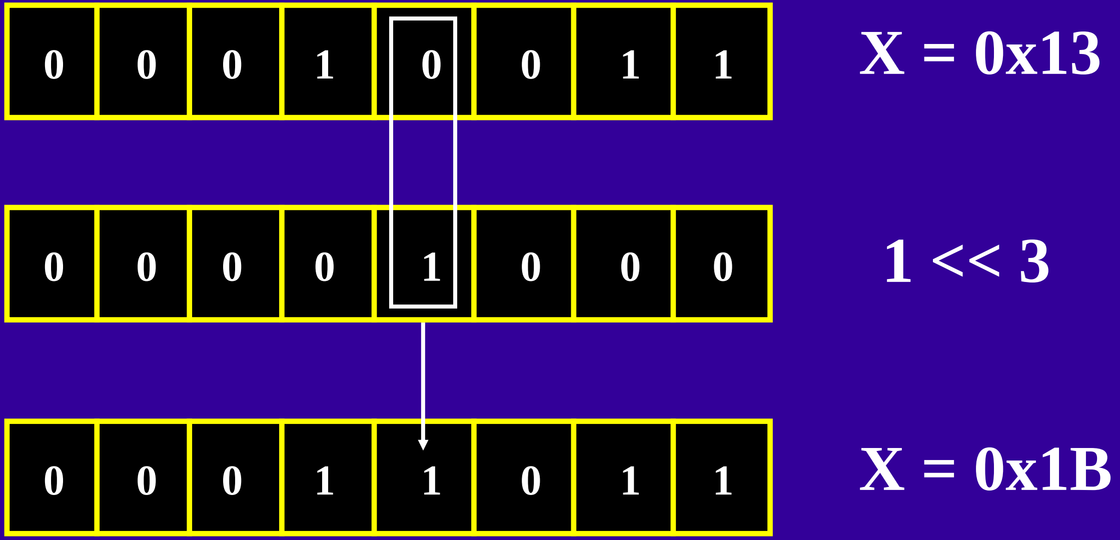
- How to clear especial bit of the variable ?
- Clear the k bit:
  - **x &= ~(1 << k)**
  - For example clear bit 3 **X &= ~(1 << 3)**:



0	1	1	1	1	1	0	0	**X = 0x7C**
1	1	1	1	0	1	1	1	**~(1 << 3)**
0	1	1	1	0	1	0	0	**X = 0x74**

Bit Manipulations

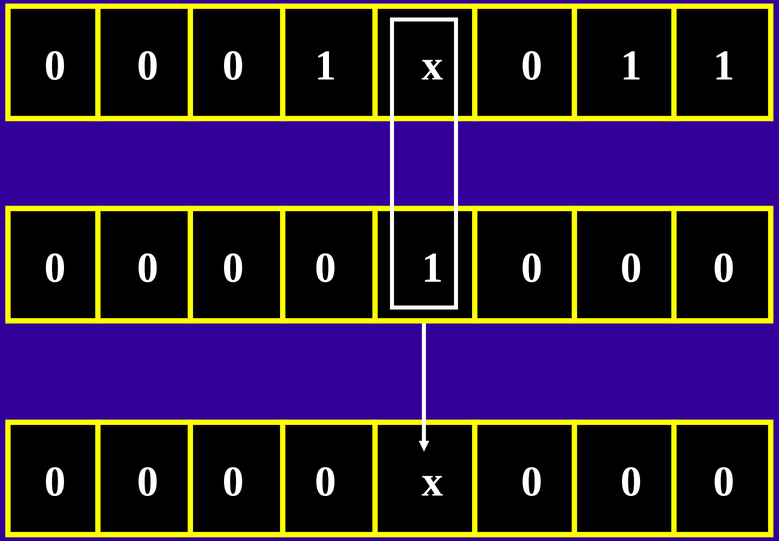
- How to toggle especial bit of the variable ?
- Toggle the k bit:
  - **x ^= (1 << k)**
  - For example set bit 3 **X ^= (1 << 3)**:



0	0	0	1	0	0	1	1	X = 0x13
0	0	0	0	1	0	0	0	1 << 3
0	0	0	1	1	0	1	1	X = 0x1B

Bit Sensing

- How to check especial bit of the variable ?
- Check the k bit of variable Y:
  - **Y & (1 << k)**
  - For example check bit 3 **Y & (1 << 3)**:

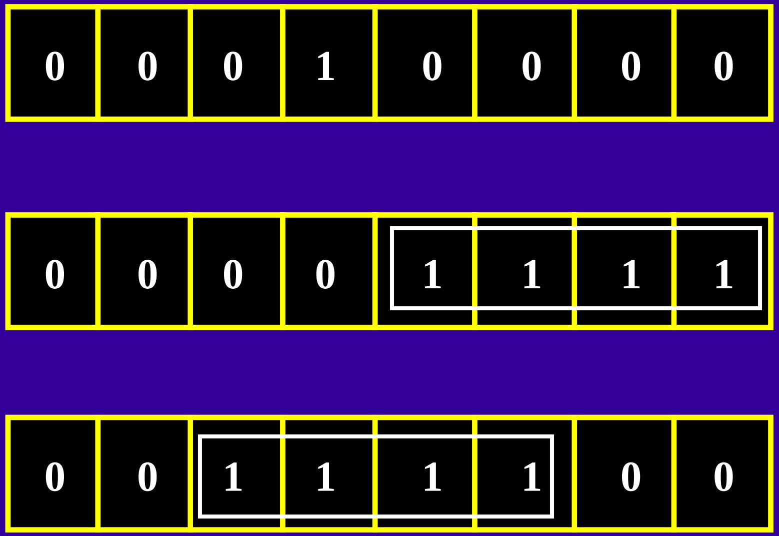


0	**0**	**0**	**1**	**x**	**0**	**1**	**1**	**Y**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1 << 3**
0	**0**	**0**	**0**	**x**	**0**	**0**	**0**	

If (x == 0) the all expression is FALSE
If (x == 1) the all expression is TRUE

Bit Manipulations

- How to mask N bits of the variable starting at K bit ?
- Mask N bit at K bit:
  - **((1 << N)-1) << K**
  - For example set 4 bit from bit 2 (**(1 << 4) - 1) << 2**:



0	0	0	1	0	0	0	0	**1<<4 = 0x10**
0	0	0	0	1	1	1	1	**(1<<4) – 1 = 0x0F**
0	0	1	1	1	1	0	0	**0x0F << 2**