Bit Manipulations

Comp 1402/1002

Octal and Hex Constants

Octal constant Nos. : Preceeded by 0 (zero)

Hexadecimal constants : Preceeded by 0x

```
int octalNum = 077;
int decNum = 77;
int hexNum = 0x77;
```

Bitwise Operators

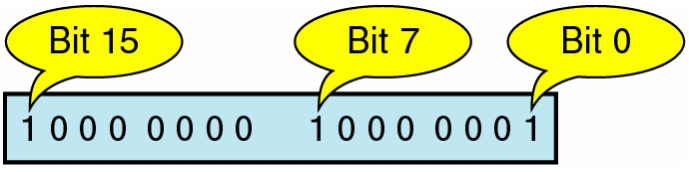
Name	Operator	Description
Bitwise And	&	Logical AND all bits
Bitwise Inclusive Or	|	Logical OR all bits
Bitwise Exclusive Or	^	Logical XOR all bits
Bit 1's Complement	~	Negation of all bits
Shift Right	>>	Shift bits right filling with zeroes
Shift Left	<<	Shift bits left filling with zeroes

Bitwise Operators

Bits are numbered from right to left

In space efficient systems bits represent data

Chess – Bit board representation…



Bitwise AND

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	0
1	0	0
1	1	1

Example : Bitwise AND

Hex numbers AF and 33:

```
  10101111   (AF)
& 00110011   (33)
  00100011   (23)
```

Masking to Find a bit

All zero except the bit to Find

```
 1010101x          110011x0
&00000001 Mask    &00000010 Mask
 0000000x          000000x0
```

The result : Either = Mask or Zero

Masking to Clear a bit

All ones except the bit to clear

```
 11101111          11001110
&11111101 Mask    &10111111 Mask
 11101101          10001110
```

Bit is set to zero...

Bitwise OR

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	1
1	0	1
1	1	1

Example : Bitwise OR

Hex numbers AF and 33:

```
  10101111   (AF)
 |00110011   (33)
  10111111   (BF)
```

Masking to set a bit

All zeroes except the bit to set

```
 111011x1          1x001110
|00000010 Mask    |01000000 Mask
 11101111          11001110
```

The bit is set to one...

Masking to Find a bit

All ones except the bit(s) to test

```
 1010101x          110011x0
|11111110 Mask    |11111101 Mask
 1111111x          111111x1
```

The result : all ones or not...

Bitwise XOR

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	1
1	0	1
1	1	0

Example : Bitwise XOR

Hex numbers AF and 33:

```
  10101111   (AF)
^ 00110011   (33)
  10011100   (9C)
```

Masking to flip a bit

All zeroes except the bit to flip

```
 11101101          11001110
^00000010 Mask    ^01000000 Mask
 11101111          10001110
```

Bit is flipped...

Bitwise Complement

Operand Bit	Result
0	1
1	0

Example : Bitwise Complement

Hex number 33 :

```
~00110011  (33)
 11001100  (CC)
```

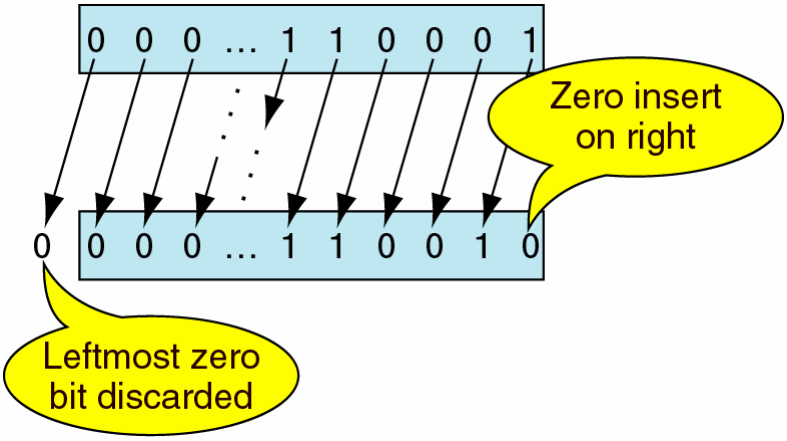
Shifting Bits

Two bit shifting operators:

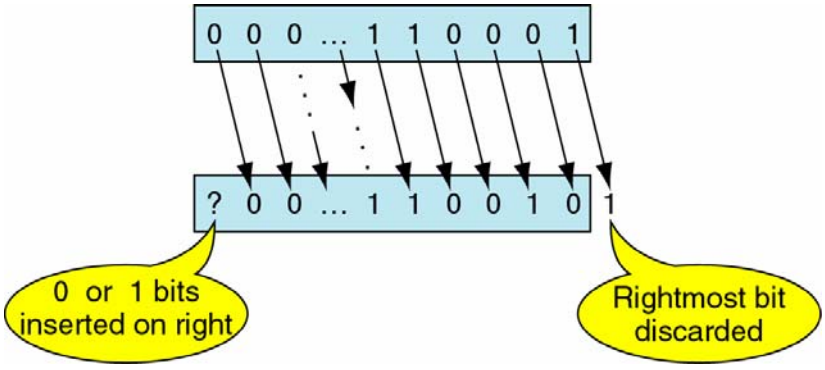
```
int x = 49;
x << 1; /* shift bits left 1 place */
The new value of x is 98.

x = 49;
x >> 1; /* shift bits right 1 place */
The new value of x is 24.
```

Shifting Bits - LEFT



Shifting Bits - RIGHT



Shifting Bits

Shifting left "adds" zeros to the right

Shifting right is implementation specific:

- zeros "added" for **`unsigned`**
- Sign bit added otherwise.

Shifting Bits - LEFT

Programmer can shift arbitrary number of bits..

```
unsigned int x;
x = (1 << 1);  /* x = 2 */
x = (1 << 2);  /* x = 4 */
x = (1 << 15); /* x = 32768 */
```

Shifting **`unsigned ints`** **LEFT** by 1 is multiplying by 2

Shifting Bits - RIGHT

Can shift an arbitrary number of bits RIGHT too!

```
unsigned int x;
x = (32768 >> 1);  /* x = 16384 */
x = (32768 >> 2);  /* x =  8192 */
x = (32768 >> 14); /* x =     2 */
```

Shifting **`unsigned ints`** **RIGHT** by 1 is DIV by 2

Creating Masks

Use these Operations Intelligently

```
m1 = 1 << 4; /* mask m1 is 00010000 */

m2 = 1 << 7; /* mask m2 is 10000000 */

m3 = m1 | m2;/* mask m3 is 10010000 */

m4 = ~m1;    /* mask m4 is 11101111 */

m5 = m1 - 1; /* mask m5 is 00001111 */
```

Use Bits : Changing the Case

'A' and 'a' differ in the fifth bit!

```
'A' is 01000001   41 (Hex)
'a' is 01100001   61 (Hex)
XOR with 1 << 5    00100000 (MASK)
Result :'a'-> 'A' 01000001
AND with ~(1 << 5)11011111 (MASK)
Result :'a'-> 'A' 01000001
```

Set, Flip, Get : Groups of bits

Masks can be more than one bit

Bitwise Independence is crucial

Allows arbitrary bits to be manipulated

Inside one integer we can store a database!

All Inside ONE Integer

Consider : 4 bytes on some machines

Question : Which of given 32 numbers > 1 are even/odd?

Answer: rightmost bit refers to the number 2

```
0010 1000 0010 0010 1000 1010 0010 1011
```

Other Applications

- Control operations in microprocessor control such as room example.
- A byte may represent a series of house switches and bit 4 might then have the meaning:
  0 : light off in living room
  1 : light on in living room
  How to turn the light on ? "set" bit 4 to one.
- Operating system programming.
- Internet applications – Passing bits down the stream. Always bits are transmitted !!!

Printing Bits of Character

```
#include <stdio.h>
/* Prototype of function */
void printbits(char);
int main(void)
{
    unsigned char x=' ';
    printbits(x);
    return 0;
}
```

Printing Bits of Character

```
/* Outputs bit pattern for a byte */
void printbits (char character){
    unsigned char temp;
    unsigned char mask =0x80; /*fixed size of mask ; 10000000 */

    int size;
    size = 8*sizeof(character) - 1;   // could use size = 7
    printf ("The bit pattern for %c is :", character);
          /* Continued on Next Slide */
```

Printing Bits of Character

```
/* After Previous Slide */
for (int i= 0; i<= size ; i++){
      temp = character & mask;
      if (temp == 0)              /*Then this bit is 0 */
          printf("0") ;
     else                         /*Then this bit is 1 */
          printf("1") ;
     mask = mask>>1;
}
 printf(”\n") ;
 return ;
}
```

Printing Bits of Character (2)

```
/* Outputs bit pattern for a byte */
void printbits(char character) {
    unsigned char mask =0x80; /*fixed size of mask ; 10000000 */
    int i, size;
    size = (8*sizeof(character) - 1) ;
    for (i= 0; i<= size ; i++) {
          printf(“%1u”, (int)((character & mask) >> (size-i))
/*Shifting this right to make this bit the LSB */
/* Cast into an integer */
          mask = mask >>1 ;
    }
printf(”\n") ; return ;
}
```

Error Detection : The Parity Bit.

- The ASCII code is a seven bit code
- Leaves the most significant bit (bit 7) = 0.
- This bit can be used in data transmission
- Checks an error in transmission.
- The parity bit, bit 7, is set so that the number of bits in the byte is either:
  Even: Even Parity
  Odd: Odd Parity
- Example : Assume Even Parity.
  The letter A is 01000001
  So the parity bit is left as zero

Error Detection : The Parity Bit.

- Assume Even Parity
- The letter C is 01000011
  So the parity bit set to 1 to give even parity : 11000011.
- When the byte is transmitted, if one bit is flipped (error)
- The parity would no longer be even
- Receiver would know that there is an error.
- However, the user would not know which bit is incorrect.

Error Detection : Set Parity Bit.

```
#include <stdio.h >
/* prototype of function  */
char parity(char);
void printbits(char);

void main(void) { /*outputs bit pattern for a byte inc. parity */
    char character ='C';
    printbits(character);
    printbits(parity(character));
    return;
}
```

Error Detection : The Parity Bit.

```
char parity(char localchar) {
unsigned char temp;
unsigned char mask =0x80;
unsigned char setmask = 0x80;
int count=0, i, size;
    size = 8*sizeof(localchar)-1; /*Could start at bit ‘6’ instead of ‘7’ */
for (i= 0; i<= size ; i++) {
            temp = localchar & mask;
            if (temp != 0)
               count++;
            mask=mask >>1;
    }
    if(count%2 != 0)
             localchar |= setmask;
    return (localchar) ;
}
```