Intel Assembly I – Arithmetic & Logic Operations

Arithmetic & Logic Operations

Arithmetic Operations

- Addition
- Subtraction
- Multiplication
- Division
- Comparison
- Negation
- Increment
- Decrement

Logic Operations

- AND
- OR
- XOR
- NOT
- shift
- rotate
- compare (test)

Arithmetic & Logic Operations

- Some bits of EFLAGS can change for arithmetic and logic instructions
  - Z (result zero)
  - C (carry out)
  - A (half carry out)
  - S (result positive)
  - P (result has even parity)
  - O (overflow occurred)

Addition

- Addition, Increment, Add-with-carry and Exchange-and-add

```
add  al, [ARRAY + esi]
inc  byte [edi]          ;Adds 1 to any reg/mem except seg
adc  ecx, ebx            ;Adds registers + Carry flag.
                          ;Used for adding 64 bit nums.
xadd ecx, ebx            ;ecx=ecx+ebx, ebx=original ecx.
```

Subtraction

- Subtraction, Decrement and Subtract-with-borrow

```
sub  eax, ebx          ;eax=eax-ebx
dec  edi
sbb  ecx, ebx          ;Subs registers - Carry flag.
```

- Comparison
  - The operands are subtracted, but the result is not stored; Changes only the flag bits.
  - Often followed with a conditional branch:

```
cmp  al, 10H
jae  LABEL1            ;Jump if equal or above.
jbe  LABEL2            ;Jump if equal or below.
cmpxchg  ecx, edx      ;if ecx==eax, eax=edx else eax=ecx
```

INC and DEC

- INC
  - `INC reg`
  - Increment reg value
  - Same to ADD reg, 1
- DEC
  - `DEC reg`
  - Decrement reg value
  - Same to SUB reg, 1

CMP

```
        ; If the first input is larger output 1
        ; If the second input is larger output 2
        ; The program uses subtraction:
        ; B < A is true if and only if B − A is negative.
        ;
        MOV EDX, 0
        IN EAX, [DX]
        MOV EBX, EAX     ;The first input is now in EBX
        IN EAX, [DX]     ;The second input is now in EAX.
        CMP EBX, EAX     ; This is first − second.
        JL SIB           ; Second is Bigger
        MOV EAX, 1       ; Otherwise First is Bigger
        JMP END          ; Don't drift into the other case!
SIB:    MOV EAX, 2       ;
END:    MOV EDX, 1       ; Either way now EAX is ready.
        OUT [DX], EAX
        RET
```

Program 4.6

Multiplication and Division

- Multiplication and Division
  - mul/div: ***Unsigned***.
  - imul/idiv: ***Signed*** integer multiplication/division.
  - al always holds the *multiplicand* (or ax or eax).
  - Result is placed in ax (or dx and ax or edx or eax).

```
mul  bl                 ;ax=al*bl (unsigned)
imul  bx                ;dx|ax=ax*bx (signed)
imul  cx, dx, 12H       ;Special, cx=dx*12H (signed only)
mul  ecx                ;edx|eax=eax*ecx
```

Multiplication and Division

- ***C*** and ***O*** bits are cleared if most significant 8 bits of the 16-bit product are zero (result of an 8-bit multiplication is an 8-bit result).
- Division by zero and overflow generate errors.
- Overflow occurs when a small number divides a large dividend.

```
div  cl                ;ah|al=ax/cl, unsigned quotient
                         ; in al, remainder in ah
idiv  cx               ;dx|ax=(dx|ax)/cx
```

AND, OR, and XOR

- Allow bits to be set, cleared and complemented
  - Commonly used to control I/O devices.
  - Logic operations always clear the *carry* and *overflow* flags.

- ***AND***: 0 ***AND*** anything is 0.
  - Commonly used with a MASK to clear bits

```
XXXX XXXX   Operand
0000 1111   Mask          and al, bl    ;al=al AND bl
-----------
0000 XXXX   Result
```

AND, OR, and XOR

- OR: 1 OR anything is 1.
  - Commonly used with a MASK to set bits

```
XXXX XXXX   Operand
0000 1111   Mask          or eax, 10    ;eax=eax OR 0000000AH
-----------
XXXX 1111   Result
```

- XOR: Truth table: 0110.
  - Commonly used with a MASK to complement bits:

```
XXXX XXXX   Operand
0000 1111   Mask          xor ah, ch    ;ah=ah XOR ch
-----------
XXXX XXXX   Result
```

TEST

- ***TEST***: Operates like the AND but doesn't effect the destination.
  - Sets the Z flag to the ***complement*** of the bit being tested:

```
test al, 4              ;Tests bit 2 in al -- 00000100
jz LABEL                ;Jump to LABEL if bit 2 is zero.
```

- ***BT***: Test the bit, ***BTC***: Tests and complements...

- ***NOT*** (logical one's complement)
- ***NEG*** (arithmetic two's complement - sign of number inverted)

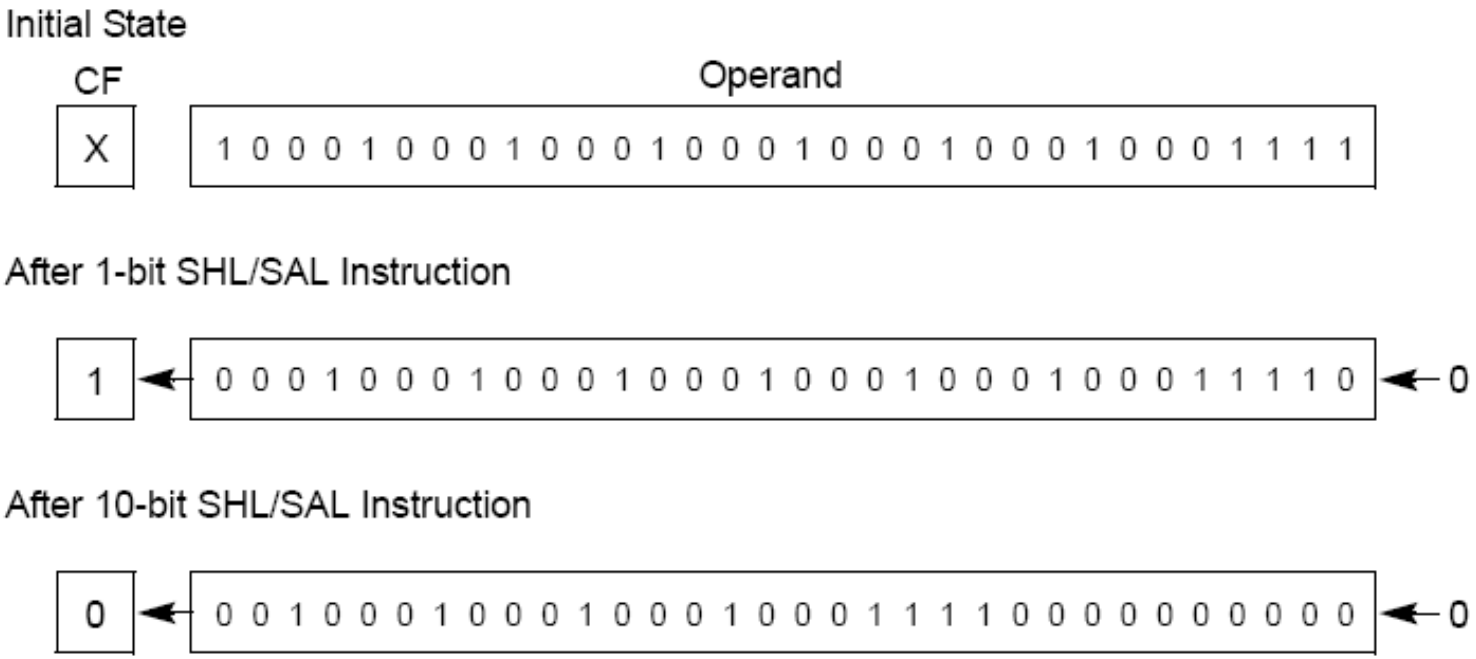
```
not ebx
neg TEMP
```

Shift

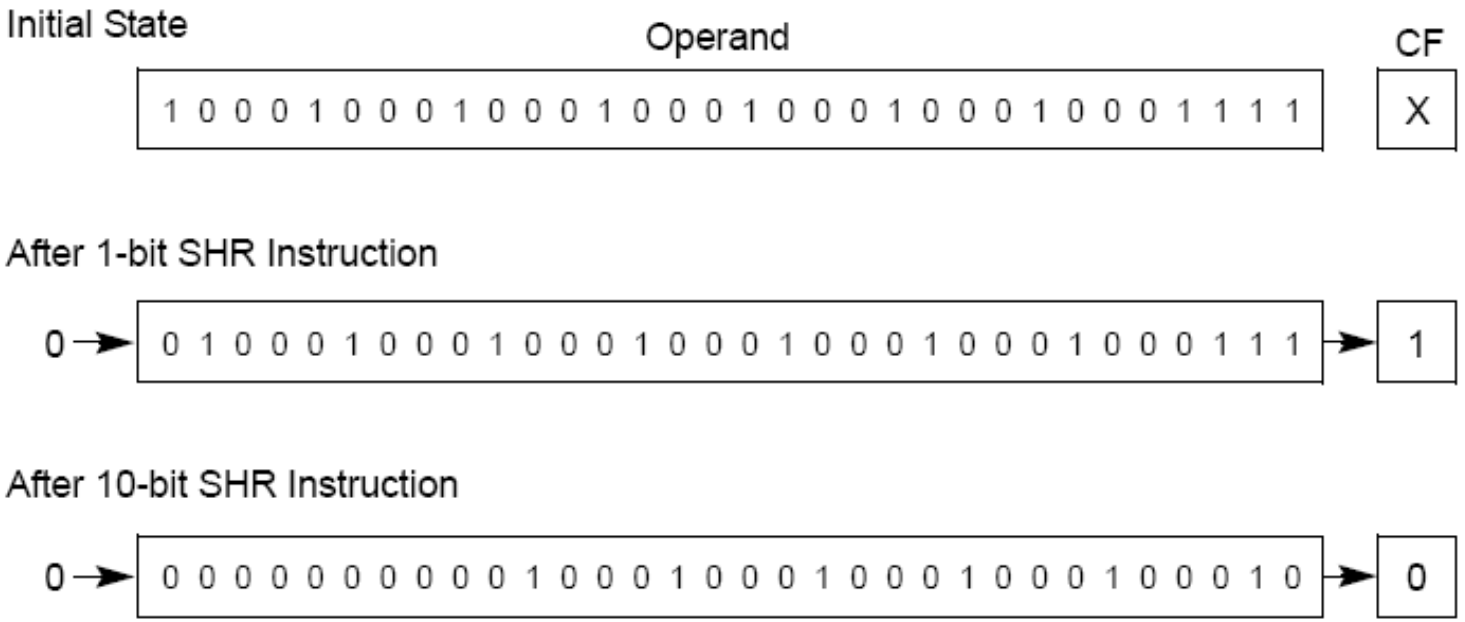
- Shift: Logical shifts insert 0, arithmetic right shifts insert sign bit.
  - Shift 대상 오퍼랜드와 shift count 오퍼랜드로 구성; count 오퍼랜드는 최대 5bit만 유의함
  - shift 된 비트는 CF로 들어감
- SHL과 SAL은 기본적으로 동작이 동일
  - unsigned multiplication of the destination operand by powers of 2
  - do not work for signed values
- SHR
  - unsigned division of the destination operand by powers of 2.
- SAR
  - 빈자리는 원래 값의 sign bit로 채움
  - signed division of the destination operand by powers of 2.

```
shl eax, 1      ;eax is logically shifted left 1 bit pos.
sar esi, cl     ;esi is arithmetically shifted right
```

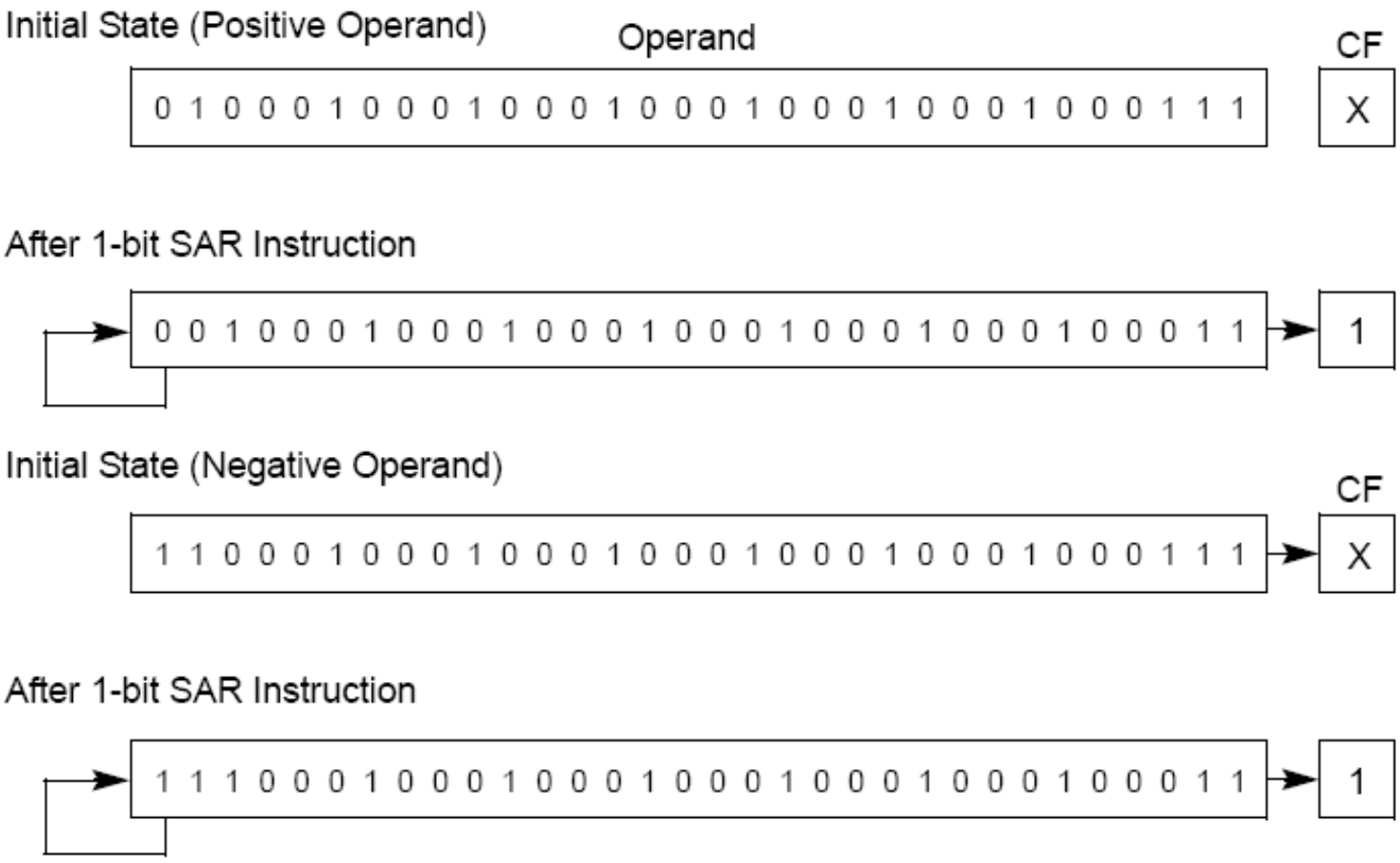
SHL/SAL Instruction Operation



SHR Instruction Operation



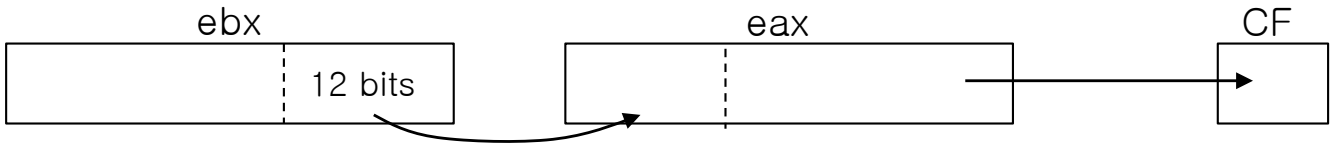
SAR Instruction Operation



Double Precision Shift

- shld / shrd

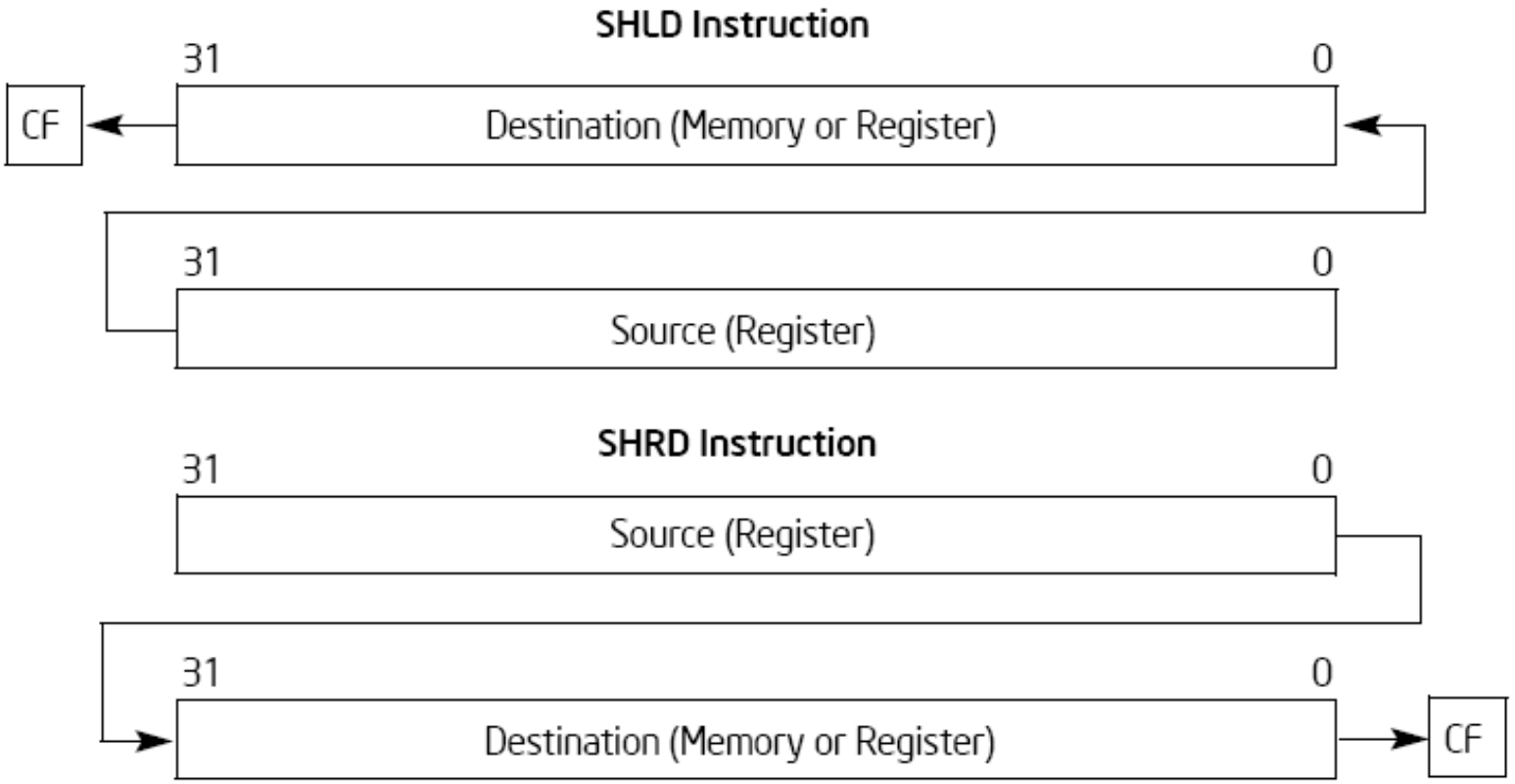
```
shdr eax, ebx, 12  ;eax shifted right by 12 and filled
                   ;from the left with the right
                   ;12 bits of ebx.
```




```
shdl ax, bx, 14
```

- shld dst, src, count
  - dst: reg / mem
  - src : register-only
  - count : CL or 8-bit immediate

SHLD and SHRD Instruction Operations



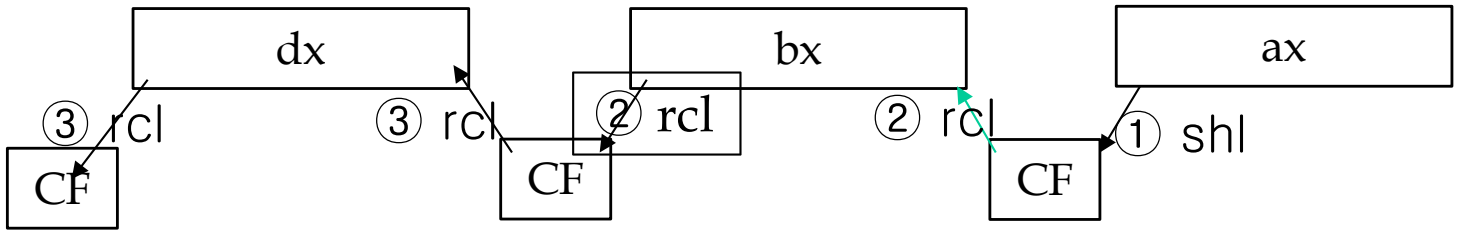
Rotate

- Rotate: Rotates bits from one end to the other or through the carry flag.

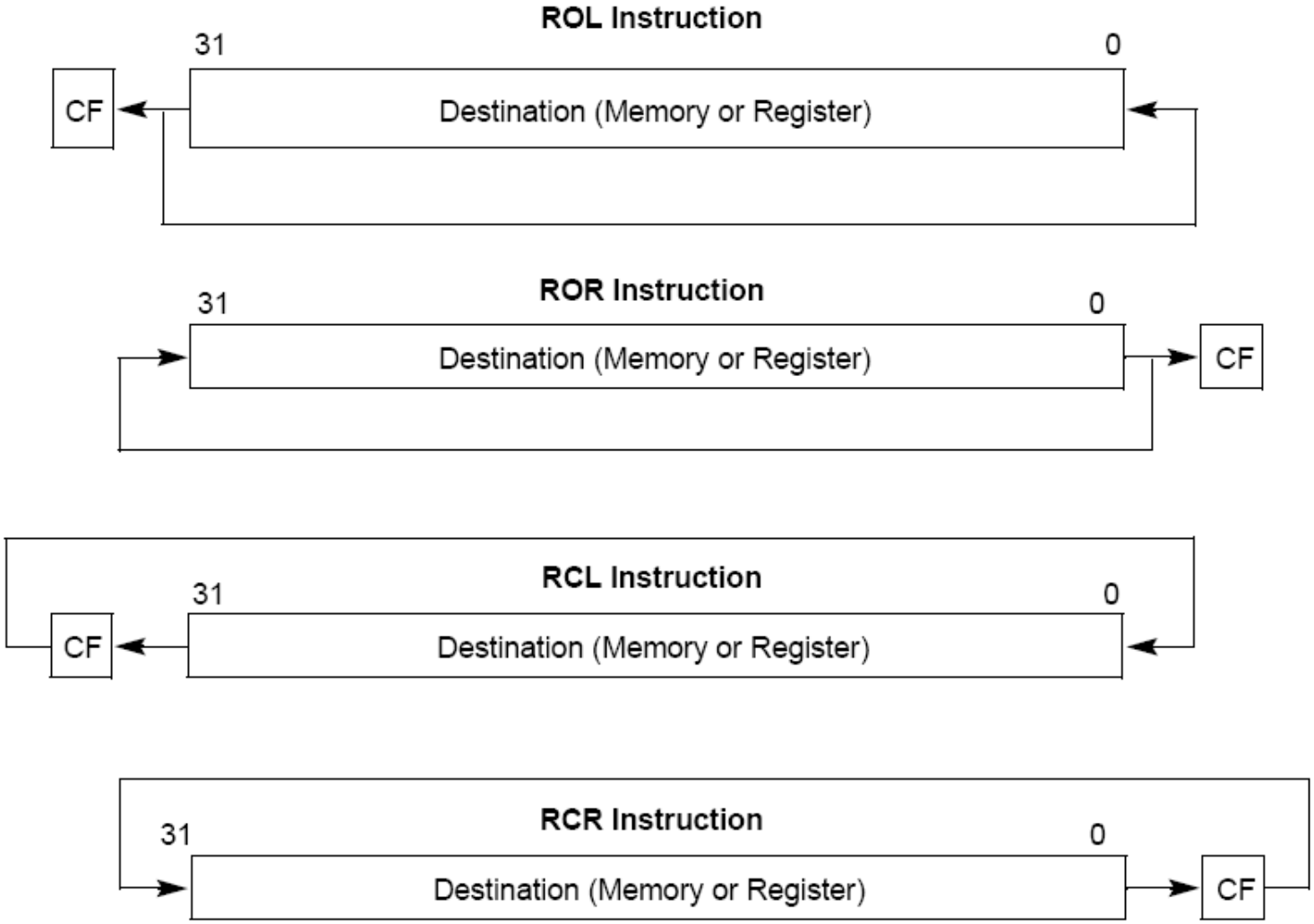
```
rol si, 14      ;si rotated left by 14 places.
rcr bl, cl      ;bl rotated right cl places through carry.
```

- Commonly used to operate on numbers wider than 32-bits:

```
shl ax, 1       ;Original 48-bit number in dx, bx and ax.
                 ;Shift ax left 1 binary place.
rcl bx, 1       ;Rotate carry bit from previous shl into
                 ;low order bit of bx.
rcl dx, 1      ;Rotate carry bit from previous rcl in dx.
```



ROL, ROR, RCL, and RCR Instruction Operations



Bit/String Scan

- Bit Scan Instruction (80386 and up):
  - Scan through an operand searching for a 1 bit.
  - Zero flag is set if a 1 bit is found, position of bit is saved in destination register.

```
bsl ebx, eax  ;eax scanned from the left.
bsr bl, cl    ;cl scanned from the right.
```

- String Scan Instructions:
  - ***scasb/w/d*** compares the al/ax/eax register with a byte block of memory and sets the flags.
  - Often used with ***repe*** and ***repne***
  - ***cmpsb/w/d*** compares 2 sections of memory data.