Bit Operations

Ned Nedialkov

McMaster University
Canada

SE 3F03
February 2014

Outline

Logical shifts

- Left and right shifts: the new bits are always 0
- **shl** shift left, **shr** shift right
  - number of positions is a constant or stored in **CL**
  - the last bit shifted out is in `CF`
- E.g.

```
mov ax, 0123h          ;ax = 0000 0001 0010 0011
shl ax, 1              ;ax = 0000 0010 0100 0110, CF=0
shl ax, 7              ;ax = 0010 0011 0000 0000, CF=1
```

- Suitable for fast multiplication/division by powers of 2
  - ...but for unsigned numbers

Arithmetic shifts

- Suitable for signed numbers
- **sal** shift arithmetic left
  - same as **shl**
  - if the sign bit is unchanged, correct result
- **sar** shift arithmetic right
  - the new bits are copies of the sign bit
- The last bit shifted out is in `CF`
- E.g.

```
mov ax, C123H    ;ax=1100 0001 0010 0011
sal ax, 1        ;ax=1000 0010 0100 0110, CF=1
                 ;ax=8246h
sal ax, 1        ;ax=0000 0100 1000 1100, CF=1
                 ;ax=048Ch
sar ax, 2        ;ax=0000 0001 0010 0011, CF=0
                 ;ax=0123h
```

Rotate shifts

- **ror** rotate right, **rol** rotate left
- Like logical shifts
- Shifted bits are moved on the other side
- The last bit shifted out is in `CF`
- E.g.

```
mov ax, 0C123H  ;ax=1100 0001 0010 0011
rol ax, 1       ;ax=1000 0010 0100 0111, CF=1
                ;ax=8247h
rol ax, 1       ;ax=0000 0100 1000 1111, CF=1
                ;ax=048Fh
rol ax, 1       ;ax=0000 1001 0001 1110, CF=0
                ;ax=091Eh
ror ax, 2       ;ax=1000 0010 0100 0111, CF=1
                ;ax=8427h
ror ax, 1       ;ax=C123h
```

- **rcl** rotate carry left, **rcr** rotate carry right
- The bits go through the carry flag, `CF` (see the text for details)

Where are rotate shifts used?

- To shift data one way and then the other way without losing data
- Assume `x` is 16 bits and consider

  `(x << 6) | (x >> 10)`

  Can we do it with a rotate shift?
  E.g.

```
x = 1010 0110 0001 0001
x<<6  = 1000 0100 0100 0000
x>>10 = 0000 0000 0010 1001
|     = 1000 0100 0110 1001
rotate x to the left by 6
        1000 0100 0110 1001
```

- Hashing
- CRC, Cyclic Redundancy Check

Example: count 1's

From http://www.drpaulcarter.com/pcasm/

```
        ;; count the 1 bits in eax
        mov bl, 0                   ;number of bits
        mov ecx, 32                 ;32 bits
count_loop:
        shl eax, 1                  ;shift left by one
        ;; CF contains the shifted out bit
        jnc     skip_inc            ;if CF=0, continue
        inc bl                      ;bl++
skip_inc:
loop    count_loop
```

Bitwise operations

- **and**, **or**, **xor**, **not**
- E.g.

```
mov ax, 1234h
and eax, 00FFh
;; eax = 0034h
```

- **test**
  - Like **and** but does not store the result
  - Sets the `FLAGS` register

Bit operations in C

What is the output of this program:

```
#include <stdio.h>
int main() {
  int a = -2;
  printf("a = %X\n", a );
  int a1 = a >> 5;
  printf("a1 = %X\n", a1);
  int a2 = a << 5;
  printf("a2 = %X\n", a2);
  int a3 = a & (a^a);
  printf("a3 = %X\n", a3);
  int a4 = a & 0x5;
  printf("a4 = %X\n", a4);
  int a5 = 0x10000000;
  a5 -= 1;
  printf("a5 = %X\n", a5);
  return 0;
}
```

What is the output of this program:

```
#include <stdio.h>
int main() {
  unsigned short int a = 2;
  printf("a  =  %X\n", a );
  unsigned short int a1 = a >> 5;
  printf("a1  =  %X\n", a1);
  unsigned short int a2 = a << 5;
  printf("a2  =  %X\n", a2);
  unsigned short int a3 = a & (a^a);
  printf("a3  =  %X\n", a3);
  unsigned short int a4 = a & 0x5;
  printf("a4  =  %X\n", a4);
  unsigned short int a5 = 0x1000;
  a5 -= 1;
  printf("a5  =  %X\n", a5);
  return 0;
}
```

```
a  = FFFFFFFE
a1 = FFFFFFFF
a2 = FFFFFFC0
a3 = 0
a4 = 4
a5 = FFFFFFF

a  = 2
a1 = 0
a2 = 40
a3 = 0
a4 = 0
a5 = FFF
```