Bit Operations: Examples and Sample Problems

ICS312
Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)
Bit Shifts

Consider the following instructions

- `mov ax, 0F471h`
- `sar ax, 3`
- `shl ax, 7`
- `sar ax, 10`

At each step give the content of register ax (in hex and binary) and the value of CF (assuming that initially it is equal to 0)
Bit Shift (Solutions)

```assembly
mov   ax, 0F471h
    ax = 1111 0100 0111 0001
    ax=F471h                  CF=0
sar   ax, 3
    ax = 1111 1110 1000 1110
    ax=FE8Eh                  CF=0
shl   ax, 7
    ax = 0100 0111 0000 0000
    ax=4700h                  CF=1
sar   ax, 10
    ax = 0000 0000 0001 0001
    ax=0011h                  CF=1
```
Example Using Shifts

- Let’s go through Example 3.1.5 in the book
- Say you want to count the number of bits that are equal to 1 in register EAX
- One easy way to do this is to use shifts
  - Shift 32 times
  - Each time the carry flag contains the last shifted bit
  - If the carry flag is 1, then increment a counter, otherwise do not increment a counter
  - When you’re done the counter contains the number of 1’s
- Let’s write this in x86 assembly
  - The textbook has it written a bit differently (uses the loop instruction)
Example Using Shifts

; Counting 1 bits in EAX
mov bl, 0 ; bl: the number of 1 bits
mov cl, 32 ; cl: the loop counter
loop_start:
    shl eax, 1 ; left shift
    jnc not_one ; if carry != 1, jump to not_one
    inc bl ; increment the number of 1 bits
not_one:
    dec cl ; decrement the loop counter
    jnz loop_start ; if more iterations then ; goto loop_start
The same, with the adc instruction

- Convenient instruction: **adc** (add carry)
  - `adc dest, src ; dest += src + cf`

; Counting 1 bits in EAX
  mov  bl, 0 ; bl: the number of 1 bits
  mov  cl, 32 ; cl: loop counter

loop_start:
  shl  eax, 1 ; left shift
  adc  bl, 0 ; add the carry to bl
  dec  cl ; decrement the loop counter
  jnz loop_start ; if more iterations then
      ; goto loop_start
The same, with the loop instruction

Remember the loop instruction
- \texttt{loop <label>}; decrements loop index (in ecx)
  ; and branches if ecx isn’t 0

; Counting 1 bits in EAX
  \texttt{mov bl, 0}; \texttt{bl}: the number of 1 bits
  \texttt{mov ecx, 32}; \texttt{ecx}: the loop counter

\texttt{loop}_\texttt{start}:
  \texttt{shl eax, 1}; left shift
  \texttt{adc bl, 0}; add the carry to bl
  \texttt{loop loop_start}; decrement ecx and
  ; then loop if needed
Bit Mask Operations Examples

mov eax, 04F346BA2h
or ax, 0F000h ; turns on 4 leftmost bits of ax
; eax = 4F34FBA2
xor eax, 000400000h ; inverts bit 22 of EAX
; eax = 4F74FBA2
xor ax, 0FFFFh ; 1’s complement of ax
; eax = 4F74045D
Remainder of a Division by $2^i$

- To find the remainder of a division of an operand by $2^i$, just AND the operand by $2^i - 1$
- Why does this work?

Let $a$ be an $s$-bit quantity, $b$ be the upper $s-i$ bits, and $c$ be the lower $i$ bits.

Therefore, $a = b \times 2^i + c$, and $c$ is the remainder!

The remainder is simply the lowest $i$ bits!
Remainder of a Division by $2^i$

Let’s compute the remainder of the integer division of $123\text{d}$ by $2^5=32\text{d}$ (unsigned) by doing an AND with $2^5-1$

```
mov ax, 123          0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1
mov bx, 0001Fh       0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
and  bx, ax          0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1
```

The remainder when dividing 123 by 32 is $11011_2 = 27_{10}$
Boolean Bitwise Instructions

mov ax, 0C123h

and ax, 082F6h ; ax = C123 AND 82F6 = 8022

or ax, 0E34Fh ; ax = 8022 OR E34F = E36F

xor ax, 036E9h ; ax = E36F XOR 36E9 = D586

not ax ; ax = NOT D586 = 2A79
Example: max(a,b)

Say we want to store into ecx the maximum of two (signed) numbers, one stored in eax and the other one in [num]

Here is a simple code to do this

```
cmp    eax, [num]
jge    next ; conditional branch
mov    ecx, [num]
jmpend
next:
mov    ecx, eax
end:
```

Let’s rewrite this without a conditional branch!

- Conditional branches are bad for performance
Example: max(a, b)

- To avoid the conditional branch, one needs a SETxx instruction and clever bit masks
- We use a helper register, ebx, which we set to all zeros
  
  ```
  xor ebx, ebx
  ```
- We compare the two numbers
  
  ```
  cmp eax, [num]
  ```
- We set the value of bl to 0 or 1 depending on the result of the comparison
  
  ```
  setg bl
  ```
  - If eax > [num], ebx = 1 = 0...01b
  - If eax <= [num], ebx = 0 = 0...00b
- We negate ebx (i.e., take 1’s complement and add 1)
  
  ```
  neg ebx
  ```
  - If eax > [num], ebx = FFFFFFFFFFFh
  - If eax <= [num], ebx = 0000000000h
Example: \texttt{max(a,b)}

- We now have:
  - \texttt{eax} contains one number, \texttt{[num]} contains the other
  - If \texttt{eax} > \texttt{[num]}, \texttt{ebx} = FFFFFFFFH (we want to “return” \texttt{eax})
  - If \texttt{eax} <= \texttt{[num]}, \texttt{ebx} = 0000000000h (we want to “return” \texttt{[num]})
- If \texttt{eax} is the maximum and we AND \texttt{eax} and \texttt{ebx}, we get \texttt{eax}, otherwise we get zero
- If \texttt{[num]} is the maximum and we AND \texttt{[num]} and NOT(\texttt{ebx}), we get \texttt{[num]}, otherwise we get zero
- So if we compute \((\texttt{eax AND ebx}) \text{ OR } ([\texttt{num]} \text{ AND NOT(\texttt{ebx}))})\) we get the maximum!
  - If \texttt{eax} is the maximum (\texttt{ebx} = FFFFFFFFH):
    - \((\texttt{eax AND ebx}) \text{ OR } ([\texttt{num]} \text{ AND NOT(\texttt{ebx}))}) = \texttt{eax OR 0...0 = eax}\)
  - If \texttt{[num]} is the maximum (\texttt{ebx} = 0000000000h):
    - \((\texttt{eax AND ebx}) \text{ OR } ([\texttt{num]} \text{ AND NOT(\texttt{ebx}))}) = 0...0 \text{ OR } [\texttt{num}] = [\texttt{num}]\)
- Let’s just write the code to compute \((\texttt{eax AND ebx}) \text{ OR } ([\texttt{num]} \text{ AND NOT(\texttt{ebx}))})\)
Example: max(a,b)

- Computing ((eax AND ebx) OR ([num] AND NOT(ebx))):
  - mov ecx, ebx ; ecx = eax AND ebx
  - and ecx, eax ; ecx = eax AND ebx
  - not ebx ;
  - and ebx, [num] ; ebx = [num] AND NOT(ebx)
  - or ecx, ebx ; voila!

- Whole program:
  - xor ebx, ebx; ebx = 0
  - cmp eax, [num] ; compare eax and [num]
  - setg bl ; bl = 1 if eax > [num], 0 otherwise
  - neg ebx ; take one's complement + 1
  - mov ecx, ebx ;
  - and ecx, eax ; ecx = eax AND ebx
  - not ebx ;
  - and ebx, [num] ; ebx = [num] AND NOT(ebx)
  - or ecx, ebx ; voila!
Example Operations in C

short int s; // 2-byte signed
short unsigned int u; // 2-byte unsigned
s = -1; // s = 0xFFFF
u = 100; // u = 0x0064
u = u | 0x0100; // u = 0x0164
s = s & 0xFFFF0; // s = 0xFFFF0
s = s ^ u; // s = 0xFE94
u = u << 3; // u = 0x0B20
s = s >> 2; // s = 0xFFA5