Chapter 7

Integer Arithmetic

Shifting bits to the left

- To shift 1 bit to the left we use:
  - `SHL dest,1`
    - each bit is shifted one position to the left
    - the lsb (least significant bit) is filled with 0
    - the msb (most significant bit) is moved into CF (so the previous content of CF is lost)
    - dest can be either byte, word or dword
- Ex:
  - `mov bl,80h ; BX = 0080h`
  - `shl bl,1 ; BX = 0000h, CF=1` (only BL is affected)

Shifting multiple times to the left

- Two forms are permitted:
  - `SHL dest,CL` ; value in CL = number of shifts
  - `SHL dest, imm8` ; needs .286 or higher
- SHL affects SF, PF, ZF according to the result
- **CF contains the last bit shifted**
- OF = 1 iff the **last shift** changes the sign bit
  - `mov bh,82h ; BH = 1000 0010b`
  - `shl bh,2 ; BH = 0000 1000b, CF=0, OF=0`
- Hence overflows are signaled only for the last bit shifted
- SAL is the same instruction as SHL but used with the signed interpretation (more later)

Fast Multiplication

- Each left shift multiplies by 2 the operand for both signed and unsigned interpretations. Ex:
  - `mov ax,4 ; AX = 0004h`
  - `mov bx,-1 ; BX = FFFFh`
  - `shl ax,2 ; AX = 0010h = 16`
  - `shl bx,3 ; BX = FFF8h = -8`
- Multiplication by shifting is very fast. Try to factor your multiplier into powers of 2:
  - `BX * 36 = BX * (32 + 4) = BX*32 + BX*4`
  - So add (BX shifted by 5) to (BX shifted by 2)
Shifting bits to the right

- To shift to the right use either:
  - SHR dest, CL ;value of CL = number of shifts
  - SHR dest, imm8 ;must use .286 or higher
    - the msb of dest is filled with 0
    - the lsb of dest is moved into CF
- Each single-bit right shift divides the unsigned value by 2.
  Ex:
  - mov bh,13 ;BH = 0000 1101b = 13
  - shr bh,2 ;BH = 0000 0011b = 3 (div by 4), CF = 0
  - (the remainder of the division is lost)

Arithmetic Shift SAR

- Is needed to divide the signed value by 2:
  - SAR dest, CL ;value of CL = number of shifts
  - SAR dest, imm8 ;must use .286 or higher
    - the msb of dest is filled with its previous value (so the sign is preserved)
    - the lsb of dest is moved into CF
  - mov ah,-15 ;AH = 1111 0011b = -8
  - sar ah,1 ;AH = 1111 1000b = -8
  - the result is rounded to the smallest integer (-8 instead of -7...)
  - in contrast, shr ah,1 gives 0111 1000b = 78h

Rotate (without the CF)

- ROL rotates the bits to the left (same syntax)
  - CF gets a copy of the msb
  - ROR rotates the bits to the right (same syntax)
  - CF gets a copy of the lsb
  - CF and OF reflect the action of the last rotate

Examples of ROL

- mov ah,40h ;ah = 0100 0000b
- rol ah,1 ;ah = 0100 0001b, CF = 0
- rol ah,1 ;ah = 0100 0010b, CF = 1
- rol ah,1 ;ah = 0100 0011b, CF = 0

- mov ax,1234h ;ax = 0001 0010 0011 0100b
- rol ax,4 ;ax = 2341h
- rol ax,4 ;ax = 3412h
- rol ax,4 ;ax = 4123h
Rotate with CF

- RCL rotates to the left with participation of CF
- RCR rotates to the right with participation of CF

Ex: inverting the content of AL

- Ex: if AL = 1100 0001b, we want to have AL = 1000 0011b
  mov cx,8 ;number of bits to rotate
  start:
  shl al,1 ;CF = msb of AL
  rcr bl,1 ;push CF into msb of BL
  loop start ;repeat for 8 bits
  mov al,bl ;store result into AL

Application: Binary Output

- To display the binary number in BX:
  mov ah,2 ;display
  mov cx,16 ;16 chars
  start:
  rol bx,1 ;CF gets msb
  jc one ;if CF =1
  mov dl,'0'
  jmp disp
  one: mov dl,'1'
  disp: int 21h
  loop start

Integer Multiplication

- Contrary to addition, the multiplication operation depends on the interpretation:
  - no interpretation: FFh x 2h = ??
  - unsigned interp.: 255 x 2 = 510
  - signed interpret.: -1 x 2 = -2
- We thus have two different multiplication instructions:
  - MUL source ;for unsigned multiplication
  - IMUL source ;for signed multiplication
- Where source must be either mem or reg
Multiplication (cont.)

- Source is being multiplied by:
  - AL if source is of type byte
  - AX if source is of type word
  - EAX if source is of type dword
- The result of MUL/IMUL is stored in:
  - AX if source is of type byte
  - DX:AX if source is of type word
  - EDX:EAX if source is of type dword
- Hence, there is always enough storage to hold the result

Examples of MUL and IMUL

- Say that AX = 1h and BX = FFFFh, then:
  - Instruction Result DX AX CF/OF
    - mul bx 655350000 FFFF 0
    - imul bx -1 FFFF FFFF 0
- Say that AX = FFFFh and BX = FFFFh, then:
  - Instruction Result DX AX CF/OF
    - mul bx 4294836225 FFFE 0001 1
    - imul bx 1 0000 0001 0
- AL = 30h and BL = 4h, then:
  - Instruction ResultAH AL CF/OF
    - mul bl 192 00 C0 0
    - imul bl 192 00 C0 1
- AL = 80h and BL = FFh, then
  - Instruction ResultAH AL CF/OF
    - mul bl 326407F 80 1
    - imul bl 128 00 80 1
Integer Division

- Notation for integer division:
  - Ex: \( 7 \div 2 = 3 + 1 \)
  - dividend \( \div \) divisor = quotient + remainder
- We have 2 instructions for division:
  - DIV divisor ;unsigned division
  - IDIV divisor ;signed division
- The divisor must be reg or mem
- Convention for IDIV: the remainder has always the same sign as the dividend.
  - Ex: \(-5 \div 2 = -2 + -1 \) (not \(-3 + 1\))

Examples of DIV and IDIV

- DX = 0000h, AX = 0005h, BX = FFFEh:
  - Instruction          Quot. Rem.  AX  DX
    div  bx               0   5  0000  0005
    idiv bx              -2   1  FFFE  0001
- DX = FFFFh, AX = FFFBh, BX = 0002h:
  - Instruction          Quot. Rem.  AX  DX
    idiv bx            -2   -1  FFFE  FFFF
    div  bx          Divide Overflow

Division (cont.)

- The divisor determines what will hold the dividend, the quotient, and the remainder:
  - Divisor Dividend Quotient Remainder
    - byte AX AL AH
    - word DX:AX AX DX
    - dword EDX:EAX EAX EDX
- The effect on the flags is undefined
- We have a divide overflow whenever the quotient cannot be contained in its destination (AL if divisor is byte...)
  - execution then traps into INT 0h which displays a message on screen and returns control to DOS

Examples of DIV and IDIV (cont.)

- AX = 0007, BX = FFFEh:
  - Instruction          Quot. Rem.  AL  AH
    div  bl               0   7  00  07
    idiv bl            -3   1  FC  01
- AX = 00FBh, BX = 0CFFh:
  - Instruction          Quot. Rem.  AL  AH
    div  bl              0 251  00  FB
    idiv bl        Divide Overflow
Preparing for a division

- Recall that:
  - For a byte divisor: the dividend is in AX
  - For a word divisor: the dividend is in DX:AX
  - For a dword divisor: the dividend is in EDX:EAX

- If the dividend occupies only its least significant half (lsh) we must prepare its most significant half (msh) for a division
  - For DIV: the msh must be zero
  - For IDIV: the msh must be the sign extension of the lsh

Preparing for IDIV

- To fill the msh of the dividend with the sign extension of its lsh, we use:
  - CBW (convert byte to word): fills AH with the sign extension of AL
  - CWD (convert word to double word): fills DX with the sign extension of AX
  - CDQ (convert double to quad): fills EDX with the sign extension of EAX

- Sign extension (recall):
  - if AX = 8AC0h, then CWD will set DX to FFFh
  - if AX = 7F12h, then CWD will set DX to 0000h

Preparing for DIV or IDIV

- If AX and BX contain unsigned numbers, to perform AX / DX, you must do
  xor dx,dx ;to fill DX with 0
  div bx

- If AX and BX contain signed numbers, to perform AX / BX, you must do
  cwd ;to fill DX with sign extension of AX
  idiv bx

- Never assign the msh of the dividend to zero before performing IDIV