Chapter 5

8088/8086 Microprocessor Programming

Introduction

5.1 Data-Transfer Instructions—✓
5.2 Arithmetic Instructions—✓
5.3 Logic Instructions—
5.4 Shift Instructions—
5.5 Rotate Instructions —

5.1 Data Transfer Instructions- Move Instruction

- Move instruction
  - Used to move (copy) data between:
    - Registers
    - Register and memory
    - Immediate operand to a register or memory
  - General format:
    MOV D,S
  - Operation: Copies the content of the source to the destination
    (S) → (D)
    - Source contents unchanged
    - Flags unaffected
  - Allowed operands
    Register
    Memory
    Accumulator (AH,AL,AX)
    Immediate operand (Source only)
    Segment register (Seg-reg)
  - Examples:
    MOV [SUM],AX
    (AL) → (address SUM)
    (AH) → (address SUM+1)

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	MOV D,S	(S) → (D)	None

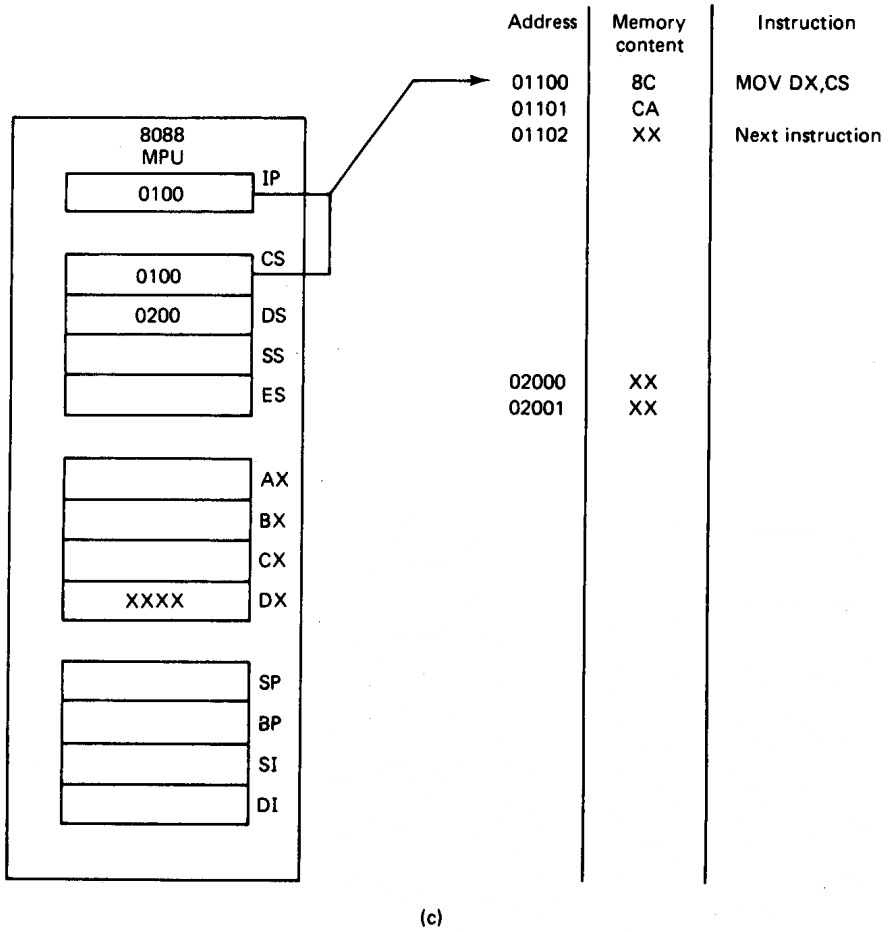
(a)

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg-reg	Reg16
Seg-reg	Mem16
Reg16	Seg-reg
Memory	Seg-reg

(b)

1. Is the destination in a register or memory?
2. What is the addressing mode of the source?
3. The destination?
4. What is SUM?

5.1 Data Transfer Instructions- Move Instruction



(c)

- Example

  MOV DX,CS

  Source = CS → word data

  Destination = DX → word data

  Operation: (CS) → (DX)

- State before fetch and execution

  CS:IP = 0100:0100 = 01100H

  Move instruction code = 8CCAH

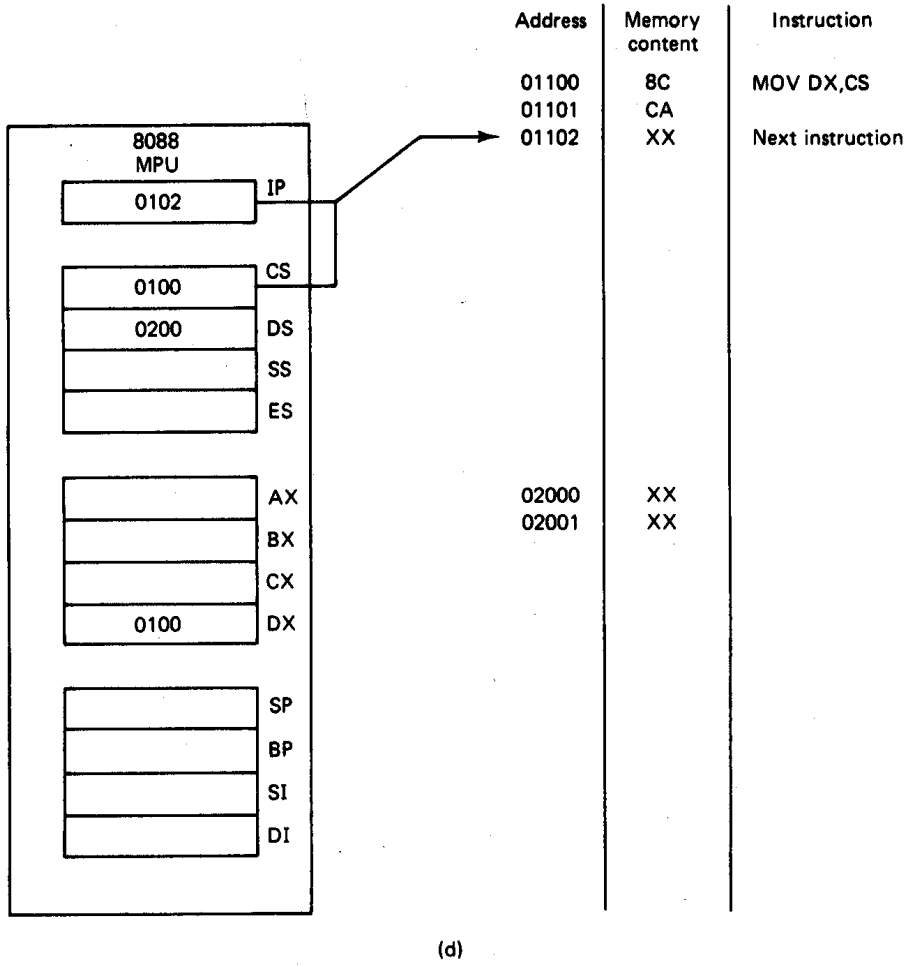
  (01100H) = 8CH

  (01101H) = CAH

  (CS) = 0100H

  (DX) = XXXX → don't care state

5.1 Data Transfer Instructions- Move Instruction



(d)

- Example (continued)
- State after execution

  CS:IP = 0100:0102 = 01102H

  01002H → points to next sequential instruction

  (CS) = 0100H

  (DX) = 0100H → Value in CS copied into DX

  Value in CS unchanged

5.1 Data Transfer Instructions- Move Instruction

- **Debug execution example**

  MOV CX,[20]

  DS = 1A00

  (DS:20) = AA55H

  (1A00:20) → (CX)

1. Where is the source operand located?
2. What is the addressing mode of the source operand?

```
C:\DOS>DEBUG
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 0F              DB      0F
-A
1342:0100 MOV   CX,[20]
1342:0104
-R DS
DS 1342
:1A00
-E 20 55 AA ◄——  How could you verify loading of this data?
-T

AX=0000  BX=0000  CX=AA55  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1A00  ES=1342  SS=1342  CS=1342  IP=0104   NV UP EI PL NZ NA PO NC
1342:0104 FFF3            PUSH    BX
-Q

C:\DOS>
```

5.1 Data Transfer Instructions- Move Instruction

```
MOV  AX,2000H
MOV  DS, AX        ; init_seg_reg
MOV  ES, AX
MOV  AX,3000H
MOV  SS,AX
MOV  AX,0H
MOV  BX,AX         ; init_data_reg
MOV  CX,0AH
MOV  DX,100H
MOV  SI,200H
MOV  DI,300H       ; init_index_reg
```

1. What addressing modes are in use in this program?

- Example—Initialization of internal registers with immediate data and address information
  - DS, ES, and SS registers initialized from immediate data via AX
    IMM16 → (AX)
    (AX) → (DS) & (ES) = 2000H
    IMM16 → (AX)
    (AX) → (SS) = 3000H
  - Data registers initialized
    IMM16 → (AX) =0000H
    (AX) → (BX) =0000H
    IMM16 → (CX) = 000AH and (DX) = 0100H
  - Index register initialized from immediate operations
    IMM16 → (SI) = 0200H and (DI) = 0300H

5.1 Data Transfer Instructions- Exchange Instruction

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(D) ↔ (S)	None

(a)

Destination	Source
Accumulator	Reg16
Memory	Register
Register	Register
Register	Memory

(b)

1. Why do you think this is known as a "complex instruction?"
2. How else could this operation be performed?
3. What is the benefit of using XCHG?

- Exchange instruction
  - Used to exchange the data between two data registers or a data register and memory
  - General format:

    XCHG D,S
  - Operation: Swaps the content of the source and destination
    - Both source and destination change

      (S) → (D)

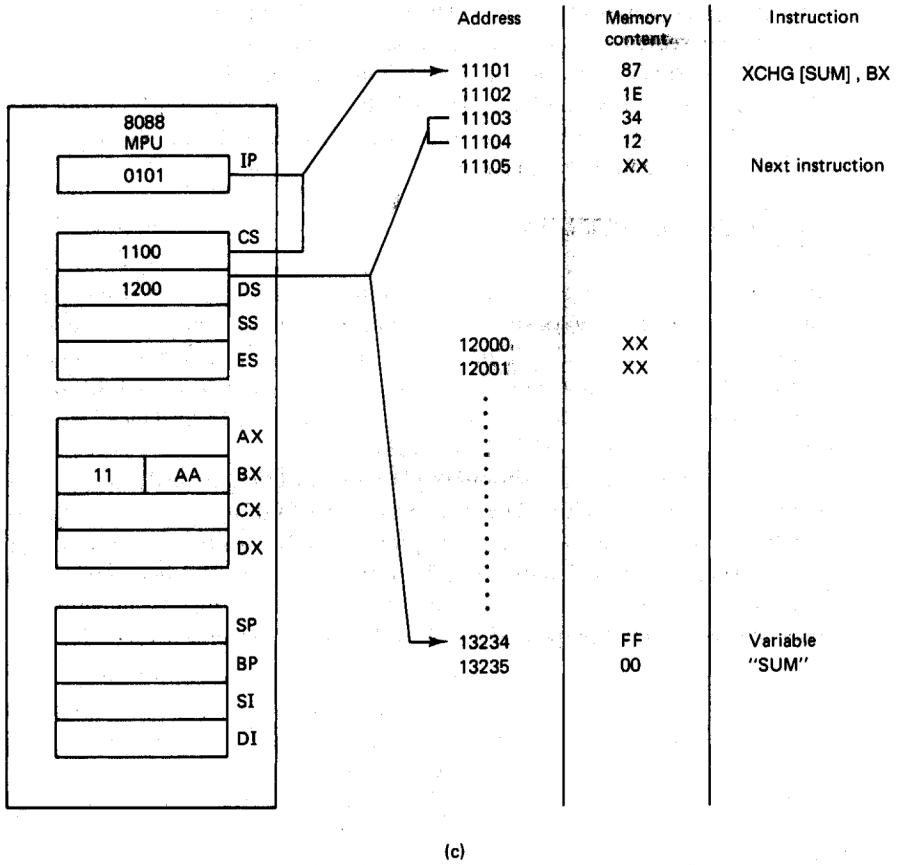
      (D) → (S)
    - Flags unaffected
  - Special accumulator destination version executes faster
  - Examples:

    XCHG AX,DX

    (Original value in AX) → (DX)

    (Original value in DX) → (AX)

5.1 Data Transfer Instructions- Exchange Instruction



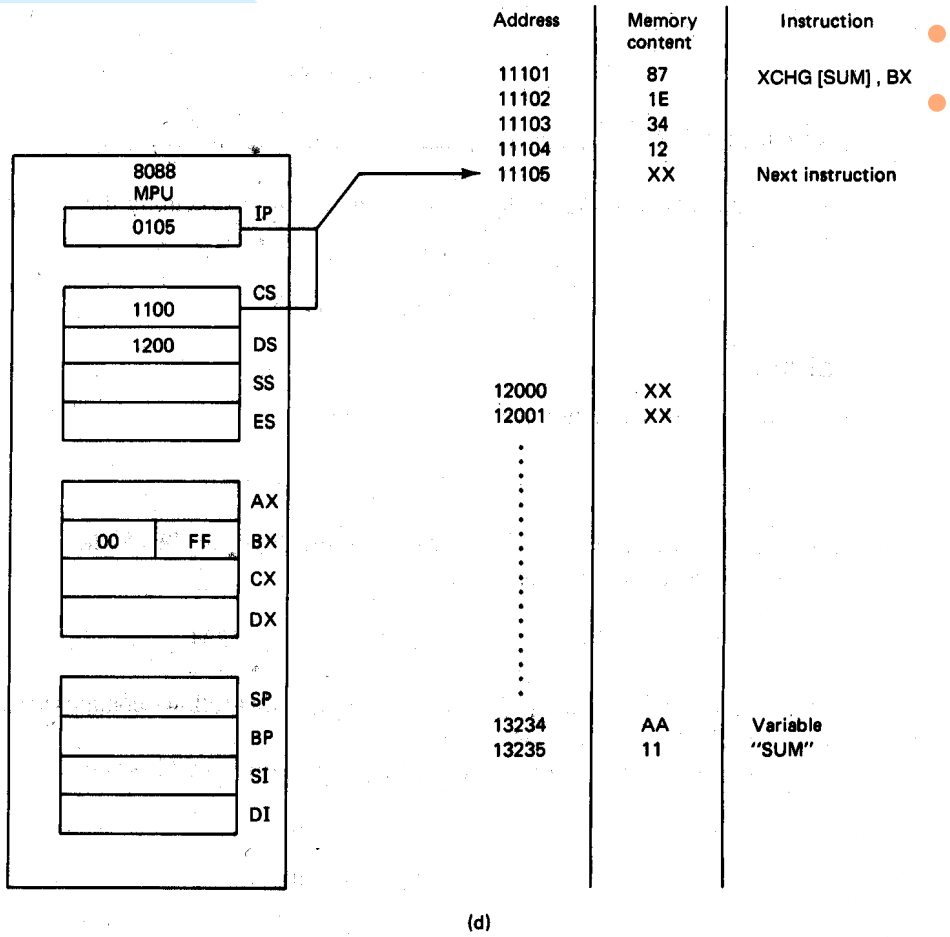
(c)

- Example
  XCHG [SUM],BX
  - Source = BX → word data
  - Destination = memory offset
    SUM → word data
  - Operation: (SUM) → (BX)
    (BX) → (SUM)

  What is the general logical address of the destination operand?
- State before fetch and execution
  - CS:IP = 1100:0101 = 11101H
  - Move instruction code = 871E3412H
  - (01104H,01103H) = 1234H = SUM
  - (DS) = 1200H
  - (BX) = 11AA
  - (DS:SUM) = (1200:1234) = 00FFH

  What is this type data organization called?

5.1 Data Transfer Instructions- Exchange Instruction



(d)

- Example (continued)
- State after execution

  CS:IP = 1100:0105 = 11105H

  11005H → points to next sequential instruction

  - Register updated

    (BX) = 00FFH
  - Memory updated

    (1200:1234) = AAH

    (1200:1235) = 11H

5.1 Data Transfer Instructions- Exchange Instruction

- Debug execution of example

```
C:\DOS>DEBUG
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 0F            DB      0F
-A 1100:101
1100:0101 XCHG [1234],BX
1100:0105
-R BX
BX 0000
:11AA
-R DS
DS 1342
:1200
-R CS
CS 1342
:1100
-R IP
IP 0100
:101
-R
AX=0000  BX=11AA  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1200  ES=1342  SS=1342  CS=1100  IP=0101   NV UP EI PL NZ NA PO NC
1100:0101 871E3412      XCHG    BX,[1234]                   DS:1234=0000
-E 1234 FF 00
-U 101 104
1100:0101 871E3412      XCHG    BX,[1234]
-T

AX=0000  BX=00FF  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1200  ES=1342  SS=1342  CS=1100  IP=0105   NV UP EI PL NZ NA PO NC
1100:0105 8946FE        MOV     [BP-02],AX                  SS:FFFE=0000
-D 1234 1235
1200:1230               AA 11                                       ..
-Q

C:\DOS>
```

Write a dump command?

5.1 Data Transfer Instructions- Translate Instruction

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	((AL)+(BX)+(DS)0) → (AL)	None

- Translate instruction
  - Used to look up a byte-wide value in a table in memory and copy that value in the AL register
  - General format:

    XLAT
  - Operation: Copies the content of the element pointed to in the source table in memory to the AL register

    ((AL)+(BX) +(DS)0) → (AL)

    Where:

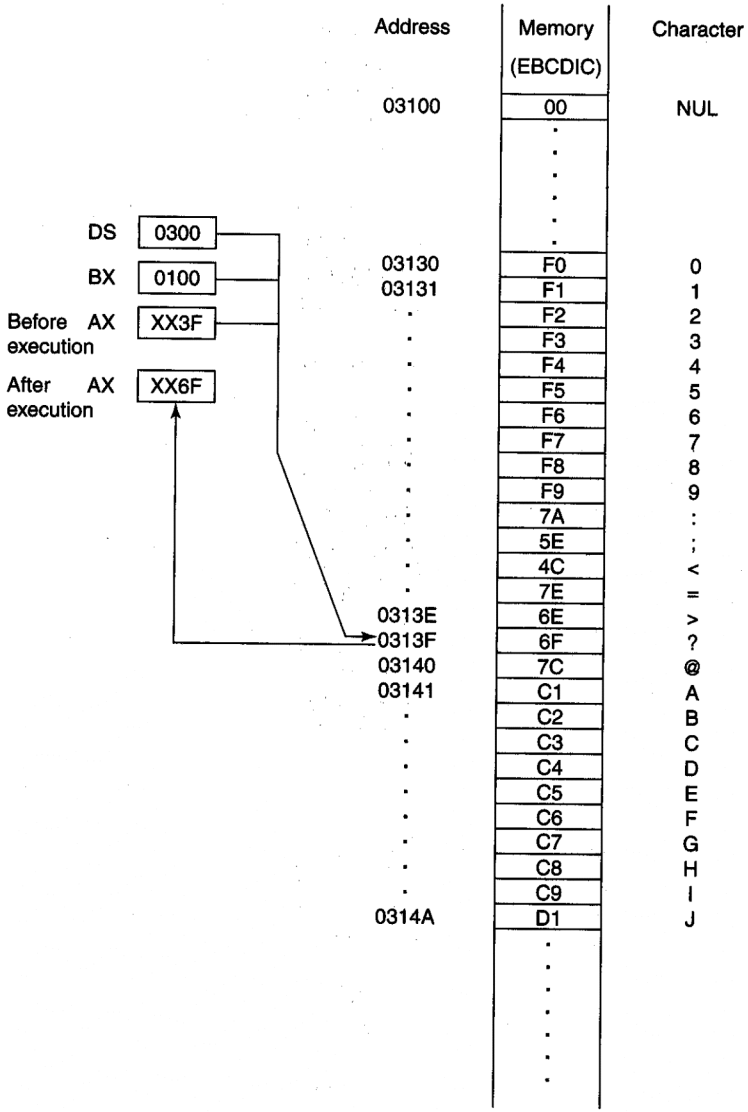
    (DS)0 = Points to the active data segment

    (BX) = Offset to the first element in the table

    (AL) = Displacement to the element of the table that is to be accessed*

    *8-bit value limits table size to 256 elements

5.1 Data Transfer Instructions- Translate Instruction



- Application: ASCII to EBCDIC Translation
  - Fixed EBCDIC table coded into memory starting at offset in BX
  - Individual EBCDIC codes placed in table at displacement (AL) equal to the value of their equivalent ASCII character
    - A = 41H in ASCII, A = C1H in EBCDIC
    - Place the value C1H in memory at address (A1H+(BX) +(DS)0), etc.
- Example

  XLAT
  (DS) = 0300H
  (BX) = 0100H
  (AL) = 3FH → 6FH = ? (Question mark)

5.1 Data Transfer Instructions- Load Effective Address and Load Full Pointer Instructions

Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load effective address	LEA Reg16,EA	EA → (Reg16)	None
LDS	Load register and DS	LDS Reg16,Mem32	(Mem32) → (Reg16) (Mem32+2) → (DS)	None
LES	Load register and ES	LES Reg16,Mem32	(Mem32) → (Reg16) (Mem32+2) → (ES)	None

(a)

- Load effective address instruction
  - Used to load an address pointer offset from memory into a register
  - General format:

    LEA Reg16,EA
  - Operation:

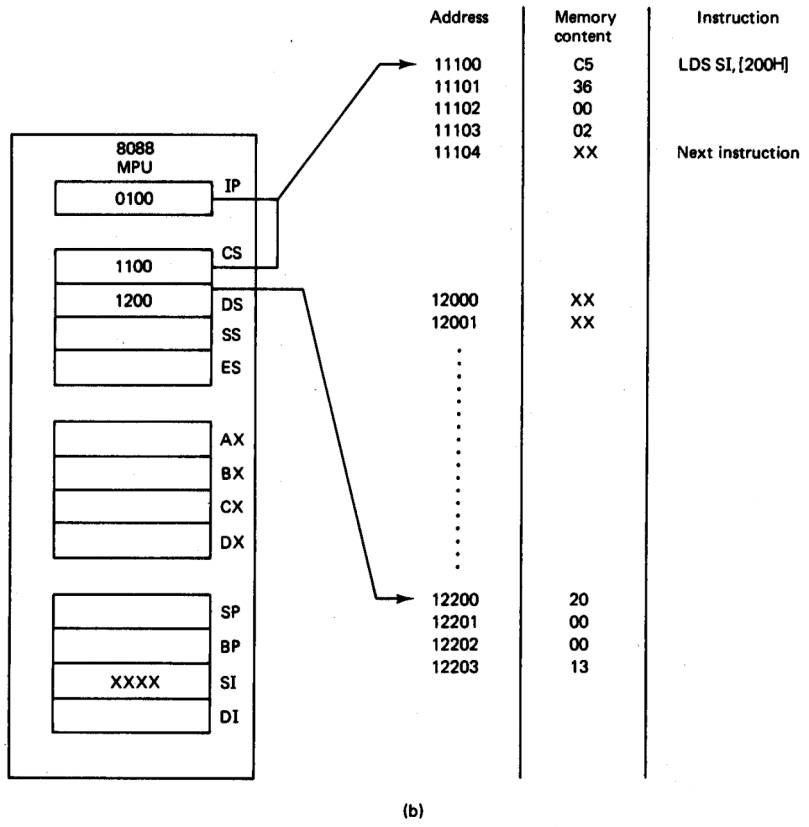
    (EA) → (Reg16)
    - Source unaffected:
    - Flags unaffected
- Load full pointer
  - Used to load a full address pointer from memory into a segment register and a register
    - Segment base address
    - Offset
  - General format and operation for LDS

    LDS Reg16,EA

    (EA) → (Reg16)

    (EA+2) →(DS)
  - LES operates the same, except initializes ES

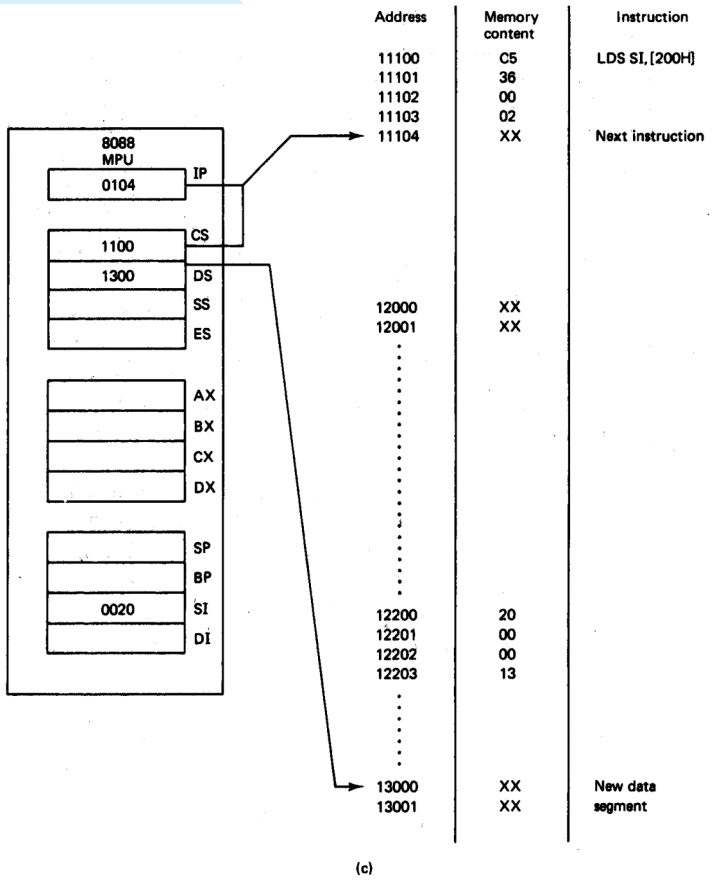
5.1 Data Transfer Instructions- Load Effective Address and Load Full Pointer Instructions



(b)

- Example
  - LDS SI,[200H]
    - Source = pointer to DS:200H→ 32 bits
    - Destination = SI → word pointer offset
      - DS → word pointer SBA
    - Operation: (DS:200H) → (SI)
      - (DS:202H) → (DS)
- State before fetch and execution
  - CS:IP = 1100:0100 = 11100H
  - LDS instruction code = C5360002H
  - (11102H,11103H) = (EA) = 0200H
  - (DS) = 1200H
  - (SI) = XXXX → don't care state
  - (DS:EA) = 12200H = 0020H = Offset
  - (DS:EA+2) = 12202H = 1300H = SBA

5.1 Data Transfer Instructions- Load Effective Address and Load Full Pointer Instructions



(c)

- Example (continued)
- State after execution

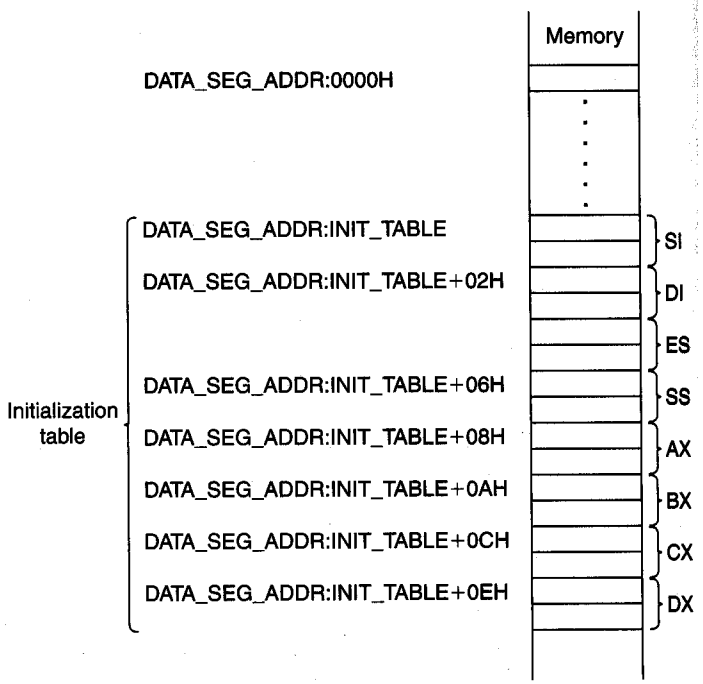
  CS:IP = 1100:0104 = 11104H

  01004H → points to next sequential instruction

  (DS) = 1300H → defines a new data segment

  (SI) = 0020H → defines new offset into DS

5.2 Data Transfer Instructions- Load Effective Address and Load Full Pointer Instructions




```
MOV  AX, DATA_SEG_ADDR
MOV  DS,AX
MOV  SI,[INIT_TABLE]
LES  DI,[INIT_TABLE+02H]
MOV  AX,[INIT_TABLE+06H]
MOV  SS,AX
MOV  AX,[INIT_TABLE+08H]
MOV  BX,[INIT_TABLE+0AH]
MOV  CX,[INIT_TABLE+0CH]
MOV  DX,[INIT_TABLE+0EH]
```

Example—Initialization of internal registers from memory with data and address information

- DS loaded via AX with immediate value using move instructions
  DATA_SEG_ADDR → (AX) → (DS)
- Index register SI loaded with move from table
  (INIT_TABLE,INIT_TABLE+1) → SI
- DI and ES are loaded with load full pointer instruction
  (INIT_TABLE+2,INIT_TABLE+3) → DI
  (INIT_TABLE+4,INIT_TABLE+5) → ES
- SS loaded from table via AX using move instructions
  (INIT_TABLE+6,INIT_TABLE+7) → AX → (SS)
- Data registers loaded from table with move instructions
  (INIT_TABLE+8,INIT_TABLE+9) → AX
  (INIT_TABLE+A,INIT_TABLE+B) → BX
  (INIT_TABLE+C,INIT_TABLE+D) → CX
  (INIT_TABLE+E,INIT_TABLE+F) → DX

5.2 Arithmetic Instructions- Addition Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
ADD	Addition	ADD D, S	(S) + (D) → (D) Carry → (CF)	OF, SF, ZF, AF, PF, CF
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) → (D) Carry → (CF)	OF, SF, ZF, AF, PF, CF
INC	Increment by 1	INC D	(D) + 1 → (D)	OF, SF, ZF, AF, PF
AAA	ASCII adjust for addition	AAA		AF, CF OF, SF, ZF, PF undefined
DAA	Decimal adjust for addition	DAA		SF, ZF, AF, PF, CF, OF, undefined

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Destination
Reg16
Reg8
Memory

(c)

- Variety of arithmetic instruction provided to support integer addition—core instructions are
  - ADD→ Addition
  - ADC → Add with carry
  - INC → Increment
- Addition Instruction—ADD
  - ADD format and operation:

    ADD D,S

    (S) +(D) → (D)
    - Add values in two registers

      ADD AX,BX

      (AX) + (BX) → (AX)
    - Add a value in memory and a value in a register

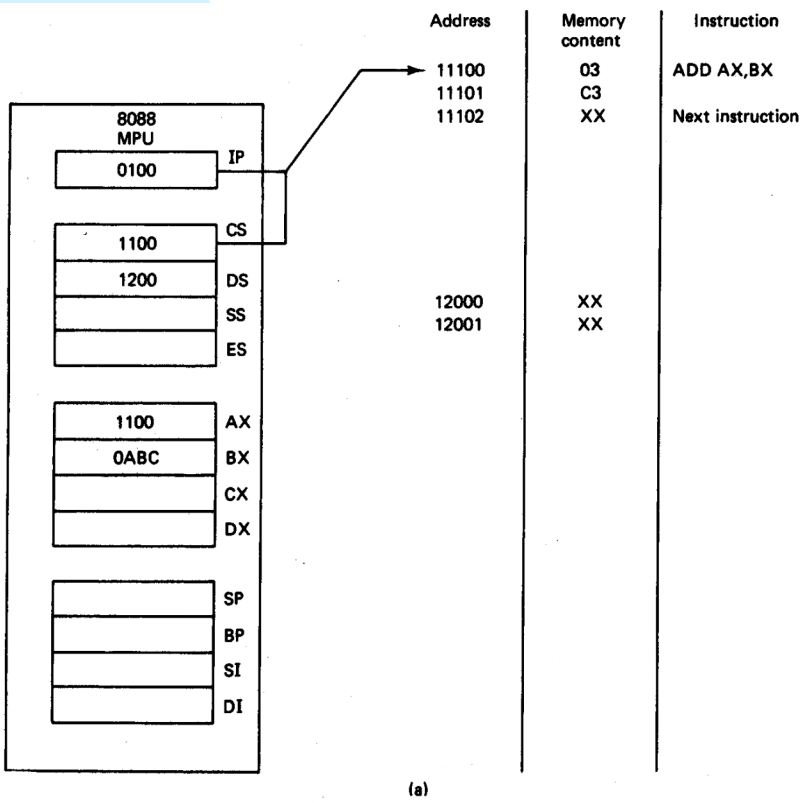
      ADD [DI],AX

      (DS:DI) + (AX) → (DS:DI)
    - Add an immediate operand to a value in a register or memory

      ADD AX,100H

      (AX) + IMM16 → (AX)
- Flags updated based on result
  - CF, OF, SF, ZF, AF, PF

5.2 Arithmetic Instructions- Addition Instructions



- Example

  ADD AX,BX

  (AX) + (BX) → (AX)

  - Word-wide register to register add
  - Half adder operation
- State before fetch and execution

  CS:IP = 1100:0100 = 11100H

  ADD machine code = 03C3H

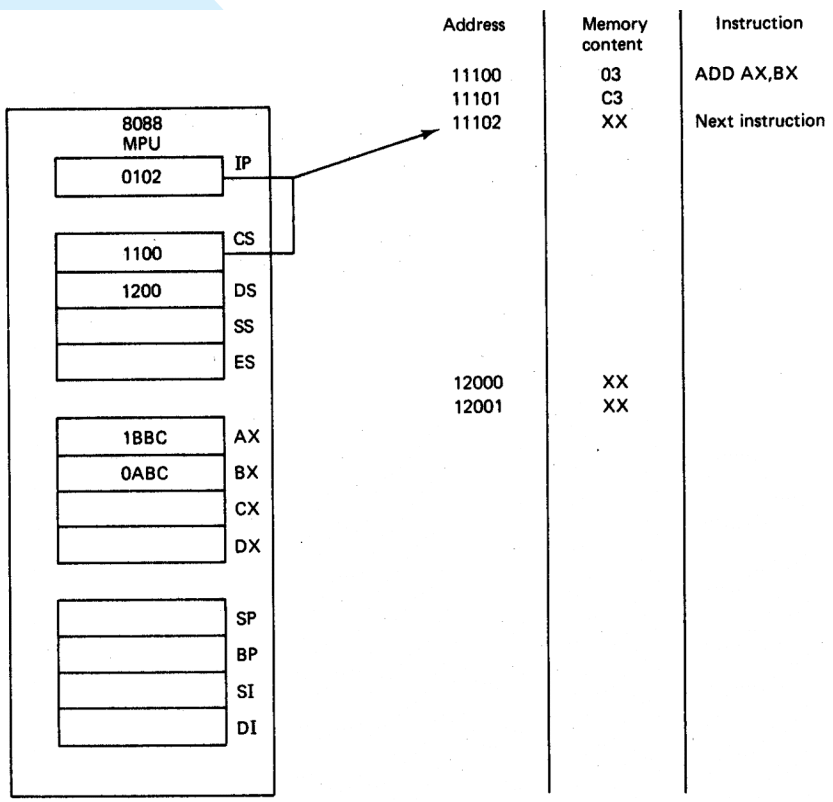
  (AX) = 1100H

  (BX) = 0ABCH

  (DS) = 1200H

  (1200:0000) = 12000H = XXXX

5.2 Arithmetic Instructions- Addition Instructions



- Example (continued)
- State after execution

  CS:IP = 1100:0102 = 11102H

  11102H → points to next sequential instruction
  - Operation performed

    (AX) + (BX) → (AX)

    (1100H) + (0ABCH) → 1BBCH

    (AX) = 1BBCH

    = 0001101110111100_2

    (BX) = unchanged
  - Impact on flags
    - CF = 0 (no carry resulted)
    - ZF = 0 (not zero)
    - SF = 0 (positive)
    - PF = 0 (odd parity)—parity flag is only based on the bits of the least significant byte

5.2 Arithmetic Instructions- Addition Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
ADD	Addition	ADD D, S	(S) + (D) → (D) Carry → (CF)	OF, SF, ZF, AF, PF, CF
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) → (D) Carry → (CF)	OF, SF, ZF, AF, PF, CF
INC	Increment by 1	INC D	(D) + 1 → (D)	OF, SF, ZF, AF, PF
AAA	ASCII adjust for addition	AAA		AF, CF OF, SF, ZF, PF undefined
DAA	Decimal adjust for addition	DAA		SF, ZF, AF, PF, CF, OF, undefined

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Destination
Reg16
Reg8
Memory

(c)

- Add with carry instruction—ADC
  - ADC format and operation:
    ADC D,S
    (S) +(D) + (CF) → (D)
    - Full-add operation
    - Used for extended addition
  - Add two registers with carry
    ADC AX,BX
    (AX) + (BX) + (CF) → (AX)
  - Add register and memory with carry
    ADC [DI],AX
    (DS:DI) + (AX)+ (CF) → (DS:DI)
  - Add immediate operand to a value in a register or memory
    ADC AX,100H
    (AX) + IMM16 + (CF) → (AX)
  - Same flags updated as ADD
- Increment instruction—INC
  - INC format and operation
    INC D
    (D) + 1 → (D)
  - Used to increment pointers (addresses)

5.2 Arithmetic Instructions- Addition Instructions

- Example—Arithmetic computations
  - Initial state:
    (AX) = 1234H
    (BL) = ABH
    (SUM) = 00CDH
    (CF) = 0
  - Operation of first instruction
    (DS:SUM) + (AX) → (AX)
    00CDH + 1234H = 1301H
    (AX) = 1301H
    (CF) = unchanged
  - Operation of second instruction
    (BL) + IMM8 +(CF) → BL
    ABH + 05H + 0 = B0H
    (BL) = B0H
    (CF) = unchanged
  - Operation of third instruction
    (DS:SUM) + 1 → (DS:SUM)
    00CDH + 1 = 00CEH
    (SUM) = 00CEH
    (CF) = unchanged

Instruction	(AX)	(BL)	(SUM)	(CF)
Initial state	1234	AB	00CD	0
ADD AX, [SUM]	1301	AB	00CD	0
ADC BL, 05H	1301	B0	00CD	0
INC WORD PTR [SUM]	1301	B0	00CE	0

1. Does the column (SUM) stand for a value in a register code memory, or a storage location in memory?

2. Why is the operand of the INC instruction preceded by WORD PTR?

5.2 Arithmetic Instructions- Addition Instructions

- Example—Execution of the arithmetic computation sequence

```
C:DOS>DEBUG A:EX511.EXE
-U 0 12
0D03:0000 1E            PUSH    DS
0D03:0001 B80000        MOV     AX,0000
0D03:0004 50            PUSH    AX
0D03:0005 B8050D        MOV     AX,0D05
0D03:0008 8ED8          MOV     DS,AX
0D03:000A 03060000      ADD     AX,[0000]
0D03:000E 80D305        ADC     BL,05
0D03:0011 FF060000      INC     WORD PTR [0000]
-G A

AX=0D03  BX=0000  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0D05  ES=0CF3  SS=0D06  CS=0D03  IP=000A   NV UP EI PL NZ NA PO NC
0D03:000A 03060000      ADD     AX,[0000]                   DS:0000=00CD
-R AX
AX 0D03
:1234
-R BX
BX 0000
:AB
-R F
NV UP EI PL NZ NA PO NC  -
-E 0 CD 00
-D 0 1
0D05:0000  CD 00
-T

AX=1301  BX=00AB  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0D05  ES=0CF3  SS=0D06  CS=0D03  IP=000E   NV UP EI PL NZ AC PO NC
0D03:000E 80D305        ADC     BL,05
-T

AX=1301  BX=00B0  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0D05  ES=0CF3  SS=0D06  CS=0D03  IP=0011   NV UP EI NG NZ AC PO NC
0D03:0011 FF060000      INC     WORD PTR [0000]             DS:0000=00CD
-T

AX=1301  BX=00B0  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0D05  ES=0CF3  SS=0D06  CS=0D03  IP=0015   NV UP EI PL NZ NA PO NC
0D03:0015 CB            RETF
-D 0 1
0D05:0000  CE 00
-G

Program terminated normally
-Q

C:\DOS>
```

Missing = sign

1. What is the value of SUM?

2. What is logical address is accessed in memory?

5.2 Arithmetic Instructions- Subtraction Instructions

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D) − (S) → (D) Borrow → (CF)	OF, SF, ZF, AF, PF, CF
SBB	Subtract with borrow	SBB D,S	(D) − (S) − (CF) → (D)	OF, SF, ZF, AF, PF, CF
DEC	Decrement by 1	DEC D	(D) − 1 → (D)	OF, SF, ZF, AF, PF
NEG	Negate	NEG D	0 − (D) → (D) 1 → (CF)	OF, SF, ZF, AF, PF, CF
DAS	Decimal adjust for subtraction	DAS		SF, ZF, AF, PF, CF OF undefined
AAS	ASCII adjust for subtraction	AAS		AF, CF OF, SF, ZF, PF undefined

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Accumulator	Immediate
Register	Immediate
Memory	Immediate

(b)

Destination
Reg16
Reg8
Memory

(c)

Destination
Register
Memory

(d)

- Variety of arithmetic instruction provided to support integer subtraction—core instructions are
  - SUB → Subtract
  - SBB → Subtract with borrow
  - DEC → Decrement
  - NEG → Negative
- Subtract Instruction—SUB
  - SUB format and operation:

    SUB D,S

    (D) - (S) → (D)
    - Subtract values in two registers

      SUB AX,BX

      (AX) - (BX) → (AX)
    - Subtract a value in memory and a value in a register

      SUB [DI],AX

      (DS:DI) - (AX) → (DS:DI)
    - Subtract an immediate operand from a value in a register or memory

      SUB AX,100H

      (AX) - IMM16 → (AX)
- Flags updated based on result
  - CF, OF, SF, ZF, AF, PF

5.2 Arithmetic Instructions- Subtraction Instructions

```
C:\DOS>DEBUG
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 0F            DB      0F
-R BX
BX 0000
:1234
-R CX
CX 0000
:0123
-R F
NV UP EI PL NZ NA PO NC  -
-A
1342:0100 SBB BX,CX
1342:0102
-R
AX=0000  BX=1234  CX=0123  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 19CB          SBB     BX,CX
-U 100 101
1342:0100 19CB          SBB     BX,CX
-T

AX=0000  BX=1111  CX=0123  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   NV UP EI PL NZ NA PE NC
1342:0102 B98AFF        MOV     CX,FF8A
-Q

C:\DOS>
```

- Subtract with borrow instruction—SBB
  - SBB format and operation:
    SBB D,S
    (D) - (S) - (CF) → (D)
    - Used for extended subtractions
  - Subtracts two registers and carry (borrow)
    SBB AX,BX
  - Example:
    SBB BX,CX
    (BX) = 1234H
    (CX) = 0123H
    (CF) = 0
    (BX) - (CX) - (CF) → (BX)
    1234H - 0123H - 0H = 1111H
    (BX) = 1111H
    - What about CF?

5.2 Arithmetic Instructions- Subtraction Instructions

```
C:\DOS>DEBUG
-R BX
BX 0000
:3A
-A
1342:0100 NEG BX
1342:0102
-R BX
BX 003A
:
-U 100 101
1342:0100 F7DB          NEG     BX
-T

AX=0000  BX=FFC6  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   NV UP EI NG NZ AC PE CY
1342:0102 B98AFF        MOV     CX,FF8A
-Q

C:\DOS>
```

- Negate instruction—NEG
  - NEG format and operation
    - NEG D
    - (0) - (D) → (D)
    - (1)→ (CF)
  - Example:
    - NEG BX
    - (BX) =003AH
    - (0) - (BX) → (BX)
    - 0000H – 003AH=
    - 0000H + FFC6H (2's complement) = FFC6H
    - (BX) =FFC6H; CF =1
- Decrement instruction—DEC
  - DEC format and operation
    - DEC D
    - (D) - 1 → (D)
  - Used to decrement pointer—addresses
  - Example
    - DEC SI
    - (SI) = 0FFFH
    - (SI) - 1 → SI
    - 0FFFH - 1 = 0FFEH
    - (DI) = 0FFEH

5.2 Arithmetic Instructions- Multiplication Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
MUL	Multiply (unsigned)	MUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interupt occurs	OF, SF, ZF, AF, PF, CF undefined
AAM	Adjust AL for multiplication	AAM	Q((AL)/10) → (AH) R((AL)/10) → (AL)	SF, ZF, PF OF, AF,CF undefined
AAD	Adjust AX for division	AAD	(AH) · 10 + (AL) → (AL) 00 → (AH)	SF, ZF, PF OF, AF, CF undefined
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None

(a)

Source
Reg8
Reg16
Mem8
Mem16

(b)

- Integer multiply instructions—MUL and IMUL
  - Multiply two unsigned or signed byte or word operands
  - General format and operation
    - MUL S = Unsigned integer multiply
    - IMUL S = Signed integer multiply
      - (AL) X (S8)→ (AX) 8-bit product gives 16 bit result
      - (AX) X (S16) → (DX), (AX) 16-bit product gives 32 bit result
    - Source operand (S) can be an 8-bit or 16-bit value in a register or memory
    - AX assumed to be destination for 16 bit result
    - DX,AX assumed destination for 32 bit result
    - Only CF and OF flags updated; other undefined

5.2 Arithmetic Instructions- Multiplication Instructions

```
C:\DOS>DEBUG
-R AX
AX 0000
:FF
-R CX
CX 0000
:FE
-A
1342:0100 MUL CL
1342:0102
-R AX
AX 00FF
:
-R CX
CX 00FE
:
-U 100 101
1342:0100 F6E1          MUL     CL
-T

AX=FD02  BX=0000  CX=00FE  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   OV UP EI NG NZ AC PE CY
1342:0102 B98AFF        MOV     CX,FF8A
-Q

C:\DOS>
```

- **Example:**

  MUL CL

  (AL) = -1_{10}

  (CL) = -2_{10}

  Expressing in 2's complement

  (AL) = -1 = 11111111_2 = FFH

  (CL) = -2 = 11111110_2 = FEH

  Operation:

  (AL) X (CL) → (AX)

  11111111_2 X 11111110_2 = 1111110100000010

  (AX) = FD02H

5.2 Arithmetic Instructions- Division Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
MUL	Multiply (unsigned)	MUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interupt occurs	OF, SF, ZF, AF, PF, CF undefined
AAM	Adjust AL for multiplication	AAM	Q((AL)/10) → (AH) R((AL)/10) → (AL)	SF, ZF, PF OF, AF,CF undefined
AAD	Adjust AX for division	AAD	(AH) · 10 + (AL) → (AL) 00 → (AH)	SF, ZF, PF OF, AF, CF undefined
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None

(a)

Source
Reg8
Reg16
Mem8
Mem16

(b)

- Integer divide instructions—DIV and IDIV
  - Divide unsigned– DIV S
    - Operations:

      (AX) / (S8) → (AL) =quotient

      (AH) = remainder
      - 16 bit dividend in AX divided by 8-bit divisor in a register or memory,
      - Quotient of result produced in AL
      - Remainder of result produced in AH

      (DX,AX) / (S16) → (AX) =quotient

      (DX) = remainder
      - 32 bit dividend in DX,AX divided by 16-bit divisor in a register or memory
      - Quotient of result produced in AX
      - Remainder of result produced in DX
    - Divide error (Type 0) interrupt may occur

5.2 Arithmetic Instructions- Convert Instructions

```
C:\DOS>DEBUG A:EX520.EXE
-U 0 9
0D03:0000 1E            PUSH    DS
0D03:0001 B80000        MOV     AX,0000
0D03:0004 50            PUSH    AX
0D03:0005 B0A1          MOV     AL,A1
0D03:0007 98            CBW
0D03:0008 99            CWD
0D03:0009 CB            RETF
-G 5

AX=0000  BX=0000  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000 DI=0000
DS=0CF3  ES=0CF3  SS=0D04  CS=0D03  IP=0005   NV UP EI PL NZ NA PO NC
0D03:0005 B0A1          MOV     AL,A1
-T

AX=00A1  BX=0000  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000 DI=0000
DS=0CF3  ES=0CF3  SS=0D04  CS=0D03  IP=0007   NV UP EI PL NZ NA PO NC
0D03:0007 98            CBW
-T

AX=FFA1  BX=0000  CX=0000  DX=0000  SP=003C  BP=0000  SI=0000 DI=0000
DS=0CF3  ES=0CF3  SS=0D04  CS=0D03  IP=0008   NV UP EI PL NZ NA PO NC
0D03:0008 99            CWD
-T

AX=FFA1  BX=0000  CX=0000  DX=FFFF  SP=003C  BP=0000  SI=0000 DI=0000
DS=0CF3  ES=0CF3  SS=0D04  CS=0D03  IP=0009   NV UP EI PL NZ NA PO NC
0D03:0009 CB            RETF
-G

Program terminated normally
-Q

C:\DOS>
```

(c)

- Convert instructions
  - Used to sign extension signed numbers for division
  - Operations
    - CBW = convert byte to word
      (MSB of AL) → (all bits of AH)
    - CWD = convert word to double word
      (MSB of AX) → (all bits of DX)
  - Application:
    - To divide two signed 8-bit numbers, the value of the dividend must be sign extended in AX
      - Load into AL
      - Use CBW to sign extend to 16 bits
  - Example

    A1H → AL

    CBW sign extends to give

    FFA1H → AX

    CWD sign extends to give

    FFFFH → DX

5.3 Logic Instructions- Available Instructions and their Operation

- Variety of logic instructions provided to support logical computations
  - AND → Logical AND
  - OR → Logical inclusive-OR
  - XOR → Logical exclusive-OR
  - NOT → Logical NOT
- Logical AND Instruction—AND
  - AND format and operation:
    AND D,S
    (S) AND (D) → (D)
    - Logical AND of values in two registers
      AND AX,BX
      (AX) AND (BX) → (AX)
    - Logical AND of a value in memory and a value in a register
      AND [DI],AX
      (DS:DI) AND (AX) → (DS:DI)
    - Logical AND of an immediate operand with a value in a register or memory
      AND AX,100H
      (AX) AND IMM16 → (AX)
  - Flags updated based on result
    - CF, OF, SF, ZF, PF
    - AF undefined

Mnemonic	Meaning	Format	Operation	Flags Affected
AND	Logical AND	AND D,S	(S) · (D) → (D)	OF, SF, ZF, PF, CF AF undefined
OR	Logical Inclusive-OR	OR D,S	(S) + (D) → (D)	OF, SF, ZF, PF, CF AF undefined
XOR	Logical Exclusive-OR	XOR D,S	(S) ⊕ (D) → (D)	OF, SF, ZF, PF, CF AF undefined
NOT	Logical NOT	NOT D	$(\overline{D}) \rightarrow (D)$	None

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Destination
Register
Memory

(c)

1. Describe the AND operations.
2. Describe the OR operations.
3. Describe the XOR operations.
4. Describe the NOT operations.

5.3 Logic Instructions- Example

Instruction	(AL)
MOV AL,01010101B	01010101
AND AL,00011111B	00010101
OR AL,11000000B	11010101
XOR AL,00001111B	11011010
NOT AL	00100101

```
C:\DOS>DEBUG
-A
1342:0100 MOV AL,55
1342:0102 AND AL,1F
1342:0104 OR AL,C0
1342:0106 XOR AL,0F
1342:0108 NOT AL
1342:010A
-T

AX=0055  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   NV UP EI PL NZ NA PO NC
1342:0102 241F          AND     AL,1F
-T

AX=0015  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0104   NV UP EI PL NZ NA PO NC
1342:0104 0CC0          OR      AL,C0
-T

AX=00D5  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0106   NV UP EI NG NZ NA PO NC
1342:0106 340F          XOR     AL,0F
-T

AX=00DA  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0108   NV UP EI NG NZ NA PO NC
1342:0108 F6D0          NOT     AL
-T

AX=0025  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=010A   NV UP EI NG NZ NA PO NC
1342:010A 2B04          SUB     AX,[SI]                     DS:0000=20CD
-Q

C:\DOS>
```

5.3 Logic Instructions- Mask Application

- Application– Masking bits with the logic instructions
  - Mask—to clear a bit or bits of a byte or word to 0
    - AND operation can be used to perform the mask operation
    - 1 AND 0 → 0; 0 and 0 → 0
      - A bit or bits are masked by ANDing with 0
    - 1 AND 1 → 1; 0 AND 1 → 0
      - ANDing a bit or bits with 1 results in no change
    - Example: Masking the upper 12 bits of a value in a register

      AND AX,000FH

      (AX) =FFFF

      IMM16 AND (AX) → (AX)

000FH AND FFFFH = 0000000000001111_2 AND 1111111111111111_2

= 0000000000001111_2

= 000FH

1. Write an AND instruction to clear the 5th bit in the AL.

(AL) = b7b6b5b4b3b2b1b0?

5.3 Logic Instructions- Mask Application

- OR operation can be used to set a bit or bits of a byte or word to 1
  - X OR 0 → X; result is unchanged
  - X or 1 → 1; result is always 1
  - Example: Setting a control flag in a byte memory location to 1

```
MOV AL,[CONTROL_FLAGS]
OR   AL, 10H            ; 00010000 sets fifth bit –b4
MOV [CONTROL_FLAGS],AL
```

General Operation:

$(AL) = XXXXXXXX_2$ OR $00010000_2 = XXX1XXXX_2$

1. What is CONTROL_FLAGS?

2, What is it relative to?

5.4 Shift Instructions- Available Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
SAL/SHL	Shift arithmetic left/shift logical left	SAL/SHL D,Count	Shift the (D) left by the number of bit positions equal to Count and fill the vacated bits positions on the right with zeros	CF, PF, SF, ZF AF undefined OF undefined if count ≠ 1
SHR	Shift logical right	SHR D,Count	Shift the (D) right by the number of bit positions equal to Count and fill the vacated bit positions on the left with zeros	CF, PF, SF, ZF AF undefined OF undefined if count ≠ 1
SAR	Shift arithmetic right	SAR D,Count	Shift the (D) right by the number of bit positions equal to Count and fill the vacated bit positions on the left with the original most significant bit	SF, ZF, PF, CF AF undefined OF undefined if count ≠1

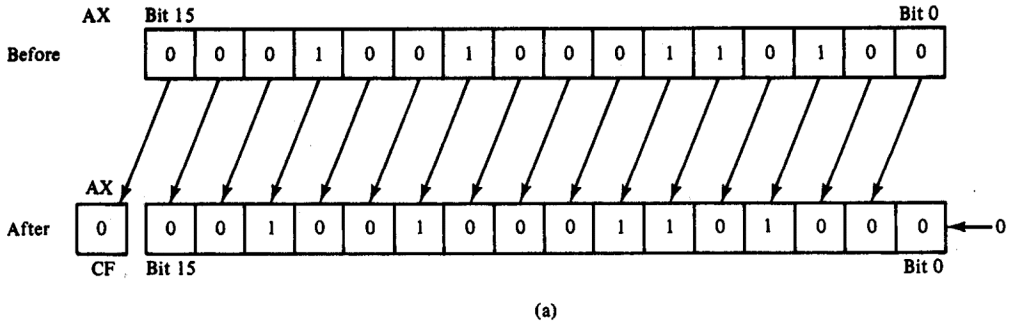
(a)

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

(b)

- Variety of shift instructions provided
  - SAL/SHL → Shift arithmetic left/shift logical left
  - SHR → Shift logical right
  - SAR → Shift arithmetic right
- Perform a variety of shift left and shift right operations on the bits of a destination data operand
- Basic shift instructions—SAL/SHL, SHR, SAR
  - Destination may be in either a register or a storage location in memory
  - Shift count may be:
    - 1= one bit shift
    - CL = 1 to 255 bit shift
  - Flags updated based on result
    - CF, SF, ZF, PF
    - AF undefined
    - OF undefined if Count ≠ 1

5.4 Shift Instructions- Operation of the SAL/SHL Instruction

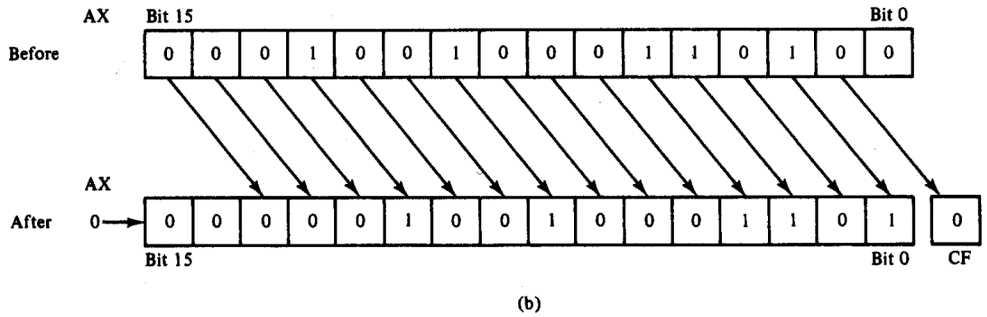


(a)

- SAL/SHL instruction operation
  - Typical instruction—count of 1
    SHL AX,1
  - Before execution
    Dest = (AX) = 1234H
    = 0001001000110100_2
    Count = 1
    CF = X
  - Operation
    - The value in all bits of AX are shifted left one bit position
    - Emptied LSB is filled with 0
    - Value shifted out of MSB goes to carry flag
  - After execution
    Dest = (AX) = 2468H
    = 0010010001101000_2
    CF = 0
  - Conclusion
    - MSB has been isolated in CF and can be acted upon by control flow instruction– conditional jump
    - Result has been multiplied by 2

Note: Signed or unsigned data answer is the same

5.4 Shift Instructions- Operation of the SHR Instruction

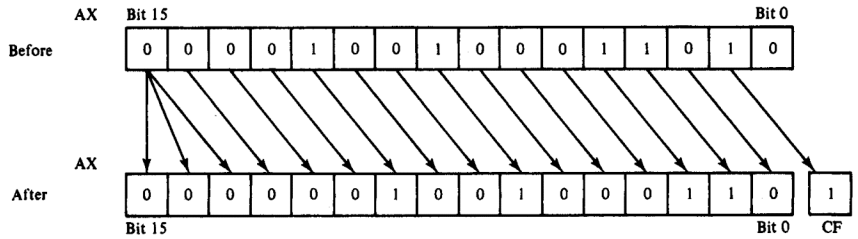


(b)

- SHR instruction operation
  - Typical instruction—count in CL
    SHR AX,CL
  - Before execution
    Dest = (AX) = 1234H = 4660_{10}
    = 0001001000110100_2
    Count = (CL) = 02H
    CF = X
  - Operation
    - The value in all bits of AX are shifted right two bit positions
    - Emptied MSBs are filled with 0s
    - Value shifted out of LSB goes to carry flag
  - After execution
    Dest = (AX) = 048DH = 1165_{10}
    = 0000010010001101_2
    CF = 0
  - Conclusion
    - Bit 1 has been isolated in CF and can be acted upon by control flow instruction– conditional jump
    - Result has been divided by 4
      - 4 X 1165 = 4660

Note: processes unsigned data

5.4 Shift Instructions- Operation of the SAR Instruction



- SAR instruction operation
  - Typical instruction—count in CL
    SAR AX,CL
  - Before execution—arithmetic implies signed numbers
    Dest = (AX) = 091AH = 0000100100011010_2 = +2330
    Count = CL = 02H
    CF = X
  - Operation
    - The value in all bits of AX are shifted right two bit positions
    - Emptied MSB is filled with the value of the sign bit
    - Values shifted out of LSB go to carry flag
  - After execution
    Dest = (AX) = 0246H = 0000001001000110_2 = +582
    CF = 1
  - Conclusion
    - Bit 1 has been isolated in CF and can be acted upon by control flow instruction– conditional jump
    - Result has been signed extended
    - Result value has been divided by 4 and rounded to integer
      - 4 X +582 = +2328

Note: processed data treated as signed number

5.4 Shift Instructions- SAR Instruction Execution

- Debug execution of example

```
C:\DOS>DEBUG
-A
1342:0100 SAR AX,CL
1342:0102
-R AX
AX 0000
:091A
-R CX
CX 0000
:2
-R F
NV UP EI PL NZ NA PO NC  -
-T

AX=0246  BX=0000  CX=0002  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   NV UP EI PL NZ AC PO CY
1342:0102 B98AFF        MOV     CX,FF8A
-Q

C:\DOS>
```

5.4 Shift Instructions- Application

- Application–Isolating a bit of a byte of data in memory in the carry flag
  - Example:
    - Instruction sequence

      MOV AL,[CONTROL_FLAGS]
      MOV CL, 04H
      SHR AL,CL
    - Before execution

      (CONTROL_FLAGS) = B7B6B5B4B3B2B1B0
    - After execution

      (AL) = 0000B7B6B5B4
      (CF) = B3

5.5 Rotate Instructions- Available Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
ROL	Rotate left	ROL D,Count	Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the leftmost bit goes back into the rightmost bit position.	CF OF undefined if count ≠ 1
ROR	Rotate right	ROR D,Count	Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes into the leftmost bit position.	CF OF undefined if count ≠ 1
RCL	Rotate left through carry	RCL D,Count	Same as ROL except carry is attached to (D) for rotation.	CF OF undefined if count ≠ 1
RCR	Rotate right through carry	RCR D,Count	Same as ROR except carry is attached to (D) for rotation.	CF OF undefined if count ≠ 1

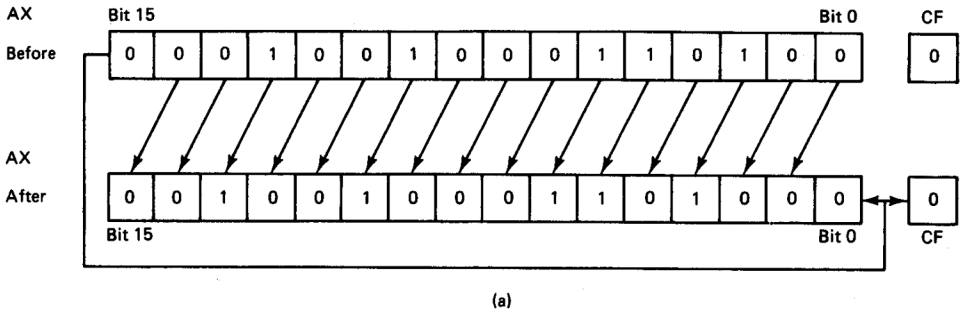
(a)

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

(b)

- Variety of rotate instructions provided
  - ROL → Rotate left
  - ROR → Rotate right
  - RCL → Rotate left through carry
  - RCR → Rotate right through carry
- Perform a variety of rotate left and rotate right operations on the bits of a destination data operand
- Overview of function
  - Destination may be in either a register or a storage location in memory
  - Rotate count may be:
    - 1= one bit rotate
    - CL = 1 to 255 bit rotate
  - Flags updated based on result
    - CF
    - OF undefined if Count ≠ 1
  - Used to rearrange information

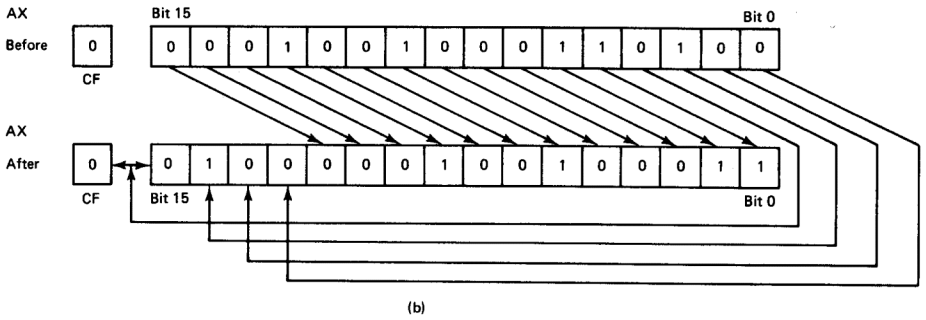
5.5 Rotate Instructions- Operation of the ROL Instruction



(a)

- ROL instruction operation
  - Typical instruction—count of 1
    ROL AX,1
  - Before execution
    Dest = (AX) = 1234H
    = $0001\ 0010\ 0011\ 0100_2$
    Count = 1
    CF = 0
  - Operation
    - The value in all bits of AX are rotated left one bit position
    - Value rotated out of the MSB is reloaded at LSB
    - Value rotated out of MSB is copied to carry flag
  - After execution
    Dest = (AX) = 2468H
    = $0010\ 0100\ 0110\ 1000_2$
    CF = 0

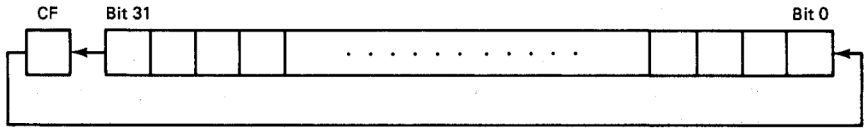
5.5 Rotate Instructions- Operation of the ROR Instruction



(b)

- ROR instruction operation
  - Typical instruction—count in CL
    ROR AX,CL
  - Before execution
    Dest = (AX) = 1234H = 0001001000110100_2
    Count = 04H
    CF = 0
  - Operation
    - The value in all bits of AX are rotated right four bit positions
    - Values rotated out of the LSB are reloaded at MSB
    - Values rotated out of MSB copied to carry flag
  - After execution
    Dest = (AX) = 4123H = 0100000100100011_2
    CF = 0
  - Conclusion:
    - Note that the position of hex characters in AX have be rearranged

5.5 Rotate Instructions- Operation of the RCL Instruction



- RCL instruction operation
  - Typical instruction—count in CL

    RCL BX,CL
  - Before execution

    Dest = (BX) = 1234H = 0001001000110100_2

    Count = (CL) = 04H

    CF = 0
  - Operation
    - The value in all bits of AX are rotated left four bit positions
    - Emptied MSBs are rotated through the carry bit back into the LSB
    - First rotate loads prior value of CF at the LSB
    - Last value rotated out of MSB retained in carry flag
  - After execution

    Dest = (BX) = 2340H = 0010001101000000_2

    CF = 1

5.5 Rotate Instructions- RCR Example

```
C:\DOS>DEBUG
-A
1342:0100 RCR BX,CL
1342:0102
-R BX
BX 0000
:1234
-R CX
CX 0000
:4
-R F
NV UP EI PL NZ NA PO NC  -
-T

AX=0000  BX=8123  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   OV UP EI PL NZ NA PO NC
1342:0102 B98AFF        MOV     CX,FF8A
-Q

C:\DOS>
```

- RCR instruction debug execution example
  - Instruction—count in CL

    RCR BX,CL
  - Before execution

    Dest = (BX) = 1234H = 0001001000110100_2

    Count = 04H

    CF = 0
  - After execution

    Dest = (BX) = 8123H = 1000000100100011_2

    CF = 0

5.5 Rotate Instructions- Application

```
MOV  AL,[HEX_DIGITS]
MOV  BL,AL
MOV  CL,04H
ROR  BL,CL
AND  AL,0FH
AND  BL,0FH
ADD  AL,BL
```

- Disassembling and adding 2 hex digits
  - 1st Instruction → Loads AL with byte containing two hex digits
  - 2nd Instruction →Copies byte to BL
  - 3rd Instruction → Loads rotate count
  - 4th instruction → Aligns upper hex digit of BL with lower digit in AL
  - 5th Instruction → Masks off upper hex digit in AL
  - 6th Instruction → Masks off upper four bits of BL
  - 7th Instruction → Adds two hex digits