Logic Instruction Types

```
                        BITWISE LOGICAL
not     ax              ;1's Complement-Logical Invert
and     ax,     bx      ;Bitwise logical and operation
or      ax,     bx      ;Bitwise logical inclusive-or operation
xor     ax,     bx      ;Bitwise logical exclusive-or operation
test    ax,     fffh    ;Bitwise and but result discarded

                            SHIFT
shl     ax,     4       ;Logical shift left
sal     ax,     3       ;Arithmetic shift left
shr     ax,     4       ;Logical shift right
sar     ax,     3       ;Arithmetic shift right

                           ROTATE
rol     bx,     3       ;Rotate left
ror     cx,     4       ;Rotate right
rcl     ax,     1       ;Rotate left through carry
rcr     dx,     6       ;Rotate right through carry
```

Logic Instruction Types (386+)

```
                            SHIFT
shld    ax,     12      ;Double precision logical shift left
shrd    ax,     14      ;Double precision logical shift right

                          BIT TEST
bt      ax,     12      ;CF<--12th bit from right in ax
bts     bx,     8       ;CF<--10th bit of bx and bx[10]<--1
btr     cx,     1       ;CF<--1st bit in cx and cx[1]<--0
btc     dx,     2       ;CF<--2nd bit of dx and dx[2]<--dx[2]'

                          BIT SCAN
bsf     ax,     bx      ;ZF=1 if all bits in bx=0
                        ;else ZF=0 and ax gets index of first
                        ;set bit (1) starting from right (LSB) of bx
bsr     ax,     bx      ;ZF=1 if all bits in bx=0
                        ;else ZF=0 and ax gets index of first
                        ;set bit (1) starting from left (MSB) of bx
```

Bit Level Logic

```
and, or, xor, not, test, bt, btc, btc, btr, bts
```

- Affect Status Flags as Follows:
  1) Always Clears CF and OF
  2) SF, ZF, AF, PF Change to Reflect Result

- Common Usage:

```
and     ax,     ax      ;clear CF and OF

xor     ax,     ax      ;clear ax=CF=OF=PF=AF=SF=0 and ZF=1
                        ;does more than mov ax, 0h
                        ;faster than push 00h then popf
```

Masking Operations

```
        XXXX XXXX (unknown word)
(AND)   0000 1111 (mask word)
        0000 XXXX (result)
```

What if we wanted 1111 XXXX instead?

EXAMPLE: Convert ASCII to BCD to Binary

```
;First convert to BCD - change 3235h into 0025h
mov     bx,     3235h   ;bx<-- '25'
and     bx,     0f0fh   ;bx<--0205h
mov     dx,     bx      ;dx<--0205h
shl     bh,     4       ;bh<--20h
or      bl,     bh       ; bl = bh or bl = 20 or 05 = 25h
xor     bh,     bh      ;zero out bh, so bx = 0025 (BCD value)
;Now convert to binary - change 3235h into 0019h
mov     al,     dh      ;al<--02h
mov     cl,     10      ;cl<--0ah
mul     cl              ;ax = 2 * 0Ah = 14h (decimal value is 20)
add     al,     dl      ;al<--14h+05h=19h (decimal value is 25)
```

Bit Test Instruction, **`test`**

- Same as **and** But Result is Discarded
- Only Affects Flags (like **`cmp`**)
- Use **`test`** for Single Bit and **`cmp`** for Byte, Word
- ZF=1 if Tested Bit=0 and ZF=0 if Tested Bit=1

```
test    al,     1       ;XXXX XXXX (AND) 0000 0001
test    al,     128     ;XXXX XXXX (AND) 1000 0000
```

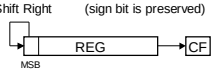
Shifts

`shl` - Logical Shift Left

`shr` - Logical Shift Right

`sal` - Arithmetic Shift Left (same as logical)

`sar` - Arithmetic Shift Right (sign bit is preserved)



Simple Arithmetic Using Shifts

```
;Compute (-3)*VALUE Using Only Shifts and Adds
mov     ax,     VALUE   ;ax <-- Word from memory with label VALUE
mov     bx,     ax      ;bx <-- Word from memory with label VALUE
shl     ax,     2       ;ax <-- 4*VALUE
add     ax,     bx      ;ax <-- 5*VALUE
shl     bx,     3       ;bx <-- 8*VALUE
sub     ax,     bx      ;ax <-- (-3)*VALUE
```

Double Precision Shifts

- 386+
- **`shld`** - Logical Shift Left
- **`shrd`** - Logical Shift Right
- Uses 3 Operands Instead of 2
- Example

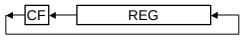
```
shrd    ax,     bx,     12      ;logical right shift of ax by 12
                                ;rightmost 12 bits of bx into
                                ;leftmost 12 bits of ax
```

- Contents of **`bx`** Remain *Unchanged !!!!!!!*

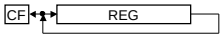
Rotates

`rol` - Rotate Left

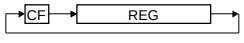
`rcl` - Rotate Through Carry Left



`ror` - Rotate Right



`rcr` - Rotate Through Carry Right



Example Using Rotates

```
;Multiply a 48-bit value in dx:bx:ax by 2
shl     ax,     1       ;ax <-- 2*ax
rcl     bx,     1       ;bx <-- 2*bx + CF(lsb)
rcl     dx,     1       ;dx <-- 2*dx + CF(lsb)

;End Result is dx:bx:ax <-- 2*(dx:bx:ax)
```

- Operand for Rotates and Shifts can be Either:
  1) Immediate Value
  2) Quantity in **`cl`**

String Scan Instruction, **`scas`**

- **`scasb`**, **`scasw`**, **`scasd`** (386+)
- Compares **`al`**, **`ax`**, **`eax`** with Memory Data
- Does an Integer Subtraction - Result Not Saved
- Generally Used With a REPEAT Prefix
- DF Controls Auto-increment/decrement
- Example:

```
mov     di,     OFFSET BLOCK    ;di <-- address of memory location BLOCK
cld                             ;DF <-- 0, auto-increment mode
mov     cx,     100             ;cx <-- 64h, initialize counter to 100
xor     al,     al              ;clear al
repne scasb                     ;test for 00h in location es:di
                                ;if es:di not equal to 00h then
                                ;  cx <-- cx - 1, di<-- di + 1, repeat
                                ;else if cx = 00h
                                ;  do not repeat test
                                ;else if es:di equals 00h
                                ;  ZF = 1, do not repeat test
```

Skip ASCII Space Character

```
lea     di,     STRING  ;di <-- offset of memory location labeled STRING
cld                     ;DF=0 auto-increment mode
mov     cx,     256     ;cx <-- ffh, initialize counter to 256
mov     al,     20h     ;al <-- ' ', an ASCII <space> Character
repe scasb              ;while es:di=20h, continue scanning
                        ;when cx=0 or es:di not equal 20h stop
                        ;after stopping cx contains offset from
                        ; STRING where first non-20h resides (if not 0)
```

Compare String Instruction, **cmps**

- **cmpsb**, **cmpsw**, **cmpsd** (386+)
- Compares 2 Sections of Memory
- Does an Integer Subtraction - Result Not Saved
- Generally Used With a REPEAT Prefix
- **si**, **di** Auto-increment/decrement Depending on DF
- Example: Test Two Strings for Equivalence

```
;Assume that ds and es are already set-up (NOTE:ds can equal es)
lea     si,     LINE    ;si gets offset of location labeled LINE
lea     di,     TABLE   ;di gets offset of location labeled TABLE
cld                     ;DF=0, auto-increment mode
moc     cx,     10      ;initialize counter register to 10
repe cmpsb              ;while ds:si=es:di decrement cx and incr. si, di
                        ;if cx=0 stop testing
                        ;after complete, if cx not equal 0, then
                        ;  strings do not match
```

Skip ASCII Space Character

```
lea     di,     STRING  ;di <-- offset of memory location labeled STRING
cld                     ;DF=0 auto-increment mode
mov     cx,     256     ;cx <-- ffh, initialize counter to 256
mov     al,     20h     ;al <-- ' ', an ASCII <space> Character
repe scasb              ;while es:di=20h, continue scanning
                        ;when cx=0 or es:di not equal 20h stop
                        ;after stopping cx contains offset from
                        ; STRING where first non-20h resides (if not 0)
```