MA 511: Computer Programming

Lecture 25

http://www.iitg.ernet.in/psm/indexing_ma511/y08/index.html

Partha Sarathi Mandal

psm@iitg.ernet.ac.in

Dept. of Mathematics, IIT Guwahati

Semester 1, 2008-09

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55 Class: 1G2

MA512 Lab : Wed 14:00-16:55

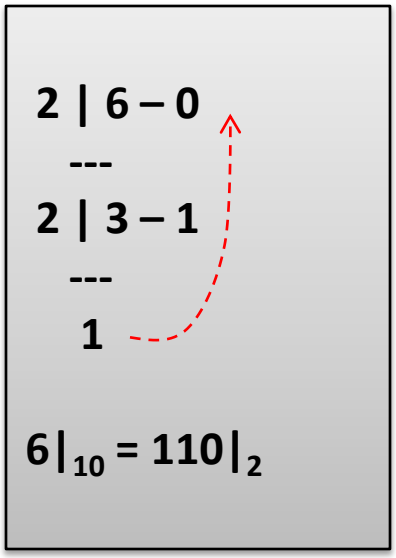
LOW-LEVEL PROGRAMMING

Number systems

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	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

Decimal & Binary conversion

Decimal	Binary	Decimal
0	= 000 = $0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	
1	= 001	
2	= 010	
3	= 011 = $0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	
4	= 100	
5	= 101	
6	= 110 = $1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	
7	= 111 = $1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	



One’s complement Operator (~)

- It’s a unary operator that causes the bits of its operand to be inverted so that 1s become 0s and 0s become 1s.
- Operator always precedes its operand.
- The operand must be an integer-type quantity (including, long, short, unsigned, char, etc).
- Generally, the operand will be an unsigned octal or an unsigned hexadecimal quantity, though this is not a firm requirement.

Ex: $07ff|_{16} = 0000\ 0111\ 1111\ 1111|_2$

~(0000 0111 1111 1111) = 1111 1000 0000 0000

$= f800|_{16}$

~(07ff) = f800

Bases

- Normally , C assumes that integer constants are decimal (base 10)
- How can a computer tell whether a integer const is meant to be a **decimal**, **hexadecimal**, or **octal** value?
- A prefix 0x or 0X (Zero-exe) that you are specifying a hexadecimal value
- A prefix 0 (zero) means that you are writing in octal

  Ex: 16 (decimal) = 020 (octal) = 0x10 (hexadecimal)
- In the binary code used internally by computers.
- To display an integer in octal use %o in hexadecimal %x.
- If you want to display prefixes %#o , %#x , %#X

Bases

```
#include<stdio.h>
main(){
    unsigned i= 0x5b3c;
        printf("Hex i = %x ~i = %x \n", i, ~i);
        printf("Dec i = %u ~i = %u \n", i, ~i);
} /* end of main */
```

```
#include<stdio.h>
main(){
    int x = 100;
    printf(“hex = %x;  oct = %o; dec = %d\n", x, x, x);
    printf(“hex = %#x;  oct = %#o; dec = %d\n", x, x, x);
}
```

hex = 64; oct = 144; dec = 100

hex =0x64; oct= 0114; dec= 100

Bit Level Operations

- Operations on bits and string of bits.
  - & : Bitwise AND
  - | : Bitwise OR
  - ^ : Bitwise exclusive OR
  - ~ : One's complement
  - << : Left shift
  - >> : Right shift
- This a special feature of C programming language

a	0	0	1	1
b	0	1	0	1
a&b	0	0	0	1
a|b	0	1	1	1
a^b	0	1	1	0
~a	1	1	0	0

String p	p >> 3	Action
01011010	00001011	P shifted right by 3 position. Left most bits filled with 0s
String q	q << 4	Action
11000110	01100000	Q shifted left by 4 positions. Right most bits filled with 0s

Bit Level Operations

The AND Gate

0&0 = 0
1&0 = 0
0&1 = 0
1&1 = 1
1010 &1100
=1000

The OR Gate

0|0 = 0
1|0 = 1
0|1 = 1
1|1 = 1
1010 | 1100
=1110

Exclusive-OR, or XOR

0^0 = 0
1^0 = 1
0^1 = 1
1^1 = 0

1010 ^ 1100
= 0110

The NOT Gate, or Inverter

~0 = 1
~1 = 0
~10101 = 01010

Bit Level Operations

Adding Binary Numbers

0+0=0
1+0=1
1+1=10

```
  11    (carry)
  1010
+1111
_______
11001
```

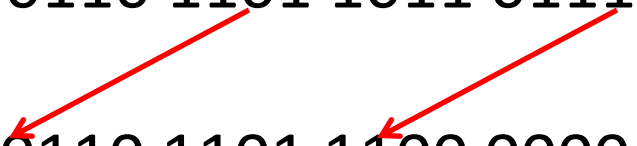
Binary Multiplication

1*1=1
1*0=0
0*1=0

```
 101
 * 11
_____
 101
1010
_____
1111
```

Shifting

a = 0110 1101 1011 0111 =0x6db7



a<<6 = 0110 1101 1100 0000 =0x6dc0

```
main(){
    unsigned a = 0xf05a;
    int b = a;
    printf("%u %d\n", a, b); 1111 0000 0101 1010 = 0xfo5a
    printf("%x\n", a>>6);     0000 0011 1100 0001 = 0x3c1
    printf("%x\n", b>>6);     1111 1111 1100 0001 = 0xffc1
```

Masking

- Masking is a process in which a given bit pattern is transformed into another bit pattern by means of a logical bitwise operation.

Example:

Unsigned int a=0x6db7, b;

Extract the rightmost 6 bits and assign to b

b =a & 0x3f

a = 0110 1101 1011 0111

mask = 0000 0000 0011 1111

---

b = 0000 0000 0011 0111 = 0x37

The mask prevents the leftmost 10 bits from being copies from a to b

Example:

Initially a=0x6db7 assume 16-bit word size

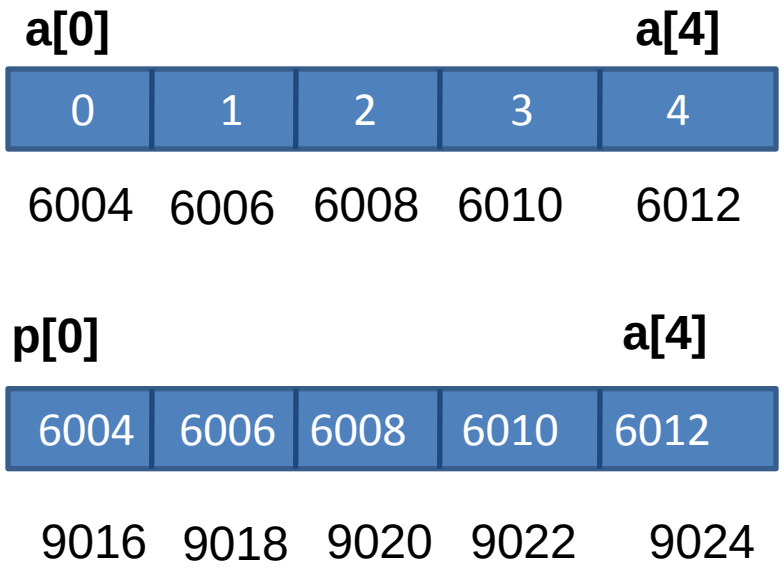
- a&=0x7f => a=a & 0x7f a= 0x37
- a^=0x7f 0x6dc8
- a|=0x7f 0x6dff
- a<<=5 0xb6e0

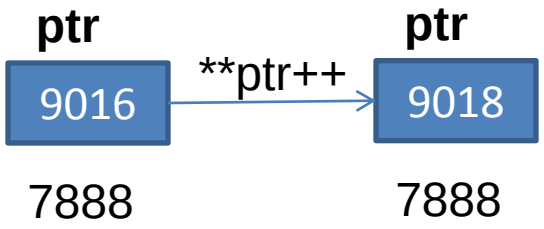
Pointer to pointer

```
main(){
   static int a[] ={0,1,2,3,4};
   static int *p[] ={a, a+1, a+2, a+3, a+4};
    int **ptr;
    ptr=p;

    **ptr++;

}
```





Recursive function

```
function fact(N){
    if(N==1) return 1;
    else return N*fact(N-1);
}
```

- Let N = 3, the 3!= 3 x 2 x 1 = 6.
- First time the function is called, N =3 the value of N, but not of fact(N-1), pushes N (value=3) on the stack, and calls itself for the second time with the value 2. …
- Third time function calls itself with the value 1
- as N=1, the function returns 1. Since the value of fact(1) is now known, it reverts back to it's second execution by popping the last value (2) from the stack and multiplying it by 1.
- This operation gives the value of fact(2), so the function reverts to it's first execution by popping the next value (3) from the stack, and multiplying it with fact(2), giving the value 6, that’s the value the function finally returns.