Arithmetic and Logic Instructions

Addition(ADD and ADC)

- ✓ The bulk of the arithmetic instructions found in any microprocessor include addition, subtraction, and comparison
- ✓ More than 32,000 variations of the ADD instruction in the instruction set
- ✓ The only types of addition not allowed are memory-to-memory and segment registers
- ✓ The segment registers can only be moved, pushed, or popped

Examples:

i)	ADD AL,BL	; AL=AL+BL
ii)	ADD CX,DI	; CX=CX+DI
iii)	ADD CL,44H	; CL=CL+44H
iv)	ADD BX,245FH	; BX=BX+245FH
v)	ADD [BX],AL	; AL adds to the byte contents of the data segment memory location addressed by BX with the sum stored in the same memory location
vi)	ADD CL,[BP]	;The byte contents of the stack segment memory location addressed by BP add to CL with the sum stored in CL
vii)	ADD CL,TEMP	;The byte contents of data segment memory location TEMP add to CL with the sum stored in CL
viii)	ADC AL,AH	; AL=AL+AH+**carry**
ix)	ADC CX,BX	; CX=CX+BX+**carry**

[An addition-with-carry) instruction (ADC) adds the bit in the carry flag (C) to the operand data]

Increment Addition:

INC BL; BL=BL+1

INC SP ; SP=SP+1

(Increment addition adds 1 to a register or a memory location)

Subtraction:

i)	SUB CL,BL	; CL=CL-BL
ii)	SUB AX,SP	; AX=AX-SP
iii)	SUB DH,6FH	; DH=DH-6FH
iv)	SUB AH,TEMP	; subtracts the byte contents of memory location TEMP from AH and store the difference in AH

Decrement(DEC):

i)	DEC BH	; BH=BH-1
ii)	DEC CX	; CX=CX-1

Subtraction with Borrow:

i)	SBB AH,AL	; AH=AH-AL-Carry
ii)	SBB CL,2	; CL=CL-2-Carry

Comparison

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

```
i)    CMP CL,BL        ; CL-BL
ii)   CMP AX,SP        ; AX-SP
iii)  CMP [DI],CH      ; CH subtracts from the byte
                         contents of the data segment
                         memory location addressed by DI
```

Example 5-12:

```
CMP AL,10H             ; compare AL against 10H
JAE SUBER              ; if AL is 10H or above jump to location SUBER
```

Normally used:

JA(Jump Above)
JB(Jump Below)
JAE(Jump Above or Equal)
JBE(Jump Below or Equal)

Multiplication

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

Note:

i) Multiplication is performed on bytes or words

ii) The product after a multiplication is always a **double-width** product. If two 8-bit numbers are multiplied they generate a 16-bit product. If two 16-bit numbers are multiplied, they generate a 32-bit product

iii) Some flag bits [**overflow (O)** **and** **carry (C)**] change when the multiply instruction executes and produce predictable outcomes

8-bit Multiplication

The multiplication instruction contains one operand because it always multiplies the operand times the contents of AL

i) MUL CL ; AL is multiplied by CL and the unsigned product is in AX

ii) IMUL DH ; AL is multiplied by DH and the signed product is in AX

Example 5-13

Write a short instruction to multiply the content of BL and CL. Load BL with 5, CL with 10. Store the result in DX register

```
MOV BL,5          ;load data
MOV CL,10
MOV AL,CL         ;position data
MUL BL            ;multiply , result is in AX
MOV DX,AX         ;hence move AX to DX
```

Use **IMUL** instead of **MUL** for signed 8-bit multiplication

16-bit Multiplication

- ✓ AX contains the multiplicand instead of AL
- ✓ The 32-bit product appears in DX-AX instead of AX
- ✓ DX register contains the Most Significant 16 bits of the product
- ✓ AX contains the Least Significant 16 bits

i)	MUL CX	; AX is multiplied by CX and the unsigned product is in DX-AX
ii)	IMUL DI	; AX is multiplied by DI and the signed product is in DX-AX

Note:

8086/8088 microprocessors can not perform immediate multiplication

Division

- ✓ Division occurs on 8-or 16-bit numbers
- ✓ The dividend is always a double-width dividend that is divided by the operand
- ✓ This means that an 8-bit division divides a 16-bit number by an 8-bit number; a 16-bit division divides a 32-bit number by a 16-bit number
- ✓ None of the flag bits change predictably for a division. A division can result in two different types of errors
  1) One is an attempt to divide by zero
  2) Other is a divide overflow

  For Example:

  If AX=3000 is divided by 2, answer = 1500, which does not fit into AL

8-bit Division

- ✓ An 8-bit division uses the AX register to store the dividend that is divided by the contents of any 8-bit register or memory location
- ✓ The quotient moves into AL after division with AH containing a whole number remainder
- ✓ Zero-extension is needed in 8-bit division

i) DIV CL ; AX is divided by CL; the unsigned quotient is in AL and the remainder is in AH

ii) IDIV BL ; AX is divided by BL; the signed quotient is in AL and the signed remainder is in AH

Example 5-14

Divide number NUMB by number NUMB1. Store the quotient at ANSQ and remainder at ANSR

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

16-bit Division

DIV CX ; DX-AX is divided by CX and the unsigned quotient is in AX and unsigned remainder is in DX

Example 5-15:

-100 is in AX. Divide by **+9** in CX register

Solution:

```
MOV AX, -100          ; load -100
MOV CX, 9             ; load +9
CWD                   ; CWD convert word to double word
                      (i.e  -100 in AX is converted to -100 in DX-AX)
IDIV CX               ;quotient -11 will be in AX and remainder -1 in DX
```

BCD Arithmetic

DAA: Decimal Adjust after Addition

DAS: Decimal Adjust after Subtraction

- ✓ DAA follows BCD addition i.e ADD or ADC
- ✓ DAS follows BCD subtraction
- ✓ The adjustment instructions function only with the AL register after BCD addition and subtraction
- ✓ DAS instruction functions as does the DAA instruction, except that it follows a subtraction instead of an addition

ACII Arithmetic

The ASCII arithmetic function with ASCII-coded numbers. There are four instructions used with ASCII arithmetic operations

i) AAA (ASCII adjust after addition)

ii) AAD(ASCII adjust before Division)

iii) AAM(ASCII adjust after Multiplication)

iv) AAS(ASCII adjust after Subtraction)

These instructions use register AX as the source and as the destination.

AAA

Example 5-10:ASCII addition in 8086

```
MOV AX,31H          ;load ASCII ' 1'
ADD AX,39H          ;load ASCII ' 9'
AAA                 ; AX will contain 10
ADD AX,3030H        ; answer to ACII if we wish to display
```

Explanation:

- ✓ If 31H and 39H are added , result is 6AH. The ASCII addition (1+9) should produce a two-digit ASCII result equivalent to a decimal 10, which is a 31H and 30H in ASCII code
- ✓ If the **AAA** instruction is executed after the addition, the AX register will contain 0100H
- ✓ Although this is not ASCII code, it can be converted to ASCII code by adding **3030H** to AX which generates 3130H

AAD

Example 5-21: Divide 72 in unpacked BCD by 9

```
MOV BL,9            ;load divisor
MOV AX,72H          ;load dividend
AAD                 ; adjust
DIV BL              ; divide
```

AAM Instruction:

It is followed after Multiplication

Example 5-22:

```
MOV AL,5          ;load multiplicand
MOV CL,3          ;load multiplier
MUL CL            ;multiply
AAM               ; adjust
```

AAS Instruction:

AAS instruction adjusts the AX register after ASCII Subtraction

Basic Logic Instructions

AND:

i) AND AL,BL ; AL=AL and BL

ii) AND CX,DX ; CX=CX and DX

iii) AND CL,33H ; CL=CL and 33H

OR:

i) OR AH,BL ; AH=AH or BL

ii) OR SI,DX ; SI=SI or DX

iii) OR SP,990DH ; SP=Sp or 990DH

XOR:

i) XOR CH,DL ; CH=CH xor DL

ii) XOR DI,BX ; DI=DI xor BX

iii) XOR AH,0EEH ; AH=AH xor EEH

TEST:

The TEST instruction performs the AND operation. The difference is that the AND instruction changes the destination operand, whereas the TEST instruction does not.

i) TEST DL,DH ; DL is ANDed with DH

ii) TEST CX,BX ; CX is ANDed with BX

NOT and NEG:

i) NOT CH ; CH is **one's** complemented

ii) NEG CH ; CH is **two's** complemented

iii) NEG AX ; AX is **two's** complemented

Shift and Rotate Instructions

Shift:

Shift instructions position or move numbers to the left or right within a register or memory location. The microprocessors instruction set contains four different shift operations. Two are logical shifts and two are arithmetic shifts

```
i)    SHL AX,1          ; AX is logically shifted left 1 place
ii)   SHR BX,12         ; BX is logically shifted right 12 places
iii)  SAR SI,2          ; SI is arithmetically shifted right 2 places
iv)   SAL DATA1,CL      ; The contents of data segment memory
                          location DATA1 are arithmetically shifted
                          left the number of spaces specified by CL
```

Note:

- ✓ **The arithmetic left shift and logical left shift are identical**
- ✓ **The arithmetic right shift and logical right shift are different because the arithmetic right shift copies the sign-bit through the number, whereas the logical right shift copies a 0 through the number**
- ✓ Segment register cannot be shifted

(Please see **example 5-31** for an example to see how to multiply AX by 10, 18, or 5)

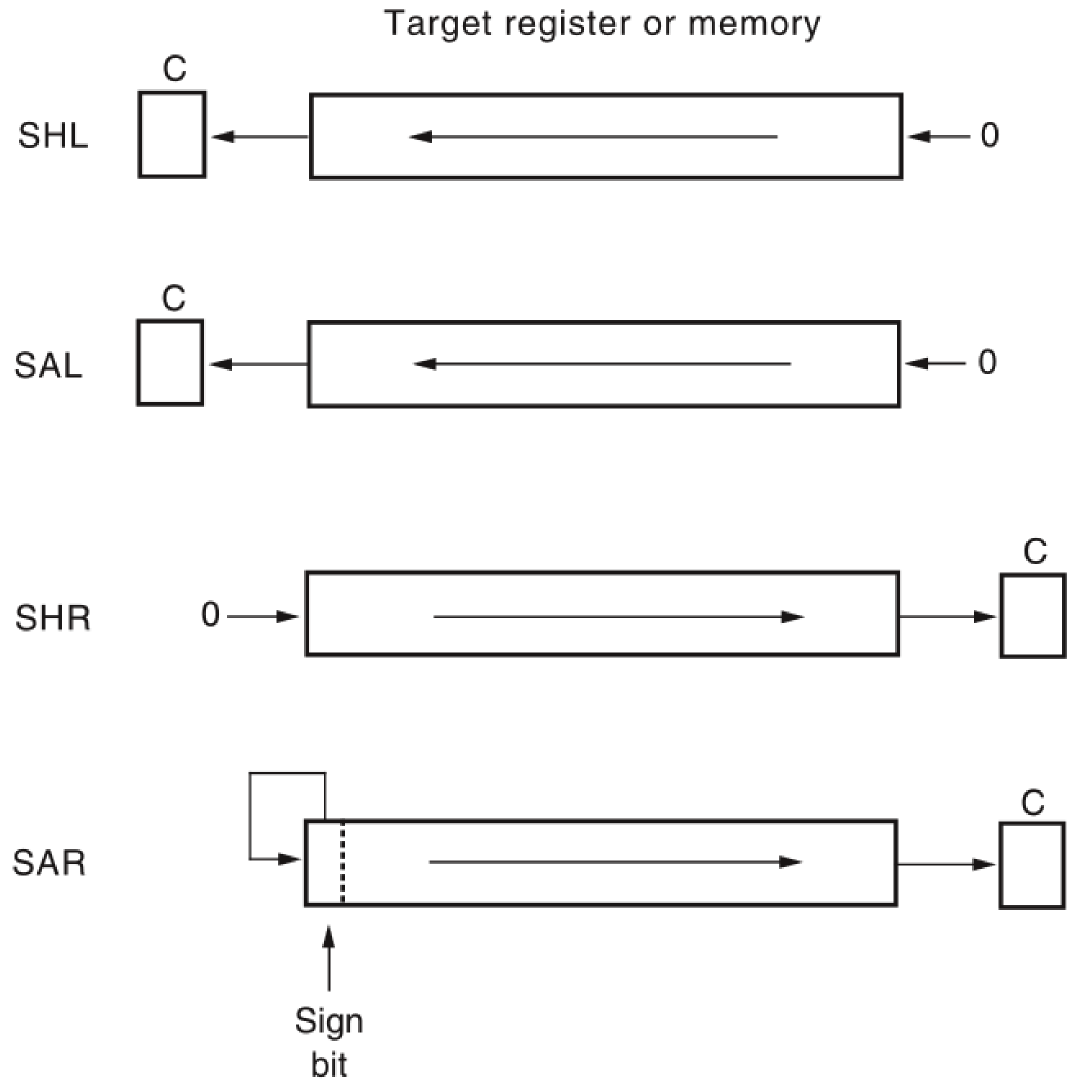


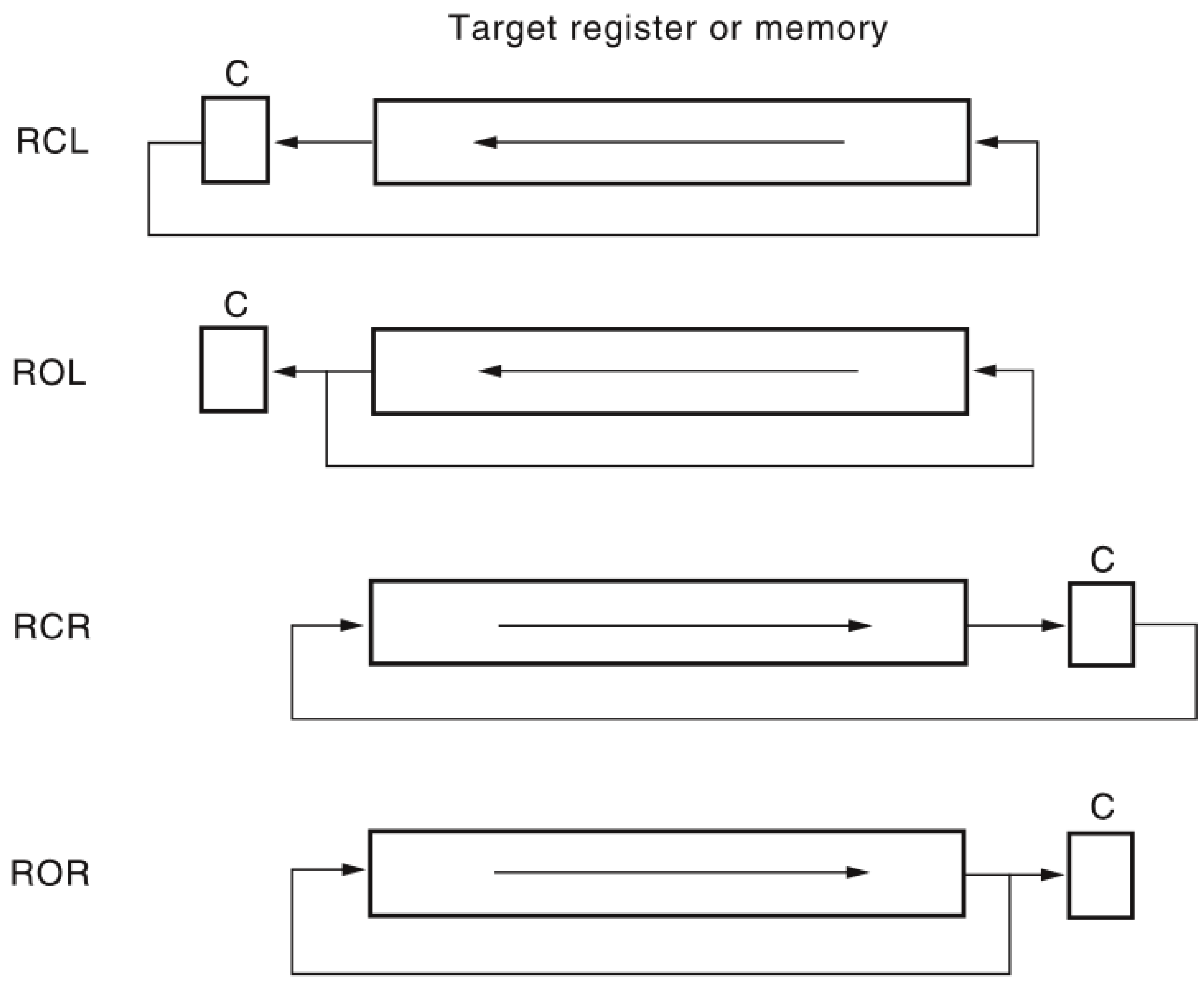
FIGURE 5–9 The shift instructions showing the operation and direction of the shift.

Rotate:

Rotate instructions position binary data by rotating the information in a register or memory location, either from one end to another or through the carry

i)	ROL SI,14	; SI rotates left 14 places
ii)	RCR AH,CL	; AH rotates right through carry the number of places specified by CL

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



NEXT CLASS

Program Control Instructions

THANK YOU