CPSC 232
Intro to Assembly Language Programming

Dr. Jochen

Carter: Chapter 3 – Bit Operations

Bit Operations

- Operations/Manipulations on the lowest level/unit of data storage
- Bit Operations include:
  - Logical – AND, OR, NOT, XOR, NOR, NAND
  - Shifts
    - Arithmetic
    - Logical
- Many assembly programmers use bit manipulations to:
  - check/set bit fields
  - check for certain results after computation
  - perform quick arithmetic operations

Why Bit-Level Operations?

- At Assembly Level (low level code), often work with bits for:
  - Device drivers – talking to device hardware
  - Communications – specific data format for protocol
  - Clearing/setting values in register, memory, etc.
  - & Much, much more!

- What sort of operations?
  - Bit masking (like subnet masks in networking)
  - Searching for specific bit sequences in input/data
  - Conditional branching (true/false)
  - Comparisons
  - Faster math
    - Multiply/divide, left-shift/right-shift

Shifts

- All take the form:
  - op reg, imm
  - op mem, imm
  - op reg, cl
  - op mem, cl
  - Last bit shifted out captured in CF (carry flag)

That's "CL" (low 8-bits of CX)
Why CL? ECX, counter register!

- Logical – shift bits left/right by amount
  - SHL: logical shift left (shift in zeros)
  - SHR: logical shift right (shift in zeros)

- Arithmetic
  - SAL: arithmetic shift left (same as SHL)
  - SAR: arithmetic shift right (shift in copy of sign bit)

- Rotate (same as shifts, last bit rotated out captured in CF)
  - ROL: rotate left (MSB becomes LSB)
  - ROR: rotate right (LSB becomes MSB)
  - RCL: rotate left w/carry (carry becomes LSB, MSB becomes carry)
  - RCR: rotate right w/carry (carry becomes MSB, LSB becomes carry)

Shift Examples

- **Arithmetic Shift:**

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

- **Rotate:**

```
1100 0001 0010 0011          mov    ax, 0C123H
1000 0010 0100 0111          rol    ax, 1       ;ax=8247H,CF=1
0000 0100 1000 1111          rol    ax, 1       ;ax=048FH,CF=1
0000 1001 0001 1110          rol    ax, 1       ;ax=091EH,CF=0
1000 0010 0100 0111          ror    ax, 2       ;ax=8247H,CF=1
1100 0001 0010 0011          ror    ax, 1       ;ax=C123H,CF=1
```

- **Rotate w/Carry:**

```
1100 0001 0010 0011          mov ax, 0C123H
1100 0001 0010 0011 (0)      clc            ;clear the carry flag (CF = 0)
1000 0010 0100 0110 (1)      rcl ax, 1      ;ax=8246H,CF=1
0000 0100 1000 1101 (1)      rcl ax, 1      ;ax=048DH,CF=1
0000 1001 0001 1011 (0)      rcl ax, 1      ;ax=091BH,CF=0
1000 0010 0100 0110 (1)      rcr ax, 2      ;ax=8246H,CF=1
1100 0001 0010 0011 (0)      rcr ax, 1      ;ax=C123H,CF=0
```

Fun Time!

- Reversing Bits:
  - How reverse all bits in register?
  - Combine shifts & rotates to reverse all bits in a register

- Lets implement it!
  - Read in an integer
  - Reverse value in register
  - Print result back to screen

Fun Time!

```
        mov   cl, 32
rloop:  shr   eax, 1 ; move low order bit in EAX to CF
        rcl   ebx, 1 ; shift the bit back into EBX, backwards!
        loop  rloop
```

Counting Bits Example
(two methods)

```
    mov    bl, 0       ; bl will contain the count of ON (“1”) bits
    mov    ecx, 32     ; ecx is the loop counter (32 bits to count)
count_loop:
    shl    eax, 1      ; shift bit into carry flag
    jnc    skip_inc    ; if CF == 0, goto skip_inc
    inc    bl
skip_inc:
    loop   count_loop  ; decrements ecx, if ecx!=0 goto count_loop
```

```
    mov    bl, 0       ; bl will contain the count of ON bits
    mov    ecx, 32     ; ecx is the loop counter
count_loop:
    shl    eax, 1      ; shift bit into carry flag
    adc    bl, 0       ; if CF == 1, bl = bl + 1
    loop   count_loop  ; decrements ecx, if ecx!=0 goto count_loop
```

Boolean Bit-Ops

- Instruction Format:
  - AND, OR, XOR:

```
op      reg, reg
op      reg, mem
op      reg, imm
op      mem, reg
op      mem, imm
```

  - NOT – negation (one’s compliment)

```
op      reg
op      mem
```

  - TEST – similar to AND, but does not store result (just sets flags)

```
op      reg, reg
op      reg, imm
op      mem, reg
op      mem, imm
```

FLAGs Register

- Recall what the flags represent:

Flag	Function	Description
CF	Carry Flag	Set if unsigned math result will not fit
OF	Overflow Flag	Set if signed math result will not fit
SF	Sign Flag	Set if math result is negative
ZF	Zero Flag	Set if result is zero
AC	Auxiliary Carry	Set if math result carries 2nd MSB to MSB
PF	Parity Flag	Set if LS byte in result has even # ones

AND, OR, XOR & Flags

- Logic operations AND, OR, XOR
  - Will clear OF & CF flags
  - Will set SF (if MSB is one, clears SF otherwise)
  - Will Set/Clear ZF appropriately

- What does this mean to you?
  - Do not rely on CF or OF to be preserved past logic operations
  - Can efficiently carry out operations dependent upon single bit values
    - If bit 27 of value stored in EAX is important to me:

```
AND    EAX, 0x8000000
JZ     target_label
```

    - or

```
TEST   EAX, 0x8000000
JZ     target_label
```

  - Note: don’t have to use immediate values, can use register, too

Boolean Bit-Ops

- Sometimes need to manipulate individual bits
  - Settings, flags, bit-packing, etc.
  - Other uses:

Turn on bit i	*OR* the number with 2^i (which is the binary number with just bit i on)
Turn off bit i	*AND* the number with the binary number with only bit i off. This operand is often called a *mask*
Complement bit i	*XOR* the number with 2^i

Bit Operations – Examples

```
1100 0001 0010 0011 mov   ax, C123H
1100 0001 0010 1011 or    ax, 8       ; turn on bit 3, ax = C12BH
1100 0001 0000 1011 and   ax, FFDFH   ; turn off bit 5, ax = C10BH
0100 0001 0000 1011 xor   ax, 8000H   ; invert bit 15, ax = 410BH
0100 1111 0000 1011 or    ax, 0F00H   ; turn on nibble, ax = 4F0BH
0100 1111 0000 0000 and   ax, FFF0H   ; turn off nibble, ax = 4F00H
1011 1111 0000 1111 xor   ax, F00FH   ; invert nibbles, ax = BF0FH
0100 0000 1111 0000 xor   ax, FFFFH   ; 1's complement, ax = 40F0H
```

Speculative Execution

```
; file: max.asm
%include "asm_io.inc"
segment .data

message1 db "Enter a number: ",0
message2 db "Enter another number: ", 0
message3 db "The larger number is: ", 0

segment .bss

input1  resd    1        ; first number entered

segment .text
        global  _asm_main
_asm_main:
        enter   0,0               ; setup routine
        pusha

        mov     eax, message1     ; print out first message
        call    print_string
        call    read_int          ; input first number
```

Speculative Execution

```
mov     [input1], eax

mov     eax, message2     ; print out second message
call    print_string
call    read_int          ; input second number (in eax)

xor     ebx, ebx          ; ebx = 0
cmp     eax, [input1]     ; compare second and first number
setg    bl                ; ebx = (input2 > input1) ?          1 : 0
neg     ebx               ; ebx = (input2 > input1) ? 0xFFFFFFFF : 0
mov     ecx, ebx          ; ecx = (input2 > input1) ? 0xFFFFFFFF : 0
and     ecx, eax          ; ecx = (input2 > input1) ?     input2 : 0
not     ebx               ; ebx = (input2 > input1) ?          0 : 0xFFFFFFFF
and     ebx, [input1]     ; ebx = (input2 > input1) ?          0 : input1
or      ecx, ebx          ; ecx = (input2 > input1) ?     input2 : input1

mov     eax, message3     ; print out result
call    print_string
mov     eax, ecx
call    print_int
call    print_nl
```

Endian-ness & BitOps

- Recall:
  - Intel x86 is little endian
  - Rest of the world is big endian
    - Example: TCP/IP headers are big endian

- What's an Assembly Writer to do?!
  bswap to the rescue!
  - Swaps byte order of any 32-bit or larger register

- Example:

```
bswap      edx          ;swap byte order for edx
```

- If want to swap bytes of 16-bit register, cannot do that!

```
bswap      dx        ;(illegal syntax)
```

- Use the XCHG instruction instead:

```
xchg     dh, dl        ;(do this instead)
```

Bit Tricks

- Some cool tricks:
  - Want to zero out a register?
    - XOR it with itself:

```
xor   EAX, EAX     ; set EAX to zero
```

  - Want to set all bits to one?
    - Two paths:

```
xor   EAX, EAX
not   EAX
```

    - Or:

```
or   EAX, 0FFFFH        (or   EAX, 0xFFFF)
```

  - Divide by 2:

    right shift (question: arithmetic or logical??)

  - Multiply by 2:

    left shift (arithmetic or logical??)

More Bit Tricks

- Insert a bit string into some destination:
  - Want to set bits 8-11 of EAX to 0110:

```
mov   EBX, 6
shl   EBX, 8
and   EAX, 0xFFFFF0FF
or    EAX, EBX
```

- Searching for a bit pattern (example, 4 bits):

```
        mov  ecx, 28        ; searching for 4 out of 32 bits (28 attempts)
        mov  dh, 0FH        ; mask for the compare bits (four ones)
        mov  al, <pattern>  ; search pattern loaded into al
        and  al, dh         ; mask unneeded (high four) bits in al
        mov  ebx, <source>  ; get source value
scan:   mov  dl, bl         ; copy low bits of EBX
        and  dl, dh         ; mask unneeded (high four) bits in dl
        cmp  al, dl         ; do we have a match?
        jz   match          ; if zero, match found!
        shr  ebx, 1
        loop scan           ; no match this time
match:
        <found it, do something cool>
```