ARITHMETIC AND LOGIC INSTRUCTIONS

Prima Dewi Purnamasari
Sistem Berbasis Komputer—Computer Based Systems
Computer Engineering Study Program
Department of Electrical Engineering
Universitas Indonesia

Introduction

- The arithmetic instructions include
  - addition, subtraction, multiplication, division, comparison, negation, increment, and decrement.
- The logic instructions include
  - AND, OR, Exclusive-OR, NOT, shifts, rotates, and the logical compare (TEST).

Addition

- ADD – adding one value with another
- ADC - **add-with-carry**
- Adding memory-to-memory and segment register are not allowed.
  - segment registers can only be copied (MOV), pushed, or popped
- INC – Increment, special addition that adds 1 to a number.

- When arithmetic and logic instructions execute, contents of the flag register change.
  - interrupt, trap, and other flags do not change
- Any ADD instruction modifies the contents of the sign, zero, carry, auxiliary carry, parity, and overflow flags.

ADD

- ***Register Addition***
  - *ADD AX, BX*
  - *ADD AX, CX*
  - ADD AX, DX

- ***Immediate Addition***

  Immediate addition is employed whenever constant or known data are added.
  - MOV DL, 12H
  - ADD DL, 33H

Memory-to-Register Addition

Moves memory data to be added to the AL (and other) register.

```
MOV DI, OFFSET NUMB
MOV AL, 0
ADD AL, [DI]
ADD AL, [DI+1]
```

Array Addition

Memory arrays are sequential lists of data

```
MOV AL, 0               ; clear sum
MOV SI, 3               ; address element 3
ADD AL,ARRAY[SI]        ; add element 3
ADD AL,ARRAY[S+2]        ; add element 5
ADD AL,ARRAY[SI+4]       ; add element 7
```

INC

- The INC instruction adds 1 to any register or memory location, except a segment register.
- The size of the data must be described by using the BYTE PTR, WORD PTR, DWORD PTR, or QWORD PTR directives.
- The assembler program cannot determine if the INC [DI] instruction is a byte-, word-, or doubleword-sized increment.

INC BL	; BL = BL + 1
INC EAX	; EAX = EAX + 1
INC BYTE PTR[BX]	;increments the byte data at the data ;segment addressed by BX
INC WORD PTR[SI]	;increments the word data at the data ;segment addressed by BX
INC DATA1	; adds 1 to the data at memory ; location DATA1

Similar program using INC and ADD

```
MOV DI, OFFSET NUMB
MOV AL, 0
ADD AL, [DI]
ADD AL, [DI+1]
```

```
MOV DI, OFFSET NUMB
MOV AL, 0
ADD AL, [DI]
INC DI
ADD AL, [DI]
```

ADC

- ADC - adds the bit in the carry flag (C) to the operand data.

ADC AX, CX ;AX + CX + carry

ADC BL, DL ;BL + DL + carry

ADC BX,[BP+2] ; BX + data in stack segment + carry

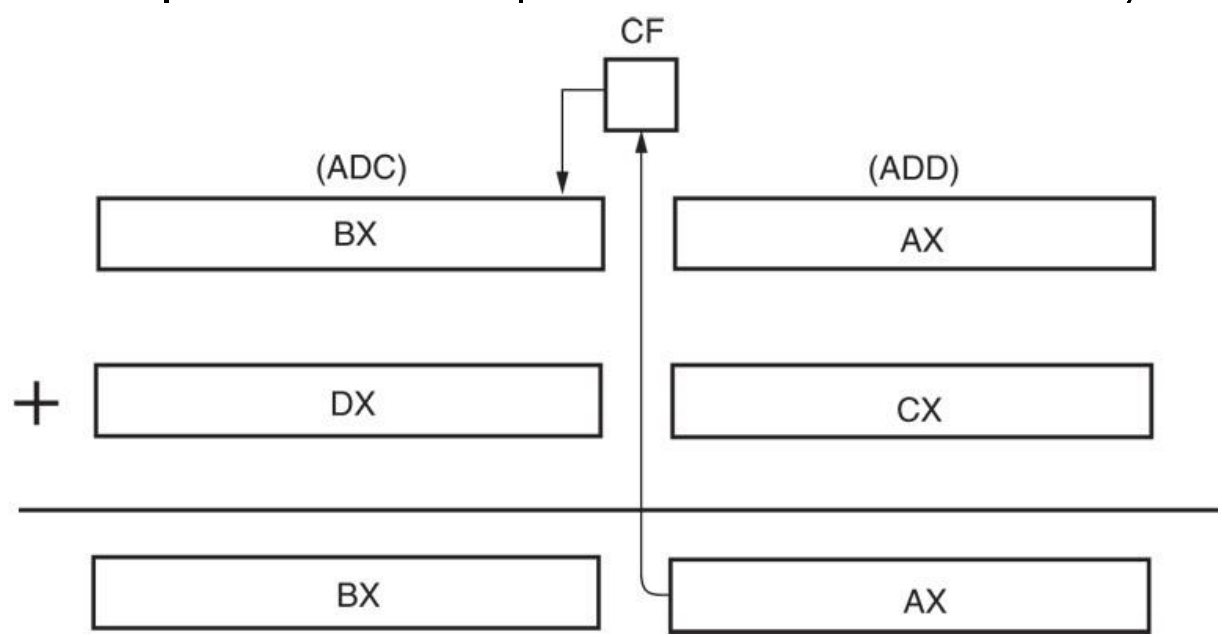
  - mainly appears in software that adds numbers wider than 16 or 32 bits in the 80386–Core2
  - like ADD, ADC affects the flags after the addition

Addition of 32-bit numbers in 8086-80286

- Cannot be easily performed without adding the carry flag bit because the 8086–80286 only adds 8- or 16-bit numbers

Add BX|AX + DX|CX

```
ADD   AX, CX
ADC   BX, DX
```



Subtraction

- Many forms of subtraction (**SUB**) appear in the instruction set.
  - these use any addressing mode with 8-, 16-, or 32-bit data
  - a special form of subtraction (decrement, or **DEC**) subtracts 1 from any register or memory location
- Numbers that are wider than 16 bits or 32 bits must occasionally be subtracted.
  - the **subtract-with-borrow instruction** (SBB) performs this type of subtraction

SUB

- ***Register Subtraction***

  After each subtraction, the microprocessor modifies the contents of the flag register.

  - flags change for most arithmetic/logic operations

- ***Immediate Subtraction***

  The microprocessor also allows immediate operands for the subtraction of constant data.

- ***Decrement Subtraction***

  Subtracts 1 from a register/memory location.

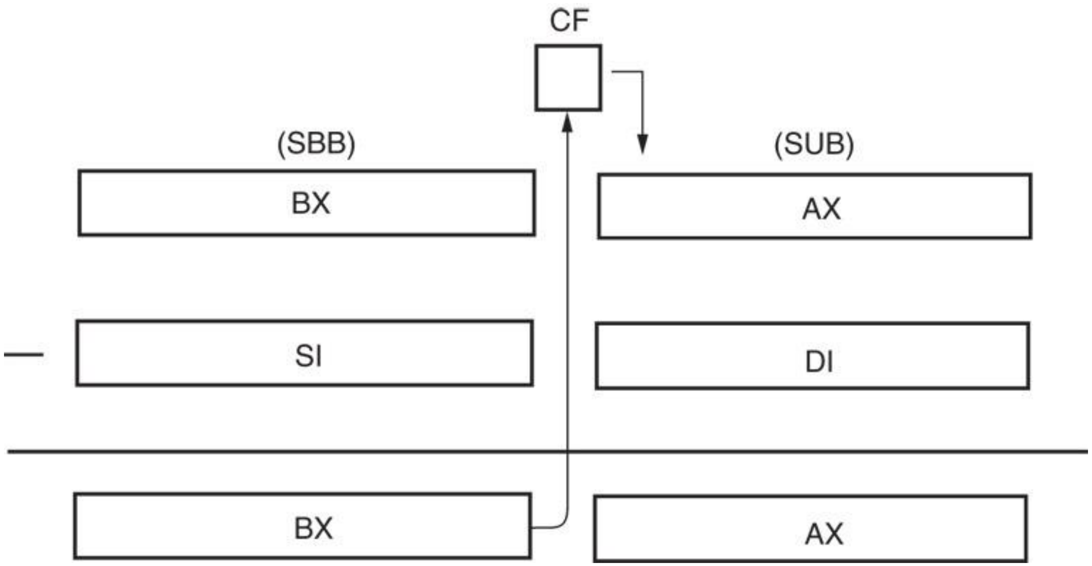
Examples

SUB BL, AL
SUB AX, SP
SUB DH, 6FH
SUB AX, 0CCABH
SUB [DI], CH
SUB AH, TEMP
SUB AX, 12

DEC BH
DEC CX
DEC BYTE PTR[DI]
DEC WORD PTR[BP]
DEC NUMB

SBB

- A subtraction-with-borrow (SBB) instruction functions as a regular subtraction, except that the carry flag (C), which holds the borrow, also subtracts from the difference.
  - most common use is subtractions wider than 16 bits in the 8086–80286 microprocessors or wider than 32 bits in the 80386–Core2.
  - wide subtractions require borrows to propagate through the subtraction, just as wide additions propagate the carry



SUB AX, DI
SBB BX, SI

EXAMPLES

```
SBB AH, AL
SBB AX, BX
SBB CL, 2
SBB BYTE PTR[DI], 3
SBB [DI], AL
SBB DI, [BP+2]
```

Comparison

- The comparison instruction (CMP) is a subtraction that changes only the flag bits.
  - destination operand never changes
- Useful for checking the contents of a register or a memory location against another value.
- A CMP is normally followed by a conditional jump instruction, which tests the condition of the flag bits.

CMP

- Examples

CMP CL, BL

CMP AX, SP

CMP AX, 2000H

CMP [DI], CH

- Examples

CMP CL, [BP]

CMP AH, TEMP

CMP DI, TEMP[BX]

CMP AL, [DI+SI]

```
CMP AL, 10H        ; AL – 10H
JAE SUBER          ; jump if above or equal (C=0)
```

Example – collect capital and small letters

```
data segment
   char_string db 'AK1as4BNZSdfg?23'
   capital db '                  '
   small  db '                  '
ends
code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

    mov di, offset capital
    mov si, offset small
    mov bx, offset char_string
   mov cx,0010h
again:
    mov al, [bx]
    cmp al, 41h
    jb continue       ;not a letter
    cmp al,7Ah
    ja continue       ;not a letter
    cmp al,5Ah
    jbe lettercapital
    cmp al, 61h
    jb continue       ;not a letter
    mov [si], al      ; small letter
    inc si
    jmp continue
lettercapital:
    mov ds[di], al
    inc di
continue:
    inc bx
    loop again
```

MULTIPLICATION AND DIVISION

- Earlier 8-bit microprocessors could not multiply or divide without the use of a program that multiplied or divided by using a series of shifts and additions or subtractions.
  - manufacturers were aware of this inadequacy, they incorporated multiplication and division into the instruction sets of newer microprocessors.

8-Bit Multiplication

- With 8-bit multiplication, the multiplicand is always in the AL register, signed or unsigned.
  - multiplier can be any 8-bit register or memory location
- Immediate multiplication is not allowed unless the special signed immediate multiplication instruction appears in a program.
- The multiplication instruction contains one operand because it always multiplies the operand times the contents of register AL.

Multiplication

- Performed on bytes, words, or doublewords,
  - can be signed (IMUL) or unsigned integer (MUL)
- Product after a multiplication always a double-width product.
  - two 8-bit numbers multiplied generate a 16-bit product; two 16-bit numbers generate a 32-bit;
    two 32-bit numbers generate a 64-bit product
  - in 64-bit mode of Pentium 4, two 64-bit numbers are multiplied to generate a 128-bit product

Multiplication

- Examples 8-bit multiplication instructions

MUL CL ; ALxCL unsigned product in AX

IMUL DH ; ALxDH signed product in AX

IMUL BYE PTR[BX] ; ALx(Mem by BX), signed product in AX

MUL TEMP ; AL xTEMP ; unsigned product in AX

Example

- Code for DX = BL x CL

```
MOV AL, CL
MUL BL
MOV DX, AX
```

16-Bit Multiplication

- Word multiplication is very similar to byte multiplication.
- AX contains the multiplicand instead of AL.
  - 32-bit product appears in DX–AX instead of AX
- The DX register always contains the most significant 16 bits of the product; AX contains the least significant 16 bits.
- As with 8-bit multiplication, the choice of the multiplier is up to the programmer.

16-Bit Multiplication

```
MUL CX              ;AX multiplied by CX; unsigned result in DX-AX
IMUL DI             ;AX by DI; signed result in DX-AX
MUL WORD PTR[SI]    ;AX by mem; unsignedproduct in DX-AX
```

A Special Immediate 16-Bit Multiplication

- 80186 - Core2 processors can use a special version of the multiply instruction.
  - immediate multiplication must be signed;
  - instruction format is different because it contains three operands

Division

- Occurs on 8- or 16-bit and 32-bit numbers depending on microprocessor.
  - signed (IDIV) or unsigned (DIV) integers
- Dividend is always a double-width dividend, divided by the operand.
- There is no immediate division instruction available to any microprocessor.
- In 64-bit mode Pentium 4 & Core2, divide a 128-bit number by a 64-bit number.

- A division can result in two types of errors:
  - attempt to divide by zero
  - other is a divide overflow, which occurs when a small number divides into a large number
    - If AX=3000, division by 2 results in quotient of 1500 which don't fit into AL (8 bits).

- In either case, the microprocessor generates an interrupt if a divide error occurs.

8-Bit Division

- Uses AX to store the dividend divided by the contents of any 8-bit register or memory location.
- Quotient moves into AL after the division with AH containing a whole number remainder.
  - quotient is positive or negative; remainder always assumes sign of the dividend; always an integer
- Numbers usually 8 bits wide in 8-bit division .
  - the dividend must be converted to a 16-bit wide number in AX ; accomplished differently for signed and unsigned numbers
- CBW (**convert byte to word**) instruction performs this conversion.

- DIV CL ;AX is divided by CL, unsigned quotient is in AL,
;and the unsigned remainder is in AH
- IDIV BL ;AX is divided by BL, signed quotient is in AL,
;and the signed remainder is in AH
- DIV BYTE PTR[BP] ;AX is divided by the byte content of
;the stack segment memory location addresed by
;BP the unsigned quotient is in AL and the
;unsigned remainde is in AH

EXAMPLE

```
MOV AL, NUMB        ;get NUMB
MOV AH, 0           ; zero extend
DIV   NUMB1         ;divide  AX by NUMB1
MOV ANSQ, AL        ;save quotient
MOV ANSR, AH        ;save remainder
```

16-Bit Division

- Sixteen-bit division is similar to 8-bit division
  - instead of dividing into AX, the 16-bit number is divided into DX–AX, a 32-bit dividend
- As with 8-bit division, numbers must often be converted to the proper form for the dividend.
  - if a 16-bit unsigned number is placed in AX, DX must be cleared to zero
- CWD (**convert word to double word**) instruction performs this conversion.

EXAMPLE

```
MOV AX, -100        ; load AX -100
MOV CX, 9           ; load CX 9
CWD                 ;sign extend -100 in DX-AX
IDIV CX             ;signed quotient in AX, signed remainder in
                    ;DX
```

The Remainder

- Could be used to round the quotient or dropped to truncate the quotient.
- If division is unsigned, rounding requires the remainder be compared with half the divisor
  to decide whether to round up the quotient

```
    DIV BL
    ADD AH, AH
    CMP AH, BL
    JB NEXT
    INC AL
NEXT:
```

BCD and ASCII Arithmetic

- The microprocessor allows arithmetic manipulation of both BCD (**binary-coded decimal**) and ASCII (**American Standard Code for Information Interchange**) data.
- BCD operations occur in systems such as point-of-sales terminals (e.g., cash registers) and others that seldom require complex arithmetic.

BCD Arithmetic

- Two arithmetic techniques operate with BCD data: addition and subtraction.
- DAA (**decimal adjust after addition**) instruction follows BCD addition,
- DAS (**decimal adjust after subtraction**) follows BCD subtraction.
- The adjustment instructions work with AL register

DAA

- DAA follows the ADD or ADC instruction to adjust the result into a BCD result.
- Add two 4-digit BCD numbers in DX and BX and store the result in CX.

```
MOV DX, 1234H      ; load 1234 BCD
MOV BX, 3099H      ; load 3099 BCD
MOV AL, BL         ; sum BL and DL
ADD AL, DL         ;AL = CD
DAA                ;AL = 33; C=1
MOV CL,AL          ; store result in CL
MOV AL, BH         ; add the high order bytes
ADC AL, DH         ;AL = 43
DAA
MOV CH,AL          ; store the result in CH , CX = 4333H
```

DAS

- Functions as does DAA instruction, except it follows a subtraction instead of an addition.

```
MOV DX, 1234H      ; load 1234 BCD
MOV BX, 3099H      ; load 3099 BCD
MOV AL, BL
SUB AL, DL ;subtract DL from BL
DAS
MOV CL,AL          ; store result in CL
MOV AL, BH         ; subtract the high order bytes
SBB AL, DH
DAS
MOV CH,AL          ; store the result in CH
```

ASCII Arithmetic

- ASCII arithmetic instructions function with coded numbers, value 30H to 39H for 0–9.
- Four instructions in ASCII arithmetic operations:
  - AAA (**ASCII adjust after addition**)
  - AAS (**ASCII adjust after subtraction**)
  - AAD (**ASCII adjust before division**)
  - AAM (**ASCII adjust after multiplication**)
- These instructions use register AX as the source and as the destination.

ASCII Arithmetic

Example 1

34H = 00110100B

35H = 00110101B +

69H = 01101001B

Should be 09H

Ignore 6

Example 2

36H = 00110110B

37H = 00110111B +

6DH = 01101101B

Should be 13H → AH=01 AL=03

Ignore 6 and add 6 to D

- The **aaa instruction performs these adjustments to** the byte in AL register

ASCII Arithmetic

The **aaa instruction works as follows:**

* If the least significant four bits in AL are > 9 or
  if AF =1, it adds 6 to AL and 1 to AH.
    Both CF and AF are set
* In all cases, the most significant four bits in AL are cleared
* Example:

```
sub AH,AH        ; clear AH
mov AL,'6'       ;AL := 36H
add AL,'7'       ;AL := 36H+37H = 6DH
aaa              ;AX := 0103H
or AX,3030H      ;AX := 3133H
```

BASIC LOGIC INSTRUCTIONS

- Logic operations provide binary bit control in low-level software (control the I/O devices in a system).
  - allow bits to be set, cleared, or complemented

- Include AND, OR, Exclusive-OR, and NOT.
  - also TEST, a special form of the AND instruction
  - NEG, similar to the NOT instruction

- Logic operations always clear the carry and overflow flags
  - other flags change to reflect the result

AND

- Performs logical multiplication
- AND clears bits of a binary number.
  - called **masking**
- AND uses any mode except memory-to-memory and segment register addressing.
- An ASCIInumber can be converted to BCD by using AND to mask off the leftmost four binary bit positions.

A	B	T
0	0	0
0	1	0
1	0	0
1	1	1

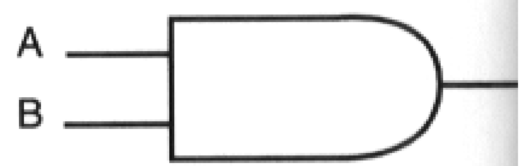


Figure 5–4 The operation of the AND function showing how bits of a number are cleared to zero.

	x x x x x x x x	Unknown number
•	0 0 0 0 1 1 1 1	Mask
	0 0 0 0 x x x x	Result

Example Masking with AND

- Example masked bits in BX = 3135H → this is ASCII for 15
- To clear byte 2 and byte 4 of a value:

$$\left.\begin{matrix} 0000 & 1111 & 0000 & 1111 \\ 0 & F & 0 & F \end{matrix}\right\} \text{Mask value}$$

  - AND with 0F0FH
- The result = 0105H → this is a BCD number

EXAMPLE 5–26

```
0000 BB 3135        MOV   BX,3135H        ;load ASCII
0003 81 E3 0F0F     AND   BX,0F0FH        ;mask BX
```

AND variations

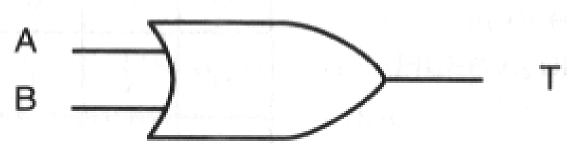
TABLE 5–14 AND instructions

Assembly Language	*Operation*
AND AL,BL	AL = AL AND BL
AND CX,DX	CX = CX AND DX
AND ECX,EDI	ECX = ECX AND EDI
AND CL,33H	CL = CL AND 33H
AND DI,4FFFH	DI = DI AND 4FFFH
AND ESI,34H	ESI = ESI AND 00000034H
AND AX,[DI]	AX is ANDed with the word contents of the data segment memory location addressed by DI
AND ARRAY[SI],AL	The byte contents of the data segment memory location addressed by the sum of ARRAY plus SI is ANDed with AL; the result moves to memory
AND [EAX],CL	CL is ANDed with the byte contents of the data segment memory location addressed by EAX; the result moves to memory

OR - *Inclusive-OR* function

- Performs logical addition
- OR uses any mode except memory-to-memory and segment register addressing.
- OR function may be used to set bits of a number

A	B	T
0	0	0
0	1	1
1	0	1
1	1	1




```
  x x x x  x x x x   Unknown number
+ 0 0 0 0  1 1 1 1   Mask
-------------------
  x x x x  1 1 1 1   Result
```

Example Masking with OR

- Example: masked bits in AX = 0305 → this is BCD for 35
  - On the example below 0305 is the result of 5 x7 followed by AAM instruction
- To convert the number to ASCII, make byte 2 and 4 into 3. thus the mask is 3030H
  - OR with 3030H
- The result = 3335H → this is ASCII for 35

EXAMPLE 5–27

```
0000   B0 05        MOV   AL,5         ;load data
0002   B3 07        MOV   BL,7
0004   F6 E3        MUL   BL
0006   D4 0A        AAM                ;adjust
0008   0D 3030      OR    AX,3030H     ;to ASCII
```

OR variations

TABLE 5–15 OR instructions

Assembly Language	*Operation*
OR AH,BL	AH = AH OR BL
OR SI,DX	SI = SI OR DX
OR EAX,EBX	EAX = EAX OR EBX
OR DH,0A3H	DH = DH OR A3H
OR SP,990DH	SP = SP OR 990DH
OR EBP,10	EBP = EBP OR 0000000AH
OR DX,[BX]	DX is ORed with the word contents of the data segment memory location addressed by BX
OR DATES[DI+2],AL	The byte contents of the data segment memory location addressed by the sum of DATES, DI, and 2 are ORed with AL

Exclusive-OR

- XOR uses any addressing mode except memory-memory and segment register addressing.
- Exclusive-OR is useful if some bits of a register or memory location must be inverted.
- A common use for the Exclusive-OR instruction is to clear a register to zero

just part of an unknown quantity can be inverted by XOR.

when a 1 Exclusive-ORs with X, the result is X

if a 0 Exclusive-ORs with X, the result is X

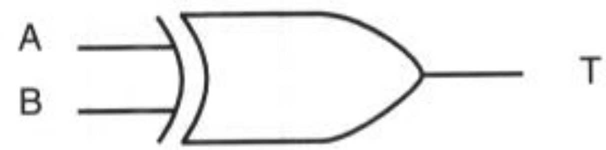
$$
\begin{array}{rl}
x\,x\,x\,x\;\;x\,x\,x\,x & \text{Unknown number} \\
\oplus\, 0\,0\,0\,0\;\;1\,1\,1\,1 & \text{Mask} \\
\hline
x\,x\,x\,x\;\;\bar{x}\,\bar{x}\,\bar{x}\,\bar{x} & \text{Result}
\end{array}
$$

X-OR

FIGURE 5–7 (a) The truth table for the Exclusive-OR operation and (b) the logic symbol of an Exclusive-OR gate

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0

(a)



(b)

FIGURE 5–8 The operation of the Exclusive-OR function showing how bits of a number are inverted

$$\begin{array}{rl} x\,x\,x\,x\;\;x\,x\,x\,x & \text{Unknown number} \\ \oplus\,0\,0\,0\,0\;\;1\,1\,1\,1 & \text{Mask} \\ \hline x\,x\,x\,x\;\;\bar{x}\,\bar{x}\,\bar{x}\,\bar{x} & \text{Result} \end{array}$$

XOR variations

TABLE 5–16 Exclusive-OR instructions

Assembly Language	*Operation*
XOR CH,DL	CH = CH XOR DL
XOR SI,BX	SI = SI XOR BX
XOR EBX,EDI	EBX = EBX XOR EDI
XOR AH,0EEH	AH = AH XOR EEH
XOR DI,0DDH	DI = DI XOR 00DDH
XOR ESI,100	ESI = ESI XOR 00000064H
XOR DX,[SI]	DX is Exclusive-ORed with the word contents of the data segment memory location addressed by SI
XOR DATES[DI+2],AL	AL is Exclusive-ORed with the byte contents of the data segment memory location addressed by the sum of DATES, DI, and 2

Example of AND, OR and XOR

EXAMPLE 5–28

```
0000  81 C9 0600      OR    CX,0600H      ;set bits 9 and 10
0004  83 E1 FC        AND   CX,0FFFCH     ;clear bits 0 and 1
0007  81 F1 1000      XOR   CX,1000H      ;invert bit 12
```

Test Instruction

- **TEST** performs the AND operation.
  - only affects the condition of the flag register, which indicates the result of the test
  - functions the same manner as a CMP
- The destination operand is tested against the source operand, usually and immediate data.
  - TEST DL, DH
  - TEST CX, BX
  - TEST AH, 4 ;test the third bir from right
  - TEST AH, 128 ;test the leftmost bit

Test

- Usually followed by either the JZ (jump if zero) or JNZ (jump if not zero) instruction.

```
TEST AL, 1        ; test right bit
JNZ RIGHT         ; if set jump RIGHT
TEST AL, 128      ; test left bit
JNZ LEFT          ; if set jump LEFT
```

Bit Test

- Bit Test instruction tests single bit position

TABLE 5–18 Bit test instructions

Assembly Language	*Operation*
BT	Tests a bit in the destination operand specified by the source operand
BTC	Tests and complements a bit in the destination operand specified by the source operand
BTR	Tests and resets a bit in the destination operand specified by the source operand
BTS	Tests and sets a bit in the destination operand specified by the source operand

EXAMPLE 5–30

```
0000  0F BA E9 09        BTS   CX,9     ;set bit 9
0004  0F BA E9 0A        BTS   CX,10    ;set bit 10
0008  0F BA F1 00        BTR   CX,0     ;clear bit 0
000C  0F BA F1 01        BTR   CX,1     ;clear bit 1
0010  0F BA F9 0C        BTC   CX,12    ;invert bit 12
```

NOT and NEG

- NOT and NEG can use any addressing mode except segment register addressing.
- The NOT instruction inverts all bits of a byte, word, or doubleword.
- NEG two's complements a number.
  - the arithmetic sign of a signed number changes from positive to negative or negative to positive
- The NOT function is considered logical, NEG function is considered an arithmetic operation.

NOT and NEG

- Examples

NOT CH ; CH is one's complemented

NEG CH ; CH is two's complemented

NOT TEMP ;The content of the data segment memory
;location TEMP is one's complemented

NOT BYTE PTR[BX] ;The byte content of the data
;segment memory location pointed by BX is one's
;complemented

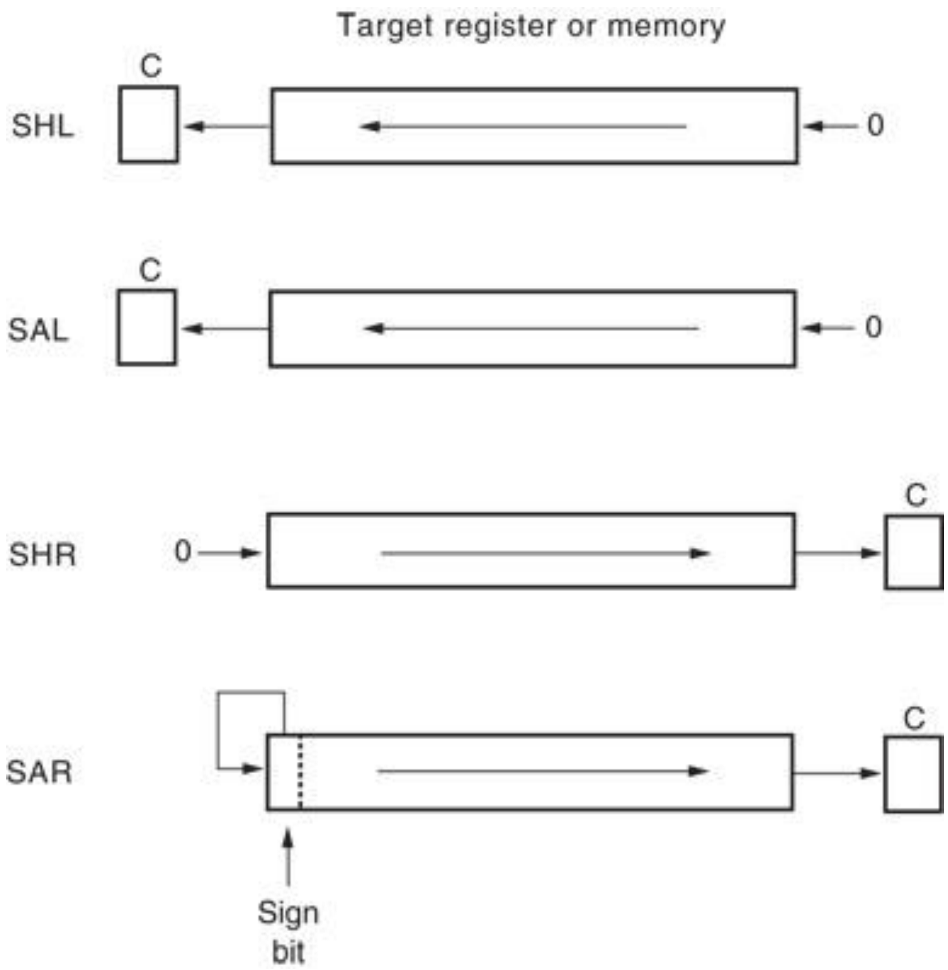
Shift and Rotate

- Shift and rotate instructions manipulate binary numbers at the binary bit level.
  - as did AND, OR, Exclusive-OR, and NOT
- Common applications in low-level software used to control I/O devices.
- The microprocessor contains a complete complement of shift and rotate instructions that are used to shift or rotate any memory data or register.

Shift

- Position or move numbers to the left or right within a register or memory location.
  - also perform simple arithmetic as multiplication by powers of 2^{+n} (left shift) and division by powers of 2^{-n} (right shift).
- The microprocessor's instruction set contains four different shift instructions:
  - two are logical; two are arithmetic shifts
- All four shift operations appear in Figure 5–9.

Figure 5–9 The shift instructions showing the operation and direction of the shift.



- logical shifts move 0 in the rightmost bit for a logical left shift;
- 0 to the leftmost bit position for a logical right shift
- arithmetic right shift copies the sign-bit through the number
- logical right shift copies a 0 through the number.

Shift

- Logical shifts multiply or divide unsigned data; arithmetic shifts multiply or divide signed data.
  - a shift left always multiplies by 2 for each bit position shifted
  - a shift right always divides by 2 for each position
  - shifting a two places, multiplies or divides by 4

Shift - Examples

SHL AX, 1

SHR BX, 12

SAL DATA1, CL

SAR SI, 2

SHL DX, 14
OR
MOV CL, 14
SHL DX, CL

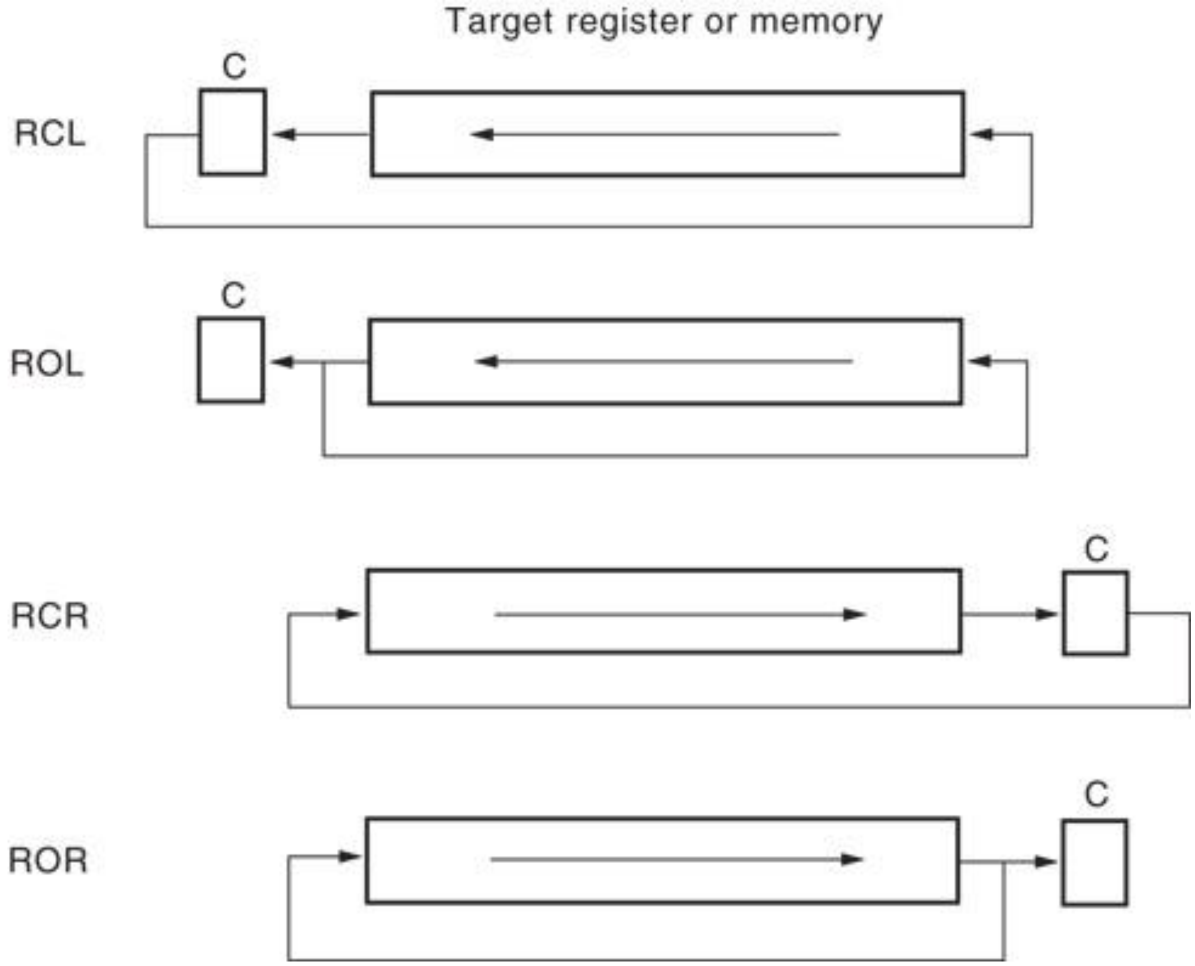
What does the following instructions do?

```
SHL AX, 1
MOV BX, AX
SHL AX, 2
ADD AX, BX
```

Rotate

- Positions binary data by rotating information in a register or memory location, either from one end to another or through the carry flag.
  - used to shift/position numbers wider than 16 bits
- With either type of instruction, the programmer can select either a left or a right rotate.
- Addressing modes used with rotate are the same as those used with shifts.
- Rotate instructions appear in Figure 5–10.

Figure 5–10 The rotate instructions showing the direction and operation of each rotate.



- A rotate count can be immediate or located in register CL.
  - if CL is used for a rotate count, it does not change
- Rotate instructions are often used to shift wide numbers to the left or right.

```
ROL SI, 14          ;SI rotates left 14 times
RCL BL, 6
RCR AH, CL          ;AH rotates right thru C the number of
                    ;places as specified in CL

ROR WORD PTR[BP], 2     ; word data in stack section
                        ;addressed by BP is rotated right 2 times
```

Bit Scan Instructions

- BSF (bit scan forward) and BSR (bit scan reverse) scan through a number searching for the first 1-bit encountered

String Comparisons

It is very powerful because allows to manipulate large blocks of data with relative ease

SCAS

- SCAS compares the accumulator register with a block of memory in ES:[DI]
  - Reg AL with a byte block of memory (**SCASB**), the AX register with a word block of memory (**SCASW**), or the EAX register with a doubleword block of memory (**SCASD**)

EXAMPLE 5–34

```
0000  BF 0011 R      MOV    DI,OFFSET BLOCK   ;address data
0003  FC             CLD                      ;auto-increment
0004  B9 0064        MOV    CX,100            ;load counter
0007  32 C0          XOR    AL,AL             ;clear AL
0009  F2/AE          REPNE  SCASB             ;search
```

EXAMPLE 5–35

```
0000                 SKIP   PROC   FAR

0000  FC                    CLD                 ;auto-increment
0001  B9 0100               MOV    CX,256       ;counter
0004  B0 20                 MOV    AL,20H       ;get space
0006  F3/AE                 REPE   SCASB        ;search
0008  CB                    RET

0009                 SKIP   ENDP
```

CMPS

- It always compares two sections of memory ES:[DI] with DS:[SI]
  - data as bytes (CMPSB), word (CMPSW), or doubleword (CMPSD)

EXAMPLE 5–36

```
0000               MATCH   PROC    FAR

0000 BE 0075 R             MOV     SI,OFFSET LINE       ;address LINE
0003 BF 007F R             MOV     DI,OFFSET TABLE      ;address TABLE
0006 FC                    CLD                          ;auto-increment
0007 B9 000A               MOV     CX,10                ;counter
000A F3/A6                 REPE    CMPSB                ;search
000C CB                    RET

000D               MATCH   ENDP
```

EXAMPLE 5–37

```
                  ;program that tests the video display for the word BUG
                  ;if BUG appears anywhere on the display, a Y is
                  ;displayed
                  ;if BUG does not appear, the program displays N
                  ;
                          .MODEL SMALL                ;select SMALL model
0000                      .DATA                       ;start of DATA segment
0000 42 55 47     DATA1   DB      'BUG'               ;define BUG
0000                      .CODE                       ;start of CODE segment
                          .STARTUP                    ;start of program
0017 B8 B800              MOV     AX,0B800H           ;address segment B800 with ES
001A 8E C0                MOV     ES,AX
```

```
C B9 07D0           MOV    CX,25*80          ;set count
F FC                CLD                      ;select increment
0 BF 0000           MOV    DI,0              ;address display
3            L1:
3 BE 0000 R         MOV    SI,OFFSET DATA1   ;address BUG
6 57                PUSH   DI                ;save display address
7 A6                CMPSB                    ;test for B
8 75 0A             JNE    L2                ;if display is not B
A 47                INC    DI                ;address next position
B A6                CMPSB                    ;test for U
C 75 06             JNE    L2                ;if display is not U
E 47                INC    DI                ;address next position
F A6                CMPSB                    ;test for G
0 B2 59             MOV    DL,'Y'            ;load Y for possible BUG
2 74 09             JE     L3                ;if BUG is found
4            L2:
4 5F                 POP   DI                ;restore display address
5 83 C7 02           ADD   DI,2              ;point to next position
8 E2 E9              LOOP  L1                ;repeat for whole screen
A 57                 PUSH  DI                ;save display address
B B2 4E              MOV   DL,'N'            ;indicate N if no BUG
D            L3:
D 5F                 POP   DI                ;clear stack
E B4 02              MOV   AH,2              ;display DL function
0 CD 21              INT   21H               ;display ASCII from DL
                     .EXIT                   ;exit to DOS
                     END                     ;end of file
```

Reference/Text Book

- "The Intel Microprocessors", 8th Edition, Brey, Barry, B., Prentice Hall, USA, 2009