CSE213

Microcontroller Programming

Arithmetic and Logic Instructions

Introduction

- We examine the arithmetic and logic instructions. 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).

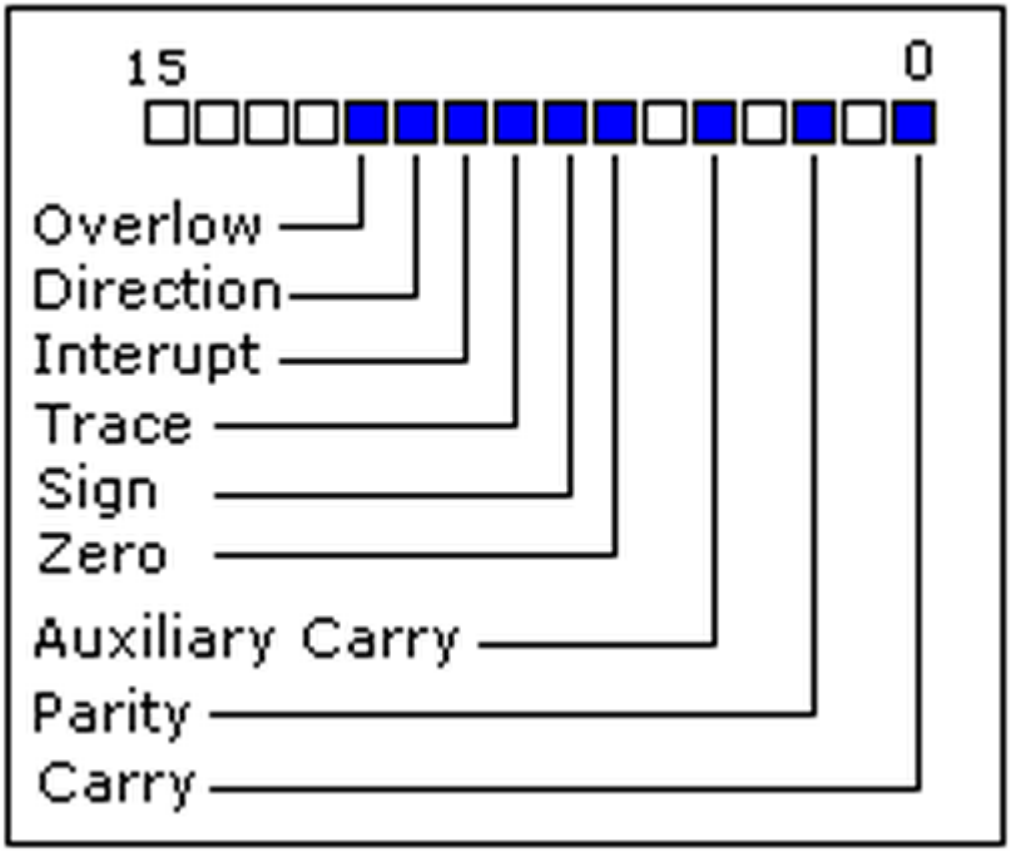
Chapter Objectives

Upon completion of this chapter, you will be able to:

- Use arithmetic instructions to accomplish simple binary, BCD, and ASCII arithmetic.
- Use logic instructions to accomplish binary bit manipulation.
- Use the shift and rotate instructions.

Arithmetic and Logic Instructions

- Most arithmetic instructions affect the **FLAGS** register.



Arithmetic and Logic Instructions

- **Carry Flag (C)** - this flag is set to **1** when there is an **unsigned overflow**. For example when you add bytes **255 + 1** (result is not in range 0...255). When there is no overflow this flag is set to **0**.
- **Zero Flag (Z)** - set to **1** when result is **zero**. For none-zero result this flag is set to **0**.
- **Sign Flag (S)** - set to **1** when result is **negative**. When result is **positive** it is set to **0**. Actually this flag take the value of the most significant bit.
- **Overflow Flag (O)** - set to **1** when there is a **signed overflow**. For example, when you add bytes **100 + 50** (result is not in range -128...127).

Arithmetic and Logic Instructions

- **Parity Flag (P)** - this flag is set to **1** when there is even number of one bits in result, and to **0** when there is odd number of one bits. Even if result is a word only 8 low bits are analyzed!
- **Auxiliary Flag (A)** - set to **1** when there is an **unsigned overflow** for low nibble (4 bits).
- **Interrupt enable Flag (I)** - when this flag is set to **1** CPU reacts to interrupts from external devices.
- **Direction Flag (D)** - this flag is used by some instructions to process data chains, when this flag is set to **0** - the processing is done forward, when this flag is set to **1** the processing is done backward.

ADDITION, SUBTRACTION AND COMPARISON

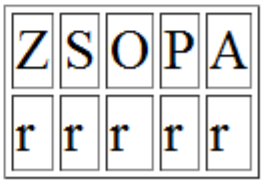
- The bulk of the arithmetic instructions found in any microprocessor include addition, subtraction, and comparison.
- Addition, subtraction, and comparison instructions are illustrated.
- Also shown are their uses in manipulating register and memory data.

Addition

- Addition (ADD) appears in many forms in the microprocessor.
- A second form of addition, called **add-with-carry**, is introduced with the ADC instruction.
- The only types of addition *not* allowed are memory-to-memory and segment register.
  - segment registers can only be moved, pushed, or popped.
- Increment instruction (INC) is a special type of addition that adds 1 to a number.

- Any ADD instruction modifies the contents of the sign, zero, carry, auxiliary carry, parity, and overflow flags.

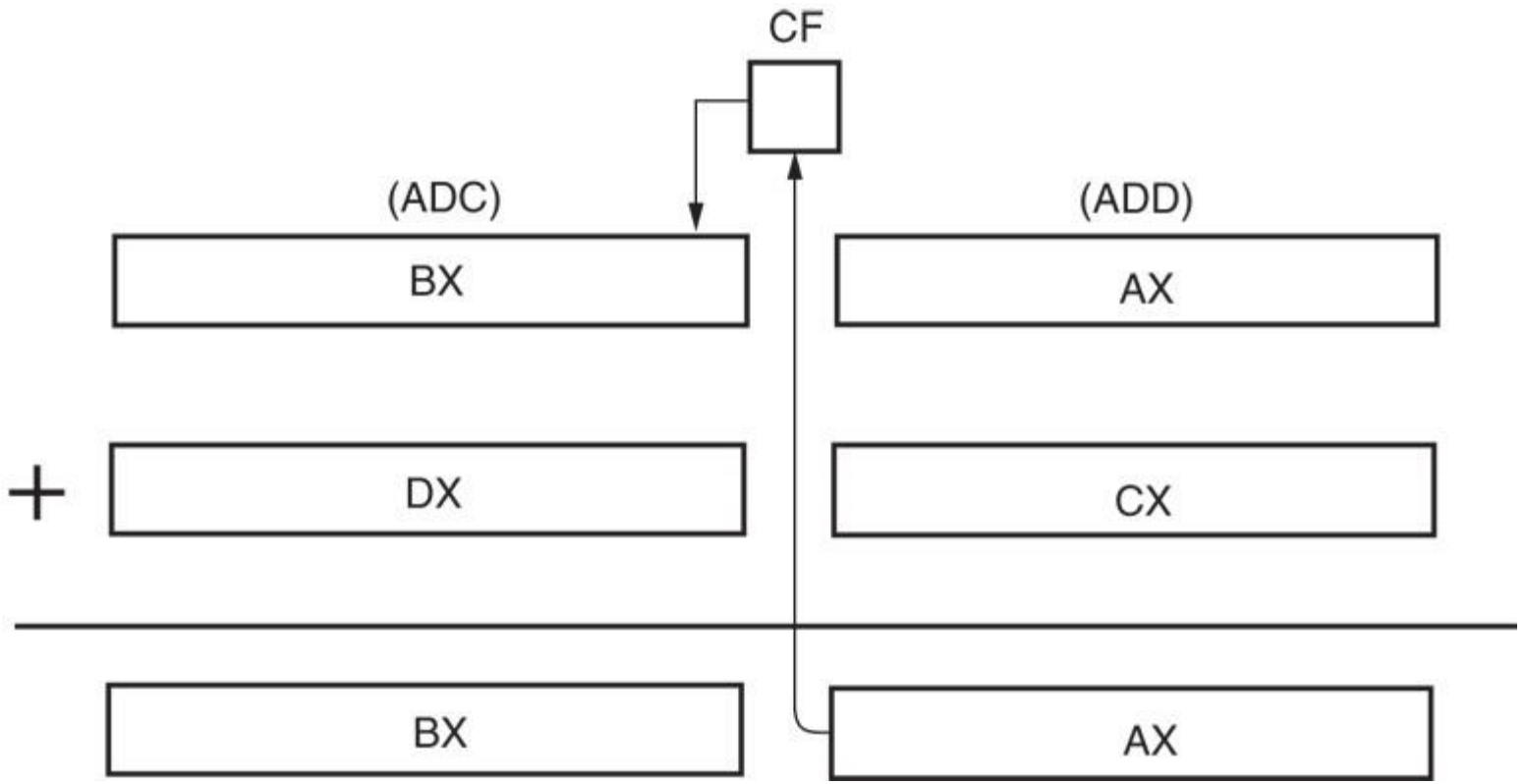
ADD	REG, memory
	memory, REG
	REG, REG
	memory, immediate
	REG, immediate
		Add.
		
		Algorithm:
		
		operand1 = operand1 + operand2
		
		Example:
		
		MOV AL, 5 ; AL = 5
		ADD AL, -3 ; AL = 2
		RET
		
		C Z S O P A
		r r r r r r

INC	REG memory	Increment. Algorithm: operand = operand + 1 Example: MOV AL, 4 INC AL ; AL = 5 RET  CF - unchanged!

Addition-with-Carry

- ADC) adds the bit in the carry flag (C) to the operand data.
  - mainly appears in software that adds numbers wider than 16 or 32 bits.
  - like ADD, ADC affects the flags after the addition
- Figure 5–1 illustrates this so placement and function of the carry flag can be understood.
  - cannot be easily performed without adding the carry flag bit because the 8086–80286 only adds 8- or 16-bit numbers

Figure 5–1 Addition-with-carry showing how the carry flag (C) links the two 16-bit additions into one 32-bit addition.



ADC	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Add with Carry. Algorithm: operand1 = operand1 + operand2 + CF Example: STC ; set CF = 1 MOV AL, 5 ; AL = 5 ADC AL, 1 ; AL = 7 RET C Z S O P A r r r r r r

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
- After each subtraction, the microprocessor modifies the contents of the flag register.
  - flags change for most arithmetic/logic operations

SUB	REG, memory
	memory, REG
	REG, REG
	memory, immediate
	REG, immediate
		Subtract.
		
		Algorithm:
		
		operand1 = operand1 - operand2
		
		Example:
		
		MOV AL, 5
		SUB AL, 1 ; AL = 4
		
		RET
		
		C Z S O P A
		r r r r r r

DEC

REG
memory

Decrement.

Algorithm:

operand = operand - 1

Example:

```
MOV AL, 255  ; AL = 0FFh (255 or -1)
DEC AL       ; AL = 0FEh (254 or -2)
RET
```

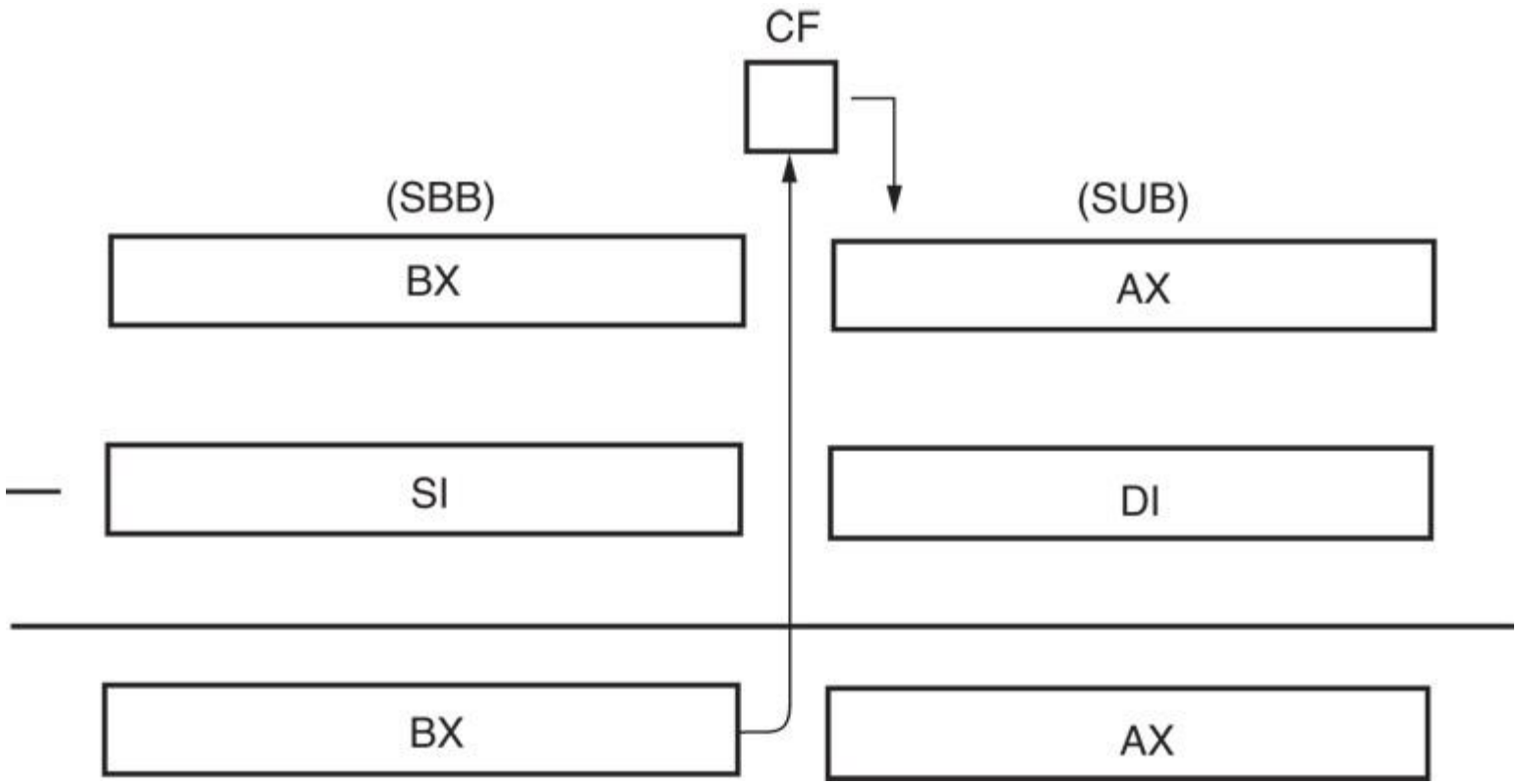
Z	S	O	P	A
r	r	r	r	r

CF - unchanged!

Subtraction-with-Borrow

- 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

Figure 5–2 Subtraction-with-borrow showing how the carry flag (C) propagates the borrow.



SBB	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Subtract with Borrow. Algorithm: operand1 = operand1 - operand2 - CF Example: STC MOV AL, 5 SBB AL, 3 ; AL = 5 - 3 - 1 = 1 RET C Z S O P A r r r r r r

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	REG, memory
	memory, REG
	REG, REG
	memory, immediate
	REG, immediate
		Compare.
		
		Algorithm:
		
		operand1 - operand2
		
		result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.
		
		Example:
		
		MOV AL, 5
		MOV BL, 5
		CMP AL, BL ; AL = 5, ZF = 1 (so equal!)
		RET
		
		C Z S O P A
		r r r r r r

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.
- Pentium–Core2 contains special circuitry to do multiplication in as few as one clocking period.
  - over 40 clocking periods in earlier processors

Multiplication

- The multiplication instruction contains one operand because it always multiplies the operand times the contents of register AL/AX.
- 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

MUL	REG
	memory
		Unsigned multiply.
		
		Algorithm:
		
		when operand is a **byte**:
		AX = AL * operand.
		
		when operand is a **word**:
		(DX AX) = AX * operand.
		
		Example:
		
		MOV AL, 200 ; AL = 0C8h
		MOV BL, 4
		MUL BL ; AX = 0320h (800)
		RET
		
		C Z S O P A
		r ? ? r ? ?
		CF=OF=0 when high section of the result is zero.

IMUL	REG
	memory
		Signed multiply.
		
		Algorithm:
		
		when operand is a **byte**:
		AX = AL * operand.
		
		when operand is a **word**:
		(DX AX) = AX * operand.
		
		Example:
		
		MOV AL, -2
		MOV BL, -4
		IMUL BL ; AX = 8
		RET
		
		C Z S O P A
		r ? ? r ? ?
		CF=OF=0 when result fits into operand of IMUL.

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.

- 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
- In either case, the microprocessor generates an interrupt if a divide error occurs.
- In most systems, a divide error interrupt displays an error message on the video screen.

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
- The remainder could also be converted to a fractional remainder.

DIV	REG
	memory
		Unsigned divide.
		
		Algorithm:
		
		when operand is a **byte**:
		AL = AX / operand
		AH = remainder (modulus)
		
		when operand is a **word**:
		AX = (DX AX) / operand
		DX = remainder (modulus)
		
		Example:
		
		MOV AX, 203 ; AX = 00CBh
		MOV BL, 4
		DIV BL ; AL = 50 (32h), AH = 3
		RET
		
		C Z S O P A
		? ? ? ? ? ?

IDIV	REG
	memory
		Signed divide.
		
		Algorithm:
		
		when operand is a **byte**:
		AL = AX / operand
		AH = remainder (modulus)
		
		when operand is a **word**:
		AX = (DX AX) / operand
		DX = remainder (modulus)
		
		Example:
		
		MOV AX, -203 ; AX = 0FF35h
		MOV BL, 4
		IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)
		RET
		
		C Z S O P A
		? ? ? ? ? ?

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.
  - both correct the result of addition or subtraction so it is a BCD number

DAA Instruction

- DAA follows the ADD or ADC instruction to adjust the result into a BCD result.
- After adding the BL and DL registers, the result is adjusted with a DAA instruction before being stored in CL.

DAS Instruction

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

DAA	No operands	Decimal adjust After Addition.
		Corrects the result of addition of two packed BCD values.
		
		Algorithm:
		
		if low nibble of AL > 9 or AF = 1 then:
		• AL = AL + 6
		• AF = 1
		
		if AL > 9Fh or CF = 1 then:
		• AL = AL + 60h
		• CF = 1
		
		Example:
		
		MOV AL, 0Fh ; AL = 0Fh (15)
		DAA ; AL = 15h
		RET
		
		C Z S O P A
		r r r r r r

DAS	No operands	Decimal adjust After Subtraction.
		Corrects the result of subtraction of two packed BCD values.
		
		Algorithm:
		
		if low nibble of AL > 9 or AF = 1 then:
		
		• AL = AL - 6
		• AF = 1
		
		if AL > 9Fh or CF = 1 then:
		
		• AL = AL - 60h
		• CF = 1
		
		Example:
		
		MOV AL, 0FFh ; AL = 0FFh (-1)
		DAS ; AL = 99h, CF = 1
		RET
		
		C Z S O P A
		r r r r r r

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**)
  - AAD (**ASCII adjust before division**)
  - AAM (**ASCII adjust after multiplication**)
  - AAS (**ASCII adjust after subtraction**)
- These instructions use register AX as the source and as the destination.

AAA Instruction

- Addition of two one-digit ASCII-coded numbers will not result in any useful data.

AAD Instruction

- Appears before a division.
- The AAD instruction requires the AX register contain a two-digit unpacked BCD number (not ASCII) before executing.

AAM Instruction

- Follows multiplication instruction after multiplying two one-digit unpacked BCD numbers.
- AAM converts from binary to unpacked BCD.
- If a binary number between 0000H and 0063H appears in AX, AAM converts it to BCD.

AAS Instruction

- AAS adjusts the AX register after an ASCII subtraction.

BASIC LOGIC INSTRUCTIONS

- Include AND, OR, Exclusive-OR, and NOT.
  - also TEST, a special form of the AND instruction
  - NEG, similar to the NOT instruction
- Logic operations provide binary bit control in low-level software.
  - allow bits to be set, cleared, or complemented
- Low-level software appears in machine language or assembly language form and often controls the I/O devices in a system.

- All logic instructions affect the flag bits.
- Logic operations always clear the carry and overflow flags
  - other flags change to reflect the result
- When binary data are manipulated in a register or a memory location, the rightmost bit position is always numbered bit 0.
  - position numbers increase from bit 0 to the left, to bit 7 for a byte, and to bit 15 for a word
  - a doubleword (32 bits) uses bit position 31 as its leftmost bit and a quadword (64-bits) position 63

AND

- Performs logical multiplication, illustrated by a truth table.
- AND can replace discrete AND gates if the speed required is not too great
  - normally reserved for embedded control applications
- In 8086, the AND instruction often executes in about a microsecond.
  - with newer versions, the execution speed is greatly increased

Figure 5–3 (a) The truth table for the AND operation and (b) the logic symbol of an AND gate.

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

(a)

(b)

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

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
```

```
MOV BX,3135H          ;load ASCII
AND BX,0F0FH          ;mask BX
```

AND	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical AND between all bits of two operands. Result is stored in operand1. These rules apply: 1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0 Example: MOV AL, 'a' ; AL = 01100001b AND AL, 11011111b ; AL = 01000001b ('A') RET C Z S O P 0 r r 0 r

Examples

TABLE 5–16 Example 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 RDX,RBP	RDX = RDX and RBP (64-bit mode)
AND CL,33H	CL = CL and 33H
AND DI,4FFFH	DI = DI and 4FFFH
AND ESI,34H	ESI = ESI and 34H
AND RAX,1	RAX = RAX and 1 (64-bit mode)
AND AX,[DI]	The word contents of the data segment memory location addressed by DI are ANDed with AX
AND ARRAY[SI],AL	The byte contents of the data segment memory location addressed by ARRAY plus SI are ANDed with AL
AND [EAX],CL	CL is ANDed with the byte contents of the data segment memory location addressed by ECX

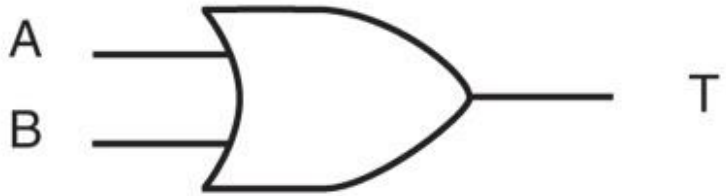
OR

- Performs logical addition
  - often called the *Inclusive-OR* function
- The OR function generates a logic 1 output if any inputs are 1.
  - a 0 appears at output only when all inputs are 0
- Figure 5–6 shows how the OR gate sets (1) any bit of a binary number.
- The OR instruction uses any addressing mode except segment register addressing.

Figure 5–5 (a) The truth table for the OR operation and (b) the logic symbol of an OR gate.

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

(a)



(b)

Figure 5–6 The operation of the OR function showing how bits of a number are set to one.

```
   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
```

```
MOV   AL,5              ;load data
MOV   BL,7
MUL   BL
AAM                     ;adjust
OR    AX,3030H          ;convert to ASCII
```

OR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical OR between all bits of two operands. Result is stored in first operand. These rules apply: 1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1 0 OR 0 = 0 Example: MOV AL, 'A' ; AL = 01000001b OR AL, 00100000b ; AL = 01100001b ('a') RET C Z S O P A 0 r r 0 r ?

Examples

TABLE 5–17 Example OR instructions.

Assembly Language	*Operation*
OR AH,BL	AL = AL or BL
OR SI,DX	SI = SI or DX
OR EAX,EBX	EAX = EAX or EBX
OR R9,R10	R9 = R9 or R10 (64-bit mode)
OR DH,0A3H	DH = DH or 0A3H
OR SP,990DH	SP = SP or 990DH
OR EBP,10	EBP = EBP or 10
OR RBP,1000H	RBP = RBP or 1000H (64-bit mode)
OR DX,[BX]	DX is ORed with the word contents of data segment memory location addressed by BX
OR DATES[DI + 2],AL	The byte contents of the data segment memory location addressed by DI plus 2 are ORed with AL

Exclusive-OR

- Differs from Inclusive-OR (OR) in that the 1,1 condition of Exclusive-OR produces a 0.
  - a 1,1 condition of the OR function produces a 1
- The Exclusive-OR operation *excludes* this condition; the Inclusive-OR *includes* it.
- If inputs of the Exclusive-OR function are both 0 or both 1, the output is 0; if the inputs are different, the output is 1.
- Exclusive-OR is sometimes called a comparator.

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

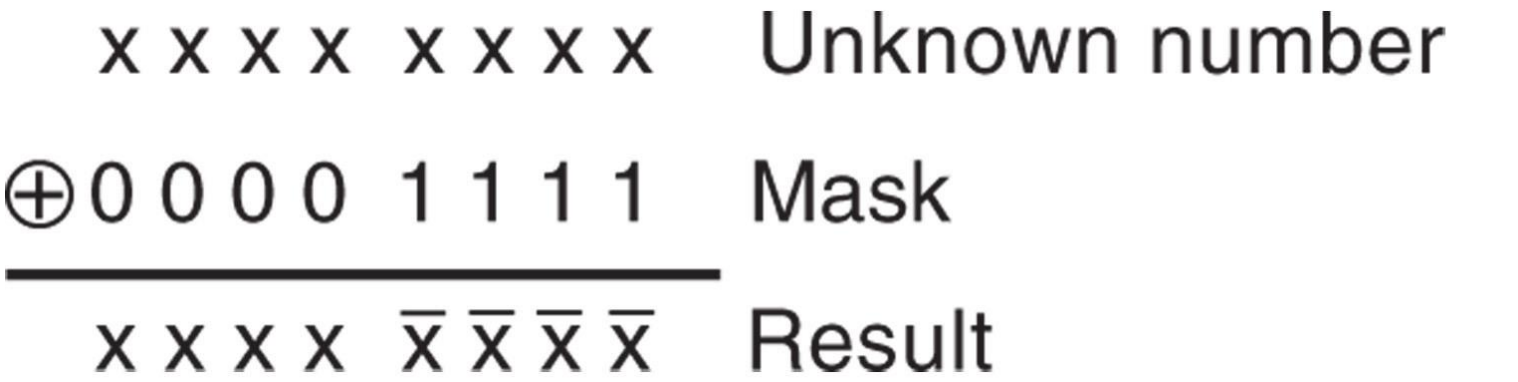
A	B	T
0	0	0
0	1	1
1	0	1
1	1	0

(a)

(b)

- XOR uses any addressing mode except segment register addressing.
- Exclusive-OR is useful if some bits of a register or memory location must be inverted.
- Figure 5–8 shows how 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
- A common use for the Exclusive-OR instruction is to clear a register to zero

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




```
OR    CX,0600H          ;set bits 9 and 10
AND   CX,0FFFCH         ;clear bits 0 and 1
XOR   CX,1000H          ;invert bit 12
```

XOR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand. These rules apply: 1 XOR 1 = 0 1 XOR 0 = 1 0 XOR 1 = 1 0 XOR 0 = 0 Example: MOV AL, 00000111b XOR AL, 00000010b ; AL = 00000101b RET C Z S O P A 0 r r 0 r ?

Examples

TABLE 5–18 Example 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 RAX,RBX	RAX = RAX xor RBX (64-bit mode)
XOR AH,0EEH	AH = AH xor 0EEH
XOR DI,00DDH	DI = DI xor 00DDH
XOR ESI,100	ESI = ESI xor 100
XOR R12,20	R12 = R12 xor 20 (64-bit mode)
XOR DX,[SI]	DX is Exclusive-ORed with the word contents of the data segment memory location addressed by SI
XOR DEAL[BP+2],AH	AH is Exclusive-ORed with the byte contents of the stack segment memory location addressed by BP plus 2

Test and Bit Test Instructions

- **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
- Usually followed by either the JZ (jump if zero) or JNZ (jump if not zero) instruction.
- The destination operand is normally tested against immediate data.

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

TEST	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical AND between all bits of two operands for flags only. These flags are effected: **ZF, SF, PF.** Result is not stored anywhere. These rules apply: 1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0 Example: MOV AL, 00000101b TEST AL, 1 ; ZF = 0. TEST AL, 10b ; ZF = 1. RET C Z S O P 0 r r 0 r

Examples

TABLE 5–19 Example TEST instructions.

Assembly Language	*Operation*
TEST DL,DH	DL is ANDed with DH
TEST CX,BX	CX is ANDed with BX
TEST EDX,ECX	EDX is ANDed with ECX
TEST RDX,R15	RDX is ANDed with R15 (64-bit mode)
TEST AH,4	AH is ANDed with 4
TEST EAX,256	EAX is ANDed with 256

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	REG memory	Invert each bit of the operand. Algorithm: • if bit is 1 turn it to 0. • if bit is 0 turn it to 1. Example: MOV AL, 00011011b NOT AL ; AL = 11100100b RET C Z S O P A unchanged

NEG	REG memory	Negate. Makes operand negative (two's complement). Algorithm: • Invert all bits of the operand • Add 1 to inverted operand Example: MOV AL, 5 ; AL = 05h NEG AL ; AL = 0FBh (-5) NEG AL ; AL = 05h (5) RET C Z S O P A r r r r r r

Examples

TABLE 5–21 Example NOT and NEG instructions.

Assembly Language	*Operation*
NOT CH	CH is one's complemented
NEG CH	CH is two's complemented
NEG AX	AX is two's complemented
NOT EBX	EBX is one's complemented
NEG ECX	ECX is two's complemented
NOT RAX	RAX is one's complemented (64-bit mode)
NOT TEMP	The contents of data segment memory location TEMP is one's complemented
NOT BYTE PTR[BX]	The byte contents of the data segment memory location addressed by BX are one's complemented

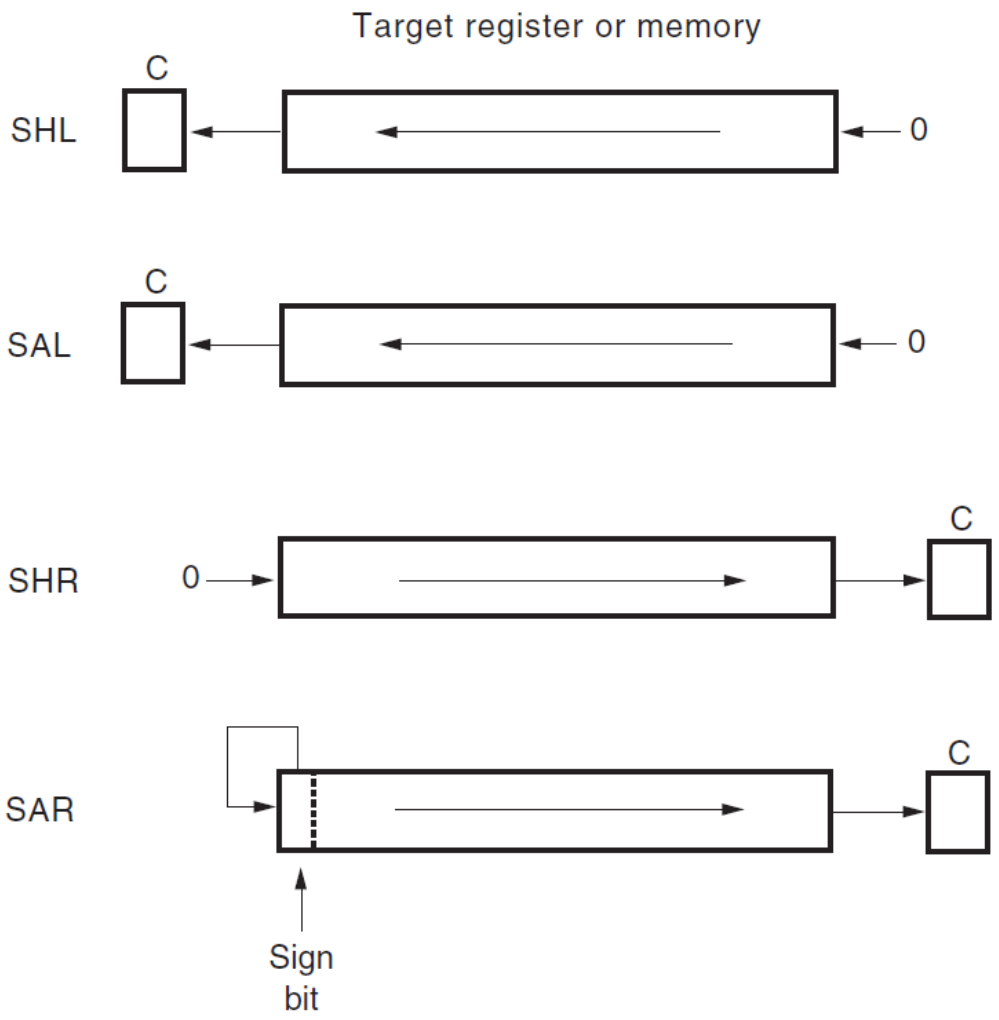
Shift and Rotate

- 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.

Logical shifts multiply or divide unsigned data; arithmetic shifts multiply or divide signed data.

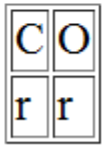
- a shift left always multiplies by 2 for each bit position shifted
- a shift right always divides by 2 for each position
- shifting a two places, multiplies or divides by 4

```
        SHL DX,14

or

        MOV CL,14
        SHL DX,CL
```

```
;Multiply AX by 10 (1010)
;
      SHL  AX,1            ;AX times 2
      MOV  BX,AX
      SHL  AX,2            ;AX times 8
      ADD  AX,BX           ;AX times 10
;
;Multiply AX by 18 (10010)
;
      SHL  AX,1            ;AX times 2
      MOV  BX,AX
      SHL  AX,3            ;AX times 16
      ADD  AX,BX           ;AX times 18
;
;Multiply AX by 5 (101)
;
      MOV  BX,AX
      SHL  AX,2            ;AX times 4
      ADD  AX,BX           ;AX times 5
```

SHL	memory, immediate REG, immediate memory, CL REG, CL	Shift operand1 Left. The number of shifts is set by operand2. Algorithm: • Shift all bits left, the bit that goes off is set to CF. • Zero bit is inserted to the right-most position. Example: MOV AL, 11100000b SHL AL, 1 ; AL = 11000000b, CF=1. RET  OF=0 if first operand keeps original sign.

Examples

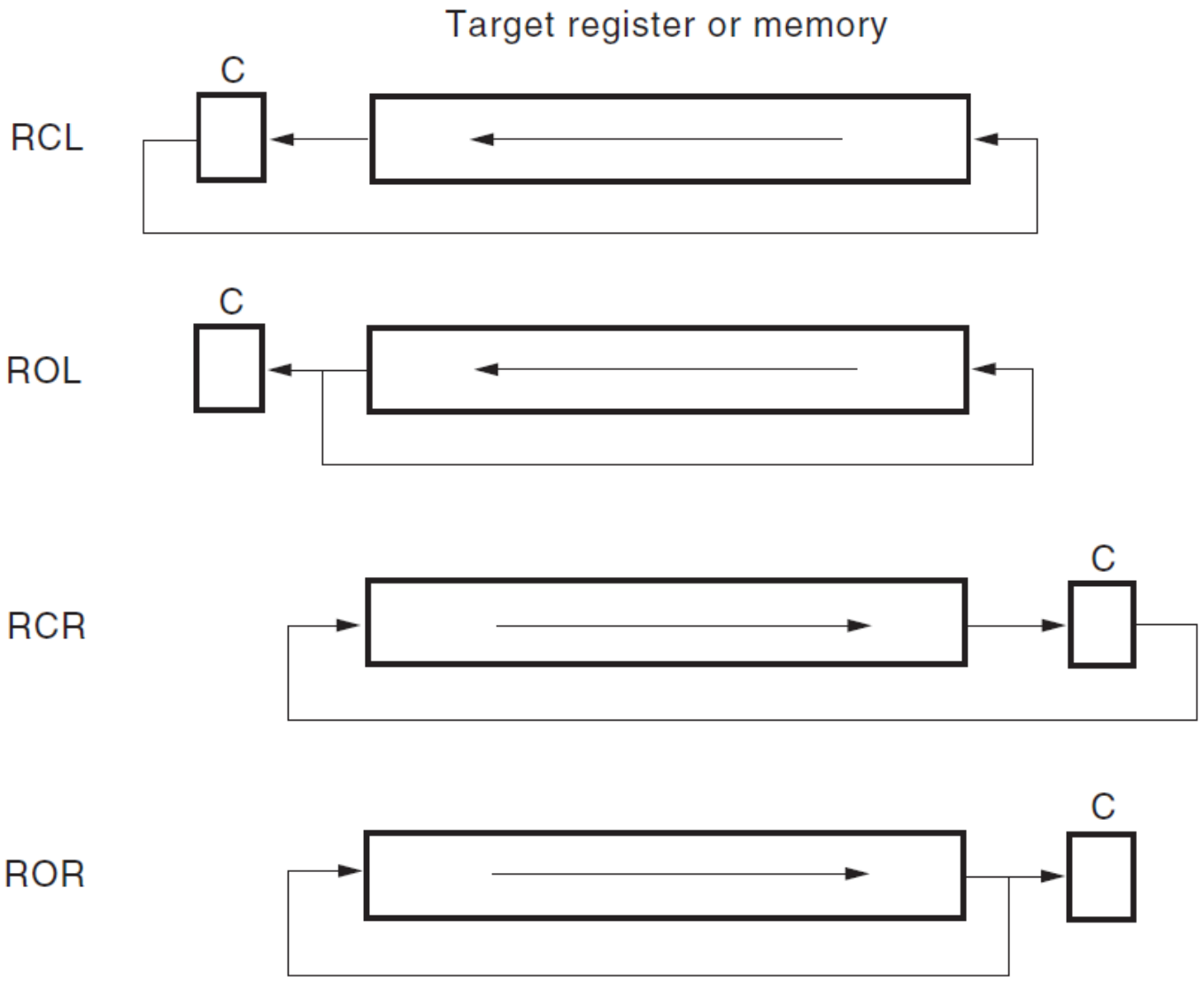
TABLE 5–22 Example shift instructions.

Assembly Language	*Operation*
SHL AX,1	AX is logically shifted left 1 place
SHR BX,12	BX is logically shifted right 12 places
SHR ECX,10	ECX is logically shifted right 10 places
SHL RAX,50	RAX is logically shifted left 50 places (64-bit mode)
SAL DATA1,CL	The contents of data segment memory location DATA1 are arithmetically shifted left the number of spaces specified by CL
SHR RAX,CL	RAX is logically shifted right the number of spaces specified by CL (64-bit mode)
SAR SI,2	SI is arithmetically shifted right 2 places
SAR EDX,14	EDX is arithmetically shifted right 14 places

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.

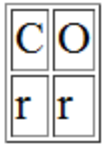
Figure 5–10 The rotate instructions showing the direction and operation of each rotate.



- 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.

```
SHL AX,1
RCL BX,1
RCL DX,1
```

- This program shifts the 48-bit number in registers DX, BX, and AX left one binary place.
- Notice that the least significant 16 bits (AX) shift left first.
- This moves the leftmost bit of AX into the carry flag bit.
- Next, the rotate BX instruction rotates carry into BX, and its leftmost bit moves into carry.
- The last instruction rotates carry into DX, and the shift is complete

ROL	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 left. The number of rotates is set by operand2. Algorithm: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position. Example: MOV AL, 1Ch ; AL = 00011100b ROL AL, 1 ; AL = 00111000b, CF=0. RET  OF=0 if first operand keeps original sign.

Examples

TABLE 5–23 Example rotate instructions.

Assembly Language	*Operation*
ROL SI,14	SI rotates left 14 places
RCL BL,6	BL rotates left through carry 6 places
ROL ECX,18	ECX rotates left 18 places
ROL RDX,40	RDX rotates left 40 places
RCR AH,CL	AH rotates right through carry the number of places specified by CL
ROR WORD PTR[BP],2	The word contents of the stack segment memory location addressed by BP rotate right 2 places

SUMMARY

- Flags
- Addition, Subtraction and Comparison
- Multiplication and Division
- BCD and ASCII Arithmetic
- Basic Logic Instructions
- Shift and Rotate

The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486 Pentium, Pentium Pro Processor, Pentium II, Pentium, 4, and Core2 with 64-bit Extensions Architecture, Programming, and Interfacing, Eighth Edition
Barry B. Brey

Copyright ©2009 by Pearson Education, Inc.
Upper Saddle River, New Jersey 07458 • All rights reserved.